Names: Tim Fitzpatrick, Nicholas Newton, Dalton Dove, Hye Soo Jeon

Database Concepts - Final Project Report

Muse (GitHub: https://github.com/tfitzpatrick0/Databases-MuseWebApp)

What was accomplished

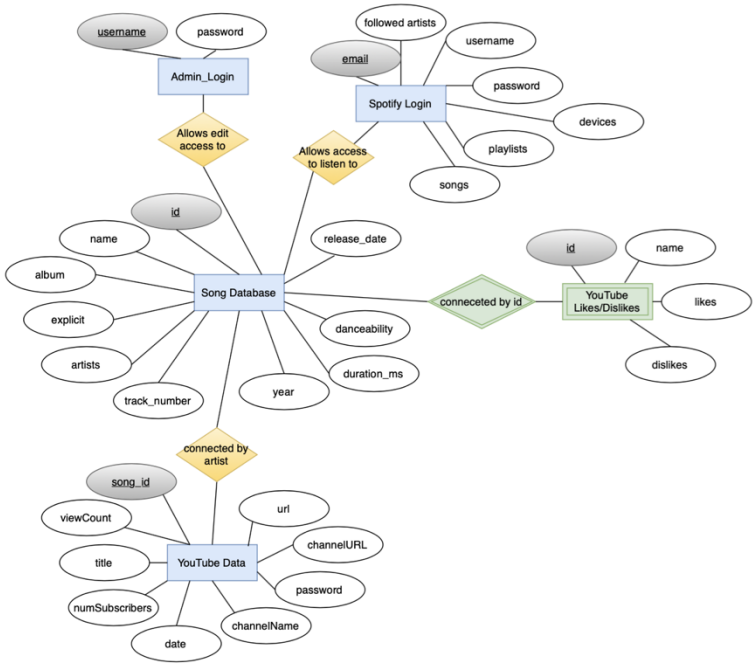
Project Muse offers a platform on which distinct musical properties of songs are presented and allows users to learn more about the music of their interest. Users can use Muse to search for songs and view musical attributes and visualize metrics of a song. Users can also log in to their personal Spotify accounts to listen to the song, update their playlists (add/delete songs), follow artists, and more - in real time. Muse supports functionalities to search, update, delete, and insert songs that administrators need to maintain the user platform.

Usefulness

Muse is an interactive application through which users can learn more about their musical taste by exploring new songs/artists, learning about songs/artists, and receiving recommendations. It facilitates and simplifies this process by displaying properties about a selected song in visual representations through which users can discover their music preferences. To improve accuracy and provide up-to-date info for the users, administrators can change and update song searches, and these changes are reflected on the user's platform promptly.

Data Usage

ER Diagram [Updated from Original]



Schema [Updated from Original]

Song(spotify_id, song_name, album, artist, track_number, explicit, danceability, energy, loudness, speechiness, acousticness, instrumentalness, valence, liveness, tempo, duration_ms, year, full_release_date)

Admin_login(username, password)

Spotify_login(email, password, username, followed artists, devices, playlists, songs)

YouTube Likes/Dislikes(name, id, likes, dislikes)

YouTube Data(song_id, viewCount, title, numSubscribers, date, channelName, password, channelURL, url)

Database Collection

The database table **track3** was downloaded from Kaggle's Spotify 1.2M+ Songs. It is comprised of the following attributes: general information - song id (the primary key), song name, album, album id, artists, explicitness, and duration, as well as characteristics pertaining to music style - danceability, acousticness, energy, tempo, etc. Within the app, we used PHP to display the explicitness as 'Yes' or 'No, because the **track3** table represents the explicitness of a song as '1' or '0' respectively. The database **youtube_video_data** has been obtained from Kaggle's YouTube Music Data and is populated by the attributes music/artist(s) channel name, channel URL, artist/group name, number of likes, number of dislikes, and number of subscribers. The last database used was **login_data** acquired from Kaggle's Bruteforce Database - Password dictionaries. The different tuples of username and password from this data were used to create dummy accounts of administrators and users in the testing process. In order to facilitate a table join, we also created a csv using python/pandas to match song ids from Spotify to hypothetical likes and dislikes on a given song for songs not included in **youtube_video_data** (this database contained much less information than **track3**, and we wanted a consistent display for demo purposes). We also created a secondary table, **youtube_like_dl**, to specifically contain like/dislike values for songs, and randomly generated like/dislike values for songs not contained in the original **youtube_video_data** dataset.

Functionalities

Search - retrieves a table of columns song name, song id, artist, album name, duration, and year of release; the query is the same on the user page and admin page. On the user page, the song name is hyperlinked to a separate page with song specific visual representations of attributes from the databases track3 and youtube_video_data. The search has an adjustable table/page size and navigation through pages at the bottom of the table. There is also a filter option to search anything that shows up in the table results.

Update (admin) - by providing the unique song id and the track number to be updated, the associated track number is updated for that song id in the database track3. We only implemented

the ability to modify the track number for demo purposes. Updates are reflected/can be tested instantly through Search.

Delete (admin) - by providing the unique song id, entry represented by that song id is removed from the database track3. Deletes are reflected/can be tested instantly through Search.

Insert (admin) - new song tuples can be inserted to the database track3 by providing the new song name, album name, artist, and release date. We only implemented the ability to insert these 4 fields for demo purposes. Inserts are reflected/can be tested instantly through Search.

Basic Function Call (Search)

The following is an SQL call made in our PHP code:
SELECT * FROM track3 WHERE name = '$song_name'

This call retrieves the tuple(s) from track3 database with the corresponding song name that the user provided ($song_name = $_POST['searched_song_name'] from the code captures the user input into *song_name* variable). The *mysqli_query* function is then used to take the active mySQL connection and the above SQL call to save the resulting tuple(s) in the *$query* variable. The *$query* variable is then passed into the *mysqli_fetch_assoc* function to iterate over each tuple of the query through looping. Each attribute of the current tuple is accessed through indexing by attribute name - *name, id, album,* and *artists*, respectively - to be returned in a tabulated data for the user.

Using the Application

The first thing to look at is the admin components. From the 'Admin' tab, you will be prompted to sign in as an admin. When you sign in as an admin user, you will be redirected to the demo page, which will allow you to search for a song, update/insert/delete a song from the database and add a new admin account. If you try to access the demo page without logging in as an admin, you will be redirected back to the login page.

- admin username: admin
- admin password: admin_password

To access the full interactive user features, you will want to navigate to the 'Spotify' tab to sign into Spotify (note: without signing into Spotify you will still be able to use the application without having access to the additional Spotify-related features).

Once logged in, you will be redirected to the home/search page. You can input a song name into the search bar and click search or 'enter' on your keyboard. After a couple of seconds, a table will populate (it may take a little longer for the first search; the song database is indexed on the song id, not name).

When you click on a song, you will be directed to the song page. If you try to access the song page with an empty or invalid song ID, you will be redirected back to the home page. The header

for the song page displays the song name, artist, and a ‘return to search’ button. On the page, there are also 4 panels that support different interactive functionality. The top left panel, ‘DEVICES’, allows the user to select a device connected to their Spotify account and play/pause from our app. The top right panel, ‘PLAYLISTS’, allows the user to view their playlists and decide whether to add/remove the song. These two panels require that the user be signed in to Spotify to use. The bottom left panel, ‘SONG ATTRIBUTES’, displays song attributes to the user and allows the user to visualize metrics and explore similar songs. The bottom right panel, ‘SONG INFORMATION’, displays general song information to the user and allows the user to follow the artist with Spotify, redirect to the song on Spotify’s web app, and search Youtube for a music video.

Through our testing, we found that some of the songs in our Spotify database are no longer available on Spotify. This means that some features like playback and similar artists are not always available. For a benchmark test, we tested all of these features working on *Shape of You* by Ed Sheeran.

Advanced Functions

OAuth for Spotify

An important component of the project involved the user signing in to be able to access data from Spotify. To successfully implement this into our app, we had to take the time to learn about the OAuth authentication flow and establishing a secure connection through the Spotify API using the redirect URI, access tokens, and refresh tokens. This process was implemented in the file *app.js*.

Search Result Tabulation

For the search results on the home page, we wanted a clean display that could paginate our results and filter our results. This involved learning to load content with ajax in JavaScript and adding an additional function to convert the time duration from ms to minutes:seconds. The data results were retrieved using *request_handler.php*, and the response was passed back to be rendered on the home page.

Song Page Charts

The song page involves three main graphics. The bar chart involved integrating data retrieved using a MySQL query from the **track3** table into a Chart.js graph. The pie chart followed a similar pattern, but used data from the **youtube_like_dl** table. Lastly, the chart displaying similar songs involved writing an API call to first retrieve similar artists, and parsed these artists into an array which we used to make a subsequent call to retrieve the top song from each artist.

Spotify Media Player and Additional Functions

The media player was a challenge to implement because it involved accessing the API to play the song as well as detect a device to play the music from. Another challenge to this was that the

album id and track number (offset by 1, because indexing started at 0) needed to be queried from the database in order to locate the song. Among other features, we were able to find a method to add/delete songs to owned playlists. These functions involved a significant amount of code to manage the requests in app.js to achieve the interface displayed on the webpage. It should be noted that some API calls were implemented in the testing stage but did not make it into the final application (these can be viewed by loading the muse/testing directory).

Technical Challenge

Because *Muse* was designed to support both the user and administrator versions, it was a challenge to distinguish between shared and restricted functionalities between each version. For example, the *Search* functionality is shared between the two versions, but the user version includes the hyperlink to a separate song attributes' analyses page whereas the administrator version does not (the user version is the home page where the administrator version is a simple display). The *Delete*, *Update*, and *Insert* functionalities are only provided to the administrator version. Administrators are also given permission to create another admin account once they login with a correct admin credential. The administrator version can be accessed through the login system implemented on *Muse*'s backend, and the user version can be accessed through the Spotify OAuth framework. To maintain the administrator version's login system, *Muse* maintains a separate database for administrator username and password pairs.

One big issue that we ran into while adding and fixing functions was that the JavaScript files we were using did not always update on our web browser. This resulted in a lot of confusion when the page was not detecting new functions because the cache and local storage was full/using the old script. Knowing to clear this ahead of time would have saved a lot of time.

Development Process

Our group was able to get the project functioning up to most of the original goals that were set out. It was able to meet all the basic and advanced requirements for the project, but we personally fell short on only one of the features we had planned on implementing. The search and song page is functioning with all of the available features working once the user has logged into their Spotify account. The administrator settings and login credentials are also functioning correctly. The only feature we did not have time to implement was an additional metric on the song page to display if a song received any awards from RIAA. This feature would have involved scraping the song page from the RIAA website using the artist and song name in the URL query. We did not have time to learn web scraping and implement it into our tech stack (most tools would involve using python) on top of building the page and learning ajax, Chart.js, Spotify API, and OAuth. Despite the lack of this small missing feature, our group was very satisfied with the end product.

One other thing to note is that we were limited by the Spotify API because the application created is still in Developer Mode. Because of this, only specified users can successfully access the Spotify features through OAuth, so anyone trying to use the application would need to request their account to be authorized to interact with the Spotify API.

Division of Labor

Tim Fitzpatrick

- project leader, finalized deliverables and organized group meetings
- implemented OAuth for Spotify authentication with refresh tokens
- designed the overall layout for the website
- worked on designing the style sheet for the website, specifically the body
- wrote functions needed to access Spotify API including:
    - media playback
    - find devices
    - playlist search, add/delete song
    - following artists
    - retrieving top songs and related artists
- implemented Spotify functionality onto song page
- worked on delete functionality for song database

Nicholas Newton

- worked on designing the style sheet for the website, specifically the navbar
- input csv files into MySQL database and wrote search queries needed
- worked on admin page and prevented user access to admin pages via url (redirect to login)
- set up admin pages and navigation through them to add mods and update database
- implemented graphs for song page
- final debugging and final project report

Dalton Dove

- worked on admin login frontend
- learned ajax to implement table on search homepage with filter and pagination
- integrated database information into website with MySQL through PHP
- worked on update functionality to song database for admin

Hye Soo Jeon

- worked on managing sessions for admin login
- integrated adding admin and displaying admin users
- worked on insert functionality to song database for admins
- set up initial search functionality that displayed results from MySQL queries
- final project report