Sapresis

ACTA DE APLICACIÓN DE PRUEBAS Y ACEPTACIÓN

Evidencia: GA8-220501096-AA1-EV02

ABSTRACT

Se recopilan la información y los detalles de los resultados de las pruebas unitarias realizadas para la aplicación Sapresis.

Mauricio Alberto Monroy Calle

Tecnología en Análisis y Desarrollo de Software - 2721455

SENA

3/10/2024

CODELICHT
SOFTWARE SOLUTIONS

Contenido

1 Introducción

1.1 Descripción general de la aplicación

El Sapresis (Sistema Integral para la Prestación de Servicios de Salud) es un proyecto académico desarrollado para la Tecnología de Análisis y Desarrollo de Software (2721455) del SENA. Este sistema está diseñado para ser utilizado por el personal de una Institución Prestadora de Salud (IPS). El sistema se compone de varias entidades que permiten consultar y gestionar información básica sobre los empleados de la institución, como el número de identificación, el nombre y los datos de contacto. Además, permite consultar la asignación de consultorios, consultas y pacientes. También se puede almacenar información detallada sobre los pacientes, como su identificación, el nombre, los datos de contacto, su afiliación a la prestación de salud (EPS), el personal asignado, las fórmulas médicas y las facturas.

1.2 Propósito del documento

El presente documento tiene como objetivo documentar las pruebas unitarias llevadas a cabo en la aplicación Sapresis, desarrollada con una arquitectura de servicios basada en Java y Spring Boot para el backend, y React para el frontend.

Las pruebas unitarias se centrarán en validar la lógica de negocio de los módulos de autenticación (inicio de sesión y registro) y en las operaciones CRUD sobre las entidades que conforman el núcleo del sistema.

La metodología de prueba consistirá en diseñar casos de prueba que cubran los escenarios más comunes y críticos de cada funcionalidad, utilizando herramientas como JUnit y Mockito. Se priorizarán las pruebas de creación de nuevos registros para cada entidad, lo que permitirá asegurar la integridad de los datos y la correcta interacción con la base de datos. También se llevarán a cabo pruebas de recuperación y eliminación para verificar la consistencia de los datos y la gestión de las referencias entre entidades.

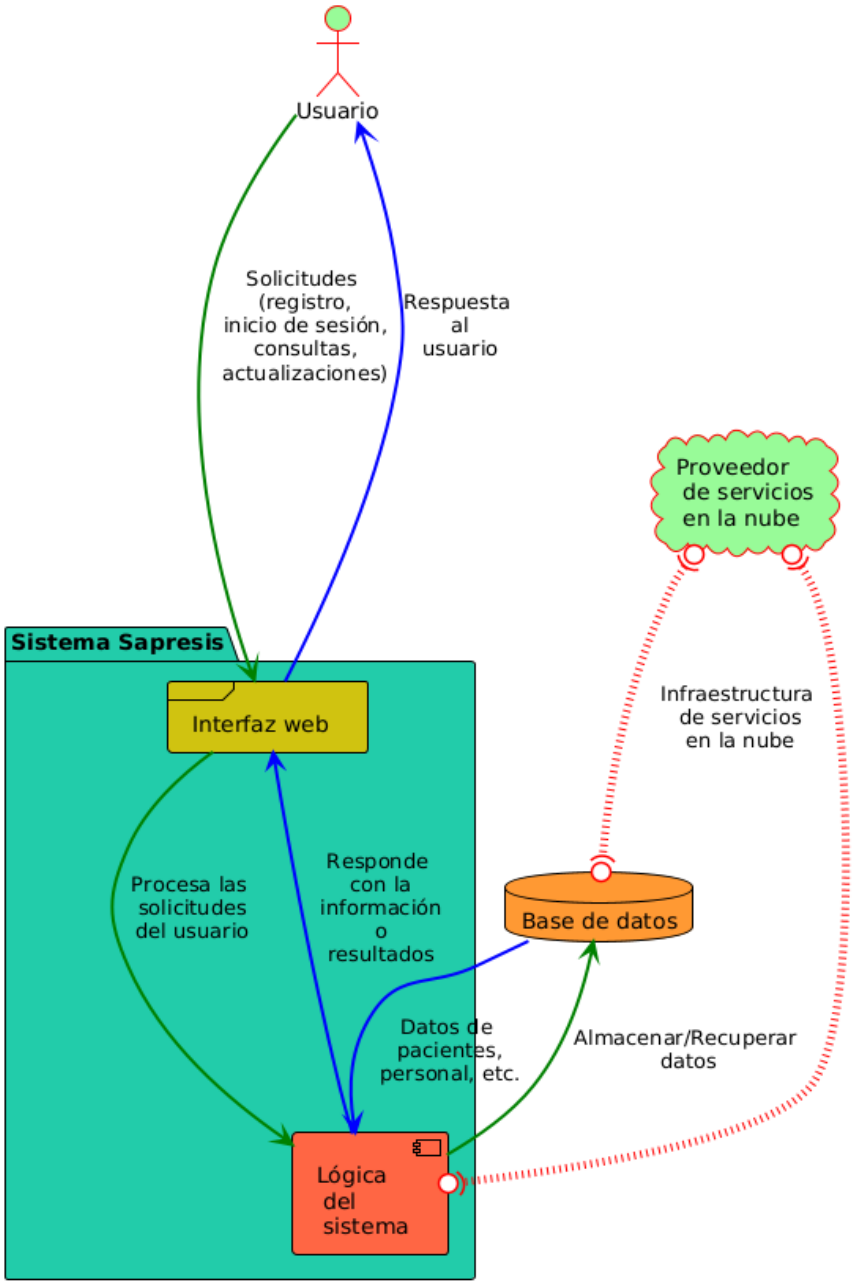
Esta primera fase de pruebas unitarias es fundamental para garantizar la calidad del código y detectar posibles errores antes de pasar a etapas más avanzadas de desarrollo y testing.

1.3 Alcance del documento

Este documento está dirigido a todas las partes interesadas en la implementación y uso de Sapresis, incluyendo desarrolladores, administradores del sistema, personal médico y directivos de la institución. Proporciona una base para la formación, referencia y mejora continua del sistema.

2 Diagrama de contexto

Antes de presentar la descripción y el resultado de las pruebas realizadas, es necesario contextualizar el funcionamiento general de la aplicación mediante un diagrama de contexto. Este diagrama ofrece una visión general de alto nivel de las interacciones entre los actores principales y el sistema Sapresis, y destaca los flujos de datos más importantes.



En el diagrama, los actores principales son:

- **Usuario**: persona que interactúa con la aplicación a través de la interfaz web. Los usuarios pueden ser personal sanitario o administrativo, y su rol determina el acceso y las operaciones que pueden realizar (registro de pacientes, consultas, actualizaciones, etc.).
- **Base de datos:** Almacena la información clave del sistema, como los datos de los pacientes, el personal médico, las instituciones y otras entidades relacionadas con la gestión hospitalaria. El sistema accede a esta base de datos para almacenar y recuperar información.
- **Proveedor de servicios en la nube:** facilita la infraestructura necesaria para el funcionamiento del sistema, ofreciendo servicios como almacenamiento en la nube, procesamiento de datos y seguridad.

El sistema Sapresis actúa como intermediario entre el usuario y la base de datos. A través de la interfaz web, el usuario envía solicitudes (como registrar o consultar datos), y el sistema procesa estas solicitudes, interactuando con la base de datos para gestionar la información y devolver las respuestas adecuadas. El proveedor de servicios en la nube apoya la infraestructura técnica para asegurar la continuidad y eficiencia de estos procesos.

Este diagrama de contexto simplifica la comprensión del flujo de información dentro del sistema y ayuda a identificar las principales interacciones que se evaluarán a través de las pruebas unitarias descritas en las secciones posteriores.

3 Desarrollo de las pruebas

En este apartado y en los siguientes, se describe el proceso de pruebas realizado sobre la aplicación Sapresis. Las pruebas se han centrado en dos áreas clave: la autenticación de usuarios y la lógica del sistema. Esto abarca tanto el registro como el acceso a la aplicación y la validación de la capa de servicio que gestiona las operaciones de negocio y las transacciones CRUD (crear, leer, actualizar y eliminar) para las entidades del sistema.

Estas pruebas tienen como objetivo principal garantizar que la lógica implementada en Sapresis funcione de manera óptima, asegurando no solo el rendimiento, sino también la consistencia y la correcta ejecución de las operaciones fundamentales. Además, ofrecen evidencia de una sólida estructura base en la arquitectura de la aplicación, la cual es esencial para su estabilidad y escalabilidad.

A lo largo del proceso de desarrollo, estas pruebas unitarias han sido esenciales para verificar que los componentes individuales del sistema funcionen como se espera. En el futuro, se prevé la inclusión de pruebas de integración para asegurar la comunicación fluida entre los diferentes módulos y capas del sistema. También se considera la implementación de pruebas de carga, cuyo fin será evaluar la capacidad de la aplicación para manejar grandes volúmenes de datos y solicitudes simultáneas, y garantizar su estabilidad en escenarios de alta demanda.

Este enfoque de pruebas permitirá una evolución continua de Sapresis, asegurando la calidad técnica y la robustez del sistema en un entorno real.

3.1 Pruebas unitarias para la autenticación y el registro de usuarios

Antes de mostrar el detalle de las pruebas, es necesario describir el proceso relacionado con la autenticación y el registro de usuarios en la aplicación *Sapresis*. La autenticación es un componente crítico para garantizar la seguridad y el control de acceso dentro del sistema. En *Sapresis*, este proceso se implementa utilizando un enfoque basado en roles, gestionado mediante la tecnología JWT (JSON Web Token), que genera "bearer tokens" únicos. Estos tokens se

incluyen en los encabezados de las peticiones HTTP y tienen una duración de una hora, tras la cual expiran y requieren renovación para mantener el acceso.

El sistema maneja tres tipos de roles: **USER**, **ADMIN** y **SUPERADMIN**. Estos roles se preconfiguran y se crean al iniciar la aplicación por primera vez. Posteriormente, el usuario con el rol **SUPERADMIN** es creado, con acceso completo a todas las funcionalidades de *Sapresis*. A los usuarios nuevos que se registran se les asigna automáticamente el rol **USER**, que tiene permisos limitados a operaciones de solo lectura. Para obtener permisos de edición y gestión, es necesario contar con el rol **ADMIN**, que solo puede ser asignado por un **SUPERADMIN**. Este último tiene la capacidad exclusiva de gestionar los registros de usuarios, modificar sus credenciales o eliminarlos del sistema.

Las pruebas de autenticación y registro se centran en validar la correcta implementación de esta lógica a nivel del controlador. En este nivel, se comprueba que las solicitudes HTTP para el inicio de sesión y el registro de usuarios se gestionen adecuadamente y que las respuestas, como el token JWT y el manejo de roles, se generen de manera correcta.

La prueba de autenticación de usuarios se realizó simulando el proceso de inicio de sesión a través del controlador, verificando que se genere un token JWT válido para las credenciales correctas y que la autenticación falle con las credenciales incorrectas.

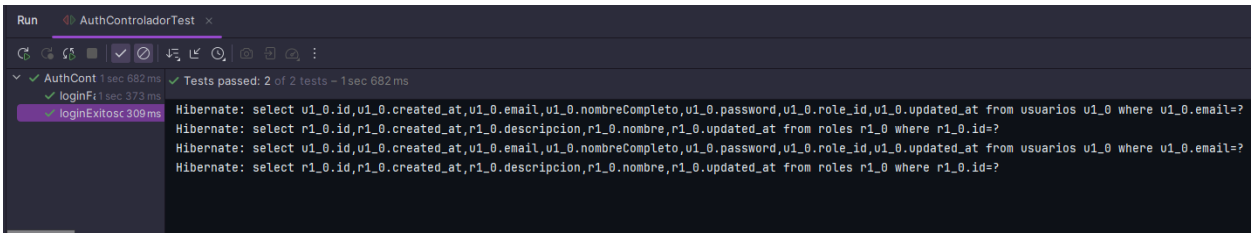
El código utilizado para la prueba de autenticación fue el siguiente:

```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class AuthControladorTest {

    @Autowired
    private MockMvc mockMvc;

    @Setter
    @Getter
    @Autowired
    private JwtService jwtService;

    @Test
    public void loginExitosoTest() throws Exception {
        // Implementar el test para el endpoint de autenticación con
credenciales correctas
        String loginRequest =
```

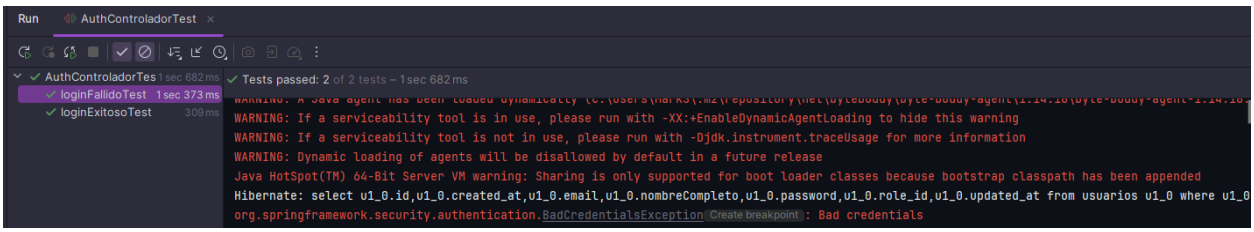
```
                "{ \"email\": \"super.admin@correo.com\", " +
                        "\"password\": \"G)T,T_Yr8]c6:YM\" }";

        mockMvc.perform(post("http://localhost:8080/sapresis/auth/login")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(loginRequest))
                .andExpect(status()
                        .isOk())
                .andExpect(jsonPath("$.token")
                        .exists());
    }

    @Test
    public void loginFallidoTest() throws Exception {
        // Implementar el test para el endpoint de autenticación con
credenciales incorrectas
        String loginRequest =
                "{ \"email\": \"user@correo.com\", " +
                        "\"password\": \"wrongPassword\" }";

        mockMvc.perform(post("http://localhost:8080/sapresis/auth/login")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(loginRequest))
                .andExpect(status()
                        .isUnauthorized());
    }

}
```

El resultado de la prueba fue satisfactorio, como se muestra a continuación:

Autenticación exitosa



Autenticación con credenciales incorrectas



En cuanto al registro de usuarios, la prueba verificó que un usuario con rol USER pudiera registrarse correctamente en el sistema y que, al intentar registrarse con datos incompletos o inválidos, el sistema devolviera los errores correspondientes. Esta prueba garantiza que el controlador gestione adecuadamente las validaciones y restricciones establecidas.

El código utilizado para la prueba de registro de usuarios fue el siguiente:

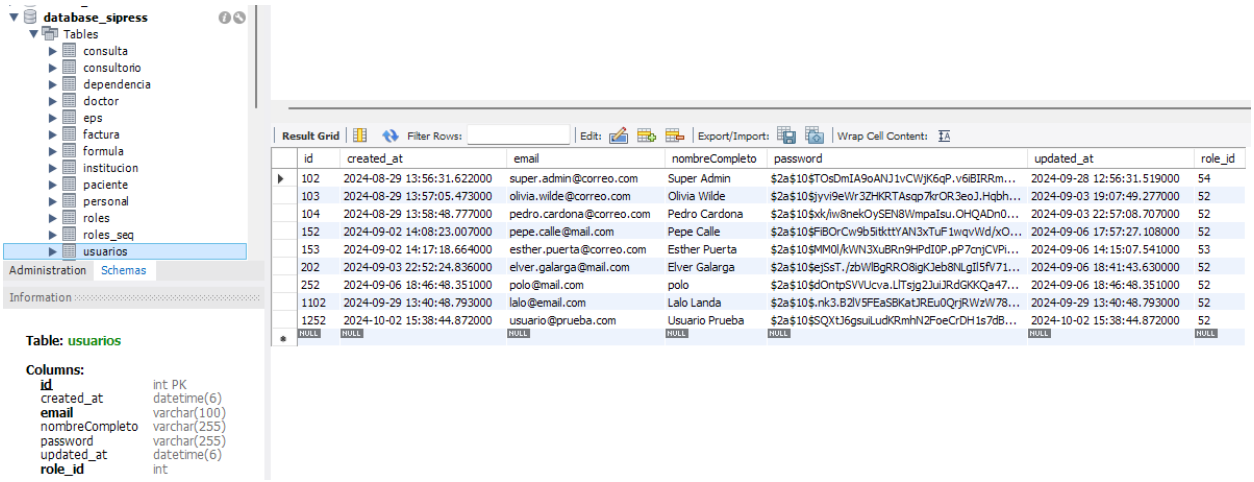
```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class UsuarioControladorTest {

    @Autowired
    private MockMvc mockMvc;

    @Setter
    @Getter
    @Autowired
    private JwtService jwtService;

    @Test
    public void registroExitosoTest() throws Exception {
        // Implementar el test para el endpoint de registro de usuarios
        String registrationRequest =
                "{ \"email\": \"usuario@prueba.com\", " +
                        "\"password\": \"password\", " +
                        "\"nombreCompleto\": \"Usuario Prueba\" }";
        mockMvc.perform(post(
                        "http://localhost:8080/sapresis/auth/registro")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(registrationRequest))
                .andExpect(status()
                        .isCreated())
                .andExpect(jsonPath("$.message")
                        .value("Usuario registrado exitosamente"));
    }

    @Test
    public void registroFallidoTest() throws Exception {
        // Implementar el test para el endpoint de registro de usuarios
        String registrationRequest =
                "{ \"email\": \"\", " +
                        "\"password\": \"\", " +
                        "\"nombreCompleto\": \"\" }";
        mockMvc.perform(post(
                        "http://localhost:8080/sapresis/auth/registro")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(registrationRequest))
                .andExpect(status()
                        .isBadRequest());
    }
}
```

El resultado de esta prueba se puede observar a continuación:

Registro exitoso

Registro fallido

De igual forma, aunque no es lo recomendado en este tipo de pruebas, se hizo uso de la base de datos real para guardar el usuario registrado:



3.2 Pruebas unitarias de la lógica del sistema

En este apartado se describe el proceso de pruebas relacionado con la lógica central del sistema, que abarca las operaciones fundamentales de creación, lectura, actualización y eliminación de datos (CRUD). Estas pruebas se han realizado en la capa de servicio, que es la base esencial para la gestión de las transacciones y la lógica de negocio en Sapresis. Aunque se podría haber optado por pruebas a nivel de dominio o de controlador, las pruebas de servicios ofrecen una visión más completa y efectiva sobre la interacción entre la aplicación y la base de datos, que es un punto neurálgico en la arquitectura.

Cada una de las entidades clave del sistema tiene su propia capa de servicio dedicada, que actúa como intermediaria entre los controladores y los repositorios de datos. Las entidades evaluadas en las pruebas incluyen: Consulta, Consultorio, Dependencia, Doctor, Eps, Factura, Fórmula, Institución, Paciente y Personal. Estas entidades están vinculadas a un repositorio JPA (Java Persistence API) que facilita las operaciones de persistencia en la base de datos.

El proceso de prueba consiste en validar que cada método de servicio implemente correctamente las operaciones CRUD. Para ello, se instancian los repositorios en las clases de servicio correspondientes mediante inyección de dependencias, lo que asegura que la lógica de negocio funcione de manera fluida y sin errores. Este servicio es crucial para que los controladores, que se encargan de manejar las solicitudes HTTP, puedan interactuar con la base de datos de manera eficiente y controlada.

Entre las operaciones verificadas en estas pruebas se encuentran:

- **Listar registros**: se comprueba que el servicio pueda recuperar correctamente todos los registros de la base de datos.
- **Buscar por ID**: se asegura que el servicio pueda encontrar y devolver un registro específico mediante su identificador único.
- **Guardar un nuevo registro**: se comprueba que el servicio pueda insertar correctamente un nuevo registro en la base de datos.
- **Eliminar registros**: se valida que el servicio pueda eliminar registros existentes de manera efectiva.

Cada una de estas pruebas se escribió simulando el comportamiento de las operaciones reales a través de un mock de los repositorios JPA, lo que permite verificar el funcionamiento del servicio sin necesidad de acceder a la base de datos en sí misma. Esto garantiza que el servicio funcione correctamente en condiciones controladas y reproduce el comportamiento esperado durante las operaciones CRUD.

A continuación, se muestra el código utilizado para las pruebas de cada una de las entidades. Este código incluye casos de prueba para cada operación CRUD, y comprueba que el servicio funcione según lo esperado.

3.2.1 Código de la prueba `ConsultaServicioTest`

```
class ConsultaServicioTest {

    @Mock
    private ConsultaRepositorio consultaRepositorio;  // Simula el
repositorio de Consulta

    @InjectMocks
    private ConsultaServicio consultaServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }

    // Prueba para recuperar la lista de registros de Consulta
    @Test
    void testListarConsultas() {
        Consulta consulta1 = new Consulta();
        Consulta consulta2 = new Consulta();

when(consultaRepositorio.findAll()).thenReturn(Arrays.asList(consulta1,
consulta2));

        List<Consulta> consultas = consultaServicio.listarConsultas();

        assertEquals(2, consultas.size());
        verify(consultaRepositorio, times(1)).findAll();
    }

    // Prueba para buscar una Consulta por su ID de Paciente
    @Test
    void testBuscarConsultaPorIdPaciente() {
        Consulta consulta = new Consulta();

when(consultaRepositorio.findByConsultaPK_PacienteId(1)).thenReturn(List.of(c
onsulta));
```

```
        Consulta resultado =
consultaServicio.buscarConsultaPorIdPaciente(1).get(0);

        assertNotNull(resultado);
        verify(consultaRepositorio, times(1)).findByConsultaPK_PacienteId(1);
    }

    // Prueba para buscar una Consulta por su ID de Doctor
    @Test
    void testBuscarConsultaPorIdDoctor() {
        Consulta consulta = new Consulta();

when(consultaRepositorio.findByConsultaPK_DoctorId(1)).thenReturn(List.of(con
sulta));

        Consulta resultado =
consultaServicio.buscarConsultaPorIdDoctor(1).get(0);

        assertNotNull(resultado);
        verify(consultaRepositorio, times(1)).findByConsultaPK_DoctorId(1);
    }

    // Prueba para guardar una Consulta
    @Test
    void testGuardarConsulta() {
        Consulta consulta = new Consulta();
        when(consultaRepositorio.save(consulta)).thenReturn(consulta);

        Consulta resultado = consultaServicio.guardarConsulta(consulta);

        assertEquals(consulta, resultado);
        verify(consultaRepositorio, times(1)).save(consulta);
    }

    // Prueba para eliminar una Consulta
    @Test
    void testEliminarConsulta() {
        Consulta consulta = new Consulta();

        consultaServicio.eliminarConsulta(consulta);

        verify(consultaRepositorio, times(1)).delete(consulta);
    }
}
```

3.2.2 Código de la prueba `ConsultorioServicioTest`

```
class ConsultorioServicioTest {

    @Mock
    private ConsultorioRepositorio consultorioRepositorio;  // Simula el
repositorio de Consultorio
```

```
    @InjectMocks
    private ConsultorioServicio consultorioServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }

    // Prueba para recuperar la lista de registros de Consultorio
    @Test
    public void testListarConsultorios() {
        Consultorio consultorio1 = new Consultorio();
        Consultorio consultorio2 = new Consultorio();

when(consultorioRepositorio.findAll()).thenReturn(Arrays.asList(consultorio1,
consultorio2));

        List<Consultorio> consultorios =
consultorioServicio.listarConsultorios();

        assertEquals(2, consultorios.size());
        verify(consultorioRepositorio, times(1)).findAll();
    }

    // Prueba para buscar un Consultorio por su ID
    @Test
    public void testBuscarConsultorioPorId() {
        Consultorio consultorio = new Consultorio();

when(consultorioRepositorio.findById(1)).thenReturn(Optional.of(consultorio))
;

        Consultorio resultado =
consultorioServicio.buscarConsultorioPorId(1);

        assertNotNull(resultado);
        verify(consultorioRepositorio, times(1)).findById(1);
    }

    // Prueba para guardar un Consultorio
    @Test
    public void testGuardarConsultorio() {
        Consultorio consultorio = new Consultorio();

when(consultorioRepositorio.save(consultorio)).thenReturn(consultorio);

        Consultorio resultado =
consultorioServicio.guardarConsultorio(consultorio);

        assertEquals(consultorio, resultado);
        verify(consultorioRepositorio, times(1)).save(consultorio);
    }

    // Prueba para eliminar un Consultorio
    @Test
    public void testEliminarConsultorio() {
```

```
        Consultorio consultorio = new Consultorio();

        consultorioServicio.eliminarConsultorio(consultorio);

        verify(consultorioRepositorio, times(1)).delete(consultorio);
    }
}
```

3.2.3 Código de la prueba `DependenciaServicioTest`

```
class DependenciaServicioTest {

    @Mock
    private DependenciaRepositorio dependenciaRepositorio;  // Simula el
repositorio de Dependencia

    @InjectMocks
    private DependenciaServicio dependenciaServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }

    // Prueba para recuperar la lista de registros de Dependencia
    @Test
    public void testListarDependencias() {
        Dependencia dependencia1 = new Dependencia();
        Dependencia dependencia2 = new Dependencia();

when(dependenciaRepositorio.findAll()).thenReturn(Arrays.asList(dependencia1,
dependencia2));

        List<Dependencia> dependencias =
dependenciaServicio.listarDependencias();

        assertEquals(2, dependencias.size());
        verify(dependenciaRepositorio, times(1)).findAll();
    }

    // Prueba para buscar una Dependencia por su ID
    @Test
    public void testBuscarDependenciaPorId() {
        Dependencia dependencia = new Dependencia();

when(dependenciaRepositorio.findById(1)).thenReturn(Optional.of(dependencia))
;

        Dependencia resultado =
dependenciaServicio.buscarDependenciaPorId(1);

        assertNotNull(resultado);
        verify(dependenciaRepositorio, times(1)).findById(1);
```

```
    }

    // Prueba para guardar una Dependencia
    @Test
    public void testGuardarDependencia() {
        Dependencia dependencia = new Dependencia();

when(dependenciaRepositorio.save(dependencia)).thenReturn(dependencia);

        Dependencia resultado =
dependenciaServicio.guardarDependencia(dependencia);

        assertEquals(dependencia, resultado);
        verify(dependenciaRepositorio, times(1)).save(dependencia);
    }

    // Prueba para eliminar una Dependencia
    @Test
    public void testEliminarDependencia() {
        Dependencia dependencia = new Dependencia();

        dependenciaServicio.eliminarDependencia(dependencia);

        verify(dependenciaRepositorio, times(1)).delete(dependencia);
    }
}
```

3.2.4 Código de la prueba `DoctorServicioTest`

```
public class DoctorServicioTest {

    @Mock
    private DoctorRepositorio doctorRepositorio;  // Simula el repositorio de
Doctor

    @InjectMocks
    private DoctorServicio doctorServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }

    // Prueba para recuperar la lista de registros de Doctor
    @Test
    public void testListarDoctores() {
        Doctor doctor1 = new Doctor();
        Doctor doctor2 = new Doctor();
        when(doctorRepositorio.findAll()).thenReturn(Arrays.asList(doctor1,
doctor2));

        List<Doctor> doctores = doctorServicio.listarDoctores();
```

```
        assertEquals(2, doctores.size());
        verify(doctorRepositorio, times(1)).findAll();
    }

    // Prueba para buscar un Doctor por su ID
    @Test
    public void testBuscarDoctorPorId() {
        Doctor doctor = new Doctor();
        when(doctorRepositorio.findById(1)).thenReturn(Optional.of(doctor));

        Doctor resultado = doctorServicio.buscarDoctorPorId(1);

        assertNotNull(resultado);
        verify(doctorRepositorio, times(1)).findById(1);
    }

    // Prueba para guardar un Doctor
    @Test
    public void testGuardarDoctor() {
        Doctor doctor = new Doctor();
        when(doctorRepositorio.save(doctor)).thenReturn(doctor);

        Doctor resultado = doctorServicio.guardarDoctor(doctor);

        assertEquals(doctor, resultado);
        verify(doctorRepositorio, times(1)).save(doctor);
    }

    // Prueba para eliminar un Doctor
    @Test
    public void testEliminarDoctor() {
        Doctor doctor = new Doctor();

        doctorServicio.eliminarDoctor(doctor);

        verify(doctorRepositorio, times(1)).delete(doctor);
    }
}
```

3.2.5 Código de la prueba `EpsServicioTest`

```
class EpsServicioTest {

    @Mock
    private EpsRepositorio epsRepositorio;  // Simula el repositorio de Eps

    @InjectMocks
    private EpsServicio epsServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }
```

```
    // Prueba para recuperar la lista de registros de Eps
    @Test
    public void testListarEpsS() {
        Eps eps1 = new Eps();
        Eps eps2 = new Eps();
        when(epsRepositorio.findAll()).thenReturn(Arrays.asList(eps1, eps2));

        List<Eps> epsS = epsServicio.listarEpsS();

        assertEquals(2, epsS.size());
        verify(epsRepositorio, times(1)).findAll();
    }

    // Prueba para buscar una Eps por su ID
    @Test
    public void testBuscarEpsPorId() {
        Eps eps = new Eps();
        when(epsRepositorio.findById(1)).thenReturn(Optional.of(eps));

        Eps resultado = epsServicio.buscarEpsPorId(1);

        assertNotNull(resultado);
        verify(epsRepositorio, times(1)).findById(1);
    }

    // Prueba para guardar una Eps
    @Test
    public void testGuardarEps() {
        Eps eps = new Eps();
        when(epsRepositorio.save(eps)).thenReturn(eps);

        Eps resultado = epsServicio.guardarEps(eps);

        assertEquals(eps, resultado);
        verify(epsRepositorio, times(1)).save(eps);
    }

    // Prueba para eliminar una Eps
    @Test
    public void testEliminarEps() {
        Eps eps = new Eps();

        epsServicio.eliminarEps(eps);

        verify(epsRepositorio, times(1)).delete(eps);
    }
}
```

3.2.6 Código de la prueba `FacturaServicioTest`

```
class FacturaServicioTest {

    @Mock
```

```
    private FacturaRepositorio facturaRepositorio;  // Simula el repositorio
de Factura

    @InjectMocks
    private FacturaServicio facturaServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }

    // Prueba para recuperar la lista de registros de Factura
    @Test
    public void testListarFacturas() {
        Factura factura1 = new Factura();
        Factura factura2 = new Factura();
        when(facturaRepositorio.findAll()).thenReturn(Arrays.asList(factura1,
factura2));

        List<Factura> facturas = facturaServicio.listarFacturas();

        assertEquals(2, facturas.size());
        verify(facturaRepositorio, times(1)).findAll();
    }

    // Prueba para buscar una Factura por su ID
    @Test
    public void testBuscarFacturaPorId() {
        Factura factura = new Factura();

when(facturaRepositorio.findById(1)).thenReturn(Optional.of(factura));

        Factura resultado = facturaServicio.buscarFacturaPorId(1);

        assertNotNull(resultado);
        verify(facturaRepositorio, times(1)).findById(1);
    }

    // Prueba para guardar una Factura
    @Test
    public void testGuardarFactura() {
        Factura factura = new Factura();
        when(facturaRepositorio.save(factura)).thenReturn(factura);

        Factura resultado = facturaServicio.guardarFactura(factura);

        assertEquals(factura, resultado);
        verify(facturaRepositorio, times(1)).save(factura);
    }

    // Prueba para eliminar una Factura
    @Test
    public void testEliminarFactura() {
        Factura factura = new Factura();

        facturaServicio.eliminarFactura(factura);
```

```
        verify(facturaRepositorio, times(1)).delete(factura);
    }
}
```

3.2.7 Código de la prueba `FormulaServicioTest`

```
class FormulaServicioTest {

    @Mock
    private FormulaRepositorio formulaRepositorio;  // Simula el repositorio
de Formula

    @InjectMocks
    private FormulaServicio formulaServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }

    // Prueba para recuperar la lista de registros de Formula
    @Test
    public void testListarFormulas() {
        Formula formula1 = new Formula();
        Formula formula2 = new Formula();
        when(formulaRepositorio.findAll()).thenReturn(Arrays.asList(formula1,
formula2));

        List<Formula> formulas = formulaServicio.listarFormulas();

        assertEquals(2, formulas.size());
        verify(formulaRepositorio, times(1)).findAll();
    }

    // Prueba para buscar un Formula por su ID
    @Test
    public void testBuscarFormulaPorId() {
        Formula formula = new Formula();

when(formulaRepositorio.findById(1)).thenReturn(Optional.of(formula));

        Formula resultado = formulaServicio.buscarFormulaPorId(1);

        assertNotNull(resultado);
        verify(formulaRepositorio, times(1)).findById(1);
    }

    // Prueba para guardar un Formula
    @Test
    public void testGuardarFormula() {
        Formula formula = new Formula();
        when(formulaRepositorio.save(formula)).thenReturn(formula);
```

```
        Formula resultado = formulaServicio.guardarFormula(formula);

        assertEquals(formula, resultado);
        verify(formulaRepositorio, times(1)).save(formula);
    }

    // Prueba para eliminar un Formula
    @Test
    public void testEliminarFormula() {
        Formula formula = new Formula();

        formulaServicio.eliminarFormula(formula);

        verify(formulaRepositorio, times(1)).delete(formula);
    }
}
```

3.2.8 Código de la prueba `InstitucionServicioTest`

```
class InstitucionServicioTest {

    @Mock
    private InstitucionRepositorio institucionRepositorio;  // Simula el
repositorio de Institucion

    @InjectMocks
    private InstitucionServicio institucionServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }

    // Prueba para recuperar la lista de registros de Institucion
    @Test
    public void testListarInstituciones() {
        Institucion institucion1 = new Institucion();
        Institucion institucion2 = new Institucion();

when(institucionRepositorio.findAll()).thenReturn(Arrays.asList(institucion1,
institucion2));

        List<Institucion> instituciones =
institucionServicio.listarInstituciones();

        assertEquals(2, instituciones.size());
        verify(institucionRepositorio, times(1)).findAll();
    }

    // Prueba para buscar un Institucion por su ID
    @Test
    public void testBuscarInstitucionPorId() {
```

```
        Institucion institucion = new Institucion();

when(institucionRepositorio.findById(1)).thenReturn(Optional.of(institucion))
;

        Institucion resultado =
institucionServicio.buscarInstitucionPorId(1);

        assertNotNull(resultado);
        verify(institucionRepositorio, times(1)).findById(1);
    }

    // Prueba para guardar un Institucion
    @Test
    public void testGuardarInstitucion() {
        Institucion institucion = new Institucion();

when(institucionRepositorio.save(institucion)).thenReturn(institucion);

        Institucion resultado =
institucionServicio.guardarInstitucion(institucion);

        assertEquals(institucion, resultado);
        verify(institucionRepositorio, times(1)).save(institucion);
    }

    // Prueba para eliminar un Institucion
    @Test
    public void testEliminarInstitucion() {
        Institucion institucion = new Institucion();

        institucionServicio.eliminarInstitucion(institucion);

        verify(institucionRepositorio, times(1)).delete(institucion);
    }
}
```

3.2.9 Código de la prueba `PacienteServicioTest`

```
class PacienteServicioTest {

    @Mock
    private PacienteRepositorio pacienteRepositorio;  // Simula el
repositorio de Paciente

    @InjectMocks
    private PacienteServicio pacienteServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }
```

```
    // Prueba para recuperar la lista de registros de Paciente
    @Test
    public void testListarPacientes() {
        Paciente paciente1 = new Paciente();
        Paciente paciente2 = new Paciente();

when(pacienteRepositorio.findAll()).thenReturn(Arrays.asList(paciente1,
paciente2));

        List<Paciente> pacientes = pacienteServicio.listarPacientes();

        assertEquals(2, pacientes.size());
        verify(pacienteRepositorio, times(1)).findAll();
    }

    // Prueba para buscar un Paciente por su ID
    @Test
    public void testBuscarPacientePorId() {
        Paciente paciente = new Paciente();

when(pacienteRepositorio.findById(1)).thenReturn(Optional.of(paciente));

        Paciente resultado = pacienteServicio.buscarPacientePorId(1);

        assertNotNull(resultado);
        verify(pacienteRepositorio, times(1)).findById(1);
    }

    // Prueba para guardar un Paciente
    @Test
    public void testGuardarPaciente() {
        Paciente paciente = new Paciente();
        when(pacienteRepositorio.save(paciente)).thenReturn(paciente);

        Paciente resultado = pacienteServicio.guardarPaciente(paciente);

        assertEquals(paciente, resultado);
        verify(pacienteRepositorio, times(1)).save(paciente);
    }

    // Prueba para eliminar un Paciente
    @Test
    public void testEliminarPaciente() {
        Paciente paciente = new Paciente();

        pacienteServicio.eliminarPaciente(paciente);

        verify(pacienteRepositorio, times(1)).delete(paciente);
    }
}
```

3.2.10 Código de la prueba `PersonalServicioTest`

```
class PersonalServicioTest {

    @Mock
    private PersonalRepositorio personalRepositorio;  // Simula el
repositorio de Personal

    @InjectMocks
    private PersonalServicio personalServicio;  // Servicio a probar

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);  // Inicializa las anotaciones de
Mockito
    }

    // Prueba para recuperar la lista de registros de Personal
    @Test
    public void testListarPersonales() {
        Personal personal1 = new Personal();
        Personal personal2 = new Personal();

when(personalRepositorio.findAll()).thenReturn(Arrays.asList(personal1,
personal2));

        List<Personal> personales = personalServicio.listarPersonalS();

        assertEquals(2, personales.size());
        verify(personalRepositorio, times(1)).findAll();
    }

    // Prueba para buscar un Personal por su ID
    @Test
    public void testBuscarPersonalPorId() {
        Personal personal = new Personal();

when(personalRepositorio.findById(1)).thenReturn(Optional.of(personal));

        Personal resultado = personalServicio.buscarPersonalPorId(1);

        assertNotNull(resultado);
        verify(personalRepositorio, times(1)).findById(1);
    }

    // Prueba para guardar un Personal
    @Test
    public void testGuardarPersonal() {
        Personal personal = new Personal();
        when(personalRepositorio.save(personal)).thenReturn(personal);

        Personal resultado = personalServicio.guardarPersonal(personal);

        assertEquals(personal, resultado);
        verify(personalRepositorio, times(1)).save(personal);
    }
```

```java
    // Prueba para eliminar un Personal
    @Test
    public void testEliminarPersonal() {
        Personal personal = new Personal();

        personalServicio.eliminarPersonal(personal);

        verify(personalRepositorio, times(1)).delete(personal);
    }
}
```

Resultados de la ejecución con el comando **`mvn test`**

```
Hibernate: select r1_0.id,r1_0.created_at,r1_0.descripcion,r1_0.nombre,r1_0.updated_at from roles r1_0 where r1_0.nombre=?
Hibernate: select u1_0.id,u1_0.created_at,u1_0.email,u1_0.nombreCompleto,u1_0.password,u1_0.role_id,u1_0.updated_at from usuarios u1_0 where u1_0.email
=?
Hibernate: select r1_0.id,r1_0.created_at,r1_0.descripcion,r1_0.nombre,r1_0.updated_at from roles r1_0 where r1_0.id=?
Hibernate: select r1_0.id,r1_0.created_at,r1_0.descripcion,r1_0.nombre,r1_0.updated_at from roles r1_0 where r1_0.nombre=?
codelicht.sapresis.auth.entidad.Role@6e863469
Hibernate: select r1_0.id,r1_0.created_at,r1_0.descripcion,r1_0.nombre,r1_0.updated_at from roles r1_0 where r1_0.nombre=?
codelicht.sapresis.auth.entidad.Role@3ba55a6b
Hibernate: select r1_0.id,r1_0.created_at,r1_0.descripcion,r1_0.nombre,r1_0.updated_at from roles r1_0 where r1_0.nombre=?
codelicht.sapresis.auth.entidad.Role@fb79241
[main] INFO : codelicht.sapresis.SapresisApplicationTests - Started SapresisApplicationTests in 14.519 seconds (process running for 16.567)
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 16.33 s -- in codelicht.sapresis.SapresisApplicationTests
[INFO] Running codelicht.sapresis.servicio.implementacion.ConsultaServicioTest
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.628 s -- in codelicht.sapresis.servicio.implementacion.ConsultaServicioTest
[INFO] Running codelicht.sapresis.servicio.implementacion.ConsultorioServicioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.182 s -- in codelicht.sapresis.servicio.implementacion.ConsultorioServicioTest
[INFO] Running codelicht.sapresis.servicio.implementacion.DependenciaServicioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.103 s -- in codelicht.sapresis.servicio.implementacion.DependenciaServicioTest
[INFO] Running codelicht.sapresis.servicio.implementacion.DoctorServicioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.102 s -- in codelicht.sapresis.servicio.implementacion.DoctorServicioTest
[INFO] Running codelicht.sapresis.servicio.implementacion.EpsServicioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.092 s -- in codelicht.sapresis.servicio.implementacion.EpsServicioTest
[INFO] Running codelicht.sapresis.servicio.implementacion.FacturaServicioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.124 s -- in codelicht.sapresis.servicio.implementacion.FacturaServicioTest
[INFO] Running codelicht.sapresis.servicio.implementacion.FormulaServicioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.104 s -- in codelicht.sapresis.servicio.implementacion.FormulaServicioTest
[INFO] Running codelicht.sapresis.servicio.implementacion.InstitucionServicioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.074 s -- in codelicht.sapresis.servicio.implementacion.InstitucionServicioTest
[INFO] Running codelicht.sapresis.servicio.implementacion.PacienteServicioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.080 s -- in codelicht.sapresis.servicio.implementacion.PacienteServicioTest
[INFO] Running codelicht.sapresis.servicio.implementacion.PersonalServicioTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.076 s -- in codelicht.sapresis.servicio.implementacion.PersonalServicioTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 42, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  23.620 s
[INFO] Finished at: 2024-10-02T17:25:37-05:00
[INFO] ------------------------------------------------------------------------
PS C:\Projects\sapresis\backend>
```

Finalmente, los resultados de las pruebas se presentan de forma general. Las pruebas se ejecutaron mediante Maven y el plugin **`mvn site`** se utilizó para generar un informe detallado que incluye la cobertura de las pruebas y la información general del proyecto El enlace al informe generado se puede consultar a continuación: https://mauriciomonroy.github.io/sapresis/

4 Conclusiones

El desarrollo de una aplicación como Sapresis requiere una revisión meticulosa de cada una de sus funcionalidades, con el fin de asegurar que la lógica de transferencia y manejo de la información se ejecute de forma precisa y eficiente. Las pruebas unitarias presentadas en este documento validan que la lógica interna del sistema, tanto en los servicios como en la autenticación y en las operaciones CRUD de las entidades, funciona correctamente. Estas pruebas demuestran que la aplicación cuenta con una base sólida para su correcto despliegue y uso en un entorno real.

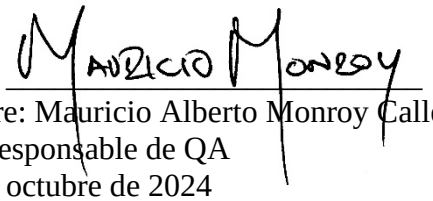
En el futuro, se planea implementar pruebas de mayor complejidad, como pruebas de integración que evalúen la interacción entre los diferentes componentes del sistema y garanticen un flujo adecuado de la información a lo largo de la arquitectura de la aplicación. También se desarrollarán pruebas de carga para evaluar la estabilidad de la aplicación en condiciones de uso intensivo.

Finalmente, se recomienda revisar la documentación técnica adjunta, especialmente la relacionada con el ambiente de desarrollo y pruebas, para obtener una visión completa del entorno utilizado y confirmar la validez de los resultados obtenidos en las pruebas unitarias realizadas hasta el momento.

5 Aceptación de Pruebas

El responsable del proceso de pruebas de la aplicación **Sapresis** certifica que todas las pruebas se realizaron exitosamente. El sistema ha demostrado un comportamiento estable y se ha validado que cumple con los criterios de calidad esperados.

- **Firma del Auditor de Pruebas:**

Firma: 
Nombre: Mauricio Alberto Monroy Calle
Rol: Responsable de QA
Fecha: octubre de 2024