UNIVERSITÀ DI PISA

Dipartimento di Informatica
Corso di Laurea in Informatica

Relazione Progetto WINSOME

RETI DI CALCOLATORI
E LABORATORIO DI
RETI

Prof.ssa Paganelli Federica
Prof.ssa Laura Emilia Maria Ricci

Carmine Vitiello
578070 - Corso A

Anno Accademico 2021/2022

Introduzione

Il Progetto prevedeva la creazione di due **applicativi** che parlassero, come descrive il paradigma **client-server**. Infatti, attraverso un **server** dotato di una **pool di thread** che soddisfa varie richieste o **task** fatte da più **clients**. Le task rappresentano la metodologia di **interazione** tra **l'utente** finale e le varie **feature** che offre il server. Inoltre, si fruttano le varie metodologie di connessione affrontate a lezione con i loro protocolli associati, cioè **TCP**, **UDP** con il **multicast** e **RMI**. Tutto il codice è stato scritto con **l'IDE IntelliJ**.

Dettagli implementativi

Il **server** attraverso un **selector** gestisce le comunicazioni con i **client**, poi per ogni **richiesta** crea una **task**, che viene assegnata al **pool di thread,** che la esegue e al termine dell'esecuzione il thread **restituisce** una **risposta,** che verrà inviata al client che ha fatto la richiesta. Genericamente il gruppo di thread sarà un **Executors.newCachedThreadPool()**, ma a seconda del numero degli utenti e delle richieste sottomesse al server si potrà limitare ulteriormente il numero dei thread cambiando la tipologia del pool con uno più consono. Ora andremo a trattare alcune **risorse** che sono state create per aumentare l'efficenza e la stabilità del progetto.

1 ListUser

Un oggetto utile e indispensabile per il funzionamento del progetto, dal momento che sarà incaricato di tenere traccia degli **utenti** che **popolano la piattaforma**. Infatti, incapsulerà 4 informazioni:

1. La **listUsers** rappresenta l'insieme degli utenti che fanno uso dell'applicativo con le loro informazioni personali
2. Una **marca temporale** chiamata **timeStamp** per tenere traccia di eventuali cambiamenti e salvataggi. Serve maggiormente per chi dovrà mantenere l'intero software in caso che ci siano bug o problemi.
3. Un booleano **modified** che tiene traccia dell'ultima modifica effettuata alla lista

2 User

L'oggetto che tiene tutte le informazioni di **un singolo utente** ed è formato da:

1. L' **username** un campo **univoco** che viene inserito quando si registra l'utente la prima volta nella piattaforma
2. **L'idUser** è un id **univoco** che può servire in caso che l'username debba essere cambiato
3. La **lista dei tags** che viene popolata durante la registrazione e ogni tags viene inserito con i caratteri minuscoli come chiesto nelle specifiche
4. Una **lista dei followers e dei following**, dove verranno inseriti gli **id** di queste due "tipologie" di utenti in relazione con il proprio utente.
5. Un **seed** cioè una sequenza casuale di caratteri generata durante la registrazione per evitare che utenti con la stessa password abbiano valori hash identici.
6. **L'hash della password concatenata al seed** per aumentare la **sicurezza** della piattaforma da attacchi interni.

3 ListPost

Questo oggetto tiene traccia di tutti gli oggetti **Post** della piattaforma per **centralizzare** le informazioni generate dagli utenti e per avere maggiori possibilità, nel momento che sia necessario fare ricerche filtrate(**query**). Inoltre, durante la creazione di questo oggetto si andrà a **recuperare** le informazioni dai file scritti sul **disco rigido**, oppure in caso che non esistano, si andrà a creare il **file**.

4 Post

Il **Post** viene creato da un utente attraverso il comando **post <Titolo> <Contenuto>**, che verrà salvato nell'oggetto **ListPost**. L'oggetto in se tiene tutte le informazioni che sono descritte nella specifica del progetto.

5 Comment e Vote

Sono due oggetti associati al post per la risoluzione di alcune problematiche relative all'assegnamento di **voti** o **commenti** ai post degli altri utenti. Infatti, i due oggetti ricordano 4 informazioni fondamentali, cioè:

- **L'id** e **l'username** dell'utente che ha messo il voto o il commento
- L'informazioni relativa al **voto**, cioè se è **positivo** o **negativo**, oppure nel caso del **commento** il **contenuto** testuale.
- Un **booleano** per verificare se il commento o il voto fosse **recente**, così da poter eseguire un controllo semplice ed accurato per l'assegnazione del **guadagno** relativo ai vari **portafogli**.

Grazie a queste informazioni è possibile gestire al meglio i contenuti degli utenti evitando problemi di **incongruenza** e **inconsistenza**.

6 Comando

Un oggetto molto semplice per passare al thread la stringa inerente al **comando** inviato dall'utente e **l'id del client** per verificare se l'utente si fosse **loggato**.

7 ListUsersConnessi

Rappresenta una **mappa** dove ad ogni **idClient** corrisponde **un oggetto User** per verificare se l'utente fosse **loggato**, invece in caso che l'utente **non** si sia loggato al posto dell'oggetto ci sarà **null**. Ogni volta che un client si **connette** o si **disconnette** il server modifica la mappa.

8 ListWallet, Transaction e Wallet

Gli oggetti qui presentati, rappresentano le risorse fondamentali per la gestione dei **portafogli** e **l'incremento** dei loro **valori**. Infatti, La **ListWallet** ha lo stesso scopo che hanno le altre **liste**, cioè **centralizzare** le **informazioni** e **sincronizzarle**, invece gli **wallet** rappresentano il **portafoglio** digitale che contiene i **wincoin** e altre informazioni essenziali per il **tracciamento**. Infine l'oggetto **Transaction** rappresenta le **transazioni**, cioè un campo per **l'incremento** del **valore del portafoglio** e una **marca temporale**.

9 Register RMI

La fase di **registrazione** viene effetuata attraverso il componente **RMI**, cioè un oggetto **intermedio** tra il server e il client.
Il componente viene implementato dal server e sarà accessibile dai clients attraverso la sua **interfaccia**.
Quando viene istanziato l'oggetto remoto nel server il costruttore gestisce la **creazione** o la **lettura** del file **json** degli **utenti**, visto che l'oggetto remoto contiene la **ListUser,** così durante la registrazione ho la possibilità di **aggiungere** il nuovo utente all'oggetto ListUser aggiornato.

La fase di registrazione, come tutte le features del server, comprende la creazione di una **task** che viene eseguita dai **thread del RMI**, durante la quale si controlla **l'univocità dell'username**, il **numero** dei **tags** e la **password**. Infine, se tutti i controlli sono andati a buon fine si aggiunge l'utente alla lista mettendo le informazioni in un oggetto **User**.

10 Callback RMI e TaskNotifyUser

La **callback RMI** viene utilizzata durante il **follow** o **l'unfollow** tra due **utenti**, visto che la piattaforma vuole **notificare** l'utente che **perde** oppure **ottiene follower**.
Per soddisfare al meglio questa richiesta è stato ideato un **comando invisibile all'utente** , dove il client si connette e manda un **comando iniziale chiamato "primo"** per ottenere **l'idClient**, visto che da solo non può recuperarlo. Una volta ottenuto l'id, la callback è molto facile, visto che **notificherà solo l'utente** con un determinato **idClient**.

11 TaskEarn e UDP Multicast

La **TaskEarn** viene implementata attraverso un **runnable** e ha il compito di occuparsi del **calcolo del guadagno su tutti i post** e l'aggiunta di eventuali **transazioni**, in caso che ci siano stati nuovi **voti** o **commenti** ai post.
Il multicast viene gestito all'interno della **TaskEarn** da parte del **server** e nella **TaskMultiCast** da parte del **client**. Infatti, il client rimmarà in ascolto attraverso un **thread demone** che eseguirà la **TaskMultiCast**, visto che la ricezione di un messaggio multicast è **bloccante**, infine il server invierà **due messaggi**, cioè uno ad **inizio** di esecuzione dalla TaskEarn e uno a **fine**.

12 TaskSave e Thread associato

La **TaskSave** è una task assegnata ad un **thread speciale**, dal momento che è un thread **demone**, così in caso che ci sia una **terminazione corretta del server** il server morirà, pure se **il thread save sta lavorando**.

Durante la progettazione non è stata pensata una terminazione corretta del server, dal momento che un server non dovrebbe mai terminare, a meno di cause di forza maggiore, però in caso che si voglia approfondire l'argomento sarebbe possibile intercettare alcuni segnali, per poi gestire la terminazione in maniera corretta.
Momentanemente l'aggiornamento delle informazioni su disco vengono assegnate al thread save che dopo una **frazione di tempo** aggiorna i file, se le informazioni sulla memoria **volatile** sono state **modificate**.

13 Task varie

Come detto negli altri paragrafi, per ogni **feature** descritta nelle specifiche viene associata una **task** implementata attraverso un oggetto **Callable**, che una volta soddisfatta, restituirà una **risposta** da mandare al client oppure altre **risorse** a seconda della task.

La struttura **ricorrente** tra tutte le task consiste nel controllo primario, che l'utente si sia **connesso** correttamente attraverso il comando **login** <Username> <Password> e poi a seconda della richiesta si effettueranno dei **controlli specifici** sui vari input mandati dall'utente.

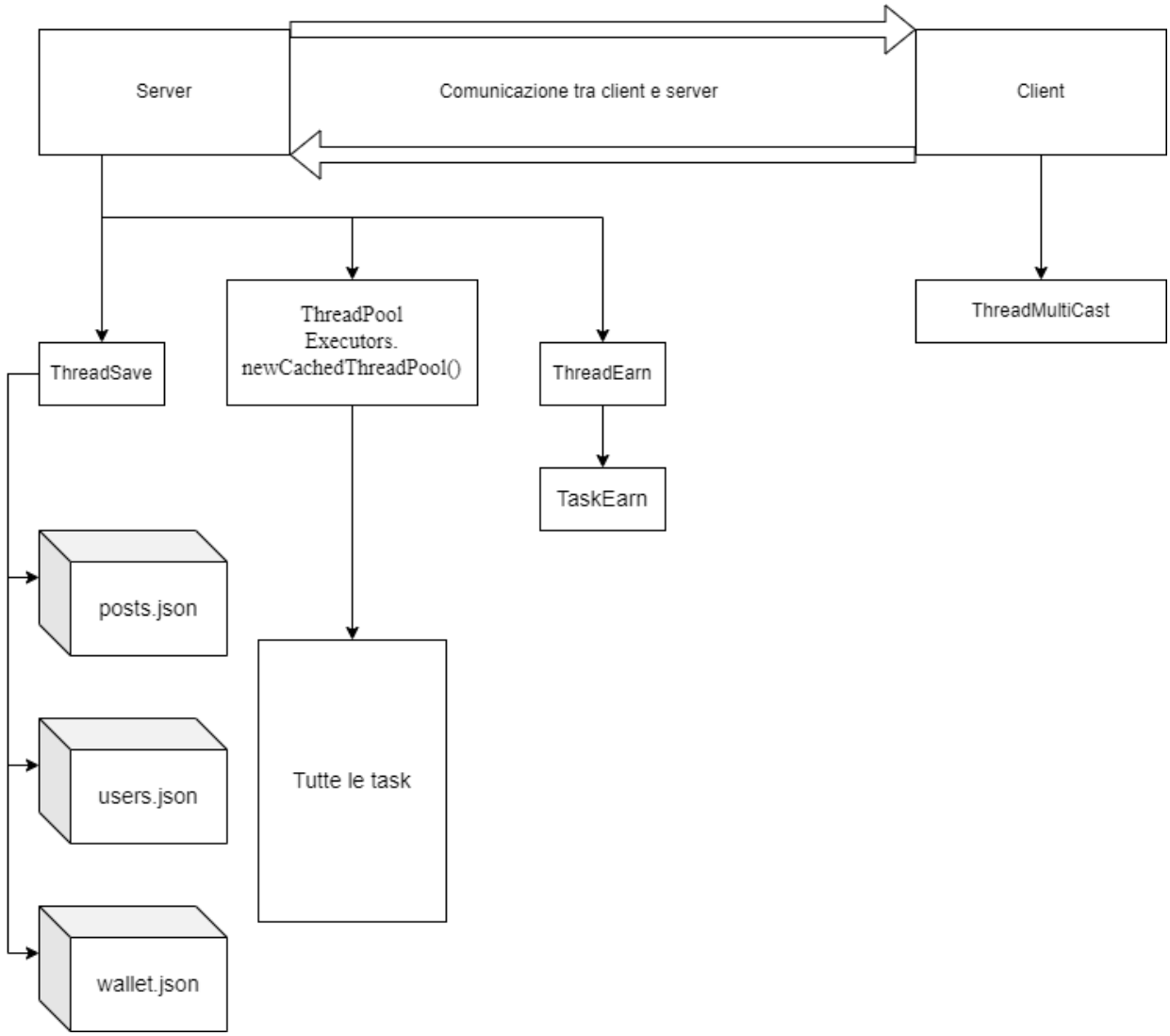
14 UtilFile

L'UtilFile è un oggetto **statico** con alcune funzioni per la **lettura del file di configurazione**, dal momento che attraverso la lettura del file denominato "*serverConfig.txt*" è necessario **filtrare** tutte le righe che contengono il **simbolo #**, perchè vanno ad indicare dei **commenti** e non sono delle informazioni indispensabili. Inoltre, le funzioni assegnano i **valori** letti ai campi **dell'oggetto ConfigField** che contiene tutte le informazioni **basilari** per una comunicazione pulita tra client e server. Infatti, viene **istanziato** all'inizio dell'esecuzione del **server** e del **client** per una corretta esecuzione.

15 Generators

Un oggetto **statico** che contiene alcune funzioni di **supporto** tra cui la funzione **hash** per il salvataggio della password e il suo calcolo, poi la **generazione** di sequenza di caratteri **casuali** per la creazione del **seed** durante la registrazione, ed infine la richiesta di un **numero casuale** dal sito RANDOM.ORG

16 Schema dei Thread attivi e delle strutture dati



Organizzazione del Progetto e GitHub

Il Progetto ha una cartella sorgente (src) per dividere la parte di codice da quella testuale, poi all'interno della src si trovano varie cartelle:

- src/main/java/*:contiene tutti gli oggetti inerenti al progetto
- src/main/java/config: contiene l'oggetto **ConfigField** e il file di configurazione **serverConfig.txt**

Nella cartella **documentation** c'è:

- un file con alcune note che sono state utili per la creazione delle varie risorse presenti nel progetto
- il testo che descrive le specifiche del progetto
- a meno che si cambi nel file config, si andranno a creare le varie strutture dati necessarie

Per avviare correttamente gli applicativi si carica il progetto su **l'IDE IntelliJ** e avviando prima il server e poi il client si può testare tutte le funzionalità.
Durante lo sviluppo del Progetto ho utilizzato una repository pubblica su **GitHub** con il seguente link:
https://github.com/itakartor/ProgettoRetiUnipi-2122.git