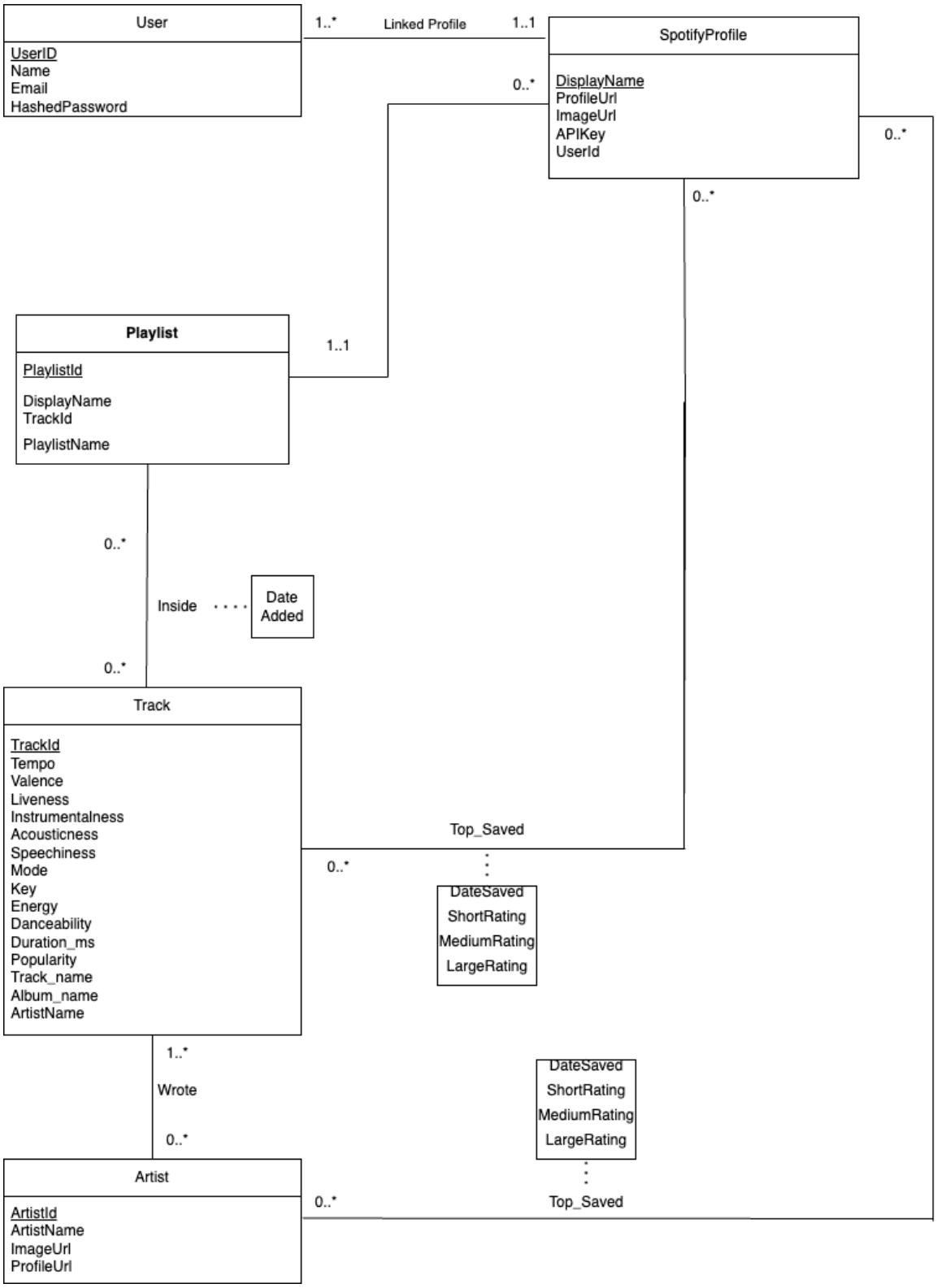
# Wave Match DB Design

## UML Diagram:



## Relational Schema

**User:**
UserID: VARCHAR(255) [PK]
Name: VARCHAR(255)
Email: VARCHAR(255)
HashedPassword: VARCHAR(255)

**Playlist:**
PlaylistID: VARCHAR(255) [PK]
PlaylistName: VARCHAR(255)
DisplayName: VARCHAR(255) [FK]
TrackId: VARCHAR(255) [FK]

**Track:**
TrackId: VARCHAR(255) [PK]
ArtistName: VARCHAR(255) [FK]
Album_name: VARCHAR(255)
Track_name: VARCHAR(255)
Popularity: INT
Duration_ms: INT
Danceability: FLOAT (5,5)
Energy: FLOAT (5,5)
Key: INT
Mode: INT
Speechiness: FLOAT (5,5)
Acousticness: FLOAT (5,5)
Instrumentalness: FLOAT (5,5)
Liveness: FLOAT (5,5)
Valence: FLOAT (5,5)
Tempo: INT
Time_signature: INT
Track_genre: VARCHAR(255)

**SpotifyProfile:**
DisplayName: VARCHAR(255) [PK]
UserID: VARCHAR(255) [FK]
ProfileUrl: VARCHAR(255)

ImageURL: VARCHAR(255)
APIKey: VARCHAR(255)

**Artist:**
ArtistID: VARCHAR(255) [PK]
ArtistName: VARCHAR(255)
ImageURL: VARCHAR(255)
ProfileURL: VARCHAR(255)

# Normalization:

1. User Table:
   - The User table has the following attributes: UserID (Primary Key), Name, Email, and HashedPassword.
   - This table does not contain any partial dependencies or transitive dependencies.
   - It is already in BCNF because UserID is the primary key and determines all other attributes.
   - User(UserID -> Name, Email, HashedPassword)

2. Playlist Table:
   - PlaylistID (Primary Key), PlaylistName, DisplayName (Foreign Key), and TrackId (Foreign Key).
   - There are no partial dependencies.
   - The DisplayName determines PlaylistName and indirectly TrackId.
   - The TrackId determines nothing.
   - To bring this table to 3NF, we need to remove the transitive dependency between DisplayName and PlaylistName.

- We can separate the dependency by splitting up DisplayName as the key and having it lead to PlaylistID. This removes the transitive dependency.
- UserPlaylist(DisplayName -> PlaylistID)
- Playlist(PlaylistID -> PlaylistName)
- PlaylistTrack(PlaylistID, TrackId)

3. Track Table:
- TrackId (Primary Key), ArtistName, Album_name, Track_name, Popularity, Duration_ms, Danceability, Energy, Key, Mode, Speechiness, Acousticness, Instrumentalness, Liveness, Valence, Tempo, Time_signature, and Track_genre.
- This table does not have any partial or transitive dependencies.
- It is already in BCNF because TrackId is the primary key and determines all other attributes.
- Song(TrackId -> ArtistName, Album_name, Track_name, Popularity, Duration_ms, Danceability, Energy, Key, Mode, Speechiness, Acousticness, Instrumentalness, Liveness, Valence, Tempo, Time_signature, Track_genre)

4. SpotifyProfile Table:
- DisplayName (Primary Key), UserID (Foreign Key), ProfileUrl, ImageURL, and APIKey.
- This table does not have any partial or transitive dependencies.
- It is already in BCNF because DisplayName is the primary key and determines all other attributes.
- SpotifyProfile(DisplayName -> UserID, ProfileUrl, ImageURL, APIKey)

5. Artist Table:

- ArtistID (Foreign Key), ArtistName, ImageURL, ProfileURL
- There are no partial dependencies.
- This table does not contain any transitive dependencies.
- It is already in BCNF because ArtistId is the primary key.
- Artist(ArtistId -> ArtistName, ImageURL, ProfileURL)

# Relationships & Cardinality:

One-to-one:  Every user would have one Spotify profile with a unique UserId which is the primary key. Since each record in the SpotifyProfile table is linked to exactly one record in the Users table, and each record in the Users table can only have one record in the SpotifyProfile table, we can state that this is a one-to-one relationship

One-to-many: Every record in the SpotifyProfile table would have multiple records in the Playlist table, but every record in the Playlist table can only have one record in the SpotifyProfile table. Therefore, we can state that this is a one-to-many relationship.

Many-to-many: Every record in the Playlist table can have multiple records in the Track table, and every record in the Track table can have multiple records in the Playlist table. Therefore, this is a many-to-many relationship.

Many-to-many: Every record in the SpotifyProfile table can have multiple records in the Track table and every record in the Track table can have multiple records in the SpotifyProfile table. Therefore, this is a many-to-many relationship.

Many-to-many: Every record in the Track table can have multiple records in the Artist table and every record in the Artist table can have multiple records in the Track table. Therefore, this is a many-to-many relationship.

Many-to-many: Every record in the SpotifyProfile table can have multiple records in the Artist table and every record in the Artist table can have multiple records in the SpotifyProfile table. Therefore, this is a many-to-many relationship.

# Assumptions of UML Diagram:

- For our database, we will make every UserId unique
- Our Playlists table must have at least one value and one song associated with it
- We can assume that if the URL is invalid then the default URL picture will be updated
- All information is required for the User table
- Other information relating to music is purely derived from the dataset, and the user will have choices to pick their songs to receive recommendations accordingly