**Documento de Desarrollo: Simulador de Gestión de Consultas Médicas**

**Nombre del estudiante:** Edgar Felipe Beltrán Cárdenas

Parcial #2 – Programación Orientada a Objetos

## 1. Introducción al Proyecto

Este documento detalla el diseño e implementación de un sistema para gestionar consultas médicas en una clínica, siguiendo el patrón de arquitectura MVVM (Model-View-ViewModel) y utilizando Java Swing para la interfaz gráfica, Firebase Realtime Database para la persistencia de datos, y excepciones personalizadas para un manejo robusto de errores.

### 1.1. Requerimientos Funcionales

El sistema cumple con los siguientes requisitos funcionales:

| REQUISITOS FUNCIONALES | CUMPLIDO |
|---|---|
| Login de usuario médico o administrativo | |
| Registrar nuevos pacientes y médicos | |
| Asignar una consulta a un paciente con un médico y registrar síntomas, diagnóstico y tratamiento | |
| Consultar el historial médico de un paciente | |
| Listar todas las consultas realizadas por un médico | |

### 1.2. Requisitos Técnicos

- Se han implementado los siguientes requisitos técnicos:
- Implementación del patrón MVVM.
- Uso de al menos tres relaciones entre clases (herencia, composición, asociación).
- Implementación de una clase abstracta e interfaces.
- Uso de colecciones genéricas para almacenar pacientes, médicos y consultas.
- Persistencia de datos en base de datos (Firebase Realtime Database).
- Manejo de excepciones personalizadas (UsuarioNoEncontradoException, CampoVacioException).
- Interfaz gráfica en JSwing, con menús funcionales.

## 2. Estructura del Proyecto

El proyecto está organizado en paquetes siguiendo una estructura modular para separar las responsabilidades.

```
Clinica/
│
├── model/
│   ├── Persona.java // Clase abstracta
│   ├── Paciente.java
│   ├── Medico.java
│   ├── Consulta.java
│   ├── Clinica.java // Contiene colecciones
│   └── IPersistencia.java // Interfaz
│
├── view/
│   ├── VentanaPrincipal.java // JFrame con menú principal
│   ├── PanelRegistro.java // Panel para registrar personas
│   ├── PanelConsulta.java // Panel para asignar y registrar consultas
│   └── PanelHistorial.java // Panel para consultar datos
│
├── viewmodel/
│   ├── ClinicaViewModel.java // Conecta modelo y vista
│
├── persistencia/
│   └── PersistenciaArchivo.java // Implementación de IPersistencia
│
├── excepciones/
```

```
│ ├── UsuarioNoEncontradoException.java
│ └── CampoVacioException.java
│
└── Main.java
```

**3. Desarrollo Paso a Paso**

**3.1. Modelo:**

El paquete “model” contiene la lógica de negocio central y las entidades de datos del sistema.

- Persona.java
  Define atributos y métodos comunes para “Paciente` y `Medico”, estableciendo una relación de herencia.

```
package com.simulador.model;

public abstract class Persona {

   protected String id;
   protected String nombre;
   protected String telefono;
   protected String email;

   public Persona(String id, String nombre, String telefono, String email) {
      this.id = id;
      this.nombre = nombre;
      this.telefono = telefono;
      this.email = email;
   }

   // Getters y Setters
   public String getId() {
      return id;
   }

   public void setId(String id) {
      this.id = id;
   }

   public String getNombre() {
      return nombre;
   }

   public void setNombre(String nombre) {
      this.nombre = nombre;
```

```
    }

    public String getTelefono() {
        return telefono;
    }

    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    // Método abstracto (ejemplo, puedes añadir más según necesidad)
    public abstract String getTipoPersona();

    @Override
    public String toString() {
        return "ID: " + id + ", Nombre: " + nombre;
    }
}
```

- Paciente.java
  Hereda de `Persona` y añade atributos específicos como `fechaNacimiento` e `historialMedicoGeneral`.

```
package com.simulador.model;

public class Paciente extends Persona {

    private String fechaNacimiento;
    private String historialMedicoGeneral; // Podría ser una lista de condiciones, etc.

    public Paciente(String id, String nombre, String telefono, String email, String fechaNacimiento, String historialMedicoGeneral) {
        super(id, nombre, telefono, email);
        this.fechaNacimiento = fechaNacimiento;
        this.historialMedicoGeneral = historialMedicoGeneral;
    }

    // Constructor vacío para Firebase (muy importante para deserialización)
```

```
    public Paciente() {
        super("", "", "", ""); // Llama al constructor de Persona con valores
    predeterminados
    }

    // Getters y Setters específicos de Paciente
    public String getFechaNacimiento() {
        return fechaNacimiento;
    }

    public void setFechaNacimiento(String fechaNacimiento) {
        this.fechaNacimiento = fechaNacimiento;
    }

    public String getHistorialMedicoGeneral() {
        return historialMedicoGeneral;
    }

    public void setHistorialMedicoGeneral(String historialMedicoGeneral) {
        this.historialMedicoGeneral = historialMedicoGeneral;
    }

    @Override
    public String getTipoPersona() {
        return "Paciente";
    }

    @Override
    public String toString() {
        return "Paciente " + super.toString() + ", Fecha Nacimiento: " +
    fechaNacimiento;
    }
}
```

- Medico.java
  También hereda de `Persona` y añade atributos como `especialidad` y `licenciaMedica`.

```
package com.simulador.model;

public class Medico extends Persona {

    private String especialidad;
    private String licenciaMedica;

    public Medico(String id, String nombre, String telefono, String email, String
    especialidad, String licenciaMedica) {
        super(id, nombre, telefono, email);
```

```
        this.especialidad = especialidad;
        this.licenciaMedica = licenciaMedica;
    }

    // Constructor vacío para Firebase
    public Medico() {
        super("", "", "", ""); // Llama al constructor de Persona con valores
predeterminados
    }

    // Getters y Setters específicos de Medico
    public String getEspecialidad() {
        return especialidad;
    }

    public void setEspecialidad(String especialidad) {
        this.especialidad = especialidad;
    }

    public String getLicenciaMedica() {
        return licenciaMedica;
    }

    public void setLicenciaMedica(String licenciaMedica) {
        this.licenciaMedica = licenciaMedica;
    }

    @Override
    public String getTipoPersona() {
        return "Medico";
    }

    @Override
    public String toString() {
        return "Medico " + super.toString() + ", Especialidad: " + especialidad;
    }
}
```

- Consulta.java

  Representa una consulta médica con detalles como ID, paciente, médico, fecha, síntomas, diagnóstico y tratamiento.

```
package com.simulador.model;

import java.util.Date;
```

```
public class Consulta {

    private String idConsulta;
    private String idPaciente; // Referencia al paciente por ID
    private String idMedico;   // Referencia al médico por ID
    private String fechaConsulta; // Podría ser Date, pero String para simplicidad inicial con Firebase
    private String sintomas;
    private String diagnostico;
    private String tratamiento;

    public Consulta(String idConsulta, String idPaciente, String idMedico, String fechaConsulta, String sintomas, String diagnostico, String tratamiento) {
        this.idConsulta = idConsulta;
        this.idPaciente = idPaciente;
        this.idMedico = idMedico;
        this.fechaConsulta = fechaConsulta;
        this.sintomas = sintomas;
        this.diagnostico = diagnostico;
        this.tratamiento = tratamiento;
    }

    // Constructor vacío para Firebase
    public Consulta() {
    }

    // Getters y Setters
    public String getIdConsulta() {
        return idConsulta;
    }

    public void setIdConsulta(String idConsulta) {
        this.idConsulta = idConsulta;
    }

    public String getIdPaciente() {
        return idPaciente;
    }

    public void setIdPaciente(String idPaciente) {
        this.idPaciente = idPaciente;
    }

    public String getIdMedico() {
        return idMedico;
    }

    public void setIdMedico(String idMedico) {
```

```
        this.idMedico = idMedico;
    }

    public String getFechaConsulta() {
        return fechaConsulta;
    }

    public void setFechaConsulta(String fechaConsulta) {
        this.fechaConsulta = fechaConsulta;
    }

    public String getSintomas() {
        return sintomas;
    }

    public void setSintomas(String sintomas) {
        this.sintomas = sintomas;
    }

    public String getDiagnostico() {
        return diagnostico;
    }

    public void setDiagnostico(String diagnostico) {
        this.diagnostico = diagnostico;
    }

    public String getTratamiento() {
        return tratamiento;
    }

    public void setTratamiento(String tratamiento) {
        this.tratamiento = tratamiento;
    }

    @Override
    public String toString() {
        return "Consulta ID: " + idConsulta + ", Paciente ID: " + idPaciente + ", Medico ID: "
+ idMedico
                + ", Fecha: " + fechaConsulta + ", Diagnóstico: " + diagnostico;
    }
}
```

- Clinica.java
  Es la clase principal del modelo, gestionando las colecciones de `pacientes`, `medicos` y `consultas`. Implementa relaciones de **composición** con estas colecciones. Utiliza una **asociación** con `IPersistencia` para la carga y guardado de datos.

```
package com.simulador.model;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID; // Para generar IDs únicos si no vienen de Firebase

public class Clinica {

   private List<Paciente> pacientes;
   private List<Medico> medicos;
   private List<Consulta> consultas;

   // Aunque la persistencia se manejará en una clase aparte,
   // Clinica podría tener una referencia a la interfaz de persistencia
   // para coordinar las operaciones del modelo.
   private IPersistencia persistenciaService;

   public Clinica() {
      this.pacientes = new ArrayList<>();
      this.medicos = new ArrayList<>();
      this.consultas = new ArrayList<>();
   }

   // Setter para la persistencia (se inyectará desde Main o ViewModel)
   public void setPersistenciaService(IPersistencia persistenciaService) {
      this.persistenciaService = persistenciaService;
   }

   // Métodos para cargar datos iniciales (usando el servicio de persistencia)
   public void cargarDatosIniciales() throws Exception {
      if (persistenciaService != null) {
         this.pacientes = persistenciaService.cargarTodosLosPacientes();
         this.medicos = persistenciaService.cargarTodosLosMedicos();
         this.consultas = persistenciaService.cargarTodasLasConsultas();
         System.out.println("Datos cargados exitosamente desde Firebase.");
      } else {
         System.err.println("Servicio de persistencia no configurado.");
      }
   }

   // 1. Métodos para registrar nuevos pacientes y médicos [cite: 3]
   public void registrarPaciente(Paciente paciente) throws Exception {
      // Validación de campos vacíos (ejemplo, se mejorará con excepciones
personalizadas)
      if (paciente.getId().isEmpty() || paciente.getNombre().isEmpty() ||
paciente.getEmail().isEmpty()) {
         throw new IllegalArgumentException("Campos de paciente
incompletos.");
```

```
        }
        // Verificar si el paciente ya existe
        if (pacientes.stream().anyMatch(p -> p.getId().equals(paciente.getId()))) {
            throw new IllegalArgumentException("Paciente con ID " +
paciente.getId() + " ya existe.");
        }
        pacientes.add(paciente);
        if (persistenciaService != null) {
            persistenciaService.guardarPaciente(paciente);
        }
    }

    public void registrarMedico(Medico medico) throws Exception {
        // Validación de campos vacíos
        if (medico.getId().isEmpty() || medico.getNombre().isEmpty() ||
medico.getEspecialidad().isEmpty()) {
            throw new IllegalArgumentException("Campos de médico
incompletos.");
        }
        // Verificar si el médico ya existe
        if (medicos.stream().anyMatch(m -> m.getId().equals(medico.getId()))) {
            throw new IllegalArgumentException("Médico con ID " + medico.getId()
+ " ya existe.");
        }
        medicos.add(medico);
        if (persistenciaService != null) {
            persistenciaService.guardarMedico(medico);
        }
    }

    // 2. Método para asignar una consulta [cite: 4]
    public Consulta asignarConsulta(String idPaciente, String idMedico, String
sintomas, String diagnostico, String tratamiento) throws Exception {
        Paciente paciente = buscarPacientePorId(idPaciente);
        Medico medico = buscarMedicoPorId(idMedico);

        if (paciente == null) {
            throw new IllegalArgumentException("Paciente con ID " + idPaciente + "
no encontrado.");
        }
        if (medico == null) {
            throw new IllegalArgumentException("Médico con ID " + idMedico + " no
encontrado.");
        }
        if (sintomas.isEmpty() || diagnostico.isEmpty() || tratamiento.isEmpty()) {
            throw new IllegalArgumentException("Campos de la consulta
incompletos.");
        }
```

```
        String idConsulta = UUID.randomUUID().toString(); // Generar ID único para
la consulta
        String fechaActual = new java.text.SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new java.util.Date());
        Consulta nuevaConsulta = new Consulta(idConsulta, idPaciente, idMedico,
fechaActual, sintomas, diagnostico, tratamiento);
        consultas.add(nuevaConsulta);

        if (persistenciaService != null) {
            persistenciaService.guardarConsulta(nuevaConsulta);
        }
        return nuevaConsulta;
    }

    // 3. Método para consultar el historial médico de un paciente [cite: 5]
    public List<Consulta> consultarHistorialPaciente(String idPaciente) {
        List<Consulta> historial = new ArrayList<>();
        for (Consulta consulta : consultas) {
            if (consulta.getIdPaciente().equals(idPaciente)) {
                historial.add(consulta);
            }
        }
        return historial;
    }

    // 4. Método para listar todas las consultas realizadas por un médico [cite: 5]
    public List<Consulta> listarConsultasPorMedico(String idMedico) {
        List<Consulta> consultasMedico = new ArrayList<>();
        for (Consulta consulta : consultas) {
            if (consulta.getIdMedico().equals(idMedico)) {
                consultasMedico.add(consulta);
            }
        }
        return consultasMedico;
    }

    // Métodos de búsqueda auxiliares
    public Paciente buscarPacientePorId(String id) {
        return pacientes.stream()
                .filter(p -> p.getId().equals(id))
                .findFirst()
                .orElse(null);
    }

    public Medico buscarMedicoPorId(String id) {
        return medicos.stream()
                .filter(m -> m.getId().equals(id))
```

```
                .findFirst()
                .orElse(null);
        }

        public List<Paciente> getPacientes() {
            return new ArrayList<>(pacientes); // Devuelve una copia para evitar
    modificación externa directa
        }

        public List<Medico> getMedicos() {
            return new ArrayList<>(medicos); // Devuelve una copia
        }

        public List<Consulta> getConsultas() {
            return new ArrayList<>(consultas); // Devuelve una copia
        }
}
```

- IPersistencia.java (interfaz)

Define el contrato para los servicios de persistencia, permitiendo que la lógica de negocio sea independiente de la implementación de almacenamiento.

```
package com.simulador.model;

import java.util.List;

// Interfaz para la persistencia de datos
public interface IPersistencia {
    // Métodos para pacientes
    void guardarPaciente(Paciente paciente) throws Exception;
    Paciente cargarPaciente(String id) throws Exception;
    List<Paciente> cargarTodosLosPacientes() throws Exception;
    void eliminarPaciente(String id) throws Exception;

    // Métodos para médicos
    void guardarMedico(Medico medico) throws Exception;
    Medico cargarMedico(String id) throws Exception;
    List<Medico> cargarTodosLosMedicos() throws Exception;
    void eliminarMedico(String id) throws Exception;

    // Métodos para consultas
    void guardarConsulta(Consulta consulta) throws Exception;
    Consulta cargarConsulta(String id) throws Exception;
    List<Consulta> cargarTodasLasConsultas() throws Exception;
    void eliminarConsulta(String id) throws Exception;
```

```
}
```

### 3.2 Persistencia

Contiene la implementación concreta de la interfaz `IPersistencia`.

- PersistenciaFirebase.java
  Implementa `IPersistencia` usando Firebase Realtime Database. Incluye la inicialización de Firebase y métodos para guardar y cargar `Paciente`, `Medico` y `Consulta`.

```
package com.simulador.persistencia;

// Firebase
import com.google.auth.oauth2.GoogleCredentials;
import com.google.firebase.FirebaseApp;
import com.google.firebase.FirebaseOptions;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

// Paquete model
import com.simulador.model.Consulta;
import com.simulador.model.IPersistencia;
import com.simulador.model.Medico;
import com.simulador.model.Paciente;

// Paquete excepciones
import com.simulador.excepciones.UsuarioNoEncontradoException; // Se
usarán más adelante
import com.simulador.excepciones.CampoVacioException; // Se usarán más
adelante

import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.CountDownLatch; // Para esperar operaciones
asíncronas

public class PersistenciaFirebase implements IPersistencia {

   private DatabaseReference database;
   private static final String DATABASE_URL =
"https://simuladorgestionconsultas-default-rtdb.firebaseio.com/"; //
¡Reemplaza con la URL de tu base de datos Firebase!
```

```
    public PersistenciaFirebase() {
        try {
            // Cargar el archivo de credenciales de Firebase
            FileInputStream serviceAccount =
                    new
FileInputStream("C:\\Users\\pipe_\\OneDrive\\Documentos\\NetBeansProjects
\\SimuladorGestionConsultasMedicas\\SimuladorGestionConsultasMedicas\\sr
c\\simuladorgestionconsultas-firebase-adminsdk-fbsvc-59db2dce2f.json"); //
Asegúrate de que esta ruta sea correcta

            FirebaseOptions options = new FirebaseOptions.Builder()
                    .setCredentials(GoogleCredentials.fromStream(serviceAccount))
                    .setDatabaseUrl(DATABASE_URL)
                    .build();

            // Inicializar Firebase si aún no ha sido inicializado
            if (FirebaseApp.getApps().isEmpty()) {
                FirebaseApp.initializeApp(options);
            }

            database = FirebaseDatabase.getInstance().getReference();
            System.out.println("Conexión a Firebase establecida exitosamente.");

        } catch (IOException e) {
            System.err.println("ERROR: No se pudo cargar el archivo de credenciales
de Firebase: " + e.getMessage());
            e.printStackTrace();
        }
    }

    // Métodos para pacientes
    @Override
    public void guardarPaciente(Paciente paciente) throws Exception {
        if (paciente.getId() == null || paciente.getId().isEmpty()) {
            throw new CampoVacioException("El ID del paciente no puede estar
vacío para guardar en Firebase.");
        }
        DatabaseReference pacientesRef =
database.child("pacientes").child(paciente.getId());
        CountDownLatch latch = new CountDownLatch(1);
        pacientesRef.setValue(paciente, (databaseError, databaseReference) -> {
            if (databaseError != null) {
                System.err.println("Error al guardar paciente: " +
databaseError.getMessage());
            } else {
                System.out.println("Paciente " + paciente.getNombre() + " guardado
exitosamente en Firebase.");
```

```
            }
            latch.countDown();
        });
        latch.await(); // Espera a que la operación asíncrona termine
    }

    @Override
    public Paciente cargarPaciente(String id) throws Exception {
        CountDownLatch latch = new CountDownLatch(1);
        final Paciente[] pacienteHolder = {null};
        database.child("pacientes").child(id).addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                pacienteHolder[0] = dataSnapshot.getValue(Paciente.class);
                latch.countDown();
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                System.err.println("Error al cargar paciente: " +
databaseError.getMessage());
                latch.countDown();
            }
        });
        latch.await();
        if (pacienteHolder[0] == null) {
            throw new UsuarioNoEncontradoException("Paciente con ID " + id + " no
encontrado en Firebase.");
        }
        return pacienteHolder[0];
    }

    @Override
    public List<Paciente> cargarTodosLosPacientes() throws Exception {
        CountDownLatch latch = new CountDownLatch(1);
        List<Paciente> pacientes = new ArrayList<>();
        database.child("pacientes").addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                    Paciente paciente = snapshot.getValue(Paciente.class);
                    if (paciente != null) {
                        pacientes.add(paciente);
                    }
                }
                latch.countDown();
```

```
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                System.err.println("Error al cargar todos los pacientes: " +
databaseError.getMessage());
                latch.countDown();
            }
        });
        latch.await();
        return pacientes;
    }

    @Override
    public void eliminarPaciente(String id) throws Exception {
        CountDownLatch latch = new CountDownLatch(1);
        database.child("pacientes").child(id).removeValue((databaseError,
databaseReference) -> {
            if (databaseError != null) {
                System.err.println("Error al eliminar paciente: " +
databaseError.getMessage());
            } else {
                System.out.println("Paciente con ID " + id + " eliminado exitosamente
de Firebase.");
            }
            latch.countDown();
        });
        latch.await();
    }

    // Métodos para médicos (similar a pacientes)
    @Override
    public void guardarMedico(Medico medico) throws Exception {
        if (medico.getId() == null || medico.getId().isEmpty()) {
            throw new CampoVacioException("El ID del médico no puede estar vacío
para guardar en Firebase.");
        }
        DatabaseReference medicosRef =
database.child("medicos").child(medico.getId());
        CountDownLatch latch = new CountDownLatch(1);
        medicosRef.setValue(medico, (databaseError, databaseReference) -> {
            if (databaseError != null) {
                System.err.println("Error al guardar médico: " +
databaseError.getMessage());
            } else {
                System.out.println("Médico " + medico.getNombre() + " guardado
exitosamente en Firebase.");
            }
```

```
            latch.countDown();
        });
        latch.await();
    }

    @Override
    public Medico cargarMedico(String id) throws Exception {
        CountDownLatch latch = new CountDownLatch(1);
        final Medico[] medicoHolder = {null};
        database.child("medicos").child(id).addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                medicoHolder[0] = dataSnapshot.getValue(Medico.class);
                latch.countDown();
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                System.err.println("Error al cargar médico: " +
databaseError.getMessage());
                latch.countDown();
            }
        });
        latch.await();
        if (medicoHolder[0] == null) {
            throw new UsuarioNoEncontradoException("Médico con ID " + id + " no
encontrado en Firebase.");
        }
        return medicoHolder[0];
    }

    @Override
    public List<Medico> cargarTodosLosMedicos() throws Exception {
        CountDownLatch latch = new CountDownLatch(1);
        List<Medico> medicos = new ArrayList<>();
        database.child("medicos").addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                    Medico medico = snapshot.getValue(Medico.class);
                    if (medico != null) {
                        medicos.add(medico);
                    }
                }
                latch.countDown();
            }
```

```
            @Override
            public void onCancelled(DatabaseError databaseError) {
                System.err.println("Error al cargar todos los médicos: " +
databaseError.getMessage());
                latch.countDown();
            }
        });
        latch.await();
        return medicos;
    }

    @Override
    public void eliminarMedico(String id) throws Exception {
        CountDownLatch latch = new CountDownLatch(1);
        database.child("medicos").child(id).removeValue((databaseError,
databaseReference) -> {
            if (databaseError != null) {
                System.err.println("Error al eliminar médico: " +
databaseError.getMessage());
            } else {
                System.out.println("Médico con ID " + id + " eliminado exitosamente
de Firebase.");
            }
            latch.countDown();
        });
        latch.await();
    }

    // Métodos para consultas (similar a pacientes y médicos)
    @Override
    public void guardarConsulta(Consulta consulta) throws Exception {
        if (consulta.getIdConsulta() == null || consulta.getIdConsulta().isEmpty()) {
            throw new CampoVacioException("El ID de la consulta no puede estar
vacío para guardar en Firebase.");
        }
        DatabaseReference consultasRef =
database.child("consultas").child(consulta.getIdConsulta());
        CountDownLatch latch = new CountDownLatch(1);
        consultasRef.setValue(consulta, (databaseError, databaseReference) -> {
            if (databaseError != null) {
                System.err.println("Error al guardar consulta: " +
databaseError.getMessage());
            } else {
                System.out.println("Consulta " + consulta.getIdConsulta() + " guardada
exitosamente en Firebase.");
            }
            latch.countDown();
```

```
        });
        latch.await();
    }

    @Override
    public Consulta cargarConsulta(String id) throws Exception {
        CountDownLatch latch = new CountDownLatch(1);
        final Consulta[] consultaHolder = {null};
        database.child("consultas").child(id).addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                consultaHolder[0] = dataSnapshot.getValue(Consulta.class);
                latch.countDown();
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                System.err.println("Error al cargar consulta: " +
databaseError.getMessage());
                latch.countDown();
            }
        });
        latch.await();
        if (consultaHolder[0] == null) {
            throw new UsuarioNoEncontradoException("Consulta con ID " + id + " no
encontrada en Firebase.");
        }
        return consultaHolder[0];
    }

    @Override
    public List<Consulta> cargarTodasLasConsultas() throws Exception {
        CountDownLatch latch = new CountDownLatch(1);
        List<Consulta> consultas = new ArrayList<>();
        database.child("consultas").addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                    Consulta consulta = snapshot.getValue(Consulta.class);
                    if (consulta != null) {
                        consultas.add(consulta);
                    }
                }
                latch.countDown();
            }
```

```
                @Override
                public void onCancelled(DatabaseError databaseError) {
                    System.err.println("Error al cargar todas las consultas: " +
            databaseError.getMessage());
                    latch.countDown();
                }
            });
            latch.await();
            return consultas;
        }

        @Override
        public void eliminarConsulta(String id) throws Exception {
            CountDownLatch latch = new CountDownLatch(1);
            database.child("consultas").child(id).removeValue((databaseError,
            databaseReference) -> {
                if (databaseError != null) {
                    System.err.println("Error al eliminar consulta: " +
            databaseError.getMessage());
                } else {
                    System.out.println("Consulta con ID " + id + " eliminada exitosamente
            de Firebase.");
                }
                latch.countDown();
            });
            latch.await();
        }
}
```

**3.3 Excepciones**

Clases para el manejo de errores específicos, heredando de `Exception`.

- UsuarioNoEncontradoException.java

```
package com.simulador.excepciones;

public class UsuarioNoEncontradoException extends Exception {

    public UsuarioNoEncontradoException(String message) {
        super(message);
    }
}
```

- CampoVacioException.java

A

```
package com.simulador.excepciones;

public class CampoVacioException extends Exception {

   public CampoVacioException(String message) {
      super(message);
   }
}
```

### 3.4 ViewModel

- ClinicaViewModel
  Actúa como intermediario entre el modelo (`Clinica`) y las vistas. Expone datos y comandos a la vista, maneja lógica de negocio secundaria y delegua operaciones principales al modelo. También maneja y relanza excepciones a la vista. Es el corazón del patrón MVVM en este proyecto.

```
package com.simulador.viewmodel;

// Paquete model
import com.simulador.model.Clinica;
import com.simulador.model.Consulta;
import com.simulador.model.Medico;
import com.simulador.model.Paciente;
import com.simulador.model.IPersistencia;

// Paquete persistencia
import com.simulador.persistencia.PersistenciaFirebase; // Importamos nuestra
implementación de persistencia

// Paquete excepciones
import com.simulador.excepciones.UsuarioNoEncontradoException;
import com.simulador.excepciones.CampoVacioException;

import java.util.List;

import java.util.ArrayList; // Necesario para la lista mutable que devuelve el
ViewModel

public class ClinicaViewModel {

   private Clinica clinicaModel; // La instancia de nuestro modelo de negocio

   // Constructor que recibe el servicio de persistencia
   public ClinicaViewModel() {
      this.clinicaModel = new Clinica();
      // Inyectamos la dependencia de persistencia al modelo
      this.clinicaModel.setPersistenciaService(new PersistenciaFirebase());
```

```
        try {
            // Cargamos los datos al iniciar el ViewModel
            this.clinicaModel.cargarDatosIniciales();
        } catch (Exception e) {
            System.err.println("Error al cargar datos iniciales en ViewModel: " +
e.getMessage());
            // En una aplicación real, esto se mostraría en la UI
        }
    }

    // --- Métodos para el Login de usuario (Requisito funcional 1) ---
    // Aunque no tenemos un sistema de autenticación completo, podemos
simular
    // un "login" verificando si el ID existe en pacientes o médicos.
    // Esto se podría expandir a tener roles (médico/administrativo).
    // --- Métodos para el Login de usuario (Requisito funcional 1) ---
    // Ahora, este método solo permitirá el "login" a Médicos (considerados
administrativos)
    // Los pacientes serán rechazados si intentan usar esta función de login.
    public String login(String idUsuario) throws UsuarioNoEncontradoException,
CampoVacioException {
        try {
            if (idUsuario == null || idUsuario.trim().isEmpty()) {
                throw new CampoVacioException("El ID de usuario no puede estar
vacío.");
            }

            // Primero, buscar si es un médico
            Medico medico = clinicaModel.buscarMedicoPorId(idUsuario);
            if (medico != null) {
                // Si es un médico, se le permite el acceso y se considera un rol
"administrativo"
                return "Medico";
            }

            // Segundo, buscar si es un paciente. Si es un paciente, no se le permite
acceder a esta interfaz.
            Paciente paciente = clinicaModel.buscarPacientePorId(idUsuario);
            if (paciente != null) {
                // Si el ID corresponde a un paciente, lanzamos una excepción
específica
                throw new UsuarioNoEncontradoException("Los pacientes no tienen
acceso a esta interfaz. Por favor, inicie sesión con un ID de
médico/administrador.");
            }

            // Si no se encuentra ni como médico ni como paciente
```

```
            throw new UsuarioNoEncontradoException("Usuario con ID " + idUsuario
+ " no encontrado o no autorizado.");

        } catch (CampoVacioException e) {
            throw e; // Relanzar CampoVacioException
        } catch (UsuarioNoEncontradoException e) {
            throw e; // Relanzar la excepción de usuario no encontrado/no
autorizado
        } catch (Exception e) {
            // Capturar otras excepciones inesperadas
            throw new UsuarioNoEncontradoException("Error inesperado al intentar
iniciar sesión: " + e.getMessage());
        }
    }

    // --- Métodos para Registrar nuevos pacientes y médicos (Requisito funcional
2) ---
    public void registrarPaciente(String id, String nombre, String telefono, String
email, String fechaNacimiento, String historialMedicoGeneral) throws
CampoVacioException, Exception {
        if (id.trim().isEmpty() || nombre.trim().isEmpty() ||
telefono.trim().isEmpty() || email.trim().isEmpty() ||
fechaNacimiento.trim().isEmpty()) {
            throw new CampoVacioException("Todos los campos obligatorios del
paciente deben ser llenados.");
        }
        Paciente nuevoPaciente = new Paciente(id, nombre, telefono, email,
fechaNacimiento, historialMedicoGeneral);
        try {
            clinicaModel.registrarPaciente(nuevoPaciente);
        } catch (IllegalArgumentException e) {
            // Capturar la excepción del modelo y relanzar con un mensaje más
específico
            throw new Exception("Error al registrar paciente: " + e.getMessage());
        }
    }

    public void registrarMedico(String id, String nombre, String telefono, String
email, String especialidad, String licenciaMedica) throws CampoVacioException,
Exception {
        if (id.trim().isEmpty() || nombre.trim().isEmpty() ||
telefono.trim().isEmpty() || email.trim().isEmpty() ||
especialidad.trim().isEmpty() || licenciaMedica.trim().isEmpty()) {
            throw new CampoVacioException("Todos los campos obligatorios del
médico deben ser llenados.");
        }
        Medico nuevoMedico = new Medico(id, nombre, telefono, email,
especialidad, licenciaMedica);
```

```
        try {
            clinicaModel.registrarMedico(nuevoMedico);
        } catch (IllegalArgumentException e) {
            throw new Exception("Error al registrar médico: " + e.getMessage());
        }
    }

    // --- Método para asignar una consulta (Requisito funcional 3) ---
    public Consulta asignarConsulta(String idPaciente, String idMedico, String
sintomas, String diagnostico, String tratamiento) throws CampoVacioException,
UsuarioNoEncontradoException, Exception {
        if (idPaciente.trim().isEmpty() || idMedico.trim().isEmpty() ||
sintomas.trim().isEmpty() || diagnostico.trim().isEmpty() ||
tratamiento.trim().isEmpty()) {
            throw new CampoVacioException("Todos los campos de la consulta
deben ser llenados.");
        }
        try {
            return clinicaModel.asignarConsulta(idPaciente, idMedico, sintomas,
diagnostico, tratamiento);
        } catch (IllegalArgumentException e) {
            // Podríamos distinguir si el error es por paciente/médico no encontrado
o campos incompletos
            if (e.getMessage().contains("Paciente con ID") ||
e.getMessage().contains("Médico con ID")) {
                throw new UsuarioNoEncontradoException(e.getMessage());
            } else {
                throw new Exception("Error al asignar consulta: " + e.getMessage());
            }
        }
    }

    // --- Método para consultar el historial médico de un paciente (Requisito
funcional 4) ---
    public List<Consulta> consultarHistorialPaciente(String idPaciente) throws
UsuarioNoEncontradoException, CampoVacioException {
        if (idPaciente.trim().isEmpty()) {
            throw new CampoVacioException("El ID del paciente no puede estar
vacío para consultar su historial.");
        }
        if (clinicaModel.buscarPacientePorId(idPaciente) == null) {
            throw new UsuarioNoEncontradoException("Paciente con ID " +
idPaciente + " no encontrado.");
        }
        return new
ArrayList<>(clinicaModel.consultarHistorialPaciente(idPaciente)); // Devuelve
una copia
    }
```

```
    // --- Método para listar todas las consultas realizadas por un médico
(Requisito funcional 5) ---
    public List<Consulta> listarConsultasPorMedico(String idMedico) throws
UsuarioNoEncontradoException, CampoVacioException {
        if (idMedico.trim().isEmpty()) {
            throw new CampoVacioException("El ID del médico no puede estar vacío
para listar sus consultas.");
        }
        if (clinicaModel.buscarMedicoPorId(idMedico) == null) {
            throw new UsuarioNoEncontradoException("Médico con ID " + idMedico
+ " no encontrado.");
        }
        return new ArrayList<>(clinicaModel.listarConsultasPorMedico(idMedico));
// Devuelve una copia
    }

    // Métodos para obtener listas de objetos para la UI (ej. para ComboBoxes o
tablas)
    public List<Paciente> getListaPacientes() {
        return new ArrayList<>(clinicaModel.getPacientes()); // Devuelve una copia
    }

    public List<Medico> getListaMedicos() {
        return new ArrayList<>(clinicaModel.getMedicos()); // Devuelve una copia
    }

    public List<Consulta> getListaConsultas() {
        return new ArrayList<>(clinicaModel.getConsultas()); // Devuelve una copia
    }

    // Métodos para buscar Paciente/Medico por ID (útil para validaciones en la
UI o para obtener detalles)
    public Paciente buscarPacientePorId(String id) {
        return clinicaModel.buscarPacientePorId(id);
    }

    public Medico buscarMedicoPorId(String id) {
        return clinicaModel.buscarMedicoPorId(id);
    }
}
```

**3.5 Vista**

La capa de la vista implementa la interfaz gráfica de usuario con Java Swing.

- VentanaPrincipal.java

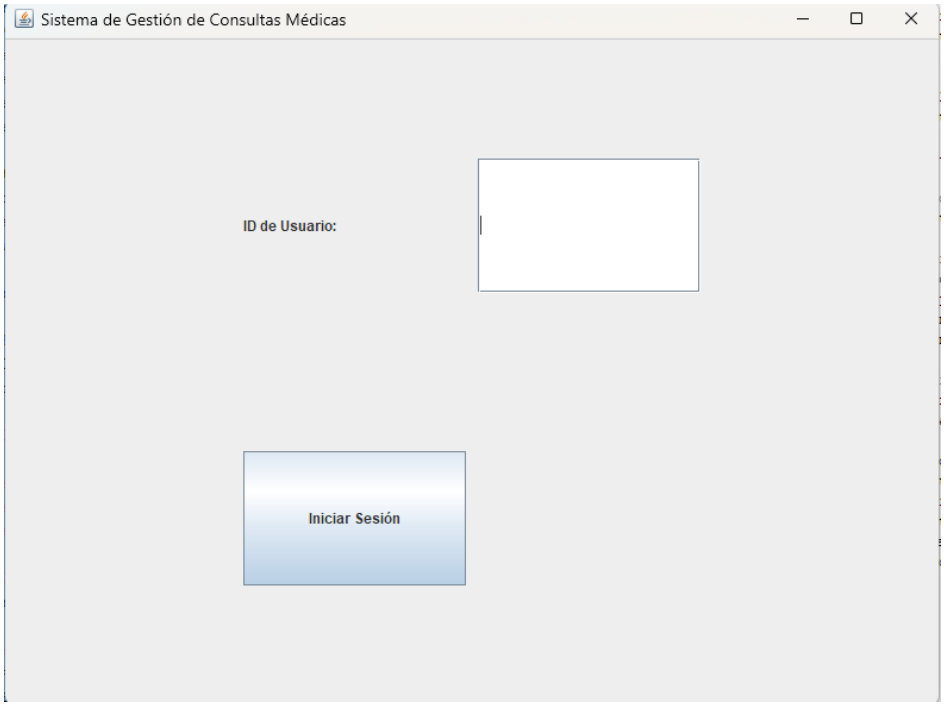
Es la ventana principal de la aplicación. Contiene la pantalla de login inicial y, tras un login exitoso de un médico/administrador, habilita un menú para navegar entre las funcionalidades.

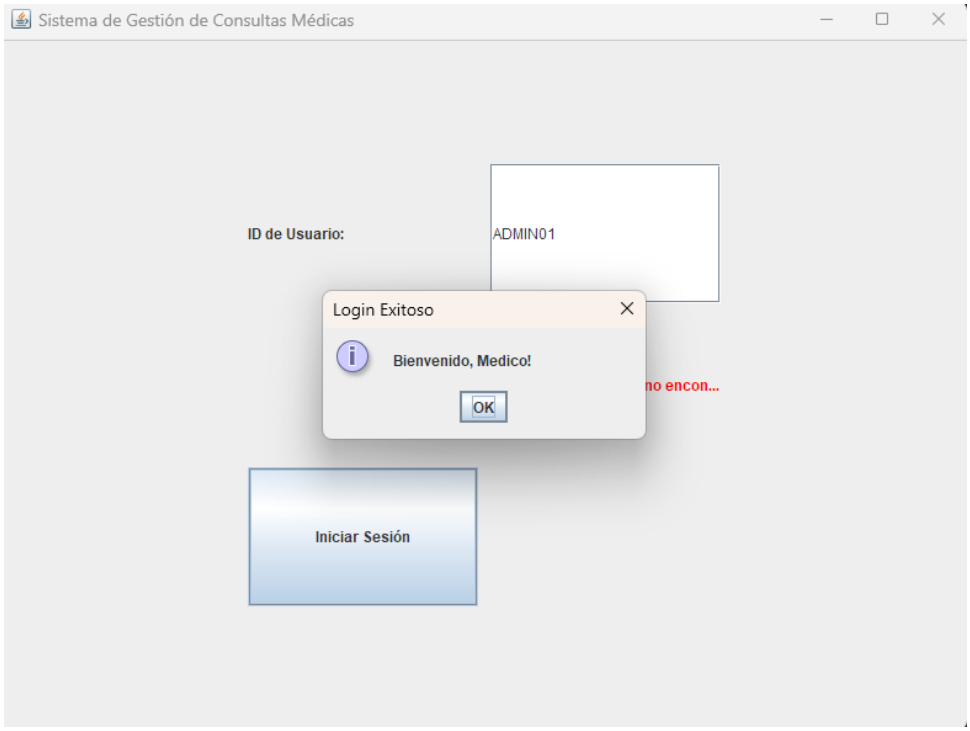
```
package com.simulador.view;

import com.simulador.viewmodel.ClinicaViewModel; // Importamos el
ViewModel

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import com.simulador.excepciones.UsuarioNoEncontradoException; // Para el
login
import com.simulador.excepciones.CampoVacioException; // Para el login

public class VentanaPrincipal extends JFrame {
   private ClinicaViewModel viewModel; // Instancia del ViewModel
   private JPanel contentPanel; // Panel para cambiar el contenido de la ventana
   private JMenuBar menuBar;

   public VentanaPrincipal() {
      setTitle("Sistema de Gestión de Consultas Médicas");
      setSize(800, 600);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLocationRelativeTo(null); // Centrar la ventana en la pantalla

      viewModel = new ClinicaViewModel(); // Inicializamos el ViewModel

      contentPanel = new JPanel();
      contentPanel.setLayout(new BorderLayout());
      add(contentPanel, BorderLayout.CENTER);

      // Pantalla de Login inicial
      showLoginPanel();

      // No mostraremos el menú hasta que el usuario inicie sesión
      menuBar = new JMenuBar();
      setJMenuBar(menuBar); // Se establecerá visible después del login exitoso
   }

   private void showLoginPanel() {
      JPanel loginPanel = new JPanel(new GridLayout(3, 2, 10, 10));
      loginPanel.setBorder(BorderFactory.createEmptyBorder(100, 200, 100,
200));

      JLabel idLabel = new JLabel("ID de Usuario:");
```

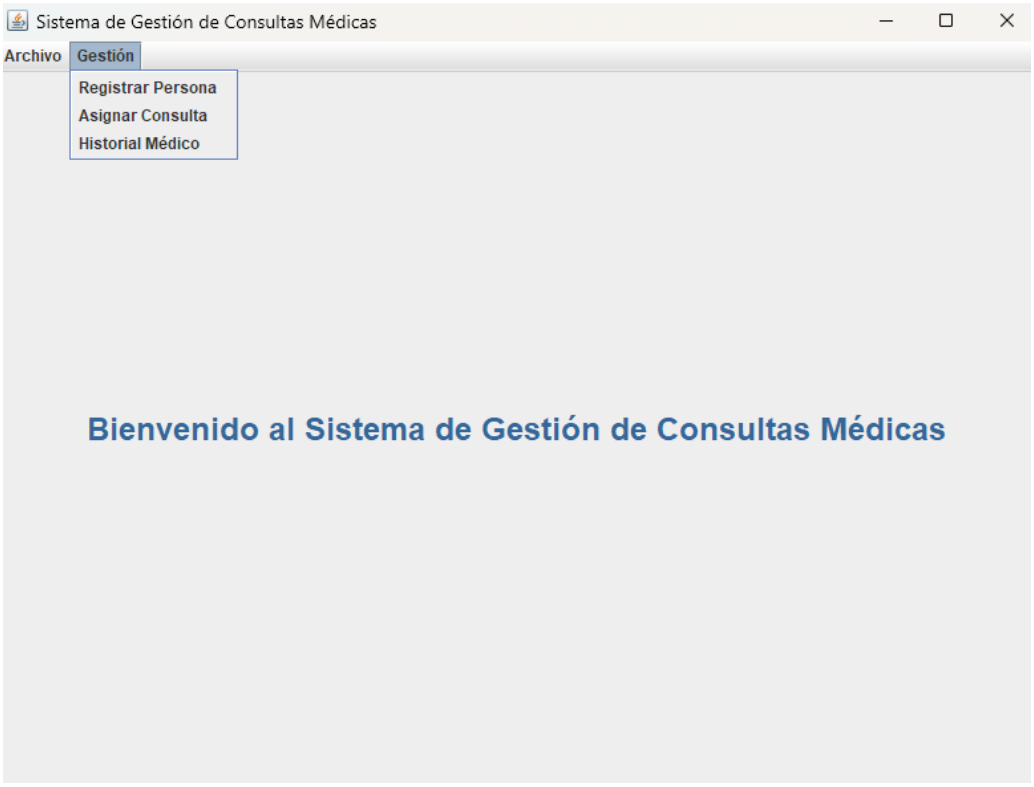
```
        JTextField idField = new JTextField();
        JButton loginButton = new JButton("Iniciar Sesión");
        JLabel messageLabel = new JLabel("");
        messageLabel.setForeground(Color.RED);

        loginPanel.add(idLabel);
        loginPanel.add(idField);
        loginPanel.add(new JLabel("")); // Espacio
        loginPanel.add(messageLabel);
        loginPanel.add(loginButton);


        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String id = idField.getText();
                try {
                    String tipoUsuario = viewModel.login(id);
                    JOptionPane.showMessageDialog(VentanaPrincipal.this,
"Bienvenido, " + tipoUsuario + "!", "Login Exitoso",
JOptionPane.INFORMATION_MESSAGE);
                    setupMenu(tipoUsuario); // Configura el menú según el tipo de
usuario
                    contentPanel.removeAll(); // Limpia el panel de contenido
                    contentPanel.revalidate();
                    contentPanel.repaint();
                    // Opcional: Mostrar un panel por defecto después del login (ej. un
panel de bienvenida)
                    showDefaultPanel();
                } catch (CampoVacioException ex) {
                    messageLabel.setText(ex.getMessage());
                } catch (UsuarioNoEncontradoException ex) {
                    messageLabel.setText(ex.getMessage());
                } catch (Exception ex) {
                    messageLabel.setText("Error inesperado: " + ex.getMessage());
                    ex.printStackTrace();
                }
            }
        });

        contentPanel.removeAll();
        contentPanel.add(loginPanel, BorderLayout.CENTER);
        contentPanel.revalidate();
        contentPanel.repaint();
    }

    private void setupMenu(String tipoUsuario) {
        // Habilitar la barra de menú una vez que el usuario ha iniciado sesión
```

```
        setJMenuBar(menuBar);

        JMenu archivoMenu = new JMenu("Archivo");
        JMenuItem salirItem = new JMenuItem("Salir");
        salirItem.addActionListener(e -> System.exit(0));
        archivoMenu.add(salirItem);
        menuBar.add(archivoMenu);

        JMenu gestionMenu = new JMenu("Gestión");
        JMenuItem registrarItem = new JMenuItem("Registrar Persona");
        JMenuItem consultaItem = new JMenuItem("Asignar Consulta");
        JMenuItem historialItem = new JMenuItem("Historial Médico");

        registrarItem.addActionListener(e -> showPanel(new
PanelRegistro(viewModel)));
        consultaItem.addActionListener(e -> showPanel(new
PanelConsulta(viewModel)));
        historialItem.addActionListener(e -> showPanel(new
PanelHistorial(viewModel)));

        gestionMenu.add(registrarItem);
        gestionMenu.add(consultaItem);
        gestionMenu.add(historialItem);
        menuBar.add(gestionMenu);

        // Dependiendo del tipo de usuario, podríamos ocultar/mostrar elementos
del menú
        // Por ahora, todos ven todo, pero es un punto de extensión.
    }

    private void showDefaultPanel() {
        JPanel welcomePanel = new JPanel();
        welcomePanel.setLayout(new GridBagLayout()); // Usar GridBagLayout
para centrar
        JLabel welcomeLabel = new JLabel("<html><h1 style='color:
#336699;'>Bienvenido al Sistema de Gestión de Consultas
Médicas</h1></html>");
        welcomeLabel.setFont(new Font("Arial", Font.BOLD, 24));
        welcomeLabel.setHorizontalAlignment(SwingConstants.CENTER);
        welcomePanel.add(welcomeLabel);
        contentPanel.removeAll();
        contentPanel.add(welcomePanel, BorderLayout.CENTER);
        contentPanel.revalidate();
        contentPanel.repaint();
    }

    // Método para cambiar el panel visible en la ventana principal
    private void showPanel(JPanel panel) {
```

```
                contentPanel.removeAll();
                contentPanel.add(panel, BorderLayout.CENTER);
                contentPanel.revalidate();
                contentPanel.repaint();
            }
}
```






- PanelRegistro.java

  Permite registrar nuevos pacientes y médicos a través de dos pestañas separadas.

```
package com.simulador.view;

import com.simulador.viewmodel.ClinicaViewModel;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class PanelRegistro extends JPanel {
    private ClinicaViewModel viewModel;

    // Componentes para el registro de Pacientes
    private JTextField txtIdPaciente, txtNombrePaciente, txtTelefonoPaciente,
txtEmailPaciente, txtFechaNacimiento, txtHistorialMedico;
    private JButton btnRegistrarPaciente;

    // Componentes para el registro de Médicos
    private JTextField txtIdMedico, txtNombreMedico, txtTelefonoMedico,
txtEmailMedico, txtEspecialidad, txtLicenciaMedica;
```

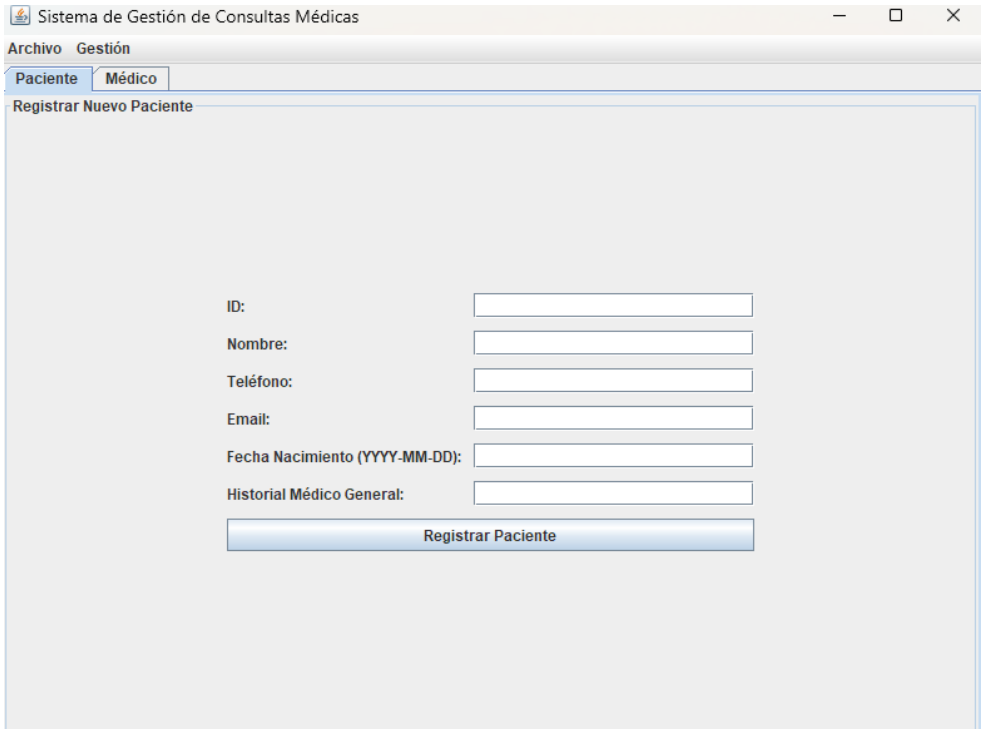
```
    private JButton btnRegistrarMedico;

    public PanelRegistro(ClinicaViewModel viewModel) {
        this.viewModel = viewModel;
        setLayout(new BorderLayout(10, 10)); // Layout principal del panel

        // Crear pestañas para Pacientes y Médicos
        JTabbedPane tabbedPane = new JTabbedPane();

        // --- Panel de Registro de Pacientes ---
        JPanel panelPaciente = new JPanel(new GridBagLayout());
        panelPaciente.setBorder(BorderFactory.createTitledBorder("Registrar
Nuevo Paciente"));
        setupPacientePanel(panelPaciente);
        tabbedPane.addTab("Paciente", panelPaciente);

        // --- Panel de Registro de Médicos ---
        JPanel panelMedico = new JPanel(new GridBagLayout());
        panelMedico.setBorder(BorderFactory.createTitledBorder("Registrar
Nuevo Médico"));
        setupMedicoPanel(panelMedico);
        tabbedPane.addTab("Médico", panelMedico);

        add(tabbedPane, BorderLayout.CENTER);
    }

    private void setupPacientePanel(JPanel panel) {
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(5, 5, 5, 5); // Márgenes internos
        gbc.fill = GridBagConstraints.HORIZONTAL;

        // Fila 1: ID
        gbc.gridx = 0;
        gbc.gridy = 0;
        panel.add(new JLabel("ID:"), gbc);
        gbc.gridx = 1;
        txtIdPaciente = new JTextField(20);
        panel.add(txtIdPaciente, gbc);

        // Fila 2: Nombre
        gbc.gridx = 0;
        gbc.gridy = 1;
        panel.add(new JLabel("Nombre:"), gbc);
        gbc.gridx = 1;
        txtNombrePaciente = new JTextField(20);
        panel.add(txtNombrePaciente, gbc);

        // Fila 3: Teléfono
```

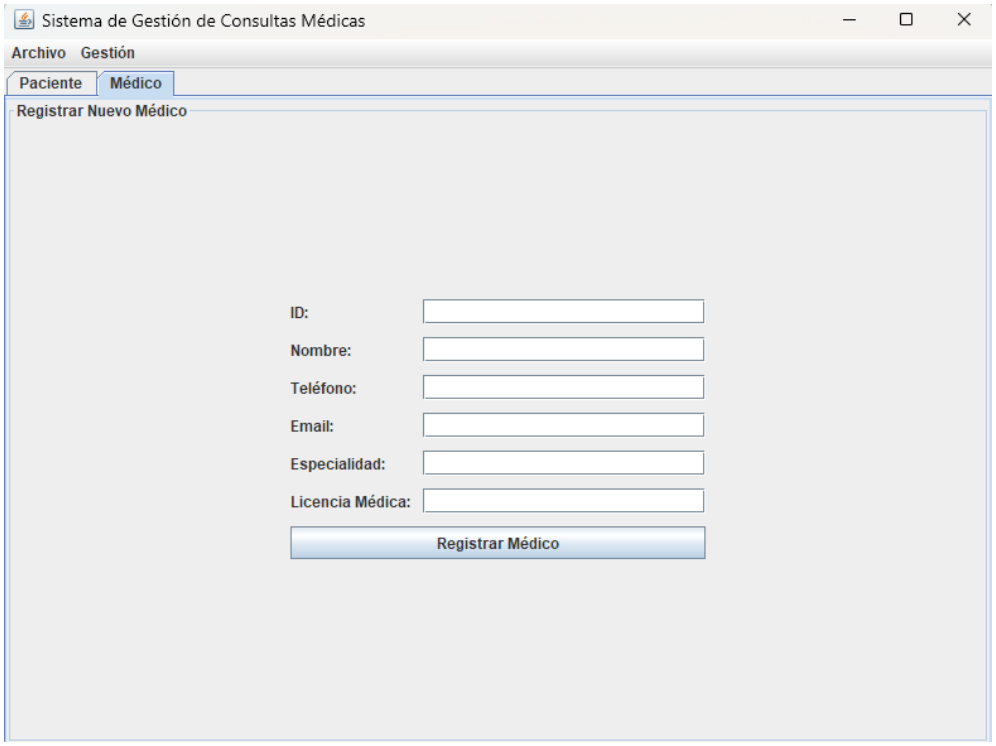
```
        gbc.gridx = 0;
        gbc.gridy = 2;
        panel.add(new JLabel("Teléfono:"), gbc);
        gbc.gridx = 1;
        txtTelefonoPaciente = new JTextField(20);
        panel.add(txtTelefonoPaciente, gbc);

        // Fila 4: Email
        gbc.gridx = 0;
        gbc.gridy = 3;
        panel.add(new JLabel("Email:"), gbc);
        gbc.gridx = 1;
        txtEmailPaciente = new JTextField(20);
        panel.add(txtEmailPaciente, gbc);

        // Fila 5: Fecha Nacimiento
        gbc.gridx = 0;
        gbc.gridy = 4;
        panel.add(new JLabel("Fecha Nacimiento (YYYY-MM-DD):"), gbc);
        gbc.gridx = 1;
        txtFechaNacimiento = new JTextField(20);
        panel.add(txtFechaNacimiento, gbc);

        // Fila 6: Historial Médico General
        gbc.gridx = 0;
        gbc.gridy = 5;
        panel.add(new JLabel("Historial Médico General:"), gbc);
        gbc.gridx = 1;
        txtHistorialMedico = new JTextField(20);
        panel.add(txtHistorialMedico, gbc);

        // Fila 7: Botón Registrar
        gbc.gridx = 0;
        gbc.gridy = 6;
        gbc.gridwidth = 2; // Ocupa dos columnas
        btnRegistrarPaciente = new JButton("Registrar Paciente");
        panel.add(btnRegistrarPaciente, gbc);

        btnRegistrarPaciente.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                registrarPaciente();
            }
        });
    }

    private void setupMedicoPanel(JPanel panel) {
        GridBagConstraints gbc = new GridBagConstraints();
```

```
gbc.insets = new Insets(5, 5, 5, 5);
gbc.fill = GridBagConstraints.HORIZONTAL;

// Fila 1: ID
gbc.gridx = 0;
gbc.gridy = 0;
panel.add(new JLabel("ID:"), gbc);
gbc.gridx = 1;
txtIdMedico = new JTextField(20);
panel.add(txtIdMedico, gbc);

// Fila 2: Nombre
gbc.gridx = 0;
gbc.gridy = 1;
panel.add(new JLabel("Nombre:"), gbc);
gbc.gridx = 1;
txtNombreMedico = new JTextField(20);
panel.add(txtNombreMedico, gbc);

// Fila 3: Teléfono
gbc.gridx = 0;
gbc.gridy = 2;
panel.add(new JLabel("Teléfono:"), gbc);
gbc.gridx = 1;
txtTelefonoMedico = new JTextField(20);
panel.add(txtTelefonoMedico, gbc);

// Fila 4: Email
gbc.gridx = 0;
gbc.gridy = 3;
panel.add(new JLabel("Email:"), gbc);
gbc.gridx = 1;
txtEmailMedico = new JTextField(20);
panel.add(txtEmailMedico, gbc);

// Fila 5: Especialidad
gbc.gridx = 0;
gbc.gridy = 4;
panel.add(new JLabel("Especialidad:"), gbc);
gbc.gridx = 1;
txtEspecialidad = new JTextField(20);
panel.add(txtEspecialidad, gbc);

// Fila 6: Licencia Médica
gbc.gridx = 0;
gbc.gridy = 5;
panel.add(new JLabel("Licencia Médica:"), gbc);
gbc.gridx = 1;
```

```
        txtLicenciaMedica = new JTextField(20);
        panel.add(txtLicenciaMedica, gbc);

        // Fila 7: Botón Registrar
        gbc.gridx = 0;
        gbc.gridy = 6;
        gbc.gridwidth = 2;
        btnRegistrarMedico = new JButton("Registrar Médico");
        panel.add(btnRegistrarMedico, gbc);

        btnRegistrarMedico.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                registrarMedico();
            }
        });
    }

    private void registrarPaciente() {
        String id = txtIdPaciente.getText();
        String nombre = txtNombrePaciente.getText();
        String telefono = txtTelefonoPaciente.getText();
        String email = txtEmailPaciente.getText();
        String fechaNacimiento = txtFechaNacimiento.getText();
        String historialMedicoGeneral = txtHistorialMedico.getText();

        try {
            viewModel.registrarPaciente(id, nombre, telefono, email,
fechaNacimiento, historialMedicoGeneral);
            JOptionPane.showMessageDialog(this, "Paciente " + nombre + "
registrado exitosamente.", "Registro Exitoso",
JOptionPane.INFORMATION_MESSAGE);
            clearPacienteFields();
        } catch (com.simulador.excepciones.CampoVacioException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error de
Validación", JOptionPane.WARNING_MESSAGE);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error al registrar paciente: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
        }
    }

    private void registrarMedico() {
        String id = txtIdMedico.getText();
        String nombre = txtNombreMedico.getText();
        String telefono = txtTelefonoMedico.getText();
        String email = txtEmailMedico.getText();
```

```
            String especialidad = txtEspecialidad.getText();
            String licenciaMedica = txtLicenciaMedica.getText();

            try {
                viewModel.registrarMedico(id, nombre, telefono, email, especialidad,
        licenciaMedica);
                JOptionPane.showMessageDialog(this, "Médico " + nombre + "
        registrado exitosamente.", "Registro Exitoso",
        JOptionPane.INFORMATION_MESSAGE);
                clearMedicoFields();
            } catch (com.simulador.excepciones.CampoVacioException e) {
                JOptionPane.showMessageDialog(this, e.getMessage(), "Error de
        Validación", JOptionPane.WARNING_MESSAGE);
            } catch (Exception e) {
                JOptionPane.showMessageDialog(this, "Error al registrar médico: " +
        e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                e.printStackTrace();
            }
        }

        private void clearPacienteFields() {
            txtIdPaciente.setText("");
            txtNombrePaciente.setText("");
            txtTelefonoPaciente.setText("");
            txtEmailPaciente.setText("");
            txtFechaNacimiento.setText("");
            txtHistorialMedico.setText("");
        }

        private void clearMedicoFields() {
            txtIdMedico.setText("");
            txtNombreMedico.setText("");
            txtTelefonoMedico.setText("");
            txtEmailMedico.setText("");
            txtEspecialidad.setText("");
            txtLicenciaMedica.setText("");
        }
}
```



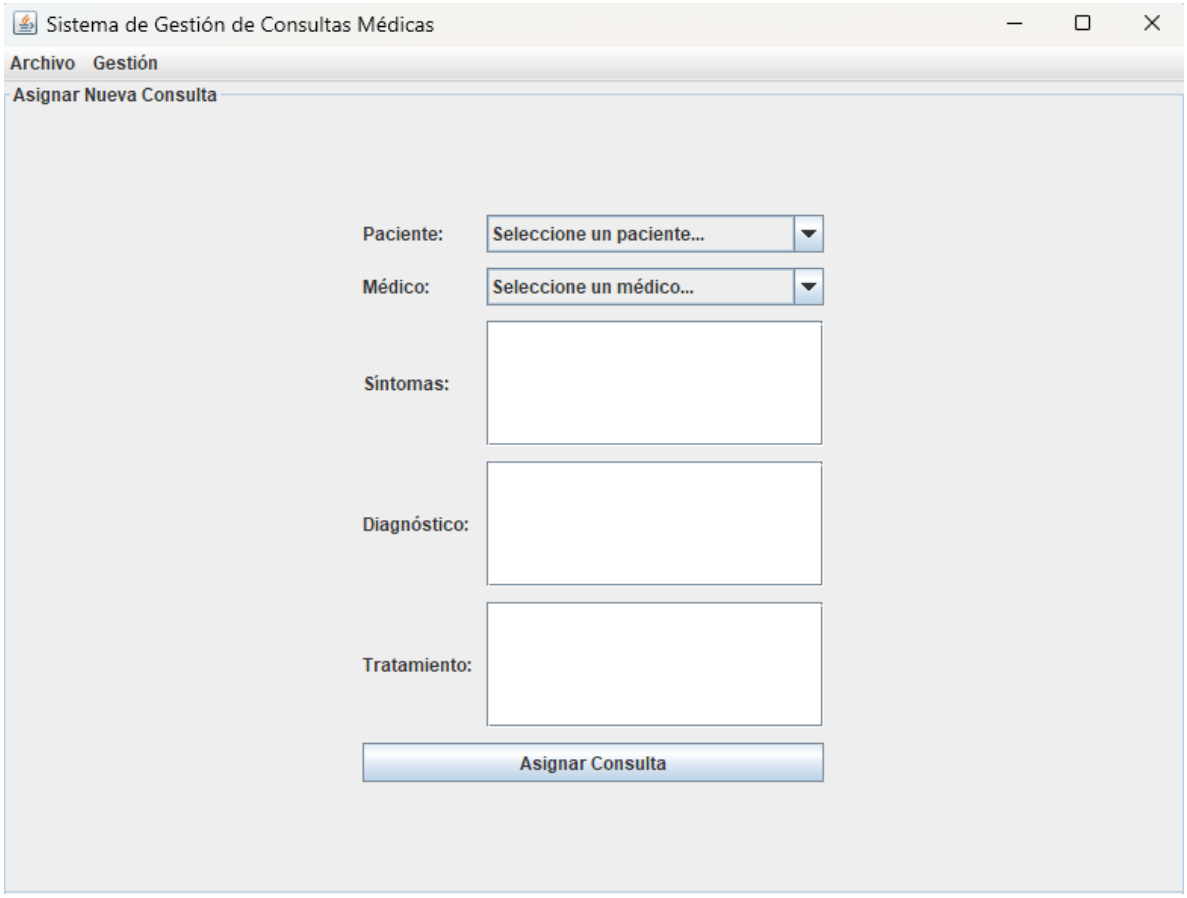

- PanelConsulta.java
  Permite asignar una consulta a un paciente con un médico existente, registrando síntomas, diagnóstico y tratamiento.

```
package com.simulador.view;

import com.simulador.model.Consulta; // Para el objeto Consulta
import com.simulador.model.Paciente; // Para cargar los pacientes en el
ComboBox
import com.simulador.model.Medico;   // Para cargar los médicos en el
ComboBox
import com.simulador.viewmodel.ClinicaViewModel;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import com.simulador.excepciones.CampoVacioException;
import com.simulador.excepciones.UsuarioNoEncontradoException;

public class PanelConsulta extends JPanel {
   private ClinicaViewModel viewModel;

   private JComboBox<String> cmbPacientes;
   private JComboBox<String> cmbMedicos;
   private JTextArea txtAreaSintomas;
   private JTextArea txtAreaDiagnostico;
   private JTextArea txtAreaTratamiento;
   private JButton btnAsignarConsulta;

   public PanelConsulta(ClinicaViewModel viewModel) {
      this.viewModel = viewModel;
      setLayout(new BorderLayout(10, 10));
      setBorder(BorderFactory.createTitledBorder("Asignar Nueva Consulta"));

      JPanel formPanel = new JPanel(new GridBagLayout());
      GridBagConstraints gbc = new GridBagConstraints();
      gbc.insets = new Insets(5, 5, 5, 5);
      gbc.fill = GridBagConstraints.HORIZONTAL;

      // Paciente
      gbc.gridx = 0;
      gbc.gridy = 0;
      formPanel.add(new JLabel("Paciente:"), gbc);
      gbc.gridx = 1;
      cmbPacientes = new JComboBox<>();
      formPanel.add(cmbPacientes, gbc);

      // Medico
      gbc.gridx = 0;
```

```
        gbc.gridy = 1;
        formPanel.add(new JLabel("Médico:"), gbc);
        gbc.gridx = 1;
        cmbMedicos = new JComboBox<>();
        formPanel.add(cmbMedicos, gbc);

        // Síntomas
        gbc.gridx = 0;
        gbc.gridy = 2;
        formPanel.add(new JLabel("Síntomas:"), gbc);
        gbc.gridx = 1;
        txtAreaSintomas = new JTextArea(5, 20);
        JScrollPane scrollSintomas = new JScrollPane(txtAreaSintomas);
        formPanel.add(scrollSintomas, gbc);

        // Diagnóstico
        gbc.gridx = 0;
        gbc.gridy = 3;
        formPanel.add(new JLabel("Diagnóstico:"), gbc);
        gbc.gridx = 1;
        txtAreaDiagnostico = new JTextArea(5, 20);
        JScrollPane scrollDiagnostico = new JScrollPane(txtAreaDiagnostico);
        formPanel.add(scrollDiagnostico, gbc);

        // Tratamiento
        gbc.gridx = 0;
        gbc.gridy = 4;
        formPanel.add(new JLabel("Tratamiento:"), gbc);
        gbc.gridx = 1;
        txtAreaTratamiento = new JTextArea(5, 20);
        JScrollPane scrollTratamiento = new JScrollPane(txtAreaTratamiento);
        formPanel.add(scrollTratamiento, gbc);

        // Botón
        gbc.gridx = 0;
        gbc.gridy = 5;
        gbc.gridwidth = 2;
        btnAsignarConsulta = new JButton("Asignar Consulta");
        formPanel.add(btnAsignarConsulta, gbc);

        add(formPanel, BorderLayout.CENTER);

        // Cargar datos en los ComboBoxes al inicializar el panel
        loadComboBoxData();

        btnAsignarConsulta.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
```

```
                asignarConsulta();
            }
        });
    }

    private void loadComboBoxData() {
        // Cargar Pacientes
        cmbPacientes.removeAllItems();
        cmbPacientes.addItem("Seleccione un paciente..."); // Opción por defecto
        List<Paciente> pacientes = viewModel.getListaPacientes();
        if (pacientes != null) {
            for (Paciente p : pacientes) {
                cmbPacientes.addItem(p.getId() + " - " + p.getNombre());
            }
        }

        // Cargar Médicos
        cmbMedicos.removeAllItems();
        cmbMedicos.addItem("Seleccione un médico..."); // Opción por defecto
        List<Medico> medicos = viewModel.getListaMedicos();
        if (medicos != null) {
            for (Medico m : medicos) {
                cmbMedicos.addItem(m.getId() + " - " + m.getNombre());
            }
        }
    }

    private void asignarConsulta() {
        String pacienteSeleccionado = (String) cmbPacientes.getSelectedItem();
        String medicoSeleccionado = (String) cmbMedicos.getSelectedItem();
        String sintomas = txtAreaSintomas.getText();
        String diagnostico = txtAreaDiagnostico.getText();
        String tratamiento = txtAreaTratamiento.getText();

        if (pacienteSeleccionado == null ||
pacienteSeleccionado.contains("Seleccione")) {
            JOptionPane.showMessageDialog(this, "Debe seleccionar un paciente.",
"Error de Selección", JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (medicoSeleccionado == null ||
medicoSeleccionado.contains("Seleccione")) {
            JOptionPane.showMessageDialog(this, "Debe seleccionar un médico.",
"Error de Selección", JOptionPane.WARNING_MESSAGE);
            return;
        }

        // Extraer solo el ID de las cadenas seleccionadas
```

```
            String idPaciente = pacienteSeleccionado.split(" - ")[0];
            String idMedico = medicoSeleccionado.split(" - ")[0];

            try {
                Consulta nuevaConsulta = viewModel.asignarConsulta(idPaciente,
        idMedico, sintomas, diagnostico, tratamiento);
                JOptionPane.showMessageDialog(this, "Consulta asignada exitosamente.
        ID: " + nuevaConsulta.getIdConsulta(), "Éxito",
        JOptionPane.INFORMATION_MESSAGE);
                clearFields();
                // Recargar datos en caso de que necesitemos actualizar algo (aunque
        para asignación no es crítico aquí)
                // loadComboBoxData(); // Si las listas cambian o la consulta influye
            } catch (CampoVacioException e) {
                JOptionPane.showMessageDialog(this, e.getMessage(), "Error de
        Validación", JOptionPane.WARNING_MESSAGE);
            } catch (UsuarioNoEncontradoException e) {
                JOptionPane.showMessageDialog(this, e.getMessage(), "Error",
        JOptionPane.ERROR_MESSAGE);
            } catch (Exception e) {
                JOptionPane.showMessageDialog(this, "Error al asignar consulta: " +
        e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                e.printStackTrace();
            }
        }

        private void clearFields() {
            cmbPacientes.setSelectedIndex(0); // Seleccionar la opción por defecto
            cmbMedicos.setSelectedIndex(0);   // Seleccionar la opción por defecto
            txtAreaSintomas.setText("");
            txtAreaDiagnostico.setText("");
            txtAreaTratamiento.setText("");
        }
}
```




- PanelHistorial.java
  Permite consultar el historial médico de un paciente específico o listar todas las consultas realizadas por un médico.

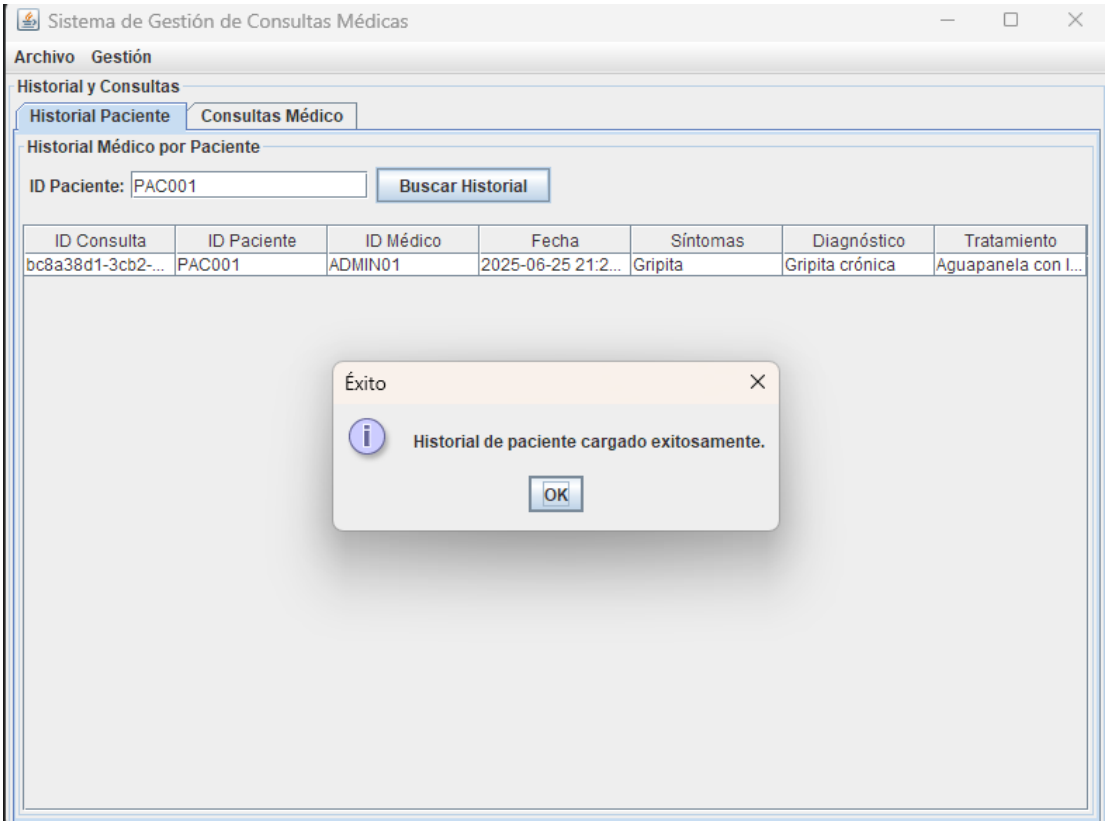
```
package com.simulador.view;

import com.simulador.model.Consulta;
import com.simulador.model.Paciente;
import com.simulador.model.Medico;
import com.simulador.viewmodel.ClinicaViewModel;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
import com.simulador.excepciones.CampoVacioException;
import com.simulador.excepciones.UsuarioNoEncontradoException;

public class PanelHistorial extends JPanel {
   private ClinicaViewModel viewModel;
```

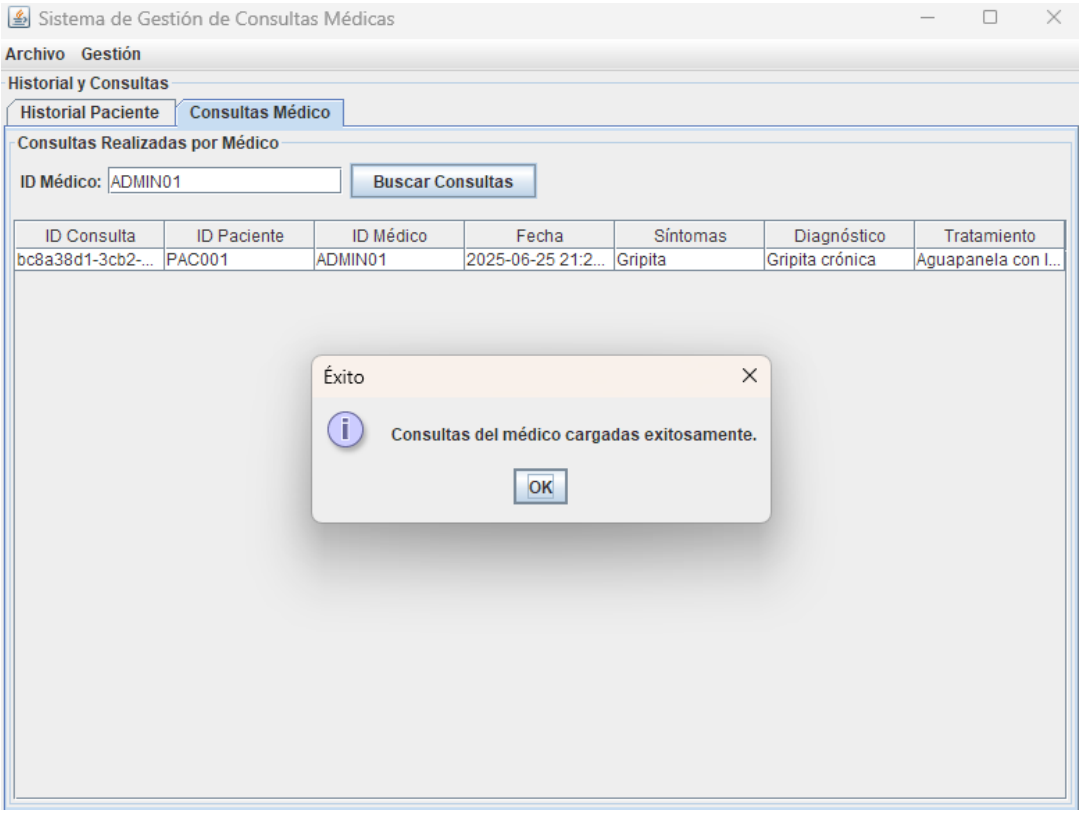
```
    // Componentes para Historial de Paciente
    private JTextField txtIdPacienteHistorial;
    private JButton btnBuscarHistorialPaciente;
    private JTable tablaHistorialPaciente;
    private DefaultTableModel modeloTablaHistorialPaciente;

    // Componentes para Consultas por Médico
    private JTextField txtIdMedicoConsultas;
    private JButton btnBuscarConsultasMedico;
    private JTable tablaConsultasMedico;
    private DefaultTableModel modeloTablaConsultasMedico;

    public PanelHistorial(ClinicaViewModel viewModel) {
        this.viewModel = viewModel;
        setLayout(new BorderLayout(10, 10));
        setBorder(BorderFactory.createTitledBorder("Historial y Consultas"));

        JTabbedPane tabbedPane = new JTabbedPane();

        // --- Panel de Historial de Paciente ---
        JPanel panelHistorialPaciente = new JPanel(new BorderLayout(10, 10));

panelHistorialPaciente.setBorder(BorderFactory.createTitledBorder("Historial
Médico por Paciente"));
        setupHistorialPacientePanel(panelHistorialPaciente);
        tabbedPane.addTab("Historial Paciente", panelHistorialPaciente);

        // --- Panel de Consultas por Médico ---
        JPanel panelConsultasMedico = new JPanel(new BorderLayout(10, 10));

panelConsultasMedico.setBorder(BorderFactory.createTitledBorder("Consultas
Realizadas por Médico"));
        setupConsultasMedicoPanel(panelConsultasMedico);
        tabbedPane.addTab("Consultas Médico", panelConsultasMedico);

        add(tabbedPane, BorderLayout.CENTER);
    }

    private void setupHistorialPacientePanel(JPanel panel) {
        JPanel inputPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        inputPanel.add(new JLabel("ID Paciente:"));
        txtIdPacienteHistorial = new JTextField(15);
        inputPanel.add(txtIdPacienteHistorial);
        btnBuscarHistorialPaciente = new JButton("Buscar Historial");
        inputPanel.add(btnBuscarHistorialPaciente);

        panel.add(inputPanel, BorderLayout.NORTH);
```

```
        // Tabla
        String[] columnNames = {"ID Consulta", "ID Paciente", "ID Médico",
    "Fecha", "Síntomas", "Diagnóstico", "Tratamiento"};
        modeloTablaHistorialPaciente = new DefaultTableModel(columnNames, 0);
        tablaHistorialPaciente = new JTable(modeloTablaHistorialPaciente);
        JScrollPane scrollPane = new JScrollPane(tablaHistorialPaciente);
        panel.add(scrollPane, BorderLayout.CENTER);

        btnBuscarHistorialPaciente.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                buscarHistorialPaciente();
            }
        });
    }

    private void setupConsultasMedicoPanel(JPanel panel) {
        JPanel inputPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        inputPanel.add(new JLabel("ID Médico:"));
        txtIdMedicoConsultas = new JTextField(15);
        inputPanel.add(txtIdMedicoConsultas);
        btnBuscarConsultasMedico = new JButton("Buscar Consultas");
        inputPanel.add(btnBuscarConsultasMedico);

        panel.add(inputPanel, BorderLayout.NORTH);

        // Tabla
        String[] columnNames = {"ID Consulta", "ID Paciente", "ID Médico",
    "Fecha", "Síntomas", "Diagnóstico", "Tratamiento"};
        modeloTablaConsultasMedico = new DefaultTableModel(columnNames, 0);
        tablaConsultasMedico = new JTable(modeloTablaConsultasMedico);
        JScrollPane scrollPane = new JScrollPane(tablaConsultasMedico);
        panel.add(scrollPane, BorderLayout.CENTER);

        btnBuscarConsultasMedico.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                buscarConsultasMedico();
            }
        });
    }

    private void buscarHistorialPaciente() {
        String idPaciente = txtIdPacienteHistorial.getText();
        try {
            List<Consulta> historial =
    viewModel.consultarHistorialPaciente(idPaciente);
            modeloTablaHistorialPaciente.setRowCount(0); // Limpiar tabla
```

```
            if (historial.isEmpty()) {
                JOptionPane.showMessageDialog(this, "No se encontraron consultas
para el paciente con ID: " + idPaciente, "Sin Resultados",
JOptionPane.INFORMATION_MESSAGE);
            } else {
                for (Consulta c : historial) {
                    modeloTablaHistorialPaciente.addRow(new Object[]{
                        c.getIdConsulta(),
                        c.getIdPaciente(),
                        c.getIdMedico(),
                        c.getFechaConsulta(),
                        c.getSintomas(),
                        c.getDiagnostico(),
                        c.getTratamiento()
                    });
                }
                JOptionPane.showMessageDialog(this, "Historial de paciente cargado
exitosamente.", "Éxito", JOptionPane.INFORMATION_MESSAGE);
            }
        } catch (CampoVacioException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error de
Validación", JOptionPane.WARNING_MESSAGE);
        } catch (UsuarioNoEncontradoException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error al buscar historial: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
        }
    }

    private void buscarConsultasMedico() {
        String idMedico = txtIdMedicoConsultas.getText();
        try {
            List<Consulta> consultas =
viewModel.listarConsultasPorMedico(idMedico);
            modeloTablaConsultasMedico.setRowCount(0); // Limpiar tabla
            if (consultas.isEmpty()) {
                JOptionPane.showMessageDialog(this, "No se encontraron consultas
para el médico con ID: " + idMedico, "Sin Resultados",
JOptionPane.INFORMATION_MESSAGE);
            } else {
                for (Consulta c : consultas) {
                    modeloTablaConsultasMedico.addRow(new Object[]{
                        c.getIdConsulta(),
                        c.getIdPaciente(),
                        c.getIdMedico(),
```

```
                        c.getFechaConsulta(),
                        c.getSintomas(),
                        c.getDiagnostico(),
                        c.getTratamiento()
                    });
                }
                JOptionPane.showMessageDialog(this, "Consultas del médico cargadas
        exitosamente.", "Éxito", JOptionPane.INFORMATION_MESSAGE);
            }
        } catch (CampoVacioException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error de
        Validación", JOptionPane.WARNING_MESSAGE);
        } catch (UsuarioNoEncontradoException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error",
        JOptionPane.ERROR_MESSAGE);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error al buscar consultas del
        médico: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
        }
    }
}
```






### 3.6 Clase Main

La clase `Main` es el punto de inicio de la aplicación, encargada de asegurar que la interfaz de usuario se ejecute.

```
// Main.java
package com.simulador.main;

import com.simulador.view.VentanaPrincipal;
import javax.swing.SwingUtilities;
//import com.simulador.persistencia.PersistenciaFirebase;
//import com.simulador.model.Medico;
//import com.simulador.model.Paciente;

public class Main {
    public static void main(String[] args) {

        // Ejecutar la interfaz de usuario en el Event Dispatch Thread (EDT)
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                VentanaPrincipal ventana = new VentanaPrincipal();
                ventana.setVisible(true);
            }
        });
    }
```

```
}
```

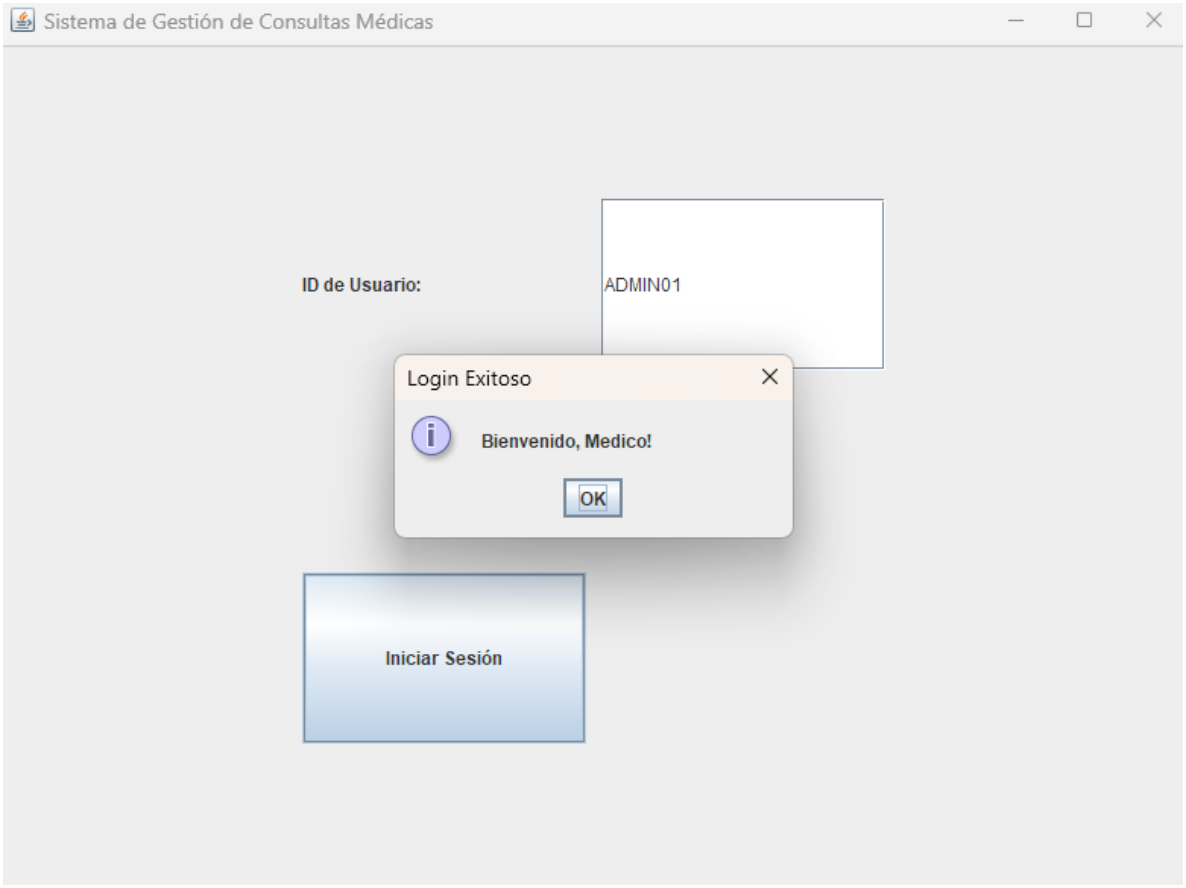
**4. Evidencia de Funcionamiento y Casos de Uso**

En esta sección, se presentarán las capturas de pantalla que demuestran el correcto funcionamiento de cada requerimiento funcional.

**4.1. Login de Usuario Médico o Administrativo (Requerimiento 1)**

- **Paso a paso:**
    1. Iniciar la aplicación.
    2. Ingresar un ID de médico registrado previamente en Firebase (ej. ADMIN001).
    3. Hacer clic en "Iniciar Sesión".

- **Evidencia:**

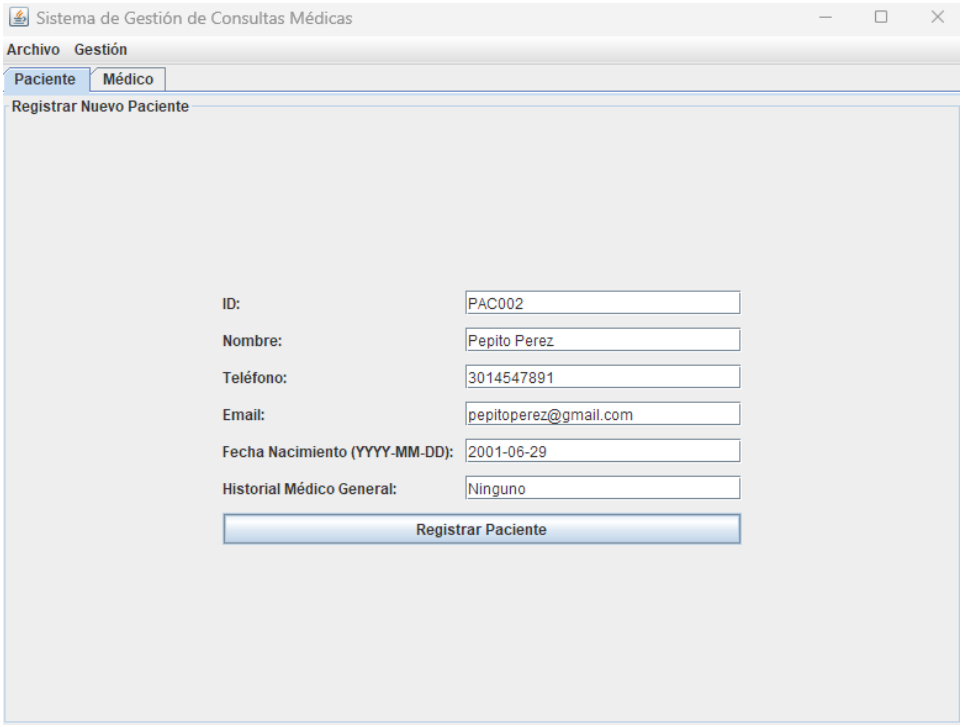


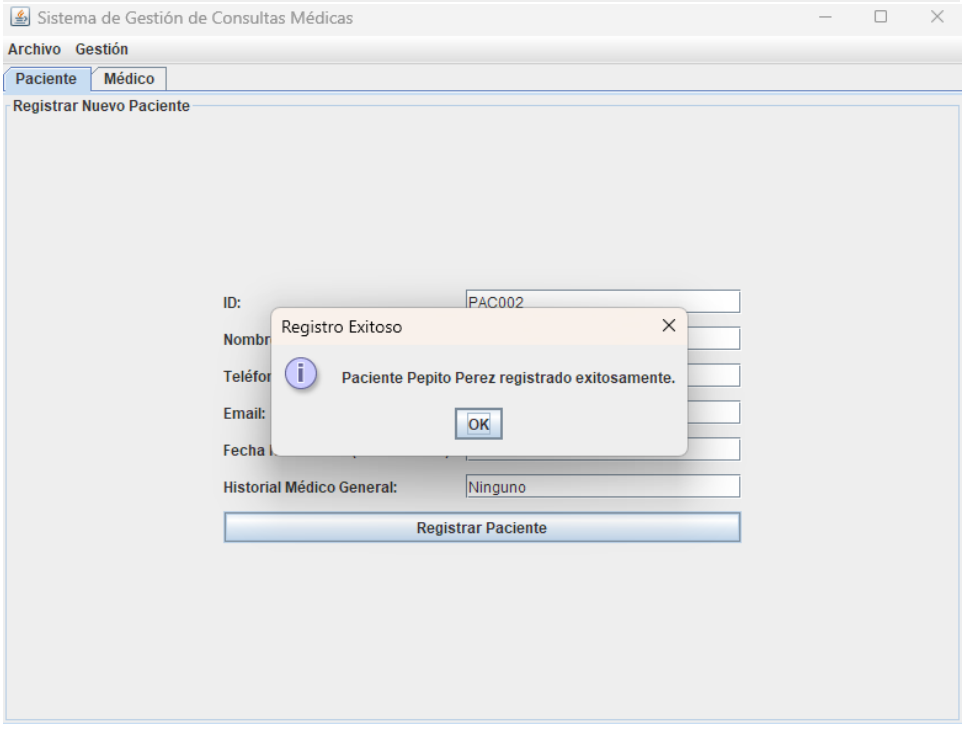

- **Caso de Fallo (Paciente intentando iniciar sesión):**
    1. Ingresar un ID de paciente (ej. PAC001).
    2. Hacer clic en "Iniciar Sesión".

- **Evidencia de Falla:**


**4.2. Registrar Nuevos Pacientes y Médicos (Requerimiento 2)**

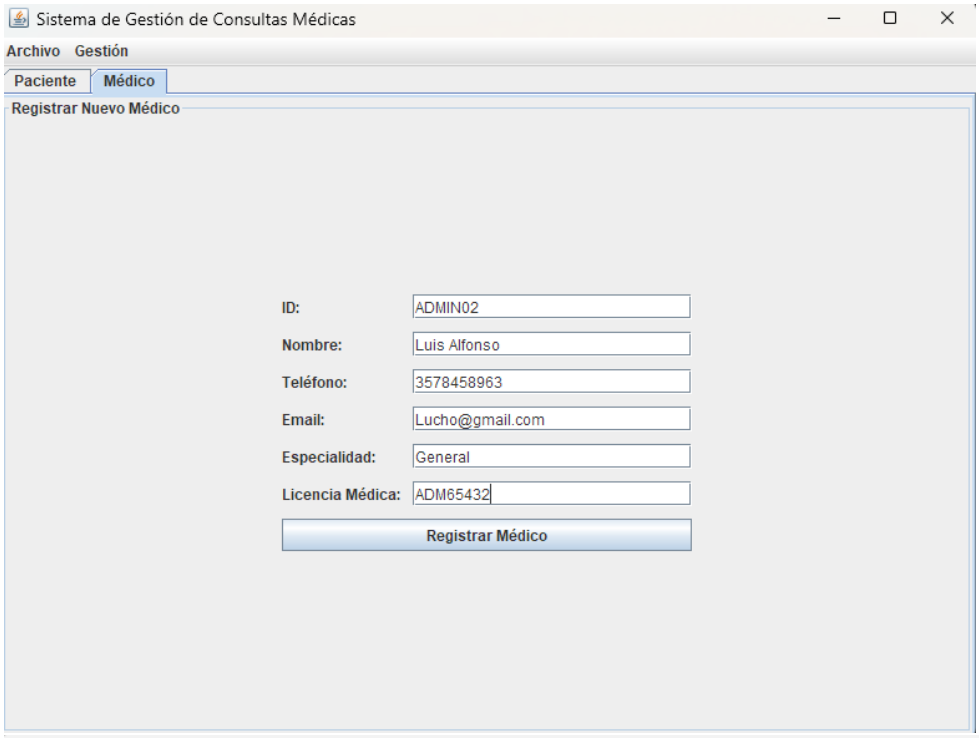
- **Paso a paso (Paciente):**
    1. Iniciar sesión como médico/administrador.
    2. Navegar a "Gestión" -> "Registrar Persona".
    3. Seleccionar la pestaña "Paciente".
    4. Completar todos los campos del formulario de paciente.
    5. Hacer clic en "Registrar Paciente".
- **Evidencia (Registro Paciente):**

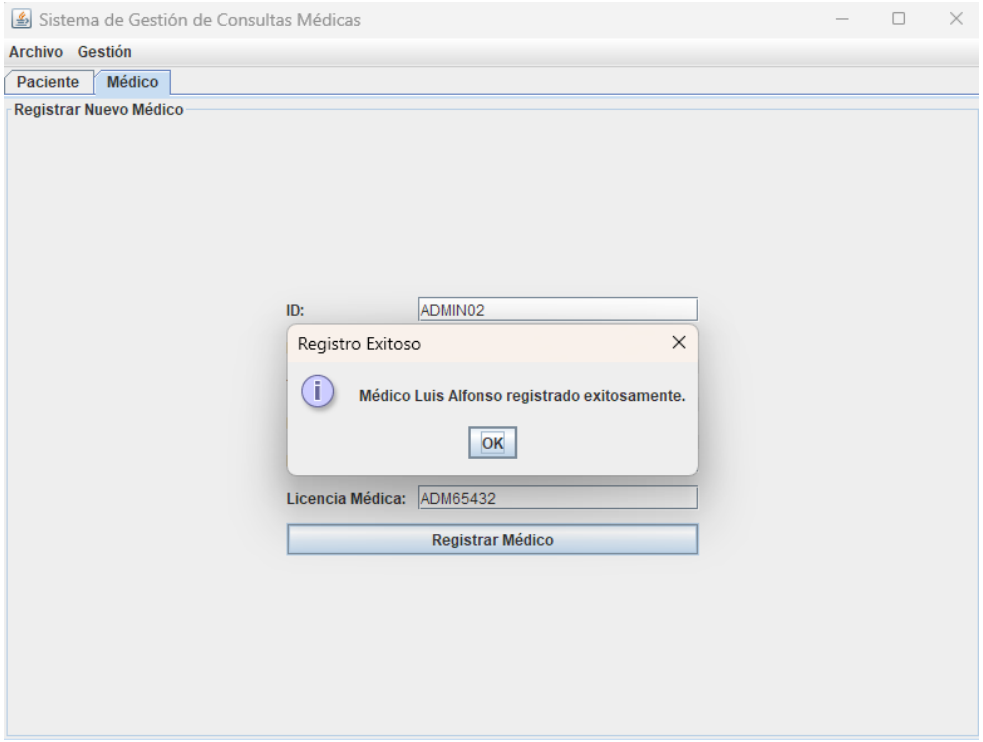



- **Paso a paso (Médico):**
    1. Iniciar sesión como médico/administrador.
    2. Navegar a "Gestión" -> "Registrar Persona".
    3. Seleccionar la pestaña "Médico".
    4. Completar todos los campos del formulario de médico.
    5. Hacer clic en "Registrar Médico".

- **Evidencia (Registro Médico):**



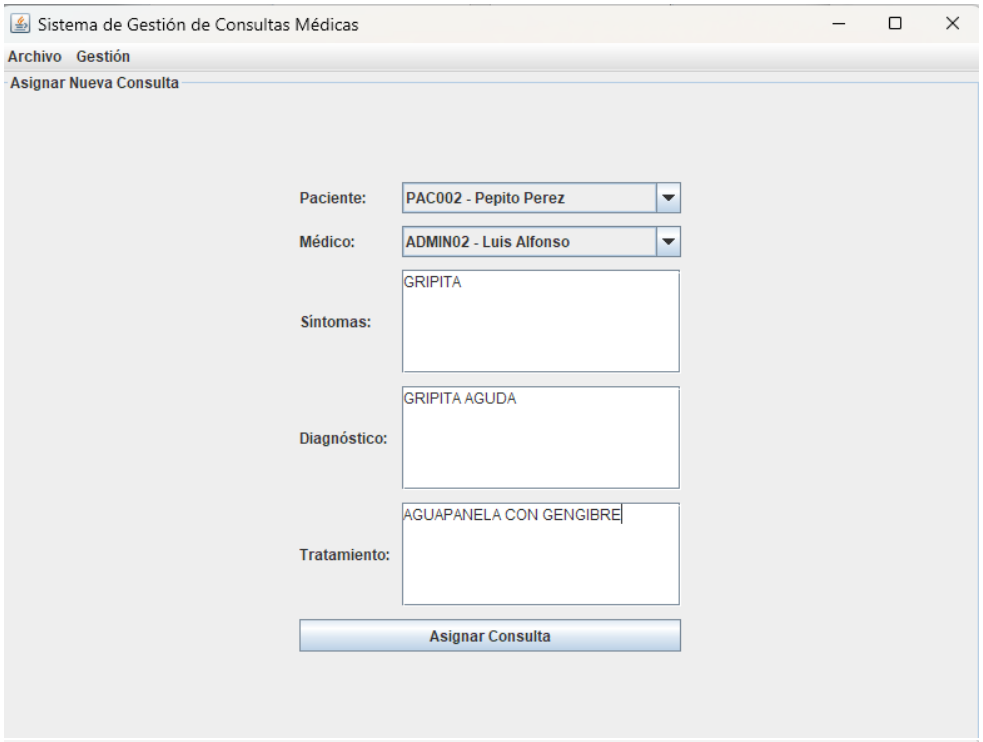



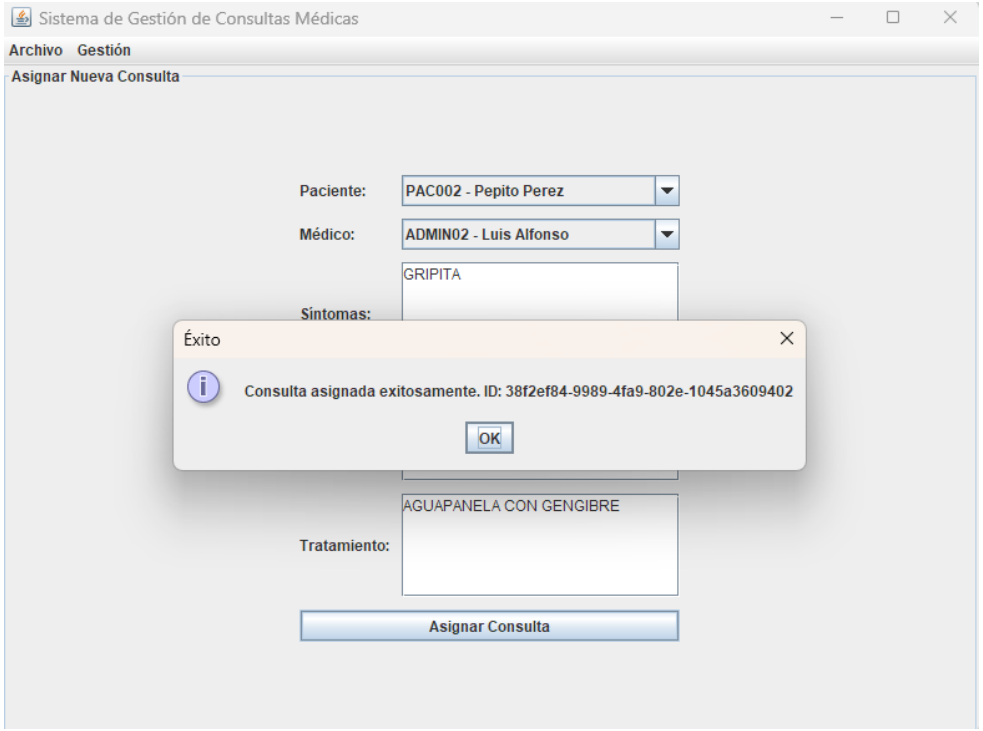

### 4.3. Asignar una Consulta (Requerimiento 3)

- **Paso a paso:**
    1. Iniciar sesión como médico/administrador.
    2. Navegar a "Gestión" -> "Asignar Consulta".
    3. Seleccionar un paciente y un médico de las listas desplegables.

4. Ingresar los síntomas, diagnóstico y tratamiento.
5. Hacer clic en "Asignar Consulta".

- **Evidencia:**



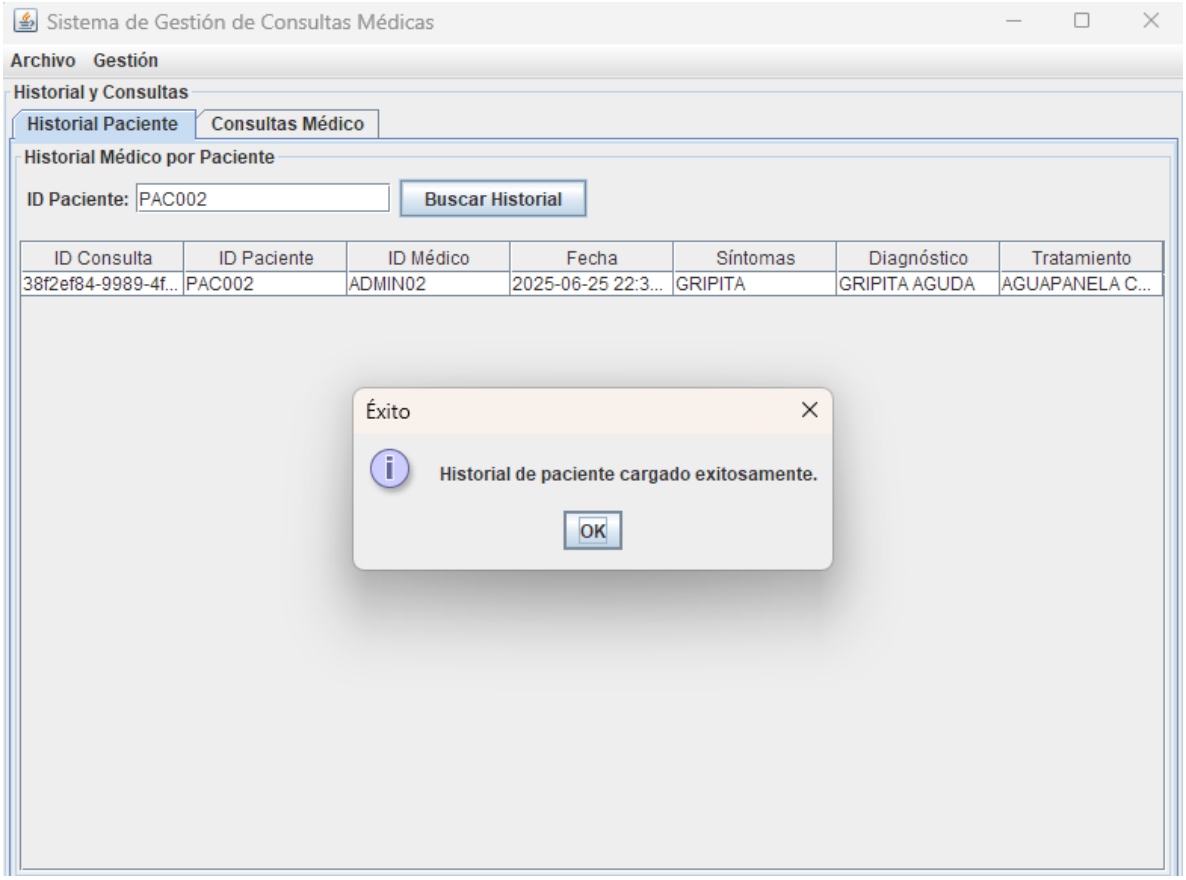




**4.4. Consultar el Historial Médico de un Paciente (Requerimiento 4)**

- **Paso a paso:**
  1. Iniciar sesión como médico/administrador.

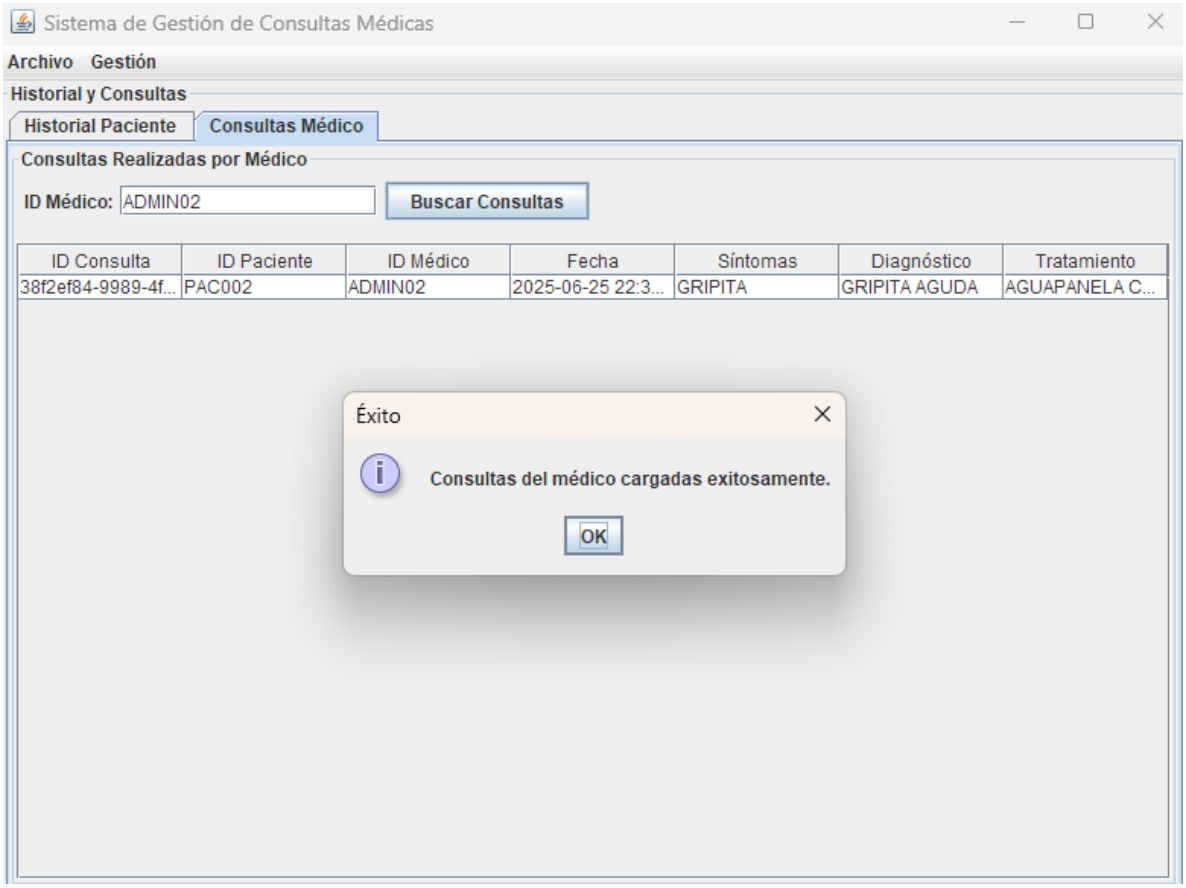
2. Navegar a "Gestión" -> "Historial Médico".
3. Asegurarse de estar en la pestaña "Historial Paciente".
4. Ingresar el ID de un paciente existente.
5. Hacer clic en "Buscar Historial".

- **Evidencia:**




**4.5. Listar Todas las Consultas Realizadas por un Médico (Requerimiento 5)**

- **Paso a paso:**
  1. Iniciar sesión como médico/administrador.
  2. Navegar a "Gestión" -> "Historial Médico".
  3. Seleccionar la pestaña "Consultas Médico".
  4. Ingresar el ID de un médico existente.
  5. Hacer clic en "Buscar Consultas".
- **Evidencia:**




---

**5. Repositorio GitHub**

El código fuente completo del proyecto está disponible en el siguiente repositorio de GitHub:
https://github.com/Felipe-un/SimuladorConsultasMedicas.git

---