UNIVERSIDAD NACIONAL AUTÓNOMA DE HONDURAS

FACULTAD DE CIENCIAS ECONÓMICAS, ADMINISTRATIVAS Y CONTABLES

DEPARTAMENTO DE INFORMÁTICA ADMINISTRATIVA

ASIGNATURA:

Programación Intermedia

CATEDRÁTICO:

Marvin Josue Aguilar Romero

ACTIVIDAD:

Guía de ejercicios UML y POO

GRUPO

Erinson Josué Alvarez Garcia 20211023800

Cristina Lisbeth Cruz Aguilera 20231031044

Kevin Alejandro Laínez Salinas 20231031586

Jonatan Isai Varela Giron 20231000159

Dany Josue Noguera Tinoco 20212020372

FECHA:

27/11/2025

2. INDICE DE CONTENIDO

3. ÍNDICE DE TABLAS

4. ÍNDICE DE FIGURAS

5. TABLA DE CUMPLIMIENTO

Integrante	Porcentaje de Cumplimiento (%)
Kevin Alejandro Laínez Salinas	100%
Jonatan Isai Varela Giron	100%
Cristina Lisbeth Cruz Aguilera	75%
Dany Josue Noguera Tinoco	100%
Erinson Josué Alvarez Garcia	100%

6. INTRODUCCIÓN

El crecimiento de cualquier comunidad religiosa genera una complejidad administrativa crítica, especialmente en lo referente a la organización de sus miembros en grupos de crecimiento y el seguimiento de su desarrollo personal y espiritual. La gestión tradicional de esta información vital (datos de contacto, historial personal, registro de asistencia a reuniones) suele ser manual, descentralizada y altamente susceptible a errores e inconsistencias.

Este proyecto nace de la necesidad de modernizar y centralizar los procesos internos, dotando a la iglesia de una herramienta robusta y segura. El objetivo es transformar la administración pastoral al facilitar una visión clara y estadísticas precisas sobre la participación y el compromiso de cada feligrés.

El sistema es una aplicación de escritorio desarrollada en Java cuyo principal objetivo es funcionar como una plataforma integral de Gestión de Grupos y Seguimiento de Miembros.

Su alcance funcional abarca tres pilares esenciales:

1. Registro y Expedientes: Centraliza la información personal del miembro y permite mantener un historial detallado a través de un Expediente individualizado, forzando una relación uno a uno a nivel de base de datos para garantizar la unicidad del registro.

2. Gestión de Reuniones y Asistencia: Permite a los líderes agendar, finalizar y registrar la asistencia a las reuniones de sus grupos. Se incluye la funcionalidad de Memoria Personal, donde el miembro puede registrar notas de la reunión para su propia reflexión.

3. Control de Seguridad y Reportes: Implementa un estricto Control de Acceso Basado en Roles (RBAC) que restringe la visibilidad de los módulos y las acciones del sistema a los perfiles de Líder de Iglesia, Líder de Grupo y Miembro, asegurando que solo el personal autorizado acceda a la información sensible.

El desarrollo está cimentado en el lenguaje Java utilizando la librería Swing para la interfaz gráfica de usuario (GUI), la cual fue estilizada con FlatLaf para ofrecer una apariencia moderna y profesional.

A nivel interno, la arquitectura sigue rigurosamente un modelo de Capas (MVC/DAO) para lograr una alta cohesión y bajo acoplamiento:

- Modelo y Patrones POO: Se utilizó la herencia a través de la clase abstracta EntidadBase para estandarizar el ID y los métodos de validación en todos los objetos de dominio. La interfaz genérica ICRUD<T> normaliza el acceso a datos. Además, el patrón Singleton se aplicó estratégicamente en ConexionSQLite y ControladorPrincipal para optimizar recursos y centralizar la lógica de negocio.

- Persistencia Mixta: La aplicación cumple con el requisito de persistencia mixta utilizando SQLite 3 como base de datos relacional principal para la información transaccional y un archivo de texto plano (usuarios.txt) administrado por GestorAutenticacion para el registro seguro de las credenciales de acceso.

- Usabilidad y Robustez: El sistema incorpora soporte para múltiples idiomas mediante *Resource Bundles* y utiliza una clase Validador para asegurar la calidad de los datos de entrada, como correos electrónicos y números de teléfono.

7. PROBLEMA

Problema Principal

El problema principal radica en la dificultad de **llevar el control** de los miembros de una congregación que están organizados en grupos de comunidad o ministerios. La gestión y el seguimiento manual de la pertenencia a estos grupos y, consecuentemente, de la **asistencia a sus reuniones**, se convierte en un reto serio a medida que la iglesia crece, haciendo necesaria una **solución informática** para centralizar la organización y el control.

Enunciado del Proyecto (Requisitos Funcionales)

El sistema propuesto pertenece a una **iglesia en particular** y debe gestionar su información central, sus miembros, y la organización en grupos.

1. **Consideraciones Generales**

- La iglesia posee miembros y cada miembro debe tener un expediente.
- La iglesia posee grupos, y un miembro solo puede estar asignado a un único grupo.
- La aplicación debe gestionar la autenticación y los permisos basados en roles.

2. **Funcionalidades del Líder de Iglesia (Pastor)**

Los líderes de la iglesia pueden:

- Editar la información institucional de la iglesia.
- Agregar nuevos miembros a la iglesia, registrando su información de contacto.
- Designar líderes a partir de los miembros existentes.
- Crear grupos, asignándoles un nombre, descripción, miembros y designando un único líder por grupo.
- Ver Estadísticas, incluyendo el total de grupos con su total de miembros y líderes, el total de reuniones por grupo y la asistencia a cada reunión.

3. **Funcionalidades del Líder de Grupo**

Cada líder de grupo debe tener la capacidad de:

- Crear una reunión, indicando sus datos propios y la persona o personas responsables de la misma.
- Finalizar una reunión que no haya sido completada, indicando las inasistencias y redactando una minuta de lo tratado.
- Listar todas las reuniones de su grupo, mostrando primero las que no han finalizado.
- Llevar el Control de Asistencia, viendo cuántos miembros asistieron y cuántos no a cada reunión.

- Ver el Control de Inasistencias por cada miembro, incluyendo el total de faltas, la reunión a la que no asistió y la excusa presentada.

4. **Funcionalidades del Miembro**

Todo miembro del sistema (incluyendo los líderes) debe poder:

- Editar su propio perfil.
- Consultar las reuniones programadas para su grupo.
- Agregar una memoria o nota personal a cada reunión a la que haya asistido.

8. ANÁLISIS DEL PROBLEMA

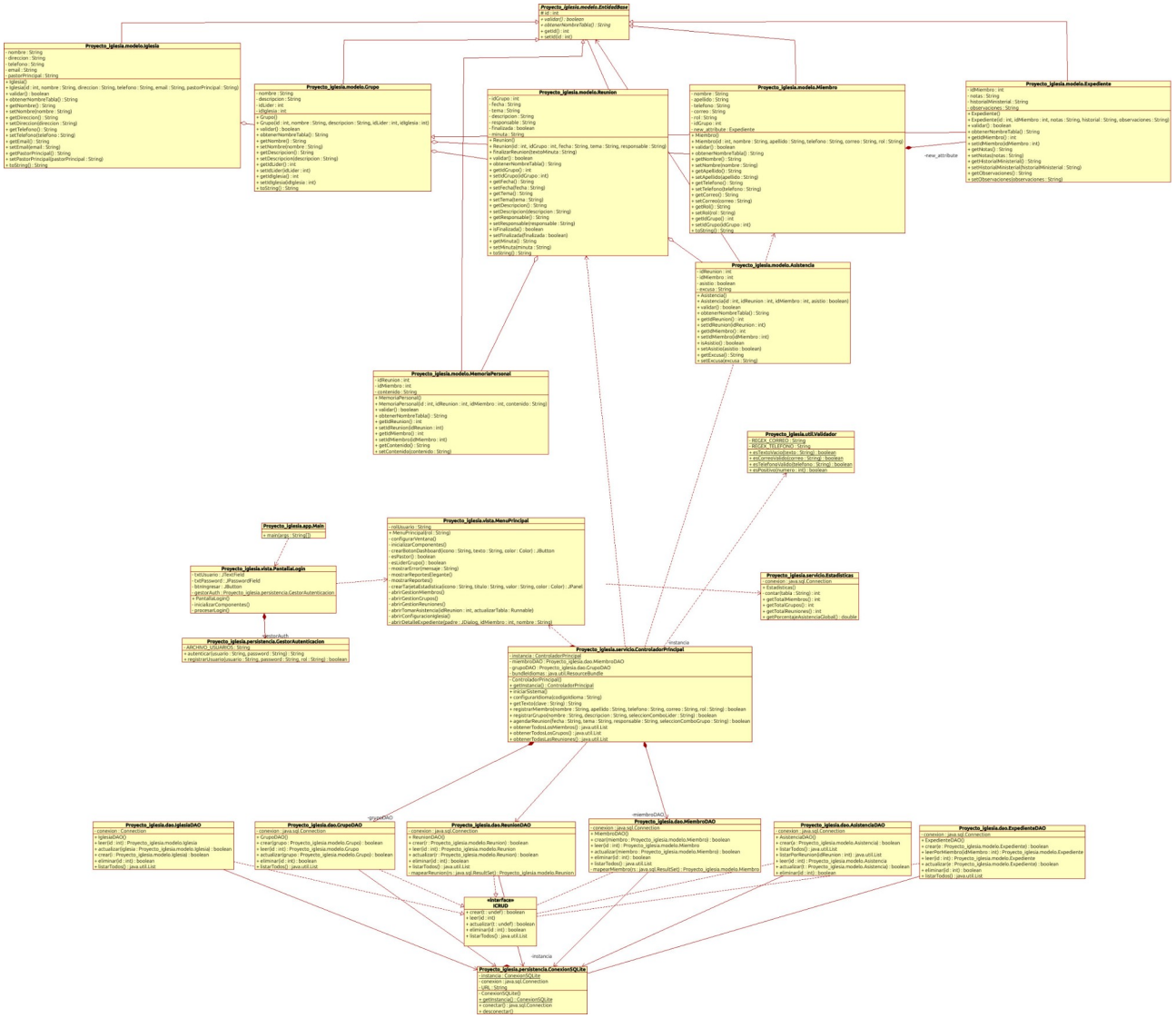

Figura 1. Diagrama de Clases **Fuente:** Elaboración propia

El reto de gestionar una congregación organizada en grupos de crecimiento se aborda mediante la aplicación rigurosa del paradigma POO, descomponiendo los requisitos funcionales en un conjunto de clases, atributos, y relaciones bien definidos. Esta aproximación no solo resuelve el problema logístico de control, sino que también asegura la escalabilidad, fácil mantenimiento, y reutilización del código.

1. El Diseño de la Jerarquía de Objetos (Herencia y Abstracción)

La primera decisión arquitectónica crucial es la implementación de la Herencia para promover la reutilización de código y el polimorfismo, cumpliendo con un requisito estricto del proyecto.

- **Clase Abstracta: EntidadBase**

  Se define como la raíz de la jerarquía de las entidades de la base de datos. Su propósito es centralizar las características comunes:

  - Atributo: id (protegido).

- Comportamiento Abstracto: validar() 1 y obtenerNombreTabla(), obligando a cada clase hija a implementar sus propias reglas para la persistencia y la integridad de los datos.

- **Clases Heredadas (Dominio):**

  Todas las entidades principales del problema (Iglesia, Miembro, Grupo, Reunion, Asistencia, MemoriaPersonal, Expediente) extienden de EntidadBase. Esto garantiza el polimorfismo en el manejo de IDs y validaciones en las capas superiores del sistema.

2. Clases del Dominio y sus Responsabilidades

Las siguientes clases representan los objetos clave del negocio, cada uno con una responsabilidad única (Principio de Responsabilidad Única – SRP)

Tabla 1. Clases del Dominio y sus Responsabilidades

Clase	Responsabilidad Principal	Métodos de Lógica de Negocio	Relación Clave
Miembro	Almacenar datos personales y rol del feligrés.	esLider()	Agregación con Grupo (FK idGrupo)2.
Grupo	Organización y estructura de liderazgo.	validar() (nombre obligatorio)	Agregación con Miembro (FK idLider)3.
Reunion	Gestión del evento y su estado.	finalizarReunion(minuta) que cambia el estado a finalizada.	Agregación con Grupo (FK idGrupo).
Expediente	Historial eclesiástico detallado.	leerPorMiembro() (en su DAO)	1 a 1 con Miembro4.
Asistencia	Registro de la participación.	Ninguno (Data Holder)	Clase asociativa (N:M lógica).

Fuente: Elaboración propia del equipo

3. Descomposición de la Arquitectura (Capas de Infraestructura)

Para reforzar la Separación de Responsabilidades, el sistema se divide en los paquetes modelo, dao, persistencia y servicio:

A. **Capa de Acceso a Datos (dao/)**

Esta capa es el puente entre los objetos Java y la base de datos SQLite.

- Interfaz Genérica: ICRUD<T>: Define el contrato para el acceso a datos (métodos crear, leer, actualizar, eliminar, listarTodos). Al ser genérica, promueve la reutilización y estandarización del código.

- Clases DAO: Cada clase DAO (ej., MiembroDAO, ReunionDAO) implementa ICRUD<T> y contiene exclusivamente las sentencias SQL necesarias. Todas las consultas SQL utilizan PreparedStatement para garantizar la seguridad y prevenir ataques de inyección SQL.

B. **Capa de Servicios y Lógica de Negocio (servicio/)**

Actúa como el corazón del sistema, coordinando las acciones entre la interfaz y la persistencia.

- ControladorPrincipal: Implementado como Singleton para garantizar una única instancia central que maneje el flujo del sistema. Contiene la lógica transaccional de alto nivel, como registrarMiembro o agendarReunion.

- Estadísticas: Se encarga de la lógica compleja de cálculo, como determinar el porcentaje de asistencia global.

C. **Capa de Persistencia y Utilidad (persistencia/ y util/)**

Maneja recursos externos y funciones auxiliares.

- Conexión y Drivers: ConexionSQLite (Singleton) gestiona la conexión JDBC.

- Autenticación y Seguridad: GestorAutenticacion lee el archivo de texto (usuarios.txt) y verifica las credenciales, determinando el rol del usuario (LIDER_IGLESIA, LIDER_GRUPO, MIEMBRO) que será usado para la lógica de seguridad (RBAC) en la vista.

- Validaciones: Validador centraliza la comprobación de formatos de entrada, crucial para la integridad de los datos antes de ser enviados a la base de datos.

Esta descomposición en objetos y capas garantiza que el sistema sea modular, limpio y cumpla con todos los requisitos funcionales y arquitectónicos exigidos.

9. MARCO TEÓRICO

El presente proyecto se fundamenta en un marco teórico riguroso que no solo abarca los pilares del paradigma POO, sino que también integra patrones de diseño avanzados y criterios de robustez a nivel de base de datos, esenciales para un sistema de gestión profesional.

1. Principios de Programación Orientada a Objetos (POO)

El proyecto se desarrolla bajo el lenguaje Java, garantizando la aplicación estricta de la POO.

1.1. Abstracción, Herencia y Polimorfismo

La estructura del dominio se basa en una jerarquía de clases que promueve la reutilización y el diseño modular:

- Clase Abstracta: EntidadBase: Actúa como la raíz del modelo y cumple con el requisito de herencia del curso.
    - Centraliza el atributo protected int id, evitando la duplicación del campo de clave primaria en las siete clases de entidad hijas.
    - Define los métodos abstractos validar() y obtenerNombreTabla(), obligando a cada entidad concreta (Miembro, Reunion, etc.) a implementar sus propias reglas de negocio y a declarar la tabla de persistencia a la que corresponde.
- Interfaz Genérica: ICRUD<T>: Define el contrato de acceso a datos (crear, leer, actualizar, eliminar, listarTodos). El uso del tipo genérico <T> permite la reutilización de la interfaz por todas las clases DAO, logrando un alto grado de polimorfismo y cohesión en la capa de persistencia.

1.2. Encapsulamiento

Todos los atributos de las clases del modelo (Miembro, Grupo, etc.) son declarados como private, exponiendo su acceso únicamente a través de métodos públicos Getters y Setters. Este principio protege la integridad de los datos, previniendo modificaciones no deseadas fuera de las clases de dominio.

2. Patrones de Diseño Arquitectónicos y Lógica de Flujo

2.1. Arquitectura en Capas (MVC y DAO)

El diseño sigue una arquitectura en capas para la separación de responsabilidades (SRP), siendo el patrón Data Access Object (DAO) clave para desacoplar la lógica de negocio de las sentencias SQL:

- Persistencia (DAO): Las clases DAO implementan la interfaz ICRUD<T> y utilizan exclusivamente PreparedStatement para interactuar con SQLite. El uso de PreparedStatement es una medida de seguridad crítica para prevenir la inyección SQL.
- Servicio (Controlador): La clase ControladorPrincipal maneja la lógica de negocio y coordina el flujo. El patrón Singleton se aplica a ConexionSQLite para asegurar que solo exista una instancia de conexión, optimizando el rendimiento y evitando accesos concurrentes al archivo .sqlite.

2.2. Seguridad y Concurrencia

- RBAC y Persistencia Mixta: La seguridad se implementa mediante un Control de Acceso Basado en Roles (RBAC), diferenciando los perfiles de LIDER_IGLESIA, LIDER_GRUPO y MIEMBRO. La autenticación se logra mediante un esquema de persistencia mixta, utilizando SQLite para los datos transaccionales y archivos de texto (usuarios.txt) para las credenciales de *login*.

- Manejo de Hilos (Threading): La interfaz gráfica utiliza SwingUtilities.invokeLater() en la clase Main. Esta práctica es estándar en Java Swing para garantizar que la actualización de la interfaz se ejecute en el Event Dispatch Thread (EDT), previniendo que la aplicación se "congele" durante las operaciones pesadas de persistencia.

3. Integridad y Robustez de la Base de Datos (SQLite)

La base de datos SQLite 3 fue diseñada para la máxima integridad de los datos, utilizando validaciones a nivel de Base de Datos (BD) que complementan la lógica de validación de Java:

3.1. Integridad Referencial

Se implementaron Foreign Keys con diferentes políticas ON DELETE para gestionar las eliminaciones de manera coherente con la lógica de negocio:

- ON DELETE CASCADE: Utilizada en entidades dependientes (ej., asistencia y memoria_personal) para eliminar automáticamente sus registros si la Reunion o el Miembro asociado se eliminan.

- ON DELETE SET NULL: Permite que un Miembro sea desasignado de un Grupo si este último es eliminado, sin borrar al miembro (preserva el historial del miembro).

- ON DELETE RESTRICT: Evita la eliminación accidental de registros clave (ej., la Iglesia o un Grupo que aún tiene reuniones) hasta que no se eliminen sus referencias.

3.2. Restricciones y Estándares de Datos

- CHECK Constraints: Se implementaron para asegurar la consistencia de los datos, forzando campos obligatorios (ej., length(nombre) > 0) y garantizando los valores de tipo ENUM (rol IN ('MIEMBRO', 'LIDER_GRUPO', 'LIDER_IGLESIA')).

- UNIQUE Constraint: Se aplica al campo idMiembro en la tabla expediente, asegurando que exista una relación uno a uno (1:1) estricta, tal como lo requiere el modelo.

- Manejo de Fechas/Booleanos: Las fechas se almacenan como TEXT (ISO 8601) y los booleanos como INTEGER (0/1) con CHECK constraint, estandarizando los tipos para la máxima compatibilidad con SQLite.

3.3. Resolución de Problemas de Diseño

Durante la implementación, se resolvió un desafío técnico crucial: la Dependencia Circular entre las tablas miembro (que referencia a grupo mediante idGrupo) y grupo (que referencia a miembro mediante idLider). La solución implicó la ejecución temporal de PRAGMA foreign_keys = OFF durante la inicialización del esquema y la posterior reactivación de la integridad referencial.

10. DESCOMPOSICIÓN DEL PROYECTO

La descomposición del proyecto sigue un diseño jerárquico y por capas, alineado con los principios de la Programación Orientada a Objetos (POO). A continuación, se detalla la estructura completa de clases e interfaces, incluyendo sus atributos, métodos y relaciones de herencia e implementación.

1. Clases e Interfaces del Modelo de Dominio

Estas clases representan las entidades de la base de datos y heredan de la clase abstracta EntidadBase para estandarizar el campo de identidad (id) y la validación.

Tabla 2. Clases e Interfaces del Modelo de Dominio

Nombre	**Tipo / Herencia**	**Atributos Principales (Privados)**	**Métodos (Públicos) Relevantes**
EntidadBase	**Clase Abstracta**	id (protected) 1	**validar()** (Abstracto), **obtenerNombreTabla()** (Abstracto) 2
Iglesia	Derivada de EntidadBase	nombre, direccion, telefono, pastorPrincipal 3	Iglesia(...), validar() 4, obtenerNombreTabla() 5
Miembro	Derivada de EntidadBase	nombre, apellido, telefono, correo, rol, idGrupo 6	Miembro(), validar() 7, esLider(), toString() 8
Grupo	Derivada de EntidadBase	nombre, descripcion, idLider, idIglesia 9	Grupo(...), validar() 10, obtenerNombreTabla() 11
Reunion	Derivada de EntidadBase	idGrupo, fecha, tema, responsable, finalizada (boolean), minuta 12	Reunion(...), validar() 13, **finalizarReunion(minuta)** 14, obtenerNombreTabla() 15
Asistencia	Derivada de EntidadBase	idReunion, idMiembro, asistio (boolean), excusa 16	Asistencia(...), validar() 17, obtenerNombreTabla() 18
Expediente	Derivada de EntidadBase	idMiembro, fechaBautismo, profesion, observaciones 19	Expediente(...), validar() 20, obtenerNombreTabla() 21
MemoriaPersonal	Derivada de EntidadBase	idReunion, idMiembro, contenido 22	MemoriaPersonal(...), validar() 23, obtenerNombreTabla() 24

Fuente: Elaboración propia del equipo

2. Interfaz Principal (ICRUD)

La interfaz ICRUD<T> define el contrato para las operaciones de persistencia, implementando el patrón DAO.

Tabla 3. Interfaz Principal (ICRUD)

Interfaz	**Clases que Implementan (DAOs)**	**Diferencia en la Implementación (Lógica SQL)**
ICRUD<T>	IglesiaDAO, MiembroDAO, GrupoDAO, ReunionDAO, AsistenciaDAO, MemoriaPersonalDAO, ExpedienteDAO	Cada clase implementa el acceso a datos para su entidad específica, por ejemplo:
	MiembroDAO	Maneja la lógica CRUD para la tabla miembro, incluyendo la columna rol y idGrupo.
	IglesiaDAO	Principalmente implementa leer(id) y actualizar(t), ya que solo existe un registro de iglesia.
	ExpedienteDAO	Contiene el método especializado leerPorMiembro(idMiembro) para manejar la relación 1:1, consultando por la FK única en lugar del ID primario25.
	AsistenciaDAO	Contiene el método especializado listarPorReunion(idReunion), esencial para la toma de asistencia en la vista.

Fuente: Elaboración propia del equipo

3. Clases de Infraestructura y Control

Tabla 4. Clases de Infraestructura y Control

Nombre	**Función (Paquete)**	**Atributos Clave (Privados)**	**Métodos Relevantes (Comportamiento)**
ConexionSQLite	Persistencia (Singleton)	instancia, conexion, URL_DB 26	conectar(), desconectar() 27
GestorAutenticacion	Persistencia (Seguridad) 28	rutaArchivoUsuarios 29	autenticar(usuario, clave), registrarUsuario(...) 30
ControladorPrincipal	Servicio (Lógica de Negocio) 31	usuarioActual, idiomaActual 32	registrarMiembro(), agendarReunion(), getTexto(clave)
Estadisticas	Servicio (Reportes) 33	Conexión a BD 34	getTotalMiembros(), getTotalGrupos(), getPorcentajeAsistenciaGlobal()
Validador	Utilidad (Validaciones) 35	*REGEX constantes*	esTextoVacio(), esCorreoValido(), esTelefonoValido()
PantallaLogin	Vista (GUI) 36	txtUsuario, txtPassword	procesarLogin()
MenuPrincipal	Vista (Dashboard) 37	rolUsuario	abrirGestionMiembros(), mostrarReportesElegante()

Fuente: Elaboración propia del equipo

11. CONSIDERACIONES Y COMPLICACIONES DEL PROYECTO

El desarrollo del sistema de gestión presentó varios desafíos técnicos y logísticos que requirieron decisiones de diseño específicas, tal como se documenta en la Bitácora Técnica.

1. **Desafíos Técnicos de Persistencia (SQLite)**

- Rutas de Archivos y Conexión: Inicialmente, existieron conflictos de rutas (Path) al intentar ubicar la base de datos iglesia.sqlite dentro de paquetes anidados, lo que provocaba errores de conexión.
    - Solución: Se estandarizó la ubicación del archivo en la raíz del proyecto, simplificando la URL de conexión JDBC (jdbc:sqlite:iglesia.sqlite) y asegurando la portabilidad.
- Conflicto de Integridad Referencial (Dependencia Circular): Se detectó una dependencia circular entre las tablas Miembro (que requiere idGrupo) y Grupo (que requiere idLider).
    - Solución: Se implementó una secuencia de comandos: se creó la tabla miembro primero, luego grupo (referenciando miembros existentes), y finalmente se actualizaron los miembros para asignarles un grupo. Adicionalmente, se utilizó PRAGMA foreign_keys = OFF durante la inicialización para evitar la violación de FK, y se reactivó después para garantizar la integridad.
- Dificultades con Librerías: Java no reconocía el driver de SQLite (org.xerial:sqlite-jdbc) de forma nativa.
    - Solución: Se tuvo que agregar manualmente la librería .jar a la configuración de módulos del IDE para que el proyecto pudiera compilar y conectarse a la BD.

2. **Desafíos del Lenguaje y Paradigma (Java/POO)**

- Convención de Nombres (POO Estricto): Hubo una complicación inicial para asegurar que todos los desarrolladores respetaran la convención de los Getters y Setters (ej., getIdGrupo() en lugar de getidGrupo), lo cual es crucial para la reflexión y el uso de librerías avanzadas de Java.
- Lectura de Archivos Planos y Sanitización: La lectura del archivo usuarios.txt para la autenticación generó falsos negativos debido a espacios en blanco invisibles (.trim()) en las credenciales.
    - Solución: Se optimizó la clase GestorAutenticacion implementando el método .trim() en la lectura de credenciales para eliminar estos espacios y asegurar la correcta comparación.
- Manejo de la Interfaz (Threading): Se enfrentó el riesgo de que la interfaz se congelara durante operaciones de persistencia.
    - Solución: Se aplicó el estándar de Java Swing, modificando el Main para ejecutar la ventana principal dentro del Event Dispatch Thread (EDT) usando SwingUtilities.invokeLater().

3. **Decisiones de Diseño (Complicaciones Arquitectónicas)**

- Estrategia de Vistas: Para mantener limpio el Diagrama UML y evitar la proliferación de clases, se decidió implementar la mayoría de los módulos de gestión (Miembros, Grupos, Reuniones) como Ventanas Modales (JDialog) dentro de métodos privados de la clase MenuPrincipal, en lugar de crear un JFrame separado para cada uno.

- Diseño de Datos (Expediente): Hubo un debate sobre dónde almacenar el expediente del miembro (si como una columna TEXT dentro de la tabla miembro o como una tabla separada).

    - Decisión Final: Se optó por la tabla separada expediente con una restricción UNIQUE en idMiembro, manteniendo la normalización y la consistencia con el diseño UML.

12. TABLA DE PREMIACIONES

Tabla 5. Tabla de premiaciones

Premiaciones	Integrante
Integrante con solución más creativa	Dany Josue Noguera Tinoco
Integrante más puntual.	Jonatan Isai Varela Girón
Integrante que se llevo mejor con todos los integrantes del grupo	Kevin Alejandro Laínez

Fuente: Elaboración propia del equipo

13. RECURSOS DE INTERÉS

Recursos de Interés para el Sustento Técnico

1. Arquitectura y Patrones de Diseño (POO)

Estos recursos son esenciales para justificar la estructura modular del código, la herencia, y la separación de responsabilidades.

- POO y Abstracción: Para fundamentar la implementación de la clase abstracta EntidadBase y el encapsulamiento de datos.
- Patrón de Diseño DAO (Data Access Object): Para validar la separación de la lógica de negocio de la capa de persistencia (paquete dao/ y la interfaz ICRUD<T>).
- Patrón Singleton: Para justificar la unicidad de la conexión a la base de datos a través de ConexionSQLite.
- UML y Diagramas de Clases: Para validar la descomposición del problema en objetos, sus relaciones de herencia (EntidadBase) y sus relaciones de implementación (ICRUD<T>).

2. Tecnologías de Persistencia y Base de Datos

Estos recursos validan el uso de las tecnologías de *backend* seleccionadas y las decisiones de diseño de la BD.

- SQLite y JDBC: Para sustentar el uso de la base de datos embebida SQLite 3 en un entorno de escritorio y la integración con Java a través del *driver* org.xerial:sqlite-jdbc.
- Seguridad SQL: Para fundamentar la necesidad de utilizar PreparedStatement como mecanismo principal de protección contra ataques de inyección SQL en la capa DAO.
- Integridad Referencial y DDL: Para justificar las validaciones a nivel de base de datos (CHECK constraints, UNIQUE constraints) y las políticas de eliminación (ON DELETE CASCADE, ON DELETE SET NULL) aplicadas en el *script* DDL.

3. Entorno de Desarrollo y Librerías de Interfaz

- Java Swing y Concurrencia: Para explicar el uso de la librería Swing para la GUI y la necesidad técnica de utilizar SwingUtilities.invokeLater() para garantizar que el código se ejecute en el Event Dispatch Thread (EDT).
- Internacionalización: Para documentar el proceso de soporte de múltiples idiomas a través de Resource Bundles.
- FlatLaf: Para citar la librería utilizada para el *look and feel* moderno de la interfaz de usuario.

14. BIBLIOGRAFÍA

E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA, USA: Addison-Wesley, 1994. (Fundamento para los patrones DAO y Singleton).

B. J. Stroustrup, The C++ Programming Language, 4th ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2013. (Fundamental para principios de Herencia y Abstracción en POO).

M. Fowler, Patterns of Enterprise Application Architecture. Reading, MA, USA: Addison-Wesley, 2003. (Sustento del patrón MVC y la arquitectura en capas).

J. R. Watterson and T. M. Alspaugh, "Implementing the Data Access Object pattern in Java applications," *Software Engineering Notes*, vol. 35, no. 1, pp. 1–4, 2010. (Sustento del patrón DAO en Java).

15. MANUAL DE USUARIO

Este manual describe los procedimientos básicos y las funcionalidades de los módulos operativos más importantes del sistema, enfocándose en la interacción según el rol asignado.

Módulo 1: Acceso al Sistema y Tablero de Control (Dashboard)

Este módulo es la puerta de entrada y el centro de navegación, donde la seguridad basada en roles (RBAC) entra en vigor.

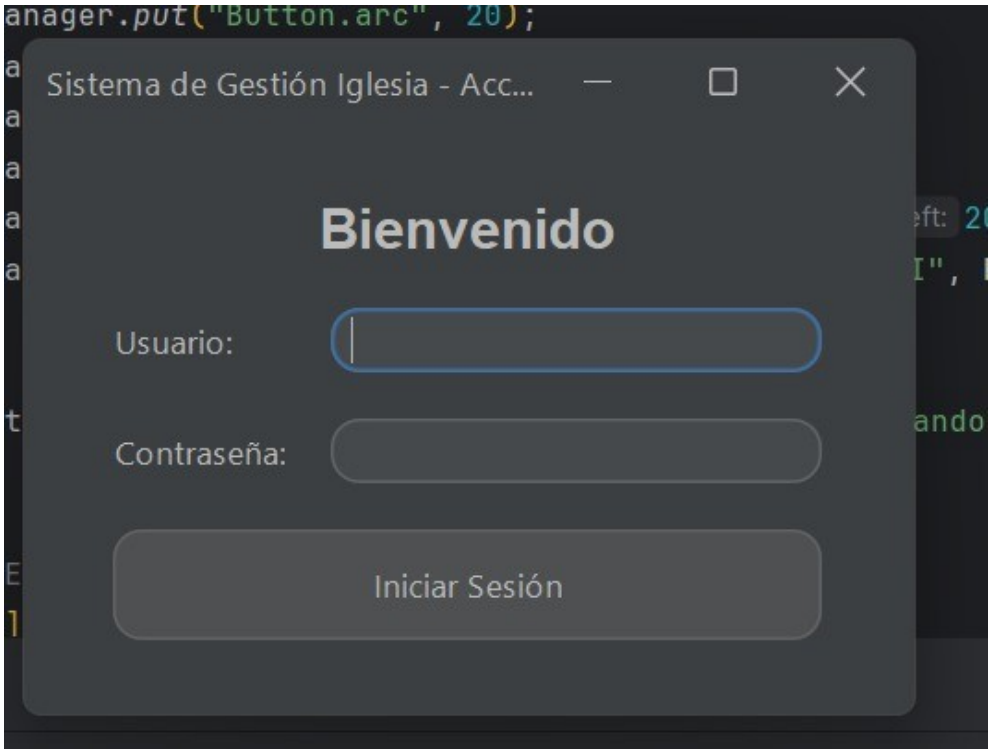

Figura 2. Módulo de Autenticación (Login).**Fuente:** Elaboración propia

1.1. **Inicio de Sesión**

El sistema valida al usuario y la contraseña contra el archivo de seguridad (usuarios.txt) para determinar el nivel de privilegios.

- Paso 1: Credenciales: Ingrese su Usuario y Contraseña en la PantallaLogin.
- Paso 2: Autenticación: El sistema llama al GestorAutenticacion para verificar las credenciales y obtener el Rol asociado.
- Paso 3: Acceso: Si el inicio es exitoso, la ventana de *login* se cierra (dispose()) y se lanza el MenuPrincipal.

1.2. **Tablero de Control y Seguridad por Rol**

El MenuPrincipal actúa como el *hub* central, donde los botones y funcionalidades están limitados según el Rol del usuario.

Tabla 6. Tablero de Control y Seguridad por Rol

Rol de Usuario	Acceso y Restricciones Clave
Líder de Iglesia (Pastor)	Acceso Total. Puede ver todos los módulos, incluyendo la Gestión de Grupos, Reportes Gerenciales, y Configuración de la Iglesia para edición.
Líder de Grupo	Acceso Operativo. Puede acceder a la Gestión de Miembros y Reuniones, pero tiene el acceso bloqueado a Reportes y a la creación de nuevos Grupos.
Miembro	Acceso Limitado / Lectura. Puede editar su perfil y consultar la información de su grupo. Si intenta acceder a un módulo restringido (ej. Reportes), recibirá un mensaje de "ACCESO DENEGADO".

Fuente: Elaboración propia del equipo

Módulo 2: Gestión de Reuniones y Asistencia (Rol: Líder de Grupo)

Figura 3. Módulo de Gestión de Reuniones. **Fuente:** Elaboración propia

Este módulo permite al Líder de Grupo organizar los eventos de su célula o ministerio y registrar la participación, cumpliendo con el requisito de control de inasistencias.

2.1. Agendar una Nueva Reunión

El Líder accede a la opción "Gestión de Reuniones" desde el MenuPrincipal.

1. Selección del Grupo: El líder selecciona el grupo al que pertenece la reunión (si aplica).

2. Ingreso de Datos: Se ingresan el Tema, la Fecha (en formato AAAA-MM-DD), y el nombre del Responsable del evento.

3. Validación y Creación: El sistema verifica que los campos obligatorios estén llenos. Si la validación es exitosa, se crea un nuevo registro de Reunion en la base de datos, con el estado inicial de "ABIERTA" (finalizada = false).

2.2. Proceso de Toma de Asistencia y Finalización

La asistencia solo puede ser registrada mientras la reunión esté en estado "ABIERTA".

1. Seleccionar Reunión: Desde la tabla, el líder selecciona una reunión que esté en estado "ABIERTA" y presiona "Tomar Asistencia".

2. Registro de Asistencia: El sistema lista a todos los Miembros y el líder procede a marcar el estado:

    - "Marcar PRESENTE": Crea un registro de Asistencia con el campo asistio = true.

    - "Marcar AUSENTE": Crea un registro de Asistencia con asistió = false y se puede ingresar una excusa.

3. Finalizar Reunión (Minuta): Una vez registrada la asistencia para todos, el líder presiona "Finalizar Reunión".

    - El sistema solicita la Minuta o Resumen del evento.

    - La reunión se actualiza en la base de datos (ReunionDAO.actualizar()), se cambia el estado a "CERRADA" (finalizada = true) y se guarda el texto de la minuta.

    - La reunión ya no podrá ser modificada ni se podrá volver a tomar asistencia.

15. CÓDIGO FUENTE MAIN ()

```
package Proyecto_iglesia.app;

import Proyecto_iglesia.vista.PantallaLogin;
import javax.swing.*;
import java.awt.*;

public class Main {

    public static void main(String[] args) {

        // ============================================================
        // 1. ACTIVAR TEMA (FlatLaf) — MODO BLINDADO
        // ============================================================
        try {
            // TRUCO: Cargamos la clase por nombre (Reflection).
            // Así, si el profesor no tiene la librería, NO da error de compilación.
            // Simplemente salta al catch.
            UIManager.setLookAndFeel("com.formdev.flatlaf.FlatDarkLaf");

            // Configuración estética (solo se aplica si lo anterior funcionó)
            UIManager.put("Component.accentColor", Color.decode("#3498db"));
            UIManager.put("Button.arc", 20);
            UIManager.put("Component.arc", 20);
            UIManager.put("TextComponent.arc", 20);
            UIManager.put("ProgressBar.arc", 20);
            UIManager.put("Button.margin", new Insets(10, 20, 10, 20));
            UIManager.put("defaultFont", new Font("Segoe UI", Font.PLAIN, 14));

            System.out.println("✔ Tema FlatLaf cargado correctamente.");
        }
        catch (Exception ex) {
            // SI FALLA (Falta la librería): Usamos el diseño del sistema operativo
            System.err.println("⚠ No se encontró la librería FlatLaf. Usando diseño estándar.");
            try {
                UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        // ============================================================
        // 2. SELECCIÓN DE IDIOMA
        // ============================================================
        Object[] opciones = {"Español", "English"};
        int seleccion = JOptionPane.showOptionDialog(
                null,
                "Language / Idioma:",
                "Configuración",
                JOptionPane.DEFAULT_OPTION,
                JOptionPane.QUESTION_MESSAGE,
```

```
            null,
            opciones,
            opciones[0]
        );

        Proyecto_iglesia.servicio.ControladorPrincipal controlador =
            Proyecto_iglesia.servicio.ControladorPrincipal.getInstancia();

        controlador.configurarIdioma(seleccion == 1 ? "en" : "es");

        // ============================================================
        // 3. INICIAR LA APP
        // ============================================================
        SwingUtilities.invokeLater(() -> {
            PantallaLogin login = new PantallaLogin();
            login.setVisible(true);
        });
    }
}
```

16. AUTORIZACIÓN Y RENUNCIA

Los (as) autores (as) facultan a la carrera de Informática Administrativa del Departamento de Informática adscrito a la Facultad de Ciencias Económicas, Administrativas y Contables de la Universidad Nacional Autónoma de Honduras a través de sus docentes para socializar el presente trabajo en los medios que estimen conveniente así mismo se seden los derechos sobre el mismo. La carrera de Informática Administrativa o los docentes no son responsables por el contenido y las implicaciones de lo que esta expresado en este documento.