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

1. **Set Up JUnit and Mockito**

<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>

**2. Using MockMvc to Write Unit Tests**

**MockMvc** allows you to simulate HTTP requests and test your REST controllers without starting a full web server.

**Example Test for the BookController:**

Here’s a step-by-step example of how you can use MockMvc and Mockito to test your BookController.

**Step 1: Annotate the Test Class with Necessary Annotations**

* Use @WebMvcTest to focus on testing the web layer.
* Inject MockMvc to perform requests.

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

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

import static org.mockito.Mockito.\*;

import com.example.bookstore.controller.BookController;

import com.example.bookstore.model.Book;

import com.example.bookstore.service.BookService;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

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

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

import org.springframework.boot.test.mock.mockito.MockBean;

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

import java.util.Arrays;

import java.util.List;

@WebMvcTest(BookController.class)

public class BookControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private BookService bookService;

@Test

public void testGetAllBooks() throws Exception {

// Arrange: Prepare mock data and behavior

Book book1 = new Book(1L, "Effective Java", "Joshua Bloch", 45.99);

Book book2 = new Book(2L, "Clean Code", "Robert C. Martin", 39.99);

List<Book> books = Arrays.asList(book1, book2);

when(bookService.getAllBooks()).thenReturn(books);

// Act & Assert: Perform the GET request and validate the response

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

.andExpect(status().isOk())

.andExpect(jsonPath("$.size()").value(2))

.andExpect(jsonPath("$[0].title").value("Effective Java"))

.andExpect(jsonPath("$[1].title").value("Clean Code"));

// Verify that the service method was called

verify(bookService, times(1)).getAllBooks();

}

}

Step 2: Test POST and Other Methods

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

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

import org.springframework.http.MediaType;

@Test

public void testCreateBook() throws Exception {

// Arrange

Book newBook = new Book(null, "Domain-Driven Design", "Eric Evans", 59.99);

Book savedBook = new Book(3L, "Domain-Driven Design", "Eric Evans", 59.99);

when(bookService.saveBook(any(Book.class))).thenReturn(savedBook);

String bookJson = "{ \"title\": \"Domain-Driven Design\", \"author\": \"Eric Evans\", \"price\": 59.99 }";

// Act & Assert

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

.content(bookJson)

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isCreated())

.andExpect(jsonPath("$.id").value(3))

.andExpect(jsonPath("$.title").value("Domain-Driven Design"));

}

**3. Ensure Comprehensive Test Coverage**

Best practices for unit testing REST controllers:

* **Test all HTTP methods**: Ensure you test GET, POST, PUT, DELETE for various scenarios.
* **Test edge cases**: Test for cases like invalid input, missing fields, or bad requests (400 errors).
* **Mock dependencies**: Use @MockBean to mock services so you’re not relying on actual service methods, ensuring isolated tests.
* **Validate responses**: Use jsonPath() to validate the exact structure of the JSON response.

@Test

public void testGetBookNotFound() throws Exception {

// Arrange

when(bookService.getBookById(99L)).thenThrow(new ResourceNotFoundException("Book not found"));

// Act & Assert

mockMvc.perform(get("/api/books/99"))

.andExpect(status().isNotFound())

.andExpect(jsonPath("$.message").value("Book not found"));

}

@Test

public void testGetBookNotFound() throws Exception {

// Arrange

when(bookService.getBookById(99L)).thenThrow(new ResourceNotFoundException("Book not found"));

// Act & Assert

mockMvc.perform(get("/api/books/99"))

.andExpect(status().isNotFound())

.andExpect(jsonPath("$.message").value("Book not found"));

}