Relazione Progetto

Lorenzo Ermini 637417

12 gennaio 2024

1 Introduzione

Il presente documento riporta dettagli sul progetto Java realizzato, il quale mira a sviluppare un sistema client-server che si ispira a Tripadvisor. L'applicazione permette agli utenti di registrarsi, effettuare il login, cercare hotel, lasciare recensioni e ricevere notifiche in caso di aggiornamenti.

2 Scelte Progettuali

2.1 Algoritmo di Calcolo del Ranking degli Hotel

Nel corso dello sviluppo di questo sistema particolare attenzione è stata dedicata alla definizione dell'algoritmo di calcolo del ranking degli hotel. Il meccanismo adottato per determinare il punteggio si basa su tre componenti principali: la valutazione media da parte degli utenti, il numero delle recensioni e l'attualità dell'ultima recensione.
La formula utilizzata è la seguente:

$$ranking = 0.6 * valutazioneMedia + 0.3 * log(numeroRecensioni) + 0.1 * giorniUltimaRecensione$$

Questa si compone di:

- **Valutazione Media (60%)**: Il peso maggiore è stato assegnato alla valutazione media, ritenendo che la percezione qualitativa degli ospiti sia il fattore più significativo.
- **Numero di Recensioni (30%)**: Il logaritmo del numero totale di recensioni contribuisce a riflettere la popolarità e l'affidabilità di un hotel, senza tuttavia sovrastare il merito qualitativo.
- **Recentità delle Recensioni (10%)**: Un minor peso è stato dato all'attualità dell'ultima recensione tramite il conteggio dei giorni trascorsi da essa, per dare evidenza agli hotel che ricevono feedback continuativi nel tempo.

L'intento dietro alla mia formula di calcolo è dare maggiore rilievo alla valutazione media e all'entità delle recensioni, riconoscendo così la qualità e la popolarità di un hotel; ma ho voluto anche assicurarmi di includere l'elemento della "freschezza" delle opinioni, perché credo che un feedback recente sia altrettanto importante.

3 Gestione dei dati tramite file JSON

Altre scelte significative hanno riguardato l'archiviazione e la gestione dei dati. Ho optato per l'impiego di file JSON per la loro leggibilità e facilità di manipolazione. L'eco-sistema dei dati comprende:

- **Hotels.json**: Un file dedicato alla memorizzazione degli hotel con i dettagli come nome, descrizione, servizi offerti, valutazioni e altro, consentendo una veloce ricerca e aggiornamento dei dati.
- **Users.json**: Per garantire la sicurezza e la privacy, i dati degli utenti registrati sono conservati in un file separato che include username e password.
- **Reviews.json**: Le recensioni rappresentano il cuore dell'interazione utente ed è perciò fondamentale la loro persistenza in un file dedicato, che include informazioni come valutazioni e data di pubblicazione.
- **Ranking.json** : Per poter fornire una vista immediata dei migliori hotel per ciascuna città, mantengo un file che riflette la classifica aggiornata per ogni città.
- **Config.json**: La flessibilità del progetto è garantita da un file di configurazione che permette di adattare il comportamento dell'applicazione senza necessità di intervenire direttamente sul codice.

L'uso di questi file JSON si rivela particolarmente efficace in termini di scalabilità e mantenibilità del sistema, offrendo al contempo una soluzione altamente modulare per la gestione dei diversi tipi di dati.

4 Strutture Dati Utilizzate

- **HashMap per la Gestione Comandi**: Per ogni possibile scenario nel flusso di interazione con il client, ho scelto l'impiego di una HashMap per associare a ogni comando un identificativo univoco (la chiave) e una descrizione del comando stesso (il valore). Questo consente di presentare all'utente un set di azioni disponibili piacevolmente navigabile e di gestirle in modo dinamico nel codice.
- **HashMap per gli Utenti Loggati**: Per mantenere un tracciamento degli utenti attualmente collegati e per permettere l'invio di notifiche in maniera efficiente, ho implementato un'HashMap per mappare l'username dell'utente al socket di connessione corrispondente. Questo permette un rapido recupero del socket necessario alla comunicazione quando si verificano cambiamenti nel ranking o si deve inviare un aggiornamento specifico.

5 Gestione del Multithreading

5.1 Lato Server

Nel design del sistema server, la gestione del multithreading gioca un ruolo cruciale per garantire efficienza e reattività alle richieste degli utenti. A tale scopo, ho adottato l'utilizzo di due diversi ThreadPoolExecutor per separare le preoccupazioni e gestire le connessioni in maniera ordinata:

1. **ThreadPool per Operazioni Standard**: Il primo pool di thread è dedicato alla gestione delle richieste di operazioni standard da parte dei client, come registrazione, login, inserimento di recensioni e ricerca di hotel. Questo permette di gestire simultaneamente molteplici connessioni, ognuna delle quali affidata a un thread separato che si occupa dell'intera sessione di comunicazione con il client.

2. **ThreadPool per Notifiche**: Il secondo pool è invece usato per gestire le comunicazioni di notifiche con client specifici quando necessario. Ciò accade, ad esempio, quando vi è un aggiornamento nel ranking degli hotel che potrebbe risultare di interesse per gli utenti attualmente loggati.

Questa separazione garantisce che il sistema di notifiche funzioni in modo indipendente e non rallenti o interferisca con le normali operazioni del server.

5.2 Lato Client

Lato client, ho implementato un approccio simile per garantire una comunicazione simultanea e non bloccante:

1. **Thread di Ascolto per Notifiche**: Una volta stabilita la connessione col server, il client avvia un thread dedicato che rimane in attesa passiva di notifiche. Questo thread è in ascolto in modo continuativo e indipendente dalla UI principale o dalle altre operazioni condotte dall'utente. In questo modo, qualsiasi notifica inviata dal server viene ricevuta e gestita in tempo reale, senza che ciò influenzi l'esecuzione di altre attività sul client.

Utilizzare thread separati per le operazioni e la gestione delle notifiche contribuisce significativamente a un migliore isolamento delle funzionalità all'interno dell'architettura client-server. Tale approccio previene le interferenze tra le diverse comunicazioni, riducendo al minimo il rischio di blocchi o altri problemi di concorrenza. La scelta di strutturare il multithreading in questo modo enfatizza l'importanza di un sistema affidabile e performante, capace di gestire in modo ottimale l'interazione parallela con numerosi utenti.

6 Sincronizzazione

Nel progetto è stata posta una particolare attenzione alla sincronizzazione dei thread nel contesto di accesso e modifica dei file JSON per garantire l'integrità dei dati. Questi file, essendo centrali nel sistema per memorizzare le informazioni di hotel, utenti, recensioni e ranking, possono essere oggetto di accessi concorrenti da parte di più thread.

Per evitare condizioni di gara, stati inconsistenti e potenziali sovrascritture dei dati, ho fatto uso del costrutto **synchronized**. Con questa keyword, è possibile sincronizzare sia blocchi di codice che interi metodi, assicurandosi che un singolo thread per volta possa eseguire il codice critico. Questo significa che, quando un thread sta eseguendo un blocco di codice sincronizzato, tutti gli altri thread che tentano di eseguire quel blocco di codice sono bloccati fino a quando il primo thread non ha terminato le sue operazioni.

Per esempio, nel caso di aggiornamento delle recensioni, il metodo addReview del ReviewManager è marcato come sincronizzato. Quando un thread inizia ad eseguire questo metodo, acquisisce il lock sull'oggetto ReviewManager e altri thread devono attendere fino a quando il metodo non viene completato e il lock rilasciato.

Analogamente, quando si aggiorna il ranking di un hotel dopo una nuova recensione, si utilizza un blocco sincronizzato all'interno del metodo rankHotels del RankingManager. Così facendo, si evita il rischio che operazioni simultanee di ranking portino a risultati incoerenti o errati.

Il vantaggio di utilizzare synchronized è che offre un semplice meccanismo di alto livello per la sincronizzazione, integrato nativamente nel linguaggio Java. Ciò semplifica la gestione dei thread e l'accesso alle risorse condivise riducendo la complessità dell'implementazione e gli errori potenziali.

Una tale strategia garantisce che le interazioni con i file JSON siano atomiche, offrendo una lettura e una scrittura sicure e coerenti nello scenario multithread del nostro sistema client-server.

7 Istruzioni per l'uso

Il sistema client-server che ho sviluppato è stato progettato per essere semplice da avviare e utilizzare.

7.1 Avvio del Server

1. Aprire un terminale.
2. Navigare nella directory contenente il file eseguibile ServerMain.jar.
3. Eseguire il comando:

```
java -jar ServerMain.jar
```

7.2 Avvio del Client

1. Aprire un nuovo terminale.
2. Navigare nella directory contenente il file eseguibile ClientMain.jar.
3. Eseguire il comando:

```
java -jar ClientMain.jar
```

7.3 Configurazione del Sistema

Per default, tutte le impostazioni necessarie per il funzionamento dell'applicazione sono preimpostate e salvate all'interno del file Config.json. Per apportare eventuali modifiche a parametri come porte di ascolto, indirizzi IP del server o altre impostazioni avanzate:

1. Aprire il fileConfig.json con un editor di testo.
2. Effettuare le modifiche necessarie ai valori dei campi.
3. Salvare le modifiche prima di riavviare l'applicazione.

7.4 Uso dell'Applicazione

L'interfaccia utente testuale è stata realizzata per essere intuitiva e fruibile anche da chi si avvicina per la prima volta all'applicazione. Al lancio del client, verranno visualizzate tutte le operazioni possibili: registrazione, login, ricerca hotel e altro ancora. Seguire semplicemente le istruzioni a schermo e digitare i comandi quando richiesto.

Ogni passaggio è stato accuratamente progettato per guidare l'utente attraverso un'esperienza soddisfacente ed efficiente.