Names: Nicholas Newton, Tim Fitzpatrick, Dalton Dove, Hye Soo Jeon
Database Concepts Final Project Report

Muse

Project Website: http://db.cse.nd.edu/cse30246/muse/
YouTube Link: https://youtu.be/B5DpJHDTtXM
Zip Link: http://db.cse.nd.edu/cse30246/muse.zip

What was accomplished

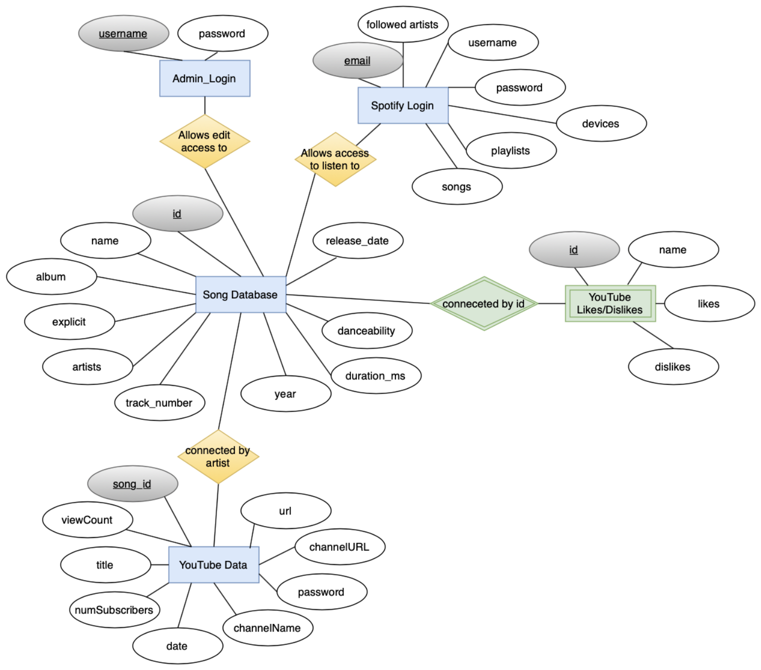
Project Muse offers a platform on which distinct musical properties of songs are presented and allows users to learn more about the music of their interest. Users can access their personalized accounts on which they can search, listen, view musical attributes and visual analyses of popularity of a song. They may also update their playlists - add or delete a song - in real time. Muse also supports functionalities to search, update, delete, and insert songs that administrators need to maintain the user platform.

Usefulness

Muse acts as an informative medium through which users learn about their musical taste and receive song recommendations. It facilitates and simplifies this process by displaying select song properties in visual representations through which users can discover their preferred patterns of music from the playlists built. To improve accuracy and promote currentness in the user's learning process, 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 tables track3 and trackexp were downloaded from Kaggle's Spotify 1.2M+ Songs. These tables hold the same data, but trackexp was revised to correctly handle the csv information on the explicitly of a song. It is comprised of the attributes song id (the primary key), song name, album, album id, artists, characteristics pertaining to music style - such as danceability, explicit, acousticness - and duration. 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. This table was joined with trackexp to provide the general song information as well as the pie chart's data.

Functionalities
Search - retrieves a chart of columns song name, song id, artist, album name, duration, and year of release. This chart is displayed for both user's and administrator's version of Search. On the user's version, the song name would be hyperlinked to a separate page with song specific visual representations of attributes from the databases tracks_features and youtube_video_data. The user search also 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 - by providing the unique song id and the track number to be updated with, the administrator can modify the new track number associated with that song. Updates are reflected/can be tested instantly through Search.

Delete - by providing the unique song id, the administrator can delete the corresponding song data from tracks_features database. Deletes are reflected/can be tested instantly through Search.

Insert - new song tuples can be inserted to the database tracks_features by providing the new song name, album name, artist, and release date. Inserts are reflected/can be tested instantly through Search.

Basic Function Call
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). *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. *query* variable is then passed into *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.

Operating the Website (most of this is demonstrated in the project video)
The first thing to look at is the admin components. From the admin tab, you will be prompted to sign in as an admin, or to add a new admin moderator. If you sign in as an admin user, you will be taken to the page called demo.php which will allow you to search for a song, and update/insert/delete a song from the database. One thing to note is that demo.php will redirect you back to the login until you have correctly signed in on the browser.

admin username: admin
admin password: admin_password

This password can also be used to add a new admin account. This will take you to the mods.php page where current admins are displayed and new ones can be added. Like demo.php, this redirects until add_admin.php sends the signal that the user has logged in.

Next, you will want to click the 'login' button to sign into Spotify for the additional website features. Because we have a Developer Application for the Spotify OAuth/API, you will need to use the following login credentials (we do not have your account as an authorized user):
username:
password:

Once logged in, you should be redirected to the home/search page. You can type 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; song database is indexed on the song id, not name). The table size can be adjusted, as well as navigated through different pages with buttons at the bottom of the table. You can keyword search anything that is populated in the table using the filter box in the upper right of the table.

When you click on a song, you will be directed to the song page. This page has a button to return to search at the top. When you are signed into Spotify, you will be able to click refresh devices and select a device to play media from. To play the music and detect a device, you can open the

Spotify Web Player on a separate tab and sign in with the same credentials (devices also works on other laptops with spotify open as well as smartphone devices when the app is open). Once the device is detected/selected, you can press play/pause to start and stop the song from the beginning.

You can also refresh playlists and display the songs in the playlist. Once refreshed, you are able to click on the playlist and select different playlists. Because you can only add songs to playlists owned by the user, select the MUSE playlist from the playlists. It should be populated already. You can add a song and when it is added, refreshing the songs/playlists will show this update. The same will occur for deleting the song from the playlist. You can add the same song multiple times, but deleting it will delete all instances in the playlist.

There are also buttons to take you to the spotify webpage, and go to a youtube search on the artist. Below this is general song information obtained from a table join.

The button get song attributes will show the different song attributes of the song which can be displayed when the bar chart is created. The pie chart will display the likes and dislikes with a legend in the upper right.

The last button 'Show Similar Songs' will show songs that are similar to the one selected as well as the artist of the song in a table. This feature works for most songs still working on Spotify

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. OAuth required time to learn about the authentication process as well as passing the uri, access tokens, and refresh tokens in app.js and song.js.

Search Result Tabulation

For the search results on the homepage, 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 called by a requestHandler to get the data, and the response was passed back to be parsed.

Song Page Charts

The song page involves three main graphics. The bar chart involved integrating Spotify API data into a Chart.js graph learning how to implement the graph in the correct display. The pie chart followed a similar pattern, but used data from a MySQL query that was compiled into a list for easy access to the data. Lastly, the bottom chart displaying similar songs involved writing a call to fetch similar songs from other artists for the user to discover more songs.

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 song.js to achieve the results on the webpage.

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 *Delete*, *Update*, and *Insert* functionalities are only provided to the administrator version. Administrators are also given permission to create another moderator/admin account once they login with a correct admin credential. To restrict users from accessing the administrator version, and vice versa, a separate login process was implemented. 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 in real time.
Another issue was found during the debugging process. Dealing with Spotify's API and OAuth required refresh tokens and a significant set of functions in a javascript file. One issue 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 of the requirements for the project but 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 you will need to use my Spotify Login until another account is manually added to the list of 25 available users.

Division of Labor

Nicholas Newton

- project leader, finalized deliverables and organized group meetings
- frontend styling for website, including 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)
- designed overall project and web page layouts
- 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

Tim Fitzpatrick

- implemented OAuth for Spotify authentication with refresh tokens
- 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

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