Lab 2 -Chat Application

Distributed System

18th of February 2024

—

Submitted by Groupe K

PHAN Manh Tung

ESCAT Gregory

MOSIG M1

INTRODUCTION

The chat application is a simple application allowing different users to chat in real time. This application was made in the context of a lab in the *Introduction to Distributed Systems* course of M1 MoSIG at UGA. The goal was to familiarise ourselves with using RMI objects with a server proposing services used on the client side. The project can be found in the following github page: https://github.com/Graigauri/DS_Chat_Application.

INSTALLATION AND SET UP

How to compile

Before compiling this project, make sure you have installed the java JDK. To compile the project, open a terminal and place yourself in the main project file. Then simply type the following command in the terminal:

```
javac *.java
```

How to run the program

To use our chat application, you will need two terminals, one running rmiregistry and the other running the chat server. To run rmiregistry run the following command in the first terminal:

```
rmiregistry
```

And in the second terminal run this command to start the chat server:

```
java ChatServer
```

Now that these two programs are running, users can now connect to the server to chat. Each user will need its own terminal and will have to run this command:

```
java ChatClient
```

DIAGRAM

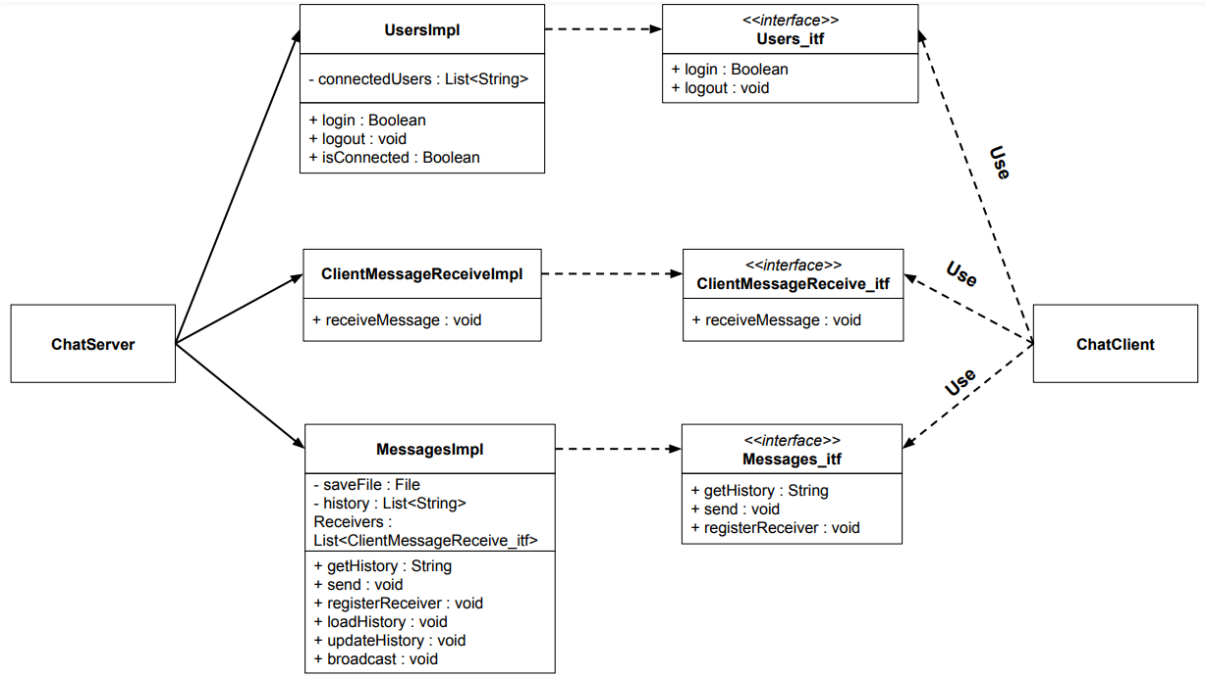

Figure 1: Class Diagram

FUNCTIONALITY

The *ChatServer.java* file contains the code for the server. At launch the server will set up services that are used remotely. These services are either related to the messaging system (see *MessagesImpl.java* file explanation) or the user system (see *UsersImpl.java* file explanation). It will then load the chat history from *chat_history.txt* if the file exists or create a new file.

The *ChatClient.java* file contains the program used by clients/users. Before any interaction with the user, this program will connect to the server and get access to the services offered by it. The program will then ask the user to choose a username to join the chat, while checking if the username is available. Once the user joins the chat, he/she will be shown the chat history and will be able to send/receive messages. Two commands are available to the user; *-quit* to quit the chat and *-hist* to print the history of the chat.

The *MessagesImpl.java* file implements the message system. It loads and updates the chat history in real time, saving the date and time of each message.

The *MessagesImpl.java* file encapsulates the functionality related to the messaging system in the chat application. When the server is initiated, this class is responsible for setting up services that handle message processing. *MessagesImpl* manages the chat history, updating it in real-time as messages are sent and received. The registerReceiver registers clients as receivers, ensuring that they receive messages sent by other users. And of course, the class provides the methods for sending messages, broadcasting them to all connected users.

The *UsersImpl.java* file maintains a list of connected users. The login method checks if the username is already taken before allowing a user to login. If the login is successful, the user is added to the list. As the opposite, the logout method removes a user from the list.

PROBLEM

One problem with our chat application is that a user will receive the message he sent, not only messages from other users. It could be fixed by implementing a check system in the *broadcast* method of *MessagesImpl* to check if the message is from the user or another client.