HOTELIER

Lorenzo Bandini

May 17, 2024

Contents

1 Introduzione

Questo progetto, svolto nell'ambito del modulo Laboratorio III del corso di Reti per l'anno accademico 2023/24, ha come obiettivo l'implementazione di un servizio di recensioni per hotel, ispirato al popolare sito TripAdvisor.
Il servizio, denominato **HOTELIER**, si propone di offrire una versione semplificata di alcune delle principali funzionalità di TripAdvisor, focalizzandosi esclusivamente sulla gestione delle recensioni degli hotel situati nelle città capoluogo delle 20 regioni italiane.

TripAdvisor è una piattaforma che permette agli utenti di recensire strutture turistiche come hotel e ristoranti. Gli utenti possono registrarsi, effettuare il login e rilasciare recensioni, assegnando punteggi su diverse categorie quali posizione, pulizia, servizi e rapporto qualità/prezzo. Le recensioni includono punteggi sintetici e testi liberi, e contribuiscono al calcolo del ranking delle strutture, che viene aggiornato quotidianamente.

HOTELIER riprende la logica di base di TripAdvisor, implementando le seguenti funzionalità:

- Registrazione e login degli utenti.
- Ricerca di hotel per nome e città.
- Inserimento di recensioni da parte degli utenti registrati e loggati.
- Calcolo e aggiornamento periodico del ranking degli hotel basato su qualità, quantità e attualità delle recensioni.
- Assegnazione di distintivi agli utenti in base al numero di recensioni effettuate.

Le recensioni degli utenti su **HOTELIER** si limitano ai punteggi numerici per le categorie specifiche e al punteggio sintetico complessivo, senza possibilità di testo libero.
Ogni utente può ottenere distintivi che indicano il livello di esperienza, suddivisi in cinque livelli: Recensore, Recensore esperto, Contributore, Contributore esperto, Contributore Super.

La struttura del progetto prevede una componente client, denominata **HOTELIER_Client**, che gestisce l'interazione con l'utente tramite un'interfaccia a linea di comando (CLI), e una componente server, **HOTELIER_Server**, che gestisce le richieste del client, memorizza le informazioni sugli utenti e sugli hotel, e aggiorna i ranking.

Il presente documento descrive in dettaglio le scelte progettuali effettuate, le strutture dati utilizzate, lo schema dei thread attivati e le primitive di sincronizzazione adottate per la gestione delle risorse condivise. Inoltre, viene

fornito un manuale di istruzioni per la compilazione e l'esecuzione del progetto, al fine di agevolare gli utilizzatori del sistema.

2 Scelte Progettuali

In questa sezione vengono descritte le principali scelte progettuali effettuate durante lo sviluppo del progetto **HOTELIER**.

2.1 Architettura Client-Server

È stata adottata un'architettura client-server per separare le responsabilità tra l'interfaccia utente e la logica di gestione dei dati. Il client (**HOTELIER_Client**) gestisce l'interazione con l'utente tramite la CLI (Command Line Interface), mentre il server (**HOTELIER_Server**) gestisce le richieste del client e mantiene la persistenza dei dati.

2.2 Protocolli di Comunicazione

Per migliorare l'efficienza e la sicurezza delle comunicazioni, sono stati utilizzati diversi protocolli di rete. In particolare, il protocollo TCP viene utilizzato per le operazioni di registrazione, login e gestione delle recensioni, mentre il protocollo UDP è impiegato per le notifiche di aggiornamento dei ranking.

2.3 Persistenza dei Dati

Per garantire la persistenza dei dati tra diverse sessioni del server, sono stati utilizzati file in formato JSON. Le informazioni relative agli utenti, agli hotel, alle recensioni vengono aggiornate e salvate ad ogni modifica mentre la classifica viene aggiornata a intervalli regolari indicati nel file di configurazione.

2.4 Algoritmo di Calcolo del Ranking

Il calcolo del ranking degli hotel è basato su un algoritmo che tiene conto della qualità, quantità e attualità delle recensioni. Per ogni recensione, viene considerato un punteggio globale e dei punteggi specifici per diverse categorie (Posizione, Pulizia, Servizio, Qualità). Il punteggio di ogni recensione diminuisce nel tempo per riflettere l'attualità delle recensioni. L'algoritmo di calcolo del punteggio totale per un hotel può essere descritto come segue:

$$\text{Score} = \left(\frac{1}{N} \sum_{i=1}^{N} \left(\left(\sum_{j=1}^{4} C_{ij} \right) \times \left(e^{-\alpha \times \frac{D_i}{7}} \right) \times G_i \right) \right) \times \log(1+N) \qquad (1)$$

con $N > 0$ dove:

- N è il numero di recensioni dell'hotel.
- C_{ij} sono i punteggi delle categorie (Posizione, Pulizia, Servizio, Qualità) della recensione i.
- D_i è il numero di giorni trascorsi dalla data della recensione i.
- G_i è il punteggio globale della recensione i.
- α è il coefficiente di decadimento, che controlla il tasso di decadimento esponenziale fissato a 0.01.

Il punteggio di ogni recensione viene calcolato sommando i punteggi delle categorie ponderati dal punteggio globale, e diminuendo il risultato utilizzando un decadimento esponenziale basato sul tempo trascorso dalla recensione. Il punteggio complessivo dell'hotel è la media dei punteggi delle recensioni, incrementata dal logaritmo del numero di recensioni.

Questa metodologia consente di considerare sia la quantità che la qualità delle recensioni, nonché la loro attualità, garantendo che le recensioni più recenti abbiano un peso maggiore nel calcolo del ranking.

Alcuni esempi con alpha differenti:

1. $\alpha = 0.005$:
   - Dopo un mese: il punteggio diminuirà di circa il 0.78%.
   - Dopo un anno: il punteggio diminuirà di circa il 27.11%.
   - Dopo due anni: il punteggio diminuirà di circa il 49.07%.
2. $\alpha = 0.008$:
   - Dopo un mese: il punteggio diminuirà di circa il 1.26%.
   - Dopo un anno: il punteggio diminuirà di circa il 43.02%.
   - Dopo due anni: il punteggio diminuirà di circa il 66.25%.
3. $\alpha = 0.01$:
   - Dopo un mese: il punteggio diminuirà di circa il 1.83%.

- Dopo un anno: il punteggio diminuirà di circa il 59.87%.
- Dopo due anni: il punteggio diminuirà di circa il 80.32%.

Alcuni esempi con vari numeri di recensioni N per hotel:

1. Con $N = 10$ recensioni, il punteggio totale aumenta del 7.84%.
2. Con $N = 50$ recensioni, il punteggio totale aumenta del 39.9%.
3. Con $N = 100$ recensioni, il punteggio totale aumenta del 77.7%.

2.5 Gestione della Concorrenza

Per evitare condizioni di gara e inconsistenze durante l'accesso simultaneo ai file JSON da parte dei vari thread del server, sono state implementate primitive di sincronizzazione lock per ogni specifico file JSON.

2.6 Livelli di Esperienza degli Utenti

Per incentivare la partecipazione e la recensione, è stato implementato un sistema di distintivi basato sul numero di recensioni. Sono previsti 5 livelli di esperienza:

- **Recensore**: Questo badge è assegnato a coloro che hanno scritto meno di 5 recensioni
- **Recensore esperto**: Questo badge è assegnato a coloro che hanno scritto almeno 5 ma meno di 10
- **Contributore**: Questo badge è assegnato a coloro che hanno scritto almeno 10 ma meno di 20
- **Contributore esperto**: Questo badge è assegnato a coloro che hanno scritto almeno 20 ma meno di 30
- **Contributore Super**: Questo badge è assegnato a coloro che hanno scritto più di 30

3 Classi e Strutture Dati

3.1 Lato Server

3.1.1 Hotel.java

La classe `Hotel` rappresenta un hotel e contiene le seguenti informazioni:

- `id`: L'ID univoco dell'hotel.
- `name`: Il nome dell'hotel.
- `description`: La descrizione dell'hotel.
- `city`: La città in cui si trova l'hotel.
- `phone`: Il numero di telefono dell'hotel.
- `services`: Una lista dei servizi offerti dall'hotel.
- `rate`: Il tasso di valutazione dell'hotel.
- `ratings`: Le valutazioni dell'hotel, che includono le categorie di pulizia, posizione, servizio e qualità.

3.1.2 User.java

La classe `User` rappresenta un utente e contiene le seguenti informazioni:

- `username`: Il nome utente dell'utente.
- `hashPassword`: La password dell'utente in formato hash.
- `isLogged`: Un flag che indica se l'utente è attualmente loggato.
- `badge`: Il distintivo assegnato all'utente in base al numero di recensioni inviate.
- `reviewCount`: Il numero totale di recensioni inviate dall'utente.

Ha il metodo `addReview()` che viene utilizzato per incrementare il conteggio delle recensioni e aggiornare automaticamente il distintivo dell'utente in base al numero di recensioni effettuate.

3.1.3 HotelReviews.java

La classe `HotelReviews` rappresenta le recensioni relative a un hotel e contiene le seguenti informazioni:

- `hotelName`: Il nome dell'hotel a cui si riferiscono le recensioni.
- `city`: La città in cui si trova l'hotel.
- `numReviews`: Il numero totale di recensioni relative a quell'hotel.

- `reviews`: Una lista delle recensioni dell'hotel.

Ha il metodo `addReview()` che consente di aggiungere una nuova recensione alla lista e incrementare automaticamente il contatore delle recensioni per quell'hotel.

3.1.4 Review.java

La classe `Review` rappresenta una recensione di un hotel e contiene le seguenti informazioni:

- `reviewer`: Il nome dell'autore della recensione.
- `date`: La data in cui è stata pubblicata la recensione.
- `globalScore`: Il punteggio globale assegnato alla recensione.
- `scores`: I punteggi dettagliati assegnati alle varie categorie (Posizione, Pulizia, Servizio, Qualità) nella recensione.

3.1.5 Ratings.java

La classe `Ratings` rappresenta i punteggi assegnati alle diverse categorie (Pulizia, Posizione, Servizi, Qualità) di un hotel in una recensione. La classe include le seguenti proprietà:

- `cleaning`: Il punteggio assegnato alla pulizia dell'hotel.
- `position`: Il punteggio assegnato alla posizione dell'hotel.
- `services`: Il punteggio assegnato ai servizi offerti dall'hotel.
- `quality`: Il punteggio assegnato alla qualità dell'hotel.

3.1.6 Chart.java

La classe `Chart` rappresenta la classifica delle prestazioni degli hotel in una determinata città. La classe include le seguenti proprietà:

- `city`: La città per cui è stato creata la classifica.
- `hotels`: Una lista di oggetti `HotelInChart` di quella città per formare la classifica.

La classe fornisce un costruttore per inizializzare la classifica con il nome della città e i seguenti metodi:

- `addHotel`: Aggiunge un nuovo hotel alla classifica specificando il nome dell'hotel e il suo punteggio.
- `updateHotel`: Aggiorna il punteggio di un hotel esistente in classifica specificando il nome dell'hotel e il nuovo punteggio.
- `getTopHotelInChart`: Restituisce l'hotel con il punteggio più alto presente in classifica.
- `getHotels`: Restituisce l'elenco completo degli hotel inclusi nella classifica.

3.1.7 HotelInChart.java

La classe `HotelInChart` rappresenta un hotel all'interno di un grafico, con il relativo punteggio. La classe include le seguenti proprietà:

- `name`: Il nome dell'hotel.
- `score`: Il punteggio associato all'hotel.

La classe fornisce un costruttore per inizializzare un'istanza di hotel nel grafico specificando il nome dell'hotel e il suo punteggio. Include inoltre i seguenti metodi:

- `getName`: Restituisce il nome dell'hotel.
- `setName`: Imposta il nome dell'hotel.
- `getScore`: Restituisce il punteggio dell'hotel.
- `setScore`: Imposta il punteggio dell'hotel.

3.1.8 ServerGroupProperties.java

La classe `ServerGroupProperties` viene utilizzata per racchiudere tutte le informazioni prese dal file di configurazione `"server_properties.properties"` tra cui:

- `address`: L'indirizzo IP del server.
- `portNumber`: Il numero di porta del server.
- `minPoolSize`: La dimensione minima del pool di connessioni.
- `maxPoolSize`: La dimensione massima del pool di connessioni.

- `keepAliveTime`: Il tempo di mantenimento inattivo dei thread prima di essere uccisi.
- `timerUpdates`: L'intervallo in secondi per l'aggiornamento della classifica.
- `multicastAddress`: L'indirizzo IP multicast utilizzato per la comunicazione di gruppo.
- `multicastPort`: La porta multicast utilizzata per la comunicazione di gruppo.

3.2 Lato Client

3.2.1 GroupProperties.java

La classe `GroupProperties` viene utilizzata per racchiudere tutte le informazioni prese dal file di configurazione `"client_properties.properties"` tra cui:

- `address`: L'indirizzo IP del server.
- `portNumber`: Il numero di porta del server.
- `multicastAddress`: L'indirizzo IP multicast utilizzato per la comunicazione di gruppo.
- `multicastPort`: La porta multicast utilizzata per la comunicazione di gruppo.

4 Schema dei Thread

Schema generale dei thread attivati sia lato server che lato client.

4.1 Lato Server

Nella classe Main `HotelierServerMain.java` vengono attivati 2 tipi di thread principali per la gestione dei client condivisi ma con funzionalità e servizi differenti:

- `HotelierClientHandler.java` attraverso una connessione TCP instaurata singolarmente con ogni nuovo client che si connette, si occupa di gestire e servire ogni richiesta attuando modifiche sincrone con gli

altri thread sui file JSON. I vari `HotelierClientHandler.java` avviati vengono gestiti da un `ThreadPoolExecutor` i quali parametri vengono forniti attraverso il file di configurazione.

- `HotelierUpdaterChart.java` attraverso una connessione UDP multicast singola ad un gruppo a cui si uniscono tutti i client connessi, attraverso l'utilizzo di due `ScheduledExecutorService`, uno per tenere connessi gli utenti inviando un messaggio predefinito e l'altro per aggiornare regolarmente la classifica e notificare i client in caso di cambiamenti nella posizione di una città al primo posto. Il tempo di aggiornamento della classifica è specificato nel file di configurazione.

4.2 Lato Client

Nella classe Main `HotelierClientMain.java` vengono attivati 3 thread mediante l'utilizzo di un `ExecutorService`:

- `ClientListener.java` per ascoltare e scrivere su CLI tutto ciò che ci viene scritto dal server
- `ClientWriter.java` per scrivere al server tramite CLI i comandi da eseguire
- `MulitcastListener.java` per ascoltare e scrivere su CLI tutte le notifiche che arrivano dal gruppo multicast gestito dal server

5 Manuale di Istruzioni

I progetti sono stati sviluppati separando server e client. Il client è stato sviluppato secondo un progetto Java standard mentre il server è stato sviluppato mediante l'utilizzo di Maven per facilitare la compilazione e l'esecuzione del codice e delle sue dipendenze.

5.1 Compilazione e Esecuzione del Progetto

5.1.1 Compilazione e Esecuzione Server

Per prima cosa bisogna installare tutte le dipendenze mediante:

```
mvn clear install
```

Per eseguire il codice possiamo fare semplicemente:

```
mvn exec:java
```

Se invece vogliamo creare il file .jar dobbiamo eseguire:

```
mvn clean package
```

ciò creerà il due file jar nella cartella target, uno originale senza dipendenze, chiamato `original-hotelier-1.0-SNAPSHOT.jar` e un altro, creato grazie al plugin Shade di Maven chiamato `hotelier-1.0-SNAPSHOT.jar` che conterrà anche le dipendenze. Adesso ci basterà eseguire quest'ultimo con:

```
java -jar target/hotelier-1.0-SNAPSHOT.jar
```

5.1.2 Compilazione ed Esecuzione Client

Per prima cosa bisogna bisogna spostarsi nella cartella con tutto il codice tramite

```
cd src
```

All'interno di questa cartella dobbiamo per prima cosa compilare il codice mediante:

```
javac HotelierClientMain.java
```

e poi lo possiamo eseguire con:

```
java HotelierClientMain
```

Se ci troviamo su una macchina Windows è stato creato uno script per agevolare queste manovre che possiamo eseguire semplicemente con:

```
.\client.bat
```

Altrimenti possiamo eseguire il .jar con:

```
java -jar hotelier-1.0-SNAPSHOT.jar
```

5.2 Dipendenze Esterne Utilizzate

Nel server sono state utilizzate le seguenti dipendenze, gestite direttamente da Maven:

- **Gson (Google)**: Libreria Java per la serializzazione e deserializzazione di oggetti Java in JSON e viceversa.

- **Javax XML bind**: Libreria per la mappatura di oggetti Java a documenti XML e viceversa. (Usata per hashing delle password)
- **Maven Shade Plugin**: Plugin Maven per creare un file JAR "ombra" con tutte le dipendenze.
- **Maven Jar Plugin**: Plugin Maven per creare un file JAR contenente il codice sorgente compilato.
- **Maven Exec Plugin**: Plugin Maven per eseguire applicazioni Java durante il processo di build.