SPRING TESTING EXERCISE HANDS-ON

Exercise 1: Basic Unit Test for a Service Method

Task: Write a unit test for a service method that adds two numbers.

Code:

CalculatorService.java

**package** TestSuit;

**public** **class** CalculatorService {

**public** **int** add(**int** a, **int** b) {

**return** a + b;

}

}

CalculatorServiceTest.java

**package** TestSuit;

**import** org.junit.jupiter.api.Test;

**import** org.junit.jupiter.api.BeforeEach;

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

**class** CalculatorServiceTest {

**private** CalculatorService calculatorService;

@BeforeEach

**void** setUp() {

calculatorService = **new** CalculatorService();

}

@Test

**void** testAdd\_PositiveNumbers() {

// Given

**int** a = 5;

**int** b = 3;

// When

**int** result = calculatorService.add(a, b);

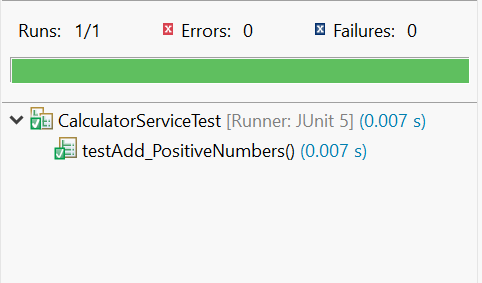
// Then

*assertEquals*(8, result);

}

}

Output:



Exercise 2: Mocking a Repository in a Service Test

Task: Test a service that uses a repository to fetch data.

Code :

Entity:

@Entity

**public** **class** User {

@Id

**private** Long id;

**private** String name;

// getters and setters

}

Repository:

**public** **interface** UserRepository **extends** JpaRepository<User, Long> {

}

Service:

@Service

**public** **class** UserService {

@Autowired

**private** UserRepository userRepository;

**public** User getUserById(Long id) {

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

}

}

}

UserServiceTest.java

**import** org.junit.jupiter.api.Test;

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

**import** org.mockito.InjectMocks;

**import** org.mockito.Mock;

**import** org.mockito.junit.jupiter.MockitoExtension;

**import** java.util.Optional;

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

**import** **static** org.mockito.Mockito.\*;

@ExtendWith(MockitoExtension.**class**)

**class** UserServiceTest {

@Mock

**private** UserRepository userRepository;

@InjectMocks

**private** UserService userService;

@Test

**void** testGetUserById\_UserExists() {

// Arrange

Long userId = 1L;

User mockUser = **new** User();

mockUser.setId(userId);

mockUser.setName("John Doe");

*when*(userRepository.findById(userId)).thenReturn(Optional.*of*(mockUser));

// Act

User result = userService.getUserById(userId);

// Assert

*assertNotNull*(result);

*assertEquals*(userId, result.getId());

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

*verify*(userRepository, *times*(1)).findById(userId);

}

}

Exercise 3: Testing a REST Controller with MockMvc

Task: Test a controller endpoint that returns a user.

Code:

Controller:

@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));

}

}

UserControllerTest.java

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** org.junit.jupiter.api.Test;

**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.\*;

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

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

@WebMvcTest(UserController.**class**)

**class** UserControllerTest {

@Autowired

**private** MockMvc mockMvc;

@MockBean

**private** UserService userService;

@Autowired

**private** ObjectMapper objectMapper;

@Test

**void** testGetUser\_UserExists() **throws** Exception {

// Arrange

Long userId = 1L;

User mockUser = **new** User();

mockUser.setId(userId);

mockUser.setName("John Doe");

*when*(userService.getUserById(userId)).thenReturn(mockUser);

// Act & Assert

mockMvc.perform(get("/users/{id}", userId)

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

.andExpect(content().contentType(MediaType.APPLICATION\_JSON))

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

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

*verify*(userService, *times*(1)).getUserById(userId);

}

}

Exercise 4: Integration Test with Spring Boot

Task: Write an integration test that tests the full flow from controller to database.

Code :

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** org.junit.jupiter.api.Test;

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

**import** org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureWebMvc;

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

**import** org.springframework.boot.test.web.client.TestRestTemplate;

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.test.context.ActiveProfiles;

**import** org.springframework.test.context.junit.jupiter.SpringJUnitConfig;

**import** org.springframework.transaction.annotation.Transactional;

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

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)

@ActiveProfiles("test")

@Transactional

**class** UserIntegrationTest {

@Autowired

**private** TestRestTemplate restTemplate;

@Autowired

**private** UserRepository userRepository;

@Test

**void** testGetUser\_FullFlow\_UserExists() {

// Arrange - Insert test data directly into database

User testUser = **new** User();

testUser.setName("Integration Test User");

User savedUser = userRepository.save(testUser);

// Act - Make HTTP request to controller

ResponseEntity<User> response = restTemplate.getForEntity(

"/users/" + savedUser.getId(),

User.**class**

);

// Assert - Verify full flow

*assertEquals*(HttpStatus.OK, response.getStatusCode());

*assertNotNull*(response.getBody());

*assertEquals*(savedUser.getId(), response.getBody().getId());

*assertEquals*("Integration Test User", response.getBody().getName());

}

}

Exercise 5: Test Controller POST Endpoint

Task: Test a POST endpoint that creates a user.

Code :

Controller:

@PostMapping

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

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

}

UserControllerPost.test

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** org.junit.jupiter.api.Test;

**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.ArgumentMatchers.*any*;

**import** **static** org.mockito.Mockito.\*;

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

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

@WebMvcTest(UserController.**class**)

**class** UserControllerPostTest {

@Autowired

**private** MockMvc mockMvc;

@MockBean

**private** UserService userService;

@Autowired

**private** ObjectMapper objectMapper;

@Test

**void** testCreateUser\_ValidUser() **throws** Exception {

// Arrange

User inputUser = **new** User();

inputUser.setName("John Doe");

User savedUser = **new** User();

savedUser.setId(1L);

savedUser.setName("John Doe");

*when*(userService.saveUser(*any*(User.**class**))).thenReturn(savedUser);

// Act & Assert

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(objectMapper.writeValueAsString(inputUser)))

.andExpect(status().isOk())

.andExpect(content().contentType(MediaType.APPLICATION\_JSON))

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

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

*verify*(userService, *times*(1)).saveUser(*any*(User.**class**));

}

}

Exercise 6: Test Service Exception Handling

Task: Test how a service handles a missing user.

Code :

UserServicExceptionTest.java

**import** org.junit.jupiter.api.Test;

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

**import** org.mockito.InjectMocks;

**import** org.mockito.Mock;

**import** org.mockito.junit.jupiter.MockitoExtension;

**import** java.util.Optional;

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

**import** **static** org.mockito.Mockito.\*;

@ExtendWith(MockitoExtension.**class**)

**class** UserServiceExceptionTest {

@Mock

**private** UserRepository userRepository;

@InjectMocks

**private** UserService userService;

@Test

**void** testGetUserById\_UserNotFound\_ReturnsNull() {

// Arrange

Long userId = 999L;

*when*(userRepository.findById(userId)).thenReturn(Optional.*empty*());

// Act

User result = userService.getUserById(userId);

// Assert

*assertNull*(result);

*verify*(userRepository, *times*(1)).findById(userId);

}

}

Exercise 7: Test Custom Repository Query

Task: Add and test a custom query method.

Code :

Repository:

public interface UserRepository extends JpaRepository

{

List findByName(String name);

}

UserRepostioryCustomerQueryTest.java

**import** org.junit.jupiter.api.Test;

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

**import** org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;

**import** org.springframework.boot.test.autoconfigure.orm.jpa.TestEntityManager;

**import** org.springframework.test.context.ActiveProfiles;

**import** java.util.List;

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

@DataJpaTest

@ActiveProfiles("test")

**class** UserRepositoryCustomQueryTest {

@Autowired

**private** TestEntityManager entityManager;

@Autowired

**private** UserRepository userRepository;

@Test

**void** testFindByName\_SingleUserFound() {

// Arrange

User user = **new** User();

user.setName("John Doe");

entityManager.persistAndFlush(user);

// Act

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

// Assert

*assertNotNull*(result);

*assertEquals*(1, result.size());

*assertEquals*("John Doe", result.get(0).getName());

}

}

Exercise 8: Test Controller Exception Handling

Task: Add and test a @ControllerAdvice for handling exceptions.

Code :

@ControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler(NoSuchElementException.**class**)

**public** ResponseEntity<String> handleNotFound(NoSuchElementException ex) {

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

}

}

GlobalExceptionHandlerTest.java

**import** org.junit.jupiter.api.Test;

**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** java.util.NoSuchElementException;

**import** **static** org.mockito.Mockito.\*;

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

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

@WebMvcTest({UserController.**class**, GlobalExceptionHandler.**class**})

**class** GlobalExceptionHandlerTest {

@Autowired

**private** MockMvc mockMvc;

@MockBean

**private** UserService userService;

@Test

**void** testHandleNotFound\_UserNotFound() **throws** Exception {

// Arrange

Long userId = 999L;

*when*(userService.getUserById(userId)).thenThrow(**new** NoSuchElementException("User not found"));

// Act & Assert

mockMvc.perform(get("/users/{id}", userId)

.contentType(MediaType.APPLICATION\_JSON))

.andExpect(status().isNotFound())

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

*verify*(userService, *times*(1)).getUserById(userId);

}

}

Exercise 9: Parameterized Test with JUnit

Task: Use @ParameterizedTest to test multiple inputs.

Code :

UserServiceParamiterizedTest.java

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

**import** org.junit.jupiter.params.ParameterizedTest;

**import** org.junit.jupiter.params.provider.\*;

**import** org.mockito.InjectMocks;

**import** org.mockito.Mock;

**import** org.mockito.junit.jupiter.MockitoExtension;

**import** java.util.Optional;

**import** java.util.stream.Stream;

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

**import** **static** org.mockito.Mockito.\*;

@ExtendWith(MockitoExtension.**class**)

**class** UserServiceParameterizedTest {

@Mock

**private** UserRepository userRepository;

@InjectMocks

**private** UserService userService;

@ParameterizedTest

@ValueSource(longs = {1L, 2L, 100L, 999L, 5000L})

**void** testGetUserById\_MultipleValidIds(Long userId) {

// Arrange

User mockUser = **new** User();

mockUser.setId(userId);

mockUser.setName("Test User " + userId);

*when*(userRepository.findById(userId)).thenReturn(Optional.*of*(mockUser));

// Act

User result = userService.getUserById(userId);

// Assert

*assertNotNull*(result);

*assertEquals*(userId, result.getId());

*assertEquals*("Test User " + userId, result.getName());

*verify*(userRepository, *times*(1)).findById(userId);

}

}