Filipe Rodrigues - 2021218054
Joás Silva - 2021226149
André Louro - 2021232388

Installation Guide:

Creating the database

First, you need to log into the postgres user, or any other user with permissions. You can do this by running the following command:

```
psql -h localhost -p 5432 -d postgres -U postgres
```

Then, you need to create the **Database** and the **Admin user**. You can do this by running the following commands:

```
CREATE DATABASE oblivionDB;
CREATE USER admin PASSWORD 'admin'; -- Then you will be prompted to enter a password
\c oblivionDB -- Change the current database to oblivionDB (current user is postgres)
GRANT ALL ON SCHEMA public TO admin; -- grant all privileges to the admin user on this database
exit
```

Create the tables

Now that you have created the database and the user, you need to run the **createTables.sql** file. You can do this by running the following command:

```
psql -h localhost -p 5432 -d oblivionDB -U admin -f createTables.sql
# Enter password when prompted (admin)
```

Connect to the database using the admin user

You can connect to the database using the admin user by running the following command:

```
psql -h localhost -p 5432 -d oblivionDB -U admin
```

Importing libraries

You just need to run one of the following commands:

```
py -m pip install -r requirements.txt # Windows
```

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

User Manual:

Main Operations:

Administrator

- Create artist accounts
- Create a specific number of pre-paid cards (10, 25 or 50)
- Create labels

Consumer

- Create their account
- Buy a subscription (month, quarter or semester) using pre-paid cards
- Have a playlist with their top 10 most played songs (last 30 days)
- Comment on songs or on another comments
- Access to all songs, albums and public playlists

Consumer w/ subscription

- Create playlists (public or private)

Artist

- Create albums
- Create songs

POST /dbproj/user/

Example to create a consumer:

Body (json)

```
{
    "username": "john_doe",
    "password": "johndoe1234",
    "name": "João",
    "address": "Ponte 25 de Abril",
    "email": "doe_john@example.pt",
    "birthdate": "1843-02-22"
}
```

Response (json)

```
{
    "status": status_code,
    "errors": errors (if any occurs),
    "results": user_id
}
```

Example to create an artist:

Header

- token (admin token)

Body (json)

```
{
    "username": "Dom_quixote",
    "password": "cool_password",
    "name": "Jacinto Leite",
    "address": "Rua do Loreto",
    "email": "dom@quixote.pt",
    "birthdate": "1947-06-19",
    "label_id": 1,
    "artistic_name": "Dom Quixote"
}
```

Response (json)

```
{
    "status": status_code,
    "errors": errors (if any occurs),
    "results": user_id
}
```

PUT /dbproj/user

Example to login a user/admin/artist:

Body (json)

```
{
    "username": "john_doe",
    "password": "johndoe1234"
}
```

Response (json)

```
{
    "status": status_code,
    "errors": errors (if any occurs),
    "results": auth_token
}
```

POST /dbproj/song/

Example to create a song:

Header

- token (artist token)

Body (json)

```
{
    "title":"Song Trio",
    "release": "2022-07-02",
    "duration": "2023-05-02 00:01:34.000",
    "genre": "Funk",
    "label_id": 1,
    "other_artists": [3, 7]
}
```

Response (json)

```
{
    "status": status_code,
    "results": song_id,
    "error": error_message (if any)
}
```

POST /dbproj/label/

Example to create a label:

Header

- token (admin token)

Body (json)

```
{
    "name": "Disco",
    "contact": "919361213"
}
```

Response (json)

```
{
    "status": status_code,
    "results": label_id,
    "error": error_message (if any)
}
```

POST /dbproj/album/

Example to create an album:

Header

- token (artist token)

Body (json)

```
{
    "title": "Epic Songs Fantastic",
    "release": "2023-05-06",
    "label_id": 1,
    "other_artists": [5, 3, 10],
    "songs": [2, 1, 19, 3, 4, 20]
}
```

Response (json)

```
{
    "status": status_code,
    "results": album_id,
    "error": error_message (if any)
}
```

POST /dbproj/card/

Example to generate a Pre-paid Card:

Header

- token artist(token)

Body (json)

```
{
    "number_cards": 5,
    "card_price": 10
}
```

Response (json)

```
{
    "status": status_code,
    "results": [card_id1, card_id2, …],
    "error": error_message (if any)
}
```

POST /dbproj/playlist/

Example to create a playlist:

Header

- token (consumer token)

Body (json)

```
{
    "playlist_name": "Depressive Again :(",
    "visibility": "public",
    "songs": [1, 3, 5, 9, 11]
}
```

Response (json)

```
{
    "status": status_code,
    "results": playlist_id,
    "error": error_message (if any)
}
```

POST /dbproj/subscription/

Example to create a subscription:

Header

- token (consumer token)

Body (json)

```
{
    "period": "quarter",
    "cards": [
        "MCTJINWLEMIZ1ZAY",
        "P6Z6TV3IUNEXSLV0",
        "XQ4LFLZ50PXKF9B8",
        "QXHIF3WXGRVXYBK6",
        "ZWLJTSWH9FVZB8CX",
        "CCISQLZDN7NNHGWD"
    ]
}
```

Response (json)

```
{
    "status": status_code,
    "results": subscription_id,
    "error": error_message (if any)
}
```

POST /dbproj/comments/<song_ismn>

Example of commenting a song:

Header

- token (consumer token)

Body (json)

```
{
    "comment": "eu sou um comentário em uma música!!"
}
```

Response (json)

```
{
    "status": status_code,
    "results": comment_id,
    "error": error_message (if any)
}
```

POST /dbproj/comments/<song_ismn>/<parent_comment_id>

Example of replying to a comment:

Header

- token (consumer token)

Body (json)

```
{
    "comment": "eu sou um comentário em outro comentário lols!!"
}
```

Response (json)

```
{
    "status": status_code,
    "results": comment_id,
    "error": error_message (if any)
}
```

GET /dbproj/report/YYYY-MM

Example of a

Header

- token (consumer token)

Response (json)

```
{
    "status": status_code,
    "results": [
      {"month": "4", "genre": "Pop", "playbacks": total_songs_played},
      …
    ],
    "error": error_message (if any)
}
```

GET /djproj/song/<keyword>

Header

- token (consumer token)

Response (json)

```
{
    "status": status_code,
    "results": [
      {"title": "title", "label_id": label_id,
       "artists": [artist_id, ...], "albums": [album_id, ...]},
      …
    ],
    "error": error_message (if any)
}
```

GET /djproj/artist_info/<artist_id>

Header

- token (consumer token)

Response (json)

```
{
    "status": status_code,
    "results": [
      {
          "name": artist_name,
          "songs": [song_id, ...],
          "albums": [album_id, ...],
          "playlists": [playlist_id, ...]
      },
      …
    ],
    "error": error_message (if any)
}
```

PUT /dbproj/<song_ismn>

Header

- token (consumer token)

Response (json)

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

Relevant Information

Trigger:
The triggers is activated every time a user plays a song, it calls a function that regenerate the top10 consumer most played songs in the month

Locks:
We used locks in six different situations:

- Subscriptions
    - Locks the “transaction” table and blocks the cards for update (only the ones the user used);
- User Registration
    - Locks the “credentials” and “persons” tables;
- Commenting
    - Locks the “comments” table;
- Song Creation
    - Locks the “artist” and “label” tables;
- Label Creation
    - Locks the “label” table;
- Album Creation
    - Locks the “album” table;

`.env` **file:**
This is the file where we keep our configs, in the **.zip** of the project, there is a **.env.sample** where you should remove **.sample** from the name and change the things as you want.