Proyecto Unidad 1. Chat Offline con Gestión de Conversaciones en XML y Streams

- Acceso a datos

Francisco Javier Soler Aguado
2º DAM

Indice

1. Fase de Diseño

1.1 Descripción del proyecto

Este proyecto implementa una aplicación de chat offline desarrollada en Java, que permite la comunicación entre usuarios sin conexión a Internet, almacenando toda la información localmente mediante archivos XML.

El sistema gestiona usuarios, mensajes y archivos adjuntos, manteniendo la persistencia de los datos con la tecnología JAXB (Java Architecture for XML Binding) y aprovechando Java Streams para realizar análisis y exportación de datos.

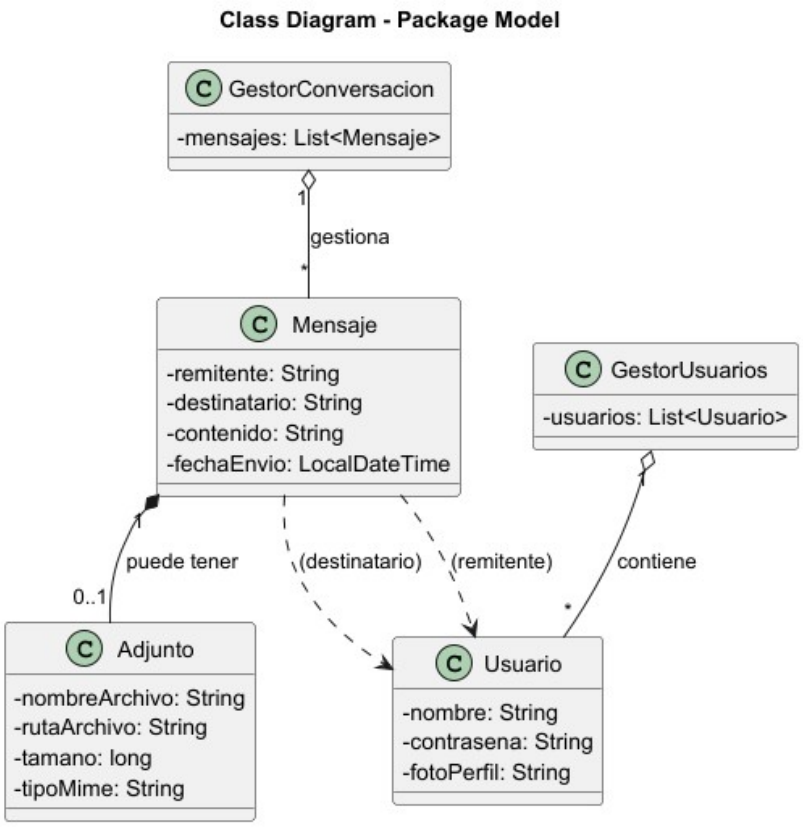
Algunas de sus funcionalidades destacables son:

- Registro y validación de usuarios almacenados en un XML estructurado.
- Envío de mensajes entre usuarios con soporte para adjuntos multimedia.
- Persistencia automática de las conversaciones y archivos mediante JAXB y Streams binarios.
- Análisis estadístico del historial de chat (mensajes por usuario, palabras más frecuentes, etc.).
- Exportación de conversaciones a texto o CSV.
- Interfaz JavaFX que facilita la interacción con las conversaciones.

1.2 Diagrama de clases

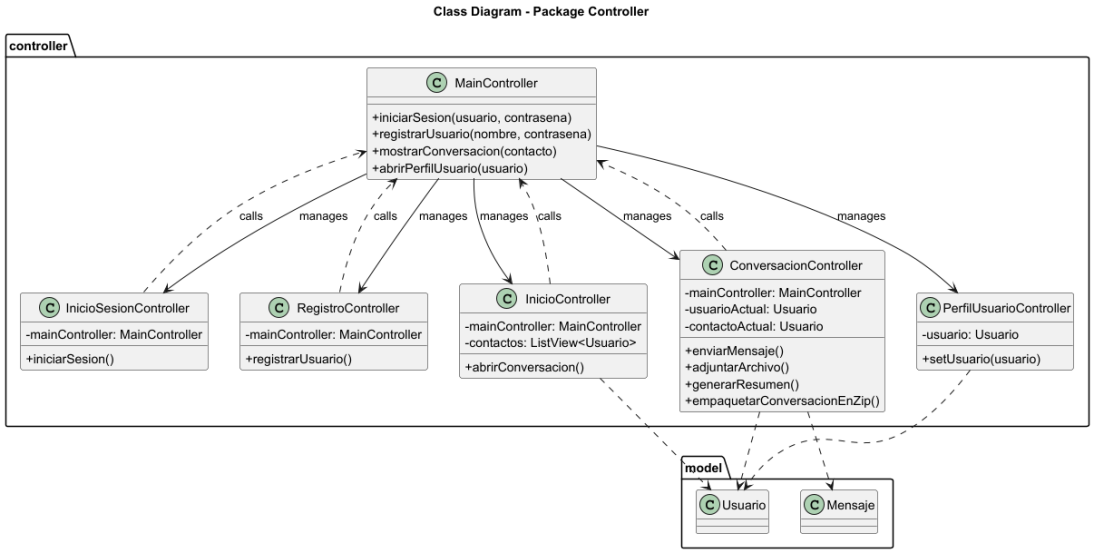
Model

- GestorConversacion: Administra las conversaciones, manteniendo una lista de objetos `Mensaje`.
- Mensaje: Contiene la información de cada mensaje enviado, incluyendo remitente, destinatario, contenido y fecha de envío. Cada mensaje puede incluir un `Adjunto`.
- Adjunto: Representa un archivo asociado a un mensaje, con información sobre su nombre, ruta, tamaño y tipo MIME.
- Usuario: Define los datos de un usuario (nombre, contraseña y foto de perfil). Puede ser tanto remitente como destinatario de los mensajes.
- GestorUsuarios: Gestiona la colección de usuarios registrados en el sistema.



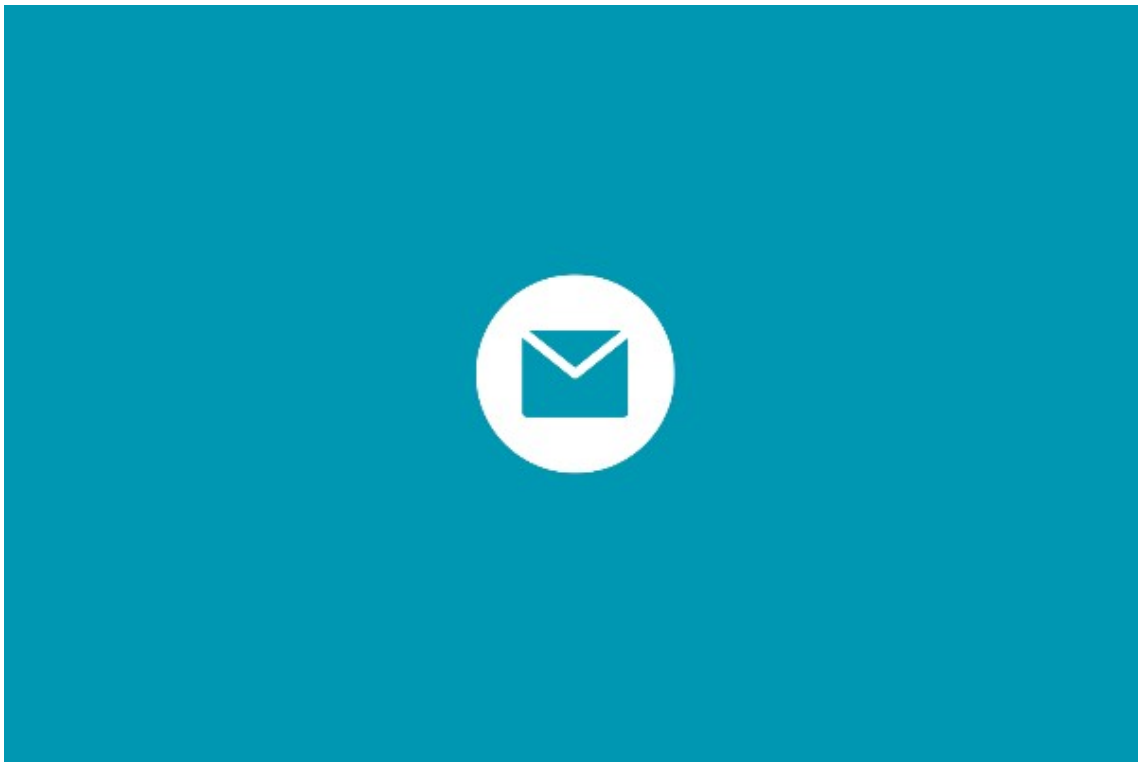
Paquete controller

- MainController: Controlador central del sistema. Gestiona las operaciones principales como el inicio de sesión, registro de usuarios, visualización de conversaciones y apertura de perfiles. Coordina la comunicación entre los demás controladores.
- InicioSesionController: Controla el proceso de autenticación del usuario, validando las credenciales y comunicándose con el `MainController` para iniciar sesión.
- RegistroController: Gestiona el registro de nuevos usuarios, conectándose al `MainController` para añadirlos al sistema.
- InicioController: Se encarga de la vista inicial del sistema, mostrando la lista de contactos y permitiendo abrir conversaciones con ellos.
- ConversacionController: Administra la lógica relacionada con las conversaciones. Permite enviar mensajes, adjuntar archivos, generar resúmenes y empaquetar conversaciones en formato ZIP.
- PerfilUsuarioController: Gestiona la visualización y actualización del perfil de usuario.

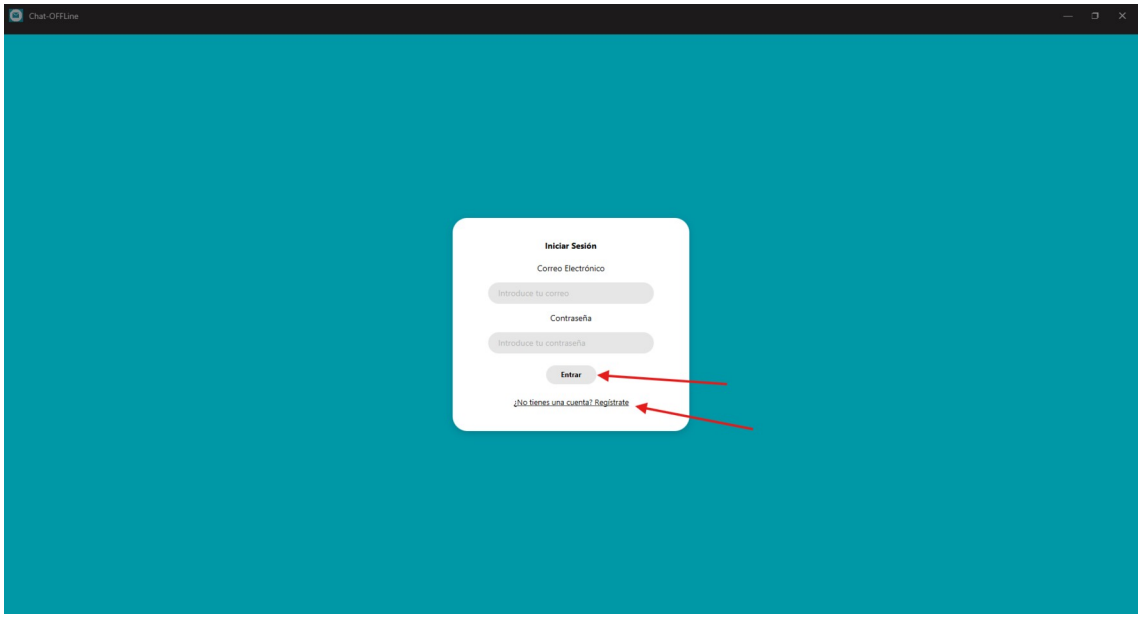


1.3 Diseño de pantallas

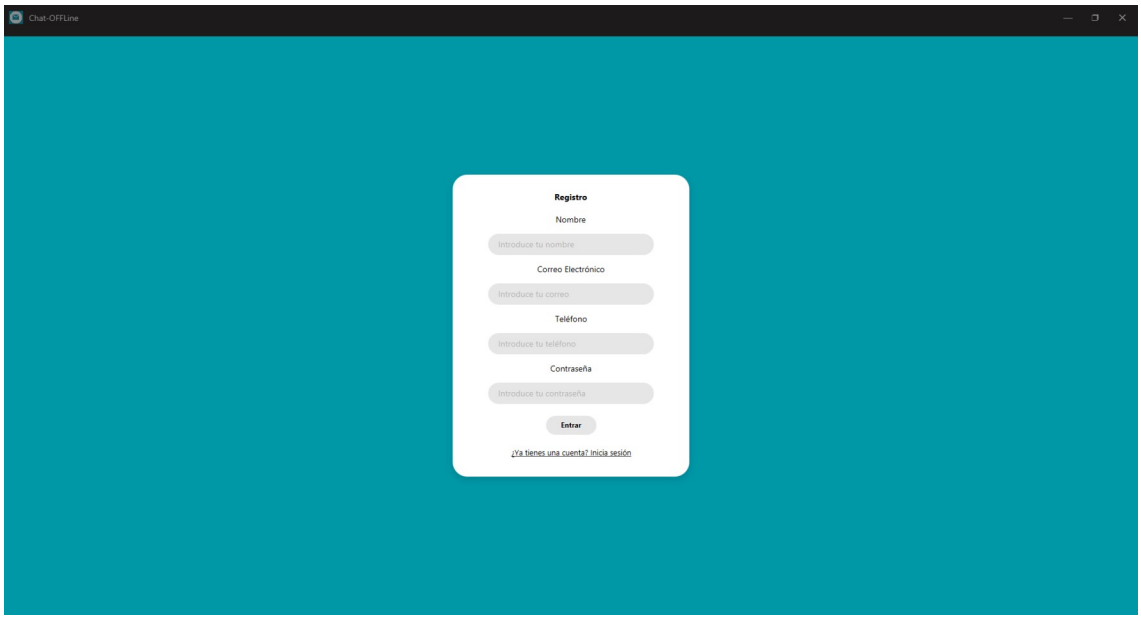
Primera pantalla: Una pantalla de carga que aparece cuando ejecutamos el programa



Segunda pantalla: Inicio de sesion, aquí introducimos los datos del inicio de sesión, si no tenemos cuentas, hacemos clic en “¿No tienes una cuenta? Registrate” que nos lleva a la pantalla de registro de usuario, en cambio, si ya tenemos cuenta y entramos nos llevará a la pantalla principal



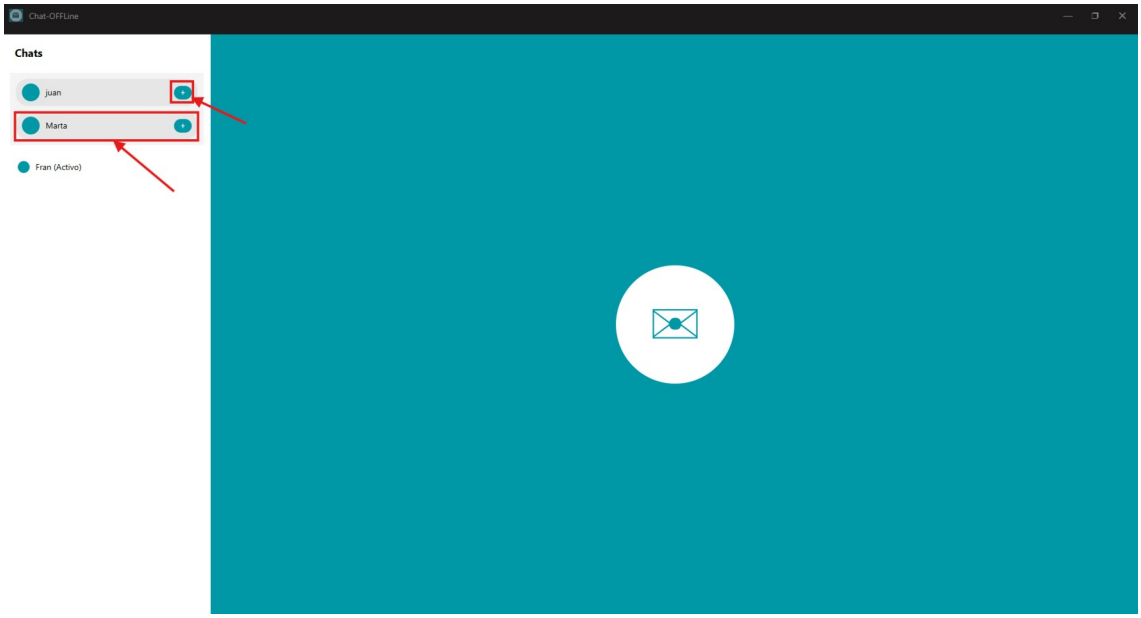
Tercera pantalla: Registro de usuario



Cuarta pantalla: Pantalla principal, aquí vemos los chats, con que usuario hemos logueado y nuestro estado(Activo o desconectado).

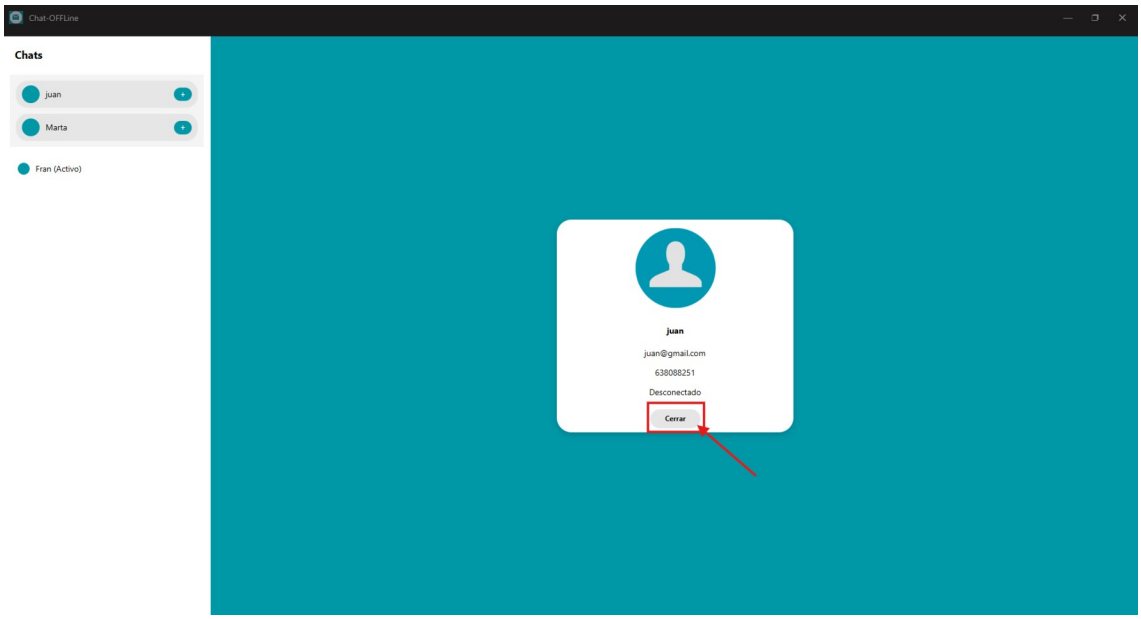
Si clicamos en el boton azul con el símbolo de “+”, nos abrirá el perfil de ese usuario.

Si hacemos clic en el contorno gris, nos abrirá el chat.



Quinta pantalla: Perfil de usuario

Si clicamos en cerrar nos devolverá a la pantalla anterior.



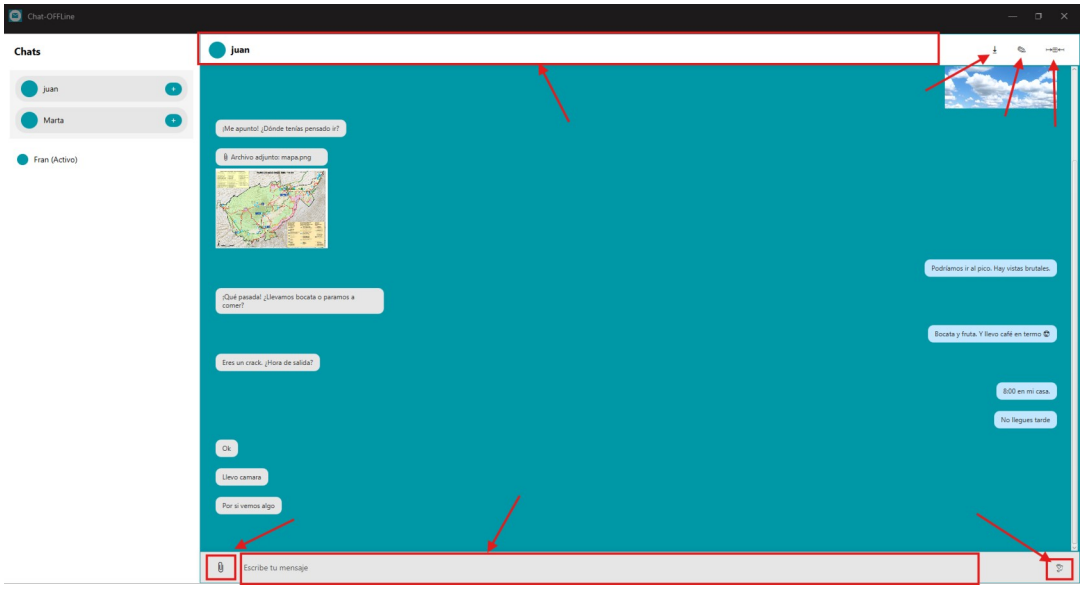
Sexta pantalla: Conversacion

Empezando por arriba si clicamos en la barra superior donde se encuentra el nombre nos llevará a la pantalla de perfil de usuario antes comentada.

Un poco mas a la derecha tenemos tres botones, cada uno realiza una función (Exportar conversacion a csv, generar un resumen del chat y empaquetar la conversación en un archivo comprimido).

En la parte de abajo tenemos, el boton de adjuntar archivo que nos abrirá la ventana del explorador de nuestro SO para elegir el archivo.

A la derecha del boton adjunto podemos escribir y enviar los mensajes.



2. Fase de desarrollo

Fase inicial:

Para empezar a desarrollar este proyecto primero diseñe toda la interfaz gráfica, primero en Canva para tener una idea del diseño y entender la lógica del proyecto, despues empeze a crear la estructura del proyecto, sus clases, paquetes, atributos, métodos que necesitaría y planificación.

Una vez tengo la estructura del proyecto, diseño la interfaz gráfica completamente, para saber que métodos me van a hacer falta y tener una guía clara, de qué hacer, qué metodos tengo que crear métodos..

Fase intermedia:

Con todo lo anterior, esta fase es puramente de escribir código, test, correción de errores,hasta acabar finalizar el programa completamente y que funcione correctamente y de acorde al planteamiento inicial.

Fase final:
Documentación de código, últimas revisiones y documentación como esta, diagramas de clases etc...

2.1 Enlace a Git

https://github.com/FranciscoSolerAguado/ClinicaDental.git

3. Funcionalidades a destacar

Uso de la clase LocalDateTimeAdapter, que sirve como un traductor entre el formato de fecha y hora de Java (LocalDateTime) y una representación de cadena (String) que puede ser fácilmente almacenada en un archivo XML.

```
public class LocalDateTimeAdapter extends XmlAdapter<String, LocalDateTime> {  4 usages  FranciscoSolerAguado

    private static final DateTimeFormatter formatter = DateTimeFormatter.ISO_LOCAL_DATE_TIME;  2 usages

    @Override  FranciscoSolerAguado
    public LocalDateTime unmarshal(String v) throws Exception {
        return LocalDateTime.parse(v, formatter);
    }

    @Override  FranciscoSolerAguado
    public String marshal(LocalDateTime v) throws Exception {
        return v.format(formatter);
    }
}
```

Uso de la clase LoggerUtil para mostrar información por terminal mientras se ejecuta el programa, muy útil para detectar errores

```java
public class LoggerUtil {  10 usages  FranciscoSolerAguado
    private static final Logger logger = Logger.getLogger( name: "ClinicaLogger");  3 usages

    static {
        try {
            // Configurar el FileHandler para escribir en clinica.log
            FileHandler fileHandler = new FileHandler( pattern: "chat-offline.log",  append: true);
            fileHandler.setLevel(Level.INFO); // Nivel de registro
            fileHandler.setFormatter(new SimpleFormatter()); // Formato simple
            logger.addHandler(fileHandler);

            // Configurar el nivel global del logger
            logger.setLevel(Level.INFO);
        } catch (IOException e) {
            System.err.println("Error al configurar el logger: " + e.getMessage());
        }
    }

    public static Logger getLogger() { return logger; }
}
```

Uso de la clase ZipOutPutStream para crear archivos comprimidos.

```java
/**
 * Empaqueta la conversación actual, incluyendo el historial de mensajes y todos los archivos adjuntos,
 * en un único archivo ZIP.
 */
@FXML  FranciscoSolerAguado
private void empaquetarConversacionEnZip() {
    LOGGER.info( msg: "Empaquetando conversación en ZIP.");
    File conversacionFile = getConversacionesFile();
    if (conversacionFile == null || !conversacionFile.exists() || gestorMensajesVacio(conversacionFile)) {
        Utils.mostrarAlerta( mensaje: "No hay conversación para empaquetar.");
        return;
    }

    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Empaquetar Conversación en ZIP");
    fileChooser.setInitialFileName("conversacion_" + contactoActual.getNombre() + ".zip");
    fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter( s: "ZIP (*.zip)",  ...strings: "*.zip"));
    File zipFile = fileChooser.showSaveDialog( window: null);
    if (zipFile == null) return;

    GestorConversacion gestor = XMLManager.readXML(new GestorConversacion(), conversacionFile.getAbsolutePath());
    List<Mensaje> mensajes = gestor.getMensajes().stream()
            .filter( Mensaje m -> (m.getRemitente().equals(usuarioActual.getNombre()) && m.getDestinatario().equals(contactoActual.getNombre())) ||
                    (m.getRemitente().equals(contactoActual.getNombre()) && m.getDestinatario().equals(usuarioActual.getNombre())))
            .sorted(Comparator.comparing(Mensaje::getFechaEnvio))
            .collect(Collectors.toList());

    try (ZipOutputStream zos = new ZipOutputStream(new FileOutputStream(zipFile))) {
        // Añadir el historial de chat
        zos.putNextEntry(new ZipEntry( name: "conversacion.txt"));
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        for (Mensaje m : mensajes) {
            String linea = String.format("[%s] %s: %s\n", m.getFechaEnvio().format(formatter), m.getRemitente(), m.getContenido());
            zos.write(linea.getBytes(StandardCharsets.UTF_8));
        }
        zos.closeEntry();
```