EECS 447 Course Project: Goop

Zai Erb, Nicholas Nguyen, Chinh Nguyen

Contents

1 Introduction

Goop is an application that allows users to view, save, share, and find songs, albums, DJ Mixes, and artists. Its primary feature is its fine-tuned searching and cataloging of music based on any user-given parameters. The database allows users to see specific information about music such as release date, record label, performing artists, DJ mixes, and more. For example, if a user wants to display every DJ mix that contains a specific song or artist or if they want to see a record label's discography from a specified timeframe.

Each user account with their profiles and account information are stored in the database. Additionally, users are able to save and organize their music. Users are allowed to follow each other in addition to artists, labels, genres. This service is primarily useful for DJs/performers who are "crate digging" or finding new music to play in their mixes. Being able to fine-tune search results based on adjacent information (such as which mixes a song was played in or the labels an artist is on) in a collection will assist in organizing record libraries in external applications like Rekordbox, Serato, or VirtualDJ.

The concept for Goop was born from frustrations we have had with the existing options for music cataloging. There are lots of options but all of them have significant drawbacks, the biggest of which is that they all have a narrow scope, designed to focus on one specific aspect of music cataloging and categorization, the other issue that is ubiquitous with these sites is a severely outdated, often hard to navigate UI. Rate Your Music and Album of the Year exist for rating, reviewing, and cataloging tracks and albums but neither of them provide tracklists for DJ mixes and while Rate Your Music has more data, depth, and flexibility the UI is far worse than the simpler but cleaner Album of the Year site. For getting tracklists of DJ mixes there are a few options such as 1001Tracklists, TracklistsDB and Tracklists.net all provide similar functionality but their interfaces are outdated and the integration of the tracklists with other information such as the album tracks are from, their release dates, their associated record labels, the genre of the individual tracks etc. For finding events in your area and finding electronic music recommendations Resident Advisor is fantastic however it is limited mostly to electronic music and the mixes posted on the site do not have tracklists. All of this makes for a frustrating and tedious experience for those that want to dig for music using one platform with a database that connects Albums, Genres, Mixes, Labels, Artists, Events and more.

The biggest challenge for creating an application like this is populating the database. A lot of the aforementioned sites rely on community contribution and/or integrate with streaming services such as spotify to ensure new music is added to the database. Seeing as implementing solutions like this would not be feasible for our timeline we relied on scraping data from some of these existing sites to test our database schema and user interface. If we were to continue with this project we would enable to user contributions and could potentially integrate with streaming services to constantly update the database with newly released music.

In this report we will detail what we cover what our app is capable of currently, how it was designed including how we constructed our database and built the front and backend. We also cover plans to expand the functionality in the future.

2 Design

2.1 SQL

SQL code to create tables and views in database hosted on phpMyAdmin

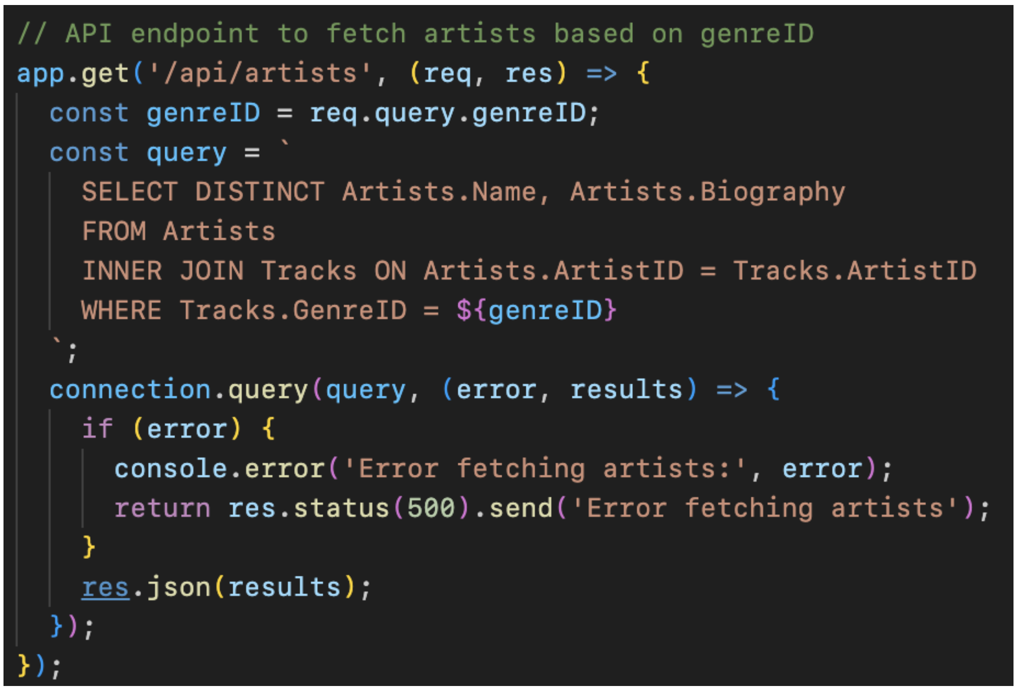
- SQL used to initialize our tables - *data_handling/sql/CreateTables.sql* is the SQL used to initialize our tables
- SQL used to initialize views - *data_handling/sql/CreateViews.sql* makes querying our database easier especially when there are additional tables used to connnect tables such as the Albums table which references the AlbumGenre table which stores album genres and subgenres.

2.2 PHP Backend Server

Our backend: *api/akiserver.js* facillitates interaction with the database, this is where our queries are happening
We use the following queries:

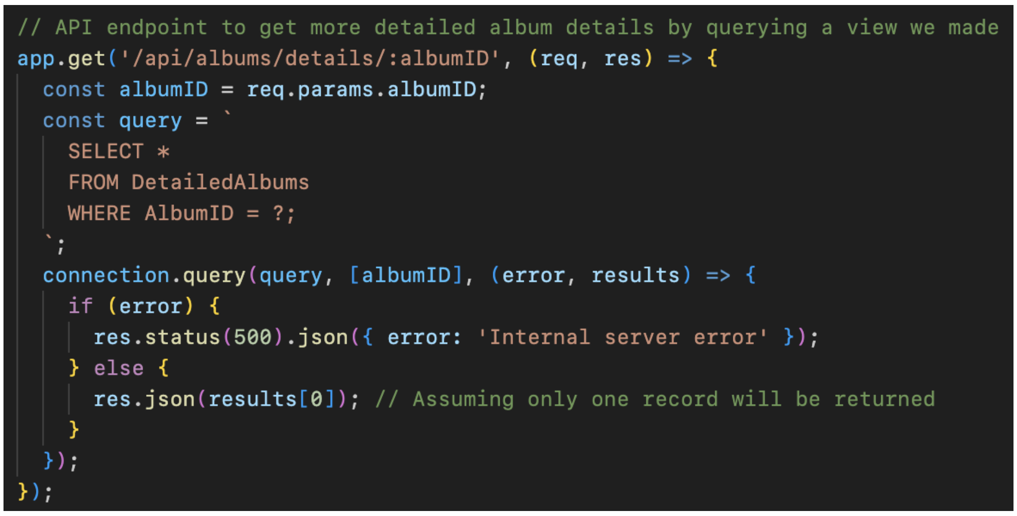
2.2.1 Queries

- Fetching Artists by GenreID:



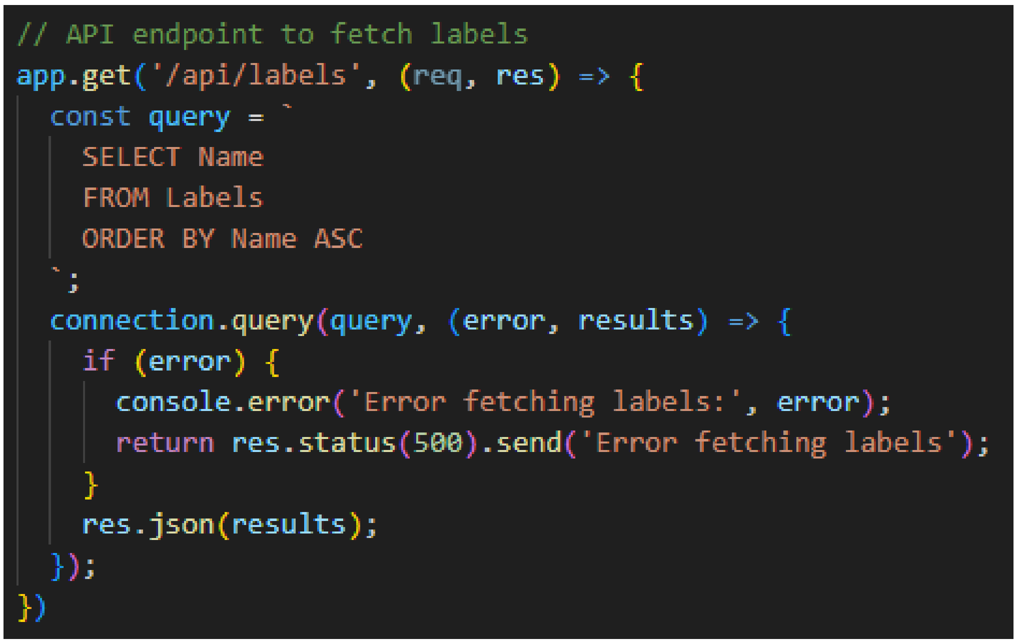
```
// API endpoint to fetch artists based on genreID
app.get('/api/artists', (req, res) => {
  const genreID = req.query.genreID;
  const query = `
    SELECT DISTINCT Artists.Name, Artists.Biography
    FROM Artists
    INNER JOIN Tracks ON Artists.ArtistID = Tracks.ArtistID
    WHERE Tracks.GenreID = ${genreID}
  `;
  connection.query(query, (error, results) => {
    if (error) {
      console.error('Error fetching artists:', error);
      return res.status(500).send('Error fetching artists');
    }
    res.json(results);
  });
});
```

- Fetching Albums Information



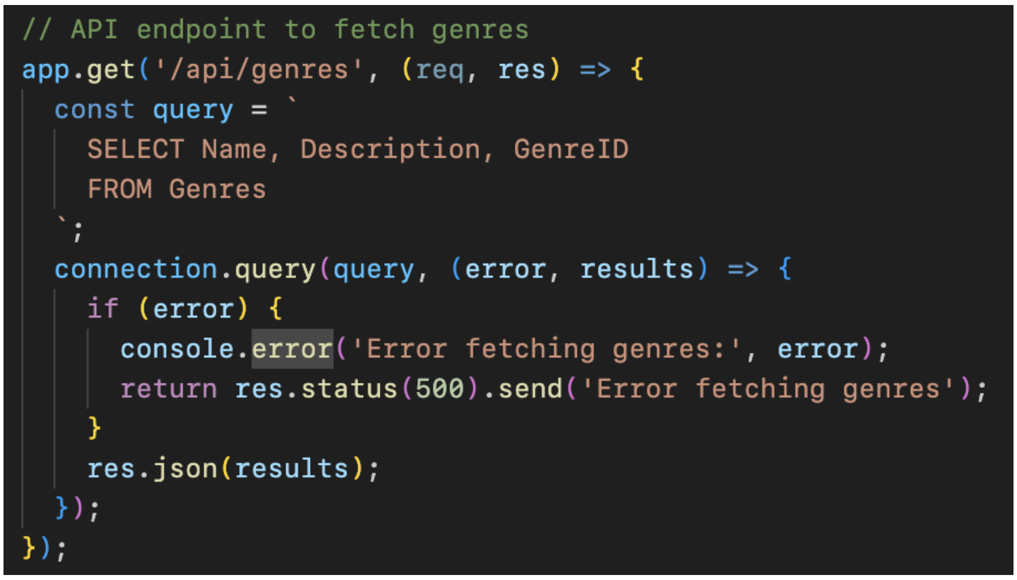
```
// API endpoint to get more detailed album details by querying a view we made
app.get('/api/albums/details/:albumID', (req, res) => {
  const albumID = req.params.albumID;
  const query = `
    SELECT *
    FROM DetailedAlbums
    WHERE AlbumID = ?;
  `;
  connection.query(query, [albumID], (error, results) => {
    if (error) {
      res.status(500).json({ error: 'Internal server error' });
    } else {
      res.json(results[0]); // Assuming only one record will be returned
    }
  });
});
```

- Fetching Labels



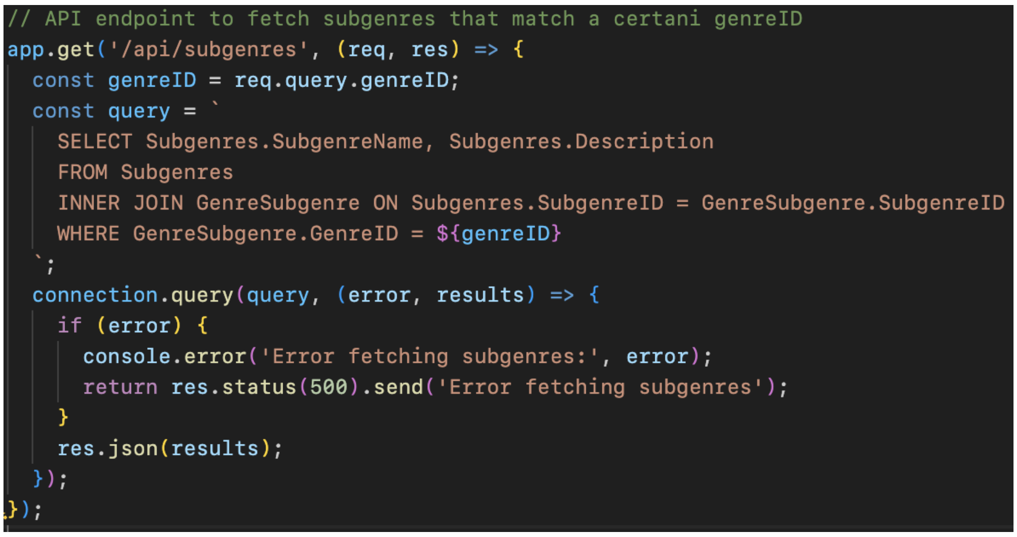
```
// API endpoint to fetch labels
app.get('/api/labels', (req, res) => {
  const query = `
    SELECT Name
    FROM Labels
    ORDER BY Name ASC
  `;
  connection.query(query, (error, results) => {
    if (error) {
      console.error('Error fetching labels:', error);
      return res.status(500).send('Error fetching labels');
    }
    res.json(results);
  });
})
```

- Fetching Genres



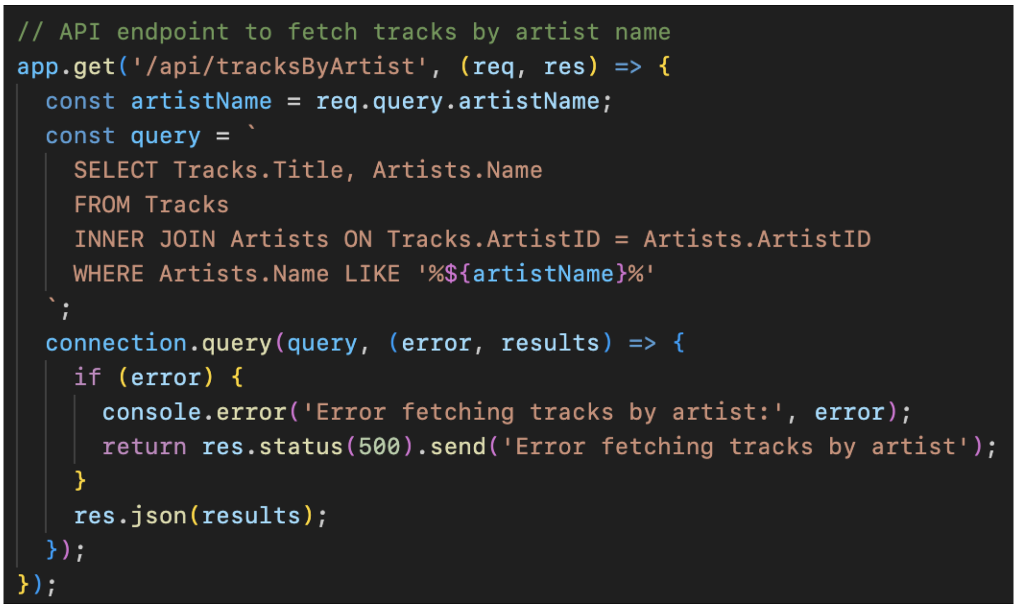
```
// API endpoint to fetch genres
app.get('/api/genres', (req, res) => {
  const query = `
    SELECT Name, Description, GenreID
    FROM Genres
  `;
  connection.query(query, (error, results) => {
    if (error) {
      console.error('Error fetching genres:', error);
      return res.status(500).send('Error fetching genres');
    }
    res.json(results);
  });
});
```

- Fetching Subgenres by GenreID



```
// API endpoint to fetch subgenres that match a certani genreID
app.get('/api/subgenres', (req, res) => {
  const genreID = req.query.genreID;
  const query = `
    SELECT Subgenres.SubgenreName, Subgenres.Description
    FROM Subgenres
    INNER JOIN GenreSubgenre ON Subgenres.SubgenreID = GenreSubgenre.SubgenreID
    WHERE GenreSubgenre.GenreID = ${genreID}
  `;
  connection.query(query, (error, results) => {
    if (error) {
      console.error('Error fetching subgenres:', error);
      return res.status(500).send('Error fetching subgenres');
    }
    res.json(results);
  });
});
```

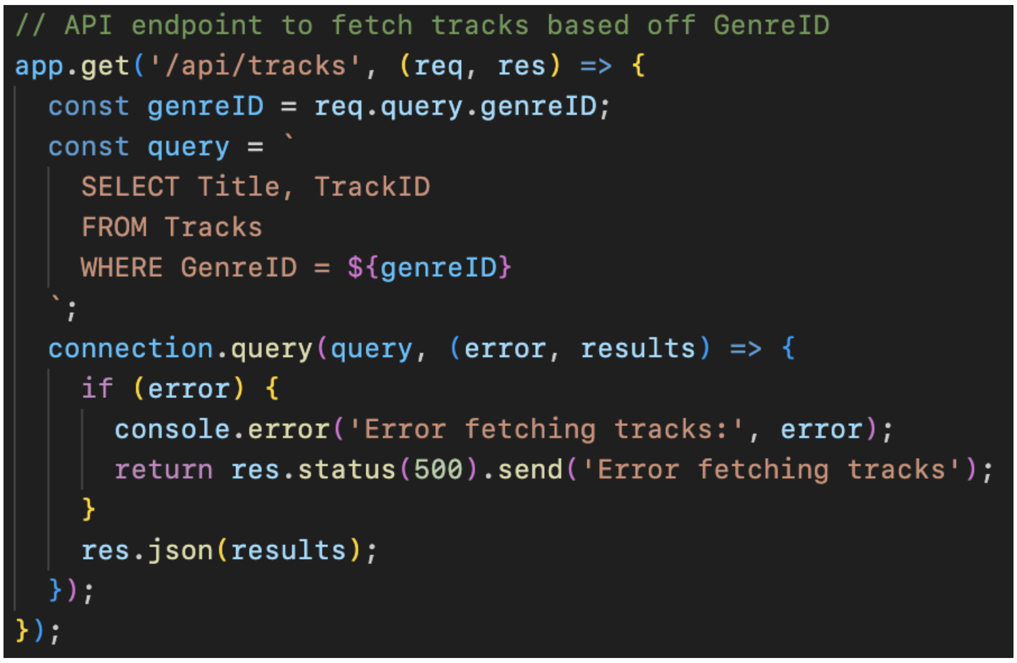
- Fetching Tracks by Artist Name

```javascript
// API endpoint to fetch tracks by artist name
app.get('/api/tracksByArtist', (req, res) => {
  const artistName = req.query.artistName;
  const query = `
    SELECT Tracks.Title, Artists.Name
    FROM Tracks
    INNER JOIN Artists ON Tracks.ArtistID = Artists.ArtistID
    WHERE Artists.Name LIKE '%${artistName}%'
  `;
  connection.query(query, (error, results) => {
    if (error) {
      console.error('Error fetching tracks by artist:', error);
      return res.status(500).send('Error fetching tracks by artist');
    }
    res.json(results);
  });
});
```

  –

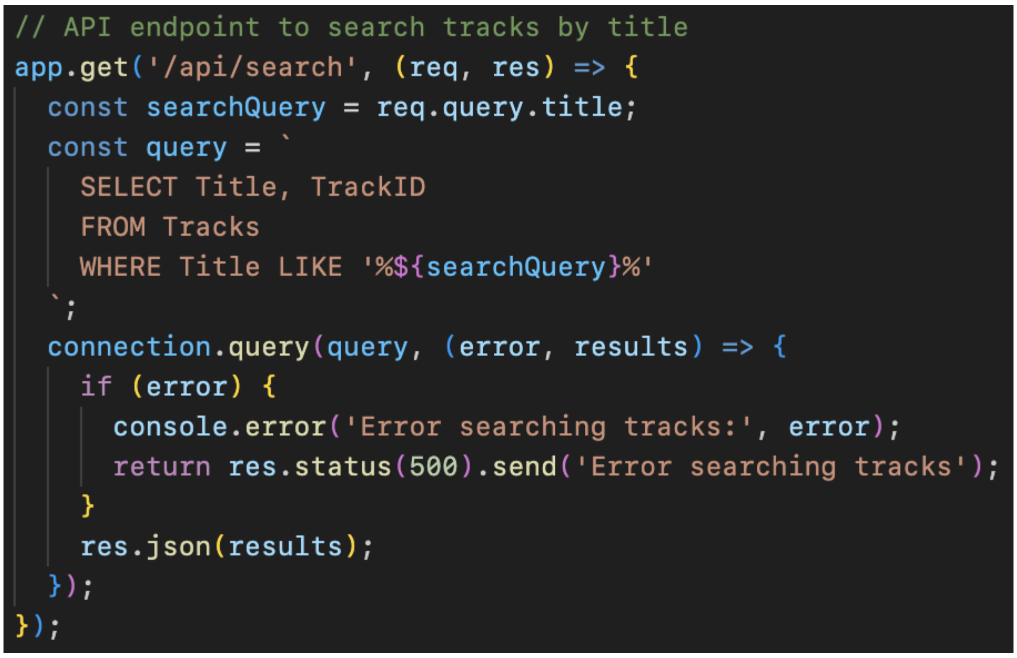
- Fetching Tracks by GenreID

```javascript
// API endpoint to fetch tracks based off GenreID
app.get('/api/tracks', (req, res) => {
  const genreID = req.query.genreID;
  const query = `
    SELECT Title, TrackID
    FROM Tracks
    WHERE GenreID = ${genreID}
  `;
  connection.query(query, (error, results) => {
    if (error) {
      console.error('Error fetching tracks:', error);
      return res.status(500).send('Error fetching tracks');
    }
    res.json(results);
  });
});
```

  –

- Fetching Tracks by Title

– 

```
// API endpoint to search tracks by title
app.get('/api/search', (req, res) => {
  const searchQuery = req.query.title;
  const query = `
    SELECT Title, TrackID
    FROM Tracks
    WHERE Title LIKE '%${searchQuery}%'
  `;
  connection.query(query, (error, results) => {
    if (error) {
      console.error('Error searching tracks:', error);
      return res.status(500).send('Error searching tracks');
    }
    res.json(results);
  });
});
```

2.3 Python Data Scraping Scripts

Python scripts used for webscraping

- Main Scripts:
  - *GenresScraper.py* reads the data from our mostly manually generated JSON files containing genres and subgenres.
  - *SetRaAlbumns.py* imports the data cleaned by raScraper into our tables.
- Scraper to collect data from resident advisor using GraphQL SQL Queries: *data/handling/scripts/raScraper* The python scripts use json files as templates for their queries.
  - *album_detail_fetcher.py* queries individiual album details
  - *album_fetcher.py* queries lists of albums by genre.
  - *album_parser.py* calls *album_detail_fetcher* and *album_fetcher* scrapes all of the raw album data and then cleans the data to prepare it to add to our database

2.4 React App

We built a react application to handle the front end which is responsible for the UI and asking the backend to communicate with our database

- Component Design All of our pages are built using the same general component design. Let's use albums as an example. There is an *albumPage.js* which renders the necessary *DataGrid.js* components for each category matching the current filter. So if for example genre is the filter *AlbumPage.js* creates a *DataGrid* component for each genre. Each grid is loaded onto the page as a dropdown titled according to its category which when clicked reveals grid of *Album.js* components. While all of the categories follow this same general structure, the filters and thus the queries used are slightly different for each category.
- Search and Filter
  - Search functionality allows user to query on any of the tables
  - Filters vary by page allowing users to implement filters on the page they are on. To get entries matching specific parameters.
- Genre page
  - Displays genres as clickable dropdown menus to display subgenres belonging to a genre. Each subgenre shows its description when you hover over them.

- Artists page
  - Displays artists according to filters set by the user: options to filter by genre, year and label.
- Mixes Page
  - Displays DJ Mixes according to filters set by the user: options to filter by recommended, genre, year and label.
- Albums Page
  - Displays Albums according to filters set by the user: options to filter by recommended, genre, year and label.
- Current Features
  - Interactive pages built on Genres, Albums, Artists, Mixes and Labels tables.
  - Filter and search by album, genre, artist, track, label, year and more.
- Future Features
  - Users database that allows users to follow other users as well as any other category within the database.
  - Ability for users to add entries.
  - Integration with streaming services to regularly and automatically update the database.
  - Tree view to more easily visualize connections.

2.5 Testing

Although we did not implement any test suites or frameworks, if we were to continue the project we would. For the scale of our project we were able to test our queries using the query interface on myPHP admin and then gradually test each component involved in a page in conjuction with the queries they used. We used debugging tools as well as including extensive error checking to make it easy to trace bugs in our code.

3 Design Analysis

3.1 Constraints

3.1.1 Entity-Relationship Constraints

- Each artist, genre, subgenre, label, album, track and mix has a unique ID

3.1.2 Referential Integrity Constraints

- All track, albums, and mixes have foreign keys which link them to genres, subgenres, artists, and labels.
- Subgenres have a foreign key to link them to genres
- Artists have foreign keys to link them to genres, subgenres, and labels.

3.1.3 Consistency Constraints

- Any updates to an item that appears in multiple tables are reflect across all tables containing the item.

3.2 Operations

3.2.1 Create Operations

- Add new artists, genres, labels, albums, tracks, and DJ mixes.

3.2.2 Read Operations

- Retrieve information about users, artists, genres, labels, albums, tracks, and DJ mixes.
- Fetch lists of albums, tracks, and DJ mixes belonging to specific artists, genres, or labels.
- Fetch recommended Albums and Mixes.

3.2.3 Update Operations

- Modify artist information, genre details, label information, album details, track details, and DJ mix details.

3.2.4 Delete Operations

- Remove artists, genres, labels, albums, tracks, and DJ mixes from the database.

3.2.5 Search Operations

- Search for users, artists, genres, labels, albums, tracks, and DJ mixes based on various criteria (e.g., name, genre, publication date).
- Perform advanced searches, such as finding tracks by artist or album name, or finding users with similar tastes.

3.2.6 Aggregate Operations

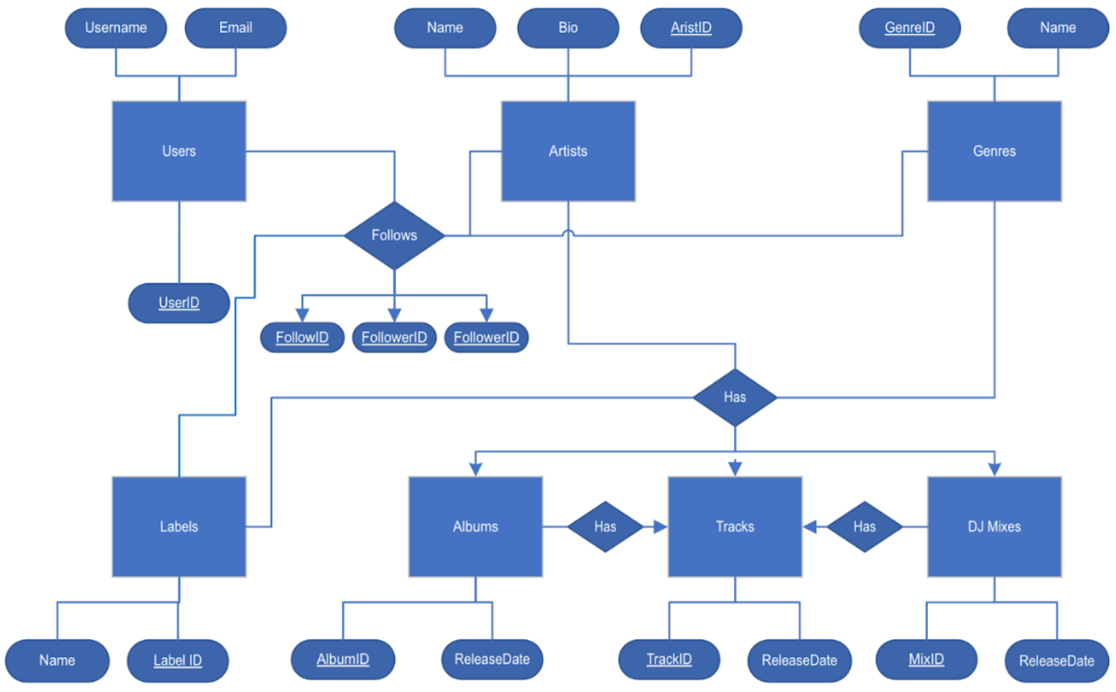
- Aggregate data, such as counting the total number of albums, tracks, or DJ mixes in the database.

3.2.7 Transactional Operations

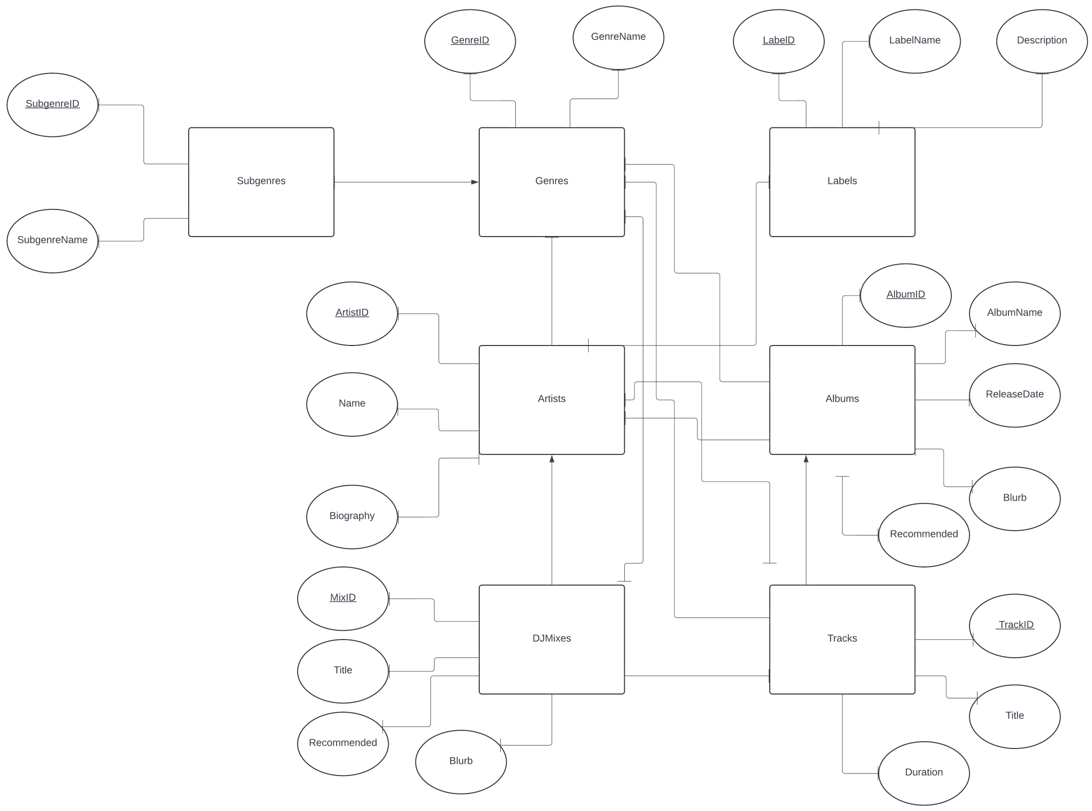
- Execute transactions to maintain data consistency and integrity (e.g., updating multiple entities atomically).

4 System Architecture

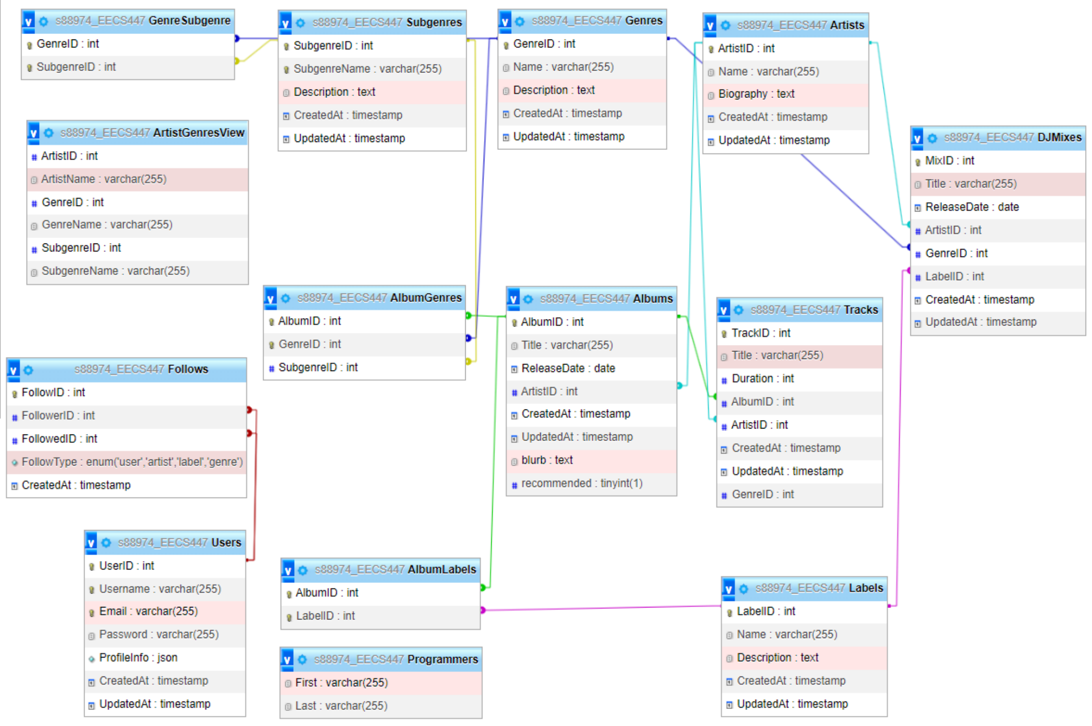
4.1 Original Entity Relationship Diagram



4.2 Current Entity Relationship Diagram



4.3 Relational Schema



5 Project Log

5.1 General Responsibilities

- Zai Erb:

  - Wrote the reports
  - Handled implementation of web scrapers
  - Created SQL views and cleaned up database format after importing data
  - Handled designing the react application components and much of the styling
  - Handled adding additional queries to the *api/server* to get the filters for the various pages working.

- Chinh Nguyen
  - Implemented the server used in the backend to reference the database through phpMyAdmin
  - Implemented and tested many of the queries used for searches and filters
  - Implemented a search page to test all of the searches/queries

- Nicholas Nguyen:
  - Assisted in research for building webscrapers
  - Wrote the SQL to create the initial tables
  - Helped to style the frontend and modify component behavior
  - Created the project presentation

5.2 Specific Tasks

The below tables more specifically detail what each group member did.

Contributor	**Task**	**Time Spent**
Zai Erb	Outlined and wrote first report	2-3 hours
Zai Erb, Nicholas Nguyen	Initialized React App and outlined component design	2 hours
Zai Erb	Experimented with webscraping for various sites	5 hours
Zai Erb, Chinh Nguyen, Nicholas Nguyen	Manually created JSON files for genres and subgenres	4 hours
Zai Erb, Chinh Nguyen	Implemented data cleaner to put JSON genre files into DB	1 hour
Zai Erb, Nicholas Nguyen	Attempted to implement 1001 tracklists scraper	3 hours
Zai Erb, Nicholas Nguyen	Updated styling	2 hours
Zai Erb	Modified database to make reference tables for foreign keys	30 minutes
Zai Erb	Implemented resident advisor scraper and data cleaner	6+ hours
Zai Erb	Updated backend and created references to display the front end according to filters on the page	
Nicholas Nguyen	Began early front end work/design	2 hours
Nicholas Nguyen	Created initial SQL queries and attempted to implement them in the backend	4 hours
Nicholas Nguyen, Chinh Nguyen	Set up back end server and database for the project	2 hours
Nicholas Nguyen	Edited project video together, fixed audio for recordings.	4 hours
Chinh Nguyen, Nicholas Nguyen	Created project video slides and script	2 hours
Chinh Nguyen	Implemented and tested many of the queries used for searches and filters	4 hours

Table 1: Contributions and Time Allocation

Author	Date	Message
Zai Erb	2024-05-06	Added Report and Todo List
Zai Erb	2024-05-06	Added Report and Todo List
Zai Erb	2024-05-02	Added SQL views to simplify queries
Zai Erb	2024-05-02	Improved Scraper
Zai Erb	2024-04-29	Updated Styles
Zai Erb	2024-04-29	Added resident advisor scraper
Zai Erb	2024-04-18	styling
Chinh Nguyen	2024-04-18	Navigation bar now reflects the current page when site is refreshed
Zai Erb	2024-04-18	Search Styling
Zai Erb	2024-04-17	Styling
Chinh Nguyen	2024-04-17	STYLING!
Chinh Nguyen	2024-04-17	Fixed fetching tracks by artistName, doesn't have to be exact match now
Zai Erb	2024-04-17	Styling
Chinh Nguyen	2024-04-17	Cleaned up CSS, made scrollable, fixed naming
Chinh Nguyen	2024-04-17	Added search for tracks by artist name
Chinh Nguyen	2024-04-17	Added fetching tracks by artist name
Zai Erb	2024-04-17	Updated styling
Zai Erb	2024-04-17	Added GenreClick functionality for all pages.
Zai Erb	2024-04-17	Added all pages to sort items by Genre
Chinh Nguyen	2024-04-17	Implemented search artists by genreID
Chinh Nguyen	2024-04-17	Added 5th query with JOIN: Get artists based off GenreID
Chinh Nguyen	2024-04-17	Results Textarea
Zai Erb	2024-04-17	Genre Components
Chinh Nguyen	2024-04-17	Updated to include Search page
Chinh Nguyen	2024-04-17	Two new queries: Search Tracks based off genreID and title
Chinh Nguyen	2024-04-17	Basic Search Page
Zai Erb	2024-04-17	Added 1001 scraper. Not working
Chinh Nguyen	2024-04-17	Added more function
Chinh Nguyen	2024-04-17	Added Subgenre fetching using INNERJOIN on GenreID and SubgenreID
Chinh Nguyen	2024-04-17	Subgenre Fetch Function + Dropdown Divs
Chinh Nguyen	2024-04-17	Added query for Subgenre table: SubgenreName and Description
Chinh Nguyen	2024-04-17	Oops. Fixed reading of config.json
Nicholas Nguyen	2024-04-17	Fixed Genre alignment (along with other elements) fixed weird redundancies too shared between each file. added heart easter egg too because why not
Chinh Nguyen	2024-04-17	Necessary Packages Required, Added Localhost with Port 3001 as Proxy (IMPORTANT FOR BACKEND)
Chinh Nguyen	2024-04-17	Connected to the backend and displays "Genres" table with "Name" and "Description" column
Chinh Nguyen	2024-04-17	Working starter backend – run with "node akiserver.js" in a second terminal, make sure that no other program is running port 3001
Chinh Nguyen	2024-04-17	Password Leak – Rotated Password / See Discord
Zai Erb	2024-04-16	Updated Genres Page - functional for presentation
Zai Erb	2024-04-16	Updated components and genre jsons
Zai Erb	2024-04-16	Updated JSON format
Zai Erb	2024-04-16	Fixed gitignore merge conflict
Zai Erb	2024-04-16	data changes
Zai Erb	2024-04-16	Data Changes
Nicholas Nguyen	2024-04-16	Furthered Front-End Design/Development
Nicholas Nguyen	2024-04-15	Add files via upload
Zai Erb	2024-04-15	Updated Scraper
Nicholas Nguyen	2024-04-15	Create .gitignore
Zai Erb	2024-04-15	Added raw data for genres in JSON files and added script to convert JSON into SQL tables
Zai Erb	2024-04-03	Added react project files
Zai Erb	2024-04-03	first commit

Table 2: Git Commits