

POLITECNICO DI TORINO

DIPARTIMENTO DI INGEGNERIA GESTIONALE E
DELLA PRODUZIONE

Corso di Laurea in Ingegneria Gestionale
Classe L8 – Ingegneria dell'Informazione



Sviluppo di un tool per organizzatore di festival di musica dal vivo

Relatore

Prof. Fulvio Corno

Candidato

Matteo Rizzi
244853

A.A. 2019/2020

SOMMARIO

1. Proposta di progetto.....	2
1.1. Studente proponente	2
1.2. Titolo della proposta.....	2
1.3. Descrizione del problema proposto	2
1.4. Descrizione della rilevanza gestionale del problema.....	2
1.5. Descrizione dei data-set per la valutazione	2
1.6. Descrizione preliminare degli algoritmi coinvolti.....	3
1.5. Descrizione preliminare delle funzionalità previste per l'applicazione software	3
2. Descrizione dettagliata del problema affrontato	5
3. Descrizione del data-set utilizzato.....	6
4. Strutture dati e algoritmi.....	8
4.1. Descrizione delle strutture dati utilizzate	8
4.2. Descrizione degli algoritmi utilizzati.....	9
5. Diagramma delle classi principali.....	14
6. Videate dell'applicazione realizzata e link al video dimostrativo del software	15
6.1. Videate dell'applicazione realizzata con risultati sperimentali ottenuti	15
6.2. Link al video dimostrativo del software	18
7. Valutazioni sui risultati ottenuti e conclusioni	19

1. Proposta di progetto

1.1. Studente proponente

s244853 Rizzi Matteo

1.2. Titolo della proposta

Tool per organizzatore di festival di musica dal vivo

1.3. Descrizione del problema proposto

L'applicazione si propone l'obiettivo di aiutare un organizzatore di festival di musica dal vivo nella ricerca di artisti da ingaggiare per il proprio festival. L'utente può cercare da sé gli eventuali artisti, oppure può lasciare il compito all'applicazione, che deve trovare l'insieme ottimale di artisti in base al budget dell'organizzatore.

1.4. Descrizione della rilevanza gestionale del problema

Una buona parte di concerti di musica dal vivo è composta dai festival musicali, in cui più cantanti/band hanno la possibilità di suonare nell'arco della stessa giornata e ciò fa sorgere il problema della scelta di quali artisti ingaggiare. L'applicazione quindi intende aiutare nella ricerca di tali artisti e nell'ottimizzazione della gestione del budget a disposizione dell'organizzatore, in modo da massimizzare il numero di presenze al festival.

1.5. Descrizione dei data-set per la valutazione

I dati sono racchiusi in un'unica tabella che ha come nome *artisti*. La tabella in questione presenta innanzitutto i dati riguardanti i cantanti e le band in attività, dati relativi ad uno degli ultimi tour dell'artista in questione (racchiusi nei campi- `biglietti_venduti`, `numero_di_show`, `numero_medio_biglietti_venduti`), i quali sono estrapolati dai siti <https://en.wikipedia.org/> e <https://touringdata.wordpress.com/>. Inoltre è presente un campo riguardante il cachet medio percepito, le cui fonti sono <http://www.eventresourcespresents.com/> e <https://www.celebritytalent.net/> e un campo contenente il genere musicale di appartenenza e il nome della band o del cantante in questione. Vi è, infine, un campo riguardante il numero di ascolti medi sulla piattaforma streaming on demand Spotify nell'ultimo mese.

1.6. Descrizione preliminare degli algoritmi coinvolti

Il programma presenta due funzioni principali, la prima implementata tramite un algoritmo di ricerca, la seconda mediante un algoritmo di ricorsione:

1. Per quanto riguarda la parte di ricerca, dove è fondamentale l'utilizzo del Pattern DAO, è possibile filtrare i vari artisti in diversi modi (si possono visualizzare tutti gli artisti, un artista in particolare, oppure tutti gli artisti facenti parte di un dato genere musicale, o che abbiano venduto almeno un certo numero medio di biglietti, ecc.). Da questa interfaccia l'utente può selezionare uno o più artisti e aggiungerli direttamente nella soluzione ottima che viene trovata tramite l'algoritmo ricorsivo descritto nel punto successivo (questo perché magari l'organizzatore ha già avuto la disponibilità degli artisti selezionati, che quindi faranno sicuramente parte del festival).
2. L'algoritmo ricorsivo, impostato un budget massimo e tenuto conto degli artisti già aggiunti (se la somma del cachet percepito da quelli già inseriti dovesse superare l'importo del budget verrà notificato l'errore e l'utente dovrà modificare la lista degli artisti inseriti oppure il budget), trova l'insieme di artisti che rientra nel budget inserito e che garantirebbe il maggior pubblico possibile al festival. A scelta dell'utente, la massimizzazione del pubblico può essere fatta sulla base della media dei biglietti venduti in uno degli ultimi tour dell'artista, oppure sulla base degli streaming totalizzati dallo stesso su Spotify nell'ultimo mese. E' inoltre possibile specificare altri vincoli, tra cui il numero di artisti da ingaggiare, e selezionare quali generi musicali considerare. Rispetto a quest'ultimo vincolo è inoltre possibile identificare dei generi, tra quelli selezionati, "privilegiati": gli artisti che rientrano in tale genere musicale, al momento dell'avvio della ricorsione, hanno un peso maggiore in termini di biglietti venduti/streaming. Il peso da dare ai generi "privilegiati" può essere impostato attraverso uno slider (con valori che vanno da 1, ovvero nessun privilegio, a 2, ovvero con peso doppio rispetto agli altri).

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

L'applicazione ha tre interfacce grafiche. La prima interfaccia grafica è quella di *Homepage*, in cui attraverso due bottoni l'utente potrà accedere alla parte di ricerca o a quella di ricorsione. La seconda interfaccia è quella in cui l'utente, tramite appositi nodi, può

consultare il database secondo i tipi di filtraggio proposti (per genere musicale, per numero medio di biglietti venduti, ecc.). Da qui si possono inoltre aggiungere gli artisti che faranno parte della soluzione dell'algoritmo ricorsivo e passare all'interfaccia di ricorsione tramite apposito bottone. Nella terza interfaccia l'utente può gestire tutti i vincoli della parte ricorsiva e, una volta eseguito l'algoritmo ricorsivo, può visualizzare l'insieme ottimale di artisti e il budget rimanente.

2. Descrizione dettagliata del problema affrontato

L'applicazione sviluppata mira ad aiutare un organizzatore di festival di musica dal vivo nella ricerca e nell'ottimizzazione degli artisti da ingaggiare tramite un algoritmo di ricerca e un algoritmo ricorsivo.

Nell'ambito dei concerti di musica dal vivo sono molto frequenti i *festival*, in cui in una o più giornate cantanti o band si susseguono sul palco per eseguire i propri brani. Man mano che la fama dell'artista aumenta, cresce di conseguenza il cachet percepito dallo stesso per l'ingaggio, per cui i costi di questi eventi (tralasciando altri notevoli spese come scelta della location, la costruzione del palco, o tutti gli altri costi di gestione) possono arrivare a svariati MMEuro, o svariati MM\$, come per gli artisti presenti nel database.

Per questo motivo l'obiettivo dell'applicazione è quello di ottimizzare il budget a disposizione dell'organizzatore, inserito in input dall'utente, trovando la combinazione ottima degli artisti che garantirebbe la maggiore presenza di pubblico al festival e, dunque, maggiori ricavi. Ciò è reso possibile considerando l'affluenza media ad uno degli ultimi tour dell'artista oppure attraverso il numero di ascoltatori dell'ultimo mese dello stesso sulla piattaforma di streaming Spotify.


In particolare il problema affrontato è quello del *problema dello zaino (knapsack problem)*, in cui lo “zaino” è da riempire con gli artisti che sono ottimizzati tramite i parametri sopra citati, rispettando il budget prefissato e alcuni vincoli (è possibile selezionare solo alcuni generi musicali e, tra questi, dei generi *privilegiati* che avranno un peso maggiore nella ricerca della combinazione ottima, oppure è possibile limitare il numero di artisti tramite un valore inserito da input) che tendono a rendere flessibile la ricerca e ad adattarsi alle esigenze dell'utente. L'utente inoltre può aggiungere un artista nella combinazione ottima tramite l'interfaccia in cui è implementato l'algoritmo di ricerca, usando l'apposito bottone, dopo aver selezionato un cantante/band a seguito dell'avvio della ricerca. Questa funzionalità è stata implementata in quanto l'organizzatore può aver avuto dei contatti precedenti con artisti che quindi faranno sicuramente parte della *line-up* dell'evento.

L'applicazione di cui alla presente relazione è utile per dare una idea all'organizzatore di quella che può essere la combinazione degli artisti da ingaggiare; tuttavia lo scopo del programma non è quello di fornire una soluzione esatta: non sono infatti presi in considerazione alcuni parametri importanti, come il luogo dell'evento, che sicuramente influiscono sulla possibile affluenza di pubblico (ad esempio gli artisti con maggiore seguito in Italia sono diversi da quelli ascoltati in Spagna).

3. Descrizione del data-set utilizzato

Il data-set utilizzato per l'applicazione è stato costruito da zero attingendo da varie fonti sul web (i dati sui biglietti venduti nei vari tour sono estrapolati dai siti <https://en.wikipedia.org/> e <https://touringdata.wordpress.com/>, mentre i dati sul cachet medio percepito dagli artisti provengono da <http://www.eventresourcespresents.com/> e <https://www.celebritytalent.net/>).

La tabella che contiene tutte le informazioni dei cantanti e delle band è la tabella *artisti* che è l'unica presente nel database *festivaldimusica*. Gli artisti presi in considerazione sono artisti di fama internazionale, i cui dati necessari per l'applicazione sono stati di più facile reperibilità. Alcune informazioni, inoltre, risultano essere solo verosimili e non del tutto esatte (ad esempio il cachet percepito da un artista dipende da vari fattori e non è stato possibile determinarlo con certezza a priori).

#	Nome	Tipo di dati
 1	id	INT
2	nome	VARCHAR
3	genere	VARCHAR
4	biglietti_venduti	INT
5	numero_di_show	INT
6	numero_medio_biglietti_venduti	DOUBLE
7	ascolti_Spotify_ultimo_mese	INT
8	cachet_medio	INT

La tabella *artisti* è composta da 8 campi, la cui chiave primaria è il campo *id*.

- **id**: valore numerico intero incrementale. Il valore iniziale è pari a 1.
- **nome**: stringa che indica il nome dell'artista, che sia un cantante o una band.
- **genere**: stringa il cui valore è il genere musicale di appartenenza di tale artista.
- **biglietti_venduti**: valore intero numerico, indica il numero totale di biglietti venduti dall'artista in uno dei suoi ultimi tour.
- **numero_di_show**: valore intero numerico, è il numero di show realizzati nel tour considerato nel campo precedente.
- **numero_medio_biglietti_venduti**: è un valore numerico decimale riguardante la media di biglietti venduti nel tour preso in considerazione dell'artista. E' il rapporto tra i due campi precedentemente descritti.
- **ascolti_Spotify_ultimo_mese**: è un campo numerico intero che indica il valore totale di ascolti dell'artista sulla piattaforma streaming Spotify nell'ultimo mese.

- ***cachet_medio***: valore numerico intero che indica il cachet percepito dall'artista. E' un valore medio in quanto, per alcuni artisti, il valore è stato estrapolato da due fonti diverse, in cui i cachet presentati sono risultati diversi.

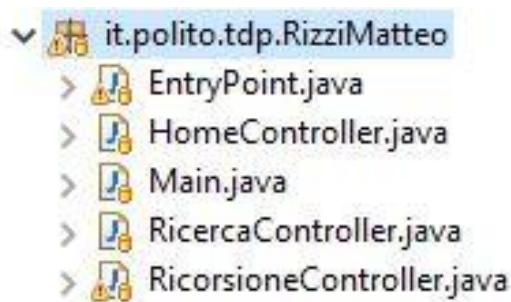
4. Strutture dati e algoritmi

4.1. Descrizione delle strutture dati utilizzate

L'applicazione è stata scritta in linguaggio Java, con l'utilizzo delle interfacce JavaFX. Sono stati inoltre implementati il pattern MVC (Model-View-Controller) e il pattern DAO (Data Access Object), il primo per separare la componente applicativa del programma, delegata al *Model*, dalla parte di interazione con l'utente, il cui compito spetta al *Controller*, il secondo per gestire separatamente l'accesso e l'interazione con il database.

Per quanto riguarda la struttura dati dell'applicazione, sono presenti 3 *package* di seguito descritti:

- Il package *it.polito.tdp.RizziMatteo* contiene tutte le classi che si occupano della parte di interazione con l'utente, dalle due classi che servono per il lancio del programma *Main* e *EntryPoint* alle tre classi che implementano l'interazione con le tre interfacce grafiche presenti *HomeController*, *RicercaController* e *RicorsioneController*.



- Il package *it.polito.tdp.RizziMatteo.model* contiene le classi che si occupano della logica applicativa. La classe Java Bean *Artista* serve per poter salvare nelle apposite strutture dati le informazioni sugli artisti; la classe *ArtistaPesato* contiene le informazioni sull'artista e sul peso che questo ha durante la ricorsione in base al genere di appartenenza; la classe *Model* contiene l'effettiva logica applicativa del programma; la classe *TestModel* è dotata di *main* per poter testare gli algoritmi.



- Il package *it.polito.tdp.RizziMatteo.db* si occupa della connessione e dell'interazione col database *festivaldimusica*. In particolare le classi presenti sono *DBConnect*, che si occupa della connessione col database e *FestivalDiMusicaDAO*, che si occupa dell'interazione col database. E' presente inoltre una classe di test dotata di *main*, la classe *TestDAO*.



4.2. Descrizione degli algoritmi utilizzati

Il package *it.polito.tdp.RizziMatteo.db* contiene le già citati classi *TestDAO* e *DBConnect*, attraverso cui è possibile effettuare l'accesso al database attraverso la libreria *HikariCP*. La classe più interessante, però, è *FestivalDiMusicaDAO*, che contiene una serie di metodi utili per l'interazione con il database. A scopo di esempio viene di seguito mostrato il codice del metodo `listAllArtists()` che ritorna la lista di tutti gli oggetti di tipo *Artista* presenti nel database:

```
public List<Artista> listAllArtists() {
    String sql = "SELECT * FROM artisti";
    List<Artista> list = new ArrayList<>();
    Connection conn = DBConnect.getConnection();

    try {
        PreparedStatement st = conn.prepareStatement(sql);
        ResultSet res = st.executeQuery();
        while (res.next()) {

            Artista artista = new Artista(res.getInt("id"), res.getString("nome"),
                res.getString("genere"),
                res.getLong("biglietti_venduti"),
                res.getInt("numero_di_show"),
                res.getDouble("numero_medio_biglietti_venduti"),
                res.getInt("ascolti_Spotify_ultimo_mese"),
                res.getInt("cachet_medio"));

            list.add(artista);
        }
        conn.close();
        return list;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}
```

Il canovaccio presente in questo metodo si ripete per tutti gli altri metodi della classe: viene preparata la stringa SQL e la struttura dati che conterrà i dati, viene aperta la connessione col database e si prepara un blocco *try-catch* per prevenire lo scatenamento di eventuali eccezioni. All'interno del blocco viene associata la query alla connessione, si ottiene il risultato della query e la si salva nella struttura dati, che infine viene ritornata tramite il comando **return**.

La logica applicativa, come già detto, è implementata nel *Model*. Qui vi sono vari metodi che servono per interfacciarsi con il package in cui è presente la classe *FestivalDiMusicaDAO*, che sono utilizzati nella classe *RicercaController* per implementare la parte di ricerca. Per quanto riguarda la parte di ricerca vi è anche il metodo `intersezioneArtisti(List<Artista> artisti, List<Artista> intersezione)`, utilizzato nella ricerca combinata per ritornare solamente gli artisti che rispettano tutte le condizioni specificate.

```
public List<Artista> intersezioneArtisti(List<Artista> artisti, List<Artista> intersezione) {
    List<Artista> risultato = new ArrayList<>();
    for (Artista a1 : artisti) {
        for (Artista a2 : intersezione) {
            if (a1.equals(a2)) {
                risultato.add(a1);
            }
        }
    }
    return risultato;
}
```

Per quanto riguarda la parte ricorsiva, il metodo `calcolaCombinazioneMigliore(Integer budgetMassimo, Integer numeroArtisti, List<String> generiSelezionati, List<String> generiPrivilegiati, Double fattoreCorrettivo, char tipo)` è quello che fa partire la ricorsione, tramite il bottone *Avvia ricerca della combinazione migliore* dall'apposita interfaccia. Il metodo riceve da input vari parametri che costituiscono i vincoli della ricorsione, tra cui il budget massimo, il numero di artisti da chiamare, i generi privilegiati e il tipo di ricorsione (ovvero se l'ottimizzazione deve essere fatta in base al numero medio di biglietti venduti o in base agli ascolti su Spotify nell'ultimo mese). Vi è inoltre l'inizializzazione del *best*, ovvero della combinazione migliore che ritorna il metodo, ma non della *soluzione parziale*, in quanto questa viene inizializzata nel costruttore del *Model*, per poter dare la possibilità di aggiungere gli artisti alla soluzione ottima tramite l'interfaccia di ricerca.

```

public List<Artista> calcolaCombinazioneMigliore(Integer budgetMassimo,
        Integer numeroArtisti, List<String> generiSelezionati,
        List<String> generiPrivilegiati, Double fattoreCorrettivo, char tipo) {

    this.best = new ArrayList<>();
    List<ArtistaPesato> artistiConsentiti = this.getArtistiPesati(generiPrivilegiati,
        generiSelezionati, fattoreCorrettivo, budgetMassimo, tipo);
    Collections.sort(artistiConsentiti);

    // System.out.println("Size artisti consentiti: " + artistiConsentiti.size());
    List<Artista> parziale = new ArrayList<>(this.parziale);

    this.ricorsione(budgetMassimo, parziale, artistiConsentiti, numeroArtisti, 0);

    return best;
}

```

In questo metodo, inoltre, viene chiamato un altro metodo, `getArtistiPesati(List<String> generiPrivilegiati, List<String> generiSelezionati, Double fattoreCorrettivo, Integer budgetMassimo, char tipo)`, che ritorna una lista di *ArtistaPesato*, la quale verrà considerata nella ricorsione vera e propria, e che ha una doppia funzione: quella di filtrare gli artisti che hanno un cachet inferiore al budget massimo e che appartengano ad un genere musicale tra quelli selezionati in input e quella di assegnare un peso in base al genere musicale di appartenenza dell'artista. Se l'artista in questione appartiene ad un genere musicale presente nella lista dei *generiPrivilegiati* a questo viene applicato un peso maggiore a seconda del fattore correttivo specificato in input. Se la lista dei *generiPrivilegiati* è vuota il peso è lo stesso per tutti gli artisti.

```

private List<ArtistaPesato> getArtistiPesati(List<String> generiPrivilegiati, List<String> generiSelezionati,
        Double fattoreCorrettivo, Integer budgetMassimo, char tipo) {
    List<ArtistaPesato> artistiConsentiti = new ArrayList<>();

    if (generiPrivilegiati.size() == 0) {
        fattoreCorrettivo = 1.0;
    }

    Double pesoMinore = 1 / (fattoreCorrettivo * generiPrivilegiati.size()
        + (generiSelezionati.size() - generiPrivilegiati.size()));
    Double pesoMaggiore = fattoreCorrettivo * pesoMinore;

    for (Artista artista : this.dao.listAllArtists()) {
        if (artista.getCachetMedio() <= budgetMassimo) {
            if (generiPrivilegiati.size() != 0 && generiPrivilegiati.contains(artista.getGenere())) {
                // 'B' sta per numero medio di biglietti venduti, 'S' sta per ascolti su Spotify
                // nell'ultimo mese
            }
        }
    }
}

```

```

        if (tipo == 'B') {
            ArtistaPesato a = new ArtistaPesato(artista,
                artista.getNumeroMedioBigliettiVenduti() * pesoMaggiore);
            artistiConsentiti.add(a);
        } else if (tipo == 'S') {
            ArtistaPesato a = new ArtistaPesato(artista,
                artista.getAscoltiSpotifyUltimoMese() * pesoMaggiore);
            artistiConsentiti.add(a);
        }
    } else if (generiSelezionati.size() != 0 && generiSelezionati.contains(artista.getGenere())) {
        if (tipo == 'B') {
            ArtistaPesato a = new ArtistaPesato(artista,
                artista.getNumeroMedioBigliettiVenduti() * pesoMinore);
            artistiConsentiti.add(a);
        } else if (tipo == 'S') {
            ArtistaPesato a = new ArtistaPesato(artista,
                artista.getAscoltiSpotifyUltimoMese() * pesoMinore);
            artistiConsentiti.add(a);
        }
    }
}

return artistiConsentiti;
}

```

Il metodo in cui viene effettuata la ricorsione vera e propria è il metodo `ricorsione(Integer budgetMassimo, List<Artista> parziale, List<ArtistaPesato> artistiConsentiti, Integer numeroArtisti, Integer L)`, il quale riceve in input il budget massimo, la lista parziale riempita con gli artisti eventualmente aggiunti nella parte di ricerca, gli artisti consentiti trovati col metodo precedente, l'eventuale vincolo sul numero di artisti e il livello L della ricorsione con valore iniziale uguale a 0. All'interno del metodo nei casi intermedi la ricorsione viene chiamata prima provando ad aggiungere l'artista alla soluzione parziale, poi provando a non aggiungerlo. I casi terminali, invece, sono di due tipi e la scelta di uno o dell'altro dipende dal parametro *numeroArtisti*, a seconda che il suo valore sia pari a *null* o meno.

```

private void ricorsione(Integer budgetMassimo, List<Artista> parziale, List<ArtistaPesato> artistiConsentiti,
    Integer numeroArtisti, Integer L) {
    if (this.spesaAggiunti > budgetMassimo)
        return;
    if (numeroArtisti == null) {
        if (sommaPesi(parziale, artistiConsentiti) > sommaPesi(best, artistiConsentiti)) {
            best = new ArrayList<>(parziale);
        }
    } else {

```

```

    if (parziale.size() > numeroArtisti) {
        return;
    }
    if (parziale.size() == numeroArtisti) {

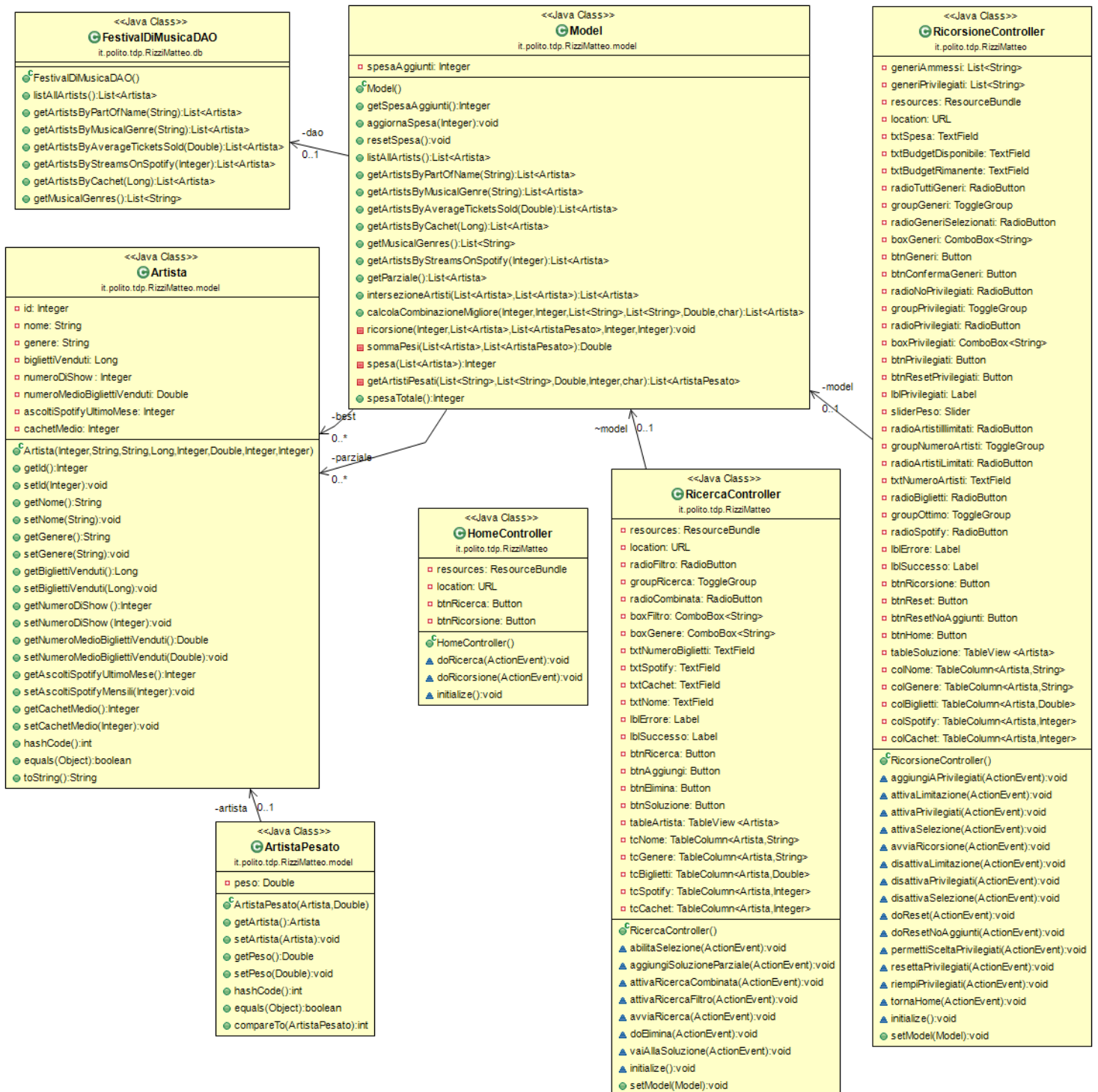
        if (sommaPesi(parziale, artistiConsentiti) > sommaPesi(best, artistiConsentiti)) {
            best = new ArrayList<>(parziale);
            return;
        }
    }
    // sicuramente parziale.size() < numeroArtisti
    if (spesa(parziale) > budgetMassimo)
        return;
}
if (L == artistiConsentiti.size())
    return;

Integer spesaIpotetica = spesa(parziale) + artistiConsentiti.get(L).getArtista().getCachetMedio();
if (spesaIpotetica <= budgetMassimo) {
    // provo ad aggiungerlo
    parziale.add(artistiConsentiti.get(L).getArtista());
    this.ricorsione(budgetMassimo, parziale, artistiConsentiti, numeroArtisti, L + 1);
    parziale.remove(artistiConsentiti.get(L).getArtista());
}
// provo a non aggiungerlo
this.ricorsione(budgetMassimo, parziale, artistiConsentiti, numeroArtisti, L + 1);
}

```

Sono presenti inoltre ulteriori semplici metodi per il calcolo della somma dei pesi, `sommaPesi(List<Artista> artisti, List<ArtistaPesato> artistiConsentiti)`, oppure metodi per sommare, resettare o ritornare la spesa effettuata.

5. Diagramma delle classi principali



6. Videate dell'applicazione realizzata e link al video dimostrativo del software

6.1. Videate dell'applicazione realizzata con risultati sperimentali ottenuti

La prima interfaccia con cui ci si relaziona è la schermata di *Homepage*, dalla quale si può accedere alla parte di ricerca, tramite il bottone *Inizia la ricerca*, oppure direttamente alla parte ricorsiva, tramite il bottone *Trova la combinazione migliore*.



Per quanto riguarda l'interfaccia *Ricerca degli artisti*, il programma offre due modi di effettuare una ricerca: la prima è una *ricerca tramite filtro* (di seguito viene mostrato un esempio in cui il filtro è il genere musicale di appartenenza, nello specifico R&B), in cui si sceglie il filtro da adoperare, dal più generico “Trova tutti gli artisti” a “Trova gli artisti dato il nome o una parte di esso”, e la seconda è una *ricerca combinata*, in cui è possibile combinare i filtri presenti e il risultato mostrerà solamente gli artisti che soddisfano tutte le condizioni (nell'esempio della figura successiva vengono mostrati gli artisti appartenenti al genere pop, che abbiano avuto almeno 20 milioni di ascoltatori nell'ultimo mese su Spotify e che nel nome sia contenuta la stringa *sha*).

Spesa artisti già aggiunti	Budget disponibile	Generi selezionati	Generi privilegiati	Numero di artisti	Tempo di esecuzione
1.250.000 \$ (1 artista)	2.000.000 \$	R&B, rock, punk, rap/trap	rap/trap	Nessuna limitazione	107,5552 ms
0 \$	400.000 \$	Tutti	Nessuno	Nessuna limitazione	3038,1484 ms
0 \$	1.500.000 \$	classical, comedy music, dance, folk, j-pop/k-pop, latin	classical, folk	Nessuna limitazione	14,7271 ms
3.500.000 \$ (3 artisti)	4.000.000 \$	Tutti	Nessuno	5	1031,047 ms
0 \$	3.000.000 \$	country, alternative	Nessuno	3	521,6724 ms
0 \$	1.750.000 \$	jazz, rap/trap, latin, j-pop/k-pop	j-pop/k-pop	Nessuna limitazione	107,6691 ms
0 \$	2.500.000 \$	pop	Nessuno	3	1461,4107 ms

6.2 Link al video dimostrativo del software

Il video di dimostrazione di utilizzo del software è disponibile al seguente link di *YouTube*:

<https://youtu.be/oGdNnVZf9ms>

7. Valutazioni sui risultati ottenuti e conclusioni

L'obiettivo dell'applicazione è quello di rendere flessibile, veloce e intuitiva per gli organizzatori dei festival di musica dal vivo la ricerca degli artisti da ingaggiare, e il software da questo punto di vista ha raggiunto questo traguardo. Il programma, infatti, permette di effettuare ricerche in diversi modi (ricerca tramite filtro o ricerca combinata), permette all'utente di aggiungere alla soluzione *ottima* degli artisti, se necessario, e permette di inserire vari vincoli nella ricerca della soluzione *ottima* andando incontro alle varie esigenze degli organizzatori.

Per migliorare ancora maggiormente l'interattività con l'utente, su cui si insiste molto nel software presentato, si potrebbe implementare anche una gestione anagrafica degli artisti, in cui l'utente può a proprio piacimento eliminare, inserire o modificare gli artisti presenti. Non aggiungendo valore alla prova finale, questa funzionalità non è stata implementata.

Il limite maggiore del software è, in alcuni casi, il tempo impiegato nella ricerca della combinazione ottima. Infatti, nel caso in cui si inserisca un budget elevato (all'incirca maggiore di 500.000\$) e si considerino tutti i generi musicali, o quelli a cui appartengono un elevato numero di cantanti, il programma non riesce a elaborare una soluzione in tempi ragionevoli, a meno che non si ponga il vincolo della limitazione di artisti ad un numero ristretto (2 o 3). Il programma supera questo limite nel momento in cui vengono specificati vincoli più stringenti e incontra problemi solamente nei casi più generici.

Nonostante questo limite, l'applicazione si rivela molto utile e raggiunge gli obiettivi prefissati con successo: l'utente ha una vasta scelta di possibili interazioni col programma e, nella maggior parte dei casi, trova la combinazione ottima in tempi ragionevoli o molto rapidi.



Quest'opera è distribuita con Licenza [Creative Commons
Attribuzione - Non commerciale - Condividi allo stesso modo 4.0
Internazionale](https://creativecommons.org/licenses/by-nc-sa/4.0/).