Topic and Purpose

My original topic was GeoTweet, which involved a visualizer of world leaders' tweets across a world map. I'll explain below, but I had to pivot my site twice since then due to API obstacles. I have kept a theme of global culture/insight by now instead of having my website centered around a global music playlist that the user can scroll through. The music is called World Music, and it is curated by a random Spotify user, however it is the highest rated world-genre playlist.

Here is a link to it: https://open.spotify.com/playlist/0q6c4fBoe3XFqkPgHo01KT

How It Works, (+) Data

The main page (M2) is the homepage of the website. On that page, basic info about the site can be found. However, if user wants to access the API-driven program, they have to make an account.

Making an account

Making an account is straightforward. Enter the proper information when clicking on the "sign up button". If not entered correctly, several different warning prompts will be displayed. After successfully creating an account, you will automatically be redirected to the program.php file where the music playlist is dynamically displayed.

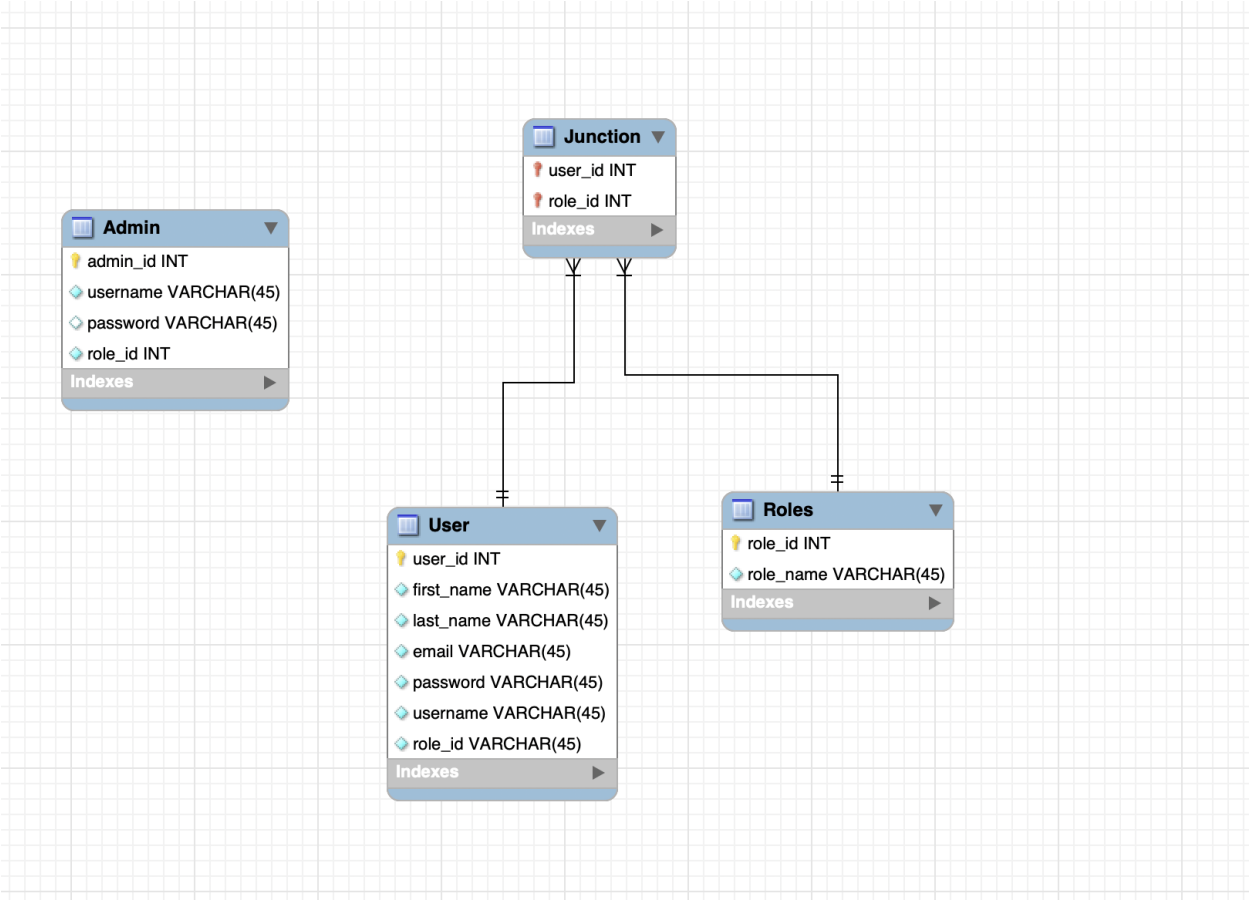
Signing into your account

Same logic as making an account. Will check if your password/username exists in the database. On top of that, if the account you are logging into has a special role assigned to it, AKA 'admin' instead of 'user' (role_id is equal to 2 instead of 1), the username/password will be checked to see if it exists in the Admin database. If it is, you will be redirected to the admin page.

Admin page

The admin page prints all of the information for each row, and then displays a form of each record so that the admin can edit any of the information. **NOTE*** in order to visually see the updates when clicking a submit button for a specific row, you must refresh the page after the page reloads to see the updated table display. The same applies for when you click the delete button to delete a certain row.

Database Design:



User: first name, last name, email, password, username, and role_id. User_id is the primary key and is automatically incremented.

Roles: role_name represents the type of role that the user has. Right now, there are only two roles, 1 being a regular user and 2 being an admin. Role_id is the primary key and is automatically incremented.

Junction: for the future, junction is used to facilitate the relationships between how many roles a user has and how many users categorize under a single role (many-to-many relationship). Right now, it is just configured at a one-to-many relationship for the sake of the assignment, however I designed it like this for a proof of scalability to show that one can add multiple different user roles and users are capable of obtaining more than one role.

Admin: admin_id is an auto incremented primary key. It has its own unique role_id category for its own special labeling system that can be used for future scalability. It possesses a username and password, so that users can be tested if they have admin privileges in the login portal.

*NOTES ABOUT ADMIN:

A new admin account cannot be created within the application itself. You must use the one I already created, which has the username ‘admin’ and password ‘Admin123!’. This can also be viewed in the sql database. You can add your own admin account into the database if you’d like. However you will need to create a new account, fill out the entire form, and make sure that the password and username are the same password and username that you input into the admin database. Remember, all users despite roll are stored in the Users database. Only admins specifically are additionally stored in the Admin database for cross-testing security.

How the API script works:
Every time the program.php page is opened, using my API account a new access token is generated. Access tokens last for an hour however I didn’t want there to be problems in case an access token expired, which is why I decided to generate a new token each time. With the access token generation, I query through the spotify playlist that I shared above to get the returned JSON object. I manipulate the object by only pulling the information that I deem useful. I use cURL for making the server-side requests.

API Documentation: https://developer.spotify.com/documentation/web-api

EXTRAS:

- Rest-API: the Spotify API dynamically displays the playlist based on server-side requests when the page is opened. Therefore, anytime a change is made by the curator of the playlist on the Spotify platform, the display on my program will be configured as a result.
- User Tier System: clearly, there are guest users who can’t access the API-driven program, there are normal users that must login to use the program, and then there are admins which have special privileges.
- Session Data: I also take advantage of storing session data in several instances, such as in the admin page and when logging in. I know that this is a 4th extra point but it was needed for the program

CHALLENGES AND OBSTACLES - reflection (please read)

When grading my project, please take the following into consideration:

I had to work through 3 APIs to find one that worked. I spent hours connecting to the twitter API and downloaded special package managers etc. only to find that there is now a paywall to use the API. I then spent more hours connecting to the Facebook API, and once I got it configured and working in Terminal it wouldn’t allow me to access needed metadata. Therefore, using the Spotify API wasn’t my first choice and I have literally spent dozens of hours working on this

project to reorient it using the Spotify API and to make the animated video background etc.. It was very challenging to get everything setup and working. Also, for the CSS, most of my CSS settings were pretty much unique for all pages, I know that a few design components were consistently the same, however it didn't make sense to me to make an external stylesheet for just 5 classes for example. Last thing, when putting my HTML through the error-checking program, there were some errors I got that were due to the hidden PHP that the HTML IDE can't read and also due to some poorly cited data from the Spotify API. They were out of my control to fix but I made sure all fixable errors were fixed.