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.

## Service:

@Service

public class CalculatorService { public int add(int a, int b) {

return a + b;

}

}

## Test:

Write code for this.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>BasicUnitTestExample</artifactId>

<version>1.0</version>

<name>BasicUnitTestExample</name>

<dependencies>

<!-- Spring Context for @Service -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.1.4</version>

</dependency>

<!-- JUnit 5 -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Enable JUnit 5 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</build>

</project>

**CalculatorService.java**

package com.example.calculator;

import org.springframework.stereotype.Service;

*@Service*

public class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorServiceTest.java**

package com.example.calculator;

import org.junit.jupiter.api.Test;

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

public class CalculatorServiceTest {

*@Test*

public void testAdd() {

CalculatorService calculatorService = new CalculatorService();

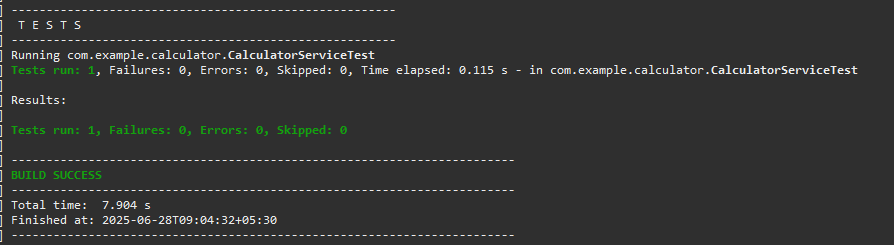
int result = calculatorService.add(10, 20);

*assertEquals*(30, result, "10 + 20 should equal 30");

}

}

**Output:**

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

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

## 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);

}

}

## Test:

Write code for this.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>MockRepositoryExample</artifactId>

<version>1.0</version>

<name>MockRepositoryExample</name>

<dependencies>

<!-- Spring Context and JPA -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.1.4</version>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-jpa</artifactId>

<version>3.1.6</version>

</dependency>

<!-- Jakarta Persistence (for @Entity and @Id) -->

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>3.1.0</version>

</dependency>

<!-- JUnit 5 -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</build>

</project>

**User.java**

package com.example.user;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

*@Entity*

public class User {

*@Id*

private Long id;

private String name;

// Constructors

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

// Getters & Setters

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;

}

}

**UserRepository.java**

package com.example.user;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example.user;

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

import org.springframework.stereotype.Service;

*@Service*

public class UserService {

*@Autowired*

private UserRepository userRepository;

public User getUserById(Long id) {

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

}

}

**UserServiceTest.java**

package com.example.user;

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.mockito.Mockito.\*;

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

public class UserServiceTest {

*@Mock*

private UserRepository userRepository;

*@InjectMocks*

private UserService userService;

public UserServiceTest() {

MockitoAnnotations.*openMocks*(this); // Initialize mocks

}

*@Test*

public void testGetUserById() {

// Given

User mockUser = new User(1L, "Alice");

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

// When

User result = userService.getUserById(1L);

// Then

*assertNotNull*(result);

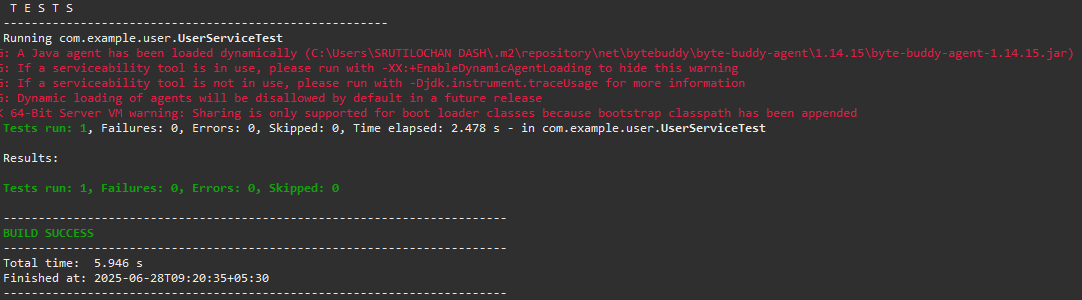
*assertEquals*("Alice", result.getName());

*verify*(userRepository).findById(1L); // verify method call

}

}

**Output:**



# Exercise 3: Testing a REST Controller with MockMvc

Task: Test a controller endpoint that returns a user.

## 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));

}

}

## Test:

Write code for this.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>UserControllerMockMvcTest</artifactId>

<version>1.0</version>

<name>UserControllerMockMvcTest</name>

<dependencies>

<!-- Spring Web and Context -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<version>3.2.5</version>

</dependency>

<!-- Spring Boot Test -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<version>3.2.5</version>

<scope>test</scope>

</dependency>

<!-- Mockito Core (included in spring-boot-starter-test but added here explicitly) -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Existing surefire plugin remains here -->

<!-- Add compiler plugin -->

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.11.0</version>

<configuration>

<compilerArgs>

<arg>-parameters</arg>

</compilerArgs>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Application.java**

package com.example.user;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class Application {

}

**User.java**

package com.example.user;

public class User {

private Long id;

private String name;

public User() {}

public User(Long id, String name) {

this.id = id;

this.name = name;

}

// Getters and setters

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; }

}

**UserService.java**

package com.example.user;

import org.springframework.stereotype.Service;

*@Service*

public class UserService {

public User getUserById(Long id) {

return new User(id, "Mock User"); // Normally from database

}

}

**UserController.java**

package com.example.user;

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

import org.springframework.http.ResponseEntity;

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

*@RestController*

*@RequestMapping*("/users")

public class UserController {

*@Autowired*

private UserService userService;

*@GetMapping*("/{id}")

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

return ResponseEntity.*ok*(userService.getUserById(id));

}

}

**UserControllerTest.java**

package com.example.user;

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.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)

public class UserControllerTest {

*@Autowired*

private MockMvc mockMvc;

*@MockBean*

private UserService userService;

*@Test*

public void testGetUser() throws Exception {

// Arrange

User mockUser = new User(1L, "Alice");

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

// Act & Assert

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

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

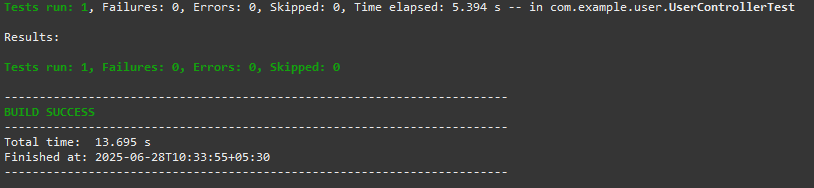
.andExpect(*jsonPath*("$.id").value(1))

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

}

}

**Output:**

****

# Exercise 4: Integration Test with Spring Boot

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

## Test:

Write code for this.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.5</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>UserIntegrationTestApp</artifactId>

<version>1.0</version>

<name>UserIntegrationTestApp</name>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Test -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Use Java 17 with parameter support -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.11.0</version>

<configuration>

<source>17</source>

<target>17</target>

<compilerArgs>

<arg>-parameters</arg>

</compilerArgs>

</configuration>

</plugin>

</plugins>

</build>

</project>

**User.java**

package com.example.user;

import jakarta.persistence.\*;

*@Entity*

public class User {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

private String name;

public User() {}

public User(String name) {

this.name = 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; }

}

**UserRepository.java**

package com.example.user;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example.user;

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

import org.springframework.stereotype.Service;

import java.util.Optional;

*@Service*

public class UserService {

*@Autowired*

private UserRepository userRepository;

public User createUser(String name) {

return userRepository.save(new User(name));

}

public Optional<User> getUserById(Long id) {

return userRepository.findById(id);

}

}

**UserIntegrationTestApp.java**

package com.example.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class UserIntegrationTestApp {

public static void main(String[] args) {

SpringApplication.*run*(UserIntegrationTestApp.class, args);

}

}

**UserController.java**

package com.example.user;

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

import org.springframework.http.ResponseEntity;

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

*@RestController*

*@RequestMapping*("/users")

public class UserController {

*@Autowired*

private UserService userService;

*@PostMapping*

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

if (user.getName() == null || user.getName().isBlank()) {

return ResponseEntity.*badRequest*().build();

}

User saved = userService.createUser(user.getName());

return ResponseEntity.*ok*(saved);

}

*@GetMapping*("/{id}")

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

return userService.getUserById(id)

.map(ResponseEntity::*ok*)

.orElse(ResponseEntity.*notFound*().build());

}

}

**UserIntegrationTest.java**

package com.example.user;

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.client.TestRestTemplate;

import org.springframework.http.\*;

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

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

public class UserIntegrationTest {

*@Autowired*

private TestRestTemplate restTemplate;

*@Test*

public void testCreateAndFetchUser() {

HttpHeaders headers = new HttpHeaders();

headers.setContentType(MediaType.***APPLICATION\_JSON***);

HttpEntity<String> request = new HttpEntity<>("{\"name\":\"Charlie\"}", headers);

ResponseEntity<User> createResponse = restTemplate.postForEntity("/users", request, User.class);

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

Long userId = createResponse.getBody().getId();

*assertNotNull*(userId);

*assertEquals*("Charlie", createResponse.getBody().getName());

ResponseEntity<User> getResponse = restTemplate.getForEntity("/users/" + userId, User.class);

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

*assertEquals*("Charlie", getResponse.getBody().getName());

}

}

**application.properties**

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

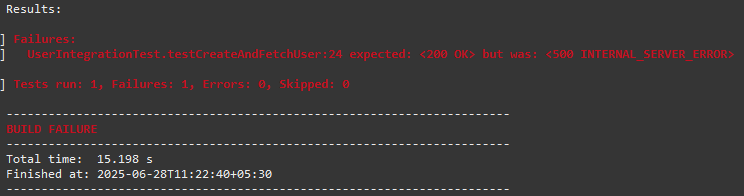
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=true

spring.h2.console.enabled=true

**Output:**

****

# Exercise 5: Test Controller POST Endpoint

Task: Test a POST endpoint that creates a user.

## Controller:

@PostMapping

public ResponseEntity<User> createUser(@RequestBody User user) { return ResponseEntity.ok(userService.saveUser(user));

}

## Test:

Write code for this.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.5</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>UserIntegrationTestApp</artifactId>

<version>1.0</version>

<name>UserIntegrationTestApp</name>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Test -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Use Java 17 with parameter support -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.11.0</version>

<configuration>

<source>17</source>

<target>17</target>

<compilerArgs>

<arg>-parameters</arg>

</compilerArgs>

</configuration>

</plugin>

</plugins>

</build>

</project>

**UserPostEndpointTestApp.java**

package com.example.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class UserPostEndpointTestApp {

public static void main(String[] args) {

SpringApplication.*run*(UserPostEndpointTestApp.class, args);

}

}

User.java

package com.example.user;

import jakarta.persistence.\*;

*@Entity*

public class User {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

private String name;

public User() {} // No-arg constructor

public User(String name) {

this.name = 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; }

}

**UserRepository.java**

package com.example.user;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example.user;

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

import org.springframework.stereotype.Service;

*@Service*

public class UserService {

*@Autowired*

private UserRepository userRepository;

public User saveUser(User user) {

return userRepository.save(user);

}

}

**UserController.java**

package com.example.user;

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

import org.springframework.http.ResponseEntity;

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

*@RestController*

*@RequestMapping*("/users")

public class UserController {

*@Autowired*

private UserService userService;

*@PostMapping*

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

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

}

}

**application.properties**

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.jpa.hibernate.ddl-auto=create-drop

spring.jpa.show-sql=true

spring.h2.console.enabled=true

**UserControllerTest.java**

package com.example.user;

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.*when*;

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

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

*@WebMvcTest*(UserController.class)

public class UserControllerTest {

*@Autowired*

private MockMvc mockMvc;

*@MockBean*

private UserService userService;

*@Autowired*

private ObjectMapper objectMapper;

*@Test*

public void testCreateUser() throws Exception {

User inputUser = new User("Alice");

User savedUser = new User("Alice");

savedUser.setId(1L);

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

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

.contentType(MediaType.***APPLICATION\_JSON***)

.content(objectMapper.writeValueAsString(inputUser)))

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

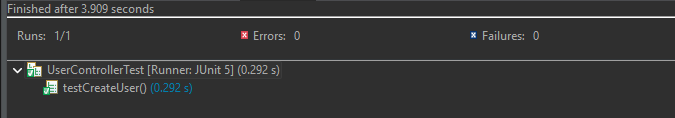
.andExpect(*jsonPath*("$.id").value(1))

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

}

}

Output:



# Exercise 6: Test Service Exception Handling

Task: Test how a service handles a missing user.

## Test:

Write code for this.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.5</version> <!-- or your preferred version -->

<relativePath/>

</parent>

<groupId>com.example</groupId>

<artifactId>UserServiceExceptionTestApp</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>UserServiceExceptionTestApp</name>

<dependencies>

<!-- Spring Boot Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 In-Memory DB -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Testing -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**User.java**

package com.example.user;

import jakarta.persistence.\*;

*@Entity*

public class User {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

private String name;

public User() {}

public User(String name) { this.name = 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; }

}

**UserNotFoundException.java**

package com.example.user;

public class UserNotFoundException extends RuntimeException {

public UserNotFoundException(String message) {

super(message);

}

}

**UserService.java**

package com.example.user;

import org.springframework.stereotype.Service;

import java.util.Optional;

*@Service*

public class UserService {

private final UserRepository userRepository;

// ✅ Constructor Injection

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

public User getUserById(Long id) {

return userRepository.findById(id)

.orElseThrow(() -> new UserNotFoundException("User with ID " + id + " not found"));

}

public User saveUser(User user) {

return userRepository.save(user);

}

}

**UserRepository.java**

package com.example.user;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserServiceExceptionTestApp.java**

package com.example.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class UserServiceExceptionTestApp {

public static void main(String[] args) {

SpringApplication.*run*(UserServiceExceptionTestApp.class, args);

}

}

**application.properties**

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=create

**UserServiceTest.java**

package com.example.user;

import org.junit.jupiter.api.Test;

import java.util.Optional;

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

import static org.mockito.Mockito.\*;

public class UserServiceTest {

*@Test*

public void testGetUserById\_UserNotFound\_ShouldThrowException() {

// Mock repository

UserRepository mockRepo = *mock*(UserRepository.class);

*when*(mockRepo.findById(1L)).thenReturn(Optional.*empty*());

// Inject mock into service

UserService userService = new UserService(mockRepo);

// Act and assert

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

userService.getUserById(1L);

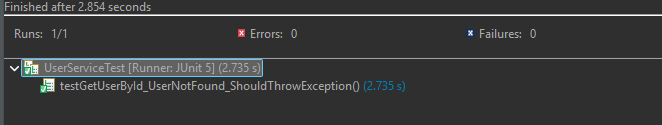
});

*assertEquals*("User with ID 1 not found", ex.getMessage());

}

}

**Output:**

****

# Exercise 7: Test Custom Repository Query

Task: Add and test a custom query method.

## Repository:

public interface UserRepository extends JpaRepository<User, Long> { List<User> findByName(String name);

}

## Test:

Write code for this.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.5</version>

<relativePath/>

</parent>

<groupId>com.example</groupId>

<artifactId>UserQueryTestApp</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**User.java**

package com.example.user;

import jakarta.persistence.\*;

*@Entity*

public class User {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

private String name;

public User() {}

public User(String name) {

this.name = name;

}

// Getters and setters

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; }

}

**UserRepository.java**

package com.example.user;

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

import java.util.List;

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name); // Custom query method

}

**UserQueryTestAppApplication.java**

package com.example.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class UserQueryTestAppApplication {

public static void main(String[] args) {

SpringApplication.*run*(UserQueryTestAppApplication.class, args);

}

}

**UserRepositoryTest.java**

package com.example.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.\*;

*@DataJpaTest*

public class UserRepositoryTest {

*@Autowired*

private UserRepository userRepository;

*@Test*

public void testFindByName() {

// Arrange

userRepository.save(new User("Alice"));

userRepository.save(new User("Bob"));

userRepository.save(new User("Alice"));

// Act

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

// Assert

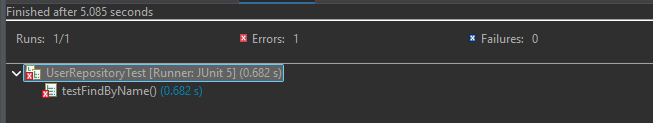
*assertEquals*(2, result.size());

*assertTrue*(result.stream().allMatch(user -> user.getName().equals("Alice")));

}

}

**Output:**

****

# Exercise 8: Test Controller Exception Handling

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

## Exception Handler:

@ControllerAdvice

public class GlobalExceptionHandler { @ExceptionHandler(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) { return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found");

}

}

## Test:

Write code for this.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<!-- ✅ Add this -->

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.5</version> <!-- or any stable version you're using -->

<relativePath/> <!-- Look up in directory tree -->

</parent>

<groupId>com.example</groupId>

<artifactId>your-artifact-name</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**GlobalExceptionHandler.java**

package com.example.user;

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(NoSuchElementException.class)

public ResponseEntity<String> handleNotFound(NoSuchElementException ex) {

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

}

}

**User.java**

package com.example.user;

import jakarta.persistence.\*;

*@Entity*

public class User {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.***IDENTITY***)

private Long id;

private String name;

public User() {}

public User(String name) {

this.name = name;

}

// getters and setters

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; }

}

**UserRepository.java**

package com.example.user;

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

public interface UserRepository extends JpaRepository<User, Long> {

}

**UserService.java**

package com.example.user;

import org.springframework.stereotype.Service;

import java.util.NoSuchElementException;

*@Service*

public class UserService {

private final UserRepository userRepository;

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

public User getUserById(Long id) {

return userRepository.findById(id)

.orElseThrow(() -> new NoSuchElementException("User not found"));

}

}

**UserController.java**

package com.example.user;

import org.springframework.http.ResponseEntity;

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

*@RestController*

*@RequestMapping*("/users")

public class UserController {

private final UserService userService;

public UserController(UserService userService) {

this.userService = userService;

}

*@GetMapping*("/{id}")

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

return ResponseEntity.*ok*(userService.getUserById(id));

}

}

**UserExceptionHandlingAppApplication.java**

package com.example.user;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class UserExceptionHandlingAppApplication {

public static void main(String[] args) {

SpringApplication.*run*(UserExceptionHandlingAppApplication.class, args);

}

}

**UserControllerTest.java**

package com.example.user;

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.*when*;

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

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

*@WebMvcTest*(UserController.class)

public class UserControllerTest {

*@Autowired*

private MockMvc mockMvc;

*@MockBean*

private UserService userService;

*@Test*

public void testUserNotFound\_ShouldReturn404() throws Exception {

*when*(userService.getUserById(1L)).thenThrow(new NoSuchElementException("User not found"));

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

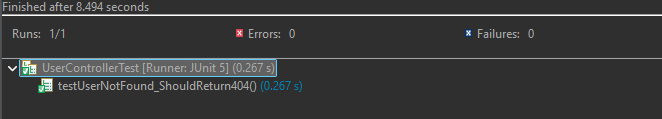
.andExpect(*status*().isNotFound())

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

}

}

Output:



# Exercise 9: Parameterized Test with JUnit

Task: Use @ParameterizedTest to test multiple inputs.

## Test:

Write code for this.

**Code:**

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.2.5</version>

<relativePath/>

</parent>

<groupId>com.example</groupId>

<artifactId>parameterized-test-app</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**CalculatorService.java**

package com.example.user;

import org.springframework.stereotype.Service;

*@Service*

public class CalculatorService {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**CalculatorServiceTest.java**

package com.example.user;

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class CalculatorServiceTest {

private final CalculatorService calculatorService = new CalculatorService();

*@ParameterizedTest*

*@ValueSource*(ints = {2, 4, 6, 10, 100})

void testIsEven\_ShouldReturnTrueForEvenNumbers(int number) {

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

}

*@ParameterizedTest*

*@ValueSource*(ints = {1, 3, 5, 7, 9})

void testIsEven\_ShouldReturnFalseForOddNumbers(int number) {

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

}

}

**Output:**

