MINI PROJECT REPORT

**Apollo: The Music Porting App**

*Submitted in partial fulfillment of the requirements for the degree of*

**Bachelor of Engineering**

in

**Computer Science and Business System**

*Submitted By:*

**Ayush Anand - 101918060**

**Nishtha Wakankar - 101918061**

**Neelaksh Tayal - 101918069**

**Akshat Bhatia - 101918076**

Under the supervision of:

**Dr. Manju Khurana**

Assistant Professor



**COMPUTER SCIENCE AND ENGINEERING DEPARTMENT**

**THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**PATIALA – 147004**

**December 2021**

# Acknowledgement

We would like to express our thanks to our mentor. She has been of great help in our venture, and an indispensable resource of technical knowledge. She is truly an amazing mentor to have. We are also thankful to, Head, Computer Science and Engineering Department, entire faculty and staff of Computer Science and Engineering Department, and also our friends who devoted their valuable time and helped us in all possible ways towards successful completion of this project. We thank all those who have contributed either directly or indirectly towards this project.

**Date: 19 - 12 - 2021**

|  |  |  |
| --- | --- | --- |
| **Roll No.** | **Name** | **Signature** |
| 101918060 | Ayush Anand |  |
| 101918061 | Nishtha Wakankar |  |
| 101918069 | Neelaksh Tayal |  |
| 101918074 | Akshat Bhatia |  |

Counter Signed by:



Dr. Manju Khurana

(Faculty Mentor)

# Table of Contents

[Chapter 1 - Introduction 1](#_Toc90923120)

[1.1 Overview 1](#_Toc90923121)

[1.2 Project Scope 2](#_Toc90923122)

[Chapter 2 - Background 3](#_Toc90923123)

[2.1 Motivation 3](#_Toc90923124)

[2.2 Project Layout 3](#_Toc90923125)

[2.3 Constraints 4](#_Toc90923126)

[Chapter 3 - Objectives 5](#_Toc90923127)

[3.1 Fundamental Objectives 5](#_Toc90923128)

[3.1.1 Music Snippet Playing 5](#_Toc90923129)

[3.1.2 Playlist Porting to an All Accessible Platform 5](#_Toc90923130)

[3.1.3 Playlist Creation on the Same Platform 5](#_Toc90923131)

[3.1.4 Seeking, Volume Toggle and Basic Music Player Application Features 5](#_Toc90923132)

[3.2 Additional Objectives 5](#_Toc90923133)

[3.2.1 Music Taste Analysis 6](#_Toc90923134)

[3.2.2 Multi-platform Porting and Creation 6](#_Toc90923135)

[3.2.3 Recommended Playlist Creation 6](#_Toc90923136)

[Chapter 4 - Methodology 7](#_Toc90923137)

[4.1 Preliminary Research 7](#_Toc90923138)

[4.2 Application Programming Interfaces (API) 8](#_Toc90923139)

[4.2.1 Music API 8](#_Toc90923140)

[4.2.2 Spotify API 10](#_Toc90923141)

[4.2.3 YouTube API 14](#_Toc90923142)

[4.3 Application Environment 18](#_Toc90923143)

[4.3.1 Front End 18](#_Toc90923144)

[4.3.2 Back End 20](#_Toc90923145)

[4.3 Application 21](#_Toc90923146)

[4.3.1 Home 21](#_Toc90923147)

[4.3.2 Songs 23](#_Toc90923148)

[4.3.3 Recently Played 23](#_Toc90923149)

[4.3.3 Footer 24](#_Toc90923150)

[4.3.4 Albums 24](#_Toc90923151)

[4.3.5 Artists 25](#_Toc90923152)

[4.3.6 Playlists 26](#_Toc90923153)

[4.3.7 Porting 26](#_Toc90923154)

[Chapter 5 - Challenges and Limitations 28](#_Toc90923155)

[5.1 Challenges 28](#_Toc90923156)

[5.1.1 Error Status 403: Forbidden 28](#_Toc90923157)

[5.1.2 Error Status 400: Invalid Body Snippet 28](#_Toc90923158)

[5.1.3 Asynchronous Code Error 28](#_Toc90923159)

[5.1.4 Spotify API Scope Error 28](#_Toc90923160)

[5.2 Limitations 29](#_Toc90923161)

[5.2.1 Error Status 403: Forbidden 29](#_Toc90923162)

[5.2.2 Porting of Multiple Playlists 29](#_Toc90923163)

[5.2.3 Stream Duration 29](#_Toc90923164)

[Chapter 6 - Future Scope 30](#_Toc90923165)

[Chapter 7 - References 31](#_Toc90923166)

[Chapter 8- Plagiarism Report 32](#_Toc90923167)

# 

# List of Figures

|  |  |  |
| --- | --- | --- |
| **Figure No** | **Title** | **Page No** |
| 4.1 | Market share of popular music streaming apps | 7 |
| 4.2 | Music App Usage Among Peers | 8 |
| 4.3 | Spotify API Authorization | 10 |
| 4.4 | Spotify API Dashboard | 11 |
| 4.5 | Spotify API Documentation | 11 |
| 4.6 | Spotify Web API | 12 |
| 4.7 | Handling Playlists in Spotify | 14 |
| 4.8 | YouTube Data API | 14 |
| 4.9 | YouTube Data Documentation | 15 |
| 4.10 | Google OAuth Consent Screen | 16 |
| 4.11 | Creating a Playlist in YouTube | 17 |
| 4.12 | Inserting a song in a playlist | 18 |
| 4.13 | Structure of our Frontend | 19 |
| 4.14 | Difference between Prop-Drilling and React-Redux | 20 |
| 4.15 | Structure of Backend | 21 |
| 4.16 | Home Page of our Application (Explore) | 22 |
| 4.17 | Home Page of our Application (Featured) | 22 |
| 4.18 | Song List | 24 |
| 4.19 | Recently Played Songs | 23 |
| 4.20 | Footer of our Application | 23 |
| 4.21 | Albums | 24 |
| 4.22 | Artists | 24 |
| 4.23 | Artist Playlist | 25 |
| 4.24 | Our Playlist | 25 |
| 4.25 | Playlist created in YouTube | 26 |
| 4.26 | Transferring songs to YouTube Playlist | 27 |

# List of Tables

|  |  |  |
| --- | --- | --- |
| **Figure No** | **Title** | **Page No** |
| 2.1 | Constraints of the Proposed System | 4 |

# Chapter 1 - Introduction

## 1.1 Overview

There were times when the lone source of music was CD and cassette collections, and in order to listen to something new, people were required to exchange the CDs or buy new ones altogether. It was followed by an era of mp3 players such as Apple iPod and Sony Walkman which gave birth to portable music. Today on the other hand, thanks to music streaming apps such as Spotify and Apple Music, people have access to millions of tracks by almost any artist in the world through the internet. These applications have also set new standards for music streaming services – they should be fast, responsive, consume fewer resources, and deliver tracks to users in a matter of clicks.

Each existing application has its own benefits and shortcomings, and with the rapid development of new music streaming applications, the end user is spoiled for choice. They may prefer the wide variety of artists found on one platform while the freemium, ad free model of another platform may also sound inviting. In the end, whatever may be the choice of application, users often find themselves migrating from one platform to the other or using multiple platforms at the same time in order to reap the most benefits.

However, due to this, the users face a huge problem: They cannot maintain their collection of songs, artists, playlists etc across the various platforms that they use. In order to do so, they would have to go through the hassle of exhaustive searching for each track, and the repetitive process of creating playlists.

The concept of our application came into existence with the idea of providing users with a means of maintaining their digital music collection across various platforms through an easy to use and automated process which saves them time. They would not be required to manually perform the tasks involved in transferring their data from platform to platform, moreover they would be able to do so through a seamless and interactive experience.

## 1.2 Project Scope

There often comes a time when people are looking to switch music providers. Whether it's because they have discovered one has better and more desirable features or that they want to take advantage of a promotional offer. In such a situation the biggest issue they face is that they can’t take their pre-existing playlists with them and it’s a tedious task to create new playlists all over again. Our aim is to solve this problem through our application.

Our project, Apollo, is a playlist porting application that allows the user to transfer their playlists between two of the most popular music streaming platforms: from Spotify to Youtube. Our application is specially developed to help users fetch and transfer music playlists while providing them with an interactive and personalized experience since it not only allows them to port playlists but also lets the user play short snippets of their favorite songs.

# Chapter 2 - Background

## 2.1 Motivation

The light bulb idea we came up with was simple, it originated from a real life experience that our group as a whole experienced. Our motivation to create this project was not just to make something for our college project, but to solve our daily life problem of sharing music. It was the drive of solving the pain point of our team and our friends that drove us to make APOLLO a successful project.

As discussed, our project idea stems from the trouble of using multiple music streaming services or switching from one to the other. Even though users want to use different applications to reap their respective benefits, they avoid doing so because of the same reasons people do not switch from Android devices to Apple devices and vice versa: Exhausting process and potential data loss. The application aims to resolve this very conflict which users face.

After conceiving the idea of our application, we conducted a thorough research to find out the most popular music streaming platforms, the details of which have been discussed further in the report. We selected two of the biggest streaming services, Spotify and Youtube, mainly because of the following factors:- Popularity, Usage, Developer’s tools, Application Programming Interface, Costs, Ease of Use and Documentation of API.

## 2.2 Project Layout

What we are building is a cloud integrated music streaming platform with a streaming, seeking, listening and most importantly playlist porting feature, that lets the user share their music, anytime, anywhere. It is a seamlessly working interface on the web that constantly interacts with multiple streaming platforms Application Programming Interface, Spotify and YouTube as per the current scope of our project, in order to create a bridge among all platforms and enable the user to listen and share their music easily and quickly.

## 2.3 Constraints

The following table enlists the constraints of the proposed system.

|  |  |
| --- | --- |
| **ID** | **Constraints** |
| 1 | Must have a valid music platform account for Spotify. |
| 2 | Must have a valid google account to log into YouTube. |
| 3 | Stable internet connection is a must. |
| 4 | YouTube account must not be private |
| 5 | The application requires user to give permission to access their account data |
| 6 | User must have a playlist created prior to porting |

##### Table 2.1: Constraints of the Proposed System

# Chapter 3 - Objectives

## 3.1 Fundamental Objectives

These objectives are the main features of the application which it must include.

### 3.1.1 Music Snippet Playing

To provide the user with 30 second snippets of their favorite songs playing on the application to get a taste of the vibe of the playlist and decide if they want to share the experience with their friends or not.

### 3.1.2 Playlist Porting to an All Accessible Platform

To allow porting of a playlist’s songs into a platform which is most commonly used by everyone. Through our research, we found out that YouTube among all platforms is the most used, and reliable.

### 3.1.3 Playlist Creation on the Same Platform

To use the songs ported, we wanted to create a playlist with the same name from where it has been ported.

### 3.1.4 Seeking, Volume Toggle and Basic Music Player Application Features

To provide the custom features was not enough to attract users for this application. We need to provide a 360 degree experience. Providing all features of a simple music application, along with the custom features of APOLLO.

## 3.2 Additional Objectives

These are additional and more complex features which go beyond the current scope of the project, but are envisioned to be part of the future versions of the application.

### 3.2.1 Music Taste Analysis

Using machine learning, we want to analyze the music taste and recommend, genres, songs and increase the overall user experience

### 3.2.2 Multi-platform Porting and Creation

Not just limiting porting from Spotify to YouTube, but porting and creating playlists, to and from multiple streaming platforms for convenience and according to the scale of the application.

### 3.2.3 Recommended Playlist Creation

Creating recommended playlists automatically by accessing multiple platforms, understanding users' taste and creating a recommended playlist on multiple platforms at the same time every month.

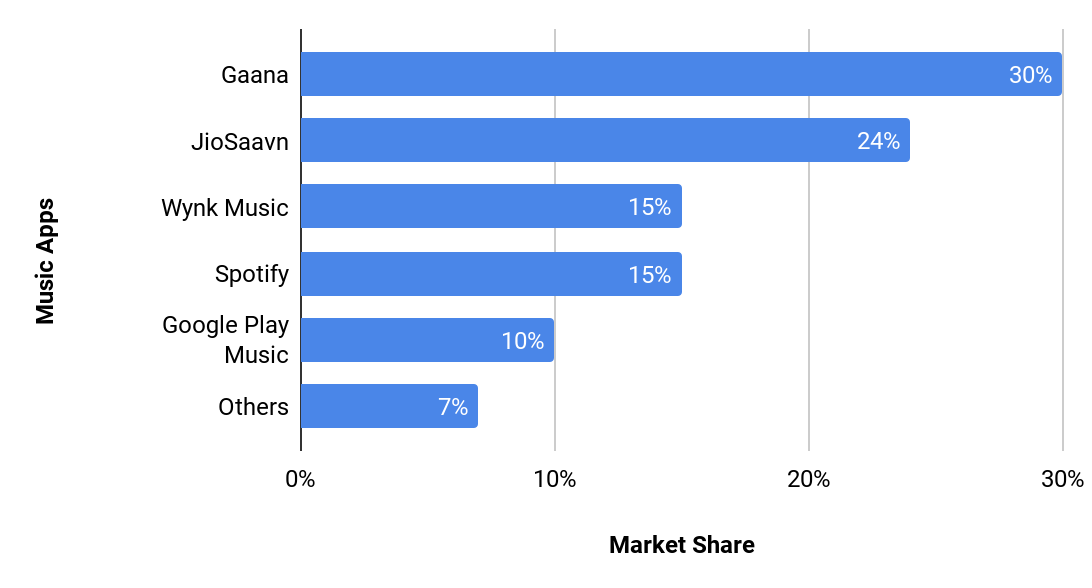
# Chapter 4 - Methodology

## 4.1 Preliminary Research

The sources of listening to music have undergone a metamorphosis by traveling through various periods i.e. listening to our favorite songs on FM radio to on-demand music streaming services. Today music streaming services have shown a great impact on users due to various features that they are offering. The music streaming sector has significantly grown in 2020 with a reported increase of 40% in usage (in India).Music Streaming Services Market Overview in India:

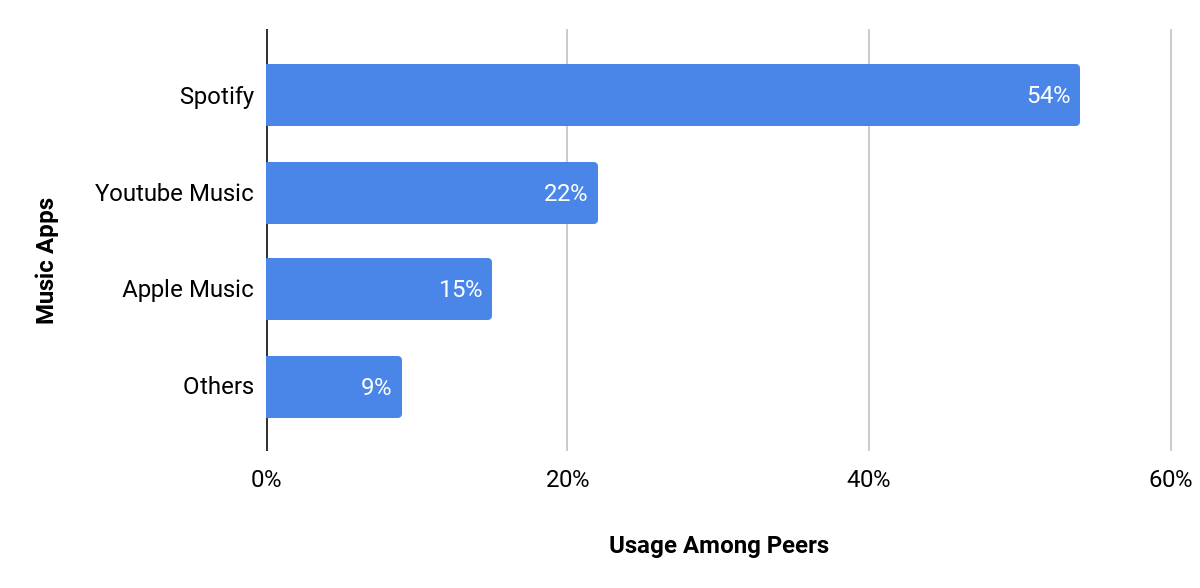
1. Revenue in the Music Streaming segment amounts to US$371m in 2020.
2. Revenue is expected to show an annual growth rate (CAGR 2020–2024) of 7.8%, resulting in a market volume of US$500m by 2024.

To start off the project, the first step was to check out the various music apps like Gaana, Spotify, Apple Music, etc. that are popular among the users. We decided to look at the music apps’ market share of the year 2020. This statistic showed that among the listeners in India, Gaana is the most popular application with 30% of the market share, followed by JioSaavn and Wynk Music with 24% and 15% market share respectively.



##### Figure 4.1: Market share of popular music streaming apps

We also asked around for the most commonly used music apps amongst our peers and found out that Spotify, Youtube Music and Apple Music were being used by the majority of the people with Spotify taking up more than 60% of the user base since its launch. This lead to the conclusion that we should focus our application on these three apps primarily.



##### Figure 4.2: Music App Usage Among Peers

## 4.2 Application Programming Interfaces (API)

API is a type of software interface, which provides service to other pieces of software. The term API may refer to the specification or application and assist in connecting computers or software pieces to each other. The main purpose of APIs is to hide the internal details of how the system works, to display only those parts that the editor will find useful and keep them unchanged even if the internal details change later. An API can be customized for a particular system pair, or it can be a shared standard that allows interaction between multiple systems.

### 4.2.1 Music API

Who doesn’t love music? Some of the most widely downloaded and used music apps are Spotify, Youtube and iTunes. Other music apps like Shazam have the amazing capability to recognize songs just by hearing them. So wouldn’t it be great to integrate these amazing features into one app or use the functionalities of these apps and make something better out of it?

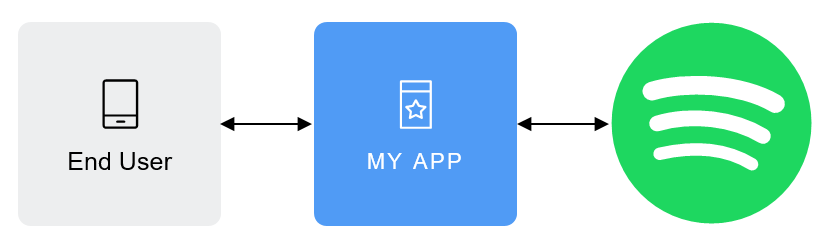
Music API is a command template for music applications that can be used as part of its overall design. It may contain a large library of songs and other audio files that a third party application can access for live streaming purposes. The API can also be designed to help build playlists or find names, album cover art, chart info, or other song metadata. Unlimited opportunities are open to developers using audio APIs. Song APIs contain a pre-programmed set of multi-code commands needed to create a complete music app. They can simplify the work that engineers have to do to create applications for streaming music, acquiring songs and song information, and so on..

There are a lot of music data APIs that an engineer can access and make their job easier. We have researched and reviewed the various music platforms and their APIs to plan our options and know which one will work best for our needs and applications. Some of the popular API titles we looked at were Spotify, iTunes, Youtube, and Deezer. APIs for some popular apps like Saavan, Wynk and Gaana do not exist so we had to remove those options. By doing the most comprehensive research on available APIs, we found that the Spotify Web API is best for managing music metadata and playlists. The Apple Music API was paid for so we removed that option and continued with the YouTube API as YouTube is used by a large number of people and its API is also well organized.

### 4.2.2 Spotify API

**Authorization:**

We are granting a user or application access permissions to Spotify data and features for which spotify implements the OAuth 2.0 authorization framework.

****

##### Figure 4.3: Spotify API Authorization

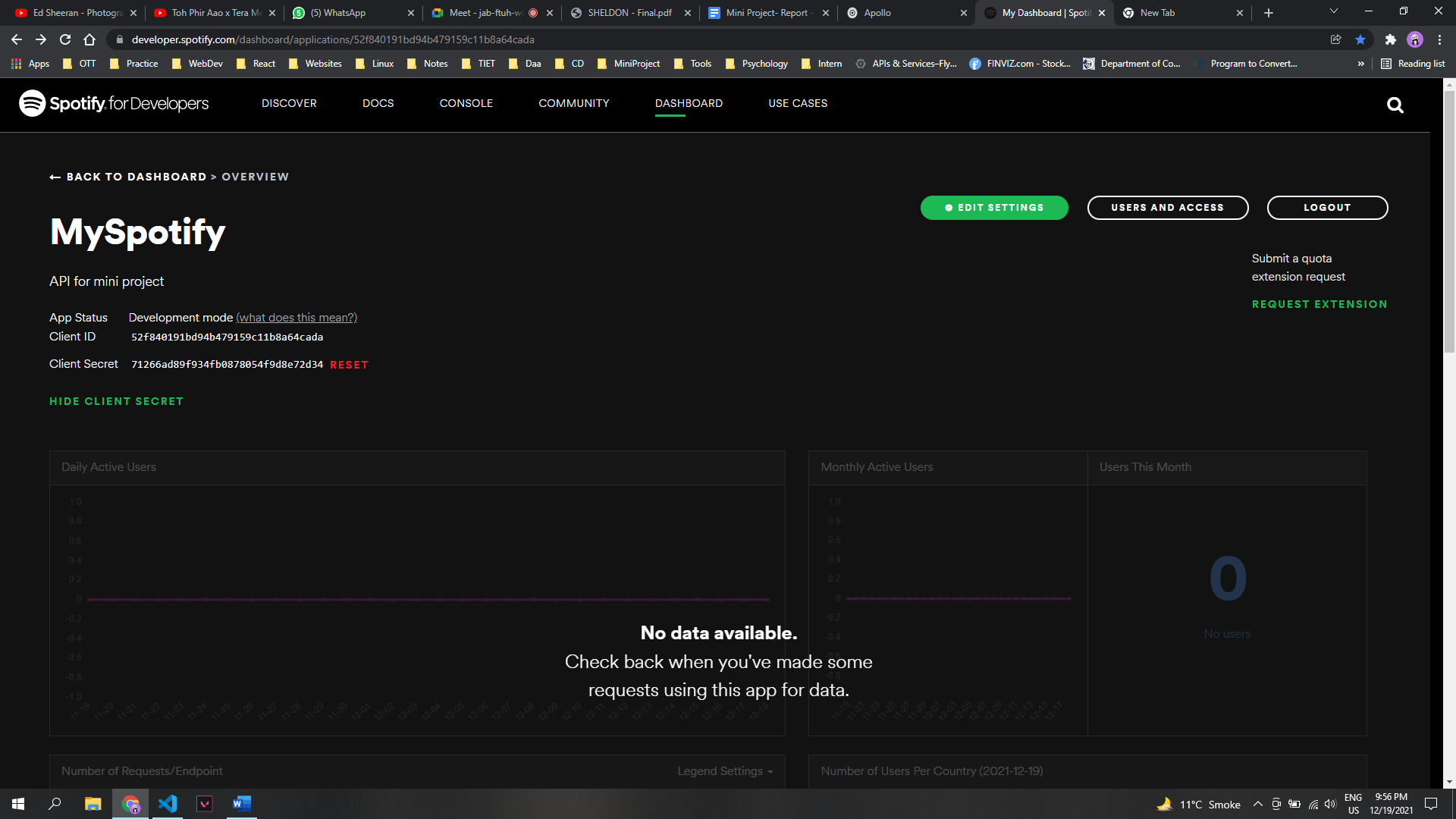
Here,

1. End User corresponds to the Spotify user. The End User will then grant access to the protected resources (e.g. playlists, user information etc)
2. My App is the client that requests access to the protected resources (e.g. a mobile or web app).
3. The server hosts the protected resources and provides authentication and authorization via OAuth 2.0.

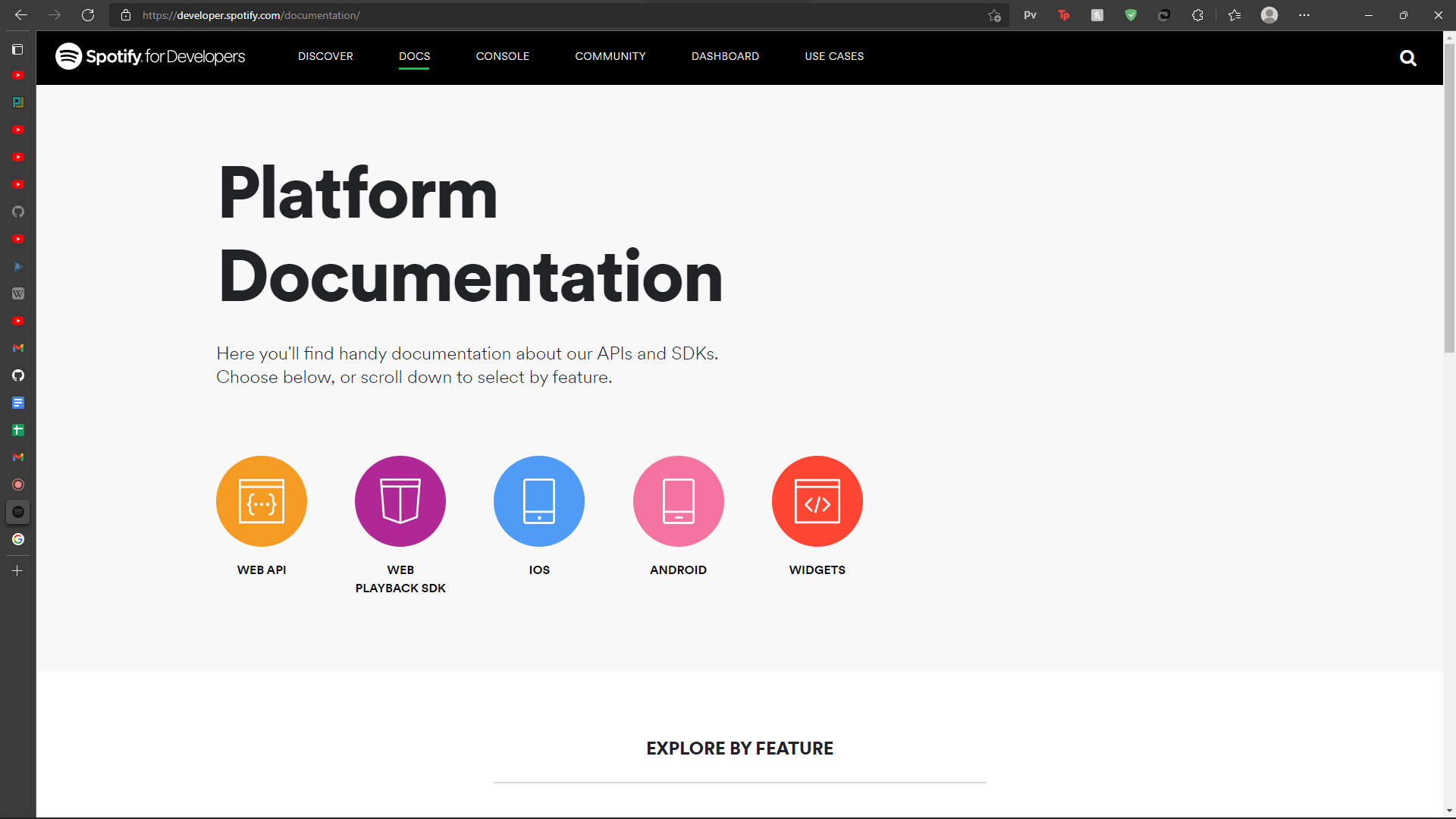
The access to the protected resources is determined by one or several scopes. They allow the app to access a specific feature such as reading a playlist, editing a library or streaming live.The set of scopes that are set during the authorization, would determine the access permissions that the user is asked to grant.

The scopes used by us for our web application are:-

Playlist-read-private, playlist-read-collaborative, playlist-modify-public, user-read-recently-played, playlist-modify-private, ugc-image-upload, user-follow-modify, user-follow-read, user-library-read, user-library-modify, user-read-private, user-read-email, user-top-read, user-read-playback-state.

The authorization process requires valid client information: Client ID and client privacy. Once authorization has been granted, the authorization server issues an access token, which is used to make API calls on behalf of a user or application.

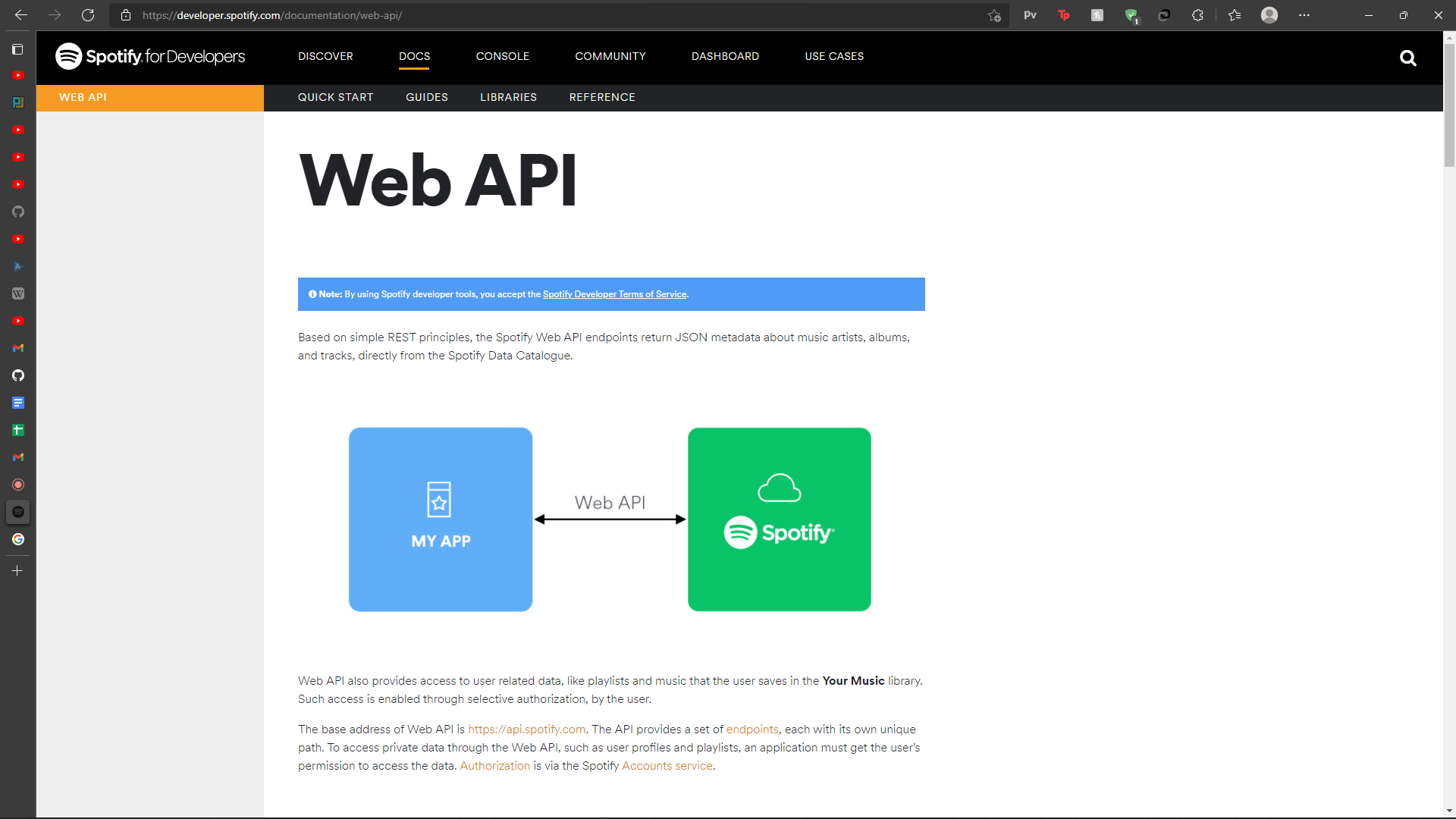
##### Figure 4.4: Spotify API dashboard



##### Figure 4.5: Spotify API Documentation

Spotify offers a very thorough and fleshed out documentation of their platform and the various functionalities the API offers to emulate and use those features for the development of another application or website.

The documentation is divided into different sections for different platforms and aspects. For example, the documentation on how to use the Spotify API for a web-based application is separated from the other sections to allow easy access to the functions and features that a developer may require.

****

##### Figure 4.6: Spotify Web API

With Spotify APIs and SDKs for JavaScript, iOS, and Android, one can develop unique experiences in as little as a few lines of code.

1. With the Spotify Developer Platform, the developer is able to read the calculated audio features of tracks to learn about their danceability, energy, and more. For advanced cases, it is possible to read the in-depth analysis data about tracks such as the bars, beats, pitches, etc.

* Mood: Danceability, Energy, Tempo
* Properties: Loudness, Speechiness, Instrumentalness
* Context: Liveness, Acousticness
* Segments, Tatums, Bars, Beats, Pitches, Timbre, and more.

1. The API offers multiple, fully featured playback options depending on the requirement.

* Ability to play music directly in the web browser, with the Web Playback SDK.
* Usage of Spotify Connect to control and transfer playback between any of a user’s active devices.
* Embedding of play widgets with Spotify’s look and feel.

1. The developer is able to easily search and get information about any artist, album, track, or playlist on Spotify based on a search query.

* Search based on the user's market.
* Search based on any market.
* Search up to 10,000 results.

1. With specific controls such as market, seeds (artists, genres, tracks), ranged audio features (danceability, tempo, liveness, etc) and popularity, one can generate very specific recommendations based on Spotify’s algorithms.
2. The developer can access artists, albums, or songs in the user’s local market or a specific Spotify market.

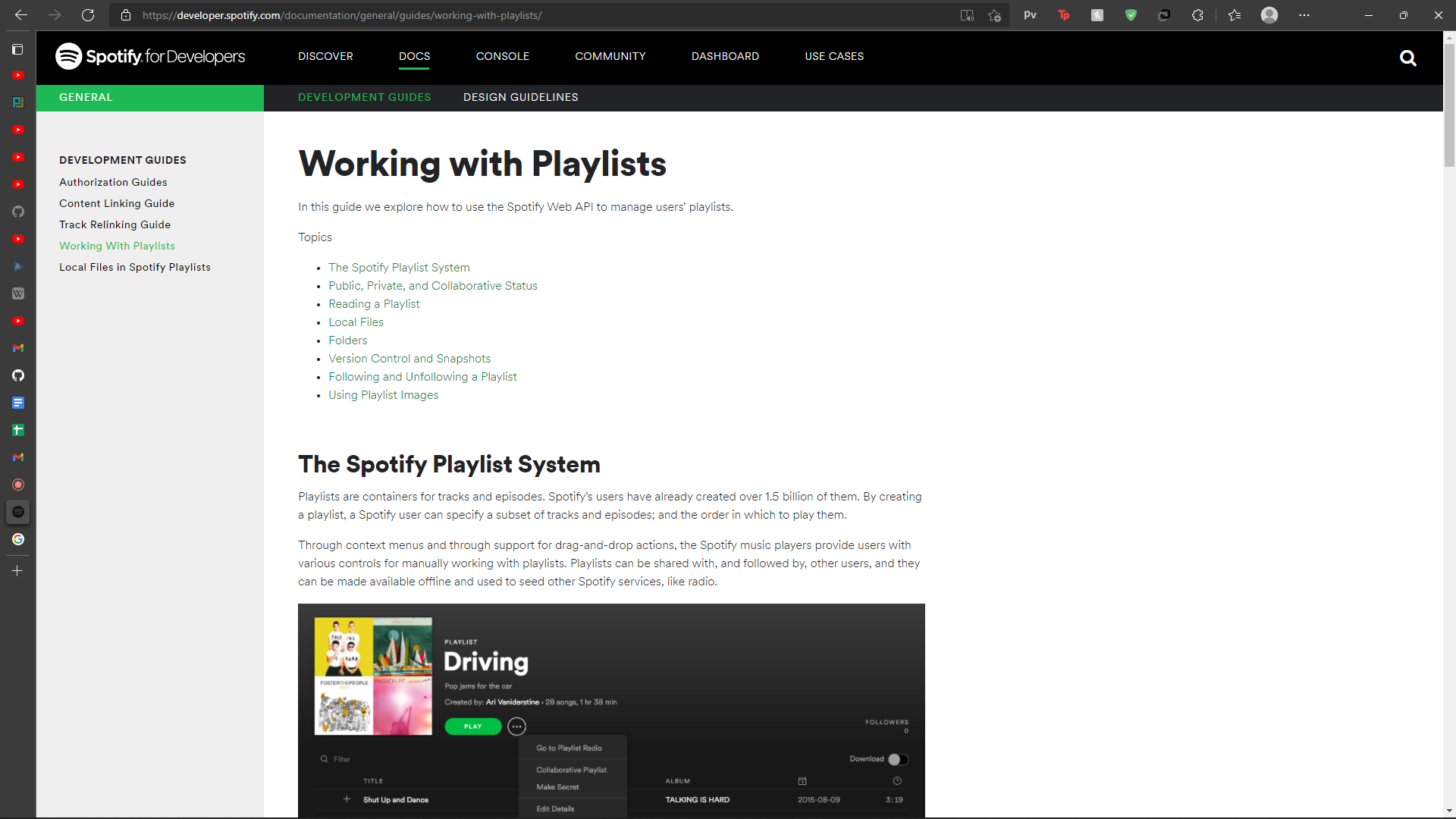
* Read metadata information about tracks, artists, or albums.
* Read algorithmically related artists to a particular artist.
* Play a 30 second snippet of available tracks.
* Receive responses based on the user's market, or a specific market.

1. The API also allows the developer to read featured new releases and curated playlists created by Spotify’s editorial team, based on popularity, mood, international events, and genres.

* List of Featured Playlists.
* List of Featured New Releases.
* Browse Categories and their Playlists.

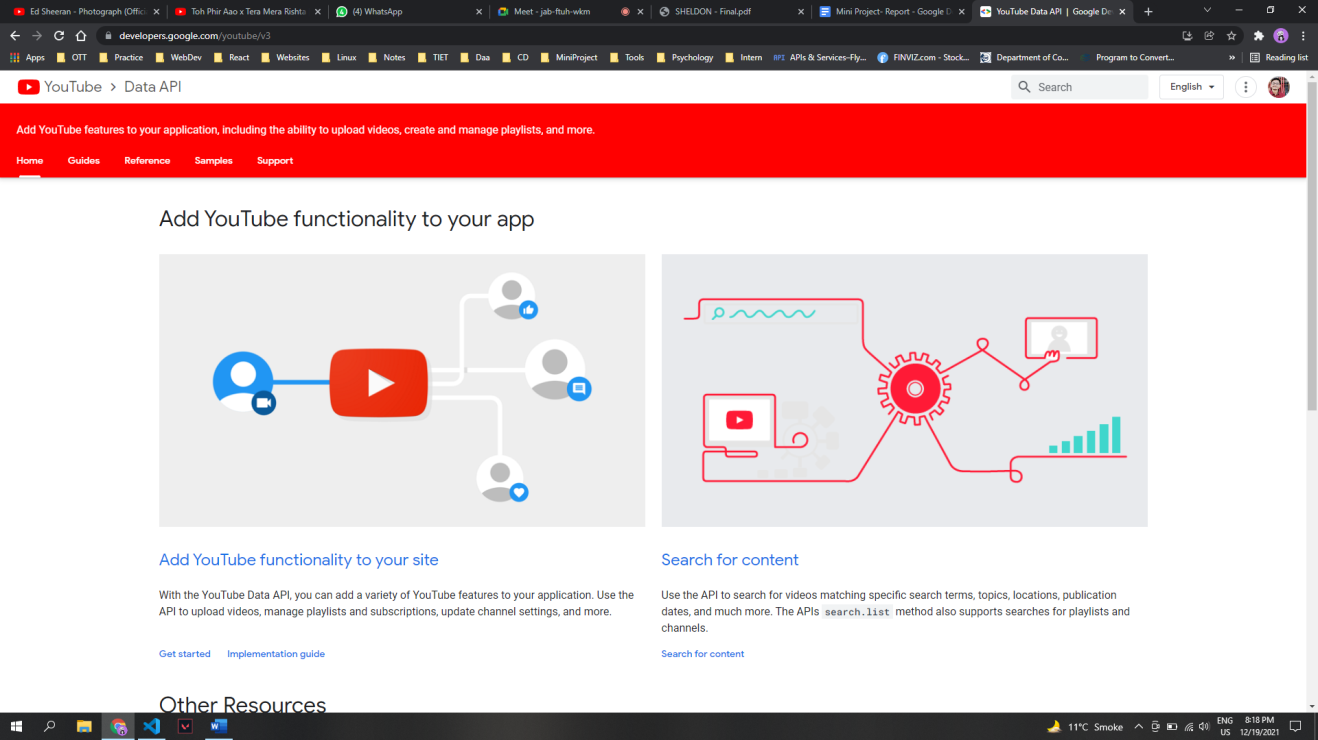
1. It is also possible to fetch a user’s listening activity such as recently played music and most listened to tracks and artists. The users can also be allowed to save or unsave tracks and albums, as well as follow and unfollow artists.

* Read data for a user’s top tracks and artists.
* Read data for a user’s recently played tracks (up to 50).
* Save and unsave tracks and albums.

****

##### Figure 4.7: Handling playlists in Spotify

### 4.2.3 YouTube API

The YouTube Application Programming Interface (YouTube API) allows developers to access statistics and data for YouTube channels. The YouTube API is a tool that allows us to bring YouTube activity to our web page, application or device.

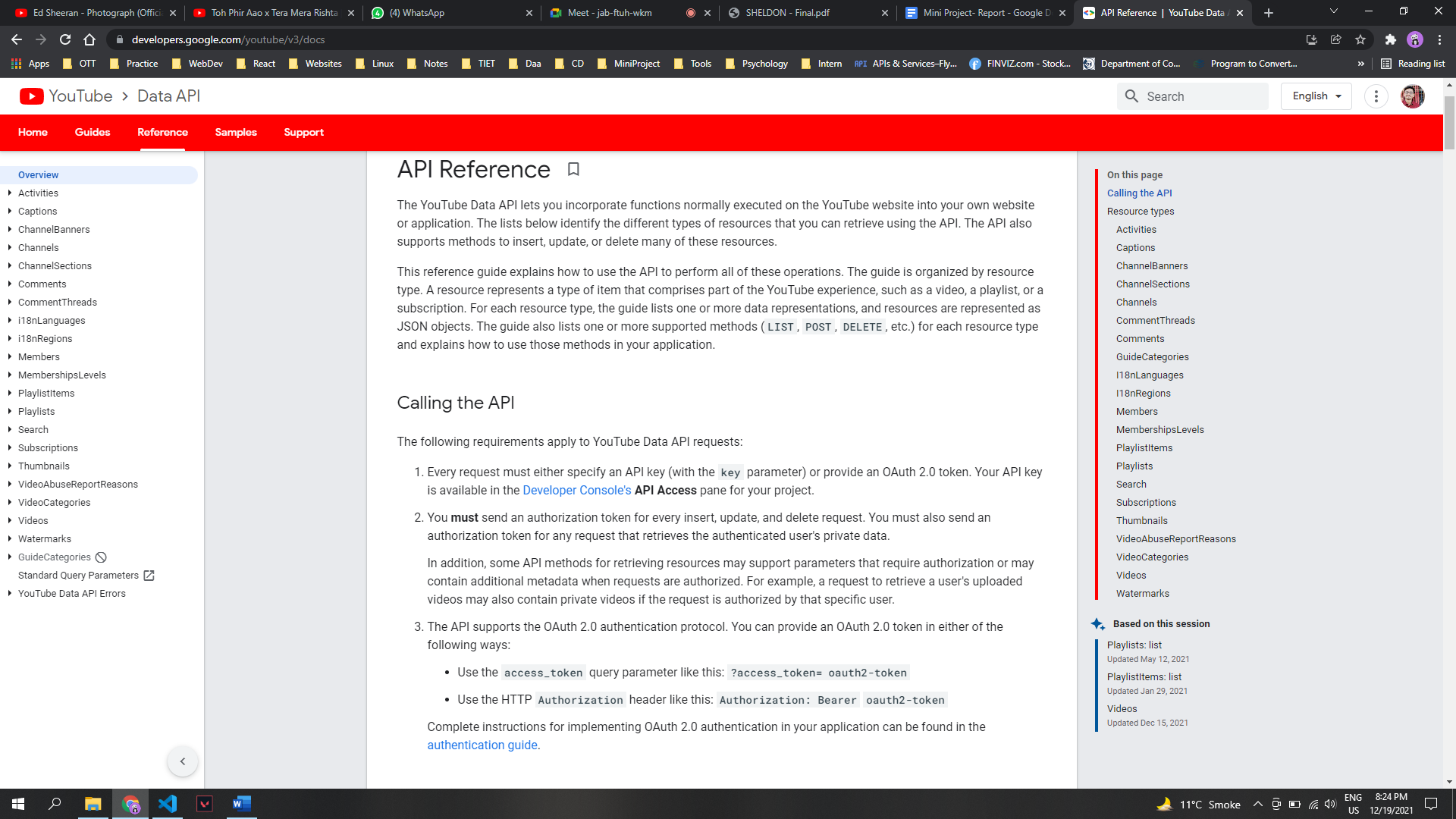
##### Figure 4.8: YouTube Data API

The YouTube Data API lets us incorporate functions normally executed on the YouTube website into our own website or application. It also supports methods to insert, update, or delete many of these resources as per our requirement.

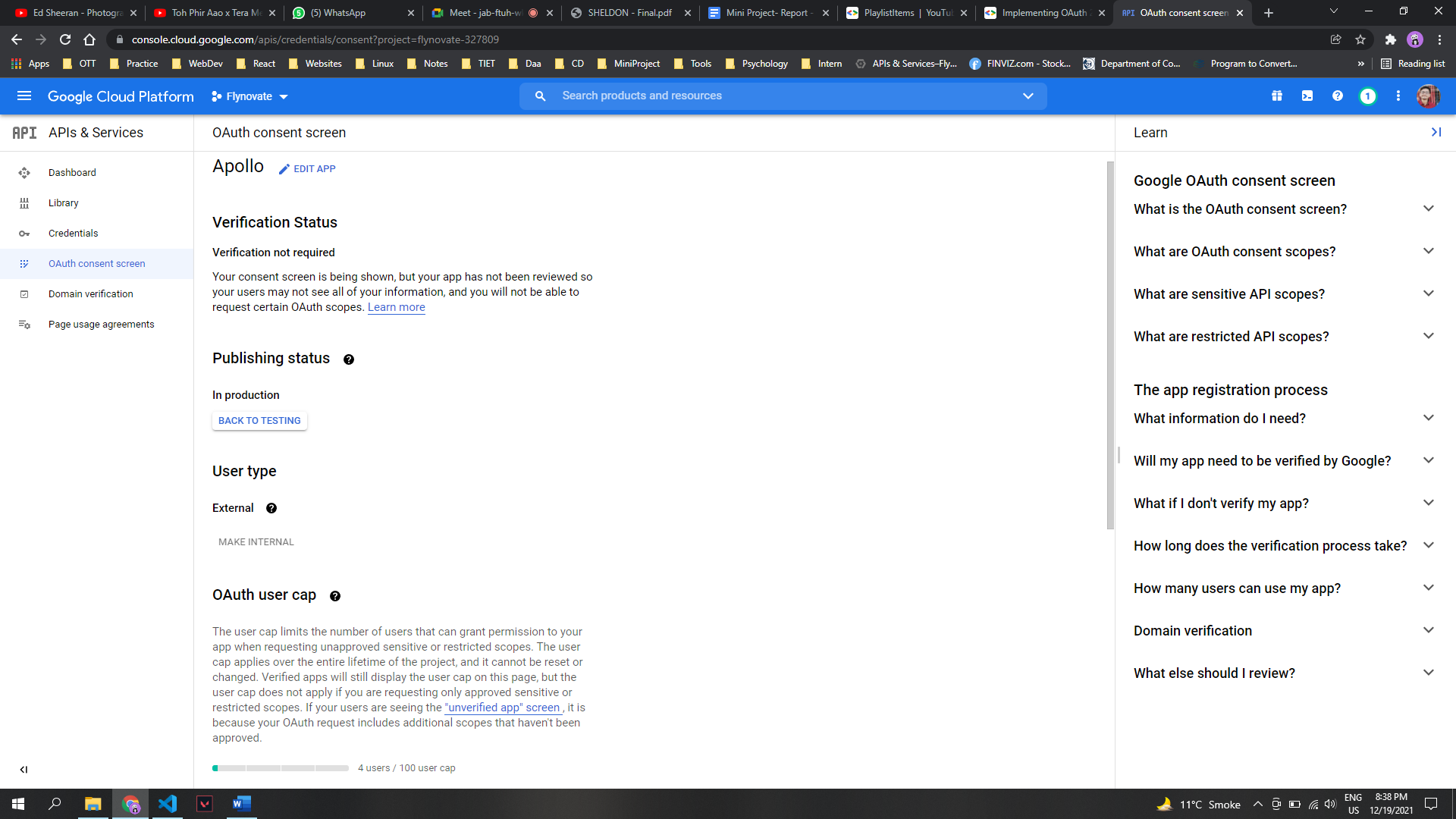
**Authentication:**

The YouTube Data API supports the OAuth 2.0 protocol for authorizing access to private user data. Some core OAuth 2.0 concepts:

1. When we first attempt to use functionality in our application that requires the user to be logged in to a Google Account or YouTube account, our application initiates the OAuth 2.0 authorization process.
2. Our app directs the user to Google's authorization server. The link to that page specifies the scope of access that our application is requesting for the user's account.
3. If the user consents to authorize our application to access those resources, Google returns a token to our application and depending on the application's type, it either validates the token or exchanges it for a different type of token.



##### Figure 4.9: YouTube API Documentation



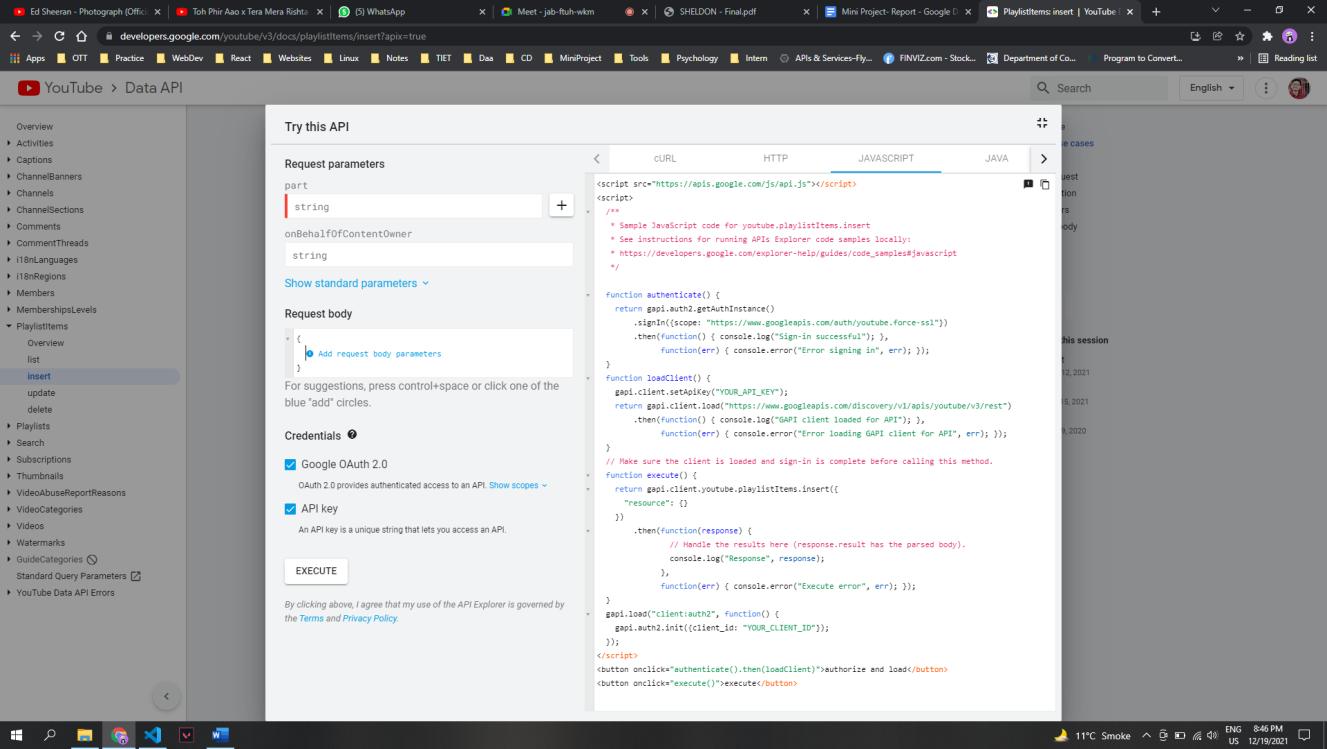
##### Table 4.10: Google OAuth Consent Screen

**Playlist Actions:**

A playlistItem resource which is a part of Youtube API that identifies another resource, such as a video, that is included in a playlist. In addition, the playlistItem resource contains details about the included resource that pertain specifically to how that resource is used in that playlist. We can retrieve the playlist ID for the list from the channel resource for a given channel.

**Creating a Playlist:**

We call the playlists.insert method to create a private playlist in the authenticated user's channel. Any API request to this method must be authorized using OAuth 2.0. The request body contains a playlist resource, which defines information like the playlist's title, description, and privacy status.

****

##### Figure 4.11: Creating a playlist in YouTube

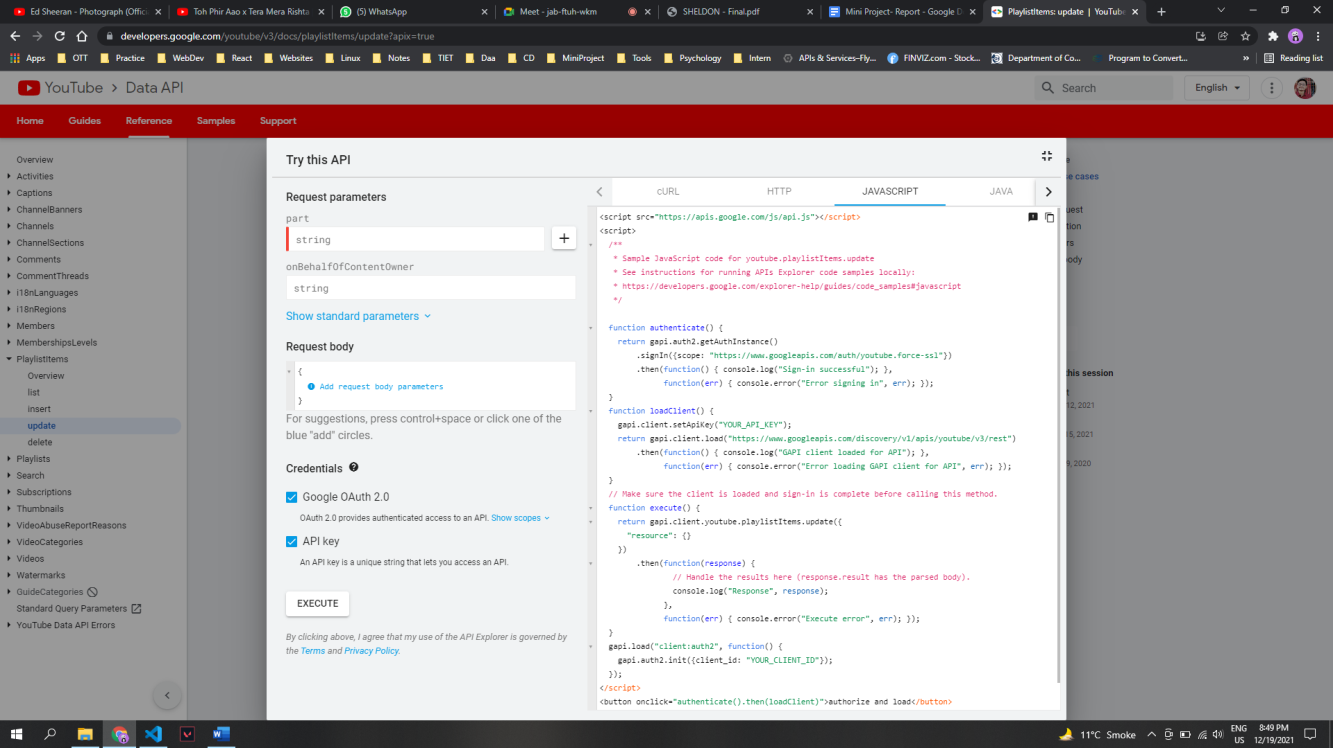
**Inserting in a playlist:**

We follow following steps to insert a playlist in our application:

1. Retrieve the appropriate playlist ID: We call the playlist.list method to retrieve the playlists in the currently authenticated user's channel. The application calling the API could process the API response to display a list of playlists, using each playlist's ID as a key.
2. Add a video to the playlist: We call the playlistItems.insert method to add a video to the playlist. This request must be authorized using OAuth 2.0.

The request body is a playlistItem resource that specifies at least the following values:

1. The snippet.playlistId identifies the playlist to which you are adding the video. This is the playlist ID we obtained in step 1.
2. The snippet.resourceId.kind contains the value “youtube#video”.
3. The snippet.resourceId.videoId identifies the video that you are adding to the playlist. The property value is a unique YouTube video ID.

****

##### Table 4.12: Inserting a song in a playlist

## 4.3 Application Environment

This section discusses the languages and packages used along with the overall environment for the development of the application. The code editor and IDE used for the development of the application was Visual Studio Code.

### 4.3.1 Front End

The Front End of the application is built using React JS which is an open-source front-end JavaScript library which is used for building User Interfaces based on different components. Any application built using React JS is a single page application which is concerned with state management and rendering that state through the Document Object Model or DOM. We are using JSX as a programming language which stands for JavaScript XML. It makes it easier to write and add HTML in React. The designing of the UI has been done using CSS.

So our react app starts with running an index.js file which supports rendering of our entire application. It contains a store which is an object that holds the complete state of our app. Next runs our app.js file which supports rendering of all the components and defines the basic layout of our applications. Components let us split the UI into independent, reusable pieces and work around each piece in isolation. To render a component in React we initialize an element with a user-defined component and pass this element as the first parameter to ReactDOM.render( ) method.

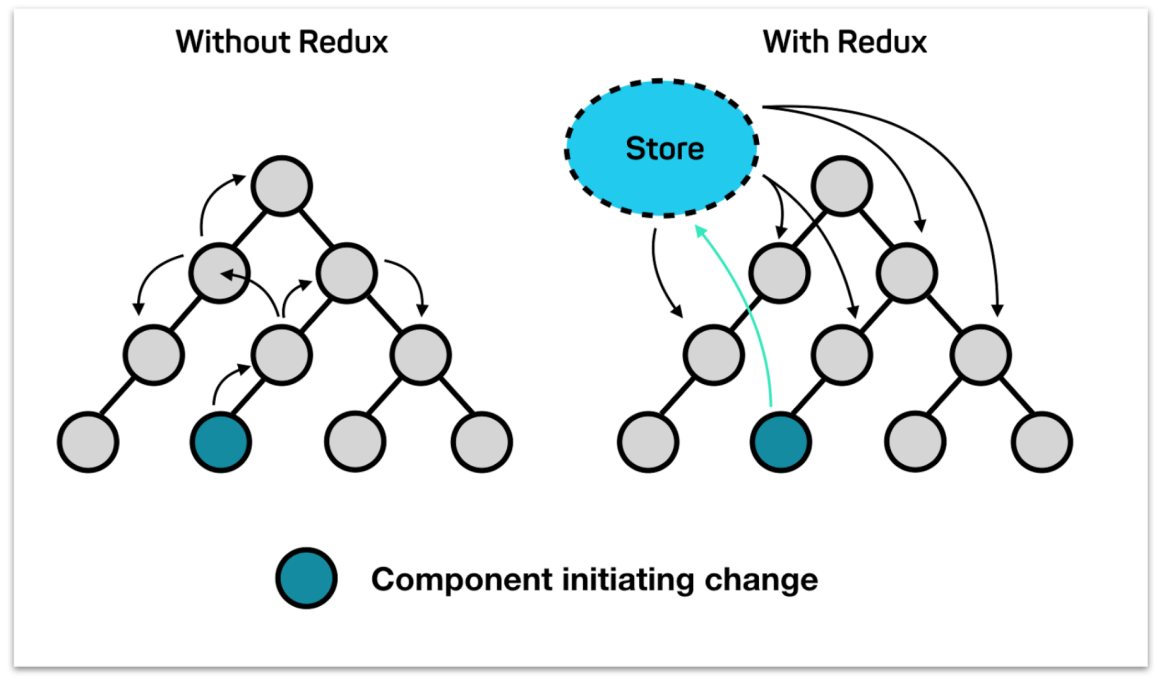


##### Figure 4.13: Structure of our Frontend

Since our application requires building a lot of components, storing too many states and fetching a large amount of data, we are using React-Redux which is the best state management tool in our case. Generally in React, we use props (properties) in a component which allows the usage of non-static variables with the help of which we can pass these variables into various other components (child components) from the parent component. With a few components in our application, we can pass these states as props to the child components. But as the count of components increases in the application in accordance with its large objective, we need to pass these states to other components located far away from each other in the component tree which sometimes makes things messy. This brings the need for react-redux in our application. React-redux makes it easier to pass these states from one component to another irrespective of their position in the component tree and hence prevents the complexity of the application. React-redux introduces actions and reducers which work simultaneously with stores to make the state more predictable.

The working is as follows:

1. Store: It contains the state of the components which need to be passed to other components. The store makes this passing along much easier as we no longer need to maintain a state inside a parent component in order to pass the same to its children components.
2. Actions: The actions part of the react-redux basically contains the different actions that are to be performed on the state present in the store. The actions included must contain the type of the action and can also contain payload.
3. Reducers: The reducers in react-redux are the pure functions that contain the operations that need to be performed on the state. These functions accept the initial state of the state being used and the action type. It updates the state and responds with the new state. This updated state is sent back to the view components of the react to make the necessary changes. A reducer must contain the action type.



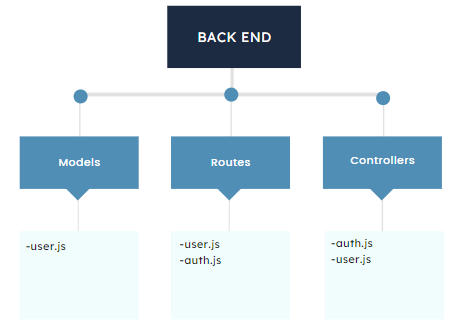
##### Figure 4.14: Difference between Prop-Drilling and React-Redux

### 

### 4.3.2 Back End

The backend of this application is built completely using Node.js. Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js brings event-driven programming to web servers, enabling development of fast web servers in JavaScript. Developers can create scalable servers without using threading, by using a simplified model of event-driven programming that uses callbacks to signal the completion of a task.

Our backend starts with an app.js file, which connects the entire web application. To ease the working we have used Express.js wrapper around Node for faster development. The application is connected with 3 folders, models, routes, controllers. The routes folder does the complete routing of different URLs, the controllers folder handles the business logic of the application, and models folder creates the basic structure of the objects used in the application i.e. the user of the application connected to the API.



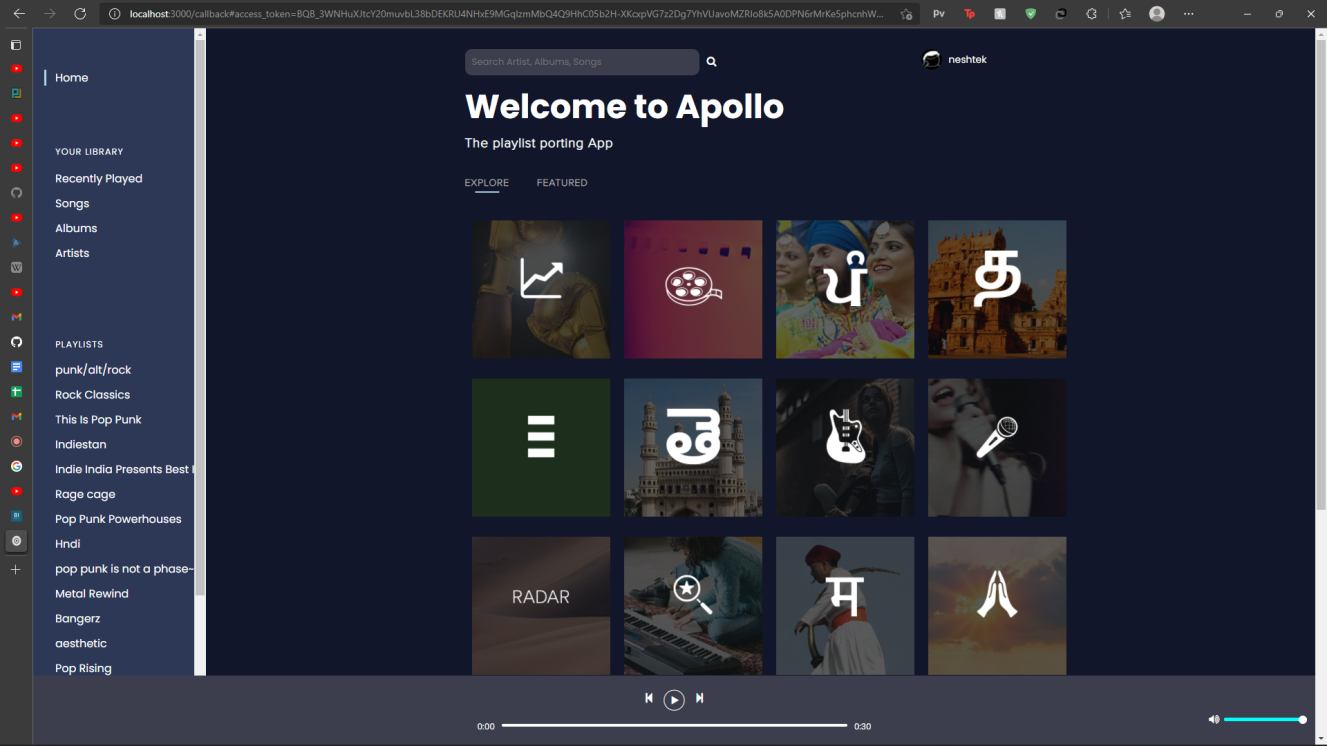
##### Figure 4.15: Structure of our Backend

## 4.3 Application

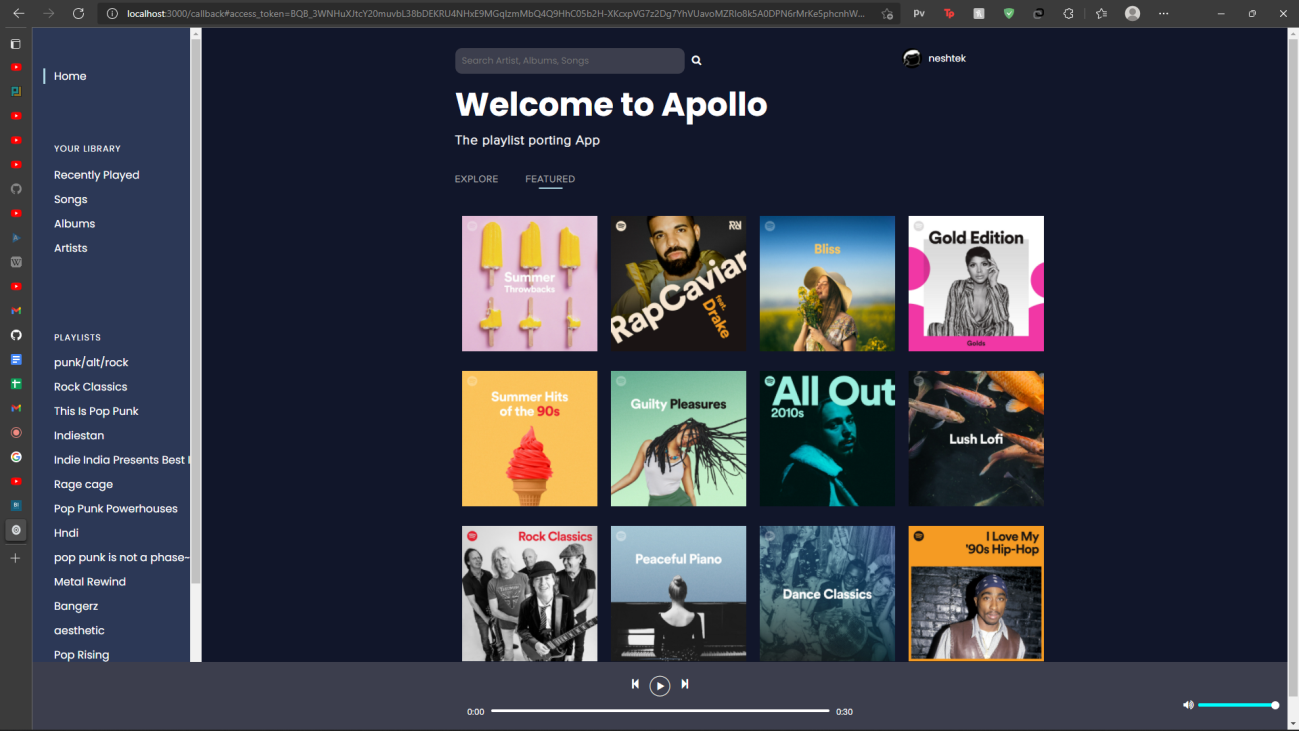
Below are the snapshots of our working application along with the description of the functionalities.

### 4.3.1 Home

The explore tab under the home section fetches the artwork being displayed through the UI Action. While the featured tab fetches the data of playlists curated by Spotify through the browse action.



##### Figure 4.16: Home Page of our Application (Explore)

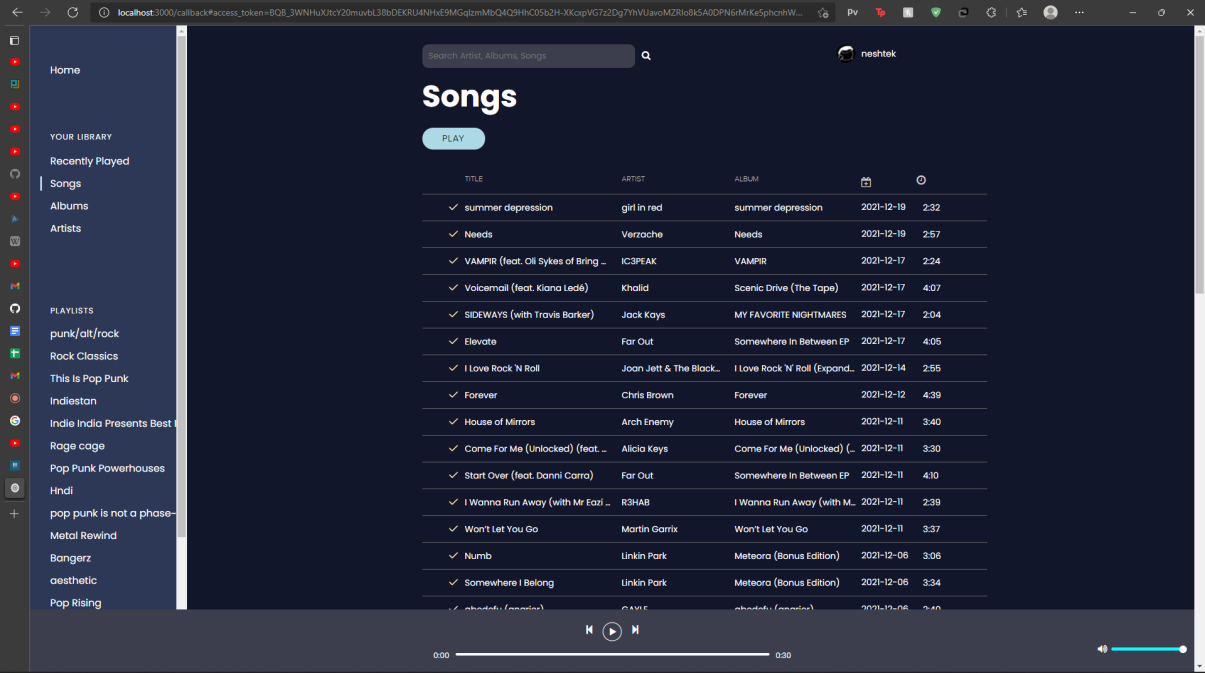


##### Figure 4.17: Home Page of our Application (Featured)

In the featured section, we retrieve custom made playlists by Spotify for the users which can also be ported to YouTube using our application.

### 4.3.2 Songs

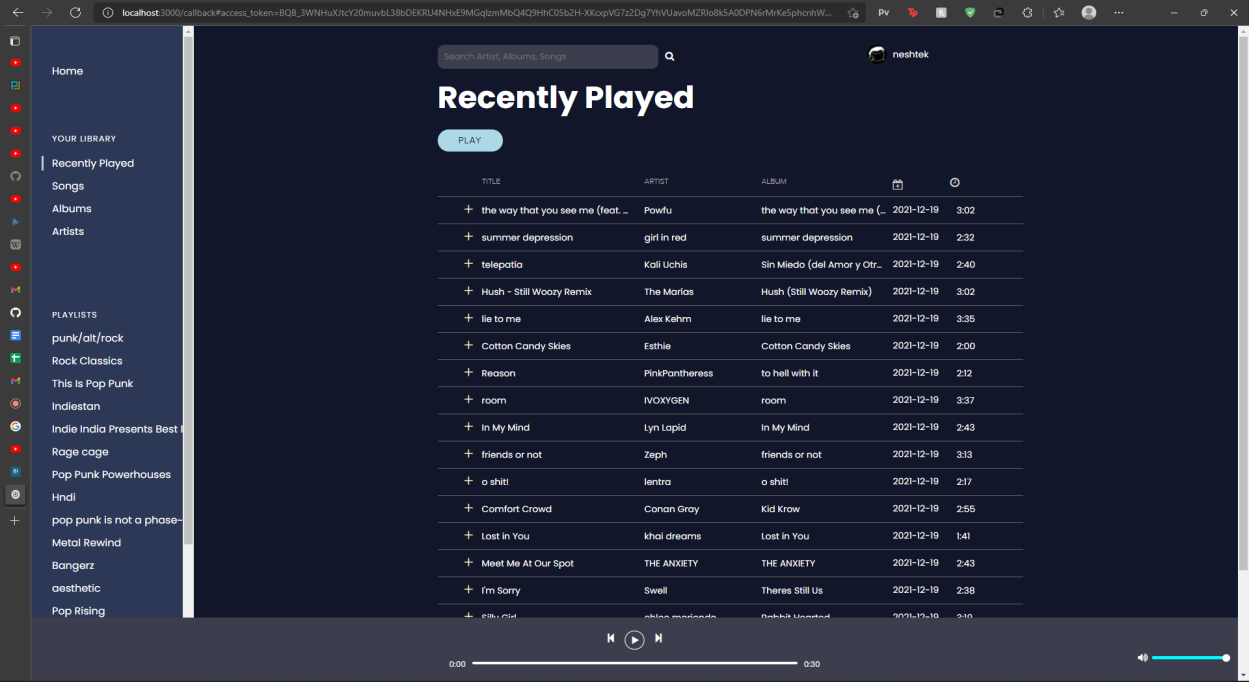
The Songs section displays a list of songs and their details such as title, artists, album, date added, and duration. The data is fetched using the song action.



##### Figure 4.18: Songs List

### 4.3.3 Recently Played

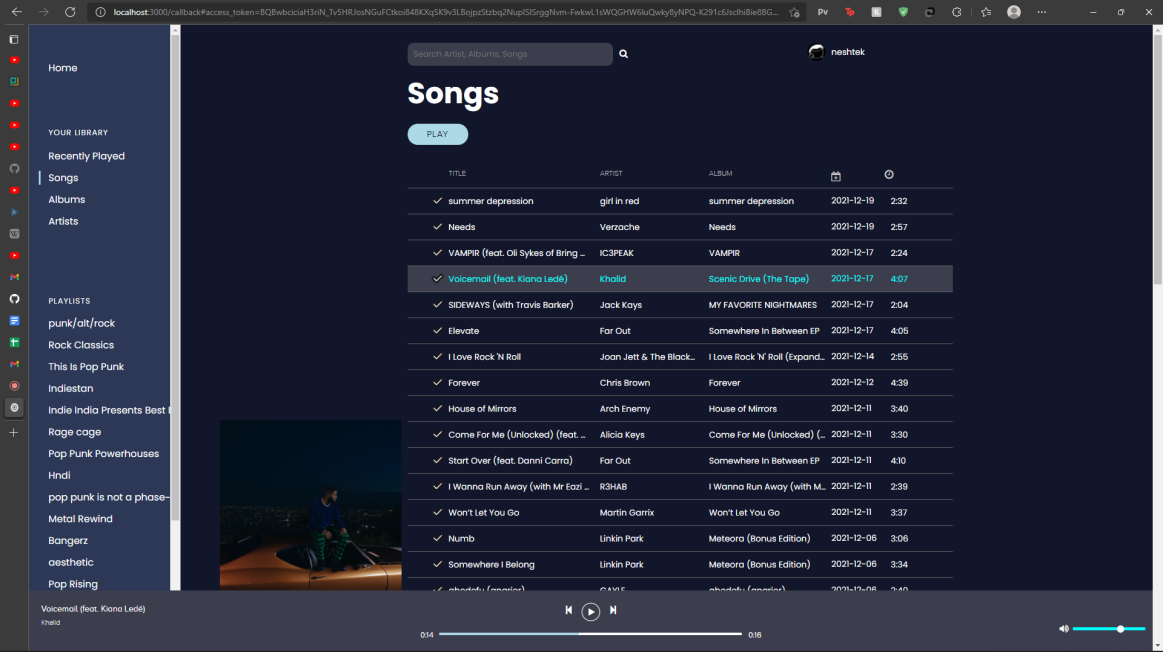
The Recently Played section displays a list of songs that the user plays recently along with their details such as title, artists, album, date added, and duration.



##### Figure 4.19: Recently Played Songs

### 4.3.3 Footer

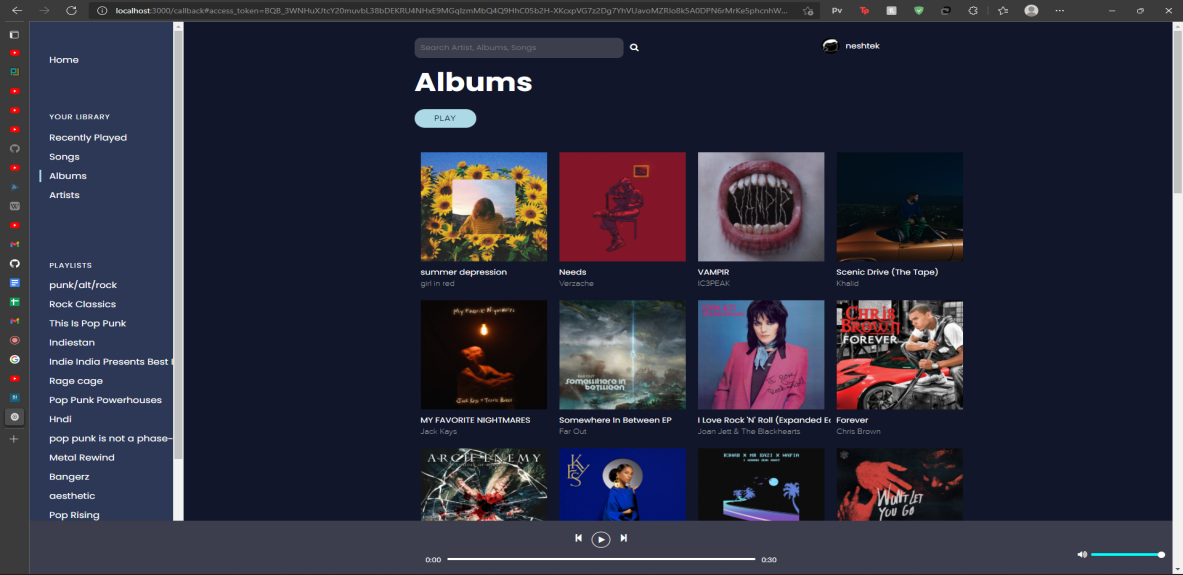
The footer is our mini music player which is divided into two components: song control and volume control. Navigation between songs in the song control takes place through the song action while the volume control uses the sound action to provide the ability to increase or decrease the volume of the song that is being played.



##### Figure 4.20: Footer of our Application

### 4.3.4 Albums

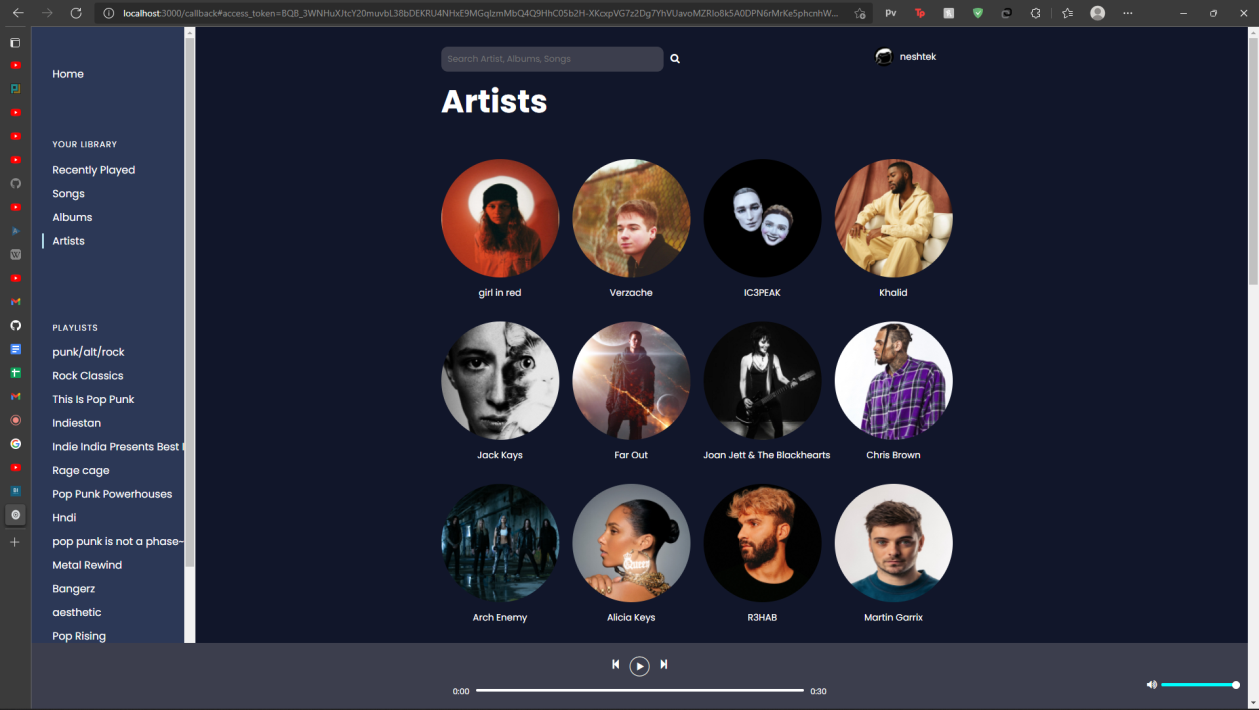
This section allows albums that are recently played by the user to be displayed using the Album Action.



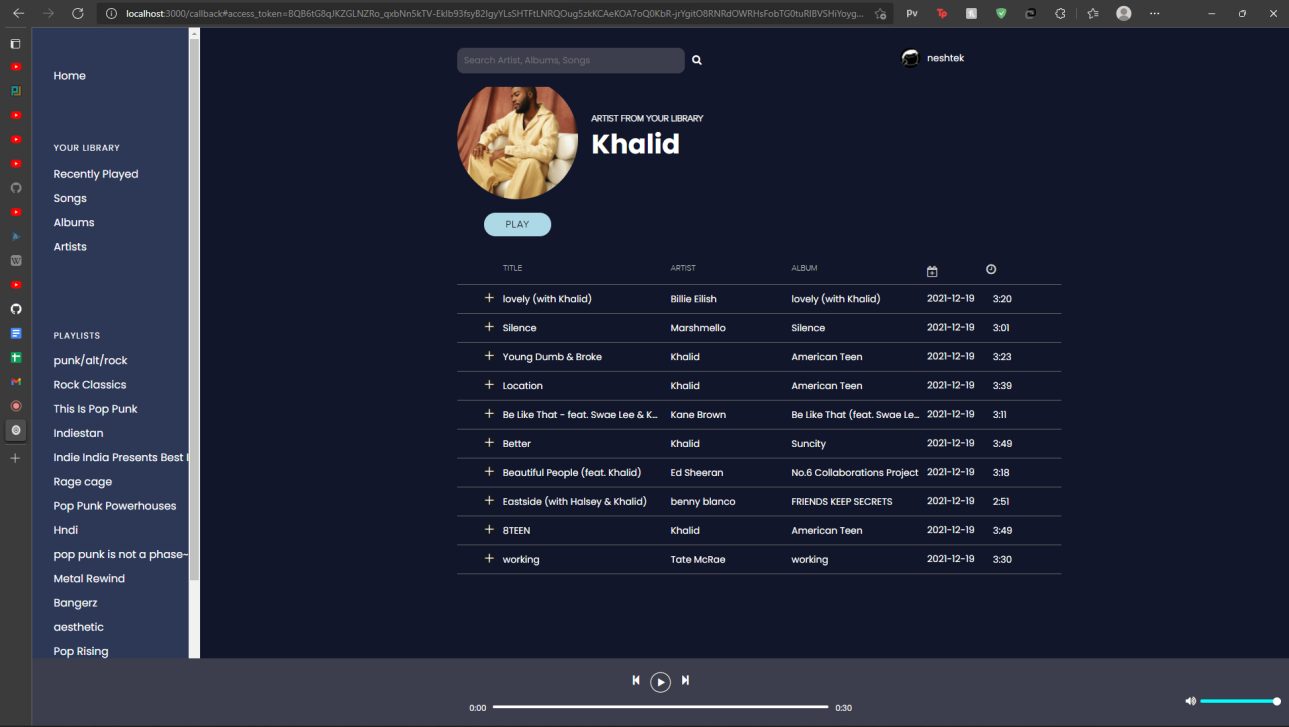
##### Figure 4.21: Albums

### 4.3.5 Artists

This section displays the user’s recently played artists, which on click opens a playlists of their songs. This is done through the artist action (recently played artist’s songs)



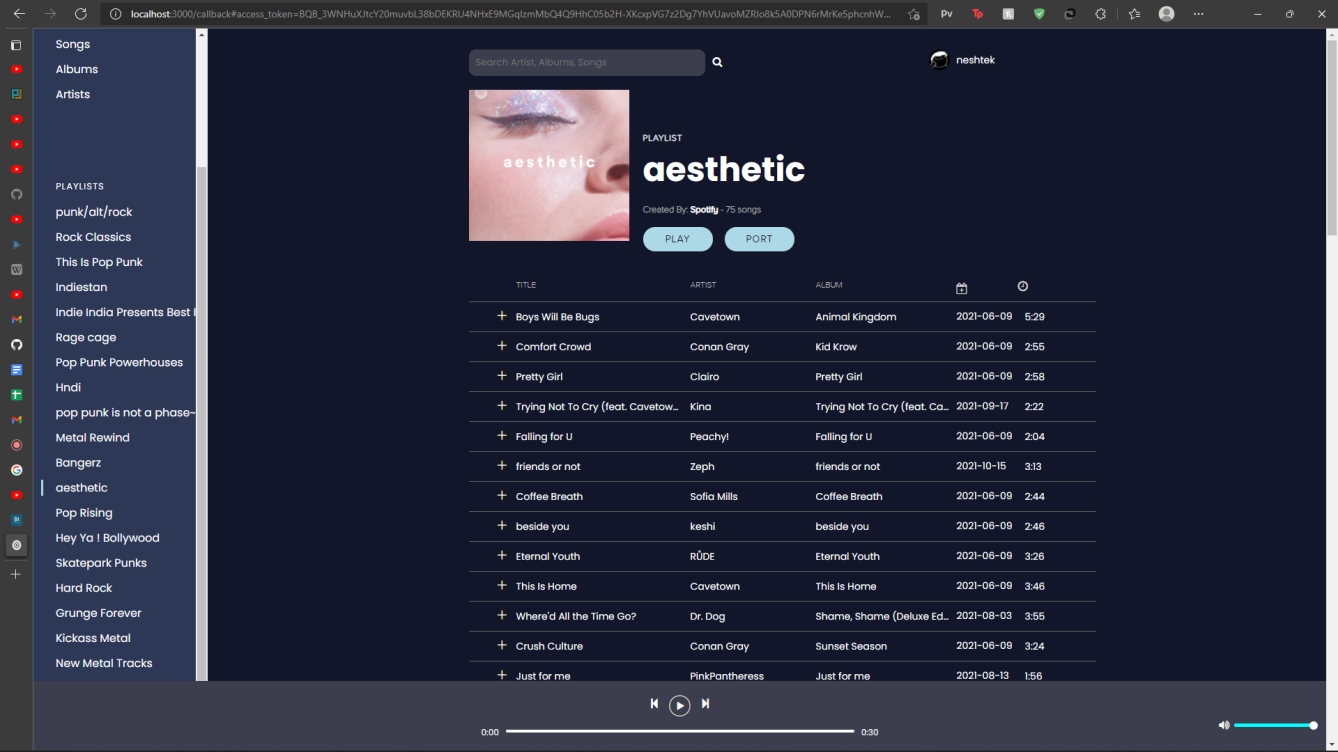
##### Figure 4.22: Artists



##### Figure 4.23: Artist Playlist

### 4.3.6 Playlists

The Playlists section displays upto 20 of the user’s playlists or liked playlists including their title, artwork, songs and the details of each song using the playlist action and song action.

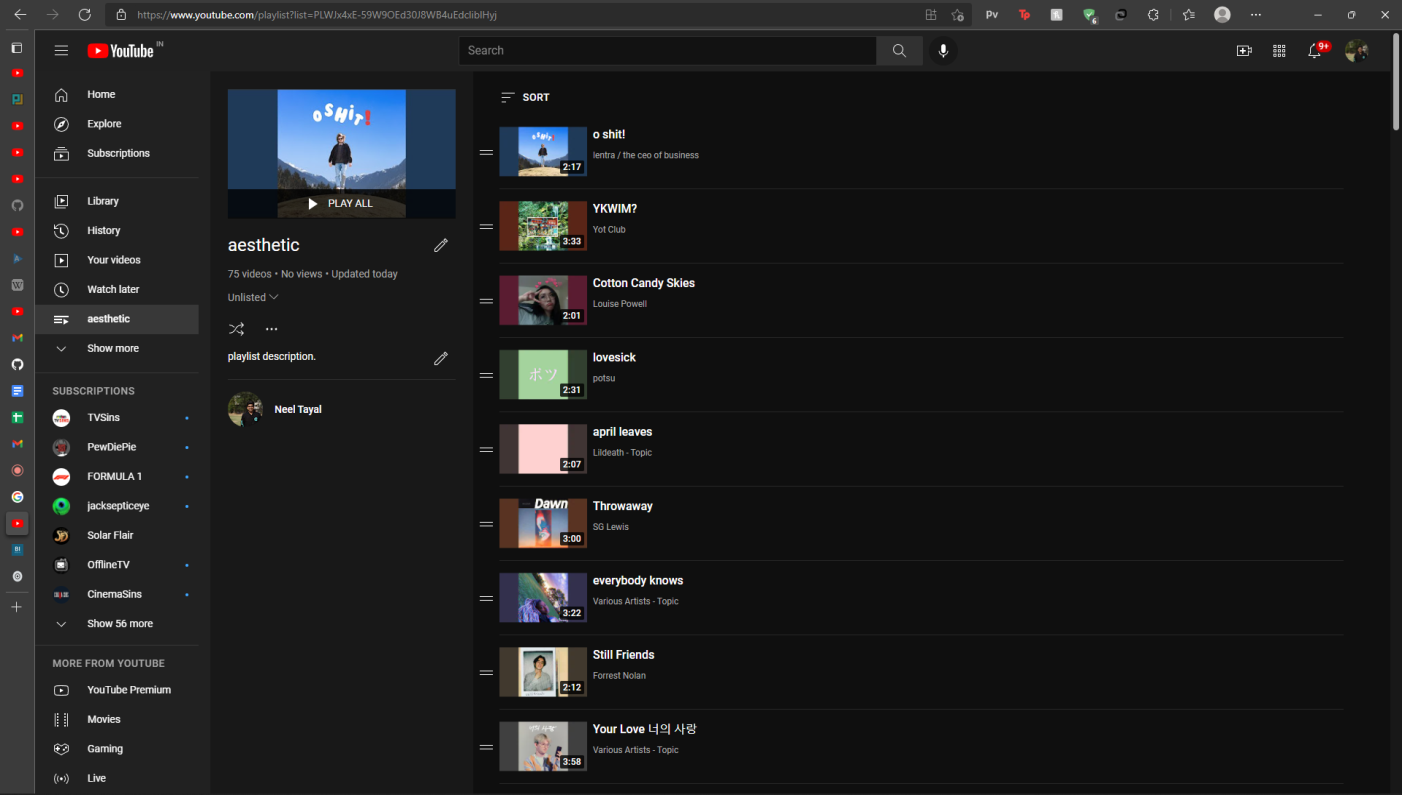


##### Figure 4.24: Our Playlist

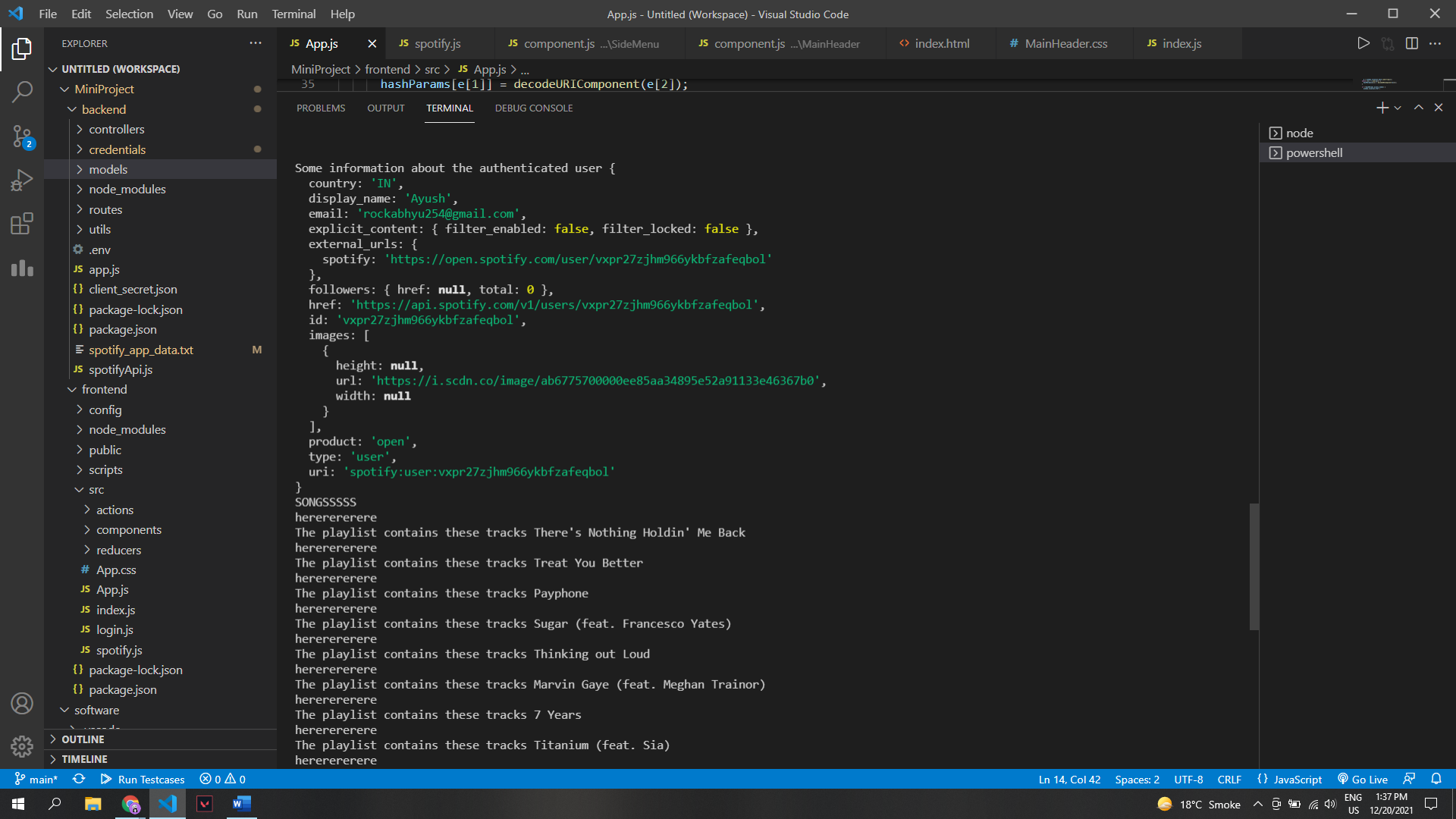
### 

### 4.3.7 Porting

Porting: First we are fetching playlist ID of the playlist that we want to port from spotify. Next through looping, we are sending repetitive requests (fetch songs) to fetch song ids from Spotify’s API which is synonymously matched through the song title, artist and album to YouTube by sending requests to YouTube’s API and then the song ID format is converted to the corresponding id of the song in Youtube in order to create a new playlist and add the respective songs to it. The playlist created in Youtube has the same name as we have in our spotify and all the songs are added in real time. We are porting one playlist at a time to make this process faster.



##### Figure 4.25: Playlist created in YouTube

****

##### Figure 4.26: Transferring songs to YT playlist

# Chapter 5 - Challenges and Limitations

## 5.1 Challenges

### 5.1.1 Error Status 403: Forbidden

To port songs and playlists to Youtube, proper authorization is required in the Youtube API. This could not be achieved initially and would throw an error. This issue was solved by creating a new source code file from scratch to change the scope of YouTube’s authorization.

### 5.1.2 Error Status 400: Invalid Body Snippet

YouTube accepts only a specific format of requests that need to be followed by the playlists. So during porting of playlists from Spotify to YouTube the format of the request was different resulting in Error 400s. We compared the request formats that YouTube accepted by fetching a pre-existing playlist from YouTube and what the Spotify API was using. After comparison we replicated the YouTube format during porting.

### 5.1.3 Asynchronous Code Error

During the initialization of both the APIs, a promise is returned which needs to be received by the corresponding code. But because of the asynchronicity of our code this process could not be completed. We re-wrote the code and switched to synchronous code to solve this issue.

### 5.1.4 Spotify API Scope Error

Spotify API needs a specific scope during initialization that details what data of the authorized user can or cannot be accessed which resulted in repeated Error 403s. This was solved by defining the scope of what we can access from a user’s account beforehand.

## 5.2 Limitations

### 5.2.1 Error Status 403: Forbidden

YouTube Quota Expired: YouTube only allows up to 10,000 requests per day per user for free. In order to extend this Quota, the developer is required to pay extra charges.

### 5.2.2 Porting of Multiple Playlists

Multiple Playlists cannot be ported to YouTube at the same time due to the fact that no particular request can be tied to the list of songs, and hence a mismatch occurs.

### 5.2.3 Stream Duration

Music streaming is limited to 30s per song as per Spotify’s API.

## 

# Chapter 6 - Future Scope

1. Our first and foremost aim for the future would be to incorporate multiple music streaming platform support in our app and extend our porting functionality to these apps too.
2. The creation of our very own recommended system using Machine Learning tools that would recommend the users new songs, artists, playlists or albums from all the incorporated apps according to their tastes in music.
3. Adding the functionality of creating new custom playlists for the users on our application on a daily or monthly basis that would include both songs that they already listen to and similar songs that they can explore.

# Chapter 7 - References

1. Michael Thelin. (Last Updated: 2021, Jan 25). *Spotify Web API Node* [Online]. Available: <https://github.com/thelinmichael/spotify-web-api-node>
2. Google. (Last Updated: 2021, May 05). *Node.js Quickstart*[Online]. Available: <https://developers.google.com/youtube/v3/quickstart/nodejs>
3. Frederick Rosales. (2018, Sep 28). *Add video to Youtube playlist NodeJS* [Online]. Available:<https://stackoverflow.com/questions/46476169/add-video-to-youtube-playlist-nodejs>
4. Maximilian Schwarzmüller. (Last Updated: 2021 Dec). *React- The Complete Guide* [Online]. Available: <https://www.udemy.com/course/react-the-complete-guide-incl-redux/>
5. Dev Ed. (2019, Jun 22). *Redux For Beginners | React Redux Tutorial* [Online]. Available: <https://www.youtube.com/watch?v=CVpUuw9XSjY&list=LL&index=16>
6. Spotify. (Last Updated: 2021). *Web API Tutorial* [Online]. Available: <https://developer.spotify.com/documentation/web-api/quick-start/>

# Chapter 8- Plagiarism Report

