**Helping users find**

**song suggestions &**

**information about**

**their favourite music**

**Computer Science NEA**

**Alexander Petch**

**30 April 2023**

Contents

[Analysis 4](#_Toc132915415)

[Project Intentions 4](#_Toc132915416)

[Problem Identification 4](#_Toc132915417)

[Reasons for Change 4](#_Toc132915418)

[Understanding the Problem – Data Collection and Analysis 5](#_Toc132915419)

[Prospective Users 9](#_Toc132915420)

[Potential High-level Solutions 10](#_Toc132915421)

[Project Objectives 11](#_Toc132915422)

[Chosen processes, data and solutions 12](#_Toc132915423)

[Design 16](#_Toc132915424)

[High level overview 16](#_Toc132915425)

[UI 16](#_Toc132915426)

[Gathering music data from Spotify and Wikipedia 21](#_Toc132915427)

[User Database and Handling 23](#_Toc132915428)

[Recommendation Algorithm 24](#_Toc132915429)

[Main Function 29](#_Toc132915430)

[Internal and external data storage 33](#_Toc132915431)

[Technical solution 35](#_Toc132915432)

[UI.py 35](#_Toc132915433)

[DataGathering.py 36](#_Toc132915434)

[Main.py 43](#_Toc132915435)

[Login.py 54](#_Toc132915436)

[Hashing.py 56](#_Toc132915437)

[DataSetCreation3.py 57](#_Toc132915438)

[UI State 0 – Login Screen 59](#_Toc132915439)

[UI State 1 – Song Search 59](#_Toc132915440)

[UI State 2 – User Settings 60](#_Toc132915441)

[UI State 3 – Album Info 60](#_Toc132915442)

[UI State 4 – Artist Info 61](#_Toc132915443)

[UI State 5 -Song Info 61](#_Toc132915444)

[UI State 6 – Find Similar Song 62](#_Toc132915445)

[UI State 7 – Similar Song Results 62](#_Toc132915446)

[UI State 8 – No Internet 63](#_Toc132915447)

[Testing 63](#_Toc132915448)

[Introduction 63](#_Toc132915449)

[Test Plan 63](#_Toc132915450)

[Test Results 68](#_Toc132915451)

[Evaluation 69](#_Toc132915452)

[Outcome vs. objectives 69](#_Toc132915453)

[Feedback 71](#_Toc132915454)

[Discussion 72](#_Toc132915455)

[Possible Improvements 72](#_Toc132915456)

## Analysis

### Project Intentions

My project aims to provide users with an intuitive platform to find out new information about their favourite music artists, as well as allowing the users to access an alternative to their preferred music streaming platform to get new music recommendations based on a song, artist or album they input.

### Problem Identification

Research has demonstrated that music offers several psychological and physiological advantages. It has been utilised as a therapeutic treatment to improve cognitive function, lessen stress and anxiety, and increase physical performance. A key part of appreciating music is being able to listen to new music, which not only provides novelty, but also helps develop and expand musical taste. Lack of access to new music can have a significant impact on someone in many ways, including how it affects their ability to find new music, their mood, their immune system, and their general well-being.

The most common way young people today find new music is based on Spotify recommendations of similar artists. The second most common way is recommendations from friends, however many of these recommendations have also come from Spotify’s recommendations. In summary, Spotify is the most significant way most young people are exposed to new music.

Spotify's current recommendation algorithm falls short in introducing consumers to fresh and varied music. The lack of variety and less satisfying listening experience are caused by the fact that many users frequently receive recommendations for the same albums and artists that they have already listened to. This issue is significant because it restricts users' capacity to discover new music, denies them the chance to broaden their musical preferences, and eventually causes user annoyance and a drop in platform happiness.

### Reasons for Change

The Spotify algorithm creates recommendations for each individual user based on information about the songs, albums, and artists that users have previously listened to as well as information about the listening preferences and habits of similar users. According to the algorithm, if a person has previously listened to an artist or album and other users with similar listening habits and tastes have done the same, the user is likely to appreciate it and want to hear it again. If a lot of listeners of a particular song, Song A, also listen to Song B, then anyone who listens to Song A or B will be recommended Song B or A – they become linked. This approach is known as collaborative filtering.

Collaborative filtering is a reasonable approach, but it suffers from the problem of ‘echo chambers’. As soon as Spotify starts to recommend Song B to people who like Song A, then lots of people who like Song A will start to listen to Song B, reinforcing the link within Spotify’s algorithm. It is a positive feedback system. Soon the links will end up focussing on a small number of very strongly linked songs, and it will be very hard for a new song to be recommended as it will always be way down the list of recommendations.

The current Spotify recommendation algorithm, which frequently repeats the same choices and prevents users from discovering new artists and genres, falls short of offering customers a wide variety of music recommendations. This not only makes it more difficult for users to find new music, but also makes listening less enjoyable overall, which annoys users and lowers their satisfaction with the site.

In conclusion, Spotify's subpar recommendation system is a serious issue that restricts the accessibility of new music and makes it difficult for users to diversify their musical tastes. This ultimately causes user annoyance and a drop in platform satisfaction.

Artists are one group that is most impacted by the issue of Spotify's poor music recommendations. Users are less likely to find and listen to new artists when they are not being recommended a wide variety of music. This reduces the potential audience for those artists and makes it harder for them to become known and develop a following. This is a concern because it makes it difficult for up-and-coming musicians to develop popularity and establish themselves in the music business, which ultimately reduces the variety and range of music that is accessible to platform users.

The fact that the Spotify algorithm is built on collaborative filtering may be one explanation for why it regularly suggests the same music and songs that the user has already listened to. Many recommendation systems, like Spotify, use the collaborative filtering technique to create recommendations based on the tastes and actions of similar users.

In the case of Spotify, the algorithm creates recommendations for each individual user based on information about the songs, albums, and artists that users have previously listened to as well as information about the listening preferences and habits of similar users. According to the algorithm, if a person has previously listened to an artist or album and other users with similar listening habits and tastes have done the same, the user is likely to appreciate it and want to hear it again.

### Understanding the Problem – Data Collection and Analysis

To prove that other users of Spotify share the same issue as me, I created a questionnaire and asked my peers to answer five questions.

#### 1. How much do people listen to music

Chart, pie chart

Description automatically generated

*Figure 1. Chart showing the proportion of people consider they listen to a lot of music*

The results here are quite clear, showing an overwhelming majority of people saying that they listen to a lot of music. Most people considering music to be a large part of their life would make finding new music an important issue for many people.

However the question asked was fairly vague, someone may have listened to 100,000 minutes of music in a year, and someone else 20,000, but they both may put that they listen to a lot of music. Not specifying a numerical value means that the data is less accurate as “a lot of music” is very subjective.

#### Question 2. How difficult is it to find new music

Chart, bar chart, histogram

Description automatically generated

*Figure 2. Distribution showing how difficult people find it to find new music*

On a scale from 1 to 10, 12 participants responded to the question, "How difficult is finding new music to listen to," with a mean score of 5.583. This suggests that finding new music was often a challenge for participants. It is surprising that Spotify's recommendation system is not more effective in recommending new music to customers given that the company is massive and has access to cutting-edge technology.

However, it is important to keep in mind that the subjective nature of the rating scale and the 10-point scale's lack of precision could lead to bias in the survey's results. Furthermore, the results may not be generalizable to the full population due to the small sample size of only 12 participants, which may not be indicative of Spotify's total user base. When analysing the findings and evaluating the merits of the recommendation algorithm, these constraints should be taken into account.

#### Question 3. How do you find new music?

*Figure 3. Different ways people use to find new music*

Participants were asked how they find out about new music. The results show that for nearly all of those asked (11 out of 12), used Spotify. It also shows that Spotify is much more commonly used than any other music app (11 participants using Spotify vs. 2 using any other music app), and that it is used far more than any other source. This gives a strong indication of the power that Spotify has in influencing the new music that people listen to.

However, it is important to keep in mind the results may not be generalizable to the full population due to the small sample size of only 12 participants, which may not be indicative of the general population.

#### Question 4. How good are music apps at recommending new music?

Chart, bar chart, histogram

Description automatically generated

*Figure 4. Distribution showing how well people think their preferred music apps are at recommending new music*

In response to the question "How good would you say your preferred music app's ability to recommend new music to you is?" with a 10-point scale where 1 is extremely bad and 10 is amazing, 12 participants provided responses with a mean score of 4.6. This indicates that, on average, participants rated their preferred music app as somewhat poor at recommending new music.

The relatively wide range of replies indicates that participants have a range of thoughts on the ability of their preferred music app to make new music recommendations. While some participants have given their app good ratings, others have given it low ratings; this variation in opinions may be brought on by personal preferences, listening habits, and app experiences.

Furthermore, the mean score of 4.6 is lower than the midpoint of the scale, indicating that a significant number of participants rated their app poorly. This suggests that many users are not satisfied with the recommendations provided by their preferred music app and that there may be a need for improvements in the recommendation algorithm.

In conclusion, according to the survey's findings, participants generally thought their favourite music app did a mediocre job of introducing them to new music. This indicates that there are a variety of user opinions and that a sizable portion of users are dissatisfied with the recommendations made by their app. Even accounting for drawbacks of the questionnaire, the effectiveness of the recommendation algorithm and the concerns that users indicated should be taken into account.

#### Question 5. Have do you have positive experiences from music?

Chart, pie chart

Description automatically generated

*Figure 5. Graph shown the proportion of people who have positive experiences from music*

Graphical user interface, text, application

Description automatically generated

A picture containing graphical user interface

Description automatically generatedBackground pattern

Description automatically generated

*Figure 6. Examples of people’s positive experiences from music*

In response to the question "Have you experienced positive impacts of listening to music?", 83.3% of participants answered affirmatively, providing reasons such as "It helps me regulate my mood and cheers me up," "Relaxing, feel better on a bad day," "More motivation and concentration," and "Form of Therapy & Relaxation". This indicates that a significant majority of participants have experienced positive impacts from listening to music, such as improved mood, relaxation, and increased motivation.

These responses demonstrate how music is significant to a wide range of people and how it significantly affects their wellbeing. However, if a music recommendation algorithm is ineffective at introducing users to fresh and varied music, it may prevent them from benefiting from the benefits of music listening. For instance, users might not find new musicians or genres that could have a positive effect on their motivation or mood if a wide variety of music is not recommended to them.

In conclusion, this survey's findings show that a sizable majority of respondents have benefited from listening to music. This emphasises the significance of music in people's lives and implies that poor music recommendations may be detrimental to users' wellbeing.

#### Questionnaire Conclusions

In conclusion, the four questionnaire questions' results show that participants had trouble discovering new music, thought their favourite music app did a poor job of recommending it, and that listening to music had positive effects on their lives. However, there are problems with the subjective nature of the rating scales and the small sample sizes of the questionnaire.

These findings imply that methods and tools for finding new music need to be improved, and a more effective music recommendation algorithm, or just an alternative is also required. Furthermore, when analysing the findings and evaluating the efficiency of the recommendation algorithm, it is important to take into account the questionnaire's constraints, such as the subjective rating scales and small sample sizes.

The answers to these survey questions demonstrate the significance of music in people's lives and the shortcomings of the current music recommendation algorithms.

#### Questionnaire limitations

The users I received feedback from will be my classmates, so 17-18 year old students. This comes with advantages and disadvantages. One advantage that comes from having a young demographic for testing my project is that they will be more likely to be using music streaming services and more up to date technology, allowing them to access a greater amount of music. This can provide valuable insights into the preferences and behaviours of young users, which can inform the design and development of the recommendation algorithm. Additionally, because me and my users are in regular contact, we can facilitate direct communication and collaboration, allowing for a more efficient and effective feedback loop.

The drawback of having such a limited age demographic is that it might not be representative of all music streaming platform users. Users who are older may have different preferences and listening styles, and users from various geographic or cultural backgrounds may have different needs and expectations. As a result, the results may be less generalizable and the recommendation algorithm may be less successful. It might be necessary to gather feedback from a wider range of users in order to address this problem, as well as to take into account the unique requirements and preferences of various age groups, geographical locations, and cultural groups when designing and developing the algorithm.

In conclusion, having classmates between the ages of 17 and 18 as potential users on my music recommendation project has several benefits, including the possibility for open dialogue and collaboration as well as their familiarity with music and technology. The results may not be as generalizable as they could be because they might not be a representative sample of all users. It will be crucial to take into account the unique requirements and preferences of a variety of users in order to maximise the relevance and effectiveness of the algorithm.

### Prospective Users

As my program will both recommend music and also display information about artists, my program’s main users will just be people interested. These users will find it valuable to find new music outside of just Spotify, as they will be able to set personal preferences about what music they would like to be recommended e.g., how acoustic/energetic it is.

In addition, users who are less interested in music may enjoy it more after using my program; which, like mentioned above, may have various positive effects on a persons’ mental health. However they may just simply be curious and want to see some information about their favourite artists. Also just because a person has less interest in music, an app that can recommend new music to a person may still have some use, just less frequently.

As shown by my questionnaire, a large proportion of people consider themselves having an interest in music meaning my programs potential userbase is very broad.

### Potential High-level Solutions

The table below shows the high-level solutions I considered, together with their positives and negatives.

| **Solution** | **Positives** | **Negatives** |
| --- | --- | --- |
| Website | More easily accessible to users assuming I have put the site on the internet and secured a domain and server.  Allows for more freedom in terms of graphical interface with many built in functions and easily manipulatable objects.  Support for mobile users assuming I have added support for it.  Requires no downloading from the user because they are accessing information from a server running from me or the cloud | Requires HTML CSS and JS knowledge to make my website up to date and effective.  I have limited knowledge of HTML and CSS meaning I would have to put in much more work to create a complex product.  Limited knowledge of the Spotify Web API  Would require internet access to use |
| Python Application | Have much more experience with creating python application as well as multiple projects using PyGame.  The lack of in built GUI objects makes my UI incredibly easy to edit and change to my original vision  Python being a high level language makes my code much easier to understand.  The entire program can mostly be built in one language rather than 3 or 4  Certain features such as the song recommendation could be used without an internet connection | Lack of GUI objects means I have to program them all from scratch  Requires downloading making it harder to access for casual users  Lack of animations or a more fluid GUI experience |

However overall, I decided on using python for my project. I started with trying to create a website and many problems became apparent to me; that is not to say that I have not encountered problem with using python, but my progress is much faster and I believe I will end up with a more polished and complex product.

### Project Objectives

To deliver an application that allows users to find information about music, to make suggestions based on songs, and to store their searches, the program will need to meet the following objectives.

1. New users must be able to register their username and password
   1. New users must be able to select a username and password
   2. The username and/or password cannot be blank
   3. The username and password must both be a minimum of four characters
   4. The username and/or password cannot include special characters (i.e. only letters and numbers are allowed)
   5. The username must be unique
   6. If a username or password is invalid, the user must be informed
   7. The password must be stored in an encrypted way
2. Users must be able to log in
   1. Users must be able to log in based on a username and password
   2. If a username cannot be found, the user must be informed
3. Users must be able to change their password
   1. Users must be able to change their password
   2. Users must only be able to change their password if they know their old password
   3. New passwords must not be blank
   4. New passwords must be at least four characters
   5. New passwords must not contain any special characters (i.e. only letters and numbers are allowed)
   6. If a new password is invalid, the user must be informed
4. Users must be able to search for information about a song/artist/album
   1. The user must be able to select if they would like to search for either a song, artist, or album
   2. The name of the song, artist, or album cannot be blank
   3. If the name of the song, artist, or album cannot be found, the user must be informed
   4. The user must be able to see the information retrieved by the programme
   5. The user must be able to continue to look at the information unitl they are ready to move on
5. Users must be able to get suggestions for other songs they might like, based on a song they do like
   1. The user must be able to input the name of a song
   2. The name of the song cannot be blank
   3. If the name of the song cannot be found, the user must be informed
   4. The user must be able to see four song suggestions, based on the name of a song that they input
   5. The user must be able to continue to look at the suggested songs until they are ready to move on
6. Users must be able to review their search history
   1. Users must be able to review their previous searches (whether for information about songs/artists/albums, or suggestions for similar songs)
   2. Users must be able to clear their search history
7. General
   1. Users must be able to quit the programme
   2. Users must be informed if the programme cannot establish connection with the internet

### Chosen processes, data and solutions

#### User Interface (UI)

For my UI I had 2 main choices to choose from to display information to my users. PyGame and TKinter; in the end I chose PyGame. Although my program is not a video game, PyGame has many advantages over TKinter.

| **Library** | **Advantages** | **Disadvantages** |
| --- | --- | --- |
| Pygame | Designed specifically for creating games and interactive applications | May be more difficult to learn and use compared to Tkinter. |
|  | Generally faster and more efficient than Tkinter, especially for rendering graphics | Not included with the standard Python distribution, so it must be installed separately |
|  | Has a wider range of features and functionality, including support for sprite rendering |  |
|  | Has a larger and more active community of users and developers |  |
| Tkinter | Included with the standard Python distribution, so it does not need to be installed separately | Less efficient and slower than Pygame for rendering graphics. |
|  | Generally easier to learn and use compared to Pygame | Has a more limited range of features and functionality compared to Pygame |
|  | Well-suited for creating simple GUI applications |  |

Separate installation is not a problem for me or my users as the completed program can be turned into an executable without needing separate libraries or the need to add python to path/pip install every third party library. Although Tkinter may be slightly simpler and easier to use, I am already familiar with the syntax of PyGame and enjoy the better flexibility of the library. In my personal opinion, Tkinter looks very dated, while PyGame allows you to customise your display much more. However PyGame does not have in built structures like buttons, menus and input boxes meaning I have to code them myself.

In conclusion, I chose PyGame for its better efficiency, higher customisability and its more modern look.

#### Database

A database is vital for storing sensitive user data outside the program, in my database I will be storing information such as usernames and passwords.

For my user’s data, I decided to use an SQL database.

| **Advantages** | **Disadvantages** |
| --- | --- |
| SQL Queries can be used to retrieve large amounts of records from a database quickly and efficiently | Not well-suited for handling unstructured data, such as images, videos, and audio files |
| Uniform language that is easy to learn and use, even with a limited knowledge of SQL |  |
| Easy to use and integrate with python |  |

My project will not be using unstructured data, so SQL is an overall good choice.

#### Hashing

Hashing is the practice of taking a string or input key and representing it with a hash value, which is typically determined by an algorithm and constitutes a much shorter string than the original. The hash value is used to index the original value or key and make it easier to find or employ the original string.

In my project I must use hashing for obscuring password data to make sure that users passwords are safe and secure, so they cannot be accessed by anyone opening my database. The hashing algorithm I chose to use for my project was SHA 256. A widely used and efficient hashing algorithm.

The input for hashing would be a users’ password e.g., “password123,” the hash code outputted would be “ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f,” a completely unintelligible output that is unable to be translated back into the original string. The user would input their password, the password would then be hashed and compared to the hash value that is already in the database. This means that the users’ password would not have to be in the database preventing security issues.

Hashing is easy to integrate into python with the library Hash lib.

#### Object Oriented Programming

The majority of the my program will be based off objects I create and reuse throughout my program, most of the procedural code in my program will be in my main file, the backbone of the program will be based on object oriented programming.

Object-oriented programming is a programming paradigm that is based on the concept of "objects," which can contain data and code. One of the main advantages of object-oriented programming is that it allows for code reusability. This means that I can reuse code that has already been written, which can save time and reduce the amount of code that needs to be written. Another advantage of object-oriented programming is that it allows for better code organization. By breaking code down into smaller, more manageable pieces, I can more easily understand and maintain their code.

When breaking down my large program into smaller problems, it can be extremely useful to essentially solve each problem by creating an object that I can reuse, especially for my GUI.

#### Gathering data on songs

In order to successfully recommend a new song to a user, my project will need to collect data from many thousands of songs. It will also need a way to evaluate how similar these songs are to each other in a way that is likely to give good recommendations.

Even through Spotify’s algorithm is based on collaborative filtering, Spotify also includes an evaluation of each song along ten different song attributes. These include:

1. Acousticness: A measure of how acoustic the track is, on a scale from 0.0 to 1.0. The higher the value, the more acoustic the track is.
2. Danceability: A measure of how danceable the track is, on a scale from 0.0 to 1.0. The higher the value, the more danceable the track is.
3. Energy: A measure of the intensity and activity of the track, on a scale from 0.0 to 1.0. The higher the value, the more energetic the track is.
4. Instrumentalness: A measure of the presence of vocals in the track, on a scale from 0.0 to 1.0. A value of 0.0 indicates that the track has vocals, while a value of 1.0 indicates that the track is instrumental.
5. Loudness: The overall loudness of the track in decibels (dB), ranging from -60.0 to 0.0. The higher the value, the louder the track is.
6. Speechiness: A measure of the presence of spoken words in the track, on a scale from 0.0 to 1.0. The higher the value, the more spoken words there are in the track.
7. Tempo: The tempo of the track in beats per minute (BPM).
8. Valence: A measure of the positivity of the track, on a scale from 0.0 to 1.0. The higher the value, the more positive the track is.
9. Key: the key of a song
10. Liveness: judging if a song is live or not from 0 to 1

For my project, I decided to base song recommendations based on these ten attributes – if one song is very similar to another, in terms of these attributes, then a user who likes one is likely to like another. This is not a perfect way of making recommendations, for example it does not include song genre, however my main aim is to come up with a different set of recommendations to that generated by Spotify, and this approach is likely to achieve this while still making recommendations that many users will enjoy. It also allows me to gather structured data on many thousands of songs, which otherwise would be a hugely time-consuming activity.

The main way of gathering song attribute data is by using Python to access Spotify's database. By accessing different types of data available on Spotify, including track, artist, and album information, Spotipy, a Python library, makes this data collection process easier. I created a free Spotify developer account in order to use this library, and I was given a client ID and password to authenticate my access to the platform's data.

Spotify is a great tool for gathering information because it has a sizable music library that largely contains the music that users may want to access. I can also get information that might not be easily accessible through the Spotify app by using the Spotipy library. This makes it possible for me to compile a larger set of information to help users with music recommendations. Ultimately, more accurate music recommendation functionality is made possible by using Spotify as a data source and Spotipy to gather that data.

#### Data gathered from Wikipedia

Python libraries like requests and Beautiful Soup were used to retrieve data from Wikipedia. While the Beautiful Soup library parsed the HTML code to provide a more structured and user-friendly data format, the requests library allowed for the retrieval of HTML from a given URL. Wikipedia is a useful tool for compiling information on a variety of topics because it contains a vast amount of information. Information on even relatively obscure artists is included in this. It is important to keep in mind, however, that despite Wikipedia's impressive depth of knowledge, the data's accuracy is not always ensured as it depends on the quality level and dependability of the sources used to produce the content.

## Design

### High level overview

I will split my programme into the following five parts:

1. User Interface – Displays objects and data the user will interact with, or simply just display information
2. Gathering music data from Spotify and Wikipedia – Includes the objects that store the information I will access, manipulate and display about music that the user queries
3. User database and handling – The database of users and how in my program they are interacted within my program
4. Song comparison hashing and dataset population– How my program finds similar songs to recommend, and my song dataset which holds the hashes of the songs I will recommend
5. Main Function – Uses all my other files e.g., UI objects, and music data I gather; while running this all a main loop

These parts are described in more detail in the subsections below.

### UI

#### Windows and navigation

My UI will include different windows depending on what the user wants to see. Each window has a set amount of objects per window. For example, the song info page holds bar charts, images and text that will only be displayed once the user asks for song information.

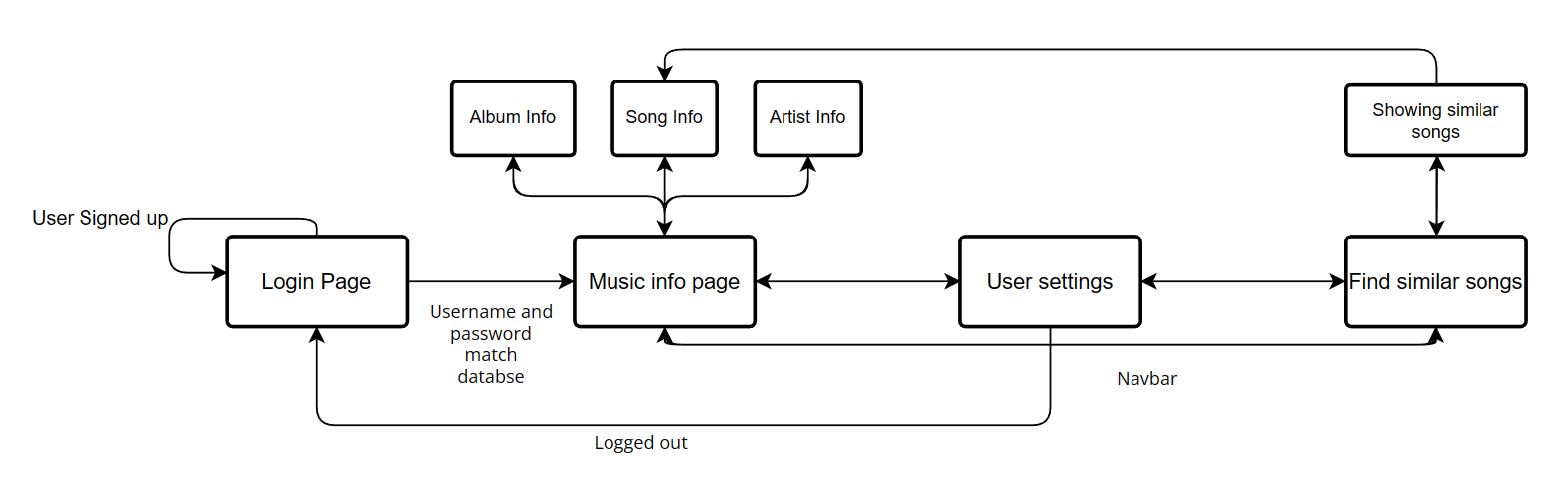
#### UI Windows

* Registration and login. This will allow the user to add themselves into my users database, and validate their credentials while logging in
* Information query. This will allow the user to type in a song/artist/album title and start the search.
* Display song information – shows all the information about the song the user queried, including audio features
* Display artist information – shows information about the artist the user queries, and accesses information from Wikipedia
* Display album information – shows information about the album, including mean audio features
* User settings – allows the user to view their history, delete their account and change their password
* Find similar song query – allows the user to enter the song they would like recommendations for
* Display similar song information – displays 4 similar songs and allows the user to redirect themselves onto the song information for these similar songs
* Disconnected from the internet – displays a message telling the user they will be allowed to access the program once connection to the internet is established

#### Use of states

I will use UI states in my programme to control which objects are shown to the user at any given time. These UI states will vary depending on the information that needs to be displayed to the user and will be determined by a UI state variable. For instance, on the program's home screen, the user must enter specific data into a text box before pressing the enter key to continue. The programme will switch to a different UI state and possibly display different objects on the screen depending on the input given by the user.

This allows less memory usage as un-needed objects do not render without being used, and allows a simple way to only display the information that is necessary.



*Figure 7. UI state navigation*

#### Classes and data types

As mentioned before, PyGame does not have integrated objects for a lot of what I want my UI to display, meaning I have to code each object in individually. Although this takes more time than using Tkinter, it allows me customise each object to the exact needs of my program. These are the objects I have had to code into my UI:

| Name of Object | Parameters | Data Types of Parameters | Function |
| --- | --- | --- | --- |
| BarChart | rect\_x, rect\_y, rect\_width, rect\_height, text, bar\_max, bar\_value | int, int, int, int, str, float, float | The BarChart class creates a few rectangles with a text box underneath. It takes in variables for position, width, height, text and the values to create a bar chart to scale with the values entered; e.g., if you were to input a max bar value of 100, and have the actual value of 25, the bar chart will display a rectangle to scale with the max height, and display a different rectangle on top proportional to the max height/4. It then displays the text under to show what the graph is representing |
| Text\_Info\_Box | text, x, y, width, height | str or list, int, int, int, int | The Text\_Info\_Box class is used to display a block of text on the screen. It has a scroll attribute that allows the text to be scrolled up or down by adjusting its vertical position. The handle\_event method listens for scroll events (mouse wheel scroll up and down) and adjusts the scroll value accordingly. The render method is responsible for drawing the text box and displaying the text within it. |
| Button | x, y, height, width, text | int, int, int, int, str | The Button class is used to create a button that can be clicked by the user. It has a pressed attribute that keeps track of whether the button is currently being pressed, and a colour attribute that changes to a different value when the button is pressed. The handle\_event method listens for mouse button down events and sets the pressed attribute to True and the colour attribute to a different value if the mouse is hovering over the button. The render\_button method is responsible for drawing the button and displaying the button's text. |
| Nav\_Bar | button\_choices, button\_spacing | list of str, int | The NavBar class is simply an array of buttons, which creates a number of buttons depending on the variables entered. Its use is to display the different menus at the top of the screen to change the UI state. |
| DropDown | x, y, height, width, text, choices | int, int, int, int, str, list of str | The DropDown class is a subclass of the Button class, meaning it inherits all the attributes and methods of the Button class and can add additional functionality on top of that. It has a selected attribute that keeps track of whether the dropdown menu is currently open or closed, and a list of choices that represent the options available in the dropdown menu. The handle\_event method has been modified to toggle the selected attribute and to allow the user to select an option from the choices list by clicking on it. The render\_button method has also been modified to display the dropdown menu and its options when the selected attribute is True, and to display the text of the selected option or the default text of the button when the selected attribute is False. |
| TextBox | x, y, width, initial\_text | int, int, int, str | The TextBox class creates an interactive text input field that displays text the user enters. When the user clicks within the field, they can begin typing. Pressing the "Enter" key will store the entered text in an output variable and clear the field. Pressing the "Backspace" key will delete the last character in the field. When the user clicks outside of the field, they can no longer enter text and the field becomes inactive. The appearance of the field changes to reflect whether it is active or inactive. |
| PasswordBox | x, y, width, initial\_text | int, int, int, str | PasswordBox, is a child class of the text box class, but instead of displaying the text on the screen as its written, it takes however many characters have been entered and displays them as stars so it is not visible to other people. |
| Text | text\_x, text\_y, text, text\_size, rect\_width, rect\_height | int, int, str, int, int, int | Creates a block of text to be displayed on the screen, taking in position parameters as well as allowing the user to specify the size of the text. It also has functionality to place a rectangle behind the text to make it more visible |
| Song\_Result | x, y, song\_name, artist\_name, album\_cover\_dir | int, int, str, str, str | The song result class if for displaying the similar songs in an effective way. Its made up of 2 text objects, a button and the album art. This allows the program to display all the necessary info about a similar song. |

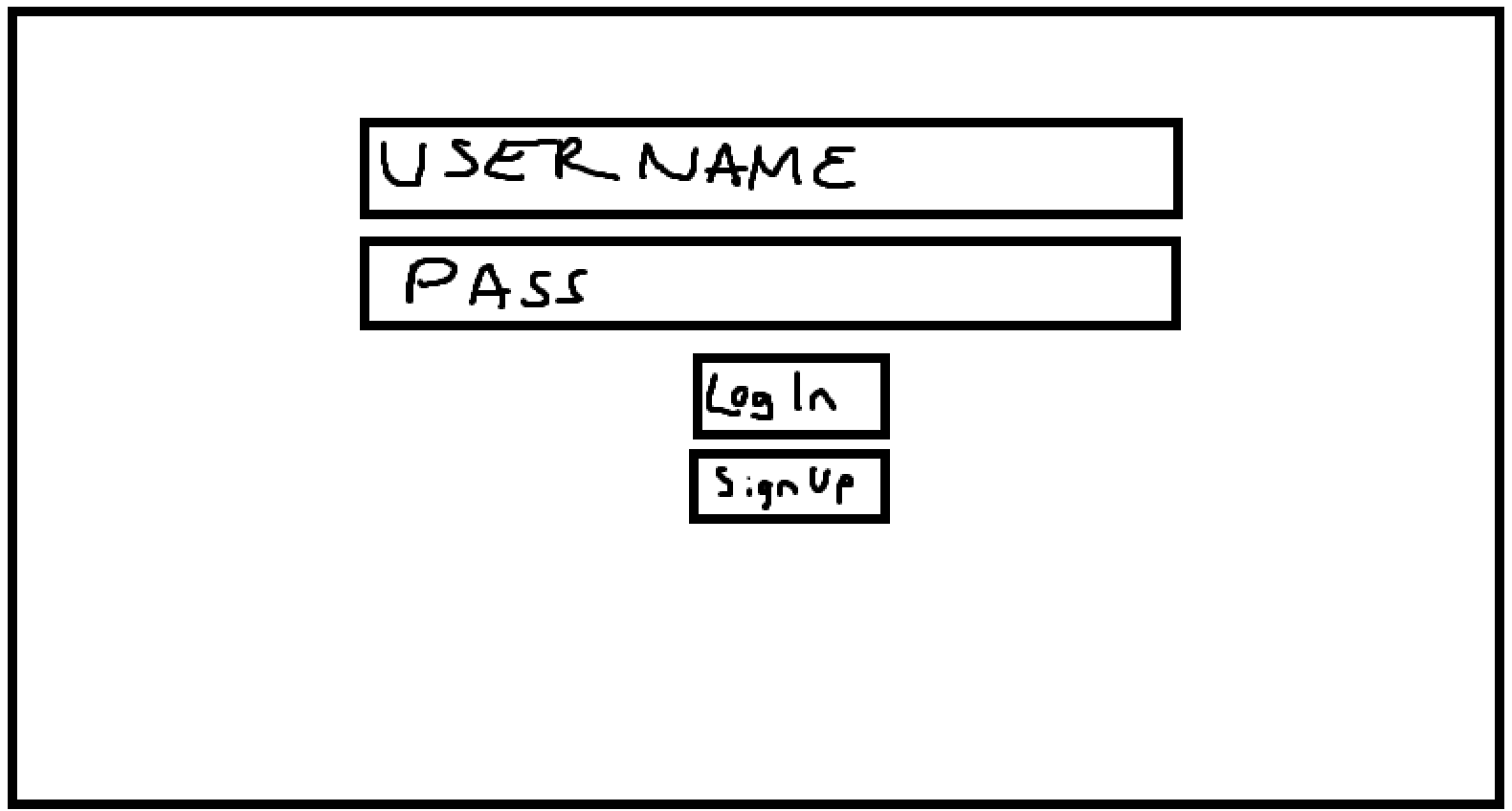
The Text class creates text on the screen, with attributes like position, width and text size. It also creates a rectangle behind the text to make it more visible.



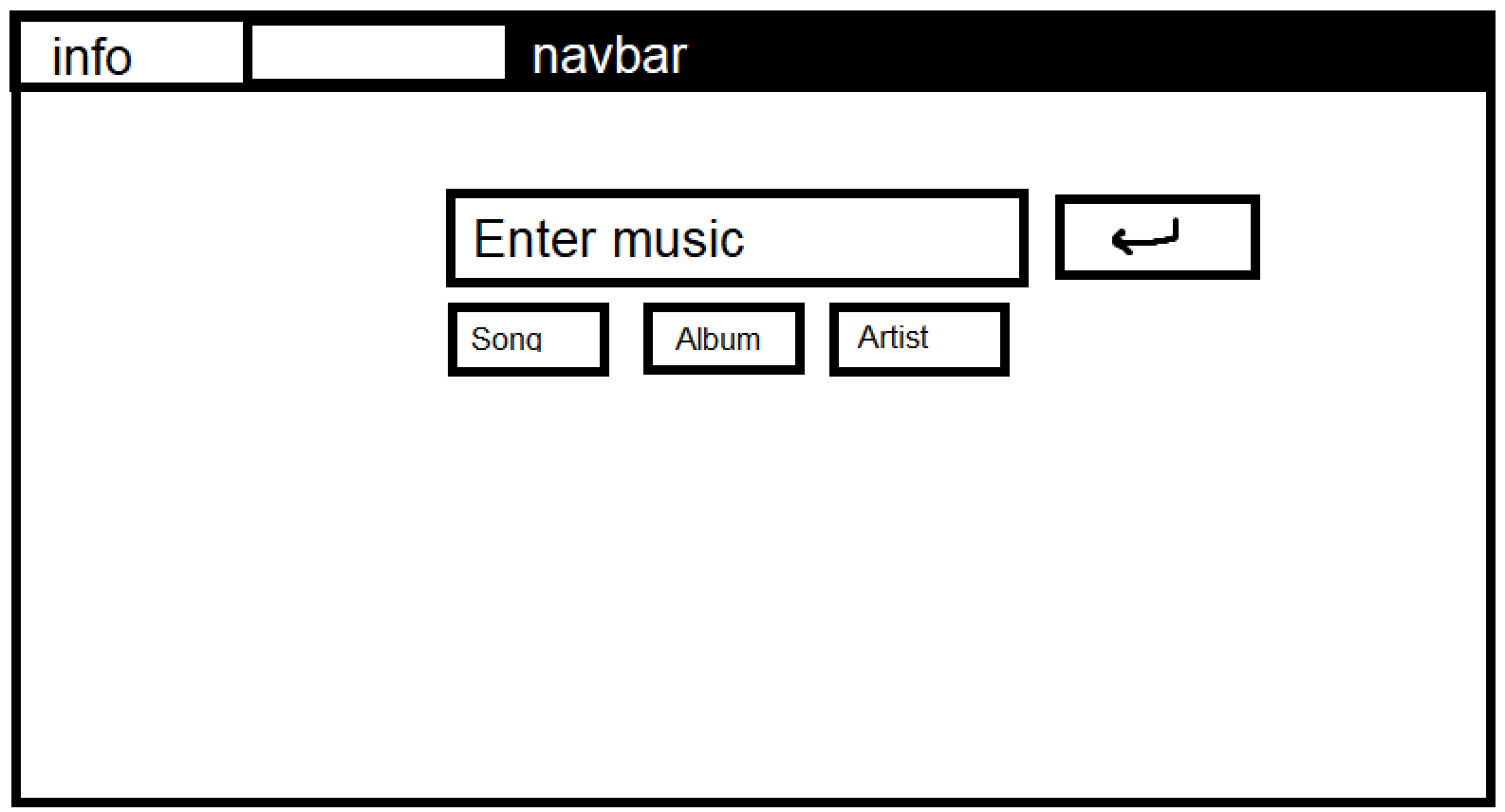
*Figure 8.* Hierarchy diagram of the UI

#### Mock ups for UI

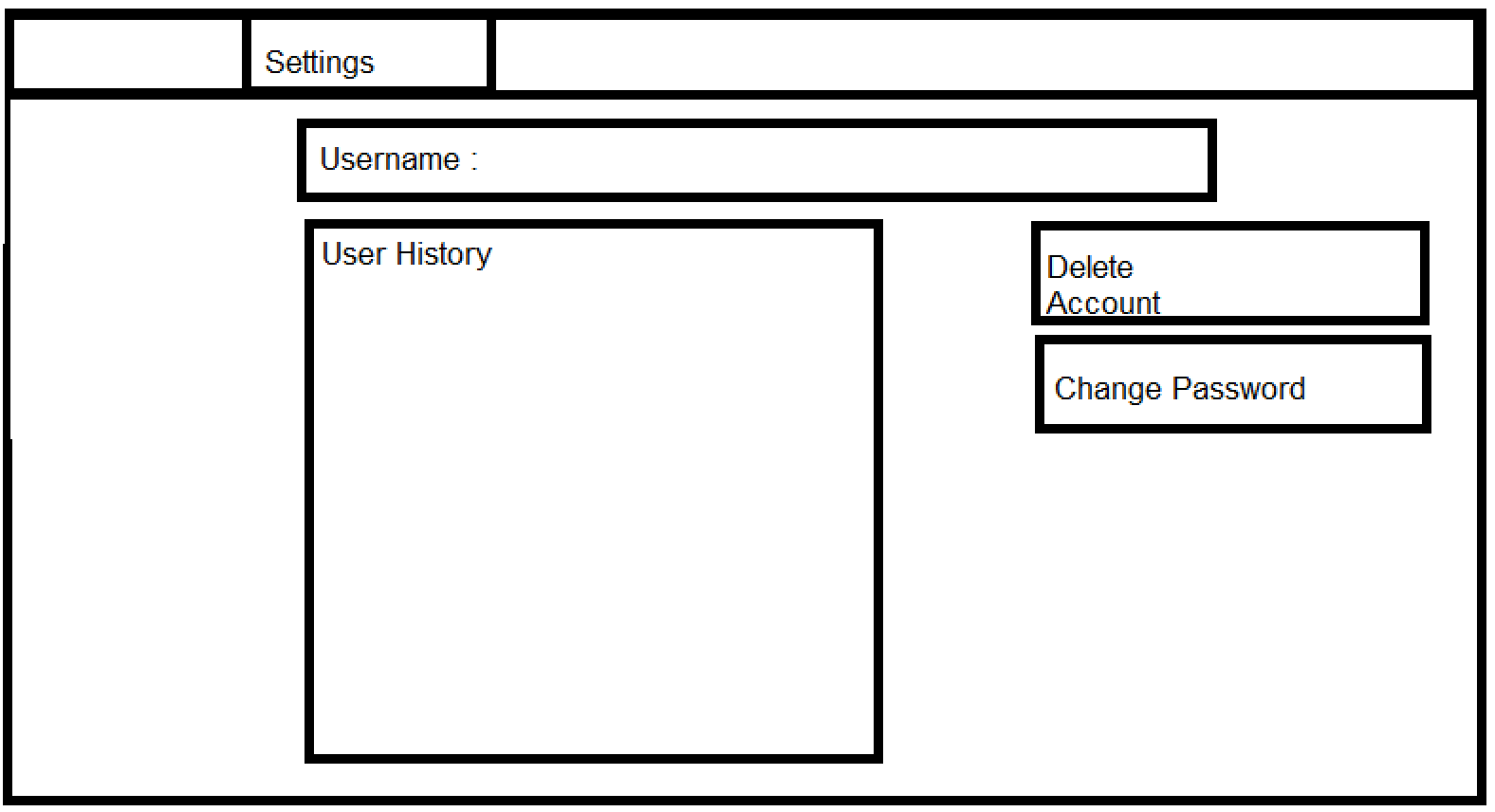
Before creating my UI, I tried to create what I thought would look like a good user interface in MS paint; I based most of my UI around my mock ups.



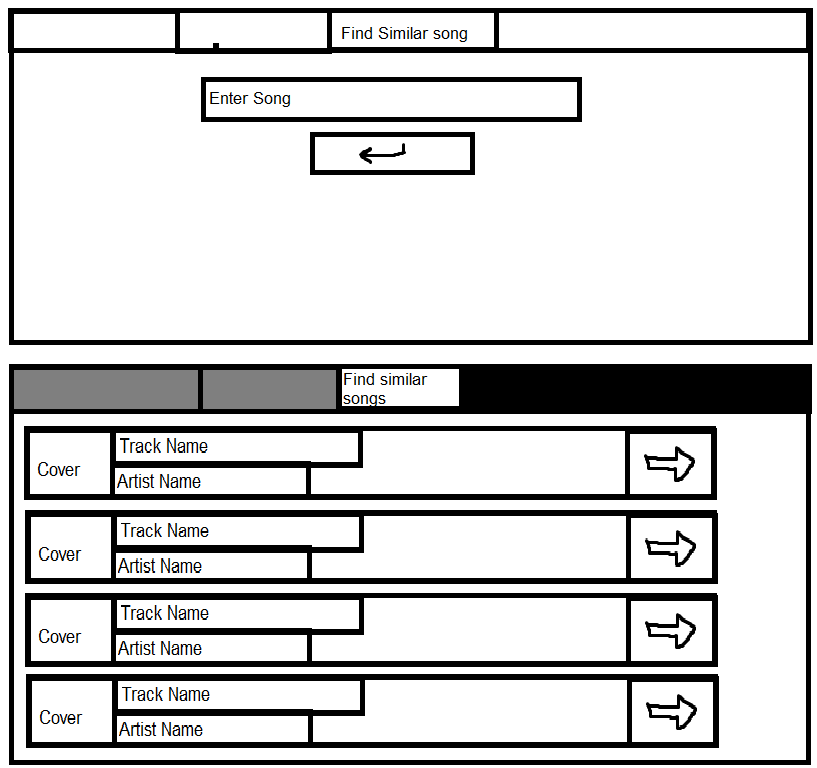
*Figure 9. Mock-up of the user log-in and registration window*



*Figure 10. Mock-up of information query window*



*Figure 11 – Mock up of the user information window*



*Figure 12. Mock-ups of the find similar song query and the display similar song information windows*

### Gathering music data from Spotify and Wikipedia

#### Introduction

To display information about the music my user wants to query, I have created a file makes gathering data much easier and reusable. I use spotipy, as mentioned in the analysis, it communicates the music data in the form of a large dictionary. In each one of my classes, I have getters which make accessing the data much less convoluted, as instead of typing something like sp.artist(“placeholder artist”)[0][“artist”][“id”][0], I can just type dataGathering.artist(“placeholder artist”).get\_id() making my code much more readable.

To access the Wikipedia I use the python libraries requests and BS4, which gather the HTML and parse it. The function accesses the artists name and generates a wikipedia link. The HTML is gathered and the correct text within the correct header is extracted and returned in array form.

For each class, I also need to display the cover art, however using spotipy, the art is returned as a URL, which pygame does not support. So I have to save the URL as an image in my directory.

The following diagram is the basic hierarchical overview.



*Figure 13. Hierarchy diagram showing the getters for each type of query*

#### Data requirements to be displayed:

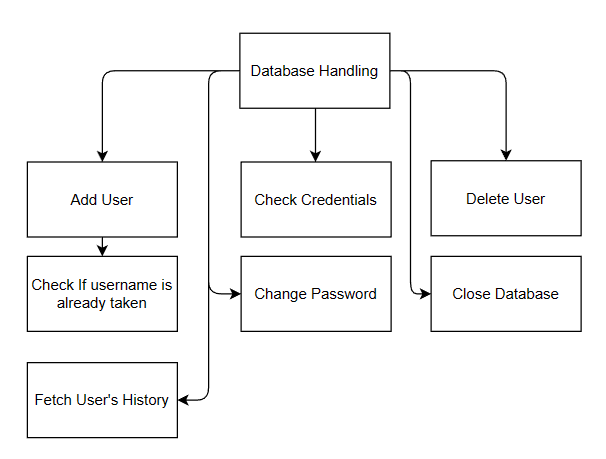
For each of my users possible inputs, I decided to create 3 separate objects each storing data important to either, album, song or artist.

* Album:
  + Album art - string
  + Release date - date
  + Genres - array
  + Album name - string
  + Artist name - string
  + Track list - array
  + Label - string
  + Followers - int
  + Length - int
* Song:
  + Track name - string
  + Popularity - int
  + Artist Name - string
  + Song art - string
  + Release Date - date
  + Audio features such as: tempo, mode, key, valence, dancability, energy - ints
  + Explicit – bool
  + Track ID – int/string
  + Disc number – int
  + Track number – int
  + Hash – int
  + Similar songs - array
* Artist:
  + Artist name- string
  + Artist Wikipedia information - array
  + Similar Artists - array
  + Top albums - array
  + Art - string
  + Top Songs - array
  + Followers - int
  + Popularity - int

### User Database and Handling

My project holds user information externally in an SQL database. My database consists of 1 table, which holds the username, password and user history of every user. The database must be stored externally as to make sure the data is saved, and not lost every time the program in terminated.

1. Username – as a string
2. Password – as a hashed string
3. History – an array of strings



*Figure 14. Hierarchy diagram for user handling*

In my code I have used the SQLite 3 library to open, read and manipulate my database.

| Function Name | Parameters | Purpose |
| --- | --- | --- |
| Add user | Username, password | This function takes in the username and password, passed in by the user on the login page. The function gets the index of the last empty row, however before appending the user details to the database, it must check if the username is not already taken. If the username is taken the function prints “repeat detected”, otherwise the updates and committed to the database |
| Check repeat | Username | This function takes in just the username as an argument. There is a check to see if any data is fetched when there is a query for information about the row with the username in it. If no data is returned. The output is false, if there is already data; the output is true. |
| Check credentials | Username, password | This function is meant to check if the user details for an existing user is correct or not. The data gathered from the login page is passed in as a parameter, the password is then hashed. Next there is a search for data that matches both the username entered and the hashed password, if there is data fetched, the credentials are correct and the program returns true. If there is no data, the function return false. |
| Delete user | Username, password | This function removes a user from the database. First the function checks if the credentials are correct. It then removes the user from the database and updates the rest of the user IDs in the database to iterate down by 1. |
| Fetch history | Username | This function takes in the username as an argument. The function will then gets the user history column of the matching username and returns it. If there is no result the function returns False |
| Change password | Username, new\_password | This function updates the users password to a new hashed password entered by the user on the user settings tab |
| Close database |  | Closes the database to ensure changes are saved. |

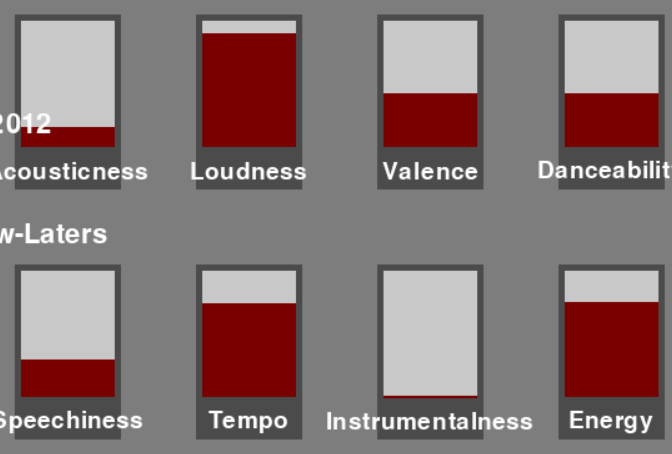
### Recommendation Algorithm

As explained earlier in the analysis, spotipy can gather information about the sound of a track provided its ID. The parameters are as follows:

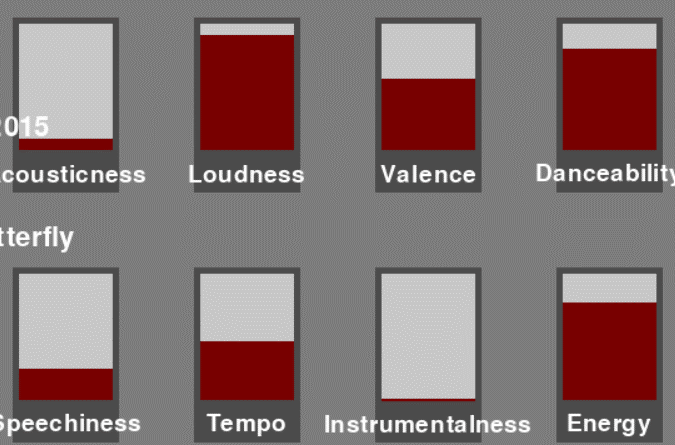
* Dancability
* Energy
* Key
* Loudness
* Speechiness
* Acousticness
* Instrumentalness
* Liveness
* Valence
* Tempo

I form a single value based on these 10 parameters, and have a dataset full of songs with their hash values. So when the user enters a song, the hash is gathered and similar songs are found based on the neighbours of the hash value.

To check if my solution worked we can first visually check within my own program two songs that my program ranked as similar, my input was “Alright – Kendrick Lamar,” one of the similar songs was “Dream II – BJ The Chicago Kid.”



Dream II



Alright

*Figure 15. Scores of the ten attributes for two example songs*

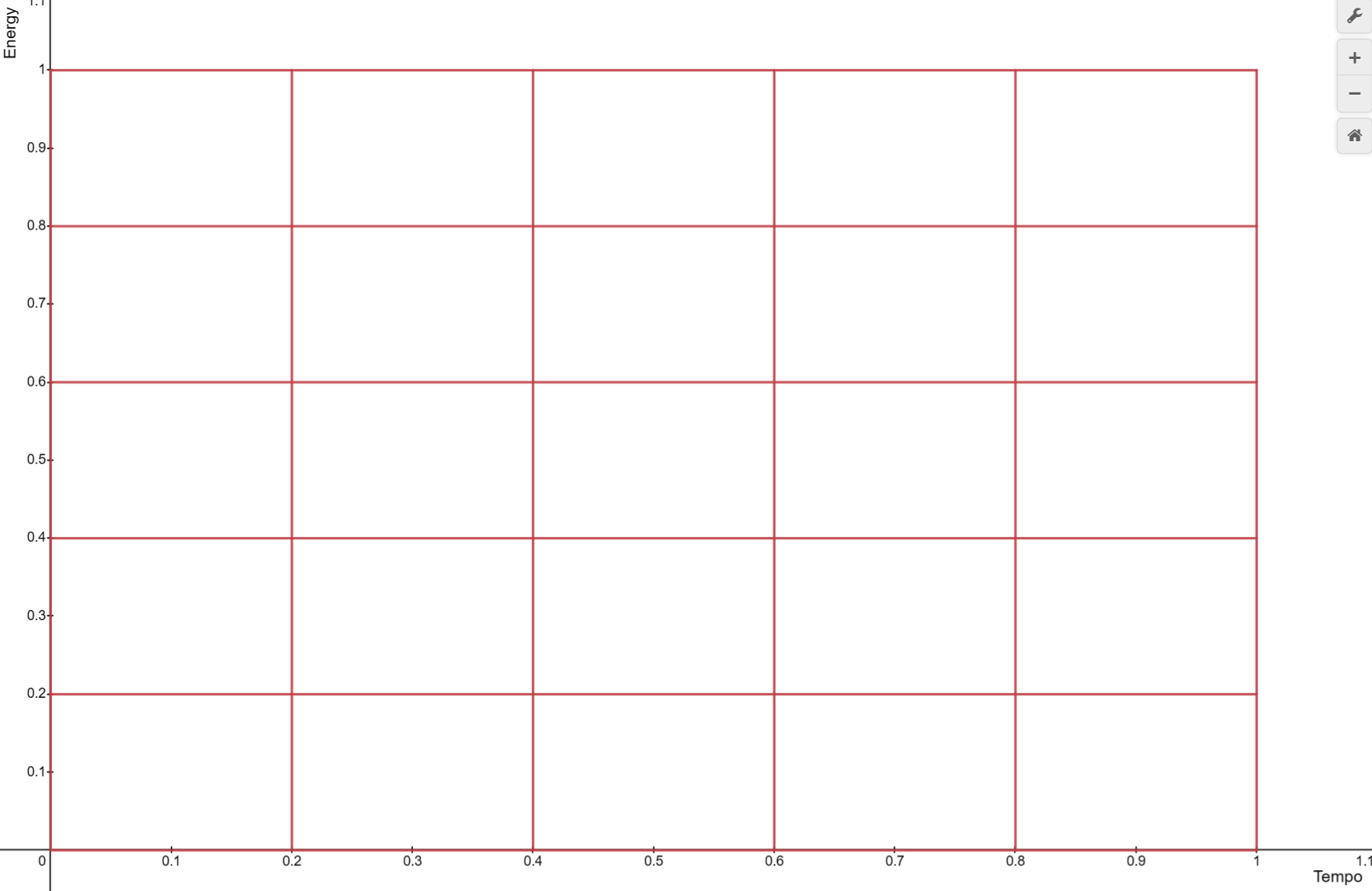
Clearly the songs are very similar sonically according to spotify. We can also calculate the similarity mathematically between the 2 songs. We do this by first calculating the maximum distance between 2 hashes, with 10 parameters; as the maximum distance between one parameter is 4 (4-0 = 4), we can find the overall maximum distance by doing pythagoras in a 10-dimensional space. √(42\*10), which gives us a max distance of 4√10. Given that is the farthest distance songs can be sonically, to calculate how similar they are, we take in 2 hashes, reverse them back to base an array, and do Pythagoras again to find the distance between them. However, this value is almost of dissimilar they are, as a lower value means they are less alike sonically. So the final calculation is: ((4√10 – dissimilarity)/4√10) \* 100, giving us a final percentage similarity. Lets put the two songs above in this process and find out how similar they are:

*The array I get for “Alright” is [3.980, 3.83, 2.916, 4.502, 1.19, 0.371, 0, 0.4135, 2.79, 1.833]*

*The array I get for “Dream II” is [2.876, 2.664, 2.6, 3.548, 0.712, 0.364, 0, 0.92, 2.64, 1.4964]*

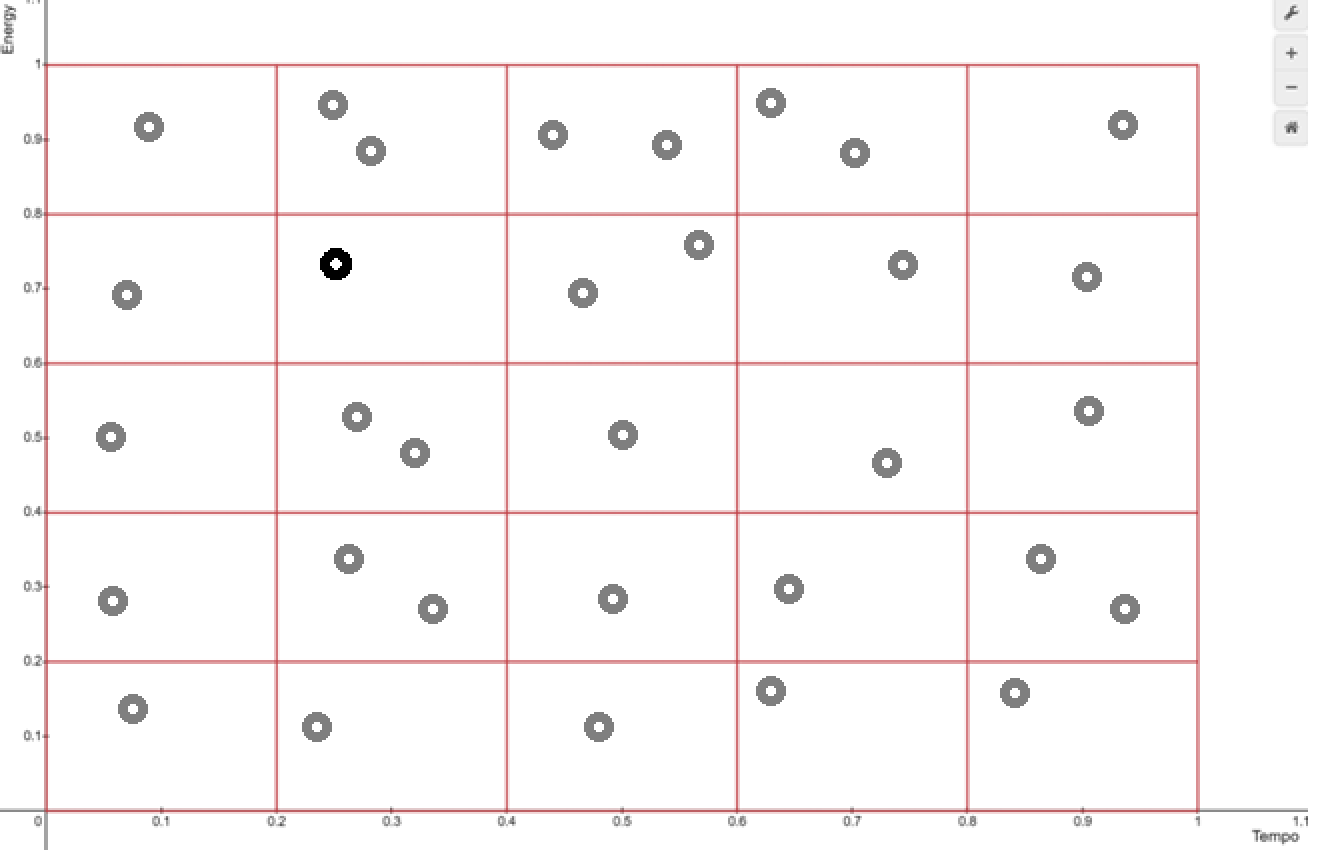
When putting these 2 values in the above process we get a similarity of 84%, proving they are sonically similar.

The way I can get similar song results using a normalised array is well represented as a graph in 2d. Let’s imagine that I was only sorting by 2 values, tempo and energy; as each value is converted in one of 5 possible number 0,1,2,3 and 4, we essentially create 5 buckets for each parameter, splitting the 2 parameters into 25 total buckets, visually represented here:



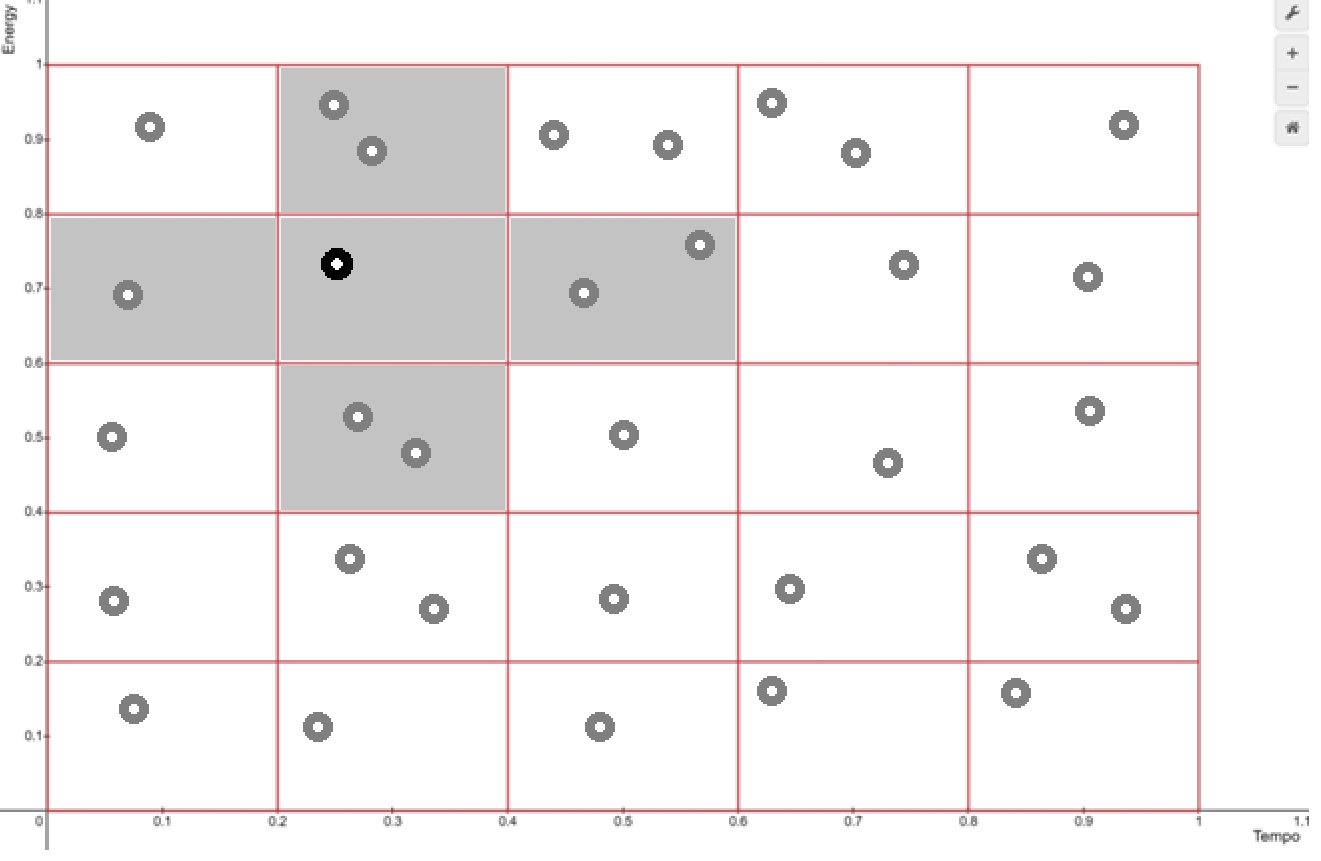
*Figure 16. An example 5 by 5 grid for two attributes (empty)*

What my program does is it find all songs in the same and adjacent buckets. Imagine our queried song was in this bucket in a populated dataset:



*Figure 17. An example 5 by 5 grid for two attributes with 32 songs*

The black point representing out song, plotted by its energy and tempo. My algorithm would return all songs its bucket and adjacent buckets, shown like this:



*Figure 18. Example of the five grid positions (shown in grey) that are closest to an example song (represented by the black dot)*

The 7 songs deemed sonically similar would then be checked if they were the right genre, and returned if they are.

This is a representation of how the algorithm works against 2 parameters, and the same concept is applied to 10 dimensions, so instead of 5 buckets being checked for songs, 201 are; giving me a large number of similar songs.

However, the concept is the same, and this is how many program finds similar songs.

#### Song comparison hashing and dataset population

Finding an efficient way to sort songs based on a similarity was a difficult task; however, I used the spotipy audio features function to first gather data about the sound of a track. This data includes the 10 parameters mentioned above.

After this the 10 parameters must be normalised to a value 0 to 1, e.g., if the loudness of a song is -35, I would add 60, making the loudness 25. I would then divide the value by 60, giving us a normalised value of 0.42.

I now have an array of 10 parameters that look something like this: [0.33, 0.532, 0.16, 0.8, 0.0263, 0.00101, 0.709, 0.101, 0.218, 0.311]. I now get the hash of this array by multiplying each value in the array by 5, and converting it to an integer. Now the array looks something like this: [1, 2, 0, 4, 0, 0, 3, 0, 1, 1]. Every parameter is sorted into a bucket with a value of 0-4.

The next step ins converting this into a base 5 number. This is done by reversing the array and multiplying them all by 5 to a power depending on the position of the array. At this point we have the hash value and it would look something like this: 2797256

Now we have the hash value me must find neighbours of this hash value, as a way to find songs with only a small difference in their sound based on the original parameters. Now I run a function to find the 2nd degree neighbours of my hash.

Now I check my dataset for every song in those hashes and add them all to a list. Finally, I iterate through every song in the array until I find 4 with the same genre of the original song I entered.

#### Drawbacks

This method is not perfect and there are definitely some issues with it, such as:

* Time: this algorithm takes multiple seconds at least to find 4 similar songs as it needs to find the genres of every song until 4 are found, If I were to repeat my process, I would add the genres as a row in my song dataset
* Songs with niche genres: songs with less common genres such as shoegaze or hyperpop take much longer to return a result as simply there are less songs with those genres, so pop or rock songs will return results much faster
* Less popular songs: songs that are less popular will generally have less genres attached to them as they are not as well known, meaning if a song is less well known, finding similar songs will take longer as it has less genres to compare and whitelist.

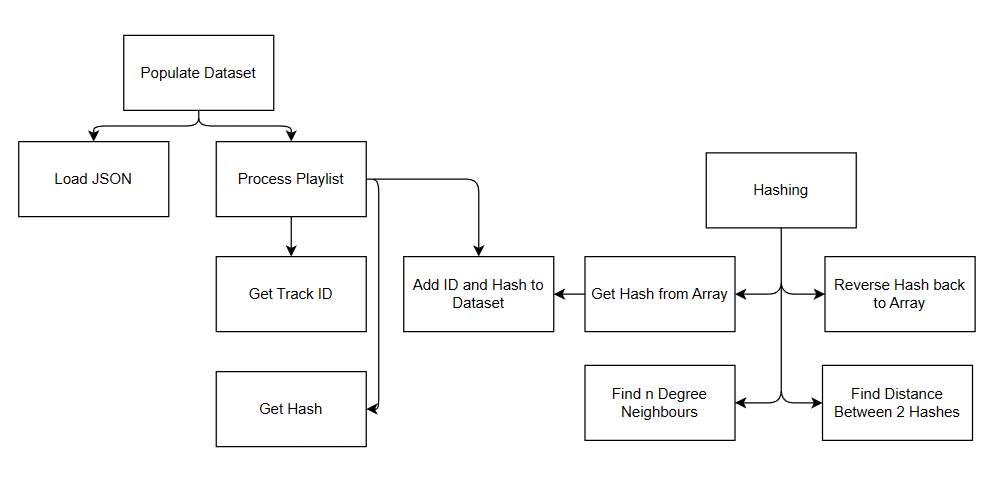
#### Populating my Song Dataset

To actually add entries to my song dataset, I needed to get a large amount of spotify IDs to generate hashes of, and within the spotify API, there is no in built way to iterate through a large amount of IDs. So I used the Spotify one million playlist challenge. This was a challenge with an extremely large amount of data, 1 million playlists, with dozens of tracks in each playlist and with the song IDs available. So I downloaded the dataset and wrote a programme to iterate through each playlist and each song in the playlist, generating a hash and adding to my dataset. This was also not a perfect solution and there are also drawbacks, here are the advantages and disadvantages.

| **Advantages** | **Disadvantages** |
| --- | --- |
| An extremely large amount of data, approximately 50 million potential song IDs | Created in 2018 so 5 years out of date, will not contain and recommend more recent songs |
| Data is created by Spotify so we know that the data is accurate and trustworthy | Every song is not unique, and filtering repeats takes time |

Additionally had I left my program running for longer, I would’ve had a larger dataset.

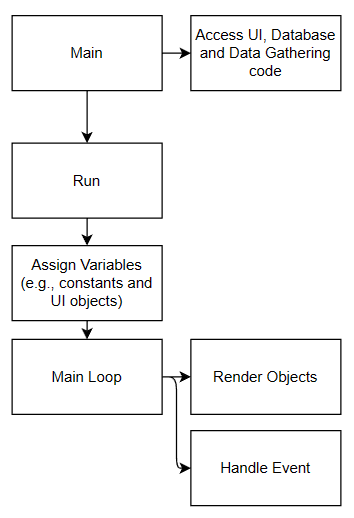
The one million playlist dataset is split up into 1000 files each containing 1000 playlists. My program reads the current file, then creates a loop through every playlist. Within the loop processing each playlist, there is another loop iterating through every track in the JSON file. Each song has a hash generated, and the track id and hash appending to my output CSV file. Once I left my code running for 2 days, I created a program to remove any repeats, leaving me with approximately 250,000 entries/



*Figure 19. Hierarchy diagram for song comparison hashing and dataset population*

### Main Function

My main function is responsible for using all the UI objects and data gathering objects I created in my separate files, to create the actual working program. Firstly in instantiates all the necessary objects and other variables at the start to ensure time is not taken creating them later. Then the main loop is started. Responsible for handling user interaction with my program and rendering the objects on my screen.



*Figure 20. High-level hierarchy diagram for the Main Function*

#### Event handling

1. Buttons – respond to presses when the mouse is within the bounds of the object
2. Scrolling text box – responds to the scroll wheel moving up and down, making sure the you cannot scroll past the end of the text, or before the beginning of the text.
3. The navbar is an array of buttons to also responds to presses
4. The drop down menu has a state when it is not dropped down, once pressed it will show the options and again respond to presses
5. Text box responds to presses of the keyboard, updating the text displayed on the screen and the value of the variable within the object
6. The password text box responds to key presses as well, however it obscures the text entered on the screen, instead outputting \* instead of the key pressed
7. Song result contains a button redirecting the user to song info page of the similar song so also responds to clicks

List of buttons and what happens when pressed

|  |  |
| --- | --- |
| Button Name | Functionality |
| login\_box | Allows user to log in with their username and password |
| sign\_up\_box | Allows user to sign up with a new username and password |
| Dropdown | Displays a dropdown list of search options |
| enter\_box | Executes the search for the entered term |
| nav\_bar.buttons[0] | Navigates to the search screen |
| nav\_bar.buttons[1] | Navigates to the user settings screen |
| nav\_bar.buttons[2] | Navigates to the find similar screen |
| clear\_history\_button | Clears the user's search history |
| sign\_out\_button | Logs the user out of their account |
| delete\_acc\_button | Deletes the user's account |
| change\_pass\_button | Allows the user to change their password |
| yes\_button | Confirms the deletion of the user's account |
| no\_button | Cancels the deletion of the user's account |
| song\_box.next\_button | Navigates to the song analysis screen |
| similar\_enter\_box | Executes the search for similar songs |
| song\_box[0-2].play\_button | Plays the corresponding song in the song box |
| song\_box[0-2].add\_button | Adds the corresponding song in the song box to the user's saved songs |
| song\_box[0-2].remove\_button | Removes the corresponding song in the song box from the user's saved songs |
| song\_box[0-2].next\_button | Navigates to the song analysis screen for the corresponding song in the song box |

#### Data validation

The following rules are applied:

* Username and password validation during login and signup:
  + Username must be at least 4 characters long.
  + Password must be at least 4 characters long and contain valid characters (alphanumeric and certain special characters).
* Account deletion confirmation:
  + The user must confirm their intent to delete their account.

#### Error handling

The following error checks are performed:

* Song name validation during search:
  + The search box cannot be empty.
* Artist name validation during artist search:
  + The artist search box cannot be empty.
* Similar song search validation:
  + The search box cannot be empty.
* Internet connection check:
  + The program checks whether the user is connected to the internet before making any API requests. If the user is not connected, an error message is displayed and the program waits for 1 second before checking again.
* Input validation for user password changes:
  + When the user is changing their password, the program checks that the new password meets the same requirements as when creating a new password.

#### Flowcharts showing how users interact with each UI State

##### Login

Diagram

Description automatically generated

*Figure 21. Flowchart for the log-in*

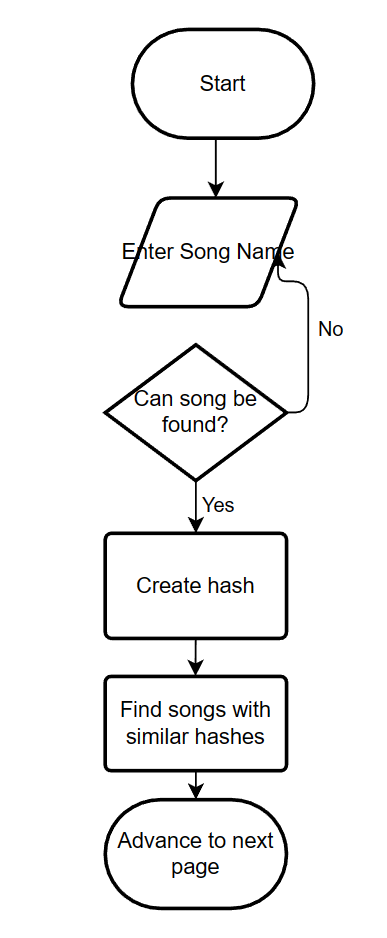
##### Music info

Diagram

Description automatically generated

*Figure 22. Flowchart for music information query*

##### Music recommendation



*Figure 23. Flowchart for similar song query*

### Internal and external data storage

Internally, within my main file I store data such as:

* UI data: each object that appears on the screen must be assigned data such as, position, width, height and text. All this data is stored internally. Additionally data about the font I want to use for my UI is stored in my program
* User data: although my users password and user data is stored externally in a database, multiple times throughout my code the database is accessed and local variables such as “username” and “password” are assigned internally to verify credentials and edit user data
* Music data: when a user enters a song data must be stored in my program to be displayed on the screen. This data includes attributes like: name, id etc.
* UI state: keeping track of what screen the user is on is essential to make the program functional and keep the program from taking up memory and CPU power rendering and handling objects that are not needed

Externally I store:

* User database: my database of users includes information such as, username, password and user history data. This is necessary to be stored outside of my program because otherwise whenever the program is terminated, all user data would be lost
* Song dataset: when a user enters a song to find recommendations, my program must check the hash of the entered song to find recommendations. To examine the data of an extremely large number of songs in real time is not feasible and would take a very large amount of time; so my dataset is essential for the function of my algorithm and saving time.
* Images: When I need to display images of music cover art on my screen, pygame requires a directory of the image I want to display. So I must convert the cover art URL into an image in my programs directory

## Technical solution

### UI.py

import csv  
import json  
import time  
  
import spotipy  
import os  
import hashing  
from spotipy.oauth2 import SpotifyClientCredentials  
  
os.environ['SPOTIPY\_CLIENT\_ID'] = '3887776ea50a469b81ab216075517bcf'  
os.environ['SPOTIPY\_CLIENT\_SECRET'] = '52bf4482e5ec4256b23b5e6ef6c49a6f'  
  
  
def load\_json(file\_path):  
 with open(file\_path, 'r') as f:  
 data = json.load(f)  
 return data  
  
  
def process\_playlist(tracks, csv\_path):  
 sp = spotipy.Spotify(client\_credentials\_manager=SpotifyClientCredentials())  
 with open(csv\_path, 'a', newline='') as f:  
 writer = csv.writer(f)  
 # Load existing entries into a set  
 existing\_entries = set()  
 if os.path.exists(csv\_path):  
 with open(csv\_path, 'r', newline='') as csv\_file:  
 reader = csv.reader(csv\_file)  
 next(reader) # Skip header row  
 for row in reader:  
 existing\_entries.add(row[0])  
 for track in tracks:  
 try:  
 song\_uri = track["track\_uri"]  
 # Check if entry already exists  
 if song\_uri not in existing\_entries:  
 # Add track ID to existing entries  
 existing\_entries.add(song\_uri)  
 start = time.time()  
 audio\_features = sp.audio\_features(song\_uri)[0]  
 arr = [  
 audio\_features['danceability'],  
 audio\_features['energy'],  
 (audio\_features['key'] / 12),  
 ((audio\_features['loudness'] + 60) / 60),  
 audio\_features['speechiness'],  
 audio\_features['acousticness'],  
 audio\_features['instrumentalness'],  
 audio\_features['liveness'],  
 audio\_features['valence'],  
 (audio\_features['tempo'] / 300)  
 ]  
 song\_hash = hashing.get\_hash(arr)  
 writer.writerow([song\_uri, song\_hash])  
 f.flush()  
 end = time.time()  
 print(f"Processed track with URI {song\_uri} in playlist, hash is {song\_hash}, time taken is {end-start}s")  
 except:  
 print('sleeping')  
 time.sleep(60)  
  
 # sort the csv by hash  
 with open(csv\_path, 'r', newline='') as f:  
 reader = csv.reader(f)  
 sorted\_rows = sorted(reader, key=lambda x: x[1])  
 with open(csv\_path, 'w', newline='') as f:  
 writer = csv.writer(f)  
 writer.writerow(['track\_uri', 'hash'])  
 writer.writerows(sorted\_rows)  
  
  
def main(start\_index=0):  
 directory = "D:\Desktop\COMSPCI NEA\desktop\data"  
 for filename in os.listdir(directory):  
 if filename.endswith(".json"):  
 print(filename)  
 file\_path = os.path.join(directory, filename)  
 with open(file\_path, "r") as f:  
 data = json.load(f)  
 playlists = data['playlists']  
 for i in range(start\_index, len(playlists)):  
 playlist = playlists[i]  
 tracks = playlist['tracks']  
 process\_playlist(tracks, "output.csv")  
 print("Processing complete!")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main(0)

### DataGathering.py

import math  
import random  
  
import pandas as pd  
import spotipy  
from spotipy.oauth2 import SpotifyClientCredentials  
from bs4 import BeautifulSoup  
import os  
import requests  
import re  
import datetime  
from PIL import Image  
from io import BytesIO  
import hashing  
  
os.environ['SPOTIPY\_CLIENT\_ID'] = '3887776ea50a469b81ab216075517bcf'  
os.environ['SPOTIPY\_CLIENT\_SECRET'] = '52bf4482e5ec4256b23b5e6ef6c49a6f'  
sp = spotipy.Spotify(client\_credentials\_manager=SpotifyClientCredentials())  
  
  
class Artist:  
 def \_\_init\_\_(self, name):  
 results = sp.search(q=name, type="artist", limit=1)  
 self.results = results['artists']['items'][0]  
 self.albums = sp.artist\_albums(self.get\_id())  
  
 def get\_name(self):  
 return self.results['name']  
  
 def get\_followers(self):  
 return self.results['followers']['total']  
  
 def get\_genres(self):  
 return self.results['genres']  
  
 def get\_id(self):  
 return self.results['id']  
  
 def get\_popularity(self):  
 return self.results["popularity"]  
  
 def get\_spotify\_url(self):  
 return self.results['uri']  
  
 def get\_web\_url(self):  
 return self.results['external\_urls']['spotify']  
  
 def get\_top\_song(self):  
 top\_tracks = sp.artist\_top\_tracks(self.get\_id())  
 tracks = []  
 for i in range(5):  
 tracks.append(top\_tracks['tracks'][i - 1]['name'])  
 return tracks  
  
  
 def find\_intro\_paragraphs(self):  
 name = f"{self.get\_name()}"  
 link = "https://en.wikipedia.org/wiki/" + name  
 response = requests.get(link)  
 paras = []  
 if response is not None:  
 html = BeautifulSoup(response.text, 'html.parser')  
 title = html.select("#firstHeading")[0].text  
 paragraphs = html.select("p")  
 para = ' '.join([para.text.strip() for para in paragraphs[0:5]])  
 paras.append(para)  
 return paras  
  
 def get\_album\_imgs(self):  
  
 album\_names = []  
 album\_ids = [album['id'] for album in self.albums['items'] if album['name'] not in album\_names]  
 albums = [sp.album(id) for id in album\_ids]  
 popularity = [album['popularity'] for album in albums]  
  
 sorted\_pop = sorted(popularity, reverse=True)[:3]  
 popular\_albums = [album for popularity, album in zip(popularity, albums) if popularity in sorted\_pop]  
  
 image\_urls = []  
 for album in popular\_albums:  
 image\_urls.append(album['images'][0]['url'])  
 return image\_urls  
  
 def similar\_artists(self):  
 related\_artists = sp.artist\_related\_artists(self.get\_id())  
 artists = []  
 for i in related\_artists['artists']:  
 artists.append(i['name'])  
 return artists  
  
  
class Album:  
 def \_\_init\_\_(self, name):  
 self.data = sp.search(q=name, type='album', limit=1)['albums']['items'][0]  
 self.data2 = sp.album((self.get\_id()))  
 self.tracks = sp.album\_tracks(self.get\_id())['items']  
 self.artist = sp.artist(self.get\_artist\_id())  
  
 def get\_genres(self):  
 return self.artist['genres']  
  
 def get\_followers(self):  
 return self.artist['followers']['total']  
  
 def get\_label(self):  
 return self.data2['label']  
  
 def get\_name(self):  
 return self.data['name']  
  
 def get\_id(self):  
 return self.data['id']  
  
 def get\_tracks(self):  
 track\_list = []  
 for i in self.data2['tracks']['items']:  
 track\_list.append(i['name'])  
 result = ', '.join(track\_list)  
 return result  
  
 def get\_album\_length(self):  
 duration = sum(track['duration\_ms'] for track in self.data2['tracks']['items']) / 3.6e6  
 hours, mins = divmod(duration, 1)  
 hours = math.floor(hours)  
 mins = math.floor(mins \* 60)  
 if hours:  
 return f"{hours} hours and {mins} minutes."  
 else:  
 return f"{mins} minutes."  
  
 def get\_artist(self):  
 return self.data['artists'][0]['name']  
  
 def get\_artist\_id(self):  
 return self.data['artists'][0]['id']  
  
 def get\_album\_image(self):  
 return self.data['images'][0]['url']  
  
 def get\_popularity(self):  
 return self.data2['popularity']  
  
 def get\_web\_link(self):  
 return self.data['external\_urls']['spotify']  
  
 def get\_release\_date(self):  
 return self.data['release\_date']  
  
 def get\_total\_tracks(self):  
 return self.data['total\_tracks']  
  
 def get\_mean\_audio\_feature(self, feature):  
 total = 0  
 count = 0  
 for track in self.tracks:  
 audio\_features = sp.audio\_features(track['id'])[0]  
 total += audio\_features[feature]  
 count += 1  
 return total / count  
  
 def get\_members(self):  
 return self.artist['members']  
  
 def format\_info(self):  
 data = [f"Date Created: {self.get\_release\_date()}",  
 f"Genres: {self.get\_genres()}",  
 f"Total Tracks: {self.get\_total\_tracks()}",  
 f"Track Names: {self.get\_tracks()}",  
 f"Total Length: {self.get\_album\_length()}"]  
 return data  
  
 def get\_average\_features(self):  
 features = ["acousticness", "speechiness", "loudness", "tempo", "valence", "instrumentalness", "danceability",  
 "energy"]  
 averages = []  
 for feature in features:  
 average = self.get\_mean\_audio\_feature(feature)  
 averages.append(average)  
 rounded\_values = [round(value, 2) for value in averages]  
 rounded\_values[2] += 60  
 if rounded\_values[3] > 240: rounded\_values[3] = 240  
 return rounded\_values  
  
  
class Song:  
 def \_\_init\_\_(self, name, form):  
 if form == "name":  
 self.name = name  
 self.data = sp.search(q=name, type='track', limit=1)['tracks']['items'][-1]  
 elif form == "uri":  
 self.data = sp.track(name)  
 elif form == "id":  
 self.data = sp.track(track\_id=name)  
 self.audio\_features = sp.audio\_features(self.data['id'])[0]  
  
 def get\_track\_name(self):  
 return self.data['name']  
  
 def get\_track\_popularity(self):  
 return self.data['popularity']  
  
 def get\_track\_spotify\_web\_url(self):  
 return self.data['external\_urls']['spotify']  
  
 def get\_track\_id(self):  
 return self.data['id']  
  
 def get\_artist\_name(self):  
 return self.data['artists'][0]['name']  
  
 def get\_artist\_id(self):  
 return self.data['artists'][0]['id']  
  
 def get\_track\_preview\_url(self):  
 return self.data['preview\_url']  
  
 def get\_spotify\_url(self):  
 return self.data['uri']  
  
 def get\_album\_name(self):  
 return self.data['album']['name']  
  
 def get\_album\_id(self):  
 return self.data['album']['id']  
  
  
 def get\_album\_image(self):  
 get\_image\_png(self.data['album']['images'][0]['url'], "album1.png")  
  
 # -----  
  
 def get\_disc\_number(self):  
 return self.data['disc\_number']  
  
 def get\_track\_number(self):  
 return self.data['track\_number']  
  
 def get\_release\_date(self):  
 release = self.data['album']['release\_date']  
 try:  
 date = datetime.datetime.strptime(release, "%Y-%m-%d")  
 formatted\_date = date.strftime("%d/%m/%Y")  
 return formatted\_date  
 except:  
 return release  
  
 def get\_year(self):  
 return self.get\_release\_date().split('/')[-1]  
  
 def get\_danceability(self):  
 return self.audio\_features['danceability']  
  
 def get\_energy(self):  
 return self.audio\_features['energy']  
  
 def get\_key(self):  
 return self.audio\_features['key']  
  
 def get\_loudness(self):  
 return self.audio\_features['loudness']  
  
 def get\_mode(self):  
 return self.audio\_features['mode']  
  
 def get\_speechiness(self):  
 return self.audio\_features['speechiness']  
  
 def get\_acousticness(self):  
 return self.audio\_features['acousticness']  
  
 def get\_instrumentalness(self):  
 return self.audio\_features['instrumentalness']  
  
 def get\_liveness(self):  
 return self.audio\_features['liveness']  
  
 def get\_valence(self):  
 return self.audio\_features['valence']  
  
 def get\_tempo(self):  
 if self.audio\_features['tempo'] >= 300: return 300  
 else: return self.audio\_features['tempo']  
  
 def get\_duration\_ms(self):  
 return self.audio\_features['duration\_ms']  
  
 def get\_time\_signature(self):  
 return self.audio\_features['time\_signature']  
  
 def is\_explicit(self):  
 match self.data['explicit']:  
 case True:  
 return "TRUE"  
 case False:  
 return "FALSE"  
  
 def get\_genres(self):  
 genres = sp.artist(self.get\_artist\_id())['genres']  
 return genres  
  
  
 def format\_string\_data(self):  
 return [f"Year Released: {self.get\_release\_date()} ",  
 f"Disc Number : {self.get\_disc\_number()} ",  
 f"Track Number : {self.get\_track\_number()} ",  
 f"Popularity : {self.get\_track\_popularity()}/100 ",  
 f"Album : {self.get\_album\_name()}"]  
 def get\_hash(self):  
 arr = [  
 self.get\_danceability(),  
 self.get\_energy(),  
 (self.get\_key() / 12),  
 ((self.get\_loudness() + 60) / 60),  
 self.get\_speechiness(),  
 self.get\_acousticness(),  
 self.get\_instrumentalness(),  
 self.get\_liveness(),  
 self.get\_valence(),  
 (self.get\_tempo() / 300)  
 ]  
 hash\_value = hashing.get\_hash(arr)  
 return hash\_value  
  
 def format\_data(self):  
 return [  
 self.get\_track\_id(),  
 self.get\_hash()  
 ]  
  
 def return\_similar(self):  
 neighbours = hashing.find\_neighbours(self.get\_hash(), 2)  
 df = pd.read\_csv("without repeats.csv")  
 matching\_tracks = []  
 for number in neighbours:  
 matching\_rows = df[df[' hash'] == number]  
 if not matching\_rows.empty:  
 matching\_tracks.extend(matching\_rows['track\_id'].tolist())  
 genres = self.get\_genres()  
 random.shuffle(matching\_tracks)  
 similar = []  
 names = []  
 for song\_uri in matching\_tracks:  
 try:  
 current\_data = sp.track(song\_uri)  
 current\_genres = sp.artist(current\_data['artists'][0]['id'])['genres']  
 for genre in current\_genres:  
 if genre in genres:  
 if current\_data['name'] not in names:  
 print('SUCCESS')  
 similar.append(Song(song\_uri,"uri"))  
 names.append(current\_data['name'])  
 url = current\_data['album']['images'][0]['url']  
 get\_image\_png(url,f"D:\\Desktop\\COMSPCI NEA\\laptop\\album{len(similar)}.png")  
  
 if len(similar) >= 4: break  
 except:  
 print("error")  
  
 return similar  
  
 def get\_arrray(self):  
 arr = [  
 self.get\_danceability(),  
 self.get\_energy(),  
 (self.get\_key() / 12),  
 ((self.get\_loudness() + 60) / 60),  
 self.get\_speechiness(),  
 self.get\_acousticness(),  
 self.get\_instrumentalness(),  
 self.get\_liveness(),  
 self.get\_valence(),  
 (self.get\_tempo() / 300)  
 ]  
 return arr  
  
  
def get\_image\_png(url, name):  
 response = requests.get(url)  
  
 image = Image.open(BytesIO(response.content))  
  
 image.save(name)

### Main.py

import time  
  
from UI import \*  
import pygame  
import sys  
import dataGathering  
import login  
from PIL import Image  
import requests  
  
  
def gather\_data(name, choice, username):  
 df = login.Database("usernames.db")  
 new\_history = df.fetch\_history(username)  
 match choice:  
 case "Artist":  
 artist = dataGathering.Artist(name)  
 new\_history += f"{choice}: {artist.get\_name()}, "  
 return artist  
 case "Album":  
 album = dataGathering.Album(name)  
 new\_history += f"{choice}: {album.get\_name()}, "  
 return album  
 case "Song":  
 song = dataGathering.Song(name, "name")  
 new\_history += f"{choice}: {song.get\_track\_name()}, "  
 return song  
 df.update\_history(username, new\_history)  
 df.close\_database()  
  
  
def is\_password\_valid(password): # 48-57, 65-122  
 if len(password) < 4: return False  
 for letter in password:  
 char = ord(letter)  
 if not ((48 <= char <= 57) or (65 <= char <= 90) or (97 <= char <= 122)):  
 return False  
 return True  
  
  
def run():  
 base\_font = pygame.font.Font(None, 32)  
 clock = pygame.time.Clock()  
 ui\_state = 8  
  
 username = ''  
 # ENTER MUSIC STATE  
 box = TextBox(280, 100, 400, "Enter Query")  
 dropdown = DropDown(420, 150, 32, 120, "Toggle", ['Song', 'Artist', 'Album'])  
 enter\_box = Button(700, 100, 32, 100, "Enter")  
 nav\_bar = Nav\_Bar(['Search', 'User Settings', 'Find Similar'], 175)  
 # LOGIN STATE  
 username\_box = TextBox(280, 50, 400, "Enter Username")  
 password\_box = PasswordBox(280, 100, 400, "Enter Password")  
 login\_box = Button(280, 200, 32, 100, "Login")  
 sign\_up\_box = Button(400, 200, 32, 100, "Sign Up")  
 login\_result\_timer = 0  
  
 result = 0 # the result of the user entered details  
 # USER SETTINGS  
 sign\_out\_button = Button(750, 50, 80, 200, "Sign Out")  
 delete\_acc\_button = Button(750, 150, 80, 200, "Delete Account")  
 clear\_history\_button = Button(35, 50, 80, 200, "Clear History")  
 username\_text = Text(280, 50, 'placeholder username', 40, 400, 40)  
 user\_history\_text = Text(280, 110, "User History", 32, 160, 40)  
 user\_history = Text\_Info\_Box("placeholder user history", 280, 150, 395, 325)  
 password\_delete = PasswordBox(750, 250, 200, "Enter Password")  
 change\_pass\_button = Button(750, 300, 80, 200, "Change Password")  
 new\_change\_password = PasswordBox(750, 400, 200, "New Password")  
 old\_password = PasswordBox(750, 450, 200, "Old Password")  
 yes\_button = Button(260, 350, 100, 100, "Yes")  
 no\_button = Button(385, 350, 100, 100, "No")  
 delete = False  
  
 # ALBUM RESULTS  
 bar\_charts = [BarChart(400, 225, 75, 100, 'Acousticness', 100, 75),  
 BarChart(400, 375, 75, 100, 'Speechiness', 100, 75),  
 BarChart(545, 225, 75, 100, 'Loudness', 100, 75), BarChart(545, 375, 75, 100, 'Tempo', 100, 75),  
 BarChart(690, 225, 75, 100, 'Valence', 100, 75),  
 BarChart(690, 375, 75, 100, 'Instrumentalness', 100, 75),  
 BarChart(835, 225, 75, 100, 'Danceability', 100, 75), BarChart(835, 375, 75, 100, 'Energy', 100, 75)]  
 album\_name\_text = Text(400, 50, 'placeholder album', 40, 500, 45)  
 album\_artist\_text = Text(400, 110, 'placeholder artist', 32, 250, 42)  
 maxes = [1, 1, 60, 240, 1, 1, 1, 1]  
 album\_info\_box = Text\_Info\_Box("placeholder album info", 45, 315, 255, 200)  
 album\_info\_box.font\_size = 20  
 album\_name, artist\_name = "placholder", "placholder"  
 # ARTIST RESULTS  
 artist\_name\_text = Text(180, 55, "placeholder artist", 40, 765, 45)  
 artist\_bio\_text = Text\_Info\_Box("placholder artist bio", 180, 110, 530, 420)  
 top\_songs\_text = Text\_Info\_Box("placeholder top songs", 730, 110, 210, 185)  
 similar\_artists\_text = Text\_Info\_Box("placeholder artists", 730, 300, 210, 230)  
 # SONG RESULTS  
 song\_name\_text = Text(300, 50, "placeholder song name", 40, 645, 40)  
 song\_artist\_text = Text(300, 100, "placeholder artist name", 30, 345, 30)  
 offsetx = 100  
 offsety = 50  
 song\_bar\_charts = [BarChart(400 - offsetx, 225 - offsety, 75, 100, 'Acousticness', 100, 75),  
 BarChart(400 - offsetx, 375, 75, 100, 'Speechiness', 100, 75),  
 BarChart(545 - offsetx, 225 - offsety, 75, 100, 'Loudness', 100, 75),  
 BarChart(545 - offsetx, 375, 75, 100, 'Tempo', 100, 75),  
 BarChart(690 - offsetx, 225 - offsety, 75, 100, 'Valence', 100, 75),  
 BarChart(690 - offsetx, 375, 75, 100, 'Instrumentalness', 100, 75),  
 BarChart(835 - offsetx, 225 - offsety, 75, 100, 'Danceability', 100, 75),  
 BarChart(835 - offsetx, 375, 75, 100, 'Energy', 100, 75)]  
 song\_info\_box = Text\_Info\_Box("placeholder song info", 45, 245, 205, 250)  
 song\_info\_box.font\_size = 20  
 # FIND SIMILAR SONG  
 similar\_text\_box = TextBox(280, 100, 400, "Enter Song")  
 similar\_enter\_box = Button(700, 100, 32, 100, "Enter")  
  
 # SIMILAR RESULTS  
 song\_boxes = [Song\_Result(20, 50, "Song1", "Artist", "album1.png"),  
 Song\_Result(20, 160, "Song2", "Artist", "album2.png"),  
 Song\_Result(20, 270, "Song3", "Artist", "album3.png"),  
 Song\_Result(20, 380, "Song4", "Artist", "album4.png")]  
 #NOT CONNECTED  
 not\_connected = Text(000,200,"Not Connected to the Internet, Will Redirect When Connection is Back",40,0,0)  
  
  
 while True:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 sys.exit()  
  
 # LOGIN STATE  
 # ------------------------------------------  
 if ui\_state == 0:  
 username\_box.handle\_event(event)  
 password\_box.handle\_event(event)  
 login\_box.handle\_event(event)  
 sign\_up\_box.handle\_event(event)  
 if (password\_box.user\_text != ''):  
 if (username\_box.user\_text != ''):  
 if login\_box.pressed:  
 df = login.Database("usernames.db")  
 if df.check\_credentials(username\_box.user\_text, password\_box.user\_text):  
 username = username\_box.user\_text  
 username\_text.text = f"Username: {username}"  
 user\_history.lines = [substring.strip(",") for substring in  
 df.fetch\_history(username).split(", ")]  
 username\_box.user\_text = ''  
 password\_box.user\_text = ''  
 ui\_state = 1  
 else:  
 result = 1  
 df.close\_database()  
 if sign\_up\_box.pressed:  
 df = login.Database("usernames.db")  
 if df.check\_repeat(username\_box.user\_text):  
 result = 2  
 else:  
 username = username\_box.user\_text  
 if is\_password\_valid(username):  
 if not is\_password\_valid(password\_box.user\_text):  
 result = 9  
 else:  
 df.add\_user(username, password\_box.user\_text)  
 result = 3  
 else:  
 result = 8  
 df.close\_database()  
 username\_box.user\_text = ''  
 password\_box.user\_text = ''  
 else:  
 if login\_box.pressed or sign\_up\_box.pressed:  
 result = 12  
 else:  
 if login\_box.pressed or sign\_up\_box.pressed:  
 result =12  
 # ------------------------------------------  
 # ENTER MUSIC STATE  
 elif ui\_state == 1:  
 box.handle\_event(event)  
 dropdown.handle\_event(event)  
 enter\_box.handle\_event(event)  
 change = nav\_bar.handle\_event(event)  
 if change is not None:  
 ui\_state = change  
  
 if box.user\_text != '' and enter\_box.pressed:  
 choice = dropdown.selected\_option  
 last\_ouput = box.user\_text  
 if choice is not None:  
 df = login.Database('usernames.db')  
 user\_history\_str = df.fetch\_history(username)  
 user\_history\_str += f"{choice}: {box.user\_text}, "  
 df.update\_history(username, user\_history\_str)  
 user\_history.lines = [substring.strip(",") for substring in  
 df.fetch\_history(username).split(", ")]  
 df.close\_database()  
 try:  
 music\_data = gather\_data(last\_ouput, choice, username)  
 except:  
 result = 10  
 break  
 if choice == 'Album':  
  
 album\_name = music\_data.get\_name()  
 artist\_name = music\_data.get\_artist()  
 dataGathering.get\_image\_png(music\_data.get\_album\_image(), 'image.png')  
 mean\_audio\_features = music\_data.get\_average\_features()  
 feature\_map = {"Acousticness": 0, "Speechiness": 1, "Loudness": 2, "Tempo": 3, "Valence": 4,  
 "Instrumentalness": 5, "Danceability": 6, "Energy": 7}  
  
 for i in bar\_charts:  
 index = feature\_map.get(i.text)  
 if index is not None:  
 i.bar\_value = mean\_audio\_features[index]  
 i.bar\_max = maxes[index]  
  
 album\_info\_box.get\_lines(music\_data.format\_info())  
 ui\_state = 3  
 # USER CHOOSES ARTIST  
 if choice == 'Artist':  
 artist\_name\_text.text = music\_data.get\_name()  
 artist\_bio\_text.get\_lines(music\_data.find\_intro\_paragraphs())  
 img\_urls = music\_data.get\_album\_imgs()  
 image = Image.open("noAlbumFound.png")  
 for i in range(1, 4):  
 if i > 3:  
 image.save(f"album{i}.png")  
 for count, i in enumerate(img\_urls):  
 dataGathering.get\_image\_png(i, f"album{count + 1}.png")  
 top\_songs\_text.get\_lines(f"Top songs: {music\_data.get\_top\_song()}")  
 similar\_artists\_text.get\_lines(f"Similar Artists: {music\_data.similar\_artists()}")  
 ui\_state = 4  
 if choice == 'Song':  
 song\_bar\_charts = [BarChart(400 - offsetx, 225 - offsety, 75, 100, 'Acousticness', maxes[0],  
 music\_data.get\_acousticness()),  
 BarChart(400 - offsetx, 375, 75, 100, 'Speechiness', maxes[1],  
 music\_data.get\_speechiness()),  
 BarChart(545 - offsetx, 225 - offsety, 75, 100, 'Loudness', maxes[2],  
 music\_data.get\_loudness() + 60),  
 BarChart(545 - offsetx, 375, 75, 100, 'Tempo', maxes[3],  
 music\_data.get\_tempo()),  
 BarChart(690 - offsetx, 225 - offsety, 75, 100, 'Valence', maxes[4],  
 music\_data.get\_valence()),  
 BarChart(690 - offsetx, 375, 75, 100, 'Instrumentalness', maxes[5],  
 music\_data.get\_instrumentalness()),  
 BarChart(835 - offsetx, 225 - offsety, 75, 100, 'Danceability', maxes[6],  
 music\_data.get\_danceability()),  
 BarChart(835 - offsetx, 375, 75, 100, 'Energy', maxes[7],  
 music\_data.get\_energy())]  
 song\_name\_text.text = music\_data.get\_track\_name()  
 song\_artist\_text.text = music\_data.get\_artist\_name()  
 song\_info\_box.lines = music\_data.format\_string\_data()  
  
 try:  
 music\_data.get\_album\_image()  
 image = pygame.image.load("album1.png")  
 except:  
 image = pygame.image.load("noAlbumFound.png")  
  
 ui\_state = 5  
 # USER SETTINGS STATE  
 elif ui\_state == 2:  
 sign\_out\_button.handle\_event(event)  
 delete\_acc\_button.handle\_event(event)  
 user\_history.handle\_event(event)  
 password\_delete.handle\_event(event)  
 new\_change\_password.handle\_event(event)  
 change\_pass\_button.handle\_event(event)  
 old\_password.handle\_event(event)  
 clear\_history\_button.handle\_event(event)  
 if sign\_out\_button.pressed:  
 ui\_state = 0  
 if delete\_acc\_button.pressed:  
 df = login.Database("usernames.db")  
 if df.check\_credentials(username, password\_delete.user\_text):  
 password = password\_delete.user\_text  
 delete = True  
  
 else:  
 delete = False  
 result = 6  
 password\_delete.user\_text = ''  
 df.close\_database()  
 if change\_pass\_button.pressed:  
 if is\_password\_valid(new\_change\_password.user\_text):  
 df = login.Database("usernames.db")  
 if df.check\_credentials(username, old\_password.user\_text):  
 df.change\_password(username, new\_change\_password.user\_text)  
 df.close\_database()  
 old\_password.user\_text = ''  
 new\_change\_password.user\_text = ''  
 ui\_state = 0  
 result = 5  
 else:  
 result = 6  
 else:  
 result = 9  
 if clear\_history\_button.pressed:  
 df = login.Database("usernames.db")  
 df.update\_history(username, "")  
 user\_history.lines = [""]  
 df.close\_database()  
  
 result = 7  
 change = nav\_bar.handle\_event(event)  
 if change is not None:  
 ui\_state = change  
  
 if delete:  
 yes\_button.handle\_event(event)  
 no\_button.handle\_event(event)  
 if yes\_button.pressed:  
 df = login.Database("usernames.db")  
 df.delete\_user(username, password)  
 ui\_state = 0  
 result = 4  
 delete = False  
 df.close\_database()  
 yes\_button.pressed = False  
 if no\_button.pressed:  
 delete = False  
 no\_button.pressed = False  
  
 # ALBUM INFO  
 elif ui\_state == 3:  
 change = nav\_bar.handle\_event(event)  
 if change is not None:  
 ui\_state = change  
 album\_info\_box.handle\_event(event)  
 # ARTIST INFO  
 elif ui\_state == 4:  
 change = nav\_bar.handle\_event(event)  
 if change is not None:  
 ui\_state = change  
 artist\_bio\_text.handle\_event(event)  
 similar\_artists\_text.handle\_event(event)  
 top\_songs\_text.handle\_event(event)  
 # SONG INFO  
 elif ui\_state == 5:  
 change = nav\_bar.handle\_event(event)  
 if change is not None:  
 ui\_state = change  
  
 song\_info\_box.handle\_event(event)  
 # SIMILAR SONG  
 elif ui\_state == 6:  
 change = nav\_bar.handle\_event(event)  
 if change is not None:  
 ui\_state = change  
  
 similar\_text\_box.handle\_event(event)  
 similar\_enter\_box.handle\_event(event)  
 if similar\_enter\_box.pressed and len(similar\_text\_box.user\_text) != 0:  
 try:  
 similar\_songs = dataGathering.Song(similar\_text\_box.user\_text,"name").return\_similar()  
 for count in range(len(song\_boxes)):  
 current\_song = similar\_songs[count]  
 song\_boxes[count].song\_name.text = current\_song.get\_track\_name()  
 song\_boxes[count].artist\_name.text = current\_song.get\_artist\_name()  
 song\_boxes[count].image = pygame.image.load(f"album{count+1}.png")  
 song\_boxes[count].image = pygame.transform.scale(song\_boxes[count].image,(100,100))  
 song\_boxes[count].uri = current\_song.get\_track\_id()  
 ui\_state = 7  
 except:  
 result = 10  
 else:  
 if similar\_enter\_box.pressed:  
 result = 11  
  
 # SIMILAR RESULTS  
 elif ui\_state == 7:  
 change = nav\_bar.handle\_event(event)  
 if change is not None:  
 ui\_state = change  
 for song\_box in song\_boxes:  
 song\_box.handle\_event(event)  
 if song\_box.next\_button.pressed:  
 try:  
 music\_data = dataGathering.Song(song\_box.uri,"id")  
 song\_bar\_charts = [BarChart(400 - offsetx, 225 - offsety, 75, 100, 'Acousticness', maxes[0],  
 music\_data.get\_acousticness()),  
 BarChart(400 - offsetx, 375, 75, 100, 'Speechiness', maxes[1],  
 music\_data.get\_speechiness()),  
 BarChart(545 - offsetx, 225 - offsety, 75, 100, 'Loudness', maxes[2],  
 music\_data.get\_loudness() + 60),  
 BarChart(545 - offsetx, 375, 75, 100, 'Tempo', maxes[3],  
 music\_data.get\_tempo()),  
 BarChart(690 - offsetx, 225 - offsety, 75, 100, 'Valence', maxes[4],  
 music\_data.get\_valence()),  
 BarChart(690 - offsetx, 375, 75, 100, 'Instrumentalness', maxes[5],  
 music\_data.get\_instrumentalness()),  
 BarChart(835 - offsetx, 225 - offsety, 75, 100, 'Danceability', maxes[6],  
 music\_data.get\_danceability()),  
 BarChart(835 - offsetx, 375, 75, 100, 'Energy', maxes[7],  
 music\_data.get\_energy())]  
 song\_name\_text.text = music\_data.get\_track\_name()  
 song\_artist\_text.text = music\_data.get\_artist\_name()  
 song\_info\_box.lines = music\_data.format\_string\_data()  
  
 try:  
 music\_data.get\_album\_image()  
 image = pygame.image.load("album1.png")  
 except:  
 image = pygame.image.load("noAlbumFound.png")  
  
 ui\_state = 5  
 except: result = 10  
  
  
 # -------------------------------------------  
  
 screen.fill((125, 125, 125))  
  
 # ------------------------------------------  
 # LOGIN STATE  
 if ui\_state == 0:  
 password\_box.render\_text()  
 username\_box.render\_text()  
 login\_box.render\_button()  
 sign\_up\_box.render\_button()  
  
  
 # ENTER MUSIC SEARCH STATE  
 elif ui\_state == 1:  
 pygame.draw.rect(screen, (100, 100, 100), (0, 0, 960, 40))  
  
 box.render\_text()  
 dropdown.render\_button()  
 enter\_box.render\_button()  
 for i in nav\_bar.buttons:  
 i.colour\_change = False  
 i.colour = (100, 100, 100)  
 nav\_bar.buttons[0].colour = (125, 125, 125)  
 nav\_bar.render\_buttons()  
  
 # ENTER USER SETTINGS STATE  
 elif ui\_state == 2:  
 pygame.draw.rect(screen, (100, 100, 100), (0, 0, 960, 40))  
 clear\_history\_button.render\_button()  
 sign\_out\_button.render\_button()  
 delete\_acc\_button.render\_button()  
 username\_text.render\_text()  
 password\_delete.render\_text()  
 new\_change\_password.render\_text()  
 change\_pass\_button.render\_button()  
 old\_password.render\_text()  
 user\_history\_text.render\_text()  
 user\_history.render()  
  
 if delete:  
 pygame.draw.rect(screen, (150, 150, 150), (250, 250, 250, 250))  
 Text(250, 250, "Are you sure?", 40, 0, 0).render\_text()  
 yes\_button.render\_button()  
 no\_button.render\_button()  
  
 for i in nav\_bar.buttons:  
 i.colour\_change = False  
 i.colour = (100, 100, 100)  
 nav\_bar.buttons[1].colour = (125, 125, 125)  
 nav\_bar.render\_buttons()  
  
 # ALBUM INFO STATE  
 elif ui\_state == 3:  
 pygame.draw.rect(screen, (100, 100, 100), (0, 0, 960, 40))  
 for i in nav\_bar.buttons:  
 i.colour\_change = False  
 i.colour = (100, 100, 100)  
 nav\_bar.buttons[0].colour = (125, 125, 125)  
 nav\_bar.render\_buttons()  
 for i in bar\_charts:  
 i.render()  
  
 album\_name\_text.text, album\_artist\_text.text = album\_name, artist\_name  
 album\_name\_text.render\_text()  
 album\_artist\_text.render\_text()  
  
 pygame.draw.rect(screen, (100, 100, 100), (45, 45, 260, 260))  
 image = pygame.image.load("image.png")  
 image = pygame.transform.scale(image, (250, 250))  
 screen.blit(image, (50, 50))  
 album\_info\_box.render()  
 # artist search screen  
 elif ui\_state == 4:  
 pygame.draw.rect(screen, (100, 100, 100), (0, 0, 960, 40))  
 for i in nav\_bar.buttons:  
 i.colour\_change = False  
 i.colour = (100, 100, 100)  
 nav\_bar.buttons[0].colour = (125, 125, 125)  
 nav\_bar.render\_buttons()  
  
 artist\_bio\_text.render()  
 artist\_name\_text.render\_text()  
  
 pygame.draw.rect(screen, (100, 100, 100), (15, 55, 155, 455))  
  
 album1\_img = pygame.image.load("album1.png")  
 album2\_img = pygame.image.load("album2.png")  
 album3\_img = pygame.image.load("album3.png")  
 album1\_img = pygame.transform.scale(album1\_img, (145, 145))  
 album2\_img = pygame.transform.scale(album2\_img, (145, 145))  
 album3\_img = pygame.transform.scale(album3\_img, (145, 145))  
 screen.blit(album1\_img, (20, 60))  
 screen.blit(album2\_img, (20, 210))  
 screen.blit(album3\_img, (20, 360))  
  
 top\_songs\_text.render()  
 similar\_artists\_text.render()  
  
 elif ui\_state == 5:  
 pygame.draw.rect(screen, (100, 100, 100), (0, 0, 960, 40))  
 for i in nav\_bar.buttons:  
 i.colour\_change = False  
 i.colour = (100, 100, 100)  
 nav\_bar.buttons[0].colour = (125, 125, 125)  
 nav\_bar.render\_buttons()  
  
 song\_name\_text.render\_text()  
 song\_artist\_text.render\_text()  
 for i in song\_bar\_charts:  
 i.render()  
  
 pygame.draw.rect(screen, (100, 100, 100), (45, 45, 210, 210))  
 image = pygame.transform.scale(image, (200, 200))  
 screen.blit(image, (50, 50))  
 song\_info\_box.render()  
  
 # FIND SIMILAR  
 elif ui\_state == 6:  
 pygame.draw.rect(screen, (100, 100, 100), (0, 0, 960, 40))  
 for i in nav\_bar.buttons:  
 i.colour\_change = False  
 i.colour = (100, 100, 100)  
 nav\_bar.buttons[2].colour = (125, 125, 125)  
 nav\_bar.render\_buttons()  
  
 similar\_enter\_box.render\_button()  
 similar\_text\_box.render\_text()  
  
 elif ui\_state == 7:  
 pygame.draw.rect(screen, (100, 100, 100), (0, 0, 960, 40))  
 for i in nav\_bar.buttons:  
 i.colour\_change = False  
 i.colour = (100, 100, 100)  
 nav\_bar.buttons[2].colour = (125, 125, 125)  
 nav\_bar.render\_buttons()  
 for song\_box in song\_boxes:  
 song\_box.render()  
 elif ui\_state == 8:  
 screen.fill((100, 100, 100))  
 not\_connected.render\_text()  
 try:  
 requests.get("http://www.google.com")  
 ui\_state = 0  
 except:  
 time.sleep(1)  
  
 # Flash Text On Screen  
  
 if result != 0 and login\_result\_timer < 60:  
 match result:  
 case 1:  
 display\_text = "Incorrect Username or Password"  
 case 2:  
 display\_text = "Already Existing User"  
 case 3:  
 display\_text = "User Added"  
 case 4:  
 display\_text = "User Deleted"  
 case 5:  
 display\_text = "Changed Password"  
 case 6:  
 display\_text = "Incorrect Password"  
 case 7:  
 display\_text = "Cleared History"  
 case 8:  
 display\_text = "Username must contain valid character and atleast 4 characters"  
 case 9:  
 display\_text = "Password must contain valid character and atleast 4 characters"  
 case 10:  
 display\_text = "Error finding music"  
 case 11:  
 display\_text = "Must Enter Song"  
 case 12:  
 display\_text = "Must Enter characters into the text box"  
 case \_:  
 display\_text = ""  
 text\_surface = base\_font.render(display\_text, True, (255, 255, 255))  
  
 screen.blit(text\_surface, (280, 375))  
 login\_result\_timer += 1  
 if login\_result\_timer == 60:  
 login\_result\_timer = 0  
 result = 0  
  
 # -------------------------------------------  
  
 pygame.display.flip()  
 clock.tick(60)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 run()

### Login.py

import sqlite3  
import hashlib  
  
class Database:  
 def \_\_init\_\_(self, filename):  
 self.datab = sqlite3.connect(filename)  
  
 self.cursor = self.datab.cursor()  
  
 def add\_user(self, username, password):  
 if not self.check\_repeat(username):  
 password\_hash = hashlib.sha256(password.encode()).hexdigest()  
  
 self.cursor.execute("SELECT MAX(id) FROM users")  
 max\_id = self.cursor.fetchone()[0]  
  
 try:  
 self.cursor.execute("INSERT INTO users (id, name, password, user\_history) VALUES (?,?,?,?)", (max\_id + 1, username, password\_hash,''))  
 except:  
 self.cursor.execute("INSERT INTO users (id, name, password, user\_history) VALUES (?,?,?,?)",  
 (1, username, password\_hash,''))  
 self.datab.commit()  
 print('user added')  
 else:  
 print('repeat detected')  
  
 def check\_repeat(self, username):  
 result = self.cursor.execute("SELECT 1 FROM users WHERE name = ?", (username,))  
  
 if result.fetchone():  
 return True  
 else:  
 return False  
  
 def check\_credentials(self, username, password):  
 password\_hash = hashlib.sha256(password.encode()).hexdigest()  
  
 result = self.cursor.execute(  
 "SELECT \* FROM users WHERE name = ? AND password = ?",  
 (username, password\_hash))  
  
 if result.fetchone():  
 return True  
 else:  
 return False  
  
 def delete\_user(self, username, password):  
 password\_hash = hashlib.sha256(password.encode()).hexdigest()  
  
 result = self.datab.execute(  
 "SELECT \* FROM users WHERE name = ? AND password = ?",  
 (username, password\_hash))  
  
 user = result.fetchone()  
 if user:  
 user\_id = user[0]  
 print(user\_id)  
 self.cursor.execute("DELETE FROM users WHERE name=? AND password=?", (username, password\_hash))  
  
 self.datab.execute("UPDATE users SET id = id - 1 WHERE id > ?", (user\_id,))  
 self.datab.commit()  
 print(f'User {username} deleted')  
 else:  
 print('User not found')  
  
 def fetch\_history(self, username):  
 self.cursor.execute('SELECT user\_history FROM users WHERE name=?', (username,))  
  
 result = self.cursor.fetchone()  
  
 if result:  
 return result[0]  
 else:  
 return None  
  
 def update\_history(self, username, new\_history):  
 self.cursor.execute('UPDATE users SET user\_history=? WHERE name=?', (new\_history, username))  
  
 self.datab.commit()  
  
 def change\_password(self, username, new\_password):  
 # Hash the new password  
 new\_password\_hash = hashlib.sha256(new\_password.encode()).hexdigest()  
  
 self.cursor.execute('UPDATE users SET password=? WHERE name=?', (new\_password\_hash, username))  
  
 self.datab.commit()  
  
 def close\_database(self):  
 self.datab.close()

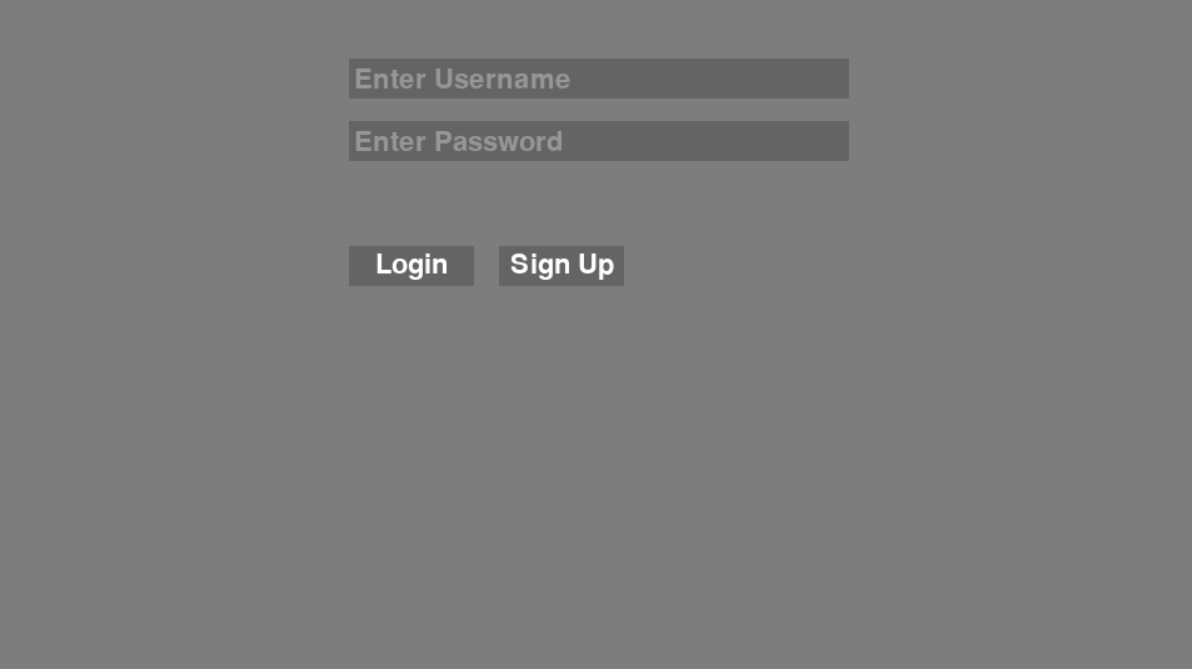
### Hashing.py

import math  
  
  
def get\_hash(arr):  
 result = []  
 for value in arr:  
 temp = int(value \* 5)  
 if temp == 5: temp = 4  
 result.append(temp)  
 print(result)  
 result.reverse()  
 result2 = 0  
 for count, value in enumerate(result):  
 result2 += value \* (5 \*\* count)  
 return result2  
  
  
  
def find\_neighbours(value, n):  
 neighbours = {value}  
 for nth\_neighbour in range(n):  
 new\_neighbours = set()  
 for neighbour in neighbours:  
 for i in range(10):  
 low\_neighbour = neighbour + (5 \*\* i)  
 high\_neighbour = neighbour - (5 \*\* i)  
 if low\_neighbour not in new\_neighbours: new\_neighbours.add(low\_neighbour)  
 if high\_neighbour not in new\_neighbours:new\_neighbours.add(high\_neighbour)  
 neighbours = new\_neighbours  
 neighbours = sorted(neighbours)  
 return neighbours  
  
  
def reverse(n):  
 digits = []  
 while n > 0:  
 remainder = n % 5  
 digits.append(int(remainder))  
 n //= 5  
 while len(digits) < 10:  
 digits.append(0)  
 digits.reverse()  
 return digits  
  
  
def find\_distance(hash1, hash2):  
 arr1 = reverse(hash1)  
 arr2 = reverse(hash2)  
 distance = 0  
 for i in range(10):  
 distance += (abs(abs(arr1[i]) - abs(arr2[i]))) \*\* 2  
 distance = math.sqrt(distance)  
 similar = (4 \* math.sqrt(10) - distance) / (4 \* math.sqrt(10))  
 return distance, similar

### DataSetCreation3.py

import csv  
import json  
import time  
  
import spotipy  
import os  
import hashing  
from spotipy.oauth2 import SpotifyClientCredentials  
  
os.environ['SPOTIPY\_CLIENT\_ID'] = '3887776ea50a469b81ab216075517bcf'  
os.environ['SPOTIPY\_CLIENT\_SECRET'] = '52bf4482e5ec4256b23b5e6ef6c49a6f'  
  
  
def load\_json(file\_path):  
 with open(file\_path, 'r') as f:  
 data = json.load(f)  
 return data  
  
  
def process\_playlist(tracks, csv\_path):  
 sp = spotipy.Spotify(client\_credentials\_manager=SpotifyClientCredentials())  
 with open(csv\_path, 'a', newline='') as f:  
 writer = csv.writer(f)  
 # Load existing entries into a set  
 existing\_entries = set()  
 if os.path.exists(csv\_path):  
 with open(csv\_path, 'r', newline='') as csv\_file:  
 reader = csv.reader(csv\_file)  
 next(reader) # Skip header row  
 for row in reader:  
 existing\_entries.add(row[0])  
 for track in tracks:  
 try:  
 song\_uri = track["track\_uri"]  
 # Check if entry already exists  
 if song\_uri not in existing\_entries:  
 # Add track ID to existing entries  
 existing\_entries.add(song\_uri)  
 start = time.time()  
 audio\_features = sp.audio\_features(song\_uri)[0]  
 arr = [  
 audio\_features['danceability'],  
 audio\_features['energy'],  
 (audio\_features['key'] / 12),  
 ((audio\_features['loudness'] + 60) / 60),  
 audio\_features['speechiness'],  
 audio\_features['acousticness'],  
 audio\_features['instrumentalness'],  
 audio\_features['liveness'],  
 audio\_features['valence'],  
 (audio\_features['tempo'] / 300)  
 ]  
 song\_hash = hashing.get\_hash(arr)  
 writer.writerow([song\_uri, song\_hash])  
 f.flush()  
 end = time.time()  
 print(f"Processed track with URI {song\_uri} in playlist, hash is {song\_hash}, time taken is {end-start}s")  
 except:  
 print('sleeping')  
 time.sleep(60)  
  
 # sort the csv by hash  
 with open(csv\_path, 'r', newline='') as f:  
 reader = csv.reader(f)  
 sorted\_rows = sorted(reader, key=lambda x: x[1])  
 with open(csv\_path, 'w', newline='') as f:  
 writer = csv.writer(f)  
 writer.writerow(['track\_uri', 'hash'])  
 writer.writerows(sorted\_rows)  
  
  
def main(start\_index=0):  
 directory = "D:\Desktop\COMSPCI NEA\desktop\data"  
 for filename in os.listdir(directory):  
 if filename.endswith(".json"):  
 print(filename)  
 file\_path = os.path.join(directory, filename)  
 with open(file\_path, "r") as f:  
 data = json.load(f)  
 playlists = data['playlists']  
 for i in range(start\_index, len(playlists)):  
 playlist = playlists[i]  
 tracks = playlist['tracks']  
 process\_playlist(tracks, "output.csv")  
 print("Processing complete!")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main(0)

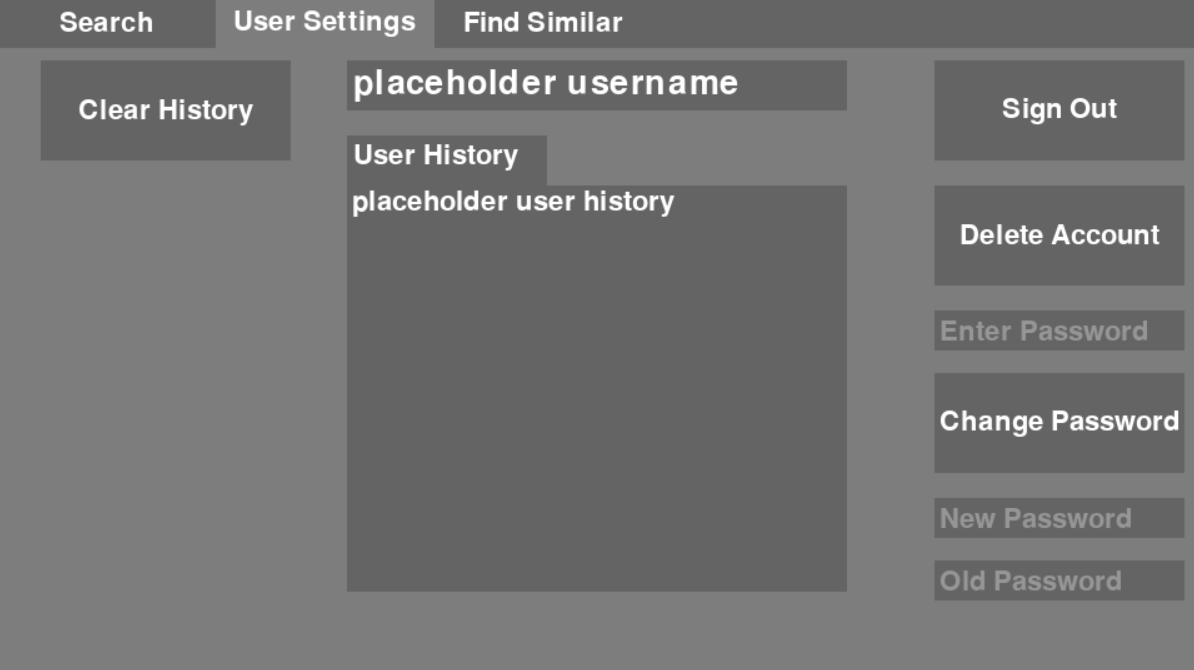
### UI State 0 – Login Screen



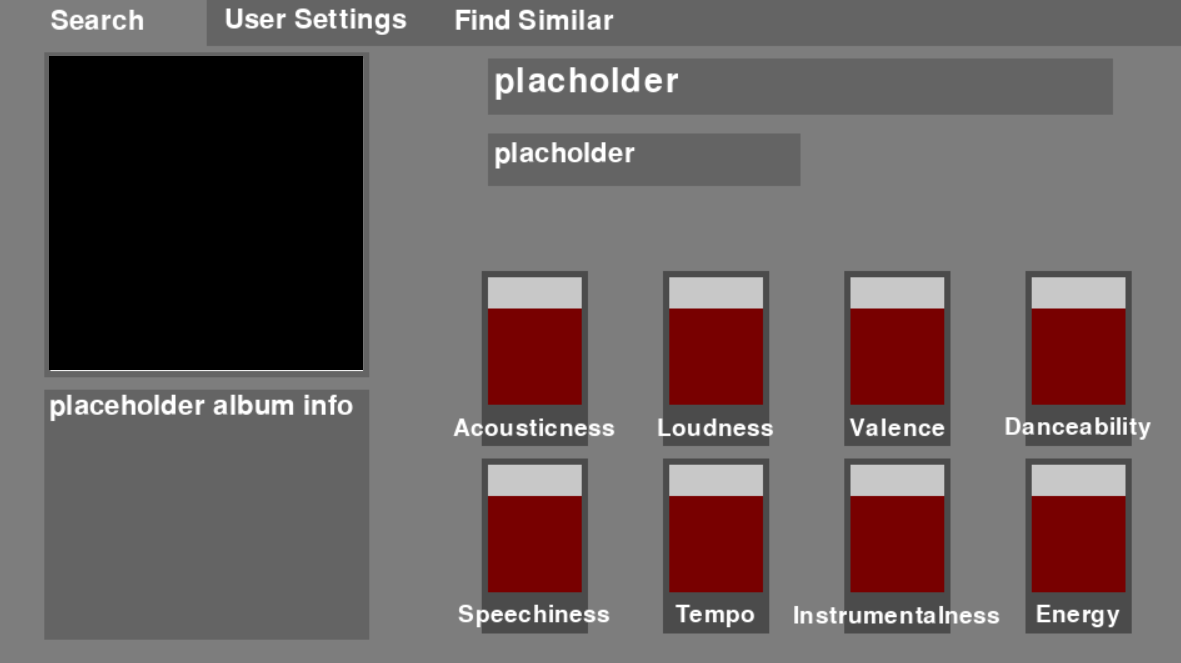
### UI State 1 – Song Search



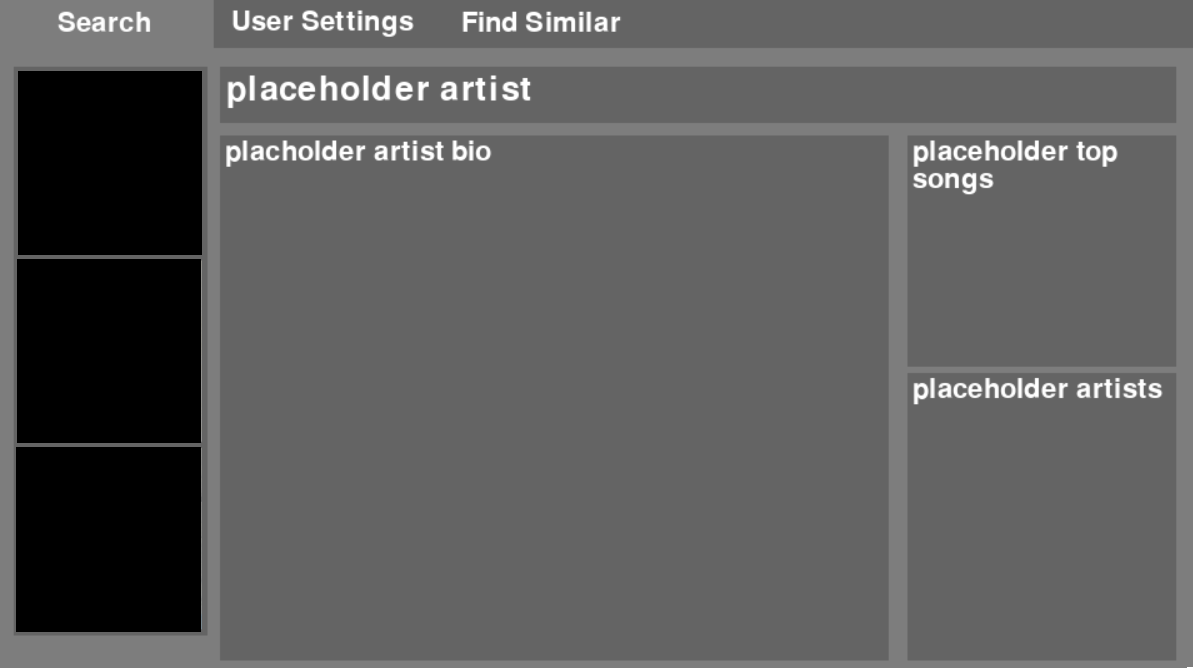
### UI State 2 – User Settings



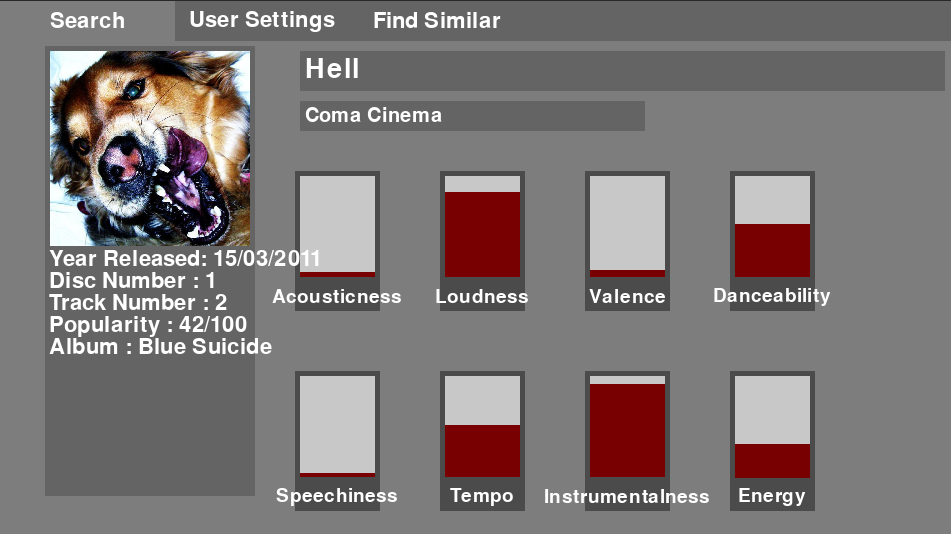
### UI State 3 – Album Info



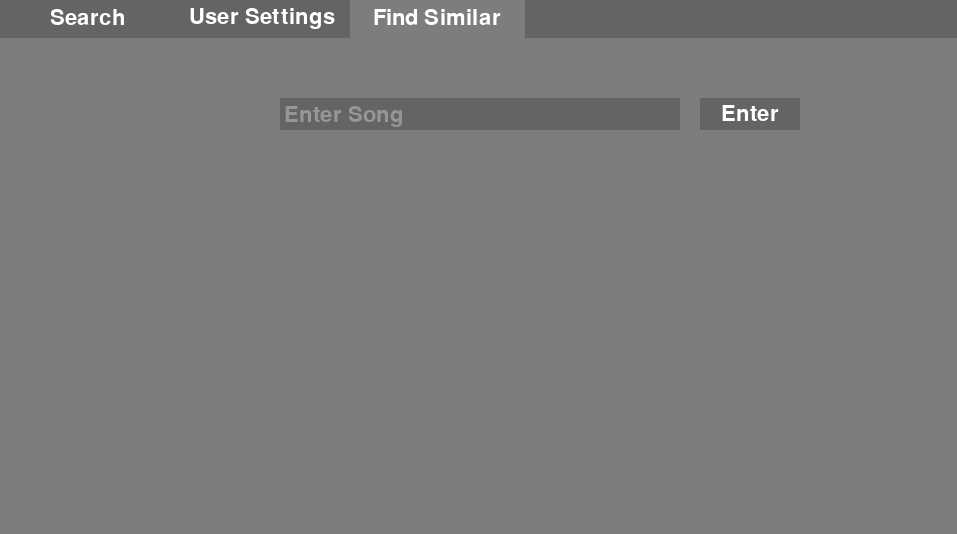
### UI State 4 – Artist Info



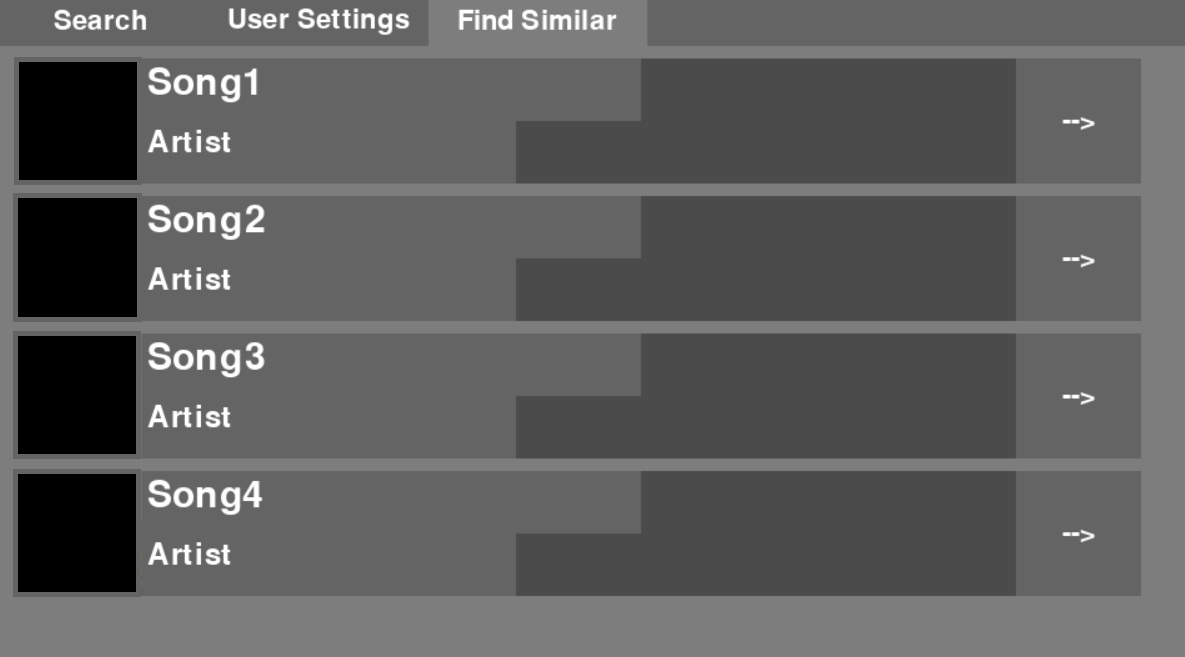
### UI State 5 -Song Info



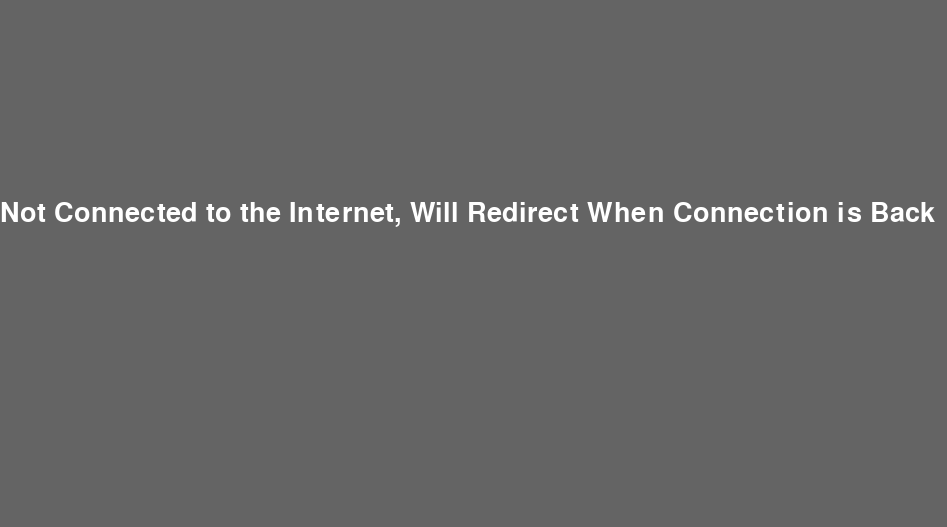
### UI State 6 – Find Similar Song



### UI State 7 – Similar Song Results



### UI State 8 – No Internet



## Testing

### Introduction

To ensure that my code is robust, I developed a comprehensive test plan. This plan covers the following areas:

* The registration process
* The login process
* The user entering a song, album or artist and initiating the search
* The code working under conditions where there is not access to Wikipedia or Spotify

For the elements of code that require a user to input data, including their user information and the name of the song, album, or artist that they are interested in, I ensured that the test plan covered the following kinds of input data:

* Normal data (e.g. the name of an artist), used in Test IDs 1,2,11,12,17-25,27,28,32 & 33
* Erroneous data (e.g. blank entries, special characters where normal text is expected), used in Test IDs 4-8, 10, 12-15 and 16
* Boundary data (e.g. passwords that do not meet the minimum requirements for length – namely four characters or more), used in Test IDs 3, 9, and 30

### Test Plan

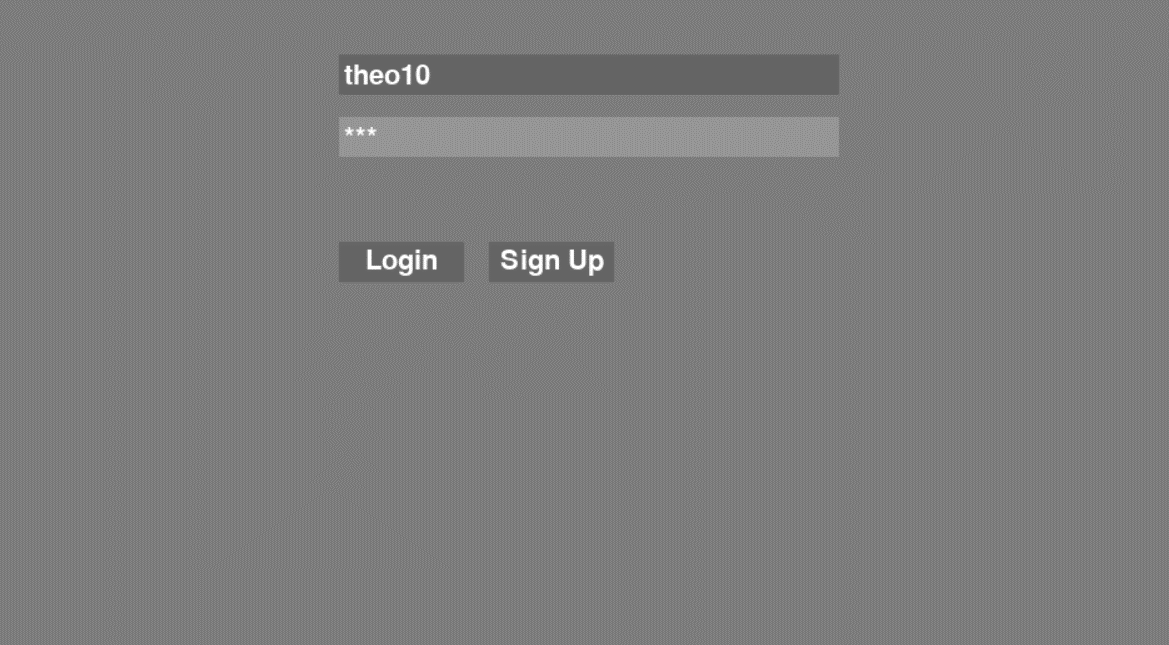
The table below shows that unique tests that the code must pass in order for me to be confident that it is robust.

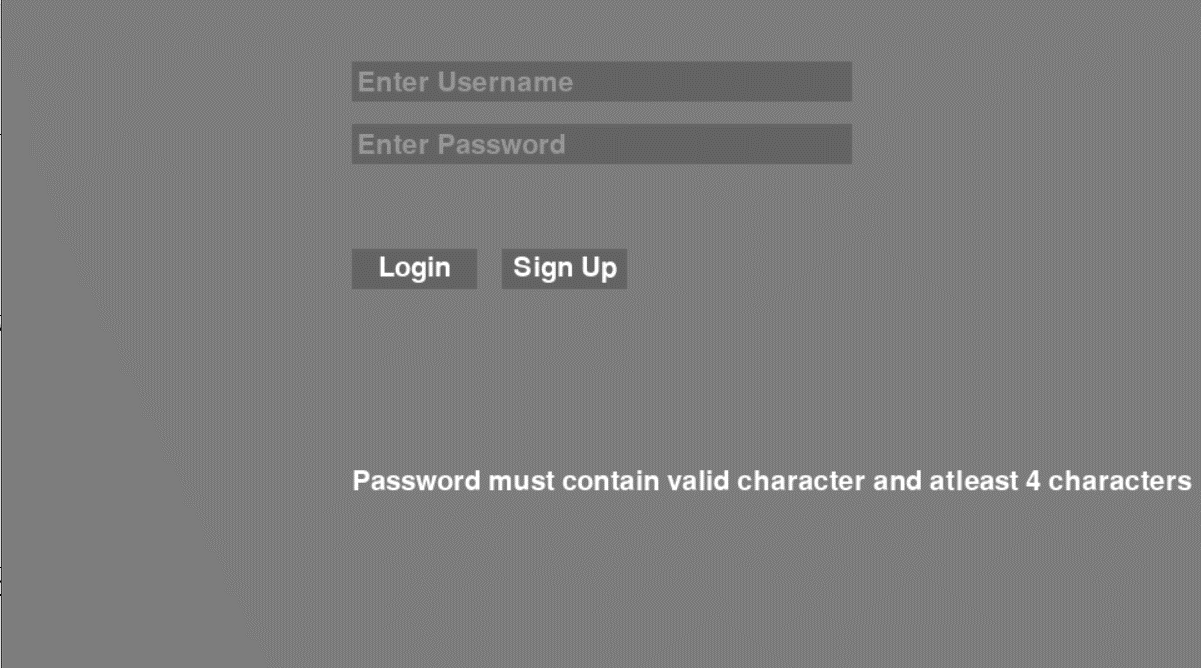
| **Test ID** | **Description** | **Test Data** | **Expected Results** | **Test done successfully and Time Stamp** |
| --- | --- | --- | --- | --- |
| New user registration | | | | |
| 1 | Check that a new user can register with normal data – both normal username and normal password | Name: ‘Jacob17’  Password: ‘H3ll0’  Click ‘Sign Up’ | 1. Message telling the user that they have been registered 2. User data added to the database of users | Yes  00:15 |
| 2 | Check that a new user cannot use the same username as an existing user | Name: ‘Jacob17’  Password: ‘G00dby3’  Click ‘Sign Up’ | 1. Error message telling user that there is already a user of that name and asking for another username 2. No user data added to the database | Yes  00:28 |
| 3 | Check that a new user cannot use a username with fewer than four characters | Name: ‘B0b’  Password: ‘Hell0'  Click ‘Sign Up’ | 1. Error message telling the user that their proposed username is invalid and explaining what the validation rules are 2. No user data added to the database | Yes  00:37 |
| 4 | Check that a new user cannot register using a null username | Name: <null>  Password: Hell0  Click ‘Sign Up’ | Same as test ID 3 | Yes  00:46 |
| 5 | Check that a new user cannot use a username with a special character | Name: #478783 or  Password: S33Y0u  Click ‘Sign Up’ | Same as test ID 3 | Yes  1:05 |
| 6 | Check that a new user cannot use a username with a space | Name: ‘Hi Th3r3’  Password: ‘S33Y0u’  Click ‘Sign Up’ | Same as test ID 3 | Yes  1:16 |
| 7 | Check that a new user cannot use a null password | Name: ‘Jackie3’  Password: <null>  Click ‘Sign Up’ | 1. Error message telling the user that their proposed password is invalid and explaining what the validation rules are 2. No user data added to the database | Yes  1:28 |
| 8 | Check that a new user cannot use a password involving special characters | Name: ‘Marly01’  Password: ‘#92kl’  Click ‘Sign Up’ | Same as test ID 7 | Yes  1:42 |
| 9 | Check that a new user cannot use a password with insufficient characters | Name: ‘Theo10’  Password: ‘B3y’  Click ‘Sign Up’ | Same as test ID 7 | Yes  SEE SCREENSHOT BELOW |
| 10 | Check that a new user cannot use a password with spaces | Name: ‘Theo10’  Password: ‘H1 Th3r3’  Click ‘Sign Up’ | Same as test ID 7 | Yes  1:53 |
| Login | | | | |
| 11 | Check that a user can login with normal data – both normal username and normal password, and a user that has already registered | Name: ‘Jacob17’  Password: H3ll0  Click ‘Login’ | 1. Closing of the login UI object 2. Opening of the Search window | Yes  2:09 |
| 12 | Check that a user cannot log in if there is no one with that username | Name: ‘Christian93’  Password: ‘Lat3rM8’  Click ‘Login’ | 1. Error message telling the user that their username and/or password cannot be found | Yes  2:22 |
| 13 | Check that a user cannot login with a username that includes special characters | Name: ‘#ksfuio1934’  Password: ‘T3st17’  Click ‘Login’ | Same as test ID 12 | Yes  2:40 |
| 14 | Check that a user cannot login with a password that contains special characters | Name: ‘Jacob17’  Password: ‘#ksfuio1934’  Click ‘Login’ | Same as test ID 12 | Yes  3:01 |
| 15 | Check that a user cannot login with a <null> username | Name: <null>  Password: ‘Hell0’  Click ‘Login’ | Same as test ID 12 | Yes  3:12 |
| 16 | Check that a user cannot login with a <null> password | Name: ‘Jacob17’  Password: <null>  Click ‘Login’ | Same as test ID 12 | Yes  3:19 |
| Search – Song/Artist/Album | | | | |
| 17 | Check that the user can get information with a real song | Toggle: ‘Song’  Query: ‘Blind’  Click on ‘Enter’ | 1. Information for that song is displayed 2. The search for that song is added to the search history in the database | Yes  3:30 |
| 18 | Check that the user is informed if no song with that title can be found | Toggle: ‘Song’  Query: Random Characters  Click on ‘Enter’ | 1. An error message is displayed telling the user that no song with that title was found | Yes  3:45 |
| 19 | Check that the user can get information with a real artist | Toggle: ‘Artist’  Query: ‘Panchiko’  Click on ‘Enter’ | 1. Information for that artist is displayed 2. The search for that artist is added to the search history in the database | Yes  4:00 |
| 20 | Check that the user can get information with a real album | Toggle: ‘Album’  Query: ‘To be kind’  Click on ‘Enter’ | 1. Information for that album is displayed 2. The search for that album is added to the search history in the database | Yes  4:25 |
| 21 | Check that the user is informed if no album with that title can be found | Toggle: ‘Album’  Query: Random Characters  Click on ‘Enter’ | 1. An error message is displayed telling the user that no album with that name was found | Yes  4:36 |
| User Settings (logged on as ‘Jacob17’) | | | | |
| 22 | Check that the user cannot delete their account if they do not enter the correct password | Enter Password: ‘Hello’ (note that this is incorrect)  Click on ‘Delete Account’ | 1. A message is displayed telling the user that their password is incorrect 2. The account is NOT deleted from the database of users | Yes  4:59 |
| 23 | Check that the user can change their mind about deleting the account if they are not sure they want to | Click on ‘Delete Account’  Click on ‘No’ to show that the user is NOT sure they want to proceed | 1. The User Settings window is displayed 2. The account is NOT deleted from the database of users | Yes  5:09 |
| 24 | Check that the user can delete their account | Click on ‘Delete Account’  Enter Password: ‘H3ll0’  Click on ‘Yes’ to confirm that the user is sure they want to proceed  Note: after test 24, is completed successfully, redo Test 1 to re-create the user ‘Jacob17’ with password ‘H3ll0’, and Tests 17, 19, and 21 to create a search history | 1. A message is displayed telling the user that their account is now deleted 2. The Login/Sign Up window is displayed 3. The account is deleted from the database of users | Yes  5:30 |
| 25 | Check that ‘Clear History’ works | Click on ‘Clear History’ | 1. The ‘User History’ is cleared for the User Settings 2. User history is cleared in the database | Yes  6:23 |
| 26 | Check that the user can change their password | New Password: ‘H1Th3r3’  Old Password: ‘H3ll0’  Click on ‘Change Password’ | 1. A message is displayed telling the user that their password has now been changed 2. The new password replaces the old password in the database | Yes  6:38 |
| 27 | Check that the user cannot change their password if they do not know their old password (however the password is alphanumeric) | New Password: ‘G00dby3’  Old Password: ‘HiThere’  Click on ‘Change Password’ | 1. A message is displayed telling the user that their password has not been changed as their old password is incorrect 2. The old password in the database is not changed | Yes  7:16 |
| 28 | Check that the user cannot change their password if they use <null> as the old password | New Password: ‘G00dby3’  Old Password: <null>  Click on ‘Change Password’ | Same as test ID 27 | Yes  7:22 |
| 29 | Check that the user cannot change their password if new password has fewer than four characters | New Password: ‘By3’  Old Password: ‘H1Th3r3’  Click on ‘Change Password’ | 1. A message is displayed telling the user that their password has not been changed as their new password does not meet the password rules 2. The old password in the database is not changed | Yes  7:44 |
| 30 | Check that the user cannot change their password if new password contains special characters | New Password: ‘#S33Y0u’  Old Password: ‘H1Th3r3’  Click on ‘Change Password’ | Same as test ID 29 | Yes  7:53 |
| Find Similar | | | | |
| 31 | Check that the user can find similar songs if they enter a real song | Enter Song: ‘Blind’  Click on ‘Enter’ | 1. A list of similar songs is displayed 2. The search for that similar song is added to the search history in the database | Yes  8:10 |
| 32 | Check that the user is informed if the song title is not found | Enter Song: Random Characters  Click on ‘Enter’ | 1. A message is displayed telling the user that the song could not be found | Yes  8:19 |
| Internet access | | | | |
| 33 | Check that user is informed if there is no internet access | Start programme with no internet | 1. A message is displayed telling the user informing them that there is no internet access, and they will be redirected when internet access is established 2. The program checks on a regular basis to see if internet connection has been established 3. When it has been established, the program goes to the log-in window | Yes  8:39, reconnected at 8:46 |

### Test Results

The above tests were all performed successfully. I have taken a video to provide evidence of this, which is available on YouTube under the following url: [NEA Testing - YouTube](https://www.youtube.com/watch?v=b0QgS5vMWbs)

Forgot to include test ID 9:





## Evaluation

### Outcome vs. objectives

The table below shows the outcome of my project against the objectives.

| **Objective** | **Was the outcome achieved** |
| --- | --- |
| 1. New users must be able to register their username and password | |
| a.       New users must be able to select a username and password | Yes |
| b.      The username and/or password cannot be blank | Yes |
| c.       The username and password must both be a minimum of four characters | Yes |
| d.      The username and/or password cannot include special characters (i.e. only letters and numbers are allowed) | Yes |
| e.      The username must be unique | Yes |
| f.        If a username or password is invalid, the user must be informed | Yes |
| g.       The password must be stored in an encrypted way | Yes |
| 2.      Users must be able to log in | |
| a.       Users must be able to log in based on a username and password | Yes |
| b.      If a username cannot be found, the user must be informed | Yes |
| 3.      Users must be able to change their password | |
| a.       Users must be able to change their password | Yes |
| b.      Users must only be able to change their password if they know their old password | Yes |
| c.       New passwords must not be blank | Yes |
| d.      New passwords must be at least four characters | Yes |
| e.      New passwords must not contain any special characters (i.e. only letters and numbers are allowed) | Yes |
| f.        If a new password is invalid, the user must be informed | Yes |
| 4.      Users must be able to search for information about a song/artist/album | |
| a.       The user must be able to select if they would like to search for either a song, artist, or album | Yes – however, since I am collecting information mainly from Spotify, except for artist information which also includes information from Wikipedia, this is not information that would be hard for users to find for themselves in another way e.g. directly from Spotify |
| b.       The name of the song, artist, or album cannot be blank | Yes |
| c.       If the name of the song, artist, or album cannot be found, the user must be informed | Yes – however, since I am using the Spotify search algorithm, this tends to generate *some* results even if there is not even a close match.  It is very rare for nothing to be found, and this can mean that the information displayed is not what the user wanted. |
| d.       The user must be able to see the information retrieved by the programme | Yes |
| e.       The user must be able to continue to look at the information until they are ready to move on | Yes |
| 5.      Users must be able to get suggestions for other songs they might like, based on a song they do like | |
| a.       The user must be able to input the name of a song | Yes |
| b.       The name of the song cannot be blank | Yes |
| c.       If the name of the song cannot be found, the user must be informed | Yes |
| d.       The user must be able to see four song suggestions, based on the name of a song that they input | Yes, however there are some limitations to the song suggestions. This is discussed in more detail below. |
| e.       The user must be able to continue to look at the suggested songs until they are ready to move on | Yes |
| 6.      Users must be able to review their search history | |
| a.       Users must be able to review their previous searches (whether for information about songs/artists/albums, or suggestions for similar songs) | Yes |
| b.       Users must be able to clear their search history | Yes |
| 7.      General | |
| a.       Users must be able to quit the programme | Yes |
| b. Users must be informed if the programme cannot establish connection with the internet | Yes, however this is only checked the first time they open the program. If they lose internet once it is running, users are not informed. |

I consider that the objectives of my programme have been mainly achieved. There are some limitations:

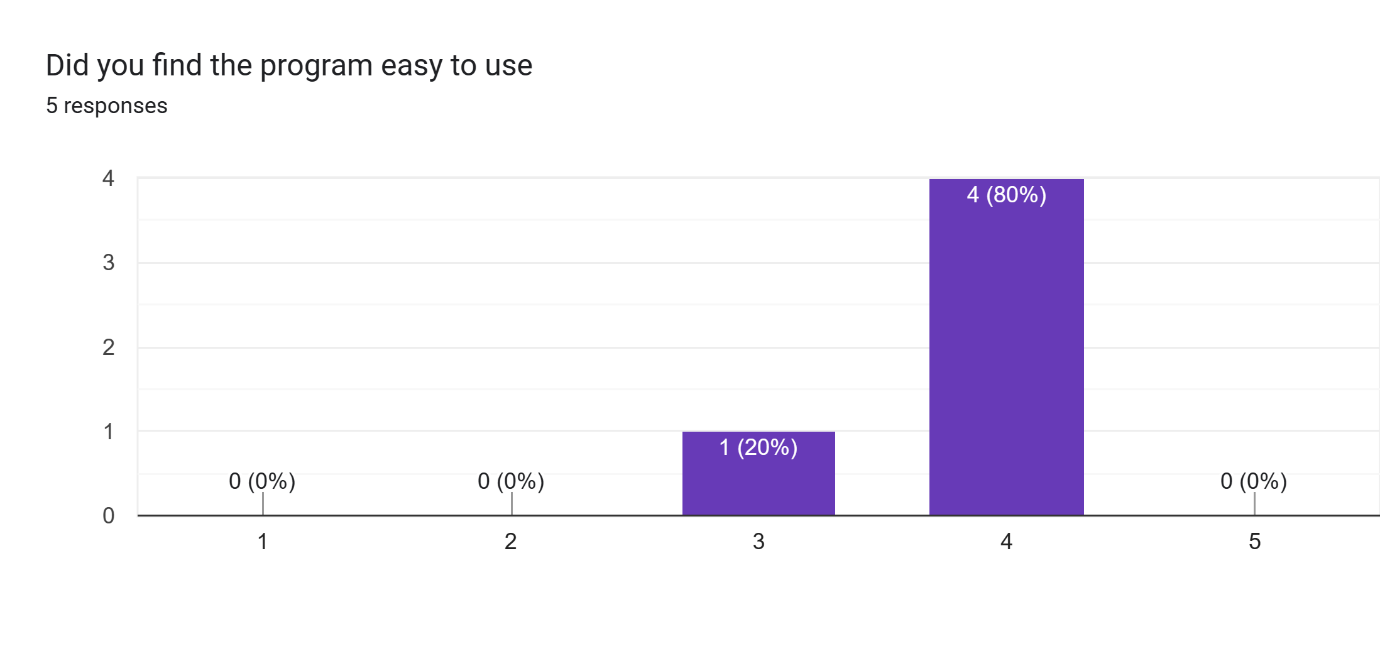
1. The song database used to find similar songs contains 250,000 songs. This is actually quite a small number of songs, and this limits the ability of users to find the best suggestions. I did not collect a larger database of songs as this would have taken a long time.
2. The song suggestion algorithm I used has its own limitations. In particular:
   1. Just because two songs share many attributes doesn’t mean the same people who like one song will always like the other.
   2. If a song is quite unusual, the algorithm can take a long time to run (up to 1 minute). Many users will be too impatient to wait for this and might stop using the program because they are frustrated.
3. The artist information, which comes from Wikipedia, takes information from the first Wikipedia page related to the name of the artist. If there are a number of famous people with the same name as the artist, the information shown might be about someone else.
4. The programme is not completely robust. If the user loses access to the internet in the middle of using it, it will crash.

### Feedback

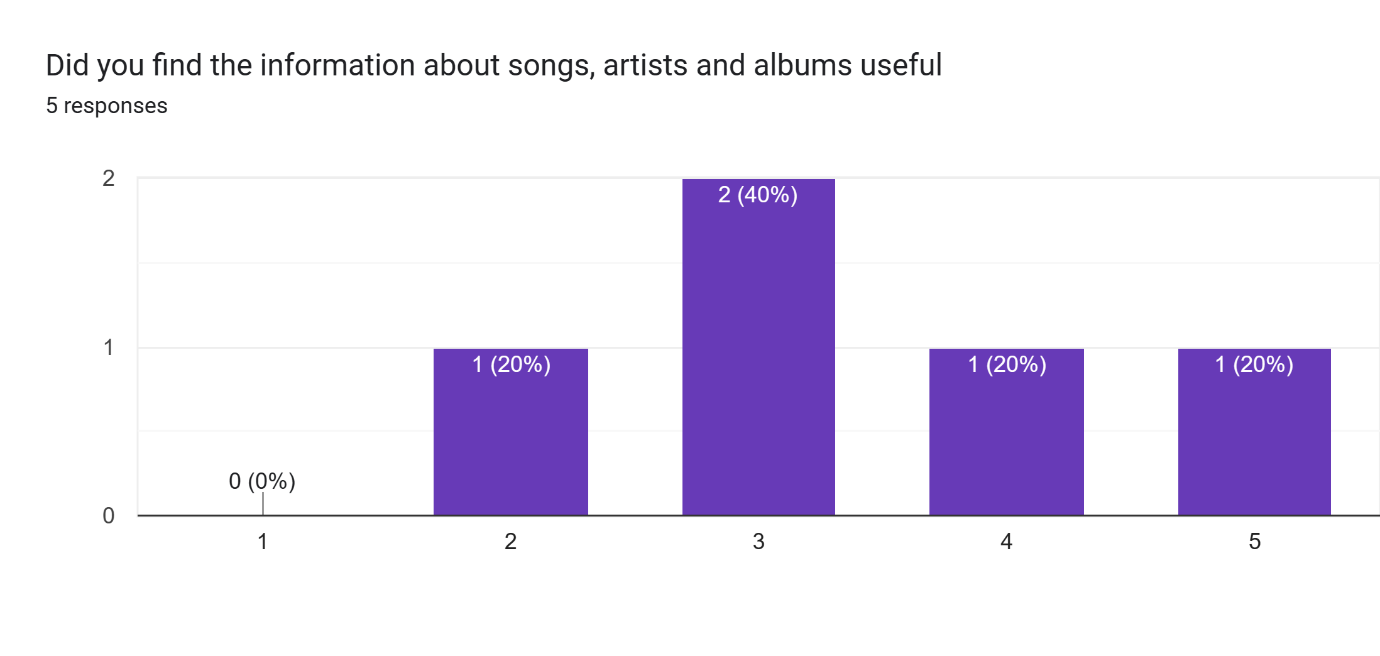
I gave my programme to 12 people and asked them to use it for a week. After this, I asked them a series of questions to get their feedback.

#### Question 1. Did you find the programme easy to use?

*Figure 24. Distribution of how easy users found it to use the program*

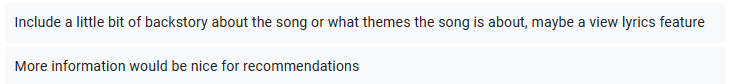
 According to the results of the 5 users I asked for feedback. I got 1 score of 3, and 4 scores of 4; to me this shows that my app is relatively easy to use, but there is definitely room for improvement. An issue my users had is that I didn’t have a separate page for signing up and logging in, which is not intuitive to the user. However, for the most part other than some small issues, my app seems to be easy to use.

#### Question 2. Did you find the information about songs/artists/albums useful?

*Figure 25. Distribution of how useful users found the information about songs/artists/albums*

The data here shows that the information my app shows about the music they enter is satisfactory, but it seems it is not enough to provide all users with satisfaction. I believe the reason for this is because the information about the music is relatively easy to access. However, most of my users said that the information about the sound of the song was interesting. I believe to create a better experience for my users I should add more information that the users cannot access to easily.

#### Question 3. What improvements would you make to the information provided about the songs/artists/albums?



**

*Figure 26. List of improvements users suggested for the part of the program that gets information about songs/artists/albums*

Here is a list of what I could improve about my music information pages:

Most of the

#### Question 4. Did you like the song suggestions?

[Graph]

*Figure 27. Distribution showing how much people liked the song suggestions made by the program*

[Discussion]

[Conclusion]

#### Question 5. Did you think that the song suggestions helped you try new music that you might not otherwise have tried?

[Graph]

*Figure 28. Distribution of how helpful users found the program in introducing them to new music that they might not have tried without using the program*

[Discussion]

[Conclusion]

#### Question 6. What improvements would you make to the song suggestion part of the program?

[Graph]

*Figure 29. List of improvements users suggested for the song suggestion part of the program*

[Discussion]

[Conclusion]

#### Conclusions from feedback

[Summary of conclusions – did the programme work! If not, why not, what improvements would you make?]

#### Limitations to the feedback

Like the previous survey I did to better understand the problem this project tried to solve, the users I received feedback from were mainly my classmates, so 17-18 year old students. This is a very narrow demographic. The main advantage of choosing this demographic is easy access and a high response rate. There are other advantages too, for example this is a demographic who are likely to want to use a program like this, this age group spends a lot of time listening to music, and they are also at a more formative time in developing their music taste.

The drawback of having such a limited age demographic is that it might not be representative of all music streaming platform users. Users who are older may have different preferences and listening styles, and users from various geographic or cultural backgrounds may have different needs and expectations. As a result, the results may be less generalizable. If I had more time, I would gather feedback from a wider range of users in order to see how easy and useful they found the program, as well as to take into account the unique requirements and preferences of various age groups, geographical locations, and cultural groups.

### Discussion

[To be included once you have the conclusions. Maybe talk about whether the algorithm works, whether or not it needs to be perfect, how there is a place for different approaches to helping people find new music, how it would be better with a wider set of songs]

### Possible Improvements

There are many possible improvements to the program. The ones I think are the highest priorities are:

1. Increase the size of the song database so there are a wider variety of songs
2. Make the program suggest more than 4 songs e.g. 10 songs
3. Add functionality that allows an admin account. This account would include the ability to update the song database as well as track how users are using the program
4. Allow the user to specify which of the 10 song attributes they think are most important, and increase the weighting on these (e.g. make them twice as important)
5. Increase the data sources the program looks at to get information about artists, songs, and albums. For example, this could use GPT4 functionality.
6. Make multiple tables in normalised form
7. [Any others you like, and any others suggested by the feedback]

Of all of these improvements, if I had more time I would choose number [you choose]. This is because [your reasons].

The approach I would take to make this improvement would include the steps below:

1. [Your suggested implementation steps]