



**Politecnico  
di Torino**

## **Politecnico di Torino**

Corso di Laurea in Ingegneria Gestionale

A.a. 2023/2024

Sviluppo di un'applicazione software  
per la generazione automatica di playlist  
attraverso la ricerca ricorsiva e grafi pesati

Relatore: Fulvio Corno

Candidato: Carlo Marra

# Contents

<b>1</b>	<b>Problema proposto</b>	<b>1</b>
1.1	Studente proponente . . . . .	1
1.2	Titolo della proposta . . . . .	1
1.3	Descrizione del problema proposto . . . . .	1
1.4	Descrizione della rilevanza gestionale del problema . . . . .	1
1.5	Descrizione dei data-set per la valutazione . . . . .	1
1.6	Descrizione preliminare degli algoritmi coinvolti . . . . .	2
1.7	Descrizione preliminare delle funzionalità previste per l'applicazione software . . . . .	3
<b>2</b>	<b>Descrizione del problema affrontato</b>	<b>5</b>
2.1	Contesto operativo . . . . .	5
2.2	Descrizione del sotto-problema . . . . .	5
<b>3</b>	<b>Dataset utilizzato</b>	<b>6</b>
3.1	Introduzione . . . . .	6
3.2	Tabella <i>tracks</i> . . . . .	6
<b>4</b>	<b>View (scene.fxml)</b>	<b>8</b>
4.1	Elementi comuni . . . . .	8
4.2	Surprise Me . . . . .	8
4.3	My Mix . . . . .	10
<b>5</b>	<b>Strutture dati e algoritmi principali</b>	<b>12</b>
5.1	Package it.polito.PlaylistGen . . . . .	12
5.1.1	FXMLController . . . . .	12
5.2	Package it.polito.PlaylistGen.db . . . . .	17
5.2.1	DBConnect . . . . .	17
5.2.2	TracksDAO . . . . .	18
5.3	Package it.polito.PlaylistGen.model . . . . .	21
5.3.1	Model . . . . .	21
5.3.2	Preference . . . . .	23
5.3.3	Track . . . . .	23
5.3.4	Vertex . . . . .	24
<b>6</b>	<b>Ricorsione</b>	<b>25</b>
<b>7</b>	<b>Diagramma delle classi</b>	<b>28</b>
<b>8</b>	<b>Risultati ottenuti e considerazioni finali</b>	<b>29</b>

# 1 Problema proposto

## 1.1 Studente proponente

s249863

Carlo Marra

## 1.2 Titolo della proposta

Sviluppo di un'applicazione software per la generazione automatica di playlist attraverso la ricerca ricorsiva e grafi pesati.

## 1.3 Descrizione del problema proposto

Creare una playlist soddisfacente è una sfida comune per ogni individuo. Ogni persona ha gusti musicali unici e mutevoli, rendendo difficile la selezione di brani che rispecchino perfettamente il proprio umore e le preferenze del momento.

La vasta gamma di generi musicali, artisti e canzoni disponibili rende ancora più complicata la creazione di una playlist che riesca a catturare l'essenza di chi siamo. La soluzione proposta mira alla ricerca della miglior playlist possibile a seguito di alcune linee guida dell'utente.

## 1.4 Descrizione della rilevanza gestionale del problema

Per molti individui, trovare una playlist adatta alla situazione può svolgere un ruolo significativo nella motivazione per iniziare un progetto o un viaggio. La musica ha il potere di influenzare le nostre emozioni, creando l'atmosfera e l'energia desiderate. Una playlist ben selezionata può stimolare la creatività, migliorare il focus, la produttività e amplificare l'entusiasmo per l'attività imminente.

## 1.5 Descrizione dei data-set per la valutazione

Il data-set utilizzato sarà ispirato a "Spotify - All Time Top 2000s Mega Dataset". Questo dataset contiene le statistiche audio di 2000 brani tra i più ascoltati su Spotify. Sono inclusi brani pubblicati dal 1956 al 2019 di artisti noti e famosi come Queen, The Beatles, Guns N' Roses, ecc.

Questi dati sono estratti dalla playlist di Spotify "Top 2000s"<sup>1</sup> utilizzando Selenium con Python.

---

<sup>1</sup><http://sortyourmusic.playlistmachinery.com>

## 1.6 Descrizione preliminare degli algoritmi coinvolti

L'applicazione si dividerà in due sezioni indipendenti: la sezione "Surprise Me" e la sezione "My Mix".

### Sezione "Surprise Me"

L'utente avrà il compito di specificare alcuni requisiti che il risultato dovrà soddisfare. Tra queste vi sono:

- Il genere musicale della playlist desiderata e, una volta scelto, facoltativamente l'artista preferito appartenente al genere scelto precedentemente.
- Il grado di presenza richiesta delle cinque caratteristiche principali, Energia (E), Ballabilità (D), Positività (V), Acustica (A) e Popolarità (P) attraverso un punteggio da 1 a 10.

Gli input ricevuti saranno usati per creare un grafo pesato in cui i nodi saranno tutte le canzoni che soddisfano il genere scelto. Ad ogni traccia verrà attribuito un punteggio di affinità con le cinque caratteristiche richieste. Dati i cinque gradi ( $G_i$ ) ricevuti e ciascun brano  $B_i$  che rientra nel genere scelto:

$$\text{Punteggio}(B_i) = G_E \cdot E(B_i) + G_D \cdot D(B_i) + G_V \cdot V(B_i) + G_A \cdot A(B_i) + G_P \cdot P(B_i)$$

Attraverso questo punteggio associato a ciascun vertice, il peso di ciascun arco sarà calcolato attraverso la differenza (in valore assoluto) di punteggio dei due nodi considerati, di conseguenza un peso basso corrisponderà ad un alta affinità.

Una volta ottenuto il grafo, si procederà con una ricerca ricorsiva (opportunamente ottimizzata) che troverà la playlist con il più basso valore totale di archi percorsi. In caso di più possibili affinità con il brano successivo e la presenza di un brano dell'artista preferito, la scelta ricadrà su quest'ultimo in caso di pesi degli archi simili.

L'idea di limitare la ricerca ad un singolo genere potrà essere allargata a due o più in caso di buone prestazioni dell'algoritmo. La playlist trovata sarà poi inserita all'interno della tabella dell'applicazione software e il suo punteggio all'intento della casella di testo sottostante.

## Sezione "My Mix"

L'utente potrà aggiungere, prima della ricerca, le sue canzoni preferite inserendo rispettivamente genere, artista e brano. Una volta aggiunte, potrà decidere il numero massimo di canzoni nella playlist o il numero massimo di minuti totali. A quel punto si procederà allo studio delle canzoni inserite per creare una playlist che sia coerente per genere e attributi.

Per la determinazione dei gusti dell'utente si farà la media della somma di ciascun attributo del brano  $i$ -esimo ( $B_i$ ). Preso come esempio l'attributo "Popolarità" e  $N$  brani in input:

$$\bar{P} = \frac{\sum_{i=1}^N P(i)}{N}$$

Il punteggio di ciascun potenziale brano sarà quindi:

$$\text{Punteggio}(B_i) = \frac{\bar{E}}{10} \cdot E(B_i) + \frac{\bar{D}}{10} \cdot D(B_i) + \frac{\bar{V}}{10} \cdot V(B_i) + \frac{\bar{A}}{10} \cdot A(B_i) + \frac{\bar{P}}{10} \cdot P(B_i)$$

La divisione di ciascun fattore per 10 è solamente per rendere i punteggi consistenti con quelli della prima sezione (da 1 a 10).

Si procederà quindi, come nella sezione precedente, alla creazione del grafo pesato con le relative affinità e alla ricerca ricorsiva della playlist ottima.

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

L'applicazione software di ciascuna delle due sezioni sarà divisa graficamente in due VBox, una a sinistra e una a destra, rispettivamente per la descrizione delle preferenze e per la visione del risultato ottenuto. In ciascuna Tab si avrà nella parte bassa una Label che conterrà alcune linee guida per la buona riuscita dell'applicazione e una piccola descrizione di ciascun valore degli Slider.

### Sezione "Surprise Me"

La parte di sinistra della sezione "Surprise Me" conterrà:

- Due TextArea per la scelta del numero massimo di brani richiesto e per i minuti massimi totali della playlist. Queste due TextArea saranno accompagnate da due CheckBox per segnalare quali dei due vincoli l'utente preferisce che venga rispettato.
- Una ChoiceBox per la scelta del genere preferito e una ChoiceBox per la scelta dell'artista preferito. Questa verrà compilata al momento della scelta del genere preferito.
- Cinque Slider orizzontali che, attraverso una scala da 1 a 10, serviranno all'utente per condividere i propri gusti e aspettative sulla playlist finale. I cinque attributi degli Slider saranno Energia, Ballabilità, Positività, Acustica e Popolarità.
- Due bottoni per il reset degli input e per l'avvio della ricerca.

La VBox di destra conterrà:

- Una tabella per la visualizzazione del risultato finale.
- Una casella di testo non modificabile per alcune statistiche legate al risultato finale.

### **Sezione "My Mix"**

La parte di sinistra conterrà nello specifico:

- Due TextArea per la scelta del numero massimo di brani richiesto e per i minuti massimi totali della playlist. Queste due TextArea saranno accompagnate da due CheckBox per segnalare quali dei due vincoli l'utente preferisce che venga rispettato.
- Una ChoiceBox per la scelta del genere preferito. Il genere scelto servirà a costruire il grafo su cui eseguire la ricerca.
- Una ChoiceBox per la scelta dell'artista preferito. Questa verrà compilata al momento della scelta del genere preferito.
- Una ChoiceBox per la scelta del brano dell'artista preferito. Questa verrà compilata al momento della scelta dell'artista preferito.
- Quattro Buttons, rispettivamente per il reset degli input, per l'aggiunta della canzone selezionata alla playlist di partenza, per l'eliminazione dell'ultimo brano inserito e per l'avvio della ricerca.

La VBox di destra conterrà:

- Una tabella per la visualizzazione del risultato finale.
- Una casella di testo non modificabile per alcune statistiche legate al risultato finale.

## 2 Descrizione del problema affrontato

### 2.1 Contesto operativo

Mettendosi nei panni di un'azienda che opera nel campo del mercato musicale, risulta evidente come la possibilità di consigliare playlist personalizzate ai propri utenti offre diversi vantaggi sia per gli utenti che per l'azienda stessa. Tra questi è possibile trovare:

- **Fidelizzazione degli utenti:** Offrire playlist personalizzate crea un'esperienza più coinvolgente per gli utenti, che si sentono compresi e apprezzati.
- **Aumento dell'engagement:** Le playlist personalizzate tengono gli utenti impegnati sulla piattaforma musicale più a lungo, aumentando il tempo complessivo trascorso e il coinvolgimento complessivo degli utenti.
- **Opportunità di cross-selling:** Attraverso la conoscenza dei gusti musicali degli utenti, un'azienda musicale può anche suggerire altri servizi o prodotti correlati che potrebbero interessare, creando opportunità di vendita incrociata.
- **Raccolta di dati utili:** L'analisi dei dati generati dalle preferenze musicali degli utenti fornisce informazioni preziose sull'andamento del mercato, i trend emergenti e le preferenze della clientela. Questi dati possono essere sfruttati per migliorare ulteriormente i servizi offerti e adeguare l'offerta alle esigenze degli utenti.

### 2.2 Descrizione del sotto-problema

Come accennato nel punto dell'**Aumento dell'engagement** del sotto-paragrafo precedente, un algoritmo per la creazione di playlist musicali basato su input numerici o preferenze personali offre una serie di vantaggi pratici e emotivi, migliorando l'accesso alla musica e arricchendo l'esperienza degli utenti nel loro contesto quotidiano. L'utente finale trarrà diversi vantaggi nell'impiego di questo algoritmo, tra cui:

- **Supporto emotivo e motivazionale:** La capacità di considerare parametri come il grado di positività o energia può essere utilizzata per creare playlist che si adattano alle diverse esigenze emotive degli utenti. Ad esempio, una playlist di alta energia potrebbe essere adatta per un allenamento fisico, mentre una playlist più rilassante potrebbe essere preferita per il relax serale.
- **Risparmio di tempo:** Nel contesto di playlist generate automaticamente, un algoritmo può risparmiare tempo agli utenti nell'individuare manualmente brani adatti ai loro stati d'animo o attività. L'automazione del processo di selezione della musica consente alle persone di concentrarsi su altre attività senza dover creare manualmente playlist o cercare singoli brani.
- **Contesto sociale e condivisione:** Algoritmi di questo tipo possono anche facilitare la condivisione e la scoperta di nuovi brani tra gli utenti con gusti simili. La capacità di suggerire canzoni in base a preferenze simili può favorire una maggiore connessione sociale attraverso la condivisione di esperienze musicali.

## 3 Dataset utilizzato

### 3.1 Introduzione

Il dataset utilizzato è basato su “Spotify - All Time Top 2000s Mega Dataset”<sup>2</sup>. Quest’ultimo contiene le statistiche audio di 2000 brani tra i più ascoltati su Spotify. I dati sono estratti dalla playlist di Spotify “Top 2000s” su PlaylistMachinery<sup>3</sup> utilizzando Selenium con Python.

Al fine di ottenere risultati consistenti per ogni tipo di playlist richiesta, il dataset di partenza è stato alterato mediante l’aggregazione di tracce di sottogeneri poco frequenti all’interno di un genere principale, con l’obiettivo di assicurare almeno 20 brani associati per ciascun genere.

### 3.2 Tabella *tracks*

<i>tracks</i>	
PK	Index
	Title
	Artist
	Top Genre
	Year
	Beats per Minute (BPM)
	Length
	Loudness (DB)
	Energy
	Danceability
	Acoustic
	Speechiness
	Popularity

La tabella utilizzata per la creazione del software è composta dai seguenti attributi:

- Index: Identificativo della traccia e PK della tabella;
- Title: Nome della traccia;
- Artist: Nome dell’artista;
- Top Genre: Genere della traccia;
- Year: Anno di pubblicazione della traccia;
- Beats per Minute (BPM): Il tempo della canzone in battiti per minuto;
- Length: La durata della canzone in secondi;
- Loudness (dB): Maggiore è il valore, più alta è la canzone;

---

<sup>2</sup><https://www.kaggle.com/datasets/iamsumat/spotify-top-2000s-mega-dataset>

<sup>3</sup><http://sortyourmusic.playlistmachinery.com>



I seguenti attributi, descrittivi delle caratteristiche principali di ciascun brano, sono espressi in valori interi compresi tra 0 e 100:

- Energy: L'energia di una canzone - maggiore è il valore, più energica è la canzone;
- Danceability: Maggiore è il valore, più facile è ballare su questa canzone;
- Valence: Maggiore è il valore, più positivo è l'umore della canzone;
- Acoustic: Maggiore è il valore, più acustica è la canzone;
- Speechiness: Maggiore è il valore, più parole parlate contiene la canzone;
- Popularity: Maggiore è il valore, più popolare è la canzone.

## 4 View (scene.fxml)

La View è responsabile della presentazione dei dati e dell'interazione con l'utente, assicurando che l'interfaccia utente sia intuitiva, risponda dinamicamente ai cambiamenti nel Model e fornisca un feedback chiaro durante l'utilizzo dell'applicazione.

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

La scena ideata è divisa in due Tab denominati rispettivamente "Surprise Me" e "My Mix".

### 4.1 Elementi comuni

Nel lato in alto a sinistra di entrambi i Tab è presente la possibilità di selezionare uno dei due vincoli che la playlist dovrà rispettare, cioè il numero di minuti totali dei brani al suo interno o più semplicemente il numero totale di brani.

I due vincoli sono stati implementati con due *RadioButton* affiancati da una *ComboBox* per selezionare il numero di brani desiderato (valori che vanno da 10 fino ad un massimo di 30) e da un *TextField* in cui specificare la somma di minuti totali. I valori accettati di quest'ultimo sono stati limitati a 180 minuti.

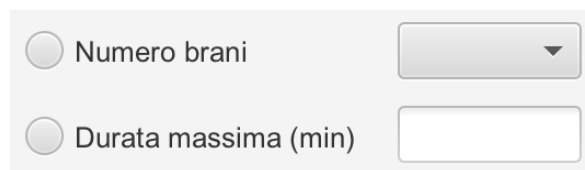


Figure 1: Vincoli comuni

Le due Tab condividono un *TableView* con due colonne, "Artista" e "Titolo", in cui sarà inserita la playlist ottima al termine della ricerca. Al di sotto della *TableView* vi è una *TextArea* non modificabile dall'utente dove sarà possibile ispezionare il numero di brani della playlist e il numero di minuti totali.

Un'altra caratteristica presente in entrambi i Tab è una *Label* nella parte bassa della window in cui sono presenti le istruzioni per poter utilizzare al meglio l'applicazione.

### 4.2 Surprise Me

La sezione "Surprise Me" è composta da una *SplitPane* in cui nella zona a sinistra sono presenti i controlli dedicati all'utente per la specificazione delle proprie preferenze e nella zona di destra una tabella in cui verrà visualizzata la playlist in output.

Al di sotto dei vincoli in comune (v. Elementi comuni) vi è la possibilità di scegliere il proprio genere preferito e, facoltativamente, l'artista preferito associato attraverso due *ComboBox*.

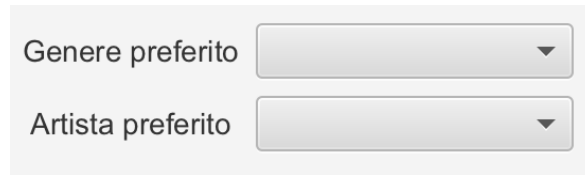


Figure 2 shows two dropdown menus. The first is labeled "Genere preferito" and the second is labeled "Artista preferito". Both menus have a downward arrow on the right side, indicating they are expandable.

Figure 2: ComboBox per la scelta del genere e dell'artista preferito

Sono presenti infine cinque Slider per indicare rispettivamente il grado di presenza richiesta delle cinque caratteristiche principali associate a ciascun brano, Energy (E), Danceability (D), Valence (V), Acoustic (A) e Popularity (P) attraverso un punteggio da 1 a 10.



Figure 3 displays five horizontal sliders, each corresponding to a characteristic: Energy, Danceability, Valence, Acoustic, and Popularity. Each slider has a circular handle at the left end (value 1) and a vertical tick mark at the right end (value 10). The sliders are arranged vertically, with the labels on the left and the numerical scale (1 to 10) below the handle.

Figure 3: Sliders

I cinque slider sono seguiti da due Button, uno per il reset dei valori fino a quel momento specificati e l'altro per l'avvio della ricerca della miglior playlist.

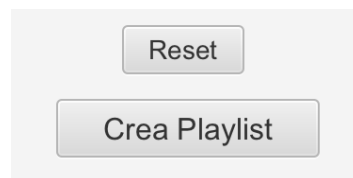


Figure 4 shows two buttons stacked vertically. The top button is labeled "Reset" and the bottom button is labeled "Crea Playlist". Both buttons have a light gray background and a thin border.

Figure 4: Buttons per reset e avvio della ricerca

### 4.3 My Mix

La sezione "My Mix" è composta da una SplitPane la cui zona a sinistra permette all'utente l'inserimento dei brani preferiti su cui verrà effettuato lo studio dei gusti e la conseguente ricerca. Nella zona di destra, come nella sezione precedente, è presente una tabella in cui verrà visualizzata la playlist in output.

L'inserimento all'interno della playlist dei brani preferiti avviene attraverso la selezione in fila del genere, artista e infine del brano desiderato attraverso tre ComboBox che si riempiono conseguentemente alle precedenti scelte effettuate.

The image shows a vertical stack of three ComboBox controls. Each control consists of a text label on the left and a rectangular dropdown box on the right. The labels are "Genere", "Artista", and "Brano" from top to bottom. Each dropdown box has a small downward-pointing arrow on its right side, indicating it is a selection menu.

Figure 5: ComboBox per la selezione del brano da inserire nella TableView

Le precedenti ComboBox sono seguite da quattro Buttons:

- Aggiungi: Aggiungere il brano selezionato alla TableView;
- Reset: Inizializzazione delle scelte effettuate all'interno delle ComboBox;
- Inizializza Playlist: rimozione di tutti i brani fino a quel momento aggiunti alla TableView;
- Rimuovi ultimo brano: rimozione dell'ultimo brano aggiunto alla TableView.

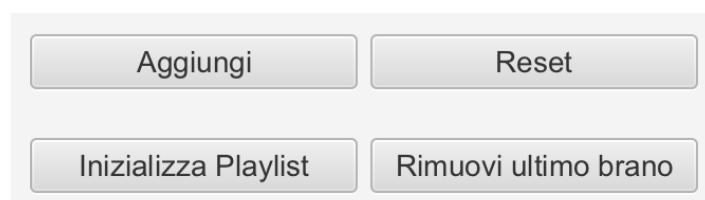
The image shows four rectangular buttons arranged in a 2x2 grid. The buttons are labeled "Aggiungi", "Reset", "Inizializza Playlist", and "Rimuovi ultimo brano" from top-left to bottom-right. Each button has a light gray background and a thin border.

Figure 6: Buttons della sezione "My Mix"

La splitPane di sinistra è infine conclusa con un Button per l'avvio della ricerca della playlist ottima.

Surprise Me

My Mix

☐ Numero brani

▼

☐ Durata massima (min)

Genere preferito

▼

Artista preferito

▼

Energy

1

2

3

4

5

6

7

8

9

10

Danceability

1

2

3

4

5

6

7

8

9

10

Valence

1

2

3

4

5

6

7

8

9

10

Acoustic

1

2

3

4

5

6

7

8

9

10

Popularity

1

2

3

4

5

6

7

8

9

10

Reset

Crea Playlist

Artista

Titolo

Nessun contenuto nella tabella

Benvenuto nella sezione Surprise Me! Seleziona il tuo genere preferito e, facoltativamente, il relativo artista. Dichiarare poi il grado di presenza richiesta attraverso un punteggio da 1 a 10 delle cinque caratteristiche principali:

- Energy: Più alto è il valore, più sarà energica la playlist.
- Danceability: Più alto è il valore, più facile sarà ballare.
- Valence: Più alto è il valore, più positivo sarà l'umore della playlist.
- Acoustic: Più alto è il valore, più la playlist comprenderà musica che usa principalmente strumenti acustici, anziché elettrici o elettronici.
- Popularity: Più alto è il valore, più popolari saranno i brani della playlist.

Seleziona infine il vincolo che preferisci tra un numero di canzoni o un minutaggio massimo e clicca "Crea playlist".

ATTENZIONE: Ai fini di ottenere un risultato in tempi ragionevoli, si sconsiglia di richiedere una playlist di durata massima di più di 3 ore (180 minuti).

Figure 7: Surprise Me tab

Surprise Me

My Mix

☐ Numero brani

▼

☐ Durata massima (min)

Genere

▼

Artista

▼

Brano

▼

Aggiungi

Reset

Inizializza Playlist

Rimuovi ultimo brano

Crea Playlist

Artista

Titolo

Nessun contenuto nella tabella

Benvenuto nella sezione MyMix!

Aggiungi all'interno della tabella fino a 5 brani del tuo genere preferito. Seleziona infine il vincolo che preferisci tra un numero di canzoni o un minutaggio massimo e clicca "Crea Playlist".

Figure 8: My Mix tab

11

## 5 Strutture dati e algoritmi principali

Nella realizzazione dell'applicazione, sono stati adottati sia il modello architetturale MVC (Model-View-Controller) che il design pattern DAO (Data Access Object).

Queste metodologie permettono di separare chiaramente le principali componenti di un software, ovvero la logica, l'accesso ai dati e l'interfaccia utente. A seguire, verranno analizzati in dettaglio.

### 5.1 Package `it.polito.PlaylistGen`

#### 5.1.1 FXMLController

Il Controller funge da mediatore tra la View e il Model, gestendo gli input utente, coordinando le azioni dell'applicazione e assicurandosi che l'interfaccia utente sia sincronizzata con lo stato interno dell'applicazione. Di seguito sono riportati i metodi principali:

- **Acquisizione del vincolo scelto dall'utente**

Il vincolo scelto avviene prima apprendendo in quale sezione si trovi l'utente:

---

```
// 0 = Surprise Me, 1 = My Mix
Integer nTab = tabPane.getSelectionModel().getSelectedIndex();
```

---

La definizione dei vincoli avviene attraverso le due variabili **isVincoloBrani** e **vincolo**. Il primo è *True* se l'utente vuole una playlist con  $n$  brani, *False* se, invece, desidera che la lunghezza della playlist sia limitata dai minuti totali di ogni traccia.

È riportato come esempio l'acquisizione dei vincoli nel caso  $nTab = 0$ .

---

```
if (nTab == 0) {
    if (nBraniSurprise.isSelected()) {
        Integer vincolo = cmbNBraniSurprise.getValue();
        if (vincolo != null) {
            errorLabelSurprise.setText(null);
            this.setVincolo(vincolo);
            this.setIsVincoloBrani(true);
            return true;
        } else {
            errorLabelSurprise.setText("Inserire numero di brani
                                     massimo richiesto e riprovare");
            return false;
        }
    } else if (minBraniSurprise.isSelected()) {
        String input = choiceDurataMaxSurprise.getText();
        try {
            Integer vincolo = Integer.parseInt(input);
            if (vincolo > 180) {
                errorLabelSurprise.setText("Inserire valore inferiore
                                           a 180 minuti e riprovare");
            }
        } catch (NumberFormatException e) {
            errorLabelSurprise.setText("Inserire un numero valido");
        }
    }
}
```

---

```

        return false;
    } else {
        errorLabelSurprise.setText(null);
        this.setVincolo(vincolo);
        this.setIsVincoloBrani(false);
        return true;
    }
} catch (Exception e) {
    errorLabelSurprise.setText("Inserire valore valido e
        riprovare");
    return false;
}
} else {
    else {

        ...
        // Segnalazione errori all'utente
        ...

    }
}
}
}

```

---

Il motivo per cui non si accetti un vincolo di minutaggio superiore a 180 minuti è puramente a fine prestazionale. Questa scelta sarà motivata nella sezione *Risultati ottenuti*.

- Acquisizione dei valori degli Slider (Sezione Surprise Me)

---

```

private void collectSliders() {

    Double energy = sliderEnergy.getValue();
    Double dance = sliderDance.getValue();
    Double valence = sliderValence.getValue();
    Double acoustic = sliderAcoustic.getValue();
    Double popularity = sliderPop.getValue();

    model.createPreference(energy, dance, valence, acoustic,
        popularity);

}

```

---

- Acquisizione dei brani inseriti nella TableView (Sezione My Mix)

---

```
@FXML
void addTrackMix(ActionEvent event) {

    // Acquisisco valori scelti
    String genre = cmbGenreMix.getValue();
    String artist = cmbArtistMix.getValue();
    String title = cmbTrackMix.getValue();

    if (playlistTableMix.getItems().size() < 5) {
        if (title != null) {

            Track track = model.getTrackbyInput(genre, artist, title);
            if (track != null) {

                // Resetto ErrorLabel
                errorLabelMix.setText(null);

                // Aggiungo traccia a tabella MyMix e tabella di
                // riferimento nel modello
                playlistTableMix.getItems().add(track);
                model.addTrackToMapId(track);

                // Blocco cambio del genere
                cmbGenreMix.setDisable(true);
                cmbGenreMix.setValue(genre);

                // Preparo prossima scelta
                this.resetChoiceMix(event);

            } else {

                ...
                // Segnalazione errori all'utente
                ...

            }

        }

    }

}
```

---

Il primo controllo impedisce all'utente di inserire più di cinque brani all'interno della tabella, i restanti invece assicurano che l'utente non abbia lasciato nessuna Combobox a valore nullo.

Una volta inserito il primo brano, l'unico limite è l'aggiunta di brani di genere diverso dall'appena selezionato.



- Avvio della ricerca Surprise Me

---

@FXML

```
void avviaRicercaSurprise(ActionEvent event) {

    if (this.findVincolo()) {
        this.disableVincoliSurprise();
        this.collectSliders();
        String genere = cmbGenreSurprise.getValue();
        String artist = cmbArtistSurprise.getValue();
        if (genere != null) {
            // Creo Grafo
            this.model.creaGrafoSurprise(genere, artist,
                model.getPreference());

            // Avvio ricerca
            List<Track> playlist = new ArrayList<>();
            playlist = model.calcolaPlaylist(artist,
                this.isVincoloBrani, this.vincolo, false);

            // Aggiungo output alla tabella
            this.playlistTableSurprise.getItems().clear();
            this.playlistTableSurprise.getItems().addAll(playlist);

            // Informazioni sulla playlist creata
            String durata = model.getDurataPlaylistSec(playlist);
            textAreaProgressSurprise
                .setText("Playlist Creata!\nNumero brani: " +
                    playlist.size() + ".\nDurata totale: " + durata);

        } else {
            errorLabelSurprise.setText("Selezionare un genere e
                riprovare");
        }
    }
}
```

---

Gli unici due controlli effettuati nel controller prima di avviare la ricerca sono l'inserimento di un vincolo tra i due disponibili e di un genere preferito su cui basare la creazione del grafo, poiché ogni vertice sarà un brano del genere scelto.

- Avvio della ricerca My Mix

---

@FXML

```
void avviaRicercaMix(ActionEvent event) {

    if (this.findVincolo()) {
        this.disableVincoliMix();
        if (!this.playlistTableMix.getItems().isEmpty()) {
            // Blocco opzione "Aggiungi" e "Reset"
            addTrack.setDisable(true);
            resetChoiceMix.setDisable(true);
            buttonremoveLastTrack.setDisable(true);

            // Creo Grafo
            model.creaGrafoMix(cmbGenreMix.getValue());

            // Avvio ricerca
            List<Track> playlist = new ArrayList<>();
            playlist = model.calcolaPlaylist(null, this.isVincoloBrani,
                this.vincolo, true);

            // Aggiungo output alla tabella
            this.playlistTableMix.getItems().clear();
            this.playlistTableMix.getItems().addAll(playlist);

            // Informazioni sulla playlist creata
            String durata = model.getDurataPlaylistSec(playlist);
            textAreaProgressMix
                .setText("Playlist Creata!\nNumero brani: " +
                    playlist.size() + ".\nDurata totale: " + durata);

        } else {

            ...
            // Segnalazione errori all'utente
            ...

        }
    }
}
```

---

Per la sezione "My Mix" invece il controllo sulla scelta del genere è sostituito dalla necessità che l'utente abbia scelto almeno un brano su cui calcolare i propri gusti musicali.

## 5.2 Package `it.polito.PlaylistGen.db`

Questo pacchetto agevola l'interfacciamento con la base di dati seguendo il design pattern DAO. Comprende due classi:

- `DBConnect.java`, che contiene i parametri necessari per l'accesso al database e stabilisce la connessione con il medesimo.
- `TracksDAO.java`, responsabile dell'estrazione dei dati dal database. Al suo interno, sono implementati tutti i metodi contenenti le query SQL utilizzate per interrogare la base di dati.

### 5.2.1 `DBConnect`

Il framework scelto per accedere database relazionale è HikaruCP<sup>4</sup>. Questo framework, sviluppato da Brett Wooldridge nel 2012, è basato sul concetto di connection pooling.

La connection pooling è un noto modello di accesso ai dati. Il suo scopo principale è ridurre il sovraccarico coinvolto nell'esecuzione delle connessioni al database e nelle operazioni di lettura/scrittura del database.

Le connessioni al database sono operazioni piuttosto costose e, pertanto, dovrebbero essere ridotte al minimo in ogni possibile caso d'uso (nei casi estremi, semplicemente evitate).

Il connection pooling implementa un contenitore di connessioni al database, che ci consente di riutilizzare un certo numero di connessioni esistenti, potendo così risparmiare efficacemente il costo di eseguire un grande numero di costose operazioni di accesso al database. Questo migliora le prestazioni complessive delle applicazioni basate su database.

Di seguito è riportato il codice per l'accesso al database:

---

```
import java.sql.Connection;
import java.sql.SQLException;

import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

public class DBConnect {

    private static final String jdbcURL = "jdbc:mariadb://localhost/Spotify";
    public static HikariDataSource ds;

    public static Connection getConnection() {

        if (ds == null) {
            HikariConfig config = new HikariConfig();
            config.setJdbcUrl(jdbcURL);
            config.setUsername("user");
```

---

<sup>4</sup><https://github.com/brettwooldridge/HikariCP>

```

        config.setPassword("password");

        // configurazione MySQL
        config.addDataSourceProperty("cachePrepStmts", "true");
        config.addDataSourceProperty("prepStmtCacheSize", "250");
        config.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");

        ds = new HikariDataSource(config);
    }

    try {
        return ds.getConnection();
    } catch (SQLException e) {
        System.err.println("Errore connessione al DB");
        throw new RuntimeException(e);
    }
}
}

```

---

L'username e la password per l'accesso al database SQL sono state sostituite per motivi di sicurezza.

### 5.2.2 TracksDAO

All'interno della classe `TracksDAO.java` troviamo i seguenti metodi per l'interrogazione del database, il metodo completo sarà riportato solo nell'ultimo esempio a scopo dimostrativo:

- Richiesta di tutti i generi presenti nel dataset

---

```

public List<String> listAllGenre(){

    String sql = "SELECT Tracks.'Top Genre', COUNT(Tracks.'Top
                  Genre') "
                + "FROM Tracks "
                + "GROUP BY Tracks.'Top Genre' "
                + "HAVING COUNT(Tracks.'Top Genre') >= 30";
    List<String> result = new ArrayList<String>();

    // ...
}

```

---

- Richiesta di tutti gli artisti associati ad uno specifico genere

---

```
public List<String> listAllArtistByGenre(String genre){

    String sql = "SELECT DISTINCT Tracks.'Artist'\n"
        + "FROM Tracks\n"
        + "WHERE Tracks.'Top Genre' = ? ";
    List<String> result = new ArrayList<String>();

    // ...

}
```

---

- Richiesta di tutti i brani associati ad uno specifico artista

---

```
public List<String> listAllTracksByArtist(String artist){

    String sql = "SELECT DISTINCT Tracks.'Title'\n"
        + "FROM Tracks\n"
        + "WHERE Tracks.'Artist' = ? ";
    List<String> result = new ArrayList<String>();

    // ...

}
```

---

- Richiesta di tutti i brani associati ad uno specifico genere.

---

```
public List<Track> listTracksByGenre(String genre) {

    String sql = "SELECT * \n"
        + "FROM Tracks\n"
        + "WHERE Tracks.'Top Genre' = ? ";
    List<Track> result = new ArrayList<Track>();

    // ...

}
```

---

- Richiesta dei valori corrispondenti di un brano associato ad uno specifico genere, artista e titolo.

---

```
public List<Track> listTracksByGenre(String genre) {

    String sql = "SELECT * \n"
        + "FROM Tracks\n"
        + "WHERE Tracks.'Top Genre' = ? ";
    List<Track> result = new ArrayList<Track>();

    try {
        Connection conn = DBConnect.getConnection();
        PreparedStatement st = conn.prepareStatement(sql);
        st.setString(1, genre);
        ResultSet res = st.executeQuery();

        while (res.next()) {

            Track track = new Track(res.getInt("Index"),
                res.getString("Title"), res.getString("Artist"), genre,
                res.getInt("Length (Duration)", res.getInt("Year"),
                res.getInt("Energy"), res.getInt("Danceability"),
                res.getInt("Valence"), res.getInt("Acousticness"),
                res.getInt("Popularity"));

            result.add(track);
        }
        conn.close();
        return result;

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }

}
```

---

## 5.3 Package `it.polito.PlaylistGen.model`

Nel pattern architetturale Model-View-Controller (MVC), il "Model" rappresenta la componente responsabile della gestione dei dati e della logica dell'applicazione.

Il ruolo principale del Model è separare la logica di presentazione (View) dalla logica di elaborazione (Model) e dal controllo delle interazioni dell'utente (Controller).

### 5.3.1 Model

Nella classe Model (`Model.java`), troviamo tutti i metodi di rilevanza algoritmica tra cui la ricorsione che verrà analizzata nel capitolo successivo. Gli algoritmi principali sono i seguenti:

- Calcolo e aggiunta degli archi del grafo

Poiché ogni vertice del grafo è rappresentato dalla classe `Vertex`, il vero problema computazionale è stato aggiungere gli archi che avessero un peso al di sotto di una soglia prestabilita, al fine di non creare un grafo completo e rendere la ricerca scarsa dal punto di vista prestazionale. Come soglia massima di peso è stato scelta la media del peso degli archi del grafo completo precedentemente creato (funzione `trovaAvg()`), al fine di eliminare circa metà degli archi e assicurarsi comunque una soluzione finale soddisfacente.

---

```
private void creaArchi() {  
  
    // Creo Archi  
    for (Vertex vi : this.grafo.vertexSet()) {  
        for (Vertex vj : this.grafo.vertexSet()) {  
            Integer viTrackId = vi.getTrack().getTrackId();  
            Integer vjTrackId = vj.getTrack().getTrackId();  
            if (viTrackId != vjTrackId) {  
                Double weight = Math.abs(vi.getScore() - vj.getScore());  
                Graphs.addEdgeWithVertices(this.grafo, vi, vj, weight);  
            }  
        }  
    }  
  
    // Al fine di ottimizzare la ricerca, elimino ogni arco di peso  
    // inferiore alla media degli archi  
    Integer avg = this.trovaAvg();  
    for (Vertex vi : this.grafo.vertexSet()) {  
        for (Vertex vj : this.grafo.vertexSet()) {  
            if (this.grafo.getEdge(vi, vj) != null) {  
                if (this.grafo.getEdgeWeight(this.grafo.getEdge(vi, vj))  
                    < avg) {  
                    this.grafo.removeEdge(vi, vj);  
                }  
            }  
        }  
    }  
}
```

---

- Calcolo dei gusti musicali usando i brani in input (Sezione "My Mix")

---

```
private Preference calcolaPreferenze() {

    // Energy
    Double sumEnergy = 0.0;
    // Danceability
    Double sumDanceability = 0.0;
    // Valence
    Double sumValence = 0.0;
    // Acoustic
    Double sumAcoustic = 0.0;
    // Popularity
    Double sumPopularity = 0.0;

    for (Track ti : this.tracksInput.values()) {
        sumEnergy += ti.getEnergy();
        sumDanceability += ti.getDanceability();
        sumValence += ti.getValence();
        sumAcoustic += ti.getAcousticness();
        sumPopularity += ti.getPopularity();
    }

    Double avgEnergy = sumEnergy / this.tracksInput.size();
    Double avgDanceability = sumDanceability / this.tracksInput.size();
    Double avgValence = sumValence / this.tracksInput.size();
    Double avgAcoustic = sumAcoustic / this.tracksInput.size();
    Double avgPopularity = sumPopularity / this.tracksInput.size();

    // Divido ciascun fattore per 10 per rendere i punteggi consistenti
    // con quelli della prima sezione (da 1 a 10)
    Double finalEnergy = (double) Math.round(avgEnergy / 10);
    Double finalDanceability = (double) Math.round(avgDanceability /
        10);
    Double finalValence = (double) Math.round(avgValence / 10);
    Double finalAcoustic = (double) Math.round(avgAcoustic / 10);
    Double finalPopularity = (double) Math.round(avgPopularity / 10);

    preference = new Preference(finalEnergy, finalDanceability,
        finalValence, finalAcoustic, finalPopularity);
    return preference;
}
```

---



### 5.3.2 Preference

La classe Preference (Preference.java) offre la possibilità di conservare le informazioni relative ai gusti dell'utente.

---

```
public class Preference {

    private Double energy;
    private Double danceability;
    private Double valence;
    private Double acousticness;
    private Double popularity;

    public Preference(Double energy, Double danceability, Double valence,
        Double acousticness, Double popularity) {
        super();
        this.energy = energy;
        // ...
        this.popularity = popularity;
    }
}
```

---

### 5.3.3 Track

La classe Track (Track.java) è responsabile per definire le istanze delle canzoni e permette di memorizzare all'interno del programma i dati estratti dal dataset.

---

```
public class Track {

    private Integer trackId;
    private String title;
    private String artist;
    private String genre;
    private Integer length;
    private Integer year;
    private Integer energy;
    private Integer danceability;
    private Integer valence;
    private Integer acousticness;
    private Integer popularity;

    public Track(Integer trackId, String title, String artist, String genre,
        Integer length, Integer year, Integer energy, Integer danceability,
        Integer valence, Integer acousticness, Integer popularity) {
        super();
        this.trackId = trackId;
        // ...
        this.popularity = popularity;
    }
}
```

---

### 5.3.4 Vertex

La classe Vertex (`Vertex.java`), invece, racchiude le due informazioni sopracitate in un'unica classe, rappresentando il singolo vertice del grafo su cui verrà effettuata la ricorsione.

---

```
public class Vertex {  
  
    private Track track;  
    private Double score;  
  
    public Vertex(Track track, Double score) {  
        super();  
        this.track = track;  
        this.score = score;  
    }  
}
```

---

## 6 Ricorsione

L'approccio per l'ottenimento della playlist ottima si basa su una ricerca esaustiva (backtracking) per esplorare tutte le possibili combinazioni di brani e costruire la playlist migliore rispettando i vincoli dati. Questo tipo di approccio, pur potendo risultare dispendioso in termini di memoria e complessità temporale, è giustificato dal vincolo di un massimo di 40 brani totali (circa 2 ore e 30 minuti). L'inserimento di questo vincolo permette di ottenere un risultato in tempi sotto il secondo e confida nel fatto che difficilmente un utente possa ascoltare musica per più del tempo sopracitato senza cambiare manualmente brano o genere.

Il metodo principale `CalcolaPlaylist` inizia inizializzando e resettando le variabili di stato dell'algoritmo, ottiene la lista di brani validi (`canzoniValide`) e una lista parziale di brani (`parziale`) inizializzata in base alle preferenze dell'utente (Mix o SurpriseMe) ed effettua infine la chiamata al metodo ricorsivo `calcola` per trovare la migliore playlist possibile.

---

```
public List<Track> calcolaPlaylist(String artist, Boolean isVincoloBrani,
    Integer vincolo, Boolean isMix) {

    this.resetSearch();

    List<Vertex> canzoniValide = new ArrayList<Vertex>();
    ConnectivityInspector<Vertex, DefaultWeightedEdge> ci = new
        ConnectivityInspector<>(this.grafo);
    List<Vertex> parziale = new ArrayList<>();

    // Controllo se i vincoli richiesti sono raggiungibili dal grafo
    if (isVincoloBrani && this.getNVertici() < vincolo) {
        vincolo = this.getNVertici();
    } else if (!isVincoloBrani && this.getTotalDurataGraph() < vincolo) {
        vincolo = this.getTotalDurataGraph();
    }

    // Sezione Mix, aggiungo vertici corrispondenti ai brani inseriti alla
    // soluzione parziale
    if (isMix) {
        for (Track ti : this.tracksInput.values()) {
            for (Vertex vi : this.getVertici(artist, preference)) {
                if (ti.equals(vi.getTrack())) {
                    parziale.add(vi);
                }
            }
        }
        // Comincio ricerca a partire dall'ultimo brano inserito
        canzoniValide.addAll(ci.connectedSetOf(parziale.get(parziale.size()
            - 1)));
    } else { // Sezione SurpriseMe, aggiungo miglior inizio possibile
        Vertex start = findBestStart(artist);
```

```

    parziale.add(start);

    // Trovo componente connessa di "start"
    canzoniValide.addAll(ci.connectedSetOf(start));

}

calcola(parziale, canzoniValide, isVincoloBrani, vincolo);

for (Vertex vi : this.bestPlaylist) {
    this.finalPlaylist.add(vi.getTrack());
}
return this.finalPlaylist;

```

---

Il metodo ricorsivo `calcola` prende come input una lista `parziale` che rappresenta una possibile playlist parziale, una lista di `canzoniValide` che sono i brani disponibili per essere aggiunti alla playlist, un flag `isVincoloBrani` che indica se esiste un vincolo sul numero di brani o sulla durata totale della playlist, e un parametro `vincolo` che rappresenta il valore massimo del vincolo. Il metodo segue i seguenti step:

1. Itera attraverso la lista di `canzoniValide` e cerca di aggiungere ciascun brano (`Vertex vi`) alla playlist parziale, rispettando eventuali vincoli imposti.
2. Aggiorna ricorsivamente la playlist e calcola la somma degli archi pesati di questa playlist parziale.
3. Se la somma degli archi è inferiore al miglior punteggio (`bestScore`), continua la ricorsione. Altrimenti, verifica se la playlist parziale attuale è migliore di quella salvata come migliore fino a quel momento (`bestPlaylist`). Se è così, aggiorna la migliore playlist e il miglior punteggio.

---

```

private void calcola(List<Vertex> parziale, List<Vertex>
    canzoniValide, Boolean isVincoloBrani, Integer vincolo) {

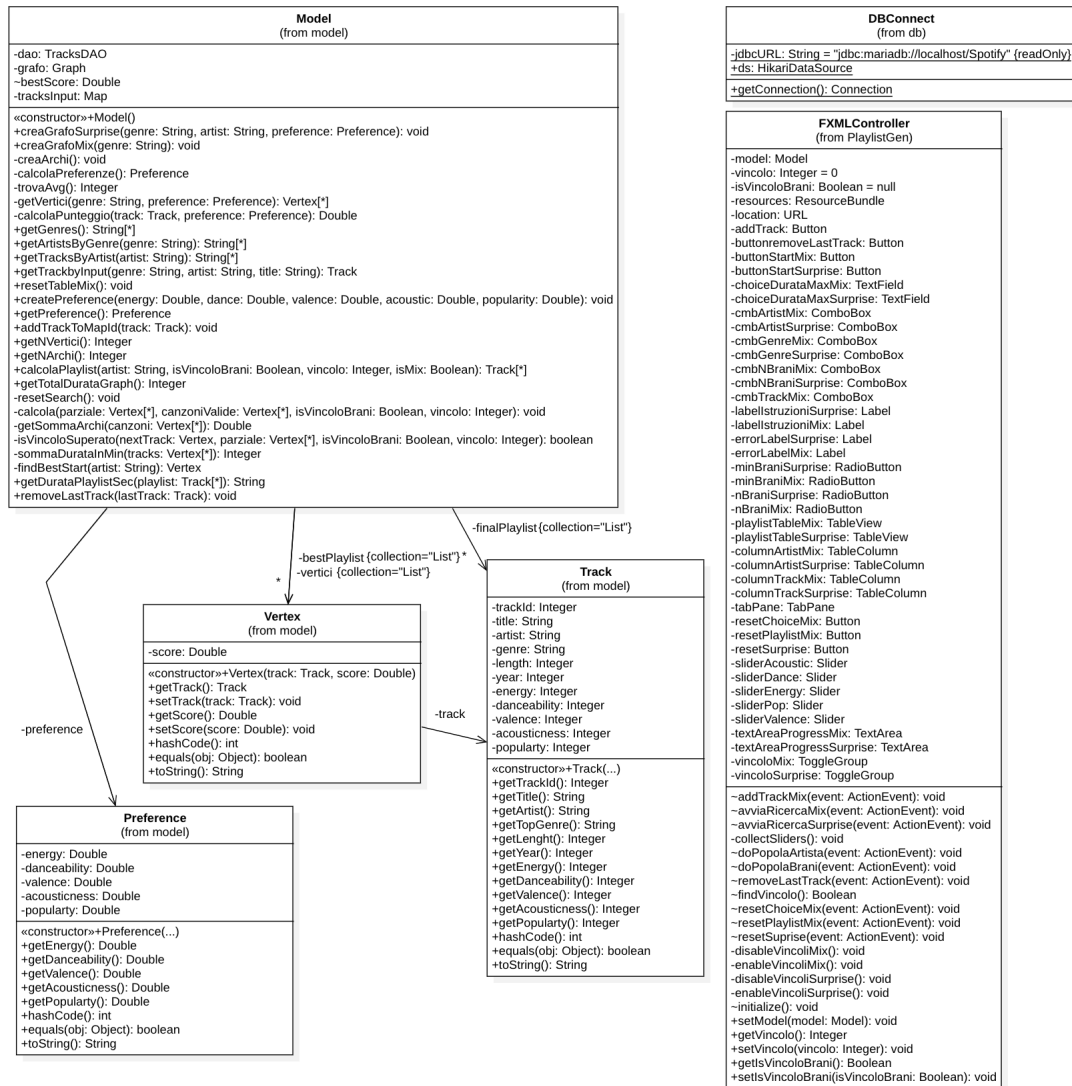
    for (Vertex vi : canzoniValide) {
        if (!parziale.contains(vi)) {
            if (!this.isVincoloSuperato(vi, parziale, isVincoloBrani,
                vincolo)) {
                parziale.add(vi);
                Double sommaArchiProvvisoria =
                    this.getSommaArchi(parziale);
                if (sommaArchiProvvisoria < bestScore) {
                    this.calcola(parziale, canzoniValide, isVincoloBrani,
                        vincolo);
                }
                parziale.remove(parziale.size() - 1);
            } else {
                // Controllo soluzione migliore La soluzione migliore
                // corrisponde al cammino con somma dei pesi degli archi
                // piu' bassa
            }
        }
    }
}

```

```
        Double sommaArchiProvvisoria =  
            this.getSommaArchi(parziale);  
        if (sommaArchiProvvisoria < bestScore) {  
            this.bestPlaylist = new ArrayList<>(parziale);  
            this.bestScore = sommaArchiProvvisoria;  
        }  
    }  
}  
}
```

---

## 7 Diagramma delle classi



## 8 Risultati ottenuti e considerazioni finali

Il prodotto finale risulta particolarmente performante anche lavorando con generi contenenti numerosi brani, permettendo di produrre il risultato in tempistiche sempre al di sotto del secondo fino a un massimo vincolo di 40 canzoni (in media poco meno di tre ore di minuti totali).

Nella figura sottostante è possibile notare l'andamento delle performance all'aumentare del numero di canzoni richiesto nella playlist della Tab Surprise Me. In questo esempio, a fine puramente esemplificativo, gli Slider Energia (E), Ballabilità (D), Positività (V), Acustica (A) e Popolarità (P) sono stati posizionati rispettivamente a 2, 4, 6, 8 e 10.

I generi presi in considerazione nell'esempio sono i tre maggiormente popolati di brani, al fine di visionare l'andamento nella peggiore delle situazioni computazionali. I generi in questione sono Rock (164 brani), Pop (156 brani) ed European Pop (150 brani).

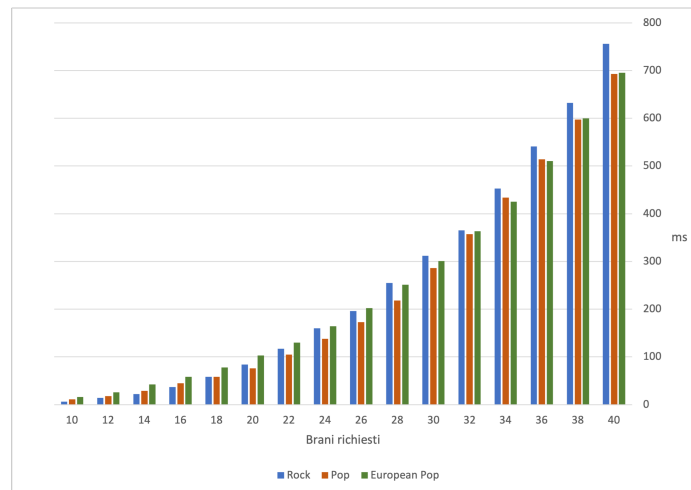


Figure 9: Tempo di completamento all'aumentare del numero di brani richiesti

È possibile ora concludere indicando quali siano le criticità di questo progetto e possibili implementazioni future:

- Scalabilità: andando ad aumentare il volume di dati musicali, l'algoritmo deve essere in grado di gestire la crescita senza perdere in efficienza, cosa che non può essere garantita con il tipo di ricorsione esaustiva implementato. Nel futuro sarebbe il caso di sostituirlo con un algoritmo più pratico e accettabile in grado di trovare soluzioni approssimate ma che richiedono meno risorse.
- Valutazione oggettiva della playlist: definire metriche oggettive per valutare la qualità di una playlist può essere complicato. L'algoritmo dovrebbe cercare di fornire un risultato che sia gradevole ed apprezzato dagli utenti.
- Requisiti dell'utente e gusti musicali variabili: gli utenti hanno gusti musicali molto variabili e soggettivi. Un algoritmo potrebbe avere difficoltà a soddisfare le preferenze individuali di ciascun utente, specialmente se sono mutevoli nel tempo. Una possibilità futura potrebbe essere quella di implementare nuovi vincoli più specifici e meglio rappresentativi della richiesta dell'utente.

Il progetto completo è disponibile al seguente indirizzo:

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

Al seguente link è disponibile una video dimostrazione del progetto:

<https://youtu.be/eR-s5fUmcfQ>



## Licenza d'uso

Sviluppo di un'applicazione software per la generazione automatica di playlist  
attraverso la ricerca ricorsiva e grafi pesati © 2023

by Carlo Marra

is licensed under CC BY-NC-SA 4.0