MusicRec: Databases Final Project

Samer Aslan & Owen Bianchi
Sections: 415 & 615
saslan1@jh.edu & obianch1@jh.edu

December 20th, 2023

Application Domain: This report outlines the accomplishments of our final project, focusing on critically acclaimed music albums. We extracted data from RateYourMusic and Spotify, utilizing advanced data scraping and machine learning techniques to generate album recommendations and insights. This project demonstrates work in data extraction, database management, machine learning, and web development.

Deployed Page: Our music recommendation website is fully operational and can be accessed at https://db-project-music.vercel.app/.

1 Introduction: Target Domain

Our project focuses on music album recommendations. We build a database that uses data from RateYourMusic and Spotify to provide users with album recommendations. The database includes many music features for albums, such as mood and audio features, and provides insights on what albums are similar, answering questions about specific albums and artists.

2 Project Overview and Changes

Since Phase I, our project has undergone a few changes, primarily in the scope of data retrieval and database design. Initially, we aimed to include a broader range of data tables; however, due to the time-consuming nature of data retrieval and processing, we focused on a more streamlined database structure. This approach allowed us to delve deeper into the integration of machine learning for album recommendations and front-end development for interesting insights.

3 Data Acquisition and Preprocessing

3.1 RateYourMusic Data Extraction

We sourced our primary dataset from RateYourMusic (RYM), a user-generated site known for its extensive collection of music album reviews and ratings. To extract this data, we utilized a modified version of the `rymscraper` repository available on GitHub. This repository was drastically outdated, leading us to make many changes to make it work. These modifications were crucial in optimizing the scraping process to effectively extract information about the top 5000 albums.

3.2 Data Enrichment with Spotify API

To augment our dataset, we integrated data from the Spotify API. This allowed us to access detailed audio features for each track in the albums obtained from RYM. By averaging these audio features across all tracks in an album, we created a rich dataset that combines subjective RYM descriptors with objective audio features from Spotify.

3.3 Preprocessing and Encoding

The final step in data preparation involved preprocessing the collected data. We focused on ensuring data consistency and accuracy, especially for mood descriptors associated with albums. These descriptors were one-hot encoded. Our encoding strategy also included a weighting system, giving precedence to the primary descriptor of each album over others. This approach enabled us to capture the nuanced characteristics of each album in our dataset.

4 Data Loading and Integration

4.1 Consolidation of Data Sources

This project's success hinged on the effective integration of data from two distinct sources: RateYourMusic for user-generated content and Spotify for objective audio features. The URLs and methodologies used to access these sources are comprehensively documented in our project's repository, specifically the data-retrieval subfolder.

4.2 Database Loading Strategy

The culmination of our data acquisition and preprocessing efforts was the loading of this data into a PostgreSQL database. We developed a specialized Python script using the psycopg2 library to handle this process. The script was designed to perform several critical tasks:

1. Establish a connection to our PostgreSQL database using the credentials provided.
2. Read and process data from the CSV files, which contained the combined data from RYM and Spotify.
3. Execute data insertion into various database tables, such as Artists, Albums, Tracks, and SpotifyAudioFeatures.
4. Utilize the machine learning capabilities of the sci-kit learn library to generate album recommendations. These recommendations, based on the K-nearest-neighbors algorithm, were then systematically stored in the AlbumRecommendations table for efficient access.

This methodical approach not only streamlined the database population process but also laid the groundwork for our music recommendation system.

5 Database Design and Implementation

5.1 Database Architecture

Our database is hosted on Amazon RDS and is based on the PostgreSQL platform. The design includes several key tables: Albums, Artists, SpotifyAudioFeatures, and RYMDescriptors. These tables were thoughtfully structured to capture and relate all necessary data elements for our application.

5.2 Integration with Frontend Technologies

We employed `psycopg2` for database operations, ensuring seamless connectivity and interaction with our frontend technologies. This integration was critical for real-time data processing and user interaction on our web platform.

5.3 Precomputed Album Recommendations

A standout feature of our database is its ability to provide precomputed album recommendations. Leveraging the power of machine learning, specifically the K-nearest-neighbors algorithm from sci-kit learn, we created a system that suggests albums based on a combination of RYM descriptors and Spotify audio features. These precomputed recommendations are stored efficiently in the database, enabling quick and relevant suggestions to users.

6 Frontend Development and User Interface

The frontend of our application was developed using Next.js with React TypeScript. Hosted on Vercel (`https://db-project-music.vercel.app/`), the site offers an intuitive user interface for accessing our data insights.

The primary functionality, showcasing our machine learning application, is accessed through a dedicated section for album recommendations. This GUI, crafted with Chakra UI, highlights our project's unique approach to transcending genre boundaries in album similarity.

Additionally, we provide access to various SQL queries through a simpler UI, enabling users to explore different aspects of our dataset, like identifying the longest albums or artists with the most albums in the top 500.

7 Project Specializations

Our project primarily specializes in three major areas:

1. **Complex Data Extraction:** We successfully extracted and processed data from online sources such as RateYourMusic and Spotify, overcoming challenges like rate limits and data preprocessing.

2. **Innovative User Interface:** Our frontend development showcases significant accomplishment and originality, providing a user-friendly interface to interact with our data and machine learning models.

3. **Knowledge Discovery:** Our backend clustering algorithm that creates the recommendations for albums utilizing machine learning for knowledge discovery.

8 Project Strengths and Unique Aspects

Our project stands out for its:

- Integration of Spotify API for detailed audio features.
- Advanced webscraping techniques for data extraction.
- Application of KNN machine learning for album recommendations.

- Use of React and Prisma ORM for frontend and database connectivity.

9 Challenges and Improvements

The most challenging aspects of our project were:

- Navigating the complexities of the Spotify API and RYM scraping.
- Handling rate limits and data retrieval hurdles.

Potential improvements include expanding our dataset with additional album information and refining our machine-learning model for more nuanced recommendations.

10 Code Organization

Our codebase is divided into two main directories:

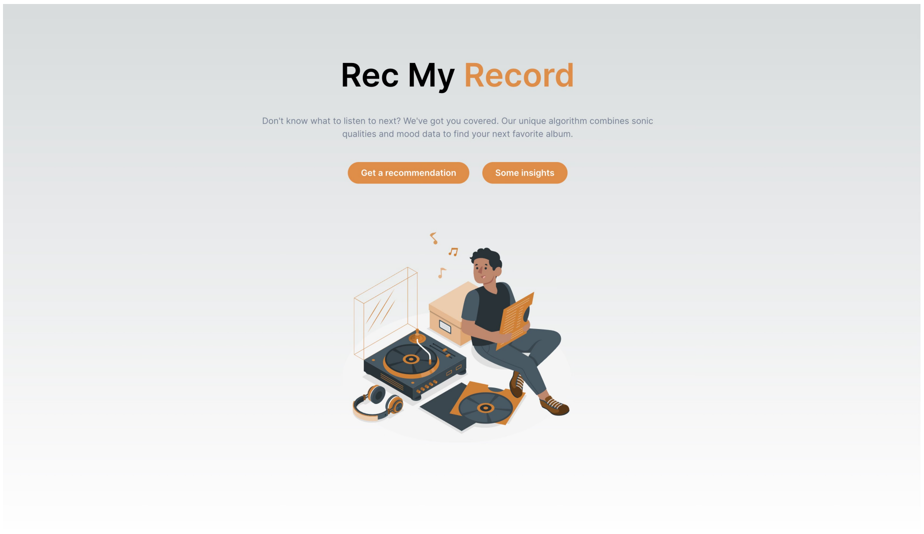
- **frontcreck:** Contains all Next.js API routes, React TypeScript code, etc.
- **data-retrieval:** Includes scripts for scraping RYM, interfacing with the Spotify API, data preprocessing, machine learning application, and database population.

Note that there is no need to run any of the code as the application is fully deployed at the URL referenced in the cover page. If you would like to run this locally, cd frontcreck and run 'npm run dev'. To reproduce any of the results such as filling the database or running the clustering algorithm, the code is well structured and commented so it should be trivial.

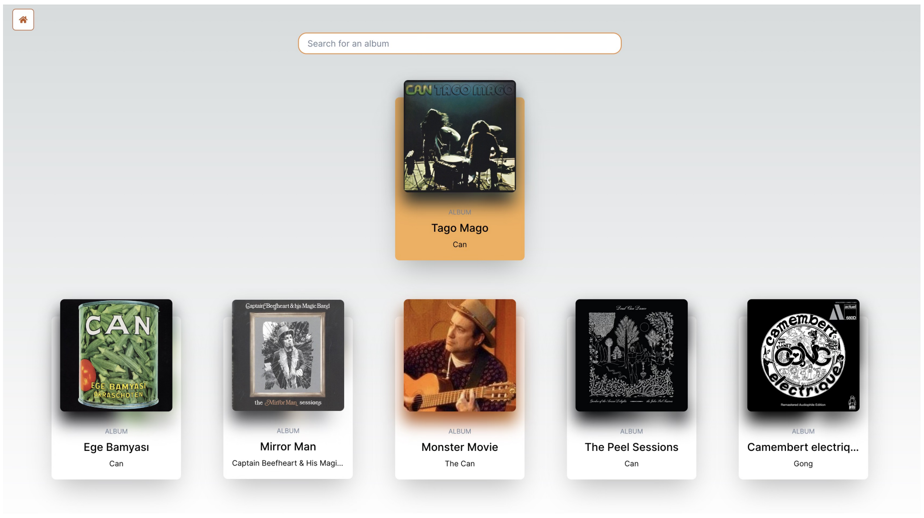
The full relational table specification of our database is in the data-retrieval/create.sql. A copy of the SQL code we initially wrote is in the data-retrieval/queries.sql file. We replicated some of these queries using Prisma in order to display them on the front end of our queries by utilizing Prisma.

11 Final Product

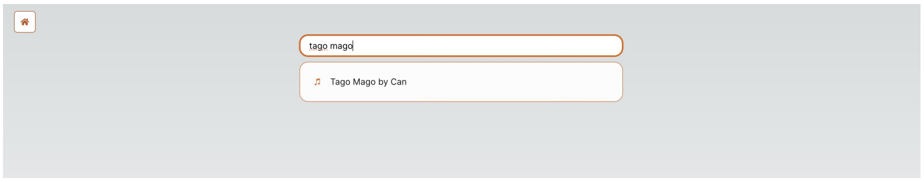
As aforementioned, the final product is a fully deployed live website at the following link https://db-project-music.vercel.app/. In addition to this as its own live demo, we have the following screenshots which showcase some of the abilities.


Rec My Record

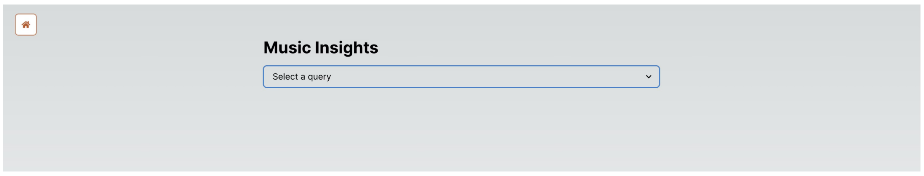
Don't know what to listen to next? We've got you covered. Our unique algorithm combines sonic qualities and mood data to find your next favorite album.

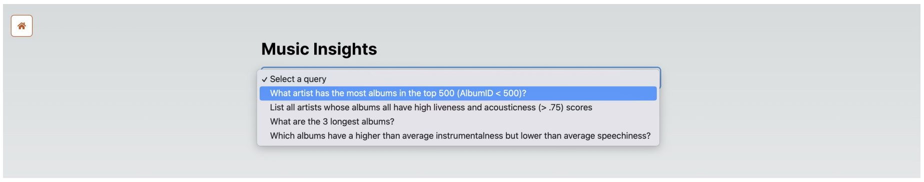


Album	Artist
Tago Mago	Can
Ege Bamyası	Can
Mirror Man	Captain Beefheart & His Magi...
Monster Movie	The Can
The Peel Sessions	Can
Camembert electriq...	Gong


tago mago

Tago Mago by Can


Music Insights


Music Insights
✓ Select a query
What artist has the most albums in the top 500 (AlbumID < 500)?
List all artists whose albums all have high liveness and acousticness (> .75) scores
What are the 3 longest albums?
Which albums have a higher than average instrumentalness but lower than average speechiness?

12 Phase 1

12.1 Target Domain

Our project is focused around a music recommendation system. We're building a database that uses existing data from RateYourMusic, Spotify, and possibly Last.fm, to provide users with album recommendations. We also want the database to be able to provide many insights about why certain music is similar, answering questions about specific genres, albums, songs, etc.

12.2 English Questions for the System

With our data sources, we aim to answer questions such as:

1. How similar are the albums "Abbey Road" by The Beatles and "Long Season" by Fishmans based on their descriptors and audio features?
2. Which artist has the most similar songs to those of Bob Dylan based on audio features?
3. List all tracks in the "Rock" genre released between 1970 and 1980.
4. What are the most common genres for albums released in the 1990s?
5. Find artists who have "Folk" listed as a primary genre and have been active since 2000.
6. Which albums have the most tracks in the "Electronic" genre?
7. What are the average tempo and energy levels of tracks in the "Metal" genre?
8. Find albums with descriptors like "experimental" and "psychedelic" released after 2010.
9. List the tracks with the highest danceability scores according to Spotify's audio features.
10. Which artists have the widest range of genres in their albums?
11. For a specific genre, what are the most common descriptors used in its albums?

12. Identify albums that have a significant number of tracks with a key signature of C major.
13. What are the common descriptors for albums released by artist "Kendrick Lamar"?
14. Determine the average duration of tracks in albums by "Radiohead."
15. Find artists similar to "Lana Del Rey" based on genre and audio features.
16. List the albums that contain tracks with a tempo greater than 120 BPM.
17. What are the genres of the top 10 longest duration albums in the database?
18. Which albums released in the last 5 years have the highest average energy level according to Spotify's audio features?

12.3 Relational Data Model Design

SQL Schema

```
CREATE TABLE Artists (
    ArtistID INT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Origin VARCHAR(100),
    ActiveYearsStart YEAR,
    ActiveYearsEnd YEAR,
    SpotifyID VARCHAR(255),
    LastFMID VARCHAR(255),
    GenreID INT,
    FOREIGN KEY (GenreID) REFERENCES Genres(GenreID)
);

CREATE TABLE Genres (
    GenreID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Description TEXT
);

CREATE TABLE Albums (
    AlbumID INT PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    ReleaseYear YEAR NOT NULL,
    GenreID INT,
    ArtistID INT,
    LabelID INT,
    Rating FLOAT,
    TotalTracks INT,
    SpotifyID VARCHAR(255),
    LastFMID VARCHAR(255),
    FOREIGN KEY (GenreID) REFERENCES Genres(GenreID),
    FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID)
);

CREATE TABLE Tracks (
    TrackID INT PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    AlbumID INT,
    Duration INT,
    TrackNumber INT,
    SpotifyID VARCHAR(255),
    LastFMID VARCHAR(255),
    FOREIGN KEY (AlbumID) REFERENCES Albums(AlbumID)
);

CREATE TABLE RYM_Descriptors (
    DescriptorID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Description TEXT
);

CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    SignUpDate DATE NOT NULL,
    LastActiveDate DATE,
    PreferenceID INT,
    LastFMUsername VARCHAR(50),
    FOREIGN KEY (PreferenceID) REFERENCES UserPreferences(PreferenceID)
);

CREATE TABLE UserFavorites (
    FavoriteID INT PRIMARY KEY,
    UserID INT,
    TrackID INT,
    AlbumID INT,
    ArtistID INT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (TrackID) REFERENCES Tracks(TrackID),
    FOREIGN KEY (AlbumID) REFERENCES Albums(AlbumID),
    FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID)
);
```

```
CREATE TABLE ListeningHistory (
    HistoryID INT PRIMARY KEY,
    UserID INT,
    TrackID INT,
    Timestamp DATETIME NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (TrackID) REFERENCES Tracks(TrackID)
);

CREATE TABLE UserPlaylists (
    PlaylistID INT PRIMARY KEY,
    UserID INT,
    Name VARCHAR(100) NOT NULL,
    CreationDate DATE NOT NULL,
    PublicPrivateIndicator BOOLEAN NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

CREATE TABLE Playlist (
    PlaylistID INT,
    TrackID INT,
    FOREIGN KEY (PlaylistID) REFERENCES UserPlaylists(PlaylistID),
    FOREIGN KEY (TrackID) REFERENCES Tracks(TrackID)
);

CREATE TABLE Reviews (
    ReviewID INT PRIMARY KEY,
    UserID INT,
    AlbumID INT,
    Content TEXT,
    Rating INT,
    Date DATE NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (AlbumID) REFERENCES Albums(AlbumID)
);

CREATE TABLE UserSpotifyFollowing (
    FollowingID INT PRIMARY KEY,
    UserID INT,
    FollowedUserID INT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (FollowedUserID) REFERENCES Users(UserID)
);

CREATE TABLE SpotifyAudioFeatures (
    FeatureID INT PRIMARY KEY,
    TrackID INT,
    Tempo FLOAT,
    Key INT,
    Energy FLOAT,
    Danceability FLOAT,
    FOREIGN KEY (TrackID) REFERENCES Tracks(TrackID)
);

CREATE TABLE ArtistInfluences (
    InfluenceID INT PRIMARY KEY,
    ArtistID INT,
    InfluencedArtistID INT,
    FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID),
    FOREIGN KEY (InfluencedArtistID) REFERENCES Artists(ArtistID)
);

CREATE TABLE SampledMusic (
    SampleID INT PRIMARY KEY,
    TrackID INT,
    SampleSource VARCHAR(255),
    SampleTimestamp TIME,
    FOREIGN KEY (TrackID) REFERENCES Tracks(TrackID)
);

CREATE TABLE UserPreferences (
    PreferenceID INT PRIMARY KEY,
    UserID INT,
    GenreID INT,
    DescriptorID INT,
    PreferredArtists TEXT,
    PreferredAlbums TEXT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (GenreID) REFERENCES Genres(GenreID),
    FOREIGN KEY (DescriptorID) REFERENCES RYM_Descriptors(DescriptorID)
);

CREATE TABLE ContextualData (
    ContextID INT PRIMARY KEY,
    UserID INT,
    ContextType VARCHAR(50),
    Preference TEXT,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

CREATE TABLE AlbumRecommendations (
    RecommendationID INT PRIMARY KEY,
    AlbumID INT,
    RecommendedAlbumID INT,
    SimilarityScore FLOAT,
    FOREIGN KEY (AlbumID) REFERENCES Albums(AlbumID),
    FOREIGN KEY (RecommendedAlbumID) REFERENCES Albums(AlbumID)
);

-- Insert statements examples
INSERT INTO Genres (GenreID, Name, Description) VALUES
(1, 'Rock', 'A genre of popular music...');

INSERT INTO Artists (ArtistID, Name, Origin, ActiveYearsStart, GenreID) VALUES
(1, 'The Beatles', 'Liverpool, England', 1960, 1);

INSERT INTO Albums (AlbumID, Title, ReleaseYear, GenreID, ArtistID) VALUES
(1, 'Abbey Road', 1969, 1, 1);

INSERT INTO Tracks (TrackID, Title, AlbumID, Duration, TrackNumber) VALUES
(1, 'Come Together', 1, 259, 1);
```

12.4 Representative SQL Statements for Target Queries

```
Finding Similar Albums Based on Descriptors and Audio Features
SELECT a1.Title AS Album1, a2.Title AS Album2, AVG(saf.SimilarityScore) AS Similarity
FROM Albums a1
JOIN AlbumRecommendations ar ON a1.AlbumID = ar.AlbumID
JOIN Albums a2 ON ar.RecommendedAlbumID = a2.AlbumID
JOIN SpotifyAudioFeatures saf ON a2.AlbumID = saf.AlbumID
WHERE a1.Title = 'Abbey Road' AND a1.ArtistID =
(SELECT ArtistID FROM Artists WHERE Name = 'The Beatles')
  AND a2.Title = 'Long Season' AND a2.ArtistID =
  (SELECT ArtistID FROM Artists WHERE Name = 'Fishmans')
GROUP BY a1.Title, a2.Title;

Artist with Most Similar Songs to Bob Dylan
SELECT a.Name, COUNT(*) AS SimilarSongs
FROM Artists a
JOIN Tracks t ON a.ArtistID = t.ArtistID
JOIN SpotifyAudioFeatures saf ON t.TrackID = saf.TrackID
WHERE saf.FeatureID IN (SELECT saf.FeatureID
                        FROM SpotifyAudioFeatures saf
                        JOIN Tracks t ON saf.TrackID = t.TrackID
                        JOIN Artists a ON t.ArtistID = a.ArtistID
                        WHERE a.Name = 'Bob Dylan')
GROUP BY a.Name
```

```
ORDER BY SimilarSongs DESC
LIMIT 1;

List of Rock Tracks from 1970-1980
SELECT t.Title, a.ReleaseYear
FROM Tracks t
JOIN Albums a ON t.AlbumID = a.AlbumID
WHERE a.GenreID = (SELECT GenreID FROM Genres WHERE Name = 'Rock')
  AND a.ReleaseYear BETWEEN 1970 AND 1980;

Most Common Genres in 1990s Albums
SELECT g.Name, COUNT(*) AS AlbumCount
FROM Genres g
JOIN Albums a ON g.GenreID = a.GenreID
WHERE a.ReleaseYear BETWEEN 1990 AND 1999
GROUP BY g.Name
ORDER BY AlbumCount DESC;

Artists with Folk Genre Active Since 2000
SELECT Name
FROM Artists
WHERE GenreID = (SELECT GenreID FROM Genres WHERE Name = 'Folk')
  AND ActiveYearsStart >= 2000;

Albums with Most Tracks in Electronic Genre
SELECT a.Title, COUNT(t.TrackID) AS TrackCount
FROM Albums a
JOIN Tracks t ON a.AlbumID = t.AlbumID
WHERE a.GenreID = (SELECT GenreID FROM Genres WHERE Name = 'Electronic')
GROUP BY a.Title
ORDER BY TrackCount DESC
LIMIT 1;
```

12.5 Data Loading Plan

We've already scraped data from RateYourMusic, collected data via the Spotify API, and are considering additional sources like Last.fm. Our next step is to convert the data from the tabular format to relational database format. Specifically, we want to populate the relational schema we provided.

12.6 Output or Result Type and User Interface Issues

The main output on the frontend will be, given an album, get X most similar albums, and display them nicely in the GUI. However, our database will also be able to answer much more complex questions, and if time permits, we'll attempt to get these working on the frontend as well.

12.7 Specialized/Advanced Topics Focus

Our main advanced topic is the GUI for the frontend, to give the users a proper experience for getting similar albums. We're also considering scheduling a machine learning algorithm (likely K-Nearest-Neighbors) to run on our database and automatically update the AlbumRecommendations table, and are wondering if this is considered an additional advanced topic. Otherwise, for our minor advanced topic, we're considering natural language interfaces or data mining if we manage to expand our dataset.

Additionally, does a React frontend constitute as a GUI, or are we limited to the MySQL web interface or Perl-based data-extraction from text or on-line data sources?