Wk11 Assignment: Junit

Problem Statement

Test any existing application Controller using Spring Junit & Mockito framework.

Junit:

Requirement:

Use Junit test cases for Controller CRUD operations.

Mockito:

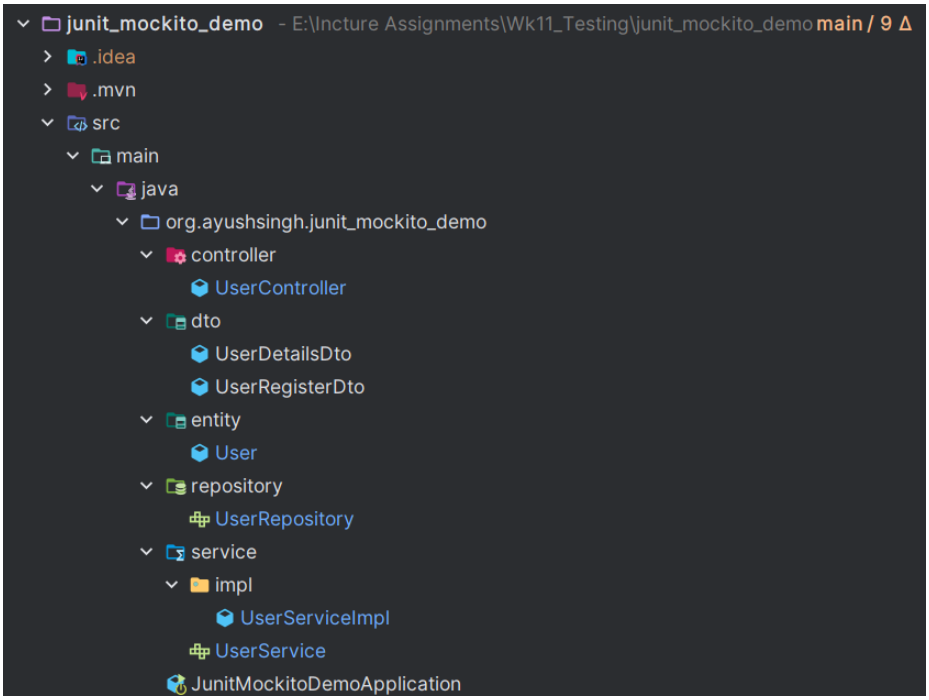
Requirement:

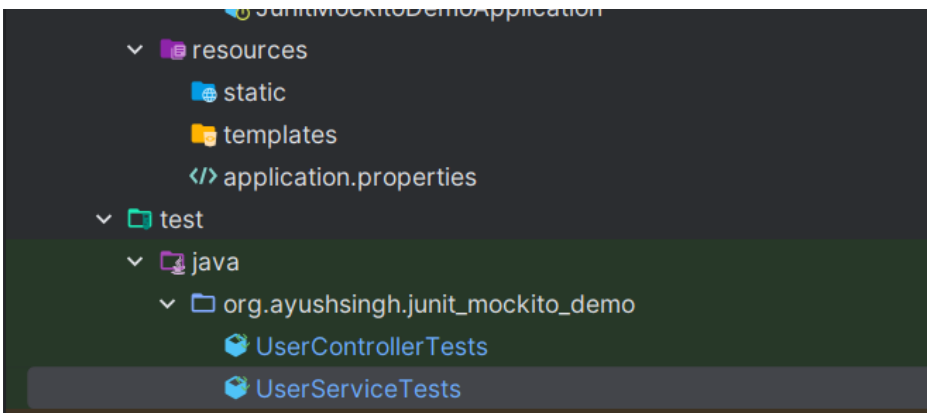
Use Mockito test cases for Controller CRUD operations.

Code Repository

The complete code can be found here-
https://github.com/singhayush20/Assignments/tree/main/Wk11_Testing/junit_mockito_demo

Project Structure





Required dependencies

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>5.11.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.4.0</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Entity

```
package org.ayushsingh.junit_mockito_demo.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.Setter;

import java.util.Objects;


/**
 * Entity class representing a user in the system.
 * This class is mapped to the 'user' table in the database.
 *
 * @author Ayush Singh
 * @version 1.0.0
 * @since 2024-04-12
 */
@Table(name = "user")
@Entity

@Getter
```

```java
@Setter
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Long userId;

    @Column(name = "username", nullable = false)
    private String username;

    @Column(name = "password", nullable = false)
    private String password;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "email", nullable = false)
    private String email;

    @Column(name = "phone", nullable = false, length = 10)
    private Long phone;


    public User() {

    }

    public User(Long userId, String username, String password, String name, String email,
Long phone) {
        this.userId = userId;
        this.username = username;
        this.password = password;
        this.name = name;
        this.email = email;
        this.phone = phone;
    }

    public User(String username, String password, String name, String email, Long phone) {
        this.username = username;
        this.password = password;
        this.name = name;
        this.email = email;
        this.phone = phone;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(username, user.username) && Objects.equals(password,
user.password) && Objects.equals(name, user.name) && Objects.equals(email, user.email) &&
Objects.equals(phone, user.phone);
    }

    @Override
    public int hashCode() {
        return Objects.hash(username, password, name, email, phone);
    }
}
```

Service and Implementation

```java
package org.ayushsingh.junit_mockito_demo.service;

import org.ayushsingh.junit_mockito_demo.dto.UserDetailsDto;
import org.ayushsingh.junit_mockito_demo.dto.UserRegisterDto;
```

```java
import java.util.List;

/**
 * Service interface for managing user-related operations.
 * This interface defines methods to create, retrieve, update, and delete users,
 * as well as retrieve a list of all users.
 *
 * @author Ayush Singh
 * @since 2024-04-12
 * @version 1.0.0
 */
public interface UserService {

    UserDetailsDto createUser(UserRegisterDto userRegisterDto);

    UserDetailsDto getUser(Long userId);

    List<UserDetailsDto> getAllUsers();

    void deleteUser(Long userId);

    UserDetailsDto updateUser(UserDetailsDto userDto);
}
```

```java
package org.ayushsingh.junit_mockito_demo.service.impl;

import lombok.RequiredArgsConstructor;
import org.ayushsingh.junit_mockito_demo.dto.UserDetailsDto;
import org.ayushsingh.junit_mockito_demo.dto.UserRegisterDto;
import org.ayushsingh.junit_mockito_demo.entity.User;
import org.ayushsingh.junit_mockito_demo.repository.UserRepository;
import org.ayushsingh.junit_mockito_demo.service.UserService;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

/**
 * Implementation of the UserService interface to manage user-related operations.
 * This service class provides methods to create, retrieve, update, and delete user
information.
 *
 * @author Ayush Singh
 * @since 2024-04-12
 * @version 1.0.0
 */
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

    private final UserRepository userRepository;


    /**
     * Create a new user with the provided details.
     *
     * @param userRegisterDto The details of the user to be created.
     * @return UserDetailsDto with the details of the created user.
     */
    @Override
    public UserDetailsDto createUser(UserRegisterDto userRegisterDto) {
        User newUser = new User();
        newUser.setUsername(userRegisterDto.getUsername());
        newUser.setPassword(userRegisterDto.getPassword());
        newUser.setName(userRegisterDto.getName());
        newUser.setEmail(userRegisterDto.getEmail());
        newUser.setPhone(userRegisterDto.getPhone());
```

```
        User savedUser=userRepository.save(newUser);
        return UserDetailsDto.builder()
                .userId(savedUser.getUserId())
                .username(savedUser.getUsername())
                .name(savedUser.getName())
                .email(savedUser.getEmail())
                .phone(savedUser.getPhone())
                .build();
    }


    /**
     * Retrieve user details by userId.
     *
     * @param userId The unique identifier of the user.
     * @return UserDetailsDto with the details of the user if found, otherwise null.
     */
    @Override
    public UserDetailsDto getUser(Long userId) {
        Optional<User> user = userRepository.findById(userId);
        if (user.isEmpty()) {
            return null;
        }
        return UserDetailsDto.builder()
                .userId(user.get().getUserId())
                .username(user.get().getUsername())
                .name(user.get().getName())
                .email(user.get().getEmail())
                .phone(user.get().getPhone())
                .build();
    }


    /**
     * Retrieve details of all users.
     *
     * @return List<UserDetailsDto> with details of all users.
     */
    @Override
    public List<UserDetailsDto> getAllUsers() {
        List<User> users = userRepository.findAll();
        return users.stream().map(user -> {
            UserDetailsDto userDetailsDto = new UserDetailsDto();
            userDetailsDto.setUserId(user.getUserId());
            userDetailsDto.setUsername(user.getUsername());
            userDetailsDto.setName(user.getName());
            userDetailsDto.setEmail(user.getEmail());
            userDetailsDto.setPhone(user.getPhone());
            return userDetailsDto;
        }).collect(Collectors.toList());
    }


    /**
     * Delete a user by userId.
     *
     * @param userId The unique identifier of the user to be deleted.
     */
    @Override
    public void deleteUser(Long userId) {
        userRepository.deleteById(userId);
    }


    /**
     * Update user details.
     *
     * @param userDto The updated details of the user.
     * @return UserDetailsDto with the updated user details if found, otherwise null.
```

```
     */
    @Override
    public UserDetailsDto updateUser(UserDetailsDto userDto) {
        Optional<User> userOptional = userRepository.findById(userDto.getUserId());
        if (userOptional.isEmpty()) {
            return null;
        }
        User user = userOptional.get();
        user.setUsername(userDto.getUsername());
        user.setName(userDto.getName());
        user.setEmail(userDto.getEmail());
        user.setPhone(userDto.getPhone());
        return UserDetailsDto.builder()
                .userId(userRepository.save(user).getUserId())
                .username(user.getUsername())
                .name(user.getName())
                .email(user.getEmail())
                .phone(user.getPhone())
                .build();
    }
}
```

User Repository

```
package org.ayushsingh.junit_mockito_demo.repository;

import org.ayushsingh.junit_mockito_demo.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
/**
 * Repository interface for managing user entities in the database.
 * This interface extends JpaRepository provided by Spring Data JPA,
 * which provides various methods for CRUD operations on the User entity.
 *
 * @author Ayush Singh
 * @since 2024-04-12
 * @version 1.0.0
 */
public interface UserRepository extends JpaRepository<User, Long> {
}
```

UserController

```
package org.ayushsingh.junit_mockito_demo.controller;

import lombok.RequiredArgsConstructor;
import org.ayushsingh.junit_mockito_demo.dto.UserDetailsDto;
import org.ayushsingh.junit_mockito_demo.dto.UserRegisterDto;
import org.ayushsingh.junit_mockito_demo.service.UserService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/**
 * Controller class to handle user-related HTTP requests.
 * This controller provides endpoints for creating, retrieving, updating, and deleting user
information.
 *
 * @author Ayush Singh
 * @version 1.0.0
 * @since 2024-04-12
 */
@RestController
@RequestMapping("/api/v1/user")
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;
```

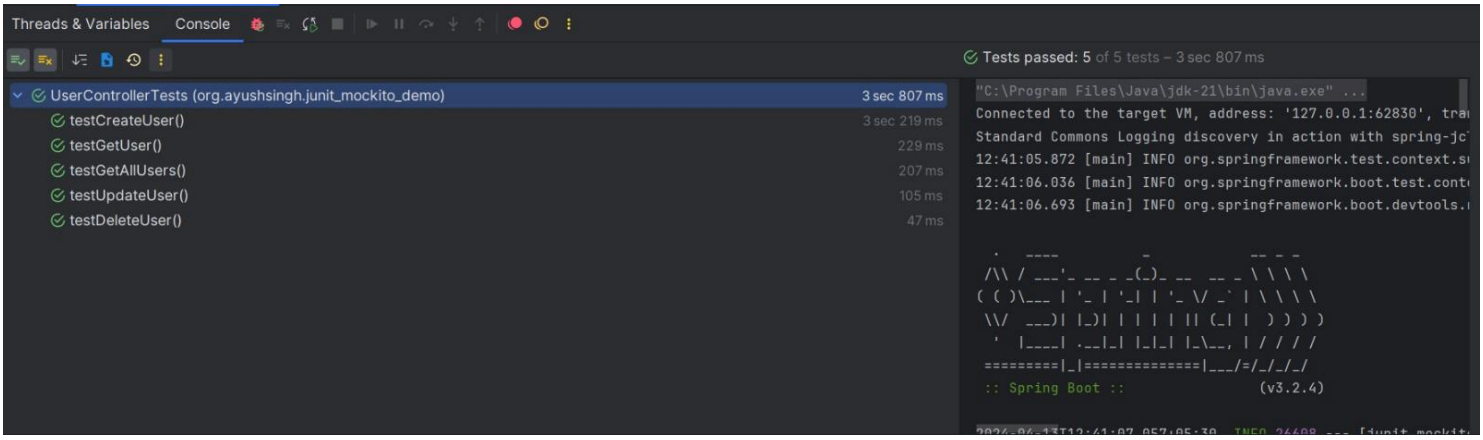
```java
    /**
     * Endpoint to create a new user.
     *
     * @param userRegisterDto The details of the user to be created.
     * @return ResponseEntity with the created user details.
     */
    @PostMapping("/create")
    public ResponseEntity<UserDetailsDto> createUser(@RequestBody UserRegisterDto
userRegisterDto) {
        UserDetailsDto userDetailsDto = userService.createUser(userRegisterDto);
        return ResponseEntity.status(HttpStatus.CREATED).body(userDetailsDto);
    }

    /**
     * Endpoint to retrieve user details by userId.
     *
     * @param userId The unique identifier of the user.
     * @return ResponseEntity with the user details.
     */
    @GetMapping("/{userId}")
    public ResponseEntity<UserDetailsDto> getUser(@PathVariable Long userId) {
        UserDetailsDto userDetailsDto = userService.getUser(userId);

        return ResponseEntity.ok(userDetailsDto);

    }

    /**
     * Endpoint to retrieve details of all users.
     *
     * @return ResponseEntity with a list of all users' details.
     */
    @GetMapping("/all")
    public ResponseEntity<List<UserDetailsDto>> getAllUsers() {
        List<UserDetailsDto> userDetailsDtoList = userService.getAllUsers();
        return ResponseEntity.ok(userDetailsDtoList);
    }

    /**
     * Endpoint to delete a user by userId.
     *
     * @param userId The unique identifier of the user to be deleted.
     * @return ResponseEntity with no content.
     */
    @DeleteMapping("/{userId}")
    public ResponseEntity<Void> deleteUser(@PathVariable Long userId) {
        userService.deleteUser(userId);
        return ResponseEntity.noContent().build();
    }

    /**
     * Endpoint to update user details.
     *
     * @param userDetailsDto The updated details of the user.
     * @return ResponseEntity with the updated user details.
     */
    @PutMapping("/update")
    public ResponseEntity<UserDetailsDto> updateUser(@RequestBody UserDetailsDto
userDetailsDto) {
        UserDetailsDto updatedUser = userService.updateUser(userDetailsDto);

        return ResponseEntity.ok(updatedUser);
    }
}
```

Test using Junit and Rest Assured

```java
package org.ayushsingh.junit_mockito_demo;

import io.restassured.RestAssured;
import io.restassured.http.ContentType;
import org.ayushsingh.junit_mockito_demo.dto.UserDetailsDto;
import org.ayushsingh.junit_mockito_demo.dto.UserRegisterDto;
import org.junit.jupiter.api.*;
import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;
import org.springframework.boot.test.context.SpringBootTest;

import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;


/**
 * Tests for the {@link org.ayushsingh.junit_mockito_demo.controller.UserController}.
 * These tests use RestAssured to interact with the UserController endpoints.
 *
 * @author Ayush Singh
 * @since 2024-04-12
 * @version 1.0.0
 */
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT)
@TestMethodOrder(OrderAnnotation.class)
public class UserControllerTests {

    private static Long userId;

    @BeforeAll
    public static void setUp() {
        final String BASE_URL = "http://localhost:8086" + "/api/v1/user";
        RestAssured.baseURI = BASE_URL;
        userId=1L;
        RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();
    }

    @Test
    @Order(1)
    public void testCreateUser() {
        UserRegisterDto userRegisterDto = new UserRegisterDto();
        userRegisterDto.setName("Ayush Singh");
        userRegisterDto.setEmail("ayushsingh20april@gmail.com");
        userRegisterDto.setUsername("ayush_20");
        userRegisterDto.setPhone(7867567434L);
        userRegisterDto.setPassword("password");

        given()
                .contentType(ContentType.JSON)
                .body(userRegisterDto)
                .when()
                .post("/create")
                .then()
                .statusCode(201)
                .body("name", equalTo(userRegisterDto.getName()))
                .body("email", equalTo(userRegisterDto.getEmail()))
                .body("username", equalTo(userRegisterDto.getUsername()))
                .body("phone", equalTo(userRegisterDto.getPhone()));


    }

    @Test
    @Order(2)
    public void testGetUser() {
        given()
                .when()
                .get("/{userId}", userId)
                .then()
```

```
                .statusCode(200)
                .body("name", notNullValue())
                .body("email", notNullValue())
                .body("username", notNullValue())
                .body("phone", notNullValue());
    }

    @Test
    @Order(3)
    public void testGetAllUsers() {
        given()
                .when()
                .get("/all")
                .then()
                .statusCode(200)
                .body("size()", greaterThan(0));
    }

    @Test
    @Order(4)
    public void testUpdateUser() {
        UserDetailsDto userDetailsDto = new UserDetailsDto();
        userDetailsDto.setUserId(userId);
        userDetailsDto.setName("Ayush Pratap Singh");
        userDetailsDto.setEmail("ayush@outlook.com");
        userDetailsDto.setUsername("ayush201");
        userDetailsDto.setPhone(6756473234L);

        given()
                .contentType(ContentType.JSON)
                .body(userDetailsDto)
                .when()
                .put("/update")
                .then()
                .statusCode(200)
                .body("name", equalTo(userDetailsDto.getName()))
                .body("email", equalTo(userDetailsDto.getEmail()))
                .body("username", equalTo(userDetailsDto.getUsername()))
                .body("phone", equalTo(userDetailsDto.getPhone()));
    }

    @Test
    @Order(5)
    public void testDeleteUser() {
        given()
                .when()
                .delete("/{userId}", userId)
                .then()
                .statusCode(204);
    }
}
```

Test Results



Test using Junit and Mockito

```java
package org.ayushsingh.junit_mockito_demo;

import org.ayushsingh.junit_mockito_demo.dto.UserDetailsDto;
import org.ayushsingh.junit_mockito_demo.dto.UserRegisterDto;
import org.ayushsingh.junit_mockito_demo.entity.User;
import org.ayushsingh.junit_mockito_demo.repository.UserRepository;
import org.ayushsingh.junit_mockito_demo.service.impl.UserServiceImpl;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.springframework.test.context.junit.jupiter.SpringExtension;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.mockito.Mockito.*;


/**
 * Unit tests for the {@link UserServiceImpl}.
 * This class tests the CRUD operations using Mockito
 *
 * @author Ayush Singh
 * @since 2024-04-12
 * @version 1.0.0
 */
@ExtendWith(SpringExtension.class)
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class UserServiceTests {

    @Mock
    private UserRepository userRepository;

    @InjectMocks
    private UserServiceImpl userService;

    @Test
    @Order(1)
    void testCreateUser() {
        UserRegisterDto userRegisterDto = UserRegisterDto.builder()
                .username("ayush")
                .password("123abc")
                .name("Ayush Singh")
                .email("ayushsingh20november@gmail.com")
                .phone(1234567890L)
                .build();
        User savedUser = new User(1L, "ayush", "123abc", "Ayush Singh",
"ayushsingh20november@gmail.com", 1234567890L);
        when(userRepository.save(any(User.class))).thenReturn(savedUser);


        UserDetailsDto actualUserDto = userService.createUser(userRegisterDto);


        assertNotNull(actualUserDto.getUserId());
        assertEquals(userRegisterDto.getUsername(), actualUserDto.getUsername());
        assertEquals(userRegisterDto.getName(), actualUserDto.getName());
        assertEquals(userRegisterDto.getEmail(), actualUserDto.getEmail());
        assertEquals(userRegisterDto.getPhone(), actualUserDto.getPhone());
        verify(userRepository, times(1)).save(any(User.class));
    }
```

```
    @Test
    @Order(2)
    void testGetUser() {

        Long userId = 1L;
        User user = new User(userId, "ayush", "123abc", "Ayush Singh",
"ayushsingh20november@gmail.com", 7867564563L);
        when(userRepository.findById(userId)).thenReturn(Optional.of(user));


        UserDetailsDto actualUserDto = userService.getUser(userId);


        assertEquals(user.getUserId(), actualUserDto.getUserId());
        verify(userRepository, times(1)).findById(userId);
    }

    @Test
    @Order(3)
    void testUpdateUser() {

        User user = new User(1L, "ayush", "123abc", "Ayush Singh",
"ayushsingh20november@gmail.com", 7867564563L);
        User expectedUpdatedUser = new User(1L, "himanshu", "123abc123", "Himanshu Singh",
"himanshur@gmail.com", 7867564563L);
        when(userRepository.findById(1L)).thenReturn(Optional.of(user));
        when(userRepository.save(any(User.class))).thenReturn(expectedUpdatedUser);

        UserDetailsDto userDto = new UserDetailsDto(1L,"himanshu", "Himanshu Singh",
"himanshur@gmail.com", 7867564563L);
        UserDetailsDto updatedUserDto = userService.updateUser(userDto);


        assertEquals(userDto.getUserId(), updatedUserDto.getUserId());
        assertEquals(userDto.getUsername(), updatedUserDto.getUsername());
        assertEquals(userDto.getName(), updatedUserDto.getName());
        assertEquals(userDto.getEmail(), updatedUserDto.getEmail());
        assertEquals(userDto.getPhone(), updatedUserDto.getPhone());
        verify(userRepository, times(1)).findById(1L);
        verify(userRepository, times(1)).save(any(User.class));
    }



    @Test
    @Order(4)
    void testDeleteUser() {
        Long userId = 1L;


        userService.deleteUser(userId);

        verify(userRepository, times(1)).deleteById(userId);
    }


    @Test
    @Order(5)
    void testGetAllUsers() {

        User user1 = new User(1L, "ayush", "123abc", "Ayush Singh",
"ayushsingh20november@gmail.com", 7867564563L);
        User user2 = new User(2L, "himanshu", "123abc123", "Himanshu Singh",
"himanshur@gmail.com", 7867564563L);
        when(userRepository.findAll()).thenReturn(Arrays.asList(user1, user2));
        List<UserDetailsDto> expectedUserDtos = Arrays.asList(
                new UserDetailsDto(1L, "ayush", "Ayush Singh",
"ayushsingh20november@gmail.com", 7867564563L),
                new UserDetailsDto(2L, "himanshu", "Himanshu Singh", "himanshur@gmail.com",
```

```
7867564563L)
        );


        List<UserDetailsDto> actualUserDtos = userService.getAllUsers();


        assertEquals(expectedUserDtos.size(), actualUserDtos.size());
        for (int i = 0; i < expectedUserDtos.size(); i++) {
            assertEquals(expectedUserDtos.get(i).getUserId(),
actualUserDtos.get(i).getUserId());
        }
        verify(userRepository, times(1)).findAll();
    }


}
```

Test Results