

# PROGETTO BASI DATI

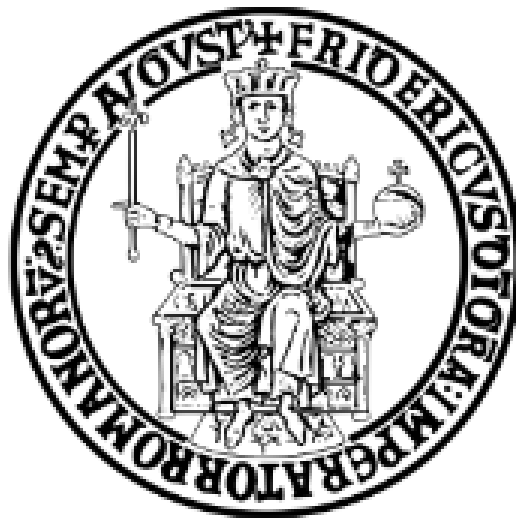
## Documentazione:

Traccia 1:

Sistema di gestione di una libreria musicale di un gruppo di utenti.

Dario Morace N86003778 Mattia Marucci N86003853

9 Marzo



Pagina volutamente bianca.

# Indice

<b>1</b>	<b>Descrizione del progetto</b>	<b>4</b>
1.1	Descrizione della traccia e analisi del problema . . . . .	4
<b>2</b>	<b>Progettazione concettuale</b>	<b>5</b>
2.1	Introduzione . . . . .	5
2.2	Class Diagram . . . . .	6
2.3	Ristrutturazione del Class Diagram . . . . .	6
2.4	Analisi degli identificativi . . . . .	6
2.5	Rimozione delle gerarchie di specializzazione . . . . .	7
2.6	Class Diagram ristrutturato . . . . .	7
2.7	Dizionario delle classi . . . . .	8
<b>3</b>	<b>Progettazione Logica</b>	<b>10</b>
3.1	Schema Logico . . . . .	10
<b>4</b>	<b>Progettazione Fisica</b>	<b>10</b>
4.1	Definizione Tabelle . . . . .	10
4.1.1	Definizione tabella Utente . . . . .	10
4.1.2	Definizione tabella Libreria . . . . .	11
4.1.3	Definizione tabella Playlist . . . . .	11
4.1.4	Definizione tabella Traccia . . . . .	11
4.1.5	Definizione tabella Aggiungi . . . . .	12
4.1.6	Definizione tabella Ascolto . . . . .	12
4.1.7	Definizione tabella Artista . . . . .	12
4.1.8	Definizione tabella Album . . . . .	13
4.1.9	Definizione tabella Collab . . . . .	13
4.2	Definizione Trigger e Funzioni . . . . .	13
4.2.1	Definizione trigger creazione Libreria . . . . .	13
4.2.2	Definizione trigger update Libreria . . . . .	14
4.2.3	Definizione trigger per l'incremento/decremento della grandezza di una Playlist	14
4.2.4	Definizione trigger partecipazione Artisti di una Traccia . . . . .	15
4.3	Sequenze utilizzate . . . . .	16
4.3.1	Sequenze . . . . .	16
<b>5</b>	<b>Esempi d'uso</b>	<b>17</b>
5.1	Log-in . . . . .	17
5.2	Registrazione . . . . .	18
5.3	Pannello Home . . . . .	19
5.4	Pannello Libreria . . . . .	20
5.5	Pannello Ricerca . . . . .	21
5.6	Pannello Info . . . . .	22
<b>6</b>	<b>Conclusioni</b>	<b>23</b>
6.1	Repository GitHub utilizzata . . . . .	23

# 1 Descrizione del progetto

## Breve descrizione del progetto e preview di risoluzione

### 1.1 Descrizione della traccia e analisi del problema

Si è progettato ed implementato un database in PostgreSQL tramite interfaccia pgAdmin4 per la gestione di un applicativo che permette la creazione di una libreria musicale da parte di vari utenti dovendo tenere conto di differenziare il loro ruolo all'interno di quest'ultimo. Si provvederà quindi in primis alla progettazione di uno schema delle classi che definisca il problema per passare poi alla sua ristrutturazione per poter poi ottenere uno schema logico con cui sarà, per finire, possibile implementare il DataBase finale in PostgreSQL che affiancherà il nostro applicativo. Nell'applicativo sono quindi stati implementati vari pannelli che permettono il display di: un pannello Home dove è possibile "esplorare" tutto quello che è presente nel DataBase tramite varie richieste, un pannello Libreria per utente che contiene varie playlist che possono essere gestite(create, eliminate, rese preferite) e a loro volta conterranno varie tracce anche loro possono essere aggiunte, rimosse o ascoltate, un pannello Search che permette la ricerca di tracce con vari parametri(da qui possiamo poi aggiungerle ad una playlist o ascoltarle) ed un pannello Info che permette solo agli admin di recuperare informazioni riguardo gli utenti, le fasce orarie in cui usano l'applicativo e precisamente i loro ascolti.

## 2 Progettazione concettuale

Tutto il materiale presente in questa documentazione sotto forma di immagine è presente e reperibile in formato originale all'interno della repository GitHub linkata alla fine della documentazione.

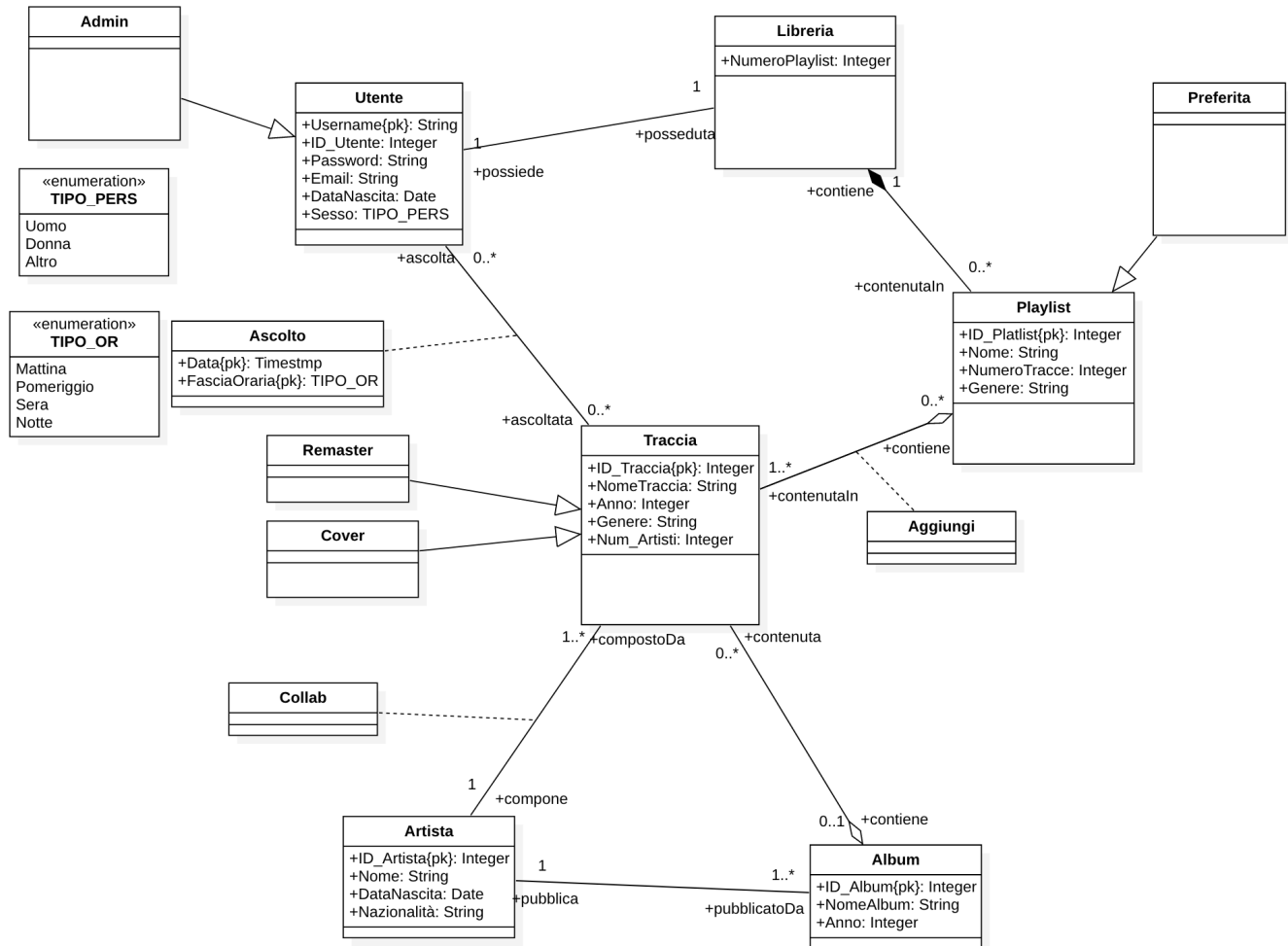
### 2.1 Introduzione

In questo capitolo inizia la progettazione della base di dati. Dal risultato dell'analisi dei requisiti che devono essere soddisfatti, si arriverà ad uno schema concettuale indipendente dalla struttura dei dati e dall'implementazione fisica: in tale schema concettuale, che verrà rappresentato usando un Class Diagram UML, si evidenzieranno le entità rilevanti ai fini della rappresentazione dei dati e le relazioni che intercorrono tra esse; si delineeranno anche eventuali vincoli da imporre. Al fine di semplificare la lettura dei Class Diagram UML che seguono, si è scelto di adottare le seguenti convenzioni:

- La molteplicità di una associazione è situata a destra se la linea di collegamento è verticale, viceversa in alto se la linea di collegamento è orizzontale.
- Alcuni nomi di associazioni e ruoli sono stati omessi. La descrizione completa di questi è comunque riportata in seguito nel dizionario delle associazioni.

## 2.2 Class Diagram

Di seguito il modello concettuale NON ristrutturato:



## 2.3 Ristrutturazione del Class Diagram

Al fine di rendere il Class Diagram idoneo alla traduzione in schema relazionale e di migliorare l'efficienza dell'implementazione si procede alla ristrutturazione dello stesso. Al termine del procedimento il Class Diagram non conterranno attributi strutturati, attributi multipli e gerarchie di specializzazione.

## 2.4 Analisi degli identificativi

Risulta conveniente ai fini dell'efficienza l'introduzione di chiavi "tecniche" nella quasi totalità delle entità. Tali chiavi altro non saranno che identificativi interi e/o varchar che permetteranno una migliore differenziazione dei contenuti e di poter prelevare/inserire/modificare i dati nel database con molta più affidabilità e sicurezza. Difatti troviamo chiavi tecniche molto importanti in svariate entità come ad esempio nella tabella Utente troviamo l'ID Utente che una volta assegnato

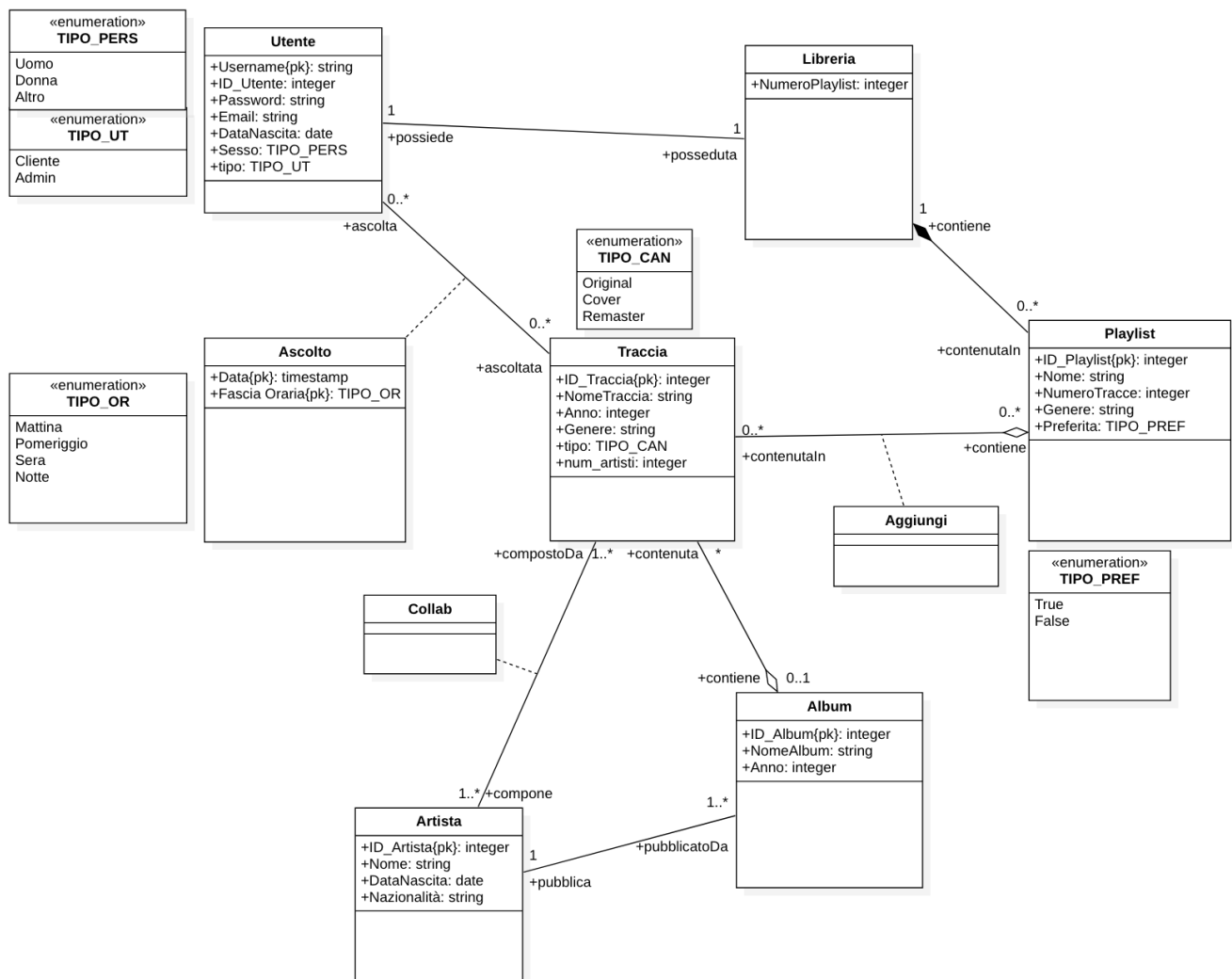
utilizzeremo per la creazione della libreria musicale associata a quell'utente o nella tabella Playlist troviamo l'ID Playlist che utilizzeremo per riferirci ad una determinata playlist quando effettuiamo un inserimento/ eliminazione/modifica di una canzone in essa contenuta.

## 2.5 Rimozione delle gerarchie di specializzazione

All'interno del modello concettuale non ristrutturato sono presenti varie generalizzazioni poi rimosse. Queste sono:

- Admin, rimodellato utilizzando una enumeration chiamata tipo utente.
- Remaster e Cover, rimodellate utilizzando una enumeration chiamata tipo canzone.
- Preferita, rimodellata utilizzando un enumeration chiamata tipo preferita.

## 2.6 Class Diagram ristrutturato



## 2.7 Dizionario delle classi

Tabella	Descrizione	Attributi
<b>Utente</b>	L'insieme degli oggetti rappresenta la somma dei clienti che utilizzano l'applicativo e degli admin che invece la gestiscono.	<b>Username(String)</b> PK: Nome utente scelto in fase di registrazione. <b>IDUtente(integer)</b> : Id univoco di un utente automaticamente assegnato. <b>Password(String)</b> : Password scelta dall'utente in fase di registrazione. <b>Email(String)</b> : Email inserita in fase di registrazione. <b>DataNascita(date)</b> : Data di nascita inserita in fase di registrazione. <b>Sesso(enumeration)</b> : Sesso della persona selezionabile fra uomo, donna o altro. <b>tipo(enumeration)</b> : Identifica il tipo di utente. Cliente o Admin.
<b>Libreria</b>	L'insieme degli oggetti rappresenta la relativa Libreria associata ad ogni utente.	<b>NumeroPlaylist(Integer)</b> : Contiene il numero di playlist possedute da un determinato utente.
<b>Playlist</b>	L'insieme degli oggetti rappresenta la somma di tutte le Playlist create da tutti gli utenti.	<b>IDPlaylist(Integer)</b> PK: Id univoco di una playlist di un determinato utente. <b>Nome(String)</b> : Nome della playlist scelto dall'utente. <b>NumeroTracce(Integer)</b> : Numero di tracce contenute in una determinata Playlist. <b>Genere(String)</b> : Genere della playlist, non obbligatorio. <b>Preferita(enumeration)</b> : Enumeration che esprime se una determinata playlist è la preferita di un determinato utente.



<b>Traccia</b>	L'insieme degli oggetti rappresenta tutte le tracce che l'applicativo contiene e può quindi gestire.	<b>IDTraccia</b> (Integer) PK: ID univoco di una determinata traccia. <b>NomeTraccia</b> (String): Nome della traccia. <b>Anno</b> (Integer): Anno di uscita della traccia in questione. <b>Genere</b> (String): Genere della traccia in questione. <b>tipo</b> (enumeration): Enumeration che esprime se la canzone è originale, una cover o una remaster. <b>numArtisti</b> (Integer): Esprime quanti artisti hanno collaborato alla suddetta canzone.
<b>Album</b>	L'insieme degli oggetti rappresenta tutti gli album che l'applicativo contiene e può quindi gestire.	<b>IDAlbum</b> (Integer): Codice univoco di Album. <b>NomeAlbum</b> (String): Nome dell'album in questione. <b>Anno</b> (Integer): Anno di uscita di un determinato Album.
<b>Artista</b>	L'insieme degli oggetti rappresenta tutti gli artisti che l'applicativo contiene e può quindi gestire.	<b>IDArtista</b> (Integer) PK: ID univoco di un artista. <b>Nome</b> (String): Nome dell'artista. <b>DataNascita</b> (date): Data di nascita dell'artista. <b>Nazionalità</b> (String): Nazionalità dell'artista in questione.
<b>Collab</b>	L'insieme degli oggetti rappresenta la collaborazione fra determinati artisti.	
<b>Aggiungi</b>	L'insieme degli oggetti rappresenta l'aggiunta di una determinata traccia in una determinata playlist di un utente.	
<b>Ascolto</b>	L'insieme degli oggetti rappresenta la somma degli ascolti effettuati dagli utenti che utilizzano l'applicativo.	<b>Data</b> (timestamp) PK: Data e orario dell'ascolto effettuato da un utente. <b>FasciaOraria</b> (enumeration) PK: enumeration che esprime la fascia oraria di ascolto che può essere Mattina, Pomeriggio, Sera, Notte.

## 3 Progettazione Logica

In questo capitolo sarà trattata la fase successiva della progettazione della base di dati. Si tradurrà lo schema concettuale (già predisposto in seguito alla ristrutturazione) in uno schema logico. Negli schemi relazionali che seguiranno le chiavi primarie sono indicate in grassetto mentre le chiavi esterne con una sottolineatura.

### 3.1 Schema Logico

- UTENTE(**USERNAME**, PASSWORD, **ID\_UTENTE**, EMAIL, DATANASCITA, TIPO\_PERS, TIPO\_UT)
- LIBRERIA(**ID\_LIBRERIA**, NUMEROPLAYLIST)
- PLAYLIST(**ID\_PLAYLIST**, ID\_LIBAPPARTENENZA, NOME, NUMEROTRACCE, GENERE, TIPO\_PREF)
- TRACCIA(**ID\_TRACCIA**, ID\_ALBUM, NOMETRACCIA, ANNO, GENERE, TIPO\_CAN, NUM\_ARTISTI)
- ALBUM(**ID\_ALBUM**, ID\_ARTISTA, NOMEALBUM, ANNO)
- ARTISTA(**ID\_ARTISTA**, NOME, DATANASCITA, NAZIONALITA')
- AGGIUNGI(ID\_PLAYLIST, ID\_TRACCIA)
- COLLAB(ID\_ARTISTA, ID\_TRACCIA)
- ASCOLTO(**ORARIO**, **FASCIAORARIA**, ID\_UTENTE, ID\_TRACCIA)

## 4 Progettazione Fisica

Di seguito è possibile visionare le dichiarazioni delle tabelle, trigger e function realizzate. Per implementare alcuni vincoli più complessi con i trigger in PostgreSQL, si devono implementare Function che restituiscono un Trigger. Non vi è stato il bisogno di utilizzare viste.

### 4.1 Definizione Tabelle

Di seguito è possibile trovare le definizioni delle tabelle create all'interno di PostgreSQL.

#### 4.1.1 Definizione tabella Utente

```
--TABELLA UTENTE
CREATE TABLE Utente
(
    username character varying(20) NOT NULL,
    password character varying(20) NOT NULL,
    id_utente integer NOT NULL DEFAULT nextval('utente_id_utente_seq'::regclass),
    email character varying(30) NOT NULL,
    datanascita date NOT NULL,
    sesso character varying(20),
    tipo_ut character varying(10) DEFAULT 'Cliente'::character varying,
    CONSTRAINT utente_pkey PRIMARY KEY (username),
    CONSTRAINT email_unique UNIQUE (email),
    CONSTRAINT id_utente_unique UNIQUE (id_utente),
    CONSTRAINT tipo_utente CHECK (tipo_ut::text = 'Cliente'::text OR
                                tipo_ut::text = 'Admin'::text),
    CONSTRAINT tipo_pers CHECK (sesso::text = 'Uomo'::text OR
                               sesso::text = 'Donna'::text OR sesso::text = 'Altro'::text)
)
```

#### 4.1.2 Definizione tabella Libreria

```
--TABELLA LIBRERIA
CREATE TABLE Libreria
(
    id_libreria integer NOT NULL,
    num_playlist integer,
    CONSTRAINT id_libreria_unique PRIMARY KEY (id_libreria),
    CONSTRAINT libreria_fkey FOREIGN KEY (id_libreria)
        REFERENCES public.Utente (id_utente)
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)
```

#### 4.1.3 Definizione tabella Playlist

```
--TABELLA PLAYLIST
CREATE TABLE Playlist
(
    id_playlist integer NOT NULL DEFAULT nextval('playlist_id_playlist_seq'::regclass),
    id_libappartenenza integer NOT NULL,
    nome character varying(20) NOT NULL,
    numerotracce integer NOT NULL DEFAULT 0,
    genere character varying(20),
    preferita character varying(6) NOT NULL DEFAULT false,
    CONSTRAINT playlist_pkey PRIMARY KEY (id_playlist),
    CONSTRAINT playlist_fkey FOREIGN KEY (id_libappartenenza)
        REFERENCES public.libreria (id_libreria)
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT tipo_pref CHECK (preferita::text = 'true'::text OR
                                preferita::text = 'false'::text)
)
```

#### 4.1.4 Definizione tabella Traccia

```
--TABELLA TRACCIA
CREATE TABLE Traccia
(
    id_traccia integer NOT NULL DEFAULT nextval('"Traccia_id_traccia_seq"'::regclass),
    nometraccia character varying(50) NOT NULL,
    anno integer NOT NULL,
    genere character varying(20) NOT NULL,
    tipo_can character varying(20) NOT NULL,
    id_album integer,
    num_artisti integer DEFAULT 0,
    CONSTRAINT traccia_pkey PRIMARY KEY (id_traccia),
    CONSTRAINT traccia_fk FOREIGN KEY (id_album)
        REFERENCES public.album (id_album)
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT tipo_traccia CHECK (tipo_can::text = 'Original'::text OR
                                    tipo_can::text = 'Cover'::text OR
                                    tipo_can::text = 'Remaster'::text)
)
```

)

#### 4.1.5 Definizione tabella Aggiungi

```
--TABELLA AGGIUNGI
CREATE TABLE Aggiungi
(
    id_playlist integer NOT NULL DEFAULT nextval('aggiungi_id_playlist_seq'::regclass),
    id_traccia integer NOT NULL DEFAULT nextval('aggiungi_id_traccia_seq'::regclass),
    CONSTRAINT aggiungi_pkey PRIMARY KEY (id_playlist, id_traccia),
    CONSTRAINT aggiugni_fkey1 FOREIGN KEY (id_playlist)
        REFERENCES public.playlist (id_playlist)
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT aggiungi_fkey2 FOREIGN KEY (id_traccia)
        REFERENCES public.traccia (id_traccia)
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)
```

#### 4.1.6 Definizione tabella Ascolto

```
--TABELLA ASCOLTO
CREATE TABLE Ascolto
(
    id_utente integer NOT NULL DEFAULT nextval('ascolto_id_utente_seq'::regclass),
    id_traccia integer NOT NULL DEFAULT nextval('ascolto_id_traccia_seq'::regclass),
    fasciaoraria character varying(20) NOT NULL,
    data timestamp without time zone NOT NULL,
    CONSTRAINT ascolto_pk PRIMARY KEY (id_utente, data, fasciaoraria),
    CONSTRAINT ascolto_fkey1 FOREIGN KEY (id_utente)
        REFERENCES public.utente (id_utente)
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT ascolto_fkey2 FOREIGN KEY (id_traccia)
        REFERENCES public.traccia (id_traccia)
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT tipo_orario CHECK (fasciaoraria::text = 'Mattina'::text OR
                                   fasciaoraria::text = 'Pomeriggio'::text OR
                                   fasciaoraria::text = 'Sera'::text OR
                                   fasciaoraria::text = 'Notte'::text)
)
```

#### 4.1.7 Definizione tabella Artista

```
--TABELLA ARTISTA
CREATE TABLE Artista
(
    id_artista integer NOT NULL DEFAULT nextval('artista_id_artista_seq'::regclass),
    nome character varying(25) NOT NULL,
    datanascita date,
    "nazionalità" character varying(25) NOT NULL,
    CONSTRAINT artista_pkey PRIMARY KEY (id_artista)
)
```

)

#### 4.1.8 Definizione tabella Album

```
--TABELLA ALBUM
CREATE TABLE Album
(
    nomealbum character varying(30),
    id_album integer NOT NULL DEFAULT nextval('album_id_album_seq'::regclass),
    id_artista integer NOT NULL,
    anno integer,
    CONSTRAINT album_pk PRIMARY KEY (id_album),
    CONSTRAINT album_fk FOREIGN KEY (id_artista)
        REFERENCES public.artista (id_artista)
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)
```

#### 4.1.9 Definizione tabella Collab

```
--TABELLA COLLAB
CREATE TABLE Collab
(
    id_artista integer NOT NULL,
    id_traccia integer NOT NULL,
    CONSTRAINT collab_pk PRIMARY KEY (id_artista, id_traccia),
    CONSTRAINT collab_fk1 FOREIGN KEY (id_artista)
        REFERENCES public.artista (id_artista)
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT collab_fk2 FOREIGN KEY (id_traccia)
        REFERENCES public.traccia (id_traccia)
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)
```

### 4.2 Definizione Trigger e Funzioni

Di seguito sono presenti le funzioni ed i trigger utilizzati per gestire le funzionalità descritte.

#### 4.2.1 Definizione trigger creazione Libreria

Di seguito è riportata la funzione ed il relativo trigger che crea una nuova libreria a partire da un utente che si è appena registrato.

```
--TRIGGER PER CREAZIONE LIBRERIA
CREATE OR REPLACE FUNCTION public.insert_lib()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    INSERT INTO libreria (id_libreria, num_playlist)
```

```

        VALUES (NEW.id_utente, 0);
    RETURN NEW;
END;
$BODY$;

CREATE TRIGGER ins_lib
    AFTER INSERT
    ON public.utente
    FOR EACH ROW
    EXECUTE FUNCTION public.insert_lib();

```

#### 4.2.2 Definizione trigger update Libreria

Di seguito è riportata la funzione ed il relativo trigger che aggiorna la libreria quando viene creata o eliminata una playlist dalla libreria di un utente.

```

--TRIGGER PER AGGIORNARE LA LIBRERIA DOPO UN'AGGIUNTA/RIMOZIONE DI UNA PLAYLIST
CREATE OR REPLACE FUNCTION public.incrementa_libreria()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    CASE TG_OP
    WHEN 'INSERT' THEN          -- single quotes
        UPDATE libreria AS l
        SET     num_playlist = l.num_playlist + 1
        WHERE   l.id_libreria = NEW.id_libappartenenza; -- fixed
    WHEN 'DELETE' THEN
        UPDATE libreria AS l
        SET     num_playlist = l.num_playlist - 1
        WHERE   l.id_libreria = OLD.id_libappartenenza
        AND     l.num_playlist > 0;
    ELSE
        RAISE EXCEPTION 'Unexpected TG_OP: "%". Should not occur!', TG_OP;
    END CASE;

    RETURN NULL;          -- for AFTER trigger this can be NULL
END
$BODY$;

CREATE TRIGGER inc_lib
    AFTER INSERT OR DELETE
    ON public.playlist
    FOR EACH ROW
    EXECUTE FUNCTION public.incrementa_libreria();

```

#### 4.2.3 Definizione trigger per l'incremento/decremento della grandezza di una Playlist

Di seguito è riportata la funzione ed il relativo trigger che aggiorna la playlist quando viene aggiunta o rimossa una traccia da una determinata playlist di un determinato utente.

```

--TRIGGER PER AGGIORNARE UNA PLAYLIST DOPO L'AGGIUNTA/RIMOZIONE DI UNA TRACCIA
CREATE OR REPLACE FUNCTION public.incrementa_playlist()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    CASE TG_OP
    WHEN 'INSERT' THEN          -- single quotes
        UPDATE playlist AS p
        SET     numerotracce = p.numerotracce + 1
        WHERE  p.id_playlist = NEW.id_playlist; -- fixed
    WHEN 'DELETE' THEN
        UPDATE playlist AS p
        SET     numerotracce = p.numerotracce - 1
        WHERE  p.id_playlist = OLD.id_playlist
        AND    p.numerotracce > 0;
    ELSE
        RAISE EXCEPTION 'Unexpected TG_OP: "%". Should not occur!', TG_OP;
    END CASE;

    RETURN NULL;      -- for AFTER trigger this can be NULL
END
$BODY$;

CREATE TRIGGER inc_plst
    AFTER INSERT OR DELETE
    ON public.aggiungi
    FOR EACH ROW
    EXECUTE FUNCTION public.incrementa_playlist();

```

#### 4.2.4 Definizione trigger partecipazione Artisti di una Traccia

Di seguito è riportata la funzione ed il relativo trigger che tiene traccia della partecipazione di uno o più artisti ad una singola traccia.

```

--TRIGGER PER TENERE CONTO DEL NUMERO DI ARTISTI CHE HANNO PARTECIPATO AD UNA TRACCIA
CREATE OR REPLACE FUNCTION public.inc_collab()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    UPDATE traccia SET num_artisti = num_artisti + 1
    WHERE NEW.id_traccia = traccia.id_traccia;
    RETURN NEW;
END;
$BODY$;

CREATE TRIGGER inc_artisti
    AFTER INSERT
    ON public.collab

```

```
FOR EACH ROW  
EXECUTE FUNCTION public.inc_collab();
```

## 4.3 Sequenze utilizzate

Di seguito sono presenti le sequenze utilizzate per l'auto-incremento di numerosi ID univoci di varie tabelle.

### 4.3.1 Sequenze

```
CREATE SEQUENCE public."Playlist_id_playlist_seq"  
INCREMENT 1  
START 1  
MINVALUE 1  
MAXVALUE 2147483647  
CACHE 1;
```

```
CREATE SEQUENCE public."Traccia_id_traccia_seq"  
INCREMENT 1  
START 1  
MINVALUE 1  
MAXVALUE 2147483647  
CACHE 1;
```

```
CREATE SEQUENCE public.album_id_album_seq  
INCREMENT 1  
START 1  
MINVALUE 1  
MAXVALUE 2147483647  
CACHE 1;
```

```
CREATE SEQUENCE public.artista_id_artista_seq  
INCREMENT 1  
START 1  
MINVALUE 1  
MAXVALUE 2147483647  
CACHE 1;
```

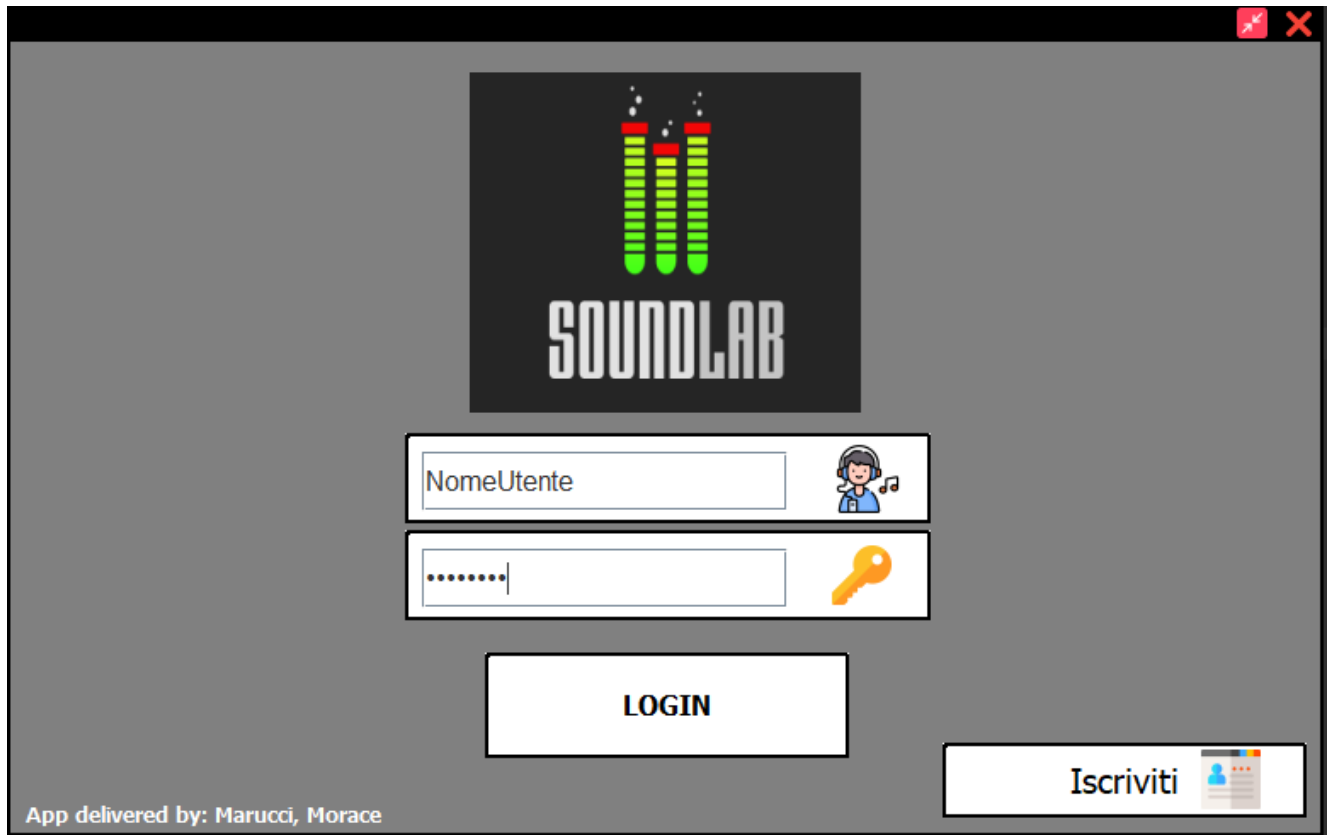
```
CREATE SEQUENCE public.utente_id_utente_seq  
INCREMENT 1  
START 1  
MINVALUE 1  
MAXVALUE 2147483647  
CACHE 1;
```



## 5 Esempi d'uso

### 5.1 Log-in

Qui di seguito la pagina che permette il Log-In.



The image shows a login window for 'SOUNDLAB'. At the top center is a logo consisting of three green test tubes with red caps and the word 'SOUNDLAB' in a stylized, blocky font. Below the logo are two input fields: the first is labeled 'NomeUtente' and has a small icon of a person with a headset; the second is a password field with a key icon. Below these fields is a large white button with the text 'LOGIN'. In the bottom right corner, there is a button labeled 'Iscriviti' with a small icon of a person and a document. At the bottom left, there is a small text label: 'App delivered by: Marucci, Morace'.

NomeUtente

.....

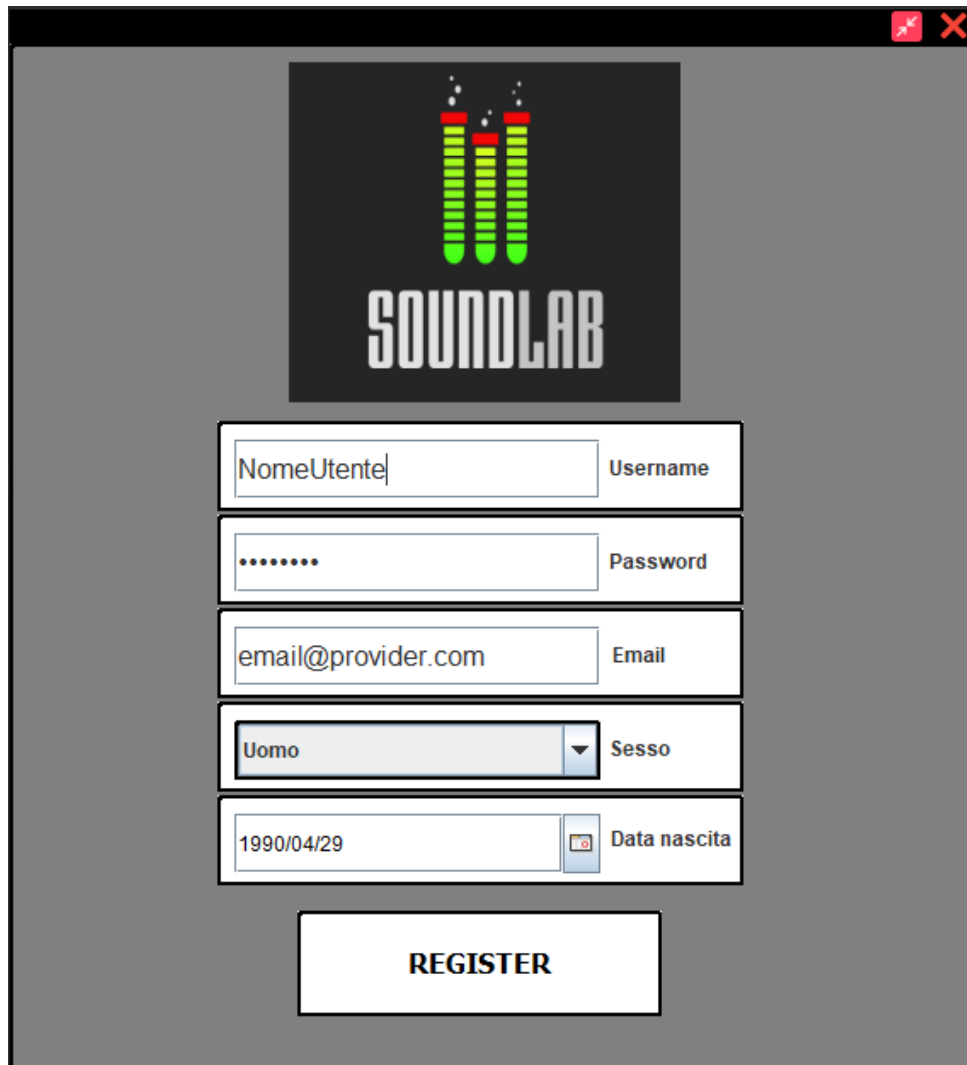
**LOGIN**

Iscriviti

App delivered by: Marucci, Morace

## 5.2 Registrazione

Qui di seguito la pagina che permette la Registrazione.



The image shows a web browser window with a registration form for 'SOUNDLAB'. The browser's title bar is black with standard red, yellow, and green window control buttons. The page has a grey background. At the top center is a logo consisting of three green test tubes with red caps and white smoke, with the word 'SOUNDLAB' in a bold, white, sans-serif font below them. Below the logo are five form fields, each with a label to its right: 1. A text input field containing 'NomeUtente' with the label 'Username'. 2. A text input field containing seven dots with the label 'Password'. 3. A text input field containing 'email@provider.com' with the label 'Email'. 4. A dropdown menu showing 'Uomo' with a downward arrow and the label 'Sesso'. 5. A date input field containing '1990/04/29' with a calendar icon and the label 'Data nascita'. Below these fields is a large white rectangular button with the word 'REGISTER' in bold black capital letters.

## 5.3 Pannello Home

Qui di seguito la pagina Home che permette di esplorare tutto il contenuto del nostro DB.

The screenshot shows a web application interface for 'SoundLab'. On the left is a dark sidebar with the SoundLab logo (three green test tubes) and the text 'SOUNDLAB'. Below the logo, it says 'Benvenuto: mattia'. The sidebar contains five menu items: 'HOME' (with a house icon), 'LIBRARY' (with a bookshelf icon), 'SEARCH' (with a magnifying glass icon), 'SIGN OUT' (with a door icon), and 'INFO' (with a person icon). At the bottom of the sidebar, it says 'App delivered by: Marucci, Morace'. The main content area is titled 'Home' and contains the text 'Esplora categorie:'. Below this, there are three sections: 'Artisti', 'Album', and 'Tracce'. Each section has a dropdown menu with 'Select' as the current selection, a text input field, and a blue button labeled 'VAI'.

# SOUNDLAB

Benvenuto: mattia

- HOME
- LIBRARY
- SEARCH
- SIGN OUT
- INFO

App delivered by: Marucci, Morace

## Home

Esplora categorie:

### Artisti

Select ▼

VAI

### Album

Select ▼

VAI

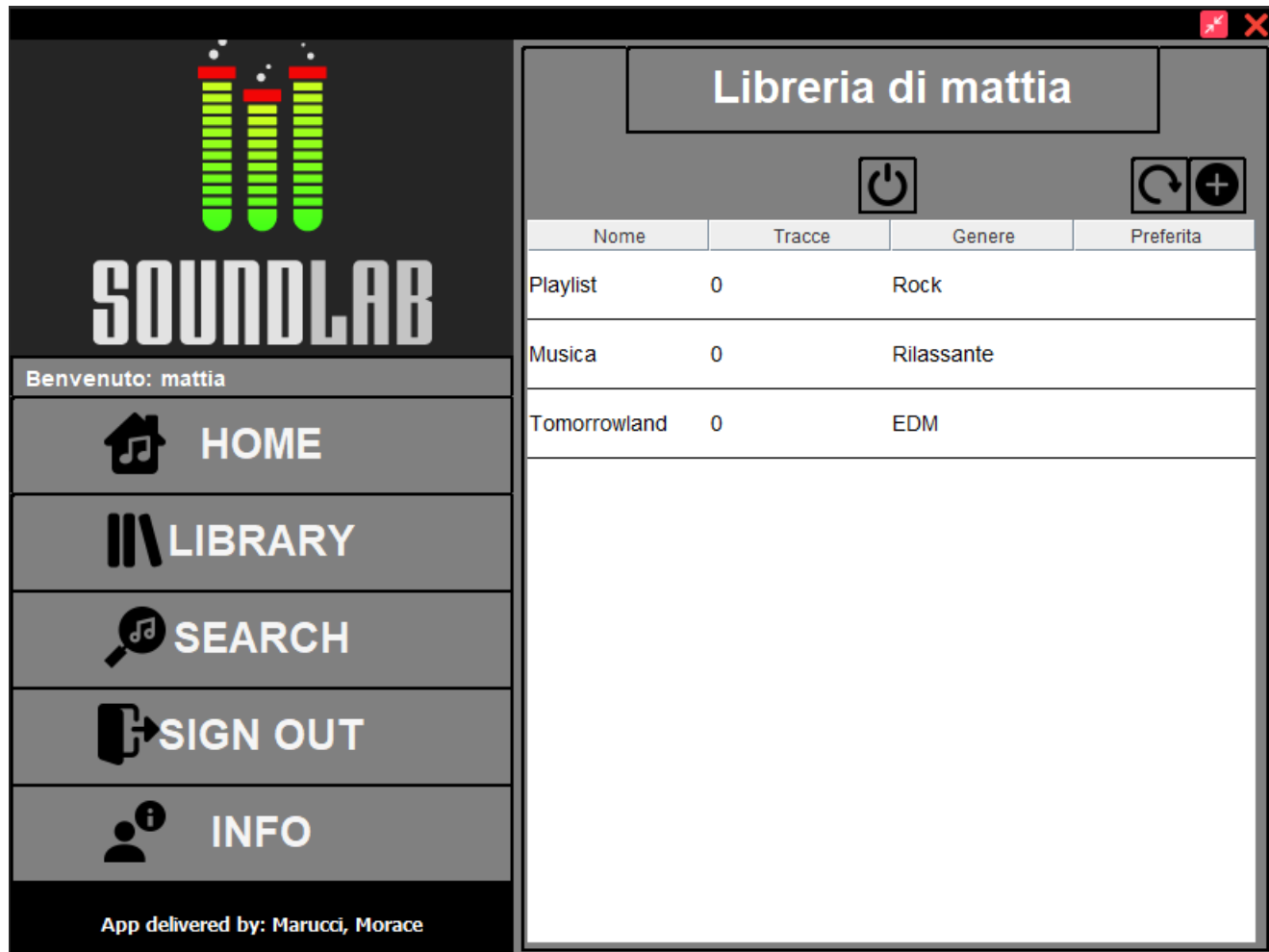
### Tracce

Select ▼

VAI

## 5.4 Pannello Libreria

Qui di seguito la pagina Libreria che permette di esplorare, ascoltare e gestire le proprie playlist e canzoni.



## 5.5 Pannello Ricerca

Qui di seguito la pagina di Ricerca che permette di esplorare canzoni in base a diversi tipi di ricerca.



**SOUNDLAB**

Benvenuto: mattia

- HOME
- LIBRARY
- SEARCH
- SIGN OUT
- INFO

App delivered by: Marucci, Morace

### Ricerca tracce

Traccia

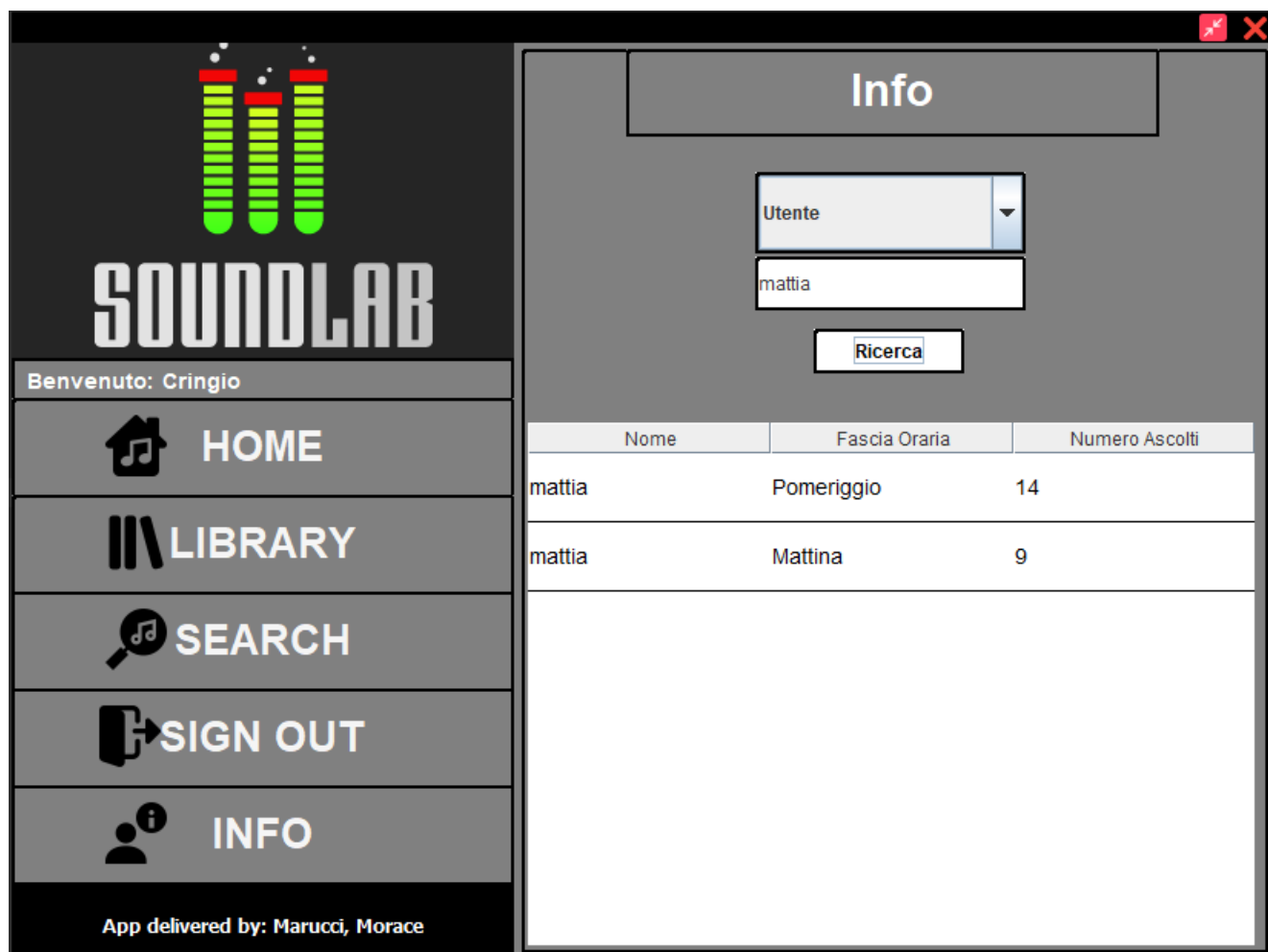
Poker Face

Ricerca

Nome	Genere	Tipo Canzone	Anno	Artista
Poker Face	Pop	Original	2008	Lady Gaga

## 5.6 Pannello Info

Qui di seguito la pagina di Info che permette di esplorare ad un Admin gli ascolti di determinati utenti o tracce specifiche.



The screenshot displays the SoundLab application interface. On the left is a dark sidebar with the SoundLab logo (three green test tubes) and the text 'SOUNDLAB'. Below the logo, it says 'Benvenuto: Cringio'. The sidebar contains five menu items: 'HOME' (house icon), 'LIBRARY' (book icon), 'SEARCH' (magnifying glass icon), 'SIGN OUT' (logout icon), and 'INFO' (person icon). At the bottom of the sidebar, it says 'App delivered by: Marucci, Morace'.

The main content area is titled 'Info'. It features a search form with a dropdown menu labeled 'Utente' showing 'mattia', a text input field containing 'mattia', and a 'Ricerca' button. Below the search form is a table with three columns: 'Nome', 'Fascia Oraria', and 'Numero Ascolti'.

Nome	Fascia Oraria	Numero Ascolti
mattia	Pomeriggio	14
mattia	Mattina	9

## 6 Conclusioni

L'applicazione sviluppata ha bisogno del suo DB per funzionare, quest'ultimo può essere trovato insieme al codice sorgente all'interno della repository GitHub in formato .sql o restore di Postgres. Inoltre all'interno della repository è possibile visionare tutti gli schemi UML presenti in questa documentazione e tutta la linea temporale dei commit effettuati.

### 6.1 Repository GitHub utilizzata

La repository utilizzata è stata la seguente: [LINK](#)