

Senior Project
music, colors

Laya Paladugu, Lipi Paladugu
Advisor: Geoffrey Towell

Submitted in Partial Fulfillment of the Requirements of the BA in Computer Science
Bryn Mawr College



Spring 2021

Abstract

Our app, *music, colors*, aims to enhance music discovery through colors. Our app allows users to select three colors and in return it plays a song that corresponds with the mood associated with the colors. We converted HSL color values to numerical musical qualifiers such as valence, danceability and energy to translate the mood of the colors chosen to a song. We used the Spotify API to implement our project and Glitch to host it. Our evaluation found that majority of the users found that the songs returned matched the mood of the colors they selected. Since music and colors are both subjective topics, these results were promising. The users also found the website to be aesthetically pleasing. Therefore, we satisfied our initial goal of creating a music listening platform that positively stimulates a user's visual and auditory experience.

Acknowledgments

Laya

I want to thank my professors Geoffery Towell and Chris Murphy for advising me throughout my thesis work. I want to thank my coach, Rebecca Tyler, for constantly supporting me on and off the court. I also want to thank my friends for being there for me these past few years. I am also grateful for my parents, grandparents and Loki who have always shown unconditional love and support. Finally, I want to thank my sister, Lipi. I am very thankful we got to share this experience and so many more throughout life.

Lipi

First, I want to thank my sister for being my partner in this project and my biggest supporter for 22 years. I would also like to thank god, my parents, my grandparents, my friends, and Loki. Finally, I'm grateful to Professor Towell, Professor Murphy, Coach Tyler and everyone at Bryn Mawr who's helped me over the last 4 years.

Table of Contents

1. Introduction	7
2. Solution (Approach, Features, Requirements)	7
2.1 App Features	7
2.2 Quantify Songs	9
2.3 Quantify Colors	9
2.4 Relating Colors and Music	10
2.5 Legal Considerations	11
2.6 Accessibility	11
3. Design	11
3.1 Spotify	11
3.1.1 Spotify Web Playback SDK	11
3.1.2 Spotify Web API	12
3.2 Glitch	12
3.2.1 JavaScript, HTML and CSS	12
3.3 JSON Data File	12
4. Implementation	13
4.1 Data	13
4.2 Spotify Web Playback SDK	14
4.2.1 Spotify App Authentication	14
4.2.2 Spotify User Authentication	15
4.2.3 Using the SDK	16
4.3 Playing a Track	17
4.3.1 Colors	17
4.3.2 Convert Data File	18
4.3.3 Picking a Song	18
4.3.4 Play a Song	19
4.3.5 Changing a Track	19
4.4 Spotify Web API	19
4.5 UI	19
5. Evaluation	21
5.1 Results	21
6. Related Work	22
7. Conclusion	23
7.1 Future Work	23
7.2 Reflection	23
8. References	24

1. Introduction

The goal of our project is to combine music discovery and visual aesthetics to enhance the music listening experience. Many existing forms of music discovery rely on genre tagging, keywords, and other explicit searches. While these are effective methods for selecting music, they rely on the listener knowing what they want, and on the listener being able to use words to describe their search. This can often narrow the range of the user's discovery as users are limited to what they already know when searching for something new. Similarly, music discovery that relies on the listener's past listening also restricts the range of music the listener is being exposed to.

Music is innately tied to emotions. People are moved by music, and songs can be used to set the right mood or to create a powerful moment. This, however, is what makes songs so hard to describe. There are an infinite number of emotional ranges and people often find it difficult to put their emotions into words, let alone the emotions of a song. Therefore, we believe it is important to find new ways to describe and search for music. Music can be a source of comfort, joy, and so much more. We want to find ways to make listening pleasant and music discovery easy.

By including a visual aspect to the music search and listening experience, a user has an alternative way to explore music. Music is primarily an auditory experience, but we know from album artworks and music videos that a visual component to music can deepen the impact of a song. Generally, songs precede these album covers and music videos, but we want to use something visual to find a song that matches the vibe of the visual element. We have chosen to use colors as a medium to search for and listen to music.

Colors have deeply encoded emotional meanings, so this project explores the relationship between the sentiments a color has and the sentiments a song has. Popular color theory is a result of the shared experiences humans have had as well as the conditioning and implementation of this theory through visual media like movies, books, and art.

The challenging part of this project is that it attempts to link the emotions of these colors and songs. Both songs and colors are perceived subjectively and independently, so in order to form a connection between the two, we will have to find a viable link. If we can successfully form a bridge between music and colors, music lovers will have a fresh experience to do what they already love. This will benefit many- from people who don't quite have the vocabulary to describe what they're feeling to seasoned music veterans, looking for something new to add to their precious playlists.

To deliver our solution, we have created a web app for Spotify users that takes colors from users and uses color theory as well as various song characteristics to suggest and play songs for the user from a refined list of the Spotify music library. While our application is usable by all Spotify users, our target audience is primarily individuals between the ages of 18-25. Our solution will include a unique algorithm that converts colors into values comparable to the quantitative song characteristics in the Spotify API. Since we are using an established platform like Spotify, we can also access immediate and secure playback on the user's device.

2. Solution (Approach, Features, Requirements)

2.1 App Features

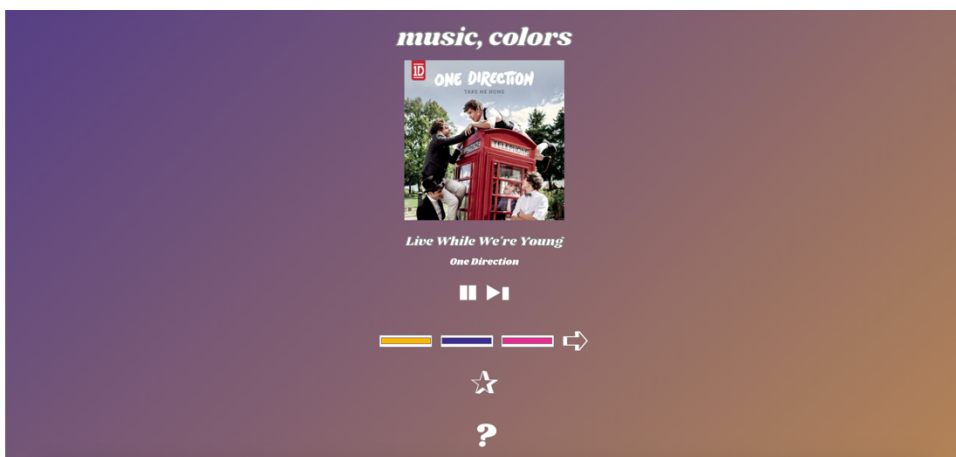
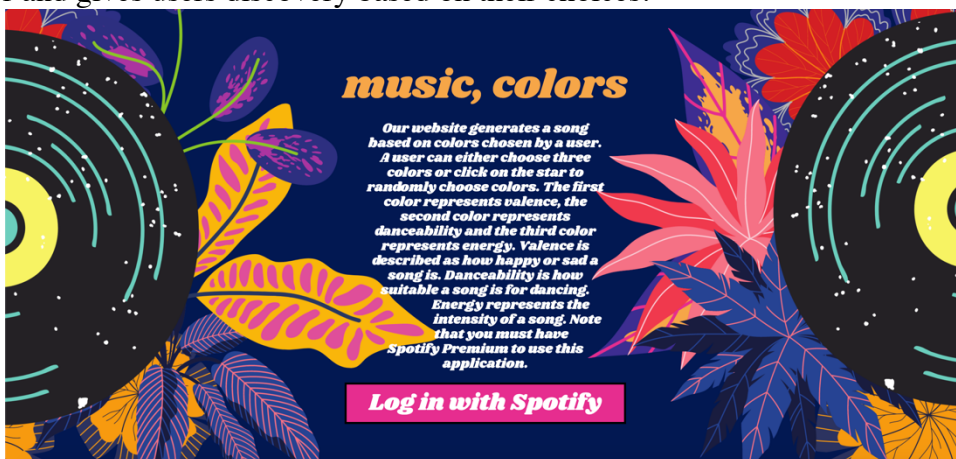
Our web app aims to be both minimalistic and visually satisfying. We start with a home page that includes information about what our website does. Then after the user successfully logs in, the user will have access to our playing screen. It includes three separate color pickers. When

a user submits their colors, the webpage begins playing a song that “matches” the colors. While playing a track, the page includes the album artwork, title and artist of the song. In addition, there is a “random” button that a user can use to generate random colors and a song that corresponds to those randomly chosen colors also plays. We also incorporated an animated background that includes the colors that were used to generate the current song. This background changes when the user submits a request for a new song. Additionally, we include both pause/play and skip buttons to enable playback right from our website. Finally, we have included a help button to explain the functionality of the elements on our page.

By keeping the focus on just the colors and music, we have created a place where the user isn’t distracted by too many moving pieces and can focus on the music itself. Having a platform that’s visually appealing will also make the user want to spend more time with the app. As for music discovery, we used an extensive library of songs that we filtered by popularity and time period so that we produce contemporary music that has an established listening.

In order to play music, we used a music streaming service that gave us access to all the track information we would need. Music streaming sites dominate the music industry today and include extensive libraries so it would give us a wide range of music to work with.

We believe that this is a good solution to our project because it incorporates the visual search components that we wanted and gives the user a music discovery channel that doesn’t rely on their past listening or linguistic searches. This search links the emotions in songs and music together and gives users discovery based on their choices.



2.2 Quantify Songs

We used a dataset that includes characteristics for each track and quantifies qualitative aspects of a song. Each song is labeled on a scale of 0.0 - 1.0 for characteristics such as acousticness, danceability, energy, instrumentalness, liveness, speechiness, tempo and valence. For our purpose, we have chosen to use valence, danceability, and energy. The following table includes a description of each of these.

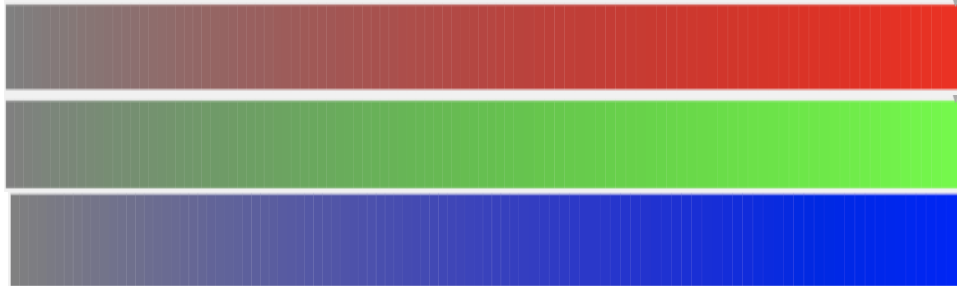
Feature	Definition
valence	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
danceability	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
energy	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

We chose these features because we believe they cover the most comprehensive emotion-related data. Some features like instrumentalness and speechiness rely more on the vocal presence in the song and this wouldn't give us an indication of the emotions of the song. Additionally, a feature like danceability includes tempo in its calculation so it's a broader option. Energy and valence relate directly to mood as well and include a wide range of emotions on each end. Overall, all three of the features are similar enough that they can be compared to each other on the same scale, and yet include enough variations to give a diverse range of results.

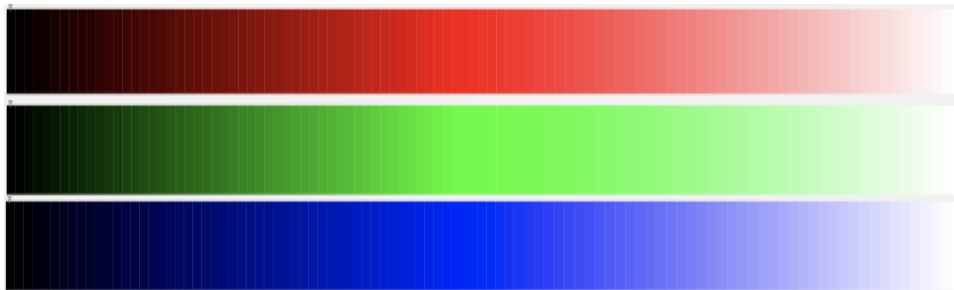
2.3 Quantify Colors

Colors have been numerated in multiple formats. The most popular formats include RGB, HEX, and HSL. We will use HSL for this project because it provides us with the most workable scales in relation to this project. HSL stands for Hue, Saturation, and Lightness. Hue, the general gradation in color, exists on a 360° wheel, where 0° is true red, 120° is true green and 240° is true blue. Saturation refers to the "grayness" of a color where 0% means a shade of gray and 100% means full color. Finally, lightness is also on a percentage scale where 0% is the color black and 100% is the color white. Since we need to be able to compare the HSL values to the track feature values, we need to use an algorithm to convert these values to a range of 0.0-1.0. Saturation and lightness are easily convertible to decimal values, so we only needed to manipulate hue values. This is one of the reasons we chose HSL as it would require the least amount of manipulation.

Since saturation indicates that the fullest form of a color is at 100%, we can say that a “fuller” color is comparable to more positive emotions, while a color that lower in saturation can be related to more negative emotions. Below is an example of what true red, blue and green look like at on a saturation scale from 0% to 100%. Lightness is set at 50% because, as we will explain next, it captures the fullest color.



Adjacently, lightness preserves most of its color in the middle third of the scale. The bottom third of the scale, where most of the shades of black exist indicate darkness or negative emotions, while the top third of the scale includes the palest forms of colors. While this isn't as negative as the dark shades it doesn't correlate to as much energy as the colors in the middle of the lightness spectrum have either. The diagram below shows the changes in lightness for true red, green and blue, with the saturation set to 100%.



To break up hue, we divide the colors based on what we interpret as positive or negative from our understanding of colors. Based on this, our scale of positive to negative colors is: yellows, oranges, reds and pinks, purples, greens, blues. By combining this with the logic we used to decode saturation and lightness, we can classify HSL colors into an emotional scale. All our track features will use the same scale.

2.4 Relating Colors and Music

Once the colors and the music are matched on a scale of 0.0-1.0 as described, it becomes a matter of pairing each color sent in by a user to each of the track features. We have chosen to map the first color to valence, the second to danceability and the third to energy. This means that when we choose a song to play for the user, it will have the valence of the first color, danceability of the second color and the energy of the third color. Generally, this means that a color with 0% saturation, 50% lightness, and with warmer hues (yellow, orange, red) will give us the most positive valence, and highest danceability and energy. Conversely, cooler colors (blues, greens, darker purples) and lower saturation and lightness will give us negative values with lower energy and danceability. For example, if a user were to submit a bright yellow, darker blue, and orange, then we can expect a song that is extremely happy and has a lot of energy but isn't necessarily a song you would dance too. This color mapping of cooler and warmer colors comes from the way in which we experience these colors in our lives.

Each color picker will have a tooltip that we show which color is being mapped to which value. While we want the focus to remain on the visual experience of picking colors, we want to give users a sense of understanding behind the code as well. The user can adjust the colors they are picking to get a song more akin to their taste. We chose three colors and features because this narrows the songs in the library to enough choices. If we were to include more colors, it would become more tiresome for a user to pick colors each time and if we had more track features it would be hard to find a song that matched perfectly with all the colors.

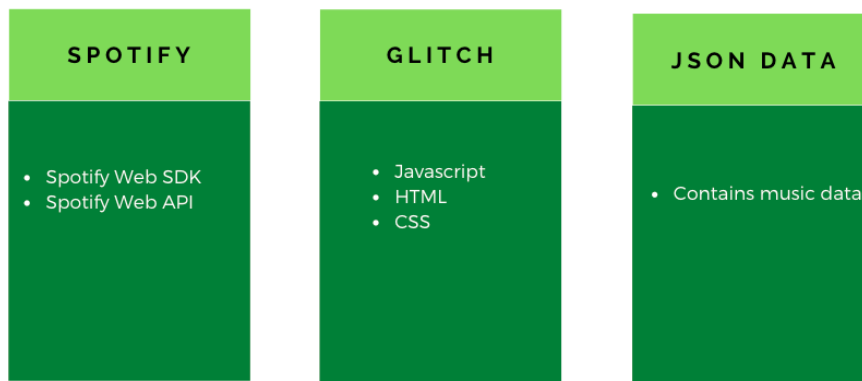
2.5 Legal Considerations

In order to play music, we need to have EME protection. EME or Encrypted Media Extensions allows encrypted media, such as audio, to be played over a webpage (Dutton, 2021). To ensure that we can play music, we need to use a server that provides this protection for playing music. Furthermore, when using a streaming sites library, we need to abide by it's policies as well. This includes login and authorization capabilities that protect it's users.

2.6 Accessibility

In order to make our service more accessible to people who are color blind, we accept color inputs as RGB, HEX or HSL numeric values. We have also included tooltips and help functions to provide a clearer understanding of our site.

3. Design



3.1 Spotify

In order to execute this goal, we will be using Spotify's development tools to create a web page that connects to the user's Spotify premium account. Spotify is the most popular streaming site and controls 34% of the music streaming market currently so this will help us reach a wider audience (Iqbal, 2021). On the development side, Spotify provides extensive documentation and guides which means the platform is easily adoptable to our needs. For example, Spotify provides developers with a dashboard for their applications so that they can protect their app from cyber-attacks and track the usage of the app itself. In order to use Spotify as a developer, we must also comply with design guidelines and be mindful of how we access a user's account.

3.1.1 Spotify Web Playback SDK

The Spotify Web Playback SDK (beta) is a client-side JavaScript library which allows us to play a track from Spotify in the browser. According to Spotify's documentation, the SDK uses

the same technology that Spotify's Web Player does. The SDK works by creating a "new player" on a user's Spotify app. At first, we considered using the Web API to play music, but the SDK accomplishes our goal with fewer technologies. Therefore, we decided to use the SDK which although is a beta version, comes with a lot of helpful documentation.

3.1.2 Spotify Web API

The Spotify Web API allowed us to make calls to various endpoints which we needed to pause and resume songs.

3.2 Glitch

We chose to host our application on Glitch for many reasons. First, Glitch has Electronic Media Encryption. Initially we were hosting our website on Bryn Mawr College servers, but ran into permission errors when we tried playing music. Glitch allows us to play music securely. Additionally, Glitch also provides us with an IDE through which we can collaborate on coding efficiently.

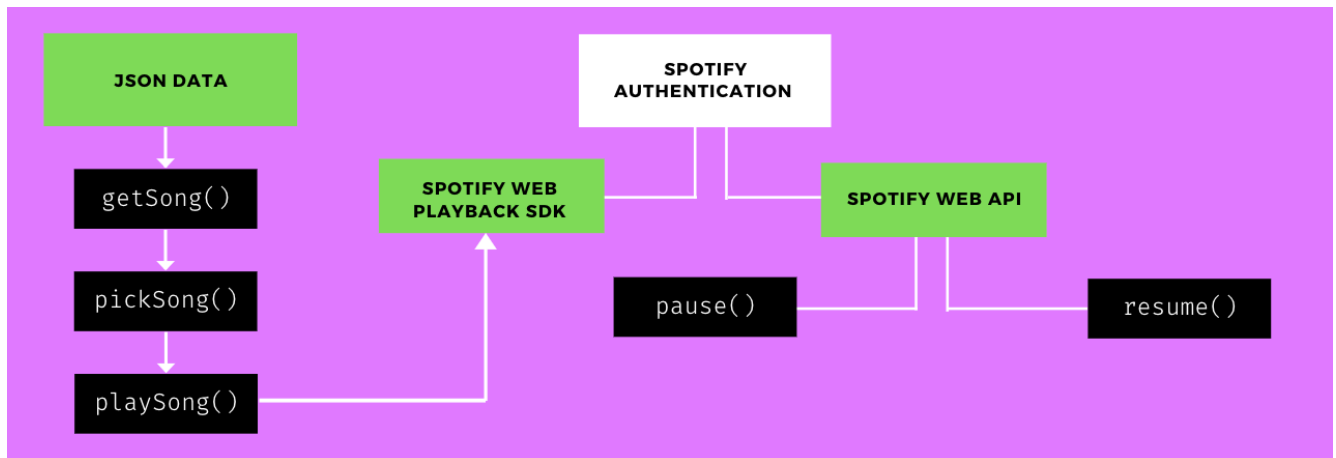
3.2.1 JavaScript, HTML and CSS

All of our code is written in JavaScript, HTML and CSS and hosted through Glitch. Initially, we were also going to use Node.js, but learned this is not compatible with the SDK. Since using the SDK is a higher priority than Node.js, we designed our entire app using these three languages.

3.3 JSON Data File

The Spotify API does not have a search feature that allows us to look up songs based on their features (such as valence, danceability and energy). The Web Playback SDK also does not have a mechanism to search for songs, and it requires a specific track id to play a song. Therefore, we decided to use a dataset that maps a song's features and track id together. We found a dataset on Kaggle that contains over 175k songs from 1920 to 2021 that contains all the Spotify data available on the tracks. Originally, the data file was formatted as CSV file. However, since JSON files allow easier and more efficient access to data, we decided to convert the CSV file to a JSON file. This JSON file is used to look up songs by their features and provide their track id's. This will feed into the Web Playback SDK.

4. Implementation



The image above summarizes our implementation. We used a JSON data file that contained the songs and their corresponding information. The `getSong()` function converts all the songs in the data file into an object to help access the data more efficiently. This object is used by the `pickSong()` function. This function filters the songs based on their valence, danceability, energy and popularity values. The filtered songs are saved in a list and then one song is randomly chosen. The track id for the chosen song is sent to the `playSong()` function which connects to the Spotify Web Playback SDK and then sends the request to play the song by providing the track id and the user's device id. On the other hand, the Spotify Web API allows us to pause and resume songs in the user's playback. Spotify authentication gives us permission to use the Web Playback SDK and Web API.

4.1 Data

As mentioned previously, we are using a dataset that contains Spotify tracks and their features. The original CSV file contained 19 fields of data. We cleaned the dataset using Microsoft Excel, to include only 8 fields to improve runtime when loading the file. The field names and their description from the Spotify API are listed below:

1. `artists`: The artists credited for the track.
2. `name`: The name of the track.
3. `id`: A 62-base unique identifier provided by Spotify for a track.
4. `year`: The release year of the track.
5. `popularity`: The popularity of a track lately in United States, on a scale of 1 to 100.
6. `valence`: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
7. `danceability`: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
8. `energy`: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual

features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

We further cleaned this dataset by filtering out songs released before 1990. We made this decision to appeal to the younger audience that is more likely to use our application. However, it is important to note that songs that were originally made before 1990 but released again (for example as a “remastered” version) are still available in our database. This left us with 62,700 tracks in our database. Next, we converted the CSV file to a JSON file using an online converter. Initially, we attempted to convert the CSV file to a JSON file using JavaScript. However, we realized that this would require using external libraries that could parse the CSV file and convert it to a JSON object. We decided to use an online converter instead to avoid environment issues and save time. The online converter provided us with a JSON file of which a sample can be seen below:

```
[
  {
    "artists": "['Gerry & The Pacemakers']",
    "danceability": 0.484,
    "energy": 0.265,
    "id": "6catF1lDhNTjjGa2GxRQNN",
    "name": "You'll Never Walk Alone - Mono; 2002 Remaster",
    "popularity": 55,
    "valence": 0.285,
    "year": 2008
  },

```

4.2 Spotify Web Playback SDK

The Spotify Web Playback SDK has three requirements listed in their documentation:

1. A Client ID and Client Secret, which can be obtained from our Dashboard.
2. Authenticated users must have a valid Spotify Premium subscription.
3. A supported web browser running on the user’s device.

The third requirement is described further in the table below:

Supported Browsers

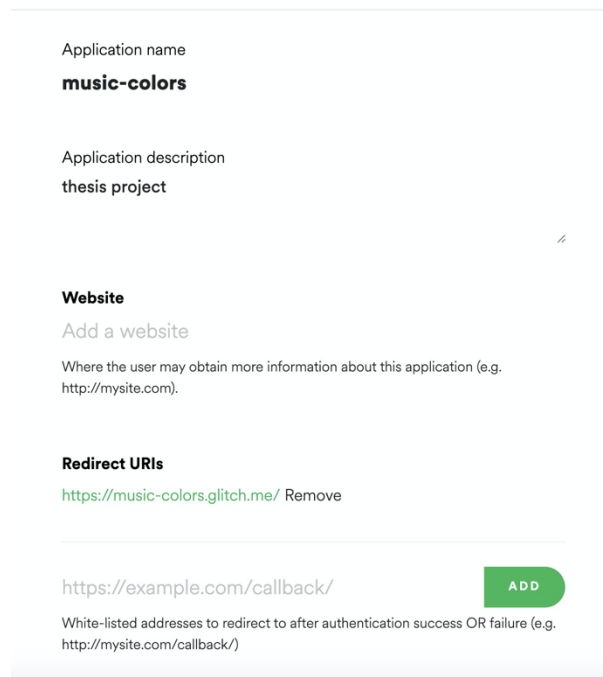
OPERATING SYSTEM	BROWSERS	STATUS
Mac/Windows/Linux	Chrome, Firefox, IE*	✓ Supported
	Microsoft Edge	✓ Supported
	Safari	× Not Supported
Android	Chrome, Firefox	× Not Supported
iOS	Safari, Chrome	× Not Supported

The supported web browsers are a limitation of Spotify that we understand are a limitation of our project as well. This section will address how we satisfied the other two requirements.

4.2.1 Spotify App Authentication

The first step to creating an app using the Spotify API is to register it with Spotify. In doing so, we are provided with a client id and client secret which we need in order to use the SDK. We

registered our app using the Spotify Developer Dashboard. Although apps can be built without registration, we chose to register our app because of the first requirement to use the SDK listed above. Another advantage of registering the app is that Spotify provides developers with a visual analysis of the usage statistics their app such as how many monthly users they have on the Dashboard. In order to successfully authenticate a user and allow them to use all the features on our website, we must provide a Redirect URI (our website). The addresses entered as Redirect URIs are whitelisted with Spotify. This URI enables the Spotify authentication service to automatically re-launch our app every time a user logs in successfully.



The screenshot shows the Spotify Developer Dashboard configuration page for an application named "music-colors". The application description is "thesis project". Under the "Website" section, there is a link to "Add a website" and a note that it is where the user can obtain more information about the application. Under the "Redirect URIs" section, a URI "https://music-colors.glitch.me/" is listed with a "Remove" button. At the bottom, there is an input field containing "https://example.com/callback/" and an "ADD" button. A note below the input field states: "White-listed addresses to redirect to after authentication success OR failure (e.g. http://mysite.com/callback/)".

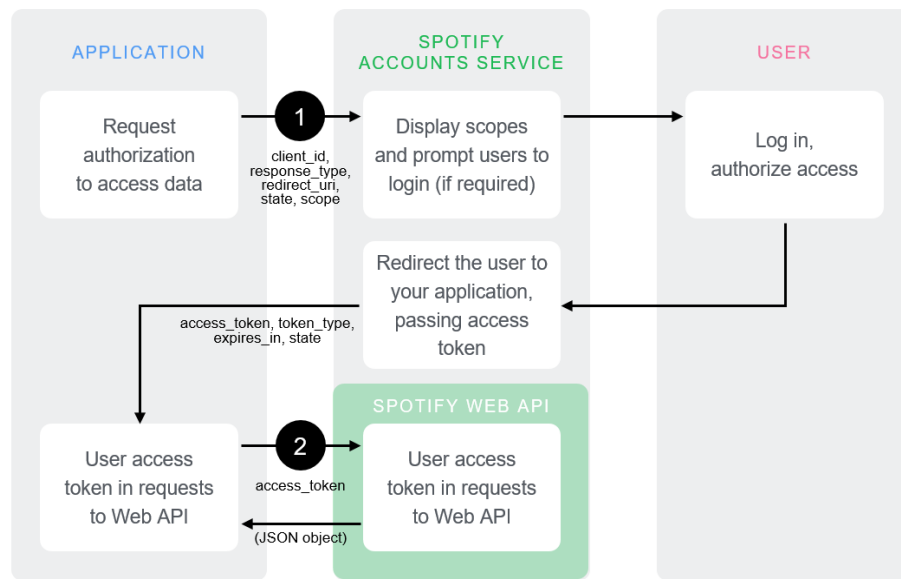
After registering the app, we receive the client id and client secret. As defined in the documentation, the client id is the unique identifier of our application and the client secret is the key that you pass in secure calls to the Spotify Accounts and Web API services.

4.2.2 Spotify User Authentication

The second requirement for using the SDK calls for users to be authenticated. However, the SDK itself does not have the ability to authenticate users. Therefore, we have to use a Spotify User Authentication Flow to authenticate our users. There are four different authorization flows provided by Spotify:

1. Authorization Code Flow
2. Authorization Code Flow with Proof Key for Code Exchange (PKCE)
3. Implicit Grant
4. Client Credentials Flow

Initially we attempted to implement the Authorization Code Flow. However, this flow requires sending our secret key, which would have to be performed on a secure location to avoid revealing it. Therefore, we decided to use Implicit Grant instead because it only used JavaScript and does not require sharing secret keys. In other words, it does not require any server-side code, which fits in with our design better. One disadvantage to Implicit Grant is that users have to log in every time the page is refreshed. Below is an image provided by Spotify that maps out Implicit Grant Flow.



For our project, we use the access token provided after successful authorization to make calls to the Spotify Web Playback SDK instead of the Web API. The entire flow was implemented using JavaScript only. In implementing the Implicit Grant flow, we redirect a user to `/authorize` endpoint of the Accounts service. To make this call, we also have to provide client id (obtained from requirement 1), a response type which will be set to “token”, and a redirect UI which is our website. The user will be directed to enter their credentials if they have not already done so previously. After successful authentication, we receive an “access token” which will be used to access the SDK.

In addition to user authentication, the second requirement for using the SDK notes that only users with a Spotify Premium account can access the application. We acknowledge that this is a drawback of our application, but the only resource Spotify provides to play full songs over a browser as of right now is the SDK so we believe this is the best way to implement our solution.

4.2.3 Using the SDK

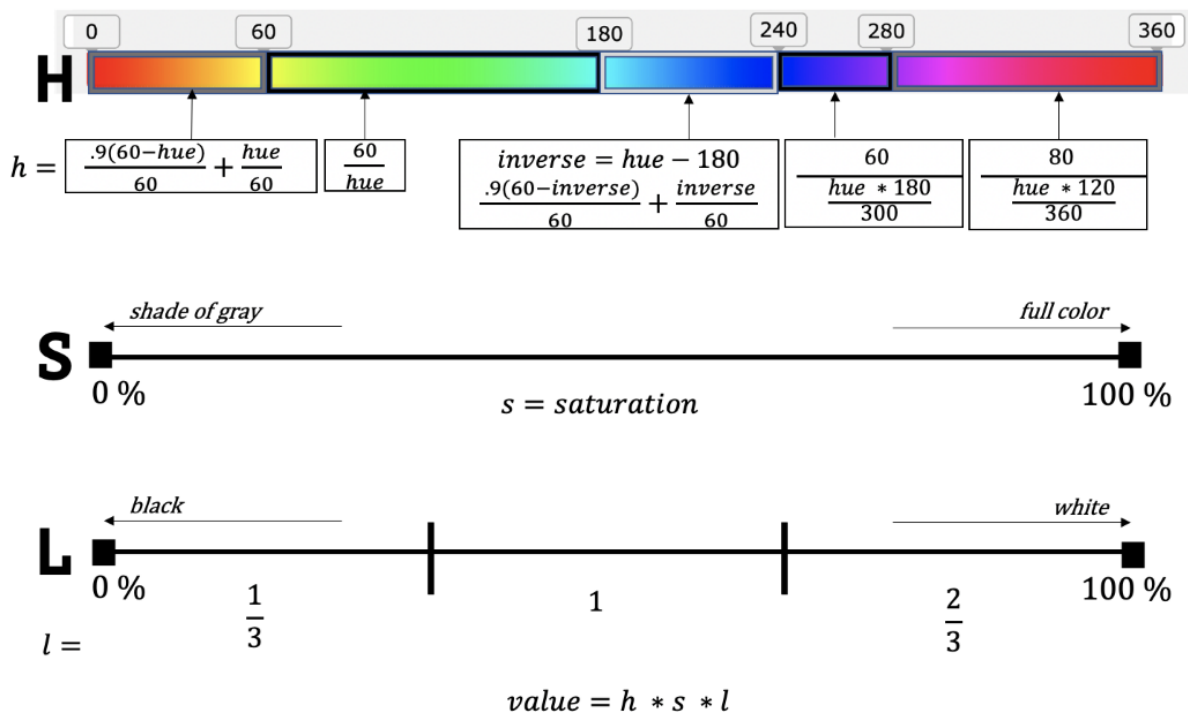
Since we have addressed all the requirements for the SDK, we can now use it in our app. To install the SDK, we have to embed the JS library for it in our HTML code. Then we initialize the player in our JavaScript file. To do so we must first define a `window.onSpotifyWebPlaybackSDKReady` method. Once this is executed, the player will be ready to load on the SDK. To load the player, we use a “ready” event which is emitted when the Web Playback SDK has successfully connected and returns a `WebPlaybackPlayer` object. This object contains the device id for the player we have created. Finally, this device id will be used to send our request and play the track through the player we have created.

4.3 Playing a Track

4.3.1 Colors

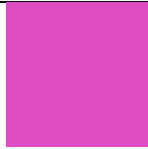

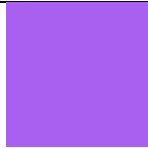
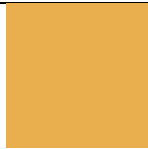
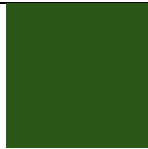
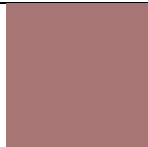
To simplify the color picker and algorithm we are using Spectrum, a jQuery project by Brian Grinstead that includes one CSS file and one JavaScript file to complete all the functions we need in a color picker. Mainly we will use this color picker to create a form through which a user can submit colors in many formats- including by typing in RGB, HSL, or HEX values or by using the “eye dropper” tool that selects any color on the user's entire screen when the user hovers over and selects it.

Users can either choose three colors manually or use a “random” button which will choose three colors for the user. We used a form with input type “color” from the spectrum package to create a gradient drop down from which users can pick colors. The same algorithm is used to get values for all three colors given. This means that if we have a color which gives us a value of 0.7, the value will not change based on a track feature. Every time that color is chosen, no matter if it is first input (which is later matched to valence), second input (which is later matched to danceability) or third input (which is later matched to energy) the value will always be 0.7. The values calculated are then stored in local storage. The image below summarizes the logic behind the conversion.



The divisions in hue were based on the color positivity scale we created, where we expected warmer colors to give us higher values and cooler colors to give us lower values. Since the goal of this algorithm was to convert the HSL values to a scale of 0.0-1.0, we first converted each field, H, S, and L to that range first.

Here are some examples of colors and their corresponding values:

Color	HSL value	0.0-1.0 range value
	H: 313 S: 88% L: 58%	0.67
	H: 173 S: 70% L: 56%	0.24
	H: 276 S: 93% L: 65%	0.34
	H: 38 S: 88% L: 58%	0.85
	H: 113 S: 86% L: 19%	0.15
	H: 0 S: 28% L: 57%	0.25

4.3.2 Convert Data File

Once the colors are saved in local storage, the JSON file is read and saved as an object called `songData` which contains all the tracks and their features as objects. This helps us access tracks based on their features efficiently. We use the `$.getJSON()` query to accomplish this.

4.3.3 Picking a Song

Next, we call the function `pickSong()` to pick a song given the valence, danceability, and energy values we previously stored in local storage. The function looks for songs which have the same valence value and danceability and energy features within a 0.001 range of the values we calculated. In other words, the danceability and energy of the song we choose should have corresponding danceability and energy values within a range 0.001, while the valence has to match exactly. Originally, we looked for songs that perfectly matched the values for all features we were looking for, but it was harder to find tracks this way and, in many instances, there were no results. In addition, we also added an if condition that filters out songs that have a popularity that is less than 50. We made this decision so that users can enjoy songs that are generally considered popular. It also increases the chance that users come across songs they are familiar with. This implementation did not compromise our goal of music discovery, since most songs in the dataset have a popularity greater than 50. For all the songs that satisfy our conditions, their track ids are added to a list from which we randomly pick one. The track id is stored as a global variable called `randomSong`.

4.3.4 Play a Song

As mentioned in section 4.2, we are using the Spotify Web Playback SDK to play songs. We use ajax to send a request to play the chosen track. To make the request, we must provide the device id (refer to section 4.2.3), the track id of the song (refer to section 4.3.3) and the access token (refer to section 4.2.2). Finally, we connect to the player using `player.connect()` and our song is played.

4.3.5 Changing a Track

One important part of our implementation was handling the case when a user picks new colors to play a new song before the current song is done playing. Before we play a new song, we must remove our current listener and disconnect our current player. This avoids lags that result from creating multiple players at the same time. By disconnecting our player and listener, when new colors are submitted, the player is initialized again.

4.4 Spotify Web API

To use the Spotify Web API, we must have authentication, which we have already satisfied (explained in section 4.2.2). Therefore, we have access to the numerous endpoints provided by Spotify. In our project, we only used the pause and play end points defined below.

Pause:

API Reference	Pause a User's Playback
Endpoint	<code>https://api.spotify.com/v1/me/player/pause</code>
HTTP Method	PUT
OAuth	Required

Play:

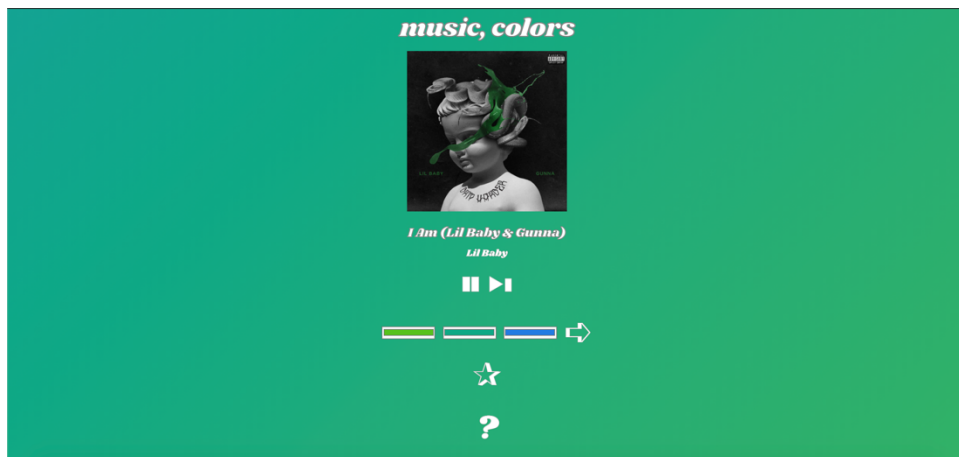
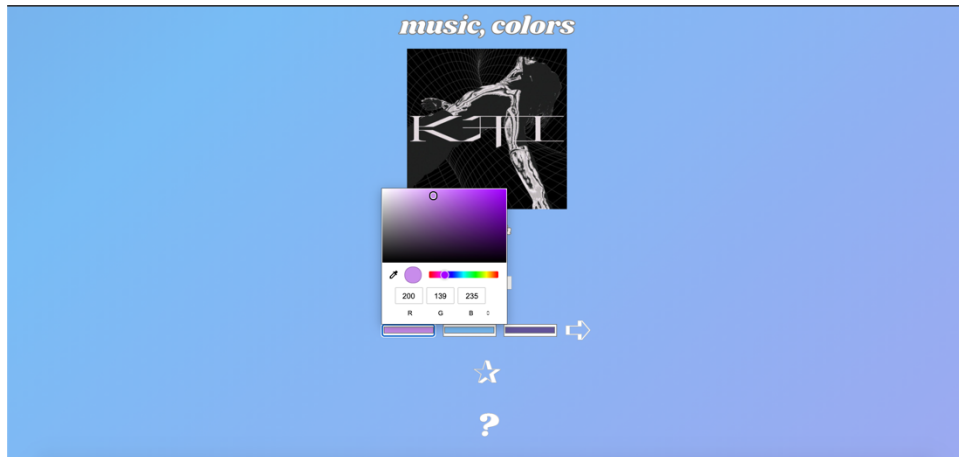
API Reference	Start/Resume a User's Playback
Endpoint	<code>https://api.spotify.com/v1/me/player/play</code>
HTTP Method	PUT
OAuth	Required

As indicated in the tables above, both endpoints require authentication (OAuth). We used Ajax to make these calls and pass the authorization tokens.

4.5 UI

The background of our application is an animated gradient that corresponds to the colors chosen by the user. We use the linear-gradient function from CSS to create the color gradient. We then set the animation at “gradient 15s ease infinite”. In our original plan, we wanted to create a more complex animation with moving characters. However, we decided to use a simple gradient instead so that users could have a visually aesthetic listening experience.

Our layout has three color selectors, a submit button and a random button. Additionally, there is a pause/play button and a help button that reveals a modal. The buttons are themed as 3D icons. Overall, our UI is very minimalistic. We believe this will make it easier for people to use regardless of their technical expertise. Here are some screenshots of our webpage in action:



5. Evaluation

Listening to music is an extremely subjective experience and so we believed that the best way to test the success of our project was to have users/testers provide feedback. We used a Google form to receive responses. The questions on the form are:

1. What colors did you choose?
2. What mood did you expect the song to be?
3. What mood was the song that played actually?
4. Did the song match the mood you were expecting?
5. Did you like the song?
6. Did you discover a NEW song you like?
7. Did you like the aesthetic of the website?

Questions 1-3 allowed the respondent to enter their own short answer responses. Questions 4-7 only had yes or no options. This helped us quantify our results better. The form could be filled out multiple times by the same user because the questions were answered for a new song each time. Questions 1-4 evaluated the strength of our algorithm that attempts to draw a connection between colors and music. Questions 5 and 6 will focus on our goal to improve music discovery. Finally question 7 will measure our goal to make the website aesthetically pleasing.

While our target demographic is people aged 18-25, the app is intended to be usable by everyone. We personally asked our own classmates and friends to try this website out and provide feedback to ensure that we get at least a handful of responses. We acknowledge that music and colors are both very subjective topics. Therefore, we asked people if they believe the song matched the mood they expected. Even if the song matched the mood we expected as developers, our users could've disagreed with our algorithm so we wanted to address this.

In addition, Spotify provided us with a useful dashboard that highlights analytics about our app. This dashboard provides the number of users per month, daily active users, users by country, and total number of requests to the app. This data gave us insight into how many people we can reach (i.e. are people sharing the app with their friends, family) and if people are returning to the app as well. While we don't have a global audience in mind, it will be cool to track the spread of our project as well.

5.1 Results

After our initial evaluation round, we received feedback asking for some additional features to be implemented. In order to better the user experience, we added the following elements:

1. An initial homepage with general information about what the website does, and the requirements needed by a user to access it.
2. A help feature on our main page that lets the user know what each of the elements on the page is and how to use it
3. A pause/play toggle button so a user can control playback right from this screen.
4. A next/skip button that will generate a new song for the same set of colors.

As for the quantifiable aspects of our survey, these are the results we received (based on 15 responses):

<i>Question</i>	<i>Yes</i>	<i>No</i>
Did the song match the mood you were expecting?	73.3%	26.7%
Did you like the song?	86.7%	13.3%
Did you discover a NEW song you like?	60%	40%
Did you like the aesthetic of the website?	100%	0%

Generally, the reactions we received to our project were positive. Our results say that the song matched the mood 73.3% of time. While we would like to see that number be higher, we understand how subjective the music listening experience can be, and so we think this is a good result. In future work, we believe our algorithm can be tweaked to increase this number as well. Furthermore, some of the users indicated in their feedback that while the song was not what they were expecting, they understood why it corresponded to the colors they chose.

One of our goals for this project was music discovery. In our evaluation, we found that users found a new song that they liked only 60% of the time. However, we found through this project that many people preferred getting music that was recognizable to them. Our users found it fun and exciting to see what kinds of colors would give them their favorite artists and songs. Therefore, while this 60% is quite low, we believe that the enjoyment people got out of using our project was worth the trade-off.

Additionally, the aesthetic of our website was received well by all our users. This was an extremely important aspect of our project, and we believe it was the most successful part of our project as well.

6. Related Work

MANGOMOJI

This website (Ahlin et. al, 2018) suggests a song to the user based on a combination of emojis that the user chooses. Although it is fun to use at first, the songs get repetitive soon and does not hold a user's attention for long. Our application will be better because we are using a data base with over 160k songs. Since one of our initial goals was music discovery, this will give user a chance to discover a wide range of new music. Also, there were only twenty-two emojis to choose from on MANGOMOJI. In contrast, our website allows users to select from color gradient which immensely increases the possible options. Also, the music played on the MANAGOMOJI app is only a preview for a song, and our application plays songs in its entirety. This allows our app to serve as a music listening service.

Color Therapy:

Similar to our website, Color Therapy (Virji, 2020) suggests songs based on a color the user suggests. The website includes infinite scrolling which allows for a wide range of color options, but we believe a gradient still gives a user more freedom. The website tries to make colors and mood less subjective by starting with a quiz that asks the user various questions, but we think this could be cumbersome for a casual user. The purpose of Color Therapy is to generate a playlist, whereas our goal is to play music in real time and encourage music discovery. Users may spend more time on our site because it doesn't have a definite end goal like generating a playlist. Since our website plays music through the Spotify app, users can still save songs generated by our website to their playlists directly using the Spotify app if they choose to.

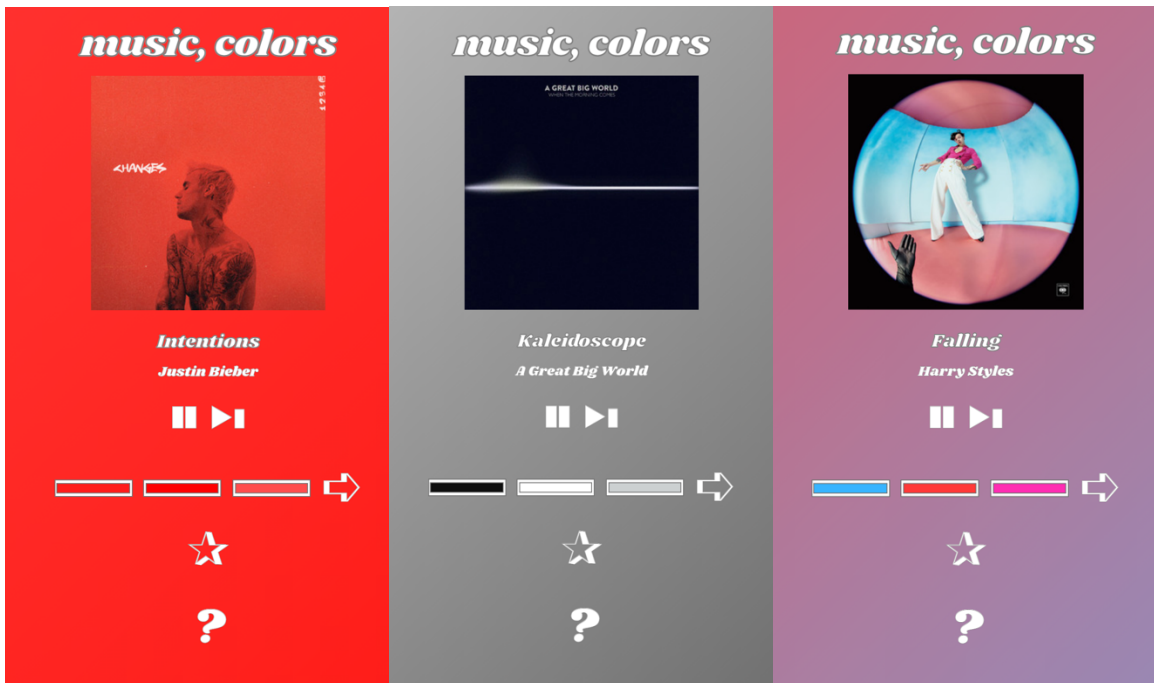
Predominantly:

Based on a color chosen from a gradient, this website (Predominantly, 2015) returns albums with a similar cover art color. Although we really liked this project, it solves a different problem than us. We are not choosing songs based on their album cover, but rather it is based on what they actually sound like. The website also doesn't actually play any music either, it only suggests albums. Our website will display the album cover and play music.

7. Conclusion

7.1 Future Work

One unexpected result of our project was the relation between the colors chosen by users and the album cover art. In several instances, we noticed that the colors chosen matched the colors of the album artwork. Below are a few occurrences of this.



Although we do not have enough evidence to draw a definite relationship between album artwork and the colors chosen, we think this will be an interesting topic for further analysis. Another possible project could involve generating songs whose album covers match the colors chosen.

In addition, one feature that could be implemented is continuous music play. In other words, as soon as one song ends, the next song that satisfies the colors chosen should start playing. Similarly, there could also be a feature that generates a playlist with all the songs that satisfy the conditions and save it to the user's library. These are popular features that were requested in our evaluation since people enjoy listening to music for long periods of time without interruption.

7.2 Reflection

With this project, we set out to create a place where an intersection between music, colors and emotions could be used to enhance music discovery and the listening experience. By converting HSL color values to valence, energy and danceability, we found a way to connect colors and music through emotion. Our evaluation showed that majority of listeners found that

the songs they chose matched the mood they expected. People experience a wide range of emotions and songs remain a source of comfort and understanding when it is hard to express what we want in words. In combining audio and visual elements in this project we've created a multi-sensory experience that remains fun and exciting for music lovers.

8. References

“Home: Spotify for Developers.” *Spotify for Developers*, developer.spotify.com/.

Ahlin, Richard, et al. *Mangomoji*, 2018, www.mangomoji.com/.

Virji, Kabir. *Color Therapy*, 2020, colortherapy.io/.

Predominantly, Open Work, 2015, predominant.ly/.

Dutton, Sam. “What Is EME? | Web Fundamentals | Google Developers.” *Google*, Google, 2021, developers.google.com/web/fundamentals/media/eme.

Grinstead, Brian. *Spectrum*, 1.8.1, MIT, 22 Aug. 2016, bgrins.github.io/spectrum/.

Iqbal, Mansoor. “Spotify Revenue and Usage Statistics (2021).” *Business of Apps*, 2 Apr. 2021, www.businessofapps.com/data/spotify-statistics/.

Yamac Eren Ay. 2020-05-23. Spotify Dataset 1921-2020, 160k+ Tracks, Version 10. Retrieved 2021-03-02 from <https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks>.