PROGETTO LABORATORIO RETI 2023/2024: HOTELIER

Nome: Natalia
Cognome: Ceccarini
Matricola: 598537

BREVE ILLUSTRAZIONE DEL MATERIALE CONSEGNATO E DEGLI STRUMENTI DI SVILUPPO UTILIZZATI.

Il progetto è stato sviluppato ed eseguito utilizzando l'ambiente di sviluppo integrato (IDE) IntelliJ IDEA. Successivamente, è stato sottoposto a un processo di testing mediante compilazione ed esecuzione attraverso i comandi *javac* e *java* da terminale su diverse piattaforme, tra cui macOS (versione Java 20.0.1), Windows 11 (versione Java 20.0.1), Ubuntu 20.04.6 LTS (versione 20.0.1).
La cartella principale, denominata *progettoCeccarini*, contiene diverse sottocartelle organizzate come segue:

- **Client**: contiene i file sorgente (estensione *.java*) esclusivamente relativi al lato client dell'applicazione.
- **Server**: contiene i file sorgente (estensione *.java*) esclusivi per il lato server.
- **Shared**: include i file sorgente (estensione *.java*) delle classi condivise tra client e server.
- **libs**: contiene le librerie esterne utilizzate nel progetto sotto forma di file *.jar* (nello specifico, *gson-2.10.1* e *java-json*).
- **bin**: raccoglie i file eseguibili (estensione *.class*), generati dal processo di compilazione dei sorgenti del client e del server tramite il comando *javac*.

Oltre alle suddette sottocartelle, la directory *progettoCeccarini* include i seguenti file di supporto:

- **client_config.properties**: file di configurazione contenente i parametri specifici del client.
- **server_config.properties**: file di configurazione contenente i parametri relativi al server.
- **RegistrazioniUtenti.txt**: file contenente dati in formato JSON, generato da un thread lato server, che si occupa della serializzazione degli utenti registrati all'applicazione "Hotelier".
- **ListaHotel.txt**: file contenente l'elenco delle strutture alberghiere presenti nel sistema.
- **ReviewsHotel.txt**: file che raccoglie le recensioni degli utenti sugli hotel presenti nel sistema.

ISTRUZIONI PER COMPILARE ED ESEGUIRE IL PROGETTO SU VARI SISTEMI OPERATIVI.

Windows 11

Per compilare il progetto su Windows 11 seguire le seguenti istruzioni: aprire un terminale nella cartella progettoCeccarini oppure aprire il terminale in una cartella qualsiasi e posizionarsi nella cartella progettoCeccarini.

Dalla cartella progettoCeccarini compilare server e client digitando:

Compilazione server: javac -cp ".\libs*" -d .\bin\ -sourcepath ".\Server;.\Shared" .\Server*.java .\Shared*.java

Esecuzione server: java -cp ".\bin;.\libs\gson-2.10.1.jar;.\libs\java-json.jar" Server.ServerMain

Compilazione client: javac -cp ".\libs*" -d .\bin\ -sourcepath ".\Client;.\Shared" .\Client*.java .\Shared*.java

Esecuzione client: java -cp ".\bin;.\libs\gson-2.10.1.jar;.\libs\java-json.jar" Client.ClientMain

MacOS e Ubuntu

Compilazione server: javac -cp ./libs/gson-2.10.1.jar:./libs/java-json.jar -d ./bin/ -sourcepath ./Server:./Shared ./Server/*.java ./Shared/*.java

Esecuzione server: java -cp ./bin:./libs/gson-2.10.1.jar:./libs/java-json.jar Server.ServerMain

Compilazione client: javac -cp ./libs/gson-2.10.1.jar:./libs/java-json.jar -d ./bin/ -sourcepath ./Client:./Shared ./Client/*.java ./Shared/*.java

Esecuzione client: java -cp ./bin:./libs/gson-2.10.1.jar:./libs/java-json.jar Client.ClientMain

ISTRUZIONI PER CREARE I FILE JAR

Prima di eseguire i seguenti comandi è necessario creare i due file manifest per il server e per il cliet (MANIFESTSERVER.TXT e MANIFESTCLIENT.TXT) nella directory centrale del progetto.

Questi sono i comandi utilizzati in MacOS:

mkdir buildServer

cp -r bin/Server buildServer/

cp -r bin/Shared buildServer/

jar cmf MANIFESTSERVER.txt Server.jar -C buildServer/ .

java -jar Server.jar

mkdir buildClient

cp -r bin/Client buildClient/

cp -r bin/Shared buildClient/

jar cmf MANIFESTCLIENT.txt Client.jar -C buildClient/ .

java -jar Client.jar

INTRODUZIONE HOTELIER

Il progetto Hotelier consiste nell'implementazione di un applicativo software che si ispira all'applicazione di TripAdvisor, concentrandosi esclusivamente sul settore alberghiero. L'obiettivo è quello di creare una piattaforma che consenta agli utenti di registrarsi, accedere al proprio profilo, ricercare hotel, pubblicare recensioni dettagliate assegnando un punteggio globale e a varie categorie come posizione, pulizia, servizio e qualità, e monitorare i propri distintivi, che rappresentano il livello di esperienza all'interno della piattaforma.

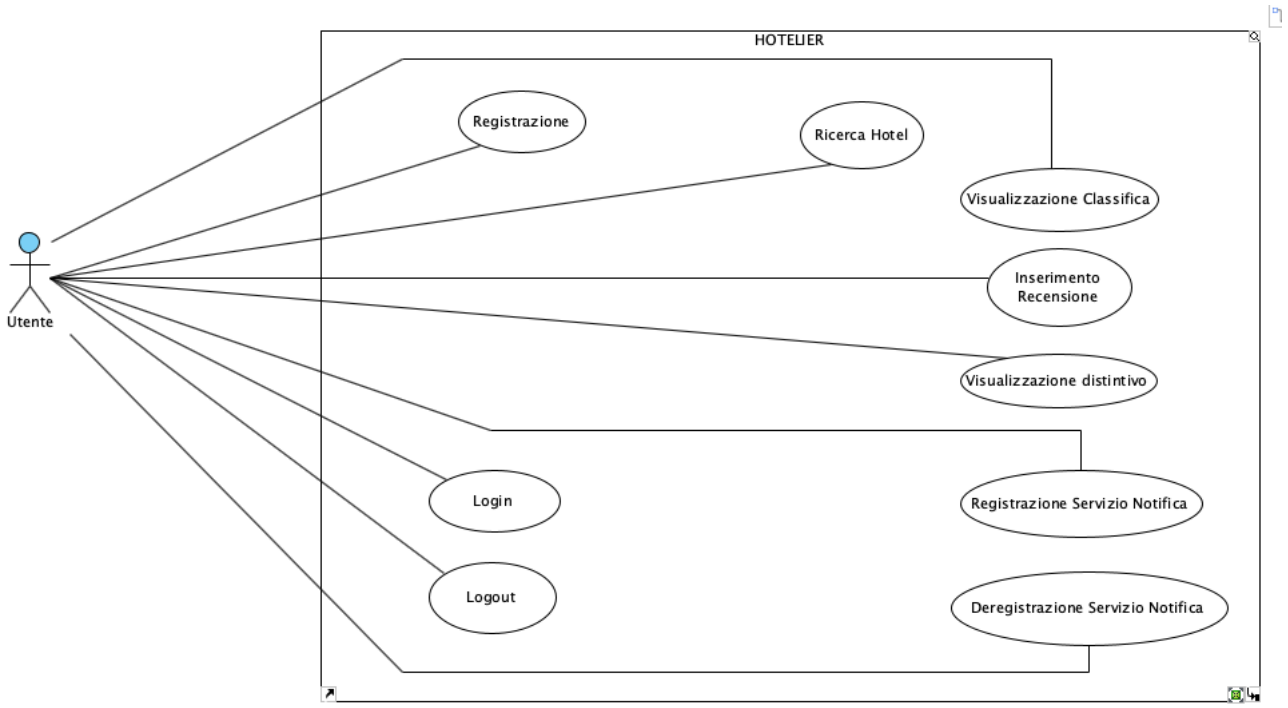
Il dominio di Hotelier si limita alla gestione di strutture alberghiere nelle città capoluogo delle 20 regioni italiane. A differenza di TripAdvisor, che copre un ampio spettro di strutture come ristoranti e case vacanze.

Il sistema calcola periodicamente il ranking di ogni hotel, aggiornando le classifiche in base a un'analisi dettagliata. Il valore del ranking viene determinato considerando tre fattori principali: i punteggi globali assegnati dagli utenti, la data di pubblicazione delle recensioni e il numero totale di recensioni ricevute.

Dal punto di vista dei requisiti funzionali, il sistema offre le seguenti funzionalità:

- Registrazione e Login: Gli utenti possono creare un account personale e accedere alla piattaforma per usufruire delle funzionalità di inserimento recensioni e monitoraggio dei distintivi.
- Logout: Gli utenti possono uscire dal proprio account.
- Ricerca di Hotel: Gli utenti possono effettuare ricerche mirate per trovare specifici hotel in determinate città, con la possibilità di consultare il rank delle singole categorie e il rank globale.
- Inserimento di Recensioni: Gli utenti loggati possono inserire recensioni assegnando un punteggio sintetico globale e punteggi specifici per ciascuna categoria. Quest'ultimi contribuiscono al calcolo del rank dell'hotel.
- Visualizzazione classifica: Quando gli utenti ricercano gli hotel per città, le strutture sono ordinate in base al ranking, fornendo agli utenti una classifica aggiornata delle strutture.
- Visualizzazione distintivo: Gli utenti autenticati accumulano esperienza in base al numero di recensioni che inseriscono, avanzando attraverso vari livelli di esperienza. Ogni livello è rappresentato da un distintivo che riflette il grado di attività e contributo dell'utente. Gli utenti possono visualizzare il loro distintivo attuale per monitorare i progressi e riconoscimenti raggiunti.
- Notifiche in tempo reale su gruppo multicat: La piattaforma invia notifiche agli utenti loggati e uniti al gruppo multicast, ogni volta che un hotel diventa il nuovo primo classificato in una delle città.
- Iscrizione alle notifiche di aggiornamento classifica: Gli utenti possono iscriversi a un servizio di notifica per ricevere aggiornamenti sui cambiamenti nelle classifiche degli hotel delle città di loro interesse. Questo servizio consente loro di monitorare le variazioni delle posizioni degli hotel.
- Disiscrizione dal servizio di notifica di aggiornamenti classifica: Gli utenti hanno la possibilità di annullare l'iscrizione al servizio di notifica in qualsiasi momento. Questo permette loro di interrompere la ricezione degli aggiornamenti relativi alle variazioni delle classifiche degli hotel nelle città di loro interesse.

DIAGRAMMA DEI CASI D'USO



Caso d’uso: Registrazione

Breve descrizione: Un utente desidera registrarsi al sistema fornendo un nome utente e una password. Il sistema verifica la validità delle credenziali e registra l'utente se il nome inserito non è già in uso.

Attore Primario**:** Utente

Precondizioni**:**

- L'utente deve fornire un nome utente e una password validi.
- L'utente deve avere accesso a un UUID associato alla chiave di decrittazione.

Sequenza degli Eventi Principale:

1. L'utente invia una richiesta di registrazione fornendo un nome utente, una password cifrata e un UUID.
2. Il sistema verifica che il nome utente e la password non siano nulli o vuoti.
3. Il sistema recupera la chiave di decrittazione associata all'UUID fornito.
4. Il sistema decifra la password utilizzando la chiave di decrittazione.
5. Il sistema verifica che il nome utente non sia già presente nel database.
6. Se il nome utente è unico, il sistema crea un nuovo utente e lo aggiunge al database.
7. Il sistema restituisce una risposta positiva, indicando che l'utente è stato registrato con successo.

Postcondizioni:

- L'utente viene registrato nel sistema.
- Il database degli utenti viene aggiornato con le nuove informazioni.

Sequenze Alternative degli Eventi:

- Nome utente o password nulli o vuoti: Il sistema restituisce un messaggio di errore (Codice 400).
- Chiave di decrittazione non trovata per l'UUID: Il sistema restituisce un messaggio di errore (Codice 404).
- Password decifrata vuota: Il sistema restituisce un messaggio di errore (Codice 400).
- Nome utente già in uso: Il sistema restituisce un messaggio di errore (Codice 401) e non registra l'utente.

Caso d'Uso: Login

Breve Descrizione**:** Un utente desidera effettuare l'accesso al sistema fornendo il proprio nome utente e la password cifrata. Il sistema verifica le credenziali e, se corrette, consente l'accesso.

Attore Primario: Utente

Precondizioni:

- L'utente deve essere già registrato nel sistema.
- L'utente deve fornire un nome utente e una password validi.
- L'utente deve avere accesso a un UUID associato alla chiave di decrittazione.

Sequenza degli Eventi Principale:

1. L'utente invia una richiesta di login fornendo il nome utente, la password cifrata e un UUID.
2. Il sistema verifica che il nome utente e la password non siano nulli o vuoti.
3. Il sistema verifica se l'utente è registrato nel database.
4. Il sistema recupera la chiave di decrittazione associata all'UUID fornito.
5. Il sistema decifra la password utilizzando la chiave di decrittazione.
6. Il sistema confronta la password decifrata con quella registrata per l'utente.
7. Se le credenziali sono corrette, il sistema verifica se l'utente è già loggato sulla specifica connessione.
8. Se l'utente non è già loggato, il sistema associa l'utente al canale di comunicazione e conferma il login.
9. Il sistema restituisce una risposta positiva, indicando che il login è stato effettuato con successo.

Postcondizioni:

- L'utente viene autenticato e ha accesso al sistema.
- Il canale di comunicazione dell'utente viene associato al suo nome utente nel sistema.

Sequenze Alternative degli Eventi:

- Nome utente o password nulli o vuoti: Il sistema restituisce un messaggio di errore (Codice 400) e non effettua il login.
- Utente non registrato: Il sistema restituisce un messaggio di errore (Codice 401) indicando che l'utente deve prima registrarsi.
- Credenziali errate: Il sistema restituisce un messaggio di errore (Codice 401) e non effettua il login.
- Utente già loggato**:** Il sistema restituisce un messaggio di errore (Codice 403) indicando che l'utente ha già effettuato il login.

Caso d'Uso: Logout Utente

Breve Descrizione**:** Un utente desidera disconnettersi dal sistema. Il sistema gestisce la disconnessione rimuovendo l'associazione tra il canale di comunicazione e l'utente, terminando così la sessione di login attiva.

Attore Primario: Utente

Precondizioni:

- L'utente deve essere autenticato.

Sequenza degli Eventi Principale:

1. L'utente invia una richiesta di logout.
2. Il sistema identifica il canale di comunicazione associato all'utente tramite la chiave di selezione.
3. Il sistema verifica se il canale di comunicazione è associato a una sessione di login attiva per l'utente.
4. Se la sessione di login è attiva, il sistema rimuove l'associazione tra il canale di comunicazione e l'utente
5. Il sistema invia una risposta di conferma, indicando che il logout è stato effettuato con successo.

Postcondizioni:

- L'utente viene disconnesso dal sistema.

Sequenze Alternative degli Eventi:

- Utente non loggato: Se l'utente non ha una sessione di login attiva, il sistema restituisce un messaggio di errore (Codice 401) indicando che l'utente non è loggato.

Caso d'Uso: Visualizzazione del Distintivo

Breve Descrizione: Un utente autenticato desidera visualizzare il proprio distintivo attuale, che riflette il livello di esperienza raggiunto attraverso l'attività di inserimento recensioni.

Attore Primario: Utente

Precondizioni:

- L'utente deve essere autenticato
- L'utente deve essere registrato nel sistema.

Sequenza degli Eventi Principale:

1. L'utente invia una richiesta per visualizzare il proprio distintivo.
2. Il sistema verifica se l'utente è loggato tramite il canale di comunicazione associato.
3. Il sistema recupera i dati dell'utente.
4. Il sistema ottiene il distintivo dell'utente.
5. Se l'utente ha un distintivo, il sistema lo restituisce come risposta.
6. Se l'utente non ha ancora ottenuto un distintivo, il sistema invia un messaggio che informa l'utente della necessità di inserire almeno una recensione per ricevere un distintivo.

Postcondizioni:

- L'utente visualizza il proprio distintivo attuale o riceve un messaggio che lo informa che non ha ancora ottenuto un distintivo.

Sequenze Alternative degli Eventi:

- Utente non loggato: Se l'utente non è loggato, il sistema restituisce un messaggio di errore (Codice 401) che invita l'utente a effettuare il login prima di visualizzare il distintivo.
- Utente non registrato: Se l'utente non è registrato, il sistema restituisce un messaggio di errore (Codice 401) che invita l'utente a registrarsi per poter visualizzare il distintivo.

Caso d'Uso: Ricerca Hotel

Breve Descrizione: Un utente desidera cercare un hotel specifico in una determinata città. Il sistema verifica la presenza dell'hotel nella città richiesta e fornisce le informazioni relative all'hotel se trovato.

Attore Primario: Utente

Precondizioni:

- L'utente deve specificare il nome dell'hotel e la città in cui desidera effettuare la ricerca.
- La lista degli hotel deve essere caricata nel sistema.

Sequenza degli Eventi Principale:

1. L'utente inserisce il nome dell'hotel e la città in cui vuole cercare l'hotel.
2. Il sistema verifica che i dati di input (nome dell'hotel e città) siano validi.
3. Il sistema controlla se la città esiste nel database degli hotel.
4. Se la città esiste, il sistema cerca l'hotel specificato all'interno della città.
5. Se l'hotel viene trovato, il sistema restituisce le informazioni sull'hotel all'utente.
6. Se l'hotel non viene trovato, il sistema restituisce un messaggio che informa l'utente che l'hotel non è presente nella città specificata.

Postcondizioni:

- L'utente riceve le informazioni sull'hotel richiesto se è presente nel database.
- Se l'hotel non è presente, l'utente viene informato che l'hotel non è stato trovato.

Sequenze Alternative degli Eventi:

- Input nullo o vuoto: Se il nome dell'hotel o la città sono nulli o vuoti, il sistema restituisce un messaggio di errore (Codice 400) che richiede di fornire i dati necessari per la ricerca.
- Città non trovata**:** Se la città non è presente nel database, il sistema restituisce un messaggio di errore (Codice 404) che informa l'utente che non ci sono hotel nella città cercata.
- Hotel non trovato**:** Se l'hotel non è presente nella città specificata, il sistema restituisce un messaggio di errore (Codice 404) che informa l'utente che l'hotel non è stato trovato.

Caso d'Uso: Inserimento Recensione

Breve Descrizione: Un utente desidera inserire una recensione per un hotel specifico in una determinata città. Il sistema registra la recensione associandola all'hotel e aggiorna il numero totale di recensioni effettuate dall'utente.

Attore Primario: Utente

Precondizioni:

- L'utente deve essere autenticato e associato a una sessione attiva.
- L'utente deve specificare un nome di hotel, una città, un punteggio globale, e i punteggi specifici per le varie categorie.

- Il database degli hotel deve essere disponibile e caricato nel sistema.

Sequenza degli Eventi Principale:

1. L'utente inserisce il nome dell'hotel, la città, il punteggio globale e i punteggi specifici per le categorie.
2. Il sistema verifica la validità degli input forniti (ad esempio, che i punteggi siano compresi tra 0 e 5).
3. Il sistema verifica se l'utente è autenticato.
4. Il sistema cerca l'hotel specificato nella città indicata.
5. Se l'hotel è trovato, il sistema:
    - Crea una nuova recensione utilizzando i dettagli forniti dall'utente e la data corrente.
    - Aggiunge la nuova recensione alla lista delle recensioni associate all'hotel.
    - Aggiorna il numero totale di recensioni effettuate dall'utente.
    - Invia un messaggio di conferma all'utente che la recensione è stata aggiunta con successo.
6. Se l'hotel non è trovato, il sistema informa l'utente che l'hotel specificato non è stato trovato nella città indicata.
7. Se la città non è trovata, il sistema informa l'utente che non ci sono hotel nella città cercata.

Postcondizioni:

- La recensione viene aggiunta al database delle recensioni e associata all'hotel specificato.
- Il numero di recensioni effettuate dall'utente viene incrementato.

Sequenze Alternative degli Eventi:

- Input non valido: Se uno o più dei dati di input (nome dell'hotel, città, punteggio globale o punteggi specifici) sono nulli, vuoti o fuori dai limiti accettabili, il sistema restituisce un messaggio di errore (Codice 400) che informa l'utente della necessità di fornire dati validi.
- Utente non autenticato: Se l'utente non è autenticato, il sistema restituisce un messaggio di errore (Codice 401) che informa l'utente che deve essere autenticato per poter inserire una recensione.
- Hotel non trovato: Se l'hotel non è presente nella città specificata, il sistema restituisce un messaggio di errore (Codice 404) che informa l'utente che l'hotel non è stato trovato.
- Città non trovata: Se la città non è presente nel database, il sistema restituisce un messaggio di errore (Codice 404) che informa l'utente che non ci sono hotel nella città cercata.

Caso d'Uso: Visualizzazione Classifica

Breve Descrizione: Un utente richiede la lista degli hotel in una città specifica, ordinati in base al punteggio globale (rate). Il sistema recupera e invia la classifica degli hotel in quella città all’utente.

Attore Primario: Utente

Precondizioni:

- Il database degli hotel deve essere disponibile e caricato nel sistema.

Sequenza degli Eventi Principale:

1. L'utente invia una richiesta per visualizzare la lista degli hotel in una città specifica.
2. Il sistema valida l'input per assicurarsi che il nome della città sia fornito e non sia vuoto.
3. Il sistema cerca la città.
4. Se la città è trovata, il sistema:
    - Recupera la lista degli hotel per quella città, ordinata in base al rate globale.
    - Invia la classifica degli hotel all'utente, insieme a un messaggio di successo.
5. Se la città non viene trovata, il sistema invia un messaggio all'utente indicando che non ci sono hotel nella città specificata.

Postcondizioni:

- L'utente riceve la classifica degli hotel per la città richiesta, se disponibile.
- Se la città non è presente nel database, l'utente viene informato dell'assenza di hotel nella città specificata.

Sequenze Alternative degli Eventi:

- Input non valido: Se il nome della città è nullo o vuoto, il sistema restituisce un messaggio di errore (Codice 400) che informa l'utente dell'input non valido.
- Città non trovata: Se la città non è presente nella classifica, il sistema restituisce un messaggio di errore (Codice 404) che informa l'utente che non ci sono hotel nella città cercata.

Caso d'Uso: Registrazione al servizio di notifica

Breve Descrizione: Un utente si registra al servizio di notifica per ricevere aggiornamenti sui ranking di una città specifica. Il sistema memorizza lo stub dell'oggetto remoto del client per inviare notifiche future.

Attore Primario: Utente

Precondizioni:

- Il client deve essere autenticato al sistema.
- Il database degli hotel deve essere disponibile e accessibile.
- Il client deve fornire una città valida e lo stub dell'oggetto remoto del client per la registrazione.

Sequenza degli Eventi Principale:

1. L'utente invia una richiesta per registrarsi al servizio di notifica per una città specifica, includendo lo stub dell'oggetto remoto del client.
2. Il sistema valida l'input per assicurarsi che il nome della città sia fornito e non sia nullo o vuoto.
3. Il sistema verifica se la città esiste nel database degli hotel.
4. Se la città esiste:
    - Il sistema aggiunge stub dell'oggetto remoto del client alla lista degli stub dei client.
    - Il sistema restituisce un messaggio di successo al client, confermando la registrazione.
5. Se la città non esiste nel database:
    - Il sistema restituisce un messaggio di errore al client, informandolo che la città specificata non è presente nel database.

Postcondizioni:

- Il client è registrato con successo per ricevere notifiche sugli hotel nella città specificata, se esistente.

Sequenze Alternative degli Eventi:

- Input non valido: Se il nome della città è nullo o vuoto, il sistema restituisce un messaggio di errore (Codice 400) che informa il client dell'input non valido.
- Città non trovata: Se la città non è presente nel database degli hotel, il sistema restituisce un messaggio di errore (Codice 404) informando il client che non ci sono hotel nella città cercata.

Caso d'Uso: Deregistrazione dal Servizio di Notifica

Descrizione: Un utente richiede di essere deregistrato dal servizio di notifiche di una o più città a cui era precedentemente iscritto. Il sistema rimuove lo stub dell'oggetto remoto del client dalla lista degli stub degli oggetti remoti dei client iscritti ad una specifica cittò

Attore Primario: Utente

Precondizioni:

- Il client deve essere registrato per ricevere notifiche.

Sequenza degli Eventi Principale:

1. Il client invia una richiesta di disiscrizione al servizio di notifica e come parametri: la città e lo stub dell'oggetto remoto del client da rimuovere.
2. Il sistema verifica che lo stub e la città non siano nulli.
3. Il sistema itera sulla lista delle callback della città e rimuove clientInterface.
4. Se lo stub è stato rimosso, il sistema:
    - Restituisce un messaggio di successo (Codice 200) confermando la deregistrazione.
5. Altrimenti:
    - Restituisce un messaggio di errore (Codice 404) indicando che il client non era registrato in nessuna città.

Postcondizioni:

- Il client è stato rimosso dalla lista di callback per la città specificata.

Sequenze Alternative:

- Input nullo: Se stub è nullo, il sistema restituisce un messaggio di errore (Codice 400) informando il client che l'input è nullo.
- Stub non trovato: Se stub non era presente, il sistema restituisce un messaggio di errore (Codice 404) indicando che il client non era registrato in nessuna città.

ARCHITETTURA SOFTWARE

Il funzionamento dell'applicazione si basa sul paradigma client-server, un modello architetturale ampiamente utilizzato nelle applicazioni moderne. In questo modello, le responsabilità sono suddivise tra due entità principali: il client e il server. Il server, in qualità di fornitore di servizi, è incaricato di elaborare e gestire tutte le richieste inviate dai client. Al contrario, i client sono i consumatori di questi servizi e inoltrano al server le loro richieste specifiche.

La comunicazione tra client e server avviene tramite stringhe formattate in JSON, un formato di scambio dati ampiamente riconosciuto e utilizzato per la sua leggibilità e semplicità. Ogni richiesta dal client al server segue una struttura ben definita:

```
{

  "operation": "",

  "param": {

    "": ""

  }

}
```

In questo schema, "operation" specifica l'operazione richiesta dal client, mentre "param" è un dizionario di coppie chiave-valore, che fornisce al server le informazioni dettagliate di cui ha bisogno per elaborare la richiesta.

Le risposte del server, una volta elaborata la richiesta del client, sono anch'esse formattate in JSON. La struttura di una risposta JSON è la seguente:

```
{

  "statusCode": codice stato HTTP,

  "message": {

    "result": risultato restituito,

    "body": "messaggio esplicativo"

  }

}
```

In questo schema, "statusCode" rappresenta il codice di stato HTTP che indica l'esito della richiesta. Quest'ultimo è essenziale per comunicare se la richiesta è stata elaborata con successo, o se si è verificato un errore. La chiave "message" contiene ulteriori dettagli sulla risposta. All'interno di "message", la chiave "result" fornisce il risultato concreto restituito dal server al client, come ad esempio dati recuperati. La chiave "body" contiene un messaggio esplicativo che offre una descrizione chiara e comprensibile dell'esito della richiesta, aiutando l'utente a comprendere meglio la risposta ricevuta.

La maggior parte delle comunicazioni tra server e client nell'applicazione avviene prevalentemente tramite API Socket, che sfrutta il protocollo TCP/IP per garantire un trasferimento dati affidabile e sicuro. Per gestire le connessioni di rete, è stato adottato un approccio basato su Multiplexed Non-Blocking Sockets utilizzando Java NIO. Questa architettura consente di gestire numerose connessioni simultaneamente attraverso un unico thread, riducendo il sovraccarico associato al cambio di contesto e ottimizzando l'uso delle risorse di sistema.

Il cuore di questa implementazione è il Selector, un componente che monitora uno o più canali di I/O (NIO Channels) e determina quali sono pronti per operazioni di lettura o scrittura. Il Selector utilizza il metodo select per rilevare i canali che sono pronti per un'operazione I/O. Solo quando essi sono effettivamente pronti, il Selector sblocca il thread e consente al programma di iterare attraverso i canali.

Questo approccio è cruciale per l'efficienza del sistema, poiché garantisce che le risorse siano utilizzate solo quando necessario. Il Selector assicura che le operazioni vengano eseguite solo quando ci sono dati disponibili, come byte da leggere, evitando che i thread rimangano bloccati su operazioni di I/O.

DESCRIZIONE FLUSSO DI ESECUZIONE SERVER

Quando il server viene avviato, la prima operazione eseguita è la lettura del file di configurazione. Questo file contiene tutti i parametri di setting necessari per inizializzare l'oggetto della classe HotelierServer, responsabile dell'elaborazione delle richieste dei client. I parametri di configurazione includono:

- **Indirizzo IP del gruppo multicast:** Utilizzato per inviare messaggi al gruppo
  multicast.multicast_address=226.226.226.226
- **Porta del gruppo multicast:** Porta attraverso cui avviene la comunicazione multicast.
  multicast_port=8989
- **Porta del registry RMI:** Porta associata al registry RMI per la gestione degli oggetti remoti.
  registry_port=3333
- **Porta TCP del server:** Porta utilizzata per le connessioni TCP con i client.
  server_port_tcp=8000
- **Timeout per la serializzazione dei dati degli hotel:** Tempo in millisecondi prima di salvare i dati degli hotel su file.
  timeoutHotel=130000
- **Timeout per il calcolo della classifica degli hotel:** Tempo in millisecondi per l'aggiornamento delle classifiche.
  timeoutRanking=120000
- **Timeout per la serializzazione delle recensioni:** Tempo in millisecondi prima di salvare i dati delle recensioni su file.
  timeoutReviews=60000
- **Timeout per la serializzazione degli utenti registrati:** Tempo in millisecondi prima di salvare i dati degli utenti su file.
  timeoutUser=90000
- **Identificatore per l'oggetto di registrazione RMI:** Numero identificativo dell'oggetto registrato nel registry RMI.
  registr_obj=0
- **Identificatore per l'oggetto di notifica RMI:** Numero identificativo per la gestione delle notifiche nel registry RMI.
  notific_obj=39000
- **Numero pubblico per il calcolo della chiave pubblica (Diffie-Hellman):** Numero utilizzato nell'algoritmo di scambio chiavi.
  P_number=9973
- **Generatore per il calcolo delle chiavi intermedie (Diffie-Hellman):** Generatore utilizzato nell'algoritmo di scambio chiavi.
  G_generator=7
- **Percorso del file per i dati degli utenti registrati:** Posizione del file in cui sono salvati i dati degli utenti.
  filePathUser=/RegistrazioniUtenti.txt
- **Percorso del file per i dati degli hotel:** Posizione del file che contiene le informazioni sugli hotel.
  filePathHotel=/ListaHotel.txt
- **Percorso del file per i dati delle recensioni:** Posizione del file in cui sono salvate le recensioni degli hotel.
  filePathReviews=/ReviewsHotel.txt

Una volta che l'oggetto HotelierServer è stato creato e tutte le variabili d'istanza sono state inizializzate, il sistema procede al caricamento dei dati dal disco per inizializzare le strutture dati

relative agli utenti registrati, agli hotel e alle recensioni. Queste strutture dati sono fondamentali per il funzionamento dell'applicazione e sono organizzate come segue:

- **Database degli hotel (hotelDB):** È una HashMap<String, HashMap<String, Hotel>>, dove la prima chiave rappresenta la città e la seconda il nome dell'hotel. Questa struttura permette di effettuare ricerche rapide e efficienti in tempo costante (O(1)), in base alla città e al nome dell'hotel.
- **Recensioni degli hotel (reviewsHotel):** È una HashMap<Integer, List<Review>> che associa l'ID di ogni hotel a una lista di recensioni. Anche questa struttura è pensata per consentire un accesso rapido ai dati.
- **Database degli utenti registrati (registredUserDB):** È una ConcurrentHashMap<String, User> che memorizza gli utenti registrati nel sistema. La scelta di una ConcurrentHashMap per gli utenti è stata fatta per bilanciare l'efficienza e la sicurezza nelle operazioni di accesso concorrente.

Scelte Implementative delle Strutture Dati

La scelta di utilizzare una HashMap per gli hotel e le recensioni, anziché una ConcurrentHashMap, è stata motivata dalla necessità di gestire operazioni di modifica e serializzazione in maniera atomica sull'intera struttura. Sebbene una ConcurrentHashMap offra sincronizzazione a livello di singolo bucket, non sarebbe sufficiente per garantire la consistenza durante operazioni che richiedono l'accesso esclusivo all'intera struttura, come la serializzazione dei dati su disco. L'uso di una ConcurrentHashMap in questi casi comporterebbe un overhead aggiuntivo, senza offrire un reale beneficio in termini di sicurezza o performance.
Per il database degli utenti, invece, una ConcurrentHashMap è risultata adeguata poiché solo due operazioni (registrazione di un nuovo utente e aggiornamento del file sul disco) richiedono la sincronizzazione globale. Le altre operazioni possono essere gestite in modo efficiente utilizzando le lock a livello di singolo bucket, che la ConcurrentHashMap gestisce automaticamente, migliorando la concorrenza senza compromettere la sicurezza.

Dopo il caricamento nelle rispettive strutture dati, il server procede con la creazione e l'esportazione degli oggetti remoti RegistrationService e NotificationService tramite RMI (Remote Method Invocation).

RegistrationService
RegistrationService è un'interfaccia remota che viene implementata dalla classe RegistrationServiceImpl. Questa interfaccia definisce il metodo addRegistration, che i client possono invocare per registrare un nuovo utente nel sistema. L'implementazione di addRegistration si occupa di validare i dati forniti dal client e aggiornare il database degli utenti registrati.

NotificationService
NotificationService è un'altra interfaccia remota, implementata dalla classe NotificationServiceImpl. Questa interfaccia fornisce due metodi principali che permettono ai client di gestire la loro iscrizione al servizio di notifica:

1. **registerForCallback**: Questo metodo consente ai client di iscriversi al servizio di notifica per essere avvisati ogni volta che avviene una modifica nella classifica degli hotel di una città a cui sono interessati. Quando un client si registra, il server aggiunge il client a una lista di callback associata alla città di interesse. Ogni volta che la classifica di quella città viene aggiornata, il server invia una notifica ai client registrati, informandoli dei cambiamenti.

2. **unregisterForCallback**: Questo metodo permette ai client di annullare la loro iscrizione al servizio di notifica. Una volta che un client si deregistra, il server rimuove il client dalla lista di callback, interrompendo così l'invio di ulteriori notifiche.

Dopo aver verificato l'esistenza di un registro RMI sulla porta specificata (registry_port) e, se necessario, aver creato un nuovo registro, il server avvia tre thread dedicati. Questi thread, a intervalli di tempo definiti nel file di configurazione, si occupano di scrivere periodicamente i dati rilevanti nei rispettivi file sul disco. Questa operazione è fondamentale per garantire la persistenza delle informazioni, tra cui:

- **Gli utenti registrati:** I dati relativi agli utenti che si sono iscritti alla piattaforma.
- **Le valutazioni degli hotel:** i punteggi assegnati agli hotel dato dal calcolo del rank
- **Le recensioni degli hotel:** le valutazioni lasciate dagli utenti per ogni hotel.

Ogni thread esegue un ciclo infinito in cui, al termine di ogni intervallo di tempo predefinito, si attiva per scrivere le informazioni, aggiornate nelle strutture dati corrispondenti, sui rispettivi file. Questa operazione è cruciale per assicurare che, in caso di interruzione o riavvio del server, le strutture dati siano inizializzate con informazioni coerenti e aggiornate.

Un aspetto chiave di questi thread è la gestione della loro terminazione. Quando viene chiamata l'interrupt del thread, questo non si arresta immediatamente, ma esegue un ultimo ciclo di scrittura sul disco. Questo passaggio finale è essenziale per garantire che tutti gli aggiornamenti recenti delle strutture dati vengano salvati correttamente, evitando la perdita di eventuali modifiche apportate nelle ultime operazioni. Solo dopo aver completato quest'ultima scrittura, il thread termina il proprio task in modo sicuro e ordinato.

Successivamente, viene avviato un altro thread dedicato al calcolo del ranking degli alberghi. Questo thread opera a intervalli di tempo regolari, definiti nel file di configurazione, e si occupa di aggiornare la classifica degli hotel per ogni città presente nel sistema. Il ranking di ciascun hotel è un punteggio complessivo assegnato sulla base delle recensioni lasciate dagli utenti.

Come Funziona il Calcolo del Ranking
Il thread inizia esaminando la lista degli hotel per ciascuna città. Per ogni hotel, raccoglie tutte le recensioni disponibili nel sistema.
Il punteggio di un hotel viene calcolato pesando sia il voto globale, che i voti assegnati alle singole categorie delle recensioni. Un aspetto importante è l'attualità delle recensioni: i punteggi delle recensioni più recenti hanno un peso maggiore rispetto a quelli delle recensioni più vecchie. L'attualità viene calcolata con la formula:

recencyWeight = 1.0 - (0.05 * Math.floor(yearsBetween));

Questa formula riduce il peso di una recensione del 5% per ogni anno trascorso dalla sua pubblicazione. A questo punto, viene calcolata una media ponderata che include sia il punteggio globale sia i punteggi delle singole categorie. A questa media viene sottratta una quantità determinata dalla seguente formula:

quantityWeight = 0.3 - (0.02 * (reviewsHotel.size() -1));

Man mano che aumenta il numero di recensioni, la quantità sottratta diminuisce, riflettendo il fatto che, con un numero elevato di recensioni, aumenta la valutazione del ranking. Quando questa quantità scende sotto lo zero, diventa trascurabile, indicando che il numero di recensioni ha raggiunto un livello tale per cui la quantità non influisce più significativamente sulla valutazione complessiva dell'hotel.

A seguito del calcolo del ranking globale degli alberghi, la classifica della città viene aggiornata di conseguenza. Se si verifica una variazione nelle posizioni degli hotel all'interno di questa classifica, viene eseguita una procedura per informare gli utenti interessati. In particolare, il sistema scorre la lista degli utenti iscritti al servizio di notifica per verificare se qualcuno è registrato per ricevere aggiornamenti relativi a quella specifica città. Per ciascun utente registrato, viene invocato il metodo notifyEvent sull'oggetto remoto, precedentemente esportato al client tramite RMI. Questo metodo invia al client la nuova classifica aggiornata, garantendo che gli utenti siano tempestivamente informati dei cambiamenti. Se l'invocazione del metodo provoca una RemoteException, lo stub viene rimosso dalla lista perchè si considera che il client non è attivo. Inoltre, se la variazione coinvolge la prima posizione della classifica, viene attivato un ulteriore meccanismo di notifica. In questo caso, viene inviato un datagramma (Datagram Packet) tramite un socket UDP al gruppo di multicast, informando tutti gli utenti attualmente loggati.

Al termine di tutte le operazioni preliminari, viene aperto un socket per gestire le connessioni TCP in ingresso. A questo socket viene associato uno specifico indirizzo IP e una porta designata, in seguito configurato in modalità non bloccante.
Grazie all'uso del Selector, il canale di comunicazione può essere registrato su questo selettore per gestire l'accettazione di nuove connessioni. Il thread principale entra in un ciclo continuo, durante il quale si blocca sul metodo select(). Tale metodo attende che almeno uno dei canali registrati sia pronto per un'operazione (ad esempio, per accettare una nuova connessione o per leggere dati in arrivo). Quando un canale è pronto, la select() si sblocca, e il thread principale itera sulle chiavi associate ai canali identificati come pronti.
Per ciascuna chiave, il thread verifica il tipo di evento associato:

1. **Accettazione di una Nuova Connessione:**
    - Se la chiave è pronta per accettare una nuova connessione, il server gestisce l'evento di accettazione restituendo il canale associato alla connessione con il client. Questo canale viene registrato sul selettore per le operazioni di lettura, e come *attachment* viene associato un array di ByteBuffer. Questo array contiene due ByteBuffer: il primo buffer è destinato a contenere la lunghezza del messaggio di richiesta da parte del client, mentre il secondo buffer memorizzerà il contenuto effettivo del messaggio.
2. **Lettura dei Dati dal Canale:**
    - Se la chiave è pronta per la lettura, il server gestisce l'evento recuperando l'attachment associato al canale. Viene eseguita un'operazione di lettura sul canale, con i dati che vengono memorizzati nei ByteBuffer corrispondenti. Dopo aver letto i dati, Una volta completata la lettura, il messaggio ricevuto viene convertito in un JsonObject utilizzando la libreria GSON, consentendo l'estrazione dei campi operation e param.

A questo punto, in base al valore del campo operation, il server identifica l'operazione richiesta e sottomette il task corrispondente all'esecutore del thread pool. Il thread pool è stato configurato utilizzando il metodo newFixedThreadPool(), impostando un numero massimo di thread uguale al

numero di core che la macchina può gestire in parallelo. Questa configurazione è stata scelta per evitare di creare troppi thread rispetto alla capacità della CPU, il che causerebbe un eccessivo overhead dovuto al cambio di contesto.

Terminazione del server causata da eccezione

La procedura di terminazione del server è progettata per garantire che tutte le risorse vengano rilasciate in modo sicuro e che le operazioni in corso vengano completate correttamente prima dell'arresto definitivo.

Fasi della terminazione del server:

1. Cessazione dei Servizi Remoti:
    - Prima di tutto, il server procede alla deregistrazione dei servizi remoti (RMI) precedentemente registrati. Questo avviene tramite il metodo unregisterServices(), che assicura che gli oggetti remoti non siano più accessibili dai client.
2. Chiusura dei Canali di Comunicazione:
    - Successivamente, il server si occupa della chiusura del ServerSocketChannel e del Selector. Questi componenti sono responsabili della gestione delle connessioni TCP e degli eventi di I/O.
3. Interruzione dei Thread di Background:
    - I thread responsabili della scrittura periodica dei dati su disco e del calcolo dei ranking degli hotel, vengono interrotti utilizzando il metodo interrupt(). Questo metodo segnala ai thread di terminare le loro attività in corso. Tuttavia, prima di terminare completamente, i thread eseguono un ultimo ciclo di scrittura per garantire che tutti i dati aggiornati siano salvati in modo sicuro.
4. Arresto del Thread Pool:
    - Il thread pool viene arrestato utilizzando il metodo shutdownNow(), non accettando nuovi task ed eliminando quelli in coda. Successivamente, viene chiamato awaitTermination() per attendere un periodo di tempo definito (5 secondi) al fine di permettere ai thread in corso di completare le loro operazioni in modo ordinato. Se i thread non terminano entro questo intervallo, l'esecutore si arresta forzatamente.
5. Chiusura del Datagram Socket:
    - Infine, viene chiuso il DatagramSocket utilizzato per le comunicazioni UDP, come ad esempio l'invio di notifiche multicast.

CLASSI UTILIZZATE DAL SERVER

RegistrationServiceImpl

Classe che implementa l'interfaccia remota **RegistrationService**. Questa interfaccia definisce il metodo **addRegistration**, il quale viene utilizzato dai client per registrarsi al sistema. Quando un client desidera effettuare una registrazione, invoca questo metodo tramite lo stub dell'oggetto remoto che è stato precedentemente esportato dal server.
L'implementazione di **addRegistration** in **RegistrationServiceImpl** gestisce l'intero processo di registrazione e rappresenta l'implementazione concreta del caso d'uso **Registrazione**.

NotificationServiceImpl

Classe che implementa l'interfaccia remota **NotificationService**. Questa interfaccia definisce il metodo **registerForCallback**, il quale consente ai client di iscriversi al servizio di notifica per gli aggiornamenti del ranking locale delle città di loro interesse.
Quando un client desidera ricevere notifiche su eventuali variazioni della classifica di una città specifica, invoca il metodo **registerForCallback** il quale implementa il caso d'uso **Registrazione al servizio di notifica**. Questo metodo registra lo stub dell'oggetto remoto del client, esportato dal client stesso, all'interno di **hotelCallbacks**.
HotelCallbacks è una struttura dati che associa a ciascuna città una lista di utenti registrati per le notifiche. Grazie a questa associazione, ogni volta che si verifica un cambiamento nella classifica degli hotel di una città, il server può invocare il metodo dell'oggetto remoto del client utilizzando il riferimento allo stub, garantendo così che l'utente riceva tempestivamente l'aggiornamento richiesto.
L'interfaccia NotificationService definisce anche il metodo unregisterForCallback, che implementa il caso d'uso **deregistrazione dal Servizio di Notifica,** permettendo ai client di cancellare la loro iscrizione al servizio di notifica. Questo metodo rimuove il client dalla struttura dati hotelCallbacks, garantendo che non riceva più notifiche relative alle variazioni della classifica della città specifica.

User

Classe che rappresenta un utente registrato. Tra i dati memorizzati in questa classe vi sono:
Username: l'identificativo unico dell'utente all'interno del sistema
Password: la credenziale segreta associata all'utente.
Numero di recensioni effettuate: un contatore che tiene traccia del numero di recensioni che l'utente ha inserito nel sistema, utile per la valutazione della sua attività e partecipazione.
Distintivo: un attributo che può rappresentare un riconoscimento, assegnato in base al numero di recensioni.

LoginTask

È la classe responsabile dell'implementazione del caso d'uso **Login**. Questa classe implementa il metodo run dell'interfaccia Runnable, che esegue la logica di autenticazione. Il processo di autenticazione prevede la verifica delle credenziali dell'utente e la conferma del suo accesso al sistema.
Una volta verificate con successo la registrazione dell'utente e le sue credenziali, la classe associa il canale di comunicazione del client allo username dell'utente all'interno della struttura dati ConcurrentHashMap, denominata socketUserMap. Questa associazione permette all'utente di accedere al sistema contemporaneamente da più dispositivi, rispecchiando il comportamento delle applicazioni moderne.
Inoltre, conservando il riferimento al canale, il sistema è in grado di identificare quale utente ha chiuso una connessione, semplicemente risalendo al canale corrispondente.
L'uso di una ConcurrentHashMap è stato scelto per garantire l'efficienza delle operazioni di accesso concorrente senza la necessità di bloccare l'intera mappa durante l'accesso da parte di più thread.

LogoutTask

È la classe designata all'implementazione del caso d'uso **Logout**. Questa classe implementa l'interfaccia Runnable, e all'interno del metodo run viene eseguita la procedura di disconnessione dell'utente dal sistema.
Nello specifico, il metodo run si occupa di rimuovere l'associazione chiave-valore tra il canale di comunicazione e lo username dell'utente nella struttura dati utilizzata per la gestione delle sessioni attive. Successivamente, viene inviata una risposta all'utente, indicante l'esito dell'operazione. Se la rimozione dell'associazione è avvenuta con successo, viene confermata la corretta esecuzione della richiesta di logout; in caso contrario, l'utente viene informato dell'eventuale fallimento.

ShowMyBadgeTask

È la classe responsabile dell'implementazione del caso d'uso **Visualizzazione del Distintivo**. Questa classe implementa l'interfaccia Runnable, e all'interno del metodo run esegue la procedura per recuperare il distintivo associato all'utente che ha richiesto l'operazione.
Inizialmente, il sistema verifica che l'utente abbia effettuato il login e sia correttamente registrato. Questa verifica è essenziale per garantire che solo gli utenti autenticati possano accedere alle informazioni personali. Se queste condizioni sono soddisfatte, il sistema procede a recuperare il distintivo dell'utente.
Il distintivo, simbolo delle recensioni effettuate dall'utente, viene poi inviato come parte della risposta. Tuttavia, se l'utente non ha ancora effettuato alcuna recensione e quindi non possiede un distintivo, il sistema invia un messaggio informativo.

SearchHotelTask

È la classe incaricata dell'implementazione del caso d'uso **Ricerca Hotel**. Questa classe implementa l'interfaccia Runnable e, all'interno del metodo run, esegue l'operazione di ricerca di un hotel basandosi sul nome della città e dell'hotel forniti dall'utente.
La ricerca avviene in tempo costante, grazie all'utilizzo di una struttura dati altamente efficiente: una HashMap che associa il nome della città a un'altra HashMap, la quale a sua volta mappa i nomi degli hotel presenti in quella città. Questo approccio garantisce rapidità nella ricerca e ottimizza le performance, anche in presenza di un gran numero di hotel.
Una volta completata la ricerca, l'utente viene informato sull'esito dell'operazione: se l'hotel richiesto è stato trovato, vengono restituite tutte le informazioni rilevanti su di esso; in caso contrario, viene inviata una risposta che informa l'utente dell'esito negativo della ricerca.

SearchAllHotelInCity

È la classe responsabile dell'implementazione del caso d'uso **Visualizzazione Classifica**. Questa classe implementa l'interfaccia Runnable e, nel metodo run, si occupa di recuperare la classifica aggiornata degli hotel per una città specifica.
Utilizzando il nome della città fornito dall'utente, il sistema restituisce la classifica calcolata nell'ultimo aggiornamento del ranking. Se la città è presente nel sistema, l'utente riceverà le informazioni relative alla classifica degli hotel situati in quella località. In caso contrario, il sistema rispenderà dell'impossibilità di soddisfare la richiesta a causa dell'assenza della città nel database.

InsertReviewTask

È la classe responsabile dell'implementazione del caso d'uso **Inserimento Recensione**. Questa classe implementa l'interfaccia Runnable e, all'interno del metodo run, gestisce l'inserimento di una recensione relativa a un hotel nel database delle recensioni.
Il processo inizia con la verifica della validità dei punteggi forniti dall'utente, dell'esistenza dell'hotel nel sistema e dell'autenticazione dell'utente. Se tutte le condizioni sono soddisfatte, la recensione viene aggiunta alla lista delle recensioni associate a quell'hotel, conservata in una struttura dati di tipo HashMap.
Ho scelto di utilizzare una HashMap anziché una ConcurrentHashMap per la gestione delle recensioni degli hotel, in quanto le operazioni di inserimento e aggiornamento comportano il blocco e la sincronizzazione di tutta la struttura. La ConcurrentHashMap, pur offrendo operazioni thread-safe, non garantiva sufficiente protezione.
Inoltre, il numero di recensioni effettuate dall'utente viene aggiornato e il distintivo dell'utente viene modificato di conseguenza, riflettendo l'inserimento della nuova recensione. Infine viene informato l'utente se l'operazione è andata a buon fine.

DHKeyExchangeTask

La classe DHKeyExchangeTask è responsabile della gestione dell'effettivo scambio di chiavi tra client e server utilizzando l'algoritmo Diffie-Hellman (DH). Implementa l'interfaccia Runnable, e all'interno del metodo run(), esegue l'algoritmo dello scambio delle chiavi, il quale permette di stabilire una chiave di cifratura sicura attraverso l'uso di valori pubblici e privati, senza necessità di trasmettere direttamente la chiave segreta.
Utilizzando la chiave pubblica del client, la classe calcola una chiave segreta che sarà utilizzata per decifrare i dati sensibili inviati dal client. Questa chiave segreta viene memorizzata in una ConcurrentHashMap, associata a un identificatore univoco (UUID) generato appositamente per ciascun client. L'UUID, associato alla chiave segreta, viene inviato al client insieme alla chiave pubblica del server. Questo permette al client di calcolare la propria chiave segreta e utilizzare tale chiave per cifrare i dati sensibili prima di inviarli al server. L'UUID sarà incluso nei parametri quando ci sono dati cifrati per consentire al server di recuperare la chiave di decifratura corretta.

RemoveChannelTask

La classe **RemoveChannelTask** gestisce la rimozione del canale associato a un utente nel caso in cui la connessione venga interrotta. Implementa l'interfaccia Runnable.

Nel metodo run, la classe si occupa di rimuovere il canale dell'utente dalla struttura dati socketUserMap, una ConcurrentHashMap che associa ciascun canale di comunicazione allo username dell'utente.

SINCRONIZZAZIONE

Per gestire la concorrenza e la sincronizzazione delle strutture dati, ho utilizzato ReadWriteLock. Questa scelta è stata dettata dall'efficienza di questo meccanismo rispetto all'uso delle lock implicite.
La ReadWriteLock consente una sicnronizzazione ottimale delle operazioni di lettura e scrittura sui dati condivisi. Quando un thread deve accedere ai dati in modalità di sola lettura, acquisisce la lock di lettura (ReadLock). Questo approccio permette a più thread lettori di accedere contemporaneamente ai dati, migliorando significativamente la concorrenza e le prestazioni nelle

situazioni in cui le operazioni di lettura sono prevalenti. D'altra parte, quando un thread deve eseguire una modifica ai dati, acquisisce una lock di scrittura (WriteLock). Questo lock garantisce che solo un thread scrittore possa accedere ai dati, impedendo a qualsiasi altro thread di accederci.
Ho evitato l'uso dei monitor con i metodi synchronized perchè ogni volta che un thread worker del pool di thread deve eseguire una richiesta, viene creata una nuova istanza della classe designata per gestire quella richiesta. Poiché le lock implicite dei monitor (synchronized) sono associate all'istanza dell'oggetto, due thread worker che eseguono la stessa operazione su istanze distinte non condividono la stessa lock. Di conseguenza, entrambi i thread potrebbero accedere contemporaneamente alle stesse strutture dati, senza alcuna mutua esclusione, il che potrebbe portare a race condition e inconsistenza dei dati.

Inoltre, poiché classi diverse potrebbero accedere alle medesime strutture dati condivise, è essenziale utilizzare un meccanismo di locking condiviso tra queste classi.

Per garantire una sincronizzazione adeguata e prevenire l'accesso concorrente non sicuro, passo una ReadWriteLock come parametro ai costruttori delle classi che gestiscono le richieste. In questo modo, tutte le classi e i thread che accedono alle strutture dati condivise utilizzano la stessa lock, assicurando una corretta mutua esclusione e mantenendo la coerenza dei dati durante l'accesso concorrente.

DESCRIZIONE FLUSSO DI ESECUZIONE CLIENT

Il programma main del client è progettato per avviare e gestire l'interazione iniziale e continua con il server, oltre a configurare l'ambiente per la comunicazione sicura e la gestione remota degli oggetti. Di seguito è illustrato il flusso operativo del programma.

1. **Lettura del File di Configurazione**: Il client inizia il suo processo leggendo un file di configurazione, che contiene tutti i parametri essenziali per il suo corretto funzionamento.
    - **multicast_address=226.226.226.226:** l'indirizzo IP 226.226.226.226 è un indirizzo IP di classe D, riservato per la comunicazione multicast, utilizzato per consentire al client di ricevere messaggi inviati a questo gruppo.
    - **multicast_port=8989**: porta associata al gruppo di multicast
    - **registry_port=3333:** porta associata al registry
    - **callback_obj = 0**: porta per l'oggetto callback
    - **server_port_tcp=8000:** questo parametro specifica la porta TCP sulla quale il server è in ascolto per le connessioni in ingresso dal client.
    - **socket_tcp_timeout = 10000:** timeout per la ricezione di una risposta dal server durante una connessione TCP.
    - **debug = false:** questo parametro specifica se il client deve funzionare in modalità debug (true) o in modalità release (false).
    - **P_number=9973:** questo parametro rappresenta un grande numero primo pubblico utilizzato nell'algoritmo di scambio di chiavi Diffie-Hellman.
    - **G_generator=7**
    - **Generatore per Diffie-Hellman:** questo parametro rappresenta il generatore utilizzato in combinazione con P_number per calcolare le chiavi intermedie durante l'algoritmo Diffie-Hellman.

Se durante questa fase si verifica un'eccezione, il client viene immediatamente terminato, in quanto le informazioni necessarie alla sua esecuzione non sono recuperate correttamente.

2. **Connessione al Server**: Dopo aver caricato con successo la configurazione, il client tenta di stabilire una connessione con il server tramite un SocketChannel. In caso di mancato successo nella connessione, all'utente viene offerta la possibilità di scegliere tra due opzioni: ritentare la connessione o terminare l'esecuzione del programma utilizzando il comando "esci". Il processo viene ripetuto fino a quando non si stabilisce una connessione valida o l'utente decide di interrompere l'operazione.
3. **Scambio di Chiavi Diffie-Hellman**: Se la connessione al server ha esito positivo, il client procede con la prima comunicazione, che consiste nello scambio delle chiavi pubbliche tra client e server. Questo scambio è realizzato utilizzando l'algoritmo di Diffie-Hellman (DH), che permette a entrambe le parti di calcolare una chiave privata condivisa. Questa chiave sarà utilizzata per cifrare e decifrare i dati sensibili scambiati durante la sessione. Anche lo scambio delle chiavi avviene tramite la costruzione e l'invio di una richiesta sul canale di comunicazione.
4. **Costruzione delle Richieste del Client**: Per ogni operazione richiesta dall'utente, il client costruisce una stringa in formato JSON che rappresenta la richiesta da inviare al server. Questa stringa contiene un campo operation, che specifica l'operazione desiderata, e un campo param, una mappa chiave-valore dove sono memorizzati i parametri necessari per eseguire l'operazione richiesta.
5. **Esportazione e Recupero di Oggetti Remoti tramite RMI**: Successivamente, il client si occupa di esportare e recuperare oggetti remoti utilizzando Java RMI (Remote Method Invocation). Il client esporta lo stub di un oggetto remoto che implementa l'interfaccia NotifyEventInterface, attraverso la classe NotifyEventImpl. Questo oggetto remoto è utilizzato dal server per notificare il client riguardo gli aggiornamenti dei vari ranking locali a cui l'utente è iscritto. Inoltre, il client recupera gli stub degli oggetti remoti NotificationService e RegistrationService, che permettono di invocare metodi remoti per registrare un nuovo utente o per iscriversi/deregisrarsi ai servizi di notifica.
6. **Creazione dell'Oggetto Multicast**: Infine, il client crea un oggetto multicast, che sarà successivamente utilizzato per unirsi a un gruppo multicast. Questo permette al client di ricevere e inviare comunicazioni all'interno di un gruppo di utenti collegati.

Al corretto avvio del client, viene visualizzato un menù di opzioni nella Command Line Interface (CLI). L'utente è invitato a selezionare un'operazione tra quelle disponibili, inserendo il numero corrispondente all'operazione desiderata. Fino a quando non viene immesso un input valido, il sistema continuerà a richiedere l'inserimento. Una volta ricevuto un input valido, il processo di gestione della richiesta viene avviato.
Il client richiede all'utente di fornire i parametri necessari per eseguire l'operazione selezionata. La richiesta viene quindi costruita utilizzando l'operazione selezionata e i parametri forniti dall'utente, e successivamente serializzata in formato JSON. Per l'invio della richiesta sul canale di connessione socket, vengono utilizzati due ByteBuffer: il primo per memorizzare la lunghezza della richiesta e il secondo per contenere i dati della richiesta stessa. Entrambi i buffer vengono inviati attraverso il canale.
Dopo l'invio della richiesta, il client entra in uno stato di attesa bloccante sulla chiamata receive del SocketChannel, in attesa di una risposta da parte del server. Se la risposta viene ricevuta entro un intervallo di 10 secondi, il client si sblocca e legge la lunghezza della risposta e i relativi dati, anch'essi memorizzati in un ByteBuffer. Nel caso in cui la risposta non venga ricevuta entro il limite

di 10 secondi, la chiamata receive si sblocca automaticamente, evitando di rimanere in attesa indefinitamente.
La stringa della risposta viene quindi deserializzata in un oggetto Response. Attraverso i metodi della classe Response, il client può estrarre il codice di stato, che indica l'esito dell'operazione, eventuali risultati e un messaggio esplicativo. Infine, il menù viene ripristinato, consentendo all'utente di effettuare ulteriori operazioni.

Terminazione del client

Quando l'utente sceglie di uscire dall'applicazione selezionando l'opzione corrispondente nel menù, il client inizia il processo di deregistrazione dal servizio di notifica per tutte le città a cui era stato precedentemente registrato. Successivamente, effettua l'unexport dell'oggetto remoto precedentemente esportato.

Il passo successivo prevede l'interruzione del MulticastSocket, inviando un segnale al thread che è in attesa di ricevere dati attraverso la receive(). A questo punto, un pacchetto contenente la stringa "EXIT_MULTICAST_GROUP" viene inviato tramite la DatagramSocket per sbloccare il thread dalla receive() e permettere l'uscita dal ciclo di ricezione. Il client abbandona quindi il gruppo multicast e chiude il MulticastSocket.

Infine, viene terminata la connessione TCP, inviando un segmento con il flag FIN impostato a 1, completando così il processo di chiusura del socket.

CLASSI USATE DAL CLIENT

SocketClient

Questa classe è responsabile della gestione delle comunicazioni TCP con il server. Quando viene creata un'istanza della classe, si apre immediatamente una connessione utilizzando l'indirizzo IP e la porta associati al socket del server.
Il metodo sendRequest viene utilizzato dal client per inviare una richiesta, sfruttando due ByteBuffer: uno dedicato a contenere la lunghezza della richiesta e l'altro per la richiesta vera e propria.
Il metodo readResponse attende la risposta del server, che viene successivamente deserializzata in un oggetto Response. Da questo oggetto, verranno estratti e letti il codice di stato, eventuali dati associati e il messaggio esplicativo.
Nel caso si verifichi una IOException, l'eccezione viene propagata e il sistema tenterà di ristabilire la connessione, qualora fosse necessario.

MulticastManager

La classe MulticastManager è responsabile della gestione delle comunicazioni multicast all'interno del sistema. Quando un utente effettua il login, si iscrive al gruppo multicast specificato, identificato da un indirizzo IP e una porta predefiniti. Questa iscrizione consente all’utente di

ricevere aggiornamenti in tempo reale riguardanti le prime posizioni nei ranking locali, perciò viene avviato un thread dedicato che rimane in ascolto continuo sulla MulticastSocket per ricevere pacchetti UDP contenenti gli aggiornamenti dei ranking.

Quando l'utente decide di effettuare il logout o di chiudere l'applicazione, MulticastManager esegue una serie di operazioni per terminare correttamente le comunicazioni multicast:

1. **Interruzione del Thread di Ascolto:** Il thread responsabile della ricezione dei pacchetti UDP viene interrotto, e viene inviato un pacchetto contenente un messaggio di uscita.
2. **Abbandono del Gruppo Multicast:** Il client si disiscrive dal gruppo multicast.
3. **Chiusura del MulticastSocket:** Viene chiuso il MulticastSocket, liberando le risorse di rete associate e assicurando che non vi siano connessioni pendenti.

NotifyEventImpl

La classe **NotifyEventImpl** implementa l'interfaccia remota **NotifyEventInterface**, che definisce il metodo remoto notifyEvent. Questo metodo viene invocato dal server per inviare notifiche al client tramite lo stub dell'oggetto esportato.

Quando il metodo notifyEvent viene chiamato, viene mostrata la classifica aggiornata della città a cui l'utente si è registrato per ricevere notifiche. Contestualmente, l'aggiornamento viene memorizzato in una struttura dati che tiene traccia di tutte le città e delle rispettive classifiche a cui l'utente è iscritto.

Nel momento in cui l'utente si deregistra dalle notifiche relative a una specifica città, sia la città che la sua classifica vengono rimosse dalla struttura dati. Al contrario, quando l'utente si registra per ricevere notifiche da una nuova città, questa viene aggiunta alla struttura con una lista vuota, pronta per essere popolata non appena arriverà il primo aggiornamento della classifica.

Request

La classe **Request** rappresenta una richiesta inviata al server. Gli attributi principali della classe sono operation e param, che indicano rispettivamente il tipo di operazione da eseguire e i parametri necessari per eseguirla.

Quando l'utente desidera avviare un'operazione, il client crea un'istanza di **Request** contenente il nome dell'operazione da eseguire e i relativi parametri. Questi attributi vengono successivamente inviati al server, che processa la richiesta in base all'operazione specificata e ai dati forniti.

CLASSI CONDIVISE DA CLIENT E SERVER

Hotel

La classe Hotel rappresenta un'entità di tipo hotel, memorizzando tutte le informazioni rilevanti per ciascun hotel presente nel sistema. Questa classe implementa l'interfaccia Serializable, che permette di serializzare e deserializzare gli oggetti di tipo Hotel, permettendo di notificare la lista degli hotel al client. Gli Attributi di questa classe sono:

serialVersionUID: identificatore univoco della versione della classe che consente di verificare la compatibilità durante il processo di deserializzazione.
id: Un identificatore univoco per ciascun hotel.
name: Il nome dell'hotel.
description: Una descrizione dettagliata dell'hotel.
city: La città in cui si trova l'hotel.
phone: Il numero di telefono dell'hotel.
services: Una lista dei servizi offerti dall'hotel (es. Wi-Fi gratuito, piscina, palestra, ecc.), che aiuta a descrivere le comodità e i vantaggi disponibili per gli ospiti.
rate: Il punteggio globale dell'hotel, calcolato dall'algoritmo di ranking in base alle recensioni degli utenti. Questo punteggio riflette la qualità complessiva dell'hotel.
ratings: I punteggi delle singole categorie (come pulizia, posizione, servizi e qualità), anch'essi calcolati dall'algoritmo di ranking.

Review

La classe Review rappresenta una recensione lasciata da un utente per un determinato hotel, includendo sia il punteggio globale assegnato sia le valutazioni specifiche per diverse categorie. La classe contiene una serie di attributi.

- **user**: username dell'utente che ha lasciato la recensione.
- **idHotel**: L'identificatore univoco dell'hotel a cui si riferisce la recensione.
- **nameHotel**: Il nome dell'hotel oggetto della recensione.
- **globalscor**: Il punteggio globale assegnato dall'utente all'hotel.
- **data**: La data in cui è stata lasciata la recensione.
- **ratings**: Un oggetto di tipo Categories che contiene i punteggi dettagliati assegnati dall'utente a diverse categorie dell'hotel (come pulizia, servizio, posizione e qualità).

Pair

La classe Pair è una struttura dati generica che rappresenta una coppia di oggetti generici.
Attributi della Classe

- **first**: Rappresenta il primo elemento della coppia.
- **second**: Rappresenta il secondo elemento della coppia.

Response

La classe Response è progettata per rappresentare una risposta che viene inviata dal server a un client. Questa risposta include un codice di stato e un messaggio.
Attributi

- **statusCode**:
    - È un intero che rappresenta il codice di stato della risposta (ho utilizzato i codici del protocollo http).
- **message**:

- Un'istanza della classe interna Message, che rappresenta il contenuto effettivo della risposta. Message contiene il risultato dell'operazione richiesta dal client e un corpo testuale che può fornire ulteriori dettagli.

Nella classe ho inserito il metodo **writeResponse:** questo metodo è responsabile di inviare la risposta al client tramite un SocketChannel, utilizzando il canale associato alla SelectionKey.
La risposta viene convertita in una stringa JSON utilizzando la libreria Gson, prima di inviarla al client, il server manda la lunghezza del messaggio. Questo permette al client di sapere quanto è lungo il messaggio che sta per ricevere. Una volta inviata la lunghezza, il server invia il messaggio JSON vero e proprio.

SecurityClass

La classe SecurityClass è utilizzata per la gestione della sicurezza all'interno di un sistema di comunicazione, in particolare per l'applicazione del protocollo Diffie-Hellman (DH) e la cifratura delle password scambiate tra client e server. La sicurezza delle comunicazioni è garantita attraverso due principali meccanismi: la generazione di chiavi segrete utilizzando il protocollo DH e la cifratura dei dati sensibili mediante l'algoritmo AES.