**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**

<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>

<scope>test</scope>

</dependency>

**BookController.java**

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.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 ResponseEntity.ok(books);

}

@GetMapping("/{id}")

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

Book book = bookService.getBookById(id);

return ResponseEntity.ok(book);

}

@PostMapping

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

Book createdBook = bookService.createBook(book);

return ResponseEntity.ok(createdBook);

}

@PutMapping("/{id}")

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

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

return ResponseEntity.ok(updatedBook);

}

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

return ResponseEntity.noContent().build();

}

}

**BookControllerTest.java**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.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.beans.factory.annotation.Autowired;

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.Collections;

import static org.mockito.ArgumentMatchers.any;

import static org.mockito.ArgumentMatchers.anyLong;

import static org.mockito.Mockito.when;

@WebMvcTest(BookController.class)

public class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@Mock

private BookService bookService;

@InjectMocks

private BookController bookController;

private ObjectMapper objectMapper;

@BeforeEach

public void setup() {

MockitoAnnotations.openMocks(this);

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

objectMapper = new ObjectMapper();

}

@Test

public void testGetAllBooks() throws Exception {

Book book = new Book(1L, "Test Book", "Author", 9.99);

when(bookService.getAllBooks()).thenReturn(Collections.singletonList(book));

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

.accept(MediaType.APPLICATION\_JSON))

.andExpect(MockMvcResultMatchers.status().isOk())

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

}

@Test

public void testGetBookById() throws Exception {

Book book = new Book(1L, "Test Book", "Author", 9.99);

when(bookService.getBookById(anyLong())).thenReturn(book);

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

.accept(MediaType.APPLICATION\_JSON))

.andExpect(MockMvcResultMatchers.status().isOk())

.andExpect(MockMvcResultMatchers.jsonPath("$.title").value("Test Book"));

}

@Test

public void testCreateBook() throws Exception {

Book book = new Book(1L, "Test Book", "Author", 9.99);

when(bookService.createBook(any(Book.class))).thenReturn(book);

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

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(book)))

.andExpect(MockMvcResultMatchers.status().isOk())

.andExpect(MockMvcResultMatchers.jsonPath("$.title").value("Test Book"));

}

@Test

public void testUpdateBook() throws Exception {

Book book = new Book(1L, "Updated Book", "Author", 19.99);

when(bookService.updateBook(anyLong(), any(Book.class))).thenReturn(book);

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

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(book)))

.andExpect(MockMvcResultMatchers.status().isOk())

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

}

@Test

public void testDeleteBook() throws Exception {

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

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

}

}