WINSOME: a reWardINg SOcial Media

Progetto di Fine Corso di Reti A.A. 2021/22

**Come sono memorizzati i dati di WINSOME**

La classe *Database* si occupa di gestire tutti i dati necessari al funzionamento del server in modo altamente concorrente. Tutti i metodi che offre sono esenti da race-condition e le strutture dati sottostanti sono in grado di scalare linearmente all’aumento del numero di thread.

Le strutture dati principali sono due *ConcurrentHashMap: postMap* e *usersMap.* La *postMap* mappa stringhe (id dei post) ad elementi della classe *Post*,mentre la *usersMap* mappa stringhe (id degli utenti) ad elementi della classe *Utente*. In questo modo il server è in grado di accedere in tempo O(1) a qualsiasi utente o post dato il rispettivo identificativo.

Le classi usate, come detto al punto precedente, sono *Utente* e *Post*. Queste classi si occupano di mantenere in memoria le informazioni importanti delle entità che rappresentano; in particolare la classe *Utente* memorizza followers, following, posts e feed dell’utente intesi tutti come **Identificativi** dell’oggetto che rappresentano, che si trova in realtà nelle varie HashMap. Considerando che le operazioni di richiesta del feed e visione del proprio blog sono probabilmente le due più comuni in qualsiasi social network, questo mi è sembrato il perfetto compromesso tra occupazione in spazio e in tempo. Infatti, al prezzo di un array di stringhe in più per ogni utente, guadagniamo un accesso al feed lineare nel numero di post del feed stesso, cioè ottimo, e lo stesso discorso vale per il proprio blog personale.

Classe Utente:

* Username, password
* Lista tag
* Lista di follower
* Lista following
* Lista Post scritti (attraverso gli ID, perché i post sono debolmente associati all’utente)

Classe Post:

* Id univoco
* Autore (username)
* Titolo
* Contenuto
* Lista di Likes
  + Username
  + Positive/negativo
* Lista di Commenti
  + Username
  + testo
* Guadagno del post
* iterazioni

Problemi:

* Cosa succede se un post che viene cancellato era stato precedentemente “rewind” cioè è presente nella bacheca di un altro utente?

Modifica importante: quando viene pubblicato un post viene messo nel feeed di tutti gli utenti che sono follower del creatore. In questo modo l’operazione show feed è piu veloce, inoltre è molto più semplice verificare se un utente aveva un post nel proprio feed e quindi può aggiungere like e commenti

Architettura di massima:

Database: si occupa di gestire le strutture dati necessarie al social e garantisce la gestione corretta della concorrenza, offre delle API al server per permettergli di eseguire le operazioni del social

Implementa i metodi:

* 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 (livello database), o se la password è vuota (livello server) , 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 , dove è una lista di tag separati da uno spazio. Per semplicità i tag devono essere salvati in caratteri minuscoli (livello server). 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 . (livello server)
* logout(username): effettua il logout dell’utente dal servizio. Corrisponde al comando logout. (livello server)
* 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 .
* unfollowUser(idUser): l’utente chiede di non seguire più l’utente che ha per username idUser. Il comando associato è unfollow .
* 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 (livello server). Se l’operazione va a buon fine, il post è creato e disponibile per i follower dell’autore del post. 3 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
* 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): attenzione! Non c’è bisogno di sapere chi ha effettuato la richesta 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 .
* deletePost(idPost): attenzione! Non c’è bisogno di sapere chi ha effettuato la richesta, si può lasciare il controllo a livello server in modo da poter implementare un eventuale servizio di moderazione dei contenuti. 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 . 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 . 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 . 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