


Basi di Dati Mod. 2 - Progetto A.A. 2021/2022

Piattaforma di streaming audio  **Spotifake**

Gruppo: Furellato

Componenti: Furegon Luca 886933, Pistellato Martino 886493

1. Premesse

L'applicazione è stata realizzata in Flask utilizzando SQLAlchemy come ORM per interfacciarsi al DBMS in PostgreSQL. Di conseguenza, per il corretto funzionamento dell'app è necessario creare un database chiamato "proge_db" ed andare a modificare nome e password dell'admin nei file models.py e config.py al fine di garantire la corretta connessione.

Infine, tramite i comandi

```
set FLASK_APP=blueprint
```

```
flask run
```

è possibile avviare l'applicazione.

2. Introduzione

"Spotifake" è una piattaforma di streaming audio dove viene gestita solamente la parte relativa ai metadati. Non è quindi possibile ascoltare e/o caricare effettive canzoni, tuttavia se ne può simulare il funzionamento e la gestione.

3. Funzionalità principali

È possibile accedere all'applicazione attraverso la creazione di un account che può essere rappresentato da tre categorie: Free, Premium o Artista.

Un Artista è libero di pubblicare canzoni e album indicandone peraltro la visibilità. Un brano il cui accesso è ristretto sarà visibile solamente ad utenti Premium. L'artista ha anche accesso ad una schermata dove sono indicate le principali informazioni e statistiche legate ai suoi brani: verrà mostrata la distribuzione geografica, delle età e del sesso degli ascoltatori. Infine, così come ogni altro utente, potrà mettere "Mi piace" ai brani (o agli album) di altri artisti e creare playlist personali.

Un utente Premium sarà libero di accedere a qualsiasi tipo di canzone o album, mettere "Mi piace" e creare playlist.

Un utente Free potrà accedere solamente ai carichi con visibilità "Free", pur mantenendo la possibilità di esprimere preferenze e creare raccolte.

Dalla pagina legata al proprio profilo è sempre possibile modificare il nome e foto (l'upload delle immagini non è implementato e perciò sono standardizzate)

Ogni utente ha la possibilità di crearsi una sua libreria andandosi a salvare brani e/o album e raccoglierli in playlist delle quali può modificare sia nome che copertina (le copertine sono gestite come le foto profilo). Non è necessario però che le canzoni all'interno di una playlist siano quelle salvate in libreria: è possibile anche fare un mix.

Tra le funzionalità incluse c'è anche un algoritmo in grado di consigliare brani in base a ciò che un utente ascolta.

4. Progettazione

1. Progettazione concettuale

Il database consiste di quattro tabelle principali e due sotto-tabelle figlie:

- Users

Rappresenta gli utenti iscritti. È composta dai campi Email (primary key), Password, Name, BirthDate, Country, Gender e Profile. Più in particolare, i campi con le informazioni personali servono per i dati e le statistiche mostrate all'artista rispetto i suoi ascoltatori. Infatti, gli verrà mostrato in quali paesi (e con che percentuale) vivono, la distribuzione del sesso e delle età. Il campo Profile viene utilizzato all'interno dell'applicazione per identificare le azioni disponibili all'utente, mentre il nome che viene scelto e mostrato può essere cambiato quando si desidera.

Questa tabella ha due figli per separare le azioni possibili tra le diverse classi di utenti. In particolare, gli Artisti hanno a disposizione molte più funzioni rispetto un utente Free o Premium, per cui era logico separarli piuttosto che lasciare tutto in uno. Dal punto di vista del modello concettuale, la tabella Artists non presenta campi diversi dalle altre, tuttavia dal punto di vista implementativo essa racchiude anche una serie di funzioni utili ad effettuare le azioni concesse.

- Premium

Nello schema concettuale non porta differenze rispetto la tabella genitore, tuttavia non è detto che rimanga così in futuro e la possibilità di aggiungere funzioni ad un utente premium è aperta.

- Artists

Nello schema concettuale non porta differenze rispetto la tabella genitore.

- Playlists

Rappresenta le raccolte che possono essere create dagli utenti. Non hanno una durata massima e possono contenere sia le canzoni salvate dall'utente (ossia brani ai quali ha messo "Mi piace" e che appaiono nella libreria) sia quelle per le quali non ha espresso preferenze.

- Songs

Contiene tutte le canzoni caricate nel sistema, hanno una durata minima di un minuto e massima di trenta minuti. Il campo Is_Restricted serve a limitarne la visibilità e se viene settato a true allora gli utenti Free non potranno vederlo. Il genere poteva essere un collegamento ad un'altra tabella che conteneva tutti i generi, tuttavia per semplicità è stato sostituito con una semplice stringa che, al momento della creazione, assume il valore di uno dei generi preimpostati

(ovviamente con possibilità di scelta partendo da un elenco di generi disponibili).

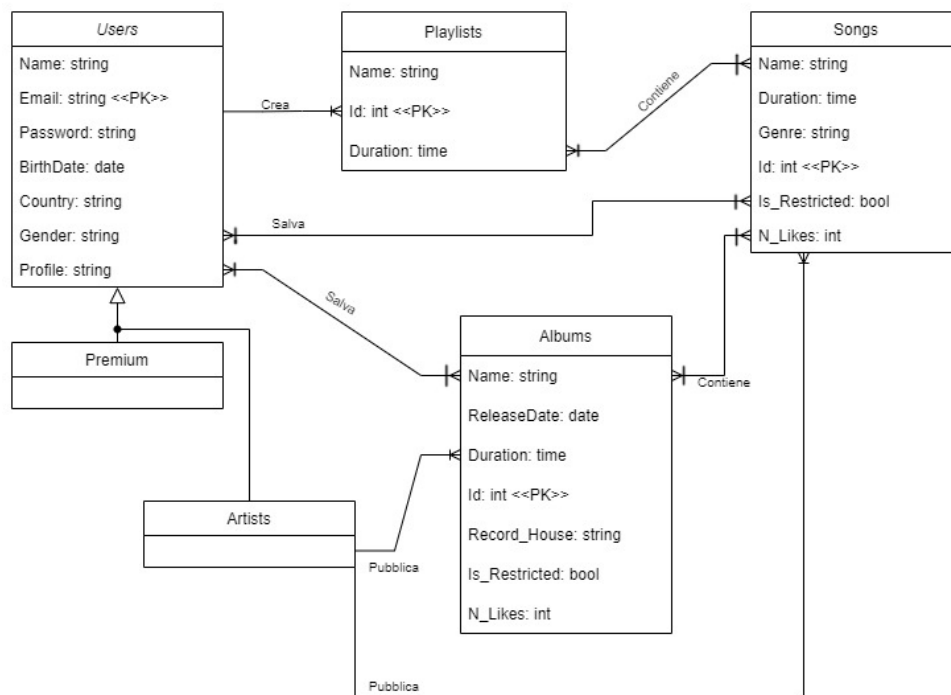
- Albums

Identifica gli album all'interno del sistema. Essi hanno un limite massimo di durata di un'ora e mezza, la data di rilascio viene impostata automaticamente a quella attuale al momento della creazione e il campo Record_House simboleggia la casa discografica che può essere o una di quelle preimpostate o "indipendente".

Le relazioni tra le tabelle sono le seguenti:

- Users - (Crea) - Playlists 1 : N
- Users - (Salva) - Songs N : N
- Users - (Salva) - Albums N : N
- Artists - (Pubblica) - Songs 1 : N
- Artists - (Pubblica) - Albums 1 : N
- Playlists - (Contiene) - Songs N : N
- Albums - (Contiene) - Songs N : N

Di seguito la rappresentazione visiva

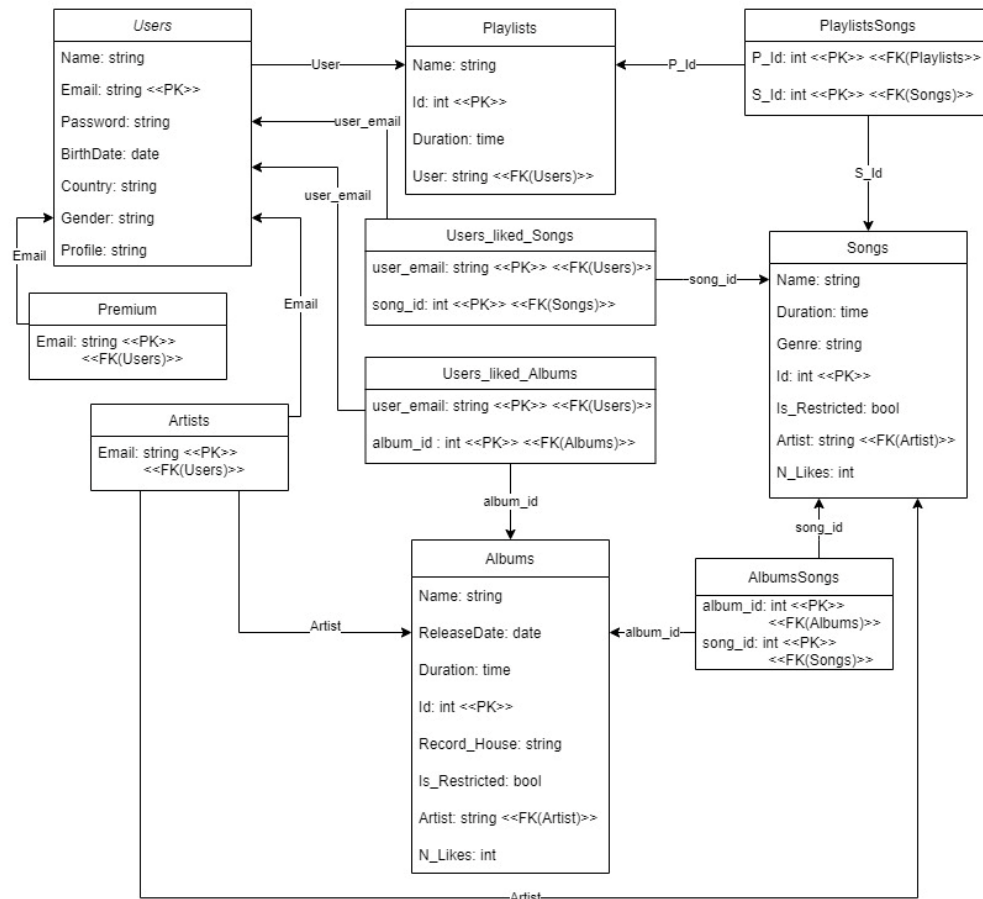


2. Progettazione logica

Per questo passaggio è necessario modificare le quattro relazioni molti a molti inserendo una tabella intermedia. Si sono andate così a creare altre quattro tabelle:

- PlaylistsSongs che rappresenta la relazione Playlists - (Contiene) - Songs
- Users_liked_Songs che rappresenta la relazione Users - (Salva) - Songs
- Users_liked_Albums che rappresenta la relazione Users - (Salva) - Albums
- AlbumsSongs che rappresenta la relazione Albums - (Contiene) - Songs

Per questioni di leggibilità abbiamo ommesso dallo schema grafico sia i vincoli NOT NULL presenti in ogni campo, sia altri vincoli esprimibili poi come check che verranno mostrati e commentati nell'apposito paragrafo.



5. Trigger e check

Per il mantenimento dell'integrità della base di dati e del funzionamento corretto dell'applicazione, abbiamo inserito molteplici check e trigger che ora analizzeremo:

- Check

```

CREATE TABLE "public"."Users" (
    "Email" varchar COLLATE "pg_catalog"."default" NOT NULL,
    "Name" varchar(20) COLLATE "pg_catalog"."default" NOT NULL,
    "BirthDate" date NOT NULL,
    "Country" varchar COLLATE "pg_catalog"."default" NOT NULL,
    "Gender" varchar COLLATE "pg_catalog"."default" NOT NULL,
    "Password" varchar COLLATE "pg_catalog"."default" NOT NULL,
    "Profile" varchar COLLATE "pg_catalog"."default" NOT NULL,
    "SubscribedDate" date NOT NULL,
    CONSTRAINT "Users_pkey" PRIMARY KEY ("Email"),
    CONSTRAINT "Users_BirtDate_check" CHECK ("BirthDate" > '1900-01-01'::date OR "BirthDate" < '2008-01-01'::date),
    CONSTRAINT "Users_Email_check" CHECK ("Email"::text ~~ '%@%':text),
    CONSTRAINT "Users_Gender_check" CHECK ("Gender"::text = 'M'::text OR "Gender"::text = 'F'::text)
);
    
```

- In questa tabella, i check servono per: imporre un limite alle età selezionabili in modo che sia compresa tra i 14 e i 122 anni, controllare che le email abbiano la forma adatta, imporre che il sesso di un utente sia M o F.

```
CREATE TABLE "public"."Albums" (
  "Name" varchar(10) COLLATE "pg_catalog"."default" NOT NULL,
  "ReleaseDate" date,
  "Duration" time(6),
  "Id" int4 NOT NULL DEFAULT nextval("Albums_Id_seq"::regclass),
  "Record_House" varchar COLLATE "pg_catalog"."default",
  "Is_Restricted" bool NOT NULL,
  "Artist" varchar COLLATE "pg_catalog"."default",
  "N_Likes" int4,
  CONSTRAINT "Albums_pkey" PRIMARY KEY ("Id"),
  CONSTRAINT "Albums_Artist_fkey" FOREIGN KEY ("Artist") REFERENCES "public"."Artists" ("Email") ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT "Albums_Duration_check" CHECK ("Duration" >= '00:00:00'::time without time zone AND "Duration" <= '01:30:00'::time without time zone),
  CONSTRAINT "Albums_N_Likes_check" CHECK ("N_Likes" >= 0)
);
```

- I constraint bloccano il numero di like di ogni album dal diventare negativo e ne limitano la durata tra 00:00:00 e 01:30:00

```
CREATE TABLE "public"."Songs" (
  "Name" varchar(10) COLLATE "pg_catalog"."default" NOT NULL,
  "Duration" time(6),
  "Genre" varchar COLLATE "pg_catalog"."default",
  "Id" int4 NOT NULL DEFAULT nextval("Songs_Id_seq"::regclass),
  "Is_Restricted" bool NOT NULL,
  "Artist" varchar COLLATE "pg_catalog"."default",
  "N_Likes" int4,
  CONSTRAINT "Songs_pkey" PRIMARY KEY ("Id"),
  CONSTRAINT "Songs_Artist_fkey" FOREIGN KEY ("Artist") REFERENCES "public"."Artists" ("Email") ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT "Songs_Duration_check" CHECK ("Duration" >= '00:01:00'::time without time zone AND "Duration" <= '00:30:00'::time without time zone),
  CONSTRAINT "Songs_N_Likes_check" CHECK ("N_Likes" >= 0)
);
```

- I limiti funzionano allo stesso modo di quelli sulla tabella Albums, solo che la durata massima è di mezz'ora

- Trigger

```
CREATE TRIGGER "no_same_name_album" BEFORE INSERT OR UPDATE ON "public"."Albums"
FOR EACH ROW
EXECUTE PROCEDURE "public"."no_same_name_alb"();
CREATE OR REPLACE FUNCTION "public"."no_same_name_alb"()
RETURNS "pg_catalog"."trigger" AS $BODY$BEGIN
  IF (NEW."Name" IN (SELECT a."Name"
                    FROM "public"."Albums" AS a
                    WHERE a."Artist" = NEW."Artist" and a."Id" != NEW."Id")) THEN
    RAISE EXCEPTION 'Album con questo nome già presente';
  END IF;

  RETURN NEW;
END$BODY$
LANGUAGE plpgsql;
```

- no_same_name_album è un trigger che impedisce ad un artista di creare più album con lo stesso nome

```
CREATE TRIGGER "no_same_name_song" BEFORE INSERT OR UPDATE ON "public"."Songs"
FOR EACH ROW
EXECUTE PROCEDURE "public"."no_same_name_sng"();

CREATE OR REPLACE FUNCTION "public"."no_same_name_sng"()
RETURNS "pg_catalog"."trigger" AS $BODY$BEGIN
    IF (NEW."Name" IN (SELECT s."Name"
                        FROM "public"."Songs" AS s
                        WHERE s."Artist" = NEW."Artist" and s."Id" != NEW."Id")) THEN
        RAISE EXCEPTION 'Brano con questo nome già presente';
    END IF;

    RETURN NEW;
END$BODY$
LANGUAGE plpgsql;
```

- no_same_name_sng è un trigger che impedisce ad un artista di pubblicare più canzoni con lo stesso nome

```
CREATE TRIGGER "no_same_name_playlist" BEFORE INSERT OR UPDATE ON "public"."Playlists"
FOR EACH ROW
EXECUTE PROCEDURE "public"."no_same_name_pl"();

CREATE OR REPLACE FUNCTION "public"."no_same_name_pl"()
RETURNS "pg_catalog"."trigger" AS $BODY$BEGIN
    IF (NEW."Name" IN (SELECT p."Name"
                        FROM "public"."Playlists" AS p
                        WHERE p."User" = NEW."User" and p."Id" != NEW."Id")) THEN
        RAISE EXCEPTION 'Playlist con questo nome già presente';
    END IF;

    RETURN NEW;
END$BODY$
LANGUAGE plpgsql;
```

- no_same_name_pl è un trigger che impedisce ad un utente qualsiasi di creare più playlist con lo stesso nome

6. Principali scelte progettuali

Completata l'analisi di ciò che riguarda direttamente la creazione delle tabelle, passiamo ora ad affrontare altri aspetti relativi alla base di dati nella sua interezza.

1. Indici

Per quanto riguarda la creazione di indici e/o viste materializzate, abbiamo scelto di affidarci a degli indici realizzati sulle chiavi primarie delle principali tabelle. Questo è

stato un argomento discusso perché la nostra applicazione richiede un frequente accesso al database in modifica, aggiunta o rimozione. Infatti, l'utente può creare canzoni, album, playlist e modificarle o cancellarle liberamente, per cui l'utilizzo di indici poteva rivelarsi controproducente perché potenzialmente per ogni azione di modifica devono essere aggiornati. Tuttavia, abbiamo comunque deciso di implementarli legandoli alle sole chiavi primarie allo scopo di mitigare il problema dell'aggiornamento, poiché così facendo l'indice viene modificato solo con le operazioni di aggiunta e rimozione. Abbiamo quindi pensato che la situazione ottimale potesse essere raggiungibile unendo tali indici alla possibilità del DBMS di scegliere autonomamente se e quando usare determinati indici/viste.

2. Ruoli

I ruoli creati sono abbastanza auto esplicativi, ma ci soffermeremo comunque ad analizzarli. Così come i tipi di account disponibili all'utente sono tre, anche i ruoli creati (fatta eccezione dell'admin) sono tre:

- free

È il ruolo con il minor numero di libertà.

SELECT	UPDATE	INSERT	DELETE
Users, Artists, Playlists, Songs, Albums, PlaylistsSongs, Users_liked_Songs, Users_liked_Albums	Users, Playlists	Playlists, PlaylistsSongs, Users_liked_Songs, Users_liked_Albums	Playlists, Users_liked_Songs, Users_liked_Albums

- premium

Da un punto di vista di permessi, è uguale al free. In pratica però ha accesso anche alle canzoni con visibilità ristretta.

- artist

Ha più libertà rispetto i due ruoli precedenti, ma comunque meno di un admin.

SELECT	UPDATE	INSERT	DELETE
Users, Artists, Playlists, Songs, Albums, PlaylistsSongs, Users_liked_Songs, Users_liked_Albums, AlbumsSongs	Artists, Playlists, Songs, Albums	Playlists, Songs, Albums, PlaylistsSongs, Users_liked_Songs, Users_liked_Albums, AlbumsSongs	Playlists, Songs, Albums, Users_liked_Songs, Users_liked_Albums, AlbumsSongs

A nessun ruolo è concessa la GRANT OPTION.

7. Query principali

La prima query degna di nota non lo è tanto per la sua complessità quanto per la frequenza con la quale appare nel codice, poiché è necessaria per visualizzare sempre tutte le playlist dell'utente nel menù a sinistra.

```
playlists=session.query(Playlists).filter(Playlists.User==current_user.Email)
```

Un'altra interrogazione al database che appare spesso è quella per richiedere o tutte le canzoni che piacciono ad un utente, o tutte le canzoni che piacciono di un artista. Per esempio, la prima versione viene utilizzata per mostrare le canzoni salvate nella sezione apposita in libreria, mentre la seconda per raccogliere i dati necessari per le statistiche. Query simili vengono effettuate per cercare gli album.

1. **liked_songs**=session.query(Songs.Id).filter(Songs.Id.in_(session.query(Users_liked_Songs.song_id).filter(Users_liked_Songs.user_email==current_user.Email)))
2. **all_liked_songs**=session.query(Users_liked_Songs.song_id)
my_liked_songs=session.query(Songs.Id).filter(and_(Songs.Artist==current_user.Email, Songs.Id.in_(all_liked_songs)))

Altre query ricorrenti sono quelle per recuperare tutte le canzoni o gli album di un artista, usate per esempio per mostrare la lista di canzoni che un artista può aggiungere ad un album.

```
all_my_songs=session.query(Songs.Id).filter(Songs.Artist==current_user.Email)
```

Queste sono le query principali che compongono la maggior parte degli accessi al database, quindi ora per completezza inseriremo una delle richieste più complesse utilizzate per l'ottenimento di informazioni utili a mostrare ad un artista delle statistiche riguardo i suoi ascoltatori. Più nel dettaglio, tramite questa serie di comandi vengono estratti tutti i paesi dai quali essi provengono e, per ognuno di essi, il numero di ascoltatori che ci vivono.

```
all_liked_songs=session.query(Users_liked_Songs.song_id)  
my_liked_songs=session.query(Songs.Id).filter(and_(Songs.Artist==current_user.  
.Email, Songs.Id.in_(all_liked_songs)))  
users_like_songs=session.query(Users.Email).filter(Users.Email.in_(session.qu  
ery(Users_liked_Songs.user_email).filter(Users_liked_Songs.song_id.in_(my_lik  
ed_songs))))
```

```
all_liked_albums=session.query(Users_liked_Albums.album_id)  
my_liked_albums=session.query(Albums.Id).filter(and_(Albums.Artist==current_u  
ser.Email, Albums.Id.in_(all_liked_albums)))  
users_like_albums=session.query(Users.Email).filter(Users.Email.in_(session.q  
uery(Users_liked_Albums.user_email).filter(Users_liked_Albums.album_id.in_(my  
_liked_albums))))
```

```
countries=session.query(Users.Country, func.count(Users.Email)).filter(or_(Use  
rs.Email.in_(users_like_songs), Users.Email.in_(users_like_albums))).group_by(  
Users.Country).all()
```

8. Ulteriori informazioni

Il progetto è stato realizzato con Flask mediante l'utilizzo di Blueprint. Questa scelta è stata compiuta per motivi di pulizia ed organizzazione dell'app. Tra le librerie di Flask importate, oltre i Blueprint, ci sono: flask_login per tutto ciò che riguarda il login e la gestione degli accessi e logout, flask_wtf per i form ed inserire tutte le informazioni (per esempio nella creazione utente, nell'upload delle canzoni o simili), flask_bcrypt per criptare le password,

sqlescapy per sanitizzare gli input dei form ed evitare che vengano eseguiti comandi SQL indesiderati.

Sempre legata a SQL è la scelta dell'ORM SQLAlchemy, usato come interfaccia per accedere facilmente al database PostgreSQL da semplici comandi Python.

Per quanto riguarda invece il front-end, nel progetto sono stati utilizzati quasi esclusivamente HTML e CSS, dove nel primo abbiamo fatto largo uso del sistema di template, mentre per il secondo ci siamo affidati a W3School e ad un nostro file CSS per definire lo stile globale del sito. Ridotto l'apporto di Javascript, che è servito principalmente (unito ad AJAX) per mostrare i grafici realizzati con Google Chart nella schermata "Statistiche" degli utenti Artisti.

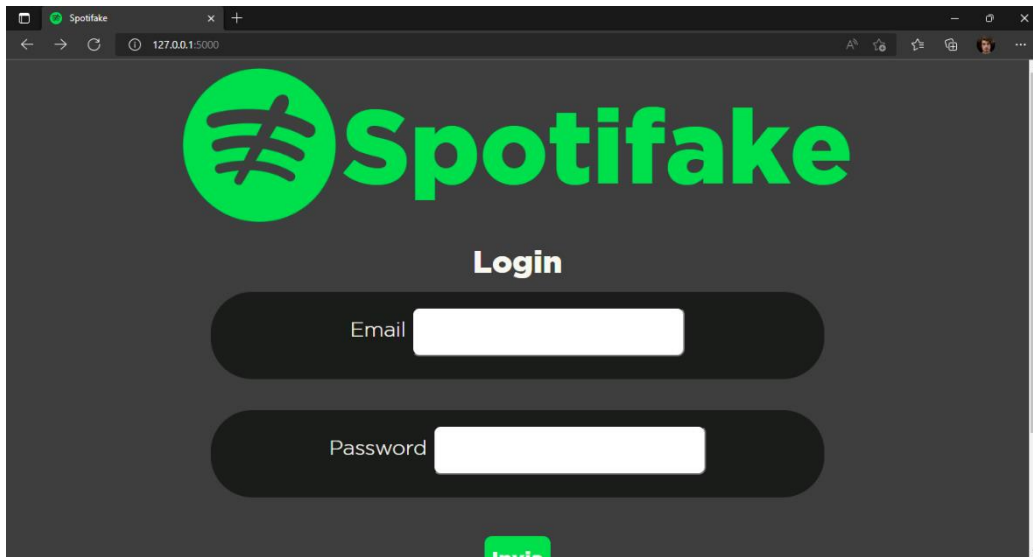
Dal punto di vista della sicurezza, tre sono le precauzioni adottate. Innanzitutto, le password vengono sempre crittate tramite flask_bcrypt in modo da evitare che possano essere lette da intrusioni esterne. Il problema della SQL injection invece viene affrontato in due modi: da un lato c'è SQLAlchemy che come ORM opera in modo da contrastare tale debolezza, dall'altro c'è sqlescapy che sanitizza tutti gli input non controllati che l'utente può inserire, per esempio come nome di un album o come nome utente. Inoltre, abbiamo inserito in ogni route dei controlli per evitare che scrivendo a mano gli URL specifici delle varie pagine vi si possa accedere anche se non autorizzati, in modo da garantire il corretto funzionamento dell'app. Infine, per risolvere il problema del Cross Site Scripting, utilizziamo il sistema di template di Flask che tra i suoi vari vantaggi ha anche questo.

Come spiegato in precedenza, esiste un sistema di "Mi piace" tramite il quale un utente può esprimere delle preferenze e salvarsi brani o album in una sezione apposita della libreria.

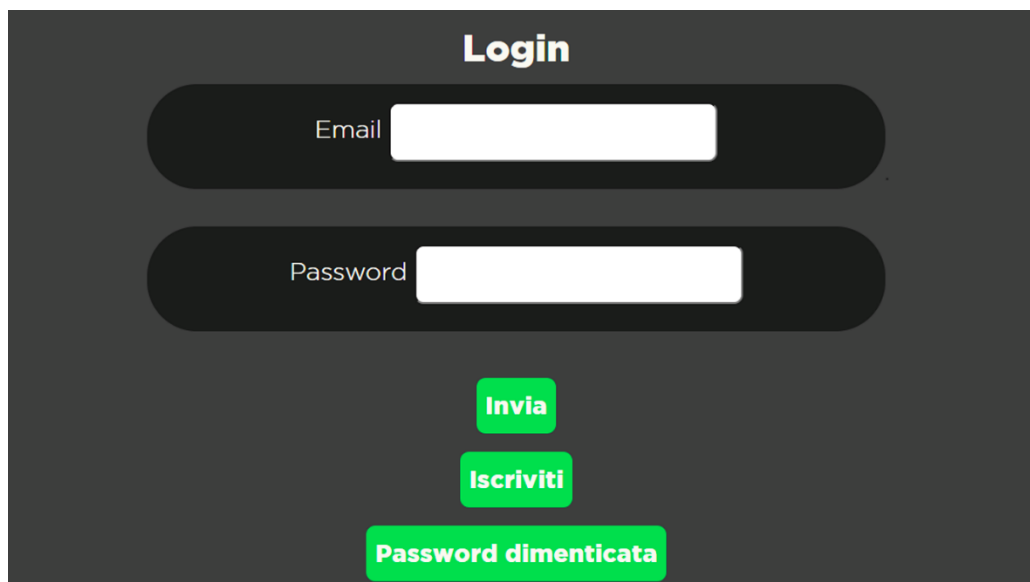
Tra le varie caratteristiche dell'applicazione, due sono degne di nota. La prima è già stata spiegata precedentemente e consiste nella politica di visibilità dei brani o degli album, per la quale se un utente è "Free" non può accedere ai contenuti segnati come ristretti, mentre un utente "Premium" può. La seconda caratteristica è l'algoritmo per raccomandare dei nuovi brani (mostrati nella home) ad un utente in base a ciò che ascolta. Il funzionamento è il seguente: essendo impossibile riprodurre realmente le canzoni, abbiamo considerato i "Mi Piace" come se fossero degli ascolti. Di conseguenza, se un utente non ha mai messo mi piace a nessun brano, nella home gli verranno consigliati in ordine i 5 brani con più likes. Invece, nel caso in cui abbia effettivamente mostrato preferenze, viene eseguita una query che ritornerà una lista dei generi che ascolta ordinati in modo decrescente in base al numero di brani salvati per ognuno di essi. Essendo tale ordine decrescente, i primi generi saranno quelli con più canzoni salvate, per cui verranno consigliate 5 canzoni seguendo sia l'ordine dei generi, sia un ordine interno per numero di likes.

Infine, è presente un algoritmo che declassa un Artista ad utente Free se per un certo numero di giorni non pubblica né canzoni né album e sfrutta l'account Artist per evitare le limitazioni di quello base. Per semplicità e non potendo effettuare dei reali pagamenti, è possibile cambiare il tipo di account semplicemente nella sezione di modifica del profilo personale.


9. Screenshots ed immagini



1. Schermata iniziale, login



2. Schermata iniziale, login



Spotifake

Sign up

Nome

Indica un nome utente

Email

3. Iscrizione pt.1

Password

Scegli una password

Profilo Free

Scegli con che tipo di account vuoi registrarti

Sesso M

Indica il tuo sesso

4. Iscrizione pt.2

Indica il tuo sesso

Paese Afghanistan

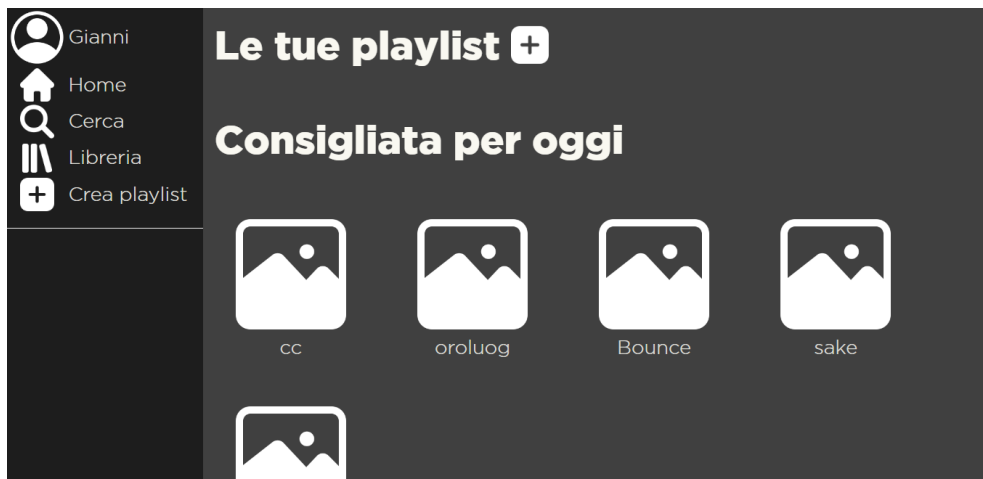
Seleziona una nazione

La tua data di nascita gg/mm/aaaa

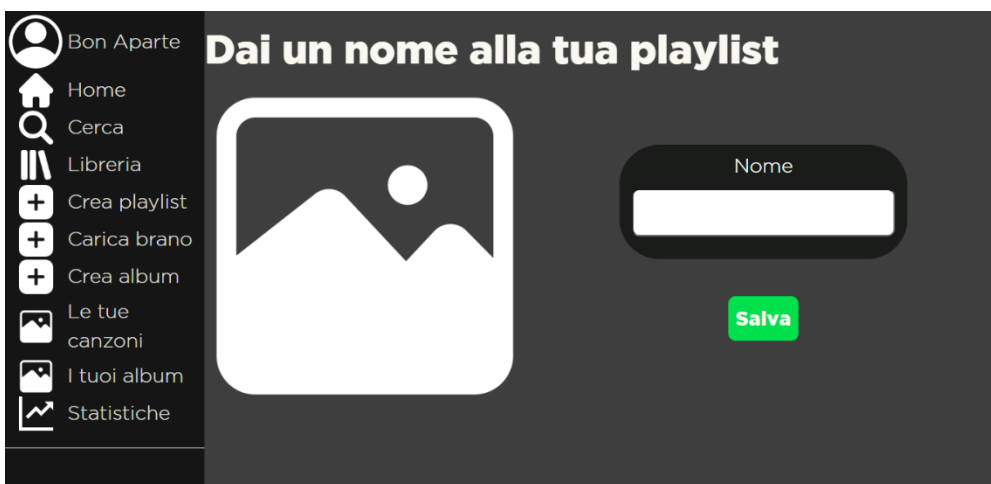
Indica la tua data di nascita

Invia

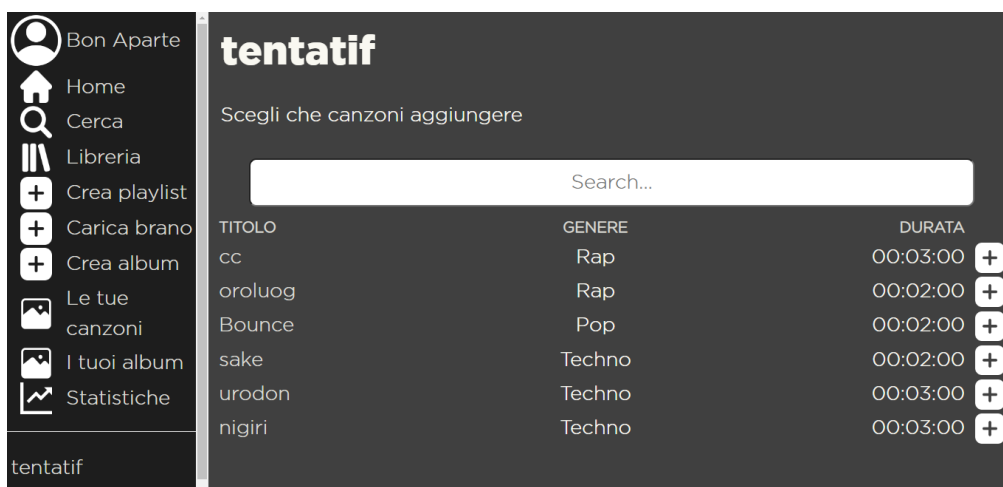
5. Iscrizione pt.3



6. Home, senza playlist create e con canzoni raccomandate



7. Creazione playlist



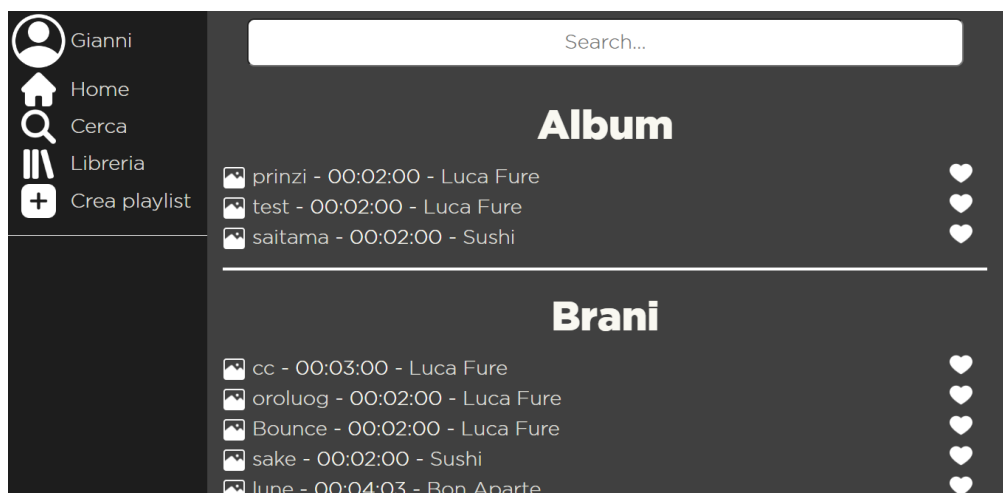
8. Aggiunta brani alla playlist



9. Utilizzo barra di ricerca



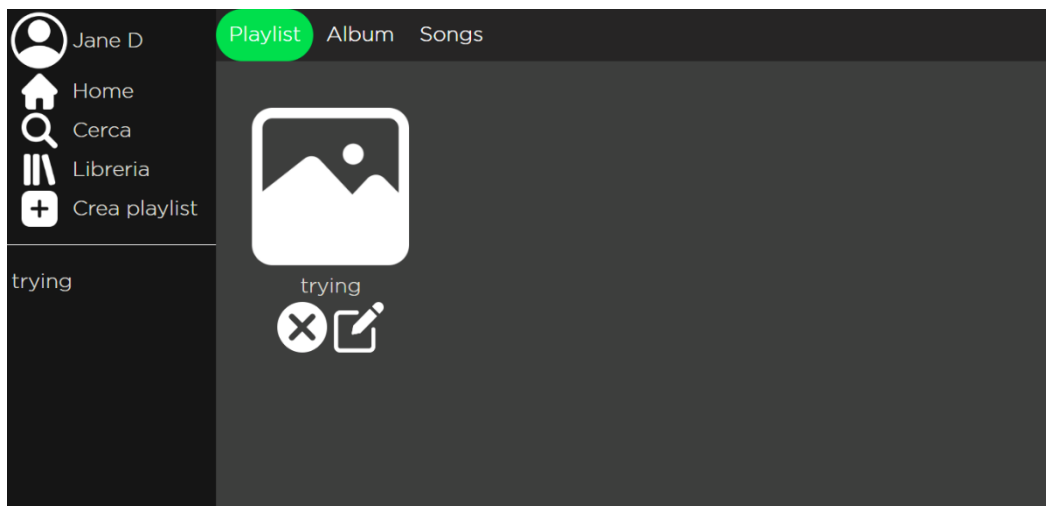
10. Visualizzazione playlist



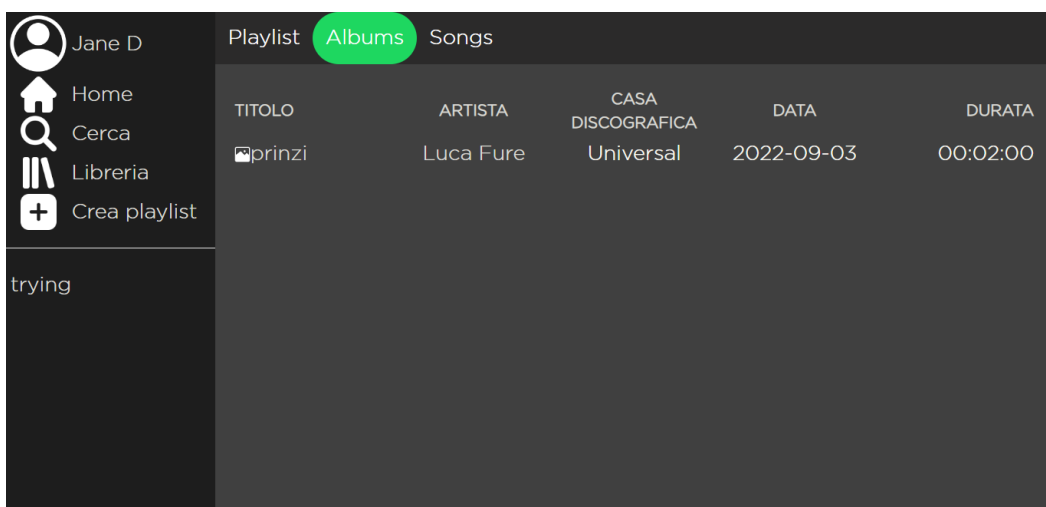
11. Funzione "Cerca" pt.1



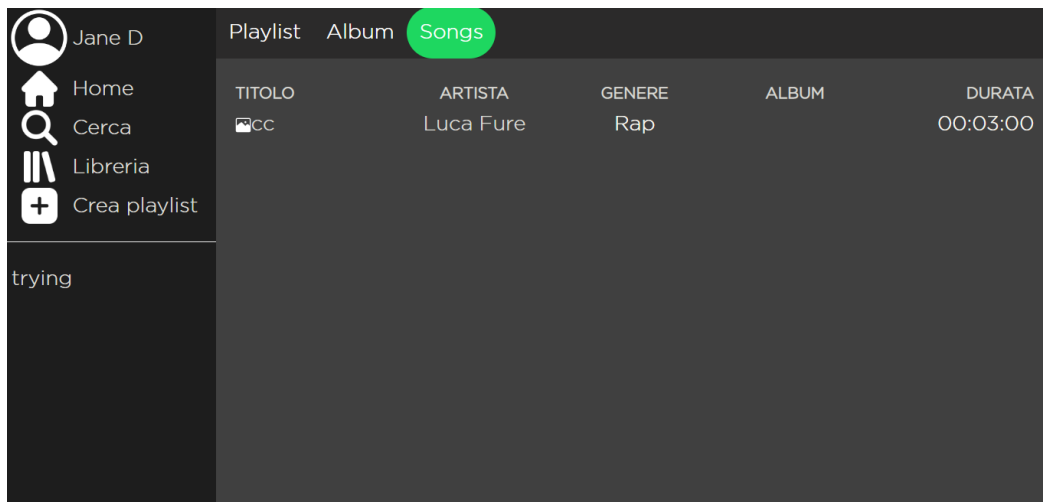
12. Funzione "Cerca" pt.2



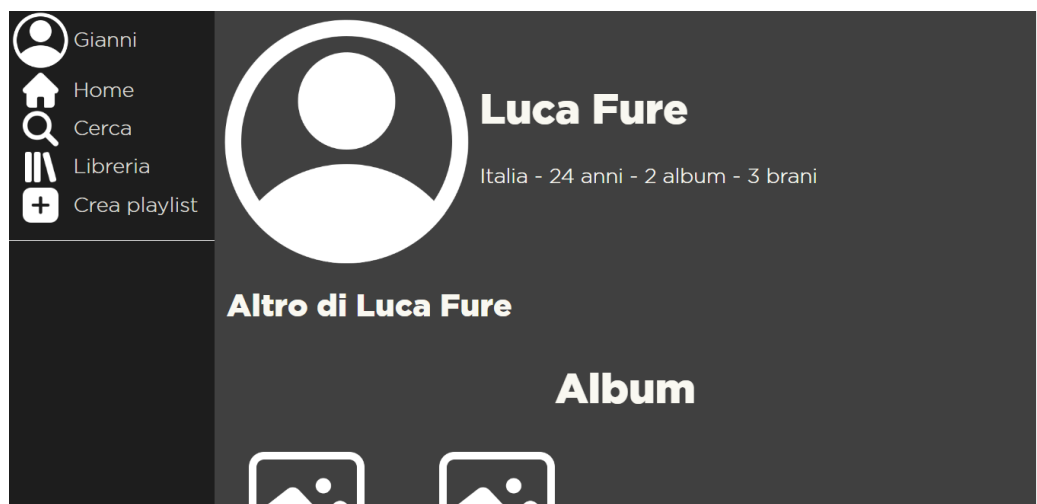
13. Visualizzazione libreria, sezione playlist



14. Visualizzazione libreria, sezione album



15. Visualizzazione libreria, sezione songs



16. Visualizzazione profilo di un artista



17. Visualizzazione album

Luca Fure

Home

Cerca

Libreria

Crea playlist

Carica brano

Crea album

Le tue canzoni

I tuoi album

Statistiche

Condividi il tuo brano con il mondo

Nome

Bounce

Durata 00:02:00

Genere

Pop

Visibilità Free

18. Creazione canzone

Luca Fure

Home

Cerca

Libreria

Crea playlist

Carica brano

Crea album

Le tue canzoni

I tuoi album

Statistiche

Condividi il tuo album con il mondo

Nome

test

Casa discografica

Universal

Visibilità Free

Salva

19. Creazione album

Luca Fure

Home

Cerca

Libreria

Crea playlist

Carica brano

Crea album

Le tue canzoni

I tuoi album

Statistiche

Il tuo profilo

LOGOUT

Luca Fure

Modifica profilo

Cancella profilo

20. Visualizzazione profilo personale

Il tuo profilo

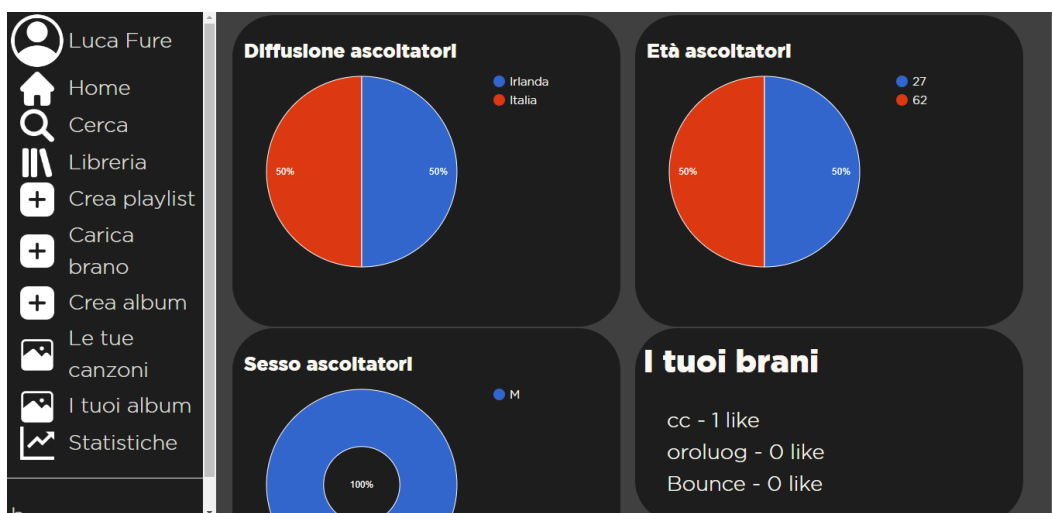
Aggiorna il tuo profilo

Nome

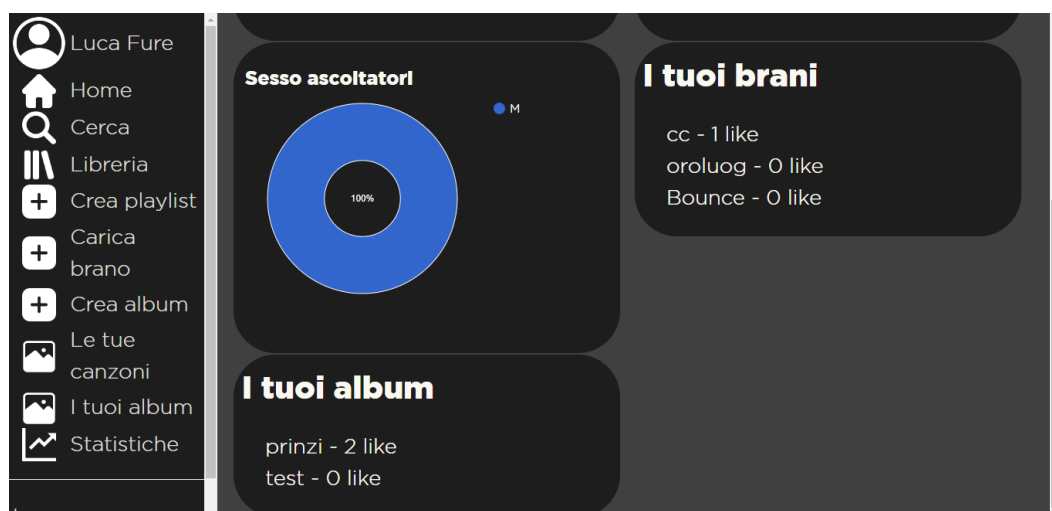
Profilo

Invia

21. Modifica informazioni profilo personale



22. Visualizzazione schermata con statistiche pt.1



23. Visualizzazione schermata con statistiche pt.2

Le schermate per la modifica di brani, album o playlist sono uguali a quelle relative alla loro creazione

La suddivisione dei lavori è stata la seguente:

LAVORO	RESPONSABILE
Progettazione concettuale e logica	Furegon Luca 50%, Pistellato Martino 50%
Sviluppo back-end	Furegon Luca 70%, Pistellato Martino 30%
Sviluppo front-end	Furegon Luca 30%, Pistellato Martino 70%
Revisione, commenti e documentazione	Furegon Luca 50%, Pistellato Martino 50%