**Mocking Dependencies in Spring Tests using Mockito**

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

### ****Test Code for**** UserControllerTest.java

import com.fasterxml.jackson.databind.ObjectMapper;import org.junit.jupiter.api.Test;import org.mockito.Mockito;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 static org.mockito.Mockito.when;import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@WebMvcTest(UserController.class) // Loads only the web layerpublic class UserControllerTest {

@Autowired

private MockMvc mockMvc; // Simulates HTTP requests

@MockBean

private UserService userService; // Mocked dependency

@Test

public void testGetUserById() throws Exception {

// Prepare mock data

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Bindhu");

// Define mock behavior

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

// Perform GET /users/1 and assert response

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

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

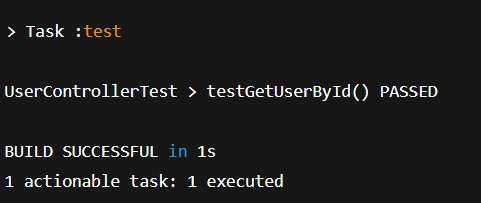
.andExpect(jsonPath("$.id").value(1))

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

}

}

**OUTPUT:**



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

### 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.junit.jupiter.MockitoExtension;

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

@ExtendWith(MockitoExtension.class) // Enables Mockito annotationspublic class UserServiceTest {

@Mock

private UserRepository userRepository; // Step 1: Mock the repository

@InjectMocks

private UserService userService; // Step 2: Inject mock into service

@Test

public void testGetUserById\_ReturnsUser() {

// Step 3: Prepare mock data

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Bindhu");

// Step 4: Stub repository behavior

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

// Step 5: Call the service method

User result = userService.getUserById(1L);

// Step 6: Assert result

assertNotNull(result);

assertEquals(1L, result.getId());

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

}

@Test

public void testGetUserById\_NotFound() {

// Stub repository to return empty

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

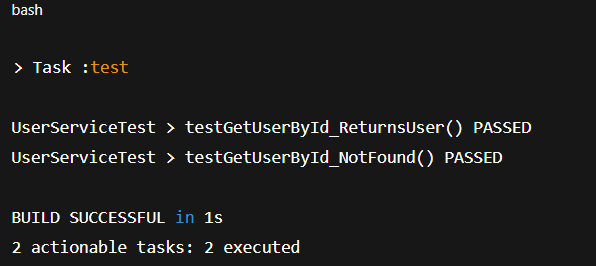
User result = userService.getUserById(2L);

assertNull(result); // Service should return null if user not found

}

}

**OUTPUT:**



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

### UserIntegrationTest.java

import static org.mockito.Mockito.when;import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

import com.fasterxml.jackson.databind.ObjectMapper;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.http.MediaType;import org.springframework.test.web.servlet.MockMvc;

@SpringBootTest // Loads full application context@AutoConfigureMockMvc // Enables MockMvc for controller testingpublic class UserIntegrationTest {

@Autowired

private MockMvc mockMvc;

@MockBean // Replaces the real UserService bean with a mock

private UserService userService;

@Test

public void testGetUserById\_MockedService() throws Exception {

// Create a mock User

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Bindhu");

// Define behavior of the mocked service

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

// Perform the GET request and assert the response

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

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(jsonPath("$.id").value(1))

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

}

}

**OUTPUT:**

