Worth

Progetto di Laboratorio di Reti B 2020/21

Pace Antonio 559397

1. Introduzione

Il progetto è composto da un client e da un server.
Il client si connette al server con una connessione TCP alla porta 6789, inviando poi i comandi scelti dall'utente, fatta eccezione per:
- l'operazione di registrazione effettuata tramite RMI.
- Lettura e scrittura su chat, metodi implementati internamente al client, utilizzando le informazioni contenute nella lista `Chats` (vedi paragrafo 2)
- listUsers e listOnlineUsers, metodi implementati internamente al client, stampando `Userlist` (vedi paragrafo 2).
- listProjects, in quanto i nomi dei progetti sono associati agli oggetti contenuti nella lista `Chats`.

RMI è usato anche per effettuare le callback, necessarie per avvisare i client dello stato degli altri utenti e per notificare l'aggiunta a nuovi progetti o la rimozione degli stessi.

2. Classi

Oltre le già descritte `client` e `server` le altre classi sono:

- `ResponseHelper<T>:`

Rappresenta una risposta del server verso il client. Contiene sempre una stringa di risposta per comunicare il successo o il fallimento dell'operazione e una `List<T>` (`List<String>` in risposta ai metodi showCards, showMembers... e `List<Chat>` in risposta al login).

- `Card`

Rappresenta le card dei progetti: nome, descrizione, stato attuale e stati passati. Contiene inoltre metodi per cambiare stato e recuperare informazioni su una card.

- `Project:`

Rappresenta i progetti: ID, cards, membri, indirizzo e porta della chat, directory del progetto. Contiene metodi per la gestione del progetto (aggiunta e spostamento di cards, cancellazione directory, verifica membro...).

- `User:`

Rappresenta gli utenti: nickname, password e stato (boolean).

- `Utils`

Metodi utili (generazione numero random in un range, generazione indirizzo multicast, printList).

2.1 Interfacce

`Notify` e `ServerRMI`, interfacce implementate rispettivamente da ClientMain e ServerMain, contenenti le intestazioni dei metodi RMI.

3. Scelte implementative

Per quanto riguarda la gestione di connessioni multiple il server utilizza un selettore per il multiplexing dei SocketChannel.
I metodi per le callback sono synchronized per la possibilità che vengano eseguiti da più client.
I thread attivati sono quindi uno per il server e uno per ogni client.

Il server utilizza le seguenti strutture dati:

- `Projects`

Arraylist per salvare i progetti creati dagli utenti.

- `Users`

Arraylist per salvare gli utenti registrati ed il loro stato.

- `Clientstubs`
HashMap per salvare la corrispondenza fra Stub del client e nickname del relativo utente.

- `Socketnickname`
HashMap per salvare la corrispondenza fra SocketChannel aperta verso un client e il nickname del relativo utente.

Il client utilizza le seguenti strutture dati:

- `Chats`
Arraylist per salvare gli oggetti di tipo Chat relativi ai progetti di cui sono membri, ricevuta dopo il login e aggiornata alla ricezione delle callback (utente aggiunto ad un nuovo progetto).

- `Userlist`
HashMap per salvare gli utenti registrati ed il loro stato.
Viene ottenuta dal server in risposta alla register per le callback RMI.

Come già anticipato, client e server interagiscono tramite connessione TCP, il server riceve i comandi dal client, effettua il comando e invia il risultato.
Il risultato è inviato come oggetto `ResponseHelper`, dopo essere stato serializzato in JSON con il metodo `writeValueAsByte.` Verrà quindi deserializzato lato client.

4. Esecuzione

Viene eseguito ServerMain e successivamente ClientMain, senza argomenti. L'unica libreria esterna usata è Jackson.

All'avvio del client viene stampata la lista di possibili comandi.Per la chiusura del client è previsto il comando `quit.`