**Mocking Dependencies in Spring Tests using Mockito**

**Exercise 1: Mocking a Service Dependency in a Controller Test**

Task: Write a unit test for a Spring controller that uses a service to fetch data.

Mock the service dependency using Mockito.

1. Create the User Entity

2. Create the UserService

3. Create the UserController

**CODE:**

package com.example.junitdemo;

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); // Initializes mocks

}

@Test

public void testGetUserById() {

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

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

User result = userService.getUserById(1L);

assertNotNull(result);

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

}

}

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

Task: Write a unit test for a Spring service that uses a repository to fetch data.

Mock the repository dependency using Mockito.

1. Create the User Entity

2. Create the UserRepository

3. Create the UserService

4. Create the UserServiceTest

**CODE**

package com.example.junitdemo;

import static org.mockito.Mockito.\*;

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

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

*@WebMvcTest*(UserController.class)

public class UserControllerTest {

*@Autowired*

private MockMvc mockMvc;

*@MockBean*

private UserService userService;

*@Test*

public void testGetUser() throws Exception {

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

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

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

.andExpect(*status*().isOk())

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

}

}

**Exercise 3: Mocking a Service Dependency in an Integration Test**

Task:Write an integration test for a Spring Boot application that mocks a service dependency using Mockito

1. Create the User Entity

2. Create the UserService

3. Create the UserController

4. Create the UserIntegrationTest

**CODE:**

package com.example.junitdemo;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

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

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

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

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

import org.springframework.test.web.servlet.\*;

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

*@SpringBootTest*

*@AutoConfigureMockMvc*

public class UserIntegrationTest {

*@Autowired*

private MockMvc mockMvc;

*@MockBean*

private UserService userService;

*@Test*

public void testIntegrationGetUser() throws Exception {

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

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

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

.andExpect(*status*().isOk())

.andExpect(*jsonPath*("$.name").value("Jane"));

}

}

