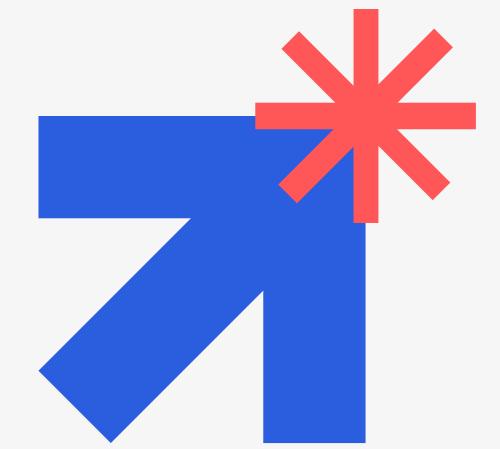




Servicios Médicos UDLAP

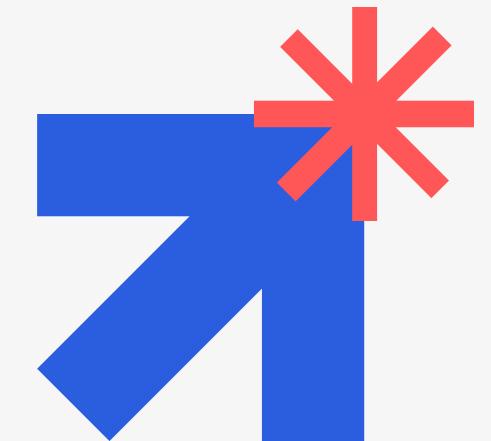
Alberto Amador Loza
Brian Carbajal Ortiz
Jeronimo Sánchez Armas Torres
Marcos Miguel Gutierrez Toniz
Ricardo Montiel Bibiesca



índice

1. Problemática
2. Solución
3. Ventanas
4. Base de Datos
5. Pruebas
6. Video

Problemática



En la UDLAP, los estudiantes enfrentan largos tiempos de espera en el servicio médico, incluso para malestares menores. Esto provoca pérdida de clases, saturación innecesaria y una mala experiencia en la atención.

Actualmente no existe un sistema que permita agendar citas, consultar disponibilidad o gestionar urgencias, lo que genera desorganización tanto para alumnos como para el personal médico.

Se necesita una solución que optimice la atención, reduzca tiempos de espera y mejore la eficiencia del servicio.

Ventana General (Login)

Descripción

La Ventana General permite a los usuarios ingresar sus credenciales para acceder al sistema.

El campo para el ID y la contraseña es validado antes de acceder al panel correspondiente, según si el usuario es estudiante o administrador.

¿Que ve el usuario?



Ventana Alumno

Descripción

El Menú Principal del Alumno actúa como un hub central con botones para acceder rápidamente a las funciones clave:

- Agendar Cita
- Modificar Cita
- Chatbot
- Urgencias

¿Que ve el usuario?



Sección “Agendar Cita”

Descripción

En esta sección, el alumno ingresa sus datos (nombre, edad, peso) y, tras la validación, selecciona un horario disponible para su cita.

Si los datos son incorrectos o el horario está ocupado, se muestra un mensaje de advertencia.

¿Que ve el usuario?

The image shows two windows side-by-side. The left window is titled "Agendar Cita - Registro" and has a header bar with minimize, maximize, and close buttons. It contains a section titled "REGISTRO DE DATOS" and three input fields labeled "Nombre:", "Edad:", and "Peso (kg)". Below these fields is a large blue button labeled "Continuar". The right window is titled "Seleccionar Horario" and also has a header bar with minimize, maximize, and close buttons. It contains a section titled "SELECCIONAR HORARIO" and a dropdown menu labeled "Horario: -- Seleccione un horario --". Below this is a large blue button labeled "Guardar Cita". At the bottom of the right window, there is a status message: "Nombre: Ricardo Villalba | Edad: 27 | Peso: 67.8".

Sección "Modificar Cita"

Descripción

Permite al alumno gestionar citas existentes.

Se puede modificar o cancelar una cita con la validación de disponibilidad de horarios y confirmaciones de modificación o cancelación.

¿Que ve el usuario?

Modificar Cita

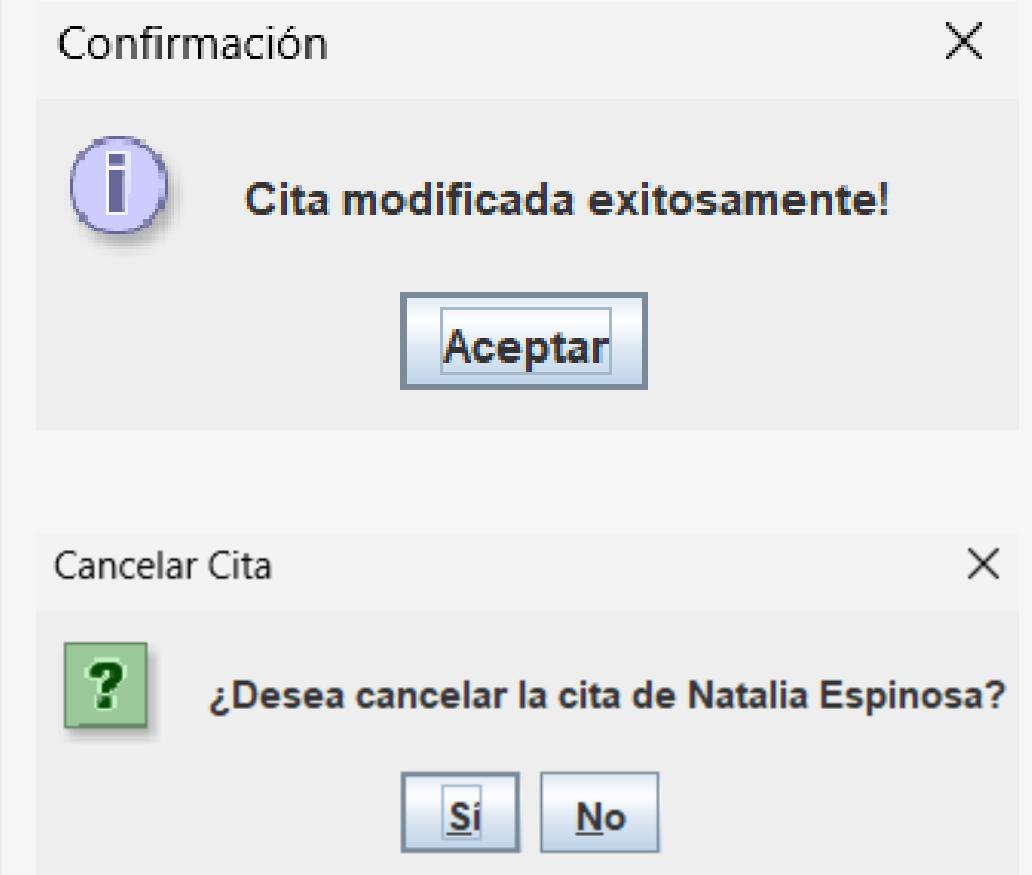
MODIFICAR CITA

Nombre:

Edad:

Peso (kg):

Horario: -- Seleccione un horario --



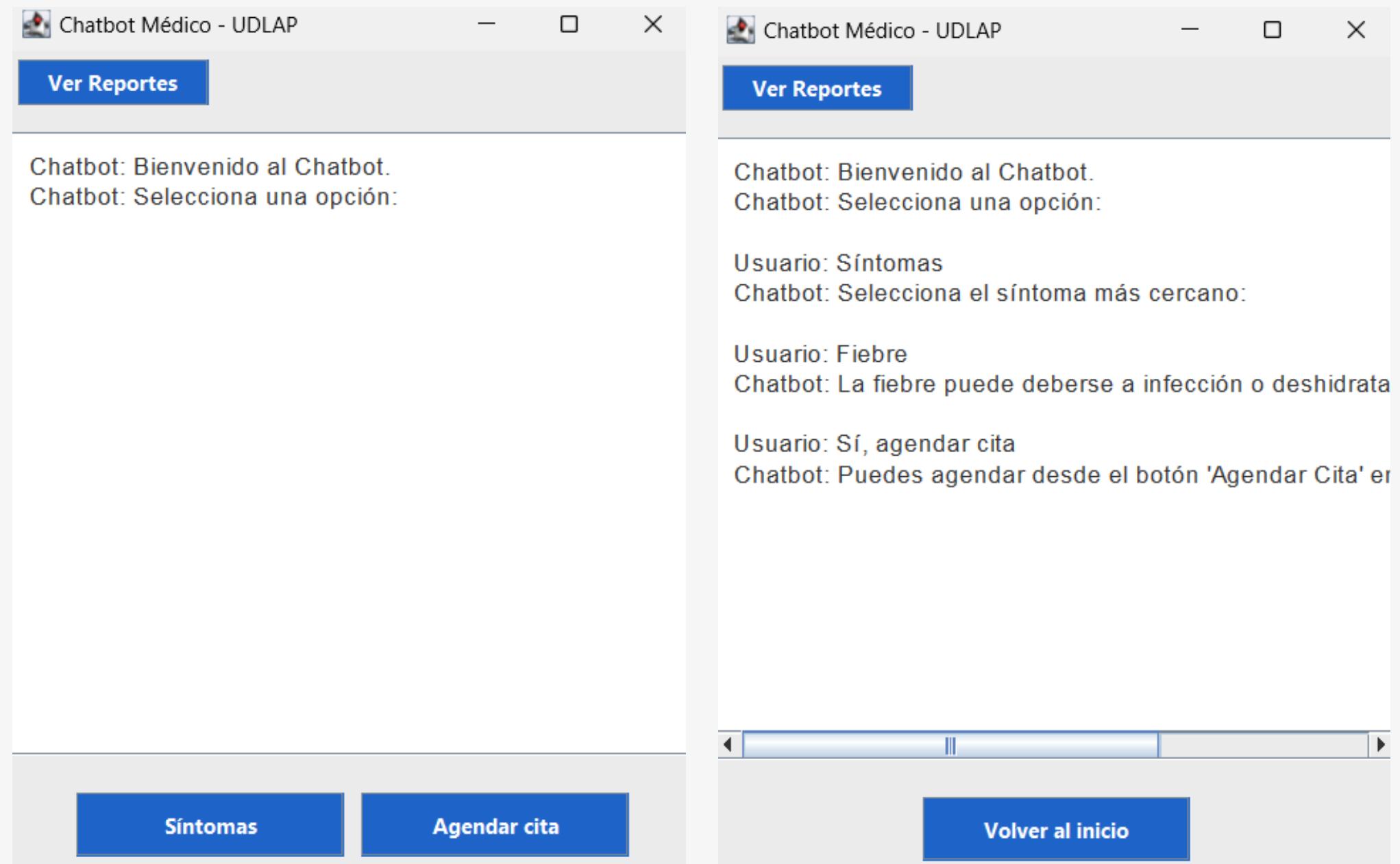
Sección “Chatbot”

Descripción

El Chatbot guía al alumno a través de un árbol de decisiones para ofrecer orientación sobre síntomas comunes.

La interacción se guarda como historial en un archivo de texto.

¿Que ve el usuario?



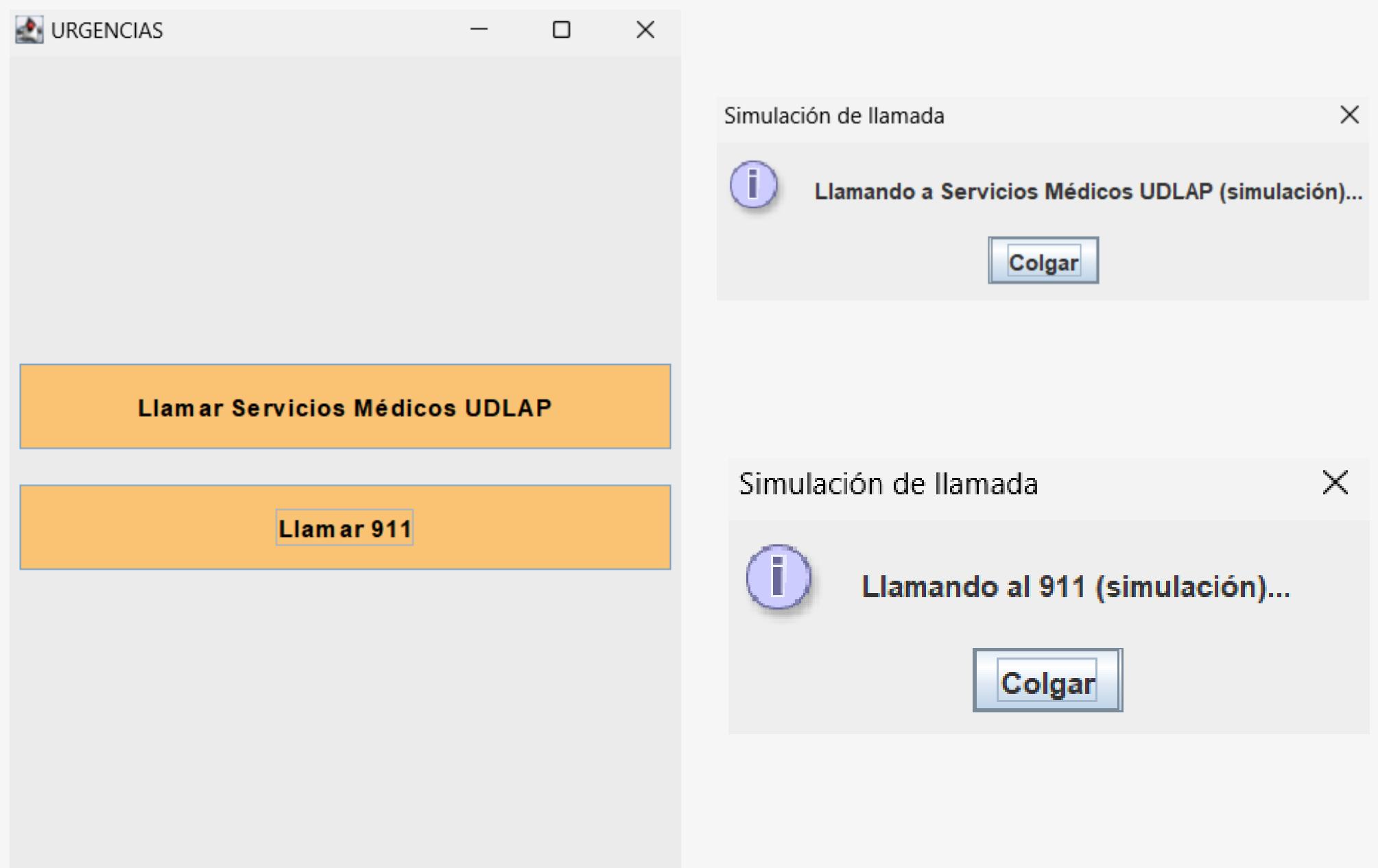
Sección “Urgencias”

Descripción

En caso de emergencia, el alumno puede acceder rápidamente a dos opciones: contactar los servicios médicos del campus o llamar a emergencias externas (911).

Ambas acciones generan alertas visuales.

¿Que ve el usuario?



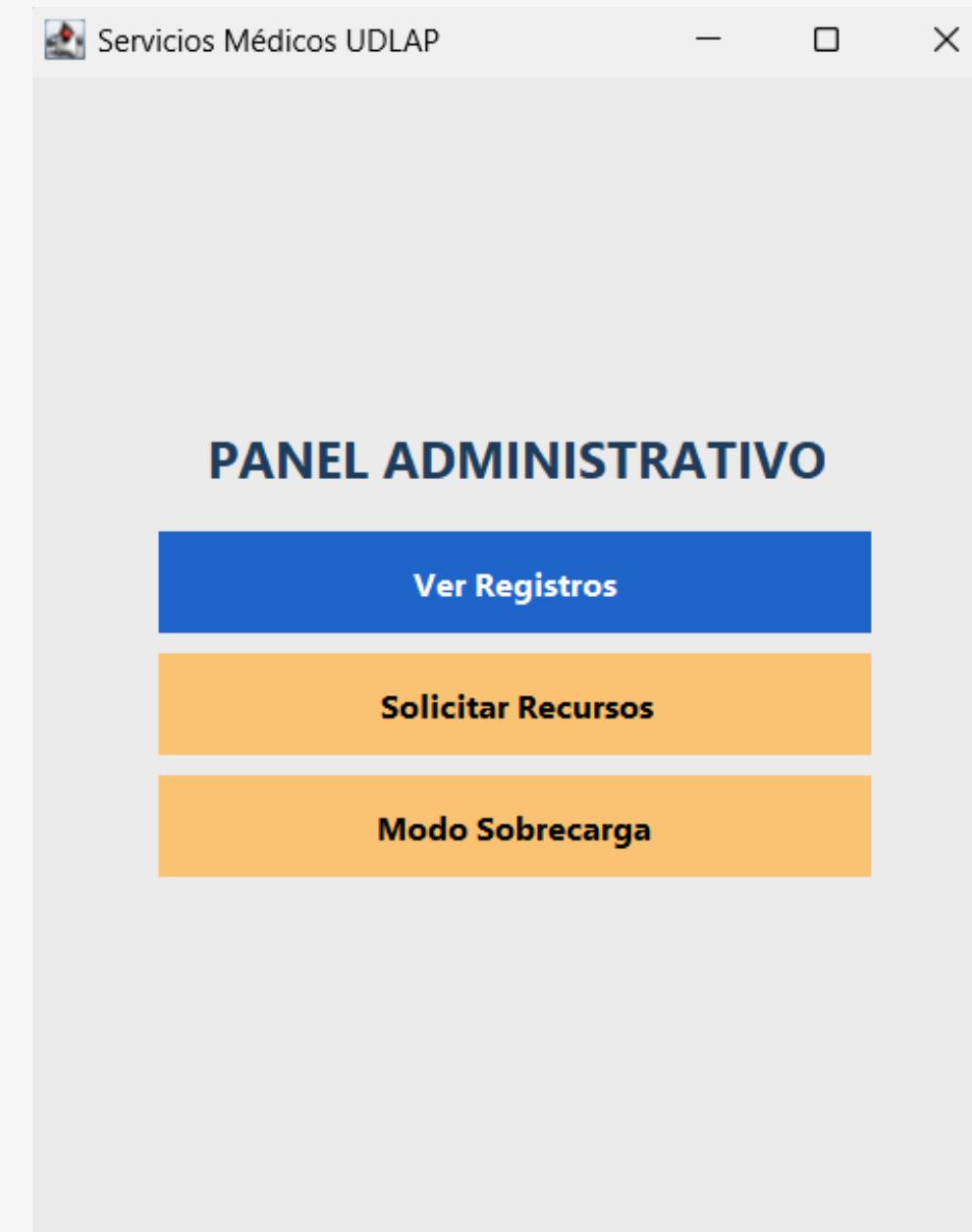
Ventana Administrador

Descripción

El Menú Administrador es el centro de control para personal médico y administrativo.

Permite gestionar registros, recursos y situaciones de sobrecarga.

¿Que ve el usuario?



Sección “Ver Registros”

Descripción

Permite al administrador visualizar registros de citas pasadas y otros datos almacenados en la base de datos.

La visualización se presenta en formato tabular y es solo lectura.

¿Que ve el usuario?

Registros - Base de datos				
Recursos faltantes	Sobrecarga	Citas		
Samuel Pacheco	Pérdida momentánea de visión	Alta	Hospital	27-11-2025 10:46:54
Mónica Serrano	Dificultad respiratoria leve	Media	Campus	27-11-2025 10:46:45
Eduardo Palma	Inflamación en rodilla	Media	Campus	27-11-2025 10:46:35
Arlet Castañeda	Dolor dental	Baja	Campus	27-11-2025 10:46:22
Tomás Calderón	Quemadura de segundo grado	Alta	Hospital	27-11-2025 10:46:13
Carmen Vázquez	Problemas de equilibrio	Media	Campus	27-11-2025 10:46:02
Adrián Farias	Reacción alérgica severa	Alta	Hospital	27-11-2025 10:45:47
Ximena Ponce	Dolor estomacal leve	Baja	Campus	27-11-2025 10:45:36
Benjamín Soto	Tos con sangre	Alta	Hospital	27-11-2025 10:45:25
Regina Alba	Mareo súbito	Media	Campus	27-11-2025 10:45:13
Mauricio Luján	Dolor de muñeca	Baja	Campus	27-11-2025 10:45:03
Danna Arce	Golpe en costillas	Media	Campus	27-11-2025 10:44:53
Lorenzo Vidal	Inflamación moderada	Media	Campus	27-11-2025 10:44:43
Ariana Leiva	Dificultad para tragar	Alta	Hospital	27-11-2025 10:42:54
Hugo Arellano	Possible intoxicación	Alta	Hospital	27-11-2025 10:42:36
Brenda Lozano	Dolor postoperatorio	Media	Hospital	27-11-2025 10:42:24
Gael Navarro	Sarpullido	Baja	Campus	27-11-2025 10:42:06
Natalia Rivas	Hipoglucemia	Alta	Hospital	27-11-2025 10:41:52
Félix Bravo	Hemorragia nasal	Media	Campus	27-11-2025 10:41:35
Ingrid Sandoval	Lesión de tobillo	Media	Campus	27-11-2025 10:41:23
Joaquín Beltrán	Faringitis	Baja	Campus	27-11-2025 10:41:07
Teresa Juárez	Dolor menstrual fuerte	Media	Campus	27-11-2025 10:40:55
Omar Valdivia	Golpe en la cabeza	Alta	Hospital	27-11-2025 10:40:42
Aline Rosas	Dolor de oído	Baja	Campus	27-11-2025 10:40:33
Esteban Ortiz	Palpitaciones	Alta	Hospital	27-11-2025 10:40:18
Daniela Correa	Quemadura leve	Media	Campus	27-11-2025 10:39:56
Héctor Varela	Dolor renal intenso	Alta	Hospital	27-11-2025 10:39:44
Claudia Ramírez	Crisis de ansiedad	Alta	Campus	27-11-2025 10:39:31
Pablo Silva	Deshidratación leve	Baja	Campus	27-11-2025 10:39:20
Adriana Fuentes	Migraña severa	Media	Campus	27-11-2025 10:39:08

Actualizar Cerrar

Sección “Falta de Recursos”

Descripción

Muestra una tabla con recursos médicos faltantes.

El administrador puede registrar acciones, como la solicitud de reposición, y validar la falta de recursos.

¿Que ve el usuario?

The screenshot shows a software interface titled "Falta de recursos médicos - Gestión de...". The main area displays a table titled "Recursos faltantes detectados" with the following data:

Tipo	Detalle	Severidad
Medicamento	Paracetamol 500 mg (tabletas)	Alta
Medicamento	Ibuprofeno 400 mg (tabletas)	Alta
Medicamento	Sales de rehidratación oral	Alta
Medicamento	Solución salina (bolsa IV)	Alta
Material	Guantes desechables	Alta
Material	Cubrebocas quirúrgicos	Alta
Material	Jeringas	Alta
Material	Vendas	Alta
Material	Catéter IV	Media

Below the table, there is a section titled "Confirmación de Autoridad" with a checkbox labeled "Validado por personal médico autorizado". At the bottom, there is a section titled "Notas / acciones tomadas" which is currently empty. At the very bottom of the window are two buttons: "Guardar acción" (in blue) and "Cancelar" (in red).

Sección “Modo Sobre carga”

Descripción

En situaciones de alta demanda, esta sección permite registrar rápidamente a los pacientes y clasificar su urgencia.

También incluye la opción de derivación a un hospital.

¿Que ve el usuario?

The screenshot shows a software window titled "Modo Sobre carga - Registro de Paciente". The main title is "MODO SOBRECARGA". The form contains the following fields:

- Nombre: (Name) - An input field.
- Síntomas: (Symptoms) - A large input field.
- Prioridad: (Priority) - A dropdown menu set to "Alta".
- Acción: (Action) - A checkbox group with options "Campus" and "Hospital".

At the bottom is a blue button labeled "Registrar al Paciente".

Conexión a la Base de Datos

Descripción

Archivo: formularios.db

Contiene tablas para guardar citas registradas, recursos faltantes y pacientes en modo sobrecarga

Los datos se encuentran diferenciados por columnas.

La conexión a la base de datos es por medio de la clase: ConexionSQLite.java

```
6 import java.sql.Connection;
7 import java.sql.DriverManager;
8 import java.sql.SQLException;

9
10 public class ConexionSQLite {
11     @SuppressWarnings("CallToPrintStackTrace")
12     public static Connection conectar() {
13         Connection conexion = null;
14         try {
15             // Cargar el driver JDBC para SQLite
16             Class.forName(className: "org.sqlite.JDBC");
17             // Ruta a la base de datos SQLite
18             String url = "jdbc:sqlite:formularios.db";
19             conexion = DriverManager.getConnection(url);
20             System.out.println(x: "Conexión exitosa a SQLite");
21         } catch (ClassNotFoundException e) {
22             System.err.println(x: "Error: No se encontró el driver SQLite");
23         } catch (SQLException e) {
24             System.err.println(x: "Error de conexión a la base de datos");
25             e.printStackTrace();
26         }
27         return conexion;
28     }
29     public static void main(String[] args) {
30         conectar();
31     }
32 }
33 
```

Base de Datos

DB Browser for SQLite - C:\Users\monti\OneDrive\Documentos\VSCode\Servicios Medicos UDLAP\Servicios Medicos UDLAP\formularios.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragmas Execute SQL

Create Table Create Index Modify Table Delete Table Print Refresh

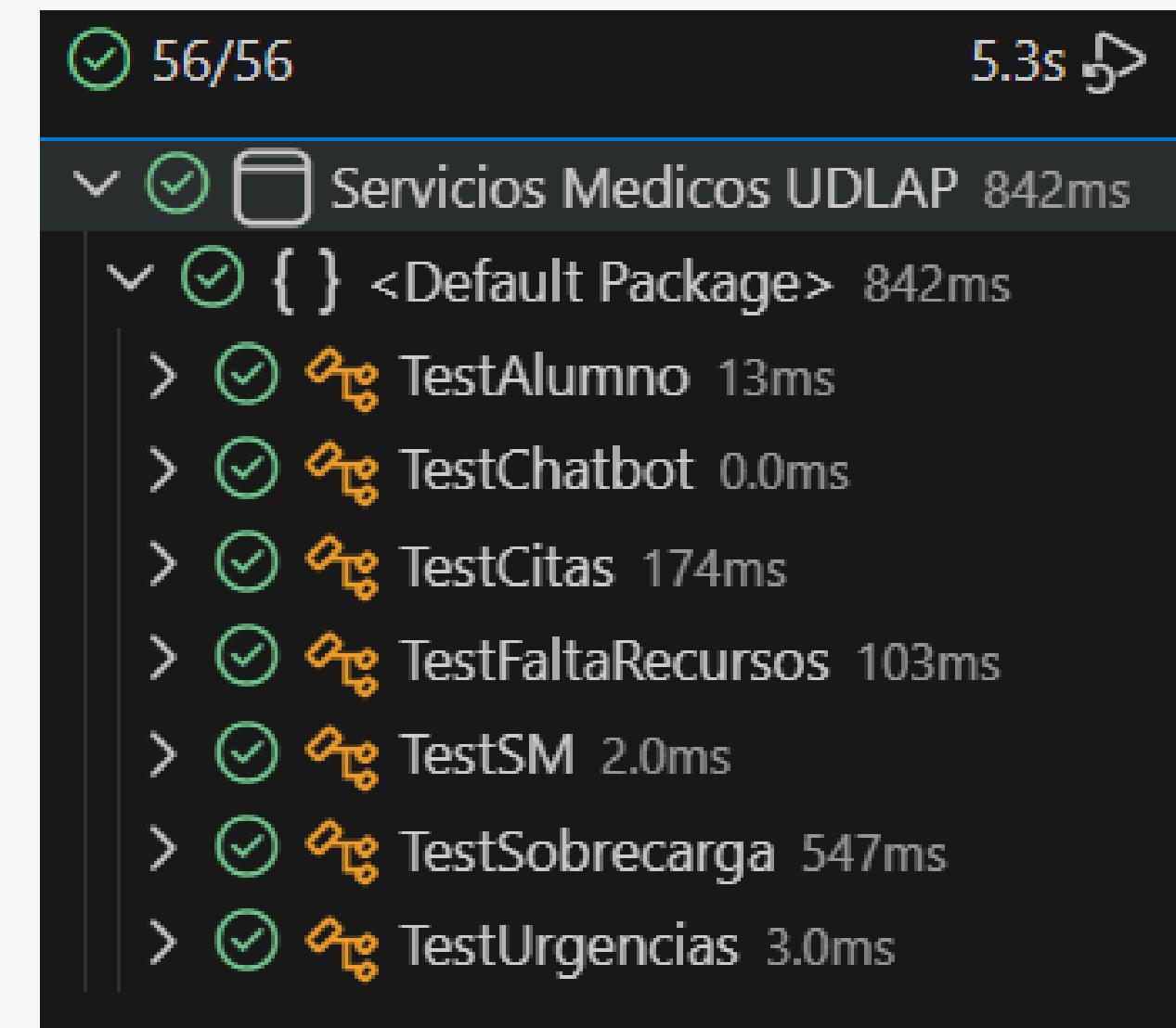
Name	Type	Schema
Tables (4)		
citas		CREATE TABLE citas (id INTEGER PRIMARY KEY AUTOINCREMENT, nombre TEXT UNIQUE, edad INTEGER, peso REAL, horario TEXT UNIQUE, creado_at TEXT)
id	INTEGER	"id" INTEGER
nombre	TEXT	"nombre" TEXT UNIQUE
edad	INTEGER	"edad" INTEGER
peso	REAL	"peso" REAL
horario	TEXT	"horario" TEXT UNIQUE
creado_at	TEXT	"creado_at" TEXT
recursos_faltantes		CREATE TABLE recursos_faltantes (id INTEGER PRIMARY KEY AUTOINCREMENT, tipo TEXT, detalle TEXT, severidad TEXT, validado INTEGER, notas TEXT, creado_at TEXT)
id	INTEGER	"id" INTEGER
tipo	TEXT	"tipo" TEXT
detalle	TEXT	"detalle" TEXT
severidad	TEXT	"severidad" TEXT
validado	INTEGER	"validado" INTEGER
notas	TEXT	"notas" TEXT
creado_at	TEXT	"creado_at" TEXT
sobrecarga_pacientes		CREATE TABLE sobrecarga_pacientes (id INTEGER PRIMARY KEY AUTOINCREMENT, nombre TEXT, sintomas TEXT, prioridad TEXT, accion TEXT, creado_at TEXT)
id	INTEGER	"id" INTEGER
nombre	TEXT	"nombre" TEXT
sintomas	TEXT	"sintomas" TEXT
prioridad	TEXT	"prioridad" TEXT
accion	TEXT	"accion" TEXT
creado_at	TEXT	"creado_at" TEXT
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
name		"name"
seq		"seq"
Indices (0)		
Views (0)		
Triggers (0)		

UTF-8

Pruebas

El sistema fue sometido a diversas pruebas para validar su funcionamiento.

Se realizaron pruebas unitarias, de integración y de interfaz para asegurar la fiabilidad del sistema.

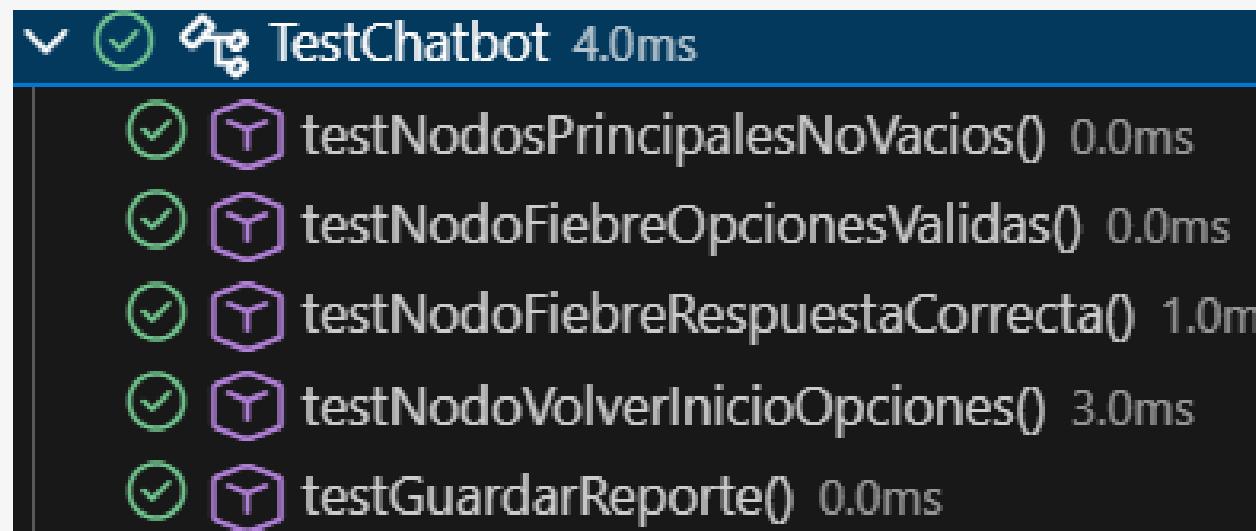


Test Alumno

```
✓ ✘ ⚙ TestAlumno 13ms
  ✓ ✘ ⚙ testAlumnoConstructor() 1.0ms
  ✓ ✘ ⚙ testAlumnoSetNombre() 0.0ms
  ✓ ✘ ⚙ testAlumnoSetNombreNull() 0.0ms
  ✓ ✘ ⚙ testAlumnoSetEdad() 0.0ms
  ✓ ✘ ⚙ testAlumnoSetEdadNegativa() 1.0ms
  ✓ ✘ ⚙ testAlumnoSetEdadCero() 0.0ms
  ✓ ✘ ⚙ testAlumnoSetPeso() 0.0ms
  ✓ ✘ ⚙ testAlumnoSetPesoNegativo() 4.0ms
  ✓ ✘ ⚙ testAlumnoSetPesoCero() 0.0ms
  ✓ ✘ ⚙ testAlumnoSetHorarioCita() 0.0ms
  ✓ ✘ ⚙ testAlumnoMostrarInfo() 7.0ms
  ✓ ✘ ⚙ testAlumnoMostrarInfoConCita() 0.0ms
```

```
6 ✓ import org.junit.Test;
7   import org.junit.Before;
8   import static org.junit.Assert.*;
9
10  ✓ public class TestAlumno {
11
12    private Alumno alumnoTest;
13
14    @Before
15    ✓ public void setup() {
16      alumnoTest = new Alumno(nombre: "Juan Pérez", edad: 20, peso: 70.5);
17    }
18
19    ✓ @Test
20    public void testAlumnoConstructor() {
21      assertEquals(message: "Nombre debe ser Juan Pérez", expected: "Juan Pérez", alumnoTest.getNombre());
22      assertEquals(message: "Edad debe ser 20", expected: 20, alumnoTest.getEdad());
23      assertEquals(message: "Peso debe ser 70.5", expected: 70.5, alumnoTest.getPeso(), delta: 0.01);
24      assertNull(message: "Horario debe ser null inicialmente", alumnoTest.getHorarioCita());
25    }
26
27    ✓ @Test
28    public void testAlumnoSetNombre() {
29      alumnoTest.setNombre(nombre: "María García");
30      assertEquals(message: "Nombre debe actualizar a María García", expected: "María García", alumnoTest.getNombre());
31    }
32
33    ✓ @Test
34    public void testAlumnoSetNombreNull() {
35      alumnoTest.setNombre(nombre: null);
36      assertNull(message: "Nombre puede ser null", alumnoTest.getNombre());
37    }
38  }
```

Test Chatbot



```
38     // ===== TEST DEL NODO FIEBRE =====
39
40     @Test
41     public void testNodoFiebreOpcionesValidas() {
42         assertNotNull(message: "Las opciones de fiebre no deben ser nulas", opcionesFiebre);
43         assertEquals(message: "El nodo fiebre debe tener 2 opciones", expected: 2, opcionesFiebre.length());
44
45         for (String opcion : opcionesFiebre) {
46             assertTrue(message: "Las opciones del nodo fiebre deben tener texto", opcion.length() > 0);
47         }
48     }
49
50     @Test
51     public void testNodoFiebreRespuestaCorrecta() {
52         assertNotNull(message: "La respuesta del nodo Fiebre no debe ser nula", respuestaFiebre);
53         assertTrue(message: "La respuesta debe mencionar fiebre", respuestaFiebre.toLowerCase().contains(s: "fiebre"));
54     }
55
56     // ===== TEST DEL NODO VOLVER AL INICIO =====
57
58     @Test
59     public void testNodoVolverInicioOpciones() {
60         assertNotNull(message: "El nodo Volver al inicio no debe ser nulo", opcionesVolver);
61         assertEquals(message: "Volver al inicio debe tener 3 opciones", expected: 3, opcionesVolver.length());
62     }
63 }
```

Test Citas

✓ TestCitas 179ms

- ✓ testReservarCitaInsertaEnBDYElimina() 34ms
- ✓ testActualizarCitaEnBD() 55ms
- ✓ testHorarioDisponibilidadViaBD() 37ms
- ✓ testUpsertCitaInsertaYActualiza() 53ms

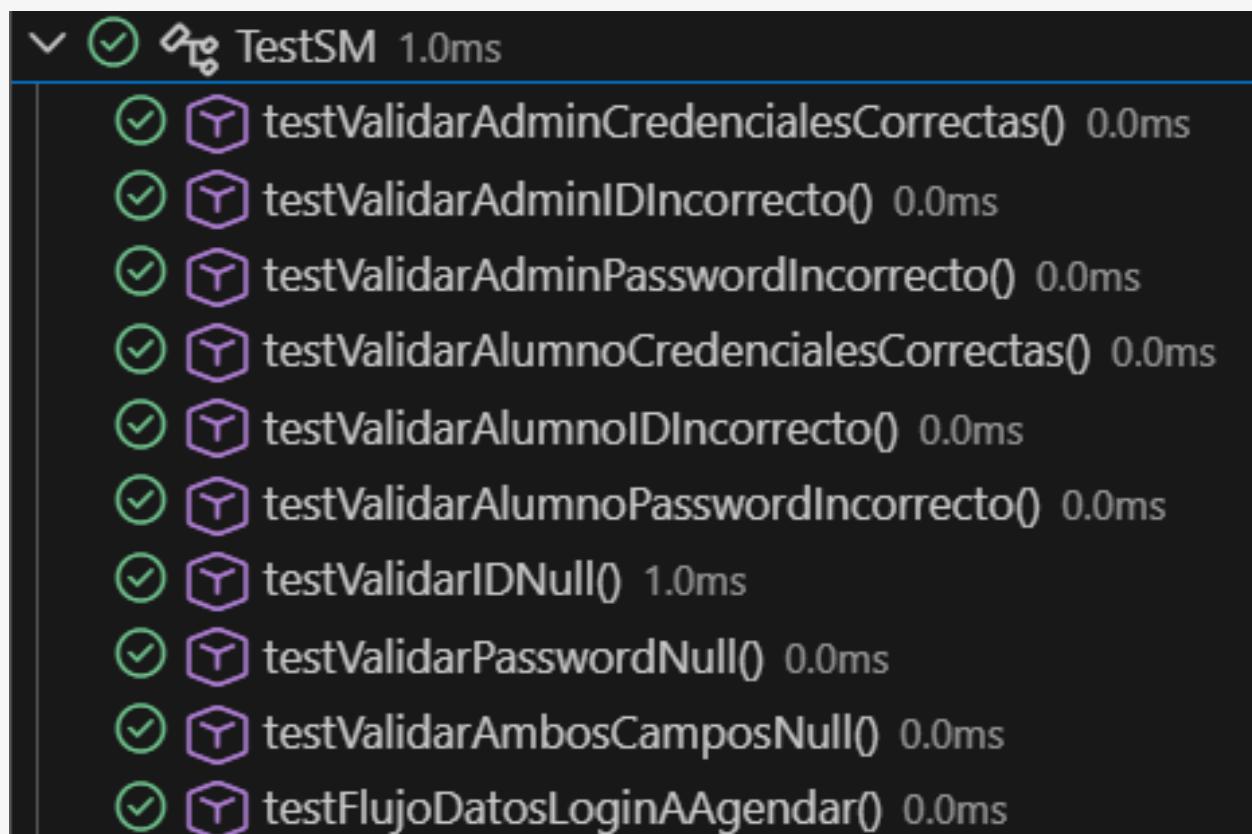
```
97     @Test
98     public void testUpsertCitaInsertaYActualiza() {
99         String nombre = "UpsertUser";
100        // Asegurar limpieza previa en horarios objetivo
101        DatabaseHelper.deleteCitaByNombre(nombre);
102        DatabaseHelper.deleteCitaByHorario(horario: "08:00 AM - 08:30 AM");
103        DatabaseHelper.deleteCitaByHorario(horario: "10:00 AM - 10:30 AM");
104
105        // Usar horarios libres (08:00 y 10:00) para evitar colisiones con datos existentes
106        DatabaseHelper.upsertCita(nombre, edad: 20, peso: 60.0, horario: "08:00 AM - 08:30 AM");
107
108        // Volver a upsert con mismo nombre y horario distinto (libre)
109        DatabaseHelper.upsertCita(nombre, edad: 21, peso: 61.5, horario: "10:00 AM - 10:30 AM");
110
111        List<String[]> citas = DatabaseHelper.getCitas();
112        boolean found = false;
113        for (String[] c : citas) {
114            if (c[0].equals(nombre) && "10:00 AM - 10:30 AM".equals(c[3]) && Integer.parseInt(c[1]) == 21) {
115                found = true;
116                break;
117            }
118        }
119        assertTrue(message: "Upsert debe actualizar la cita existente para el mismo nombre", found);
120
121        // Limpieza
122        DatabaseHelper.deleteCitaByNombre(nombre);
123        List<String[]> despues = DatabaseHelper.getCitas();
124        boolean sigue = false;
125        for (String[] c : despues) if (c[0].equals(nombre)) { sigue = true; break; }
126        assertFalse(message: "La cita debe eliminarse en la limpieza", sigue);
127    }
128}
129}
```

Test Falta de Recursos

```
✓ (T) TestFaltaRecursos 92ms
  ✓ (T) testInventarioRecursosNoNulo() 0.0ms
  ✓ (T) testRecursoGetters() 0.0ms
  ✓ (T) testRecursoDisponibleEstaEnFalta() 0.0ms
  ✓ (T) testRecursoDisponibleNoEstaEnFalta() 0.0ms
  ✓ (T) testInventarioGetTodos() 5.0ms
  ✓ (T) testInventarioGetEnFalta() 0.0ms
  ✓ (T) testInsertRecursoDirectDB() 28ms
  ✓ (T) testInsertRecursoHandlesNullsAndDefaults()...
  ✓ (T) testInsertRecursoValidadoFlag() 29ms
```

```
131     // Limpieza
132     try (Connection conn = ConexionSQLite.conectar(); PreparedStatement ps = conn.prepareStatement(sql: "DELETE FROM recursos_faltantes WHERE detalle = ?"));
133         ps.setString(parameterIndex: 1, detalle);
134         ps.executeUpdate();
135     }
136
137     assertTrue(message: "Registro con valores nulos debe insertarse y encontrarse", found);
138 }
139
140 @Test
141 public void testInsertRecursoValidadoFlag() throws Exception {
142     String tipo = "TEST_VALIDADO_" + System.currentTimeMillis();
143     String detalle = "DetalleValFlag" + System.currentTimeMillis();
144
145     DatabaseHelper.insertRecurso(tipo, detalle, severidad: "Baja", validado: false, notas: "");
146
147     boolean found = false;
148     for (String[] r : DatabaseHelper.getRecursos()) {
149         if (r.length >= 6 && detalle.equals(r[1])) {
150             found = true;
151             assertEquals(message: "Validado debe ser 'No' para false", expected: "No", r[3]);
152             break;
153         }
154     }
155
156     // Limpieza
157     try (Connection conn = ConexionSQLite.conectar(); PreparedStatement ps = conn.prepareStatement(sql: "DELETE FROM recursos_faltantes WHERE detalle = ?"));
158         ps.setString(parameterIndex: 1, tipo);
159         ps.executeUpdate();
160     }
161
162     assertTrue(message: "Registro insertado debe encontrarse en la BD", found);
163 }
164 }
```

Test SM



```
5 import org.junit.Test;
6 import static org.junit.Assert.*;
7
8 public class TestSM {
9
10 // ===== TESTS PARA DatosInicio =====
11 @Test
12 public void testValidarAdminCredencialesCorrectas() {
13     assertTrue(message: "Admin debe validarse con ID 12345 y PW Admin.1",
14     DatosInicio.validar(id: "12345", contra: "Admin.1"));
15 }
16
17 @Test
18 public void testValidarAdminIDIncorrecto() {
19     assertFalse(message: "Admin con ID incorrecto no debe validarse",
20     DatosInicio.validar(id: "99999", contra: "Admin.1"));
21 }
22
23 @Test
24 public void testValidarAdminPasswordIncorrecto() {
25     assertFalse(message: "Admin con PW incorrecta no debe validarse",
26     DatosInicio.validar(id: "12345", contra: "WrongPassword"));
27 }
28
29 @Test
30 public void testValidarAlumnoCredencialesCorrectas() {
31     assertTrue(message: "Alumno debe validarse con ID 123456 y PW Alumno.1",
32     DatosInicio.validar(id: "123456", contra: "Alumno.1"));
33 }
34 }
```

Test Sobrecarga

```
✓ ✅ 🏁 TestSobrecarga 378ms
  ✓ ✅ 🏁 testComponentsExist() 5.0ms
  ✓ ✅ 🏁 testValidarFormulario_emptyAndFilled() 269ms
  ✓ ✅ 🏁 testClearFormResetsFields() 16ms
  ✓ ✅ 🏁 testInsertSobrecargaDirectDB() 57ms
  ✓ ✅ 🏁 testInsertSobrecargaHandlesEmptyFields() 31ms
```

```
109
110 @Test
111 public void testInsertSobrecargaDirectDB() throws Exception {
112     String nombre = "TEST_SORECARGA_" + System.currentTimeMillis();
113     String sintomas = "Sintomas de prueba";
114     String prioridad = "Media";
115     String accion = "Campus";
116
117     // Insertar directamente
118     DatabaseHelper.insertSobrecarga(nombre, sintomas, prioridad, accion);
119
120     boolean found = false;
121     for (String[] row : DatabaseHelper.getSobrecarga()) {
122         if (row.length > 0 && nombre.equals(row[0])) {
123             found = true;
124             assertEquals(message: "Prioridad debe coincidir", prioridad, row[2]);
125             assertEquals(message: "Acción debe coincidir", accion, row[3]);
126             break;
127         }
128     }
129
130     // Limpieza de datos de prueba
131     try (Connection conn = ConexionSQLite.conectar(); PreparedStatement ps = conn.prepareStatement(sql: "DELETE FROM sobrecarga_pacie
132         ps.setString(parameterIndex: 1, nombre);
133         ps.executeUpdate();
134     }
135     assertTrue(message: "Registro insertado debe encontrarse en la BD", found);
```

Test Urgencias

- ✓  TestUrgencias 4.0ms
 - ✓  testTituloVentanaEsCorrecto() 0.0ms
 - ✓  testBotonServiciosExisteYTextoCorrecto() 3.0ms
 - ✓  testBoton911ExisteYTextoCorrecto() 1.0ms
 - ✓  testBotonesTienenListeners() 0.0ms

```
● 29     @Test
30     public void testBotonServiciosExisteYTextoCorrecto() {
31         // El botón de Servicios Médicos debe existir y tener el texto correcto
32         JButton btnServicios = urgencias.getServiciosButton();
33
34         assertNotNull(message: "El botón de Servicios Médicos no debe ser null", btnServicios);
35         assertEquals(message: "Texto del botón de Servicios Médicos incorrecto",
36                     |           |           | expected: "Llamar Servicios Médicos UDLAP", btnServicios.getText());
37         assertTrue(message: "El botón de Servicios Médicos debe estar habilitado",
38                     |           |           | btnServicios.isEnabled());
39     }
40
41     @Test
42     public void testBoton911ExisteYTextoCorrecto() {
43         // El botón de 911 debe existir y tener el texto correcto
44         JButton btn911 = urgencias.getEmergenciaButton();
45
46         assertNotNull(message: "El botón de 911 no debe ser null", btn911);
47         assertEquals(message: "Texto del botón 911 incorrecto",
48                     |           |           | expected: "Llamar 911", btn911.getText());
49         assertTrue(message: "El botón de 911 debe estar habilitado",
50                     |           |           | btn911.isEnabled());
51     }
```

Video

