Jonathan Fjeld, Ying Wan, Crystal Zhang
CMPUT 291, Fall 2022

CMPUT 291 Mini-Project 1: Design

General Overview of System with Small User Guide

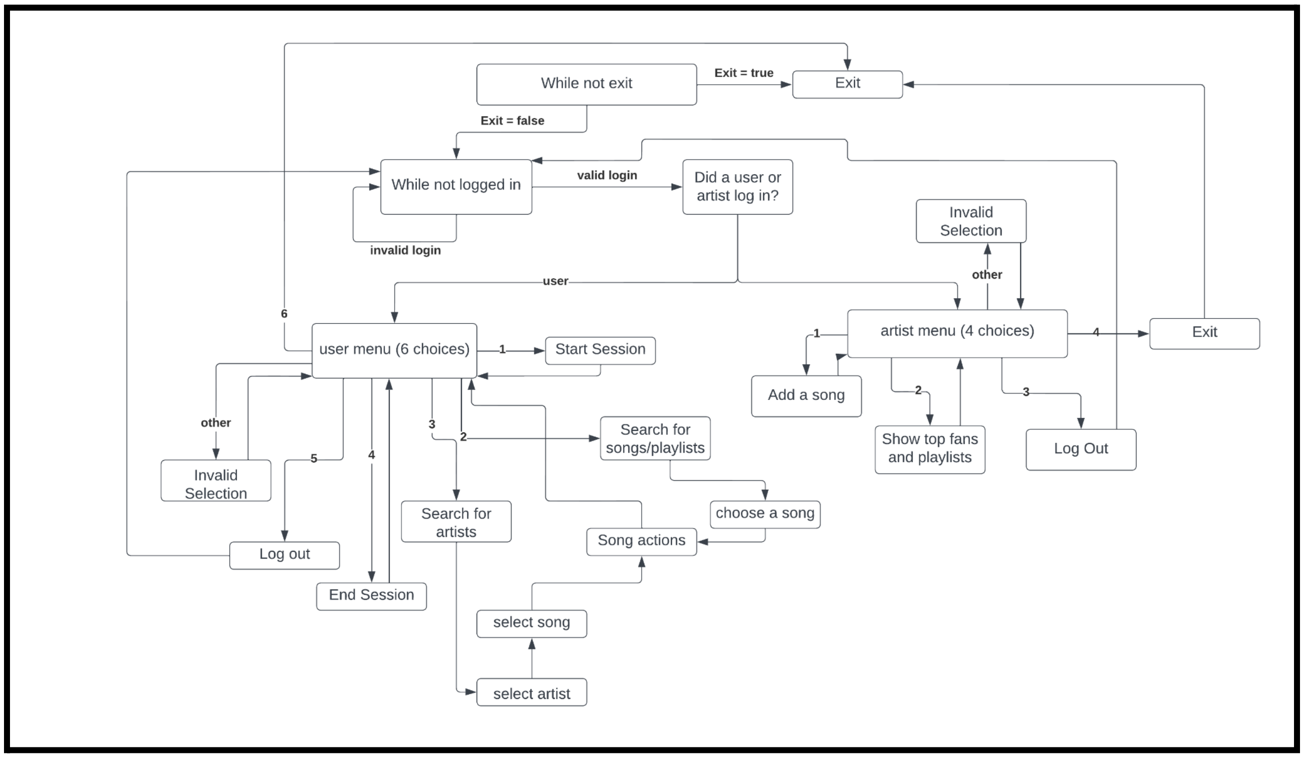

Figure 1: High-level flowchart of the possible choices a user can make in our program

To run our program from the command line, type python3 c291Mini1.py <database_path> into the command input. In order to run the program correctly, the user must run the above command from a directory containing the files c291Mini1.py, artistActions.py, searchArtist.py, searchPlaySong.py, and the database to be used. Upon starting the program, prompts will show up on screen, directing the user on what is to be input. Error handling is in place to ensure the user is not heavily punished for misinterpreting the input prompts, or typing an incorrect command.

To ensure a user is always logged in, and the program is closed when the user wants to close it, a double-nested while loop is used.
For any search functions, users may search for multiple keywords at once by separating them with the semicolon (;) character. For example, if a user is searching for songs and playlists, and they input ‘apple;banana’, all songs and playlists that include the terms apple, or banana will be displayed. Inputs that contain only whitespace are not considered valid and the user will be prompted back to the menu.

Other than passwords, which are case-sensitive, all user inputs are case-insensitive. Searching for a song called “SONG” or “SonG” will have the same result. When a playlist is created, its name in the playlists table will be case-sensitive, but when a user adds songs to the playlist, the playlist can be referred to by case-insensitive keywords. If a playlist is created called “SonGs”, it will show up in the playlists table as “SonGs”, but will still have songs added to a playlist if the input is “SONGS” or “songs”.

Detailed Software Design with Focus on Main Functional Components

Jonathan Fjeld, Ying Wan, Crystal Zhang
CMPUT 291, Fall 2022

In the "main" file, c291Mini1.py, the code runs starting from the `if __name__=="__main__":` statement near the bottom. Main loop runs here, and users can login or exit the program.

c291Mini1.py

setup(dbName): Connects the code to the database specified in the command line. Returns a cursor and connection for sqlite3 in python.
getID(): Loops until the user or artist enters a valid ID.
newUser(id, pwd, c, conn): Creates a new user if the input user id and password does not match that of a current user or artist. Gets the name for a user and inserts it into the users table.
login(id, pwd, c, conn): Checks if the given ID and password correspond to a new user, a user, an artist, or both. Logs the user in based on matching values.
startSession(c, conn, id, snoNext): Creates a session with a unique session number (snoNext), the current date as the start date, and NULL as the end date for the current user. Will only create a session if there is not a current session in progress.
endSession(c, conn, id): Ends the current session by setting a NULL end date to the current date. If there is no session currently in progress, print an error message, and do nothing.
userMenu(id, c, conn): This function is called continuously in the inner while loop which ensures a user stays logged in, and an outer loop to ensure that the program runs until a user exits. If a user is logged in, they can either logout to exit the inner while loop, or exit to logout and exit the outer while loop. If a user is not logged in, they can login to enter the inner while loop, or exit the program.
They are given the options to:

1. Start a session (Calls startSession())
2. Search for songs and playlists (Calls searchPlaySong.search())
3. Search for artists (Calls searchArtist.search())
4. End the current session (Calls endSession())
5. Logout (Exits the inner while loop, does not exit the outer while loop)
6. Exit the program (Exits the outer while loop)

- All other inputs will print an error message, and repeat the prompt.

artistMenu(id, c, conn): This function is called continuously in the inner while loop (explained in part a) if the user is logged in as an artist. They are given the options to:

1. Add a song (Calls artistActions.addSong())
2. Find their top fans and playlists (Calls artistActions.findTop())
3. Logout (Exits the inner while loop, does not exit the outer while loop)
4. Exit the program (Exits the outer while loop)

- All other inputs will print an error message, and repeat the prompt.

searchPlaysong.py

search(connection, cursor, id): Collects keywords from users and searches through 'songs' and 'playlists' for titles matching the keywords. Results are ordered based on the combined occurrences of keywords. For example, if the user wants to see all titles with 'a' and 'e' in them, and the titles are 'aaaee' and 'abcde', the query will count 5 occurrences in the former and 2 occurrences in the latter. The output would then be organized with 'aaaee' first, and 'abcde' second. A message will be shown if there are no titles with that match with the keywords, and the user will be prompted back to the user menu.

get_five(name_list, rows, query, cursor, connection, id): After the queries are executed in search(), the output is printed into the terminal in get_five(). It will first output a maximum of 5 matching results and prompt the user to either go to the next page, the previous page, select a number from the output or exit to the user menu. Since it's possible for there to be the same ID for a playlist and a song, we decided to add an order column as these will be unique, and user-friendly. If the user selects a playlist, the code will print out the necessary information. If the user selects a song, the code will then go into the songOptions() function. This function will display the song actions the user can take:

1. Listen to a song (Calls listen_song())
2. Get more song information(Calls song_info())
3. Add song to a playlist (Calls addPlaylist())
4. Exit (Returns to user menu)

listen_song(song_id, cursor, connection, uid): Checks if a session is already in progress. If not, a session will be started and the song will be inserted into the 'listen' table. If a session has started, another check will be performed to see if the user has listened to the song before this session. Depending on these factors, it will either update the count in 'listen' or insert a new row into 'listen'. A message will then be printed to inform the user that they're listening to a song.
song_info(order, query, cursor, connection, song_id): Queries are performed in order to acquire necessary information, results are then printed out into the terminal.
addPlaylist(song_id, cursor, connection, uid): If the user chooses to add the song to a playlist, the playlist title will be prompted from the user. After that, the code will check if the playlist already exists. If it does, the code will acquire the session number from the existing playlist in preparation for inserting a new entry into plinclude. If not, a new playlist entry will be inserted by taking the current maximum pid from playlists and increasing that pid by 1. This will ensure that it is unique, and follows an organized pattern. Either way, the song will be inserted into the plinclude table to complete the correct process of adding a song to a playlist.

searchArtist.py
search(connection, cursor, id): Searches through the 'songs' and 'artists' tables for song titles and artist names that contain the user inputted keywords. Results are then ordered based on the combined occurrences of keywords. Any inputted keywords that contain only spaces are not considered to be valid and the user will be prompted back to the user menu.
get_five(name_list, rows, cursor, connection, uid): Outputs a maximum of 5 matching results (using the print_five method) from the search method and prompts the user to either go to the next page, the previous page, or select an artist from the output (which will then go into the printSongInfo() function).
printSongInfo(cursor, connection, uid, ip, rows): Prints information for all songs performed by the selected artist. Also prompts users to either exit back to the main menu or select a song from the output.
If the user selects a song, the code will then go into the songActions() function. This function will display the song actions the user can take:

1. Listen to a song (calls searchPlaySong.listen_song())
2. Get more song information(calls song_info())
3. Add song to a playlist (calls addPlaylist())

song_info(cursor, song_id): Displays information about the selected song.
addPlaylist(song_id, cursor, connection, uid): Adds a song to a playlist using user inputted title. If the inputted title is the same as a pre-existing playlist, the program will acquire the session number from the existing playlist to

insert a new entry into 'plinclude'. If the playlist does not yet exist, a new playlist will be made with a unique pid. The song will then be inserted into the 'plinclude' table.

artistActions.py

addSong(cursor, aid, conn): Adds a song using artist inputted title and duration and links the song to the artist(s) who performed it. If the song has the same title and duration as another song by the same artist, the program will print a warning. The warning will notify the artist that they may be entering a duplicate song and will allow the artist to accept or reject the warning. The method also allows for an artist to input other artists that they performed with using their aid. If the other artist(s) are not in the 'artists' table, they will not be added to the 'perform' table and a note will be printed.

findTop(cursor, aid): Prints the top 3 user's uid and total duration listened and the top 3 playlists's pid, title, and number of artist's songs included. The top 3 users are determined by total duration listened and the top 3 playlists are determined by number of artist's songs. If there are less than 3 results in either the top users or the top playlists, the method will only print however many results there are.

Testing Strategy

We tested with Aron Gu's database which was provided during assignment 2. Each time the user was prompted to give input, the code was tested by providing numerous different types of expected input, as well as numerous types of unexpected input. This included negative numbers, inputs of incorrect type, blank inputs, and inputs of which the query would give null results. After every update and insert statement, separate SQL statements were run outside of the program to display the table results in order to verify that the tables had changed. During testing, many ValueErrors, and TypeErrors were thrown as some inputs were of incorrect type or would result in no data. Logical errors like failing to consider case insensitivity also occurred numerous times as we compared outputs that had mixed case versus outputs that didn't.

Group Work Break-Down

We communicated with each other in person and through Discord messaging. Group division of work was decided after an estimate of the difficulty and workload of each task. We also kept the files on a GitHub repository, where we could share our code with each other. We mainly coded our work in separate files for better visibility of code, organization and to prevent any overwriting conflicts.

Jonathan (~18 hours**)**: Completed the login screen as well as set up empty spaces in the user menu (6 options) and artist menu (4 options) for Crystal and Ying to implement. Created almost all code in the file c291mini.py. Also thoroughly tested as many possibilities for user input as possible in the completed program for error handling purposes, and made attempts to fix errors or let Crystal and Ying know what errors were found.

Crystal (~16 hours**)**: Completed and implemented the necessary queries relating to searching songs and playlists based on keywords. This also included implementing the Song Actions task. Troubleshooted code to make sure it covered the general use cases. The code for the above tasks are all carried out in searchPlaySong.py.

Ying (~13 hours**)**: Adapted and implemented the necessary queries relating to searching artists and based on keywords. Also completed and implemented artist actions for the artist menu. Troubleshooted code to make sure it covered the general use cases. The code for the above tasks are all carried out in searchArtist.py and artistActions.py.