Università di Pisa

Relazione per il progetto di reti WORTH

Samuele Iacoponi - 531852 - corso B

A.A 2020/2021

Indice

Capitolo 1

Introduzione

Relazione per descrivere il progetto di reti

1.1 Descrizione del problema

Il progetto consiste nell'implementazione di un programma che sfrutta la metodologia kanban. Tramite il metodo Kanban si rovescia il punto di osservazione e si concepisce quello produttivo come un processo che va da valle a monte in cui si svolgono le attività necessarie solo nel **momento in cui ce n'è effettivamente bisogno.** [2]

1.2 Scelte progettuali

Il server ho deciso di renderlo multithread invece di utilizzare il multiplexing dei canali con NIO. Ho scelto anche di utilizzare una **Command Line Interface (CLI)** al posto di una **GUI** usando uno switch all'interno del server e del client dove gestisco tutti i possibili comandi che il progetto comprende. Per aiutare l'utente ho inserito anche un comando **help** che va a stampare tutte i possibili comandi che è possibile inserire.
Dato che il server deve essere **persistente** sui progetti e sugli utenti, per garantire questa proprietà ho salvato le strutture all'interno di vari file **.json** che il server andrà a creare nella prima esecuzione oppure andrà a recuperare e leggere ogni volta che verrà riavviato.
Per implementare la registrazione di un nuovo utente usando l'RMI ho creato l'interfaccia che comprende il metodo **register** e i metodi necessari per le **callback.**
Per le operazioni di serializzazione/deserializzazione ho deciso di utilizzare le librerie esterne **GSON**.
Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson is an open-source project Gson can work with arbitrary Java objects including pre existing objects that you do not have source code of.[1]

Ho utilizzato diverse **strutture dati**, una per ogni *dato* da gestire tra cui:

- una per gli utenti dove vengono specificati username password e stato(online,offline)
- una per le cards usata per memorizzare il nome, la sua descrizione e una lista che contiene la *storia* dei suoi spostamenti
- una per i progetti dove vengono specificati il nome, le card che contiene, i membri e tutte le liste del percorso che una card può seguire

1.2.1 Gestione della concorrenza

Per gestire la concorrenza ho usato delle concurrent hashmap ed ho utilizzato un **Threadpoolexecutor** per avviare un thread client implementato attraverso la classe **Loggedinhandler**.
Nella classe ClientMain ho aperto la connessione TCP e creato il registro per l' RMI.
Per inviare i comandi dal client al server utilizzo un Object output stream.
Una volta che il server riceve la connessione dal client, mette a punto i 2 stream che vengono utilizzati per costruire le informazioni del client attraverso la classe **ClientInfo** necessarie per avviare il thread.
Per gli utenti e per i progetti ho utilizzato delle **concurrent hashmap** in modo da garantire che più thread possano accedervi, essendo questo tipo di hashmap thread-safe. Per aggiungere oggetti nelle hashmap il metodo usato è stato **putIfAbsent** che va ad aggiungere una chiave con il suo valore solo se quella chiave non esiste.

Capitolo 2

Descrizione programma

2.1 Descrizione classi principali

Le classi principali da cui avviare il programma sono

- **ServerMain:** dove vengono create le strutture dati, creato il registro per l'RMI e avviato la connessione TCP a cui vengono passati gli utenti e i progetti.
- **ClientMain:** questa classe consiste nel stabilire la connesione TCP con il server ed entrare in un loop infinito nel quale inviare comandi fino a quando non viene digitato il logout.
- **TCPServer:** Appena avviata la classe ServerMain viene richiamata anche questa classe che si occupa di aprire il socket per la connessione TCP e di creare oppure recuperare informazioni dai file .json.
  entra in un loop infinito dove attende connessioni dai client, successivamente costruisce le informazioni di connessione riferite al client per poi avviare il thread
- **LoggedInHandler:** Tramite le informazioni ottenute dalla classe ***TCPServer*** viene avviato il comando execute del **Threadpoolexecutor**, avviando il thread e, tramite il metodo **start()** gestire effettivamente i comandi inviati dal client fino a quando non viene effettuato il logout.

2.1.1 Altre classi

Le principale altre classi che ho utilizzato sono:

- **User:** Descrive la struttura dati per gli utenti
- **Project:** Descrive la struttura dati per i progetti
- **SignedUpUsers:** Memorizza gli utenti nel file **Users.json**.
- **SignedUpProjects:** Memorizza i progetti nel file **Projects.json**.
- **Card:** descrive la struttura dati per le card contenente le informazioni relative.
- **Multicastgen:** è la classe per generare un indirizzo multicast da associare al progetto per poi utilizzare la chat.

- **Login:** Ha il compito di salvare il risultato del login, aggiunge alla lista degli utenti il nuovo stato di online (nel caso l'operazione vada a buon fine) e aggiunge alla lista del multicast del progetto la segnalazione che lui è membro.
- **ToClient**$< T >$**:** Questa classe l'ho resa generica in quanto l'ho usata per rispondere con diversi tipi di oggetto al client.

Ho usato inoltre 3 **interfacce:**

- **RMI register interface:** Per la registrazioni e i metodi RMI che poi è stata implementata attraverso la classe ***RMI register class***
- **NotifyInterface:** Usata per notificare i cambiamenti di stato da parte di un utente. Contiene il metodo **NotifyEvent**
- **ServerInterface:** Questa interfaccia contiene tutti i metodi tranne la registrazione e viene implementata nel server attraverso la classe ***TCPServer***

2.2 Modalità di esecuzione

Per avviare il programma le 2 classi principali da compilare sono **ServerMain** e **ClientMain**.
Non sono necessari argomenti da passare al programma. Una volta avviato, il server, andrà a creare i file degli utenti, dei progetti, e degli indirizzi multicast, a quel punto si potrà avviare il client.
Per questo motivo è necessario **avviare il server prima di avviare il client**

Capitolo 3

Istruzioni sul programma

Per compilare correttamente il progetto ed eseguirlo è necessario scaricare una libreria esterna, ***GSON*** dal sito
https://search.maven.org/artifact/com.google.code.gson/gson/2.8.6/jar

Una volta avviato il server e il client è possibile iniziare ad utilizzare il programma. **Tutti i comandi devono essere lowercase in quanto il programma è case-sensitive**.
Come prima operazione è necessario registrare un nuovo utente questo è possibile attraverso il comando

register username password

dopo sarà necessario effettuare l'operazione di login digitando

login username password

Dopo sarà possibile cominciare a creare progetti ed usufruire degli altri comandi. Alla fine della sessione è possibile uscire effettuando il logout con la sintassi

logout username

Gli altri comandi disponibili:

- ***listusers:*** per recuperare la lista degli utenti registrati con il loro stato
- ***listonlineusers:*** per recuperare la lista degli utenti registrati che sono online
- ***listprojects:*** per recuperare la lista dei progetti di cui l'utente loggato fa parte
- ***createproject projectName:*** questa comando crea un nuovo progetto avente come membro l'utente che ha richiesto la creazione
- ***addmember projectName Nickutente:*** aggiunge al progetto projectName l'utente Nickutente
- ***showmembers projectName:*** Recupera la lista dei membri del progetto projectName
- ***showcards projectName:*** recupera la lista delle cards associate al progetto projectname

- ***addcard projectName cardName description:*** aggiunge al progetto projectName una nuova card di nome CardName con una breve descrizione **da inserire senza spazi**
- ***movecard projectName cardName beginList endList:*** muove la card cardName nel progetto projectName dalla lista beginList alla lista endList, se soddisfa i vincoli di spostamento elencati qui sotto.

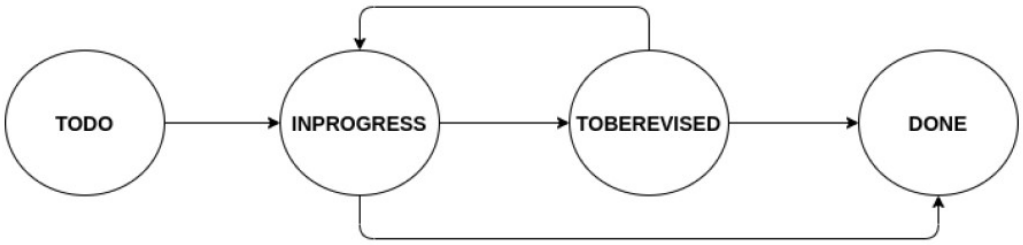

Figura 3.1: Vincoli spostamento cards

- ***getcardhistory projectName cardName:*** recupera la lista degli spostamenti della card cardName all'interno del progetto projectName
- ***readchat projectName:*** recupera i messaggi della chat riferiti al progetto projectName se disponibili
- ***sendchatmsg projectName:*** una volta lanciato il commando si potrà digitare il messaggio da inviare
- ***cancelproject projectName:*** se le card sono tutte finite sarà possibile eliminare il progetto

Bibliografia

[1] *Gson*. URL: https://sites.google.com/site/gson/gson-user-guide#TOC-Overview.

[2] *Metodo Kanban*. URL: https://www.makeitlean.it/blog/il-sistema-kanban-un-esempio.