Nicoletta Carrubba 578287

UNIVERSITÀ DI PISA

Università di Pisa
Corso di Laura in Informatica
Progetto di Laboratorio di Reti di Calcolatori
A.A. 2020/21

WORTH: WORkTogetHer

Indice

1.Introduzione

WORTH (WORkTogetHer) è un progetto didattico che consiste nell'implementazione di uno strumento per l'organizzazione e la gestione di progetti in modo collaborativo.
Questo è stato realizzato ispirandosi al metodo Kanban cioè un metodo di gestione agile che fornisce una vista di insieme delle attività e ne visualizza l'evoluzione. Per questa ragione le principali caratteristiche del metodo sono state implementate come segue.

2.Sviluppo

Il progetto è stato sviluppato considerando due sistemi comunicanti, il main Client e il main Server che scambiano messaggi di richiesta e risposta attraverso una connessione TCP.
Quando viene avviato il server questo crea una welcoming socket che consente di ricevere richieste di contatto dai client che non appena vengono avviati faranno richiesta di connessione, a questo punto quando il server accetta le richieste creerà nuove connection socket dedicate ai singoli client.

3.Main Client

Gli utenti si interfacciano con i Client attraverso linea di comando. Al momento dell'avvio solo due operazioni vengono effettuate senza generare errori, l'operazione di registrazione e l'operazione di login.
L'operazione di registrazione a worth permette ad un utente di identificarsi tramite username e password. Quest'operazione viene fatta invocando il metodo *registerUser(Username,Password)* dell'oggetto remoto *Registration*. Il metodo si occupa di creare una nuova istanza utente della classe *utente* e dopo aver accertato che la coppia di parametri in input username e password siano validi, quindi non già presenti all'interno del database utenti, inserisce il nuovo utente all'interno di quest'ultimo e serializza il database all'interno del file *"./src/Backup/utentiregistrati.json"* per renderlo consistente. Infine, se la richiesta di registrazione sarà andata a buon fine, il main Client chiederà al server tramite richiesta TCP di aggiornare gli altri utenti dell'avvenuta registrazione di un nuovo utente.
L'operazione di login permette agli utenti di risultare online e di poter avere accesso ai servizi offerti dal server. Quest'operazione viene richiesta al server, che dopo aver controllato la validità di username e password, cambierà lo stato dell'utente in online. Una volta ricevuta la risposta, se questa conterrà la dicitura "Operation login done succesfully" il main Client farà richiesta di registrazione al servizio di notifiche implementato attraverso RMICallback e invierà attraverso TCP altre due richieste al

Server. La prima richiesta, *update*, richiede al server di notificare gli altri utenti dell'avvenuto login, la seconda, *getChat*, richiede al server di inviare i progetti e gli indirizzi IP multicast associati ad essi e inserirà l'associazione nome del progetto e indirizzo all'interno dell'hashmap *projectAddress*. Infine con l'invocazione del metodo *joinChats()* il clientMain si occuperà delle chat dei progetti, più in dettaglio, creerà un task *ProjectChat* per ogni progetto all'interno di *projectAddress,* lo passerà ad un thread e inserirà l'associazione Nome del progetto e chat del progetto dentro l'hashmap *pjChat.*
Dopo aver eseguito il login l'utente avrà accesso sulla base dei permessi che possiede ai servizi di worth.

4.Main Server

All'avvio del Main Server come prima operazione viene chiamato il metodo *backup()*, questo si occupa di ripristinare lo stato del sistema utilizzando le informazioni all'interno della cartella *"./src/Backup"* e deserializzando i file contenuti in essa, più in dettaglio:

- Lo stato del database degli utenti registrati viene ripristinato attraverso il file *"utentiregistrati.json"* che al suo interno contiene l'associazione username e password, l'informazione sullo stato degli utenti non viene salvata perché ritenuta superflua;
- Lo stato dei progetti creati viene ripristinato osservando le cartelle, ogni cartella corrisponde ad un progetto e a loro interno sono contenuti dei file riguardanti le cards e i membri del progetto. Le cards sono memorizzate come file json denominati come "Card_nomeCard" e all'interno si trovano tutte le informazioni utili a deserializzare un istanza di tipo Card. I membri sono memorizzati in un file json *"members.json"* che contiene tutti gli username degli utenti membri del progetto. Per ogni progetto viene poi generato un nuovo indirizzo IP multicast.

Eseguita la prima fase di ripristino dello stato del sistema, il server gestirà le connessioni RMI e TCP. Infine poichè il server deve essere in grado di gestire diverse richieste provenienti da più client, viene implementato come multithreaded utilizzando il threadpool di tipo cached a cui viene sottomesso come task quello individuato dalla classe *operations*.

5.Operations

Operations è una classe che implementa l'interfaccia *Runnable* e le sue istanze rappresentano i task mandati in esecuzione dal MainServer quando viene accettata una nuova connessione tcp con un client.
All'istanza di operations viene passata la socket per la comunicazione con il client, in questa i messaggi verranno inviati con il formato "comando parametri" e successivamente separati tramite il metodo split(" "), l'unico messaggio che viene passato differentemente è quello riguardante il comando addCard che viene passato

con il formato “comando,parametri” per non separare la descrizione che potrebbe riportare degli spazi.
Dopo aver ricevuto il comando viene chiamato il metodo corrispondente per rispondere alla richiesta, ogni metodo contiene dei controlli riguardanti i permessi dell’utente e nel caso in cui questi non siano conformi alle richieste vengono generati messaggi di errore.
I thread in esecuzione che gestiscono le richieste dei client condividono il database utenti *udb* e l’hashmap *projects* contenente tutti i progetti presenti in worth, quindi per permettere accessi controllati e che non generano incostistenze, sono stati utilizzati i metodi di sincronizzazione synchronized.
Il metodo synchronized è stato utilizzato sul database utenti per i metodi di login e logout e sull’hashmap *projects* per gli altri metodi.
Di seguito vengono riportati i metodi più significativi:

- *createProject(String projectName):* Chiamato quando un utente fa richiesta di creazione di un progetto. All’interno del blocco sincronizzato viene creata una nuova istanza della classe *project* e gli viene assegnato un nuovo indirizzo IP multicast tramite l’utilizzo del metodo *newAddress()* dell’istanza *chatAddress.*
  Infine, viene reso persistente il progetto creando all’interno della cartella *“./src/Backup”* una nuova cartella con il nome del progetto e un file *“members.json”* che contiene i membri del progetto.
- *addMember(String projectName,String nickUtente):* Chiamato quando un utente fa richiesta di aggiungere un nuovo membro ad un progetto. All’interno del blocco sincronizzato, come prima operazione viene aggiunto l’utente come nuovo membro del progetto tramite il metodo *newMember(Utente)* dopo si serializzano i membri del progetto all’interno del file member.json e infine si notifica il nuovo membro attraverso RMI dell’avvenuta operazione.
- *addCard(String projectName,String cardName,String descrizione):*Chiamato quando un utente fa richiesta di aggiungere una nuova Card ad un progetto. All’interno del blocco sincronizzato viene creata un’istanza della classe *Card* tramite il metodo *newCard()* della classe *Project,* a questo punto viene inizializzata la storia della card che è stata sviluppata con il supporto della classe *CardLog.* Infine la card viene serializzata come nuovo file json per renderla consistente.
- *cancelProject(String projectName):* Chiamato quando un utente fa richiesta di eliminazione di un progetto. All’interno del blocco sincronizzato dopo aver controllato che tutte le cards siano nella lista “DONE” si procede con l’eliminazione del progetto da *projects* e con la cancellazione della cartella di backup data dal nome del progetto. Se queste operazioni sono andate a buon fine si aggiornano tutti gli utenti membri al progetto dell’avvenuta cancellazione in modo tale che le loro strutture dati contenenti le chat del progetto possano essere sempre consistenti.
- *getChat():* Chiamato per richiedere gli indirizzi IP multicast di tutti i progetti di cui l’utente è membro.
- *Reply(BufferedWriter writer, int result, String operation):* Utilizzato come metodo di supporto per rispondere con messaggi standard alle richieste degli utenti.

6.Rmi

I metodi remoti sono stati utilizzati per due operazioni, l'operazione di registrazione e l'implementazione di un servizio di notifica.
Per la registrazione è stata definita un'interfaccia remota *Registration_interface* e una classe *Registration* che implementa *Registration_interface*. L'unico metodo all'interno di questa classe è il metodo *registerUser* che è stato precedentemente descritto.
L'istanza registration viene esportata come oggetto remoto che viene collegato al nome simbolico "REGISTRATION" subito dopo l'operazione di ripristino del sistema dal server.
Il servizio di notifica utilizza il meccanismo delle callback,vengono definite due interfacce remote, *NotifyServer_interface* utilizzata dal client per registrarsi al servizio di notifica e *NotifyClient_interface* utilizzata dal server per notificare i client attraverso i metodi remoti. L'interfaccia *NotifyServer_interface* viene implementata dalla classe *NotifyEventServer* e l'interfaccia *NotifyClient_interface* viene implementata dalla classe *NotifyEventClient.* Un'utente si registra al servizio di notifica al momento del login e si disiscrive al momento del logout.

All'interno della classe *NotifyEventServer* si utilizza l'hashmap *clientsRegistrati* contenente un'associazione tra username Utente e *NotifyClient_interface* in modo tale che la notifica possa essere mandata anche attraverso la conoscenza del nome utente.
Sono stati definiti tre metodi remoti per le notifiche ai client e tre metodi per chiamarli:

- *NotifyEventClient.notifyClient(UtentiDB udbUpdated)* aggiorna tutti i client registrati al servizio di notifica quando avviene un cambiamento di stato degli utenti. Viene chiamato da *NotifyEventServer.update(UtentiDB udb).*
- *NotifyEventClient.notifyMember(String projectName,String address)* aggiorna l'utente che è stato aggiunto come membro di un progetto e crea la nuova chat del progetto per l'utente. Viene chiamato da *NotifyEventServer.updateMember(String username, String projectName,String Address).*
- *NotifyEventClient.notifyCancel(String projectName)* aggiorna gli utenti membri di un progetto dell'avvenuta cancellazione di quest'ultimo. Viene chiamato da *NotifyEventServer.updateMembers(LinkedList<String>members,String projectName).*

7.Chat

Le chat di progetto sono state implementate con UDP multicast,in modo tale che i client possano inviare datagrammi UDP ad un gruppo definito dai membri di un progetto.
E' stata definita la classe *ProjectChat* che implementa l'interfaccia runnable, infatti per facilitare la lettura dei messaggi le chat sono state gestite come thread. All'interno della classe viene gestita tutta la fase di instaurazione della comunicazione, nel metodo

costruttore si fa una join del gruppo del progetto definito dall'indirizzo IP multicast passato come parametro di input.
Il metodo run() contiene un ciclo che permette la continua lettura di messaggi tramite il metodo receive, quando un nuovo messaggio arriva viene inserito nella linkedList *chatMsgs,* che rappresenta i messaggi ancora non letti. Per fermare il ciclo deve essere invocato il metodo *stop()* questo cambia la condizione del ciclo e manda un messaggio fittizio in modo tale da risvegliare la receive che è bloccante. Inoltre sono presenti i metodi *sendMessage()* e *readMessages()* utilizzati per mandare e leggere i messaggi.
Per scambiare le informazioni riguardanti le chat sono state utilizzate sia la connessione TCP sia RMIcallback, infatti la prima viene utilizzata quando un utente esegue il login per richiedere gli indirizzi IP dei progetti di cui è membro in quanto il modello richiesta-risposta è stato considerato adeguato al compito, la seconda in quanto il modello a notifica asincrona viene ritenuta conveniente quando ci sono dei cambiamenti di stato come l'aggiunta di un nuovo membro ad un progetto o la cancellazione di un progetto.

8.Directories

La directory principale del progetto è *src* che contiene le tre classi principali *MainClient*, *MainServer* e *Operations* e altre sotto-directory.
Quest'ultime sono :

- *Backup,* contiene tutti i file che devono essere utilizzati per ripristinare lo stato del sistema dopo un riavvio.
- *Interfaces,* contiene le interfacce.
- *Rmi,* contiene le classi utilizzate per gestire i meccanismi RMI e RMI callback.
- *Support,* contiene le classi che descrivono le istanze principali di worth.

8.1 Support

Le classi all'interno della directory Support sono:

- *Card,* contiene le informazioni riguardanti una card, cioè il nome, la descrizione, la lista corrente, la linkedList<*CardLog*>, i metodi getters e setters e altri metodi considerati convenienti nell'implementazione.
- *CardLog,* contiene le informazioni riguardanti la storia di una card, cioè la lista di partenza, la lista di destinazione, il membro che ha eseguito lo spostamento e i metodi getters.
- *Utente,* contiene le informazioni riguardanti un utente, cioè username,password, status e i metodi getters e setters.
- *UtentiDB,* simula un database utenti quindi come tale contiene tutte le istanze utenti nella linkedList<utenti> *utenti* e i metodi utili per l'interazione con la classe.

- *ChatAddress*, utilizzata per generare indirizzi IP multicast, questi vengono generati a partire dall'indirizzo 224.0.0.0 utilizzando un incremento graduale dei byte che compongono l'indirizzo.
- *Project*, contiene le informazioni riguardanti un progetto cioè il nome, l'indirizzo IP multicast, i membri e le quattro liste "TODO, INPROGRESS, TOBEREVISED, DONE". Queste sono state implementate come hashmap<String,Card> per semplificare la ricerca delle cards, la chiave è il nome della card. Inoltre contiene i metodi per l'interazione con il progetto.
- *ProjectChat*, implementa l'interfaccia runnable e viene utilizzata per implementare la chat di un progetto, contiene l'indirizzo ip della chat multicast, la lista dei messaggi ancora non letti, la multicastSocket, l'inetAddress e la porta utilizzati per la gestione della multicastSocket .

9.Compilazione ed esecuzione

Il progetto è stato sviluppato e testato tramite l'ambiente di sviluppo Eclipse per windows ed utilizza le librerie esterne jackson. I jar della libreria sono stati inseriti nella cartella "lib" e sono jackson-core-2.9.7.jar, jackson-annotations-2.9.7.jar e jackson-databind-2.9.7.jar.

Dopo essersi posizionati nella cartella contenente il progetto, è possibile compilare il codice utilizzando il seguente comando:

```
javac --release 8 -d "bin" -cp ".\lib\jackson-core-2.9.7.jar;.\lib\jackson-annotations-
2.9.7.jar;.\lib\jackson-databind-2.9.7.jar" .\src\*.java .\src\Support\*.java .\src\Rmi\*.java
.\src\Interfaces\*.java
```

Nel comando è presente il flag --release 8 che configura automaticamente il compilatore a produrre classi di una data versione, questo è stato specificatamente inserito in relazione alle impostazioni della macchina in cui il progetto è stato sviluppato.
Inoltre si fa uso del flag -d per impostare una cartella di destinazione per i file class e -cp per specificare un classpath per i file jar.

A questo punto sarà possibile avviare il server con il seguente comando:

```
java -cp ".\lib\jackson-core-2.9.7.jar;.\lib\jackson-annotations-2.9.7.jar;.\lib\jackson-databind-2.9.7.jar;bin"
src.MainServer
```

e i client con il seguente comando :

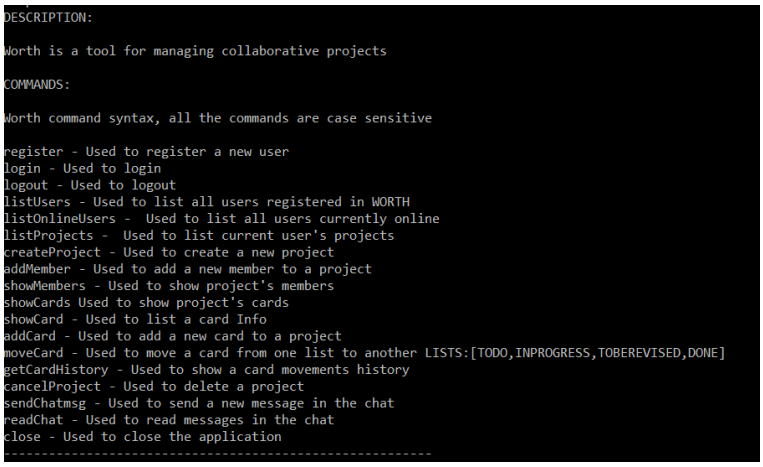
java -cp ".\lib\jackson-core-2.9.7.jar;.\lib\jackson-annotations-2.9.7.jar;.\lib\jackson-databind-2.9.7.jar;bin" src.MainClient

I comandi sono stati riportati nella pagina 9 in modo tale che possano essere copiati ed incollati con facilità.

E’ importante considerare le proprietà del sistema, l’esecuzione del server deve sempre precedere l’esecuzione dei client.

10.Comandi

Avviato il client è possibile richiedere la sintassi dei comandi digitando “help”, i comandi sono key sensitive e nel caso in cui verrà inserito un testo diverso da quelli nella lista verrà generato un messaggio di errore.



I parametri di ogni singolo comando verranno richiesti dopo la digitazione di quest’ultimo, quindi basterà scrivere i comandi che si trovano nella lista help e poi si verrà guidati nella compilazione della richiesta.

```
javac --release 8 -d "bin" -cp ".\lib\jackson-core-2.9.7.jar;.\lib\jackson-annotations-2.9.7.jar;.\lib\jackson-databind-2.9.7.jar" .\src\*.java .\src\Support\*.java .\src\Rmi\*.java .\src\Interfaces\*.java

java -cp ".\lib\jackson-core-2.9.7.jar;.\lib\jackson-annotations-2.9.7.jar;.\lib\jackson-databind-2.9.7.jar;bin" src.MainServer

java -cp ".\lib\jackson-core-2.9.7.jar;.\lib\jackson-annotations-2.9.7.jar;.\lib\jackson-databind-2.9.7.jar;bin" src.MainClient
```