Music Shared Application

Anthony Chevrolet

Haaga-Helia University of Applied Sciences

Bachelor’s Thesis

2022

Bachelor of Business Administration

Abstract

Author(s)
Anthony Chevrolet

Degree
Bachelor of Business Administration

Report/thesis title
Music Shared Application

Number of pages and appendix pages
77 + 28

The author has elaborated this thesis as part of the validation of his degree. This work concerns the elaboration of a project of music sharing through different streaming platforms.

The objective of this work is to see if it's possible to share content across platforms, which may be useful if a user wants to send a playlist to someone who does not use the same service.

The scope of this work will be divided into three distinct parts. The first is to study different platforms and to see the specifications and limitations of each in order to determine what will be possible to put in place.

The second part concerns the elaboration of a server allowing to execute the different operations in order to link the data through the platforms in a database. This step will be done in a way that will allow the possibility to integrate other services in the future, through a Node.Js server. It will also provide a web page allowing to share the content of the database directly via a URL.

The last part is to develop a mobile application that will be connected to the server. It will be able to allow the user to log in with his account and access his content which can be shared with other users. It's through a React Native application that this work will be done so that it can be deployed on IOS and Android.

The author completed this work in 400 hours over a period of 10 weeks in parallel with these courses. This work was done between 7 March and 19 May 2022.

The result is a web server developed with Node.Js, a cross-platform mobile application in React Native and this document to understand the whole project.

This work ends with a conclusion allowing the author to shed light on the various obstacles encountered and to suggest ways of improving this work. He will also share his learning throughout the project and his personal interest.

Keywords
Node.js, React Native, Expo, Mobile App, API, Music Content

Table of contents

1 Introduction

Today, most of the population owns a smartphone. It's like a life accessory that becomes more and more a technological extension of ourselves with a very high level of personalization. Whether it’s the visual, the design or the content with the different mobile applications available. A mobile application is a fast way to access a service, especially thanks to the mobility of smartphones. The most popular applications of the last few years, in addition to social networks, are those that offer a streaming service. Especially those that offer a music service that allows us to listen to an incredible amount of music directly from our smartphone. There’s an existing market in this domain and therefore several mobile applications exist that offer the same service. The problem that I've seen is when I want to share a content from a certain platform with a person using another application. In most cases, he rewrites the name of the music I dictate to him and it works, but sometimes this process can be more or less time consuming. In the same context, if I want to share my playlist of the moment, it can take even more time. So, the goal of my thesis is to develop a mobile application that would allow to share music content across different music streaming services quickly and easily. It will be necessary to make links and several verifications between the different resources in order to have the highest level of fiability possible. All these links between the different data will be stored in the same database. Therefore, a remote server containing the database will be created. In order not to overload the phone, the link verification process will be done on the server associated with the database. During this thesis, I will also be able to apply the knowledge acquired through my studies. The technologies I will use are mainly React Native for the mobile application and Node.Js for the server part.

2 Specifications of requirements

In this section I will explain in detail the overall objectives of my thesis. My work is separated into three quite distinct parts: Data recovery, Server processing and Mobile application.

The first part of my work consists in analysing the data structure of the different platforms in order to know the specifications of each one. I imagine that each of the studied platforms does not have the same implementation whether it’s for the access to the data, the authentication or the structure of the resources. This work will allow me to be clear and to see what is possible in my project. The platforms I decided to study in my work are the following ones: Spotify, YouTube Music and Deezer.

In a second time, I could set up the logical part of the application. It’s through a Node.Js server that I will be able to perform the different links to the platforms in order to record them in the database. Initially, the database is empty and will fill up by the users when they will use the application. The more the database is filled, the faster the analysis processing will be done.

The last part will be the mobile application where the goal is mainly to exploit the server set up beforehand. It’s through this application that users will be able to search for content and feed the database. It will also be possible to open the searched content directly on the platform of their choice.

In view of the amount of work , I have decided to separate the empirical part into three parts. It’s necessary to understand the different parts separately and to put them all together in one block made it more difficult to understand the work as a whole.

2.1 Objectives

2.1.1 Data recovery

The first feature I would like to implement is the authentication process for each platform. I would like, through this functionality, to understand how to interact with the different platforms I have chosen in my work. I would also like, if possible, to be able to understand the authentication steps of a user to be able to use the latter within the mobile application.

The second one is defining the data that will be processed. By studying the data structures, I would like to be able to define the key data that will be processed by the server.

I want to have the possibility of collecting data on the different platforms. I would like to be able to retrieve data about an artist, an album and a song through the different platforms and then connect them together. Also check the possibility to retrieve all of an artist's albums across each platform so that I can compare the content.

After retrieve a song, I would like to be able to find the identifiers of an artist and an album associated with it.

Another functionality that I want implement is to be able to search for a song with the search engine of each platform. I would like to be able to use the search engine of each platform to find resources. It is through this that the database will be fed mainly.

In my work, I will not be able to study all the platforms that exist. That's why I will code in such a way as to have a progressive integration. I would like to be able to code the different services throughout my work in a flexible way and structure them without impacting the whole application. The idea would be to use different design patterns.

I would like to know the implementation specifics of each platform studied in order to have an overview of each one and make a decision on how the data will be analysed.

The last part is to know how I can exploit data user. With the help of this feature, I would like to be able to understand and check the access and content creation possibilities of a user, allowing me to see if it is possible to create, for example, a playlist directly on his account.

2.1.2 Server

The server is able, depending on the search, to perform and analyse the content to see if the results are in the database. Otherwise, it fills the database. Starting from a referenced piece in the database, it is able to find the similar piece on the different platforms offered and link data with a “standardized” notation sufficient to unify the content.

I want to have the possibility to create playlists with references specific to the database, with the possibility of exporting it to any platform. Also let the possibility to import the user's existing playlists from this platform and convert them to another platform.

All this functionality will be defined in a programming interface allowing communication between it and users from the mobile application. In addition, add an interactive interface allowing content to be shared on the Internet.

2.1.3 Mobile application

First, I want to setup authentication process inside the app to allows you to identify yourself on one of the platforms available . If the user does not have a subscription, he can simply create an account or use it as a guest.

Once he is connected, he has the possibility of carrying out a search according to the platform with which he is identified. He also can manage his musical content inside the app like create a playlist and link it directly with the platform he uses. He can also put albums or artists in favorites which will be found in a page dedicated to this. (Note that a guest user will also be able to do so, but this data will not be transferred if he uses the application on another device). The user can modify his personal information (password) at any time.

The application is directly connected to the server. Several contents of the application can be shared via a sharing link. This link can be access by a browser and allow everyone to open the content on a specified platform or inside the app.

2.2 Organisation

2.2.1 Deliverable

My thesis will come with a Node.js project connected to a database as well as a React Native mobile application that will be connected to my server.

2.2.2 Project schedule

400 hours has been reserved for the project. The initial and real main planning is on the Appendix 42.
The project started on March 7th 2022 and will end around May 19th 2022.

2.2.3 Backlog

In order to better organize myself during my work, I decided to make a to-do list. Organisation is important, even in an individual project. When you are under a time constraint, it is easy to get caught up and spend a lot of time on one task over others (see Appendix 43.

2.3 Out of scope

The most important part of my work is the data link processing part. That's why the latter is my priority compared to the mobile application where the design will be put aside. Indeed, for me, the realization of the mobile application brings me less added value in terms of research and analysis. I see it more as a way to test and see if the analyses were successful. In the long term, the application could extend to a whole other project of social network based on music where the result of my thesis would be a part of this project.

Initially, I wanted to be able to search for links from any type of resource regardless of the platform used. During my work, I could see that I would not have enough time to implement all possible use cases. That's why I will mainly focus on one platform and use it to connect to the others.

3 Theoretical Framework

In this part I will first define the data set studied on the different platforms in order to understand how the resources are structured individually and how they are accessed. Once this is done, I can proceed to the development of a relational database. Within the database, the data will be separated independently so that it can evolve with the integration of other platforms in the future. It will also distinguish between platform specific data.

In this chapter, I will also make an inventory of the different technologies and libraries that I will use for this project, both on the server and on the mobile application. I will also add different screen mock-ups made at the beginning of the thesis to give me an idea of the interface I will present to the users.

In the empirical part, I will try to understand how each resource works together and also describe the implementation specifications of each platform in the clearest possible way illustrated by examples in the Appendix.

3.1 Spotify

Spotify is an audio streaming platform founded on the 23 April 2006. It provides an SDK that enables developers to access their music data and various other features (Wikipedia 2022a). For my project, I used the Spotify API to retrieve data from the platform and analyse it. I also need to implement a secure authentication method to allow the users to interact with the API.

3.1.1 Resources available

The first resource identified is an album. There are various characteristics of an album resource. Here's a list of things I'll be using for my project (Table 1). The entire additional details can be seen on Spotify's website (Spotify 2022a).

Table 1. Essentials attributes from Spotify API for an Album

Key	Description
album_type	The type of the album. Allowed values: album, single, compilation
id	The Spotify ID for the album.
images	Array with cover art for the album in various sizes.
name	The name of the album.
release_date	The date where the album was released
release_date_precision	The precision of the release_date attribute Allowed values: year, month, day
external_ids	The upc of an album **Note**: only accessible by the endpoint: v1/albums/:id
artists	Array of owner artists of the album.
tracks	Array of tracks of the album **Note**: only accessible by the endpoint: v1/albums/:id

The second resource used is a track. A song specified as such using the API provides access to a variety of data (Table 2). A song is linked to an album and one or many artists. The Spotify website has all the details (Spotify 2022b).

Table 2. Essentials attributes from Spotify API for a Track

Key	Description
album	Album where the track is linked
artists	Array of Artist from the tracks
name	Name of the track
duration_ms	The track length in milliseconds
id	The Spotify ID for the track.
track_number	The number of the track. If an album has several discs, the track number is the number on the specified disc.
disc_number	The disc number (usually 1 unless the album consists of more than one disc).
external_ids	The isrc of the track **Note**: only accessible by the endpoint: v1/tracks/:id

The last one I identified is an Artist. He is mostly associated with a song or album. The following is a list of characteristics that I will employ in my work (Table 3). The entire details can be seen on Spotify's website (Spotify 2022c).

Table 3. Essentials attributes from Spotify API for an Artist

Key	Description
id	The Spotify ID for the artist.
image	Array of artist in various sizes.
name	The name of the artist.

3.1.2 Access data

Spotify has made a number of endpoints available. I will use a part of that in my project and the detail is in Appendix 1. Essential's endpoints used from Spotify's API.

3.2 Deezer

Deezer is a French music streaming service that was created in Paris in August 2007. Deezer offers 90 million music, 30,000 radio channels, and 100 million playlists available (Wikipedia 2022b). This company also provides an API for developers to access the data. In terms of Spotify's API, I use Deezer to get the information I need. I utilize Deezer to get identifiers for each available resource, just like I do with Spotify's API (Artist, Album and song).

3.2.1 Resources available

The different resources are the same as on Spotify. The characteristics will be very similar, although with somewhat different names. The complete description is available on Deezer's website. (Deezer 2022a). The data set used for each resource (album, track and artist) is defined in tables Table 4, Table 5 and Table 6.

Table 4. Essentials attributes from Deezer API for an Album

Key	Description
record_type	The record type of the album (EP / ALBUM / etc...) **Note**: only accessible by the endpoint: /album/:id
id	Identification of the album
cover	Cover of the album
title	Name of the album
release_date	Date of the release **Note**: only accessible by the endpoint: /album/:id
upc	The UPC code of an album **Note**: only accessible by the endpoint: /album/x
contributors	Array with artists of the album **Note**: only accessible by the endpoint: /album/:id
tracks	Array with tracks of the album **Note**: only accessible by the endpoint: /album/:id

Table 5. Essentials attributes from Deezer API for a Track (Deezer 2022b)

Key	Description
album	Album where the track is referred
contributors	Array with artists of the album **Note**: only accessible by the endpoint: /track/:id
title	Name of the track
duration	The track length in seconds
id	ID for the track.
track_position	The position of the track in the album **Note**: only accessible by the endpoint: /track/:id
disk_number	The number of disk available on the disk. **Note**: only accessible by the endpoint: /track/:id
isrc	The ISRC code of a track **Note**: only accessible by the endpoint: /track/:id

Table 6. Essentials attributes from Deezer API for an Artist (Deezer 2022c)

Key	Description
id	The Deezer ID for the artist.
picture	Link of artist picture
name	The name of the artist.

3.2.2 Access data

Deezer allows in most cases, the same necessary endpoints as Spotify that I need to use for my project. The detail is in Appendix 2. Essential’s endpoints used from Deezer’s API.

3.3 YouTube Music

YouTube Music is a music streaming service that YouTube has created. It was launched in November 2015, a little bit recently than Spotify and Deezer. The data is based on YouTube, and one of the platform's strengths is the ability to watch a song's clip immediately from the application (Wikipédia 2022c).

There is no official API for YouTube Music. I discovered an unofficial Python Library and quickly checked if it could be used from Node JS, which it could. I'll use the Official YouTube API for personal functions like creating a playlist. I checked and using python Library for authentication from the mobile app will be challenging, whereas using Google auth from the mobile app will be easier and safe. I'll probably set up a security link between the Mobile App and the server in the future so that the Python Library may be used.

3.3.1 Resources available

There is no dedicated documentation to describe each resource's attribute. However, we can access the documentation for the python function (sigma67 2020). In addition to the three types of resources, on this library, we can find a 4th one that corresponds to a video. Indeed, YouTube Music allows you to see music videos from the YouTube platform. The data set used for each resource is defined in tables Table 7,
Table 8, Table 9 and Table 10.

Table 7. Essentials attributes from YouTube Python Library for an Album

Key	Description
type	The record type of the album (EP / ALBUM / etc...)
browseId	Identification of the album to take details of the album in the API
audioPlaylistId	Id of the content of the album to show in YouTube and YouTube music app → https://music.youtube.com/playlist?list=**{audioPlaylistId}**
thumbnails	Array of cover of the album
title	Name of the album
year	Year of the release **Note**: it is not possible to have more accurate value
artists	Array with artists of the album **Note**: id and name only are referred
tracks	Array with tracks of the album **Note**: only accessible by the endpoint: /album/:browseId

Table 8. Essentials attributes from YouTube Python Library for a Track

Key	Description
album	Album where the track is referred **Note**: id (browseId) and name only are referred
artists	Array with artists of the album **Note**: id (browseId) and name only are referred
title	Name of the track
duration_seconds	The track length in seconds
videoId	ID for the track video to take details of the track and to open the track in YouTube and YouTube Music App.
track_position	Index of the array of track **Note**: only accessible by the endpoint: /album/:browseId
disk_number	Not existing Maybe use the index of the array of track in the: /album/:browseId

Table 9. Essentials attributes from YouTube Python Library for an Artist

Key	Description
channelId	The ID for the YouTube channel of the artist.
thumbnails	Array with background of the channel
name	The name of the artist.
id	The id of the channel (Name – Topic) automatically generated by YouTube Music

Table 10. Essentials attributes from YouTube Python Library for a Video

Key	Description
artists	Array with artists of the album **Note**: id (browseId) and name only are referred
title	Name of the track
duration_seconds	The track length in seconds
videoId	ID for the track video to take details of the video and to open the track in YouTube and YouTube Music App.

3.3.2 Access data

The creator of the python library has created different functions to communicate with the API (sigma67 2020). These functions are the same finality that the endpoints from Spotify and Deezer. I mean, it’s the basic access to the different resources. The detail is available on the Appendix 3. Essentials functions of the YouTube Python Library.

There are some endpoints I will use from the YouTube API that allowing me to interact with the user's content. Each endpoint requires the **part** parameter in the URL that is a comma-separated list of channel resource properties that the API response will include. The table is defined on the Appendix 4. Essentials Endpoints use from YouTube's API.

3.4 Database

Through my studies I was able to follow several courses concerning databases. The approach I used to build the database is the following: ER Diagram, Relational Schema and Physical Schema. The first one is a conceptual approach that allows me to first define the data in the different entities. And to link them together with associations. I used the UML language to make the different associations. On Figure 1, you can see the result of the data separation.

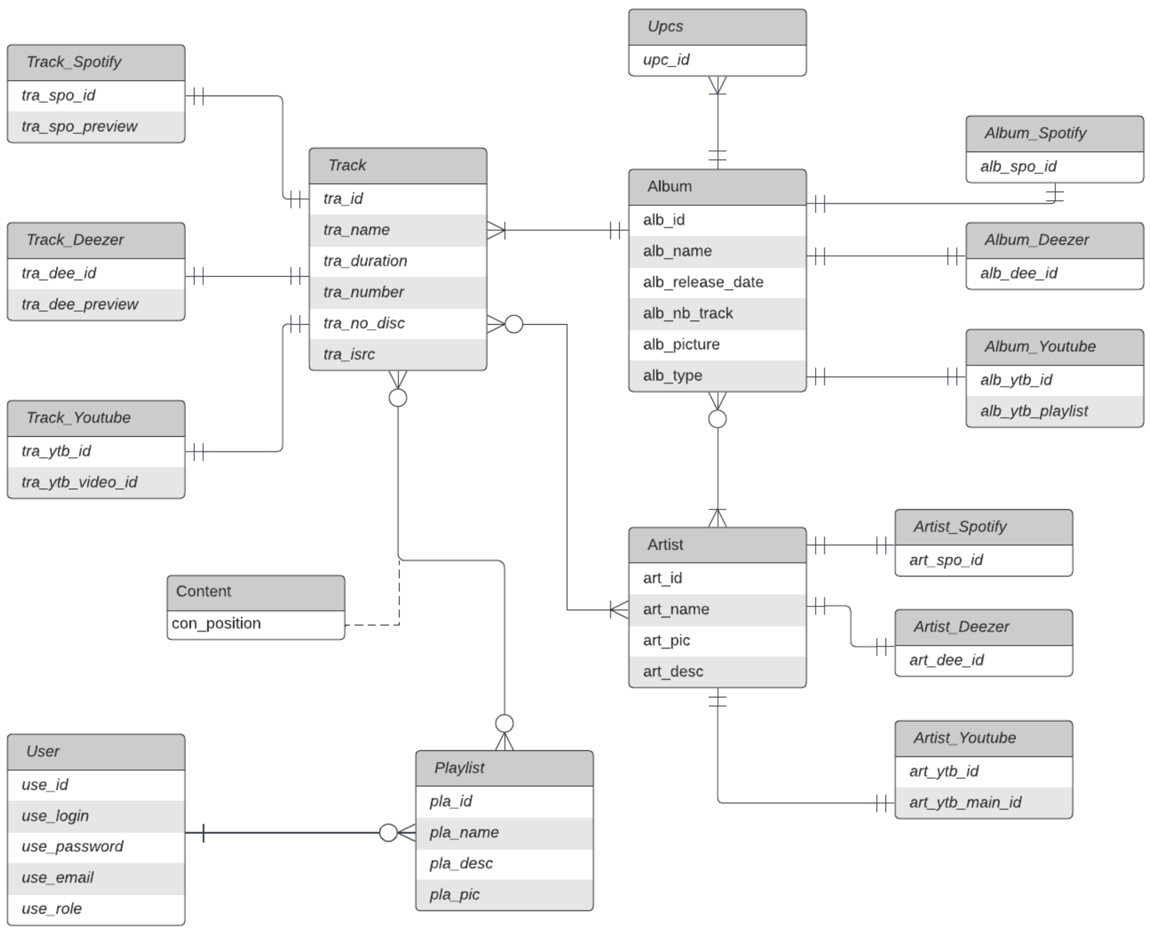


Figure 1. Conceptual Data Model

The second step is to transform these different entities into a relationship. It is in this part that we will add the different primary and foreign keys. It is also in this part that we will create the future association tables according to the cardinalities (see Figure 2).

Users(**use_id**,use_login,use_password,use_email,use_role)

Playlists(**pla_id**,pla_name,pla_pic,***#pla_use_id***)

Content(**con_position,*#con_tra_id,#con_pla_id***)

Tracks(**tra_id**,tra_name,tra_duration,tra_number,tra_no_disc,tra_isrc,***#tra_alb_id***)

Tracks_Spotify(tra_spo_id,tra_spo_preview,***#tra_spo_tra_id***)

Tracks_Deezer(tra_dee_id,tra_dee_preview,***#tra_dee_tra_id***)

Tracks_Youtube(tra_ytb_id,tra_ytb_video_id,***#tra_ytb_tra_id***)

Upcs(**upc_id**,***#upc_alb_id***)

Albums(**alb_id**,alb_name,alb_release_date,alb_nb_track,alb_picture,alb_type)

Albums_Spotify(alb_spo_id,***#alb_spo_alb_id***)

Albums_Deezer(alb_dee_id,***#alb_dee_alb_id***)

Albums_Youtube(alb_ytb_id,alb_ytb_playlist,***#alb_ytb_alb_id***)

Artists(**art_id**,art_name,art_pic,art_desc)

Artists_Spotify(art_spo_id,***#art_spo_art_id***)

Artists_Deezer(art_dee_id,***#art_dee_art_id***)

Artists_Youtube(art_ytb_id,art_ytb_main_id,***#art_ytb_art_id***)

Track_Artist(***#tra_id,#art_id***)

Artist_Album(***#art_id,#alb_id***)

PrimaryKey | ***#ForeignKey***

Figure 2. Logical Data Model

The last step is to transform this into a physical model, i.e., the creation of tables in the database with the different associations based on the model. It is in this step that we will set up the type of each attribute and define which fields are mandatory for an insertion. The final visual is available in Figure 3. This also allows us to establish the data dictionary. In this part, which is in Appendix 5, the aim is to see the whole data structure. I have also added a short description of each table to better understand its role.

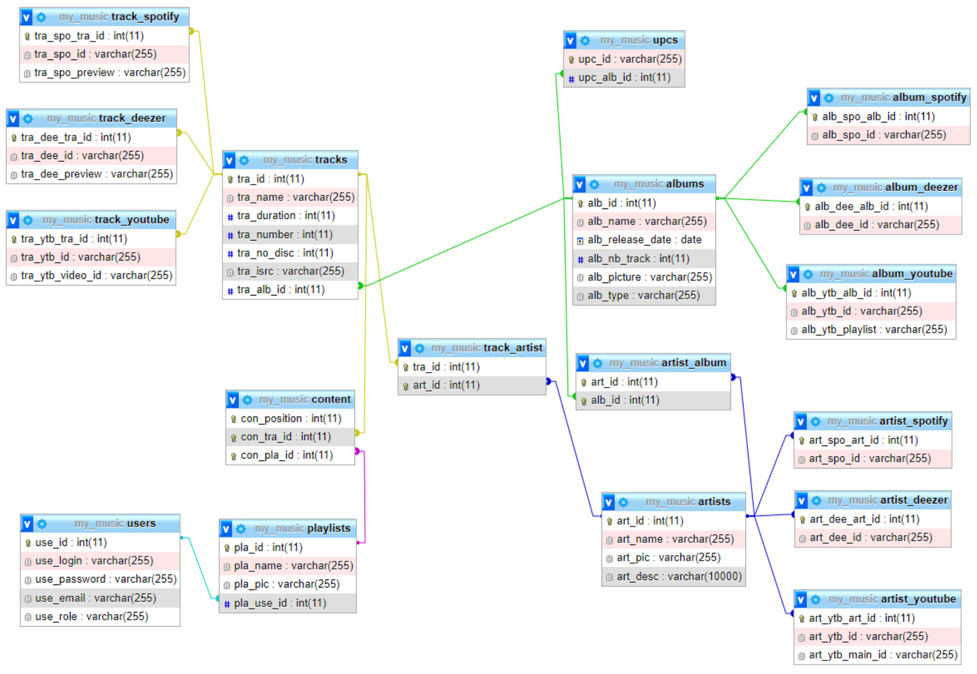


Figure 3. Physical Data Model

3.5 Server

3.5.1 Libraries

Node.js

Node.js is an open-source, cross-platform, back-end asynchronous JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser (Wikipedia 2022d).

I discovered server development with Node.js during this semester and I chose to use it to manage the back-end of my application. After comparing the different technologies where I have knowledge (PHP/Apache) or (Java/Spring Boot) I preferred Node.js, because in the context of my project, there are a lot of requests and its ability to be non-blocking and asynchronous compared to Spring Boot and PHP which will have more difficulty to manage this convinced me (see Figure 4). Moreover, with its flexibility unlike PHP, which is more web oriented, it will be easier to integrate my Node.js server with my mobile application.

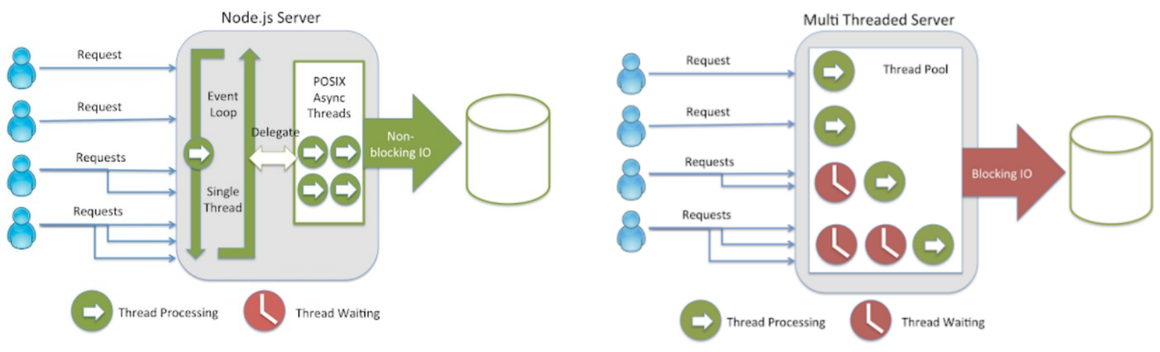


Figure 4. Single thread vs multi thread server (Chapter247 2020)

Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications (Express 2017). Because of its popularity in the Node package manager, and its ease of use, Express is for me the best way to easily create an API and that's why I used it.

Cors (Cross-origin resource sharing)

CORS is a node.js package providing a Connect/Express middleware that can be used to enable CORS with various options (cors 2019). Through the mobile application, it will be necessary to make fetch calls to the server in order to retrieve the data. By default, the JavaScript code of an application running in a browser can only communicate with a server of the same **origin** (**Same-origin policy**). By adding CORS, the server allows requests from other **origin** to access the server resources.

Sequelize

Sequelize is a modern TypeScript and Node.js **ORM** for Postgres, MySQL, MariaDB, SQLite and SQL Server, and more (Sequelize 2022). In order to facilitate the exchanges with the database and to save time in the development, I chose to use Sequelize because I have some basic knowledge with this tool.

Mysql2

Mysql2 is a library allowing me to use a MySQL database in my project. I wanted to have a relational database for this project and during my studies the one I used the most was MySQL. This library works in collaboration with Sequelize.

Winston

Winston is a library that allows you to make logs of what you want and to save it in a file. I was able to use it in one of my courses this year and it was for me essential to make logs in particular to be able to follow the analysis of the various data in order to see how the server will proceed to connect the data of the various platforms, to see step by step the verifications and to have a traceability of what worked and what didn’t work.

Jaro-winkler

Jaro-Winkler is a library which, as its name indicates, uses the Jaro-Winkler distance which allows to see the similarity between two strings. Indeed, during the development, I could see that some comparison variables between platforms such as the name of the artist was sometimes not written in the same way from one platform to another. That's why a strict comparison was not possible. The Jaro-Winkler distance returns a result between 0 and 1 where 1 represents a total equality between the two strings.

Jimp

Jimp for JavaScript Image Manipulation Program is an image processing library. It allows to read an image, to resize it, to change its quality or to recreate a new image in the filing system. This library allowed me to have another variable of comparison which is the "cover" of an album. Indeed, Jimp allows me to compare images and to see the percentage resemblance between them. After stumbling upon this article, I thought it could be interesting to use it and it was quite easy to use (Webb 2020).

Node-fetch

Being a JavaScript developer, the use of fetch was normal for me and my surprise was when I noticed that it was not implemented natively. So, I used this library to be able to make HTTP requests on the different APIs used for my project.

Async

Async is a utility module which provides straight-forward, powerful functions for working with asynchronous JavaScript (Async 2022). During the development of my project, I encountered a lot of performance and execution time problems of the different data analysis processes. I found this library which allowed me to implement asynchronous parallel processing and to optimize the analysis performances (see Thread).

Nodemon

Nodemon is a tool that helps developing Node.js based applications by automatically restarting the node application when file changes in the directory are detected (Nodemon 2022).

3.5.2 Mock-up

In this part, I will detail all the mock-ups I have made for the server part of the application. These pages will be accessible via a web browser and the goal is to be able to share content with users who do not use the application directly with a URL.

The first page is the share page. This page allows content to be shared via the server with all the references available for the mentioned resource. It can very well be an artist, a song, or an album. Each button redirects to a platform and points directly to the mentioned resource. In the event of an error, it is possible to report it using the button at the bottom right (see Figure 5).

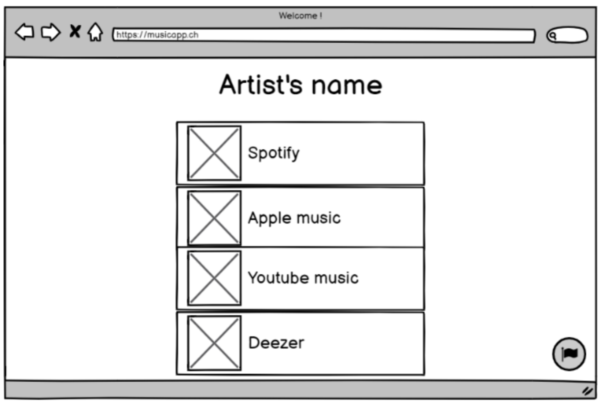


Figure 5. Mock-up Share page

The following mock-ups are different ways of displaying this page through different modal windows. The report popup appears when a user clicks on the flag at the bottom right. It is used to describe the problem encountered on this page (see Figure 6).

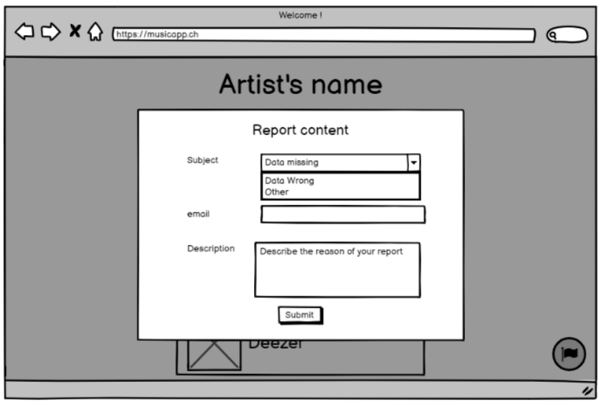


Figure 6. Mock-up Popup report

The second modal window appears when the resource is incomplete. It will then analyse the missing resources in order to complete it (see Figure 7). Sometimes, once the analysis is done, the confirmation window may appear when the analysis has found more than one possible result. It allows the user to confirm the result if it is correct. For example, two artists may have the same name (see Figure 8).

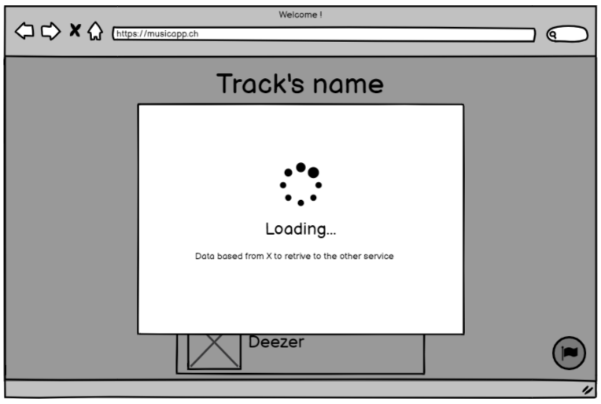


Figure 7. Mock-up Analysis popup

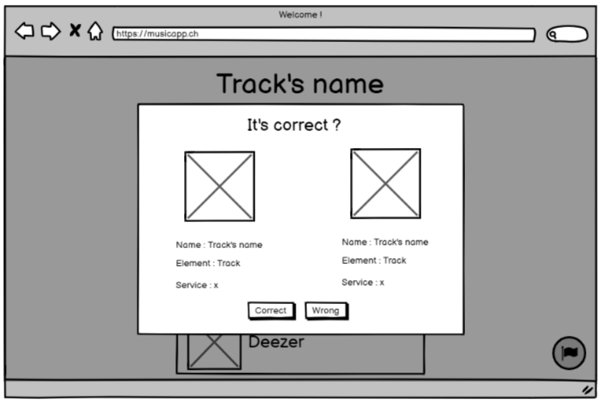


Figure 8. Mock-up Confirm page

3.6 Mobile App

3.6.1 Libraries

Here is the list of libraries I used in addition to those added by default when creating the expo project.

react-redux

Redux is an open-source JavaScript library for managing and centralizing application state. It is most commonly used with libraries such as React or Angular for building user interfaces (Wikipedia 2022e).

@reduxjs/toolkit

The Redux Toolkit package is intended to be the standard way to write Redux logic. It allowed me to use redux in a simpler way with a faster and more comprehensive set up.

@react-navigation/native ,native-stack & bottom-tabs

React Navigation is one of the most well-known navigation libraries available for React (LogRocket 2021). I used this library because the application will need multi-screen. I had the occasion to use it several times during my studies.

react-native-elements

React-native-elements is a framework in React Native allowing to create more advanced visual components. I used it to make my application more visually pleasing.

react-native-paper

React Native Paper is a high-quality, standard-compliant Material Design library that has you covered in all major use-cases (Paper 2022). As for react-native-elements, I also used this library to make the design of the application cleaner.

react-native-modal

The goal of react-native-modal is expanding the original React Native <Modal> component by adding animations, style customization options, and new features, while still providing a simple API (Github 2022a).

react-native-webview

React Native WebView is a modern, well-supported, and cross-platform WebView for React Native. It’s intended to be a replacement for the built-in WebView (which will be removed from core) (Github 2022b). It allowed me to integrate a web page allowing a user to authenticate himself according to the platform he has chosen.

react-native-youtube-iframe

This library allows to insert a YouTube video iframe on React Native. I used it to be able to insert the different video content found on the YouTube platform in order to be able to play them directly in the application.

3.6.2 Mock-up

The login page is the first one the user sees when opening the application. It offers to login using one of the platforms in order to use the application. It is possible to use it as a guest with the bottom button. At the first start-up (depending on the button chosen), a login popup will appear allowing entry of the username and password (see Figure 9).

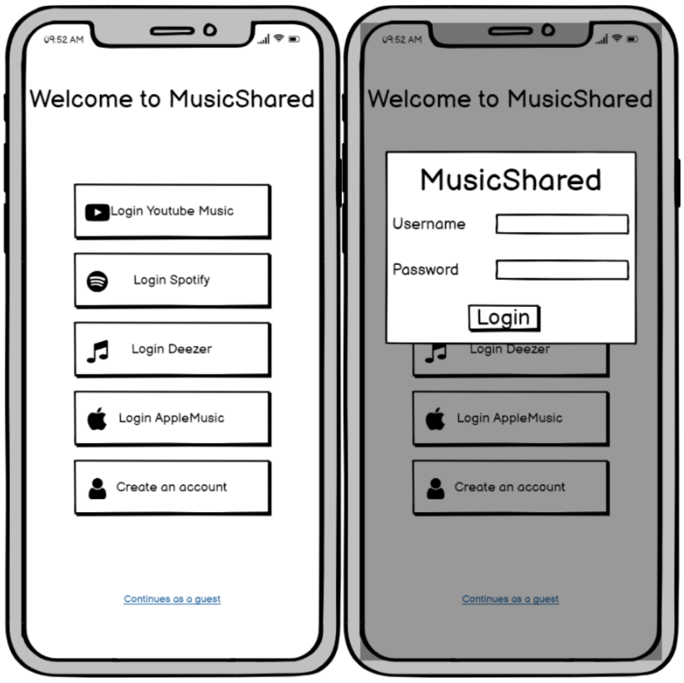


Figure 9. Mock-up of login page

The second page is the account creation page. If the user does not have an account, this page allows him to create one on one of the supported platforms (see Figure 10).

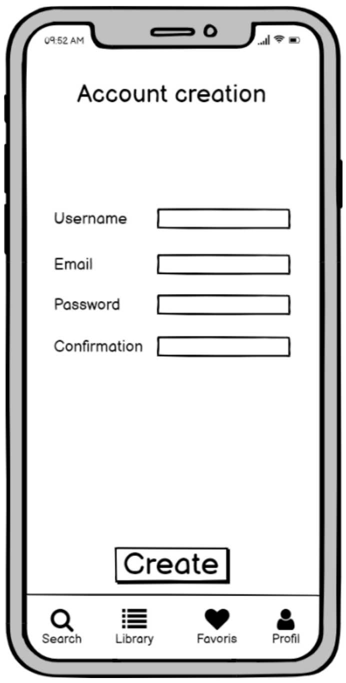


Figure 10. Mock-up of creation page account

Once logged in, the user can access his profile page which allows him to have an over-view of your account: service used, profile photo and the possibility to change the pass-word (see Figure 11).

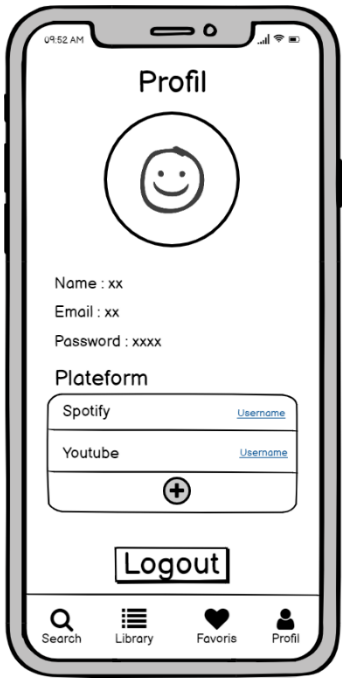


Figure 11. Mock-up of the account detail page

The search page allows the user to search for music. It is by this process that the database will be fed. A drop-down menu allows you to choose the platform on which to search (see Figure 12).

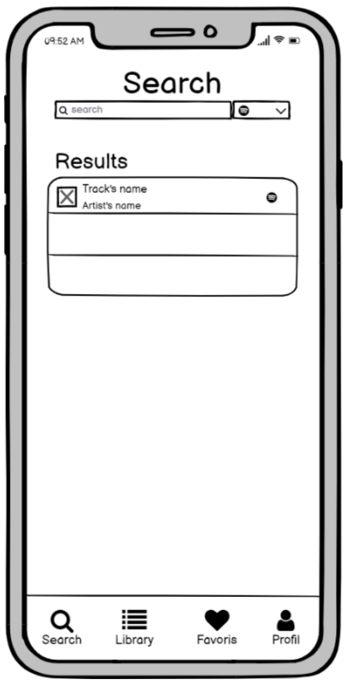


Figure 12. Mock-up of the search page

The resource pages provide a detailed view of the resource you are looking for. It can be an artist or an album (see Figure 13).

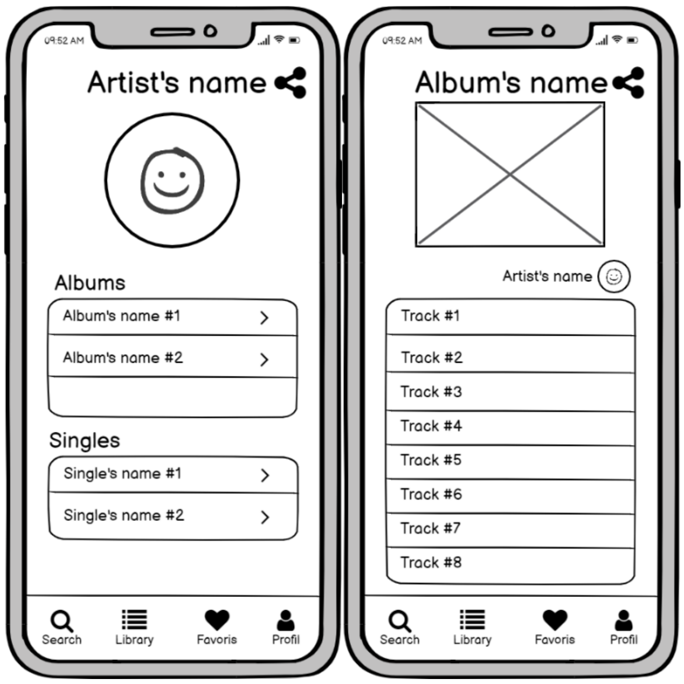


Figure 13. Mock-up of detail artist and album

The playlist page allows a user to see the content of a playlist. It is possible to add a new song to the playlist which will then open up a search page as a popup (see Figure 14).

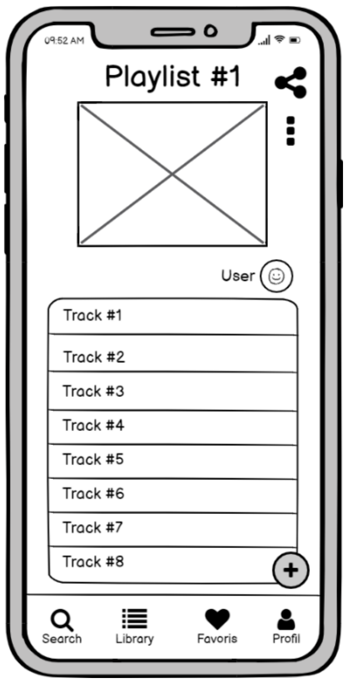


Figure 14. Mock-up playlist page

The library page provides an overview of the user's playlists. A user can favorite an album or an artist in order to have direct access from his favorites page (see Figure 15).

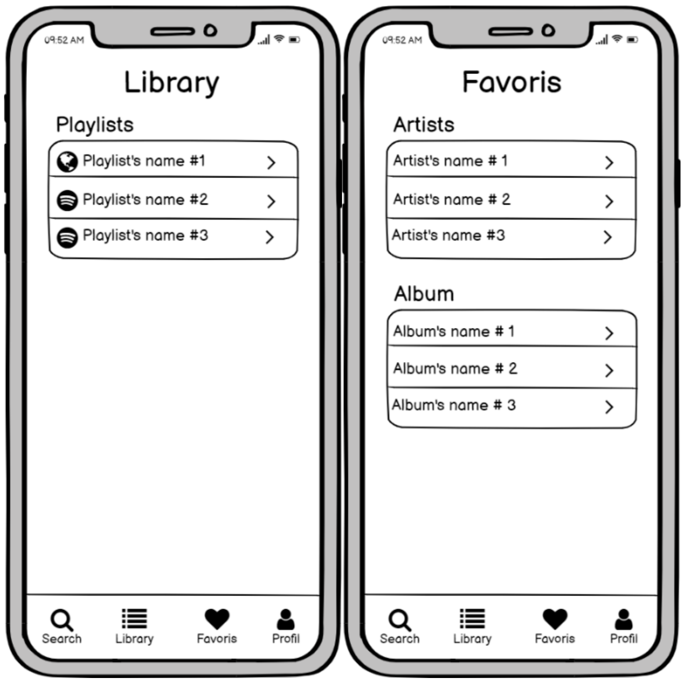


Figure 15. Mock-up library page

3.7 Technology used

Singleton

The Singleton design pattern allows to reduce a class to a single instance and to use the same instance in several places in the code. Most of the time, this design pattern is used for authentications to another system (for example a database or an API) in order to avoid opening several similar connections to a system. It is in particular to connect to the Spotify API that I implemented this design pattern.

ORM

An ORM is a type of computer program that interfaces between an application program and a relational database to simulate an object-oriented database (Wikipedia 2022f). Concretely, it works with a model class that will represent a table in the database (see Figure 16). It is possible to define attributes and specificities related to the database such as primary and foreign keys, data type and size. It is also possible to make associations between models.

Then a connection to the database must be set up in order to be able to generate tables based on previously defined models. An ORM also provides a query system allowing to retrieve data from the database and to format them automatically in a model class instance.

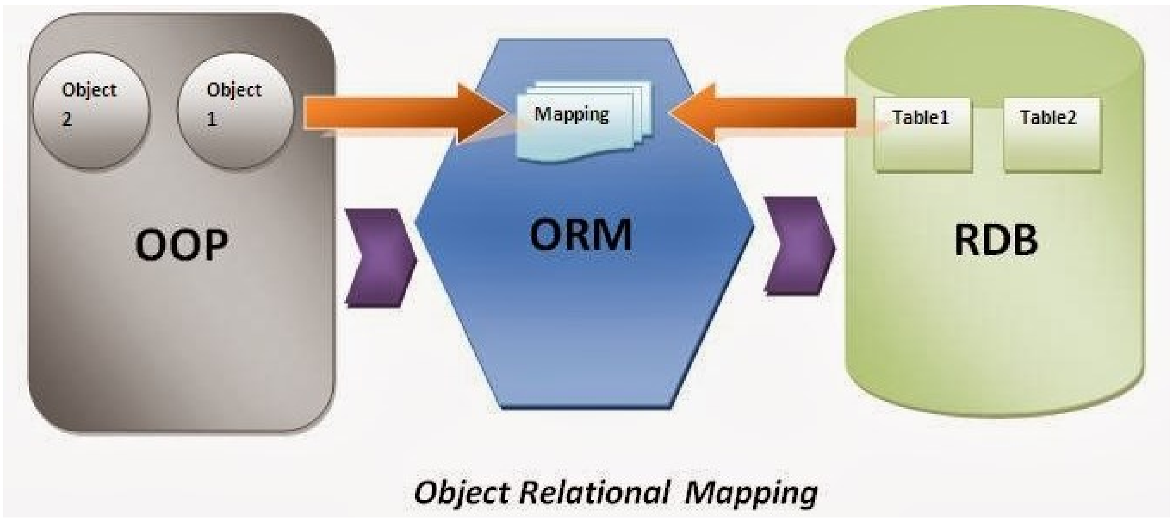


Figure 16. Representation of an ORM (Interviewquestionjava 2014)

Repository Pattern

A repository is a class that defines the methods that will make the link between the models and the database. It allows to translate the logical business of the application into queries to the database. The goal of this approach is to centralize in the code the different accesses to the database. It’s illustrated with this diagram (see Figure 17).

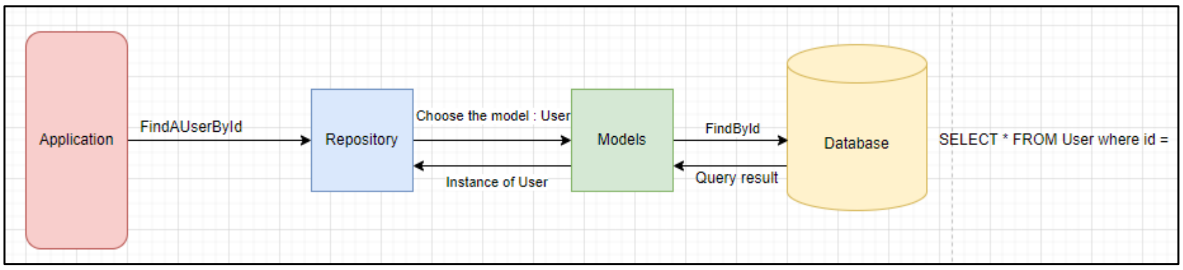


Figure 17. Illustration of how a repository works in my application

Concurrent processing

Concurrent processing is a computing model in which multiple processors execute instructions simultaneously for better performance. Concurrent means something that happens at the same time as something else (TechTarget 2005). As mentioned above, it is with the Async library that I could use this technology on my server.

Inheritance in database

On my database schema, there is a notion of inheritance allowing to define, for example, a simple track (name, duration and collaborator) and a track coming from Spotify where we add additional attributes such as its identifier on the platform. I was able to experiment with inheritance with Hibernate which is another ORM during my studies. Hibernate allows you to choose between several inheritance strategies in a database: (Mapped Superclass, Table per Class, Single Table or Joined) (Janssen 2022). So, I looked for a solution to implement the Joined strategy with Sequelize and I simply made One to One relationship between the primary keys of the SuperClass and the SubClass.

4 Empirical part Data Analysis

In this chapter, I will transcribe the analysis result I arrived at after studying each platform, followed by a section where I will review the differences between each of them. I will also explain how the authentication system works within each platform according to the respective documentation.

It's critical to comprehend the data structure. When I say data structure, I'm referring to how the platform makes its data accessible through the API. Indeed, as previously said, the database will be filled mostly through user searches. In this step, besides reading the documentation, I tested the different possible endpoints a bit randomly. With my personal knowledge, I know some songs or albums removed from the platforms and I could see how to manage these removed data depending on the target platform.

4.1 Specification implementation Spotify

4.1.1 Search result

When we examine the results of a song search, we can see that we have quick access to all of the song's metadata as well as the artists that composed it. It's possible sometimes that the **preview_url** field is null.

In the **album** field, there is also a lot of data available except for the other songs linked to this album which can be accessed via the album **id** with another Endpoint. Because the author of a song is not always the same as the author of the entire album, there is an **artists** field within the album data associated with the song.

The album filter allows you to see the connection between an album and its performer. The list of songs will also be available through the Endpoint / v1/albums/{id}. However, the **external_ids** attribute is missing from the accessible songs of the album, which could be useful to exploit during the process of linking data. There are albums with more than 50 tracks, and because the restriction is 50 tracks, you'll have to make two requests to get all of them.

The data will not be retrieved if a search by artist is did. Indeed, because it will not be linked to a song or an album during the search method, it is difficult to enter this information into the database using just this information. Because one artist type res-source may have the same name as another, this type of search might be difficult to exploits.

4.1.2 Data duplication

When you search for a specific music, the search engine proposes the same song multiple times, but from various albums (see Appendix 6). This is conceivable in some situations; however, most compilation albums (such as Summer 2020 Hits) are compilation albums, making it difficult to populate the database. To remedy this problem, the **album_type** column makes it easy to determine if it's a compilation or not, but it's not always stated that way. This difficulty can also be solved by glancing at the name of the album's author, which is "Various Artist" (see Appendix 7).

It's possible, though, that this is a "genuine" collection and that the tracks aren't duplicates (see Appendix 8). A verification by ISRC code can be processed with another service to attest that entity is not existing in another album.

Sometimes an artist will release a "fake reissue" of his album. What I mean by this is a new record with the same name, cover, and only one additional sound, for example. The "reissue" in this situation is a new resource, resulting in duplicates. When you search for an album, all available versions are displayed (for more information, see Appendix 9).

The album with the most recent date is the one that is valid on Spotify's mobile app. From the album details page on Spotify site, you can select the one you want. Apart from the name of the album, the label associated with it, and the artist linked with it, no one data can be used to link the two versions. There are no other distinguishing characteristics.

4.1.3 Deleting data

It’s not uncommon for some music to be taken off the platform. The resource in question can no longer be listened to, and the latter's referencing has been disabled. It's also possible that a song or album is in the same scenario, but we may still listen to it because the program instantly refers us to a duplicate of the accessible music. The difference between the two can be examined at the data level using the Endpoint "/v1/tracks/{id}" with the option **market**, which will return a different id if the song is accessible. If a **restrictions** attribute is returned, it signifies that it’s not available.

Unlike the other valid data types, the resources remove returns an empty array on the **available_markets** attribute (if the **market** parameter is not used).

4.2 Specification implementation Deezer

4.2.1 Result of a search

Searches utilizing a filter, unlike Spotify, are divided by different end points. Song searches do not provide all of the necessary information to perform an insertion. All artists associated with the song and artists associated with the album, for example, are absent. As a result, you will need to call other endpoints to complete the missing information for each result.

The artists and tracks associated to the album are not directly available from the result when searching by album. As a result, in order to complete this data, you must also go to the album's End-point detail. There is no restriction to the amount of tracks that can be viewed in the album's Endpoint information if the album includes a large number of tracks.

4.2.2 Data duplication

The same situation happens in terms of repetition as with the Spotify API. That is, when looking for a music, there are duplicates. There are also compilations with "Various Artist" as the artist associated with the album, however the **record_type** column does not distinguish between an album and a compilation. Similar compilations between Spotify and Deezer may exist, and the question is whether or not they will be saved in the database (for more information, see Appendix 10).

There's also this "fake reissue" execution this time on Deezer, where both albums are shown as two separate albums in the discography from the app and the web page. Looking at the MD5 of the record cover may reveal a link. Only one of them will be featured on the artist page if their values are the same (for more information, see Appendix 11).

4.2.3 Deleting data

The variable **available_countries** can be used to determine whether or not a song has been deleted. Because this field isn't defined in an album resource, it's not possible to tell which albums are cited (by the search engine) and which aren't.

4.3 Specification implementation YouTube

4.3.1 Outcome of a search

When a user conducts a search, all of the results are displayed in one occurrence. Although "fake reissue" is present, the search engine ensures that it is removed. Duplicates do occasionally appear, but this has only happened once, so I believe it is a YouTube bug (see Appendix 12).

The search result is returned in a single array via the library. It is possible to determine the resource type of an element in the array using the **resultType** field. After several checks, I don't see any particular reason for the selection of duplicate tracks, whether it's based on the popularity of the track with the number of views, the release date, the popularity of the album, or even the one with the most views before or after a potential clip if one exists. To observe the differences between each duplication, I created a comparison table for two songs (see Appendix 13).

When searching for an album, the referencing is different. An album can be released as a reissue, but is not necessarily related to its predecessor. In this case, both entities are referenced by the search engine (see Appendix 14). On the other hand, if there is a link between these entities, only one reference will be put forward (see Appendix 15).

When I use the search engine and type in the videoId of a piece, it will be displayed if it is already referred by the search engine, otherwise an empty result will be displayed (for more information, see Appendix 16). I was able to compare the data of two similar songs one of which is referred by the search engine and the other is not. But it's not 100% accurate to perform a search like this because sometimes you receive another result that the videoId mentioned.

4.3.2 Data duplication

It's difficult to tell if compilation albums exist because the engine doesn't reference duplicates. However, it is not essential to rule out this option, and by conducting a search, I was able to locate one. In the case of the other services, the term "Various Artist" is used to identify the album's author (see Appendix 17).

4.3.3 Deleting data

Because some songs are disabled on other platforms, the API does not provide an identity for these songs. As a result, these tunes are inaccessible and unlistenable (they are set to private). This will have to be taken into account while processing the data (see Appendices Appendix 18 and Appendix 19 for more information).

4.3.4 Link between different resources

The **get_album** function can be used to acquire the album's individual tracks. Unfortunately, the album cannot be obtained using the **get_song** method. That's why, in order to access the album ID associated with the tune, I've hijacked the API used by YouTube music on the web browser (see Get track v2).

4.3.5 Resource Implementation

Each album is available in two versions: one with all of the tracks but no YouTube videos, and another with YouTube videos instead of tracks with a video on the YouTube platform. This could be a sound clip or an official recording (see Appendix 20 for further information).

The **get_album** method in the Python library allows me to receive a list of songs that have video clips, but it does not allow me to retrieve the first version. It is feasible to access album data in by hijacking the API that YouTube employs on the browser once more (see Get album v2).

There are also two YouTube channels linked to a particular artist. The one where the artist uploads these video and the one that YouTube Music automatically generates. The latter is known as <**ArtistName**> - Topic and is used for all song and video-related resources. The **get_artist** method can be used to acquire the artist's channel ID, which is stored in the **channelId** field. The id can also be obtained from the **videoDetails.channelId** field using the **get_song** method.

4.3.6 Clip/Video

Because the information is shared with YouTube, it's likely that a song will be paired with a clip or its YouTube video equivalent (Official Audio for example). It's possible that this isn't always done appropriately. A clip can be linked to several songs, while a song is linked to a single clip.

4.3.7 References are occasionally incorrect

References to authors are frequently omitted. The majority of the time, only the lead artist is acknowledged. It's also conceivable that the artist mentioned isn't the one you're looking for.

4.4 Brief

4.4.1 Content that is currently available

Some contents are available on public platforms, while others are no longer available or are hidden but can still be accessed via a direct connection, as I've noticed (see Appendix 21 for more information).

One approach would be to start with a default platform and then go on to another if the content isn't available there.

To continue with the material, some content on YouTube is structured differently than on other platforms. To be honest, it's unlikely, but it's possible that the albums aren't complete (see Appendix 22).

A "certification" system that distinguishes between well-formatted and unformatted data would be the solution. Because this is an unusual occurrence, the unformatted ones will be displayed, and manual processing will be possible.

In comparison to the other platforms, YouTube has the most varied implementation. It will be considered last in the content search process at the level of data linkages during development.

4.4.2 Link between albums

Each album stands on its own. There may be ties between different albums depending on the platform, such as a reissue of the latter, however this is platform specific.

Each API implemented makes no indication of a connection between these various entities; it is sufficient not to link them at the database level and to keep these resources separate (see Appendix 23).

4.5 Authentication process

4.5.1 Spotify

set up the environment, from the Spotify dashboard, I had to create a new application that gave me a client ID needed to use the API. To set up personal authentication, I use the authorization code flow (OAuth 2.0). To explain simply, this is a security process where the app calls Spotify service with the required authorization, and the service returns a prompt to let the user type in the username and password. After Spotify redirects to a safe website (defined in the application settings in the dashboard) with a code. We need to call the Spotify API again with this code which gives a token. Now we can use the API. It is also possible to use the API without personal authentication but, we have limited access which is sufficient in most cases of my project (Spotify 2022d).

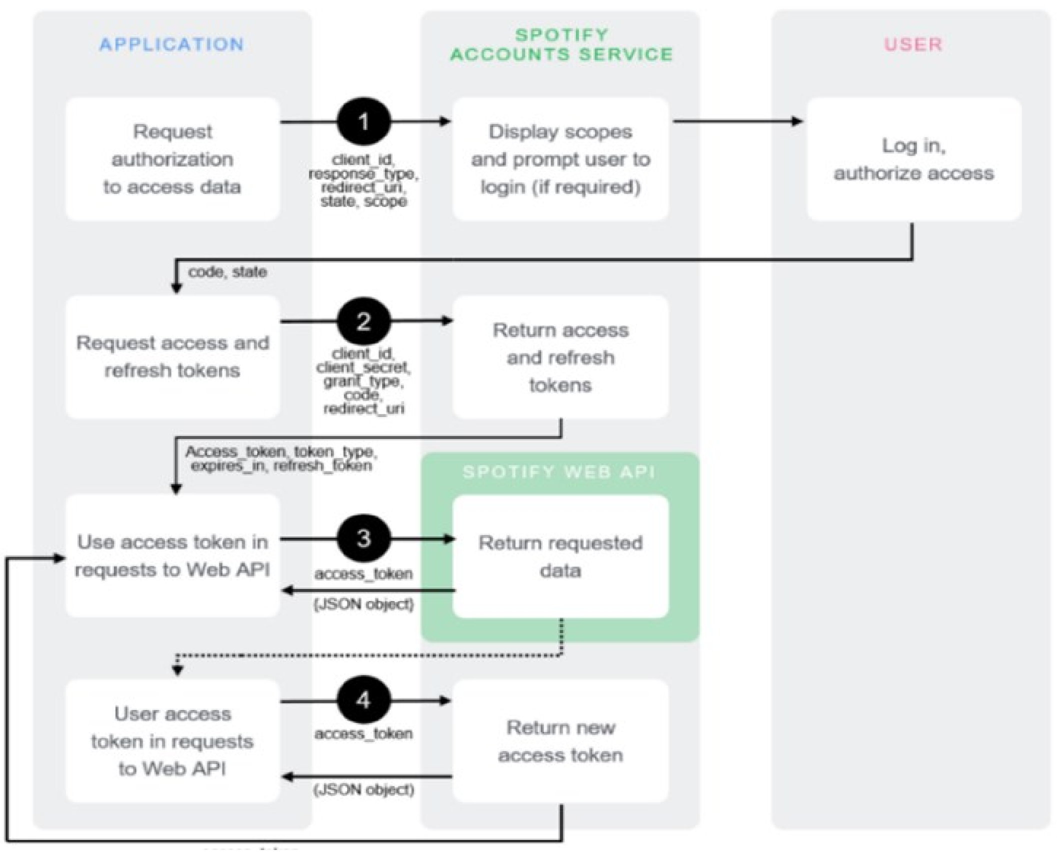

APPLICATION
SPOTIFY ACCOUNTS SERVICE
USER
Request authorization to access data
1
client_id, response_type, redirect_uri, state, scope
Display scopes and prompt user to login (if required)
Log in, authorize access
code, state
Request access and refresh tokens
2
client_id, client_secret, grant_type, code, redirect_uri
Return access and refresh tokens
Access_token, token_type, expires_in, refresh_token
SPOTIFY WEB API
Use access token in requests to Web API
3
access_token
{JSON object}
Return requested data
User access token in requests to Web API
4
access_token
{JSON object}
Return new access token

Figure 18. Process of authorization code flow

4.5.2 Deezer

It's pretty much the same procedure for setting up authentication (based on OAuth 2.0) on the API as it is for Spotify. To begin, I must first establish a new application with a secret key and a client id. It is used to verify personal authenticity (Deezer 2022d).

For my thesis, I'll employ the Client-side Flow, which is broken down into three steps:

- **User Authentication**: Confirm the identity of the user.
- **App Authorization**: Deezer asks the user to agree to give my application permission to utilize the API.
- **App Authentication**: Verify that the user grants the permissions; the browser then redirects to the URL specified in the application's configuration, which includes an authorization number. You can use this code to request a token from the Deezer website, which will be returned as an access token.

This setup is not required for the basic endpoint (search, track or album details).

4.5.3 YouTube

I discovered a way to run python scripts on the Node JS server (Stackoverflow 2014). The next step is to figure out how to send parameters to the script and how to get the script's output value. We can run a python script in node JS using the child process package, and we can send parameters that will be accessible in the python script. I'm going to incorporate JSON to return the data because it's easier to maintain data in the backend.

For the authentication part, I followed the Client-Side Web Apps OAuth 2.0 process with the endpoints coming from the official YouTube API. To start, as for the other services, we have to create an application on the Google Cloud Platform site. At the end, we need to get a Client ID in order to set up the authentication (Google 2022).

The first step is to go to the specific google login page mentioning our Client id, the redirection site defined beforehand and the scope of the authorizations we want. The user connects and is redirected to our site directly with an access token that we can return to our mobile application. This token allows us to have access to the user's personal data according to the authorizations defined in the scope.

5 Empirical part Backend Server

The server is the main part of my work. Its role is to analyse the content of the different platforms, to make the link between the different entities and to record the result in the database.

Thanks to the previous chapter, I was able to get a clearer idea of the possibilities of setting up the different analysis processes. I will start this chapter by talking about the organisation of the project with an explanation of the file tree and the access point implementation structure. This will be followed by the details of the database implementation with the Sequelize ORM.

I will also explain the implementation of the access points on each platform using classes. Indeed, I have wrapped the different endpoints studied in the theoretical framework in such a way as to be able to manage errors within the application. It is once having set up these accesses that I will be able to explain the processes of analysis with in particular threading and to finish by the installation of Web interface and the definition of each endpoint to be able to reach the resources online.

5.1 Organisation

5.1.1 File trees

Assets

The assets folder contains various images used for the layout of web pages.

Controllers

This folder contains all the controllers of the application, there is one for each platform used.

Logs

It is in the logs folder that all the log files are saved. In the newArtist folder, there are all the logs for the analysis of the artists to see how the links are made. There are also log files for every error made when accessing each platform to facilitate maintenance.

Python

This folder contains all the python files that are used to access the data of the YouTube Music library.

Routes

The routes folder contains the definition of each route associated with an HTTP method (GET, POST, PUT, DELETE) and an access URL.

Scripts

The scripts folder contains all the scripts of the application.

Scripts/db

This folder contains all the scripts related to the database. Whether it is the models, repositories or the connection to the database.

Scripts/CustomError

This folder contains error classes that inherit from Error to be able to generate custom error messages.

Scripts/ utility.js

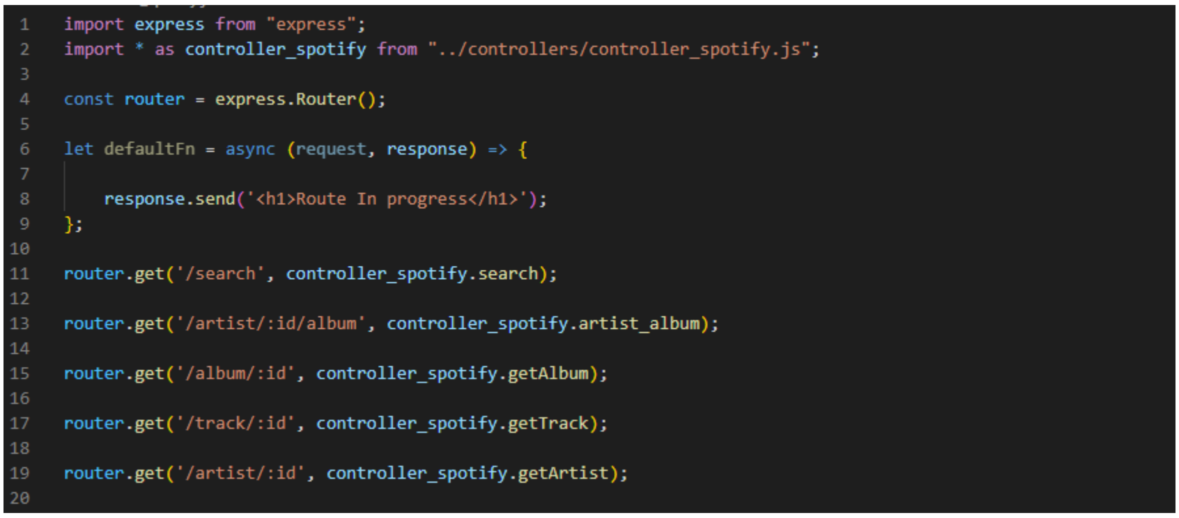
This file contains several utility functions that are used on the server. In particular, there are the comparison functions thanks to the Jaro and Jimp libraries.

View

It is in the view folder that the html code is created for the visual part of the application.

5.1.2 Routes

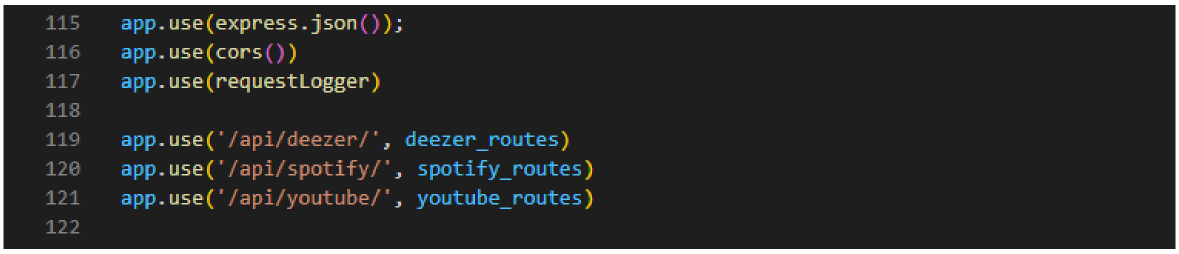
A route is a defined path to redirect HTTP requests from a client to the appropriate controller (Mozilla 2022). In order to access the data of the different APIs, each will have a route file with all the different endpoints available. It is interesting to separate the routing and controller part to gain readability in the code reading (see Figure 19).



```javascript
import express from "express";
import * as controller_spotify from "../controllers/controller_spotify.js";

const router = express.Router();

let defaultFn = async (request, response) => {

    response.send('<h1>Route In progress</h1>');
};

router.get('/search', controller_spotify.search);

router.get('/artist/:id/album', controller_spotify.artist_album);

router.get('/album/:id', controller_spotify.getAlbum);

router.get('/track/:id', controller_spotify.getTrack);

router.get('/artist/:id', controller_spotify.getArtist);
```

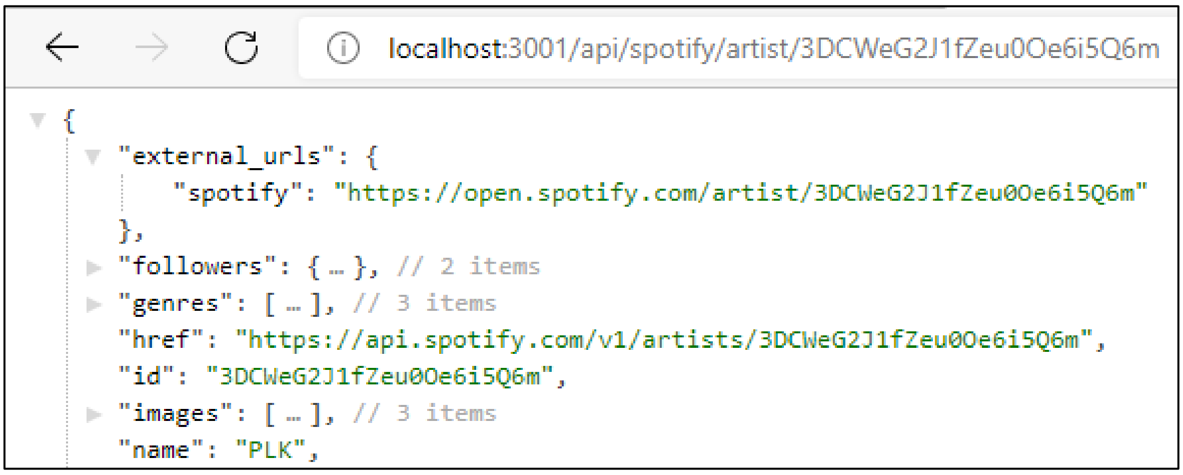
Figure 19. Example route file for endpoints available for access on the Spotify API

The next sept is to import these routes inside the app. All the routes are exported in the index.js file and implemented in the following way (see Figure 20). You can see an example of call from a browser in the Figure 21.



```javascript
app.use(express.json());
app.use(cors())
app.use(requestLogger)

app.use('/api/deezer/', deezer_routes)
app.use('/api/spotify/', spotify_routes)
app.use('/api/youtube/', youtube_routes)
```

Figure 20. Implementation of routes for each platform



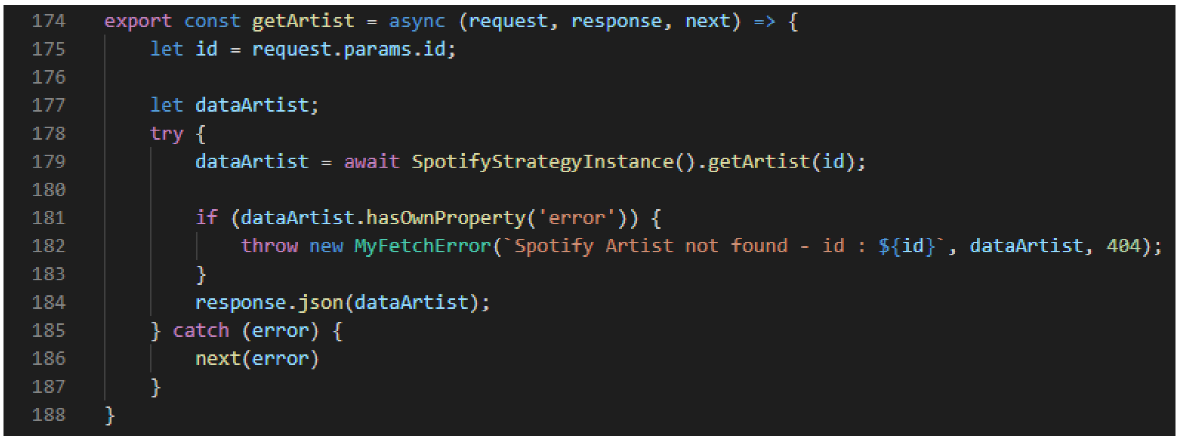
localhost:3001/api/spotify/artist/3DCWeG2J1fZeu0Oe6i5Q6m

```
{
  "external_urls": {
      "spotify": "https://open.spotify.com/artist/3DCWeG2J1fZeu0Oe6i5Q6m"
  },
  "followers": { … }, // 2 items
  "genres": [ … ], // 3 items
  "href": "https://api.spotify.com/v1/artists/3DCWeG2J1fZeu0Oe6i5Q6m",
  "id": "3DCWeG2J1fZeu0Oe6i5Q6m",
  "images": [ … ], // 3 items
  "name": "PLK",
```

Figure 21. Example of a route to access an artist resource in the Spotify API

5.1.3 Controller

A controller allows to make the link between the model part of the application and the routing part. It is in this file that the logical part of the application will be implemented. A function is triggered when a user takes the route associated with it (see Figure 22).



```
export const getArtist = async (request, response, next) => {
    let id = request.params.id;

    let dataArtist;
    try {
        dataArtist = await SpotifyStrategyInstance().getArtist(id);

        if (dataArtist.hasOwnProperty('error')) {
            throw new MyFetchError(`Spotify Artist not found - id : ${id}`, dataArtist, 404);
        }
        response.json(dataArtist);
    } catch (error) {
        next(error)
    }
}
```

Figure 22. Example of a function coming from the Spotify controller

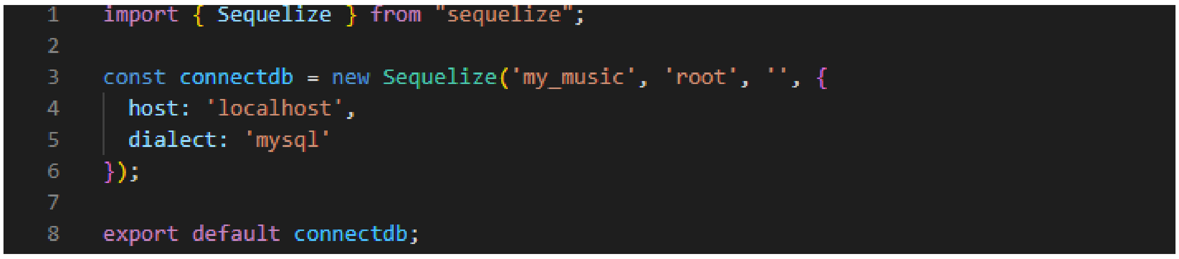
5.1.4 Errors management

This type of error is generated at the controller level when a call to an API does not return an adequate result. Its name is MyFetchError and It allows to generate an error in this case and to return to the client a JSON with the error encountered. It was important for me to handle these errors because it is through these routes that I will retrieve data during the analysis procedures.

5.2 Database

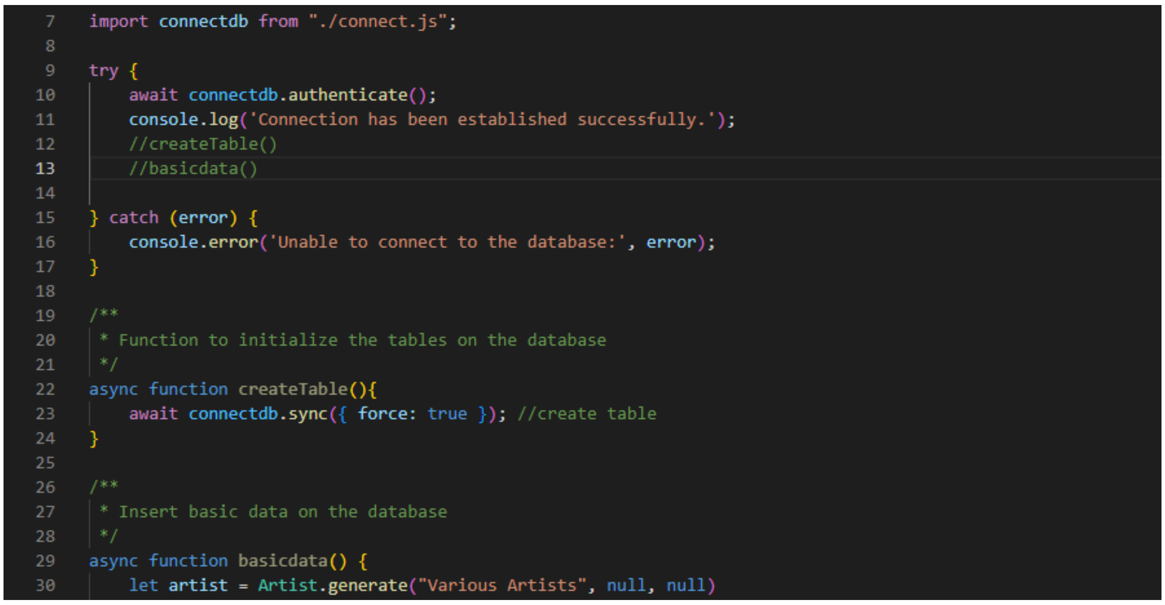
5.2.1 Connection

The connection is defined in the connect.js file (see Figure 23). It allows me with Sequelize in a first step to establish the connection to the database on MySQL with the file se-tup.js and in a second step to retrieve the data, from the repositories with the models defined in the models folder. From the setup.js file (see Figure 24), there is the **createTable** function which allows to create the tables in the database (at the initialization of the server) and a **basicdata** function which will make the basic insertions of the data necessary for the good functioning of the server.



```
import { Sequelize } from "sequelize";

const connectdb = new Sequelize('my_music', 'root', '', {
    host: 'localhost',
    dialect: 'mysql'
});

export default connectdb;
```

Figure 23. connect.js

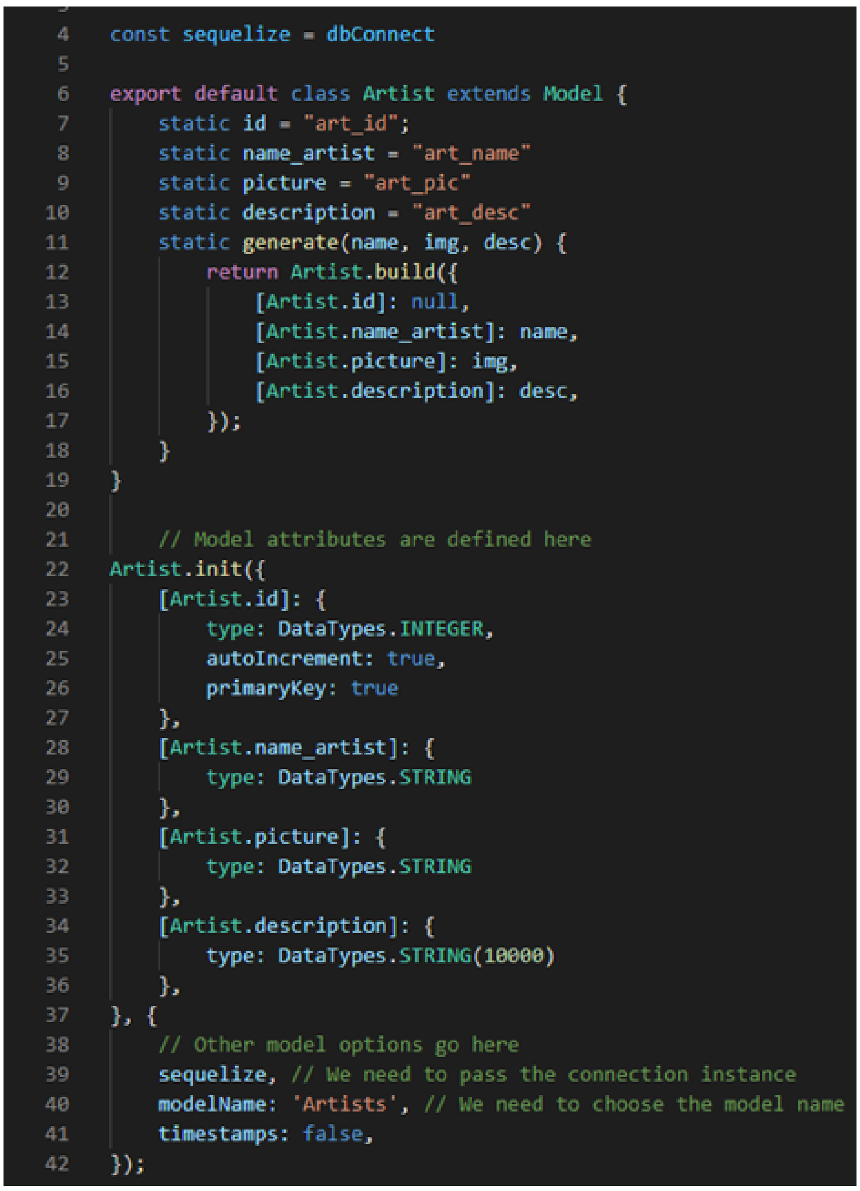


```
import connectdb from "./connect.js";

try {
    await connectdb.authenticate();
    console.log('Connection has been established successfully.');
    //createTable()
    //basicdata()

} catch (error) {
    console.error('Unable to connect to the database:', error);
}

/**
 * Function to initialize the tables on the database
 */
async function createTable(){
    await connectdb.sync({ force: true }); //create table
}

/**
 * Insert basic data on the database
 */
async function basicdata() {
    let artist = Artist.generate("Various Artists", null, null)
```

Figure 24. extract of setup.js file

5.2.2 Model

In my project, I created 18 tables that allowed me to separate the different types of data. In order to gain flexibility, the attribute names of the database are defined only once in static variables to avoid problems with attribute names in repositories or controllers. In this section, I will explain how a model is structured and the different associations I have used. Each class follows this same format, it inherits from the Model class of Sequelize and has static attributes with the names of the fields in the database. There is a constructor which is the **generate** function which creates an instance. To save it in the database, you must use the Repository class corresponding to the Model. An example of the artist class is defined in Figure 25.



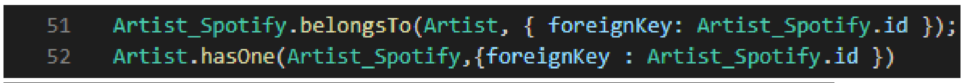
```
const sequelize = dbConnect

export default class Artist extends Model {
    static id = "art_id";
    static name_artist = "art_name"
    static picture = "art_pic"
    static description = "art_desc"
    static generate(name, img, desc) {
        return Artist.build({
            [Artist.id]: null,
            [Artist.name_artist]: name,
            [Artist.picture]: img,
            [Artist.description]: desc,
        });
    }
}

    // Model attributes are defined here
Artist.init({
    [Artist.id]: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        primaryKey: true
    },
    [Artist.name_artist]: {
        type: DataTypes.STRING
    },
    [Artist.picture]: {
        type: DataTypes.STRING
    },
    [Artist.description]: {
        type: DataTypes.STRING(10000)
    },
}, {
    // Other model options go here
    sequelize, // We need to pass the connection instance
    modelName: 'Artists', // We need to choose the model name
    timestamps: false,
});
```

Figure 25. Class Artist representing the artists table in the database

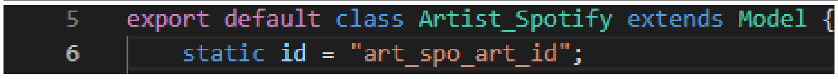
5.2.3 Associations

One-to-One

In this example, an artist is associated with a single artist_spotify and vice versa. To avoid overflow errors, the foreign key of artist_spotify is also a primary key.



```
Artist_Spotify.belongsTo(Artist, { foreignKey: Artist_Spotify.id });
Artist.hasOne(Artist_Spotify,{foreignKey : Artist_Spotify.id })
```

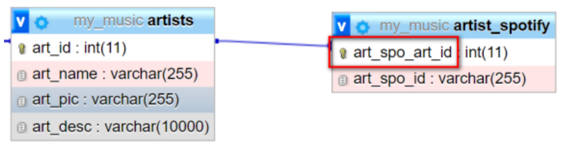


```
export default class Artist_Spotify extends Model {
    static id = "art_spo_art_id";
```

Figure 26. Instruction to make a One-to-One association with Sequelize

Only one of the following two lines (see Figure 26) is enough to make the association at the database (see Figure 27) level but it is different within the instances in the code. The

Artist.hasOne statement allows the Artist instance to access the Artist_Spotify instance if the model is joined during the query. While **Artist_Spotify.belongsTo** allows access to the Artist model if the model is joined during the request. An error is generated by Sequelize when accessing one of the models and the association is not implemented.

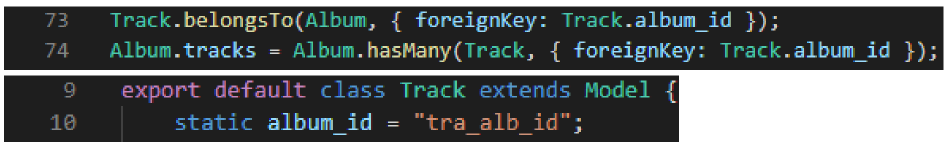

my_music artists
art_id : int(11)
art_name : varchar(255)
art_pic : varchar(255)
art_desc : varchar(10000)

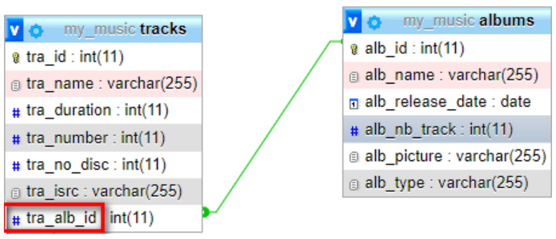
my_music artist_spotify
art_spo_art_id int(11)
art_spo_id : varchar(255)

Figure 27. Result of the One-to-One association in the database

One-to-Many

This association is illustrated as follows: A Track is linked to a single Album and an Album can be linked to several tracks. As with the previous association, a single line is sufficient in the database (see Figure 28 and Figure 29).


```
Track.belongsTo(Album, { foreignKey: Track.album_id });
Album.tracks = Album.hasMany(Track, { foreignKey: Track.album_id });
```
```
export default class Track extends Model {
    static album_id = "tra_alb_id";
```
Figure 28. Instruction to make a One-to-Many association with Sequelize

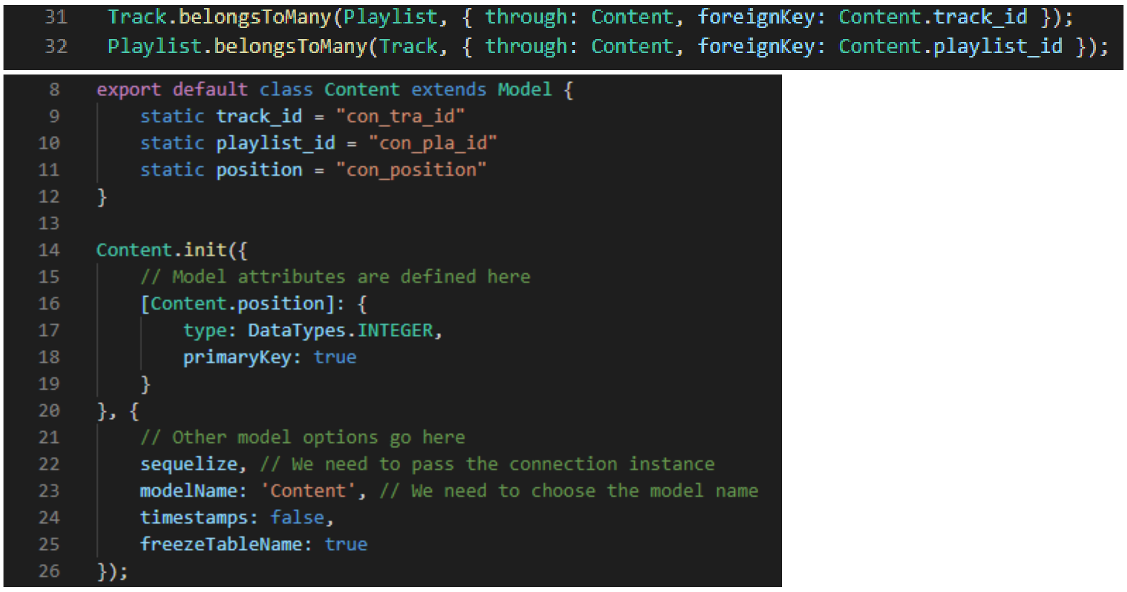

my_music tracks
tra_id : int(11)
tra_name : varchar(255)
tra_duration : int(11)
tra_number : int(11)
tra_no_disc : int(11)
tra_isrc : varchar(255)
tra_alb_id int(11)

my_music albums
alb_id : int(11)
alb_name : varchar(255)
alb_release_date : date
alb_nb_track : int(11)
alb_picture : varchar(255)
alb_type : varchar(255)

Figure 29. Result of the One-to-Many association in the database

Many-To-Many

This association means that a track can be associated with zero or more playlists and vice versa. In general, it is not necessary to create another model, but in my case I did it to be able to add an additional attribute to the association table (see Figure 30). This attribute allows me at the database level to have the order of the songs in the selected playlist (see Figure 31).



```
Track.belongsToMany(Playlist, { through: Content, foreignKey: Content.track_id });
Playlist.belongsToMany(Track, { through: Content, foreignKey: Content.playlist_id });
```

```
export default class Content extends Model {
    static track_id = "con_tra_id"
    static playlist_id = "con_pla_id"
    static position = "con_position"
}

Content.init({
    // Model attributes are defined here
    [Content.position]: {
        type: DataTypes.INTEGER,
        primaryKey: true
    }
}, {
    // Other model options go here
    sequelize, // We need to pass the connection instance
    modelName: 'Content', // We need to choose the model name
    timestamps: false,
    freezeTableName: true
});
```

Figure 30. Instruction to make a One-to-Many association with Sequelize

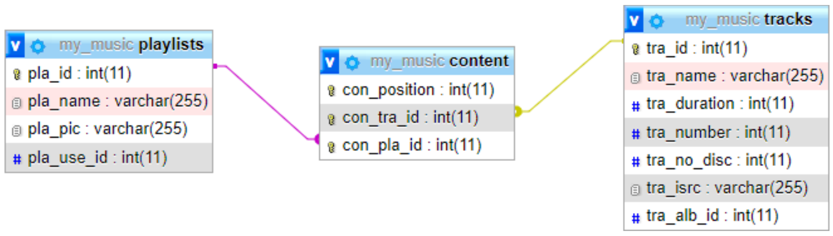
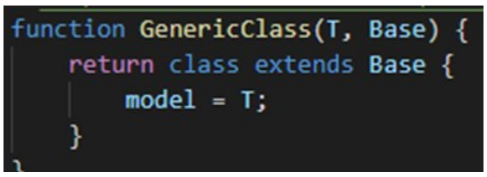


Figure 31. Result of the One-to-Many association in the database

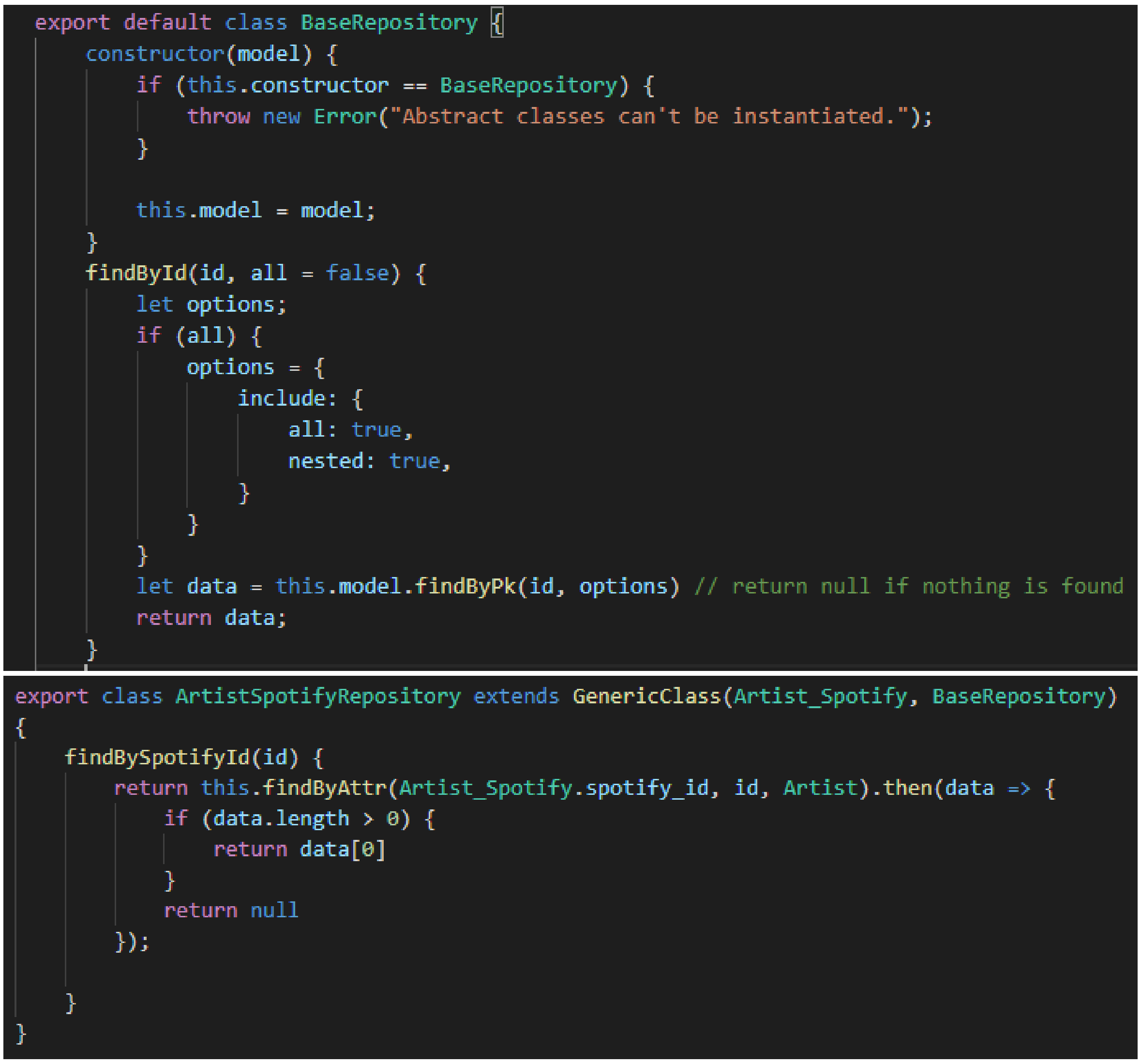
5.2.4 Repository and Generic Class

Each table has a repository allowing to centralize all the functions specific to each table. To gain simplicity, I created a Base class with all the methods common to all the tables, such as **findById**. As JavaScript is not a basic object-oriented language, making a generic class can be more complex. While searching, I found a solution where the subclass inherits from a function that will generate a class with a generic type (see Figure 32). An example of repository implementation is available on Figure 33.



```
function GenericClass(T, Base) {
    return class extends Base {
        model = T;
    }
```

Figure 32. The Generic Class mechanism (Stackoverflow 2018).



```
export default class BaseRepository {
    constructor(model) {
        if (this.constructor == BaseRepository) {
            throw new Error("Abstract classes can't be instantiated.");
        }

        this.model = model;
    }
    findById(id, all = false) {
        let options;
        if (all) {
            options = {
                include: {
                    all: true,
                    nested: true,
                }
            }
        }
        let data = this.model.findByPk(id, options) // return null if nothing is found
        return data;
    }
```

```
export class ArtistSpotifyRepository extends GenericClass(Artist_Spotify, BaseRepository)
{
    findBySpotifyId(id) {
        return this.findByAttr(Artist_Spotify.spotify_id, id, Artist).then(data => {
            if (data.length > 0) {
                return data[0]
            }
            return null
        });

    }
}
```

Figure 33. Example of Repository

5.3 Strategy Class

These classes allow to define all the functions allowing the access to a specific API to a platform. It serves as an interface between the application and the API in order to facilitate the integration of the latter in my project. The class PlatformStrategy allows to simulate the concept of abstract class in JavaScript. Indeed, it provides all the methods for each platform. Afterwards, it will be possible to make polymorphism if needed. I was inspired by this thread on Stackoverflow (Stackoverflow 2011). Each platform has in common some generic features such as the use of the search engine or the detailed access to a resource (Track, Album, Artists and Playlist).

5.3.1 Logs / Errors

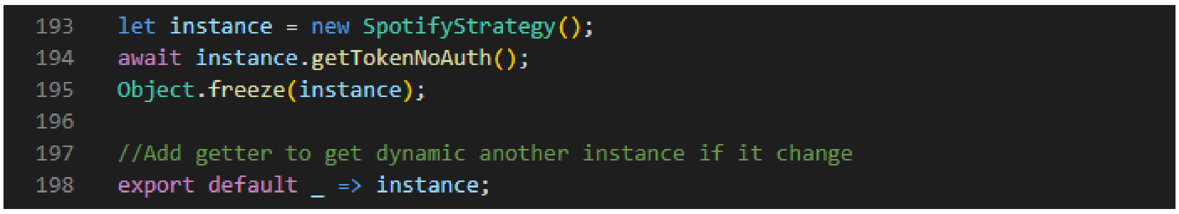
A logging system is set up so that query errors can be recorded in a log file. Errors are handled according to the error messages of the specific platform and are processed in a try catch to be able to catch the error and process it. It was important to be able to handle errors within the class and to differentiate this type of error from the errors generated by MyFetchError.

5.3.2 Regulation

A constraint I discovered during my work is the overload of requests on an API. Indeed, in order not to overload the servers, the platforms offering an official API (Spotify and Deezer) have set up a limitation of requests and return a specific message when I execute too many calls. That is why I set up a regulation system that puts a breakpoint during the execution of a request when the API is overloaded. It restarts the request once the service is available so as not to impact the client.

5.3.3 Singleton

Each class implements the singleton design pattern, but in a rather particular way, because JavaScript is not a basic object-oriented language and this concerns each concrete class, I made sure not to export the class in the other files, which makes the instance of the latter only possible within the file where it is created and to export only a function which will play the role of the singleton (see Figure 34).



```
let instance = new SpotifyStrategy();
await instance.getTokenNoAuth();
Object.freeze(instance);

//Add getter to get dynamic another instance if it change
export default _ => instance;
```

Figure 34. Example of the singleton concept in the SpotifyStrategy class

5.4 SpotifyStrategy Class

This class is the link between the Spotify API and the server. It implements the Spotify specific authentication process "Authorization code flow" to make requests on the platform.

5.4.1 Renew token

With a temporary authentication token, there will come a time when the token will be expired and will no longer allow requests to be made without restarting the authorization process. So, we had to set up a token renewal system without impacting the client when he makes a request.

5.5 DeezerStrategy Class

This class is the link between the Deezer API and the application. There is no authentication system or big constraint compared to other APIs.

5.6 YoutubeStrategy Class

The YoutubeStrategy class makes the link between the application and the unofficial YouTube music API through Python. Its access to the resource is particularly different from the others. Indeed, I used the native node package (child_process) to run a python script that will fetch the data and return it to me in JSON format. The python script takes as parameter a function name according to the requested resource (for example getArtist) and provides an object variable with the necessary parameters. In the python script, there is a switch that will execute the desired function and do the necessary logical processing to return the requested data.

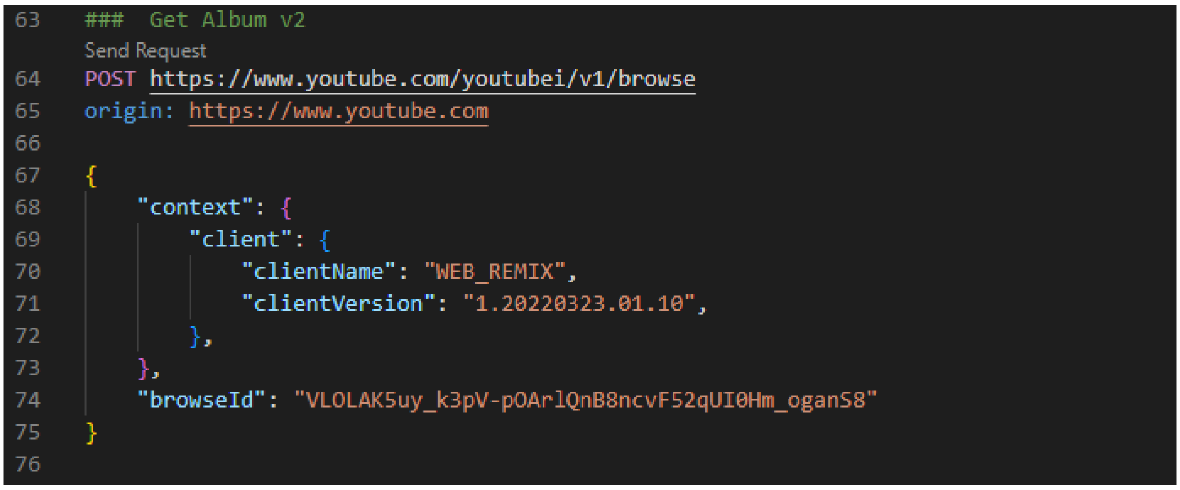
5.6.1 Manage Errors

An error handling system is made by this script allowing to distinguish between Generate errors when the resource does not exist (code: INTERNAL_404) and script errors of bad implementation when creating this script and for the future (code: GENERAL). The latter is then passed on to the Node.js server and is processed as appropriate.

5.6.2 Get album v2

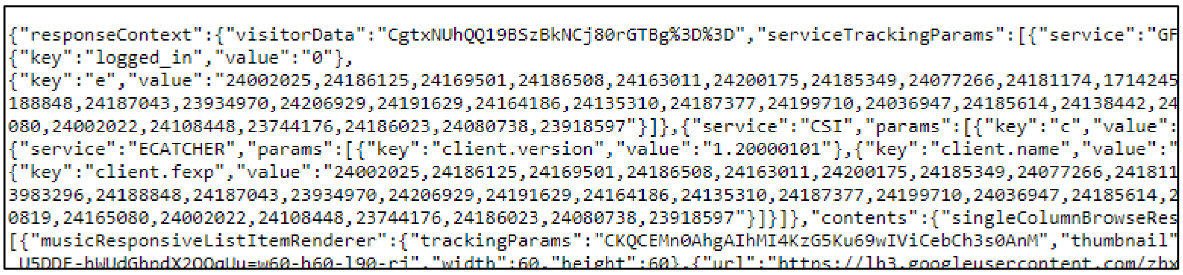
The YouTube API does not provide me with all the data I need. Indeed, one thing to know is that a track with a YouTube video version also has a music version. The difference between the two is that the video version is specific to YouTube unlike the music version which is linked to YouTube music. The latter is for example linked to an album unlike the video version. By default, when accessing an album, the sounds that have a video version are defined with it and therefore the audio version is not readable from the API.

I could find by chance that loading the album with its playlistid on the site youtube that it is possible to see the version of the album with only audios. So, I analysed how the network traffic of the site youtube.com and I managed to have access to this data and to recover the identifiers (see Figure 35). There was a work of analysis of the data beforehand in order to return an easily readable result (see Figure 36). During this process, I was able to meet new use cases where there were errors, for example a piece with a null identifier without any real explanation or an unmentioned artist that returned an error. I had to manage all these possibilities and others like a badly formatted identifier, or one that could not be found and I made several tests in order to determine a valid resource or not. And return an adequate result (see Figure 37) with the **getAlbumFromFetched** function.



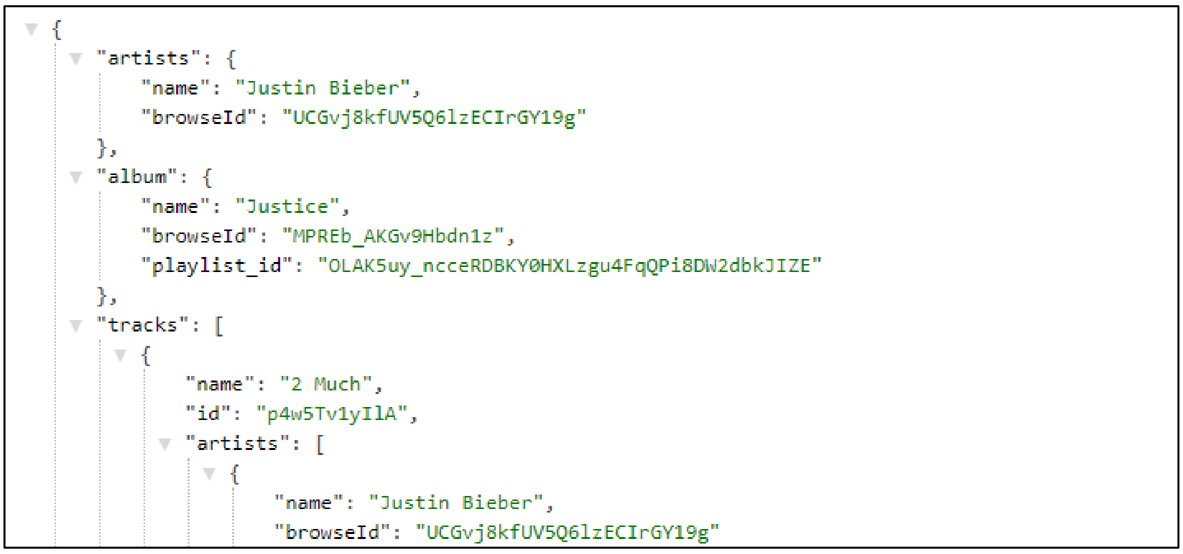
```
###  Get Album v2
Send Request
POST https://www.youtube.com/youtubei/v1/browse
origin: https://www.youtube.com

{
    "context": {
        "client": {
            "clientName": "WEB_REMIX",
            "clientVersion": "1.20220323.01.10",
        },
    },
    "browseId": "VLOLAK5uy_k3pV-pOArlQnB8ncvF52qUI0Hm_oganS8"
}

```

Figure 35. Example of request to find album with only all music version of track

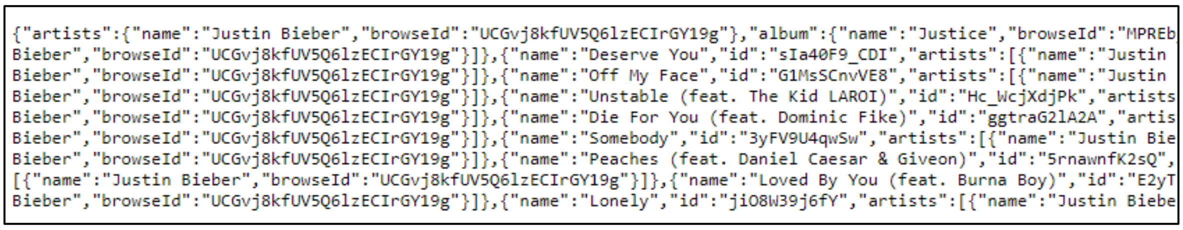


```
{"responseContext":{"visitorData":"CgtxNUhQQ19BSzBkNCj80rGTBg%3D%3D","serviceTrackingParams":[{"service":"GF
{"key":"logged_in","value":"0"},
{"key":"e","value":"24002025,24186125,24169501,24186508,24163011,24200175,24185349,24077266,24181174,1714245
188848,24187043,23934970,24206929,24191629,24164186,24135310,24187377,24199710,24036947,24185614,24138442,24
080,24002022,24108448,23744176,24186023,24080738,23918597"}]},{"service":"CSI","params":[{"key":"c","value":
{"service":"ECATCHER","params":[{"key":"client.version","value":"1.20000101"},{"key":"client.name","value":"
{"key":"client.fexp","value":"24002025,24186125,24169501,24186508,24163011,24200175,24185349,24077266,241811
3983296,24188848,24187043,23934970,24206929,24191629,24164186,24135310,24187377,24199710,24036947,24185614,2
0819,24165080,24002022,24108448,23744176,24186023,24080738,23918597"}]}]},"contents":{"singleColumnBrowseRes
[{"musicResponsiveListItemRenderer":{"trackingParams":"CKQCEMn0AhgAIhMI4KzG5Ku69wIViCebCh3s0AnM","thumbnail"
U5DDE-hWUdGhndX2OOqUu=w60-h60-l90-rj","width":60,"height":60},{"url":"https://lh3.googleusercontent.com/zhx
```

Figure 36. Result of unprocessed data (about 1500 lines)



```
{
    "artists": {
        "name": "Justin Bieber",
        "browseId": "UCGvj8kfUV5Q6lzECIrGY19g"
    },
    "album": {
        "name": "Justice",
        "browseId": "MPREb_AKGv9Hbdn1z",
        "playlist_id": "OLAK5uy_ncceRDBKY0HXLzgu4FqQPi8DW2dbkJIZE"
    },
    "tracks": [
        {
            "name": "2 Much",
            "id": "p4w5Tv1yIlA",
            "artists": [
                {
                    "name": "Justin Bieber",
                    "browseId": "UCGvj8kfUV5Q6lzECIrGY19g"
```

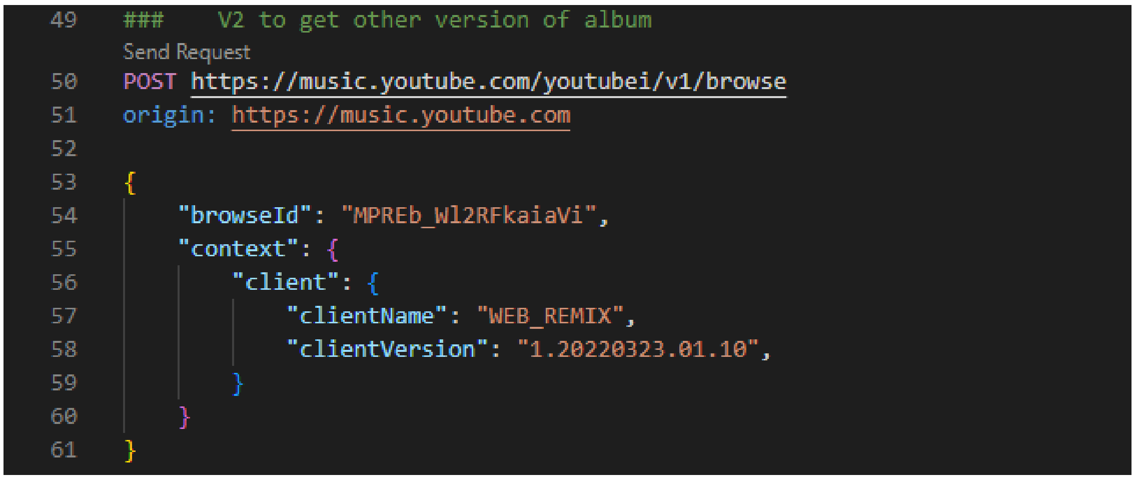


```
{"artists":{"name":"Justin Bieber","browseId":"UCGvj8kfUV5Q6lzECIrGY19g"},"album":{"name":"Justice","browseId":"MPREb
Bieber","browseId":"UCGvj8kfUV5Q6lzECIrGY19g"}]},{"name":"Deserve You","id":"sIa40F9_CDI","artists":[{"name":"Justin
Bieber","browseId":"UCGvj8kfUV5Q6lzECIrGY19g"}]},{"name":"Off My Face","id":"G1MsSCnvVE8","artists":[{"name":"Justin
Bieber","browseId":"UCGvj8kfUV5Q6lzECIrGY19g"}]},{"name":"Unstable (feat. The Kid LAROI)","id":"Hc_WcjXdjPk","artists
Bieber","browseId":"UCGvj8kfUV5Q6lzECIrGY19g"}]},{"name":"Die For You (feat. Dominic Fike)","id":"ggtraG21A2A","artis
Bieber","browseId":"UCGvj8kfUV5Q6lzECIrGY19g"}]},{"name":"Somebody","id":"3yFV9U4qwSw","artists":[{"name":"Justin Bie
Bieber","browseId":"UCGvj8kfUV5Q6lzECIrGY19g"}]},{"name":"Peaches (feat. Daniel Caesar & Giveon)","id":"5rnawnfK2sQ",
[{"name":"Justin Bieber","browseId":"UCGvj8kfUV5Q6lzECIrGY19g"}]},{"name":"Loved By You (feat. Burna Boy)","id":"E2yT
Bieber","browseId":"UCGvj8kfUV5Q6lzECIrGY19g"}]},{"name":"Lonely","id":"ji08W39j6fY","artists":[{"name":"Justin Biebe
```

Figure 37. Result of the same formatted data

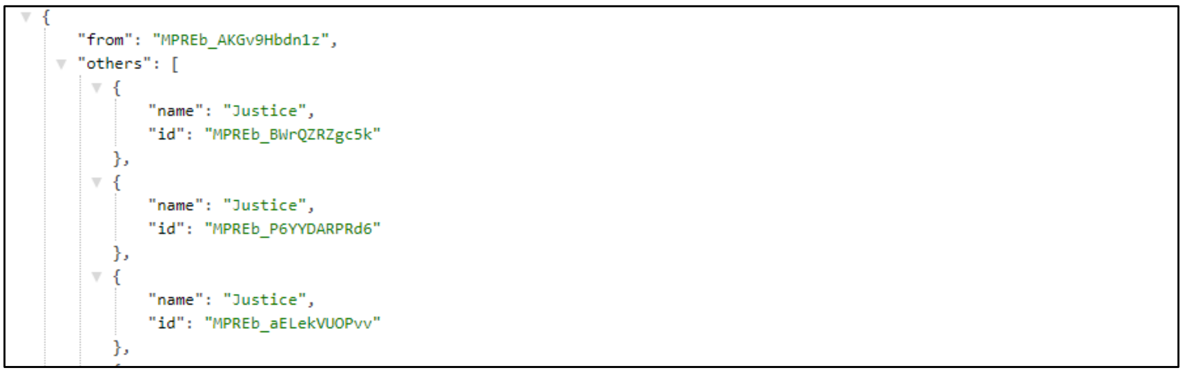
5.6.3 Other's version of album

Another problem encountered was that the search engine, unlike other platforms, does not mention duplicates. This is interesting for the end user but in the verification of data can be more difficult if the resource is not accessible from the search engine. For example, an album re-released with a strict link (see Appendix 15) is only referenced once. So, I tried to implement a function that will allow me to retrieve other versions of an album if it has other versions. I was finally able to use the same principle as Get album v2 (see Figure 38) to retrieve the data, analyse it and then format it in such a way as to have a concrete result (see Figure 39) that I implemented in the **getAlbumOtherVersion** method.



```
###    V2 to get other version of album
Send Request
POST https://music.youtube.com/youtubei/v1/browse
origin: https://music.youtube.com

{
    "browseId": "MPREb_Wl2RFkaiaVi",
    "context": {
        "client": {
            "clientName": "WEB_REMIX",
            "clientVersion": "1.20220323.01.10",
        }
    }
}
```

Figure 38. Example of a query to retrieve other versions of an album

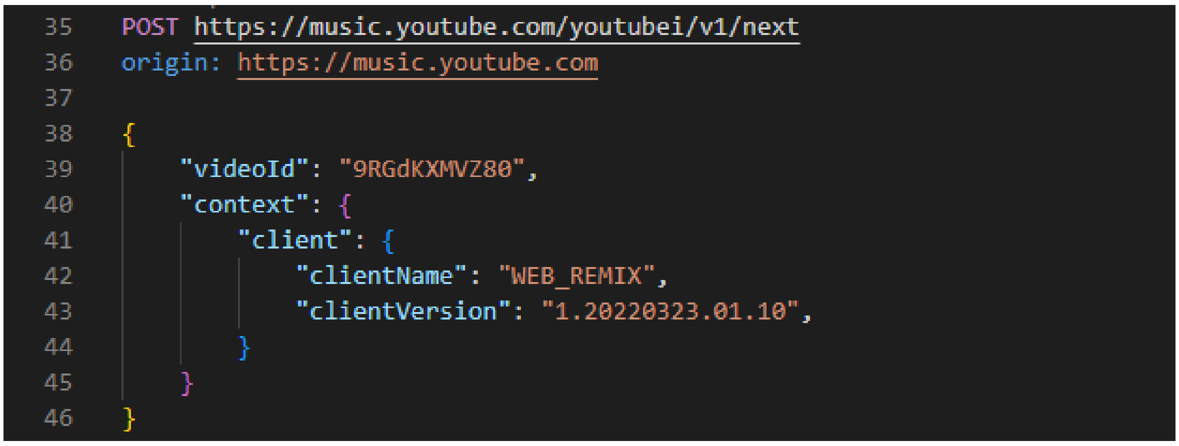


```
{
    "from": "MPREb_AKGv9Hbdn1z",
    "others": [
        {
            "name": "Justice",
            "id": "MPREb_BWrQZRZgc5k"
        },
        {
            "name": "Justice",
            "id": "MPREb_P6YYDARPRd6"
        },
        {
            "name": "Justice",
            "id": "MPREb_aELekVUOPvv"
        },
```

Figure 39. Result of the query to retrieve the other versions of an album

5.6.4 Get track v2

One of the last complications encountered is that when accessing the details of a track from the python library, the album ID is not mentioned as well as the other artists collaborating on the track, which can be quite problematic when linking data. So, I implemented a function that will analyse a track by simulating the addition of the track in playback and to be able to have access to these different data initially not accessible (see Figure 40). There was once again a work of analysis and formatting of the data in order to have readable data thanks to the method **getTrackFromContent**.

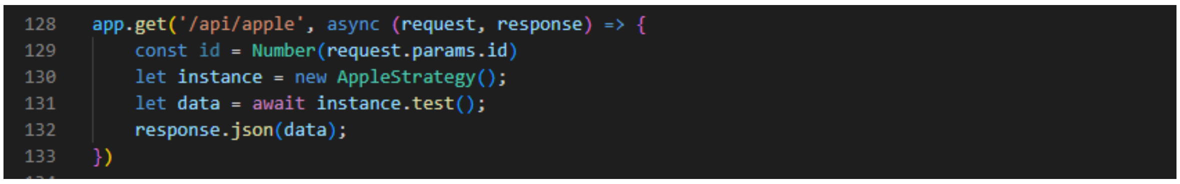


```
POST https://music.youtube.com/youtubei/v1/next
origin: https://music.youtube.com

{
    "videoId": "9RGdKXMVZ80",
    "context": {
        "client": {
            "clientName": "WEB_REMIX",
            "clientVersion": "1.20220323.01.10",
        }
    }
}
```

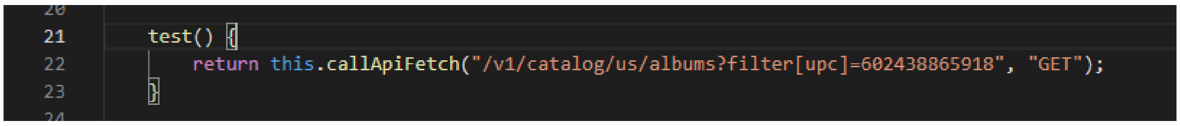
Figure 40. Example of a request to retrieve a track

5.7 AppleStrategy Class

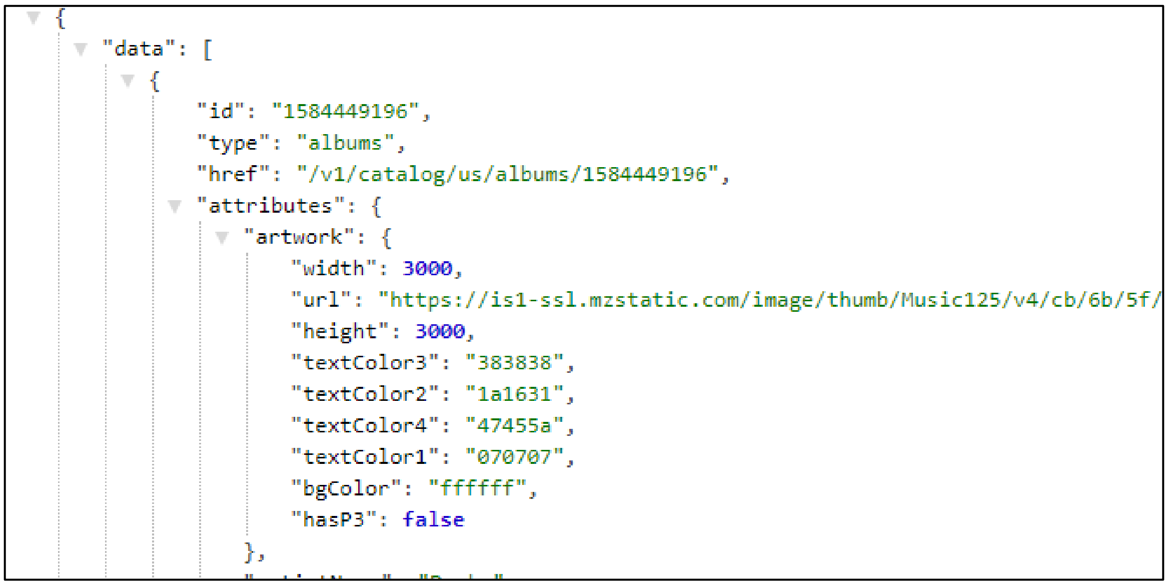
I have looked at how the https://music.apple.com website works by analysing network traffic. By copying the access token, which also does not expire, I can access the Apple API. Unfortunately for reasons of time, I only tested latter (see Figure 41).



```javascript
app.get('/api/apple', async (request, response) => {
    const id = Number(request.params.id)
    let instance = new AppleStrategy();
    let data = await instance.test();
    response.json(data);
})
```



```javascript
    test() {
        return this.callApiFetch("/v1/catalog/us/albums?filter[upc]=602438865918", "GET");
    }
```



```json
{
    "data": [
        {
            "id": "1584449196",
            "type": "albums",
            "href": "/v1/catalog/us/albums/1584449196",
            "attributes": {
                "artwork": {
                    "width": 3000,
                    "url": "https://is1-ssl.mzstatic.com/image/thumb/Music125/v4/cb/6b/5f/
                    "height": 3000,
                    "textColor3": "383838",
                    "textColor2": "1a1631",
                    "textColor4": "47455a",
                    "textColor1": "070707",
                    "bgColor": "ffffff",
                    "hasP3": false
                },
```

Figure 41. Result of the test query from Apple API

5.8 Thread

During the development of my project, I was confronted with several performance obstacles. Indeed, the function that allows to find an artist on other platforms can be more or less long in some cases and in each analysis function listed in the previous section, it's taken into account that the artist is inserted in the database in order to reduce the complexity of the link search. This means that without the artist in the database, it is not possible to perform the analyses correctly. The initial synchronous code I had done slowed down the whole process. It was stuck at certain points and this had no influence on the rest. So, I had to respond to this with asynchronous programming in order to be able to run processing in parallel.

5.8.1 Class ThreadParallel

Initially I had simply made all my code asynchronous but this generated execution errors on the different APIs by launching so many tasks at the same time and also errors by triggering searches for the same artist twice which would generate a duplicate in the database. So, we had to find a way to limit the number of functions that would be executed in parallel. With the Async library, I was able to meet this need by developing a Thread Parallel class that takes parameter the number of tasks that can be executed in parallel.

This class has a task queue and a dictionary that lists the tasks that are in the queue as well as an array containing the tasks that are currently running. It’s possible with this class to add or set an existing task to priority in the queue, which means that it will be executed as quickly as possible by moving to the head of the queue. It’s also possible to delete a task if it isn’t yet executed or to interrogate the class to find out if a task exists in the queue. The advantage of this class is that any asynchronous function can be added to the queue and that it is not mandatory for each task to execute the same action as was the case by default with the Async library.

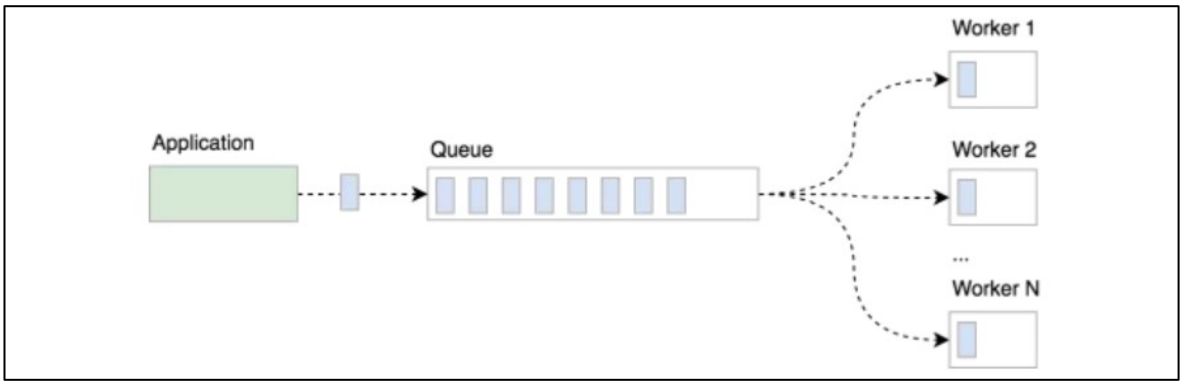


Figure 42. Illustration of the execution process (Antolovic 2018)

5.8.2 Deadlock

Thanks to this class, I was able to set up a parallel processing of the creation of the artists in the database. To solve this problem of a deadlock in the analysis processing, I implemented the **checkArtistInDatabase** function which checks for the existence of processing of an artist's creation and returns it’s identifier from the database only if the processing is complete. If processing is still in progress, it will pause the analysis that invoked the function with the await keyword until it returns a value. In case the artist's processing mentioned in the function is in the queue but not executed, it will prioritise it to speed up the process. If the process does not exist and the artist is not in the database, it will create a new task with the content of the artist's re-search on the different platforms.

5.9 Processing to analyse data

This is the most complex part of my job, it's the different processes of data analysis and link resolution across the different platforms. Due to time constraints the development of these processes is based on Spotify only.

The first thing I did, and for all the features, was to organically analyse the data recovery. This so-called bottom-up approach allows me to know what exists before I start programming and run into potential failure. I continued my analysis by drawing up an activity diagram to give me a graphical view of the code I was going to implement (see Appendices Appendix 24 and Appendix 25).

The problem is that coding this represents a lot of if-else checks, so a series of deeply nested if-else. So, I coded my analysis to reduce the depth of the ifs as much as possible (Lewis 2019).

This approach allows me to see visually how I will proceed initially. Note that each diagram does not look 100% like the final code as this was a first version and during development the process evolved.

Each check is recorded in a log file. This allowed me during development to more easily see which links were not found so that I could check for myself if it was an error in the implementation or because the specified resource did not exist.

5.9.1 Analyse Artists

The function **FindArtistInAllPlatform** is called when we try to find an artist in the database by his spotify_id and no result is returned. It will then proceed to retrieve the necessary data for the creation of the latter in the database, including links from all platforms in order to save time during the next checks (diagram in Appendices Appendix 24 and Appendix 25).

The first thing to validate is the identity of the artist. In fact, before you can insert a track or an album, you must have an artist in the database. The first thing to know is that on Spotify, it is possible that two artists have the same name, it is with its identifier that we can distinguish them quickly, but this cannot be considered as a comparison variable since the numbering is specific to each platform. One possible point of comparison is the artist's discography. That is, if we can identify a track or an album between the different platforms and the artist mentioned has the same name, we can deduce that it is the same artist.

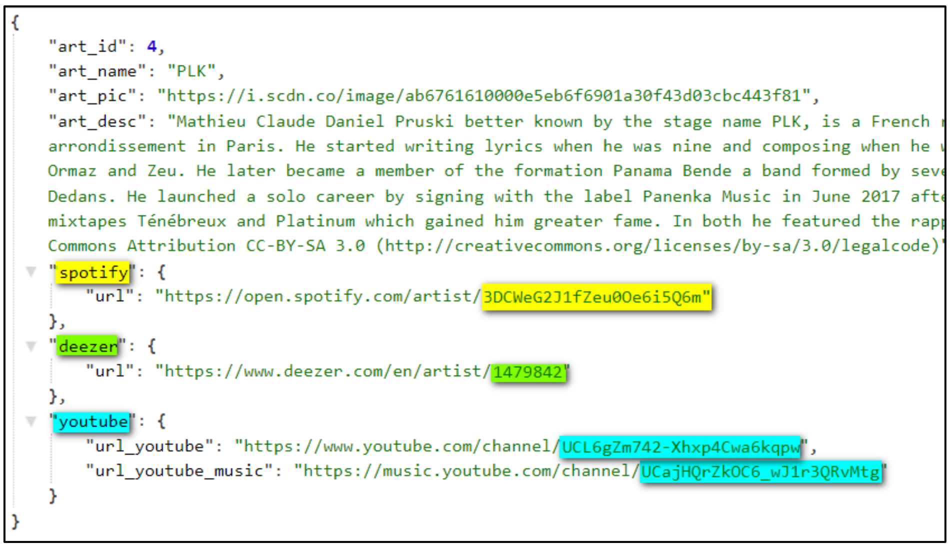
The technical explanation is therefore by starting from a track (obtained by the search engine for example) that we will find the identity of the artist. A track is linked to an album and we will try to find it on the other platforms initially using its UPC number.

If this does not work, we will check the ISRC code of the song to find the artist. Please note that we cannot base our search on the artist who authored the album because it may be a compilation album that is analysed and therefore the artist linked to the album is "Various Artist".

If this still doesn't work, we search the target platform's search engine looking for the artist's name followed by the album name filtered by album.

The last reliable way to check if the other processes were failures is to check by searching one by one each element of the discography (each album) on the target platform. So, we'll repeat the step where we search for the artists name with the albums name on the search engine and retrieve the UPC number of the target platform to see if it exists on the Spotify platform. Once this has been done, we can be sure that the artist is the same on both platforms. Once the data has been retrieved, we can proceed to create the artist within the database in the different tables with the **insertArtists** function.

On the Deezer platform, we just get the artist's ID. For YouTube, it’s different. It provides a description of the artist which can be retrieved as well as two identifiers, the one generated by YouTube music which is used for all resource references and the other which corresponds to the artist's official YouTube channel. It is in the latter that the artist can post video clips. The result of data can of an artist can be seen on Figure 43.



```
{
    "art_id": 4,
    "art_name": "PLK",
    "art_pic": "https://i.scdn.co/image/ab6761610000e5eb6f6901a30f43d03cbc443f81",
    "art_desc": "Mathieu Claude Daniel Pruski better known by the stage name PLK, is a French
    arrondissement in Paris. He started writing lyrics when he was nine and composing when he
    Ormaz and Zeu. He later became a member of the formation Panama Bende a band formed by sev
    Dedans. He launched a solo career by signing with the label Panenka Music in June 2017 aft
    mixtapes Ténébreux and Platinum which gained him greater fame. In both he featured the rap
    Commons Attribution CC-BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0/legalcode)
  ▼ "spotify": {
        "url": "https://open.spotify.com/artist/3DCWeG2J1fZeu0Oe6i5Q6m"
    },
  ▼ "deezer": {
        "url": "https://www.deezer.com/en/artist/1479842"
    },
  ▼ "youtube": {
        "url_youtube": "https://www.youtube.com/channel/UCL6gZm742-Xhxp4Cwa6kqpw",
        "url_youtube_music": "https://music.youtube.com/channel/UCajHQrZkOC6_wJ1r3QRvMtg"
    }
}
```

Figure 43. Example of final data of an artist in the database

5.9.2 Analyse Track

The function **analyseTrack** is called when trying to access a track in the database by its spotify_id and the result is empty. It will analyse the track's data from Spotify and perform several checks within the database to insert the missing data based on Spotify only (diagram in Appendices Appendix 26 and Appendix 27).

In technical term, the first thing to do is to check the track by the spotify_id of the album to see if the album already exists in the database. If the album is found, we will check by retrieving all the tracks of the album from the database and check by the title of the track if it exists in the database or not. This process is important if the album was added by another platform and the Spotify link was made.

If the track is found in the database, we check if it has a Spotify id, if so we generate an error in a log file otherwise we update the track by adding a link with the Spotify id. In both cases, we return the track from the database to complete the process. In case the song is not found, a song creation procedure is performed using the existing album in the database as a foreign key.

If the album was not found by its spotify_id, we will check by searching for it by its UPC number in the database. If a result was found, we will check if the album has a spotify_id. If so, we will generate an error in a log file. The rest of the checks are similar to the previous steps (starting from the verification of the track within the database) with the addition of linking the album in the database with the Spotify ID of the album linked to the track analysed. We will also check to see if it has a UPC link. In this case, a log is generated stating that an album now has two UPC numbers. Indeed, it is possible that a UPC number is different from one platform to another but that it is the same entity.

At this stage, if the album is not found, we will check by retrieving all the albums of the artist from the database and test one by one by checking the name, the release date, the number of tracks and the cover of the album. If we find a conclusive result, we will proceed with the same checks as the previous paragraph and add the link with the UPC number and the album found.

Finally, if nothing is found, we can certify that the album does not exist in the database, so we will create the album in the database and consequently create the song in the database.

5.9.3 Link Album from Spotify to other platform

The function **analyseAlbumForPlatformName** is called when we want to retrieve the different missing IDs of an album from the database based on its spotify_id. It’s used when the user wants to find a song or an album on other platforms and will allow to update the database (see Appendix 28).

First of all, with the album to be analysed coming from the database, we will check with its UPC number(s) if it's possible to find it on the target platform. If the first check is unsuccessful, we will proceed to search for the album by the name of the associated artist followed by the name of the album on the platform's search engine.

For each result, we will first check the artist's ID if it corresponds to the one registered in the database. Then we will check the name, the number of songs, the date and the cover. Once a result is found we insert the link in the database and link the UPC from the platform if it's available. In case the album cannot be found, we log the error in a file.

5.9.4 Link Track from Spotify to other platform

The function **FindOtherTrackRef** is called when one wishes to retrieve the identifiers of a song from the database on other platforms. It's used when the user wishes to access the details of a song or an album from the mobile application (see Appendix 29).

Technically, the first step is to identify the album on the target platform with the function **analyseAlbumForXX** described above. Then the second step is to retrieve all the songs from the platform and search by name. If nothing is found, a search by ISRC is carried out if the platform makes this available. If this doesn't work, we simply link them by their indexes and make a log in order to put a warning on the reliability of the piece.

5.10 Endpoints

In this last part, I will explain the different endpoints I have set up for the backend. First of all, the set of endpoints linked to the API. These are simple interfaces to each of the APIs with error handling in case a resource cannot be found. The second section will concern the endpoints that will trigger the analysis processes and thus fill the database. The last section concerns the endpoints for reading data from the database.

5.10.1 API access

Each platform has these basic endpoints that allow access to the different functionalities available on the APIs. These endpoints will only be used within the server.

GET album/:id - Detail album

This route allows to access the detailed information of an album with an identifier specific to the API used. It is also possible to retrieve an album by its UPC number by mentioning "**upc:**" as a prefix to the UPC number.

A UPC barcode (or Universal Product Code) is used to represent and track your music as an entire physical or digital product, in contrast to an ISRC code which represents individual tracks or sound recordings (Ditto 2015).

On the Spotify specific endpoint, there is an additional parameter "full" that allows you to retrieve or not all the tracks of an album. By default, without this parameter, only the first 50 tracks are displayed and in case the album has more tracks, it is possible to retrieve all of them with this parameter. For YouTube, the endpoint also has an additional parameter "playlist" allowing to retrieve an album via its identifier "playlist_id" instead of "browseId".

GET track/:id - Detail track

This route allows to access the detailed information of a track with an identifier specific to the API used. It is also possible to retrieve a track by its ISRC number by mentioning "**isrc:**" as a prefix to the number.

The ISRC code is a globally recognized standard numbering system that is used exclusively in the music industry to uniquely identify sound recordings (Mason 2017).

GET artist/:id - Detail artist

This route allows access to the detailed information of an artist with an identifier specific to the API used.

GET search/ - Search engine

The search engine allows you to search for an item through the selected API. It is possible to filter by resource type (track, album or artist).

5.10.2 Analysis processing (Result)

`GET /spotify/search - Searching for a song`

This access point is used to search using the Spotify API. It returns the search result to the user, triggering the analysis process for each of the tracks and artists mentioned in the result.

`GET /spotify/track/:id - Detailed display of a track`

The detailed access point of a track allows the user to trigger the cross-platform link search processing (see 5.9.4) of the selected track. If the song doesn't exist in the database, it will first trigger the track analysis function to insert it into the database.

`GET /spotify/album/:id - Detailed display of an album`

This endpoint triggers the processing of linking a Spotify album (see 5.9.3) on each other platform. Once this processing is done it will link each track across each platform found.

`GET /spotify/playlist/:id - Displaying a playlist from Spotify`

This endpoint allows you to see the detailed content of a playlist on Spotify. It triggers the artist analysis process and the detailed track display process (see 5.9.4) for each song mentioned in the playlist.

5.10.3 Database operation

These public endpoints allow to see every essential element of the database. It is used to retrieve the data and display it on the mobile application.

`GET /artist/:id`

This access point allows an artist to be retrieved by its own identifier from the database. It also retrieves the identifiers of other platforms where this artist was found.

`GET /album/:id`

Using its database identifier, the following endpoint allows you to retrieve an album with all the available links that have been made. It also retrieves the artist associated with the album and all the tracks inserted in the database.

`GET /track/:id`

This path allows you to retrieve a track by its ID from the database or by its Spotify ID. If it’s with the latter that one tries to access, it will return a loading message if the track analysis process is in progress.

`POST /playlists`

This endpoint allows you to create a playlist in the database for the authenticated user.

`GET /user/:id/playlists`

All the playlists of a user are retrieved with this endpoint. You have to provide the user's ID in the URL.

`GET /playlists/:id`

This endpoint provides access to the details of a playlist, allowing you to retrieve all the tracks with which it’s associated.

5.10.4 Web browser

It is in this part that I will explain the details of the web server interface. It allows you to share content via a URL with people who doesn’t use the application, using a web page.

`GET /web/:type/:id`

It is through this page that you can share content online. It is enough to pass in parameter the type of resource and its identifier to be able to generate a link of sharing. The possible resources to share are artists, albums and tracks. It is based on the mock-up Figure 5 and the visual is available in Appendix 30.

6 Empirical part Mobile App

The mobile application is the last part of my thesis. Its purpose is to be able to exploit the endpoints of the server set up beforehand. It will also allow the user to authenticate himself with the platform of his choice. This information will not be communicated to the server for confidentiality reasons. With this mobile application, it will be possible to share content across different platforms.

As for the previous chapter, the first part concerns the organisation of the files with notably the explanation of the navigation system within the application. For the technical part, I will explain the different components I created as well as the communication with the different platforms in order to access the user's music library. The whole application was elaborated with Redux for which I devoted a part to explain its implementation. I will finish this chapter with the result by detailing the functionalities of each page implemented.

6.1 Organisation

6.1.1 File trees

Assets

This folder contains different images for the aesthetic part of the application.

Components

The components folder contains all the components I have developed in order to be able to use them several times in the different pages.

Pages

It's in this folder that the different screens are created. Each screen is a folder with different possible pages inside, each managed by a navigation component.

Redux

In this folder are defined the different files for the correct functioning of redux in the application.

Scripts

The scripts folder contains all the classes used to operate the various APIs.

Helper.js

This utility file contains several functions and variables for use throughout the application.

6.1.2 Navigation

I was inspired by the authentication navigation flows of the react navigation documentation (React 2022). The principle is to define two navigations, one for the disconnected part and the other for the authenticated part. I use the principle of nestled navigator which is to include a navigation system inside another one. It's illustrated as follows in my application (see Figure 44).

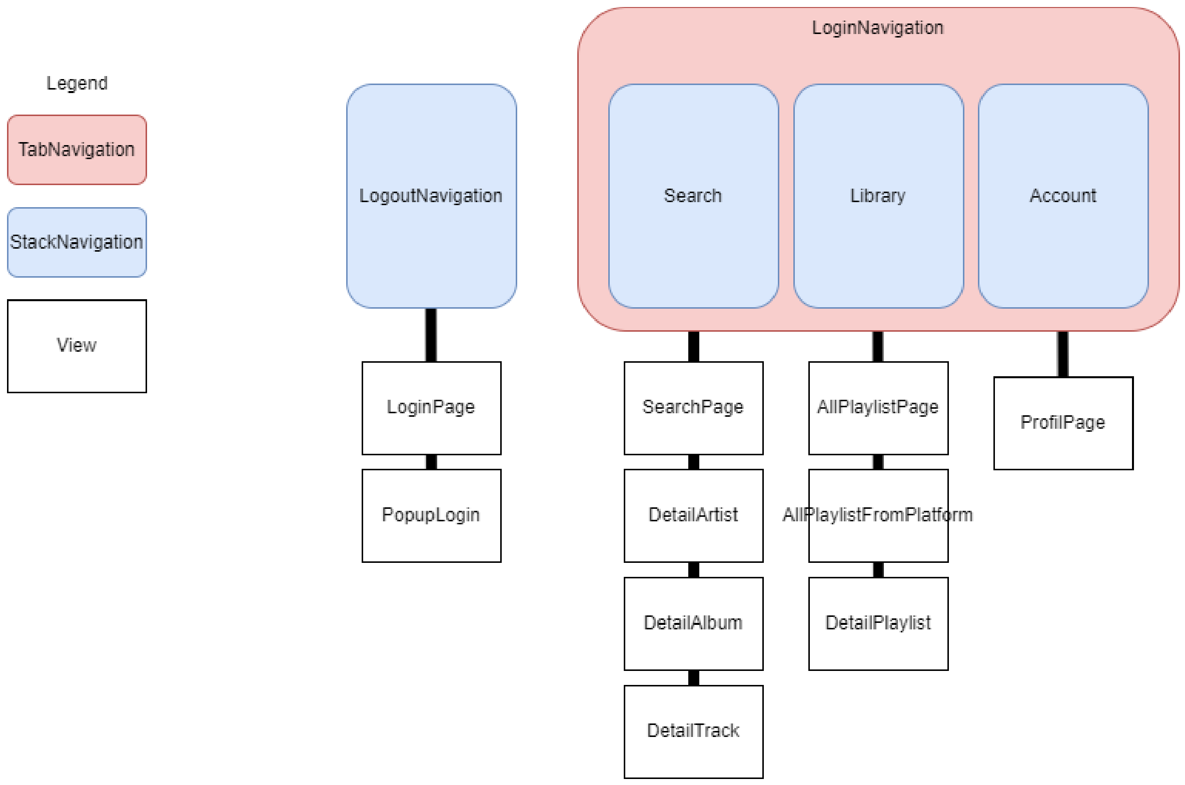


Figure 44. Navigation diagram of the mobile application

6.2 Components

6.2.1 AudioPlayable

This component allows you to incorporate audio content into the application. It allows to use a player to play an audio file (.mp3). The added value of this functionality being less than other parts of the application, I simplified it by using the audio tag coming from HTML because it corresponds to what I wanted and its implementation wasn't too complex compared to other possible libraries on React Native. It takes the path to an audio file as a parameter (see Figure 45).

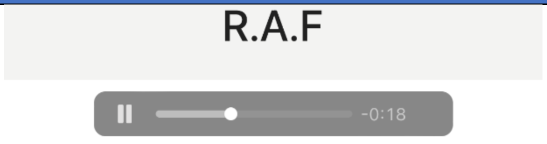
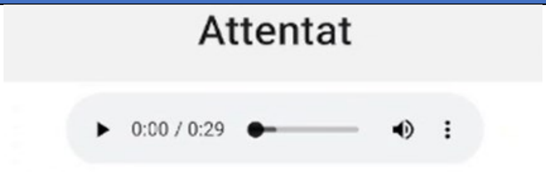
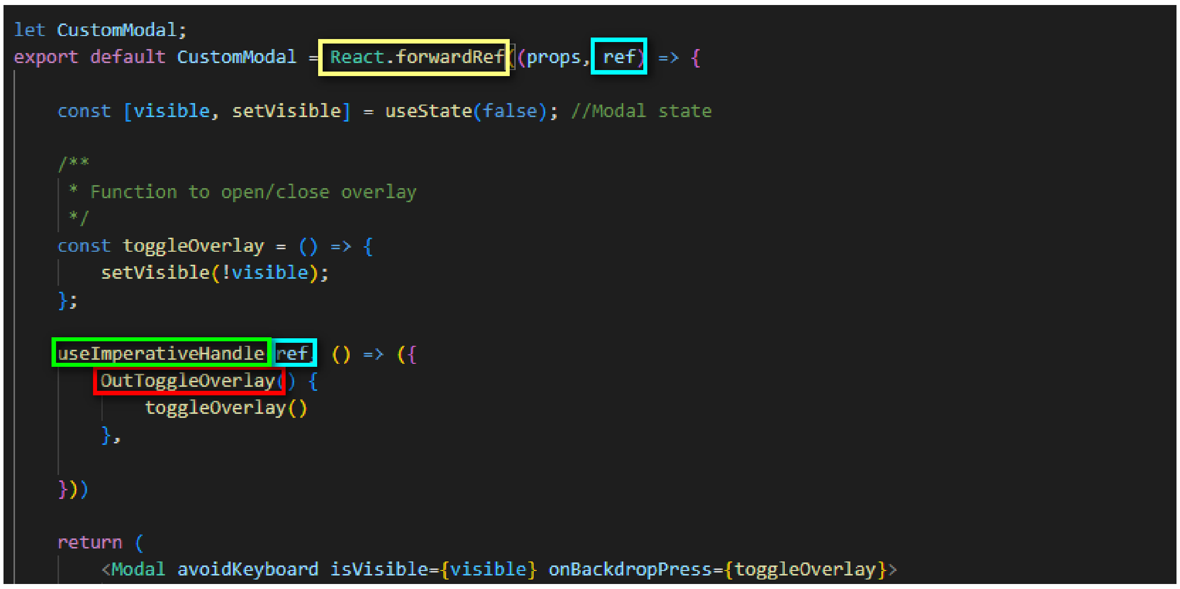
IOS	Android
	

Figure 45. Visual of the AudioPlayable component on the different platforms

6.2.2 CustomModal

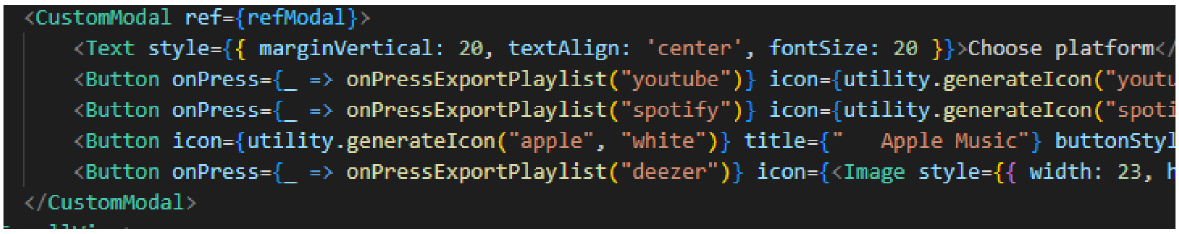
The CustomModal component allows you to create a structure for a fast implementation of a modal window. It also provides the function to trigger the display of the latter. The purpose of this component is to wrap the Modal component coming from react-native-modal with a predefined layout and to avoid making DRY on the management of the display of a modal window. I also used **useImperativeHandle** coming from React in order to make it possible to call a function outside the component using a **useRef**. For that it’s necessary that the exported component is wrapped in the **forwardRef** function (see Figure 46).



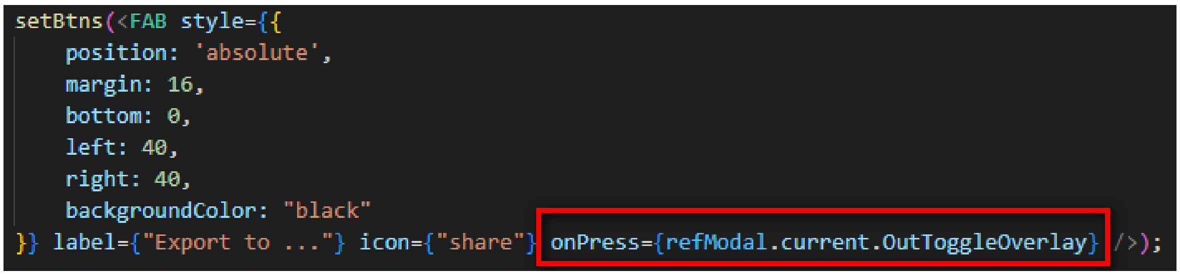
```
let CustomModal;
export default CustomModal = React.forwardRef((props, ref) => {

    const [visible, setVisible] = useState(false); //Modal state

    /**
     * Function to open/close overlay
     */
    const toggleOverlay = () => {
        setVisible(!visible);
    };

    useImperativeHandle(ref, () => ({
        OutToggleOverlay() {
            toggleOverlay()
        },

    }))

    return (
        <Modal avoidKeyboard isVisible={visible} onBackdropPress={toggleOverlay}>
```

Figure 46. Functioning of the CustomModal component

In this example (see Figure 47), the useRef variable is named refModal. The visual result is in the Appendix 31.



```
<CustomModal ref={refModal}>
    <Text style={{ marginVertical: 20, textAlign: 'center', fontSize: 20 }}>Choose platform</
    <Button onPress={_ => onPressExportPlaylist("youtube")} icon={utility.generateIcon("youtu
    <Button onPress={_ => onPressExportPlaylist("spotify")} icon={utility.generateIcon("spoti
    <Button icon={utility.generateIcon("apple", "white")} title={"   Apple Music"} buttonStyl
    <Button onPress={_ => onPressExportPlaylist("deezer")} icon={<Image style={{ width: 23, h
</CustomModal>
```



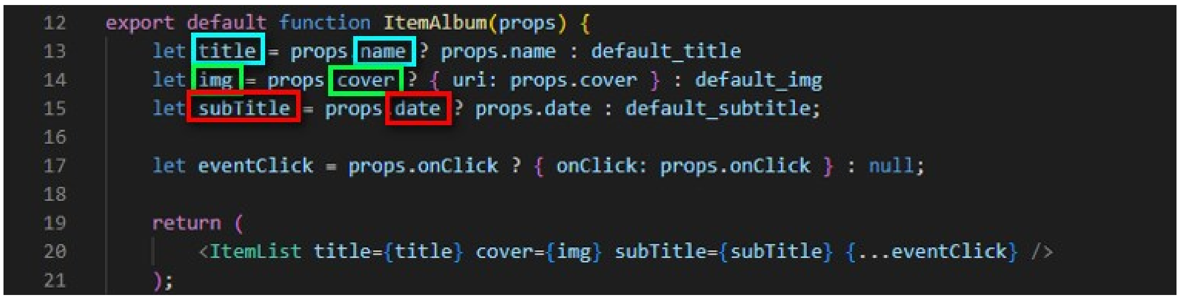
```
setBtns(<FAB style={{
    position: 'absolute',
    margin: 16,
    bottom: 0,
    left: 40,
    right: 40,
    backgroundColor: "black"
}} label={"Export to ..."} icon={"share"} onPress={refModal.current.OutToggleOverlay} />);
```

Figure 47. Extract of the use of the CustomModal component

6.2.3 ItemList

This component, based on ListItem from react-native-elements, allows to create a ListItem with the possibility to put an image, a title, a subtitle, a share link and to add an action when clicking on this element. The reason for this component was, as for the previous one, to avoid making DRY. Indeed, the implementation of a ListItem directly with the react-native-elements library made a lot of nesting of tags to display only one. Thanks to the ItemList component it's easier to read the application specific code. It also makes it possible to have a similar visual on each ItemList used in the application. This means that if in the future I want to change the layout, I can do it directly in this component and the operation will be done everywhere in the project.

In the application, in several places I use the ItemList component. During the development process I realised that the unique use of this component could be confusing if in the future, a developer would take over my project. That's why I created the ItemAlbum component which is a kind of sub-component of ItemList
with more album specific attribute names (see Figure 48). Each sub-component created has a sharing URL to display the resource sharing page from the server.



```
export default function ItemAlbum(props) {
    let title = props.name ? props.name : default_title
    let img = props.cover ? { uri: props.cover } : default_img
    let subTitle = props.date ? props.date : default_subtitle;

    let eventClick = props.onClick ? { onClick: props.onClick } : null;

    return (
        <ItemList title={title} cover={img} subTitle={subTitle} {...eventClick} />
    );
```

Figure 48. The code of the ItemAlbum component with specific attributes of an album

As with the previous component, the ItemSong component serves as an interface to display a song in the application. It has two different displays. One with the associated album cover used in the search page and the other without image used in the album detail page. The different visuals are available in Appendix 32. There are also the ItemArtist component to display a ListItem of an artist. It displays the artist's name and profile picture. The last one is the ItemPlaylist component. This component allows you to display a playlist in a ListItem. You can see the name of the playlist and its cover photo if it has one. There is a default picture if it doesn’t have one. The different visuals are available in Appendix 33.

6.3 Strategy Class

As for the server part of my project, this abstract class makes it possible to define the whole of the functions of access to each API so as to be able to make polymorphism within the application. We can find there in particular the functions of recovery of the data of the user such as its profile and the whole of its playlists.

In the concrete classes, each function returns data in the same way in order to avoid attribute problems within the application. The Singleton design pattern is also implemented for each of these classes.

6.4 Spotify Class

The Spotify class inherits from the Strategy class. It’s used to authenticate the user using Spotify's own authentication process, and to access the user's personal data once logged in. The authentication process is as follows: The user clicks on the login button, then a web page opens allowing the user to enter their personal information. Once connected the user is redirected to a website with the associated code and the code is returned to the instance to make a second call to obtain the access token.

6.5 Deezer Class

As with the previous class, the Deezer class allows you to use the different access points once authenticated. The authentication steps are the same as for the Spotify class.

6.6 YouTube Class

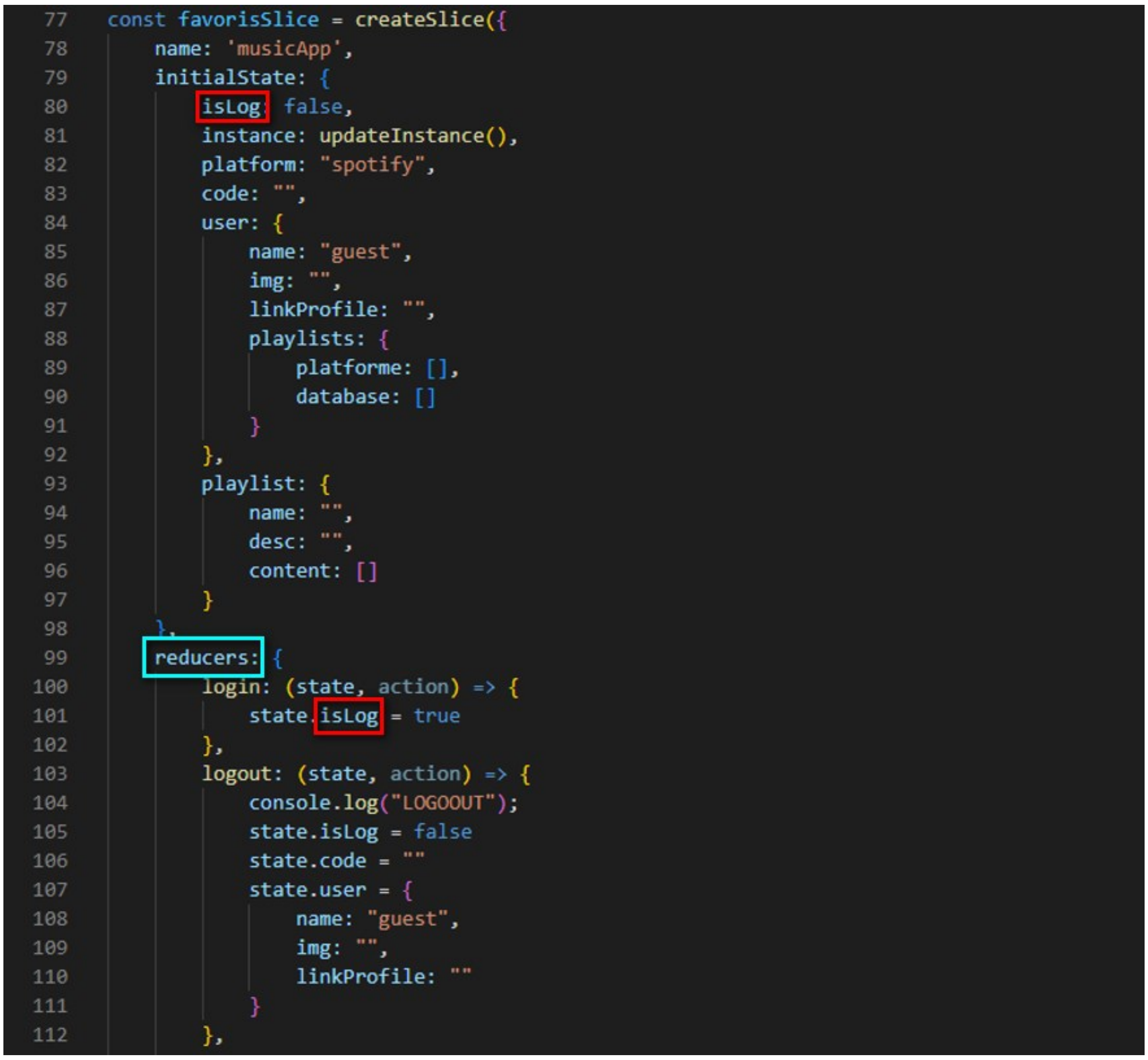
The YouTube class allows the user to authenticate with their Google account. Unfortunately authenticating within the application was quite complicated from the WebView, in the time available. Indeed, Google considers this to be an insecure browser and so authentication was done from the phone's browser. Once the steps are done, the user is redirected to a trusted site with the access token. To simplify the process, the user simply has to copy and paste the URL and enter it into the input field.

6.7 Redux

Redux allowed me to have a kind of centralization of the variables and to be able to centralize the different states of the application. I used the redux toolkit to simplify the installation of the store within the application.

6.7.1 Setup

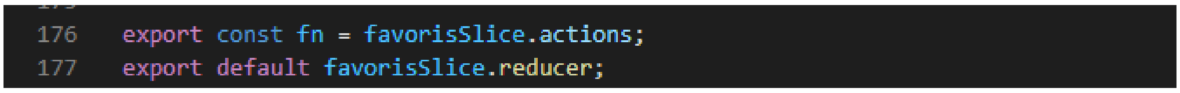
The first thing to do is to create a Reducer which will contain all the states using the **createSlice** function. This function also allows you to create the methods for updating the states in the **reducers** attribute. In the example (see Figure 49) shown below, the login method allows you to change the value of the isLog variable.



```
const favorisSlice = createSlice({
    name: 'musicApp',
    initialState: {
        isLog: false,
        instance: updateInstance(),
        platform: "spotify",
        code: "",
        user: {
            name: "guest",
            img: "",
            linkProfile: "",
            playlists: {
                platforme: [],
                database: []
            }
        },
        playlist: {
            name: "",
            desc: "",
            content: []
        }
    },
    reducers: {
        login: (state, action) => {
            state.isLog = true
        },
        logout: (state, action) => {
            console.log("LOGOOUT");
            state.isLog = false
            state.code = ""
            state.user = {
                name: "guest",
                img: "",
                linkProfile: ""
            }
        },
```

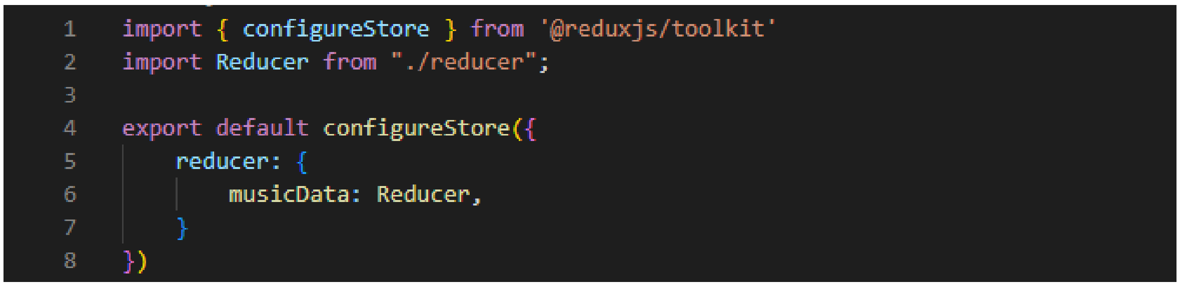
Figure 49. Creation of a Reducer with the toolkit

Once this was done, I exported the different functions so that I could use them in my application (see Figure 50). The next step is to set up a store. Using the **configureStore** function, I can use previously created Reducer and insert it into this method as follows (see Figure 51).



```
export const fn = favorisSlice.actions;
export default favorisSlice.reducer;
```

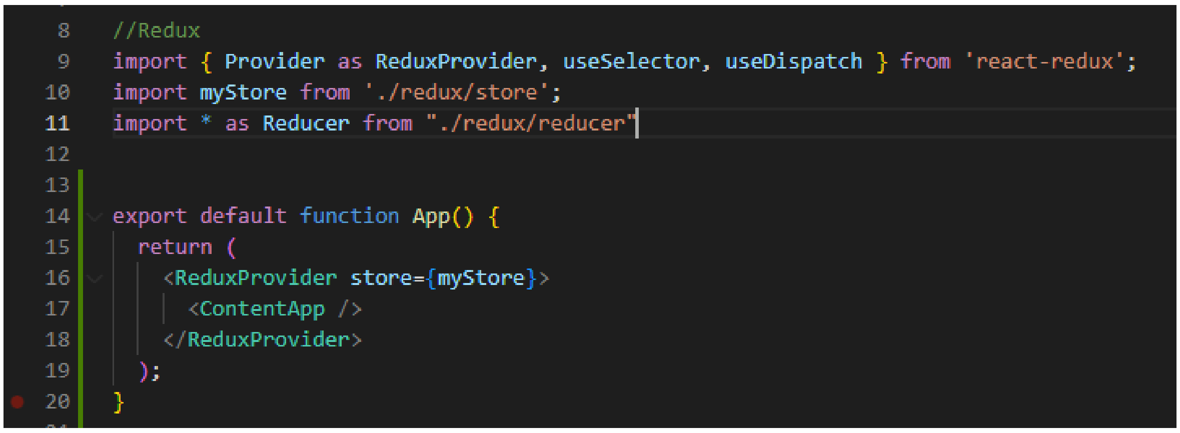
Figure 50. Export of Redux functions



```
import { configureStore } from '@reduxjs/toolkit'
import Reducer from "./reducer";

export default configureStore({
    reducer: {
        musicData: Reducer,
    }
})
```

Figure 51. Redux store configuration in the store.js file

The next step is to create a Provider at the root of the application to handle the different states of the Reducer. The store created in the store.js file must be mentioned as an attribute (see Figure 52).

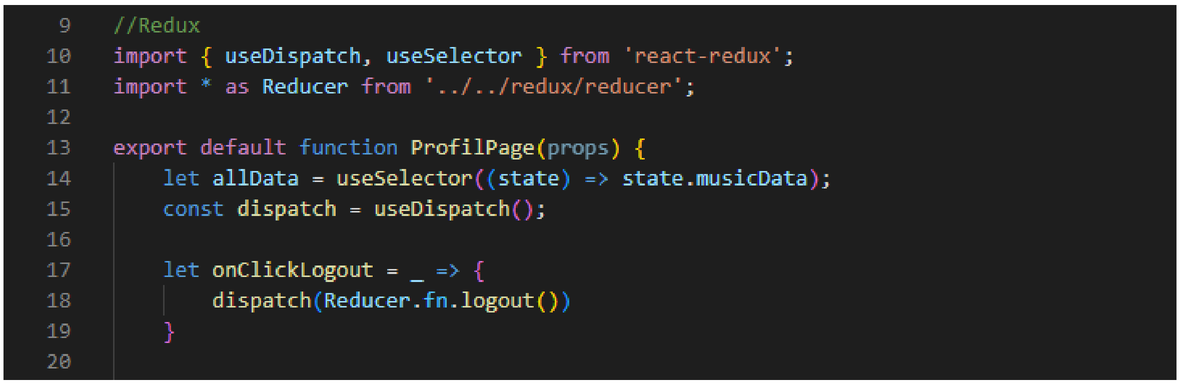


```
//Redux
import { Provider as ReduxProvider, useSelector, useDispatch } from 'react-redux';
import myStore from './redux/store';
import * as Reducer from "./redux/reducer"


export default function App() {
  return (
    <ReduxProvider store={myStore}>
      <ContentApp />
    </ReduxProvider>
  );
}
```

Figure 52. Setting up a Redux provider

6.7.2 Using Redux within the application

After the installation, it’s now possible to use Redux within the application. To do so, you just have to use the **useSelector** function of the react-redux library to get the states defined in the Reducer. To use the functions allowing the update of the states, it’s enough to **useDispatch** which allows to call a function of update. In this example (see Figure 53), thanks to the export of actions (see Figure 50), it’s possible to call the **logout** function. It’s possible to pass values in parameter which are accessible from the action.payload object in the reducers (see Appendix 34).

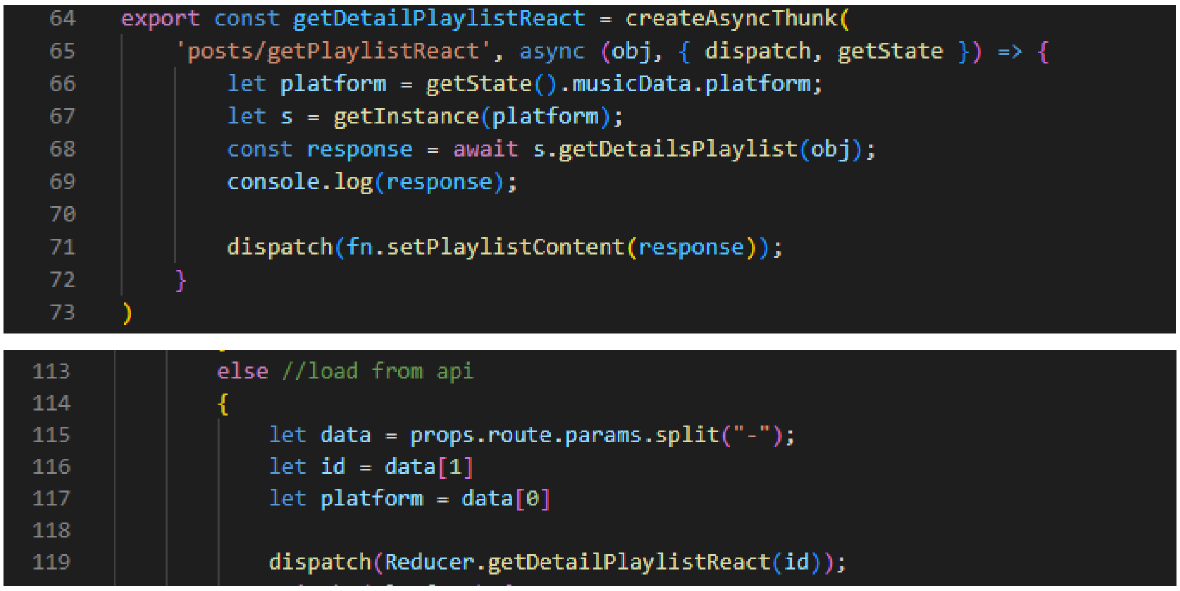


```
//Redux
import { useDispatch, useSelector } from 'react-redux';
import * as Reducer from '../../redux/reducer';

export default function ProfilPage(props) {
    let allData = useSelector((state) => state.musicData);
    const dispatch = useDispatch();

    let onClickLogout = _ => {
        dispatch(Reducer.fn.logout())
    }

```

Figure 53. Example of a Redux function call in the application

6.7.3 Asynchronous call with redux

The update functions within redux are synchronous by default but it's possible to make asynchronous calls using the **createAsyncThunk** function provided by the redux toolkit. It could be useful when you want to make fetch calls within a state update. Within a Thunk, the way to retrieve state values is slightly different than within a reducer, using the **getState** method. Updating the values is also different, it's not enough to do a simple affectation of value, it is necessary to pass by a reducer and a dispatch in order to be able to update the values. The redux toolkit documentation provides more information on the possibilities of using async thunk (ReduxToolkit 2022). It's also possible to retrieve values as parameters, as shown in this example (see
Figure 54) with the variable obj.



```
export const getDetailPlaylistReact = createAsyncThunk(
    'posts/getPlaylistReact', async (obj, { dispatch, getState }) => {
        let platform = getState().musicData.platform;
        let s = getInstance(platform);
        const response = await s.getDetailsPlaylist(obj);
        console.log(response);

        dispatch(fn.setPlaylistContent(response));
    }
)
```

```
        else //load from api
        {
            let data = props.route.params.split("-");
            let id = data[1]
            let platform = data[0]

            dispatch(Reducer.getDetailPlaylistReact(id));
```

Figure 54. Use of an Asynchronous Redux method

6.8 Interfaces

This part presents the different pages that have been implemented in the mobile application. Initially during the functional analysis and the elaboration of the screen mock-ups, I had thought of several possibilities of implementation (see Mock-up). Unfortunately, and due to lack of time, I only developed the essential parts of the different implemented pages and added other pages necessary to the functioning of the application.

LoginPage

The LoginPage is the home page of the application. This is the page that the user arrives at the first launch of the application when logged out. It allows the user to authenticate himself with the streaming service of his choice. He is then redirected to the platform specific page to authenticate himself. It’s based on the Figure 9. The visual is available in Appendix 35.

PopupLogin

The PopupLogin page is a WebView to which the user is redirected once they have chosen their login details. Once the user enters their login information and the platform validates this, they are redirected to the authenticated user navigation system. The visual is available in Appendix 35.

SearchPage

This page is the first page the user arrives on once logged in. It allows the user to search for a track and to trigger the analysis process on each track. It’s based on the Figure 12. The visual is available in Appendix 36.

DetailArtist, Album and Track

The DetailArtist page is the page that allows you to see the detailed information of an artist. You can see his profile picture as well as the different buttons allowing access to his profile on the different platforms. It’s based on the Figure 13. The visual is available in Appendix 37

The page DetailAlbum allows you to see the details of an album with all the tracks associated with it. As with the previous page, it is possible to open the album on the different platforms. It’s also based on the Figure 13 and the visual is available in Appendix 37.

The page DetailTrack allows you to have more concrete information about a song. In particular, you can find its video clip from the YouTube platform if it exists. The visual is available in Appendix 37.

AllPlaylistPage / AllPlaylistFromPlatform

The AllPlaylistPage page allows to see all the playlists of a user coming from the database. You can also import playlists from different platforms using the import button. It's inspired by the Figure 15. The AllPlaylistFromPlatform page is the page that displays all of the user's playlists from the platform with which they have authenticated. The visual is available in Appendix 38.

DetailPlaylist

This page allows you to see the details of a playlist within the database or a playlist specific to the user's platform. It's inspired by the Figure 14. The visual is available in Appendix 39.

ProfilPage

The next page shows the specific personal information of the logged-in user. It's with this page that he has the possibility to access his profile on his music platform. It's also through this page that he can disconnect from the application. It's inspired by the Figure 11. The visual is available in Appendix 40.

7 Discussion

To conclude, I could see some changes from the initial specification and the work I thought I was doing. I think I overestimated the workload. Indeed, once in the course of the realization of my thesis, I could see that I would not be able to entirely finish the whole of the functionalities which I had anticipated. The part that took too long compared to my estimate was all that concerns the YouTube API for reasons of using an unofficial library because there isn't one specific program for YouTube Music. I have tried to transcribe the work that I have done in this thesis as best.

7.1 Obstacles

For the first part of my work, I was confronted with several possibilities related to the data structure. I could simply stop at the basic data that works without taking into account exceptions such as a missing identifier, an unavailable reference, a null value etc... but this would later cause me a problem in the second part. The data testing steps could have been documented in detail. For example, by illustrating concretely the tested data that isn't available on all platforms such as an artist or a missing album and include it in the thesis.

For the server part, initially, I had not taken into account the fact that the API used would put limits on my use. Indeed, after a large number of requests, the API would block, which would not allow me to continue my process. This is why I set up a regulation system (see Regulation).

The performance problem was also the number of possible processes. By doing a one by one analysis, I avoided the possibility of collision of double insertion of the same data. This is why I implemented Threads (see Thread).

During the development of the features, I could see when making requests to the Spotify API that sometimes, for no particular reason, it would return empty results when the request was supposed to work. I've seen from various forums that this has happened to other developers (Github 2019) as well, and to fix this problem I've implemented a function that will restart the query if the result isn't consistent.

Sometimes the python script would return a null result when it should in any case return JSON content. It was difficult to identify this problem but I was finally able to find out what was wrong with the debugging of visual studio code and I set up a function like Spotify that will restart the request if it was supposed to return data.

I had also obstacle about python script optimisation. Initially, I had coded the logic in JavaScript and the problem was for example for the album search by upc. I had to make more than one call to the API and by doing this directly in the python script I was able to see a performance gain of up to two times faster (see Appendix 41).

When executing multiple calls from the controller endpoints, each call executes a python script. This means that if I make 100,000 requests at the same time, it will execute the script 100,000 times, which isn't necessarily optimal. A first solution was to set up a python server, which would lighten the workload, but it also doesn't support large quantities of requests. Having discovered this problem rather late in my work, I wasn't able to fix this problem but I tried to reduce the workload as much as possible when running the data analysis processes.

There are also obstacle during the development of the mobile application. The first one was the login with google. As with the other two platforms, I wanted to set up an authentication within the application with a WebView. Unfortunately, Google considered the latter as an insecure browser. A solution was possible but it was necessary to export the project in Native project allowing me to integrate the SDK IOS and Android of Google and to create a module for each of the two operating systems. This required a lot of work and with the time I had left, I put it aside.

The second obstacle was the possibility to debug the application. I was able to see quite late that it was possible to debug React Native. I didn't encounter too many problems that required debugging but it would have been interesting for me to set this up so I could do breakpoints or see the state of all the variables on the computer.

In addition to breaking my computer at the beginning of my work, the computer provided by the school was having problems with the internet connection. Sometimes the connection would skip a certain amount of time (3 seconds) and this would generate a fetch request error saying that there is no internet connection. I used several adapters to put an RJ-45 cable on the computer and it had the same effect as with Wi-Fi.

7.2 Personal learning

This work allowed me to learn several new notions and to confirm my bases in certain domain. I was able to do concurrent programming with threading that I implemented. This Thread class that I developed, was made so that it could be reused later in another project.

I was also able to improve my knowledge in terms of analysis, indeed having spent several days studying different data structures, allowed me to identify the useful data to achieve the goal of my project and also to create a scalable database allowing me to integrate later on other data.

I was able to discover new technologies during this work. This is my first project with Node.js and I was able to discover several new libraries and to solidify my bases with those I already knew. In React Native, I also learned how to create more advanced components with the possibility of integrating functions accessible from outside the Class.

7.3 Opportunities for improvement

During the last days of my work, I could see that there were several possibilities for improvement to make this project more concrete so that it could be put online in the future.

One of the improvements would be to apply the principle of the **checkArtistInDatabase** function to album and track resources. This would make the application even faster and allow more data to be manipulated in parallel.

Another improvement is about the python script. It is very efficient if it's an operation that happens with few times. In the context of this project, it’s probably the YouTube API that I use the most. On the one hand because of its lack of reliability compared to other platforms and on the other hand because I would have to make several calls to retrieve the different resources (album other version, album v2) given its implementation. The ideal would be to run the script on a python server and allow it to be multithreaded.

There are some features that have not been implemented in this work but that have been stated during the functional analysis such as the use of the application as a guest, the creation of new playlists or the persistence of data. Many possibilities of improvement are possible and even to integrate all these functionalities in another application which will be larger is possible.

The Design and ergonomics are very important for a mobile application and the website. It’s what makes the user want to stay on the application. Fast, intuitive and very simple to understand, this is what will make it pleasant to use.

In the long term, it would also be interesting to be able to integrate other streaming services into this project. I am thinking in particular of Apple Music, which is very popular.

Also, an improvement about maintenance of the project with test plan. As mentioned above, having a written and maintained test plan to see the different possible use cases is essential in development. It allows you to have a written record of possible uses so that you can redo them later and it greatly improves the level of maintenance of the application.

7.4 Personal interest

After a little more than two months of work, I am happy with the result I was able to provide. Whether it was the analysis, the server, the mobile application or the writing of my thesis, I had fun doing this work and it is because I took a subject that interests me that I was able to keep motivated during this period. It was still difficult to see the end of the tunnel, I had trouble seeing the final result that I was going to get as I worked with all these performance problems but once the database was filled, I had an easier time seeing the end result. Overall, I was able to apply everything I learned through my studies.

References

Antolovic 2018. Using Background Processing to Speed Up Page Load Times. URL: https://www.sitepoint.com/using-background-processing-to-speed-up-page-load-times/. Accessed: 20 May 2022

Async 2022. Home page. URL: http://caolan.github.io/async/v3/. Accessed: 15 April 2022

Chapter247 2020. Node.js vs Springboot Java -Which one to choose and when?. URL: https://www.chapter247.com/blog/node-js-vs-springboot-java-which-one-to-choose-and-when/. Accessed: 29 April 2022

cors 2019. cors - npmjs.com. URL: https://www.npmjs.com/package/cors. Accessed: 29 April 2022

Deezer 2022a. Ressource : Album. URL: https://developers.deezer.com/api/album. Accessed: 22 March 2022

Deezer 2022b. Ressource Track. URL: https://developers.deezer.com/api/track. Accessed: 22 March 2022

Deezer 2022c. Ressource Artist. URL: https://developers.deezer.com/api/artist. Accessed: 22 March 2022

Deezer 2022d. Deezer Developers. URL: https://developers.deezer.com/api/oauth. Accessed: 22 March 2022

Ditto 2015. Do I Need A Barcode To Sell My Music?. URL: https://dittomusic.com/en/blog/do-i-need-a-barcode-to-sell-my-music/. Accessed: 30 April 2022

Express 2017. Express. URL: https://expressjs.com. Accessed: 29 April 2022

Github 2019. Web API randomly returns 404 for most endpoints. URL: https://github.com/spotify/web-api/issues/1160. Accessed: 10 May 2022

Github 2022a. react-native-modal. URL: https://github.com/react-native-modal/react-native-modal. Accessed: 17 May 2022

Github 2022b. React Native WebView. URL: https://github.com/react-native-webview/react-native-webview. Accessed: 17 May 2022

Google 2022. Using OAuth 2.0 for JavaScript Web Applications. URL: https://developers.google.com/youtube/v3/guides/auth/client-side-web-apps. Accessed: 6 May 2022

Interviewquestionjava 2014. FRAMEWORK: ORM – Object Relational Mapping. URL: http://interviewquestionjava.blogspot.com/2014/01/framework-orm-object-relational-mapping.html. Accessed: 29 April 2022

Janssen 2022. Inheritance Strategies with JPA and Hibernate – The Complete Guide. URL: https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/. Accessed: 30 April 2022

Lewis 2019. How to write clean deeply nested if..else statements. URL: https://medium.com/@jnlewis/how-to-write-clean-deeply-nested-if-else-statements-579d29043f9a. Accessed: 30 April 2022

LogRocket 2021. React Native navigation: React Navigation examples and tutorial. URL: https://blog.logrocket.com/navigating-react-native-apps-using-react-navigation/. Accessed: 17 May 2022

Mason 2017. ISRC and UPC codes. URL: https://www.repostnetwork.com/resources/isrc-and-upc-codes/. Accessed: 30 April 2022

Mozilla 2022. Express Tutorial Part 4: Routes and controllers. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes. Accessed: 17 April 2022

Nodemon 2022. Nodemon. URL: https://www.npmjs.com/package/nodemon. Accessed: 29 April 2022

Paper 2022. Making your React Native apps look and feel native. URL: https://reactnativepaper.com/. Accessed: 17 May 2022

React 2022. Authentication flows. URL: https://reactnavigation.org/docs/auth-flow/. Accessed: 3 May 2022

ReduxToolkit 2022. createAsyncThunk. URL: https://redux-toolkit.js.org/api/createAsyncThunk. Accessed: 3 May 2022

Sequelize 2022. Home page. URL: https://sequelize.org. Accessed: 29 April 29

sigma67 2020. ytmusicapi. URL: https://ytmusicapi.readthedocs.io/en/latest/reference.html. Accessed: 24 March 2022

Spotify 2022a. Get album. URL: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-an-album. Accessed: 18 March 2022

Spotify 2022b. Get track. URL: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-track. Accessed: 18 March 2022

Spotify 2022c. Get artist. URL: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-an-artist. Accessed: 18 March 2022

Spotify 2022d. Spotify for developers. URL: https://developer.spotify.com/documentation/general/guides/authorization/code-flow/. Accessed: 18 March 2022

Stackoverflow 2011. How do I create an abstract base class in JavaScript?. URL: https://stackoverflow.com/questions/597769/how-do-i-create-an-abstract-base-class-in-javascript. Accessed: 20 May 2022

Stackoverflow 2014. Call a python from node js. URL: https://stackoverflow.com/questions/23450534/how-to-call-a-python-function-from-node-js. Accessed: 24 March 2022

Stackoverflow 2018. Creating generic classes and functions in javascript ES06. URL: https://stackoverflow.com/questions/47278140/creating-generic-classes-and-functions-in-javascript-es06. Accessed: 20 May 2022

TechTarget 2005. concurrent processing. URL: https://www.techtarget.com/searchoracle/definition/concurrent-processing. Accessed: 29 April 2022

Webb 2020. Codedrome.com. URL: https://www.codedrome.com/comparing-images-node-jimp/. Accessed: 21 April 2022

Wikipedia 2022a. Spotify. URL: https://en.wikipedia.org/wiki/Spotify. Accessed: 16 March 2022

Wikipedia 2022b. Deezer. URL: https://en.wikipedia.org/wiki/Deezer. Accessed: 22 March 2022

Wikipédia 2022c. YouTube Music. URL: https://en.wikipedia.org/wiki/YouTube_Music. Accessed: 24 March 2022

Wikipedia 2022d. Node.js. URL: https://en.wikipedia.org/wiki/Node.js. Accessed: 29 April 2022

Wikipedia 2022e. Redux (JavaScript library). URL: https://en.wikipedia.org/wiki/Redux_(JavaScript_library). Accessed: 15 May 2022

Wikipedia 2022f. Mapping objet-relationnel. URL: https://fr.wikipedia.org/wiki/Mapping_objet-relationnel. Accessed: 29 April 2022

Appendices

Appendix 1. Essential's endpoints used from Spotify's API

Method	Endpoint	Param	Desc	Auth
Pull of data				
GET	/v1/albums/**{id}**	**id**: The Spotify ID of the album.	Get details of an al-bum	No
GET	/v1/search	Query param **q**: search query **type**: A comma-separated list of item types to search across. Allowed values: album, artist, playlist, track, show, episode **limit**: number of results.	Search an element with filter	No
GET	/v1/tracks/**{id}**	**id**: The Spotify ID of the track. Query param **market**: An ISO 3166-1 alpha-2 country code	Get details of a track	No
GET	/v1/tracks	Query param **ids:** A comma-separated list of the Spotify IDs. **market**: An ISO 3166-1 alpha-2 country code	Get several tracks by List of Spotify ID.	No
GET	/v1/artists/**{id}**	**id**: The Spotify ID of the artist	Get a single artist identified by their unique Spotify ID.	No
GET	/v1/albums	Query param **ids:** A comma-separated list of the Spotify IDs. **market**: An ISO 3166-1 alpha-2 country code	Get several albums by List of Spotify ID.	No
GET	/v1/al-bums/**{id}**/tracks	**id**: The Spotify ID of the album Query param **offset**: The index of the first item to return **limit**: The maximum number of items to return	Get Spotify album's tracks.	No
GET	/v1/artists/**{id}**/al-bums	**id**: The Spotify ID of the artist Query param **offset**: The index of the first item to return **limit**: The maximum number of items to return	Get Spotify catalog information about an artist's albums.	No
Playlist management				
GET	/v1/me/playlists	None	Get playlist of user connected	Yes
GET	/v1/me	None	Get detail infor-mation of the cur-rent user	Yes
POST	/v1/us-ers/**{user_id}**/playlis ts	**user_id**: id of the user connected Request body **name**: the name of the playlist **description**: an optional description **public**: Boolean to define visibility **collaborative**: Boolean to define if the playlist is collaborative	Create a playlist	Yes
POST	/v1/playlists/**{playlist _id}**/tracks	**playlist_id**: The Spotify ID of the playlist Query param **position**: The position to insert the items **uris**: A comma-separated list of Spotify URIs to add	Add items to playlist	Yes
GET	/v1/playlists/**{playlist _id}**	**playlist_id**: The Spotify ID of the playlist	Get playlist	Yes

Appendix 2. Essential's endpoints used from Deezer's API

Method	Endpoint	Param	Desc	Auth
Pull of data				
GET	/search/album	Query param **q**: search query	Search albums	No
GET	/search/artist	Query param **q**: search query	Search artists	No
GET	/search/track	Query param **q**: search query	Search tracks	No
GET	/track/**{id}**	**id**: the id of the track. It can be a ISRC code with the acronym on the begin: isrc:id	Get details of a track	No
GET	/album/**{id}**	**id**: the id of the album. It can be a UPC code with the acronym on the begin: upc:id	Get details of album	No
GET	/artist/**{id}**/al-bums	**id**: the id of the artist	The list of the album of an artist	No
Playlist management				
GET	/user/me/playlist s		Get playlist of user connected	Yes
GET	/user/me		Get detail of the user connected	Yes
GET	/user/me/playlist s	Query param **request_method**: the HTTP verb (in this case it' POST) **title**: the name of the playlist	Create a playlist	Yes
GET	/playlist/**{playlis t_id}**	**playlist_id**: the id of the playlist	Get playlist	Yes
GET	/playlist/**{playlis t_id}**/tracks	**playlist_id**: the id of the playlist Query param **request_method**: the HTTP verb (in this case it' POST) **songs**: A comma separated list of track ids	Add items to playlist	Yes

Appendix 3. Essentials functions of the YouTube Python Library

Method	Param	Description	Auth
Pull of data			
get_album(**id**)	**id**: the browseId of the album. It's start by: MPREb_	Get details of album	No
get_song(**id**)	**id**: the videoId of the track or the video.	Get details of a song or a video	No
search(**q**,**filter**)	**q**: search query **filter**: the type of resource in the re-sult. It's possible to use only one filter	Search an element with filter	No
get_artist(**id**)	**id:** the channelId	Get the YouTube channel of the artist	No
get_playlist**(id)**	**id:** the playlist id	Get a playlist from YouTube	No
get_album_browse_id**(id)**	**id :** audioPlaylistId of an album	Get the browseId of an al-bum from a audioPlaylistId	No

Appendix 4. Essentials Endpoints use from YouTube's API

Playlist management				
Method	**Endpoint**	**Param**	**Description**	**Auth**
GET	/youtube/v3/channel		Get personal information of the user	Yes
GET	/youtube/v3/playlists		Get all playlists of the con-nected user	Yes
POST	/youtube/v3/playlists	Request body **snippet**: contain the playlist id and the video id **status**: the visibility of the playlist (public or private)	Create a playlist in the li-brary of the user.	Yes
POST	/youtube/v3/playlis-tItems	Request body **snippet:** contain the playlist id and the video id	Add a video to a playlist	Yes

Appendix 5. Data dictionary

Table users

The users table contains all the users of the application. The password is hash with SHA256.

Metadata

Column	Type	Null	Default	Comments
use_id *(Primary)*	int(11)	No		
use_login	varchar(255)	No		
use_password	varchar(255)	No		
use_email	varchar(255)	No		
use_role	varchar(255)	No		User or Admin

Table playlists

The playlists table contains all the playlists that have been created in the database. Each playlist is associated with a single user.

Metadata

Column	Type	Null	Default	Comments
pla_id *(Primary)*	int(11)	No		
pla_name	varchar(255)	No		
pla_pic	varchar(255)	Yes	*NULL*	path of the image
pla_use_id	int(11)	No		Foreign key from users table

Table content

The content table is an association table between a song and a playlist. There is also a position attribute that allows you to know the place of the song in the playlist.

Metadata

Column	Type	Null	Default	Comments
con_position *(Primary)*	int(11)	No		PK
con_tra_id *(Primary)*	int(11)	No		Foreign key from tracks table
con_pla_id *(Primary)*	int(11)	No		Foreign key from playlists table

Table tracks

This table contains all the information specific to a track.

Metadata

Column	Type	Null	Default	Comments
tra_id *(Primary)*	int(11)	No		
tra_name	varchar(255)	No		
tra_duration	int(11)	Yes	*NULL*	duration in ms
tra_number	int(11)	Yes	*NULL*	position of track in the album
tra_no_disc	int(11)	Yes	*NULL*	disc number
tra_isrc	varchar(255)	No		
tra_alb_id	int(11)	No		Foreign key from albums table

Table track_spotify

This table is an inheritance table of the tracks table. It contains all the elements specific to a track from Spotify.

Metadata

Column	Type	Null	Default	Comments
tra_spo_tra_id *(Primary)*	int(11)	No		Primary and Foreign key from tracks table
tra_spo_id	varchar(255)	No		
tra_spo_preview	varchar(255)	Yes	*NULL*	preview of the song (30sec)

Table track_deezer

This table is also an inheritance table of the tracks table. It contains all the information specific to a track from Deezer.

Metadata

Column	Type	Null	Default	Comments
tra_dee_tra_id *(Primary)*	int(11)	No		Primary and Foreign key from tracks table
tra_dee_id	varchar(255)	No		
tra_dee_preview	varchar(255)	Yes	*NULL*	preview of the song (30sec)

Table track_youtube

This table is also an inheritance table of the tracks table. It contains all the details specific to a track from YouTube.

Metadata

Column	Type	Null	Default	Comments
tra_ytb_tra_id *(Primary)*	int(11)	No		Primary and Foreign key from tracks table
tra_ytb_id	varchar(255)	No		
tra_ytb_video_id	varchar(255)	Yes	*NULL*	link of video clip if existing

Table upcs

The upcs table contains all the UPC codes associated with an album. It is possible that an album with the same metadata has different UPC numbers.

Metadata

Column	Type	Null	Default	Comments
upc_id *(Primary)*	varchar(255)	No		
upc_alb_id	int(11)	Yes	*NULL*	Foreign key from albums table

Table albums

It is in the albums table that we will insert all the information specific to an album.

Metadata

Column	Type	Null	Default	Comments
alb_id *(Primary)*	int(11)	No		
alb_name	varchar(255)	No		
alb_release_date	date	Yes	*NULL*	
alb_nb_track	int(11)	No		
alb_picture	varchar(255)	No		path of the image
alb_type	varchar(255)	No		Album, EP, Mixtape, Single

Table album_spotify

This table is an inheritance table of the albums table. It's used to define the Spotify specific data.

Metadata

Column	Type	Null	Default	Comments
alb_spo_alb_id *(Primary)*	int(11)	No		Primary and Foreign key from albums table
alb_spo_id	varchar(255)	No		

Table album_deezer

This table is also an inheritance table of the albums table. It is used to define data specific to the Deezer platform.

Metadata

Column	Type	Null	Default	Comments
alb_dee_alb_id *(Primary)*	int(11)	No		Primary and Foreign key from albums table
alb_dee_id	varchar(255)	No		

Table album_youtube

This table is an inheritance table of the albums table. It allows to define all the attributes specific to the YouTube platform.

Metadata

Column	Type	Null	Default	Comments
alb_ytb_alb_id *(Pri-mary)*	int(11)	No		Primary and Foreign key from al-bums table
alb_ytb_id	varchar(255)	No		BrowseId
alb_ytb_playlist	varchar(255)	No		playlist_id

Table artists

The artists table contains information specific to an artist. The artist is a person who performs a song.

Metadata

Column	Type	Null	Default	Comments
art_id *(Primary)*	int(11)	No		
art_name	varchar(255)	No		
art_pic	varchar(255)	Yes	*NULL*	
art_desc	varchar(10000)	Yes	*NULL*	

Table artist_spotify

The artist_spotify table is an inheritance table of the artists table. It contains attributes specific to the Spotify platform.

Metadata

Column	Type	Null	Default	Comments
art_spo_art_id *(Primary)*	int(11)	No		Primary and Foreign key from artists table
art_spo_id	varchar(255)	No		

Table artist_deezer

The artist_deezer table is an inheritance table of the artists table. It contains data specific to the Deezer platform.

Metadata

Column	Type	Null	Default	Comments
art_dee_art_id *(Primary)*	int(11)	No		Primary and Foreign key from artists table
art_dee_id	varchar(255)	No		

Table artist_youtube

The artist_youtube table is an inheritance table of the artists table. It contains the attributes specific to the YouTube platform.

Metadata

Column	Type	Null	Default	Comments
art_ytb_art_id *(Primary)*	int(11)	No		Primary and Foreign key from artists table
art_ytb_id	varchar(255)	No		
art_ytb_main_id	varchar(255)	No		

Table track_artist

The track_artist table is an association table between the tracks table and the artists table.

Metadata

Column	Type	Null	Default	Comments
tra_id *(Primary)*	int(11)	No		
art_id *(Primary)*	int(11)	No		

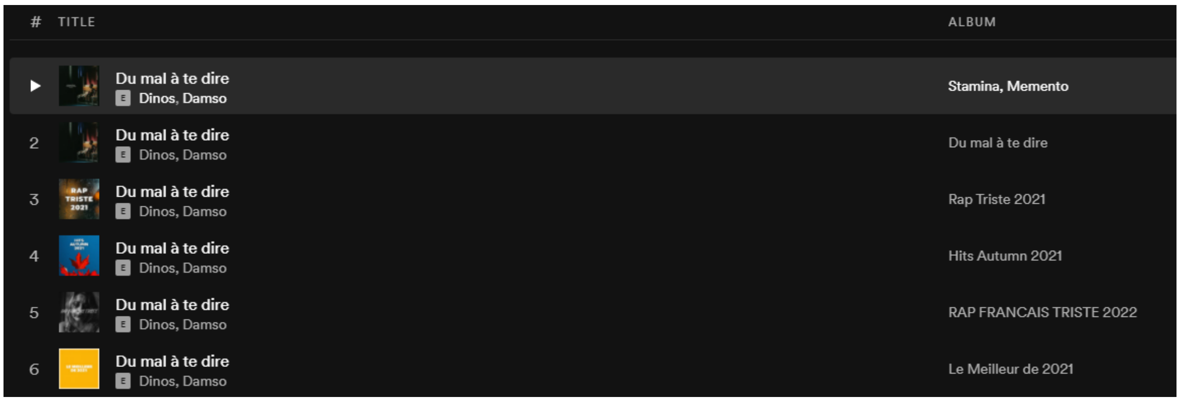
Table artist_album

The artist_album table is an association table between the artist table and the album table.

Metadata

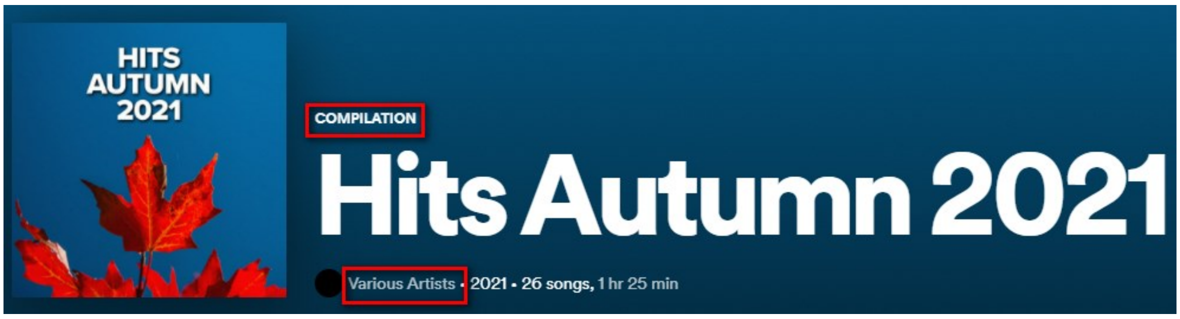
Column	Type	Null	Default	Comments
art_id *(Primary)*	int(11)	No		
alb_id *(Primary)*	int(11)	No		

Appendix 6. Example of duplicate entry

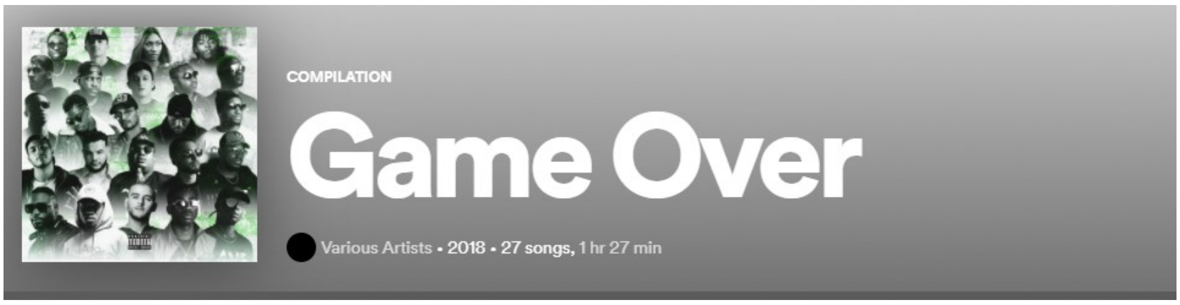


#	TITLE	ALBUM
▶	Du mal à te dire E Dinos, Damso	Stamina, Memento
2	Du mal à te dire E Dinos, Damso	Du mal à te dire
3	Du mal à te dire E Dinos, Damso	Rap Triste 2021
4	Du mal à te dire E Dinos, Damso	Hits Autumn 2021
5	Du mal à te dire E Dinos, Damso	RAP FRANCAIS TRISTE 2022
6	Du mal à te dire E Dinos, Damso	Le Meilleur de 2021

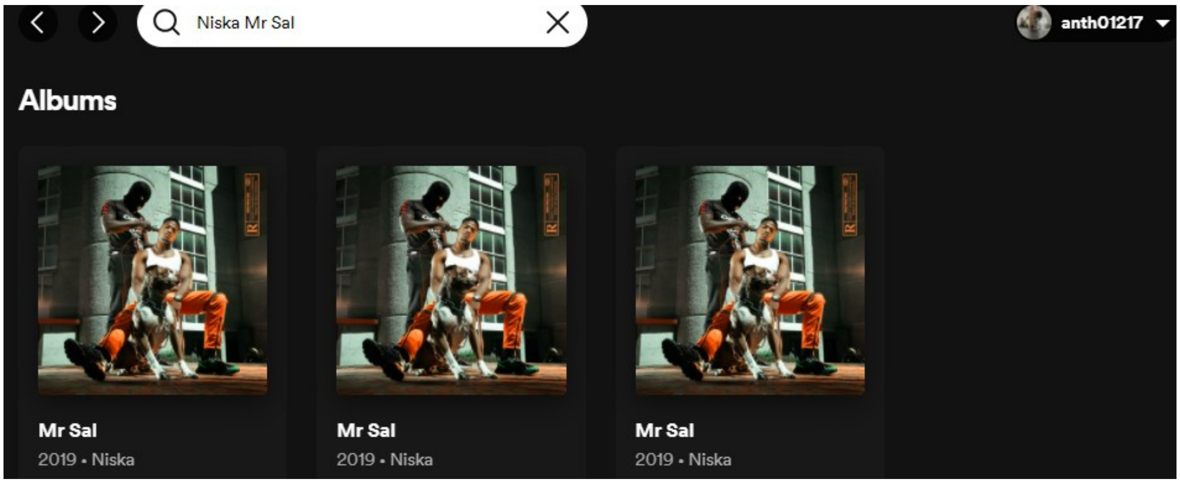
Appendix 7. Example of compilation



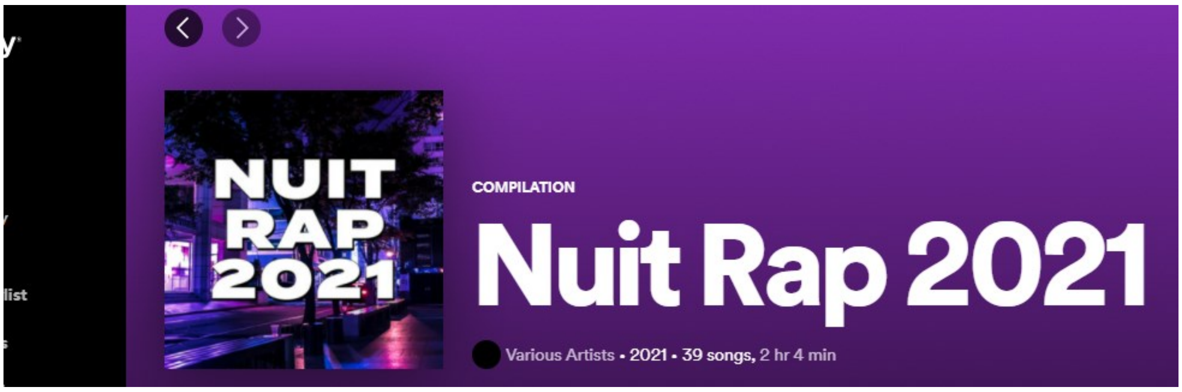
Appendix 8. Compilation with tracks that are not available elsewhere.

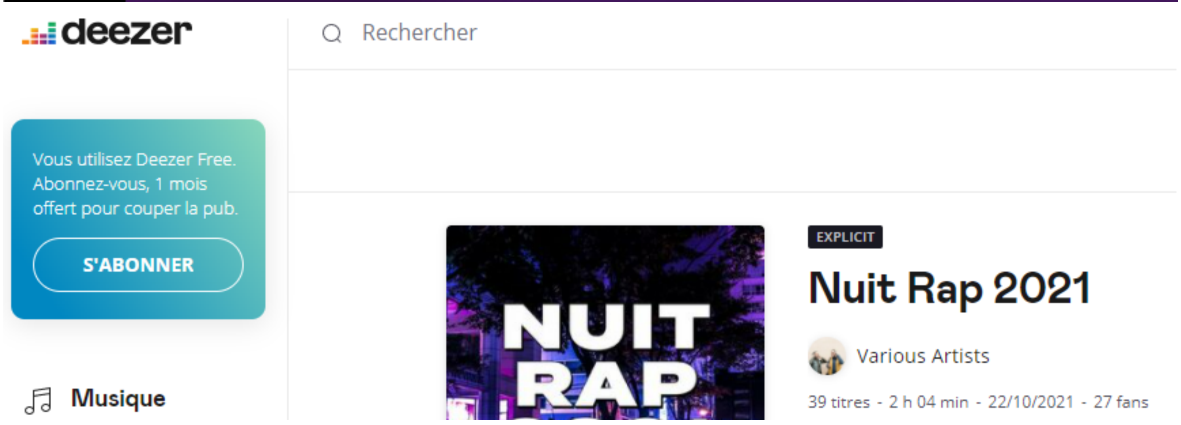


Appendix 9. Result of a search that mentions all versions of the same album

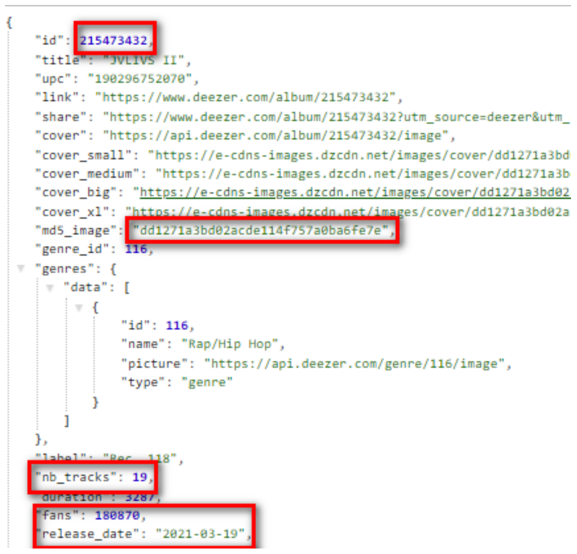


Appendix 10. Example of a compilation available on several platforms

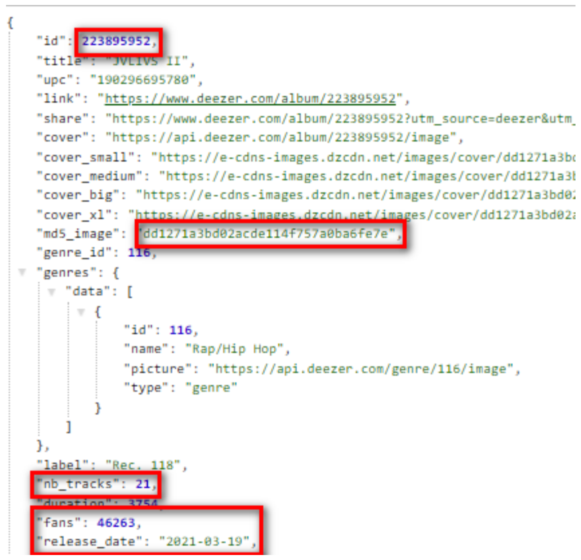



deezer
Rechercher
Vous utilisez Deezer Free. Abonnez-vous, 1 mois offert pour couper la pub.
S'ABONNER
Musique
EXPLICIT
Nuit Rap 2021
Various Artists
39 titres - 2 h 04 min - 22/10/2021 - 27 fans

Appendix 11. Comparison of MD5 between two version of an album

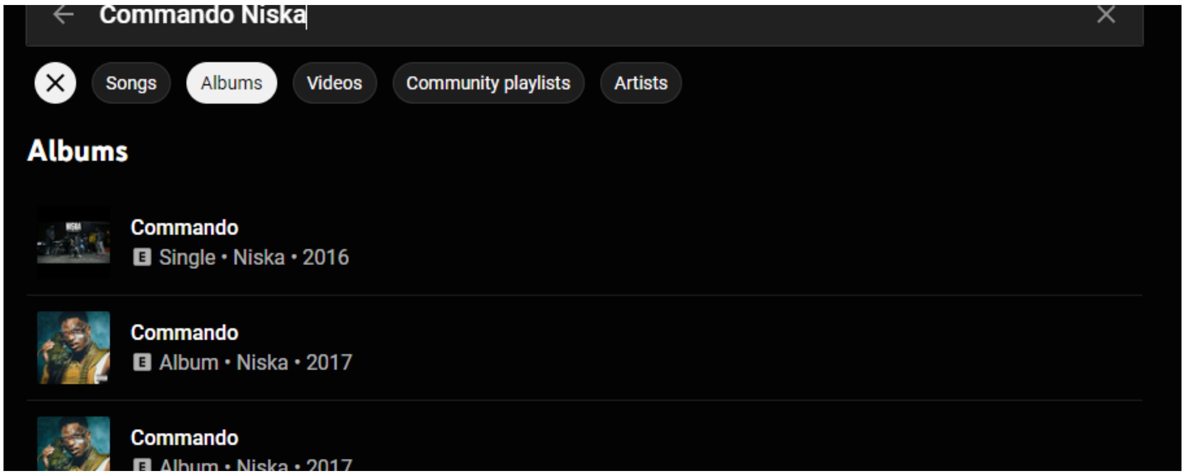


```
{
    "id": 215473432,
    "title": "JVLIVS II",
    "upc": "190296752070",
    "link": "https://www.deezer.com/album/215473432",
    "share": "https://www.deezer.com/album/215473432?utm_source=deezer&utm_
    "cover": "https://api.deezer.com/album/215473432/image",
    "cover_small": "https://e-cdns-images.dzcdn.net/images/cover/dd1271a3bd
    "cover_medium": "https://e-cdns-images.dzcdn.net/images/cover/dd1271a3b
    "cover_big": "https://e-cdns-images.dzcdn.net/images/cover/dd1271a3bd02
    "cover_xl": "https://e-cdns-images.dzcdn.net/images/cover/dd1271a3bd02a
    "md5_image": "dd1271a3bd02acde114f757a0ba6fe7e",
    "genre_id": 116,
  ▾ "genres": {
      ▾ "data": [
          ▾ {
                "id": 116,
                "name": "Rap/Hip Hop",
                "picture": "https://api.deezer.com/genre/116/image",
                "type": "genre"
            }
        ]
    },
    "label": "Rec. 118",
    "nb_tracks": 19,
    "duration": 3287,
    "fans": 180870,
    "release_date": "2021-03-19",
```



```
{
    "id": 223895952,
    "title": "JVLIVS II",
    "upc": "190296695780",
    "link": "https://www.deezer.com/album/223895952",
    "share": "https://www.deezer.com/album/223895952?utm_source=deezer&utm_
    "cover": "https://api.deezer.com/album/223895952/image",
    "cover_small": "https://e-cdns-images.dzcdn.net/images/cover/dd1271a3bd
    "cover_medium": "https://e-cdns-images.dzcdn.net/images/cover/dd1271a3b
    "cover_big": "https://e-cdns-images.dzcdn.net/images/cover/dd1271a3bd02
    "cover_xl": "https://e-cdns-images.dzcdn.net/images/cover/dd1271a3bd02a
    "md5_image": "dd1271a3bd02acde114f757a0ba6fe7e",
    "genre_id": 116,
  ▾ "genres": {
      ▾ "data": [
          ▾ {
                "id": 116,
                "name": "Rap/Hip Hop",
                "picture": "https://api.deezer.com/genre/116/image",
                "type": "genre"
            }
        ]
    },
    "label": "Rec. 118",
    "nb_tracks": 21,
    "duration": 3754,
    "fans": 46263,
    "release_date": "2021-03-19",
```

Appendix 12. Example of bug with the search



Commando Niska

Songs Albums Videos Community playlists Artists

Albums

Commando
Single • Niska • 2016

Commando
Album • Niska • 2017

Commando
Album • Niska • 2017

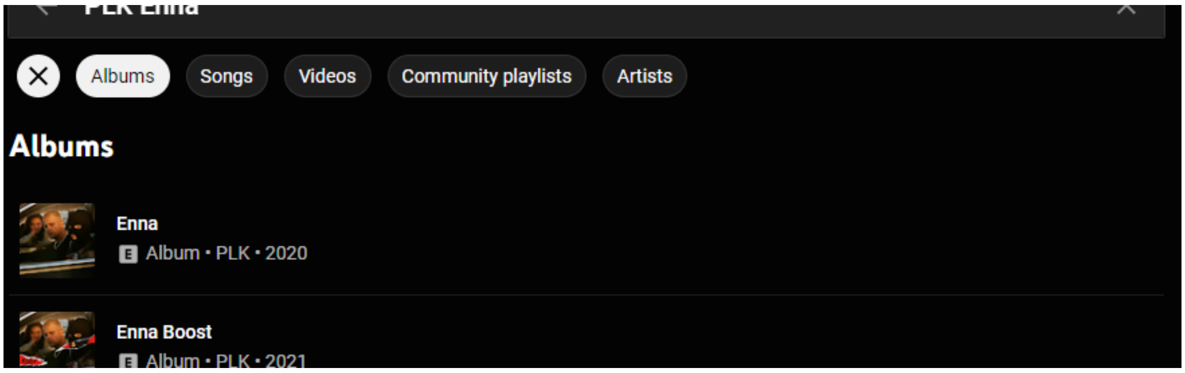
Appendix 13. Comparison table of a song in several copies

Type	Title	Album's name	videoId	View	Release date
Song	Grand bain	Grand bain (feat. Ninho) – Single MPREb_KSNrRFSV9KB	Glu_FyvFgLM	6.1M	19/06/20
Video	DADJU - Grand Bain ft. Ninho (Audio Officiel)	X	P7ZjlJINFk8	37,7M	24/06/20
Video	DADJU - Grand Bain ft. Ninho (Clip Officiel)	X	NPSvCJ9v7vI	64,2M	30/07/20
Song*	Grand bain	Poison Ou Antidote – Album MPREb_01sLfYiTP8c	YsD7zx3s3I0	5,2M	24/09/20
Song	Grand bain	Poison ou Antidote (Miel Book Edition) – Album MPREb_lu8F0tiIHb6	WmHSb6tSrDw	55K	03/06/21
Song	Grand bain	Poison ou Antidote (Miel Book Edition) – Album MPREb_w1E1vs1NmsG	YaAOSfKcoEg	7,5K	17/12/21

Type	Title	Album's name	videoId	View	Release date
Song*	Bénef	Bénef – Single MPREb_KjhTS8HtYf7	d27x8hu1EsE	2,9M	31/07/20
Song	Bénef	Enna – Album MPREb_tOoh1DHKdnb	c81jGqiPe0U	2,2M	28/08/20
Song	Bénef	Enna Boost – Album MPREb_dWq8taw0HjQ	8wsuzb3sRP8	123K	12/11/21

* These songs are referred to the search engine.

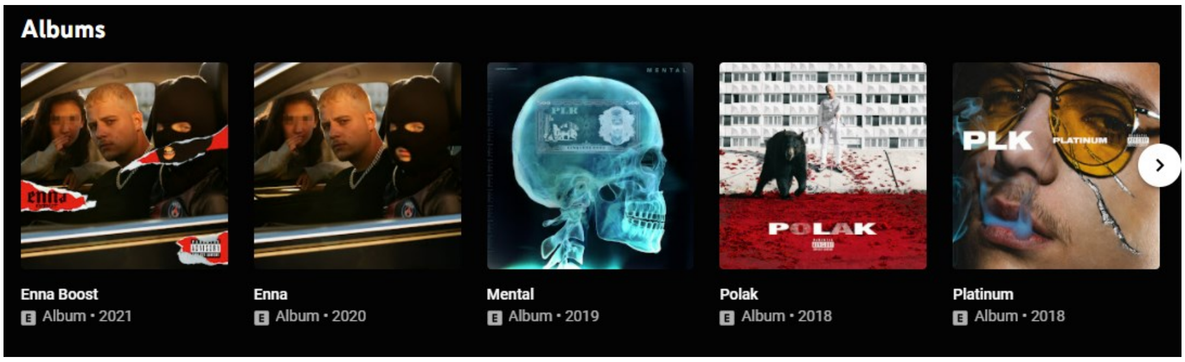
Appendix 14. Example of reissue separated into two elements on YouTube Music



Albums

Enna
E Album • PLK • 2020

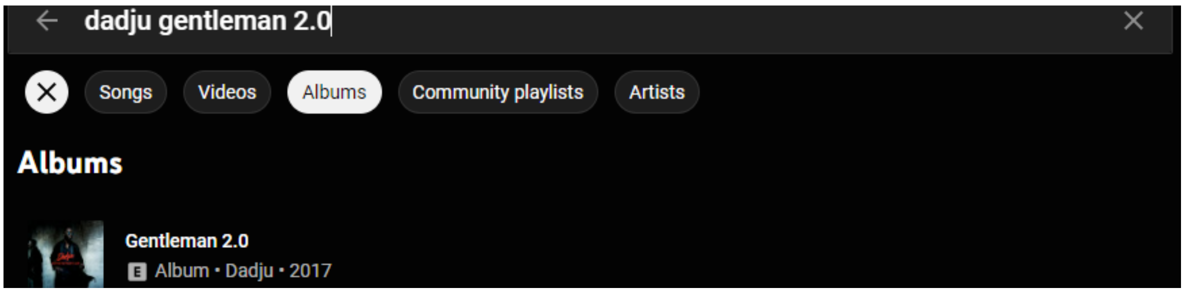
Enna Boost
E Album • PLK • 2021



Albums

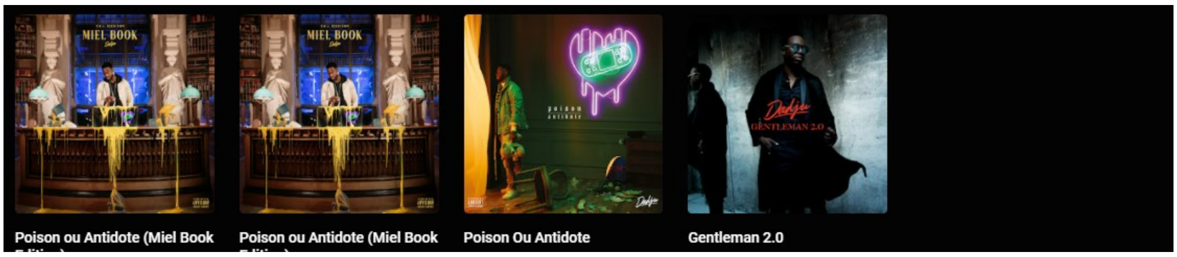
Enna Boost	Enna	Mental	Polak	Platinum
E Album • 2021	E Album • 2020	E Album • 2019	E Album • 2018	E Album • 2018

Appendix 15. Example of reissue tying together



Albums

Gentleman 2.0
E Album • Dadju • 2017

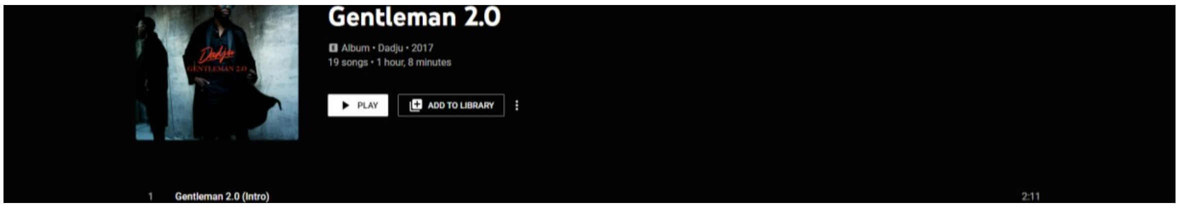


Poison ou Antidote (Miel Book Edition)

Poison ou Antidote (Miel Book Edition)

Poison Ou Antidote

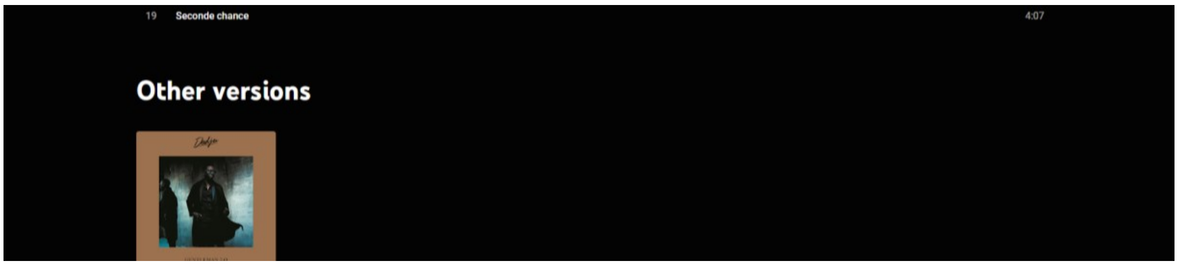
Gentleman 2.0



Gentleman 2.0

E Album • Dadju • 2017
19 songs • 1 hour, 8 minutes

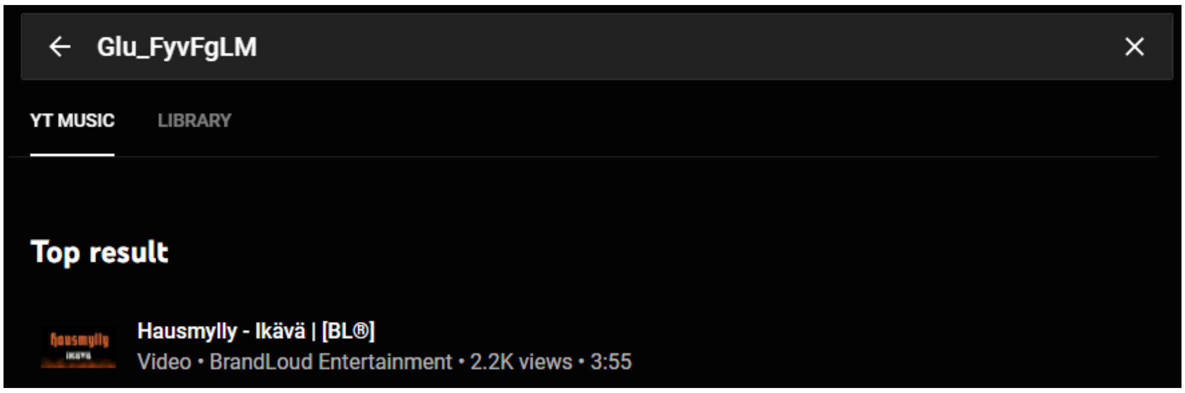
1 Gentleman 2.0 (Intro) 2:11



19 Seconde chance 4:07

Other versions

Appendix 16. Search by videoId from the comparison table in Appendix 13

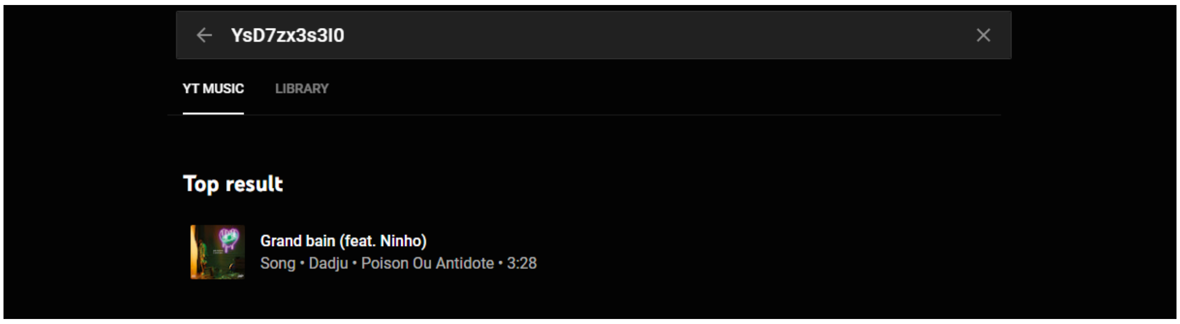


Glu_FyvFgLM

YT MUSIC LIBRARY

Top result

Hausmylly - Ikävä | [BL®]
Video • BrandLoud Entertainment • 2.2K views • 3:55

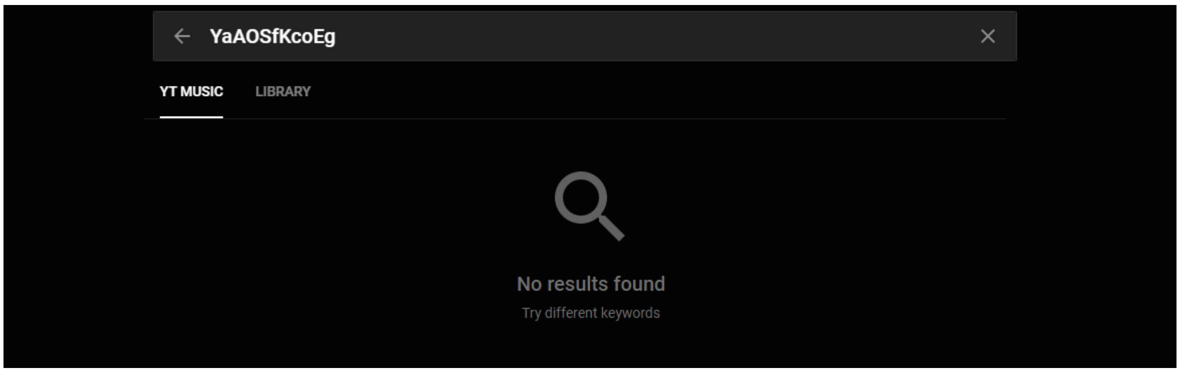


YsD7zx3s3I0

YT MUSIC LIBRARY

Top result

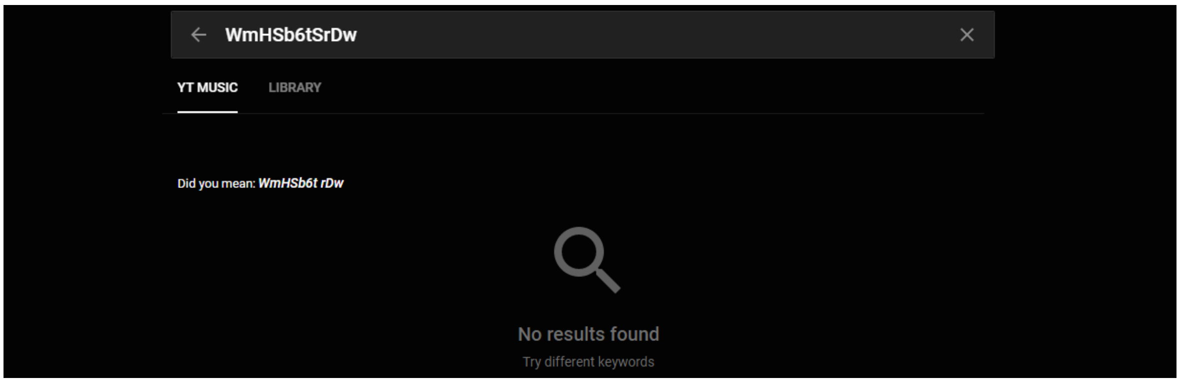
Grand bain (feat. Ninho)
Song • Dadju • Poison Ou Antidote • 3:28



YaAOSfKcoEg

YT MUSIC LIBRARY

No results found
Try different keywords



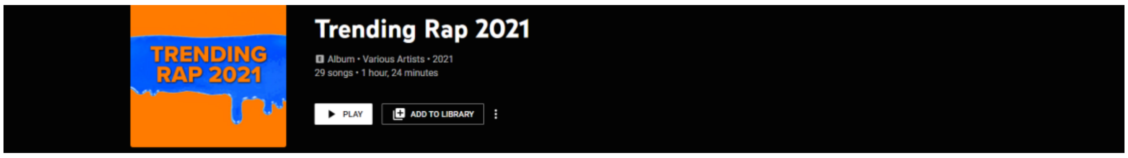
WmHSb6tSrDw

YT MUSIC LIBRARY

Did you mean: *WmHSb6t rDw*

No results found
Try different keywords

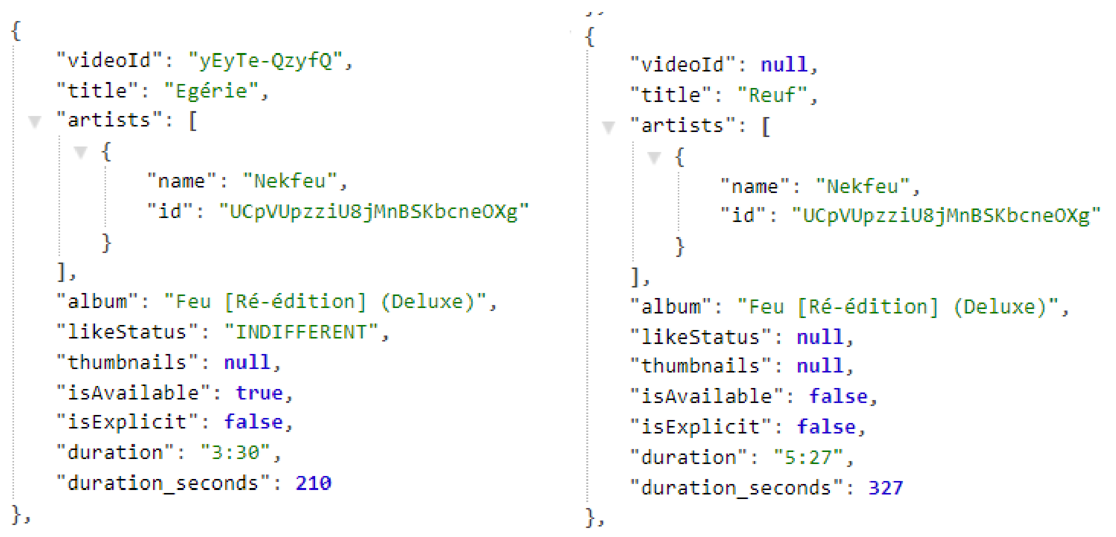
Appendix 17. Compilation album with song redundancy



Trending Rap 2021
Album • Various Artists • 2021
29 songs • 1 hour, 24 minutes

The URL of this album is : https://music.youtube.com/playlist?list=OLAK5uy_nk8lQWJHg5aJUyR4Ezd55Jlu12YBeMzUI

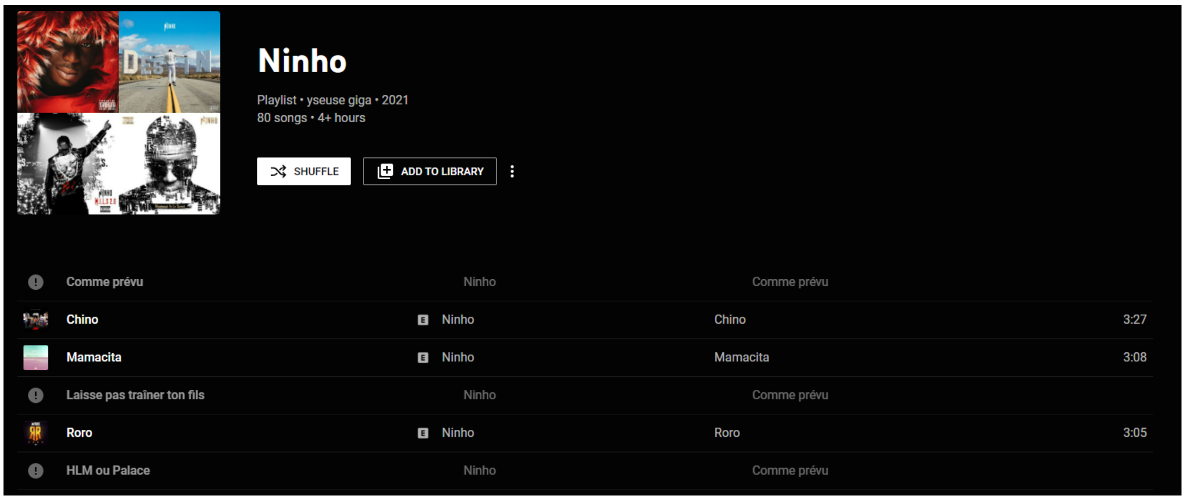
Appendix 18. Difference between two tracks of an album where one of them is disabled



```
{
    "videoId": "yEyTe-QzyfQ",
    "title": "Egérie",
    "artists": [
        {
            "name": "Nekfeu",
            "id": "UCpVUpzziU8jMnBSKbcneOXg"
        }
    ],
    "album": "Feu [Ré-édition] (Deluxe)",
    "likeStatus": "INDIFFERENT",
    "thumbnails": null,
    "isAvailable": true,
    "isExplicit": false,
    "duration": "3:30",
    "duration_seconds": 210
},
```

```
{
    "videoId": null,
    "title": "Reuf",
    "artists": [
        {
            "name": "Nekfeu",
            "id": "UCpVUpzziU8jMnBSKbcneOXg"
        }
    ],
    "album": "Feu [Ré-édition] (Deluxe)",
    "likeStatus": null,
    "thumbnails": null,
    "isAvailable": false,
    "isExplicit": false,
    "duration": "5:27",
    "duration_seconds": 327
},
```

Appendix 19. Example of a playlist with deactivated songs



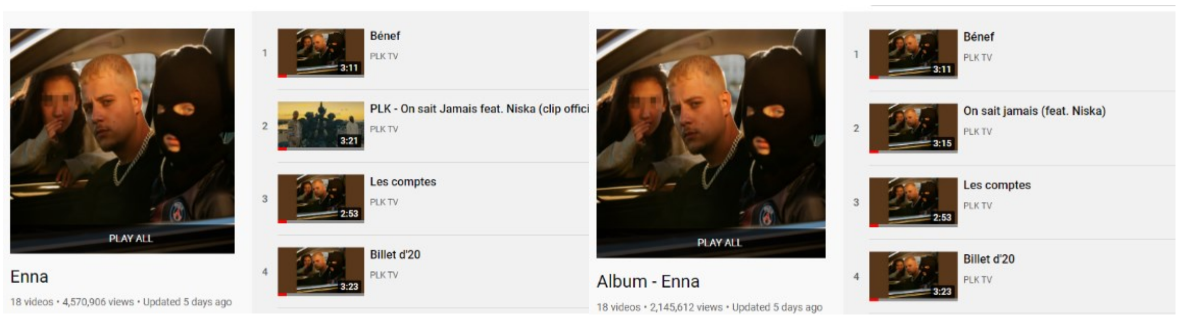
Ninho

Playlist • yseuse giga • 2021
80 songs • 4+ hours

Comme prévu		Ninho	Comme prévu	
Chino	E	Ninho	Chino	3:27
Mamacita	E	Ninho	Mamacita	3:08
Laisse pas traîner ton fils		Ninho	Comme prévu	
Roro	E	Ninho	Roro	3:05
HLM ou Palace		Ninho	Comme prévu	

The URL of this playlist is https://music.youtube.com/playlist?list=PLDYH6Z7D5z6vuige-ToxIHFISTeK-rRzft

Appendix 20. Two versions of the same album where one is made of video clips.



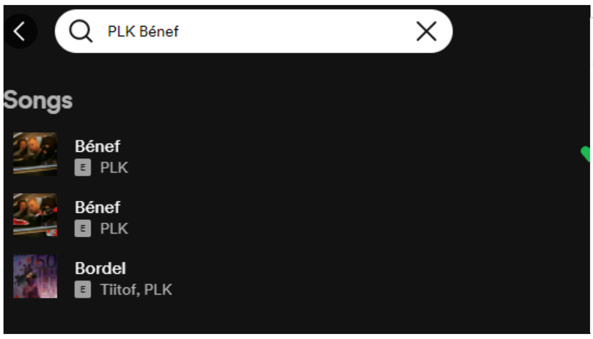
Enna
18 videos • 4,570,906 views • Updated 5 days ago

1	3:11	Bénef	PLK TV
2	3:21	PLK - On sait Jamais feat. Niska (clip offici	PLK TV
3	2:53	Les comptes	PLK TV
4	3:23	Billet d'20	PLK TV

Album - Enna
18 videos • 2,145,612 views • Updated 5 days ago

1	3:11	Bénef	PLK TV
2	3:15	On sait jamais (feat. Niska)	PLK TV
3	2:53	Les comptes	PLK TV
4	3:23	Billet d'20	PLK TV

Appendix 21. Comparison of track available on each platform

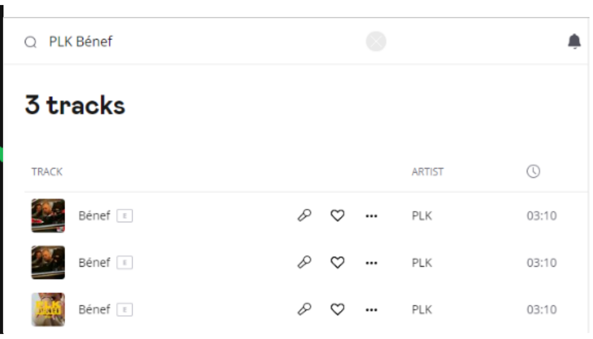

PLK Bénef

Songs

Bénef
E PLK

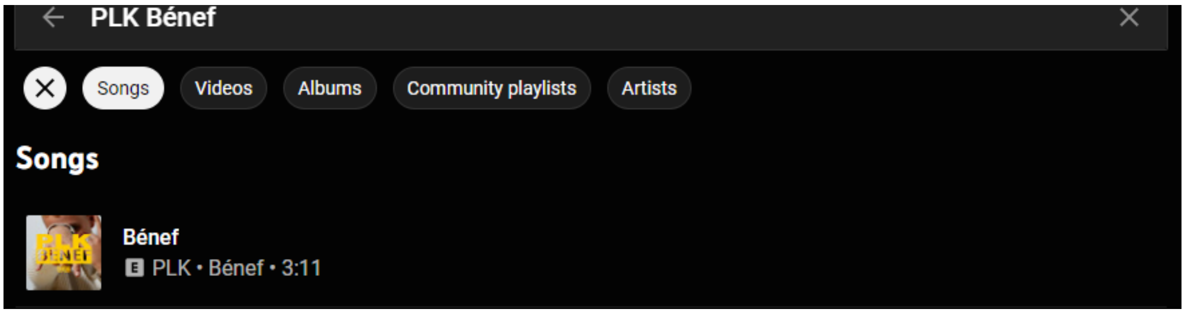
Bénef
E PLK

Bordel
E Tiitof, PLK


PLK Bénef

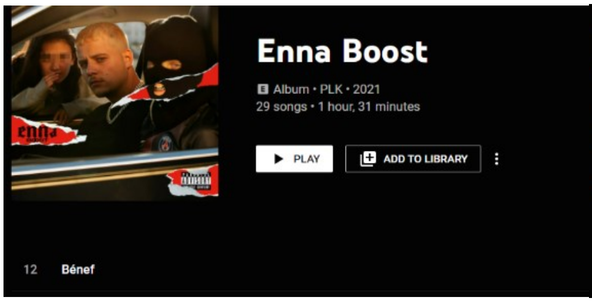
3 tracks

TRACK	ARTIST	
Bénef	PLK	03:10
Bénef	PLK	03:10
Bénef	PLK	03:10


PLK Bénef

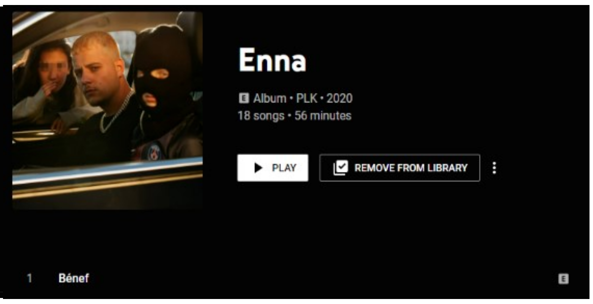
Songs

Bénef
E PLK • Bénef • 3:11


Enna Boost

E Album • PLK • 2021
29 songs • 1 hour, 31 minutes

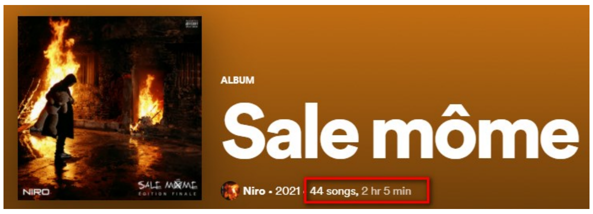
12 Bénef


Enna

E Album • PLK • 2020
18 songs • 56 minutes

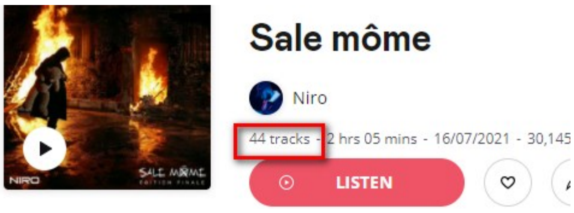
1 Bénef

Appendix 22. Different structure on YouTube Music compared to other platforms


ALBUM

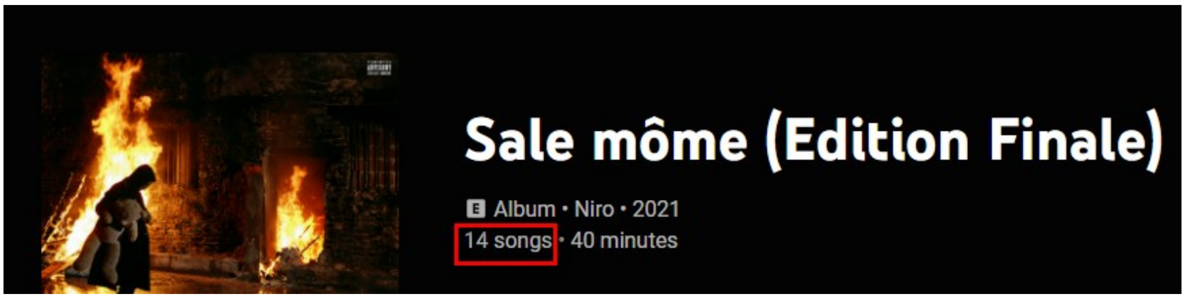
Sale môme

Niro • 2021 44 songs, 2 hr 5 min


Sale môme

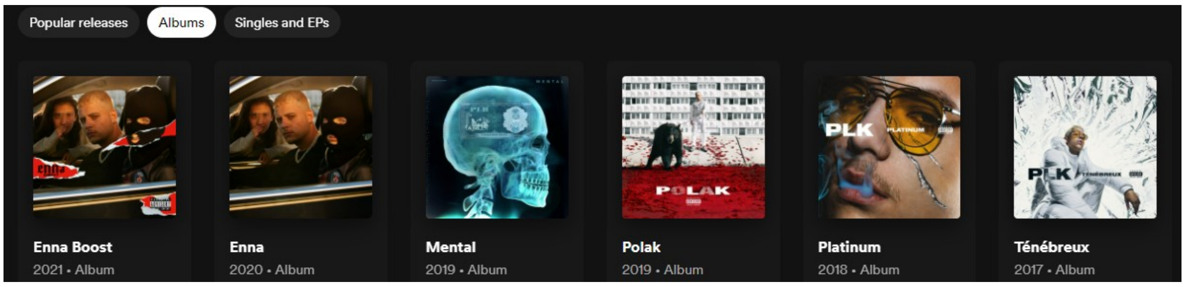
Niro

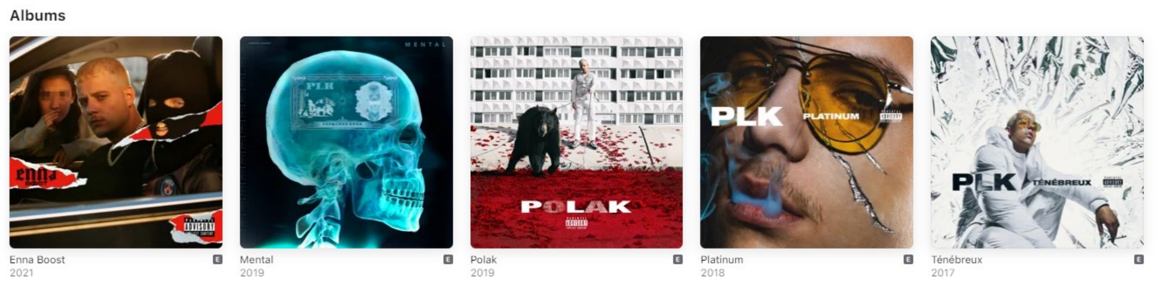
44 tracks - 2 hrs 05 mins - 16/07/2021 - 30,145

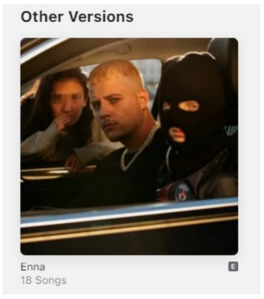

Sale môme (Edition Finale)

E Album • Niro • 2021
14 songs • 40 minutes

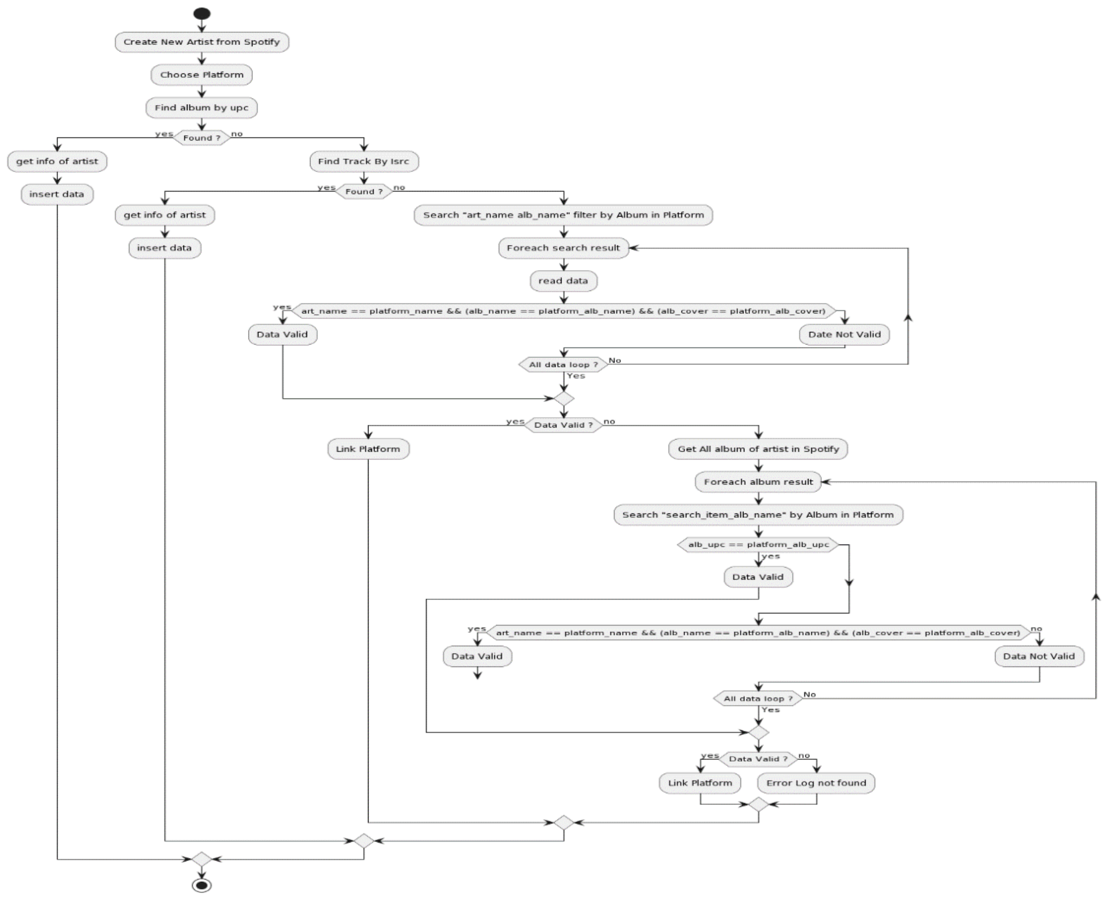
Appendix 23. Implementation on Spotify and Apple music. The missing album is referenced as a version.







Appendix 24. Use case diagram for Analyse Artist based on Appendix 25

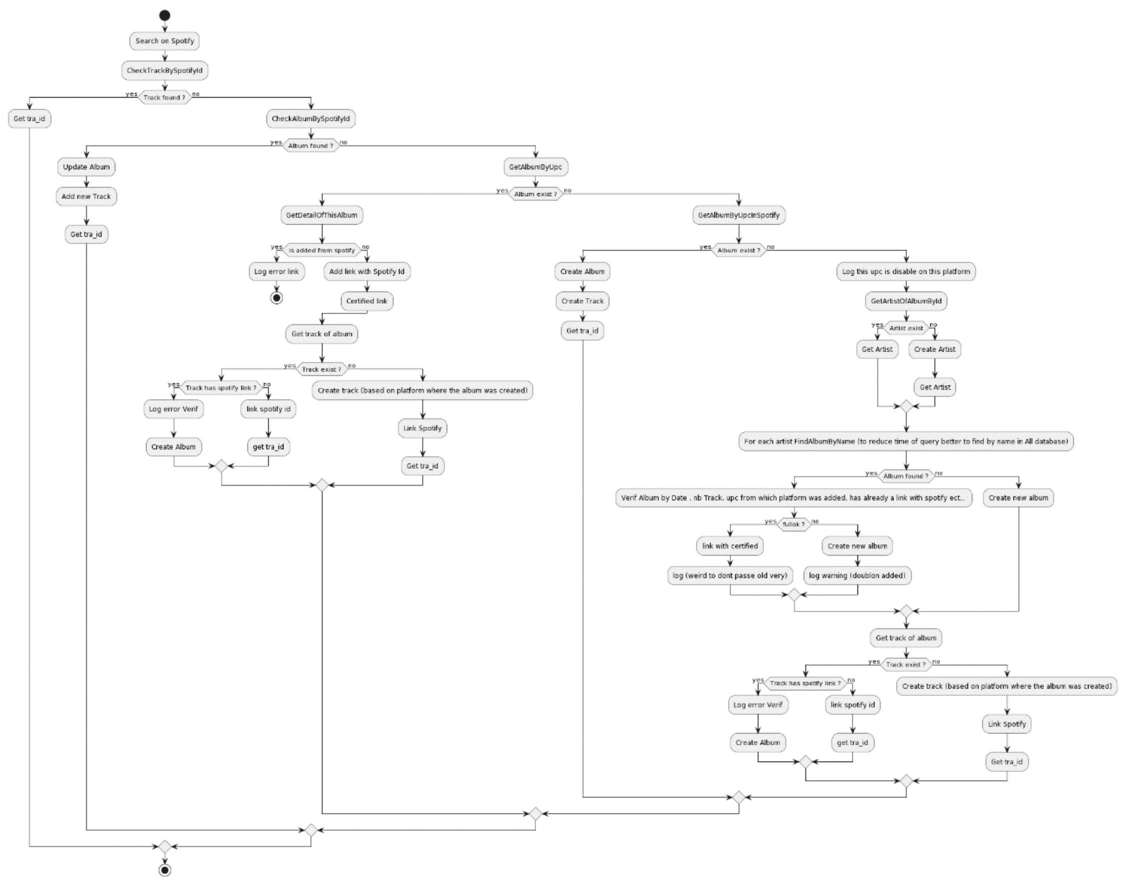

Create New Artist from Spotify
Choose Platform
Find album by upc
yes
Found ?
no
get info of artist
insert data
Find Track By Isrc
yes
Found ?
no
get info of artist
insert data
Search "art_name alb_name" filter by Album in Platform
Foreach search result
read data
yes
art_name == platform_name && (alb_name == platform_alb_name) && (alb_cover == platform_alb_cover)
Data Valid
Date Not Valid
All data loop ?
No
Yes
yes
Data Valid ?
no
Link Platform
Get All album of artist in Spotify
Foreach album result
Search "search_item_alb_name" by Album in Platform
alb_upc == platform_alb_upc
yes
Data Valid
yes
art_name == platform_name && (alb_name == platform_alb_name) && (alb_cover == platform_alb_cover)
no
Data Valid
Data Not Valid
All data loop ?
No
Yes
yes
Data Valid ?
no
Link Platform
Error Log not found

Appendix 25. Code executed to create diagram Appendix 24.

The code can be executed on this website http://www.plantuml.com/plantuml/uml

```
@startuml
start
:Create New Artist from Spotify;
:Choose Platform;
:Find album by upc;
if(Found ?) then (yes)
  :get info of artist;
  :insert data;
else (no)
:Find Track By Isrc;
if(Found ?) then (yes)
  :get info of artist;
  :insert data;
else (no)
:Search "art_name alb_name" filter by Album in Platform;
repeat :Foreach search result;
  :read data;
  if (art_name == platform_name && (alb_name == platform_alb_name) && (alb_cover == plat-
form_alb_cover)) then (yes)
     :Data Valid;
     break
  else
    :Date Not Valid;
  endif
repeat while (All data loop ?) is (No) not (Yes)
if (Data Valid ?) then (yes)
  :Link Platform;
else (no)
  :Get All album of artist in Spotify;
  repeat :Foreach album result;
   :Search "search_item_alb_name" by Album in Platform;
  if(alb_upc == platform_alb_upc) then (yes)
    :Data Valid;
    break
  endif
   if(art_name == platform_name && (alb_name == platform_alb_name) && (alb_cover == plat-
form_alb_cover)) then (yes)
     :Data Valid;
     break
   else (no)
    :Data Not Valid;
   endif
  repeat while (All data loop ?) is (No) not (Yes)
  if (Data Valid ?) then (yes)
    :Link Platform;
  else (no)
    :Error Log not found;
  endif
endif
endif
endif
stop
@enduml
```

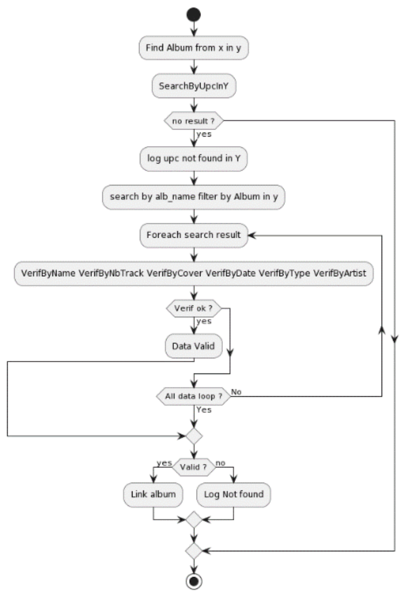
Appendix 26. Use case diagram for Analyse Track based on Appendix 27



Appendix 27. Code executed to create diagram Appendix 26

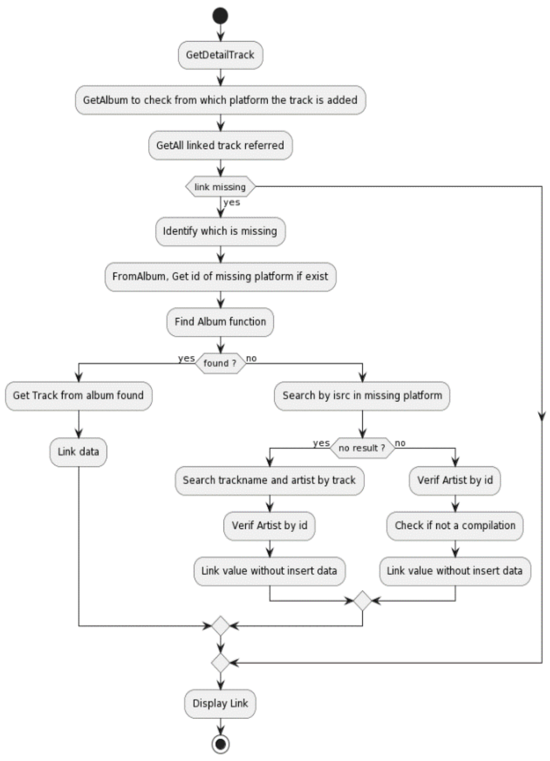
```
@startuml
start
:Search on Spotify;
:CheckTrackBySpotifyId;
if (Track found ?) then (yes)
  :Get tra_id;
else (no)
  :CheckAlbumBySpotifyId;
  if (Album found ?) then (yes)
     :Update Album;
     :Add new Track;
     :Get tra_id;
  else (no)
    :GetAlbumByUpc;
    if (Album exist ?) then (yes)
      :GetDetailOfThisAlbum;
      if(is added from spotify) then (yes)
        :Log error link;
        stop
      else (no)
        :Add link with Spotify Id;
        :Certified link;
      endif
      :Get track of album;
      if (Track exist ?) then (yes)
          if (Track has spotify link ?) then (yes)
            :Log error Verif;
            :Create Album;
          else (no)
            :link spotify id;
            :get tra_id;
          endif
        else (no)
          :Create track (based on platform where the album was created);
          :Link Spotify;
          :Get tra_id;
        endif
    else (no)
      :GetAlbumByUpcInSpotify;
      if (Album exist ?) then (yes)
        :Create Album;
        :Create Track;
        :Get tra_id;
      else (no)
        :Log this upc is disable on this platform;
        :GetArtistOfAlbumById;
        if (Artist exist) then (yes)
          :Get Artist;
        else (no)
          :Create Artist;
          :Get Artist;
        endif
        :For each artist FindAlbumByName (to reduce time of query better to find by name in All
database);
        if(Album found ?) then (yes)
          :Verif Album by Date , nb Track, upc from which platform was added, has already a link
with spotify ect... ;
          if(fullok ?) then (yes)
            :link with certified;
            :log (weird to dont passe old very);
          else (no)
            :Create new album;
            :log warning (doublon added);
          endif
        else (no)
          :Create new album;
        endif
        :Get track of album;
        if (Track exist ?) then (yes)
          if (Track has spotify link ?) then (yes)
            :Log error Verif;
            :Create Album;
          else (no)
            :link spotify id;
            :get tra_id;
          endif
        else (no)
          :Create track (based on platform where the album was created);
          :Link Spotify;
          :Get tra_id;
        endif
      endif
    endif
  endif
endif
stop
@enduml
```

Appendix 28. Code and diagram for Link Album from Spotify to other platform process



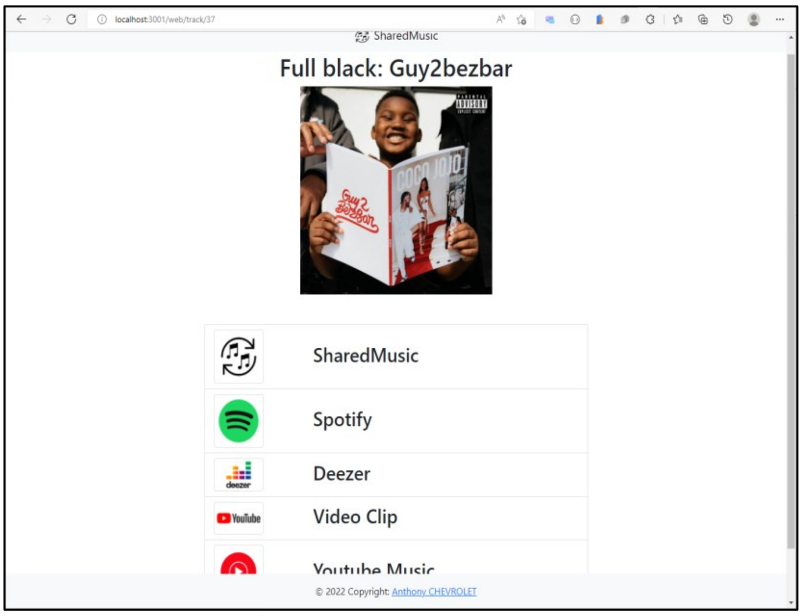
```
start
:Find Album from x in y;
:SearchByUpcInY;
if (no result ?) then (yes)
  :log upc not found in Y;
  :search by alb_name filter by Album in y;
  repeat :Foreach search result;
    :VerifByName VerifByNbTrack VerifByCover
VerifByDate VerifByType VerifByArtist;
    if (Verif ok ?) then (yes)
      :Data Valid;
      break
    endif
  repeat while (All data loop ?) is (No) not
(Yes)
  if (Valid ?) then (yes)
    :Link album;
  else (no)
    :Log Not found;
  endif
endif
stop
```

Appendix 29. Code and diagram for Link Track from Spotify to other platform process


GetDetailTrack
GetAlbum to check from which platform the track is added
GetAll linked track referred
link missing
yes
Identify which is missing
FromAlbum, Get id of missing platform if exist
Find Album function
yes
found ?
no
Get Track from album found
Search by isrc in missing platform
Link data
yes
no result ?
no
Search trackname and artist by track
Verif Artist by id
Verif Artist by id
Check if not a compilation
Link value without insert data
Link value without insert data
Display Link

```
@startuml
start
:GetDetailTrack;
:GetAlbum to check from which platform the
track is added;
:GetAll linked track referred;
if (link missing) then (yes)
  :Identify which is missing;
  :FromAlbum, Get id of missing platform if
exist;
  :Find Album function;
  if (found ?) then (yes)
    :Get Track from album found;
    :Link data;
  else (no)
     :Search by isrc in missing platform;
        if(no result ?) then (yes)
          :Search trackname and artist by
track;
          :Verif Artist by id;
          :Link value without insert data;
        else (no)
          :Verif Artist by id;
          :Check if not a compilation;
          :Link value without insert data;
        endif
  endif
endif
:Display Link;
stop
@enduml
```

Appendix 30. Visual of the different resource sharing web pages on computer and smartphone


localhost:3001/web/track/37

SharedMusic

Full black: Guy2bezbar

SharedMusic

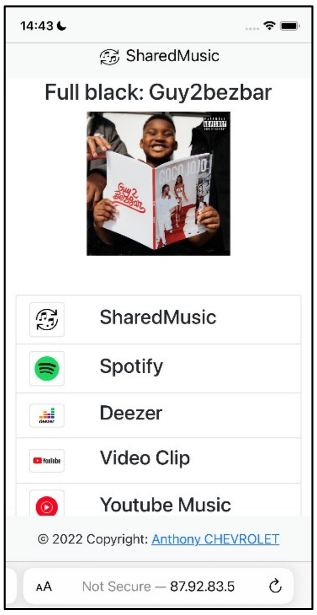
Spotify

Deezer

Video Clip

Youtube Music

© 2022 Copyright: Anthony CHEVROLET


14:43

SharedMusic

Full black: Guy2bezbar

SharedMusic

Spotify

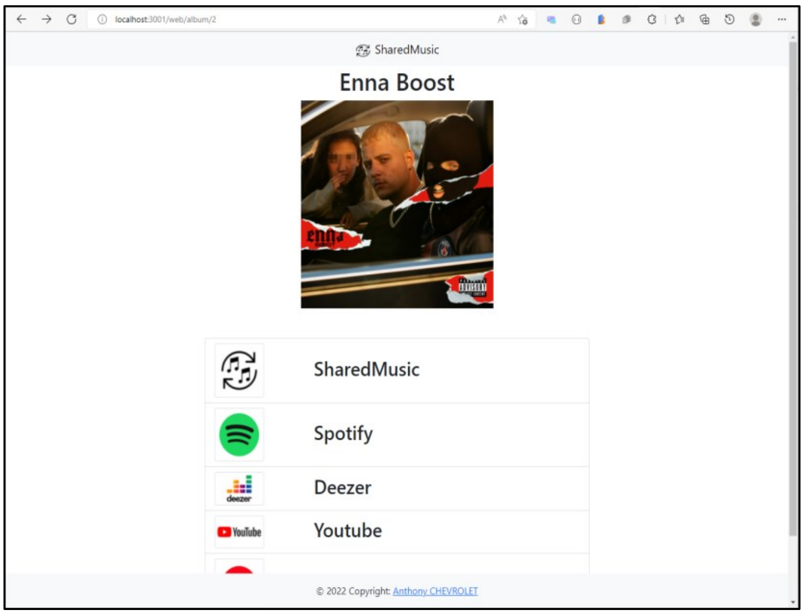
Deezer

Video Clip

Youtube Music

© 2022 Copyright: Anthony CHEVROLET

AA Not Secure — 87.92.83.5


localhost:3001/web/album/2

SharedMusic

Enna Boost

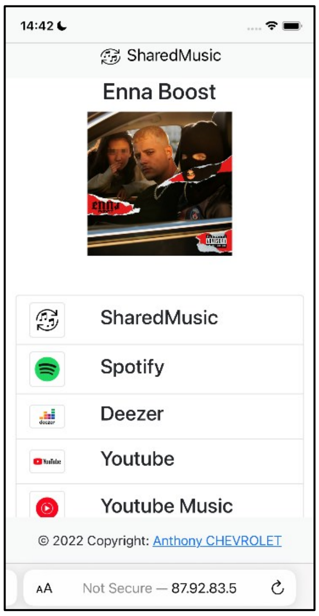
SharedMusic

Spotify

Deezer

Youtube

© 2022 Copyright: Anthony CHEVROLET


14:42

SharedMusic

Enna Boost

SharedMusic

Spotify

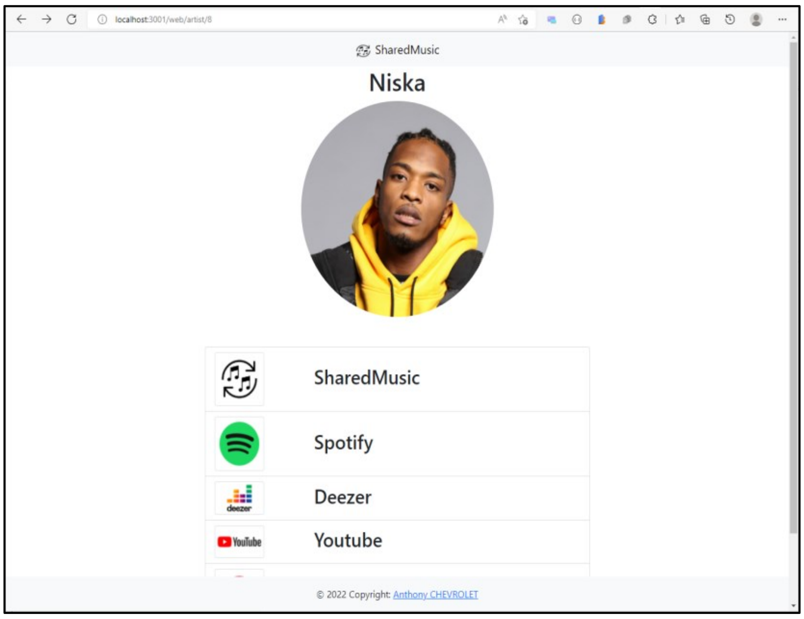
Deezer

Youtube

Youtube Music

© 2022 Copyright: Anthony CHEVROLET

AA Not Secure — 87.92.83.5


localhost:3001/web/artist/8

SharedMusic

Niska

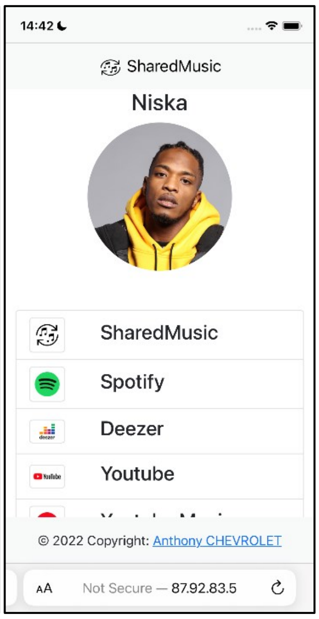
SharedMusic

Spotify

Deezer

Youtube

© 2022 Copyright: Anthony CHEVROLET


14:42

SharedMusic

Niska

SharedMusic

Spotify

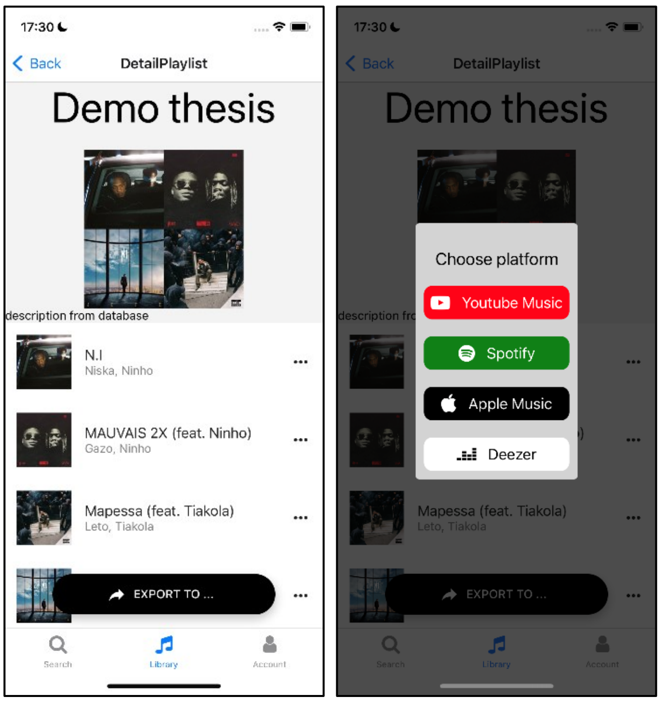
Deezer

Youtube

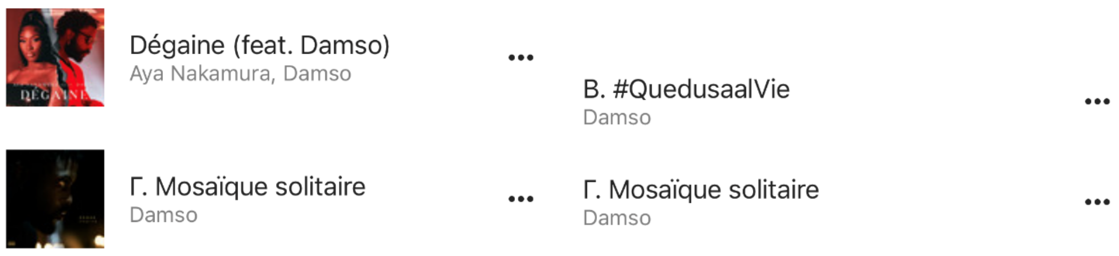
© 2022 Copyright: Anthony CHEVROLET

AA Not Secure — 87.92.83.5

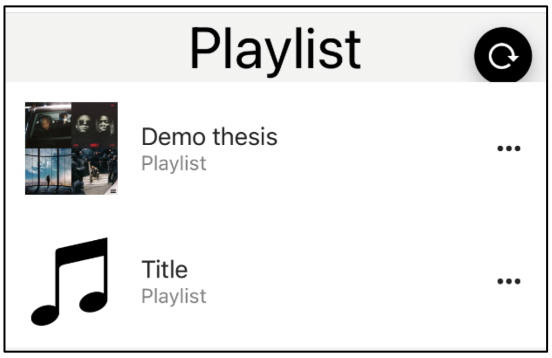
Appendix 31. Visual of the CustomModal component



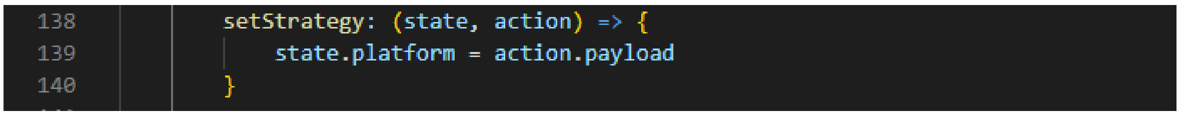
Appendix 32. Visual of an ItemSong with and without cover



Appendix 33. Visual of an ItemPlaylist with and without image

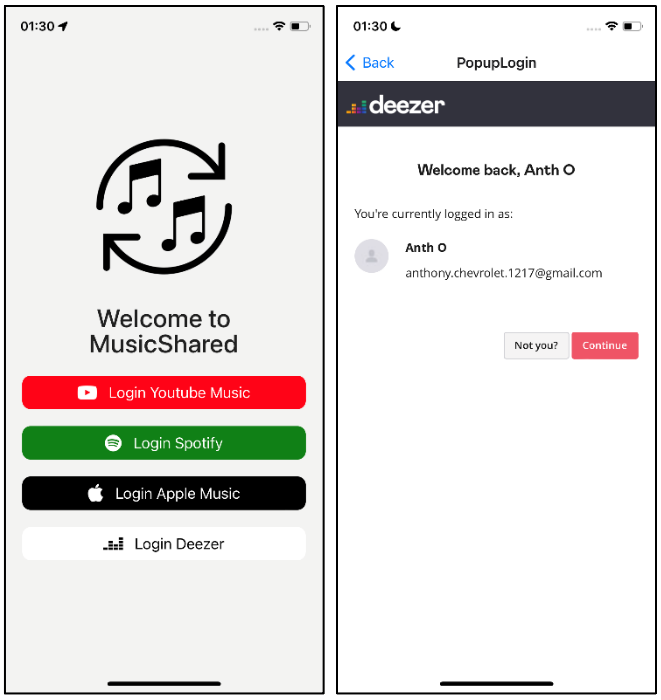


Appendix 34. Example of using a Redux function by passing a parameter



```
    setStrategy: (state, action) => {
      state.platform = action.payload
    }
    let OnPressLoginGoogle = _ => {
        dispatch(Reducer.fn.setStrategy("youtube"))
        console.log(urlPaste);
```

Appendix 35. Visual of the login page and login popup page



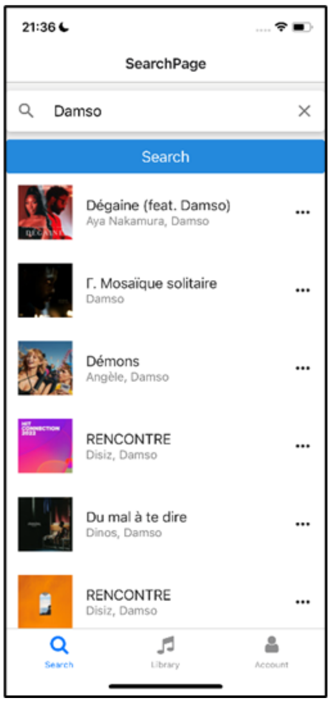
Welcome back, Anth O

You're currently logged in as:

Anth O

anthony.chevrolet.1217@gmail.com

Appendix 36. Visual of the search page



Dégaine (feat. Damso)
Aya Nakamura, Damso

Γ. Mosaïque solitaire
Damso

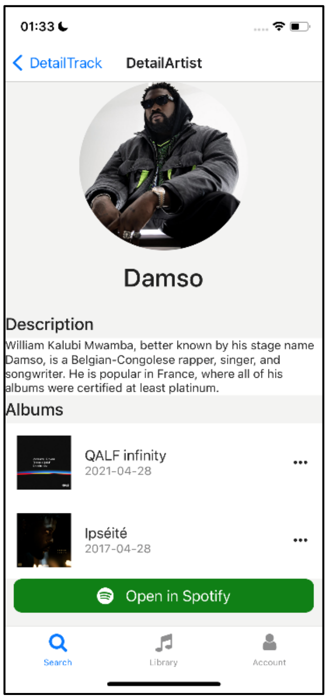
Démons
Angèle, Damso

RENCONTRE
Disiz, Damso

Du mal à te dire
Dinos, Damso

RENCONTRE
Disiz, Damso

Appendix 37. Visual of the detailed page of the different resources


01:33

DetailTrack DetailArtist

Damso

Description

William Kalubi Mwamba, better known by his stage name Damso, is a Belgian-Congolese rapper, singer, and songwriter. He is popular in France, where all of his albums were certified at least platinum.

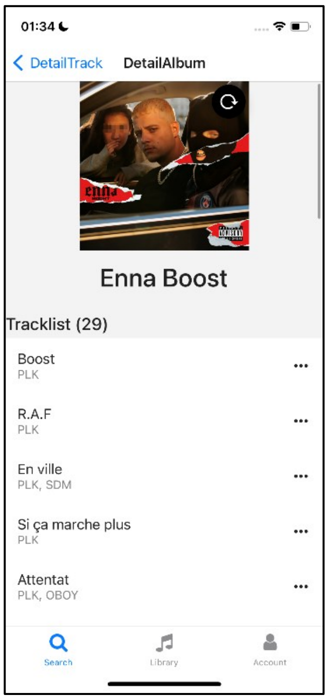
Albums

QALF infinity
2021-04-28

Ipséité
2017-04-28

Open in Spotify

Search Library Account


01:34

DetailTrack DetailAlbum

Enna Boost

Tracklist (29)

Boost
PLK

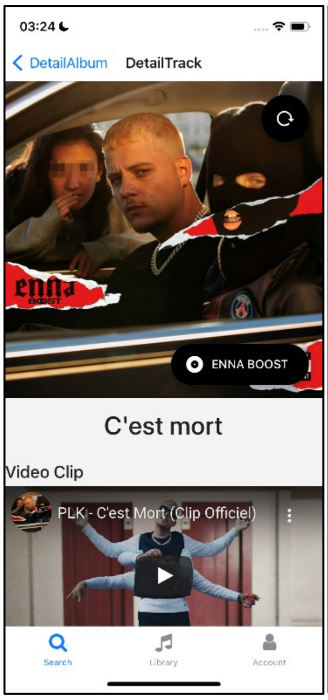
R.A.F
PLK

En ville
PLK, SDM

Si ça marche plus
PLK

Attentat
PLK, OBOY

Search Library Account


03:24

DetailAlbum DetailTrack

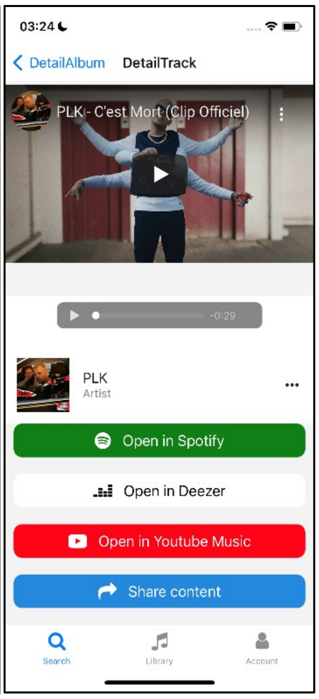
ENNA BOOST

C'est mort

Video Clip

PLK - C'est Mort (Clip Officiel)

Search Library Account


03:24

DetailAlbum DetailTrack

PLK - C'est Mort (Clip Officiel)

-0:29

PLK
Artist

Open in Spotify

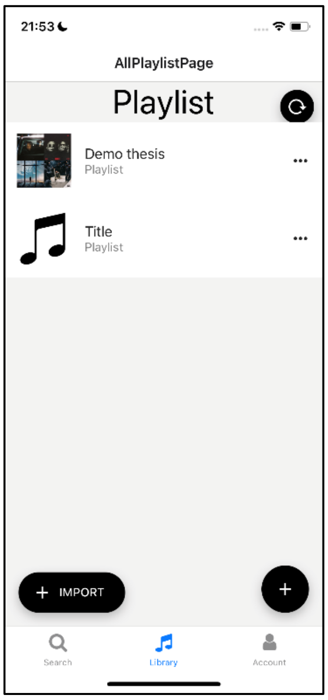
Open in Deezer

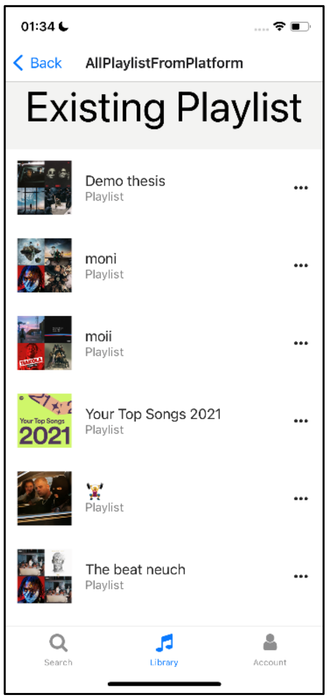
Open in Youtube Music

Share content

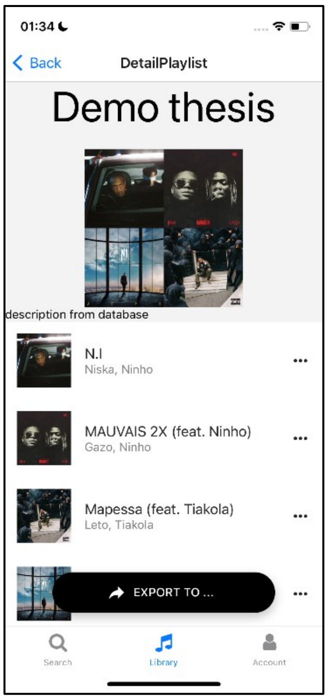
Search Library Account

Appendix 38. Visual of the different playlist display pages

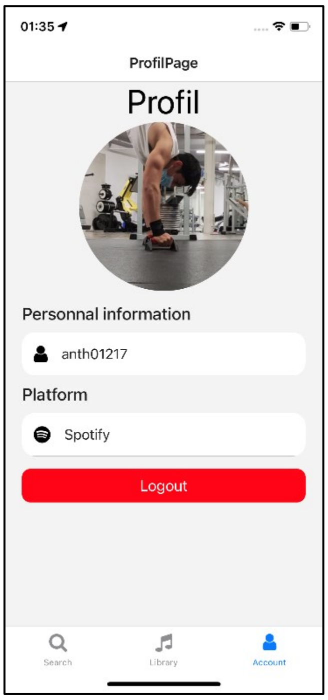




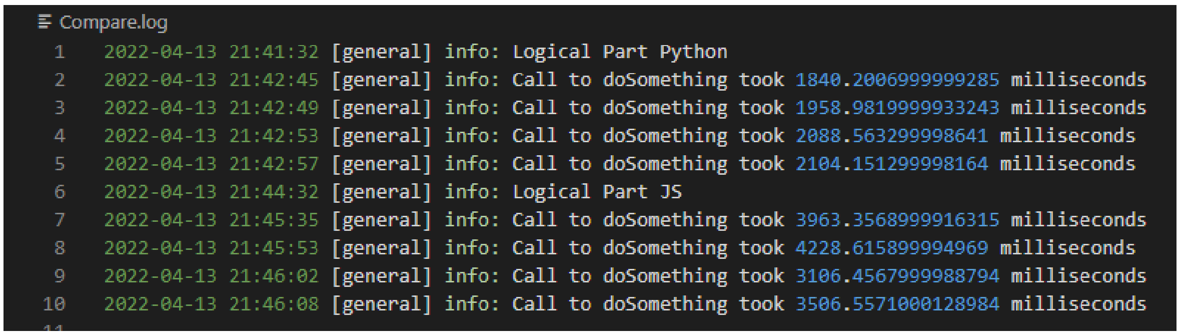
Appendix 39. Visual of a playlist detail page



Appendix 40. Visual of a user's profile page



Appendix 41. Runtime comparison between python and JavaScript


Compare.log

```
2022-04-13 21:41:32 [general] info: Logical Part Python
2022-04-13 21:42:45 [general] info: Call to doSomething took 1840.2006999999285 milliseconds
2022-04-13 21:42:49 [general] info: Call to doSomething took 1958.9819999933243 milliseconds
2022-04-13 21:42:53 [general] info: Call to doSomething took 2088.563299998641 milliseconds
2022-04-13 21:42:57 [general] info: Call to doSomething took 2104.151299998164 milliseconds
2022-04-13 21:44:32 [general] info: Logical Part JS
2022-04-13 21:45:35 [general] info: Call to doSomething took 3963.3568999916315 milliseconds
2022-04-13 21:45:53 [general] info: Call to doSomething took 4228.615899994969 milliseconds
2022-04-13 21:46:02 [general] info: Call to doSomething took 3106.4567999988794 milliseconds
2022-04-13 21:46:08 [general] info: Call to doSomething took 3506.5571000128984 milliseconds
```

Appendix 42. Planning

ask .no	Task	Outcome	Beginning criterion	Hours											
1	**Starting the project**	**Accepted project plan**	**An advisor is assigned to the**	9.0	3/7	3/14	3/21	3/28	4/4	4/11	4/18	4/25	5/2	5/9	
1.1	Write a first version of the project plan	Project plan		5.0	🟥										
1.2	Preparation of the starting meeting	All relevant material is delivered		1.0		🟥									
1.3	Starting meeting	Project has been started	Project plan	1.0	🟥	🟥									
1.4	Writing the minutes	Minutes is delivered	Starting meeting	2.0		🟥									
2	**Functional Analyze - Pull data**	**Overview of the way to pull the data correctly**	**Project has been started**	2.0											
2.1	Define the limites	Limites defined for the project		2.0	🟥										
3	**Functional Analyze - Server**	**Overview of the project Architecture**	**Project has been started**	9.0											
3.1	Define the features	Vision of the functionality		2.0	🟥										400
3.2	Database Diagram	Diagram with description of each columns of each data		4.0	🟥	🟥									
3.3	Do Mock-up	Mock-up		3.0	🟥	🟥									
4	**Functional Analyze - Mobile App**	**Overview of the Mobile App features**	**Project has been started**	9.0											
4.1	Define the features	Vision of the functionality of the mobile app		2.0	🟥										
4.2	Database Diagram	Design database inside the application		4.0	🟥										
4.3	Do Mock-up	Mock-up		3.0	🟥	🟥									
5	**Organic Analyze - Pull data**	**Operational Server with the database**	**Functional Analyze is done**	127.5											
5.1	How work each service	Document that explain how work the service		27		🟥	🟥	🟥							
5.2	How it's the possibility to connect each service			30		🟥	🟥	🟥							
5.3	Programming	Access to each data		34.5		🟥	🟥	🟥							
5.4	Writting part			36		🟥	🟥	🟥							
6	**Organic Analyze - Server**	**Operational Server with the database**	**Functional Analyze is done**	127											
6.1	Develop database	operational DBMS		4					🟥	🟥	🟥				
6.2	REST API	Endpoint accessible		16					🟥	🟥	🟥				
6.3	View Page	Interface usable		12					🟥	🟥	🟥				
6.4	Programming	Logical part to fill the database (search analyse content, ect...)		55					🟥	🟥	🟥				
6.5	Writting part			40					🟥	🟥	🟥				H
7	**Organic Analyze - Mobile App**	**Mobile App that colabore with the server**	**Functional Analyze is done**	60											E
7.1	Navigation page	Application usable		4								🟥	🟥		U
7.2	View Page			24								🟥	🟥		R
7.3	Authentifiction from service	Login from any API Service		8								🟥	🟥		E
7.4	Setup connection with Server	Get data from the server		4								🟥	🟥		S
7.5	Writting part			20								🟥	🟥		
8	**Project management**	**Managed project**	**Project has been started**	56											
8.1	Reporting and follow-up of the progress	Daily Report / Update of planning and backlog		31	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	
8.2	Writing the minutes	Minutes is delivered	Meetings	4		🟥					🟥			🟥	
8.3	Meetings	Ensure about the project		4		🟥					🟥			🟥	
8.4	Create planning	To manage coorectly time during the project		7	🟥	🟥									
8.5	Backlog	Vision of what i need to do for the project in a Excel file		10	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	

Appendix 43. Extract of the backlog

Action (I want to ...)	Results	Priority	Status	Estimate time (included documentati on)	~Time to documentati on	Time to work	Real time	Start date	End date
Product a technical documentation to have the possibility to give the project to another	An documentation	High	Permanent	69.5	x	x	69.5	07 March 2022	
Do a project plan to explain my thesis to my teacher	Project plan	Normal	Done	5	x	5	5	07 March 2022	07 March 2022
Do a planning to have a estimation of the amount of work	Planning for the thesis	High	Done	4	x	4	4	08 March 2022	08 March 2022
Do a template to describe my daily report	A Template for the daily report	Normal	Done	2	x	2	2	09 March 2022	09 March 2022
Graphical Architecture of the technology	A diagram	Normal	To do	2	0.5	1.5			
fill my planning and adjust correctly during all the time of the thesis	Maintened planning	High	Permanent	12	x	12	12	14 March 2022	
Do my daily planning at the begin and the end of the works days	Daily planning fill	High	Permanent	19	x	19	19	11 March 2022	