Relazione

Progetto di Laboratorio III

Hotelier

Studente: Chicca Julie

Matricola: 559364

ANNO ACCADEMICO 2023/2024

Indice

1. Introduzione

TripAdvisor è una piattaforma online completa che offre una serie di servizi per i viaggiatori, che includono la possibilità di prenotare alloggi, voli, noleggio auto ed altre attività. La piattaforma si distingue per il suo servizio di recensioni, che permette agli utenti di condividere le loro esperienze personali riguardanti alberghi, bed and breakfast, ristoranti e altre strutture legate ai viaggi. Queste recensioni forniscono un feedback prezioso per altri viaggiatori in fase di pianificazione del viaggio, anche per il fatto che TripAdvisor permette agli utenti di condividere fotografie personali delle loro esperienze di viaggio all'interno delle recensioni, arricchendo così la comunità di viaggiatori con una ricchezza di informazioni visive.

Nella piattaforma, le recensioni hanno 5 punteggi: Punteggio sintetico, Posizione, Pulizia, Servizio, Qualità-Prezzo.

La piattaforma offre un forum dove gli utenti possono fare domande, condividere consigli e discutere di vari argomenti legati ai viaggi, e fornisce anche delle guide per pianificare al meglio l'esperienza.

Un altro aspetto importante di TripAdvisor è il sistema di "badge", che vengono assegnati agli utenti ogni volta che raggiungono un particolare goal. I suddetti bad-

ge, pur non dando diritto a premi, servono a indicare il livello di ”esperienza” del recensore. Per esempio, questi sono i ”Traguardi” di un utente su TripAdvisor:

Come è possibile notare, i badge (riferiti ai Traguardi) possono riguardare il numero di recensioni, il numero di foto pubblicate e quante visualizzazioni le recensioni hanno avuto.

Inoltre, ci sono dei badge che riguardano anche gli interessi personali o sottocategorie di strutture, come ad esempio il numero di recensioni fatte su Hotel di Lusso in figura:

I gestori di strutture come hotel e ristoranti possono registrarsi su TripAdvisor per pubblicare i dati delle loro strutture, e possono anche iscriversi a un servizio di notifica che li avvisa ogni volta che viene pubblicata una nuova recensione sulla loro

struttura. Questo permette loro di rimanere aggiornati sul feedback dei clienti e di migliorare continuamente i loro servizi.

2. Progetto di Laboratorio III

Hotelier

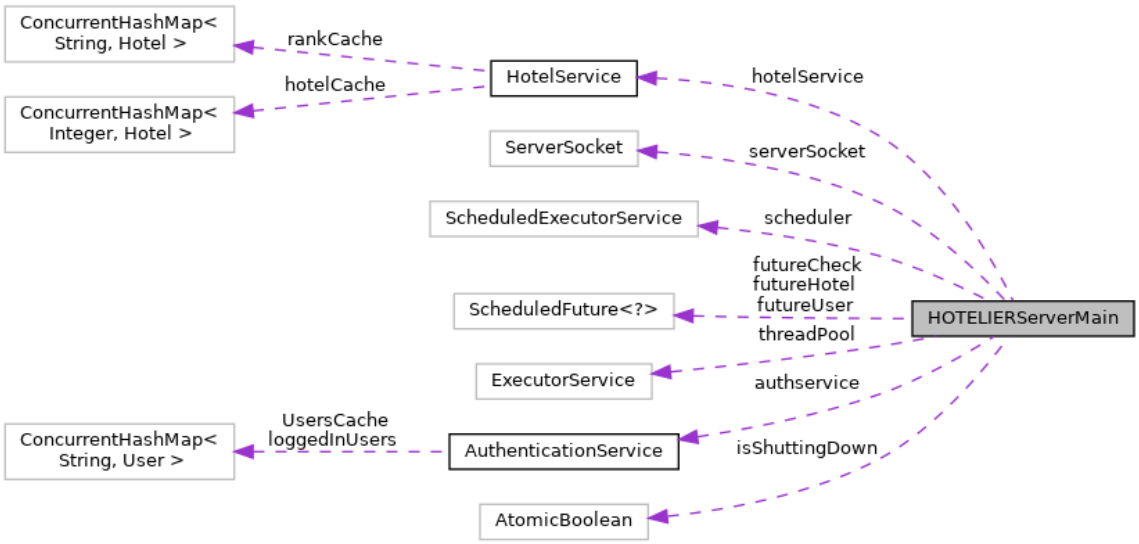
Il progetto finale del corso di Laboratorio III si focalizza su una versione semplificata della piattaforma TripAdvisor, chiamata Hotelier, che implementa un sottoinsieme delle funzionalità dell'applicazione originale, e come strutture riguarderà solamente gli Hotel. Ho realizzato questo progetto seguendo lo standard Java 8 ed usando la libreria GSON. Il progetto è strutturato seguendo un modello client-server, rispettando il paradigma della programmazione orientata agli oggetti. Ogni funzionalità è attentamente incapsulata in classi distinte, garantendo così una chiara separazione delle responsabilità. In seguito, analizzerò la struttura del Server, del Client e delle altre classi.

2.1 Server

Il server sta in ascolto sulla porta 12000 all'indirizzo 127.0.0.1 ed è composto dalle seguenti classi:

- `HOTELIERServerMain.java`
- `AuthenticationService.java`
- `ClientHandler.java`
- `HotelService.java`
- `Classi di utilità`

2.1.1 HOTELIERServerMain.java



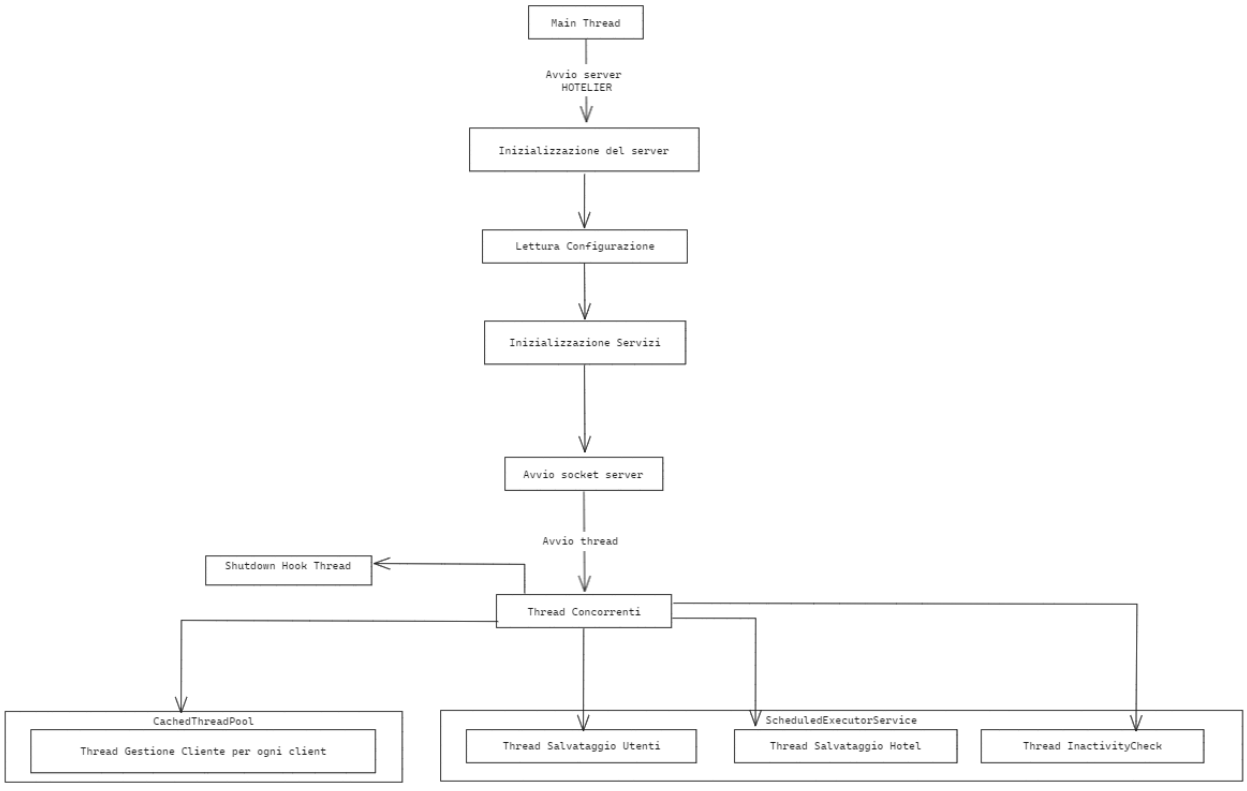
I metodi presenti in questa classe, oltre al main, sono:

- `private static void readConfig()`
- `private static void begin()`
- `private static void shutdown()`

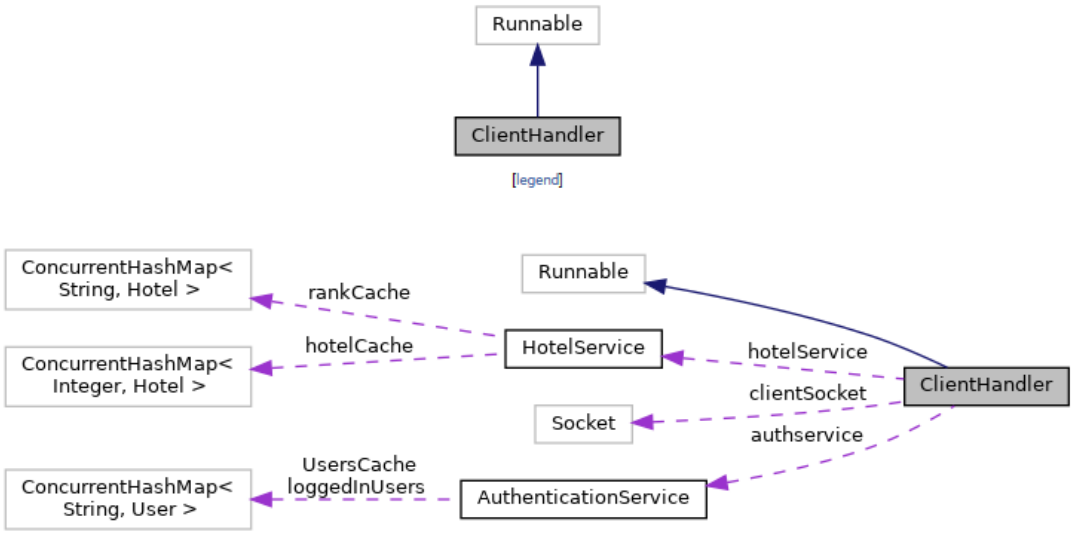
Il metodo `readConfig()` legge tutti i parametri necessari dal file di configurazione, come porte, indirizzi, tempi di timeout, path dei file, e li assegna alle variabili appropriate. Una volta letti i parametri di configurazione, il metodo `main()` aggiunge il gestore delle interruzioni e invoca il metodo `begin()`, che avvia effettivamente il server. Nel metodo `begin()`, vengono inizializzati i servizi autenticazione (`AuthenticationService.java`) e il servizio che gestisce gli hotel (`HotelService.java`), e vengono schedulati i due metodi che persistono periodicamente le informazioni nei file e il checker di inattività, utilizzando `newScheduledThreadPool` con 3 threads, i percorsi dei file e timeout specificati

nelle informazioni di configurazione. Per gestire le richieste dei client in modo efficiente, viene creato un pool di thread utilizzando `Executors.newCachedThreadPool()`, associando un thread ad ogni client, il quale esegue come task `ClientHandler.java`. L'utilizzo di una CachedThreadPool è particolarmente vantaggioso quando il server deve gestire un elevato numero di richieste di connessione, perché il pool di thread è in grado di creare nuovi thread secondo necessità e, allo stesso tempo, riutilizzare i thread inattivi. Questo approccio migliora l'efficienza del server, permettendogli di gestire un numero elevato di richieste in modo efficace.

Qui sotto uno schema sintetico dei thread attivi durante l'esecuzione del server:



2.1.2 ClientHandler.java



Questa classe svolge un ruolo cruciale nella gestione dell'interazione tra il server e ciascun client: infatti, funziona come un intermediario che utilizza i metodi forniti dai servizi di autenticazione e di gestione degli hotel. È importante sottolineare il protocollo di comunicazione tra client e server che ho deciso di utilizzare. Per indicare al client che ha terminato l'invio di messaggi, il server utilizza come segnale di fine trasmissione una linea vuota.

I metodi che appartengono a questa classe sono:

- `public ClientHandler(Socket clientSocket, AuthenticationService authService, HotelService hotelService)`
  Costruttore della classe, inizializza le variabili di istanza.

- `private void logErrorAndPrintMessage(PrintWriter out, Exception e)`
  Logga un errore e stampa il messaggio associato.

- `private void print_protocol(String msg, PrintWriter out)`
  Stampa un messaggio utilizzando il protocollo definito.
- `private User login(User user, BufferedReader in, PrintWriter out)`
  Gestisce la procedura di accesso (login) per un utente.
- `private void signup(User user, BufferedReader in, PrintWriter out)`
  Gestisce la procedura di registrazione (signup) per un utente.
- `private void showBadge(User user, PrintWriter out)`
  Mostra il badge associato a un utente.
- `private void logout(User user, PrintWriter out)`
  Gestisce la procedura di logout per un utente.
- `private void searchHotel(PrintWriter out, BufferedReader in)`
  Esegue la ricerca di un hotel e ne stampa i dettagli.
- `private void searchAllHotels(PrintWriter out, BufferedReader in)`
  Esegue la ricerca di tutti gli hotel in una città e ne stampa i dettagli.
- `private void insertReview(User user, PrintWriter out, BufferedReader in)`
  Gestisce l'inserimento di una recensione da parte di un utente.
- `public void run()`
  Implementa il comportamento del thread associato al gestore del client.

Durante l'esecuzione del programma lato client, all'utente viene proposto un menù a 8 scelte:

- Signup, valore 1 nel menù
- Login, valore 2 nel menù
- Show badge, valore 3 nel menù
- Search Hotel, valore 4 nel menù
- Search All Hotels, valore 5 nel menù
- Insert review, valore 6 nel menù
- Logout, valore 7 nel menù
- Exit, valore 8 nel menù

In risposta alla selezione dell'utente, il ClientHandler attiva uno dei metodi precedentemente menzionati per processare la richiesta, gestendo così il flusso di dati in ingresso e in uscita tra il client e il server. Per assicurare un'esperienza utente fluida, ho scelto di gestire eventuali eccezioni di input/output (IOException) o di autenticazione (AuthenticationException) registrando le eccezioni e inviando un messaggio di errore a scopo informativo all'utente attraverso il metodo `logErrorAndPrintMessage(out, e)` se si tratta di un errore di autenticazione.

La gestione della sessione utente si basa sull'oggetto user: infatti, quando un utente si registra o effettua il login con successo, l'oggetto user viene impostato con le informazioni sull'utente che ha effettuato l'azione. Nel caso della registrazione, non c'è alcuna gestione specifica della sessione, poichè l'utente deve ancora fare il login (come richiesto da testo del progetto), ma quando si effettua il login, l'oggetto user viene impostato con l'utente autenticato. Una possibile alternativa poteva consistere in una autologin implementata subito dopo la registrazione. Mentre l'utente è loggato, tutte le azioni successive eseguite da quel client (ad esempio, ricerca di hotel, inserimento di recensioni, ecc.) avranno accesso alle informazioni dell'utente attraverso l'oggetto user, il quale è diverso da null finchè non viene fatto il logout/exit: infatti, se l'utente decide di eseguire il logout/exit, l'oggetto user vie-

ne impostato a null. L'oggetto user è quindi utilizzato per tracciare lo stato della sessione dell'utente.

Prendendo come esempio il metodo di registrazione, noto come `signup(User user, BufferedReader in, PrintWriter out)`, vediamo una panoramica generale del funzionamento dei metodi: signup gestisce l'intero processo di registrazione di un utente. Il suo flusso di esecuzione inizia con l'inizializzazione di una variabile booleana denominata validPassword e di un oggetto Pattern, entrambi utilizzati per verificare la presenza di caratteri speciali nella password inserita dall'utente. Il metodo effettua anche una verifica per assicurarsi che l'utente corrente non sia già loggato nel sistema, poiché, secondo le mie scelte implementative illustrate in seguito, un utente non può creare un nuovo profilo se è già autenticato con un altro account nella sessione corrente. Superati questi controlli preliminari, il metodo entra in un ciclo iterativo. Durante ciascuna iterazione, all'utente viene richiesto di inserire un nome utente unico e una password che rispetti specifici requisiti di sicurezza. Il sistema controlla anche se il nome utente esiste già usando il metodo `authservice.checkSignup(username)` del servizio di autenticazione, e se sì, annulla l'operazione dato che lo username è un valore univoco e deve essere diverso per ogni utente. Il ciclo continua fino a quando l'utente fornisce una password valida. Per comunicare con il client, il metodo fa uso di `print_protocol`, inviando messaggi informativi durante l'intero processo di registrazione. Inoltre, utilizza il metodo `authservice.signup(user, username, password)` per registrare l'utente nel sistema.

Gli altri metodi si comportano più o meno allo stesso modo: gestiscono il flusso di dati in ingresso e in uscita tra il client e il server e fanno uso dei metodi dei due servizi per soddisfare le richieste dell'utente.

2.1.3 AuthenticationService.java

Questa classe fornisce un servizio di autenticazione e gestione degli utenti. Utilizza due cache: `UsersCache` e `loggedInUsers`. `UsersCache` è utilizzata per memorizzare gli utenti registrati che non sono ancora stati salvati in modo persistente e garantisce che le informazioni sugli utenti siano sempre aggiornate. `loggedInUsers` tiene traccia degli utenti attualmente autenticati nel sistema ed è fondamentale per inviare notifiche tramite messaggi UDP agli utenti loggati. Per garantire l'accesso sicuro e consistente al file degli utenti, `signedUpUsers.json`, i metodi della classe che accedono a sezioni critiche del codice, dove le risorse condivise sono coinvolte, si sincronizzano su un oggetto lock. Una possibile alternativa, lasciata come commenti nel codice, è l'uso di una `ReentrantReadWriteLock` che consente a più thread di leggere i dati contemporaneamente, ma garantisce che solo un thread alla volta possa scrivere (o modificare) i dati. Questo previene qualsiasi stato inconsistente dei dati.
I metodi di questa classe sono:

- `public AuthenticationService(String user_path)`
  Costruttore della classe. Inizializza le strutture dati e le variabili necessarie.

- `protected void checkSignup(String user)`
  Verifica se un utente già esiste al momento della registrazione.

- `protected void checkAlreadyLogin(String user)`
  Verifica se un utente è già loggato quando tenta di loggarsi.

- `protected void checkLogin(String user)`
  Verifica se un utente esiste durante il tentativo di login.

- `protected void signup(User user, String username, String password)`
  Registra un utente nel sistema.

- `protected User login(String username, String password)`
  Effettua il login di un utente.

- `protected String logout(User user)`
  Effettua il logout di un utente.

- `protected Badge showBadge(User user)`
  Restituisce il badge di un utente.

- `protected void printLoggedIn()`
  Stampa le informazioni sugli utenti loggati.

- `protected void printSignedUp()`
  Stampa le informazioni sugli utenti registrati.

- `private boolean isUserInFile(String username)`
  Verifica se l'utente è presente nel file JSON.

- `private User getUserFromFile(String username)`
  Ottiene le informazioni di un utente dal file JSON.

- `protected void saveUsersToJson(File users_file)`
  Salva le informazioni degli utenti su un file JSON: è il metodo invocato da HOTELIERServerMain per persistere i dati degli utenti nel file.

Questa classe implementa metodi 'check' che eseguono controlli preliminari sullo stato dell'utente prima che possa effettuare il login o registrare un account. Le regole implementative adottate sono le seguenti:

- Un utente non può loggarsi se è già loggato con un account in un'altra sessione.

- Un utente non può registrare un nuovo account se è loggato con un altro.

- Un utente non può registrarsi con uno username già presente, per fare in modo di rispettare il vincolo di univocità degli username.

Il metodo `signup` registra un nuovo utente, creando un oggetto User e aggiungendolo alla mappa `UsersCache`. Il metodo `login` effettua il login di un utente, recuperandolo dal file JSON oppure, se ancora presente, dalla cache, restituendo l'utente loggato e aggiungendolo alla mappa `loggedInUsers`. Il metodo `logout` esegue il logout di un utente, rimuovendolo dalla mappa `loggedInUsers` e riportandolo nella mappa `UsersCache` per essere persistito in memoria. La cache `loggedInUsers` è utilizzata per tenere traccia degli utenti attualmente loggati durante l'esecuzione del programma. Ogni volta che un utente esegue il login con successo, l'oggetto `User` corrispondente viene aggiunto a questa cache. Viceversa, quando un utente esegue il logout, l'oggetto `User` viene rimosso. Questa cache è utile per effettuare rapidamente verifiche sullo stato di autenticazione di un utente: infatti, prima di consentire a un utente di effettuare il login, il sistema può controllare se l'utente è già loggato in un'altra sessione, evitando così di consentire l'accesso multiplo con lo stesso account.

È essenziale esaminare attentamente i metodi che interagiscono con il file JSON per la gestione delle informazioni relative agli utenti.

getUserFromFile(String username)

Il metodo `getUserFromFile` preleva un utente dal file JSON degli utenti. Prende come parametro una stringa username e restituisce un oggetto User se l'utente è presente nel file, altrimenti restituisce null. Il metodo può lanciare un'eccezione IOException se si verificano errori durante la lettura del file JSON. Il metodo inizia con un blocco synchronized che utilizza un oggetto lock per garantire l'accesso sicuro alle risorse condivise. Con una ReentrantReadWriteLock, avrei acquisito il lock di lettura. All'interno di questo blocco, il metodo apre un JsonReader per leggere il

file JSON specificato dalla variabile user_path. Il contenuto del file viene quindi analizzato in un oggetto JsonElement utilizzando JsonParser e, se questo JsonElement rappresenta un array JSON, il metodo lo converte in un JsonArray. Successivamente, il metodo itera su ogni elemento dell'array e, per ognuno di essi, estrae il nome utente. Se corrisponde a quello cercato, estrae anche tutte le altre informazioni, crea e restituisce un nuovo oggetto User con le informazioni estratte.

saveUsersToFile(File users_file)

Simmetricamente, il metodo `saveUsersToFile` è un metodo che salva le informazioni degli utenti nel file. Il metodo si sincronizza, come il precedente, sull'Oggetto lock, mentre se avessi usato ReentrantReadWriteLock il metodo avrebbe acquisito il lock di scrittura prima di iniziare la sezione di codice che accede e modifica la struttura dati UsersCache e scrive nel file JSON. Successivamente, viene creato un oggetto GSON, ed il metodo inizia a leggere il file JSON esistente utilizzando un oggetto FileReader. Il contenuto del file viene convertito in una lista di utenti e, per ogni utente nella cache, il metodo confronta se l'utente è già presente nella lista di utenti letta dal file e, se lo è, le informazioni vengono aggiornate con quelle presenti nella cache. Se l'utente non è presente, cioè è un nuovo iscritto, viene aggiunto alla lista e la lista aggiornata di utenti viene convertita in una stringa JSON e successivamente viene scritta nel file. Dopo aver scritto le informazioni degli utenti nel file JSON, la cache degli utenti viene svuotata ed infine, nel caso di Reentrant lock, il lock di scrittura viene rilasciato per consentire ad altri thread di accedere alle operazioni di lettura e scrittura.

isUserInFile(String username)

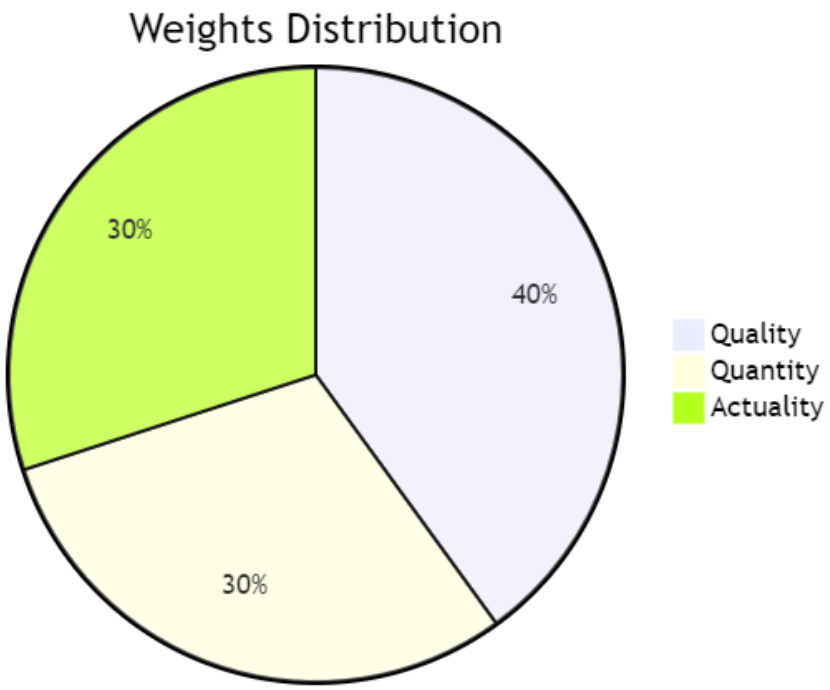
Il metodo isUserInFile, verifica se un utente specifico, identificato dal suo username, è presente nel file JSON degli utenti. Prende come parametro una stringa username e restituisce un valore booleano. Il metodo può lanciare un'eccezione IOException. Inizia con un blocco synchronized che utilizza un oggetto lock (oppure acquisisce la lock di lettura), per garantire che solo un thread alla volta possa eseguire il codice all'interno del blocco, prevenendo così le condizioni di gara. All'interno del blocco, legge l'intero file specificato dalla variabile user_path in una stringa: se la stringa è vuota il metodo restituisce immediatamente false, indicando che il file è vuoto. Altrimenti, la converte in un array JSON utilizzando JsonParser ed itera su ogni elemento dell'array, che rappresenta un utente. Per ogni oggetto, il metodo estrae il valore del campo "username". Se il nome utente estratto corrisponde al nome utente passato come parametro al metodo, il metodo restituisce true, indicando che l'utente è presente nel file. Se nessun utente corrispondente è stato trovato dopo aver esaminato tutti gli elementi dell'array JSON, il metodo restituisce false.

2.1.4 HotelService.java

La classe HotelService si occupa della gestione delle informazioni relative agli hotel ed il suo scopo principale è fornire il servizio per eseguire operazioni come la lettura e la scrittura di informazioni sugli hotel da un file JSON, l'aggiornamento delle classifiche degli hotel, l'invio di notifiche tramite UDP e la scrittura di recensioni. La classe utilizza diverse strutture dati per memorizzare le informazioni: `hotelCache` è una mappa concorrente che tiene traccia degli hotel in memoria, utilizzando l'ID dell'hotel come chiave, e consente un accesso efficiente alle informazioni degli hotel recentemente acceduti senza dover leggere continuamente il file.

La classe gestisce anche una `rankCache`, che mappa ogni città al miglior hotel

in termini di classifica. Questa cache è utile per individuare rapidamente l'hotel più valutato ed inviare una notifica agli utenti loggati qualora ci sia un cambiamento nelle prime posizioni. A tal proposito, per comprendere come vengono ordinati nella classifica gli hotel per ogni città, è necessario introdurre come opera l'algoritmo che ho pensato. Ogni volta che il file degli hotel viene aggiornato, viene calcolato anche il nuovo ranking, utilizzando il valore ottenuto dal metodo `calculateScore()` della classe Hotel.java, descritta in seguito. Questo `score` è basato su tre fattori: qualità, quantità e attualità delle recensioni.



- La qualità viene calcolata come la media delle valutazioni (rate) delle recensioni per quell'hotel.
- La quantità è il numero totale di recensioni per quell'hotel.
- L'attualità è calcolata in base alla differenza in minuti tra la data della recensione e la data attuale.

Il metodo, inizialmente, somma i punteggi sintetici (rate) di tutte le recensioni associate all'hotel. Questa somma è poi divisa per il numero totale di recensioni per ottenere una media ponderata. Successivamente, calcola la differenza in minuti tra la data della recensione e la data attuale, utilizzata per assegnare un punteggio di attualità alla recensione: più recente è la recensione, più alto sarà il punteggio. Questo punteggio è normalizzato dividendo per il numero totale di recensioni. Per il punteggio di quantità, viene considerato il numero di recensioni presenti. Infine, combina i tre punteggi calcolati utilizzando pesi specifici (WEIGHT_QUALITY, WEIGHT_QUANTITY, WEIGHT_ACTUALITY): il punteggio finale è la somma pesata di questi tre fattori.
La selezione dei pesi per i vari fattori è stata effettuata con l'obiettivo di mantenere un equilibrio tra di essi, evitando discrepanze eccessive. Si è scelto di attribuire un peso leggermente maggiore alla qualità, considerata di particolare rilevanza in questo contesto e allo stesso tempo, si è voluto dare un'adeguata considerazione anche all'aspetto temporale e quantitativo, ritenuti importanti per la valutazione complessiva.

I metodi presenti in questa classe sono:

- `public HotelService(String temp_file, String hotel_file, String review_file, String UDP_addr, String UDP_port)`
  Costruttore della classe HotelService.

- `private void sendUDPMessage(String message, String ipAddress, int port)`
  Invia un messaggio UDP a un indirizzo e una porta specifici.

- `private Review readReview(JsonReader reader) throws IOException, ParseException`

Legge una recensione dal lettore JSON fornito e restituisce un oggetto Review corrispondente.

- `private Ratings readRatings(JsonReader reader) throws IOException`
  Legge un Oggetto Ratings dal lettore JSON fornito e restituisce un oggetto Ratings corrispondente.

- `private Hotel readHotel(JsonReader reader) throws IOException, ParseException`
  Legge un hotel dal lettore JSON fornito e restituisce un oggetto Hotel corrispondente.

- `protected List<Hotel> searchAllHotels(String city) throws IOException`
  Cerca tutti gli hotel in una determinata città leggendo il file JSON degli hotel.

- `protected Hotel findHotelByNameAndCity(String hotelName, String city)`
  Trova un hotel per nome e città nella cache o leggendolo dal file JSON degli hotel.

- `protected Hotel searchHotel(String hotelName, String city) throws IOException`
  Trova un hotel per nome e città nella cache o leggendolo dal file JSON degli hotel.

- `protected void updateJsonFileAndRankCache()`
  Aggiorna il file JSON degli hotel e la cache delle classifiche.

- `protected void writeReview(User user, Hotel hotel, Review review) throws IOException`
  Scrive una recensione di un hotel, aggiornando il punteggio e la cache degli hotel.

updateJsonFileAndRankCache()

Il metodo più complesso e notevole di questa classe è `updateJsonFileAndRankCache()`. Questo metodo è responsabile del download dei dati dal file JSON e dell'aggiornamento del ranking degli hotel. Secondo le mie scelte implementative, ho deciso di gestire contemporaneamente l'aggiornamento del ranking e la persistenza dei dati relativi agli hotel in memoria all'interno di questa funzione. Per questo progetto, ho impostato un timeout di aggiornamento abbastanza breve, mirando a evidenziare il funzionamento del programma e ad inviare notifiche frequenti. In contesti più ampi e professionali, potrebbe essere opportuno separare queste due attività. Ad esempio, si potrebbe persistere la cache in memoria meno frequentemente, ma rivalutare più spesso il ranking.

Qui sotto, in breve, una spiegazione di come ho deciso di implementare questa funzionalità.
Viene inizializzata una variabile `changes` di tipo StringBuilder per registrare le modifiche e viene acquisito il blocco (synchronized) per garantire l'accesso sicuro alle risorse condivise. Anche qui, vale l'osservazione fatta precedentemente sulle Reentrant Lock. Viene utilizzato Gson per deserializzare il file contenente le informazioni sugli hotel in una lista di oggetti Hotel. Successivamente, il metodo gestisce la sincronizzazione tra la cache degli hotel e i dati letti dal file: vengono identificati gli hotel presenti nel file ma non nella cache e viceversa, aggiornando opportunamente la cache che viene usata come memoria temporanea. Il passo successivo coinvolge il calcolo del punteggio aggiornato per ciascun hotel. Gli hotel vengono quindi rag-

gruppati per città e ordinati in modo decrescente in base al punteggio e viene creata una mappa contenente l'hotel con il punteggio più alto per ogni città. Successivamente, il metodo verifica se ci sono state modifiche nelle classifiche rispetto alla cache delle classifiche precedente. Se viene rilevato un cambiamento, vengono registrate le modifiche in changes. Infine, aggiorna la cache delle classifiche, converte la lista di hotel aggiornata in una stringa JSON, sovrascrive il file JSON con i dati aggiornati e vengono svuotate le cache temporanee. Se sono state rilevate modifiche nelle classifiche, viene inviata una notifica tramite UDP con il messaggio contenente le informazioni sui cambiamenti registrati. Da sottolineare che, al primo avvio del server, essendo la cache vuota, verrà inviata la lista di tutti gli hotel in prima posizione. Il buffer in ricezione è stato dimensionato per ricevere, nel caso pessimo, una linea di aggiornamento per oggi città.

2.1.5 Classi di utilità

Le classi che il server usa per gestire gli hotel, gli utenti, le recensioni e tutto il sistema nel suo complesso sono:

- `AuthenticationException.java`: Eccezione per gestire errori di autenticazione
- `User.java`: Rappresenta un utente nel sistema.
- `Hotel.java`: Rappresenta un hotel nel sistema.
- `Review.java`: Rappresenta una recensione scritta da un utente per un hotel.
- `Ratings.java`: Contiene le valutazioni per vari aspetti di un hotel all'interno di una recensione.
- `Badge.java`: Rappresenta un distintivo assegnato a un utente in base alle sue azioni nel sistema.

- `Level.java`: Rappresenta il livello di un utente all'interno del sistema.

Hotel.java

La classe Hotel descrive gli hotel presenti nel sistema, utilizzando come appoggio le classi precedentemente descritte per gestire le recensioni.

Come già accennato, il metodo `calculateScore()` ricopre un ruolo importante nel sistema, ed abbraccia una filosofia ponderata nell'analizzare fattori chiave come la qualità delle recensioni, la loro quantità e la freschezza temporale.

L'implementazione di metodi come `equals()` e `hasChanged()` aggiunge uno strato di intelligenza, consentendo il confronto e la rilevazione di eventuali modifiche tra diverse istanze di hotel.

User.java

La classe User descrive un utente nel sistema, utilizzando come classi di appoggio Badge e Level. Il metodo di questa classe degno di nota è `public void updateBadge()`: il codice verifica il numero di recensioni dell'utente e assegna un livello e un distintivo in base a determinati intervalli di recensioni. Ad esempio, se il numero di recensioni è maggiore del valore associato a
Level.CONTRIBUTORE_SUPER, il metodo termina senza ulteriori assegnazioni. Altrimenti, vengono eseguiti controlli successivi per assegnare il livello e il distintivo appropriati.

2.2 File utilizzati

I file Json usati durante l'esecuzione di questo lavoro sono `Hotels.json`, fornito insieme al testo del progetto e `signedupUsers.json`.

Così è come appare il file Hotels.json dopo gli aggiornamenti:


```
{
  "id": 40,
  "name": "Hotel Milano 10",
  "description": "Un ridente hotel a Milano, in Via del tramonto, 35",
  "city": "Milano",
  "phone": "320-4042951",
  "services": [],
  "rate": 4.0,
  "ratings": {
    "cleaning": 4.0,
    "position": 4.0,
    "services": 4.0,
    "quality": 4.0
  },
  "reviews": [
    {
      "ratings": {
        "cleaning": 4.0,
        "position": 4.0,
        "services": 4.0,
        "quality": 4.0
      },
      "rate": 4.0,
      "date": "Mar 03, 2024, 6:17:10 PM",
      "user": "exe"
    }
  ],
  "Number_reviews": 1,
  "score": 2.199998
},
```


Così è come appare il file signedupUsers.json:


```
[{
  "username": "exe",
  "password": "ciaociao?",
  "badge": {
    "level": "CONTRIBUTORE_SUPER",
    "date": "Wed Feb 28 13:07:02 CET 2024"
  },
  "number_review": 11
},
```


Inoltre, sono stati utilizzati due file di configurazione, `client.properties` e `server.properties` per configurare rispettivamente client e server.

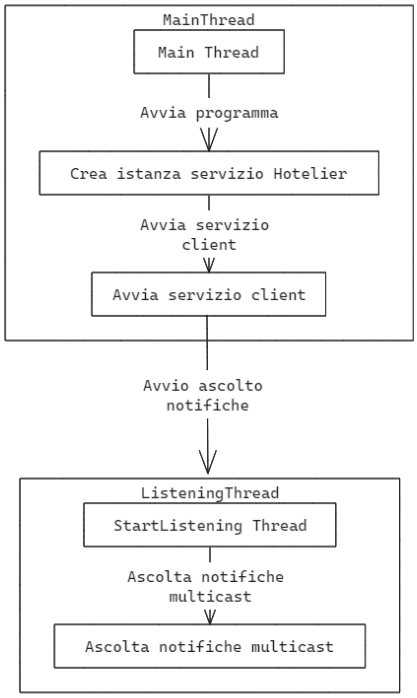
2.3 Client

Il client si appoggia sostanzialmente su due classi:

- `HOTELIERCustomerClientMain.java`
- `HOTELIERCustomerClientService.java`

2.3.1 HOTELIERCustomerClientMain.java

La classe `HOTELIERCustomerClientMain` rappresenta il punto di ingresso principale per avviare il client del sistema Hotelier. La sua responsabilità principale è leggere le configurazioni dal file `client.properties` tramite il metodo `readConfig()` e avviare il servizio client. Qui sotto un semplice schema che mostra i thread attivi durante l'esecuzione del client.



2.3.2 HOTELIERCustomerClientService.java

La classe è progettata per gestire la connettività con il server Hotelier attraverso una connessione TCP e per ricevere notifiche automatiche tramite multicast UDP. I dettagli di configurazione, come l'indirizzo del server, la porta, l'indirizzo e la porta UDP, sono forniti attraverso il costruttore della classe. E' responsabile anche di avviare il thread che si metterà in ascolto per ricevere le notifiche UDP.

I metodi della classe sono:

- `void printColored(String colorStart, String msg)`
  Stampa a schermo il messaggio utilizzando le sequenze di escape ANSI.

- `private void joinMulticastGroup(String UDP_addr, String UDP_port)`
  Unisce il socket multicast al gruppo specificato.

- `private void receiveMulticastMessage()`
  Riceve un messaggio multicast e lo stampa.

- `private void startListening()`
  Avvia un thread per ascoltare i messaggi multicast in modo asincrono.

- `public HOTELIERCustomerClientService (String serverAddress, int serverPort, String UDP_addr, String UDP_port)`
  Costruttore della classe HOTELIERCustomerClientService.

- `private int wait_response(BufferedReader in)`
  Attende e gestisce le risposte dal server.

- `private void signup(int action, BufferedReader in, PrintWriter out)`
  Gestisce il processo di registrazione dell'utente lato client.

- `private void auth(int action, BufferedReader in, PrintWriter out, String UDP_addr, String UDP_port)`
  Gestisce l'autenticazione dell'utente lato client.

- `private void search_hotel(int action, BufferedReader in, PrintWriter out)`
  Invia una richiesta di ricerca di un hotel al server e gestisce la risposta.

- `private void search_all_hotel(int action, BufferedReader in, PrintWriter out)`
  Invia una richiesta di ricerca di tutti gli hotel al server e gestisce la risposta.

- `private void insert_review(int action, BufferedReader in, PrintWriter out)`
  Invia una richiesta di inserimento di una recensione al server e gestisce la risposta.

- `private void readInputAndSendToServer(PrintWriter out)`
  Rileva l'input da console e lo invia al server.

- `protected void begin()`
  Inizia l'esecuzione del client Hotelier.

In base alla scelta dell'utente, il client chiamerà uno dei metodi sopra elencati per comunicare con il server e soddisfare la richiesta.

Questa classe fa uso di `ErrorMessages`, una classe statica che raccoglie costanti di stringa per i messaggi di errore associati alle diverse situazioni nel sistema Hotelier.

Per garantire un'esperienza utente più fluida durante l'uso della CLI, ho implementato un meccanismo per ritardare l'invio delle notifiche UDP fino a quando l'utente ha completato le operazioni in corso. Questo viene ottenuto attraverso una sin-

cronizzazione specifica sulla CLI, che evita interruzioni indesiderate durante l'esecuzione di comandi e operazioni. In questo modo, le notifiche UDP vengono inviate solo dopo che l'utente ha concluso le sue interazioni con la CLI.

Il metodo che gestisce la comunicazione con il server, rispettando il protocollo precedentemente descritto è `wait_response`. Questo metodo gestisce l'elaborazione delle risposte ricevute dal server dopo l'invio di una richiesta, leggendole da un BufferedReader e visualizzandole a console. Questa funzione gestisce in modo specifico la formattazione e la visualizzazione delle risposte, compresa la gestione di eventuali messaggi di errore, restituendo un codice che indica lo stato della risposta, utile per la logica di controllo del flusso nel client Hotelier.

Di seguito, degli esempi di flusso di esecuzione. Il client, quando avviato, mostra una schermata iniziale ed offre il menù delle opzioni. Se vengono scelte le opzioni che richiedono un utente loggato, si tenta di loggarsi con un utente inesistente, o si cerca di registrarsi con uno username già in uso, viene visualizzato un messaggio di errore e viene riproposto il menù. Scegliendo, ad esempio il login, il sistema richiede username e password, facendo i dovuti check. Il software risponde in inglese poiché il file json degli Hotel era redatto in questa lingua, ma non rappresenterebbe un problema tradurlo in italiano.

```
****************************************
*        Welcome to Hotelier! 😁         *
****************************************
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout,
 [8]Exit
3
Option Show Badge
[SERVER]: User is not logged in this session
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout,
 [8]Exit
6
Option Insert Review
[SERVER]: User must be logged in to post a review
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout,
 [8]Exit
1
Option Signup
[SERVER]: Insert a username
juju
[SERVER]: User already exist
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout,
 [8]Exit
2
Option Login
[SERVER]: Insert your username
juji
[SERVER]: This username doesn't exist
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout,
 [8]Exit
8
Option Exit
[SERVER]: Goodbye visitor 👋
```

```
****************************************
*        Welcome to Hotelier! 😁         *
****************************************
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout,
 [8]Exit
2
Option Login
[SERVER]: Insert your username
juju
[SERVER]: Insert your password
ciaociao?
[SERVER]: Access succeeded
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout,
 [8]Exit
7
Option Logout
[SERVER]: Logout. Goodbye juju👋
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout,
 [8]Exit
2
Option Login
[SERVER]: Insert your username
juju
[SERVER]: Insert your password
ciaociao
[SERVER]: Password not valid
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout,
 [8]Exit
8
Option Exit
[SERVER]: Goodbye visitor 👋
```

```
ju@Ju-PC:~/provaProgetto$ java -jar client.jar
****************************************
*        Welcome to Hotelier!          *
****************************************
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
2
Option Login
[SERVER]: Insert your username
exe
[SERVER]: Insert your password
ciaociao?
[SERVER]: Access succeeded
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
4
Option Search Hotel
[SERVER]: Insert Hotel
Hotel Milana 2
[SERVER]: Insert City
Milana
[SERVER]: Hotel "Hotel Milana 2" in Milana not found
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
```

Se scelta l'opzione `logout`, l'utente viene disconnesso ma la sessione mantenuta attiva, permettendo di utilizzare le funzionalità che non richiedono l'autenticazione, come la ricerca degli hotel. Così è come avviene la ricerca di un hotel (non è case sensitive):

```
****************************************
*        Welcome to Hotelier!          *
****************************************
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
4
Option Search Hotel
[SERVER]: Insert Hotel
Hotel Milano 10
[SERVER]: Insert City
Milano
[SERVER]:  Hotel Milano 10
| Un ridente hotel a Milano, in Via del tramonto, 35
| ------------------------------
|  Milano
| ------------------------------
|  320-4042951
| ------------------------------
|  Services
| ------------------------------
|  Overall Rating 5.0
| ------------------------------
|  Ratings
|  Cleaning: 5.0
|  Position: 5.0
| Services: 5.0
|  Quality: 5.0
| ------------------------------
|  Reviews
|  User: exe
|  Date: 06/03/2024 10:44:52
|  Overall Rating: 5.0
|  Ratings
|  Cleaning: 5.0
|  Position: 5.0
| Services: 5.0
|  Quality: 5.0
| ------------------------------
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
```

Un esempio di notifica inviata tramite UDP:

```
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
[Automatic Notification] 2024-03-07 22:49:36:
1st ranked Hotel in Udine is now Hotel Udine 10
```

Un esempio di inserimento di recensione:

```
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
6
Option Insert Review
[SERVER]: User must be logged in to post a review
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
2
Option Login
[SERVER]: Insert your username
juju
[SERVER]: Insert your password
ciaociao?
[SERVER]: Access succeeded
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
6
Option Insert Review
[SERVER]: Insert Hotel 🏨
Hotel Firenze 10
[SERVER]: Insert City 🏙️
Firenze
[SERVER]: Insert a synthetic review from 0 to 5 ⭐ for the hotel
4
[SERVER]: Insert rating from 0 to 5 for 1) 🧼 Cleaning 2) 📍 Position 3) 🛋️Services 4) 💯 Quality
4 4 5 4
[SERVER]: Review posted ✅
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
```

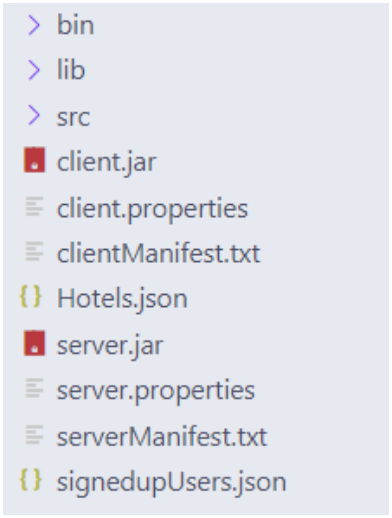
Un esempio di parametri errati per l'inserimento di recensione:

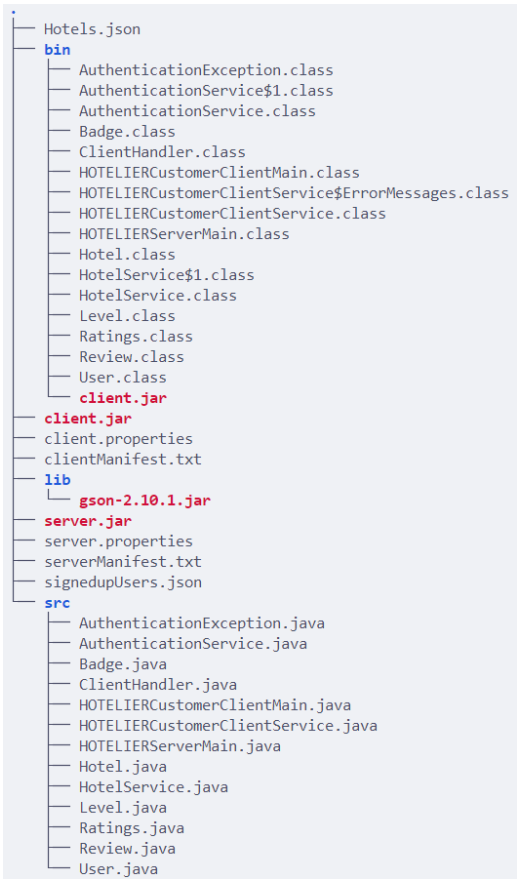
```
Please choose an option: [1]Signup, [2]Login, [3]Show badge, [4]Search Hotel, [5]Search all hotels, [6]Insert review, [7]Logout, [8]Exit
6
Option Insert Review
[SERVER]: Insert Hotel 🏨
Hotel milano 2
[SERVER]: Insert City 🏙️
milano
[SERVER]: Insert a synthetic review from 0 to 5 ⭐ for the hotel
6
[SERVER]: Please enter a rate between 0 and 5
[SERVER]: Insert a synthetic review from 0 to 5 ⭐ for the hotel
6
[SERVER]: Please enter a rate between 0 and 5
[SERVER]: Insert a synthetic review from 0 to 5 ⭐ for the hotel
5
[SERVER]: Insert rating from 0 to 5 for 1) 🧼 Cleaning 2) 📍 Position 3) 🛋️Services 4) 💯 Quality
4 5 7 4
[SERVER]: Please enter ratings between 0 and 5
[SERVER]: Insert rating from 0 to 5 for 1) 🧼 Cleaning 2) 📍 Position 3) 🛋️Services 4) 💯 Quality
4 5 4
[SERVER]: Not enough ratings provided. Please provide ratings for cleaning, position, services, and quality
[SERVER]: Insert rating from 0 to 5 for 1) 🧼 Cleaning 2) 📍 Position 3) 🛋️Services 4) 💯 Quality
4 5 4 5
[SERVER]: Review posted ✅
```

2.4 Guida per la compilazione ed esecuzione

Per questo progetto, ho svolto il lavoro di sviluppo sia su IntelliJ IDEA in ambiente Windows sia su Visual Studio Code su Ubuntu in Windows Subsystem for Linux (WSL). Mentre IntelliJ IDEA si occupava automaticamente della gestione delle librerie tramite Maven o Gradle e della compilazione del progetto, ho successivamente optato per la gestione manuale della compilazione e dell'importazione delle librerie migrando il progetto su Visual Studio Code.

Così è come appare il directory tree del progetto nel mio sistema:





Nella cartella `src` sono presenti tutti i file `.java`, nella cartella `lib` è presente la libreria esterna di gson e nella cartella `bin` andranno i file compilati. Nella cartella

principale ci sono i file `.properties`, i manifest e `.json`. Per compilare tutti i `.java` nella cartella `bin`, mi sono spostata nella cartella `src` e ho compilato con il comando:

```
1    javac -cp .:/home/ju/ProgettoLab3Chicca/lib/gson-2.10.1.jar -d ../bin *.
   java
```

Successivamente ho creato i due file manifest e nella cartella `bin` ho eseguito questi due comandi:

```
1    jar cfm ../client.jar ../clientManifest.txt HOTELIERCustomerClientMain.
   class HOTELIERCustomerClientService.class
```

```
1    jar cfm ../server.jar ../serverManifest.txt AuthenticationException.
   class AuthenticationService.class Badge.class ClientHandler.class
   HOTELIERServerMain.class Hotel.class HotelService.class Level.class
   Ratings.class Review.class User.class
```

Quindi questo programma può essere eseguito in due modi:

- Posizionandosi nella directory principale senza utilizzare i `.jar` :

  Per eseguire il server:

  `java -cp bin:lib/gson-2.10.1.jar HOTELIERServerMain`

  Per eseguire il client:
  `java -cp bin:lib/gson-2.10.1.jar HOTELIERCustomerClientMain`

- Posizionandosi nella directory principale e utilizzando i `.jar`:

Per eseguire il server:

```
java -jar server.jar
```

Per eseguire il client:

```
java -jar client.jar
```