**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**

<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>

**Configuration**

# src/test/resources/application-test.properties

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

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

spring.datasource.username=sa

spring.datasource.password=password

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

spring.h2.console.enabled=true

**BookController.java**

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.repository.BookRepository;

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 BookRepository bookRepository;

@GetMapping

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

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

return ResponseEntity.ok(books);

}

@GetMapping("/{id}")

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

return bookRepository.findById(id)

.map(book -> ResponseEntity.ok().body(book))

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

}

@PostMapping

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

Book createdBook = bookRepository.save(book);

return ResponseEntity.ok(createdBook);

}

@PutMapping("/{id}")

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

return bookRepository.findById(id)

.map(existingBook -> {

existingBook.setTitle(book.getTitle());

existingBook.setAuthor(book.getAuthor());

existingBook.setPrice(book.getPrice());

Book updatedBook = bookRepository.save(existingBook);

return ResponseEntity.ok(updatedBook);

})

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

}

@DeleteMapping("/{id}")

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

return bookRepository.findById(id)

.map(existingBook -> {

bookRepository.delete(existingBook);

return ResponseEntity.noContent().build();

})

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

}

}

**BookControllerIT.java**

package com.example.bookstoreapi.integration;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.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.autoconfigure.web.servlet.WebMvcTest;

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

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

import org.springframework.test.context.ActiveProfiles;

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

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

import org.springframework.web.context.WebApplicationContext;

import javax.sql.DataSource;

import java.util.List;

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

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

@SpringBootTest

@ActiveProfiles("test")

public class BookControllerIT {

@Autowired

private MockMvc mockMvc;

@Autowired

private BookRepository bookRepository;

@Autowired

private ObjectMapper objectMapper;

@BeforeEach

public void setUp() {

bookRepository.deleteAll();

}

@Test

public void testCreateBook() throws Exception {

Book book = new Book(null, "Integration Test Book", "Author", 12.99);

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

.contentType("application/json")

.content(objectMapper.writeValueAsString(book)))

.andExpect(status().isOk())

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

}

@Test

public void testGetAllBooks() throws Exception {

Book book = new Book(null, "Another Test Book", "Author", 15.99);

bookRepository.save(book);

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

.andExpect(status().isOk())

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

}

@Test

public void testGetBookById() throws Exception {

Book book = new Book(null, "Book to Find", "Author", 9.99);

Book savedBook = bookRepository.save(book);

mockMvc.perform(get("/api/books/" + savedBook.getId()))

.andExpect(status().isOk())

.andExpect(jsonPath("$.title").value("Book to Find"));

}

@Test

public void testUpdateBook() throws Exception {

Book book = new Book(null, "Old Title", "Author", 5.99);

Book savedBook = bookRepository.save(book);

Book updatedBook = new Book(null, "Updated Title", "Author", 10.99);

mockMvc.perform(put("/api/books/" + savedBook.getId())

.contentType("application/json")

.content(objectMapper.writeValueAsString(updatedBook)))

.andExpect(status().isOk())

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

}

@Test

public void testDeleteBook() throws Exception {

Book book = new Book(null, "Book to Delete", "Author", 7.99);

Book savedBook = bookRepository.save(book);

mockMvc.perform(delete("/api/books/" + savedBook.getId()))

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

mockMvc.perform(get("/api/books/" + savedBook.getId()))

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

}

}