CPSC471 Project

Winter 2021
Group 15

1 Abstract

The following report outlines the functionalities and usage of Rateify, a web-based service that allows users and artists to connect through music. Ratify allows listeners to listen to music, create playlists, and most importantly rate songs. This makes Rateify different from other music services as it gives listeners a chance to provide feedback on music right on the system. This bridges the gap between artists and listeners, unifying the community in the process. This report will describe different features available to users as well as gives an outline of what each type of user is able to do on Rateify. Additionally, the report will highlight how the database is being maintained and how SQL queries are used to update the database. The interaction with the frontend and backend will also be discussed, giving a high-level description of how the system works.

2 The Problem

Ever since the Internet grew in popularity in the mid to late 1990s, the demand for access to services over the Web has grown. Among these services, access to music is in very high demand. There are many music service platforms that exist, including the likes of Spotify, Apple Music, Google Play Music, Amazon Music, and SoundCloud. While each of these services provide different user experiences from one another, the most important feature among these platforms is the ability for users to listen to songs from their favourite artists. A feature that some of these platforms lack, in particular, Spotify, is the ability for users to create ratings for songs. That is where Rateify comes in. Rateify simulates the basic features of a music service platform, but enables users to rate songs created by artists.

3 The System

Rateify is a web application that simulates the basic features included in the user experience of an online music service platform. It facilitates the experience of four different user types, each with different permissions on how they can interact with the database connected to the platform (see “4 Project Design”). The language of choice for the frontend and API are in php, and the backend is designed using a MySQL database. In order to avoid copyright issues, the platform is limited from containing audio files that correspond to songs; however, the action of listening to a song can be tracked by its number of plays. Upon loading the landing page, the user is prompted to either sign up or create an account. Once logged in, the user can navigate

the site to perform various actions that interact with the database, each of which are dependent on the account type of the user.

4 Project Design

The system has four different types of users: artists, listeners, producers and administrators.

All Users are able to:

- **Login**

  The user is prompted to input their username and password by the system. After this information is inputted in, the information is sent to the backend and the corresponding function for logging in is called. This information is checked in the database if the info is correct. If the information is incorrect then the system displays a notification stating that the credentials are incorrect. If a matching row in the “account” table is found, the user logs into the system and the options available to the users account type are displayed. The user’s username is also saved, as this will be used for a lot of the other options available.

- **Sign Up**

  When this option is selected, the user is prompted to input their desired username, password and account_type. This information is then sent to the backend and the signup function is called. This information is first checked to see if a duplicate username (the PK of the “account” table) exists. If there is a duplicate entry, the user is notified that they cannot create their account with that username. If the check is passed, the new entry is added to the “account” table and the user is notified that the sign up was successful.

- **Logout**

  The user is returned to the main page of Rateify

Artists are able to:

- ***Create song***

  When this option is selected, the user is prompted to input the song name, it’s duration and the date that it was created. After this information is gathered, the information is sent to the backend and a function is called to create the song. This new entry is first crossed check for any duplicates in the database and if a duplicate is found, the user will be notified that they cannot create the song. If the check is passed, then the new entry is added to the “song” table and “artist_song” table to the database.

- ***View all created songs***

  This option allows the user to see all their songs that are currently in the database. The information is pulled by traversing the “artist_song” table and

displaying information of the songs that were found. The “artist_song” table contains the primary key of each song, and that is used to gather all the information for each song. Afterwards the artist is allowed to click on the song’s name to see more information about it as well as the rating given to it by the community.

- ***Create album***

  Similar to “create song”, when this option is selected, the user is prompted to input the album’s name and the date it was created. After this information is gathered, the information is sent to the backend and a corresponding function for creating an album is called. This new entry is first crossed check for any duplicates in the database and if a duplicate is found, the user will be notified that they cannot create the song. If the check is passed, then the new entry is added to the “album” table and the “artist_album” table in the database.

- ***View owned albums***

  This option allows the user to see all the albums that they have currently created (and are in the database). The information is pulled by traversing the “artist_album” table and displaying information of the albums that were found. The “artist_album” table contains the primary key of each album, and that is used to gather all the information for each album. Afterwards the artist is allowed to click on the album’s name to see more information about it as well as all the songs that are currently in that album. Once they are on this page, they can also add songs to their album.

  - ***Adding owned songs to owned albums***

    The user can add any of their created songs to the selected album. When this option is selected, the user is taken to a screen that shows all their songs. Clicking on the name of any song, will add it to the current album and display it on the “AlbumPage”. This also updates the database and changes the selected song’s album name. Also the “album_songs” table is updated accordingly.

Listeners are able to:

- **Search for song**

  The user is able to search for songs in the database by inputting the name of the song that they would like to view. Since the song’s name is not it’s primary key, the page will display all song’s that have the corresponding name. All the resulting song’s information is displayed on this page and the user can select a particular song to view more information about it.

  - **View song**

Once the user selects a song, the page displays the exact information that was shown to the artist (see above Artist - View Song) with the added feature of being able to create their own ratings.

- **Create Rating**

  The user can add their own rating/comment to the selected song. When this option is selected, the user is prompted to input their comment and “star_rating” (a rating out of 5) on the selected song. The user is then taken back to the main “listener” page. This rating can be viewed by researching the same song.

- **Create Rating (without selecting a specific song)**

  The above “create Rating” option is also available to the listener without selecting a specific song. This option requires to search for the song that they want to rate, and all the songs that match the user's input will be displayed. The user can select one of those songs to rate and the user will be taken to the commenting/rating page.

- **View created playlists**

  The listener is able to view all of their created Playlist on this page. This information is pulled from the “playlist” table in the database and displays the playlists owned by the current user. The user can also click on each playlist’s name to see all the songs that are currently in the playlist.

  - **View Songs in Selected Playlist**

    Once the user selects a particular playlist, they are shown the information of each song in that playlist. The song_id is first found by traversing through the “playlist_song” table, finding all songs that are in the selected playlist. Using the song_id the rest of the information is collected from the “song” table and displayed on the screen. The user can choose to play a song by clicking on the play button on the screen. This functionality will increase the no_of_plays that the selected song has. The listener can also remove songs from the playlist, changing the affected information.

  - **Create a New Playlist**

    The listener can choose to create a new playlist as well. This will prompt the user to input the name of the new playlist in a text field. The inputted name will be cross checked with the database to ensure no duplicates are present. Afterward, the user is taken back to the list of all their playlists and the newly added playlist will also be present in this list.

Producers are able to:

- **Produce Songs**

- The producer can choose to produce a new song. When this option is chosen the user is prompted to input the song's name, duration and date_created. This new entry is first crossed check for any duplicates in the database and if a duplicate is found, the user will be notified that they cannot produce this song. If the check is passed, then the new entry is added to the "song" table and "producer_song" table to the database.

Administrators are able to:

- **Search for song**

  When this option is selected, the user is prompted to input the song's name that they would like to search. The inputted information is then checked for in the "song" table and all corresponding song's are displayed. If no songs are found, The displayed table will be empty, and a text saying "No Results" will be displayed. The song_name, artist and song_id are displayed for each song. This option is primarily used to find the song's information if the admin decided to delete a song and need's the song_id (the PK of song)

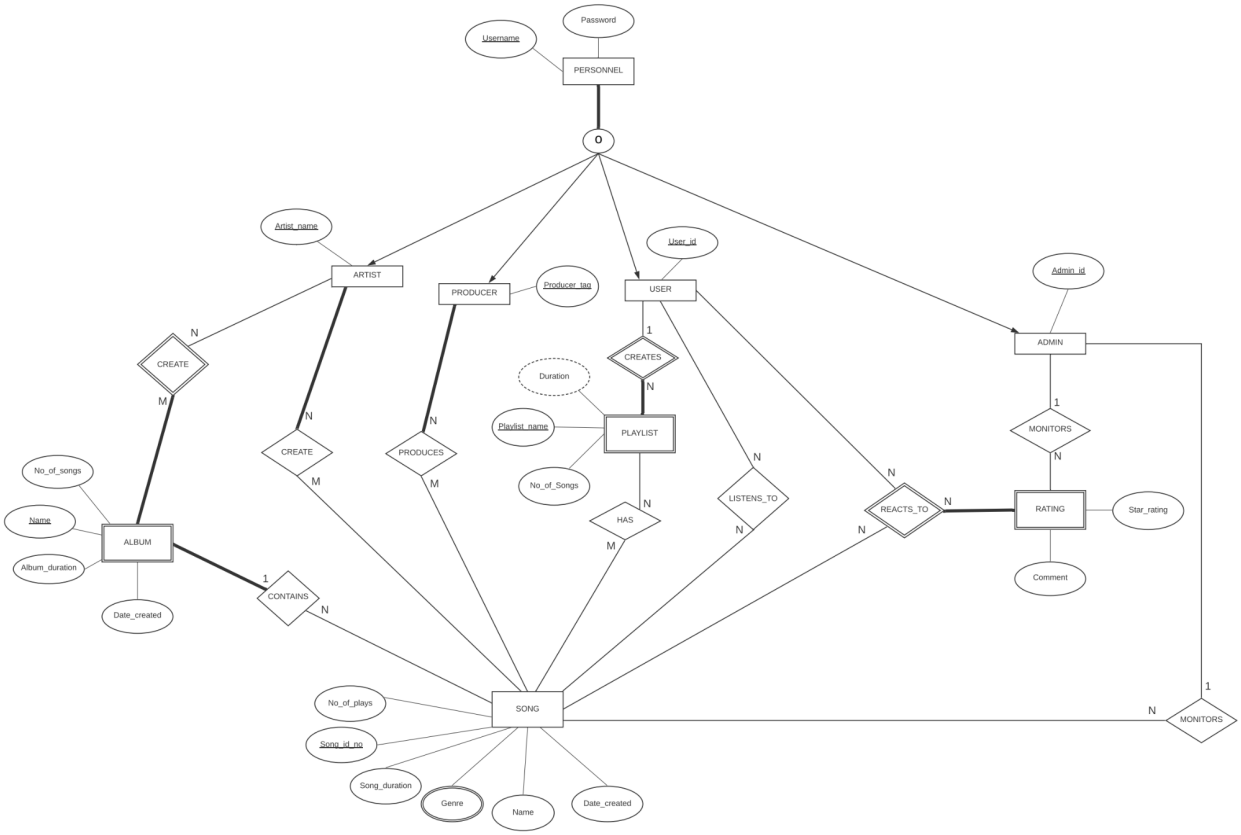
- **Delete Song**

  The admin is able to delete songs that are deemed inappropriate for the system. To do this, the user is prompted to input the song_id of the song that they want to delete. This information is first checked to see if this is an existing id and then the deletion process is started. All references to this song_id are first deleted, these are in; "playlist_song", "album_song", "artist_song", and "rating". Then the song_id in the "song" table is deleted from the database. After this is done, the user is notified that the deletion was completed.

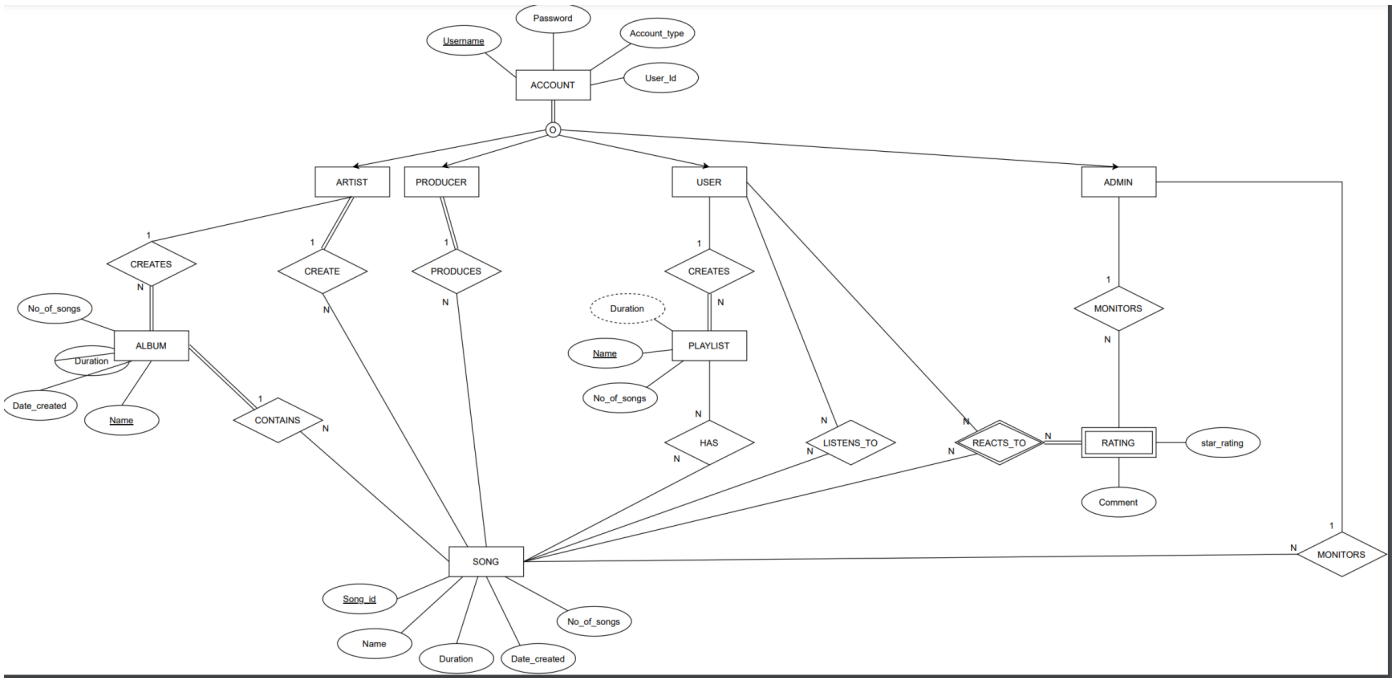
- **Delete Rating**
  - The admin is also able to delete ratings that are deemed inappropriate for the system. To do this, the user is prompted to input the username of the user who made the rating as well as the song_id of the song the rating is under. This information is checked to see if this information is present in the "rating" table and then the deletion process is started. The information that matched the user's input is deleted in the "rating" table and the user is notified that the deletion was completed.

ER DIAGRAM: (If the ER Diagram are difficult to read, the ER diagrams are also available on our github)

Original-



Updated-



The following changes have been made to the ER Model:

- Instead of having an N to N relation between Artist and Song, it is changed to a 1 to N relation. This means that only 1 artist creates a song (a song can't have more than 1 artist)
- Instead of having an N to N relation between Producer and Song, it is changed to a 1 to N relation. This means that only 1 producer creates a song (a song can't have more than 1 producer)
- The multivalued genre variable in song was removed due to the time constraint of completing the project in time. There was too much functionality to complete and as such the group felt the genre was a minor component that can be left out, and added if the group had enough time to finish everything else.
- Artist, producer, Users and Admin do not have anything specific (i.e Artist_name only for artist) to them. Instead, an account_type attribute is added to the personnel entity to distinguish between the different types of users.
- Personnel was changed to Account
- Album and Playlist are no longer weak entities

Changes made since the presentation:

- The sql queries were changed to prevent any SQL injections
- Decremented the album duration when a song is removed from the album

5 Implementation

- **Relational model of the database:**

The relational model for this database is too big and would be broken into different pictures to include it here, accessing to this relational model can be found here https://github.com/Christopher-Schultze/Rateify/blob/main/Diagrams/CPSC471%20Relational%20Model.pdf

- The DBMS that was selected:
    - The DBMS that was selected for this project is a relational database called spotify whose functionalities depend on the two main tables “account” and “song”. “account” table keeps track of all users (including Artist, Producer, and Admin) and is used to reference all other tables depending on the purpose of each table. “song” is referenced in most tables since all actions on this application involve the song that is being selected.
- **SQL statements for all functionalities:**
    - **SQL statements used by User/Listener**
        - searchSongByName($conn, $song_name)
            - SQL statement: $sql = "SELECT * FROM song AS songs, artist_song AS artists WHERE songs.id = artists.song_id AND songs.name = $song_name";
        - searchArtistSong($conn, $username, $song_id)
            - SQL statement: $sql = "SELECT * FROM artist_song WHERE song_id = $song_id AND artist_username = $username";
        - searchSongByAritst($conn, $username)
            - SQL statement: $sql = "SELECT * FROM artist_song WHERE artist_username = $username";
        - addRating($conn, $username, $songId, $comment, $star_rating)
            - SQL statement: $sql = "INSERT INTO rating (user_username, song_id, star_rating, comment) VALUES('$username', '$songId', '$star_rating', '$comment')";
        - createPlaylist($conn, $name, $user_username)
            - SQL statement: $sql = "INSERT INTO playlist (user_username, name, no_of_songs, duration) VALUES('$user_username', '$name', 0, 0)";
        - searchPlaylistsByUser($conn, $username)
            - SQL statement: $sql = "SELECT * FROM playlist WHERE user_username = $username";
        - searchSongsInPlaylist($conn, $playlist_name)
            - SQL statement: $sql = "SELECT song_id FROM playlist_song WHERE playlist_name = '$playlist_name'";
        - searchSong($conn, $songId)

            - SQL statement: $sql = "SELECT * FROM spotify.song WHERE id = $songId";
        - searchArtistBySong($conn, $songID)
            - SQL statement: $sql = "SELECT * FROM artist_song WHERE song_id = $songID";
        - increaseNoOfPlays($conn, $songId)
            - SQL statement: $sql = "UPDATE song SET no_of_plays = no_of_plays + 1 WHERE id = '$songId' ";
        - removeSongFromPlaylist($conn, $playlist_name, $songID, $username)
            - SQL statement: $sql = "DELETE FROM playlist_song WHERE song_id = $songID AND playlist_name = '$playlist_name'";
            - SQL statement: $sql = "UPDATE playlist SET no_of_songs = no_of_songs - 1 WHERE name = '$playlist_name'";
            - SQL statement: $sql = "UPDATE playlist SET duration = duration - $dur WHERE name = '$playlist_name'";
    - **SQL statements used by Artist**
        - searchSong($conn, $songId)
            - SQL statement: $sql = "SELECT * FROM spotify.song WHERE id = $songId";
        - searchRatings($conn, $songID)
            - SQL statement: $sql = "SELECT * FROM rating WHERE song_id = $songID";
        - searchSongsInAlbum($conn, $album_name)
            - SQL statement: $sql = "SELECT * FROM album_song WHERE album_name = $album_name";
        - searchAlbum($conn, $album_name)
            - SQL statement: $sql = "SELECT * FROM album WHERE name = $album_name";
        - searchSongByName($conn, $song_name)
            - SQL statement: $sql = "SELECT * FROM song AS songs, artist_song AS artists WHERE songs.id = artists.song_id AND songs.name = $song_name";
        - searchArtistSong($conn, $username, $song_id)
            - SQL statement: $sql = "SELECT * FROM artist_song WHERE song_id = $song_id AND artist_username = $username";
        - getMaxSongID($conn)
            - SQL statement $sql = "SELECT MAX(id) AS max_id FROM song";
        - createSong($conn, $id, $album_name, $no_of_plays, $duration, $name, $date_created, $username, $type)
            - SQL statement: $sql = "INSERT INTO song (id, album_name, no_of_plays, duration, name, date_created) VALUES ('$id', NULL, '$no_of_plays', '$duration', '$name', '$date_created')";
            - SQL statement: $sql = "INSERT INTO song (id, album_name, no_of_plays, duration, name, date_created) VALUES ('$id',

'$album_name', '$no_of_plays', '$duration', '$name', '$date_created')";
- SQL statement: $sql2 = "INSERT INTO album_song(album_name, song_id) VALUES('$album_name', '$id')";

- addSongToAlbum($conn, $a_name, $songId)
  - SQL statement: $sql = "INSERT INTO album_song (album_name, song_id) VALUES('$a_name', '$songId')";
  - SQL statement: $sql = "UPDATE song SET album_name = '$a_name' WHERE id = '$songId'";
- makeChangestoAlbum($conn, $a_name, $songId)
  - SQL statement: $sql = "UPDATE album SET duration = duration + $dur WHERE name = '$a_name' ";
  - SQL statement: $sql = "UPDATE album SET no_of_songs = no_of_songs + 1 WHERE name = '$a_name' ";
- addToArtistSong($conn, $username, $id)
  - SQL statement: $sql = "INSERT INTO artist_song (artist_username, song_id) VALUES('$username', '$id')";
- searchAlbumArtist($conn, $username, $album_name)
  - SQL statement: $sql = "SELECT * FROM artist_album WHERE artist_username = '$username' AND album_name = '$album_name'";
- createAlbum($conn, $album_name, $date_created, $username)
  - SQL statement: $sql = "INSERT INTO album (name, no_of_songs, duration, date_created) VALUES('$album_name', 0, 0, '$date_created')";
  - SQL statement: $sql = "INSERT INTO artist_album(artist_username, album_name) VALUES ('$username', '$album_name')";
  - SQL statement: $sql = "INSERT INTO artist_album(artist_username, album_name) VALUES(?,?)";

- **SQL statement used by Producer**
  - getMaxSongID($conn)
    - SQL statement $sql = "SELECT MAX(id) AS max_id FROM song";
  - createSong($conn, $id, $album_name, $no_of_plays, $duration, $name, $date_created, $username, $type)
    - SQL statement: $sql = "INSERT INTO song (id, album_name, no_of_plays, duration, name, date_created) VALUES ('$id', NULL, '$no_of_plays', '$duration', '$name', '$date_created')";
    - SQL statement: $sql = "INSERT INTO song (id, album_name, no_of_plays, duration, name, date_created) VALUES ('$id', '$album_name', '$no_of_plays', '$duration', '$name', '$date_created')";

        - SQL statement: $sql2 = "INSERT INTO album_song(album_name, song_id) VALUES('$album_name', '$id')";
    - addToProduceSong($conn, $username, $id)
        - SQL statement: $sql = "INSERT INTO producer_song (producer_username, song_id) // VALUES('$username', '$id')";
- **SQL statement used by Admin**
    - searchSongByName($conn, $song_name)
        - SQL statement: $sql = "SELECT * FROM song AS songs, artist_song AS artists WHERE songs.id = artists.song_id AND songs.name = $song_name";
    - searchSong($conn, $songId)
        - SQL statement: $sql = "SELECT * FROM spotify.song WHERE id = $songId";
    - deleteSong($conn, $songID)
        - SQL statement: $sql = "DELETE FROM album_song WHERE song_id = $songID";
        - SQL statement: $sql = "DELETE FROM playlist_song WHERE song_id = $songID";
        - SQL statement: $sql = "DELETE FROM producer_song WHERE song_id = $songID";
        - SQL statement: $sql = "DELETE FROM rating WHERE song_id = $songID";
        - SQL statement:$sql = "DELETE FROM artist_song WHERE song_id = $songID";
        - SQL statement: $sql = "DELETE FROM song WHERE id = $songID";
    - searchSpecificRatings($conn, $songID, $username)
        - SQL statement: $sql = "SELECT * FROM rating WHERE song_id = $songID AND user_username = '$username'";
    - deleteRating($conn, $username, $songId)
        - SQL statement: $sql = "DELETE FROM rating WHERE song_id = $songId AND user_username = '$username'";

6 API Documentation

See generated API documentation through Postman here:
https://documenter.getpostman.com/view/15040025/TzJsfdYo

7 User Guide

- **Setting up the application:**
    - Download xampp to host Apache server and MySQL server at https://www.apachefriends.org/download.html
    - Set up xampp following the instruction of the executable file downloaded. Then save xampp in a local directory in your computer. (Recommended default directory at C:/xampp)
    - Download git bash. (Suggested link https://git-scm.com/downloads)
    - Choose any directory on your computer, right click and click Git Bash here and type git clone https://github.com/Christopher-Schultze/Rateify.git
    - Once cloning is done, copy the Rateify folder and copy it in C:/xampp/htdocs.
    - Now, run the server by searching up xampp-control in your windows search bar. (If not found then simply go to C:/xampp and scroll down to the bottom of this directory there should be an application file called xampp-control). Once the xampp-control window pops up, click Start for Apache and MySQL.
    - Start the database by simply just type localhost:80/phpmyadmin in any web browser. Once phpmyadmin is loaded, click Import on the top bar, then click choose file and choose the spotify.sql file in the Rateify folder that you cloned from git bash.
    - Run the project by opening another tab and type localhost:80/Rateify/frontend/index.php
- **How to use the application**
    - **User/Listener**
        - Sign up by clicking “Get Started” or “Sign up”, then enter your preferred Username and Password, your Username is what everyone will identify you as on this application, then choose the option of “Listener” and click sign up. If a username already existed, the application will ask you to choose a different one.
        - Log in with your username and password, then a user page is presented with all the functionalities that a user has as discussed in section 4.
    - **Producer**
        - Repeat the same process as User/Listener, but when signing up choose the option “Producer” instead.
    - **Artist**
        - Repeat the same process as User/Listener, but when signing up choose the option “Artist” instead.
    - **Admin**
        - Repeat the same process as User/Listener, but when signing up choose the option “Admin” instead.

8 References

Sources used to create the project include:

- Primary means of communications:
    - Discord
    - Github
- Setting up a php connection using XAMPP:
    - https://www.edureka.co/blog/how-to-run-a-php-program-in-xampp/
- For understanding prevention of SQL injections in php:
    - https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php