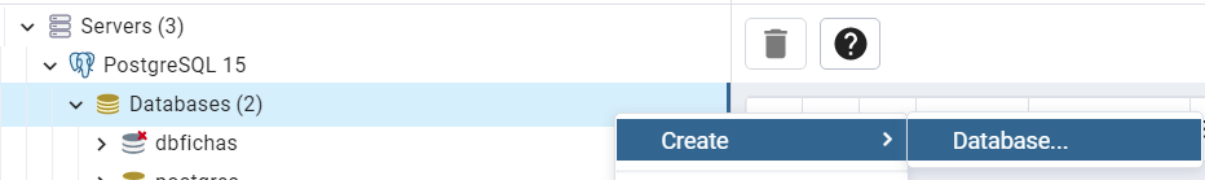
Installation manual

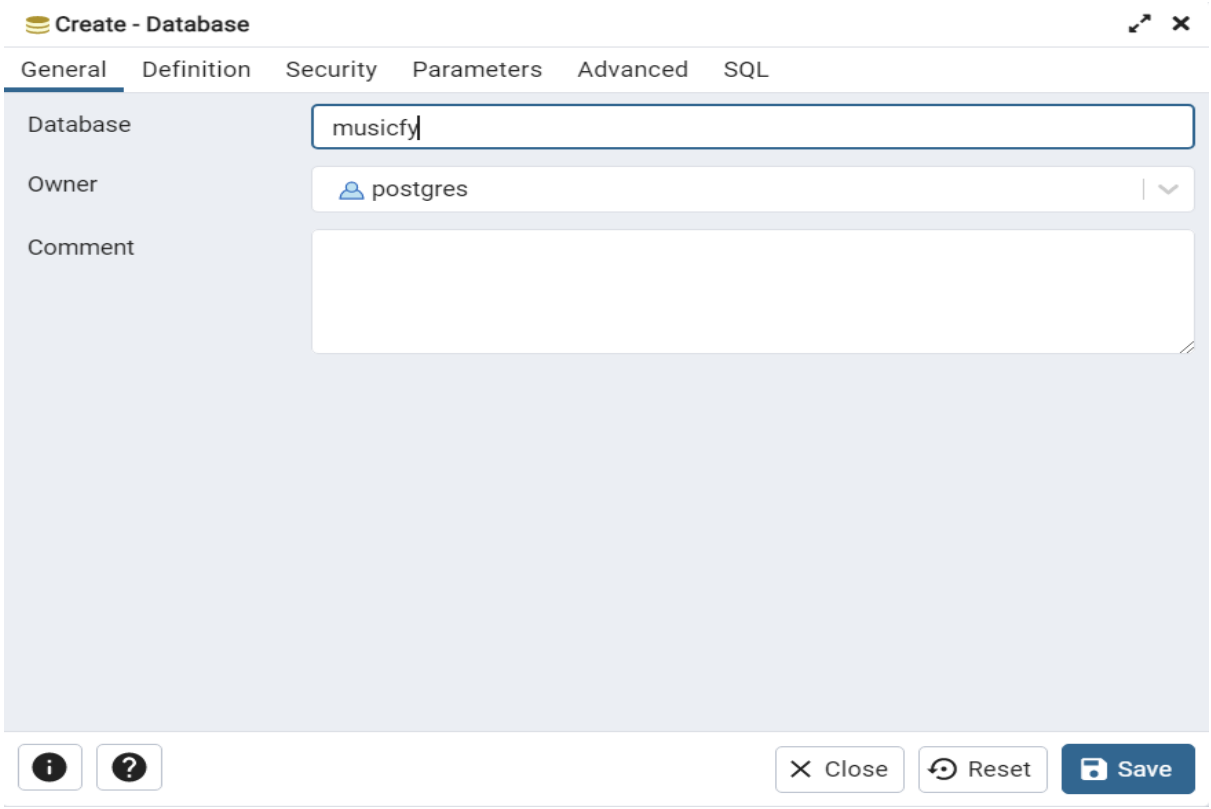
Create the database

Open pgAdmin 4 and log into the postgres default user.

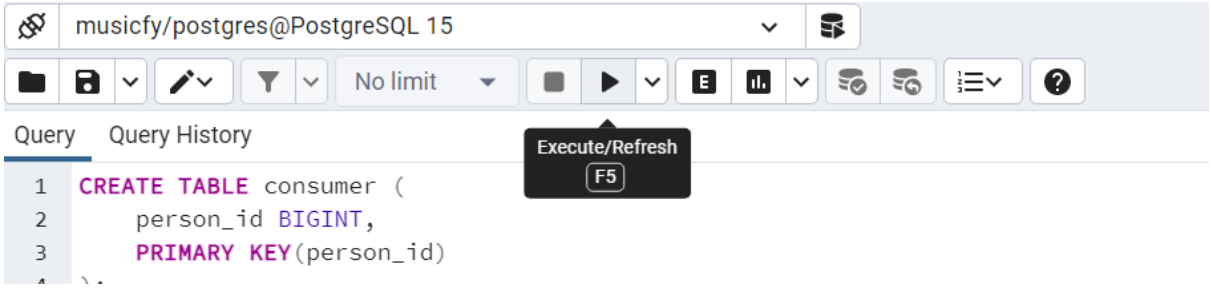
Then right click Databases and create the database.



Insert the name “musicfy” and save the changes.



Right click the created database and select Query Tool. Open the tables.sql file and copy and paste it into the query tool and run it.



In order to create admins and record labels you need to insert them manually into the query tool. Use the following queries:

Query Query History

```
INSERT INTO person(username, password, email, name, birthdate)
                VALUES ('john_doe', '123', 'example@gmail.com', 'John', '16/06/200') RETURNING id;
```

Data Output Messages Notifications

	id [PK] bigint
1	1

Query Query History

```
INSERT INTO administrator (person_id)
                VALUES (1);
INSERT INTO record_label (name)
                VALUES ('example');
```

Be extra careful inserting the id into the administrator table.

Install the libraries

Run the following commands in your terminal in the folder directory:

```
python -m pip install --upgrade pip
pip install virtualenv
python-m virtualenv venv
pip install -U Flask
pip install psycopg2
pip install PyJWT
```

User manual

Operations

Consumer:

Register a new account;

Subscribe;

Create a playlist (subscription only);

Play a song;

Admin:

Register artist accounts;

Generate a pre-paid card;

Artist:

Create songs;

Create albums;

Everyone:

Login;

Search songs with a specific keyword;

Search artist details;

Leave a comment or reply to an existing one;

Generate a monthly report of played songs;

Note: Every operation besides login and registering as consumer can only be executed after the user logs in.

POST http://localhost:8080/dbproj/user

Body (json)

Register consumer:

```
{
    "username": "john_doe",
    "password": "123",
    "email": "aaa",
    "name": "john",
    "birthdate": "22/06/2003"
}
```

Register artist(admin only):

```
{
    "artistic_name": "mc nooob gamer",
    "label_id": 1,
    "username": "bbb",
    "password": "123",
    "email": "aaa",
    "name": "bbb",
    "birthdate": "22/06/2003"
}
```

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: user_id (if it succeeds)}
```

PUT http://localhost:8080/dbproj/user

Body (json)

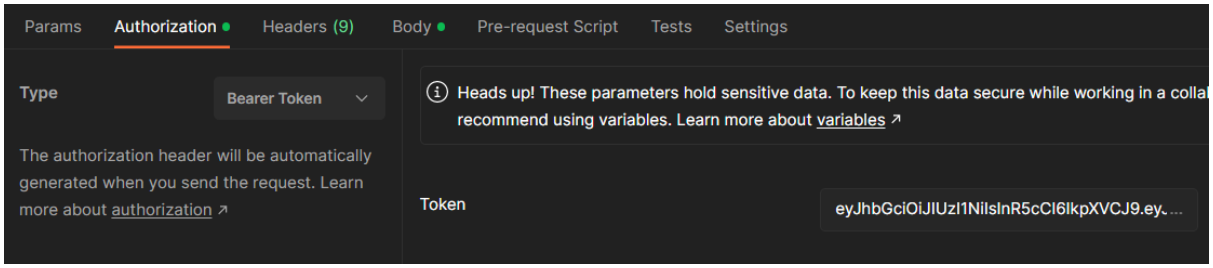
Login:

```
{
    "username": "john_doe",
    "password": "123"
}
```

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: auth_token (if it succeeds)}
```

After logging in, copy and paste the authentication token into Authorization→Bearer Token



POST http://localhost:8080/dbproj/song

Body (json)

Add a song(artist only):

```
{
    "name": "aaa",
    "genre": "Rap",
    "release_date": "05/05/2010",
    "duration": 210, //seconds
    "publisher": 1,
    "other_artists": []
}
```

Resp:

```
{"status": status_code, "errors": errors (if any occurs), "results": song_id (if it succeeds)}
```

POST http://localhost:8080/dbproj/album

Body (json)

Add album (can add new songs or ids of songs that already exist and artist only):

```
{
    "name": "album fixe",
    "release_date": "22/06/2003",
    "genre": "Rap",
    "publisher": "1",
    "songs": [{"name": "bcb",
    "genre": "Rap",
    "release_date": "05/05/2010",
    "duration": 210,
    "publisher": 1,
    "other_artists": []}, 2]
}
```

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: album_id (if it succeeds)}
```

GET http://localhost:8080/dbproj/song/{keyword}

Search songs with a specific keyword.

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: [{“title”: “song_title”, “artists”: [“artist_name1”, “artist_name2”, (…)], “albums”: [“album_id1”, “album_id2”, (…)]}, (…)]}
```

GET http://localhost:8080/dbproj/artist_info/{artist_id}

Get the details of a specific artist.

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: {“name”: “artist_name”, “songs”: [“song_id1”, “song_id2”, (…)], “albums”: [“album_id1”, “album_id2, (…)], “playlists”: [“playlist_id1”, “playlist_id2”, (…)]}}
```

POST http://localhost:8080/dbproj/subcription

Subscribe to premium as a consumer:

```
{
    "period": "semester",
    "cards": ["H65MKN633K6ZBOWY",
        "YLZUV4WKFYHKZCQI",
        "L8MJA8YAU20VLQJW"]
}
```

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: subscription_id (if it succeeds)}
```

POST http://localhost:8080/dbproj/playlist

Body (json)

Create a playlist(premium consumers only):

```
{
    "playlist_name": "ALL",
    "visibility": "public",
    "songs": [1, 2, 3, 4]
}
```

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: playlist_id (if it succeeds)}
```

PUT http://localhost:8080/dbproj/{song_id}

Play a song(consumer only).

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs)}
```

POST http://localhost:8080/dbproj/card

Body (json)

Generate pre-paid cards(admin only):

```
{
    "number_cards": 3,
    "card_price": 50
}
```

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: [id_card1, (…)] (if it succeeds)}
```

POST http://localhost:8080/dbproj/comments/{song_id}

POST http://localhost:8080/dbproj/comments/{song_id}/{parent_comment_id}

Body (json)

Leave or reply to an existing comment.

```
{
    "comment": "Musica boa!"
}
```

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: comment_id (if it succeeds)}
```

GET http://localhost:8080/dbproj/report/{year-month}

Generate a monthly report.

Resp:

```
{“status”: status_code, “errors”: errors (if any occurs), “results”: [
   {“month”: “month_0”, “genre”: “genre1”, “playbacks”: total_songs_played},
   {“month”: “month_0”, “genre”: “genre2”, “playbacks”: total_songs_played},
   {“month”: “month_1”, “genre”: “genre1”, “playbacks”: total_songs_played},
(…)
]}
```

Details

Locks

We used locks for updating the pre-paid cards balance and when adding a reply comment using the select for update query, locking the pre-paid card ids and the parent comment ids respectively.

Trigger

The trigger calls a function every time a consumer plays a song and deletes every song in the top ten playlist and then inserts the correct songs.

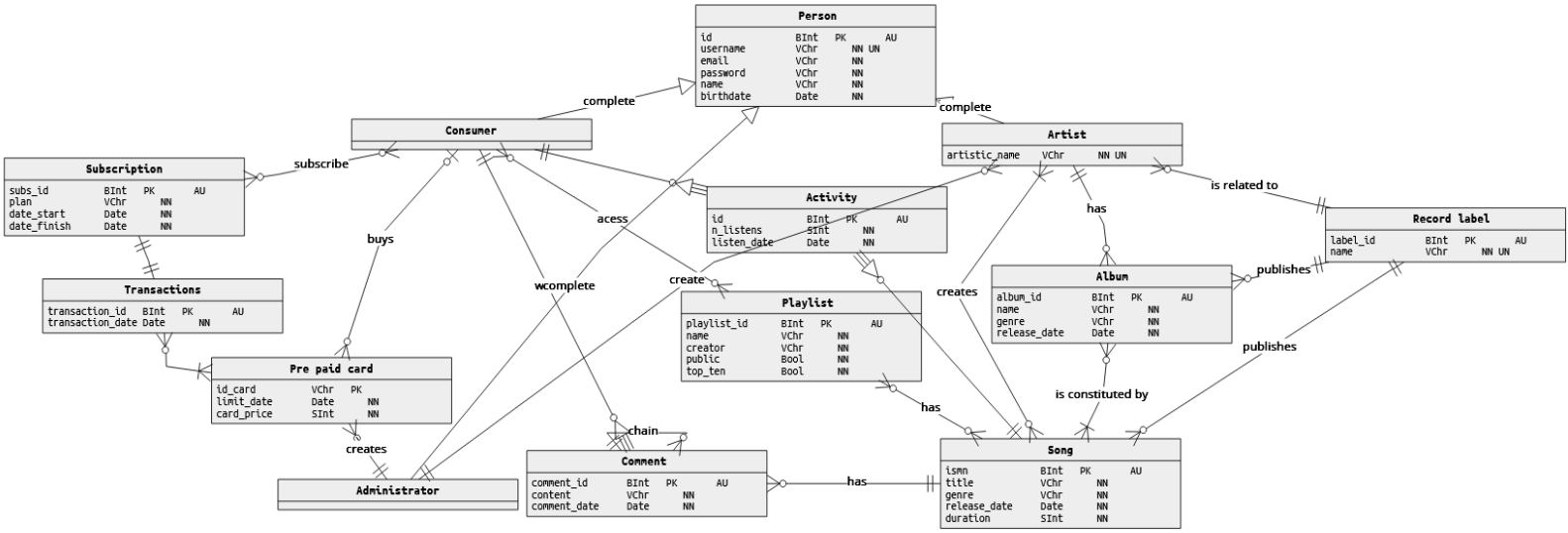
Extra

The admins and record labels should be added manually to the database.

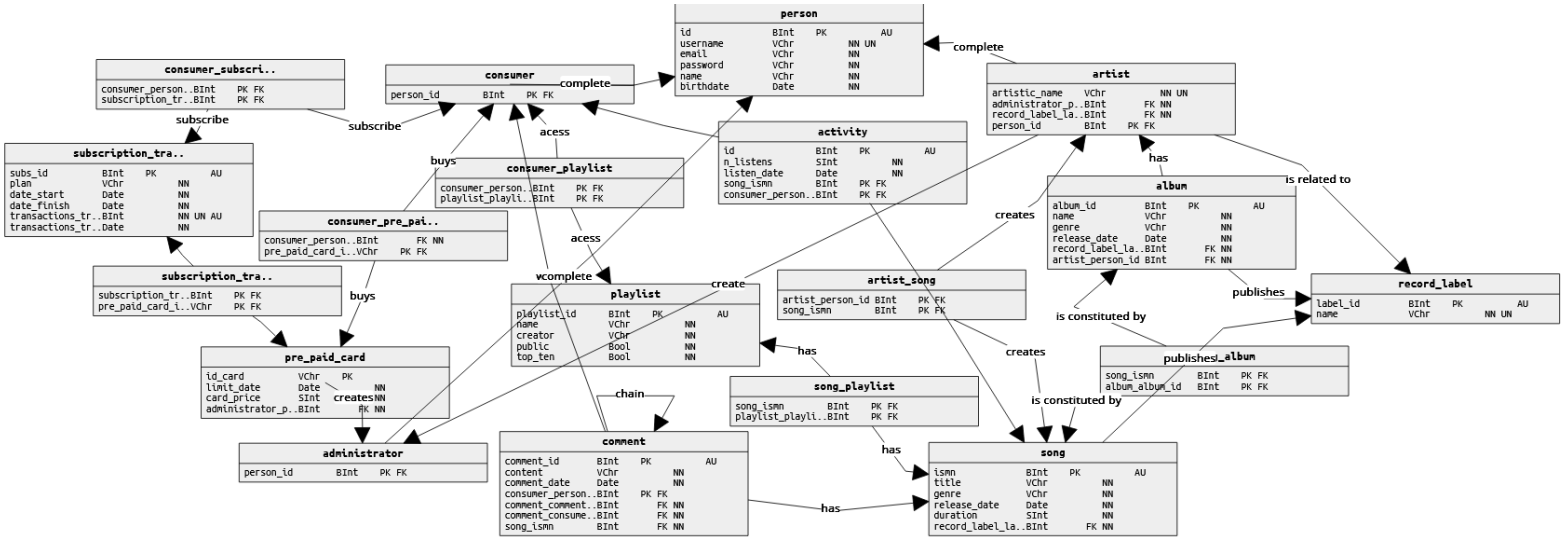
Development plan

The team members worked equally in this project with both around 50 hours of work.

Conceptual ER



Physical ER



Marcelo Gomes nº2021222994

Pedro Brites nº2021226319