DUBLIN INSTITUTE of TECHNOLOGY

Institiúid Teicneolaíochta Bhaile Átha Cliath

Mood Streamer
Final Year Project Report

DT228

BSc in Computer Science

Matthew O'Neill

Supervisor: Dr. Paul Doyle

School of Computing

Dublin Institute of Technology

Thursday, 26th March 2015

Abstract

Music has the power evoke strong emotions in the listener. From a melancholic piano piece which helps the listener through difficult times, to an uplifting pop song which evokes happiness and makes the listener want to dance, music shapes how we feel daily.

This project's goal is to develop an application which will analyse the music collection of its users and categorize the tracks by the emotional traits that it assesses they contain. This will enable the user to select how they are feeling and be played a selection of music collection which matches that mood.

The application was developed as an Android Application which streams the user's categorized music collection from the web service which was developed to store and analyse the user's music collection.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Matthew O'Neill

26/03/2015

Acknowledgments

I would like to express thanks to my supervisor, Dr Paul Doyle, whose guidance, input and patience I was very fortunate to have throughout the entirety of this project.
Thanks also to Dr Susan McKeever for her advice and guidance in the form of weekly seminars.
Finally, thank you to my parents and brothers for their love and support throughout this project and final year.

Contents

Table of Figures

Table of Tables

1 Introduction

1.1 Project Aim

The aim of this project is to develop a software system that will allow the user to select a mood or degrees of multiple moods that they are currently experiencing by inputting this choice in a mobile application. The user will then be played music that the system has deemed to be matching this mood. Many music listeners create playlists of music categorized by how they are feeling, for example they might create a 'Rainy Day' playlist containing pieces of music that evoke in them the emotions of a rainy day. This application will automatically create mood based playlists. Should the user disagree with a tracks categorisation, he or she is offered the option of providing feedback as to how they think it should be categorized, influencing the categorization for both that user and all other users (to a lesser extent) on future uses of the application.

1.2 Structure of this Document

This document is structured into 8 sections (including this introduction); they are as follows:

1.2.1 Section 2 – Research

This section explores the research undertaken before and during development of the system. It first details existing solutions and systems in similar areas to this project as well as outlines pros and cons of these systems relative to this project.
This chapter then goes on to outline the technologies researched before development began, as well as the justification for the choices made in this area.

1.2.2 Section 3 – Design

This section outlines the design methodologies employed in this project. It also looks at the UI design approach taken and compares the approach taken with other possible approaches.

1.2.3 Section 4 – Architecture and Development

This section outlines the overall technical architecture of the system and documents the development of the system that has taken place over the course of this project.

1.2.4 Section 5 – System Validation

Validating the completed system was a major part of this project; section 5 attempts to outline this validation. Validation consisted of comparisons with existing system as well as user testing, automated and unit testing.

1.2.5 Section 6 – Project Plan

This section documents the planning stage of the project and evaluates how well the final system stuck to that plan as well as identifies areas in which it differed.

1.2.6 Section 7 – Future Work

This section outlines some possible future work that may be done on the project, including additional features that may be implemented and improvements to existing features and user interfaces.

1.2.7 Section 8 – Conclusion

This final section attempts to sum up the outcome of this project and offers speculation as to whether the project was successful.

1.3 Challenges Faced

Over the course of this project many challenges had to be overcome in order to reach the desired result.

1.3.1 Subjectivity of Music

Music is a particularly subjective topic; a piece of music may have particular meaning to a listener, such as having been played during a particularly difficult period in that individual's life. Therefore it is necessary to not simply rely on the system's analysis of a piece of music but also include the option for the user to provide feedback which changes the rating of the track in future should the categorisation not match their expectation.

1.3.2 Learning How to Develop Android Applications

While there was an existing familiarity with some of the technologies used in this project, Android Development had to be learned in order to develop the mobile application which forms part of this project.

1.3.3 Use of Xamarin

The use of Xamarin brought some challenges to this project which could have been avoided by using the more traditional method of developing Android Applications – Java and either Android Studio/Intellij IDEA or Eclipse.
All examples on the Android Developer website are using the Java language, and while the languages are fundamentally similar, conventions mean that changes had to be made to the recommended methods of achieving something laid out in the documentation.
Libraries written for the android platform cannot be used without first porting them to the Xamarin platform. This is quite a challenging process and sometimes does not work at all, at one point this prevented the use of a grid library in the Android application; a transparent image of the desired grid was used instead.
Another challenge introduced by the use of Xamarin was that the 'Indie' Xamarin tier, which is the one made available to students, prevents the use of Visual Studio, and instead forces the user to use the included Xamarin Studio. This is a much less fully featured IDE, which crashed many times over the course of this project causing data loss on more than one occasion.
The choice of Xamarin, has yet to present any benefits over simple language preference, as only one platform's application has been developed for this project; Xamarin's main advantage is the sharing of code across applications developed for different mobile operating systems.

2 Research Undertaken and Requirements

2.1 Musical Features Which Relate to Mood

Music theory is the study of the elements which comprise music. The Cambridge History of Western Music Theory describes the field as asking the question "what is the essential nature of music?" [1]
In order to effectively build a system which analyses music to discern some information about it, it is necessary to first understand the fundamental elements which make up the music. It is commonly put forward that there are seven elements of music, they are as follows [2]:

- Rhythm (Tempo, Meter)
- Dynamics
- Melody
- Harmony
- Tone Colour
- Texture
- Form

In order to understand which elements are relevant to the interests of this project, these elements were investigated, the rest of this subsection outlines that research.

2.1.1 Tempo

Tempo, which measures the Beats per Minute (BPM) of a piece of music is perhaps the element of music which is most directly related to the mood of that piece. Tempo markings on sheet music, often make reference to the mood that the tempo of the piece evokes. For example, a piece of music which should be played at a tempo of 150-160 BPM will be marked 'Allegro Vivace', meaning 'happy with spirit' in the Italian language [3].
It was decided that a system which is to detect mood from a piece of music must consider the tempo of that music in order to be effective.

2.1.2 Meter

The Meter of a piece of music, also known as the time signature, tells us how many beats will occur in a measure of the music. This feature plays an important role in how the piece 'feels'. For example, a piece of music in the 3/4 (waltz) time signature will feel differently to a piece in common time (4/4) even if they are played at the same tempo. While these meters feel different, it was decided not to directly use the meter in the mood analysis process, this was due to a lack of correlation between it and mood.
It will, however, have an effect on the process of detecting the tempo of the piece of music. This is due to the similarities between such time signatures as

2/4 and 4/4. Because the beat is the same, but the measure shorter, without knowing the meter of the piece, a piece at 120bpm in 2/4 time could erroneously be detected as being 60bpm in 4/4 time. This difference in tempo would cause huge variation in the mood the track is estimated to be in.

2.1.3 Key

The key a particular track is tells us which scale the notes of a piece primarily fall into.
The Harvard Dictionary of Music says the following about key [4]:
"According to the 12 tones of the chromatic scale, there are 12 keys" and that due to the choice of modality (major or minor) there is "a total number of 24 keys".
For the purposes of this project, the specific key of a piece of music will not be considered, instead only whether that key is a minor or major one, i.e. the modality. For example, the fact that between two pieces one is in D Major, whereas another is in C Major would not be considered. One piece in D Major and the other in C Minor, would be considered, however.

2.1.4 Modality

The modality of a piece music – whether it is in a major or minor key - plays perhaps the largest role in determining whether that piece will be interpreted as either positive or negative.
A piece of music which is in a major key, is often "associated with positive emotional valence (happiness, contentment, serenity, grace, tenderness, elation, joy, victory, majesty…)" whereas a piece which is in a minor key is associated with "negative emotional valence (sadness, anger, fear, tension, solemnity, lament, tragedy…)" [5]

2.1.5 Use of Staccato and Legato

Staccato, meaning 'detached' is where notes are played in a separated fashion, as opposed to legato. Repp described the two techniques as "When one tone is terminated near the beginning of a following tone, the tones are perceived as connected or legato; but when the first tone is terminated earlier, so that a noticeable gap occurs between the tones, they are perceived as separated or non-legato" [6]. Research found that the use of staccato led
It was noted, however, that extracting with this feature any accuracy has great difficulty, and, often will produce results which sound incorrect to the tester's ears meaning the system may need to be developed without this feature as wrong values will lower then usefulness of the system.

2.1.6 Lyrics

While not a feature present in all pieces of music, the content of a song's lyrics can affect our emotional response to that piece of music.

2.1.6.1 Sentiment Analysis

Sentiment analysis is described as seeking "to identify the viewpoint(s) underlying a text span" [7]. While primarily used to identify how people feel about a particular brand, product, or event by analysing text from such mediums as social media and opinion polls, techniques from the field can also be applied to the lyrics of a song.
It was conjectured that should a track contain primarily words which have mostly positive connotations, then it is likely that song is more positive than one which contains more words which are usually attributed as being negative in nature.
In order to prove or disprove this theory, research into the area of sentiment analysis was undertaken.
The most basic approach could be to simply maintain a list of negative words and positive words, and during a song's analysis, each instance of both negative and positive words could be counted, and an overall positivity score be determined by simply subtracting the number of negative words from the number of positive words occurring in the song's lyrics.

2.1.6.2 Obtaining the Lyrics for a Given Song

While there exist many community driven websites where users can add lyrics for a given song, due to the copyrighted nature of song lyrics, there isn't an authoritative database with a freely accessible API in the public domain containing song lyrics. Public APIs, such as ChartLyrics, do exist which are free for non-commercial use, such as this project, they are non-exhaustive and it was decided that a combination of this free API as well as screen scraping the lyrics from community driven song lyric sites was necessary to ensure it would be possible to obtain lyrics for the greatest number of songs.

2.1.7 Focus on Western Music in this Project

Bowling et al. noted [8] in their study comparing emotional expression in Western and Carnatic (South Indian) music, that while "the tonal relationships used to express positive/excited and negative/subdued emotions in classical South Indian music are much the same as those used in Western music", the tones themselves are different; with western music associating music in the major mode with "positive or excited emotion", and that in the minor mode with "negative or subdued" emotion. On the other hand, the Carnatic music focused on in their study uses the Raga in much the same way, which, while "analogous to modes in western music" these are more complex.
Therefore, these differences, in combination with the western background of this project, influenced the decision to focus completely on western music for the purposes of this project.

2.1.8 Summary

In conclusion, in order to produce an effective mood detection model, many of the features outlined above must be acquired about the music added to the system, either by direct analysis, an existing knowledge base, or a combination. Moreover, some musical features spoken about can safely be ignored as they contain little to no relevance to emotion. It was decided that Texture (whether a piece of music is Monophonic, Polyphonic or Homophonic) and Form (how a piece is structured) should be ignored for the purposes of this project.
It was decided that both modality and tempo were the most important features to use in determining the mood of a piece of music and would be the primary focus of the analysis phase. It was also decided to analyse the sentiment of a track's lyrics (if present) and factor this into the emotional rating of a song.

2.2 Extracting Relevant Features from Music

Music Information Retrieval (MIR) is the study of extracting information from the characteristics of music and is a widely researched area in disciplines such as computer science as well as music studies and psychology.

2.2.1 Using Experts, not Algorithms

Many commonly used music recommendation services use a combination of and algorithmic recommendation system and human listeners who categorize the music on the service. This is unfeasible for this project as it is not a commercial venture and there are no employees available to do this. This may have the effect, however, of reducing the quality of the analysis.

2.2.2 ISMIR Conference

The International Society for Music Information Retrieval is "a non-profit organisation which, among other things, oversees the organisation of the ISMIR Conference. The ISMIR conference is held annually and is the world's leading research forum on processing, searching, organising and accessing music-related data." [9]
The proceedings from this conference are made available online and proved very useful during the research of this project.

2.3 Alternative Existing Solutions

When evaluating similar existing solutions in the area of mood recognition music players, several criteria were used to assess the feasibility of developing a system in this field.

Some existing systems perform mood analysis only on the tracks found on the user's device, using similar techniques to ones which are to be employed on this project: tempo and beat detection, key recognition and pitch analysis. This localized approach has the downside of limiting the tracks a user may listen to those on the device; this in turn is limited by the relatively low storage capacity found on many devices today. A further limitation of using only those tracks that reside on the device is the inability to learn from other music tracks. A sophisticated system in this area is one that can learn from other music and build ratings models from large amounts of music. As such, a system that learns from such a limited number of tracks might not be as accurate as one that can learn from a user's entire music library, as well as the libraries of other users.

Other applications do not perform any analysis on the actual file to be played, but instead consult an existing online database of rankings for tracks. This method has the drawback of potentially trying to ascertain the mood of a track in a user's library that has not been ranked by the system.

Other applications do not allow the user to use their own music collection at all; instead choosing music from an existing streaming service. This practice comes with some downsides, one of which is cost. Should the user be streaming music that has not been bought by them, it will usually need to be paid for. Other applications have circumvented this barrier by streaming from popular free music streaming sites such as SoundCloud and BandCamp, where the music artist allows their music to be listened to for free. These services suffer from a greatly reduced music selection as most record labels and musicians do not offer their work for free.

A requirement of this project is to ensure the user can use his or her entire music collection. This choice, rather than maintaining a collection of music that all users listen to was made in part because no streaming service can ever be complete; even services such as Spotify do not contain every track a music fan of a particular genre wishes to listen to.

2.3.1 SensMe

2.3.1.1 Introduction

SensMe is an application developed by SONY, which has been included on a selection of their MP3 players, smartphones, and games consoles since 2009. This application allows the user to transfer their music collection to the device. Once music has been added to the device, the software analyses a subset of the

music according to such factors as beats per minute (BPM) and key. Once analysis has been performed on the tracks, the software visualises them as small white dots and scatters them on a four-axis graph. These axes are labelled 'Happy', 'Sad', 'Fast' and 'Slow'. Once the user touches a point on this graph, a playlist of tracks which have been deemed to match the labels of the axis/axes closest to the contact point.

2.3.1.2 Evaluation

A downside to the SensMe system that shall be addressed with this system is the storage limitations present on the devices on which it runs. As no music is stored on the client, but rather on the server ready to be streamed to a client device, much more storage space can utilised. Another advantage of this system over one which analyses local files only is that other users' files can be analysed alongside one another, enabling the system to learn.

2.3.2 Moodagent

2.3.2.1 Introduction

Another existing application, Moodagent, which can be downloaded from the Google Play store for Android devices and from the Apple App Store for IOS devices addresses the task in a slightly different manner. Instead of performing the analysis locally, the application consults a pre-existing online database of mood ratings for a track. This has the advantage of not being limited to learning from the relatively small set of music added by the user. The user is presented with a series of sliders labelled 'Sensual', 'Tender', 'Happy', 'Angry' and 'Tempo', and sliding these up or down adjusts the mood of the playlist of tracks to be played to the user accordingly.

2.3.2.2 Evaluation

Moodagent limits the track selection that can be played to those which are on the device, greatly limiting selection, it does however allow the user to hear new music similar to the current mood-based playlist and gives them the ability to buy it on services such as Amazon.com. It also limits the rating of a user's music to those tracks which have already been rated and stored in the Moodagent database.

2.3.3 StereoMood

StereoMood is an Android and iOS application as well as web-based music streaming service which, according to the company's website "plays music

tailored to your mood and daily activities" [10]. The music recommended by this service is aggregated from music blog postings and streamed using the popular SoundCloud music hosting platform.

2.3.4 Conclusion

As a result of comparing applications that presently exist in the area of music players which play tracks based on their mood it was discovered that while a combination of such applications offer all the features of this project, one which offers all of them was not found.

Feature	SensMe	Moodagent	StereoMood
Streaming from the Web	No	No	Yes
Analyse local files before they're played	Yes	No	N/A
Use user's own music collection	Yes	Yes	No

2.4 Technologies Researched

Many decisions had to be made as to the technologies to employ for this project as it consists of three distinct software components that all need to communicate together effectively.

2.4.1 Mobile Development Platform(s)

When it came to deciding on the mobile platform to develop the application, certain factors needed to be taken into account. These factors included:

- Platforms which run on devices currently accessible
- Cost of licences for those platforms
- Additional hardware required to develop for a platform (Mac OS is required to develop applications for iOS, for example)

It was decided that initial development will be done for the android platform as there is no cost to develop for it. Development took place using the Xamarin [11] toolkit for mobile application development. It was decided to use this technology as it allows for the development of an application for other platforms, namely Windows Phone and iOS further into the project. A decision

had to be made whether to develop native applications for one or more platforms using the Java language for Android, Objective C or Swift for iOS and C# for Windows phone, or to use Xamarin to write for any of these platforms using the C# language and the mono runtime. It was decided to use Xamarin for a number of reasons. One such reason is that a shared code base can be used across all platforms for functionality which does not relate to the user interface; UI features are largely specific to the mobile platform, but all are wrapped in C# for uniformity.

Another reason Android was chosen as the first of the mobile platforms to develop for was its dominance of the mobile operating systems market share; more mobile phones run Android than any other platform [12], therefore the most users can be reached by developing for this platform initially.

An influencing factor in the decision to use the Xamarin platform was an existing familiarity with the C# language and the .NET framework, which the Mono framework attempts to emulate. This familiarity with these technologies will lead to increased productivity during the development of the mobile application, as well as cleaner, more idiomatic code.

While Xamarin affords the developer the opportunity to write mobile applications in the C# language, it is still necessary to become familiar with the APIs specific to each platform; Xamarin doesn't re-implement these, it merely affords idiomatic C# access to them. As Android was the primary development platform throughout the course of this project, some time was spent reading the android tutorials found on the developer.android.com website. These tutorials and supporting documentation primarily use the Java language, however given the similarity between the two programming languages, they proved highly useful in the development of the Android application for this project.

2.4.2 Android Emulators

2.4.2.1 Google Android SDK Emulator

Provided with the Android SDK used to develop Android application is an Android device emulator. The performance of this emulator is extremely poor as it attempts to emulate the ARM instruction set while running on an x86_64 based desktop computer. The poor performance of this emulator was deemed unacceptable as it slowed down the development of the application greatly. For this reason an alternative was required.

2.4.2.2 Genymotion

Genymotion [13] is a third part commercial vendor of android device emulators which can be installed as a virtual machine into software such as VirtualBox. While the performance is much better than that of the standard Google emulator, this solution was decided to be unfeasible and cost-prohibitive for use on this project.

2.4.2.3 Debugging on a Physical Device

Debugging on a physical Android device was discovered to be the most efficient method of development for the Android platform as well as the most convenient due to possession of a mobile phone running the latest version of the Android operating system, 5.1 ("Lollipop"). The phone used for development was an LG/Google Nexus 5.

2.4.3 Novel Methods of Interaction with Mobile Devices

Many smartphone and tablet devices on the market today come equipped with a multitude of different sensors. One use for such sensors is to detect movement of the device itself. This provides the opportunity to take such movement and use it as input for of an application. It was decided to utilise the accelerometer of the device to introduce a novel way of skipping to the next track.

2.4.4 Database Selection

In the planning stages it was decided that two databases were needed for this project; one for the web service and one for the Windows upload client, with a possible third should the settings facilities provided by the mobile platform not be sufficient to store the information required by the client.

2.4.4.1 NoSQL Databases

In recent years NoSQL (Not Only SQL) databases have seen an upturn in interest and adoption. One such database system is MongoDB. MongoDB is, according to its website [14], "a document database that provides high performance, high availability, and easy scalability". Rather than interacting with this database using SQL, the developer or DBA instead uses mongo's built in functions, such as find, findOne and insert.
Moving from a relational database to a NoSQL database such as MongoDB can prove a challenge to users used to the concept of joins.

2.4.4.2 Open Source Relational Database Systems

A more traditional data storage route is that of the Relational Database Management System (RDBMS).
PostgreSQL offers some advantages over MySQL, one such advantage is its support of multiple [15] procedural languages for the development of stored procedures and triggers, including python, and Java using a third party plugin.

Another advantage is that PostgreSQL offers many datatypes not found in its open source competitors, such as a JSON type for inserting and querying data found in Java Script Object Notation blocks.

2.4.4.3 File-Based Relational Databases

SQLite is a small, lightweight relational database that stores the entire contents of the database in a single file.
A downside to SQLite is its reduced concurrency support relative to a larger RDBMS, meaning it is not well suited to an environment which may have many simultaneous connections to the database at once, such as the database used by the web service for this project. While there would be no such difficulties with the low number of users the system has at present, it was decided that its use may cause difficulties should the application need to scale in the future.

2.4.4.4 Conclusion

It was concluded that a relational database be used on both the upload application and the web service. This was the case for the web service as much of the data to be stored by the service was thought to be of a relational nature, and it was seen to require a number of joins in queries. The database selected for this component was PostgreSQL due to the advantages laid out in the above comparison.
A relational database, SQLite, was also employed in the Windows upload client. This choice was not made primarily due to its being a relational database, but rather because it is a lightweight, single file-based database, meaning it can easily be packaged with the application, without the need to install a database server or connect to one remotely. The database file can be stored in the AppData directory found on windows on first run of the application and accessed again on each subsequent run.

2.4.5 Version Control Selection

Version control is an invaluable tool for software developers which enables them to keep track of all changes they have made to the files in their project and easily revert to a previous version of a file should a change which introduced problems have taken place.

2.4.5.1 Distributed Version Control vs Centralized Version Control

There are two prevailing styles of Source Control systems: Distributed and Centralized.
Git is a distributed version control system which was developed by Linus Torvalds, the author of the Linux kernel, initially for use in developing that kernel. Git allows the developer to work offline as information about all

revisions of every file in the repository is stored locally in a '.git' folder in the root of the repository. This folder does not get very large however, as only the differences between the files is stored, not each version of the file itself.
Other distributed version control systems include mercurial and bazaar.
Apache Subversion (SVN) and Perforce are centralised revision control systems where there is a central server containing the main copy of the code in the repository. Developers 'checkout' files from this server to get a local copy of that file on their machines. Once changes have been made, the files are committed back to the central server. A problem with this method is that should the developer be away from a network connection, the latest versions of any files are unable to be accessed and no work can be done.

2.4.5.2 Conclusion

It was decided to use the git version control system due to the features it provides above that of SVN and perforce outlined above, as well as an existing familiarity with this particular tool. GitHub was chosen to remotely host the code repository for the project as it provides very useful features such as issue tracking, as well as private repositories, five of which are free for use by students for educational purposes.

2.4.6 Virtual Private Server (VPS) Companies

Rather than deploy the web service to a local server running on a home network, it was instead decided to rent a VPS from a 'cloud computing' provider.

2.4.6.1 Amazon Web Services EC2

Amazon as part of their AWS package provide EC2 (Elastic Compute Cloud). Users can activate 'instances' from a template of server images and are billed by how much computing time their instances use. Users can select the number of CPUs and amount of RAM that should be allocated to their VPS and can 'scale' this amount should it prove necessary due to increased performance demands.

2.4.6.2 DigitalOcean

DigitalOcean is a company which provides "Simple cloud hosting, built for developers." [16] All plans come with SSD only storage, and start from $5USD per month for a VPS with a single core processor, 512MB of RAM, 20GB of storage and 1TB of data transfer. As a student, one can receive $100 in free credit from this company in combination with a GitHub education account. This amount is enough to run one VPS on the lowest performance tier for 20 months.

2.4.6.3 Conclusion

It was decided to use DigitalOcean down to their pricing structure, use of SSD only storage, as well as existing free credit received from the use of a Student GitHub account. The interface was also judged to be simpler to use than that of the other choices. However, it was noted that any of the options presented in this subsection were judged to be adequate for use in this project and the choice was largely down to personal preference.

2.4.7 Storage Options

As the storage provided with most Virtual Private Servers is not enough to store large amount of data, it was decided to investigate cloud storage providers which would offer enough storage for the music collections of the users of the system.

2.4.7.1 Amazon S3

Amazon S3 [17] is a storage service offered as part of Amazon's AWS (Amazon Web Services) cloud platform. With Amazon S3, users are assigned 'Buckets' in which they can store computer files as large as 5TB.

2.4.7.2 Microsoft Azure Storage

Azure Storage [18] is a competitor to S3 offering up to 500TB of storage per account. It is part of the Azure suite of cloud tools offered by Microsoft.

2.4.7.3 Conclusion

It was decided that either of the two storage options investigated here would be sufficient for storing the large number of music files required by the system. Amazon's S3 was chosen for use in the future as the system scales due to pricing and familiarity with the service.

3 Design and Methodology

3.1 UI Design

3.1.1 Client Mobile Application

The user interface of the mobile application, it was decided, should adhere to the guidelines of the platform it is running on. As Android is the first platform developed for, the guidelines found on Google's User Interface Guidelines web page [19]. The purpose of following these guidelines is to retain a familiarity with other applications a user use, in order to minimize the learning a user has to undertake before using the application.
The first screen the user is presented with is the login screen. If the user has left the application previously while logged in, they will automatically be re-logged in to the service and this screen skipped.

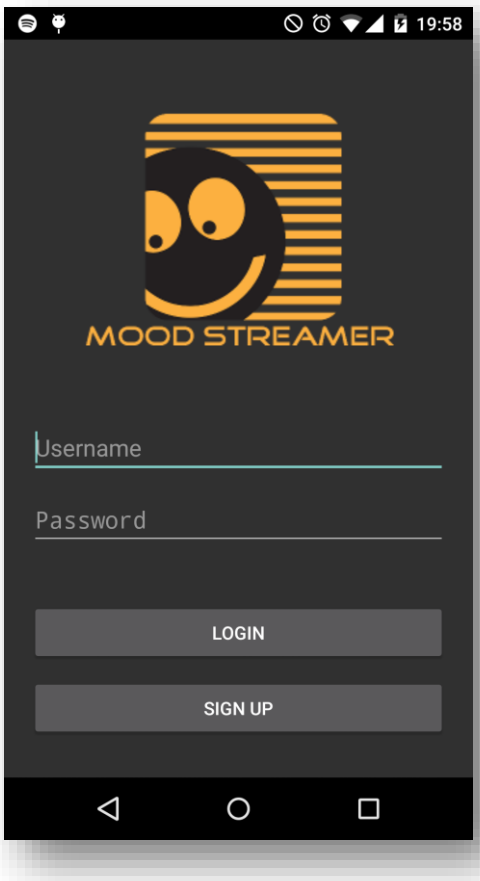

Figure 1: The Login Screen of the Mobile Application

In order for the user to be able to login, they must first be registered with the system. This is performed in-app by pressing the 'Sign Up' button. Once pressed, the user is taken to the registration screen where they must enter a username, password and email address.

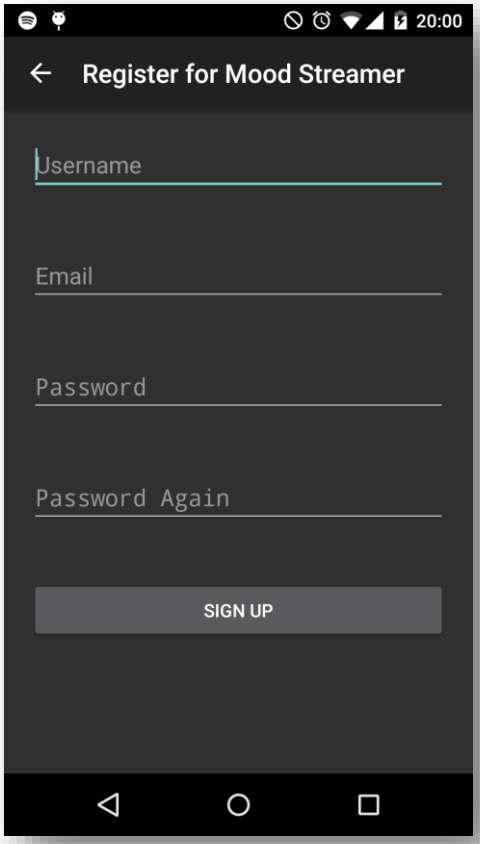

Figure 2: Registration Screen of the Mobile Application

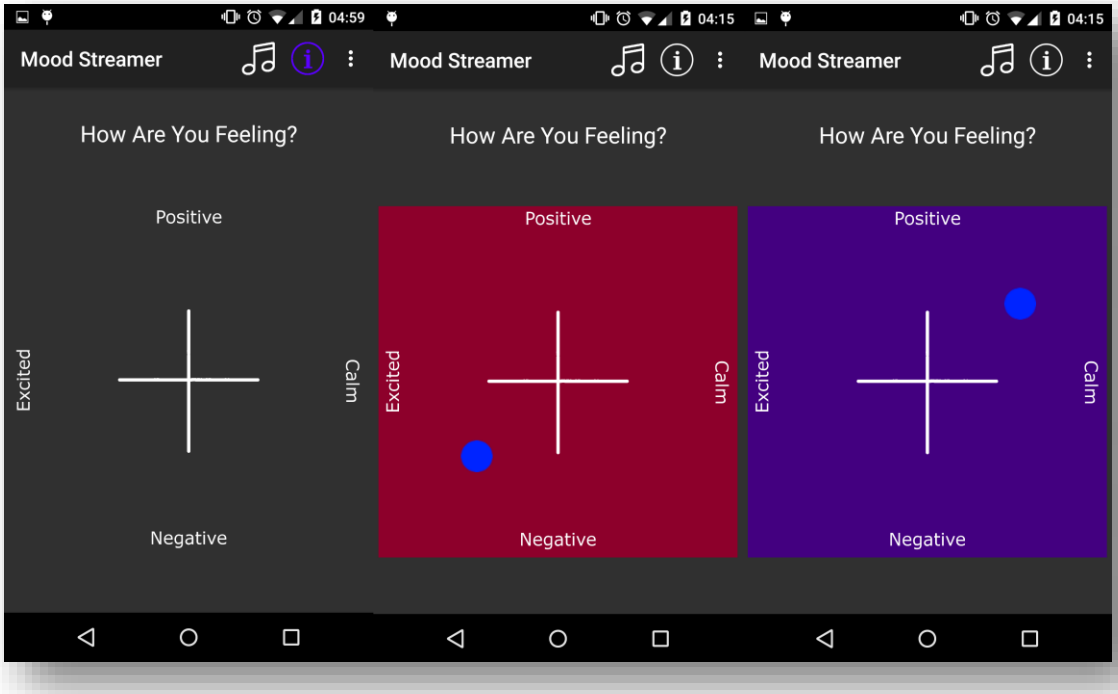

Figure 3: The main screen of the application. From Right to Left: No selection; 'Excited' and 'Negative'; 'Positive' and 'Calm'

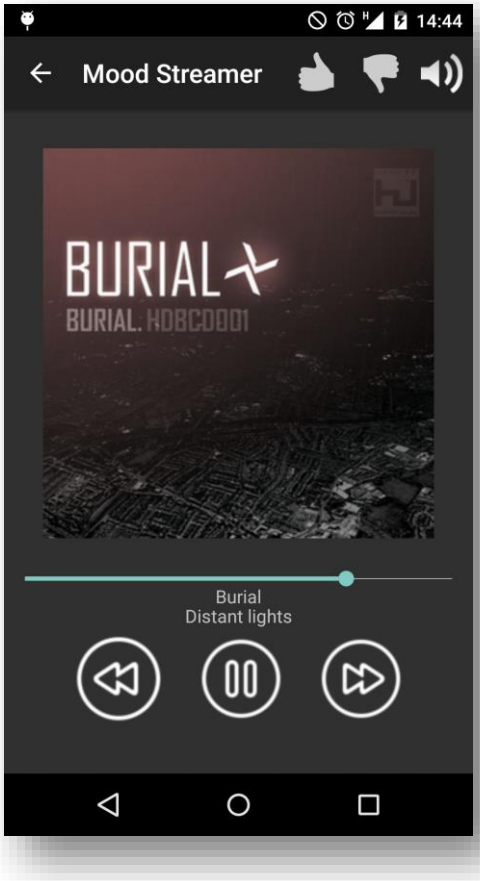

Figure 4: The Mobile Application Streaming a Track by Burial from the Web Service Which was deemed to match the Selected Mood

On the player screen when music is streaming the user is presented with the usual music player options:

- Play /Pause the current track
- Skip to the Next track in the mood playlist
- Go back to the previous track

As well as these familiar options, some new options are presented to the user.

3.1.1.1 Suggest Improvement to the Current Track's Classification

If the user taps the 'thumbs down' icon on the action bar at the top of the screen, a popup dialog appears allowing them to select from four options:

- More Positive
- Less Positive
- More Excited
- Less Excited

When one of these four buttons is tapped the corresponding feedback is set back to the web service with the goal of improving this track's mood rating in the future.

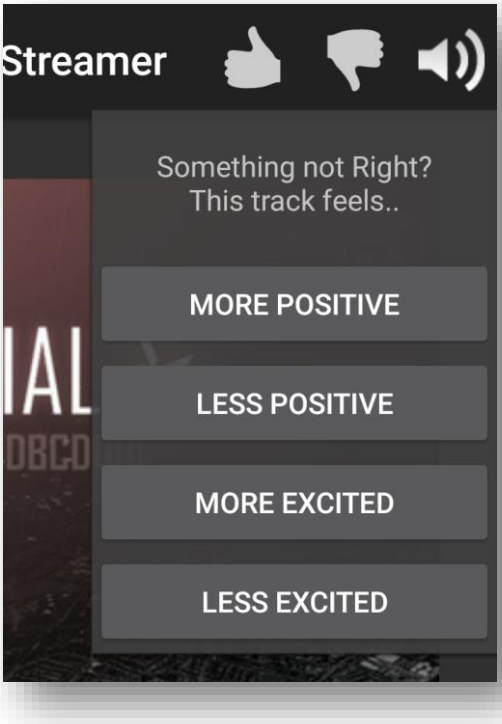

Figure 5: The Popup Window Shown when the User Presses the 'Thumbs Down' Button

3.1.1.2 Agree with the Current Track's Classification

Similarly, if the user believes that the currently playing track accurately fits the mood they requested from the application, then the user can tap the 'thumbs up' button in the action bar in order to indicate this with the goal of strengthening the rating this track has been given.

3.1.2 Adjust the System Media Volume

For convenience, there is a volume icon located in the action bar of the player screen which when pressed shows the system's default slider for media volume.

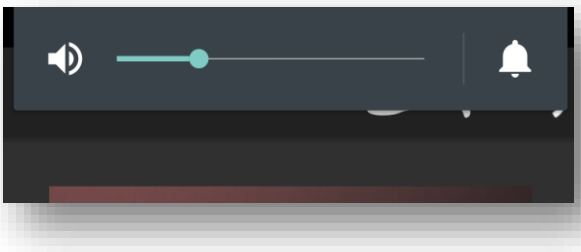

Figure 6: The Result of Pressing the 'Volume' Button on Android Version 5.1 (Lollipop)

3.1.3 Upload Client

The upload client, being the simplest of the three components has the simplest user interface. As it is a program that is designed to run on computer start up, be minimized to the system tray, and essentially be forgotten about by the user, its user interaction is kept to a minimum.

The user can select one or more folders on their computer which contain music for the application to monitor for music files. When one or more music file is added to this folder, such as when the user has purchased a digital copy of an album, it is uploaded to the web service for analysis and storage. It is envisioned that the user will only have to interact with this application once: the first time it is installed. The only buttons to be available to the user are those to add a folder, remove a folder and access the minimal settings where they can log in. Logging in is necessary before it is possible for uploading to occur.

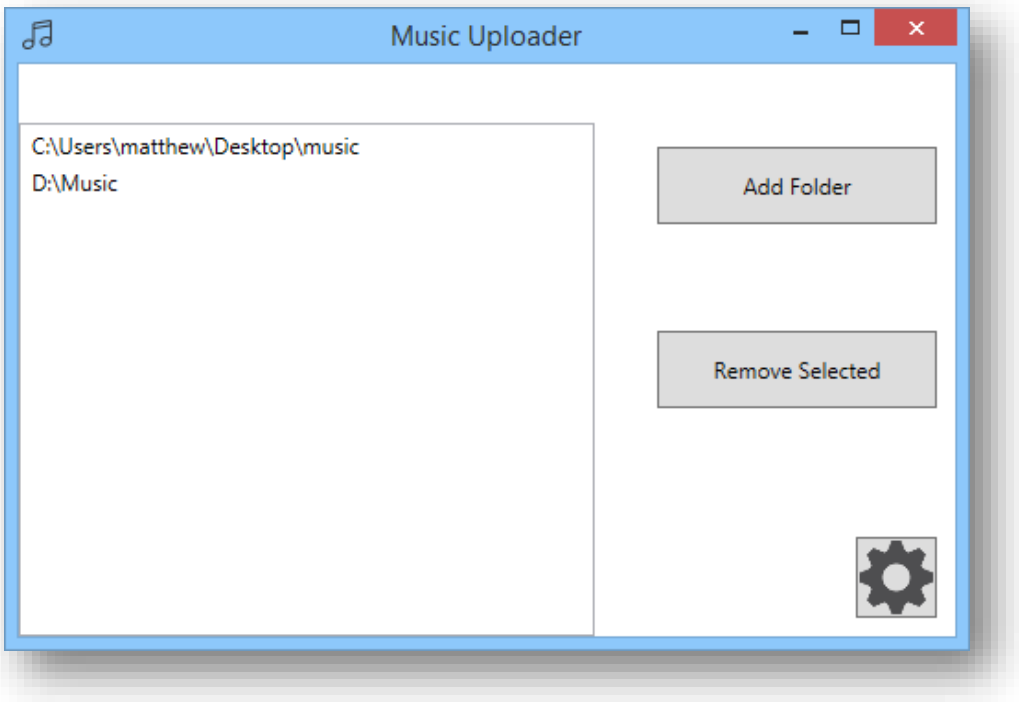

Figure 7: The Main Window of the Upload Client

It was necessary to require the user to log in to the service before the application can upload music files to their account on the service. This was facilitated by a simple settings pane accessed by clicking the settings 'cog' in the bottom right corner. This settings panel simply contains a username and password field, which, should the user already be logged in, are prepopulated with the current username and password, as well as a 'Save and Exit' and 'Cancel' button. When the save button is pressed, the application attempts to log in to the service. If

this was unsuccessful then the user is prompted to re-enter their credentials otherwise they are returned to the main folder selection screen. The cancel button returns the user to the main screen with no changes being made.

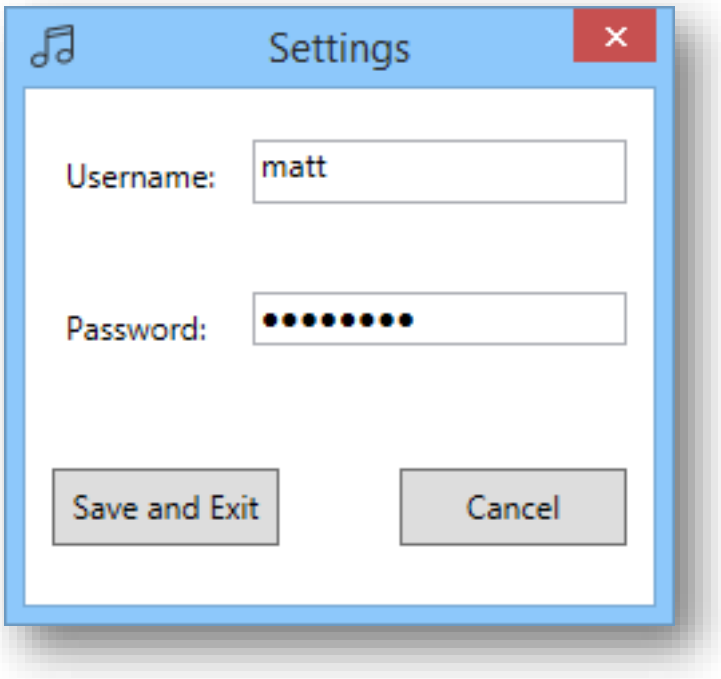

Figure 8: The Settings Pane of the Upload Application

As this application's goal is to silently watch a number of folders without requiring action from the user, the application minimizes not to the Windows Task Bar as most applications do, but rather the System Tray. Users can view the main window by double clicking the application's icon in the system tray, or by right clicking it and selecting an appropriate action from the menu.
After the first run of the application it launches to this minimized state to provide no interruption to the user.

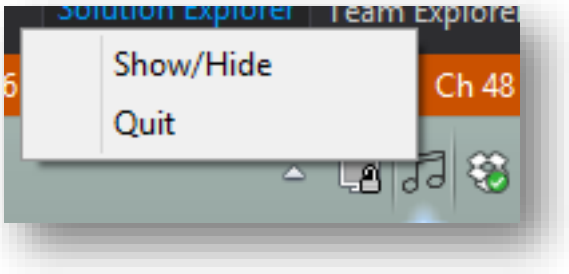

Figure 9: The Upload Application in its Default State, Minimized to the System Tray

3.1.4 Logo

Due to a lack of design skills, it was decided to outsource the design of a logo for the application to a third party. The website fiverr [20] was used to contract

a designer to design a logo which could be used on the login screen of the application, as well as the icon used on the application launcher to identify the application.
All other icons found in the application were either sourced from iconfinder.com [21] and are free for non-commercial use, which this project falls under or designed by hand using the GIMP image editor.

3.1.5 Representing Mood

It was necessary to investigate methods of conveying mood to the user and allowing them to input how they are feeling.

3.1.5.1 Colour

It was decided to represent the user's selection of their current mood by using colours to differentiate different feelings. Much research has been undertaken in the area of correlating human emotions to different colours. According to a study from Columbia University regarding our emotional response to colouring found in film [22], red is often associated with "…, Hatred, Life, Noble" feelings, while blue is often associated with "peace" and "tranquillity". Black, which was described as "indefinite", was selected for use in this project due to its achromaticity, to symbolise neutrality. This lets the user see they have not selected any mood yet. White could also have been used here, however it was decided that black matched the existing aesthetic of the application more appropriately.
Red and blue, therefore, were selected to represent aggressiveness and calmness respectively.

3.2 Technical Architecture

3.2.1 Previous Technical Design

3.2.1.1 Standalone Application

Initially it was decided to develop the project as a standalone mobile application whereby the files located on the device would be analysed, rankings stored in a databases on the mobile device and files played directly from local storage.
It was decided that this approach did not provide enough novelty; this approach has been taken before by many similar applications. Implementing the project in this way also means that only the music found on the (usually very small) local device storage may be used where as many people's music collection is often much larger. This option also reduces the possibility of

pooling suggested improvements to the track analysis from the user; a server would be required to coordinate the suggestions among all the devices.

3.2.1.2 Web Application Only

Another planned technical design that was considered was implementing the system as a web application; removing the native mobile application and replacing it with a web page the user visits when they want to listen to their music. This option has the benefit of being accessible on any internet browser-equipped device by utilising a modern, responsive design.
Mobile usage has now overtaken desktop computer usage [23], and the experience on mobile would suffer with this method; web pages to not get the same priority from a mobile operating system and tabs may be killed to free memory, interrupting music playback. Moreover, some mobile platform cease audio output from a browser tab when that tab loses focus.

3.2.2 Choice of Three-Tier Architecture

It was decided to break the system into three separate components which each have a separate task, these components were:

- Web Service which stores and analyses a user's music files
- Mobile Application which streams the music files from the web service after specifying the desired mood
- Upload Utility which watches a user's music collection on their computer and uploads all tracks to the web service

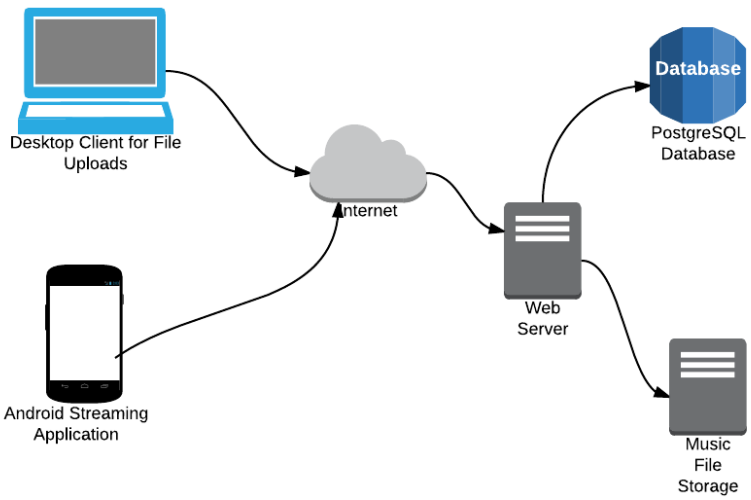

Figure 10: Technical Architecture Diagram

An influencing factor in the choice to separate the music analysis from the client was that by having the analysis algorithms and the resulting analysis data reside on the server as opposed to the client, the initial analysis of a track only had to be performed once; subsequent uploads of the same track could be discarded and the previous analysis and file could be used. This also means that the users can provide feedback on the analysis of a particular track which would improve the quality of track selection relative to mood for all users with that track in future.

3.2.3 Drawbacks

Using native applications for each platform increases the workload of developing the project. Any user of a platform that an application has not been developed for is unable to use the system.
Desktop only users are also left out here as they are unable to run mobile applications. A solution here may be to develop a Windows Universal application [24], this would enable users of both Windows Phone and Windows 8 and above to run the same application.
Adding a web frontend as another client could also improve this technical architecture as Desktop users from all Operating Systems would be able to use the system.

3.3 Database Design

While the database schemas for both the web service and the upload desktop utility were relatively simple, designing them in such a way as to minimize redundancy and improve query times was an important step in the design of this project.

3.3.1 Web Service Database Schema

The database for the web service needs to store the metadata for each track in the web service, this includes artist name, track name and duration – all information which the mobile client will need to know about the tracks it is receiving from the service. Other information stored about tracks here is the path to the file on disk. The database must also contain information about users of the system – their username, email address and a SHA256 hash of their password. Another piece of data about a user that is required to store is whether a reanalysis of a user's tracks is currently in progress. This is because such an analysis can take a substantial amount of time should that user have a large amount of tracks in the system, and some features may be unavailable until the reanalysis has been completed.
It is also necessary to be able to associate tracks with users, as duplicate tracks are not stored, only one copy of a particular track is saved despite how many

user have uploaded this tracks. Therefore it is important to know which users own a copy of the track.
As a result of these requirements of the database, three tables were created, they are:

- tracks
- users
- user_has_track

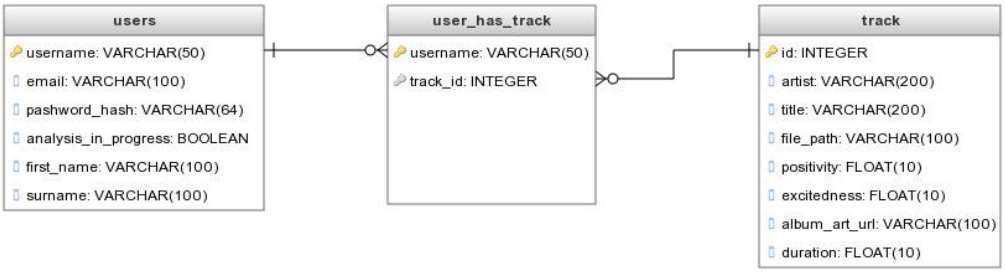

Figure 11: Web Service Database Entity Relationship Diagram

3.3.2 Upload Utility Database Schema

The database design required for the Upload Utility was very simple; the only information required to be stored is the hash of each file processed by it (uploaded to the service) and the list of folders that a user has selected to be watched. Therefore only two tables with no relationships were needed.

Figure 12: The Upload Utility Has Only Simple Storage Requirements, no Relationships

3.4 Designing a RESTful API

Designing an API was an important part of this project as there are three different software systems that must communicate with each other. It was decided to design this API in a RESTful manner.

REST stands for Representational State Transfer, and is a widely used alternative to SOAP APIs. IBM defines [25] a REST service as one which follows the following design principles:

1. Use HTTP Methods Explicitly
2. Be stateless
3. Expose directory structure-like URIs.
4. Transfer XML, JavaScript Object Notation (JSON), or both.

The API has a base end point of 'api/' and as such all routes following this prefix. Variable parts of the endpoint are show in [UPPERCASE] and described in the description. Should an endpoint which requires a user to be logged in be requested by a non-authenticated client, a 401 UNAUTHORIZED response is sent.

End Point	Description	Login Required	HTTP Method(s)
/upload	Handles files uploaded to the system by the upload utility. When files are received, they are organized and analysed by mood.	Yes	POST
/statistics	Returns a JSON response containing the total number of tracks the user has in the system as well as whether a reanalysis is taking place.	Yes	GET
/reanalyse	Start a reanalysis for the logged in user by starting the reanalayse.py script as a sub process	Yes	GET
/whoami	Returns the username of the user that is logged in	Yes	GET
/analysisInProgress	Returns a Boolean value indicating whether an analysis is in progress for the logged in user	Yes	GET
music/[ARTIST]/[ALBUM]/[FILEPATH]	Returns a raw data file containing the track specified	Yes	GET
/track/[EXCITEDNESS]/[POSITIVITY]	Returns a JSON track object containing track information and streaming URL for above method for a random track deemed to match the specified excitedness and positivity.	Yes	GET
/users/new	Takes as a body a JSON object containing a new user's username, password and email address and creates that user if one with the given username does not already exist.	No	POST
/disagree/[ARTIST]/[TRACK]/[THOUGHTS]	How the client indicates a disagreement by the user for the given track by the given artist. THOUGHTS can be one of 'mp', 'lp', 'me', 'le' indicating More Positive, Less Positive, More Exciting, and Less Exciting respectively.	Yes	GET
/	Base end point, redirects to this project's repository on GitHub [26].	No	GET
/login	Body contains JSON object with username and password to attempt to log in with	No	POST
/logout	Clear the current session	Yes	GET
/images/[PATH]	Returns an image found at the given path inside the main 'images' directory.	No	GET

Table 1: Description of the End Points of the REST API

3.5 Model View ViewModel (MVVM)

MVVM is a design pattern introduced [27] by Microsoft to be used in conjunction with their XAML-based user interface technologies such as WPF, Silverlight and, later, Windows 8/8.1 Store applications and Windows Phone applications.
The pattern recommends that the written code to create graphical user interfaces is split into three distinct parts: Views, ViewModels, and Models.
As the Upload Client is written using the WPF graphical toolkit, it was decided to use this pattern in the course of its development.

3.5.1 Views

Views in MVVM represent the visual elements. This differs from the 'code-behind' technique often employed in applications using the older Windows Forms graphical toolkit where business logic is mixed in with presentation logic, causing the resulting code to be difficult to maintain and understand. This often results in almost no C# code being written in a Window class, with all visual elements being defined in XAML. The only code written is that which registers the View's ViewModel:

```
DataContext = new MyViewModel();
```

3.5.2 ViewModels

User interface elements in the View are bound properties of the ViewModel, for example, a Textbox's Text is bound to a string property in the ViewModel. When the user enters text into this Textbox, the property in the ViewModel's set method is called. In order to programmatically set this property and have the changes reflected in the user interface, the ViewModel implements the INotifyPropertyChanged interface and calls an OnPropertyChanged method which conveys the changes back to the View.

3.5.3 Models

The model represents the business logic of the application. It may also represent a database access object which retrieves data from a database, as is the case in this application: information about the folders being watched as well as the hashes of uploaded files are stored in a SQLite database.

3.6 Methodologies

3.6.1 Research of Methodologies

When deciding on a methodology to use for this project, the fact that this project was to be undertaken by only one developer had to be taken into account.
Ultimately it was decided to not strictly follow any one methodology, but rather incorporate elements from many, as well as follow many best practices.
Several software development methodologies were researched before development of the system began, including Agile with scrum, and extreme programming. However, as mentioned, some of these were found unsuitable or irrelevant for a one person development team such as the one developing this system. Scrum, for example, encourages daily stand up meetings between members of the development team in which progress updates are given. These meetings are called stand ups as all team members stand up during the meeting, to, in the words of Martin Fowler [28], "keep the meeting short". It is thought that the discomfort of standing for an extended period of time encourages the developers to keep their progress reports short and to the point. Stand ups are not compatible with a single person development team, however weekly meetings with the project supervisor served a similar purpose; progress to date was discussed, as were tasks for the coming week. These meetings were, for the most part, conducted in a seated position, however.
Extreme programming provides a set of rules [29] to follow, however many of these are strongly focussed towards a team, rather than a single developer. Such unsuitable rules include 'Give the team an open work space' and 'all production code is Pair Programmed'. Pair programming here means two developers at one machine, both working on the same piece of code; this is obviously unsuited to this project. However some of the rules of extreme programming will be followed in the development of this system, such as: "All code must pass all unit tests before it can be released." and "when a bug is found, tests are created". Extreme programming is a set of guidelines which encourages a very iterative development process, and in that respect, it is quite similar to the 'custom' methodology being utilized for this project.

3.6.2 Use of Source Control

Source control played an integral part in the development of this project. Development on the system to date was performed using the git Source Control Management system (SCM). The repository [26] is located on GitHub, using a private repository, and commits on the local git repository are pushed to this remote repository. Code was committed early and often to ensure ease of rolling back to an earlier point without losing any other, potentially unrelated work. For example, before a feature was to be implemented, all code should have already been committed, then once that feature was implemented, the

code was committed again. A similar process was used for refinement and bug fixes: a bug in the http routing of the web service won't be addressed if there's a file from the mood analysis subcomponent open already; those changes will either be committed, if ready, or stashed for a later date. While multiple commits take place during the development day, code is typically only pushed once a day, as there are no other team members who rely on quickly accessing my changes, and vice-versa. Commit messages were typically kept relevant and helpful for quickly rolling back to a previous commit should it become necessary.

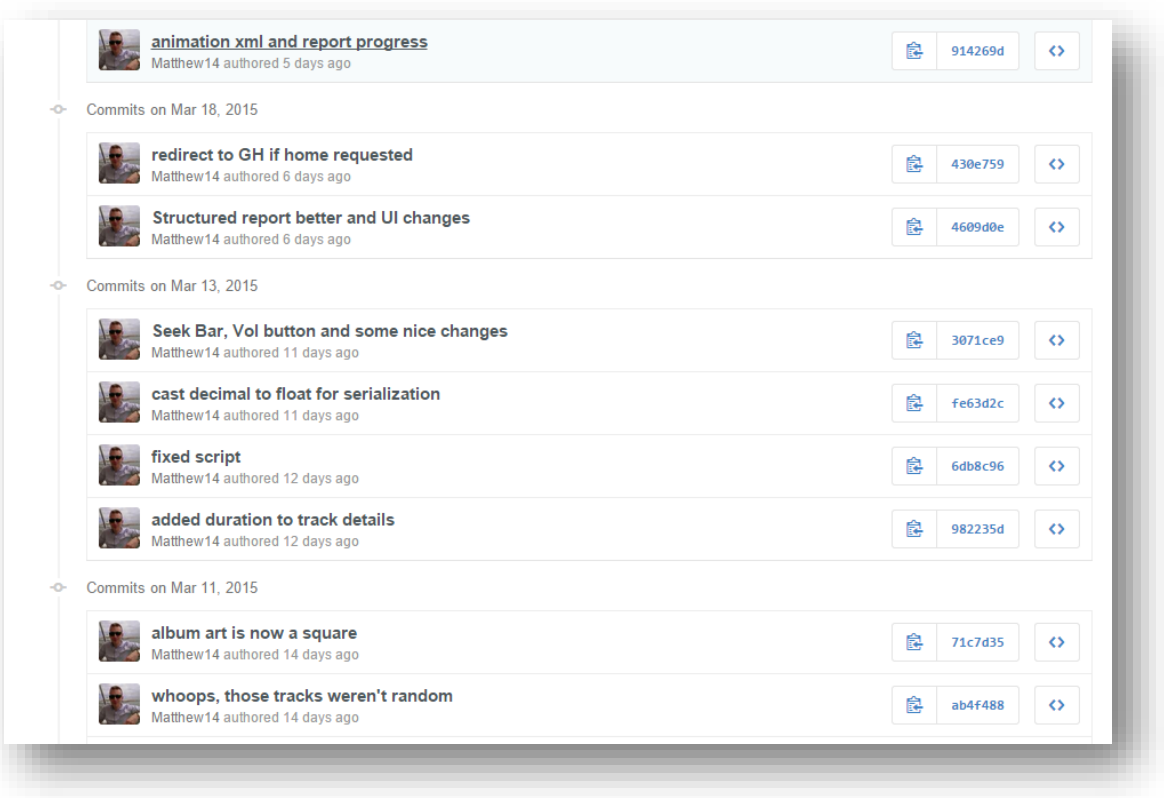

Figure 13: Selection of Recent Commits to the Git Repository Hosted on GitHub

Git's 'diff' tools were also regularly used as a quick reminder of what changes had taken place since the last commit, and served as a useful debugging tool, providing a quick overview of all changes in a file, to easily see which change might have introduced the unwanted behaviour.

Figure 14: A Git Diff showing additions and removals to the Login Method of this Android Activity

3.6.3 Adhering to Coding Guidelines/Standards

For readability and uniformity it was decided to follow the respective guidelines and standards of the two development languages used, C# and Python.

3.6.3.1 PEP-8

Python Enhancement Proposal (PEP) No. 8 [30] is the de-facto python style guide. Written in part by Guido van Rossum, Python's creator, the guidelines outline recommends such practices as using spaces rather than tabs for indentation, limiting line length as well as naming styles. There is also a PEP8 software tool which takes as input python source code and identifies where the source does not conform to the standards. This tool was incorporated into the Sublime Text editor during development to ensure constant conformance.
Similarly, PEP 257, Docstring Conventions [31] outlines best practices for documenting

```python
def save_track_in_right_place(track, tmp_file_path):
    """Works out where a track should be saved based on artist/album info and
        saves it in that correct place

        Arguments:
        track -- the TrackDetails object to move
        tmp_file_path -- temp directory to write to (differs between prod/dev)
    """

    artist_dir = os.path.join(upload_directory, track.artist)
    album_dir = os.path.join(artist_dir, track.album)

    if not os.path.isdir(artist_dir):
        os.mkdir(os.path.join(artist_dir))

    if not os.path.isdir(album_dir):
        os.mkdir(album_dir)

    final_path = os.path.join(upload_directory, track.filepath)
    if not os.path.isfile(final_path):
        os.rename(tmp_file_path, final_path)
```

Figure 15: Part of a python function which adheres to pep8 and pep257

3.6.3.2 C# Coding Conventions

The C# specification does not specify a standard for C# code, however Microsoft provide a set of guidelines for developers to follow [32]. Such recommendations made here include use of implicitly typed variables (using the 'var' keyword) when the type of a variable is evident from the right hand side of the equals sign. Also outlined is when to use upper case or lower case characters in variable, class and method names.

In addition to what Microsoft outlines, the commonly used convention of prefacing instance variables with an underscore (_) was employed to quickly distinguish them from locally scoped variables.

4 Architecture and Development

The system is developed using a multi-tier architecture consisting of:

- Web Service
- Client Application
- Desktop Upload Utility

4.1 Programming Languages Employed

For both the mobile application and the Windows upload utility the C# programming language was used. These applications used the Xamarin Platform/Mono Framework and Microsoft's .NET framework respectively.
For the web service which performs the mood analysis, the python programming language was used.

4.2 Development Environment

4.2.1.1 IDE and Text Editors

Three editors were used in the development of this project: Visual Studio for the Windows upload client, Sublime Text for the Python web service code, and Xamarin Studio for the mobile application. The choice of Xamarin for the mobile application was purely out of necessity: the student edition of the Xamarin platform allows development to take place only in that company's IDE. Visual Studio would have been preferred for uniformity with the Upload Client, the rich feature set available not found in Xamarin Studio as well as an existing familiarity and comfort with Microsoft's IDE.

4.2.1.2 Version Control

Git was used for version control as discussed in the Methodology subsection of the previous section. GitHub was chosen for the remote hosting of the repository. A private repository was used during ongoing development but this will be made public after submission. The repository can be found at https://www.github.com/matthew14/Final-Year-Project

4.2.2 Language / Framework

It was decided to use the Python programming language to implement the web service along with Flask 'micro-framework' to handle common web application tasks such as handling http requests as well as forming and sending

http responses to these request. Additional features of Flask used in the web service included its session handling.

4.2.3 Database

PostgreSQL was the database used for the web service. A development database was configured on a headless Debian Virtual Machine located on the Windows development computer. A Production database was also set up on the production server. This database mirrored the development database: changes were first made to the development database, then, when they were ready to be deployed, they were made to the production database using the same SQL statements.

4.2.4 Development Virtual Machine

A virtual machine was created on Oracle's VirtualBox virtualisation software and was configured to mimic the production environment as well as possible. This included installing the apache webserver and PostgreSQL database engine as well as all other software found on the production server. This was done in order to minimize the changes that needed to be made in a production environment due to differences between the Windows development environment and the Debian GNU/Linux production environment.

This also allowed the development database to be installed, offering a separation between it and the production database. This allowed mistakes that were made during development to have no effect on the running production snapshot at that particular point in time.

Using a virtual machine also allowed transferring the development environment from one computer to another with ease. This proved extremely useful when a computer developed a fault and had to be replaced midway through this project.

4.2.5 Python Virtual Environments (virtualenv)

Virtual environments in python allow easy containment of a python program's dependencies and allow the environment to be easily rebuilt when the application is deployed on a different machine. Requirements of the program can be defined in a 'requirements.txt' file and installed using the pip package manager which comes bundled in a virtual environment along with a local python interpreter separate to the system one. Setting up a virtual environment and installing the necessary modules can be achieved using the following

commands in a terminal or command window from the same directory as the python source:

```
virtualenv env
source env/bin/activate
pip install -r requirements.txt
```

Using virtual environments in this way rather than using system-wide modules avoids problems which arise from two python programs which expect a different version of a package located on the system.

4.2.6 Operating System

Windows was the primary development environment for all three components primarily due to an existing familiarity and access to a Windows machine. Another reason for the use of Windows to develop on was the requirement of the Upload Client to run on Windows; using the platform it runs on simplifies development.

4.2.7 Deployment

4.2.7.1 Operating System / Machine

A Debian GNU/Linux environment for the deployment of the web service for its stability - Debian focusses on this rather than bleeding-edge features- as well as an existing familiarity with this distribution over those running other package managers such as Fedora, CentOS or Arch Linux.
Version 7 of Debian, Wheezy, was chosen for its stability of packages rather than the newer 'testing' version, Jessie.

4.2.7.2 Setting Up a Virtual Private Server with DigitalOcean

The web service is deployed on a Virtual Private Server (VPS) hosted by DigitalOcean. The VPS runs version 7 (aka Wheezy) of the Debian distribution of GNU/Linux. At present this VPS has access to 512mb of ram and 20 gigabytes of solid state storage, but may be expanded as more users of the system begin to tax these resources.

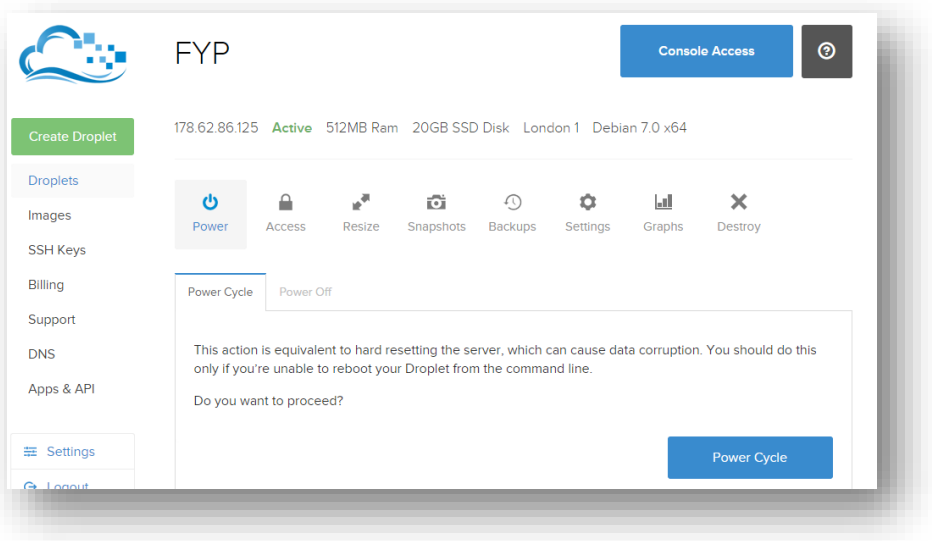

Figure 16: Overview of the DigitalOcean VPS

4.2.7.3 Domain Name

A subdomain of an existing personal domain, matthewoneill.com, was created for the project: fyp.matthewoneill.com. The DNS A record for this subdomain was set to map to the IP address of the DigitalOcean VPS mentioned above. While easier to remember than the IP address of the server, this has another advantage: should the machine that the web service is deployed to need to be changed, only the DNS record requires changing, not the service address that each installation of the client application is configured to connect to.

4.2.7.4 Web Server

Apache was the web server of choice for the project. This choice was made based on the existence of quality documentation regarding using the web server with flask, as well as an existing familiarity with Apache.
In order to make the apache web server work with Flask the mod_wsgi extension to apache had to be installed WSGI, or the Web Server Gateway Interface is an interface Python uses to enable web servers and python web enabled applications to communicate. While it is somewhat similar in nature to the Common Gateway Interface (CGI), it does not come as standard with the apache web server.

4.2.7.5 Using git to Deploy

As mentioned, git was used for version control, it did however have another use in this project: deployment. The steps to deploy the application from the development machine to the production machine were as follows:

1. Commit changes to local repository on the development machine
2. Push the latest commit to the remote repository on GitHub
3. SSH into the production VPS
4. Pull the latest changes from the remote repository
5. Restart the apache webserver to ensure it is serving the latest version of the python

A simple bash script was written to simplify the deployment on the server:

```
cd /home/fypuser/Final-Year-Project/ServerCode
git pull
sudo service apache2 restart
cd ~
```

4.3 Web Service

4.3.1 Handling Uploads

Music files are uploaded to the web service over using HTTP POST, which is handled by the Flask framework. A file does not necessarily have to be uploaded by the designated file upload application, however, the user uploading the file does need to be logged in at the time of upload, and locations where a user can log in are limited.

4.3.2 Organizing Files

Once a file is uploaded to the service by a logged in user, the file's ID3 tags are read to discern the Artist, Track Name, Album name and Duration of the track. Once this information has been gathered, the system checks to see if a file matching this latest upload has previously been uploaded, and if not it is stored in a folder using the following naming convention:

```
Uploads/{ARTISTNAME}/{ALBUMNAME}/{TRACKNAME}
```

4.3.2.1 Data Deduplication

Data Deduplication is the process of "finding and removing duplication within data without compromising its fidelity or integrity" [33]. It was decided to employ this progress when organising tracks that are uploaded to the web service. Should a file be uploaded by a user which appears to be the same file as that which another user has previously uploaded, then it is not saved again, rather that file is associated to the user that has uploaded it. This is achieved by

inserting that user's username and the track's internal id in the service into the 'user_has_track' table in the database. Using this technique means that no user will ever be played a track that they do not own, but also that no track will ever be stored twice meaning that both copyright issues and unnecessary data redundancy will be avoided.

4.3.3 Developing an Emotion Model

When tracks have been received by the service and have been organised, the analysis begins on the newly uploaded track. The required features of the track are gathered by querying the APIs utilised as well as analysing the file itself using the Essentia library. There are failovers in place should the first choice fail. For example, if the method employed to detect the tempo of the piece of music should fail, another method is used to ensure the tempo for that piece is found.

Two numbers per track comprise the model used by this system. They are:

- positivity
- excitedness

Both numbers can be above or below zero and the combination of these numbers is how a track is judged to be of a certain mood.

4.3.4 Hashing Users' Passwords

Storing users' passwords in plain text is strongly discouraged in the security community. There are many reasons for this, one such reason is that should an attacker gain access to the database storing the passwords, they would be able to read the passwords. This is a concern as many users reuse their passwords across various different websites and services, so gaining access to the password and email address for one service could mean gaining access to logins for many different services.

Hashing a password is the process of putting the text of a password through a hashing algorithm such as SHA256, BCrypt or MD5, and storing the output of this algorithm in the database rather than the password itself. When a user logs in, the password they enter is hashed using the same process and the resulting hash is compared with the hash in the database to see if it is correct. Using this method means only the user ever knows the password the use for the system. MD5 however, is not recommended, as it is prone to collisions and was proven insecure in the year 1996 by cryptographer Hans Dobbertin [34].

The SHA256 algorithm was chosen for use in this project as it has yet to be proven insecure and is included as standard in python's hashlib built in module. Following is a sample of how the passwords are hashed on registration by the web service:

```
def hash_password(password):
    return hashlib.sha256(password).hexdigest()
#recieve password from POST request…
```

```
password_hash = hash_password(json['password'])
#insert into DB…
```

4.3.5 Selection of Tracks

When a request is received from a client for tracks with a certain positivity and 'excitedness', it is not simply sufficient to return tracks from the database which match these two numbers as it highly unlikely for any track to match the requested numbers exactly. Instead a threshold is used where any tracks with rankings plus or minus this threshold are included in the selection. This threshold is dependent on the number of tracks a user has in the system; a user with few tracks requires a larger threshold to ensure some tracks are selected as a result of the request.

4.3.6 Reanalysis

The mobile client can request a reanalysis of the logged in user's tracks. This was implemented using a separate python program and calling it as a sub-process of the web application process. This was done as such because the reanalysis can take multiple minutes, and performing this as the result of a request to the web service would mean that the request would not return a response until the reanalysis has completed and leave the waiting for a response. It is important that the client is able to continue using the service after the reanalysis has been requested. The sub-process will continue in the background until it is completed allowing the request to be completed before that.
The following code snippet shows the calling of the analysis script ('script') as a sub-process from within the web service request handling code.

```
with open(error_file, 'a') as f:
        p = subprocess.Popen([sys.executable, script, username], stdout=
subprocess.PIPE, stderr=f)
```

4.4 Mobile Application

4.4.1 Statistics and Reanalysis

By tapping the 'i' (information) icon in the action bar, a statistics dialog is shown to the user which informs them how many tracks they currently have in the system as well as how many of that number were successfully analysed by the system. Ideally this number should be the same. A button is also presented to the user to start a reanalysis of their files which when pressed sends an analysis request to the web service which is handled as discussed above.
When a reanalysis is in progress for the current user, the 'i' icon turns blue to indicate this. This is achieved by having a thread spawned by the Main Activity

poll the 'api/analysisInProgress' endpoint of the web service and setting the icon accordingly.

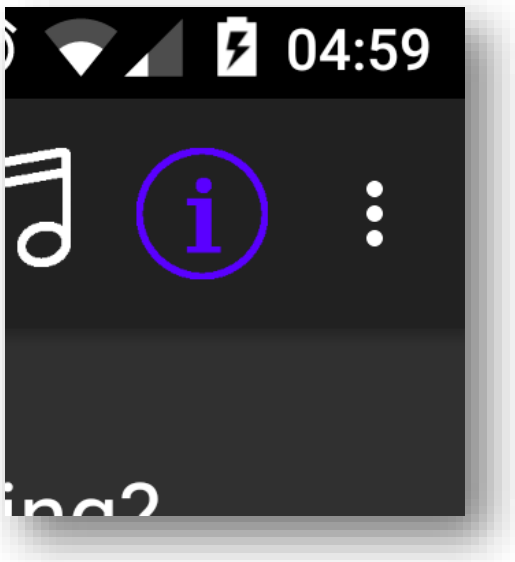

Figure 17: The 'info' Icon Turns Blue to Notify the User that Reanalysis is in Progress

4.4.2 Shake to Change Track

Mobile devices come equipped with a large selection of sensors these days. One such sensor is the accelerometer. An accelerometer, as the name implies, measures the acceleration of the device and can pinpoint its X, Y and Z coordinates in space relative to their previous position.
It was decided to use the device's accelerometer to detect when a user shakes the device while on the player screen and to skip to the next track when this happens. To detect a shake, every time the device moves, its position is stored in instance variables in the player activity, as is the time of the last movement. Should the position of the device change in a short enough period of time, it is deemed to be a shake and the PlayNext method of the Player activity is called as if the 'next' button had been pressed.

4.4.3 Disagreement

By pressing the 'thumbs down' icon while a music track is streaming from the web service, the user is presented with four options:

- More Positive
- Less Positive
- More Excited
- Less Excited

Choosing one of these options sends that feedback back to the service where the rankings are slightly altered
It is unlikely that one user's feedback will change whether it is categorised with the same characteristics as before the feedback, however if many people suggest the same feedback for the same tracks then it is likely that the analysis

of this track is not in line with how the majority of users feel it should be and is adjusted accordingly.

4.5 Upload Utility

The Upload Utility is a small Windows application written in C# .NET using the WPF graphical user interface toolkit. It starts minimized to the system try and runs a background thread watching all selected folders on the computer for new music which has been added to them. When a file is found in one of these folders which has not yet been uploaded to the web service it is uploaded and marked as such.

4.6 APIs Utilised

4.6.1 Echo Nest

The Echo Nest API [35] is a RESTful API which provides a large amount of information about music tracks. Such information provided includes:

- Tempo
- Modality
- Loudness
- Energy
- Key
- Liveness

It was decided to utilise this API over the preferred method of analysing the music files directly due to what is known as the 'double tempo' problem, spoken briefly in the research subsection 'meter' above. Essentially, the readings for tempo by analysing the files directly are often double or half what the actual tempo of the track. This erroneous reading would mean a drastic deviation from what an average listener would assess the mood of the piece to be and what the system assesses it to be. This proved the case in a large number of popular methods and software systems of analysing musical features from music files as noted in a study by Zapata and Gómez [36].

4.6.2 ChartLyrics

The ChartLyrics API [37] is a free for non-commercial use RESTful API which when queried with the title and artist of a particular song, attempts to return the lyrics for that song which are stored in its community-sourced lyric database.
Unfortunately, this API has an extremely low request limit, and requires a large amount of time to pass between requests. This proved problematic as it is

necessary to acquire the lyrics for many tracks at a time for a new user when they are uploading their entire music library.

4.6.3 Last.FM

Last.FM [38] is a service which enables users to 'scrobble' (send information about) music they are listening to from compatible music players to the service in order to view data about their listening habits and receive recommendations for new music. It is also a database of a very large collection of information about different music releases, with an accompanying API. This API was utilised to get album artwork images for tracks uploaded to the service. The API is queried for a link to the artwork image located on the last.fm servers and that URL is used in the mobile application to display the 'now playing' artwork. The artwork is not stored by any system comprising this project for copyright reasons. This usage of the API is permitted for non-commercial use.

4.7 Third Party Frameworks and Libraries Used

Like a majority of non-trivial software projects, it was decided to utilize some third party libraries and modules in this project to avoid 'reinventing the wheel'. This subsection outlines each such module and describes the role they play in the system.

4.7.1 Essentia

Essentia [39] is a C++ library containing a large collection of low-level audio analysis functions including Fast Fourier Transform, Windowing as well as beat detection. Due to the complexity of performing these low level analytical algorithms it was decided to utilize this library.
Essentia comes with a light python wrapper of the C++ functions which enabled the library to be used without any language interop.

4.7.2 RestSharp

RestSharp [40] is a rest client for .NET applications which offers classes and methods for interacting with HTTP RESTful services, such as the one implemented for this project. While similar functionality could be achieved by using the HTTPClient class provided with the .NET Framework, much more boilerplate code would have had to written to achieve the same result, and no benefit would have been gained by 'rolling' one's own Rest Client.
This library is used both in the Upload Client as well as the mobile application, enabled by the RestSharp.MonoDroid package.

4.7.3 Mutagen ID3

Mutagen is a python module which enables easy reading of metadata from differing music formats including MP3's ID3 metadata format. The library is given as input the path to a music file and returns an object containing relevant information about the track including artist name, track name, album name and track duration. Following is a sample of methods written in the web service code which make use of this library.

```
def get_album(filepath):
    track = ID3(filepath)
    return track['TALB'].text[0]
def get_artist_and_track_names(filepath):
    track = ID3(filepath)
    return track['TPE1'].text[0], track['TIT2'].text[0]
def get_duration(filepath):
    audio = MP3(filepath)
    return audio.info.length
```

4.7.4 pylast

Pylast is a python wrapper of the last.fm API mentioned above. Last.FM API data objects are wrapped in python classes and the majority of API methods provided are provided by way of a function. Use of this python module avoided 'reinventing the wheel' by calling the API using python's built in HTTP modules such as urllib2.
This module was used only to retrieve the album art for a given track in the web service's code. Usage of this module taken from the ServerCode/last_fm.py file:

```
import pylast
import config

api_key = config.lastfm_api_key
api_secret = config.lastfm_api_secret

network = pylast.LastFMNetwork(api_key = api_key, api_secret = api_secret)

def get_artwork(artist, album):
    try:
        image_path = pylast.Album(artist, album, network).get_cover_image()
    except pylast.WSError as e:
        image_path = None
    return image_path
```

4.7.5 psycopg2

Psycopg is "the most popular PostgreSQL adapter for the Python programming language" [41]. This module was used in the python code of the web service to interact with the PostgreSQL database. The module follows the Python database API specification [42] meaning the interaction with the database is similar to how it would be with a different database engine such as MySQL or Oracle. This is useful as swapping out the database engine used would require minimal code changes.

4.7.6 Beautiful Soup

BeautifulSoup is a python library designed to ease parsing html documents for screen scraping tasks including those with malformed tags. This module was used in the scraping of a music lyric website for retrieving the lyrics for tracks in the system.

4.7.7 Log4net

Log4net [43] is a project of the Apache Software Foundation to develop a logging framework for .NET applications which is similar in usage to log4j, long the de facto logging framework for Java applications.
The library provides logging methods that can be used within a .NET application. These method include:

- Debug()
- Info()
- Warn()
- Error()

And correspond to the logging level that the statement passed to them will get logged at. The logging level is adjusted in the application's App.config file. This means that debug statements can be logged during development, but turned off for a production build for security reasons and to reduce disk IO during running. The log file can then be analysed after a crash of the application in a production environment, should this occur.

```
<log4net>
  <appender name="Console" type="log4net.Appender.ConsoleAppender">
    <layout type="log4net.Layout.PatternLayout">
      <!-- Pattern to output the caller's file name and line number -->
      <conversionPattern value="%level %date T:%thread %logger:%line - %message%newline" />
    </layout>
  </appender>

  <appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
    <file value="${APPDATA}\UploadApp\logs\" />
    <datePattern value="dd.MM.yyyy.HH.mm.ss'.log'" />
    <staticLogFileName value="false" />
    <appendToFile value="true" />
    <param name="ImmediateFlush" value="true" />
    <maximumFileSize value="5MB" />
    <maxSizeRollBackups value="2" />

    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%level %date T:%thread %logger:%line - %message%newline" />
    </layout>
  </appender>

  <root>
    <level value="DEBUG" />
    <appender-ref ref="RollingFile" />
    <appender-ref ref="Console" />
  </root>
</log4net>
```

Figure 18: log4net Configuration Section Logging at DEBUG Level

The files log to the user's AppData folder with the date and time in the log file name. A sample log line, which shows the URL the application is using as well as the class and thread which performed the logging follows.

```
DEBUG 2014-12-06 18:26:36,899 T:9 UploadApp.Uploader.FolderWatcher:28 - Url:
http://127.0.0.1:5000
```

4.8 Other Third Party Sources

In addition to the lyric API mentioned above, some community editable song lyric sites were 'scraped' by the web service. The sites scraped are:

- azlyrics.com

A query is made using the URL http://search.azlyrics.com/search.php?q=artistname+trackname, and the first result of this is followed. On the resulting page if any, the lyrics for the track are 'scraped using python in conjunction with the BeautifulSoup module. A sample of the scraping can be found above.

4.9 Overview of Code Written

The following table is a non-exhaustive list of source code files written in the development of this project. As there are three separate components, there are a large number of files to outline (approx. 50). Files which have been auto-generated by the development environment are excluded, apart from those

which have seen a large amount of modification since generation (android XML layouts, for example, are initially auto-generated, but all xml was hand written, so as such are included here).

Filename	Description	No. Lines
ServerCode/	***Folder Containing all code for the web service***	
routes.py	This file contains all route mappings for the service's end-points. Requests are received here and dealt with accordingly.	235
app.py	The main entry point of the web service. This is the file the web server runs to start run the application.	14
config.py	In order to avoid "hard-coding" and values, all configurable elements are specified here, or loaded by this file from external flat text files.	61
pg_db.py	Provides a Database Access Object (DAO) which includes methods to store and retrieve track, user and other information from the server's PostgreSQL database.	179
analysis/mood_assesment.py	This file defines an EchoNestTrack class, which uses that API to retrieve information about itself. Methods in this class exist to build an "excitedness" and "positivity" score using this information.	127
analysis/lyric_analysis.py	This file Defines a LyricAnalyser class which attempts to retrieve lyrics of a track from the ChartLyrics API. As calls to this API often fail, it falls back to attempting to scrape the lyrics from a variety of lyric sites. Once lyrics have been obtained, an "excitedness" and "positivity" score is assigned to them.	
analysis/essential.py	Uses the Essentia library to load the track and extract relevant information	
user_details.py	Provides a class to encapsulate details about a user of this service.	7
lastfm.py	Provides functions for interaction with the last.fm API to retrieve album artwork.	15
reanalyse.py	A script designed to be run as a sub-process by the server in the background as execution can take a long time which reanalyses all tracks that a user has stored in the system.	
http_codes.py	File containing all HTTP error codes for use by the web service	61
ClientApp/	***Folder containing all code for the mobile application***	
MainActivity.cs	Code implementing the behaviour of the application's main screen.	175
Main.axml	XML layout of the main screen	24
PlayerActivity.cs	Code implementing the behaviour of the application's music playing screen	400
Player.axml	XML layout of the player screen	80
LoginActivity.cs	Code implementing the login screen of the application	132
Login.axml	XML layout of the login screen	27
RegisterActivity.cs	Code implementing the registration screen of the application	
Register.xaml	XML layout of the registration screen	
SettingsActivity.cs	Code implementing the settings page using android's shared preferences	
Prefs.xml	The layout of the Settings screen	
Menu.xml	Defines the menu items which are located in	
Track.cs	A class which encapsulates the track information which is received from the web service	24
TrackManager.cs	Class which exposes methods to the player to get the next track from the service	30
LoginManager.cs	Class which exposes user authentication and registration methods to the main application	50
MoodRestClient.cs	Internal class which wraps the web service api methods. This is used by the publically accessible Login and Track managers.	100
UploadApp/	***Folder containing all code for the windows WPF upload application***	
MainWindow.xaml.cs	The code for the main View of the upload application	
MainWindow.xaml	The layout xaml for the main view	
MainViewModel.cs	The ViewModel associated with the main view	69
SettingsWindow.xaml.cs	The code for the Settings View	
SettingsWindow.xaml	Layout XAML for the Settings View	
RelayCommand.cs	The relay command [44] which is bound to buttons in the Views	30
Database.cs	A DAO which provides methods to record which files have been uploaded by the system.	109
UploadClient.cs	This class wraps the REST API methods for uploading files and provides helper methods to do so	50
FolderWatcher.cs	The class which spawns a thread to continuously check all the files in the configured directories for new files which have not been uploaded to the web service.	136

Table 2: A Non-Exhaustive List of Source Files Developed for this Project

5 System Validation

Testing the quality of the mood analysis of the application comprised a large part of testing process for this project.

5.1 Comparison with Existing Systems

Many existing systems attempt to recommend similar music to a user based on music they listen to using the system. Such systems include Spotify and iTunes Genius. It was also decided to compare the analysis with existing music mood rating systems such as Mood Agent.

A starting point was chosen as a song that was subjectively sad to the ear of the tester as well as one which scored a low rating in the positivity and energy analysis by the system. This song was then played in iTunes and the 'start Genius' feature was used to obtain a list of tracks that iTunes considers similar to this starting track. These recommended tracks were then put through the system to be analysed and it was expected that the ratings for these tracks were to be similar. A subset of tracks analysed using this process is shown in the table. For this test the starting song was 'Lullaby' by Low. While this test is not comparing like for like - iTunes Genius is not a mood-only rating system – it is useful to determine whether tracks this system considers similar are considered similar to other, more established systems.

Artist	Track	Positivity Rating	Excitement Rating
Low	**Lullaby**	**-2.2**	**-4.11**
Slowdive	Sing	1.8	-3.65
Have a Nice Life	Bloodhail	-1.38	4.91
Swans	Song for a Warrior	2.2	0
Godspeed You! Black Emperor	Storm	1	-3.2
This Will Destroy You	Reprise	-1	4.1
Múm	A little bit, sometimes	2.45	1.05
Slowdive	Celia's Dream	1.72	1.1
Low	Cut	-2.62	-3.92
Caspian	Our Breath in Winter	1.27	2.58
Mogwai	Radar Maker	-0.81	2.17

Mono	Are You There?	2.31	-1.21
Slowdive	Dagger	1.23	3.21
Amiina	Sogg	2.412	2.33
Low	Sea	-2.34	-0.9
Mogwai	Katrien	1.75	1.15

Table 3: Results of Comparison to iTunes Genius

It was noted that these seemingly similar tracks provided varying results when passed through the mood analysis; this provided a basis on which to tweak the weightings assigned to the various parameters used in the formation of the rankings of tracks.

5.2 Automated Testing

Roy Osherove defines Unit Testing in his Book 'The Art of Unit Testing' as follows:

"A unit test is a piece of a code (usually a method) that invokes another piece of code and checks the correctness of some assumptions after-ward. If the assumptions turn out to be wrong, the unit test has failed. A unit is a method or function." [45]

Unit testing was carried out during development of the system using the NUnit framework for the upload application, NUnitLite, a lightweight testing framework for testing mobile applications developed using Xamarin for the mobile application, as well as pytest for the web service.

5.2.1.1 Subset of Unit Tests

Test	Expected Result	Pass/Fail
TestWithIncorrectCredentials	Does Not Proceed	Pass
TestWithCorrectCredentials	Proceeds	Pass
TestFileUpload	File stored in correct place	Pass

Table 4: Subset of Unit Tests

5.3 Manual Black Box Testing

In addition to automated testing, it was decided to apply some manual 'black box' testing to the project.
Nidhra and Dondeti described black box testing, also known as functional testing as testing where the code "is purely considered to be a 'big black box' to the tester who can't see inside the box." [46] The tests outlined in this subsection were undertaken with no regard to the inner workings of the

system, they merely perform an action on the system with the expectation of a result, and compare the actual result to the expected one.

Action	Expected Result	Pass/Fail
Launch application for the first time	Login screen displayed with, username, password textbox, login and register buttons	Pass
Press 'Login' button without filling in fields	Relevant error message shows, don't proceed to main application screen	Pass
Fill out login form with known incorrect credentials	Relevant error message shows, don't proceed to main application screen	Pass
Fill out login form with correct username and password	Proceed to main application screen	Pass
Press 'Register' button on login screen	Register screen appears with relevant information fields, register button and button to return to 'login' screen	Pass
Press 'Register' button on the register screen without filling out any fields	Error message shown informing user all fields need to be filled out	Pass
Fill some, but not all of the fields out	Error message shown informing user all fields need to be filled out	Pass
Fill all fields out including the username field with a username of an existing user	Error message shown informing user that the username is already taken	Pass
Fill all fields out including email field with an email that isn't correctly formed	Error message shown informing the user that a valid email address is required to register	Pass
Fill all fields out including different values in 'password' and 'repeat password' fields	Error message shown informing the user that passwords must match	Pass
Press back button on 'register' screen	Be returned to 'login' screen	Pass
Press register on 'register' screen with all fields correctly filled out	Confirmation of registration message shown, new user logged in and taken to main screen where instructions to	Pass

	the user for adding music are displayed	
Close application and reopen while logged in.	Be logged in again automatically using last credentials and skip 'login' screen	Pass
Press 'logout' on main screen action bar	Return to 'login' screen	Pass
Press 'i' icon on action bar	See summary of all current user's tracks from the web service	Pass
Press 'reanalyse' on summary popup	Receive visual confirmation that all current user's tracks are being reanalysed	Pass
Touch and drag coloured grid on main screen	Colour changes to indicate position on grid, blue dot follows touch	Pass
Release touch on grid when logged in as user with music added	Proceed to player activity, first track of mood playlist for selected mood playing	Pass
Release touch on grid when logged in as user with no music added	Display instructions on adding music again, do not proceed to player screen	Pass
Press 'play/pause' button on player when music playing	Music pauses	Pass
Press 'play/pause' button on player when music paused	Music resumes from same spot as paused on	Pass
Press back arrow button on player screen	Music continues, main screen presented	Pass
Press player back button	Last track played begins to play again	Pass
Press player forward button	Next track in mood playlist plays	Pass
Press semiquaver icon on main screen	Returns to player screen	Pass
Press 'settings' menu item in action bar on main screen	Settings page opens	Pass
Press 'thumbs down' icon on player screen	User is presented with four options: 'more positive', 'less positive', 'more excited', 'less excited'	Pass
Any of the four options above pressed	Visual confirmation that feedback has been received	Pass

Volume icon pressed on player screen	System media volume slider appears	Pass

Table 5: Manual Tests on the Android Application

Action	**Expected Result**	**Pass/Fail**
Utility is run for the first time	Main window shows, empty list of folders presented to the user	Pass
'Add folder' button pressed	Windows folder selection dialog appears, can select folder and it is added to the list in main window	Pass
'Remove folder' button pressed when no folders in the list	Nothing	Pass
'Remove folder' button pressed when folder in list selected	Selected folder removed from list	Pass
Settings 'cog' clicked	Setting dialog appears presenting username and password textboxes, save and cancel buttons	Pass
'Cancel' button pressed on settings menu	Settings dialog closes, returned to main window	Pass
'Save and Exit' button pressed with empty fields	Error message shown informing the user that all fields are required	Pass
'Save and Exit' button clicked with incorrect login details entered	Error message shown informing the user that their credentials are incorrect	Pass
'Save and Exit' button clicked with correct credentials entered in settings fields	Settings dialog closes, main window visible	Pass
Main window minimized	Window minimized, icon not visible in taskbar, icon visible in system tray	Pass
Icon in system tray right clicked	Options to show window or exit application presented to user	Pass
Icon in system tray double clicked	Main window un-minimized	Pass
Application run second and subsequent times	Application starts minimized to tray requiring no input from the user	Pass

Table 6: Manual Tests Performed on the Upload Utility

5.4 Performance Testing

As the service should be able to handle a large amount of users simultaneously streaming their music collection from the service, it was necessary to undertake performance testing on it to ensure it performed as expected.

5.4.1 Load Testing the Web Service

Apache, the web server that the web service is served by provides a benchmarking tool to test how a website performs under a heavy load, ab (Apache Benchmark). This tool was used on the web server to find out if it performs well enough with a large number of
The tool was run in the following way:

```
ab -k -c 200 -n 40000 http://fyp.matthewoneill.com/{end_point_to_test}
```

The flags specified to this tool are:

- 'k' – Send the KeepAlive header
- -c 200 – Simulate 200 concurrent requests at a time
- -n 40000 – Send a total of 40000 requests in this benchmark

5.5 Summary

In summary, the validation of this system helped to ensure that the whole system works well together. The comparison with existing systems demonstrated how well the overall analysis compares to other systems, and while the accuracy of music mood is still a very subjective topic, this process showed if tracks that this system considers similar to each other are considered as such by other, more established systems.
Unit tests were used where possible to reduce repetition with automation. When changes were made to the system, these unit test could be run to ensure the changes did not break existing functionality.
The manual black box testing performed gave an easy checklist to follow of expected behaviour from the client-facing parts of the system and ensured that everything is working as expected once this checklist had been exhausted. While the upload utility is a relatively small piece of the overall system, it still needed to be tested thoroughly as it is the only method a user has of adding their music collection to the system.

6 Future Work

6.1 Web Frontend

As web technologies have progressed, more software services are being made available as a web application rather than, or in addition to the native applications offered for platforms such as iOS, Windows Phone and Android. Popular music streaming services such as Spotify, Google Play Music and Rdio all offer a web application in addition to their native applications. This can prove useful to the user when they are away from their primary machines and wish to listen to music using the service.

6.2 Securing the Service

At the moment the web service providing the analysis and streaming of the user's music files is not encrypted using SSL or TLS standards. This could prove problematic as user credentials and copyrighted content associated with the user could easily be intercepted by an attacker.

6.3 Improving the Audio Analysis

The mood analysis performed on the tracks the user uploads to the service can be improved upon greatly with a combination of user feedback and investigation into other methods of extracting relevant features from music. Other musical features which are not currently being extracted may also prove useful in improving the system.

6.4 Python 3 Support

Python 3 was released in December 2008 and brought with it many improvements and new features, such as Unicode strings by default and a difference in division results. A result of this major overhaul of the language is the backwards incompatibility with Python 2 code that it was necessary to introduce.
This application was written using Python version 2 as library support at the time of development, primarily for the Essentia library used for music feature detection, was solely for this version.

While it is not presently possible to port the application to python 3 due to the outlined concerns, steps such as using python 2's future module to ease the transition may be employed in the near future.

6.5 Better Support for Classical Music

At present the system does not take into account the composer of the piece of music it is analysing, but instead uses the artist data. This could prove difficult as multiple recording artists have recorded pieces by popular classical composers, and the system would see these not as the same piece of music but instead two tracks by different artists.

6.6 Ethnic and Non-Western Music

At present, the system focuses on the music of western cultures. The reasoning behind this choice is outlined in Section 2, Research Undertaken, above. The system could, however, be improved to take music from other cultures and regions into account. A location detection system could be employed using the devices GPS sensor to detect whether the user is likely to be interested in a particular type of music. Moreover, the system could maintain a database mapping between musical tracks and the culture to which they belong; when a track is uploaded, this mapping could be consulted and the relevant analysis on the track be performed. For example, should a known Middle Eastern folk song be uploaded to the service, the features known to convey mood in this style of music could be searched for in the track.

6.7 Music with Lyrics not in the English Language

At present, the system only searches for English words in the lyrics of a track. Moreover, lyrics are only gathered from English language sources.
The system could be improved, therefore, by expanding to

6.8 Tablet and Landscape Layouts

At present, the Android application does not scale well to make use of the additional screen 'real estate' afforded to the developer by a tablet. Users of tablets expect tailor made layouts for tablet screens and a blown up version of the phone version of the application is often seen as unacceptable.
Moreover, at present the application does not adjust well to the client device being rotated into landscape mode; the mood selection grid in particular does not scale to the edges of the screen in this orientation. To provide a more polished user experience, the application at present will not adjust orientation

when the device is rotated. This, however, may frustrate the user who is not accustomed to this behaviour.

6.9 Expansion to Other Platforms

At present the application is only available on the Android platform. While Android leads market share of mobile operating systems at present, with 76.6% percent of smartphones running Google's platform [12], due to the extremely large number of smartphone users in the world, Apple iOS's 19.7% of users and Microsoft Windows Phone's 2.8% of users represent millions of users who are unable to access the application due to incompatibility with their device.
The use of Xamarin and the Mono framework for the development of the client application means that porting the application to the largest two other platforms will be relatively trivial; only user interface components need be redeveloped per platform, code to interact with the web service can largely be shared across platforms. The requirement of an Apple Mac to develop Xamarin for iOS applications is the only foreseeable obstacle.

7 Conclusion and Reflection

In the introduction to this thesis it was asserted that the goal of this project was to "develop a software system that will allow the user to select a mood or degrees of multiple moods that they are currently experiencing by inputting this choice in a mobile application. The user will then be played music that the system has deemed to be matching this mood." by this definition it could be argued that this project be considered a successful one: the final product is a software system which analyses mood from music and allows the user to play back music with regards to this analysis.

However, should the opportunity to undertake such a project again arise, some approaches would be changed. For example, more research can be undertaken into more features which influence a piece of music's emotional response in the listener. As some problems arose with the detection of certain musical features, more time should have been devoted to resolving these issues or investigating alternatives with the hope of improving the subjective quality of the mood rankings.

Time could have been managed better throughout this project, with the amount of work that the project took being greater than that which was initially estimated. This lead to a disproportionate amount of work being undertaken towards the latter half of the project time frame. Better time management would have enabled a better distribution of the workload.

Finally, the project could be deemed a success in that brought with it lessons in project management, time keeping, as well as facilitated the learning of new skills such as mobile application development.

8 Bibliography

[1] T. Christensen, The Cambridge History of Western Music Theory, T. Christensen, Ed., Cambridge University Press, 2002.

[2] W. M. University, "The Elements of Music," Western Michigan University, Michigan.

[3] J. Paparone, "How to express mood," 2006. [Online]. Available: http://www.musicarrangers.com/star-theory/t16.htm. [Accessed 2014].

[4] W. Apel, The Harvard Dictionary of Music, Harvard University Press, 1950.

[5] R. PARNCUTT, "Major-Minor Tonality, Schenkerian Prolongation, and Emotion: A commentary on Huron and Davis," *Empirical Musicology Review,* vol. 7, no. 3-4, 12 2012.

[6] B. H. Repp, "Perception and Production of Staccato Articulation on the Piano," New Haven, 1998.

[7] B. Pang and L. Lee, "A Sentimental Education: Sentiment Analysis Using Subjectivity," *CoRR,* vol. cs.CL/0409058, 2004.

[8] D. L. Bowling, J. Sundararajan, S. Han and D. Purves, "Expression of Emotion in Eastern and Western Music Mirrors Vocalisation," *PLoS ONE,* vol. 7, 03 2012.

[9] ISMIR, "The international society for music information retrieval," 2014. [Online]. Available: http://www.ismir.net/. [Accessed 2014].

[10] Stereomood srl., "About - StereoMood," [Online]. Available: http://www.stereomood.com/company/about.

[11] Xamarin inc., [Online]. Available: http://xamarin.com/platform.

[12] IDC Corporate USA, "IDC: Smartphone OS Market Share 2014, 2013, 2012, and 2011," IDC, 2015. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Accessed 2015].

[13] Genymotion, "Genymotion," 2014. [Online]. Available: https://www.genymotion.com/. [Accessed 2014].

[14] MongoDB inc., "Introduction to MongoDB," 2013. [Online]. Available: http://www.mongodb.org/about/introduction/. [Accessed 2014].

[15] T. P. G. D. Group, "Chapter 39. Procedural Languages," [Online]. Available: http://www.postgresql.org/docs/9.3/static/xplang.html.

[16] D. Inc., "SSD Cloud Server, VPS Server, Simple Cloud Hosting | DigitalOcean," [Online]. Available: https://www.digitalocean.com/. [Accessed 2014].

[17] Amazon, "AWS | Amazon Simple Storage Service (S3) - Online Cloud Storage for Data & Files," [Online]. Available: http://aws.amazon.com/s3/. [Accessed 2014].

[18] Microsoft, "Azure Storage - Secure cloud storage | Microsoft Azure," 2015. [Online]. Available: http://azure.microsoft.com/en-us/services/storage/. [Accessed 2015].

[19] G. Inc., "User Interface | Android Developers," Google, 2014. [Online]. Available: https://developer.android.com/guide/topics/ui/index.html. [Accessed 2014].

[20] Fiverr International Ltd., "Fiverr," 2015. [Online]. Available: https://www.fiverr.com/. [Accessed 2015].

[21] Iconfinder, "Iconfinder - 450,000+ free and premium icons," 2015. [Online]. Available: https://www.iconfinder.com/. [Accessed 2015].

[22] N. D. S.-F. C. Cheng-Yu Wei, "Color-Mood Analysis of Films Based on Syntactic and Psychological Model," 2004.

[23] R. Murtagh, "Mobile Now Exceeds PC: The Biggest Shift Since the Internet Began," [Online]. Available: http://searchenginewatch.com/sew/opinion/2353616/mobile-now-exceeds-pc-the-biggest-shift-since-the-internet-began.

[24] Microsoft, "Building Universal Windows Apps," 2015. [Online]. Available: https://dev.windows.com/en-us/develop/building-universal-windows-apps. [Accessed 2015].

[25] IBM, "RESTful Web services: The basics," [Online]. Available: http://www.ibm.com/developerworks/library/ws-restful/. [Accessed 23 03 2015].

[26] "This Project on GitHub," [Online]. Available: https://github.com/Matthew14/final-year-project.

[27] Microsoft, "THE MODEL-VIEW-VIEWMODEL (MVVM) DESIGN PATTERN FOR WPF," 2005. [Online]. Available: https://msdn.microsoft.com/en-us/magazine/dd419663.aspx. [Accessed 2014].

[28] M. Fowler, "It's Not Just Standing Up," [Online]. Available: http://www.martinfowler.com/articles/itsNotJustStandingUp.html.

[29] D. Wells, "Extreme Programming Rules," [Online]. Available: http://www.extremeprogramming.org/rules.html.

[30] G. v. Rossum, B. Warsaw and N. Coghlan, "PEP 8 - Style Guide for Python Code | Python.org," 05 July 2001. [Online]. Available: https://www.python.org/dev/peps/pep-0008/. [Accessed 2014].

[31] G. v. R. David Goodger, "PEP 257 - Docstring Conventions | Python.org," 2001. [Online]. Available: https://www.python.org/dev/peps/pep-0257/. [Accessed 2014].

[32] Microsoft, "C# Coding Conventions (C# Programming Guide)," [Online]. Available: https://msdn.microsoft.com/en-us/library/ff926074.aspx. [Accessed 2014].

[33] Microsoft, "Data Deduplication Overview," 2012. [Online]. Available: https://technet.microsoft.com/en-ie/library/hh831602.aspx. [Accessed 2015].

[34] H. Dobbertin, "The Status of MD5 After a Recent Attack," 1996.

[35] Echonest, "Echo Nest API Overview," [Online]. Available: http://developer.echonest.com/docs/v4. [Accessed 2014].

[36] J. R. ZAPATA and E. GÓMEZ, "COMPARATIVE EVALUATION AND COMBINATION OF AUDIO TEMPO ESTIMATION APPROACHES".

[37] ChartLyrics, "ChartLyrics Lyric API," [Online]. Available: http://www.chartlyrics.com/api.aspx.

[38] Last.fm Ltd., "Home - Last.fm," [Online]. Available: http://www.last.fm/home. [Accessed 2015].

[39] B. The Music Technology Group (MTG) of the Universitat Pompeu Fabra, "ESSENTIA | Open-source C++ library for audio analysis and audio-based music information retrieval," [Online]. Available: http://essentia.upf.edu/. [Accessed 2015].

[40] Restsharp, "RestSharp - Simple REST and HTTP Client for .NET," [Online]. Available: http://restsharp.org/.

[41] D. Varrazzo, "PostgreSQL + Python | Psycopg," [Online]. Available: http://initd.org/psycopg/. [Accessed 2015].

[42] M.-A. Lemburg, "PEP 249 - Python Database API Specification v2.0," [Online]. Available: https://www.python.org/dev/peps/pep-0249/. [Accessed 2015].

[43] A. S. Foundation, "Apache log4net: Home," [Online]. Available: http://logging.apache.org/log4net/. [Accessed 2015].

[44] Microsoft, "Commands, RelayCommands and EventToCommand," 2013. [Online]. Available: https://msdn.microsoft.com/en-us/magazine/dn237302.aspx. [Accessed 2014].

[45] R. Osherove, The Art of Unit Testing, 2nd Edition, New York: Manning Publications Co..

[46] S. Nidhra and J. Dondeti, "Black boxx and White Box Testing - A Literature Review," *International Journal of Embedded Systems and Applications (IJESA)* , vol. 2, no. 2, 2012.