

# POLITECNICO DI TORINO

Dipartimento di Ingegneria Gestionale e della Produzione



Corso di Laurea in Ingegneria Gestionale  
Ingegneria dell'Informazione, Classe di Laurea L-8

## **Sviluppo di un software per la generazione automatica di una playlist musicale**

**Relatore**

Prof. Fulvio Corno

**Candidato**

Walter De Carne

# Indice

<b>1. Proposta di progetto</b>	<b>2</b>
<b>2. Descrizione del problema affrontato</b>	<b>4</b>
Contesto operativo	4
Descrizione del sotto-problema	4
<b>3. Descrizione del data-set utilizzato per l'analisi</b>	<b>5</b>
<b>4. Descrizione delle strutture dati e degli algoritmi</b>	<b>7</b>
Strutture dati	7
<i>Package provafinale</i>	7
<i>Package provafinale.database</i>	8
<i>Package provafinale.model</i>	8
Algoritmi	9
1° tab: Ricerca Brani	9
2° tab: Genera Playlist	11
<b>5. Diagramma delle classi principali</b>	<b>15</b>
<b>6. Screenshot dell'applicazione e risultati ottenuti</b>	<b>16</b>
<b>7. Valutazioni e conclusioni</b>	<b>22</b>

# **1. Proposta di progetto**

## **Studente proponente**

s227538 De Carne Walter

## **Titolo della proposta**

Sviluppo di un software per la generazione automatica di una playlist musicale

## **Descrizione del problema proposto**

Il problema nasce dall'assenza di una funzione automatica per la creazione di una playlist personalizzata all'interno dell'applicazione Spotify. L'applicazione proposta consentirebbe l'automazione della generazione di playlist ottimizzata sulla base delle preferenze dell'utente.

## **Descrizione della rilevanza gestionale del problema**

Il software risulterebbe di interesse per l'azienda stessa, in quanto andrebbe ad aggiungere un nuovo servizio alla propria offerta, oppure ad un'azienda terza, che potrebbe fornire lo stesso servizio avendo accesso all'account Spotify dell'utente.

Il software sarebbe comunque flessibile all'utilizzo su un qualsiasi altro servizio di musica in streaming, ampliando il target verso più aziende, previo adattamento del nuovo data-set al format e agli indici utilizzati.

## **Descrizione dei data-set per la valutazione**

L'applicazione si basa sul data-set contenente la Top 50 annuale di ogni anno dal 2010 al 2019, stilata da Billboard che classifica le canzoni più ascoltate. Ogni brano è rappresentato da un record che contiene, oltre a un numero progressivo da 1 a 603, il titolo, l'artista, il genere e l'anno in cui il brano si è classificato nella Top 50. Inoltre, vi sono una serie di parametri:

- bpm, beats per minute;
- nrgy (energy), indica quanto una canzone è energica;
- dnce (danceability), indica quanto è danzabile;
- dB, indica i decibel;
- live, più è alto il valore maggiore è la probabilità che sia live;
- val (valence), indica quanto mette di buon umore;
- dur (duration), indica la durata;
- acous (acousticness), indica quanto è acustica;
- spch (speechiness), più è basso più è strumentale;
- pop (popularity), indica la popolarità.

## **Descrizione preliminare degli algoritmi coinvolti**

La prima parte sarà costituita da algoritmi di ricerca per la visualizzazione di record e l'elaborazione dei dati estratti.

Per la seconda parte, si utilizzerà un algoritmo ricorsivo che creerà la playlist ottimale. Affinché la canzone sia aggiunta alla playlist, sarà vincolante che rispetti le preferenze dell'utente, in termini di popularity, energy e danceability. L'utente sceglierà la durata della playlist, in questo modo l'algoritmo genererà la migliore combinazione di canzoni per quella durata complessiva. Si tratta quindi di un problema dello zaino (knapsack problem).

## **Descrizione preliminare delle funzionalità previste per l'applicazione software**

Quest'applicativo, nel primo tab, permetterà la ricerca di brani per artista (ricercabile tramite Text Field, mostrerà tutte le canzoni di quell'artista con le varie informazioni; se non sarà presente restituirà un messaggio di errore) e per genere (selezionandolo dal Choice Box), mostrando delle statistiche a riguardo (durata media, canzone più popolare).

Nel secondo tab, sulla base delle preferenze (ad esempio energy, popolarità, ecc.) selezionabili dall'utente tramite slider, sarà possibile creare una playlist di canzoni. L'utente dovrà inoltre scegliere la durata massima complessiva della playlist.

Saranno mostrati grafici che analizzano le caratteristiche della playlist generata.

## **2. Descrizione del problema affrontato**

### **Contesto operativo**

Spotify è un servizio di streaming musicale che ormai accompagna la vita di milioni di persone. Questo tipo di servizio permette a chiunque in possesso di un dispositivo elettronico (cellulare, tablet, computer, ecc.) di accedere ad un catalogo vastissimo di brani di tutto il mondo. Nel caso in questione, l'azienda afferma di possedere un catalogo di oltre 50 milioni di brani.

Servizi di questo genere hanno rivoluzionato l'industria della musica, favorendo l'utente in termini economici (si parla di pochi euro al mese per abbonato) e di comodità.

Spotify è solo uno dei tanti servizi di streaming musicale. Essi competono tra di loro per ottenere il maggior numero di abbonati, e sono pertanto interessanti ad offrire costantemente un miglioramento del servizio e delle funzionalità, affinché l'utente non decida di estinguere l'abbonamento.

### **Descrizione del sotto-problema**

Come detto, è necessario per questo tipo di aziende fornire sempre più funzionalità aggiuntive oltre al semplice streaming audio. Una delle richieste maggiori degli utenti riguarda la personalizzazione.

Ad oggi, su Spotify l'utente può creare delle playlist personali, ma l'aggiunta dei brani all'interno è delegata allo stesso, manualmente. È assente quindi una funzione che permetta la generazione automatica della playlist sulla base delle preferenze dell'utente, sia in termini di gusti musicali che in termini di durata della playlist.

Ad esempio, un utente che vuole ascoltare della musica durante un allenamento, che in genere ha una durata fissa, potrebbe voler creare una playlist rapidamente, affidando ad un software la scelta delle canzoni sulla base delle preferenze che inserisce.

Da qui, nasce l'idea per questo progetto di tesi di Laurea triennale.

### 3. Descrizione del data-set utilizzato per l'analisi

Il data-set utilizzato per lo sviluppo di questo software è basato sulla classifica annuale delle 50 canzoni più ascoltate (Top 50) in tutto il mondo nel decennio scorso (2010-2019).

Il data-set è composto da una sola tabella, chiamata *top10s*. Ogni record rappresenta un brano, caratterizzato da un numero identificativo progressivo, un titolo, un artista, il genere musicale principale, l'anno nel quale ha ottenuto una posizione in classifica, la sua durata, espressa in secondi, e nove diversi parametri che caratterizzano la canzone, che analizzeremo nel dettaglio.

In realtà, il data-set contiene più delle 500 canzoni previste, arrivando ad un totale di 603 canzoni. Tra queste, sono presenti 19 duplicati di canzoni che hanno conquistato una posizione in classifica per più di un anno. È presente quindi un totale di 584 canzoni distinte.

I parametri caratteristici di ogni canzone sono i seguenti:

- **BPM**, Beats Per Minute, nel data-set sotto la colonna *bpm*, indica il tempo della canzone;
- **Energy**, nel data-set sotto la colonna *nrngy*, indica l'energia di una canzone: più è alto il valore, più energetico è il brano;
- **Danceability**, nel data-set sotto la colonna *dnce*, indica la danzabilità di una canzone: più è alto il valore, più è facile ballare con la stessa in sottofondo;
- **Loudness** (dB), nel data-set sotto la colonna *dB*, indica la rumorosità di un brano, espressa in decibel;
- **Liveness**, nel data-set sotto la colonna *live*, indica la probabilità che una canzone sia una registrazione dal vivo: più è alto il valore, maggiore è la probabilità;
- **Valence**, nel data-set sotto la colonna *val*, indica la "positività" della canzone: più è alto il valore, più è probabile che la canzone metta di buon umore;
- **Acousticness**, nel data-set sotto la colonna *acous*: più è alto il valore, più è probabile che si tratti di una canzone acustica;
- **Speechiness**, nel data-set sotto la colonna *spch*: più è alto il valore, più parole la canzone contiene;

- **Popularity**, nel data-set sotto la colonna *pop*, indica la popolarità: più è alto il valore, più quella canzone è famosa.

Di questi parametri, nello sviluppo dell'applicazione sono stati presi in considerazione soltanto Energy, Danceability e Popularity.

Di seguito, il nome delle caratteristiche principali delle canzoni all'interno del data-set e uno screenshot della struttura di quest'ultimo.

- **Numero identificativo**, nel data-set sotto la colonna *id*;
- **Titolo**, nel data-set sotto la colonna *title*;
- **Artista**, nel data-set sotto la colonna *artist*;
- **Genere**, nel data-set sotto la colonna *top\_genre*;
- **Anno**, nel data-set sotto la colonna *year*;
- **Durata**, nel data-set sotto la colonna *dur*.

A lato è mostrato l'elenco completo delle colonne della tabella, e il tipo di dato che rappresentano. Si nota che quasi tutte le colonne sono rappresentate da numeri interi, ad eccezione di titolo, artista e genere che sono rappresentati da stringhe.

Il data-set utilizzato è stato ottenuto dal sito di ricerca di data-set Kaggle. Tutti i dati sono stati elaborati ed estratti dal sito [organizeyourmusic.playlistmachinery.com](http://organizeyourmusic.playlistmachinery.com)

Copia del data-set è disponibile a questo indirizzo:

<https://www.kaggle.com/leonardopena/top-spotify-songs-from-20102019-by-year>

Una copia è disponibile anche all'interno di questo progetto, presente su GitHub:

<https://github.com/TdP-prove-finali/DeCarneWalter/blob/master/db/top10s.sql>

#	column_name	data_type
1	id	int(11)
2	title	text
3	artist	text
4	top_genre	text
5	year	int(11)
6	bpm	int(11)
7	nrgy	int(11)
8	dnce	int(11)
9	dB	int(11)
10	live	int(11)
11	val	int(11)
12	dur	int(11)
13	acous	int(11)
14	spch	int(11)
15	pop	int(11)

## 4. Descrizione delle strutture dati e degli algoritmi

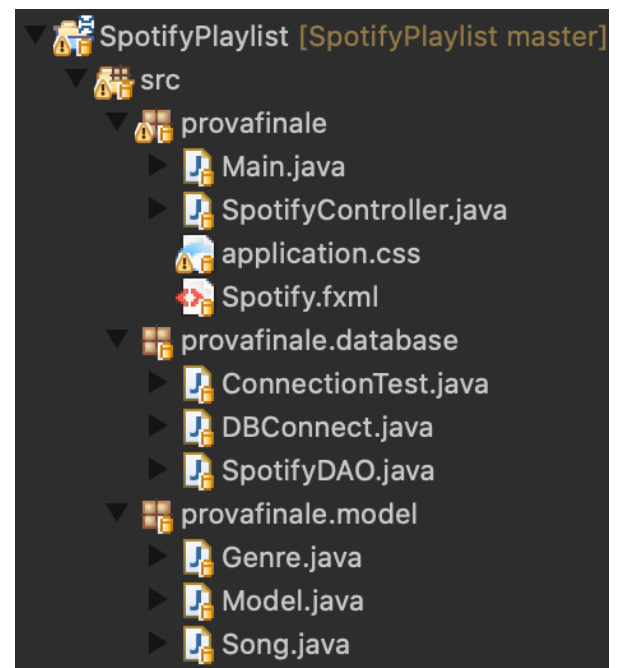
### Strutture dati

L'intero software è stato realizzato in linguaggio *Java*.

Le interfacce grafiche sono state realizzate in *JavaFX*, utilizzando l'applicativo *SceneBuilder*.

Parte dell'interfaccia grafica è stata modificata e formattata tramite *Cascading Style Sheet (CSS)*.

Per lo sviluppo dell'applicazione si è seguito il *pattern MVC (Model View Controller)* e il *pattern DAO (Data Access Object)*. Grazie a questi pattern, è possibile mantenere distinte le tre parti fondamentali di un programma, ovvero la logica, l'accesso al database e l'interfaccia utente, che vengono sviluppate in tre packages diversi, come è possibile notare nell'immagine a lato. Di seguito, sono analizzati nel dettaglio.



### Package *provafinale*

Si tratta del package principale, che contiene la classe *Main* (*Main.java*), ovvero la classe principale, che esegue l'interfaccia grafica e la sua formattazione. Essa infatti esegue il file FXML (*Spotify.fxml*) e il file CSS (*application.css*), contenuti entrambi nello stesso package. La classe *Controller* (*SpotifyController.java*) collega l'interfaccia grafica con la logica del programma ed esegue le azioni che servono a mostrare i dati a video.



## **Package *provafinale.database***

Questo package consente l'accesso alla base dati, seguendo il pattern DAO. È composto da 3 classi:

- *DBConnect.java*, all'interno della quale sono inseriti i parametri per accedere al database, con il quale stabilisce la connessione;
- *ConnectionTest.java*, che consente di verificare se la connessione al database è effettivamente avvenuta o meno, restituendo il risultato in output;
- *SpotifyDAO.java*, che effettua l'estrazione dei dati dal database. All'interno di essa sono infatti contenuti tutti i metodi che contengono le query SQL con le quali la base dati è interrogata.

## **Package *provafinale.model***

È il package che contiene la parte logica dell'applicazione. In particolare, nella classe Model (*Model.java*) sono contenuti tutti i metodi che hanno rilevanza algoritmica.

La classe Song (*Song.java*) è la classe che definisce le istanze di una canzone, serve appunto a salvare all'interno del programma i dati delle canzoni estratte dal data-set.

Allo stesso modo, la classe Genre (*Genre.java*) consente di salvare i generi.

## Algoritmi

Il software sviluppato è composto da due tab: Ricerca Brani e Genera Playlist.

Di seguito, sono analizzati nel dettaglio gli algoritmi che permettono a entrambi i tab il corretto funzionamento.

### 1° tab: Ricerca Brani

Questa prima sezione è composta principalmente da algoritmi di ricerca, elaborazione e visualizzazione dei record estratti dal database, in base alle esigenze dell'utente.

In particolare, è possibile ricercare un artista, un genere o un anno. Il Controller richiede alla classe Model le canzoni sulla base dei parametri inseriti dall'utente e quest'ultima richiede i dati alla classe DAO, che estrae i dati tramite query SQL e li restituisce sotto forma di lista prima al Model, e poi al Controller, il quale li mostra sull'interfaccia grafica.

```
else if(!artista.equals("") && genere==null) {
    if(anno!=0) {
        canzoni = model.getAllYearArtistSongs(artista, anno);
        yearsBarChart.getData().clear();
        genresPieChart.getData().clear();
        txtYearPieChart.setText("");
        yearsBarChart.setVisible(false);
    }
    else {
        canzoni = model.getAllArtistSongs(artista);
        disegnaBarChartArtista(canzoni, artista);
        genresPieChart.getData().clear();
        txtYearPieChart.setText("");
    }
}
```

*Estratto del metodo doCerca() all'interno della classe SpotifyController*

Si nota dagli estratti di codice qui riportati la separazione di logica, database e interfaccia utente come da pattern MVC e DAO.

```

public List<String> getAllGenres(){
    return dao.getAllGenres();
}

public List<Integer> getAllYears(){
    return dao.getAllYears();
}

public List<String> getAllArtists(){
    return dao.getAllArtists();
}

public List<Song> getAllYearArtistSongs(String artista, int anno){
    return dao.getAllYearArtistSongs(artista, anno);
}

```

*Metodi di accesso al DAO nella classe Model*

```

public List<Song> getAllYearArtistSongs(String artist, int year) {
    final String sql = "SELECT DISTINCT * FROM top10s WHERE artist = ? AND year = ? GROUP BY title";
    List<Song> songs = new ArrayList<>();

    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(sql);
        st.setString(1, artist);
        st.setInt(2, year);

        ResultSet rs = st.executeQuery();

        while (rs.next()) {
            Song s = new Song (rs.getInt("id"), rs.getString("title"), rs.getString("artist"),
                rs.getString("top_genre"), rs.getInt("year"), rs.getInt("bpm"),
                rs.getInt("nrgy"), rs.getInt("dnce"), rs.getInt("dB"),
                rs.getInt("live"), rs.getInt("val"), rs.getInt("dur"), rs.getInt("acous"),
                rs.getInt("spch"), rs.getInt("pop"));
            songs.add(s);
        }
        conn.close();
        return songs;
    } catch (SQLException e) {
        e.printStackTrace();
        throw new RuntimeException("Errore Db");
    }
}

```

*Metodo getAllYearArtistSongs() nella classe SpotifyDAO. Dato un artista e un anno, inseriti in input, restituisce tutte le canzoni di quell'artista in quell'anno.*

È possibile notare come all'interno del DAO siano state effettuate delle query SQL utilizzando dei *Prepared Statement*, che riducono il rischio di SQL Injection da parte di un utente malevolo, garantendo una maggiore sicurezza per il database.

## 2° tab: Genera Playlist

Questa seconda parte include un algoritmo ricorsivo, che è il cuore dell'applicazione essendo l'algoritmo che genera la playlist.

Questo metodo, chiamato `generaPlaylistOttima()`, riceve dal Controller i parametri di durata, popularity, energy e danceability. Successivamente, richiama il DAO per ottenere una lista di tutte le canzoni che hanno affinità con i valori inseriti dall'utente.

```
public Set<Song> generaPlaylistOttima(int durata, double popularity, double energy, double danceability) {
    long start = System.currentTimeMillis();
    affinitaMin = Integer.MAX_VALUE;
    parzialeGiaAnalizzato.clear();

    List<Song> canzoniEstrate = new ArrayList<>(dao.getCanzoniAffini(durata, popularity, energy, danceability));
    Set<Song> listaCanzoniAffini = new HashSet<>();

    double indice = popularity + energy + danceability;

    for(Song s : canzoniEstrate) {
        s.setIndicePlaylist(indice);
        s.setAffinita();
    }
    Collections.sort(canzoniEstrate);

    if (canzoniEstrate.size()>20) {
        listaCanzoniAffini.addAll(canzoniEstrate.subList(0, 20));
    } else {
        listaCanzoniAffini.addAll(canzoniEstrate);
    }

    best = new HashSet<>();
    Set<Song> parziale = new HashSet<>();

    int sommaDurata = 0;
    int affinitaTot = 0;

    if(durata<0)
        return parziale;

    if (durata<4) {
        return parziale;
    }

    cerca(parziale, durata, listaCanzoniAffini, indice, sommaDurata, affinitaTot);

    int durataCanzoniAffini = 0;
    for(Song s : listaCanzoniAffini) {
        durataCanzoniAffini += s.getDur();
    }

    if(best.isEmpty()) {
        if(durata>(durataCanzoniAffini + 180))
            return listaCanzoniAffini;
    }

    long end = System.currentTimeMillis();

    System.out.println("Durata: "+(int)(end-start)/1000+" s");

    return best;
}
```

Una volta estratte le canzoni affini, procede con la ricerca ricorsiva della soluzione ottima. In particolare, richiama il metodo `cerca()`, che riceve come parametri due set di canzoni denominati *parziale* e *listaCanzoniAffini*, la *durata* inserita da tastiera, l'*indice* di affinità generale della canzone (dato dalla somma dei 3 parametri), e due valori interi *sommaDurata* e *affinitaTot* che danno indicazione della durata e dell'affinità della playlist parziale man mano che viene costruita.

Il metodo `cerca()` consiste nella vera e propria ricorsione: questo metodo richiama sé stesso fin quando non si verifica la condizione di terminazione.

In questo caso la condizione di terminazione è determinata dal raggiungimento, da parte della playlist *parziale*, della durata scelta dall'utente  $\pm 3$  minuti. È stato scelto questo intervallo di tolleranza per rendere la creazione flessibile, in modo tale da evitare che una playlist che differisca leggermente della durata imposta possa essere esclusa.

```
private void cerca(Set<Song> parziale, int durata, Set<Song> lista, double indice, int sommaDurata, int affinitaTot) {  
    //caso terminale  
    if (sommaDurata >= (durata) - 180 && sommaDurata <= (durata) + 180) {  
        if (!parziale.isEmpty()) {  
            if (affinitaTot / parziale.size() < affinitaMin) {  
                affinitaMin = affinitaTot / parziale.size();  
                affinitaTot = 0;  
                sommaDurata = 0;  
                this.best.clear();  
                this.best.addAll(parziale);  
            }  
        }  
        return;  
    }  
  
    for (Song song : lista) {  
        if (!parziale.contains(song)) {  
            parziale.add(song);  
            if (aggiuntaValida(parziale)) {  
                parzialeGiaAnalizzato.add(parziale);  
                sommaDurata += song.getDur();  
                affinitaTot += Math.abs(song.getAffinita() - indice);  
                cerca(parziale, durata, lista, indice, sommaDurata, affinitaTot);  
                parziale.remove(song);  
                sommaDurata -= song.getDur();  
                affinitaTot -= Math.abs(song.getAffinita() - indice);  
            }  
            else {  
                parziale.remove(song);  
            }  
        }  
    }  
}  
  
private boolean aggiuntaValida(Set<Song> parziale) {  
    if (parzialeGiaAnalizzato.contains(parziale)) {  
        return false;  
    }  
    return true;  
}
```

Finché la condizione di terminazione non è verificata, l'algoritmo continua a richiamare se stesso aggiungendo ad ogni passaggio una canzone all'interno della playlist *parziale*.

Per evitare il ripetersi delle varie combinazioni di canzoni in ordine diverso, che graverebbero sul tempo di esecuzione rendendolo inaccettabile, l'algoritmo verifica preventivamente se il set di canzoni è stato già analizzato, a prescindere dall'ordine con cui sono state inserite le canzoni all'interno.

Una volta raggiunta la condizione di terminazione, l'algoritmo verifica se questa combinazione di canzoni è quella con maggiore affinità con i

parametri scelti dall'utente. Se ciò è verificato, lo aggiunge al set *best*, che contiene la playlist migliore.

Una volta terminate tutte le iterazioni, il metodo *generaPlaylistOttima()* restituisce il set al Controller, che lo stampa nella text area dell'interfaccia grafica.

```
@FXML
void doGenera(ActionEvent event) {
    txtAreaGenera.clear();
    txtDurataPlaylist.setText("");
    txtNumeroCanzoni.setText("");
    pieChartPlaylist.getData().clear();
    int durata = 0;
    try {
        durata = Integer.parseInt(txtFieldDurata.getText());
    } catch (NumberFormatException e) {
        txtAreaGenera.appendText("Inserire nel campo Durata i minuti di durata massima della playlist");
        return;
    }

    if(durata<0) {
        txtAreaGenera.setText("Valore durata non valido\n");
        return;
    }

    if(durata<4) {
        txtAreaGenera.setText("Valore durata troppo basso\n");
        return;
    }

    double popularity = sliderPopularity.getValue();
    double energy = sliderEnergy.getValue();
    double danceability = sliderDanceability.getValue();

    durata = durata*60;

    if(!canzoniAggiunteManualmente.isEmpty()) {
        for(Song s : canzoniAggiunteManualmente) {
            durata -= s.getDur();
        }
    }

    Set<Song> risultato = model.generaPlaylistOttima(durata, popularity, energy, danceability);

    if(risultato == null) {
        return;
    }

    int duplicati = 0;
    for(Song s : canzoniAggiunteManualmente) {
        if(risultato.contains(s)) {
            risultato.remove(s);
            duplicati++;
        }
    }

    List<Song> playlistFinale = new ArrayList<>();
    playlistFinale.addAll(canzoniAggiunteManualmente);
    playlistFinale.addAll(risultato);

    int dur = 0;
    for(Song s : playlistFinale) {
        dur+=s.getDur();
        txtAreaGenera.appendText(s.getArtist()+" - "+s.getTitle()+"\n");
    }
}
```

*Estratto del metodo doGenera() nel Controller, che viene richiamato alla pressione del pulsante Genera*

```

public List<Song> getCanzoniAffini(int durata, double popularity, double energy, double danceability) {
    List<Song> canzoniAffini = new ArrayList<>();
    final String sql = "SELECT * FROM top10s WHERE pop BETWEEN ? AND ? "
        + "AND nrgy BETWEEN ? AND ? "
        + "AND dnce BETWEEN ? AND ? GROUP BY title";

    double popInf = popularity - 5;
    double popSup = popularity + 5;
    double nrgyInf = energy - 5;
    double nrgySup = energy + 5;
    double dnceInf = danceability - 5;
    double dnceSup = danceability + 5;

    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(sql);
        st.setDouble(1, popInf);
        st.setDouble(2, popSup);
        st.setDouble(3, nrgyInf);
        st.setDouble(4, nrgySup);
        st.setDouble(5, dnceInf);
        st.setDouble(6, dnceSup);

        ResultSet rs = st.executeQuery();

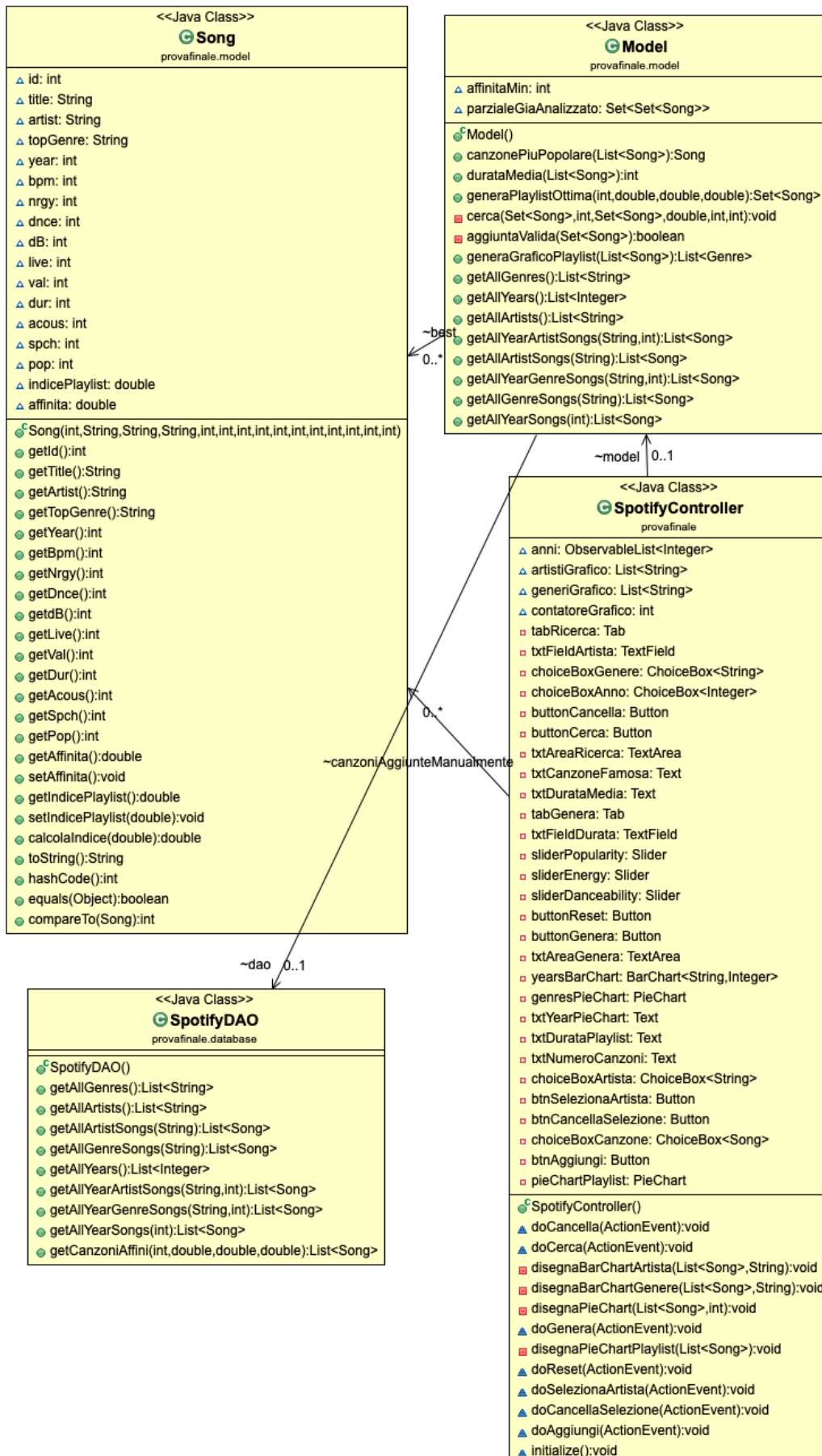
        while (rs.next()) {
            Song s = new Song (rs.getInt("id"), rs.getString("title"), rs.getString("artist"),
                rs.getString("top_genre"), rs.getInt("year"), rs.getInt("bpm"), rs.getInt("nrgy"),
                rs.getInt("dnce"), rs.getInt("dB"), rs.getInt("live"), rs.getInt("val"), rs.getInt("dur"),
                rs.getInt("acous"), rs.getInt("spch"), rs.getInt("pop"));
            canzoniAffini.add(s);
        }
        conn.close();
        return canzoniAffini;
    } catch (SQLException e) {
        e.printStackTrace();
        throw new RuntimeException("Errore Db");
    }
}

```

*Metodo getCanzoniAffini() all'interno del DAO*

Vi sono ulteriori metodi che consentono l'estrazione e la visualizzazione dei dati e l'elaborazione di grafici, ma che non saranno trattati in questa relazione poiché di rilevanza algoritmica limitata.

## 5. Diagramma delle classi principali





## 6. Screenshot dell'applicazione e risultati ottenuti

Come accennato precedentemente, l'interfaccia grafica di questo applicativo si divide in due tab.

Il primo tab, denominato *Ricerca brani*, mostra all'utente un text field che gli permette di ricercare un artista e due choice box che gli permettono di selezionare il genere e l'anno, oltre ai pulsanti *Cancella* (che permette di cancellare ogni selezione e risultato generato) e *Cerca* (che avvia la ricerca). I risultati generati vengono inseriti nella text area.

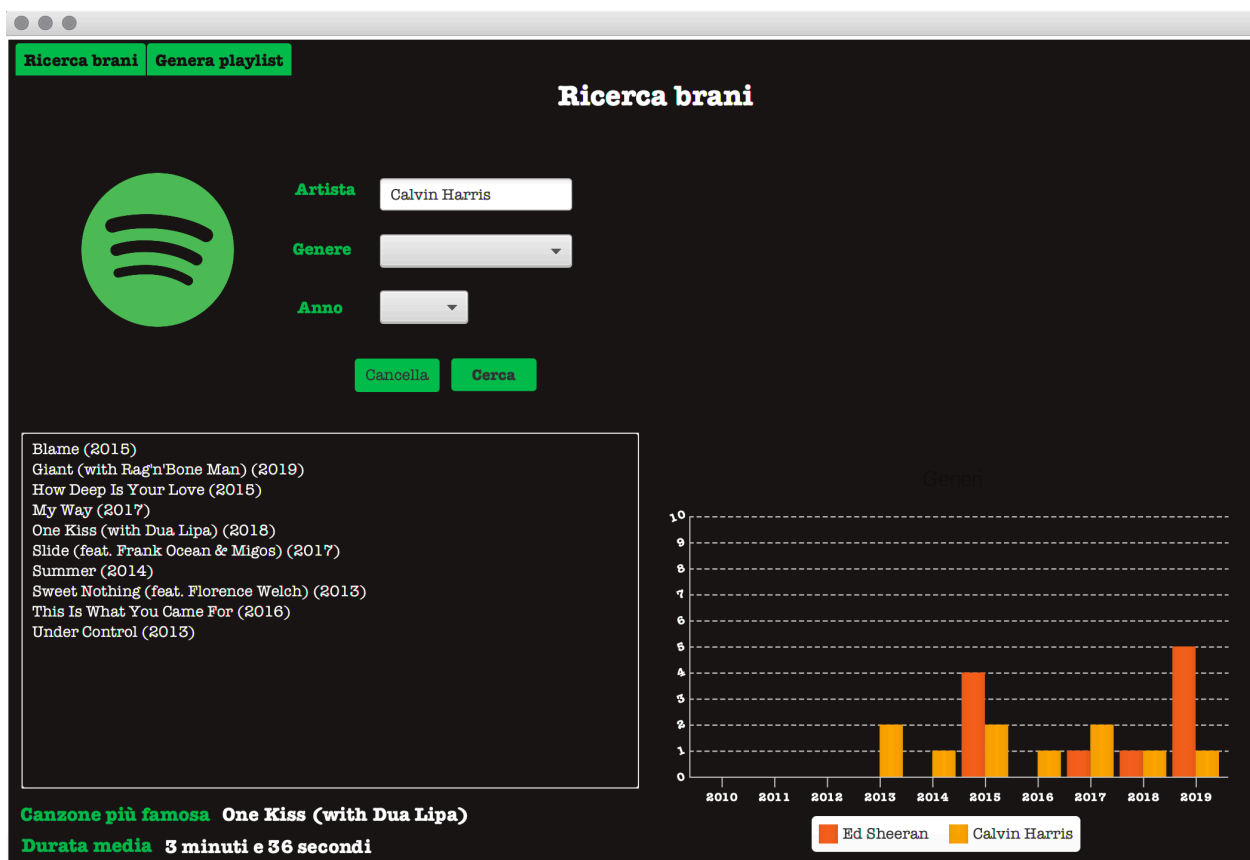
Al fondo, viene mostrata la canzone più famosa e la durata media delle canzoni ottenute come risultato.

La ricerca non può avvenire né combinando tutti e tre i parametri, né combinando artista e genere, in quanto quasi sempre un artista ha un suo genere principale sotto cui possono categorizzarsi le sue canzoni, e quindi la ricerca perderebbe di significato.

The screenshot shows a web application window titled "Ricerca brani". At the top, there are two tabs: "Ricerca brani" (active) and "Genera playlist". The main content area has a dark background. On the left, there is a large green Spotify logo. To its right, there are three input fields: "Artista" (a text field), "Genere" (a dropdown menu), and "Anno" (a dropdown menu). Below these fields are two buttons: "Cancella" and "Cerca". Below the buttons is a large, empty rectangular box for displaying search results. At the bottom left, there are two labels: "Canzone più famosa" and "Durata media".

È possibile ricercare le canzoni solo per artista, solo per genere e solo per anno oppure ricercare la combinazione artista e anno o genere e anno.

Nel primo caso, sono mostrate le canzoni dell'artista o del genere o dell'anno selezionato.

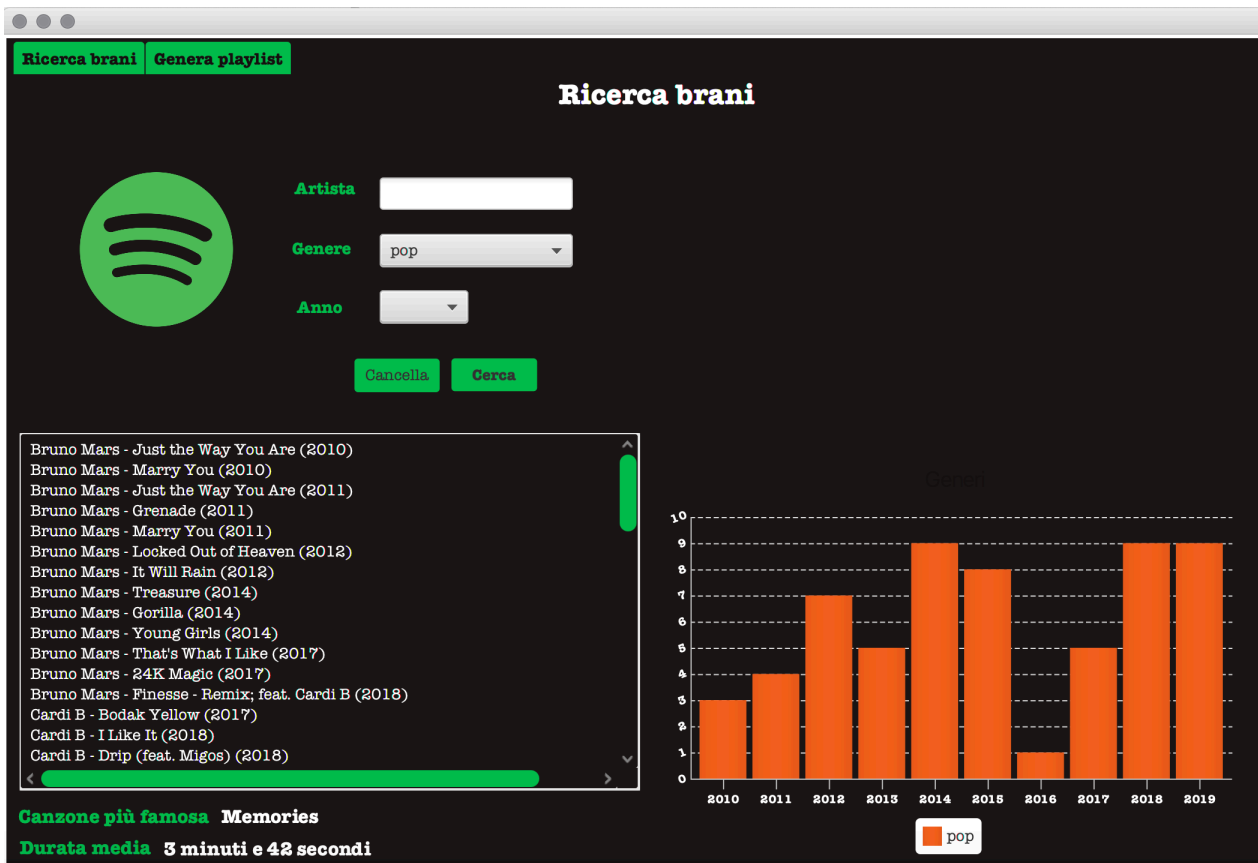


Se si cerca un artista o un genere, nello spazio in basso a destra compare un diagramma a barre, mostrando la frequenza per anno delle canzoni ottenute.

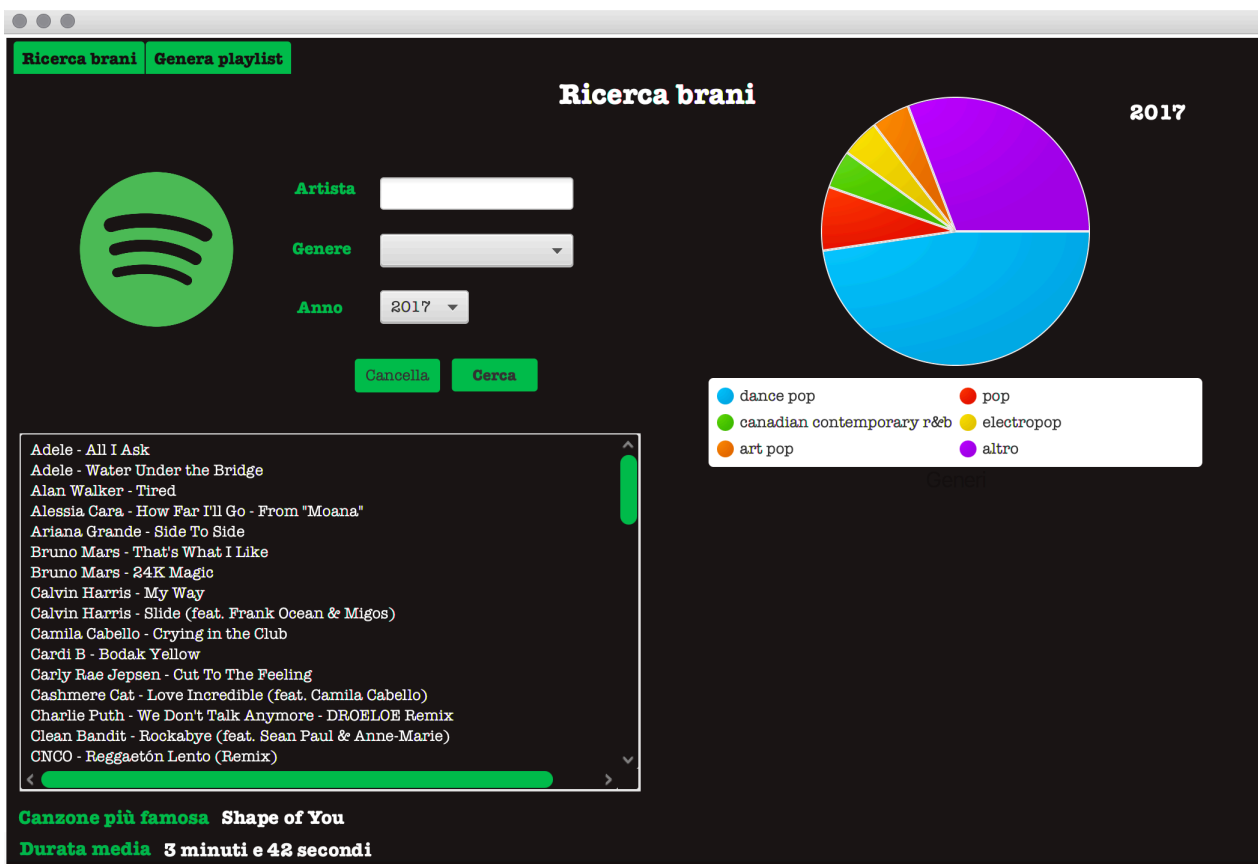
È inoltre possibile ricercare un altro artista o un altro genere, senza premere sul pulsante *Reset*, per poter effettuare un confronto grafico, come mostrato nell'immagine sopra.

Se la ricerca avviene per anno, sono mostrate tutte le canzoni di quell'anno specifico, e nella parte destra superiore dell'applicazione è mostrato un grafico a torta che mette in evidenza i principali generi delle canzoni dell'anno.

Nel secondo caso, la ricerca può essere combinata e quindi sono mostrate le canzoni di un artista o un genere limitate all'anno selezionato.



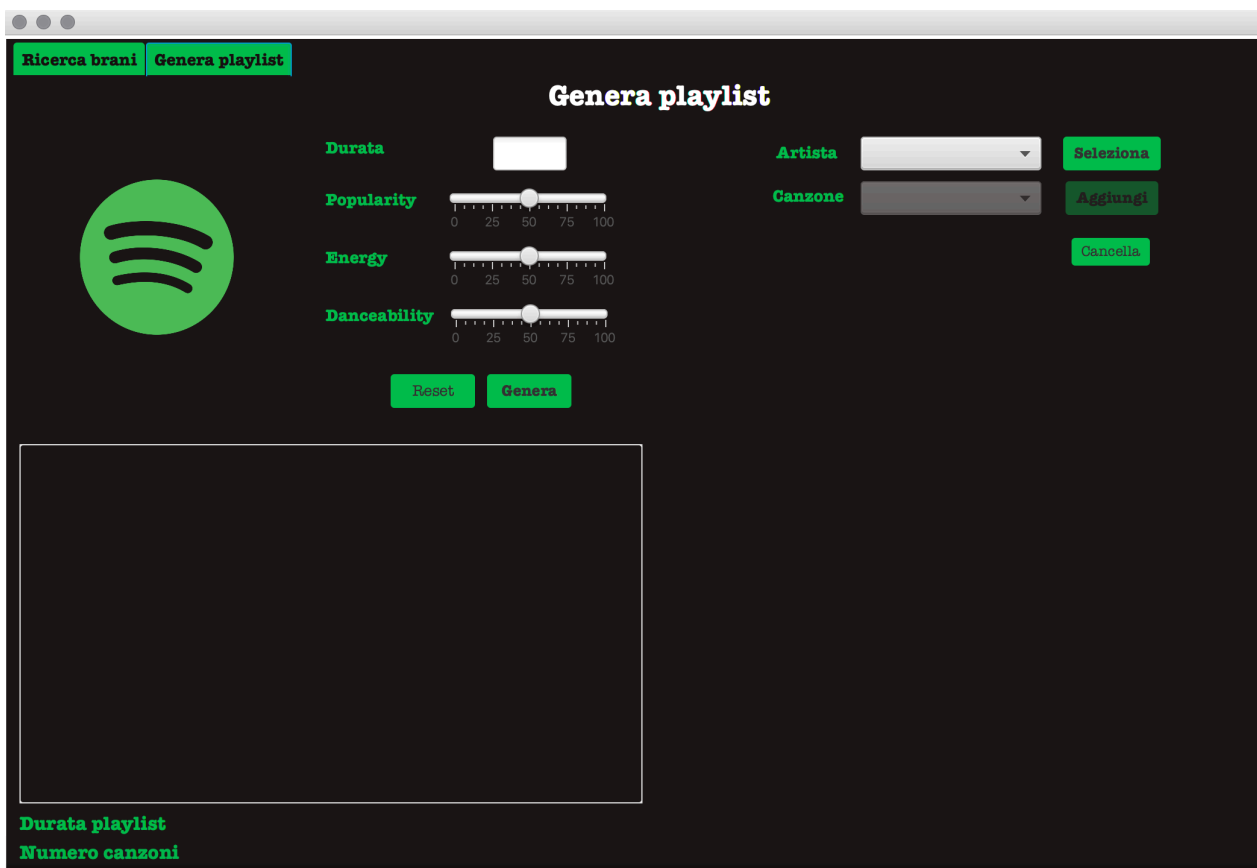
Ricerca per genere



Ricerca per anno

Il secondo tab, denominato *Genera Playlist*, mostra nella parte sinistra un text field per l'inserimento della durata e tre slider relativi ai parametri Popularity, Energy e Danceability, con i quali è possibile selezionare un valore da 0 a 100.

Sono presenti i pulsanti *Reset*, che ha le stesse funzioni del suo analogo del primo tab, e *Genera*, che avvia la generazione della playlist. La playlist è mostrata all'interno della text area.

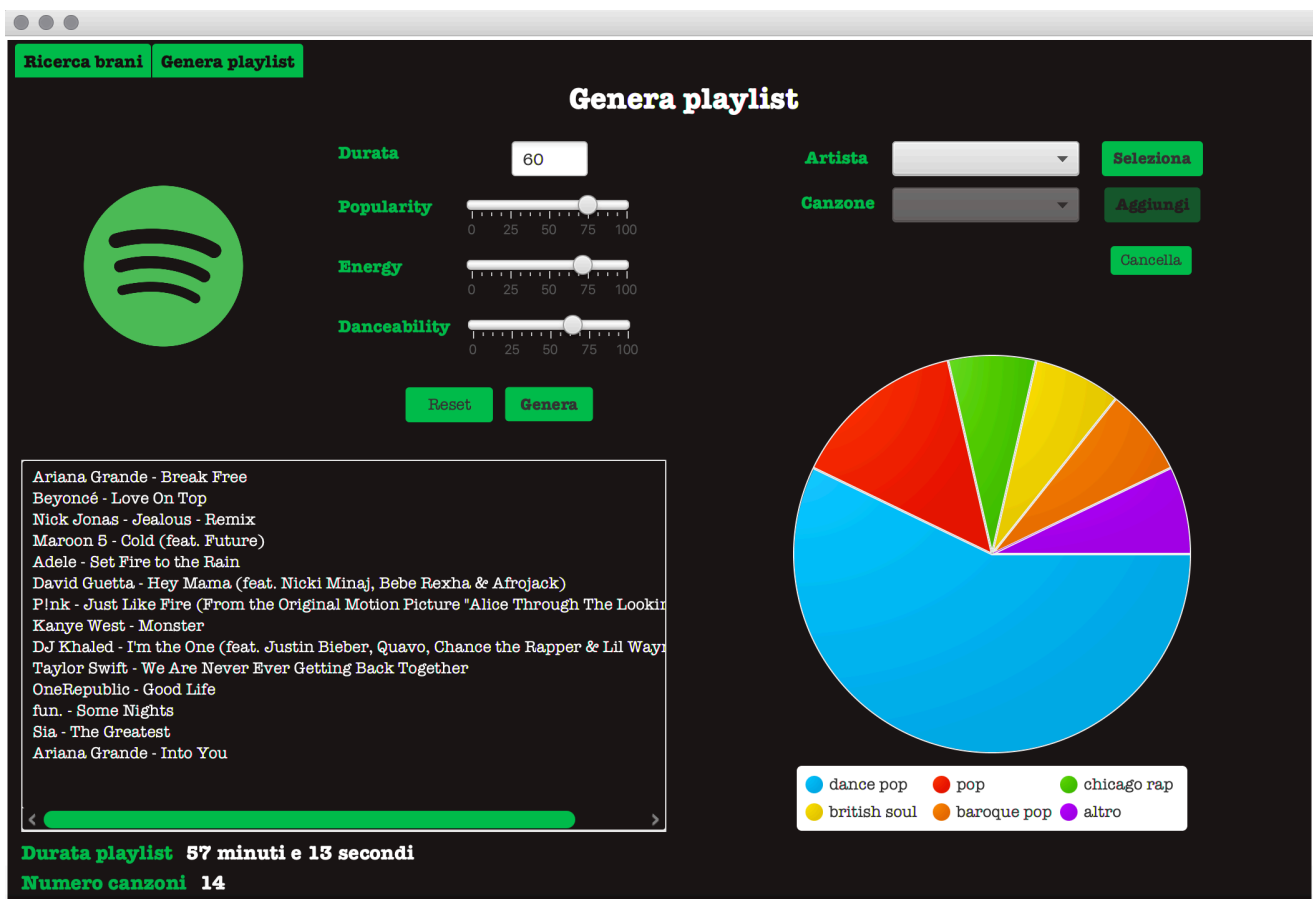


Al fondo, sono mostrate la durata della playlist e il numero di canzoni che contiene.

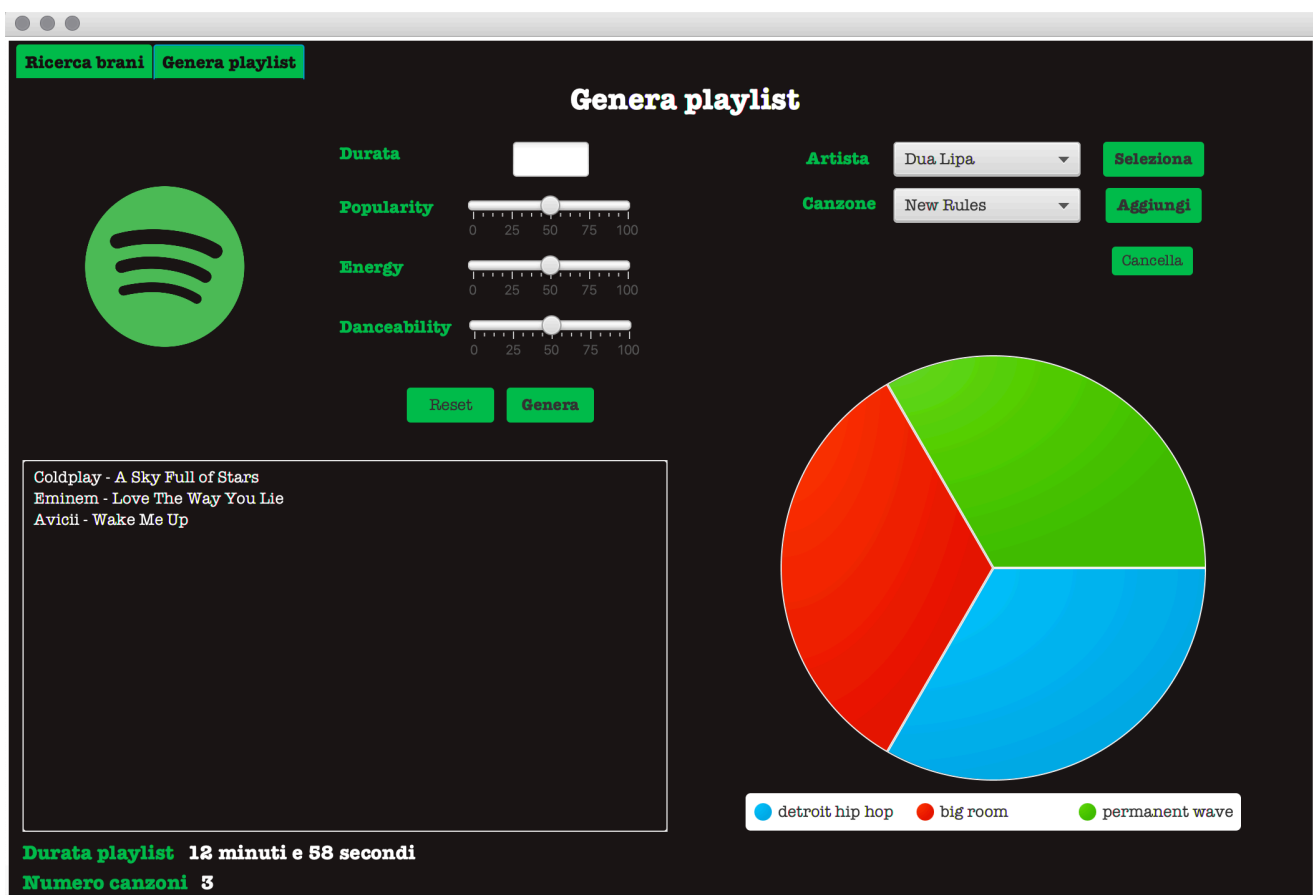
Nella parte destra, sono presenti due choice box. Il primo permette di selezionare un artista tra quelli presenti nel database; una volta confermata la scelta tramite il tasto *Seleziona*, nel secondo choice box sono mostrate tutte le canzoni di quell'artista.

Cliccando sul tasto *Aggiungi*, è possibile aggiungere manualmente una canzone all'interno della playlist.

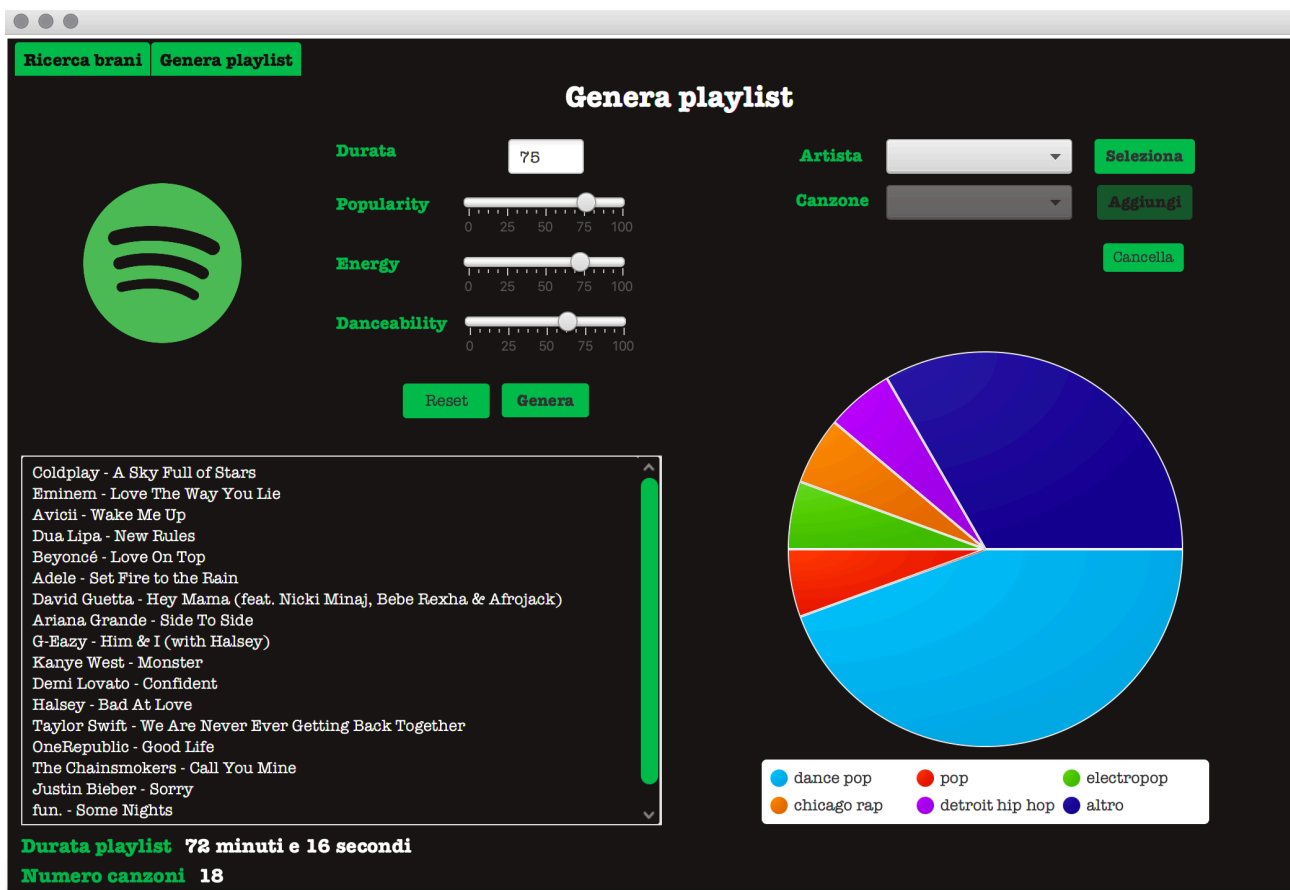
Ogni playlist che viene generata successivamente contiene sempre le canzoni aggiunte manualmente, fin quando l'utente non preme il pulsante *Cancella*, che elimina la selezione manuale effettuata.



Generazione della playlist, durata selezionata 60 minuti



Aggiunta manuale delle canzoni alla playlist



*Playlist generata automaticamente, contenente anche canzoni aggiunte manualmente*

In basso a destra, viene generato un grafico a torta che mostra i generi principali della playlist.

Nell'immagine sopra, è stata generata una playlist di durata 72 minuti e 16 secondi, composta da 4 canzoni aggiunte manualmente e 14 aggiunte automaticamente dall'algoritmo. La durata ottenuta rispetta la durata massima inserita dall'utente, ma in ogni caso sarebbe stata accettabile anche una playlist con durata leggermente superiore a quella massima imposta.

Ai fini di garantire bassi tempi di esecuzione, la generazione della playlist migliore è stata limitata ad un elenco delle 20 canzoni più affini, ovvero circa 70-80 minuti di durata complessiva; questo perché con un numero di canzoni superiore i tempi sarebbero stati più lunghi e non necessari ai fini della dimostrazione del progetto.

Con questa impostazione, i tempi di esecuzione sono garantiti tra 0 e 3 secondi.

Al seguente link è disponibile una video dimostrazione del progetto:

<https://youtube.com/watch?v=flzCuRuCNtk>

## 7. Valutazioni e conclusioni

L'applicazione è stata realizzata con lo scopo di fornire all'utente abbonato un servizio automatico aggiuntivo integrato all'interno di Spotify.

Potenzialmente, questo software è implementabile all'interno di qualunque servizio di streaming musicale, utilizzando il data-set corretto.

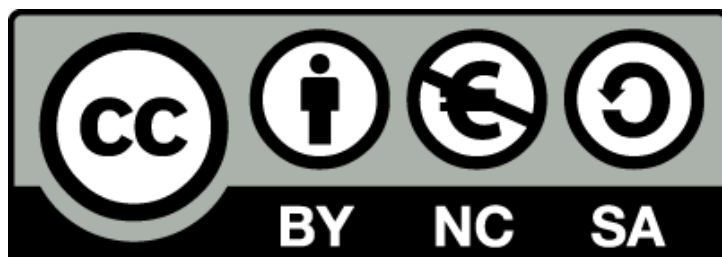
Tra le criticità principali, possiamo evidenziare le seguenti:

- Il data-set che è stato utilizzato contiene soltanto 584 canzoni, mentre sarebbe stato più utile utilizzare un data-set più completo, ma più difficile da reperire in rete;
- Si sarebbero ottenuti dei risultati più completi evitando di limitare la lista di canzoni più affini, ma questo avrebbe gravato troppo sui tempi di elaborazione. Sarebbe stato possibile aggiungere la possibilità all'utente di scegliere tra generazione rapida, più limitata in termini di durata, e generazione lunga, ma che avrebbe permesso la creazione di playlist di durata più lunga;
- Poiché si tratta di canzoni presenti nella classifica Top 50, la maggior parte dei brani ha un'alta popolarità e generalmente un'alta energia e danzabilità; di conseguenza impostando questi valori bassi, la playlist generata potrebbe avere una durata inferiore al valore scelto dall'utente a causa della carenza di brani con quelle specifiche;
- Si potrebbero inoltre aggiungere numerose altre funzionalità, ad esempio aumentare, su richiesta dell'utente, il numero di parametri selezionabili per fornire un risultato più accurato oppure dare la possibilità all'utente di generare una playlist di canzoni di uno o più generi a scelta.

L'obiettivo del progetto, seppur con qualche limitazione, è stato raggiunto. L'interfaccia grafica, a mio parere, risulta intuitiva e l'applicativo è pienamente funzionante e utilizzabile da chiunque volesse usufruirne.

Il progetto completo è disponibile al seguente indirizzo:

<https://github.com/TdP-prove-finali/DeCarneWalter>



This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

To view a copy of this license, visit  
<http://creativecommons.org/licenses/by-nc-sa/4.0/>