



Proyecto Unidad 1 Acceso a Datos

Chat offline XML



Proyecto Final

Ciclo Superior Desarrollo Aplicaciones Multiplataforma

Mayo de 2024

Autor:

Juan María Ariza Serrano

Profesores:

Maria del Carmen Palacios Gutierrez

Ariza Serrano, Juan María. (2024). *Proyecto Chat Acceso a Datos*. Instituto IES Francisco de los rios.

RESUMEN

En este proyecto de la unidad 1 de acceso a datose, se ha aplicado todo lo aprendido durante este primer mes. Este trabajo ha implicado el uso de de la clase File y Stream así como JAXB que es la librería de Java para trabajar con archivos XML. Se utilizó para crear las clases que representan los elementos del archivo XML donde se guardan los mensajes, usuarios, etc. así como los métodos para leer y escribir en dicho archivo.

Índice

1. Introducción.....	2
2. Descripción.....	2
3. Fase de diseño.....	2
3.1. Clases principales.....	3
3.2. Diagrama de clases.....	5
3.3. Diseño de pantallas.....	6
4. Fase de desarrollo.....	9
4.1. Temporalización.....	9
4.2. Explicación de los puntos fuertes del diseño.....	9
5. Lista de comprobación.....	10
5.1. Lectura y escritura en XML.....	10
5.2. Registro de usuarios.....	11
5.3. Envío de mensajes.....	12
5.4. Consulta de conversaciones.....	13
5.5. Streams para análisis de conversación.....	13
5.6. Exportar conversación.....	15
5.7. Interfaz.....	16
6. Fase de Presentación.....	16

1. Introducción

Este proyecto es una Aplicación desarrollada como parte de la asignatura de Acceso a Datos. La aplicación permite a los usuarios enviar y recibir mensajes entre sí dentro del mismo sistema. Los mensajes se almacenan en un archivo XML y se actualizan automáticamente en las interfaces gráficas de los usuarios.

2. Descripción.

Se trata de una Aplicación de chat, diseñada para facilitar la comunicación entre usuarios. La aplicación permite a los usuarios enviar y recibir mensajes de forma sincronizada, almacenando toda la información en archivos XML.

La aplicación cuenta con una interfaz gráfica intuitiva que simplifica la interacción con el usuario. Las principales características del proyecto incluyen:

- **Gestión de Chats:** Los usuarios pueden iniciar, mantener y gestionar conversaciones con sus contactos, creando una experiencia de chat personalizada.
- **Envío y Recepción de Mensajes:** Permite a los usuarios enviar mensajes instantáneamente, con actualizaciones en tiempo real en la interfaz gráfica, asegurando que todos los participantes en la conversación estén al tanto de las comunicaciones.
- **Añadir Amigos:** Los usuarios pueden agregar contactos fácilmente a su lista de amigos, facilitando la interacción y el envío de mensajes.
- **Exportación de Conversaciones:** Opción para exportar las conversaciones seleccionadas a archivos .txt, lo que permite a los usuarios guardar y revisar sus chats fuera de la aplicación.
- **Análisis de Mensajes:** Implementación de funcionalidades que utilizan Java Streams para analizar y procesar mensajes, ofreciendo insights sobre las conversaciones.

3. Fase de diseño

Al iniciar el diseño de la aplicación, surgió una pregunta clave: cómo abordar los primeros pasos del desarrollo. Es esencial tener una visión clara de la estructura y funcionalidad del proyecto desde el principio. Una vez establecida esta base, se procede a trabajar primeramente en papel y así definir cómo interactúan los diferentes elementos dentro de la aplicación. Posteriormente, se diseña la interfaz gráfica, enfocándose en la usabilidad y en la experiencia del usuario.

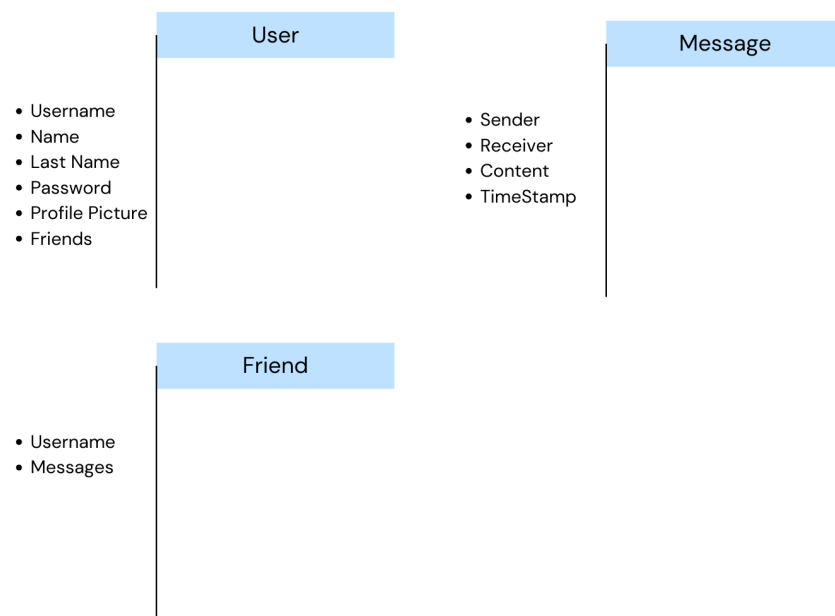
El programa se divide en dos partes principales, cada una con un propósito específico:

- **Interfaz de Chat Principal:** Esta parte central de la aplicación proporciona un acceso intuitivo a las conversaciones activas. Los usuarios pueden seleccionar amigos de su lista de contactos, ver el historial de mensajes, enviar nuevos mensajes y exportar conversaciones a archivos .txt. Además, se integran funciones de análisis que utilizan Java Streams para ofrecer estadísticas de las interacciones. Esta sección es fundamental, ya que constituye la esencia de la aplicación, permitiendo a los usuarios comunicarse de manera efectiva y gestionar sus interacciones.
- **Perfil de Usuario:** En esta sección, los usuarios pueden ver sus datos de perfil y la lista de amigos. Aunque esta parte es menos funcional en comparación con la interfaz de chat principal, proporciona a los usuarios un espacio para revisar información personal y gestionar su red de contactos. A pesar de su menor importancia, es esencial para la experiencia general del usuario al ofrecer un sentido de identidad dentro de la aplicación.

La combinación de estas dos secciones garantiza que los usuarios no solo puedan comunicarse de manera efectiva, sino también gestionar sus contactos y revisar sus conversaciones de forma sencilla.

3.1. Clases principales

En la aplicación, se han definido tres clases principales: User, Message y Friend. La clase User representa a los usuarios y encapsula su información personal, mientras que Message gestiona los atributos de los mensajes enviados y recibidos. Ambas clases incluyen su correspondiente DAO (Data Access Object), que maneja la lectura y escritura de datos en archivos XML, conectando eficientemente con los controladores que gestionan la lógica de la aplicación. La clase Friend se ha creado para representar las relaciones de amistad entre usuarios; aunque podría parecer lógico incluir esta funcionalidad directamente en User, la separación en la clase Friend permite una gestión más clara y modular de las relaciones, además de mejorar la organización del código y su mantenibilidad a largo plazo.



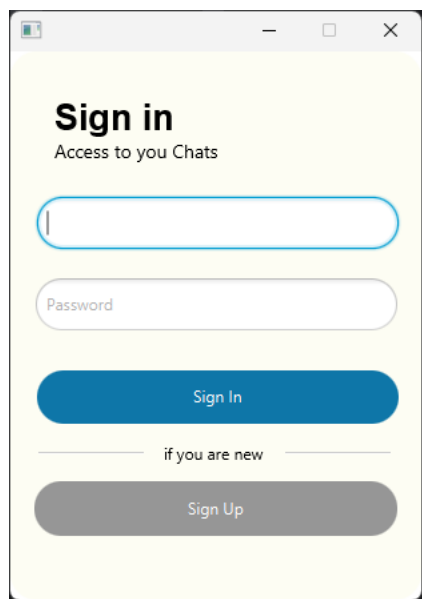
3.2. Diagrama de clases



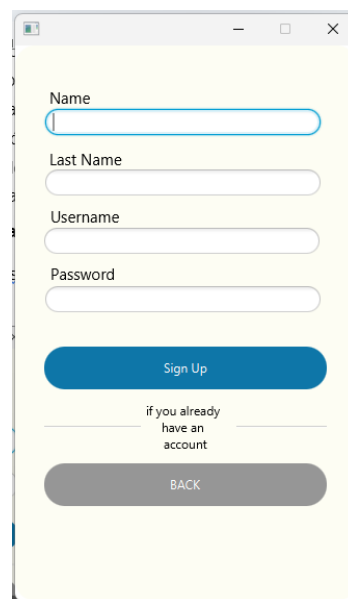
En el diagrama de clases de la aplicación, se puede observar que la mayoría de las clases se centran en los controladores de las vistas, que están estrechamente vinculados a los daos, en este caso los llamo UserDataManager y MessageDataManager. Los DAOs son cruciales, ya que se encargan de gestionar todas las operaciones de almacenamiento y recuperación de datos, así como de ejecutar las funcionalidades requeridas en la aplicación. Esto implica que los controladores de vistas actúan como intermediarios entre la interfaz gráfica de usuario y la capa de acceso a datos, asegurando una comunicación fluida y eficiente entre los distintos componentes del sistema. Además, se incluyen las clases de objetos, que están diseñadas con sus constructores, así como métodos getters y setters, facilitando la manipulación y el acceso a los atributos de cada objeto. Esta estructura modular no solo promueve un código más organizado, sino que también permite una fácil escalabilidad y mantenimiento del sistema a medida que se añaden nuevas funcionalidades o se realizan ajustes en la lógica de negocio.

3.3. Diseño de pantallas

- Primero de todo tenemos que iniciar sesión o registrarse con un usuario nuevo.

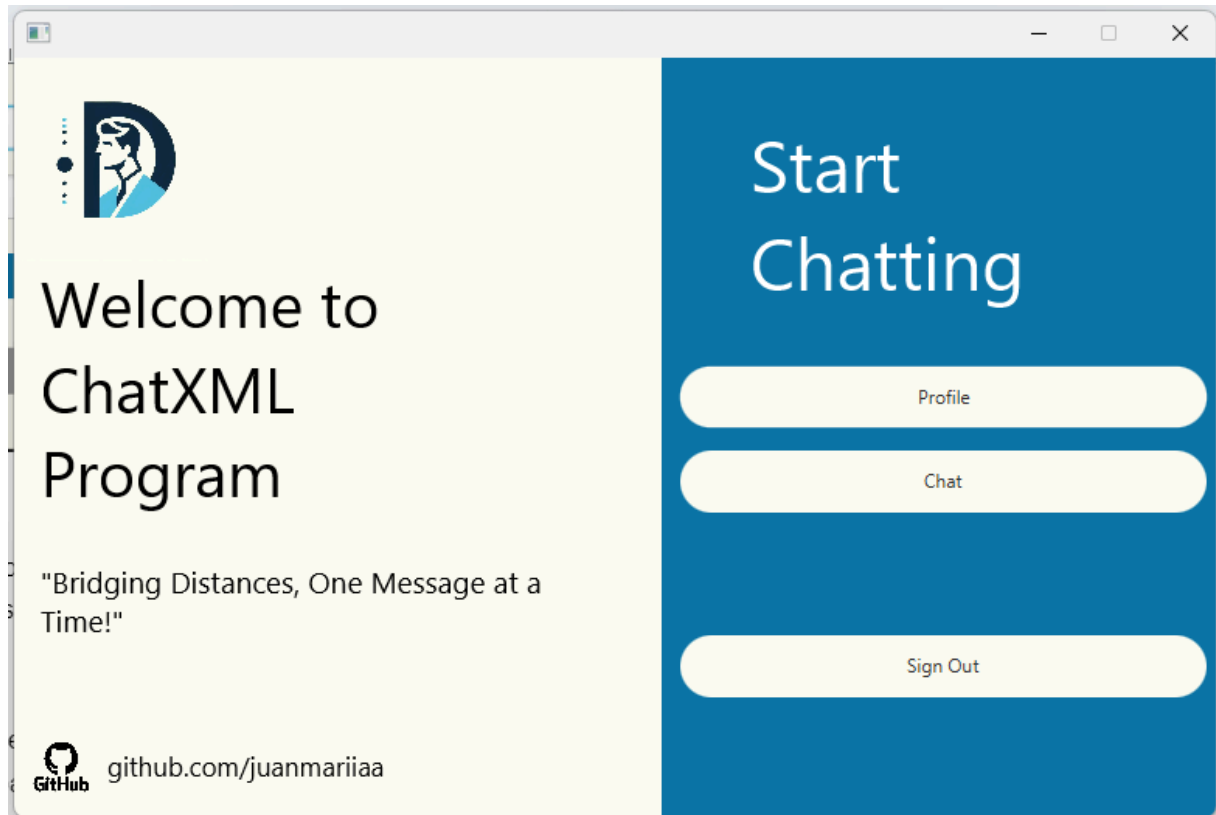


The image shows a mobile app screen for signing in. At the top, it says "Sign in" in bold, followed by "Access to you Chats". Below this are two input fields: one for email/username and one for password. A blue "Sign In" button is positioned below the password field. At the bottom, there is a link "if you are new" followed by a grey "Sign Up" button.

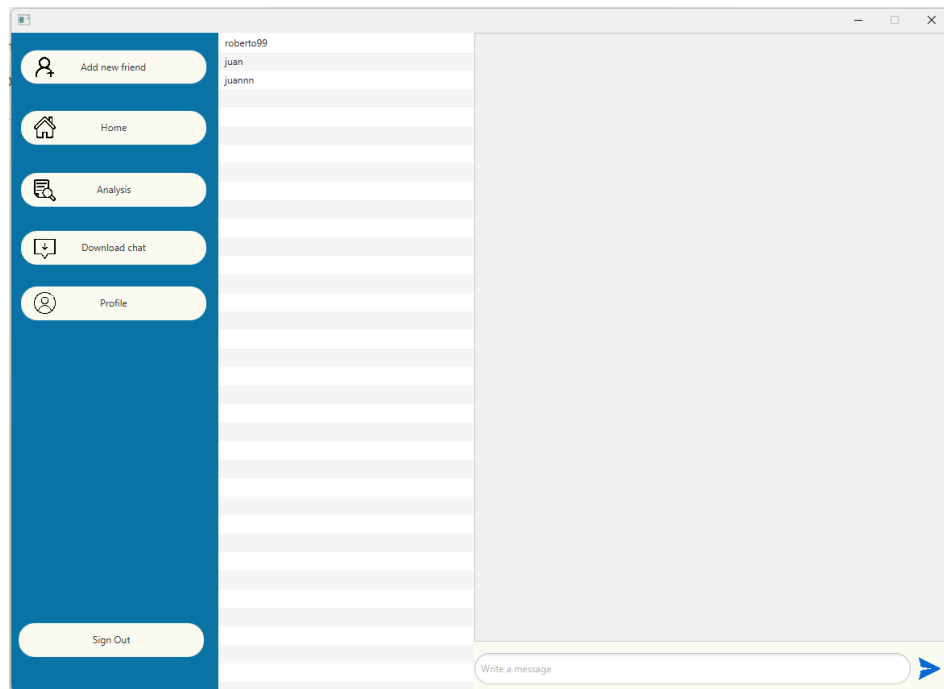


The image shows a mobile app screen for signing up. It features four input fields labeled "Name", "Last Name", "Username", and "Password". Below these fields is a blue "Sign Up" button. At the bottom, there is a link "if you already have an account" followed by a grey "BACK" button.

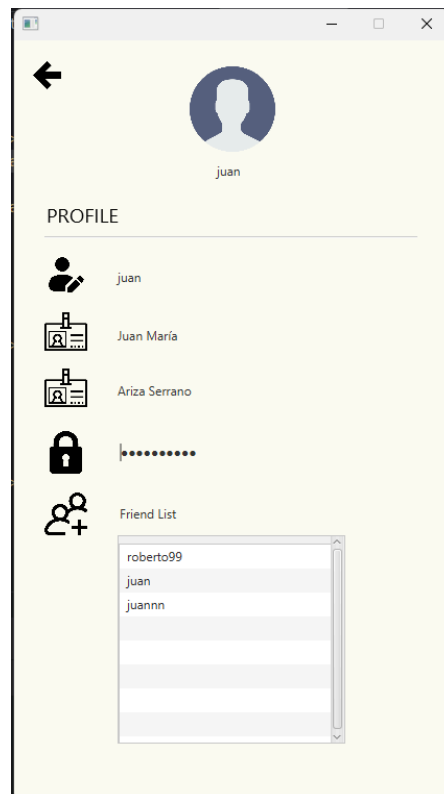
- Segundo tenemos un home, que nos redirecciona o a la parte del perfil o a la parte principal como tal que es la de chat..



- Nos centraremos primero en la parte principal que es la de chat.
- Nos encontramos con una pantalla principal donde a la izquierda de esta tendremos las funcionalidades que podemos hacer así como el intercambio entre diferentes pantallas, pudiendo trasladarte tanto como a cerrar sesión y así irse de nuevo al login, a la pantalla de home y a la pantalla de perfil.
- En el centro de la pantalla nos encontramos la lista de amigos del usuario que tiene iniciada la sesión. Se seleccionarán y se abrirá el chat para poder enviarle mensajes.
- Y por último nos encontramos a la derecha el chat como tal donde se podrán enviar los mensajes al amigo seleccionado.



- Respecto a la pantalla de perfil nos encontramos con los datos del usuario, esta pantalla muestra los datos de perfil y la lista de amigos del usuario. Si bien esta sección carece de las múltiples funcionalidades de la interfaz de chat principal, permite a los usuarios acceder a su información personal y revisar su red de contactos de manera sencilla y organizada.



4. Fase de desarrollo

4.1. Temporalización

Durante la fase de desarrollo de mi proyecto, enfrenté diversos desafíos, especialmente en la sincronización de mensajes. Lo que más me costó fue entender cómo utilizar JAXB para manejar la serialización y deserialización de los mensajes en archivos XML. Hubo momentos en los que me resultó complicado comprender cómo implementar ciertas funcionalidades y cómo hacer que la interfaz respondiera adecuadamente a las acciones del usuario. Además, dediqué muchas horas en casa a experimentar con diferentes enfoques y a consultar recursos en línea para mejorar mi comprensión sobre JavaFX y JAXB.

Hubo ocasiones en las que me sentí atascado durante largos periodos, luchando por implementar ciertas características por resolver errores de código que no sabía cómo abordar. Este proceso de prueba y error fue frustrante, pero al mismo tiempo enriquecedor, ya que cada obstáculo superado me brindó una mejor comprensión de la tecnología utilizada.

4.2. Explicación de los puntos fuertes del diseño

Los puntos fuertes del diseño de mi aplicación se centran en la claridad de la interfaz y su capacidad para facilitar la comunicación entre usuarios de manera eficiente. La interfaz ha sido diseñada para ser intuitiva, lo que permite a los usuarios navegar fácilmente por todas las funciones sin la necesidad de instrucciones complicadas.

Una de las fortalezas clave del diseño es su estructura. Que se enfoca en la interfaz de chat, donde los usuarios pueden seleccionar amigos de su lista de contactos, ver el historial de mensajes, enviar nuevos mensajes y exportar conversaciones a archivos .txt.

Además, he implementado el patrón de diseño Modelo-Vista-Controlador (MVC), lo que proporciona una separación clara entre la lógica de negocio, la presentación y el manejo de datos. Esta estructura no solo hace que el código sea más fácil de mantener y escalar, sino que también permite una mejor organización del proyecto. Las clases están claramente definidas y documentadas, lo que facilita su comprensión y uso. En conjunto, estas decisiones de diseño hacen que la aplicación sea robusta y amigable para el usuario.

<https://github.com/juanmariiaa/ChatXML>

5. Lista de comprobación.

5.1. Lectura y escritura en XML.

En mi aplicación, la lectura y escritura de datos en XML se gestionan mediante JAXB, lo que permite la persistencia de usuarios, mensajes y amigos. Cada clase principal, como User, Message, está anotada con `@XmlRootElement`, lo que facilita la conversión de objetos a elementos XML.

Un Manager se encarga de la serialización y deserialización de estos datos. Utiliza Marshaller para guardar las instancias en un archivo XML y Unmarshaller para leer los datos y recrear los objetos. Esto asegura que la comunicación entre los usuarios se almacene de manera eficiente y estructurada, manteniendo la integridad y disponibilidad de la información.

```
@XmlRootElement
@XmlType(propOrder = {"username", "name", "password", "lastName", "profilePic", "friends"})
public class User {
    8 usages
    private String username;
    5 usages
    private String password;
    5 usages
    private String name;
    5 usages
    private String lastName;
```

```
17 usages
public class UserDataManager {
    8 usages
    private static File file = new File(pathname: "users.xml");

    /**
     * Creates a new XML file if it does not exist and initializes it with an empty UserWrapper
     *
     * @throws Exception If an error occurs during file creation or writing.
     */
    2 usages
    private static void writeXMLUser() throws Exception {
        if (!file.exists()) {
            file.createNewFile();
        }
        UserWrapper wrapper = new UserWrapper();
        wrapper.setUsers(new ArrayList<>());
        JAXBContext context = JAXBContext.newInstance(UserWrapper.class);
        Marshaller marshaller = context.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        marshaller.marshal(wrapper, file);
    }
}
```

5.2. Registro de usuarios.

La clase `UserDataManager` gestiona el registro de nuevos usuarios a través del método `addUser(User user)`. Este método verifica si el archivo XML de usuarios existe; si no, lo inicializa. Luego, recupera la lista actual de usuarios y añade el nuevo usuario a esa lista, para finalmente guardar los cambios en el archivo XML.

Al iniciar sesión, además, se utiliza un patrón Singleton para la clase `UserDataManager`, asegurando que haya una única instancia de esta clase a lo largo de la aplicación. Esto es fundamental para mantener la coherencia de los datos y evitar la creación de múltiples instancias que puedan llevar a condiciones de carrera o inconsistencias en la información de los usuarios.

```
public static void addUser(User user) throws Exception {  
    if (!file.exists()) {  
        writeXMLUser();  
    }  
    List<User> users = getAllUsers();  
    users.add(user);  
    saveUsers(users);  
}
```

```
public class SingletonUserSession {  
    3 usages  
    private static SingletonUserSession instance;  
    6 usages  
    private User user;  
  
    1 usage  
    private SingletonUserSession() {}  
    |  
    5 usages  
    public static SingletonUserSession getInstance() {  
        if (instance == null) {  
            instance = new SingletonUserSession();  
        }  
        return instance;  
    }  
    1 usage  
    public static void login(User user) {  
        SingletonUserSession session = getInstance();  
        session.user = user;  
    }  
    2 usages  
    public static void logout() {  
        SingletonUserSession session = getInstance();  
        session.user = null;  
    }  
    public User getUser() {  
        return user;  
    }  
  
    public String getUsername() {  
        return user != null ? user.getUsername() : null;  
    }  
    no usages
```

5.3. Envío de mensajes.

La clase MessageDataManager gestiona el envío de mensajes mediante el método addMessage(Message message). Este método recupera la lista existente de mensajes del archivo XML, añade el nuevo mensaje a la lista y guarda la lista actualizada de mensajes en el archivo.

```
1 usage
public static void addMessage(Message message) throws Exception {
    List<Message> messages = recoverMessages();
    messages.add(message);
    saveMessages(messages);
}

1 usage
public static List<Message> recoverMessages() throws JAXBException {
    if (file.exists() && file.length() > 0) {
        JAXBContext context = JAXBContext.newInstance(MessageWrapper.class);
        Unmarshaller unmarshaller = context.createUnmarshaller();
        MessageWrapper wrapper = (MessageWrapper) unmarshaller.unmarshal(file);
        return wrapper.getMessages();
    } else {
        return new ArrayList<>();
    }
}
```

5.4. Consulta de conversaciones.

La clase `MessageDataManager` permite recuperar todas las conversaciones entre dos usuarios a través del método `getMessagesBetweenUsers(String user1, String user2)`. Este método filtra los mensajes existentes en el archivo XML, verificando cada mensaje para determinar si el remitente y el destinatario coinciden con los usuarios especificados.

Si se encuentran coincidencias, estos mensajes se añaden a una lista que se devuelve al usuario. Esta funcionalidad permite acceder fácilmente a las interacciones pasadas entre dos usuarios, proporcionando un contexto completo de sus conversaciones.

```
*/  
3 usages  
public List<Message> getMessagesBetweenUsers(String user1, String user2) throws Exception {  
    List<Message> allMessages = getAllMessages();  
    List<Message> filteredMessages = new ArrayList<>();  
    for (Message msg : allMessages) {  
        if ((msg.getSender().equals(user1) && msg.getReceiver().equals(user2)) ||  
            (msg.getSender().equals(user2) && msg.getReceiver().equals(user1))) {  
            filteredMessages.add(msg);  
        }  
    }  
    return filteredMessages;  
}
```

5.5. Streams para análisis de conversación.

La clase `ChatAnalyzer` utiliza `Streams` para realizar un análisis detallado de las conversaciones. A través de su método `chatAnalyzer(List<Message> messages, String filePath)`, se llevan a cabo diversas operaciones estadísticas sobre la lista de mensajes proporcionada.

- **Conteo de Mensajes:** Se calcula el número total de mensajes usando `messages.size()`.
- **Conteo de Palabras:** Se determina el total de palabras en todas las conversaciones dividiendo el contenido de cada mensaje y sumando los resultados.
- **Mensajes por Usuario:** Se agrupan los mensajes por remitente utilizando `Collectors.groupingBy()`, lo que permite obtener cuántos mensajes envió cada usuario.
- **Frecuencia de Palabras:** Se obtiene un mapa de frecuencia de palabras en los mensajes, convirtiendo todo a minúsculas y contando las ocurrencias de cada palabra.
- **Palabra Más Común:** Se identifica la palabra más repetida en la conversación utilizando `max()` sobre las entradas del mapa de frecuencia.

- Longitud Promedio de Mensajes: Se calcula la longitud promedio de los mensajes utilizando `mapToInt()` y `average()`.

Finalmente, se escribe un informe con estos resultados en un archivo de texto especificado por el usuario, proporcionando un análisis completo y accesible de la conversación.

```
public void chatAnalyzer(List<Message> messages, String filePath) {
    if (messages.isEmpty()) {
        System.out.println("No messages to analyze.");
        return;
    }

    long totalMessages = messages.size();
    long totalWords = messages.stream() Stream<Message>
        .mapToInt(msg -> msg.getContent().split(regex: "\\s+").length) IntStream
        .sum();

    // Count messages by user
    Map<String, Long> messagesPerUser = messages.stream()
        .collect(Collectors.groupingBy(Message::getSender, Collectors.counting()));

    // Calculate word frequency
    Map<String, Long> wordFrequency = messages.stream() Stream<Message>
        .flatMap(msg -> Arrays.stream(msg.getContent().toLowerCase().split(regex: "\\s+"))) Stream<String>
        .collect(Collectors.groupingBy(word -> word, Collectors.counting()));

    // Find the most common word
    Optional<Map.Entry<String, Long>> mostCommonWord = wordFrequency.entrySet().stream()
        .max(Map.Entry.comparingByValue());

    // Calculate average message length
    double averageMessageLength = messages.stream() Stream<Message>
        .mapToInt(msg -> msg.getContent().length()) IntStream
        .average() OptionalDouble
        .orElse(other: 0);

    // Write the summary to a file
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
        writer.write(str: "Chat Analysis Report:\n");
        writer.write(str: "Total Messages: " + totalMessages + "\n");
        writer.write(str: "Total Words: " + totalWords + "\n");
        writer.write(str: "Average Message Length: " + averageMessageLength + " characters\n");

        messagesPerUser.forEach((user, count) -> {
            try {
                writer.write(str: "Messages by " + user + ": " + count + "\n");
            } catch (IOException e) {}
        });

        mostCommonWord.ifPresent(entry -> {
            try {
                writer.write(str: "Most Common Word: '" + entry.getKey() + "' (" + entry.getValue() + " times)\n");
            } catch (IOException e) {}
        });

        System.out.println("Chat summary generated at: " + filePath);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```


5.6. Exportar conversación.

La clase ChatExporter permite exportar conversaciones entre dos usuarios a un archivo de texto. Aquí están las funciones clave:

- **Inicialización:** Se construye con un MessageDataManager para gestionar la recuperación de mensajes.
- **Selección de Archivo:** El método downloadChat(User currentUser, Friend currentFriend, Stage stage) abre un cuadro de diálogo para que el usuario seleccione dónde guardar el archivo. Si no se ha seleccionado un amigo, se muestra un error.
- **Exportación de Mensajes:** El método exportMessagesToFile(User currentUser, Friend currentFriend, File file) usa un BufferedWriter para crear el archivo.
- **Recupera los mensajes entre los usuarios y escribe cada mensaje en el archivo, formateando la línea con el remitente, la marca de tiempo y el contenido.**
- **Manejo de Errores:** Captura excepciones y muestra mensajes de error si la exportación falla.

```
public ChatExporter(MessageDataManager messageDataManager) { this.messageDataManager = messageDataManager; }

1 usage
public void downloadChat(User currentUser, Friend currentFriend, Stage stage) {
    if (currentFriend == null) {
        Utils.showPopup( title: "Error", header: "No Friend Selected", text: "Please select a friend to download the chat.", Alert.AlertType.ERROR);
        return;
    }

    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Save Chat");
    fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter( description: "Text Files", extensions: "*.txt"));

    File file = fileChooser.showSaveDialog(stage);

    if (file != null) {
        exportMessagesToFile(currentUser, currentFriend, file);
    }
}

1 usage
private void exportMessagesToFile(User currentUser, Friend currentFriend, File file) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
        List<Message> messages = messageDataManager.getMessagesBetweenUsers(currentUser.getUsername(), currentFriend.getUsername());

        for (Message msg : messages) {
            String messageLine = msg.getSender().equals(currentUser.getUsername())
                ? "You (" + msg.getTimestamp() + "): " + msg.getContent()
                : currentFriend.getUsername() + " (" + msg.getTimestamp() + "): " + msg.getContent();
            writer.write(messageLine);
            writer.newLine();
        }

        Utils.showPopup( title: "Success", header: "Chat Exported", text: "The chat has been successfully exported to " + file.getAbsolutePath(), Alert.AlertType.INFORMATION);
    } catch (IOException e) {
        e.printStackTrace();
        Utils.showPopup( title: "Error", header: "Export Failed", text: "An error occurred while exporting the chat. Please try again.", Alert.AlertType.ERROR);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

5.7. Interfaz.

La aplicación ofrece una interfaz gráfica basada en JavaFX. Destaca por su funcionalidad y diseño. La interfaz gráfica utiliza clases de controlador que se encargan de gestionar las acciones del usuario, facilitando una experiencia intuitiva y fluida. Estas clases se aseguran de que la comunicación entre los componentes visuales y la lógica de negocio sea eficiente, permitiendo una navegación sencilla y un acceso rápido a las distintas funcionalidades de la aplicación.

6. Fase de Presentación

<https://github.com/juanmariiaa/ChatXML>