Relazione progetto reti WORTH

Fabio Bocci Matricola: 580100

Architettura del sistema:

Il progetto WORTH è un sistema Client-Server dove più client si possono connettere al server centrale per gestire i propri progetti e "task" (chiamati d'ora in avanti CARD). Il Server dopo aver identificato il client attraverso un Login 'Username' 'Password' permetterà all'utente di Modificare, Creare o Eliminare/Chiudere progetti (NB: per chiudere un progetto serve che tutte le card siano in stato DONE). Se un utente non è Registrato nel sistema è possibile utilizzare la funzione Register 'Username' 'Password' per creare il nuovo utente. Ogni utente ha la possibilità di utilizzare una Chat relativa ad ogni progetto di cui fa parte (Una per ogni progetto diverso), la chat è implementata utilizzando un Thread in ascolto per ogni chat, ed una struttura di support (**MessagingQueue**) per memorizzare e far leggere i messaggi all'utente.

Il server utilizza due nuove classi di support: **User**, **Project**. Ogni istanza di Project identifica in modo univoco un progetto all'interno del DB. Ogni istanza di User identifica in modo univoco un utente registrato nel DB.

Quando il Server verrà acceso controllerà e leggerà ogni singolo progetto salvato nel DB e ogni singola associazione utente-Password nel DB, inoltre metterà a disposizione una interfaccia RMI per: la registrazione di un nuovo utente, per la registrazione ed eliminazione di interfacce per le CallBacks dei client successivamente si metterà in ascolto di istruzioni dai client.

Classi Create

Per la realizzazione del progetto WORTH ho creato delle classi di supporto:

- **Card** -> Classe usata per la lettura e scrittura dei file Json delle CARD, contiene:
  - *Il nome della CARD*
  - *Descrizione*
  - Stato attuale (TODO-INPROGRESS-TOBEREVISITED-DONE)
  - Una lista contenente la storia degli stati
- **Project** -> Classe che identifica un progetto nel DB, il costruttore permette la costruzione della cartella con il nome del progetto se non già esistente. Contiene:
  - *Una lista di CARD*
  - *Una lista di string con tutti gli utenti appartenenti al progetto*
  - Tutti i metodi che si possono invocare su esso (Es. MoveCard AddMember IsMember ...)
- **User** -> Classe usata per lettura e scrittura del file Json contenente tutti gli username con relative password. Contiene
  - *Username*
  - *Password*
  - *Una variabile boolean che identifica se l'utente è Online o no*
  - La relativa interfaccia per le callbacks di quel utente
- **ServerMain** -> Classe main del server. Contiene:
  - *Lista di Project*
  - *Lista di User*
  - Metodi condivisi con l'interfaccia RMI
  - Metodi per gestire i comandi dei diversi client connessi
- **MulticastIpGenerator** -> Classe usata per generare sempre nuovi ip di Multicast tutti differenti

- **MessagingQueue** -> Classe di support utilizzata per rendere le operazioni di inserimento e lettura atomiche. Contiene:
  - *Lista di string, i messaggi della chat*
  - Metodi di put
  - Metodi di read
- **ChatThread** -> Classe usata lato client per generare un nuovo thread per ogni chat. Contiene:
  - *MessagingQueue*
  - *MulticastSocket*
  - Metodi per inviare messaggi sulla chat
  - Metodi per rimanere in ascolto sulla chat ed inserire i nuovi messaggi sulla MessagingQueue
- **ClientMain** -> Classe main del client. Contiene:
  - *Map relativa a gli utenti e il loro attuale stato (Online-Offline)*
  - *Due Map relative ai progetti di cui si fa parte utilizzate per la MessagingQueue e il ChatThread*
  - Metodi condividi con il server tramite l'interfaccia RMI
  - Metodi per leggere e gestire ogni comando dell'utente
- **Excpetions** -> inoltre sono state create diverse eccezioni per i diversi casi tipo:
  - Il non ritrovamento di una CARD dentro un progetto
  - Il movimento di una CARD da uno stato ad un altro illegale
  - La creazione di un nuovo utente già presente nel DB degli utenti
  - Il non ritrovamento di un progetto
  - La creazione un nuovo progetto con un nome già utilizzato
  - Il non ritrovamento di un utente

Thread

Server

Il Server in questo progetto è costruito come un unico Thread che rimane in ascolto dei Client attraverso la connessione TCP, utilizzando i Selector per gestire più client allo stesso tempo. Inoltre essendo il server anche utilizzabile come RemoteObject tramite l'interfaccia RMI è stato opportuno mettere tutti i metodi sia quelli dell'interfaccia RMI, sia quelli chiamabili dal Thread main come Syncronized evitando in questo modo problemi di corse critiche .

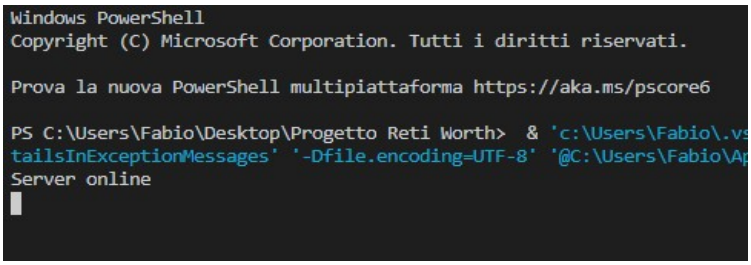
Client

I client sono progettati principalmente come un unico Thread che legge ed invia i comandi inseriti da console. Essedo anche i client dei RemoteObject e quindi chiamabili dal server tramite delle callbacks ho implementato i metodi di Excetute e delle callbacks come Syncronized evitando problemi di corse crittiche.

La chat invece essendo un thread a parte per ogni chat, ho implementato la MessagingQueue con solo metodi Syncronized in questa maniera solo un thead per volta puo accedere alla relativa Queue per depositare un nuovo messaggio o leggere tutti i messaggi presenti

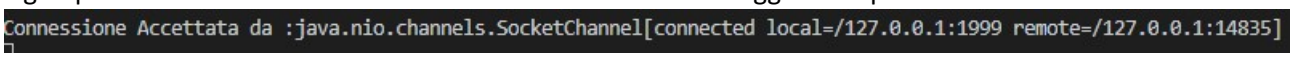
Come utilizzare?

Per testare il programma bisogna prima mandare in esecuzione il server, quindi quando il server è pronto (stamperà la scritta "Server online").



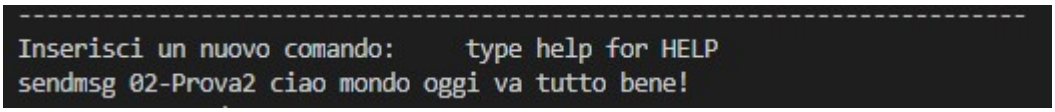
Si potranno mandare in esecuzione i diversi client (sia i client che il server sono settati automaticamente con l'indirizzo di "Localhost" e con porta 1999).

Ogni qualvolta un client si connette al server arriverà un messaggio del tipo:

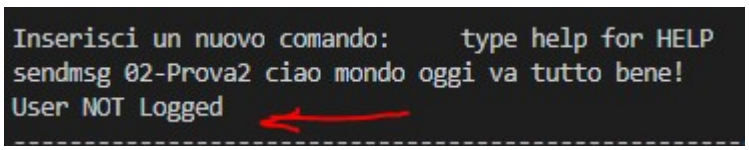


Ed il client aspetterà che si scriva un comando. È stata aggiunta una funzione di Help per avere la lista dei comandi. Tutti i comandi non sono CaseSensitive a differenza degli username, password, nome dei progetti e nome delle CARD.

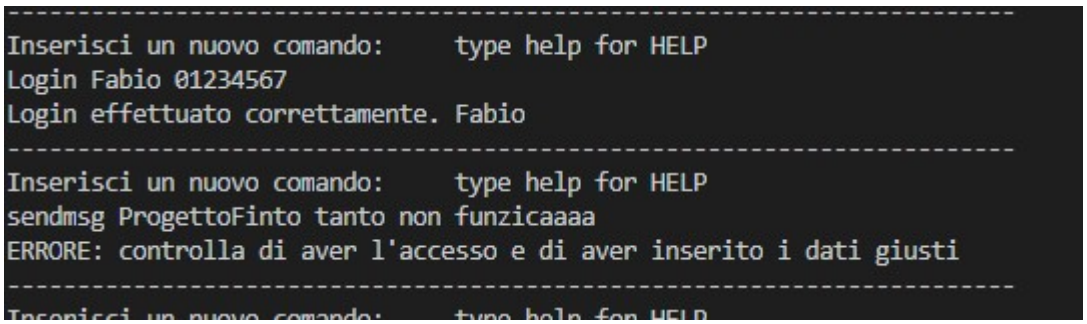
Per comandi come AddCard e SendMSG dove si vuole scrivere la descrizione della carta o il messaggio da inviare, non serve aggiungere nessun carattere speciale per identificare l'inizio e la fine del messaggio/descrizione, farà da solo il programma leggendo ogni singola parola inserita dopo i dati di destinazione:



Bisogna aver effettuato il Login prima di eseguire tutti i comandi (ad eccezione dei comandi come: Register Login Exit Help) altrimenti riceveremo un segnale del tipo:



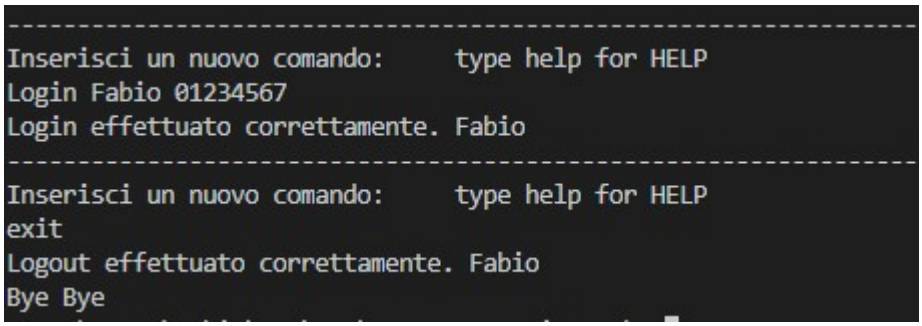
Invece se si è fatto l'accesso ma si vuole accedere ad un progetto di cui non si fa parte riceveremo un segnale di questo tipo:



Per il corretto utilizzo del programma si chiede di evitare di utilizzare Username Password NomiProgetti e NomiDelleCARD con degli spazzi all'interno e si prega di evitare l'utilizzo dei seguenti caratteri: #, &.

Infine è stato aggiunto anche un comando nascosto, non inserito nei comandi scritti in HELP, chiamato SaveAll. Serve semplicemente a salvare tutti i dati del server: progetti, CARD, user.

Quando un client si staccarsi, basta digitare il comando exit. In questo modo il ciclo continuo di lettura di comandi terminerà, e il client si spengerà. (NB: non è obbligatorio utilizzare il comando logout prima del comando exit poiché controllerà se si è loggati e se lo siamo lo eseguirà in automatico)



Librerie

Per il progetto non sono state utilizzate librerie se non quelle di java base ad eccezione delle librerie per l'utilizzo delle Json.

Per il corretto funzionamento del programma si chiede di imporle nell'ambiente in cui si eseguirà il Server.

Le librerie più utlizzate sono state quelle per la comunicazione Client-Server java.nio.* , java.rmi.*. inoltre è stata utilizzata molto la libreria java.util.* per l'elaborazione dei dati.

Possibili miglioramenti

Il Progetto WORTH è un sistema che permette di gestire ai diversi User, dei progetti con relative task.

Sarebbe possibile in futuro aggiungere un'interfaccia grafica ai client e permettere alle card di avere oltre la descrizione delle immagini che aiutino a capire di cosa si stia parlando, stessa cosa per i progetti. Un'altra miglioria possibile potrebbe essere l'aggiunta di chiamate e videochiamate/presentazioni per ogni chat di progetto.