INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Peer-to-peer Real-time Multi User Chat System Project Report

28/01/2022- 4IR

Mariétou SARR
Xuan Hai Minh Tran

Peer-to-peer Real-time Multi User Chat System Project Report

Table of Contents

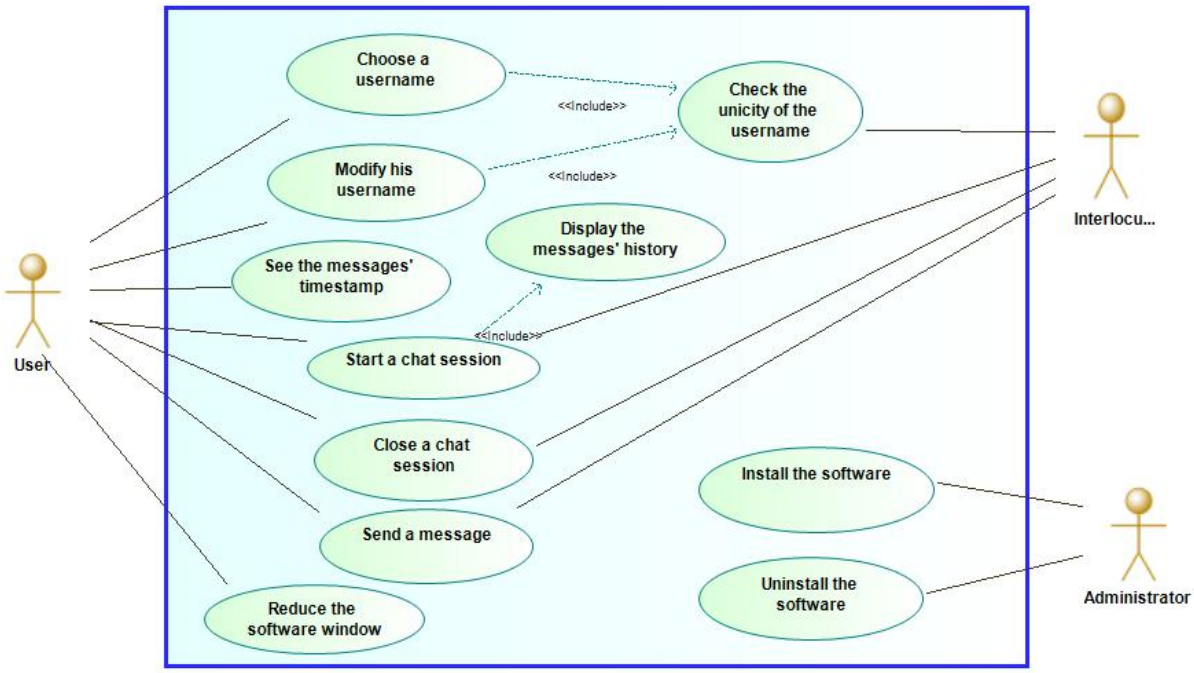
I. Introduction

This report documents our work on the Peer-to-peer Real-time Multi User Chat System Project. It contains initial UML Diagrams we make to design the software, our system architecture and technical choices and a guide to install and use the ChatSystem.

II. Design diagrams

1. Use Case Diagram

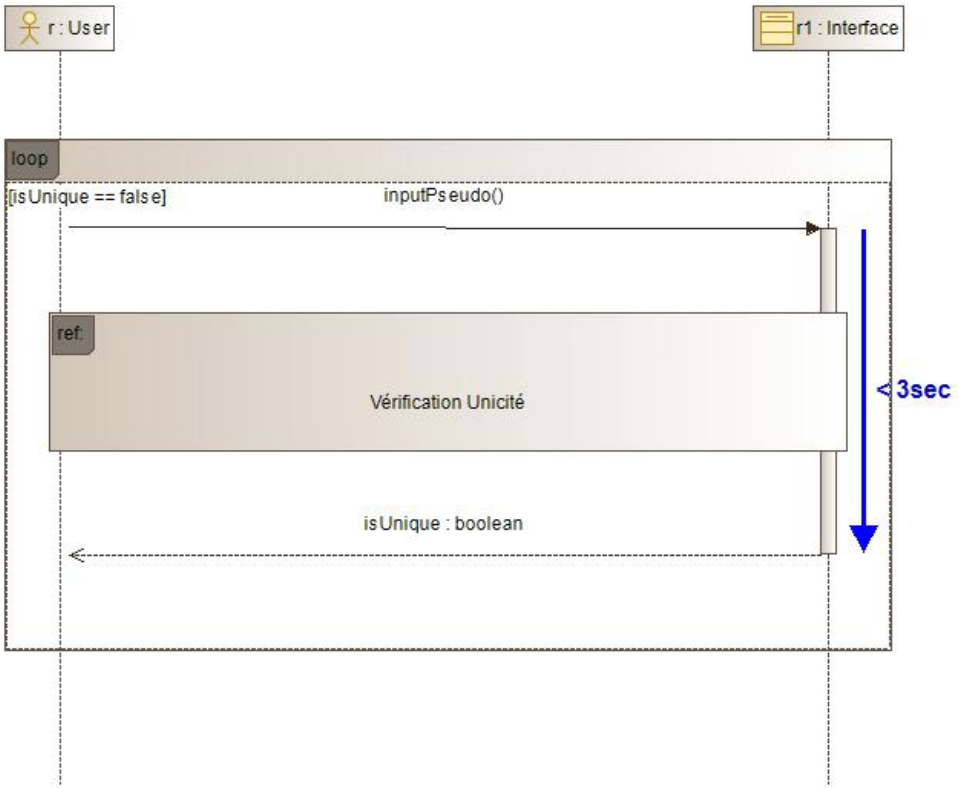
This diagram represents all the functionalities of our project.



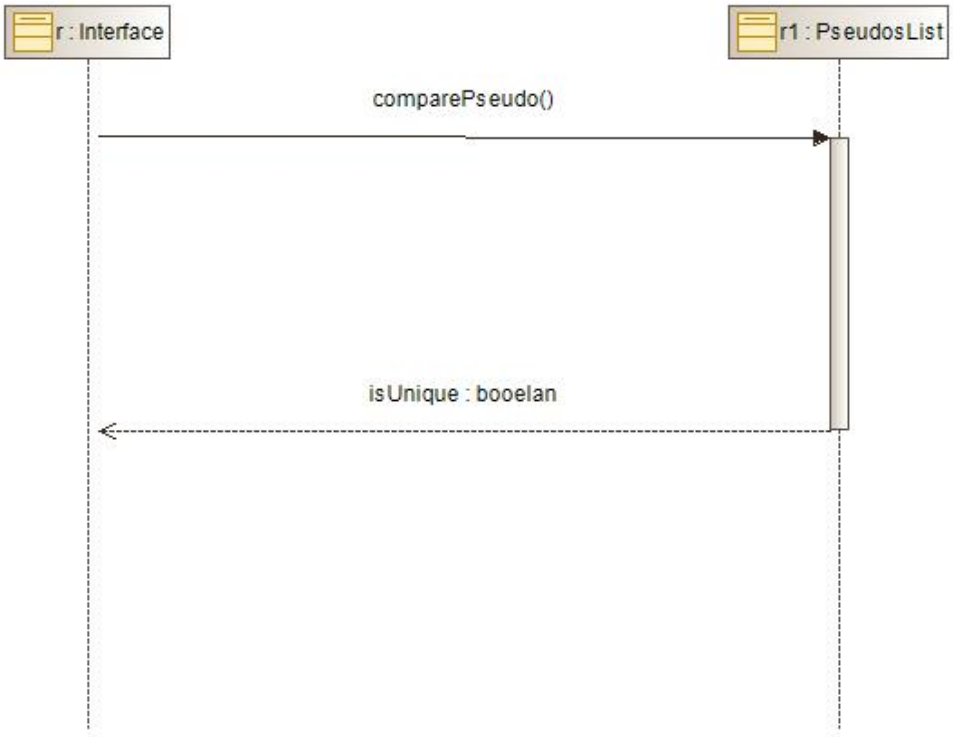
2. Sequence Diagrams

These are the first version of our sequence diagrams. They are not very representative of the final software because we had to modify a lot of things in order to provide a better quality product.

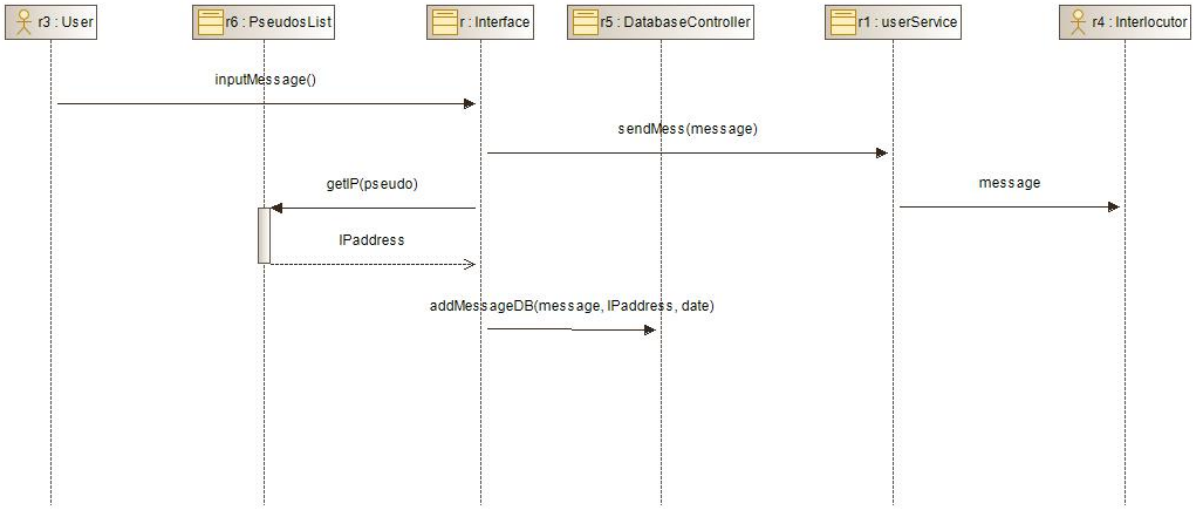
a) Choose a username



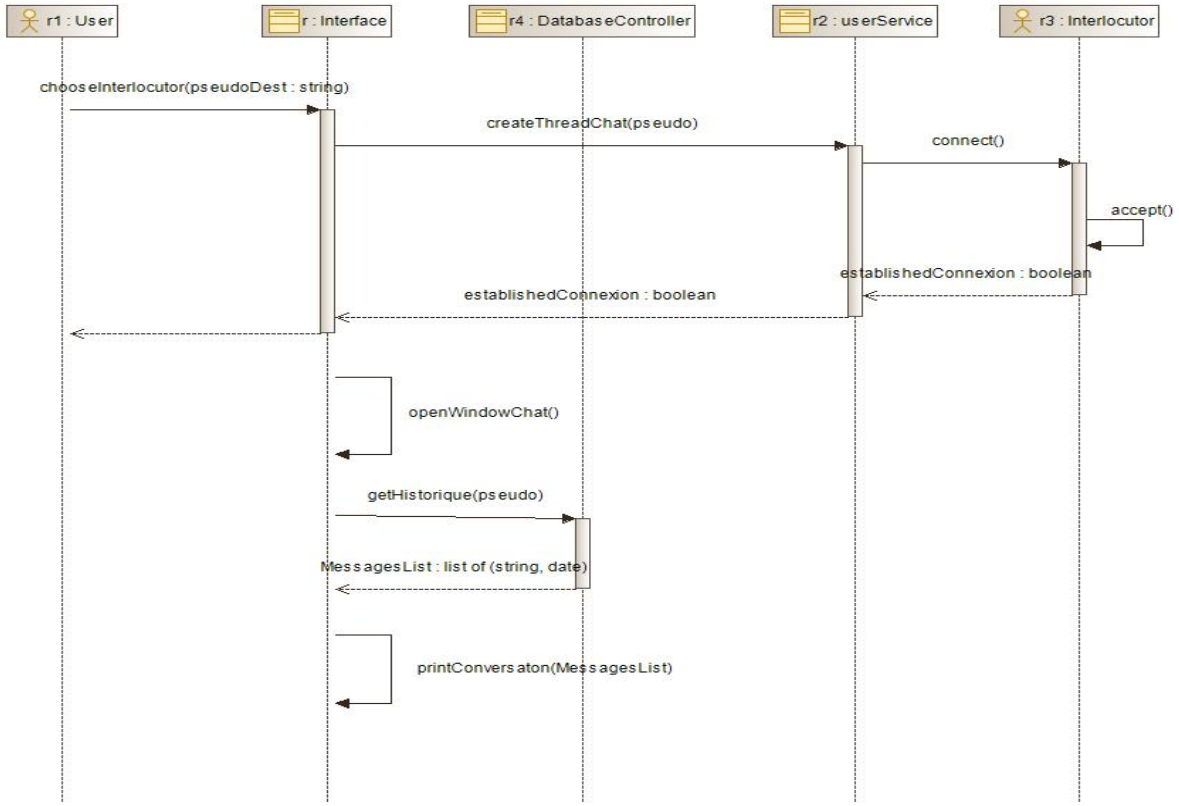
b) Check the unicity of the username



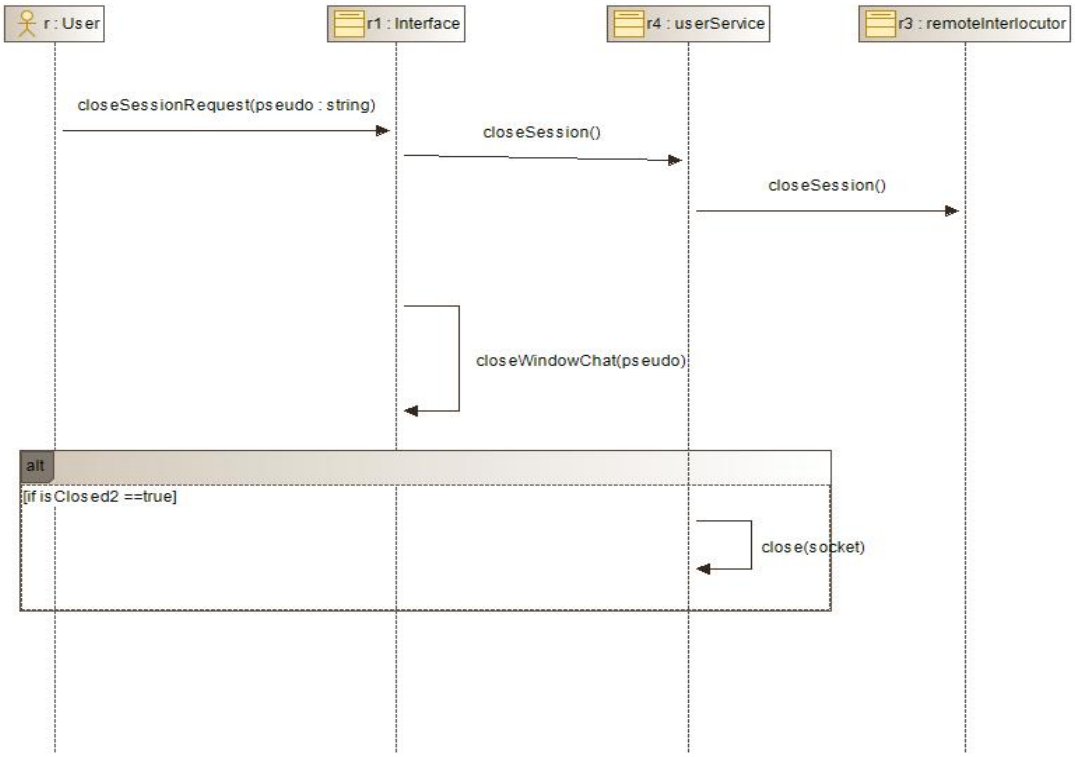
c) Send a message



d) Start a chat session

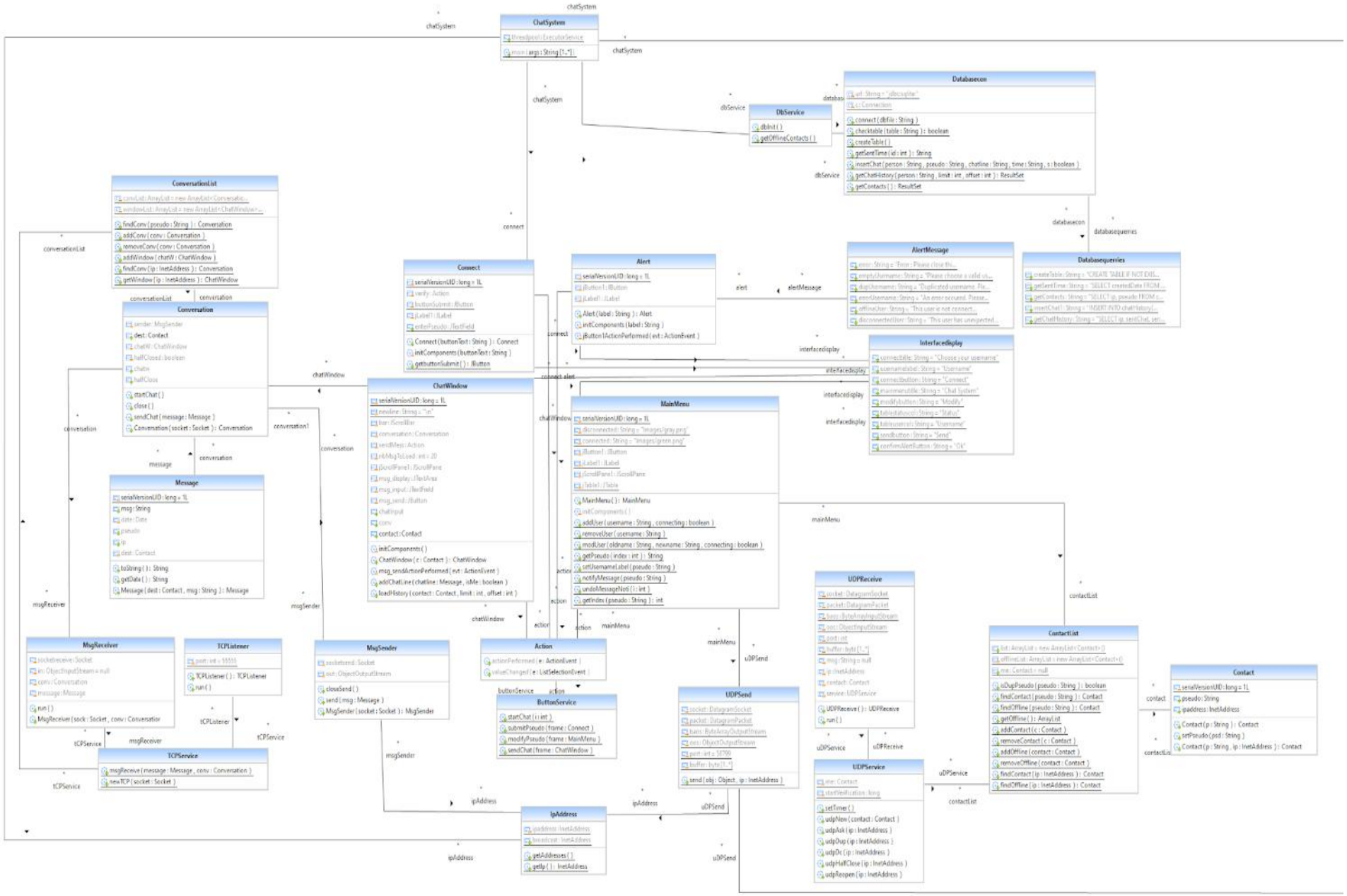


e) Close a chat session



3. Class Diagram

This is the final product Class Diagram.


ChatSystem
ConversationList
Conversation
Message
Connect
Alert
AlertMessage
Databasecon
DbService
Databasequeries
ChatWindow
MainMenu
InterfaceDisplay
MsgReceiver
TCPListener
TCPService
MsgSender
Action
ButtonService
IpAddress
UDPSend
UDPReceive
UDPService
ContactList
Contact

III. System architecture and technical choices

To make our project, we used 5 packages : ***chatsystem*, *entities, userinterface*, *database*, *network*, *service* and *ressources***.

The package ***chatsystem*** contains our main class ChatSystem.java.

The package ***entities*** has our special data structures representing a message (a contact = the destinator, a string = the message and a date) , a conversation (a chat window, a MsgSender, a contact = the interlocutor and a boolean representing the state of the conversation), conversations lists, a contact (a string = username and an IpAddress) and contacts lists.

The package ***userinterface*** contains all the software interfaces with which the user interacts. We have the main software's window MainMenu.java, the chat window ChatWindow.java for every conversation the user has, the connecting window Connect.java for choosing a username when starting the program and when the user wants to modify his username. We used the library Java SWING for these classes.

The package ***database*** contains a class Databasecon.java. It represents our SQLite database (we used sqlite-jdbc). We made a decentralised database. The database is the file ChatSystemHistory.db and it contains all the messages the user has sent or received. Every message is identified with the Ip address of the interlocutor.

The package ***network*** contains the java classes to send and receive UDP datagrams (UDPSend.java and UDPReceive.java) and to send and receive messages via TCP (MsgSender and MsgReceiver). It also contains TCPListener.java that acts as a TCP server (accepting sockets) and the final class is IpAddress.java to make it easier to get the broadcast and the user addresses.

The package ***service*** makes the link between the database, the TCPListeners, Msg Sender and MsgReceiver and our UDP classes and the program entities and interfaces. It also contains 2 classes to manage events from the interfaces (Action.java and ButtonService.java).

The package ***ressources*** has all the database queries, the Strings displayed on our interfaces and the different alerts messages.

IV. How to install and deploy the ChatSystem

To install the ChatSystem, you need to have Maven and Java . We used Maven 3.8.4 and JDK 11 and 14.

You can download the project on this github link : https://github.com/tranxuanhaiminh/ChatSystem with the option *Download ZIP* or clone the repository on your computer.

Then, go to the directory ***chatsystem/***.

If your computer can execute make commands, you can just use the terminal command : *make install*.

If it can't, use this terminal command : *mvn compile package*.

You will now have a new directory named ***target/***.

On it you will find 2 .jar files : ***ChatSystem.jar*** and ***ChatSystem-jar-with-dependencies.jar***.

As we have some dependencies, the project runs with the fat jar ***ChatSystem-jar-with-dependencies.jar***.

You have successfully installed the ChatSystem !
To use the software, you will just need Java and the fat jar ***ChatSystem-jar-with-dependencies.jar***.

V. Simplified User's Guide

This software will allow you to chat with people on the same network or broadcast domain. You must be connected to a network for it to work.

You can launch the software with the commands :

- *make launch* (if your computer can execute make commands)
- *java -jar target/ChatSystem-jar-with-dependencies.jar*
- *java -cp target/ChatSystem-jar-with-dependencies.jar chatsystem.ChatSystem*

When using the software, the first step is to connect. A small window will appear asking for your username. After writing it, you can click on the connect button or push Enter.

If the username is correct, the connecting window will disappear then the main window will appear. If it is not correct, meaning that someone is using the software with that username, an alert window will appear notifying you that you have to change it. You can close it and retry until the value is valid.

On the main window, you will see your username displayed at the top and a button next to it to modify it. To modify your username, you have to click on the button and enter your new username. This process is similar to the connecting phase.
At the centre of the main window, you will be able to see users. These users are connected when their status is green. If it's grey, it means they are disconnected.
By clicking on a username, a window will be opened so that you can discuss with that person if they are connected, elsewise you can look at the messages history but you won't be able to send them messages (an alert will appear if you try).
To send a message you can click on the send button or push Enter after entering your message.
A username in bold means that person has sent a new message.
If you close a chat window with a connected user, you will still be able to receive their messages as long as both of you are connected.

When a user disconnects, his status will change immediately and if you were having a chat session, the window will close. The user will still remain on your users' list.

To close the program correctly, close the main window with its closing button. If you don't do this, you may cause errors to others using the ChatSystem in your network.

An alert window telling you that you have to close the ChatSystem will appear when the system bugs.

The program may close itself if it was already running on your computer.

VI. Project organisation

Here are tools (from our "Gestion de projet" and "Processus de développement automatisé" courses) we used to make working together easier.

1. Git

Link : https://github.com/tranxuanhaiminh/ChatSystem

As it was a team project, GitHub allowed us to work at the same time on different tasks and each task was done on a special git branch. When we finished, we would merge our branch on the master.
We were also working on multiple computers so by pushing our work on the git repository, we were able to get it on another computer.
Git was also very useful when we pushed something that has a lot of errors because we could go back to old versions.

2. Jira

Link : https://test65535.atlassian.net/jira/software/projects/SYS/boards/2/roadmap

With Jira, we were able to organise the project in an efficient way.
As you can see with our project link, we separated the project into 5 sprints, each sprint was about a week. Each sprint had a certain number of tasks.
However, we had a problem with the database as we didn't make it a special task but we put it with sending messages, starting a session etc and because of that we didn't finish every task of some sprints on time. The database took much longer than planned.
The last sprint was for enhancing the project with new functionalities but we used it to optimise our code and didn't have enough time to add the functionalities.
Jira was also a good tool when working on a team project because as we defined tasks in detail, each task was assigned to someone and they knew exactly what they had to do. It was also great to be able to see our progress.

3. Jenkins

We each did 3 Jenkins jobs that were all Maven projects.

Our first job had to scan our GitHub repository and build (mvn clean compile) the project that was on the master branch when something changed, the second job had to build the project on our computer and the last one had to deploy the project (mvn package to create the fat jar).
We created a pipeline between the jobs following the same sequence.
Hence, whenever we pushed something on the git repository, the first job compiles the master branch and if there was no error, the project on our computer would be compiled and the .jar file created.
As we were pushing our work on GitHub even if it was not finished (to be able to pull it on another computer), we had a few compilation errors because of the second job but it was not an issue as it was the unfinished version on our computer which was on a local git branch.

VII. Conclusion

This project allowed us to make a real IT project from scratch. We run through the process of making software starting from the design to the deployment. We also discovered organisation methods for groups projects and uses interesting IT tools to make coding, integrating and working with a team a lot easier.

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