Tempofiy Report

Zhou Yuan Tao   
*Department of Math and Science Mount Royal University*Calgary, AB Canada  
yzhou177@mtroyal.ca

*Abstract*—This is report about a student driven project working with the Spotify API to build an application for personal uses. Developing features such organizing existing playlist by tempo (Tempo feature); Second feature as making suggestion on an existing playlist (Suggestion feature); Third feature as concentrating an existing playlist by a selected track feature (Optimized Feature).

Keywords—React, Spotify API, Application, JavaScript, Client satisfaction, Node

# Introduction

This project is organically inspired by Zhou Yuan Tao, from Mount Royal University. During leisure hours he really enjoyed listening to music and as a musician himself he naturally practices along with his music on Spotify. The problem arises when he would start making his own playlists. Even thou, the songs he would select he would enjoy very much, but in a playlist, it is hard to practice songs that were very different in tempo. After searching for an existing app that does this for him, he found there was no such app. So that’s why the Tempoify was born. It turns out that there a good reason why nobody has developed an app like this, the process was very complicated. The first step was gaining access to Spotify API and user access. Second step was translating backend (Node.js) to frontend (React.js). Third step is implementing the API calls and working with the data. The final step was putting together a user interface.

# Tools

## Spotify API

Spotify API provides the information we need, and the existing framework enable all the functionality in the application to work.

Graphical user interface, text

Description automatically generatedTo have access to the Spotify API, you must have a Spotify account and developer account on their dashboard. After having the accounts setup, then you would have register to an application, then a client Id and a client secret is granted. That information is used later for authentication. A redirect URL must be assigned for the API to go to after authentication.

## Node.js React.js

Both frameworks stem from the JavaScript language, the initial code for authentication is from Spotify for developers Website [1], was originally in Node.js. Working off the initial code, the back end had the user authenticated and the front-end phrase the hash from the webpage and authenticated through that. The example was using j-query, which was abandoned for react.js. Using React.js and Axios API calls, I was able to achieve the same authentication for my personal use. However, the authentication was not extended to other Spotify users for unknow reason.

Node.js is back-end server language, that React is build off. There are many other functionalities which node is used for. When running the app, the app is running on a live server, which means if the terminal for the app is running is closed then the application will seize to exist. Node can also create a custom API for the app to work with. Although not currently, using this functionality, it will be soon implemented.

With Express library Node is able to create a API that the app will eventually make use of, and in order to achieve that a different type of routing will have to be made to have the front end to make use of the Node generated access. This will also require Cross origin request calls.

React,js is a mainly front-end programming framework, it specialized in single page application, and it is famous for fast reload, and clear division of components. The reason to select this framework is because of previous experience with the framework has been positive and this makes the developing process very enjoyable. The fast reload has been beneficial on saving time, but also sometimes it messes with the testing.

Axios was another framework that allows me to make API calls with ease. Unlike other library such as node fetch, and requests, the Axios library cam make API calls that work with the front and back-end. This makes the API call from Spotify API accessible for the React framework.

Axios is a promise-based HTTP Client for node.js and the browser. It is isomorphic (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js http module, while on the client (browser) it uses XMLHttpRequests [9]

When testing for bugs there can be a value that is filled by a previous iteration. Although it can be ok to make progress with consecutive steps of functionality, it is important to keep in mind that when running the application from the start might cause errors.

## Differentiate from the Spotify

Spotify API provides the information and the functionality that allows this program to work, so why wouldn’t the User just you the main Spotify application? It is well established, and it provides all the functionalities that I would be using.

Although Spotify has all the power that can make this happens, Spotify is catering to as wide as an audience as possible. From market research, there is very few people who are working with their own playlists. Most people are listening to albums, which is pre-determinant by the artists, or they are listening to playlist that is generated by Spotify.

Timeline

Description automatically generatedAs a company Spotify has decided to have the process of suggestion and the variable that determines the suggestions to be hidden. This application intent to put the power back in the hands of the user. This app doesn’t intent on replacing Spotify, but rather an add-on to the existing service for the users who are enthusiastic about their own selections.

Although the attempt to have the variable revive for me and the user to use has been fruitful, there is still so many other hidden variables Spotify has keep private. This app is working with the variables that are one level deeper than what is normally visible.

# Methodology

Working with professor Khosro Salmani, we used Agile methodology to determine the long-term goals (over the school semester), and determine short term goals (over the week)

Goals determined over long periods of time was subject to change as we progress through and see what was possible. Weekly goals were mostly met and the suggestions for the next focus was determined through weekly meetings.

## Understanding existing code

There is a existing code that run through the authentication process. This uses Node.js to authenticate to the Spotify API. The front-end HTML search for the hash and the token to generate the access token. [10]

I have elected to forgo this approach. As the front-end use j-query and that is not some thing I am comfortable with.

The approach I ended up using is like this, but the everything is done through React. The access token is grabbed directly through the API. This token has an expiring time. So there is an log in and log out button to have the token to be refreshed.

## API calls

An Application Program Interface is what API stands for. For this project the API I will be working with is the RESTful API that allows the GET method, POST method, and PUT method.

API calls will be very common in this application. Almost all the functionality requires an API call to supply the data using the GET method, to get the data from Spotify. There are two calls that requires the Post method, to create a new playlist for the user, using the Spotify API, the playlist will generate inside of the user’s account. Then the PUT Method is called insert the desirable track s to this new playlist.

Each of the API call requires an authentication header. The process of obtaining the token will explained in the following section

## Authentication

Authentication has been the first big hurdle of this project. Later the full authentication process will be displayed, but as of today, the authentication processes the app is currently using is listed as below. Since a Spotify account is already logged on to the machine, I am using so the authentication process would be a simple API call

## Difficulity through out the semester

A lot of time weekly goals weren’t met because of unrealistic expectations. The learning curve was underestimated to create the first functionality. Time required for the project during that period was severely underestimated.

# Functionality

Before each of the functionality, I wanted to have all the user’s playlist to be displayed in the user interface.

Each functionality was inspired differently. Tempo functionality was the core inspiration for the application. Then it expanded to different way to filter the features of the tracks, that transformed into Optimize. Then the suggestion feature was inspired by the recently added API by Spotify.

## Tempo

Tempo functionality allows the user to select a range of tempo base on the number of bpm (beats per minute). After selecting a playlist to work with, the user can input the minimum bpm and maximum of bpm they desire. Then songs will be displayed as preview and if the results are to the user’s liking, then the user can generate a new playlist with those songs, and they will have it in their Spotify account.

The process for making this process had to go through numbers of API calls. After having the selected playlist in a state, then an API call had to made to get all the tracks [2]. Then Tempo feature for each track had to had to have an API call each [3]. After having the desire tracks, to generate a playlist on Spotify, there must be an API call to get the current user [4], then create a new playlist [5], then add all the songs to the new playlist [6].

Tempo was the first and the most difficult part of this project. Number of difficulties was encountered throughout this feature

### UseState. Working with the useStates from React was difficult, as the state didn’t update in real time, as JavaScript is an asynchronous language and the way useStates is structured.

“Use State is a Hook (function) that allows you to have state variables in functional components. You pass the initial state to this function and it returns a variable with the current state value (not necessarily the initial state) and another function to update this value.”( [useState in React: A complete guide - LogRocket Blog](https://blog.logrocket.com/a-guide-to-usestate-in-react-ecb9952e406c/#:~:text=useState%20is%20a%20Hook%20(function,function%20to%20update%20this%20value.))

### Promises. Some API calls has to not go through as I expected. A lot of data has been undefined. The GET API call are very quick, and there is no practical difference when excucting. When using GET, POST or PUT request together that is when the problem arise. POST and Put request takes longer than GET requests. When working with a loop of combination of GET and POST API calls, is when the logic falls through, sometime not all request go through fully, or the requests are stacked on top of each other, and mutiple of the same call are made.

### To have the promises work in order. The code had to be wrapped in a promise and resovle. When the Post requests gets resloved then the next tasks will be excuated.

### Scopes.[12] Making the API call through Spotify API requires different scopes making a call out side of the scope will still result in a complete call.

## Optimized

Optimizing for a particular feature was all encompassing of every feature that was provided by the Spotify API. Like the Tempo functionality, the user can select a track analysis feature they chose to focus on. Feature such as Acousticness, Instrumentalness, Energy, Danceability and Speechiness, all have a number value from 0 to 1, and closer the value is to 1 the stronger the track is to embody the feature. For example, if a track has a Danceablity value of 0.8, then the track is very likely to get people dancing. That’s the case with every feature with this analysis.

Unlike Tempo functionality, user was able to select a value to work with, but the Optimized has simplified the process and work of the average value for each playlist’s feature. The songs are ranked based on how close their feature value is to the average. The user will select how many “top” songs they elect to have in their new playlist. Then a preview is shown, and if it is to the users liking then a new playlist is generated and available in the user’s Spotify.

Tempo was the example feature that helped with this functionality. Similar calls were made, and in addition there must be a switch case to handle all the different features.

## Suggestions

Unlike the other two functionalities, user was filtering through existing playlists to make them more compact, the suggestion functionality is working with the new API call that get the suggestion for a max number of 5 songs. This way the functionality is pulling songs that are like the seed songs.

To have the seed songs, the user, like all the other functionalities, select a playlist to work with. Then all the track features analysis are added up and averaged out. Feature such as Acousticness, Instrumentalness, Energy, Danceability and Speechiness. The five tracks that are closest to the average is selected as seed songs and up to 20 suggestions are made by Spotify.

Similar API to the other functionality were called and in addition the Get recommendation API call [7]. was called.

## Error handling

After the user click on the generate playlist button, there is an API call that generates the playlist and adds the playlist to the landing playlist. To capture if there the call makes it through without and error, there has to be an error checking method to check what the status code for the API call is.

| Status Code | API Status Code Table | | |
| --- | --- | --- | --- |
| Description | Use case | Frequency in this app |
| 200 | Successful API call | GET | Frequent |
| 201 | Created something with the API call | POST, PUT | Once per playlist creation |
| 401 | Unauthorized API call | GET, POST, PUT | Avoided |
| 429 | Too many requests | GET | Avoided |
| 555 | Token status | POST, PUT | Once per playlist creation |

The 555 status was manually set by me to refresh the status states. To have the message to display there must be a reset. When there is a change in the use States The pages refresh and automatically reload all the component on the page. The component that is connected to display the message upon a successful playlist creation has a disappearing effect, that way it avoids confusion.

## User Inferface

The general layout is setup using a grid layout from the bootstrap library. The back-ground color is by default white and unchanged for this project. The backdrop for the playlist and the track in that playlist.

The user will have their profile picture displayed on the top left corner of the screen. This is grabbed through the Spotify API and the first, the most recent, profile picture is displayed. It is displayed in a circle style to emulate other popular app.

All the user’s playlists are listed on the left side under the profile picture, all the playlists are borderless buttons that stacks on top of each other, for readability, the section is set to the remining height of the first page. To let the user access all the playlist, there is a scroll feature that let the user to stay on the first page. After the use click on a playlist then the style of the button changes to have green highlights. This way the user can keep track of the current playlist they are working with.

The top section has a brief introduction about the App. Then all the functionality is spread out evenly along the page. Before any Playlist is selected, an indicator is directing the user to select on of the playlist. All the user’s playlists are listed on the left side under the profile picture, all the playlists are borderless buttons that stacks on top of each other, for readability, the section is set to the remining height of the first page. To let the user access all the playlist, there is a scroll feature that let the user to stay on the first page. After the use click on a playlist then the style of the button changes to have green highlights. This way the user can keep track of the current playlist they are working with.

After the playlist is selected all the songs are displayed in the top section of the app, under the functionality buttons.

The layout will have the bottom section indicate to select a functionality up top, the bottom section will change and have a brief description about the functionality. There will also be user input field that let the user interact with the app.

All the functionalities are conditional rendered. All the working arrays are emptied upon change of functionality.

Table

Description automatically generated

Graphical user interface, text, application, Teams

Description automatically generated

Table

Description automatically generated

Graphical user interface, text, application

Description automatically generated

##### Future Development

Diagram

Description automatically generatedThe next functionality for the project is to have been making the application available to other Spotify users. The authentication process for Spotify API is outlined in the figure below.

As previous section has covered, the current application is only making use of the first request.

Figure Authentication code process

## React Router

What is React Router? React.js is specialized for have a single page application and React Router is a tool that let the developer to have different pages using different routes (different URL) more easily while using the same skills as managing components in React.

React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL. [13]

Right now, the application is all in a single page. Everything is generated on top of the same background. Using React Router it allows me to have more dynamic approach to the project.

Since the Tempo feature is focused on musicians. The feature tab will be named as For Musicians. The Optimized feature will be named as For Causal since it doesn’t require the client to have any working knowledge with the track analysis. The suggestion feature will be under the same tab, this way the app can specialized for different groups of people.

The background will have to different for each of these features and it would be so much easier with React router and not have to hide the other components and have a clear division.

## Last.fm

One of the original ideas is to have how many times a song is played and filter out the lowest number of plays amounts a playlist. Unfortunately, this information is not available through Spotify API.

But the saving grace is there is an API that allows the user to keep track of that data. Last.fm keeps track of how many times a use has played a song. This requires the user to link their Spotify account to last.fm

This service will only keep track of the numbers of time play upon linking the services. This means that the previous data will only be available to Spotify.

Last.fm have can also link other music services, such as apple music, google music and many others.

## User suggestions

Through many users’ suggestion there is a over whelming demand for the sorting of the playlists.

Unlike what I personality wanted is to have the song to be completely filtered off, many users like to keep all their songs, but rather sort the Tempo (or other feature) from highest to lowest, or the other way around.

This was not all challenging, just a simple grabbing the playlist and make the same call as the functionality to grab the data and spit it out to a new playlist.

I have done this in the spare time, by copying their playlist to have a copy and work off my own account. However, this model will not be sustainable. This will not be a problem when the application allows for multiple users

##### Conculsion

The user interface is base of the bootstrap template [8].Tempoify is an ongoing project there are still functionalities and features to be added. The project is in the early stages of user testing. There has been a lot of positive feedback amounts the users and a few suggestions for improvements.

##### References

1. Developer.spotify.com. 2022. Web API | Spotify for Developers. [online] Available at: <https://developer.spotify.com/documentation/web-api/> [Accessed 10 April 2022]
2. Developer.spotify.com. 2022. Web API Reference | Spotify for Developers. [online] Available at: <https://developer.spotify.com/documentation/web-api/reference/#/operations/get-playlist> [Accessed 10 April 2022].
3. Developer.spotify.com. 2022. Web API Reference | Spotify for Developers. [online] Available at: <https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features> [Accessed 10 April 2022]
4. Developer.spotify.com. 2022. Web API Reference | Spotify for Developers. [online] Available at: <https://developer.spotify.com/documentation/web-api/reference/#/operations/get-current-users-profile> [Accessed 10 April 2022].
5. Developer.spotify.com. 2022. Web API Reference | Spotify for Developers. [online] Available at: <https://developer.spotify.com/documentation/web-api/reference/#/operations/create-playlist> [Accessed 10 April 2022].
6. Developer.spotify.com. 2022. Web API Reference | Spotify for Developers. [online] Available at: <https://developer.spotify.com/documentation/web-api/reference/#/operations/add-tracks-to-playlist> [Accessed 10 April 2022].
7. Developer.spotify.com. 2022. Web API Reference | Spotify for Developers. [online] Available at: <https://developer.spotify.com/documentation/web-api/reference/#/operations/get-recommendations> [Accessed 10 April 2022].

Mark Otto, a., 2022. Bootstrap. [online] Getbootstrap.com. Available at: <https://getbootstrap.com/> [Accessed 10 April 2022].

Getting Started | Axios Docs. (2022). Retrieved 12 April 2022, from

https://axios-http.com/docs/intro

GitHub - spotify/web-api-auth-examples: Basic examples to authenticate and fetch data using the Spotify Web API. (2022). Retrieved 12 April 2022, from https://github.com/spotify/web-api-auth-examples

useState in React: A complete guide - LogRocket Blog. (2022). Retrieved 12 April 2022, from https://blog.logrocket.com/a-guide-to-usestate-in-react-ecb9952e406c/#:~:text=useState%20is%20a%20Hook%20(function,function%20to%20update%20this%20value

Authorization Scopes | Spotify for Developers. (2022). Retrieved 12 April 2022, from https://developer.spotify.com/documentation/general/guides/authorization/scopes/

ReactJS | Router - GeeksforGeeks. (2022). Retrieved 12 April 2022, from https://www.geeksforgeeks.org/reactjs-router/#:~:text=React%20Router%20is%20a%20standard,how%20the%20React%20Router%20works.