WORTH: WORkTogetHer

Stefano Pea

A.A. 2020/2021

Indice

Capitolo 1

architettura del Sistema

Il progetto prevedeva l'implementazione di uno strumento per la gestione di progetti condivisi con altri utenti: Il sistema e' basato sul paradigma Client-Server e gli utenti utilizzano il Client tramite interfaccia da riga di comando per inviare le richieste al Server, che si occupera' di elaborarle e restituire a quest'ultimo le informazioni richieste.

La maggior parte della comunicazione fra questi due oggetti avviene tramite protocollo TCP, con alcune funzioni che utilizzano RMI(registrazione di un utente al servizio) e UDP Multicast(invio e ricezione di messaggi delle chat di gruppo).

Il Client invia le sue richieste tramite oggetti di tipo 'request' mentre il Server risponde con oggetti 'response'. Ciascuno dei due poi si occupa di interpretare i comandi e le risposte che questi oggetti rappresentano.

Per conoscere con precisione tutte le varie operazioni che e' possibile eseguire e' possibile inviare il comando 'help' per ricevere dal Server la lista completa dei comandi con una breve descrizione.

Capitolo 2

Connessioni

Come gia' specificato nel progetto sono presenti vari tipi di connessioni, che sono state riportate di seguito insieme ad una rapida overview del loro utilizzo:

2.1 Java RMI

il comando register(nomeUtente, password) utilizza Java RMI per registrare un nuovo utente al servizio Worth, ma solo se non ne e' gia' presente uno con lo stesso nome.

2.2 Java RMI Callback

Il sistema utilizza le callback per inviare ai vari Client la lista degli utenti(OFFLINE/ONLINE) nel momento in cui si connettono con il Server e quando viene effettuato un login o logout da parte di un utente, in modo da avere ognuno la loro lista di utenti con il relativo status.

2.3 TCP

All'attivazione di un Client questo si connette tramite protocollo TCP con il Server fino alla sua chiusura. Quasi tutte le operazioni che e' possibile effettuare tramite il programma utilizzano questo tipo di connessione. Per gestirle il Server effettua un multiplexing delle richieste dei Client grazie ad un selettore, cosi' da essere efficiente e scalabile.

2.4 UDP multicast

UDP Multicast e' utilizzato esclusivamente per le chat che appartengono ad ogni progetto, e che permettono a ciascun membro di conversare con gli altri.

Quando un utente crea un progetto oppure effetua il login viene attivato un thread per ogni chat e si mette in ascolto in attesa di messaggi.

Per inviare un messaggio alla chat occorre utilizzare il comando 'sendMessage nomeProgetto messaggio'.

I messaggi cosi' inviati vengono salvati in una struttura dati e poi riportati all'utente in seguito al comando 'readMessages nomeProgetto'.

Capitolo 3

Implementazione

3.1 Interfaccia

Per il progetto e' stata scelta una interfaccia a riga di comando (CLI), come e' possibile vedere dall'immagine sottostante.

```
Client: connected
Type 'exit' to quit the program or 'help' for a list of commands:
register Stefano passwordsicura
Server response: [ok]
login Stefano passwordsicura
Server response: [SUCCESS: login successful]
createProject nuovo_progetto
Server response: [SUCCESS: project created successfully, nuovo_progetto, 224.0.0.1, 4002]
addCard nuovo_progetto report descrizione breve della nuova card
Server response: [SUCCESS: report added to the project]
```

3.2 comunicazione

La comunicazione tra Client e Server avviene attraverso l'invio e la ricezione di oggetti di tipo 'request' e 'response'.

Il Client quando invia una richiesta al Server invia per prima cosa la dimensione dell'oggetto e poi la request, per permettere al Server di allocare il buffer per la lettura.

Le request vengono create dal Client e contengono l'username dell'utente che ha inviato l'oggetto, il comando da inviare al Server e la lista degli argomenti necessari alla richiesta.

Le response invece contengono un codice relativo al successo o no dell'operazione, la lista di argomenti da ritornare al Client e altri due argomenti utilizzati nel momento in cui viene richiesto un login: 'nProg' indica quanti progetti hanno come membro l'utente che ha richiesto il login e 'addresses' e' un' Arraylist di stringhe che contiene $\langle NomeProgetto, indirizzoMulticast, Porta \rangle$.

Come gia' accennato quest'ultima struttura dati e' utilizzata per mettersi in ascolto delle chat di cui fa parte l'utente.

(Per maggiori dettagli su 'request' e 'response' si consiglia di vedere le classi corrispondenti)

3.3 Persistenza Sistema

Il sistema garantisce la persistenza delle informazioni in questo modo:

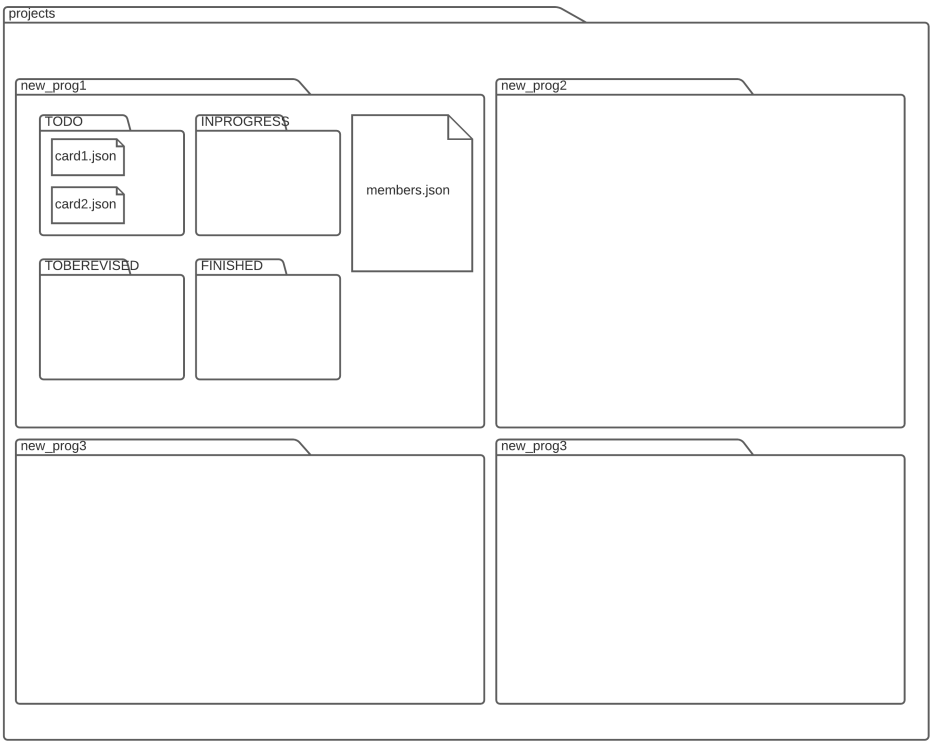
- Per prima cosa viene creato un file JSON contenente la lista degli utenti registrati al servizio.
- Viene inoltre creata una directory "projects" che contiene le sottocartelle che rappresentano i progetti stessi .
  - Queste cartelle contengo ulteriori sottocartelle nominate "TODO", "INPROGRESS", "TOBEREVISED" e "FINISHED",che rappresentano le quattro possibili liste in cui puo' essere salvata una card, oltre ad un file JSON contenente i membri che appartengono al progetto.
    - Dentro ognuna di queste cartelle sono presenti i file JSON relatevi alle card appartenenti al progetto(contenenti una hashmap con descrizione della card e history di quest'ultima).

Le principali funzioni che si occupano di queste operazioni sono:

backupProjects() chiamata quando si crea un progetto, quando si aggiunge un membro ad un progetto, quando si aggiunge una card al progetto oppure quando una di queste viene spostata in un'altra lista.

backupUsers() chiamata al momento della registrazione di un nuovo utente.

restoreBackups() chiamata all'attivazione del Server per ricostruire lo stato precedente.



La figura sopra mostra in maniera semplificata come sono suddivise le varie cartelle per garantire la persistenza.

3.4 Sicurezza

Visto lo scopo di questo progetto le password sono inviate in chiaro al Server; nel caso di un utilizzo non didattico per aumentare la sicurezza del sistema si potrebbe effettuare un hashing delle password, rendendo quindi sicure le informazioni degli utenti.

Capitolo 4

Threads e concorrenza

4.1 Server

Il Server gestisce le varie connessioni TCP con i client tramite multiplexing dei canali con NIO. Per quanto riguarda la concorrenza fra le varie operazioni, quelle problematiche sotto questo punto di vista sono state implementate tramite metodi di tipo Synchronized.

4.2 Client

Il Client al momento del login avvia un thread per ogni progetto di cui e' membro l'utente, che rimane in ascolto per i messaggi della relativa chat. Un thread viene creato anche nel momento in cui si richiede la creazione di un progetto. Da notare che quelli sopra elencati sono gli unici modi che un utente ha per conoscere l'indirizzo della chat dei progetti.

Capitolo 5

Descrizione Classi

Di seguito e' riportata una breve descrizione delle classi che compongono il Sistema(per aver maggiori informazioni riguardo i vari campi o metodi delle classi si consiglia di controllare il codice):

card/cards Classi che si occupano della creazione delle card, delle liste che le contengono e dei metodi per aggiungerle, rimuoverle, ecc...

chatAddress Classe che si occupa di generare indirizzi e porte per le varie chat dei progetti.

client Classe che implementa tutte le operazioni relative al Client.

clientMain Main class del Client.

multicastThread/listener Classi che si occupano della ricezione dei messaggi delle chat.

member/members Classi che si occupano delle funzioni inerenti all'aggiunta e alla rimozione di membri da un progetto.

notifyEventInterface Interfaccia relativa al Callback.

project/projects Classi che si occupano sia delle funzioni relative al singolo progetto(quindi ricerca e aggiunta di card, di membri, rimozione di quest'ultimi ecc...) che della lista di tutti i progetti presenti nel sistema.

request Classe che si occupa della costruzione e dei metodi relativi ad una request, ovvero la struttura che utilizza il Client per inviare il comando da eseguire al Server.

response Classe che si occupa della costruzione e dei metodi relativi ad una response, ovvero la struttura che utilizza il Server per inviare il risultato della request effettuata dal Client.

server Classe che implementa tutte le operazioni relative al Server.

serverInterface Interfaccia relativa al Server, in particolare alle callback e alla registrazione di un utente.

serverMain Main class del Server.

user/users Classi che si occupano della gestione delle operazioni sugli utenti quali l'aggiunta e la rimozione del sistema.

Capitolo 6

Utilizzo del Programma

6.1 Librerie esterne

L'unica libreria esterna utilizzata e' Jackson per lavorare con JSON ed e' gia' fornita con il programma all'interno della cartella 'jackson'.

6.2 Esecuzione del programma

Per eseguire il programma e' sufficiente recarsi all'interno della cartella in cui e' stato salvato ed eseguire gli script bash presenti in quest'ultima:

`compileAll.sh`

`serverStart.sh`

`clientStart.sh`

`serverStop.sh`

In particolare occorre per prima cosa eseguire compileAll.sh che si occupa di compilare tutti i file necessari per il programma, e poi serverStart.sh e clientStart.sh che avviano rispettivamente il Server e il Client. Alla chiusura del programma e' inoltre possibile eseguire serverStop.sh per uccidere il processo Server.