**JUnit Testing Exercises**

Calculator.java

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

App.java

package com.example;

import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class AppTest

{

/\*\*

\* Rigorous Test :-)

\*/

@Test

public void shouldAnswerWithTrue()

{

assertTrue( true );

}

}

CalculatorTest.java

package com.example;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class CalculatorTest {

@Test

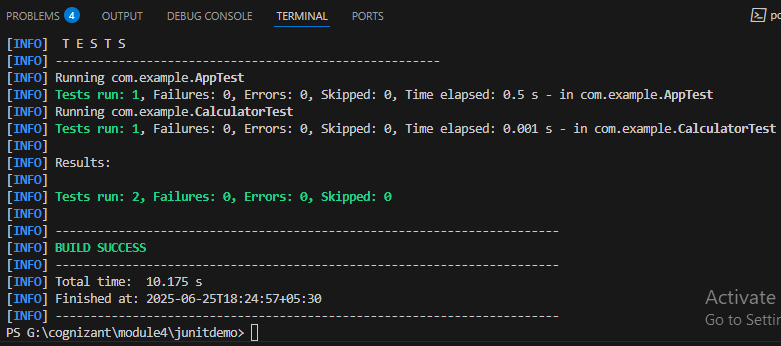
public void testAdd() {

Calculator calc = new Calculator();

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

}

}



**Advanced JUnit Testing Exercises**

**Exercise 1: Parameterized Tests**

package com.example.junitexercises;

public class EvenChecker {

public static boolean isEven(int number) {

return number % 2 == 0;

}

}

package com.example.junitexercises;

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

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

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

public class EvenCheckerTest {

@ParameterizedTest

@ValueSource(ints = {0, 2, 4, -6, 100})

void testEvenNumbers(int number) {

assertTrue(EvenChecker.isEven(number), number + " should be even");

}

@ParameterizedTest

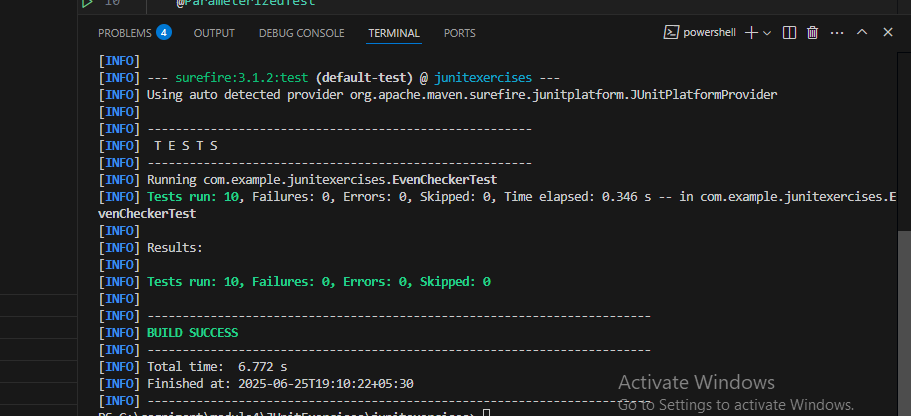
@ValueSource(ints = {1, 3, -5, 7, 99})

void testOddNumbers(int number) {

assertFalse(EvenChecker.isEven(number), number + " should be odd");

}

}



**Exercise 2: Test Suites and Categories**

package com.example.junitexercises;

public class EvenChecker {

public static boolean isEven(int number) {

return number % 2 == 0;

}

}

package com.example.junitexercises;

public class MathUtils {

public static int square(int number) {

return number \* number;

}

}

package com.example.junitexercises;

import org.junit.platform.suite.api.SelectClasses;

import org.junit.platform.suite.api.Suite;

@Suite

@SelectClasses({

EvenCheckerTest.class,

MathUtilsTest.class

})

public class AllTests {

// no methods needed

}

package com.example.junitexercises;

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

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

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

public class EvenCheckerTest {

@ParameterizedTest

@ValueSource(ints = {0, 2, 4, -6, 100})

void testEvenNumbers(int number) {

assertTrue(EvenChecker.isEven(number), number + " should be even");

}

@ParameterizedTest

@ValueSource(ints = {1, 3, -5, 7, 99})

void testOddNumbers(int number) {

assertFalse(EvenChecker.isEven(number), number + " should be odd");

}

}

package com.example.junitexercises;

import org.junit.jupiter.api.Test;

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

public class MathUtilsTest {

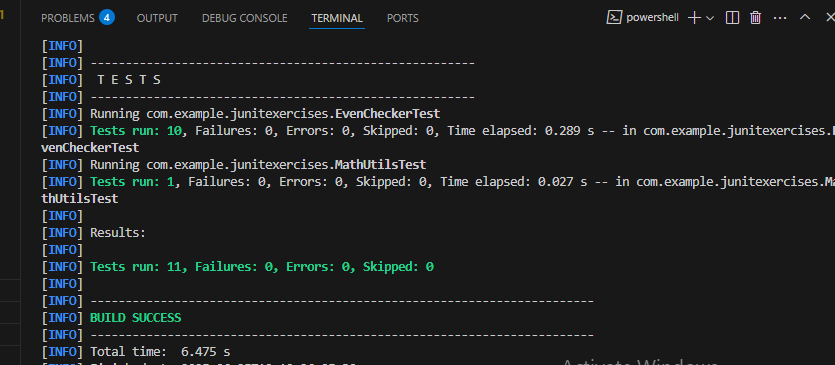
@Test

public void testSquare() {

assertEquals(25, MathUtils.square(5));

}

}



**Exercise 3: Test Execution Order**

package com.example.junitexercises;

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

import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;

import org.junit.jupiter.api.Order;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestMethodOrder;

@TestMethodOrder(OrderAnnotation.class)

public class OrderedTests {

@Test

@Order(1)

void testA() {

System.out.println("Running testA");

assertEquals(2, 1 + 1);

}

@Test

@Order(3)

void testC() {

System.out.println("Running testC");

assertEquals(6, 2 \* 3);

}

@Test

@Order(2)

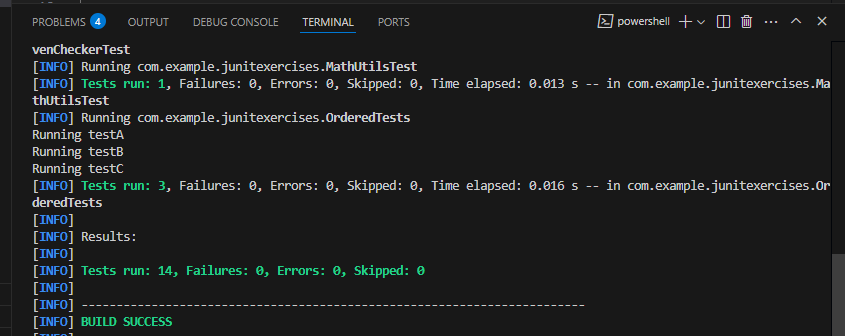
void testB() {

System.out.println("Running testB");

assertEquals(4, 2 + 2);

}

}



**Exercise 4: Exception Testing**

package com.example.junitexercises;

public class ExceptionThrower {

public static void throwException(boolean shouldThrow) {

if (shouldThrow) {

throw new IllegalArgumentException("Invalid input!");

}

}

}

package com.example.junitexercises;

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

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

import org.junit.jupiter.api.Test;

public class ExceptionThrowerTest {

@Test

void testExceptionIsThrown() {

assertThrows(IllegalArgumentException.class, () -> {

ExceptionThrower.throwException(true);

});

}

@Test

void testNoExceptionThrown() {

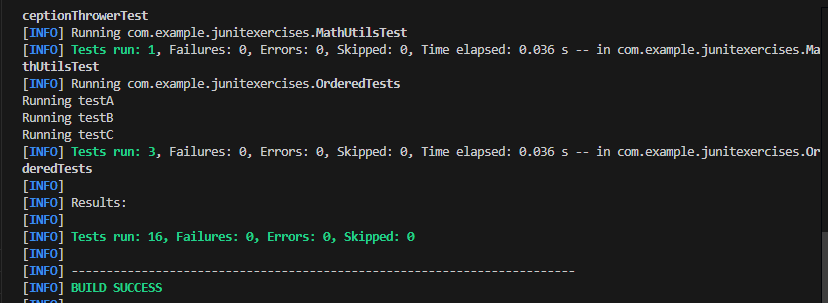
assertDoesNotThrow(() -> {

ExceptionThrower.throwException(false);

});

}

}



**Exercise 5: Timeout and Performance Testing**

package com.example.junitexercises;

public class PerformanceTester {

public void performTask() {

// Simulate a task that takes some time (e.g., 200ms)

try {

Thread.sleep(200); // Task taking 200ms

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

package com.example.junitexercises;

import java.time.Duration;

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

import org.junit.jupiter.api.Test;

public class PerformanceTesterTest {

@Test

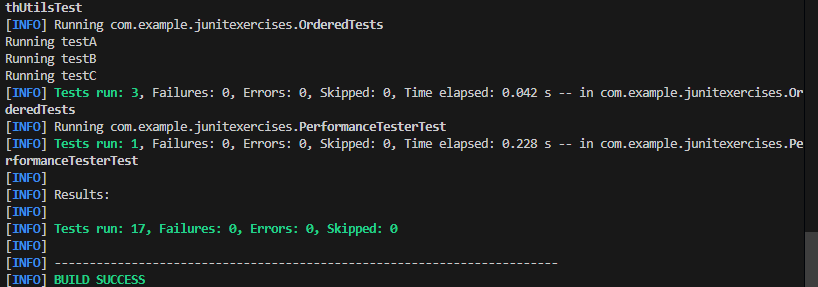
public void testPerformTaskWithinTimeout() {

PerformanceTester tester = new PerformanceTester();

assertTimeout(Duration.ofMillis(500), tester::performTask);

}

}



**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

package com.example;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

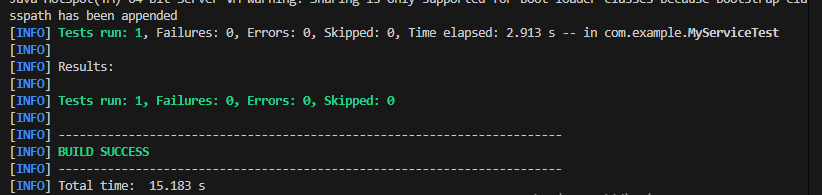
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}



**Exercise 2: Verifying Interactions**

package com.example;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.verify;

import static org.mockito.Mockito.when;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

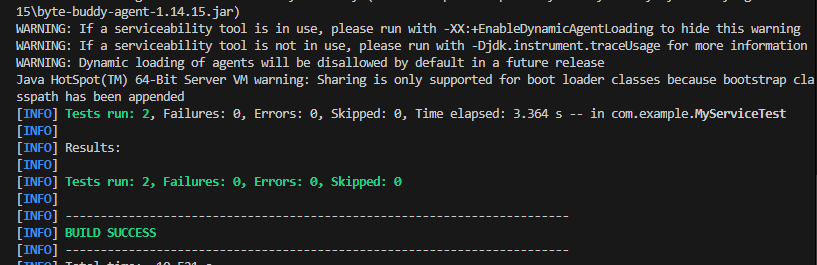
MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}



**Exercise 3: Argument Matching**

package com.example;

import org.mockito.ArgumentMatchers;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.ArgumentMatchers.anyString;

import static org.mockito.ArgumentMatchers.eq;

import static org.mockito.Mockito.verify;

import static org.mockito.Mockito.when;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

@Test

public void testArgumentMatching() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

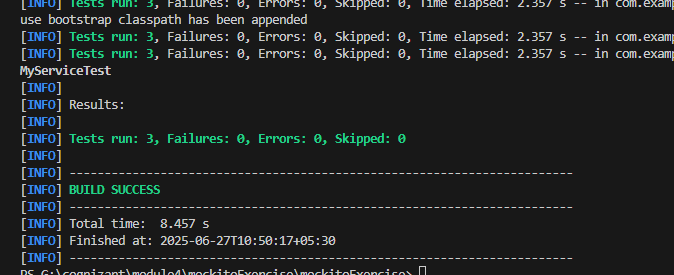
service.processAndSend("hello");

verify(mockApi).sendData(eq("HELLO"));

verify(mockApi).sendData(anyString());

}

}



**Exercise 4: Handling Void Methods**

@Test

public void testVoidMethodInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

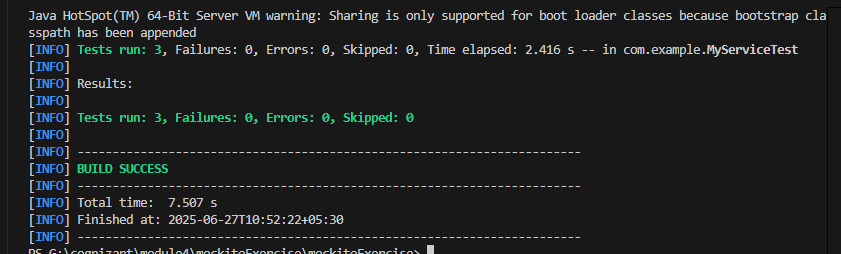
MyService service = new MyService(mockApi);

doNothing().when(mockApi).sendData("HELLO");

service.processAndSend("hello");

verify(mockApi).sendData("HELLO");

}



**Exercise 5: Mocking and Stubbing with Multiple Returns**

@Test

public void testMultipleReturnValues() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

when(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call")

.thenReturn("Third Call");

MyService service = new MyService(mockApi);

String result1 = service.fetchData();

String result2 = service.fetchData();

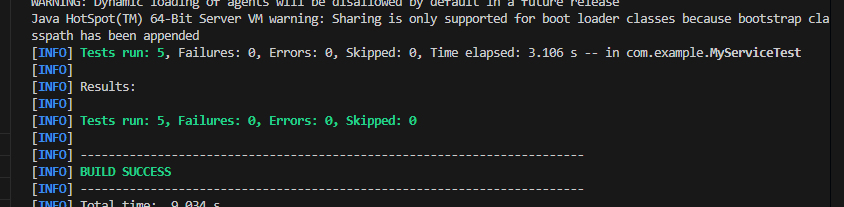
String result3 = service.fetchData();

assertEquals("First Call", result1);

assertEquals("Second Call", result2);

assertEquals("Third Call", result3);

}



**Exercise 6: Verifying Interaction Order**

@Test

public void testVoidMethodThrowsException() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

doThrow(new RuntimeException("Test Exception"))

.when(mockApi).sendData("ERROR");

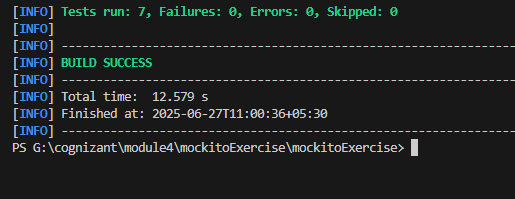
assertThrows(RuntimeException.class, () -> {

service.processAndSend("error");

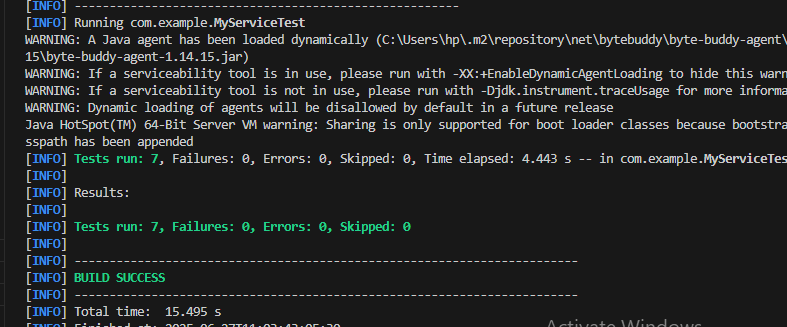
});

verify(mockApi).sendData("ERROR");

}



**Exercise 7: Handling Void Methods with Exceptions**



**Spring Testing Exercises**

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

package com.example.testingdemo.service;

import org.springframework.stereotype.Service;

@Service

public class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

package com.example.testingdemo.service;

import org.junit.jupiter.api.Test;

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

public class CalculatorServiceTest {

CalculatorService calculatorService = new CalculatorService();

@Test

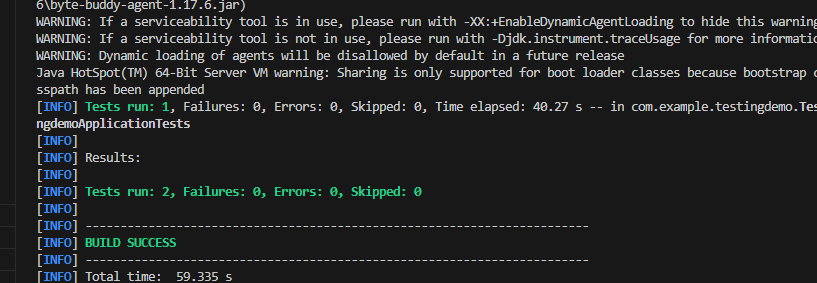
public void testAdd() {

int result = calculatorService.add(10, 5);

assertEquals(15, result);

}

}



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

package com.example.testingdemo.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id;

private String name;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

package com.example.testingdemo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.testingdemo.model.User;

public interface UserRepository extends JpaRepository<User, Long> {

}

package com.example.testingdemo.service;

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

import org.springframework.stereotype.Service;

import com.example.testingdemo.model.User;

import com.example.testingdemo.repository.UserRepository;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

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

}

}

package com.example.testingdemo.service;

import com.example.testingdemo.model.User;

import com.example.testingdemo.repository.UserRepository;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import java.util.Optional;

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

import static org.mockito.Mockito.when;

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

public UserServiceTest() {

MockitoAnnotations.openMocks(this);

}

@Test

public void testGetUserById() {

User mockUser = new User();

mockUser.setId(1L);

mockUser.setName("John");

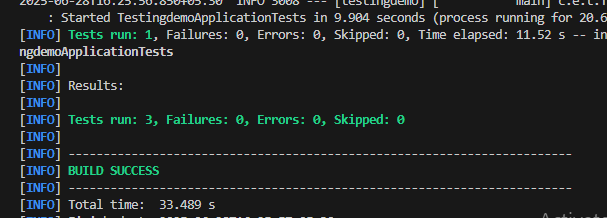
when(userRepository.findById(1L)).thenReturn(Optional.of(mockUser));

User result = userService.getUserById(1L);

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

}

}



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

package com.example.testingdemo.controller;

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.example.testingdemo.model.User;

import com.example.testingdemo.service.UserService;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

User user = userService.getUserById(id);

return ResponseEntity.ok(user);

}

}

package com.example.testingdemo.controller;

import com.example.testingdemo.model.User;

import com.example.testingdemo.service.UserService;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.when;

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

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 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 mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Alice");

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

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

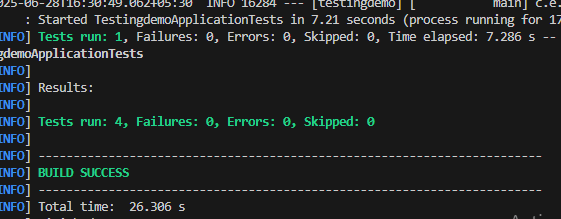
.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

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

}

}



E**xercise 4: Integration Test with Spring Boot**

**spring.application.name=testingdemo**

**spring.datasource.url=jdbc:h2:mem:testdb**

**spring.datasource.driverClassName=org.h2.Driver**

**spring.datasource.username=sa**

**spring.datasource.password=**

**spring.jpa.database-platform=org.hibernate.dialect.H2Dialect**

**spring.h2.console.enabled=true**

**spring.jpa.hibernate.ddl-auto=update**

**package com.example.testingdemo;**

**import com.example.testingdemo.model.User;**

**import com.example.testingdemo.repository.UserRepository;**

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

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

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

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

**import org.springframework.boot.test.web.server.LocalServerPort;**

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

**import org.springframework.web.client.RestTemplate;**

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

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

**public class IntegrationTest {**

**@LocalServerPort**

**private int port;**

**@Autowired**

**private UserRepository userRepository;**

**private String baseUrl;**

**private RestTemplate restTemplate;**

**@BeforeEach**

**public void setUp() {**

**baseUrl = "http://localhost:" + port + "/users";**

**restTemplate = new RestTemplate();**

**// Insert a user into the H2 DB before test**

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

**user.setId(100L);**

**user.setName("Integration User");**

**userRepository.save(user);**

**}**

**@Test**

**public void testGetUserIntegration() {**

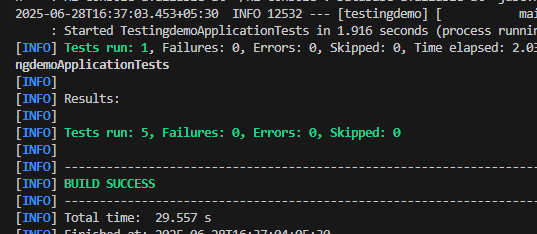
**ResponseEntity<User> response = restTemplate.getForEntity(baseUrl + "/100", User.class);**

**assertEquals(200, response.getStatusCodeValue());**

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

**}**

**}**



**Exercise 5: Test Controller POST Endpoint**

package com.example.testingdemo.controller;

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.example.testingdemo.model.User;

import com.example.testingdemo.service.UserService;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService userService;

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

User user = userService.getUserById(id);

return ResponseEntity.ok(user);

}

@PostMapping

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

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

}

}

package com.example.testingdemo.service;

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

import org.springframework.stereotype.Service;

import com.example.testingdemo.model.User;

import com.example.testingdemo.repository.UserRepository;

@Service

public class UserService {

@Autowired

private UserRepository userRepository;

public User getUserById(Long id) {

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

}

public User saveUser(User user) {

return userRepository.save(user);

}

}

package com.example.testingdemo.controller;

import com.example.testingdemo.model.User;

import com.example.testingdemo.service.UserService;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.when;

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

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 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 mockUser = new User();

mockUser.setId(1L);

mockUser.setName("Alice");

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

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

.accept(MediaType.APPLICATION\_JSON))

.andExpect(status().isOk())

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

}

@Test

public void testCreateUser() throws Exception {

User mockUser = new User();

mockUser.setId(2L);

mockUser.setName("Bob");

when(userService.saveUser(Mockito.any(User.class))).thenReturn(mockUser);

String userJson = "{\"id\":2,\"name\":\"Bob\"}";

mockMvc.perform(MockMvcRequestBuilders.post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(userJson))

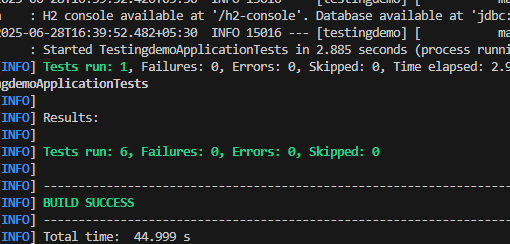
.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Bob"))

.andExpect(jsonPath("$.id").value(2));

}

}



**Exercise 6: Test Service Exception Handling**

**package com.example.testingdemo.service;**

**import com.example.testingdemo.model.User;**

**import com.example.testingdemo.repository.UserRepository;**

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

**import org.mockito.InjectMocks;**

**import org.mockito.Mock;**

**import org.mockito.MockitoAnnotations;**

**import com.example.testingdemo.exception.UserNotFoundException;**

**import java.util.Optional;**

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

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

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

**public class UserServiceTest {**

**@Mock**

**private UserRepository userRepository;**

**@InjectMocks**

**private UserService userService;**

**public UserServiceTest() {**

**MockitoAnnotations.openMocks(this);**

**}**

**@Test**

**public void testGetUserById() {**

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

**mockUser.setId(1L);**

**mockUser.setName("John");**

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

**User result = userService.getUserById(1L);**

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

**}**

**@Test**

**public void testUserNotFoundThrowsException() {**

**when(userRepository.findById(999L)).thenReturn(Optional.empty());**

**Exception exception = assertThrows(UserNotFoundException.class, () -> {**

**userService.getUserById(999L);**

**});**

**assertEquals("User not found with id: 999", exception.getMessage());**

**}**

**}**

**package com.example.testingdemo.exception;**

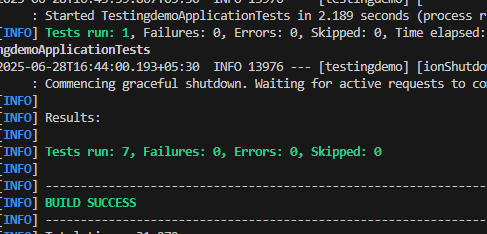
**public class UserNotFoundException extends RuntimeException {**

**public UserNotFoundException(String message) {**

**super(message);**

**}**

**}**



**Exercise 7: Test Custom Repository Query**

package com.example.testingdemo.repository;

import com.example.testingdemo.model.User;

import org.junit.jupiter.api.Test;

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

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

import java.util.List;

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

@DataJpaTest

public class UserRepositoryTest {

@Autowired

private UserRepository userRepository;

@Test

public void testFindByName() {

// Save users

User user1 = new User();

user1.setId(101L);

user1.setName("Alice");

User user2 = new User();

user2.setId(102L);

user2.setName("Bob");

User user3 = new User();

user3.setId(103L);

user3.setName("Alice");

userRepository.save(user1);

userRepository.save(user2);

userRepository.save(user3);

// Test findByName

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

assertEquals(2, result.size());

}

}

package com.example.testingdemo.repository;

import java.util.List;

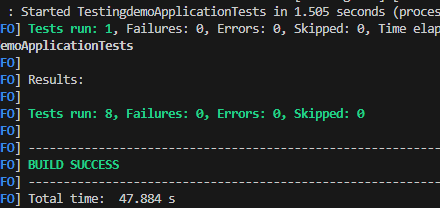
import org.springframework.data.jpa.repository.JpaRepository;

import com.example.testingdemo.model.User;

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name);

}



**Exercise 8: Test Controller Exception Handling**

@Test

public void testGetUser\_NotFound() throws Exception {

when(userService.getUserById(999L)).thenThrow(new UserNotFoundException("User not found"));

mockMvc.perform(MockMvcRequestBuilders.get("/users/999"))

.andExpect(status().isNotFound())

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

}

package com.example.testingdemo.exception;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.NoSuchElementException;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(UserNotFoundException.class)

public ResponseEntity<String> handleUserNotFound(UserNotFoundException ex) {

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

}

// Optional: handle other exceptions

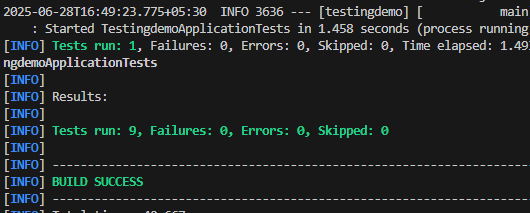
@ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNoSuchElement(NoSuchElementException ex) {

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

}

}



**Exercise 9: Parameterized Test with JUnit**

package com.example.testingdemo;

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

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;

import com.example.testingdemo.service.CalculatorService;

public class ParameterizedTestExample {

CalculatorService calculatorService = new CalculatorService();

@ParameterizedTest

@CsvSource({

"1, 2, 3",

"10, 5, 15",

"-3, 3, 0",

"0, 0, 0"

})

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

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

assertEquals(expected, result);

}

}

