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

Service:

@Service

public class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

**Test:**

@SpringBootTest

public class CalculatorServiceTest {

@Autowired

private CalculatorService calculatorService;

@Test

public void testAdd() {

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

}

}

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

@ExtendWith(MockitoExtension.class)

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

public void testGetUserById() {

User user = new User();

user.setId(1L);

user.setName("John");

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

User result = userService.getUserById(1L);

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

}

}

**Exercise 3: Test Controller with MockMvc**

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testGetUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("Alice");

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

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

.andExpect(status().isOk())

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

}

}

**Exercise 4: Full Integration Test**

@SpringBootTest

@AutoConfigureMockMvc

public class IntegrationTest {

@Autowired

private MockMvc mockMvc;

@Autowired

private UserRepository userRepository;

@BeforeEach

public void setup() {

userRepository.save(new User(1L, "Test User"));

}

@Test

public void testGetUserFlow() throws Exception {

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

.andExpect(status().isOk())

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

}

}

**Exercise 5: Test POST Endpoint**

@Test

public void testCreateUser() throws Exception {

User user = new User(1L, "New User");

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

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content("{\"id\":1,\"name\":\"New User\"}"))

.andExpect(status().isOk())

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

}

**Exercise 6: Test Missing User Handling**

@Test

public void testUserNotFound() {

when(userRepository.findById(1L)).thenReturn(Optional.empty());

User result = userService.getUserById(1L);

assertNull(result);

}

**Exercise 7: Test Custom Repository Query**

@Test

public void testFindByName() {

List<User> users = List.of(new User(1L, "Alice"));

when(userRepository.findByName("Alice")).thenReturn(users);

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

assertEquals(1, result.size());

}

**Exercise 8: Test Exception Handling in Controller**

@Test

public void testGlobalExceptionHandler() throws Exception {

when(userService.getUserById(1L)).thenThrow(new NoSuchElementException());

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

.andExpect(status().isNotFound())

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

}

**Exercise 9: Parameterized Test**

@ParameterizedTest

@CsvSource({

"2, 3, 5",

"0, 0, 0",

"-1, -1, -2"

})

public void testAddMultiple(int a, int b, int expected) {

CalculatorService service = new CalculatorService();

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

}