Reti: Modulo Laboratorio III
HOTELIER : an HOTEL advIsor sERvice
Progetto di Fine Corso A.A 2023/2024

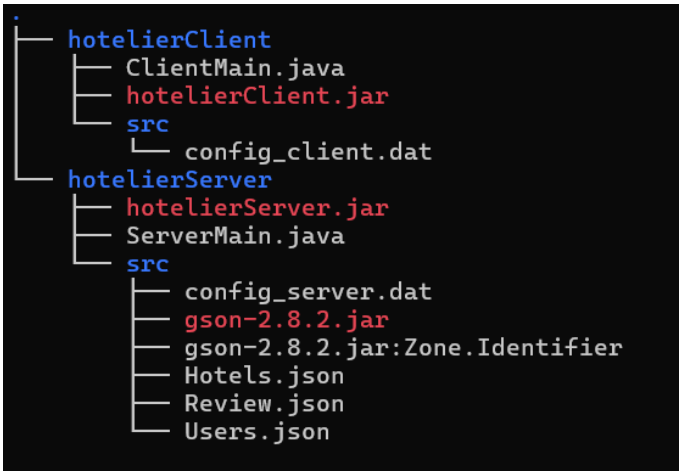
Nome cognome: Mattia Laszlo Daday
Matr: 637784
Data: 17/05/24

Sommario

Nota: Per comprendere al meglio gli algoritmi implementati si possono consultare i commenti nei codici sorgenti

Istruzioni per la compilazione e esecuzione dei due software

In questa sezione si forniscono istruzioni su come compilare e utilizzare il progetto di gestione hotel in Java. Il progetto è diviso in due parti: un client e un server. Il client è responsabile dell'interazione con gli utenti finali, mentre il server gestisce le richieste dei client e utilizza file json per archiviare in spazio fisico i dati. L'albero delle directory del progetto consegnato è il seguente:



Ho separato il codice del client e del server in due directory: hotelierClient e hotelierServer.

hotelierClient: Contiene il codice sorgente e i file necessari per eseguire il client.

- ClientMain.java: Il punto di ingresso del client.
- hotelierClient.jar: Il file eseguibile del client.
- src: La directory che contiene il file di configurazione del client.

hotelierServer: Contiene il codice sorgente e i file necessari per eseguire il server.

- ServerMain.java: Il punto di ingresso del server.
- hotelierServer.jar: Il file eseguibile del server.
- src: La directory che contiene i file json e di configurazione del server.

Per utilizzare il progetto ci sono due modi:

1. Compilare il codice sorgente manualmente:
    - Per il client: bisogna entrare nella cartella hotelierClient e utilizzare il classico comando `javac *.java`. Successivamente si può eseguire il programma compilato con il comando `java ClientMain`
    - Per il server: bisogna entrare nella cartella hotelierServer e utilizzare il comando `javac -cp .:src/gson-2.8.2.jar ServerMain.java` e

posso poi eseguire il progetto con il codice `java -cp .:src/gson-2.8.2.jar ServerMain`

2. Eseguire i due programmi usando i file jar forniti:
   - Per il client: entrare nella cartella hotelierClient e usare il comando `java -jar hotelierClient.jar`
   - Per il server: entrare nella cartella hotelierServer e usare il comando `java -jar hotelierServer.jar`

IMPORTANTE: Come da specifiche sul testo si dà per scontato che il server sia acceso per primo rispetto ai vari client.

NB: nella compilazione del client per alcune versioni di java si potrebbe vedere su console due note poichè il sistema riconosce che si usano alcune funzioni deprecate. La compilazione e l'esecuzione funzionano comunque.

NB: in questo progetto si è deciso di utilizzare la versione di gson 2.8.2. . Si può benissimo modificare stando però attenti a posizionarlo nella cartella hotelierServer/src/ così da poter usare i codici forniti prima.

Guida all'utilizzo dei software

Dopo essersi assicurati che il server è attivo e pronto per la ricezione di nuove connessioni possiamo iniziare a collegarci lato client.

All'attivazione del client si nota comparire il menu e pronto per l'inserimento dell'input per la navigazione. Come da menù si può scegliere uno di quelle funzionalità tramite l'inserimento dello specifico numero.
NB: Ci sono anche alcune funzionalità in più per debug oltre a quelle richieste.

I menù in questione sono di due tipo e cambiano se il client è loggato al servizio o no:

Scelte Progettuali

Il progetto Java Client-Server è stato concepito per implementare un sistema di gestione che permette a più client di interagire con un server centrale.

Descrizione del Server:
Il server, sviluppato attraverso il file ServerMain.java, agisce da gestore delle richieste provenienti dai client utilizzando un'architettura basata su selettori. Inoltre, include un thread dedicato per l'invio periodico di messaggi tramite UDP.
Una delle funzioni principali del server è serviceGest(), che gestisce le richieste in ingresso e in uscita dal socket utilizzando l'NIO (Non-blocking I/O) tramite l'utilizzo dei selettori.

La funzione serviceGest() si occupa di monitorare le richieste dei client utilizzando un selettore (Selector) per determinare quali operazioni possono essere eseguite su ogni canale di comunicazione. Utilizzando un ciclo infinito, la funzione rimane in attesa di eventi pronti da elaborare.

All'interno del ciclo, viene eseguita una serie di operazioni per gestire gli eventi associati alle varie chiavi di selezione (SelectionKey), tra cui l'accettazione di nuove connessioni, la lettura dei messaggi inviati dai client e la scrittura delle risposte ai client.

La gestione delle richieste avviene in modo efficiente e non bloccante, consentendo al server di gestire simultaneamente le richieste di più client senza bloccarsi su una singola connessione.

I messaggi che vengono ricevuti dal server sono formattati in un modo specifico in modo tale da attivare funzioni specifiche per richieste specifiche.
I messaggi che i vari client chiedono sono del tipo : [code] - [val] - [val] - … - [val].
All'interno di serviceGest() vi è un algoritmo che una volta prelevato il messaggio (richiesta) di un client la tokenizza e in base al code che ha quello specifico messaggio si esegue una determinata funzione

Utilizzo della libreria esterna Gson:
Per la gestione della serializzazione e deserializzazione degli oggetti JSON, il server utilizza la libreria esterna Gson 2.8.2. Questa libreria semplifica la trasformazione degli oggetti Java in formato JSON e viceversa, facilitando la comunicazione con i client attraverso una rappresentazione dati standardizzata. Si usa questa libreria per aggiornare la lista degli hotel, inserire nuove recensioni in tempo reale e dare l'opportunità a nuovi utenti di registrarsi in modo persistente.

File config server:
Il server utilizza un file di configurazione denominato config_server.dat per inizializzare le impostazioni necessarie al suo funzionamento. Le proprietà definite in questo file vengono caricate al momento della creazione dell'oggetto Server attraverso il costruttore. Di seguito vengono descritte le proprietà utilizzate e il loro ruolo nel funzionamento del server:

address: Specifica l'indirizzo IP o il nome host su cui il server è in ascolto sul protocollo TCP.

port: Specifica il numero di porta su cui il server è in ascolto per le connessioni dei client.

path_hotel: Rappresenta il percorso del file contenente le informazioni sugli hotel. Questo file viene utilizzato dal server per leggere e organizzare le informazioni sugli hotel, consentendo ai client di eseguire ricerche e interrogazioni.

path_users: Indica il percorso del file contenente le informazioni sugli utenti registrati. Questo file viene utilizzato per autenticare gli utenti che cercano di accedere al sistema e per gestire le loro credenziali di accesso.

path_reviews: Rappresenta il percorso del file contenente le recensioni degli hotel. Questo file viene utilizzato dal server per leggere, aggiornare e gestire le recensioni degli hotel inviate dai client.

period_scheduled: Specifica l'intervallo di tempo in secondi tra l'invio periodico di messaggi tramite UDP. Questa proprietà determina la frequenza con cui il server invia notifiche ai client tramite una connessione UDP.

bytebuff_dim_alloc: Specifica la dimensione del buffer di allocazione in byte utilizzato per la lettura e la scrittura dei dati durante le operazioni di I/O con i client. Questa proprietà influisce sulle prestazioni del server e può essere regolata in base ai requisiti specifici del sistema.

groupUDP e portUDP: Indicano l'indirizzo IP multicast e la porta UDP utilizzati per l'invio dei messaggi periodici ai client. Queste informazioni consentono ai client di ricevere aggiornamenti o notifiche dal server tramite una connessione UDP multicast.

serviceStatus: Rappresenta lo stato del servizio del server, che di default è impostato su falso (false), indicando che il servizio non è attivo. Una volta avviato il server, questo stato viene impostato su vero (true) per indicare che il servizio è attivo e in ascolto per le connessioni dei client.

Lato Client
Il progetto del client consente agli utenti di interagire con il server per eseguire varie operazioni come la registrazione, il login, la ricerca di hotel, l'inserimento di recensioni e altro ancora.
Classe MulticastReceiver: Questa classe rappresenta un thread in background per la ricezione di messaggi UDP dal server tramite multicast. Viene attivato solo se l'utente è loggato e visualizza i messaggi ricevuti dal server. A seguito di un logout dell'hotel i messaggi del server non vengono ricevuti più. Notare che i messaggi arrivano solo se si è loggati.
Classe clientObject: Questa classe gestisce il funzionamento principale del client, inclusi la gestione della connessione al server, l'invio di richieste al server e la ricezione di risposte. Contiene anche metodi per le operazioni principali come la registrazione, il login, la ricerca di hotel, l'inserimento di recensioni, ecc.

File config client:
port :questa variabile port del client ha il valore della porta del server a cui il client si connetterà. Il valore è estratto dalle proprietà del file di configurazione utilizzando la chiave "port".

bytebuff_dim_alloc: Qui viene caricata e assegnata alla variabile bytebuff_dim_alloc la dimensione del buffer in byte. Questo buffer viene utilizzato per la gestione dei dati durante le operazioni di comunicazione con il server. Il valore è estratto dalle proprietà del file di configurazione utilizzando la chiave "bytebuff_dim_alloc".

portUDP : Integer.parseInt(prop.getProperty("portUDP"));: Qui viene caricata e assegnata alla variabile portUDP il numero di porta UDP utilizzato per la comunicazione multicast con il server. Il valore è estratto dalle proprietà del file di configurazione utilizzando la chiave "portUDP".

groupUDP : Qui si viene assegnato l'indirizzo del gruppo multicast utilizzato per la comunicazione UDP con il server. Il valore è estratto dalle proprietà del file di configurazione utilizzando la chiave "groupUDP".

state_login: Variabile che tiene traccia dello stato di login di un client. mi permette di attivare o disattivare funzionalità nel menu del client.

Schema generale thread lato server

Main Thread:

Il programma viene avviato eseguendo il metodo main() del file ServerMain.java.
All'interno di main(), vengono inizializzati i parametri e le risorse necessarie al funzionamento del server, inclusi oggetti come Server e Properties utilizzati per leggere le impostazioni di configurazione.
Viene creato un'istanza della classe Server passando le proprietà lette dal file di configurazione. Questo oggetto rappresenta il server principale e sarà responsabile della gestione delle connessioni e delle richieste dei client.
Viene creato e avviato il thread MulticastSender, responsabile dell'invio periodico di messaggi tramite UDP a un gruppo multicast specificato.
MulticastSender Thread:

Il thread MulticastSender viene creato come un'istanza della classe interna MulticastSender, passando il gruppo multicast e la porta UDP su cui inviare i messaggi, oltre all'oggetto Server principale.
Quando viene avviato, il metodo run() viene eseguito.
All'interno di run(), il thread esegue una serie di operazioni periodiche, tra cui il calcolo dei LocalRanking di ogni hotel, la scrittura delle informazioni degli hotel su un file JSON, la creazione di una mappa contenente i primi hotel di ogni città e l'invio di un messaggio UDP

contenente le informazioni aggiornate ai client tramite un gruppo multicast (solo se vi è stato un aggiornamento negli ho).

Calcolo del ranking locale

Il ranking Locale degli hotel di ogni città è gestito dal thread scheduler di tipo ScheduledExecutorService che aggiorna con questo metodo updateLocalRanking() il valore di ranking di ogni hotel. Modificare il ranking vuol dire aggiornare ad ogni hotel il proprio attributo localRanking tramite una formula matematica che tiene conto di rate globale dell'hotel, ratings delle singole generalità , numero recensioni collegate ad esso e il numero di giorni che intercorrono tra la data attuale e la data della recensione più recente.
Facendo questo aggiornamento ci permette di dare ad ogni hotel una valutazione generale così da poter riordinare la graduatoria degli hotel di ogni città
La formula scelta valuta gli hotel con le caratteristiche più alte con un numero sempre più basso in modo da poter poi ordinare in modo crescente gli hotel e avere come primo hotel quello migliore.
La formula è una somma di tutti i fattori detti prima moltiplicati ognuno dei pesi scelti per dare più o meno importanza ai singoli fattori. Per avere l'ordinamento ottimale si usa il reciproco di questa somma

$$localRanking = \frac{1}{(rate \times 0.5)+(cleaning + position + quality + services) \times 0.2)+(numreview \times 0.2)+(datediff \times 0.1)}$$

dove nel codice corrispondono a:
rate = hotel.rate // rate globale associato a un hotel
cleaning = hotel.cleaning //ratings sulla pulizia
position =hotel.position //ratings sulla posizione
quality = hotel.quality //ratings sulla qualità
services = hotel.services //ratings sui servizi
numreview = hotel.numReview // numero di recensioni associate al singolo hotel
datadiff = date.intValue() // Numero di giorni che intercorrono dalla data odierna all'ultima recensione
Da notare che ci sono dei momenti in cui il localRanking non può essere calcolato (es: l'hotel in questione è appena stato creato e ha i rate a 0 e non ha recensioni collegate) si preferisce assegnarli un valore simbolico di 1000 in modo da esser sicuri che questi hotel siano posizionati in fondo alla graduatoria.

Strutture Dati importanti

Il funzionamento delle seguenti sono spiegate più dettagliatamente con i commenti nel codice sorgente
Lato server:

Le strutture dati ConcurrentHashMap<String, List<Hotel>> hotelMap e ConcurrentHashMap<String, List<Review>> reviewMap sono di vitale importanza per il corretto funzionamento del sistema server. Esse sono progettate per essere thread-safe, il

che significa che possono essere accessibili e modificabili da più thread simultaneamente senza incorrere in problemi di concorrenza.

hotelMap: Questa mappa associa una stringa rappresentante il nome della città a una lista di hotel situati in quella città. È utilizzata per archiviare e gestire informazioni sugli hotel presenti nel sistema. La struttura dati ConcurrentHashMap garantisce che le operazioni di lettura e scrittura su questa mappa siano sicure in ambienti multi-threaded, consentendo a più thread di accedere e modificare contemporaneamente i dati degli hotel senza incorrere in inconsistenze o errori.

reviewMap: Analogamente alla hotelMap, questa mappa associa una stringa rappresentante il nome della città a una lista di recensioni relative agli hotel di quella città. È utilizzata per archiviare e gestire le recensioni lasciate dagli utenti sui vari hotel. Anche in questo caso, la struttura dati ConcurrentHashMap garantisce la sicurezza delle operazioni di lettura e scrittura, consentendo a più thread di aggiungere e recuperare recensioni senza causare problemi di concorrenza.

NB: per avere una spiegazione approfondita sulle strutture dati e gli algoritmi consultare i commenti nel codice sorgente