EDA095 - Network programming

ShitChat

Hanna Björgvinsdóttir
Robin Seibold
Meris Bahtijaragic
Bengt Ericsson

dat11hbj, dat11rse, dat11mba, ada08ber@student.lu.se

2015-05-17

Contents

1 Introduction

The ShitChat application was made in conjunction with the project part of the course Network Programming at LTH. The assignment was to make an application related to the course, and more specifically where the application functionality comprises of network communication.

The application itself is a simple, easy to use chat application, similar to the classic application IRC.

2 Requirements Review

The requirements for the system are divided into two subcategories: mandatory and optional. Naturally the mandatory requirements are the ones with the most functionality, and they state that the user should be able to

- send and receive public messages to and from all users on the server
- see a list of all users currently in the chat
- send private messages, which are only seen by the sender and receiver
- create, join and leave channels
- send and receive messages to and from users in a specific channel

The alternative requirements were to be implemented after the mandatory ones, given that time allowed. The alternative requirements state that the user should be able to

- log in
- create a personal profile, and have the possibility to view others profiles
- see which users are typing
- use emotes and emoticons
- write and see mentions
- obtain achievements
- get feedback from a karmabot

3 System Outline and Implementation

The two major building blocks of ShitChat are the client and the server, which communicate over TCP, using a text-based protocol. Both server and client use separate threads to handle input and output.

3.1 Client

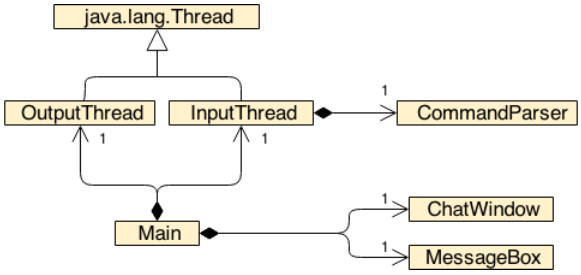

Figure 1: A simplified UML class diagram for the client.

The most important classes of the client are:

`ChatWindow` - Is the application GUI and contains an input field, textarea, and a list of online users.
`InputThread` - receives input from server, and sends it to `CommandParser` for parsing.
`OutputThread` - sends user input obtained from the synchronized `MessageBox` to the server.
`CommandParser` - parses commands received from the server by `InputThread`, and acts according to the command.

3.2 Server

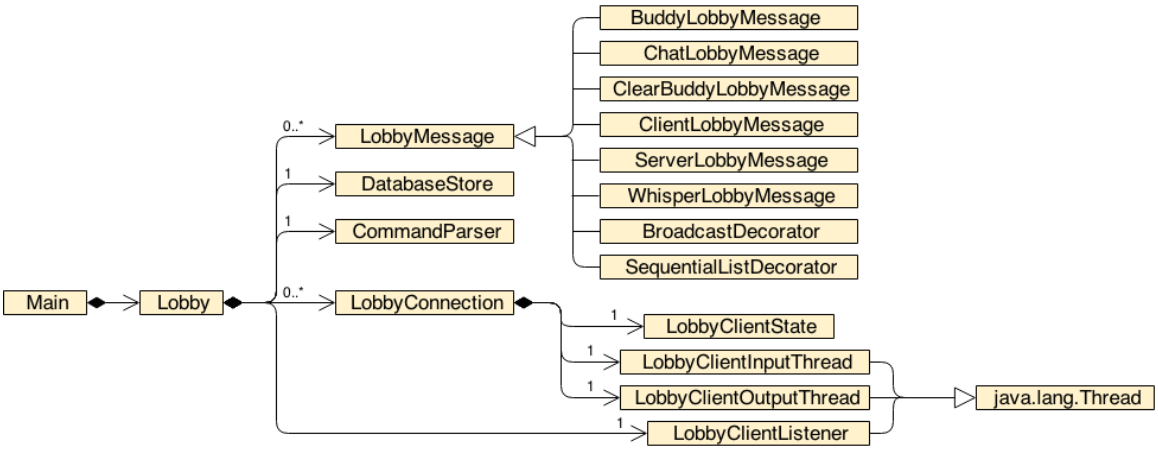

Figure 2: A simplified UML class diagram for the server.

The server is slightly more complex than the client, and its most important classes are described below.

`Lobby`

The lobby is in charge of client communication. It parses messages with `CommandParser`, and acts according to the commands received. It starts one `LobbyConnection` for each client, and handles user information with `DatabaseStore` - an SqLite database. `Lobby` also starts a `LobbyClientListener`, which is a thread that waits for and handles client connections.

`LobbyConnection`

`LobbyConnection` represents a client connection, holding a `LobbyClientInputThread`, a `LobbyClientOutputThread`, and a `LobbyClientState`. Messages are sent by `LobbyClientInputThread`, and received by `LobbyClientOutputThread`. The client's state is stored in `LobbyClientState`.

`*Decorator`

The `BroadcastDecorator` is used for broadcasting a message to all users, and the `SequentialListDecorator` is used for sending multiple sequential messages. All messages sent are of types extending `LobbyMessage`.

4 User Manual

In order to run ShitChat, the user must have Git, Gradle, and Java 8 installed. A presentation of the program can, along with the source code, be found at http://blog.zolomon.com/ShitChat/.

4.1 Install and Run Instructions

The server and client are downloaded, built, and run by executing the following commands in the console:

```
git clone https://github.com/Zolomon/ShitChat.git
gradle uberjar_server
gradle uberjar_client
java -jar build/lib/server.jar port
java -jar build/lib/client.jar hostname port
```

If no port is specified, the default port used is 8888. The default hostname is localhost.

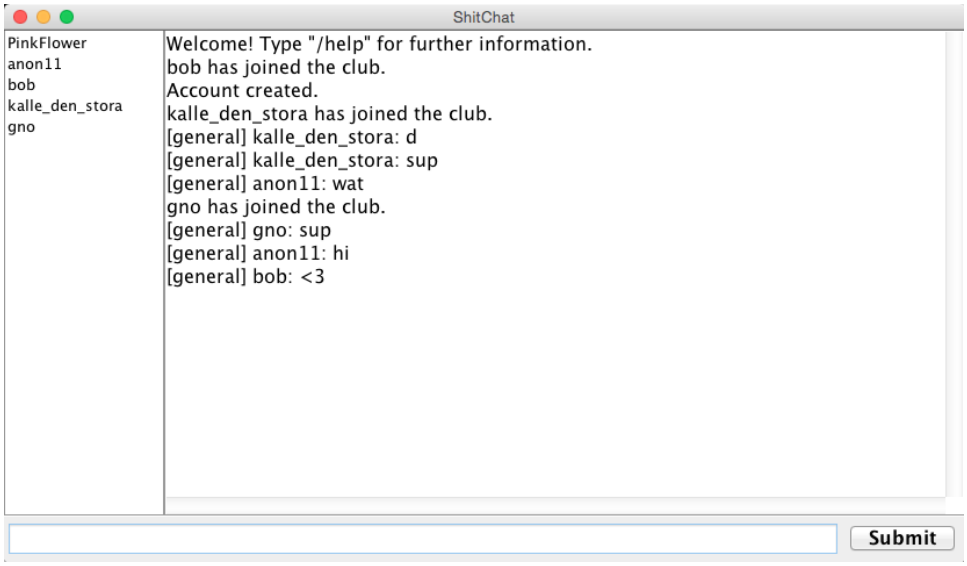

Figure 3: An image depicting the client user interface of ShitChat

4.2 Usage

The client itself is easy to use. As can be seen in figure 3 the list to the left displays all the current users, and the chat is shown in the middle. The user input is written in the bottom most field. Plain text is sent as a public message in the general chat, the more advanced features in ShitChat are accessed with the following commands

help - list all available commands.

quit - quit the chat client.

login <**username**> - login with the username provided.

whisper <**recipient**> <**message**> - send a private message to recipient.

join <**channel**> - join a channel.

leave <**channel**> - leave the channel.

msg <**channel**> <**message**> - send message to all users in channel.

finger <**username**> - view the profile of a user.

editprofile <**name**> <**title**> <**location**> <**avatar URI**> - create/edit profile.

5 Conclusion

All mandatory requirements listed in the requirements review are fulfilled. In addition, the user can log in, create a profile containing name, title, location, and avatar URI, and view other people's profiles. The chat client contains a list of all users connected to the server, and when selecting a username in the list, the input field is filled with the commands needed to send a private message to the selected user. The user can also get a list of all available commands, by typing `\help`.

These additional features were judged as the most important requirements of the alternative ones, and were therefore implemented right after the mandatory ones. Implementing all alternative requirements would not require any restructuring of the program, only addition of code.

Given more time, the next feature to implement would be a list of all active channels, possibly by having one chat tab per channel in the upper regions of the client GUI.