Computer Networks Assignment 1

Francesco Mazzini, 19080; Alice Frezza, 19229; Michael Von Troyer, 19625;
fmazzini@unibz.it, alfrezza@unibz.it, mvontroyer@unibz.it

I. Introduction

Assignment 1

1.1 Purpose

Lo scopo è di sviluppare un Online Chat, in cui multipli Clients possono comunicare tra di loro. Devono poter essere in grado di connettersi, disconnettersi dalla chat, di poter inviare e ricevere messaggi agli/dagli altri client che sono connessi nella Chat Room.

1.2 User needs and Dependencies

Chiunque può essere un utente e non ci sono distinzioni di privilegi. L'applicazione deve essere eseguibile su Windows e MacOS, così come qualsiasi distribuzione di Linux. Ciò sarà possibile data la versatilità del linguaggio in cui sarà scritto: Java. Il progetto non è soggetto ad alcun fattore esterno se non dalle tecnologie di cui sarà composto: Maven (gestore di dipendenze), Git, librerie importate per Maven (JavaFX, altri plugin di Maven riguardanti il suo lifecycle).

1.3 Functional Requirements

L'utente deve poter connettersi alla chat digitando l'indirizzo IP, porta del server e nickname con la quale vuole esser riconosciuto nella conversazione. L'utente deve poter inviare messaggi e poter leggerli. L'utente deve poter essere in grado di disconnettersi dalla conversazione in qualsiasi momento chiudendo la finestra di dialogo o digitando '/quit'.

1.4 Non-Functional Requirements

L'applicazione deve garantire il funzionamento delle suddette funzioni per ogni singolo utente tenendo conto che qualsiasi possibile problema non deve interferire con altre connessioni con altri utenti. Per motivi di sicurezza il server deve mantenere un file di log in cui viene scritto tutto ciò che accade nella chat room compreso l'orario e ogni possibile informazione necessaria.
I messaggi inviati devono essere recapitati agli altri utenti senza possibilità che essi vengano persi e che vengano recapitati in un ordine diverso da quelli invati. Per tale motivo verrà utilizzato il protocollo TCP.

2. Methodology

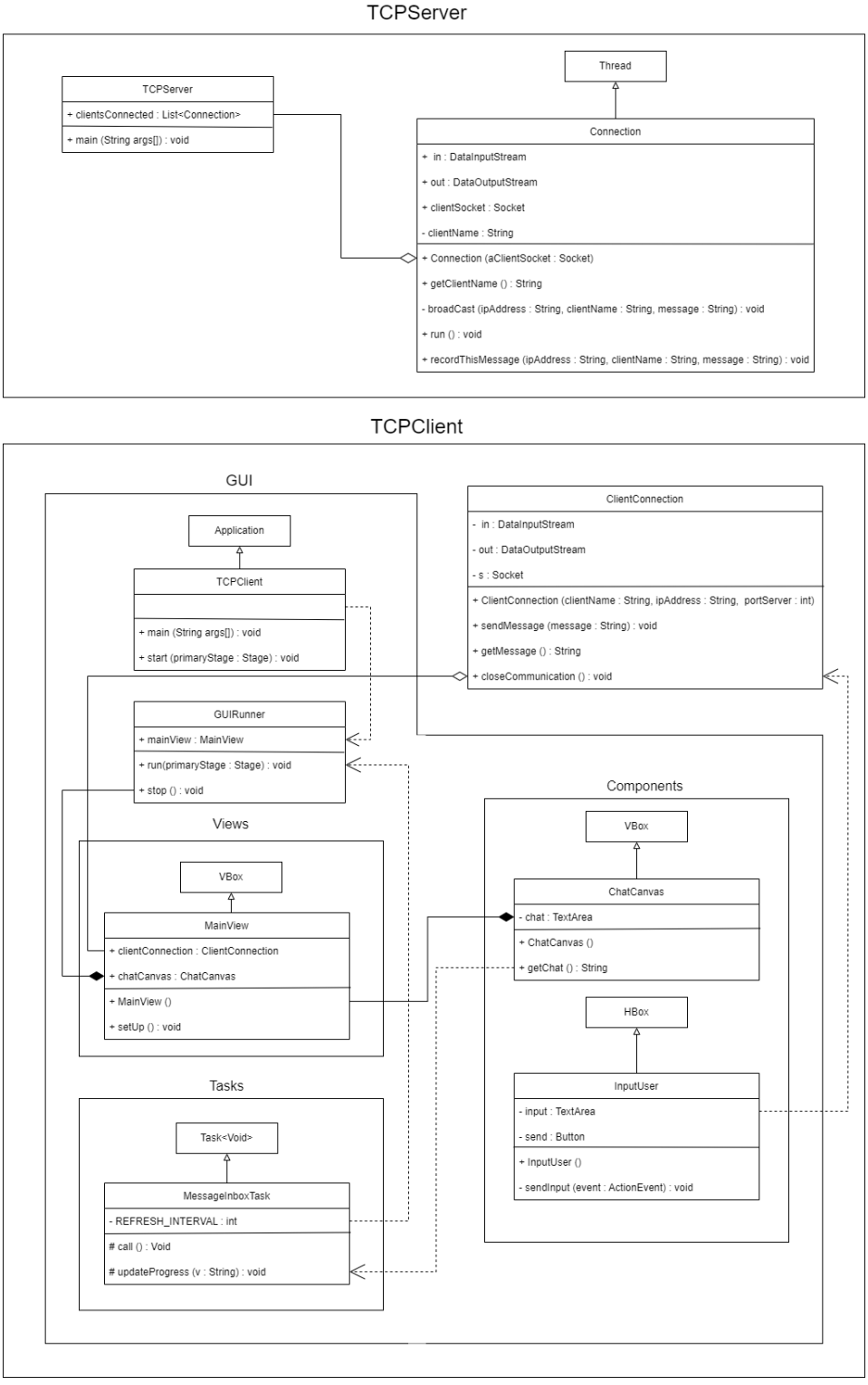
Per soddisfare i requisiti richiesti, si è sviluppata un'applicazione Maven scritta in Java, la quale contiene sia il programma del server che il programma del client, in due folder separati.

Il programma del server si occupa con la classe TCPServer di mettersi in ascolto su una porta arbitraria a scelta, per la connessione di client che vogliono connettersi alla chat. Si occupa poi di fare l'handshake e di aprire un nuovo thread con la classe Connessione in cui viene gestita la connessione del client, in modo tale da liberare il thread principale. Questo nuovo thread si occuperà di inviare i messaggi ricevuti dalla connessione a tutti gli altri utenti connessi e di salvare nel file di log i messaggi inviati da quella connessione.

Il programma del client si occupa con la classe TCPClient di avviare la GUI gestita da GUIRunner. Essa definisce le caratteristiche principali dell'interfaccia utente da un punto di vista funzionale, andando poi ad avviare la MainView che sarà la prima vera finestra che apparirà all'utente. MainView appena prende il controllo dell'esecuzione, chiede all'utente tramite dei Dialogs con quale nome si voglia connettere, a quale IP e a quale porta. Ciò gli permette di istanziare una ClientConnection (class dedicata a mantenere tutte le informazioni della connessione dell'utente tra cui anche il suo socket) per poi riprendere il flow di esecuzione e creare in maniera definita l'interfaccia utente che verrà usata durante la chat. Questo viene fatto istanziando all'interno della finestra la ChatCanvas (che gestisce graficamente l'update dei messaggi in arrivo) e l'InputUser che invece gestisce graficamente e funzionalmente l'invio di messaggi da parte dell'utente al server. Siccome il flow di esecuzione è mantenuto dalla classe InputUser per l'invio di messaggi, ChatCanvas farà riferimento a un thread parallelo di tipo grafico, chiamato MessageInboxTask. Esso assume una struttura leggermente diversa dal solito Thread, perché appunto in grado di

modificare componenti grafiche durante l'esecuzione, cosa non possibile tramite i normali thread. Si metterà quindi in ascolto di messaggi e eseguirà l'update grafico non appena saranno ricevuti.

È possibile visualizzare la struttura del programma nel diagramma UML che segue.



3. Progettazione Algoritmo

Per rendere possibile un'interazione tra server e chat client, l'applicazione server deve essere avviato prima che il client si connetti. Il port sul quale il server ascolta è impostato in modo statico col valore 7896. Quando un utente lancia l'applicazione client, viene chiesto sia l'IP che la porta del server (anche se la porta non cambierà ameno che il codice non venga ritoccato). Solo adesso l'applicazione client cerca di creare il socket per collegarsi al server. Se fallisce chiede di nuovo IP e porta.

Come descritto sopra il server ascolta in continuazione sulla porta 7896 e quando un client si connette, trasferisce la gestione della nuova connessione ad un thread separato. Per tenere traccia delle connessioni il server salva una referenza al thread nella lista clientsConnected. La prima azione di un thread Connection è di salvare il nickname che viene automaticamente inviato dal client come primo messaggio. Il nickname è poi accessibile tramite il metodo 'getClientName' del thread Connection. In seguito, il server manda una notifica a tutti i clienti per avvertirli che un nuovo cliente si è connesso alla chat. Una limitazione dell'applicazione è che la scelta di nickname è completamente libera. Perciò non è garantito che i clienti siano identificabili dal loro nickname. Ciò nonostante, il server riesce a distinguere clienti in base al loro indirizzo IP. I messaggi provenienti dai clienti vengono trasmessi a tutti i clienti dal thread Connection rispettivo che gestisce la connessione. Questo è possibile grazie alla lista di connessioni 'clientsConnected' all'interno della classe 'TCPServer'. Quando una finestra client viene chiusa o un user manda il messaggio "/quit", una 'EOFException' viene lanciata. Nel catch block dell'exception il server rimuove la connessione dalla lista clientsConnected e manda il messaggio "<client name>: Has left the chat" a tutti i client che sono ancora online. Questo atto termina il rapporto tra il cliente che si è disconnesso e il server.

4. Testing Feedback

I vari test eseguiti non hanno riportato alcuna anomalia nel funzionamento del programma, sia lato Client che lato Server.

5. Alternativa UDP

In questa applicazione l'uso del protocollo UDP è sconsigliato. Questo perchè con UDP c'è il rischio di perdere alcune informazioni e in una chat room perdere dati o parti di dati, come messaggi, potrebbe rendere l'applicazione inutilizzabile. Nel caso in cui la trasmissione fallisce, TCP proverà infatti ad inviare i dati ancora, cosa che UDP non prevede. Inoltre, TCP garantisce che i dati arrivano in maniera ordinata, caratteristica necessaria per un messaggio di testo.

Uno svantaggio di TCP è che è più lento a trasmettere i dati rispetto a UDP, visto che prevede l'handshaking e il congestion control. Ma comunque per la trasmissione di messaggi, ritardi sono abbastanza accettabili.

Per questi motivi, la maggior parte di social networks ai giorni d'oggi utilizza il protocollo TCP. Alcuni esempi:

- Whatsapp
- Telegram
- Instagram