INSTITUTO TECNOLÓGICO SUPERIOR DE SAN ANDRÉS TUXTLA

INGENIERÍA INFORMÁTICA

Materia:

CRIPTOGRAFÍA

Catedrático:

M.T.I. ERICK DE JESÚS TÉLLEZ VERA

Grupo:

910-B

Alumno:

LUIS ENRIQUE RIVAS CHAMPALA

EDDI JOSUÉ ZÚÑIGA CHÁVEZ

ÁNGEL JOSUÉ ZÚÑIGA CHÁVEZ

UIII

PROYECTO MENSAJERÍA

25/NOV/2025

CONTENIDO

ESTRUCTURA Y CONTENIDO 3

1. ESPECIFICACIONES DE REQUERIMIENTOS 3

1.1 Definición General del Proyecto de software 3

1.2 Especificación de Requerimientos del Proyecto 3

1.3 Procedimientos de Instalación y Prueba 4

2. ARQUITECTURA DEL SISTEMA 5

2.1 Descripción Jerárquica 5

2.2 Diagrama de Módulos 5

2.3 Descripción Individual de los Módulos 6

3. DISEÑO DEL MODELO DE DATOS 9

3.1 Diagrama Entidad-Relación (Modelo Lógico) 9

3.2 Modelo de Datos 9

4. DESCRIPCIÓN DE PROCESOS Y SERVICIOS 10

4.1 Servicio: Envío Seguro de Mensajes 10

4.2 Servicio: Inicio de Sesión (Login) 12

4.3 Manual de Invocación 12

CONCLUSIONES 13

MANUAL DE USO 14

ESTRUCTURA Y CONTENIDO

1. ESPECIFICACIONES DE REQUERIMIENTOS

1.1 Definición General del Proyecto de software

El proyecto consiste en el desarrollo de un Sistema de Mensajería Segura basado en la arquitectura Cliente-Servidor. La funcionalidad principal es permitir la comunicación en tiempo real entre múltiples usuarios a través de salas de chat protegidas por contraseña.

El propósito fundamental del desarrollo es garantizar la confidencialidad de la información. A diferencia de los chats convencionales, este sistema implementa cifrado **AES-GCM** (Advanced Encryption Standard con Galois/Counter Mode) antes de persistir cualquier dato en la base de datos. Esto asegura que ni siquiera el administrador del sistema pueda leer los mensajes o las contraseñas de los usuarios, ya que estos se almacenan como texto cifrado ilegible.

1.2 Especificación de Requerimientos del Proyecto

El sistema cumple con los siguientes requerimientos técnicos y funcionales

a) Requerimientos Funcionales:

- **Gestión de Usuarios:** Registro de nuevas cuentas y autenticación segura (Login) mediante validación de credenciales cifradas.
- **Gestión de Salas**: Creación de nuevas conversaciones protegidas por contraseña y unión a salas existentes mediante ID o nombre.
- **Comunicación**: Envío y recepción de mensajes en tiempo real (Broadcast).
- **Historial**: Recuperación y descifrado automático de mensajes anteriores al ingresar a una sala.

b) Requerimientos No Funcionales (Restricciones):

- **Lenguaje y Plataforma**: Desarrollado íntegramente en Java (JDK 25).
- **Interfaz Gráfica**: Uso de JavaFX para una experiencia de usuario moderna y responsiva.
- **Concurrencia:** Capacidad para manejar múltiples clientes simultáneos mediante Hilos (Threads) y Sockets TCP.
- **Persistencia**: Almacenamiento relacional utilizando MariaDB.
- **Red**: Capacidad de operación en Red de Área Local (LAN) mediante configuración de IP externa.
- **Seguridad**: Implementación de criptografía simétrica para mensajes y credenciales.

c) Alcances y Limitaciones:

El proyecto es una versión funcional académica (1.0-SNAPSHOT). Permite registro, login, creación de salas y chat en vivo. No incluye funciones de envío de archivos multimedia o llamadas de voz en esta versión.

1.3 Procedimientos de Instalación y Prueba

Para el despliegue del software en un entorno de producción o pruebas, se deben seguir los siguientes pasos:

- **Pre-requisitos**: Disponer de una máquina servidor y máquinas clientes con Java JDK 25 instalado.
- **Base de Datos**: Ejecutar el script SQL suministrado en un servidor MariaDB escuchando en el puerto 3306. Se debe crear el usuario administrador con permisos remotos (%).
- **Empaquetado**: El software se distribuye como un archivo "Fat JAR" (Chat.jar) que contiene todas las dependencias.
- **Configuración de Red:**

- Crear un archivo de texto plano llamado “server.properties” en la misma carpeta del ejecutable.
- Definir la IP del servidor: server.ip=192.168.X.X.

- **Ejecución:**
  - **Modo Servidor**: java -cp Chat.jar com.tarea.ChatServer
  - **Modo Cliente:** java -jar Chat.jar (o doble clic en Windows).

2. ARQUITECTURA DEL SISTEMA

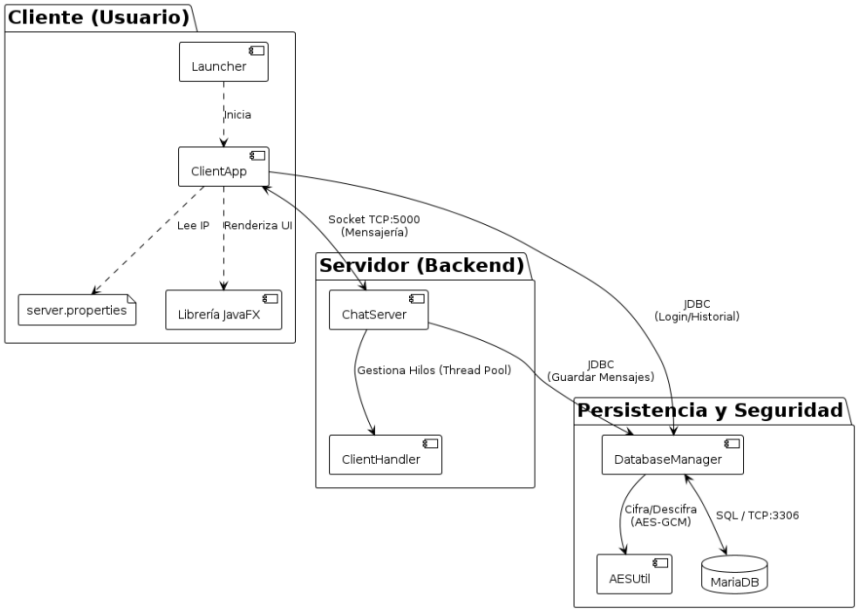
2.1 Descripción Jerárquica

El sistema sigue una arquitectura modular por capas, organizada dentro del espacio de nombres (package) com.tarea, separando lógicamente las clases de utilidad, interfaz, red y datos. Aunque físicamente se distribuye en un único artefacto ejecutable, lógicamente se divide en:

- **Capa de Presentación (UI):** Responsable de la interacción visual.
- Capa de Lógica de Negocio (Backend): Responsable de la gestión de sockets y reglas de comunicación.
- **Capa de Acceso a Datos (DAO):** Responsable de la persistencia y las transacciones SQL.
- **Capa Transversal de Seguridad**: Responsable de los servicios criptográficos invocados por la capa de datos.

2.2 Diagrama de Módulos

El siguiente diagrama ilustra los componentes del sistema y sus interrelaciones de alto nivel:



2.3 Descripción Individual de los Módulos

A) Módulo de Entrada (Launcher.java)

- **Descripción y Propósito**: Es la clase principal (Main Class) definida en el manifiesto del JAR. Su propósito es servir de puente para inicializar el entorno de ejecución JavaFX.
- **Responsabilidad**: Invocar al método main de ClientApp.
- **Restricciones**: No contiene lógica de negocio. Su existencia es estrictamente técnica para evitar errores de carga de módulos en el ClassLoader al empaquetar librerías gráficas.
- **Implementación**: src/main/java/com/tarea/Launcher.java.

B) Módulo de Interfaz (ClientApp.java)

- **Descripción y Propósito**: Constituye el frontend de la aplicación. Gestiona el ciclo de vida de las ventanas y la experiencia del usuario.
- **Responsabilidad**:

- Gestionar la navegación entre escenas (Bienvenida → Login/Registro → Selección Sala → Chat).
- Validar datos de entrada (ej. .trim() para evitar mensajes vacíos, límite de 50 caracteres).
- Renderizar dinámicamente las "burbujas" de chat usando componentes VBox y HBox dentro de un ScrollPane.
- Mantener un hilo secundario (Thread) escuchando el Socket para actualizar la interfaz en tiempo real (Platform.runLater).

- **Dependencias**: Requiere DatabaseManager para operaciones de lectura y ChatServer para el envío de mensajes.
- **Implementación**: src/main/java/com/tarea/ClientApp.java.

C) Módulo de Servidor (ChatServer.java)

- **Descripción y Propósito**: Actúa como el nodo central de la red. Escucha peticiones y orquesta la comunicación entre clientes.
- **Responsabilidad**:
  - Iniciar el ServerSocket en el puerto 5000.
  - Implementar un ExecutorService (Pool de hilos) para manejar concurrencia sin bloqueos.
  - Mantener un mapa concurrente (activeChats) de sesiones activas.
  - Ejecutar la lógica de difusión (Broadcast): recibir un mensaje de un cliente y reenviarlo a todos los demás en la misma sala.
- **Dependencias**: Invoca a DatabaseManager para persistir cada mensaje que transita por la red.
- **Implementación**: src/main/java/com/tarea/ChatServer.java.

D) Módulo de Persistencia (DatabaseManager.java)

- Descripción y Propósito: Funciona como un DAO (Data Access Object). Abstrae la complejidad de SQL del resto de la aplicación.

- **Responsabilidad**:
  - Establecer conexiones JDBC seguras leyendo dinámicamente la IP desde server.properties.
  - Ejecutar sentencias DML (INSERT, SELECT) para usuarios, conversaciones y mensajes.
  - Responsabilidad Crítica: Es el único módulo autorizado para invocar a AESUtil. Garantiza que ningún dato salga hacia la BD sin estar cifrado y que ningún dato venga de la BD sin ser descifrado.
- **Implementación**: src/main/java/com/tarea/DatabaseManager.java.

E) Módulo de Seguridad (AESUtil.java)

- **Descripción y Propósito**: Biblioteca de utilidad estática que encapsula la lógica criptográfica.
- **Responsabilidad**:
  - Configurar el algoritmo AES/GCM/NoPadding.
  - Generar aleatoriamente un IV (Vector de Inicialización) de 12 bytes para cada operación de cifrado.
  - Realizar la codificación y decodificación en Base64 de los textos cifrados.
- **Restricciones**: Utiliza una clave simétrica predefinida (hardcodeada) de 128 bits para fines académicos.
- **Implementación**: src/main/java/com/tarea/AESUtil.java.

2.4 Dependencias Externas

El proyecto utiliza librerías gestionadas por Maven:

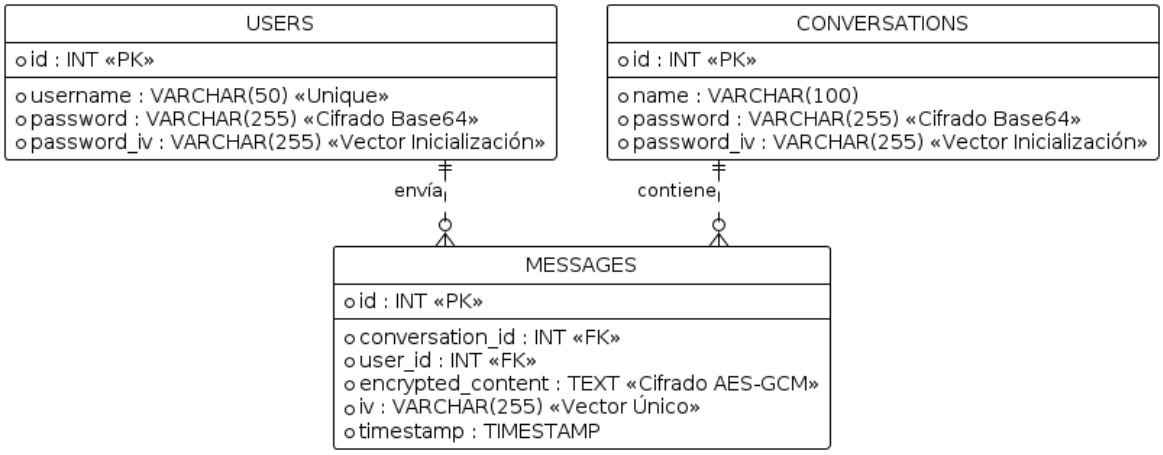
- **org.openjfx:javafx-controls (25.0.1):** Framework para la GUI. Se eligió sobre Swing por su capacidad de estilización CSS y manejo moderno de eventos.

- **org.mariadb.jdbc:mariadb-java-client (3.3.0):** Conector JDBC. Se eligió por su alta compatibilidad, rendimiento y licencia open-source (LGPL).
- **maven-shade-plugin (3.5.0):** Plugin de construcción. Utilizado para crear el "Fat JAR", permitiendo que la aplicación se ejecute en cualquier PC con Java instalado sin configurar el CLASSPATH manualmente.

3. DISEÑO DEL MODELO DE DATOS

El diseño del modelo de datos es **Relacional**, enfocado en la integridad referencial y la seguridad. A continuación, se describen las entidades en un formato agnóstico al lenguaje.

3.1 Diagrama Entidad-Relación (Modelo Lógico)



3.2 Modelo de Datos

A) Datos de entrada (Input)

Datos en texto plano proporcionados por la interacción humana.

- **Credenciales**: Usuario y Contraseña (String).
- **Contenido**: Cuerpo del mensaje (String, máx 50 caracteres).
- **Configuración**: Dirección IP del servidor (String en archivo properties).

B) Datos Internos (En Memoria)

Representación de los datos durante el procesamiento.

- **Clase ChatMessage**: Estructura temporal que mantiene el mensaje descifrado (String content) junto con el nombre del remitente (String username) para ser mostrados en la UI.
- **Mapa de Sesiones**: ConcurrentHashMap<Integer, List<PrintWriter>> en el servidor para enrutar mensajes a los sockets correctos.

C) Datos de Salida (Persistencia)

Datos finales almacenados en disco.

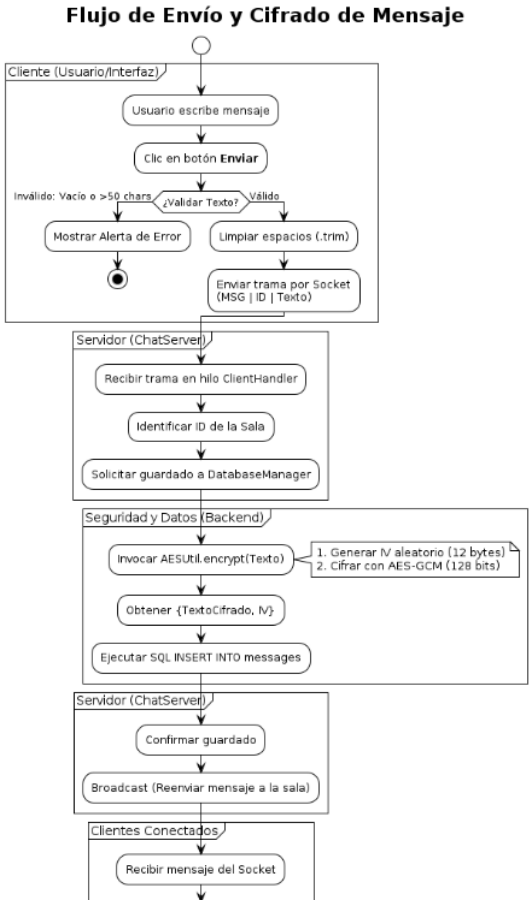
- **Cifrado**: Los campos sensibles (password, encrypted_content) se almacenan como cadenas Base64 ininteligibles.
- **Metadatos de Seguridad**: Cada registro sensible va acompañado obligatoriamente de su iv (Vector de Inicialización) correspondiente, sin el cual la recuperación es imposible.

4. DESCRIPCIÓN DE PROCESOS Y SERVICIOS

4.1 Servicio: Envío Seguro de Mensajes

Este proceso describe el ciclo de vida completo de un mensaje desde que se escribe hasta que se almacena y distribuye.

- **Diagrama de Flujo:**



- **Descripción del Algoritmo:**
  - **Entrada**: Texto plano del TextField.
  - **Validación**: Se aplica text.trim().isEmpty() para evitar envíos en blanco.
  - **Cifrado**: Se invoca AESUtil.encrypt(texto).
    - Genera IV aleatorio (12 bytes).
    - Cifra usando llave secreta.
    - Retorna {TextoCifradoBase64, IVBase64}.
- **Persistencia**: Se ejecuta el INSERT SQL con los valores cifrados.
- **Distribución**: Se envía la versión original (o descifrada) a través de los Sockets conectados a la sala específica.

4.2 Servicio: Inicio de Sesión (Login)

Proceso para autenticar usuarios sin exponer contraseñas reales.

- **Búsqueda**: Se busca el registro en la tabla users filtrando por username.
- **Recuperación**: Se obtienen password (cifrado) y password_iv.
- **Descifrado**: Se invoca AESUtil.decrypt(password, iv).
  - Decodifica Base64 a bytes.
  - Reconstruye el cifrador AES-GCM con la llave y el IV recuperado.
  - Obtiene el texto plano original.
- **Comparación**: Se valida textoPlanoRecuperado.equals(inputUsuario).
- **Resultado**: Retorna el ID de usuario si coincide, o -1 si falla.

4.3 Manual de Invocación

El sistema no requiere parámetros complejos de línea de comandos, ya que utiliza un archivo de propiedades para la configuración.

A) Invocación al cliente:

java -jar Chat.jar

- **Comportamiento**: Busca server.properties en el directorio actual. Si no existe, intenta conectar a localhost. Abre la interfaz gráfica.

B) Invocación del servidor

java -cp Chat.jar com.tarea.ChatServer

- **Comportamiento**: Inicia el servicio en el puerto 5000. Imprime logs en la consola estándar (System.out). No tiene interfaz gráfica.

CONCLUSIONES

El desarrollo de este Sistema de Mensajería Segura permitió abordar y resolver desafíos críticos en la ingeniería de software moderna, específicamente en el área de la ciberseguridad aplicada.

A) Retos y Soluciones:

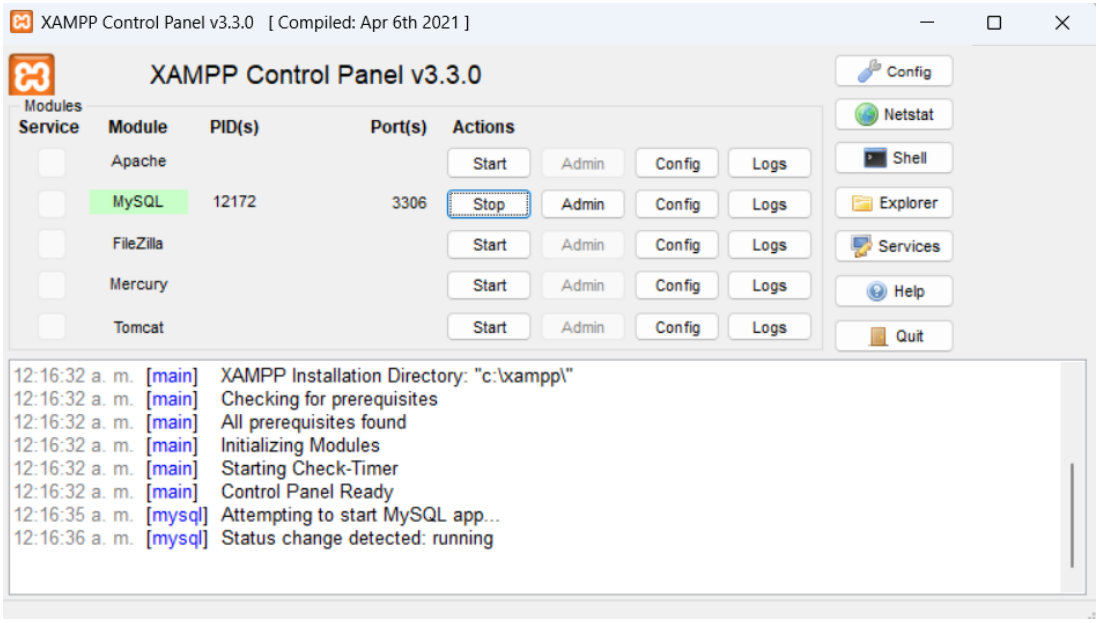
- **Integridad del Cifrado**: La principal complicación fue adaptar el modelo de base de datos relacional para soportar el cifrado AES-GCM. Dado que este algoritmo requiere un Vector de Inicialización (IV) único para cada encriptación, fue necesario reestructurar todas las tablas (users, conversations, messages) para incluir columnas dedicadas al almacenamiento de los IVs (password_iv, iv). Sin este cambio, el descifrado era matemáticamente imposible.
- **Despliegue JavaFX:** La creación de un ejecutable portable presentó dificultades debido a la arquitectura modular de las versiones recientes del JDK. Se adoptó la política de implementar una clase Launcher separada que desacopla la carga de la aplicación principal, permitiendo el uso exitoso del maven-shade-plugin.
- **Sincronización de Red**: Se implementaron mecanismos de hilos (Threads) y Platform.runLater para asegurar que los mensajes recibidos por la red (hilo de fondo) pudieran actualizar la interfaz gráfica (hilo UI) sin causar excepciones de concurrencia.

B) Experiencia Obtenida

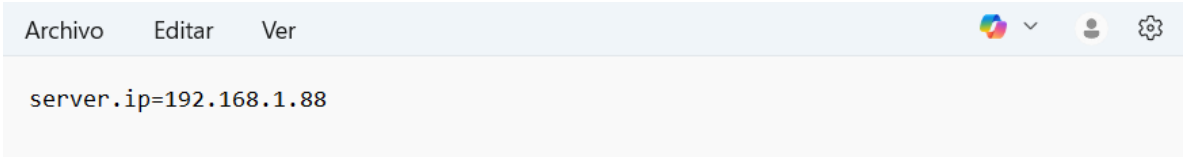
El proyecto demostró exitosamente que es posible implementar capas de seguridad robustas (grado militar AES-128) en aplicaciones académicas sin sacrificar la usabilidad ni el rendimiento, siempre que se respete una arquitectura limpia y modular.

MANUAL DE USO

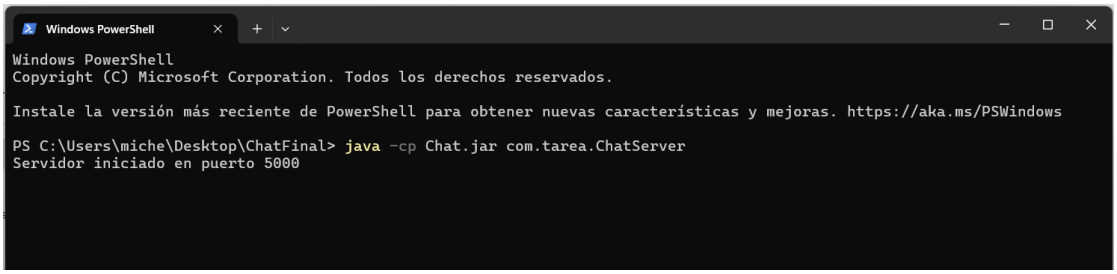
1. Hay que iniciar Mariadb en Xampp



2. Editar la dirección IP del servidor en el archivo “server.properties.txt”

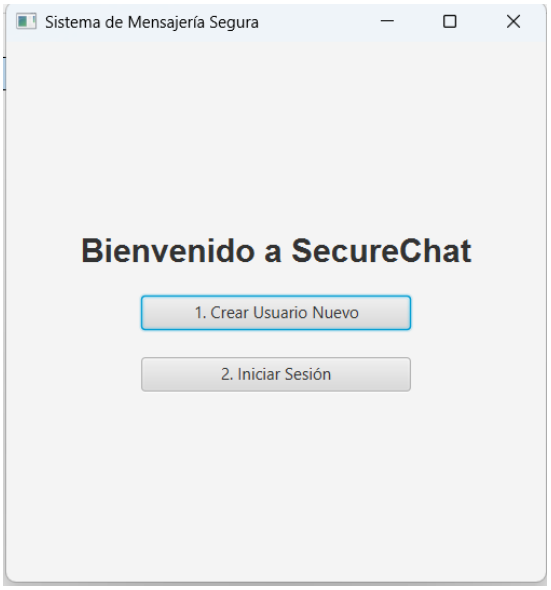


3. Abrir la terminal donde se encuentra el archivo “chat.jar”, se procede a ejecutar el servidor: java -cp Chat.jar com.tarea.ChatServer

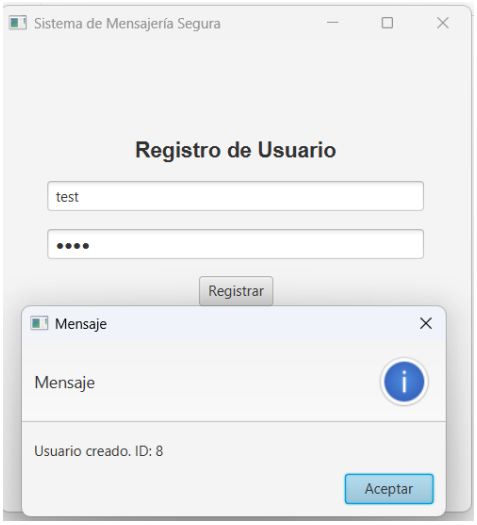


4. Iniciar la Interfaz de Mensajería. Abrir el archivo “Chat.jar”, tomar en cuenta que se debe tener el JDK versión 25 para que funcione correctamente.

5. Pantalla inicial. En dicha interfaz el usuario creará sus credenciales por primera vez para que se registre en la BD.

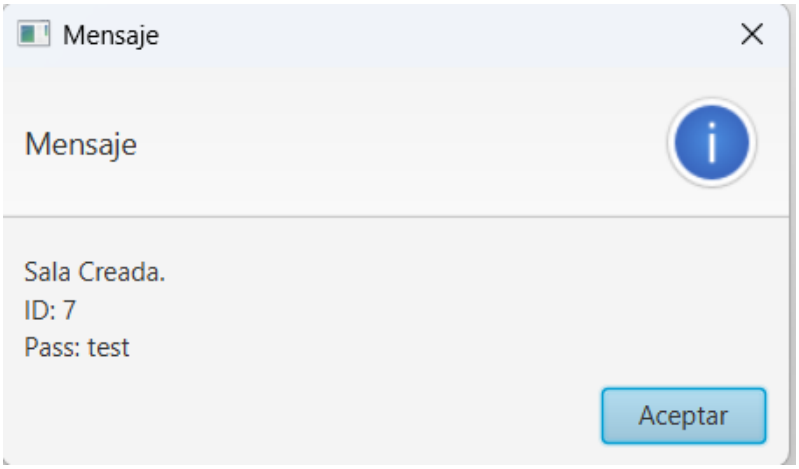


6. Al haber creado el usuario correctamente se mostrará una nueva ventana que indica el ID de ese usuario.

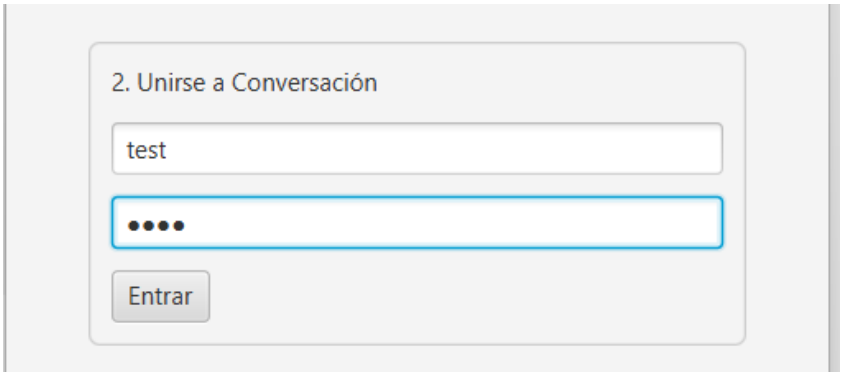


7. Se procede a loguearse. Y ahora es momento de crear una conversación o unirse a alguna que ya haya sido creada con anterioridad.

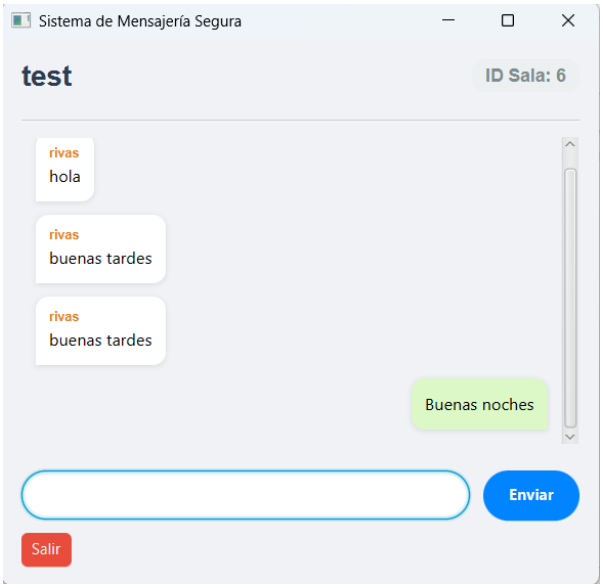
8. Una vez se haya creado la sala de la conversación, se muestra la siguiente ventana con el nuevo ID.



9. Posteriormente el usuario debe loguearse para entrar a la sala de chat



10. Después se inicia la conversación con el usuario 2. Estos pasos se repiten para agregar a más usuarios a la misma conversación.



11. Desde mysql se puede observar las salas de chat creadas así como los usuarios

```
MariaDB [(none)]> use chat_secure_db;
Database changed
MariaDB [chat_secure_db]> show tables;
+--------------------------+
| Tables_in_chat_secure_db |
+--------------------------+
| conversations            |
| messages                 |
| users                    |
+--------------------------+
3 rows in set (0.002 sec)

MariaDB [chat_secure_db]> select * from conversations;
+----+---------------+------------+
| id | name          | password   |
+----+---------------+------------+
|  1 | conversaci├n1 | 1234       |
|  2 | conversaci├n  | 1234       |
|  3 | nueva         | 1234       |
|  4 | nueva2        | nueva2     |
|  5 | pruebarivas   | pruebarivas |
|  6 | test          | test       |
|  7 | test          | test       |
+----+---------------+------------+
7 rows in set (0.001 sec)

MariaDB [chat_secure_db]>
```

```
MariaDB [chat_secure_db]> select * from users;
+----+----------+----------+
| id | username | password |
+----+----------+----------+
|  1 | prueba1  | prueba1  |
|  3 | prueba2  | prueba2  |
|  4 | nuevo    | nuevo    |
|  5 | nuevo2   | nuevo2   |
|  6 | rivas    | rivas    |
|  7 | champala | champala |
|  8 | test     | test     |
+----+----------+----------+
7 rows in set (0.002 sec)

MariaDB [chat_secure_db]>
```