

**Richard Jones**

A Level Computer Science Coursework

Music Centre Program

Centre Number – 52423

Candidate Number - 2389

Module - H446/03~04

Draft

Authors : Richard Jones

Date : 7 February 2017

Version : V0.1

E-mail : richardj99@sky.com

Contents

[1 Analysis 4](#_Toc481139468)

[1.1 Suitability for Solubility by computational Methods 4](#_Toc481139469)

[1.1.1 Description of Relevant Features of problem 4](#_Toc481139470)

[1.1.2 Why is this problem Suitable for solution by computational Methods? 4](#_Toc481139471)

[1.2 Stakeholder Management 5](#_Toc481139472)

[1.2.1 Who are the stakeholders and how were they chosen? 5](#_Toc481139473)

[1.2.2 How will they make use of the proposed solution? 5](#_Toc481139474)

[1.2.3 Why will the approach taken be suitable for stakeholder needs 6](#_Toc481139475)

[1.2.4 Research into Alternative Solutions 6](#_Toc481139476)

[1.2.5 Minimum Essential features required 8](#_Toc481139477)

[1.2.6 Limitation of Minimum Features 10](#_Toc481139478)

[1.2.7 Identification of proposed solution with reference to Research 10](#_Toc481139479)

[1.2.8 Hardware and software requirements for solution 11](#_Toc481139480)

[2 Design 12](#_Toc481139481)

[2.1 Breakdown of Problem 12](#_Toc481139482)

[2.1.1 Problem Elements suitable for computational solutions 12](#_Toc481139483)

[2.1.2 Justifying the process 12](#_Toc481139484)

[2.2 Detailed Structure of Solution to be developed 12](#_Toc481139485)

[2.2.1 User Interface Design 12](#_Toc481139486)

[2.2.2 Database Design 19](#_Toc481139487)

[2.2.3 Test Data to be used 20](#_Toc481139488)

[2.2.4 Algorithms required in solution 20](#_Toc481139489)

[2.3 Key Features of solution and their justification 20](#_Toc481139490)

[2.3.1 Structure of solution 20](#_Toc481139491)

[2.3.2 Features of solution 20](#_Toc481139492)

[3 System Development 21](#_Toc481139493)

[3.1 Database Initialisation 21](#_Toc481139494)

[3.1.1 Problem Decomposition 21](#_Toc481139495)

[3.1.2 Creating the Tables 21](#_Toc481139496)

[3.1.3 Creating Relationships 24](#_Toc481139497)

[3.1.4 Loading the Test Data 25](#_Toc481139498)

[3.1.5 Final View 25](#_Toc481139499)

[3.2 Class-less code 26](#_Toc481139500)

[3.2.1 Imported Libraries 26](#_Toc481139501)

[3.2.2 Linking the Database with the code 26](#_Toc481139502)

[3.2.3 Connecting the User Interface Files 26](#_Toc481139503)

[3.3 Login Screen Class 26](#_Toc481139504)

[3.3.1 Class Definition 26](#_Toc481139505)

[3.3.2 1st Iteration 27](#_Toc481139506)

[3.3.3 Testing for Iteration 01 28](#_Toc481139507)

[3.3.4 2nd Iteration 33](#_Toc481139508)

[3.3.5 Testing for Iteration 02 34](#_Toc481139509)

[3.4 Create Account Screen 36](#_Toc481139510)

[3.4.1 Project Decomposition 36](#_Toc481139511)

[3.4.2 Final Code 36](#_Toc481139512)

[3.5 36](#_Toc481139513)

[3.5.1 Problem Decomposition 36](#_Toc481139514)

[3.6 Songs Table Code 36](#_Toc481139515)

[3.7 Song Playback Code 36](#_Toc481139516)

[3.8 36](#_Toc481139517)

[3.8.1 37](#_Toc481139518)

[4 Evaluation 38](#_Toc481139519)

[4.1 Final Code 38](#_Toc481139520)

[4.1.1 Beginning of Code 38](#_Toc481139521)

[4.1.2 Login Window Class 38](#_Toc481139522)

[4.1.3 Main Window Class 39](#_Toc481139523)

[4.1.4 Settings Window Class 43](#_Toc481139524)

[4.1.5 Playlist Creation Dialog Window Class 46](#_Toc481139525)

[4.2 Final Testing 46](#_Toc481139526)

# Analysis

## Suitability for Solubility by computational Methods

### Description of Relevant Features of problem

From the late 1800s, music has been able to be recorded and distributed through many different media. For most of the 20th century, music was mainly released on either vinyl or cassette formats. However, as computers began to become more and more commonplace in households and society as a whole, digital solutions for storing recorded music have been released. With this advancement, the music needs to be organised (similar to a music library of physical music). A physical music library is normally sorted by a category, such as the record’s artist name, album name or song name. Usually this order is done by hand and is a laborious task. This organisation is done to allow people to easily and quickly navigate their library in search for a specific record. Alongside their library, music listeners also keep a music player of some kind. This could be anything from a vinyl record turntable to a CD player. In order to cater for this feature of the problem, I will need to make sure that the solution can play this music.

However, the idea of a music player program is no new concept. Many other companies have managed to create monumentally successful programs that organise music libraries and play music. Later, I will conduct research into similar solutions to gain a further understanding of how I can make my proposed solution more complex and relatable to consumer requirements.

I have been approached by Samuel Barrett, one of my fellow students, who expressed his own opinions about a music player program. He felt that many of the other music playing programs on the market today have become too complex in hopes to achieve a wider target market. In doing so, the programs have become too confusing to use and are becoming over encumbering on the machines that they are being run on. From this conversation, I decided to aim to create a solution that is lightweight and simple in nature. I also made Samuel Barrett one of my end users for the solution and a stakeholder in the finished product.

Though I am taking ideas from several previous solutions to this problem. It is important to understand that I do not want this program to be a main competitor in this market. This solution is not meant to be a fully official and profitable attempt, it is just a prototyped solution that will aim to solve the basic problems mentioned earlier.

### Why is this problem Suitable for solution by computational Methods?

The solution can be solved by computational methods for a multitude of reasons:

First, the library organisation. Using a database and tag reading, information about the songs (including their artist, and album) can be stored. Then, using a selection of methods, which are yet to be decided on (such as whether to use search queries or iterative table searching), a view similar to a physical music library can be customised. In fact, using this method, the library can be improved upon. If somebody wanted to change the view of their library, say from sorted by artist to sorted by album, they would have to physically reorder their records which could be physically strenuous and time consuming. Using a computer, the exact same task can be done by clicking a button.

Second, searching for a song, album or artist would be even quicker. In a physical music library, the user would have to manually scan the library for the record. While this could be quickened with an index, it would not reach the speed that it could be searched for using a computer. Using either an iterative search function or an SQL query, the record could be found in a matter or milliseconds.

Third, music can be played using computational methods. Audio files exist for computers and can be played very easily. Many operating systems even come built in with integrated music players. By simply recording the location of each music file, the file can be played instantly if the correct codec is used. This would be very efficient compared to the physical solution, where the record would have to be taken from the library and manually placed in the CD player or turntable.

In conclusion, many improvements can be made through solving the problem by computational methods. Making the overall task of listening and organising music more enjoyable and less strenuous to the user.

## Stakeholder Management

### Who are the stakeholders and how were they chosen?

There could be many potential stakeholders for this solution as most of the population have a mutual enjoyment of music and would be interested in the media player program described.

Many members of my tutor group have expressed a deep interest in music and would be suitable stakeholders in this project. As I have already mentioned, Samuel Barrett has already expressed a deep interest in my program, and hopes to see it to be a successful, yet lightweight solution to the problem. I aim to choose stakeholders who are genuinely interested in the program’s wellbeing and who will give detailed yet constructive criticism to ensure its success.

To collate as wide a target market as possible, I have randomly selected six students to be my stakeholders. These people will originate from my tutor group, my school’s staff and my family. They will express their wants and needs out of a solution to this problem and will be consulted throughout the design, development and testing process of this program.

### How will they make use of the proposed solution?

Several studies (and my own personal experience) have shown that people tend to listen to music while they are working and relaxing, either at home or at a workplace. I aim for my solution to be used in these situations.

The solution will be in the form of an application that people can keep open on their desktop, allowing them to concentrate on work or relax while using the solution.

Furthermore, I have conducted several interviews with the stakeholders to create a more specific picture of what my solution needs to satisfy their needs. Here is the survey that I gave to all of them:

|  |
| --- |
| **Name:** |
| **Age:** |
| **Vocation:** |
| **Which program do you currently use to play music?** |
| **What, in your opinion, are the unique strengths of this solution?** |
| **What, in your opinion, are the limitations of this solution?** |
| **What feature would you personally want most out of a solution tailored to your needs?** |

The results from this survey were then collated to allow the data to be viewed and concluded more easily. Here are the results:

* **Which program do you currently use to play music?**
  + Windows Media Player **– 2**
  + ITunes – **3**
  + Other – **1**
* **What, in your opinion, are the unique strengths of this solution?**
  + Wide support for files – **2**
  + Online store – **1**
  + Video Playback – **1**
  + Smart playlists/Genius Playlists - 2
* **What, in your opinion, are the** **limitations of this solution?**
  + Slows down machine – **5**
  + Too many pointless features – **1**
* **What feature would you personally want most out of a solution tailored to your needs?**
  + Lightweight solution/won’t interfere with machine performance – **3**
  + Playlists, both automatically created – **3**

Conclusion

Some conclusions can be made from the data gathered in this interview. First, my selected stakeholders show a great care in how the solution will affect the performance of their machine when it is running.

### Why will the approach taken be suitable for stakeholder needs

As previously mentioned, my solution is going to be a lightweight, yet useful, attempt at the ‘music player’ problem. This will be suitable as many of the stakeholders mentioned how their preferred music players produced issues with the performance of their PC.

Furthermore, my stakeholder interviews have showed only a small number of users are interested in video playback. This could be because of the advancements in video streaming products such as Netflix and Amazon Prime Instant Video. These solutions do not require a player to play their files, it is all included in one system. However, my will to focus on music playback perfectly suits the stakeholders’ needs.

### Research into Alternative Solutions

The main competitors for this solution are Microsoft’s Windows Media Player and Apple’s ITunes.

|  |  |
| --- | --- |
| Product Name: | Windows Media Player |
| Features: | * Plays music (with music navigation buttons and progress bar). * Oorganises local media files into libraries * Allows video and photo playback * Allows connection to Network Accessible Devices * Provides plugins for cosmetic features such as player themes and visualizations for music. |
| Supported File Formats: | * Windows Media Formats (.asf, .wma, .wmv, .wm) * Windows Media Metafiles (.asx, .wax, .wvx, .wmx) * Windows Media Metafiles (.wpl) * Microsoft Digital Video Recording (.dvr-ms) * Windows Media Download Package (.wmd) * Audio Visual Interleave (.avi) * Moving Pictures Experts Group (.mpg, .mpeg, .m1v, .mp2, .mp3, .mpa, .mpe, .m3u) * Musical Instrument Digital Interface (.mid, .midi, .rmi) * Audio Interchange File Format (.aif, .aifc, .aiff) * Sun Microsystems and NeXT (.au, .snd) * Audio for Windows (.wav) * CD Audio Track (.cda) * Indeo Video Technology (.ivf) * Windows Media Player Skins (.wmz, .wms) * QuickTime Movie file (.mov) * MP4 Audio file (.m4a) * MP4 Video file (.mp4, .m4v, .mp4v, .3g2, .3gp2, .3gp, .3gpp) * Windows audio file (.aac, .adt, .adts) * MPEG-2 TS Video file (.m2ts) |
| Support: | 24/7 Help Centre provided by Microsoft |

|  |  |
| --- | --- |
| Product Name: | ITunes |
| Features: | Plays Music, organises local media files into libraries, allows video and photo playback, contains a store where users can buy music and video. |
| Supported Music File Formats: | .mp3, .aiff, .wav, .mpeg, .aac, .m4a, .mov, .aac, .ogg |
| Support: | 24/7 Help Centre provided by Apple |

Conclusion

While both of these solutions have strengths and weaknesses in areas, both provide exemplar solutions to the problem in discussion. The most prominent being Windows Media Player’s unprecedented selection of supported file types and ITune’s online store, providing original features to the problem in question.

However, through research, I discovered that both solutions have had many reports of slowing down the performance of machines, particularly ITunes. While the additional features that they provide are appreciated by many people, it seems they come at a cost in the usability of the program. The main point that my stakeholders brought to my attention in the interview is that they require a solution that has little to no impact on their machine. This means that some features of the solution (including some that are in these solutions) must be ignored to ensure its usability.

It is also appropriate to mention that both solutions are professional attempts, meant for a large-scale audience in a largely commercial environment. Because of this, these solutions are likely to have taken several years to perfect, requiring the help of a large team working on the solution at the same time. My solution is meant for a smaller audience and is scheduled to be done in less than a year, meaning that many of the features must be forgotten to meet the target by the required deadline.

It is also unlikely that my solution will be able to provide the 24/7 support, that these two solutions allow, in the immediate future. To address this, I have decided that the program will be open-source. This will have a very small effect on the solution in question, just that the program will have to be well structured with clear variable names to accommodate third-party development.

### Minimum Essential features required

#### Database

1. Form a normalized database to store information relating to the music.
2. Create a table for the songs.
3. Create a table for the albums.
4. Create a table for the artists.
5. Create a table for the playlists.
6. Create a table that stores information about the program’s users.

#### User Interface

1. Provide a login window
2. Provide a create account window
3. Provide a Media Player window
4. Provide a settings window
5. Provide a playlist manager screen
6. Provide a dialog box to change a playlist’s name.

#### Login Screen

1. Provide an input box where users can enter their username.
2. Provide an input box where users can enter their password.
3. Provide a button that users can use to log in to the program.
4. Validate the username and password against the program’s database.
5. Provide access to the main program if validation is successful.
6. Provide a button that allow the user to navigate to the create account window.
7. Provide a label that notifies the user of any errors and gives instructions.

#### Create Account Screen

1. Provide a text box that allows the user to input their desired username.
2. Provide a text box that allows the user to input their desired password.
3. Provide a button that allows the user to create their account.
4. Validate the username and password using regex to make sure the inputs are long enough, contain numbers and contain letters.
5. Provided the validation is successful, append the details to the database alongside a 0 for the administrator field. Then close the window.

#### Media Player Screen

1. Provide a table that allows the user to navigate the library.
2. When an artist is clicked in the table, show all of the albums by the respective artist in the table.
3. When an album is clicked in the table, show all of the song from the respective album in the table.
4. When a song is clicked, play the respective song.
5. Create a queue of the next songs in the table when a song is clicked.
6. When a playlist is clicked, display all the song from the respective playlist.
7. Provide a button that allows the user to change the view of the library to scrutinize artists.
8. Provide a button that allows the user to change the view of the library to scrutinize albums.
9. Provide a button that allows the user to change the view of the library to scrutinize songs.
10. Provide a button that allows the user to change the view of the library to scrutinize playlists. If the user isn’t an administrator, only show the playlists that are owned by the user or are ownerless (such as the ‘most played playlist’.
11. Provide a button that allows the user to navigate to the settings window.
12. Provide a button that allows the user to navigate to the playlist manager window.
13. Provide a text box that allows the user to search for an item in the database.
14. Provide a dropdown box that allows the user to select what item type they want to search for (e.g. artist, album, song).
15. Provide a button that allows the user to confirm the search.
16. Provide a button that allows the user to exit the application.
17. Provide a button that allows the user to pause the song that is currently being played.
18. Provide a button that allows the user to resume the song if a song is paused.
19. Provide a button that allows the user to skip to the next song in the queue when clicked.
20. Add the previously played song to a stack when said button is activated.
21. Provide a button that allows the user to play the previously played track when clicked.
22. Provide labels that notify the user what song is playing.
23. Provide labels that indicate what artist or album the user is currently viewing in the table.

#### Settings Screen

1. Provide an administrator section that only users with administrator permissions can see.
   1. Provide a table that shows all the program’s users (excluding the current user).
   2. Allows users to be selected from the table.
   3. Provide a label that displays the selected user at a given time.
   4. Provide a button that deletes the selected user from the Users table in the database.
   5. Provide a button that toggles the selected user’s administrator settings in the Users table.
2. Provide an import section:
   1. Provide a text box that allows the user to input a selected directory.
   2. Provide an import button that reads in the user’s input when clicked.
   3. Validate the input to make sure the directory exists.
   4. Scan through the inputted directory and catch any .mp3 files.
   5. Read the relevant information from the ID3 tags of each .mp3 file, e.g. Song name, artist name, album name, genre, song length.
   6. Append the relevant information from the tags into the database, namely the Songs, Artists and Albums tables.
   7. Input N/A if any of the tags are not found.
3. Provide a button that allows the user to exit the settings window when clicked.
4. Provide a button that allows the user to log out of the program, taking them back to the login screen.

#### Playlist Manager Window

1. Provide a table that will display all of the user’s playlists.
2. When a playlist in this table is clicked, songs from the respective playlist will be shown.
3. When a song in this table is clicked, the respective song will be removed from the playlist it belongs to (these changes will also be made in the database).
4. Provide a table that will display all of the songs in the database.
5. When a song in this table is clicked, add the song to the selected playlist provided one is selected (this change will also be made in the database).
6. Provide a button that will deselect the currently selected playlist and display all of the playlists in the playlist tables (see objective 52).
7. Provide a button that will open the new playlist dialog box when clicked.
8. Provide a button that will close this window when clicked.

#### Playlist Dialog Box

1. Provide a text box that will allow the user to input their desired name for the playlist.
2. Provide a button that will allow the user to create their playlist using their input when clicked.
3. Provide a button that will allow the user to close the dialog box when clicked.

### Limitation of Minimum Features

There are several limitations to my proposed solution:

Firstly, my solution will only cater for music playback and not photo or video. This decision was made to reduce the strain that the solution had over the machine it was running on. Better suiting the stakeholder’s requirements. I also feel that it would be an important decision to keep this project focused on music for the near future. This is to make sure that all efforts stays on perfecting the music playing problem and making a better all-round solution. I felt that this would be more suitable than diverting resources to incorporate phot and video.

Secondly, my solution will only support .mp3 files. MP3 files are the most used audio files to date, but there are many other widely used formats which could need support. However, many of these files do not contain the same tags as mp3 (ID3), meaning that my program will not be able to read them into the database. This decision was made to keep the solution as ‘cut down’ as possible as well as reducing the time of development to fit in with the deadline. Better suiting the stakeholder’s needs.

I have personally had thoughts of incorporating access to an internet database of music. This could be utilized in many ways, including finding missing information for the local music files as well as recommending new music to the user depending on what music they listen to more. However, given the time pressures that this project has, I feel that it would be inappropriate to take on this responsibility.

### Identification of proposed solution with reference to Research

Through the research into similar solutions, I discovered that both solutions have had many reports of slowing down the performance of machines, particularly ITunes. While the additional features that they provide are appreciated by many people, it seems they come at a cost in the usability of the program. The main point that my stakeholders brought to my attention in the interview is that they require a solution that has little to no impact on their machine. This means that some features of the solution (including some that are in these solutions) must be ignored to ensure its usability.

It is also appropriate to mention that both solutions are professional attempts, meant for a large-scale audience in a largely commercial environment. Because of this, these solutions are likely to have taken several years to perfect, requiring the help of a large team working on the solution at the same time. My solution is meant for a smaller audience and is scheduled to be done in less than a year, with minimal help from outside sources. This means that many of the features must be forgotten to meet the target by the required deadline.

### Hardware and software requirements for solution

This solution will be programmed in Python 3.4. Python was chosen because it is an object-oriented language, allowing the solution to use models, methods and classes. These constructs will aid the solution’s development because it will allow the problem to be split up into more solvable sections, reducing space and producing a more efficient solution.

Python was also chosen due to its large collection of libraries. Libraries will be used in this solution to read the ID3 tags and allow the playback of songs. The use of these libraries will reduce the amount of time that the solution will be developed, as well as decrease the amount of code in the core program.

# Design

## Breakdown of Problem

### Problem Elements suitable for computational solutions

I aim to decompose the problem down into smaller sub-problems in the hopes that this will make the problem easier to solve. These are the steps that will be taken:

1. First, the application’s User Interface will be designed. The UI will comprise of several windows each serving different purposes. The windows will be the login screen, the create account screen, the media player screen and the settings window. Later in this section, the initial design of these screens will be established by the stakeholders. The UI will be developed in PyQt Designer.
2. After the UI has been designed, the database will be established. This database will hold all the data used by the program, including data relating to the music files (songs, artists, albums and playlists) and data related to the users (usernames and passwords).
3. Once the infrastructure for the application has been implemented, the solution will be coded. This process will start with the coding for the login section. This section will accept account information from the user and validate it in the program’s database. If the details are accepted, the user is admitted into the main program.
4. After the login section, the import function will be programmed in the Settings Window. The function will accept a user specified directory and scan it for any mp3 files, inputting the details from the files into the database.
5. Then the view of the Music Player’s library will be coded. Buttons will be implemented to allow the user to view all the tables in the database (artists, songs, albums etc.) and allow the user to traverse the music library (selecting an artist will retrieve the albums by that artist, selecting an album will retrieve the songs by that artist and selecting a song will play it).
6. The music playback functions will then be coded. This will play a song when it is selected from the view. A pause button will also allow a song to be stopped momentarily. A play button will also be provided to resume a paused song.
7. The queue function will then be coded. The function will add all of the songs under a selected song to a queue. Once the selected song has finished playing, the next song in the queue will be retrieved. This will continue until the queue is empty or a new song is selected.
8. After the queue function has been implemented into the program, the skip buttons will be programmed to traverse the queue backwards and forwards.
9. After the song queueing function has been programmed, the search function will be programmed. The search function will accept a string and use it to search the Music Library for any similar fields in the records. All those that match the User’s input will be outputted in the view. The function will also be able to scrutinize what to search for (over artists, albums, songs and playlists).
10. Then the playlist feature will be coded. The feature will allow users to add new playlists into the database as well as access existing ones. A window can then be accessed that will allow the user to add and remove songs from the playlist.

### Justifying the process

## Detailed Structure of Solution to be developed

### User Interface Design

After having a discussion with the stakeholders for this product. I drew up a plan of the interface that everyone could agree on. Slight changes are likely to be made to these in order to improve functionality.

#### **Login Window**

Username:

Password:

Log-In

Create Account

Please Login

|  |  |  |  |
| --- | --- | --- | --- |
| **Object** | **Number** | **Purpose** | **Example Input** |
| txt\_username | 1 | This is a textbox where the user can input their username. | “user1” 🡪 String |
| txt\_pass | 2 | This is a textbox where the user can input their password. | “password1” 🡪 String |
| btn\_login | 3 | This is a button that, once clicked, uses the information that the user has provided and attempts to gain access to the main program. | N/A |
| btn\_crtacc | 4 | This is a button that, once clicked, will take the user to the ‘create account’ window. The user can then proceed to create a new account. | N/A |
| lbl\_username | 5 | This label currently displays ‘Please Login’. However, if any errors are brought up when checking the user’s inputs (e.g. username doesn’t exist in database), a suitable message will be displayed. | N/A |
| lbl\_password | 6 | This label is used to indicate where the user’s password should be entered. | N/A |
| lbl\_info | 7 | This label is used to indicate where the user’s username should be entered. | N/A |

#### **Create Account Window**

Create an Account

Create Account

Password:

Username:

|  |  |  |  |
| --- | --- | --- | --- |
| **Object** | **Number** | **Purpose** | **Example Input** |
| txt\_username | 1 | This object allows the user to input their desired username. | “User1” 🡪 String |
| txt\_password | 2 | This object allows the user to input their desired password. | “Password1” 🡪 String |
| lbl\_username | 3 | This label indicates where the user should enter their username. | N/A |
| lbl\_password | 4 | This label indicates where the user should enter their password. | N/A |
| lbl\_info | 5 | This label initially displays “Create an Account”. However, if any errors are raised when validating the user’s inputs, the label will show a suitable message (e.g. Username has already been taken). | N/A |
| btn\_crtacc | 6 | This button will activate the procedure that takes the user’s inputs and creates an account using them (provided the inputs are successfully validated) | N/A |

#### **Media Player Window (Main Window)**

Songs View

Artists

Albums

Songs

Playlists

Settings

Playlist Manager



Now Playing

Search

|  |  |  |  |
| --- | --- | --- | --- |
| **Object** | **Number** | **Purpose** | **Example Input** |
| Btn\_back | 1 | When this button is activated, it will play the song at the top of the ‘previously playing’ stack. | N/A |
| Btn\_forward | 2 | When this button is activated, it will play the song at the front of the “next song” queue. | N/A |
| Btn\_pause | 3 | When this button is activated, the song that is currently playing will be paused. | N/A |
| Btn\_play | 4 | When this button is activated, any song that has been paused will be resumed. | N/A |
| Lbl\_nowplaying | 5 | Initially, this label will display: “Now Playing:”. However, when a song begins playback, the name of the song will be added to the end of the label. | N/A |
| Btn\_exit | 6 | When this button is activated. The entire program will shut down. | N/A |
| Btn\_search | 7 | When this button is activated, The input from the drop-down menu and the search box will be collected and the music library will then be searched (using these inputs). | N/A |
| Drp\_search | 8 | This drop-down menu allows the user to specify whether they want to search for Albums, Songs, Artists or Playlists | N/A |
| txt\_search | 9 | This object allows the users to input the term that they want to search for in the Music Library. | “ArtistName” 🡪String |
| Lbl\_album | 10 | This label will show which album (if any) is being viewed in the Song View | N/A |
| Lbl\_artist | 11 | This label will show which artist (if any) is being viewed in the Song View | N/A |
| Lbl\_nowviewing | 12 | This label is to help the user understand what the lbl\_artist and lbl\_album objects are denoting | N/A |
| Btn\_albums | 13 | When this button is activated, the song view (tbl\_songs) will show all the albums in the music library | N/A |
| Btn\_artists | 14 | When this button is activated, the song view (tbl\_songs) will show all the artists in the music library. | N/A |
| Btn\_playlists | 15 | When this button is activated, the song view (tbl\_songs) will show all the playlists in the music library. | N/A |
| Btn\_plylstmngr | 16 | When this button is activated, the ‘Playlist Manager’ window will be opened on top of the ‘Main’ Window. | N/A |
| Btn\_settings | 17 | When this button is activated, the ‘Settings’ Window will be opened on top of the ‘Main’ Window | N/A |
| Tbl\_songs | 18 | This table will show all the fields in the libraries database. At a basic level, it will show all of artists, artists, songs and playlists (depending on which button the user selects). However, it will also display albums belonging to a certain artist, songs belonging to a certain album and songs belonging to a certain playlist | N/A |

#### **Settings Window**

Import

Directory:

Users Table

Delete User

Toggle Admin.

Logout

Exit

**User: User1**

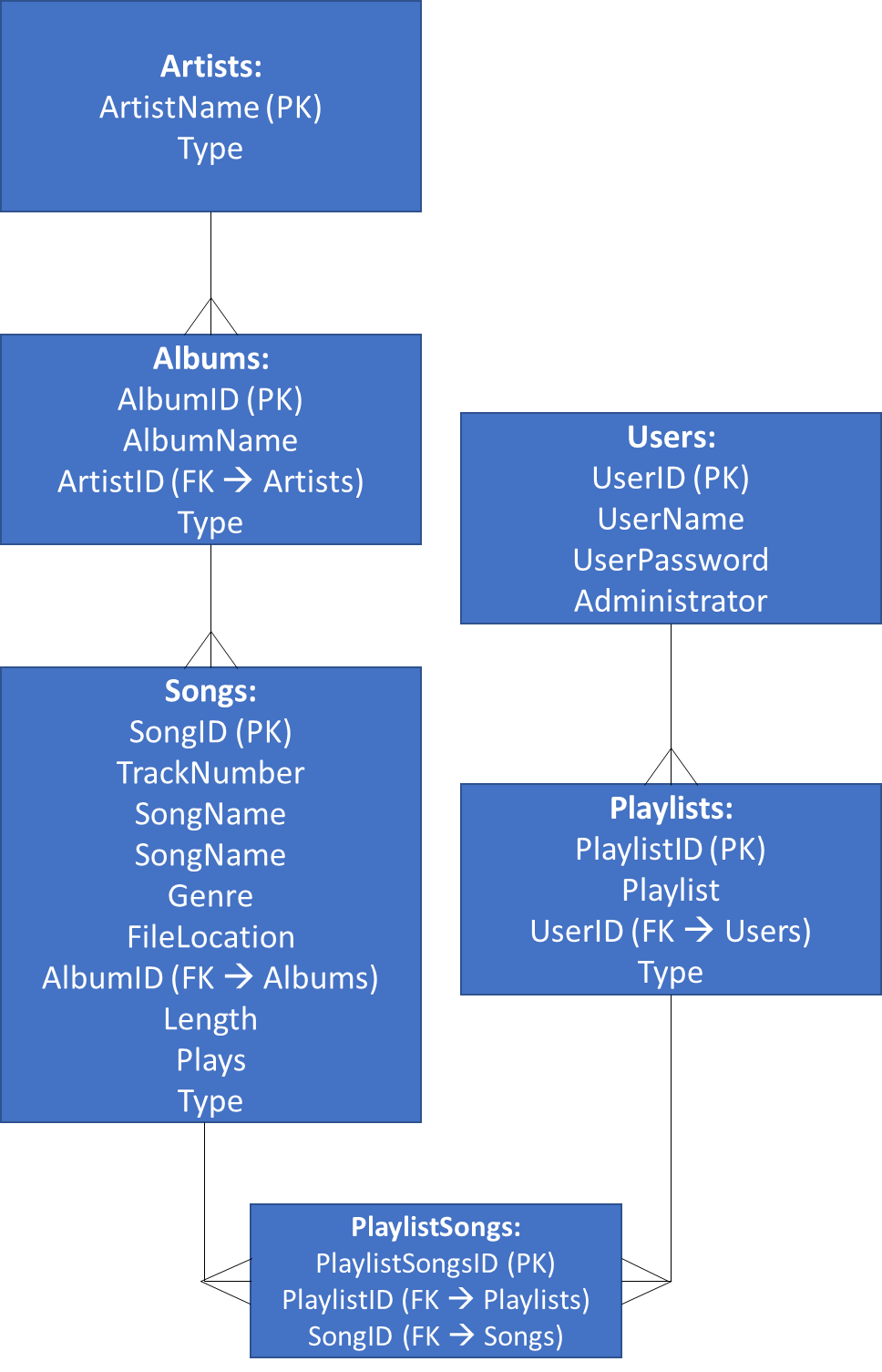
|  |  |  |  |
| --- | --- | --- | --- |
| **Object** | **Number** | **Purpose** | **Example Input** |
| Btn\_exit | 1 | When this button is activated, this window will be closed. | N/A |
| Btn\_logout | 2 | When this button is activated, the user will be taken back to the login screen. Any variables that identify the user (e.g. UserID and UserName) will be set to a null value. | N/A |
| Btn\_admin | 3 | When this button is activated, the selected user (the field selected in tbl\_users) will be made an admin or, if they are already an admin, will be made an ordinary user. | N/A |
| Btn\_delUser | 4 | When this button is activated, the selected user (the field selected in tbl\_users) will be deleted from the program and the database. | N/A |
| Lbl\_user | 5 | This object has a null value of “User:”. However, when a user is selected from tbl\_users, the object will also show the name of the user. | N/A |
| Tbl\_users | 6 | This table will show all of the users (from the users table in the database), excluding the user who is currently logged in. | N/A |
| Btn\_import | 7 | When this button is activated, the user’s input (their selected directory) from txt\_dir is taken. The music in this directory is then imported into the database. | N/A |
| Lbl\_dir | 8 | This object will indicate where the user should enter their directory. | N/A |
| Txt\_dir | 9 | This object will allow the user to input the directory from where they want their music to be imported from. | “C:/Users/User/Music/” 🡪 String |

#### **Playlist Manager Window**

|  |  |  |  |
| --- | --- | --- | --- |
| **Object** | **Number** | **Purpose** | **Example Input** |
| Btn\_close | 1 | When this button is activated, the window is closed, taking the user back to the Main Window |  |
| Btn\_newplylst | 2 | When this button is activated, a dialog box will open. Here the user can input the new playlist’s name. |  |
| Btn\_plylstreset | 3 | When this button is activated, th |  |
| Tbl\_playlists | 4 | This table will display all of the information relating to playlist. This includes the names of the playlists and the songs that each playlist contain. |  |
| Tbl\_songs | 5 | This table will show all of the songs that are currently in the music library. |  |

### Database Design

In order to maintain complete and consistent efficiency in my program, I will need to have a normalised database, preferrably in Third Normal Form.

In order to do this, I have decided to isolate the three most independent fields to describe the songs (The artist, album and title).

The diagram also shows that there will be a PlaylistSongs table. This table was used to create a relation between the Songs and Playlists tables in a more normalized format (making sure the two tables did not have a many to many relationship).

### Test Data to be used

3,600 songs as mp3s. These will be used as test data.

Identified and justified the test data to be used during the iterative development of the solution.

Identified and justified any further data to be used in the post development phase.

### Algorithms required in solution

Import Function

This function will scan all of a specified directory. It will identify any mp3 files in this directory and obtain the tags from them. These tags will then be added into the

## Key Features of solution and their justification

### Structure of solution

Defined in detail the structure of the solution to be developed.

Described the solution fully using appropriate and accurate algorithms justifying how these algorithms form a complete solution to the problem.

### Features of solution

Described, justifying choices made, the usability features to be included in the solution.

Identified and justified the key variables / data structures / classes (as appropriate to the proposed solution) justifying and explaining any necessary validation.

# System Development

## Database Initialisation

### Problem Decomposition

The database will have to hold all the data on the songs in the user’s library, particularly the artist’s name, the album’s name and the song’s name. On top of this, the database must also store all the information on the playlists created in the program and the information of all the users who have access to the program.

To maintain efficiency in the database (removing data redundancy and duplication), the database will be put in third normal form. Doing so will also prevent problems in data management by the program.

### Creating the Tables

As mentioned in the Design section of the documentation, the database will contain the following tables:

* Albums
* Artists
* Songs
* Playlists
* Users
* PlaylistSongs

The required columns for each of these tables will be created through SQLite code. The code will also have relations assigned to them through foreign keys. Primary keys will also be assigned to each table.

When the solution is distributed for the stakeholder’s use, the Albums, Artists. Songs, Users and PlaylistSongs tables will be empty, as these are filled with information specific to each user. The Playlists table will contain one field, for the most played playlist. For the testing of the program, test data will be loaded into the

First, the tables and primary keys will be established. Then, the foreign keys relating the databases will be established.

#### Albums

The albums table will hold all of the information specifically related to the albums in general:

* AlbumID (int) **(PK)**
* AlbumName (text)
* ArtistID (int) **(FK)**

Any relations to artists table will be established after the creation of the tables.

Creation Code:

CREATE TABLE `Albums` (

`AlbumID` INTEGER UNIQUE,

`AlbumName` TEXT,

`ArtistID` INTEGER,

PRIMARY KEY(`AlbumID`),

);

#### Artists

The artists table will hold all the information specifically related to the artists in general:

* ArtistsID (int) **(PK)**
* ArtistName (text)

Creation Code:

CREATE TABLE `Artists` (

`ArtistID` INTEGER UNIQUE,

`ArtistName` TEXT,

PRIMARY KEY(`ArtistID`)

);

#### Songs

The songs table will hold all of the information specifically on the songs in general:

* SongID (int)
* SongName (text)
* TrackNumber (int)
* Genre (text)
* FileLocation (str)
* AlbumID (int) **(FK)**
* Length (int)
* Plays (int)

Any relations to the albums and artists tables will be created after the initial creation of the tables.

Creation Code:

CREATE TABLE `Songs` (

`SongID` INTEGER UNIQUE,

`TrackNumber` INTEGER,

`SongName` TEXT,

`Genre` TEXT,

`FileLocation` TEXT,

`AlbumID` INTEGER,

`Length` REAL,

`Plays` INTEGER,

PRIMARY KEY(`SongID`),

);

CREATE TABLE `Songs` (

`SongID` INTEGER UNIQUE,

`TrackNumber` INTEGER,

`SongName` TEXT,

`Genre` TEXT,

`FileLocation` TEXT,

`AlbumID` INTEGER,

`Length` REAL,

`Plays` INTEGER,

PRIMARY KEY(`SongID`),

);

#### Playlists

The playlists table will hold all the information related to the playlists in the program. This will include all user-created playlists and the ‘Most Played’ playlist that will already be in the table. The table will include the following values:

* PlaylistID (int) **(PK)**
* Playlist (text)
* UserID (int) **(FK)**

Creation Code:

#### PlaylistSongs

CREATE TABLE `Playlists` (

`PlaylistID` INTEGER UNIQUE,

`Playlist` TEXT,

`UserID` INT,

PRIMARY KEY(`PlaylistID`),

);

In order to keep the database in third normal form, none of the tables can have a many-to-many relationship with another. This table is used as a go-between for the Playlists table and the Songs table. Instead of having a many-to-many relationship with one another, they now each have a one-to-many relationship with the PlaylistSongs table. The contents of this table are as follows:

* PlaylistSongsID **(PK)**
* PlaylistID **(FK)**
* SongID **(FK)**

Creation Code:

CREATE TABLE `PlaylistSongs` (

`PlaylistSongsID` INTEGER,

`PlaylistID` INTEGER,

`SongID` INTEGER,

PRIMARY KEY(`PlaylistSongsID`),

);

#### Users

This table will hold all of the information about the Users. The fields in the table are as follows:

* UserID (int) **(PK)**
* UserName (text)
* UserPassword (text)
* Administrator (int)

Creation Code:

CREATE TABLE `Users` (

`UserID` INTEGER UNIQUE,

`UserName` TEXT,

`UserPassword` TEXT,

`Administrator` INT DEFAULT 0,

PRIMARY KEY(`UserID`)

);

### Creating Relationships

After the initial tables were created, the relationships between them were established through the foreign keys. The foreign keys used in this database are as follows:

* Albums.ArtistID 🡪 Artists.ArtistID
* Playlists.UserID 🡪 Users.UserID
* PlaylistSongs.PlaylistID 🡪 Playlists.PlaylistID
* PlaylistSongs.SongID 🡪 Songs.SongID
* Songs.AlbumID 🡪 Albums.AlbumID

These relationships will be made in **DB Browser for SQLite** (the Database Management System).

#### Reformed Code

**Albums**

CREATE TABLE `Albums` (

`AlbumID` INTEGER UNIQUE,

`AlbumName` TEXT,

`ArtistID` INTEGER,

PRIMARY KEY(`AlbumID`),

FOREIGN KEY(`ArtistID`) REFERENCES `Artists`(`ArtistID`)

);

**Artists**

CREATE TABLE `Artists` (

`ArtistID` INTEGER UNIQUE,

`ArtistName` TEXT,

PRIMARY KEY(`ArtistID`)

);

**PlaylistSongs**

CREATE TABLE `PlaylistSongs` (

`PlaylistSongsID` INTEGER,

`PlaylistID` INTEGER,

`SongID` INTEGER,

PRIMARY KEY(`PlaylistSongsID`),

FOREIGN KEY(`PlaylistID`) REFERENCES `Playlists`(`PlaylistID`),

FOREIGN KEY(`SongID`) REFERENCES `Songs`(`SongID`)

);

**Playlists**

CREATE TABLE `Playlists` (

`PlaylistID` INTEGER UNIQUE,

`Playlist` TEXT,

`UserID` INT,

`Type` INT DEFAULT 3,

PRIMARY KEY(`PlaylistID`),

FOREIGN KEY(`UserID`) REFERENCES `Users`(`UserID`)

);

**Songs**

CREATE TABLE `Songs` (

`SongID` INTEGER UNIQUE,

`TrackNumber` INTEGER,

`SongName` TEXT,

`Genre` TEXT,

`FileLocation` TEXT,

`AlbumID` INTEGER,

`Length` REAL,

`Plays` INTEGER,

PRIMARY KEY(`SongID`),

FOREIGN KEY(`AlbumID`) REFERENCES `Albums`(`AlbumID`)

);

**Users**

CREATE TABLE `Users` (

`UserID` INTEGER UNIQUE,

`UserName` TEXT,

`UserPassword` TEXT,

`Administrator` INT DEFAULT 0,

PRIMARY KEY(`UserID`)

);

### Loading the Test Data

As mentioned previously, the artists, albums, songs and PlaylistSongs tables take their data as inputs from the user. These tables will be empty as a result. The Playlist table will only contain a field for the ‘Most Played’ playlist.

#### Most Played Playlist

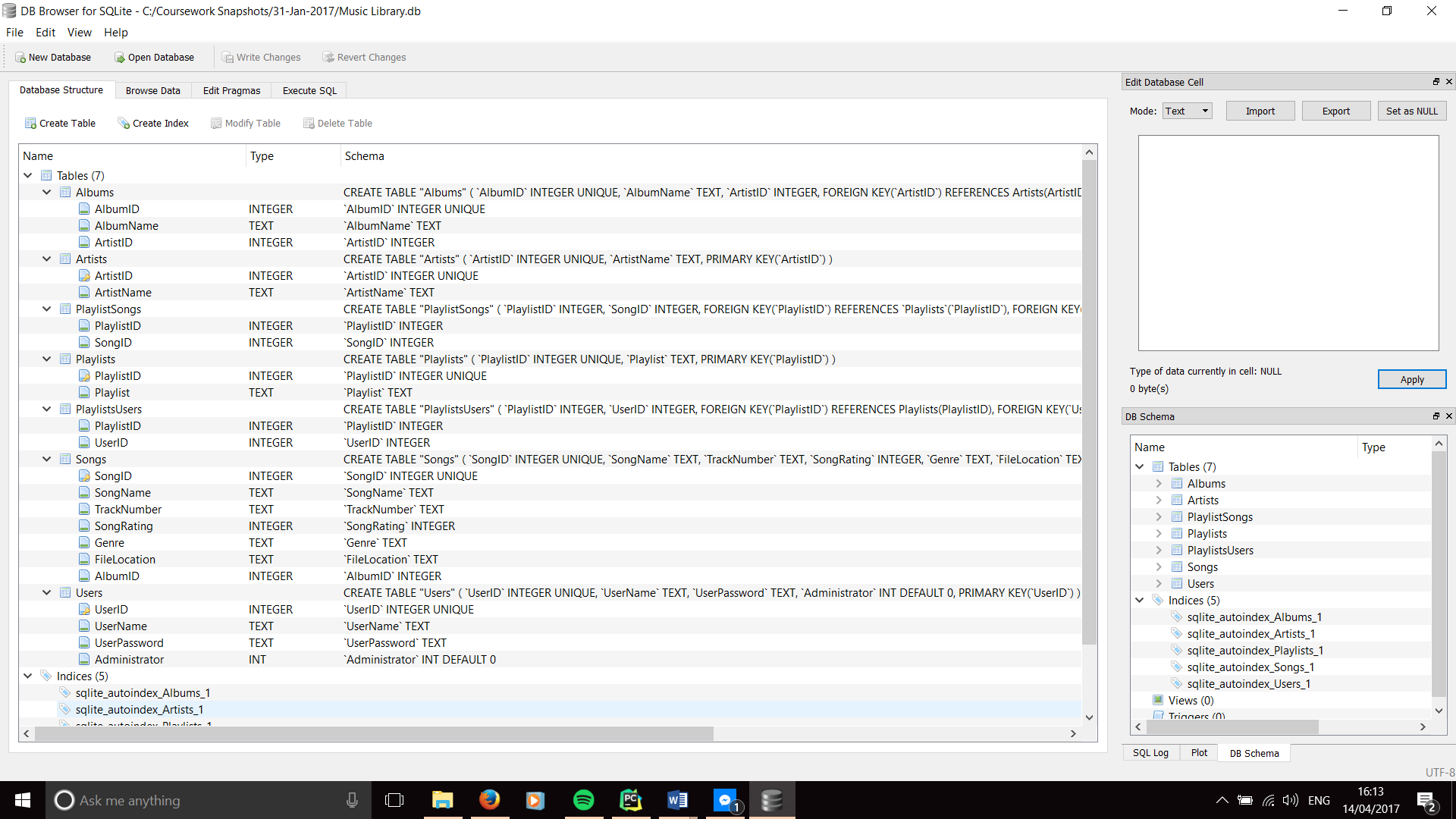
Here is the code used to insert ‘Most Played’ playlist into the Playlists database:

“INSERT INTO Playlists VALUES (1, “Most Played”, 0)”

The playlist is given a UserID of 0 as it is available for all users to use.

### Final View

Here is the final view of the database:



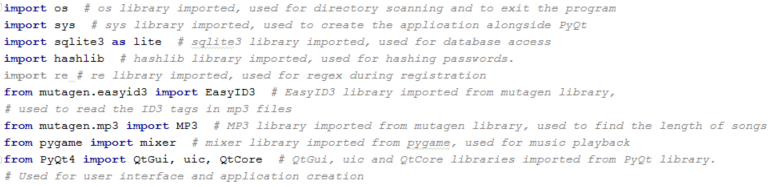
## Class-less code

Some of the code to be used in the program does not belong in any specific class. Most of these lines are to initialise components of the program (such as the database and .ui files) or to create objects.

All the imported libraries also belong in this section.

### Imported Libraries

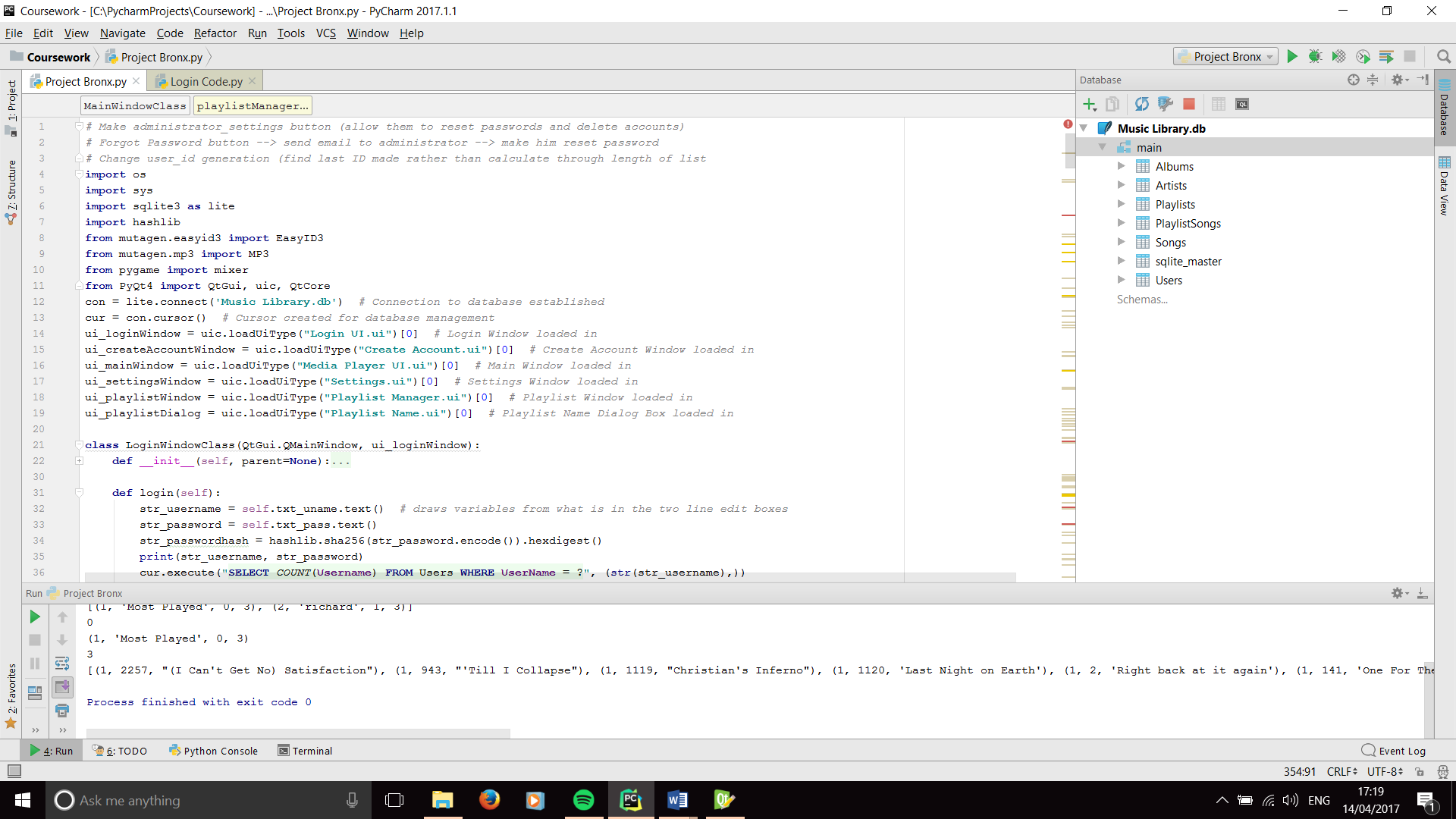
Here is the code for the imported libraries that are used by the program. The lines have also been annotated to show what each library is used for:



### Linking the Database with the code

The database created in 4.1 must be linked to the code before it can be used by the program.

Here is the code used to connect the database:

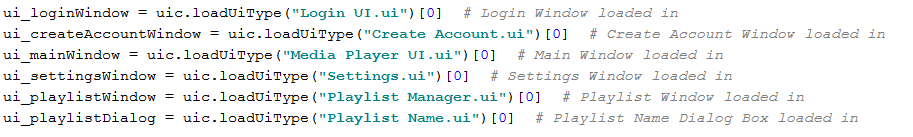


The first line connects the database file (Music Library.db) to the program, while the second line establishes the cursor to allow SQL queries to made for the database.

### Connecting the User Interface Files

The User Interface files are created in QtDesigner through .ui files. The files are then imported into the program.

As the files were added at different points in the development cycle, evidence of iterative development will be shown in later sections. However, the final code of the UI imports are as follows:



## Login Screen Class

### Class Definition

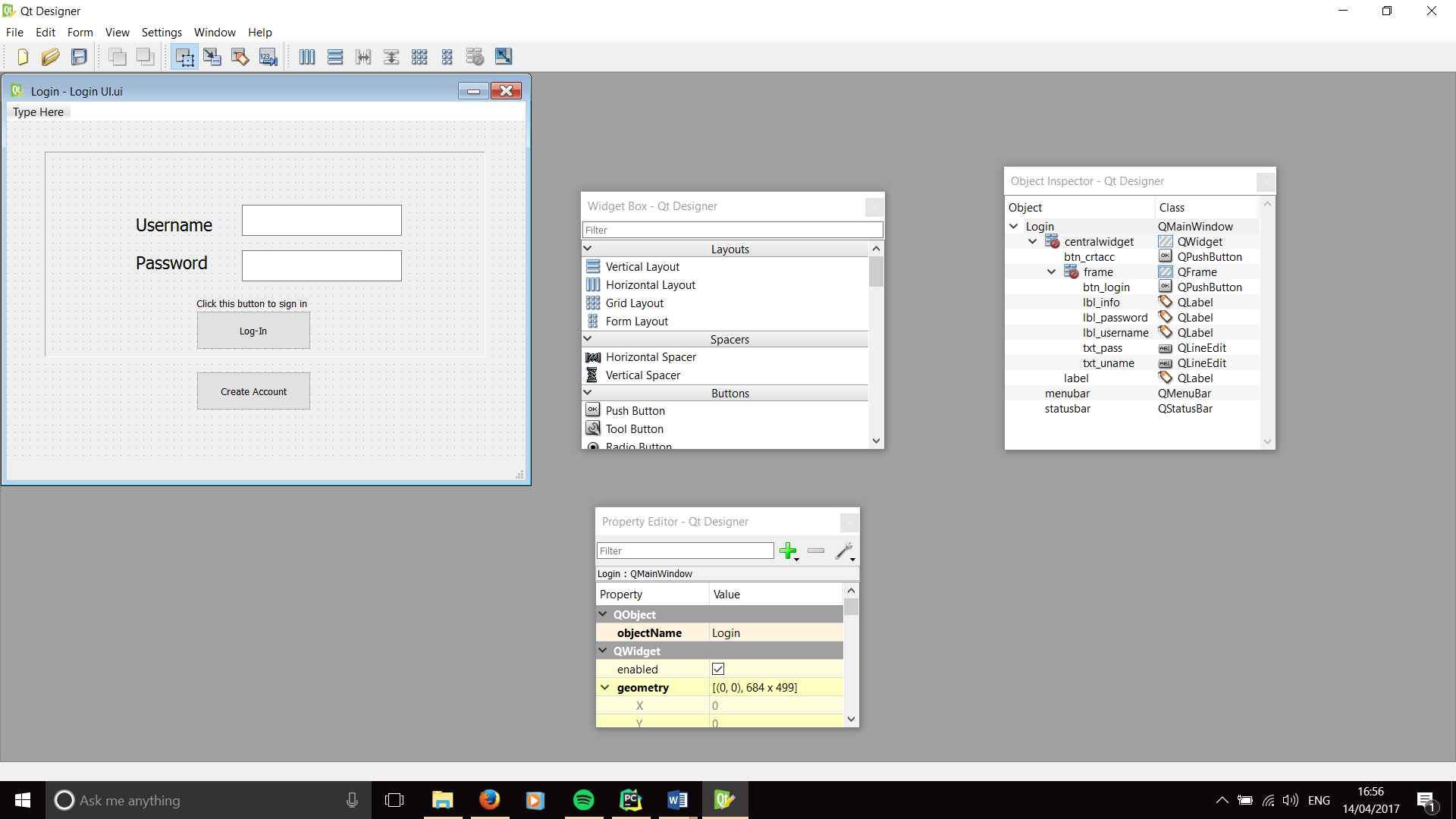
This class will hold all the code required for the ‘Log-In’ section of the program. It will be the first screen/class used by the user when the program starts.

The code will have to take two inputs (the user’s username and password) and validate them against the details in the Users table in the database. As the passwords in the database are required to be hashed, the users input must also be hashed before it is compared to the password in the database. If the validation is not successful, an appropriate message will be displayed to the user. Else, the Login window will close and the Main Window will be opened.

### 1st Iteration

#### User Interface

Before I started to code the class, I created the User Interface for the Login Screen. Changes will be made throughout the development to accommodate more features. In this iteration, I aimed to only cater for the basic login section of this class. Here is the initial look of the login screen:



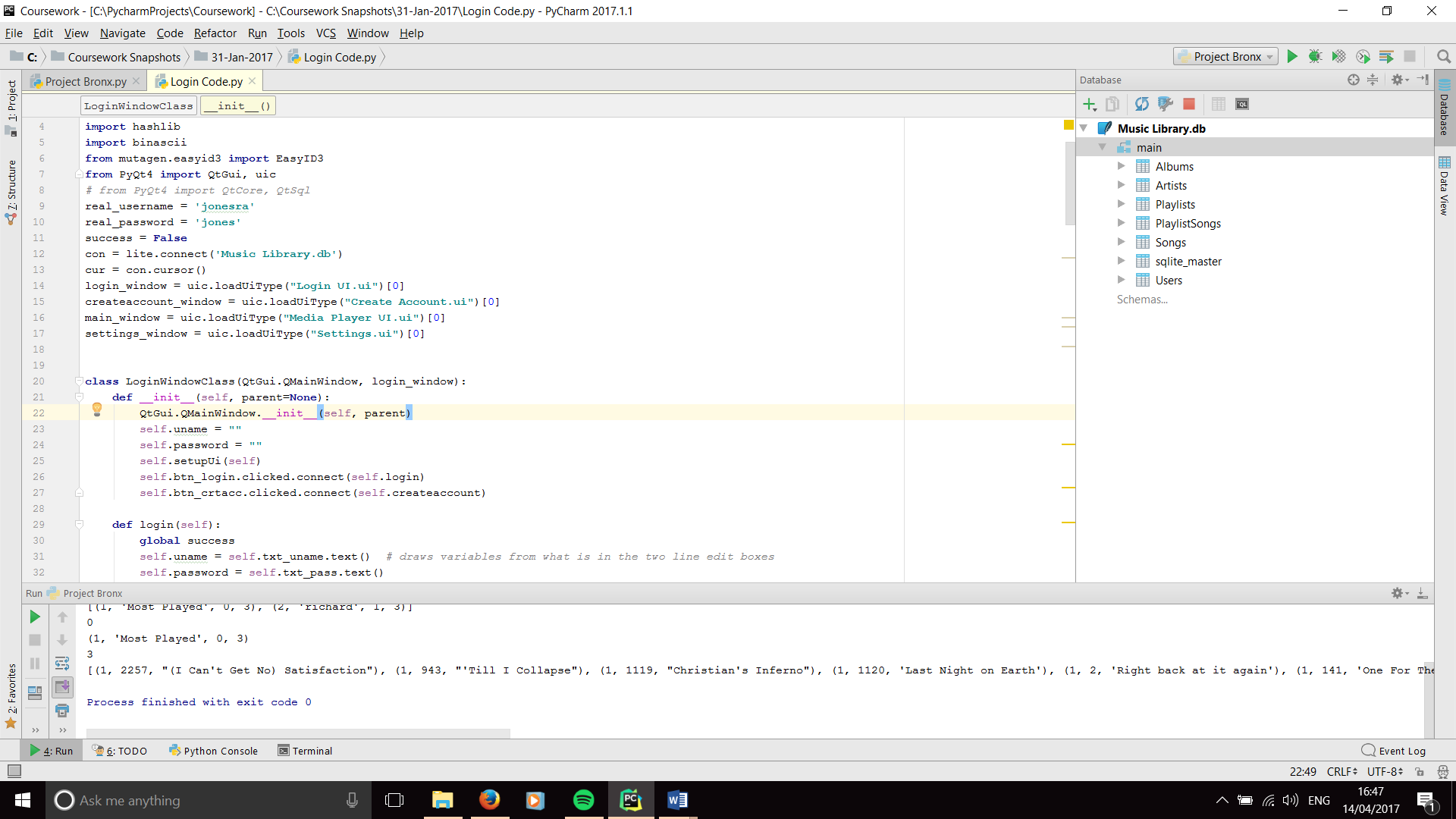
The Window uses several objects from the Qt Framework:

* The textboxes use QLineEdit objects to allow the users to input their usernames and passwords.
* QPushButton objects are used for the Log-In and Create Account buttons.
* QLabel objects are used for the Username and Password labels as well as the sign-in instruction.

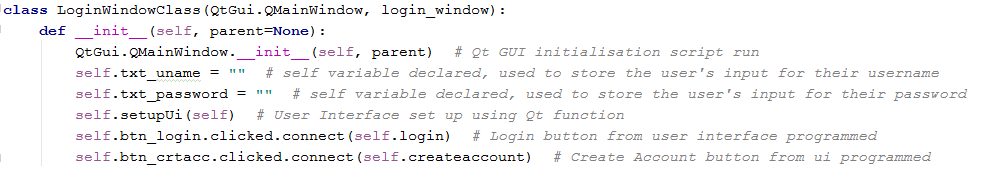
#### Code

For the first attempt at writing this class, I focused on writing just the validation process, with the hopes to implement the database and hashing function later. Here is the first iteration of the code.

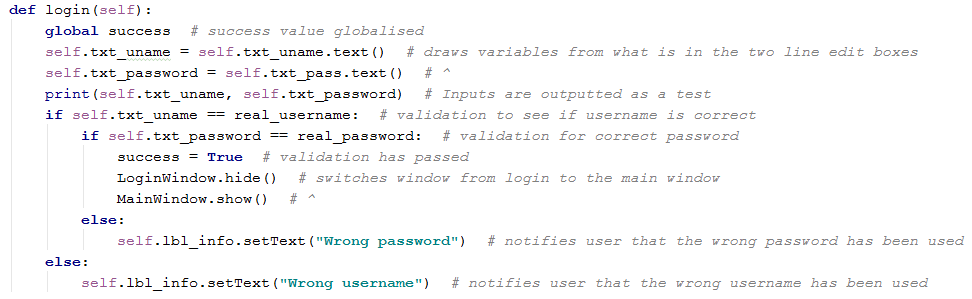
First, the correct username and password were declared outside of the class, as well as the ‘success’ variable for the validation:



Then the shell of the class was created and the initialisation function was written. The initialisation functions in each class will hold all of the definitions of attributes (including the objects in the ui) and the initialisation code for the GUI:



Then the algorithm for the login validation was written. The code retrieves the user’s inputs and validates them against the ‘real username’ and the ‘real password’ that have already been declared. If the usernames or passwords don’t match, an appropriate message will be displayed. Else, the current screen will be closed and the next screen will be opened:



The code has been written using a **Hungarian notation** standard. This is to support maintainability of code.

This code does not represent the full functionality of the program. However, it will still need to be tested.

### Testing for Iteration 01

#### Inputting the correct User Details

For this section, the inputs used will be:

Username = “jonesra”

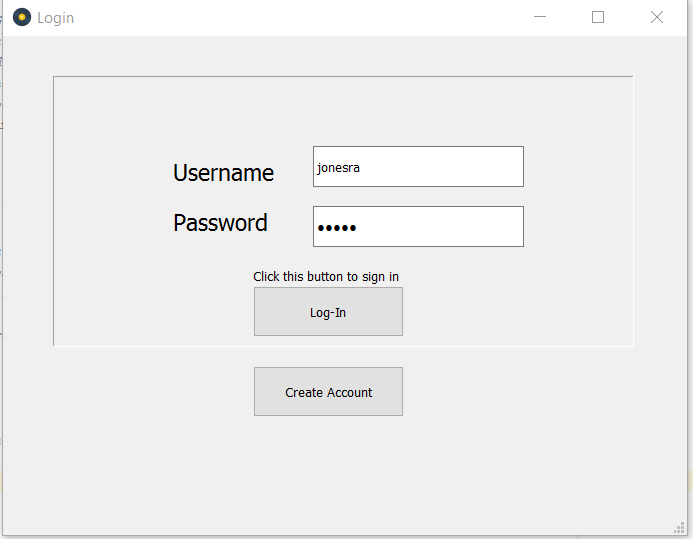
Password = “jones”

And the Login button is activated

As the username and password variables match the ‘real username’ and ‘real password’ variables, the expected output will be that the login window will close and the media player window will open.

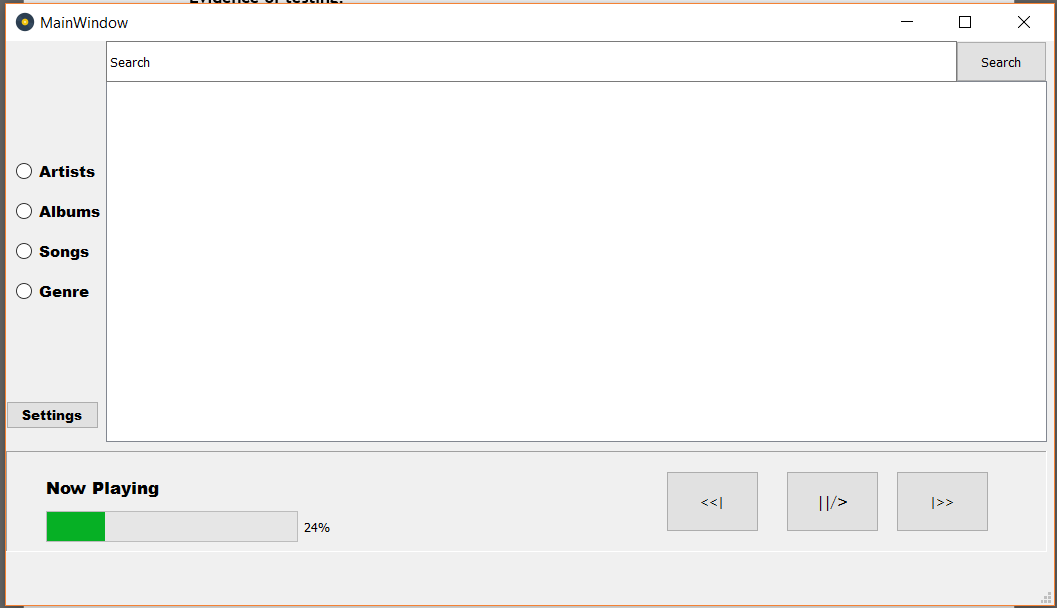
**Evidence of testing:**

**Inputs:**



**Output:**



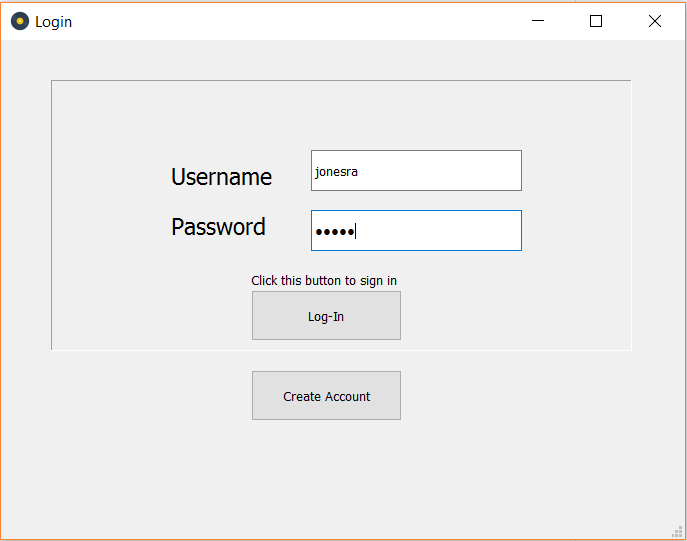


The two pictures show the inputs being inputted and the result of it. The fact that the window changed proves that the test is **Successful**

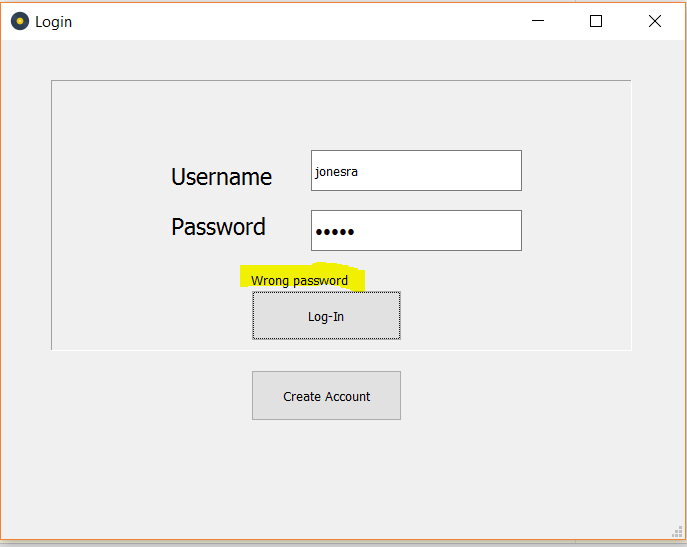
#### Inputting the correct Username and the incorrect Password

For this test, the username input will be “jonesra” and the password input will be “wrong”. The expected result is that the message “wrong password” will be displayed on the screen.

**Inputs:**



**Outputs:**

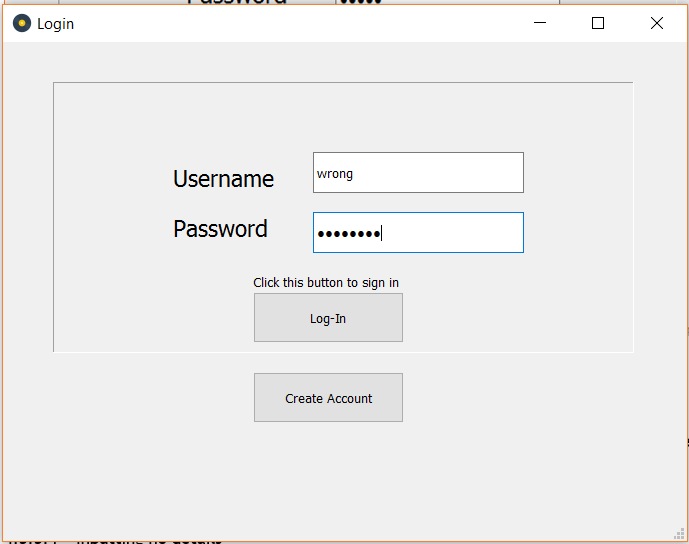


#### Note that a message has been displayed saying “wrong password”, showing the test to be **Successful**.

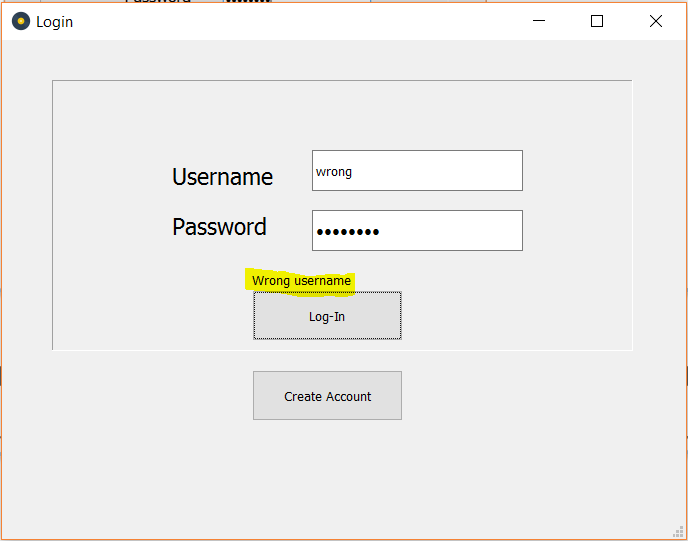
#### Inputting the correct Password and the incorrect Username

For this test, the username input will be “wrong” and the password will be “baritone”. The expected result will be that the screen will display the “wrong username” message.

**Inputs:**



**Outputs:**

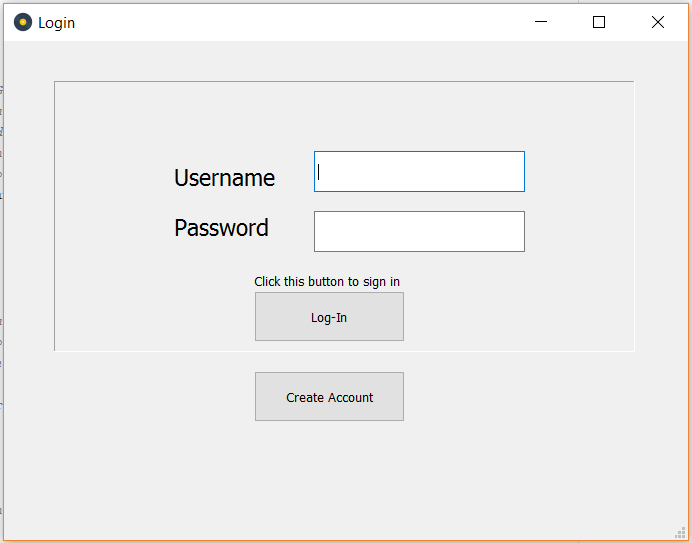


As the screenshot shows, the message “wrong username” was displayed, showing the test to be a **Success**

#### No username or password is inputted

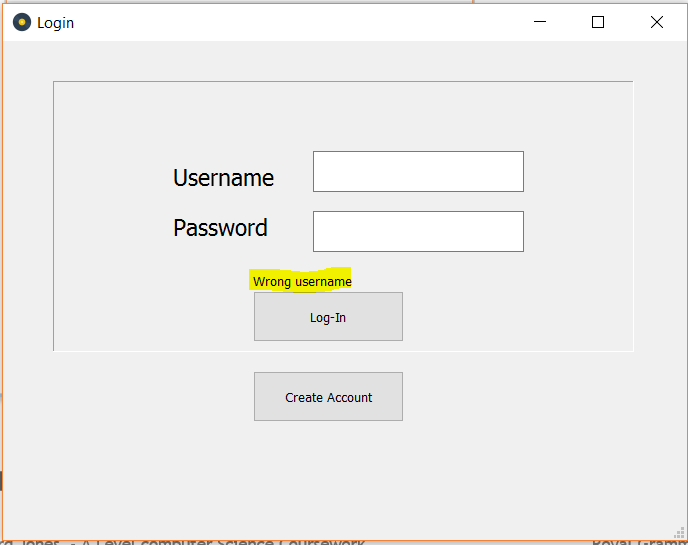
In this test, no username or password will be inputted. Just the login button will be activated. This section of the problem has not been coded yet, so the test is likely to show an error and no output. This will be fixed in the later iteration.

**Inputs:**



As the screenshot shows, there are no inputs in the text boxes.

**Outputs:**



Note that the “wrong username” message is still displayed.

However, there was no error message shown:



This test was obviously a **Failure.** Though no error was shown, a specific message will need to be shown in the final solution to meet all the objectives.

#### Conclusion

These tests show that the login problem is taking shape. However, the code still needs to check with details in the database and hash the password inputs. Furthermore, the testing showed that a specific message will need to be shown if the user doesn’t input anything in the login screen. These changes will be addressed in the second iteration of the code.

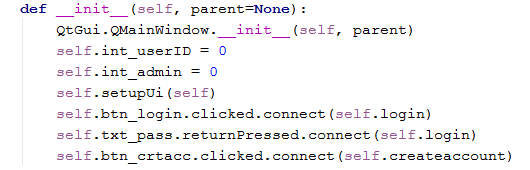
### 2nd Iteration

#### The changes mentioned in the conclusion for the 1st Iteration need to be made in this iteration. These mainly focus around:

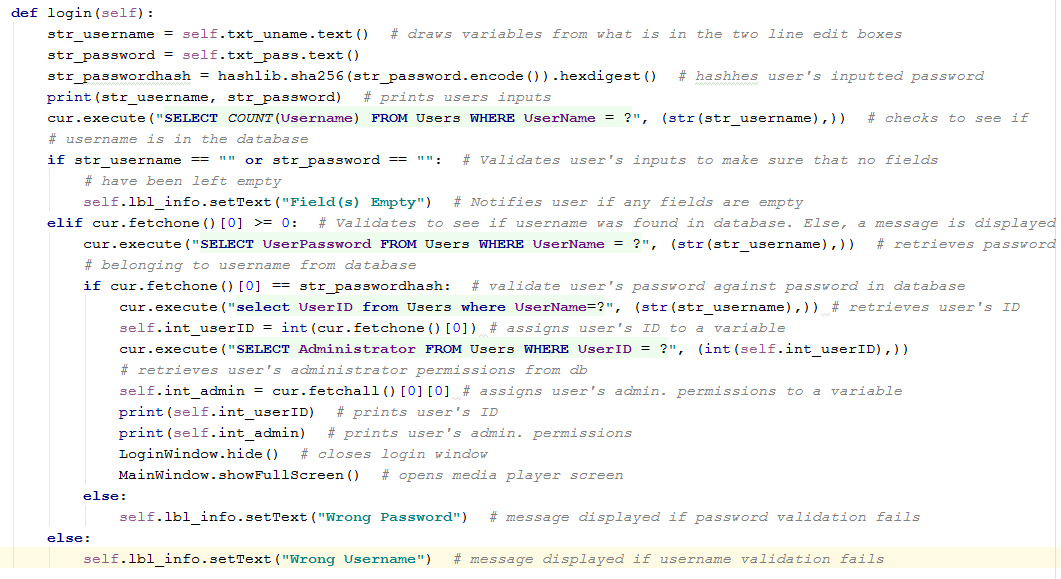
* Checking the login inputs against the users table in the database, hashing the passwords to achieve this.
* Display a specific message if the user doesn’t input a username or password.

On top of this, the User’s ID, admin permission, and username will have to be stored as variables to be accessed by later sections of code.

#### Code



The initialisation section of the class has been slightly improved. First, the username and password sections have been changed from ‘self’ variables to local variables in the login procedure. This was because the variables were not needed throughout the class or the rest of the code. Furthermore, the int\_userID and int\_admin variables were declared. These variables will store the id and administrator permissions of the user who is logged in for each session. The listener for the create account button was also set up to call the ‘createaccount’ procedure when it is clicked.



This code has been extended to meet the changes mentioned earlier. As the annotations show, the code checks the user’s login details against all of the login details in the database using SQL queries. The user’s inputs are validated for three pieces of criteria: that the user has inputted both a username and a password, that the username is in the database and that the user’s password corresponding to the inputted username is correct (using the hashing function to do so).

Furthermore, the code for opening the create account screen was coded:



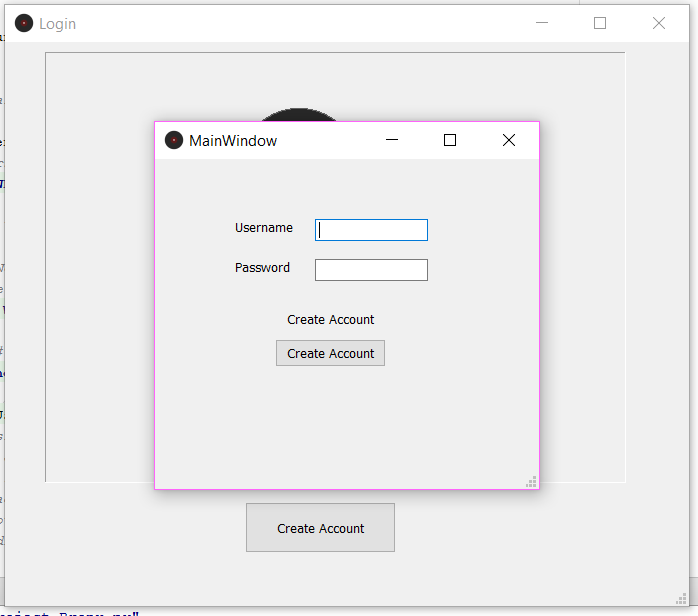
This procedure will simply open the create account window when it is run.

### Testing for Iteration 02

#### Create Account Window

For this test, the create account button was activated. The expected output is that the create account window will show.

**Output:**

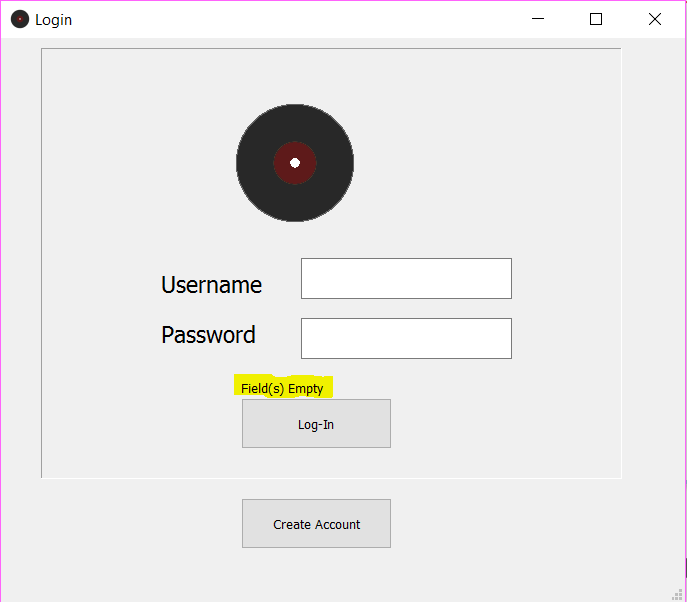


This screenshot shows that the create account window was opened. Therefore, the test was successful.

#### No username or password is inputted

For this test, the login button was activated without any inputs for the username or password. The expected output is that the message “Field(s) Empty” should be displayed in the lbl\_info object.

**Output:**



As the screenshot shows, the message was shown with the given inputs. Therefore, the test was successful.

## Create Account Window Class

### Project Decomposition

#### For this section of the code, the program will need to allow the user to input the username and password that they require for their program.

### Final Code

## Main Window Class

### Project Decomposition

### Final Code

## Settings Window Class

### Project Decomposition

### Final Code

## Playlist Manager Window Class

### Project Decomposition

### Final Code

## Playlist Name Dialog Box Class

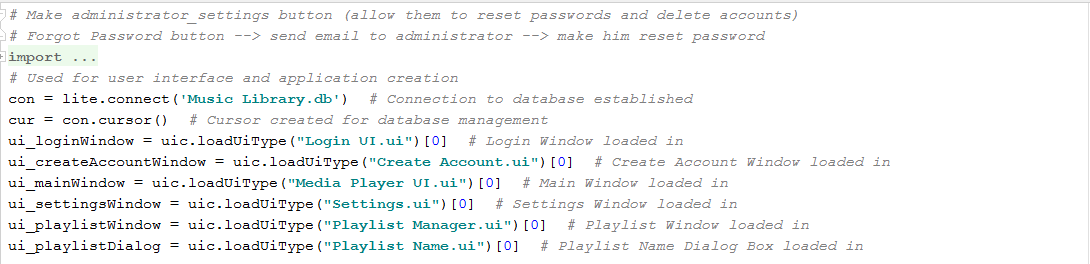
### Project Decomposition

### Final Code

# Evaluation

## Final Code

### Beginning of Code

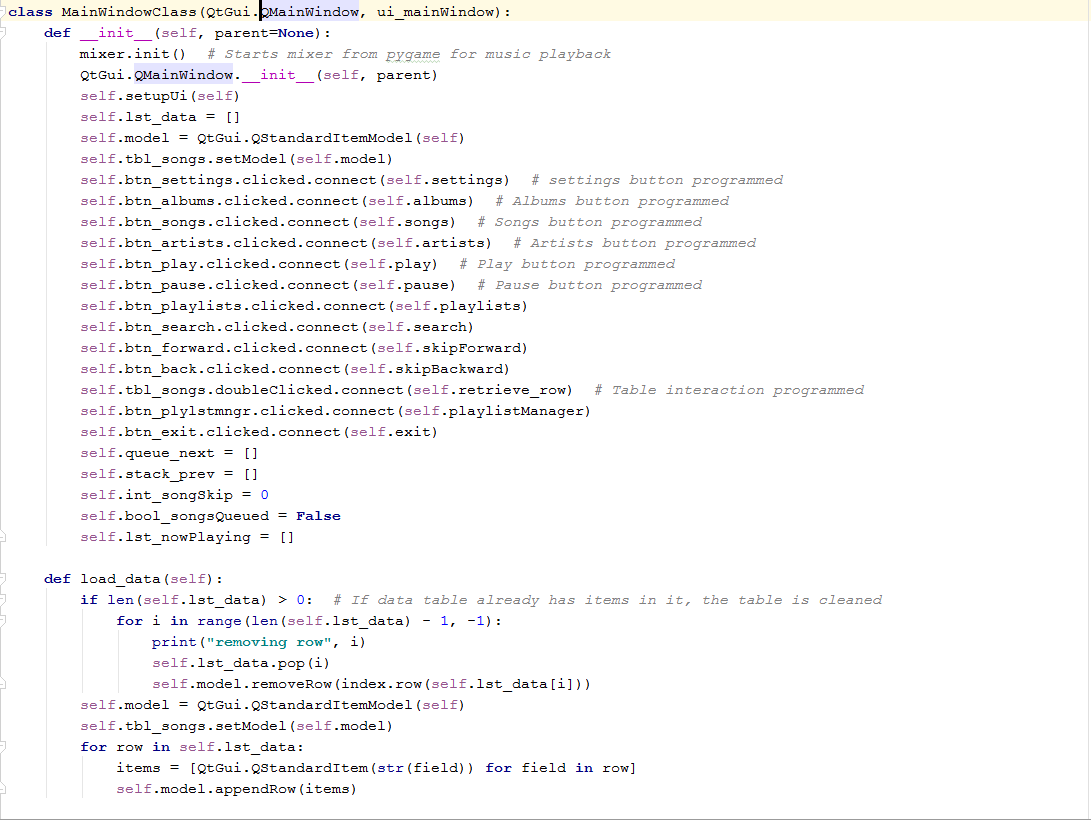


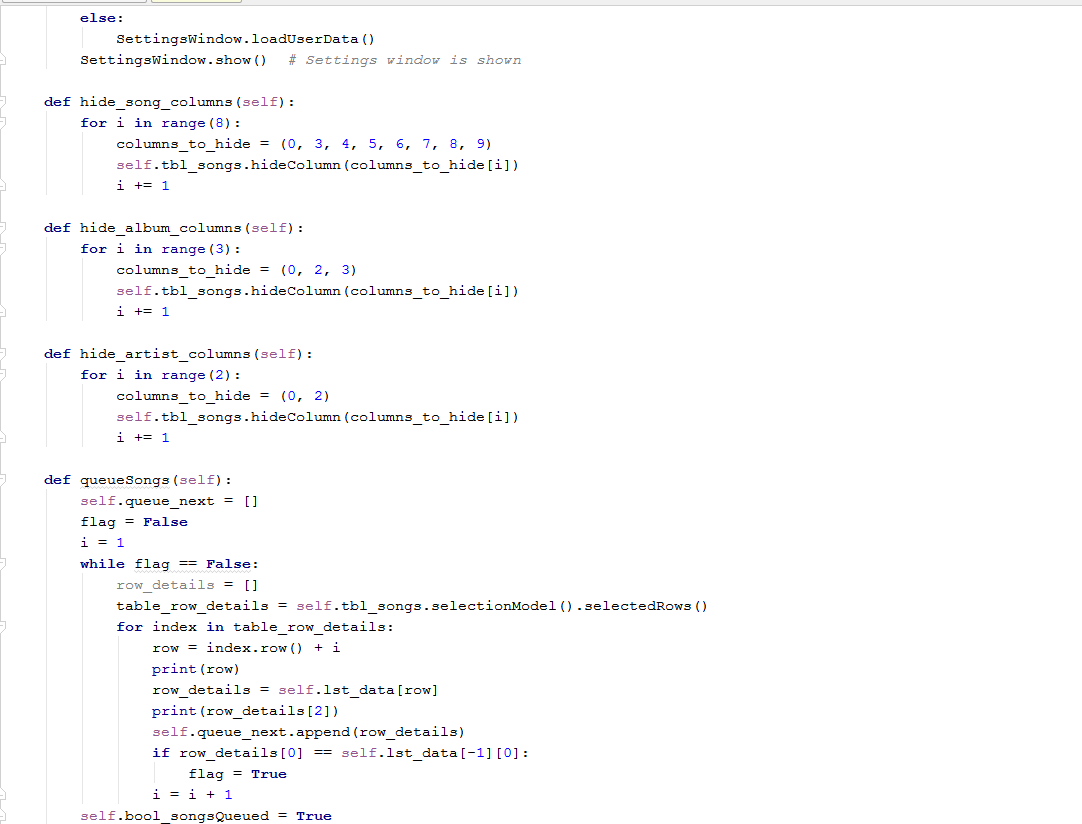
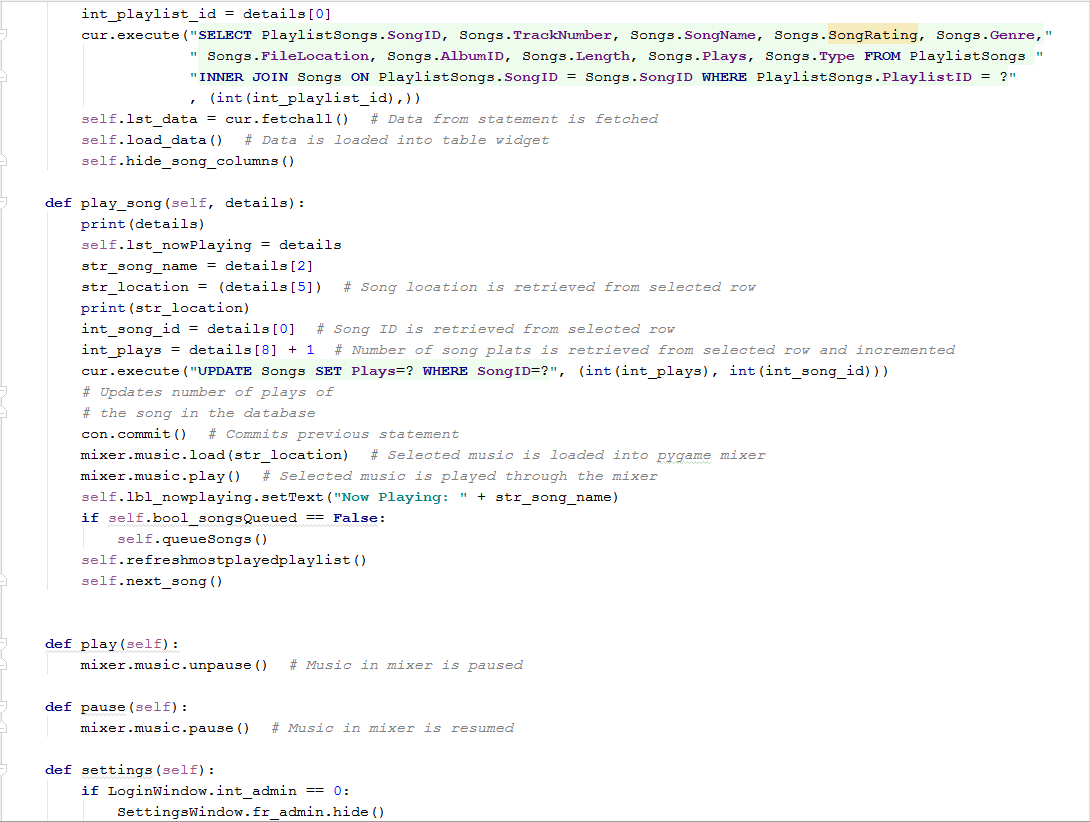
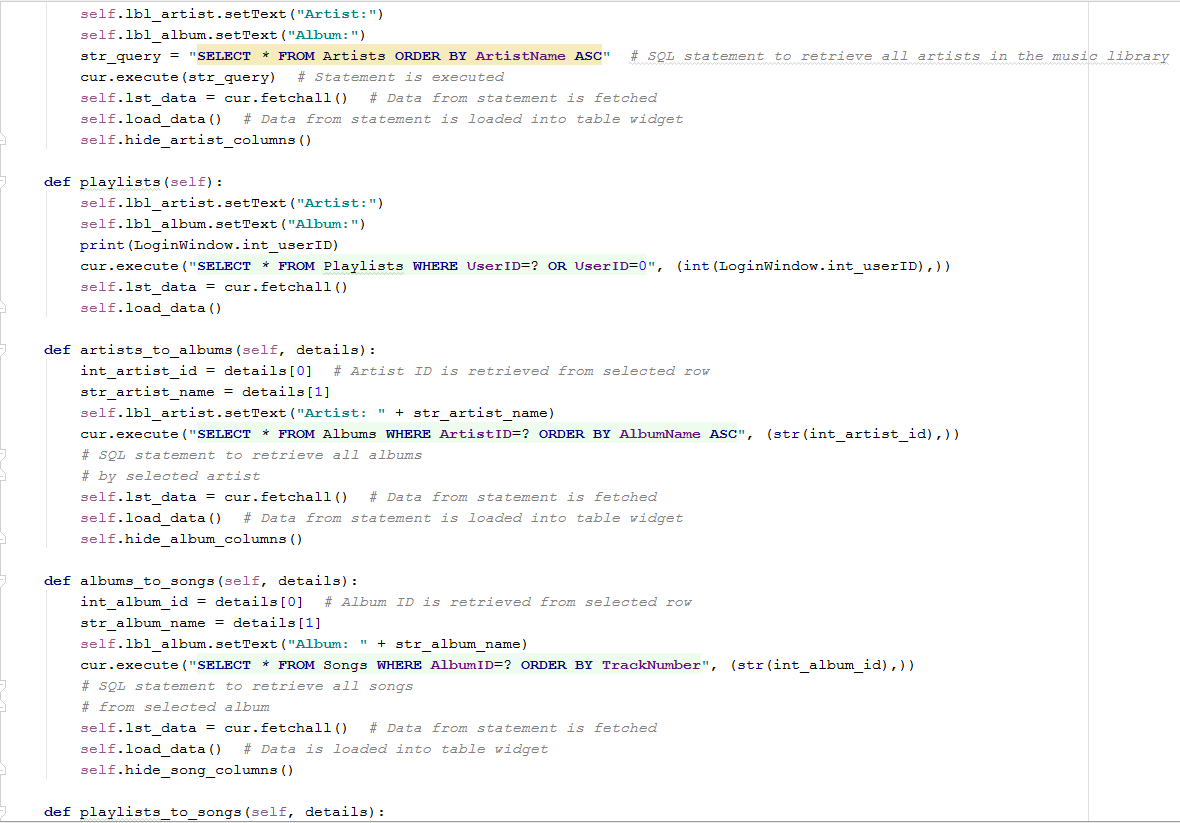
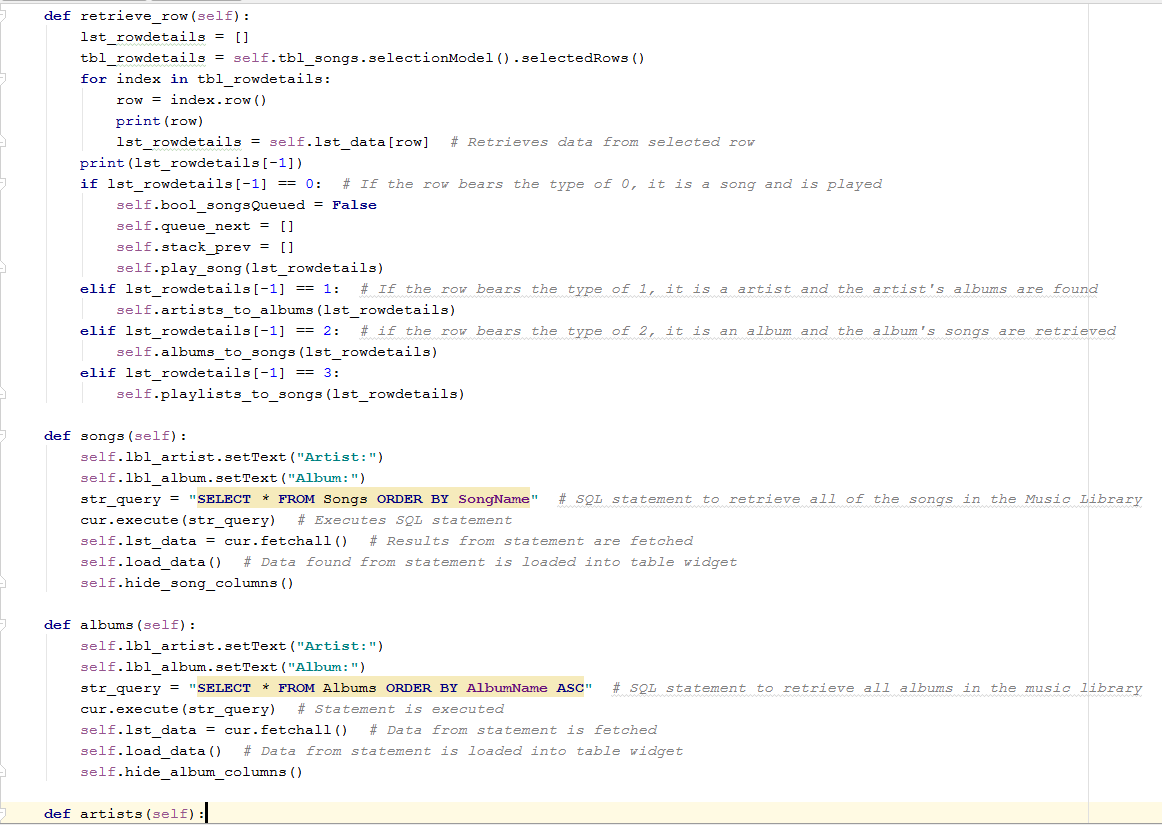
### Login Window Class



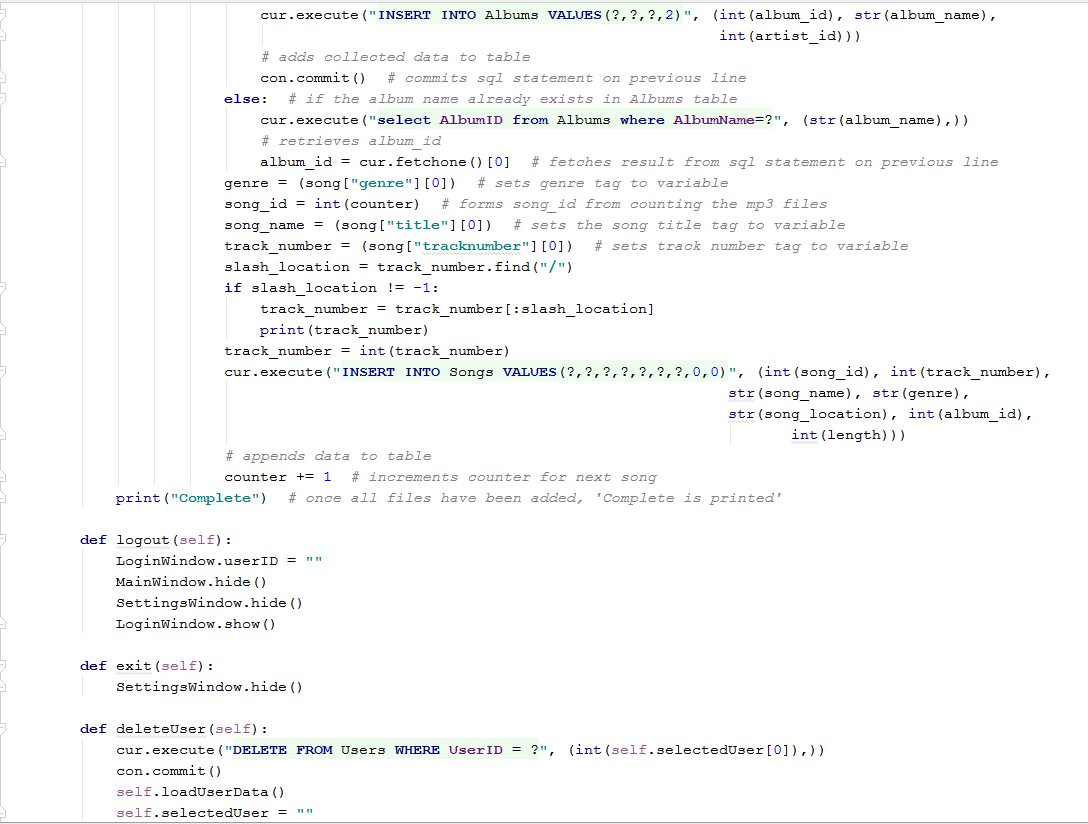
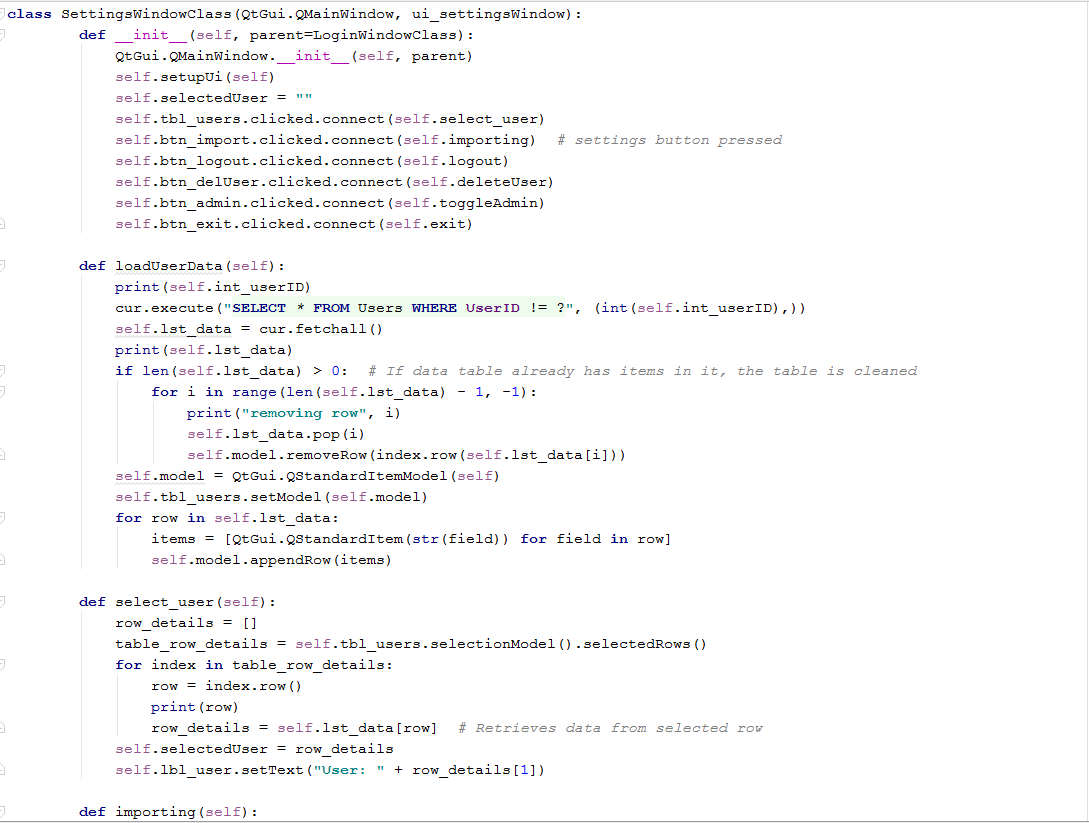


### Main Window Class

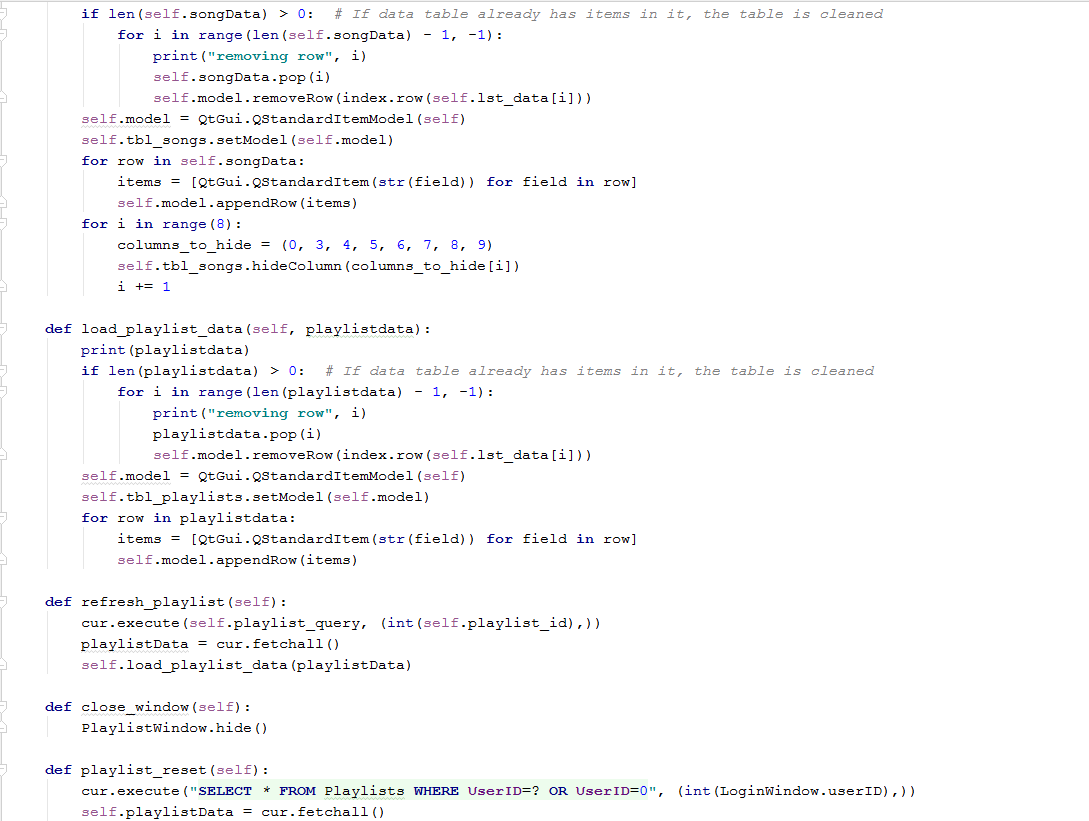
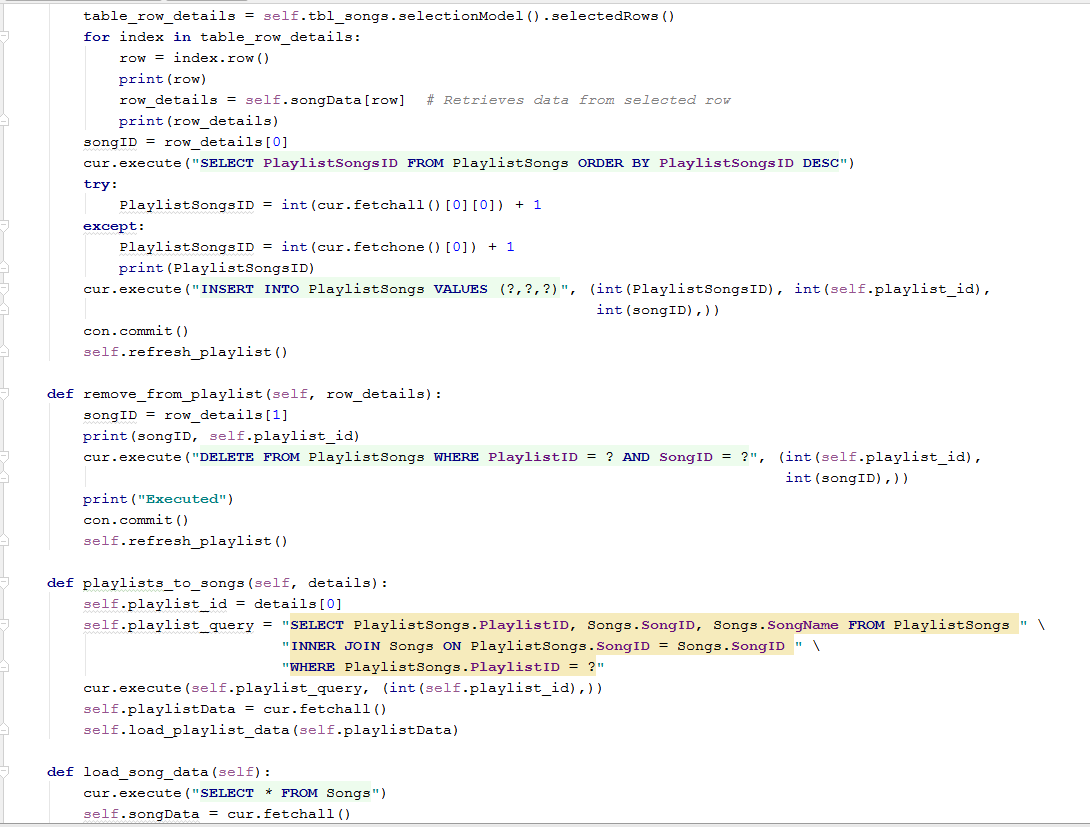




### Settings Window Class







### Playlist Creation Dialog Window Class



## Final Testing

In this section, I will test each of the objectives