HOTELIER: an HOTEL advIsor sERvice

Reti di Calcolatori - Modulo di Laboratorio III

Progetto di fine corso a.a. 2023/2024

Rebecca Rodi 579633

17 maggio 2024

Indice

1 Introduzione

Il progetto è stato realizzato usando la versione di JAVA 16, con JDK 16.0.2.

Sono previste due modalità d'uso possibili lato client, rispettivamente tramite CLI (Command Line Interface) o GUI (Graphical User Interface). Di default viene avviata la versione con GUI ma è comunque possibile scegliere l'interfaccia che si preferisce specificandola al momento dell'esecuzione da terminale. Il server, al contrario, prevede soltanto una modalità d'uso da terminale.

2 Scelte e assunzioni interpretative

- L'algoritmo per il calcolo dei punteggi degli hotel prevede l'utilizzo della funzione

$$f(t) = n + \sum_i \frac{R_i}{1+t} + U_i, \quad \text{dove } U_i = \begin{cases} \log_2 u_i, & \text{se } u_i > 0 \\ 0, & \text{altrimenti} \end{cases}$$

  Precisamente: n è il numero totale di recensioni pubblicate per l'hotel, R_i rappresenta il punteggio sintetico (rate) della recensione i-esima, scalato nell'intervallo $[0, 100]$, t il tempo trascorso dalla pubblicazione e u_i i voti utili ricevuti.

- È necessario essere iscritti al server e aver effettuato l'accesso per visualizzare il proprio distintivo, per poter pubblicare una recensione e di conseguenza visualizzare le recensioni pubblicate tramite il proprio profilo.

- Quando si fa riferimento alla ricerca, si sottintende che questa fornisca anche le informazioni relative ai risultati della ricerca in sé.

- Rispetto alle specifiche, è stato aggiunto un meccanismo per visualizzare tutte le recensioni pubblicate riferite a un hotel o pubblicate dall'utente che ne fa richiesta, il quale – solo in quest'ultimo caso – deve aver effettuato in precedenza l'accesso al proprio profilo.

- Rispetto alle specifiche, è stato aggiunto un meccanismo di votazione delle recensioni pubblicate per consigliare la recensione così votata. Non è necessario essere iscritti per votare una recensione: la votazione può avere luogo quindi sia in modalità guest, senza aver effettuato l'accesso al proprio profilo utente, sia in seguito al login. Non è possibile votare più di una volta la stessa recensione con la stessa modalità.

- In generale, non è possibile eliminare o modificare una recensione, rimuovere il proprio voto a una recensione o eliminare il proprio profilo utente, ma non è esclusa una possibile implementazione futura.

- Il server effettua il multiplexing dei canali mediante NIO, senza l'uso di un threadpool dedicato per la gestione delle richieste, dal momento che l'implementazione è stata realizzata pensando perlopiù a un utilizzo con un numero limitato di client collegati, richiedenti i servizi offerti dal server.

3 Entità

Di seguito l'elenco delle classi che rappresentano le entità in gioco.

3.1 `Hotel` package

3.1.1 `City`

La classe rappresenta le città selezionabili previste dal servizio: queste corrispondono ai capoluoghi delle regioni italiane.

Ogni città è caratterizzata dal proprio nome e dal prefisso corrispondente per il numero di telefono.

3.1.2 `Hotel`

La classe rappresenta gli hotel che fanno parte del servizio ed è caratterizzata dai seguenti attributi:

- ➤ l'id (identificatore univoco);
- ➤ il nome dell'hotel;
- ➤ l'indirizzo;
- ➤ la città in cui è situato;
- ➤ il tipo (o classe);
- ➤ il numero di telefono;
- ➤ una descrizione testuale;
- ➤ un insieme di servizi offerti;
- ➤ il punteggio sintetico (rate) complessivo[1];
- ➤ i punteggi per categoria (ratings) complessivi[2];
- ➤ la posizione (rank) nella classifica locale (local ranking);
- ➤ il punteggio necessario per determinare tale classifica.

Il metodo `updateAllRatings` si occupa di ricalcolare il rate e i ratings in seguito alla pubblicazione di una nuova recensione.

3.1.3 `Type`

La classe rappresenta i tipi di hotel possibili.

Ogni tipo è caratterizzato dal nome indicativo del tipo e il numero di stelle corrispondenti.

- ➢ 3: tre stelle.
- ➢ 4: quattro stelle.
- ➢ 4S: quattro stelle superior.
- ➢ 5: cinque stelle.
- ➢ 5S: cinque stelle superior.

3.1.4 `Address`

La classe rappresenta un insieme di indirizzi predefiniti in cui possono essere situati gli hotel.

3.1.5 `Description`

La classe rappresenta un insieme di descrizioni predefinite per gli hotel.

3.1.6 `Feature`

La classe rappresenta un insieme di servizi che possono essere offerti dagli hotel.

[1] Dato da tutte le recensioni pubblicate per l'hotel.
[2] Come sopra.

3.1.7 `Ratings`

La classe rappresenta l'insieme dei punteggi riferiti a un hotel o relativi a una recensione ed è caratterizzata dai seguenti attributi, rappresentanti le quattro categorie di voti:

- location;
- pulizia;
- servizio;
- rapporto qualità/prezzo.

Il metodo `updateRatings` si occupa di ricalcolare i ratings in seguito alla pubblicazione di una nuova recensione tramite il metodo `calculateAverage`.

3.2 `Database` package

3.2.1 `Review`

La classe rappresenta le recensioni pubblicate dagli utenti iscritti al server ed è caratterizzata dai seguenti attributi:

- l'id (identificatore univoco);
- l'autore;
- l'hotel di riferimento;
- il punteggio sintetico;
- i punteggi per categoria;
- la data;
- i voti utili.

Una volta creata e quindi pubblicata una recensione non è più possibile né modificarla né rimuoverla.

Il metodo `addUpvote` consente l'aggiunta di un voto utile, identificato dall'indirizzo IP del client, come guest, o dal nome utente con cui è stato effettuato l'accesso sul client, se vi è una corrispondente sessione attiva su quest'ultimo.

3.2.2 `User`

La classe rappresenta gli utenti iscritti al servizio ed è caratterizzata dai seguenti attributi:

- l'id (identificatore univoco);
- il nome utente;
- la password;
- le recensioni pubblicate dall'utente;
- i distintivi ottenuti in base alle recensioni pubblicate.

Il metodo `updateBadges` aggiorna i distintivi dall'utente, se è stata raggiunta la soglia minima di recensioni pubblicate per il livello corrispondente, in seguito alla pubblicazione di una recensione.

3.2.3 `Badge`

La classe rappresenta i distintivi (badges) che un utente può ottenere, corrispondenti ai cinque livelli di esperienza raggiungibili in base al numero di recensioni pubblicate.

Ogni badge è caratterizzato dal nome del livello, il livello e la soglia minima di recensioni pubblicate per ottenere il distintivo corrispondente.

- Recensore: livello 1, soglia minima = 1.

- ➢ Recensore esperto: livello 2, soglia minima = 2.
- ➢ Contributore: livello 3, soglia minima = 3.
- ➢ Contributore esperto: livello 4, soglia minima = 5.
- ➢ Contributore super: livello 5, soglia minima = 8.

3.3 `MyExceptions` package

Questo è il package contenente le eccezioni che vengono lanciate dal server durante l'esecuzione delle richieste inviate dai client.

- ➢ `InvalidCityException`: se la città non è valida, poiché non compresa tra quelle previste.
- ➢ `InvalidHotelException`: se l'hotel non è valido, poiché il nome non corrisponde a nessun hotel del servizio.
- ➢ `InvalidPasswordException`: se la password non è valida.
- ➢ `InvalidReviewException`: se la recensione non è valida, poiché non esiste nessuna recensione con quell'id.
- ➢ `InvalidScoreException`: se il punteggio non è valido, poiché non nel formato desiderato.
- ➢ `InvalidVoteException`: se il voto non è valido, poiché l'utente che desidera votare la recensione ne è anche l'autore.
- ➢ `ReviewAlreadyPostedException`: se una recensione per un dato hotel e con stesso autore è già stata pubblicata di recente.
- ➢ `UserAlreadyLoggedInException`: se l'utente ha già effettuato l'accesso (al momento dell'accesso).
- ➢ `UsernameAlreadyTakenException`: se il nome utente è già stato usato per un altro utente (al momento della registrazione).
- ➢ `UserNotLoggedInException`: se l'utente non ha effettuato l'accesso.
- ➢ `UserNotRegisteredException`: se non esiste nessun utente registrato con un dato nome utente.
- ➢ `WrongPasswordException`: se la password è errata, poiché non corrisponde a quella memorizzata per il profilo utente a cui si vuole accedere (al momento del login).

4 Server

Di seguito il funzionamento del server durante una sua normale esecuzione.

1. Inizialmente viene caricato il file di configurazione necessario per l'assegnazione dei parametri.
2. Successivamente, una volta inizializzate le strutture dati, vengono ripristinati i file di backup: in particolare, al primo avvio del server verranno ripristinati i file già definiti in precedenza, mentre per i successivi saranno scelti quelli elaborati durante l'ultima esecuzione.
   Nel dettaglio: prima si occuperà della struttura dati dedicata agli hotel e di conseguenza quella relativa ai local ranking, per poi passare a quella dedicata agli utenti, tramite cui definirà quella relativa alle recensioni per hotel.
3. In seguito, vengono attivati il gestore del salvataggio dei dati (`BackupHandler`), il gestore dell'aggiornamento delle classifiche locali e dell'invio delle rispettive notifiche (`RankingHandler`), tramite callback o con un messaggio sul gruppo di multicast specificato, e il gestore delle connessioni TCP verso i client (`TCPHandler`), in ascolto all'indirizzo e alla porta specificata.
4. Infine viene esportato l'oggetto remoto, nonché l'istanza del server stesso, creando così lo stub. Questo viene poi registrato nel registry, lanciato a sua volta sullo stesso host alla porta specificata, definendo così il collegamento tra il nome del servizio RMI `HOTELIER` e lo stub stesso.

A questo punto i gestori attivati proseguono la propria esecuzione in modo del tutto autonomo, richiedendo, a seconda dei casi, i servizi all'istanza del server. Principalmente sarà il `TCPHandler` per soddisfare le richieste dei client – dopo aver instaurato una connessione TCP con questi – a demandarli o in alternativa i client mediante il meccanismo RMI.

Quando un utente effettua l'accesso al proprio profilo sul server, l'utente viene registrato tra gli utenti online del servizio nella struttura dati corrispondente, in modo da garantire il corretto funzionamento delle operazioni che lo richiedono. Inoltre lo stub del client, su cui l'utente ha una sessione attiva e il cui riferimento è stato passato al server come parametro durante la registrazione al servizio di notifica tramite callback dopo l'accesso, viene memorizzato nella struttura dati dedicata e associato alle città per cui lo stesso ha registrato il proprio interesse: quando ci sarà un cambiamento nella classifica di una città, il server controllerà se il client abbia registrato il proprio interesse per quella città e in caso affermativo gli notificherà l'evento appena verificato.

4.1 Interfacce

4.1.1 `RMIHOTELIERServer`

Questa interfaccia estende l'interfaccia `Remote` ed è l'interfaccia dedicata alla gestione del meccanismo RMI (Remote Method Invocation) necessaria per la registrazione dell'utente e la registrazione del client al servizio di notifica.

Metodi dichiarati

- `register` per la registrazione dell'utente al server, con un nome utente e una password.
- `registerForCallback` per la registrazione tramite callback al servizio di notifica per le città di interesse specificate, al momento del login.
- `unregisterForCallback` per la cancellazione della propria registrazione al servizio di notifica, al momento del logout.

4.1.2 `HOTELIERServerInterface`

Questa interfaccia è l'interfaccia dedicata all'elaborazione delle richieste effettuate lato client.

Metodi dichiarati

- `login` per l'accesso dell'utente al proprio profilo, mediante nome utente e password. Per il login è necessario essere registrati al server con un profilo utente e non aver già effettuato l'accesso in precedenza.
- `logout` per la disconnessione dal profilo utente, specificato tramite il nome utente dello stesso. Per il logout è necessario aver effettuato in precedenza l'accesso al profilo utente.
- `searchHotel` per la ricerca della presenza di un hotel col nome indicato in una delle città previste. La ricerca prevede un match parziale, per cui verranno restituite le informazioni relative agli hotel che contengono il nome nel proprio e che sono situati nella città indicata.
- `searchAllHotels` per la ricerca totale in una delle città previste, per cui verranno restituite le informazioni relative a tutti gli hotel situati nella città indicata.
- `insertReview` per la pubblicazione di una recensione riferita a un hotel situato in una delle città previste. La recensione è identificata anche dal punteggio sintetico e dai punteggi per categoria. Per pubblicare una recensione è necessario aver effettuato l'accesso a un profilo utente.
- `showMyBadges` per la visualizzazione del distintivo dell'utente che ne fa richiesta, corrispondente all'ultimo ottenuto. Per visualizzare il distintivo ottenuto è necessario aver effettuato l'accesso al proprio profilo utente.

4.2 Implementazioni

4.2.1 `HOTELIERServer`

La classe implementa le due interfacce `RMIHOTELIERServer` e `HOTELIERServerInterface` ed estende la classe `RemoteServer`.

Si occupa della gestione delle strutture dati necessarie per il corretto funzionamento del servizio e della gestione del meccanismo RMI per l'invio delle notifiche tramite callback in seguito al verificarsi di un evento, rappresentando quindi il cuore dell'applicazione.

Metodi aggiuntivi

- `showAllReviews` per visualizzare le recensioni pubblicate riferite a un hotel di una città.
- `upvote` per votare una recensione come guest o come utente loggato. Per votare una recensione è necessario non essere l'autore della recensione stessa.
- `showMyReviews` per visualizzare le recensioni pubblicate dall'utente che ne fa richiesta. Per visualizzare le recensioni pubblicate è necessario aver effettuato l'accesso al proprio profilo utente.

4.3 Altre classi

4.3.1 `Message`

La classe rappresenta l'insieme dei messaggi, suddivisi in `Request` e `Reply`, che possono essere scambiati tra client e server e prevedono un formato predefinito in stile HTTP.

4.3.2 `Request`

La classe rappresenta i messaggi di richiesta che il client può inviare al server e ne prevede uno per ogni servizio offerto che può essere richiesto.

Si serve della classe `Method` per la definizione dei metodi corrispondenti e per il controllo sui parametri, definendo così il messaggio mediante il metodo `getMessage`.

Formato dei messaggi

- Login: `LOGIN username password`.
- Logout: `LOGOUT user:username`.
- Ricerca di un hotel: `SEARCH hotel city`.
- Ricerca totale: `SEARCHALL city`.
- Pubblicazione di una recensione: `INSERTREVIEW user:username hotel city score scores`.
- Visualizzazione del proprio distintivo: `SHOWMYBADGES user:username`.
- Visualizzazione delle recensioni pubblicate riferite a un hotel: `SHOWREVIEWS hotel city`.
- Votazione di una recensione: `UPVOTE #review`.
- Visualizzazione delle proprie recensioni pubblicate: `SHOWMYREVIEWS user:username`.

4.3.3 `Reply`

La classe rappresenta gli elementi che caratterizzano i messaggi di risposta che il server invia al client, tra cui `Status`, `Info`, `Error` e `Session`.

4.3.4 `Status`

La classe rappresenta i due possibili esiti in seguito all'elaborazione dell'operazione richiesta dal client: `SUCCESS` o `ERROR`.

4.3.5 `Info`

La classe rappresenta una descrizione informativa dell'esito dell'operazione richiesta in caso di successo, ovvero nel caso in cui il server abbia elaborato la richiesta senza errori durante la sua esecuzione.

- `OK`: richiesta andata a buon fine.
- `Done`: richiesta completata.
- `Failure`: richiesta non completata, seguito da informazioni aggiuntive relative.
- `Found`: richiesta completata, seguito dagli elementi richiesti trovati.
- `More`: richiesta andata a buon fine, seguito da informazioni aggiuntive relative.
- `NotFound`: richiesta completata, elementi richiesti non trovati.

4.3.6 `Error`

La classe rappresenta una descrizione informativa dell'esito dell'operazione richiesta in caso di errore, ovvero nel caso in cui il server abbia riscontrato degli errori durante l'elaborazione della richiesta.

- `LoginError`: errore durante il login.
- `LogoutError`: errore durante il logout.
- `SearchError`: errore durante la ricerca di un hotel.
- `SearchAllError`: errore durante la ricerca totale.
- `InsertReviewError`: errore durante la pubblicazione della recensione.
- `ShowMyBadgesError`: errore durante la visualizzazione del proprio distintivo.
- `ShowReviewError`: errore durante la visualizzazione delle recensioni di un hotel.
- `ShowMyReviewsError`: errore durante la visualizzazione delle proprie recensioni.
- `BadRequestError`: errore durante l'elaborazione della richiesta (perché non conforme).
- `SessionError`: errore di sessione.

La classe non comprende quegli errori derivanti dalle eccezioni che vengono lanciate durante l'esecuzione della richiesta, dai metodi del server che se ne occupano. Tuttavia, queste vengono trattate allo stesso modo degli errori appena descritti: prevedono infatti un nome identificativo corrispondente al nome della classe di cui sono istanza, seguite da una descrizione testuale caratterizzante l'eccezione stessa.

4.3.7 `Session`

La classe rappresenta gli elementi che consentono al server di distinguere tra i diversi tipi di sessione attiva sul client: `GUEST`, corrispondente a una sessione come ospite (quindi non propriamente attiva), e `SESSION`, corrispondente a una sessione da utente loggato. A questi viene aggiunto `USER`, necessario per fornire ulteriori indicazioni sull'utente che fa richiesta dei servizi, quando è obbligatorio specificarlo.

5 Client

Di seguito il funzionamento del client durante una sua normale esecuzione.

1. All'inizio viene caricato il file di configurazione necessario per l'assegnazione dei parametri.
2. Una volta inizializzate le strutture dati, viene aperta una socket channel all'indirizzo IP e porta specificati, con cui il client si collega al server e tramite cui invierà in seguito le richieste.
3. Dal registry attivato sul server viene reperito poi lo stub associato al servizio remoto per le richieste RMI e viene esportata l'istanza del client per un eventuale uso del meccanismo delle callback.

4. Alla fine viene creato il gestore del multicast (`MulticastClient`), che si occuperà di mostrare all'utente del servizio i messaggi in arrivo sul gruppo a cui si sarà unito.

A questo punto l'utente – che non ha ancora effettuato il login – ha a disposizione un insieme limitato di comandi disponibili, tra cui:

- `register` per registrarsi,
- `login` per effettuare l'accesso,
- `search` per cercare un hotel in una città,
- `searchall` per trovare tutti gli hotel di una città,
- `showreviews` per visualizzare tutte le recensioni di un hotel,

a cui vanno aggiunti `upvote` per votare una recensione, eseguibile solo dopo aver usato il comando `showreviews`, e i comandi di aiuto o uscita dal programma. Qualunque altro comando sarà dichiarato non valido, a esclusione di quelli previsti disponibili per un utente con sessione attiva sul client che, a ogni modo, non sarà possibile eseguire. Solo se l'utente effettua l'accesso sarà possibile per lui usufruire dei seguenti comandi aggiuntivi:

- `logout` per disconnettersi dal profilo utente con sessione attiva,
- `insertreview` per pubblicare una recensione dal profilo utente con sessione attiva,
- `showmybadges` per mostrare l'ultimo badge dell'utente con sessione attiva,
- `showmyreviews` per visualizzare le recensioni pubblicate dall'utente con sessione attiva,

mentre i comandi `register` e `login` non potranno più essere eseguiti.

A seconda del metodo richiesto, il client invierà quindi un messaggio di richiesta al server, formattato secondo le indicazioni della classe `Request`, rimanendo in attesa della risposta. Una volta ricevuto il messaggio di risposta, questo viene suddiviso in modo tale da poterne effettuare il parsing, per poi comunicare l'esito dell'operazione all'utente.

La registrazione di un nuovo utente comporta l'immediato invio al server della richiesta di login al profilo utente appena registrato.

La registrazione al servizio di notifica tramite callback e l'iscrizione al gruppo di multicast vengono effettuate esclusivamente dopo la richiesta di accesso – andata a buon fine – a un profilo utente: la prima è comunque facoltativa, dunque è possibile saltare il passaggio o non scegliere alcuna città di interesse, mentre la seconda avviene in automatico e non vi è possibilità di scelta.

La registrazione di un utente e la registrazione del client al servizio di notifica tramite callback vengono eseguite autonomamente dal client mediante il meccanismo RMI, invocando direttamente il metodo dichiarato nell'interfaccia `RMIHOTELIERServer`. A ogni logout la struttura dati dedicata ai local ranking, relativi alle città di interesse selezionate dall'utente al momento dell'accesso, viene ripristinata, per poi essere inizializzata nuovamente al login successivo con la nuova selezione.

Una volta inviata effettuata la disconnessione dal profilo utente con sessione attiva sul client – andata a buon fine – il client si disiscrive dal servizio di notifica tramite callback e dal gruppo di multicast, in modo tale da non ricevere più alcun tipo di messaggio. Nel caso in cui, sempre durante la stessa esecuzione del programma, l'utente effettuasse un nuovo accesso a un profilo qualsiasi, si ripetono i passaggi appena descritti.

5.1 Interfacce

5.1.1 `CallbackHOTELIERClient`

Questa interfaccia è l'interfaccia dedicata alla gestione del meccanismo delle callback per il servizio di notifica degli eventi.

Metodi dichiarati

- `notifyEvent` per la notifica dell'aggiornamento della classifica degli hotel in una delle città per cui il client ha registrato il proprio interesse.

5.2 Classi astratte

5.2.1 `HOTELIERClient`

La classe astratta implementa le due interfacce `Runnable` e `CallbackHOTELIERClient` ed estende la classe astratta `RemoteObject`.

Si occupa della gestione dell'interazione tra l'utente che fa uso del servizio e il server, per cui invia le richieste a quest'ultimo ed elabora le risposte ottenute, consentendo dunque la fruizione dei servizi che questo offre all'utente.

Metodi aggiuntivi

- `sendRequest` per l'invio di una richiesta da parte del client al server e la ricezione della conseguente risposta da parte di questo.
- `formatPassword` per non trasmettere in chiaro la password inserita dall'utente.

5.3 Implementazioni

5.3.1 `HOTELIERCustomerClient`

La classe estende la classe astratta `HOTELIERClient`.

Si occupa della gestione dell'interazione tra l'utente che fa uso del servizio e il server tramite CLI.

Metodi

- `register` per registrare un profilo utente al server, tramite nome utente e password inseriti. Per questa operazione è necessario non aver già effettuato l'accesso a un profilo utente sul client.
- `login` per effettuare l'accesso al profilo utente, tramite nome utente e password inseriti. Per questa operazione è necessario non aver già effettuato l'accesso a un profilo utente sul client.
- `registerForCallback` per registrare il client al servizio di notifica degli eventi tramite callback, consentendo la selezione delle città di interesse per cui si desidera ricevere aggiornamenti.
- `logout` per disconnettere dal profilo utente con sessione attiva sul client. Per questa operazione è necessario aver effettuato in precedenza l'accesso al profilo utente.
- `searchHotel` per cercare l'hotel inserito nella città inserita.
- `searchAllHotels` per trovare tutti gli hotel presenti nella città inserita.
- `insertReview` per pubblicare una recensione per l'hotel inserito nella città inserita, con il punteggio sintetico e i punteggi per categoria specificati dall'utente. Per questa operazione è necessario aver effettuato in precedenza l'accesso a un profilo utente.
- `showMyBadges` per mostrare l'ultimo distintivo ottenuto dall'utente che ha una sessione attiva sul client. Per questa operazione è necessario aver effettuato in precedenza l'accesso al profilo utente.
- `showAllReviews` per mostrare le recensioni pubblicate riferite all'hotel specificato nella città specificata.
- `upvote` per votare la recensione indicata, sia come guest sia come utente loggato con sessione attiva sul client.
- `showMyReviews` per mostrare le recensioni pubblicate dall'utente che ha una sessione attiva sul client. Per questa operazione è necessario aver effettuato in precedenza l'accesso al profilo utente.

5.3.2 `GUIHOTELIERClient`

La classe estende la classe astratta `HOTELIERClient`.

Si occupa della gestione dell'interazione tra l'utente che fa uso del servizio e il server tramite GUI.

La classe prevede un comportamento analogo a quello della classe `HOTELIERCustomerClient`, a eccezione per la visualizzazione dei contenuti che in questo caso viene demandata all'interfaccia grafica implementata.

5.4 Altre classi

5.4.1 `Command`

La classe rappresenta i comandi che possono essere eseguiti sul client[3] e ne prevede uno per ogni metodo `Method`, ovvero per ogni servizio offerto che può richiedere al server, con una descrizione testuale e con i parametri richiesti da mostrare[4].

- `LOGIN`: comando per il login.
- `LOGOUT`: comando per il logout.
- `SEARCH`: comando per la ricerca di un hotel.
- `SEARCHALL`: comando per la ricerca totale in una città.
- `INSERTREVIEW`: comando per la pubblicazione della recensione.
- `SHOWMYBADGES`: comando per la visualizzazione del proprio distintivo.
- `SHOWREVIEWS`: comando per la visualizzazione delle recensioni riferite a un hotel.
- `UPVOTE`: comando per la votazione di una recensione.
- `SHOWMYREVIEWS`: comando per la visualizzazione delle proprie recensioni.
- `HELP`*: comando per ottenere aiuto.
- `CANCEL`*: comando per annullare.
- `QUIT`*: comando per uscire dall'applicazione.

6 Strutture dati e sincronizzazione

6.1 Lato server

Le strutture dati lato server prevedono tutte delle `HashMap` sincronizzate, restituite dalla `Collections.synchronizedMap()`.

- `users` è la struttura dati dedicata agli utenti registrati e consiste in una `HashMap<String, User>`, le cui entry sono coppie di nome utente e profilo utente corrispondente.

  Si aggiorna esclusivamente in corrispondenza di una nuova registrazione. L'accesso alla struttura dati in scrittura viene effettuato direttamente mediante l'invocazione via RMI del metodo da parte dei client collegati: l'accesso è dunque prevalentemente in lettura sia da parte del `TCPHandler`, tramite l'invocazione dei metodi del server che ne prevedono l'uso, sia da parte del `BackupHandler`, quando deve essere eseguito il backup della struttura dati. Per questi motivi il metodo per la registrazione è `synchronized` così come la struttura stessa è stata

[3] Alcuni di questi (*) è possibile eseguirli soltanto mediante interfaccia testuale.

[4] Questo è possibile se il client è stato avviato con CLI.

resa sincronizzata, in modo da garantire la massima consistenza durante le letture, soprattutto durante quella per il backup.[5]

➢ `hotels` è la struttura dati dedicata agli hotel del servizio e consiste in una `HashMap<String, Hotel>`, le cui entry sono coppie di nome dell'hotel e corrispondente oggetto.

Non si aggiorna mai, sono piuttosto gli oggetti di tipo `Hotel` contenuti in essa ad essere modificati con l'aggiornamento dei diversi punteggi quando richiesto[6].
L'accesso alla struttura dati avviene quindi esclusivamente in lettura, ma per evitare inconsistenze relative ai valori è stata resa sincronizzata.

➢ `reviews` è la struttura dati dedicata alle recensioni e consiste in una `HashMap<String, Review>`, le cui entry sono coppie nome dell'hotel e lista di recensioni pubblicate per il corrispondente oggetto di tipo `Hotel`.

Si aggiorna ad ogni pubblicazione di recensione, inoltre vengono modificati anche gli oggetti di tipo `Review` contenuti in essa ad ogni richiesta di aggiunta di voto utile[7].
L'accesso – sia in scrittura che in lettura – avviene quindi prevalentemente da parte del `TCPHandler`, in seguito alla ricezione di una richiesta da parte di un client e tramite l'invocazione dei metodi del server che ne prevedono l'uso, ma anche da parte del `RankingHandler` per l'aggiornamento delle classifiche locali. Per questi motivi la struttura è stata resa sincronizzata, in modo da garantire la massima consistenza durante le letture, soprattutto durante quella necessaria al calcolo dei punteggi per le classifiche locali.

➢ `localRanking` è la struttura dati dedicata alle classifiche locali e consiste in una `HashMap<String, List<Hotel>>`, le cui entry sono coppie città e lista di hotel situati nella città in ordine di ranking non crescente. La struttura viene usata anche per le ricerche, il cui meccanismo sfrutta al meglio l'organizzazione della stessa.

Si aggiorna esclusivamente in corrispondenza dell'aggiornamento delle classifiche locali, demandata al `RankingHandler`, mentre l'accesso da parte del `TCPHandler` avviene esclusivamente in lettura. Per questi motivi la struttura è stata resa sincronizzata, in modo da poter garantire la massima consistenza durante le letture necessarie, tramite l'invocazione dei metodi che ne prevedono l'uso, per le richieste provenienti dai client collegati.

➢ `clients` è la struttura dati dedicata ai client che si sono registrati per il servizio di notifica e consiste in una `HashMap<HOTELIERClientInterface, List<String>>`, le cui entry sono coppie di client e lista di città di interesse selezionate dallo stesso.

Si aggiorna in seguito a ogni login e logout, mediante l'invocazione via RMI dei metodi dedicati da parte del client, e consente il corretto invio delle notifiche.
L'accesso in lettura alla struttura dati avviene per l'invio delle notifiche ai client in seguito all'aggiornamento della classifica locale di una città, mentre quello in scrittura durante la registrazione o la cancellazione di tale registrazione del client che lo richiede al servizio di notifica. Precisamente, l'accesso in sola lettura viene effettuato dal `RankingHandler`, mentre quello in sola scrittura viene effettuato direttamente mediante l'invocazione dei rispettivi metodi da parte dei client collegati. Per questi motivi tali metodi sono `synchronized` così come la struttura stessa è stata resa sincronizzata.[8]

[5] L'accesso ai valori mappati dalla struttura `users` invece è comunque thread safe dal momento che gli oggetti di tipo `User` prevedono dei meccanismi di sincronizzazione – mediante la keyword `synchronized` – sui metodi che leggono o scrivono le stesse strutture dati. A ogni modo, tale pratica non è strettamente necessaria per quanto implementato finora.

[6] Per questo motivo i metodi della classe riferiti alla struttura dati che ne fa uso sono `synchronized`.

[7] Come sopra.

[8] In realtà potrebbe essere sufficiente la sola dichiarazione dei metodi `synchronized`, essendo la struttura acceduta proprio mediante tali metodi, ma per garantire comunque l'atomicità delle singole operazioni, in particolare di inserimento e rimozione, è stato scelto di lasciare anche l'intera struttura sincronizzata.

- ➢ `onlineUsers` è la struttura dati dedicata agli utenti online e consiste in una `List<String>`, in cui ciascun elemento corrisponde al nome utente dell'utente che ha effettuato l'accesso al proprio profilo.

  Si aggiorna a ogni login e logout, inoltre viene usata per garantire la corretta esecuzione delle operazioni che richiedono una sessione attiva per il profilo utente corrispondente. L'accesso – sia in lettura che scrittura – alla struttura dati `onlineUsers` avviene esclusivamente da parte del `TCPHandler`, in seguito alla ricezione di una richiesta da parte di un client e tramite l'invocazione dei metodi del server che ne prevedono l'uso.[9]

Tutte le strutture dati prevedono inoltre un accesso in scrittura mediante il costruttore del server, per l'inizializzazione e il ripristino delle stesse con i dati dei diversi salvataggi.

6.2 Lato client

L'unica struttura dati che viene usata lato client è la `HashMapHashMap<String, List<String>>` per il `localRanking`, necessaria a tenere traccia delle classifiche nelle città di interesse, la quale viene aggiornata in seguito alla ricezione della notifica tramite callback che comunica l'evento appena verificato e in fase di login e logout.

L'accesso in scrittura alla struttura avviene per l'inizializzazione, nel momento in cui sul client è attivata una sessione da utente loggato, e per il ripristino, nel momento in cui viene fatto il logout dal profilo corrispondente. Inoltre si ha un accesso, sia in lettura che in scrittura, quando il client riceve una notifica dal server. Non ha bisogno di sincronizzazione, essendo tali metodi `synchronized`.

7 Threads

Di seguito i threads che vengono attivati durante l'esecuzione dell'applicazione, lato server e lato client, in aggiunta ai rispettivi `main` threads attivati all'avvio.

7.1 Lato server

I thread lato server vengono attivati alla costruzione dell'istanza in uso del server stesso.

7.1.1 `BackupHandler`

Si occupa di effettuare periodicamente il backup delle strutture dati principali, rispettivamente quella dedicata agli hotel e quella dedicata agli utenti iscritti al servizio.

Tramite l'istanza del server che lo ha in precedenza attivato, serializza una copia aggiornata delle strutture dati, persistendole in un file dedicato ciascuno e specificato come parametro del metodo corrispondente. I due file così ottenuti prevedono un riferimento nel nome alla struttura dati di cui contengono le informazioni, seguita dalla data e dall'ora in cui il thread, e di conseguenza il server, è stato attivato.

La scrittura dei dati sui file viene perpetrata a ogni intervallo di tempo specificato, sovrascrivendo quindi il contenuto scritto in precedenza. Usa il formato `JSON`, tramite la libreria `Jackson`.

7.1.2 `RankingHandler`

Si occupa di effettuare periodicamente l'aggiornamento delle classifiche locali e di notificare i client quando la classifica locale di una città cambia, inviando un messaggio per ogni città in cui cambia il primo classificato.

[9] Per questi motivi e per quanto implementato finora non ha bisogno di meccanismi di sincronizzazione, ma è stato scelto di renderla comunque sincronizzata pensando a eventuali altri thread per la gestione delle richieste dei client che potrebbero farne uso.

Tramite l'istanza del server che lo ha in precedenza attivato, aggiorna le classifiche di ogni città eseguendo l'algoritmo dedicato per il calcolo del punteggio, mediante il quale viene quindi determinata la posizione corrispondente mantenendo un ordine non crescente.

Usa la struttura dati `localRanking` per memorizzare la classifica degli hotel in ogni città. Durante l'esecuzione quindi confronta la propria struttura dati, riferita al precedente aggiornamento, con la nuova ottenuta dall'ultimo aggiornamento: per ogni città verifica se la corrispondente classifica locale sia rimasta invariata e in caso contrario invia una notifica tramite callback, specificando città e classifica aggiornata, ai client iscritti al servizio, i quali hanno registrato il proprio interesse per quella stessa città. Successivamente se è cambiato anche l'hotel in prima posizione, comunica l'evento inviando un pacchetto UDP, contenente tutte le informazioni relative, sul gruppo di multicast dedicato. [10]

L'aggiornamento delle classifiche locali viene perpetrato a ogni intervallo di tempo specificato, calcolando per ogni città i nuovi punteggi degli hotel con l'algoritmo implementato e ordinando gli stessi in ordine non crescente di punteggio, in modo da definire il posizionamento di ciascun hotel all'interno del proprio local ranking.

7.1.3 `TCPHandler`

Si occupa della gestione delle connessioni con i client e dell'elaborazione delle richieste.

Gestisce le connessioni grazie a un selettore che effettua il multiplexing dei canali pronti per le operazioni di interesse con cui si sono registrati.

Mediante il costruttore, apre una server socket channel in ascolto alla porta indicata, così da ricevere le richieste di connessione da parte dei client. Accettando la richiesta di connessione, poiché il canale risulta appunto pronto per una operazione di accettazione, viene creata la socket channel dedicata alla comunicazione vera e propria: la connessione è così stata instaurata e il client può finalmente inviare le richieste al server.

Quando l'handler riceve una richiesta, seleziona il canale (cioè il client) corrispondente da cui questa proviene – il canale viene selezionato perché risulta pronto per una operazione di lettura – e la elabora attraverso l'istanza del server. A questo punto quindi, ottenuto il risultato dell'operazione relativa, definisce la risposta da inviare e registra il canale come pronto per una operazione di scrittura, con il messaggio come attachment. In questo modo, concluso il blocco dedicato alla lettura (`handleRead`) ed essendo il canale risultato nuovamente pronto per una diversa operazione, si potrà provvedere all'invio della risposta al client tramite il blocco dedicato alla scrittura (`handleWrite`), dopo il quale il canale è registrato nuovamente per una operazione di lettura così da poter essere selezionato ancora in seguito.

Le socket channel dedicate ai client vengono definite non-bloccanti prima di essere registrate al selettore per l'operazione di interesse. L'attachment contiene l'ultimo messaggio inviato dal server al client, preceduto da una sezione che specifica il tipo sessione in uso al momento: come ospite, `guest:` seguito dall'indirizzo IP del client, o da utente loggato, `session:` seguito dal nome utente corrispondente al profilo.

In generale, come già specificato, si prevede un numero limitato di client e i metodi non richiedono lunghi tempi di esecuzione, dunque non vi è una eccessiva limitazione della responsiveness o potenziale concorrenza.

7.2 Client[11]

7.2.1 `HOTELIERClient`[12]

Si occupa dell'interazione tra l'utente che fa uso del servizio e il server.

[10] Non è necessario per il server essere iscritto al gruppo di multicast ed aver istanziato quindi una `MulticastSocket` per poter spedire questo un messaggio, ma è sufficiente una `DatagramSocket`, aperta con il costruttore, tramite cui vengono inviati i `DatagramPacket` che contengono nell'intestazione l'indirizzo IP del gruppo di multicast.

[11] Per quanto riguarda il client attivato in modalità GUI, bisogna tener conto anche dell'insieme di thread che vengono attivati per la gestione dell'interfaccia grafica, i quali non vengono direttamente gestiti dall'implementazione realizzata e che sono essenziali per il corretto funzionamento degli elementi che la compongono e per l'interazione dell'utente con questi.

[12] Con questo si fa riferimento in generale anche alle sottoclassi che vengono usate.

Viene attivato durante l'esecuzione principale del `main` thread. Invia le richieste al server ed elabora le risposte così ottenute.

7.2.2 `MulticastClient`

Si occupa della gestione del multicast e dei messaggi in arrivo sul gruppo di multicast dedicato.

Mediante il costruttore, istanzia la `MulticastSocket` necessaria per la ricezione dei messaggi sul gruppo di multicast, senza però unirsi a esso.

Viene attivato al primo login effettuato dall'utente, dopo essersi unito al gruppo di multicast per la prima volta. Successivamente, se l'utente effettua il logout ma non chiude l'applicazione, lascia il gruppo di multicast ma prosegue la propria esecuzione, finché non termina il programma. Quando iscritto al gruppo, comunica all'utente il contenuto dei pacchetti ricevuti. Nel caso in cui l'utente effettui nuovamente l'accesso, si unisce ancora una volta al gruppo di multicast, comportandosi poi come sopra.

8 Persistenza dei dati

La persistenza dei dati viene garantita dal `BackupHandler`, il quale si occupa, come già anticipato, del backup periodico delle strutture dati relative agli hotel e agli utenti del servizio.

Sono previsti quindi due file distinti in formato `JSON`, che contengono i dati serializzati, articolati in array di oggetti, rispettivamente di tipo `Hotel` e di tipo `User`.

9 Interfaccia grafica

L'interfaccia grafica implementata è stata pensata con una grafica minimal tipo web.

Il client rimane quasi sostanzialmente invariato, eccetto che per la visualizzazione dei contenuti tramite GUI e non più su terminale. Questo ha richiesto una ridefinizione dei metodi presenti nella versione con CLI, dal momento che l'invocazione degli stessi viene chiamata direttamente dal frame principale dell'applicazione e non più all'interno della classe del client.

L'interfaccia è articolata sulla base di un frame principale, il `CustomFrame`, che consente l'alternarsi delle diverse pagine definite: la home page (`HomePanel`, 1a), la pagina con vista da guest (`GuestPanel`, 1b) e la pagina con vista da utente loggato (`SessionPanel`), cioè con una sessiona attiva.

Il `GuestPanel` e il `SessionPanel` sono estensioni del `CustomPanel`, poiché prevedono un funzionamento di base simile: le due pagine infatti sono entrambe articolate in un insieme di schede. In aggiunta, nel primo sono presenti anche un pulsante per tornare alla home page e uno per effettuare il login, mentre nel secondo è presente un pulsante per effettuare il logout oltre che una etichetta dedicata per la visualizzazione del nome utente. Il `GuestPanel` tuttavia prevede unicamente la scheda dedicata alla ricerca[13] (`SearchTab`, 1b), al contrario il `SessionPanel` comprende anche:

- una scheda home personalizzata (`HomeTab`, 1c), tramite la quale è possibile inoltre pubblicare direttamente una recensione[14];
- la scheda per la visualizzazione del badge (`BadgeTab`, 1d);
- la scheda per la visualizzazione delle recensioni pubblicate (`ReviewsTab`, 1c).

A questi elementi vanno aggiunti i frame in modalità dialog, attivabili mediante i pulsanti dedicati:

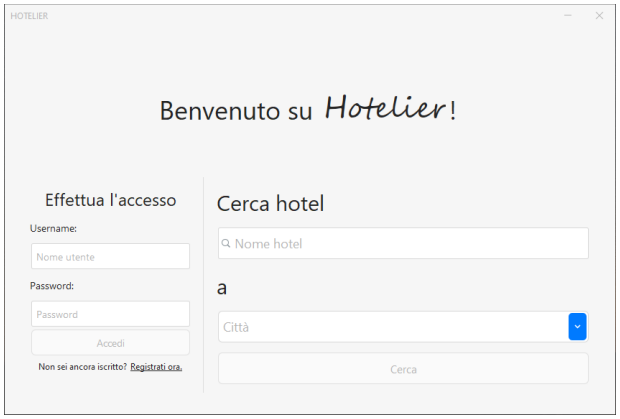
[13] Nonostante quanto appena scritto, questo è stato fatto per eventuali aggiunte e servizi usufruibili anche senza una sessione attiva, oltre alla ricerca.

[14] È stato scelto di non implementare una scheda dedicata alla sola pubblicazione di una recensione per evitare troppa ridondanza e confusione. Eventualmente si può modificare banalmente la scheda di home, così da inserire una scheda apposita, ma questo farebbe perdere la customizzazione cercata: piuttosto si potrebbe pensare a una sostituzione con un altro elemento personalizzato relativo a funzionalità o servizi aggiuntivi (che al momento non sono ancora stati implementati).

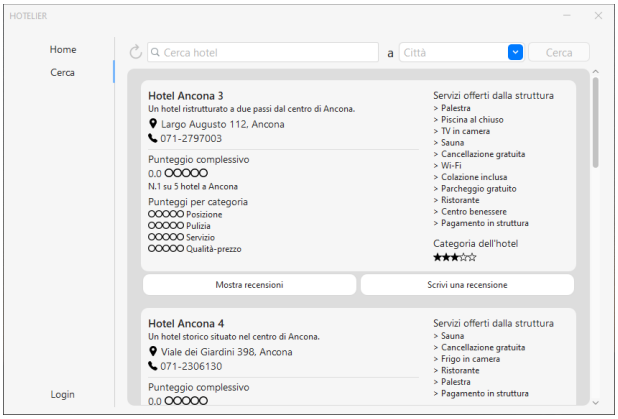
- per la registrazione (`RegistrationDialog`, 2c);
- per il login (`LoginDialog`, 2a);
- per la selezione delle città di interesse per la ricezione delle notifiche (`NotificationDialog`, 2b);
- per la pubblicazione di una recensione (`ReviewPostDialog`, 2e);
- per la visualizzazione delle recensioni di un hotel (`ReviewsDialog`, 2d).

Per quanto riguarda le notifiche e la segnalazione di eventuali errori, questi vengono tutti mostrati attraverso dei dialog personalizzati, riportanti le informazioni necessarie da comunicare all'utente.

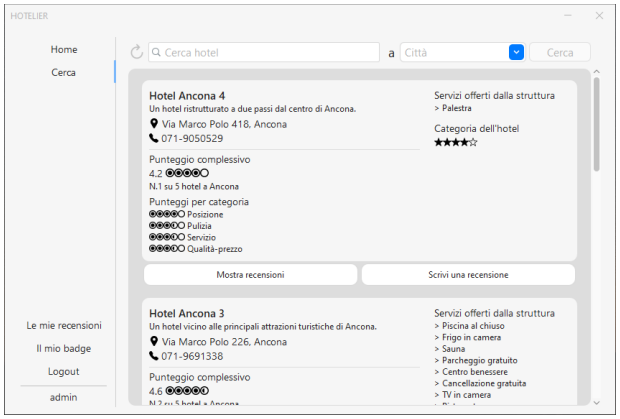
(a) `HomePanel`



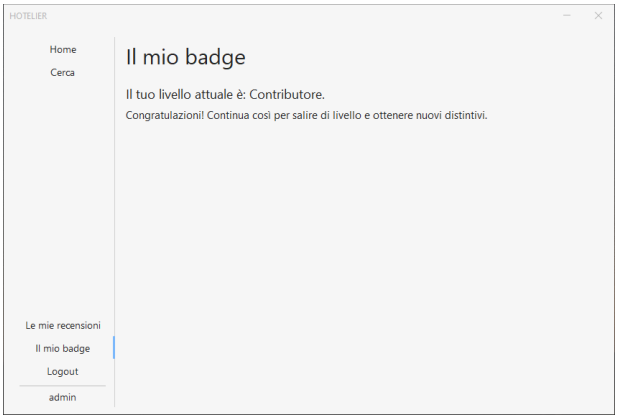
(b) `GuestPanel` con `SearchTab`



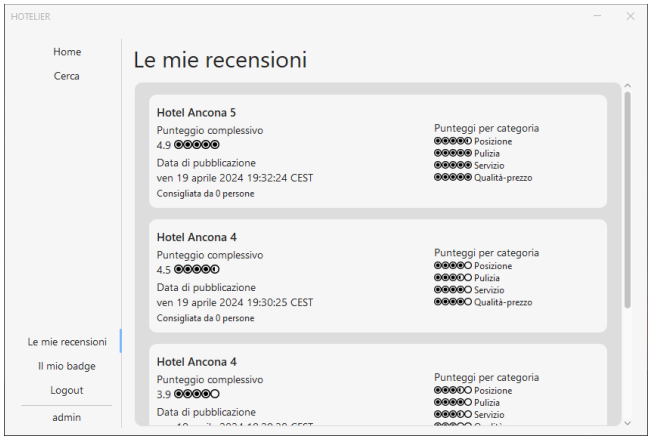
(c) `SearchTab`



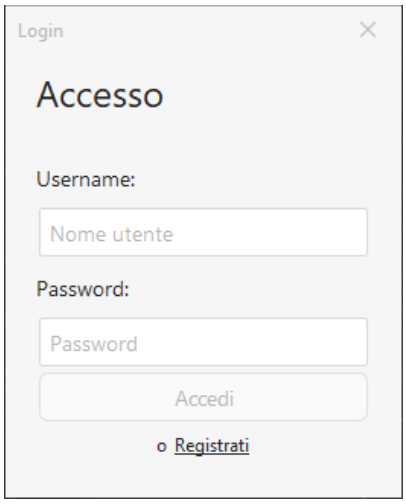
(d) `BadgeTab`



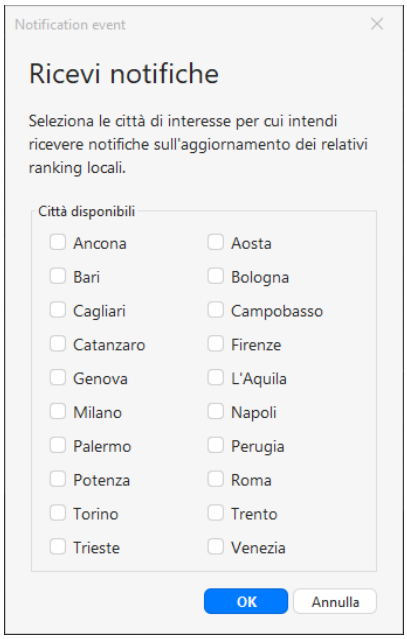
(e) `ReviewsTab`



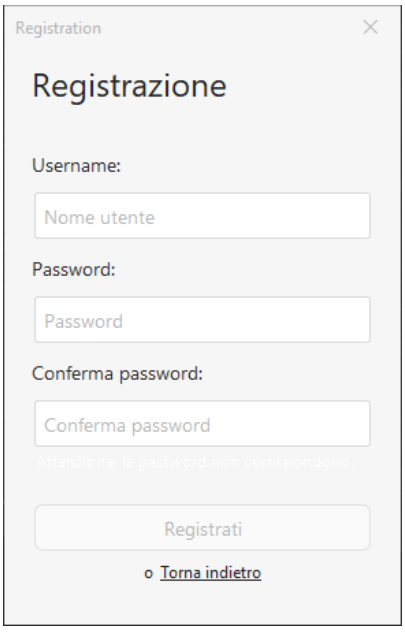
(a) `LoginDialog`



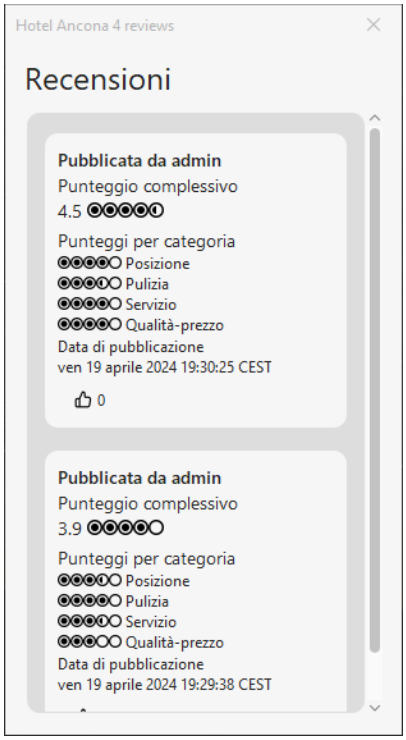
(b) `NotificationDialog`



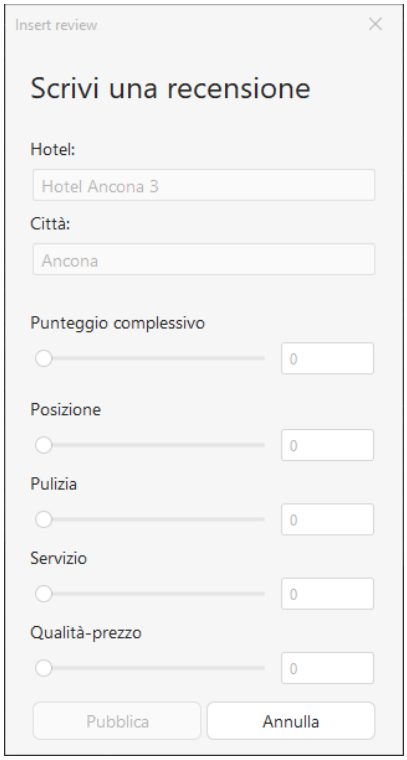
(c) `RegistrationDialog`



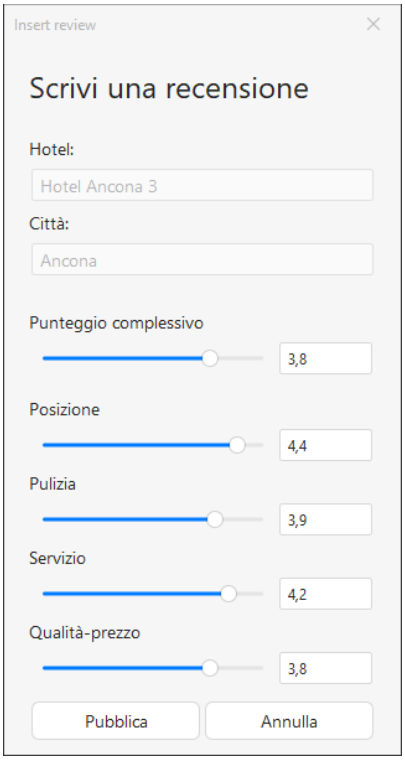
(d) `ReviewsDialog`



(e) `ReviewPostDialog`



(f) `ReviewsPostDialog` compilato



10 Altri file

10.1 `Server` package

I file contenuti all'interno del package `config`, situato nella directory `HOTELIER/Server/`, sono i file dedicati ai database che verranno caricati dal server al suo primo avvio. A questi va aggiunto il file di configurazione `server.properties`, nel quale vengono definite le proprietà e i parametri che verranno usati per inizializzare il server.

Il file `hotel_database` è quello che costituisce per l'appunto il database degli hotel, generato tramite il metodo `main` della classe `JsonFile`, il quale crea per ogni città cinque hotel, ciascuno di tipo diverso, con descrizione casuale e servizi offerti assegnati in quantità e modo casuale.

Il file `user_database` è dedicato invece agli utenti ma contiene – attualmente – solamente l'utente admin, il supposto amministratore del servizio.

10.2 `Client` package

I file contenuti all'interno del package `config`, situato nella directory `HOTELIER/Client/`, sono il file di configurazione `client.properties` (analogo a quello visto per il server), una cartella `fonts` e una cartella `icons`, contenenti rispettivamente i file `.ttf` e `.png` usati per la visualizzazione grafica degli elementi corrispondenti, nonché font e immagini come icone.

10.3 File di backup

La cartella `backup`, situata nella directory principale `HOTELIER/`, contiene i file di backup, rispettivamente degli hotel e degli utenti del servizio, suddivisi in ulteriori due cartelle: `hotel/` e `user/`.

A ogni avvio il server controlla se la directory esiste[15] e ne carica il contenuto per ripristinare le strutture dati in uso durante l'ultima esecuzione. Dopodiché sarà il `BackupHandler` a occuparsi del salvataggio periodico dei dati all'interno di questa cartella, secondo la struttura appena definita.

11 Librerie

Di seguito le librerie utilizzate per la realizzazione del progetto.

11.1 Jackson

Questa libreria viene usata per la serializzazione e la deserializzazione dei dati, che in questo modo vengono resi persistenti e possono essere ripristinati ai successivi avvii del server.

- `jackson-annotations-2.16.1.jar`
- `jackson-core-2.16.1.jar`
- `jackson-databind-2.16.1.jar`

11.2 Apache Commons CLI

Questa libreria viene usata per il parsing dei comandi all'avvio del client con CLI.[16]

- `commons-cli-1.6.0.jar`

[15] Potrebbe essere infatti il primo avvio o potrebbe anche essere stata eliminata.

[16] Al momento è utilizzata esclusivamente per la scelta dell'interfaccia che si desidera avviare ma, con l'aggiunta di nuove `properties`, può essere usata per la scelta a proprio piacimento, ad esempio, dell'indirizzo IP del gruppo di multicast o dell'indirizzo IP del server.

11.3 FlatLaf

Questa libreria viene usata per il *Look and Feel* dell'interfaccia grafica implementata.

➤ `flatlaf-3.4.jar`

11.4 SwingX

Questa libreria viene usata per l'implementazione dell'interfaccia grafica.

➤ `swingx-all-1.6.4.jar`

12 Compilazione ed esecuzione

La compilazione e l'esecuzione richiedono il posizionamento all'interno della directory `HOTELIER/`.[17]

12.1 Windows

12.1.1 Compilazione

```
javac -cp lib/jackson/jackson-annotations-2.16.1.jar;lib/jackson/jackson-core-2.16.1.jar;
lib/jackson/jackson-databind-2.16.1.jar;lib/commons-cli-1.6.0.jar;lib/flatlaf-3.4.jar;lib
/swingx-all-1.6.4.jar src/MyExceptions/*.java src/Server/Database/Hotel/*.java src/Server
/Database/*.java src/Client/GUI/*.java src/Client/*.java src/Server/*.java src/Server/con
fig/JsonFile.java src/HOTELIERServerMain.java src/HOTELIERClientMain.java
```

12.1.2 Esecuzione

Esecuzione senza `jar` eseguibile

```
Esecuzione server
java -cp lib/jackson/*;src HOTELIERServerMain
Esecuzione client
java -cp lib/*;src HOTELIERClientMain [-cli|-gui]
```

Esecuzione con `jar` eseguibile

```
Esecuzione server
java -jar hotelier-server.jar
Esecuzione client
java -jar hotelier-client.jar [-cli|-gui]
```

12.2 UNIX[18]

12.2.1 Compilazione

```
javac -cp .:lib/jackson/jackson-annotations-2.16.1.jar:lib/jackson/jackson-core-2.16.1.ja
r:lib/jackson/jackson-databind-2.16.1.jar:lib/commons-cli-1.6.0.jar:lib/flatlaf-3.4.jar:l
ib/swingx-all-1.6.4.jar src/MyExceptions/*.java src/Server/Database/Hotel/*.java src/Serv
er/Database/*.java src/Client/GUI/*.java src/Client/*.java src/Server/*.java src/Server/c
onfig/JsonFile.java src/HOTELIERServerMain.java src/HOTELIERClientMain.java
```

[17] Dopo aver copiato i comandi che seguono, prima di eseguirli sul terminale, è consigliato incollarli in un blocco note o in un file di testo per rimuovere eventuali spazi e nuove righe che possono essersi creati dal PDF.

[18] Per una corretta visualizzazione dell'interfaccia grafica si consiglia l'utilizzo con Windows o macOS.

12.2.2 Esecuzione

Esecuzione senza `jar` eseguibile

```
Esecuzione server
java -cp .:lib/jackson/*:src HOTELIERServerMain
Esecuzione client
java -cp .:lib/*:src HOTELIERClientMain [-cli|-gui]
```

Esecuzione con `jar` eseguibile: come per Windows.

13 Comandi e istruzioni

Di seguito un riepilogo dei comandi che è possibile usare lato client e che vengono visualizzati dal programma, anche su richiesta: in particolare nel menu all'avvio o al login/logout e mediante il comando `help`.[19]

```
register        per registrarsi
                > richiesti:  username, password
login           per effettuare l'accesso
                > richiesti:  username, password
search          per effettuare una ricerca di un hotel in una città
                > richiesti:  hotel, città
searchall       per trovare tutti gli hotel di una città
                > richiesti:  città
insertreview    per pubblicare una recensione riferita a un hotel di una città
                > richiesti:  hotel, città, punteggio complessivo, punteggi per categoria
showmybadges    per visualizzare il tuo badge
                > richiesti:  none
showreviews     per visualizzare tutte le recensioni riferita all'hotel di una città
                > richiesti:  hotel, città
upvote          per consigliare una recensione
                > richiesti:  #recensione
showmyreviews   per visualizzare le recensioni che hai pubblicato
                > richiesti:  none
help            per ricevere aiuto
cancel          per annullare un'operazione in qualsiasi momento
quit            per uscire dall'applicazione
```

[19] I comandi descritti sono quelli che si possono usare direttamente tramite CLI, dal momento che questi vengono eseguiti da terminale.