

# MANUALE TECNICO



Università degli studi dell'Insubria - Laurea triennale in Informatica

## **Progetto laboratorio B: Emotional Songs with Database and client-server**

Candiani Valerio, matricola: 750632 VA

Candiani Luca, matricola: 749717 VA

Airaghi Luca, matricola: 749043 VA

Mammi Matteo, matricola: 750714 VA

## REQUISITI MINIMI:

### INSTALLAZIONE:

**Cpu:** intel core duo o superiori.

**Ram:** 4GB o superiori

**Hard Disk:** nessun requisito minimo

**Java installato (meglio se ultima versione retro compatibile)**

Applicazione sviluppata e testata su Sistema Operativo Windows 10® e Windows 11® utilizzando Java SE Development kit 19.0.2, Java Jre versione 1.8.0\_361, pgAdmin 4 version 7.5 e PostgreSQL 15, pertanto se ne raccomanda l'utilizzo e se ne assicura la completa funzionalità e compatibilità esclusivamente con le configurazioni sopra citate.

Tuttavia è possibile provare l'esecuzione dell'applicazione su altri sistemi operativi Windows® (e versioni Java) antecedenti o successivi a quelli raccomandati oppure utilizzando altri SO come MacOS®, anche se la corretta funzionalità non è garantita o potrebbero verificarsi errori durante l'apertura della stessa.

Su Sistemi UNIX non è garantita la funzionalità.

### SETUP AMBIENTE:

L'applicazione funzionerà solamente avendo installato il programma "Java" che è possibile scaricare e installare direttamente dal sito del produttore. Bisognerà avere installato anche il java JDK (JDK utilizzato: 19.0.2).

Dovrà essere installato anche il programma pgAdmin (utilizzata la versione 4) che è possibile scaricare dal sito del produttore (<https://www.pgadmin.org/download/>). Una volta scaricato e seguita l'installazione dovremo creare un nuovo server tramite l'apposita funzione "Add new server":

Register - Server

General Connection Parameters SSH Tunnel Advanced

Name ⓘ

Server group ⌵ Servers

Background ✕

Foreground ✕

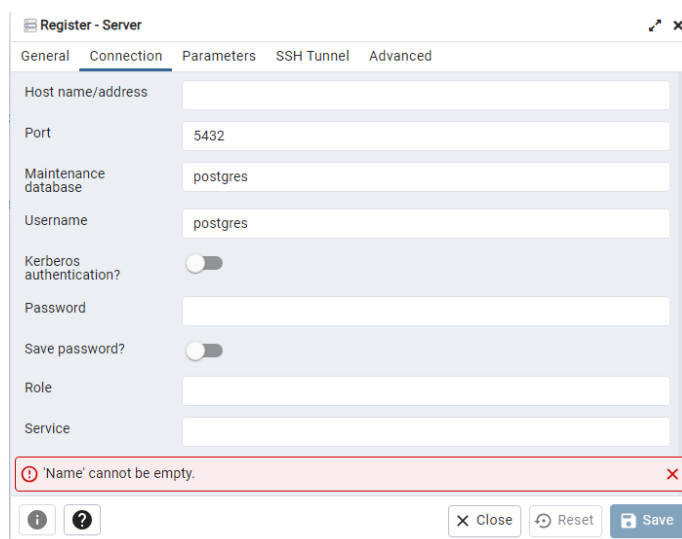
Connect now? ⓘ

Comments

ⓘ ? ✕ Close ↺ Reset 💾 Save

ⓘ 'Name' cannot be empty. ✕

sarà necessario assegnare un nome al server, uno username e una password che poi saranno utilizzati nell'applicazione per effettuare il collegamento al database.



Per poter far comunicare in modo corretto l'applicazione con il Database sarà necessario scaricare ed installare anche il driver postgresSQL che sarà disponibile e scaricabile gratuitamente e direttamente dal sito del produttore (<https://www.postgresql.org/ftp/pgadmin/pgadmin4/v7.5/windows/>)

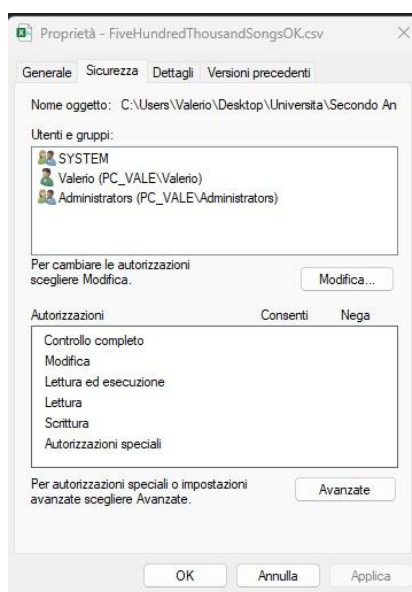
## POPOLARE IL DATABASE AL PRIMO AVVIO:

L'applicazione provvederà ad importare in automatico un elenco predefinito di canzoni tramite un file precaricato chiamato "FiveHundredThousandSongsOK.csv" situato nella cartella "src\main\resources" del progetto.

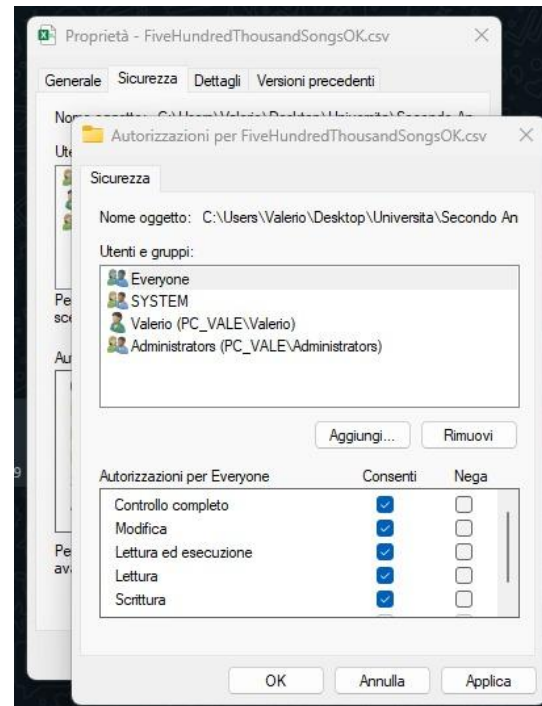
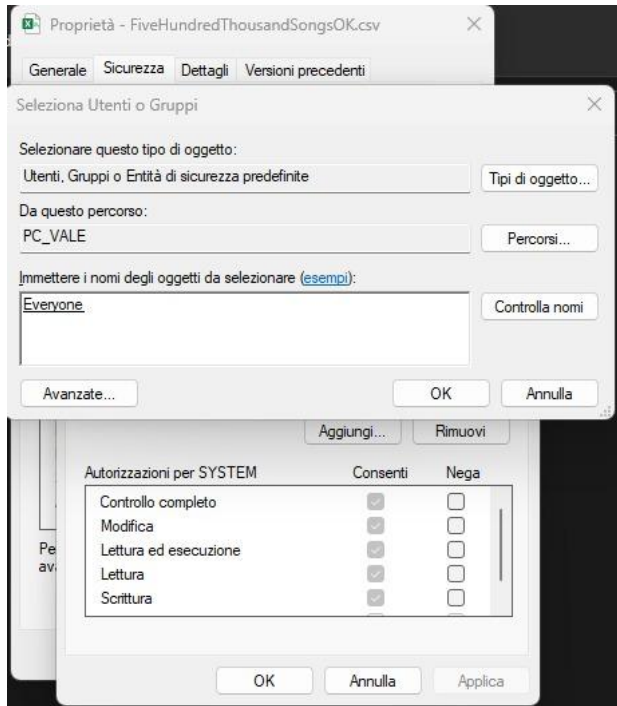
**Nel caso** in cui si dovessero verificare problemi nell'apertura del file csv all'avvio dell'applicazione lato client, per assicurarsi che il file venga correttamente caricato, sarà necessario assegnare i giusti permessi al file che altrimenti ne impedirebbero l'utilizzo (qualsiasi modifica al file potrebbe compromettere l'esecuzione dell'applicazione).

Di seguito illustrata la procedura di modifica dei permessi al file (su dispositivo windows):

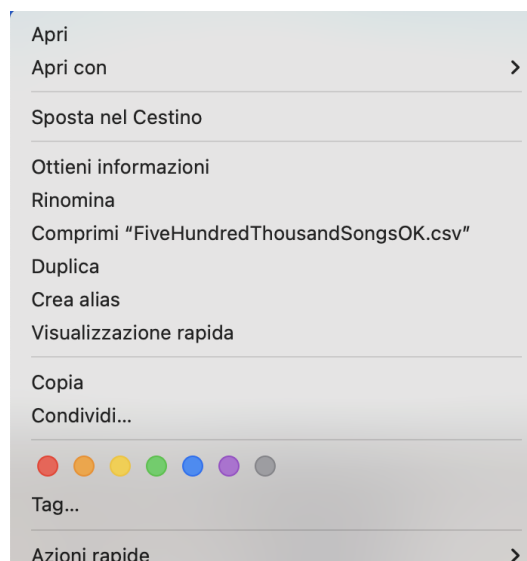
- per iniziare posizionarsi sul file contenente l'elenco delle canzoni in formato csv, premere tasto destro, selezionare la voce "Proprietà" e una volta aperto il menù scegliere il tab "Sicurezza"



ora spostarsi sulla voce “modifica” relativa ai permessi di sicurezza, cliccare sul bottone “Aggiungi” e scegliere come utente “Everyone” avendo cura di selezionare tutte le autorizzazioni elencate



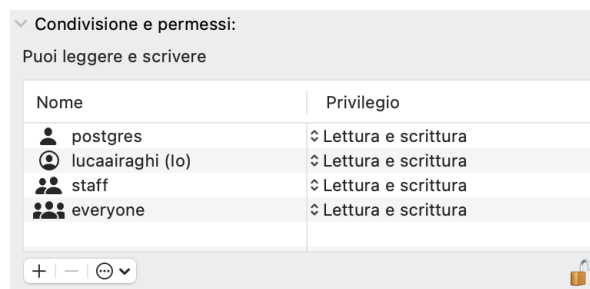
- una volta completata l'aggiunta dell'utente con i relativi permessi basterà cliccare su “OK” per confermare le modifiche apportate.
- Per quanto riguarda la modifica dei permessi su sistema operativo MacOS:
  - 1) dovremmo andare sulla seguente schermata premendo control e cliccando nello stesso momento sul file



2) Procedere cliccando ottieni informazioni



- 3) Successivamente andare nella parte in basso a destra della schermata e cliccare sul lucchetto. Vi verrà chiesta la password del vostro mac. In questo modo tramite il + aggiungerete una nuova “entità” che potrà leggere/scrivere sul file. Nel nostro caso possiamo indifferentemente selezionare everyone oppure specificatamente postgres.



- 4) Aggiunti le “entità”, basterà ricliccare sul lucchetto per salvare le modifiche effettuate

## COME APRIRE L'APPLICAZIONE:

Una volta aperta la Cartella “Emotionalsongs”, aprite la cartella “target”, dove troverete l'eseguibile jar relativo alla parte Server chiamato ServerEs.jar.

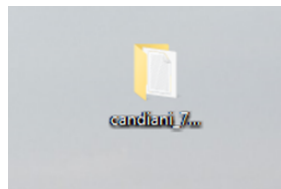
Dovranno essere inseriti i parametri di collegamento al Database precedentemente creato su pgAdmin, dove:

- hostname: rappresenta il nome o l'indirizzo del server su cui sarà ospitato il database

- username: rappresenta il nome utente per poter accedere al database
- password: rappresenta la password di accesso del database

Qualora sbagliaste ad inserire uno dei parametri sopraelencati vi comparirà una schermata di errore per avvisarvi che i dati inseriti non sono corretti e che il programma non è riuscito ad effettuare il collegamento.

Una volta aperta la Cartella “**candiani\_750632**”, aprite la cartella “target”, dove troverete l'eseguibile jar relativo ad entrambe le applicazioni.



Eseguibile relativo all'applicazione client: ClientES.jar

Eseguibile relativo all'applicazione server: ServerES.jar

		Cartella di file	
ClientES.jar	04/09/2023 14:31	Executable Jar File	14.242 KB
maven-javadoc-plugin-stale-data.txt	04/09/2023 14:47	Documento di testo	5 KB
ServerES.jar	04/09/2023 14:47	Executable Jar File	14.242 KB

## STRUTTURA APPLICAZIONE EMOTIONALSONGS

### - package ClassiSerializzabili

- Brano
- DatiUtente
- Giudizio
- ListOfBrani
- Playlist

### - package clientES

- clientEs
- IpServer

- package emotionalnew
  - Accesso.java
  - AreaRiservata.java
  - CercaBranoMusicale.java
  - ElencoPlaylist.java
  - GiudicaBrano.java
  - Lista.java
  - MainDB
  - RegistraUtente.java
  - RisultatiRicerca.java
  - SchermataAvvio.java
  - VisualizzaBranoMusicale.java
  - VisualizzaPlaylist.java
  
- package serverEs
  - CredenzialiDB
  - DataBase
  - serverES
  - ServerSlave

## PACKAGE ClientES:

### ClientES.java

- Classe che contiene uno dei due main del progetto, e che serve per inizializzare la comunicazione con il server tramite Socket.
- La classe estende la superclasse Thread in modo che sia possibile creare più istanze indipendenti di essa.
- La classe comprende diversi metodi che si vanno ad interfacciare con il Server Slave affinché sia possibile operare sul database.

### Attributi della classe:

- Socket socketClient
- ObjectInputStream in
- ObjectOutputStream out
- RegistraUtente registraUtente
- Accesso accesso
- ListOfBrani miaLista = new ListOfBrani()
- JList<String> myList = new JList<>()
- private ArrayList<Playlist> listaRicerca = new ArrayList<>()
- private Playlist p
- ListOfBrani list = new ListOfBrani()
- ElencoPlaylist elencoPlaylist
- private int[] emozioni = new int[]{0,0,0,0,0,0,0,0,0}
- private int[] valutazione = new int[]{0,0,0,0,0,0,0,0,0}
- private double[] media = new double[]{0,0,0,0,0,0,0,0,0}
- private ArrayList<Giudizio> giudizi = new ArrayList<>()

- private static String ipServerConnect
- private static boolean cambia

### Metodi della classe:

- Metodi con cui il client invia i dati alla classe "ServerSlave". Tramite una stringa avvisa la classe "ServerSlave" dell'operazione da eseguire. Successivamente il client attende un messaggio dal "ServerSlave". Il client poi nella specifica classe in cui il metodo viene richiamato gestirà l'oggetto ricevuto mostrando all'utente un messaggio di riuscita dell'operazione, oppure direttamente l'oggetto (es. una lista di canzoni)
  - public clientES() → Costruttore della classe, istanzia il socket e con esso va ad impostare i canali di comunicazione
  - public boolean invioDatiUtenti(DatiUtenti datiUtenti) → public boolean accessoUtente (DatiUtenti datiUtenti)
  - public JList<String> leggiCanzone() throws ClassNotFoundException
  - public ArrayList<Playlist> ricercaPlaylist(String username)
  - public boolean esistePlyst(String nomePI, String username) throws IOException
  - public void cancellaPlay(String username, String name)
  - public void cancellaCanzone(String idCanzone, Playlist p, String username)
  - public ArrayList<Brano> cercaBranoMusicale(int index, String filtro1, String filtro2)
  - public boolean RegistraPlaylist(String s, String nomePlaylist, String username)
  - public void visualizzaEmozioneBrano(Brano Canzone)
  - public void creaCanzone(String IdSelezionato, String nomePI, String username)
  - public ArrayList<Brano> VisualizzaCanzoniPLaylist(String username, String nomePI)
  - public boolean controllaGiudizio(String username, String IdSelezionato)
  - public String selezionaEmozione(String username, String IdSelezionato)
  - public void sovrascriviCommento(String id, String idEmozione, String username, int[] valutazione, JTextArea[] areaCommento)
  - public void inserisciCommento(String id, String username, int[] valutazione, JTextArea[] areaCommento)
  - public int[] getEmozioni()
  - public int[] getValutazione()
  - public double[] getMedia()
  - public ArrayList<Giudizio> getGiudizio()
  - public void close()
  - public static void main(String[] args) → esegue un'operazione diversa a seconda del valore dell'attributo booleano cambia

### IpServer.java

classe che viene richiamata sia all'avvio del programma dalla classe ClientES per poter impostare l'indirizzo ip/hostname a cui connettersi, sia che dall'actionListenerImpostazioni della classe SchermataAvvio per poter modificare le impostazioni una volta avviata l'applicazione. La classe presenta due costruttori uno che si avvia quando è necessario inserire l'ip a cui collegarsi e uno che si richiama quando non è possibile collegarsi ad un server facendo comparire in sovrapposizione l'errore relativo.

### La classe comprende due metodi:

- **actionListnerAccedi(ActionEvent e)** → premendo il JButton "Accedi" viene chiamato il costruttore della classe ClientES passando il parametro che servirà a identificare l'ip del server (qualora il JTextField da cui recuperare l'ip del server sia vuoto verrà passato di default la voce "localhost")

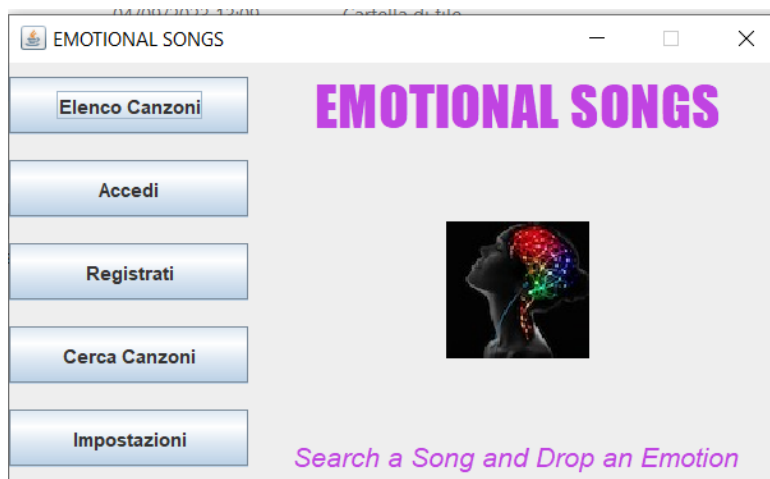


- **actionListenerAnnulla(ActionEvent e)** → premendo il *JButton* “Annulla” a seconda della variabile intera “impostazioni” passata al costruttore della classe verrà scelto se svuotare semplicemente il *TextField* qualora fossimo alla schermata di inizializzazione del programma oppure di tornare alla SchermataAvvio qualora l'applicazione fosse già in esecuzione.

## PACKAGE emotionalnew:

### SchermataAvvio.java

Definisce la prima finestra mostrata dall'applicazione, contenente una *JLabel* mostrante il nome dell'applicazione, una *JLabel* contenente l'immagine copertina dell'applicazione e da quattro *JButton* con lo scopo di far scegliere all'utente se accedere, registrarsi o accedere senza registrarsi (Cerca Canzoni).



### La classe comprende cinque metodi:

- **actionListenerLista(ActionEvent e)** → premendo il *JButton* “Elenco Canzoni” viene creata una nuova finestra “Lista” dove sarà possibile visualizzare l'elenco delle canzoni disponibili nell'archivio
- **actionListenerAccedi(ActionEvent e)** → Premendo il *JButton* “accedi” viene creata una nuova finestra “Accesso” dove sarà possibile effettuare il login e verrà chiusa la schermata “SchermataAvvio” in esecuzione.
- **actionListenerRegistrati(ActionEvent e)** → Premendo il *JButton* “registrati” viene creata una nuova finestra “RegistraUtente” dove sarà possibile effettuare la registrazione e verrà chiusa la schermata “SchermataPrincipale” in esecuzione.
- **actionListenerConsultaRepo(ActionEvent e)** → Premendo il *JButton* “Cerca Canzoni” verrà creata una nuova finestra “CercaBranoMusicale” e verranno passate tutte le stringhe vuote come parametro in ingresso al costruttore (significa che non è stato eseguito l'accesso con un'utenza registrata). Verrà inoltre passata una variabile a true e una a false per garantire il corretto funzionamento del programma. Successivamente verrà chiusa la schermata “SchermataPrincipale” in esecuzione.

- **actionListenerImpostazioni(ActionEvent e)** → Premendo il *JButton* “Impostazioni” verrà creata una nuova finestra “IpServer” e verrà passato un parametro al costruttore della classe chiamata che gestirà il funzionamento del pulsante “indietro”.  
La schermata aperta servirà nel caso sia necessario effettuare un cambio ip del server a cui connettersi

## RegistraUtente.java

- Definisce la finestra dove l'utente può effettuare la registrazione all'applicazione. È composta da dieci *JTextField*, una *JPasswordField* dove l'utente inserirà i dati richiesti e da undici *JLabel* che descrivono quale dato inserire. Infine sono presenti due *JButton*, uno per confermare la form della registrazione e uno per tornare alla schermata precedente (nuovo oggetto “SchermataAvvio”).

### La classe comprende quattro metodi:

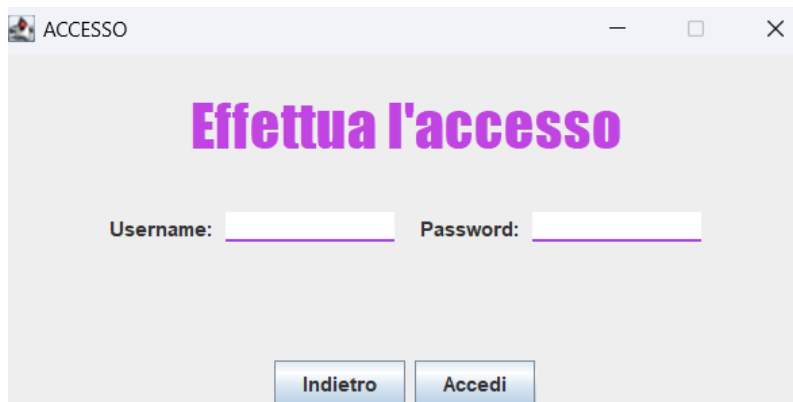
- **actionListenerIndietro(ActionEvent e)** → Al clic del bottone “indietro” verrà creata una nuova finestra “EMOTIONALSONGS” (classe SchermataAvvio) e verrà chiusa la finestra “REGISTRAZIONE” (classe registraUtente) in esecuzione. Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerConferma(ActionEvent e)** → Al click del bottone “conferma” viene effettuato un controllo su tutti i campi presenti nelle *JTextField*. Il metodo gestisce quello che è un event e sul bottone e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo. Verrà verificato se tutti i campi sono stati compilati e se rispettano i controlli sull’integrità del dato. Se la verifica è andata a buon fine (non ci sono errori) verrà richiamato il metodo “invioDatiUtenti”. Questo metodo essendo boolean ci permette tramite un if di stampare all’utente mediante “*JOptionPane.showMessageDialog*” un messaggio di conferma dell’avvenuta scrittura dei dati). Infine verrà creata una nuova finestra “SchermataAvvio” e chiusa la finestra “RegistraUtente” in esecuzione. In caso contrario

viene mostrato un messaggio di errore scrittura (tramite un `JOptionPane.showMessageDialog()`). Il metodo va a controllare gli errori ad ogni inserimento dei caratteri all'interno del `JTextField` tramite la funzione `addKeyListener()`, se non vanno bene i caratteri inseriti verrà visualizzata vicino al `JTextField` una `JLabel` contenente il problema.

- **invioDatiUtente()** → Il metodo consente di estrapolare i dati dalle label ed inserirli dentro delle stringhe. Queste stringhe successivamente verranno inserite dentro il costruttore della classe `"DatiUtente"`, che permetterà di inviare alla classe `"ServerSlave"` tutti i dati della registrazione dell'utente tramite un unico oggetto, poiché la classe è stata resa serializzabile. Questo metodo è un booleano, restituirà dunque `true` se l'operazione di invio dati al `"ServerSlave"` ed inserimento di essi nel database è andato a buon fine; `false` altrimenti

## Accesso.java

- Definisce la finestra dove l'utente può effettuare il login alla propria area riservata. È composta da un `JTextField`, un `JPasswordField` dove l'utente inserirà i dati richiesti e due `JLabel` che descrivono quale dato inserire. Infine sono presenti due `JButton`, uno per confermare la form del login e uno per tornare alla schermata precedente (nuovo oggetto `"Principale"`).

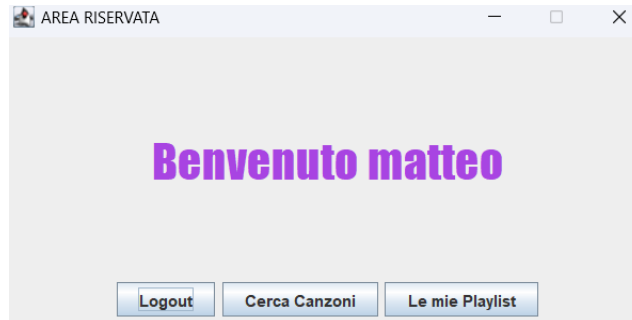


### La classe comprende tre metodi:

- **actionListenerIndietro(ActionEvent e)** → Al clic del bottone "indietro" verrà creata una nuova finestra `"SchermataAvvio"` e verrà chiusa la finestra `"Accesso"` in esecuzione. Il metodo gestisce quello che è un "event" sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerAccedi(ActionEvent e)** → Al click del bottone "accedi" viene invocato il metodo `"accessoUtente"` a cui vengono passati i campi presenti nel `JTextField` e `JPasswordField`, inseriti in una variabile di tipo `DatiUtente`. I dati saranno poi passati dal Client al `ServerSlave`, dove verrà controllata la coppia username-password all'interno del database. Il metodo restituirà una variabile booleana a seconda di come sia andato a finire il controllo. Se viene restituito `true` verrà visualizzato un messaggio di benvenuto tramite `"JOptionPane.showMessageDialog()"`, invece se verrà restituito `false` verrà visualizzato un messaggio di errore tramite `"JOptionPane.showMessageDialog()"`. Il metodo gestisce quello che è un "event" sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.

## AreaRiservata.java

- definisce la finestra dopo l'accesso dell'utente (è la sua area riservata). È composta da due *JLabel* che contengono la parola "Benvenuto" e lo username dell'utente loggato. Infine sono presenti tre *JButton*, uno per effettuare il logout, uno per cercare le canzoni e uno per visualizzare le proprie playlist.



### La classe comprende tre metodi:

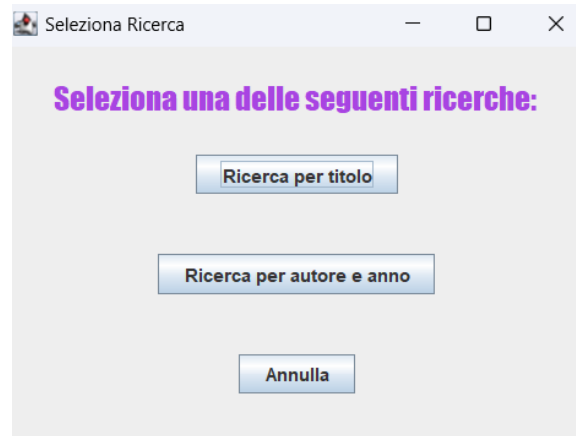
- **actionListenerLogout(ActionEvent e)** → Al clic del bottone "logout" verrà creata una nuova finestra "SchermataAvvio" e varrà chiusa la finestra "AreaRiservata" in esecuzione. Il metodo gestisce quello che è un "event" sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerCerca(ActionEvent e)** → Al clic del bottone "cerca" verrà creata una nuova finestra "CercaBranoMusicale" e varrà chiusa la finestra "AreaRiservata" in esecuzione. Il metodo gestisce quello che è un "event" sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerPlaylist(ActionEvent e)** → Al clic del bottone "playlist" verrà creata una nuova finestra "ElencoPlaylist" e varrà chiusa la finestra "AreaRiservata" in esecuzione. Il metodo gestisce quello che è un "event" sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.

## CercaBranoMusicale.java

Definisce la finestra quando l'utente deve cercare un brano musicale, sia quando lo vuole solo cercare oppure anche quando lo vuole aggiungere alla playlist. È una delle classi più importanti perché gestisce la ricerca del brano. È composta da una *JLabel* che contiene la frase "Seleziona una delle seguenti ricerche" e sono presenti tre *JButton*, uno per annullare la ricerca e tornare alla schermata precedente, uno per cercare le canzoni per titolo e uno per cercare le canzoni per autore e anno. Per la creazione questa finestra necessita di 5 campi che vengono passati alla sua chiamata e sono:

- **String username:** permette di associare qualsiasi azione ad un utente specifico.
- **String nomePlaylist:** se non è vuoto permette l'aggiunta delle canzoni alla playlist specificata.

- **String risultatiTitoli:** indica i brani che si vogliono aggiungere alla playlist e viene aggiornata sempre.
- **boolean enabled:** permette l'abilitazione o la disabilitazione del tasto indietro.



### La classe comprende tre metodi:

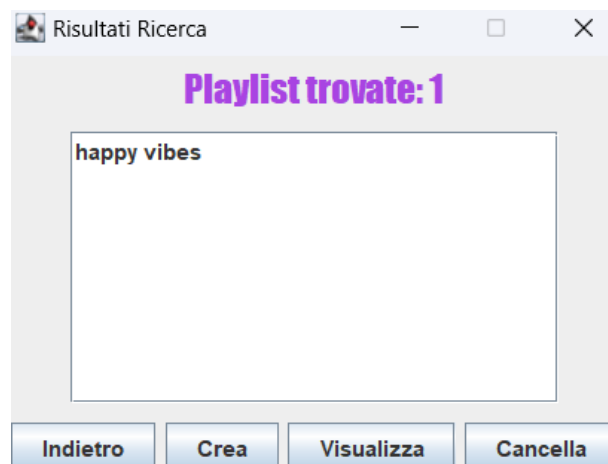
- **actionListenerIndietro(ActionEvent e)** → Al clic del bottone “indietro” verrà creata una nuova finestra e verrà chiusa la finestra “CercaBranoMusicale” in esecuzione. A seconda dei parametri passati al costruttore la finestra che andrà ad aprire sarà diversa:
  - 1) if(nomePlaylist.equals("") && username.equals("")):
    - new SchermataAvvio();
  - 2) if(!nomePlaylist.equals("") && !username.equals("")):
    - new ElencoPlaylist(username);
  - 3) if(enabled):
    - new ElencoPlaylist(username);
  - 4) if(!enabled):
    - new AreaRiservata(username);
- Permette il ritorno alla corretta finestra precedente.
- Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerTitolo(ActionEvent e)** → Al clic del bottone “titolo” verrà visualizzato un “JOptionPane” in cui dovrà essere inserito il titolo della canzone da cercare; se non viene inserito (inserimento verificato tramite if) verrà visualizzato un messaggio di errore. Se viene inserito il titolo verrà creata una nuova finestra “RisultatiRicerca” e verrà chiusa la finestra “CercaBranoMusicale” in esecuzione. Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerAutoreEanno(ActionEvent e)** → Al clic del bottone “autoreEanno” verrà visualizzato un JOptionPane in cui dovrà essere inserito l'autore della canzone da cercare, se non viene inserito (inserimento verificato tramite if) verrà visualizzato un messaggio di errore. Successivamente verrà visualizzato un altro JOptionPane in cui dovrà essere inserito l'anno della canzone da cercare; se non viene inserito (inserimento verificato tramite if) verrà visualizzato un messaggio di errore. Tramite

una variabile booleana il metodo verifica che ci siano solo numeri altrimenti dà un altro warning, dopo aver passato i “controlli” verrà creata una nuova finestra “RisultatiRicerca” e verrà chiusa la finestra “CercaBranoMusicale” in esecuzione. Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.

## ElencoPlaylist.java

- Definisce la finestra quando l’utente vuole visualizzare le proprie playlist. È composta da una *JLabel* che contiene il numero di playlist trovate, sono presenti quattro *JButton*, uno per tornare alla schermata precedente, uno per creare una playlist, uno per visualizzare la playlist e un altro per cancellare la playlist. È presente anche una *JList* che viene riempita tramite un for dagli oggetti di tipo playlist trovati dalla chiamata del metodo “ricercaPlaylist”. Quando gli oggetti trovati sono zero i *JButton* presenti sono solo due, indietro e crea. Per visualizzare le playlist nella *JList* è presente un *JScrollPane*.

Per la creazione di questa finestra è necessario un parametro che viene passato alla sua chiamata che è lo username, che permette di associare qualsiasi azione ad un utente specifico, e permette di trovare le playlist a lui collegate.



### La classe comprende sette metodi:

- **actionListenerIndietro(ActionEvent e)** → Al clic del bottone “indietro” verrà creata una nuova finestra “AreaRiservata” e verrà chiusa la finestra “ElencoPlaylist” in esecuzione. Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerCrea(ActionEvent e)** → Al clic del bottone “crea” verrà visualizzato un *JOptionPane* in cui verrà chiesto di inserire il nome della nuova playlist da creare, viene effettuato un controllo (tramite un if) sul dato immesso, se non è null e non è vuoto viene chiamato il metodo “esistePI” per verificare se il nome della playlist esiste già in modo da non creare duplicati, se non esiste il metodo restituisce false e verrà creata una nuova finestra “CreaBranoMusicale” per aggiungere i brani alla nuova playlist e verrà chiusa la finestra “ElencoPlaylist” in esecuzione. Se invece si riscontrano errori perché la playlist è vuota oppure il nome esiste già verrà visualizzato un messaggio di errore tramite *JOptionPane*.  
Il metodo gestisce quello che è un event e sul bottone e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.

- **actionListenerCancella(ActionEvent e)** → Al clic del bottone “cancella” se non si avrà schiacciato una playlist verrà visualizzato un *JOptionPane* in cui verrà chiesto di selezionare la playlist da eliminare. Se la playlist è invece stata selezionata viene cercata all’interno delle playlist e assegnata ad una variabile in modo da poterla cancellare da file tramite la funzione “cancellaPlaylist” a cui verrà passato il nome della playlist selezionata e lo username del proprietario; verrà poi visualizzato un *JOptionPane* che ci avvisa che la playlist è stata cancellata correttamente e verrà creata una nuova finestra “AreaRiservata” e chiusa la finestra “ElencoPlaylist” in esecuzione. Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerVisualizza(ActionEvent e)** → Al clic del bottone “visualizza” se non si avrà schiacciato una playlist verrà visualizzato un *JOptionPane* in cui verrà chiesto di selezionare la playlist da visualizzare. Se la playlist è invece stata selezionata viene cercata all’interno delle playlist trovate dal metodo “ricercaPlaylist” e assegnata ad una variabile in modo da poterla visualizzare tramite la classe “VisualizzaPlaylist” a cui verrà passata appunto la playlist selezionata e lo username dell’utente, verrà creata quindi una nuova finestra “VisualizzaPlaylist” e chiusa la finestra “ElencoPlaylist” in esecuzione.
- Il metodo gestisce quello che è un “event” sul bottone e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **ricercaPlaylist()** → Il metodo quando viene invocato apre il file dove sono registrate le playlist in lettura; se non esiste il file viene creato in modo da evitare problemi. Il file viene letto riga per riga finché non finisce e quando in una riga viene trovato lo username di chi invoca il metodo viene creato un oggetto playlist provvisorio a cui vengono assegnati i dettagli della playlist trovati, questi oggetti playlist vengono aggiunti ad un *ArrayList* di tipo playlist. Quando lo username non corrisponde salta la playlist. L’*ArrayList* creato sarà poi usato nel costruttore per popolare la *JList*.
- **boolean esistePI(String nome)** → Il metodo quando viene invocato riceve come parametro in ingresso il nome della playlist da creare, apre il file dove sono registrate le playlist in lettura, se non esiste il file viene creato in modo da evitare problemi. Il file viene letto riga per riga finché non finisce e quando in una riga viene trovato lo username di chi invoca il metodo viene anche confrontato il nome della playlist se il nome ricevuto come parametro di ingresso del metodo è uguale restituisce il valore true altrimenti false.
- **cancellaPlaylist(String username, String name)** → Il metodo quando viene invocato riceve come parametri in ingresso il nome della playlist da cancellare e lo username dell’utente, apre il file dove sono registrate le playlist in lettura, se non esiste il file viene creato in modo da evitare problemi. Il file viene letto riga per riga, viene salvato il contenuto e riscritto senza la playlist da cancellare.

## GiudicaBrano.java

- Definisce la finestra quando l’utente vuole giudicare un brano. È composta da più *JLabel* che conterranno le varie descrizioni delle emozioni da andare a valutare con i

relativi pulsanti per esprimere il giudizio e le aree di testo per lasciare i commenti. Sono presenti diversi *JButton*, 5 per ogni emozione (5x9 emozioni) per poter esprimere una valutazione da 1 a 5 (questi bottoni si avvalgono del riempimento con una *ImageIcon* che permette di mostrare una determinata icona al posto del normale pulsante. Avremo poi un bottone per confermare e uno per annullare. Per la creazione questa finestra sono necessari 4 parametri che vengono passati alla sua chiamata ovvero:

- Lo username che permette di associare qualsiasi azione ad un utente specifico, e permette di collegare la recensione del brano selezionato all'utente.
- Un oggetto *Brano* in modo da sapere con esattezza quale brano si sta recensendo con tutti i relativi metodi della classe *Brano.java*
- Un oggetto *Playlist* da utilizzare nei metodi e negli *actionListener* della classe anche come parametro di controllo
- Un oggetto *RisultatiRicerca* che ci servirà a capire da quale schermata viene richiamato il *GiudicaBrano.java*

## La classe comprende quattro metodi:

- ***actionListenerAnnulla(ActionEvent e)*** → Al clic del bottone “Annulla” verrà utilizzata una variabile intera per capire da quale finestra si proviene e per tornare alla finestra corretta dalla quale si è chiamato il costruttore di questa classe. Quindi verrà creata una nuova finestra “*braniPlaylist*” e verrà chiusa la finestra “*GiudicaBrano*” in esecuzione oppure verrà creata una nuova finestra “*risultatiRicerca*” e verrà chiusa la finestra “*GiudicaBrano*” in esecuzione. Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- ***actionListenerConferma(ActionEvent e)*** → Al clic del pulsante “Conferma” viene controllato se è stata rilasciata una valutazione (premendo almeno un pulsante “stella”) ed in quel caso verrà richiamato il metodo “*inserisciEmozioniBrano*” che andrà a scrivere il giudizio all’interno del file “*Emozioni.dat*” presente nella directory del file. Nel caso ci sia stato uno o più errori all’interno del metodo richiamato verranno visualizzati i relativi messaggi d'errore, altrimenti verrà mostrato un messaggio indicante la corretta riuscita dell’operazione.
- ***listenerStelle(ActionEvent e)*** → Questo metodo viene richiamato da ogni bottone “Stella” che permette di assegnare un determinato valore ad ogni pulsante, ovvero il peso del giudizio rilasciato premendo il relativo *JButton* (da 1 a 5). Viene utilizzato il



nome assegnato al *JButton* per capire a quale pulsante stiamo facendo riferimento tramite determinate condizioni avvalendosi di “switch case “

- **InserisciEmozioniBrano()** → Questo metodo si occupa di scrivere le emozioni passate alla pressione del tasto “Conferma” sul file denominato “Emozioni.dat” controllando che non si generino errori ed in quel caso, passando al chiamante tale informazione in modo da intraprendere una scelta differente a seconda del risultato.
- Durante l'esecuzione del metodo, tramite il controllo dei dati relativi al nickname e all'id del brano che si vuole commentare verrà verificato se esiste già una valutazione associata a quel brano relativa a quell'utente ed in quel caso verrà richiesto se si desidera sovrascriverla mediante un messaggio di una “*JOptionPane*”.
- Andremo quindi prima a leggere il file “Emozioni.dat” per verificare la presenza di recensioni e poi a sovrascriverlo nel caso in cui ce ne fosse la necessità.

## RisultatiRicerca.java

- Definisce la finestra che mostra i risultati contenenti i brani che corrispondono ai parametri di ricerca specificati nella finestra “cercaBranoMusicale”. È composta da *JLabel* , *JPanel*, *JList* e *JButton* che comporranno l'aspetto finito della finestra di visualizzazione dei risultati.
- La *JList* avrà dimensioni differenti e si potrà scorrere di più o di meno a seconda della quantità di risultati trovati che viene anche mostrata nel *JLabel*.
- A seconda del numero di risultati trovati verranno istanziate diverse finestre, ovvero se tramite i parametri passati non vengono trovati risultati allora verrà visualizzato un messaggio che avviserà l'utente del mancato match con i parametri di ricerca e verrà riportato alla finestra chiamante, in caso contrario verrà mostrata la finestra contenente tutti i dati richiesti. Per la creazione questa finestra sono necessari 7 parametri che vengono passati al metodo costruttore alla sua chiamata ovvero:
  - Una stringa di testo chiamata “filtro1” contenente il primo filtro di ricerca che verrà utilizzato dai metodi della classe per verificare la presenza di uno o più brani corrispondente ai criteri della ricerca
  - Una stringa di testo chiamata “filtro2” contenente il secondo filtro di ricerca che potrà essere utilizzato dai metodi della classe per verificare la presenza di uno o più brani corrispondente ai criteri della ricerca
  - Una variabile di tipo intero chiamata “tipoRicerca” che conterrà un valore che va da 1 a 2 a seconda del tipo di ricerca selezionata dell'utente e che permetterà al metodo “cercaBranoMusicale” presente nella classe di effettuare una determinata operazione piuttosto che un'altra (ovvero quali e quanti filtri utilizzare)
  - Una Stringa di testo chiamata “username” che permetterà di associare qualsiasi azione ad un utente specifico, e di collegare la ricerca del brano selezionato all'utente.
  - Una Stringa di testo chiamata “nomePlaylist” che permetterà a seconda del suo contenuto di andare ad abilitare determinate funzioni all'interno della classe
  - Una stringa di testo chiamata “risTitolo” che permetterà di scrivere all'interno del file “Playlist.dat” tutti i dati delle canzoni passati dalla finestra di ricerca ed aggiunta alla playlist anche richiamando altri metodi di altre classi

- Una variabile booleana chiamata “add” che ci permetterà di abilitare un *JButton* specifico nel caso in cui andremo ad aggiungere un singolo brano alla Playlist passando per la funzione aggiungi dal menu della singola playlist



## La classe con

- **actionListener** **aggiungi**(**ActionEvent e**) → Se la variabile “aggiungi” è “nulla” verrà chiusa la finestra corrente e la classe “CercaBranoMusicale” passerà alla classe “CercaBranoMusicale” passando tutti i parametri richiesti. Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerGiudica**(**ActionEvent e**) → Una volta selezionato un elemento dei risultati di ricerca visualizzati al clic del pulsante “Giudica” che sarà abilitato solamente se il parametro “username” non sarà vuoto (in modo da verificare che l’utente avrà eseguito l’accesso e che quindi risulti abilitato a rilasciare una valutazione per un determinato brano) verrà richiamato il costruttore della classe “GiudicaBrano” passando i parametri richiesti permettendo così di giudicare il brano.
- Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest’ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListeneraddPlaylist**(**ActionEvent e**) → Questo metodo ci permetterà di andare ad aggiungere uno o più brani alla playlist che viene passata al costruttore della classe di appartenenza.
- A seconda se la variabile “nomePlaylist” passata sarà vuota o contenente il nome della playlist ci permetterà di abilitare il pulsante “Aggiungi alla playlist” che una volta premuto richiamerà questo “actionlistener” nel quale sarà possibile scegliere se aggiungere un’altra canzone alla playlist passata mantenendo gli stessi parametri di ricerca o cambiandoli scrivendo in aggiunta alla stringa “risultatiTitoli” che poi sarà passata nuovamente al costruttore della classe “CercaBranoMusicale”.
- Una volta completata l’aggiunta di tutti i brani interessati verrà richiamato il metodo “RegistraPlaylist” al quale verrà passata la stringa “risultatiTitoli” contenente tutti i brani che dovranno essere scritti sul file “Playlist.dat” associati all’utente che ha effettuato l’accesso
- **actionListenerVisualizza**(**ActionEvent e**) → Questo metodo ci permetterà di visualizzare le valutazioni associate al brano selezionato mostrando un pop-up contenente un errore nel caso in cui non venga selezionato un brano prima di cliccare il pulsante.
- Una volta selezionato il brano e premuto il pulsante “Visualizza” verranno passati al costruttore della classe “VisualizzaBranoMusicale” i parametri per visualizzare i dettagli e le valutazioni associate al brano. Il metodo gestisce quello che è un “event” sul

pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.

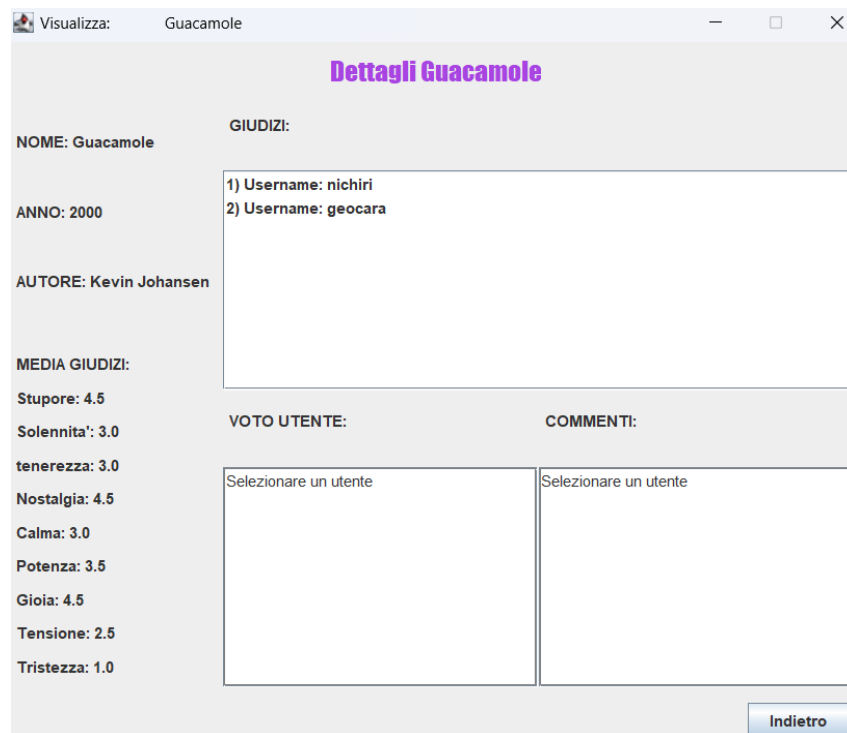
- **addListeneraddBranoSingolo(ActionEvent e)** → Questo metodo e il relativo pulsante saranno visibili e chiamabili solamente se la variabile booleana passata al metodo sarà vera, in questo caso non verrà mostrato il pulsante "Aggiungi alla playlist" ma verrà mostrato il nuovo pulsante "Aggiungi" al suo posto.
- Se è stato selezionato il brano e poi premuto il pulsante il metodo chiamerà la funzione "creaCanzone" che effettuati i controlli del caso andrà aggiungere il nuovo brano alla playlist passata al costruttore della classe dando poi un messaggio tramite un *JOptionPane* di avvenuta aggiunta. Il metodo gestisce quello che è un "event" sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **cercaBranoMusicale(int index)** → Metodo che si occupa di effettuare la ricerca dei brani a seconda dei filtri indicati e passati al costruttore. Utilizza la variabile intera che è stata dichiarata e definita come "tipoRicerca" come parametro richiesto in fase di chiamata.
- A seconda del valore di index, 1 oppure diverso (utilizzeremo il 2 per differenziare il caso) effettuerà la ricerca della canzone sul file "Canzoni.dat" tramite "FileReader e BufferedReader per verificare se qualche brano presente nel file corrisponde al/ai parametro/i di ricerca ed in caso positivo aggiunge il brano trovato all'interno di un *ArrayList* di Brani. Questa lista di array sarà poi utilizzata dalla classe per mostrare i brani ed eseguire su di essi altri metodi.
- **RegistraPlaylist(String s)** → Metodo che si occupa di scrivere e quindi registrare una nuova playlist all'interno del file "Playlist.dat" utilizzando il metodo *BufferedWriter* e che restituisce un *JOptionPane* con il risultato in caso di esito positivo mostrandoci l'elenco delle nostre playlist chiamando il costruttore della classe "ElencoPlaylist".
- **creaCanzone(String idSel)** → Metodo che si occupa di effettuare l'aggiunta di un singolo nuovo brano selezionato alla playlist già esistente che viene chiamato all'interno dell'ActionListener del pulsante "Aggiungi". In questo metodo troveremo sia un *bufferedReader* che un *BufferedWriter* che serviranno rispettivamente a leggere il file "Playlist.dat" ad individuare la playlist in cui si vuole aggiungere la nuova canzone e a scrivere la canzone formattata correttamente nel posto giusto per poi poter essere visualizzata dall'apposito metodo. Il parametro "idSel" passato al metodo conterrà l'id che identifica in modo univoco la canzone da aggiungere alla playlist. Verranno confrontati "username" e il nome della playlist per trovare il posto corretto in cui aggiungere il brano

## VisualizzaBranoMusicale.java

- Definisce la finestra che mostra i dettagli del brano che si vuole visualizzare, all'interno di questa finestra verranno presentate quelle che sono le medie delle valutazioni delle emozioni degli utenti e ci sarà la possibilità di visualizzare ogni valutazione con annessi commenti rilasciata da ogni utente. È composta da *JLabel*, *JPanel*, *JList*, *JTextArea* e *JButton* che comporranno l'aspetto finito della finestra di visualizzazione del brano.
- A seconda del numero di risultati trovati verranno istanziate diverse finestre, ovvero se tramite i parametri passati non vengono trovati risultati allora verrà visualizzato un

messaggio che avviserà l'utente della mancata presenza di valutazioni, in caso contrario verrà mostrata la finestra contenente tutti i dati richiesti. Se vengono trovate valutazioni per la canzone selezionata tramite il metodo "visualizzaEmozioneBrano", viene riempito l'oggetto *JList* con l'username di chi ha effettuato le valutazioni, queste valutazioni saranno visibili con un click sullo username (tramite il metodo "risultatiSelectionListener") e verranno visualizzate nei due *JTextArea* creati.

- Per la creazione questa finestra sono necessari 3 parametri che vengono passati al metodo costruttore alla sua chiamata ovvero:
  - Una Stringa di testo chiamata "username" che permetterà di associare qualsiasi azione ad un utente specifico, e di collegare la ricerca del brano selezionato all'utente.
  - Un oggetto di tipo Brano chiamato "b" che permetterà di visualizzare correttamente il brano selezionato.
  - Un oggetto di tipo Object chiamata "r" che ci permetterà di tornare alla schermata precedente in modo da non perdere la vecchia ricerca.



### La classe comprende tre metodi:

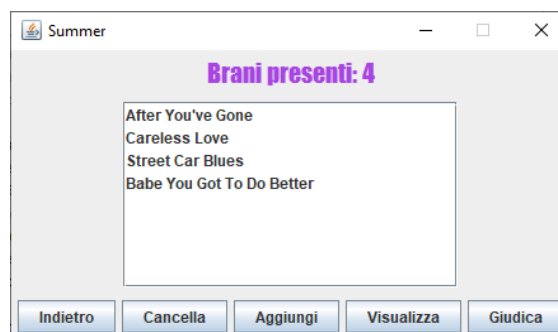
- **actionListenerIndietro(ActionEvent e)** → Al clic del pulsante "indietro" verrà chiusa la finestra corrente e verrà aperta una nuova finestra che differenzierà a seconde del parametro "Object r" ricevuto dal costruttore, se il parametro è un "instanceof" della classe "RisultatiRicerca" allora verrà aperta la finestra contenente i risultati trovati precedentemente che saranno contenuti nell'oggetto di nome "risultatiRicerca", se invece il parametro è un "instanceof" della classe "VisualizzaPlaylist" allora verrà aperta la finestra contenente i risultati trovati precedentemente che saranno contenuti nell'oggetto di nome "braniPlaylist". La finestra attuale verrà poi chiusa.
- **risultatiSelectionListener(ListSelectionEvent e)** → Questo metodo ci permetterà di visualizzare le valutazioni associate al brano selezionato per ogni singolo utente. Il metodo gestisce ogni click su un username presente nella *JList* che sarà popolata da oggetti di tipo Giudizio, quando uno di questi oggetti viene cliccato verranno inserite

all'interno dei due *JTextArea* rispettivamente le valutazioni associate ad ogni emozione e i loro commenti. Tutto ovviamente funziona se sono presenti giudizi per quel brano.

- **visualizzaEmozioneBrano()** → Questo metodo viene chiamato dal costruttore per verificare la presenza di giudizi associati alla canzone visualizzata, crea un oggetto client per istanziare una connessione con il server.
- per ottenere i dati necessari invoca i metodi del client ed assegna alle variabili i risultati ottenuti, per concludere chiude la connessione con il server.
- assegna i giudizi trovati in un *ArrayList* di tipo giudizio che poi servirà per riempire la *JList* con i nickname da far visualizzare all'utente.
- Il metodo riceve anche la media di tutte le valutazioni di ogni utente per il brano visualizzato.

## VisualizzaPlaylist.java

- Definisce la finestra che mostra i brani che corrispondono alla playlist selezionata nella finestra "ElencoPlaylist". È composta da *JLabel*, *JPanel*, *JList* e *JButton* che comporranno l'aspetto finito della finestra di visualizzazione dei brani. La *JList* avrà dimensioni differenti e si potrà scorrere di più o di meno a seconda della quantità di brani trovati. Il numero di brani presenti nella playlist condiziona il tipo di finestra visualizzata, ovvero se tramite i parametri passati non vengono riscontrati risultati allora verrà visualizzato un messaggio che avviserà l'utente della mancata presenza di brani all'interno della playlist e verrà visualizzata la finestra "Nome playlist" con all'interno il bottone indietro, il bottone aggiungi e la *JList* in cui sarà presente la scritta "non sono presenti brani".
- Per la creazione questa finestra sono necessari 2 parametri che vengono passati al metodo costruttore alla sua chiamata ovvero:
  - Una Stringa di testo chiamata "username" che permetterà di associare qualsiasi azione ad un utente specifico, e di collegare la ricerca del brano selezionato all'utente.
  - Un oggetto di tipo *Playlist* chiamato "p" che permetterà di visualizzare correttamente i brani presenti in essa.



### La classe comprende sette metodi:

- **actionListenerIndietro(ActionEvent e)** → Al clic del pulsante "indietro" verrà chiusa la finestra corrente e verrà aperta una nuova finestra "ElencoPlaylist" passando tutti i parametri richiesti. Il metodo gestisce quello che è un "event" sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.

- **actionListenerCancellaCanzone(ActionEvent e)** → Al clic del bottone “cancella” se nessuna canzone risulta selezionata verrà visualizzato un *JOptionPane* in cui sarà chiesto di selezionare il brano da eliminare. Se la canzone è invece stata selezionata verrà chiamato il metodo “cancellaCanzone” a cui verrà passato l'id della canzone selezionata, successivamente alla terminazione del metodo verrà poi visualizzato un *JOptionPane*, il quale informerà che la canzone è stata cancellata correttamente. Successivamente verrà creata una nuova finestra “ElencoPlaylist” e chiusa la finestra “VisualizzaPlaylist” in esecuzione.
- Il metodo gestisce quello che è un “event” sul pulsante e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerAggiungiCanzone(ActionEvent e)** → Al clic del bottone “aggiungi” verrà creata una nuova finestra “CreaBranoMusicale” per aggiungere un brano alla playlist e verrà chiusa la finestra “VisualizzaPlaylist” in esecuzione, alla classe “CreaBranoMusicale” vengono passati come parametri, per aggiungere la canzone alla playlist corretta, lo username e il nome della playlist dove inserire la canzone.
- Il metodo gestisce quello che è un event e sul bottone e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerGiudica(ActionEvent e)** → Una volta selezionato un brano, al clic del pulsante “Giudica” verrà richiamato il costruttore della classe “GiudicaBrano” passando i parametri richiesti permettendo così di giudicare il brano, tra cui lo username e l'oggetto di tipo Brano da giudicare.
- Se non viene selezionato un brano e schiacciato il tasto verrà visualizzato un *JOptionPane* contenente un messaggio di errore.
- Il metodo gestisce quello che è un event e sul bottone e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **actionListenerVisualizza(ActionEvent e)** → Questo metodo ci permetterà di visualizzare le valutazioni associate al brano selezionato mostrando un pop-up contenente un errore nel caso in cui non venga selezionato un brano prima di cliccare il pulsante.
- Una volta selezionato il brano e premuto il pulsante “Visualizza” verranno passati al costruttore della classe “VisualizzaBranoMusicale” i parametri per visualizzare i dettagli e le valutazioni associate al brano, tra cui lo username, il brano da visualizzare e un oggetto di tipo “VisualizzaPlaylist” necessario poi per tornare alla finestra attuale.
- Il metodo gestisce quello che è un event e sul bottone e solo quando quest'ultimo viene cliccato ci permette di svolgere una determinata azione presente nel metodo.
- **ricercaBrani()** → Il metodo viene invocato per controllare se ci sono brani presenti all'interno della playlist, ciò lo fa chiamando il metodo del client “VisualizzaCanzoniPLaylist” a cui passa come parametri l'username dell'utente e il nome della playlist da visualizzare, il metodo restituirà una *ArrayList* di tipo brano che verrà utilizzato poi per popolare la *JList* e successivamente viene invocato il metodo “Close” della classe client.
- **cancellaCanzone(String idSel)** → Il metodo quando viene invocato riceve come parametri in ingresso l'id della canzone da cancellare dalla playlist, crea un nuovo

oggetto client e invoca il metodo "cancellaCanzone" a cui passa l'id della canzone da eliminare dalla playlist, l'oggetto playlist in uso e l'username del proprietario della playlist. Una volta eliminata la canzone dalla playlist verrà invocato il metodo "Close" del client.

## PACKAGE serverES:

### credenzialiDB.java

- classe che permette la creazione del frame che precede l'avvio del server, in cui viene richiesto di inserire hostname, username e password per accedere al database.

#### Metodi della classe:

- **actionListenerAnnulla** → rende il campo dei JTextField hostname, username e password vuoto, quindi annullando eventuali inserimenti effettuati
- **actionListenerConnetti** → prova a connettersi al database, usando le credenziali inserite negli appositi campi. Qualora il tentativo fallisse, verrà stampato a video tramite un JOptionPane un messaggio d'errore. Se non fosse rilevato alcun errore in fase d'accesso, verrà richiamata la classe MainDB per la creazione del db e delle sue relative tabelle

### DataBase.java

- La classe è progettata per semplificare la connessione e l'interazione con un database PostgreSQL. Fornisce funzioni per stabilire la connessione, eseguire query e ottenere i risultati

#### Variabili di Connessione:

- protocol: Contiene il protocollo utilizzato per la connessione al database (nel caso specifico, "jdbc:postgresql://").
- host: Indica l'indirizzo dell'host del database (può essere modificato tramite i costruttori).
- db\_name: Rappresenta il nome del database (nella versione fornita, "emotionalsong").
- user: Memorizza l'username utilizzato per l'autenticazione al database.
- password: Memorizza la password utilizzata per l'autenticazione al database.
- url: Contiene l'URL completo per la connessione al database (formato: protocollo + host).
- urlnew: Contiene l'URL completo per la connessione al database con il nome del database incluso (formato: protocollo + host + db\_name).

#### Metodi della classe:

- **public DataBase(String h, String u, String p):** Costruttore che riceve l'host, l'username e la password per l'accesso al database e imposta le variabili corrispondenti.
- **public DataBase(boolean variable, int var) throws SQLException:** Secondo costruttore che prova a stabilire la connessione al database utilizzando le variabili

precedentemente inizializzate. La connessione viene creata tramite `DriverManager.getConnection()`. In base ai parametri variabile e var, decide se connettersi al database specificato o al solo host. Crea anche uno statement per eseguire query.

- **Metodo Statico getInstance:**

- `public static DataBase getInstance(boolean var, int dato) throws SQLException:` Restituisce un'istanza della classe `DataBase`. Questo metodo è utilizzato per ottenere un'istanza dell'oggetto `DataBase` preconfigurata per l'uso. L'istanza viene creata utilizzando uno dei costruttori della classe.

- **Metodo getStatement:**

- `public static Statement getStatement():` Restituisce l'oggetto `Statement` utilizzato per eseguire le query sul database.

- **Metodo submitQuery:**

- `public ResultSet submitQuery(String sql) throws SQLException:` Esegue una query fornita come argomento e restituisce un oggetto `ResultSet` contenente i risultati della query. Se la query non restituisce un risultato (ad esempio, è un'istruzione di modifica), il metodo restituirà null.

## ServerSlave.java

- La classe serve per gestire il database. In particolare, qualunque operazione che il client voglia eseguire, viene svolta dal `ServerSlave` il quale ne analizza la richiesta, esegue l'operazione e ritorna un oggetto (booleano / lista di canzoni ecc...).
- Il `ServerSlave` si interfaccia al database, quindi andrà ad eseguire delle query all'interno dei suoi metodi, per poter salvare i dati nel database.
- Ad esempio per il metodo registrazione, il `serverSlave` riceverà i dati sottoforma di oggetto serializzabile, lo spacchetterà e li andrà ad usare nella query che ne consente l'inserimento all'interno del database. Successivamente ritornerà un booleano al client se l'operazione sarà andata a buon fine.

## Attributi della classe:

- `private Socket socket;`
- `private ObjectOutputStream out;`
- `private ObjectInputStream in;`
- `private int index;`
- `private ArrayList<Brano> listaRicerca = new ArrayList<>();`
- `private ListOfBrani list = new ListOfBrani();`
- `private String filtro1;`
- `private String filtro2;`
- `private String nomePI;`
- `private String username;`
- `private String name;`
- `private String id;`
- `private Playlist p;`
- `private String stringa;`
- `private String idemozione;`
- `private int[] emozioni = new int[]{0,0,0,0,0,0,0,0};`
- `private int[] valutazione = new int[]{0,0,0,0,0,0,0,0};`
- `private double[] media = new double[]{0,0,0,0,0,0,0,0};`



- private ArrayList<Giudizio> giudizi = new ArrayList<>();
- private Brano Canzone;
- private String idSelezionato;
- private JTextArea[] areaCommento = new JTextArea[9];
- public DataBase db;
- private DatiUtenti ricevimentoDati;
- public ServerSlave(Socket s, CredenzialiDB dt) throws IOException {
- public void run() {
- private synchronized boolean registra(DataBase databs) throws IOException {
- private synchronized boolean controlloUsername(DataBase dt, Query q, String username) throws IOException, EOFException {
- private synchronized boolean accessoUtente(DataBase dt, String username, String password) throws IOException, EOFException {
- private synchronized ArrayList<Playlist> ricercaPlaylist(DataBase datab, String username) throws IOException {
- private synchronized ListOfBrani cercaBranoMusicale(int index, DataBase datab) throws IOException, EOFException {
- private synchronized ListOfBrani VisualizzaCanzoniPLaylist(DataBase datab, String namePl, String username) throws IOException, EOFException{
- private synchronized ListOfBrani leggiCanzoni(DataBase datab) {
- private synchronized boolean esistePl(DataBase datab)throws IOException {
- private synchronized void cancellaPlaylist(DataBase datab, String name, String username) throws IOException {
- private synchronized Brano dettagliCanzone(String id, DataBase datab) throws IOException{
- private synchronized void cancellaCanzone(String idCanzone, DataBase datab, Playlist p, String username) throws IOException {
- private synchronized boolean RegistraPlaylist(String s, DataBase datab, String nomePlaylist, String username) throws IOException, EOFException{
- private synchronized void visualizzaEmozioneBrano(DataBase datab, Brano Canzone) throws IOException, EOFException{
- private synchronized void creaCanzone(String idSel, String nomePlaylist,String username){
- private synchronized boolean esisteGiudizio(String username, String id, DataBase datab{
- private synchronized String selezionaEmozione(DataBase datab, String username, String id) {
- private synchronized void sovrascriviCommento(DataBase datab, String id, String username) {
- private synchronized void inserisciCommento(DataBase datab, String username){

## ServerEs.java

- Classe che permette di stabilire la connessione con il client. Al suo interno troviamo un attributo statico che rappresenta la porta al quale il client dovrà connettersi per usufruire del servizio offerto dal server.

- Contiene un solo metodo la classe, il main, il quale permette di creare la finestra “CredenzialiDB”, di connettersi al client e passare la connessione al serverSlave il quale consentirà lo scambio di informazioni Client-Server.

## PACKAGE ClassiSerializzabili:

### Brano.java

- Lo scopo principale di questa classe è quello di contenere tutte le informazioni di un brano, in modo che quando vengono creati oggetti di tipo Brano essi siano già pronti per essere utilizzati dalle altre classi. Permette di rendere univoci ogni tutti i brani passati da finestra a finestra.
- La classe inoltre è resa serializzabile grazie all'implementazione di Serializable ed all'attributo serialVersionUID. Ciò vuol dire che può essere inviata come oggetto nelle scritture e letture tra Client e ServerSlave.

#### Attributi della classe:

- private static final Long serialVersionUID = 1L
- Id
- Nome
- Titolo
- Autore
- Anno

#### Metodi della classe:

- La classe comprende i metodi “get” e “set” degli attributi sopra elencati.
  - @override del metodo “toString” che permette la corretta visualizzazione del titolo di un brano quando quest'ultimo viene cercato ed appare nella *JList*.

### Playlist.java

- Lo scopo principale di questa classe è quello di contenere tutte le informazioni di una playlist, in modo che quando vengono creati oggetti di tipo playlist essi siano già pronti per essere utilizzati dalle altre classi. Permette di rendere univoche tutte le playlist passate da finestra a finestra.
- La classe inoltre è resa serializzabile grazie all'implementazione di Serializable ed all'attributo serialVersionUID. Ciò vuol dire che può essere inviata come oggetto nelle scritture e letture tra Client e ServerSlave.

#### Attributi della classe:

- private static final Long serialVersionUID = 1L
- username
- nome
- brani

#### Metodi della classe:

- La classe comprende i metodi “get” e “set” degli attributi sopra elencati.
  - @override del metodo toString che permette la corretta visualizzazione del nome di una playlist quando quest'ultima viene cercata ed appare nella *JList*.

## Giudizio.java

- Lo scopo principale di questa classe è quello di contenere tutte le informazioni di una valutazione relativa all'utente che viene espressa sul singolo brano, in modo che quando vengono creati oggetti di tipo giudizio essi siano già pronti per essere utilizzati dalle altre classi. Permette di rendere univoche tutte le istanze giudizio passate da finestra a finestra.
- La classe inoltre è resa serializzabile grazie all'implementazione di `Serializable` ed all'attributo `serialVersionUID`. Ciò vuol dire che può essere inviata come oggetto nelle scritture e letture tra Client e ServerSlave.

### Attributi della classe:

- `private static final Long serialVersionUID = 1L`
- `username`
- `valutazioneStupore`
- `valutazioneSolennita`
- `valutazioneTenerezza`
- `valutazioneNostalgia`
- `valutazioneCalma`
- `valutazionePotenza`
- `valutazioneGioia`
- `valutazioneTensione`
- `valutazioneTristezza`
- `commentoStupore`
- `commentoSolennita`
- `commentoTenerezza`
- `commentoNostalgia`
- `commentoCalma`
- `commentoPotenza`
- `commentoGioia`
- `commentoTensione`
- `commentoTristezza`

### Metodi della classe:

- La classe comprende i metodi "get" e "set" degli attributi sopra elencati.

## DatiUtenti.java

- Lo scopo di questa classe è quello di modellare i dati inseriti dall'utente nelle classi "RegistrazioneUtente" e "Accesso".
- La classe inoltre è resa serializzabile grazie all'implementazione di `Serializable` ed all'attributo `serialVersionUID`. Ciò vuol dire che può essere inviata come oggetto nelle scritture e letture tra Client e ServerSlave.
- I suoi costruttore permettono di impacchettare gli attributi delle due classi sopra citate in modo da inviare solamente un oggetto durante gli scambi di informazioni tra client e ServerSlave

### Attributi della classe:

- `private static final Long serialVersionUID = 1L`
- `nomeDato`

- cognomeDato
- comuneDato
- provinciaDato
- viaDato
- numeroCivicoDato
- capDato
- mailDato
- cFDato
- usernameDato
- passwordDato
  - gli attributi conterranno tutti i valori estratti dai JTextField dalle classi "RegistraUtente" ed "Accesso"

### Metodi della classe:

- La classe comprende i metodi "get" degli attributi sopra elencati.
- Altri metodi della classe sono i costruttori:
  - public DatiUtenti("prende come parametro tutti gli attributi") → costruttore usato nella registrazione
  - public DatiUtenti(String usernameInvio, String passwordInvio) → costruttore usato nel login

### ListOfBrani.java

- Classe che permette la creazione di una lista di brani
- La classe inoltre è resa serializzabile grazie all'implementazione di Serializable ed all'attributo serialVersionUID. Ciò vuol dire che può essere inviata come oggetto nelle scritture e letture tra Client e ServerSlave.

### Attributi della classe:

- private static final Long serialVersionUID = 1L
- lista

### Metodi della classe:

- **public ListOfBrani()** → costruttore della classe, iniiializza un ArrayList di tipo Brano
- **public void add(Brano b)** → Permette di aggiungere un Brano alla Lista
- **public void add(int i, Brano b)** → Permette di aggiungere un Brano alla Lista in posizione i
- **public void clear()** → Elimina i dati presenti in lista
- **public ArrayList<Brano> getList()** → ritorna l'ArrayList
- **public String[] toArray()** → permette di convertire la lista di brani presente all'interno della classe in un Array di stringhe contenente le informazioni dei brani che vi erano nella lista precedentemente

### Lista.java

- la classe lista permette la corretta visualizzazione di tutto l'elenco delle canzoni presente all'interno del database. Va a creare una JList contenente tutto l'elenco delle canzoni.

## Metodi della classe:

- **actionListenerIndietro()** → metodo che annulla la visibilità della classe lista, permettendo all'utente di ritornare alla "SchermataAvvio", poichè la rende visibile
- **leggiCanzone()** → metodo con cui si richiama il metodo della classe client leggiCanzone(), metodo che interrogando il database con una query rende visibile la lista delle canzoni

## MainDB.java

- Classe in cui vengono lanciate le query per istanziare database e tabelle. Il costruttore dell'app infatti permette di creare database e tabelle

## Metodi della classe:

- **getDatabase()** → ritorna un'istanza del database

## GESTIONE DEI FILE

- All'interno della soluzione adottata andiamo ad utilizzare alcuni file .png, jpg e un file .csv. I file in questione vengono utilizzati dall'applicazione per poter rendere più accattivante la grafica (file jpg) nella schermata iniziale oltre che come metodo per esprimere le proprie valutazioni ad un'emozione (file png) tramite delle icone di "stelle". Il file csv invece è utilizzato dal sistema per popolare il database al primo avvio dell'applicazione lato Server, in modo da evitare di avere un repository vuoto o da riempire a mano.

## LIMITI DELLA SOLUZIONE ADOTTATA

La soluzione adottata presenta alcuni limiti, ad esempio non è possibile inserire tramite interfaccia grafica dedicate nuove canzoni al database, ma gli unici modi per poterlo fare sono via query che deve essere effettuata direttamente sul programma che ospita il database o modificando propriamente il file csv che viene caricato al primo avvio del programma (tuttavia in questo modo affinché le modifiche siano rilevate sarà necessario eliminare la tabella contenente i brani o l'intero database).

Per poter eseguire l'applicazione si deve necessariamente possedere l'applicativo JAVA che permetterà la corretta esecuzione del codice e di tutti i parametri ad esso associati lato Client, mentre lato server si dovrà possedere inoltre l'applicativo PgAdmin e i driver per PostgreSQL.

Le finestre dell'applicazioni non sono personalizzabili totalmente dall'utente che dovrà sottostare ai parametri scelti dal programmatore.

L'utilizzo di Maven permette di incrementare semplicemente funzionalità esterne che potranno essere sviluppate in futuro e che potrebbero velocizzare/migliorare la gestione dei dati e la velocità di ricerca nel database.

## AVVERTENZE

La modifica dei file presente nelle sottocartelle della cartella padre del progetto potrebbe causare malfunzionamenti durante l'esecuzione dell'applicativo (soprattutto una modifica non appropriata dei file presenti nella cartella "src\main\resources").

Nella maggior parte dei casi, per risolvere qualsiasi problematica che ne impedisce il corretto funzionamento basterà utilizzare la cartella originale con i file e l'applicativo inalterati, mentre lato server si potrà ripristinare il database semplicemente cancellandolo e facendo ripartire l'applicazione.

## Query

Tutte le Query utilizzate hanno come primo parametro un'istanza del database sulla quale dovranno effettuare le operazioni e che consente loro di accedere ai dati

- 1) `public void queryInsertUser(DataBase db, String query) throws SQLException {`
  - Metodo che prende in input una stringa contenente la query da eseguire. La query ha la funzione di registrare un nuovo utente all'interno del database; l'utente deve essere unico, per cui nel momento della registrazione sarà possibile solamente registrarsi all'applicazione se lo username è univoco all'interno del database
- 2) `public boolean queryControllaUser(DataBase db, String user) throws SQLException {`
  - Metodo che consente di controllare se lo username di un utente è già presente all'interno del database. Il metodo legge tutte le righe della colonna username all'interno della tabella Utenti e le confronta con lo username passato come stringa. Il metodo essendo booleano ritornerà true, nel momento in cui ci sarà uno username in database uguale allo username passato come argomento, altrimenti ritornerà false
- 3) `public boolean queryVerificaPassword(DataBase db, String user, String password) throws SQLException {`
  - Metodo utilizzato per eseguire l'accesso, che riceve come input lo username e la password di un utente inserite da esso ed effettua un controllo della coppia username - password, all'interno del database. Se la coppia è presente, verrà ritornato un boolean con esito true, altrimenti false
- 4) `public ListOfBrani queryRicercaPerTitolo(DataBase db, String titolo) throws SQLException {`
  - Metodo che tramite una query a cui viene passato il parametro titolo, effettua la ricerca delle canzoni nel database nella tabella canzoni e restituisce un oggetto di tipo ListOfBrani contenente le informazioni dei brani corrispondenti al criterio di ricerca
- 5) `public ListOfBrani queryVisualizzaCanzoniIPlaylist(DataBase db, String idPlaylist) throws SQLException {`
  - Metodo che permette la ricerca di tutti i brani presenti all'interno di una playlist ritornando un oggetto di tipo ListOfBrani ed utilizzando il parametro idPlaylist che contiene la chiave primaria della tabella canzoniplaylist
- 6) `public ListOfBrani queryRicercaPerAnnoAutore(DataBase db, String autore, String anno) throws SQLException {`
  - Metodo che tramite una query, a cui vengono passati i parametri autore e anno, permette la ricerca dei brani all'interno del database ritornando un oggetto di tipo ListOfBrani
- 7) `public Brano queryRicercaInfoCanzone(DataBase db, String idCanzone) throws SQLException {`
  - Metodo che tramite una query, a cui viene passato l'id della canzone, permette di cercare le informazioni relative a un brano ritornando un oggetto Brano

- 8) `public ArrayList<Playlist> queryRicercaPlaylist(DataBase db, String username) throws SQLException {`
- Metodo con cui attraverso una query, a cui passo lo username di un utente (parametro del metodo), riesco a verificare se un utente ha associato delle playlist. In caso affermativo, verrà ritornato un array di playlist contenente le playlist riferite all'utente
- 9) `public String queryIdPlaylist(DataBase db, String nomePlaylist, String username) throws SQLException{`
- Metodo con cui attraverso una query ottengo una stringa contenente l'id della playlist il quale è associato al nome utente e al nome della playlist passati come parametri.
- 10) `public boolean queryEsistePlaylist(DataBase db, String nomePI, String username) throws SQLException {`
- Metodo che mi permette di richiamare una query che va ad interrogare il database nello specifico la tabella playlist per controllare l'esistenza di una playlist con lo stesso nome di quella passata come argomento del metodo associata ad uno specifico username. Il metodo ritorna true, se esiste già una playlist con lo stesso nome, false altrimenti. Metodo che serve per evitare di creare due playlist associate allo stesso username con lo stesso nome
- 11) `public void queryInsertSongPI(DataBase db, String idPlaylist, String idCanzone) throws SQLException {`
- Metodo richiamato che permette di inserire nella playlist corrispondente all'idPlaylist passato come parametro uno o più brani corrispondenti ad idCanzone, scelti dall'utente operando sulla tabella canzoniplaylist..
- 12) `public void queryCreaPlaylist(DataBase db, String username, String nomePlaylist) throws SQLException{`
- Metodo con cui vado a creare tramite una query, una nuova playlist per un dato utente, specificando il nome della nuova playlist passato come parametro insieme all'id dell'utente alla quale dovrà essere associata.
- 13) `public ListOfBrani listaCanzoniCompleta(DataBase db) throws SQLException {`
- Metodo con cui interfacciandosi con il database, vado a richiamare la query che mi permette di estrarre tutte le canzoni presenti nella base dati. Viene usato all'interno della classe lista, per ritornare l'elenco completo delle canzoni
- 14) `public String queryIdCanzoneEmozione(DataBase db, String id) throws SQLException {`
- Metodo con cui vado a ritornare una stringa contenente l'id del brano che poi verrà utilizzato per associare o verificare se vi sono associate emozioni ad esse
- 15) `public String queryUsernameEmozione(DataBase db, String user, String id) throws SQLException {`
- Metodo con cui vado a ritornare l'id utente dalla tabella emozioni che corrisponde ai requisiti dei parametri passati al metodo e utilizzati dalla query
- 16) `public String[] queryUsernameGiudizio(DataBase db, String id, int k) throws SQLException {`
- Metodo con cui vado a ritornare un array di stringhe contenente l'id utente degli user che hanno lasciato una recensione per uno specifico brano, dalla tabella emozioni. I parametri passati si riferiscono all'id del brano e la dimensione per inizializzare correttamente l'array.
- 17) `public int querycontaUtenti(DataBase db, String id) throws SQLException {`

- Metodo che restituisce un intero e che va a contare quanti utenti hanno lasciato una valutazione relativo ad un determinato brano identificato tramite l'id passato come parametro
- 18) `public String queryValutazioneEmozione(DataBase db, String emozione, String idemozione) throws SQLException {`
- Metodo che restituisce la valutazione relativa ad una specifica emozione qualora sia stata rilasciata, individuata a partire dai parametri passati. Emozione si riferisce alla categoria specifica e idemozione serve ad identificare l'emozione univocamente all'interno della tabella emozione
- 19) `public String queryCommentoEmozione(DataBase db, String emozione, String idemozione) throws SQLException {`
- Metodo che restituisce il commento associato ad una specifica emozione qualora sia stato rilasciato, individuato a partire dai parametri passati. Emozione si riferisce alla categoria specifica e idemozione serve ad identificare l'emozione univocamente all'interno della tabella emozione
- 20) `public void queryInserisciCommento(DataBase db, String query) throws SQLException {`
- Metodo che tramite una query passata come parametro del metodo, va ad inserire le valutazioni e i commenti per una specifica canzone associati ad un utente univoco, all'interno del database
- 21) `public void queryCancellaEmozione(DataBase db, String idemozione) throws SQLException {`
- Metodo con il quale effettuo la cancellazione dell'emozione corrispondente al parametro passato alla funzione per poter andare poi a sovrascrivere tale valutazione tramite apposita funzione in quanto per ogni utente ci può essere una sola valutazione relativa ad un brano
- 22) `public void queryCancellaPlaylist(DataBase db, String idPlaylist) throws SQLException {`
- Metodo con cui, data un stringa contenente l'id di una playlist, mi interfaccio al database, per rimuoverla
- 23) `public void queryCancellaCanzonePlaylist(DataBase db, String idPlaylist, String idCanzone) throws SQLException {`
- Metodo tramite il quale, attraverso una query che prende in input una stringa contenente l'id di una playlist e una stringa contenente l'id di una canzone, mi permette di rimuovere dalla playlist specificata la canzone con l'id relativo passato come parametro
- 24) `public String querySelezionaEmozione(DataBase db, String username, String id) throws SQLException {`
- Query che ritorna l'id dell'emozione relativa ad uno specifico utente e ad uno specifico brano interrogando la tabella emozioni utilizzando come parametri di ricerca lo username e l'id della canzone
- 25) `public void queryPopolaDb(DataBase db) throws SQLException {`
- Metodo con cui vado ad effettuare tramite una query il popolamento del database al primo avvio del programma utilizzando un file .csv come repository qualora non fosse già stato popolato in precedenza