HOTELIER: *an HOTEL advIsor sERvice*

Progetto di Fine Corso A.A. 2023/24

Pablo Fiz - Corso B - Matricola 598825

Il progetto implementa un sistema distribuito per la gestione di hotel, comprensivo di funzionalità di registrazione utenti, aggiornamento dei ranking degli hotel, e gestione delle recensioni. Il sistema utilizza tecnologie come RMI (Remote Method Invocation) per la gestione degli utenti e la notifica di aggiornamenti, TCP per la comunicazione tra client e server, e multicast per la diffusione di aggiornamenti sugli hotel.

L'implementazione è suddivisa in tre principali componenti: il client, le RMI, e il server TCP.

Componente Client

Il client è responsabile dell'interazione utente e gestisce le operazioni come la registrazione, il login, la ricerca degli hotel e l'inserimento di recensioni. Utilizza le librerie Java IO e Java RMI per la comunicazione sia con il server RMI che con il server TCP.

Componente RMI

La componente RMI è costituita da tre principali classi, tutte implementazioni delle rispettive interfacce:

- 'UserRegisterImpl' per la registrazione e autenticazione degli utenti (l'altra parte del login è gestita invece dal TCP, che interroga l'UserRegister quando deve verificare nome utente e password ricevuti).
- 'RankingUpdateManagerImpl' per la gestione e notifica degli aggiornamenti sui ranking degli hotel nelle località ai quali i client loggati effettuano la 'subscribe'.
- 'RankingUpdateListenerImpl', utilizzato nel Client per creare dei listener da utilizzare per iscriversi agli aggiornamenti del manager e gestire una struttura dati per salvare i ranking ricevuti dal server.

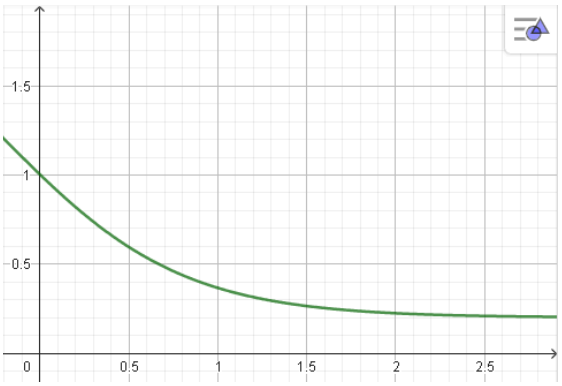
Componente Server TCP

Il server TCP gestisce le richieste provenienti dai client tramite una connessione socket. Implementa diverse operazioni come la gestione del login/logout degli utenti, la ricerca e visualizzazione degli hotel, e l'inserimento di recensioni. Utilizza thread pool per gestire le richieste concorrenti affidandole al RequestHandler e si sincronizza tramite Reentrant Read Write Lock per garantire l'accesso sicuro alle risorse condivise.

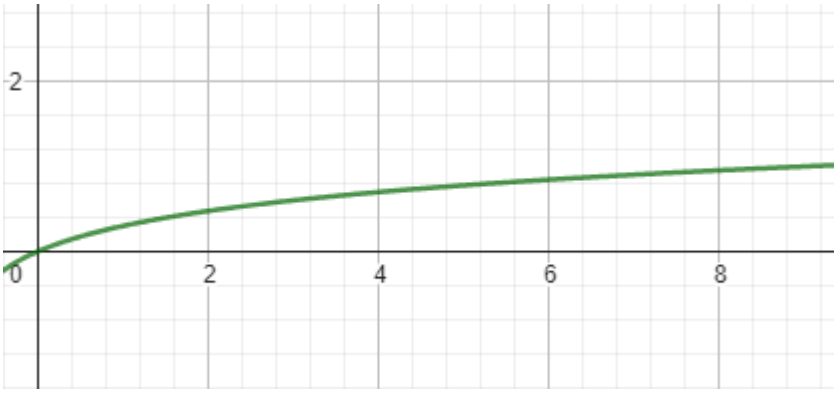
Scelte Progettuali e Strutture Dati

Algoritmo per il Calcolo del Punteggio degli Hotel:
L'algoritmo tiene conto delle diverse recensioni associate a ciascun hotel. Ogni recensione include un punteggio globale e sotto-punteggi per vari aspetti (posizione, pulizia, servizio, prezzo). Il punteggio totale di un hotel viene calcolato come media ponderata dei punteggi globali delle recensioni. In particolare:

- Il peso dei sotto-punteggi è dimezzato rispetto a quello globale.
- L'attualità di una recensione influisce decrementando il punteggio assegnato con l'avanzare dei mesi fino a stabilizzarsi dopo al terzo mese. Il peso è calcolato secondo la seguente formula: *weight = 1.2 - Math.tanh(monthsOld + 0.2)*



- Il numero di recensioni di un hotel influisce sul punteggio con un peso stabilito dalla formula *reviewFactor = Math.log(1 + reviewCount)*



Gestione degli Utenti: Utilizzo di un registro centralizzato `UserRegisterImpl` per la gestione degli utenti, con supporto per la registrazione tramite RMI e funzione di supporto per l'autenticazione TCP, caricamento e salvataggio periodico su file JSON con la libreria Gson tramite i TimerTask.
Utilizzo della classe 'User' per la gestione dei dati inerenti agli utenti. Utilizzo di password salvate dopo l'applicazione di hash+salt per evitare di mantenere un registro di username e password tutto in chiaro (dopo alcune ricerche il salt è risultato necessario per resistere agli attacchi "rainbow table"). Il campo salt viene salvato nel JSON degli utenti insieme all'hash

della password per validare in seguito i tentativi di accesso. Si tiene traccia del numero di recensioni effettuate nella classe User.

Gestione degli Hotel: Utilizzo della classe 'HotelManager', con caricamento e salvataggio periodico degli hotel su file JSON sempre tramite Gson e TimerTask. Oltre ai campi proposti per gli hotel nel testo e nel file di esempio è stato aggiunto il campo "reviews". Ogni hotel ha quindi una lista di recensioni ed ogni recensione ha associata una data e un punteggio. Non si permette l'inserimento di hotel durante l'esecuzione del programma e si gestiscono soltanto quelli caricati inizialmente dal file JSON indicato.

Comunicazione Client-Server: Utilizzo di RMI per le operazioni di registrazione e gestione degli aggiornamenti su iscrizione, TCP per le operazioni di ricerca e recensioni degli hotel, UDP multicast per gli aggiornamenti da inviare a tutti gli utenti loggati.

Strutture Dati

HotelManager:

- private List<Hotel> hotels: Lista di hotel utilizzata per il caricamento e salvataggio su file degli hotel.
- private final Map<String, List<Hotel>> rankedHotelsByCity: HashMap da città a lista di hotel **ordinata** utilizzata per la gestione e l'aggiornamento dei ranking degli hotel

TCPServer:

- private static final Map<Socket, String> loggedInUsers e
- private static final Map<String, Socket> userSockets: HashMap utilizzate per la gestione dei socket associati utenti loggati. Sono l'una la reverse HashMap dell'altra, sacrificando memoria per ottenere maggiori prestazioni in quanto alcune volte si vuole risalire all'user di una socket e viceversa.

UserRegisterImpl:

- private final Map<String, User> users: HashMap da username a istanza della classe User, utilizzata per gestire gli utenti registrati al servizio e per recuperare i dati necessari alla validazione delle loro credenziali.

RankingUpdateManagerImpl:

- private final Map<String, List<RankingUpdateListener>> cityListeners: HashMap da città a lista di Listener utilizzata per la gestione delle notifiche in RMI callback da inviare ai client iscritti.

RankingUpdateListenerImpl:

- private final Map<String, Map<Integer, String>> rankings: HashMap da città a HashMap di Posizione in classifica -> Info Hotel. Gestisce i risultati ricevuti dal listener del RMI callback, utile per un eventuale GUI da sviluppare in futuro.

Sincronizzazione e Thread Safety

L'utilizzo di *ReentrantReadWriteLock* è fondamentale per garantire la sincronizzazione dei thread su strutture dati condivise all'interno del sistema. Questo approccio è adottato per:

HotelManager: Lista di hotel (hotels) e mappa dei ranking degli hotel per città (rankedHotelsByCity).
UserRegisterImpl: Mappa degli utenti (users).

TCPServer: Mappe degli utenti loggati (LoggedInUsers) e delle socket degli utenti (UsersSockets).

ReentrantReadWriteLock consente la lettura concorrente, permettendo a più thread di accedere simultaneamente alle strutture dati per operazioni di sola lettura. Questo migliora significativamente le prestazioni in scenari di accesso concorrente, poiché le letture sono generalmente molto più frequenti delle scritture. Quando è necessaria una scrittura, il lock garantisce che solo un thread alla volta possa modificare i dati, assicurando così la coerenza e l'integrità delle informazioni. Questo approccio di lettura concorrente ma scrittura esclusiva ottimizza l'efficienza del sistema, riducendo i tempi di attesa dei thread e migliorando le prestazioni complessive rispetto all'utilizzo di meccanismi di lock che non differenziano tra lettura e scrittura.

Schema Generale dei Thread

Lato Server

Thread di Connessione TCP (ThreadPool):
Per gestire le richieste concorrenti dei client, viene utilizzato un thread pool. Ogni volta che il server riceve una connessione TCP da un client, un thread dalla pool è assegnato per gestire quella specifica connessione.
TimerTask per il Salvataggio degli Utenti:
Un thread che periodicamente salva i dati degli utenti su un file JSON.
TimerTask per il Calcolo dei Punteggi Locali degli Hotel:
Un thread che periodicamente ricalcola i punteggi locali degli hotel basandosi sulle recensioni attuali.
TimerTask per il Salvataggio degli Hotel:
Un thread che periodicamente salva i dati relativi agli hotel su un file JSON.
Thread RMI: Gestisce le richieste di registrazione e aggiornamento ranking in thread separati, creati dinamicamente per ogni invocazione di metodo remoto.

Lato Client

Thread di Interazione Utente (Main Thread):
Il thread principale che gestisce l'interazione con l'utente tramite la CLI. Riceve i comandi dall'utente e li invia al server, gestisce la visualizzazione delle risposte ricevute dal server.
Thread Listener Multicast UDP:
Ascolta sul gruppo multicast in attesa di aggiornamenti sul ranking e ne stampa il contenuto.
Thread per le notifiche RMI: Gestisce la ricezione delle notifiche di aggiornamento ranking dal server RMI.

Compilazione ed Esecuzione

Pre-requisiti:

- JDK installato sul sistema
- File di configurazione '*server-config.properties'* e '*client-config.properties*' configurati correttamente.

Compilazione: Compilare il codice sorgente posizionandosi nella root del progetto e utilizzando il comando '*javac -cp lib/* -d out/classes src/*.java*' per generare i file '.class'

Esecuzione:

- Avviare il server utilizzando il comando '*java -cp 'out/classes;lib/*' ServerMain*'.
- Avviare il client utilizzando il comando '*java -cp 'out/classes;lib/*' ClientMain*'.

Esecuzione Jar:

- Avviare il server utilizzando il comando '*java -jar Server.jar'.*
- Avviare il client utilizzando il comando '*java -jar Client.jar'.*

Interazione con il Sistema

Si è deciso di mantenere lo stile delle operazioni proposte dal testo, con il nome dell'operazione seguito dai parametri tra parentesi e separati da virgola senza spazi.

Registrazione Utente

register(username,password) | es: register(mario,password123)

Risposta:

- "User registered successfully!" se la registrazione ha avuto successo.
- "Username already exists!" se l'username è già stato registrato.

Login Utente

login(username,password) | es: login(mario,password123)

Risposta:
- “Login successful!” se il login ha avuto successo.
- “Username does not exist!” se l'username non è registrato.
- “Invalid password!” se la password inserita non è corretta.

Logout Utente

logout(username) | es: logout(mario)

Risposta:
- “Logout successful!” se il logout ha avuto successo.
- ”User is not logged in" se l’utente non è loggato.
- “Socket not authenticated for this user" se si sta cercando di effettuare il logout di un’altro utente

Iscrizione agli Aggiornamenti di Ranking per una Città

subscribe(city) | es: subscribe(Roma)

Risposta:
- “Subscribed to ranking updates for Rome” se l'iscrizione ha avuto successo.

Disiscrizione dagli Aggiornamenti di Ranking per una Città

unsubscribe(city) | unsubscribe(Roma)

Risposta:
- “Unsubscribed from ranking updates for Rome” se la disiscrizione ha avuto successo.

Ricerca di un Hotel in una Città

searchHotel(hotelName,city) | es: searchHotel(Hotel Roma 1,Roma)

Risposta:
- Dettagli dell'hotel richiesto se l'hotel è presente nella città specificata.
- “No hotel found in city“ se l'hotel non è presente nella città indicata.

Inserimento di una Recensione per un Hotel

insertReview(hotelName,city,rate,posizione,pulizia,servizio,prezzo) | es:insertReview(Hotel Roma 1,Roma,4,5,4,5,3)

Risposta:

- “Review submitted successfully!” se l'inserimento della recensione ha avuto successo.

Mostra i Badge Utente

showMyBadges() | es: showMyBadges()

Risposta:

- Badge più recente dell'utente se presente.
- “User needs to be logged in to request badges” se l’utente non è loggato

Ricerca di tutti gli Hotel in una Città

searchAllHotels(city) | es: searchAllHotels(Roma)

Risposta:

- Stampa i dettagli di tutti gli hotel presenti nella città specificata.
- “No hotels found in city” se non sono presenti hotel nella città indicata.