DOCUMENTAZIONE TECNICA

Matteo Moi 737574 Varese

Informazioni generali

Esportare in file jar eseguibili

Nome file : EA_DB_Server || Path : \\Eat_Advisor\\EatAdvisor_DB || Main : EA_Master_Server.java
Nome file : EA_Cliente || Path : \\Eat_Advisor\\EatAdvisor_Client || Main : Cliente.java
Nome file : EA_Ristorante || Path : \\Eat_Advisor\\EatAdvisor_Restaurant || Main : Ristorante.java

Le applicazioni Ristorante e Cliente sono state sviluppate tramite GUI ed esportate come file jar eseguibile nelle rispettive directory.
Il progetto è implementato integrando una terza applicazione EatAdvisor_DB(esportata in jar come EA DB_server) la quale ha funzione di database(embedded) e di server. Ciò è stato fatto per applicare le conoscenze apprese durante il secondo anno di corso, oltre a avvicinare le applicazioni ad essere il più realistiche possibili(ovvero rendendole distribuite e preimpostando l'utilizzo delle app Cliente e Ristorante su macchine differenti , a tale scopo sono state utilizzate le tecnologie RMI e Socket).
L'avvio dell'app EA_DB_server è stato reso del tutto automatico, essa verrà avviata in seguito all'apertura dell'app Cliente o Ristorante(evitando così inutile complessità per l'utente);è comunque possibile avviarla manualmente.Nel caso in cui sia necessario l'app EA_DB_server va chiusa tramite gestione risorse andando a terminare il processo "OpenJDK Platform Binary".
E' stato preferito non creare i package ristoratori e clienti per favorire la comunicazione tra le applicazioni.

Esecuzione da IDE

avviare prima EA_Master_Server (Progetto EatAdvisor_DB) e poi Cliente (Progetto EatAdvisor_Client) e/o Ristorante (Progetto EatAdvisor_Restaurant). Il Database (se le app non vengono spostate) è condiviso con le applicazioni .jar

Applicazione Database/Server

Classi :

- **EA_Master_Server**
  Tramite richiamo di metodo db_start_conn() della classe **EA_DB** avvia o inizializza (dipende se già esistente o meno) la connessione al database embedded(chiamato EM_DB) oltre a dare come argomento per il metodo new_table() sempre della classe **EA_DB** le stringe per la creazione delle due tabelle Cliente & Restaurant.
  Poi esegue create_registry() il quale rende remota (RMI) la classe **EA_Server_Skeleton** e ne fa il rebind.
  Infine esegue create_single_server() che mette il server in attesa(Socket) di una richiesta da parte dall'app Cliente o Ristorante; quando ricevuta crea una nuova istanza di **EA_Single_Server** che si occuperà di gestire la richiesta. Dopo aver chiuso il socket il server si rimette in attesa di una nuova richiesta.[Questa scelta della continua attesa è stata fatta nell'ottica di un maggiore realismo dove il server e il database sono sempre attivi]

- **EA_Single_Server**[extends Remote]
  Ogni istanza della classe si occupa di una sola singola richiesta inviata dalle app client(Cliente o Ristorante). Tramite recognize_client() la classe impiega i socket per leggere il tipo di richiesta ricevuta che può essere :
  - gestione iscrizione cliente o ristorante
  - gestione accesso utente
  - gestione ricerca ristorante
  - gestione recensione

  Dell'effettiva gestione se ne occuperà la classe **EA_DB, EA_Single_Server** richiamerà i suoi metodi e gli darà come argomento i dati ricevuti dall'utente e contenuti nell'array tupla il cui valore varia in base a cosa ritorna il metodo client_data() il quale scompone la stringa inviata dal client (tramite Socket) e inserisce in ogni index dell'array tupla il dato singolo(es. gestione accesso utente : stringa ricevuta(id password) = mmoi 2020 -> tupla[0]=mmoi tupla[1]=2020).
  Dopo aver richiamato i metodi **EA_Single_Server** viene interrotto.

- **EA_DB**
  La classe fornisce i metodi per gestire il database e le richieste dei client
  - db_start_conn() ricerca il file find_me.txt(situato nella directory EatAdvisor_DB) e ne ottiene il canonical path il quale però se l'app EA DB_server viene avviata tramite l'app Cliente o Ristorante risulta sbagliato, eventualmente quindi viene modificato.
    Viene poi avviata la connessione al database(se non già presente).[il database embedded è creato e gestito tramite i drive derby ed è basato su apache]
  - new_table(String table_str) crea (nel db) se non già esistenti le tabelle Cliente e Restaurant.
  - cliente_registration(String[] data) & restourant_registration(String[] data) iscrivono nelle rispettive table i client che ne hanno fatto richiesta
  - check_data_login(String[] data) confronta i dati inviati dall'app Cliente per effettuare il login con quelli presenti sul db avvisando poi (tramite socket) se l'accesso è consentito o meno
  - insert_comment(String[] comment) inserisce/aggiunge (nel caso in cui ce ne siano altri) all'interno della table Restaurant il commento (contenente id,numero_stelle,commento) effettuato da un utente.
  - search_restaurant(String[] query) ricerca all'interno della table Restaurant i ristoranti che hanno uguali dati a quelli inseriti dall'utente nell'app Cliente (per effettuare la ricerca)
  - show_table(String query) permette al programmatore (che esegue da IDE) di visualizzare su console il contenuto delle table del db così da avere un feedback sulle interazioni tra le applicazioni.

- **EA_Server_Skeleton** [extends UnicastRemoteObject implements Server_Client_Int]
  è la classe che viene resa remota(tramite RMI) così da permettere all'app Client di ottenere i risultati della ricerca del ristorante; essi verranno conservati (fino a nuova ricerca) nell'array restaurant_list della classe EA_DB, tramite l'implementazione del metodo get_restaurant() tali dati saranno disponibili all'app Cliente.

- **Server_Client_int** [extends Remote]
  è l'interfaccia che dichiara il metodo get_restaurant();

Applicazione Cliente

Classi :

- **Cliente**
  Verifica tramite Socket se la connessione al server è aperta, in caso contrario ottiene l'absolute path del file EA_DB_Server.jar che però risulterà sbagliato con EatAdvisor_Client al posto di EatAdvisor_DB nel path, per cui verrà sostituito e corretto. Il path verrà poi utilizzato per avviare tramite Desktop.getDesktop().open(new File(path)) il file EA_DB_Server.jar che a quel punto avvierà la connessione e il programma verrà eseguito normalmente[questa operazione è possibile solo se le cartelle EatAdvisor_DB e EatAdvisor_Client sono all'interno della stessa directory{per questioni di ottenimento del path}]
  La classe cliente inoltre fornisce due metodi frame_back_log() & frame_back_reg() usati dalla classe logIn e Registration per rendere nuovamente visibile il frame della classe Cliente.

- **Cliente_Sender**
  Gestisce le richieste effettuate dall'utente tramite la GUI in particolare
  - gestione iscrizione cliente invia* i dati al server per riconoscere la richiesta e per eseguirla.
  - gestione accesso utente richiede al server* di confrontare i dati inseriti dall'utente per il login e quelli presenti nel db se c'è una corrispondenza all'utente è permesso accedere al frame LogIn
  - gestione ricerca ristorante invia al server* i dati necessari per effettuare la ricerca nel db del ristorante/i.
  - gestione recensione invia al server* la recensione effettuata dall'utente su uno specifico ristorante.
  - gestione accesso senza login permette all'utente di accedere al frame login senza accedere con le credenziali ma verrà tolta la possibilità di fare recensioni(rendendo i componenti setEnabled(false)).
  - send_data(String tupla) invia * al server i dati contenuti nella stringa tupla.
  - recive_data() permette di leggere* la risposta del server riguardante i dati inseriti dall'utente per il login.

  *tramite socket

- **Registration**
  Permette all'utente di inserire i dati per la sua iscrizione, i quali verranno inseriti tutti nella stringa tupla e poi inviati al server tramite la classe **Cliente_Sender.**

- **LogIn**
  Elementi :
  - **btnCerca** crea una connessione socket con il server poi salva in una stringa tutti i dati inseriti dall'utente per ricercare il ristorante e li invia al server tramite **Cliente_Sender.**
    Effettua poi un LocateRegistry.getRegistry(1099) per fare il lookup del registro "Ristorante" tramite il quale poi si chiamerà il metodo remoto get_restaurant() che ritornerà un array contenente i risultati della ricerca del ristorante effettuata sul db(l'array conterra tutti i campi del ristorante) verrà poi reimpostato il modello list_result il quale andrà a contenere e a mostrare nell'interfaccia una lista contenente i nomi dei ristoranti corrispondenti ai criteri di ricerca inseriti dall'utente precedentemente.

  - **btnVedi** ottiene il nome del ristorante selezionato dall'utente nella list_result e mostra nell'info_restaurant tutti i dati di quello specifico ristorante.

- **btnInvia** permette all'utente registrato di recensire il ristorante selezionato nella list_result inviando il numero di stelle assegnato, l'eventuale commento e il suo nickname al server(tramite socket)

- **Server_Client_Int**[extends Remote]
  Permette di utilizzare il metodo remoto della classe **EA_Server_Skeleton** dell'app **EA_DB_server.**

Applicazione Ristorante

Classi :

- **Ristorante**
  Verifica tramite Socket se la connessione al server è aperta, in caso contrario ottiene l'absolute path del file EA_DB_Server.jar che però risulterà sbagliato con EatAdvisor_Restaurant al posto di EatAdvisor_DB nel path, per cui verrà sostituito e corretto. Il path verrà poi utilizzato per avviare tramite Desktop.getDesktop().open(new File(path)) il file EA_DB_Server.jar che a quel punto avvierà la connessione e il programma verrà eseguito normalmente[questa operazione è possibile solo se le cartelle EatAdvisor_DB e EatAdvisor_Restaurant sono all'interno della stessa directory{per questioni di ottenimento del path}]
  Elementi :
  - **btnConferma** salva tutti i dati inseriti dall'utente (nelle apposite textfield) in una stringa la quale verrà inviata al server (per l'iscrizione del ristorante) tramite Restaurant_Sender
  - **btnDelete** pulisce tutte le textfield

- Restaurant_Sender
  Crea un socket per inviare i dati di iscrizione del ristorante al server, ciò viene effettuato tramite il metodo send_data(String tupla)