**WEEK 1**

**EXERCISE 14**

**Online Bookstore - Integration Testing for REST Services**

**Business Scenario**

The task is to write integration tests for your bookstore's RESTful services to ensure that your application functions correctly as a whole. This involves setting up Spring Test, using MockMvc for end-to-end testing, and integrating a database for comprehensive testing.

**Instructions**

**1. Spring Test**

**Task:** Set up Spring Test for integration testing.

**Implementation:**

* **Add Dependencies:**

**For Maven:**

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>test</scope>

</dependency>

**Explanation:** The spring-boot-starter-test dependency provides the necessary tools for testing, including Spring Test, JUnit, and Mockito. The H2 database is used for in-memory database testing.

**2. MockMvc Integration**

**Task:** Use MockMvc for end-to-end testing of your REST endpoints.

**Implementation:**

1. **Create Integr**
2. **ation Test Class:**

**BookstoreIntegrationTest.java:**

import org.junit.jupiter.api.Test;

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

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

import org.springframework.boot.test.web.client.TestRestTemplate;

import org.springframework.boot.web.server.LocalServerPort;

import org.springframework.http.HttpEntity;

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpMethod;

import org.springframework.http.ResponseEntity;

import org.springframework.test.context.ActiveProfiles;

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

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)

@ActiveProfiles("test")

public class BookstoreIntegrationTest {

@LocalServerPort

private int port;

@Autowired

private TestRestTemplate restTemplate;

@Test

public void testGetBookById() {

String url = "http://localhost:" + port + "/books/1";

ResponseEntity<BookDTO> response = restTemplate.getForEntity(url, BookDTO.class);

assertThat(response.getStatusCodeValue()).isEqualTo(200);

assertThat(response.getBody().getTitle()).isEqualTo("Book Title");

}

@Test

public void testCreateBook() {

String url = "http://localhost:" + port + "/books";

BookDTO newBook = new BookDTO(null, "New Book", "New Author", 19.99, "0987654321");

ResponseEntity<BookDTO> response = restTemplate.postForEntity(url, newBook, BookDTO.class);

assertThat(response.getStatusCodeValue()).isEqualTo(201);

assertThat(response.getBody().getTitle()).isEqualTo("New Book");

}

@Test

public void testUpdateBook() {

String url = "http://localhost:" + port + "/books/1";

BookDTO updatedBook = new BookDTO(1L, "Updated Book", "Updated Author", 25.99, "1234567890");

HttpHeaders headers = new HttpHeaders();

HttpEntity<BookDTO> entity = new HttpEntity<>(updatedBook, headers);

ResponseEntity<BookDTO> response = restTemplate.exchange(url, HttpMethod.PUT, entity, BookDTO.class);

assertThat(response.getStatusCodeValue()).isEqualTo(200);

assertThat(response.getBody().getTitle()).isEqualTo("Updated Book");

}

@Test

public void testDeleteBook() {

String url = "http://localhost:" + port + "/books/1";

restTemplate.delete(url);

ResponseEntity<BookDTO> response = restTemplate.getForEntity(url, BookDTO.class);

assertThat(response.getStatusCodeValue()).isEqualTo(404);

}

}

**Explanation:**

* @SpringBootTest is used to load the full application context.
* TestRestTemplate is used to perform HTTP requests against your running application.
* @LocalServerPort injects the random port number for the test server.
* @ActiveProfiles("test") activates the test profile, ensuring the use of an in-memory database like H2.

**3. Database Integration**

**Task:** Include database integration in your tests using an in-memory database like H2.

**Implementation:**

1. **Set Up In-Memory Database:**

* **application-test.yml:**

spring:

datasource:

url: jdbc:h2:mem:testdb

driver-class-name: org.h2.Driver

username: sa

password: password

h2:

console:

enabled: true

jpa:

hibernate:

ddl-auto: update

show-sql: true

properties:

hibernate:

format\_sql: true

* **Explanation:** This configuration sets up an in-memory H2 database for testing purposes, enabling the H2 console and configuring JPA settings for automatic schema generation.

**Conclusion:**

By writing integration tests for the REST services, we can ensure thatthe application behaves as expected when interacting with the database and other components. Using Spring Test with MockMvc and an in-memory database like H2 provides a comprehensive testing setup