SetUp Spring Test for Integration Testing

<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>

Configure Spring Boot Test

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

import org.springframework.test.context.ActiveProfiles;

@SpringBootTest

@ActiveProfiles("test")

public class BookstoreIntegrationTest {

// Test methods will go here

}

Use MockMvc for End to End Testing for REST Endpoints

Setup MockMvc

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.AutoConfigureMockMvc;

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

@SpringBootTest

@AutoConfigureMockMvc

public class BookstoreIntegrationTest {

@Autowired

private MockMvc mockMvc;

@BeforeEach

void setUp() {

// Set up necessary preconditions

}

@Test

public void testGetBookById() throws Exception {

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

.andExpect(status().isOk())

.andExpect(jsonPath("$.title", is("Test Book")));

}

// Additional test methods will go here

}

Database Integration Using an In-Memory database

Configure H2 database

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

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

spring.h2.console.enabled=true

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

Test Database Integration

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

@SpringBootTest

@AutoConfigureMockMvc

public class BookstoreIntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private BookRepository bookRepository;

@Test

public void testGetBookById() throws Exception {

// Prepare data

Book book = new Book(null, "Integration Test Book", "Author", 150);

book = bookRepository.save(book);

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

.andExpect(status().isOk())

.andExpect(jsonPath("$.title", is("Integration Test Book")))

.andExpect(jsonPath("$.author", is("Author")))

.andExpect(jsonPath("$.price", is(150)));

}

@Test

public void testCreateBook() throws Exception {

String newBookJson = "{\"title\":\"New Book\",\"author\":\"New Author\",\"price\":200}";

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

.contentType("application/json")

.content(newBookJson))

.andExpect(status().isCreated());

// Verify data in the repository

assertThat(bookRepository.findAll()).hasSize(1);

}

// Additional tests for other CRUD operations will go here

}