

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE OAXACA
SISTEMAS Y COMPUTACIÓN

INGENIERÍA EN SISTEMAS COMPUTACIONALES

INGENIERÍA EN SOFTWARE

GARCIA OSORIO BOLIVAR

HIDALGO TEJADA YEUDIEL AUGUSTO

VALENTÍN SÁNCHEZ GERSON MERARI

**MANUAL DE PROGRAMADOR PARA EL SISTEMA DE LA
CLÍNICA SAGRADO CORAZÓN**

ESPINOSA PÉREZ JACOB

“6SU” - 19:00-20:00

09/OCTUBRE/2024

ÍNDICE

PROPÓSITO DEL MANUAL.....	4
ALCANCE.....	5
Conocimientos Previos.....	7
Herramientas Necesarias.....	8
Configuraciones Necesarias.....	9
¿Qué hace el sistema?.....	10
Principales Características.....	11
Objetivos del Sistema.....	11
Lenguajes de Programación.....	14
Modelado y Diseño.....	15
Metodología de Desarrollo.....	15
Errores Comunes a Evitar.....	17
Buenas Prácticas.....	18
Instrucciones de Instalación.....	19
Gestión de Dependencias.....	20
REFERENCIAS Y RECURSOS.....	71
Enlaces Útiles.....	71
Bibliografía.....	72
Glosario.....	73

INTRODUCCIÓN

En el ámbito de la salud, la gestión eficiente de la información y la optimización de los procesos internos son pilares fundamentales para garantizar una atención médica de alta calidad, centrada en las necesidades del paciente. La Clínica Sagrado Corazón, al igual que muchas instituciones del sector salud, enfrenta desafíos significativos relacionados con la administración de registros de pacientes, la gestión de citas médicas y el control de procesos internos. Estas dificultades pueden derivar en errores administrativos, demoras operativas y costos adicionales, impactando negativamente la experiencia del paciente y la calidad del servicio ofrecido.

Para abordar estos retos, resulta imprescindible implementar un sistema integral que automatice y modernice los procesos clave de la clínica. Este proyecto tiene como propósito diseñar e implementar una solución tecnológica que permita optimizar el registro de pacientes, gestionar eficientemente las citas médicas y realizar un seguimiento efectivo de los tratamientos. Además, busca integrar herramientas de soporte para la administración interna, asegurando un modelo operativo más ágil, preciso y orientado a la mejora continua.

Esta iniciativa no solo fortalecerá la capacidad operativa de la Clínica Sagrado Corazón, sino que también garantizará un servicio más seguro, accesible y centrado en la satisfacción de los pacientes, cumpliendo con los estándares de calidad y eficiencia que exige el sector salud.

PROPOSITO DEL MANUAL

El propósito de este manual es proporcionar una guía completa y estructurada para los programadores encargados de implementar, mantener y escalar el sistema de gestión de información diseñado para la Clínica Sagrado Corazón. Este manual fue creado con el objetivo de estandarizar los procesos de desarrollo, asegurar la calidad del código, y facilitar la comprensión de las herramientas y tecnologías empleadas en el proyecto.

Está destinado principalmente a desarrolladores, tanto internos como externos, que trabajen en el sistema, ofreciendo instrucciones claras y recursos necesarios para abordar las diversas tareas técnicas. También puede ser útil para analistas de sistemas y líderes de proyectos que necesiten comprender los aspectos técnicos y metodológicos involucrados.

ALCANCE

Este manual cubre las áreas y funcionalidades clave relacionadas con el desarrollo y mantenimiento del sistema de gestión de información, incluyendo:

1. Gestión de Citas:

- Desarrollo e implementación de módulos para programar, modificar y cancelar citas médicas.
- Integración con la agenda de médicos y especialistas.

2. Registro de Pacientes:

- Creación y mantenimiento de un sistema centralizado para almacenar información personal, médica y de seguros.
- Funcionalidades para actualizar y consultar registros.

3. Sistema de Facturación y Cobro:

- Automatización del proceso de facturación vinculado con los registros médicos.
- Generación y administración de facturas y registros de pago.

4. Configuración y Uso de Herramientas:

- Detalles técnicos sobre el uso de NetBeans como IDE de desarrollo.
- Configuración y conexión con MySQL como sistema de gestión de bases de datos.
- Implementación de librerías específicas para funcionalidades avanzadas.

5. Prácticas de Desarrollo Basadas en el Modelo XP:

- Metodologías ágiles para garantizar iteraciones rápidas, pruebas continuas y refactorización eficiente.

6. Resolución de Problemas Comunes:

- Diagnóstico y solución de errores habituales durante el desarrollo y la ejecución del sistema.

OBJETIVO DEL MANUAL

El objetivo principal de este manual es servir como una guía integral y estructurada para los desarrolladores, facilitando la comprensión, implementación, mantenimiento y evolución del sistema de gestión de información diseñado para la Clínica Sagrado Corazón. Este documento busca estandarizar los procesos de desarrollo y asegurar que cada componente del sistema sea creado y gestionado de manera eficiente, manteniendo altos estándares de calidad, seguridad y escalabilidad.

A través de este manual, se pretende ofrecer a los programadores una descripción detallada de la arquitectura del sistema, los lenguajes de programación utilizados, las herramientas tecnológicas aplicadas y las metodologías de desarrollo adoptadas. Asimismo, incluye las mejores prácticas para el manejo de bases de datos, la integración de módulos funcionales, la resolución de problemas comunes y la implementación de actualizaciones.

El manual también establece lineamientos claros sobre la interacción con otros sistemas y servicios, garantizando que las integraciones y funcionalidades del sistema respondan a las necesidades operativas de la clínica. Adicionalmente, proporciona ejemplos prácticos, diagramas, fragmentos de código y casos de uso para facilitar la implementación de las funcionalidades descritas.

En última instancia, este manual tiene como propósito apoyar a los programadores en la creación de un sistema robusto y confiable que mejore la gestión de pacientes, la optimización de procesos internos y la calidad del servicio ofrecido, contribuyendo al éxito del proyecto y a la excelencia operativa de la Clínica Sagrado Corazón.

REQUISITOS PREVIOS

Para garantizar una implementación eficiente y correcta del sistema de gestión de información para la Clínica Sagrado Corazón, es fundamental contar con los conocimientos, herramientas y configuraciones adecuadas.

Conocimientos Previos

1. Lenguajes de programación:

- Java: Conocimiento intermedio o avanzado para el desarrollo del sistema en NetBeans.
- SQL: Habilidad para crear, consultar y administrar bases de datos en MySQL.

2. Arquitectura del sistema:

- Comprender los principios del modelo **XP (Extreme Programming)**, incluyendo prácticas como desarrollo iterativo, refactorización y pruebas continuas.

3. Diseño y gestión de bases de datos:

- Capacidad para diseñar bases de datos relacionales, optimizar consultas y administrar permisos de usuarios.

4. Uso de entornos de desarrollo integrados (IDEs):

- Familiaridad con **NetBeans** para desarrollo de aplicaciones Java.

5. Herramientas de control de versiones:

- Conocimiento básico de herramientas como Git para el control de versiones del proyecto.

Herramientas Necesarias

1. Entorno de Desarrollo Integrado (IDE):

- **NetBeans IDE** (versión recomendada: 12.x o superior) para el desarrollo del sistema en Java.

2. Sistema de Gestión de Bases de Datos (DBMS):

- **MySQL** (versión recomendada: 8.x) para la creación y administración de la base de datos.
- Herramientas complementarias como **MySQL Workbench** para facilitar el diseño y ejecución de consultas.

3. Librerías requeridas en NetBeans:

- **JDBC Driver para MySQL (Connector/J)**: Para establecer la conexión entre la aplicación y la base de datos.
- **Librerías de interfaz gráfica**:
 - **Swing** o **JavaFX** (según los requerimientos de interfaz del sistema).
- **Apache Commons**:
 - Librerías como **Apache Commons Lang** y **Apache Commons IO** para manejo avanzado de operaciones.
- **JasperReports**:
 - Para la generación de reportes en el sistema.

4. Servidor de aplicaciones (opcional para pruebas avanzadas):

- **Apache Tomcat** si se planea integrar funcionalidades web o servicios REST en el futuro.

Configuraciones Necesarias

1. Entorno de desarrollo:

- Instalación de **Java Development Kit (JDK)** (versión recomendada: 11 o superior).
- Configuración del IDE **NetBeans** para soportar proyectos basados en Java y conectividad con MySQL.

2. Base de datos:

- Crear una base de datos en **MySQL** con las tablas iniciales diseñadas según las especificaciones del proyecto.
- Establecer permisos de usuario para la conexión segura desde la aplicación.

3. Conexión entre IDE y base de datos:

- Configurar el driver JDBC en NetBeans para conectar la aplicación con la base de datos.

4. Pruebas y validación:

- Configuración de un entorno local para pruebas, incluyendo datos de muestra para validar el correcto funcionamiento del sistema.

DESCRIPCIÓN GENERAL DEL SISTEMA

¿Qué hace el sistema?

El sistema centraliza y automatiza diversas funciones administrativas y operativas, permitiendo a los usuarios llevar a cabo las siguientes tareas:

1. Registro y Gestión de Personal Administrativo y Médico:

- El administrador tiene la capacidad de registrar y gestionar la información de médicos, secretarias y personal de caja. Esto incluye datos personales, roles y permisos dentro del sistema.

2. Registro y Gestión de Pacientes:

- La secretaria registra nuevos pacientes, actualiza su información y asigna consultas médicas según la disponibilidad de los especialistas.

3. Atención Médica y Prescripción:

- Los médicos utilizan el sistema para gestionar las consultas asignadas, registrar los diagnósticos y emitir recetas médicas directamente en el sistema.

4. Gestión de Cobros:

- El personal de caja realiza el cobro de los servicios brindados, genera facturas y administra los registros de pagos.

Principales Características

- **Interfaz intuitiva y centrada en el usuario:**

Diseñada para facilitar el uso por parte del personal administrativo y médico, minimizando la curva de aprendizaje.

- **Gestión centralizada de información:**

Base de datos central que almacena y organiza de manera segura los registros de pacientes, médicos, consultas y servicios.

- **Flujo de trabajo optimizado:**

Permite una transición eficiente entre las diferentes etapas del proceso (registro de pacientes, asignación de consultas, atención médica y cobro).

- **Seguridad y control de acceso:**

Implementación de roles y permisos para garantizar que cada usuario tenga acceso únicamente a las funcionalidades relevantes para su rol.

- **Generación de reportes:**

Funcionalidad para obtener reportes sobre consultas realizadas y estado de los pacientes.

Objetivos del Sistema

- 1. Optimizar la gestión administrativa:**

Facilitar el registro y organización de la información de médicos, secretarías y personal de caja.

- 2. Mejorar la experiencia del paciente:**

Agilizar el registro, asignación de consultas y atención médica.

- 3. Incrementar la eficiencia operativa:**

Reducir tiempos y errores en la atención médica y administrativa mediante la automatización de procesos.

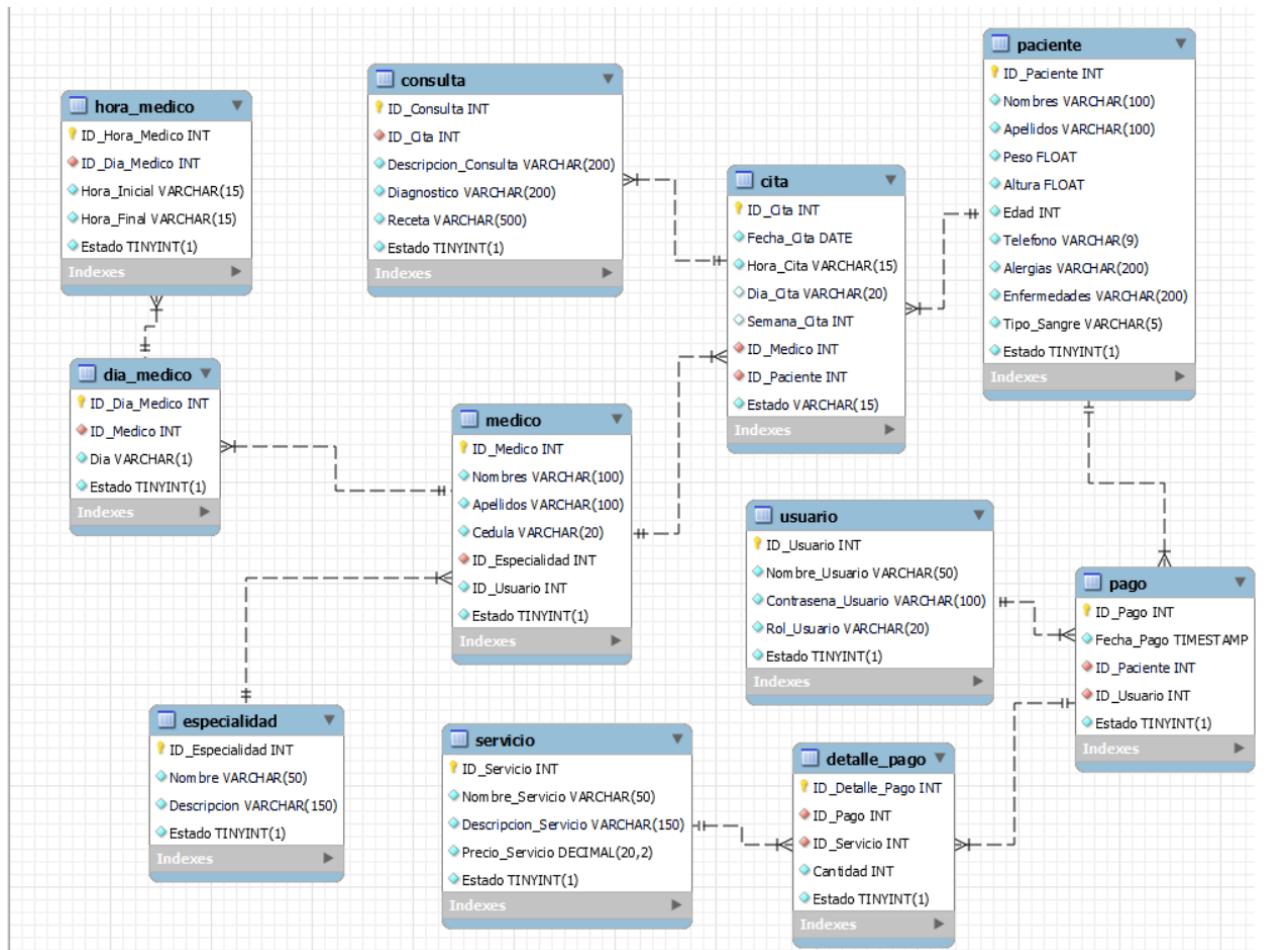
- 4. Asegurar la transparencia financiera:**

Automatizar los cobros y generar un historial claro y detallado de los servicios facturados.

- 5. Proveer una solución escalable y sostenible:**

Diseñar un sistema capaz de adaptarse a futuros requerimientos y crecimiento de la clínica.

ARQUITECTURA DEL SISTEMA



ENTORNO TÉCNICO

El desarrollo del sistema de gestión de información para la Clínica Sagrado Corazón se llevó a cabo utilizando un entorno técnico robusto y herramientas especializadas que garantizan la calidad, escalabilidad y sostenibilidad del software. A continuación, se describen los lenguajes, frameworks, herramientas y plataformas utilizadas:

Lenguajes de Programación

1. Java:

Utilizado como lenguaje principal para el desarrollo de la lógica del sistema y las interfaces gráficas. Su robustez y versatilidad lo hacen ideal para aplicaciones de escritorio y módulos escalables.

2. SQL:

Empleado para la creación y gestión de bases de datos relacionales en MySQL, incluyendo consultas, procedimientos almacenados y optimización de operaciones.

3. Bizagi Modeler:

Usado para modelar y documentar los procesos internos de la clínica, permitiendo un análisis visual y detallado del flujo de trabajo.

4. Librerías externas:

- **JDBC Driver:** Para la conexión entre la aplicación desarrollada en Java y la base de datos MySQL.
- **JasperReports:** Utilizada para la generación de reportes personalizados, como facturas y reportes médicos.
- **Apache Commons:** Para operaciones avanzadas como manejo de archivos y utilidades de desarrollo.

Modelado y Diseño

1. Diagrama Entidad-Relación (ER):

Herramienta fundamental para diseñar la estructura de la base de datos, asegurando que las tablas, relaciones y restricciones reflejan las necesidades del sistema.

2. Diagramas UML:

Utilizados para modelar y documentar la arquitectura del sistema, incluyendo:

- Diagrama de casos de uso.
- Diagrama de clases.
- Diagrama de actividades.

Metodología de Desarrollo

1. Extreme Programming (XP):

Metodología ágil seleccionada para guiar el desarrollo del proyecto, enfocándose en:

- Desarrollo iterativo con entregas parciales y continuas.
- Refactorización constante del código para mantener la calidad.
- Comunicación cercana con los interesados.
- Pruebas frecuentes para garantizar un software libre de errores.

ESTANDARES DE CODIFICACION

MUESTRA DE ESTRUCTURA PRINCIPAL DEL CÓDIGO

1. Clases

- Contiene las clases relacionadas con la lógica de negocio del sistema, como la gestión de pacientes, médicos, citas y servicios.
 - Paciente.java: Representa la entidad paciente con sus atributos y métodos.
 - Medico.java: Define los datos y comportamientos asociados a los médicos.
 - Cita.java: Gestiona la información de las citas médicas.

2. Credenciales

- Almacena configuraciones relacionadas con la conexión a la base de datos o autenticación de usuarios.
 - ConexionDB.java: Clase para manejar la conexión a la base de datos MySQL.
 - Config.properties: Archivo para almacenar credenciales o configuraciones sensibles de manera segura.

3. Images

- Carpeta destinada a almacenar recursos gráficos utilizados en la interfaz de usuario, como íconos, logotipos o imágenes decorativas.
- Ejemplo de contenido:
 - logo_hospital.png: Imagen del logotipo del hospital.
 - icono_cita.png: Ícono representativo de las citas médicas.

4. Inicio

- Contiene las clases o ventanas iniciales del sistema, como la pantalla de inicio de sesión o el menú principal.
 - Login.java: Clase que implementa la funcionalidad de inicio de sesión.
 - MenuPrincipal.java: Clase para mostrar el menú principal de la aplicación.

5. Reportes

- Incluye las clases o archivos relacionados con la generación de reportes en el sistema.
 - ReporteCitas.jasper: Plantilla para reportes de citas generados con JasperReports.

- GeneradorReportes.java: Clase encargada de gestionar la generación de reportes.

6. Ventanas

- Contiene las clases que representan las interfaces gráficas del sistema, como formularios y paneles para interactuar con el usuario.
 - VentanaRegistroPaciente.java: Interfaz para registrar nuevos pacientes.
 - VentanaAsignarCita.java: Interfaz para asignar citas médicas.

7. Libraries

- Carpeta para almacenar las librerías externas necesarias para el funcionamiento del sistema, como:
 - **JDBC Driver:** Para la conexión con MySQL.
 - **JasperReports:** Para la generación de reportes.
 - **Apache Commons:** Librerías para operaciones avanzadas.

8. Test Packages y Test Libraries

- Carpeta destinada a las pruebas del sistema.
 - **Test Packages:** Contiene las clases para pruebas unitarias o funcionales del proyecto.
 - **Test Libraries:** Librerías necesarias para ejecutar los casos de prueba (e.g., JUnit).

Errores Comunes a Evitar

1. Problemas de conexión a la base de datos:

- No olvidar cerrar conexiones JDBC después de cada operación.
- Asegurarse de manejar errores de conexión correctamente.

2. Nombres poco descriptivos:

- Evitar nombres genéricos como temp, data o obj.

3. Uso excesivo de variables globales:

- Limitar el uso de variables globales para evitar efectos colaterales inesperados.

4. Ignorar excepciones:

- No dejar bloques catch vacíos ni ignorar posibles errores.

5. Complejidad innecesaria:

- No introducir estructuras complicadas si una solución más simple es suficiente.

Buenas Prácticas

1. Código limpio y legible:

- Dividir el código en métodos pequeños y reutilizables.
- Evitar duplicación de código (principio DRY: *Don't Repeat Yourself*).
- Seguir el principio KISS (*Keep It Simple, Stupid*), manteniendo el diseño simple y comprensible.

2. Manejo de errores y excepciones:

- Usar bloques try-catch para manejar excepciones y evitar interrupciones inesperadas.
- Registrar los errores en un archivo de log para su posterior análisis.

3. Uso eficiente de recursos:

- Cerrar conexiones a la base de datos y liberar recursos en bloques finally.
- Optimizar consultas SQL para evitar operaciones innecesarias.

4. Pruebas continuas:

- Escribir pruebas unitarias para validar las funcionalidades clave.
- Probar el sistema con datos reales y casos límite.

CONFIGURACIÓN DEL ENTORNO DE DESARROLLO

Software Necesario:

- **Java Development Kit (JDK)**: Versión 11 o superior.
- **NetBeans IDE**: Versión 12.x o superior.
- **MySQL Server**: Versión 8.x.
- **MySQL Workbench**: Herramienta gráfica para el diseño y gestión de bases de datos.

- **Git:** Para el control de versiones del proyecto.
- **JasperReports Library:** Para la generación de reportes.
- **Apache Commons Library:** Utilizada para operaciones auxiliares como manejo de archivos y cadenas.

Sistema Operativo:

- Windows, macOS o Linux compatible con las herramientas mencionadas.

Configuraciones Adicionales:

- Espacio en disco: Al menos 300 GB de espacio libre para instalar las herramientas.
- Conexión a internet: Para descargar librerías y actualizaciones.

Instrucciones de Instalación

1. Instalar el JDK:

- Descargar el instalador desde [Oracle JDK](#) o [OpenJDK](#).
- Configurar la variable de entorno JAVA_HOME apuntando a la carpeta de instalación.

2. Instalar NetBeans IDE:

- Descargar el instalador desde [NetBeans](#).
- Durante la instalación, asegúrate de seleccionar soporte para Java y, opcionalmente, herramientas para bases de datos.

3. Instalar MySQL Server y Workbench:

- Descargar desde [MySQL Community Downloads](#).
- Configurar el servidor MySQL, asegurándote de establecer un usuario y contraseña para acceder a las bases de datos.

4. Configurar NetBeans para trabajar con MySQL:

- Descarga el conector JDBC para MySQL desde [MySQL Connector/J](#).
- Agrega el .jar del conector a tu proyecto en NetBeans:
 - Haz clic derecho en el proyecto → **Properties** → **Libraries** → **Add JAR/Folder** → Selecciona el archivo descargado.

5. Integrar JasperReports:

- Descarga las librerías necesarias desde JasperReports.

- Agrega las librerías a tu proyecto siguiendo el mismo proceso que con el conector JDBC.

6. Instalar Apache Commons:

- Descarga las librerías necesarias desde [Apache Commons](#).
- Agrega las librerías a tu proyecto.

Gestión de Dependencias

En este proyecto, las dependencias se manejan manualmente debido a la naturaleza del entorno NetBeans. A continuación, se describe cómo gestionarlas:

1. Agregar Dependencias:

- Todos los archivos **.jar** (e.g., JDBC, JasperReports, Apache Commons) deben ser agregados manualmente a la carpeta **Libraries** del proyecto en NetBeans.

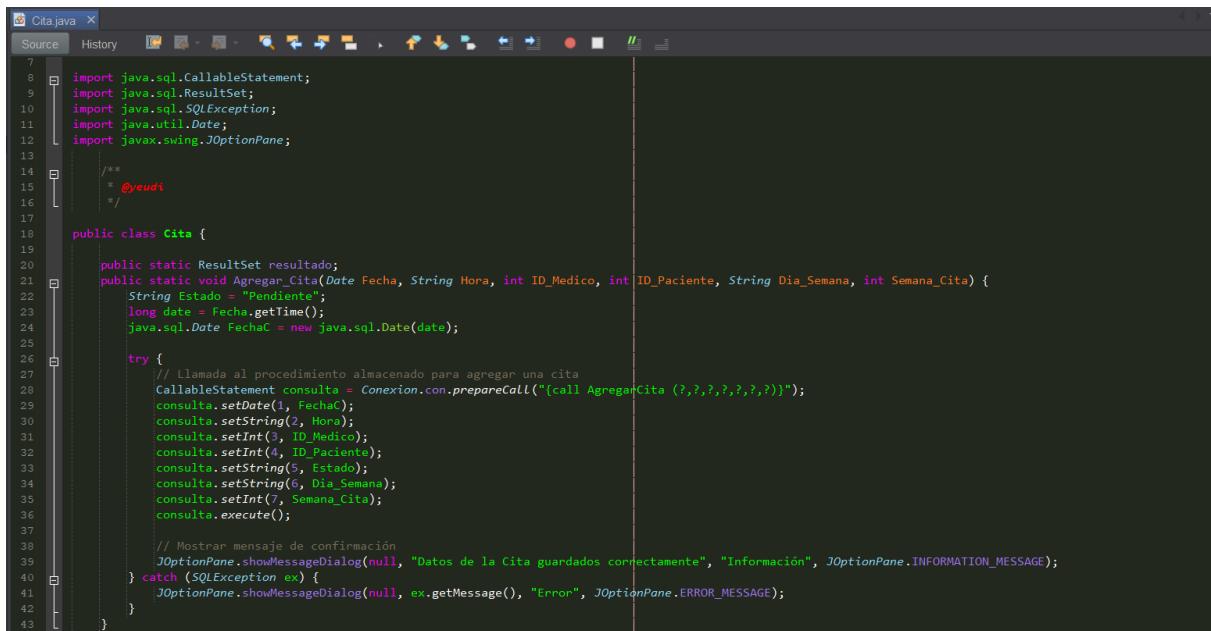
2. Actualizar Dependencias:

- Descargar las versiones más recientes desde las páginas oficiales.
- Sustituir los archivos **.jar** antiguos en la carpeta **Libraries**.

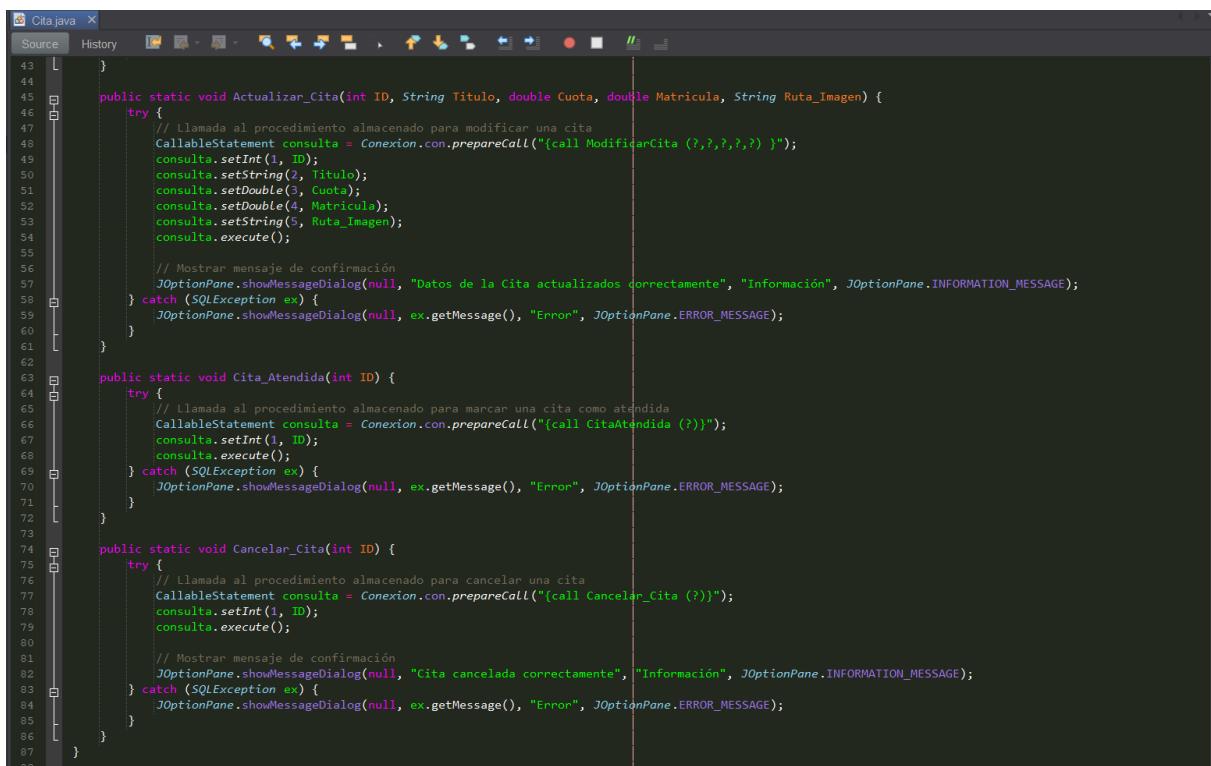
ESTRUCTURA DEL CÓDIGO (EXPLICACIÓN)

1. En esa parte del método citas se tiene registro de las consultas con la base de datos el cual existe el agregar cita , actualizar cita y el estatus de cita atendida y por último el

cancelar cita

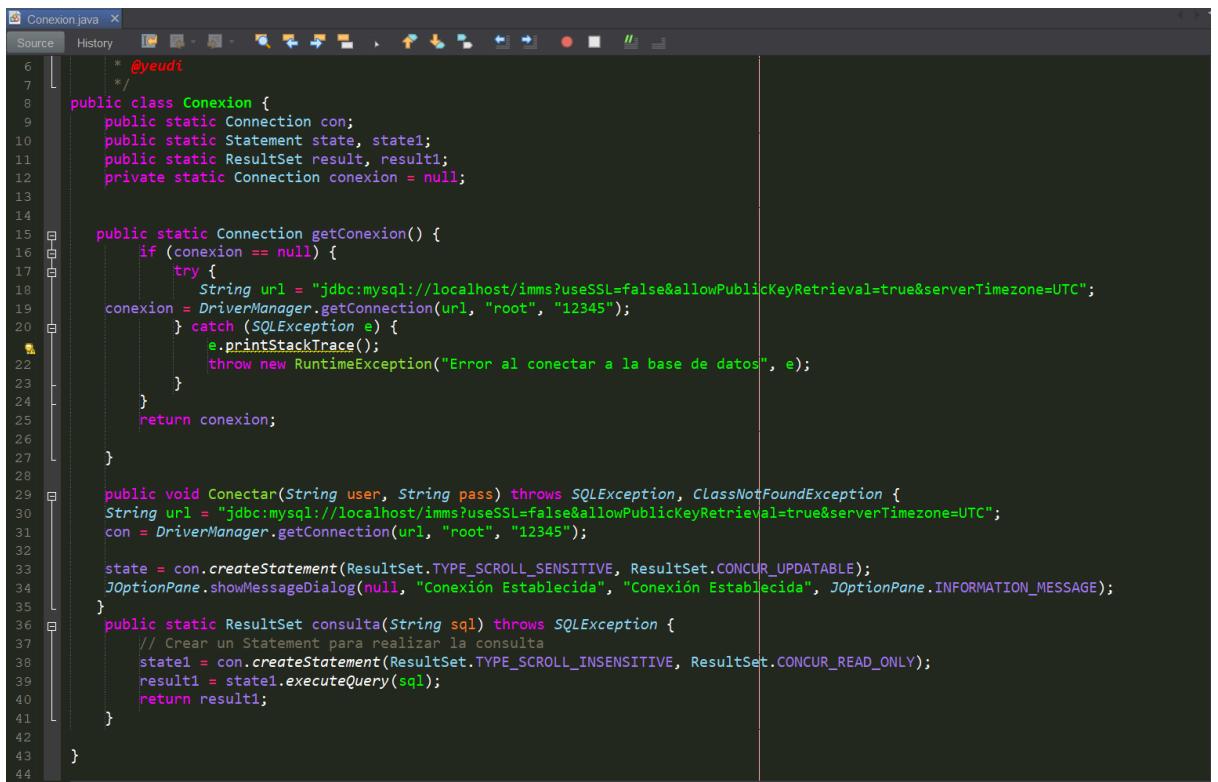


```
7 import java.sql.CallableStatement;
8 import java.sql.ResultSet;
9 import java.sql.SQLException;
10 import java.util.Date;
11 import javax.swing.JOptionPane;
12
13 /**
14 * @author
15 */
16
17 public class Cita {
18
19     public static ResultSet resultado;
20     public static void Agregar_Cita(Date Fecha, String Hora, int ID_Medico, int ID_Paciente, String Dia_Semana, int Semana_Cita) {
21         String Estado = "Pendiente";
22         long date = Fecha.getTime();
23         java.sql.Date FechaC = new java.sql.Date(date);
24
25         try {
26             // Llamada al procedimiento almacenado para agregar una cita
27             CallableStatement consulta = Conexion.con.prepareCall("{call AgregarCita (?, ?, ?, ?, ?, ?, ?)}");
28             consulta.setDate(1, FechaC);
29             consulta.setString(2, Hora);
30             consulta.setInt(3, ID_Medico);
31             consulta.setInt(4, ID_Paciente);
32             consulta.setString(5, Estado);
33             consulta.setString(6, Dia_Semana);
34             consulta.setInt(7, Semana_Cita);
35             consulta.execute();
36
37             // Mostrar mensaje de confirmación
38             JOptionPane.showMessageDialog(null, "Datos de la Cita guardados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
39         } catch (SQLException ex) {
40             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
41         }
42     }
43 }
```



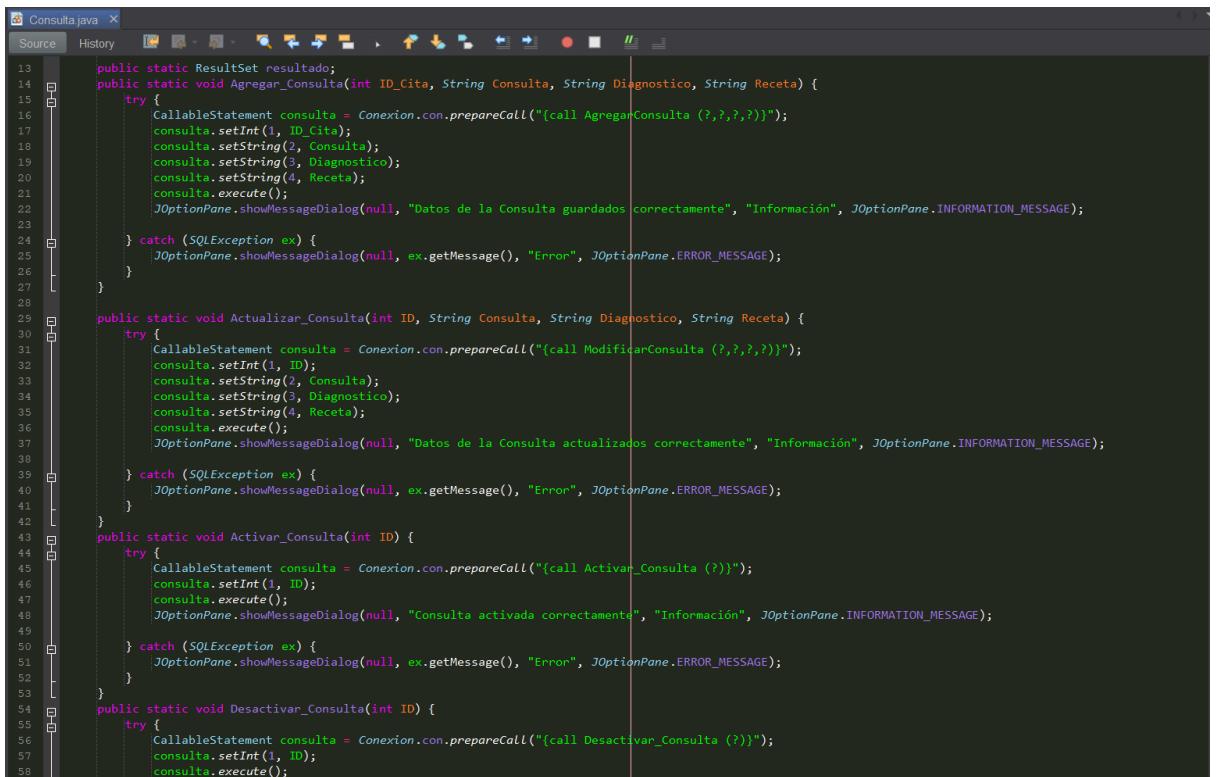
```
43 }
44
45     public static void Actualizar_Cita(int ID, String Titulo, double Cuota, double Matricula, String Ruta_Imagen) {
46         try {
47             // Llamada al procedimiento almacenado para modificar una cita
48             CallableStatement consulta = Conexion.con.prepareCall("{call ModificarCita (?, ?, ?, ?, ?)}");
49             consulta.setInt(1, ID);
50             consulta.setString(2, Titulo);
51             consulta.setDouble(3, Cuota);
52             consulta.setDouble(4, Matricula);
53             consulta.setString(5, Ruta_Imagen);
54             consulta.execute();
55
56             // Mostrar mensaje de confirmación
57             JOptionPane.showMessageDialog(null, "Datos de la Cita actualizados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
58         } catch (SQLException ex) {
59             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
60         }
61     }
62
63     public static void Cita_Atendida(int ID) {
64         try {
65             // Llamada al procedimiento almacenado para marcar una cita como atendida
66             CallableStatement consulta = Conexion.con.prepareCall("{call CitaAtendida (?)}");
67             consulta.setInt(1, ID);
68             consulta.execute();
69         } catch (SQLException ex) {
70             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
71         }
72     }
73
74     public static void Cancelar_cita(int ID) {
75         try {
76             // Llamada al procedimiento almacenado para cancelar una cita
77             CallableStatement consulta = Conexion.con.prepareCall("{call Cancelar_Cita (?)}");
78             consulta.setInt(1, ID);
79             consulta.execute();
80
81             // Mostrar mensaje de confirmación
82             JOptionPane.showMessageDialog(null, "Cita cancelada correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
83         } catch (SQLException ex) {
84             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
85         }
86     }
87 }
```

2. Se encuentra el apartado de la conexión con la base de datos , agregando el root , password y nombre de la base de datos



```
6  * @yeudi
7  */
8  public class Conexion {
9      public static Connection con;
10     public static Statement state, state1;
11     public static ResultSet result, result1;
12     private static Connection conexion = null;
13
14
15     public static Connection getConexion() {
16         if (conexion == null) {
17             try {
18                 String url = "jdbc:mysql://localhost/imms?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC";
19                 conexion = DriverManager.getConnection(url, "root", "12345");
20             } catch (SQLException e) {
21                 e.printStackTrace();
22                 throw new RuntimeException("Error al conectar a la base de datos", e);
23             }
24         }
25         return conexion;
26     }
27
28
29     public void Conectar(String user, String pass) throws SQLException, ClassNotFoundException {
30         String url = "jdbc:mysql://localhost/imms?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC";
31         con = DriverManager.getConnection(url, "root", "12345");
32
33         state = con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
34         JOptionPane.showMessageDialog(null, "Conexión Establecida", "Conexión Establecida", JOptionPane.INFORMATION_MESSAGE);
35     }
36
37     public static ResultSet consulta(String sql) throws SQLException {
38         // Crear un Statement para realizar la consulta
39         state1 = con.createStatement	ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
40         result1 = state1.executeQuery(sql);
41         return result1;
42     }
43 }
44
```

3. Se encuentran los métodos agregar consulta , actualizar , desactivar y activar consulta con sus atributos integrados desde la base de datos.



```
13
14
15     public static ResultSet resultado;
16     public static void Agregar_Consulta(int ID_Cita, String Consulta, String Diagnostico, String Receta) {
17         try {
18             CallableStatement consulta = Conexion.con.prepareCall("{call Agregar_Consulta (?, ?, ?, ?)}");
19             consulta.setInt(1, ID_Cita);
20             consulta.setString(2, Consulta);
21             consulta.setString(3, Diagnostico);
22             consulta.setString(4, Receta);
23             consulta.execute();
24             JOptionPane.showMessageDialog(null, "Datos de la Consulta guardados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
25
26         } catch (SQLException ex) {
27             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
28         }
29
30     public static void Actualizar_Consulta(int ID, String Consulta, String Diagnostico, String Receta) {
31         try {
32             CallableStatement consulta = Conexion.con.prepareCall("{call ModificarConsulta (?, ?, ?, ?)}");
33             consulta.setInt(1, ID);
34             consulta.setString(2, Consulta);
35             consulta.setString(3, Diagnostico);
36             consulta.setString(4, Receta);
37             consulta.execute();
38             JOptionPane.showMessageDialog(null, "Datos de la Consulta actualizados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
39
40         } catch (SQLException ex) {
41             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
42         }
43     public static void Activar_Consulta(int ID) {
44         try {
45             CallableStatement consulta = Conexion.con.prepareCall("{call Activar_Consulta (?)}");
46             consulta.setInt(1, ID);
47             consulta.execute();
48             JOptionPane.showMessageDialog(null, "Consulta activada correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
49
50         } catch (SQLException ex) {
51             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
52         }
53     }
54     public static void Desactivar_Consulta(int ID) {
55         try {
56             CallableStatement consulta = Conexion.con.prepareCall("{call Desactivar_Consulta (?)}");
57             consulta.setInt(1, ID);
58             consulta.execute();
59         }
60     }
61 }
```

4. Este es el apartado del día médico el cual se encuentran los métodos de agregar , actualizar , activar y desactivar el día del medico .

```

14
15     public static void Agregar_Dia_Medico(int ID_Medico, String Dia) {
16         try {
17             CallableStatement consulta = Conexion.con.prepareCall("{call AgregarDia_Medico (?,?)}");
18             consulta.setInt(1, ID_Medico);
19             consulta.setString(2, Dia);
20             consulta.execute();
21         } catch (SQLException ex) {
22             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
23         }
24     }
25
26     public static void Actualizar_Dia_Medico(int ID, String Titulo, double Cuota, double Matricula, String Ruta_Imagen) {
27         try {
28             CallableStatement consulta = Conexion.con.prepareCall("{call ModificarDia_Medico (?, ?, ?, ?, ?)}");
29             consulta.setInt(1, ID);
30             consulta.setString(2, Titulo);
31             consulta.setDouble(3, Cuota);
32             consulta.setDouble(4, Matricula);
33             consulta.setString(5, Ruta_Imagen);
34             consulta.execute();
35         } catch (SQLException ex) {
36             JOptionPane.showMessageDialog(null, "Datos de la Dia_Medico Actualizados Correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
37         } catch (SQLException ex) {
38             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
39         }
40
41     public static void Activar_Dia_Medico(int ID) {
42         try {
43             CallableStatement consulta = Conexion.con.prepareCall("{call Activar_Dia_Medico (?)}");
44             consulta.setInt(1, ID);
45             consulta.execute();
46         } catch (SQLException ex) {
47             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
48         }
49     }
50
51     public static void Desactivar_Dia_Medico(int ID) {
52         try {
53             CallableStatement consulta = Conexion.con.prepareCall("{call Desactivar_Dia_Medico (?)}");
54             consulta.setInt(1, ID);
55             consulta.execute();
56         } catch (SQLException ex) {
57             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
58         }
59     }

```

5. Es la clase especialidad el cual tambien incluira el agregar , actualizar , desactivar y activar especialidad del médico

```

15
16     public static void Agregar_Especialidad(String Nombre, String Descripcion) {
17         try {
18             CallableStatement consulta = Conexion.con.prepareCall("{call AgregarEspecialidad (?,?)}");
19             consulta.setString(1, Nombre);
20             consulta.setString(2, Descripcion);
21             consulta.execute();
22         } catch (SQLException ex) {
23             JOptionPane.showMessageDialog(null, "Datos de la Especialidad guardados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
24         }
25     }
26
27     public static void Actualizar_Especialidad(int ID, String Nombre, String Descripcion) {
28         try {
29             CallableStatement consulta = Conexion.con.prepareCall("{call ModificarEspecialidad (?, ?, ?)}");
30             consulta.setInt(1, ID);
31             consulta.setString(2, Nombre);
32             consulta.setString(3, Descripcion);
33             consulta.execute();
34         } catch (SQLException ex) {
35             JOptionPane.showMessageDialog(null, "Datos de La Especialidad Actualizados Correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
36         } catch (SQLException ex) {
37             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
38         }
39     }
40
41     public static void Activar_Especialidad(int ID) {
42         try {
43             CallableStatement consulta = Conexion.con.prepareCall("{call Activar_Especialidad (?)}");
44             consulta.setInt(1, ID);
45             consulta.execute();
46         } catch (SQLException ex) {
47             JOptionPane.showMessageDialog(null, "Activado Correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
48         } catch (SQLException ex) {
49             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
50         }
51
52     public static void Desactivar_Especialidad(int ID) {
53         try {
54             CallableStatement consulta = Conexion.con.prepareCall("{call Desactivar_Especialidad (?)}");
55             consulta.setInt(1, ID);
56             consulta.execute();
57         } catch (SQLException ex) {
58             JOptionPane.showMessageDialog(null, "Desactivado Correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
59         } catch (SQLException ex) {
60             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
61         }
62     }

```

6. Esta es la clase de la hora del médico , tiene incluido el método guardar , actualizar , activa y desactiva la hora del médico.

```

15     public static void Agregar_Hora_Medico(int ID_Dia_Medico, String HoraInicio, String HoraFinal) {
16         try {
17             CallableStatement consulta = Conexion.con.prepareCall("{call AgregarHora_Medico (?, ?, ?)}");
18             consulta.setInt(1, ID_Dia_Medico);
19             consulta.setString(2, HoraInicio);
20             consulta.setString(3, HoraFinal);
21             consulta.execute();
22         } catch (SQLException ex) {
23             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
24         }
25     }
26
27     public static void Actualizar_Hora_Medico(int ID, String Titulo, double Cuota, double Matricula, String Ruta_Imagen) {
28         try {
29             CallableStatement consulta = Conexion.con.prepareCall("{call ModificarHora_Medico (?, ?, ?, ?, ?)}");
30             consulta.setInt(1, ID);
31             consulta.setString(2, Titulo);
32             consulta.setDouble(3, Cuota);
33             consulta.setDouble(4, Matricula);
34             consulta.setString(5, Ruta_Imagen);
35             consulta.execute();
36         } catch (SQLException ex) {
37             JOptionPane.showMessageDialog(null, "Datos de La Hora_Medico Actualizados Correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
38         } catch (Exception ex) {
39             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
40         }
41     }
42     public static void Activar_Desactivar_Hora_Medico(int ID) {
43         try {
44             String estado = null;
45             resultado = Conexion.consulta("Select Estado_Hora_Medico from revista where ID_Hora_Medico = " + ID);
46             while (resultado.next()) {
47                 estado = resultado.getString(1);
48             }
49
50             if ("Activo".equals(estado)) {
51                 CallableStatement consulta = Conexion.con.prepareCall("{call Desactivar_Hora_Medico (?)}");
52                 consulta.setInt(1, ID);
53                 consulta.execute();
54             } else {
55                 JOptionPane.showMessageDialog(null, "Hora_Medico Desactivada Correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
56             }
57         } catch (SQLException ex) {
58             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
59         }
60     }

```

7. En la clase médico existe el apartado de agregar usuario médico el cual se hace el nuevo registro de usuario como médico incluido con los métodos actualizar , guardar , activar y desactivar medico

```

6     import java.sql.CallableStatement;
7     import javax.swing.JOptionPane;
8
9     /**
10      * @author
11  */
12  public class Medico {
13
14      public static ResultSet resultado;
15
16      public static void Agregar_Medico(String Nombre, String Apellido, String Cedula, int ID_Especialidad) {
17          try {
18              CallableStatement consulta = Conexion.con.prepareCall("{call AgregarMedico (?, ?, ?, ?)}");
19              consulta.setString(1, Nombre);
20              consulta.setString(2, Apellido);
21              consulta.setString(3, Cedula);
22              consulta.setInt(4, ID_Especialidad);
23              consulta.execute();
24          } catch (SQLException ex) {
25              JOptionPane.showMessageDialog(null, "Datos del Médico guardados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
26          } catch (Exception ex) {
27              JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
28          }
29      }
30
31      public static void Actualizar_Medico(int ID, String Nombre, String Apellido, String Cedula, int ID_Especialidad) {
32          try {
33              CallableStatement consulta = Conexion.con.prepareCall("{call ModificarMedico (?, ?, ?, ?, ?)}");
34              consulta.setInt(1, ID);
35              consulta.setString(2, Nombre);
36              consulta.setString(3, Apellido);
37              consulta.setString(4, Cedula);
38              consulta.setInt(5, ID_Especialidad);
39              consulta.execute();
40          } catch (SQLException ex) {
41              JOptionPane.showMessageDialog(null, "Datos del Médico actualizados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
42          } catch (Exception ex) {
43              JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
44          }
45      }
46
47      public static void Usuario_Medico(int ID_M, int ID_U, String Usuario) {
48          try {
49              CallableStatement consulta = Conexion.con.prepareCall("{call UsuarioMedico (?, ?)}");
50              consulta.setInt(1, ID_M);
51              consulta.setInt(2, ID_U);
52              consulta.execute();
53          } catch (SQLException ex) {
54              JOptionPane.showMessageDialog(null, "Médico agregado con Nombre de Usuario: " + Usuario, "Información", JOptionPane.INFORMATION_MESSAGE);
55          } catch (Exception ex) {
56              JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
57          }
58      }

```

```

56     public static void Desactivar_Medico(int ID) {
57         try {
58             CallableStatement consulta = Conexion.con.prepareCall("{call Desactivar_Medico (?)}");
59             consulta.setInt(1, ID);
60             consulta.execute();
61             JOptionPane.showMessageDialog(null, "Desactivado Correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
62         } catch (SQLException ex) {
63             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
64         }
65     }
66
67     public static void Activar_Medico(int ID) {
68         try {
69             CallableStatement consulta = Conexion.con.prepareCall("{call Activar_Medico (?)}");
70             consulta.setInt(1, ID);
71             consulta.execute();
72             JOptionPane.showMessageDialog(null, "Activado Correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
73         } catch (SQLException ex) {
74             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
75         }
76     }
77 }

```

8. Esta es la clase del paciente en donde tiene el método agregar y actualizar de igual manera activar y desactivar.

```

12     public static ResultSet resultado;
13
14     public static void Agregar_Paciente(String Nombre, String Apellido, float Peso, float Altura,
15                                         int Edad, String Alergias, String Enfermedades, String TipoSangre, String Telefono) {
16         try {
17             CallableStatement consulta = Conexion.con.prepareCall("{call AgregarPaciente (?, ?, ?, ?, ?, ?, ?, ?)}");
18             consulta.setString(1, Nombre);
19             consulta.setString(2, Apellido);
20             consulta.setFloat(3, Peso);
21             consulta.setFloat(4, Altura);
22             consulta.setInt(5, Edad);
23             consulta.setString(6, Alergias);
24             consulta.setString(7, Enfermedades);
25             consulta.setString(8, TipoSangre);
26             consulta.setString(9, Telefono);
27             consulta.execute();
28             JOptionPane.showMessageDialog(null, "Datos del Paciente guardados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
29         } catch (SQLException ex) {
30             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
31         }
32     }
33
34
35     public static void Actualizar_Paciente(int ID, String Nombre, String Apellido, float Peso, float Altura,
36                                         int Edad, String Alergias, String Enfermedades, String TipoSangre, String Telefono) {
37         try {
38             CallableStatement consulta = Conexion.con.prepareCall("{call ModificarPaciente (?, ?, ?, ?, ?, ?, ?, ?, ?)}");
39             consulta.setInt(1, ID);
40             consulta.setString(2, Nombre);
41             consulta.setString(3, Apellido);
42             consulta.setFloat(4, Peso);
43             consulta.setFloat(5, Altura);
44             consulta.setInt(6, Edad);
45             consulta.setString(7, Alergias);
46             consulta.setString(8, Enfermedades);
47             consulta.setString(9, TipoSangre);
48             consulta.setString(10, Telefono);
49             consulta.execute();
50             JOptionPane.showMessageDialog(null, "Datos del Paciente actualizados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
51         } catch (SQLException ex) {
52             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
53         }
54     }

```

```

55
56     public static void Activar_Paciente(int ID) {
57         try {
58             CallableStatement consulta = Conexion.con.prepareCall("{call Activar_Paciente (?)}");
59             consulta.setInt(1, ID);
60             consulta.execute();
61             JOptionPane.showMessageDialog(null, "Activado correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
62         } catch (SQLException ex) {
63             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
64         }
65     }
66
67     public static void Desactivar_Paciente(int ID) {
68         try {
69             CallableStatement consulta = Conexion.con.prepareCall("{call Desactivar_Paciente (?)}");
70             consulta.setInt(1, ID);
71             consulta.execute();
72             JOptionPane.showMessageDialog(null, "Desactivado correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
73         } catch (SQLException ex) {
74             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
75         }
76     }
77 }

```

9. Esta es la clase pago el cual incluye los métodos agregar -detalle , actualizar y cancelar pagos

```

22     public static void Agregar_Pago(int ID_Paciente, int ID_Usuario) {
23         try {
24             CallableStatement consulta = Conexion.con.prepareCall("{call AgregarPago (?,?)}");
25             consulta.setInt(1, ID_Paciente);
26             consulta.setInt(2, ID_Usuario);
27             consulta.execute();
28             JOptionPane.showMessageDialog(null, "Datos del Pago guardado correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
29         } catch (SQLException ex) {
30             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
31         }
32     }
33
34     public static void Agregar_DetellePago(int ID_Pago, int ID_Servicio, int Cantidad) {
35         try {
36             CallableStatement consulta = Conexion.con.prepareCall("{call AgregarDetallePago (?, ?, ?)}");
37             consulta.setInt(1, ID_Pago);
38             consulta.setInt(2, ID_Servicio);
39             consulta.setInt(3, Cantidad);
40             consulta.execute();
41         } catch (SQLException ex) {
42             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
43         }
44     }
45     public static void Actualizar_Pago(int ID, String Titulo,
46                                     double Cuota, double Matricula, String Ruta_Imagen) {
47         try {
48             CallableStatement consulta = Conexion.con.prepareCall("{call ModificarPago (?, ?, ?, ?, ?)}");
49             consulta.setInt(1, ID);
50             consulta.setString(2, Titulo);
51             consulta.setDouble(3, Cuota);
52             consulta.setDouble(4, Matricula);
53             consulta.setString(5, Ruta_Imagen);
54             consulta.execute();
55             JOptionPane.showMessageDialog(null, "Datos de La Pago Actualizados Correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
56         } catch (SQLException ex) {
57             JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
58         }
59     }
60     public static void Cancelar_Pago(int ID) {
61         try {
62             CallableStatement consulta = Conexion.con.prepareCall("{call Cancelar_DPago (?)}");
63             consulta.setInt(1, ID);
64         }
65     }

```

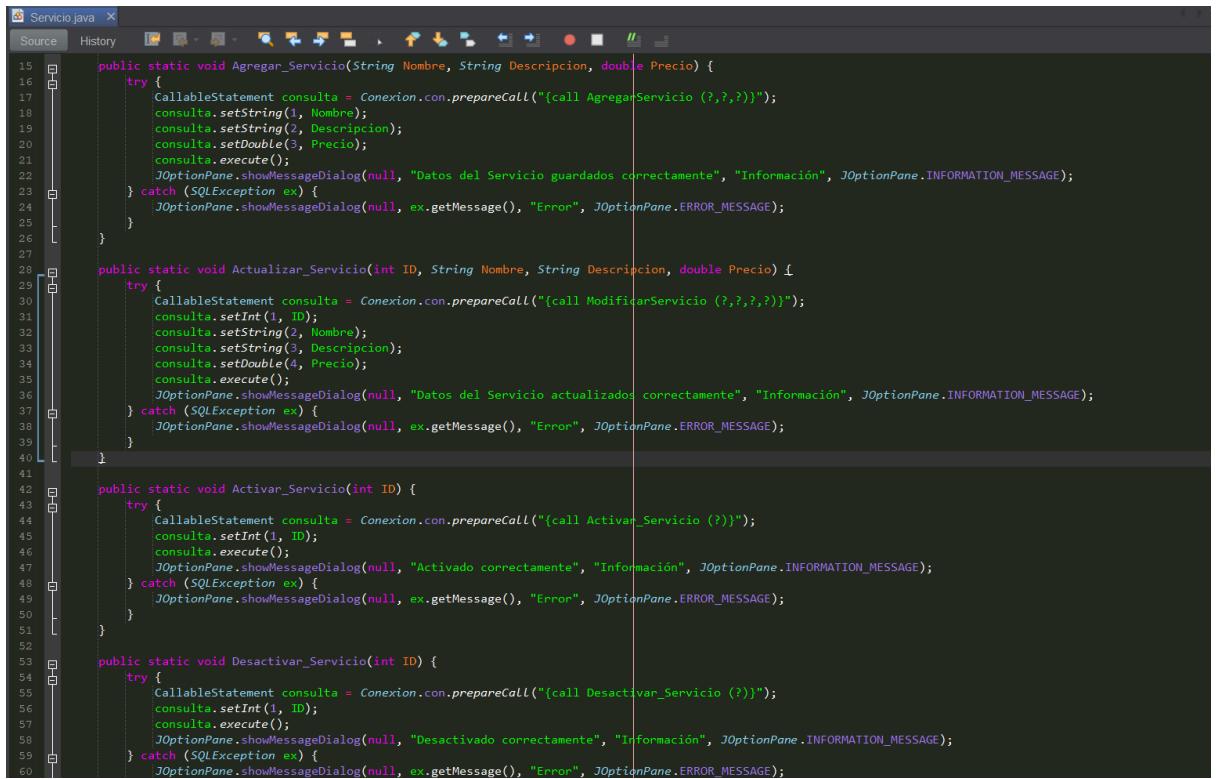
10. En esta clase seleccionamos nuestro fondo del programa mediante a carpeta img

```

1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package Clases;
7
8  import java.awt.Graphics;
9  import java.awt.Image;
10 import javax.swing.ImageIcon;
11 import javax.swing.JDesktopPane;
12
13 /**
14  * 
15  * @yeudi
16  */
17 public class Panel_Fondo extends JDesktopPane{
18
19     private Image IMG=new ImageIcon(getClass().getResource("/Images/DOC.jpg")).getImage();
20
21     @Override
22     public void paintChildren(Graphics g){
23         g.drawImage(IMG, 0, 0, getWidth(), getHeight(), this);
24         super.paintChildren(g);
25     }
26 }

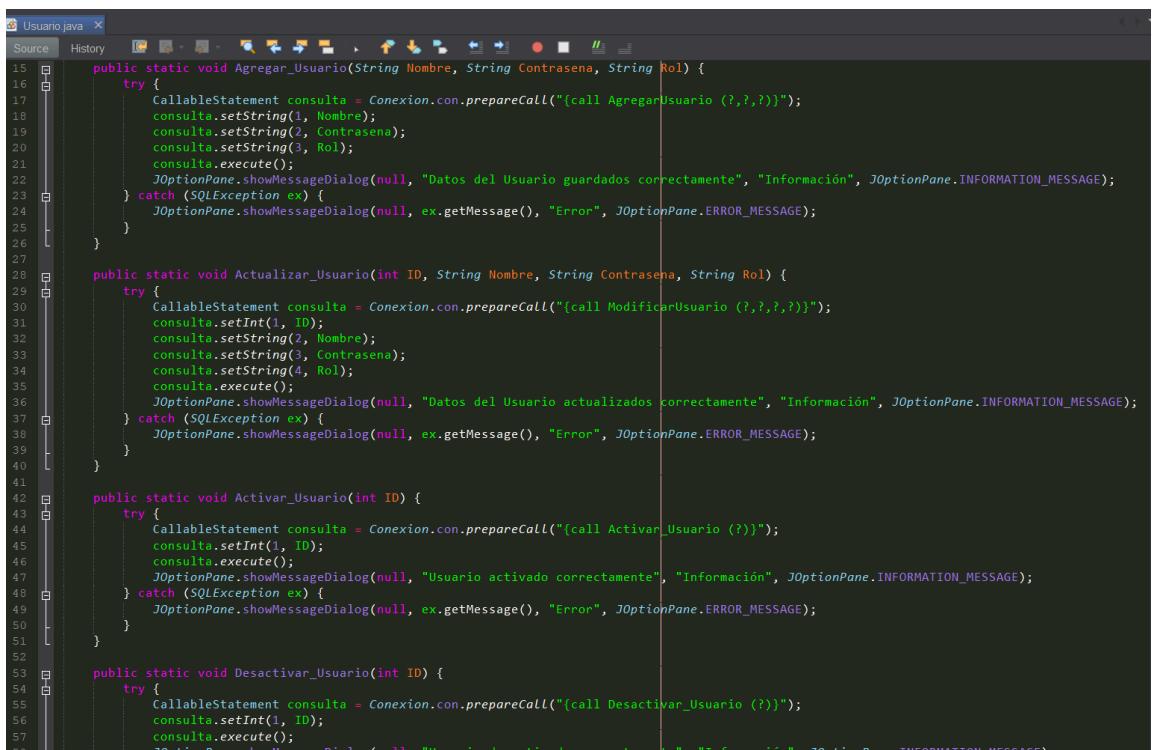
```

11. Esta es la clase servicio el cual tiene incluido los métodos agregar , actualizar , activar y desactivar servicios .



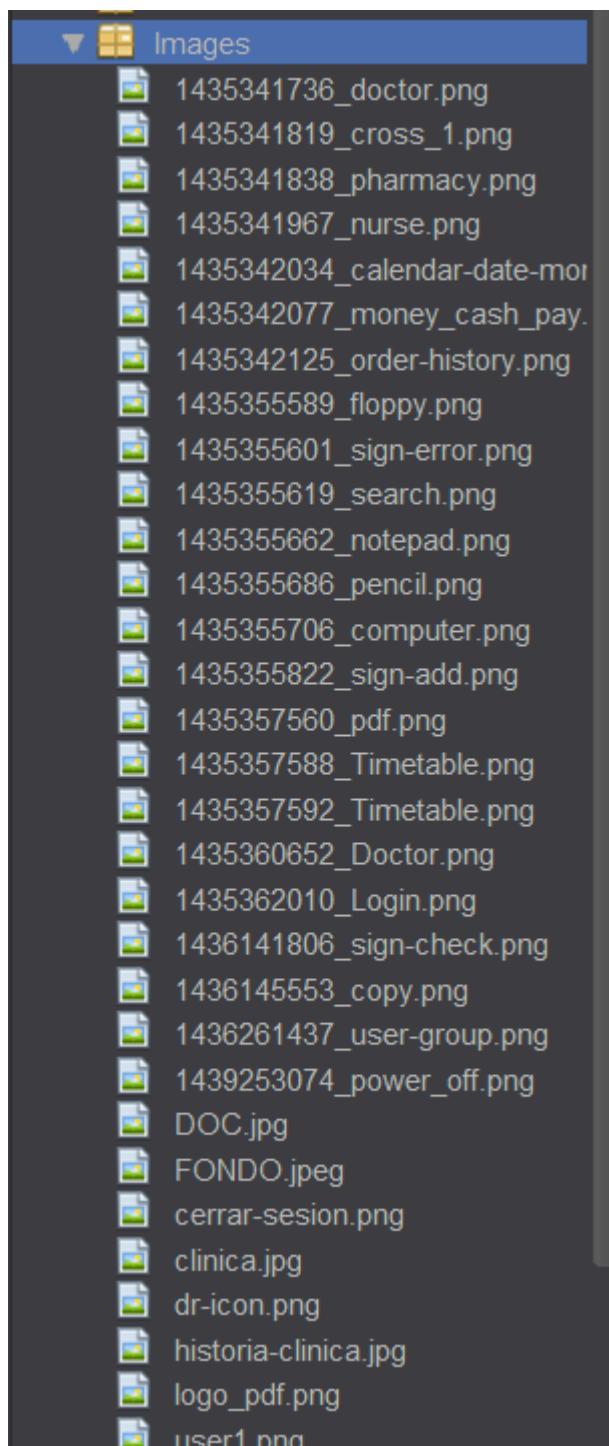
```
15 public static void Agregar_Servicio(String Nombre, String Descripcion, double Precio) {
16     try {
17         CallableStatement consulta = Conexion.con.prepareCall("{call AgregarServicio (?, ?, ?)}");
18         consulta.setString(1, Nombre);
19         consulta.setString(2, Descripcion);
20         consulta.setDouble(3, Precio);
21         consulta.execute();
22         JOptionPane.showMessageDialog(null, "Datos del Servicio guardados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
23     } catch (SQLException ex) {
24         JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
25     }
26 }
27
28 public static void Actualizar_Servicio(int ID, String Nombre, String Descripcion, double Precio) {
29     try {
30         CallableStatement consulta = Conexion.con.prepareCall("{call ModificarServicio (?, ?, ?, ?)}");
31         consulta.setInt(1, ID);
32         consulta.setString(2, Nombre);
33         consulta.setString(3, Descripcion);
34         consulta.setDouble(4, Precio);
35         consulta.execute();
36         JOptionPane.showMessageDialog(null, "Datos del Servicio actualizados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
37     } catch (SQLException ex) {
38         JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
39     }
40 }
41
42 public static void Activar_Servicio(int ID) {
43     try {
44         CallableStatement consulta = Conexion.con.prepareCall("{call Activar_Servicio (?)}");
45         consulta.setInt(1, ID);
46         consulta.execute();
47         JOptionPane.showMessageDialog(null, "Activado correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
48     } catch (SQLException ex) {
49         JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
50     }
51 }
52
53 public static void Desactivar_Servicio(int ID) {
54     try {
55         CallableStatement consulta = Conexion.con.prepareCall("{call Desactivar_Servicio (?)}");
56         consulta.setInt(1, ID);
57         consulta.execute();
58         JOptionPane.showMessageDialog(null, "Desactivado correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
59     } catch (SQLException ex) {
60         JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
61     }
62 }
```

12. Por último la clase usuario el cual es la función principal para hacer nuevos registros con los métodos agregar , actualizar , activar y desactivar usuarios

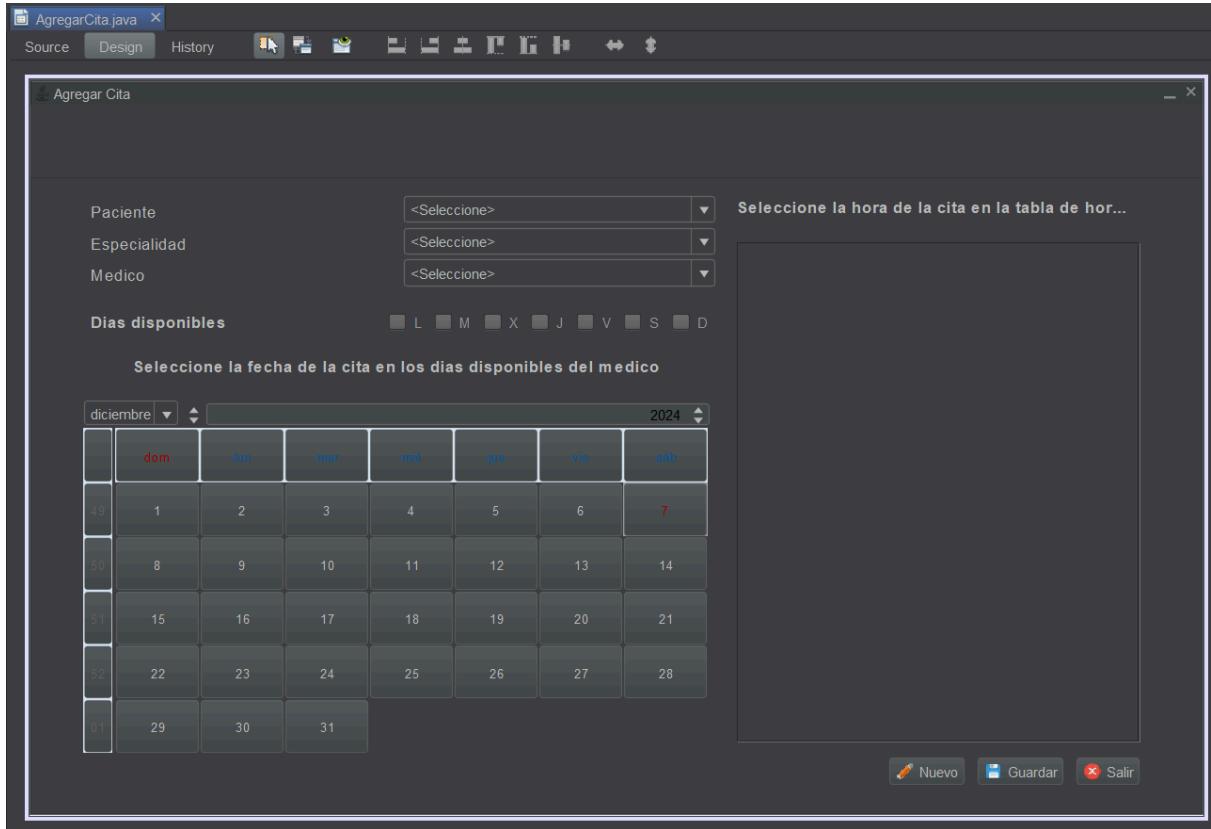


```
15 public static void Agregar_Usuario(String Nombre, String Contraseña, String Rol) {
16     try {
17         CallableStatement consulta = Conexion.con.prepareCall("{call AgregarUsuario (?, ?, ?)}");
18         consulta.setString(1, Nombre);
19         consulta.setString(2, Contraseña);
20         consulta.setString(3, Rol);
21         consulta.execute();
22         JOptionPane.showMessageDialog(null, "Datos del Usuario guardados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
23     } catch (SQLException ex) {
24         JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
25     }
26 }
27
28 public static void Actualizar_Usuario(int ID, String Nombre, String Contraseña, String Rol) {
29     try {
30         CallableStatement consulta = Conexion.con.prepareCall("{call ModificarUsuario (?, ?, ?, ?)}");
31         consulta.setInt(1, ID);
32         consulta.setString(2, Nombre);
33         consulta.setString(3, Contraseña);
34         consulta.setString(4, Rol);
35         consulta.execute();
36         JOptionPane.showMessageDialog(null, "Datos del Usuario actualizados correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
37     } catch (SQLException ex) {
38         JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
39     }
40 }
41
42 public static void Activar_Usuario(int ID) {
43     try {
44         CallableStatement consulta = Conexion.con.prepareCall("{call Activar_Usuario (?)}");
45         consulta.setInt(1, ID);
46         consulta.execute();
47         JOptionPane.showMessageDialog(null, "Usuario activado correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
48     } catch (SQLException ex) {
49         JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
50     }
51 }
52
53 public static void Desactivar_Usuario(int ID) {
54     try {
55         CallableStatement consulta = Conexion.con.prepareCall("{call Desactivar_Usuario (?)}");
56         consulta.setInt(1, ID);
57         consulta.execute();
58         JOptionPane.showMessageDialog(null, "Usuario desactivado correctamente", "Información", JOptionPane.INFORMATION_MESSAGE);
59     } catch (SQLException ex) {
60         JOptionPane.showMessageDialog(null, ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
61     }
62 }
```

13. En esta carpeta images será utilizado para los iconos de los botones del principal programa



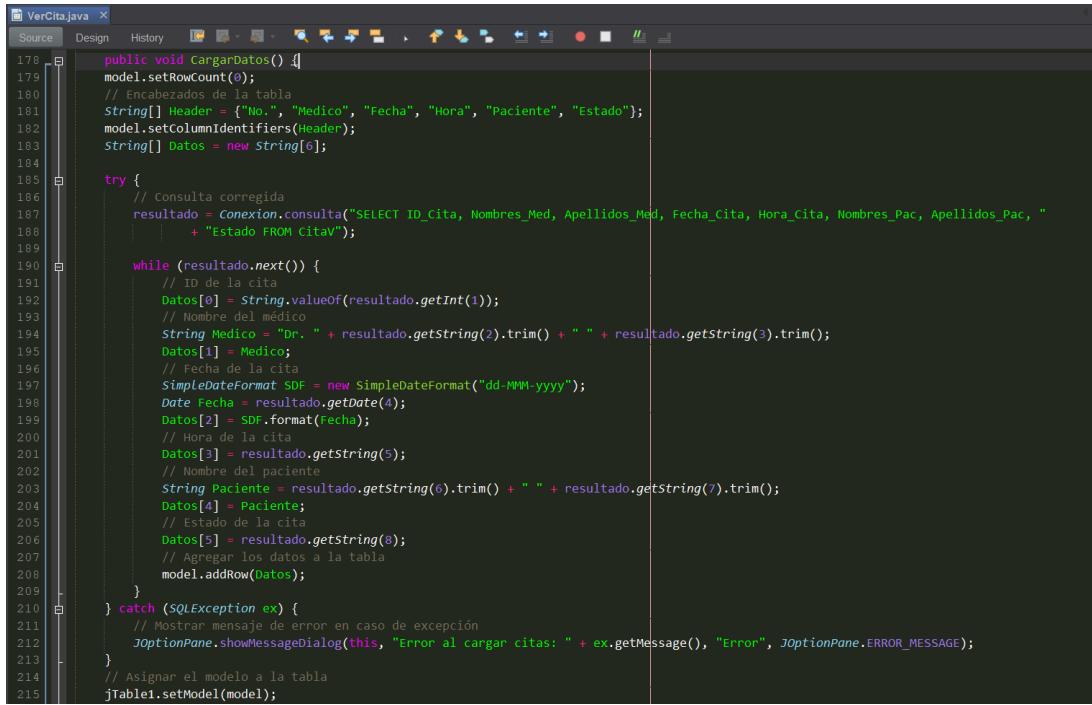
14. Este es el panel Agregar cita el cual tiene jcombobox mostrará al paciente ya registrado , su especialidad del médico y el médico seleccionado y en el horario le cargará los días disponibles que tenga el médico y se selecciona para obtener una cita para el paciente



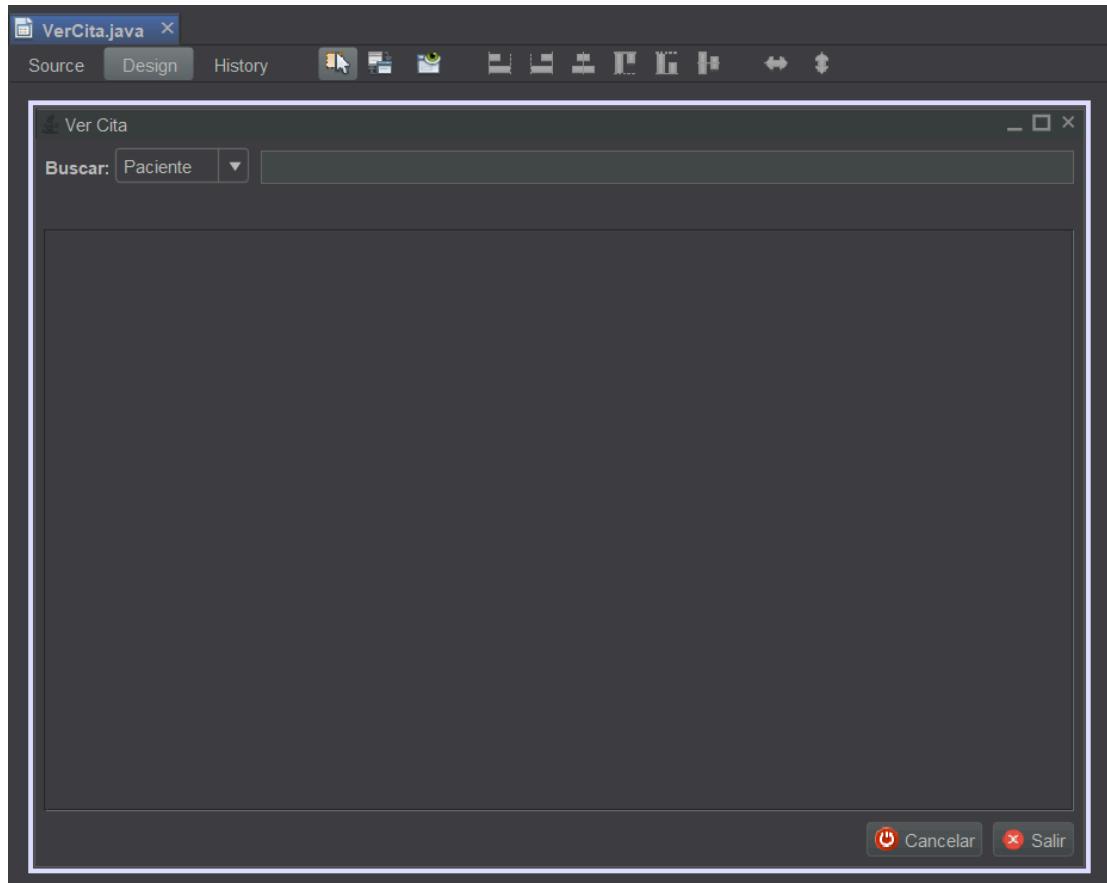
```

 319
 320     public void Guardar() {
 321         int cmbPac = cmbPaciente.getSelectedIndex();
 322         int cmbMed = cmbMedico.getSelectedIndex();
 323         Date Fecha = jCalendar1.getDate();
 324
 325         if (cmbMed == 0 || cmbPac == 0 || Fecha == null) {
 326             JOptionPane.showMessageDialog(this, "Complete todos los campos.", "Error", JOptionPane.ERROR_MESSAGE);
 327             return;
 328         }
 329         int fila = jTable1.getSelectedRow();
 330         if (fila < 0) {
 331             JOptionPane.showMessageDialog(this, "Seleccione una hora de la tabla.", "Error", JOptionPane.ERROR_MESSAGE);
 332             return;
 333         }
 334         Object horaObj = jTable1.getValueAt(fila, 0);
 335         Object estadoObj = jTable1.getValueAt(fila, 1);
 336
 337         if (horaObj == null || estadoObj == null || !"Disponible".equalsIgnoreCase(estadoObj.toString().trim())) {
 338             JOptionPane.showMessageDialog(this, "Seleccione una hora válida.", "Error", JOptionPane.ERROR_MESSAGE);
 339             return;
 340         }
 341         if (!DiaDisponible()) {
 342             return;
 343         }
 344         int ID_Paciente = ID_Pac[cmbPac];
 345         int ID_Medico = ID_Med[cmbMed];
 346         String Hora = horaObj.toString();
 347
 348         try {
 349             java.sql.Date FechaSQL = new java.sql.Date(Fecha.getTime());
 350             String query = "INSERT INTO imms.cita (Fecha_Cita, Hora_Cita, ID_Medico, ID_Paciente, Estado) VALUES (?, ?, ?, ?, ?)";
 351             java.sql.PreparedStatement ps = Conexion.getConexion().prepareStatement(query);
 352             ps.setDate(1, FechaSQL);
 353             ps.setString(2, Hora);
 354             ps.setInt(3, ID_Medico);
 355             ps.setInt(4, ID_Paciente);
 356             ps.setString(5, "Pendiente");
 357
 358             int resultado = ps.executeUpdate();
 359             if (resultado > 0) {
 360                 JOptionPane.showMessageDialog(this, "Cita registrada con éxito.", "Exito", JOptionPane.INFORMATION_MESSAGE);
 361             }
 362         }
 363     }
 364 
```

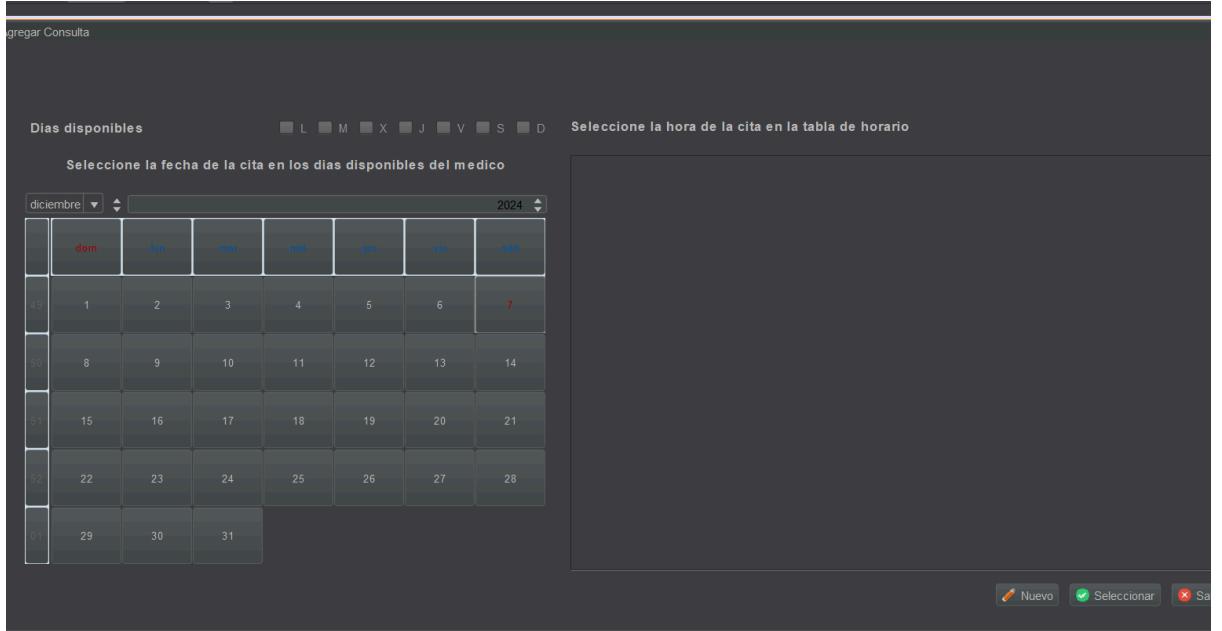
En este apartado se puede visualizar como es que el ver citas recopila información y la muestra.



```
VerCita.java
Source Design History
178 public void CargarDatos() {
179     model.setRowCount(0);
180     // Encabezados de la tabla
181     String[] Header = {"No.", "Medico", "Fecha", "Hora", "Paciente", "Estado"};
182     model.setColumnIdentifiers(Header);
183     String[] Datos = new String[6];
184
185     try {
186         // Consulta corregida
187         resultado = Conexion.consulta("SELECT ID_Cita, Nombres_Med, Apellidos_Med, Fecha_Cita, Hora_Cita, Nombres_Pac, Apellidos_Pac, "
188             + "Estado FROM Cita");
189
190         while (resultado.next()) {
191             // ID de la cita
192             Datos[0] = String.valueOf(resultado.getInt(1));
193             // Nombre del médico
194             String Medico = "Dr. " + resultado.getString(2).trim() + " " + resultado.getString(3).trim();
195             Datos[1] = Medico;
196             // Fecha de la cita
197             SimpleDateFormat SDF = new SimpleDateFormat("dd-MMM-yyyy");
198             Date Fecha = resultado.getDate(4);
199             Datos[2] = SDF.format(Fecha);
200             // Hora de la cita
201             Datos[3] = resultado.getString(5);
202             // Nombre del paciente
203             String Paciente = resultado.getString(6).trim() + " " + resultado.getString(7).trim();
204             Datos[4] = Paciente;
205             // Estado de la cita
206             Datos[5] = resultado.getString(8);
207             // Agregar los datos a la tabla
208             model.addRow(Datos);
209         }
210     } catch (SQLException ex) {
211         // Mostrar mensaje de error en caso de excepción
212         JOptionPane.showMessageDialog(this, "Error al cargar citas: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
213     }
214     // Asignar el modelo a la tabla
215     jTable1.setModel(model);
}
```



15. En este panel de agregar consulta la secretaria va seleccionar el día que se encuentra la cita con el paciente para que el médico proceda a atender al paciente



```

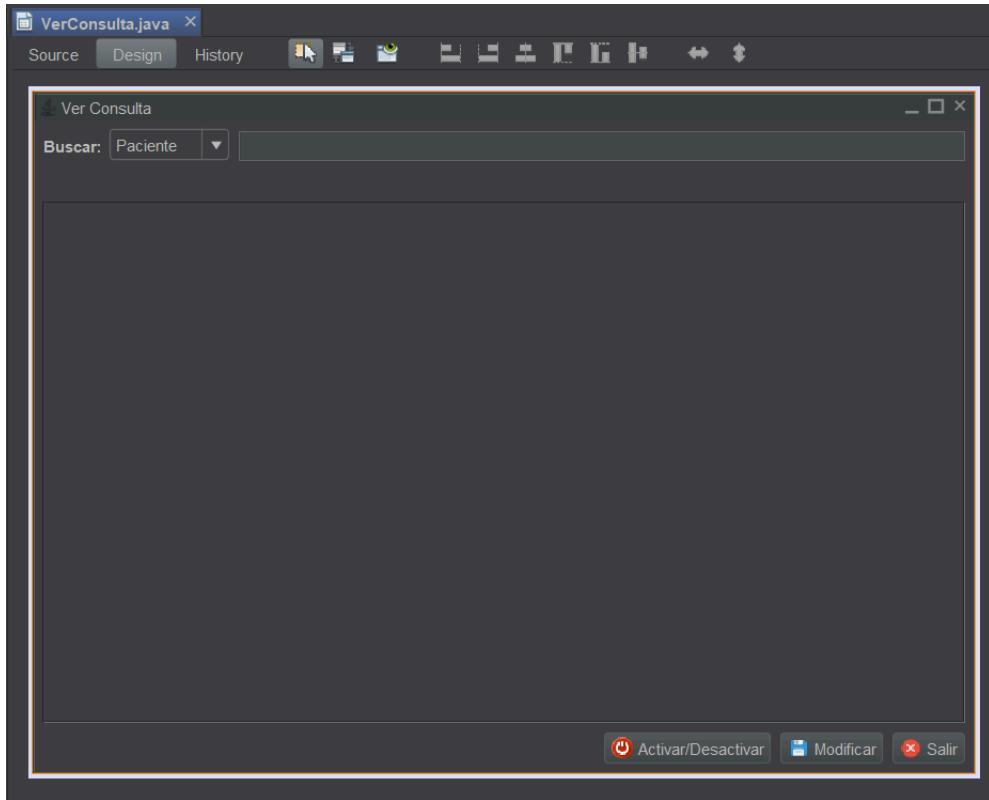
  AgregarConsulta.java x
  Source Design History
  Generated Code

  public void Guardar() {
    int fila = jTable1.getSelectedRow(); // Obtener la fila seleccionada
    if (fila >= 0) {
      String estado = ((String) model.getValueAt(fila, 1)).trim();
      String idCita = (String) model.getValueAt(fila, 2);
      // Validar si el estado es "Pendiente"
      if (!estado.endsWith("(Pendiente)")) {
        JOptionPane.showMessageDialog(this, "Seleccione una cita PENDIENTE únicamente.", "Error", JOptionPane.ERROR_MESSAGE);
        return;
      }
      // Procesar la cita pendiente
      try {
        String query = "SELECT Nombres_Pac, Apellidos_Pac, Nombres_Med, Apellidos_Med FROM CitaV WHERE ID_Cita = " + idCita;
        resultado = Conexion.consulta(query);

        String medico = "", paciente = "";
        while (resultado.next()) {
          medico = resultado.getString("Nombres_Med").trim() + " " + resultado.getString("Apellidos_Med").trim();
          paciente = resultado.getString("Nombres_Pac").trim() + " " + resultado.getString("Apellidos_Pac").trim();
        }
        // Abrir ventana de diagnóstico con datos
        AgregarDiagnostico diag = new AgregarDiagnostico(null, true);
        diag.setID_Cita(Integer.parseInt(idCita));
        diag.setPaciente(paciente);
        diag.setMedico(medico);
        diag.MostrarRptPaciente();
        diag.setVisible(true);
        diag.toFront();
      } catch (SQLException ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(this, "Error al cargar los datos de la cita.", "Error", JOptionPane.ERROR_MESSAGE);
      }
    } else {
      JOptionPane.showMessageDialog(this, "Seleccione una cita válida para continuar.", "Error", JOptionPane.ERROR_MESSAGE);
    }
  }
}

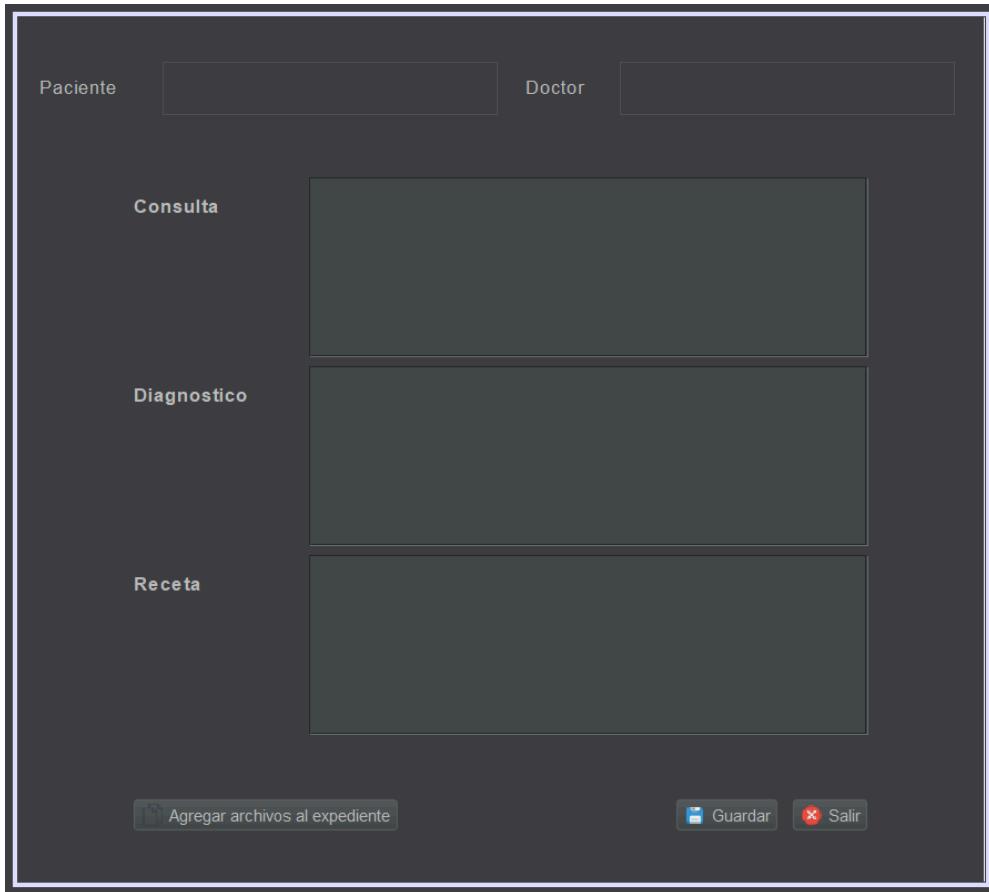
```

En este apartado se puede mostrar el apartado de ver los datos de las citas en el código se puede observar cómo se cargan los datos desde la base de datos



```
VerConsulta.java
Source Design History
182
183
184
185     ResultSet resultado;
186
187     public void CargarDatos() {
188
189         model.setRowCount(0);
190         String[] Header = {"No.", "Fecha cita", "Medico", "Paciente", "Descripcion consulta",
191                         "Diagnostico", "Receta", "Estado"};
192         model.setColumnIdentifiers(Header);
193         String[] Datos = new String[8];
194         try {
195             resultado = Conexion.consulta("Select * from ConsultaV");
196             while (resultado.next()) {
197                 Datos[0] = String.valueOf(resultado.getInt(1));
198                 SimpleDateFormat SDF = new SimpleDateFormat("dd-MMM-yyyy");
199                 Date Fecha = resultado.getDate(2);
200                 Datos[1] = SDF.format(Fecha);
201                 String Medico = "Dr. " + resultado.getString(3).trim() + " " + resultado.getString(4).trim();
202                 Datos[2] = Medico;
203                 String Paciente = resultado.getString(5).trim() + " " + resultado.getString(6).trim();
204                 Datos[3] = Paciente;
205                 Datos[4] = resultado.getString(7);
206                 Datos[5] = resultado.getString(8);
207                 Datos[6] = resultado.getString(9);
208                 boolean Estado = resultado.getBoolean(10);
209                 String Estate = "Inactivo";
210                 if (Estado) {
211                     Estate = "Activo";
212                 }
213                 Datos[7] = Estate;
214             } catch (SQLException ex) {
215             }
216         } jTable1.setModel(model);
217     }
```

16. En este apartado el médico atiende la consulta y hace un registro de una consulta , diagnóstico y receta

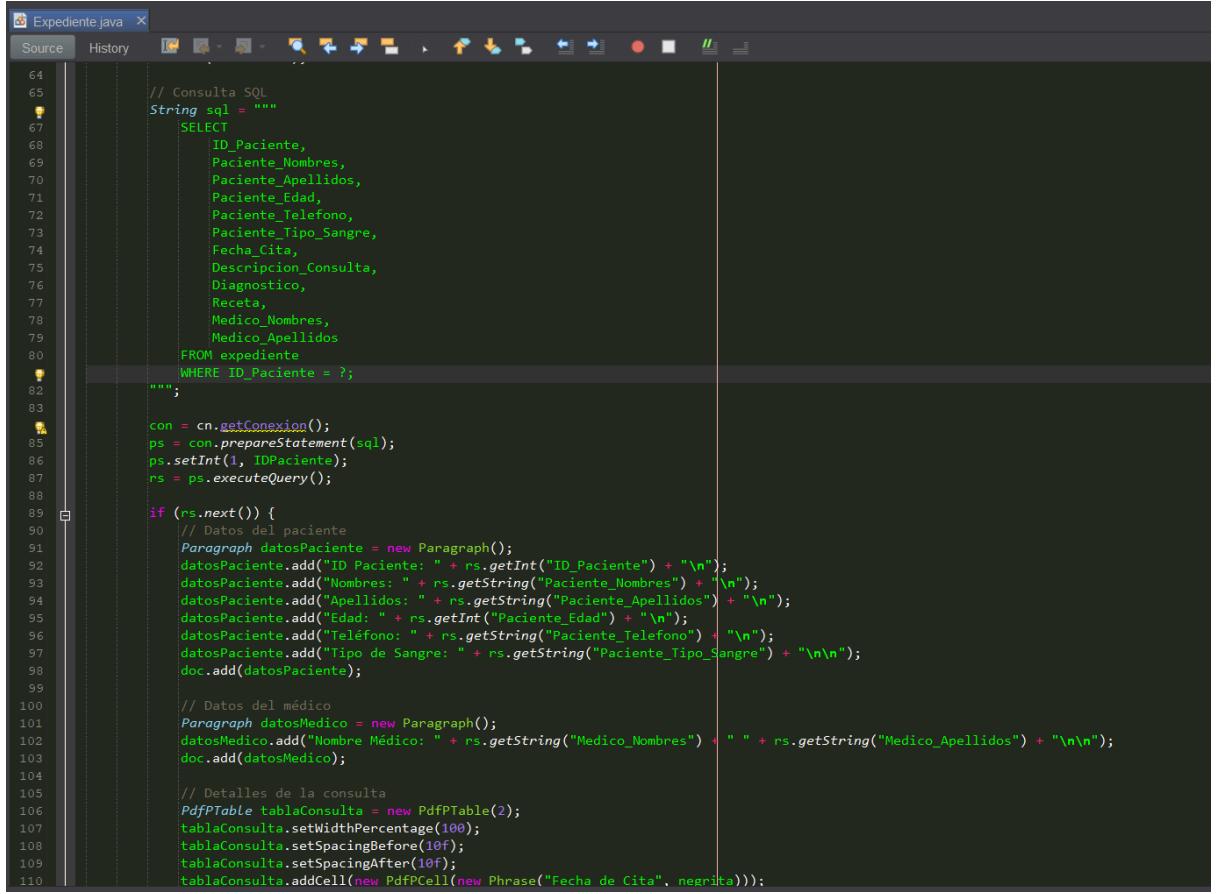


```

AregarDiagnostico.java x
Source Design History
259     txtPaciente.setText(Paciente);
260 }
261
262 public void Guardar() {
263     AC = new AgregarConsulta();
264
265     String ConsultaF = txtConsulta.getText().trim();
266     String Diagnóstico = txtDiagnóstico.getText().trim();
267     String Receta = txtReceta.getText().trim();
268     if (ID_Cita == 0 || ("").equals(ConsultaF) && ("").equals(Diagnóstico) && ("").equals(Receta)) {
269         JOptionPane.showMessageDialog(this, "Complete los campos necesarios",
270             "Complete", JOptionPane.ERROR_MESSAGE);
271         return;
272     }
273
274     Consulta.Agregar_Consulta(ID_Cita, ConsultaF, Diagnóstico, Receta);
275     Cita.Cita_Atendida(ID_cita);
276     if (AC != null) {
277         AC.dispose();
278     } else {
279         System.out.println("AC no está inicializado");
280     }
281
282     this.setVisible(false);
283     Map<String, Object> parametros = new HashMap<String, Object>();
284     parametros.put("ID_C", ID_cita);
285     File midir = new File("");
286     String reporte = midir.getAbsolutePath() + "/src/Reportes/Receta.jasper";
287     JasperPrint jp = null;
288     try {
289         jp = JasperFillManager.fillReport(reporte, parametros, Conexión.con);
290     } catch (JRException ex) {
291     }
292
293     JasperViewer view = new JasperViewer(jp, false);
294     view.setTitle("Receta - " + Paciente.trim());
295     view.setZoomRatio((float) 0.95);
296     view.setVisible(true);
297     view.setExtendedState(JFrame.MAXIMIZED_BOTH);
298     view.toFront();
299     CopiarArchivos();
300     this.dispose();
301 }
302

```

De esta manera generamos el reporte de expediente para cada paciente utilizando la consulta desde la base de datos



The screenshot shows a Java code editor with the file 'Expediente.java' open. The code is written in Java and uses JDBC to query a database for patient information. It then formats this information into a PDF document using iText. The code includes a SQL query to select patient details like ID, names, and phone number, and then processes the results to create a paragraph for each patient's data. It also handles the doctor's information and adds a table for consultation details.

```
64
65
66    // Consulta SQL
67    String sql = """
68        SELECT
69            ID_Paciente,
70            Paciente_Nombres,
71            Paciente_Apellidos,
72            Paciente_Edad,
73            Paciente_Teléfono,
74            Paciente_Tipo_Sangre,
75            Fecha_Cita,
76            Descripción_Consulta,
77            Diagnóstico,
78            Receta,
79            Médico_Nombres,
80            Médico_Apellidos
81        FROM expediente
82        WHERE ID_Paciente = ?;
83    """;

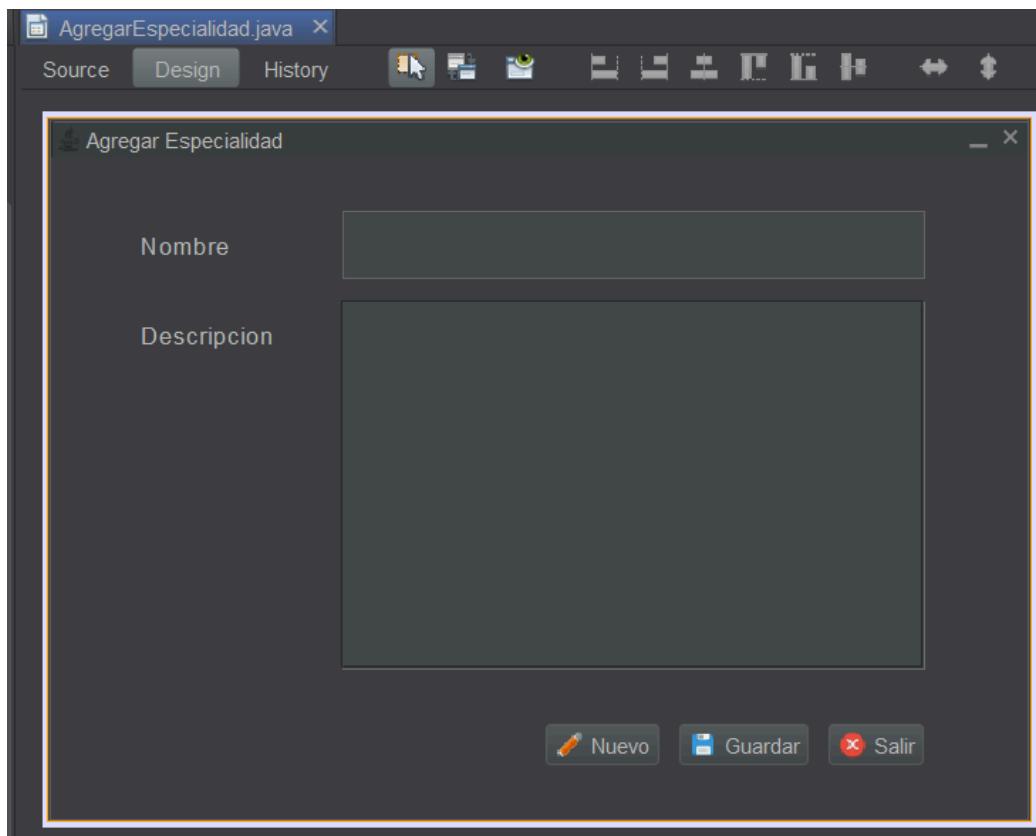
84
85    con = cn.getConnection();
86    ps = con.prepareStatement(sql);
87    ps.setInt(1, IDPaciente);
88    rs = ps.executeQuery();

89    if (rs.next()) {
90        // Datos del paciente
91        Paragraph datosPaciente = new Paragraph();
92        datosPaciente.add("ID Paciente: " + rs.getInt("ID_Paciente") + "\n");
93        datosPaciente.add("Nombres: " + rs.getString("Paciente_Nombres") + "\n");
94        datosPaciente.add("Apellidos: " + rs.getString("Paciente_Apellidos") + "\n");
95        datosPaciente.add("Edad: " + rs.getInt("Paciente_Edad") + "\n");
96        datosPaciente.add("Teléfono: " + rs.getString("Paciente_Teléfono") + "\n");
97        datosPaciente.add("Tipo de Sangre: " + rs.getString("Paciente_Tipo_Sangre") + "\n\n");
98        doc.add(datosPaciente);

99
100       // Datos del médico
101       Paragraph datosMedico = new Paragraph();
102       datosMedico.add("Nombre Médico: " + rs.getString("Médico_Nombres") + " " + rs.getString("Médico_Apellidos") + "\n\n");
103       doc.add(datosMedico);

104
105       // Detalles de la consulta
106       PdfPTable tablaConsulta = new PdfPTable(2);
107       tablaConsulta.setWidthPercentage(100);
108       tablaConsulta.setSpacingBefore(10f);
109       tablaConsulta.setSpacingAfter(10f);
110       tablaConsulta.addCell(new PdfPCell(new Phrase("Fecha de Cita", negrita))));
```

17. Al registrar especialidad son dirigidos para los médicos el cual se le asigna nombre y su descripción.

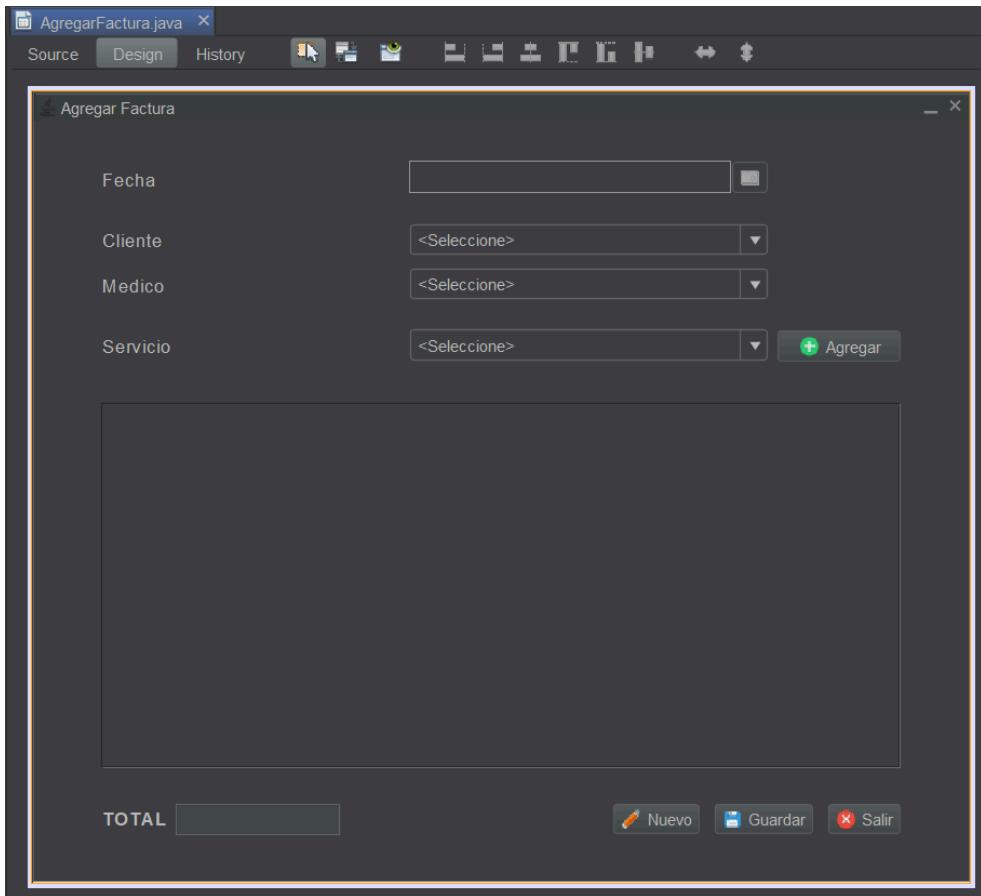


```
1 package com.yeudi.especialidad;
2
3 import javax.swing.JInternalFrame;
4 import javax.swing.JComponent;
5 import javax.swing.JDialog;
6
7 /**
8 * @yeudi
9 */
10 public class AgregarEspecialidad extends javax.swing.JInternalFrame {
11
12     public AgregarEspecialidad() {
13         initComponents();
14     }
15
16     @SuppressWarnings("unchecked")
17     [Generated Code]
18
19     public void Guardar() {
20         String Nombre = txtNombre.getText().trim();
21         String Descripcion = txtDescripcion.getText().trim();
22
23         if ("").equals(Nombre) || ("").equals(Descripcion)) {
24             JOptionPane.showMessageDialog(this, "Complete todos los campos", "Complete", JOptionPane.ERROR_MESSAGE);
25         } else {
26             Especialidad.Agregar_Especialidad(Nombre, Descripcion);
27             Limpiar();
28         }
29     }
30
31     public void Limpiar() {
32         txtDescripcion.setText("");
33         txtNombre.setText("");
34     }
35 }
```

En este apartado se pueden actualizar los datos de la especialidad ya existentes modificando los

```
25  public void setVE(VerEspecialidad VE) {
26      this.VE = VE;
27  }
28  public void CargarDatos(int ID) {
29
30      String Nombre = "";
31      String Descripcion = "";
32      try {
33          resultado = Conexion.consulta("Select * from Especialidad where ID_Especialidad = " + ID);
34          while (resultado.next()) {
35              IDD = resultado.getInt(1);
36              Nombre = resultado.getString(2);
37              Descripcion = resultado.getString(3);
38          }
39      } catch (SQLException ex) {
40      }
41      txtNombre.setText(Nombre);
42      txtDescripcion.setText(Descripcion);
43  }
44
45  @SuppressWarnings("unchecked")
46  Generated Code
150
151  public void Guardar() {
152      String Nombre = txtNombre.getText().trim();
153      String Descripcion = txtDescripcion.getText().trim();
154
155      if ("").equals(Nombre) || "".equals(Descripcion)) {
156          JOptionPane.showMessageDialog(this, "Complete todos los campos", "Complete", JOptionPane.ERROR_MESSAGE);
157      } else {
158          Especialidad.Actualizar_Especialidad(IDD, Nombre, Descripcion);
159          Limpiar();
160      }
161  }
162  public void Limpiar() {
163      VE.CargarDatos();
164      dispose();
165  }
166 }
```

18. En factura existen barras el cual en la fecha se define por la actual , también se selecciona al cliente , médico y el tipo de servicio que se le cobrará al paciente con el botón agregar y se hace el cálculo y se cobra el total

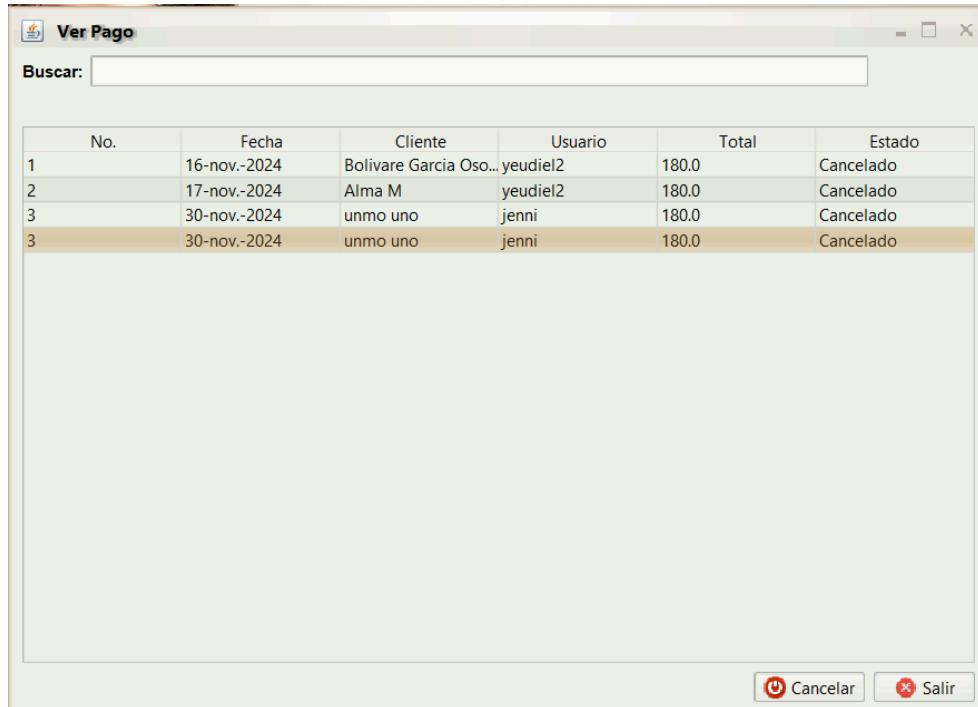


```

233
234     public void Guardar() {
235
236         int cmbPac = cmbPaciente.getSelectedIndex();
237         int cmbMed = cmbMedico.getSelectedIndex();
238
239         int ID_Paciente = ID_Pac[cmbPac];
240         int ID_Medico = ID_Med[cmbMed];
241         int Cantidad = 1;
242
243
244         int fila = jTable1.getSelectedRow();
245         if (cmbMed == 0 || cmbPac == 0) {
246             JOptionPane.showMessageDialog(this, "Complete todos los campos y seleccione correctamente",
247                                         "Agregue", JOptionPane.ERROR_MESSAGE);
248             return;
249         }
250         if (jTable1.getRowCount() == 0) {
251             JOptionPane.showMessageDialog(this, "Agregue al menos un servicio",
252                                         "Agregue", JOptionPane.ERROR_MESSAGE);
253             return;
254         }
255         Pago.Agregar_Pago(ID_Paciente, ID_Medico);
256         int ID_PMax = 0;
257         try {
258             resultado = Conexion.consulta("Select Max(ID_Pago) from Pago");
259             while (resultado.next()) {
260                 ID_PMax = resultado.getInt(1);
261             }
262         } catch (SQLException ex) {
263         }
264         for (int i = 0; i < jTable1.getRowCount(); i++) {
265
266             int ID_Servicio = Integer.parseInt(model.getValueAt(i, 0).toString());
267             Pago.Agregar_DetallePago(ID_PMax, ID_Servicio, Cantidad);
268         }
269         Limpiar();
270     }

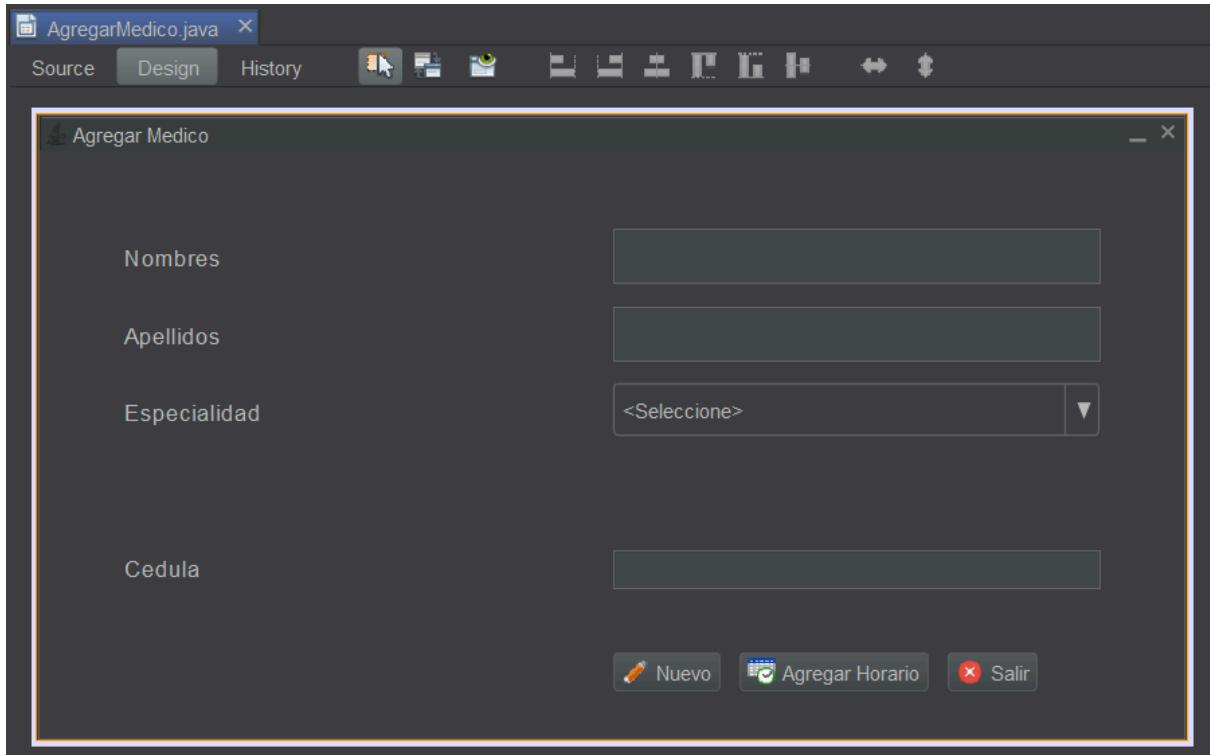
```

En esta vista se puede observar como son las facturas que ya se han cobrado con los pacientes ya atendidos



```
VerDetallePago.java
Source Design History
22 @SuppressWarnings("unchecked")
23 Generated Code
70
71 ResultSet resultado;
72
73 public void CargarDatos(int ID_Pago) {
74
75     String[] Header = {"Codigo", "No.", "Servicio", "Precio", "Cantidad", "Total", "Estado"};
76     model.setColumnIdentifiers(Header);
77     this.jTable1.setModel(model);
78     String[] Datos = new String[7];
79
80     try {
81         resultado = Conexion.consulta("Select * from Detalle_PagoV "
82             + "Where ID_Pago = " + ID_Pago);
83         while (resultado.next()) {
84             Datos[0] = String.valueOf(resultado.getInt(1));
85             Datos[1] = String.valueOf(resultado.getInt(2));
86             Datos[2] = resultado.getString(3);
87             Datos[3] = String.valueOf(resultado.getDouble(4));
88             Datos[4] = String.valueOf(resultado.getInt(5));
89             Datos[5] = String.valueOf(resultado.getDouble(6));
90             boolean Estado = resultado.getBoolean(7);
91             String Estate = "Cancelado";
92             if (Estado) {
93                 Estate = "Activo";
94             }
95             Datos[6] = Estate;
96
97             model.addRow(Datos);
98         }
99     } catch (SQLException ex) {
100 }
101     jTable1.setModel(model);
102 }
```

19. En este panel se agrega al médico asignando su nombre , apellidos , especialidad y su cédula



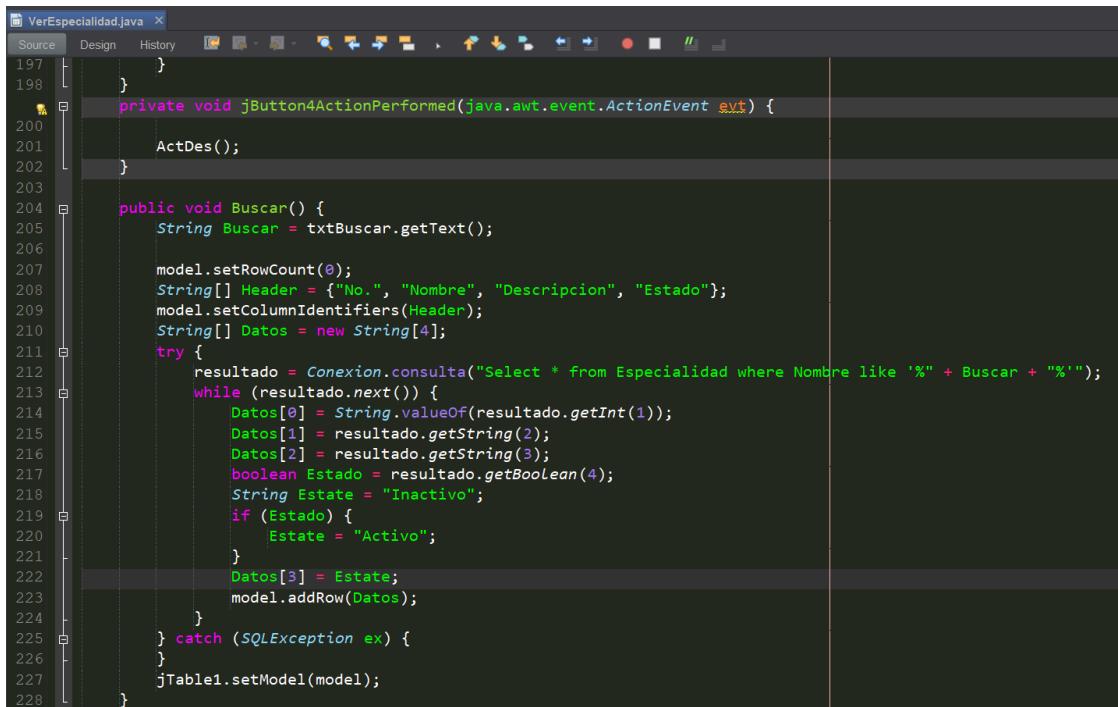
```
10  *
11  * @yeudi
12  */
13  public class AgregarMedico extends javax.swing.JInternalFrame {
14
15     public AgregarMedico() {
16         initComponents();
17     }
18
19     @SuppressWarnings("unchecked")
20     Generated Code
21
22     public void Guardar(){
23         String Nombre = txtNombre.getText().trim();
24         String Apellido = txtApellido.getText().trim();
25         String Cedula = txtCedula.getText().trim();
26         int cmbEsp = cmbEspecialidad.getSelectedIndex();
27         int ID_Especialidad = ID_Esp[cmbEsp];
28
29         if("".equals(Nombre) || "".equals(Apellido) || "".equals(Cedula) || cmbEsp == 0){
30             JOptionPane.showMessageDialog(this, "Complete todos los campos y seleccione correctamente",
31                     "Complete", JOptionPane.ERROR_MESSAGE);
32         } else if (Cedula.length() != 8) {
33             JOptionPane.showMessageDialog(this, "La cédula debe tener exactamente 8 dígitos.",
34                     "Error", JOptionPane.ERROR_MESSAGE);
35         } else {
36             Horario H = new Horario(null, true);
37             H.setAM(this);
38             H.setNombre(Nombre);
39             H.setApellido(Apellido);
40             H.setCedula(Cedula);
41             H.setID_Especialidad(ID_Especialidad);
42             H.setVisible(true);
43         }
44     }
45 }
```

The code is a Java class named 'AgregarMedico' that extends 'JInternalFrame'. It contains a constructor that calls 'initComponents()'. The 'Guardar()' method is annotated with '@SuppressWarnings("unchecked")'. It retrieves values from text fields 'txtNombre', 'txtApellido', and 'txtCedula', and an integer from a dropdown 'cmbEspecialidad'. It uses 'JOptionPane' to validate input: if any field is empty or the dropdown is at index 0, or if the cedula length is not 8, it shows an error message. Otherwise, it creates a new 'Horario' object, sets its properties (using 'this' for AM), and makes it visible.

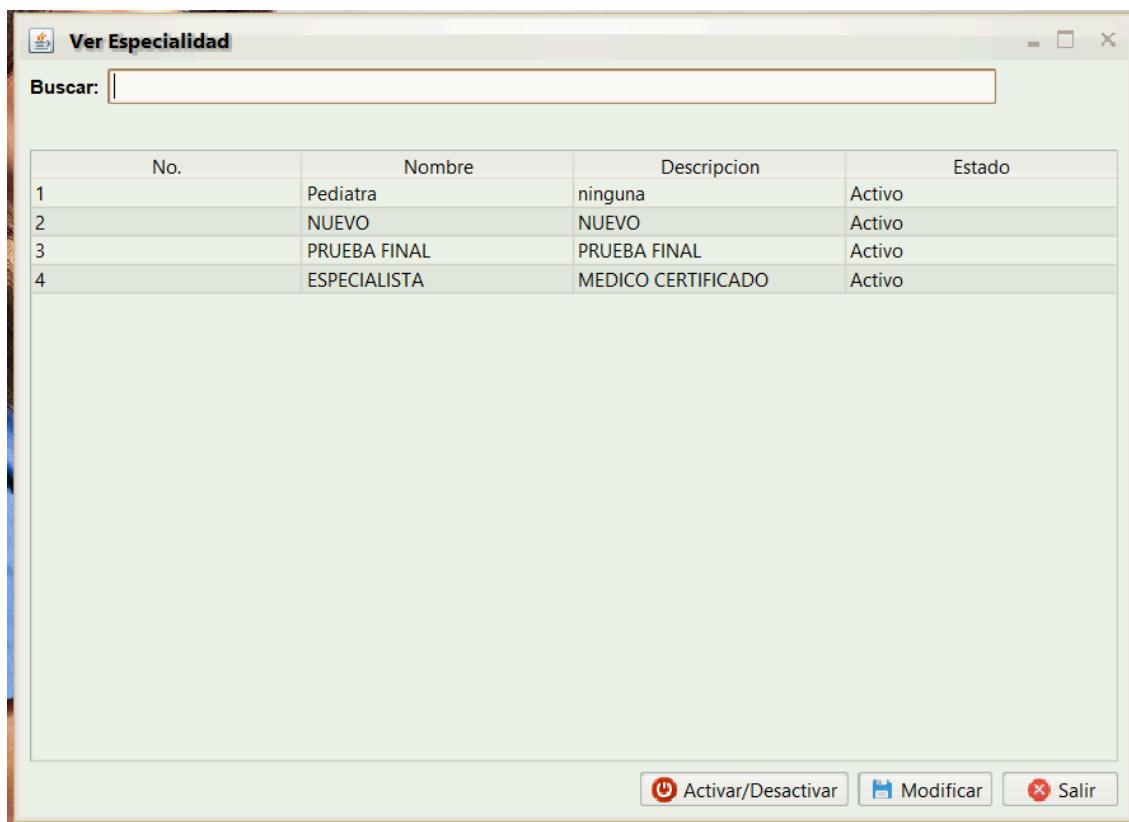
Con esta validación se puede editar los registros del médico

```
ModificarMedico.java
Source Design History
157     public void CargarDatos(int ID){
158         CargarEspecialidad();
159         String Nombre="";
160         String Apellido = "";
161         String Especialidad="";
162         try {
163             resultado = Conexion.consulta("Select ID_Medico, Nombres, Apellidos, Cedula, Nombre "
164                                         + "from MedicoV Where ID_Medico = " + ID);
165             while (resultado.next()) {
166                 IDD = resultado.getInt(1);
167                 Nombre = resultado.getString(2);
168                 Apellido = resultado.getString(3);
169                 Cedula = resultado.getString(4); // Cargar cédula desde la base de datos
170                 Especialidad = resultado.getString(5);
171             }
172         }catch(SQLException ex){}
173         txtNombre.setText(Nombre);
174         txtApellido.setText(Apellido);
175         txtCedula.setText(Cedula); // Mostrar la cédula en el campo correspondiente
176         cmbEspecialidad.setSelectedItem(Especialidad);
177     }
178     public void Guardar(){
179         String Nombre = txtNombre.getText().trim();
180         String Apellido = txtApellido.getText().trim();
181         String Cedula = txtCedula.getText().trim();
182         int cmbEsp = cmbEspecialidad.getSelectedIndex();
183         int ID_Especialidad = ID_Esp[cmbEsp];
184
185         // Validar campos
186         if ("".equals(Nombre) || "".equals(Apellido) || "".equals(Cedula) || cmbEsp == 0) {
187             JOptionPane.showMessageDialog(this, "Complete todos los campos correctamente",
188                                         "Error", JOptionPane.ERROR_MESSAGE);
189         } else {
190             // Actualizar el médico en la base de datos
191             Medico.Actualizar_Medico(IDD, Nombre, Apellido, Cedula, ID_Especialidad);
192             Limpiar();
193         }
194     }
195     public void ModificarHorario(){
196         ModificarHorario MH = new ModificarHorario(null, true);
197         MH.setID(IDD);
198         MH.CargarDatos(IDD);
199         MH.setVisible(true);
200     }
}
```

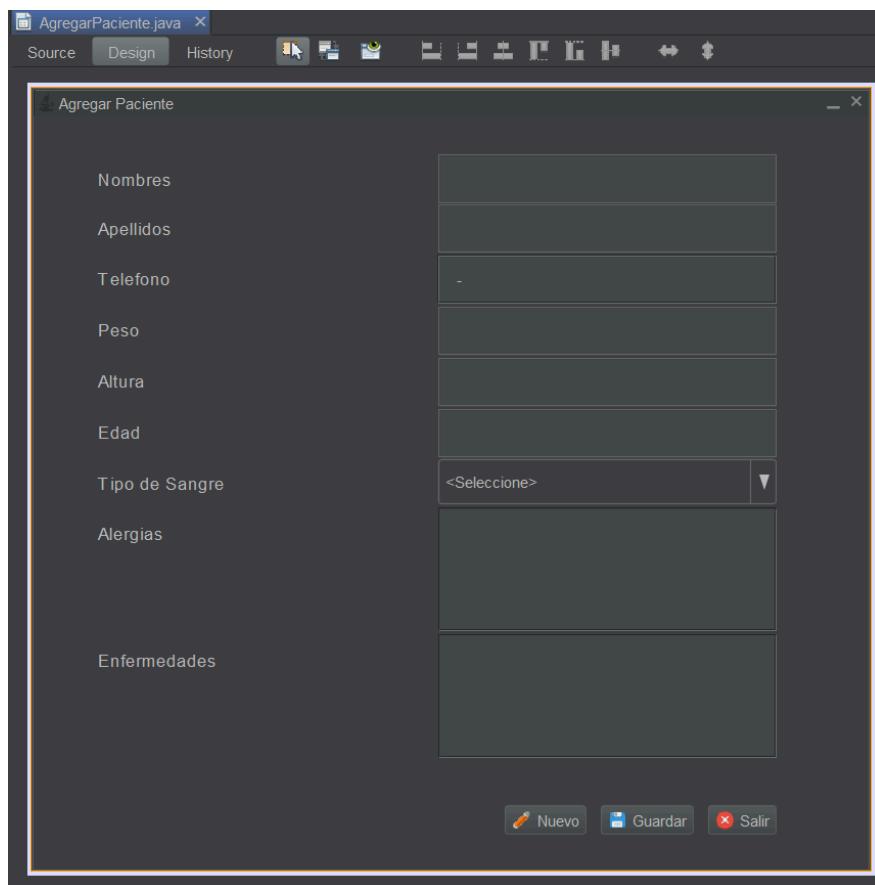
En este apartado se puede visualizar el apartado de ver especialidad con los datos extraídos desde la base de datos



```
VerEspecialidad.java
Source Design History
197     }
198 }
199 private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
200     ActDes();
201 }
202
203 public void Buscar() {
204     String Buscar = txtBuscar.getText();
205
206     model.setRowCount(0);
207     String[] Header = {"No.", "Nombre", "Descripcion", "Estado"};
208     model.setColumnIdentifiers(Header);
209     String[] Datos = new String[4];
210     try {
211         resultado = Conexion.consulta("Select * from Especialidad where Nombre like '%" + Buscar + "%'");
212         while (resultado.next()) {
213             Datos[0] = String.valueOf(resultado.getInt(1));
214             Datos[1] = resultado.getString(2);
215             Datos[2] = resultado.getString(3);
216             boolean Estado = resultado.getBoolean(4);
217             String Estate = "Inactivo";
218             if (Estado) {
219                 Estate = "Activo";
220             }
221             Datos[3] = Estate;
222             model.addRow(Datos);
223         }
224     } catch (SQLException ex) {
225     }
226     jTable1.setModel(model);
227 }
228 }
```



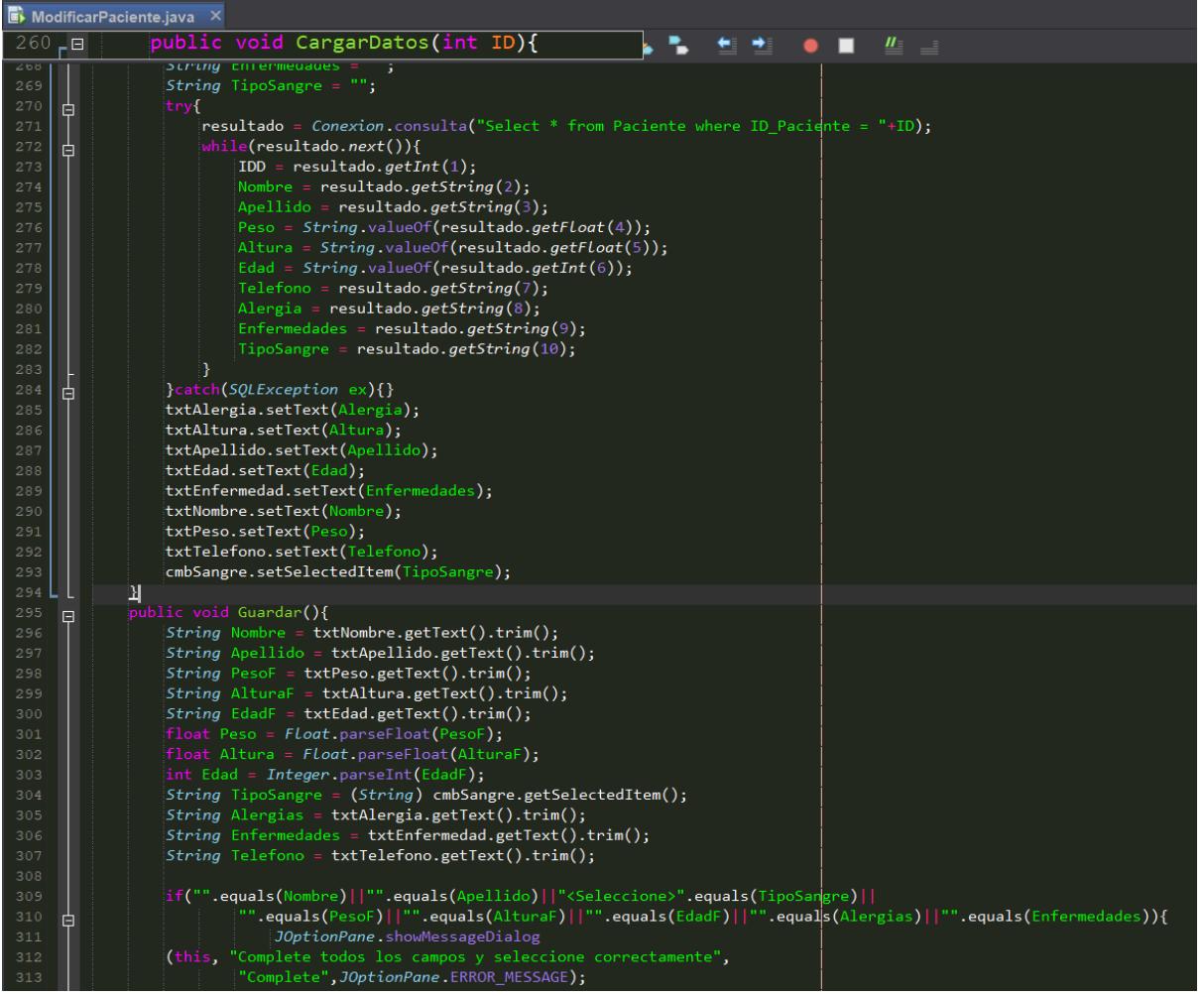
20. En este panel se agrega al paciente con los datos sugeridos nombres , apellidos , teléfono , peso , altura , edad , tipo de sangre , alergias y enfermedades



```
19  public AgregarPaciente() {
20      initComponents();
21  }
22
23  @SuppressWarnings("unchecked")
24  Generated Code
296
297  public void Guardar() {
298      String Nombre = txtNombre.getText().trim();
299      String Apellido = txtApellido.getText().trim();
300      String PesoF = txtPeso.getText().trim();
301      String AlturaF = txtAltura.getText().trim();
302      String EdadF = txtEdad.getText().trim();
303      float Peso = Float.parseFloat(PesoF);
304      float Altura = Float.parseFloat(AlturaF);
305      int Edad = Integer.parseInt(EdadF);
306      String TipoSangre = (String) cmbSangre.getSelectedItem();
307      String Alergias = txtAlergia.getText().trim();
308      String Enfermedades = txtEnfermedad.getText().trim();
309      String Telefono = txtTelefono.getText().trim();
310
311      if ("".equals(Nombre) || "".equals(Apellido) || "<Seleccione>".equals(TipoSangre)
312          || "".equals(PesoF) || "".equals(AlturaF) || "".equals(EdadF) || "".equals(Alergias) ||
313          "".equals(Enfermedades)) {
314          JOptionPane.showMessageDialog(this, "Complete todos los campos y seleccione correctamente",
315              "Complete", JOptionPane.ERROR_MESSAGE);
316      } else {
317          Paciente.Agregar_Paciente(Nombre, Apellido, Peso, Altura, Edad, Alergias, Enfermedades, TipoSangre,
318          Telefono);
319          Limpiar();
320      }
321  }
322
323 }
```

The code editor shows the implementation of the "Guardar" method. It first retrieves values from various text fields and dropdowns. It then checks if any of the required fields are empty or if the blood type selection is empty. If any condition is true, it displays an error message dialog. Otherwise, it calls the `Paciente.Agregar_Paciente` method with the provided data and then clears the form.

En este apartado del código se puede modificar los datos ya registrados en la base de datos actualizandolos

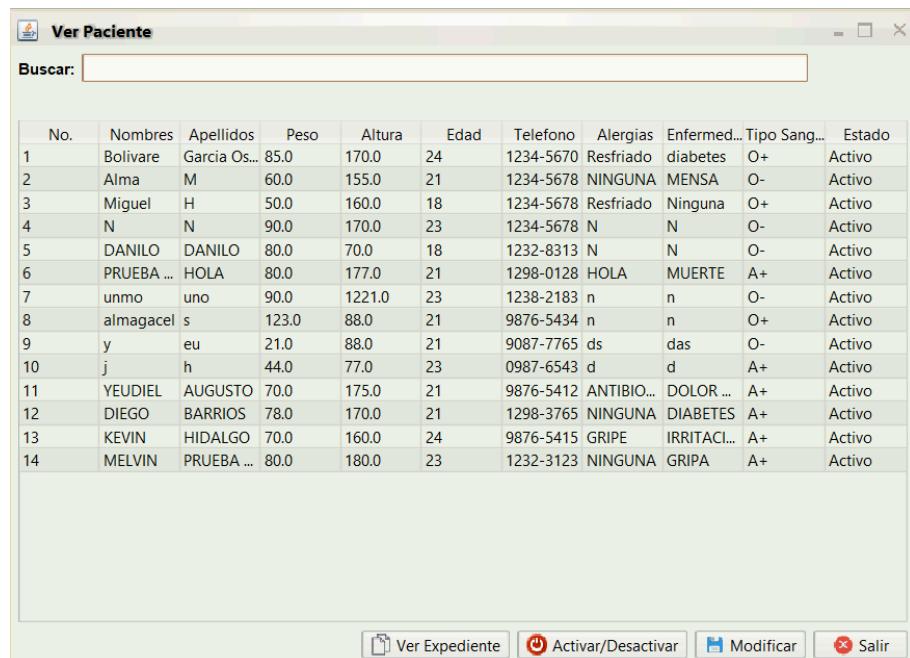


```

260     public void CargarDatos(int ID){
261         String Enfermedades = "";
262         String TipoSangre = "";
263         try{
264             resultado = Conexion.consulta("Select * from Paciente where ID_Paciente = "+ID);
265             while(resultado.next()){
266                 IDD = resultado.getInt(1);
267                 Nombre = resultado.getString(2);
268                 Apellido = resultado.getString(3);
269                 Peso = String.valueOf(resultado.getFloat(4));
270                 Altura = String.valueOf(resultado.getFloat(5));
271                 Edad = String.valueOf(resultado.getInt(6));
272                 Telefono = resultado.getString(7);
273                 Alergia = resultado.getString(8);
274                 Enfermedades = resultado.getString(9);
275                 TipoSangre = resultado.getString(10);
276             }
277         }catch(SQLException ex){}
278         txtAlergia.setText(Alergia);
279         txtAltura.setText(Altura);
280         txtApellido.setText(Apellido);
281         txtEdad.setText(Edad);
282         txtEnfermedad.setText(Enfermedades);
283         txtNombre.setText(Nombre);
284         txtPeso.setText(Peso);
285         txtTelefono.setText(Telefono);
286         cmbSangre.setSelectedItem(TipoSangre);
287     }
288
289     public void Guardar(){
290         String Nombre = txtNombre.getText().trim();
291         String Apellido = txtApellido.getText().trim();
292         String PesoF = txtPeso.getText().trim();
293         String AlturaF = txtAltura.getText().trim();
294         String EdadF = txtEdad.getText().trim();
295         float Peso = Float.parseFloat(PesoF);
296         float Altura = Float.parseFloat(AlturaF);
297         int Edad = Integer.parseInt(EdadF);
298         String TipoSangre = (String) cmbSangre.getSelectedItem();
299         String Alergias = txtAlergia.getText().trim();
300         String Enfermedades = txtEnfermedad.getText().trim();
301         String Telefono = txtTelefono.getText().trim();
302
303         if("".equals(Nombre)||"".equals(Apellido)||"<selecione>".equals(TipoSangre)||"
304             "".equals(PesoF)||"".equals(AlturaF)||"".equals(EdadF)||"".equals(Alergias)||"".equals(Enfermedades)){
305             JOptionPane.showMessageDialog(
306                 this, "Complete todos los campos y seleccione correctamente",
307                 "Complete", JOptionPane.ERROR_MESSAGE);
308         }
309     }
310 }
311
312
313

```

En este apartado se muestra la interfaz de los datos ya integrados desde la base de datos



No.	Nombres	Apellidos	Peso	Altura	Edad	Telefono	Alergias	Enfermed...	Tipo Sang...	Estado
1	Bolivare	Garcia Os...	85.0	170.0	24	1234-5670	Resfriado	diabetes	O+	Activo
2	Alma	M	60.0	155.0	21	1234-5678	NINGUNA	MENSA	O-	Activo
3	Miguel	H	50.0	160.0	18	1234-5678	Resfriado	Ninguna	O+	Activo
4	N	N	90.0	170.0	23	1234-5678	N	N	O-	Activo
5	DANILO	DANILO	80.0	70.0	18	1232-8313	N	N	O-	Activo
6	PRUEBA ...	HOLA	80.0	177.0	21	1298-0128	HOLA	MUERTE	A+	Activo
7	unmo	uno	90.0	1221.0	23	1238-2183	n	n	O-	Activo
8	almagacel	s	123.0	88.0	21	9876-5434	n	n	O+	Activo
9	y	eu	21.0	88.0	21	9087-7765	ds	das	O-	Activo
10	j	h	44.0	77.0	23	0987-6543	d	d	A+	Activo
11	YEUDIEL	AUGUSTO	70.0	175.0	21	9876-5412	ANTIBIO...	DOLOR ...	A+	Activo
12	DIEGO	BARRIOS	78.0	170.0	21	1298-3765	NINGUNA	DIABETES	A+	Activo
13	KEVIN	HIDALGO	70.0	160.0	24	9876-5415	GRIFE	IRRITACI...	A+	Activo
14	MELVIN	PRUEBA ...	80.0	180.0	23	1232-3123	NINGUNA	GRIPA	A+	Activo

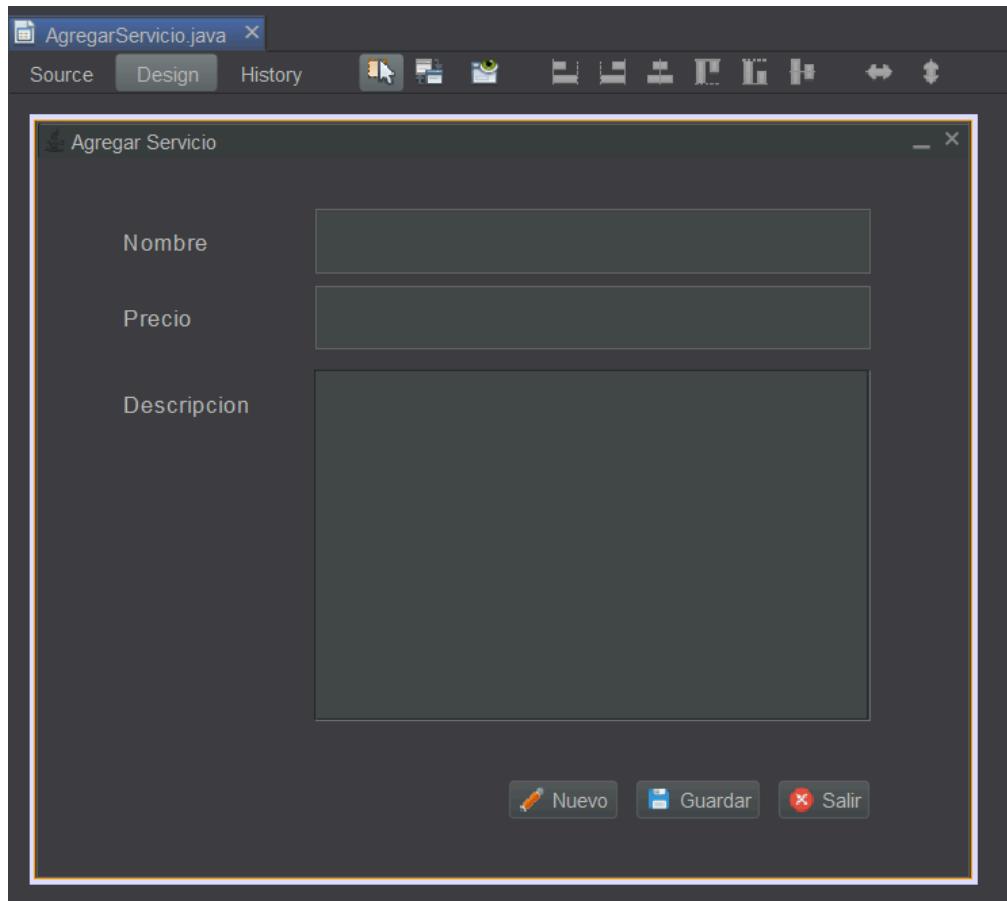
The screenshot shows a Java IDE interface with the file `VerPaciente.java` open. The code is a Java class with methods for handling button actions and loading patient data from a database.

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();
}

ResultSet resultado;

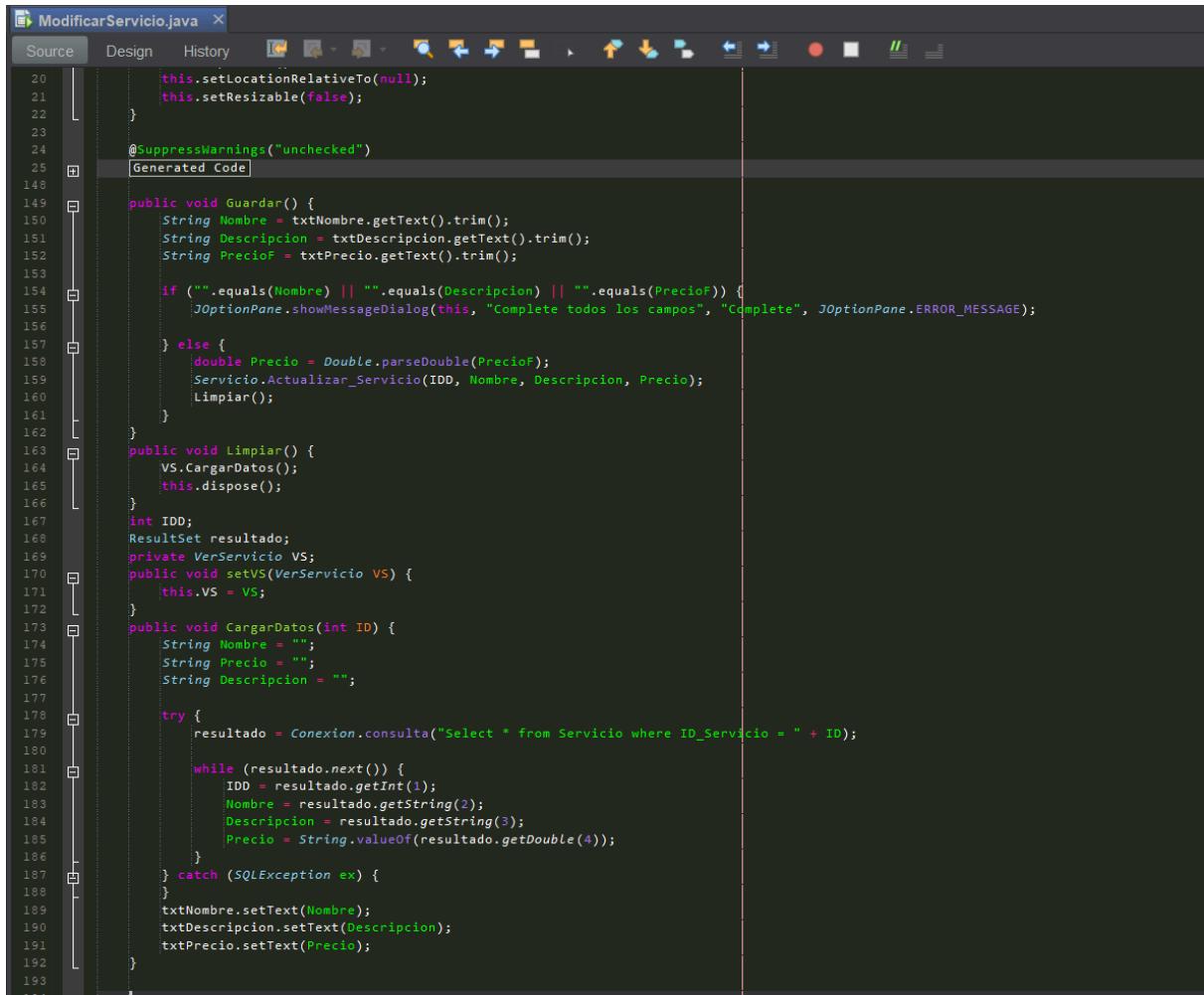
public void CargarDatos() {
    model.setRowCount(0);
    String[] Header = {"No.", "Nombres", "Apellidos", "Peso", "Altura",
        "Edad", "Telefono", "Alergias", "Enfermedades", "Tipo Sangre", "Estado"};
    model.setColumnIdentifiers(Header);
    String[] Datos = new String[11];
    try {
        resultado = Conexion.consulta("Select * from Paciente");
        while (resultado.next()) {
            Datos[0] = String.valueOf(resultado.getInt(1));
            Datos[1] = resultado.getString(2);
            Datos[2] = resultado.getString(3);
            Datos[3] = String.valueOf(resultado.getFloat(4));
            Datos[4] = String.valueOf(resultado.getFloat(5));
            Datos[5] = String.valueOf(resultado.getInt(6));
            Datos[6] = resultado.getString(7);
            Datos[7] = resultado.getString(8);
            Datos[8] = resultado.getString(9);
            Datos[9] = resultado.getString(10);
            boolean Estado = resultado.getBoolean(11);
            String Estate = "Inactivo";
            if (Estado) {
                Estate = "Activo";
            }
            Datos[10] = Estate;
            model.addRow(Datos);
        }
    } catch (SQLException ex) {
    }
    jTable1.setModel(model);
}
```

21. En este apartado se registra el tipo de servicio que se le cobrará al paciente se añade su nombre , precio y descripción



```
7
8  /**
9  * 
10 * @yeudi
11 */
12 public class AgregarServicio extends javax.swing.JInternalFrame {
13 
14     public AgregarServicio() {
15         initComponents();
16     }
17 
18     @SuppressWarnings("unchecked")
19     // Generated Code
20 
21     public void Guardar() {
22         String Nombre = txtNombre.getText().trim();
23         String Descripcion = txtDescripcion.getText().trim();
24         String PrecioF = txtPrecio.getText().trim();
25 
26         if ("").equals(Nombre) || "".equals(Descripcion) || "".equals(PrecioF)) {
27             JOptionPane.showMessageDialog(this, "Complete todos los campos", "Complete", JOptionPane.ERROR_MESSAGE);
28         } else {
29             double Precio = Double.parseDouble(PrecioF);
30             Servicio.Agregar_Servicio(Nombre, Descripcion, Precio);
31             Limpiar();
32         }
33     }
34 
35     public void Limpiar() {
36         txtDescripcion.setText("");
37         txtNombre.setText("");
38         txtPrecio.setText("");
39     }
40 }
```

En este código se puede modificar el servicio ya registrado en la base de datos



```
20     this.setLocationRelativeTo(null);
21     this.setResizable(false);
22 }
23
24 @SuppressWarnings("unchecked")
25 Generated Code
26
27
28 public void Guardar() {
29     String Nombre = txtNombre.getText().trim();
30     String Descripcion = txtDescripcion.getText().trim();
31     String PrecioF = txtPrecio.getText().trim();
32
33     if (" ".equals(Nombre) || " ".equals(Descripcion) || " ".equals(PrecioF)) {
34         JOptionPane.showMessageDialog(this, "Complete todos los campos", "Complete", JOptionPane.ERROR_MESSAGE);
35     } else {
36         double Precio = Double.parseDouble(PrecioF);
37         Servicio.Actualizar_Servicio(IDD, Nombre, Descripcion, Precio);
38         Limpiar();
39     }
40 }
41
42 public void Limpiar() {
43     VS.CargarDatos();
44     this.dispose();
45 }
46
47 int IDD;
48 ResultSet resultado;
49 private VerServicio VS;
50 public void setVS(VerServicio VS) {
51     this.VS = VS;
52 }
53
54 public void CargarDatos(int ID) {
55     String Nombre = "";
56     String Precio = "";
57     String Descripcion = "";
58
59     try {
60         resultado = Conexion.consulta("select * from Servicio where ID_Servicio = " + ID);
61
62         while (resultado.next()) {
63             IDD = resultado.getInt(1);
64             Nombre = resultado.getString(2);
65             Descripcion = resultado.getString(3);
66             Precio = String.valueOf(resultado.getDouble(4));
67         }
68     } catch (SQLException ex) {
69     }
70     txtNombre.setText(Nombre);
71     txtDescripcion.setText(Descripcion);
72     txtPrecio.setText(Precio);
73 }
```

En este apartado se muestra la interfaz de los servicios ya registrados



```
VerServicio.java x
Source Design History
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();
}

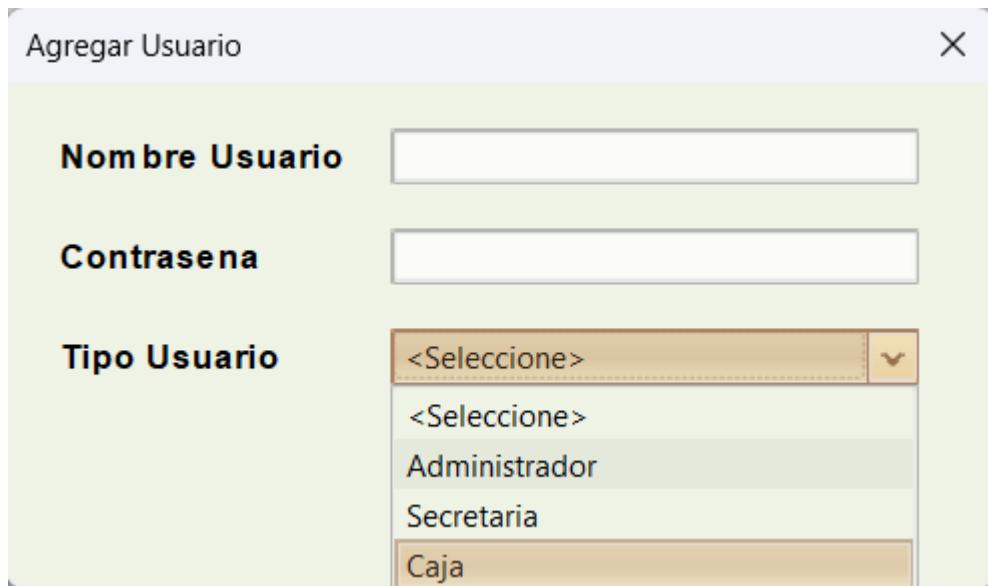
ResultSet resultado;

public void CargarDatos() {
    model.setRowCount(0);

    String[] Header = {"No.", "Nombre", "Descripcion", "Precio", "Estado"};
    model.setColumnIdentifiers(Header);
    String[] Datos = new String[5];
    try {

        resultado = Conexion.consulta("Select * from Servicio");
        while (resultado.next()) {
            Datos[0] = String.valueOf(resultado.getInt(1));
            Datos[1] = resultado.getString(2);
            Datos[2] = resultado.getString(3);
            Datos[3] = String.valueOf(resultado.getDouble(4));
            boolean Estado = resultado.getBoolean(5);
            String Estate = "Inactivo";
            if (Estado) {
                Estate = "Activo";
            }
            Datos[4] = Estate;
            model.addRow(Datos);
        }
    } catch (SQLException ex) {
    }
    jTable1.setModel(model);
}
```

22. En este apartado se registran los nuevos usuarios con sus nombres y password y su respectivo rol



```
 9  * @yeudi
10 */
11 public class AgregarUsuario extends javax.swing.JDialog {
12
13     public AgregarUsuario(java.awt.Frame parent, boolean modal) {
14         super(parent, modal);
15         initComponents();
16         this.setResizable(false);
17     }
18
19     @SuppressWarnings("unchecked")
20     Generated Code
21
22     private String NombreUsuario;
23     public void setNombreUsuario(String NombreUsuario) {
24         this.NombreUsuario = NombreUsuario;
25         this.txtUser.setText(NombreUsuario);
26     }
27     public void Guardar(){
28         String Nombre = txtUser.getText().trim();
29         String Pass = txtPass.getText();
30
31         int Rol = cmbTipo.getSelectedIndex();
32
33         if("".equals(Nombre)||"".equals(Pass) || Rol == 0){
34             JOptionPane.showMessageDialog(this, "Ingrese y seleccione todos los campos",
35                 "Ingrese y seleccione", JOptionPane.ERROR_MESSAGE);
36             return;
37         }
38         String Role = (String) cmbTipo.getSelectedItem();
39         if(Rol == 1){
40             Role = "Admin";
41         }
42         Usuario.Agregar_Usuario(Nombre, Pass, Role);
43         this.dispose();
44     }
45 }
```

The screenshot shows the Java code editor with the file "AgregarUsuario.java" open. The code defines a class "AgregarUsuario" that extends "JDialog". It includes methods for setting the user name and password, and a "Guardar" method that performs validation (ensuring both fields are filled and a role is selected). If validation fails, it displays an error message. If successful, it calls a static method "Agregar_Usuario" from the "Usuario" class, passing the user name, password, and role, and then closes the dialog.

Se valida la actualización de la modificación de usuarios

```
101 public void CargarDatos(int ID) {
102     String User = "";
103     String Pass = "";
104     String Tipo = "";
105     try {
106         resultado = Conexion.consulta("Select * from Usuario where ID_Usuario = " + ID);
107         while (resultado.next()) {
108             IDD = resultado.getInt(1);
109             User = resultado.getString(2);
110             Pass = resultado.getString(3);
111             Tipo = resultado.getString(4);
112         }
113     } catch (SQLException ex) {
114     }
115     txtUser.setText(User);
116     txtPass.setText(Pass);
117
118     if("Medico".equalsIgnoreCase(Tipo)){
119         cmbTipo.addItem("Medico");
120         cmbTipo.setEnabled(false);
121     }
122     cmbTipo.setSelectedItem(Tipo);
123     if("Admin".equalsIgnoreCase(Tipo)){
124         cmbTipo.setSelectedItem("Administrador");
125     }
126 }
127
128 public void Guardar() {
129     String Nombre = txtUser.getText().trim();
130     String Pass = txtPass.getText();
131
132     int Rol = cmbTipo.getSelectedIndex();
133     if ("".equals(Nombre) || "".equals(Pass) || Rol == 0) {
134         JOptionPane.showMessageDialog(this, "Ingrese y seleccione todos los campos",
135             "Ingresar datos", JOptionPane.ERROR_MESSAGE);
136         return;
137     }
138     String Role = (String) cmbTipo.getSelectedItem();
139     if (Rol == 1) {
140         Role = "Admin";
141     }
142     Usuario.Actualizar_Usuario(IDD, Nombre, Pass, Role);
143     Limpiar();
144 }
145
146 int IDD;
147 ResultSet resultado;
148 private VerUsuario VU;
149 public void setVU(VerUsuario VU) {
150     this.VU = VU;
151 }
152 public void Limpiar() {
153     VU.CargarDatos();
154     this.dispose();
155 }
```

Se pueden visualizar el registro de los usuarios con su password y roles ya asignados

No.	Nombre	Contrasena	Rol	Estado
1	Admin	1234	Admin	Activo
2	yeudiel2	123	Medico	Activo
3	jenni	123	Secretaria	Activo
4	bolivare3	123	Medico	Activo
5	jacob4	123	Medico	Activo
6	treintadenov5	treintadenov5	Medico	Activo
7	yeu6	1234	Medico	Activo
8	emmanuel7	123	Medico	Activo
9	gersonv8	123	Medico	Activo
10	prueba9	123	Medico	Activo

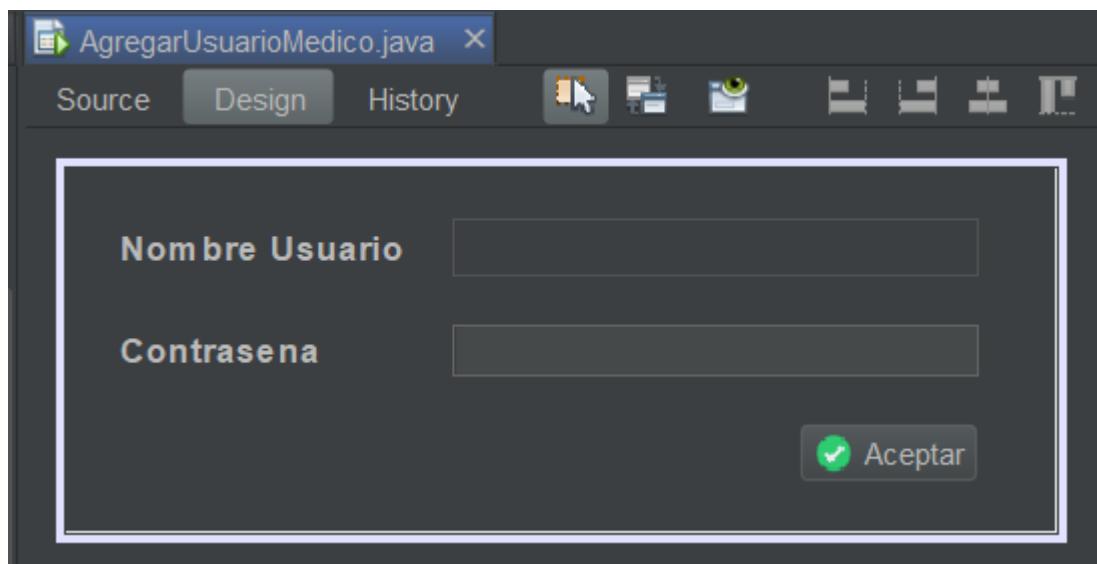
Buttons at the bottom:

- Activar/Desactivar (Activate/Deactivate)
- Modificar (Modify)
- Salir (Exit)

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    this.dispose();
}

ResultSet resultado;
public void CargarDatos(){
    model.setRowCount(0);
    String [] Header = {"No.", "Nombre", "Contrasena", "Rol", "Estado"};
    model.setColumnIdentifiers(Header);

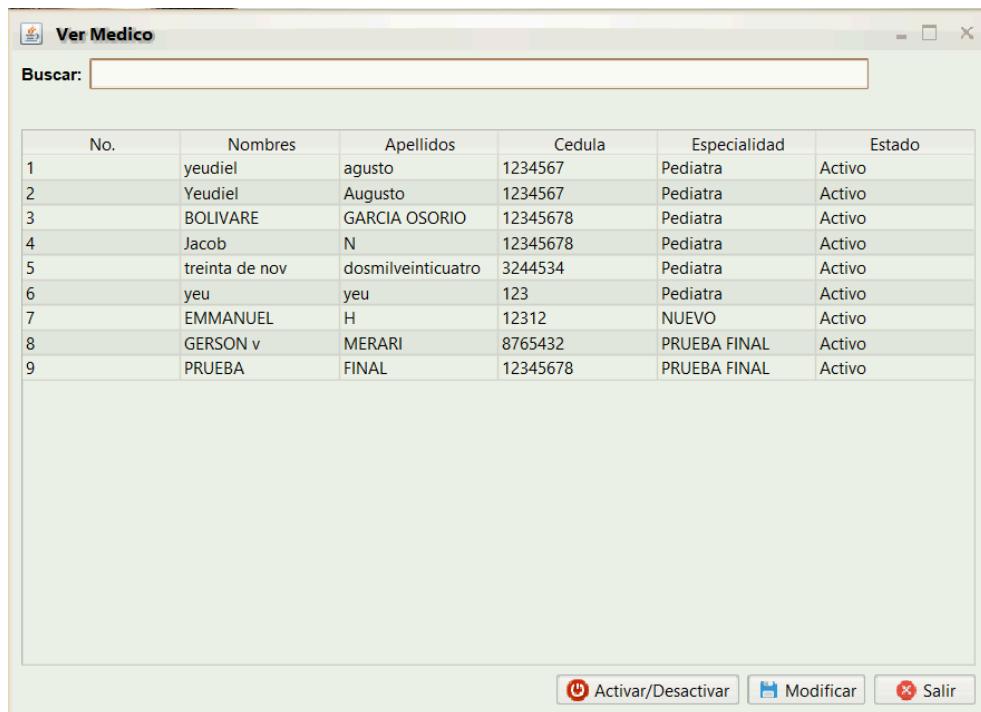
    String [] Datos = new String[5];
    try{
        resultado = Conexion.consulta("Select * from Usuario");
        while(resultado.next()){
            Datos [0] = String.valueOf(resultado.getInt(1));
            Datos [1] = resultado.getString(2);
            Datos [2] = resultado.getString(3);
            Datos [3] = resultado.getString(4);
            boolean Estado = resultado.getBoolean(5);
            String Estate = "Inactivo";
            if(Estado){
                Estate = "Activo";
            }
            Datos [4] = Estate;
            model.addRow(Datos);
        }
    }catch(SQLException ex){
    }
    jTable1.setModel(model);
}
```



```
16 /**
17 *
18 * @yeudi
19 */
20 public class AgregarUsuarioMedico extends javax.swing.JDialog {
21
22     public AgregarUsuarioMedico(java.awt.Frame parent, boolean modal) {
23         super(parent, modal);
24         initComponents();
25         this.setResizable(false);
26     }
27
28     @SuppressWarnings("unchecked")
29     private String NombreUsuario;
30
31     public void setNombreUsuario(String NombreUsuario) {
32         this.NombreUsuario = NombreUsuario;
33         this.txtUser.setText(NombreUsuario);
34     }
35     public void Seleccionar(){
36         String Nombre = txtUser.getText().trim();
37         String Pass = txtPass.getText();
38
39         if("".equals(Nombre)||"".equals(Pass)){
40             JOptionPane.showMessageDialog(this, "Ingresese todos los campos", "Ingresese", JOptionPane.ERROR_MESSAGE);
41             return;
42         }
43         Usuario.Agregar_Usuario(Nombre, Pass, "Medico");
44         this.dispose();
45     }
46 }
```

The screenshot shows the Java code for the "AgregarUsuarioMedico" dialog. The code uses annotations like `@Generated Code` and `@SuppressWarnings("unchecked")`. It includes methods for setting the user name and performing a selection operation. The code is written in Java and uses standard Swing components.

Esta es la interfaz en el que se muestran todos los médicos activos ya registrados en la base de datos

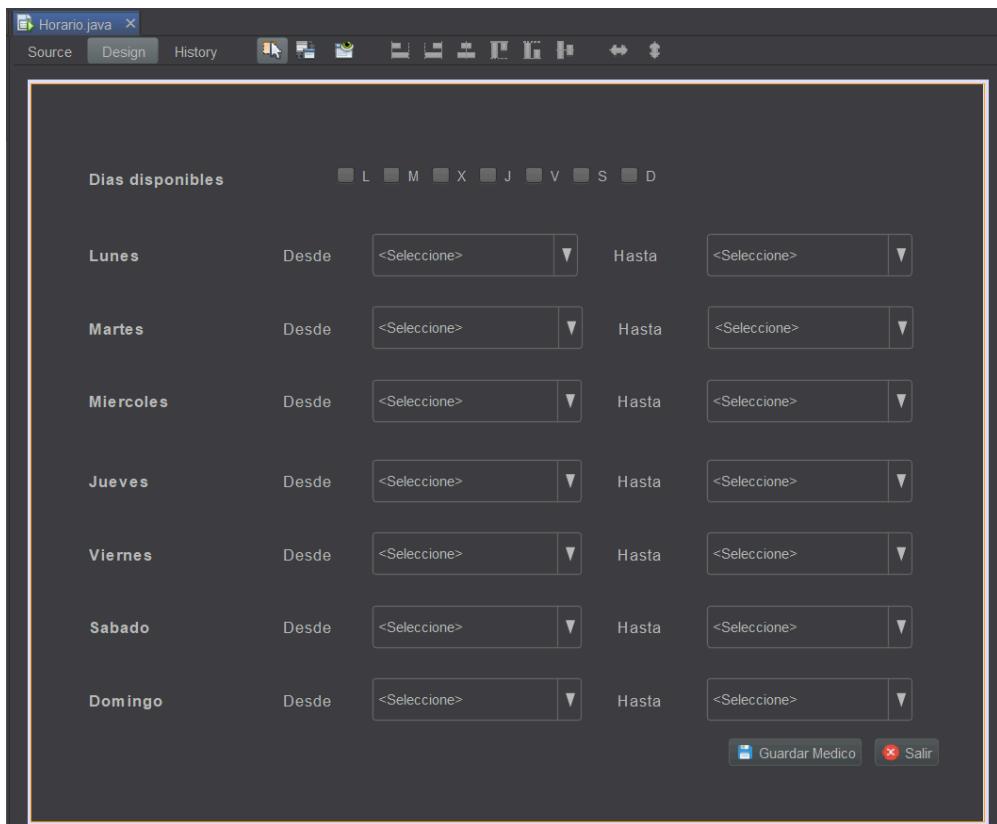


```

 160 }
 161
 162
 163
 164     ResultSet resultado;
 165
 166     public void Modificar() {
 167
 168         int Fila = jTable1.getSelectedRow();
 169         if (Fila >= 0) {
 170             int ID = Integer.parseInt(model.getValueAt(Fila, 0).toString());
 171             ModificarMedico MM = new ModificarMedico(null, true);
 172             MM.CargarDatos(ID);
 173             MM.setVM(this);
 174             MM.setVisible(true);
 175         } else {
 176             JOptionPane.showMessageDialog(this, "Debe seleccionar el registro a modificar",
 177                                         "Seleccione", JOptionPane.ERROR_MESSAGE);
 178         }
 179     }
 180     public void CargarDatos() {
 181         model.setRowCount(0);
 182         String[] Header = {"No.", "Nombres", "Apellidos", "Cedula", "Especialidad", "Estado"};
 183         model.setColumnIdentifiers(Header);
 184         String[] Datos = new String[6];
 185
 186         try {
 187             resultado = Conexion.consulta("Select * from MedicoV");
 188             while (resultado.next()) {
 189                 Datos[0] = String.valueOf(resultado.getInt(1)); // ID Médico
 190                 Datos[1] = resultado.getString(2); // Nombres
 191                 Datos[2] = resultado.getString(3); // Apellidos
 192                 Datos[3] = resultado.getString(4); // Cedula
 193                 Datos[4] = resultado.getString(5); // Especialidad
 194
 195                 boolean Estado = resultado.getBoolean(6); // Estado
 196                 String Estate = Estado ? "Activo" : "Inactivo";
 197                 Datos[5] = Estate;
 198                 model.addRow(Datos);
 199             }
 200         } catch (SQLException ex) {
 201             ex.printStackTrace();
 202         }
 203     }

```

24. En este apartado el administrador debe de hacer registro del horario del médico , de tal manera dándole click al día disponible y su horario.



```
679
680     public void Guardar(){
681         if (cedula == null || Cedula.trim().isEmpty()) {
682             JOptionPane.showMessageDialog(this, "Ingrese una cédula válida", "Error", JOptionPane.ERROR_MESSAGE);
683             return;
684         }
685         if(ckL.isSelected()){
686             int HRI = cmbDesde.getSelectedIndex();
687             int HRS = cmbHasta.getSelectedIndex();
688
689             if(HRI==0 || HRS==0){
690                 JOptionPane.showMessageDialog(this, "Seleccione la hora correctamente", "Selección", JOptionPane.ERROR_MESSAGE);
691                 return;
692             }
693             if(HRI>HRS){
694                 JOptionPane.showMessageDialog(this, "La hora de salida el Lunes no puede ser menor que la de entrada, y la hora de entrada no puede ser mayor que" +
695                     " la de salida",
696                     "Seleccione correctamente", JOptionPane.ERROR_MESSAGE);
697                 return;
698             }
699         }
700     }
701     if(ckM.isSelected()){
702         int HRI2 = cmbDesde2.getSelectedIndex();
703         int HRS2 = cmbHasta2.getSelectedIndex();
704
705         if(HRI2==0 || HRS2==0){
706             JOptionPane.showMessageDialog(this, "Seleccione la hora correctamente", "Selección", JOptionPane.ERROR_MESSAGE);
707             return;
708         }
709         if(HRI2>HRS2){
710             JOptionPane.showMessageDialog(this, "La hora de salida el Martes no puede ser menor que la de entrada, y la hora de entrada no puede ser mayor que" +
711                     " la de salida",
712                     "Seleccione correctamente", JOptionPane.ERROR_MESSAGE);
713             return;
714         }
715     }
716 }
```

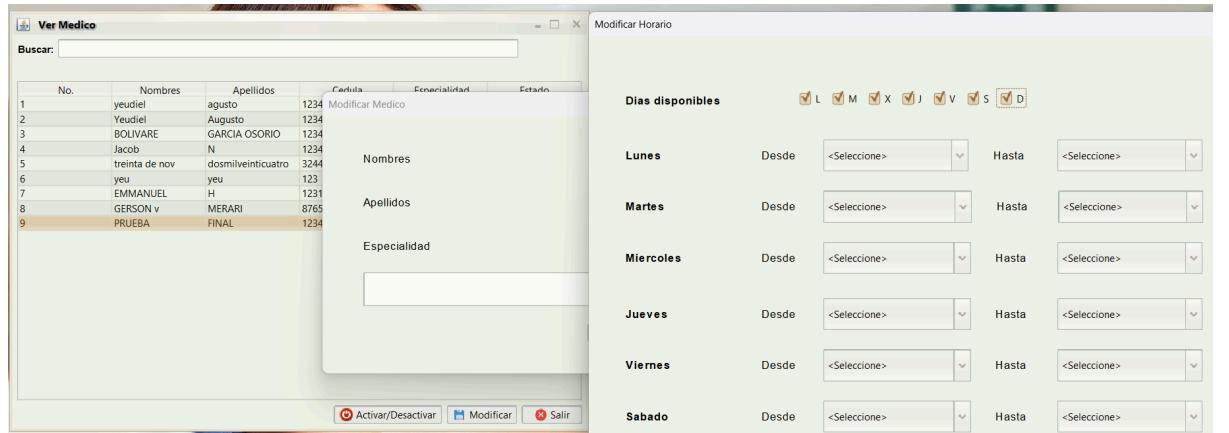
En este apartado se puede verificar el código en donde válida la actualización del horario del médico ya registrado

```

650
651     public void Guardar() {
652         if (ckL.isSelected()) {
653             int HRI = cmbDesde.getSelectedIndex();
654             int HRS = cmbHasta.getSelectedIndex();
655             if(HRI==0 || HRS==0){
656                 JOptionPane.showMessageDialog(this, "Seleccione la hora correctamente","Seleccione",JOptionPane.ERROR_MESSAGE);
657                 return;
658             }
659             if (HRI > HRS) {
660                 JOptionPane.showMessageDialog(this, "la hora de salida el lunes no puede ser menor que la de entrada, y la hora de entrada no p
661                     + " la de salida",
662                     "Seleccione correctamente", JOptionPane.ERROR_MESSAGE);
663                 return;
664             }
665         }
666         if (ckM.isSelected()) {
667             int HRI = cmbDesde2.getSelectedIndex();
668             int HRS = cmbHasta2.getSelectedIndex();
669
670             if(HRI==0 || HRS==0){
671                 JOptionPane.showMessageDialog(this, "Seleccione la hora correctamente","Seleccione",JOptionPane.ERROR_MESSAGE);
672                 return;
673             }
674             if (HRI > HRS) {
675                 JOptionPane.showMessageDialog(this, "La hora de salida el Martes no puede ser menor que la de entrada, y la hora de entrada no p
676                     + " la de salida",
677                     "Seleccione correctamente", JOptionPane.ERROR_MESSAGE);
678                 return;
679             }
680         }
681         if (ckX.isSelected()) {
682             int HRI = cmbDesde3.getSelectedIndex();
683             int HRS = cmbHasta3.getSelectedIndex();
684
685             if(HRI==0 || HRS==0){
686                 JOptionPane.showMessageDialog(this, "Seleccione la hora correctamente","Seleccione",JOptionPane.ERROR_MESSAGE);
687                 return;
688             }
689             if (HRI > HRS) {
690                 JOptionPane.showMessageDialog(this, "La hora de salida el Miércoles no puede ser menor que la de entrada, y la hora de entrada no p
691                     + " la de salida",
692                     "Seleccione correctamente", JOptionPane.ERROR_MESSAGE);
693             }
694         }
695     }

```

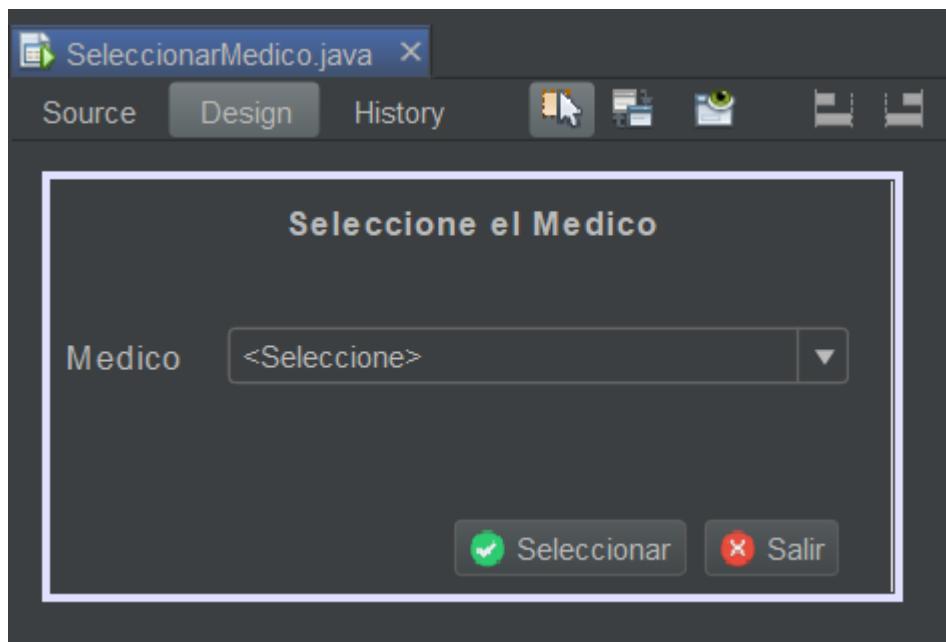
Se puede observar cómo se selecciona desde el médico y abre el apartado de editar el médico y se selecciona el horario y se cambia los días y horas



The screenshot shows a Java code editor window titled "VerHorario.java". The tab bar also includes "Design", "History", and other standard IDE icons. The code itself is a Java method named "CargarDatos" that retrieves data from a database and populates a jTable. The code uses JDBC to execute a query, map the results to days of the week, and add rows to the table model.

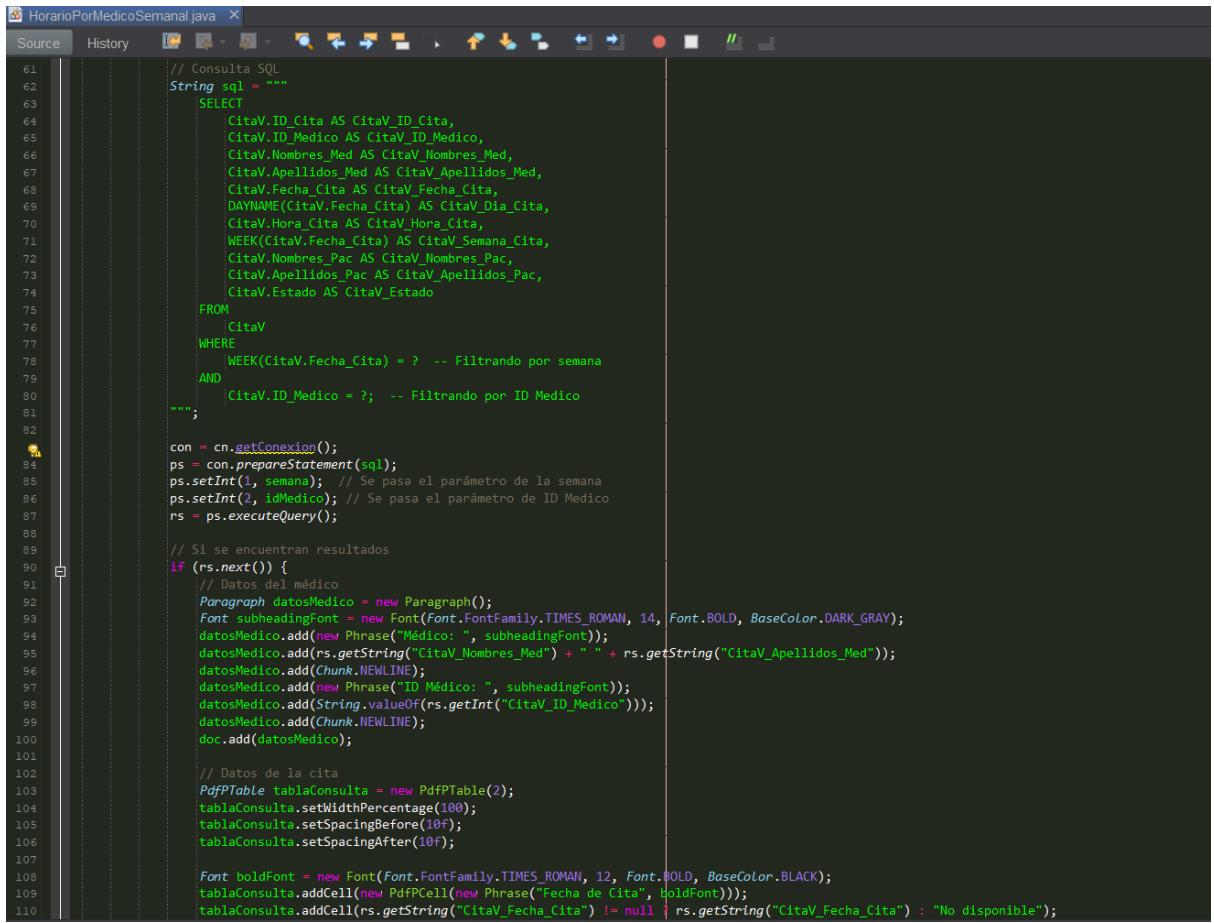
```
12     ResultSet resultado;
13     public void CargarDatos(int ID_Medico) {
14         String[] Header = {"Dia", "Desde", "Hasta"};
15         model.setColumnIdentifiers(Header);
16         this.jTable1.setModel(model);
17         String[] Datos = new String[3];
18         try {
19             resultado = Conexion.consulta("Select Dia, Hora_Inicial, Hora_Final from Horario "
20                 + "Where ID_Medico = " + ID_Medico);
21             while (resultado.next()) {
22                 String Dia = resultado.getString(1);
23                 if ("L".equals(Dia)) {
24                     Dia = "Lunes";
25                 }
26                 if ("M".equals(Dia)) {
27                     Dia = "Martes";
28                 }
29                 if ("X".equals(Dia)) {
30                     Dia = "Miercoles";
31                 }
32                 if ("J".equals(Dia)) {
33                     Dia = "Jueves";
34                 }
35                 if ("V".equals(Dia)) {
36                     Dia = "Viernes";
37                 }
38                 if ("S".equals(Dia)) {
39                     Dia = "Sabado";
40                 }
41                 if ("D".equals(Dia)) {
42                     Dia = "Domingo";
43                 }
44                 Datos[0] = Dia;
45                 Datos[1] = resultado.getString(2);
46                 Datos[2] = resultado.getString(3);
47                 model.addRow(Datos);
48             }
49         } catch (SQLException ex) {
50         }
51     }
52     jTable1.setModel(model);
```

25. Para generar los reportes primero seleccionamos al médico



```
115     }
116
117     public void Seleccionar() {
118         int cmbMed = cmbMedico.getSelectedIndex();
119         if (cmbMed < 0) {
120             JOptionPane.showMessageDialog(this, "Seleccione al Medico", "Seleccione", JOptionPane.ERROR_MESSAGE);
121             return;
122         }
123         int ID_Medico = ID_Med[cmbMed];
124
125         ModoReporte MR = new ModoReporte(null, false);
126         MR.setID_M(ID_Medico);
127         System.out.println("IDM " + ID_Medico);
128         MR.setOpcion(Opcion);
129         MR.setVisible(true);
130         MR.toFront();
131         this.dispose();
132     }
}
```

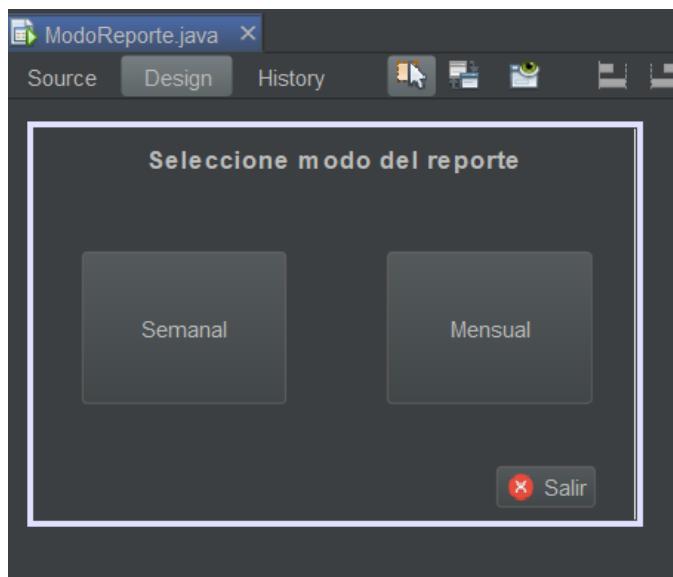
Para verificar el médico semanal se ocupó esta consulta para generar el pdf correcto con la información validada



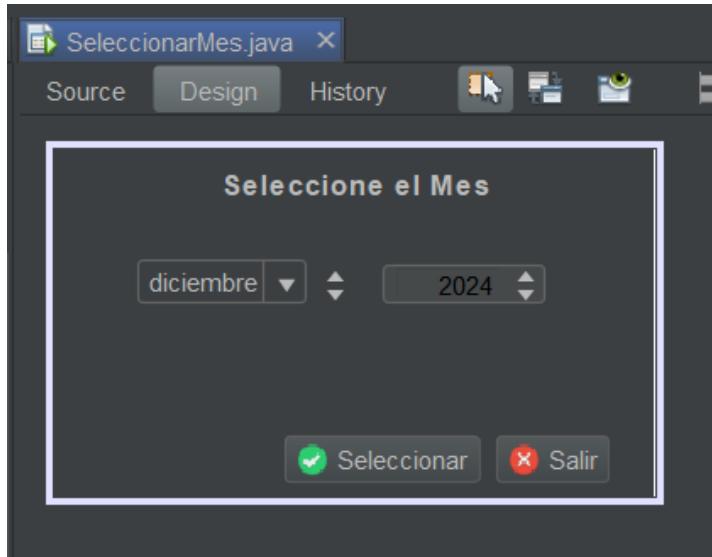
```
1 // Consulta SQL
2 String sql = """
3     SELECT
4         |CitaV.ID_Cita AS CitaV_ID_Cita,
5         |CitaV.ID_Medico AS CitaV_ID_Medico,
6         |CitaV.Nombres_Med AS CitaV_Nombres_Med,
7         |CitaV.Apellidos_Med AS CitaV_Apellidos_Med,
8         |CitaV.Fecha_Cita AS CitaV_Fecha_Cita,
9         |DAYNAME(CitaV.Fecha_Cita) AS CitaV_Dia_Cita,
10        |CitaV.Hora_Cita AS CitaV_Hora_Cita,
11        |WEEK(CitaV.Fecha_Cita) AS CitaV_Semana_Cita,
12        |CitaV.Nombres_Pac AS CitaV_Nombres_Pac,
13        |CitaV.Apellidos_Pac AS CitaV_Apellidos_Pac,
14        |CitaV.Estado AS CitaV_Estado
15    FROM
16        |CitaV
17    WHERE
18        |WEEK(CitaV.Fecha_Cita) = ?; -- Filtrando por semana
19    AND
20        |CitaV.ID_Medico = ?; -- Filtrando por ID Medico
21    """;
```

```
22
23    con = cn.getConexion();
24    ps = con.prepareStatement(sql);
25    ps.setInt(1, semana); // Se pasa el parámetro de la semana
26    ps.setInt(2, idMedico); // Se pasa el parámetro de ID Medico
27    rs = ps.executeQuery();
28
29    // Si se encuentran resultados
30    if (rs.next()) {
31        // Datos del médico
32        Paragraph datosMedico = new Paragraph();
33        Font subheadingFont = new Font(Font.FontFamily.TIMES_ROMAN, 14, Font.BOLD, BaseColor.DARK_GRAY);
34        datosMedico.add(new Phrase("Médico: ", subheadingFont));
35        datosMedico.add(rs.getString("CitaV_Nombres_Med") + " " + rs.getString("CitaV_Apellidos_Med"));
36        datosMedico.add(Chunk.NEWLINE);
37        datosMedico.add(new Phrase("ID Médico: ", subheadingFont));
38        datosMedico.add(String.valueOf(rs.getInt("CitaV_ID_Medico")));
39        datosMedico.add(Chunk.NEWLINE);
40        doc.add(datosMedico);
41
42        // Datos de la cita
43        PdfPTable tablaConsulta = new PdfPTable(2);
44        tablaConsulta.setWidthPercentage(100);
45        tablaConsulta.setSpacingBefore(10f);
46        tablaConsulta.setSpacingAfter(10f);
47
48        Font boldFont = new Font(Font.FontFamily.TIMES_ROMAN, 12, Font.BOLD, BaseColor.BLACK);
49        tablaConsulta.addCell(new PdfPCell(new Phrase("Fecha de Cita", boldFont)));
50        tablaConsulta.addCell(rs.getString("CitaV_Fecha_Cita") != null ? rs.getString("CitaV_Fecha_Cita") : "No disponible");
51    }
52
53    rs.close();
54    ps.close();
55    con.close();
56
57    response.setContentType("application/pdf");
58    response.setHeader("Content-Disposition", "attachment; filename=HorarioPorMedicoSemanal.pdf");
59    OutputStream os = response.getOutputStream();
60    doc.writeTo(os);
61    os.flush();
62    os.close();
```

26. Posteriormente seleccionamos el modo reporte que es semanal o mensual



27. También para generar el reporte mensual , se selecciona el mes así como muestra el ejemplo

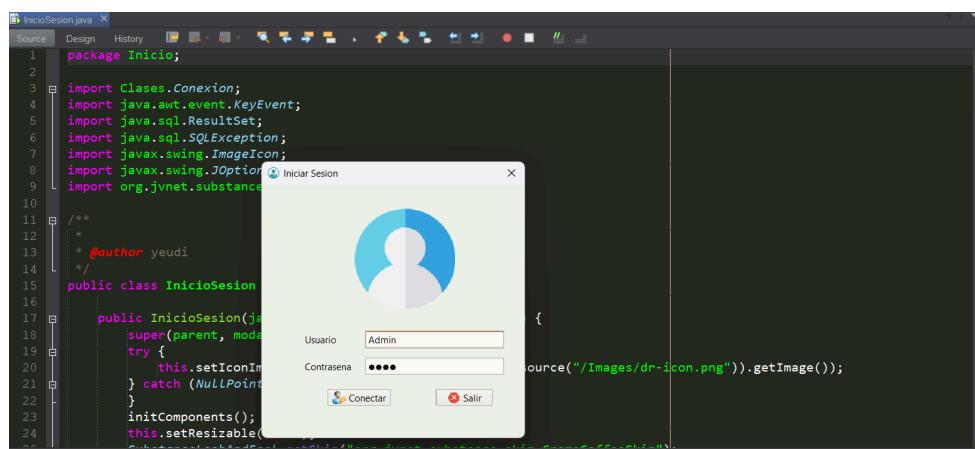


De esta manera funciona el reporte mensual ocupando una consulta y generando el pdf

```
78     // Consulta SQL para obtener citas
79     String sql = """
80         SELECT
81             CitaV.ID_Cita AS CitaV_ID_Cita,
82             CitaV.ID_Medico AS CitaV_ID_Medico,
83             CitaV.Nombres_Med AS CitaV_Nombres_Med,
84             CitaV.Apellidos_Med AS CitaV_Apellidos_Med,
85             CitaV.Fecha_Cita AS CitaV_Fecha_Cita,
86             DAY(CitaV.Fecha_Cita) AS CitaV_Dia_Cita,
87             CitaV.Hora_Cita AS CitaV_Hora_Cita,
88             WEEK(CitaV.Fecha_Cita, 1) AS CitaV_Semana_Cita,
89             CitaV.Nombres_Pac AS CitaV_Nombres,
90             CitaV.Apellidos_Pac AS CitaV_Apellidos,
91             CitaV.Estado AS CitaV_Estado
92         FROM
93             CitaV
94         WHERE
95             MONTH(CitaV.Fecha_Cita) = ? -- Filtrando por mes
96             AND YEAR(CitaV.Fecha_Cita) = ?; -- Filtrando por año
97     """;
98
99     ps = con.prepareStatement(sql);
100    ps.setInt(1, mes); // Se pasa el parámetro del mes
101    ps.setInt(2, anio); // Se pasa el parámetro del año
102    rs = ps.executeQuery();
103
104    // Si se encuentran resultados
105    if (rs.next()) {
106        // Datos del médico
107        Paragraph datosMedico = new Paragraph();
108        Font subheadingFont = new Font(Font.FontFamily.TIMES_ROMAN, 14, Font.BOLD, BaseColor.DARK_GRAY);
109        datosMedico.add(new Phrase("Médico: ", subheadingFont));
110        datosMedico.add(rs.getString("CitaV_Nombres_Med") + " " + rs.getString("CitaV_Apellidos_Med"));
111        datosMedico.add(Chunk.NEWLINE);
112        datosMedico.add(new Phrase("ID Médico: ", subheadingFont));
113        datosMedico.add(String.valueOf(rs.getInt("CitaV_ID_Medico")));
114        datosMedico.add(Chunk.NEWLINE);
115        doc.add(datosMedico);
116
117        // Datos de la cita
118        PdfPTable tablaConsulta = new PdfPTable(2);
119        tablaConsulta.setWidthPercentage(100);
120        tablaConsulta.setSpacingBefore(10f);
121        tablaConsulta.setSpacingAfter(10f);
122
123        Font boldFont = new Font(Font.FontFamily.TIMES_ROMAN, 12, Font.BOLD, BaseColor.BLACK);
124        PdfPCell celda1 = new PdfPCell(new Phrase("Fecha de Cita", boldFont));
125        PdfPCell celda2 = new PdfPCell(new Phrase("Número de Cita", boldFont));
```

PRUEBAS FUNCIONALES

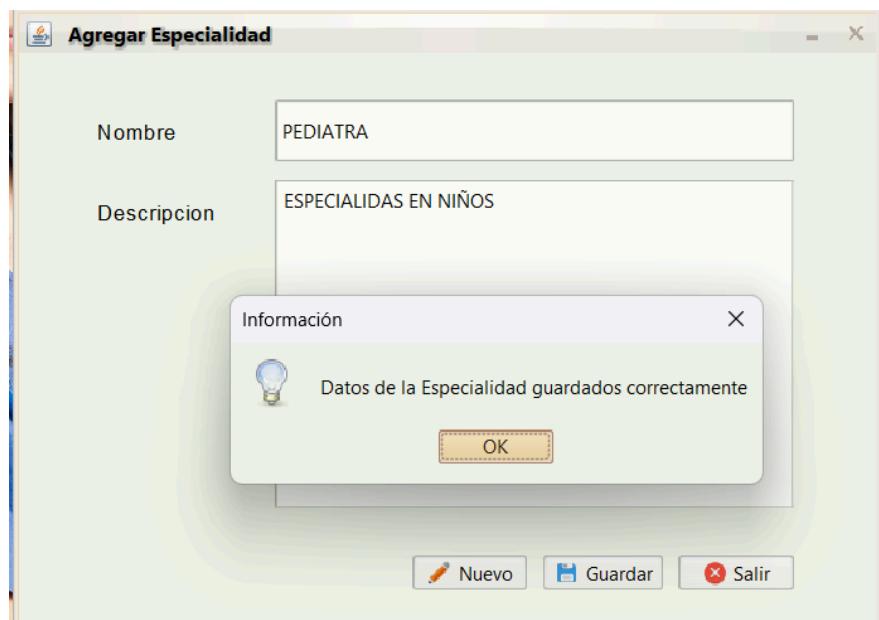
1. Se loguea primordialmente como el admin



2. Se tiene la vista general del sistema



3. El administrador da de altas las especialidades sugeridas por los medicos



4. En este apartado se puede visualizar las especialidades ya agregadas en el sistema

Ver Especialidad

Buscar:

No.	Nombre	Descripcion	Estado
1	Pediatra	ninguna	Activo
2	NUEVO	NUEVO	Activo
3	PRUEBA FINAL	PRUEBA FINAL	Activo
4	ESPECIALISTA	MEDICO CERTIFICADO	Activo
5	PEDIATRA	ESPECIALIDADES EN NIÑOS	Activo

Activar/Desactivar Modificar Salir

5. Se registra el médico con sus datos solicitados

Agregar Medico

Nombres	YEUDIEL
Apellidos	AUGUSTO
Especialidad	ESPECIALISTA
Cedula	12345678

6. Se le asigna el horario correspondiente

Agregar Horario

Dias disponibles	<input checked="" type="checkbox"/> L	<input checked="" type="checkbox"/> M	<input type="checkbox"/> X	<input type="checkbox"/> J	<input type="checkbox"/> V	<input type="checkbox"/> S	<input type="checkbox"/> D
Lunes	Desde	8:00 A.M	Hasta	2:00 P.M			
Martes	Desde	8:00 A.M	Hasta	10:00 A.M			

7. Se visualiza los médicos ya registrados en el sistema

Ver Medico

No.	Nombres	Apellidos	Cedula	Especialidad	Estado
1	yeudiel	agusto	1234567	Pediatra	Activo
2	Yeudiel	Augusto	1234567	Pediatra	Activo
3	BOLIVARE	GARCIA OSORIO	12345678	Pediatra	Activo
4	Jacob	N	12345678	Pediatra	Activo
5	treinta de nov	dosmilveinticuatro	3244534	Pediatra	Activo
6	yeu	yeu	123	Pediatra	Activo
7	EMMANUEL	H	12312	NUEVO	Activo
8	GERSON v	MERARI	8765432	PRUEBA FINAL	Activo
9	PRUEBA	FINAL	12345678	PRUEBA FINAL	Activo

8. Se loguea como la secretaría el cual asigna citas para el medico

The screenshot shows a Java IDE interface with the code for `InicioSesion.java` in the editor. The code is a Java Swing application for logging in. A modal dialog box titled "Iniciar Sesion" is displayed, featuring a user icon, two text input fields for "Usuario" (with value "jenni") and "Contrasena" (with masked value "●●●"), and two buttons labeled "Conectar" and "Salir".

```
package Inicio;
import Clases.Conexion;
import java.awt.event.KeyEvent;
import java.sql.ResultSet;
import java.sql.SQLException;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;
import org.jvnet.substance.*;

public class InicioSesion extends javax.swing.JFrame {
    /**
     * @author yeudi
     */
    public InicioSesion() {
        super("Iniciar Sesion");
        try {
            this.setIconImage(new ImageIcon(getClass().getResource("//Images/dr-icon.png")).getImage());
        } catch (NullPointerException ex) {
        }
        initComponents();
        this.setResizable(false);
    }

    private void initComponents() {
        // Initialization code here
    }
}
```

9. Vista general de la secretaria



10. La secretaria hace registro de un nuevo paciente

Agregar Paciente

Nombres	GERSON
Apellidos	MERARI
Teléfono	0192-8736
Peso	80
Altura	170
Edad	24
Tipo de Sangre	A+
Alergias	RESFRIADO
Enfermedades	FARENGITIS

Nuevo
 Guardar
 Salir

11. Se ven los registros de los pacientes agregados

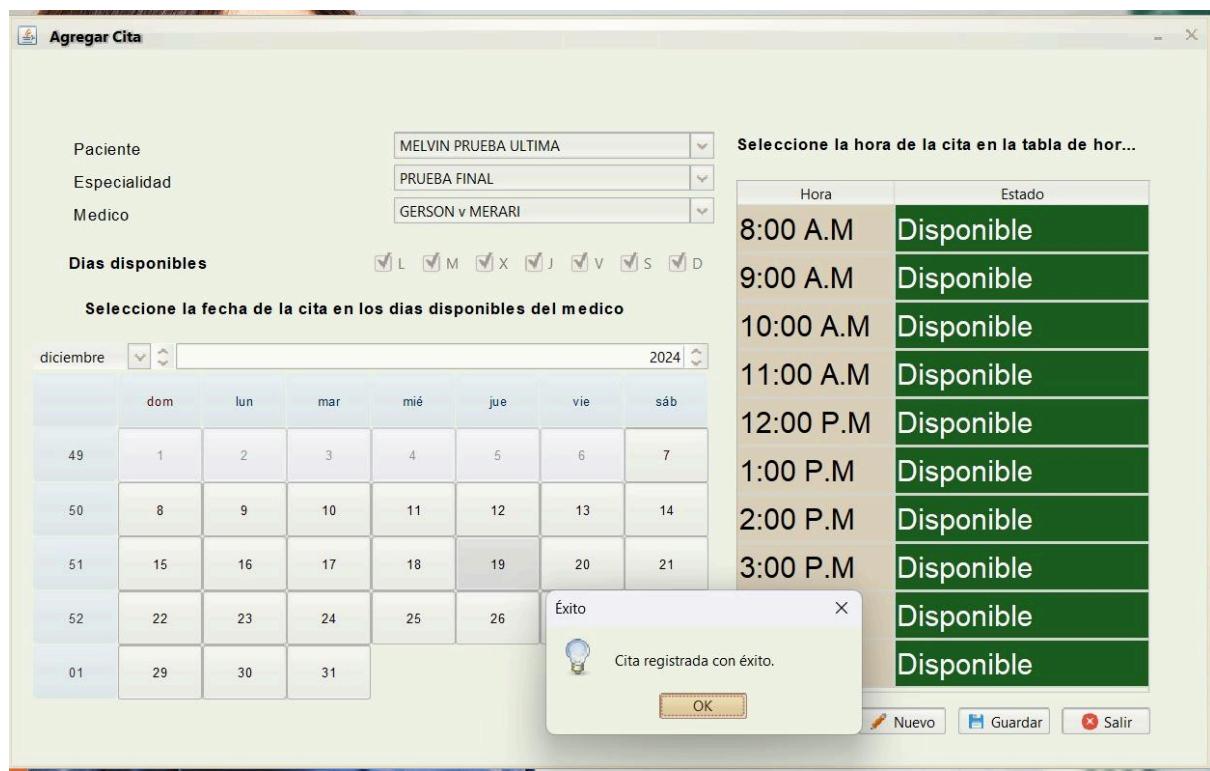
Ver Paciente

Buscar:

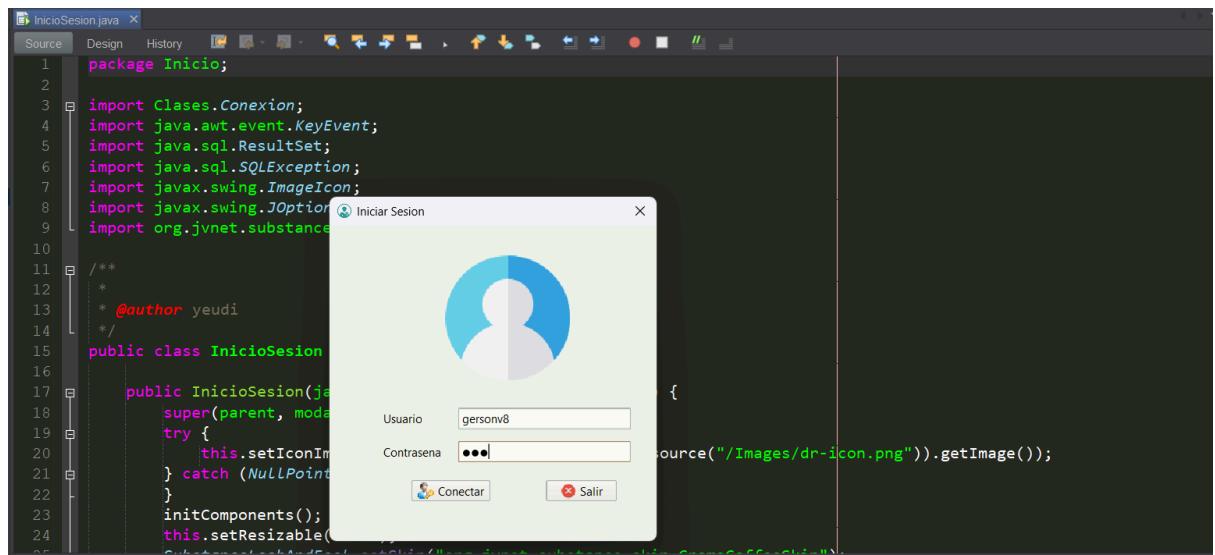
No.	Nombres	Apellidos	Peso	Altura	Edad	Telefono	Alergias	Enfermed...	Tipo Sang...	Estado
1	Bolivare	Garcia Os...	85.0	170.0	24	1234-5670	Resfriado	diabetes	O+	Activo
2	Alma	M	60.0	155.0	21	1234-5678	NINGUNA	MENSA	O-	Activo
3	Miguel	H	50.0	160.0	18	1234-5678	Resfriado	Ninguna	O+	Activo
4	N	N	90.0	170.0	23	1234-5678	N	N	O-	Activo
5	DANILO	DANILO	80.0	70.0	18	1232-8313	N	N	O-	Activo
6	PRUEBA ...	HOLA	80.0	177.0	21	1298-0128	HOLA	MUERTE	A+	Activo
7	unmo	uno	90.0	1221.0	23	1238-2183	n	n	O-	Activo
8	almagacel	s	123.0	88.0	21	9876-5434	n	n	O+	Activo
9	y	eu	21.0	88.0	21	9087-7765	ds	das	O-	Activo
10	j	h	44.0	77.0	23	0987-6543	d	d	A+	Activo
11	YEUDIEL	AUGUSTO	70.0	175.0	21	9876-5412	ANTIBIO...	DOLOR ...	A+	Activo
12	DIEGO	BARRIOS	78.0	170.0	21	1298-3765	NINGUNA	DIABETES	A+	Activo
13	KEVIN	HIDALGO	70.0	160.0	24	9876-5415	GRIPE	IRRITACI...	A+	Activo
14	MELVIN	PRUEBA ...	80.0	180.0	23	1232-3123	NINGUNA	GRIPA	A+	Activo

Ver Expediente
 Activar/Desactivar
 Modificar
 Salir

12. Se agrega las citas con el medicamento sugerido y el paciente a proceder atender



13. Se loguea como médico



14. Hace revisión de los pacientes atender y de igual manera verifica su horario en que dia tiene citas

Agregar Consulta

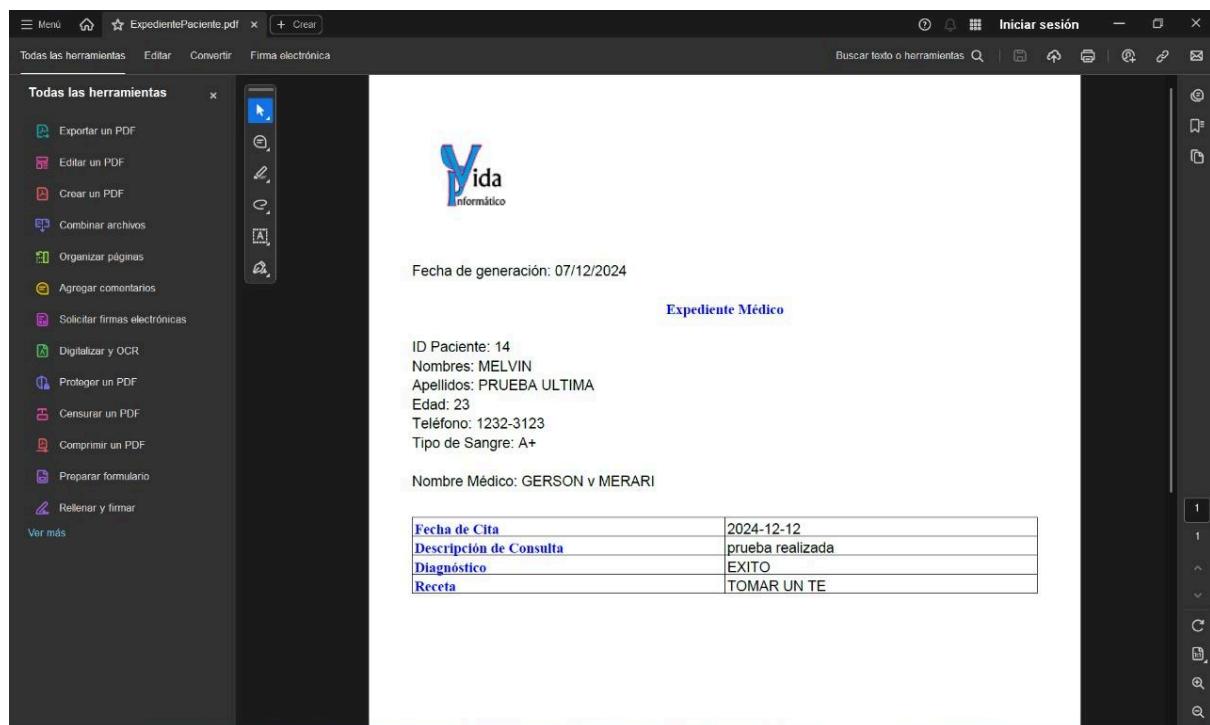
Días disponibles							Seleccione la hora de la cita en la tabla de horario																																																
							Hora	Estado																																															
Seleccione la fecha de la cita en los días disponibles del médico diciembre <input type="button" value="▼"/> <input type="button" value="▲"/> 2024 <input type="button" value="▼"/> <input type="button" value="▲"/> <table border="1"> <thead> <tr> <th>dom</th> <th>lun</th> <th>mar</th> <th>mié</th> <th>jue</th> <th>vie</th> <th>sáb</th> </tr> </thead> <tbody> <tr> <td>49</td> <td>1</td> <td>2</td> <td>3</td> <td>4</td> <td>5</td> <td>6</td> <td>7</td> </tr> <tr> <td>50</td> <td>8</td> <td>9</td> <td>10</td> <td>11</td> <td>12</td> <td>13</td> <td>14</td> </tr> <tr> <td>51</td> <td>15</td> <td>16</td> <td>17</td> <td>18</td> <td>19</td> <td>20</td> <td>21</td> </tr> <tr> <td>52</td> <td>22</td> <td>23</td> <td>24</td> <td>25</td> <td>26</td> <td>27</td> <td>28</td> </tr> <tr> <td>01</td> <td>29</td> <td>30</td> <td>31</td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>							dom	lun	mar	mié	jue	vie	sáb	49	1	2	3	4	5	6	7	50	8	9	10	11	12	13	14	51	15	16	17	18	19	20	21	52	22	23	24	25	26	27	28	01	29	30	31					8:00 A.M	Libre
dom	lun	mar	mié	jue	vie	sáb																																																	
49	1	2	3	4	5	6	7																																																
50	8	9	10	11	12	13	14																																																
51	15	16	17	18	19	20	21																																																
52	22	23	24	25	26	27	28																																																
01	29	30	31																																																				
							9:00 A.M	Libre																																															
							10:00 A.M	Libre																																															
							11:00 A.M	Libre																																															
							12:00 P.M	Libre																																															
							1:00 P.M	Libre																																															
							2:00 P.M	Cita con MELVIN PRUEBA ULTIMA (Pendiente)																																															
							3:00 P.M	Libre																																															
							4:00 P.M	Libre																																															
							5:00 P.M	Libre																																															

15. se verifica la consulta , diagnóstico y receta

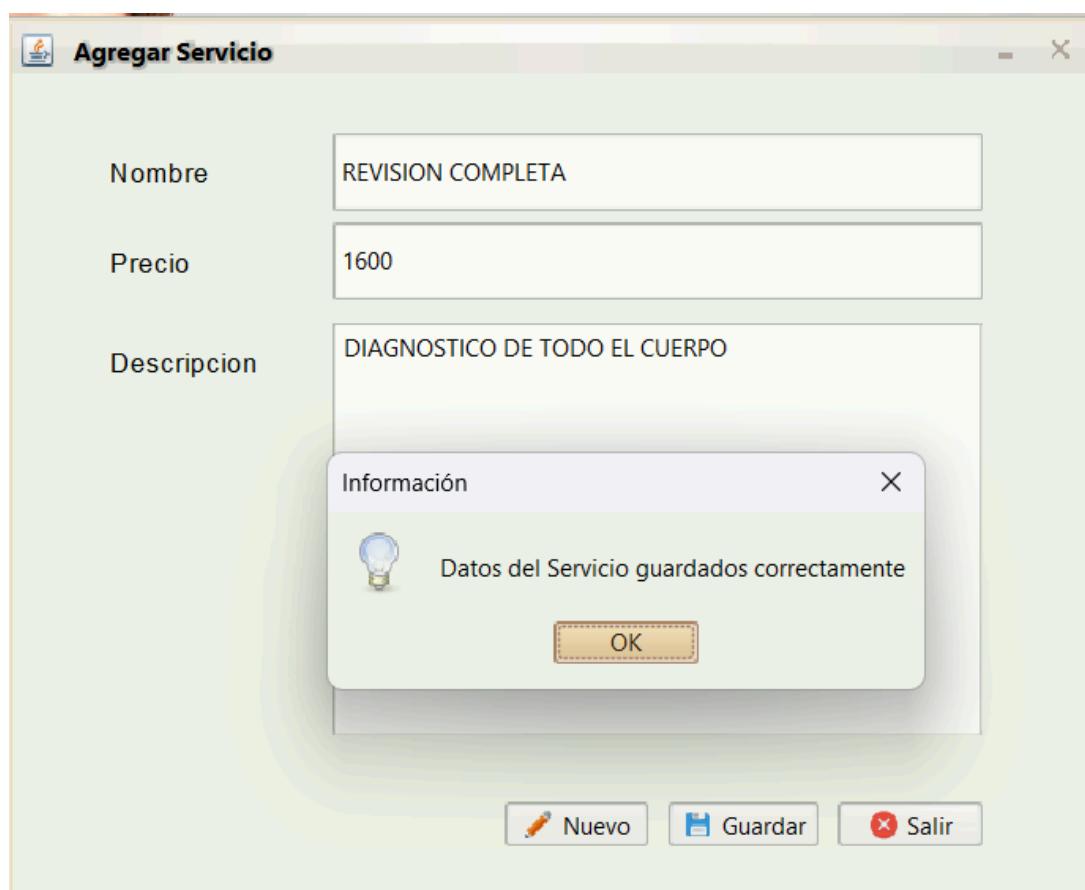
Agregar Consulta | Diagnóstico | Receta | Archivos al expediente

Paciente	MELVIN PRUEBA ULTIMA	Doctor	PRUEBA FINAL
Consulta	Se atiende por alergias fuertes		
Diagnóstico	Estado mal		
Receta	Reposo por 72 hrs <div style="border: 1px solid #ccc; padding: 5px; width: fit-content; margin: auto;">  Datos de la Consulta guardados correctamente <input type="button" value="OK"/> </div>		
<input type="button" value="Agregar archivos al expediente"/> <input type="button" value="Guardar"/> <input type="button" value="Salir"/>			

16. Se genera un expediente sobre el paciente



17. El administrador hace registros sobre los servicios existentes



18. Se visualiza los servicios registrados

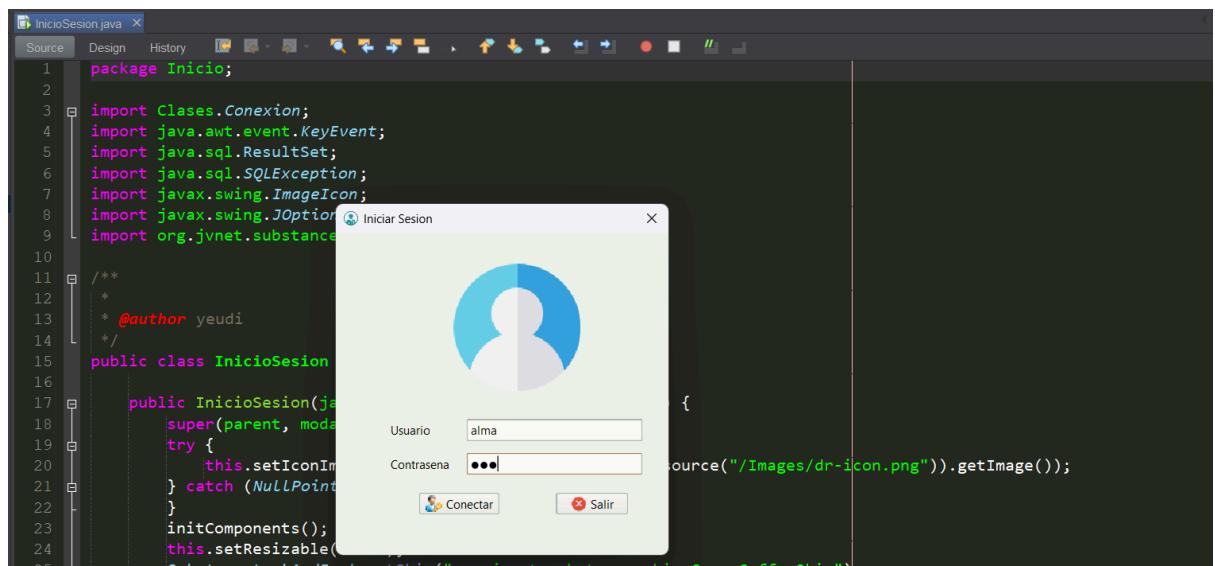
Ver Servicio

Buscar:

No.	Nombre	Descripcion	Precio	Estado
1	fiebre	nada	180.0	Activo
2	REVISION COMPLETA	DIAGNOSTICO DE TOD...	1600.0	Activo

Activar/Desactivar Modificar Salir

19. Se necesita loguearse como caja para cobrar los servicios al paciente



20. vista general del sistema para la caja



21. Se genera el registro de factura sobre los pagos

Agregar Factura

Fecha	08-dic-2024
Cliente	MELVIN PRUEBA ULTIMA
Medico	PRUEBA FINAL
Servicio	REVISION COMPLETA

Información

Datos del Pago guardado correctamente

TOTAL 1780.0

Botones: Nuevo, Guardar, Salir

22. Vista general sobre los pagos ya realizados

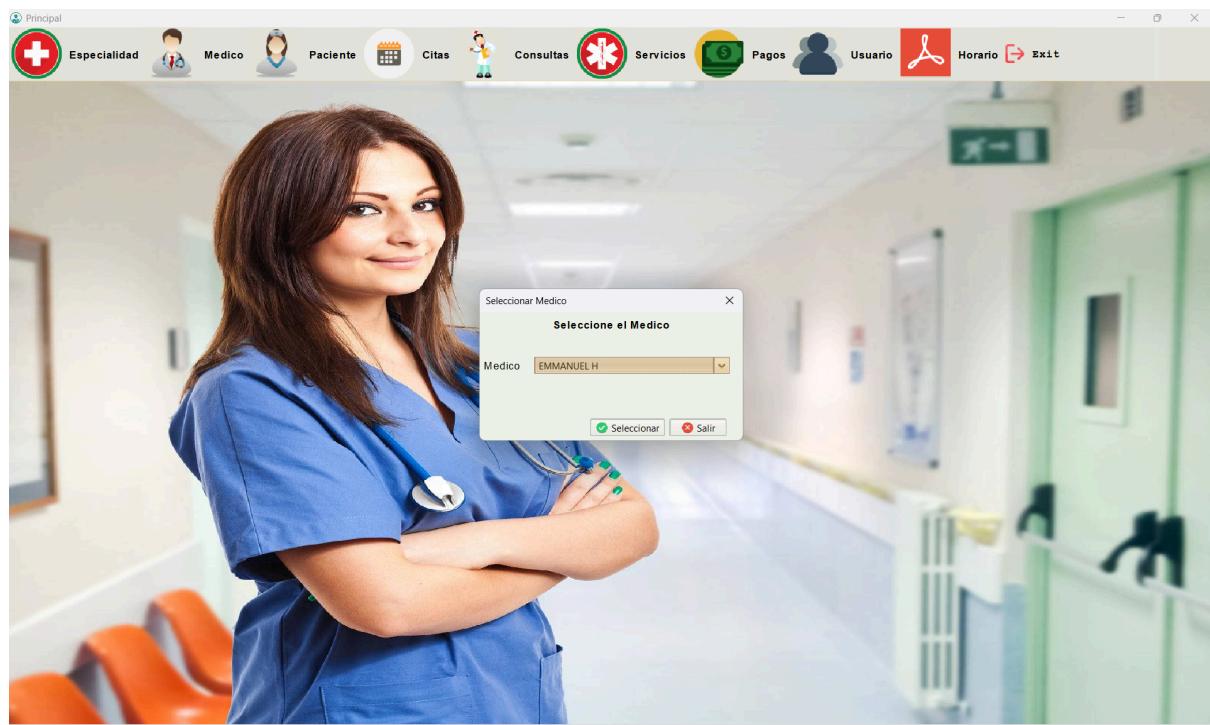
Ver Pago

Buscar:

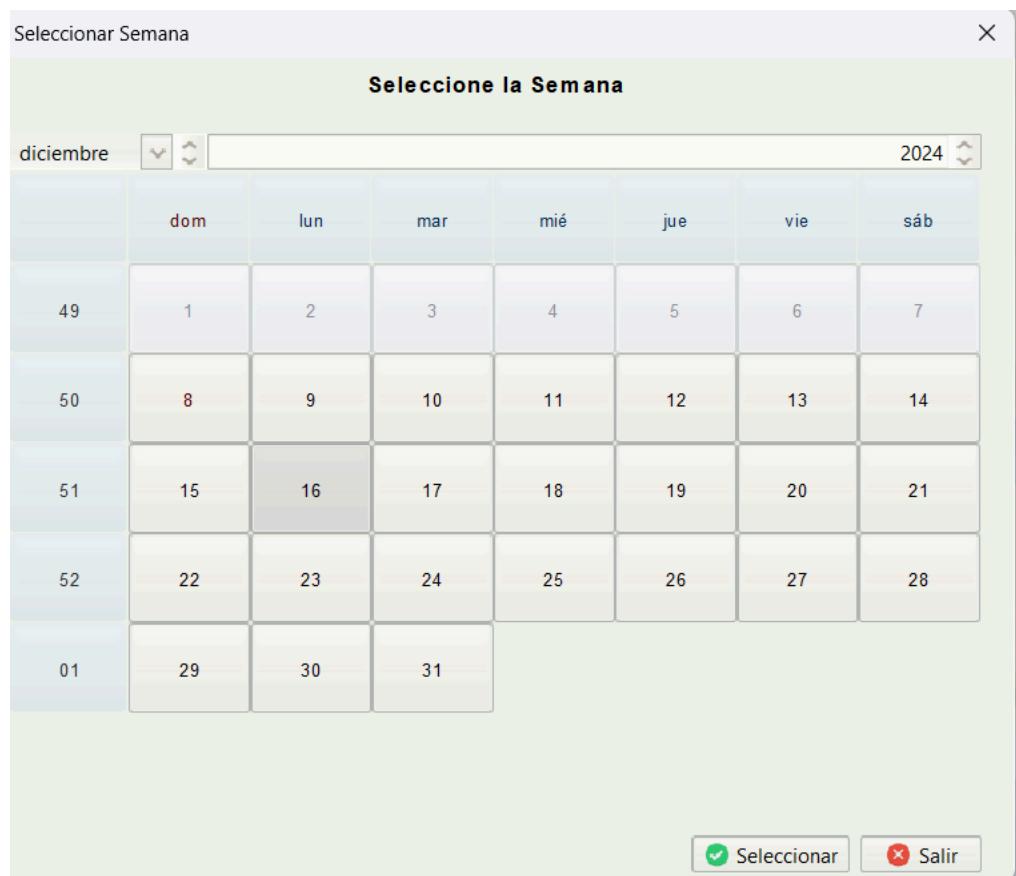
No.	Fecha	Cliente	Usuario	Total	Estado
1	16-nov.-2024	Bolivare Garcia Oso...	yeudiel2	180.0	Cancelado
2	17-nov.-2024	Alma M	yeudiel2	180.0	Cancelado
3	30-nov.-2024	unmo uno	jenni	180.0	Cancelado
3	30-nov.-2024	unmo uno	jenni	180.0	Cancelado
4	07-dic.-2024	MELVIN PRUEBA U...	gersonv8	1600.0	Activo
4	07-dic.-2024	MELVIN PRUEBA U...	gersonv8	180.0	Activo

Cancelar **Salir**

23. El administrador genera los reportes sobre los medicos



24. Se selecciona la semana del medico



25. Por último se genera el pdf del reporte

Médico: EMMANUEL H
ID Médico: 7

Fecha de Cita	2024-12-25
Dia de la Cita	Wednesday
Hora de la Cita	5:00 P.M.
Semana de la Cita	51
Paciente	Alma M
Estado de la Cita	Atendida

Firma del Médico: _____

REFERENCIAS Y RECURSOS

Enlaces Útiles

- **Documentación de NetBeans IDE:** Guía oficial del entorno de desarrollo integrado utilizado en el proyecto. <https://netbeans.apache.org/kb/>
- **Documentación de MySQL:** Manual oficial del sistema de gestión de bases de datos empleado. <https://dev.mysql.com/doc/>
- **Documentación de JasperReports:** Recursos y guías sobre la herramienta de generación de reportes utilizada. <https://community.jaspersoft.com/documentation>
- **Apache Commons Libraries:** Colección de bibliotecas Java utilizadas en el proyecto. <https://commons.apache.org/>
- **Bizagi Modeler:** Herramienta para modelado de procesos de negocio. <https://www.bizagi.com/es/productos/bpm-suite/modeler>

Bibliografía

1. García, A. (2010). *El sistema de información del hospital*. En A. García (Ed.), *Gestión hospitalaria* (pp. 45-67). Editorial Médica Panamericana.
2. González Bernaldo de Quirós, F., & Luna, D. (2010). *La historia clínica electrónica*. En A. García (Ed.), *Gestión hospitalaria* (pp. 68-90). Editorial Médica Panamericana.
3. Rojas, D. (2010). *La gestión integral de peticiones clínicas en el ámbito de la salud electrónica*. En A. García (Ed.), *Gestión hospitalaria* (pp. 91-112). Editorial Médica Panamericana.
4. Aguirre Gas, H. G. (2012). *Calidad de la atención médica: Bases para su evaluación y mejoramiento continuo* (3^a ed.). Editorial Médica Panamericana.
5. Fajardo Dolci, G., & Hernández Torres, F. (2012). *Definiciones y conceptos fundamentales para el mejoramiento de la calidad de la atención a la salud*. Secretaría de Salud.
6. De Savigny, D. (2010). *Aplicación del pensamiento sistémico al fortalecimiento de los sistemas de salud: Seguir avanzando*. Organización Mundial de la Salud.
7. Osuna, C. (2015). *Los sistemas de gestión de contenidos en Información y Documentación*. *Revista Española de Documentación Científica*, 38(2), e086.
8. Villafuerte Salas, J., & Villanueva Yana, M. (2019). *Sistema de gestión de la información de las historias clínicas en el Hospital "Augusto Bernardino Leguía" de la Policía Nacional del Perú* [Tesis de maestría, Pontificia Universidad Católica del Perú].
9. García, A. (2010). *El sistema de información del hospital*. En A. García (Ed.), *Gestión hospitalaria* (pp. 45-67). Editorial Médica Panamericana.
10. González Bernaldo de Quirós, F., & Luna, D. (2010). *La historia clínica electrónica*. En A. García (Ed.), *Gestión hospitalaria* (pp. 68-90). Editorial Médica Panamericana.
11. Rojas, D. (2010). *La gestión integral de peticiones clínicas en el ámbito de la salud electrónica*. En A. García (Ed.), *Gestión hospitalaria* (pp. 91-112). Editorial Médica Panamericana.
12. Aguirre Gas, H. G. (2012). *Calidad de la atención médica: Bases para su evaluación y mejoramiento continuo* (3^a ed.). Editorial Médica Panamericana.

Glosario

- **API (Application Programming Interface)**: Conjunto de definiciones y protocolos que permite la comunicación entre diferentes aplicaciones de software.

- **Base de Datos Relacional:** Sistema de almacenamiento de datos que organiza la información en tablas relacionadas entre sí.
- **Bizagi Modeler:** Herramienta de modelado de procesos de negocio que permite diagramar y documentar flujos de trabajo.
- **CRUD (Create, Read, Update, Delete):** Acrónimo que representa las cuatro operaciones básicas de persistencia en bases de datos.
- **Diagrama Entidad-Relación (ER):** Representación gráfica que muestra las entidades de un sistema y las relaciones entre ellas.
- **Extreme Programming (XP):** Metodología ágil de desarrollo de software que enfatiza la flexibilidad y la entrega frecuente de versiones funcionales.
- **Framework:** Conjunto de herramientas y bibliotecas que proporcionan una estructura estándar para el desarrollo de aplicaciones.
- **JDBC (Java Database Connectivity):** API de Java que permite la conexión y ejecución de operaciones sobre bases de datos.
- **Metodología Ágil:** Enfoque de desarrollo de software que promueve la entrega incremental, colaboración continua y adaptación al cambio.
- **NetBeans IDE:** Entorno de desarrollo integrado para aplicaciones Java y otros lenguajes de programación.
- **Pruebas Unitarias:** Evaluaciones individuales de las unidades más pequeñas de código para asegurar su correcto funcionamiento.
- **UML (Unified Modeling Language):** Lenguaje de modelado estándar utilizado para visualizar y documentar sistemas de software.
- **XP (Extreme Programming):** Ver Extreme Programming.