INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# Triangle – Java Application

Project Report


Written by:

**Tasnim KAMOUN - Sofiene BEN YAHIA**

4 IR-SI A1


*VERSION: 25/01/2023*

# Triangle – Java Application

Project Report

Written by:

**Tasnim KAMOUN - Sofiene BEN YAHIA**
4 IR-SI A1

*VERSION: 27/01/2023*

# Table of contents

# Table of Figures

Tasnim Kamoun
Sofiene Ben Yahia
4 IR-SI A1

# 1. Overview: Context and description

## 1.1. Context

November 2022, we received a call for tender from **Thales-Alenia Space** requesting urgently, in a matter of **3 months**, a distributed chat system, multi-user and interactive in real-time. After studying their needs carefully and discussing further details of the project with the manager, we allocated two ambitious software engineering students to prepare a sample of the app, within the deadline specified.

## 1.2. Description

Triangle is a desktop java application. The app is a communication system that allows people in the company to exchange text messages. Triangle supports the dynamicity of the context: it detects and takes into account sudden connections or disconnections of users in runtime.

Depending on the situation, the system architecture changes. It provides different and adapted communication behavior. In the enterprise, when all users are on the proprietary (local) network, the system lacks a centralized architecture. From the user's perspective, we are in charge of developing a service that eases their life. Experts and technicians responsible for administration and deployment on workstations can use the app too.

# 2. Requirement Analysis

To understand further the need of our client, we did a requirement analysis to clarify our terms.

## 2.1. Functional Requirements

- The user can create an account.
- The user can log in to the account.
- The user can log out from the account.
- The user can view users connected to the local network as one-self.
- The user can start a conversation with any of the connected users.
- The user can receive a message from any other connected user.
- The user can communicate with as many users.
- The user is notified when a message is received from another connected user.
- The user can change his/her pseudo when logged in. The other users are notified of the change.
- An admin can deploy, configure and set up the app.

## 2.2. Non-functional Requirements

- Decentralized system: no central server to handle the communications. Every host has to manage his/her communications.
- Local databases: login and message history are stored locally.
- Unique pseudo: No two users can connect simultaneously using the same Pseudo.
- Unique ID: MAC address is used to identify the user.

# 3. Conception, Architecture, and Technology choices

In this section, we will elaborate on the key strategic conceptual decisions we made after the requirement analysis.

## 3.1. Diagrams

### 3.1.1. Use case diagram

We divided the most important functions required from the app into two use case diagrams:

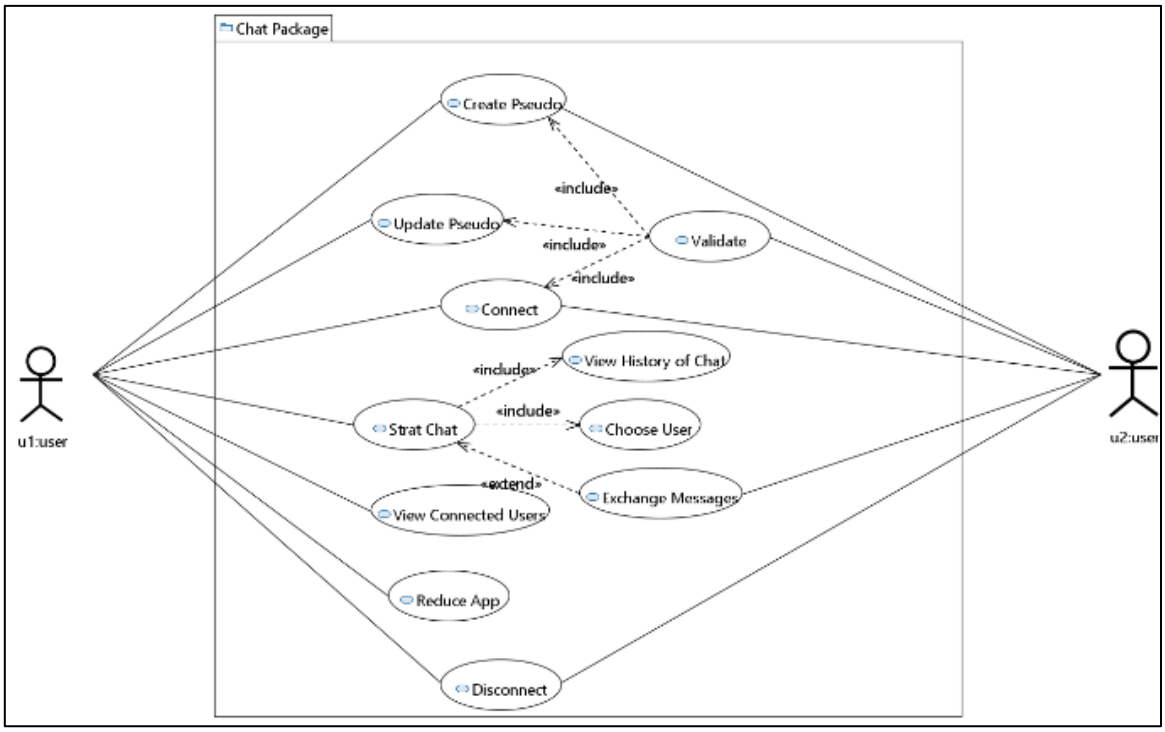


*Figure 1. Chat use case*

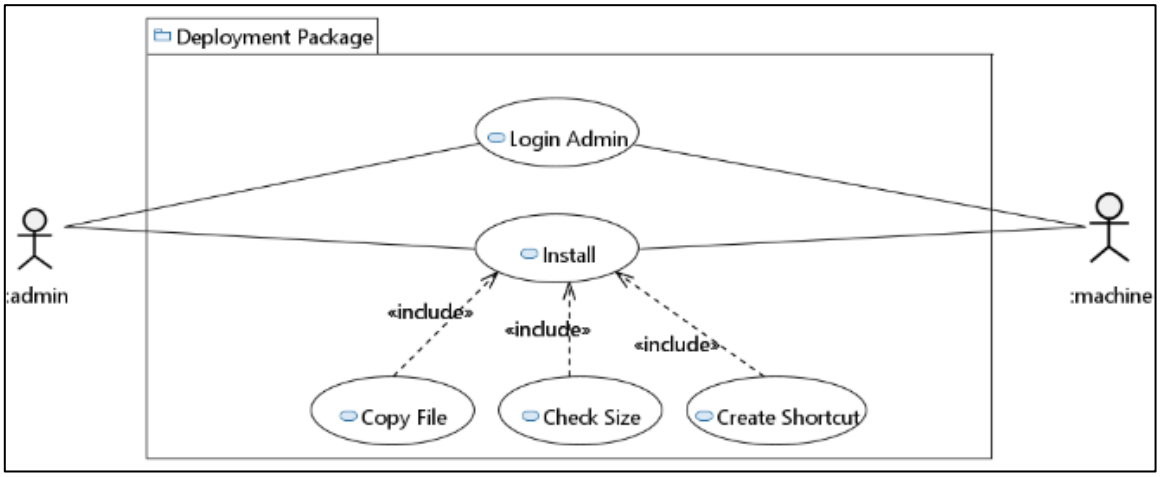


*Figure 2. Deployment use case*

### 3.1.2. Class Diagram

We chose the Model-View-Controller Model to structure our project.

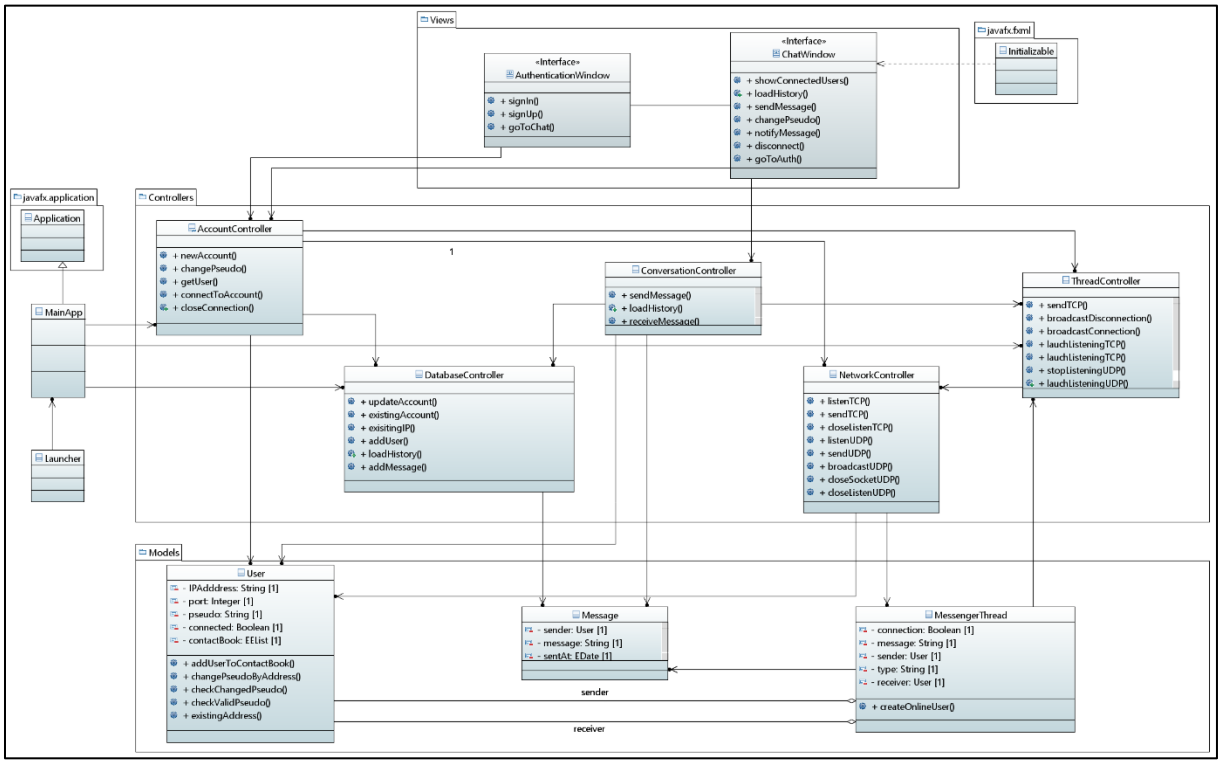


*Figure 3. Class diagram used for this project.*

Please note that:

- Launcher is the class that launches the main of our application.
- For simplicity and visibility reasons, the diagram in *Figure 3* covers the most important associations, packages, attributes, and methods.
- We chose to use an observer pattern to update our views. (view *Figure 4*)

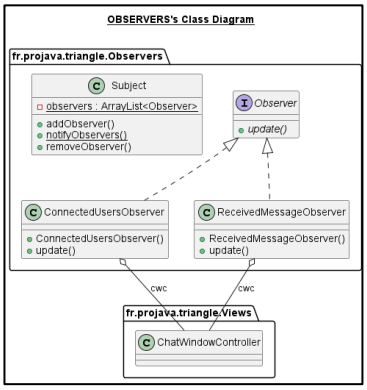


*Figure 4. Class diagram of the observer*

### 3.1.3. Sequence Diagrams

To proceed with the conception, we generated sequence diagrams to show and process interactions arranged in time between the classes, for each separate function.

Please note that these diagrams cover the most important interactions for simplicity and visibility reasons. The elements are merely conceptual and some of them might be implemented slightly differently in the code (different method calls, different class calls…).

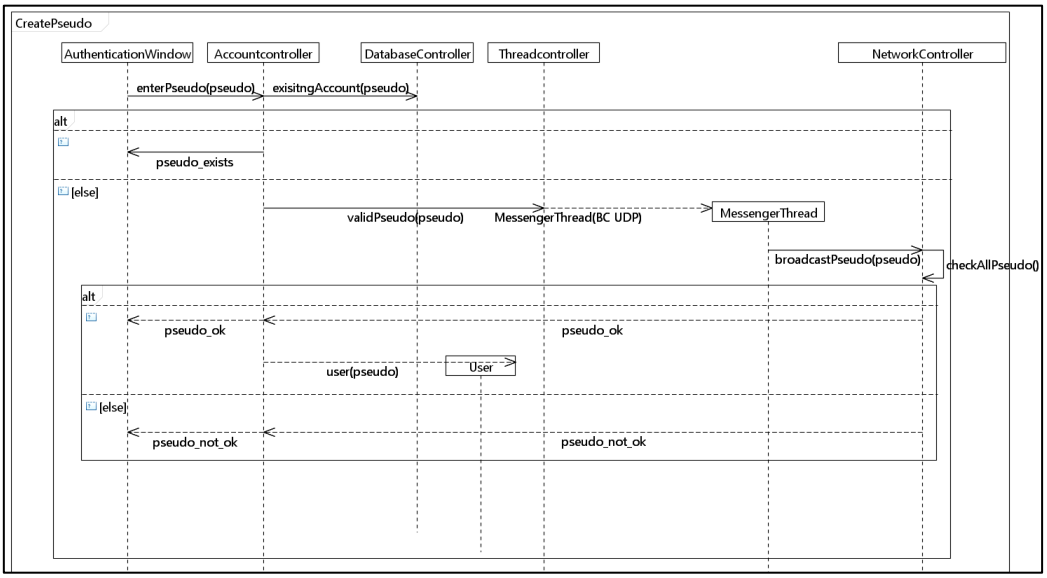


*Figure 5. Sequence diagram of the connection to the account function*

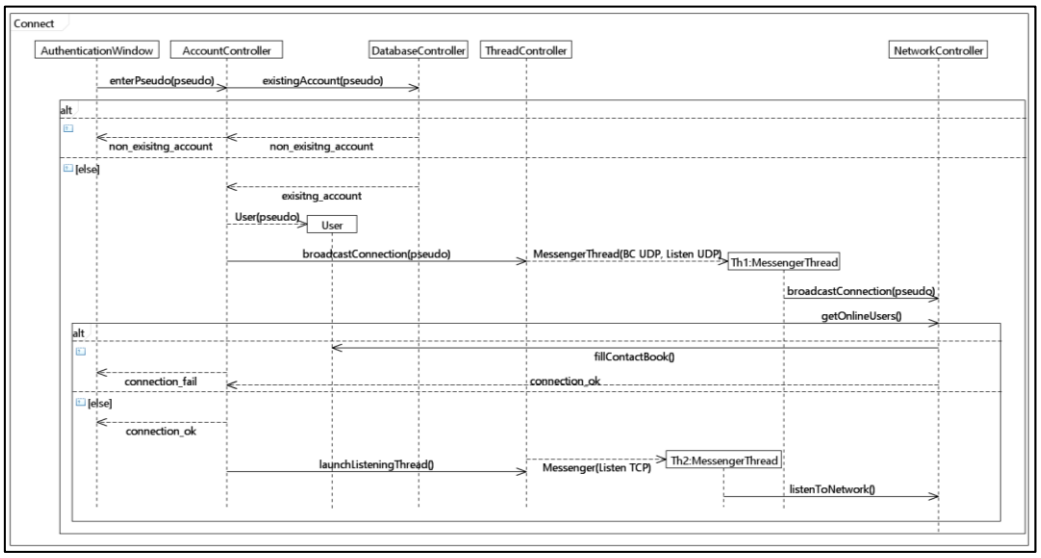


*Figure 6. Sequence diagram of the connection function*

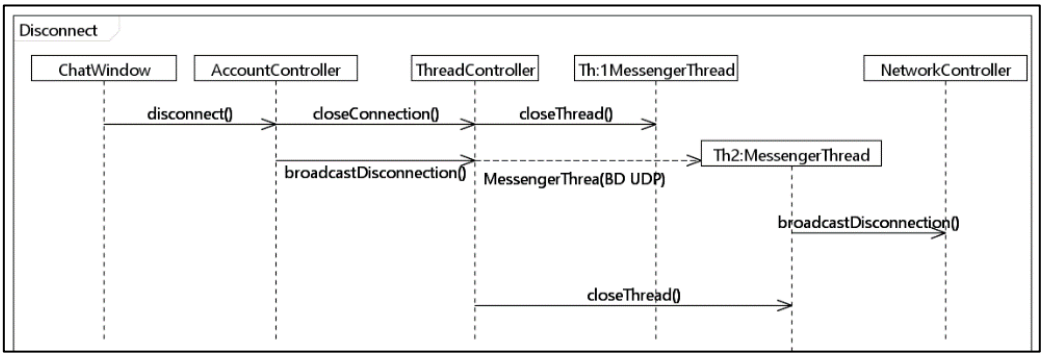


*Figure 7. Sequence diagram of the disconnection function*

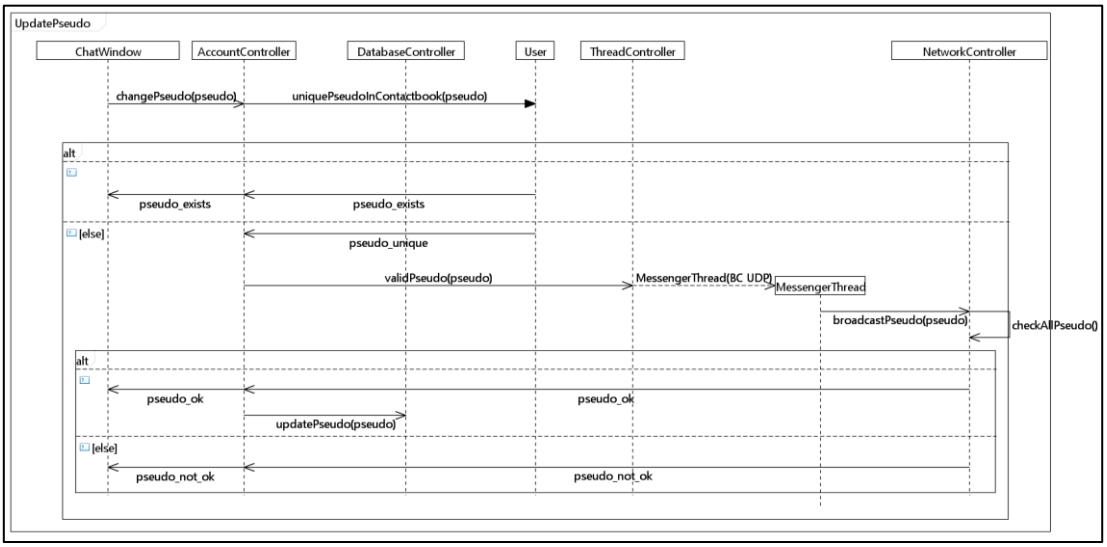


*Figure 8. Sequence diagram of the pseudo-changing function*

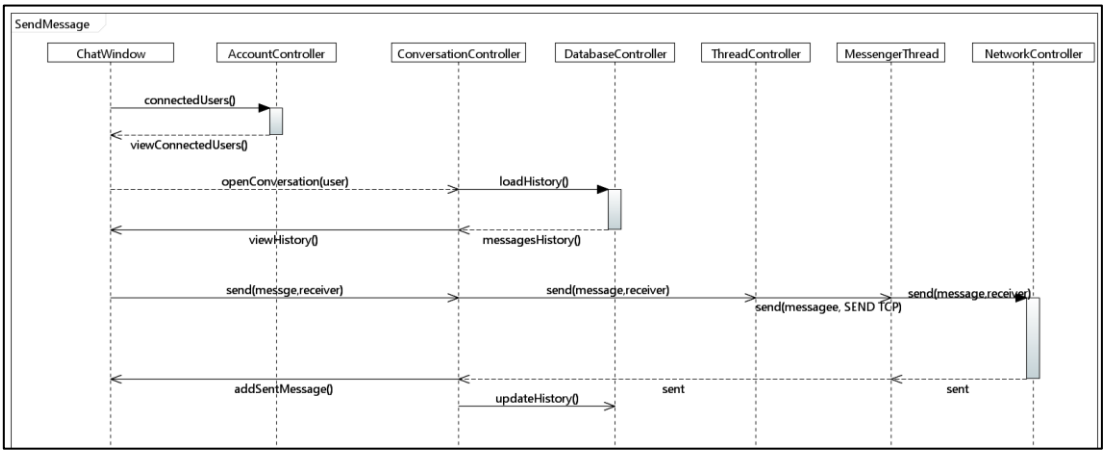


*Figure 9. Sequence diagram of sending message function*

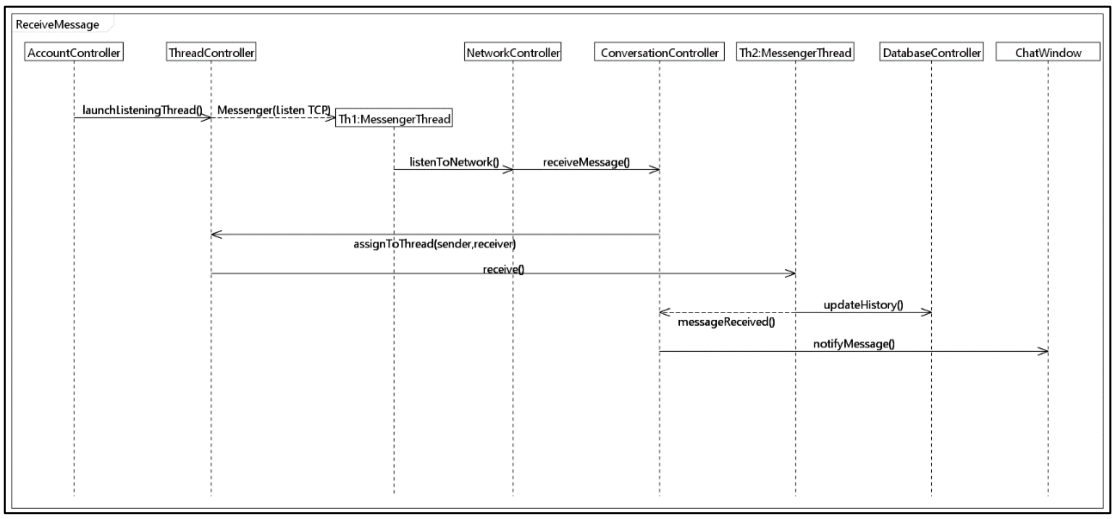


*Figure 10. Sequence diagram of receiving message function*

### 3.1.4. Component Diagram

We used this diagram to illustrate the structure of our chat system.

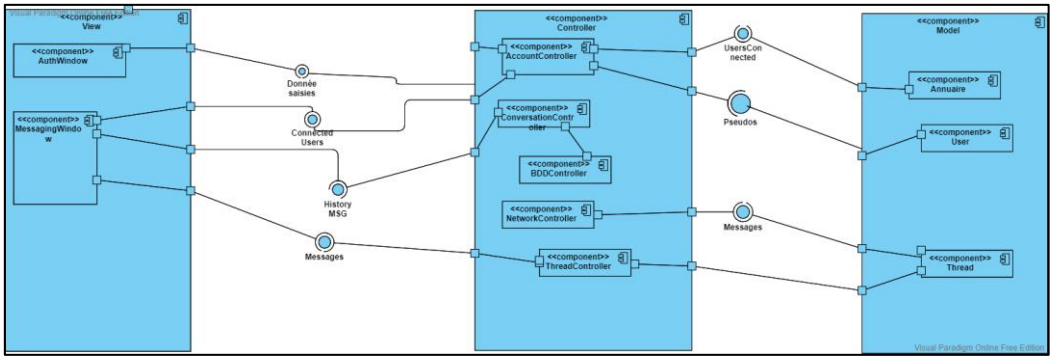


*Figure 11. Component Diagram of our chat system*

### 3.1.5. Deployment Diagram

Here is the deployment diagram, which models the physical deployment of artifacts on nodes.

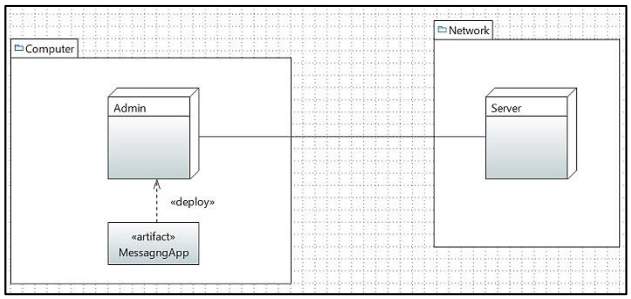


*Figure 12. Deployment Diagram of our chat system*

## 3.2. Database Diagram

The database of our system is local: each user has a database with the login table of users and the history of messages table. The Boolean **"Sender"** is used to specify if the host is the sender (1) or receiver (0) of the message.

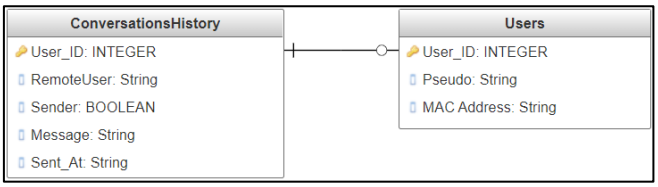


*Figure 13. Database Diagram*

## 3.3. Tools and Technologies and employed

To answer different specs of the app, we made a couple of technology decisions:

- Backend: **Java 11** → general-purpose, object-oriented language intended to let programmers write once, and run anywhere (WORA).
- Frontend: **JavaFX** → Java client-app platform desktop offering many features.
- Graphic Front building tool: **Scene Builder** graphically handles JavaFX components.
- Database: **SQLite** local and fast database.
- Dependency Management: **Maven** handles build automation and dependencies.


*Figure 14. Tools and Technologies*

- Collaboration and version control: **Git** → Here is the repository to our project.
- Issue, tracking, and product management: **Jira** supports the Agile-working method.

### 3.4. Protocols and Technical Choices

- **Client-Server Model**: Every user is a Client and a Server at the same time.
- **TCP** is used for sending and receiving messages.
- **UDP** is used during the connection phase, the update of the connected users (disconnection, pseudo validation...).
- **Multi-threaded System**: Synchronous and Asynchronous behavior.
- **Observer pattern**: update on runtime → receiving a new message and seeing a live contact book of the connected users.

## 4. Test and Validation Procedure

According to the functionality and test requirements, we adapted tests accordingly Mostly, we used two ways of testing:

- Unit tests for the backend and database management.
- Each MVP of the app was tested manually and its behavior was checked through all use cases.

Our final version was tested using three computers with the application running on each.

Here are the test titles for our test cases. Please, refer to the user manual to learn about test scenarios.

- **Create Account----------100%**
  - Pseudo existing----------100%
  - Successful account creation----------100%
- **Connect to Account----------100%**
  - Pseudo non existing----------100%
  - Account non existing----------100%
  - Pseudo in use----------100%
  - Successful Connection----------100%

- **Change Pseudo----------100%**
  - Pseudo in use----------100%
  - Successful pseudo changing----------100%
- **View Connected users----------100%**
  - Remote user connects----------100%
  - Remote user changes pseudo----------100%
  - Remote user disconnects----------100%
  - Initiate conversation----------100%
  - View history of selected user----------100%
  - New message received and sender not selected----------100%
  - New message received and sender selected----------100%
- **Send Message to user----------100%**
  - Non connected user----------100%
  - Message sent successfully----------100%
- **Disconnect----------100%**
  - Successful Disconnection----------100%
  - Closing app: disconnection----------100%

# 5. Installation and Deployment Procedure

There are many ways to deploy the app. Here are two ways we suggest:

## 5.1. First Method: Executable JAR

1. Download the jar corresponding to your OS: Linux or Windows. (mac version coming soon)

- If you are on Windows → just double-click on the jar.
- If you are on Linux → run `java -jar triangle-linux.jar` on the terminal.

## 5.2. Second Method: Git

1. Clone the git repo using the following command:

- *if you are on a Linux OS :*
  `git clone -b triangle-linux https://github.com/tasnimk11/MessagingApp.git`
- *if you are on a Windows OS :*
  `git clone -b triangle-windows https://github.com/tasnimk11/MessagingApp.git`

2. Move to the root directory of the project: that contains the pom.xml file :

- *Open "**BroadcastAddress**" with any text editor.*
  - *change the broadcast address to that of your network*
  - *if you are on a Linux OS, you can use the `ifconfig` command in the terminal.*
  - *if you are on a Windows OS, you can use the `ipconfig` command in the terminal.*
- *Make sure you use the correct address, in the correct format.*
- *Make sure you preserve the format of the text file.*

3. Open the terminal and run the following maven command: `mvn clean javafx:run`.

- *Make sure you have Java and Maven installed, and the environment variables of Java and Maven configured correctly.*

# 6. Learn about Triangle: Backstory, Interface, and User Manual

## 6.1. Name, Logo, and Theme

We chose "**Triangle"** as the name of our app for various reasons:

- It is simple, easy to remember, and easy to recognize.
- This is our first collaborative project as a team. We value project management and teamwork. The shape of a triangle represents the three main stages of a project: planning, executing, and testing.


*Figure 15. Name of the app*

- It is a reference to the three points of a triangle symbolizing the three main features of the application: authenticating, sending, and receiving.

We chose a logo for the project that is circle-shaped:

- It symbolizes unity, continuity, and cohesiveness of individuals or communities using the application or developing it.
- It is also simple and easily recognizable.

*Figure 16. Logo*

We chose a dark-mode arcade theme (purple, pink, black).

## 6.2. Interface Overview

Once the app is loaded, the default view is displayed:

- Text field to enter the pseudo
- "**Sing up**" button to create an account.
- "**Log in**" button to connect to the account.
- Any error messages will appear above the buttons.


*Figure 17. Authentication view (default view when opening the app)*

Once they logged in successfully, the main view is displayed:

- Pseudo used by the host shows up.
- The connected users are displayed right below the pseudo.
- Pink-colored users have sent messages that are not read yet.
- It is possible to enter new pseudo in the text field down on the left.
- "**Pen**" button may be used to change effectively the pseudo.
- Once the user is selected, the history of the messages is displayed on the right.
- Text field is provided to enter the message to send.
- "**Send**" button on the right of the box to send effectively the message.
- "**Log out**" button may be used to disconnect.

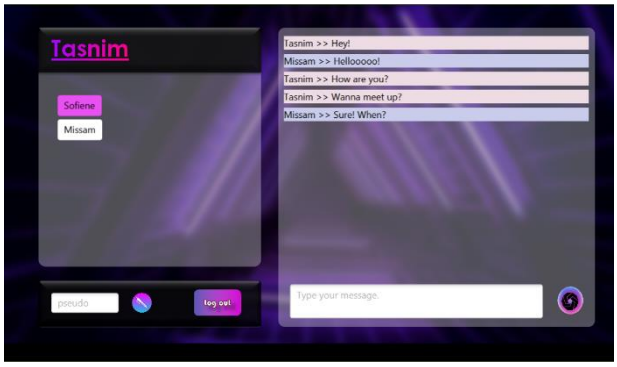


*Figure 18. Main view of the app*

## 6.3. User Manual

1. Authentication window *(Figure17)*
   - ➢ Signing up
     - Create an account by entering a pseudo and hitting the button "**Sign up**".
     - An entry is added to the database with a unique ID according to the MAC address of the computer (implementation ONLY for Linux app).
     - If an account already exists with that precise MAC, an error message is shown "**Existing Account**".
   - ➢ Logging in
     - Enter Pseudo in the box and hit the button "**Log in**".
     - If the pseudo is not linked to any account (wrong, or account not created already), an error message is displayed: "**Account not found**".
     - If another connected user is using the pseudo currently, an error message is displayed: "**Unable to connect**".
     - If pseudo is valid to use, the user is connected: switch to Chat window.
2. Chat Window *(Figure18)*
   - ➢ View Connected users + update user
     - Once logged in, you can view users disconnecting, connecting, and updating their pseudo on runtime.
   - ➢ Initiating Conversation
     - Click on one of the connected users.
     - If an ancient conversion has occurred before, the history is loaded on the right side of the screen.
     - If not, an empty history is printed.

- Send Message
  - Enter the message in the box on the right side and hit the sending button.
  - If the receiver is connected, the text is sent, added to the database, and added to the messages view.
  - If not, an error message is displayed for the user "**User disconnected**".
- Receive Message
  - If the sender is selected: the message is added to the view of the loaded history.
  - If not, the user is notified: the sender is highlighted in the list of the connected users.
- Change Pseudo
  - Enter the New Pseudo in the box and hit the "Pen" button.
  - This change is broadcasted to the connected users.
  - If the Pseudo is unique (at the time of change), it is accepted and the change appears on runtime on the list of the connected users (the table of users is updated).
  - Otherwise, an error message is displayed "Pseudo in use.
- Disconnect
  - Hit the "**Log out**" button to get back to the Authentication Window and notify the connected users of the disconnection.
- Close app
  - Using the closing button, the app stops turning and disconnection is broadcasted automatically.

## 7. Possible Enhancements

This version of the application is fully functional and reliable. It fulfills the requirements of **Thales-Alenia Space.** However, many features can be added to make the desktop even better. Here are some enhancements we suggest and that we will be working on in the future:

- Add pertinent "Enter" actions.
- Add the timestamp of the messages.
- Make it possible to see past messages even if the other user is not connected.
- Link the history table of the database to the ID of the user and not to their IP.
- Some bugs might have slipped in, so they would be fixed along the way.

## 8. Contact

Have you encountered any problems? Do need more information about our app? Do you need any help with the deployment? Do you want to report some bugs? Feel free to contact the contributors:

- Sofiene Ben Yahia (he/him) : benyahiasofiene@yahoo.com
- Tasnim Kamoun (she/ her): tasnimk1108@gmail.com

**INSA Toulouse**
135, avenue de Rangueil
31077 Toulouse Cedex 4 - France
**www.insa-toulouse.fr**

Université Fédérale Toulouse Midi-Pyrénées

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE
DE L'ÉDUCATION NATIONALE,
DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE