**Exercise 1: Basic Unit Test for Service Method**

**CalculatorService.java**

import org.springframework.stereotype.Service;

@Service

public class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorServiceTest.java**

import org.junit.jupiter.api.Test;

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

public class CalculatorServiceTest {

CalculatorService calculatorService = new CalculatorService();

@Test

void testAdd() {

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

}

}

**Exercise 2: Mocking Repository in Service Test**

**User.java**

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

// Getters and Setters

}

**UserRepository.java**

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

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

import org.springframework.stereotype.Service;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

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

}

public User saveUser(User user) {

return userRepository.save(user);

}

}

**UserServiceTest.java**

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.MockitoAnnotations;

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

public UserServiceTest() {

MockitoAnnotations.openMocks(this);

}

@Test

void testGetUserById() {

User user = new User();

user.setId(1L);

user.setName("Ashish");

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

User result = userService.getUserById(1L);

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

}

}

**Exercise 3: Test Controller with MockMvc**

**UserController.java**

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@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));

}

@PostMapping

public ResponseEntity<User> createUser(@RequestBody User user) {

return ResponseEntity.ok(userService.saveUser(user));

}

}

**UserControllerTest.java**

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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 static org.mockito.Mockito.when;

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

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("Ashish");

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

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

.andExpect(status().isOk())

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

}

@Test

void testCreateUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("Ashish");

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

mockMvc.perform(post("/users")

.contentType("application/json")

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

.andExpect(status().isOk())

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

}

}

**Exercise 4: Full Integration Test**

**UserIntegrationTest.java**

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 testFullFlow() {

User user = new User();

user.setId(1L);

user.setName("Ashish");

User saved = userService.saveUser(user);

User fetched = userService.getUserById(1L);

assertEquals(saved.getName(), fetched.getName());

}

}

**Exercise 6: Service Exception Handling**

**UserServiceExceptionTest.java**

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.MockitoAnnotations;

public class UserServiceExceptionTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

public UserServiceExceptionTest() {

MockitoAnnotations.openMocks(this);

}

@Test

void testUserNotFound() {

when(userRepository.findById(2L)).thenReturn(Optional.empty());

User result = userService.getUserById(2L);

assertNull(result);

}

}

**Exercise 7: Custom Repository Query**

**Extend Repository:**

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name);

}

**Test:**

import static org.mockito.Mockito.\*;

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

import java.util.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

public class CustomQueryTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

public CustomQueryTest() {

MockitoAnnotations.openMocks(this);

}

@Test

void testFindByName() {

List<User> list = Arrays.asList(new User() {{ setId(1L); setName("Ashish"); }});

when(userRepository.findByName("Ashish")).thenReturn(list);

List<User> result = userRepository.findByName("Ashish");

assertEquals(1, result.size());

assertEquals("Ashish", result.get(0).getName());

}

}

**Exercise 8: Controller Exception Handling**

**GlobalExceptionHandler.java**

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.NoSuchElementException;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found");

}

}

Test (within controller test):

@Test

void testExceptionHandling() 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**

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class ParameterizedAddTest {

CalculatorService calculatorService = new CalculatorService();

@ParameterizedTest

@CsvSource({

"1,2,3",

"5,5,10",

"0,0,0"

})

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

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

}

}