INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
TOULOUSE

Analysis Report

on a local network-
oriented chat application

Written by:

GUITTARD Valentin, JARNAC-
CEZETTE Yohan & SETA Ralaimady

57th promotion of INSA Toulouse

4IR - C1

- January 27, 2023 -

Contents

Conception

1 Introduction

The present document's objectives are to give an overview of our conception methods, alongside diagrams *(use case, class, composite & sequence)* and a short user manual. We developped a chat application whose name is **"CraquApp"**. Our application allows users to communicate *(send messages and all sorts of files)* via TCP connections on a local network. Each user can choose their own pseudo *(must be unique)*, and all messages sent and received to and from another user will be inserted in a database and retrieved even if the user closes then relaunches the application.

Note that all dependencies are included in the pom.xml file (javafx v19, sqlite, maven...).

2 Hypotheses

- 1 IP address = 1 agent = 1 user, which means we can identify a user with the IP address of their machine *(no need for an ID)*.
- a user must login to their account, and the app is local, which means we don't necessarily **need** to add security features *(such as a password)*.

3 Actors

3.1 Primary actors

- **Admin** = person in charge of the whole deployment of the system on different supports *(Android, Linux, OS X, Windows)*.
- **User A** = user of the chat application, on a machine where the app was previously deployed by the admin.

3.2 Secondary actors

- **User B** = User connected on another session / machine. Will receive messages, files, etc. Just a receiver here, would become *User A* if they made any action.

4 Use Case Diagram

Introduction : We started designing our application by creating a Use Case Diagram, to get and give a global understanding of how our system should work. It mainly focuses on interactions between a main user *User A* and the system, showing all of the different possibilities for them to use the application. For a description of the different actors, refer to previous paragraph.

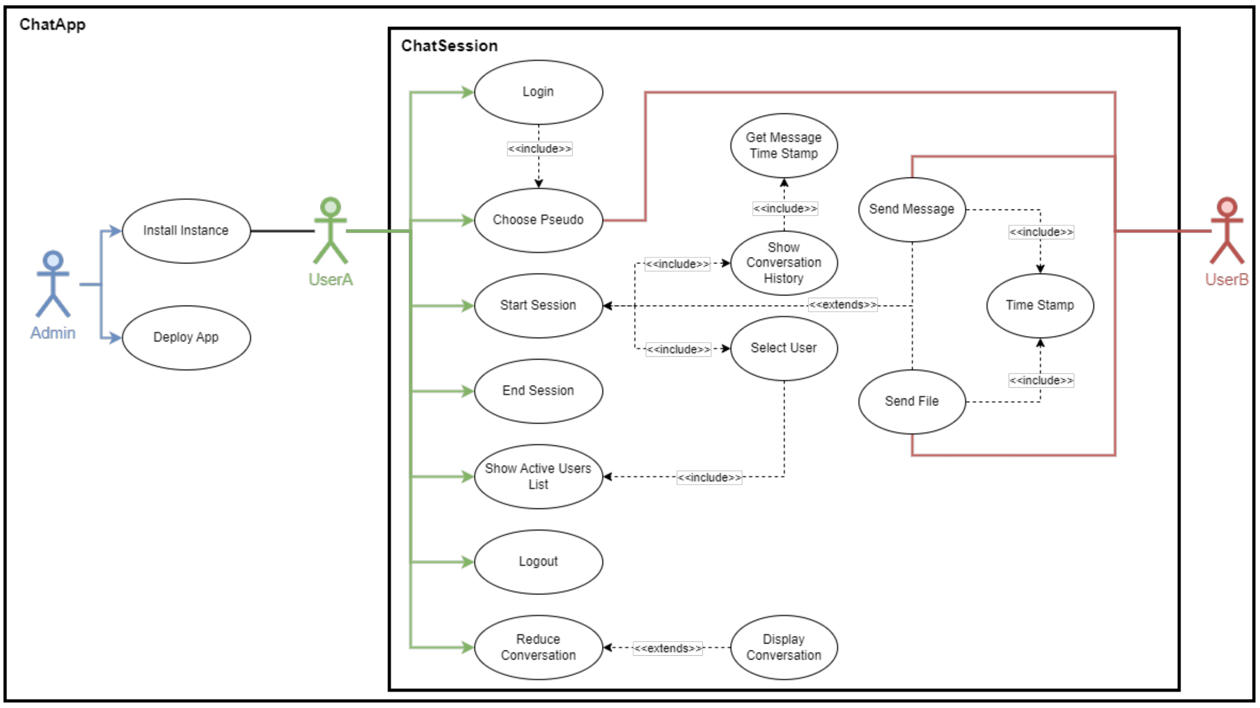
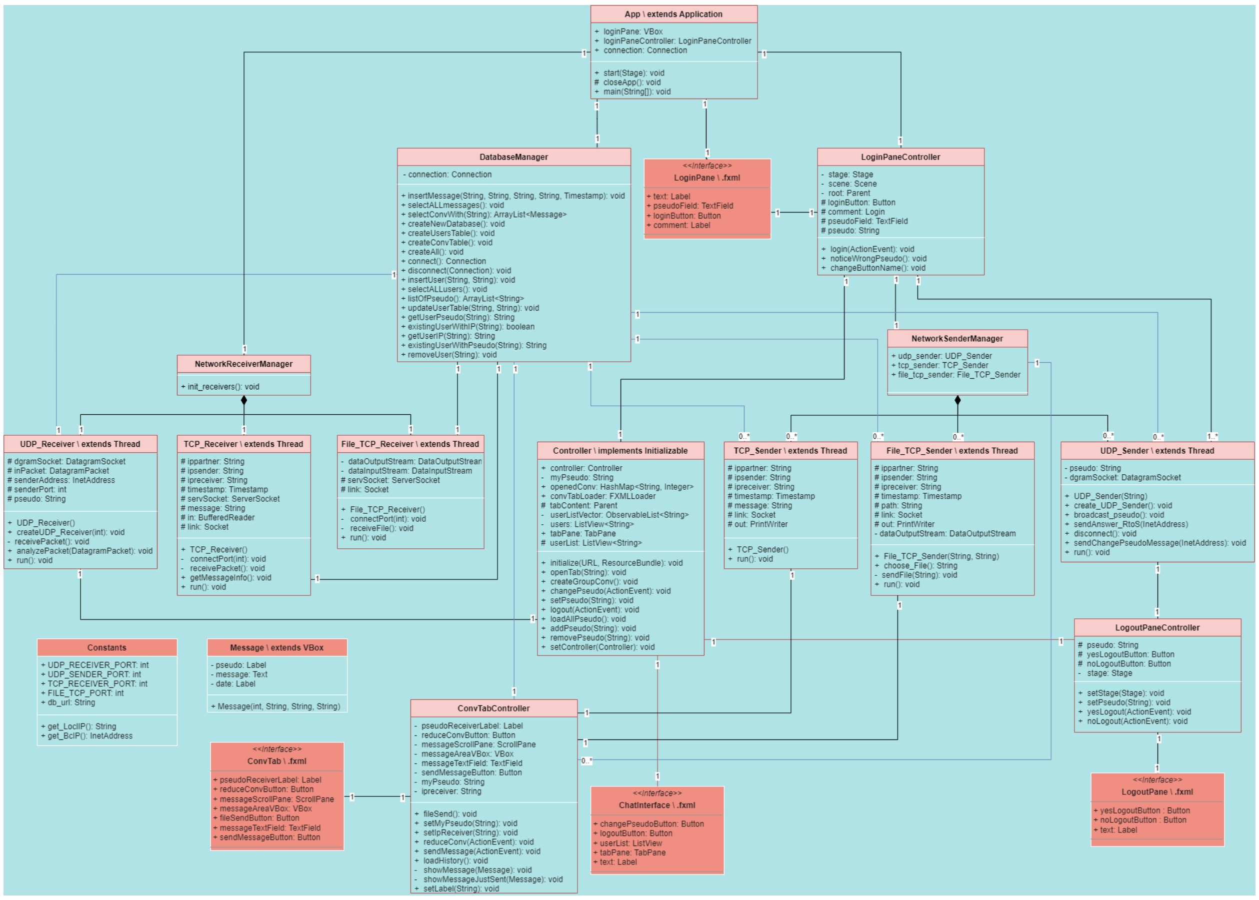


Figure 1: Use Case Diagram

5 Class Diagram

Introduction : The following Figure is the class diagram for our chat project. It allows for a better understanding of all the relations between the different classes, interfaces, etc. To give a brief presentation of how the system works, one must start with the **App**. It is the core of our system. When a user launches the app, it creates | retrieves the database *(via the DatabseManager)*, starts the receiver threads *(via NetworkReceiverManager)* and displays the correct window *(LoginPane via LoginPaneController)*. We will not go too much further in the explanation of the class diagram, the most important part to understand is that every interaction goes through the **DatabaseManager** *(add a user, add a message, a file, etc)*. Also, every sent message | file implies the creation of a temporary *TCP sender*. On the other hand, every reception goes through one of the three *receiver threads*.



App \ extends Application
DatabaseManager
<<Interface>> LoginPane \ .fxml
LoginPaneController
NetworkReceiverManager
NetworkSenderManager
UDP_Receiver \ extends Thread
TCP_Receiver \ extends Thread
File_TCP_Receiver \ extends Thread
Controller \ implements Initializable
TCP_Sender \ extends Thread
File_TCP_Sender \ extends Thread
UDP_Sender \ extends Thread
Constants
Message \ extends VBox
<<Interface>> ConvTab \ .fxml
ConvTabController
<<Interface>> ChatInterface \ .fxml
LogoutPaneController
<<Interface>> LogoutPane \ .fxml

Figure 2: Class Diagram

Indications :

- if a class extends another, it is written next to the name of the said class, *e.g App extends Application*
- light pink boxes *(e.g UDP_Sender)* represent **controllers**
- darker boxes *(e.g Message)* represent **Models**
- non transparent boxes *(e.g ConvTab)* represent **Views**. They are all *fxml* files, each controlled by a specific **controller**.

6 Composite Diagram

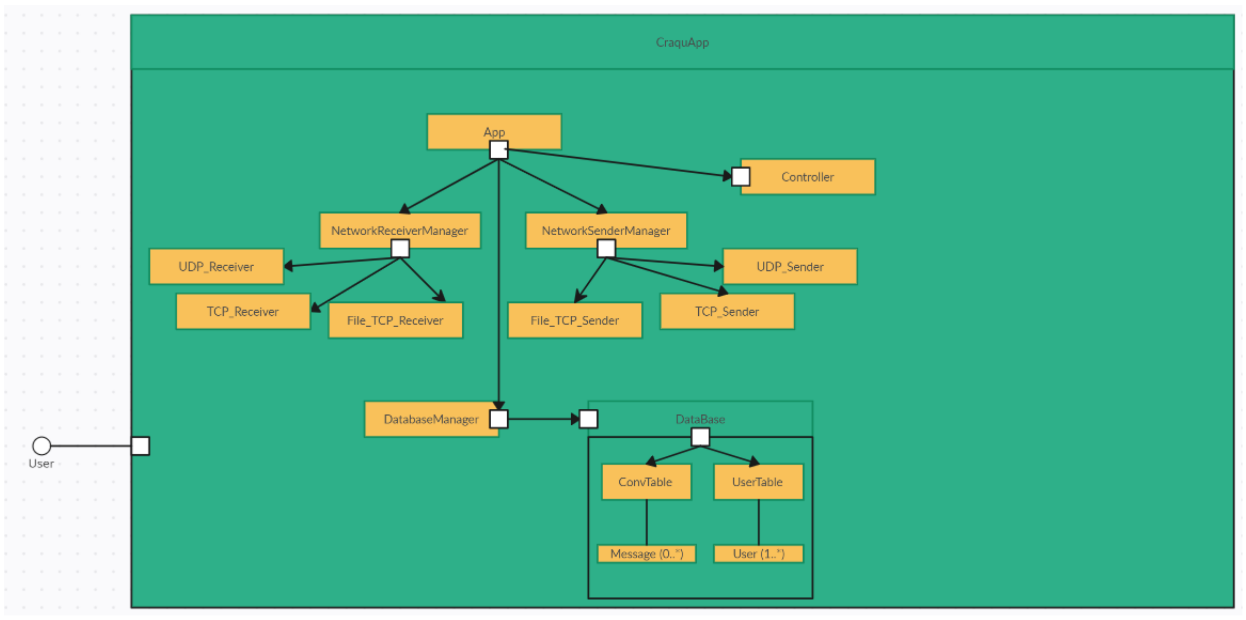


Figure 3: Composite Diagram

7 Sequence Diagrams

7.1 Login

This sequence diagram focuses on the first step of launching the app : logging in. When a user starts the app, a login window is displayed on the screen. It prompts the user to enter their pseudo. Should it be valid *(unique, and not empty)*, the system would add the user to the database's users list using their ip address and the entered pseudo, send *(via a UDP broadcast)* a notification to all connected users with the pseudo, then wait for their answer *(to add to the database the ip address and pseudo of all connected users)* and launch the chat interface. Should it not be valid, the system would refuse the entered pseudo and enquire another, until the user enters a valid one.

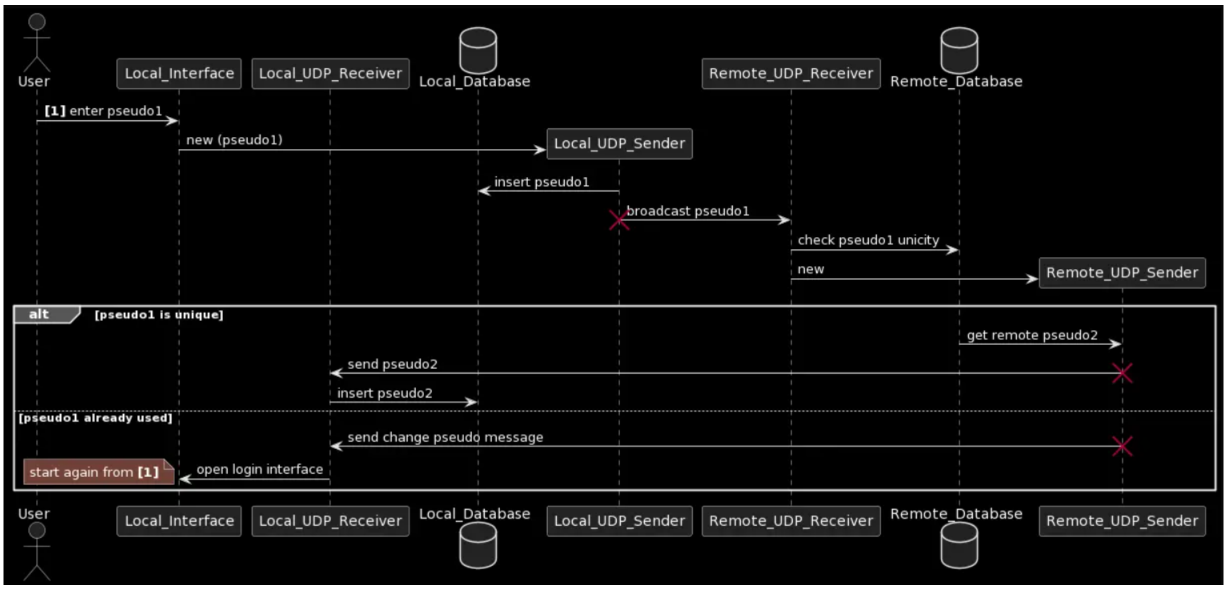


Figure 4: Login *(includes change pseudo)*

7.2 Logout

When a user decides to, they can leave the app using the **"Logout"** button on the right corner of the *chat interface*. They will then be asked whether or not they're sure they want to leave the app by displaying a new *Logout* window instead of the *chat interface* window. Should they click **"Yes"**, the app would broadcast a UDP disconnection message and terminate all running threads & the App. Should they click **"No"** instead, the *chat interface* window would be displayed again, and the user would be able to just continue using the app as before.

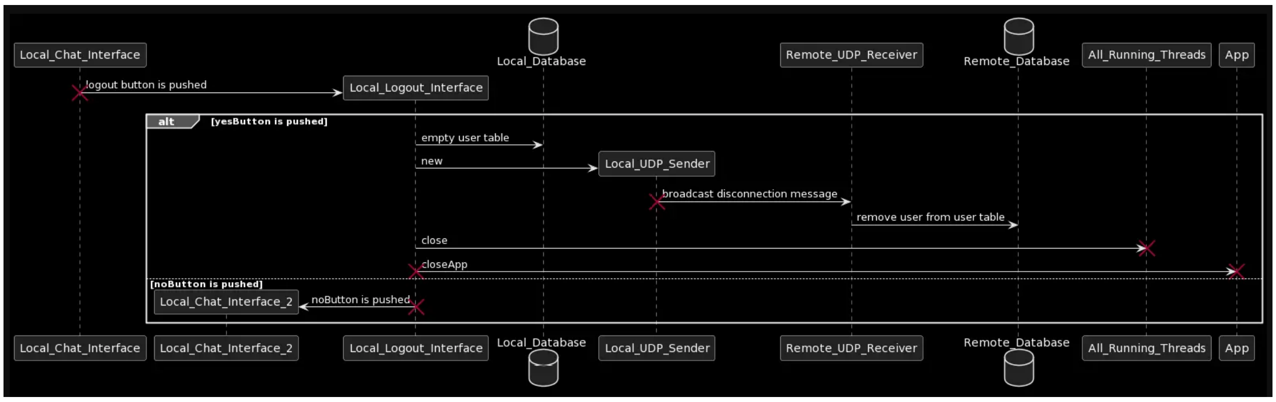


Figure 5: Logout

7.3 Exchange messages

Sender's point of view: Once a user is successfully logged in, and if other users are online, they can start a conversation and exchange messages. When a user types their message in the text field on the *chat interface*, a temporary **TCP_Sender** Thread is created. It inserts the typed message in the *database*, sends it using the recipient's IP address and destroys itself afterwards. Simultaneously, the *interface* empties the text field.

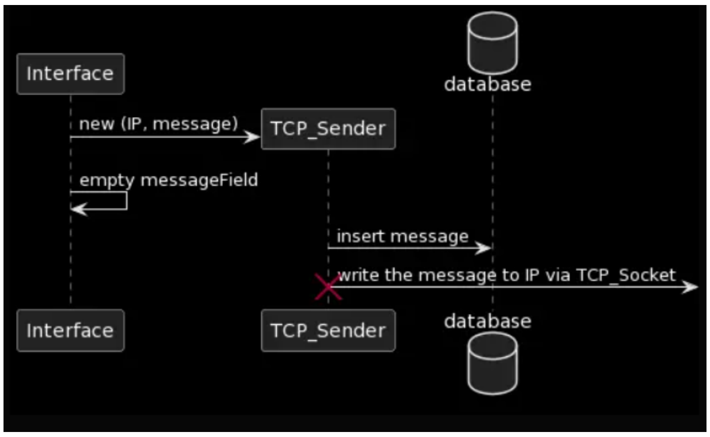


Figure 6: Send a message

Receiver's point of view: Receiving messages is done on a continuously running **TCP_Receiver** Thread. When a new message arrives, it is inserted into the *database*, then displayed on the **ChatInterface**. TCP exchanges occur on port *3000*.

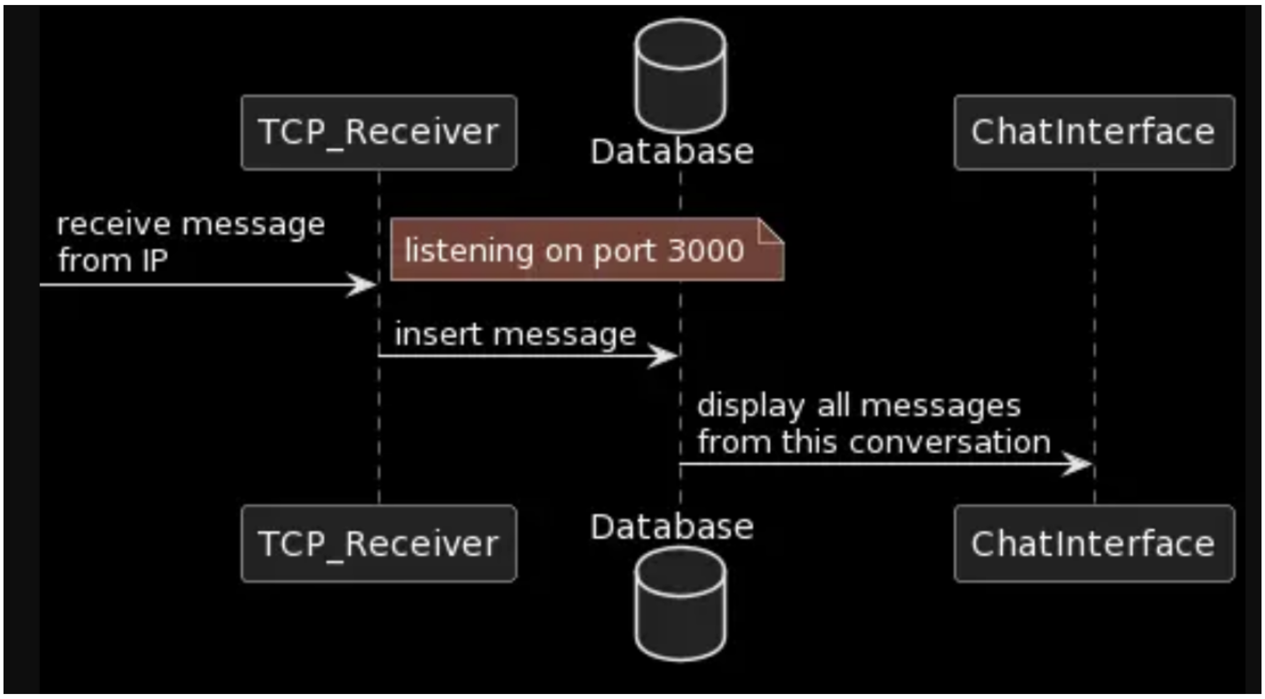
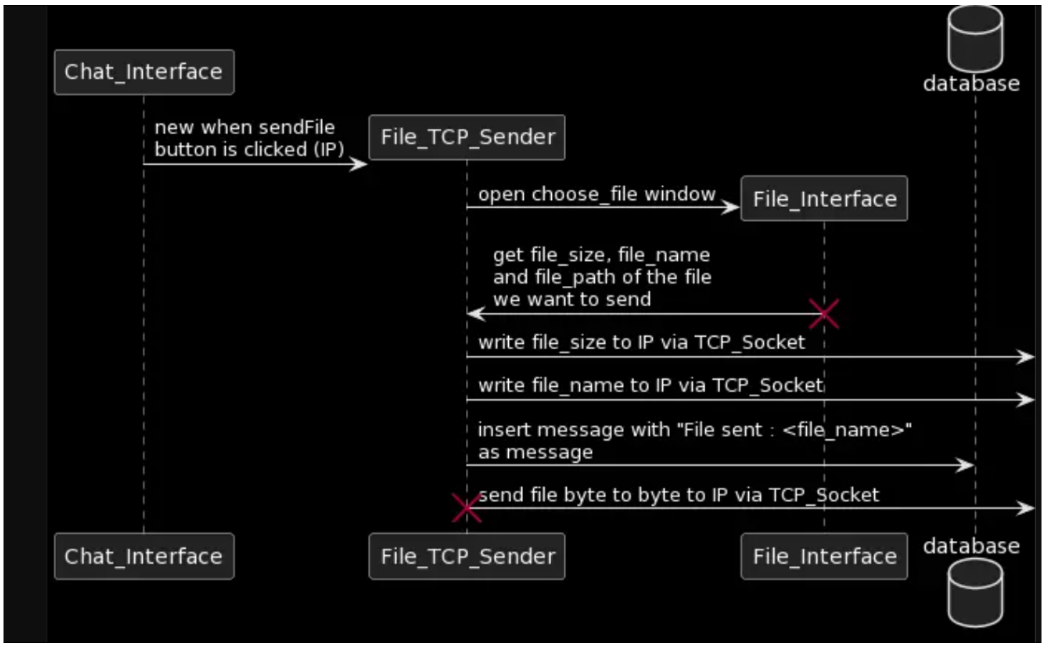


Figure 7: Receive a message

7.4 Exchange files

Sender's point of view: As for sending a message, sending a file creates a temporary running **File_TCP_Sender** Thread *(every file exchange is done on port 3001)*. A user must press the **sendFile** button to create the thread and open a **choose_file** window, then select the desired file and press a *send* button *(the choose_file window will then close itself)*. When this is done, the thread sends the *file_size* and *file_name* via TCP to the recipient, then the entire file byte to byte via TCP again. Only the *file_name* is inserted into the *database*. After all of these steps, the **File_TCP_Sender** is destroyed.



Chat_Interface
database
new when sendFile button is clicked (IP)
File_TCP_Sender
open choose_file window
File_Interface
get file_size, file_name and file_path of the file we want to send
write file_size to IP via TCP_Socket
write file_name to IP via TCP_Socket
insert message with "File sent : <file_name>" as message
send file byte to byte to IP via TCP_Socket
Chat_Interface
File_TCP_Sender
File_Interface
database

Figure 8: Send a file

Receiver's point of view: As for receiving messages, a **File_TCP_Receiver** Thread runs continuously, from the launch of the application to its closure. The steps described from the sender's point of view are the same here. Note that the information *(except the file itself)* are received on port 3000 *TCP_Receiver thread*, and receiving a file name induces the creation of a file on the machine / computer itself. After receiving and copying the entire file on the computer, the *conversation history* is loaded, and all of the threads go on listening mode again.

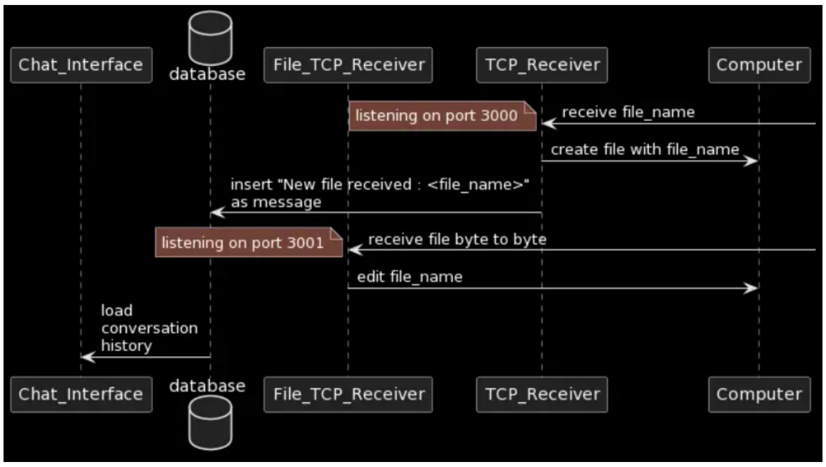


Figure 9: Receive a file

8 Login

At the launch of the application, a user will be prompted to enter their pseudo in a *Login window*. As you can see on *Figure 10*, the window contains a text field where a user can type the pseudo they wish to use. Then, they can either press *Enter* or click the **"Login"** Button to get into the application *(cf. Figure 11)*.

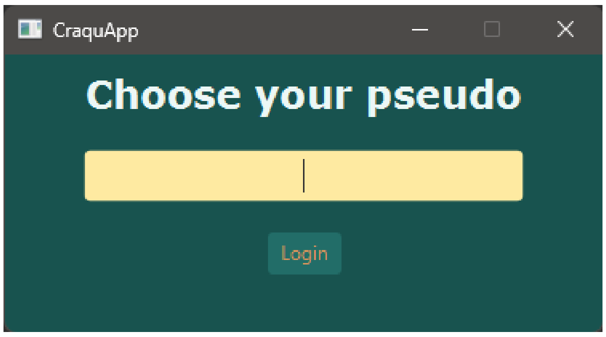


Figure 10: Login Window

9 Open conversations

When logged in, a user will be shown a window similar to the one on the figure below. On the left, we can see a list of *"Online Users" (here, "Moi" and "Toi")*. To open a conversation window and start chatting, a user must click on the desired pseudo *(see here, highlighted in blue, the opening of a conversation tab (on the right of the window) with "Moi"*.

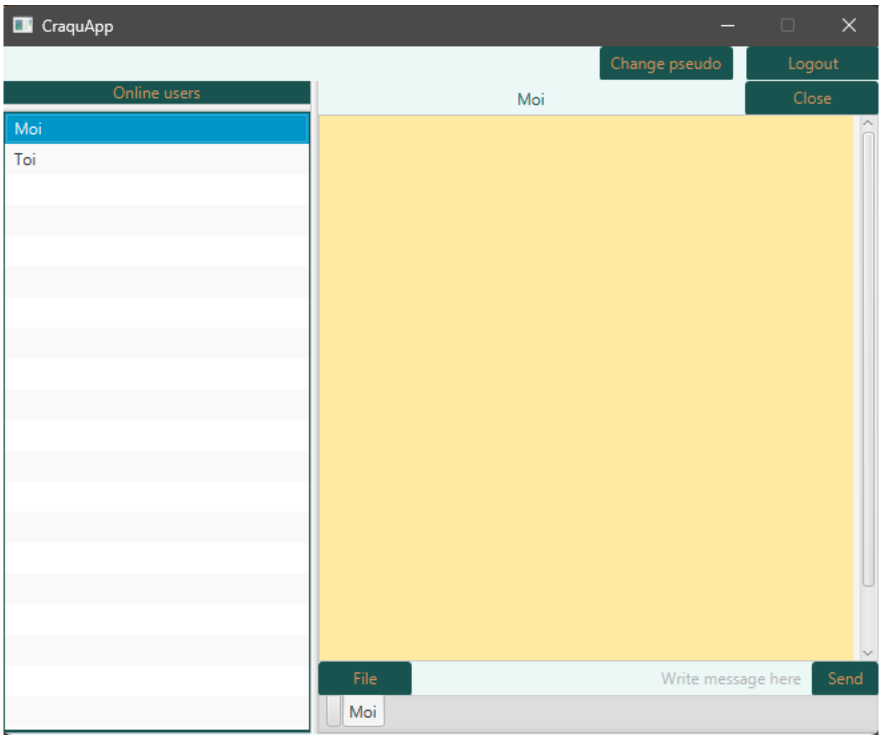


Figure 11: Chat Interface, focus on a user

10 Send a message

Once a conversation tab is opened, a user can send messages by typing the desired message in the text field at the bottom of the c*chat window (here, a user has typed "A message I want to send").* To send the message, one can either press *Enter* or push the **"Send"** button located at the bottom right corner of the window. Sent messages will appear in the *conversation tab*, right_aligned for messages the user sent *(here, "A message I already sent"*, left_aligned for messages the user received.

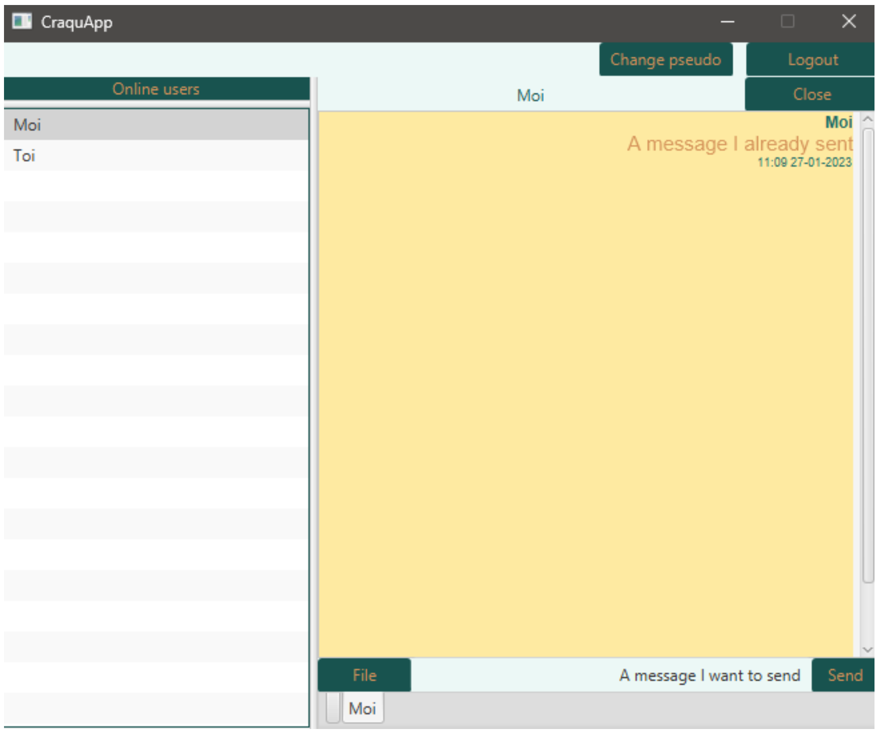


Figure 12: Conversation window with *Moi*

11 Send a file

To send a file, a *conversation tab* must first be opened. Once it is done, a **"File"** button will become clickable at the bottom center of the window. A user might click it, which will open a **Choose_file** window *(as the one shown on the figure below)*. Here, the user can navigate through their repositories to select the desired file. Note that a **"Files of Type"** list has been created, for the users to sort the files depending on what they want to send *(Images = .jpg, .png, .jpeg, .gif; PDF files = .pdf; Text files = .txt; or All files = .*.*

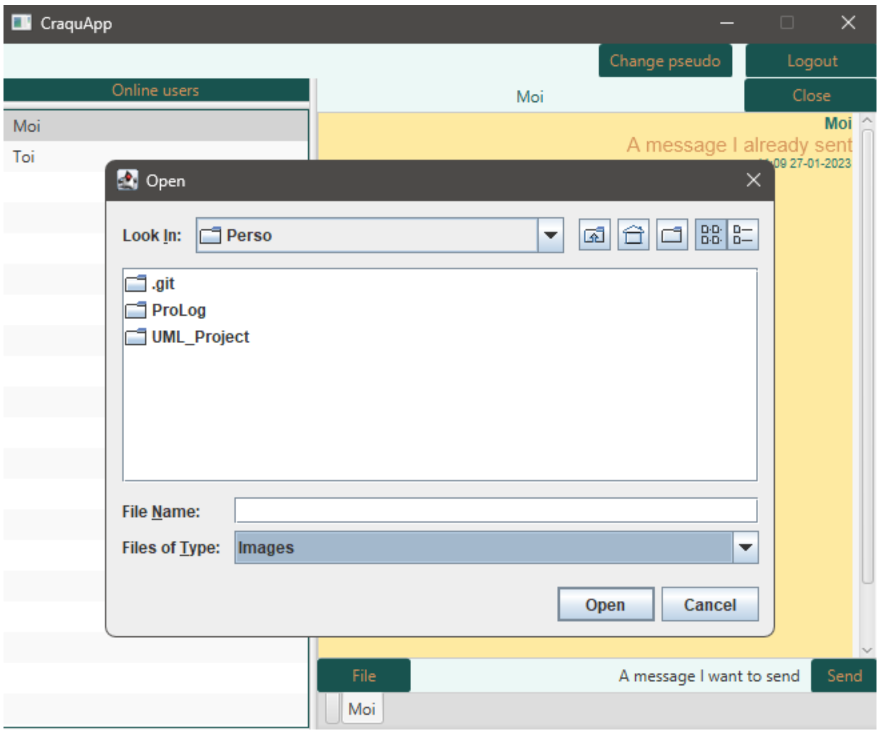


Figure 13: Choose_file window

12 Change pseudo

In order to change their pseudo, a user must press the **"Change pseudo"** button on the top right of the *chat interface (cf. Figure 11)*. They will then be prompted to enter a new pseudo on the same window as *Figure 10*.

13 Logout

A user can decide to logout at any given moment, by pressing the **"Logout"** button on the top right corner of the *chat interface (cf. Figure 11)*. A logout window will then open, as shown on the figure just below, for the user to confirm or not their will to leave the app *(**"Yes"** to leave, **"No"** to stay...)*.

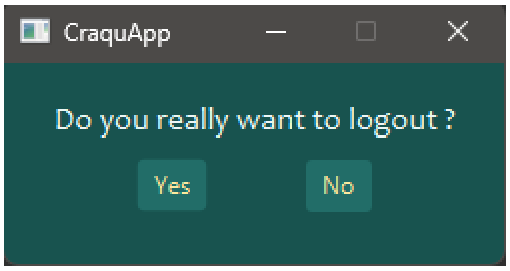


Figure 14: Use Case Diagram

14 n.b

We should have implemented a **close** functionality, allowing a user to close an opened conversation, but we couldn't implement it. Therefore, the **"close"** button is of no use.

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