CS322 Music Programming Project

Maynooth University
National University of Ireland Maynooth

(Spotify Unwrapped) A Web Application Implementing Spotify API & YouTube API v3.

Liam Aspell

17300046

Table of Contents

Introduction

Motivation

Spotify is currently the world's most popular music streaming platform, with over 180 million paying subscribers as of 2022. Personally, it is my go-to service for music consumption, mainly because I can access a nearly unlimited supply of music from all my devices with this one application, pick and choose songs to add to playlists, and start group sessions with my friends while socialising.

At the end of 2016, Spotify released 'Spotify Wrapped' where users received a collection of their streaming data, which informed the user of metrics such as "listened to tracks", "most streamed artists", and "favourite genres" in the past calendar year. I remember being blown away by the level of detail of data Spotify could measure for its premium subscriber community. Since then, I eagerly await Spotify to release my new 'wrapped' analytics at the end of each year. My focus with this project is to return this data over a shorter timeframe, with the assistance of Spotify's developer API. The Spotify API allows users to retrieve some of the same information which is posted yearly as part of wrapped, so the idea to collect and display this data over a shorter period seems a feasible one to me, and a project which could be a valuable source of information to frequent Spotify users.

The Idea

Taking inspiration from Spotify Wrapped, as mentioned above I intend to build an application which provides data to users about their streaming habits. Information such as most played artists can vary every few weeks, to correlate with different times of year (One example, some songs may have more of a summery or winter vibe) so it would be useful to track this information more frequently. To do this, I will use some of the API endpoints associated with users' personal data, such as calls to retrieve the users most listened to artists, and their most played tracks. To complement this, I want to be able to provide some information about the artists such as their genre, their number of followers and other information which may be informative to the end user. I am also planning to have a built-in player within the application, which makes a GET request to play a specific track tied one of the most streamed artists, or the specific track returned in the ‘most streamed tracks’ page of the application. I hope to enhance this by also including an audio visualizer and displaying the lyrics of the specified track.

Resources Found

To put all the parts of my project together into one single web application, I began compiling some useful resources to aid my development. This was important in shaping the initial ideas from an implementation perspective. The following is an account into some of the research I carried out before working on the project, it is not an exhaustive list.

Spotify API Documentation/Console:

The Spotify API documentation contains comprehensive information for the many different API endpoints developed by Spotify to handle users’ requests. This documentation helped me to refine the specifications for the application I want to build, by targeting the specific data I could extract, as well as other API calls for functionality I wanted to implement, such as displaying the corresponding album names associated to most played tracks, and their release dates. The documentation was also useful for information regarding the use of OAuth 2.0, which was new to me, before I began the project. Spotify’s developer resources also include a built-in console, that runs on the browser, to quickly make API calls, allowing me to quickly analyse the data that was extracted for any specific call. I used this consistently throughout the project, and it was particularly helpful when structuring query parameters for respective API calls.

React Framework / Bootstrap Documentation:

As the project was built with the aid of the React Framework, my front-end implementation would make use of components, hooks etc. To complement my understanding of how React works, I made use of the official react documentation, which was available online, as well as the textbook “React: Explained”. I chose also to use React-Bootstrap on the front end of my application. React-Bootstrap provided useful features such as styled buttons, modals and components such as navbars and cards, which I could reconfigure to aid front end development of my project.

Npmjs.com:

In my approach to building this application I watched a few tutorials online of react web applications created which pulled and represented data retrieved through API calls. One such video made use of the library ‘React-Spotify-Web-Playback’. I liked the simplicity of the library, and therefore looked for more information regarding the library. I found this information on a website consisting of all the different libraries that can be implemented with the use of Node Package Manager (npm). The specific page for this web player provided all the information needed to get the player running on my application, playing popular songs from the specified artist, or playing a specific track where necessary. It also contained all the information needed to style the player, by giving a detailed list of different styling parameters. I referenced the npmjs documentation when I had any questions about the other libraries used in my project.

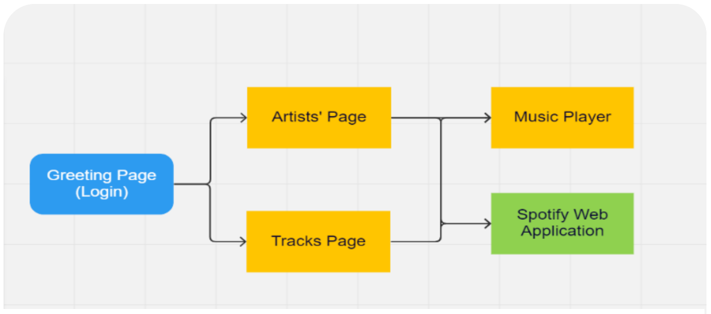
Axios Documentation:

Before the project I had good knowledge of how API’s work but had not implemented an API call inside a react web application before. Some research on this led me to the conclusion that the most popular way of doing this was by making use of Axios, a popular HTTP client library. I referenced the Axios online docs frequently, to help guide my use of Axios to make API calls to a server within the react framework.

Methodology

Design

As mentioned previously, I intend to provide useful user data using Spotify's API, while encapsulating other useful information in my application around the forementioned data, and to build a player within the app which relies upon Spotify API's endpoints around playback state. To do all this concurrently, I have enclosed my initial design in the UML diagram aside.



Basic UML Diagram of My Applications Functionality

I want to use ReactJS to handle my front-end development of the pages, complementing this with React Bootstrap to handle styling of components, buttons along with other components to enhance the UI of my application. This will provide an attractive interface for users which is easy to understand. In terms of the returned data, I will style this in a grid formation and use cards for each individual bit of returned data. Each card will also contain buttons for links to Artists Spotify Page, and to trigger modals for more artist information and the player respectively.

On the backend, I intend to construct two API calls to serve both the Artists and Tracks pages, which are fed different parameters depending on whether the user wishes to display their top artists or tracks. The artist's / tracks name is then kept as a variable to instruct the built player of the music which should play when called in a particular card, or to display a modal containing more artist or track information.

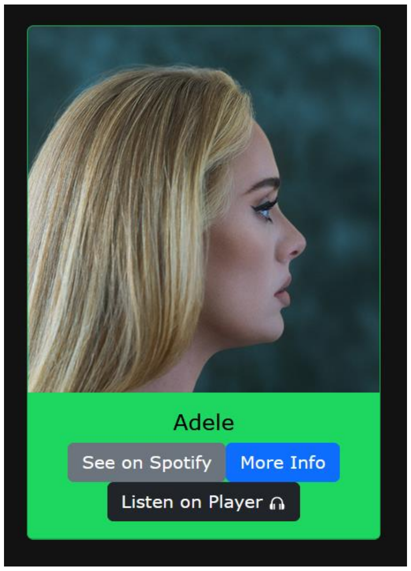
Implementation

Configuring Authentication with Spotify API:

Spotify API requires OAuth 2.0 authentication, to access the vast library of different API calls. To do this, the application would need an access token, for Spotify to recognize the application and allow API calls to be made. The scope of these tokens can vary, depending on what the app developer would like their application to do, and then specifying the scope of a particular token to do these operations. I built out the authentication aspect of the project firstly by setting up a new project on the Spotify developer console. I could then specify this project in the login process of my React application, while also specifying other parameters, such as the token scope, and which redirect URI from my predefined set to return the user to on my application. The user's login process initially requires redirection to a Spotify login form, login and return to application, but I found this only needs to happen once. Once it is completed, the user's login information is cached in the browser so they can revoke or reinstate their access token quickly without the need to repeat this process. This can be seen in the demo and has been verified to work on Chrome as well as Firefox.

Styling with React Bootstrap / CSS:

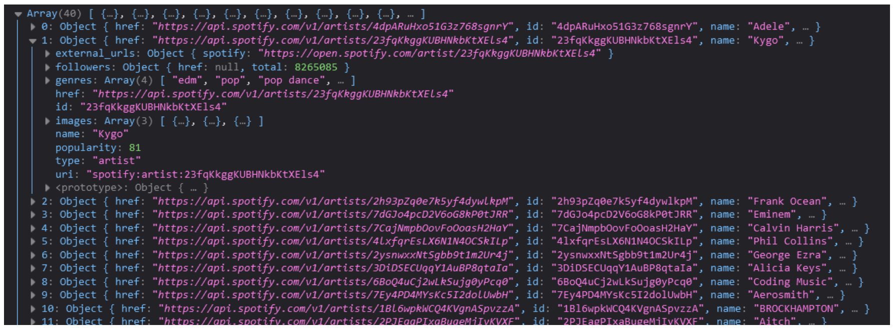
Styling on the Front-End was implemented with the use of Cascading Style Sheets (CSS) and the Bootstrap library which caters for the creation and customization of styled components in React, known as React Bootstrap. React Bootstrap helped me to customize and define components such as buttons, cards etc. to store returned data, the navigation bar along with others. In addition to this, to handle the internal spacing of elements on the pages of the application, and to define a consistent colour scheme, I used CSS. Using class names for my div elements inside my application, and then defining these class names inside my CSS code, I could easily style multiple components uniformly. This is exemplified inside my code submission.



Application of the 'Card' Styled Component

Pulling User Data through the API Calls:

I used Axios to make API calls to the Spotify servers to pull relevant data. The main body of the project was to pull an authenticated Spotify profiles top 40 streamed artists, and top 40 most streamed tracks over a period of 4 weeks. This was

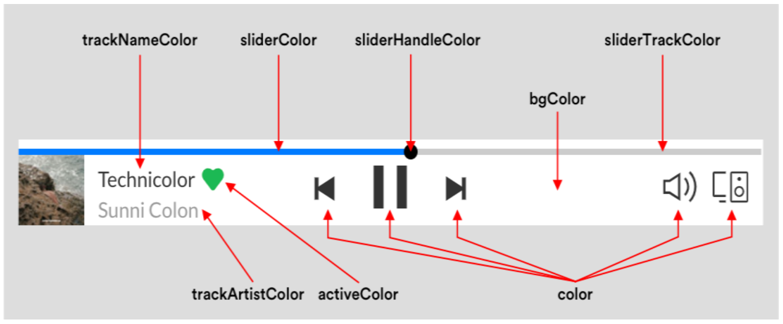


```
▾ Array(40) [ {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, … ]
  ▸ 0: Object { href: "https://api.spotify.com/v1/artists/4dpARuHxo51G3z768sgnrY", id: "4dpARuHxo51G3z768sgnrY", name: "Adele", … }
  ▾ 1: Object { href: "https://api.spotify.com/v1/artists/23fqKkggKUBHNkbKtXEls4", id: "23fqKkggKUBHNkbKtXEls4", name: "Kygo", … }
    ▸ external_urls: Object { spotify: "https://open.spotify.com/artist/23fqKkggKUBHNkbKtXEls4" }
    ▸ followers: Object { href: null, total: 8265085 }
    ▸ genres: Array(4) [ "edm", "pop", "pop dance", … ]
      href: "https://api.spotify.com/v1/artists/23fqKkggKUBHNkbKtXEls4"
      id: "23fqKkggKUBHNkbKtXEls4"
    ▸ images: Array(3) [ {…}, {…}, {…} ]
      name: "Kygo"
      popularity: 81
      type: "artist"
      uri: "spotify:artist:23fqKkggKUBHNkbKtXEls4"
    ▸ <prototype>: Object { … }
  ▸ 2: Object { href: "https://api.spotify.com/v1/artists/2h93pZq0e7k5yf4dywlkpM", id: "2h93pZq0e7k5yf4dywlkpM", name: "Frank Ocean", … }
  ▸ 3: Object { href: "https://api.spotify.com/v1/artists/7dGJo4pcD2V6oG8kP0tJRR", id: "7dGJo4pcD2V6oG8kP0tJRR", name: "Eminem", … }
  ▸ 4: Object { href: "https://api.spotify.com/v1/artists/7CajNmpbOovFoOoasH2HaY", id: "7CajNmpbOovFoOoasH2HaY", name: "Calvin Harris", … }
  ▸ 5: Object { href: "https://api.spotify.com/v1/artists/4lxfqrEsLX6N1N4OCSkILp", id: "4lxfqrEsLX6N1N4OCSkILp", name: "Phil Collins", … }
  ▸ 6: Object { href: "https://api.spotify.com/v1/artists/2ysnwxxNtSgbb9t1m2Ur4j", id: "2ysnwxxNtSgbb9t1m2Ur4j", name: "George Ezra", … }
  ▸ 7: Object { href: "https://api.spotify.com/v1/artists/3DiDSECUqqY1AuBP8qtaIa", id: "3DiDSECUqqY1AuBP8qtaIa", name: "Alicia Keys", … }
  ▸ 8: Object { href: "https://api.spotify.com/v1/artists/6BoQ4uCj2wLkSujg0yPcq0", id: "6BoQ4uCj2wLkSujg0yPcq0", name: "Coding Music", … }
  ▸ 9: Object { href: "https://api.spotify.com/v1/artists/7Ey4PD4MYsKc5I2dolUwbH", id: "7Ey4PD4MYsKc5I2dolUwbH", name: "Aerosmith", … }
  ▸ 10: Object { href: "https://api.spotify.com/v1/artists/1Bl6wpkWCQ4KVgnASpvzzA", id: "1Bl6wpkWCQ4KVgnASpvzzA", name: "BROCKHAMPTON", … }
  ▸ 11: Object { href: "https://api.spotify.com/v1/artists/2PJEagPIxaBugeMjIyKVXF", id: "2PJEagPIxaBugeMjIyKVXF", name: "Aitch", … }
```

Data from top Artists API call in console implemented by making an API call (using Axios) in the code, and authorizing the call using the token retrieved using the authentication method explained above. When making this call, I first displayed the entirety of the return message in the browser console, which was in JSON. I could then determine the data I intended to include in my project and proceed to build react components containing this data as it was required.

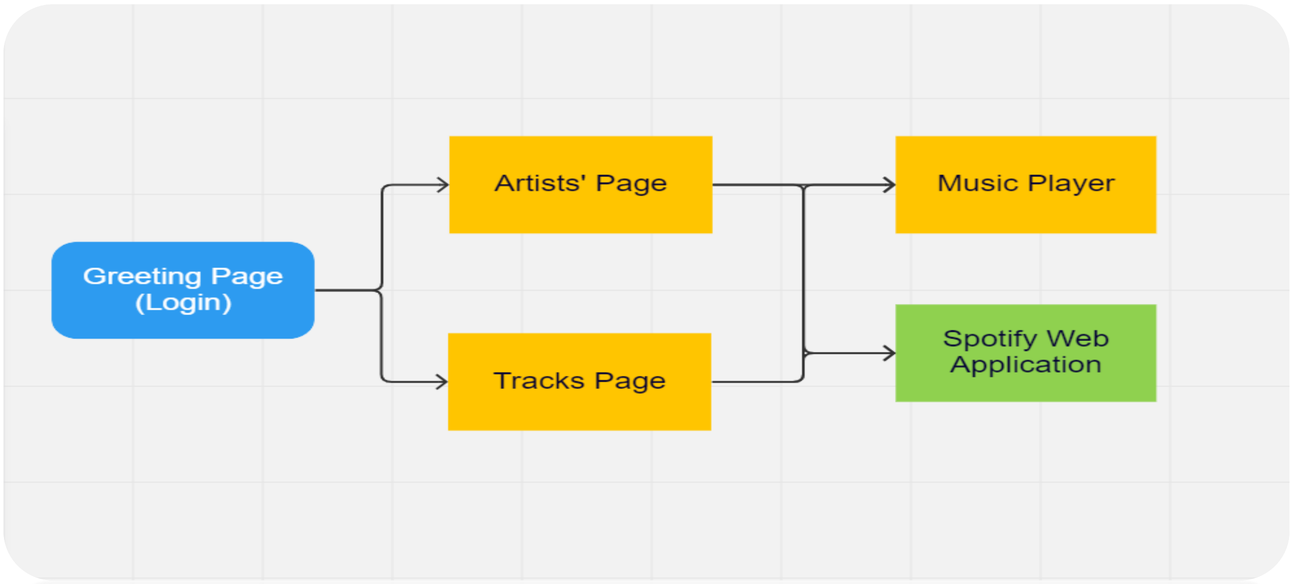
Building a player with Spotify-Web-Player library:

To facilitate the streaming of music inside my web application, I used a library known as 'react-spotify-web-playback'. This library exists within node package manager, and therefore could be easily imported to my application, using a simple np, command. Once imported, I could create a component to display the player. As the player would be inside a modal component which opens / closes through a button, I placed this player component inside a modal component. This is a helpful feature of the React framework which justifies my design choice. Once implemented, the player could be customized to include various buttons, such as volume sliders, playback control and next / previous toggles and a button to display a list of nearby devices which could be casted to. This allowed me to create a customized interface of the player with all the features I would like to be included, including those listed above.

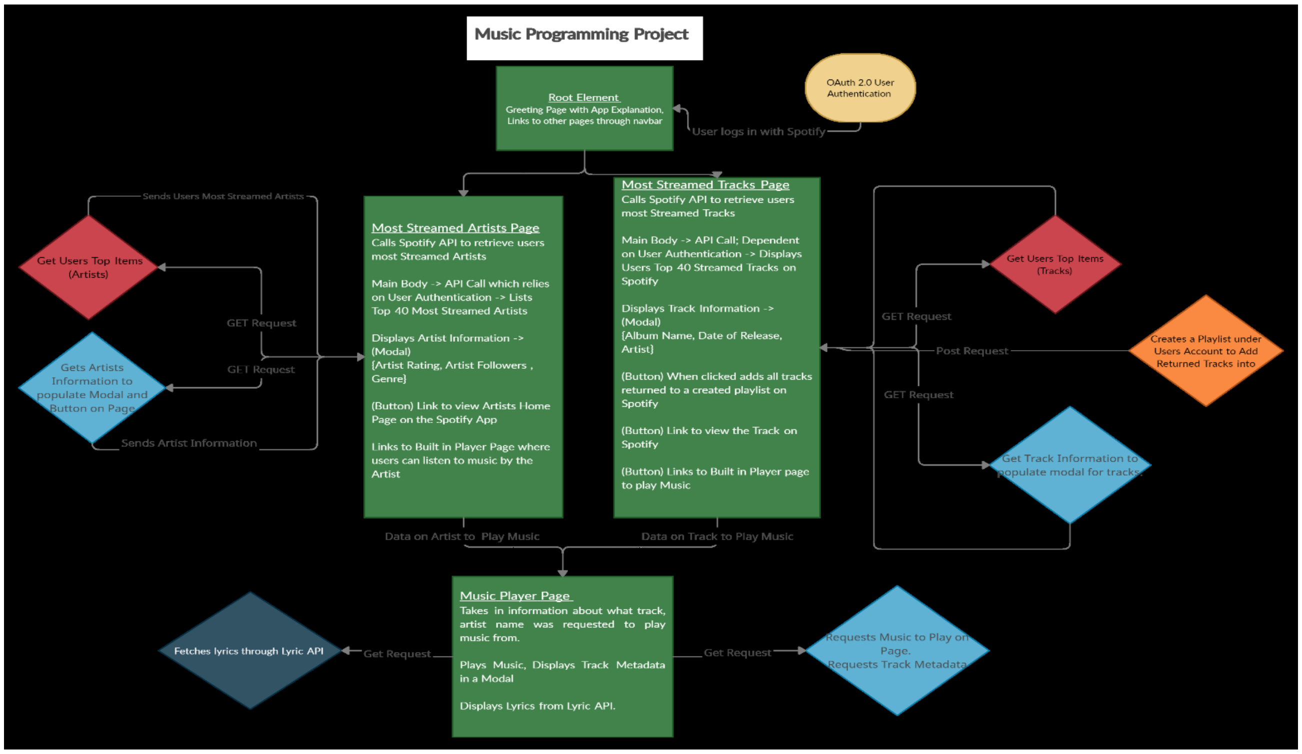


Example of the different customizable elements of the player components UI

UML Diagrams



Basic UML diagram created during the design phase of the project. This diagram was included in my first presentation.


Music Programming Project
Root Element
Greeting Page with App Explanation, Links to other pages through navbar
OAuth 2.0 User Authentication
User logs in with Spotify
Sends Users Most Streamed Artists
Get Users Top Items (Artists)
GET Request
GET Request
Gets Artists Information to populate Modal and Button on Page
Sends Artist Information
Most Streamed Artists Page
Calls Spotify API to retrieve users most Streamed Artists
Main Body -> API Call which relies on User Authentication -> Lists Top 40 Most Streamed Artists
Displays Artist Information -> (Modal) {Artist Rating, Artist Followers , Genre}
(Button) Link to view Artists Home Page on the Spotify App
Links to Built in Player Page where users can listen to music by the Artist
Most Streamed Tracks Page
Calls Spotify API to retrieve users most Streamed Tracks
Main Body -> API Call; Dependent on User Authentication -> Displays Users Top 40 Streamed Tracks on Spotify
Displays Track Information -> (Modal) {Album Name, Date of Release, Artist}
(Button) When clicked adds all tracks returned to a created playlist on Spotify
(Button) Link to view the Track on Spotify
(Button) Links to Built in Player page to play Music
Get Users Top Items (Tracks)
GET Request
Post Request
Creates a Playlist under Users Account to Add Returned Tracks into
GET Request
Get Track Information to populate modal for tracks.
Data on Artist to Play Music
Data on Track to Play Music
Music Player Page
Takes in information about what track, artist name was requested to play music from.
Plays Music, Displays Track Metadata in a Modal
Displays Lyrics from Lyric API.
Fetches lyrics through Lyric API
Get Request
Get Request
Requests Music to Play on Page.
Requests Track Metadata

UML Diagram, which was part of my second presentation, some minor changes to the project since then, but the final functionality if the same.

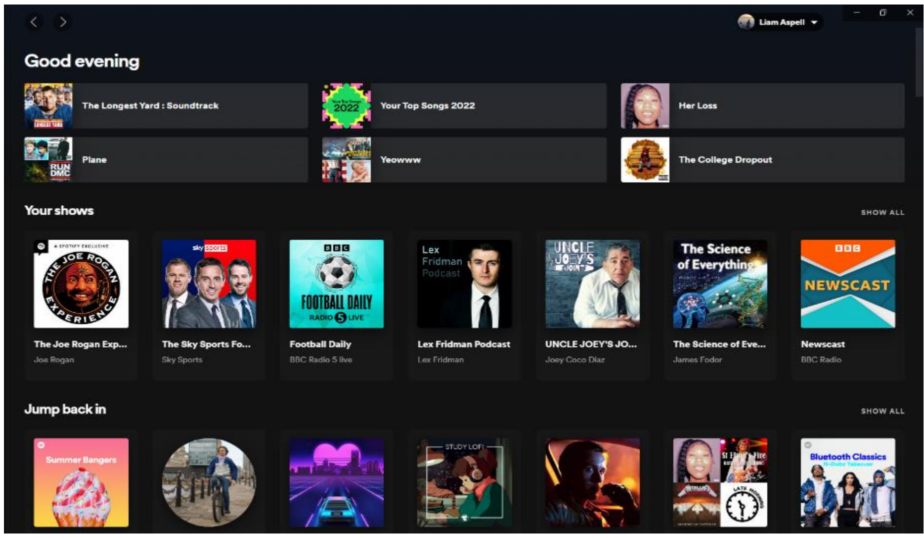
Conclusion

Successes / Shortcomings of the project

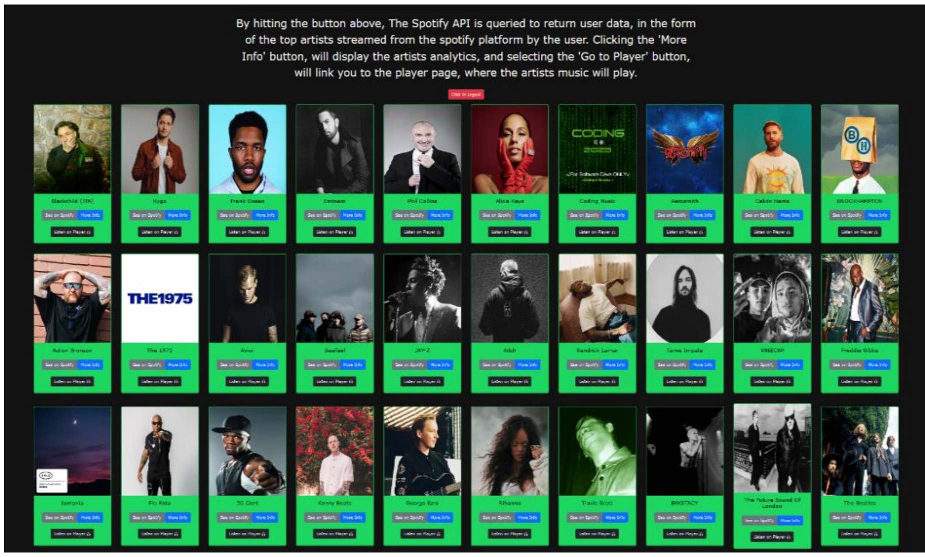
Successes

Front End Design:

Overall, I was happy with my implementation of the 'Retrieve Users Top Items (Top 40 Artists / Tracks)' API calls front-end design in which the data collected was represented in my application. This design was mentioned as feedback during my final demo of the project before the end of the semester. Taking this feedback on board, and acknowledging the need for a more compact display of the data, I decided that I wanted to design the front end in a manner which mimics a part of the design system in place within the Spotify Windows application, which made sense for my project, as that the overall scope of the project was to build a useful tool that complements and enhances the Spotify application. Designing the card system as a react component, to which the returned data would be stored



The UI in place in the current iteration of the Spotify Desktop Application



By hitting the button above, The Spotify API is queried to return user data, in the form of the top artists streamed from the spotify platform by the user. Clicking the 'More Info' button, will display the artists analytics, and selecting the 'Go to Player' button, will link you to the player page, where the artists music will play.

The UI of my Web Application

within really brought the project to life. Furthermore, I feel the modals built into the card system, the first being the modal containing unique information about the artist and the second dedicated to the player, both complement the card system very well too. Overall, upon completion of the project I am very happy with my front-end design.

API Calls with Spotify API and YouTube v3 API:

I had learned in depth about what APIs are and how they work over the course of an internship I took last summer, and initially wanted to create a project which would enable me to practice working with them, as they are an important part of software development. APIs can have a powerful impact on music-themed projects, as the data they can deliver in a short space of time is impressive, and applications such as my one can be kept light (do not require construction of a database). This idea is enhanced when personal information can be accessed as this provides a layer of personalization to any project. This is the initial reasoning why the particular set of (Users Top Items) API calls through Spotify stood out to me in the design phase. I felt I learned a good deal more about API's and how they interact with React web applications over the course of developing this project. This is one of the most valuable takeaways for me for the future, as I want to be involved in software engineering in some way when I finish my degree.

Shortcomings

User Interface Issues (Root element on all page's issue):

I encountered one issue on the front-end, where the specified root element, in my case the landing page of the web application, shows on all other pages within the application. I encountered this issue when adding a navbar with react-bootstrap and react-router-dom to my project. I carried out a large amount of research on why this was happening and concluded that the navbar I was using in my project was built for a previous version of react-router-dom (v5 whereas I was using v6 in my project).

As I wanted to keep this navbar in my project, I minimized the effect of this issue on my frontend by adding additional spacing between the div elements on each page and this root element, to 'hide' from view of the user. It wasn't a complete solution, but from my extensive research into the issue I could not find a solution where this issue would be completely rectified. From my testing there was no harmful effects to leaving the root elements in place at the bottom of my applications pages.

Could not get synced Lyrics working within the player modal:

At the beginning of the project, I included within the scope my intention to display lyrics to complement the built-in web player functionality. My initial intention was to pull lyric data using one of the many free lyric API's available online. I tried multiple times and many different methods to display the lyrics, trying API's such as musixmatch API, genius lyrics and lyrics.ovh. Unfortunately, I could not find a consistent method to display the lyrics that I was completely satisfied with. I decided to repurpose the space that was laid out for the lyrics, and after some consideration I decided that I would try to reach a midway solution, both for displaying lyrics, and for displaying an audio visualizer. I used YouTube API v3 and built a react component that would take two parameters, where I would pass the artist's name and track name into an API call and return the first video using YouTubes search algorithm. I then added the word 'lyrics' to the end of the query, and this seemed to work for most tracks with lyrics. I then applied the same logic to the artists section, but instead searched for an audio visualizer instead of song lyrics. As the Artists page plays random tracks from the artist each time, it would have been more difficult to capture a video containing lyrics for each individual song, so having the audio visualizer here worked better. As I had done some research into audio visualizers over the course of the project, and at one point decided to leave it out of the project due to the complexity of their implementation, I was happy to include one, even if it was in a more basic sense.

What I would do with more time

More customization of the 'Users Top Items' API Call:

The 'Users Top Items' API call was the call made to return the users to 40 artists/ tracks within the project. This call could be customized over the short (4 weeks), medium (6 months) and long terms (1 year). With more time to dedicate to the project, I would create a form in both the artists / tracks page which contains the three values of 4 weeks, 6 months and 1 year, and submitting the form with one of these values would update the API call in the code to retrieve data over the specified timeframe. This is a mechanism that would be straightforward to implement and would give the user more control over the data that they see, thus expanding the project further.

Add Top 40 Tracks into a Spotify Playlist:

In the tracks section of the project, I intended to create a mechanism to add the returned tracks into a playlist created on the authenticated users account on Spotify. This would involve two POST requests, the first to create a new playlist on the users account with a specific name, and the second to the list of tracks into a playlist. Although I began some work on this functionality, I ran out of time to complete this aspect of the project, and the need to put priority into other areas such as front end design and testing. If I had more time I would complete this functionality, as I feel it would be a useful feature which enhances users interaction with there Spotify account through using the project.

References

https://reactjs.org/

https://react-bootstrap.github.io/

https://www.statista.com/statistics/244995/number-of-paying-spotify-subscribers/

https://developer.spotify.com/documentation/web-api/

https://developer.spotify.com/console/get-current-user-top-artists-and-tracks/

https://www.npmjs.com/package/react-spotify-web-playback

https://developers.google.com/youtube/v3

https://www.youtube.com/watch?v=N34BM2CU_3g&ab_channel=QASCRIPT