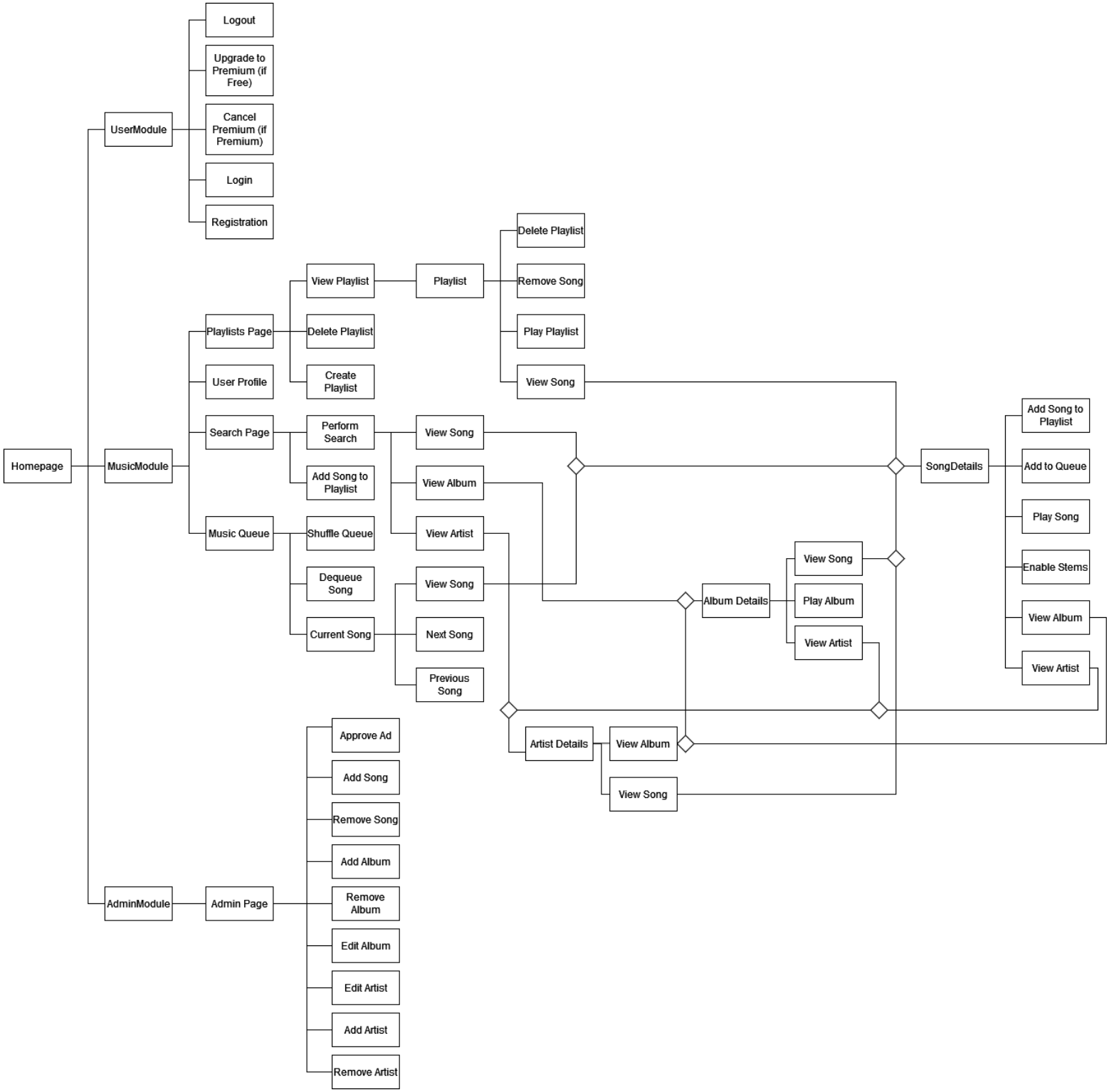
Spoofy: Functional Model

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

HIPO



HIPO Functions

Function: Add Song to Playlist
Inputs: @PlaylistID, @SongID
Outputs: None

Function: Remove Song From Playlist
Inputs: @PlaylistID, @SongID
Outputs: None

Function: Play Song
Inputs: @SongID
Outputs: [SongID]
Note: Clears the current music queue and sets the first element as the selected song

Function: Enable Stems
Inputs: @SongID
Outputs: None

Function: View Album
Inputs: @AlbumID
Outputs: Album Details

Function: View Artist
Inputs: @ArtistID
Outputs: Artist Details

Function: View Song
Inputs: @SongID
Outputs: SongDetails

Function: View Playlist
Inputs: @PlaylistID
Outputs: [SongID]

Function: Play Album
Inputs: @AlbumID
Outputs: [SongID]
Note: Clears the current music queue, sets it as the list of songs in the album

Function: Delete Playlist
Inputs: @PlaylistID, @UserID
Outputs: None

Function: Create Playlist
Inputs: @PlaylistID, @PlaylistName, @UserID
Outputs: None

Function: Play Playlist
Inputs: @PlaylistID, @UserID
Outputs: [SongID]
Note: Clears the current music queue, sets it as the list of songs in the playlist

Function: Next Song
Inputs: None
Outputs: SongID
Note: Returns the song ID of the next song in the queue

Function: Previous Song
Inputs: None
Outputs: SongID
Note: Returns the song ID of the previous song in the queue

Function: Shuffle Queue
Inputs: None
Outputs: None

Function: Dequeue Song
Inputs: @SongID
Outputs: None

Function: Add to Queue
Inputs: @SongID
Outputs: None

Function: View All Playlists
Inputs: None
Outputs: [PlaylistID, PlaylistName]

Function: View User's Playlists
Inputs: @UserID
Outputs: [PlaylistID, PlaylistName]

Function: Perform Search
Inputs: @QueryString
Outputs: [SongID], [AlbumID], [ArtistID]
Note: Returns lists of songs, albums, artists that have names that are similar to the QueryString

Function: Logout
Inputs: @UserID
Outputs: None

Function: Login
Inputs: @Username, @PasswordHash
Outputs: @UserID on success, None on failure

Function: Upgrade to Premium
Inputs: @UserID
Outputs: None

Function: Cancel Premium
Inputs: @UserID
Outputs: None

Function: Register New User
Inputs: @Username, @PasswordHash
Outputs: None

Note: The remaining functions are applicable to Admin users only.

Function: Approve Ad
Inputs: @Duration, @Company, @SoundFile
Outputs: None

Function: Add Song
Inputs: @Title, @Duration, @MusicFile
Outputs: None

Function: Remove Song
Inputs: @SongID
Outputs: None

Function: Add Album
Inputs: @IsSingle, @CoverArt, @ReleaseDate, @Genre, @TotalDuration
Outputs: None

Function: Remove Album
Inputs: @AlbumID
Outputs: None

Function: Edit Album
Inputs: @AlbumID, @IsSingle, @CoverArt, @ReleaseDate, @Genre, @TotalDuration
Outputs: None

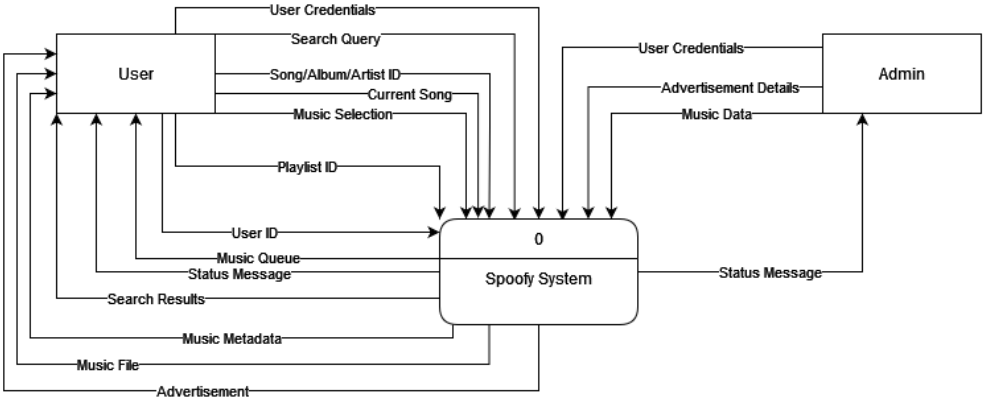
Function: Edit Artist
Inputs: @ArtistID, @Name, @About, @ProfilePicture, @BannerPicture
Outputs: None

Function: Add Artist
Inputs: @Name, @About, @ProfilePicture, @BannerPicture
Outputs: None

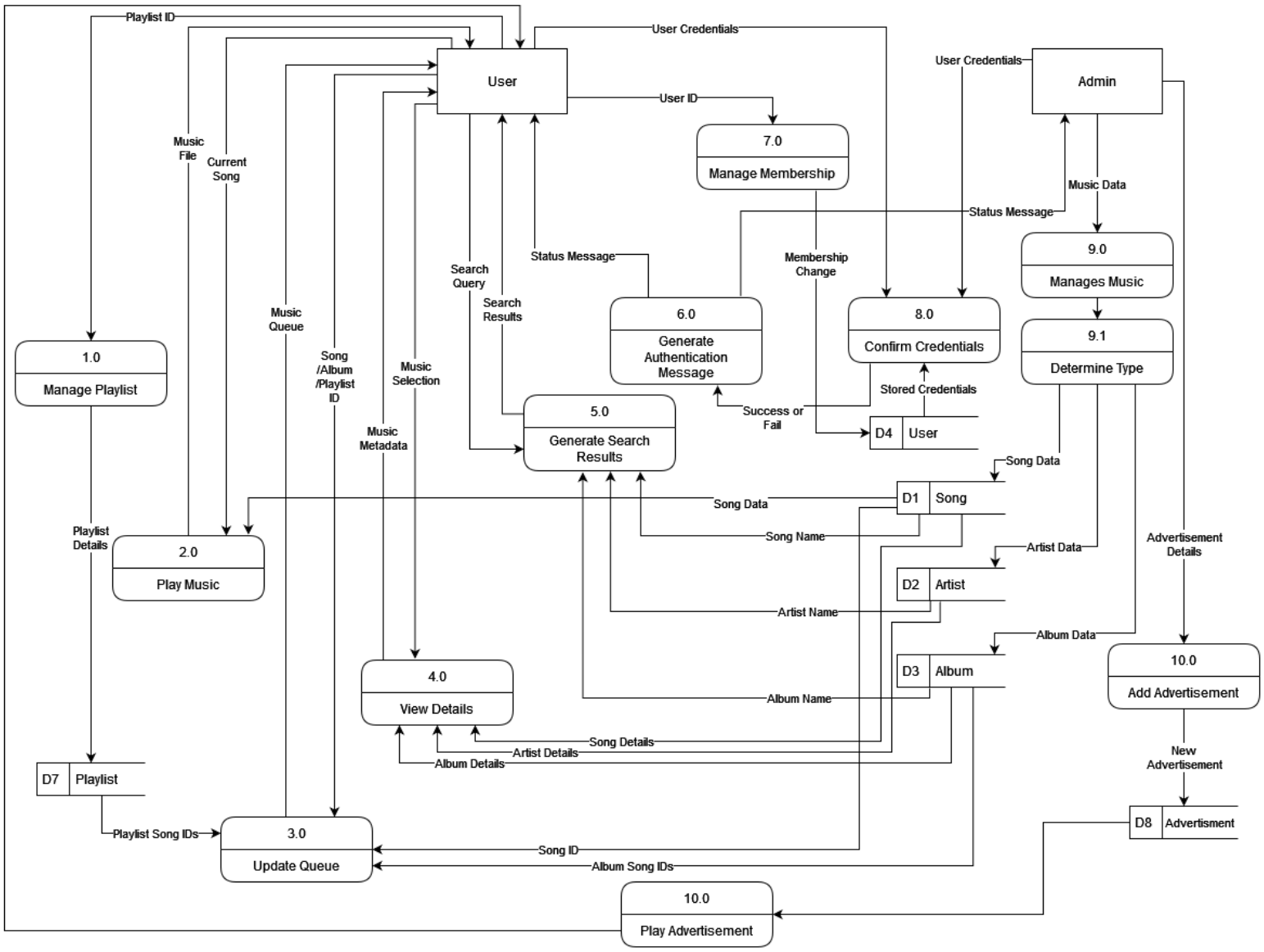
Function: Remove Artist
Inputs: @ArtistID
Outputs: None

DFD

Context DFD



Level 0 DFD



SQL Statements

Create Tables

```
CREATE  TABLE STEM
    (SongID    INT    IS NOT NULL,
    StemNo    INT    IS NOT NULL,
    MusicFile    VARCHAR(255),
    PRIMARY KEY (SongID, StemNo),
    FOREIGN KEY (SongID) REFERENCES SONG(SongID) ON DELETE CASCADE ON UPDATE CASCADE);

CREATE  TABLE SONG
    (SongID    INT    IS NOT NULL    AUTO_INCREMENT,
    TotalPlays    INT,
    MonthlyPlays    INT,
    Title    VARCHAR(32),
    Duration    VARCHAR(32),
    MusicFile    VARCHAR(255),
    PRIMARY KEY (SongID));

CREATE  TABLE WRITES
    (SongID    INT    IS NOT NULL,
    ArtistID    INT    IS NOT NULL,
    PRIMARY KEY (SongID, ArtistID),
    FOREIGN KEY (SongID) REFERENCES SONG(SongID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (ArtistID) REFERENCES ARTIST(ArtistID) ON DELETE CASCADE ON UPDATE CASCADE);

CREATE  TABLE DISTRIBUTOR
    (DistributorID    INT    IS NOT NULL    AUTO_INCREMENT,
    DistributorName    VARCHAR(32),
    PRIMARY KEY (DistributorID));

CREATE  TABLE REPRESENTS
    (ArtistID    INT    IS NOT NULL,
    DistributorID    INT    IS NOT NULL,
    PRIMARY KEY (ArtistID, DistributorID),
    FOREIGN KEY (ArtistID) REFERENCES ARTIST(ArtistID) ON DELETE CASCADE ON
    UPDATE CASCADE,
    FOREIGN KEY (DistributorID) REFERENCES DISTRIBUTOR(DistributorID)ON
    DELETE CASCADE ON UPDATE CASCADE);

CREATE  TABLE ARTIST
    (ArtistID    INT    IS NOT NULL    AUTO_INCREMENT,
    Name    VARCHAR(32),
    About    VARCHAR(1500),
    ProfilePicture    VARCHAR(255),
    BannerPicture    VARCHAR(255),
    TotalPlays    INT,
    MonthlyPlays    INT,
    PRIMARY KEY (ArtistID));

CREATE  TABLE HAS
    (AlbumID    INT    IS NOT NULL,
    ArtistID    INT    IS NOT NULL,
    PRIMARY KEY (AlbumID, ArtistID),
    FOREIGN KEY (AlbumID) REFERENCES ALBUM(AlbumID) ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (ArtistID) REFERENCES ARTIST(ArtistID) ON DELETE CASCADE ON
    UPDATE CASCADE);
```

```
CREATE  TABLE ALBUM
    (AlbumID    INT    IS NOT NULL    AUTO_INCREMENT,
    IsSingle    BOOLEAN    IS NOT NULL,
    CoverArt    VARCHAR(255),
    ReleaseDate    DATE,
    Genre    VARCHAR(255),
    NumSongs    INT,
    TotalDuration    VARCHAR(32),
    Title    VARCHAR(32),
    PRIMARY KEY (AlbumID));

CREATE  TABLE ALBUM_CONTAINS
    (AlbumID    INT    IS NOT NULL,
    SongID    INT    IS NOT NULL,
    PRIMARY KEY (AlbumID, SongID),
    FOREIGN KEY (AlbumID) REFERENCES ALBUM(AlbumID) ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (SongID) REFERENCES SONG(SongID) ON DELETE CASCADE ON
    UPDATE CASCADE);

CREATE  TABLE PLAYLIST_CONTAINS
    (PlaylistID    INT    IS NOT NULL,
    SongID    INT    IS NOT NULL,
    PRIMARY KEY (PlaylistID, SongID),
    FOREIGN KEY (PlaylistID) REFERENCES PLAYLIST(PlaylistID) ON DELETE
    CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (SongID) REFERENCES SONG(SongID)ON DELETE CASCADE ON
    UPDATE CASCADE);

CREATE  TABLE PLAYLIST
    (PlaylistID    INT    IS NOT NULL    AUTO_INCREMENT,
    PlaylistName   VARCHAR(32),
    CreatorID    INT,
    PRIMARY KEY (PlaylistID),
    FOREIGN KEY (CreatorID) REFERENCES USER(UserID) ON DELETE SET NULL
    ON UPDATE CASCADE);

CREATE  TABLE USER
    (UserID    INT    IS NOT NULL    AUTO_INCREMENT,
    Username    VARCHAR(32)    IS NOT NULL,
    PasswordHash    CHAR(32)    IS NOT NULL,
    IsPremium    BOOLEAN    IS NOT NULL,
    SubRenewDate    DATE,
    PRIMARY KEY (UserID),
    UNIQUE (Username));

CREATE  TABLE ADVERTISEMENT
    (AdID    INT    IS NOT NULL    AUTO_INCREMENT,
    Duration    VARCHAR(32),
    Company    VARCHAR(32),
    SoundFile    VARCHAR(255),
    PRIMARY KEY (AdID));

CREATE TABLE ADMIN
    (AdminID    INT    IS NOT NULL,
    PRIMARY KEY (AdminID),
    FOREIGN KEY (AdminID) REFERENCES USER(UserID) ON DELETE CASCADE ON
    UPDATE CASCADE);
```

Add Song

Input: @Title, @Duration, @MusicFilePath, @AlbumID, @ArtistID

```
    INSERT INTO SONG (TotalPlays, MonthlyPlays, Title, Duration, MusicFile)
    VALUES (0, 0, @Title, @Duration, @MusicFilePath);

    INSERT INTO ALBUM_CONTAINS
    VALUES (@AlbumID, LAST_INSERT_ID());

    INSERT INTO WRITES
    VALUES (LAST_INSERT_ID(), @ArtistID);
```

Remove Song

Input: @SongID

```
    DELETE FROM SONG
    WHERE SongID=@SongID;
```

Add Stem

Input: @SongID, @StemNo, @MusicFilePath

```
    INSERT INTO STEM
    VALUES (@SongID, @StemNo, @MusicFilePath);
```

Song Details

Input: @SongID

```
    SELECT TotalPlays, MonthlyPlays, Title, Duration
    FROM SONG
    WHERE SongID=@SongID;
```

Clear Monthly Plays

Input: @SongID

```
    UPDATE SONG
    SET MonthlyPlays=0
    WHERE SongID=@SongID;
```

Play Song

Input: @SongID

```
    UPDATE SONG
    SET MonthlyPlays=MonthlyPlays+1, TotalPlays=TotalPlays+1
    WHERE SongID=@SongID;

    SELECT MusicFile
    FROM SONG
    WHERE SongID=@SongID;
```

Add Artist Credit

Input: @SongID, @ArtistID

```
    INSERT INTO WRITES
    VALUES (@SongID, @ArtistID);
```

Add Distributor

Input: @DistributorID, @DistributorName

```
    INSERT INTO DISTRIBUTOR
    VALUES (@DistributorID, @DistributorName);
```

Add Represents

Input: @ArtistID, @DistributorID

```
    INSERT INTO REPRESENTS
    VALUES (@ArtistID, @DistributorID);
```

Add Artist

Input: @Name, @About, @ProfilePicture, @BannerPicture

```
    INSERT INTO ARTIST (Name, About, ProfilePicture, BannerPicture, TotalPlays, MonthlyPlays)
    VALUES (@Name, @About, @ProfilePicture, @BannerPicture, 0, 0);
```

Edit Artist

Input: @ArtistID, @Name, @About, @ProfilePicture, @BannerPicture

```
    UPDATE ARTIST
    SET Name =
    CASE WHEN @Name IS NOT NULL THEN @Name
    ELSE Name END
    WHERE ArtistID=@ArtistID;

    UPDATE ARTIST
    SET About =
    CASE WHEN @About IS NOT NULL THEN @About
    ELSE About END
    WHERE ArtistID=@ArtistID;

    UPDATE ARTIST
    SET ProfilePicture =
    CASE WHEN @ProfilePicture IS NOT NULL THEN @ProfilePicture
    ELSE ProfilePicture END
    WHERE ArtistID=@ArtistID;

    UPDATE ARTIST
    SET BannerPicture =
    CASE WHEN @BannerPicture IS NOT NULL THEN @BannerPicture
    ELSE BannerPicture END
    WHERE ArtistID=@ArtistID;
```

Remove Artist

Input: @ArtistID

```
    DELETE FROM SONG
    WHERE SongID IN
        SELECT SongID
        FROM WRITES
        WHERE ArtistID=@ArtistID;

    DELETE FROM ALBUM
    WHERE AlbumID IN
        SELECT AlbumID
        FROM HAS
        WHERE ArtistID=@ArtistID;

    DELETE FROM ARTIST
    WHERE ArtistID=@ArtistID;
```

Get Artist Monthly Plays

Input: @ArtistID

```
    SELECT MonthlyPlays
    FROM ARTIST
    WHERE ArtistID=@ArtistID;
```

Get Artist Details

Input: @ArtistID

```
    SELECT *
    FROM ARTIST
    WHERE ArtistID=@ArtistID;
```

Add Album

Input: @ArtistID, @IsSingle, @CoverArt, @ReleaseDate, @Genre, @NumSongs, @TotalDuration, @Title

```
    INSERT INTO ALBUM (IsSingle, CoverArt, ReleaseDate, Genre, NumSongs, TotalDuration, Title)
    VALUES (@IsSingle, @CoverArt, @ReleaseDate, @Genre, @NumSongs, @TotalDuration, @Title);

    INSERT INTO HAS (AlbumID, ArtistID)
    VALUES (@AlbumID, @ArtistID);
```

Edit Album

Input: @AlbumID, @IsSingle, @CoverArt, @ReleaseDate, @Genre, @NumSongs, @TotalDuration, @Title

```
    UPDATE ALBUM
    SET IsSingle =
    CASE WHEN @IsSingle IS NOT NULL THEN @IsSingle
    ELSE IsSingle END
    WHERE AlbumID=@AlbumID;

    UPDATE ALBUM
    SET CoverArt =
    CASE WHEN @CoverArt IS NOT NULL THEN @CoverArt
    ELSE CoverArt END
    WHERE AlbumID=@AlbumID;

    UPDATE ALBUM
    SET ReleaseDate =
    CASE WHEN @ReleaseDate IS NOT NULL THEN @ReleaseDate
    ELSE ReleaseDate END
    WHERE AlbumID=@AlbumID;

    UPDATE ALBUM
    SET Genre =
    CASE WHEN @Genre IS NOT NULL THEN @Genre
    ELSE Genre END
    WHERE AlbumID=@AlbumID;

    UPDATE ALBUM
    SET NumSongs =
    CASE WHEN @NumSongs IS NOT NULL THEN @NumSongs
    ELSE NumSongs END
    WHERE AlbumID=@AlbumID;

    UPDATE ALBUM
    SET TotalDuration =
    CASE WHEN @TotalDuration IS NOT NULL THEN @TotalDuration
    ELSE TotalDuration END
    WHERE AlbumID=@AlbumID;

    UPDATE ALBUM
    SET Title =
    CASE WHEN @Title IS NOT NULL THEN @Title
    ELSE Title END
    WHERE AlbumID=@AlbumID;
```

Remove Album

Input: @AlbumID

```
DELETE FROM SONG
WHERE SongID IN
    SELECT SongID
    FROM ALBUM_CONTAINS
    WHERE AlbumID=@AlbumID;

DELETE FROM ALBUM
WHERE AlbumID=@AlbumID;
```

Get Album Details

Input: @AlbumID

```
SELECT *
FROM ALBUM
WHERE AlbumID=@AlbumID;
```

Get Album Songs

Input: @PlaylistID

```
SELECT SongID FROM ALBUM_CONTAINS
WHERE AlbumID=@AlbumID;
```

Add Song to Playlist

Input: @PlaylistID, @SongID

```
INSERT INTO PLAYLIST_CONTAINS
VALUES (@PlaylistID, @SongID);
```

Remove Song from Playlist

Input: @PlaylistID, @SongID

```
DELETE FROM PLAYLIST_CONTAINS
WHERE SongID=@SongID
AND PlaylistID=@PlaylistID;
```

Add Playlist

Input: @PlaylistID, @PlaylistName, @CreatorID

```
INSERT INTO PLAYLIST
VALUES (@PlaylistID, @PlaylistName, @CreatorID);
```

Remove Playlist

Input: @PlaylistID

```
DELETE FROM PLAYLIST
WHERE PlaylistID=@PlaylistID;
```

Edit Playlist

Input: @PlaylistID, @PlaylistName

```
UPDATE PLAYLIST
SET PlaylistName=@PlaylistName
WHERE PlaylistID=@PlaylistID;
```

Get Playlist Songs

Input: @PlaylistID

```
SELECT SongID FROM PLAYLIST_CONTAINS
WHERE PlaylistID=@PlaylistID;
```

Get User Playlists

Input: @UserID

```
SELECT (PlaylistID, PlaylistName)
FROM PLAYLIST
WHERE CreatorID=@UserID;
```

Add User

Input: @Username, @PasswordHash

```
INSERT INTO USER(Username, PasswordHash, IsPremium)
VALUES (@Username, @PasswordHash, FALSE);
```

Remove User

Input: @UserID

```
DELETE FROM USER
WHERE UserID=@UserID;
```

View User

Input: @UserID

```
SELECT * FROM USER
WHERE UserID=@UserID;
```

Set Premium

Input: @UserID, @IsPremium

```
UPDATE USER
SET IsPremium=@IsPremium
WHERE UserID=@UserID;
```

Approve Admin

Input: @UserID

```
INSERT INTO ADMIN
VALUES(@UserID);
```

Revoke Admin

Input: @AdminID

```
DELETE FROM ADMIN
WHERE AdminID=@AdminID;
```

Search Songs

Input: @SearchQuery

```
SELECT * FROM SONG
WHERE Title LIKE '%@SearchQuery%';
```

Search Albums

Input: @SearchQuery

```
SELECT * FROM ALBUM
WHERE Title LIKE '%@SearchQuery%';
```

Search Artists

Input: @SearchQuery

```
SELECT * FROM ARTIST
WHERE Name LIKE '%@SearchQuery%';
```

Assumptions

- 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.