Spring Testing Exercises Solutions

Exercise 1: Basic Unit Test for a Service Method

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

class CalculatorServiceTest {

private final CalculatorService calculatorService = new CalculatorService();

@Test

void testAdd() {

assertEquals(5, calculatorService.add(2, 3));

assertEquals(0, calculatorService.add(0, 0));

assertEquals(-1, calculatorService.add(2, -3));

}

}

Exercise 2: Mocking a Repository in a Service Test

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.extension.ExtendWith;

@ExtendWith(MockitoExtension.class)

class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

void testGetUserById() {

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Test User");

when(userRepository.findById(1L)).thenReturn(java.util.Optional.of(mockUser));

User result = userService.getUserById(1L);

assertEquals("Test User", result.getName());

verify(userRepository).findById(1L);

}

}

Exercise 3: Testing a REST Controller with MockMvc

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.mock.mockito.MockBean;

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

import static org.mockito.Mockito.\*;

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

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

@WebMvcTest(UserController.class)

class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testGetUser() throws Exception {

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Test User");

when(userService.getUserById(1L)).thenReturn(mockUser);

mockMvc.perform(get("/users/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Test User"));

}

}

Exercise 4: Integration Test with Spring Boot

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.http.ResponseEntity;

import static org.junit.jupiter.api.Assertions.\*;

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

class UserIntegrationTest {

@Autowired

private TestRestTemplate restTemplate;

@Test

void testGetUserIntegration() {

ResponseEntity<User> response = restTemplate.getForEntity("/users/1", User.class);

assertEquals(200, response.getStatusCodeValue());

assertNotNull(response.getBody());

}

}

Exercise 5: Test Controller POST Endpoint

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.mock.mockito.MockBean;

import org.springframework.http.MediaType;

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

import com.fasterxml.jackson.databind.ObjectMapper;

import static org.mockito.Mockito.\*;

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

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

@WebMvcTest(UserController.class)

class UserControllerPostTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Autowired

private ObjectMapper objectMapper;

@Test

void testCreateUser() throws Exception {

User newUser = new User();

newUser.setName("New User");

when(userService.saveUser(any(User.class))).thenReturn(newUser);

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(newUser)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("New User"));

}

}

Exercise 6: Test Service Exception Handling

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.extension.ExtendWith;

import java.util.NoSuchElementException;

@ExtendWith(MockitoExtension.class)

class UserServiceExceptionTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

void testUserNotFound() {

when(userRepository.findById(99L)).thenReturn(java.util.Optional.empty());

assertThrows(NoSuchElementException.class, () -> {

userService.getUserById(99L);

});

}

}

Exercise 7: Test Custom Repository Query

import org.junit.jupiter.api.Test;

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

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

import static org.junit.jupiter.api.Assertions.\*;

import java.util.List;

@DataJpaTest

class UserRepositoryTest {

@Autowired

private UserRepository userRepository;

@Test

void testFindByName() {

User user1 = new User();

user1.setName("John");

userRepository.save(user1);

User user2 = new User();

user2.setName("John");

userRepository.save(user2);

List<User> users = userRepository.findByName("John");

assertEquals(2, users.size());

}

}

Exercise 8: Test Controller Exception Handling

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.mock.mockito.MockBean;

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

import static org.mockito.Mockito.\*;

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

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

@WebMvcTest

class ExceptionHandlerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testUserNotFoundHandler() throws Exception {

when(userService.getUserById(99L)).thenThrow(new NoSuchElementException());

mockMvc.perform(get("/users/99"))

.andExpect(status().isNotFound())

.andExpect(content().string("User not found"));

}

}

Exercise 9: Parameterized Test with JUnit

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;

import static org.junit.jupiter.api.Assertions.\*;

class CalculatorParameterizedTest {

private final CalculatorService calculatorService = new CalculatorService();

@ParameterizedTest

@CsvSource({

"2, 3, 5",

"0, 0, 0",

"-5, 10, 5",

"100, 200, 300"

})

void testAddWithMultipleInputs(int a, int b, int expected) {

assertEquals(expected, calculatorService.add(a, b));

}

}