YouTune

Group Members: Sherwin Fernandes, Theo Bongolan, Kyle Torres, Ben Lopez

1. Introduction and Overview

1.1 Introduction

This software will be a way of sharing music. The system should allow the user to make requests of other users to listen to a song and then will be able to give their own feedback to the requester user

1.2 Software Overview

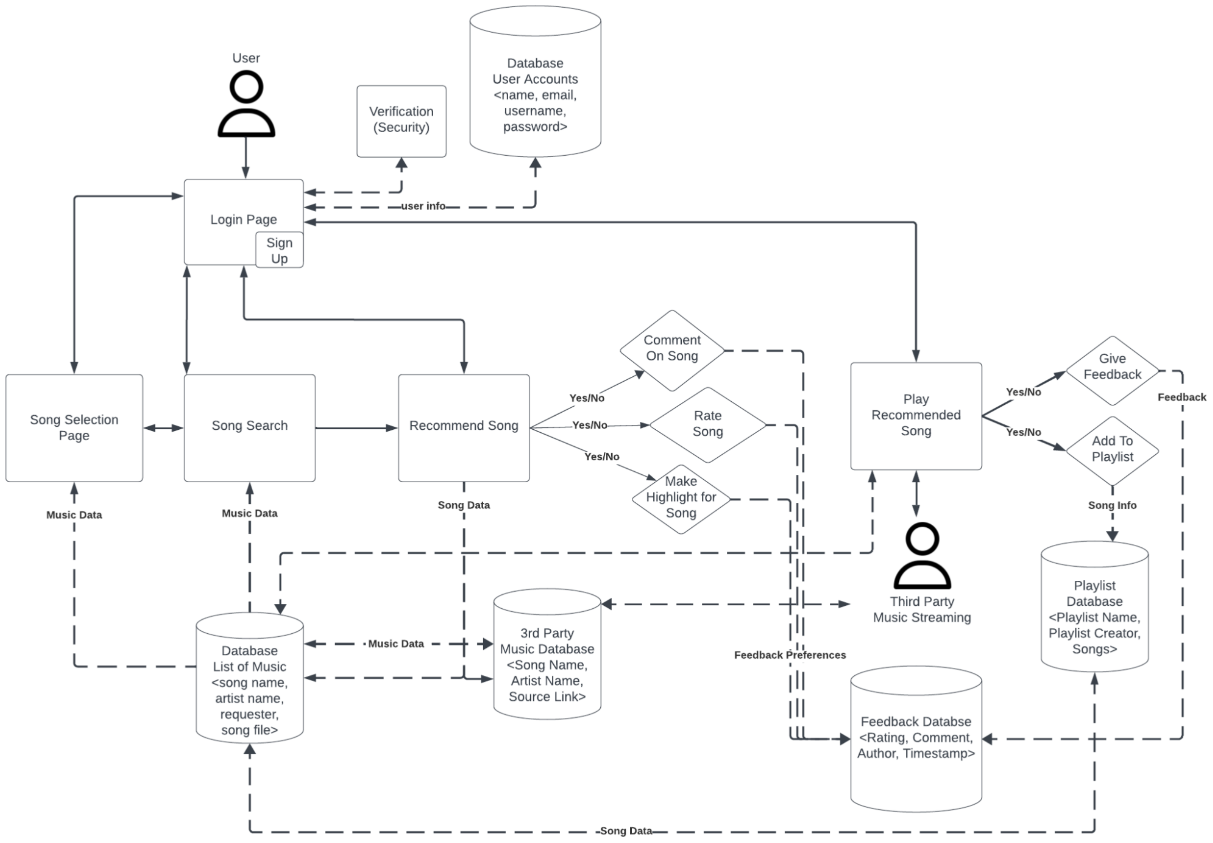
The software will mainly allow users to share songs from multiple music platforms and allow the listeners to browse requested songs made by other users. Every user will be a listener and requester. With every song, listeners will be able to give a rating and feedback.

1.3 Document Overview

Section 2: Software Architecture Diagram
Section 3: UML Class Diagram
Section 4: Development Timeline
Section 5: Test Plan
Section 6: Data Management Strategy

2. Software Architecture Diagram

2.1 Diagram


User
Verification (Security)
Database User Accounts <name, email, username, password>
user info
Login Page
Sign Up
Song Selection Page
Song Search
Recommend Song
Comment On Song
Rate Song
Make Highlight for Song
Yes/No
Yes/No
Yes/No
Play Recommended Song
Give Feedback
Add To Playlist
Yes/No
Yes/No
Feedback
Song Info
Music Data
Music Data
Song Data
Third Party Music Streaming
Playlist Database <Playlist Name, Playlist Creator, Songs>
Database List of Music <song name, artist name, requester, song file>
Music Data
3rd Party Music Database <Song Name, Artist Name, Source Link>
Feedback Preferences
Feedback Databse <Rating, Comment, Author, Timestamp>
Song Data

2.2 Component Description

User

The user is an entity that will be able to interact with the Login/Signup Page

Login/Signup Page

Gives the user access to the software and serves as a gateway to provide the user with the services available, using the verification component and user account database for added security and

User Accounts Database

Stores sensitive account data for users (name, email, username, password)

Song Selection Page

Allows users to browse through the songs available in the database

Song Search

Allows users to search for specific songs by song name, artist, or requests from a user that will look through the Music Database

Music Database

Stores data about the song name, artist, user that made the rating request, and the song file

3rd Party Music Database

Stores information on songs that are connected to external streaming platforms that'll store data on the song name and artist, with a link to get the song file

Recommend Song

Allows users to make requests to share songs by either linking a song or uploading their own (feature for future release) and setting preferences for what feedback they will receive (Allow commenting, rating, and/or highlighting timestamp)

Play Recommended Song

Pulls Song Data from the Music Database if available or else pulls from the 3rd Party Music Database and streams it for the user. Allows users to give feedback and add to a playlist

Playlist Database

Stores data on playlists created by users (playlist name, playlist creator, songs list)

Third-Party Music Streaming Service

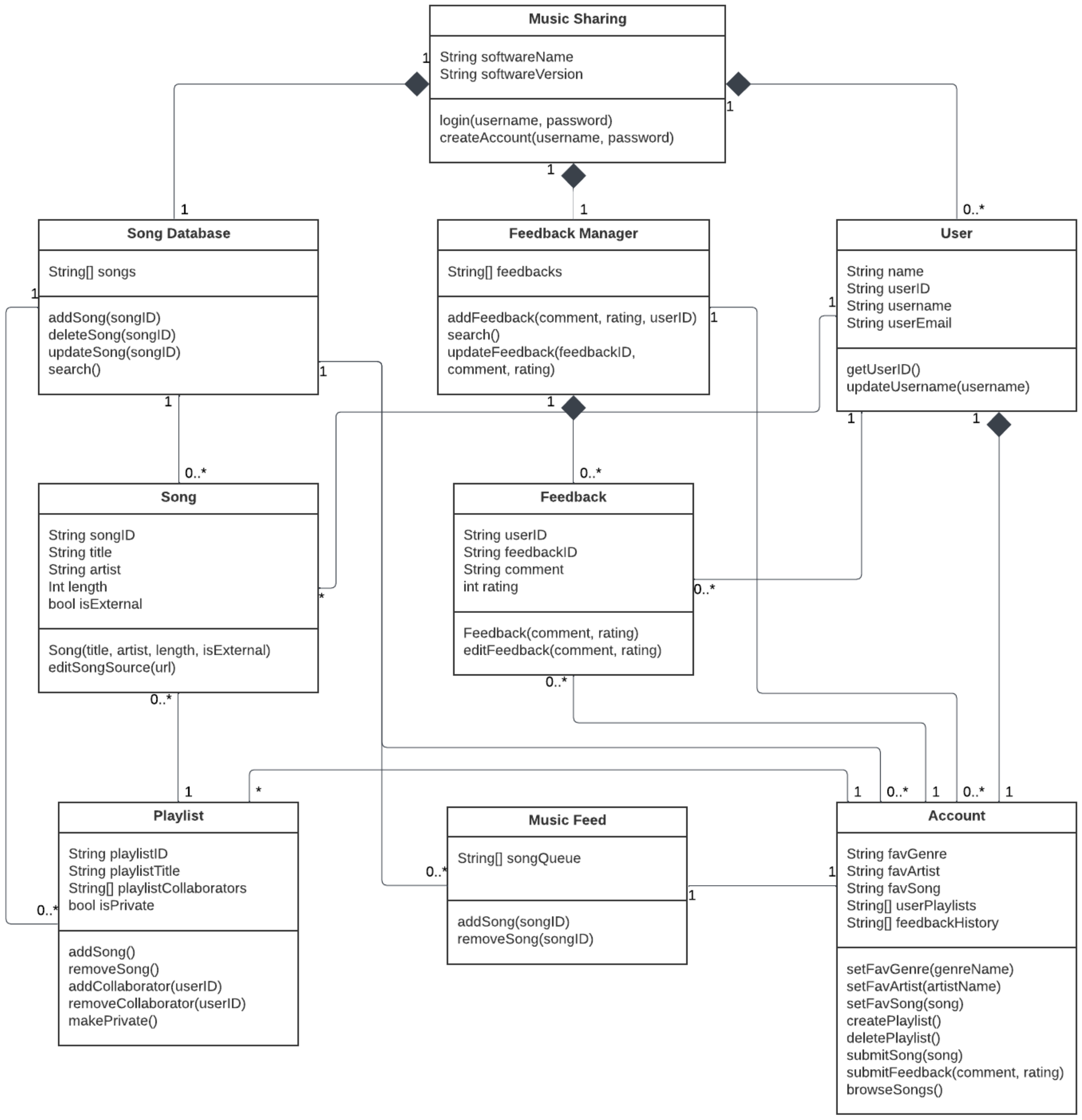
Pulls song data from a 3rd party streaming service if necessary to stream a song

Feedback Database

Stores data on user feedback whenever users submit feedback on a song recommendation

3. UML Class Diagram

3.1 Diagram



3.2 Class Description

Music Sharing: main class for the software system

Attributes:

- String softwareName: name of the software
- String softwareVersion: tracks the current version of the system

Operations:

- Login(username, password): allows users to log into the system
- createAccount(name, username, email, password): takes user input to create an account for the user

Feedback Manager: Class for storing, tracking, and managing feedback made within the system

Attributes:

- String[] feedbacks: stores an array of all user feedbacks

Operations:

- addFeedback(comment, rating, userID): add a feedback to the database
- search(): look through the feedback database
- updateFeedback(feedbackID, comment, rating): update/edit a user's feedback

Song Database: Class for storing, tracking, and managing songs added within the system

Attributes:

- String[] songs: stores an array of all songs within the system

Operations:

- addSong(songID): adds song to the database
- deleteSong(songID): deletes song from the database
- updateSong(songID): updates information about the song within the database
- search(): allows searching through the database

<u>User:</u> Class for each user account

Attributes:

- String name: store user’s name on account
- String userID: stores generated userID for account
- String username: stores account username
- String userEmail: stores user’s email on account

Operations:

- getUserID(): returns the userID for an account
- updateUsername(username): updates the username

<u>Song:</u> Class for songs submitted by users

Attributes:

- String songID: generated id connected to a song
- String title: the name of a song
- String artist: name of whoever created the song
- int length: length of the song
- bool isExternal: true if the song is sourced from a 3rd party streaming service

Operations:

- Song(title, artist, length, isExternal): sets all the variables when creating a new song object
- editSongSource(url): allows source for externally linked songs to have the url updated if needed

<u>Playlist:</u> Class for letting users store a list of songs

Attributes:

- String[] playlistID: Playlist’s unique identifier
- String playlistTitle: name of the playlist
- String playlistCollaborators: list of collaborators for the playlist
- bool isPrivate: true if the playlist is not publicly accessible

Operations:

- addSong(): adds a song to the playlist
- removeSong(): removes a song from the playlist
- addCollaborator(userID): Gives user permission to add to this playlist
- removeCollaborator(userID): Revokes user permission to add to this playlist
- toggleVisibility(): switches the playlist privacy between public/private

<u>Music Feed:</u>
Attributes:

- String[] songQueue: list of songIDs in a given queue

Operations:

- addSong(songID): adds a specified song to the song queue
- removeSong(songID): removes a specified song from the queue
- togglePlay(): allows user to play/pause the queue

<u>Account:</u>
Attributes:

- String favGenre: A string storing a listeners favorite Genre of song
- String favArtist: A string storing a listeners favorite musical Artist
- String favSong: A string storing a listeners favorite Song
- String[] userPlaylists: array of all playlistID’s created on the account
- String[] feedbackHistory: array of all feedback submissions made on the account

Operations:

- setFavGenre(genreName): sets or updates a Listener's favGenre string
- setFavArtist(artistName): sets or updates a Listener’s favArtist string
- setFavSong(song): sets or updates a Listener’s favSong string
- createPlaylist(): allows user to create a new playlist on their account
- deletePlaylist(): allows users to delete their playlists on their account
- submitSong(song): allows users to submit a song for others to rate
- submitFeedback(comment, rating): allows user to submit feedback on a song
- browseSongs(): allows user to look through available song submissions

<u>Feedback:</u>
Attributes:

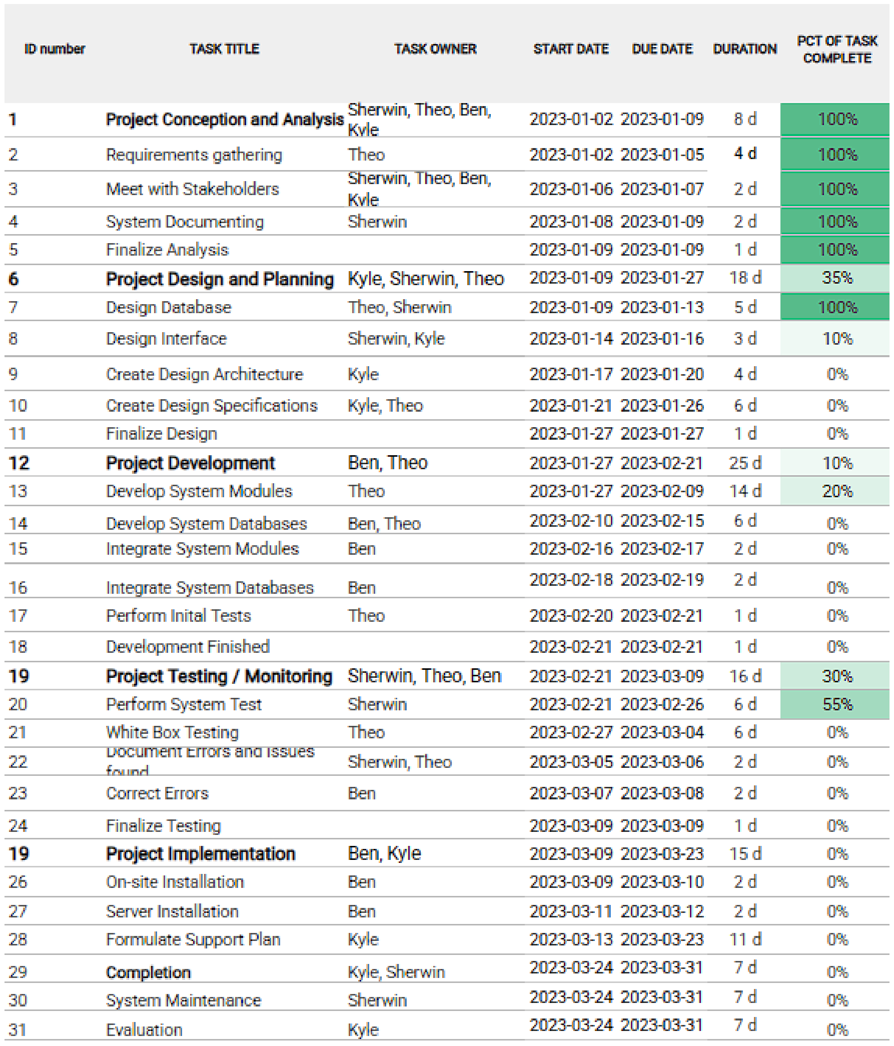
- String userID: user that created the feedback
- String feedbackID: generated ID for a user’s feedback
- String comment: user’s comment for a song
- int rating: user’s rating for a song

Operations:

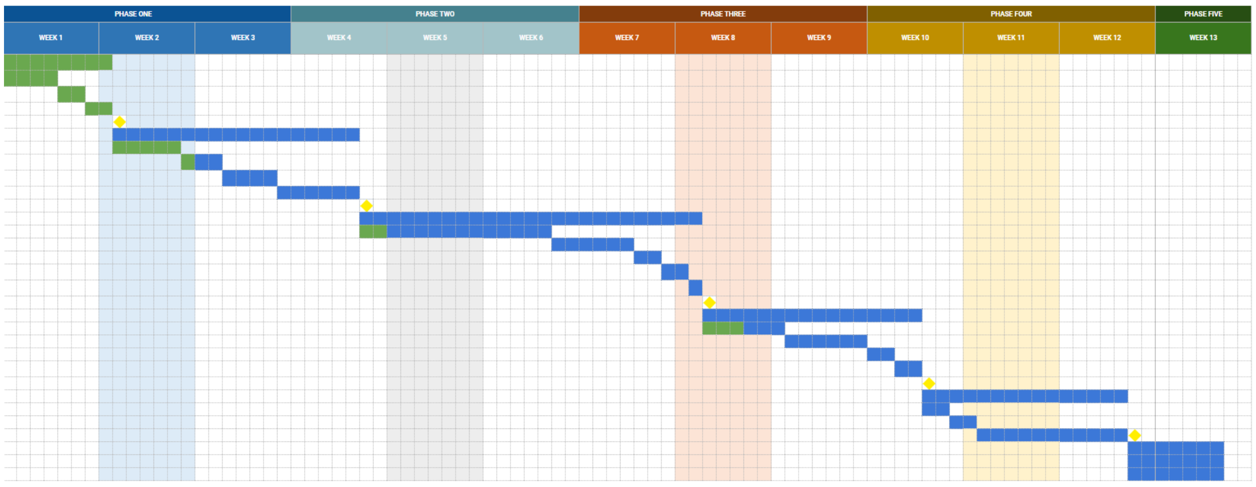
- Feedback(userID, comment, rating): constructor to store the comment/rating
- editFeedback(comment, rating): allows user to edit their feedback

4. Development Timeline

4.1 Timeline: View the full timeline here



ID number	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	**Project Conception and Analysis**	Sherwin, Theo, Ben, Kyle	2023-01-02	2023-01-09	8 d	100%
2	Requirements gathering	Theo	2023-01-02	2023-01-05	4 d	100%
3	Meet with Stakeholders	Sherwin, Theo, Ben, Kyle	2023-01-06	2023-01-07	2 d	100%
4	System Documenting	Sherwin	2023-01-08	2023-01-09	2 d	100%
5	Finalize Analysis		2023-01-09	2023-01-09	1 d	100%
6	**Project Design and Planning**	Kyle, Sherwin, Theo	2023-01-09	2023-01-27	18 d	35%
7	Design Database	Theo, Sherwin	2023-01-09	2023-01-13	5 d	100%
8	Design Interface	Sherwin, Kyle	2023-01-14	2023-01-16	3 d	10%
9	Create Design Architecture	Kyle	2023-01-17	2023-01-20	4 d	0%
10	Create Design Specifications	Kyle, Theo	2023-01-21	2023-01-26	6 d	0%
11	Finalize Design		2023-01-27	2023-01-27	1 d	0%
12	**Project Development**	Ben, Theo	2023-01-27	2023-02-21	25 d	10%
13	Develop System Modules	Theo	2023-01-27	2023-02-09	14 d	20%
14	Develop System Databases	Ben, Theo	2023-02-10	2023-02-15	6 d	0%
15	Integrate System Modules	Ben	2023-02-16	2023-02-17	2 d	0%
16	Integrate System Databases	Ben	2023-02-18	2023-02-19	2 d	0%
17	Perform Inital Tests	Theo	2023-02-20	2023-02-21	1 d	0%
18	Development Finished		2023-02-21	2023-02-21	1 d	0%
19	**Project Testing / Monitoring**	Sherwin, Theo, Ben	2023-02-21	2023-03-09	16 d	30%
20	Perform System Test	Sherwin	2023-02-21	2023-02-26	6 d	55%
21	White Box Testing	Theo	2023-02-27	2023-03-04	6 d	0%
22	Document Errors and Issues found	Sherwin, Theo	2023-03-05	2023-03-06	2 d	0%
23	Correct Errors	Ben	2023-03-07	2023-03-08	2 d	0%
24	Finalize Testing		2023-03-09	2023-03-09	1 d	0%
19	**Project Implementation**	Ben, Kyle	2023-03-09	2023-03-23	15 d	0%
26	On-site Installation	Ben	2023-03-09	2023-03-10	2 d	0%
27	Server Installation	Ben	2023-03-11	2023-03-12	2 d	0%
28	Formulate Support Plan	Kyle	2023-03-13	2023-03-23	11 d	0%
29	**Completion**	Kyle, Sherwin	2023-03-24	2023-03-31	7 d	0%
30	System Maintenance	Sherwin	2023-03-24	2023-03-31	7 d	0%
31	Evaluation	Kyle	2023-03-24	2023-03-31	7 d	0%



5. Test Plan

5.1 System Testing

__**Testing login and changing username functionality via black box testing**__

1. Start by accessing the website via an internet browser software.

   a) Upon reaching the launch page the tester will first enter invalid email with a valid password, expecting an error.

   b) Then the tester will enter a valid email with an invalid password, also expecting an error.

   c) Finally, input a valid email and password in order to successfully login.

   d) Verify That the login was successful

2. Upon logging in the tester will navigate the account to reach the change username option and page.

   a) Upon selecting the change username option tester will input an invalid username, such as a blank string and confirm a failed change.

   b) Then the tester will enter a valid new username.

3. The tester will then select ok, and verify that the new username is the same as the input username.

   a) Verify the new username had updated.

Testing creating and manipulating a Playlist via black box testing

As a prerequisite the tester will have already logged in with a valid account.

1. Tester will navigate through the account page and select the option to create a new Playlist.

   a) They will first enter an invalid name, such as a blank string, and verify failed input.

   b) The tester will then enter a valid Playlist name and verify the creation of a new Playlist.

   c) They will then verify a new playlist has been created

2. Tester will access the Playlist and add a new song.

   a) They will input an incorrect string to verify input error.

   b) Then they will input a wrong song name to verify correct string but mismatch in songs database

   c) They will then input a correctly spelled song, which is also in the songs database, and verify correct input

   d) They will then verify that the song has been added to the playlist

3. Tester will then add a collaborator to the Playlist

   a) When prompted to input collaborators name the will input an invalid string and verify input error

   b) Then they will input an invalid username, to verify correct string but mismatch in user database

   c) They will then input a correctly spelled username that is also a user in the database

   d) They will verify the added collaborator to the playlist

4. Tester will then remove a song from the playlist

   a) They will select the remove song option, then when prompted input and invalid string to verify input error

   b) They will then input an invalid song name that is not in the database to verify database mismatch

   c) They will then input a valid song in the database but one that isn't in the the playlist to verify correct input but no match in playlist

   d) They will then input a correct song name in the playlist and song database

   e) They will then verify the song has been removed from the playlist

5.2 Functional Testing

<u>Music recommending/sharing via black box testing</u>

With our software being about having the ability to recommend a song between users we will want to test whether the ability to share music is working. We want to make sure that users will be able to share or recommend music to other users as well as receive music from other users on the platform

As a prerequisite the tester will have already logged in with a valid account.

Testing steps taken:
1. Search for songs
2. Add other songs
3. Share song(s)

Expected Outcome:
Users on the platform should be able to have the option to send music and or receive music with one another

<u>Feedback System via black box testing</u>

Users will be able to interact with each other by submitting feedback on songs that have been recommended to them by other users. Users can also leave feedback on their own songs that they have shared.

As a prerequisite the tester will have already logged in with a valid account.

Testing steps taken:
1. Play a recommended song
2. Leave feedback on song (rating and comment)

Expected outcome:
When playing a song users will have different options to either share the current song with other users or leave feedback for the song such as ratings or commenting on the song that they are listening to.

5.3 Unit Testing

<u>Testing Song Requests in Queue via white box testing</u>

Intended function: The request queue will display the number of songs that others have requested the user to listen to.

Assume user is logged in & number of songs in queue is 5, the number of requested songs is correctly displayed after the following in order:

1. Another user has made a request of the listener for a song

    - Expects: 6 songs in queue

2. Another user removes their previous request

    - Expects: 5 songs in queue

3. Listener interacts with the song and removes it from queue

    - Expects: 4 songs in queue

<u>Testing Songs in playlist via white box testing</u>

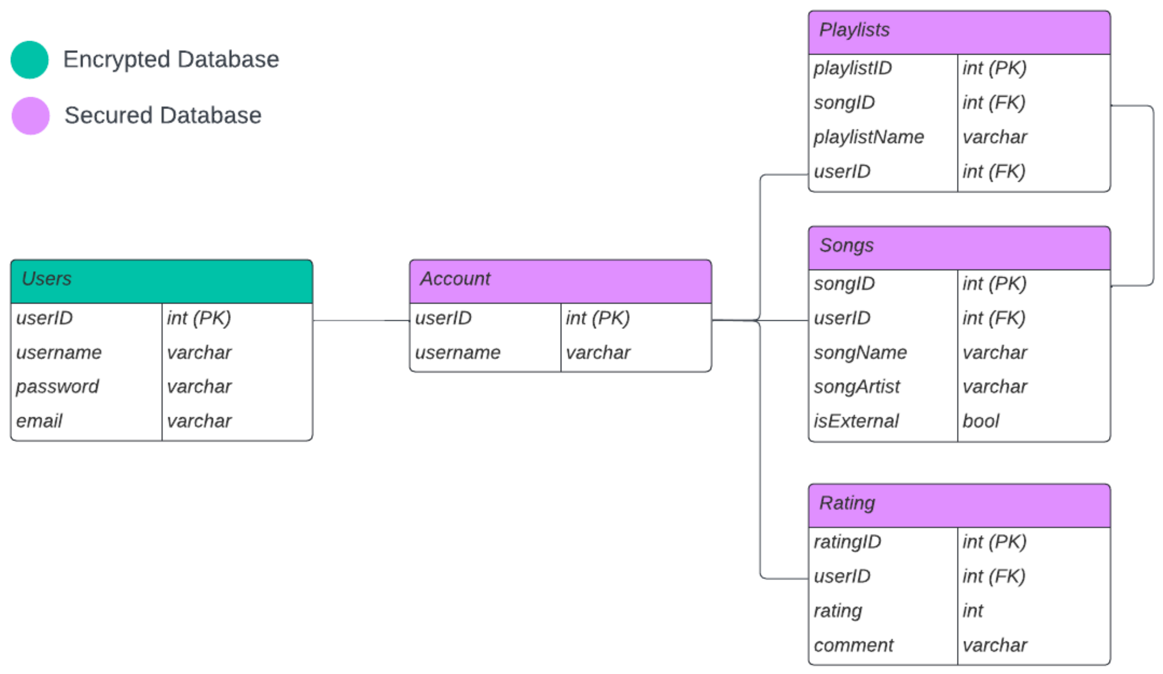
Intended function: The number of songs is correctly displayed in a created playlist.

Assume user is logged in and an empty playlist is created.

1. User adds a song
    - Expects: 1 song
2. User adds another song
    - Expects: 2 songs
3. User removes a song
    - Expects: 1 song
4. Collaborator adds a song
    - Expects: 2 songs
5. Collaborator adds another song
    - Expects: 3 songs

6. Data Management Strategy

6.1 Diagram



6.2 Diagram Description

For our Data Management we elected to use an SQL system, as this gives us good data security and retrieval as well as options to scale up in the future. We chose to use 2 databases, one encrypted database and one secured database. We chose to separate the databases to help ensure proper user data security. For instance we chose to have a separate User and Account table, one encrypted and one secured. This enables us to retrieve the user's username without accessing encrypted data, potentially putting the user's data at risk of an attack. We split the data up into blocks that each represent a part of the software. For instance, there is a table for Songs, Ratings, and Playlists. Each of these are assets that don't have enough in common to warrant a single bloated table. We link the Playlist and Songs table with the Primary Key Foreign Key pair songID as logically a playlist if composed of many songs. All the tables are linked together with the PK FK of userID. This allows all the tracking of the creator of the playlists, song uploads, and rating.

6.3 Design Alternatives

A trade-off we made when creating the database was making two separate databases between users and the account. By making this decision we are trying to reinforce security at the cost of making the system more complex. The user database being made as an encrypted database allows for an extra security layer for sensitive information about the user and allows users to access their account without always having to access said sensitive information. The alternative is making everything under one whole database which reduces the amount of security but would also reduce the complexity of the system.