Laboratorio di Reti, Corsi A e B

WINSOME: a reWardINg SOcial Media

Progetto di Fine Corso A.A. 2021/22

versione 1

1 Contesto

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 Descrizione del sistema

Il progetto consiste nello sviluppo di una rete sociale orientata ai contenuti (una "social media network") caratterizzata da un insieme minimo ma significativo di funzionalità.

In particolare, ciascun utente registrato può seguire altri utenti ed essere a sua volta seguito. Ogni utente può avere quindi un certo numero di "follower". Questo meccanismo permette di presentare a un utente U solo i contenuti pubblicati dagli utenti che U segue. L'utente viene ricompensato dal servizio se pubblica contenuti che riscuotono interesse da parte degli altri utenti e/o se contribuisce attivamente votando o commentando contenuti pubblicati da altri utenti, che vengono accolti con successo dalla comunità.

L'utente ha un profilo caratterizzato da una lista di tag che ha fornito al momento della registrazione. Quando un utente U chiede di visualizzare la lista degli utenti, tra cui potrà scegliere quelli da seguire, gli/le verrà restituita la lista degli utenti che hanno almeno un tag in comune con U.

Ogni utente U può visualizzare il suo feed, che contiene la lista dei post delle persone che segue (per cui è un follower). Può a sua volta gestire un blog, su cui pubblica i propri post. Ad esempio, se U è follower di A e B, U può vedere nel suo feed i post pubblicati da A e B nei rispettivi blog, mentre gli utenti che sono follower di U vedranno, nel proprio feed, i post pubblicati da U nel proprio blog.

WINSOME inoltre implementa un meccanismo denominato "rewin", simile al retweet di Twitter o al Resteem di STEEMIT. Un utente U può condividere sul proprio blog un post pubblicato sul blog di un utente W che U segue, se ritiene opportuno dare visibilità a quel post. In questo modo, il post di W verrà visualizzato anche su tutti i feed dei follower di U.

L'utente può assegnare un voto positivo o negativo (upvote o downvote) a ogni post visualizzato nel proprio feed, a seconda che l'utente voglia esprimere un gradimento o no. E' possibile anche inserire un commento al post. Non è tuttavia possibile inserire commenti che si riferiscono ad altri commenti.

WINSOME premia con delle ricompense gli utenti che hanno pubblicato post ritenuti interessanti e gli utenti che hanno contribuito inviando voti positivi o commenti ai post.

Le ricompense sono nella valuta di WINSOME, “wincoin”. Ogni utente ha associato un portafoglio in cui sono custodite le ricompense ricevute.

Ci sono due tipi di ricompense:

- **Ricompensa Autore**: ottenuta da chi pubblica il post se il post riscuote interesse (ovvero gli utenti inseriscono commenti e/o assegnano voti a quel post).
- **Ricompensa Curatore**: premia chi ha contribuito a determinare il gradimento del post inserendo commenti e/o assegnando voti positivi (NB non viene ricompensato chi ha dato un voto negativo).

Per ogni post viene quindi calcolata una ricompensa, divisa con una percentuale configurabile tra Ricompensa Autore e Ricompensa Curatore. Ad esempio, se il sistema calcola che un certo post ottiene una ricompensa di 100 wincoin, e la percentuale assegnata alla Ricompensa Autore è del 70%, 70 wincoin saranno assegnati all’autore, mentre i restanti 30 (30%) saranno divisi equamente tra i curatori del post.

Il sistema periodicamente (il periodo deve essere configurabile, ad es. 7 giorni o 2 minuti), controlla voti e commenti creati nell’ultimo periodo per i post gestiti, calcola le ricompense e aggiorna il portafoglio degli utenti, ove necessario.

Le somme in wincoin sono convertibili in bitcoin. Per effettuare il cambio esatto è necessario consultare un servizio esterno che fornisce il tasso di cambio wincoin/bitcoin. La fluttuazione del tasso di cambio nel corso del tempo viene simulata tramite un valore casuale (e dunque diverso ogni volta che si vuole conoscere il cambio). Si veda la Sezione 3 per ulteriori dettagli sull’implementazione.

Gli utenti possono accedere ai servizi di WINSOME dopo registrazione e login. Alla registrazione l’utente deve fornire una lista di tag (massimo 5, ad es. “sport musica cinema”). La lista dei tag non può essere cambiata. Dopo la registrazione un utente può:

- effettuare il login
- recuperare una lista di utenti che hanno in comune almeno un tag con l’utente che fa la richiesta
- scegliere gli utenti da seguire o da non seguire più
- pubblicare post (testo)
- effettuare il “rewin” di un post dal proprio feed al proprio blog
- ricevere la lista dei post presenti sul suo feed
- visualizzare un post
- assegnare un voto (positivo o negativo) a un post sul proprio feed
- inserire un commento a un post sul proprio feed
- consultare il proprio portafoglio
- convertire in bitcoin la somma di wincoin nel proprio portafoglio
- effettuare l’operazione di logout

La rete sociale deve essere implementata mediante due componenti principali, che interagiscono usando diversi protocolli e paradigmi di comunicazione di rete (TCP, UDP e RMI). Le componenti sono le seguenti:

- **WinsomeClient**. Gestisce l’interazione con l’utente, tramite una Command Line Interface (l’interfaccia grafica è opzionale), comunica con il WinsomeServer per eseguire le azioni richieste dall’utente, inoltre partecipa a un gruppo di multicast da cui riceve messaggi di aggiornamento del portafoglio. La sua struttura è quella di un “thinclient”, in quanto la realizzazione delle principali funzionalità è demandata al server.
- **WinsomeServer.** Gestisce la fase di registrazione, memorizza tutti gli utenti registrati e le loro relazioni di follower/followed, gestisce i contenuti prodotti dagli utenti (post, commenti e voti),

calcola periodicamente le ricompense e invia una notifica a tutti gli utenti per avvisare del possibile aggiornamento del portafoglio in seguito del calcolo delle ricompense (il messaggio di notifica è uguale per tutti). Ha quindi una visione completa della rete sociale.

3 Funzionalità principali

Di seguito sono specificate le operazioni offerte dal servizio e i comandi che l'utente digita da tastiera per attivarle.

register(username, password, lista di tag): per inserire un nuovo utente, il server mette a disposizione una operazione di registrazione di un utente. L'utente deve fornire username, password e una lista di tag (massimo 5 tag). Il server risponde con un codice che può indicare l'avvenuta registrazione, oppure, se lo username è già presente, o se la password è vuota, restituisce un messaggio d'errore. Lo username dell'utente deve essere univoco. Come specificato in seguito, le registrazioni sono tra le informazioni da persistere lato server. Il comando da digitare per la registrazione ha la sintassi seguente: `register <username> <password> <tags>`, dove `<tags>` è una lista di tag separati da uno spazio. Per semplicità i tag devono essere salvati in caratteri minuscoli. L'utente deve poter scegliere liberamente la lista (il server non gestisce quindi una lista predefinita di tag).

login(username, password): login di un utente già registrato per accedere al servizio. Il server risponde con un codice che può indicare l'avvenuto login, oppure, se l'utente ha già effettuato la login o la password è errata, restituisce un messaggio d'errore. Il comando corrispondente è `login <username> <password>`.

logout(username): effettua il logout dell'utente dal servizio. Corrisponde al comando `logout`.

listUsers(): utilizzata da un utente per visualizzare la lista (parziale) degli utenti registrati al servizio. Il server restituisce la lista di utenti che hanno almeno un tag in comune con l'utente che ha fatto la richiesta. Il comando associato è `list users`.

listFollowers(): operazione lato client per visualizzare la lista dei propri follower. Questo comando dell'utente non scatena una richiesta sincrona dal client al server. Il client restituisce la lista dei follower mantenuta localmente che viene via via aggiornata grazie a notifiche "asincrone" ricevute dal server. Vedere i dettagli di implementazione nella sezione successiva. Corrisponde al comando `list followers`.

listFollowing(): utilizzata da un utente per visualizzare la lista degli utenti di cui è follower. Questo metodo è corrispondente al comando `list following`.

followUser(idUser): l'utente chiede di seguire l'utente che ha per username idUser. Da quel momento in poi può ricevere tutti i post pubblicati da idUser. Il comando associato è `follow <username>`.

unfollowUser(idUser): l'utente chiede di non seguire più l'utente che ha per username idUser. Il comando associato è `unfollow <username>`.

viewBlog(): operazione per recuperare la lista dei post di cui l'utente è autore. Viene restituita una lista dei post presenti nel blog dell'utente. Per ogni post viene fornito id del post, autore e titolo. Viene attivata con il comando `blog`.

createPost(titolo, contenuto): operazione per pubblicare un nuovo post. L'utente deve fornire titolo e contenuto del post. Il titolo ha lunghezza massima di 20 caratteri e il contenuto una lunghezza massima di 500 caratteri. Se l'operazione va a buon fine, il post è creato e disponibile per i follower dell'autore del post. Il sistema assegna un identificatore univoco a ciascun post creato (idPost). Il comando che l'utente digita per creare un post ha la seguente sintassi: `post <title> <content>`.

showFeed(): operazione per recuperare la lista dei post nel proprio feed. Viene restituita una lista dei post. Per ogni post viene fornito id, autore e titolo del post. La funzione viene attivata mediante il comando `show feed`.

showPost(idPost): il server restituisce titolo, contenuto, numero di voti positivi, numero di voti negativi e commenti del post. Se l'utente è autore del post può cancellare il post con tutto il suo contenuto associato (commenti e voti). Se l'utente ha il post nel proprio feed può esprimere un voto, positivo o negativo (solo un voto, successivi tentativi di voto non saranno accettati dal server, che restituirà un messaggio di errore) e/o inserire un commento. Il comando digitato dall'utente è `show post <id>`.

deletePost(idPost): operazione per cancellare un post. La richiesta viene accettata ed eseguita solo se l'utente è l'autore del post. Il server cancella il post con tutto il suo contenuto associato (commenti e voti). Non vengono calcolate ricompense "parziali", ovvero se un contenuto recente (post, voto o commento) non era stato conteggiato nel calcolo delle ricompense perché ancora il periodo non era scaduto, non viene considerato nel calcolo delle ricompense. Il comando corrispondente alla cancellazione è `delete <idPost>`.

rewinPost(idPost): operazione per effettuare il rewin di un post, ovvero per pubblicare nel proprio blog un post presente nel proprio feed. La funzione viene attivata mediante il comando `rewin <idPost>`.

ratePost(idPost, voto): operazione per assegnare un voto positivo o negativo ad un post. Se l'utente ha il post nel proprio feed e non ha ancora espresso un voto, il voto viene accettato, negli altri casi (ad es. ha già votato il post, non ha il post nel proprio feed, è l'autore del post) il voto non viene accettato e il server restituisce un messaggio di errore. Il comando per assegnare un voto al post è `rate <idPost> <vote>`. Nel comando i voti sono così codificati: voto positivo +1, voto negativo -1.

addComment(idPost, commento): operazione per aggiungere un commento ad un post. Se l'utente ha il post nel proprio feed, il commento viene accettato, negli altri casi (ad es. l'utente non ha il post nel proprio feed oppure è l'autore del post) il commento non viene accettato e il server restituisce un messaggio di errore. Un utente può aggiungere più di un commento ad un post. La sintassi utilizzata dagli utenti per commentare è: `comment <idPost> <comment>`.

getWallet(): operazione per recuperare il valore del proprio portafoglio. Il server restituisce il totale e la storia delle transazioni (ad es. <incremento> <timestamp>). Il comando corrispondente è `wallet`.

getWalletInBitcoin(): operazione per recuperare il valore del proprio portafoglio convertito in bitcoin. Il server utilizza il servizio di generazione di valori random decimali fornito da RANDOM.ORG per ottenere un tasso di cambio casuale e quindi calcola la conversione. L'operazione è eseguita mediante il comando: `wallet btc`.

4 Specifiche per l'implementazione

Nella realizzazione del progetto devono essere utilizzate molte delle tecnologie illustrate durante il corso. In particolare:

- La fase di registrazione viene implementata mediante RMI.
- La fase di login deve essere effettuata come prima operazione dopo aver instaurato una connessione TCP con il server. In seguito alla login il client si registra a un servizio di notifica del server per ricevere aggiornamenti sulla lista dei propri "follower", tramite cui viene avvertito quando un nuovo utente lo segue oppure decide di non seguirlo più. Il servizio di notifica deve essere implementato con il meccanismo di RMI callback. Il client mantiene una struttura dati per

tenere traccia della lista dei follower dell'utente, la lista viene quindi aggiornata in seguito alla ricezione della callback.

- Dopo previa login effettuata con successo, l'utente interagisce, secondo il modello client-server (richieste/risposte), con il server sulla connessione TCP persistente creata, inviando i comandi elencati in precedenza. Tutte le operazioni sono effettuate su questa connessione TCP, eccetto la registrazione (RMI), l'operazione di recupero della lista dei follower (*listFollowers*) che usa la struttura dati locale del client aggiornata tramite il meccanismo di RMI callback (come descritto al punto precedente) e la notifica di avvenuto calcolo delle ricompense inviata dal server su un indirizzo di multicast (vedi punto più avanti).
- Il server può essere realizzato con JAVA I/O e threadpool oppure può effettuare il multiplexing dei canali mediante NIO (eventualmente con threadpool per la gestione delle richieste).
- L'utente interagisce con WINSOME mediante un client che utilizza un'interfaccia a linea di comando. E' facoltativa l'implementazione di un'interfaccia grafica.
- Dopo aver effettuato il calcolo periodico delle ricompense, il server invia una notifica ai client mediante UDP multicast). La modalità con cui il server comunica ai client i riferimenti per mettersi in ascolto sulla MulticastSocket è a scelta dello studente (da motivare nella relazione).
- Il server persiste lo stato del sistema, in particolare: le informazioni di registrazione degli utenti, il profilo degli utenti registrati (inclusi i tag, la lista di follower/followed, il portafoglio o wallet) e i contenuti pubblicati dagli utenti (post, commenti e voti). Queste informazioni devono essere rese persistenti sul file system in uno o più file json. Quando il server viene riavviato tali informazioni sono utilizzate per ricostruire lo stato del sistema.
- Calcolo delle ricompense: DA SPECIFICARE (nella prossima versione del documento).

5 Modalità di svolgimento e consegna

Il materiale da consegnare sul Moodle del corso di Laboratorio di Programmazione di Rete deve comprendere:

- Il codice dell'applicazione e di eventuali programmi utilizzati per il test delle sue funzionalità. In particolare, devono essere rispettati i seguenti vincoli.
    - Il codice deve compilare correttamente da riga di comando (ovvero invocando direttamente il compilatore *javac*). In caso contrario, il progetto non verrà considerato valido.
    - Il codice deve essere ben commentato.
    - Le classi che contengono un metodo main devono contenere "main" nel nome, es. ServerMain.java; per le altre classi non ci sono vincoli, ma nomi mnemonici sono ovviamente apprezzati.
    - Oltre al codice sorgente, è necessario consegnare un file JAR eseguibile per ogni applicazione (es. un file JAR per il client e uno per il server).
    - I parametri di input delle applicazioni (numeri di porta, indirizzi, valori di timeout, ecc.) devono essere letti automaticamente da appositi file di configurazione testuali da consegnare assieme al resto del codice (due file separati per client e server). In particolare, lato server il file di configurazione dovrà contenere: intervallo tra un calcolo della ricompensa e il successivo, percentuale di ricompensa che spetta all'Autore, numeri di porta, indirizzi, eventuali valori di timeout. Si veda la Sezione 7 per un esempio di file di configurazione. Non è consentito leggere i parametri in modo "interattivo" (ovvero facendo in modo che sia il programma a chiederli dopo essere stato avviato).
    - In caso di progetti realizzati con Eclipse, IntelliJ IDEA o altri IDE, è obbligatorio consegnare solamente il codice sorgente, rimuovendo eventuali altri file (o directory) creati dall'IDE per gestire il progetto.

- Allegare eventuali librerie esterne utilizzate (jar).

- la relazione in formato pdf che deve contenere:
  - una descrizione generale dell'architettura complessiva del sistema, in cui sono motivate le scelte di progetto;
  - **uno schema generale dei threads attivati da ogni componente e delle strutture dati utilizzate, con particolare riferimento al controllo della concorrenza**;
  - una descrizione sintetica delle classi definite e indicazioni precise sulle modalità di esecuzione.
  - una sezione di istruzioni su come compilare ed eseguire il progetto (librerie esterne usate, argomenti da passare al codice, sintassi dei comandi per eseguire le varie operazioni...). Questa sezione deve essere un manuale di istruzioni semplice e chiaro per gli utilizzatori del sistema.
  - L'organizzazione e la chiarezza della relazione influiranno sul voto finale.

Il codice che non compila non verrà considerato. Per essere considerato sufficiente e quindi ammissibile per la discussione, il progetto deve implementare tutte le funzioni precedentemente illustrate.

Relazione e codice sorgente devono essere consegnati su Moodle in un unico archivio compresso. Non è richiesta una particolare estensione, ma è fortemente consigliato attenersi alle più comuni: .zip e .tar (.gz, .rar e .7z come seconde scelte). La cartella compressa contenente la soluzione dell'esercizio deve essere chiamata seguendo il formato: MATRICOLA COGNOME NOME.zip. Per esempio, lo studente Mario Rossi con matricola e 012345 deve sottomettere la soluzione in una cartella compressa nominata 012345 ROSSI MARIO.zip

Per domande di chiarimento sul progetto verrà attivato un forum di discussione su Moodle per ciascuno dei moduli di Laboratorio A e B su cui potrete pubblicare le vostre domande.

6 Esempio di utilizzo interfaccia CLI

Di seguito mostriamo alcuni esempi dell'utilizzo dell'interfaccia a riga di comando (CLI), lato client. In tutti gli esempi assumiamo che il server sia già attivo. Le righe che iniziano con '>' sono i comandi da immettere, le righe che iniziano con '<' sono esempi di risposte. La specifica completa della CLI deve essere inclusa nella relazione ed è caldamente consigliata l'implementazione anche di un comando '`help`' che riassuma la sintassi dei comandi accettati dalla CLI.

Registrazione e login utente

Faccio partire il client ($ indica che sono sul terminale):

```
$ java ClientMain
```

Richiedo il login di user1, ma l'utente user1 non esiste:

```
> login user1 pass
< errore, utente user1 non esistente
```

Registro l'utente user1 fornendo password e lista tag:

```
> register user1 pass sports movies music
< ok
```

Registro l'utente user1, ma l'username è già esistente:

```
> register user1 pass sports movies music
< errore, utente user1 già esistente
```

Richiedo il login di user1:

```
> login user1 pass
< user1 logged in
```

Richiedo il login di un utente mentre c'è già un utente loggato:

```
> login user2 password
< c'è un utente già collegato, deve essere prima scollegato
```

Eseguo il logout dell'utente attualmente loggato:

```
> logout
< user1 scollegato
```

Interazione con utenti e post

Dopo avere effettuato il login, è possibile interagire con gli altri utenti della rete sociale cercandoli nella lista degli iscritti. Vengono visualizzati gli iscritti con almeno un tag in comune con l'utente corrente.

```
> list users
<    Utente     |    Tag
< ——————————————————————————————
<    koopa      |    tag1, tag2, tag3
<    peach      |    tag2, tag3, tag4
<    ...
```

Per seguire un utente, si usa il comando `follow`. Per visualizzare i seguiti, si digita `list following`. Analogamente, per visualizzare i propri follower, il comando è `list followers`.

```
> follow peach
< Ora segui peach
> list following
<    Utente          |    Tag
< ——————————————————————————————
<    wario           |    tag1, tag2, tag3
<    mario           |    tag2, tag3, tag4
<    peach      |    tag2, tag3, tag4
```

L'utente può decidere di creare un nuovo post tramite il comando omonimo. Poiché il titolo e il contenuto potrebbero includere spazi, è bene racchiuderli fra virgolette. Una volta creato il post, è possibile recuperarlo tramite l'identificativo e il comando `show post`.

```
> post "Post di prova" "Questo è un post di prova."
< Nuovo post creato (id=2331)
> show post 2331
< Titolo: Post di prova
< Contenuto: Questo è un post di prova
< Voti: 0 positivi, 0 negativi
< Commenti: 0
```

Supponiamo che i follower mario e toad eseguano il comando `comment 2331`, inviando rispettivamente i commenti “Bel post!” e “Complimenti per la fantasia!”. Immaginiamo inoltre che l’utente bowser esegua il comando `rate 2331 -1` per inserire un voto negativo. Il risultato di `show post` verrà aggiornato come segue.

```
< Titolo: Post di prova
< Contenuto: Questo è un post di prova
< Voti: 0 positivi, 1 negativi
< Commenti:
<    mario: “Bel post!”
<    toad: “Complimenti per la fantasia!”
```

Nota: se l’utente bowser esegue nuovamente il comando `rate 2331 -1` per votare nuovamente il post, ottiene un messaggio di errore.

```
< Hai già votato questo post.
```

L’utente corrente può decidere di esplorare il proprio feed e visualizzare tutti i post pubblicati dagli utenti seguiti. Per ogni post, viene visualizzato l’identificativo, l’autore e il titolo.

```
> show feed
< Id            | Autore   | Titolo
< ——————————————————————————————————————————————
< 2819          | peach    | “Suggerimenti per l’asta del fantacalcio”
< 2820          | peach    | “I film di Lynch dal peggiore al migliore”
< 2821          | toad     | “Partita a scacchi”
```

7 Esempio di file di configurazione

```
#
#     Questo è un esempio di file di configurazione.
#     Tutte le righe vuote o che iniziano con '#' verranno ignorate.
#


#     Indirizzo del server
SERVER=192.168.1.2
#     Porta TCP del server
TCPPORT=6666
#     Porta UDP del server
UDPPORT=33333
#     Indirizzo di multicast
MULTICAST=239.255.32.32
#     Porta di multicast
MCASTPORT=44444
#     Host su cui si trova il registry
REGHOST=localhost
#     Porta del registry RMI
REGPORT=7777
#     Timeout della socket
TIMEOUT=100000
```