

INFORME DE TEST UNITARIOS SPRINT II

CategoriaServiceTest

Codigo:

```
package Grupo8.Hairphoria;

import Grupo8.Hairphoria.dto.Categoria.CategoriaResponse;
import Grupo8.Hairphoria.entity.Categoria;
import Grupo8.Hairphoria.exceptions.ResourceNotFoundException;
import Grupo8.Hairphoria.repository.ICategoriaRepository;
import Grupo8.Hairphoria.service.CategoriaService;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
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.*;

class CategoriaServiceTest {

    @Mock
    private ICategoriaRepository categoriaRepository;

    @InjectMocks
    private CategoriaService categoriaService;


    @BeforeEach
    void setUp() {
        MockitoAnnotations.initMocks(this);
    }

    @Test
    void testFindByIdNonExistingCategoria() {
        // Arrange
        Long categoryId = 2L;

        // Mocking the repository behavior
        when(categoriaRepository.findById(categoryId)).thenReturn(Optional.empty());

        // Act and Assert
        Assertions.assertThrows(ResourceNotFoundException.class, () -> {
            categoriaService.findById(categoryId);
        });

        // Verify that the repository method was called once with the
        correct argument
        verify(categoriaRepository, times(1)).findById(categoryId);
    }
}
```



The screenshot shows the IDE interface with the test results. The test 'testFindByIdNonExistingCategoria' passed, and the process finished with exit code 0. The command line shows the test runner 'EclipseTest.cyclic.buffer.size=1648876' and the Java executable 'C:\Users\Yohana Zapata\jdk8\openjfx-19\bin\java.exe'.

El test es parte de la clase CategoriaServiceTest y se llama testFindByldNonExistingCategoria. Este test tiene como objetivo probar el escenario en el que se busca una categoría por su ID y no se encuentra en el repositorio.

En el test se realiza lo siguiente:

Se establece la configuración inicial utilizando la anotación @BeforeEach.

Se crea una variable categoryId con el valor 2L, que representa el ID de la categoría a buscar.

Se simula el comportamiento del repositorio utilizando el método when de Mockito. Se configura que al llamar al método findByld del repositorio con el categoryId, retorne un objeto Optional vacío.

Se utiliza el método assertThrows de la clase Assertions para verificar que se lance una excepción del tipo ResourceNotFoundException al llamar al método findByld del servicio de categoría (categoriaService.findByld(categoryId)).

Se verifica que el método findByld del repositorio haya sido llamado una vez con el argumento categoryId, utilizando el método verify de Mockito.

Este test se encarga de probar el manejo adecuado de la excepción ResourceNotFoundException cuando se intenta buscar una categoría que no existe en el repositorio.

Clase CategoriaServiceTest:

Test: testFindByldNonExistingCategoria

Descripción: Este test verifica el comportamiento del servicio CategoriaService cuando se busca una categoría por su ID y no se encuentra en el repositorio.

Resultado esperado: Se espera que el servicio arroje una excepción del tipo ResourceNotFoundException al buscar una categoría inexistente.

Mocking: Se utiliza la biblioteca Mockito para simular el comportamiento del repositorio. Se configura que el método findByld del repositorio retorne un objeto Optional vacío cuando se le pase el ID de la categoría.

ClienteServiceTest:

Codigo

```
package Grupo8.Hairphoria;

import Grupo8.Hairphoria.dto.Auth.ValidateMessageResponse;
import Grupo8.Hairphoria.dto.Cliente.ClienteRequest;
import Grupo8.Hairphoria.dto.Cliente.ClienteResponse;
import Grupo8.Hairphoria.entity.Cliente;
import Grupo8.Hairphoria.entity.Rol;
import Grupo8.Hairphoria.entity.Usuario;
import Grupo8.Hairphoria.entity.UsuarioRol;
import Grupo8.Hairphoria.exceptions.BadRequestException;
import Grupo8.Hairphoria.exceptions.ResourceNotFoundException;
import Grupo8.Hairphoria.repository.IClienteRepository;
import Grupo8.Hairphoria.repository.IRolRepository;
import Grupo8.Hairphoria.repository.IUsuarioRepository;
import Grupo8.Hairphoria.service.ClienteService;
import Grupo8.Hairphoria.service.Interfaces.IMailService;
```

```

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.mockito.stubbing.OngoingStubbing;
import org.springframework.security.crypto.password.PasswordEncoder;

import java.util.Collections;
import java.util.List;
import java.util.Locale;
import java.util.Optional;

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

class ClienteServiceTest {

    private ClienteService clienteService;

    @Mock
    private IClienteRepository clienteRepository;
    @Mock
    private IUsuarioRepository usuarioRepository;
    @Mock
    private IRolRepository rolRepository;
    @Mock
    private PasswordEncoder passwordEncoder;
    @Mock
    private IEmailService mailService;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
        clienteService = new ClienteService(clienteRepository,
        usuarioRepository, rolRepository, passwordEncoder, mailService);
    }

    @Test
    void findById_NonExistingId_ShouldThrowResourceNotFoundException() {
        // Arrange
        Long id = 1L;

        when(clienteRepository.findById(id)).thenReturn(Optional.empty());

        // Act & Assert
        assertThrows(ResourceNotFoundException.class, () ->
        clienteService.findById(id));
    }

    @Test
    void save_DuplicateEmail_ShouldThrowBadRequestException() {
        // Arrange
        ClienteRequest clienteRequest = new ClienteRequest("Yohana",
        "Zapata", 12345, "yohas@gmail.com", 312547, "BGRFD12*", "admin");
        clienteRequest.setEmail("test@example.com");

        when(usuarioRepository.findByEmail(clienteRequest.getEmail())).thenReturn(Optional.of(new Usuario()));

        // Act & Assert
    }

```

```
        assertThrows(BadRequestException.class, () ->
clienteService.save(clienteRequest));
    }

    @Test
    void update_NonExistingId_ShouldThrowBadRequestException() {
        // Arrange
        Long id = 1L;
        ClienteRequest clienteRequest = new ClienteRequest("Ana",
"Galarza", 123456, "ana@gmail.com", 3125478, "HNHG12*", "user");
        clienteRequest.setEmail("test@example.com");

when(clienteRepository.findById(id)).thenReturn(Optional.empty());

        // Act & Assert
        assertThrows(BadRequestException.class, () ->
clienteService.update(id, clienteRequest));
    }

    @Test
    void deleteById_NonExistingId_ShouldThrowResourceNotFoundException()
{
        // Arrange
        Long id = 1L;

when(clienteRepository.findById(id)).thenReturn(Optional.empty());

        // Act & Assert
        assertThrows(ResourceNotFoundException.class, () ->
clienteService.deleteById(id));
    }
}

Tests passed: 4 / 4 (100%) - 00:00:00
C:\Users\Yohana Zapata\.jdk\openjdk-19\bin\java.exe -ea -Didea.test.cyclic.buffer.size=1048576 "-jn
update_NonExistingId_ShouldThrowResourceNotFoundException
deleteById_NonExistingId_ShouldThrowResourceNotFoundException
save_DuplicateEmail_ShouldThrowBadRequestException
```

Test: findById_NonExistingId_ShouldThrowResourceNotFoundException

Descripción: Este test verifica el comportamiento del servicio ClienteService cuando se busca un cliente por su ID y no se encuentra en el repositorio.

Resultado esperado: Se espera que el servicio arroje una excepción del tipo `ResourceNotFoundException` al buscar un cliente inexistente.

Mocking: Se utiliza la biblioteca Mockito para simular el comportamiento del repositorio. Se configura que el método `findById` del repositorio retorne un objeto `Optional` vacío cuando se le pase el ID del cliente.

Test: `save_DuplicateEmail_ShouldThrowBadRequestException`

Descripción: Este test verifica el comportamiento del servicio `ClienteService` cuando se intenta guardar un cliente con un correo electrónico duplicado.

Resultado esperado: Se espera que el servicio arroje una excepción del tipo `BadRequestException` al intentar guardar un cliente con un correo electrónico duplicado.

Mocking: Se utiliza la biblioteca Mockito para simular el comportamiento del repositorio. Se configura que el método `findByEmail` del repositorio retorne un objeto `Optional` que representa la existencia de un usuario con el mismo correo electrónico.

Test: `update_NonExistingId_ShouldThrowBadRequestException`

Descripción: Este test verifica el comportamiento del servicio `ClienteService` cuando se intenta actualizar un cliente que no existe en el repositorio.

Resultado esperado: Se espera que el servicio arroje una excepción del tipo `BadRequestException` al intentar actualizar un cliente inexistente.

Mocking: Se utiliza la biblioteca Mockito para simular el comportamiento del repositorio. Se configura que el método `findById` del repositorio retorne un objeto `Optional` vacío cuando se le pase el ID del cliente.

Test: `deleteById_NonExistingId_ShouldThrowResourceNotFoundException`

Descripción: Este test verifica el comportamiento del servicio `ClienteService` cuando se intenta eliminar un cliente por su ID y no se encuentra en el repositorio.

Resultado esperado: Se espera que el servicio arroje una excepción del tipo `ResourceNotFoundException` al intentar eliminar un cliente inexistente.

Mocking: Se utiliza la biblioteca Mockito para simular el comportamiento del repositorio. Se configura que el método `findById` del repositorio retorne un objeto `Optional` vacío cuando se le pase el ID del cliente.

Estos tests se realizaron con el objetivo de asegurar el correcto funcionamiento

En el proyecto Hairphoria, se realizó un test en la clase ServicioServiceTest. A continuación, se presenta un resumen del test realizado:

ServicioServiceTest:

Codigo

```
package Grupo8.Hairphoria;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import static org.mockito.Mockito.*;

import java.util.*;

import Grupo8.Hairphoria.dto.Servicio.ServicioRequest;
import Grupo8.Hairphoria.dto.Servicio.ServicioResponse;
import Grupo8.Hairphoria.entity.*;
import Grupo8.Hairphoria.exceptions.ResourceNotFoundException;
import Grupo8.Hairphoria.repository.*;
import Grupo8.Hairphoria.service.Interfaces.IServicioService;
import Grupo8.Hairphoria.service.ServicioService;
import org.springframework.boot.test.context.SpringBootTest;

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

@SpringBootTest
public class ServicioServiceTest {

    @Mock
    private IServicioRepository servicioRepository;

    @Mock
    private ICategoriaRepository categoriaRepository;

    @Mock
    private IPalabraClaveRepository palabraClaveRepository;

    @Mock
    private IImagenRepository imagenRepository;


    @Mock
    private IAtributoRepository atributoRepository;

    @InjectMocks
    private ServicioService servicioService;

    @Test
    void save_InvalidEspecialidad_ShouldThrowResourceNotFoundException()
    {
        // Arrange
        ServicioRequest request = new ServicioRequest();
        request.setNombre("Servicio");
        request.setDescripcion("Descripción del servicio");
        request.setPrecio(10.0);
        request.setEspecialidad("Especialidad");

        when(categoriaRepository.findByEspecialidad(request.getEspecialidad())).
            thenReturn(Optional.empty());
    }
}
```

```
        assertThrows(ResourceNotFoundException.class, () ->
servicioService.save(request));
    }
}
```



Test: `save_InvalidEspecialidad_ShouldThrowResourceNotFoundException`

Descripción: Este test verifica el comportamiento del servicio `ServicioService` cuando se intenta guardar un servicio con una especialidad inválida.

Resultado esperado: Se espera que el servicio arroje una excepción del tipo `ResourceNotFoundException` al intentar guardar un servicio con una especialidad inválida.

Mocking: Se utiliza la biblioteca Mockito para simular el comportamiento de los repositorios. Se configura que el método `findByEspecialidad` del repositorio de categorías retorne un objeto `Optional` vacío cuando se le pase la especialidad del servicio.

Este test se realizó con el objetivo de asegurar que el servicio `ServicioService` maneje correctamente el escenario en el que se intenta guardar un servicio con una especialidad inválida.