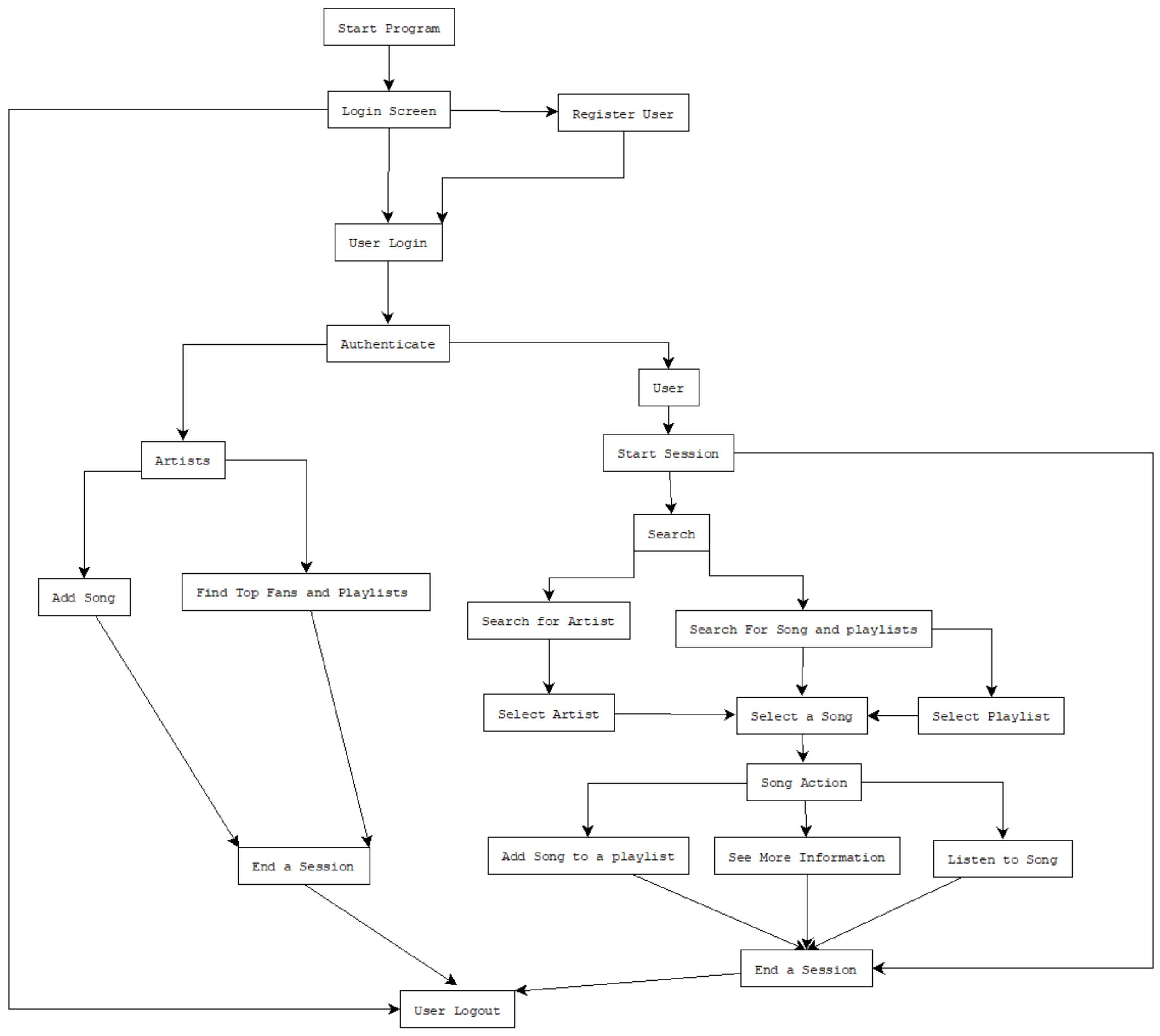
Design Document

Group Members: Josh Eastgaard, Brandon Fiogbe, Jefferson Fong

(a) General overview of your system with small user guide



The following project is designed to emulate a song streaming service. Users using the Streaming services are divided into 2 categories, a regular user or an artist. Each regular user can do a number of things such as starting a session, searching for songs/playlists and artists. Furthermore, they can select a song and listen to it and then lastly end the session. Each artist on the other hand can add a song and find the top fans who listen to their songs and the top playlists with their songs. When the program first initializes, the individual will be on a login screen where they will be prompted to login, afterwards, they can perform certain functions as a user or an artist. The following functions are listed in greater detail in Part b). Lastly, the user can log out and terminate the program.

(b) Detailed design of your software with a focus on the components required to deliver the major functions

- **Authentication**
    - **create_new_user() -** Prompts the user to create a new username, it will check if it is unique then as for a password. Password will be stored as a hashed value for security purposes. User will then be directed back to the main screen of the program.

    - **get_authentication() -** Prompts user for username and password, will check the *auth.db* database to authenticate input combination. If the user is authenticated then it will go over to the check_access() function to provide the correct interface to the user. If authentication fails then the user will be allowed to try again.

    - **load_all_names() -** This function is used to query all the uid's in the users and artists tables within the specified database.

- **Users**
    - **launch_home_screen(self, message = None) -** This function is the home screen for the users (there is one for users and one for artists). This will welcome the users and prompt them what they want to do. There are 3 valid options: '1', '2', and '3'. If the user doesn't make a valid option, they are prompted again to select another option.
        - 1: Search for songs/playlists: Executes the **search_songs()** function (read up on **search_songs()** for what it does).
        - 2: Search for artists: Executes the **search_artist()** function (read up on **search_artist()** for what it does).
        - 3: End session: Record the session end time and execute the **log_user_out()** function if a session was already in progress (read up on **log_user_out()** function for what it does) or return users back to the home screen if no session has started.

    - **search_songs(self) -** This function prompts the user for a unique keywords which then displays 5 matching songs or playlists that appear and is ordered based on the number of keywords that matches (if more than 5 songs/playlists appear, it will only show the top 5 matches at a time). All matching songs, title and duration are displayed and all matching playlists are displayed as well and all matching playlists display the id, title and duration of songs in the playlists.

    - **search_artist(self) -** This function prompts the user for a keyword which then displays all matching artist's name, nationality and number of songs and us ordered based on the number of keywords that are matching (like the search_songs() function, 5 matching artists are shown at a time).

- **log_user_out(self) -** Records and inserts a new sno with the uid, start time and end time into the existing sessions table and terminates the program.

- **song_action(self, user_selection)** - this function prompts the user to select a song from the given list using the sid. Afterwards, if the song sid exists, it prompts to select the following action ‘1’, ‘2’, ‘3’.
  - 1: Listen to it - a listening event is recorded within the current session
  - 2: See more information about the song - displays artist(s) who performed the song, song id, title, duration, names of playlists song is in
  - 3: add the song to an existing playlist or a new playlist
    - If playlist exists, add the song to the playlist with a given pid
    - If playlist doesn’t exist, create a new playlist with a new unique pid

- **Artists:**
  - **launch_home_screen(self, message = None) -** This function is the home screen for the artists (there is one for users and one for artists). This will welcome the users and prompt them what they want to do. There are 3 valid options: '1’, ‘2’, and ‘3’. If the user doesn't make a valid option, they are prompted.
    - 1: Add a new song: executes the **add_song()** function (read up on what the **add_song()** for what it does)
    - 2: Search for artists: executes the **top_3_stats()** function
    - 3: End session: **Executes the log_artist_out()** function

  - **top_3_stats(self) -** This function displays the top 3 playlists pids with the largest number of songs of that artist and the top 3 users with the highest listening time of the given artist.

  - **add_song(self) -** This function checks if the song is already in the table and then prompts the user for a title of the new song they would like to add followed by the duration of the song. This inserts the song into a song table and then inserts the artists into a perform table. It Prompts the user if there are any other artists that performed in the song and adds the aid of the artist who also performed the song.

(c) Testing strategy

We first established a database. Afterwards we used the database to test if our functions are working correctly. We also tested for invalid inputs users could do.
Testing strategies include prompting users for correct and incorrect username and password inputs, etc. Our test cases cover as many errors the user may encounter including but not limited to case sensitivity, invalid input, etc.

(d) Group Work Breakdown

*All members contributed equally over roughly 10 hours to the project.

- Josh Eastgaard
  - Created: **Authentication, launch_home_screen(self, message = None), anti-SQL injection, UserInterface**
  - Helped with integration of other classes.

- Brandon Fiogbe
  - Created: **search_songs(self), search_artist(self), song_action(self, user_selection), ArtistInterface**
  - Helped with connecting functions to main:

- Jefferson Fong
  - Created: **add_song(self), top_3_stats(self), design doc a) diagram, log_user_out(self)**
  -

- Report contributions
  - Josh - Overlooked the entire report, wrote up the authentication class documentation. Contributed comments to the design diagram.

- Communication
  - Discord, In-Person.
  - Met on Mondays and Fridays every week. 1 hour long meetings.
  - Occasional Pair programming