**Exercise 8: Online Bookstore - Implementing CRUD Operations**

**Business Scenario:**

Implement Create, Read, Update, and Delete operations for the Book and Customer entities.

**Instructions:**

1. **CRUD Endpoints:**
   * Implement endpoints for creating, reading, updating, and deleting books and customers.
2. **Validating Input Data:**
   * Use validation annotations like **@NotNull, @Size**, and **@Min** to validate input data.
3. **Optimistic Locking:**
   * Implement optimistic locking for concurrent updates using JPA versioning.

**BookstoreApplication.java:**

package com.example.bookstore;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApplication.class, args);

}

}

**Book.java:**

package com.example.bookstore.entity;

import jakarta.persistence.\*;

import jakarta.validation.constraints.Min;

import jakarta.validation.constraints.NotNull;

import jakarta.validation.constraints.Size;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull(message = "Title cannot be null")

@Size(min = 2, max = 100, message = "Title must be between 2 and 100 characters")

private String title;

@NotNull(message = "Author cannot be null")

@Size(min = 2, max = 100, message = "Author must be between 2 and 100 characters")

private String author;

@Min(value = 1, message = "Price must be greater than 0")

private double price;

@Version

private int version;

}

**Customer.java:**

package com.example.bookstore.entity;

import jakarta.persistence.\*;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotNull;

import jakarta.validation.constraints.Size;

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull(message = "Name cannot be null")

@Size(min = 2, max = 100, message = "Name must be between 2 and 100 characters")

private String name;

@NotNull(message = "Email cannot be null")

@Email(message = "Email should be valid")

private String email;

@Version

private int version;

}

**BookRepository.java:**

package com.example.bookstore.repository;

import com.example.bookstore.entity.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

**CustomerRepository.java:**

package com.example.bookstore.repository;

import com.example.bookstore.entity.Customer;

import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

**BookService.java:**

package com.example.bookstore.service;

import com.example.bookstore.entity.Book;

import com.example.bookstore.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public Book createBook(Book book) {

return bookRepository.save(book);

}

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Optional<Book> getBookById(Long id) {

return bookRepository.findById(id);

}

public Book updateBook(Book book) {

return bookRepository.save(book);

}

public void deleteBook(Long id) {

bookRepository.deleteById(id);

}

}

**CustomerService.java:**

package com.example.bookstore.service;

import com.example.bookstore.entity.Customer;

import com.example.bookstore.repository.CustomerRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class CustomerService {

@Autowired

private CustomerRepository customerRepository;

public Customer createCustomer(Customer customer) {

return customerRepository.save(customer);

}

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

public Optional<Customer> getCustomerById(Long id) {

return customerRepository.findById(id);

}

public Customer updateCustomer(Customer customer) {

return customerRepository.save(customer);

}

public void deleteCustomer(Long id) {

customerRepository.deleteById(id);

}

}

**BookController.java:**

package com.example.bookstore.controller;

import com.example.bookstore.entity.Book;

import com.example.bookstore.service.BookService;

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

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import java.util.List;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@PostMapping

public ResponseEntity<Book> createBook(@Validated @RequestBody Book book) {

return ResponseEntity.ok(bookService.createBook(book));

}

@GetMapping

public ResponseEntity<List<Book>> getAllBooks() {

return ResponseEntity.ok(bookService.getAllBooks());

}

@GetMapping("/{id}")

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

return bookService.getBookById(id)

.map(ResponseEntity::ok)

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

}

@PutMapping("/{id}")

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

if (!bookService.getBookById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

book.setId(id);

return ResponseEntity.ok(bookService.updateBook(book));

}

@DeleteMapping("/{id}")

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

if (!bookService.getBookById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

**CustomerController.java:**

package com.example.bookstore.controller;

import com.example.bookstore.entity.Customer;

import com.example.bookstore.service.CustomerService;

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

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import java.util.List;

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@PostMapping

public ResponseEntity<Customer> createCustomer(@Validated @RequestBody Customer customer) {

return ResponseEntity.ok(customerService.createCustomer(customer));

}

@GetMapping

public ResponseEntity<List<Customer>> getAllCustomers() {

return ResponseEntity.ok(customerService.getAllCustomers());

}

@GetMapping("/{id}")

public ResponseEntity<Customer> getCustomerById(@PathVariable Long id) {

return customerService.getCustomerById(id)

.map(ResponseEntity::ok)

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

}

@PutMapping("/{id}")

public ResponseEntity<Customer> updateCustomer(@PathVariable Long id, @Validated @RequestBody Customer customer) {

if (!customerService.getCustomerById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

customer.setId(id);

return ResponseEntity.ok(customerService.updateCustomer(customer));

}

@DeleteMapping("/{id}")

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

if (!customerService.getCustomerById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

**GlobalExceptionHandler.java:**

package com.example.bookstore.exception;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.context.request.WebRequest;

import org.springframework.dao.OptimisticLockingFailureException;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(OptimisticLockingFailureException.class)

public ResponseEntity<String> handleOptimisticLockingFailure(OptimisticLockingFailureException ex, WebRequest request) {

return new ResponseEntity<>("Conflict occurred: " + ex.getMessage(), HttpStatus.CONFLICT);

}

}

application.properties:

spring.jpa.properties.hibernate.jdbc.time\_zone=UTC

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

pom.xml:

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>bookstore</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.1.0</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Exercise 9: Online Bookstore - Understanding HATEOAS**

**Business Scenario:**

Enhance your REST API to follow HATEOAS principles for navigation through resources.

**Instructions:**

1. **Add Links to Resources:**
   * Use **Spring HATEOAS** to add links to resources in your API responses.
2. **Hypermedia-Driven APIs:**
   * Build and consume hypermedia-driven APIs.

**BookstoreApplication.java:**

package com.example.bookstore;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApplication.class, args);

}

}

Book.java:

package com.example.bookstore.entity;

import jakarta.persistence.\*;

import jakarta.validation.constraints.NotNull;

import jakarta.validation.constraints.Size;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 2, max = 100)

private String title;

@NotNull

@Size(min = 2, max = 100)

private String author;

@NotNull

private double price;

@Version

private int version;

}

**Customer.java:**

package com.example.bookstore.entity;

import jakarta.persistence.\*;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotNull;

import jakarta.validation.constraints.Size;

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 2, max = 100)

private String name;

@NotNull

@Email

private String email;

@Version

private int version;

}

**BookRepository.java:**

package com.example.bookstore.repository;

import com.example.bookstore.entity.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

CustomerRepository.java:

package com.example.bookstore.repository;

import com.example.bookstore.entity.Customer;

import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

**BookService.java:**

package com.example.bookstore.service;

import com.example.bookstore.entity.Book;

import com.example.bookstore.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public Book createBook(Book book) {

return bookRepository.save(book);

}

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Optional<Book> getBookById(Long id) {

return bookRepository.findById(id);

}

public Book updateBook(Book book) {

return bookRepository.save(book);

}

public void deleteBook(Long id) {

bookRepository.deleteById(id);

}

}

**CustomerService.java:**

package com.example.bookstore.service;

import com.example.bookstore.entity.Customer;

import com.example.bookstore.repository.CustomerRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class CustomerService {

@Autowired

private CustomerRepository customerRepository;

public Customer createCustomer(Customer customer) {

return customerRepository.save(customer);

}

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

public Optional<Customer> getCustomerById(Long id) {

return customerRepository.findById(id);

}

public Customer updateCustomer(Customer customer) {

return customerRepository.save(customer);

}

public void deleteCustomer(Long id) {

customerRepository.deleteById(id);

}

}

**BookModel.java:**

package com.example.bookstore.model;

import com.example.bookstore.entity.Book;

import org.springframework.hateoas.RepresentationModel;

public class BookModel extends RepresentationModel<BookModel> {

private Long id;

private String title;

private String author;

private double price;

public BookModel(Book book) {

this.id = book.getId();

this.title = book.getTitle();

this.author = book.getAuthor();

this.price = book.getPrice();

}

}

**CustomerModel.java:**

package com.example.bookstore.model;

import com.example.bookstore.entity.Customer;

import org.springframework.hateoas.RepresentationModel;

public class CustomerModel extends RepresentationModel<CustomerModel> {

private Long id;

private String name;

private String email;

public CustomerModel(Customer customer) {

this.id = customer.getId();

this.name = customer.getName();

this.email = customer.getEmail();

}

}

**BookModelAssembler.java:**

package com.example.bookstore.assembler;

import com.example.bookstore.controller.BookController;

import com.example.bookstore.entity.Book;

import com.example.bookstore.model.BookModel;

import org.springframework.hateoas.server.RepresentationModelAssembler;

import org.springframework.stereotype.Component;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.\*;

@Component

public class BookModelAssembler implements RepresentationModelAssembler<Book, BookModel> {

@Override

public BookModel toModel(Book book) {

BookModel model = new BookModel(book);

model.add(linkTo(methodOn(BookController.class).getBookById(book.getId())).withSelfRel());

model.add(linkTo(methodOn(BookController.class).getAllBooks()).withRel("books"));

return model;

}

}

**CustomerModelAssembler.java:**

package com.example.bookstore.assembler;

import com.example.bookstore.controller.CustomerController;

import com.example.bookstore.entity.Customer;

import com.example.bookstore.model.CustomerModel;

import org.springframework.hateoas.server.RepresentationModelAssembler;

import org.springframework.stereotype.Component;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.\*;

@Component

public class CustomerModelAssembler implements RepresentationModelAssembler<Customer, CustomerModel> {

@Override

public CustomerModel toModel(Customer customer) {

CustomerModel model = new CustomerModel(customer);

model.add(linkTo(methodOn(CustomerController.class).getCustomerById(customer.getId())).withSelfRel());

model.add(linkTo(methodOn(CustomerController.class).getAllCustomers()).withRel("customers"));

return model;

}

}

**BookController.java:**

package com.example.bookstore.controller;

import com.example.bookstore.assembler.BookModelAssembler;

import com.example.bookstore.entity.Book;

import com.example.bookstore.model.BookModel;

import com.example.bookstore.service.BookService;

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

import org.springframework.hateoas.CollectionModel;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import java.util.List;

import java.util.stream.Collectors;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.\*;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@Autowired

private BookModelAssembler bookModelAssembler;

@PostMapping

public ResponseEntity<BookModel> createBook(@Validated @RequestBody Book book) {

Book savedBook = bookService.createBook(book);

BookModel bookModel = bookModelAssembler.toModel(savedBook);

return ResponseEntity.ok(bookModel);

}

@GetMapping

public ResponseEntity<CollectionModel<BookModel>> getAllBooks() {

List<Book> books = bookService.getAllBooks();

List<BookModel> bookModels = books.stream()

.map(bookModelAssembler::toModel)

.collect(Collectors.toList());

return ResponseEntity.ok(CollectionModel.of(bookModels, linkTo(methodOn(BookController.class).getAllBooks()).withSelfRel()));

}

@GetMapping("/{id}")

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

return bookService.getBookById(id)

.map(bookModelAssembler::toModel)

.map(ResponseEntity::ok)

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

}

@PutMapping("/{id}")

public ResponseEntity<BookModel> updateBook(@PathVariable Long id, @Validated @RequestBody Book book) {

if (!bookService.getBookById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

book.setId(id);

Book updatedBook = bookService.updateBook(book);

BookModel bookModel = bookModelAssembler.toModel(updatedBook);

return ResponseEntity.ok(bookModel);

}

@DeleteMapping("/{id}")

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

if (!bookService.getBookById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

**CustomerController.java:**

package com.example.bookstore.controller;

import com.example.bookstore.assembler.CustomerModelAssembler;

import com.example.bookstore.entity.Customer;

import com.example.bookstore.model.CustomerModel;

import com.example.bookstore.service.CustomerService;

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

import org.springframework.hateoas.CollectionModel;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import java.util.List;

import java.util.stream.Collectors;

import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.\*;

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@Autowired

private CustomerModelAssembler customerModelAssembler;

@PostMapping

public ResponseEntity<CustomerModel> createCustomer(@Validated @RequestBody Customer customer) {

Customer savedCustomer = customerService.createCustomer(customer);

CustomerModel customerModel = customerModelAssembler.toModel(savedCustomer);

return ResponseEntity.ok(customerModel);

}

@GetMapping

public ResponseEntity<CollectionModel<CustomerModel>> getAllCustomers() {

List<Customer> customers = customerService.getAllCustomers();

List<CustomerModel> customerModels = customers.stream()

.map(customerModelAssembler::toModel)

.collect(Collectors.toList());

return ResponseEntity.ok(CollectionModel.of(customerModels, linkTo(methodOn(CustomerController.class).getAllCustomers()).withSelfRel()));

}

@GetMapping("/{id}")

public ResponseEntity<CustomerModel> getCustomerById(@PathVariable Long id) {

return customerService.getCustomerById(id)

.map(customerModelAssembler::toModel)

.map(ResponseEntity::ok)

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

}

@PutMapping("/{id}")

public ResponseEntity<CustomerModel> updateCustomer(@PathVariable Long id, @Validated @RequestBody Customer customer) {

if (!customerService.getCustomerById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

customer.setId(id);

Customer updatedCustomer = customerService.updateCustomer(customer);

CustomerModel customerModel = customerModelAssembler.toModel(updatedCustomer);

return ResponseEntity.ok(customerModel);

}

@DeleteMapping("/{id}")

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

if (!customerService.getCustomerById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

**GlobalExceptionHandler.java:**

package com.example.bookstore.exception;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.context.request.WebRequest;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(Exception.class)

public ResponseEntity<?> globalExceptionHandler(Exception ex, WebRequest request) {

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(ex.getMessage());

}

}

application.properties:

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.h2.console.enabled=true

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.show-sql=true

pom.xml:

<dependencies>

<!-- Spring Boot Starter Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-hateoas</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Test Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

**Exercise 10: Online Bookstore - Configuring Content Negotiation**

**Business Scenario:**

Support different media types (JSON, XML) for your bookstore's RESTful services.

**Instructions:**

1. **Content Negotiation:**
   * Configure Spring Boot to support content negotiation.
2. **Accept Header:**
   * Implement logic to produce and consume different media types based on the Accept header.

**BookstoreApplication.java:**

package com.example.bookstore;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApplication.class, args);

}

}

**Book.java:**

package com.example.bookstore.entity;

import jakarta.persistence.\*;

import jakarta.validation.constraints.NotNull;

import jakarta.validation.constraints.Size;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 2, max = 100)

private String title;

@NotNull

@Size(min = 2, max = 100)

private String author;

@NotNull

private double price;

@Version

private int version;

}

**Customer.java:**

package com.example.bookstore.entity;

import jakarta.persistence.\*;

import jakarta.validation.constraints.Email;

import jakarta.validation.constraints.NotNull;

import jakarta.validation.constraints.Size;

@Entity

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@NotNull

@Size(min = 2, max = 100)

private String name;

@NotNull

@Email

private String email;

@Version

private int version;

}

**BookRepository.java:**

package com.example.bookstore.repository;

import com.example.bookstore.entity.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

CustomerRepository.java:

package com.example.bookstore.repository;

import com.example.bookstore.entity.Customer;

import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends JpaRepository<Customer, Long> {

}

**BookService.java:**

package com.example.bookstore.service;

import com.example.bookstore.entity.Book;

import com.example.bookstore.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public Book createBook(Book book) {

return bookRepository.save(book);

}

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Optional<Book> getBookById(Long id) {

return bookRepository.findById(id);

}

public Book updateBook(Book book) {

return bookRepository.save(book);

}

public void deleteBook(Long id) {

bookRepository.deleteById(id);

}

}

**CustomerService.java:**

package com.example.bookstore.service;

import com.example.bookstore.entity.Customer;

import com.example.bookstore.repository.CustomerRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class CustomerService {

@Autowired

private CustomerRepository customerRepository;

public Customer createCustomer(Customer customer) {

return customerRepository.save(customer);

}

public List<Customer> getAllCustomers() {

return customerRepository.findAll();

}

public Optional<Customer> getCustomerById(Long id) {

return customerRepository.findById(id);

}

public Customer updateCustomer(Customer customer) {

return customerRepository.save(customer);

}

public void deleteCustomer(Long id) {

customerRepository.deleteById(id);

}

}

**BookController.java:**

package com.example.bookstore.controller;

import com.example.bookstore.entity.Book;

import com.example.bookstore.service.BookService;

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

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import java.util.List;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@PostMapping(consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE},

produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public ResponseEntity<Book> createBook(@Validated @RequestBody Book book) {

Book savedBook = bookService.createBook(book);

return ResponseEntity.ok(savedBook);

}

@GetMapping(produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public ResponseEntity<List<Book>> getAllBooks() {

List<Book> books = bookService.getAllBooks();

return ResponseEntity.ok(books);

}

@GetMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

return bookService.getBookById(id)

.map(ResponseEntity::ok)

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

}

@PutMapping(value = "/{id}", consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE},

produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

if (!bookService.getBookById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

book.setId(id);

Book updatedBook = bookService.updateBook(book);

return ResponseEntity.ok(updatedBook);

}

@DeleteMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

if (!bookService.getBookById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

**CustomerController.java:**

package com.example.bookstore.controller;

import com.example.bookstore.entity.Customer;

import com.example.bookstore.service.CustomerService;

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

import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;

import org.springframework.validation.annotation.Validated;

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

import java.util.List;

@RestController

@RequestMapping("/api/customers")

public class CustomerController {

@Autowired

private CustomerService customerService;

@PostMapping(consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE},

produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public ResponseEntity<Customer> createCustomer(@Validated @RequestBody Customer customer) {

Customer savedCustomer = customerService.createCustomer(customer);

return ResponseEntity.ok(savedCustomer);

}

@GetMapping(produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public ResponseEntity<List<Customer>> getAllCustomers() {

List<Customer> customers = customerService.getAllCustomers();

return ResponseEntity.ok(customers);

}

@GetMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public ResponseEntity<Customer> getCustomerById(@PathVariable Long id) {

return customerService.getCustomerById(id)

.map(ResponseEntity::ok)

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

}

@PutMapping(value = "/{id}", consumes = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE},

produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

public ResponseEntity<Customer> updateCustomer(@PathVariable Long id, @Validated @RequestBody Customer customer) {

if (!customerService.getCustomerById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

customer.setId(id);

Customer updatedCustomer = customerService.updateCustomer(customer);

return ResponseEntity.ok(updatedCustomer);

}

@DeleteMapping(value = "/{id}", produces = {MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE})

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

if (!customerService.getCustomerById(id).isPresent()) {

return ResponseEntity.notFound().build();

}

customerService.deleteCustomer(id);

return ResponseEntity.noContent().build();

}

}

**Pom.xml:**

<dependencies>

<!-- Spring Boot Starter Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<!-- Jackson DataFormat XML -->

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Test Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

**GlobalExceptionHandler.java:**

package com.example.bookstore.exception;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import org.springframework.web.context.request.WebRequest;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(Exception.class)

public ResponseEntity<?> globalExceptionHandler(Exception ex, WebRequest request) {

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(ex.getMessage());

}

}

application.properties:

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.h2.console.enabled=true

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.show-sql=true

**Exercise 11: Online Bookstore - Integrating Spring Boot Actuator**

**Business Scenario:**

Monitor and manage your bookstore's RESTful services using Spring Boot Actuator.

**Instructions:**

1. **Add Actuator Dependency:**
   * Include the Spring Boot Actuator dependency in your project.
2. **Expose Actuator Endpoints:**
   * Enable and customize Actuator endpoints.
3. **Custom Metrics:**
   * Expose custom metrics for monitoring your application.

**Pom.xml:**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.yourpackage</groupId>

<artifactId>online-bookstore</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

application.properties:

management.endpoints.web.exposure.include=\*

management.endpoint.health.show-details=always

management.endpoints.web.base-path=/actuator

**BookstoreApiApplication.java:**

package com.yourpackage;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreApiApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApiApplication.class, args);

}

}

**CustomMetrics.java:**

package com.yourpackage;

import io.micrometer.core.instrument.MeterRegistry;

import org.springframework.stereotype.Component;

@Component

public class CustomMetrics {

public CustomMetrics(MeterRegistry meterRegistry) {

meterRegistry.gauge("custom.metric.active.users", this, CustomMetrics::getActiveUsers);

}

public int getActiveUsers() {

// Example logic to return the number of active users

return 42; // Replace with actual logic

}

}

**BookController.java:**

package com.yourpackage.controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class BookController {

@GetMapping("/books")

public String getBooks() {

return "List of books";

}

}

**BookService.java:**

package com.yourpackage.service;

import com.yourpackage.model.Book;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

import java.util.List;

@Service

public class BookService {

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

public BookService() {

// Initialize with some dummy books

books.add(new Book(1L, "Effective Java", "Joshua Bloch", 42.0));

books.add(new Book(2L, "Spring in Action", "Craig Walls", 38.5));

books.add(new Book(3L, "Clean Code", "Robert C. Martin", 45.0));

}

public List<Book> getAllBooks() {

return books;

}

public Book getBookById(Long id) {

return books.stream()

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

.findFirst()

.orElse(null);

}

public void addBook(Book book) {

books.add(book);

}

public void updateBook(Long id, Book updatedBook) {

Book existingBook = getBookById(id);

if (existingBook != null) {

existingBook.setTitle(updatedBook.getTitle());

existingBook.setAuthor(updatedBook.getAuthor());

existingBook.setPrice(updatedBook.getPrice());

}

}

public void deleteBook(Long id) {

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

}

}

**BookstoreApiApplication.java:**

package com.yourpackage;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreApiApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApiApplication.class, args);

}

}

**Book.java:**

package com.yourpackage.model;

public class Book {

private Long id;

private String title;

private String author;

private Double price;

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

this.id = id;

this.title = title;

this.author = author;

this.price = price;

}

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;

}

}

**Exercise 12: Online Bookstore - Securing RESTful Endpoints with Spring Security**

**Business Scenario:**

Secure your bookstore's RESTful endpoints using Spring Security with JWT-based authentication.

**Instructions:**

1. **Add Spring Security:**
   * Integrate Spring Security into your project.
2. **JWT Authentication:**
   * Implement JWT-based authentication and authorization.
3. **CORS Handling:**
   * Configure CORS to handle cross-origin requests.

**Pom.xml:**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.yourpackage</groupId>

<artifactId>online-bookstore</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<dependencies>

<!-- Spring Boot dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<!-- Spring Security and JWT dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<!-- Test dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

application.properties:

# Spring Security CORS configuration

security.jwt.secret-key=mySecretKey

security.jwt.token-prefix=Bearer

security.jwt.header-string=Authorization

security.jwt.expiration-time=86400000 # 1 day

**BookstoreApiApplication.java:**

package com.yourpackage;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreApiApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApiApplication.class, args);

}

}

**JwtAuthenticationFilter.java:**

package com.yourpackage.security;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import javax.servlet.FilterChain;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.util.Date;

import static com.yourpackage.security.SecurityConstants.\*;

public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {

private final AuthenticationManager authenticationManager;

public JwtAuthenticationFilter(AuthenticationManager authenticationManager) {

this.authenticationManager = authenticationManager;

}

@Override

public Authentication attemptAuthentication(HttpServletRequest req,

HttpServletResponse res) throws AuthenticationException {

String username = req.getParameter("username");

String password = req.getParameter("password");

return authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(username, password, new ArrayList<>())

);

}

@Override

protected void successfulAuthentication(HttpServletRequest req,

HttpServletResponse res,

FilterChain chain,

Authentication auth) throws IOException {

String token = Jwts.builder()

.setSubject(((User) auth.getPrincipal()).getUsername())

.setExpiration(new Date(System.currentTimeMillis() + EXPIRATION\_TIME))

.signWith(SignatureAlgorithm.HS512, SECRET.getBytes())

.compact();

res.addHeader(HEADER\_STRING, TOKEN\_PREFIX + token);

}

}

**JwtAuthorizationFilter.java:**

package com.yourpackage.security;

import io.jsonwebtoken.Jwts;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;

import javax.servlet.FilterChain;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.util.ArrayList;

import static com.yourpackage.security.SecurityConstants.\*;

public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

public JwtAuthorizationFilter(AuthenticationManager authManager) {

super(authManager);

}

@Override

protected void doFilterInternal(HttpServletRequest req,

HttpServletResponse res,

FilterChain chain) throws IOException {

String header = req.getHeader(HEADER\_STRING);

if (header == null || !header.startsWith(TOKEN\_PREFIX)) {

chain.doFilter(req, res);

return;

}

UsernamePasswordAuthenticationToken authentication = getAuthentication(req);

SecurityContextHolder.getContext().setAuthentication(authentication);

chain.doFilter(req, res);

}

private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {

String token = request.getHeader(HEADER\_STRING);

if (token != null) {

String user = Jwts.parser()

.setSigningKey(SECRET.getBytes())

.parseClaimsJws(token.replace(TOKEN\_PREFIX, ""))

.getBody()

.getSubject();

if (user != null) {

return new UsernamePasswordAuthenticationToken(user, null, new ArrayList<>());

}

return null;

}

return null;

}

}

**SecurityConstants.java:**

package com.yourpackage.security;

public class SecurityConstants {

public static final String SECRET = "mySecretKey";

public static final long EXPIRATION\_TIME = 864\_000\_000; // 10 days

public static final String TOKEN\_PREFIX = "Bearer ";

public static final String HEADER\_STRING = "Authorization";

}

**WebSecurityConfig.java:**

package com.yourpackage.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

private final UserDetailsServiceImpl userDetailsService;

private final BCryptPasswordEncoder bCryptPasswordEncoder;

public WebSecurityConfig(UserDetailsServiceImpl userDetailsService, BCryptPasswordEncoder bCryptPasswordEncoder) {

this.userDetailsService = userDetailsService;

this.bCryptPasswordEncoder = bCryptPasswordEncoder;

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.cors().and().csrf().disable()

.authorizeRequests()

.antMatchers(HttpMethod.POST, "/login").permitAll()

.anyRequest().authenticated()

.and()

.addFilter(new JwtAuthenticationFilter(authenticationManager()))

.addFilter(new JwtAuthorizationFilter(authenticationManager()))

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder);

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

**UserDetailsServiceImpl.java:**

package com.yourpackage.security;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

@Service

public class UserDetailsServiceImpl implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

// Here you would typically load user details from a database or other source

return new User("user", "$2a$10$eBq0gjgk93.ZQDPF3o4GOuGyQi6d8dOZa9U7PqGTxCJ.OZmnG6Mlm", new ArrayList<>());

// Password is "password" hashed using BCrypt

}

}

**AuthController.java:**

package com.yourpackage.controller;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class AuthController {

@PostMapping("/login")

public void login(@RequestBody UserCredentials userCredentials) {

// Login is handled automatically by JwtAuthenticationFilter

}

public static class UserCredentials {

private String username;

private String password;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

}

**BookController.java:**

package com.yourpackage.controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/books")

public class BookController {

@GetMapping

public String getBooks() {

return "List of books";

}

}

**Exercise 13: Online Bookstore - Unit Testing REST Controllers**

**Business Scenario:**

Write unit tests for your bookstore's REST controllers using JUnit and Mockito.

**Instructions:**

1. **JUnit Setup:**
   * Set up JUnit and Mockito in your project.
2. **MockMvc:**
   * Use MockMvc to write unit tests for your REST controllers.
3. **Test Coverage:**
   * Ensure comprehensive test coverage and follow best practices for testing

**pom.xml:**

<dependencies>

<!-- Spring Boot Starter Test -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!-- Mockito for mocking objects -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

**BookController.java:**

package com.example.bookstore.controller;

import com.example.bookstore.model.Book;

import com.example.bookstore.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;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping

public ResponseEntity<List<Book>> getAllBooks() {

List<Book> books = bookService.getAllBooks();

return new ResponseEntity<>(books, HttpStatus.OK);

}

@GetMapping("/{id}")

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

Book book = bookService.getBookById(id);

return new ResponseEntity<>(book, HttpStatus.OK);

}

@PostMapping

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

Book createdBook = bookService.createBook(book);

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

}

@PutMapping("/{id}")

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

Book updatedBook = bookService.updateBook(id, book);

return new ResponseEntity<>(updatedBook, HttpStatus.OK);

}

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

}

**BookControllerTest.java:**

package com.example.bookstore.controller;

import com.example.bookstore.model.Book;

import com.example.bookstore.service.BookService;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

import java.util.Arrays;

import java.util.Optional;

import static org.mockito.Mockito.when;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(BookController.class)

public class BookControllerTest {

private MockMvc mockMvc;

@Mock

private BookService bookService;

@InjectMocks

private BookController bookController;

@BeforeEach

void setUp() {

MockitoAnnotations.openMocks(this);

mockMvc = MockMvcBuilders.standaloneSetup(bookController).build();

}

@Test

void testGetAllBooks() throws Exception {

Book book1 = new Book(1L, "Book One", "Author One");

Book book2 = new Book(2L, "Book Two", "Author Two");

when(bookService.getAllBooks()).thenReturn(Arrays.asList(book1, book2));

mockMvc.perform(get("/api/books"))

.andExpect(status().isOk())

.andExpect(content().contentType(MediaType.APPLICATION\_JSON))

.andExpect(jsonPath("$[0].title").value("Book One"))

.andExpect(jsonPath("$[1].title").value("Book Two"));

}

@Test

void testGetBookById() throws Exception {

Book book = new Book(1L, "Book One", "Author One");

when(bookService.getBookById(1L)).thenReturn(book);

mockMvc.perform(get("/api/books/1"))

.andExpect(status().isOk())

.andExpect(content().contentType(MediaType.APPLICATION\_JSON))

.andExpect(jsonPath("$.title").value("Book One"));

}

@Test

void testCreateBook() throws Exception {

Book book = new Book(1L, "New Book", "New Author");

when(bookService.createBook(book)).thenReturn(book);

mockMvc.perform(post("/api/books")

.contentType(MediaType.APPLICATION\_JSON)

.content(new ObjectMapper().writeValueAsString(book)))

.andExpect(status().isCreated())

.andExpect(content().contentType(MediaType.APPLICATION\_JSON))

.andExpect(jsonPath("$.title").value("New Book"));

}

@Test

void testUpdateBook() throws Exception {

Book book = new Book(1L, "Updated Book", "Updated Author");

when(bookService.updateBook(1L, book)).thenReturn(book);

mockMvc.perform(put("/api/books/1")

.contentType(MediaType.APPLICATION\_JSON)

.content(new ObjectMapper().writeValueAsString(book)))

.andExpect(status().isOk())

.andExpect(content().contentType(MediaType.APPLICATION\_JSON))

.andExpect(jsonPath("$.title").value("Updated Book"));

}

@Test

void testDeleteBook() throws Exception {

mockMvc.perform(delete("/api/books/1"))

.andExpect(status().isNoContent());

}

}

**BookstoreApplication.java:**

package com.example.bookstore;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApplication.class, args);

}

}

**Book.java:**

package com.example.bookstore.model;

public class Book {

private Long id;

private String title;

private String author;

public Book() {}

public Book(Long id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

}

// 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;

}

}

**BookRepository.java:**

package com.example.bookstore.repository;

import com.example.bookstore.model.Book;

import org.springframework.stereotype.Repository;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

@Repository

public class BookRepository {

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

public List<Book> findAll() {

return books;

}

public Optional<Book> findById(Long id) {

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

}

public Book save(Book book) {

books.add(book);

return book;

}

public void deleteById(Long id) {

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

}

}

**BookService.java:**

package com.example.bookstore.service;

import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Book getBookById(Long id) {

return bookRepository.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));

}

public Book createBook(Book book) {

return bookRepository.save(book);

}

public Book updateBook(Long id, Book book) {

Book existingBook = getBookById(id);

existingBook.setTitle(book.getTitle());

existingBook.setAuthor(book.getAuthor());

return bookRepository.save(existingBook);

}

public void deleteBook(Long id) {

bookRepository.deleteById(id);

}

}

application.properties:

# Server port

server.port=8080

# Logging configuration

logging.level.org.springframework.web=DEBUG

logging.level.com.example.bookstore=DEBUG

# Hibernate (if using JPA)

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

# Database configuration (example using H2 in-memory database)

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.h2.console.enabled=true

# Other Spring Boot settings (optional)

spring.application.name=bookstore-api

**Exercise 14: Online Bookstore - Integration Testing for REST Services**

**Business Scenario:**

Write integration tests for your bookstore's RESTful services.

**Instructions:**

1. **Spring Test:**
   * Set up Spring Test for integration testing.
2. **MockMvc Integration:**
   * Use MockMvc for end-to-end testing of your REST endpoints.
3. **Database Integration:**
   * Include database integration in your tests using an in-memory database like **H2**.

**Pom.xml:**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>bookstore-api</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>bookstore-api</name>

<description>Bookstore API Application</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Boot Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Testing Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

application.properties:

# Server port

server.port=8080

# Logging configuration

logging.level.org.springframework.web=DEBUG

logging.level.com.example.bookstore=DEBUG

# Hibernate configuration

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

# Database configuration (example using H2 in-memory database)

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.h2.console.enabled=true

# Application name

spring.application.name=bookstore-api

application-test.properties:

# Test-specific properties

spring.datasource.url=jdbc:h2:mem:testdb;DB\_CLOSE\_DELAY=-1

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=true

spring.h2.console.enabled=true

**BookController.java:**

package com.example.bookstore.controller;

import com.example.bookstore.model.Book;

import com.example.bookstore.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;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping

public ResponseEntity<List<Book>> getAllBooks() {

List<Book> books = bookService.getAllBooks();

return new ResponseEntity<>(books, HttpStatus.OK);

}

@GetMapping("/{id}")

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

Book book = bookService.getBookById(id);

return new ResponseEntity<>(book, HttpStatus.OK);

}

@PostMapping

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

Book createdBook = bookService.createBook(book);

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

}

@PutMapping("/{id}")

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

Book updatedBook = bookService.updateBook(id, book);

return new ResponseEntity<>(updatedBook, HttpStatus.OK);

}

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

}

**Book.java:**

package com.example.bookstore.model;

public class Book {

private Long id;

private String title;

private String author;

public Book() {}

public Book(Long id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

}

// 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;

}

}

**BookRepository.java:**

package com.example.bookstore.repository;

import com.example.bookstore.model.Book;

import org.springframework.stereotype.Repository;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

@Repository

public class BookRepository {

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

public List<Book> findAll() {

return books;

}

public Optional<Book> findById(Long id) {

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

}

public Book save(Book book) {

books.add(book);

return book;

}

public void deleteById(Long id) {

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

}

}

**BookService.java:**

package com.example.bookstore.service;

import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Book getBookById(Long id) {

return bookRepository.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));

}

public Book createBook(Book book) {

return bookRepository.save(book);

}

public Book updateBook(Long id, Book book) {

Book existingBook = getBookById(id);

existingBook.setTitle(book.getTitle());

existingBook.setAuthor(book.getAuthor());

return bookRepository.save(existingBook);

}

public void deleteBook(Long id) {

bookRepository.deleteById(id);

}

}

**BookControllerIntegrationTest.java:**

package com.example.bookstore.controller;

import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import org.springframework.web.context.WebApplicationContext;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@SpringBootTest

public class BookControllerIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private BookRepository bookRepository;

@BeforeEach

void setUp() {

bookRepository.deleteAll();

}

@Test

void testCreateBook() throws Exception {

Book book = new Book(1L, "Integration Book", "Integration Author");

mockMvc.perform(post("/api/books")

.contentType(MediaType.APPLICATION\_JSON)

.content(new ObjectMapper().writeValueAsString(book)))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.title").value("Integration Book"));

}

@Test

void testGetAllBooks() throws Exception {

Book book = new Book(1L, "Book One", "Author One");

bookRepository.save(book);

mockMvc.perform(get("/api/books"))

.andExpect(status().isOk())

.andExpect(jsonPath("$[0].title").value("Book One"));

}

@Test

void testGetBookById() throws Exception {

Book book = new Book(1L, "Book One", "Author One");

bookRepository.save(book);

mockMvc.perform(get("/api/books/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Book One"));

}

@Test

void testUpdateBook() throws Exception {

Book book = new Book(1L, "Book One", "Author One");

bookRepository.save(book);

Book updatedBook = new Book(1L, "Updated Book", "Updated Author");

mockMvc.perform(put("/api/books/1")

.contentType(MediaType.APPLICATION\_JSON)

.content(new ObjectMapper().writeValueAsString(updatedBook)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Updated Book"));

}

@Test

void testDeleteBook() throws Exception {

Book book = new Book(1L, "Book One", "Author One");

bookRepository.save(book);

mockMvc.perform(delete("/api/books/1"))

.andExpect(status().isNoContent());

mockMvc.perform(get("/api/books/1"))

.andExpect(status().isNotFound());

}

}

**BookRepositoryIntegrationTest.java:**

package com.example.bookstore.repository;

import com.example.bookstore.model.Book;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase;

import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;

@SpringBootTest

@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.ANY)

public class BookRepositoryIntegrationTest {

@Autowired

private BookRepository bookRepository;

@BeforeEach

void setUp() {

bookRepository.deleteAll();

}

@Test

void testSaveAndFindBook() {

Book book = new Book(1L, "Test Book", "Test Author");

bookRepository.save(book);

List<Book> books = bookRepository.findAll();

assertEquals(1, books.size());

assertEquals("Test Book", books.get(0).getTitle());

}

@Test

void testDeleteBook() {

Book book = new Book(1L, "Test Book", "Test Author");

bookRepository.save(book);

bookRepository.deleteById(1L);

List<Book> books = bookRepository.findAll();

assertEquals(0, books.size());

}

}

**BookServiceIntegrationTest.java:**

package com.example.bookstore.service;

import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

import static org.junit.jupiter.api.Assertions.assertEquals;

@SpringBootTest

public class BookServiceIntegrationTest {

@Autowired

private BookService bookService;

@Autowired

private BookRepository bookRepository;

@BeforeEach

void setUp() {

bookRepository.deleteAll();

}

@Test

void testCreateBook() {

Book book = new Book(1L, "Test Book", "Test Author");

Book createdBook = bookService.createBook(book);

assertEquals("Test Book", createdBook.getTitle());

}

@Test

void testGetAllBooks() {

Book book1 = new Book(1L, "Book One", "Author One");

Book book2 = new Book(2L, "Book Two", "Author Two");

bookService.createBook(book1);

bookService.createBook(book2);

List<Book> books = bookService.getAllBooks();

assertEquals(2, books.size());

}

@Test

void testUpdateBook() {

Book book = new Book(1L, "Old Title", "Old Author");

bookService.createBook(book);

Book updatedBook = new Book(1L, "New Title", "New Author");

Book result = bookService.updateBook(1L, updatedBook);

assertEquals("New Title", result.getTitle());

}

@Test

void testDeleteBook() {

Book book = new Book(1L, "Book to Delete", "Author");

bookService.createBook(book);

bookService.deleteBook(1L);

List<Book> books = bookService.getAllBooks();

assertEquals(0, books.size());

}

}

**Scenario 15: Online Bookstore - API Documentation with Swagger**

**Business Scenario:**

Document your bookstore's REST APIs using Swagger and Springdoc.

**Instructions:**

1. **Add Swagger Dependency:**
   * Include Swagger or Springdoc dependencies in your project.
2. **Document Endpoints:**
   * Annotate your REST controllers and methods to generate API documentation.
3. **API Documentation:**
   * Generate and review the API documentation using **Swagger UI** or **Springdoc UI**.

**BookstoreApiApplication.java:**

package com.example.bookstore;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookstoreApiApplication {

public static void main(String[] args) {

SpringApplication.run(BookstoreApiApplication.class, args);

}

}

**BookController.java:**

package com.example.bookstore.controller;

import com.example.bookstore.model.Book;

import com.example.bookstore.service.BookService;

import io.swagger.v3.oas.annotations.Operation;

import io.swagger.v3.oas.annotations.Parameter;

import io.swagger.v3.oas.annotations.tags.Tag;

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;

@RestController

@RequestMapping("/api/books")

@Tag(name = "Book Management", description = "Endpoints for managing books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping

@Operation(summary = "Get all books", description = "Retrieve a list of all books")

public ResponseEntity<List<Book>> getAllBooks() {

List<Book> books = bookService.getAllBooks();

return new ResponseEntity<>(books, HttpStatus.OK);

}

@GetMapping("/{id}")

@Operation(summary = "Get book by ID", description = "Retrieve a book by its ID")

public ResponseEntity<Book> getBookById(

@Parameter(description = "ID of the book to be retrieved") @PathVariable Long id) {

Book book = bookService.getBookById(id);

return new ResponseEntity<>(book, HttpStatus.OK);

}

@PostMapping

@Operation(summary = "Create a new book", description = "Add a new book to the system")

public ResponseEntity<Book> createBook(

@Parameter(description = "Book object that needs to be added") @RequestBody Book book) {

Book createdBook = bookService.createBook(book);

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

}

@PutMapping("/{id}")

@Operation(summary = "Update an existing book", description = "Update the details of a book")

public ResponseEntity<Book> updateBook(

@Parameter(description = "ID of the book to be updated") @PathVariable Long id,

@Parameter(description = "Updated book object") @RequestBody Book book) {

Book updatedBook = bookService.updateBook(id, book);

return new ResponseEntity<>(updatedBook, HttpStatus.OK);

}

@DeleteMapping("/{id}")

@Operation(summary = "Delete a book", description = "Remove a book from the system by its ID")

public ResponseEntity<Void> deleteBook(

@Parameter(description = "ID of the book to be deleted") @PathVariable Long id) {

bookService.deleteBook(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

}

}

**Book.java:**

package com.example.bookstore.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

// Constructors, getters, and setters

public Book() {

}

public Book(Long id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

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;

}

}

**BookRepository.java:**

package com.example.bookstore.repository;

import com.example.bookstore.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

**BookService.java:**

package com.example.bookstore.service;

import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

public Book getBookById(Long id) {

return bookRepository.findById(id).orElse(null);

}

public Book createBook(Book book) {

return bookRepository.save(book);

}

public Book updateBook(Long id, Book book) {

book.setId(id);

return bookRepository.save(book);

}

public void deleteBook(Long id) {

bookRepository.deleteById(id);

}

}

**SwaggerConfig.java:**

package com.example.bookstore.config;

import org.springdoc.core.GroupedOpenApi;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class SwaggerConfig {

@Bean

public GroupedOpenApi publicApi() {

return GroupedOpenApi.builder()

.group("bookstore-api")

.packagesToScan("com.example.bookstore")

.build();

}

}

**BookControllerIntegrationTest.java:**

package com.example.bookstore.controller;

import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import org.springframework.web.context.WebApplicationContext;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@SpringBootTest

public class BookControllerIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private BookRepository bookRepository;

@BeforeEach

void setUp() {

bookRepository.deleteAll();

}

@Test

void testCreateBook() throws Exception {

Book book = new Book(1L, "Integration Book", "Integration Author");

mockMvc.perform(post("/api/books")

.contentType(MediaType.APPLICATION\_JSON)

.content(new ObjectMapper().writeValueAsString(book)))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.title").value("Integration Book"));

}

@Test

void testGetAllBooks() throws Exception {

Book book = new Book(1L, "Book One", "Author One");

bookRepository.save(book);

mockMvc.perform(get("/api/books"))

.andExpect(status().isOk())

.andExpect(jsonPath("$[0].title").value("Book One"));

}

@Test

void testGetBookById() throws Exception {

Book book = new Book(1L, "Book One", "Author One");

bookRepository.save(book);

mockMvc.perform(get("/api/books/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Book One"));

}

@Test

void testUpdateBook() throws Exception {

Book book = new Book(1L, "Book One", "Author One");

bookRepository.save(book);

Book updatedBook = new Book(1L, "Updated Book", "Updated Author");

mockMvc.perform(put("/api/books/1")

.contentType(MediaType.APPLICATION\_JSON)

.content(new ObjectMapper().writeValueAsString(updatedBook)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Updated Book"));

}

@Test

void testDeleteBook() throws Exception {

Book book = new Book(1L, "Book One", "Author One");

bookRepository.save(book);

mockMvc.perform(delete("/api/books/1"))

.andExpect(status().isNoContent());

}

}

**BookRepositoryIntegrationTest.java:**

package com.example.bookstore.repository;

import com.example.bookstore.model.Book;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;

import java.util.Optional;

import static org.assertj.core.api.Assertions.assertThat;

@DataJpaTest

public class BookRepositoryIntegrationTest {

@Autowired

private BookRepository bookRepository;

@Test

void testSaveAndFindBook() {

Book book = new Book(null, "Test Book", "Test Author");

book = bookRepository.save(book);

Optional<Book> foundBook = bookRepository.findById(book.getId());

assertThat(foundBook).isPresent();

assertThat(foundBook.get().getTitle()).isEqualTo("Test Book");

}

}

**BookServiceIntegrationTest.java:**

package com.example.bookstore.service;

import com.example.bookstore.model.Book;

import com.example.bookstore.repository.BookRepository;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest

public class BookServiceIntegrationTest {

@Autowired

private BookService bookService;

@Autowired

private BookRepository bookRepository;

@Test

void testGetAllBooks() {

Book book1 = new Book(null, "Book One", "Author One");

Book book2 = new Book(null, "Book Two", "Author Two");

bookRepository.save(book1);

bookRepository.save(book2);

List<Book> books = bookService.getAllBooks();

assertThat(books).hasSize(2);

assertThat(books).extracting(Book::getTitle).contains("Book One", "Book Two");

}

}

pom.xml:

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/POM/4.0.0">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>bookstore-api</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>bookstore-api</name>

<description>Bookstore API Application</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Boot Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Springdoc OpenAPI -->

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>2.0.2</version>

</dependency>

<!-- Testing Dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

application.properties:

# Server port

server.port=8080

# Logging configuration

logging.level.org.springframework.web=DEBUG

logging.level.com.example.bookstore=DEBUG

# Hibernate configuration

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

# Database configuration (example using H2 in-memory database)

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.h2.console.enabled=true

# Application name

spring.application.name=bookstore-api