By:

Mingwei Ruan,
Fonda Tsang,
Huaying Gu,
Haoning Gong,

SPOTIHOLIC MUSIC

Team 404 NOT FOUND

Our Goals

1 **Search module:**
to allow user to search songs based on:
- ➔ Song name
- ➔ Artist name
- ➔ Year
- ➔ Genre

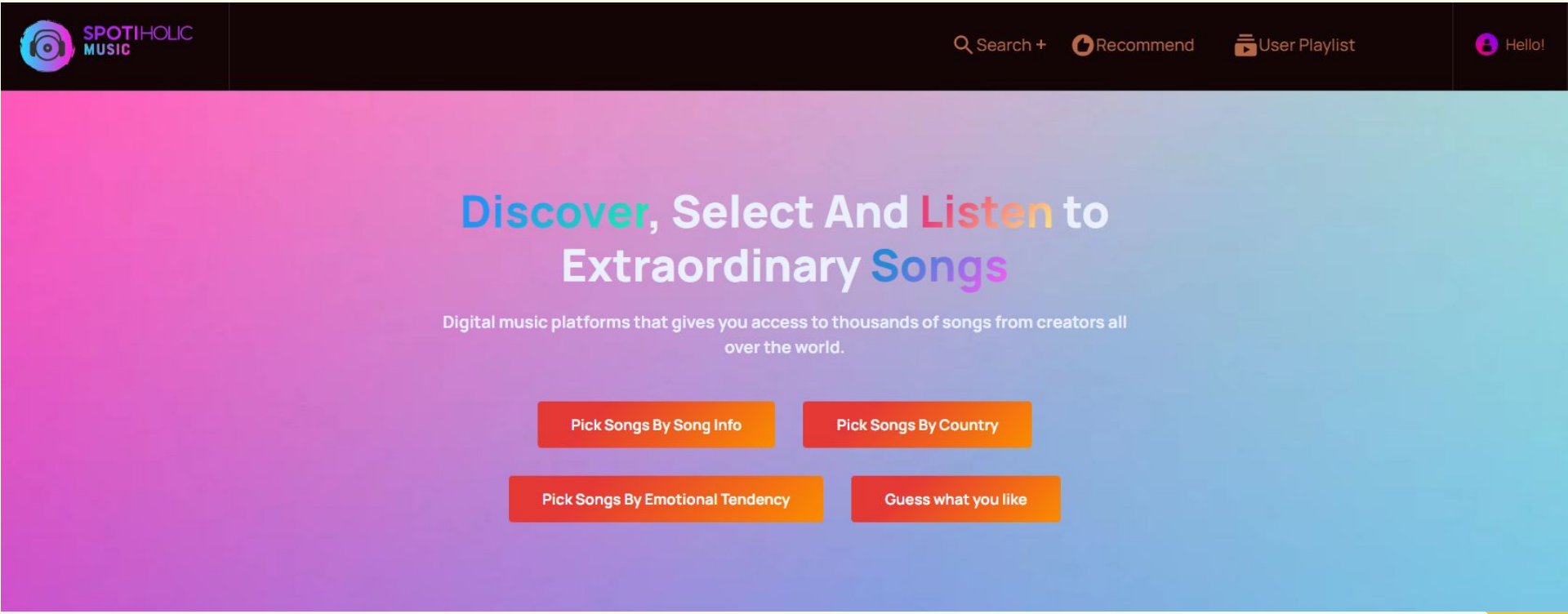
2 **Interactive module**
with a world map for users to select songs based on country.

3 **Filter page**
allow users to filter songs based on the emotional tendency.

4 **Page with a playlist**
defined by users with their selection of favorite songs.

5 **Recommendation**
recommendation module based on users' playlist preferences.

Preview


SPOTIHOLIC MUSIC
Search +
Recommend
User Playlist
Hello!
Discover, Select And Listen to Extraordinary Songs
Digital music platforms that gives you access to thousands of songs from creators all over the world.
Pick Songs By Song Info
Pick Songs By Country
Pick Songs By Emotional Tendency
Guess what you like

Datasets Acquisition

1

Source: Kaggle and Spotify

- Downloaded from Kaggle of 24008 unique songs with different song features and tagged as happy/sad
- Pulled supplementary data(song name, artist name, track image, album released date) using spotify api

2

Source: musixMatch

- Pulled data (song genre, artist country) using musixMatch api. We match songs between the two datasets with song name and artist name.

Data Cleansing

1

Song Features
- Drop duplicates

2

Artist
- Extract year format

3

Genre
- Unwind genres and regenerate list to store genres

4

Song Classifier
- Unsupervised machine learning technique

Data Normalization

Functional Dependencies

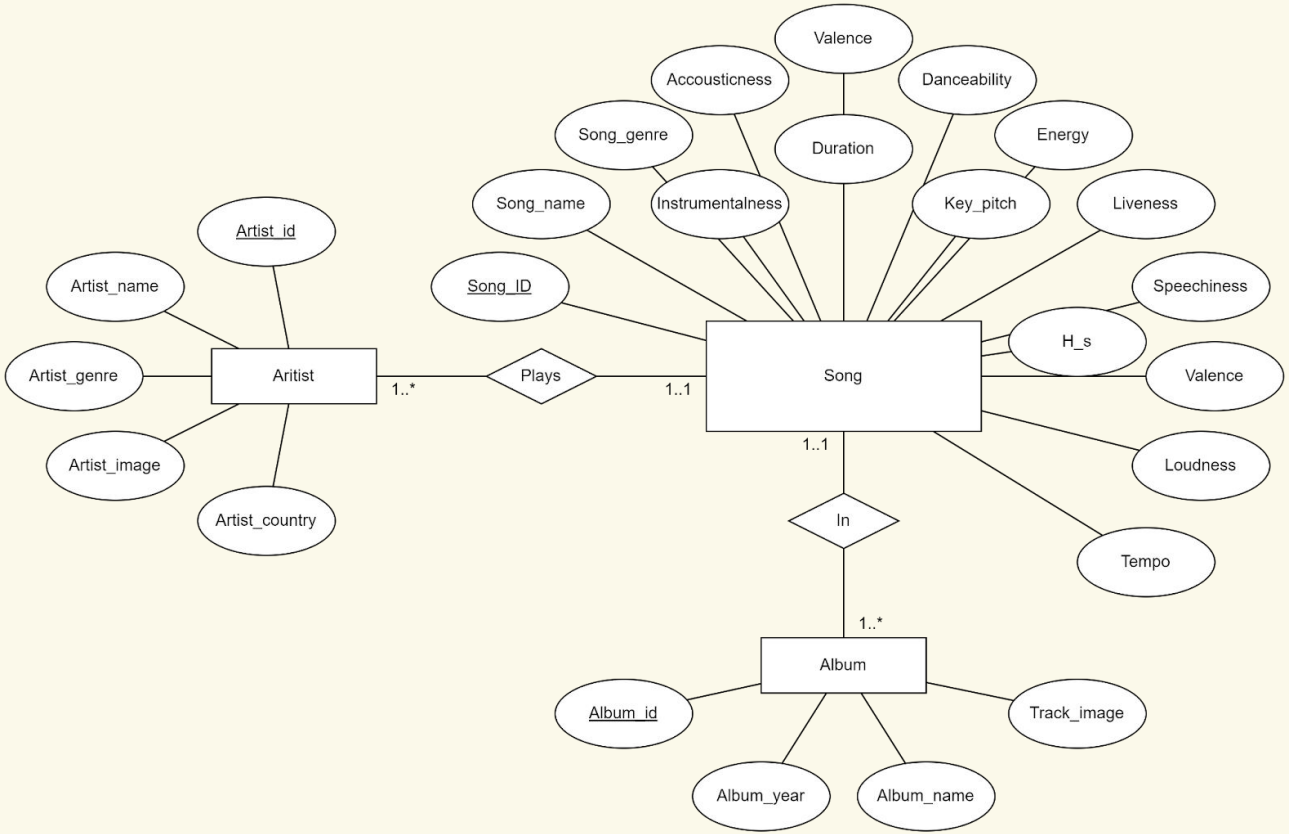
Song_ID → Song_name, Song_genre, Acousticness, Valence, Danceability, Energy, Instrumentalness, Liveness, Loudness, Speechiness, Tempo, Duration, Key_pitch, H_s, Artist_ID, Album_ID

Artist_ID → Artist_name, Artist_genre, Artist_image, Artist_country

Album_ID → Album_id, Album_name, Album_year, Track_image

Username → password, Song_ID (for account storage and user login purpose)

Schema Design (3NF)



Demo

Application Stack

Front-End

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

Back-End

1. Node.js
2. Express
3. MySQL

Example: 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
```

Performance Optimization

Indexing on Country

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.

Performance Optimization

Indexing on Year

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.

Performance Optimization

Materialized View

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 every time we make a query for song display.

Technical Challenges

User Registration and Login

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.

How we overcome

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.

Thank you!