

Final Project Report: SpotiHolic

Team name: 404 NOT FOUND

Group member:

Name	Email	GitHub Username
Mingwei Ruan	ruanm@seas.upenn.edu	mruan238
Tsang Fonda Helen	fonda@seas.upenn.edu	fondatsang
Haoning Gong	ghaoning@seas.upenn.edu	ghaoning
Huaying Gu	huaying@seas.upenn.edu	kkkhora

1 Introduction

1.1 Project Goals

The goal of this project is to build up a web application with multiple entries for users to search Spotify songs and add those songs to their own playlist once those songs are in their favor. Apart from that, this application also provides direct links to Spotify for access to that song.

1.2 Application Functionality

The Spotify application (SpotiHolic) consists of following modules:

1. A search module to allow user to search songs based on:
 - a. Song name
 - b. Artist name
 - c. Year
 - d. Genre
2. An interactive module with a world map for users to select songs based on country.
3. A filter page to allow users to filter songs based on the emotional tendency.
4. A page with a playlist defined by users with their selection of favorite songs.
5. A recommendation module based on users' playlist preferences.

1.3 Team Motivation

We describe ourselves as music enthusiasts, and we do appreciate songs in different genres. Throughout the SpotiHolic application, we strive to bring some easy-to-know and easy-to-go features for people like us.

2 Architecture

2.1 Technology Used

Data-processing

1. Python and pandas

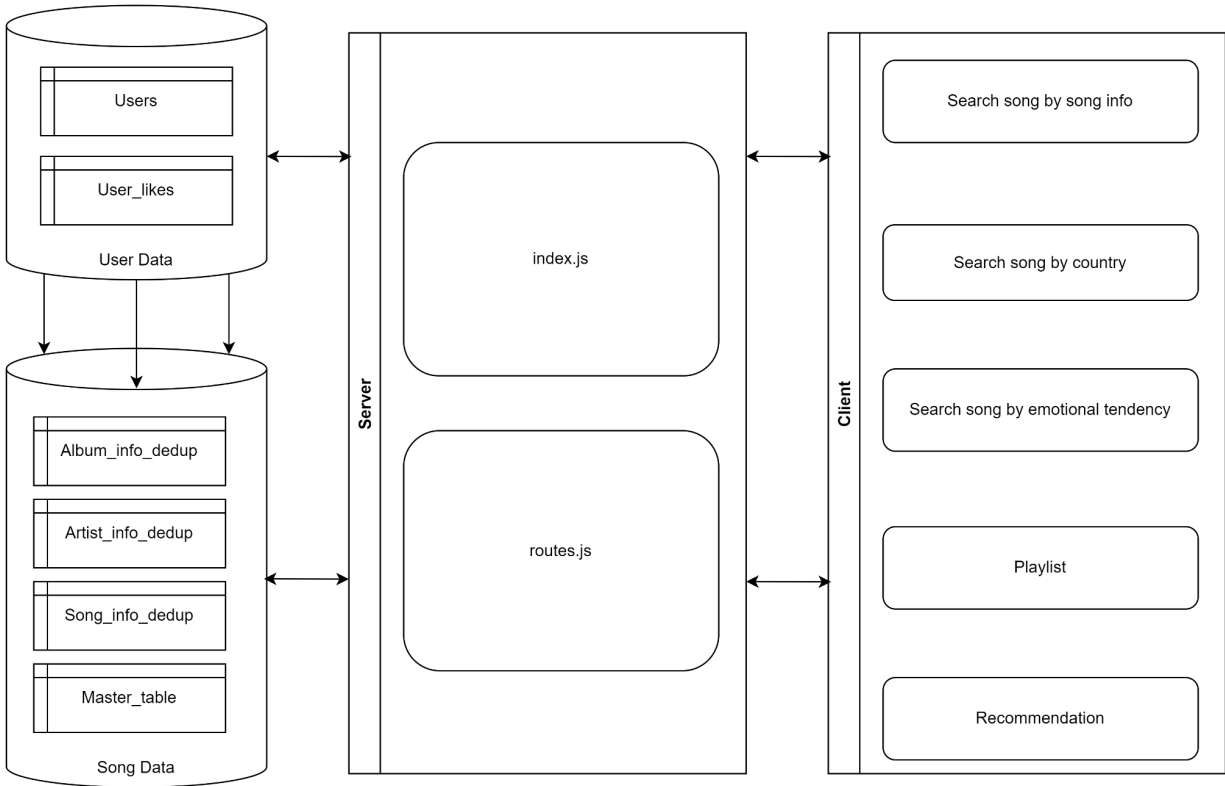
Front-end

1. React
2. Axios
3. Bootstrap
4. Next.js
5. Material-UI

Back-end

1. Node.js
2. Express
3. MySQL

2.2 System Architecture



3 Data

3.1 Dataset

Dataset 1: Spotify song information list

- Link to source: <https://www.kaggle.com/datasets/archiewebster/happysad>, <https://open.spotify.com/> (retrieve data using spotify api)
- Dataset statistics:
 - Row number: 24008
 - Attribute information:

Column Name	Data Type
Song_ID	varchar(100)
Acousticness	decimal(8,8)
Valence	decimal(4,4)

Danceability	decimal(4,4)
Energy	decimal(6,6)
Instrumentalness	decimal(8,8)
Liveness	decimal(4,4)
Loudness	decimal(10,3)
Speechiness	decimal(6,6)
Tempo	decimal(10,3)
Duration	int
Key_pitch	int
H_s	varchar(1)
Track_image	varchar(100)
Artist_image	varchar(100)

Dataset 2: Artist information list

- Link to source: <https://www.musixmatch.com/>(retrieve data using musixmatch api)
- Dataset statistics:
 - Row number: 23899
 - Attribute information:

Column Name	Data Type
Song_ID	varchar(100)
Song_name	varchar(100)
Song_popularity	int
Artist_name	varchar(100)
Artist_id	varchar(100)
Artist_genre	varchar(100)
Album_name	varchar(100)
Album_id	varchar(100)

Album_year	int
Song_genre	varchar(100)
Artist_country	varchar(4)

3.2 How Data is Used

Dataset 1 contains detailed song attributes, which are utilized in modules of search and recommendation. For example, the energy of a song indicates a perceptual measure of intensity and activity of that song, and songs with similar value of energy are more likely to be recommended simultaneously.

Dataset 2 is mostly used to display an artist image and correlate songs to the artist's country. It also provides supplementary information based on dataset 1.

3.3 How Datasets are Linked

Dataset 2 is extracted according to Song_name and Artist_name in dataset 1.

4 Database

4.1 Data Ingestion Procedure

4.1.1 Data Acquisition

- We downloaded the dataset from Kaggle: Spotify music analysis.
- Then we used spotify api to pull out other song attributes such as song name, genre, artist name, artist country, artist album, release date etc.
- We pulled supplementary data from Musixmatch using musixmatch api.

4.1.2 Data Cleansing

4.1.2.1 Song Features

1. We fix the column name and change the 'h/s' column to 'h_s', and 'key' to 'key_pitch' to circumvent MySQL constraints.
2. Remove duplicates: we use `Pandas.drop_duplicates()` to remove duplicated
3. After dropping the duplicates, we still found out that duplicate 'uri' exist as there are different 'h_s' values and 'danceability' for one 'uri'. In order to keep 'uri' unique, we further adjust to remove duplicate values for uri, we keep 'h' value for duplicate 'h_s' and larger 'danceability'.
4. Double check the dataset info and convert to csv.

4.1.2.2 Artist

1. For the Album year, we will extract the 'year' from `release_date`, the original formats of which vary from 'DD-MM-YYYY', 'MM-DD-YYYY' to 'YYYY'.
2. For the missing value, we will replace with N/A

4.1.2.3 Song Details

1. Remove the unused column 'snippet'
2. Replace other missing value with N/A

4.1.2.4 Genre

1. We also used python to 'unwind' all genres in the songs, and then regenerated a distinct list to store genre information.

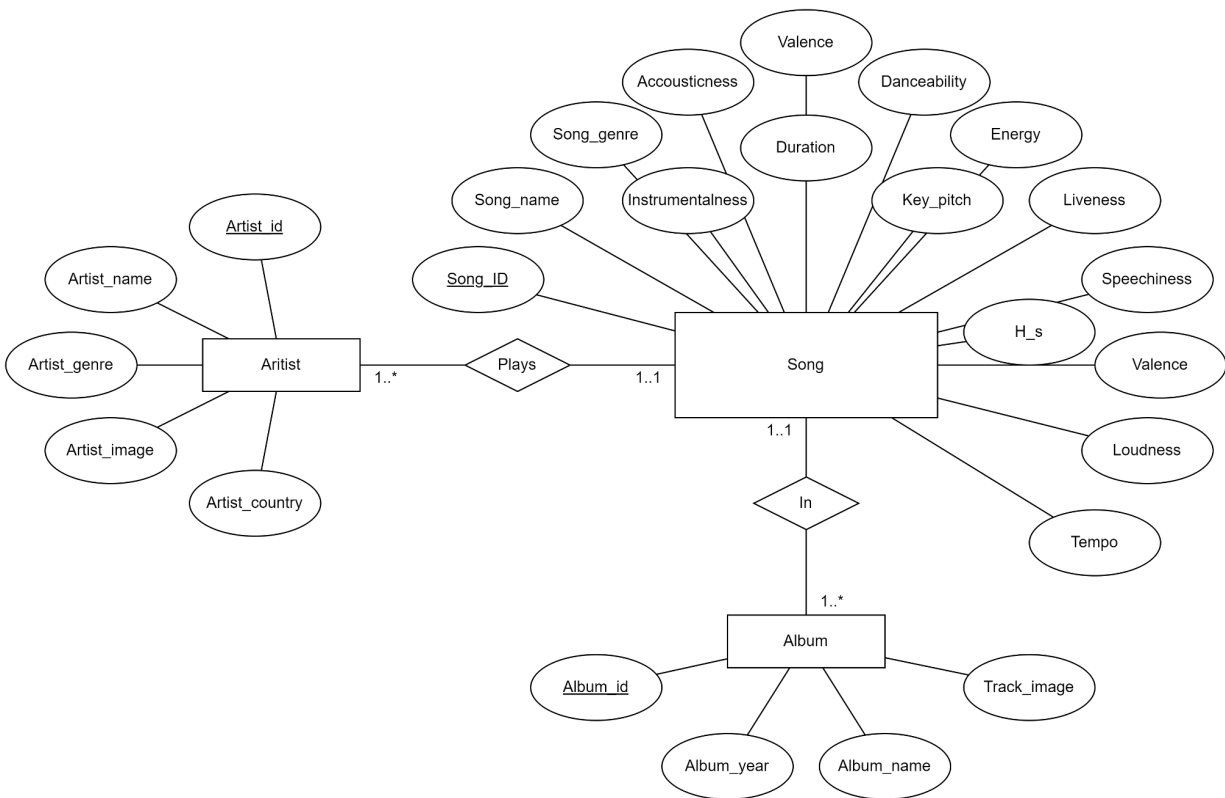
4.1.2.5 Song classifier

1. We use unsupervised machine learning technique to classifier all songs into 5 different categories based on K-means technique;
2. We first use Scale to scale all the attributes in the song features. And find out the best number of classifier using the Elbow Method. Here we use 5 as the best number;
3. K-means method to label all the songs into those 5 different categories.

4.2 Entity Resolution Efforts

To illustrate the relationship we are going to implement in SpotiHolic, 3 entities have been derived from original datasets, which are Artist, Song, Album. Clearly artists play songs, and songs are published in specific albums.

4.2 ER Diagram



4.3 Table Statistics

Statistics	Artist	Song	Album	User
Number of instance	1	1	1	2
Number of attribute	5	15	4	3
Number of row	9072	23899	18719	dynamic

4.4 Normal Form Used and Justification

In original datasets, there are following functional dependencies:

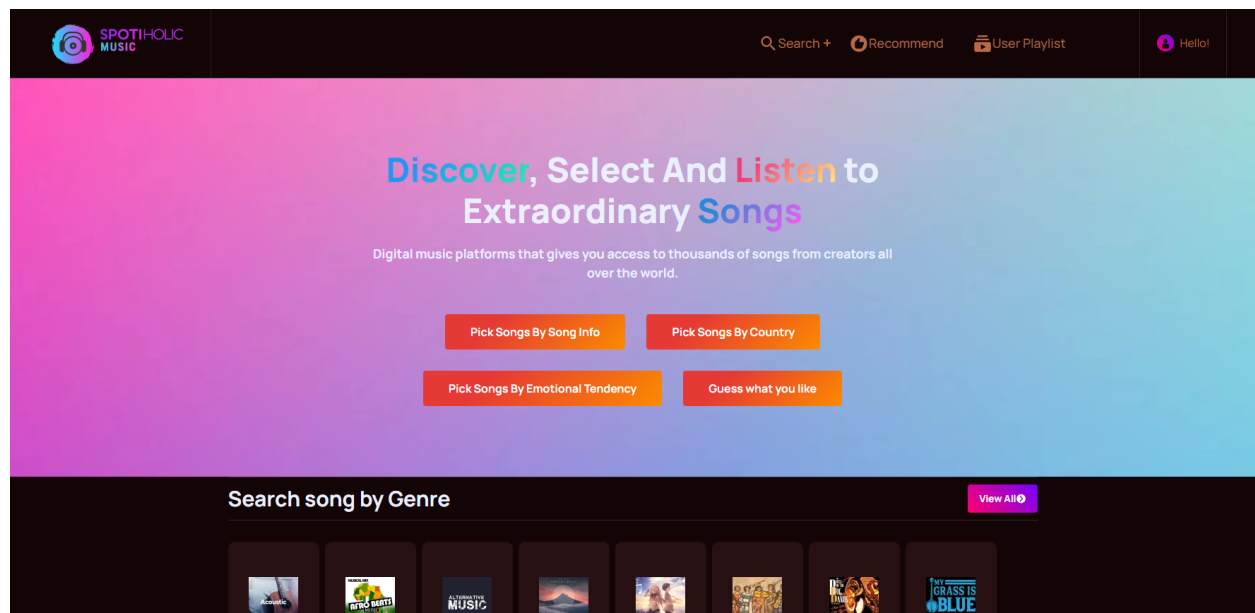
1. **Song_ID** → Song_name, Song_genre, Acousticness, Valence, Danceability, Energy, Instrumentalness, Liveness, Loudness, Speechiness, Tempo, Duration, Key_pitch, H_s, Artist_ID, Album_ID
2. **Artist_ID** → Artist_name, Artist_genre, Artist_image, Artist_country
3. **Album_ID** → Album_id, Album_name, Album_year, Track_image
4. **Username** → password, Song_ID (for account storage and user login purpose)

Since Artist_ID and Album_ID can be determined by Song_ID, according to 3NF, there are 3 tables needed for the design.

5 Web Application Description

5.1 Web Application Design

5.1.1 Home Page



Home page lists entries for other pages including:

1. Pick songs by song info
2. Pick songs by country
3. Pick songs by emotional tendency
4. Guess what you like (song recommendation)
5. A subset of song genres
6. A subset of recommended songs

Apart from that, the home page also lists developers and application logos.

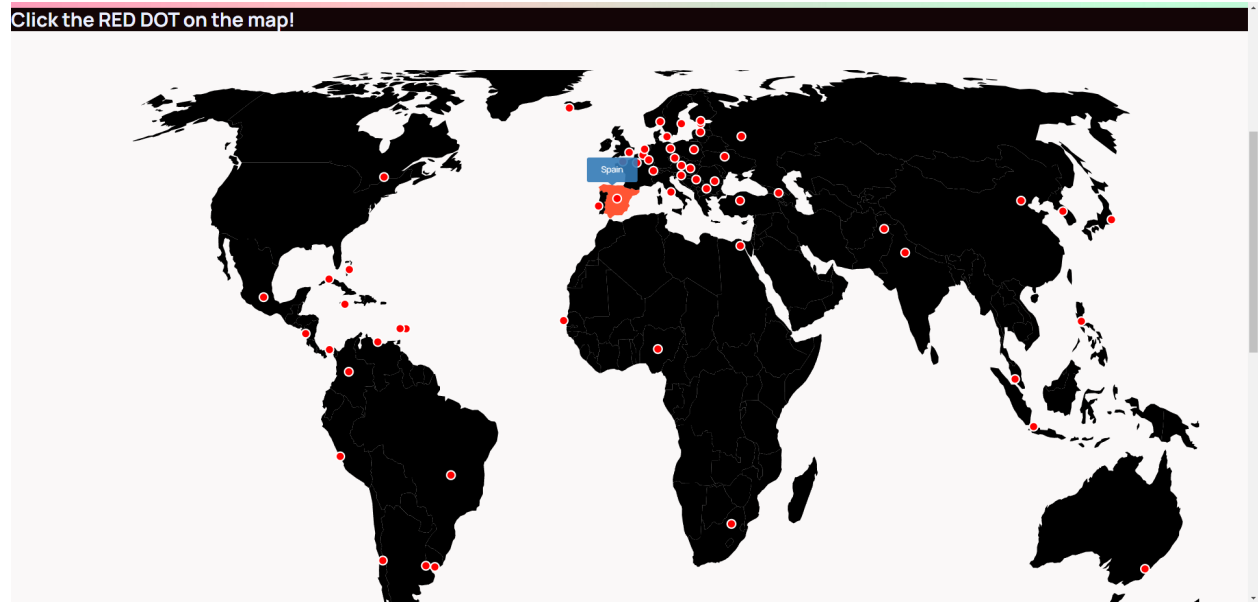
5.1.2 Pick songs by song info

The screenshot shows a web interface with a dark background. At the top, there is a section titled "Browse By Song Name or Artist Name" in white text. Below this title, there are two rounded rectangular input fields: "Search by Song Name" and "Search by Artist Name". Below these fields is a red "Submit" button. Further down, there is a section titled "Browse By Year" in white text. Below this title, there is a text label "Drag to select the year range:" in yellow. Below the label is a horizontal range slider with a blue line and two yellow circular handles. Below the slider is a red "Submit" button. At the bottom of the interface, there is a rounded rectangular input field labeled "Search by Exact Year" and another red "Submit" button below it.

This page allows users to:

1. Input song name or artist name to search songs
2. Use slide bar to specify range of years to search
3. Input the exact year to search
4. Select song genre to search

5.1.3 Pick songs by country





This page includes an interactive map for users to click the country to search.


5.1.4 Pick songs by emotional tendency


Browse By Emotional Tendency


Drag to select the emotional tendency range:


Acousticness: 


Valence: 

Danceability: 

Energy: 

Instrumentalness: 

Tempo: 

Happy ☒ / Sad ☐ / All ☐ 

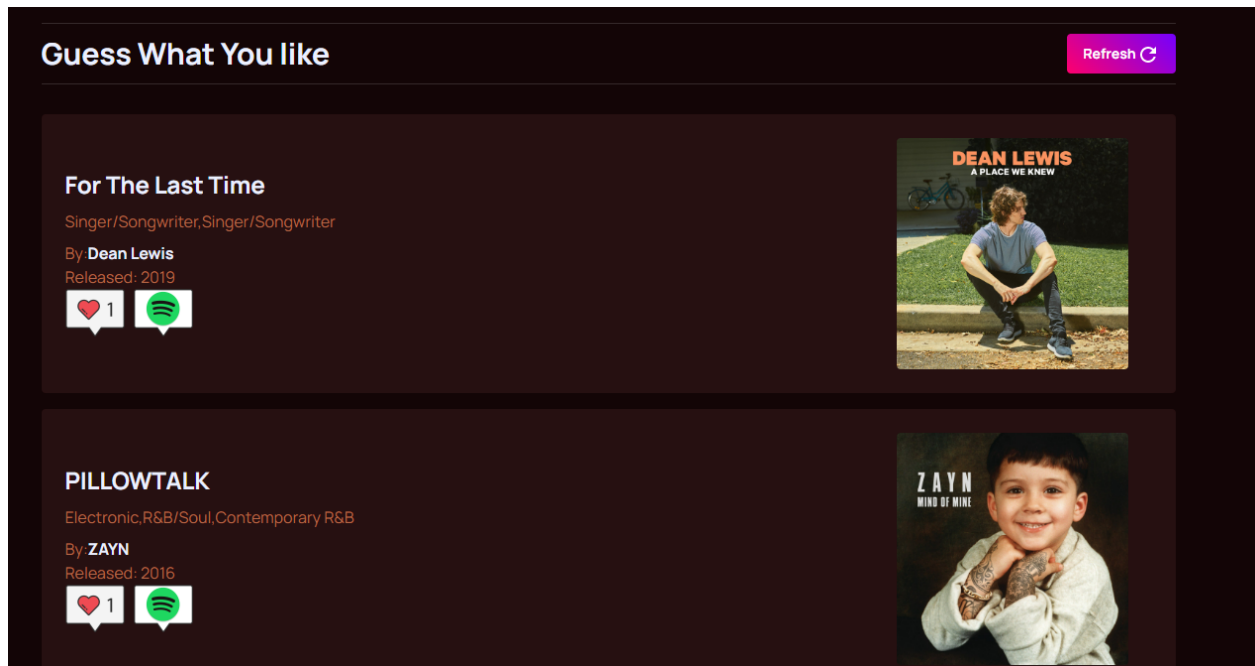
This page allows users to search songs based on following attributes:

1. Acousticness
2. Valence
3. Danceability

4. Energy
5. Instrumentalness:
6. Tempo
7. Happy or sad or all

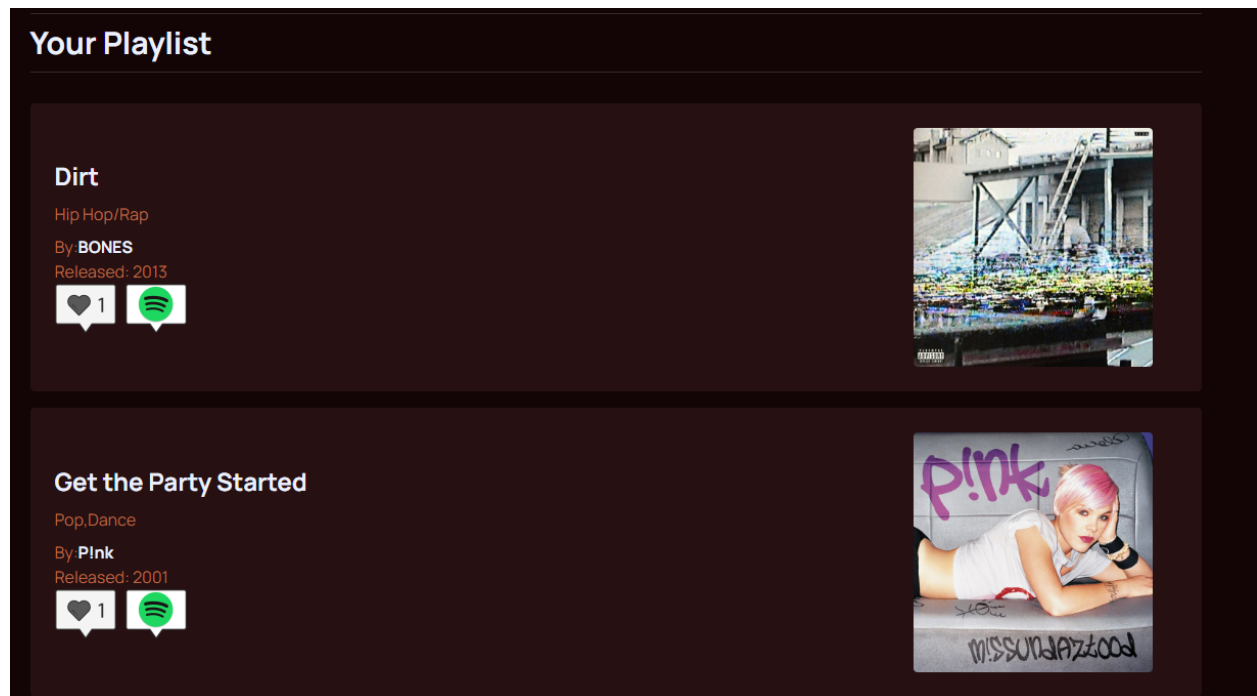
Each selectable attribute has a hint for elaboration when clicking.

5.1.5 Recommendation



This page displays a recommendation list of songs based on users existing favorites.

5.1.6 User Playlist



This page displays the songs that users mark as favorites.

6 API Specification

6.1 API 1

Route 1: </search/song>

Description: Returns an array of selected attributes for songs that match the search query, information including its name, song_id, genre, and respective attributes

Route Parameters: *None*

Query Parameters: `song_name` (string), `artist_name` (string), `page` (int)*, `pagesize` (int)* (default: 10)

Route Handler: `search_songs(req, res)`

Return Type: JSON

Return Parameters: { results (JSON array of { Song_id (string), Name (string), genre (string), release_year (string) , Track_image (String), Artist_name (string), Artist_image (string) }) }

Expected Behavior:

- Case 1: If the page parameter (page) is defined
Return artists entries with all the above return parameters for that page number by considering the page and pagesize parameters. For example, page 1 and page 7 for a page size 10 should have entries 1 through 10 and 61 through 70 respectively.
 - Case 2: If the page parameter (page) is not defined
Return all artists entries with all the above return parameters
 - Users can input either both of the parameters, or any one of them.
 - If the attributes names are found, return the singleton array of all the attributes available, but if the name is a string but is not found, return an empty array as 'results' without causing an error.
 - Values like Track_image and Artist_image might be NULL for some entries - return them as is.
-

6.2 API 2

Route 2: /search/search_country

Description: Returns an array of selected attributes for songs that match the search query, information including its artist_id, name, genre, image url

Route Parameters: None

Query Parameters: Artist_country (string),page (int)*, pagesize (int)* (default: 10)

Route Handler: search_country(req, res)

Return Type: JSON

Return Parameters: { results (JSON array of { Song_id (string), Name (string), genre (string), release_year (string) , Track_image (String), Artist_name (string), Artist_image (string) }) }

Expected Behavior:

- Case 1: If the page parameter (page) is defined
Return artists entries with all the above return parameters for that page number by considering the page and pagesize parameters. For example, page 1 and page 7 for a page size 10 should have entries 1 through 10 and 61 through 70 respectively.

- Case 2: If the page parameter (page) is not defined
Return all artists entries with all the above return parameters
 - If the attribute is found, return the singleton array of all the attributes available, but if the name is a string but is not found, return an empty array as 'results' without causing an error.
 - Values like Artist_image and Artist_genre might be NULL for some entries - return them as is.
-

6.3 API 3

Route 3: /search/search_year

Description: Returns an array of selected attributes for songs that match the search query, information including its artist_id, name, genre, image url

Route Parameters: None

Query Parameters: released_year (decimal), page (int)*, pagesize (int)* (default: 10)

Route Handler: search_year(req, res)

Return Type: JSON

Return Parameters: { results (JSON array of { Song_id (string), Name (string), genre (string), release_year (string) , Track_image (String), Artist_name (string), Artist_image (string) }) }

Expected Behavior:

- Case 1: If the page parameter (page) is defined
Return artists entries with all the above return parameters for that page number by considering the page and pagesize parameters. For example, page 1 and page 7 for a page size 10 should have entries 1 through 10 and 61 through 70 respectively.
 - Case 2: If the page parameter (page) is not defined
Return all artists entries with all the above return parameters
 - If the attribute is found, return the singleton array of all the attributes available, but if the name is a string but is not found, return an empty array as 'results' without causing an error.
 - Values like Artist_image and Artist_genre might be NULL for some entries - return them as is.
-

6.4 API 4

Route 4: /search/search_yearRange

Description: Returns an array of selected attributes for songs that match the search query, information including its artist_id, name, genre, image url and country

Route Parameters: *None*

Query Parameters: start_year (decimal),end_year (decimal),page (int)*, pagesize (int)* (default: 10)

Route Handler: search_yearRange(req, res)

Return Type: JSON

Return Parameters: { results (JSON array of { Song_id (string), Name (string), genre (string), release_year (string) , Track_image (String), Artist_name (string), Artist_image (string) }) }

Expected Behavior:

- Case 1: If the page parameter (page) is defined
Return artists entries with all the above return parameters for that page number by considering the page and pagesize parameters. For example, page 1 and page 7 for a page size 10 should have entries 1 through 10 and 61 through 70 respectively.
 - Case 2: If the page parameter (page) is not defined
Return all artists entries with all the above return parameters
 - Return all tuples within the attributes range in the form of the singleton array of all the attributes available, but if the name is a string but is not found, return an empty array as 'results' without causing an error.
 - Values like Artist_image and Artist_genre might be NULL for some entries - return them as is.
-

6.5 API 5

Route 5: /search/search_emotion_range

Description: Returns an array of selected song attributes sorted by their names in alphabetical order. Returns information including its name, genre, and respective attributes

Route Parameters: *None*

Query Parameters: Acousticness (decimal), Valence (decimal), Danceability (decimal), Energy (decimal), Instrumentalness (decimal), Tempo (decimal), H_s (string), page (int)*, pagesize (int)* (default: 10)

Route Handler: search_emption_range(req, res)

Return Type: JSON

Return Parameters: { results (JSON array of { Song_id (string), Name (string), genre (string), release_year (string) , Track_image (String), Artist_name (string), Artist_image (string) }) }

Expected Behavior:

- Case 1: If the page parameter (page) is defined
Return artists entries with all the above return parameters for that page number by considering the page and pagesize parameters. For example, page 1 and page 7 for a page size 10 should have entries 1 through 10 and 61 through 70 respectively.
 - Case 2: If the page parameter (page) is not defined
Return all artists entries with all the above return parameters
 - HappySad is a selective attribute, if all is input, an empty string is passed as attribute.
 - If the attributes are match, return the singleton array of all the attributes available, but if the name is a string but is not found, return an empty array as 'results' without causing an error.
 - Values like Artist_image and Artist_genre might be NULL for some entries - return them as is.
-

6.6 API 6

Route 6: /search/search_genre

Description: Returns an array of selected attributes for songs that match the search query, information including its artist_id, name, genre, image url and country

Route Parameters: *None*

Query Parameters: genre (string), page (int)*, pagesize (int)* (default: 10)

Route Handler: `search_genre(req, res)`

Return Type: JSON

Return Parameters: { results (JSON array of { `Song_id` (string), `Name` (string), `genre` (string), `release_year` (string) , `Track_image` (String), `Artist_name` (string), `Artist_image` (string) }) }

Expected Behavior:

- Case 1: If the page parameter (`page`) is defined
Return artists entries with all the above return parameters for that page number by considering the page and pagesize parameters. For example, page 1 and page 7 for a page size 10 should have entries 1 through 10 and 61 through 70 respectively.
 - Case 2: If the page parameter (`page`) is not defined
Return all artists entries with all the above return parameters
 - If the attribute is found, return the singleton array of all the attributes available, but if the name is a string but is not found, return an empty array as 'results' without causing an error.
 - Values like `Artist_image` and `Artist_genre` might be NULL for some entries - return them as is.
-

6.7 API 7

Route 6: `/login`

Description: User with an account contains a unique username and password. Parameters are checked from the Users Table in the database.

Route Parameters: *None*

Query Parameters: `username` (string), `password`(string)

Route Handler: `loginResponse(req, res)`

Return Type: JSON

Return Parameters:None

Expected Behavior:

- Case 1: Username unexist
User login fails, an error message "User unexist. Please sign up first!" is displayed to user.

- Case 2: Password does not match with username stored in database
User login fails, an error message “wrong password” is displayed to user.
 - Case 3: Username match with password
User login successfully, user playlist access available.
-

6.7 API 7

Route 7: /register

Description: let user register an account with unique username and password. This information will be inserted to the Users Table in the database.

Route Parameters: None

Query Parameters: username (string), password(string)

Route Handler: registerResponse(req, res)

Return Type: JSON

Return Parameters: None

Expected Behavior:

- Case 1: If the username already exists
User register failed, an alert "The user already exist!" will be displayed to the user.
 - Case 2: If the username does not exist
User register succeeded, username and password will be inserted into the User table in Database.
-

6.8 API 8

Route 8: /playlist

Description: User with an account contains a private playlist stored in the User table in the database.

Route Parameters: None

Query Parameters: `username` (string), `password`(string)

Route Handler: `search_genre`(req, res)

Return Type: JSON

Return Parameters: { results (JSON array of { `Song_id` (string), `Name` (string), `genre` (string), `release_year` (string) , `Track_image` (String), `Artist_name` (string), `Artist_image` (string) }) }

Expected Behavior:

- Case 1: If the no playlist exist
None will be return
 - Case 2: If the playlist existed
Return all liked songs by users that are stored in the database.
-

6.9 API 9

Route 9: `/userRec/song`

Description: let user register an account with unique username and password. This information will be inserted to the Users Table in the database.

Route Parameters: *None*

Query Parameters: `username` (string)

Route Handler: `userRec_song`(req, res)

Return Type: JSON

Return Parameters: { results (JSON array of { `Song_id` (string), `Name` (string), `genre` (string), `release_year` (string) , `Track_image` (String), `Artist_name` (string) }) }

- If the user parameter (`user`) is defined
Return song entries with all the above return parameters for 10 records.
 - If the user parameter is not find, random return song entries with 10 records.
 - Values like `Artist_image` and `Artist_genre` might be NULL for some entries - return them as is.
-

6.10 API 10

Route 10: `/like/:username/:songID`

Description: User with an account contains a private playlist. Liked Song will be inserted into the Users Table in the database.

Route Parameters: *None*

Query Parameters: `username` (string), `Song_id` (string)

Route Handler: `user_like`(req, res)

Return Type: JSON

Return Parameters: None

Expected Behavior:

- Case 1: If the user haven't login
An alert will be sent to require a user login
 - Case 2: If the user login correctly
The song will be updated to the user playlist that is stored in the user table in the database.
 - Case 3: repeated likes will be handle by checkLike API
-

6.11 API 11

Route 11: `/dislike/:username/:songID`

Description: User with an account contains a private playlist. Liked Song will be removed from the Users Table in the database.

Route Parameters: *None*

Query Parameters: `username` (string), `Song_id` (string)

Route Handler: `remove_like`(req, res)

Return Type: JSON

Return Parameters: None

Expected Behavior:

- Case 1: If the user haven't login
No song will display unlike button
 - Case 2: If the user login correctly
The song will be removed from the user playlist that is stored in the user table in the database.
-

6.12 API 12

Route 12: /check/:username/:songID

Description: User with an account contains a private playlist. This Parameter (Song_id) will be checked if it already exists in the Users Table in the database.

Route Parameters: None

Query Parameters: genre (string), page (int)*, pagesize (int)* (default: 10)

Route Handler: check_like(req, res)

Return Type: JSON

Return Parameters: None

Expected Behavior:

- This function only call by other function that already check user login status (user_like API)
 - Case 1: If the attributes already exist in the user table database
A fail status will be returned.
 - Case 2: If the attribute is not found in the user table database
A success status will be returned and that attribute will be inserted to the user table database.
-

7 Queries

7.1 Query Examples

7.1.1 Search songs by country

```
SELECT
    D.Song_ID,
    Song_name,
    D.Artist_name,
    Album_year,
    Song_genre,
    Track_image
FROM Display_results D JOIN Artist_info_dedup Aid on D.Artist_id =
Aid.Artist_id
WHERE Artist_country = '${req.query.countryCode}'
LIMIT ${pagesize} OFFSET ${offset}
```

This query specifies the country of the songs and returns song name, album name, album year, song genre, and link to the track image.

7.1.2 Search songs by year range

```
SELECT
    Song_ID,
    Song_name,
    Artist_name,
    Album_year,
    Song_genre,
    Track_image
FROM Display_results
WHERE Album_year >= '${req.query.startYear}' AND Album_year <=
'${req.query.endYear}'
LIMIT ${pagesize} OFFSET ${offset}
```

The query returns song information including song name, artist name, album year, song genre, track image based on year range (start year and end year).

7.1.3 Search songs by emotional tendency

```
SELECT
```

```

        D.Song_ID,
        D.Song_name,
        Artist_name,
        Album_year,
        D.Song_genre,
        Track_image
FROM Display_results D JOIN Song_info S ON D.Song_ID = S.Song_ID
WHERE
    Acousticness >= '${req.query.minAcousticness}' AND Acousticness
<= '${req.query.maxAcousticness}'
    AND Valence >= '${req.query.minValence}' AND Valence <=
    '${req.query.maxValence}'
    AND Danceability >= '${req.query.minDanceability}' AND
Danceability <= '${req.query.maxDanceability}'
    AND Energy >= '${req.query.minEnergy}' AND Energy <=
    '${req.query.maxEnergy}'
    AND Instrumentalness >=
    '${req.query.minInstrumentalness}' AND Instrumentalness <=
    '${req.query.maxInstrumentalness}'
    AND Tempo >= '${req.query.minTempo}' AND Tempo <=
    '${req.query.maxTempo}'
    AND H_s = '${req.query.happysad}'
LIMIT ${pagesize} OFFSET ${offset}

```

The query returns song information including song name, artist name, album year, song genre, track image based on a series of attributes of emotional tendency.

7.1.4 Get songs from playlist

```

SELECT
    D.Song_ID,
    Song_name,
    Artist_name,
    Album_year,
    Song_genre,
    Track_image
FROM Display_results D
WHERE Song_ID in (select Song_ID from User_likes where username =
    '${req.query.user}')

```

The query returns song information including song name, artist name, album year, song genre, track image if the song exists in the user's playlist.

7.1.5 List songs within decades of users' favorites

```
WITH User_year AS
  (SELECT Album_year, Count(Album_year) AS num
   FROM User_likes U JOIN Display_results D ON U.Song_ID =
D.Song_ID
   WHERE username = '%${req.query.user}%'
   GROUP BY Album_year
   ORDER BY num DESC
   LIMIT 1)
SELECT
  D.Song_ID,
  Song_name,
  Artist_name,
  Album_year,
  Song_genre,
  Track_image
FROM Display_results D, User_year U
WHERE
  D.Album_year >= U.Album_year - 5
  AND D.Album_year <= U.Album_year + 5
  AND D.Song_ID NOT IN (SELECT Song_ID FROM User_likes)
ORDER BY RAND()
LIMIT 5
```

This query returns songs that are within the decade of the songs in users' playlists.

7.2 Most Complex Query

```
WITH User_most_like AS
  (select label, count(1) as song_unit
   FROM User_likes a inner join Song_Classifier b
on a.Song_ID = b.Song_ID
   WHERE username = '%${req.params.user}%'
   group by label
   order by song_unit desc
   limit 1)
  select D.Song_ID, D.Song_name, Artist_name, Album_year,
D.Song_genre, Track_image
  FROM Display_results D inner join Song_Classifier SC on D.Song_ID
= SC.Song_ID
  where D.Song_ID not in (select Song_ID from User_likes)
```



```

        and label in (select label from User_most_like)
        ORDER BY RAND()
    limit 10

```

First we choose what the user most liked category of songs and then random return 10 songs of this category.

8 Query Optimization

8.1 Indexing on Country

Query to be Executed	SELECT D.Song_ID, Song_name, D.Artist_name, Album_year, Song_genre, Track_image FROM Display_results D JOIN Artist_info_dedup Aid on D.Artist_id = Aid.Artist_id WHERE Artist_country = 'JP'
Timing Before Optimization (ms)	102.02625
Timing After Optimization (ms)	50.8515
Why It Works	Originally 9014 rows were retrieved and 10% intermediate results were selected. After the optimization, only 21 rows are retrieved and 100% results are finally selected for country selection.
Challenges Faced	Challenges are trivial as data amount is tolerable.

8.2 Indexing on Year

Query to be Executed	SELECT Song_ID, Song_name, Artist_name, Album_year, Song_genre, Track_image FROM Display_results WHERE Album_year >= 2000 AND Album_year <= 2010
Timing Before Optimization (ms)	73.514

Timing After Optimization (ms)	19.2765
Why It Works	Originally 23807 rows were retrieved and 10% intermediate results were selected. After the optimization, only 2674 rows are retrieved and 100% results are finally selected for year selection.
Challenges Faced	Challenges are trivial as data amount is tolerable.

8.3 Materialized View

Query to be Executed	SELECT Song_ID, Song_name, Artist_name, Album_year, Song_genre, Track_image FROM Display_results
Why It Works	Because most of the song display views on web use above attributes. We created a view and joins several tables in order to acquire these attributes, in order to get the job done once in a lifetime. By doing this, we don't need to join tables again and again everytime we make a query for song display.

9 Technical Challenges

9.1 User Registration and Login

As the most crucial part of our application that will be widely referred to in pages like user recommendation and playlist, user registration and login is implemented with the most efforts for our project implementation.

Unlike simple redirects and information display, we need to store user's input (username, password) in the database for future reference. However, we failed a lot of times to send the request to the server with values of those inputs. In addition, we have encountered network connection (refuse-to-connect) errors frequently even though we managed to transport those values.

We overcame the challenge by leveraging `window.localStorage` to store those values and use `Axios` to post the request to the server. For the network connection, we set up a proxy on the server side to ensure the request can be sent back successfully.

We found that customization of frontend modules brings convenience and difficulties. On one hand, some functionality has been implemented by other developers and we just need to use existing functions and APIs. On the other hand, it will be really hard to debug due to interaction between modules and lack of information about the message returned.