Relazione Progetto Laboratorio di Reti dei Calcolatori

“WINSOME: a reWardINg SOcial MEdia”

Santoro Salvatore – 530410

A.A. 2021/2022

UNIVERSITÀ DI PISA

INDICE

1. Introduzione

Lo scopo di questo progetto è la realizzazione di una struttura client-server che implementa una rete sociale orientata ai contenuti. Ogni utente registrato può seguire altri utenti ed essere a sua volta seguito. Ogni utente in fase di registrazione dovrà fornire dei tag (massimo 5) caratterizzanti il proprio profilo. Inoltre ogni utente può pubblicare, commentare, votare e fare il "rewin" di post.
Ogni utente ha un proprio blog e un proprio feed. Il blog contiene l'insieme dei post di cui l'utente è autore o di cui a fatto il "rewin". Il feed contiene l'insieme dei post degli utenti di cui l'utente è follower.
L'interazione trai i vari post è ciò che permette agli utente di guadagnare "wincoin" (criptovaluta interna). Ogni utente ha un proprio portafoglio in cui sono custodite le ricompense ricevute.
Gli utenti registrati possono eseguire 17 operazioni distinte:

- Login: permette all'utente di accedere con le proprie credenziali ai contenuti del servizio.
- Logout: permette all'utente di terminare la sessione.
- List users: permette all'utente di visualizzare una lista parziale degli utenti registrati al servizio.
- List followers: permette all'utente di visualizzare la lista dei propri follower.
- List following: permette all'utente di visualizzare la lista dei propri follower.
- Follow: permette all'utente di seguire un altro utente registrato con almeno un tag in comune.
- Unfollow: permette all'utente di smettere di seguire uno dei suoi follower.
- Blog: permette all'utente di recuperare la lista dei post presenti nel proprio blog.
- Post: permette all'utente di creare un nuovo post.
- Feed: permette all'utente di recuperare la lista dei post presenti nel proprio feed.
- Show post: permette all'utente di vedere il contenuto di un post presente nel proprio blog o feed.
- Delete: permette all'utente di cancellare uno dei propri post.
- Rewin: permette all'utente di inserire nel proprio blog un post presente nel proprio feed.
- Rate: permette all'utente di valutare positivamente o negativamente un post nel proprio feed.
- Comment: permette all'utente di commentare uno dei post degli utenti di cui è follower.
- Wallet: permette all'utente di recuperare il valore del proprio portafoglio e tutte le transazioni.
- Wallet btc: permette all'utente di recuperare il valore del proprio portafoglio convertito in bitcoin.

La sintassi dei comandi per eseguire le varie operazioni è presente alla sezione 4.2.

2. Overview

Per la modellazione dello storage del sistema, sono stati creati due file json (UsersStorage.txt e PostsStorage.txt) che rappresentano lo spazio di storage e che vengono editati e letti utilizzando metodi di un oggetto di tipo Storage. Il primo file contiene le informazioni riguardanti tutti gli utenti che si registrano alla rete sociale e le relative informazioni su seguaci e seguiti, mentre il secondo file contiene tutti i post presenti nella rete sociale con le relative informazioni sulle interazioni con gli utenti.

Per la fase di registrazione il client usa un metodo remoto fornito dal server tramite il meccanismo della Remote Method Invocation (RMI). A ciascun processo client è permesso registrare un qualsiasi numero di utenti, purché esso rispetti le specifiche di registrazione richieste e non violi l'unicità degli utenti.

Il sistema prevede che, all'inizio della sessione, client e server si connettano tramite una connessione TCP e che su questa avvenga la comunicazione tra le richieste del client e le risposte del server.
Per la comunicazione dei pacchetti su tale protocollo TCP è stata utilizzata la Java IO API (package standard di I/O contenuto in java.io e basato su stream (bloccante)), in particolare sono state utilizzate due implementazioni delle classi astratte Reader e Writer, 'BufferedReader' per la lettura e 'PrintWriter' per la scrittura, entrambi con associato un InputStream ed un OutputStream rispettivamente. Dato che il modello della Java IO API è bloccante, il server utilizza un sistema di gestione dei diversi client mettendo a disposizione un threadpool che riceverà e manderà in esecuzione un nuovo thread ogni qualvolta la socket TCP riceverà una nuova connessione da parte di un client.

La "Login" è la prima operazione necessaria affinché si possano eseguire tutte le altre. Il server, una volta effettuati i controlli sulla correttezza del numero dei parametri della login e fatto il match tra la chiave inserita e quella salvata nel sistema (cifrata utilizzando SHA-256), elabora e compie le operazioni necessarie alla costruzione della risposta o altrimenti crea un messaggio di errore. Se il login è avvenuto con successo, allora per il client è possibile effettuare completamente l'interazione (richieste-risposte) con il server utilizzando tutti i comandi precedentemente elencati.

Inoltre il sistema fornisce il meccanismo delle RMI Callback che permette al server di comunicare ai vari client, previa login avvenuto con successo, quando un utente ha iniziato a seguirlo o a smesso di farlo.

Il sistema usa il meccanismo di UDP Multicast che permette al server di comunicare ai client l'avvenuto aggiornamento del contenuto del portafoglio degli utenti. Ogni client, previa login avvenuto con successo, si registra al servizio di multicast fornito dal server (nello stile del modello publish-subscribe) e lo interrompe solo dopo l'avvenuto logout.

In generale, una richiesta del client è strutturata come segue:

- Nome dell'operazione (comando da eseguire in lower-case);
- Eventuali parametri dell'operazione (separati da spazio).

Ogni riposta del server è strutturata come un unico messaggio di testo contenente tutto il necessario per soddisfare la richiesta fatta dal client (messaggio di errore nel caso di requisiti non soddisfatti).

Per la gestione della **concorrenza** all'interno del sistema, è stata utilizzata una ReentrantReadWriteLock per fare l'accesso e la modifica in maniera atomica a più strutture dati e mantenere la coerenza e la consistenza dello stato del sistema. E' fatto uso della Lock di scrittura per fare accesso da un solo thread per volta a un insieme di strutture dati e la lock di lettura per permettere a tutti i thread lettori di accedere e leggere i dati delle strutture, solo se su queste non è alcun scrittore. Mentre per il client è stato necessario l'utilizzo di una ReentrantLock necessaria per la scrittura su stdout in maniera concorrente per evitare di spezzare i messaggi da stampare in caso di arrivo di comunicazioni delle RMI Callback e del servizio di multicast.

Inoltre sono state utilizzate delle strutture dati concorrenti all'interno della classe ServerMain, come CopyOnWriteArrayList e ConcurrentHashMap per quando viene fatto l'accesso alle singole strutture dati e non è necessario modificarne più di una in maniera atomica per mantenere lo stato del sistema.

3. Implementazione

3.1 Server

Il server è stato sviluppato mediante un'architettura *multithread* in cui ad ogni client viene associato un thread (istanza della classe ClientHandler) che si occupa delle richieste. Per la memorizzazione dei client, è stata definita una struttura dati (mappa di nome clientsConnected) contenente la socket (associata al client) e una stringa contenente lo username, se il login è stato effettuato, altrimenti la stringa vuota.
Per la completa implementazione dell'intero server è stato necessario l'utilizzo di Inner Class e di altri thread eseguiti per una gestione concorrente del sistema. Inoltre sono state implementate delle classi necessarie per la strutturazione dei dati e la personalizzazione della rete sociale (queste utlime classi verrano espresse in dettaglio nella sezione 3.3).

La struttura del server è la seguente:

-Classi interne:

ClientHandler: classe che implementa l'interfaccia 'Runnable'. Viene creato un oggetto di questa classe e passato al thread pool (che manderà in esecuzione il suo metodo run) per ogni processo client che viene messo in esecuzione, ovvero dopo ogni volta che la accept() della socket riceve una richiesta.
Questa classe contiene i metodi principali richiesti per eseguire le operazioni possibili elencate precedentemente e ogni thread di questa classe gestirà le richieste di un singolo client.

ConnectionHandler: classe che implementa l'interfaccia 'Runnable'. Un thread di questa classe viene mandato in esecuzione prima che il thread principale di ServerMain si metta in attesa sulla accept() della socket e serve a gestire la mappa 'clientsConnected' (mappa che contiene le coppie <connessione TCP, utente associato>) e la lista 'loggedUsers' (lista che contiene gli username degli utenti momentaneamente loggati). Se una connessione con un client viene interrotta inaspettatamente questo thread, dopo un certo intervallo di tempo, riaggiorna le due strutture dati precedentemente citate.

ServerNotifyImpl: classe che estende 'RemoteObject' e implementa l'interfaccia 'ServerNotifyInterface'. Questa classe implementa i metodi forniti dall'interfaccia sopra citata ed è necessaria alla realizzazione del meccanismo delle RMI Callbacks. Infatti, i tre metodi presenti nell'interfaccia 'registerForCallback()', 'unregisterForCallback()' e 'restoreFollowers()' vengono implementati nella classe ServerMain, ma vengono chiamati dalla classe ClientMain (sono appunto dei metodi remoti).

Storage: classe che implementa l'interfaccia 'StorageInterface'.
Questa classe fornisce i metodi necessari per la realizzazione del servizio di backup. Un oggetto di questa classe viene creato solo in due circostanze: all'interno del metodo 'restoreStorage()' e all'interno del metodo 'storeStorage()'. Infatti, il primo metodo servirà a ristabilire lo stato del sistema all'avvio del processo ServerMain, mentre il secondo verrà utilizzato ad intervalli (intervallo stabilito nel file di configurazione) all'interno dell'esecuzione del task eseguito grazie alla prossima classe.

TimeOutStorage: classe che estende la classe 'Thread'.
Questa classe permette, tramite la creazione di una sua istanza passata a un oggetto di tipo Thread e mandata in esecuzione con il metodo start() di quest'ultimo, di chiamare periodicamente (intervallo stabilito nel file di configurazione) il metodo "storeStorage()" per scrivere e dunque mantenere negli appositi file di backup (UsersStorage.txt e PostsStorage.txt) lo stato del sistema.

CloseServer: classe che estende la classe 'Thread'.
Questa classe permette, tramite la creazione di una sua istanza passata a un oggetto di tipo Thread e mandata in esecuzione con il metodo start() di quest'ultimo, di mettersi in attesa di ricevere una stringa dallo stdin del server; se la stringa ricevuta in input coincide con la stringa ":q!", allora il processo ServerMain verrà interrotto, altrimenti la stringa verrà ignorata e il processo continuerà normalmente.

ShutDown: classe che estende la classe 'Thread'.
Un'istanza di questa classe viene creata e passata come parametro al metodo addShutdownHook().
Il codice contenuto nel metodo run() di questa classe verrà eseguito esattamente prima che il processo ServerMain venga terminato (qualsiasi sia la causa della terminazione).

-**Main**:
All'interno del metodo main vengono svolte le seguenti operazioni:

- la lettura dei parametri di configurazione necessari al processo server dal file CONFIG_Server.txt;
- il ripristino dello stato del sistema, attraverso la lettura dei file JSON e scrittura nelle apposite strutture dati utilizzate poi durante l'intera esecuzione del processo server;
- l'acquisizione dei due 'registry' necessari per la gestione dei metodi remoti di registrazione e di invio delle notifiche sui nuovi follower;
- l'avvio di tutti i thread (thread eseguiti sia dal threadpool sia singolarmente avviati) necessari a compiere le varie operazioni di:
  - gestione delle connessioni TCP dei diversi client e delle loro richieste,
  - gestione dei salvataggi (effettuati periodicamente) dei valori contenuti nelle strutture dati e riportati all'interno dei due file di storage,
  - gestione delle chiusure inaspettate del server;
- la creazione della serverSocket per la connessione TCP e la creazione del ciclo while dentro il quale la socket si mette in attesa di nuovi client (tramite la accept()) e avvia un nuovo thread ClientHandle tramite la execute() del threadpool.

-**Thread**:
Durante tutta l'esecuzione del processo server saranno attivi i seguenti thread (oltre al thread principale):

-**timeOutStorageThread** (TimeOutStorage): gestisce la scrittura periodica dello stato del sistema all'interno dei due file json per lo storage.

-**closeServerThread** (CloseServer): sta in attesa dell'unico comando che il server può ricevere in input, ovvero il comando ":q!" per la terminazione del thread.

-**rewardMulticastThread** (RewardMulticastTask): calcola periodicamente le ricompense e gestisce il gruppo multicast con cui spedirà il messaggio di avvenuta ricompensa a tutti gli iscritti al servizio.

-**checkConnThread** (ConnectionHandler): gestisce le connessioni TCP e verifica quando una di queste si è interrotta inaspettatamente, aggiornando le strutture dati che ne tengono traccia.

-**service** (ThreadPoolExecutor): questo non è un semplice thread, ma bensì un threadpool; avvia un nuovo thread (un'istanza di tipo 'ClientHandler'), per ogni nuova connessione accettata dalla socket TCP. Nel flusso di ognuno di questi thread, verrà gestita la comunicazione tra il server ed il client associato a tale thread e verranno dunque eseguiti i metodi per la gestione e l'elaborazione delle risposte alle richieste dei singoli client.

-Strutture dati:

sono state utilizzate 4 principali classi per mantenere lo stato del sistema:

-Users (CopyOnWriteArrayList<User>): contiene l'insieme degli utenti registrati nella rete sociale.

-followersMap (ConcurrantHashMap<String, List<String>>): contiene tutte le coppie <utente, lista di followers>. E' stato utilizzato il singolo nome utente e non l'intero oggetto di tipo User nella memorizzazione, poiché nel sistema l'username è una 'PrimaryKey'.

-followingMap (ConcurrantHashMap<String, List<String>>): contiene tutte le coppie <utente, lista dei seguiti>. E' stato utilizzato il singolo nome utente e non l'intero oggetto di tipo User nella memorizzazione, poiché nel sistema l'username è una 'PrimaryKey'.

-Posts (CopyOnWriteArrayList<Post>): contiene l'insieme degli post presenti nella rete sociale.

Le classi User e Post, come il nome stesso spiega, sono le classi che caratterizzano il singolo utente e il singolo post rispettivamente e fanno uso di altre classi ausiliarie come Wallet, Transaction e Comment che verrano successivamente illustrate nella sezione 3.3.

Sono presenti altre due strutture dati per la memorizzazione e gestione degli utenti loggati in quel preciso istante nel sistema e le connessioni TCP persistenti, loggedUsers e clientsConnected rispettivamente.

3.2 Client

La struttura del client e quella di un "thinclient", infatti le principali funzionalità sono demandate al server. Nel client, oltre alla registrazione dei due registry per l'uso del meccanismo di RMI e delle RMI Callback, vi è l'uso della socket TCP per connettersi con il server e la registrazione di un thread per la gestione e l'utilizzo del servizio di mutlicast offerto dal server per la gestione dell'avvenuto calcolo delle ricompense.

La struttura del client è la seguente:

-**Classe interna/Thread**:

MulticastTask: classe che estende la classe Thread.
Questa classe è necessaria per la gestione del servizio di mutlicast offerto dal server. Nel main del client, dopo un'operazione di login dell'utente avvenuta con successo, viene avviato un thread (mcastThread, creando proprio un'istanza di questa classe) e viene mandato in esecuzione con il metodo start();
Il thread appena mandato in esecuzione, dopo aver fatto la join del gruppo multicast, entra in un loop in cui si mette in attesa di ricevere il messaggio del server e una volta ricevuto lo stampa in verde nello stdout del client. Questo loop con continua ricezione di messaggi terminerà solo quando il client effettuerà una operazione di logout; verrà infatti lasciato il gruppo e interrotta l'esecuzione del thread.

-**Main**:
All'interno del metodo main vengono svolte le seguenti operazioni:

- la lettura dei parametri di configurazione necessari al processo client dal file CONFIG_Client.txt;
- l'acquisizione dei due 'registry' necessari per la gestione dei metodi remoti di registrazione e di ricezione delle notifiche sui nuovi follower;
- la creazione della socket per la connessione TCP e la creazione del ciclo while dentro il quale vi sarà l'attesa continua di un comando da tastiera nello stdin del client e che verrà poi parsato e processato per essere inviato al server che ne elaborerà la risposta.
- la riposta del server sarà poi ricevuta tramite il metodo bloccante readLine() invocato da un oggetto di tipo 'BufferedReader' e verrà mandato in stampa sullo stdout in blu.

-**Struttura dati**:
l'unica struttura dati utilizzata nel client è 'follower', un oggetto di tipo List<User> che contiene la lista dei follower dell'utente corrente aggiornata grazie alle comunicazioni del server con le RMI Callbacks.

3.3 Altre classi e interfacce

Nel progetto sono state utilizzate altre classi ed interfacce necessarie alla strutturazione del sistema.

Altre Classi:

- **User**: classe contenente le informazioni relative ad un singolo utente, tra cui il suo username univoco e identificativo, la password, la lista dei suoi tag, la lista dei suoi follower e la lista dei suoi seguaci ed il suo portafoglio.
- **Wallet**: classe contenente l'ammontare del portafoglio di un utente e la storia delle sue transazioni.
- **Transaction**: classe contenente l'ammontare e la data di una singola transazione.
- **Post**: classe contenente le informazioni relative ad un singolo post, tra cui il suo id univoco e identificativo, il titolo, il contenuto, l'autore, il tempo di creazione, la lista degli utenti a cui piace il post e la lista degli utenti a cui non piace il post, la lista dei commenti associati al post, la lista degli utenti che hanno fatto il rewin del post e tutte le informazioni sui recenti curatori del post necessarie al sistema per poter fare il calcolo periodico delle ricompense.
- **Hash**: classe che contiene i metodi necessari per la cifratura in SHA-256 della password.

Interfacce:

- StorageInterface: interfaccia contenente tutti i metodi necessari per effettuare lettura e scrittura dei due file json di storage e delle inerenti strutture dati del server.
- RegisterService: interfaccia remota che fornisce il metodo 'register' necessario all'utente per potersi registrare a Winsome Social Media.
- ServerNotifyInterface: interfaccia remota utilizzata dal client per registrare e deregistrare il proprio interesse al servizio di notifica del server sui nuovi follower o su chi ha smesso di seguire.
- NotifyFollowersInterface: interfaccia remota utilizzata dal client per notificare al client quando vi è un nuovo follower o quando un utente ha smesso di seguirlo.

1. Istruzioni per l'uso

4.1 Compilazione ed esecuzione

Per la compilazione di client, server e di tutte le altre classi del progetto è necessario aprire un terminale, recarsi nella directory "src", contenuta all'interno della directory "530410 SALVATORE SANTORO" estratta dal archivio zip inviato alla consegna, ed eseguire il comando "javac" così come segue:
javac -cp ".:../lib/gson-2.8.9.jar:.:../lib/jline-reader-3.21.0.jar:.:../lib/jline-terminal-3.21.0.jar" *.java

A questo punto, trovandosi sempre all'interno della directory "src", si procede avviando prima il server e poi il client (è ovviamente consentito l'avvio di più client da terminali diversi).

Per l'avvio del server eseguire il comando "java" così come segue:
java -cp .:../lib/gson-2.8.9.jar ServerMain

Per l'avvio del client eseguire il comando "java" così come segue:
java -cp ".:../lib/jline-reader-3.21.0.jar:.:../lib/jline-terminal-3.21.0.jar" ClientMain

Si noti nei comandi "javac" e "java" l'utilizzo del flag '-cp' con il path dei file con estensione jar.
Questi file sono librerie esterne utilizzate per la realizzazione del progetto e si trovano all'interno della directory "lib" contenuta in "530410 SALVATORE SANTORO".
Hint: si faccia attenzione nel caso in cui si copino e incollino i precedenti comandi su terminale; le virgolette potrebbero non essere quelle desiderate dal terminale, è quindi consigliata la riscrittura a mano.

E' inoltre possibile eseguire i processi server e client da due file JAR eseguibili, "Server.jar" e "Client.jar" rispettivamente, che si trovano all'interno della directory "jar" (**java -jar Server.jar** e **java -jar Client.jar**).

Quando il server è pronto, sarà stampato su *stdout* la stringa "Server ready!" in rosso, mentre quando ogni client è attivo apparirà sullo stdout (del terminale da cui è stato attivato il relativo client) la stringa "Insert a command:" in ciano per l'inserimento di un comando da tastiera (stdin) e sullo stdout del server la scritta "A new ClientHandler task is now running".

Gli utenti pre-registrati nel sistema sono 2, chiamati "userA" e "userB". Ad ognuno di essi è associata una password: per userA la password è "12345" mentre per userB è "54321". Ognuno di essi ha inoltre dei tag: per userA sono "hobbyA" e "hobby1" mentre per userB sono "hobbyB" e "hobby1.
Inoltre userA ha pubblicato un post di nome "postA" con associato l'id 0, mentre usersB ha pubblicato un post di nome "postB" con associato l'id 1, entrambi con i relativi contenuti.
Utilizzando il comando login correttamente, avverrà l'autenticazione dello user nel sistema; quando è stata completata, sullo *stdout* del server comparirà la stringa "Client registered to the Callback service" in blu mentre sullo *stdout* del client comparirà la stringa "username logged in" in blu.

Dopo aver completato con successo il login, l'utente può effettuare tutte le operazioni fornite dal servizio, tranne quella di registrazione, effettuata con il comando "register".
Quest'ultima è consentita solo prima di aver effettuato un'operazione di login o dopo aver effettuato un'operazione di logout. Se l'operazione di registrazione è avvenuta con successo il client riceve una risposta dal server con scritto "User 'username' registered" in blu.

Oltre che tramite l'operazione 'logout', l'effettivo logout di un utente dal sistema (utente correntemente loggato) può essere causato dal comando ":q!" o dal segnale di interruzione "^C".
Se il logout è avvenuto tramite il comando 'logout', allora sullo stdout del client comparirà la stringa "username logged out" e sullo stdout del server la stringa "Client unregistered from Callback service".
Negli altri due casi, nello stdout del client non comparirà alcuna risposta, poiché il processo sarà stato interrotto, e nello stdout del server comparirà comunque "Client unregistered from Callback service".

4.1 Sintassi dei comandi

CAMANDO	SINTASSI DEL COMANDO	EFFETTO (in caso di successo)
REGISTER	*register username password <tag1 … tag5>*	Registra un nuovo utente con username 'username' e password 'password' e con almeno un tag (sono ammessi al massimo 5 tag).
LOGIN	*login username password*	Login dell'utente con username 'username' registrato al servizio (se la password è corretta).
LOGOUT	*logout*	Logout dell'utente dal servizio.
LIST USERS	*list users*	Mostra la lista degli utenti presenti nella rete sociale (con almeno un tag in comune con l'utente).
LIST FOLLOWERS	*list followers*	Mostra la lista dei follower dell'utente.
LIST FOLLOWING	*list following*	Mostra la lista degli utenti che l'utente segue.
FOLLOW	*follow username*	Segue l'utente con username 'username'. Il follow va a buon fine solo se quest'ultimo è registrato ed ha almeno un tag in comune con l'utente.
UNFOLLOW	*unfollow username*	Smette di seguire l'utente con username 'username'. L'operazione di unfollow va a buon fine solo se l'utente seguiva questo quest'ultimo.
BLOG	*blog*	Mostra la lista dei post creati dall'utente o di cui quest'ultimo ha fatto il "rewin".
POST	*post "title" "content"*	Crea il post con titolo 'title' e contenuto 'content'. Alla creazione viene associato un id univoco al post.
SHOW FEED	*show feed*	Mostra la lista dei post presenti nei blog degli utenti che l'utente segue.
SHOW POST	*show post idPost*	Mostra il post con id 'idPost' con tutti i suoi dettagli. Il post viene mostrato solo se è presente nel proprio blog o nel proprio feed.
DELETE	*delete idPost*	Cancella il post con id 'idPost' solo se l'utente richiedente è anche l'autore del post.
REWIN	*rewin idPost*	Decide mi portare il post con 'idPost' nel proprio blog, nonostante esso non sia l'autore.
RATE	*rate idPost vote*	Da una valutazione positiva o negativa al post con id 'idPost' solo se l'utente ha il blog nel proprio feed e non aveva già dato un voto in precedenza. (positiva : +1, negativa : -1, altrimenti errore)
COMMENT	*comment idPost "content"*	Commenta il post con id 'idPost' con il contenuto 'content' solo se l'utente non è il suo autore.
WALLET	*wallet*	Mostra l'ammontare del portafoglio dell'utente e tutte le transizioni ad esso associate.
WALLET BTC	*wallet btc*	Mostra l'ammontare del portafoglio dell'utente e ne restituisce anche il valore convertito in bitcoin. (il cambio di valuta varia tra le varie richieste).
QUIT	*:q!*	Termina il processo client.