CPSC 471 - Spoofy

Alex Stevenson - 30073617
Eric Gantz - 30031518
Ryan Fowler - 30061742

Link to GitHub Repository: https://github.com/alexs2112/CPSC471_Spoofy
This repository contains:

- Source code for our project
- Setup instructions to link the project to an Apache web server and MySQL database
- An initializer written in python to set up the MySQL database with default data
- Copies of our diagrams

Abstract:

Our project is Spoofy, a website which allows users to view songs or advertisements, as well as organize those songs into playlists. It is inspired by other well-known music services like Spotify or SoundCloud. The back end of the website is a database management system which stores information about user accounts, songs, albums, artists, and advertisements. Users are either on the free track, in which case they only have access to advertisements, or they are on the premium track, in which case they only have access to songs, artists, and albums, as well as the ability to organize those songs into playlists or a queue to customize their music-listening experience. Users are able to upgrade to premium to be able to listen to songs, or downgrade to free to view the advertisements. Administrators of the system are given tools to manage the music in the system, including adding and removing songs, adding, editing, and removing albums, adding, editing, and removing artists, and being able to specify which artist wrote which songs and albums, as well as which songs are contained by an album. A lot of this is done on the behalf of Distributors, who exist in our system as a way for artists to be represented and communicate with the system, but these Distributors only act by communicating to Administrators and thus don't have a separate type of account. Administrators are able to add and remove advertisements for the free users. Administrators are also able to delete user profiles, as well as make any other user an administrator or revoke administrator privileges from an administrator.

Introduction:

The "information superhighway" enabled by the internet has led to the mass sharing of music between people around the world. This can be a complicated problem to solve, as it involves keeping track of the metadata for songs, artists, and albums. Additionally, it requires being able to break something as human and intuitive as music into quantifiable information that can be stored online. This problem is a common one, and there have been many websites and companies that have offered a solution in one way or another. Past services include Napster and iTunes, and some present-day streaming services involve Spotify, Tidal, and Apple Music. Spotify and Apple Music are the most similar to our proposed solution, which allow the streaming of music directly from the site along with the ability to download songs directly. These services allow users to create and listen to playlists out of the music on the site, or listen to playlists other users have created. One main improvement to this type of service could be to have music stored as separate tracks known as stems, which include things such as vocals, guitar, bass, and more. This would be helpful for musicians wanting to learn to play a song on a certain instrument, or for people who want to personalize their listening experience.

System Description:

The system we have designed to solve this problem consists of a webpage linked to a database that stores info about the various bits of metadata of individual songs, albums, and artists. Additionally, we keep track of our user-base, noting subscriptions, account details, and user statistics for each person using our service. We also keep track of the Artists that have submitted music, to be paid a fixed rate for each individual play of any song they have written at the end of each month. This database of music and users is linked to a webpage hosted by an Apache server, programmed primarily in PHP. This service allows users to create accounts, organize playlists and listen to their music, and view additional information about the different songs, albums, and artists present in our database, and also allows administrators to curate the database by adding, modifying, or deleting songs, albums, artist profiles, and user accounts.

Project Design:

Our system has 4 types of users: unregistered users, free users, premium users who pay a monthly subscription, and admins.

Unregistered users can see the list of songs, albums, and artists, and can search through them using the search functionality, they can also view any of these individually. Unregistered users can login if they have an account, or can register an account.

Free users can see the list of advertisements, can search through advertisements, can add advertisements to their queue, and can play advertisements. Free users can also view their profile where they can upgrade to a premium account by subscribing, and they can log out. Free users cannot see or play songs, albums, or artists.

Premium users can see the list of songs, albums, and artists and can search through them, and they can view songs, albums, or artists individually. Premium users can add songs and albums to their queue, they can create playlists and add songs to them, and they can play songs that are in their queue. Premium users can also view their profile, view their created playlists, cancel their premium membership to become a free user, and they can logout.

Admins are ‘normal’ users that have admin privileges, and are not an account type.

Admins can manage users, this means they can view a list of all users, view an individual user’s profile, delete a user’s account, and grant a user account admin privileges.

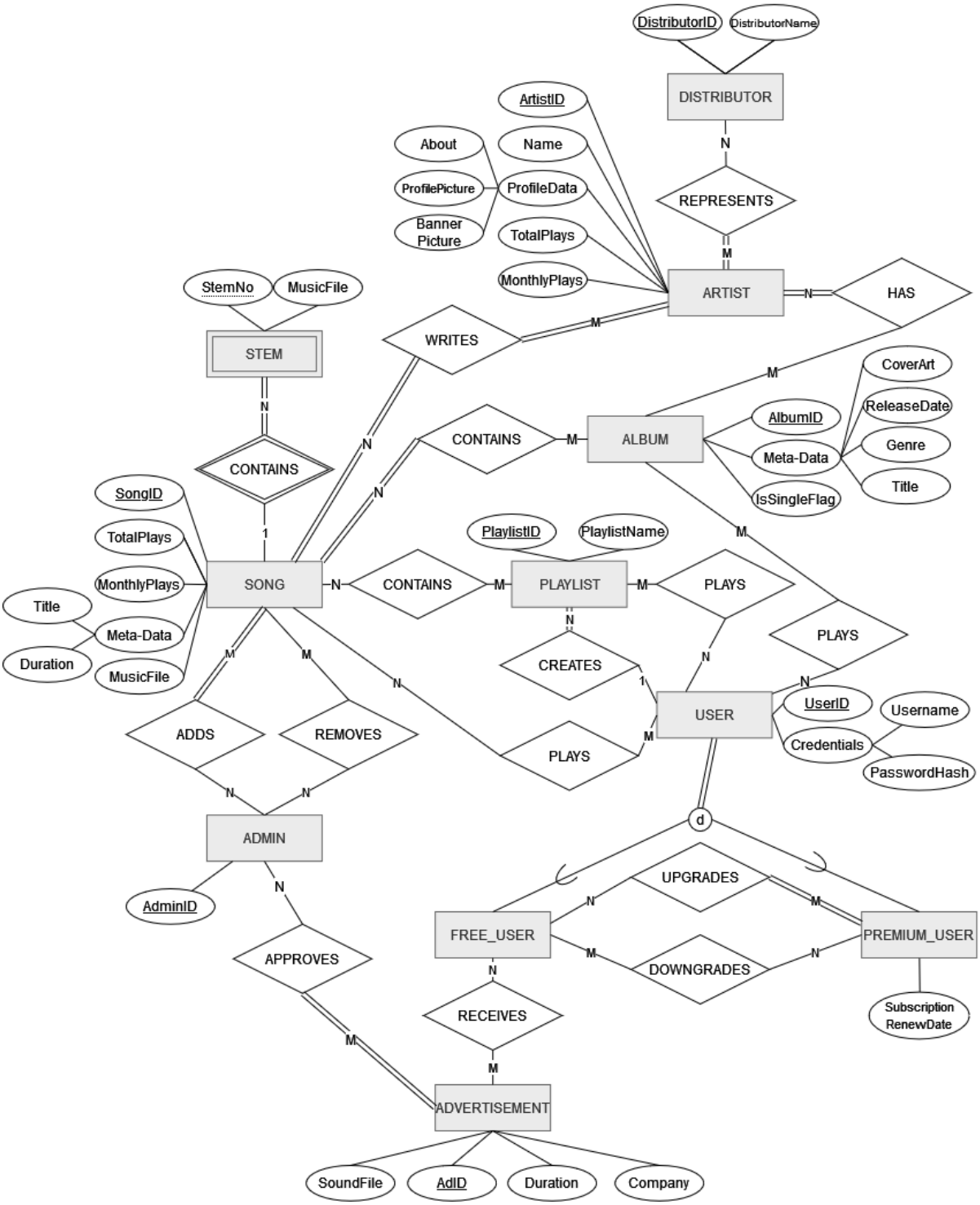
Admins can add or delete songs, reset the monthly plays of a song, add songs to albums, and add artist credits to songs.

Admins can add, edit, or delete albums, remove songs from albums, and can add artist credits to albums.

Admins can add, edit, or delete artists, and can remove albums and songs from artists, removing their credits.

Admins can view the list of advertisements, and can add and delete advertisements.

Extended Entity Relationship Diagram:



Assumptions:

- All Songs should be assigned an Album, all Albums should be assigned an Artist
- Singles will be handled as an Album with a single song

Changes:

- Removal of NumSongs and TotalLength from ALBUM
- TotalPlays and MonthlyPlays no longer derived attributes in ARTIST

Implementation:

Relational Model:

There was a slight challenge with creating these diagrams due to the ability for songs to have multiple albums and artists, and for albums to also have multiple artists. This led to the creation of the ALBUM_CONTAINS, HAS, and WRITES tables.

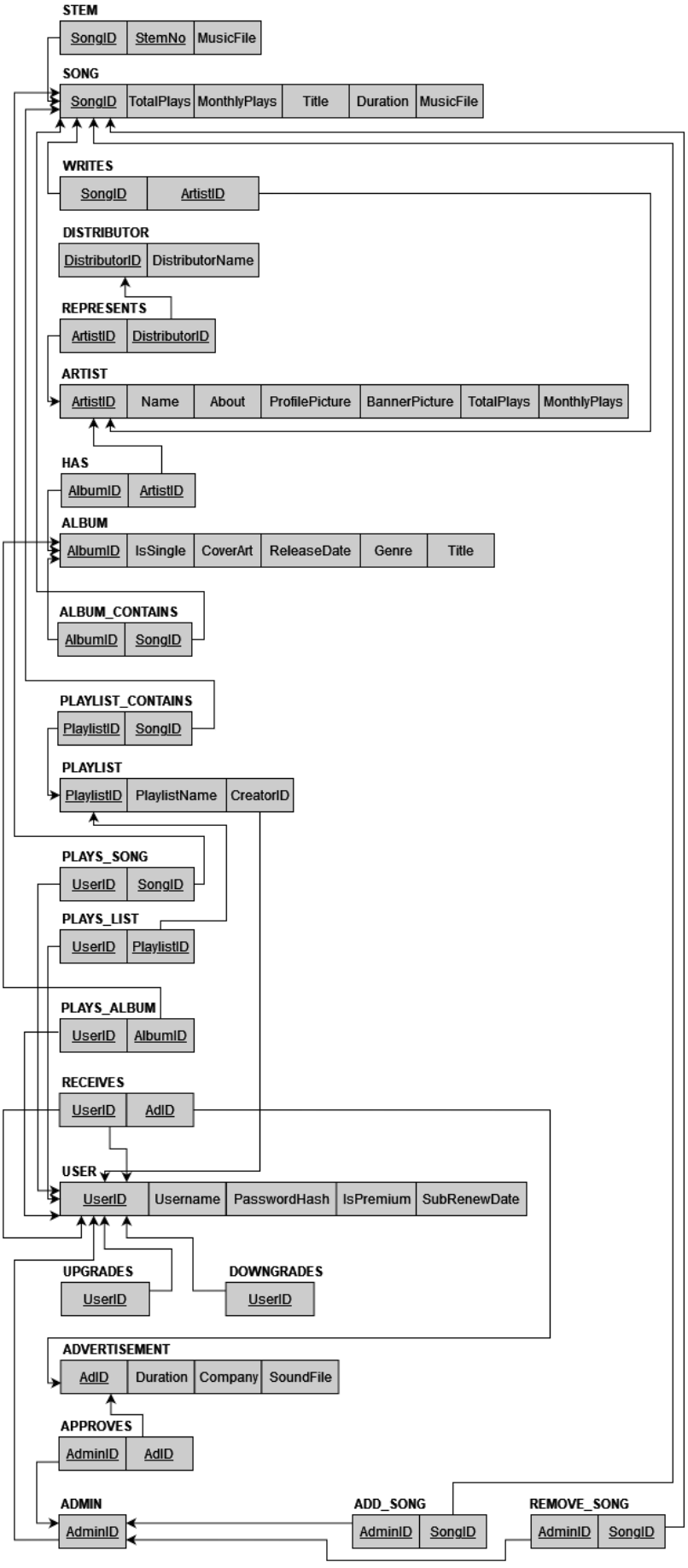
This led to unforeseen consequences for implementing the database in PHP, as these intermediary tables added an extra step for SQL queries that required pairing songs, albums, and artists.

Assumptions:

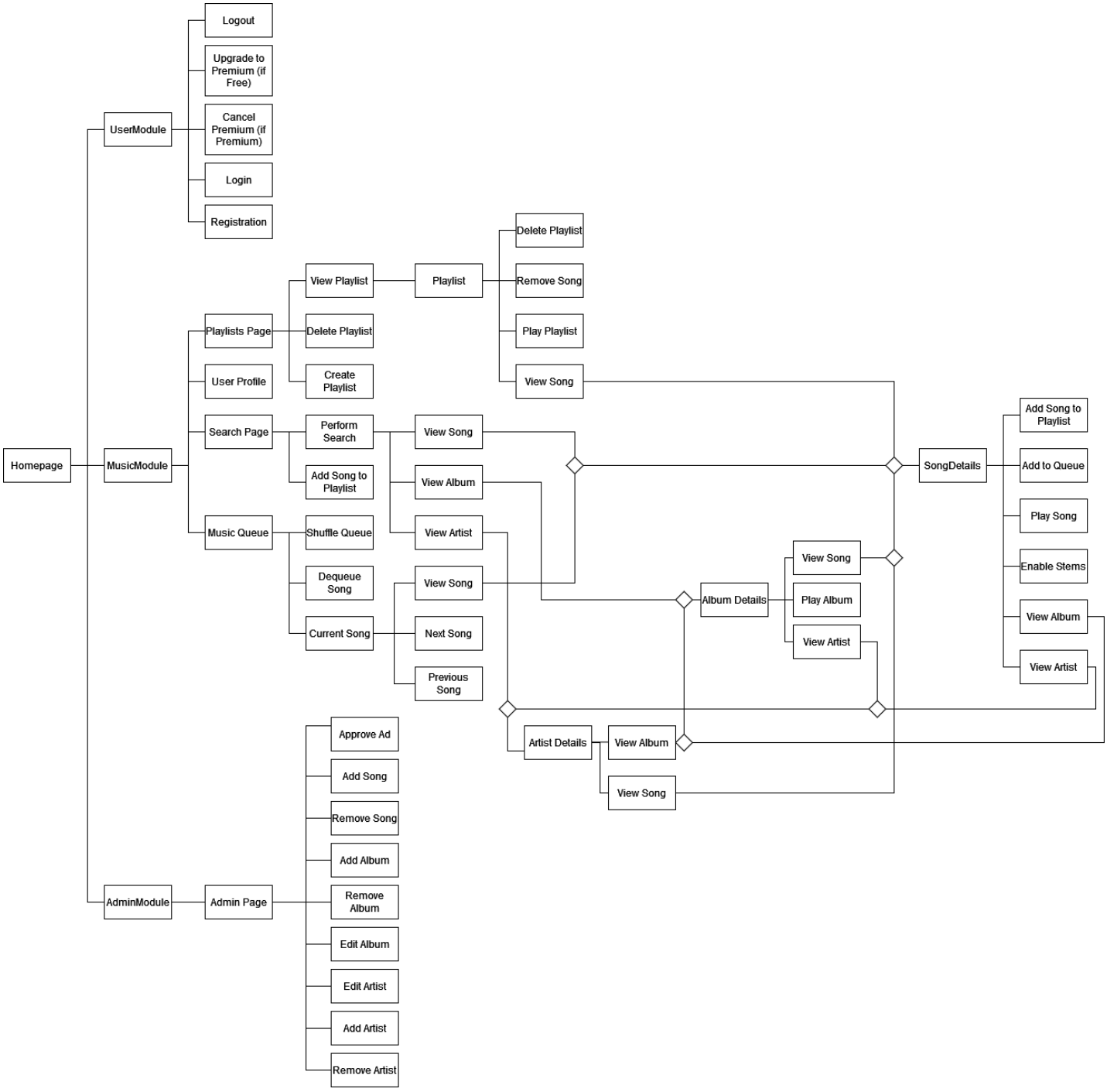
- The RECEIVES relation between USER and ADVERTISEMENT only happens if USER.IsPremium is FALSE.
- The UPGRADES and DOWNGRADES relations for USER will only happen if USER.IsPremium is TRUE or FALSE, respectively.

Changes:

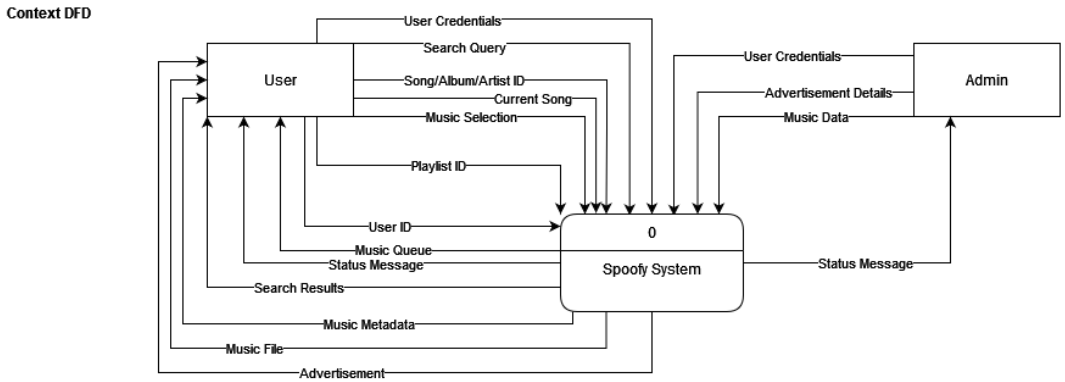
- Same changes as with the EERD:
- Removal of NumSongs and TotalLength from ALBUM
- TotalPlays and MonthlyPlays no longer derived attributes in ARTIST

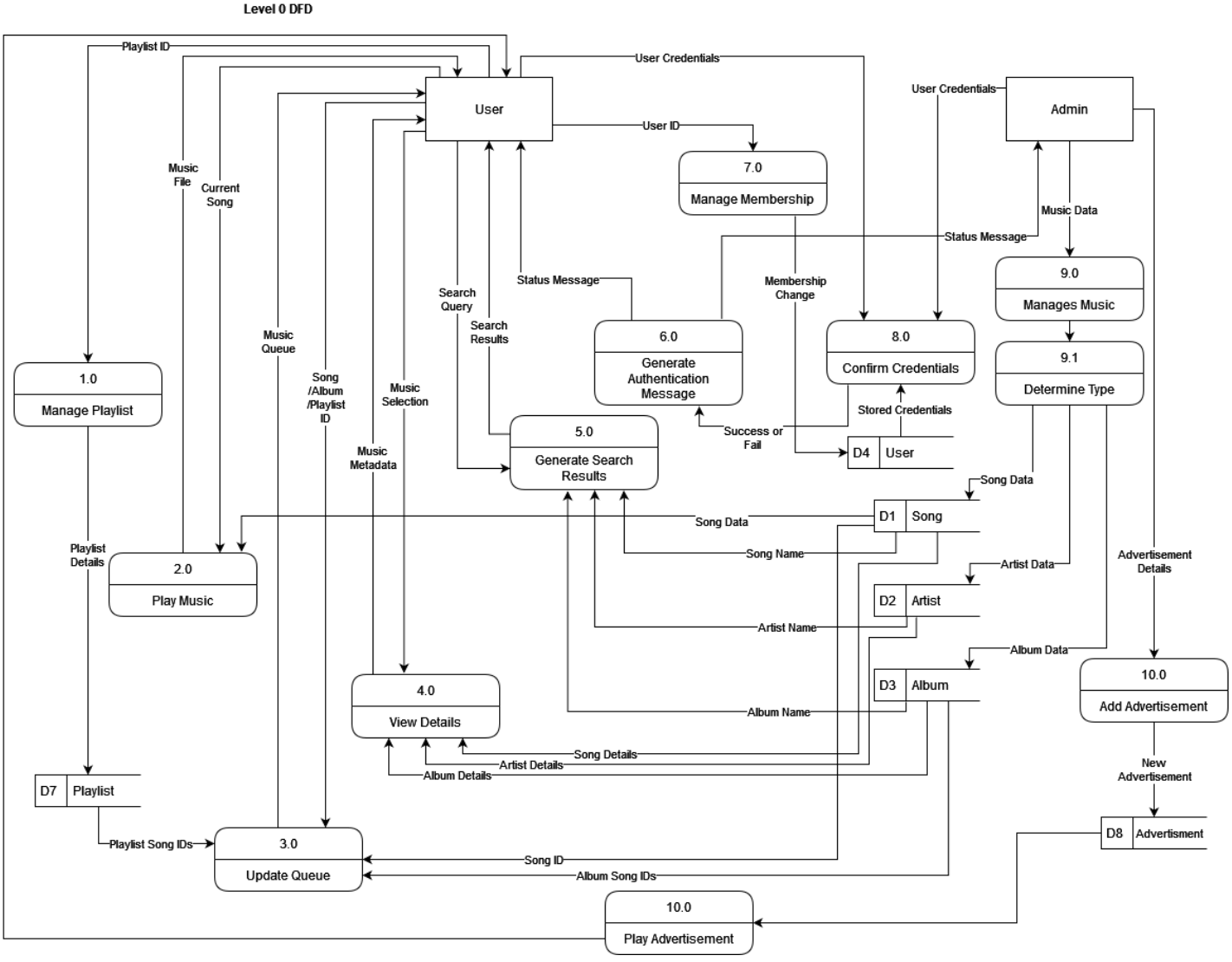


HIPO Diagram:



Data Flow Diagram:





Assumptions for both the HIPO and Data Flow diagrams:

- The UserModule, MusicModule, and AdminModule blocks are non-functional, used to group the processes in the system.
- Playing a song clears the current queue and sets it as the first queue element.
- Playing an album or playlist clears the current queue and sets the queue as that album or playlist.
- The music queue is a list of songs locally available to the user, it should not be in the database and should cease to exist when the webpage is closed.
- Functions that return lists of things are denoted with square brackets. (Example: [SongID] is a list of SongIDs).
- When editing certain database tuples, such as Edit Artist, some inputs can be null. Those null values are simply ignored and do not change the corresponding attribute.

DBMS:

The database management system we selected was MySQL 8.0.

We selected it mostly due to how widespread MySQL is as a DBMS, as well as because a large portion of the course was dedicated to teaching the SQL programming language which MySQL implements to structure its queries. 8.0 is simply the most recent full release of the DMBS.

We were also careful to prevent SQL injection. This was achieved by using a PHP library called MySQLi, which handles creating and handing off the SQL queries to the DBMS. In this library, there are many ways to call an SQL query, but we chose to use a method which involved “preparing” the user-entered variables before they were inserted into the query, and this preparation step parses input strings and escapes out of special SQL characters and commands, preventing typical SQL injection attacks which involve inputting SQL code into text fields.