**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.**

**CalculatorSeviceTest.java**

import org.junit.jupiter.api.Test;

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

public class CalculatorServiceTest {

    @Test

    public void testAdd() {

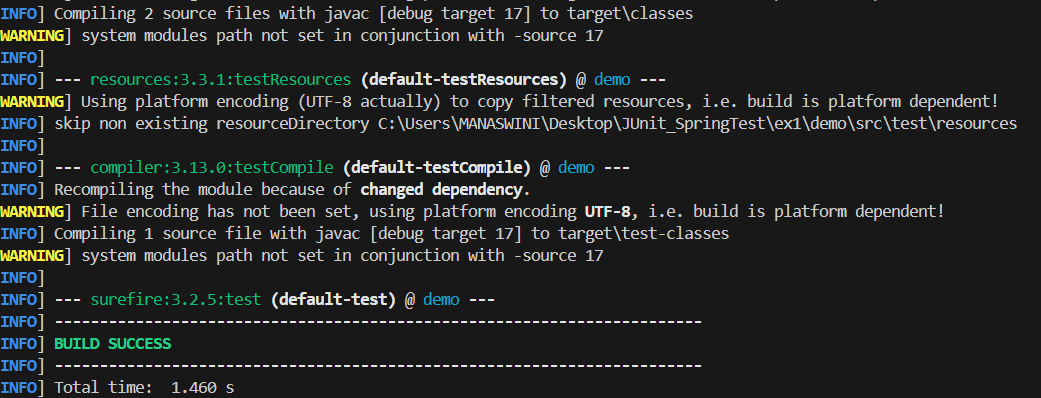
        CalculatorService cs = new CalculatorService();

        int ans = cs.add(10, 5);

        assertEquals(15, ans);

    }

}



**Exercise 2: Mocking a Repository in a Service Test Task: Test a service that uses a repository to fetch data.**

**UserServiceTest.java**

import static org.mockito.Mockito.\*;

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

import java.util.Optional;

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

@ExtendWith(MockitoExtension.class)

public class UserServiceTest {

    @Mock

    private UserRepository ur;

    @InjectMocks

    private UserService us;

    @Test

    public void testGetUserById() {

        User user = new User();

        user.setId(1L);

        user.setName("Yashaswini");

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

        User res = us.getUserById(1L);

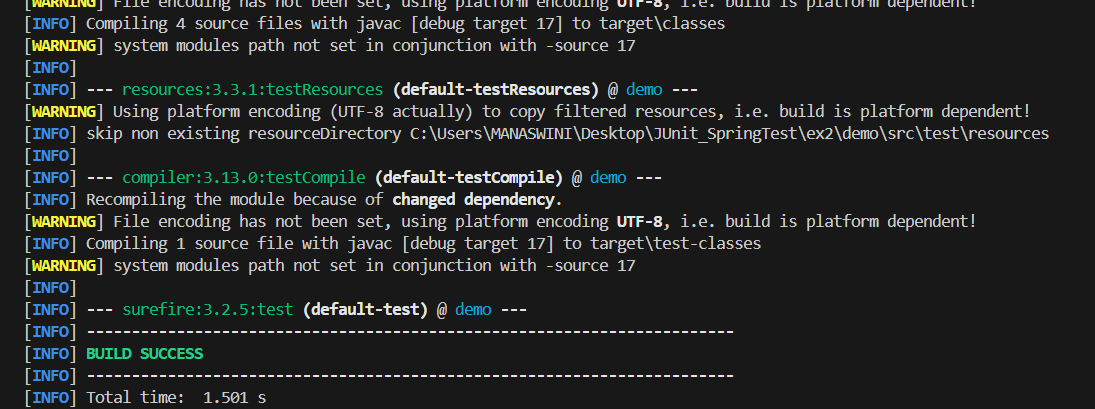
        assertNotNull(res);

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

        assertEquals("Yashaswini", res.getName());

    }

}



**Exercise 3: Testing a REST Controller with MockMvc**

**Task: Test a controller endpoint that returns a user.**

**UserController.java**

import com.example.demo.model.User;

import com.example.demo.service.UserService;

import com.example.demo.controller.UserController;

import org.springframework.http.MediaType;

@WebMvcTest(UserController.class)

public class UserControllerTest {

    @Autowired

    private MockMvc mockMvc;

    @MockBean

    private UserService userService;

    @Test

    void testGetUserById() throws Exception {

        User user = new User();

        user.setId(1L);

        user.setName("Yashaswini");

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

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

                .accept(MediaType.APPLICATION\_JSON))

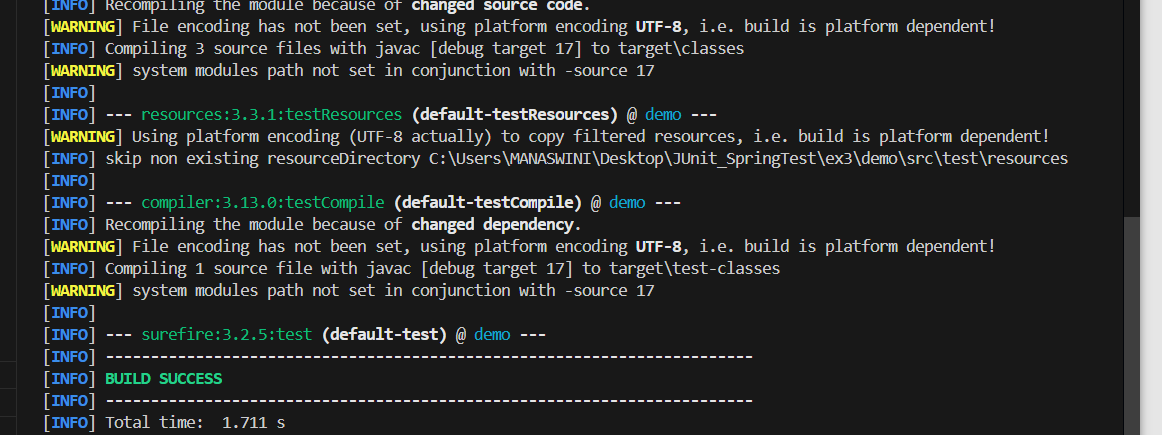
                .andExpect(status().isOk())

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

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

    }

}



**Exercise 5: Test Controller POST Endpoint Task: Test a POST endpoint that creates a user.**

**UserControllerPostTest.java**

import com.example.demo.controller.UserController;

import com.example.demo.model.User;

import com.example.demo.service.UserService;

@WebMvcTest(UserController.class)

public class UserControllerPostTest {

    @Autowired

    private MockMvc mockMvc;

    @Autowired

    private ObjectMapper objectMapper;

    @MockBean

    private UserService userService;

    @Test

    public void testCreateUser() throws Exception {

        User inputUser = new User();

        inputUser.setId(1L);

        inputUser.setName("Yashaswini");

        when(userService.saveUser(inputUser)).thenReturn(inputUser);

        mockMvc.perform(post("/users")

                .contentType(MediaType.APPLICATION\_JSON)

                .content(objectMapper.writeValueAsString(inputUser)))

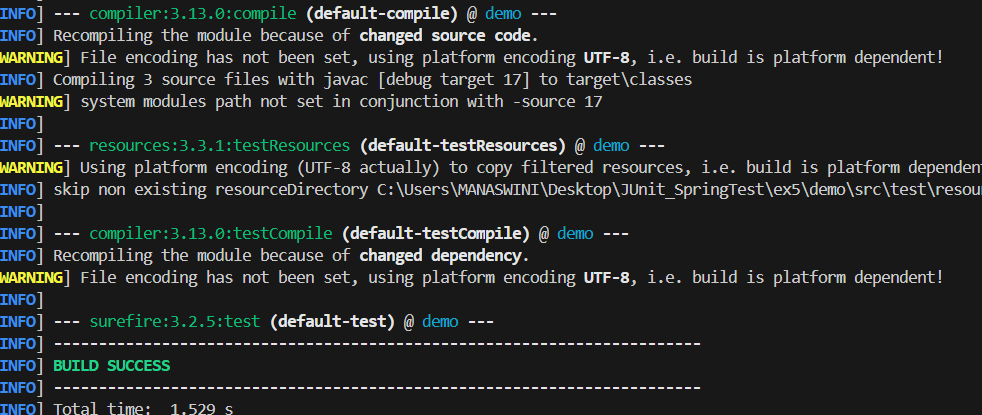
                .andExpect(status().isOk())

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

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

    }

}



**Exercise 6: Test Service Exception Handling Task: Test how a service handles a missing user.**

**UserServiceExceptionTest.java**

import static org.mockito.Mockito.\*;

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

import java.util.Optional;

@ExtendWith(MockitoExtension.class)

public class UserServiceExceptionTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@Test

void testUserNotFoundException() {

Long missingId = 100L;

when(userRepository.findById(missingId)).thenReturn(Optional.empty());

UserNotFoundException exception = assertThrows(

UserNotFoundException.class,

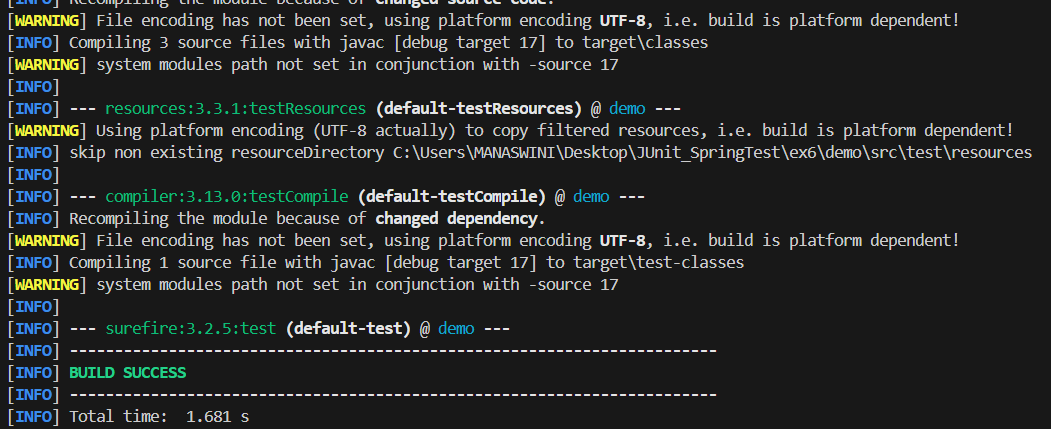
() -> userService.getUserById(missingId)

);

assertEquals("No user with id 100", exception.getMessage());

}

}



**Exercise 7: Test Custom Repository Query Task: Add and test a custom query method.**

UserRepositoryTest.java

import com.example.demo.model.User;

import com.example.demo.repository.UserRepository;

import java.util.List;

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

@DataJpaTest

public class UserRepositoryTest {

    @Autowired

    private UserRepository userRepository;

    @Test

    public void testFindByName() {

        User user1 = new User();

        user1.setId(1L);

        user1.setName("Charlie");

        User user2 = new User();

        user2.setId(2L);

        user2.setName("Eva");

        User user3 = new User();

        user3.setId(3L);

        user3.setName("Charlie");

        userRepository.save(user1);

        userRepository.save(user2);

        userRepository.save(user3);

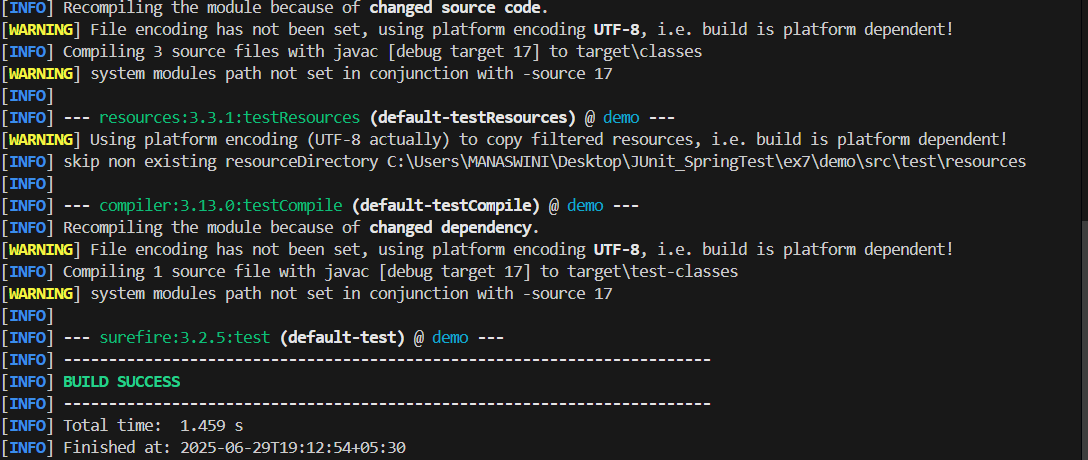
        List<User> a= userRepository.findByName("Alice");

    assertEquals(2, a.size());

        assertTrue(a.stream().allMatch(u -> u.getName().equals("Alice")));

    }

}



**Exercise 8: Test Controller Exception Handling Task: Add and test a @ControllerAdvice for handling exceptions.**

**UserControllerTest.java**

import com.example.demo.controller.UserController;

import com.example.demo.exception.GlobalExceptionHandler;

import com.example.demo.model.User;

@WebMvcTest(controllers = UserController.class)

public class UserControllerExceptionTest {

    @Autowired

    private MockMvc m;

    @MockBean

    private UserService us;

    @Test

    void testUserNotFoundHandledByControllerAdvice() throws Exception {

        Long missingId = 99L;

        when(us.getUserById(missingId)).thenReturn(Optional.empty());

m.perform(get("/users/99")

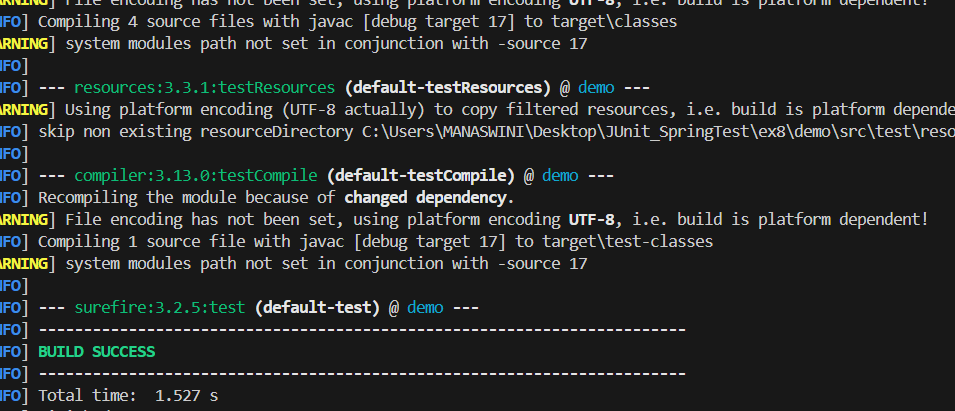
                .accept(MediaType.APPLICATION\_JSON))

                .andExpect(status().isNotFound())

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

    }

}



**Exercise 9: Parameterized Test with JUnit Task: Use @ParameterizedTest to test multiple inputs.**

**EvenChecker.java**

public class EvenChecker {

    public boolean isEven(int number) {

        return number % 2 == 0;

    }

}

**EvenCheckerTest.java**

import org.junit.jupiter.params.ParameterizedTest;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class EvenCheckerTest {

private final EvenChecker checker = new EvenChecker();

@ParameterizedTest

@ValueSource(ints = {12,14,16,18,28})

public void testIsEven(int input) {

assertTrue(checker.isEven(input), input + " should be even");

}

}

