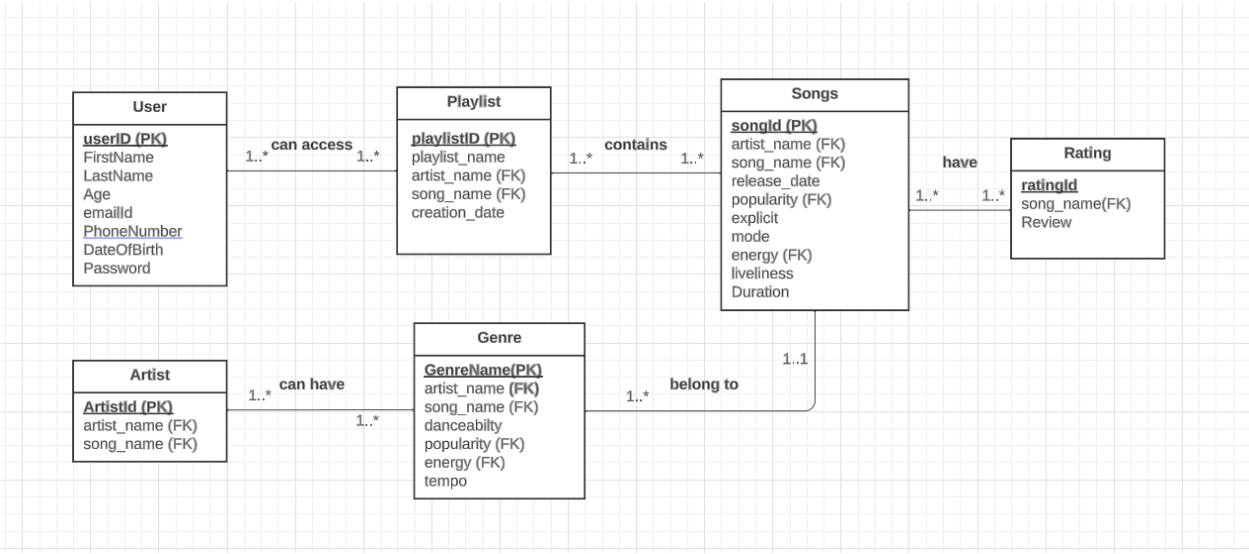
UML Diagram



Relational Schema:

1. User(UserId: INT [PK], Password: INT, FirstName: VARCHAR(255), LastName: VARCHAR(255), Age: INT, Email: VARCHAR(255), PhoneNumber: INT, DateOfBirth: DATE)

2. Playlist(PlaylistId: INT [PK], PlaylistName: VARCHAR(255), ArtistName: VARCHAR(255) [FK to Song.ArtistName], SongName: VARCHAR(255) [FK to Song.ArtistName], CreationDate: DATE)

3. Song(SongId: INT [PK], ArtistName: VARCHAR(255) [FK to Playlist.ArtistName], SongName: VARCHAR(255) [FK to Playlist.SongName], ReleaseDate: DATE, Duration: FLOAT, Popularity: INT, Explicit: INT, Mode: INT, Energy: FLOAT, Liveliness: FLOAT)

4. Genre(GenreName: VARCHAR(255), ArtistName: INT[FK to Songs.ArtistName], SongName: VARCHAR(255)[FK to Songs.SongName], danceability: FLOAT, popularity: INT [FK to Song.popularity, energy: FLOAT [FK to Song.Energy], tempo: FLOAT

5. Artist(ArtistName: INT[FK to Genre.ArtistName], SongName: VARCHAR(255)[FK to Genre.SongName], ArtistId:VARCHAR(255)

6. Rating(RatingId: INT [PK], SongName: VARCHAR(255) [FK to Song.SongName], Review: INT)

Normalizing Database:

1. User(UserId, Password, FirstName, LastName, Age, Email, PhoneNumber, DateOfBirth) with FD:
   UserId -> Password, FirstName, LastName, Age, Email, PhoneNumber, DateOfBirth

2. Playlist(PlaylistId, PlaylistName, ArtistName, SongName, CreationDate) with FD:
   PlaylistId -> PlaylistName, ArtistName, SongName, CreationDate

3. Song(SongId, ArtistName, SongName, ReleaseDate, Duration, Popularity, Explicit, Mode, Energy, Liveliness) with FD:
   SongId -> ArtistName, SongName, ReleaseDate, Duration, Popularity, Explicit, Mode, Energy, Liveliness

4. Artist(ArtistName, SongName, ArtistId) with FD:
   ArtistId -> ArtistName, SongName

5. Genre(ArtistName, SongName, danceability, popularity, energy, tempo) with FD:
   GenreName -> ArtistName, SongName, danceability, popularity, energy, tempo

6. Rating(RatingId, SongName, Review) with FD:
   RatingId -> SongName, Review

Why use BCNF over 3NF?

Because all of our tables have one primary key, it complies with the definition of BCNF. One advantage of using BCNF is to prevent not expected data insertion, deletion, and update in our data sets. Furthermore, since BCNF eliminates the partial dependencies, it reduces data redundancies to a greater extent than utilizing 3NF.

Since all the left sides in the Normalizing Database are keys, all relations stated above are in BCNF.

Assumptions

- We assume that each user can create many playlists and each playlists can be viewed by many other users. For example, in Spotify an user can create a playlist and make it public which can be viewed by other users

- We assume that each playlist can have as many songs and each song can be a part of many playlists.

- We assume that each song can get many ratings from different users and each rating can also have many songs associated with.

- We assume that each song can only have one genre but each genre can have many songs.

- We assume that each artist can have songs from different genres and each genre can have many songs.

Description of Relational Cardinality

- User to Playlist : This is a many to many relation as many users can have create and have access to many playlists

- Playlist to Songs : This is a many to many relation as many playlist can have many songs and each song can be part of a playlist

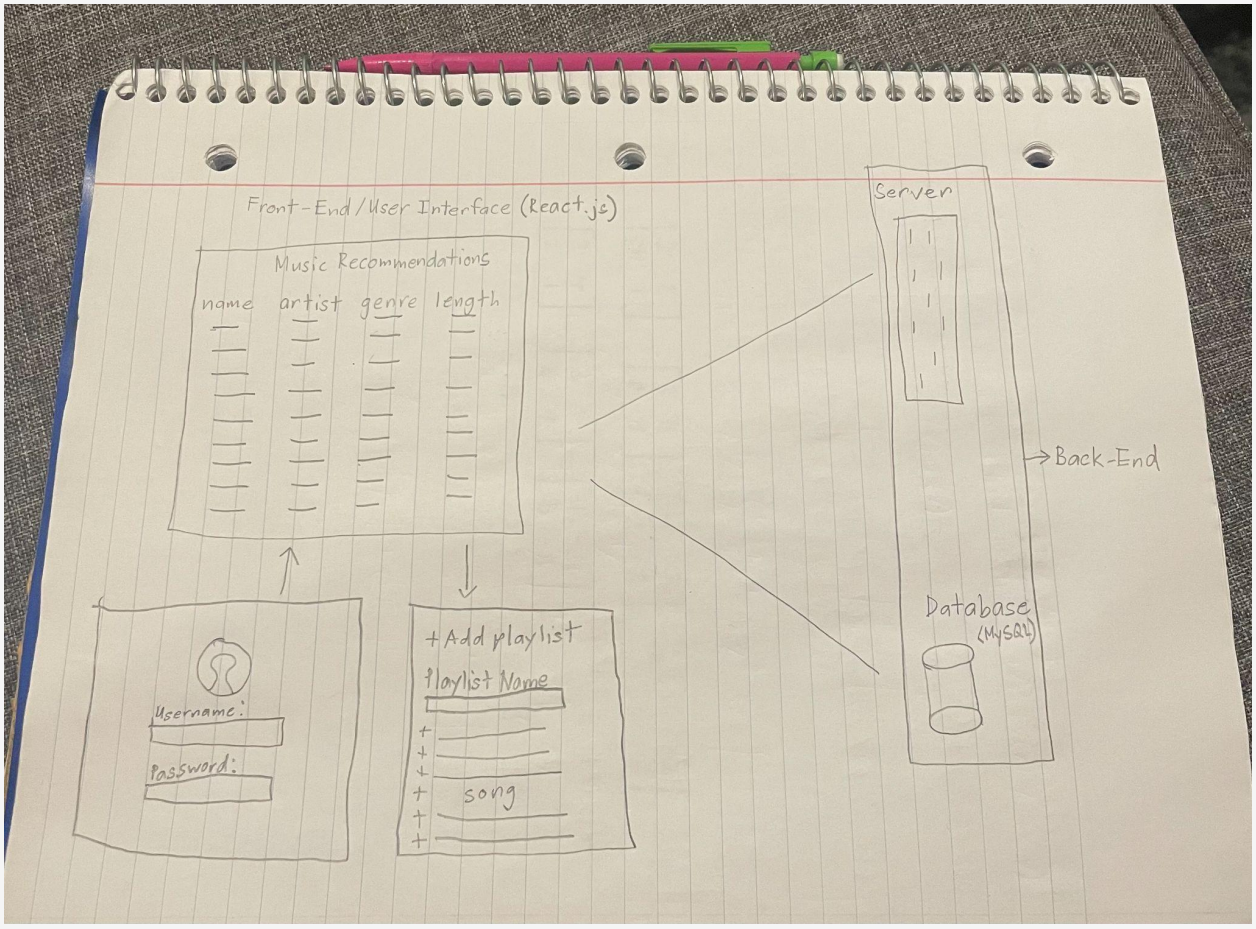
- Songs to Rating : This is a many to many relation as a song can get different ratings from different users and each rating can have different songs associated with.

- Songs to Genre : This is many to one relation as there can be many songs belonging to one genre but also each song can have only one genre

- Genre to Artist : This is a many to many relation as each genre can have many artists and many artists can have songs of different genres.

Changes made to Stage 1 Report

- TA's remarks : **Deducted 1 point because of the unclear description. We ask for a more elaborate description which is missing. The UI mockup doesn't have sufficient information, thus I am deducting 1 point.**



The application has an User login page with password support to boost security and privacy.
The basic functionalities of our application include users searching for song recommendations based on a certain set of attributes (genre, artist, etc.). We can also give users the option to input any songs that they feel people can be interested in. One cool feature that we think we could implement is a functionality where users can rate songs that they listen to (1-10, 0-5 stars, etc.). When a user gives a good rating for a song, it is more likely to get recommended to people who are looking for a new song to enjoy. We can achieve this by editing our database to allow for an option where users can input a rating for each song they listen to, and therefore take the average rating and recommend based off of the number. We've also added a playlist option where the user can create their playlist and can make it public so it can be accessed by other users. We've also added a song recommendation feature based on the user's mood. In order to prevent children from listening to obscene songs, every song with inappropriate words will be marked explicit. The user can also get song recommendations based on its popularity. If the user feels like dancing, they can select the danceability option and get song recommendations based on that.