Documentazione progetto “WORTH”

Architettura del sistema

Il sistema è composto da due eseguibili: **client** e **server.**
Il **client** appena avviato effettua una connessione al server tramite TCP e si mette in attesa di comandi che arrivato da linea di comando. Digitando ‘help’ è possibile mostrare a video la lista di tutti i comandi disponibili, ma gli unici che verranno accettati inizialmente dal server sono login o register. Il comando register effettua una registrazione al servizio tramite RMI e in caso di successo inoltra il comando di login tramite TCP. Nella procedura di login il client si registra alle notifiche di tipo RMI callback che gli permetteranno di ricevere oggetti di tipo Notification. L’oggetto di tipo Notification contiene una lista degli utenti registrati con associato lo stato attuale che può essere online oppure offline, e una lista dei progetti di cui l’utente fa parte, associato all’indirizzo IP multicast della relativa chat.
All’arrivo dell’oggetto Notification il client aggiorna la sua lista locale di coppie <username, stato>, crea una nuova chat per ogni nuovo progetto a cui l’utente è stato aggiunto (se quindi è la prima notifica che riceve, creerà le chat per tutti i progetti di cui fa parte), elimina le chat dei progetti che sono stati eliminati.
Il **server,** quando viene avviato, ripristina i dati relativi ad utenti e progetti presenti nella cartella ‘storage’. Dopodiché gestisce le chiamate RMI tramite metodi synchronized e le richieste TCP tramite il multiplexing dei canali mediante NIO. Quando un client effettua il login con successo, la key del suo canale (presente nel selettore) viene salvata nell’oggetto di tipo User corrispondente all’utente per il quale ha richiesto il login. Key e User rimangono associati fino a che non viene effettuato un logout. Nello stesso modo anche alla richiesta di registrazione alla callback RMI l’interfaccia remota del client viene salvata nell’oggetto User corrispondente e viene cancellata nel momento dell’annullamento della registrazione. Le callback RMI vengono effettuate successivamente alla creazione o cancellazione di un progetto, all’aggiunta di un nuovo membro al progetto o al login da parte di un utente.

Classi

- **Card:** diretta implementazione delle card all’interno dell’applicazione.
- **Project:** diretta implementazione dei progetti all’interno dell’applicazione, ogni Project contiene una lista di oggetti card per ogni possibile stato delle card. Fornisce metodi per cambiare lo stato di una card, aggiungere una nuova card al progetto, aggiungere un membro al progetto, ottenere l’indirizzo IP multicast della chat relativa al progetto.
- **ProjectSet:** contiene la lista degli oggetti Project e ha i metodi per creazione e cancellazione di progetti, restituire tutti i progetti di cui fa parte un utente o restituire un progetto.
- **User:** diretta implementazione degli utenti all’interno dell’applicazione, ogni oggetto è caratterizzato da username, password, stato e interfaccia remota del client che, solo se l’utente è online, è diversa da null. Quest’ultima viene usata dal metodo notify() per mandare una notifica tramite RMI callback.
- **UserSet:** contiene la lista degli oggetti User e ha i metodi per login, logout, registrazione, mandare notifiche e ottenere la lista degli utenti. Contiene inoltre una lista di coppie <key, username> che rappresenta l’associazione tra gli utenti connessi ed il loro canale di connessione.
- **Notification:** usata per le RMI callback e contiene una lista di coppie <utente,stato> e una di coppie <nomeProgetto, IPchat>
- **Esito:** viene usata per rispondere alle richieste TCP. Contiene un boolean che rappresenta l’esito dell’operazione richiesta, un messaggio e una lista opzionale di stringhe aggiuntive.
- **ChatProcess:** viene utilizzata dal client per creare thread che rimangano in ascolto di messaggi sulla chat (Ogni client avrà quindi un thread per ogni progetto di cui l’utente fa parte).
- **StorageManager:** si occupa di ripristinare dati e aggiornare i file json presenti nella cartella storage.

- **ClientMain:** classe client che legge i comandi e li inoltra al server. Mantiene in locale una lsita degli utenti, del loro stato ed una lista delle chat di progetto.
- **ServerMain:** classe server, esegue le richieste del client utilizzando i metodi forniti da ProjectSet, UserSet e persiste il suo stato grazie a StorageManager.

Concorrenza

Ci sono due casi in cui viene gestita la concorrenza:

- Il client ha un thread per ogni chat che rimane in ascolto di pacchetti UDP sul gruppo multicast; quando arriva un messaggio, viene inserito nella struttura dati MessageQueue. Questa struttura funziona come coda che viene riempita e svuotata in modo sincrono per non creare inconsistenze.
- L'oggetto di tipo UserSet mantiene al suo interno la lista degli utenti registrati che deve esser aggiornata all'invocazione di register da RMI. Si potrebbe creare un conflitto con le richieste TCP che hanno bisogno di accedere alla lista degli utenti come register, notifyAll, getByUsername che sono infatti synchronized

Utilizzo

Il server deve esser mandato in esecuzione prima del client. Una volta in esecuzione stamperà la scritta "Server in attesa di connessioni sulla porta 1919". Successivamente stamperà un messaggio ogni volta che un client si connette o manda un comando.

```
Server in attesa di connessioni sulla porta 1919
Server: accettata nuova connessione dal client /127.0.0.1:58948
comando richiesto:login fabio 1234
Server: Login avvenuto con successo inviato al client: /127.0.0.1:58948
```

Il client una volta mandato in esecuzione stamperà "CLIENT CONNESSO" se è riuscito a connettersi al server. Dopodiché stamperà questo messaggio che dà subito un'idea di come utilizzare la linea di comando

```
CLIENT CONNESSO
------------------------------------------------------------
Digitare help per mostrare la lista dei comandi disponibili
Digitare exit per uscire
>
```

Il client accetta come primi comandi solo quelli di register o login.