CANERI FRANCESCO MATRICOLA 581032

RELAZIONE PROGETTO RETI WINSOME

1. VISIONE GENERALE DEL SISTEMA

Il progetto è diviso in due classi principali MainServer e MainClient.

Connessione: Il client si connette tramite TCP al server: a connessione effettuata, può iniziare ad interagire tramite command-line inviando comandi di richiesta. Tutti i comandi disponibili possono essere visualizzati tramite 'help'.

Richieste: Le richieste del client vengono inviate al server tramite TCP, tranne 'register' che sfrutta il meccanismo RMI e 'listFollowers' che visualizza la struttura presente localmente, aggiornata tramite le RMI Callbacks. La correttezza di ogni richiesta verrà controllata dal server e, in caso sia corretta, verrà eseguita; il server invierà una risposta al client in un oggetto di tipo ResponseMessagge che contiene un codice dell'operazione ("OK" se ha avuto successo, un messaggio di errore altrimenti) ed una lista che contiene oggetti. Nel caso di operazioni di wallet invierà un ResponseWallet(che estende ResponseMessage) che contiene in aggiunta una variabile che indica il guadagno.

Gestione concorrenza: La gestione dei thread e della concorrenza è effettuata tramite Multiplexing dei canali mediante NIO. Per gestire la concorrenza del meccanismo RMI è stata usata la keyword synchronized.

Persistenza: La persistenza del server è garantita dai file JSON che vengono aggiornati ogni volta che viene effettuata una qualche operazione di modifica delle strutture dati. I file JSON sono cosi strutturati:
Nella cartella **"./Database"** si trovano 3 sottodirectory:

- **Users**: che contiene un file '.json' per ogni utente registrato
- **Posts**: che contiene un file '.json' per ogni post
- **Followers**: che contiene due file che memorizzano tutte le relazioni di followers/following degli utenti.

Ad ogni utente e ad ogni post corrisponde un unico file: questo perché, ad ogni modifica delle strutture, posso aggiornare un solo file che contiene quindi un solo oggetto, rendendo cosi l'operazione di scrittura del file meno costosa.

All'avvio, il server recupera tutte le informazioni dai file JSON.

2. Threads

Vengono attivati i threads relativi al server, al client, uno per il calcolo delle ricompense nel server e uno nel client per ricevere il messaggio UDP.
Il thread **CalcoloRicompense** viene creato all'avvio del server, poi eseguito ogni "periodoCalcolo" secondi (specificato nel file di configurazione). Questo thread calcola per ogni post la relativa

ricompensa, aggiornando ogni utente con il guadagno ricevuto dal post; al termine invia un messaggio UDP sul gruppo multicast, ricevuto da tutti gli utenti online.

Il thread **ReadUDPFromServer** riceve il pacchetto UDP e resta attivo fino a quando non avviene il logout dell'utente.

3. Classi

Oltre alle classi precedentemente elencate sono presenti:

- **CallbackInfo**: contiene le informazioni dei clients per il multicast. Viene utilizzata per indirizzare il messaggio di callback solamente al client da aggiornare e non a tutti quelli registrati.
- **Comment:** Contiene le informazioni di un commento.
- **Hash**: Classe utilizzata per cifrare la password.
- **Post**: Contiene le informazioni di un Post.
- **ResponseMessage**: classe che contiene le informazioni di risposta inviate al client.
- **ResponseWallet**: estende ResponseMessage. Aggiunge una variabile guadagno. Utilizzata nelle operazioni di 'wallet'.
- **Transaction**: contiene le informazioni di una transazione.
- **UpdateDatabase**: classe utilizzata per assicurare la persistenza del server. Ad ogni operazione di aggiunta o eliminazione di informazioni dal server, viene invocato un metodo di questa classe per scrivere (o eliminare) le informazioni nei file JSON.
- **Utente**: contiene le informazioni di un utente.
- **Voto**: contiene le informazioni di un voto.

4. Alcune funzionalità

- **Login**: al login di un utente, il client riceve la lista dei suoi followers dal messaggio ricevuto, questa lista verrà poi aggiornata tramite le callbacks. Un utente può essere collegato solamente da un client per volta.
- **Comment e post**: il testo deve essere compreso tra singoli apici(es. $> comment 1 'commento del post') e non può contenere singoli apici (es. post 'titolo' 'questo è un' commento').
- **'listUsers'** stampa tutti gli users, coi relativi tags, che hanno almeno un tag in comune con l'utente che ha eseguito il comando
- **IdPost**: Un post, una volta creato, mantiene per sempre il suo ID. Al backup gli viene riassegnato il suo id che aveva quando il server era stato chiuso. L'assegnazione degli ID avviene in modo "incrementale": ad ogni nuovo post viene assegnato il valore del contatore 'idPostGlobal', che viene in seguito incrementato di 1.
  Al riavvio del server il contatore prende come valore l'Id più alto dei post + 1, se non vi sono post il suo valore è 1.
- **Rewin**: Un utente B che fa il rewin di un post il cui autore è A, rende visibile quel post agli utenti che seguono B. Non viene creato un nuovo post (B non ne diventa autore): Il post continua ad avere come autore A ed ogni interazione avviene sul post di A.

5. Compilazione ed esecuzione

Per la compilazione va specificato il classpath del file JAR di GSON, che si trova dentro la directory "lib" e dove salvare i ".class", dentro la directory bin.

Per l'esecuzione del server va inclusa la directory del file JAR di GSON e dei ".class" e specificato il file di configurazione che si trova dentro la directory "config".

Per l'esecuzione del client va inclusa la directory che contiene i ".classe" e passato il file di configurazione del client che si trova dentro la directory "config".

ESEMPIO

- javac -d ../bin/ -cp .:../lib/gson-2.8.2.jar *.java
- java -cp ../bin/:../lib/gson-2.8.2.jar MainServer ../config/configServer.txt
- java -cp ../bin/ MainClient ../config/configClient.txt

I FILE DI CONFIGURAZIONE

configClient.txt e configServer.txt sono i file di configurazione per il client e server

- Le righe bianche vengono ignorate
- Le righe che iniziano per '#' sono righe di commento
- Un parametro è composto da <nome_parametro> = <valore>
- Tra nome_parametro, l'uguale e il valore ci deve essere uno spazio