INSA TOULOUSE

Application User Manual

Clavardeur31

Table of Contents

Clavarder31 User Manual

Application description

Description of the services provided by the application and its limitations

Clavager31 is an instantaneous discussion system between people whose machines belong to the same network. The application allows to directly exchange text messages with other users without the need for a 3rd party. The users may freely choose a username to connect the application with and identify between each other using these usernames. Users may change their usernames at will provided the username is not already taken up. The users are informed in real-time of who else is currently connected and when a person they are conversing with has closed the application. The conversation history is kept locally and persists closure of the application. The conversation history may be manually erased by the user.

The network environment is assumed to be safe and secure. The application does not implement security services. The content of the messages and IP addresses of users are not encrypted in anyway. The application only permits the emission of text messages. It is assumed that the IP addresses of the users remain unique and static in the network.

Installation

The application is released on our GitHub repository under the form of a fat-jar including all necessary dependencies for the execution as is. The installation does not need any preliminary installation and can be directly executed.

```
java -jar Clavardeur31-1.1.0-SNAPSHOT-shaded
```

```
where “Clavardeur31-1.1.0-SNAPSHOT-shaded” corresponds to the latest
release.
```

At the first start, the application will create a `conversation.db` in the parent directory. The file should be kept in the same directory as the application for the message history to be kept between sessions of the application.

Use manual

After successfully installing the application the user will be able to start it by double-clicking on the Clavager31.jar file.

Once the program is started a login interface will be displayed. The user will then have to write a nickname in the text field shown in order to log into the application. Remember that this nickname is temporary and that 2 people can't have the same one. If the one you wrote is already taken an alert will be shown.

Once you are logged a new window will appear. In the left part of the window will be displayed the list of connected users to whom you can talk. A red notification indicator will be displayed next to the corresponding user if he sends a message to you.
To start a conversation, you simply have to click on a user nickname and your history with him will be displayed in the middle of the screen. You can type a message in the text field and press ENTER to send it to the current user you're talking to.

You can erase the history with someone by clicking "Erase History" in the top right part of the windows after having selected the correct user.

You can change your nickname in the top left of the window and will be redirected to the login screen where you will have to type your new nickname.

To close the application just click on the red cross in the top right part of the window like with usual programs.

You can also minimize it to perform other tasks.

Conception choices

Use case diagrams

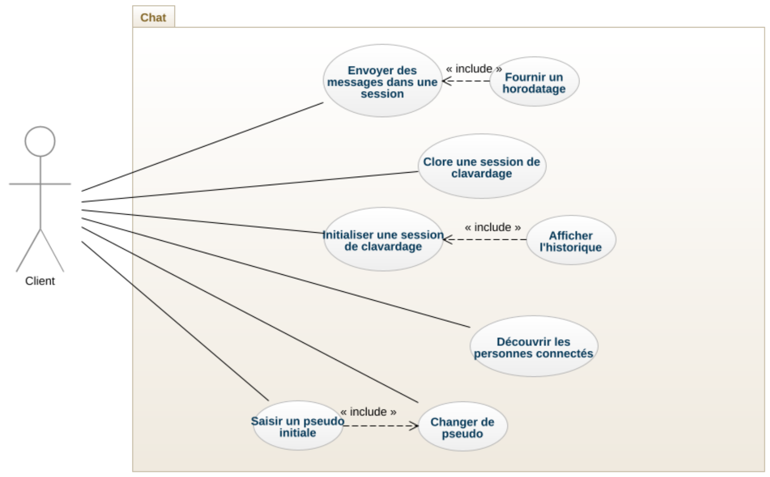
Chat
Client
Envoyer des messages dans une session
« include »
Fournir un horodatage
Clore une session de clavardage
Initialiser une session de clavardage
« include »
Afficher l'historique
Découvrir les personnes connectés
Saisir un pseudo initiale
« include »
Changer de pseudo

Figure 1: Diagram of case "chat"

Class diagram

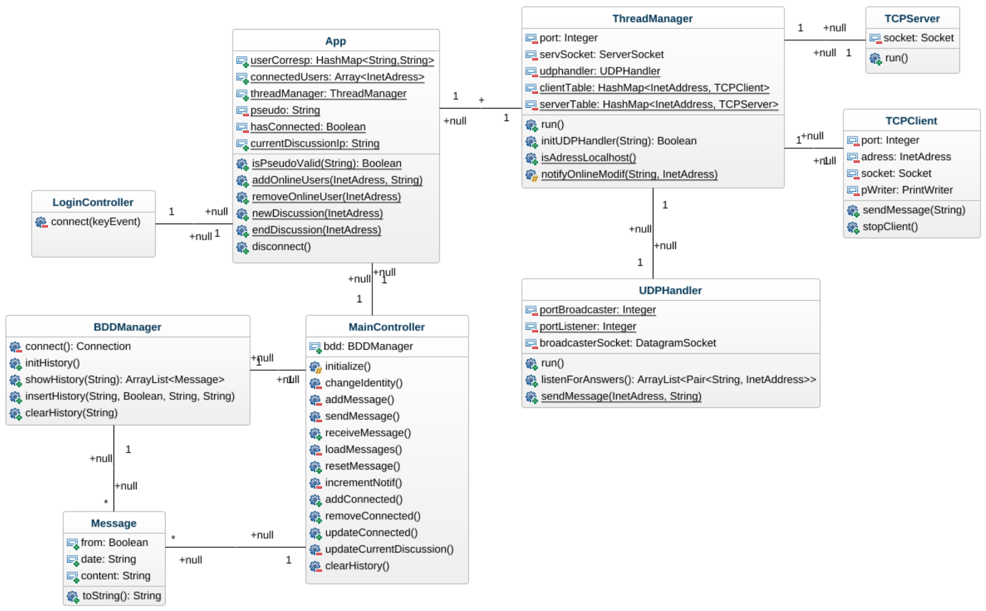

Figure 2: Class diagram of the application

Sequence diagram: General case

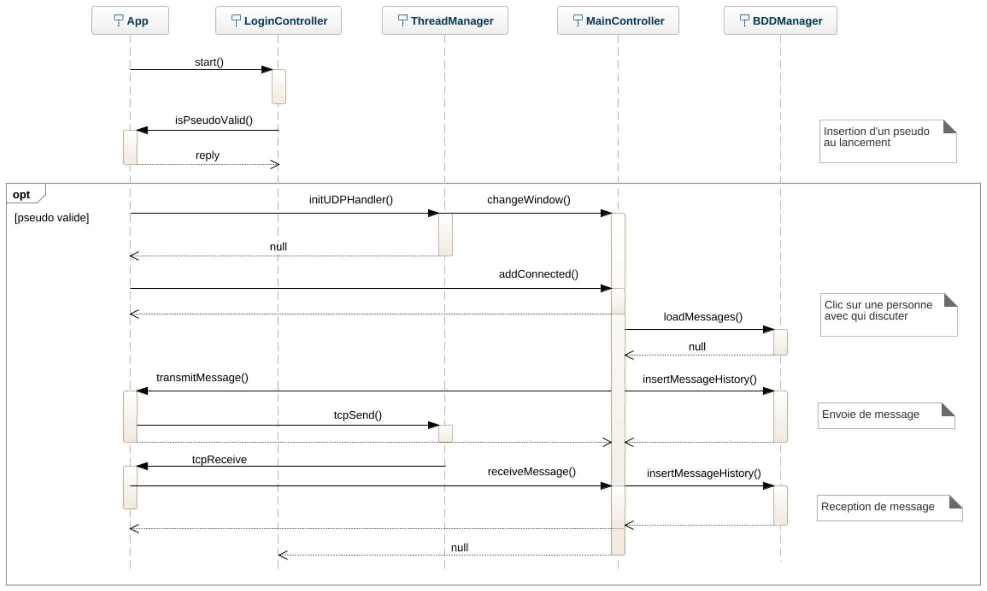

Figure 3: General Sequence Diagram

Sequence diagram: Initial connexion

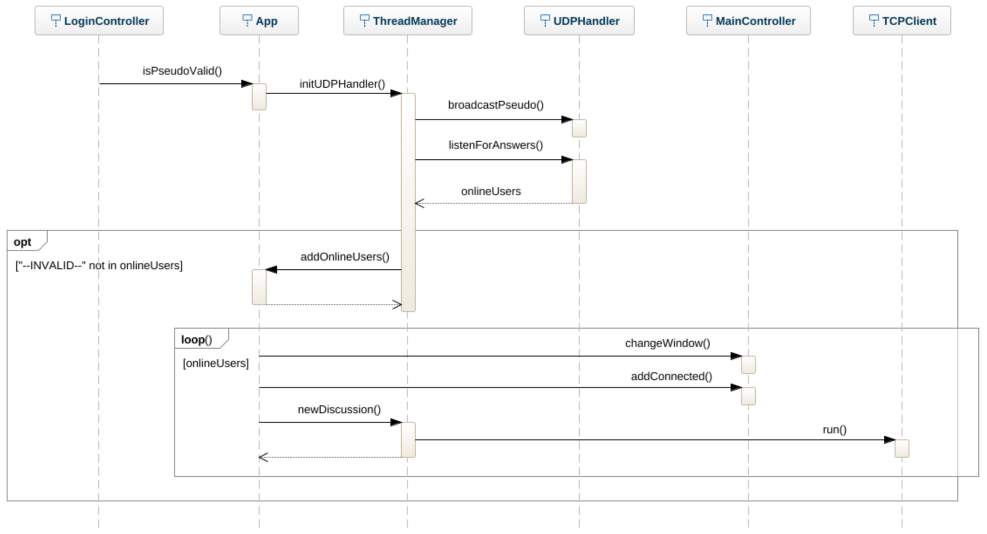

Figure 4: Initial connexion Sequence Diagram

Sequence diagram: Disconnexion

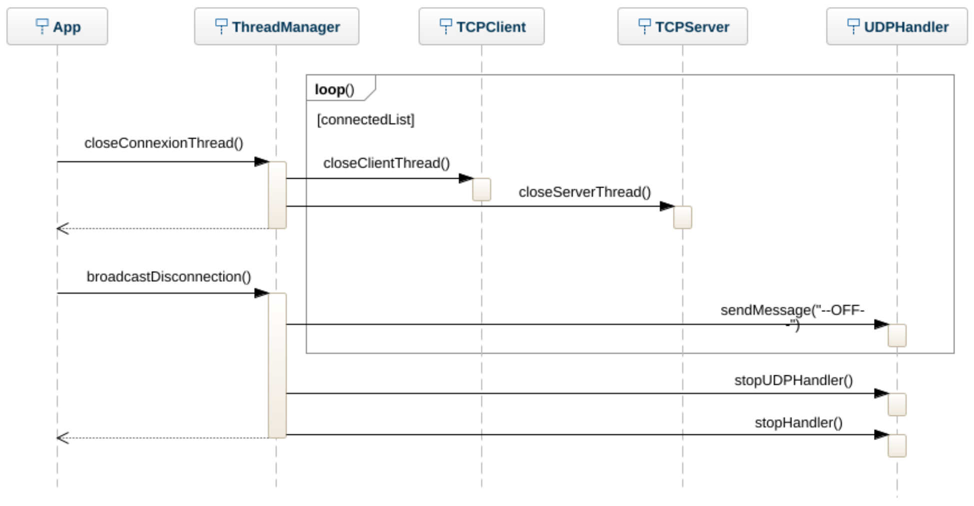

Figure 5: Disconnexion Sequence Diagram

Architectural choices

Message history storing

All messages are locally stored in an **SQLite database.** We chose SQLite because it provides local databases with relative lightweight. It also really is easy to use and there isn't anything to set up at all.

GUI

To avoid dealing with Swing while making the GUI while having a decent looking interface we chose **JavaFX**. It allows us to make creating objects way easier and can be linked with a stylesheet in **CSS** for further customization. To ease the creation of the **JavaFX** files we also used a software called **SceneBuilder** that allows to drag and drop elements in a simulated GUI. With it, we've been able to create a nice-looking GUI in a relatively short time.

Testing procedures

JUnit Tests

Some JUnit tests are available in the project to make sure everything is working properly. Tests are performed automatically with Jenkins once something is pushed through GitHub. The SQLite database, the ThreadManager and the user correspondence are tested this way. But a vast majority of our code needs more than a computer to be tested thus JUnit tests aren't enough.

Manual tests to perform

To tests the application few scenarios could be considered with 2 users:

- Starting the application and try taking the same name. An error message should appear for the last one connecting.

- Logging with a user and trying to change its nickname. Then checking if the name change has been registered by the other user.

- Trying to send a message to a connected user and checking if it is well received on the other end.

- Send a message to the other user, change nickname or disconnect and reconnect. See if the history shows when chatting with the same user.

- Press the "Clear History" button and check if all the messages are still erased after disconnecting and reconnecting.

- Closing the application and seeing if the other user registered the disconnection. You should be redirected to the default menu.

Tests to perform with 3 or more connected users:

- Chat with a user and make this user disconnect. The conversation focus should go to another user in the list.

- Chat between two users. Have one of the two participants disconnect and another one connect using the disconnected username. Check that the newly arriving user doesn't have access to the conversation history between the other two.

Git Repository

The code complete code for the project is available on our GitHub repository: **https://github.com/hmathieu-insat/Clavardeur31**

/Ressources/ Is a folder containing screenshots and diagrams.

/src/main/java/com/insa/projet4a/ Contains the main java files used for the project.

/src/main/resources/com/insa/projet4a/ Is for the GUI part. It includes fxml files used as templates to make the visual render aswell as the images and css files.

/src/test/java/com/insa/projet4a/ Records the JUnit tests we made.

For each task we opened a new temporary branch and merged it with main when the version was stable. We also used GitHub issue system to monitor our progress.

Project schedule

We used Jira to plan out our progression and set up goals for each sprint. We first listed every user story and task we could think of. Then we chose to begin with the biggest part and divide the work in 2 for each other to work on a different part and thus being faster. One of us began to work on the GUI part with JavaFX while the other handled the communication part with threads and sockets.

Because the GUI part was much lighter than the communication part, the same person continued with smaller tasks such as the database and the timestamping.

We also discussed during every work session to explain to each other what we did, what we were currently working on and what we expect to do next. Because of this, we were able to give an exterior and fresh point of view to the other concerning his part.

The details about our work process as well as sprints and user stories we've made can be found on our Jira project page.

https://insa-test-project.atlassian.net/jira/software/projects/CLAV/boards/2/reports/burnup

Automatizaton

For the automatization we used Jenkins's job system. Because our application is relatively lightweight, we didn't need many jobs.

We needed to compile the project regularly into a .jar. Maven being convenient for this task, we only needed to execute "mvn clean" and "mvn package" commands. We also wanted to carry out JUnit tests regularly but this was already included with the "mvn package" command so we didn't need to add anything else.

We chose to link our jobs to our GitHub page so that every time something was committed, test and compilation were automatically run. Furthermore, this job could also be triggered manually.

The .war containing the whole Jenkins configuration is available on our GitHub home directory.