Relazione Progetto Laboratorio

Indice

Protocolli

Connessione client-server

Il client invia messaggi sotto forma di stringa al server tramite connessione TCP. Il formato dei messaggi è il seguente:

```
"TYPE_indirizzoChiamante_parametro1_parametro2_parametro3_parametro4"
```

I parametri possono essere al massimo 4: a seconda del tipo di richiesta cambia la quantità e il tipo.
Le risposte del server, sempre sotto forma di stringhe, possono essere di 2 tipi:

- In caso di richiesta corretta viene inviata direttamente la stringa che verrà stampata all'utente sul client, che può contenere la risposta effettiva o l'eventuale problema che il server ha riscontrato.
- In caso di richiesta errata viene inviato uno specifico codice di errore:
  - `USERN_Y` se l'username è già presente (per la registrazione)
  - `USERN_N` se l'username non è presente (per il login)
  - `EMPTYF` se uno dei campi (parametri) è stato inviato vuoto
  - `WRONGPSW` se la password è errata
  - `HOTEL` se l'hotel non è presente
  - `CITY` se la città non è presente
  - `FORMAT` se la richiesta non è stata formattata correttamente

Connessione Broadcast

L'invio di notifiche da parte del server a i client loggati in caso cambi l'hotel con il maggior ranking in una città, è gestita da una connessione di tipo UDP. Anche in questo caso, il server invia la stringa finale che sarà stampata dal client.

Strutture Dati

Nella cartella `dataModels`, sono definite le strutture dati usate sia dal server che dal client.

- `Capitals` è un enumeration che contiene le città valide per contenere degli hotel. Viene usata dalla classe `HotelManagement` per fare i dovuti controlli.
- `Hotel` comprende i dati di un singolo hotel, i metodi getter e setter e qualche metodo di supporto
- `User` contiene i dati di un utente, ha i metodi getter e setter. Inoltre è presente un metodo `getBadge()` che restituisce il badge in base al punteggio.
- `Review`, oltre ai contenere i dati di una recensione e a fornire metodi getter e setter, ri-definisce il metodo `toString()` e definisce un metodo `fromString()` per riparsarla da stringa. Sono anche presenti due costruttori in base alle necessità di inizializzazione.
- `JsonUtil` comprende tutti i metodi necessari per il parsing da stringa a JSON e viceversa. Viene usata la libreria `Gson`.

Server side

`ServerMain`

La classe `ServerMain` è il punto di ingresso dell'applicazione server che legge i parametri di configurazione da un file di properties, inizializza e avvia il server `HOTELIERServer`.

`HOTELIERServer`

La classe `HOTELIERServer` rappresenta il server vero e proprio che gestisce connessioni TCP e UDP del sistema. Nel costruttore si inizializzano tutte le risorse necessarie: gestori per utenti e hotel, canale socket, selettore e ThreadPool. Il metodo `fetchHandler(SocketChannel socketChannel)` restituisce il gestore di richieste corrispondente a un certo `SocketChannel` e il metodo `removeHandler(RequestHandler handler)` rimuove un gestore di richieste dalla lista. Il metodo `run()` invece gestisce le connessioni dei client e le loro richieste, assegnando ognuna di queste ad un thread. I vari worker elaborano le richieste tramite la classe `RequestHandler` che implementa `Runnable`. La classe `NotificationService` è incaricata di gestire tutto il processo di notifica UDP.

`RequestHandler`

`RequestHandler` legge il contenuto delle richieste, elabora una risposta e la invia al client. Il metodo `run()` esegue la logica principale della classe: controlla la validità del messaggio e lo passa al metodo `dispatcher(String msg)`, il quale chiamera il metodo opportuno per gestire la richiesta a seconda del tipo. Oltre ad i metodi per gestire le signole richieste, questa classe contiene:

- `readAsString()` per leggere i dati che arrivano sulla socket e ritornarli come stringa
- `write(String message)` per inviare messaggi sulla socket
- `sendNotification(String msg)` per lanciare la notifica sul canale udp
- `quit()` per gestire il caso in cui il client chiuda la connessione

`DataPersistence`

Questa classe implementa `Runnable` e viene eseguita periodicamente da un thread creato in `HOTELIERServer` per salvare i dati di utenti e hotel nei rispettivi file JSON in `data\`. I metodi `saveUsers`, `loadUsers`, `saveHotels`, `loadHotels`, caricano e scaricano i dati rispettivamente di utenti e hotel. I dati vengono scaricati dai costruttori di `HotelManagement` e `UserManagement`. Il metodo `run()` usa i metodi per salvare i dati sincronizzandone l'accesso con una `Lock`.

`HotelManagement`

Questa classe fornisce funzionalità per la ricerca di hotel, l'aggiunta di recensioni, il recupero di recensioni e l'aggiornamento delle classifiche degli hotel. La classe utilizza un blocco per garantire la sicurezza del thread durante l'accesso e la modifica dei dati dell'hotel.
I dati dell'hotel vengono archiviati in una mappa, dove la chiave è l'ID dell'hotel e il valore è l'oggetto Hotel, in modo da avere un accesso diretto agli hotel con un riferimento univoco e di favorire un'eventuale aggiornamento dei dati.

ALGORITMO PER L'ASSEGNAMETO DEL RANK

Tramite il metodo `calculateHotelScore()` viene assegnato un punteggo agli hotel con la seguente formula:

$$
\begin{aligned}
&G = \sum_{i=0}^{n} g_i \\
&S = \sum_{j=0}^{n} \frac{cleaningScore_i + positionScore_i + serviceScore_i + qualityScore_i}{4} \\
&R = e^{-d} \\
&totalScore = G \times 0.5 + S \times 0.3 + R \times 0.2
\end{aligned}
$$

dove:

- n è il numero totale di recensioni per quell'hotel
- g_i è il valore della `globalScore` della recensione numero i
- $fieldScore_i$ è il valore del signolo campo della recensione i
- d è il numero di giorni passati dall'ultima recensione

Il metodo `cityHotelsXranking(String city)` ordina gli hotel (raggruppati per città) secondo il valore del punteggio calcolato in `calculateHotelScore()`.
In fine, `updateRanking()`, per ogni città ordina la lista di hotel, assegna il rank e controlla se è cambiato qualche primo classificato in un ranking locale.

`UserManagement`

Analogamente questa classe fornisce funzionalità per registrare nuovi utenti, accedere agli utenti esistenti, disconnettere gli utenti, salvare i dati dell'utente in un file e recuperare le informazioni dell'utente.

ASSEGNAMENTO DEI BADGE AGLI UTENTI

Ogni volta che un utente scrive una recensione, il suo punteggio viene incrementato di 1 punto. Esistono 6 tipi di badge:

- `TO_UNLOCK` se l'utente non ha scritto neanche una recensione
- `LEVEL_N`: ci sono 4 livelli, il livello viene aumentato ogni 5 punti
- `LEVEL_EXPERT` nel caso in cui l'utente abbia superato il punteggio che delimita l'ultimo livello (4)

Client-side

`ClientMain`

Analogamente al lato server, legge i parametri di configurazione per poi inizializzare ed avviare il client `HOTELIERCustomerClient`.

`HOTELIERCustomerClient`

Questa classe gestisce tutta l'interazione client-server. Si connette al server utilizzando un `SocketChannel` configurato non bloccante e un `Selector` per gestire le operazioni di I/O in modo efficiente. Usa un `ByteBuffer` per leggere e scrivere dati da e verso il server tramite `SocketChannel`, nei methodi `readAsString` e `write`.
Contiene la classe `NotificationReciever` per ricevere e stampare i messaggi arrivati dal Broadcast, quindi le notifiche di aggiornamento dei primi classificati nei ranking locali. Questa classe implementa `Runnable` e viene eseguita su un thread a sè stante.
Il metodo `handleUser()` mostra una home page per permettere all'utente di scegliere l'operazione, una volta inserita quest'ultima, il metodo seleziona l'operazione scelta dall'utente, chiamando l'opportuno metodo tra quelli di operazione.

`CLI`

`CLI` rappresenta l'interfaccia da linea di comando per l'utente. Gestisce l'interazione con metodi per ogni operazione più un metodo `homePage()`. Esegue il controllo degli input quando necessario e modifica le opzioni disponibili a seconda che l'utente sia loggato o meno.

Istruzioni

Comando per la compilazione del progetto:

```
javac -d bin -cp lib/* src/main/dataModels/*.java src/main/server/*.java src/main/client/*.java
```

Comando per l'esecuzione del client:

```
java -cp "bin;lib/*" main.client.ClientMain
```

Comando per l'esecuzione del server:

```
java -cp "bin;lib/*" main.server.ServerMain
```

Il server non necessità interazione se non per la chiusura. Per terminarlo sarà sufficente `CTRL+C`.
L'interfaccia del client è pensata per essere semplice, intuitiva e soprattutto autoesplicativa. Le operazioni disponibili sono elencate nella home page del `CLI` e sono selezionabili con il rispettivo numero indicato. Alcune operazioni sono disponibili per l'utente solo quando questo è loggato, altre solo quando non lo è. In ogni caso, quando un'opzione non è disponibile non ne verrà indicato il numero identificativo.