CMPUT 291 Mini-Project #1

A Music Platform Application using Python and SQLite3

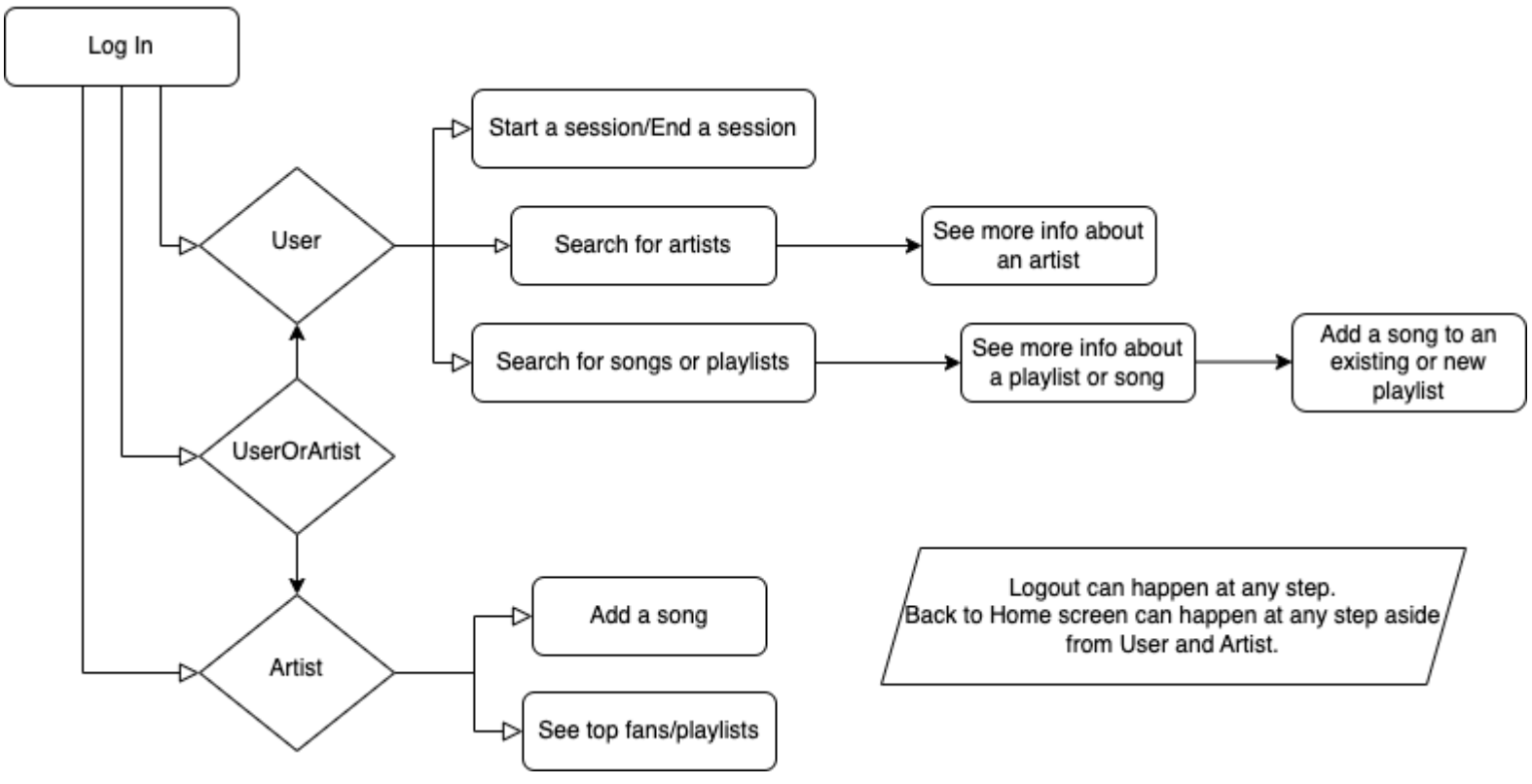
A. System Overview

Our application acquires some of a music streaming platform's basic features that provide services to the users using data from a database. It was created using Python and its standard GUI library named Tkinter for the functionalities and SQLite3 for data management from the database. The program is similar to a music streaming application where the users are either music listeners or artists. What the users can do with the application is very simple. First, they can log in using their unique ID and passwords, which they have registered and stored in the database. Then, depending on whether the user is a listener, an artist, or even both, they can perform several actions on the platform, such as starting and ending a music session or creating a song playlist (for listeners) or adding a song to the application's database (for artists).

The most dominant part of the program's design lies in using Tkinter, a Python graphical user interface. The application's features are displayed or acted on by the users through buttons, labels, and search entries on the interface, contributing to the user experience. The GUI is also useful during the creation process by helping the programmers navigate easily and thus be less likely to make mistakes due to its simplicity and visibility. Python was chosen to write the program because of its compatibility with SQLite - the go-to language for managing data using a relational model. Below is a small user guide on how to use the program.

User Guide (for Linux users):

1. Start the program using the command **make run**. You need to input the relative path of the database
2. Log in using your ID or password. If you do not have one yet, please register. After finishing registering, log in again.
3. If you are both a user and an artist, you will be asked to choose which role you want to log in as. Otherwise, you will be directed to the Home screen.
4. Users can perform some actions: start a music session, end an ongoing session, search for songs, playlists, and artists available on the platform, and see more information about the playlist/song/artist you select on another page after searching. You can add a song to an existing playlist or a new playlist. You can also log out from the Home screen and log out and return to the Home screen from any page.
5. Artists will also have some options to perform some actions: you can upload a song that you perform to the platform and see your top fans and playlists. You can also log out from the Home screen and log out and return to the Home screen from any page.



B. Software Design

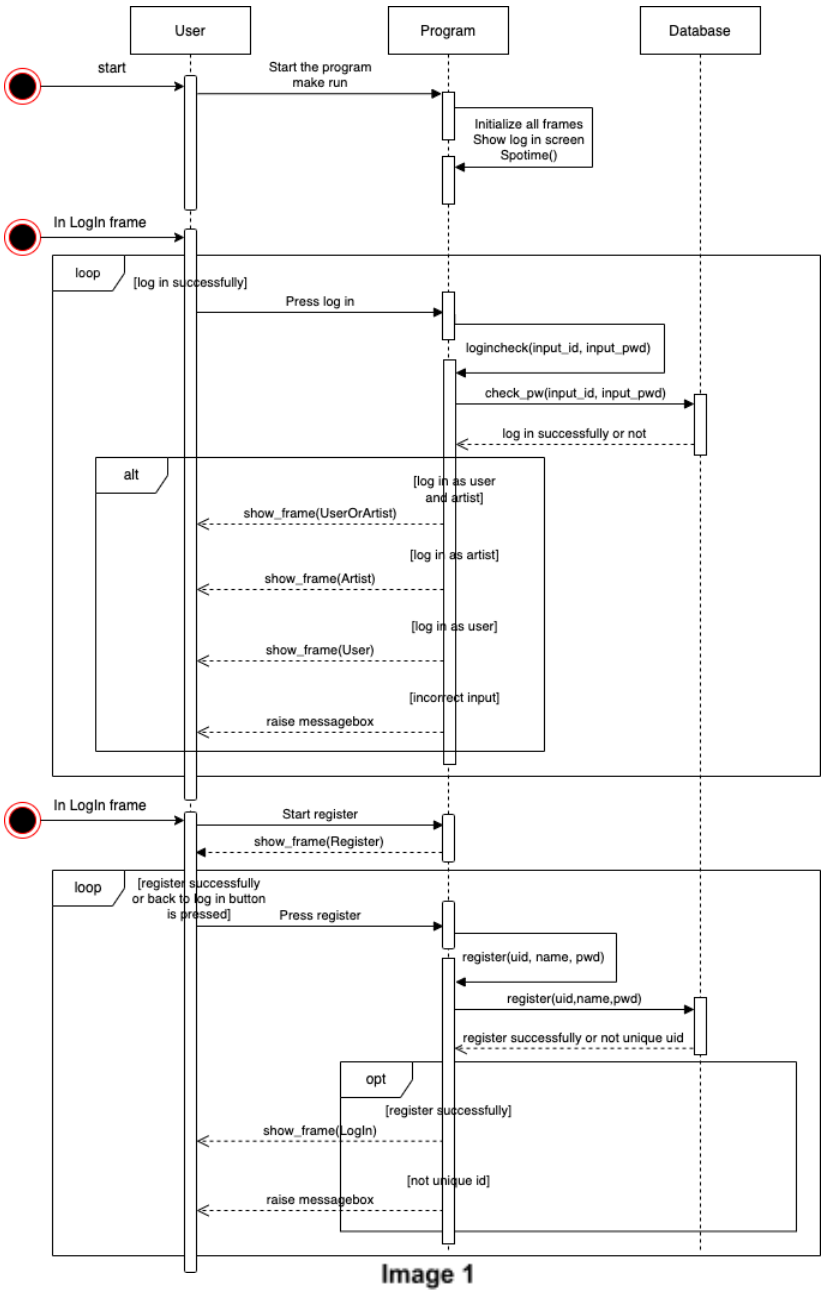


Image 1

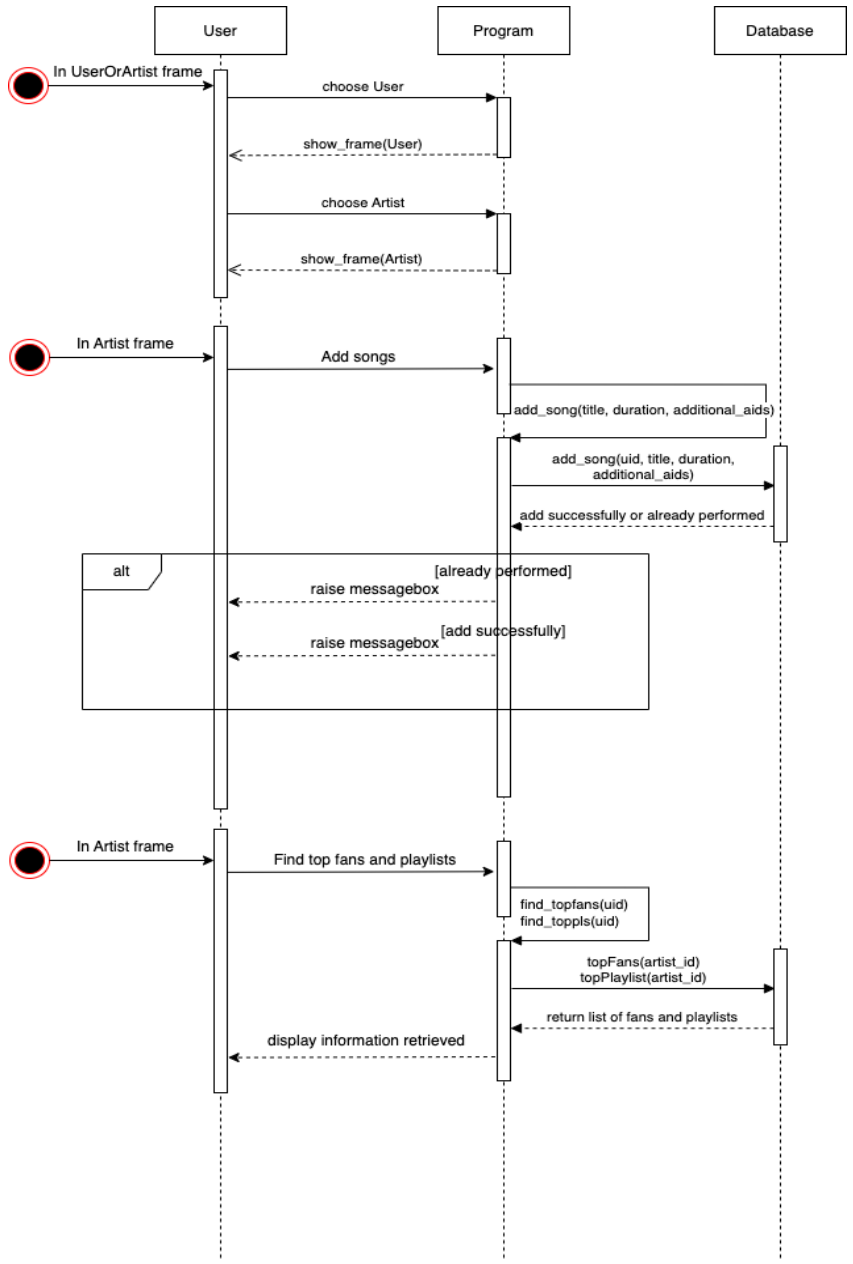


Image 2

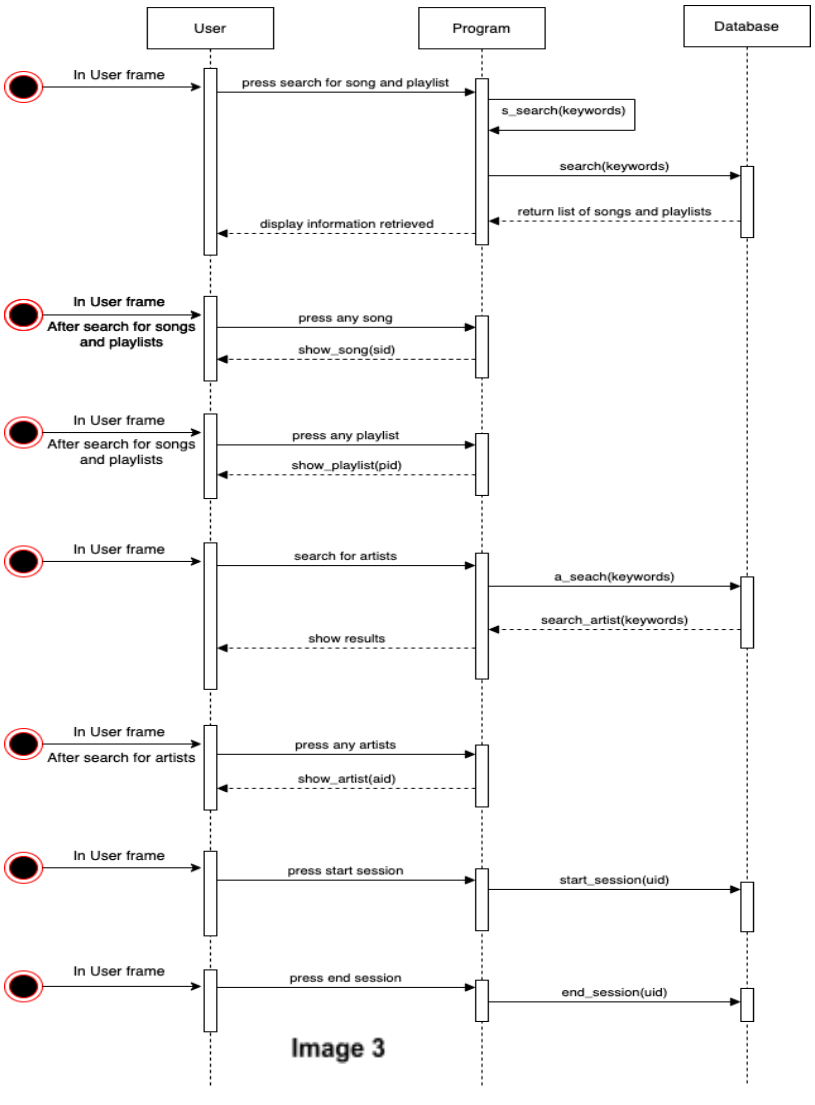


Image 3

ANNOTATIONS

- **Image 1**: Login and register
- **Image 2**: Choose to login as user or artist. Actions that artists can perform
- **Image 3**: Actions that users can perform
- **Image 4**: Information of a song selected and actions that can performed on that song
- **Image 5**: Information of a playlist or artist selected by a user

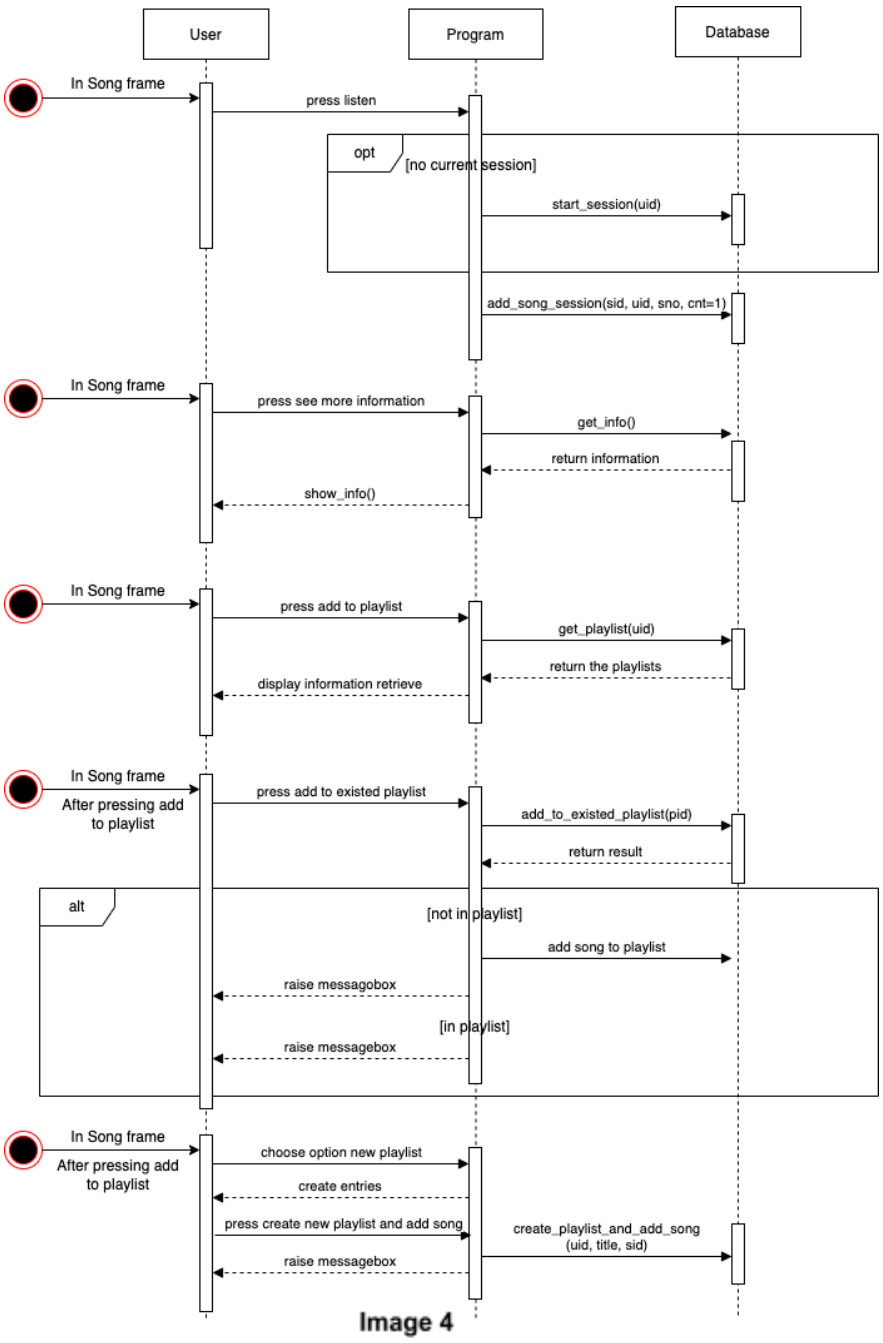


Image 4

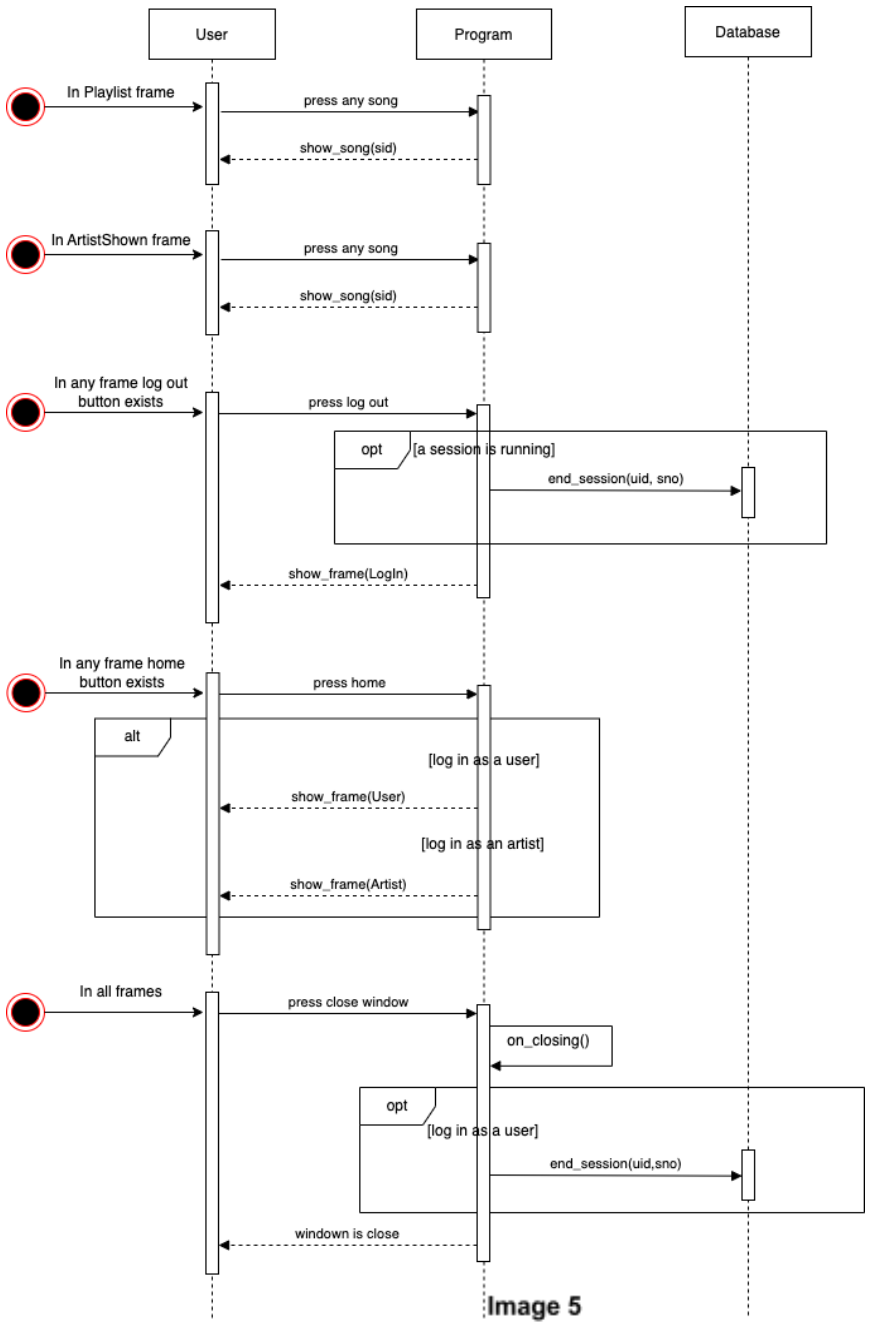


Image 5

C. Testing Strategy

The testing strategy used for the program follows a method called “Divide and conquer” that strictly follows the software’s design. As described in part B, the program is divided into smaller components, each responsible for some specific application functionality. This way, it is better to test and debug because as we progress through the creation process, we always ensure that each component works first before moving on to the next. One example is that before we got into the coding/querying part, a structured backbone was created with empty classes and their functions with details of what they do using block comments. Mainly we tested after every class and every function (which was not always the case), but the idea is to test/debug each program's functionality before moving to the next.

Due to the nature of the strategy, it is easier to narrow down where the bugs are using the process of elimination because we test part by part, so the bugs are usually small. Here are some test cases and examples of typical bugs we faced throughout the project:

1. All SQL queries were tested separately before being integrated into the host application. If a bug in a data-fetching function happens, it is difficult to tell if it comes from code or the SQL queries written. This method reduces the debugging time. Some example errors were violating the UNIQUE constraint while inserting values into plinclude and forgetting about the NULL value being returned by cursor.fetchall(). The testing process for these SQL queries all boiled down to perfecting the queries written and trying not to solve the errors using the host language.
2. For the coding part, the assert keyword and try-except blocks combined with print functions play a big role in catching errors. These are usually syntax errors, and logical errors only sometimes occur. Exceptions are caught easily with exception handling. Some examples would be missing “self” in a class function or using incorrect logical operators. The errors in the Python code happened the least often and were usually the most insignificant due to our break-down strategy as we coded and tested function by function. The more serious errors come from the SQL queries instead.
3. The use of Python Tkinter could be error-prone, despite helping a lot with envisioning the process. Therefore, it is important to check the Tkinter documentation to ensure that the correct methods and syntax are being used and the debugging process is the same as above. One example error is invalid library syntax being used because the library has been updated and is different from itself ten years ago.

4. We also strived to find some edge cases. For example, when searching for songs and playlists, what if a playlist did not have any songs, would it be returned in the result set? We resolved this problem by fixing our query for fetching the playlists that match the search keyword provided by replacing the WHERE clause with LEFT OUTER JOIN clauses in the FROM clause, putting the tuples with NULL values in their song columns into the result set.

D. Group Work Breakdown Structure

Here is a detailed overview of how the project work breakdown looks like:

- Firstly, Khac Nguyen created the program's backbone, including creating separate files for easier management, importing libraries, defining classes, functions and global variables, and finally initializing the connection with the database. Andy created a GitHub repository for the project (we chose GitHub as a collaboration method). Then, Andy would inspect the initial structure of the program, give some suggestions, and fix any problems.
- There was a meeting before getting started with the project after the backbone had been created. In the meet-up, we discussed further the program's structure to see if there was anything we could improve. We finally came up with a small tweak to the structure: there should be two files to contain data-fetching functions where SQL queries are written - one for users and one for artists. After, we also did a bit of the login part together, which helped a lot with us agreeing on certain uniform styles of naming and formatting our codes.
- Then, we proceeded to divide up the work for the project so that each person could work independently:
    + Andy: functions for artists, login screen and register screen.
    + Khac Nguyen: functions for users and perfecting the backbone.
    + The work was divided so that one person’s part does not interfere with the other’s to avoid conflict when pushing to GitHub.
    + The independent work for each member includes some Python coding but mostly SQL querying.
- Then we all did GUI.py together by arranging ahead when and where to meet, where we tested the functions we made earlier and the program and debugged everything (both in the Python and SQL codes), which took quite a long time.
- When we were done with the main components of the program, we put down a list of small action items and insignificant details/tweaks in the program that needed to be fixed/completed. The rest of the creation process was spent tackling these little details and testing some edge cases.
- Finally, we focused on writing the report together. Throughout the project, Andy made a Google Doc where both members could track their progress on the document and let the other know which part of the project has been completed. The Google Doc combined with GitHub is a great method of coordinating that kept both of us on track throughout the creation process. Also they also helped a lot with writing this report because we did not have to think back and try to remember each progress made in order to report.

PROJECT TIMELINE TABLE:

DATE	WORK COMPLETED	TIME SPENT	BY
18 October	Finished program's backbone	5 hours	Khac Nguyen
19 October	Added some functions into the file that contains SQL functions, one of which was the generate_id function	2 hours	Andy
21 October	Did a bit of the login part, agreed on uniform names and formats	2 hours	Andy and Khac Nguyen
22 October	Finished log-in page so that the GUI can be ready for more functions	1 hour	Andy
23 October	Finished “start and end session” part for the users.	1 hour	Khac Nguyen
24 October	Finished “add a new song” and “display top fans and top playlists” for artists	2 hours	Andy
24 October	Finished “register screen”	2 hours	Andy
25 October	Finished “search for songs/playlists” and “search for artists”	3 hours	Khac Nguyen
27 October	Finished “song actions”	2 hours	Khac Nguyen
28 October - 3 November	Debug, fix, report	15 hours (5-6 meetings)	Andy and Khac Nguyen