24/10/2025

Sistema de gestión de pedidos de recetas médicas en un consultorio

Seminario de Practica

Trabajo Practico Nº3

Universidad Siglo 21

Profesor Titular: PABLO ALEJANDRO VIRGOLINI

Titular Experto: ANA CAROLINA FERREYRA

Introducción general:

Este proyecto se realiza en base a mi trabajo en consultorio médico en un barrio de clase media de la ciudad de Neuquén con gran población de adultos mayores.

En este consultorio la comunicación con los pacientes se organiza a través de dos teléfonos celulares: uno destinado exclusivamente a la gestión de turnos y otro al pedido de recetas médicas. En el caso de las recetas, los pacientes solicitan la renovación de medicación crónica o puntual mediante mensajes de WhatsApp.

Actualmente, la secretaria o el personal administrativo recibe estos mensajes y los registra de manera manual en un Excel con todos los datos requerido para una receta. El medico desde un drive compartido lo visualiza y confecciona la receta en una aplicación de receta digital externa al consultorio o en forma manual. Sin embargo, las historias clínicas no están digitalizada, se sigue trabajando con sistema de fichero manual (fichas de papel), existiendo un sistema mixto la historia clínica en papel y la receta digital. No hay un sistema centralizado o de búsqueda rápida que permita el acceso a los datos y recetas confeccionadas ya que las recetas quedan guardadas en la aplicación externa de receta digital. Esta situación provoca dificultades como la pérdida de información, retrasos en la entrega de recetas, errores en la carga de medicamentos y falta de control sobre la frecuencia de renovación de cada paciente.

Este trabajo busca diseñar un sistema automatizado que gestione las solicitudes de recetas médicas recibidas por WhatsApp, registre los pedidos en una base de datos, permita generar alertas y mejore la organización interna del consultorio.

Este procedimiento provoca dificultades como:

Pérdida de información.

Retrasos en la entrega de recetas.

Errores en la carga de medicamentos.

Falta de control sobre la frecuencia de renovación de los pacientes.

Este trabajo tiene como objetivo diseñar un sistema automatizado que gestione las solicitudes de recetas recibidas por WhatsApp, registre los pedidos en una base de datos, genere alertas y mejore la organización interna del consultorio.

Antecedentes:

En la práctica tradicional, los pacientes acudían personalmente al consultorio para solicitar recetas. Con el tiempo, el teléfono fijo y posteriormente WhatsApp se convirtieron en medios habituales para realizar estos pedidos, facilitando el acceso y evitando traslados.

Algunos consultorios adoptaron aplicaciones externas de prescripción digital o integraron módulos dentro de sistemas de historia clínica electrónica. Sin embargo, estas soluciones suelen requerir una inversión elevada y conocimientos técnicos, lo que no siempre resulta viable para consultorios pequeños ni para pacientes mayores.

Por ello, muchos profesionales siguen trabajando con sistemas mixtos (parte digital y parte manual), lo que genera problemas de control y organización.

Descripción del área problemática:

El teléfono destinado a recetas recibe diariamente solicitudes de pacientes, principalmente adultos mayores. La secretaria registra estas solicitudes en Excel, que luego son consultadas por el médico para confeccionar la receta en una aplicación externa o en papel.

Problemas principales:

Ausencia de un sistema centralizado.

Pérdida de mensajes en WhatsApp.

Retrasos en la entrega de recetas por sobrecarga de trabajo.

Errores de transcripción.

Falta de control sobre la frecuencia de renovaciones.

En síntesis, el proceso es manual, fragmentado y propenso a errores, afectando tanto al personal como a los pacientes.

Justificación:

La implementación de un sistema de gestión de recetas permitirá:

Optimizar el tiempo de la secretaria y del médico.

Reducir errores humanos.

Evitar retrasos y pérdidas de información.

Facilitar el seguimiento de pacientes crónicos.

Mantener WhatsApp como canal accesible para los adultos mayores.

El sistema brindará un servicio más eficiente, seguro y confiable.

Objetivo general del proyecto:

Desarrollar un sistema automatizado de gestión de recetas médicas que integre WhatsApp con una base de datos en la nube para registrar, organizar y dar seguimiento a las solicitudes de los pacientes.

Objetivos específicos:

Analizar el proceso actual de gestión de recetas en el consultorio.

Identificar las limitaciones del sistema manual.

Diseñar un sistema que registre automáticamente las solicitudes recibidas por WhatsApp.

Implementar un historial de recetas por paciente.

Incorporar alertas y notificaciones para evitar retrasos.

Objetivo general del sistema:

El sistema permitirá registrar automáticamente las solicitudes de recetas enviadas por WhatsApp, organizarlas en una base de datos, generar historiales y emitir alertas que mejoren la eficiencia del consultorio.

Alcances y límites:

Alcances:

Recepción de solicitudes por WhatsApp.

Registro automático en la base de datos.

Organización por paciente, fecha y medicamento.

Generación de historiales.

Emisión de alertas de recetas pendientes.

Límites:

Desde la recepción del mensaje en WhatsApp.

Hasta el registro del pedido en la base de datos y confirmación de gestión.

No contempla:
Gestión de turnos médicos.

Manejo completo de historias clínicas.

Integración con farmacias u obras sociales.

Conocimiento del negocio:
Pacientes: en su mayoría adultos mayores, muchos con enfermedades crónicas.

Canal de comunicación: WhatsApp (un celular exclusivo para turnos, otro para recetas).

Personal:

Secretaria: recibe mensajes y los registra.

Médico: confecciona la receta digital o manualmente.

Almacenamiento actual:

Historias clínicas en papel.

Recetas en aplicaciones externas.

Problemas actuales:

Información fragmentada.

Pérdida de mensajes.

Retrasos por sobrecarga administrativa.

Errores de transcripción.

Imposibilidad de controlar la frecuencia de renovación.

Competencia
Método actual: WhatsApp + Google Sheets.

Ventajas: gratuito, accesible y conocido.

Desventajas: trabajo manual, errores frecuentes, falta de historial centralizado.

Opción externa (Charta): plataforma argentina reconocida de recetas digitales.

Ventajas: elimina el papel y agiliza la emisión.

Desventajas: costos adicionales, mayor complejidad y poca adaptación a pacientes mayores.

Propuesta del proyecto: migrar a SharePoint, manteniendo WhatsApp como canal principal. Esto permite mejorar la organización, seguridad y acceso a la información sin perder accesibilidad para los pacientes.

Propuesta de solución:
El sistema centralizará las solicitudes de recetas integrando WhatsApp con una base de datos y SharePoint como repositorio documental. Permitirá consultas rápidas, generación de reportes y alertas automáticas, optimizando el trabajo del consultorio sin modificar el canal de comunicación habitual de los pacientes.

Elicitación:
Área de aplicación: gestión de recetas médicas.

TICs a utilizar:

WhatsApp Business API.

SharePoint como repositorio central.

Google Drive/Sheets como respaldo.

Base de datos MySQL.

Lenguajes de programación: Java o Python.

Requerimientos:

Funcionales:

Registrar automáticamente solicitudes de recetas recibidas por WhatsApp.

Almacenar datos básicos: paciente, medicamento, fecha y estado.

Organizar pedidos por paciente y fecha.

Actualizar el estado de la receta.

Consultar historial por paciente.

Generar alertas de recetas pendientes.

Gestionar roles de acceso.

Exportar reportes en Excel o PDF.

No funcionales:

Interfaz sencilla e intuitiva.

Acceso desde distintos dispositivos.

Seguridad mediante usuario y contraseña.

Disponibilidad mínima del 95%.

Procesamiento menor a 2 segundos.

Escalabilidad para futuras funciones.

Copias de seguridad automáticas.

Compatibilidad con nube (Google Drive y SharePoint).

Casos de uso

Código	Nombre del caso de uso	Actor/es	Descripción
CU-01	Solicitar receta	Paciente	Envía mensaje de WhatsApp solicitando una receta.
CU-02	Registrar solicitud	Secretaria	Carga el pedido en el sistema con los datos básicos.
CU-03	Confeccionar receta	Médico	Elabora la receta correspondiente.
CU-04	Actualizar estado	Secretaria / Médico	Cambia el estado del pedido (pendiente, emitida, entregada).
CU-05	Consultar historial	Médico / Secretaria	Revisa historial de recetas emitidas.
CU-06	Generar alertas	Sistema	Señala recetas pendientes o próximas a renovar.

Descripción de casos de uso

CU-01: Solicitar receta

Actor principal: Paciente.

Precondición: estar registrado y contar con WhatsApp.

Flujo normal: el paciente envía un mensaje solicitando receta; el sistema lo recibe.

Excepciones: si faltan datos, la secretaria los solicita.

Postcondición: la solicitud queda registrada.

CU-02: Registrar solicitud

Actor principal: Secretaria.

Precondición: paciente envió previamente el mensaje.

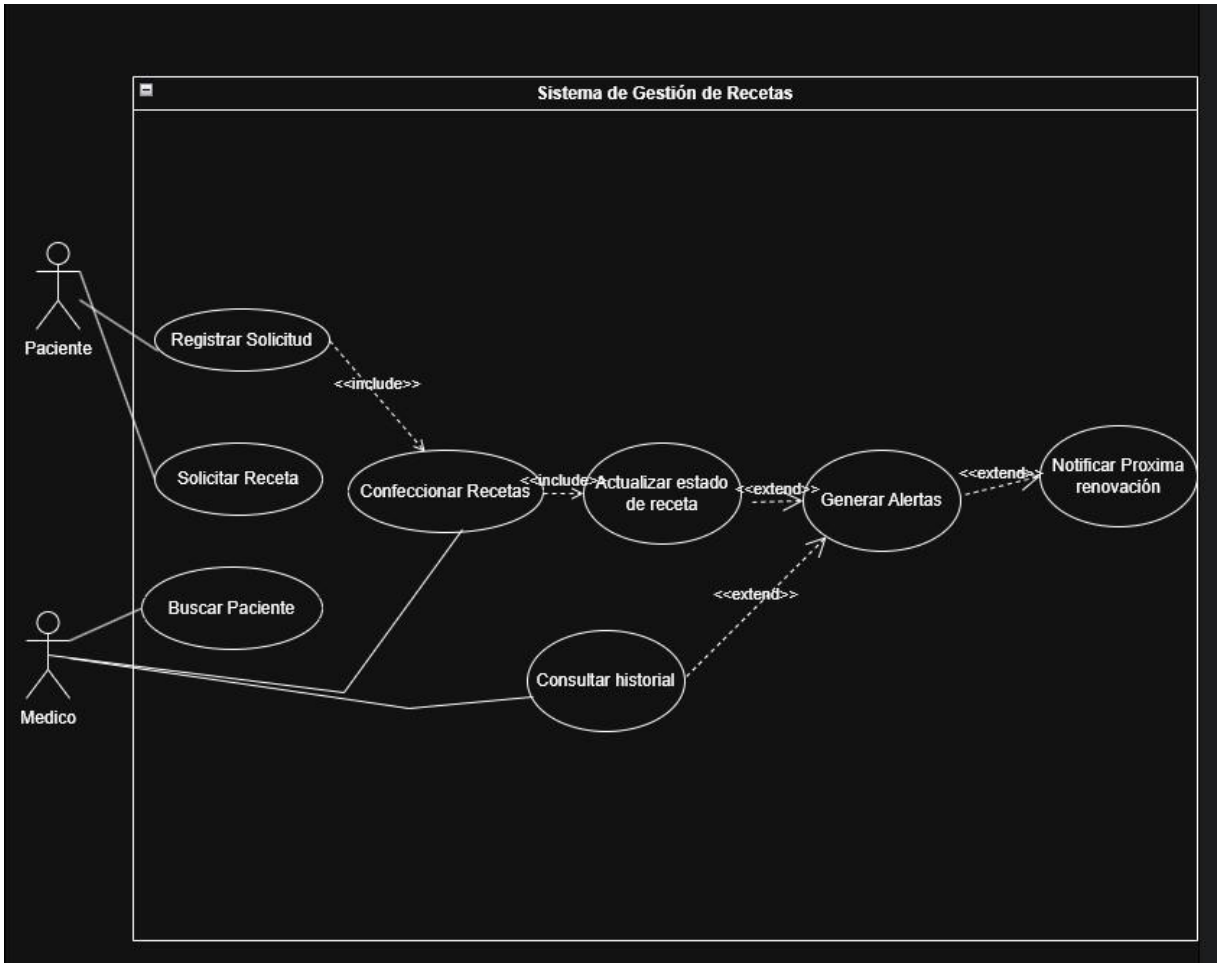
Flujo normal: la secretaria ingresa al sistema, carga los datos (paciente, medicamento, fecha, estado) y guarda.

Excepciones: si la información está incompleta, debe pedirla al paciente; si el paciente no existe, lo registra.

Postcondición: el pedido queda disponible para revisión del médico.

Cronograma

Etapa		**Duración**
Relevamiento del problema		1 semana
Definición de requerimientos		1 semana
Diseño del sistema		2 semanas
Desarrollo del prototipo		3 semanas
Pruebas y validación		2 semanas
Entrega final		1 semana



Diseño del Sistema:

Diagrama de secuencia : solicitar Receta

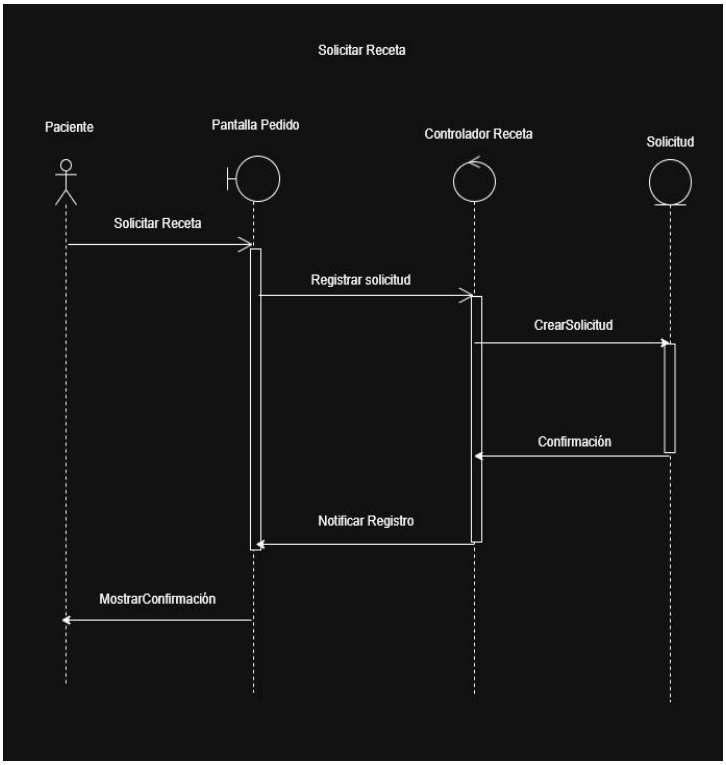


Diagrama de secuencia : Registrar Solicitud

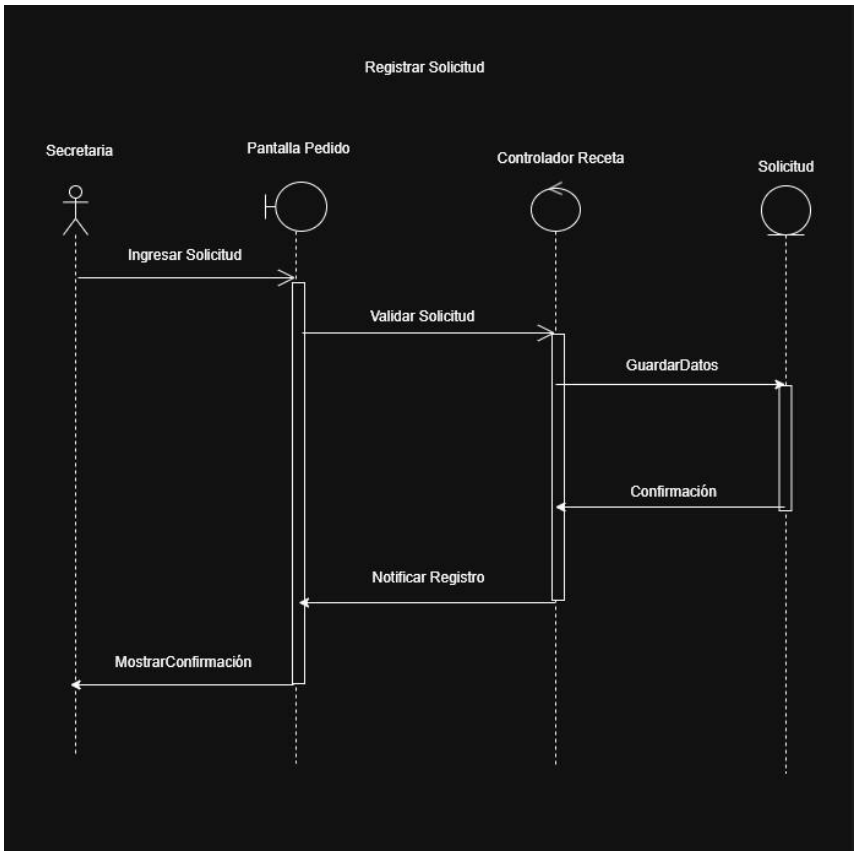


Diagrama de secuencia: Confeccionar Receta

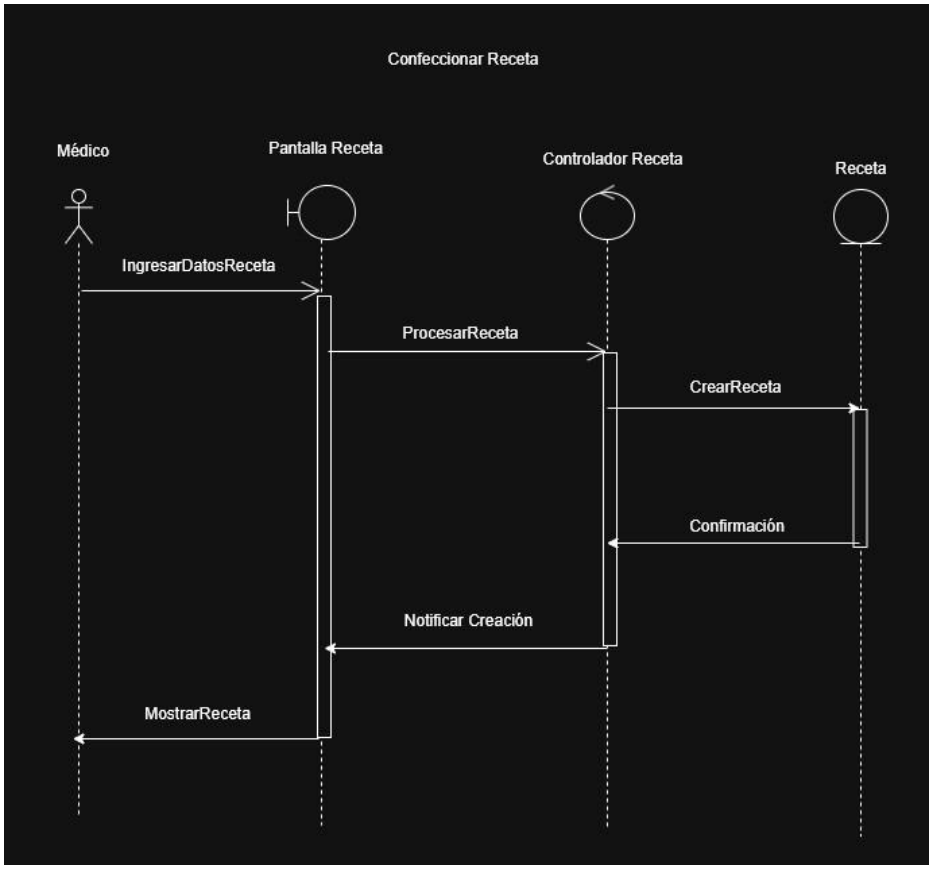


Diagrama de secuencia : Actualiza Estado

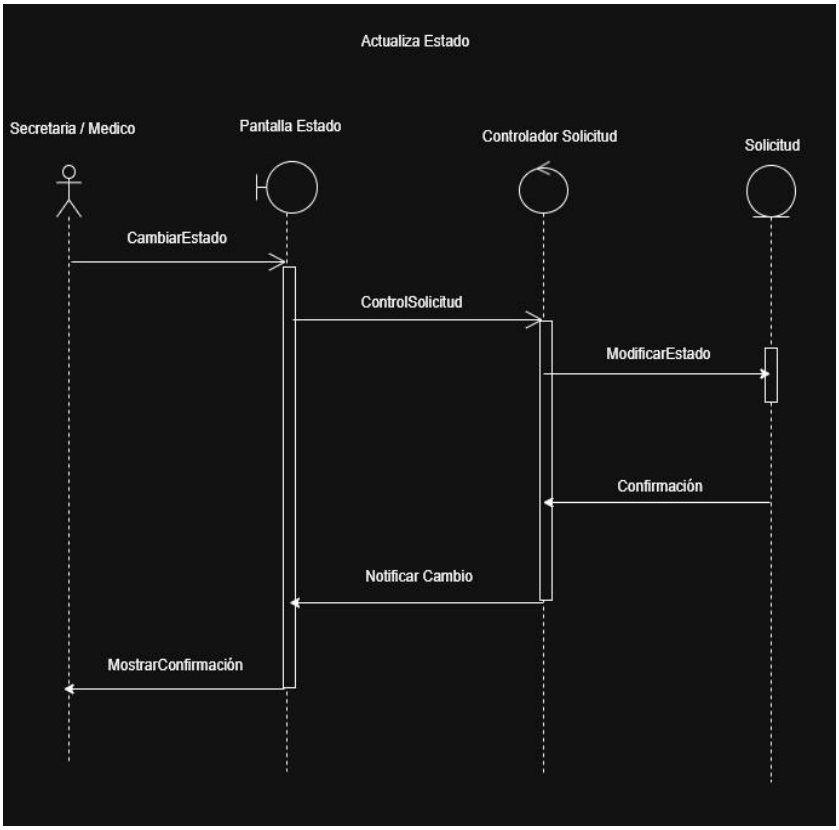


Diagrama de secuencia : Generar Alerta

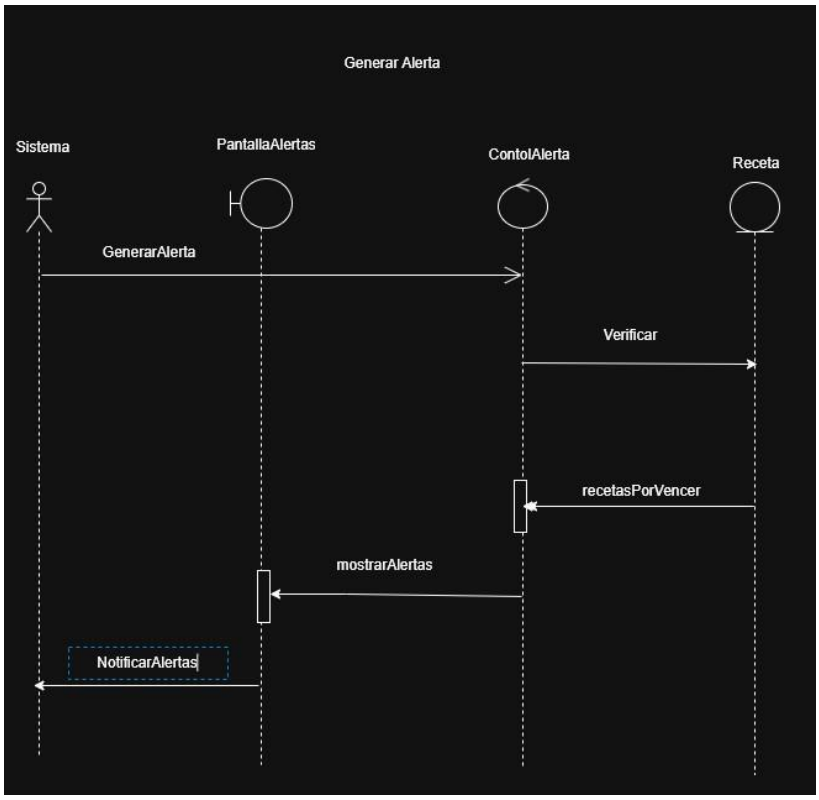
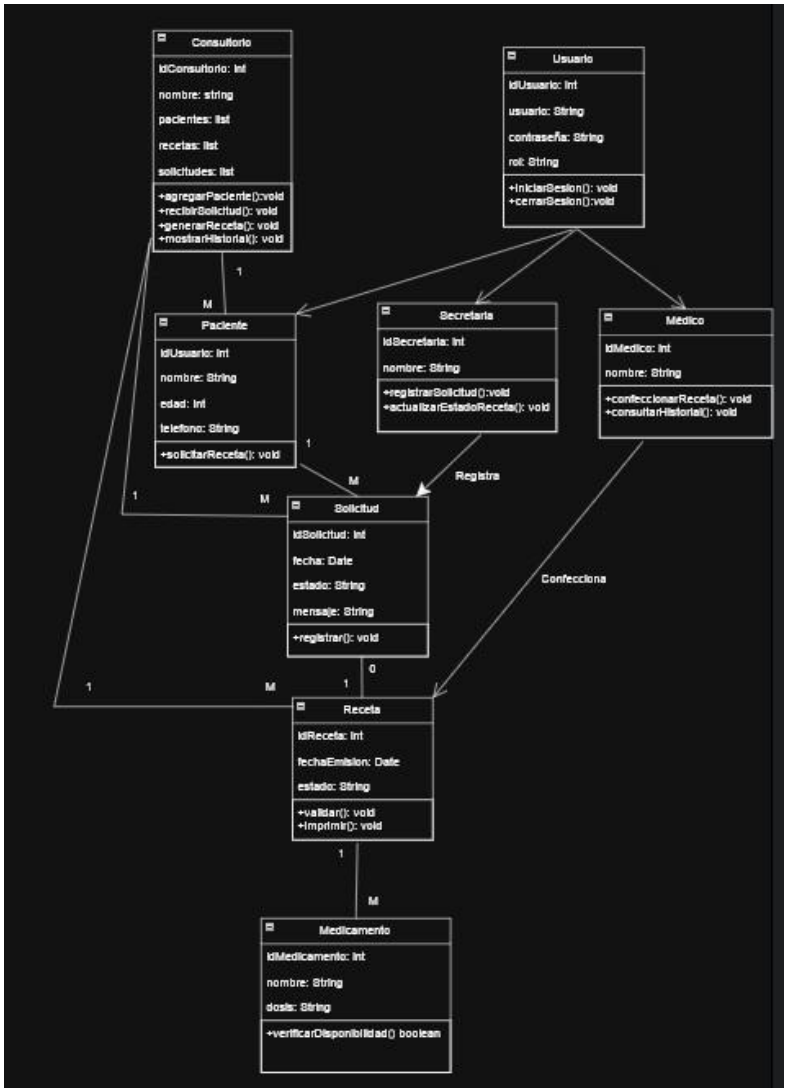


Diagrama de clases de diseño:



Etapa de Implementación:

La etapa de implementación corresponde al momento en el cual el sistema diseñado comienza a ponerse en funcionamiento en un entorno real. En esta fase se realiza la instalación del software, la configuración de los servidores y la base de datos, así como la carga inicial de los usuarios que harán uso del sistema (pacientes, secretaria y médico).

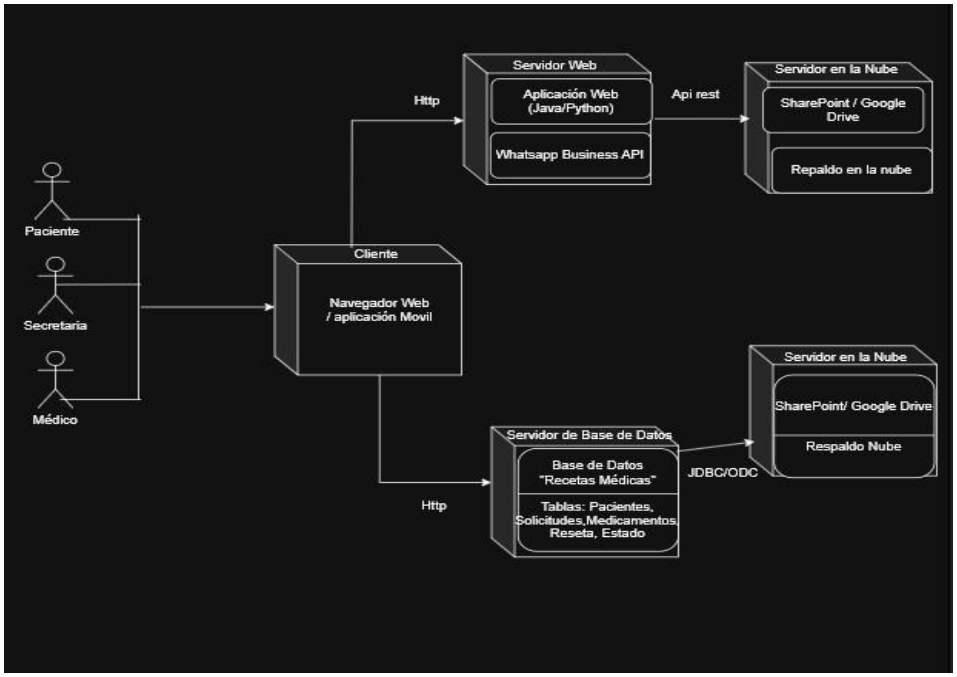
En este proyecto, el sistema de gestión de recetas médicas será desplegado utilizando Java como lenguaje de programación, MySQL como gestor de base de datos y XAMPP como entorno de desarrollo web. Estos elementos permitirán asegurar que el sistema sea accesible, escalable y fácil de mantener.

El diagrama de despliegue resulta fundamental en esta etapa, ya que permite visualizar cómo se distribuyen los diferentes componentes del sistema en los nodos físicos y cómo se comunican entre sí. En este caso, el sistema estará conformado por tres grandes bloques:

- Cliente: corresponde a los dispositivos de los usuarios (paciente, secretaria y médico), que accederán al sistema a través de un navegador web o aplicación móvil.
- Servidor de Aplicación: donde se ejecuta la lógica del sistema desarrollada en Java, incluyendo la gestión de solicitudes, recetas y usuarios.
- Servidor de Base de Datos: donde se almacenan de manera estructurada los registros de pacientes, solicitudes y recetas, garantizando la integridad de la información.

De esta manera, la etapa de implementación asegura que el sistema diseñado pase a ser una herramienta funcional dentro del consultorio, permitiendo que los usuarios interactúen con él de manera práctica y ordenada.

RF	Descripción
RF01:	Registrar automáticamente las solicitudes de recetas recibidas por WhatsApp.
RF02:	Guardar los datos básicos de cada pedido: paciente, medicamento, fecha y estado.
RF03:	Organizar los pedidos por paciente, fecha y tipo de medicamento.
RF04:	Actualizar el estado de la receta (pendiente, en proceso, entregada).
RF05:	Consultar el historial de recetas por paciente.
RF06:	Generar alertas sobre recetas pendientes o próximas a renovar.
RF07:	Gestionar diferentes roles de acceso (paciente, secretaria, médico).
RF08:	Exportar reportes (Excel o PDF) de recetas por fecha o paciente.
RNF	Descripción
RF01:	La interfaz debe ser sencilla e intuitiva para el personal.
RF02:	El sistema debe ser accesible desde distintos dispositivos (PC, notebook, celular).
RF03:	Los datos deben estar protegidos mediante usuario y contraseña.
RF04:	El sistema debe estar disponible al menos el 95% del tiempo.
RF05:	El registro de solicitudes no debe superar los 2 segundos de procesamiento.
RF06:	El sistema debe ser escalable (permitir integrar funciones futuras como turnos médicos).
RF07:	Debe tener copias de seguridad automáticas para evitar pérdida de información.
RF08:	Debe ser compatible con la nube (Google Drive y SharePoint).



Plan de prueba:

Caso de Uso	Código de Prueba	Tipo de Prueba	Técnica Propuesta	Observaciones
CU01: Solicitar receta	CP01	Funcional	Caja Negra	Verifica que el sistema reciba una solicitud por WhatsApp y la registre como pendiente.
CU02: Registrar solicitud	CP02	Funcional	Caja Negra	Verifica que la secretaria pueda registrar una solicitud con los datos del paciente y medicamento.
CU03: Confeccionar receta	CP03	Funcional	Caja Negra	Verifica que el médico pueda crear y guardar una receta digital correctamente.
CU04: Actualizar estado	CP04	Funcional	Caja Negra	Verifica que el sistema permita cambiar el estado de la solicitud (Pendiente → Emitida → Entregada).
CU05: Consultar historial	CP05	Funcional	Caja Negra	Verifica que el sistema muestre el historial de recetas de un paciente al consultar por DNI.

CU06: Generar alertas	CP06	No Funcional	Carga	Verifica que el sistema pueda generar alertas cuando hay múltiples solicitudes pendientes sin afectar el rendimiento.
CU07: Seguridad de acceso	CP07	No Funcional	Penetración	Verifica que el sistema bloquee accesos no autorizados y proteja los datos de los pacientes.
CU08: Validación interna de registro	CP08	Funcional	Caja Blanca	Verifica que la función registrarSolicitud() valide correctamente datos obligatorios antes de grabar (paciente existente, medicamento válido, fecha correcta).

Casos de prueba detallados:

Prueba de Solicitud de Receta – CP01

Objetivo: Verificar que el sistema registre correctamente las solicitudes enviadas por WhatsApp.

Pasos de la prueba:

Enviar mensaje con todos los datos (paciente, medicamento, dosis).

Enviar mensaje con datos incompletos.

Enviar mensaje con caracteres inválidos.

Resultado esperado:

El sistema registra correctamente las solicitudes completas.

Rechaza mensajes incompletos solicitando corrección.

Muestra error de formato cuando se envían datos inválidos.

1. Prueba de Registro Manual de Solicitud – CP02

Objetivo: Verificar que la secretaria pueda registrar solicitudes manualmente.

Pasos de la prueba:

Registrar solicitud con todos los datos completos.

Intentar registrar sin completar campos obligatorios.

Registrar con datos inválidos (ejemplo: fecha errónea).

Resultado esperado:

El sistema guarda correctamente solicitudes completas.

Bloquea registros incompletos y muestra “Campos obligatorios faltantes”.

Rechaza datos inválidos y muestra mensaje de error.

Prueba de Confección de Receta – CP03

Objetivo: Verificar que el médico pueda confeccionar recetas digitales.

Pasos de la prueba:

Seleccionar solicitud pendiente y completar receta.

Intentar guardar receta sin indicar medicamento.

Intentar confeccionar receta para paciente inexistente.

Resultado esperado:

El sistema guarda recetas completas y cambia estado a “Emitida”.

No permite guardar recetas incompletas.

Muestra “Paciente no registrado” si el paciente no existe.

Prueba de Actualización de Estado – CP04

Objetivo: Verificar que se pueda actualizar el estado de las solicitudes.

Pasos de la prueba:

Cambiar de “Pendiente” a “En proceso”.

Cambiar de “En proceso” a “Emitida”.

Cambiar de “Emitida” a “Entregada”.

Intentar cambiar de “Entregada” a otro estado.

Resultado esperado:

Los cambios de estado válidos se registran correctamente.

El sistema impide retrocesos luego de “Entregada”.

Prueba de Consulta de Historial – CP05

Objetivo: Verificar que se pueda consultar el historial de recetas de un paciente.

Pasos de la prueba:

Ingresar DNI de paciente con recetas.

Ingresar DNI de paciente sin recetas.

Ingresar DNI inexistente.

Resultado esperado:

El sistema muestra todas las recetas asociadas al paciente.

Muestra “No existen recetas registradas” para pacientes sin historial.

Muestra “Paciente no encontrado” si el DNI no existe.

Prueba de Generación de Alertas – CP06

Objetivo: Verificar que se generen alertas por solicitudes pendientes.

Pasos de la prueba:

Registrar varias solicitudes y dejar algunas sin atender por más de 24 horas.

Procesar solicitudes en menos de 24 horas.

Sobrecargar con múltiples solicitudes simultáneas.

Resultado esperado:

Se generan alertas para solicitudes vencidas.

No aparecen alertas para solicitudes atendidas en tiempo.

El sistema maneja la carga sin fallos.

Prueba de Seguridad de Acceso – CP07

Objetivo: Verificar que el acceso al sistema sea seguro.

Pasos de la prueba:

Ingresar credenciales válidas.

Ingresar credenciales inválidas.

Intentar acceder a datos sin login.

Repetir accesos inválidos varias veces.

Resultado esperado:

El sistema permite acceso solo con credenciales correctas.

Muestra “Usuario o contraseña inválida” para accesos incorrectos.

Bloquea accesos directos a datos sin autenticación.

Restringe al usuario tras múltiples intentos fallidos.

8. Prueba de Validación Interna de Registro – CP08

Objetivo: Verificar que la función registrarSolicitud realice validaciones internas antes de grabar en la base.

Pasos de la prueba:

Ejecutar función con datos completos y válidos.

Ejecutar función con paciente inexistente.

Ejecutar función con medicamento vacío.

Ejecutar función con fecha inválida.

Resultado esperado:

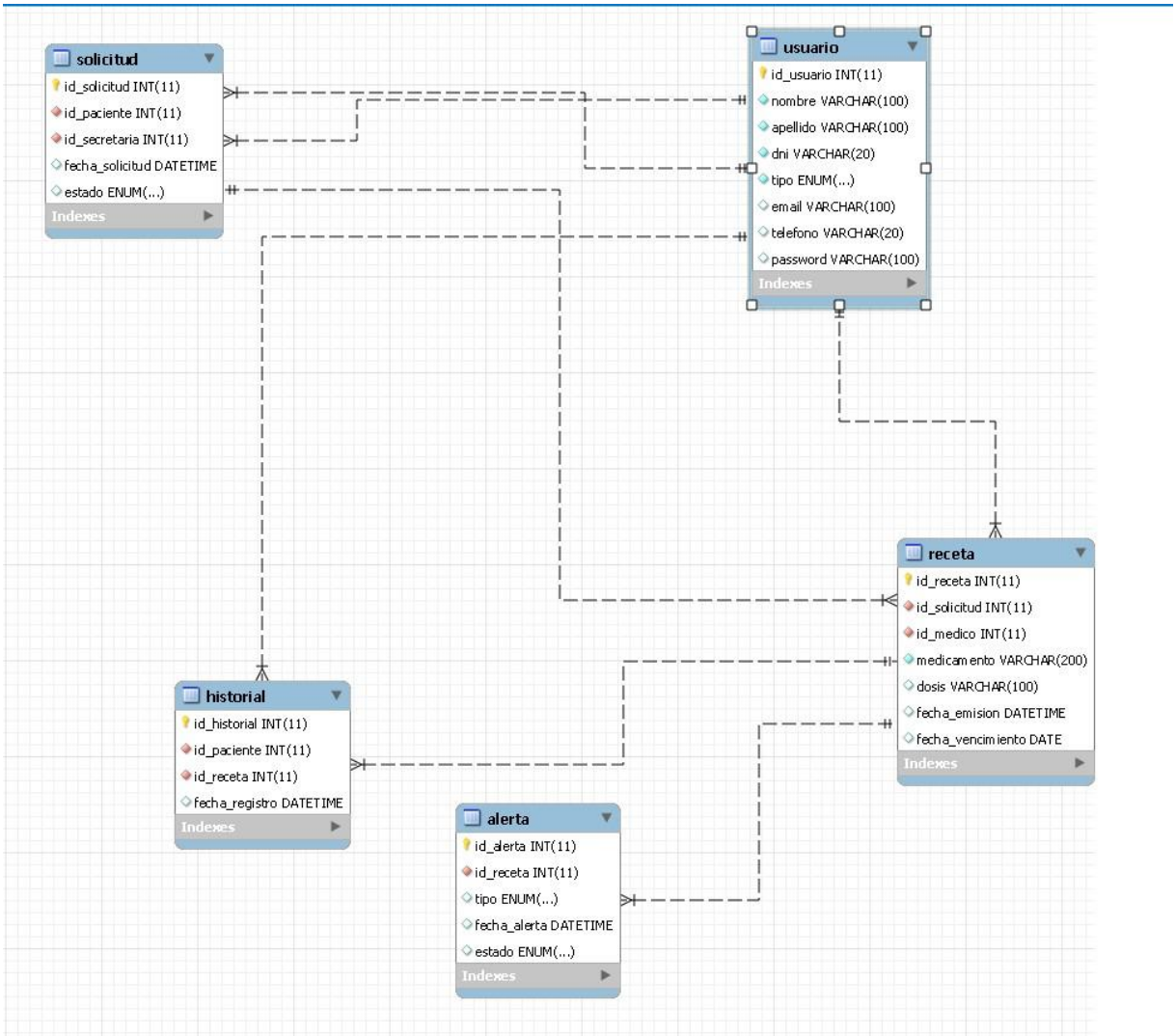
La solicitud válida se guarda correctamente.

Se rechazan solicitudes con paciente inexistente Paciente no encontrado.

No se permite registrar sin medicamento Medicamento requerido.

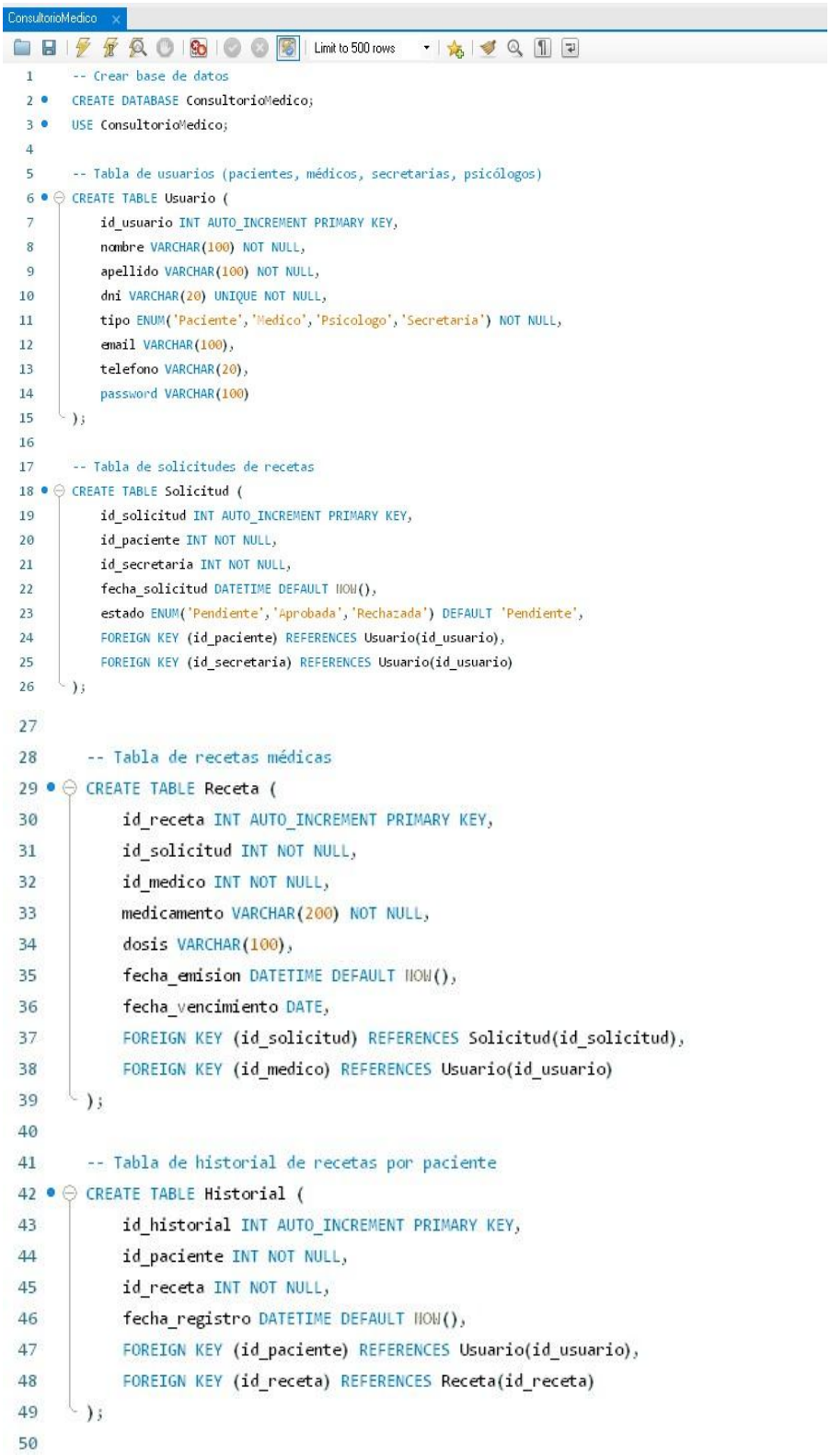
Se rechazan fechas incorrectas Formato inválido.

Diagrama Entidad-Relación:



Presentación de Base de Datos:
Creación de tablas, inserción, consulta y borrado de registros.

Creación de la base de datos, junto con la creación de las tablas correspondientes.



ConsultorioMedico

Limit to 500 rows

```sql
-- Tabla de alertas
CREATE TABLE Alerta (
    id_alerta INT AUTO_INCREMENT PRIMARY KEY,
    id_receta INT NOT NULL,
    tipo ENUM('Vencimiento','Otro') DEFAULT 'Vencimiento',
    fecha_alerta DATETIME DEFAULT NOW(),
    estado ENUM('Activa','Atendida') DEFAULT 'Activa',
    FOREIGN KEY (id_receta) REFERENCES Receta(id_receta)
);

-- Insertar usuarios
INSERT INTO Usuario (nombre, apellido, dni, tipo, email, telefono, password) VALUES
('Carlos', 'Pérez', '30123456', 'Medico', 'carlos.perez@consultorio.com', '1123456789', '1234'),
('María', 'López', '40234567', 'Secretaria', 'maria.lopez@consultorio.com', '1122334455', 'abcd'),
('Juan', 'Martínez', '50345678', 'Paciente', 'juan.martinez@gmail.com', '1133445566', 'abcd'),
('Lucía', 'Gómez', '60456789', 'Psicologo', 'lucia.gomez@consultorio.com', '1144556677', 'abcd');

-- Insertar solicitud de receta
INSERT INTO Solicitud (id_paciente, id_secretaria, fecha_solicitud, estado) VALUES
(3, 2, '2025-10-01 10:00:00', 'Pendiente'),
(3, 2, '2025-10-02 11:00:00', 'Aprobada');

-- Insertar recetas
INSERT INTO Receta (id_solicitud, id_medico, medicamento, dosis, fecha_emision, fecha_vencimiento) VALUES
(1, 1, 'Ibuprofeno 600mg', '1 cada 8 horas', '2025-10-01 12:00:00', '2025-10-15'),
(2, 1, 'Paracetamol 500mg', '1 cada 6 horas', '2025-10-02 14:00:00', '2025-10-16');

-- Insertar historial
INSERT INTO Historial (id_paciente, id_receta, fecha_registro) VALUES
(3, 1, '2025-10-01 12:05:00'),
(3, 2, '2025-10-02 14:10:00');

-- Insertar alertas
INSERT INTO Alerta (id_receta, tipo, fecha_alerta, estado) VALUES
(1, 'Vencimiento', '2025-10-13', 'Activa'),
(2, 'Vencimiento', '2025-10-14', 'Activa');

-- Consultas de ejemplo
SELECT * FROM Usuario WHERE tipo = 'Medico';
SELECT * FROM Solicitud WHERE estado = 'Pendiente';
SELECT * FROM Receta WHERE id_medico = 1;
SELECT * FROM Historial WHERE id_paciente = 3;
SELECT * FROM Alerta WHERE estado = 'Activa';
```

Resultado de una de las consultas.

```
93 •  SELECT * FROM Alerta WHERE estado = 'Activa';
94
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wra

id_alerta	id_receta	tipo	fecha_alerta	estado
1	1	Vencimiento	2025-10-13 00:00:00	Activa
2	2	Vencimiento	2025-10-14 00:00:00	Activa
NULL	NULL	NULL	NULL	NULL

Comunicaciones:

El sistema de Gestión de Recetas Médicas requiere una comunicación eficiente y segura entre los distintos módulos internos “cliente, servidor web, base de datos” y los servicios externos como: nube, APIs.

Comunicación general:

Los usuarios (pacientes, médicos y secretarias) acceden al sistema mediante un navegador web o aplicación móvil.

Las solicitudes se envían al servidor web, que procesa la información y se comunica con la base de datos.

La base de datos gestiona tablas como *Pacientes, Solicitudes, Medicamentos, Recetas y Estado*.

Los datos se respaldan periódicamente en servidores en la nube (Google Drive o SharePoint).

Protocolos y estándares:

HTTP/HTTPS: para la comunicación entre el cliente y el servidor.

API REST: para la integración con servicios externos (por ejemplo, envío de mensajes mediante WhatsApp Business API).

JDBC/ODBC: para la conexión entre la aplicación web y la base de datos.

TLS/SSL: para el cifrado de la información en tránsito.

Infraestructura y entorno de red

Servidor Web: desarrollado en Java o Python, alojado en un entorno seguro.

Servidor de Base de Datos: motor MySQL o PostgreSQL con acceso controlado.

Nube: utilizada para respaldo automático y acceso remoto seguro.

Red interna: conexión LAN del consultorio con acceso Wi-Fi protegido (WPA2 o superior).

Control y seguridad de enlace:
Autenticación con usuario y contraseña.

Cifrado de datos personales y médicos.

Control de transacciones para evitar inconsistencias.

Copias de seguridad automáticas.

Comunicación con sistemas externos.

Integración con servicios de mensajería o notificaciones (por ejemplo, WhatsApp o correo electrónico.

Posibilidad de conexión con sistemas de facturación o gestión hospitalaria mediante API REST.

Explicación del desarrollo en Java:

El desarrollo se presenta mediante menús interactivos en consola, que permiten al usuario elegir las acciones que desea realizar de manera sencilla y clara. Cada tipo de usuario tiene su propio menú:

Secretaria: puede registrar nuevas solicitudes, listar las pendientes y cambiar el estado de una solicitud.

Médica: puede confeccionar recetas y ver solicitudes en proceso.

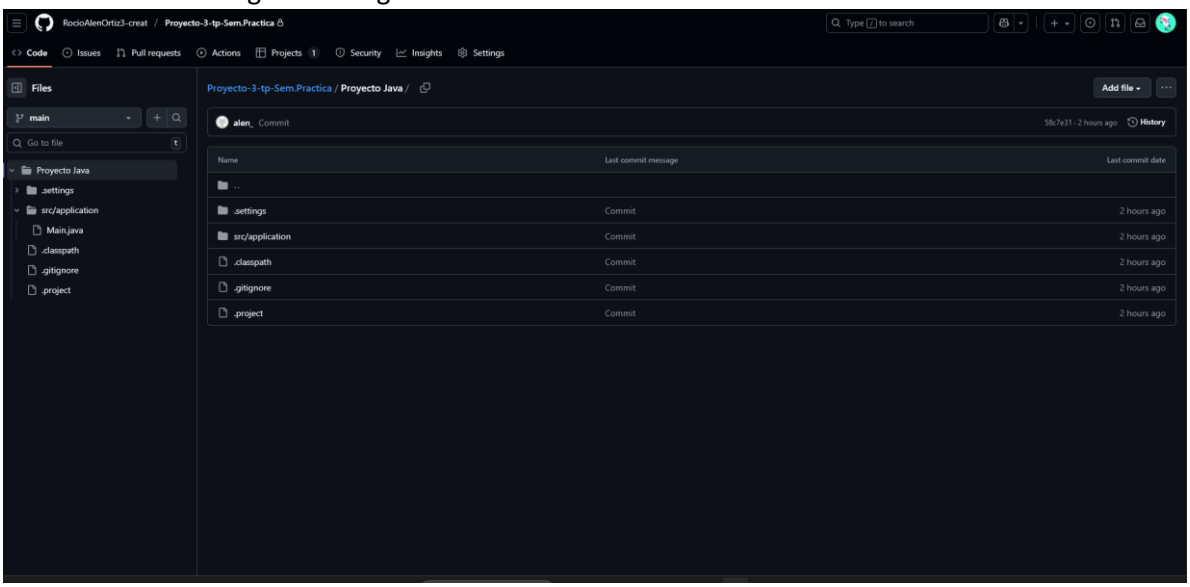
Consultas: permite buscar solicitudes por DNI de un paciente o listar todas las solicitudes ordenadas por fecha.

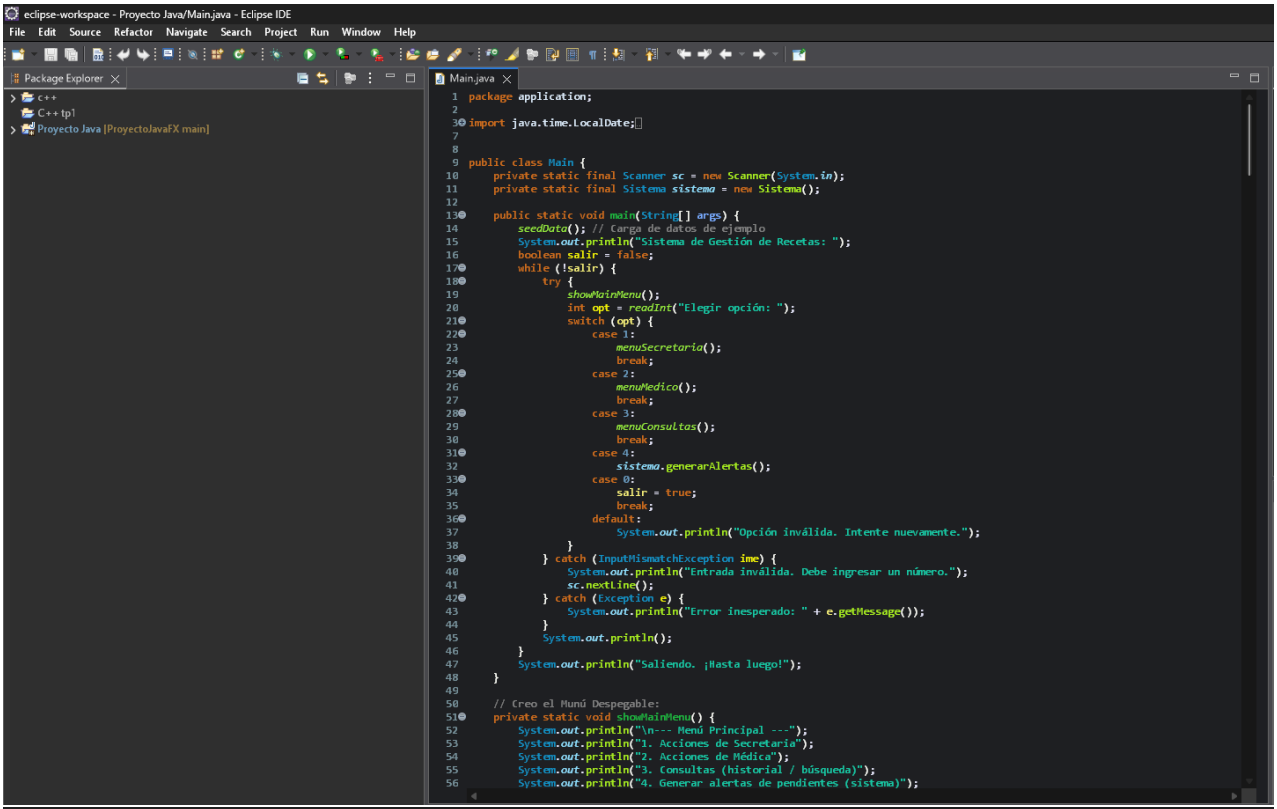
Al seleccionar una opción, el programa ejecuta la acción correspondiente y muestra los resultados en pantalla, permitiendo ver en tiempo real cómo se registran y procesan las solicitudes, cómo se generan recetas y cómo se actualizan los estados.

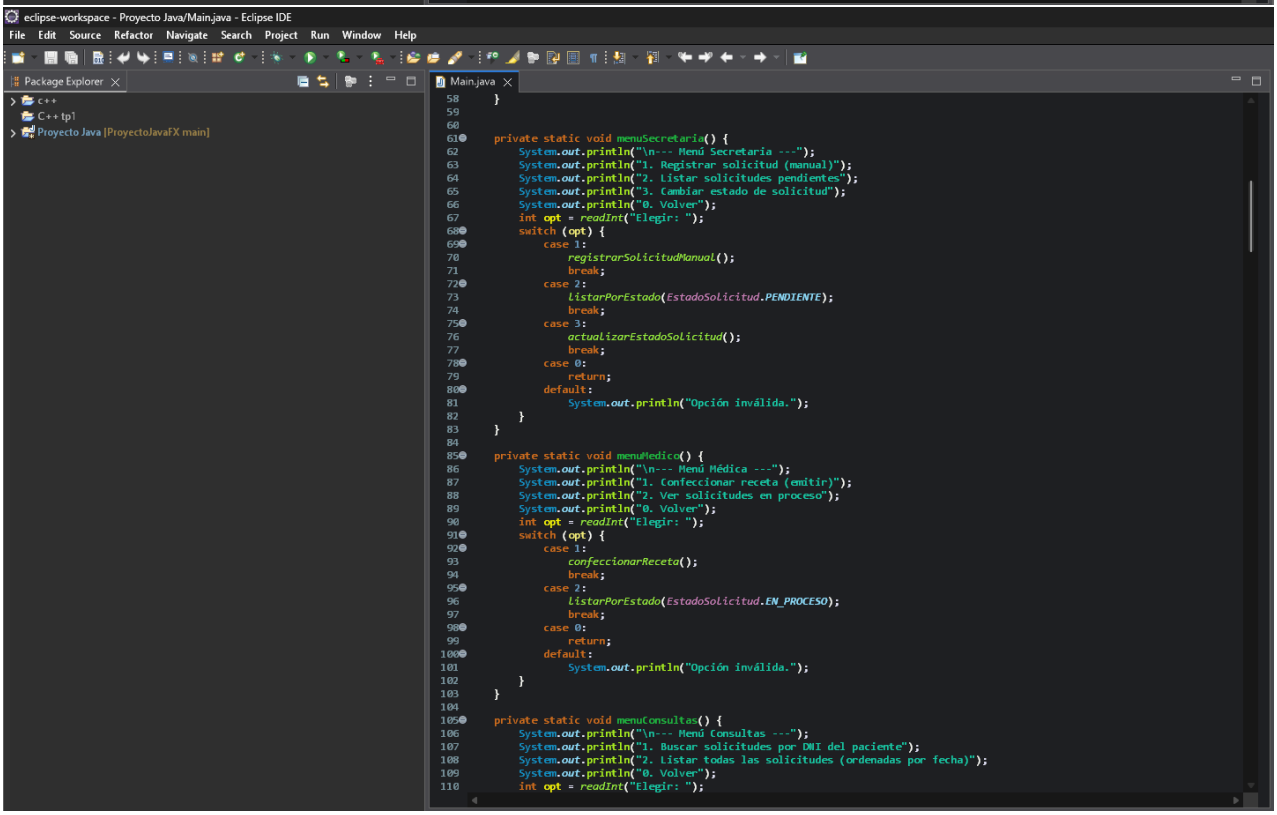
Además, el sistema incluye mensajes claros que guían al usuario durante toda la interacción, validando entradas y mostrando información relevante para garantizar que el uso sea intuitivo y entendible.

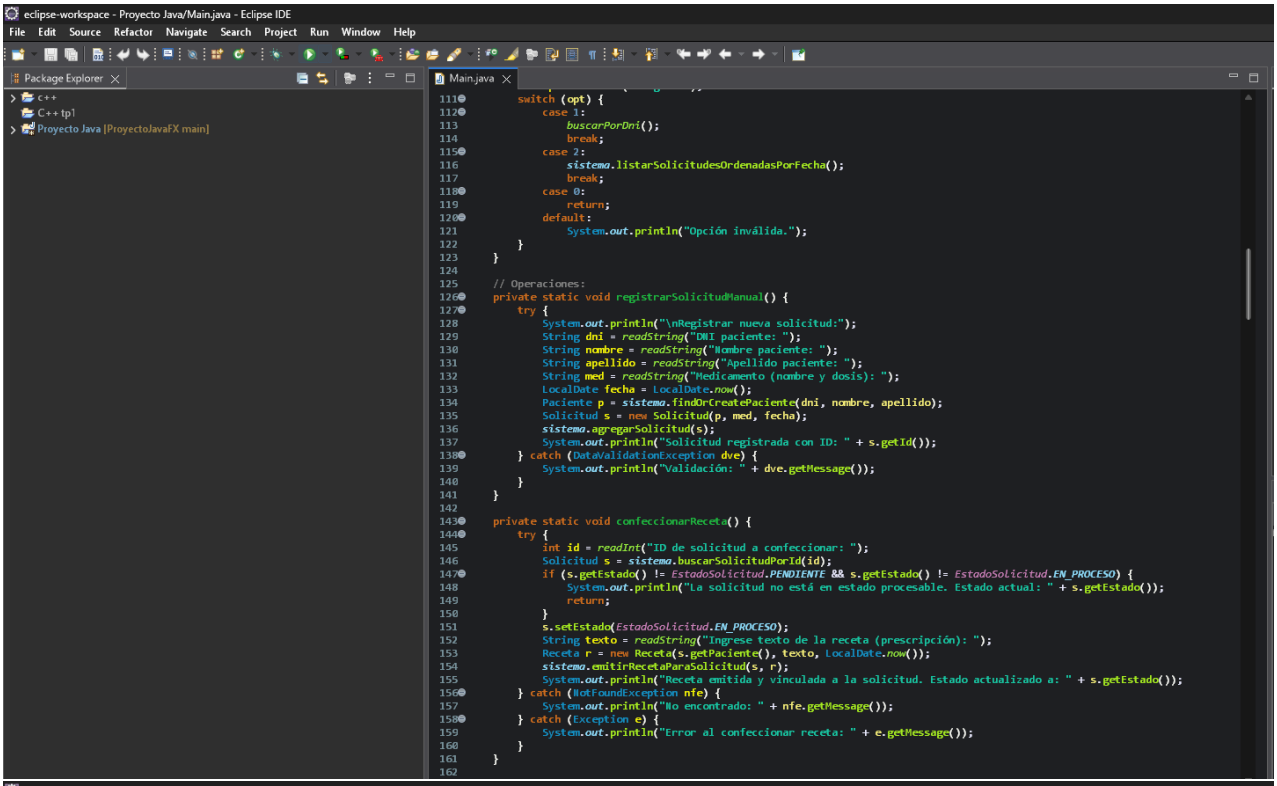
Presentación del desarrollo en Java:
Se presenta con menús interactivos por consola. Cada usuario (secretaria, médico o consultas) tiene su propio menú. Al elegir una opción, el programa ejecuta la acción correspondiente y muestra el resultado en pantalla, como listar solicitudes, generar recetas o buscar por DNI. Esto permite ver claramente cómo funciona el sistema y de forma facil.

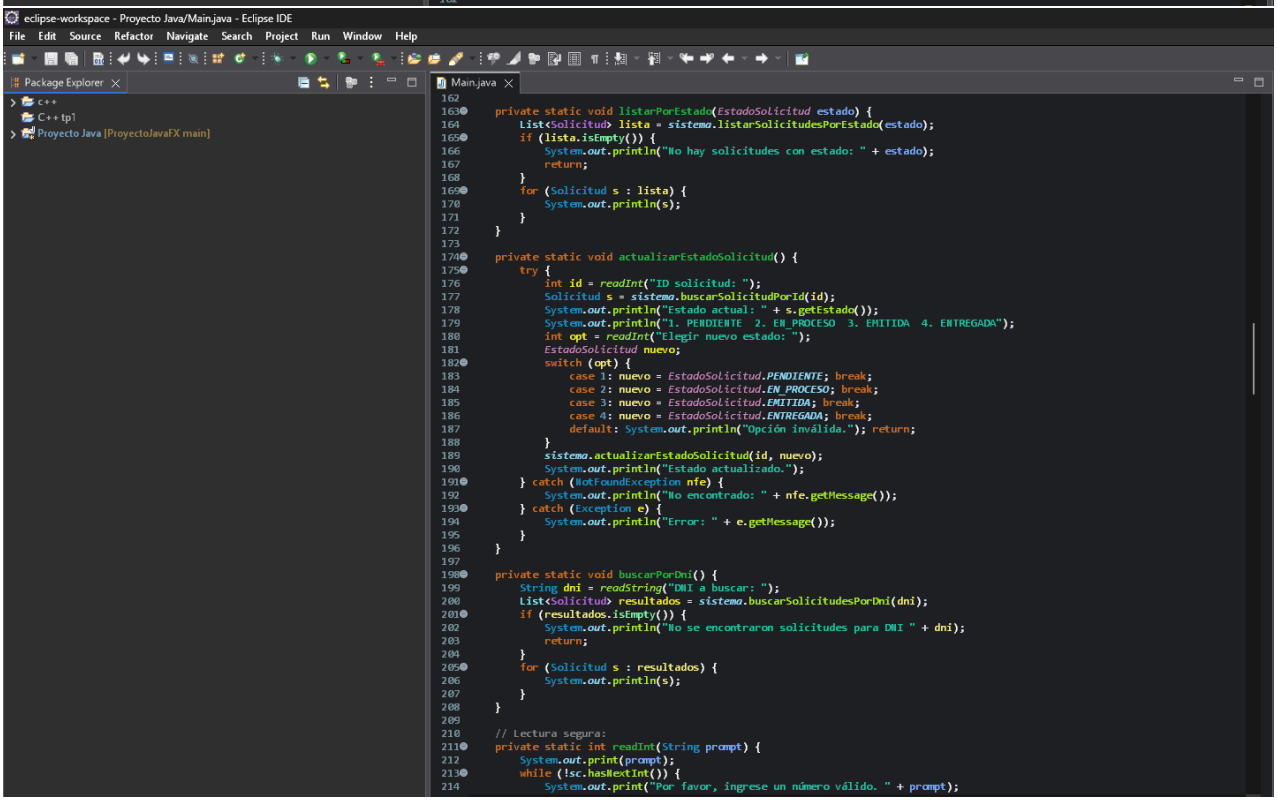
Al código lo realice mediante la aplicación Eclipse y le instale la extensión JavaFX y empecé a codear.
Por ultimo subi a Gitgub el código:

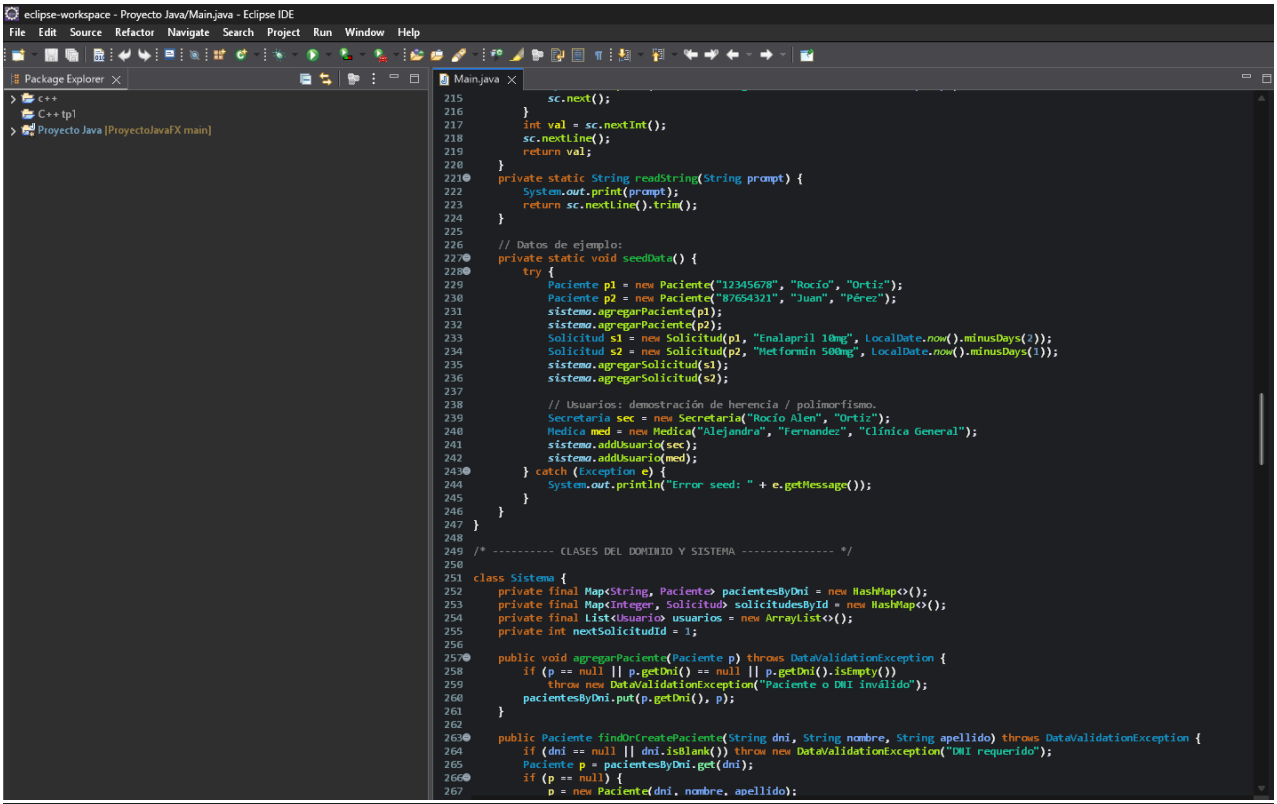


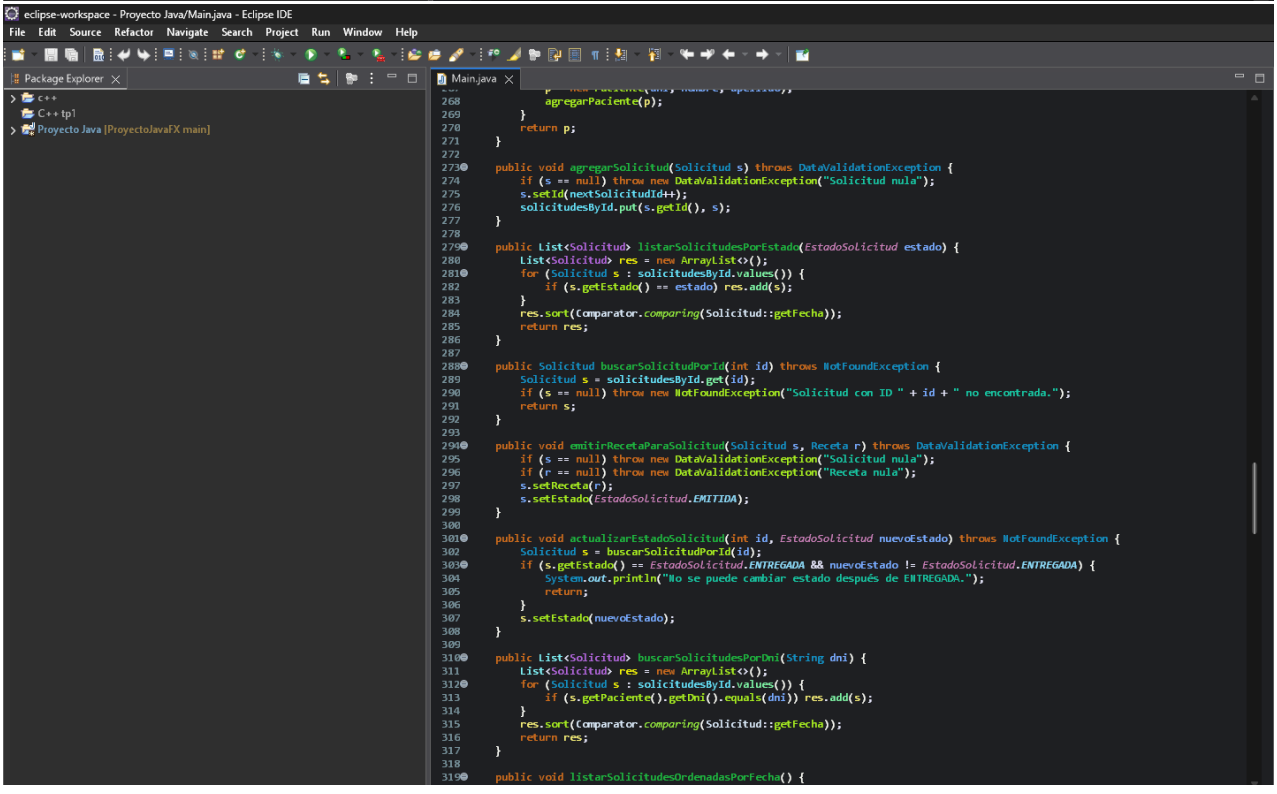

eclipse-workspace - Proyecto Java/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
c++
C++ tp1
Proyecto Java [ProyectoJavaFX main]
Main.java
package application;
import java.time.LocalDate;
public class Main {
private static final Scanner sc = new Scanner(System.in);
private static final Sistema sistema = new Sistema();
public static void main(String[] args) {
seedData(); // Carga de datos de ejemplo
System.out.println("Sistema de Gestión de Recetas: ");
boolean salir = false;
while (!salir) {
try {
showMainMenu();
int opt = readInt("Elegir opción: ");
switch (opt) {
case 1:
menuSecretaria();
break;
case 2:
menuMedico();
break;
case 3:
menuConsultas();
break;
case 4:
sistema.generarAlertas();
case 0:
salir = true;
break;
default:
System.out.println("Opción inválida. Intente nuevamente.");
}
} catch (InputMismatchException ime) {
System.out.println("Entrada inválida. Debe ingresar un número.");
sc.nextLine();
} catch (Exception e) {
System.out.println("Error inesperado: " + e.getMessage());
}
System.out.println();
}
System.out.println("Saliendo. ¡Hasta luego!");
}
// Creo el Munú Despegable:
private static void showMainMenu() {
System.out.println("\n--- Menú Principal ---");
System.out.println("1. Acciones de Secretaria");
System.out.println("2. Acciones de Médica");
System.out.println("3. Consultas (historial / búsqueda)");
System.out.println("4. Generar alertas de pendientes (sistema)");

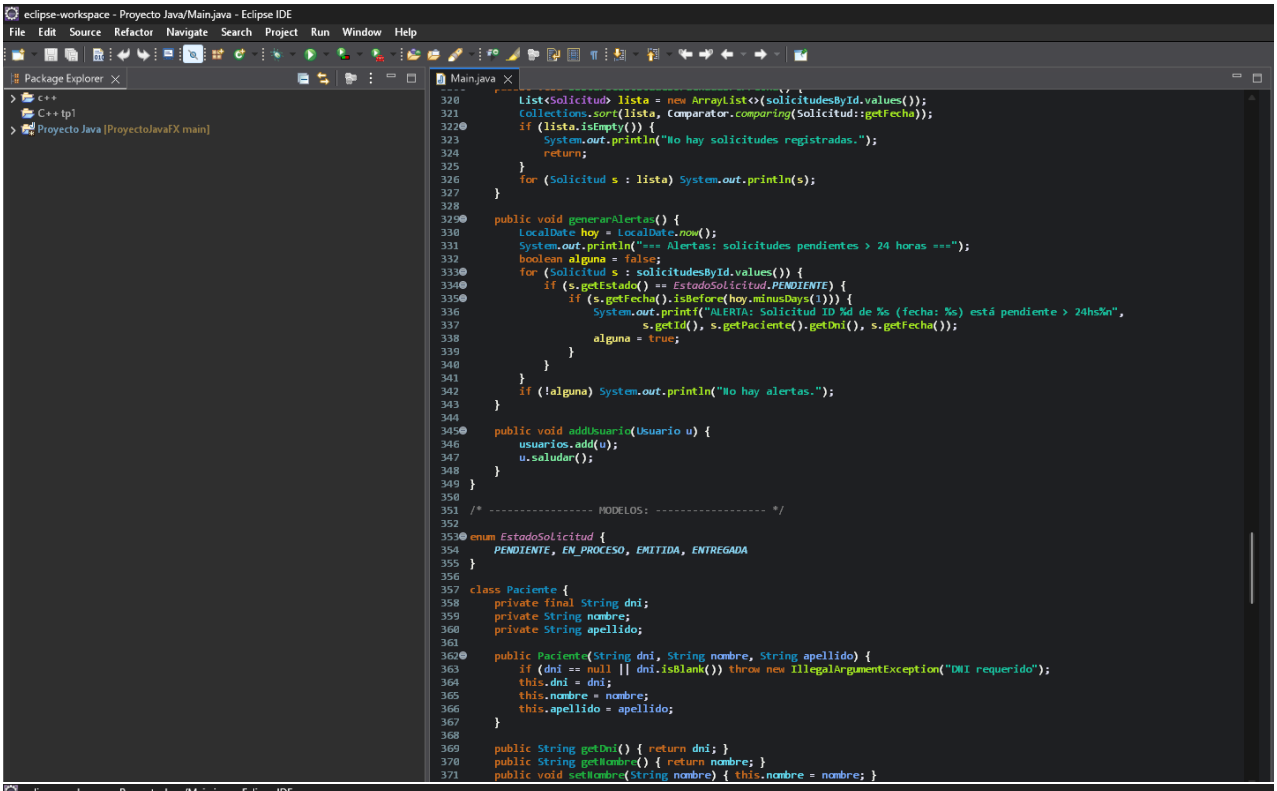

eclipse-workspace - Proyecto Java/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
c++
C++ tp1
Proyecto Java [ProyectoJavaFX main]
Main.java
}
private static void menuSecretaria() {
System.out.println("\n--- Menú Secretaria ---");
System.out.println("1. Registrar solicitud (manual)");
System.out.println("2. Listar solicitudes pendientes");
System.out.println("3. Cambiar estado de solicitud");
System.out.println("0. Volver");
int opt = readInt("Elegir: ");
switch (opt) {
case 1:
registrarSolicitudManual();
break;
case 2:
listarPorEstado(EstadoSolicitud.PENDIENTE);
break;
case 3:
actualizarEstadoSolicitud();
break;
case 0:
return;
default:
System.out.println("Opción inválida.");
}
}
private static void menuMedico() {
System.out.println("\n--- Menú Médica ---");
System.out.println("1. Confeccionar receta (emitir)");
System.out.println("2. Ver solicitudes en proceso");
System.out.println("0. Volver");
int opt = readInt("Elegir: ");
switch (opt) {
case 1:
confeccionarReceta();
break;
case 2:
listarPorEstado(EstadoSolicitud.EN_PROCESO);
break;
case 0:
return;
default:
System.out.println("Opción inválida.");
}
}
private static void menuConsultas() {
System.out.println("\n--- Menú Consultas ---");
System.out.println("1. Buscar solicitudes por DNI del paciente");
System.out.println("2. Listar todas las solicitudes (ordenadas por fecha)");
System.out.println("0. Volver");
int opt = readInt("Elegir: ");

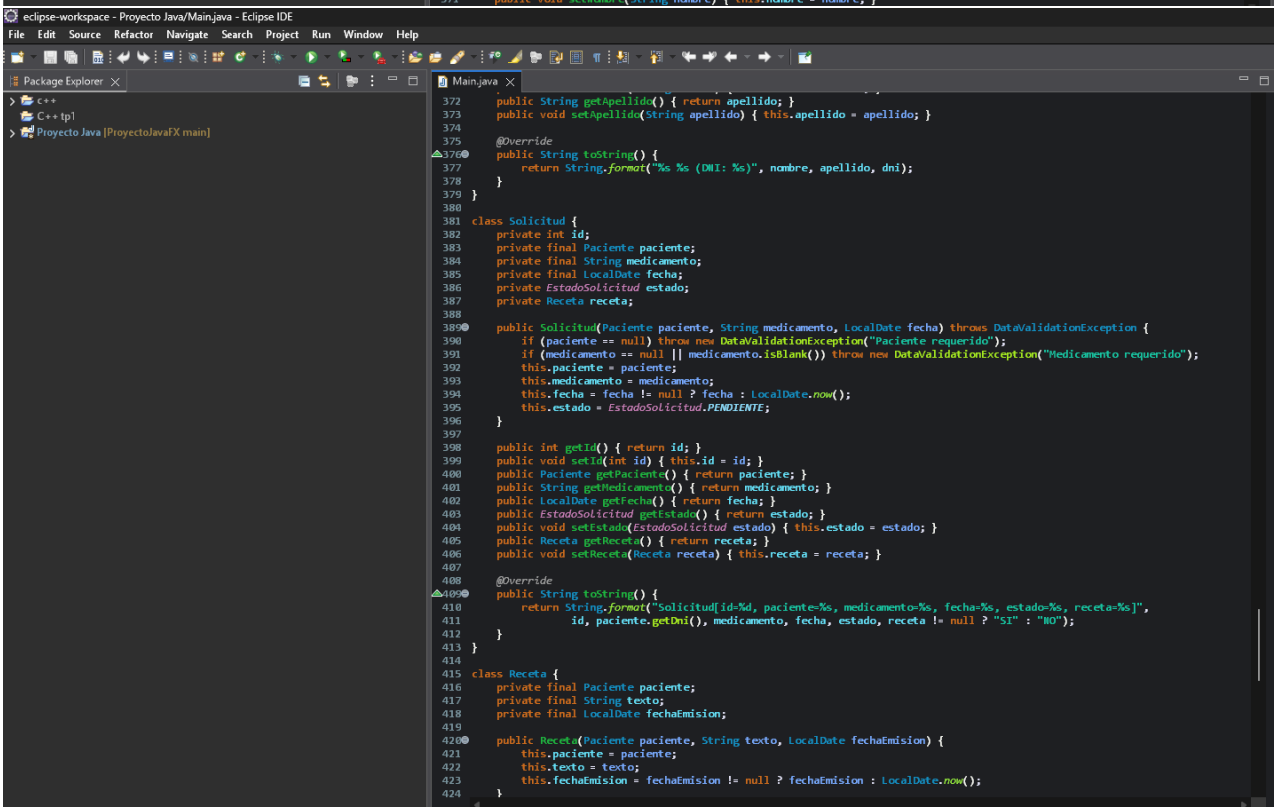

eclipse-workspace - Proyecto Java/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
c++
C++ tp1
Proyecto Java [ProyectoJavaFX main]
Main.java
111 switch (opt) {
112 case 1:
113 buscarPorDni();
114 break;
115 case 2:
116 sistema.listarSolicitudesOrdenadasPorFecha();
117 break;
118 case 0:
119 return;
120 default:
121 System.out.println("Opción inválida.");
122 }
123 }
124
125 // Operaciones:
126 private static void registrarSolicitudManual() {
127 try {
128 System.out.println("\nRegistrar nueva solicitud:");
129 String dni = readString("DNI paciente: ");
130 String nombre = readString("Nombre paciente: ");
131 String apellido = readString("Apellido paciente: ");
132 String med = readString("Medicamento (nombre y dosis): ");
133 LocalDate fecha = LocalDate.now();
134 Paciente p = sistema.findOrCreatePaciente(dni, nombre, apellido);
135 Solicitud s = new Solicitud(p, med, fecha);
136 sistema.agregarSolicitud(s);
137 System.out.println("Solicitud registrada con ID: " + s.getId());
138 } catch (DataValidationException dve) {
139 System.out.println("Validación: " + dve.getMessage());
140 }
141 }
142
143 private static void confeccionarReceta() {
144 try {
145 int id = readInt("ID de solicitud a confeccionar: ");
146 Solicitud s = sistema.buscarSolicitudPorId(id);
147 if (s.getEstado() != EstadoSolicitud.PENDIENTE && s.getEstado() != EstadoSolicitud.EN_PROCESO) {
148 System.out.println("La solicitud no está en estado procesable. Estado actual: " + s.getEstado());
149 return;
150 }
151 s.setEstado(EstadoSolicitud.EN_PROCESO);
152 String texto = readString("Ingrese texto de la receta (prescripción): ");
153 Receta r = new Receta(s.getPaciente(), texto, LocalDate.now());
154 sistema.emitirRecetaParaSolicitud(s, r);
155 System.out.println("Receta emitida y vinculada a la solicitud. Estado actualizado a: " + s.getEstado());
156 } catch (NotFoundException nfe) {
157 System.out.println("No encontrado: " + nfe.getMessage());
158 } catch (Exception e) {
159 System.out.println("Error al confeccionar receta: " + e.getMessage());
160 }
161 }
162


eclipse-workspace - Proyecto Java/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
c++
C++ tp1
Proyecto Java [ProyectoJavaFX main]
Main.java
162
163 private static void listarPorEstado(EstadoSolicitud estado) {
164 List<Solicitud> lista = sistema.listarSolicitudesPorEstado(estado);
165 if (lista.isEmpty()) {
166 System.out.println("No hay solicitudes con estado: " + estado);
167 return;
168 }
169 for (Solicitud s : lista) {
170 System.out.println(s);
171 }
172 }
173
174 private static void actualizarEstadoSolicitud() {
175 try {
176 int id = readInt("ID solicitud: ");
177 Solicitud s = sistema.buscarSolicitudPorId(id);
178 System.out.println("Estado actual: " + s.getEstado());
179 System.out.println("1. PENDIENTE 2. EN_PROCESO 3. EMITIDA 4. ENTREGADA");
180 int opt = readInt("Elegir nuevo estado: ");
181 EstadoSolicitud nuevo;
182 switch (opt) {
183 case 1: nuevo = EstadoSolicitud.PENDIENTE; break;
184 case 2: nuevo = EstadoSolicitud.EN_PROCESO; break;
185 case 3: nuevo = EstadoSolicitud.EMITIDA; break;
186 case 4: nuevo = EstadoSolicitud.ENTREGADA; break;
187 default: System.out.println("Opción inválida."); return;
188 }
189 sistema.actualizarEstadoSolicitud(id, nuevo);
190 System.out.println("Estado actualizado.");
191 } catch (NotFoundException nfe) {
192 System.out.println("No encontrado: " + nfe.getMessage());
193 } catch (Exception e) {
194 System.out.println("Error: " + e.getMessage());
195 }
196 }
197
198 private static void buscarPorDni() {
199 String dni = readString("DNI a buscar: ");
200 List<Solicitud> resultados = sistema.buscarSolicitudesPorDni(dni);
201 if (resultados.isEmpty()) {
202 System.out.println("No se encontraron solicitudes para DNI " + dni);
203 return;
204 }
205 for (Solicitud s : resultados) {
206 System.out.println(s);
207 }
208 }
209
210 // Lectura segura:
211 private static int readInt(String prompt) {
212 System.out.print(prompt);
213 while (!sc.hasNextInt()) {
214 System.out.print("Por favor, ingrese un número válido. " + prompt);


eclipse-workspace - Proyecto Java/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
c++
C++ tp1
Proyecto Java [ProyectoJavaFX main]
Main.java
sc.next();
}
int val = sc.nextInt();
sc.nextLine();
return val;
}
private static String readString(String prompt) {
System.out.print(prompt);
return sc.nextLine().trim();
}
// Datos de ejemplo:
private static void seedData() {
try {
Paciente p1 = new Paciente("12345678", "Rocío", "Ortiz");
Paciente p2 = new Paciente("87654321", "Juan", "Pérez");
sistema.agregarPaciente(p1);
sistema.agregarPaciente(p2);
Solicitud s1 = new Solicitud(p1, "Enalapril 10mg", LocalDate.now().minusDays(2));
Solicitud s2 = new Solicitud(p2, "Metformin 500mg", LocalDate.now().minusDays(1));
sistema.agregarSolicitud(s1);
sistema.agregarSolicitud(s2);
// Usuarios: demostración de herencia / polimorfismo.
Secretaria sec = new Secretaria("Rocío Alen", "Ortiz");
Medica med = new Medica("Alejandra", "Fernandez", "Clínica General");
sistema.addUsuario(sec);
sistema.addUsuario(med);
} catch (Exception e) {
System.out.println("Error seed: " + e.getMessage());
}
}
}
/* ---------- CLASES DEL DOMINIO Y SISTEMA ---------- */
class Sistema {
private final Map<String, Paciente> pacientesByDni = new HashMap<>();
private final Map<Integer, Solicitud> solicitudesById = new HashMap<>();
private final List<Usuario> usuarios = new ArrayList<>();
private int nextSolicitudId = 1;
public void agregarPaciente(Paciente p) throws DataValidationException {
if (p == null || p.getDni() == null || p.getDni().isEmpty())
throw new DataValidationException("Paciente o DNI inválido");
pacientesByDni.put(p.getDni(), p);
}
public Paciente findOrCreatePaciente(String dni, String nombre, String apellido) throws DataValidationException {
if (dni == null || dni.isBlank()) throw new DataValidationException("DNI requerido");
Paciente p = pacientesByDni.get(dni);
if (p == null) {
p = new Paciente(dni, nombre, apellido);


eclipse-workspace - Proyecto Java/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
c++
C++ tp1
Proyecto Java [ProyectoJavaFX main]
Main.java
agregarPaciente(p);
}
return p;
}
public void agregarSolicitud(Solicitud s) throws DataValidationException {
if (s == null) throw new DataValidationException("Solicitud nula");
s.setId(nextSolicitudId++);
solicitudesById.put(s.getId(), s);
}
public List<Solicitud> listarSolicitudesPorEstado(EstadoSolicitud estado) {
List<Solicitud> res = new ArrayList<>();
for (Solicitud s : solicitudesById.values()) {
if (s.getEstado() == estado) res.add(s);
}
res.sort(Comparator.comparing(Solicitud::getFecha));
return res;
}
public Solicitud buscarSolicitudPorId(int id) throws NotFoundException {
Solicitud s = solicitudesById.get(id);
if (s == null) throw new NotFoundException("Solicitud con ID " + id + " no encontrada.");
return s;
}
public void emitirRecetaParaSolicitud(Solicitud s, Receta r) throws DataValidationException {
if (s == null) throw new DataValidationException("Solicitud nula");
if (r == null) throw new DataValidationException("Receta nula");
s.setReceta(r);
s.setEstado(EstadoSolicitud.EMITIDA);
}
public void actualizarEstadoSolicitud(int id, EstadoSolicitud nuevoEstado) throws NotFoundException {
Solicitud s = buscarSolicitudPorId(id);
if (s.getEstado() == EstadoSolicitud.ENTREGADA && nuevoEstado != EstadoSolicitud.ENTREGADA) {
System.out.println("No se puede cambiar estado después de ENTREGADA.");
return;
}
s.setEstado(nuevoEstado);
}
public List<Solicitud> buscarSolicitudesPorDni(String dni) {
List<Solicitud> res = new ArrayList<>();
for (Solicitud s : solicitudesById.values()) {
if (s.getPaciente().getDni().equals(dni)) res.add(s);
}
res.sort(Comparator.comparing(Solicitud::getFecha));
return res;
}
public void listarSolicitudesOrdenadasPorFecha() {


eclipse-workspace - Proyecto Java/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
c++
C++ tp1
Proyecto Java [ProyectoJavaFX main]
Main.java
List<Solicitud> lista = new ArrayList<>(solicitudesById.values());
Collections.sort(lista, Comparator.comparing(Solicitud::getFecha));
if (lista.isEmpty()) {
System.out.println("No hay solicitudes registradas.");
return;
}
for (Solicitud s : lista) System.out.println(s);
}
public void generarAlertas() {
LocalDate hoy = LocalDate.now();
System.out.println("=== Alertas: solicitudes pendientes > 24 horas ===");
boolean alguna = false;
for (Solicitud s : solicitudesById.values()) {
if (s.getEstado() == EstadoSolicitud.PENDIENTE) {
if (s.getFecha().isBefore(hoy.minusDays(1))) {
System.out.printf("ALERTA: Solicitud ID %d de %s (fecha: %s) está pendiente > 24hs%n",
s.getId(), s.getPaciente().getDni(), s.getFecha());
alguna = true;
}
}
}
if (!alguna) System.out.println("No hay alertas.");
}
public void addUsuario(Usuario u) {
usuarios.add(u);
u.saludar();
}
}
/* ---------- MODELOS: ---------- */
enum EstadoSolicitud {
PENDIENTE, EN_PROCESO, EMITIDA, ENTREGADA
}
class Paciente {
private final String dni;
private String nombre;
private String apellido;
public Paciente(String dni, String nombre, String apellido) {
if (dni == null || dni.isBlank()) throw new IllegalArgumentException("DNI requerido");
this.dni = dni;
this.nombre = nombre;
this.apellido = apellido;
}
public String getDni() { return dni; }
public String getNombre() { return nombre; }
public void setNombre(String nombre) { this.nombre = nombre; }


eclipse-workspace - Proyecto Java/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
c++
C++ tp1
Proyecto Java [ProyectoJavaFX main]
Main.java
public String getApellido() { return apellido; }
public void setApellido(String apellido) { this.apellido = apellido; }
@Override
public String toString() {
return String.format("%s %s (DNI: %s)", nombre, apellido, dni);
}
}
class Solicitud {
private int id;
private final Paciente paciente;
private final String medicamento;
private final LocalDate fecha;
private EstadoSolicitud estado;
private Receta receta;
public Solicitud(Paciente paciente, String medicamento, LocalDate fecha) throws DataValidationException {
if (paciente == null) throw new DataValidationException("Paciente requerido");
if (medicamento == null || medicamento.isBlank()) throw new DataValidationException("Medicamento requerido");
this.paciente = paciente;
this.medicamento = medicamento;
this.fecha = fecha != null ? fecha : LocalDate.now();
this.estado = EstadoSolicitud.PENDIENTE;
}
public int getId() { return id; }
public void setId(int id) { this.id = id; }
public Paciente getPaciente() { return paciente; }
public String getMedicamento() { return medicamento; }
public LocalDate getFecha() { return fecha; }
public EstadoSolicitud getEstado() { return estado; }
public void setEstado(EstadoSolicitud estado) { this.estado = estado; }
public Receta getReceta() { return receta; }
public void setReceta(Receta receta) { this.receta = receta; }
@Override
public String toString() {
return String.format("Solicitud[id=%d, paciente=%s, medicamento=%s, fecha=%s, estado=%s, receta=%s]",
id, paciente.getDni(), medicamento, fecha, estado, receta != null ? "SI" : "NO");
}
}
class Receta {
private final Paciente paciente;
private final String texto;
private final LocalDate fechaEmision;
public Receta(Paciente paciente, String texto, LocalDate fechaEmision) {
this.paciente = paciente;
this.texto = texto;
this.fechaEmision = fechaEmision != null ? fechaEmision : LocalDate.now();
}

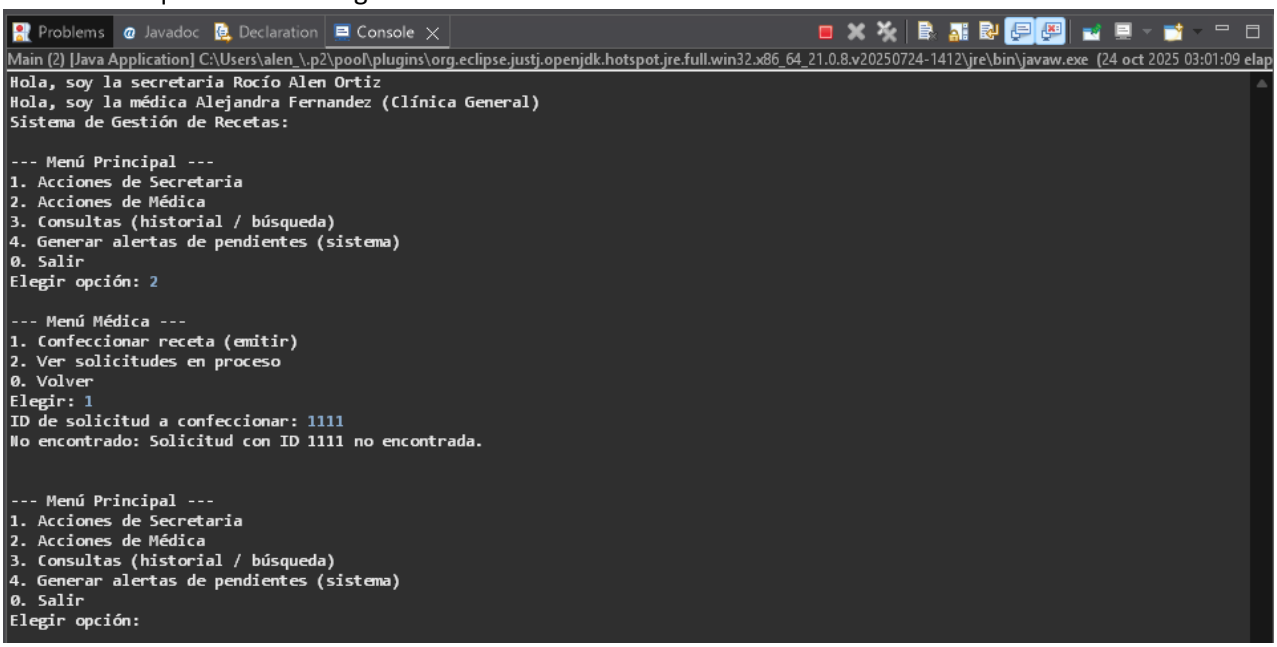
```java
    public String getEspecialidad() { return especialidad; }
    public void setEspecialidad(String esp) { this.especialidad = esp; }
}

/* ---------------- Excepciones ------------- */
class DataValidationException extends Exception {
    private static final long serialVersionUID = 1L;
    public DataValidationException(String msg) {
        super(msg);
    }
}

class NotFoundException extends Exception {
    private static final long serialVersionUID = 1L;
    public NotFoundException(String msg) {
        super(msg);
    }
}

```

El resultado que dio son las siguientes:



Explicación del código mediante un cuadro para organizarme mejor:

Parte del programa	*Qué hace / Para qué sirve*	*Qué contiene*	*Qué conceptos de programación usa*
Clase Main	Es la clase principal del programa. Desde acá se inicia todo. Muestra los menús, pide opciones al usuario y llama a los métodos correspondientes del sistema.	Tiene un Scanner para leer por consola y un objeto Sistema que maneja toda la lógica. Incluye métodos como main(), showMainMenu(), menuSecretaria(), menuMedico() y otros para registrar solicitudes o confeccionar recetas.	Uso de estructuras de control, lectura por consola y manejo de excepciones.
Clase Sistema	Es el “núcleo” del programa. Se encarga de guardar y	Usa mapas y listas para almacenar los datos (pacientesByDni,	Aplica encapsulamiento, colecciones y métodos

	administrar los datos de pacientes, solicitudes, recetas y usuarios.	solicitudesById, usuarios). Tiene métodos para agregar pacientes, registrar solicitudes, buscar por DNI, emitir recetas, actualizar estados y generar alertas.	bien definidos para dividir responsabilidades.
Enum EstadoSolicitud	Representa los diferentes estados por los que puede pasar una solicitud médica.	Incluye los valores PENDIENTE, EN_PROCESO, EMITIDA y ENTREGADA.	Uso de enumeraciones para evitar errores y dar claridad al código.
Clase Paciente	Guarda los datos básicos de cada paciente.	Atributos: DNI, nombre y apellido. Tiene getters, setters y un toString() para mostrar la información.	Ejemplo de encapsulamiento y abstracción de una entidad del mundo real.
Clase Solicitud	Representa una solicitud de receta médica que hace un paciente.	Atributos: número de solicitud, paciente, medicamento, fecha, estado y la receta asociada.	Usa composición, porque una solicitud contiene un paciente y puede tener una receta asociada.
Clase Receta	Guarda la receta emitida por el médico, con su texto y la fecha de emisión.	Tiene un paciente, el contenido de la receta y la fecha.	Ejemplo de composición (usa objetos de otras clases) y encapsulamiento.
Clase Usuario (abstracta)	Es la clase base de la que heredan los tipos de usuarios del sistema (médico y secretaria).	Atributos: nombre y apellido. Tiene un método abstracto saludar() que las subclases deben implementar.	Aplica abstracción y herencia. No se puede crear un usuario directamente, solo sus hijos.
Clase Secretaria	Representa a la secretaria del sistema, encargada de las tareas administrativas.	Hereda los datos de Usuario e implementa el método saludar().	Usa herencia y polimorfismo (porque redefine el saludo a su manera).
Clase Medica	Representa a la médica que confecciona las recetas.	Hereda de Usuario e incluye el atributo especialidad. También redefine saludar().	Aplica herencia, polimorfismo y encapsulamiento.
Excepción DataValidationException	Se usa para marcar errores cuando los datos ingresados no son válidos.	Hereda de Exception y recibe un mensaje con el error.	Ejemplo de manejo de errores personalizado.
Excepción NotFoundException	Se lanza cuando se busca algo (como una solicitud o un	También hereda de Exception y recibe un mensaje explicativo.	Otro ejemplo de excepciones personalizadas.

	paciente) que no existe.		

Concepto	¿Dónde lo vemos?	Ejemplo dentro del código
Encapsulamiento	En las clases Paciente, Medica, Solicitud, Receta (atributos privados y métodos para acceder a ellos).	private String nombre; public String getNombre()
Herencia	Las clases Medica y Secretaria heredan de Usuario.	class Medica extends Usuario
Polimorfismo	Se usa cuando las clases hijas implementan el método saludar() de diferente forma.	@Override public void saludar()
Abstracción	La clase Usuario es abstracta, define lo común pero no se puede instanciar.	abstract class Usuario
Composición	Una solicitud contiene un paciente y una receta.	private Paciente paciente; private Receta receta;
Enumeración	Para representar los estados posibles de una solicitud.	EstadoSolicitud.PENDIENTE
Manejo de excepciones	Cuando se validan datos o se capturan errores en el menú.	catch (DataValidationException e)
Colecciones	Para almacenar pacientes, usuarios y solicitudes en mapas y listas.	Map<Integer, Solicitud> solicitudesById = new HashMap<>();