SENA – CENTRO DE GESTION INDUSTRIAL

TECNOLOGIA DE ANALISIS Y DESARROLLO DE SOFTWARE
NUMERO DE FICHA (2675790)

INSTALACIÓN Y CONFIGURACIÓN DE COMPONENTES BACKEND

PRESENTADO A:
NELLY ISABEL RODRIGUEZ NAVARRO

PRESENTADO POR:
JEISON ESTEBAN ESPIT

Instalación del Backend (Java - Spring Boot)

Instalación de Java JDK 11:

- Descargar desde el sitio oficial de Oracle.
- Configurar variables de entorno (JAVA_HOME)

Instalación de NetBeans:

- Descargar e instalar NetBeans.

Clonación del repositorio:

- bash
- git clone https://github.com/Estebanfull/Ike_Asistencia_Backend.git

Configuración del archivo application.properties **:** Modificar las configuraciones de conexión:

spring.datasource.url=jdbc: mysql://localhost:3306/bd_ike
spring.datasource.username= root
spring.datasource.password= 142536
spring.datasource.driver-class-name= com.mysql.cj.jdbc.Driver

Ejecución del proyecto en NetBeans:

- Abrir el proyecto clonado del repositorio en NetBeans y ejecutar Run.

NULL,

Scripts de pruebas

Prueba de integración – conexión base de datos

```
package com.example.demo;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.transaction.annotation.Transactional;

import static org.assertj.core.api.Assertions.assertThat;

@SpringBootTest
@Transactional
public class DatabaseConfigTest {

  @Autowired
  private JdbcTemplate jdbcTemplate;

  @Test
  public void testDatabaseConnection() {
    String result = jdbcTemplate.queryForObject("SELECT 1", String.class);
    assertThat(result).isEqualTo("1");
  }
}
```

pruebas unitarias para LoginController usando Mockito y JUnit 5

```
package com.example.controller;

import com.example.model.UsuarioModel;
import com.example.service.UsuarioService;
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 org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

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

public class LoginControllerTest {

    @Mock
    private UsuarioService usuarioService;

    @InjectMocks
    private LoginController loginController;

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

    @Test
    public void testLoginSuccess() {
        // Datos de prueba
        UsuarioModel usuario = new UsuarioModel();
        usuario.setCorreoElectronico("test@example.com");
        usuario.setClave("password");
        usuario.setIdUsuario(1L);

        // Configura el mock para devolver el usuario cuando se llama al servicio
        when(usuarioService.getUsuarioByCorreoYClave("test@example.com",
"password")).thenReturn(usuario);

        // Llama al método del controlador
```

```
        ResponseEntity<?> response = loginController.login(usuario);

        // Verifica el resultado
        assertEquals(HttpStatus.OK, response.getStatusCode());
        LoginController.LoginResponse loginResponse =
(LoginController.LoginResponse) response.getBody();
        assertEquals("Credenciales válidas", loginResponse.getMessage());
        assertEquals(1L, loginResponse.getIdUsuario());
    }

    @Test
    public void testLoginFailure() {
        // Datos de prueba
        UsuarioModel usuario = new UsuarioModel();
        usuario.setCorreoElectronico("test@example.com");
        usuario.setClave("wrongpassword");

        // Configura el mock para devolver null cuando no se encuentra el usuario
        when(usuarioService.getUsuarioByCorreoYClave("test@example.com",
"wrongpassword")).thenReturn(null);

        // Llama al método del controlador
        ResponseEntity<?> response = loginController.login(usuario);

        // Verifica el resultado
        assertEquals(HttpStatus.UNAUTHORIZED, response.getStatusCode());
        String responseBody = (String) response.getBody();
        assertEquals("Credenciales incorrectas", responseBody);
    }
}
```

pruebas unitarias para AsignacionAsistenciasController usando Mockito y JUnit 5

```
package com.example.controller.Asistencia;

import com.example.model.AsistenciaModel;
import com.example.model.TecnicoModel;
import com.example.model.ProveedorModel;
import com.example.repository.AsistenciaRepository;
import com.example.repository.TecnicoRepository;
import com.example.repository.ProveedorRepository;
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 org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

import java.util.Optional;

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

public class AsignacionAsistenciasControllerTest {

    @Mock
    private AsistenciaRepository asistenciaRepository;

    @Mock
    private TecnicoRepository tecnicoRepository;

    @Mock
    private ProveedorRepository proveedorRepository;

    @InjectMocks
    private AsignacionAsistenciasController asignacionAsistenciasController;

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);
    }
```

```
  @Test
  public void testAsignarProveedorTecnicoAsistenciaSuccess() {
    // Datos de prueba
    Long idAsistencia = 1L;
    String nombreProveedor = "Proveedor A";
    String nombreTecnico = "Técnico B";

    AsistenciaModel asistencia = new AsistenciaModel();
    ProveedorModel proveedor = new ProveedorModel();
    TecnicoModel tecnico = new TecnicoModel();

    proveedor.setIdProveedor(1L);
    tecnico.setIdTecnico(2L);
    asistencia.setIdAsistencia(idAsistencia);


when(asistenciaRepository.findById(idAsistencia)).thenReturn(Optional.of(asistencia
));

when(proveedorRepository.findByNombre(nombreProveedor)).thenReturn(Optional.
of(proveedor));

when(tecnicoRepository.findByNombre(nombreTecnico)).thenReturn(Optional.of(tec
nico));

    ResponseEntity<String> response =
asignacionAsistenciasController.asignarProveedorTecnicoAsistencia(idAsistencia,
nombreProveedor, nombreTecnico);

    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertEquals("Proveedor y técnico asignados correctamente.",
response.getBody());

    verify(asistenciaRepository).save(asistencia);
  }

  @Test
  public void testAsignarProveedorTecnicoAsistenciaFailure() {
    // Datos de prueba
    Long idAsistencia = 1L;
    String nombreProveedor = "Proveedor A";
    String nombreTecnico = "Técnico B";

    when(asistenciaRepository.findById(idAsistencia)).thenReturn(Optional.empty());
```

```
when(proveedorRepository.findByNombre(nombreProveedor)).thenReturn(Optional.
empty());

when(tecnicoRepository.findByNombre(nombreTecnico)).thenReturn(Optional.empt
y());

    ResponseEntity<String> response =
asignacionAsistenciasController.asignarProveedorTecnicoAsistencia(idAsistencia,
nombreProveedor, nombreTecnico);

    assertEquals(HttpStatus.BAD_REQUEST, response.getStatusCode());
    assertEquals("Error al asignar proveedor y técnico. Verifique los datos
proporcionados.", response.getBody());

    verify(asistenciaRepository, never()).save(any(AsistenciaModel.class));
  }
}
```

pruebas unitarias para SolicitudAsistenciaController usando Mockito y JUnit 5

```
package com.example.controller.Asistencia;

import com.example.model.AsistenciaModel;
import com.example.service.AsistenciaService;
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 org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

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

public class SolicitudAsistenciaControllerTest {

  @Mock
  private AsistenciaService asistenciaService;

  @InjectMocks
  private SolicitudAsistenciaController solicitudAsistenciaController;

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

  @Test
  public void testSolicitarAsistenciaSuccess() {
    // Datos de prueba
    AsistenciaModel nuevaAsistencia = new AsistenciaModel();

    // Configura el mock para devolver la asistencia guardada

when(asistenciaService.saveAsistencia(any(AsistenciaModel.class))).thenReturn(nu
evaAsistencia);

    // Llama al método del controlador
    ResponseEntity<?> response =
```

```
solicitudAsistenciaController.solicitarAsistencia(nuevaAsistencia);

      // Verifica el resultado
      assertEquals(HttpStatus.OK, response.getStatusCode());
      SolicitudAsistenciaController.MensajeRespuesta mensajeRespuesta =
(SolicitudAsistenciaController.MensajeRespuesta) response.getBody();
      assertEquals("Asistencia solicitada exitosamente.",
mensajeRespuesta.getMessage());

      verify(asistenciaService).saveAsistencia(any(AsistenciaModel.class));
  }

  @Test
  public void testSolicitarAsistenciaFailure() {
      // Datos de prueba
      AsistenciaModel nuevaAsistencia = new AsistenciaModel();

      // Configura el mock para lanzar una excepción

when(asistenciaService.saveAsistencia(any(AsistenciaModel.class))).thenThrow(new
RuntimeException("Error"));

      // Llama al método del controlador
      ResponseEntity<?> response =
solicitudAsistenciaController.solicitarAsistencia(nuevaAsistencia);

      // Verifica el resultado
      assertEquals(HttpStatus.INTERNAL_SERVER_ERROR,
response.getStatusCode());
      SolicitudAsistenciaController.MensajeRespuesta mensajeRespuesta =
(SolicitudAsistenciaController.MensajeRespuesta) response.getBody();
      assertEquals("Ocurrió un error al solicitar la asistencia.",
mensajeRespuesta.getMessage());

      verify(asistenciaService).saveAsistencia(any(AsistenciaModel.class));
  }

  @Test
  public void testEliminarAsistenciaSuccess() {
      Long idAsistencia = 1L;

      // Llama al método del controlador
      ResponseEntity<?> response =
solicitudAsistenciaController.eliminarAsistencia(idAsistencia);
```

```
        // Verifica el resultado
        assertEquals(HttpStatus.OK, response.getStatusCode());
        SolicitudAsistenciaController.MensajeRespuesta mensajeRespuesta =
(SolicitudAsistenciaController.MensajeRespuesta) response.getBody();
        assertEquals("Asistencia eliminada exitosamente.",
mensajeRespuesta.getMessage());

        verify(asistenciaService).eliminarAsistenciaPorId(idAsistencia);
    }

    @Test
    public void testEliminarAsistenciaFailure() {
        Long idAsistencia = 1L;

        // Configura el mock para lanzar una excepción
        doThrow(new
RuntimeException("Error")).when(asistenciaService).eliminarAsistenciaPorId(idAsist
encia);

        // Llama al método del controlador
        ResponseEntity<?> response =
solicitudAsistenciaController.eliminarAsistencia(idAsistencia);

        // Verifica el resultado
        assertEquals(HttpStatus.INTERNAL_SERVER_ERROR,
response.getStatusCode());
        SolicitudAsistenciaController.MensajeRespuesta mensajeRespuesta =
(SolicitudAsistenciaController.MensajeRespuesta) response.getBody();
        assertEquals("Ocurrió un error al eliminar la asistencia.",
mensajeRespuesta.getMessage());

        verify(asistenciaService).eliminarAsistenciaPorId(idAsistencia);
    }
}
```

pruebas unitarias para DirectorioUsuarioController usando Mockito y JUnit 5

```
package com.example.controller.Buscar;

import com.example.model.UsuarioModel;
import com.example.service.DirectorioUsuarioService;
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 org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

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

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

public class DirectorioUsuarioControllerTest {

  @Mock
  private DirectorioUsuarioService directorioActivoService;

  @InjectMocks
  private DirectorioUsuarioController directorioUsuarioController;

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

  @Test
  public void testBuscarUsuarioSuccess() {
    Long idUsuario = 1L;
    UsuarioModel usuario = new UsuarioModel();
    usuario.setIdUsuario(idUsuario);
```

```
when(directorioActivoService.buscarPorId(idUsuario)).thenReturn(Optional.of(usuario));

    ResponseEntity<?> response = directorioUsuarioController.buscar(idUsuario);

    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertEquals(usuario, response.getBody());
  }

  @Test
  public void testBuscarUsuarioNotFound() {
    Long idUsuario = 1L;


when(directorioActivoService.buscarPorId(idUsuario)).thenReturn(Optional.empty());

    ResponseEntity<?> response = directorioUsuarioController.buscar(idUsuario);

    assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
    assertEquals("No se encontró ningún usuario con el ID: " + idUsuario, response.getBody());
  }

  @Test
  public void testBuscarUsuarioError() {
    Long idUsuario = 1L;

    when(directorioActivoService.buscarPorId(idUsuario)).thenThrow(new RuntimeException("Error"));

    ResponseEntity<?> response = directorioUsuarioController.buscar(idUsuario);

    assertEquals(HttpStatus.INTERNAL_SERVER_ERROR, response.getStatusCode());
    assertEquals("Ocurrió un error al buscar el usuario por ID.", response.getBody());
  }

  @Test
  public void testBuscarTodosUsuariosSuccess() {
    List<UsuarioModel> usuarios = Arrays.asList(new UsuarioModel(), new UsuarioModel());

    when(directorioActivoService.mostrarTodos()).thenReturn(usuarios);
```

```
    ResponseEntity<?> response = directorioUsuarioController.buscarTodos();

    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertEquals(usuarios, response.getBody());
  }

  @Test
  public void testBuscarTodosUsuariosError() {
    when(directorioActivoService.mostrarTodos()).thenThrow(new
RuntimeException("Error"));

    ResponseEntity<?> response = directorioUsuarioController.buscarTodos();

    assertEquals(HttpStatus.INTERNAL_SERVER_ERROR,
response.getStatusCode());
    assertEquals("Ocurrió un error al buscar usuarios. Por favor, inténtalo de nuevo.",
response.getBody());
  }
}
```

pruebas unitarias para CrearUsuarioController usando Mockito y JUnit 5

```
package com.example.controller.Usuario;

import com.example.model.UsuarioModel;
import com.example.repository.UsuarioRepository;
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 org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

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

public class CrearUsuarioControllerTest {

    @Mock
    private UsuarioRepository usuarioRepository;

    @InjectMocks
    private CrearUsuarioController crearUsuarioController;

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

    @Test
    public void testCrearUsuarioSuccess() {
        // Datos de prueba
        UsuarioModel nuevoUsuario = new UsuarioModel();

        // Configura el mock para devolver el usuario guardado
        when(usuarioRepository.save(nuevoUsuario)).thenReturn(nuevoUsuario);

        // Llama al método del controlador
        ResponseEntity<?> response =
crearUsuarioController.crearUsuario(nuevoUsuario);
```

```
        // Verifica el resultado
        assertEquals(HttpStatus.OK, response.getStatusCode());
        CrearUsuarioController.MensajeRespuesta mensajeRespuesta =
(CrearUsuarioController.MensajeRespuesta) response.getBody();
        assertEquals("Usuario creado exitosamente.", mensajeRespuesta.getMessage());

        verify(usuarioRepository).save(nuevoUsuario);
    }

    @Test
    public void testCrearUsuarioFailure() {
        // Datos de prueba
        UsuarioModel nuevoUsuario = new UsuarioModel();

        // Configura el mock para lanzar una excepción
        when(usuarioRepository.save(nuevoUsuario)).thenThrow(new
RuntimeException("Error"));

        // Llama al método del controlador
        ResponseEntity<?> response =
crearUsuarioController.crearUsuario(nuevoUsuario);

        // Verifica el resultado
        assertEquals(HttpStatus.INTERNAL_SERVER_ERROR,
response.getStatusCode());
        CrearUsuarioController.MensajeRespuesta mensajeRespuesta =
(CrearUsuarioController.MensajeRespuesta) response.getBody();
        assertEquals("Ocurrió un error al crear el usuario.",
mensajeRespuesta.getMessage());

        verify(usuarioRepository).save(nuevoUsuario);
    }
}
```

pruebas unitarias para EliminarUsuarioController usando Mockito y JUnit 5

```
package com.example.controller.Usuario;

import com.example.model.UsuarioModel;
import com.example.service.UsuarioService;
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 org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

import java.util.Optional;

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

public class EliminarUsuarioControllerTest {

  @Mock
  private UsuarioService usuarioService;

  @InjectMocks
  private EliminarUsuarioController eliminarUsuarioController;

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

  @Test
  public void testBuscarUsuarioIdSuccess() {
    Long idUsuario = 1L;
    UsuarioModel usuario = new UsuarioModel();
    usuario.setIdUsuario(idUsuario);


when(usuarioService.getUsuarioById(idUsuario)).thenReturn(Optional.of(usuario));

    ResponseEntity<?> response =
```

```
eliminarUsuarioController.buscarUsuarioId(idUsuario);

    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertEquals(usuario, response.getBody());
  }

  @Test
  public void testBuscarUsuarioIdNotFound() {
    Long idUsuario = 1L;

    when(usuarioService.getUsuarioById(idUsuario)).thenReturn(Optional.empty());

    ResponseEntity<?> response =
eliminarUsuarioController.buscarUsuarioId(idUsuario);

    assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
  }

  @Test
  public void testBuscarUsuarioIdError() {
    Long idUsuario = 1L;

    when(usuarioService.getUsuarioById(idUsuario)).thenThrow(new
RuntimeException("Error"));

    ResponseEntity<?> response =
eliminarUsuarioController.buscarUsuarioId(idUsuario);

    assertEquals(HttpStatus.INTERNAL_SERVER_ERROR,
response.getStatusCode());
    EliminarUsuarioController.MensajeRespuesta mensajeRespuesta =
(EliminarUsuarioController.MensajeRespuesta) response.getBody();
    assertEquals("Ocurrió un error al buscar el usuario por ID.",
mensajeRespuesta.getMensaje());
  }

  @Test
  public void testEliminarUsuarioSuccess() {
    Long idUsuario = 1L;

    when(usuarioService.existsById(idUsuario)).thenReturn(true);

    ResponseEntity<String> response =
eliminarUsuarioController.eliminarUsuario(idUsuario);
```

```
        assertEquals(HttpStatus.OK, response.getStatusCode());
        assertEquals("Usuario eliminado exitosamente", response.getBody());

        verify(usuarioService).deleteUsuario(idUsuario);
    }

    @Test
    public void testEliminarUsuarioNotFound() {
        Long idUsuario = 1L;

        when(usuarioService.existsById(idUsuario)).thenReturn(false);

        ResponseEntity<String> response =
eliminarUsuarioController.eliminarUsuario(idUsuario);

        assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
        assertEquals("Usuario no encontrado", response.getBody());
    }

    @Test
    public void testEliminarUsuarioError() {
        Long idUsuario = 1L;

        when(usuarioService.existsById(idUsuario)).thenReturn(true);
        doThrow(new
RuntimeException("Error")).when(usuarioService).deleteUsuario(idUsuario);

        ResponseEntity<String> response =
eliminarUsuarioController.eliminarUsuario(idUsuario);

        assertEquals(HttpStatus.INTERNAL_SERVER_ERROR,
response.getStatusCode());
        assertEquals("Error al eliminar el usuario", response.getBody());
    }
}
```

pruebas unitarias para modificarUsuarioControllerTestusando Mockito y JUnit 5

```
package com.example.controller.Usuario;

import com.example.model.UsuarioModel;
import com.example.service.UsuarioService;
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 org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

import java.util.Optional;

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

public class ModificarUsuarioControllerTest {

  @Mock
  private UsuarioService usuarioService;

  @InjectMocks
  private ModificarUsuarioController modificarUsuarioController;

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

  @Test
  public void testBuscarUsuarioPorIdSuccess() {
    Long idUsuario = 1L;
    UsuarioModel usuario = new UsuarioModel();
    usuario.setIdUsuario(idUsuario);

    when(usuarioService.findById(idUsuario)).thenReturn(Optional.of(usuario));

    ResponseEntity<?> response =
modificarUsuarioController.buscarUsuarioPorId(idUsuario);
```

```
    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertEquals(usuario, response.getBody());
  }

  @Test
  public void testBuscarUsuarioPorIdNotFound() {
    Long idUsuario = 1L;

    when(usuarioService.findById(idUsuario)).thenReturn(Optional.empty());

    ResponseEntity<?> response =
modificarUsuarioController.buscarUsuarioPorId(idUsuario);

    assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
  }

  @Test
  public void testBuscarUsuarioPorIdError() {
    Long idUsuario = 1L;

    when(usuarioService.findById(idUsuario)).thenThrow(new
RuntimeException("Error"));

    ResponseEntity<?> response =
modificarUsuarioController.buscarUsuarioPorId(idUsuario);

    assertEquals(HttpStatus.INTERNAL_SERVER_ERROR,
response.getStatusCode());
    ModificarUsuarioController.MensajeRespuesta mensajeRespuesta =
(ModificarUsuarioController.MensajeRespuesta) response.getBody();
    assertEquals("Ocurrió un error al buscar el usuario por ID.",
mensajeRespuesta.getMessage());
  }

  @Test
  public void testModificarUsuarioSuccess() {
    Long idUsuario = 1L;
    UsuarioModel usuarioExistente = new UsuarioModel();
    usuarioExistente.setIdUsuario(idUsuario);

    UsuarioModel usuarioActualizado = new UsuarioModel();
    usuarioActualizado.setNombre("Nuevo Nombre");
    usuarioActualizado.setTelefono("1234567890");
```

```
    usuarioActualizado.setCorreoElectronico("nuevo@email.com");
    usuarioActualizado.setDireccion("Nueva Dirección");
    usuarioActualizado.setTipoDeAsistencia("Nuevo Tipo");
    usuarioActualizado.setClave("Nueva Clave");


when(usuarioService.findById(idUsuario)).thenReturn(Optional.of(usuarioExistente));

when(usuarioService.saveUsuario(usuarioExistente)).thenReturn(usuarioExistente);

    ResponseEntity<?> response =
modificarUsuarioController.modificarUsuario(idUsuario, usuarioActualizado);

    assertEquals(HttpStatus.OK, response.getStatusCode());
    ModificarUsuarioController.MensajeRespuesta mensajeRespuesta =
(ModificarUsuarioController.MensajeRespuesta) response.getBody();
    assertEquals("Usuario modificado exitosamente.",
mensajeRespuesta.getMessage());
  }

  @Test
  public void testModificarUsuarioNotFound() {
    Long idUsuario = 1L;
    UsuarioModel usuarioActualizado = new UsuarioModel();

    when(usuarioService.findById(idUsuario)).thenReturn(Optional.empty());

    ResponseEntity<?> response =
modificarUsuarioController.modificarUsuario(idUsuario, usuarioActualizado);

    assertEquals(HttpStatus.NOT_FOUND, response.getStatusCode());
  }

  @Test
  public void testModificarUsuarioError() {
    Long idUsuario = 1L;
    UsuarioModel usuarioActualizado = new UsuarioModel();

    when(usuarioService.findById(idUsuario)).thenReturn(Optional.of(new
UsuarioModel()));
    when(usuarioService.saveUsuario(any())).thenThrow(new
RuntimeException("Error"));

    ResponseEntity<?> response =
```

```
modificarUsuarioController.modificarUsuario(idUsuario, usuarioActualizado);

      assertEquals(HttpStatus.INTERNAL_SERVER_ERROR,
response.getStatusCode());
      ModificarUsuarioController.MensajeRespuesta mensajeRespuesta =
(ModificarUsuarioController.MensajeRespuesta) response.getBody();
      assertEquals("Ocurrió un error al modificar el usuario.",
mensajeRespuesta.getMessage());
   }
}
```