



Progetto di Basi di Dati 2022/23 **Università Ca' Foscari Venezia**

Report Progetto Basi Di Dati **1.0**

PineappleMusic
by
ServerNotFound



Document Informations

Informazioni	
Deliverable	Report Progetto Basi dati
Data di Consegna	04/12/2022
Team members	STEFANO MASIERO , MAKSYM NOVYTSKYI , RICCARDO COMELLATO



Indice

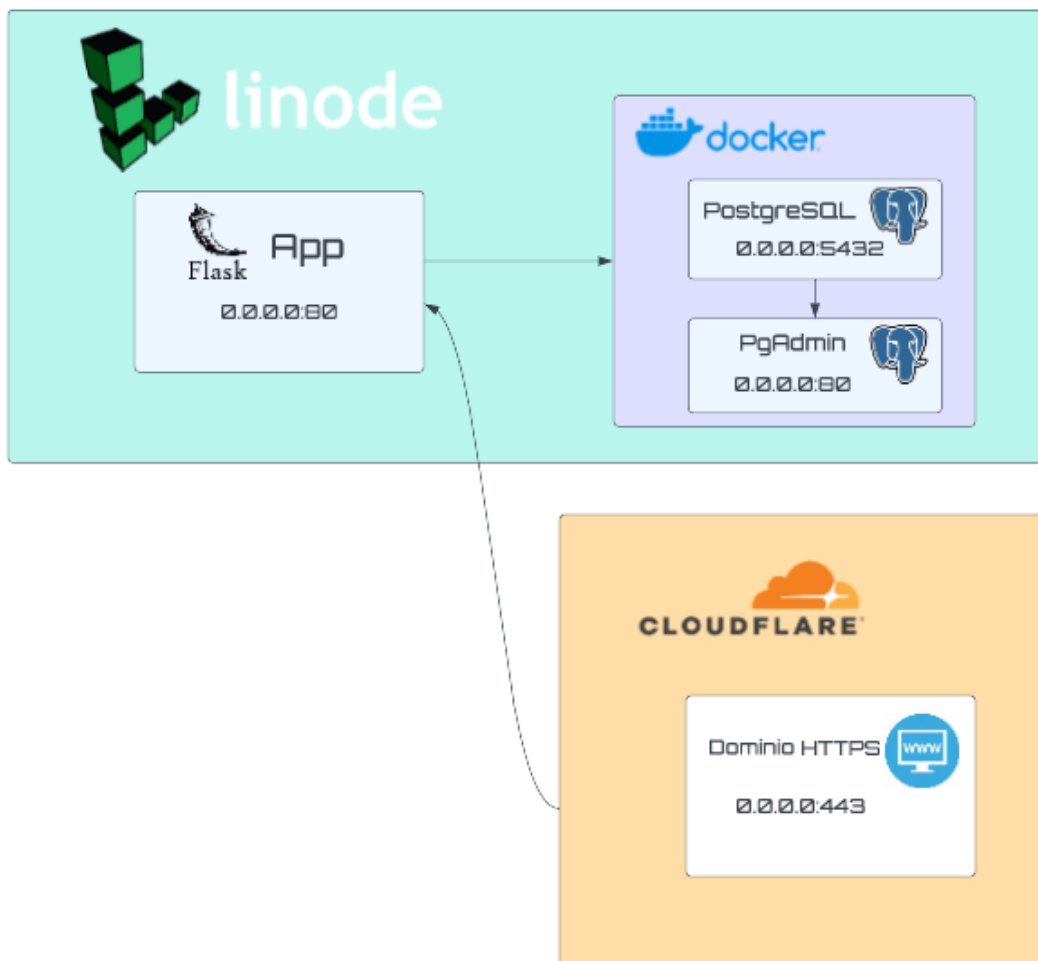
1. Introduzione	4
2. Installazione e Avvio	5
2.1. Pre-requisiti da installare	5
2.1.1. DBMS	5
2.1.2. Flask e SQLAlchemy	5
2.2. Installazione	5
2.2.1. Database	5
2.3. Avvio	6
3. Descrizione applicazione alto livello	6
4. Progettazione del DB	7
4.1. Schema Concettuale	7
4.1.1. Diagramma ad Oggetti	7
4.2. Schema Relazionale	8
4.3. Ruoli e permessi	9
4.4. Vincoli	10
4.5. Indici	10
4.6. Triggers	11
5. Progettazione Web App	13
5.1. Struttura applicazione	13
5.2. Mapping delle tabelle database nell'applicazione	15
6. Sicurezza	16
6.1. Cross-Site Scripting (XSS)	16
6.2. SQL Injection	17
6.3. Password Encryption	17
7. Front End	18
8. Audio e Image Files	19
9. Code Highlight	20
10. Schermate Illustrative	22
11. Materiale per il Development	25
12. Conclusioni	25



1. Introduzione

L'obiettivo del documento è documentare e chiarire lo sviluppo del progetto Basi di Dati (2021/2022). Il progetto consiste nella costruzione di una web application che si interfaccia con un database relazionale ed il tema selezionato consiste nello sviluppo di una Piattaforma di Streaming Audio (come [Spotify](#), [Apple Music](#)). Dal lato back-end l'applicazione è stata sviluppata in Python, utilizzando principalmente le librerie Flask e SQLAlchemy(ORM) con utilizzo di un database realizzato con PostgreSQL. Dal lato front-end l'applicazione è basata su HTML e CSS con utilizzo del framework [Bootstrap v5](#) e [Font Awesome](#) (libreria di icone) in più l'utilizzo di JavaScript per aggiungere funzionalità aggiuntive (come AudioPlayer e [Particle.js](#) usato nella homepage e nelle pagine per accedere). Al termine il progetto è stato successivamente rilasciato online tramite [Linode](#) (per hostare il Server Flask e il database). È possibile accedere al sito utilizzando il link <https://www.pineapplemusic.tech>.

Sotto viene riportata la struttura progetto usata per mettere in production la nostra applicazione online. Durante la fase di sviluppo è stato utilizzato in modo condiviso il database tramite il servizio hosting in modo tale che tutti i componenti del team accesso agli stessi dati:





2. Installazione e Avvio

In questo paragrafo saranno riportate tutte le informazioni necessarie per far partire l'applicazione in locale.

2.1. Pre-requisiti da installare

Sotto vengono riportati i componenti da installare per far partire l'applicazione:

2.1.1. DBMS

Assumendo che il lettore abbia installato un DBMS PostgreSQL oppure sappia installarlo (in caso contrario forniamo qui delle semplici guide per la sua installazione su [Windows](#) e [Linux](#)). In più si consiglia per comodità d'uso di installare PgAdmin, un'applicazione/interfaccia grafica che consente di amministrare in modo semplificato il database ([Windows](#) e [Linux](#)).

2.1.2. Flask e SQLAlchemy

Per far partire l'applicazione è necessario avere [Python](#) installato. Successivamente bisogna installare i necessari pacchetti tramite il gestore pip di python (compreso in python dalla versione 3.4). Tra questi pacchetti i principali sono Flask (micro-framework per la costruzione di applicazioni Web) e SQLAlchemy (toolkit SQL e mappatore relazionale a oggetti per il linguaggio Python). I pacchetti necessari si possono trovare in requirements.txt e per installare sarà necessario semplicemente inserire dal terminale il seguente comando:

```
pip install -r requirements.txt
```

2.2. Installazione

Dopo aver installato i requisiti possiamo passare ora all'installazione del database di PineappleMusic e la web app.

2.2.1. Database

Per far partire l'applicazione è necessario caricare il database di PineappleMusic sul DBMS. Per questo motivo vengono forniti i seguenti dump presenti nella repository (nella cartella *Utility/DumpDB*). Avrà due modi per procedere:

1. Il primo consiste nell'utilizzare un dump formato tar, però per poterlo utilizzare avrà bisogno di inserire manualmente nel database i ruoli (presenti nella sezione [Ruoli e Permessi](#)) situati anche nel file Roles nella cartella *Utilities/Other*. Successivamente sarà in grado di caricare il database con il backup formato tar. Per questo dovrà creare il database, eseguire l'opzione restore e selezionare il file **PineappleMusicTar**.
2. Il secondo consiste nel utilizzare il file plain text **PineappleMusicPlainWithRoles**. I ruoli sono inseriti al suo interno. Per utilizzarlo dovrà creare un database vuoto chiamato PineappleMusic e tramite il query tool inserire dentro i contenuti del file ed eseguire.



2.3. Avvio

Per avviare l'applicazione sarà semplicemente necessario far partire il seguente comando per avviare l'applicazione dal terminale/cmd nella cartella del progetto.

```
FLASK_APP=Codice/__init__.py FLASK_DEBUG=1 TEMPLATES_AUTO_RELOAD=1  
flask run --host 0.0.0.0
```

3. Descrizione applicazione alto livello

Il progetto si basa sulla realizzazione di un'applicazione web ovvero una piattaforma di streaming audio. In particolare viene richiesto di gestire (operazioni di CRUD) i metadati associati alle canzoni, come l'artista, il titolo, l'album, data produzione, etc. Però durante lo svolgimento del progetto è stato possibile ampliare integrando anche gli audio file e la loro effettiva riproduzione (Dettagli nel [Capitolo 8](#)).

Gli utenti hanno quindi la possibilità di registrarsi, ascoltare canzoni prive di copyright, lasciare feedback tramite pulsanti like e dislike e creare playlist private ovvero visibili al solo utente creatore con le canzoni preferite. In più l'utente avrà la possibilità di fare un upgrade del profilo da normale a premium inserendo informazioni di una carta di credito (di cui verrà semplicemente verificato il formato e non l'effettiva esistenza della carta). L'applicazione offre anche un sistema di suggerimento di canzoni all'utente in base ai like inseriti da esso in proporzione al genere e alle canzoni più ascoltate dagli utenti dell'applicazione.

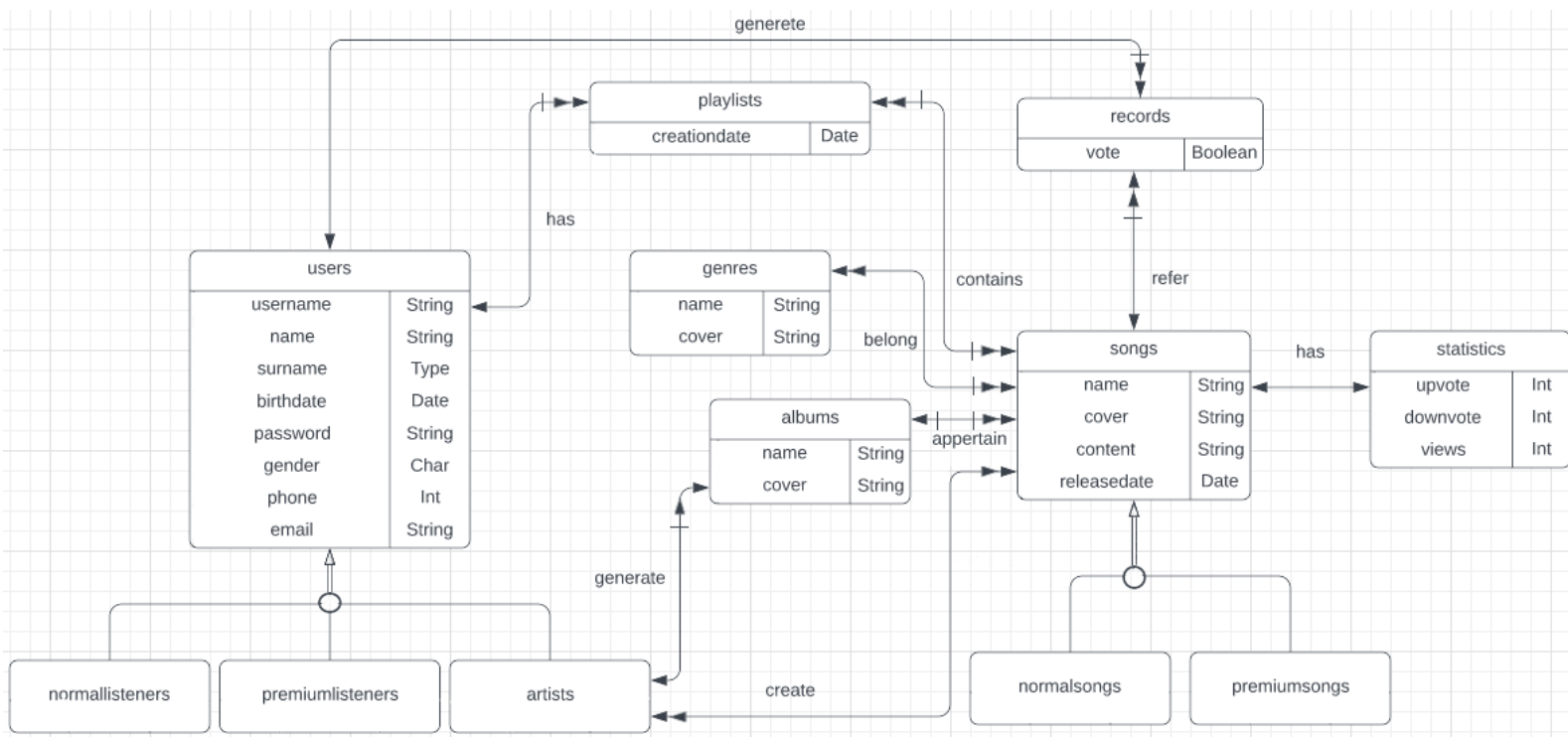
Gli artisti hanno le stesse feature del utente premium, in aggiunta hanno la possibilità di inserire canzoni e album, e poter visualizzare statistiche relative alle loro canzoni inserite. Quando l'artista si registra e inserisce i suoi dati dell'account, deve fornire una canzone nel sistema, se non lo fa e cerca di accedere senza inserire una canzone il suo account viene eliminato. Questa scelta è stata fatta per evitare che un utente normale crei un account artista vuoto per avere accesso a canzoni premium senza pagare. Le canzoni quindi sono state divise in due categorie, normali e premium, le ultime potranno essere ascoltate soltanto da utenti premium e artisti.



4. Progettazione del DB

Per il funzionamento dell'applicazione è necessario progettare una base di dati adeguata per fornire un sistema di gestione dei metadati delle canzoni, degli utenti presenti e delle loro playlist. A tale scopo vengono forniti i seguenti modelli: Modello ad Oggetti/Concettuale e Modello relazionale.

4.1. Schema Concettuale



4.1.1. Diagramma ad Oggetti

Users: questa entità rappresenta tutti i possibili utenti (Account) che utilizzano l'applicazione. Al suo interno vengono salvate le informazioni personali e i dati necessari per accedere al sito.

NormalListeners, PremiumListeners: sono sottotipi di Users usati per distinguere il tipo di account. Questi due hanno accesso a canzoni diverse sia a livello applicazione, sia al livello di ruoli del database.

Artists: anch'essa, sottotipo di Users, rappresenta gli artisti che sono registrati al interno dell'applicazione; in più avranno delle relazioni aggiuntive con Songs e Albums.

Playlists: rappresenta le liste di canzoni che un determinato User può aver creato. Al suo interno vengono salvate il nome, la data di creazione e l'autore.

Songs: rappresenta le canzoni caricate dagli artisti. Al suo interno sono presenti i metadati e i link per le cover e gli audio file.

NormalSongs, PremiumSongs: sottotipi di Songs; al loro interno viene segnato se una canzone è premium oppure no.



Albums: rappresenta gli album di un determinato artista; al suo interno contiene nome, cover e l'artista.

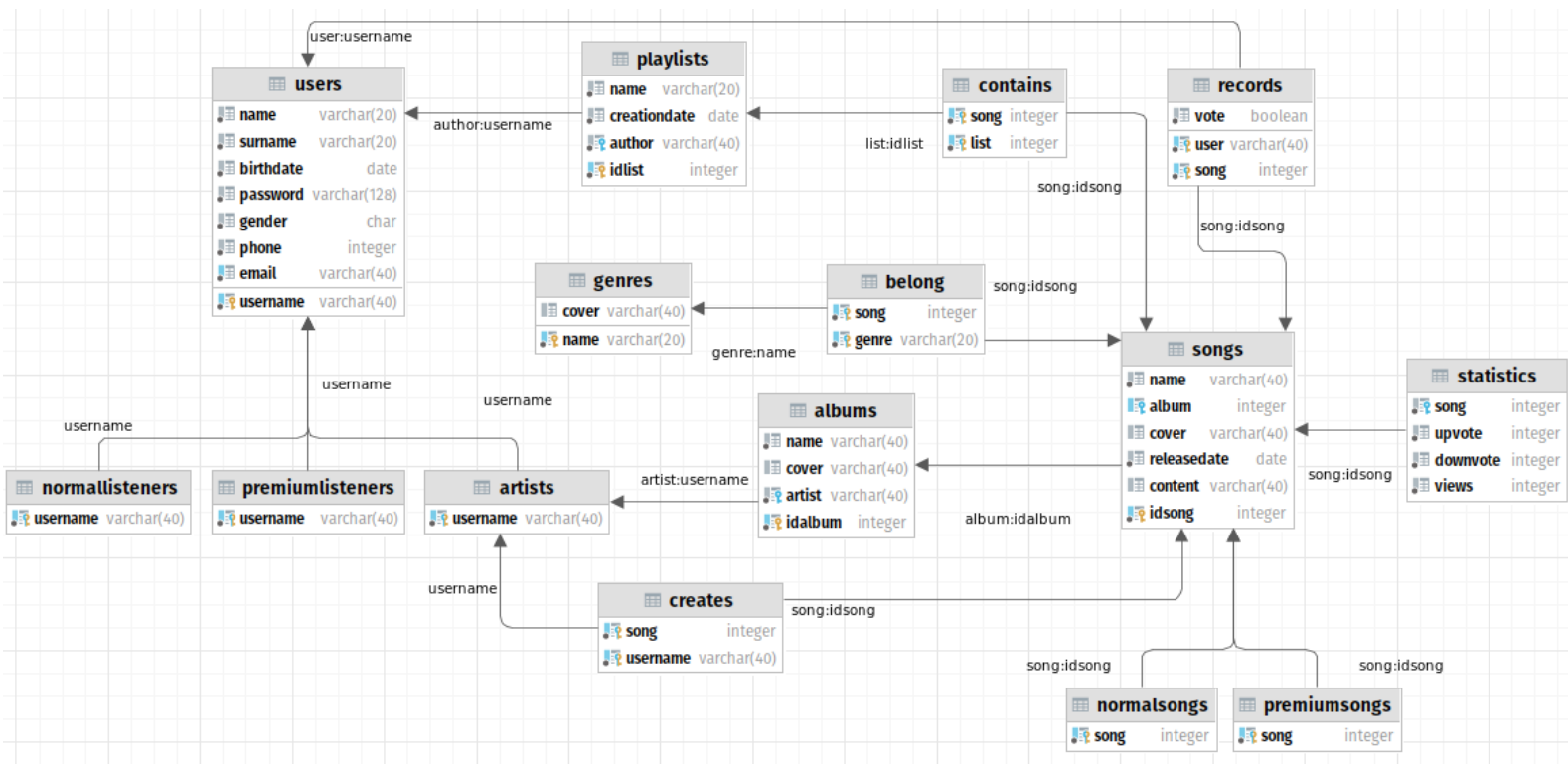
Genres: rappresenta i generi di canzoni esistenti e contiene al suo interno una cover.

Records: contiene i like/dislike di un utente per la relativa canzone.

Statistics: rappresenta le statistiche di una determinata canzone; viene usata per fare statistiche generali come le "TOP Song" e al suo interno contiene il numero di like, dislike e le views.

4.2. Schema Relazionale

Dopo aver progettato il modello Concettuale è necessario tradurlo in modello Relazionale preservando l'informazione essenziale e le relazioni tra entità. Sotto viene riportato lo schema finale.





4.3. Ruoli e permessi

Oltre ai ruoli a livello applicativo nell'App sono stati implementati dei ruoli a livello DBMS, allo scopo di limitare l'accesso allo stretto necessario per poter funzionare; in più sono stati aggiunti alcuni ruoli in più per motivi gestionali:

- Ruolo `guest_manager`: viene utilizzato per permettere all'utente di fare il login e la registrazione nell'applicazione.
- Ruolo `delete_manager`: viene utilizzato principalmente per effettuare l'eliminazione degli account e tutti i dati a lui correlati sfruttando l'effetto delete on cascade.
- Ruolo `listeners`: utilizzato dagli account normali per ascoltare la musica e sfruttare altre feature, avendo però accesso limitato.
- Ruolo `premium_listeners`: utilizzato dagli account premium, permette di avere accesso a tutte le canzoni.
- Ruolo `artists`: utilizzato dagli artisti, esso possiede le stesse feature del utente premium e offre la possibilità di inserire e manipolare le proprie canzoni e album.

```
CREATE USER guest_manager WITH PASSWORD 'PassGuestManager';
CREATE USER delete_manager WITH PASSWORD 'PassDeleteManager';
CREATE USER listeners WITH PASSWORD 'PassListeners';
CREATE USER premium_listeners WITH PASSWORD 'PassPremiumListeners';
CREATE USER artists WITH PASSWORD 'PassArtists';
```

Sotto viene riportata la tabella mostrando tutti i permessi concessi ai ruoli sovrastanti delle tabelle presenti nel database.

	US	NL	PL	AR	SO	NS	PS	ST	PY	CO	GR	BE	AL	CR	RE
G	SI	SID	SI	SI						SD					
D	SD	SD	SD	SD	SD	SD	SD	SD	SD	SD		SD	SD	SD	SD
L	SDU	SI		S	S	S		SU	SIDU	SID	S	S	S	S	SID
P	SDU		SI	S	S	S	S	SU	SIDU	SID	S	S	S	S	SID
A	SDU			SDU	SIDU	SID	SIDU	SIDU	SIDU	SID	S	SIDU	SIDU	SIDU	SID

Leggenda Tabelle:

US: users

NL: normallisteners

PL: premiumlisteners

AR: artists

SO: songs

NS: normalsongs

PS: premiumsongs

ST: statistics

PY: playlists

CO: contains

GR: genres

BE: belong

AL: albums

CR: creates

RE: records



Leggenda Ruoli:

G: guest_manager

D: delete_manager

L: listeners

P: premiumlisteners

A: Artists

Leggenda Permessi:

S: Select

I: Insert

D: Delete

U: Update

4.4. Vincoli

Sono necessarie delle scelte progettuali utili per poter delineare al meglio tutte le modalità per implementare nella modo migliore tutto il codice. Per questo motivo sono stati implementati dei vincoli, trigger, check su attributi. I vincoli sono:

- NOT NULL: ci sono delle righe delle tabelle che sono accompagnate dal vincolo Not Null. Infatti quelle righe devono avere un valore.
- PRIMARY KEY : Tutte le tabelle presentano un codice identificativo come primary key in grado di identificare univocamente ogni entry inserita nella base di dati.
- FOREIGN KEY: Il vincolo di Foreign Key è facile da posizionare per poter ottenere relazioni tra più tabelle collegandole tra di loro.
- UNIQUE: Il vincolo Unique ci torna utile per garantire che non vengano immessi valori duplicati in colonne specifiche che non fanno parte di una chiave primaria.

4.5. Indici

Dopo una breve analisi sulle richieste si è osservato che tra le query più frequenti sono presenti genere e album per estrarre canzoni. Per questo motivo sono stati implementati i seguenti indici utilizzando la struttura b-tree (permette rapida localizzazione dei file, riducendo il numero di volte necessarie al utente per accedere alla memoria in cui il dato è salvato) per migliorare le performance.

```
CREATE INDEX ON public.songs USING btree (album)
```

```
CREATE INDEX ON public.belong USING btree (genre)
```



4.6. Triggers

Durante la progettazione è stata sviluppata la tabella records per gestire i like/dislike delle canzoni conservando l'informazione dell'utente che esegue l'azione. Però è sorto il problema della loro visualizzazione: eseguire un count dei record per ogni canzone comincerebbe a richiedere molte risorse. Per tale motivo è stata realizzata la tabella statistics contenente un contatore dei like/dislike per canzone. I contatori però necessitano di un metodo per aggiornarli per questo sono stati creati i seguenti trigger update_statistics_vote e delete_statistics_vote.

La funzione e trigger sottostante incrementa l'upvote counter se viene aggiunto un like nei records altrimenti se viene aggiunto un dislike viene incrementato il downvote.

```
CREATE OR REPLACE FUNCTION update_vote()
RETURNS TRIGGER AS
$func$
BEGIN
IF new.vote='true' then
    UPDATE statistics
    SET upvote = upvote + 1
    WHERE song = new.song;
    RETURN NEW;
ELSE
    UPDATE statistics
    SET downvote = downvote + 1
    WHERE song = new.song;
    RETURN NEW;
END IF;
END
$func$ LANGUAGE plpgsql;

CREATE TRIGGER update_statistics_vote
AFTER INSERT
ON records
FOR EACH ROW
EXECUTE PROCEDURE update_vote();
```



La funzione trigger sottostante decrementa l'upvote counter se viene eliminato un like nei records, altrimenti se viene eliminato un dislike viene decrementato il downvote

```
CREATE OR REPLACE FUNCTION delete_vote()
RETURNS TRIGGER AS
$func$
BEGIN
IF old.vote='true' then
    UPDATE statistics
    SET upvote = upvote - 1
    WHERE song = old.song;
    RETURN NEW;
ELSE
    UPDATE statistics
    SET downvote = downvote - 1
    WHERE song = old.song;
    RETURN NEW;
END IF;
END
$func$ LANGUAGE plpgsql;

CREATE TRIGGER delete_statistics_vote
AFTER DELETE
ON records
FOR EACH ROW
EXECUTE PROCEDURE delete_vote();
```



5. Progettazione Web App

5.1. Struttura applicazione

Essendo un'applicazione di medie dimensioni utilizzare una struttura flatten per gestire i file complicherebbe fortemente lo sviluppo. A tale scopo è stato fondamentale creare una struttura suddivisa per avere una migliore coordinazione, aumentare la produttività, ridurre i tempi di lavoro. Per poter realizzarlo è stato adottato Flask Blueprint che permette di incapsulare funzionalità come views, templates e altre risorse come file statici. Successivamente ogni Blueprint diventa oggetto utilizzato dalla web application nel file `__init__.py`. Il progetto allora è stato suddiviso in diversi moduli specifici per argomento/feature dell'applicazione:

- **Homepage:** Include in sé la landing page.
- **Auth:** Utilizzata per la registrazione di nuovi utenti e la loro autenticazione.
- **User:** Racchiude feature utilizzate per la gestione dell'account da parte dell'utente.
- **Artist:** Incorpora feature utilizzate dagli artisti come gestione delle canzoni, album e la visualizzazione delle statistiche
- **Music:** Gestisce tutte le feature che riguardano le canzoni
- **Error:** In caso di errore l'handling viene gestito da questo blueprint.

N.B. Dentro ogni blueprint è presente il file `routes.py` e `forms.py`. Le *routes* sono un pezzo di codice che viene eseguito quando il server viene chiamato a un determinato indirizzo URL. I file *forms* contengono eventuali moduli da compilare se richiesti all'utente (ad esempio modulo registrazione e accesso), utilizzano la libreria *FlaskForms* (semplice integrazione di Flask e WTForms).

Il progetto inoltre ha un'altra suddivisione interna tra i file, sia nella cartella generale del codice sia in ogni blueprint sono presenti due sotto cartelle `static` e `templates`:

- *static*: comprende in sé tutti i file statici, come immagini ma anche css e javascript
- *templates*: contiene i template html

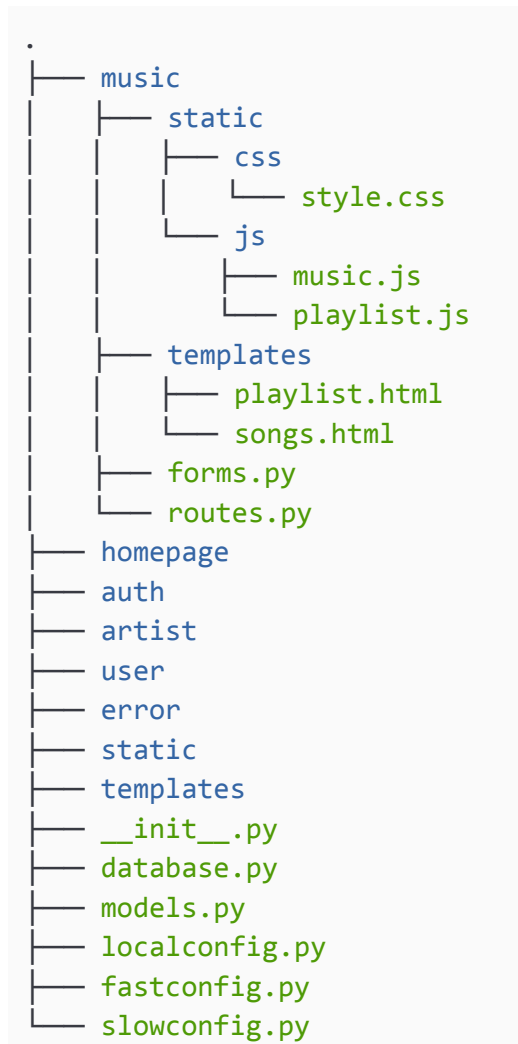
I restanti file sono organizzati nel seguente modo:

- `__init__.py`: necessario per far eseguire l'applicazione.
- `config.py`: contiene parametri di configurazione con il database. Nel nostro caso abbiamo utilizzato tre diversi file configurazione dove sono salvati credenziali di accesso usati durante il development.
- `database.py`: contiene le sessioni usate per connettersi al database
- `models.py`: contiene la definizioni dei modelli ORM per interagire con il database

N.B. Se si vuole cambiare il file configurazione utilizzato bisogna modificare gli import presenti in `__init__.py` e `database.py`



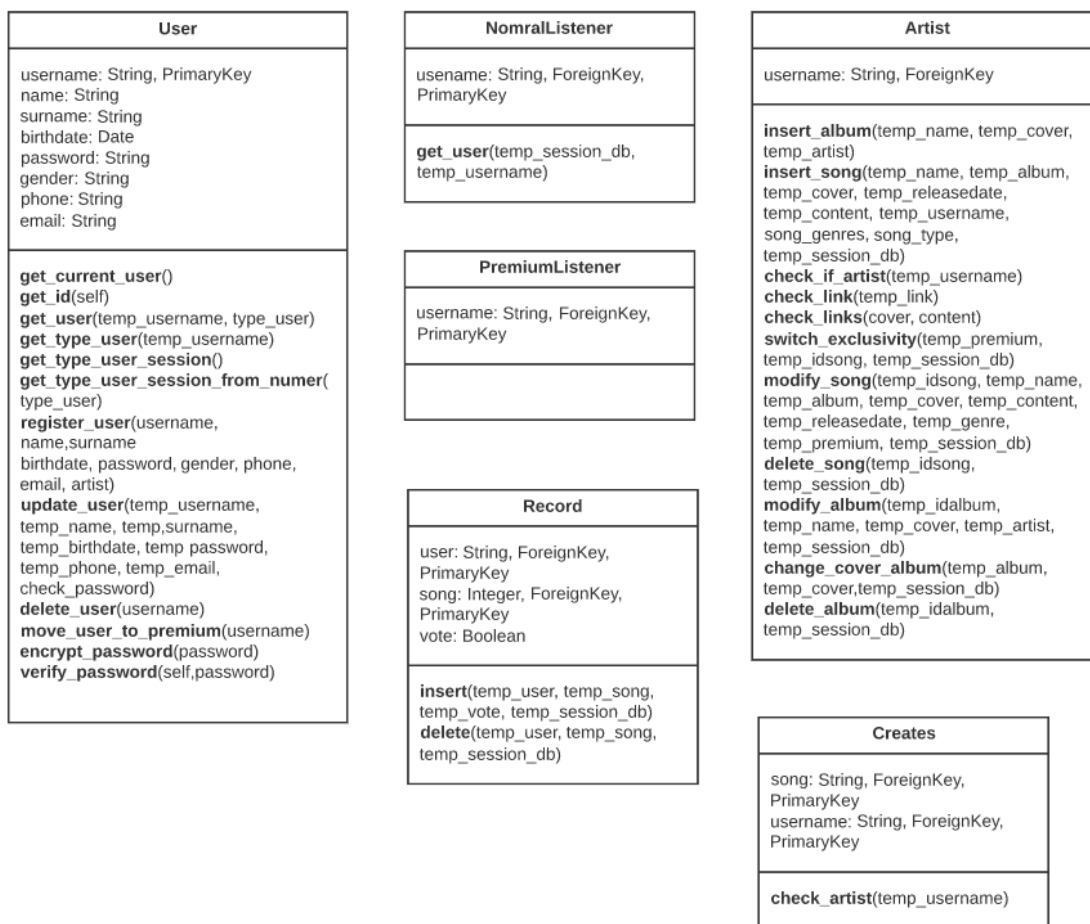
Sotto viene riportata la struttura ad albero del progetto a scopi informativi.





5.2. Mapping delle tabelle database nell'applicazione

Per avere interazioni con il database all'interno dell'applicazione è stato integrato SQLAlchemy ORM che consente la comunicazione col database tramite oggetti python. Per tale motivo sorge la necessità nella definizione di classi. Sotto vengono riportati diagrammi UML per ogni classe. Ogni classe conterrà attributi, relazioni (livello database) e anche metodi usati per interagire; tali metodi sono facilmente intuibili, però se ci sono problemi si può leggere una breve descrizione nel codice sorgente models.py.





Song
idsong: Integer, PrimaryKey name: String cover: String contente: String releasedate: Date album: Integer, ForeignKey
get_songs (temp_user, temp_session_db, genre="") get_songs_artist (temp_artist) get_song_id (temp_idsong) get_song_playlist (temp_user, idplaylist) get_top_like_songs (temp_session_db, redirect_search = False, temp_user = None) get_top_view_songs (temp_session_db, redirect_search = False, temp_user = None) get_suggestion_songs (temp_user, temp_session_db, redirect_search = False)

NormalSong
song: Integer, PrimaryKey, ForeignKey
check_song (temp_song, temp_session_db)

Genre
name: String, PrimaryKey cover: String
get_genres () get_genre_list_database ()

PremiumSong
song: Integer, PrimaryKey, ForeignKey
check_song (temp_song, temp_session_db)

Belong
genre: String, ForeignKey, PrimaryKey song: String, ForeignKey, PrimaryKey
get_genre_list (temp_song_id, temp_session_db)

Statistic
song: Integer, ForeignKey, PrimaryKey upvote: Integer downvote: Integer views: Integer
increase_views (temp_song, temp_session_db) get_statistics (temp_username, temp_session_db) insert_statistics (temp_song, temp_session_db)

Album
idalbum: Integer, PrimaryKey name: String cover: String Artist: String, ForeignKey
check_artist_album_name (temp_username, temp_name) get_albums_username (temp_username) get_albums_id (temp_idalbum) get_albums_name (temp_username, albums, choice) extract_cover_album (list_albums, choice) extract_id_album (list_albums, choice)

Playlist
idlist: Integer, PrimaryKey name: String creationdate: Date author: String, ForeignKey
create (temp_session_db, temp_name, temp_author) get_playlist_user (temp_session_db, temp_author) get_playlist_name_id (temp_username, playlists) delete_playlist (idplaylist, temp_session_db)

Contains
song: Integer, ForeignKey, PrimaryKey list: Integer, ForeignKey, PrimaryKey
create (songid, playlistid, temp_session_db) delete_song_from_playlist (idsong, idplaylist, temp_session_db)

6. Sicurezza

Nel seguente capitolo verranno elencate le pratiche e le prevenzioni usate per rendere l'applicazione più sicura.

6.1. Cross-Site Scripting (XSS)

Cross site scripting consiste nel iniettare codice arbitrario HTML (e con esso anche possibile JavaScript malevolo) all'interno del sito. Flask per evitare questo configura Jinja2 automaticamente bloccando il problema tramite escaping. Tuttavia possono sorgere problemi se non si è attenti, per questo è stata seguita la seguente guida per evitare errori comuni che possono causare vulnerabilità

<https://semgrep.dev/docs/cheat-sheets/flask-xss/> .



6.2. SQL Injection

La maggior parte delle query sono state costruite con i metodi appartenenti alla libreria SQLAlchemy, utilizzando come approccio ORM che sanitizza in automatico i parametri che gli passiamo.

```
def get_user(temp_session_db, username):  
    try:  
        user = temp_session_db.query(User).filter(  
            User.username == username).first()  
        return user  
    except:  
        temp_session_db.rollback()
```

6.3. Password Encryption

Per garantire la sicurezza agli utenti evitando possibili leak di password è stato fondamentale come minimo criptarle. Questo è stato realizzato tramite la libreria Werkzeug che offre la possibilità di criptare la password utilizzando il protocollo SHA-256.

Sotto viene riportato il codice usato dal programma per criptare le password e verificare la corrispondenza al login.

```
from werkzeug.security import generate_password_hash,  
check_password_hash  
  
def encrypt_password(password):  
    return generate_password_hash(password)  
  
def verify_password(self, password):  
    return check_password_hash(self.password, password)
```

Esempio di entry all'interno del database:

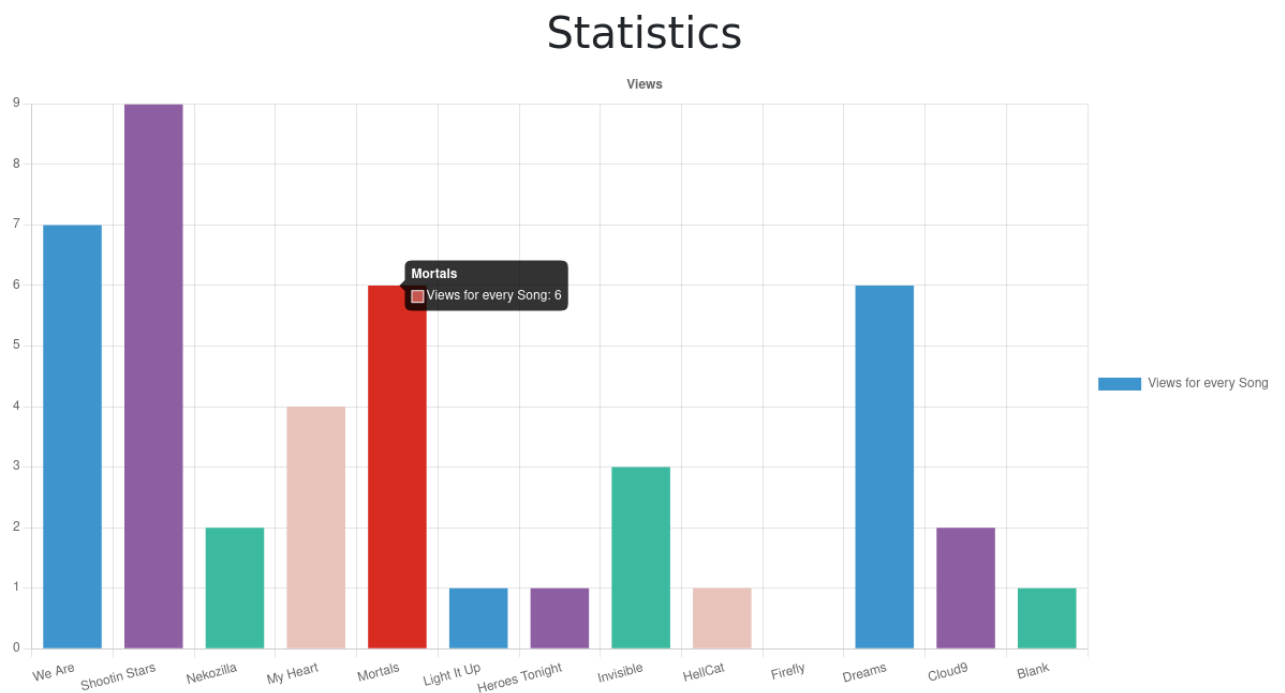
```
Username: "Beethoven"  
Name: "Ludwig"  
Surname: "Beethoven"  
Birthdate: "1770-12-17"  
Password: "pbkdf2:sha256:260000$oYN2QxRyHehbJuFU$96cc056c389f704a34d7  
621ee90e23bf4d955694170687319ceab658e11f701c"  
Gender: "M"  
Email: "Beethoven@artist.com"  
Phone: "3600702749"
```



7. Front End

Per la realizzazione del Front End sono state utilizzate alcune librerie utili per rendere più gradevole la grafica del sito senza dover allo stesso tempo eccessivamente maneggiare il CSS. Quindi è stato utilizzato [Bootstrap](#) che mette anche a disposizione un utile sistema di griglie per rendere il design del sito responsive. [Font Awesome](#) è stato integrato con lo scopo di includere icone. Infine [Particle](#) per realizzare gli effetti grafici dello sfondo tramite javascript sulla homepage e sulle schermate di autenticazione e registrazione. In più Javascript è stato utilizzato per la riproduzione delle canzoni e l'utilizzo del player musicale. Inoltre con l'aiuto di [Datatable](#) è stato possibile inserire e visualizzare le [canzoni in tabelle](#), in modo esteticamente gradevole. Per agevolare gli utenti Artisti sono stati implementati grafici per la visualizzazione di statistiche relative alle loro canzoni inserite (come like, dislike e views). Questo è stato possibile grazie alla libreria JavaScript [charJS](#).

Sotto viene riportato un esempio di tabella views nelle statistiche.





8. Audio e Image Files

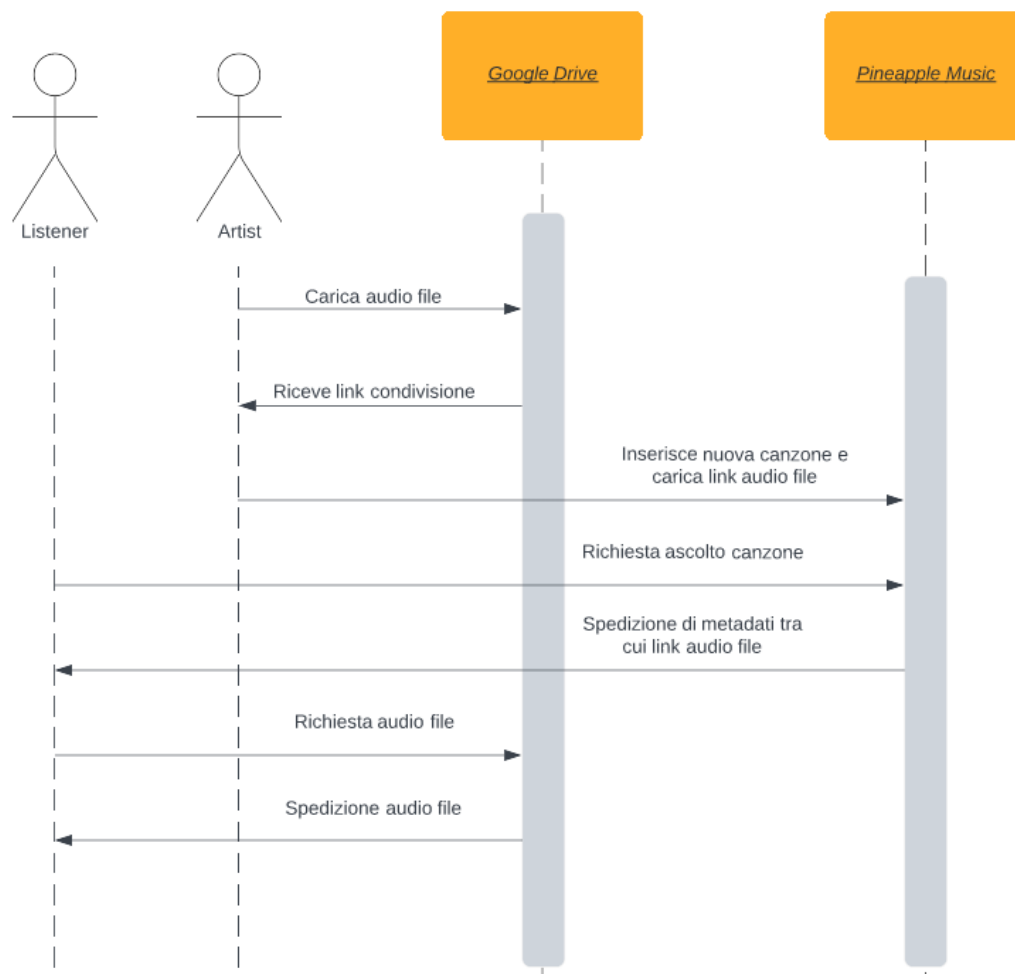
Inizialmente l'idea era di inserire i vari file audio e immagini all'interno del database come blob, ma successivamente a diverse ricerche è stato notato che la maggior parte dei siti non salva i file sul database stesso per questioni di performance e spazio. Tali file vengono salvati su un server a parte, mentre sul database vengono conservati i link a tale server, così da aver distribuito il carico. Quindi per evitare di dover mettere in piedi un server che ospiti i file audio e immagini delle canzoni è stata presa la decisione di sfruttare i servizi di hosting file offerti da Google Drive.

All'artista sarà semplicemente richiesto di caricare la sua canzone e cover sul suo Google Drive personale, selezionare l'opzione "condividi file tramite link" e infine fornire i link sul modulo di inserimento canzone.

N.B.: all'inserimento verrà verificato se i link sono validi

Successivamente se un utente vuole ascoltare la canzone inserita deve eseguire la richiesta; il sito fornirà al browser dell'utente il link Google Drive (modificato offrendogli la possibilità di ascoltare l'audio).

Per chiarire i dubbi viene fornito il seguente diagramma sequenziale che rappresenta il funzionamento di un inserimento di un file audio..





9. Code Highlight

In questa sezione viene illustrata una delle parti più interessanti del progetto. La parte di cui si sta parlando è la funzione utilizzata per suggerire canzoni ad un utente.

La prima parte consiste nel contare i like inseriti dall'utente e dividerli per genere.

```
if temp_session_db == Session_listener:
    count_of_genre = temp_session_db.query(Genre.name
                                            .label("genre"), count(Record.song)
                                            .label("likes"))
    .join(Belong)
    .join(Song)
    .join(Record)
    .join(NormalSong)
    .filter(and_(Record.vote == True,
Record.user == temp_user))
    .order_by(desc("likes"))
    .group_by(Genre.name)
    .limit(10)
    .all()
```

N.B.: Codice simile viene applicato per i *PremiumListener* e *Artists*

Successivamente viene controllato se la query ritornata è vuota; se lo è la funzione termina e viene sostituito il risultato con un'altra funzione

```
if count_of_genre == []:return None
```

Se il controllo non termina la funzione prosegue e viene creato un array bidimensionale per tenere l'informazione riguardante il genere e la percentuale di like relativa a quel genere.

```
total_likes = 0
for i in count_of_genre:
    total_likes = total_likes + i.likes

arr = []
for i in count_of_genre:    #determine percentage of song to
show per genre
    temp = round(((i.likes * 100) / total_likes)/10)
    if temp != 0:
        col = []
        col.append(i.genre)
        col.append(temp)
        arr.append(col)
```



Quando è terminato il riempimento dell'array esso viene utilizzato per creare query singole che richiedono canzoni in proporzione al numero di esse presenti per genere.

```
if i in arr: #extract songs per each genre
    if redirect_search:
        ...
    else:
        if temp_session_db == Session_listener:
            song =temp_session_db.query(Song.name.label("name"),
                                         Song.cover.label("cover"),
                                         Belong.genre.label("genre"),
                                         Creates.username.label("artist"),
                                         Album.name.label("album"),
                                         Statistic.views.label("views"))
                                         .join(Belong)
                                         .join(Creates)
                                         .outerjoin(Album)
                                         .join(NormalSong)
                                         .join(Statistic)
                                         .order_by(desc(Statistic.views))
                                         .filter(Belong.genre == i[0])
                                         .limit(i[1])
```

N.B.: Questa parte viene applicata anche per i *PremiumListener* e *Artist*, però con la lista completa delle canzoni.

N.B.: La funzione è usata non solo nella user home page, ma anche se l'utente clicca la lista, esso viene reindirizzato a un'altra pagina dove può rivedere la lista e ascoltarla. Per questo è presente il `redirect_search` che ripete il codice con piccole alterazioni.

Contemporaneamente nel loop indicato sopra viene eseguito il seguente codice che unisce tutte le query create.

```
if i == arr[0]:
    result_query = song
else:
    result_query = union(result_query, song) #union of all the queries
```

Al termine del for la query finale completa viene eseguita dal database e il risultato viene restituito dalla funzione:

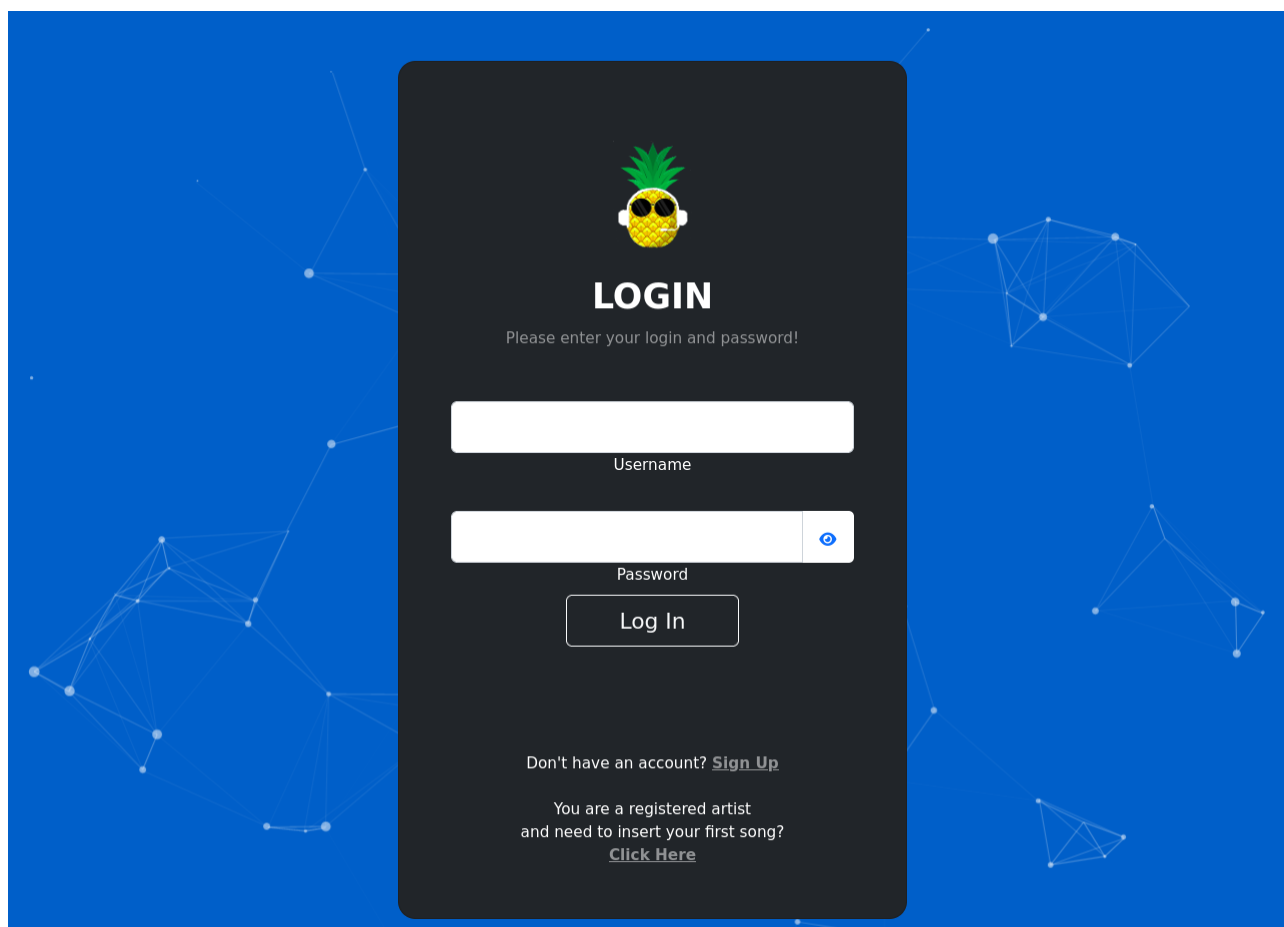
```
result = temp_session_db.execute(result_query).all()
return result
```



10. Schermate Illustrative


Sotto verranno fornite alcune schermate illustrative nel caso in cui il lettore non abbia la possibilità di accedere al sito oppure nel caso in cui il sito si trovi offline.

Schermata Login:






Schermata User Home:




PineappleMusic

- Home
- Search
- Playlist
- Profile
- Album
- Song
- Statistics


Home




Classic




Jazz




Latin




Pop




Metal



Rap



Rock





Reggae

TOP 10 VIEWS


Show entries

Search:

Cover	Name	Artist	Genre	Album	Views
	Why We Lose	Cartoon	Pop	Eternity	15
	We Are	OtherArtists	Pop	None	7



Schermata utilizzata per visualizzare le canzoni:



PineappleMusic










- Home
- Search
- Playlist
- Profile


Matrix

Search Songs

Show 10 entries

Search:

Cover	Artist	Name	Album	Genre	Vote	Add to Playlist
	OtherArtists	Nekozilla	None	Pop	<div><div></div><div></div></div>	<div>List of your playlist</div> <div>Add</div>
	OtherArtists	My Heart	None	Pop	<div><div></div><div></div></div>	<div>List of your playlist</div> <div>Add</div>
	OtherArtists	Mortals	None	Pop	<div><div></div><div></div></div>	<div>List of your playlist</div> <div>Add</div>
	OtherArtists	Light It Up	None	Pop	<div><div></div><div></div></div>	<div>List of your playlist</div> <div>Add</div>
	OtherArtists	Heroes Tonight	None	Pop	<div><div></div><div></div></div>	<div>List of your playlist</div> <div>Add</div>
	OtherArtists	Invisible	None	Pop	<div><div></div><div></div></div>	<div>List of your playlist</div> <div>Add</div>
	OtherArtists	HellCat	None	Pop	<div><div></div><div></div></div>	<div>List of your playlist</div> <div>Add</div>
	OtherArtists	Firefly	None	Pop	<div><div></div><div></div></div>	<div>List of your playlist</div> <div>Add</div>
	OtherArtists	Dreams	None	Pop	<div><div></div><div></div></div>	<div>List of your playlist</div> <div>Add</div>



Artist Name

0:00

3:15



11. Materiale per il Development

Sotto viene elencata la lista delle risorse utilizzate per lo sviluppo dell'applicazione.

Libro Flask Web Development, 2nd Edition O'Reilly Media, Inc.: per avere un'idea generale su come funziona flask

Materiale fornito dal prof. Stefano Calzavara ovvero i file: ORM.ipnyb e SQLAlchemy.ipnyb

[Documentazione flask](#): per chiarire eventuali dubbi di Flask.

[Documentazione sqlalchemy](#): per chiarire eventuali dubbi di SQLAlchemy e ORM.

[Stackoverflow](#): per risolvere eventuali problemi nella programmazione.

[Google Drive](#): Per caricare immagini e audio file

12. Conclusioni

Il progetto ha offerto al team la possibilità di apprendere nuove conoscenze nell'ambito web development. In particolare, lavorare con Flask e SQLAlchemy è stata per tutti una nuova esperienza e l'utilizzo di ORM, dopo un'iniziale fase di apprendimento, ha reso lo sviluppo del progetto molto intuibile e soddisfacente.

Inoltre questo progetto ha dato l'opportunità di simulare un vero e proprio deployment dell'applicazione, tramite il web hosting. Questo ha avuto delle sue problematiche, ovvero durante lo sviluppo il primo servizio scelto per l'hosting [Heroku](#) ha cambiato termini e condizioni per gli utenti con account gratuiti, bloccando il nostro progetto. Per tale motivo siamo stati costretti a passare al secondo fornitore [Linode](#).

In più essendo un progetto i servizi di Google Drive ci hanno fornito la possibilità di integrare gli audio file senza costi aggiuntivi. Questo metodo però non è una soluzione affidabile e adeguata per una vera applicazione, infatti abbiamo riscontrato alcuni problemi; ad esempio quando un utente esegue molte richieste verso Google esso viene bloccato per un certo periodo di tempo (timeout).

Il progetto sicuramente è una parodia semplificata di un'applicazione seria e ci sono sicuramente campi del progetto da migliorare. Alcuni possibili miglioramenti sono ampliare il database per offrire più informazioni relative agli utenti e le sue interazioni nell'applicazione; questo permetterà ad esempio lo sviluppo di algoritmi di raccomandazione più complicati ed efficaci. Un altro passo da migliorare è la sicurezza, infatti il team non ha approfondito tanto l'argomento e ha semplicemente fatto prevenzioni superficiali.

Concludendo è stato fondamentale svolgere il seguente progetto per comprendere e applicare i concetti essenziali appresi durante lo svolgimento del corso Basi di Dati [CT0006] e vogliamo fortemente ringraziare i docenti Prof. Alessandra Raffaetà e Prof. Stefano Calzavara per il loro lavoro.