PAKHI SHARMA

Spring Testing Exercises

# Exercise 1: Basic Unit Test for a Service Method

Task: Write a unit test for a service method that adds two numbers.

## Service:

@Service

public class CalculatorService { public int add(int a, int b) {

return a + b;

}

}

## Test:

Write code for this.

import org.junit.jupiter.api.Test;

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

public class CalculatorServiceTest {

private final CalculatorService calculatorService = new CalculatorService();

@Test

void testAdd() {

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

}

}

# Exercise 2: Mocking a Repository in a Service Test

Task: Test a service that uses a repository to fetch data.

## Entity:

@Entity

public class User { @Id

private Long id; private String name;

// getters and setters

}

## Repository:

public interface UserRepository extends JpaRepository<User, Long> {

}

## Service:

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

return userRepository.findById(id).orElse(null);

}

}

## Test:

Write code for this.

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)

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

void testGetUserById() {

User user = new User();

user.setId(1L);

user.setName("John");

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

User result = userService.getUserById(1L);

assertEquals("John", result.getName());

}

}

# Exercise 3: Testing a REST Controller with MockMvc

Task: Test a controller endpoint that returns a user.

## Controller:

@RestController @RequestMapping("/users") public class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) { return ResponseEntity.ok(userService.getUserById(id));

}

}

## Test:

Write code for this.

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.get;

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

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testGetUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("John");

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

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

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("John"));

}

}

# Exercise 4: Integration Test with Spring Boot

Task: Write an integration test that tests the full flow from controller to database.

## Test:

Write code for this.

import org.junit.jupiter.api.Test;

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

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

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

@SpringBootTest

public class UserIntegrationTest {

@Autowired

private UserService userService;

@Test

void testCreateAndRetrieveUser() {

User user = new User();

user.setName("Alice");

userService.saveUser(user);

User retrieved = userService.getUserById(user.getId());

assertEquals("Alice", retrieved.getName());

}

}

# Exercise 5: Test Controller POST Endpoint

Task: Test a POST endpoint that creates a user.

## Controller:

@PostMapping

public ResponseEntity<User> createUser(@RequestBody User user) { return ResponseEntity.ok(userService.saveUser(user));

}

## Test:

Write code for this.

import org.junit.jupiter.api.Test;

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

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

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

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

import com.fasterxml.jackson.databind.ObjectMapper;

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

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

import static org.mockito.Mockito.\*;

@WebMvcTest(UserController.class)

public class UserPostControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testCreateUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("Bob");

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

mockMvc.perform(post("/users")

.contentType("application/json")

.content(new ObjectMapper().writeValueAsString(user)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Bob"));

}

}

# Exercise 6: Test Service Exception Handling

Task: Test how a service handles a missing user.

## Test:

Write code for this.

import static org.mockito.Mockito.\*;

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

import java.util.Optional;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

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

@ExtendWith(MockitoExtension.class)

public class UserServiceExceptionTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

void testUserNotFound() {

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

assertNull(userService.getUserById(99L));

}

}

# Exercise 7: Test Custom Repository Query

Task: Add and test a custom query method.

## Repository:

public interface UserRepository extends JpaRepository<User, Long> { List<User> findByName(String name);

}

## Test:

Write code for this.

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import static org.mockito.Mockito.\*;

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

import java.util.List;

import org.mockito.junit.jupiter.MockitoExtension;

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

@ExtendWith(MockitoExtension.class)

public class UserRepositoryCustomQueryTest {

@Mock

private UserRepository userRepository;

@Test

void testFindByName() {

User user = new User();

user.setName("Sam");

when(userRepository.findByName("Sam")).thenReturn(List.of(user));

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

assertEquals(1, users.size());

assertEquals("Sam", users.get(0).getName());

}

}

# Exercise 8: Test Controller Exception Handling

Task: Add and test a @ControllerAdvice for handling exceptions.

## Exception Handler:

@ControllerAdvice

public class GlobalExceptionHandler { @ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) { return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found");

}

}

## Test:

Write code for this.

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 java.util.NoSuchElementException;

import static org.mockito.Mockito.\*;

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

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

@WebMvcTest(UserController.class)

public class GlobalExceptionHandlerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

void testUserNotFoundHandler() throws Exception {

when(userService.getUserById(100L)).thenThrow(new NoSuchElementException());

mockMvc.perform(get("/users/100"))

.andExpect(status().isNotFound())

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

}

}

# Exercise 9: Parameterized Test with JUnit

Task: Use @ParameterizedTest to test multiple inputs.

## Test:

Write code for this.

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class CalculatorServiceParamTest {

private final CalculatorService calculatorService = new CalculatorService();

@ParameterizedTest

@CsvSource({"2,3,5", "10,5,15", "0,0,0"})

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

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

}

}