

# Better Spotify Search

Anson Lu, Forrest Boyer, Kyran Chan, Tameron Honnellio, Will Shahbazian

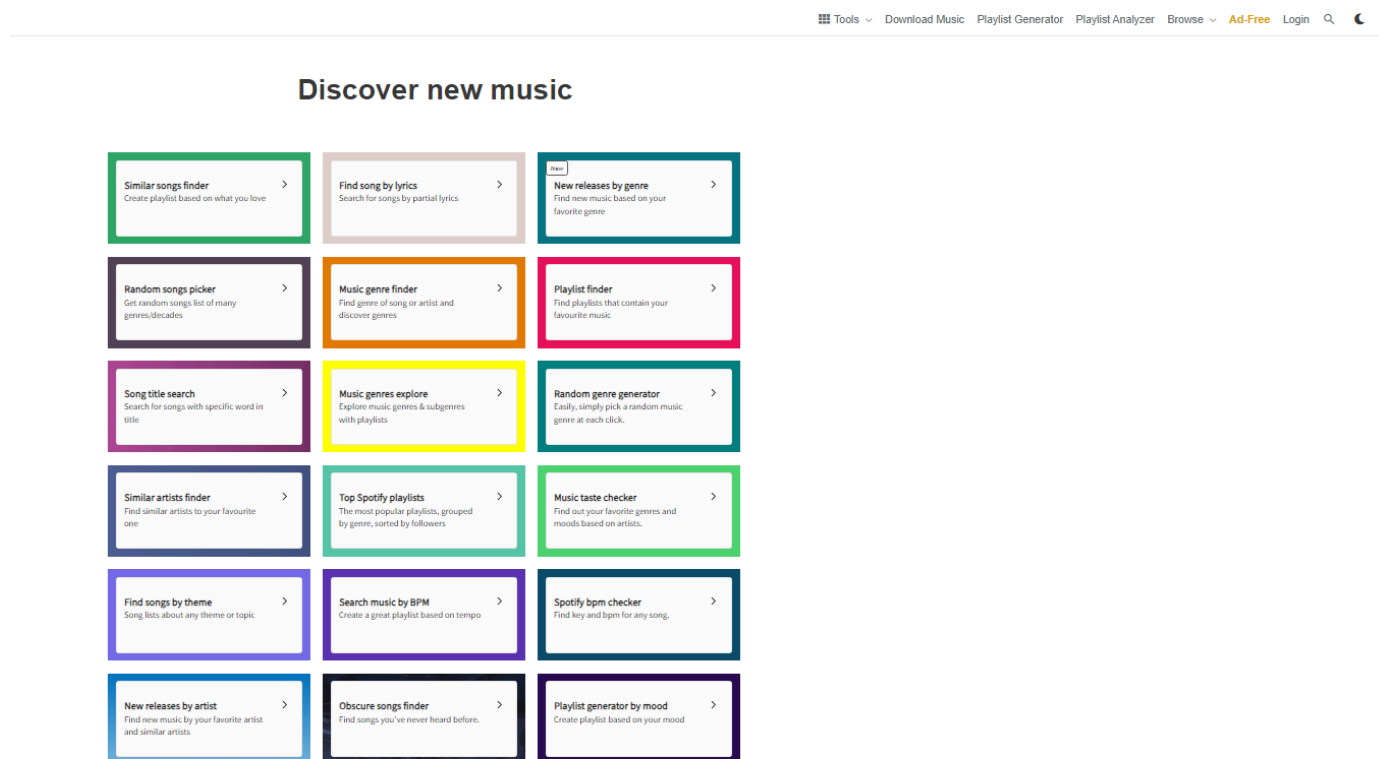
## 1 Motivation

Spotify search allows the user to look up the name of a song, an artist, an album, or a playlist that contains the song. Although usually sufficient for most, this is limiting when it comes to finding songs based on other information. A look into the Spotify Application Programming Interface, or API, reveals hidden characteristics that are stored for each song. Spotify tracks information such as danceability, energy, song key, loudness, acoustic value, valence, tempo, time signature, and more. Every single song on Spotify has this data associated with it. However, users are unable to search for this information directly. For example, a current Spotify search for off-signature songs, such as a “7/4-time signature”, yields no desirable results. A random song with the song name of “7/4” appears, but no songs are provided with a 7/4-time signature.

Therefore, our project will add these hidden features as additional options to the Spotify search query. Our project also allows users to search for hidden features on a song, search with a given song using its individual song features, as well as being able to search with standard user-input values. Because we have two different input methods for our frontend, our architecture was adjusted accordingly.

## 2 Other Instances

There are other applications for music searches similar to our project such as [Chosic](#) which has a multitude of different features such as similar song search, artist search, genre search, etc [10]. It also allows users to create playlists based on some of the resulting searches [10]. Another feature of Chosic is that it allows users to play songs from the app, while also providing song links for Apple Music, Amazon Music, Spotify, and YouTube [10]. However, this app has several drawbacks. Firstly, Chosic is full of ads, which makes it frustrating to use. The site will not allow users to browse with an ad blocker enabled, leaving the only way to combat the high ad density behind a paywall. Amidst the ads, the user experience provided by the app is also confusing to navigate. The homepage displays around 20 cluttered boxes for each different feature, with each having a very brief description of the intended functionality. Coupled with a navigation bar on the top of the screen, this makes navigation extremely confusing and intimidating.



*Figure 1. Chosic's Homepage*

On top of this, the search experience is poorly designed. For example, when searching for an instrumental song, a slider is displayed with vocals on one end and instrumental on the other. When adjusting the slider to the desired amount, there are no indicators to show the user the exact or approximate value they are selecting. Another key issue with this implementation is the lack of information about how each filter is modifying the resulting search. Users may not know what tempo or acoustic value means. This makes it difficult for the user to predict what they are searching for. Finally, the app does not contain some search filters such as time signature.

Many applications can also provide similar song statistics such as [SongData.io](https://songdata.io). This application allows users to input individual songs or playlists from Spotify and will output the statistics for the song or playlist [11]. This will also provide a short list of similar songs to the user. When searching with an individual song, SongData.io will display the values for each internal category representing the song stored within the Spotify API [11]. With a playlist, the app will display graphs for each category showing how many songs fall in certain intervals. For example, 13 songs in a playlist may fall below 30% loudness. SongData.io also provides some information about songs, and what different categories mean [11]. However, this application doesn't allow users to search using these parameters and is more targeted towards DJs or musicians who may want to analyze particular songs, instead of casual listeners.

[TuneBat](https://tunebat.com) is an application similar to SongData.io, giving users the ability to view the statistics of songs, artists, or albums, and displaying a list of similar songs to the input [12]. TuneBat also offers several other features, tailored to helping creators such as separating vocals and instrumentals from a song into different files, aid in mastering songs, a bpm tapper to allow

users to guess the bpm of songs, and an advanced song search [12]. The advanced song search is most similar to our proposal as it allows the user to search for songs based on Spotify’s hidden criteria. Users can both target a specific value for each criteria, or input a range for their search. In either case, a list of similar songs is provided to the user. However, TuneBat doesn’t display the full suite of options, such as valence and time signature [12]. Additionally, TuneBat is targeted towards DJs and musicians and is not very beginner-friendly for much the same reasons as SongData.io, as it leaves out some criteria and hides this functionality under an easily overlooked tag called “Advanced”. In the figure below showing Tunebat’s home page, you can see the “Advanced” tag next to the search feature. However there is no indication that this results in an enhanced search feature, other than its proximity to the search bar.

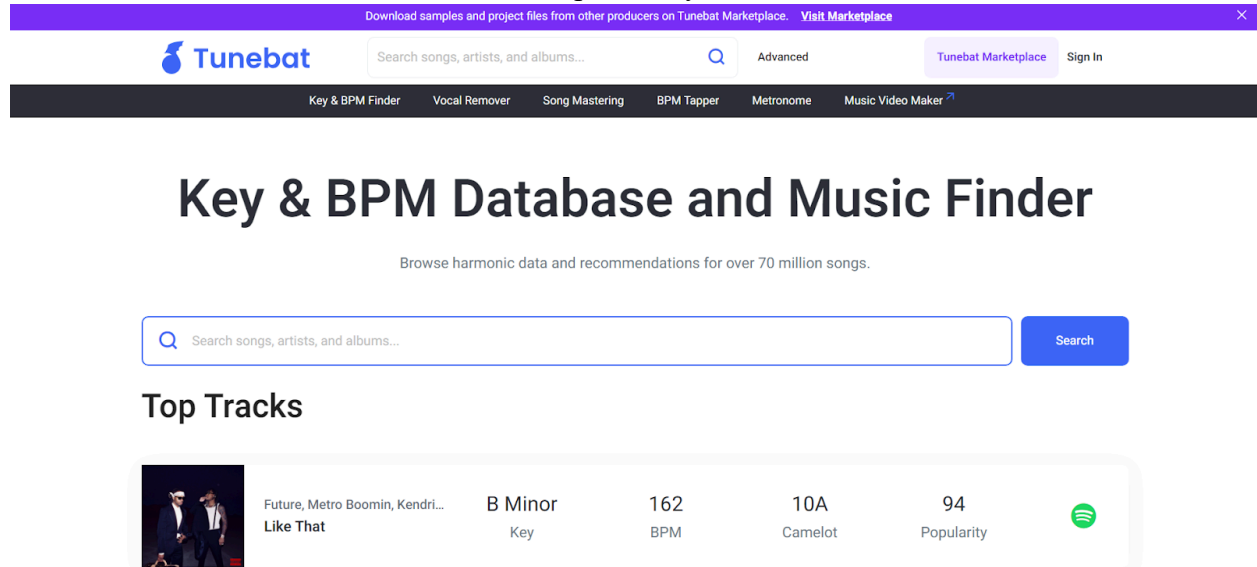


Figure 2. Tunebat’s Homepage

### 3 Methodology

Our project is a full-stack web application. It has been set up with a User-Experience frontend to interact with the user using Angular, and a flexible backend interacting with the Spotify API to send and receive requests using asp.net.

While there are applications that provide users with expanded Spotify search functionalities, none provide users with free, and easy to understand access to the powerful search capabilities that the Spotify API provides. Better Spotify Search provides users with advanced search criteria such as acoustic value, danceability, and energy, to more effectively find music they enjoy.

#### 3.1 Architecture and Technology

The design and architecture structure, shown in Figure 3, that the project uses is that of a full-stack web application. The project is hosted on GitHub for source control and continuous integration. The frontend is running on Angular, with the Jasmine testing framework and Karma task runner. The backend is using ASP.NET and xUnit as the testing framework. The frontend consists of pages for the user to use each of the search features. These pages make calls to our BFF (backend for frontend) API, which processes that information, makes calls to the Spotify API, gets necessary information about songs and users, and then sends that information back to

the frontend. As far as libraries used for the backend we are using SpotifyAPI-Net along with direct HTTP calls to interact with the Spotify API. There are two endpoints featured on this BFF one to receive user information for logging in to Spotify and a search endpoint. Unfortunately, we are not currently planning on implementing the login feature due to time constraints. The backend API is used for interacting with the frontend API but can also be used without the frontend to gather information if necessary.

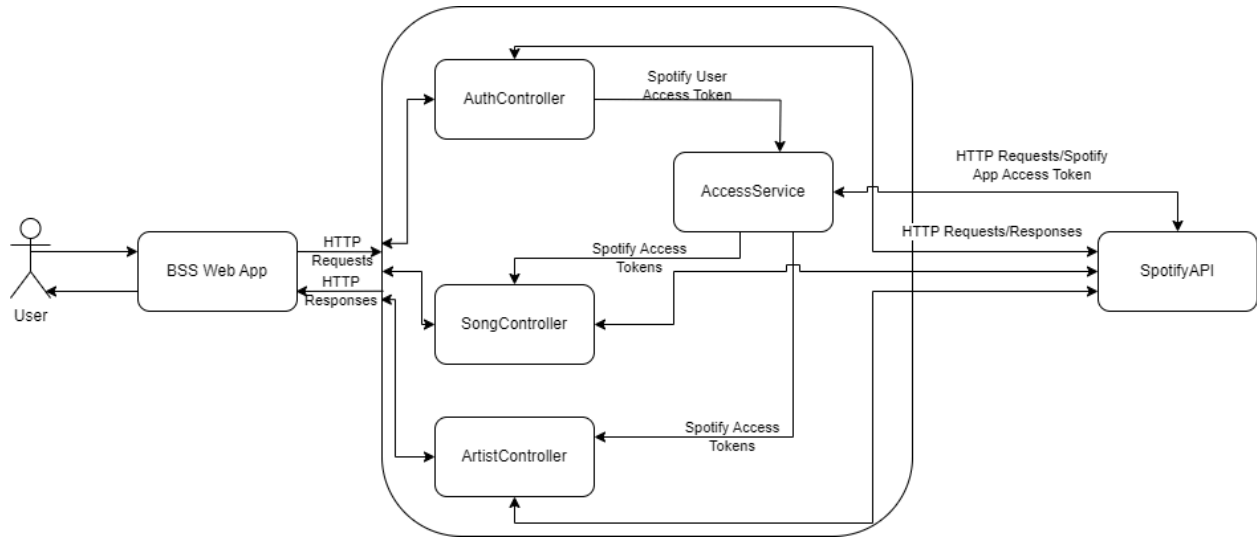


Figure 3. Architecture Diagram

## 4 Implementation

### 4.1 Backend Implementation

BetterSpotifySearch employs an API to interact with Spotify's own API. This was done so Spotify authorization and user data can be stored and handled separately from the primary web app. To do this, we used the ASP.NET Core framework as it is open source, relatively commonly used, and is written in C#; a language we are all familiar with using.

The API has three separate controllers, one for handling the user logging in to Spotify, one for handling song data, and lastly, one for handling artist data. In addition to the three controllers, the API has an access service to handle and update access tokens for the Spotify API.

There are currently five functional endpoints for the backend: one for the user to log in to their Spotify account, one for searching for artists by name, and three for searching for songs and their information. Currently, these endpoints can only be accessed by running BetterSpotifySearch locally and sending requests to `“../api/authenticate”`, `“../api/artist/{artistname}”`, `“../api/song/{songname}”`, `“../api/features/{songid}”`, and `“../api/searchby/{feature1, feature2, ...}”` respectively.

Both the song and artist search endpoints receive a string and return a list of their respective object and some basic data pertaining to them (e.g. popularity, images). The song feature endpoint receives a song's Spotify ID and returns that song's musical features (e.g. danceability, tempo, etc). Lastly, the song search by feature endpoint uses seed genre(s), song(s), and/or artist(s) alongside feature data to find a list of songs and their basic data.

## 4.2 Frontend Implementation

The frontend was implemented through Angular 13.0.1. Each unique page for every feature was created by generating components with Angular. Each component contains an html page, sass (.scss) file, and a typescript file. The html and sass files are responsible for visually displaying the page, and handling user input. The typescript file is responsible for handling the logic behind the webpage. This includes, making backend api calls, routing between pages, and sending information between pages.

When searching for songs, the frontend communicates with the backend through a series of interconnected steps. First the html file takes input from the user and stores it into variables within the typescript. Then when the user presses search, a typescript function is called which will generate the url needed to make the backend api call, call the backend endpoint for the specified feature, extract the necessary information (song name, song link, song id), store this information within a singleton, and then navigate to the results page. The results page will then retrieve the information from the singleton and perform similar logic to complete the search and display the requested information.

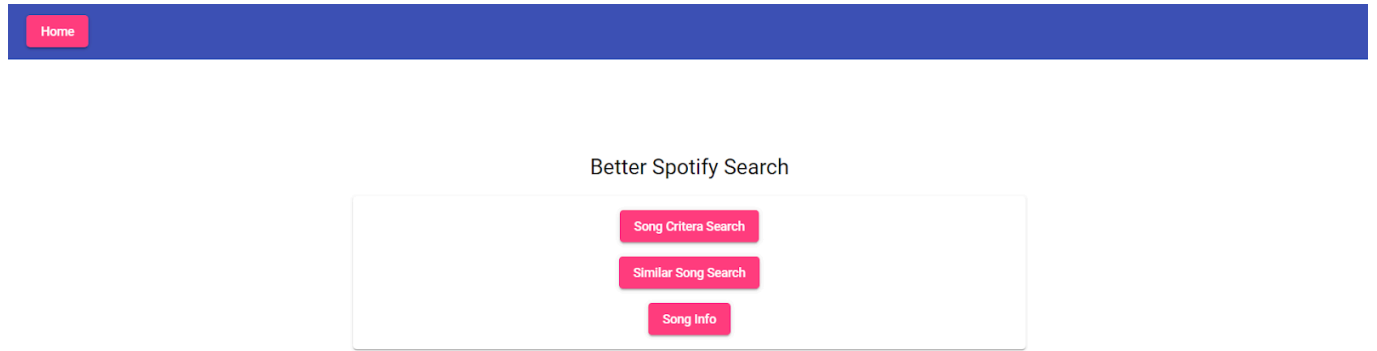
For example, the song info search works as follows. The song name is set from html keyboard input. Then once the search button is pressed, the typescript generates the url for the backend request by adding the song name to the end of the string, “api/song/search/”. Then the backend call is made and a string containing the top ten search results for that song name, along with information on each result is returned. This string is then converted to a JSON by calling `JSON.stringify()` and `JSON.parse()`. From here, the song name, id, and song link can be easily extracted and stored within a singleton. Finally the page is changed to the results page. Here, the typescript file for the results component retrieves the information stored in the singleton, and calls the backend song features request with “api/features/{songid}”. The returned string is then converted and parsed in the same way as above, and each song feature is set to its respective variable. Finally the html can access each typescript variable’s value and display them to the user.

## 5 Data and Results

Our results are as follows: the backend is feature complete and can reliably connect to and request information from the Spotify API. The frontend contains components for each feature and can read in information inputted by the user. However, we have currently only fully completed the “Song Info” search feature.

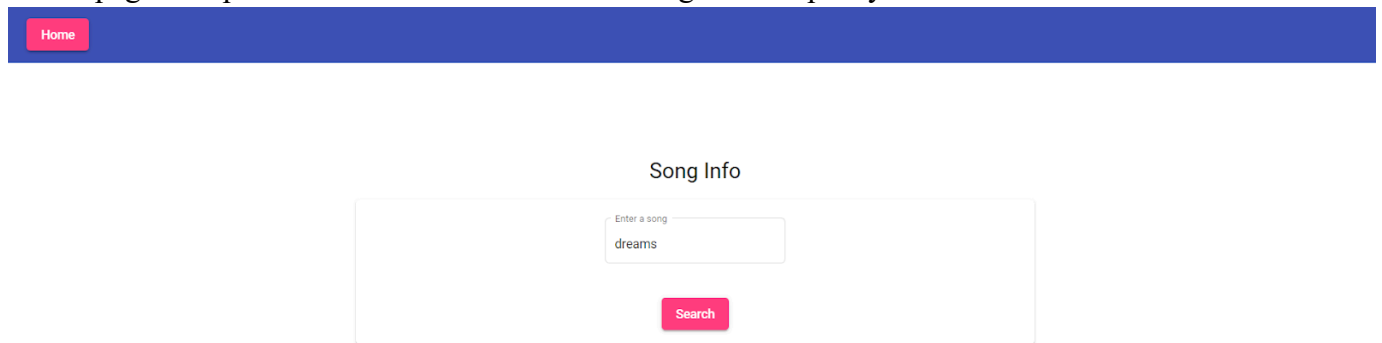
### 5.1 Frontend Interface

The frontend contains pages for the home, song criteria search, similar song search, and the song information features. Home is accessible from the home button via the navigation bar. From the home page, the user has three options: song info, song criteria search, and similar song search.

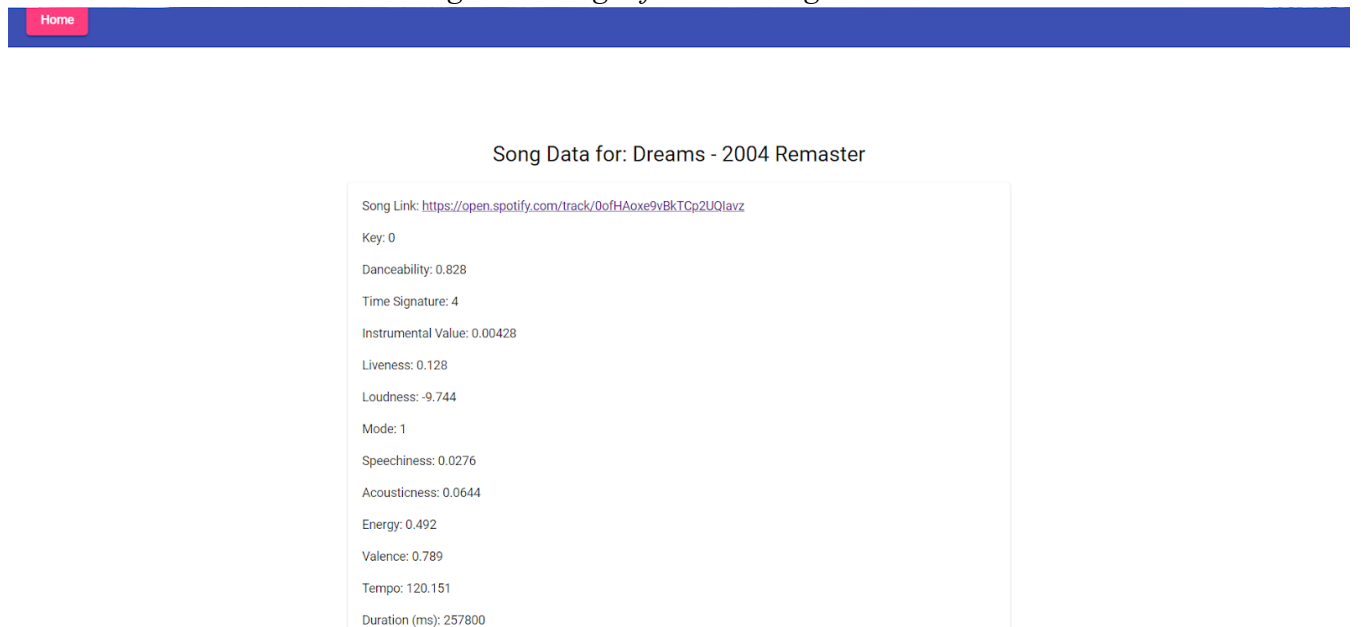


*Figure 4. Home Page*

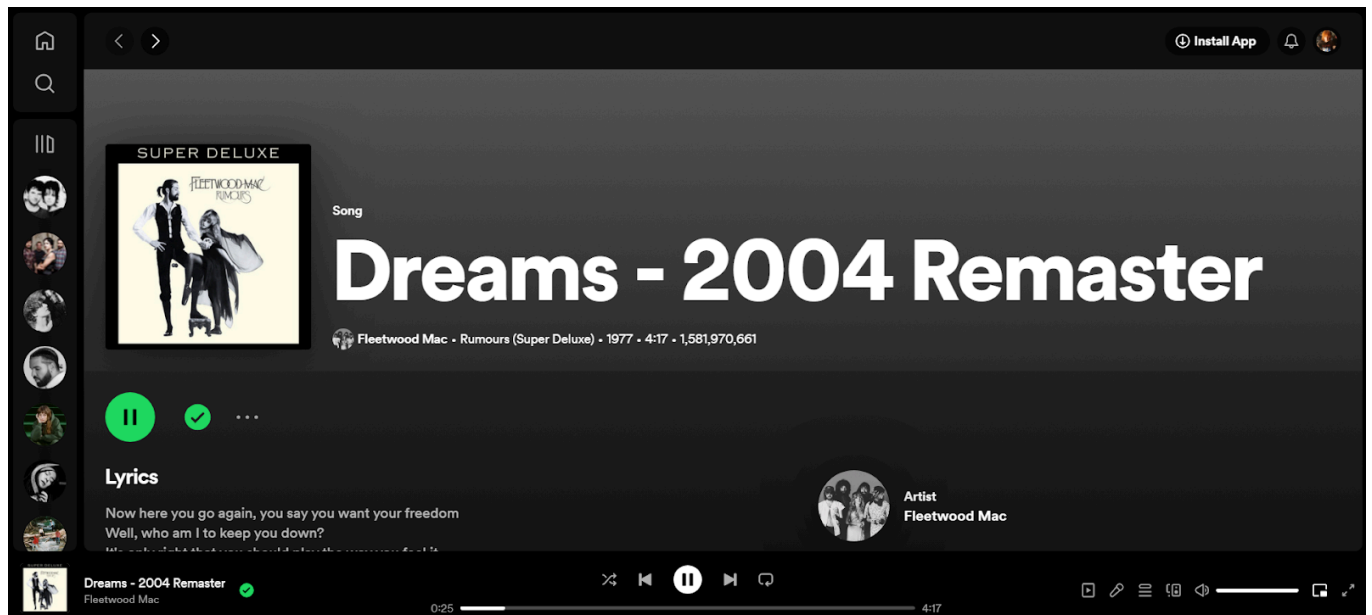
Song info prompts users to enter a song, and upon pressing the “Search” button a new page will display all of the hidden information Spotify has stored for the particular song. The results page also presents users with a link to the song on the Spotify website.



*Figure 5. Song Info search Page*



*Figure 6. Song Info Results*



*Figure 7. Song Info Results - Song Link*

Song criteria search displays a series of checkboxes and sliders for each searchable criteria. Users can check the boxes to enable certain criteria during the search. They may then proceed to adjust the value of each criteria through the use of a slider. When pressing the “Search” button, a new page containing the first song matching the search specifications will be displayed to the user. Currently, the backend implementation of this feature is complete. In the frontend, we still need to implement the search results page, as well as connecting the search to the backend API calls.

Criteria Search

☐ Duration

☐ Tempo

☐ Valence

☐ Energy

☐ Acousticness

☐ Speechiness

☐ Mode

☐ Loudness

☐ Liveliness

☐ Instrumental Value

☐ Time Signature

☐ Danceability

Search

*Figure 8. Criteria Search Page*

Finally, the similar song search feature allows users to enter a song, and select multiple checkboxes to tell the search to find songs similar to the specific value of the criteria within the input song. If no boxes are checked, then the search will use the hidden information from the input song to inform the search. Upon pressing the “Search” button a list of songs will be displayed to the user in the same format as the criteria search. This feature is implemented within the backend and in the frontend we need to implement the results page and connect it to the backend API calls.



### Similar Song Search

Enter a song

☐ Duration  
☐ Tempo  
☐ Valence  
☐ Energy  
☐ Acousticness  
☐ Speechiness  
☐ Mode  
☐ Loudness  
☐ Liveliness  
☐ Instrumental Value  
☐ Time Signature  
☐ Danceability

Figure 9. Similar Song Search Page

## 5.2 Project Use Case and Structure

Our project gives the user four distinct options to choose from, all of which perform a unique function and purpose. The following flowchart diagram displays the process pathways a user will travel through in our project.

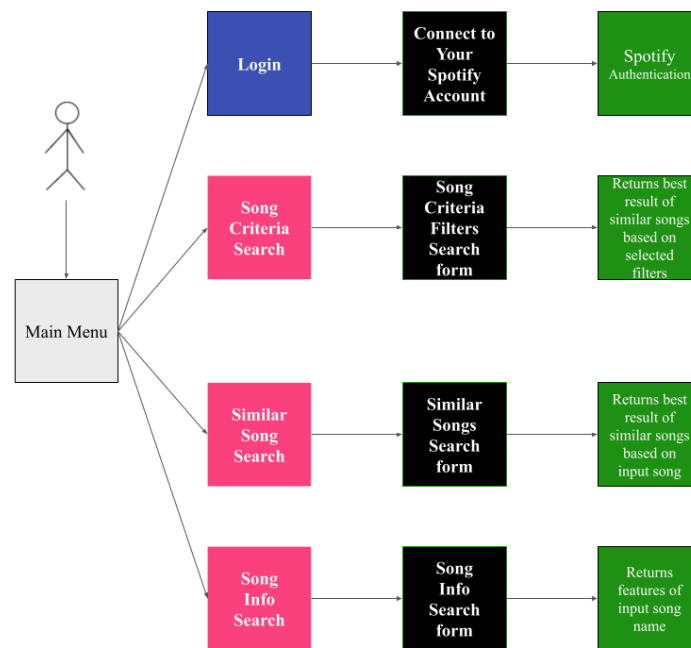
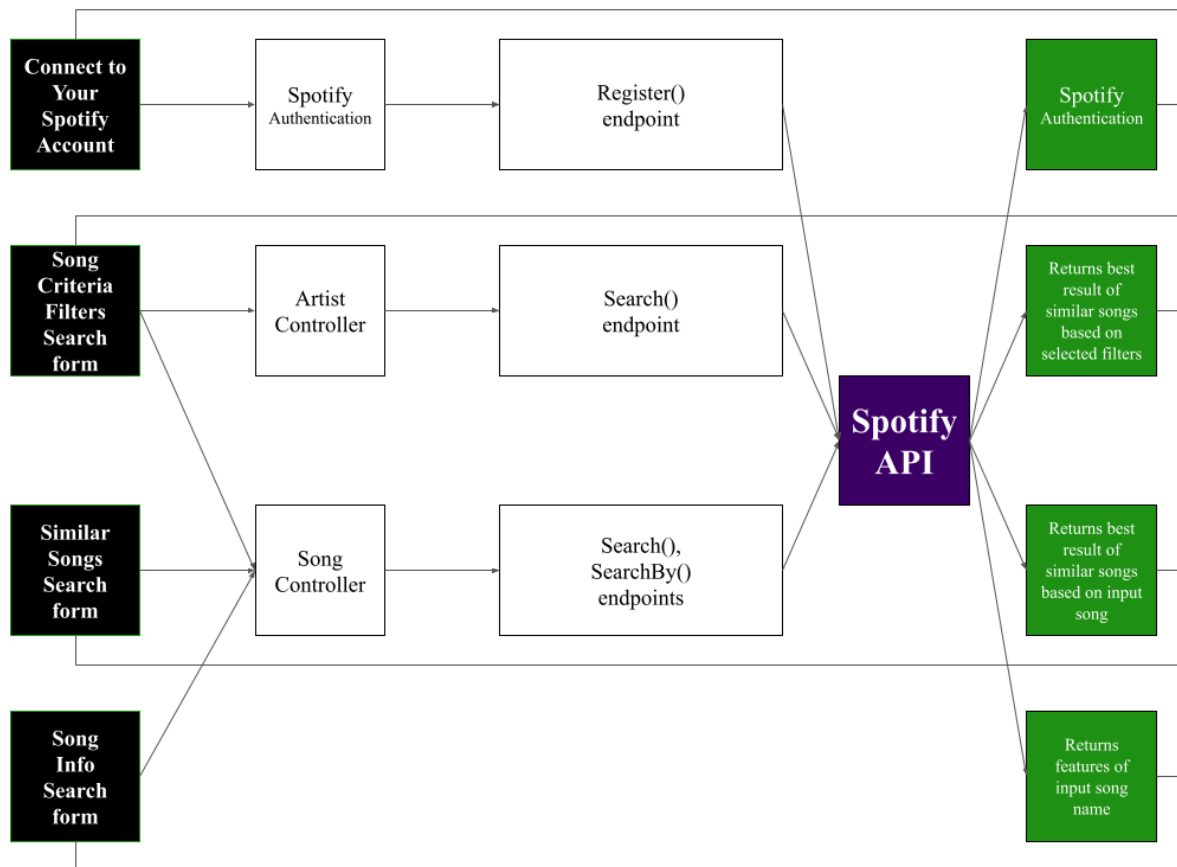


Figure 10. Application Flowchart Diagram - User's perspective

Behind the scenes, the internal components between the frontend forms, and the returned subject matter are connected like this:



*Figure 11. Application Flowchart Diagram - Behind The Scene Internals*

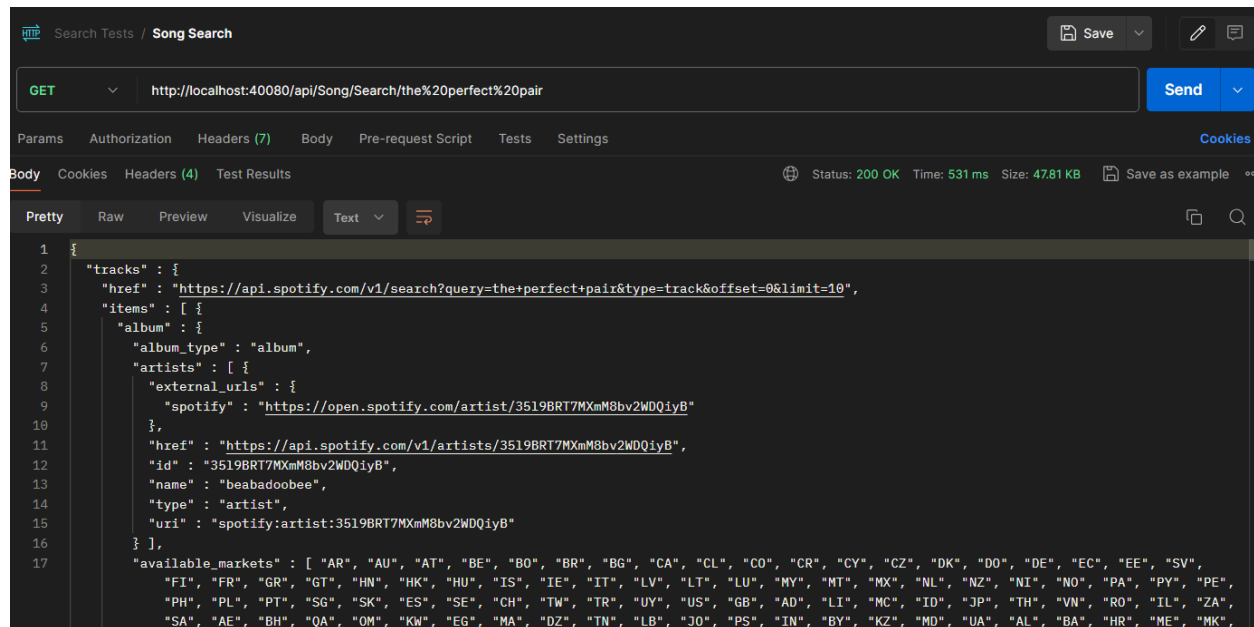
You will note the general flow from the frontend forms, to the controllers and endpoints, to the Spotify API, and returning the desired result back to the form. The return information from the API calls is passed to the frontend, and stored within typescript variables for the results page. From here, the data can be displayed within the HTML of the results page.

### 5.3 Testing

There are currently two testing suites to ensure the functionality of the BetterSpotifySearch app. The frontend tests ensure the functionality of each of the frontend components, and the backend tests individually examine each of the API endpoints and the access service. These tests run automatically when any commit is made to the main branch of the BetterSpotifySearch GitHub repository with the test history being fully visible. Additionally, the web app can be run locally using the `npm run start:all` or `npm run build:all` to also run all tests. From here the web app can be accessed through the browser to be manually tested, and the API can be sent requests using API testing platforms such as Postman (shown in the figures below) or Swagger.

## 5.4 Search Results

Below are several figures depicting the resulting information returned by the Spotify API from our backend:

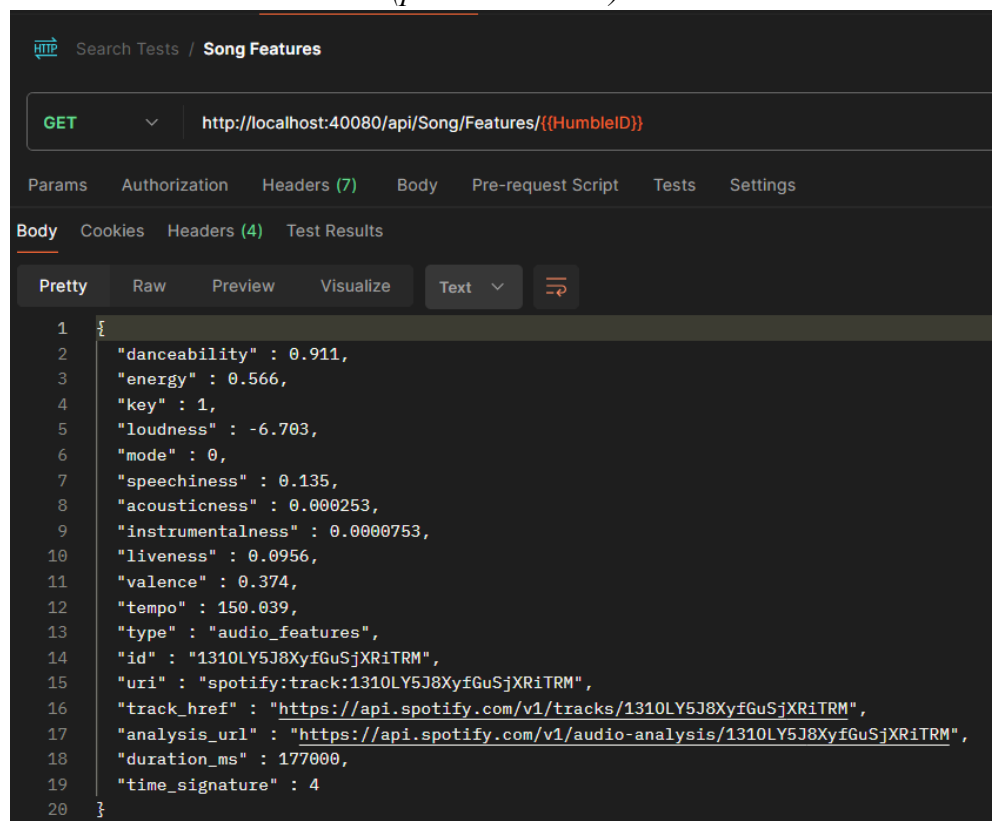


```

1 {
2   "tracks": {
3     "href": "https://api.spotify.com/v1/search?query=the+perfect+pair&type=track&offset=0&limit=10",
4     "items": [ {
5       "album": {
6         "album_type": "album",
7         "artists": [ {
8           "external_urls": {
9             "spotify": "https://open.spotify.com/artist/3519BRT7MXmM8bv2WDQiyB"
10          },
11          "href": "https://api.spotify.com/v1/artists/3519BRT7MXmM8bv2WDQiyB",
12          "id": "3519BRT7MXmM8bv2WDQiyB",
13          "name": "beabadoobee",
14          "type": "artist",
15          "uri": "spotify:artist:3519BRT7MXmM8bv2WDQiyB"
16        } ],
17        "available_markets": [ "AR", "AU", "AT", "BE", "BO", "BR", "BG", "CA", "CL", "CO", "CR", "CY", "CZ", "DK", "DO", "DE", "EC", "EE", "SV",
18          "FI", "FR", "GR", "GT", "HN", "HK", "HU", "IS", "IE", "IT", "LV", "LT", "LU", "MY", "MT", "MX", "NL", "NZ", "NI", "NO", "PA", "PY", "PE",
19          "PH", "PL", "PT", "SG", "SK", "ES", "SE", "CH", "TW", "TR", "UY", "US", "GB", "AD", "LI", "MC", "ID", "JP", "TH", "VN", "RO", "IL", "ZA",
20          "SA", "AE", "BH", "QA", "OM", "KW", "EG", "MA", "DZ", "TN", "LB", "JO", "PS", "IN", "BY", "KZ", "MD", "UA", "AL", "BA", "HR", "ME", "MK",

```

Figure 12. Part of the response to the Song Search Endpoint searching for “the perfect pair” (percent encoded)



```

1 {
2   "danceability" : 0.911,
3   "energy" : 0.566,
4   "key" : 1,
5   "loudness" : -6.703,
6   "mode" : 0,
7   "speechiness" : 0.135,
8   "acousticness" : 0.000253,
9   "instrumentalness" : 0.0000753,
10  "liveness" : 0.0956,
11  "valence" : 0.374,
12  "tempo" : 150.039,
13  "type" : "audio_features",
14  "id" : "1310LY5J8XyfGuSjXRiTRM",
15  "uri" : "spotify:track:1310LY5J8XyfGuSjXRiTRM",
16  "track_href" : "https://api.spotify.com/v1/tracks/1310LY5J8XyfGuSjXRiTRM",
17  "analysis_url" : "https://api.spotify.com/v1/audio-analysis/1310LY5J8XyfGuSjXRiTRM",
18  "duration_ms" : 177000,
19  "time_signature" : 4
20 }

```

Figure 13. The response from a request to the Song Features endpoint for the song Humble

```

1 {
2   "tracks": [ {
3     "album": {
4       "album_type": "ALBUM",
5       "artists": [ {
6         "external_urls": {
7           "spotify": "https://open.spotify.com/artist/1pPmIToKXyGdsCF6LmqLmI"
8         },
9         "href": "https://api.spotify.com/v1/artists/1pPmIToKXyGdsCF6LmqLmI",
10        "id": "1pPmIToKXyGdsCF6LmqLmI",
11        "name": "Rich The Kid",
12        "type": "artist",
13        "uri": "spotify:artist:1pPmIToKXyGdsCF6LmqLmI"
14      } ],
15      "available_markets": [ "AR", "AU", "AT", "BE", "BO", "BR", "BG", "CA", "CL", "CO", "CR", "CY", "CZ", "DK", "DO", "DE", "EC", "EE", "SV", "FI",
        "FR", "GR", "GT", "HN", "HK", "HU", "IS", "IE", "IT", "LV", "LT", "LU", "MY", "MT", "MX", "NL", "NZ", "NI", "NO", "PA", "PY", "PE", "PH",
        "PL", "PT", "SG", "SK", "ES", "SE", "CH", "TW", "TR", "UY", "US", "GB", "AD", "LI", "MC", "ID", "JP", "TH", "VN", "RO", "IL", "ZA", "SA",
        "AE", "BH", "QA", "OM", "KM", "EG", "MA", "DZ", "TN", "LB", "JO", "PS", "IN", "KZ", "MD", "UA", "AL", "BA", "HR", "ME", "MK", "RS", "SI",
        "KR", "BD", "PK", "LK", "GH", "KE", "NG", "TZ", "UG", "AG", "AM", "BS", "BB", "BZ", "BT", "BW", "BF", "CV", "CW", "DM", "FJ", "GM", "GE",
        "GD", "GW", "GY", "HT", "JM", "KI", "LS", "LR", "MW", "MV", "ML", "MH", "FM", "NA", "NR", "NE", "PW", "PG", "WS", "SM", "ST", "SN", "SC",

```

Figure 14. Part of the response from a request to the Song Search By endpoint, using many search criteria

```

1 {
2   "artists": {
3     "href": "https://api.spotify.com/v1/search?query=Queen&type=artist&offset=0&limit=10",
4     "items": [ {
5       "external_urls": {
6         "spotify": "https://open.spotify.com/artist/1dfeR4HaWDbWqFHLkxsg1d"
7       },
8       "followers": {
9         "href": "https://api.spotify.com/v1/artists/1dfeR4HaWDbWqFHLkxsg1d/followers",
10        "total": 84
11      },
12      "genres": [ "classic rock", "glam rock", "rock" ],
13      "href": "https://api.spotify.com/v1/artists/1dfeR4HaWDbWqFHLkxsg1d",
14      "id": "1dfeR4HaWDbWqFHLkxsg1d",
15      "images": [ {
16        "url": "https://i.scdn.co/image/ab67616d0000b273b8aa125a6123223b56e25c00"
17      }, {
18        "url": "https://i.scdn.co/image/ab67616d0000b273b8aa125a6123223b56e25c00"
19      }, {
20        "url": "https://i.scdn.co/image/ab67616d0000b273b8aa125a6123223b56e25c00"
21      }, {
22        "url": "https://i.scdn.co/image/ab67616d0000b273b8aa125a6123223b56e25c00"
23      }, {
24        "url": "https://i.scdn.co/image/ab67616d0000b273b8aa125a6123223b56e25c00"
25      }, {
26        "url": "https://i.scdn.co/image/ab67616d0000b273b8aa125a6123223b56e25c00"
27      }, {
28        "url": "https://i.scdn.co/image/ab67616d0000b273b8aa125a6123223b56e25c00"
29      }, {
30        "url": "https://i.scdn.co/image/ab67616d0000b273b8aa125a6123223b56e25c00"
31      } ],
32      "name": "Queen",
33      "popularity": 84,
34      "type": "artist",
35      "uri": "spotify:artist:1dfeR4HaWDbWqFHLkxsg1d"

```

Figure 15. Part of the response from a request to the Artist Search Endpoint for “Queen”

## 6 GitHub Repository and READMEs

Our current GitHub repository can be accessed here:

<https://github.com/ForrestBoyer/BetterSpotifySearch>

Our GitHub repository contains three separate READMEs:

- One for the project's home directory, intended as a reference guide containing essential information:  
<https://github.com/ForrestBoyer/BetterSpotifySearch/blob/main/README.md>
- A second for the project's frontend, containing information on how to build and run the frontend GUI locally:  
<https://github.com/ForrestBoyer/BetterSpotifySearch/tree/main/BetterSpotifySearch>
- And a third for the project's backend, containing information on the endpoints of the API:  
<https://github.com/ForrestBoyer/BetterSpotifySearch/tree/main/BetterSpotifySearchAPI>

## 7 Discussion

### 7.1 Current Functionality

We have fully implemented the Song Info search function. From the Home screen, clicking on “Song Info” leads to the Song Info screen. Enter in a song name, and the song information of that given song will be returned (assumption of a valid Spotify song). Using the song “Glitter”, the following figure shows the returned result, side-by-side with Spotify’s song “track” page:

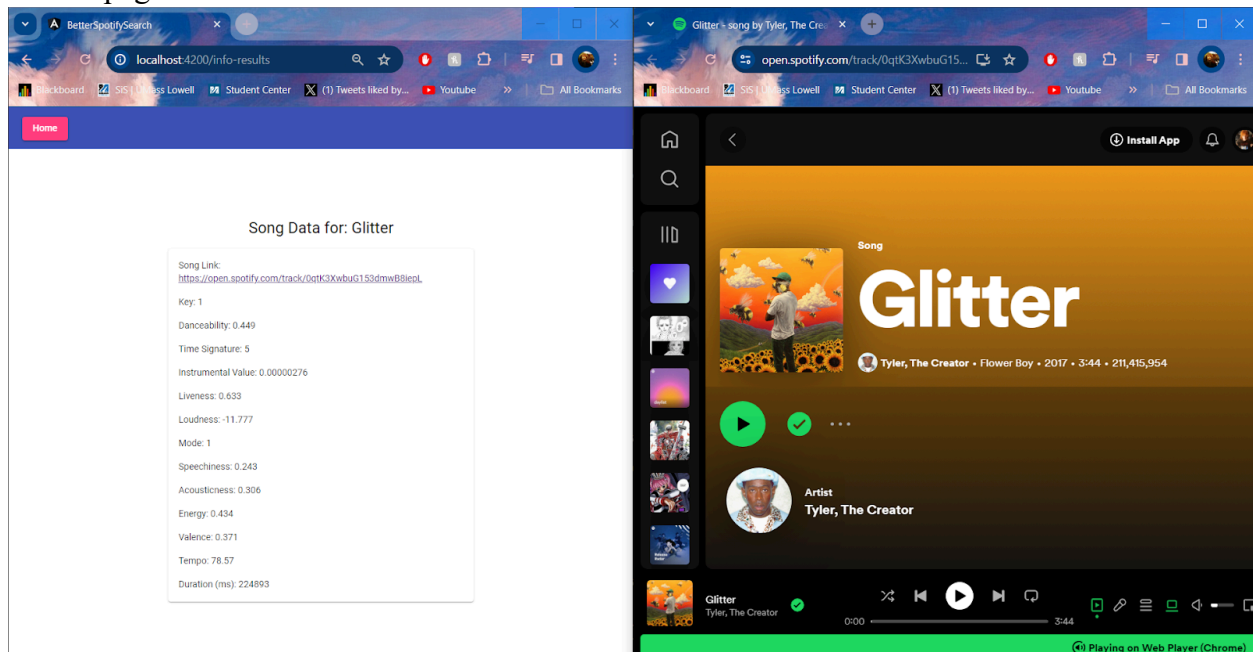


Figure 16. BSS Song Info search for “glitter” (left), song “Glitter” by artist “Tyler, The Creator” (right)

As you can see, the song data for the given song name is returned from the Spotify API and displayed in a list-type format.

## 7.2 Design Changes

As we continued to make progress on the project, we had to make decisions on omitting features that we initially had hoped to implement. For example, each search feature was originally supposed to display the top ten results, but we ended up making it so it just returned the best result.

We decided to omit the login feature as it does not currently make sense with the functionality we plan to have completed by the project deadline. The original goal was to allow users to log into their Spotify accounts, so that we could implement a feature allowing users to create a playlist from the search results, and add/save the playlist to their account. Another potential idea included being able to recommend new songs based on those currently within a given playlist, using the average song metrics of the current playlist. However, since we changed the search results to only display the top result to the user, it did not make sense to implement this feature.

Additionally, we wanted to add functionality for popup definitions for each search criteria, but could not implement it in time. We also wanted navigation to be accessible solely via keyboard but couldn't accomplish this either.

Lastly, we have deprioritized the website hosting of our project. We will still attempt to host with GitHub Actions toward the end.

## 7.3 Remaining Work

We still plan to implement the Similar Song search, and Song Criteria search functions. Currently, a temporary placeholder stub page exists in place of a functioning “returned results” page.

In the final build, pages similar to the returned results page of Song Info search will exist for Similar Song search and Song Criteria search. They will display the same features, along with the song name.

# 8 Challenges & Lessons Learned

Throughout the course of this project we encountered several challenges. The largest challenge was managing how much time we had available versus how much we wanted to accomplish. We had originally felt that this project would be relatively simple, thus allowing plenty of time for a feature dense final product.

Unfortunately, we spent a lot of time in the beginning of the project struggling to understand Angular 17.1. Forrest, who had previous experience with version 13.0.1, found that the tool had drastically changed from their previous understanding. This resulted in a couple weeks of lost development time as we had to change the entire project to use version 13.0.1 instead. The change in versions made setting up the basic frontend pages much easier. However, we found that much of the documentation for the older version 13.0.1 was inaccessible from the Angular website. As a result, we struggled to find the correct way to implement the communication between the frontend pages, and backend API calls. This led to roughly another week of slippage for our milestones.

Another key challenge we encountered during development was communication. It was often the case that we would underestimate the time required to complete a portion of the project. This meant that we ended up scrambling at the last minute to complete project deadlines. This could have been avoided if we had more effectively communicated with each other when progress was slow, and held longer team meetings more frequently.

When planning for this project, we discovered that Spotify places a limit on the number of API requests that can be made by a project. They allow 500 free requests per month and charge developers monthly for additional requests. We had originally planned to pay for a pro plan during development and switch to a free model after the completion of the project. However, we found that paying did not make sense, as we weren't close to making 500 requests during development, and we aren't planning on paying to publicly host the application on a website.

## Appendix

### 1 Definitions

(All range values use one decimal point of precision)

Acousticness: a measurement, from 0.0 to 1.0, describing how acoustic, or naturally sounding, a song is [3].

Danceability: a measurement, from 0.0 to 1.0, describing how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity [5].

Duration: the length of the track in milliseconds (ms) [5].

Energy: a measurement, from 0.0 to 1.0, representing the perceptual measure of intensity and activity [1].

Instrumental Value: a measurement, between 0.0 to 1.0, that predicts whether a track contains no vocals [5].

Key: a measurement, from -1.0 to 11.0, denoting where a song's notes fall within the major or minor scale in which the song operates in [5]. A value of -1.0 means that the song has no discernable key.

Liveliness: a measurement, from 0.0 to 1.0, describing the probability a song was recorded with a live audience [3].

Loudness: the overall loudness of a track, measurement in decibels (dB) [1].

Mode: a type of musical scale, from 0.0 to 1.0, used to describe a specific arrangement of whole and half steps, used to create a particular sound or mood [7].

Popularity: a measure, from 0.0 to 100.0, of how often a track is played [6].

Speechiness: a measurement, from 0.0 to 1.0, of the detection of the presence of spoken words in a track [5].

Tempo: the rate of speed of a musical track, calculated in beats-per-minute [2].

Time Signature: notations in sheet music that guide rhythmic structure of a piece [4].

Valence: a measurement, from 0.0 to 1.0, describing the musical positivity conveyed by a track [1]. If a song is musically positive, it elicits happy, euphoric, or cheerful feelings. If a song is not very musically positive, it elicits sad, depressing, or angry feelings.

## 2 Schedule

Week of	Goal	Done?
February 12th	Environments are set up on each group member's computer	yes
February 19th	Everyone is familiar with the tools that will be used	yes
February 26th	Basic interaction with the Spotify API can be performed	yes
March 4th	API requests can be created/formatted from provided parameters and API responses can be properly interpreted with both functions having tests. Midterm presentation started.	yes
March 11th	Frontend pages have been created for each feature. Midterm presentation completed before 3/13.	yes
March 18th	The Backend can provide the Frontend with the data received from a search request	no
March 25th	The Frontend displays the responses received by the Backend to the user.	no
April 1st	The product can connect directly to users' Spotify accounts.	no
April 8th	Completed draft of final report. The Backend can provide the Frontend with the data received from a search request, and the frontend can display the information received from the backend	yes
April 15th	The remaining two search features in the frontend are functionally complete. The product has been thoroughly polished and tested	



April 22nd	The final product is complete, including project final paper.	
------------	---	--

### 3 Responsibilities

Each team member's responsibilities during this project were distributed as follows:

- Forrest: Assisted the team with learning Angular, set up the project repository and continuous integration, and assisted with the backend and frontend.
- Tameron: Created project frontend pages for each feature, designed GUI for frontend, set up communication between frontend and backend, and assisted with documentation.
- Kyran: Created backend test cases, built a majority of the functionality within the backend, set up continuous integration, assisted with frontend and backend communication and assisted with documentation.
- Anson: Contributed by writing large portions of documentation paperwork, creating project diagrams and managing the group's progress.
- Will: Contributed by editing and revising documentation paperwork, as well as outlining responsibilities for each deliverable.

### 4 Feedback & Acknowledgements

We would like to thank Dr. Daly and our peers for aiding in the development of our project and this document.

#### 4.1 Project Proposal

Feedback we received for our first iteration of this document included: no figures were present for the proposed UI, what other sites have similar functionality, and how we will handle account information. We addressed this feedback by creating multiple UI mockup diagrams for the key proposed features, and discussing three other sites with similar functionality. We are currently undecided on how we will handle Spotify account information.

#### 4.2 Project Architecture

Feedback we received on our second iteration of this document included: citing Spotify, adjusting heading and title sizing, providing more context on how applications like Chosic, SongData.io, and Tunebat function along with discussing their disadvantages. We also received several requests to merge the design/deliverables section into the approach section. There were several people who suggested moving some information around to increase readability. For example, we moved an example from objective to motivation, where it made more sense. We

also decided to move other instances below motivation to help with the logical ordering of the document. We have added in a definitions section to help readers understand what each search criteria means. We added in a brief discussion of the advantages of our proposal over the other instances. Finally we adjusted our proposal according to spell checking, and added more information about potential legal risks.

### 4.3 Project Initial Results

Feedback we received on our initial results iteration of this document included: removing our cover page header, including our project name in the title, adding figures showing our competitor's implementations, adjusting our wording from what we were planning to accomplish to what we have accomplished, adjusting figure sizing to remove excess white space, adjusting our api implementation section, fixing inconsistent wording, and addressing work we have completed or cut from the project. We have addressed this feedback for this iteration of the document.

## References

- [1] M. Aldrete. “Spotify Music Moods.”  
<https://rpubs.com/mary18/860196#:~:text=Valence%3A%20A%20measure%20from%200.0,sad%2C%20depressed%2C%20angry>. (Accessed 21 Feb. 2024).
- [2] “tempo.”  
<https://www.merriam-webster.com/dictionary/tempo#:~:text=1,of%20motion%20or%20a%20activity%20%3A%20pace> (Accessed 21 Feb. 2024).
- [3] J. De Dios Santos. “Is my Spotify music boring? An analysis involving music, data, and machine learning.”  
<https://towardsdatascience.com/is-my-spotify-music-boring-an-analysis-involving-music-data-and-machine-learning-47550ae931de> (Accessed 21 Feb. 2024).
- [4] E. Bond. “The basics of time signatures: a beginner’s guide.”  
<https://www.skoove.com/blog/time-signatures-explained/> (Accessed 21 Feb. 2024).
- [5] “Web API • References / Tracks / Get Track's Audio Features.”  
<https://developer.spotify.com/documentation/web-api/reference/get-audio-features> (Accessed 21 Feb. 2024).
- [6] “Spotify Popularity Index: A Little Secret to Help You Leverage the Algorithm.”  
<https://www.loudlab.org/blog/spotify-popularity-leverage-algorithm/> (Accessed 21 Feb. 2024).
- [7] “Web API • References / Tracks / Get Recommendations.”  
<https://developer.spotify.com/documentation/web-api/reference/get-recommendations> (Accessed 21 Feb. 2024).
- [8] Spotify. <https://open.spotify.com/> (Accessed 20 Feb. 2024).
- [9] M. Iqbal, “Spotify Revenue and Usage Statistics (2024).”  
<https://www.businessofapps.com/data/spotify-statistics/> (accessed Feb. 7, 2024).
- [10] chosic.com. <https://www.chosic.com/> (accessed Feb. 7, 2024).
- [11] songdata.io. <https://songdata.io/bpm-key-finder> (accessed Feb. 7, 2024).
- [12] tunebat.com. <https://tunebat.com/Advanced> (accessed Feb. 7, 2024).
- [13] “Spotify for Developers / Design Guidelines.”  
<https://developer.spotify.com/documentation/design> (accessed Feb. 7, 2024).