



UNIVERSITÀ DI PISA

Corso di Laboratorio di Reti

2021/2022

Relazione progetto:

WINSOME: a reWardINg SOcial Media

Alessandra De Lucrezia

Matricola 565700

Corso A

WINSOME: a reWardINg SOcial Media

Alessandra De Lucrezia 565700

1 Introduzione

Il progetto consiste nella implementazione di WINSOME, un social media che si ispira a STEEMIT (<https://steemit.com/>), una piattaforma social basata su blockchain la cui caratteristica più innovativa è quella di offrire una ricompensa agli utenti che pubblicano contenuti interessanti e a coloro (i curatori) che votano/commentano tali contenuti. La ricompensa viene data in Steem, una criptovaluta che può essere scambiata, tramite un Exchanger, con altre criptovalute come Bitcoin o con fiat currency. La gestione delle ricompense è effettuata mediante la blockchain Steem, su cui vengono registrati sia le ricompense che tutti i contenuti pubblicati dagli utenti. WINSOME, a differenza di STEEMIT, utilizza un'architettura client server in cui tutti i contenuti e le ricompense sono gestiti da un unico server piuttosto che da una blockchain, inoltre il meccanismo del calcolo delle ricompense è notevolmente semplificato rispetto a quello di STEEMIT. Ad esempio, in WINSOME non viene considerata la possibilità di distribuire parte della ricompensa in VEST, ovvero in quote di possesso della piattaforma.

2 Struttura

2.1 Directory

La directory principale è divisa nelle seguenti sotto-directory:

- *lib*: in cui è contenuta la libreria *gson-2.8.9.jar*
- *src*: in cui sono contenuti i *file.java* ed un'ulteriore directory:
 - *file* : al cui interno troviamo i seguenti file (il cui utilizzo verrà spiegato più avanti)
 - * *SpecificheServer.txt*
 - * *SpecificheClient.txt*
 - * *RegistroUtenti.json*
 - * *RegistroPost.json*
 - * *RegistroFollowers.json*

2.2 Componenti principali

2.2.1 Server

Il Server è la prima compentene che deve essere eseguita (per vedere i dettagli di compilazione ed esecuzione leggere capitolo *Compilazione ed esecuzione*). Al suo avvio legge il file di configurazione *SpecificheServer.txt*, contenuto nella directory *file*, da cui ricava la porta TCP, la porta UDP, indirizzo multicast, porta RMI, porta RMI Callback, timeout del calcolo della ricompensa e la percentuale di ricompensa autore. Successivamente effettua il ripristino dei dati dai file di backup:

- *RegistroUtenti.json*: per recuperare gli utenti registrati a Winsome con i loro valori
- *RegistroPost.json*: per ripristinare tutti i post presenti su Winsome
- *RegistroFollowers.json*: per ricavare per ogni utente la lista dei suoi followers.

Inizialmente, nel caso in cui i file di backup siano vuoti, oppure non esistano, il sistema è vuoto.

Una volta finita la lettura dei file, il Server è in grado di stabilire una connessione TCP ed UDP e mettere a disposizione del Client dei metodi remoti tramite RMI. Questi metodi vengono usati al *login* di un client per potersi registrare al meccanismo delle Callback. Similmente, al *logout*, l'utente l'utilizzerà l'altro metodo messo a disposizione per cancellare la sua iscrizione. Una volta stabilita la connessione sulla porta TCP il server rimane in attesa di connessioni da parte dei client utilizzando un'istanza di *GestoreTask*. Il server al *logout* di ogni utente, oppure quando un client chiude la connessione, rende persistenti le informazioni (utenti, post e followers) sui *file* di backup, gli stessi che userà al riavvio per ripristinare lo stato del sistema.

Diverso è invece il meccanismo del gruppo Multicast che il server genera sulla porta UDP, in cui viene calcolato periodicamente il guadagno per ogni post, ed assegnato agli utenti, i quali, se iscritti, riceveranno una notifica dell'avvenuto calcolo e potranno decidere se visualizzare il loro *portafoglio* con il comando *wallet*, oppure convertirne il valore con il tasso di cambio attuale con il comando *wallet btc*. Il Server comunica l'indirizzo del multicast in seguito alla richiesta del client tramite il comando *partecipa*, da quel momento in poi, fino al *logout*, l'utente è iscritto al gruppo di multicast e riceverà le notifiche ogni qual volta avviene il calcolo.

2.2.2 Client

Il client, similmente a quanto fa il Server, legge il file di configurazione da cui ottiene i valori della porta TCP, della porta del *registry* RMI e della porta RMI per le Callback.

Il client deve essere "lanciato" dopo il server, e dopo essersi correttamente connesso visualizza il messaggio di benvenuto al sistema Winsome. Un client può effettuare la registrazione a Winsome mediante l'utilizzo dell'interfaccia *RMIRegistrazione* condivisa con il server, mentre l'iscrizione al sistema di notifica viene eseguita mediante l'utilizzo di due interfacce: *RMICallbackServer_Interface* e *RMICallbackClient_Interface*. Con il *login* un utente si iscrive al sistema di notifica e può effettuare una serie di operazioni che sono consultabili digitando il comando *help*. Tra tutte le operazioni le uniche gestite localmente dal client sono *listFollowers*, che stampa la lista dei followers dell'utente e *mostra-ListaOperazioni*, che stampa la lista di tutte le operazioni possibili e i comandi associati. Il server si occupa di tutte le altre, ad eccezione del metodo della registrazione, gestito con un metodo remoto.

3 Java Concurrency

3.1 Concurrent Collections

Nello sviluppo del progetto sono state scelte strutture dati concorrenti per garantire una corretta concorrenza ed avere una esecuzione *thread-safe*. Nonostante il loro utilizzo comporti un "rallentamento" dell'esecuzione sono state usate al fine garantire la concorrenza sulle strutture dati che potevano essere modificate da più client in contemporanea.

Le strutture dati concorrenti di maggiore importanza sono:

```
ConcurrentHashMap<String , Utente> utenti;  
ConcurrentHashMap<String , ConcurrentLinkedList<String>> followers;  
ConcurrentHashMap<Integer , Post> posts;
```

3.1.1 Concurrent HashMap *utenti*

L'HashMap concorrente *utenti* ha come chiave il nome dell'utente e come valore l'oggetto Utente. Nelle istanze della classe Utente vengono memorizzati nome, password e tags, che non vengono mai più modificate una volta create, e strutture dati come:

- *ConcurrentLinkedList listaPost*: contiene i post di cui l'utente è autore oppure i post di cui ha fatto il *rewind*;
- *Vector listaTransazioni*: un vettore di String contenente la lista di tutte le transazioni eseguite per il calcolo del guadagno con il rispettivo incremento; altre variabili di tipo *double* usate per il calcolo del guadagno.

3.1.2 Concurrent HashMap *posts*

L'HashMap concorrente *posts* ha come chiave un Integer che rappresenta l'id univoco del post e come valore l'oggetto Post. Nelle istanze della classe Post vengono memorizzati l'id, il titolo, il contenuto, il numero di voti positivi e negativi ed il numero di valutazione. Inoltre sono contenute strutture dati come:

- *ConcurrentHashMap tabellaCommentiUtenti*: con chiave String, rappresentante il nome dell'utente e come valore un AtomicInteger, identificativo del numero dei commenti lasciati da quell'utente; *ConcurrentHashMap listaCommenti* con chiave String, rappresentante il nome dell'utente, e come valore un Vector di stringhe, contenente tutti i commenti lasciati dall'utente;
- *Vector listaUtentiCommenti*: vettore di stringhe contenente la lista degli utenti che hanno commentato il post;
- *Vector utentiVoti*: vettore di stringhe contenente la lista degli utenti che hanno votato;
- *Vector listaLike* vettore di Integer contenente la lista dei voti (+1 oppure -1).

3.1.3 Concurrent HashMap *followers*

Infine l'HashMap *followers* con chiave *String*, rappresentante il nome dell'utente e come valore una *ConcurrentLinkedListQueue* contenente i suoi followers.

3.2 Strutture thread-safe

Oltre le strutture dati concorrenti, nel progetto è stata usata anche una struttura thread-safe: Vector. Essendo un contenitore elastico, estensibile ed accorciabile, ma non generico garantisce la thread-safeness.

3.3 Variabili Atomiche

Infine è stato utilizzato il package *java.util.concurrent.atomic.AtomicInteger* che mette a disposizione metodi atomici per incrementare e decrementare variabili. Nello specifico è stato usato AtomicInteger per il calcolo dell'id del post, il numero di voti positivi e negativi e il numero di commenti lasciati da ogni utente.

4 Controlli sui valori

4.1 Controllo valori parsati da input

Il Client per mandare e ricevere messaggi usa la CLI. Ogni volta che il client manda un messaggio tramite CLI questo viene analizzato e controllato. Una generica richiesta da parte del client è formata nel seguente modo: *j operazione parametro/i*. Inizialmente il controllo viene fatto sulla prima stringa parsata, in questo caso *operazione*, in caso di messaggio di errore l'utente può riprovare a scrivere correttamente l'operazione oppure digitare *help* per vedere tutte le operazioni accettate. Una volta superato il primo controllo si passa al controllo dei parametri.

Tra i controlli più frequenti ci sono:

- che la stringa sia diversa da null e/o diversa dallo spazio bianco;
- che i valori numerici inseriti siano degli interi;
- che la lunghezza delle stringhe sia quella attesa.

Superati questi si passa agli altri controlli:

- proibire la registrazione di utenti con lo stesso nome oppure con una lista tags troppo lunga o anche con valori vuoti o null;

- nei casi *rewin/comment/vote/create/delete* post si effettua il controllo sull'esistenza del post prima di effettuare l'azione specifica.

Nel caso di *delete* si verifica che l'utente che vuole eliminare il post sia l'autore stesso.

Nel caso di *vote* viene controllato che ogni utente voti al più una volta tenendo in considerazione che l'autore non può commentare i suoi post e/o votarli);

- al login viene controllato se l'utente è registrato e se sta accedendo con la password corretta.

5 Descrizioni delle classi

5.1 Interfacce

- *RMICallbackServer*: modella l'interfaccia dell'oggetto RMI che verrà utilizzato per le *CallBack*. Contiene le signature dei metodi per la registrazione/rimozione al servizio di notifica, *registrazioneCallback* e *rimuoviRegistrazioneCallback*.
- *RMIRregistrazione*: modella l'interfaccia dell'oggetto RMI utilizzato per le registrazioni degli utenti. Contiene la signature del metodo *register*.
- *RMICallbackClient*: modella l'interfaccia che contiene l'intestazione dei metodi *notificaFollow*, *notificaUnfollow* e *updateFollowers* che vengono utilizzati dal client per ricevere la versione più recente della struttura dati *followers*.

5.2 Classi

- *WinsomeClientMain*: classe principale del client che deve essere lanciata ogni qual volta si voglia connettere un nuovo client a Winsome, come già detto in precedenza questa classe contiene l'implementazione dei metodi locali e quelli che fanno uso RMI, come per la registrazione;
- *RMICallbackClient*: implementa i metodi dell'interfaccia *RMICallbackClient*;
- *TaskClientMulticast*: implementa *Runnable* e viene usata per gestire la ricezione dei messaggi dal gruppo Multicast;
- *WinsomeServerMain*: classe principale del server che contiene i metodi per il salvataggio e ripristino dei dati, per la generazione dei metodi remoti e per l'esecuzione di due ThreadPools, uno per gestire la connessione TCP e l'altro per gestire il gruppo Multicast;
- *RMIRregistrazione*: implementa il metodo dell'interfaccia *RMIRregistrazione*;
- *RMICallbackServer*: contiene l'implementazione dei metodi remoti definiti dall'interfaccia *RMICallbackServer* che vengono usati per notificare l'aggiornamento dei followers e il metodo che si occupa di comunicare all'utente appena connesso la struttura dati *followers* aggiornata;
- *GestoreTask*: classe che si occupa di instaurare una connessione TCP sulla porta nota. Quando viene accettata una connessione il threadpool va a gestire un task, implementato nella classe *Task*, che si occupa dell'interazione tra un utente e il sistema.
- *Task*: modella l'handler di gestione delle richieste. Si compone di un costrutto switch che in base al comando digitato sulla CLI va ad invocare il metodo opportuno che restituisce all'utente la risposta al comando da lui richiesto. In questa classe sono implementati tutti i metodi e i rispettivi controlli per gestire le operazioni richieste dagli utenti, ad eccezione di *listFollowers* che viene gestita localmente dal client;
- *TaskServerMulticast*: implementa *Runnable* e gestisce la connessione UDP del multicast con il rispettivo invio delle notifiche, il calcolo periodico delle ricompense e l'implementazione del metodo per calcolare il guadagno;
- *Utente*: modella un utente all'interno del sistema con tutte le caratteristiche richieste dalle specifiche;

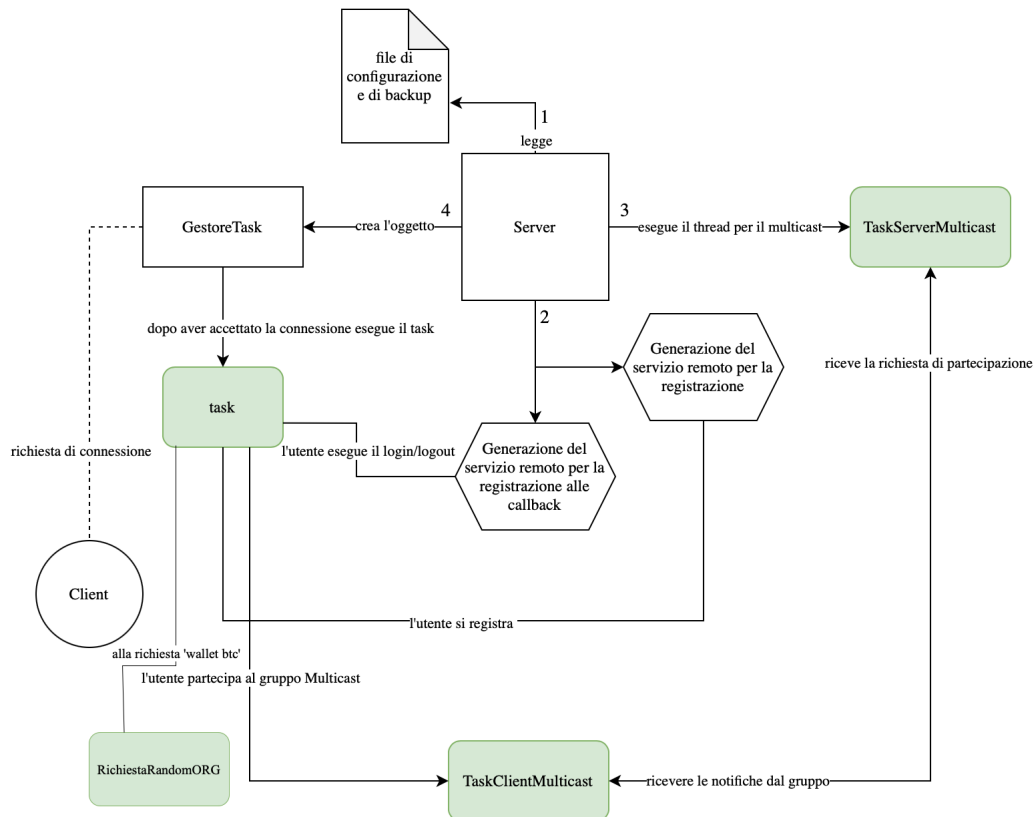
- Post: modella un post all'interno del sistema con tutte le caratteristiche richieste dalle specifiche;
- RichiestaRandomORG: classe che implementa il task del thread che si occupa di mandare una richiesta al url random.org per prendere il valore della valuta del cambio per la conversione in wincoin.

6 Thread attivati

In figura sono mostrati in verde chiaro quali sono i thread che vengono attivati. I numeri "vicino" a Server indicano in quale ordine il Server esegue le azioni.

Nello specifico i thread attivati sono:

- TaskServerMulticast: thread del Multicast lato server, gestito dal un FixedThreadPool, che si occupa di calcolare il guadagno e di mandare la notifica agli utenti iscritti al canale;
- task (per ogni client): thread che viene attivato per ogni client che si collega al servizio WINSO-ME gestito dal un CachedThreadPool;
- TaskClientMulticast (per ogni client che si iscrive al multicast): thread che rimane in attesa di notifiche dal gruppo multicast fino al timeout (settato a circa 7 minuti);
- RichiestaRandomORG: thread che viene creato ogni qual volta un utente vuole convertire il proprio portafoglio in wincoin, il thread si collega al servizio Random.ORG e con una "GET" prende un double che rappresenta la valuta del cambio.



7 RMI

Nel progetto sono presenti due utilizzi di RMI, uno per la registrazione a Winsome e uno per la registrazione alle callback.

7.1 RMI per la registrazione a Winsome

La registrazione a Winsome di un utente è gestita tramite RMI. Questo metodo remoto viene generato dal server usando l'interfaccia *RMIRegistrazioneInterface* e la classe *RMIRegistrazione*. Per la sua generazione è stato usato il meccanismo visto a lezione:

- Viene creato l'oggetto da esportare;
- Viene creato un registry su una porta nota, ovvero RMI PORT che il server acquisisce dal file di configurazione;
- Viene recuperato il registry appena creato;
- Viene pubblicato il riferimento con un nome proprio, nello specifico *registrationServiceName* in modo tale da poter essere reperito dal client per effettuare l'invocazione del metodo e quindi effettuare la registrazione.

7.2 RMI per la callback

Per tenere aggiornata l'HashMap *followers* nel client vengono usate le callback. Quando un utente esegue il login su Winsome si iscrive alle callback per le notifiche, con cui viene avvertito quando un utente inizia o smette di seguirlo. Il Server pubblica tramite il metodo *generazioneCallback*, implementato nella classe *WinsomeServerMain*, mette a disposizione del client i metodi per permettergli di iscriversi e cancellarsi al gruppo di notifiche.

Quando il server chiama *notificaFollowers/notificaUnfollow*, dichiarati nell'interfaccia *RMICallBackServer_Interface*, quando un utente vuole seguire o smettere di seguirne un altro, va a comunicare al client la struttura dati *followers* appena aggiornata, in questo modo quando il client chiamerà il metodo *listFollowers* non dovrà mandare alcuna richiesta al server, ma potrà gestirla localmente senza doversi preoccupare di avere la struttura dati non aggiornata. Sempre nella stessa interfaccia è dichiarato anche un terzo metodo *notificaUpdateFollowers*, che viene chiamato dal server quando un utente esegue il login per comunicarli la struttura dati *followers* aggiornata al momento del suo login.

8 Tipi di connessione

Così come richiesto dalle specifiche i tipi di connessioni implementati in questo progetto sono TCP e UDP. Dopo aver effettuato il login con successo, l'utente interagisce, secondo il modello client-server (richieste/risposte), con il server sulla connessione TCP creata con i comandi che sono consultabili digitando *help* sulla CLI. Tutte le operazioni sono effettuate su questa connessione TCP, eccetto la registrazione che usa RMI, le operazioni di recupero della lista dei followers, corrispondente al comando *list followers*, che usa la struttura dati locale del client aggiornata tramite il meccanismo di RMI callback e l'operazione che restituisce tutti i comandi possibili corrispondente al comando *help*.

Invece, per gestire il calcolo del guadagno, è stato creato un gruppo Multicast usando la comunicazione UDP, al quale un client può richiedere di partecipare con il comando *partecipa*.

8.1 Comunicazione TCP

Per la gestione di richieste il server è stato implementato usando *Java I/O multithreaded* con un'opportuna gestione della concorrenza garantita grazie alle strutture dati concorrenti.

Per la corretta gestione delle richieste è stato creato un *ThreadPool*. Ogni qual volta che un client si connette viene generato un task, rappresentato dall'istanza della classe *Task* che verrà passato al *ThreadPool*, nel caso specifico un *CachedThreadPool*.

Una volta stabilita la connessione correttamente, il client può registrarsi usando i metodi remoti ed effettuare il login, oppure se l'utente è già registrato, eseguirà direttamente il login. Per ogni comando eseguito dal client gli verrà restituito un messaggio o con la conferma dell'avvenuta esecuzione oppure con un messaggio di errore dove aver eseguito tutti i controlli citati nel paragrafo 4.

8.2 Comunicazione UDP

Anche l'implementazione del gruppo Multicast viene usato *Java I/O multithreaded* in questo caso però non viene usato un *CachedThreadPool* ma un *FixedThreadPool*, questo perchè il gruppo multicast è uno solo e non uno diverso per ogni client. Una volta stabilita la connessione UDP il thread incaricato si occupa di calcolare il guadagno delle ricompense, che verrà illustrato più avanti, eseguendolo ogni 4 minuti (circa), questo timeout serve per simulare il passare del tempo all'interno di Winsome. Una volta eseguito il calcolo lo comunica a tutti gli utenti iscritti inviando una notifica. L'utente quando la riceve può decidere di visualizzare il suo portafoglio appena aggiornato con digitando *wallet* oppure convertire il suo valore con la valuta di cambio attuale, digitando *wallet btc*. Il server comunica al client l'indirizzo e la porta multicast quando riceve come operazione *partecipa*, da quel momento fino al logout, l'utente è abilitato a ricevere le notifiche.

9 Calcolo guadagno

Come già detto in precedenza il calcolo del guadagno viene effettuato dal gruppo multicast seguendo la seguente formula:

$$\text{Guadagno} = \log(\max(\sum_{p=0}^{\text{newPeopleLikes}}(Lp), 0) + 1) + \log(\sum_{p=0}^{\text{newPeopleCommenting}}(\frac{2}{1+e^{(-Cp-1)}}) + 1)$$

Successivamente si divide il guadagno per il numero di iterazione del post, in questo modo i post più "vecchi" influiranno di meno rispetto ai post più recenti.

$$\frac{\text{Guadagno}}{n\text{Iterazioni}} \quad (1)$$

Il metodo con cui viene effettuato il calcolo è implementato nella classe *TaskServerMulticast*, ed è chiamato nel *run* del thread che si occupa del multicast. Sempre localmente ne è implementato un altro che calcola la lista dei 'curatori' del post, ovvero tutti quegli utenti che hanno commentato e/o votato positivamente il post. L'autore del post e i suoi curatori ricevono un compenso diverso, l'autore prende la percentualeAutore sul guadagno complessivo, questo valore gli viene passata dal Server che ha ricevuto tramite il file di configurazione, mentre i curatori ricevono la restante percentuale divisa tra tutti i curatori. Dopo aver calcolato il guadagno per tutti gli utenti viene salvata la transazione del calcolo nel portafoglio di ciascun utente contenente la data e il valore dell'incremento.

Un utente può richiedere anche di avere il valore del suo portafoglio convertito secondo la valuta attuale. Per prendere la valuta di cambio viene usato un thread che si collega al servizio [RANDOM.ORG](https://api.random.org). Quando un utente richiede la conversione viene avviato il thread che restituirà il valore della valuta.

Il calcolo eseguito verrà poi reso persistente sui file *.json*.

10 Persistenza dei dati

Come già detto in precedenza per rendere persistenti i dati, nello specifico gli utenti, i post e i followers, viene usato il salvataggio sui file *.json*. Questi file vengono usati sia per il backup che per ripristino dei dati. L'operazione di serializzazione e deserializzazione è stata realizzata mediante l'utilizzo della libreria esterna *gson-2.8.9.jar*, contenuta nella directory lib. Con l'utilizzo di questa libreria si possono convertire gli oggetti da Java a Json e viceversa. Il procedimento usato è quello visto a lezione e viene applicato nei metodi implementanti nella classe *WinsomeServerMain*, di cui distinguiamo i metodi per salvare sul file e quelli per recuperare dal file. Quelli per salvare sono: *salvaUtente*, *salvaPosts* e *salvaFollowers*.

Mentre i metodi per recuperare sono: *ripristaUtente*, *ripristinaPost*, *ripristinaFollowers* e *ripristinaValori*.

11 Compilazione ed esecuzione

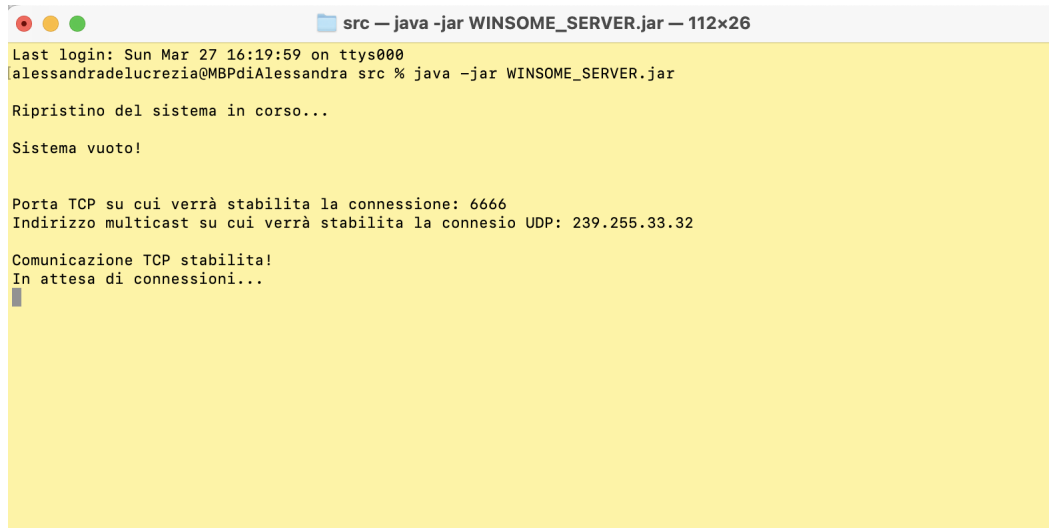
11.1 Compilazione ed esecuzione con eseguibili

L'intero progetto è stato sviluppato interamente su IntelliJ IDEA e testato su ambiente macOS Monterey.

All'interno della directory *src* sono salvati due file eseguibili, uno per il server e uno per il client.

11.1.1 Esecuzione del Server

Come già detto in precedenza il primo ad essere lanciato deve essere il server. Aprire il terminale nella directory *src* ed eseguire `java -jar WINSOME_SERVER.jar`:



```
src — java -jar WINSOME_SERVER.jar — 112x26
Last login: Sun Mar 27 16:19:59 on ttys000
alessandradelucrezia@MBPdIAlessandra src % java -jar WINSOME_SERVER.jar

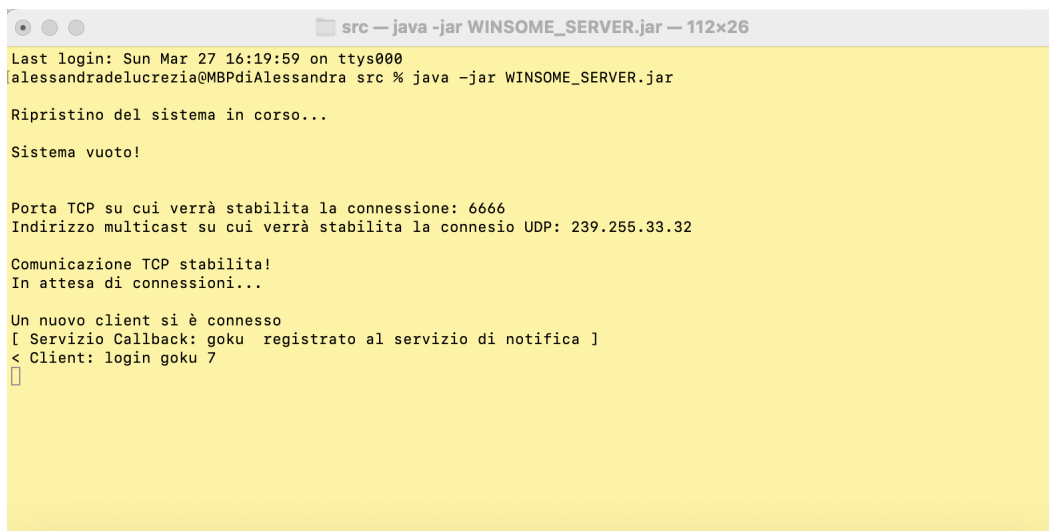
Ripristino del sistema in corso...

Sistema vuoto!

Porta TCP su cui verrà stabilita la connessione: 6666
Indirizzo multicast su cui verrà stabilita la connesio UDP: 239.255.33.32

Comunicazione TCP stabilita!
In attesa di connessioni...
█
```

In questo caso i file di backup (*RegistroUtenti.json* , *RegistroPost.json* , *RegistroFollowers.json*) sono tutti vuoti, quindi il server inizialmente è vuoto, da questo momento in poi il server rimane in attesa di richieste di connessioni da parte del client (comunicazione TCP), ed avvia anche il multicast (comunicazione UDP).



```
src — java -jar WINSOME_SERVER.jar — 112x26
Last login: Sun Mar 27 16:19:59 on ttys000
alessandradelucrezia@MBPdIAlessandra src % java -jar WINSOME_SERVER.jar

Ripristino del sistema in corso...

Sistema vuoto!

Porta TCP su cui verrà stabilita la connessione: 6666
Indirizzo multicast su cui verrà stabilita la connesio UDP: 239.255.33.32

Comunicazione TCP stabilita!
In attesa di connessioni...

Un nuovo client si è connesso
[ Servizio Callback: goku registrato al servizio di notifica ]
< Client: login goku 7
█
```

Quando un client si connette sulla comunicazione TCP il server lo "comunica" a video e registra l'utente che ha fatto il login al servizio di notifica gestito dalle callback e gli comunica anche la struttura dati *followers* aggiornata.

```
src — java -jar WINSOME_SERVER.jar — 112x26
Last login: Sun Mar 27 16:19:59 on ttys000
alessandradelucrezia@MBPdIAlessandra src % java -jar WINSOME_SERVER.jar

Ripristino del sistema in corso...

Sistema vuoto!

Porta TCP su cui verrà stabilita la connessione: 6666
Indirizzo multicast su cui verrà stabilita la connessione UDP: 239.255.33.32

Comunicazione TCP stabilita!
In attesa di connessioni...

Un nuovo client si è connesso
[ Servizio Callback: goku registrato al servizio di notifica ]
< Client: login goku 7
< Client: post "prova" "primo post"
< Client: partecipa
█
```

Da ora fin quando l'utente non esegue il logout e poi successivamente interrompe la comunicazione con il comando *control + c* il server stampa l'operazione richiesta. Con il comando *partecipa* il client si iscrive al gruppo multicast, quindi riceverà una notifica ogni qual volta il task del multicast eseguirà il calcolo del guadagno, circa ogni 4 minuti.

```
src — java -jar WINSOME_SERVER.jar — 112x26
Last login: Sun Mar 27 16:19:59 on ttys000
alessandradelucrezia@MBPdIAlessandra src % java -jar WINSOME_SERVER.jar

Ripristino del sistema in corso...

Sistema vuoto!

Porta TCP su cui verrà stabilita la connessione: 6666
Indirizzo multicast su cui verrà stabilita la connessione UDP: 239.255.33.32

Comunicazione TCP stabilita!
In attesa di connessioni...

Un nuovo client si è connesso
[ Servizio Callback: goku registrato al servizio di notifica ]
< Client: login goku 7
< Client: post "prova" "primo post"
< Client: partecipa
[ Multicast: messaggio mandato agli iscritti al gruppo multicast! ]
█
```

Al logout dell'utente il Server si occupa di rimuoverlo sia dalla lista delle callback e sia dal gruppo multicast e salva in maniera persistente le informazioni dell'utente sui file di backup.

```
src — java -jar WINSOME_SERVER.jar — 112x26
Last login: Sun Mar 27 16:19:59 on ttys000
alessandradelucrezia@MBPdIAlessandra src % java -jar WINSOME_SERVER.jar

Ripristino del sistema in corso...

Sistema vuoto!

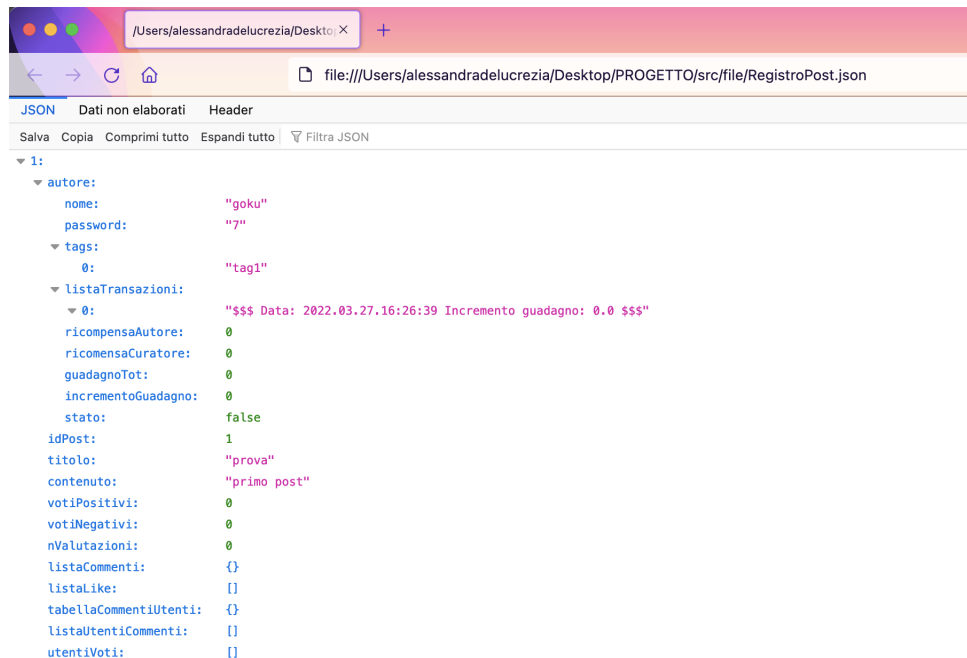
Porta TCP su cui verrà stabilita la connessione: 6666
Indirizzo multicast su cui verrà stabilita la connessione UDP: 239.255.33.32

Comunicazione TCP stabilita!
In attesa di connessioni...

Un nuovo client si è connesso
[ Servizio Callback: goku registrato al servizio di notifica ]
< Client: login goku 7
< Client: post "prova" "primo post"
< Client: partecipa
[ Multicast: messaggio mandato agli iscritti al gruppo multicast! ]
< Client: logout goku
[ Servizio Callback: goku rimosso dal servizio di notifica! ]
█
```

11.1.2 Esempio file di salvataggio

Qui di seguito è mostrato un esempio di come appare il file *RegistroPost.json* dopo il salvaggio fatto dal server, similmente accade lo stesso per gli altri due file. Per avere questa visualizzazione consiglio l'apertura dei file *.json* con il motore di ricerca Firefox.



11.1.3 Esecuzione del Client

Ora vediamo invece cosa accade lato client. Aprire il terminale nella directory src ed eseguire `java -jar WINSOME_CLIENT.jar`

Di seguito è riportato il terminale del client in relazione all'esecuzione del server vista nel paragrafo precedente.

```
src — java -jar WINSOME_CLIENT.jar — 117x49
Last login: Sat Mar 26 09:57:39 on ttys000
[alessandradelucrezia@MBPdIAlessandra src] % java -jar WINSOME_CLIENT.jar
< Benvenuta/o su Winsome!
Winsome è una piattaforma social basata su blockchain in cui vengono premiate le idee!

Per registrarti digita: register <username> <password> <tag_1,...,tag_N>
dove al posto di 'tag_1,...,tag_N' inserirai la lista dei tuoi interessi
Sei già registrato? Per effettuare il login digita: login <username> <password>

Hai bisogno di aiuto? Digita help

Per chiudere la connessione digita logout <username>
> register goku 7 tag1
> goku iscritto a Winsome!
Benvenuto su Winsome,
per iniziare a interagire con gli altri utenti digita <list users> per visualizzare gli utenti con cui hai interessi
in comune!
Quando inizierai a seguire un utente potrai vedere nel tuo feed tutti i suoi post, potrai commentarli, votarli e fare
il rewin!
Buon divertimento su Winsome!

> login goku 7
< goku sei online!

> post "Prova" "primo post"
< Post creato:
< idPost: 1
< Titolo: Prova
< Contenuto: primo post

> list users
< Lista degli utenti con interessi in comune: Non ci sono utenti registrati con i tuoi stessi interessi!

> list followers
< Lista dei tuoi followers: []

> partecipa
< 239.255.33.32 44444 indirizzo e porta multicast a cui verrai iscritto!

> [ Notifica gruppo Multicast: calcolo guadagno avvenuto, digita <wallet> o <wallet btc> per visualizzare il tuo port
afoglio! ]
> [ Notifica gruppo Multicast: calcolo guadagno avvenuto, digita <wallet> o <wallet btc> per visualizzare il tuo port
afoglio! ]
> logout goku
< goku ti sei disconnesso!
< Torna presto su Winsome per guadagnare con le idee!
```

11.2 Compilazione ed esecuzione da linea di comando

Per compilare ed eseguire il progetto senza gli eseguibili si possono usare i seguenti comandi dopo essersi posizionati sulla directory scr:

Per il Server:

- per compilare: `javac -cp ../lib/gson-2.8.9.jar *.java WinsomeServerMain.java`
- per eseguire: `java -cp ../lib/gson-2.8.9.jar WinsomeServerMain.java`

Per ogni Client:

- per compilare: `javac -cp ../lib/gson-2.8.9.jar *.java WinsomeClientMain.java`
- per eseguire: `java -cp ../lib/gson-2.8.9.jar WinsomeClientMain.java`

12 Note

Quando verrà eseguito il progetto consegnato, come si potrà notare dal terminale all'esecuzione del file .jar del server, risultano già degli utenti registrati a Winsome, di cui è già avvenuto il calcolo del guadagno con l'assegnamento delle ricompense. Nel caso in cui si voglia avviare il sistema 'vuoto' basterà eliminare tutti i file .json, i quali verranno ricreati autonomamente dal server al suo avvio. Per garantire una corretta esecuzione del programma si assume che prima di interrompere la comunicazione di un client l'utente esegua il logout.