Faculty of Science and Technology

BSc (Hons) Music & Audio Technology

May 2018

Design & Implementation Of A Music Streaming Mobile Application And Web Application

By Jake Gregory

DISSERTATION DECLARATION

This Dissertation/Project Report is submitted in partial fulfillment of the requirements for an honours degree at Bournemouth University. I declare that this Dissertation/ Project Report is my own work and that it does not contravene any academic offence as specified in the University's regulations.

Retention
I agree that, should the University wish to retain it for reference purposes, a copy of my Dissertation/Project Report may be held by Bournemouth University normally for a period of 3 academic years. I understand that my Dissertation/Project Report may be destroyed once the retention period has expired. I am also aware that the University does not guarantee to retain this Dissertation/Project Report for any length of time (if at all) and that I have been advised to retain a copy for my future reference.

Confidentiality
I confirm that this Dissertation/Project Report does not contain information of a commercial or confidential nature or include personal information other than that which would normally be in the public domain unless the relevant permissions have been obtained. In particular, any information which identifies a particular individual's religious or political beliefs, information relating to their health, ethnicity, criminal history or personal life has been anonymised unless permission for its publication has been granted from the person to whom it relates.

Copyright
The copyright for this dissertation remains with me.

Requests for Information
I agree that this Dissertation/Project Report may be made available as the result of a request for information under the Freedom of Information Act.

Signed: J.Gregory

Name: Jake Gregory

Date: 20/4/2018

Programme: BSc MAT

ACKNOWLEDGEMENTS

I would like to thank my primary supervisor Andrew Watson for his guidance throughout the duration of this project, as well as those who provided support through websites such as Stackoverflow, YouTube and Angular Support websites. Additionally, I would like to thank my girlfriend and best friend Molly, for all of her support throughout the duration of this project.

I would also like to thank all the participants who took part in the testing of the mobile application.

ABSTRACT

Research into the currently available mobile audio streaming applications revealed that there are currently no platforms to effectively promote bootlegged music.

The purpose of this dissertation is therefore to develop a mobile application that will allow artists of bootlegged music, to promote and demonstrate their content effectively. The sections within this dissertation display the approach taken to develop a web application and mobile application capable of this.

Research into the currently available applications created a foundation for the basic requirement specification, as well as basic design. The MoSCoW approach was taken to identify the importance of each requirement being implemented.

After deliberating between platforms, research suggested the application should be created using web technology frameworks, which would allow for deployment on multiple operating systems, as well as ease of implementation. The application implements several web technologies to create an effective music service.

The project has developed a mobile application compatible with iOS and Android as well as a small backend application to upload content to a custom server. The application uses a third-party service, to effectively recommend similar material/ music from a custom server. This is achieved using specific meta data that is associated with each track and using comparison algorithms to find similar music. The application has also implemented a small music player in order to play/ preview the music, on both the custom server and third party service.

It must be noted that the creation of bootleg music is a Copyright offence. Therefore, all of the music used within this project is a duplication of original pieces of music, created by the author. In turn, this project has been created with the intent to be a proof of concept and therefore certain aspects are limited/ scaled down.

CONTENTS

1.0 INTRODUCTION

1.1 Defining The Problem

The music industry is constantly growing with around 3,000 new songs being added to music services like Spotify every week, *The Guardian, 2016.* With that said, it can easy for music to slip under the radar and never be found by users on services like these. The solution to this is to use algorithms and meta data to help users find music that they would never usually find, using features such as discover weekly on Spotify. This type of feature considers the types of music that users usually listen to, and finds similar music based on those tracks.

These types of features are great for music that is already on that service, but there are thousands of other pieces of music that are being uploaded to other music services every day, which will not be considered when using a feature like discover weekly on Spotify. From a business stand point, this makes complete sense as companies would not want to promote their competitors. However, music services like Spotify do not allow certain types of music on their service, whether that be for legal reasons or a business preference, which means a lot of music is left unheard.

Therefore, the author has designed a mobile application that works in a similar way to discovery features found on Spotify, but instead is able to work across platforms, allowing similar music to be recommended from a separate source. In the case of this project, the author has created a separate music service to show this idea as a proof of concept, as well as using the idea of promoting bootleg material as the foundation.

1.2 Dissertation Structure

The project will initially identify the way in which current music streaming applications operate. The requirement specification and prototype will then be designed in accordance with certain elements that are offered by currently available applications.

Currently available mobile application development technologies will then be assessed, and the most suitable option will be chosen to produce an application. The implementation section within this dissertation will explain the in-depth approach taken to creating the mobile application and web application.

1.3 Aims & Objectives

The aims of the project consist of:

- Redirecting users to similar music they wouldn't have necessarily found
- Allowing artists to promote their content effectively
- Creating a mobile application and web application to achieve the above aims

The objectives of the project consist of:

- Identifying current music service applications
- Evaluating current music service applications
- Creating requirement specifications
- Evaluating development technologies available and choosing accordingly
- Creating application prototypes
- Evaluating application prototypes

2.0 LITERATURE REVIEW OF BOOTLEG MATERIAL

2.1 Background

The Oxford Dictionary defines the word bootleg as something that is "made, distributed, or sold illegally." *Oxford Dictionaries, 2018.* In the context of audio, this would mean re-recording, editing, chopping or modifying the original piece of music in any way, without the permission of the original artist or record label.

As suggested by *Michaelangelo Matos, 2016,* bootlegs really began to emerge in 1969 with the release of home recorded and live tapes of Bob Dylan, which were sold in a plain white vinyl sleeve, without the permission of the artist.

In the 1990's, the term "chopped and screwed" was coined within the American Hip Hop scene after DJ Screw created his distinctive sound, by slowing down, chopping and piecing together a variety of tapes and vinyl, to create modified or new pieces of music, with and without the permission of the original artist, *Shawn Lindsey, 2012.* These were usually sold or traded between music fans on an underground basis and never usually commercially, due to illegality.

2.2 Legal Issues

In the modern-day context, the same illegal bootleg activity still occurs, if not even more due to the ease of accessibility of musical software and hardware, as well as being able to share content with ease with the help of the internet. There are many YouTube channels dedicated to bootleg music as well as a large Facebook community dedicated to sharing bootlegs with one other, *Will McCartney, 2017.*

The underlying issue is that the activity on these services is illegal under UK law, without the permission from the original owner, *Gov UK, 2018.* However, there are some tools in place to help. Some services offer users the option to upload content with a creative commons license. This is for the original artist to specify when uploading, which will mean that others are automatically given permission to tweak, remix etc. with permission from the creator, without having to specifically ask, *Creative Commons, 2018.*

Another way around the copyright law, is using the parody and pastiche legislation, *Gov UK, 2018.* This piece of legislation allows certain content to be remixed, edited or modified without infringing the original creators copyright. The only exception is that the edited content must have a distinctive difference to the original piece of work. This is usually used with videos or pictures, and can be very hard to use with bootlegs due to the nature of usually directly using the original work, with only slight modifications. This in turn can make it hard for bootleg artists to claim under the parody and pastiche legislation.

2.3 Evaluating Existing Products

Soundcloud and YouTube are two of the biggest content streaming companies around. They both encourage the use of the creative commons license and the parody and pastiche legislation. This means that content like bootleg material can sometimes make it on to their services, however if the original artist or record label is not happy with the content, they are able to ask these services to remove them. In turn, these two services are mostly where the bootleg music community thrives. Soundcloud and YouTube also use recommendations to help users find new content. The algorithms that are used are very effective and promote bootleg content as well as original content effectively. These services are also free for all users, and do not require a subscription in order to upload or view content. Users are free (within bounds) to upload whatever they like, from their browser or mobile device.

Spotify is another streaming service, but is primarily music based, with the largest market share. The main difference between Spotify and the previous services mentioned, is that Spotify is a paid subscription service, only allowing the best quality material onto their service. This means the process of getting content onto Spotify is much more rigorous and in turn, bootleg content is strictly forbidden (due to illegality). The submitted content must be original or the bootlegger must have permission from the original artist, *Amuse, 2018.* For an artist to get their content on Spotify, they must go through an agency, which will in turn release the content on Spotify, if the stated requirements are met.

As described by, *Alex Moazed, 2016,* Soundcloud and Spotify are radically different business models and cannot be directly compared to one another, as one is a free service and the other is a paid service. *Alex Moazed, 2016,* also suggests that Soundcloud is used by artists to promote themselves and create a following, and Spotify is used as a form of monetization.

As far as an application that merges these two types of platform and performs the tasks suggested by the author, none were able to be found. It is the authors idea to therefore create a platform that to some extent, merges these two radically different business models.

2.4 The Market

According to, *Paul Resnikoff, 2015,* YouTube accounts for around 40% of all music listening through streaming services, which makes them one of the largest music streaming companies within the free music streaming market.

According to, *Alex Moazed, 2016*, Soundcloud has around 175 million monthly listeners which also makes them one of the largest streaming services, within the free music streaming industry.

According to, *Mark Mulligan, 2017,* Spotify has around 136.3 million subscribers globally. In the paid music subscription market, this equates to around 40% of the market, which can be seen in *figure 1.*

The author imagines that these two very large markets contain very different content and for very different reasons too. The author is unable to find any research data correlating to whether users of free services, also use the paid services, however from a personal standpoint the author foresees this as a true possibility.

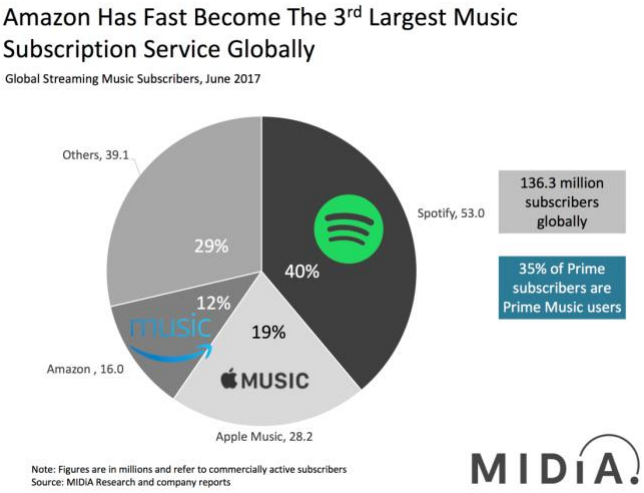

Figure 1: Global Music Streaming Subscribers 2017 (Mark Mulligan, 2017)

2.5 Evaluation Criteria

The author has mentioned a few services already and has explained a few of the limitations of the currently available software/ services. As the author proposes to create a similar service to these, it is suggested that a criterion should be created as a foundation.

All currently available applications offer similar features to each other and perform roughly the same tasks. Therefore, it is important to identify the limitation of each so a clear idea can be developed on where to expand. Below is a table with some criteria that can be applied to other products within the market.

The Criteria

- (1) Does the service offer music streaming?
- (2) Does the service offer recommendations?
- (3) Are the recommendations cross platform?
- (4) Can the user upload their own content?
- (5) Is it a paid service?
- (6) Does the service offer mobile application support?
- (7) Are bootlegs permitted?

Service Name					
Criteria	**Soundcloud**	**YouTube**	**Spotify**	**Apple Music**	**Shazam**
1	Yes	Yes	Yes	Yes	No
2	Yes	Yes	Yes	Yes	Yes
3	No	No	No	No	Yes
4	Yes	Yes	No	No	No
5	No	No	Yes	Yes	Yes
6	iOS + Android	iOS + Android	iOS + Android	iOS + Android	iOS + Android
7	Yes	Yes	No	No	No

Table 1: Other Services Evaluation

2.6 Criteria Conclusion

The above table has identified the limitations of some of the currently available services and has identified a gap, especially in the cross-platform recommendations criteria.

3.0 LITERATURE REVIEW OF MOBILE TECHNOLOGIES

3.1 Mobile Development Technologies

This section will explore the technologies available to allow developers to create applications for both iOS and Android.

3.1.1 Mobile Operating Systems

The mobile industry is dominated with two main competitors that dominate the smart phone market, Android and iOS. Both platforms allow users to connect to a variety of services using Apps, which can be downloaded from the respective stores on each platform. The mobile market is dominated by these two types of operating system, as can be seen in *figure 2.* However, development on each of these platforms varies considerably, which has an impact on the way applications are created.

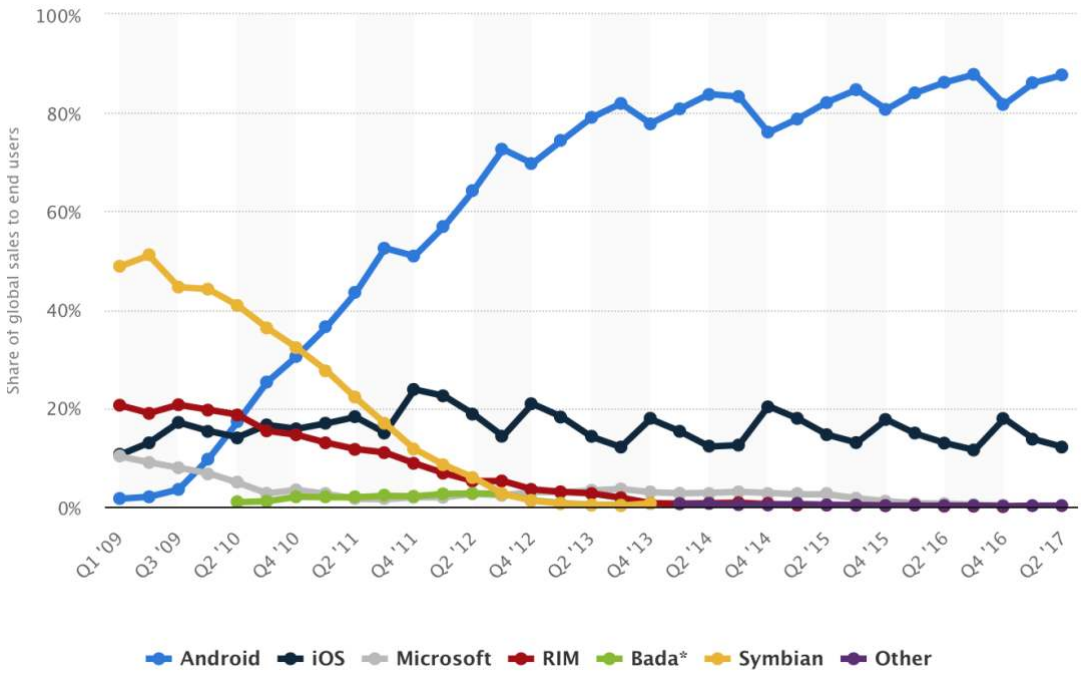

Figure 2: Mobile Market Share (Statista, 2017)

3.1.2 Developing For iOS

To develop applications for iOS, developers must use a piece of software called XCode, *Apple, 2018*. This piece of software is only available to Mac OS users, but does contain some very useful tools that aid in the development of applications, such as the virtual simulator and diagnostic command line interface. Applications made in XCode, have the option to be created using either Obj-c or Swift, which for some developers are not easy programming languages to use. This also means that developers must know how to use the specific

Integrated Development Environment (IDE) effectively. The benefit of using a platform specific IDE, is the ease of access to device hardware, making features like multi-touch gestures, GPS, camera etc. much easier to access. This also means that applications are more likely to run faster, as the IDE has been designed specifically for the platform.

3.1.3 Developing For Android

To develop applications for Android, developers required a piece of software called Android Studio, *Android Studio, 2018.* This is another platform specific IDE and is available to both Mac and Windows PC users. Android Studio allows developers to use the C and C++ programming languages as well as Java. It is a well-known fact that Java is a very popular programming language and therefore developers are more likely to use this, when creating Android apps. Again, this means that developers can gain access to device specific hardware, and applications run faster and smoother.

3.1.4 Hybrid Development

As one could expect, having to create two separate applications, on both market leading devices, with different programming languages, can be a lengthy and expensive process. However, there are several methods available to solve this.

Cordova, *Apache, 2018,* is a piece of software that allows developers to create applications for Android and iOS using Java. This allows developers to target two of the market leading platforms with only one code base. However, the drawback of using this technology is that device specific hardware can be hard to access and requires the use of 3rd party plugins. In some cases, the performance on the device is said to be reduced, due to the need for a translation layer, which transforms the Java into device readable code. In many cases, this is not a major issue unless the application is performing high intensity tasks.

The release of Cordova prompted software companies to start creating frameworks that allow developers to create native feeling applications for these devices. An example of this is the Ionic Framework, *Ionic, 2018.* Ionic uses a Java derivative called Angular, along with HTML and CSS to create and compile applications, for both iOS and Android, using Cordova integration.

Popular apps including Mc Donald's and Diesel, *Ionic, 2018,* have been created using the Ionic, suggesting the framework is effective.

3.1.5 Development Technology Conclusion

There are strengths and weaknesses associated with both native and hybrid application development. However, as the author does not intend to make use of any device specific features or perform any high intensity tasks, hybrid application development seems better

suited. The author also has previous knowledge and experience with HTML, CSS and Java, therefore making production slightly easier.

In terms of deployment, this would also allow the application to target both iOS and Android, without having to develop two separate applications. As those are the current market leaders, it would allow the author to target a wider audience. *Table 1* also suggests that all competitor products are compatible with both platforms.

The Ionic Framework can be used to give the application a native feel and target both platforms. The programming will be done using the Visual Studio Code editor, *Microsoft, 2018*, which is free from Microsoft and available for both Mac and PC. As Ionic is also a web-based technology, emulation on web browsers is also possible which can make testing easier.

3.2 Database Technologies

The application is required to fetch music from a database and display it from within the application. The application is also required to curate a playlist, using music from the database, suggesting an effective database querying method is required. Since Angular and HTML are being used, support within these is required.

3.2.1 Standard Database Technologies

For application content within an application to be dynamic, the application must make a connection to a remote server, where the data will be updated or changed. In turn the remote server is required to make use of a database.

In many cases, a relational database technology called MySQL*, MySQL, 2018*, is used to store data on a remote server. The data can then be accessed using a server-side scripting language called PHP, *PHP,2018.* PHP is embedded within HTML either on a website or application, and the server will send a HTML response containing data that is requested. As nothing is stored locally, the requested data should always be up to date, therefore allowing content to be dynamic.

These two technologies are used within many applications worldwide, as well as regular webpages that need to update content dynamically. The downside to using these technologies, is that they can be hard to set up and expensive. A good knowledge of how to use relational databases, as well as good knowledge of PHP is also required.

There are also other technologies available which offer similar features to the previously mentioned, such as MogoDB which uses the JSON format to store data. Requests can be sent to the server, and the server will respond with data from within the JSON file. Compared to the relational database structure, which can be very complex, JSON data structures are flat, meaning that they are much simpler to read and understand. The benefit of using JSON is that it can be sent easily to and from a server, as well as be used as a data

format within any programming language, *W3Schools, 2018.* However, JSON data can only contain text, which may be quite restrictive in some cases. This is not to say that data stored elsewhere cannot be linked to, within the JSON

3.2.2 Google Firebase

In April 2012, Google released a free database service that was designed to make accessing data and creating databases easy, called Firebase. This is achieved by having an online console that provides access to a database, storage and an easy to use authentication/ login system. This means there is no need to set up a server or install any technologies on a server. The developer is simply able to connect to the database, storage and authentication credentials, using a simple to use plugin, which are supplied by Firebase, *Google, 2018.*

The database uses JSON to store text data and allows users to link to the storage that is provided. Once connected to Firebase, developers can access everything on their Firebase account. The developer can then use this within an application to query the database. Data from within the storage can also be requested and displayed at the request of an application. Data is also able to be written and removed using the correct code.

As Google have supplied the plugin to connect to the database, and sometimes even the actual code to perform certain tasks, no real understanding of how the server works is needed. This makes this the perfect option for developers who do not have the time, finances or understanding of how to set up a server from scratch, therefore making Firebase a very viable option.

The main issue with Firebase, is that the data is stored with Google and therefore may cause some concern with developers in terms of data security, as the developers do not have physical access to the data. There may also be concerns regarding to what happens if Firebase itself is down, which in turn could mean that the developers applications would no longer works until the Firebase service is back up and running.

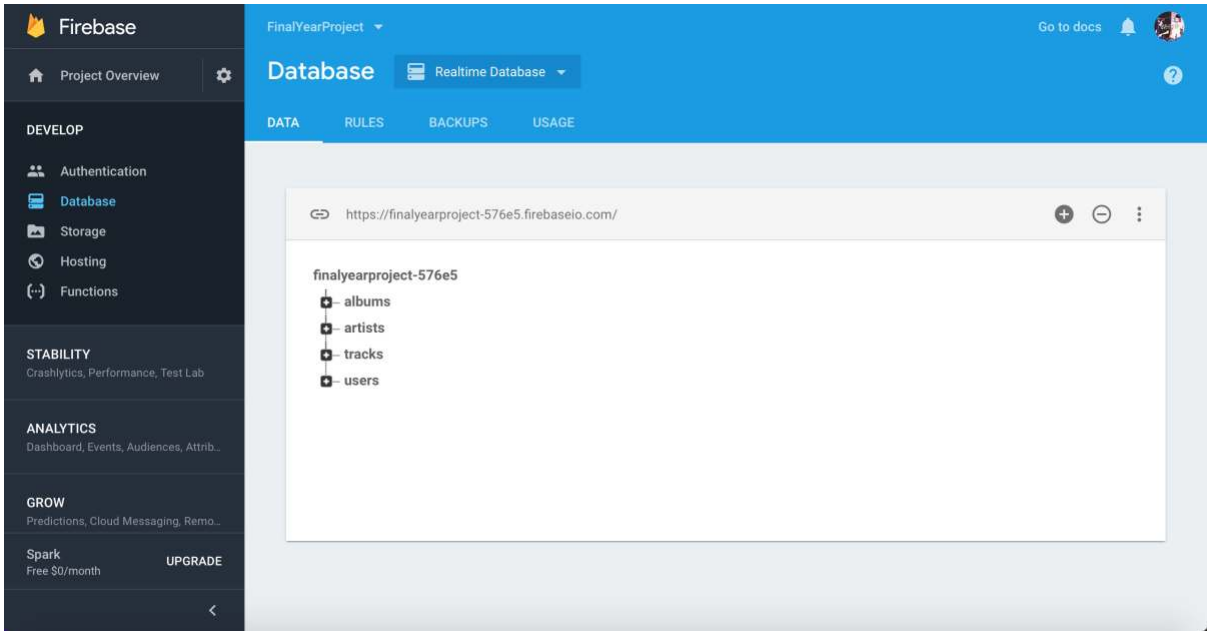

Figure 3: Firebase (Google, 2018)

3.2.3 Conclusion

The types of technologies explored have varying advantages and disadvantages for storing, accessing and writing data remotely. However, as the author has limited resources and understanding as to how traditional technologies can be implemented efficiently, it is suggested that the application created by the author, should make use of the Firebase service provided by Google.

The plugin provided by Google works well with web-based technologies such as Angular. Therefore, integration into a mobile application which uses web-based technologies, such as Angular, is simple and straightforward.

3.3 Web Technologies

Based on the model set by YouTube and Soundcloud, it is suggested that users must be able to upload their own content. Therefore, the application will be required to have a way of uploading music from a device, to the server. This can be achieved using a website.

3.3.1 Standard Web Technologies

The basics of every website is HTML, Java script and CSS. The HTML allows items to be placed around the page, as well as adding items such as buttons, radio controls, etc. The Java script is then used to add functionality to the buttons or items displayed within the HTML. Finally, the CSS is used to style the webpage. This can be changing colours, borders, placement, padding etc. When all of these are used together, a website is created. Some

websites look very basic and some look very professional, which is usually down to the styling used within the CSS. However, it can be very hard to create a professional looking website, without spending a lot of time using complex CSS code.

Fortunately, companies around the globe create templates for HTML objects, such as Bootstrap, *Bootstrap, 2018* and allows developers to use them free of charge. This is usually done by importing the CSS script and referring to it within the HTML class tag, allowing developers to create visually appealing websites easily.

3.3.2 Ng2 Admin

Ng2 Admin, *Akveo, 2018*, takes a similar approach to Bootstrap, whereby a template is provided to work with. The main difference is that Ng2 Admin is Angular based. The template provides a dashboard with complex graphs and charts, that can be used free of charge. The theming that is offered is professional and includes a login page as standard.

The layout within Ng2 Admin is simple but can be modified if needed. It is quite a heavy template and therefore can take a while to load in some circumstances. As most of the placement is predefined it leaves little room for creativity and can feel copied especially as other developers are free to use it.

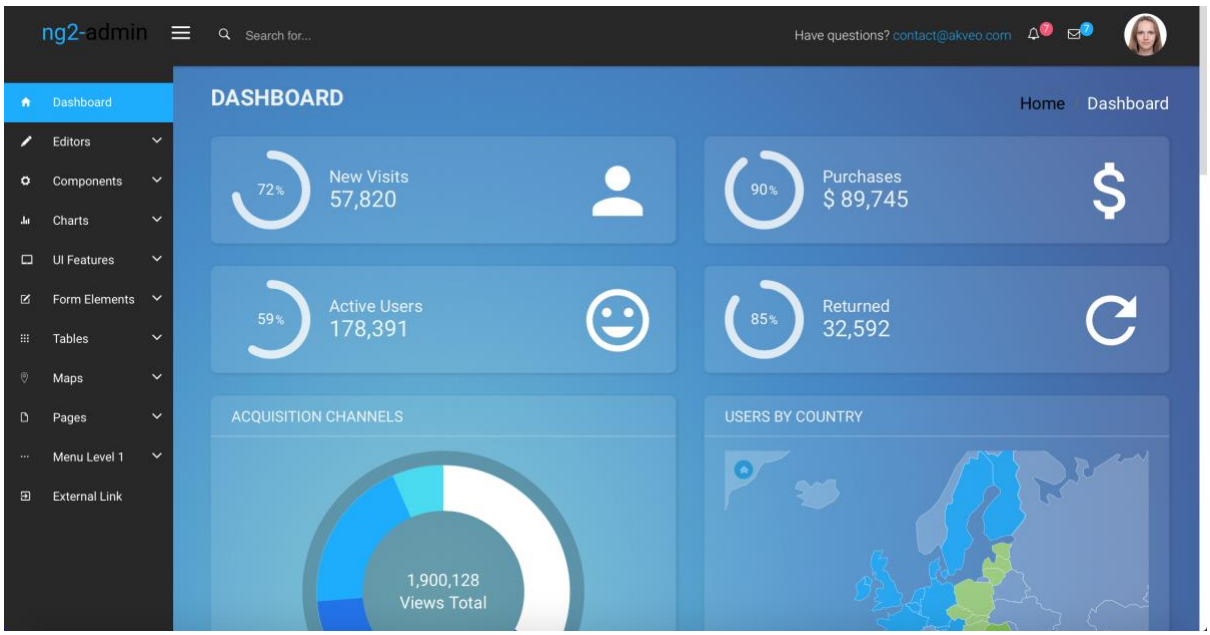

Figure 4: Ng2 Admin (Akveo, 2018)

3.3.3 Conclusion

The author believes that using Ng2 Admin will be a better suited approach to creating a website, that will allow users to upload content. As a login page is supplied as standard, it will allow quicker and easier set up in comparison to using the standard technologies. This is not to say that the template will not need to be modified to perform project specific tasks, but will allow for more time to be spent in the functionality area.

The author also believes that by using Ng2 Admin, only one code base needs to be used for the entire project, as the mobile application is likely to be Angular based too. This will mean that code can be shared between the mobile application and website, therefore saving time.

3.4 Ionic Audio & HTML 5

As the application will be required to play audio, an audio engine is required. There are many technologies available within Ionic and naturally implemented into the web technologies, which make audio playback and manipulation simple.

3.4.1 HTML 5 Audio

HTML 5 has built in audio capabilities as standard. This allows audio to be played natively without the need for a third-party plugin such as Flash Player, *W3 Schools, 2018.*

HTML 5 audio is sufficient enough for audio playback but does not offer any other capabilities in terms of processing audio, *W3 Schools, 2018.* There are very few drawbacks to HTML 5 Audio, as any browser that supports HTML 5, will be able to make use of HTML 5 Audio tags, meaning that it will work natively within many if not all browsers (mobile and desktop).

3.4.2 Web Audio

Web Audio is another audio technology that is built into HTML/ Java that allows for more complex operations of audio, including gain controls, panning controls and synthesizing basic wave shapes, *Mozilla, 2018.*

Web Audio is used in conjunction with HTML 5 and expands the capabilities of the existing technology. As Web Audio allows for more complex operations, performance can take a hit to a certain extent, in terms of latency. This could be an issue on less powerful devices such as smart phones.

3.4.3 Ionic Audio

As Ionic is web based, the same techniques that are used to play audio within HTML 5 can be used to play audio within applications. However, as ionic is a hybrid development framework, a third-party plugin is required to ensure that the correct audio engine is used, depending on the device the application is running on. For example, when testing in the browser, Web Audio Player will be used and when testing on the device another plugin is used, called Cordova Media. This will in turn route the audio to the required destination for that device, *Arielfaur, 2017.* This is necessary as hybrid development must cater for multiple platforms.

When Ionic Audio is installed to the application, a shared folder is created with provider files which contain all the necessary components to check what device the application is running on and use the correct engine. To make use of this, the associated provider file must be referenced within the application code, instead of referencing the standard code. In turn, audio will then be playable on each platform, whether that is browser, Android or iOS.

3.4.4 Conclusion

Below is a table showing the capabilities of each of the discussed technologies.

Feature	Web Audio	HTML 5 Audio	Ionic Audio
Audio Playback	Yes	Yes	Yes
Signal Processing	Yes	No	No

Table 2: Audio Technology Comparison

The author believes that a combination of these technologies will be utilised in the production of the mobile application. HTML 5 audio is less likely to be used, as only audio playback is necessary, and no signal processing is required. However, it is likely that the Ionic Audio plugin will utilise some features of Web Audio within its code.

3.5 Recommendation Algorithms

To find music users may like, services like Spotify use complex algorithms to help categorise music on their servers. To do this, data/ attributes are assigned to each track and it is suggested that songs of similar stature will have similar attributes. Therefore, songs can be recommended to users by comparing the songs attributes.

These algorithms are usually a close guarded secret, and in Spotify's case, they purchased an external company for several millions to calculate song attributes. However, delving into the Spotify API, there are several suggestions as to how they recommend songs to their users.

3.5.1 Spotify's Algorithm

Spotify allows developers to connect to an API, which allows them to make use of the content on the Spotify servers. This includes playing music, searching music and finding the attributes of songs on their servers. Whilst looking into the API and looking at the attributes associated with their tracks, three main track attributes were found. These were Dancability, Energy and Tempo. Users are not able to view this information, as it is something that is done behind the scenes to help categorise music.

It is not possible to know whether these attributes are what Spotify actually use to help recommend music to their users, but the author believes it definitely has an influence. The Spotify attribute calculations are very close guarded. Therefore, trying to calculate them will be near impossible without the algorithm.

3.5.2 Conclusion

As the author would like to implement a recommendation algorithm that works across platforms, it is suggested that something similar to that of Spotify should be used by having attributes such as Dancability, Energy and Tempo associated to tracks on the service that is developed.

As the calculations for Dancability and Energy could be complex/ difficult to get hold of, it is suggested that these will be randomly generated or simply typed in by users. However, in a future update, these could be automatically generated by the browser when a track is uploaded.

4.0 WEB APPLICATION REQUIREMENT SPECIFICATION

4.1 Background

As suggested by Ian *Sommerville and Pete Sawyer, 1997,* the requirement specification defines the functionality of the system and what it should do.

Braude, E. J. and Bernstein M. E., 2011, suggest that requirements of software can be split into sub categories called functional and non-functional requirements.

Functional requirements are described as details about what functions the application should do, and how it should perform in certain circumstances. This can be gained by looking into existing products, previous knowledge within the field and by conducting research within the market.

Non-functional requirements are described as requirements that do not directly relate to specific functions within the application. They are often a criterion that can be used to judge the operations of a system, rather than the specific behaviours.

These methods will be used in the evaluation section to ensure the application has met the criteria set in the specification.

4.2 Functional Requirements

The service the author is proposing to make, must allow users to upload their own content. For tracks to be associated with artists and artists associated with albums there will need to be a way of entering this when the user is uploading content. This will be explored further in the following sections.

4.3 MoSCoW Analysis

MOSCOW analysis is a prioritisation technique used in business analysis, software development and project management, to help identify the importance of tasks, *Agile Business, 2018.* The idea is to use the technique to identify the M: must have features of the application. The S: should have features. The C: could have features and the W: won't have features.

Below the author has put this into the context of the web application

Must Have Features:

- Login to the web application
- Create an artist
- Create an album
- Upload a track

Should Have Features:

- Ability to upload album image
- Ability to specify track dancability
- Ability to specify track energy
- Ability to specify track tempo

Could Have Features:

- Ability to edit the track the user has uploaded
- Ability to view the tracks associated with uploader

Won't Have Features:

- Ability to listen to uploaded tracks

4.4 Non-Functional Requirements

The non-functional requirements of the application are described as follows.

Stability: The application must be capable of running within the web browser with stability.

Availability: The application must be relatively small in terms of size (mb).

Portability: The application will be capable of running on all desktop web browsers.

Usability: The application must be easy to use and will resemble interfaces that users are familiar with.

5.0 MOBILE APPLICATION REQUIREMENT SPECIFICATION

This section will cover the requirements specification that will be used to evaluate the functionality of the mobile application

5.1 Functional Requirements

As explained earlier in section *2.3 Evaluating Existing Products,* there are some limitations to the existing products that are currently available in the market and they have been identified. Therefore, the limitations of these products will be used to help design the requirements specification.

5.2 MoSCoW Analysis

Below the author has used MoSCow analysis to identify the needed features of the mobile application.

Must Have Features:

- Play music on the application from the database
- Find bootlegs/ music of similar taste on the database
- Allow users to login

Should Have Features:

- Allow users to create accounts

Could Have Features:

- 3rd Party integration with Spotify or other service

Won't Have Features:

- Custom GUI design
- Monetisation

5.3 Non-Functional Requirements

The non-functional requirements of the application are described as follows.

Stability: The application must be capable of running on devices in a stable manor. This means the results should be predictable and crashing should not occur during the run time.

Availability: The application will be relatively light weight as most data will be stored on the remote server. This will ensure download times from app stores are minimal and the least amount of space is taken up on the users' device.

Portability: The application will be capable of running on iOS and Android.

Usability: The application will be easy to use and will resemble interfaces that users are familiar with. A total of 3 main views will be used and the application must be capable of exiting with the press of the home/ return button.

6.0 SOFTWARE DEVELOPMENT METHODOLOGIES

There are several software development methodologies that can be used to help with the development of software, which vary depending on the type of application being created and resources available.

6.1 Agile Software Development

Agile software development is used to minimise the risk within a project, by developing software in short time boxes called iterations, *IT Knowledge Portal, 2018.* The iterations usually last anywhere from one week to four weeks. Each iteration is like a mini project, which usually implements a new functionality. By using this method, small functions of the software can be created one at a time which will contribute to the end product. *Figure 5* shows this visually.

Each iteration includes planning, designing and testing ensuring that each function works as specified, before the next iteration begins. This allows development teams to work on separate functions, and bring them together to create the finished product.

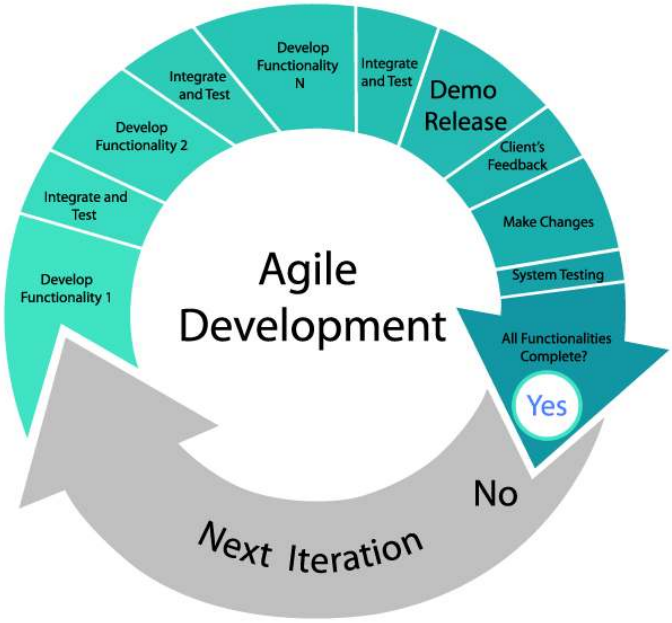

Figure 5: Agile Software Development (Jeff Watts-Roy, 2018)

6.2 Rapid Application Development

Rapid application development is a methodology that focuses less on planning and more on the implementation and creation of the software. By using this method, software can be created quickly, and risk is somewhat reduced. *Figure 6* shows this visually.

It is suggested that the risk is reduced, due to the methodology focusing on speed and user involvement. It is also suggested that the quality of the application is increased due to participants being involved in the evolution of the product, in turn creating an application better suited to the needs of the end user. As this methodology is designed to be fast, it usually means that projects are completed on time and to budget, *IT Knowledge Portal, 2018.*

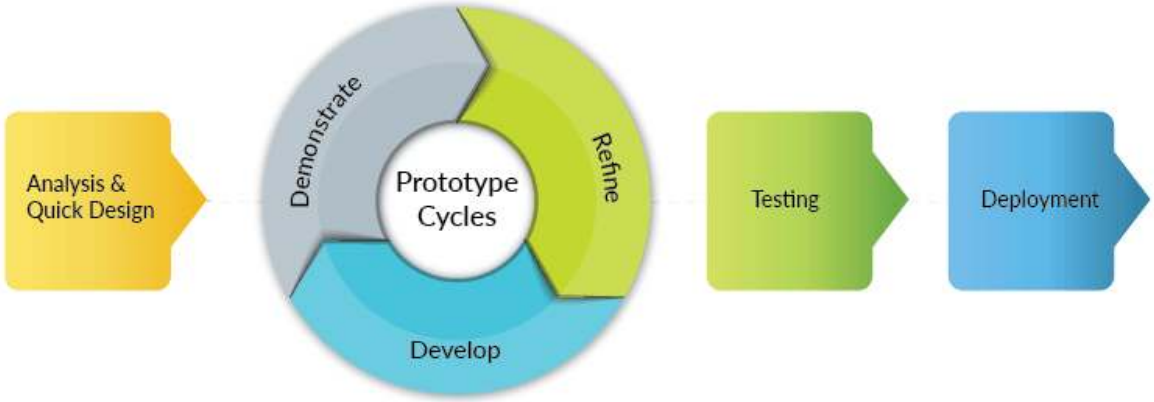

Figure 6: Rapid Application Development (Wave Maker, 2018)

6.3 Conclusion

The author believes that Agile Software Development will be better suited to the type of project this is, as small parts of the project can be made one at a time. This will allow each part of the project to be working correctly, before finally being assembled to create the end product.

The following shows the breakdown of iterations of the web application:

- Create database in Firebase
- Create login to Firebase through Ng2 Admin
- Create add artist to Firebase in Ng2 Admin
- Create add album to Firebase in Ng2 Admin
- Create add track to Firebase in Ng2 Admin

The following shows the breakdown of iterations for the mobile application:

- Create login to Firebase through mobile app
- Display track from Firebase in mobile app
- Play track from Firebase in mobile app
- Search Firebase for music from within the mobile app
- Load and display external JSON within the mobile app
- Play music from external JSON
- Utilise external selected JSON music attributes
- Curate playlist of Firebase music based on selected JSON music attributes

7.0 WEB APPLICATION DESIGN

7.1 Use Case Modelling

Use case modelling is a design approach that describes the user's interactions with the application. This method attempts to describe what the user expects, when interacting with the application. It can be a good method to help identify any additional user requirements.

7.1.1 Use Case: Logging In

This use case describes the process the user will go through to login to the web application.

Flow of events:

1. User navigates to the webpage in their browser
2. User is presented with a login screen
3. User decides to login
4. User is taken to home page

Handling Invalid Inputs:

- If the account does not match any records on the database, an alert is displayed with the corresponding warning

7.1.2 Use Case: Adding An Artist

This use case describes the process the user will go through to add an artist.

Flow of events:

1. User clicks add on the menu bar
2. User types in artist name into the corresponding input box
3. User clicks submit

Handling errors:

- If the artist name fails to submit, an alert warning will be presented to the user

7.1.3 Use Case: Adding An Album

This use case describes the process the user will go through to add an album.

Flow of events:

1. User clicks drop down menu that contains artist names
    a. If the artist name the user is looking for doesn't exist, the user must add the artist using the previous use case
    b. If the artist they are looking for is in the drop-down list, the artist can be selected

2. User types in the album name
3. User click the file selector button and adds the album photo
4. User clicks submit

Error handling:

- If there are any errors when uploading the album image, a corresponding error alert is displayed
- If there are any errors when submitting the album, a corresponding error alert is displayed
- If the image file is left empty, a warning will be presented to say the image cannot be empty
- If the album name is left empty, a warning will be presented to say the field cannot be left empty

7.1.4 Use Case: Adding A Track

This use case describes the process the user will go through to add a track.

Flow of events:

1. User selects artist name from drop-down menu
    a. If artist name doesn't exist, an artist must be added
    b. If artist name exists it can be selected
2. User selects album name from the drop-down menu that they want to add a track to
    a. If the album name doesn't exist, the album must be added
    b. If the album name exists, the album can be selected
3. User types in name of the track in the input box
4. User types in the dancability value of the track between 0 and 1
5. User types in the energy value of the track between 0 and 1
6. User types in the tempo value of the track
7. User clicks the file selector button and adds selected track
8. User clicks submit

Error handling:

- If any of the fields are left blank, an alert will be presented to the user to explain they cannot be left empty
- If the user enters a number not within the range specified (0 – 1), an error message will be presented asking them to enter a valid value
- If the submission fails to upload, a warning alert will be presented to the user explaining that the upload has failed

7.2 User Interface

The user interface will be how the users interact with the web application. This is achieved using buttons, file selectors, drop down menus etc. It is therefore essential that the user interface is clear and precise.

As mentioned earlier, Ng2 Admin is going to be used as it provides a professional template with plenty of extra features, if needed.

7.3 Navigation

Ng2 Admin provides a professional navigation system as standard. This includes a retractable side menu bar that is responsive in design. The model it uses for its navigation is based on various other services such as Twitter and Facebook and therefore is proven to be effective.

Elements within Ng2 Admin can be modified by changing the HTML, which will in turn allow for custom placement of objects around the template.

7.4 GUI

The GUI design for the web application will be somewhat limited, as the template used in Ng2 Admin can only be modified to an extent.

7.5 Design Mock Up

This section contains visual mock ups of the web application design, using the web service Wirefram.cc.

7.5.1 Login Screen

Figure 7 shows the design on the login screen that the user is presented with when they visit the web page. The user must type in their email address, password and click the login button, which will then give them access to the home page. Without logging in, the user is unable to gain access to the home page. There is no sign-up option on this page as the author would like the users to sign up on the mobile application beforehand.

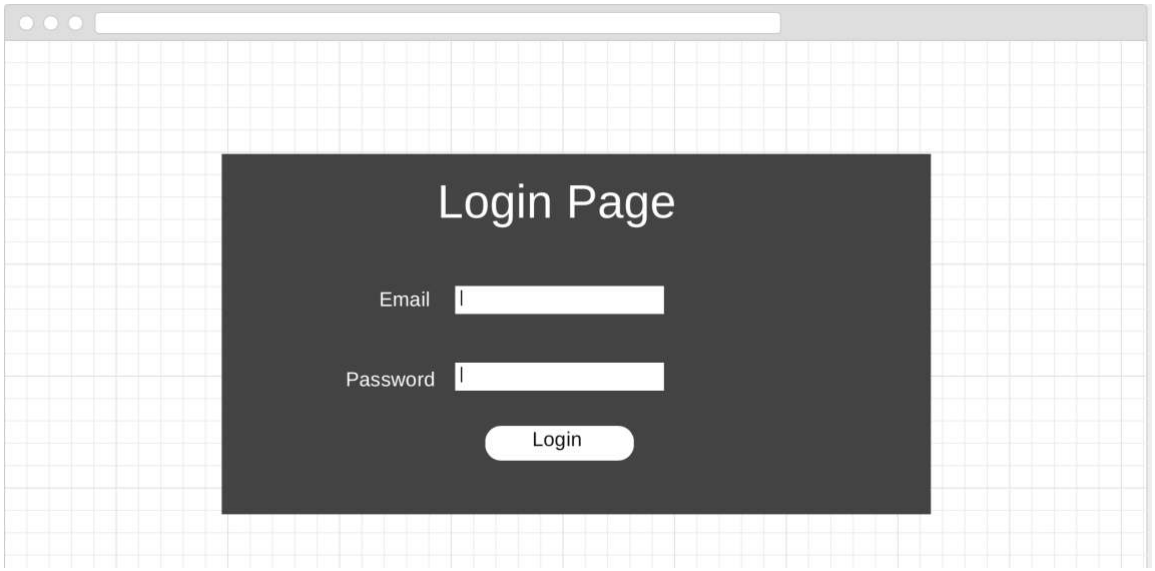

Figure 7: Web Application Login Page (Wireframe.cc, 2018)

7.5.2 Upload Screen

Once the user is passed the login screen, they are presented with upload screen which will give them all the option explained previously in the use case designs. *Figure 8* shows the design of this screen.

Validation of inputs ensures that no fields are left empty and values are between the specified range. The user will not need to fill out the entire form, but will instead be able to submit each section one at a time. The drop-down menus will be connected to the database directly.

Once a track has been submitted through the web application, the user will be able to view it from the mobile application. Tracks that are submitted to the database cannot be viewed or listened to from within the web application. It is purely a tool to allow users to upload content.

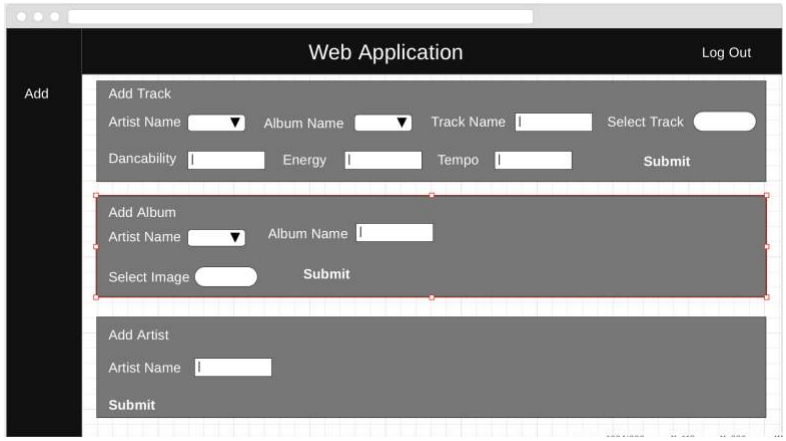

Figure 8: Upload Screen (Wireframe.cc, 2018)

8.0 MOBILE APPLICATION DESIGN

This section of the dissertation will explore the designs planned for the mobile application.

8.1 Use Case Modelling

Use case modelling will be used again to describe the suggested user interaction with the application.

8.1.1 Use Case: Creating An Account

This use case describes the process the user will go through to sign up, once the application has been opened.

Flow of events:

1. User opens the application
2. User is presented with login screen
3. User clicks sign up button
4. Users email address and password are entered
5. User clicks sign up button
6. User is taken to the home screen of the application

Error handling:

- If the account already exists on the database, the corresponding error message is presented to the user
- If a connection to the database fails, a corresponding alert is presented to the user

8.1.2 Use Case: Logging In

This use describes the process the user will go through, to login to the service, if they have already created an account.

Flow of events:

1. User opens application
2. User enters email address and password
3. User clicks login button
    a. If the user accidently clicks the sign-up button, a back button will be presented, and a sign-up error will be presented
4. Use is taken to the home screen of the application

Error handling:

- If the email address and password do not match the records on the database, and error will be presented to the user

- If a connection to the database fails, a corresponding alert is presented to the user

8.1.3 Use Case: Logging Out

This use case describes the process the user will go through to log out of the application.

Flow of events:
1. User clicks menu icon
2. User is presented with logout page
3. Logout button is clicked
4. User is taken back to the login page

Error handling:
- If audio is playing when the logout button is clicked, the playback will stop

8.1.4 Use Case: Playing A Track

This use case describes the process the user will go through to play a track once they are past the login screen.

Flow of events:
1. User has logged in and is presented with the first screen of the application
2. User clicks/ taps on a track within the view
3. The track begins to play in the applications audio player

Error handling:
- If track playback fails, a corresponding error will be presented to the user
- If a connection to the database fails, a corresponding alert is presented to the user

8.1.5 Use Case: Searching For A Track

This use case describes the process the user will go through to search for a track within the application.

Flow of events:
1. User is logged in and is presented with the first screen of the application
2. User navigates to search section
3. User enters search criteria
    a. If the search criteria do not match any track on the database, a no results alert is displayed
    b. If matching results are found, they are displayed below the search bar
4. User clicks the track to play it.

Error handling:

- If playback of the search result fails, a corresponding alert will be presented to the user
- If the search fails to connect to the database, a corresponding message if presented to the user

8.1.6 Use Case: Recommending Tracks

This use case describes the process the user will go through to use the to find recommend songs on the database.

Flow of events:

1. User is logged in and presented with the first screen in the application
2. User navigates to the explore section
3. User is presented with a screen to select artists that they like
    a. If the user cannot find an artist they are looking for, they are able to search
    b. When the user has selected an artist, a small alert notifies them that the artist has been added
4. User dismisses the view
5. User is presented with a screen containing a narrowed down list of tracks that match the style of the artists selected in the previous view
6. User clicks the preview button on the track in the list and track begins to play
7. User clicks that pause button on the track in the list and playback is paused
    a. If the user likes the track, the like button is clicked, the item is removed from the list and a small pop up notifies them that the track has been liked
    b. If the user does not like the track, the dislike button is clicked, the item is removed from the list and a small pop up notifies them that the track has been disliked
8. Step 7 is repeated until 5 tracks have been liked
9. After 5 tracks have been liked, the user is presented with another screen which will contain music from the database that matches the style attributes of the tracks they liked on the previous screen. Therefore, acting as recommendations
    a. The user can then change the settings within the recommendations to sort by different attributes
10. The track is the tapped on and begins playing in the applications audio player

Error handling:

- If the user does not select at least one artist from the artist selection page, they are presented with an alert to do so
- If a connection to the database fails, the corresponding error message is presented.
- If the audio fails to play, the corresponding error message is presented to the user

8.2 User Interface

The user interface for the application is important, as it allows the users to interact with the software through the use of buttons, menus, radio buttons etc.

The designs of these features are integrated into the Ionic framework, to ensure that they feel native on each device. The positioning of these items is also predefined within ionic to ensure the optimal positioning for its users. These can be positioned in a custom manor through the CSS. Colours for alerts, buttons, banners, etc. are also predefined to provide the best user experience possible. However, these can be modified within the CSS.

As the author will be using the ionic framework to create this application, it is understood that Ionic have put many hours of research into allowing developers to create effective user interfaces easily. It is therefore suggested that only a small amount, if any at all, of custom positioning and colouring will be used.

8.3 Navigation

The application is required to be easy to navigate, for users to be able to use the application effectively. The Ionic framework makes it easy to implement sophisticated navigation systems within applications.

The navigations available within Ionic, are based on the 3 main navigations used in mobile applications. This includes, blank application (single page), tab bar application and side menu application, *Ionic, 2018.*

After conducting research into existing music service applications, it is suggested that a tab bar application would be best suited for this type of application. *Figure 9* shows the Spotify mobile application.

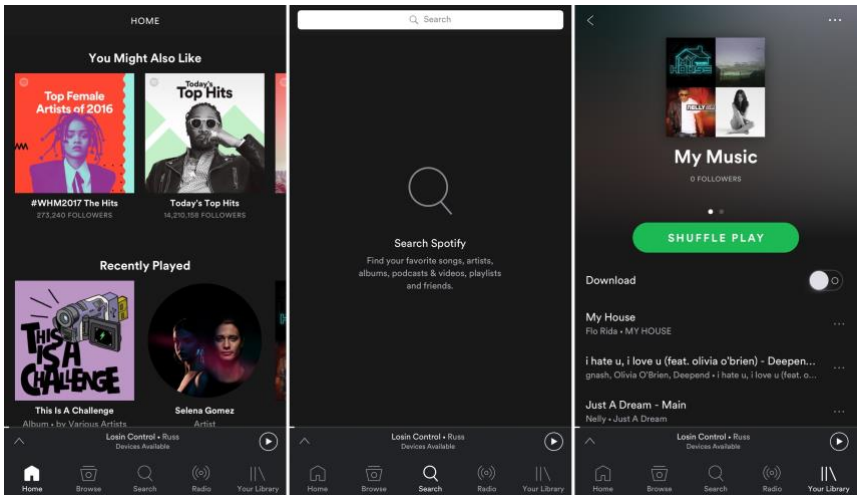

Figure 9: Spotify Mobile Application (Anthony Bouchard, 2017)

8.4 GUI

The design of the GUI will closely follow the example set by the Spotify application and it is therefore likely to be a tab bar application.

8.5 Design Mock Up

The following sections contain some visual mock ups created in Wireframe.cc, to demonstrate how the application will look visually.

8.5.1 Login Screen

Figure 10 shows the design of the login screen for the mobile application. The user can type in their email address and password, and click the login button. This will allow the user to gain access to the application. Without logging in the user is unable to use the service.

Figure 10: Login Screen (Wireframe.cc, 2018)

8.5.2 Sign Up Screen

Figure 11 shows the screen that the user is presented with if they decide to sign up to the service. They will then be prompted to enter their email address and password. Clicking the sign-up button will submit the request to the database and allow access to the application. This page is likely to be dynamic and the page buttons change, rather than a new page being displayed.

Figure 11: Sign Up Page (Wireframe.cc, 2018)

8.5.3 Home Page

Once the user has successfully logged in or signed up to the service, they will be taken to the home screen of the application. The home screen of the application will display all the tracks that are available on the database. The order of these track is likely to be either alphabetical or the order of upload. Just like the Spotify example, this page will use the album artwork associated with the tracks in the list.

This page will also be accessible using the first button on the tab bar. This page will also include the button on the top navigation bar, to enable users to navigate to the logout page. This can be seen in *figure 12.*

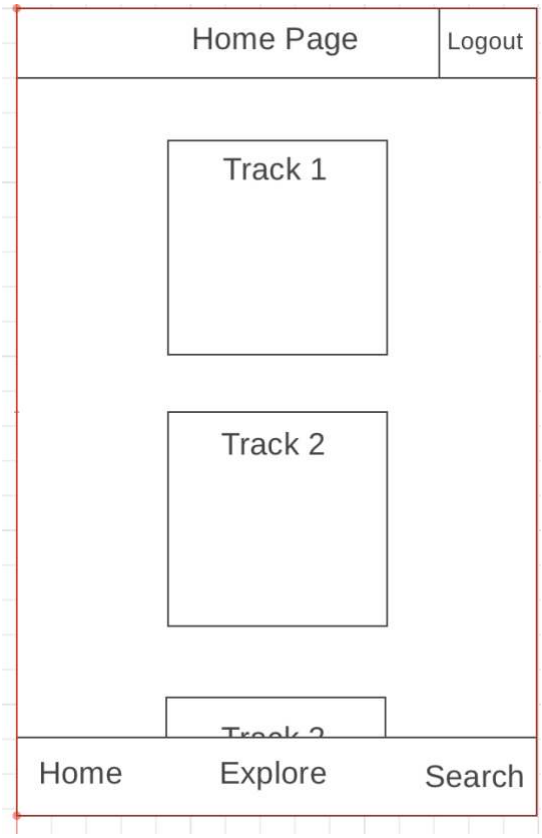

Figure 12: Home Page (Wireframe.cc, 2018)

8.5.4 Logout Page

Figure 13 shows the page that is displayed when the logout button is clicked. This is likely to be a simple page with only one button, which will in turn take the user back to the login screen.

This page will also include a back button, which will take the user back to the home page, in turn dismissing the logout view.

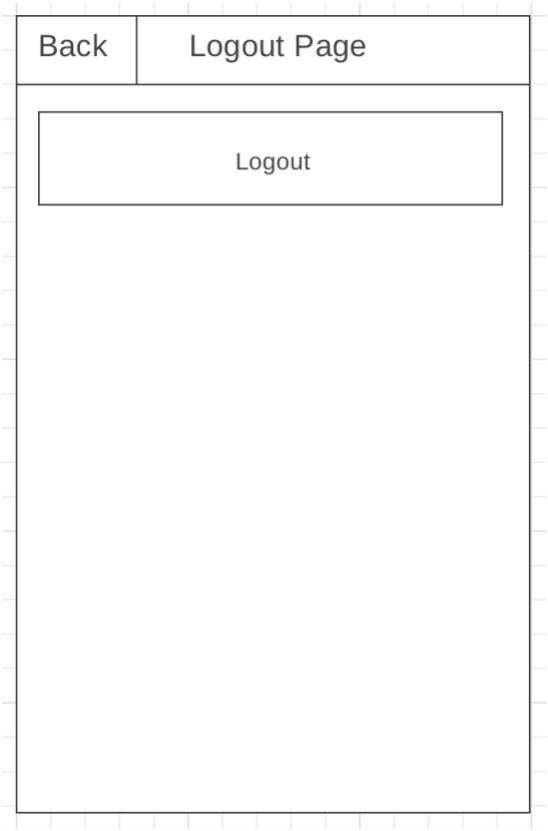

Figure 13: Logout Page (Wireframe.cc, 2018)

8.5.5 Search Page

The search page is located under the third tab bar icon within the application. This page will allow the user to search the entire database for music, based on the track title. *Figure 14* shows what the user will be presented with, when clicking on the icon in the tab bar.

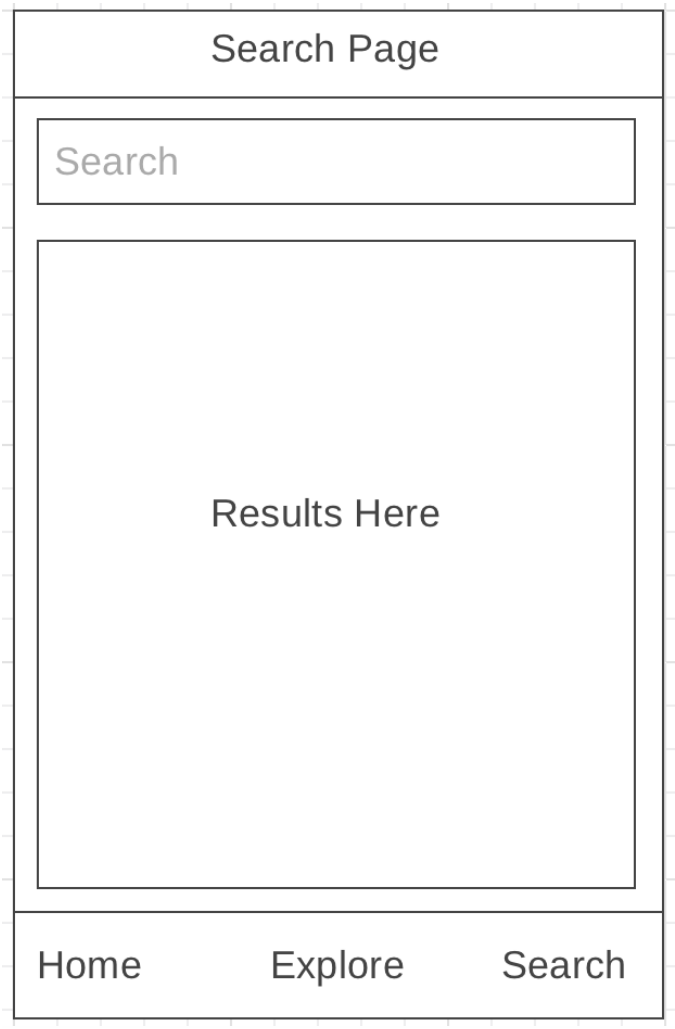

Figure 14: Search Page (Wireframe.cc, 2018)

8.5.6 Recommendation Section

As explained in section *7.1.6,* the user will need to follow a certain number of steps to have tracks from the database recommended to them. Therefore, this section will be divided further to explain each step within the Explore page.

8.5.6.1 Artist Selection Page

When tapping/ clicking the tab bar icon for the first time, the user is presented with an overlay view which prompts them to select at least one artist within a list. This view is only presented once and will need to be manually opened by the user, if they would like to change their selection. *Figure 15* shows the design of this page.

As explained in the use case, this page will need to allow users to search for artists, and at least one artist must be selected to allow for a base level to be set within the next view. Therefore, if the user clicks the close button in the top right-hand corner, before a selection has been made, an error notification will be presented. To make the selection, the user must simply click the artist name.

The artist data that is displayed on this page will need to come from an external source such as Spotify or an external JSON file. The data gathered from the user in this view will be pushed to the next view and act as a narrowing down criterion.

To ensure that the user is aware that the selection has been made, a small pop up will appear at the bottom of the screen alerting them of that. This can be seen at the very bottom of *figure 15.*

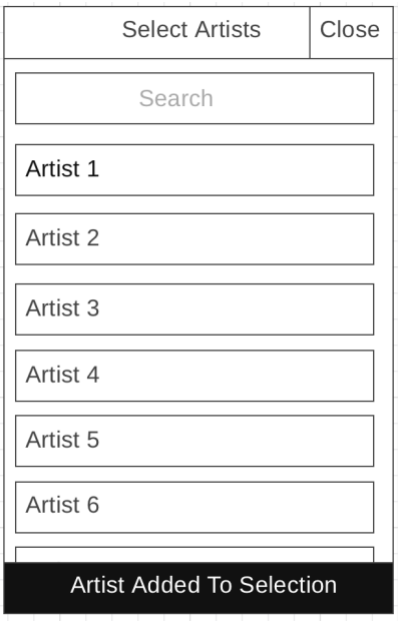

Figure 15: Artist Selection Page (Wireframe.cc, 2018)

8.5.6.2 Explore Page

When tapping on the explore icon on the tab bar, the user is taken to the explore page. As the artist selection page is only displayed once, this is what the user will be presented with if they have previously selected artists, within the artist selection view.

This design of this view will be based on the Spotify design, which will include an album art style approach. This view is also required to allow users to select whether they like or dislike a track within the view. Therefore, the author has taken a design approach similar to that of Tinder, whereby large dislike and like buttons are added to cards. This will in turn match them with someone or not match them with someone. In the authors case, it will match them or not match them with music from the database. *Figure 16* shows the Tinder style design.

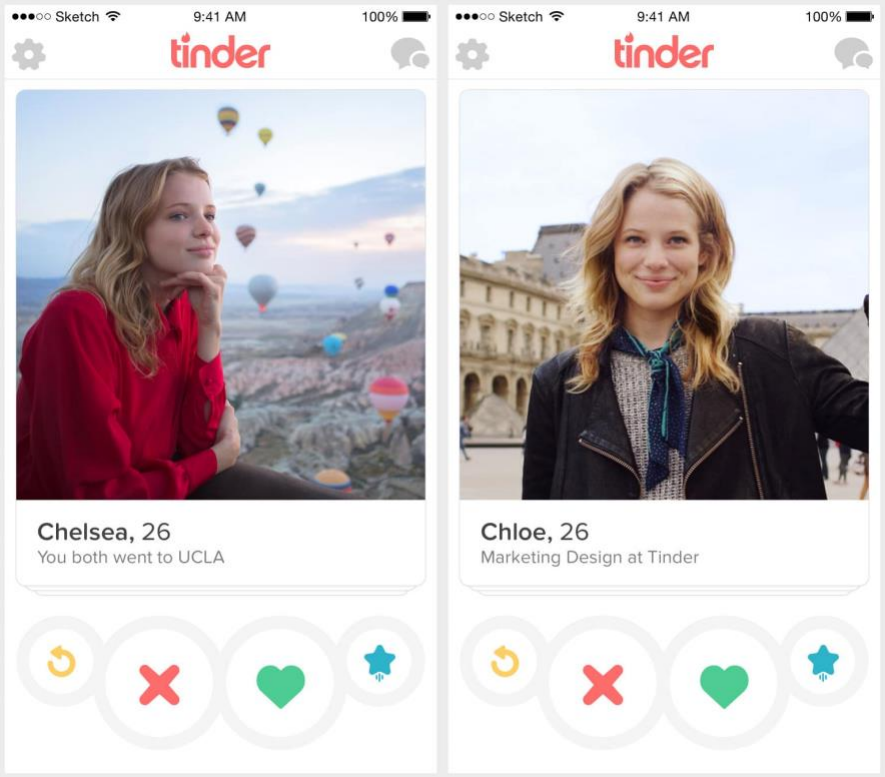

Figure 16: Tinder GUI Design (BesthookUpApps, 2018)

The cards will need to allow users to preview the track before deciding whether they like it or not, as well as allow them to pause the playback, which will need to be implemented onto each card.

Clicking on the like button will send data variables about the track to the next page. These will then be processed on the next page, to find tracks on the database of similar stature. Once the card has been liked, the card will need to be removed from the view and a small notification will need to alert the user of their decision.

Clicking on the dislike button will remove the card from the view and a small alert will need to notify the user of their decision. The data variable will not need to be sent to the next view as the user has decided they do not like the track.

The information that is displayed on this view will need to come from an external source such as Spotify or an external JSON file. This will then allow the tracks liked from the external source, to be compared against the applications database.

The author would also like to implement a way of users being able to change their artist selections made earlier. This will be done using and edit button that will be placed on the top navigation bar.

Figure 17 shows the design of how these ideas are proposed to be implemented.

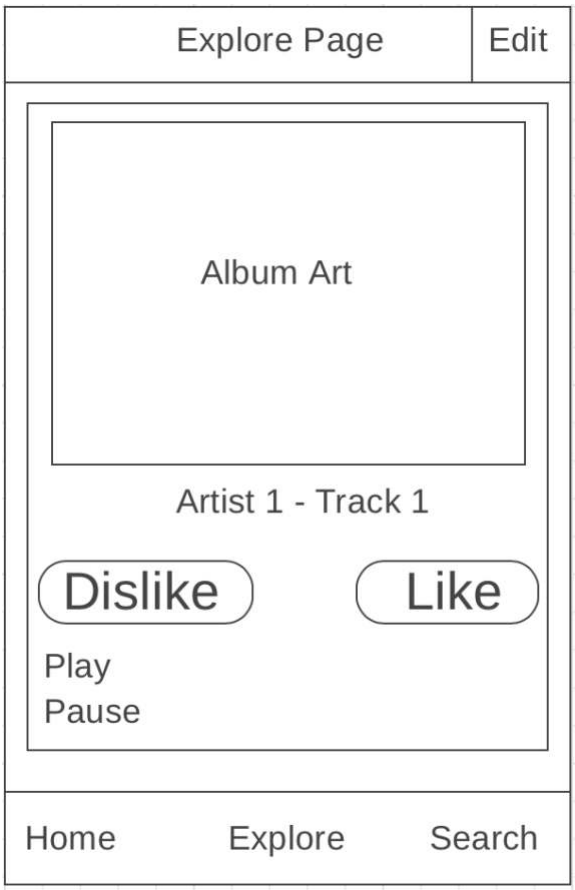

Figure 17: Explore Page (Wireframe.cc, 2018)

8.5.6.3 Recommendation Page

As explained in the use cases, the user is required to select five tracks from the previous view, before a recommendation is made. Therefore, once the user has liked 5 tracks from the previous view, the recommendation page must be automatically displayed.

The recommendation page will take the data variables gathered from the previous page and find songs on the database, that match the tracks the user has liked. The database will then return a list of music from the database, which is then displayed in a list. From here the user will be able to play the tracks in the applications audio player.

The user may decide to change the way in which the search criteria works. Therefore, a button will be added to the top navigation bar which will bring up a small menu. This small menu will allow the user to change the criteria.

A back button will also need to be added to this view to allow the user to return to the previous page. They will then be able to dislike and like more songs, until five tracks are liked again. Once five tracks have been liked, the recommendation page is automatically displayed again, and the database is queried with the new data variables from the previous page. This could potentially give a refined result.

Figure 18 shows the design of the recommendation page, with the small search criteria menu opened. This will only be presented when the user clicks the edit button in the navigation bar.

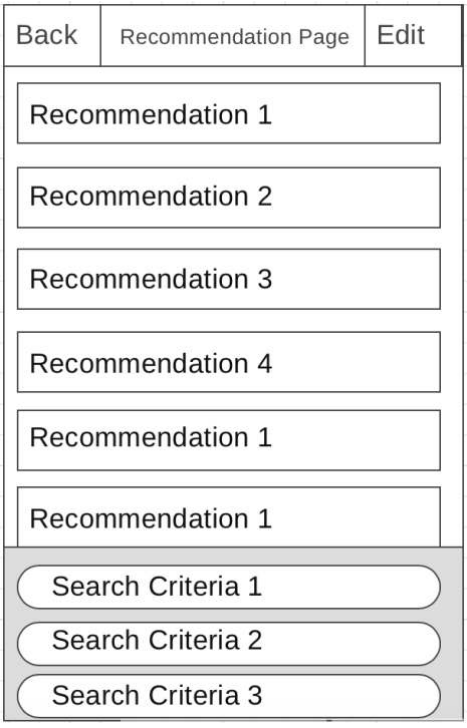

Figure 18: Recommendation Page (Wireframe.cc, 2018)

8.5.7 Player Page

The player page is arguably one of the most important parts of the application, as it lets the user know what track is playing and allows them to skip forward, backwards and pause the currently playing track.

The player page will not be directly accessible within the application but will appear once a track has been selected to play. As can be seen in *figure 19*, Spotify has a small player that appears at the bottom of the view, as well as a larger player that can be accessed by clicking on the small player. Therefore, the author will be taking a similar approach in the design of the application player.

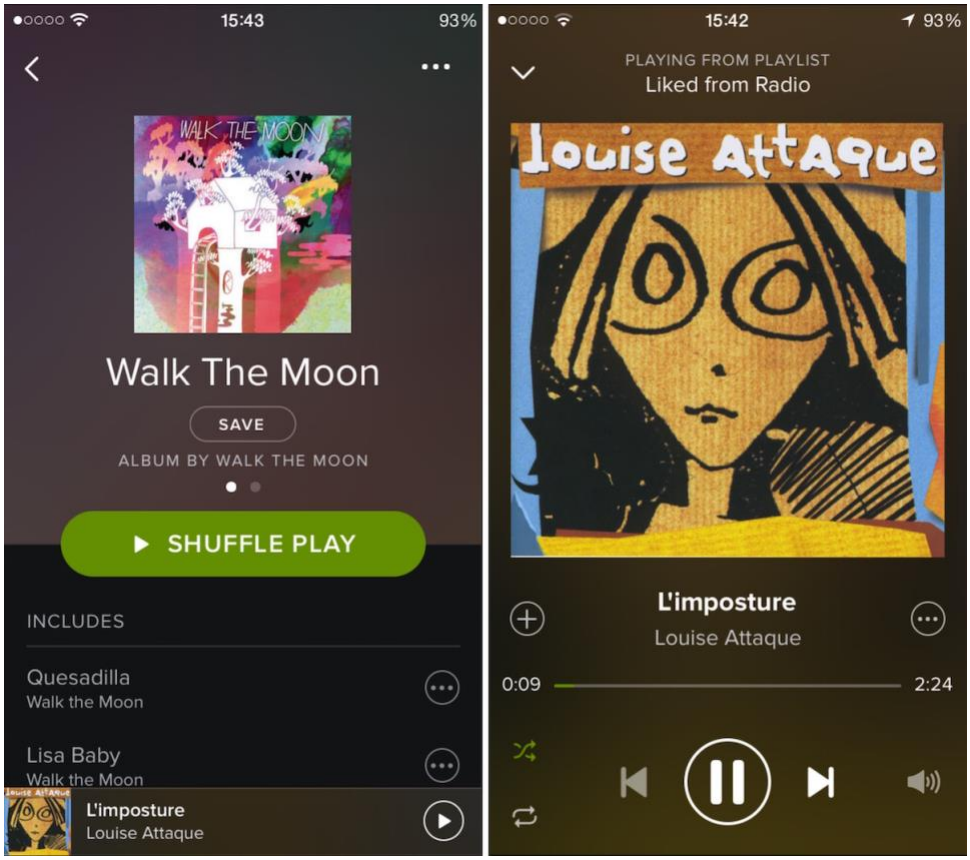

Figure 19: Spotify Player (Lory Gil, 2014)

Just like how this is implemented into the Spotify application, the author proposes to implement this into most of views within the application as well as having a larger player. Below are the designs for how this will look on each page.

Figure 20: Player Page (Wireframe, 2018)

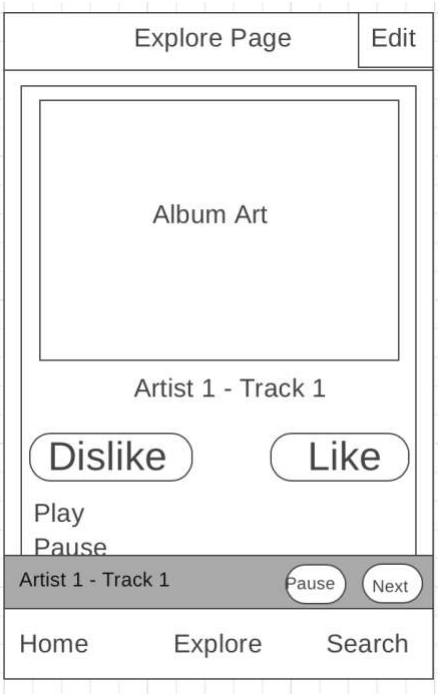

Figure 21: Explore Page With Player (Wireframe.cc, 2018)

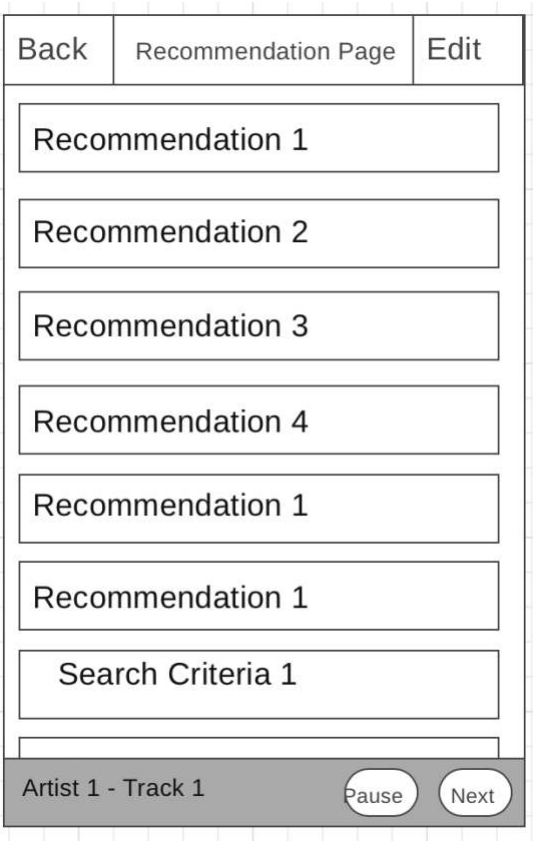

Figure 22: Recommendation Page With Player (Wireframe.cc, 2018)

Figure 23: Search Page With Player (Wireframe.cc, 2018)

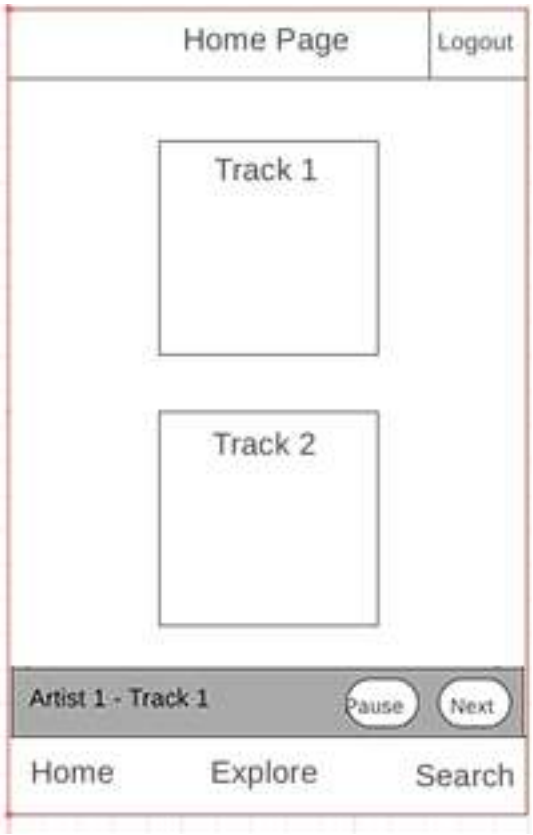

Figure 24: Home Page With Player (Wireframe.cc, 2018)

8.8 Audio Engine

As explain previously in *section 3.4,* several types of audio technologies need to be used to play audio within the application. Web Audio is going to be used throughout the application as it is simple to implement and allows the audio to be stored and loaded, before being played. This will allow time for the application to attain the audio from the database and store it temporarily in a local directory. This will ensure that playback of the audio is not intermittent.

Web Audio is going to be used in conjunction with the Ionic Audio plugin to allow audio to be routed correctly on each device.

Figure 25 shows the process of Web Audio in the application, as well as the Ionic Audio plugin routing the audio on each device.

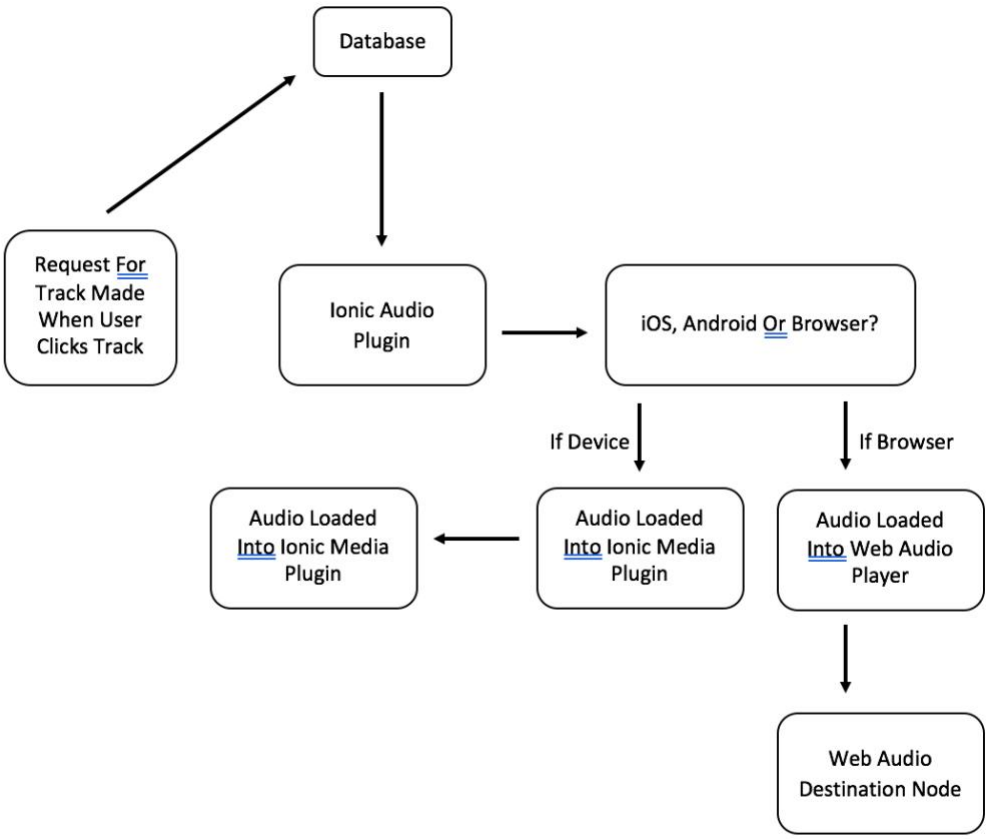

Figure 25: Audio Routing Diagram

9.0 WEB APPLICATION IMPLEMENTATION

The following chapter of this dissertation will describe the process of creating the web application. The features of the web application will be based on the web application requirement specification, found in section *4.0.*

9.1 Setting Up Ng2 Admin

To set up a working version of Ng2 Admin, two pieces of software are required to be installed. They are:

- Node.js
- Git

Once these are installed, a command line tool is required to execute the commands necessary to create the Ng2 Admin directory and boilerplate application. On a Mac, Terminal can be used, and on Windows Command Prompt can be used. In the authors case, a Mac is being used, therefore Terminal will be utilised.

9.2 Creating Ng2 Admin Boilerplate Application

To get a working version of Ng2 Admin locally, 3 commands are needed, which are entered the command line interface one at a time. These commands are as follows:

1. `git clone -b ng2-admin https://github.com/akveo/ngx-admin.git ng2-admin`
2. `cd ng2-admin`
3. `npm install`

This will create a local directory on the machine and install all the necessary dependencies that Ng2 Admin requires.

To get Ng2 Admin running within a browser the following command is used:

1. `npm start`

This will open a localhost port on the machine, which can be accessed by going to the following address within a browser:

- http://0.0.0.0:4200 or http://localhost:4200

Going to the local host address whilst Ng2 Admin is running displays the following screen:

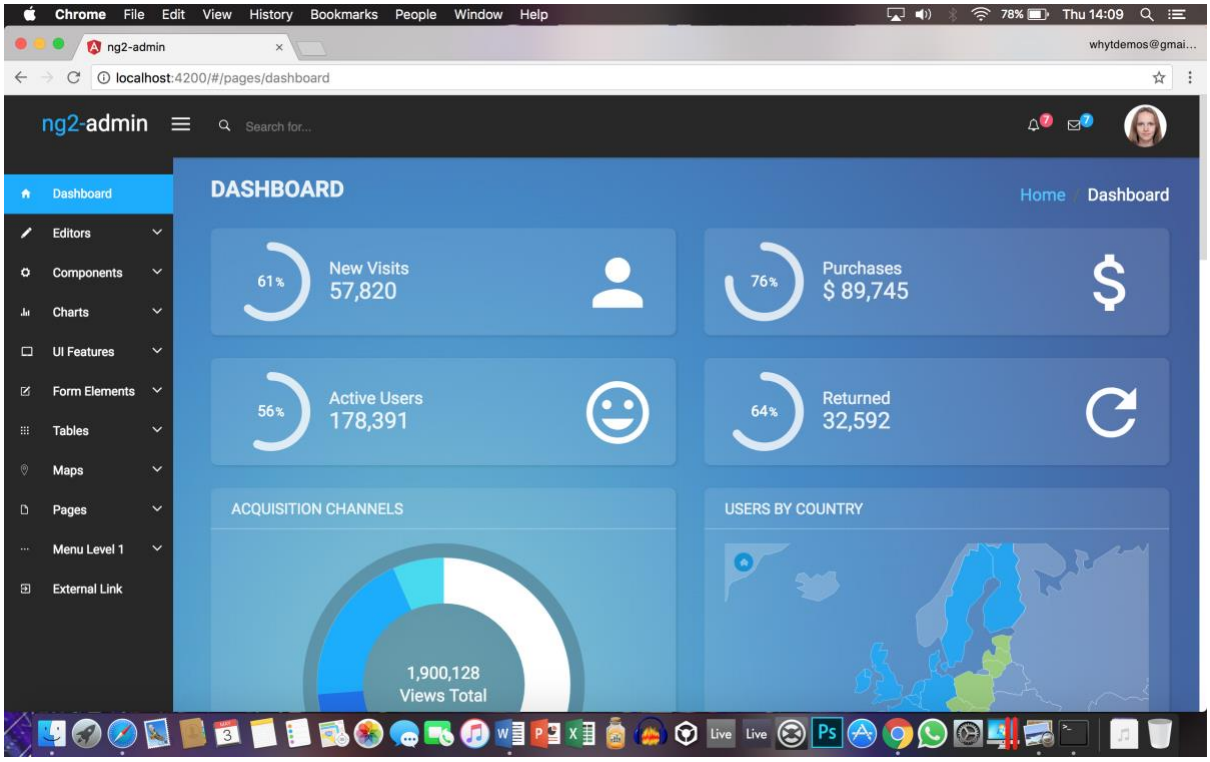

Figure 26: Ng2 Admin Boilerplate Application

9.3 User Interface And Navigation

As explained in *section 7,* Ng2 Admin comes with a nice navigation layout and user interface as standard. Therefore, only slight modifications will be needed to create the desired user interface.

9.3.1 Creating The Login Page

As stated in the web application designg, the user is required to log in. Therefore, a login page is required. This will be displayed as soon as the user

Ng2 Admin includes a login page as standard within the template but is deactivated by default. In order to display the page when the user first visits the page, a small line of code needs to be added to the src/ app/ app.component.ts, shown in *figure 27.* The code is inserted into the location that loads when the application is first opened. As this file is one of the very first to be loaded, this ensures the login page is always displayed before anything else, *Akveo, 2018.*

```
    });

    this.router.navigate(['login']);
  }
```

Figure 27: Login Page Code

For this code to work, the router needs to be imported from the Angular Router module. This is shown in *figure 28 and 29.*

```
import { Component, ViewContainerRef } from '@angular/core';
import * as $ from 'jquery';
import { Router } from '@angular/router';
```

Figure 28: Importing Angular Router

```
constructor(private _state: GlobalState,
            private _imageLoader: BaImageLoaderService,
            private _spinner: BaThemeSpinner,
            private viewContainerRef: ViewContainerRef,
            private themeConfig: BaThemeConfig,
            public router: Router ) {
```

Figure 29: Creating Accessible Router Variable

Figure 30 shows the result of this.

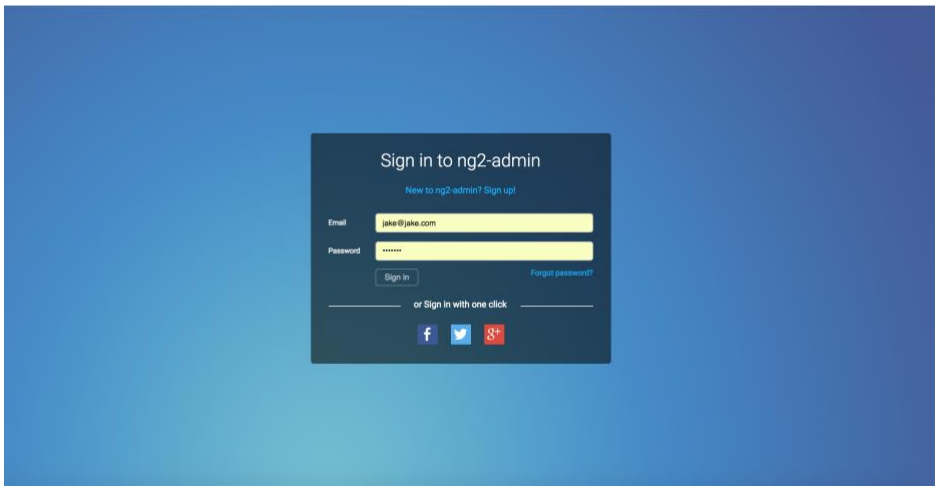

Figure 30: Ng2 Admin Login Screen

9.3.2 Removing Unwanted Login Content

As stated in the design section, the login page only needs to be capable of allowing the user to login. Creating new accounts will be done on the mobile application, therefore a lot of the content on the login screen is not needed.

To edit the content that is displayed on the login screen the HTML must be edited by heading to src/ app/ pages/ login/ login.html

Figure 31 shows the Edited HTML, with a small image inserted above the login credentials.

```
<div class="auth-main">
  <div class="auth-block" style="text-align: center">
    <h1 translate>Welcome To iugo</h1>
    <img style="width:25%" src="https://firebasestorage.googleapis.com/v0/b/finalyearproject-576e5.appspot.com/o/icon.png?alt=media&to
    <br>
    <br>
    <form [formGroup]="form"  class="form-horizontal">
      <div class="form-group row" >
        <label for="inputEmail3" class="col-sm-2 control-label" translate>{{'login.email'}}</label>

        <div class="col-sm-10">
          <input [formControl]="email" type="email" class="form-control" id="inputEmail3" placeholder="{{'login.email' | translate}}">
        </div>
      </div>
      <div class="form-group row" >
        <label for="inputPassword3" class="col-sm-2 control-label" translate>{{'login.password'}}</label>

        <div class="col-sm-10">
          <input [formControl]="password" type="password" class="form-control" id="inputPassword3" placeholder="{{'login.password' | t
        </div>
      </div>

      <div class="form-group row">
        <div class="offset-sm-2 col-sm-10">
          <button [disabled]="!form.valid" type="submit" class="btn btn-default btn-auth" translate>{{'login.sign_in'}}</button>
        </div>
      </div>

    </form>
  </div>
```

Figure 31: Login Page HTML

Figure 32 shows the web page result.

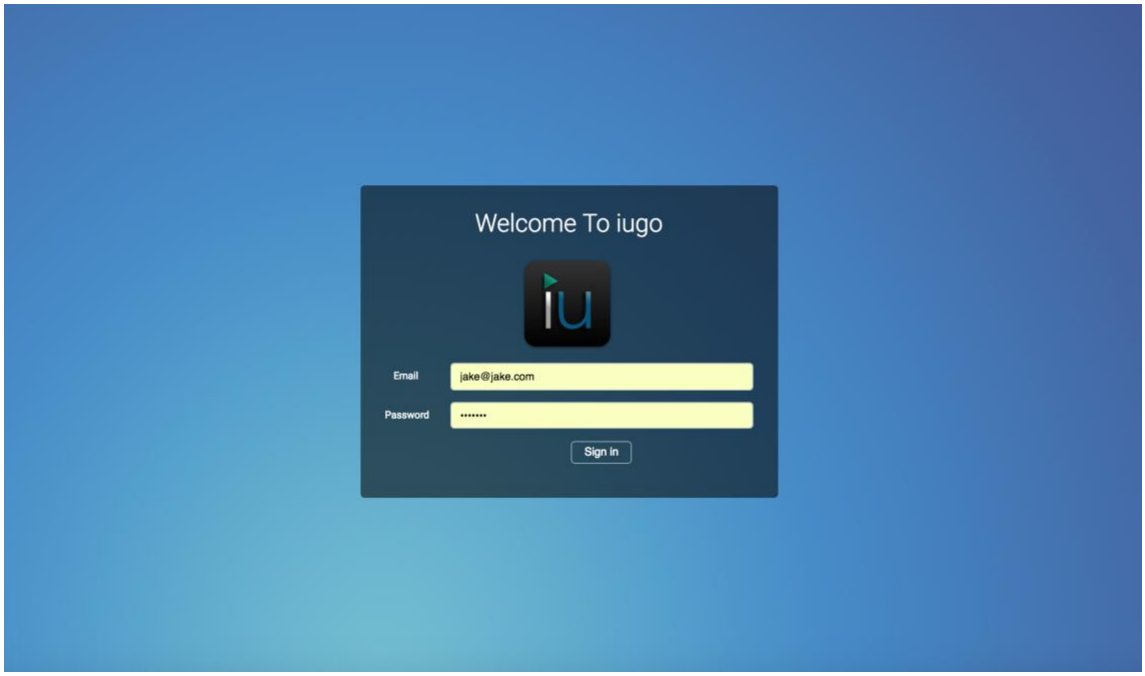

Figure 32: Web Page Result

9.3.3 Removing Unwanted Ng2 Admin Pages

The author only requires one page once logged into the application. Therefore, other elements in the menu bar will be removed and only the 'Form Elements' tab will remain. This is because the 'Form Elements' tab already contains a set layout with input forms. This can be seen in *figure 33.*

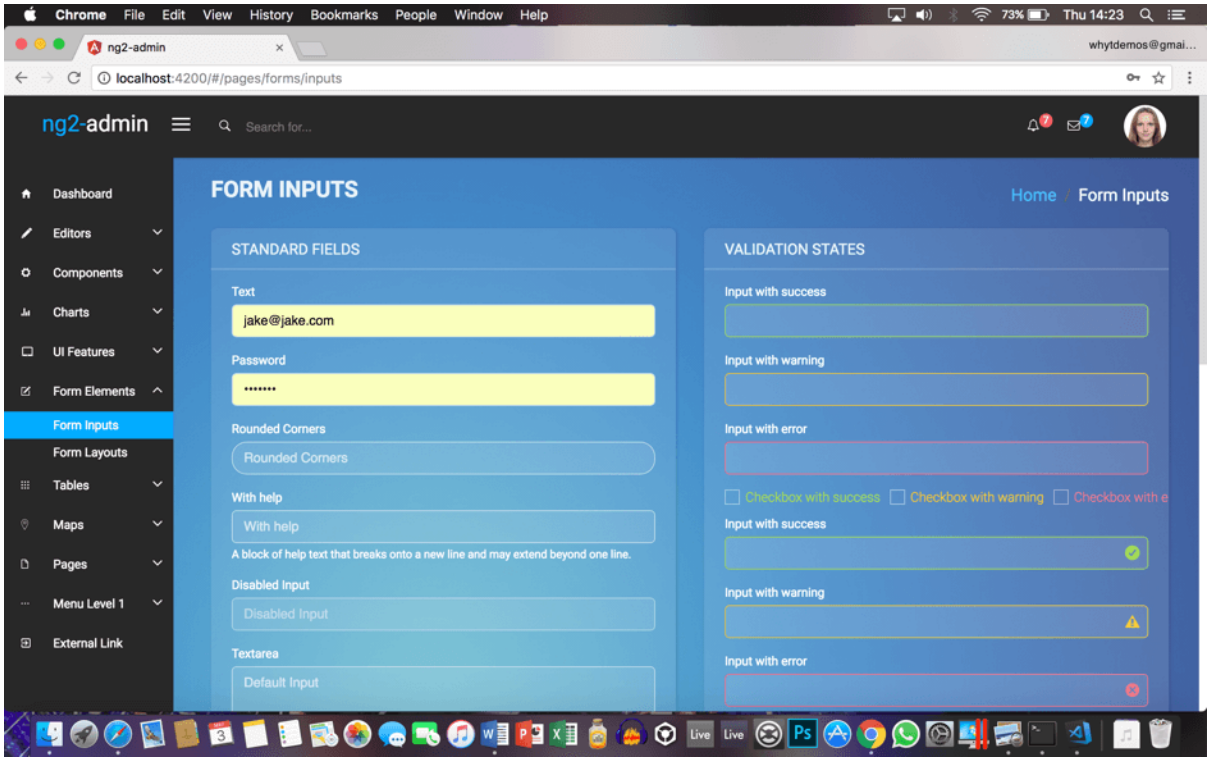

Figure 33: Ng2 Admin Form Elements Tab

Navigating to the Ng2 Admin directory/ src/ app/ pages/ pages.menu.ts, the unwanted pages can be removed by deleting the corresponding code, as seen in *figure 34. Akveo, 2018.*

```
export const PAGES_MENU = [
  {
    path: 'pages',
    children: [
      {
        path: 'dashboard',
        data: {
          menu: {
            title: 'general.menu.dashboard',
            icon: 'ion-android-home',
            selected: false,
            expanded: false,
            order: 0
          }
        }
      },
      {
        path: 'editors',
        data: {
          menu: {
            title: 'general.menu.editors',
            icon: 'ion-edit',
            selected: false,
            expanded: false,
            order: 100,
          }
        },
        children: [
          {
            path: 'ckeditor',
            data: {
              menu: {
                title: 'general.menu.ck_editor',
              }
            }
          }
```

Figure 34: Removing Unwanted Pages From Side Menu

Figure 35 shows the result of this.

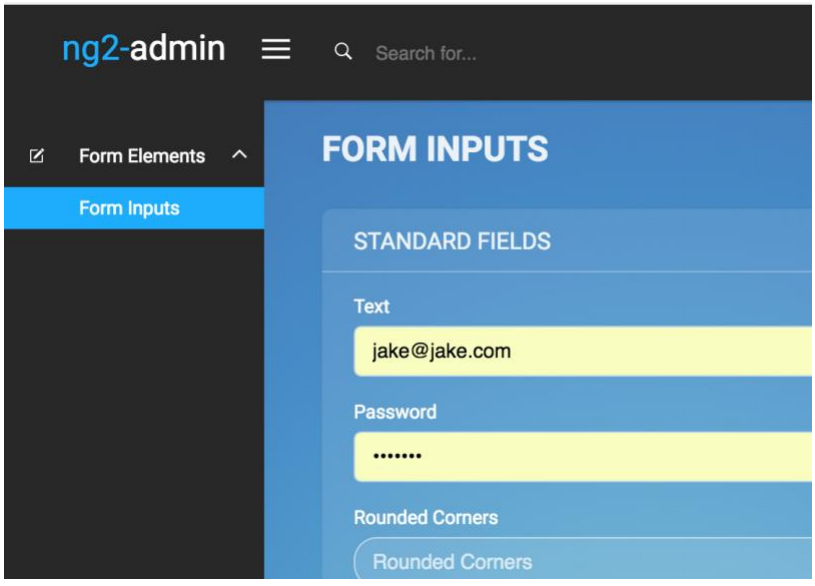

Figure 35: Menu After Unwanted Pages Have Been Removed

9.3.4 Adding Additional Forms

To create the various inputs fields that are required for the user to be able to add an artist, album and track, the 'blockForm' folder in, src/ app/ pages/ forms/ components/ layouts/ components/, is duplicated and the appropriate names are given to each.

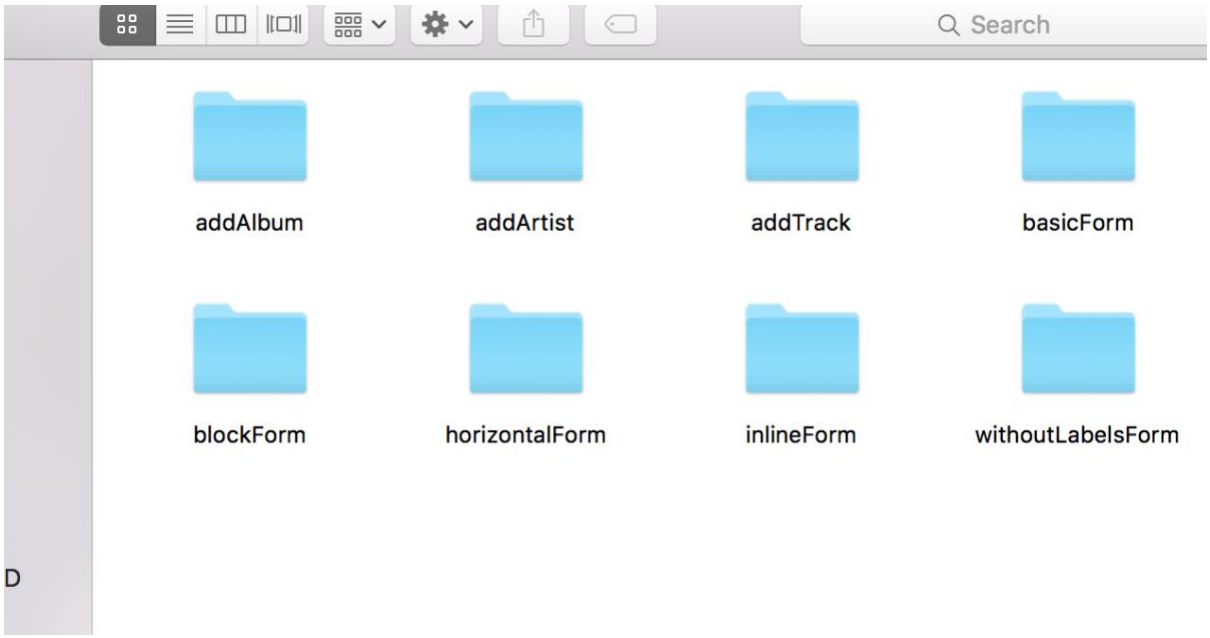

Figure 36: Block Form Duplication

It is essential that the export class and selector within the component.ts files, within these folders are updated from 'blockForm', to the appropriate name. This is to ensure the program can see it as a separate form. This can be seen in *figure 37.*

```
1   import {Component} from '@angular/core';
2
3   @Component({
4     selector: 'addAlbum-form',
5     templateUrl: './blockForm.html',
6   })
7   export class addAlbum {
8
9     constructor() {
10    }
11  }
12
```

Figure 37: Editing Forms Export Class

Once the export classes have been changed, the newly created forms need to be imported to the forms module and declared, so that they can be used. Heading to src/ app/ pages/ forms, the forms.module.ts can be found, which is where the declaration need to be made.

```
import { InlineForm } from './components/layouts/components/inlineForm';
import { BlockForm } from './components/layouts/components/blockForm';
import { addAlbum } from './components/layouts/components/addAlbum';
import { addArtist } from './components/layouts/components/addArtist';
import { addTrack } from './components/layouts/components/addTrack';
import { HorizontalForm } from './components/layouts/components/horizontalForm';
import { BasicForm } from './components/layouts/components/basicForm';
import { WithoutLabelsForm } from './components/layouts/components/withoutLabelsForm';
```

Figure 38: Importing Forms

```
@NgModule({
  imports: [
    CommonModule,
    AngularFormsModule,
    AppTranslationModule,
    NgaModule,
    NgbRatingModule,
    routing
  ],
  declarations: [
    Layouts,
    Forms,
    InlineForm,
    BlockForm,
    addAlbum,
    addTrack,
    addArtist,
    HorizontalForm,
    BasicForm,
    WithoutLabelsForm,
  ]
})
```

Figure 39: Declaring Forms

Once this has been done, the layouts.html can be edited to include the newly created forms, shown in *figure 40.*

```
<div class="row">

  <div class="col-md-12">

    <div>
      <ba-card title="Add Song" baCardClass="with-scroll">
<addTrack-form></addTrack-form>
      </ba-card>
    </div>
  </div>
```

Figure 40: Adding The Forms To The Page Within The HTML

Below is the webpage result.

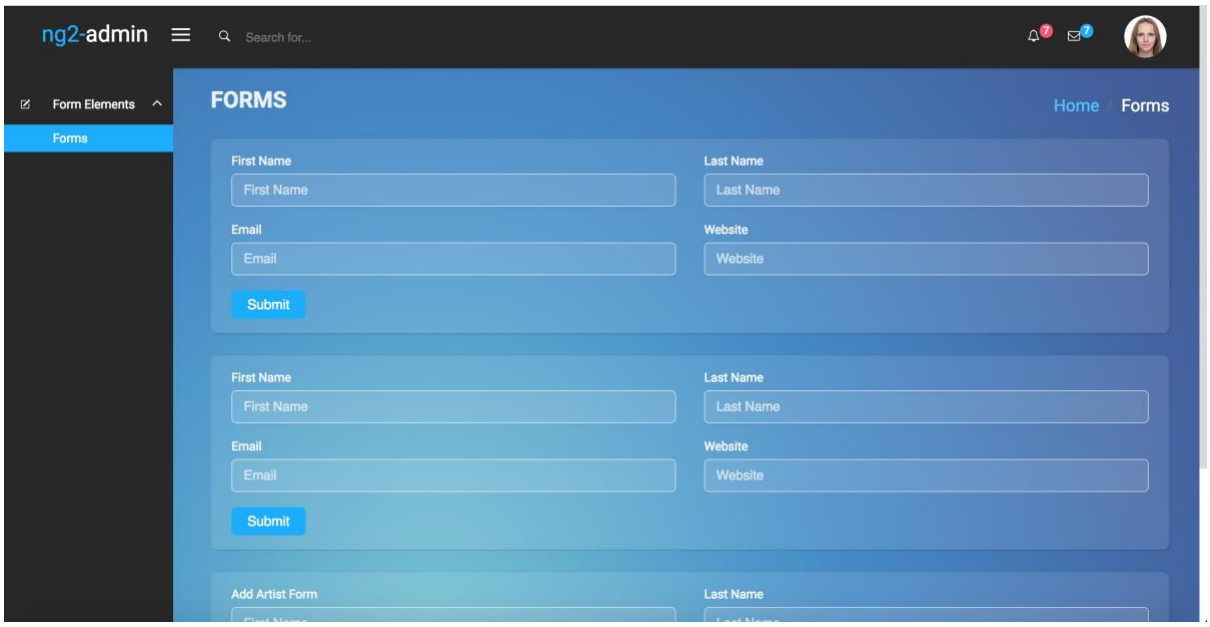

Figure 41: Webpage With Newly Created Forms

9.3.5 Designing The Forms

All that is left to do, is to edit the HTML on each form, to display the correct inputs, buttons and text as described in the designs *(section 7.5.2)*. The *figures* below show each of the forms HTML. As the author has previous experience with HTML this section of the project was fairly straightforward.

```
1  <div class="row">
2    <div class="col-sm-6">
3      <div class="form-group">
4        <label for="artistName">Artist Name</label>
5          <input type="text" class="form-control" placeholder="Artist name">
6
7      </div>
8    </div>
9
10 </div>
11
12 <div class="row">
13
14
15
16
17       <div class="col-sm-4">
18
19         <button type="submit" class="btn btn-primary">Submit</button>
20
21
22       </div>
23
24 </div>
```

Figure 42: Add Artist Form HTML

```
<div class="row">
  <div class="col-sm-4">
    <div class="form-group">
      <label for="artistName">Artist Name</label>
      <select class="form-control">
<option></option>
      </select>
    </div>
  </div>



    <div class="col-sm-4" style="min-height: 100px">
      <img>

    </div>


</div>
<div class="row">
  <div class="col-sm-4">
    <div class="form-group">
      <label for="albumName">Album/ EP Name</label>
      <input type="text" class="form-control" placeholder="Album name">
    </div>
  </div>
</div>



<div class="row">
  <div class="col-sm-12">
    <div class="form-group">
      <label for="track">Image</label>
      <input type="file" class="form-control" id="image" >
    </div>
```

Figure 43: Add Album Form HTML

```
<div class="row">
  <div class="col-sm-4">
    <div class="form-group">
      <label for="artistName">Artist Name</label>
      <select class="form-control">
<option></option>
      </select>
    </div>
  </div>
    <div class="col-sm-4">
    <div class="form-group">
      <label for="albumName">Album/ EP Name</label>
      <select class="form-control">
<option></option>
      </select>
    </div>
  </div>
      <div class="col-sm-4" style="height: 100px">
<img >

      </div>
</div>
<div class="row">
  <div class="col-sm-4">
    <div class="form-group">
      <label for="trackName">Track Name</label>
      <input type="text" class="form-control" placeholder="Song name">
    </div>
  </div>
  <div class="col-sm-4">
    <div class="form-group">
      <label for="genre">Dancability (0 - 1)</label>
 <input type="number">
      </div>
  </div>
```

Figure 44: Add Track Form HTML

The result is as follows.

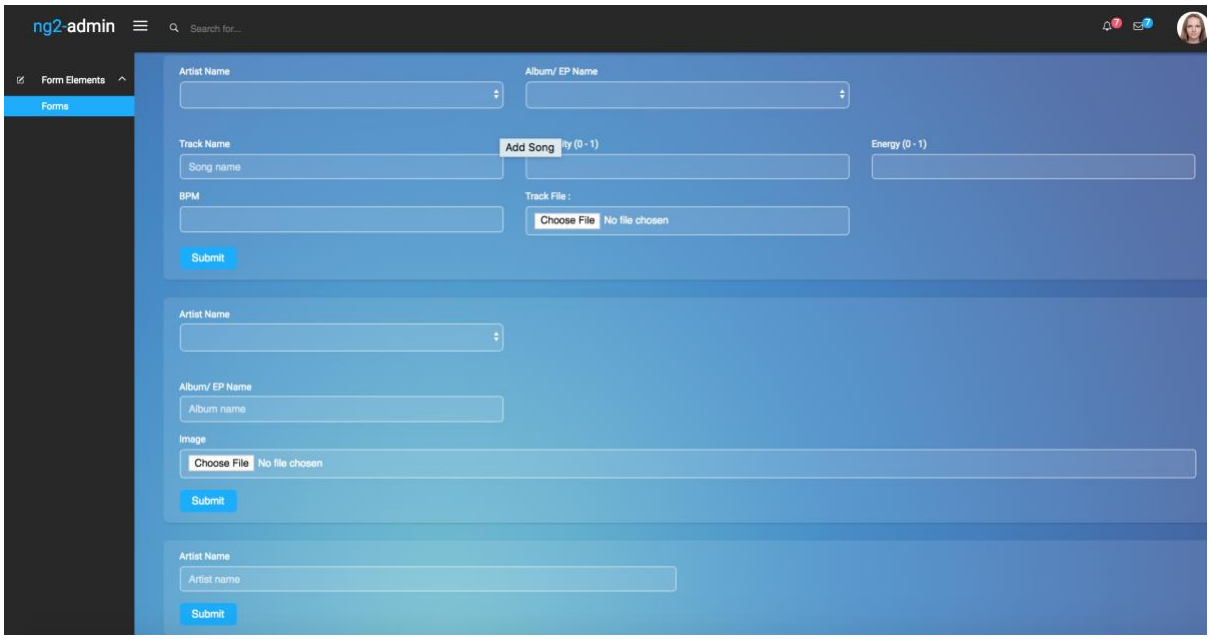

Figure 45: Completed Forms Result

9.4 Connecting To The Database

This section explains the steps necessary to connect to the database, and add functionality to the forms previously created and the login page.

To connect to the database, the Angular Fire 2 plugin must be installed. This is done adding the required package to the package.json file which is located at the root directory. *Figure 46* shows this. *F.Laurens, 2017.*

```
"@ngx-translate/http-loader": "0.0.3",
"amcharts3": "github:amcharts/amcharts3",
"ammap3": "github:amcharts/ammap3",
"angular2-datatable": "0.6.0",
"angularfire2": "^4.0.0-rc0",
"animate.css": "3.5.2",
```

Figure 46: Installing Angular Fire 2

To use the Angular Fire Plugin throughout the web application, certain attributes of the plugin must be imported and defined in the apps module. This can be found in src/ app/ app.module.ts. *Figure 47 – figure 50* shows the importation.

```
import { AngularFireModule } from 'angularfire2';
import { AngularFireDatabaseModule, AngularFireDatabase, FirebaseListObservable } from 'angularfire2/database';
import { AngularFireAuthModule, AngularFireAuth } from 'angularfire2/auth';
```

Figure 47: Importing Angular Fire Plugin

```
import { AF } from './providers/af';
import { ReversePipe } from 'ngx-pipes/src/app/pipes/array/reverse';
import * as firebase from 'firebase';
```

Figure 48: Importing Angular Fire Plugin

```
imports: [ // import Angular's modules
    AngularFireModule.initializeApp(firebaseConfig),
  AngularFireDatabaseModule,
  AngularFireAuthModule,
```

Figure 49: Defining Angular Fire Plugin

```
providers: [ // expose our Services and Providers into Angular's dependency injection
  APP_PROVIDERS, AF, ReversePipe,
],
```

Figure 50: Defining Angular Fire Plugin

Once the plugin has been imported, a key that is provided in the database console is needed to connect to the specific Firebase database. This is done by adding the code found in *figure 51*, to the app.module.ts file and code in *figure 52*, below the imported plugin.

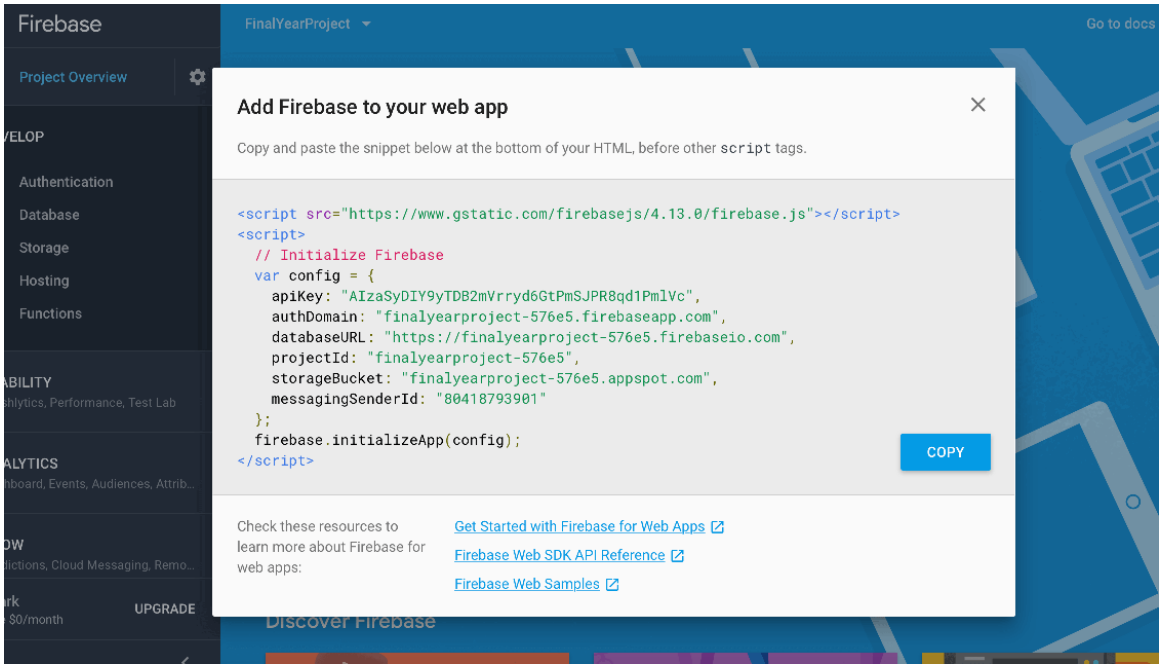

Figure 51: Firebase Console Code

```
export const firebaseConfig = {
    apiKey: "AIzaSyDIY9yTDB2mVrryd6GtPmSJPR8qd1PmlVc",
    authDomain: "finalyearproject-576e5.firebaseapp.com",
    databaseURL: "https://finalyearproject-576e5.firebaseio.com",
    projectId: "finalyearproject-576e5",
    storageBucket: "finalyearproject-576e5.appspot.com",
    messagingSenderId: "80418793901"
  };
```

Figure 52: Code Within app.module.ts

9.4.1 Adding Login Functionality

Once the plugin has been installed, functionality can be added to the login.component.ts located in src/ app/ pages/ login. First the Angular Fire plugin must be imported and added to the constructor as seen below in *figure 51. Google, 2018.*

```
1  import { Component } from '@angular/core';
2  import { FormGroup, AbstractControl, FormBuilder, Validators } from '@angular/forms';
3  import { AF } from '../../providers/af';
4  import { Router } from '@angular/router';
5
6  import { AngularFireDatabaseModule, AngularFireDatabase, FirebaseListObservable } from 'angularfire2/database';
7  import { AngularFireAuthModule, AngularFireAuth } from 'angularfire2/auth';
8  import * as firebase from 'firebase/app';
```

Figure 51: Importing Angular Fire To Login Page

```
constructor(public afService: AF, fb: FormBuilder, public router: Router)
```

Figure 52: Declaring Angular Fire In Constructor

This now allows the plugin to be called within functions. *Figure 53* is the code associated with the submit function. This function will query the database for an existing user. If the database contains the correct email address and password combination, the router.navigate is set to 'pages', which contains the home page.

```
public onSubmit(values: Object): void {
  this.submitted = true;
  if (this.form.valid) {

 this.afService.loginWithEmail(this.email.value, this.password.value).then(() => {

    this.afService.af.object('/users/' + firebase.auth().currentUser.uid).subscribe(res => {
                if ( res.role == 2 ) {
                this.router.navigate(['pages']);
              }
              else {
                                this.router.navigate(['pages']);

                //alert('Unauthorized Access');
              }

    })
  } )
    .catch((error: any) => {
      if (error) {
        alert(error);
      }
    });
```

Figure 53: Login Code

As this code is assigned to a functioned called “onSubmit”, it can be assigned to the login page HTML button, which will submit the details and allow the user to login. The HTML is shown in *figure 54.*

```
<div class="form-group row">
 <div class="offset-sm-2 col-sm-10">
  <button [disabled]="!form.valid" type="submit" class="btn btn-default btn-auth" translate>{{'login.sign_in'}}</button>
 </div>
</div>
```

Figure 54: Login HTML Submit Button

9.4.2 Adding ‘Add Artist’ Functionality

Just like before, the Angular Fire plugin needs to be imported to the .component.ts file before it can be utilised. *Figure 55* shows the function called “upload”. First it will check for the input field and ensure there is text within it, if there isn’t, an alert is shown to user. The text within the field is then sent to the database, and stored in the directory called artists. *Google, 2018.*

```
upload() {
if ( this.artistName == '' || this.artistName == null || this.artistName == undefined ) {
  alert('Artist Name Cannot be Empty');

}

else {
  this.spinner.show();

  const storageRef = this.af.af.app.storage().ref();

      const success = false;

        const af = this.af.af;
        const artistName = this.artistName;


          af.list('/artists').push({
          name: artistName,
           timestamp: Date.now(),

        approved: true,
        provider: 'Admin'
          });

       this.artistName = '';

this.spinner.hide();
        };
      }
     }
```

Figure 55: Add Artist Code

The corresponding code is then added to the Add Artist HTML button, seen in *figure 56.*

```
<div class="col-sm-4">
  <button type="submit" (click)="upload()" class="btn btn-primary">Submit</button>
```

Figure 56: Add Artist HTML With Functionality

9.4.3 Adding 'Add Album' Functionality

A similar approach is also taken with the Add Album form. After the Angular Fire plugin has been imported, another function is created called "upload". The code within the function is like before, however an image and text needs to be sent to the database and stored.

First a drop-down menu is created to show a list of all artists that the user can add an album to. If the artist they are looking for isn't within the list, they will need to add one using the previous step.

To achieve this drop-down menu, a FirebaseListObservable is needed. This will constantly look for specified data and update itself automatically. *Figure 57 and 58*, shows the code needed to populate the drop-down menu with artists from the database. *Google, 2018.*

```
export class addAlbum {
  public artists: FirebaseListObservable < any > ;
```

Figure 57: Firebase List Observable

```
  this.artists = this.af.af.list('/artists');
}
```

Figure 58: Database Location Of Firebase List Observable

An *ngFor attribute can then be assigned to the HTML, which will allow it to display the artists from the artist directory, in the drop-down menu. *Figure 59* shows the HTML and *figure 60* shows the working menu. *Angular, 2018.*

```
<div class="row">
  <div class="col-sm-4">
    <div class="form-group">
      <label for="artistName">Artist Name</label>
      <select class="form-control" [(ngModel)]="curartist" >
<option  *ngFor="let artist of artists | async" id="artistName" [ngValue]="artist">{{artist.name}}</option>
      </select>
    </div>
  </div>
```

Figure 59: Drop Down Menu HTML

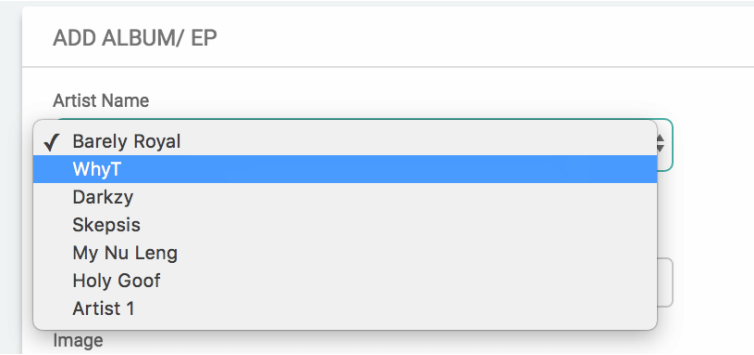

Figure 60: Working Drop Down Menu

Once the user has selected an artist to add an album to, they then need to add an album image and an album name using the fields on the form. First the fields are checked to ensure they are not empty, if they are, an alert is shown to the user. The image and text are then pulled from the HTML elements and stored within variables.

When the upload button is clicked, the image file is then uploaded to the database storage and a path/ link is created for the image file, *Google, 2018.* A new directory called “albums” is then created and an album is created with name specified by the user. The album contains data such as the artist whose album it is, the name of the album, the time it was uploaded and the path/ location of the uploaded image. This can be seen in *figure 61. Google, 2018.*

```
upload() {
  if (this.albumName == '' || this.albumName == null || this.albumName == undefined) {
    alert('Album Name Cannot be Empty');
  } else if (( < HTMLInputElement > document.getElementById('image')).files[0] == undefined ||
    ( < HTMLInputElement > document.getElementById('image')).files[0] == null) {
    alert('Album Image Cannot be Empty');
  } else {

    this.spinner.show();

    const storageRef = this.af.af.app.storage().ref();

    for (const selectedFile of [( < HTMLInputElement > document.getElementById('image')).files[0]]) {
      const path = `/${this.curartist.name}/${this.albumName}`;
      const iRef = storageRef.child(path);
      const albumName = this.albumName;
      iRef.put(selectedFile).then((snapshot) => {
        this.af.af.app.storage().ref(path)
          .getDownloadURL().then(url => {

            this.af.af.list('/albums').push({
              artist: this.curartist.$key,
              artistName: this.curartist.name,
              name: albumName,
              timestamp: Date.now(),
              image: url,
              path: path,
              approved: true,
              provider: 'Admin'
            });

          });
        this.albumName = '';
        ( < HTMLInputElement > document.getElementById('image')).files[0] = null;
      });
```

Figure 61: Adding Album Upload Code

Figure 62 shows the result within the database console.

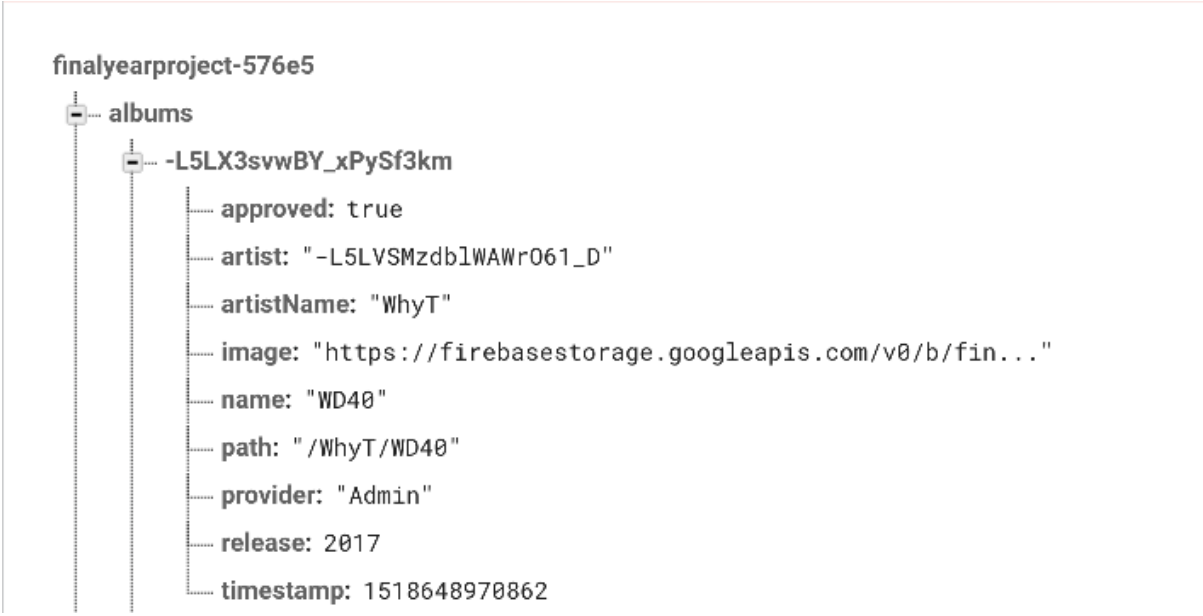

Figure 62: Database Console After Upload

9.4.4 Adding 'Add Track' Functionality

As stated in *section 3.5.2*, recommendation algorithms need variables to compare. Spotify compares its tracks based on many track features that they automatically generated for each of their tracks, based on a secret algorithm. This includes dancability, energy and tempo. As the author is unable to obtain the algorithms, the user will simply be given the option to type in the dancability, energy and tempo of the track they are uploading. For a music service, this is not a great way to implement this feature, as users could miss-categorise the track being uploaded, leading to inaccurate recommendations being made. However, as this project is a proof of concept and all the music on the server is duplications of one track, this method is more than suitable. *Google, 2018.*

Just like in the previous section, drop down menus containing the artist name and album name are needed. *Figure 63* and *figure 64* shows the FirebaseListObservable as well as the location within the database.

```
export class addTrack {
  public artists: FirebaseListObservable<any>;
  public albumsdb: FirebaseListObservable<any>;
```

Figure 63: Firebase List Observables Within Add Track

```
  constructor(public af: AF, public spinner: BaThemeSpinner) {
    this.curartist = { $key: 0 };
this.artists = this.af.af.list('/artists');
this.albumsdb = this.af.af.list('/albums', { preserveSnapshot: true });
```

Figure 64: Firebase List Observable Location

```
    <label for="artistName">Artist Name</label>
    <select class="form-control" [(ngModel)]="curartist" (ngModelChange)="change($event)">
<option  *ngFor="let artist of artists | async" id="artistName" [ngValue]="artist">{{artist.name}}</option>
    </select>
  </div>
 </div>
  <div class="col-sm-4">
  <div class="form-group">
    <label for="albumName">Album/ EP Name</label>
    <select class="form-control" [(ngModel)]="curalbum" >
<option  *ngFor="let album of albums" id="albumName"  [ngValue]="album">{{album.val().name}}</option>
    </select>
```

Figure 65 shows the HTML for the drop-down menus.

```
<div class="row">
  <div class="col-sm-4">
    <div class="form-group">
      <label for="artistName">Artist Name</label>
      <select class="form-control" [(ngModel)]="curartist" (ngModelChange)="change($event)">
<option  *ngFor="let artist of artists | async" id="artistName" [ngValue]="artist">{{artist.name}}</option>
      </select>
    </div>
  </div>
    <div class="col-sm-4">
    <div class="form-group">
      <label for="albumName">Album/ EP Name</label>
      <select class="form-control" [(ngModel)]="curalbum" >
<option  *ngFor="let album of albums" id="albumName"  [ngValue]="album">{{album.val().name}}</option>
      </select>
    </div>
  </div>
```

Figure 66: Add Track Drop Down Menu HTML

The same approach is taken to the previous section, however a few more lines of code are required. *Figure 66* shows the validation added to the HTML items, to ensure that fields or files are not left empty, *Angular, 2018*. Again, this is added to a function called “upload”, which will be associated with the submit button on the form.

```
upload() {


if ( this.curalbum == '' || this.curalbum == null || this.curalbum == undefined ) {
  alert('Album Cannot be Empty');
}


 else if ( this.trackName == '' || this.trackName == null || this.trackName == undefined ) {
  alert('Track Name Cannot be Empty');
}

else if ( this.curartist == '' || this.curartist == null || this.curartist == undefined ) {
  alert('Artist Name Cannot be Empty');
}

else if ( this.dancability == '' || this.dancability == null || this.dancability == undefined || this.dancability < 0 || this.dancability > 1)
  alert('The track must have a DANCABILTY rating between 0 & 1.');
}

else if ( this.energy == '' || this.energy == null || this.energy == undefined || this.energy < 0 || this.energy > 1 ) {
  alert('The track must have an ENERGY rating between 0 & 1.');
}

else if ( this.bpm == '' || this.bpm == null || this.bpm == undefined ) {
  alert('BPM Cannot be Empty');
}

else if ( (<HTMLInputElement>document.getElementById('track')).files[0] == undefined
|| (<HTMLInputElement>document.getElementById('track')).files[0] == null) {
  alert('Track File Cannot be Empty');
}
```

Figure 66: Add Artist Validation

If the criteria meet the requirements of the validation, the next section of the code is executed. In *figure 67*, a directory called “tracks” is created and a similar approach to the previous section is taken. The track name is added, as well as the artist whose track it is and the album it belongs to. The energy, dancability and tempo are also added to the database entry, as well as the location/ path to the audio file.

```
else {

  this.spinner.show();

  const storageRef = this.af.af.app.storage().ref();

      const success = false;
      // This currently only grabs item 0, TODO refactor it to grab them all
      for (const selectedFile of [(<HTMLInputElement>document.getElementById('track')).files[0]]) {
        // Make local copies of services because "this" will be clobbered
        const af = this.af.af;
        const path = `/${this.curartist.name}/${this.curalbum.val().name}/${this.trackName}`;
        const iRef = storageRef.child(path);
        const dancability = this.dancability || 0;
        const energy = this.energy || 0;
        const bpm = this.bpm || 0;
        const trackName = this.trackName;
        iRef.put(selectedFile).then((snapshot) => {
          this.af.af.app.storage().ref(path)
            .getDownloadURL().then(url => {
              af.list('/tracks').push({
                album: this.curalbum.key,
                albumArt: this.curalbum.val().image,
                albumName: this.curalbum.val().name,
                artist: this.curartist.$key,
                artistName: this.curartist.name,
                dancability: dancability,
                energy: energy,
                tempo: bpm,
                name: trackName,
                timestamp: Date.now(),
                url: url,
                path: path,
                chosen: true,
                approved: true,
                provider: 'Admin'
```

Figure 67: Track Upload Code

Figure 68 shows the result within the database console.

Figure 68: Track Section Within Database

9.5 Styling Ng2 Admin

Ng2 Admin comes with 3 themes as standard, and can be easily changed by editing the CSS file. These are Blur, Ng2 and Mint. Navigating to src/ app/ theme/ sass/ conf/ conf.scss, will allow one of the three themes to be set. *Figure 69* shows the location of the code that needs to be changed in order to change the theme. *Akveo, 2018.*

```
1   @import 'mixins';
2   @import 'colorSchemes/ng2';
3   @import 'variables';
4
```

Figure 69: CSS Code Location

In the authors case, mint is to be used, which produces the result in *figure 70.*

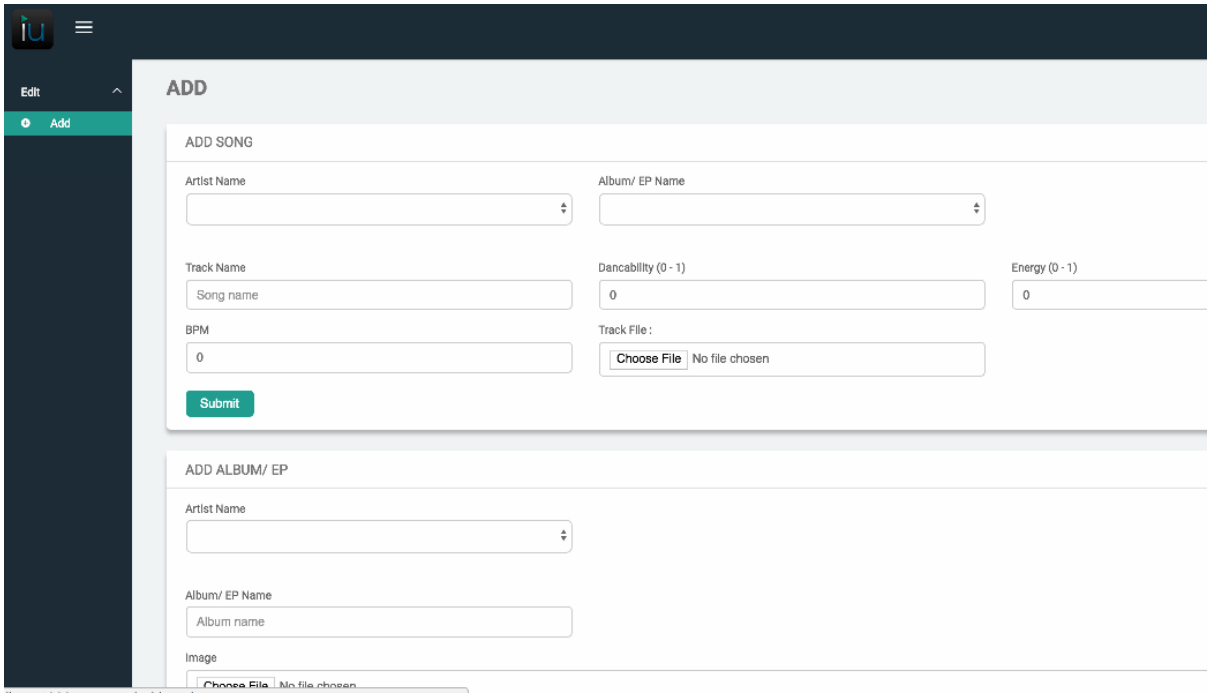

Figure 70: Ng2 Admin With Mint Theme

10.0 MOBILE APPLICATION IMPLEMENTATION

The following chapter will explain the steps taken to create the mobile application, that will utilise the requirement specification mentioned in *section 5.0.*

10.1 Installing The Ionic Framework

As mentioned previously, the mobile application is going to be created using the Ionic framework. As Ionic is a command line tool, a few things need to be installed first.

Node must be installed which can be found on the node.js website, which will then allow the command line tool to find and install the Ionic framework, *Ionic, 2018*. Once node is installed, the following command must be used within the chosen command line tool, in the authors case this is Terminal for Mac OS:

- $ npm install -g ionic cordova

10.2 Setting Up An Ionic Project

Once the Ionic framework has been installed, the following command is used to create a project, *Ionic, 2018*:

- $ ionic start {app name} {app type}

As mentioned in *section 8.3,* there are 3 main app types that can be used. This is specified in the above "{app type}". As the author is creating a tab bar application, "tabs" is used.

After executing this command, the developer is asked whether they would like to integrate the app with Cordova, to target native iOS and Android. If the developer selects no, the application can only be run within the browser. If yes is selected, all platforms can be targeted, as seen in *figure 71.*

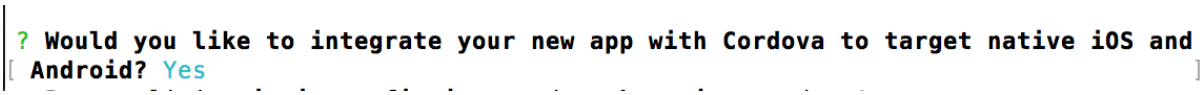

Figure 71: Cordova Installation

To get the project running within the browser for testing purposes, the project directory must be specified, and the serve command must be executed. This is achieved using the following commands, *Ionic, 2018*:

- $ cd MyIonicProject/
- $ ionic serve

Figure 72, shows the result of executing the above commands within the browser.

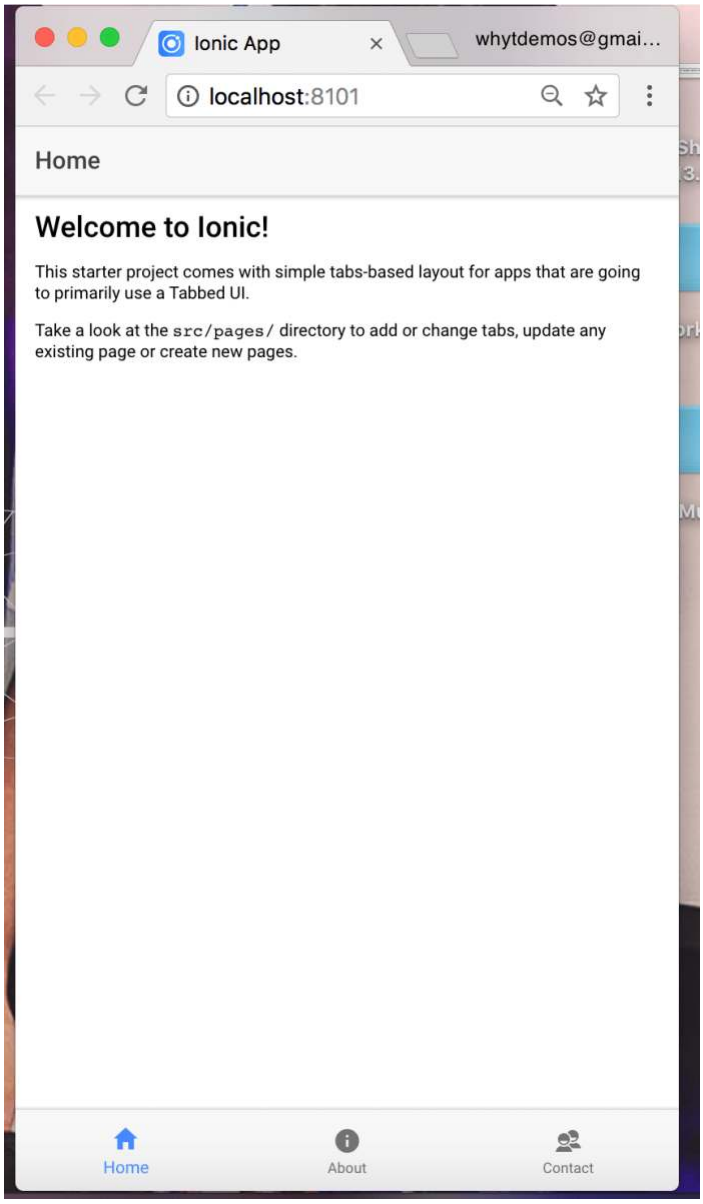

Figure 72: Boilerplate Ionic Project Running In Browser

To get the project running on a physical device, the application must be code signed for each platform (Android and iOS), which the author will not go into as it is not entirely relevant. However, to allow the Ionic project to be code signed, it must first be compiled into the correct format for each platform. For iOS, the following command is used, *Ionic, 2018*:

- `ionic cordova build ios`

And for Android, the following command is used, *Ionic, 2018*:

- `ionic cordova build android`

The newly compiled project can be found within the project directory, to then be code signed and installed to the desired platform.

Figure 73 shows the boilerplate application running on an iOS device, after being code signed and installed.

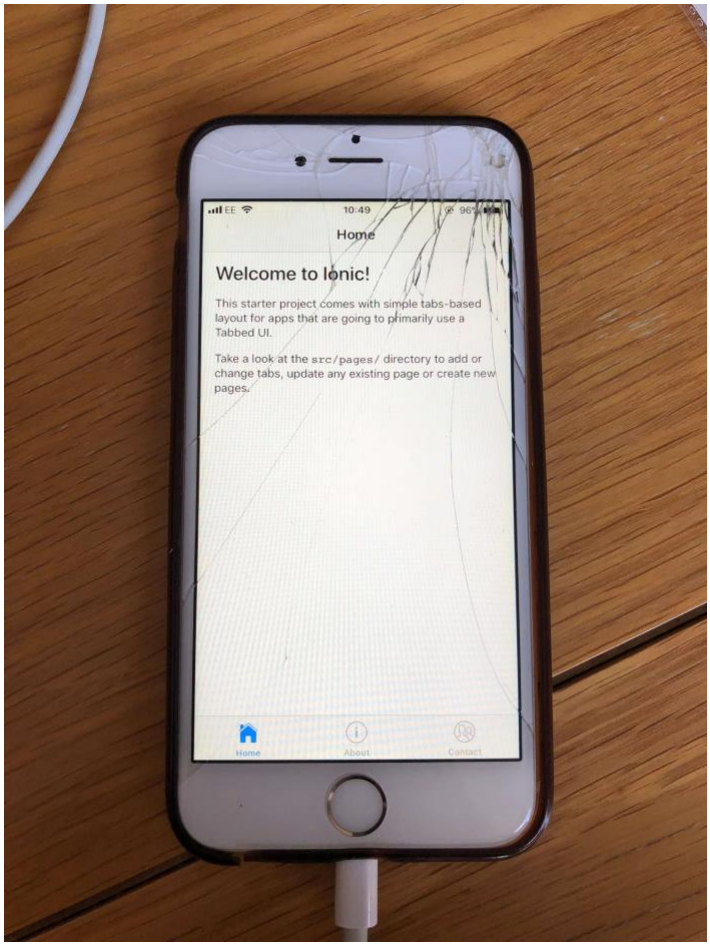

Figure 73: Boilerplate Ionic Project On iOS Device

10.3 User Interface And Navigation

As mentioned in *section 8,* the Ionic framework is designed to allow developers to create native feeling apps, whether the user is on Android or iOS. Therefore, UI elements such as buttons and search bars will have very little styling modifications added to them. As for navigation, the tab bar model provided when creating an Ionic project will be used, as well as a few extra pages. These will be used for displaying information and data previously mentioned in the design section.

As the mobile application and web application both use the Angular development technology, techniques and code can be reused across both platforms. Therefore, some parts within the following section may not be explained in as much detail, if previously mentioned within the web application implementation.

10.3.1 Creating The Required Pages

To create a new page within Ionic the following command is used in the directory of the Ionic project, *Ionic, 2018*:

- $ ionic generate page {page name}

This creates a new directory within src/ pages/ {page name}. By using this method, all required files are created for the page, including the .ts for adding function, .scss for styling and .html for element placement.

This method is similar to the method previously used in the Ng2 Admin to add pages, however, renaming the page selector and page export is not needed. The page still needs to be manually declared within the app module, which can be found in src/ app/ app.module.ts, *Ionic, 2018.*

Figure 74 shows the “pages” directory with all the required pages stated in the design section.

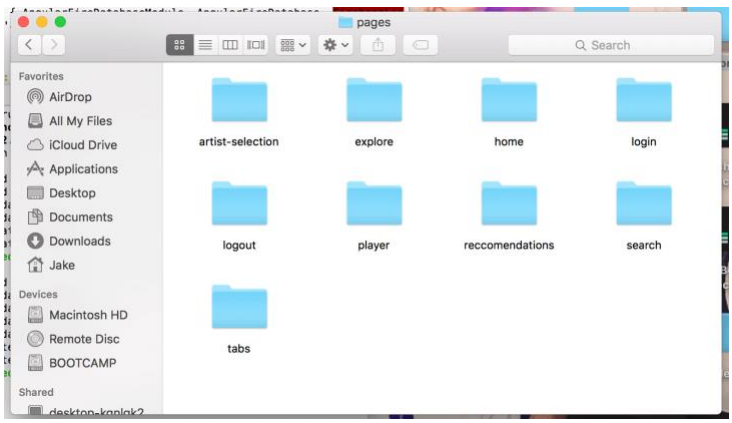

Figure 74: Pages Directory
Figure 75 and 76 show the declarations made in the app.module.ts file.

```
//Page declarations
import { HomePage } from '../pages/home/home';
import { TabsPage } from '../pages/tabs/tabs';
import { LoginPage } from '../pages/login/login';
import { PlayerPage } from '../pages/player/player';
import { LogoutPage } from '../pages/logout/logout';
import { SearchPage } from '../pages/search/search';
import { ExplorePage } from '../pages/explore/explore';
import { ArtistSelectionPage } from '../pages/artist-selection/artist-selection';
import { ReccomendationsPage } from '../pages/reccomendations/reccomendations';
```

Figure 75: App Module Page Imports

```
@NgModule({
  declarations: [
    MyApp,
    HomePage,
    TabsPage,
    LoginPage,
    PlayerPage,
    LogoutPage,
    SearchPage,
    ExplorePage,
    ArtistSelectionPage,
    ReccomendationsPage
  ],
```

Figure 76: App Module Declarations

10.3.2 Setting Up The Pages

Once the pages have been created and added to the app.module.ts file, they can be called from any other page within the application as long as it is imported. When creating a tab bar application, Ionic provides a "tabs" folder within the "pages" folder. This is where the tab bar is defined and can be edited to include other pages *Ionic, 2018.* To add the pages required by the author, the pages are first imported to the tabs.ts file, as seen in *figure 77.*

```
import { SearchPage } fro module "/Users/Jake/Deskt
import { ExplorePage } fr ome/home"
import { HomePage } from '../home/home';
```

Figure 77: Importing Required Tab Pages

Variables are then associated with the page names, so they can be called within the HTML, as seen in *figure 78.*

```
export class TabsPage {

  tab1Root = HomePage;
  tab2Root = ExplorePage;
  tab3Root = SearchPage;

  constructor() {
```

Figure 78: Associating Variables To Each Page

The pages are then called within the HTML, as seen in *figure 79.* Small icons that are present within the Ionic framework library, are also added using the "tabIcon" code.

```
<ion-tabs>
  <ion-tab [root]="tab1Root" tabTitle="Home" tabIcon="home"></ion-tab>
  <ion-tab [root]="tab2Root" tabTitle="Explore" tabIcon="color-wand"></ion-tab>
  <ion-tab [root]="tab3Root" tabTitle="Search" tabIcon="search"></ion-tab>
</ion-tabs>
```

Figure 79: Calling Page Within HTML

Figure 80 shows the result of the previous steps, within the browser.

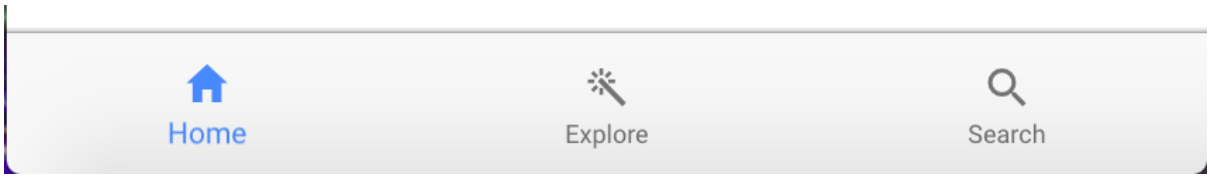

Figure 80: Complete Tab Bar In Browser

Each page can now be modified, and the changes will be presented within the corresponding tab.

10.4 Connecting To The Database

To connect to the database, the Angular Fire plugin is required. This is installed in a similar way to Ng2 Admin.

10.4.1 Installing Angular Fire

To install Angular Fire, it must be added to the package.json file, which can be found in the root directory of the application, as seen in *figure 81.*

```
"angularfire2": "^4.0.0-rc.0",
"angularfire2-offline": "^4.0.1",
"base64-img": "^1.0.3",
"cordova-android": "^6.2.3",
```

Figure 81: Angular Fire Installation

The next time the “ionic serve” command is executed in the command line, the developer will be prompted to install the corresponding module.

Just like with Ng2 Admin, the Angular Fire plugin needs to be imported and declared within the app.module.ts file, for it to be accessible throughout the application, *Simon, 2017*. *Figure 82 and 83* shows this.

```
//Angular Fire Stuff
import { AngularFireModule } from 'angularfire2';
import { AngularFireDatabaseModule, AngularFireDatabase, FirebaseListObservable} from 'angularfire2/database';
import { AngularFireAuthModule, AngularFireAuth } from 'angularfire2/auth';
import { TranslateHttpLoader } from '@ngx-translate/http-loader';
import { TranslateModule, TranslateLoader } from '@ngx-translate/core';
```

Figure 82: Importing Angular Fire

```
imports: [
  HttpModule,
  IonicStorageModule.forRoot(),
  TranslateModule.forRoot({
    loader: {
      provide: TranslateLoader,
      useFactory: (createTranslateLoader),
      deps: [Http]
    }
  }),
  SharedModule.forRoot(),
  BrowserModule,
  IonicModule.forRoot(MyApp,{
    backButtonText: '',
  }),
  IonicAudioModule.forRoot(),
  AngularFireModule.initializeApp(firebaseConfig),
  AngularFireDatabaseModule,
  AngularFireAuthModule,
```

Figure 83: Declaring Angular Fire

10.4.2 Adding The Firebase Credentials

The Firebase credentials also need to be added to the app.module.ts for it to successfully connect to the correct database. *Figure 84* shows this.

```
//Firebase database login stuff
export const firebaseConfig = {
  apiKey: "AIzaSyDIY9yTDB2mVrryd6GtPmSJPR8qd1PmlVc",
    authDomain: "finalyearproject-576e5.firebaseapp.com",
    databaseURL: "https://finalyearproject-576e5.firebaseio.com",
    projectId: "finalyearproject-576e5",
    storageBucket: "finalyearproject-576e5.appspot.com",
    messagingSenderId: "80418793901"
};
```

Figure 84: Firebase Database Credentials

10.5 Creating The Login Page

As the login page has already been created, a similar approach to the web application needs to be taken, whereby the login page is displayed first before anything else. It also needs to include all the necessary input items to allow users to enter an email address and password, as well as create an account. It must also not allow users that are not logged in, to gain access to the application.

10.5.1 Designing The Login Page

To design the login page, the HTML file will need to be modified. This can be found by heading to src/ pages/ login/ login.html. As described in the design section, the login page requires two input fields for the email address and password as well as two buttons. This can be seen in the following *figures.* Some styling has also been added to the HTML elements.

```
<ion-item>
  <ion-label color="primary" floating>Email</ion-label>
  <ion-input type="text"></ion-input>
 </ion-item>

 <ion-item>
  <ion-label color="primary" floating>Password</ion-label>
  <ion-input type="password"></ion-input>
 </ion-item>
```

Figure 85: Login Page HTML Input Fields

```
<ion-col col-4>
<button ion-button style="opacity: 0.8;box-shadow: 0px 10px 30px #151515;border-radius: 20px" color="white" block >Login</button>
</ion-col>

<ion-col col-4>
  <button ion-button style="opacity: 0.8;box-shadow: 0px 10px 30px #151515;border-radius: 20px" color="light" block icon-left >Sign Up</b
</ion-col>
/ion-row>
```

Figure 86: Login Page HTML Buttons

As the login page also requires the user to be able to create an account, the HTML hidden feature is used, which allows the elements to be toggled within the JavaScript/ Typescript, *W3Schools, 2018.* This allows the elements on the page to be dynamic, meaning the entire page does not have to change, to display different elements. *Figure 87* shows this within the HTML.

```
<ion-row style="height: 46%;text-align: center" >
  <ion-row  [hidden]="registerPage" style="width: 100%">
    <ion-col col-2>
    </ion-col>
```

Figure 87: HTML Hidden Attribute

The following HTML in *figure 88 and 89* is then used when the HTML element is toggled. This creates two separate input fields and buttons as well as change the text of the buttons. This is later used within the JavaScript/ Typescript.

```
  <ion-row  [hidden]="!registerPage" style="width: 100%">
    <ion-col col-2>
    </ion-col>
    <ion-col col-8 style="text-align: center;border-radius: 3px;">
  <ion-list no-lines>
    <ion-item>
      <ion-label color="primary" floating>Email</ion-label>
      <ion-input type="text"></ion-input>
    </ion-item>

    <ion-item>
      <ion-label color="primary" floating>Password</ion-label>
      <ion-input type="password"></ion-input>
    </ion-item>
  </ion-list>
```

Figure 88: Hidden HTML Fields

```
<ion-col col-4>
<button ion-button style="opacity: 0.8;box-shadow: 0px 10px 30px #151515;border-radius: 20px" color="white" block >Sign Up</button>
</ion-col>

<ion-col col-4>
<button ion-button style="opacity: 0.8;box-shadow: 0px 10px 30px #151515;border-radius: 20px" color="light" block icon-left >Back</button>
</ion-col>
```

Figure 89: Hidden HTML Buttons

This produces the following result in the browser.

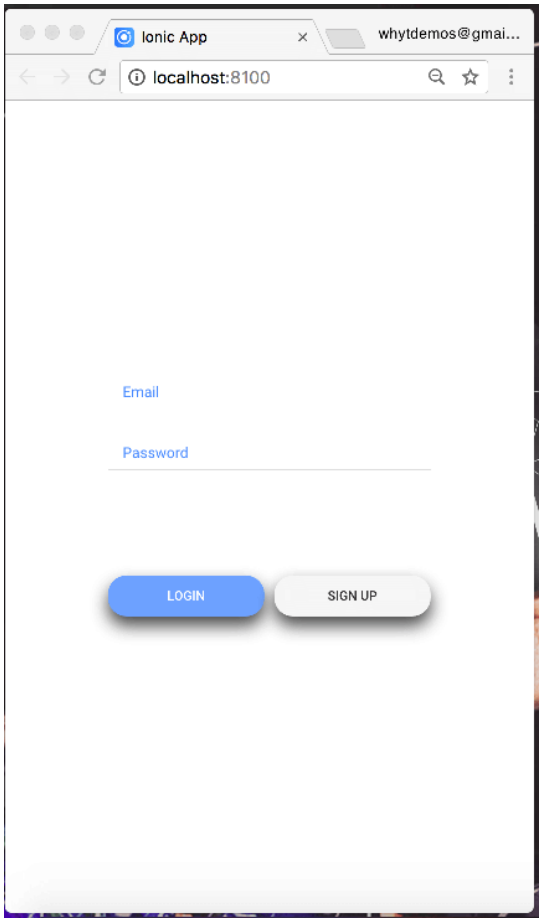

Figure 90: Login Screen Simulated Within The Browser

10.5.2 Logging In To The Application

In order to connect to the database, the Angular Fire plugin must be imported to the file. The code in *figure 91* is then inserted into the constructor within the login.ts file. This code checks to see whether the user is already logged in. If so, the user is presented with the "TabsPage" which takes the user to the main application, *Stavros Kounis, 2018.*

```
        this.afService.afAuth.authState.subscribe((user: firebase.User) => {
      if (!user) {

      }
      else{
        this.navCtrl.setRoot(TabsPage);

      }

    });
```

Figure 91: Check If User Is Already Logged In

If the user is not logged in they will need to do so. First a variable is created to hold the email address and password combination from the HTML input, which can be seen in *figure 92.*

```
this.loginData={email:"",password:""};
```

Figure 92: Email Address And Password Variable.

A function called “signIn” is then created which can then be assigned to a button. The code used within the function takes the data stored in the variable, created in the previous step, and submits it to the database. As the Angular Fire plugin does a lot of the heavy lifting, a simple .catch method can be used to identify any errors that may occur when submitting the account data to the database. In the authors case, if the email address or password combination is wrong, an error alert is displayed, which can be seen in *figure, 93.*

```
78
79    signIn(): void {
80      this.afService.signIn(this.loginData.email,this.loginData.password)
81        .then(x => {
82          this.storage.set('email', this.loginData.email);
83          this.storage.set('password', this.loginData.password);
84
85        })
86        .catch((error) => {
87      console.log(error);
88            let alert = this.alertCtrl.create({
89      title: 'Login',
90      subTitle: 'Email Address or Password Incorrect',
91      buttons: ['Dismiss']
92    });
93    alert.present();
94
95      });
96    }
```

Figure 93: Sign In Function

Now the functions need to be assigned to the HTML elements. This is done by heading to the login page HTML file, which is located within the login page folder.

In order for the input within the email and password fields to be associated with the variable within the JavaScript/ Typescript, the ngModel attribute is added to the input fields. NgModel binds to formcontrol instances and allows the variable to be manipulated within the JavaScript/ Typescript, *Angular, 2018.*

```
  <ion-label color="primary" floating>Email</ion-label>
  <ion-input type="text" [(ngModel)]="loginData.email"></ion-input>
</ion-item>

 <ion-item>
  <ion-label color="primary" floating>Password</ion-label>
  <ion-input type="password" [(ngModel)]="loginData.password"></ion-input>
```

Figure 94: NgModel Added To Input Fields

The click attribute is then added to the login button, which will execute the signIn function when the button is clicked. This can be seen in *figure 95.*

```
<ion-col col-4>
<button (click)="signIn()" ion-button style="opacity: 0.8;box-shadow: 0px 10px 30px #151515;border-radius: 20px" color="white" block >Login<
</ion-col>
```

Figure 95: SignIn Attribute Added To Button Click

10.5.3 Signing Up Within The Application

As the author requires the HTML elements to be toggled when signing up or logging in, the HTML hidden attribute is assigned to the two sections of HTML as explained previously.

To achieve this a Boolean variable first needs to be defined within the JavaScript/ Typescript. This can be seen in *figure 96.*

```
registerPage: boolean=false;
```

Figure 96: Boolean Declaration

Once the Boolean is defined, a function called “page” is created with the following code found in *figure 97.* This piece of code changes the state of the Boolean to the opposite of what it is, when the code is executed.

```
page(){
  this.registerPage = !this.registerPage;
}
```

Figure 97: Page Function

The page function is then assigned to a button within the HTML, which will allow the Boolean to be toggled. This can be seen in *figure 98.* In the authors case, this is assigned to the sign-up button.

```
<button (click)="page()" *ngIf="!registerPage"
ion-button style="opacity: 0.8;box-shadow: 0px 10px 30px #151515;border-radius: 20px"
color="light" block icon-left >Sign Up</button>
```

Figure 98: Page Function Assigned To Button

NgIf is also added to the button to check the state of the Boolean within the Javascript/ Typescript, which will in turn display or hide each section of HTML, depending on the state of the Boolean. This is also added to the back button, which is only displayed when the sign-up button is clicked, *figure 99*, in turn reversing the state of the Boolean, *W3Schools, 2018. Figure 100 and 101* show this visually.

```
<ion-col col-4>
<button (click)="page()" *ngIf="registerPage"
ion-button style="opacity: 0.8;box-shadow: 0px 10px 30px #151515;border-radius: 20px"
color="light" block icon-left >Back</button>
</ion-col>
```

Figure 99: Back Button NgIf

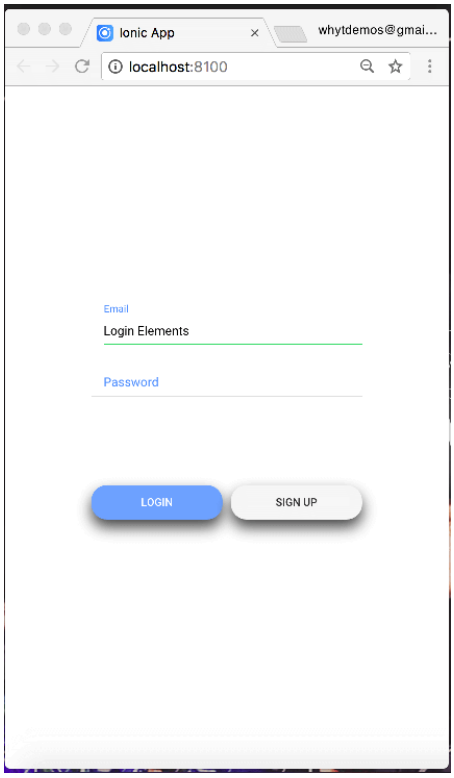

Figure 100: Login Elements

When the sign-up button is clicked, the following elements are displayed.

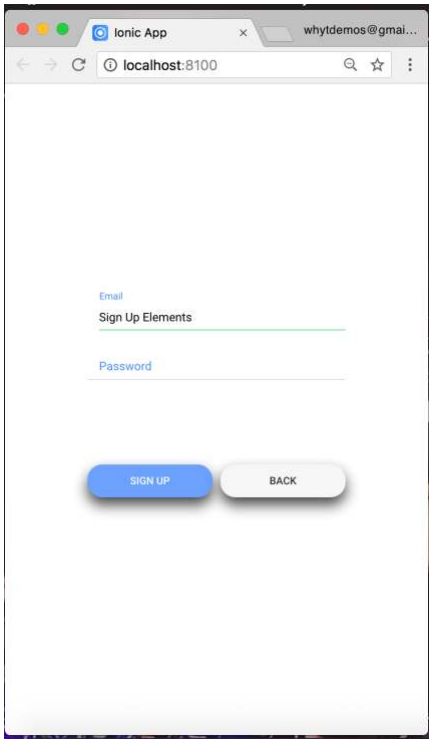

Figure 101: Sign Up Elements

Next is to add functionality to the sign-up button, to allow the application to submit the information inputted by the user, and create a new account on the database. This is done by creating another function within the JavaScript/ Typescript. *Figure 102* shows the "register" function.

```
register(): void {

  this.afService.registerUser(this.registerData.email, this.registerData.password,
  .then(x => {
      this.storage.set('email', this.registerData.email);
      this.storage.set('password', this.registerData.password);

  })
.catch((error) => {

      let alert = this.alertCtrl.create({
  title: 'Login',
  subTitle: 'Email Address is not valid or already in use',
  buttons: ['Dismiss']
});
alert.present();
```

Figure 102: Register Function

The code used within the function is very similar to the login code, except for it calls upon the "registerUser" function within the Angular Fire plugin. Again, most of the heavily lifting is performed by the plugin which allows any errors to be caught and displayed as an alert to the user.

This function is then assigned to the sign-up button, which in turn allows user accounts to be created from within the application. *Figure 103* shows the register function assigned to the sign-up button.

```
<ion-col col-4>
<button (click)="register()"
ion-button style="opacity: 0.8;box-shadow: 0px 10px 30px #151515;border-radius: 20px"
color="white" block >Sign Up</button>
</ion-col>
```

Figure 103: Sign Up Button With Register Function

Figure 104 shows the user account details entered the register elements.

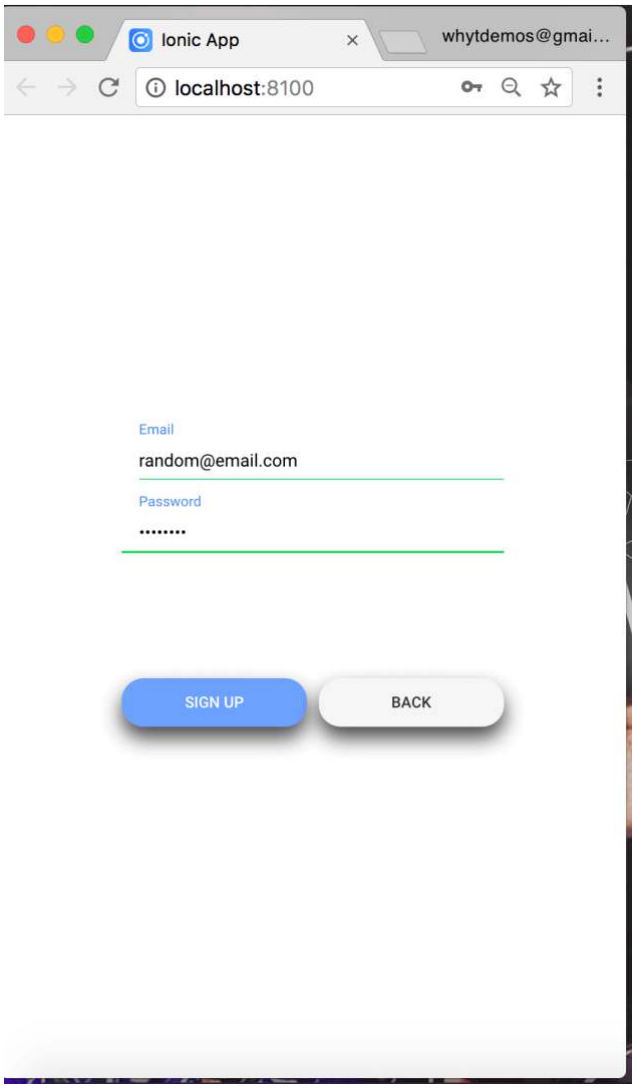

Figure 104: Creating An Account

Figure 105 shows the user account details within the Firebase console.

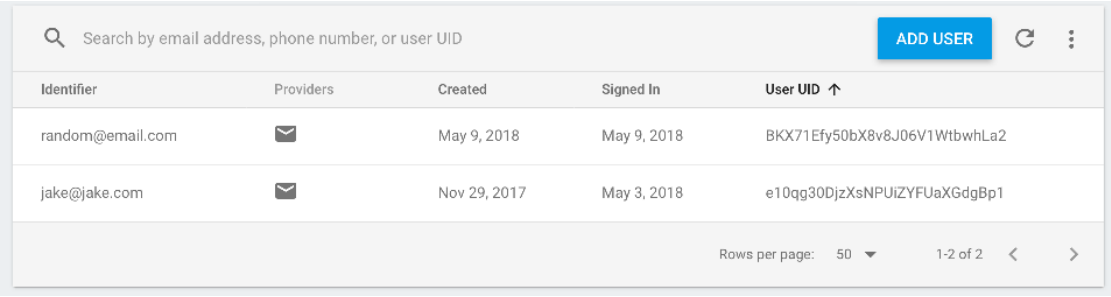

Figure 105: Submitted Credentials

10.5.4 Displaying The Login Page

As the user is required to have an account before they can use the application, the first page that is displayed, must be the login page. Just like with the web application, when the app is first opened, a few files and lines of code are executed to ensure the application is ready for use. This is achieved by adding redirecting code within the constructor of src/ app/ app.component.ts. *Figure 106* shows the code necessary to achieve this.

```
    this.afService.afAuth.authState.subscribe((user: firebase.User) => {
      if (!user) {
this.isLoggedIn = false;
        this.rootPage=LoginPage;
      }
      else{

this.isLoggedIn = true;
        this.rootPage=TabsPage;
      }

    });
```

Figure 106: Redirecting Code

The code above checks to see whether the user is already logged in or has previously logged in, by calling the “authState” from the Angular Fire plugin. If true, the user is taken to the main application. If false, the user is taken to the login page. As the app.component.ts is run every time the application is opened, the check will always be performed first, therefore only allowing users with an account into the application.

10.6 Creating The Home Page

As described in *section 8.5.3* the home page is to consist of all the available tracks that are on the database. The album artwork is to be used as the scrollable objects, which when clicked/ tapped, play the corresponding track. Playing the track will be explained in the player section.

10.6.1 Designing The Home Page

The home page is required to have scrollable album art as the main page, as well as a log out button in the top right-hand corner. *Figure 107* shows the html that is required to achieve the scrollable album artwork. This will not display the artwork, but creates an object for the artwork to sit in.

```
<ion-content padding>
    <ion-scroll scrollY style="height:100%;">

        <div class="scroll-item" style="text-align: center;margin-left:32%;width: 120px;">
          <img class="shadow"  style="width:100%;height: 100%">
          <p style="text-overflow: ellipsis;overflow: hidden"></p>
          <p style="margin-top:-10px;font-size: 80%"></p>
          <p> </p>
        </div>

    </ion-scroll>

</ion-content>
```

Figure 107: Scrollable Album Artwork Placement

Figure 108 shows the HTML code required to add a button to the navigation bar in the top right. A small icon is also added to the button, which is part of the Ionic framework, using the ion-icon attribute.

```
    <ion-title center>Home</ion-title>
    <ion-buttons end>
        <button ion-button icon-only>
          <ion-icon name="menu"></ion-icon>
        </button>
    </ion-buttons>
  </ion-navbar>
</ion-header>
```

Figure 108: Logout Navigation Button

10.6.2 Creating Scrollable Artwork

For the list within the database to be displayed, a FirebaseListObservable must be used. The FirebaseListObservable is then given a directory to monitor, in this case the “tracks” directory. The directory is then subscribed to, and every object within the directory is pushed into an array, along with all its attributes that were specified during the upload, *Jave Bratt, 2016.* The following *figures* show this within the src/ pages/ home/ home.ts file.

```
theTracks:FirebaseListObservable<any>;
```

Figure 109: FirebaseListObservable

```
this.theTracks=this.md.tracks;
```

Figure 110: Defining Directory

```
this.theTracks.subscribe(pop=>{
  if(this.trackb==false){
      this.theTracksPlaylists=[];
        pop.forEach(pope => {
          this.theTracks.subscribe(all=>{
          all.forEach(alle=>{
            if(alle.$key==pope.$key){
```

Figure 111: Subscribing To The Directory

```
this.theTracksPlaylists.push(

      {src: alle.url,
      artist: alle.artistName,
      title: alle.name,
      art: alle.albumArt,
      preload: 'metadata',
      key:alle.$key,
      artistId:alle.artist,
      albumId:alle.album,
      album:alle.albumName,
      artistName:alle.artistName,
      }
  );
```

Figure 112: Pushing The Objects From The Directory Into An Array

Once the objects/ tracks are in an array, they can be used within the HTML. This is done by using an ngFor* loop to cycle through the array, *Angular, 2018. Figure 113* shows the ngFor* loop required to display the tracks list.

```
<div class="scroll-item" *ngFor="let artist of theTracksPlaylists"
```

Figure 113: ngFor Loop HTML*

To then display the album artwork of each track within the array, a small tag needs to be added to the image source. In this case it is {{artist.art}}, *Angular, 2018*. This will in turn retrieve the corresponding image for the track, within the array and display it. *Figure 114* shows this.

```
<div class="scroll-item" *ngFor="let artist of theTracksPlaylists" style="te
  <img class="shadow" src="{{artist.art}}" style="width:100%;height: 100%">
```

Figure 114: Image Tag

This produces the result found in *figure 115.*

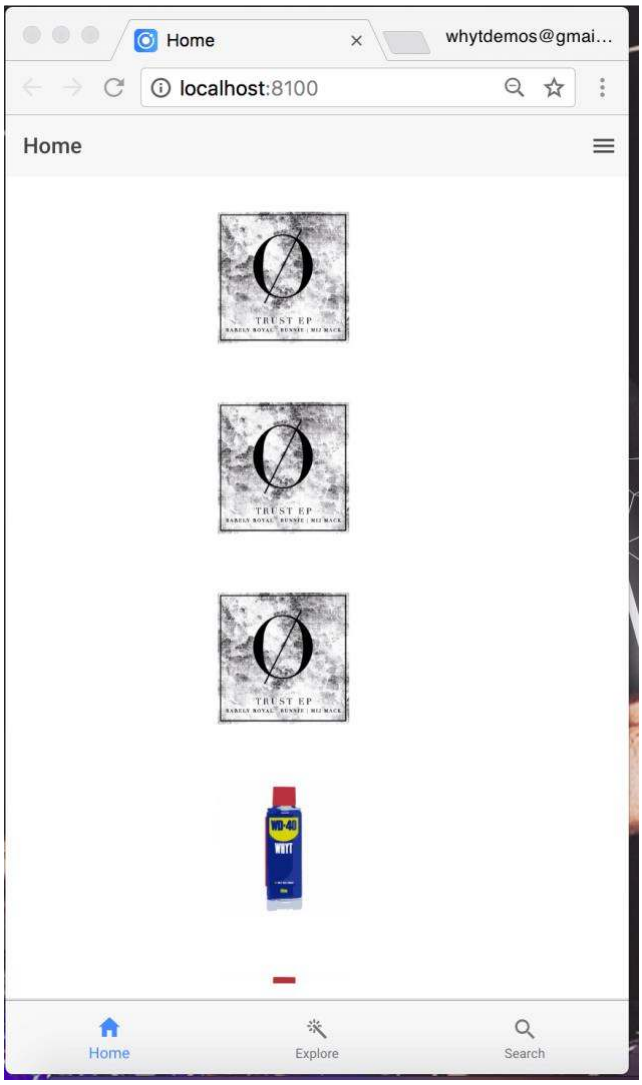

Figure 115: Home Page With Scrollable Images

The following tags are added to the paragraphs within the HTML to add the track name and artist name.

```
<img class="shadow" src="{{artist.art}}" style="width:100%;height: 100%">
<p style="text-overflow: ellipsis;overflow: hidden">{{artist.title}}</p>
<p style="margin-top:-10px;font-size: 80%">{{artist.artistName}}</p>
```

Figure 116: Artist And Track Text

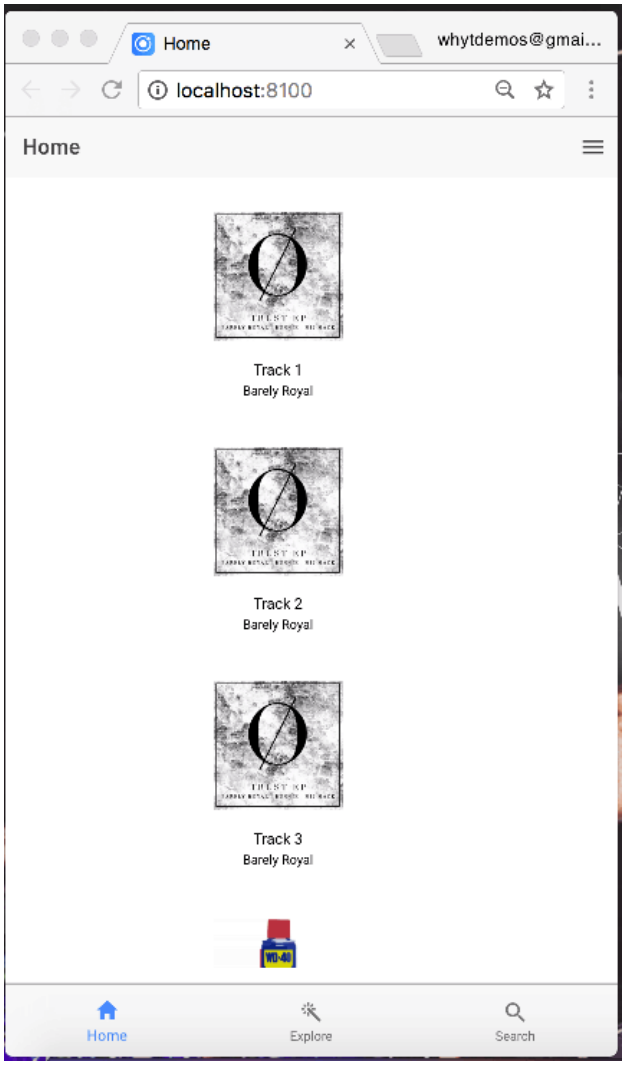

Figure 117: Complete Home Page

10.6.3 Adding Logout Functionality

As stated in the designs, the button in the top right-hand corner must display another page, and allow the user to logout of the application. To display the logout page, it must first be imported along with a navigation controller supplied within Ionic, shown in in f*igure 118 and 119.*

```
import{ LogoutPage } from '../logout/logout';
```

Figure 118: Importing Page

```
import { NavController, Platform } from 'ionic-angular';
```

Figure 119: Importing Navigation Controller

Using the navigation controller push attribute, the page can then be displayed, *Ionic, 2018*. *Figure 120* shows this added to the function called “logout”.

```
logout(){

  this.navCtrl.push(LogoutPage);

}
```

Figure 120: Navigation Controller Push Attribute

The logout function can then be assigned to the logout button within the HTML as seen in *figure 121.*

```
<ion-buttons end>
    <button (click)="logout()" ion-button icon-only >
      <ion-icon name="menu"></ion-icon>
```

Figure 121: Logout Button Function Assignment

The produces the following result when the button in the top right-hand corner of the navigation bar is clicked.

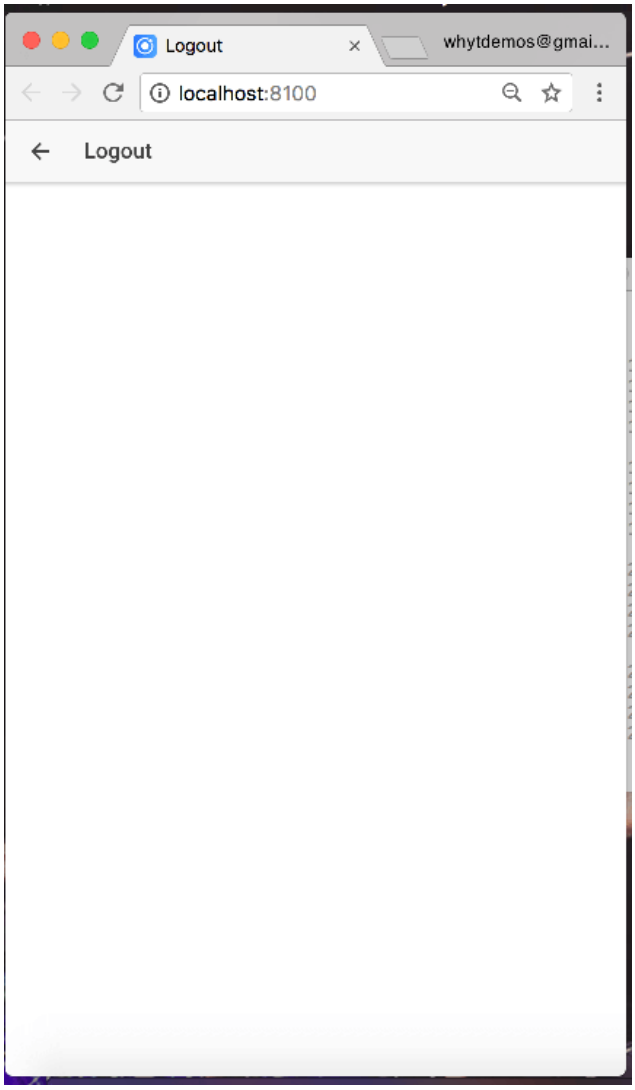

Figure 121: Logout Page Button Clicked

Now all that is left is to create a button on the logout page that executes the logout code. Heading to src/ pages/ logout.ts, the "logout" function can be created. *Figure 122* shows the logout function.

```
logout(){

  setTimeout(() => {
  this._auth.signOut();
  }, 500);
```

Figure 122: Logout Function

The function can then be added to a button within logout page HTML, as seen in *figure 123.*

```
<ion-list>
  <ion-item (click)="logout()" style="background-color: transparent">
    <h2>Logout</h2>
  </ion-item>
</ion-list>
```

Figure 123: Logout Page HTML

Which produces the following result.

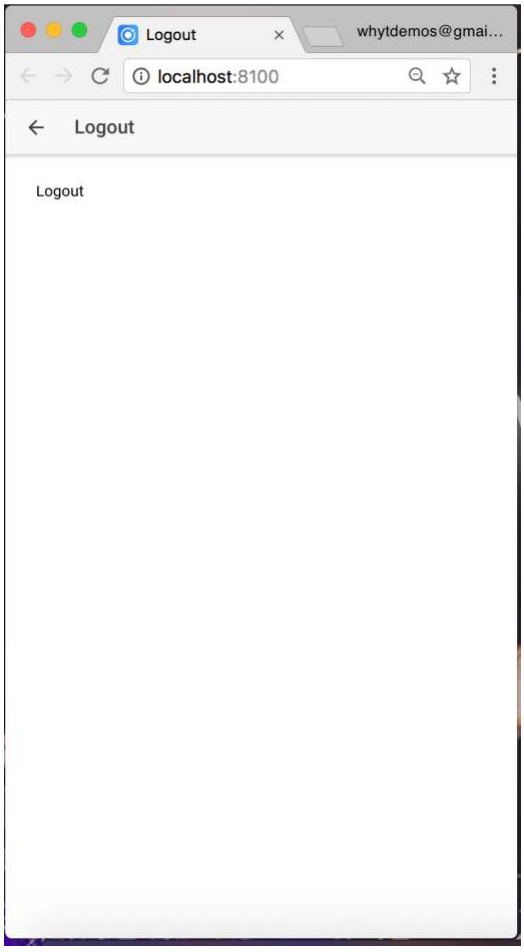

Figure 124: Complete Logout Page

10.7 Creating The Search Page

The search page must be capable of searching the entire database, and displaying tracks that match the search criteria.

10.7.1 Designing The Search Page

The Ionic framework includes a natively designed search bar, which is going to be used within the search page. This can be used by adding the ion-searchbar tag within the HTML, as seen in *figure 125,* which produces the result shown in *figure 126.*

```
    <ion-navbar>
        <ion-searchbar></ion-searchbar>
    </ion-navbar>

</ion-header>
```

Figure 125: HTML Search Bar

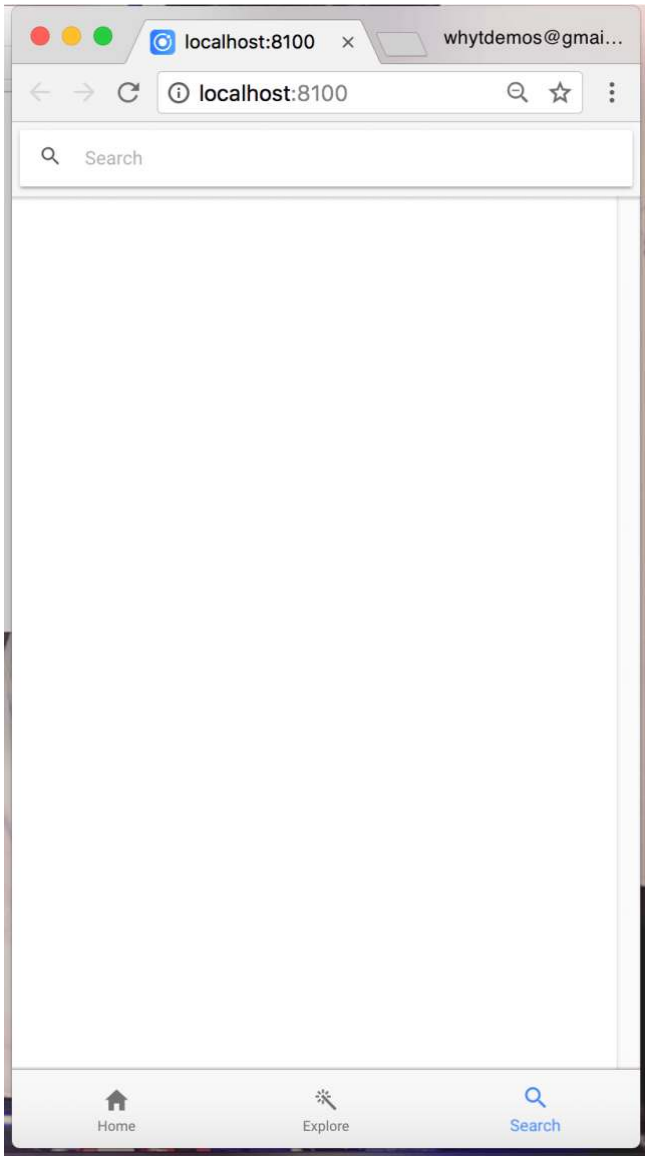

Figure 126: Search Bar Within Application

The Ionic list item is then used, to position a list underneath the search bar, which will contain the results from the search criteria.

```
<ion-list [hidden]="searchTracks.length==0" no-lines >
<ion-item  block clear    style="background-color: transparent" >
    <ion-avatar item-left>
        <img class="shadow">
    </ion-avatar>
    <h2>
        <ion-icon name="play" style="opacity: 0.5">  </ion-icon>
    </h2>
    <p></p>

  </ion-item>
</ion-list>

/ion-content>
```

Figure 127: Search List HTML

10.7.2 Creating Search Algorithm

Just like in the creation of the home page, the tracks will need to be stored in an array which can then be displayed in the HTML. However, the items that are pushed into the array must match the criteria that the user types into the search bar. A function is first created which is then assigned to the search bar. The text is then taken from the search bar and inserted into a variable, which is then passed to an algorithm from, *Thomas2017, 2016*, which can be seen in *figure 128*.

The algorithm narrows down the list of tracks within the database, and pushes the result into the array, along with their attribute.

```
if(( item.name.toLowerCase().indexOf(val.toLowerCase()) > -1) ||
( item.artistName.toLowerCase().indexOf(val.toLowerCase()) > -1))
    {
    this.result=true;

    this.searchTracks.push(item);
    this.playlists.push(

        {src: item.url,
        video: item.video,
        artist: item.artistName,
        title: item.name,
        art: item.albumArt,
        preload: 'metadata',
        key:item.$key,
        artistId:item.artist,
        albumId:item.album,
        album:item.albumName
```

Figure 128: Search Algorithm

10.7.3 Displaying Search Data

As the HTML is already set up to display the data, the functionality just needs to be added. As explained earlier, the text inputted to the search bar needs to be passed to the algorithm for processing. This is done by adding an “ionInput” tag to the search bar, along with the name of the function created in the previous step. This can be seen in *figure 129.*

```
<ion-navbar>
    <ion-searchbar (ionInput)="search($event)" style="background-color: transparent"></ion-searchbar>
</ion-navbar>
```

Figure 129: Search Bar HTML

Just like on the home page, an ngFor* tag is used to display the results from the array, in an ion-list item, which in turn displays the artist name, track name and album artwork.

```
<ion-item *ngFor="let artist of searchTracks" block clear    style="background-color: transparent" >
    <ion-avatar>
        <img class="shadow" [src]="artist.albumArt">
    </ion-avatar>
    <h2>
      <ion-icon name="play" style="opacity: 0.5">  </ion-icon>
      {{artist.name}}
    </h2>
    <p>{{artist.artistName}}</p>

  </ion-item>
</ion-list>
```

Figure 130: Search List HTML

Figure 131 shows this.

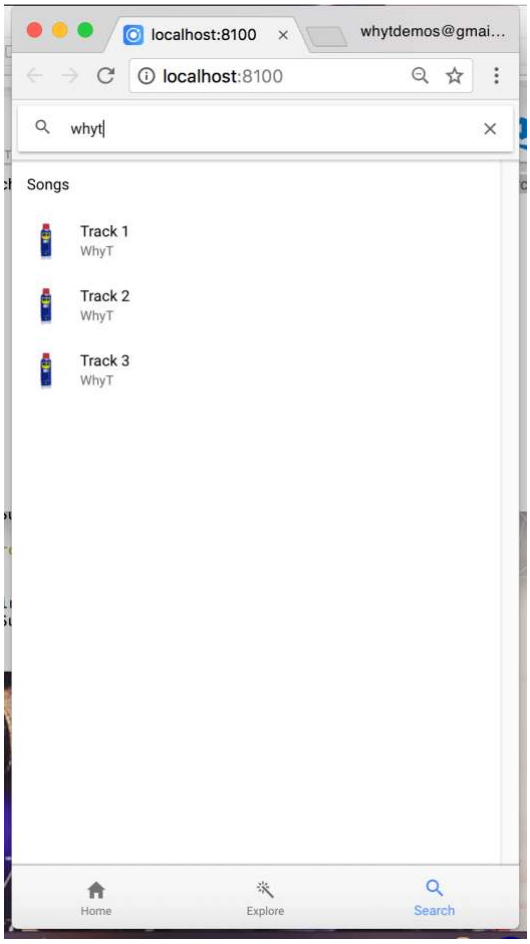

Figure 131: Complete Search Page

10.8 Creating The Explore Page

The purpose of the explore page is to allow users to find new music based on their music tastes from an external source. This was originally going to using the Spotify API, which would allow users to login to their Spotify account, and select some artists that they like. The Spotify API would then return a JSON list of tracks, that are similar in style to the artists previously chosen. The dancability, energy and tempo values would then be extracted from tracks in the returned list, and a query would be made to the Firebase database. However, due to issue with authenticating with the Spotify API, this was not possible.

In turn, the following section will be using local JSON files instead of querying the Spotify API. However, as these are JSON files, there is no reason why this could not be used with an external source, such as Spotify or YouTube in the future.

10.8.1 Designing The Explore Page

The explore page is required to display tracks from an external source and display them in a list. The tracks in the list must be able to be played, paused and must also contain buttons for the user to like or dislike them.

Therefore, an Ion-card is used within the HTML, which is used as the container for each of the tracks. The dancability, energy and tempo values have also been added to the cards. This can be seen in *figure 132.*

```
<ion-card>
    <img src="">
    <ion-card-content>
      <h2 class="card-title">
      </h2>
      <h3>
        Dancability
      </h3>
      <h3>
        Energy
      </h3>
      <h3>
        Tempo
      </h3>
    </ion-card-content>
  <ion-row>
    <ion-col width-50>
      <ion-item class="alignText">
        <ion-icon class="likeButtons" name="close"></ion-icon>
      </ion-item>
    </ion-col>
    <ion-col width-50>
      <ion-item class="alignText">
        <ion-icon class="likeButtons" name="heart"></ion-icon>
      </ion-item>
    </ion-col>
  </ion-row>
      <p> </p>
      <ion-item >
        <ion-icon class="otherButtons" name='play' item-start></ion-icon>
        Preview
      </ion-item>
      <ion-item>
        <ion-icon class="otherButtons" name='pause' item-start></ion-icon>
        Pause
      </ion-item>
```

Figure 132: Explore Page HTML Card Design

As no data has been defined, the cards are currently empty. However, the following result is produced.

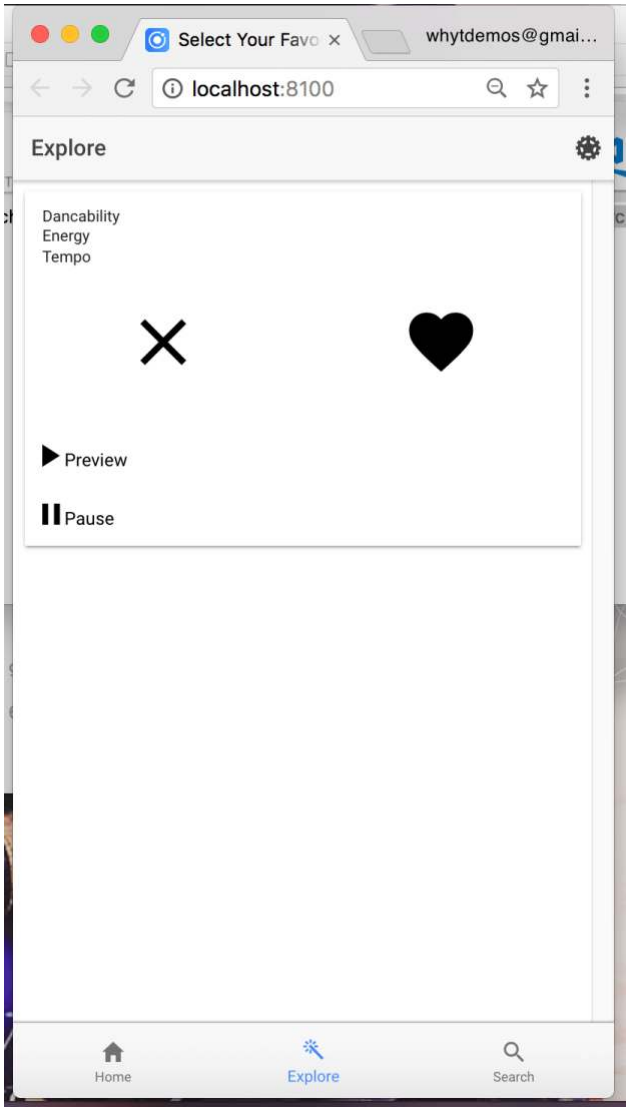

Figure 133: Explore Page Running In Browser

10.8.2 Loading Local JSON

First, the local JSON file needs to be created. JSON is a simple format and can be created in any text *editor*. Following *W3Schools, 2018* as a guide, the following JSON file is created. This contains the artist name, the track name, the energy, the dancability, the tempo, the path to a locally stored image and the path to a locally stored audio file.

```
{
    "swipableTracks": [
        {
            "artistName": "Artist 1",
            "trackName": "Track 1",
            "bpm": "140",
            "dancibility": "0.1",
            "energy": "0.1",
            "img": "./assets/data/WhyTProfilePic.png",
            "audio": "./assets/data/music.mp3"
        },
        {
```

Figure 134: Local JSON File

Just like before, to display data in a list, the data must first be inserted in an array. Using a dataFinder framework from *Chris Tutorials, 2017,* the local JSON file can be loaded with ease within the src/ pages/ explore/ explore.ts file.

```
this.dataFinder.getJSONDataAsync("./assets/data/swipableTracks.json").then(data => {
  this.SetQueryOptionsData(data);
});
```

Figure 135: Data Finder Framework

The dataFinder framework is then used to call upon another function, which then sends the data in the JSON file into an array. As seen in *figure 136.*

```
/* Sets data with returned JSON array */
SetQueryOptionsData(data : any) {
  this.swipableTracks = data.swipableTracks;
}
```

Figure 136: Data Finder Calling Set Query Options Data Function

Now that the data is in the array, the HTML can be modified accordingly to display the data, using an ngFor* and the corresponding tags. *Figure 137* shows the HTML.

```
<ion-card *ngFor="let swipableTrack of swipableTracks; let i=index;" class="item-remove-animate">
    <img src={{swipableTrack.img}}>
    <ion-card-content>
      <h2 class="card-title">
        {{swipableTrack.artistName}} - {{swipableTrack.trackName}}
      </h2>
      <h3>
        Dancability - {{swipableTrack.dancibility}}
      </h3>
      <h3>
        Energy - {{swipableTrack.energy}}
      </h3>
      <h3>
        Tempo - {{swipableTrack.bpm}}
      </h3>
    </ion-card-content>
<ion-row>
```

Figure 137: Explore Page HTML With Array Display Tags

This produces the following result.

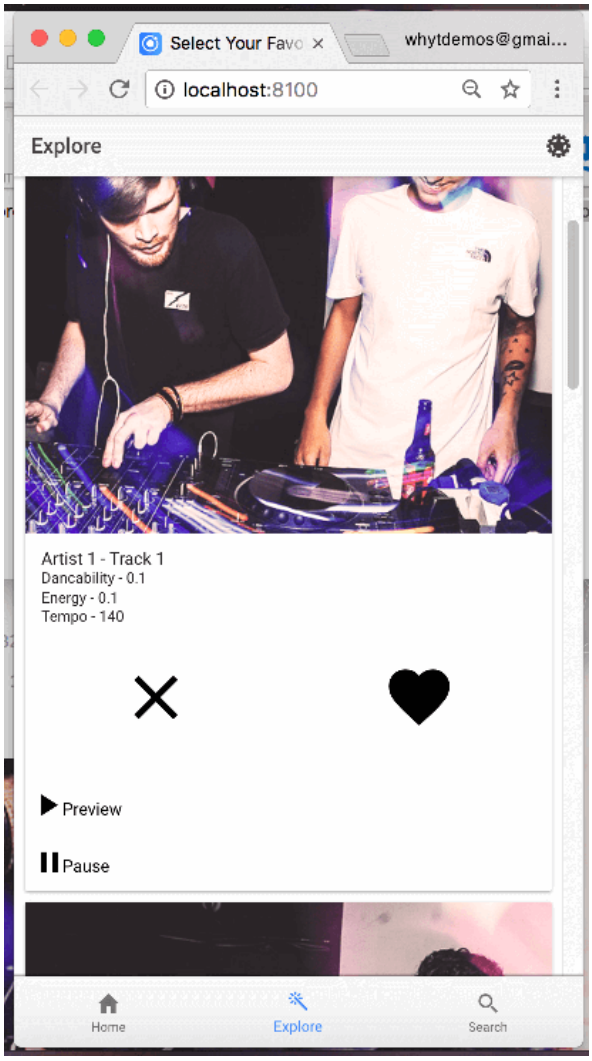

Figure 138: Explore Page In Browser

10.8.3 Playing And Pausing Local JSON Tracks

As the play and pause buttons have already been added to the cards, functionality just needs to be added to them. Within the explore.ts file a function called “playAudio” and “pauseAudio” are created. These functions utilise the native web audio features by calling webMedia, that was found with the help of *Alexintosh, 2017.*

Figure 139 shows the play function within the explore.ts file.

```
playAudio(audioLocation){

  const onStatusUpdate = (status) => console.log(status)

  if(this.webmedia) {
    this.webmedia.pause();
  }

  this.webmedia = new Audio(audioLocation);
  this.webmedia.load();
  this.webmedia.play();
}
```

Figure 139: Play Audio Function

The above code gets the audio location from the HTML element, and passes it to the play audio function. The function then checks if there is a track currently playing in webMedia. If there is, the current audio is paused, and the new audio file is then loaded into webMedia, and playback begins. *Figure 140* shows the modification made to the HTML element to utilise the function.

```
<ion-item (click)="playAudio(swipableTrack.audio)">
  <ion-icon class="otherButtons" name='play' item-start></ion-icon>
  Preview
```

Figure 140: Play Button With Function Attached

The same process is taken for the pause button, however only a small piece of code is required to pause the currently playing audio. The following figures show the function and the modified HTML.

```
pauseAudio(getIcon: string) {
  this.webmedia.pause();
  }
```

Figure 141: Pause Audio Function

```
<ion-item (click)="pauseAudio(swipableTrack.audio)">
  <ion-icon class="otherButtons" name='pause' item-start></ion-icon>
  Pause
</ion-item>
```

Figure 142: Pause Button With Function Attached

10.8.4 Processing Track Features

To recommend new tracks to users, the application needs to process the dancability, energy and tempo of each of the tracks. The author has decided that a good way to do this is to average the values of the tracks, from the local JSON. Averaging of the tracks values will only be done when a user likes a track. If the user dislikes a track, no action is taken.

Figure 143 shows averaging algorithm that is used within the explore.ts file within the "energyClick" function. The algorithm was found with the help of *Thomasnu, 2014.*

```
energyClick(parsedEnergy){
  this.energyArray.push(parsedEnergy);
  var sum = 0;
  for(var i = 0; i < this.energyArray.length; i++){
    sum += parseFloat(this.energyArray[i]);
  }
  this.averageEnergy = sum/this.energyArray.length;
  console.log("Avg energy: " + this.averageEnergy);
}
```

Figure 143: Energy Averaging Algorithm

The above code takes the energy value from the HTML element, and pushes the value into an array. The value is then averaged using the length of the array. The calculated value is then stored in a variable. This is also done for the dancability and tempo.

```
danceClick(parsedDance) {
  this.danceArray.push(parsedDance);
  var sum2 = 0;
  for(var j = 0; j < this.danceArray.length; j++){
    sum2 += parseFloat(this.danceArray[j]);
  }
  this.averageDance = sum2/this.danceArray.length;
  console.log("Avg dancability: " + this.averageDance);
}
```

Figure 144: Dancability Averaging Algorithm

```
tempoClick(parsedTempo) {
  this.tempoArray.push(parsedTempo);
  var sum3 = 0;
  for(var x = 0; x < this.tempoArray.length; x++){
    sum3 += parseInt(this.tempoArray[x],10);
  }
  this.averageTempo = sum3/this.tempoArray.length;
  console.log("Avg tempo: " + this.averageTempo);
```

Figure 145: Tempo Averaging Algorithm

The HTML element is then modified to call the created functions when the like button is pressed.

```
<ion-col width-50>
  <ion-item class="alignText"(click)="energyClick(swipableTrack.energy)" (click)="danceClick(swipableTrack.dancibility)"
  (click)="tempoClick(swipableTrack.bpm)">
    <ion-icon class="likeButtons" name="heart"></ion-icon>
</ion-item>
</ion-col>
```

Figure 146: HTML Like Button Functions

This in turn produces the following console result, when the like button is clicked.

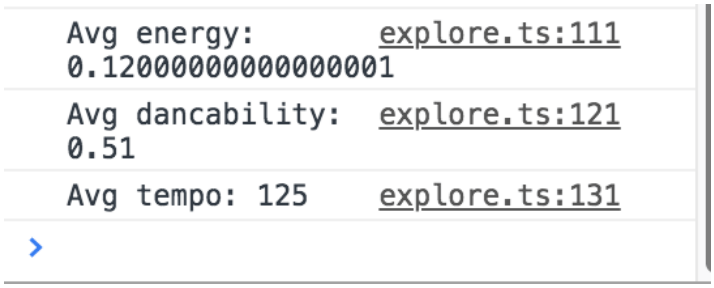

Figure 147: Console Result

10.8.5 Removing Cards

Currently, the cards that are liked or disliked stay in the array. This could become confusing for some users. In turn, applications like Tinder remove cards when they have been liked or disliked. Therefore, the author will also implement this feature.

As the explore page displays the data that is in the array, the data can just be removed, therefore removing the card from the display. First, the index of the item needs to be established so that the function can remove the correct item in the array. This is done by using the “let I = index” syntax within the card element of the HTML, *Cory Rylan, 2015.* This is shown in *figure 148.*

```
<ion-card *ngFor="let swipableTrack of swipableTracks; let i=index;" class="item-remove-animate">
    <img src={{swipableTrack.img}}>
    <ion-card-content>
      <h2 class="card-title">
        {{swipableTrack.artistName}} - {{swipableTrack.trackName}}
```

Figure 148: Let I = Index Syntax

The ngBind syntax is then used on the like and dislike buttons to grab the index, and pass it to a function within the explore.ts file, *AngularJS, 2018.*

```
<ion-col width-50>
    <ion-item class="alignText" ng-bind="i">
      <ion-icon class="likeButtons" name="close"></ion-icon>
    </ion-item>
</ion-col>
<ion-col width-50>
    <ion-item class="alignText" ng-bind="i"
    (click)="energyClick(swipableTrack.energy)" (click)="danceClick(swipableTrack.dancibility)"
    (click)="tempoClick(swipableTrack.bpm)">
      <ion-icon class="likeButtons" name="heart"></ion-icon>
</ion-item>
</ion-col>
```

Figure 149: NgBind Added To Buttons

Click events can then be added to the buttons which will pass the index value over to the corresponding function, by adding i within the brackets. In this case removeDislike and removeLike names have been used.

```
<ion-col width-50>
    <ion-item class="alignText" ng-bind="i" (click)="removeDislike(i)">
        <ion-icon class="likeButtons" name="close"></ion-icon>
    </ion-item>
</ion-col>
<ion-col width-50>
    <ion-item class="alignText" ng-bind="i" (click)="removeLike(i)"
    (click)="energyClick(swipableTrack.energy)" (click)="danceClick(swipableTrack.dancibility)"
    (click)="tempoClick(swipableTrack.bpm)">
        <ion-icon class="likeButtons" name="heart"></ion-icon>
    </ion-item>
</ion-col>
```

Figure 150: Passing Index To Button Function

The two functions can now be created within the explore.ts with the corresponding names. *Figure 151 and 152* shows the i value being pulled from the HTML, by adding the i within the brackets.

```
removeLike(i) {
```

Figure 151: Getting Index In Remove Like Function

```
removeDislike(i) {
```

Figure 152: Getting Index In Remove Dislike Function

The splice function is then used to remove the item in the array at the index that is specified in the HTML. This in turn causes the information within the array, that is on a specific card, to be removed from the display.

```
removeLike(i) {
    this.swipableTracks.splice(i, 1);
```

Figure 153: Removing Like Index Item From Array

```
removeDislike(i) {
  this.swipableTracks.splice(i, 1);
```

Figure 154: Removing Dislike Index Item From Array

10.8.6 Displaying Choice Notification

As the cards disappear from the array without any animation, the author decided it would be a good idea to display a small notification confirming the choice that the user has made. Angular has built in notification objects called Toasts. The notifications pop up for a desired amount of time and can contain text and images. They are non-obstructive and are great for confirming choices made by users, such as adding an item to a playlist, *Ionic, 2018.*

First the Toast Controller must be imported to the page within the Ionic-angular import, which can be seen in *figure 155.*

```
import { IonicPage, NavController, NavParams, Platform, ModalController, ToastController } from 'ionic-angular';
```

Figure 155: Importing Toast Controller

Once the Toast Controller has been imported it can be called from within a function. *Figure 156* shows the toast controller added to the removeLike function created previously.

```
removeLike(i) {
  this.swipableTracks.splice(i, 1);

  let toast = this.tstCtrl.create({
    message: "Hello",
    duration: 2000,
    position: 'bottom'
  })

  toast.present();

}
```

Figure 156: Creating Toast Notification In Remove Like Function

This produces the following result, which displays the hello notification at the bottom of the screen.

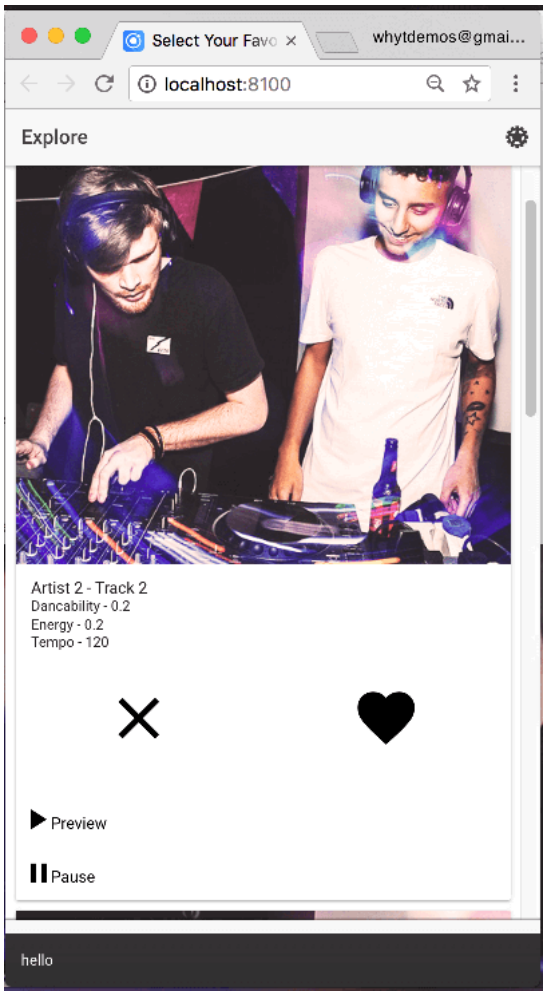

Figure 157: Test Toast Notification

To make the notification display more details about the track and the event that has happened, a similar approach to the removing cards section is needed.

First the track name and artist name are grabbed from the HTML. This is done by simply adding the required attributes to the already existing click event within the HTML.

```
(click)="removeDislike(i, swipableTrack.artistName, swipableTrack.trackName)">
```

Figure 158: Passing Track Name And Artist Name In HTML

The function is then updated within the explore.ts file, to pull the values from the HTML. As seen in *figure 159.*

```
removeLike(i, artist, track) {
```

Figure 159: Passed Artist And Track Name

These are then used within the message section of the toast notification, along with “Liked” to notify the user they have liked the track.

```
let toast = this.tstCtrl.create({
  message: artist+' '+track+' Liked',
  duration: 2000,
  position: 'bottom'
})
```

Figure 160: Artist And Track Attributes Displayed Within Like Toast Notification

The same is then done for the removeDislike function, which is assigned to the dislike button, with the appropriate message.

```
removeDislike(i, artist, track) {
  this.swipableTracks.splice(i, 1);

  let toast = this.tstCtrl.create({
    message: artist+' '+track+' Disliked',
    duration: 2000,
    position: 'bottom'
  })

  toast.present();
```

Figure 161: Artist And Track Attributes Displayed Within Dislike Toast Notification

This in turn produces the following result, when simulating the application within the browser.

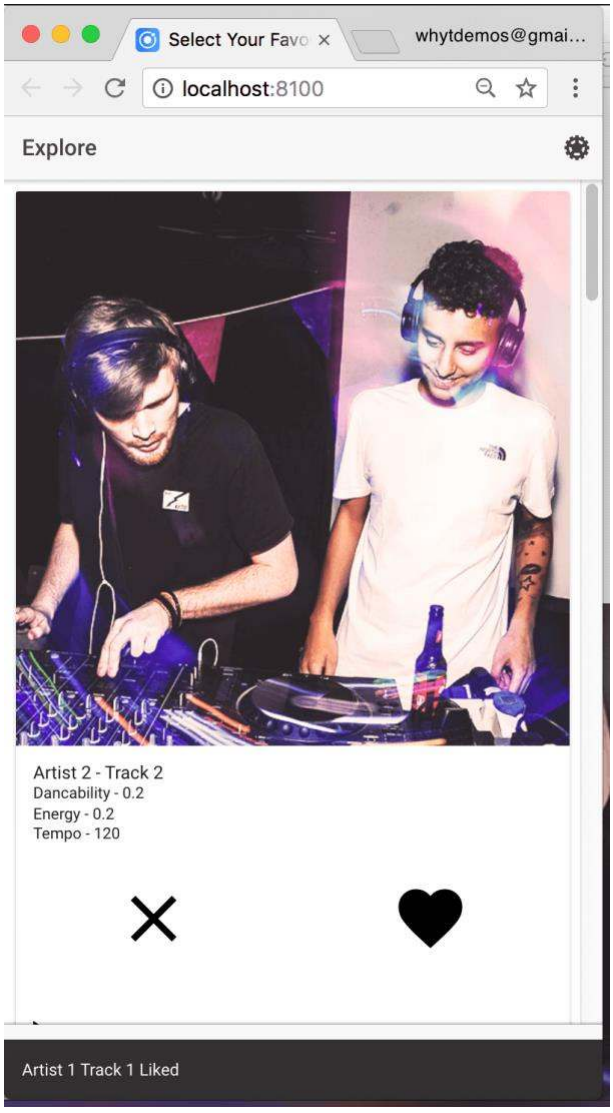

Figure 162: Liked Toast Notification

Figure 163: Disliked Toast Notification

10.9 Creating The Recommendation Page

The recommendation page is required to create a playlist of tracks, that matches the style of music that the user liked or disliked on the explore page. As the energy, dancability and tempo have been taken from the tracks and averaged, this variable can be used to query the Firebase Database for tracks that have a similar value.

10.9.1 Designing The Recommendation Page

In the design specification, the recommendation page is required to display a list of recommended tracks and must allow users to play the tracks that have been recommended.

As the page is only required to display a list, the ngFor* approach is taken once again. *Figure 164* shows the HTML design of each of the items on the page.

```
<ion-list>
    <ion-item>
        <ion-avatar item-left>
            <img src="">
        </ion-avatar>
        <h2></h2>
        <p>
            BPM:
        </p>
        <p>
            Energy:
        </p>
        <p>
            Dancability:
        </p>
    </ion-item>
</ion-list>
```

Figure 164: Recommendations Page HTML Design

In the design specification, the page must also contain a button which allows the user to choose the criteria, of which the items in the list are filtered by. *Figure 165* shows a button added to the navigation bar which will be used to achieve this.

```
<ion-navbar>
    <ion-title>iUgo Recommendation</ion-title>
    <ion-buttons end>
        <button ion-button icon-only>
          <ion-icon name="options"></ion-icon>
        </button>
    </ion-buttons>
</ion-navbar>
```

Figure 165: Navigation Bar Button

10.9.2 Retrieving The Average Variables

First, the average values that are stored in the variables from the explore page, must be sent to the recommendations page. This is done by adding data variables to the navigation parameters, which can be seen in *figure 166, Ionic, 2018.*

```
this.navCtrl.push(ReccomendationsPage, {
  data: this.averageEnergy,
  data2: this.averageDance,
  data3: this.averageTempo,
});
```

Figure 166: Sending Average Variables Through NavParams From Explore Page

The variables are then received on the recommendations page, using the following code found in *figure 167.*

```
this.averageEnergy = navParams.get('data');
console.log(this.averageEnergy);

this.averageDance = navParams.get('data2');
console.log(this.averageDance);

this.averageTempo = navParams.get('data3');
console.log(this.averageTempo);
```

Figure 167: Receiving Average Variables Through NavParams

The values are then stored within variables on the recommendations page, which can then be used within queries to the Firebase database.

10.9.3 Querying The Database

Once the average values have been received, the query to the database can be made. Firebase has many different queries that can be used to narrow down data that is sent back, which can be found at the following, *Google, 2018.*

The author is going to use the orderByChild method. This will sort through the specified attribute within the database, for example energy. The startAt method can then be used which will only return values that have at least the specified value and above. In the authors case this value is the averages received previously.

Figure 168 shows the orderByChild query code that is loaded when the page is first loaded. It shows queries the database on tempo and only returns tracks that have a tempo of higher than the average value that was calculated on the explore page.

```
this.af.list('/tracks', {
  query: {
    orderByChild:'tempo',
  startAt: this.averageTempo
  }})
```

Figure 168: OrderByChild Query

Once the tracks have been narrowed down by the query, they are then added to an array. The attributes of all the track are also added to the array, which can be seen in *figure 169.*

```
.subscribe(data => {
  this.tracksArr = [];

  data.forEach(pope => {
    this.allTracks.subscribe(all=>{
      all.forEach(alle=>{
        if(alle.$key==pope.$key){


   this.tracksArr.push(

    { src: alle.url,
      artist: alle.artistName,
      title: alle.name,
      art: alle.albumArt,
      preload: 'metadata',
      key:alle.$key,
      artistId:alle.artist,
      albumId:alle.album,
      album:alle.albumName,
      artistName:alle.artistName,
      tempo:alle.tempo,
      dancability:alle.dancability,
      energy:alle.energy
      }
```

Figure 169: Adding Track Attributes To Array

10.9.4 Change Criteria Button

The author would like the user to be able to choose the criteria they want to sort the recommendations by. This is because only one criteria at a time can be used with Firebase. Therefore, the author would like to give the user the option to choose between energy, dancability and tempo query criteria. To keep this simple, without creating another page, an ActionSheet can be used, which will pop up and allow the user to change the criteria.

Once the action sheet controller has been imported to the page, it can be called upon using the code in *figure 170.* In this case, the author has assigned the action sheet to a function called “chooseCriteria”.

```
chooseCriteria() {
  let actionSheet = this.actionSheetControl.create({
    title: 'Choose The Search Criteria',
    buttons: [
      {
```

Figure 170: Action Sheet

The same code that is executed when the page is first loaded, can then be used within the action sheet. *Figure 171* shows the code that is executed when the energy action sheet button is clicked.

```
chooseCriteria() {
  let actionSheet = this.actionSheetControl.create({
    title: 'Choose The Search Criteria',
    buttons: [
      {
        text: 'Energy',
        handler: () => {
          this.af.list('/tracks', {
            query: {
              orderByChild:'energy',
            startAt: this.averageEnergy
            }})
            .subscribe(data => {
              this.tracksArr = [];

              data.forEach(pope => {
                this.allTracks.subscribe(all=>{
                  all.forEach(alle=>{
                    if(alle.$key==pope.$key){


                this.tracksArr.push(

                { src: alle.url,
                  artist: alle.artistName,
                  title: alle.name,
                  art: alle.albumArt,
                  preload: 'metadata',
                  key:alle.$key,
                  artistId:alle.artist,
                  albumId:alle.album,
                  album:alle.albumName,
                  artistName:alle.artistName,
                  tempo:alle.tempo,
                  dancability:alle.dancability,
                  energy:alle.energy
```

Figure 171: Action Sheet Energy Button

As can be seen in the image, most of the code is the same, however the orderByChild and startAt attributes are tailored to the energy query. As this code is modifying the data in the current array, the change is instant.

The same is done for the dancability and tempo buttons on the action sheet, as seen the following *figures.*

```
text: 'Dancability',
handler: () => {
  this.af.list('/tracks', {
    query: {
      orderByChild:'dancability',
    startAt: this.averageDance
    }})
    .subscribe(data => {
      this.tracksArr = [];

      data.forEach(pope => {
        this.allTracks.subscribe(all=>{
          all.forEach(alle=>{
            if(alle.$key==pope.$key){


       this.tracksArr.push(

       { src: alle.url,
         artist: alle.artistName,
         title: alle.name,
         art: alle.albumArt,
         preload: 'metadata',
         key:alle.$key,
         artistId:alle.artist,
         albumId:alle.album,
         album:alle.albumName,
         artistName:alle.artistName,
         tempo:alle.tempo,
         dancability:alle.dancability,
         energy:alle.energy
         }
```

Figure 172: Dancability Action Sheet Button

```
text: 'Tempo',
handler: () => {
  this.af.list('/tracks', {
    query: {
      orderByChild:'tempo',
    startAt: this.averageTempo
    }})
    .subscribe(data => {
      this.tracksArr = [];

      data.forEach(pope => {
        this.allTracks.subscribe(all=>{
          all.forEach(alle=>{
            if(alle.$key==pope.$key){


       this.tracksArr.push(

       { src: alle.url,
         artist: alle.artistName,
         title: alle.name,
         art: alle.albumArt,
         preload: 'metadata',
         key:alle.$key,
         artistId:alle.artist,
         albumId:alle.album,
         album:alle.albumName,
         artistName:alle.artistName,
         tempo:alle.tempo,
         dancability:alle.dancability,
         energy:alle.energy
         }
```

Figure 173: Tempo Action Sheet Button

10.9.5 Adding The HTML

Once the tracks are in the array, the ngFor* can be used to display them along with all the required attributes, just like in previous steps. *Figure 174* shows the newly added HTML tags.

```
<ion-list>
    <ion-item *ngFor="let item of tracksArr">
      <ion-avatar item-left>
        <img src={{item.art}}>
      </ion-avatar>
      <h2>{{ item.artistName }} - {{ item.title }}</h2>
      <p>
        BPM: {{ item.tempo }}
      </p>
      <p>
        Energy: {{ item.energy }}
      </p>
      <p>
        Dancability: {{ item.dancability }}
      </p>
    </ion-item>
  </ion-list>
```

Figure 174: HTML Tags On Recommendations Page

This produces the following result when simulating the application within the browser.

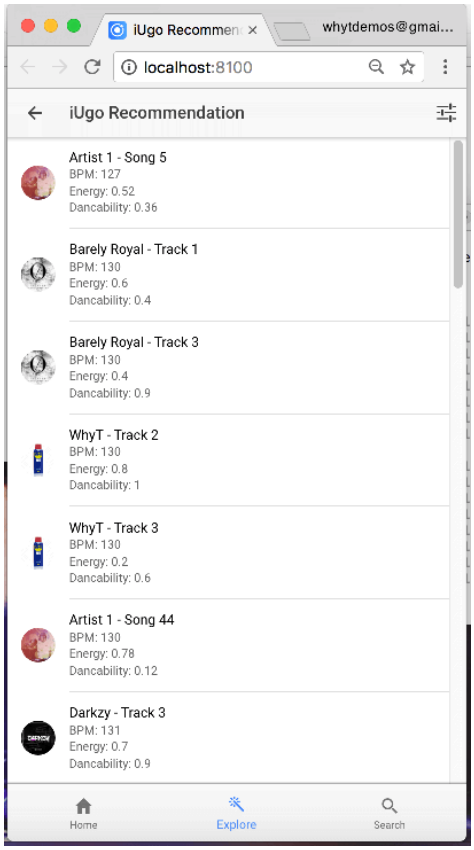

Figure 175: Recommendation Page In Simulator

The “chooseCriteria” function is then added to the HTML button on the navigation bar, as seen in *figure 176.*

```
<ion-navbar>
    <ion-title>iUgo Recommendation</ion-title>
    <ion-buttons end>
        <button ion-button icon-only (click)="chooseCriteria()">
            <ion-icon name="options"></ion-icon>
        </button>
    </ion-buttons>
</ion-navbar>
```

Figure 176: Choose Criteria Button Assignment

Which produces the following result.

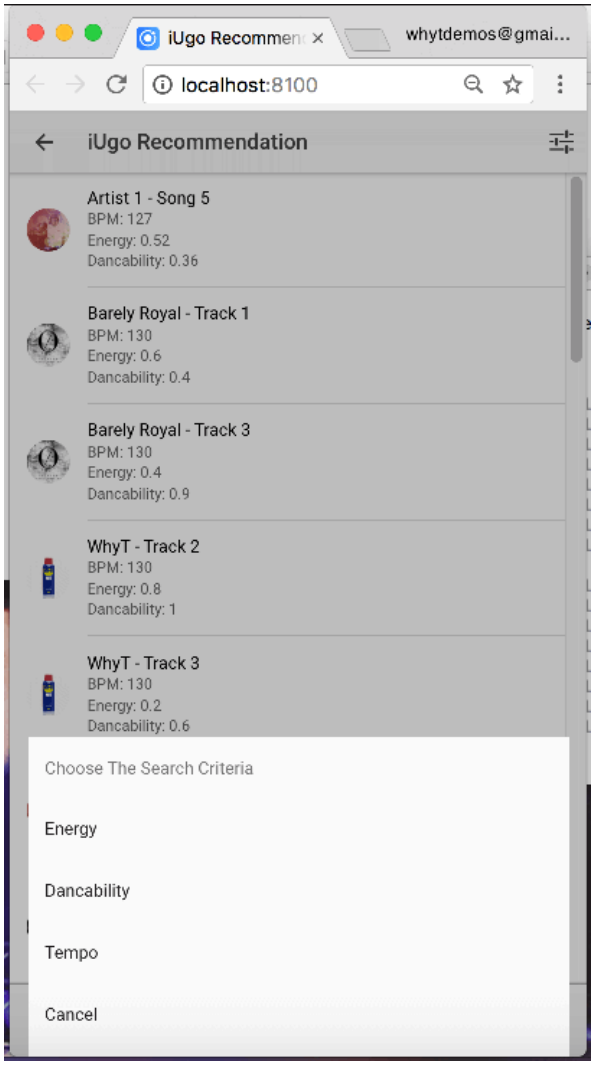

Figure 177: Choose Criteria Button In Simulator

The author has then modified the tempoClick function within the explore.ts file, to display the recommendation page when the user has liked five tracks. *Figure 178* shows this.

```
this.selectedCounter ++;
if (this.selectedCounter == 5) {
  this.navCtrl.push(ReccomendationsPage, {
    data: this.averageEnergy,
    data2: this.averageDance,
    data3: this.averageTempo,
  });
  this.selectedCounter = 0;
}
```

Figure 178: Displaying Recommendations After Five Likes

This is done by creating a variable that stores a number and is incremented every time the function is called. When the recommendation page is displayed, the number is reset to zero, allowing the user to select five more tracks.

10.10 Creating The Artist Selection Page

As the Spotify API would allow users to select a few artists before returning a list of tracks that are like the style of those artists, the author has decided to do the same with the local JSON displayed on the explore page.

The artist selection page is required to display a list of items that can be selected, therefore the ngFor* method will be used to display the items in the array. Just like the explore page, the artist selection page is going to need a local JSON file that contains the artists. For this, the author has chosen to use the energy value as the query criteria for the explore page.

10.10.1 Designing The Artist Selection Page

As mentioned previously, this page is required to display a list of artists as well as allow users to search for artists. A close button to dismiss the view is also required, which is added to the navigation bar. *Figure 179* shows the HTML design for this.

```
<ion-navbar>
    <ion-title>Select Your Favourite Artists</ion-title>
    <ion-buttons end>
        <button ion-button icon-only>
          <ion-icon name="close"></ion-icon>
        </button>
      </ion-buttons>
  </ion-navbar>

ion-header>


on-content padding>
  <ion-searchbar></ion-searchbar>
      <ion-list>
        <ion-item>
          <h2></h2>
          <h3></h3>
          <p></p>
          <ion-avatar item-left>
              <img src="">
            </ion-avatar>
        </ion-item>
      </ion-list>
```

Figure 179: Artist Selection Page Design

10.10.2 Displaying Artists JSON

Just like in the explore page, a JSON file is created with the artist name, average artist energy value, artist genre and path to artist image, as seen in *figure 180.*

```
{
    "artistsArr": [
        {
            "artistName": "Drake",
            "averageTrackBpm": "140",
            "averageEnergy": "0.1",
            "img": "./assets/data/drake.png",
            "genre": "Hip Hop"
        },
        {
```

Figure 180: Artist JSON

The JSON file is then loaded into the artist selection page using the dataFinder framework, after it has been imported. The JSON is then pushed into an array within the artist selection page.

```
ionViewDidLoad() {
  this.dataFinder.getJSONDataAsync("./assets/data/artists.json").then(data => {
    this.SetQueryOptionsData(data);
});


SetQueryOptionsData(data : any) {
  this.artistsArr = data.artistsArr;
}
```

Figure 181: Loading Artist JSON

The HTML can then be modified accordingly to display the data in the array.

```
<ion-content padding>
    <ion-searchbar></ion-searchbar>
        <ion-list *ngFor="let item of artistsArr">
            <ion-item>
              <h2>{{item.artistName}}</h2>
              <h3>{{item.genre}}</h3>
              <p>Average Energy: {{item.averageEnergy}}</p>
              <ion-avatar item-left>
                  <img src={{item.img}}>
               </ion-avatar>
            </ion-item>
        </ion-list>
```

Figure 182: Artist Selection HTML

This produces the following result.

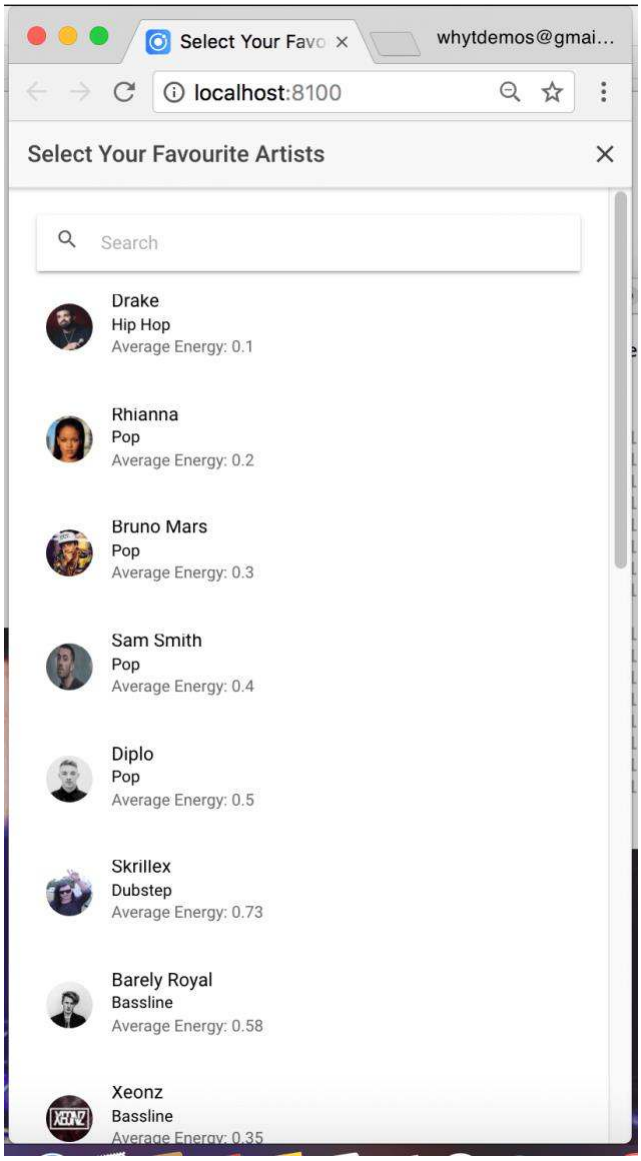

Figure 183: Artist Selection Page In Simulator

10.10.3 Selecting The Artist

Now that the artist data is being displayed within the list, a function needs to be added to each of the entries to grab the energy data from each of the artists, and store it within a variable.

This is done by assigning the HTML list item to a function, with the artist energy attached. This can be seen in *figure 184.*

```
<ion-item (click)="energyClick(item.averageEnergy"
```

Figure 184: Passing Energy Value To Function

The “energyClick” function can then be created in the artist-selection.ts file. This function must take the energy values from the JSON and average them, using the previously mentioned algorithm.

```
energyClick(energy) {
  this.energyArr.push(energy);

  var sum2 = 0;
  for(var j = 0; j < this.energyArr.length; j++){
      sum2 += parseFloat(this.energyArr[j]);
  }
  this.energyAvgVal = sum2/this.energyArr.length;

  console.log("Avg Energy: "+this.energyAvgVal);
```

Figure 185: Artist Selection Average Energy Algorithm

An average energy variable is now being stored in a variable that can be used to narrow down the tracks that are displayed on the explore page.

10.10.4 Filtering Explore Page Results

For the artist selection page to be effective, it must be displayed before the explore page. Therefore, an Ionic modal controller is used, *Ionic, 2018.* By using this method data can be passed forwards to the explore page and allow the page to be displayed when the explore page is first loaded.

Figure 186 shows the modal controller code that is executed within the explore.ts constructor, which allows it to be displayed first.

```
let modal = this.modalCtrl.create(ArtistSelectionPage);
    modal.onDidDismiss((energy) => {

      this.avgEnergyJson = energy;

      console.log(this.avgEnergyJson);
      this.swipableTracks = this.swipableTracks.filter((criteria) => {
        return criteria.energy <= this.avgEnergyJson+0.1 && criteria.energy >= this.avgEnergyJson-0.1;
      });
    });
    modal.present();
```

Figure 186: Modal Controller Code

This code also grabs the average energy variable from within the artist selection page. As the explore.ts file contains the array that contains the tracks within the local JSON. A filter criterion can be used to narrow down the results within that array.

In the authors case, the filter criteria only display results that are 0.1 above and below the average energy value taken from the artist selection page.

10.10.5 Search For Artists

As the list of artists can become quite long, the author has implemented a search bar into the artist selection page. *Figure 187* shows the searchQuery code that will monitor the search field for any text input.

```
searchQuery(param : any) : void {

let val : string = param;

this.dataFinder.getJSONDataAsync("./assets/data/artists.json").then(data => {
 this.SetQueryOptionsData(data);

// DON'T filter the technologies IF the supplied input is an empty string
if (val.trim() !== '')
{
    this.artistsArr = this.artistsArr.filter((item) =>
    {
        return item.artistName.toLowerCase().indexOf(val.toLowerCase()) > -1;

    })
  }
 }
)}
```

Figure 187: Artist Selection Page Search Query

This produces the following result.

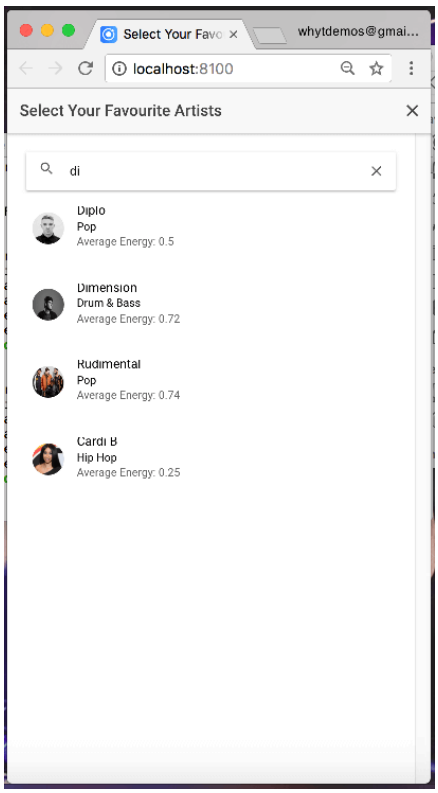

Figure 188: Search Query In Browser Simulator

10.10.6 Close Button

Once the user has selected as many artists as they wish, they must close the modal controller to start making their selections on the explore page. Therefore, a close button is added to the artist selection page navigation bar, with a function called “close”. The HTML can be seen in *figure 189.*

```
<ion-navbar>
    <ion-title>Select Your Favourite Artists</ion-title>
    <ion-buttons end>
        <button ion-button icon-only (click)="close()">
          <ion-icon name="close"></ion-icon>
        </button>
     </ion-buttons>
  </ion-navbar>
```

Figure 189: Close Button HTML

The close function is then created within the artist-selection.ts file. To ensure the user has selected at least one artist, a small amount of validation is added, which will check to ensure that the energy value is not equal to zero. If it is, an alert is displayed to the user prompting them to make at least one selection. This is necessary to ensure that the tracks displayed on the explore page are somewhere narrowed down to fit the users taste. *Figure 190* shows this.

```
close() {
  if (this.energyAvgVal == 0) {
    let alert = this.alertCtrl.create({
      title: 'Alert',
      subTitle: 'Please enure you have selected at least 1 artist.',
      buttons: ['Dismiss']
    });
    alert.present();
  }
  else {
    this.viewCtrl.dismiss(this.energyAvgVal);
  }
}
```

Figure 190: Artist Selection Page Close Button Function

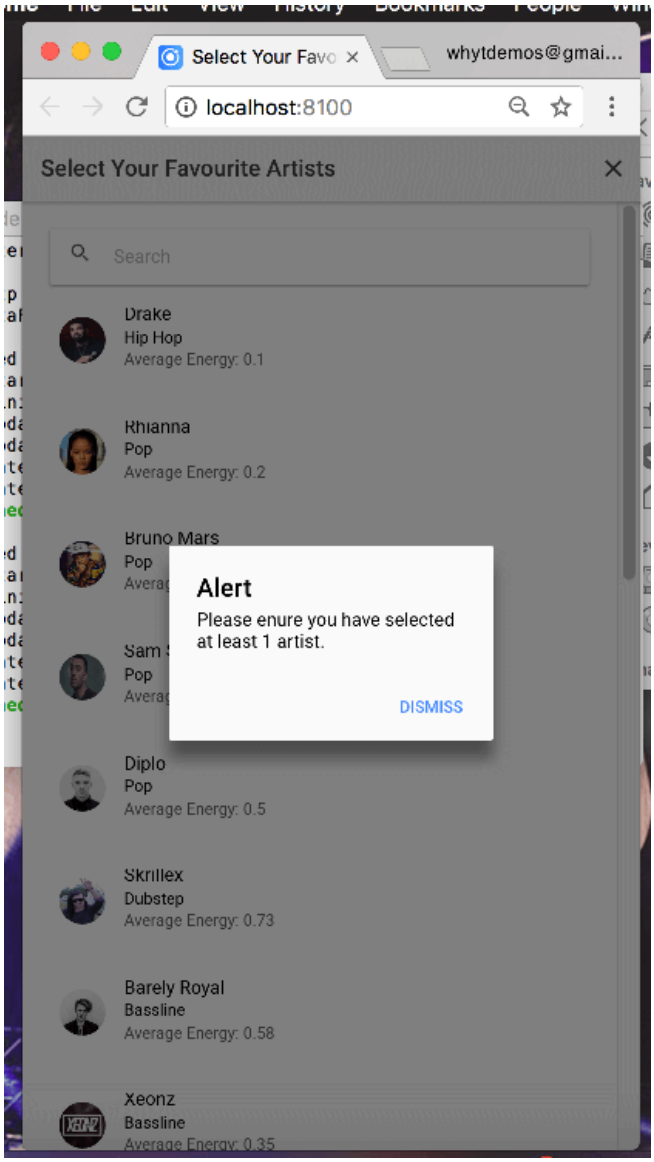

Figure 191: Close Button Validation In Simulator

10.10.7 Toast Notification

As there is no animation for when the artist has selected an artist, the toast controller has been added to the “energyClick” function, which displays a small notification to notify the user of their selection. This is done by adding the data needed to the HTML button, as seen in *figure 192*, and using it within the “energyClick” function, as seen in *figure 193.*

```
<ion-item (click)="energyClick(item.averageEnergy, item.artistName)">
```

Figure 192: Energy Click HTML

```
energyClick(energy, name) {


  let toast = this.tstCtrl.create({
    message: name +' Added',
    duration: 2000,
    position: 'bottom'
  });


  toast.present();
}
```

Figure 193: Energy Click Function

This produces the following result within the artist selection page.

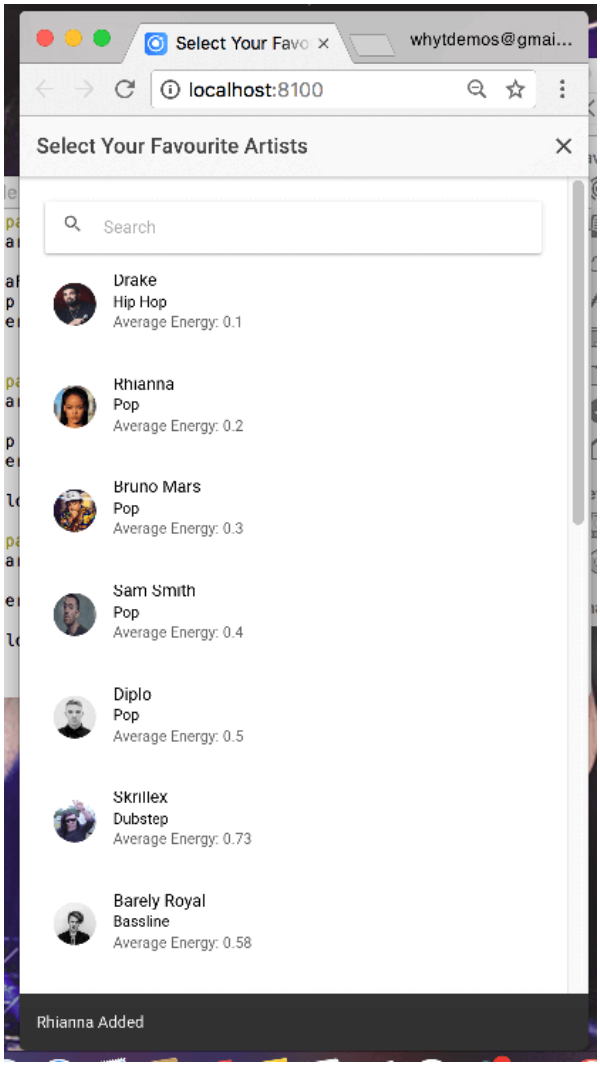

Figure 194: Toast Notification In Artist Selection Page

10.11 The Audio Player Page

The audio player page will be displayed when the user clicks on the small audio player at the bottom of each page. This player will only be displayed when a track is playing. Following the guide from *Raja Yogan, 2016,* aided this process.

10.11.1 Designing the Player Page

As mentioned in the design section, the player page must contain a play button, a pause button, a skip button, a previous button and a track duration bar. Using an example project from *Methee Banjukeaw, 2017,* the author was able to design an audio player within the player page HTML. *Figure 195* shows the player page HTML.

```
<ion-content  style="background: white !important;text-align: center" >
  <ion-row style="height:60%">
    <ion-col col-12 style="text-align: center;height: 100%;position: relative">
    <img class="shadow" style="height: 100%" src="">
    </ion-col>
  </ion-row>
  <ion-row style="height:15%">
    <ion-col col-2  style="text-align: center"></ion-col>
      <ion-col col-8 style="text-align: center">
          <h1></h1>
          <h2></h2>
      </ion-col>
    <ion-col col-2 style="text-align: center"></ion-col>
  </ion-row>
  <ion-row style="height:15%">
    <ion-col col-12>
      <audio-track-progress-bar style="color: #488aff"></audio-track-progress-bar>
    </ion-col>
  </ion-row>
  <ion-row style="height:10%">
    <ion-col col-3 style="text-align: center"></ion-col>
    <ion-col col-2 style="text-align: center">
      <button  ion-button clear style="color: #488aff" icon-only>
        <ion-icon name="skip-backward"></ion-icon>
      </button>
    </ion-col>
    <ion-col col-2 style="text-align: center">
      <audio-track-play class="large" style="font-size: 150%"></audio-track-play>
    </ion-col>
    <ion-col col-2 style="text-align: center">
      <button  ion-button clear style="color: #488aff" icon-only>
        <ion-icon name="skip-forward"></ion-icon>
      </button>
    </ion-col>
    <ion-col col-3 style="text-align: center">
    </ion-col>
```

Figure 195: Player Page HTML

This produces the following result within the simulator.

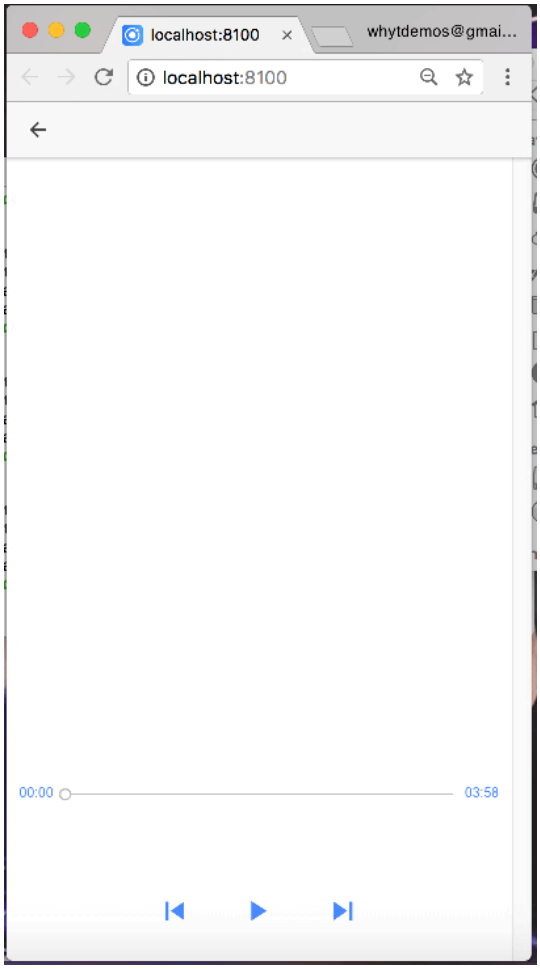

Figure 196: Player Page Within Simulator

10.11.2 Adding Audio Controls

To play audio within the application the Ionic Audio plugin must be installed along with a service provider file from, *Daniyalahmedkhan, 2017.* The plugin is installed using the following command within the project directory.

- `npm install --save ionic-audio@3.2.0`

The service provider file is installed by simply downloading it and placing it in the "providers" folder.

The provider is imported into the player page, as seen in *figure 197,* it can then be called upon within functions throughout the page.

```
import { MusicService } from '../../providers/musicservice';
```

Figure 197: Importing Music Service Provider

A function called "play" is then created in the player.ts file. This calls upon the service provider and plays the track within the track variable, as seen in *figure 198.* This also tells the music service provider to assign the track that is playing, to the playingTrack variable within it.

```
play(track){
  this.musicservice.playnext(track);
  this.tracks=this.musicservice.audio.tracks[0];
```

Figure 198: Play Function

The same approach is taken for the next and previous track functions which can be seen in the following *figures.*

```
prev(){
  if(this.musicservice.wait){
    console.log("waiting");
  }
  else{
    this.musicservice.prev();
    this.tracks=this.musicservice.audio.tracks[0];

  }
}
```

Figure 199: Next Function

```
next(){
  if(this.musicservice.wait){
    console.log("waiting");
  }
  else{
    this.musicservice.next();
    this.tracks=this.musicservice.audio.tracks[0];

  }
}
```

Figure 200: Previous Function

10.11.3 Displaying Currently Playing Audio

As the track that is currently playing is assigned to the playingTrack variable within the music service provider, this can be called upon within the HTML to display the currently playing track information. *Figure 201* shows the modified HTML to display the currently playing track information, as well as adding the previously created functions, to the buttons.

```
<ion-row style="height:15%">
  <ion-col col-2  style="text-align: center"></ion-col>
    <ion-col col-8 style="text-align: center">
        <h1>{{musicservice.playingTrack.title}}</h1>
        <h2>{{musicservice.playingTrack.artist}} </h2>
    </ion-col>
  <ion-col col-2 style="text-align: center"></ion-col>
</ion-row>
<ion-row style="height:15%">
  <ion-col col-12>
    <audio-track-progress-bar progress duration [audioTrack]="musicservice.audio.tracks[0]" style="color: #488aff"></audio-
  </ion-col>
</ion-row>
<ion-row style="height:10%">
  <ion-col col-3 style="text-align: center"></ion-col>
  <ion-col col-2 style="text-align: center">
    <button (click)="musicservice.prev()"  ion-button clear style="color: #488aff" icon-only>
      <ion-icon name="skip-backward"></ion-icon>
    </button>
  </ion-col>
  <ion-col col-2 style="text-align: center">
    <audio-track-play class="large" [audioTrack]="musicservice.audio.tracks[0]" style="font-size: 150%"></audio-track-play>
  </ion-col>
  <ion-col col-2 style="text-align: center">
    <button (click)="musicservice.next()"  ion-button clear style="color: #488aff" icon-only>
      <ion-icon name="skip-forward"></ion-icon>
    </button>
  </ion-col>
  <ion-col col-3 style="text-align: center">
  </ion-col>
```

Figure 201: Modified Player HTML

10.11.4 Playing Audio On Other Pages

To start audio playback from other pages within the application, the page must already contain an array with a track, along with all its attributes as shown in *figure 202.*

```
this.theTracksPlaylists.push(

        {src: alle.url,
        artist: alle.artistName,
        title: alle.name,
        art: alle.albumArt,
        preload: 'metadata',
        key:alle.$key,
        artistId:alle.artist,
        albumId:alle.album,
        album:alle.albumName,
        artistName:alle.artistName,
        }
    );
```

Figure 202: Track Attributes Array

Once the tracks are stored within the array and the music service provider is imported, the play function can be created. *Figure 203* shows the play function for the home page.

```
play(track){
    this.musicservice.play(this.theTracksPlaylists,track);
 this.tracks=this.musicservice.audio.tracks[0];

 }
```

Figure 203: Play Function On Home Page

As can be seen by the code, the music service provider calls upon the play function within itself and passes the track within the array to it. For the correct track to be played within the array, the let i = index must be used within the HTML. *Figure 203* shows the let i = index being passed through to the play function within the HTML.

```
<div class="scroll-item" *ngFor="let artist of theTracksPlaylists ; let i=index;"
(click)="play(i)" style="text-align: center;margin-left:32%;width: 120px;">
```

Figure 203: Home Page Play Function

This will in turn allow the audio to start playing, as well display its track information within the audio player page. *Figure 204* shows the player page with a track from the home page playing.

Figure 204: Player Page With Audio Playing

The same process is taken for the other pages which need to play audio from the Firebase database, including the search page and recommendations page.

```
<h2 (click)="play(artist,i)">
```

Figure 205: Search Page Play Audio HTML

```
play(track,i) {

  this.musicservice.play(this.playlists,i);
  this.tracks=this.musicservice.audio.tracks[0];

}
```

Figure 206: Search Page Play Function

```
<ion-item *ngFor="let item of tracksArr; let i=index;" (click)="playpop(i)">
```

Figure 207: Recommendations Page Play Audio HTML

```
playpop(track){
 this.musicservice.play(this.tracksArr,track);
this.tracks=this.musicservice.audio.tracks[0];

}
```

Figure 208: Recommendations Page Play Audio Function

10.11.5 Creating The Mini Audio Player

As mentioned in the design section, each page must contain a mini audio player that is displayed when a track is playing. To achieve this, the following HTML is added to the bottom of each page that is required to have a mini player.

```
<ion-footer style="background: □white">
  <div *ngIf="tracks" style="height: 3px;z-index: 9999; background-color:■#488aff" [style.width.%]="musicservice.progress()"></div>
  <ion-row *ngIf="tracks">
    <ion-col col-8 (click)="player()">
      <button  ion-button clear block  style="color: ■black;">
  <span style="overflow: hidden;text-overflow: ellipsis">{{tracks.artist}} - {{tracks.title}}</span>
      </button>
  </ion-col>
  <ion-col col-2 style="text-align: right">
        <audio-track-play light [audioTrack]="tracks">
        </audio-track-play>
</ion-col>
  <ion-col col-2 style="text-align: right">
    <button (click)="next()"  ion-button clear style="color:■black" icon-only>
      <ion-icon name="skip-forward"></ion-icon>
    </button>
  </ion-col>
  </ion-row>
</ion-footer>
```

Figure 209: Mini Audio Player HTML

The code utilises an *ngIf which checks to see if a track is playing. If a track is playing the HTML is displayed. If no track is playing the elements is hidden. The array is also utilised to allow the name of the track and the artist to be displayed on the mini player. A function called “player” is assigned to the body of the mini player, which when clicked, displays the player page. Shown in *figure 210.*

```
player(){
    this.navCtrl.push(PlayerPage);
}
```

Figure 210: Display Player Page Function

Next and play icons are also added to the mini player body, to allow the user to play or skip the track without having to enter the player page. The corresponding functions can be seen in *figure 211.*

```
play(track){
    this.musicservice.play(this.theTracksPlaylists,track);
  this.tracks=this.musicservice.audio.tracks[0];

  }

next(){
    this.musicservice.next();
  this.tracks=this.musicservice.audio.tracks[0];

  }
```

Figure 211: Mini Player Control Functions

This in turn produces the following result. Adding the above code to each page, allows the mini player to be displayed throughout the application.

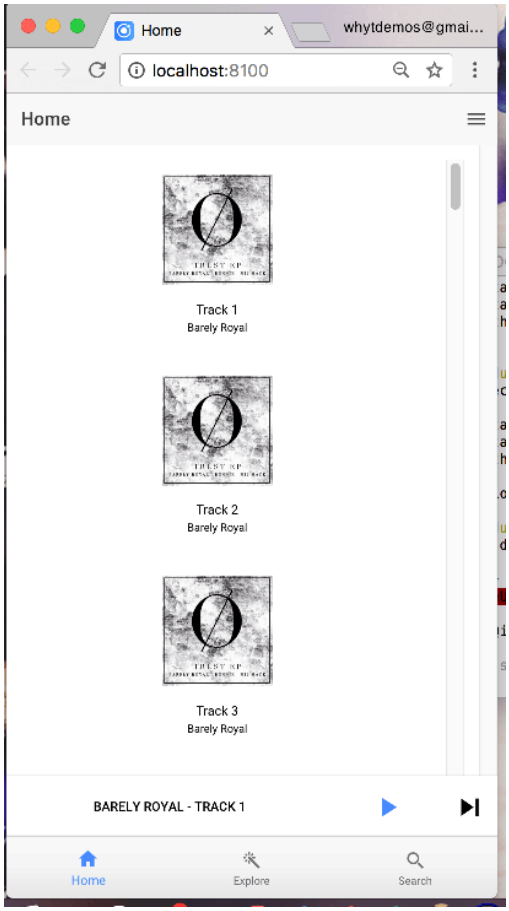

Figure 212: Mini Player In Browser Simulator

11.0 TESTING AND EVALUATION

11.1 Testing The Web Application

To ensure that the web application is working as expected and is useful to users, testing must be conducted. Some foundations for the testing were established in the requirements sections of the web application. Therefore, these will be used as a comparison for the completed product.

11.1.1 Functional Testing

The functional testing for the web application will be done using the use case model, which will go through test cases and ensure the expected output or result occurs. *Table 3* shows the test use cases and the expected outcome, as well as if it performs as expected.

Use Case	Expected Result	Pass/ Fail
After credentials are entered the login fields, the login button is pressed.	User is taken to upload area/ home page.	Pass
Clicking the add button in the side bar.	Upload form is displayed.	Pass
Typing in an artist name in the add artist form and clicking submit.	Artist with desired name is created on the database.	Pass
Clicking artist drop down menu in the add album form.	All available artists on the database are displayed and able to be chosen.	Pass
Clicking the add image button.	Allows user to choose image from their computer.	Pass
Adding an album name and clicking submit.	Adds album with desired name and photo is uploaded to storage area.	Pass
Clicking the artist name drop down menu in the add track form.	Displays a list of all available artists on the database can be selected.	Pass
Clicking the album/ ep name drop down menu in the add track form	Displays a list of albums associated with the artist and can be selected.	Pass
Clicking the track file button	A window is displayed allowing users to select an audio file from their computer.	Pass

Adding the energy, dancability, tempo and track name and clicking submit.	Creates a database entry in the tracks directory, with the stated track attributes and the audio file is uploaded to the storage area.	Pass
Sign out button is clicked	User is signed out and taken back to the login screen.	Pass

Table 3: Functional Testing Use Cases

11.1.2 Non-Functional Testing

As explained in the web application requirement specification, the web application must perform some non-functional requirements.

Portability: Testing on several browsers has been complete and the web application is confirmed to be working accordingly on Google Chrome, Safari and Firefox. Testing on Microsoft Edge and internet explorer was not completed, due to the author not having access to a Windows machine with the correct node.js packages installed. However, there should be no reason why it would not work, if the browsers are up to date.

Stability: Whilst using the web application throughout web browsers, it has been stable and performed to the task each time. Whenever there was an issue with the application, it was down to the network connection not being stable.

Availability: The size of the complete web application is 500MB. This is quite large, however as the web application would be on a server in the real world, this is not an issue for the end user. The end user would simply connect to the server through a web browser.

Usability: As the application only contains a few screens and follows design models from the likes of Facebook and Soundcloud, the usability is expected to be effective. However, the results in the following section will be able to confirm this.

11.1.3 Beta Testing

Beta testing was conducted under the supervision of the author to ensure that the web application was running properly, as the commands to get it working, could be confusing for some users. A total of five participants were used.

The following questionnaire was created using Google Forms, *Google, 2018,* to capture the users' response after using the web application.

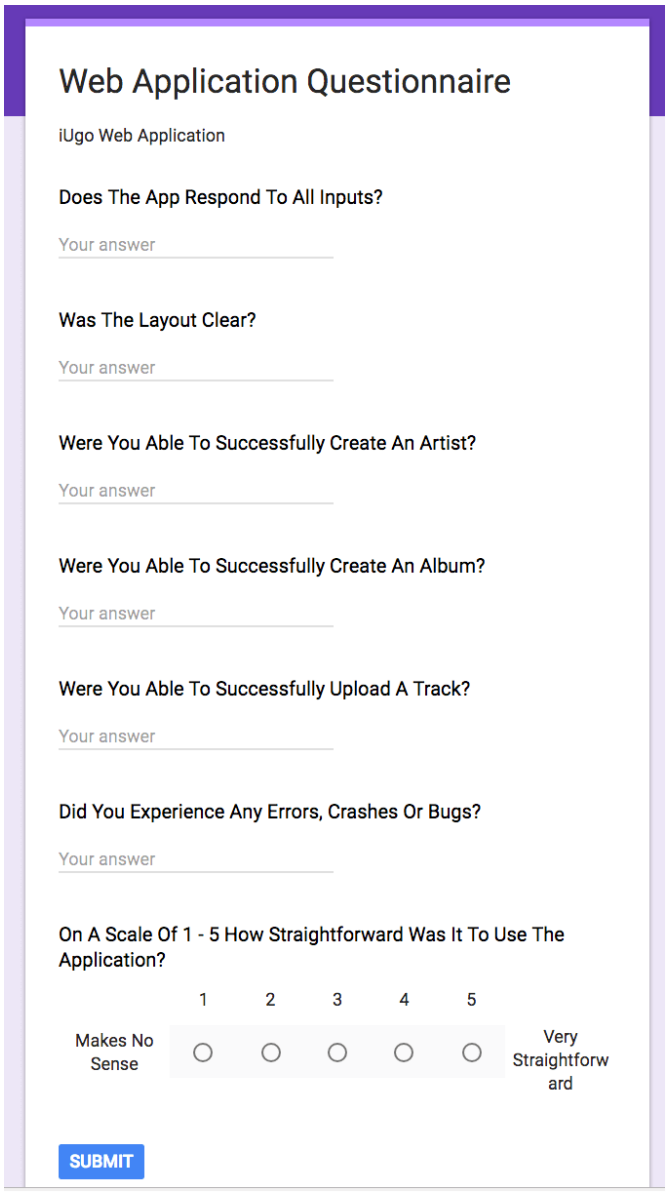

Figure 213: Web Application Questionnaire

The following *figures* contain the results of the questionnaire.

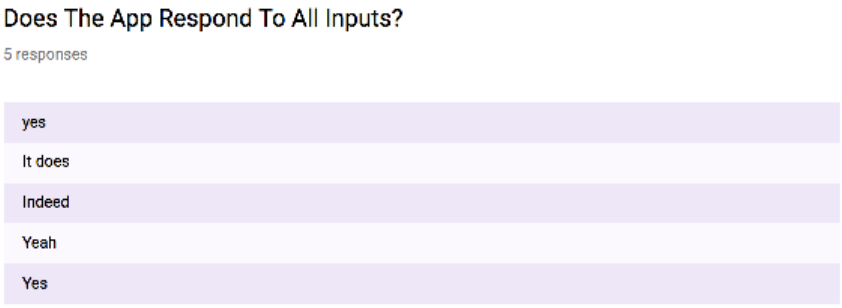

Figure 214: Does The App Respond To All Inputs Result

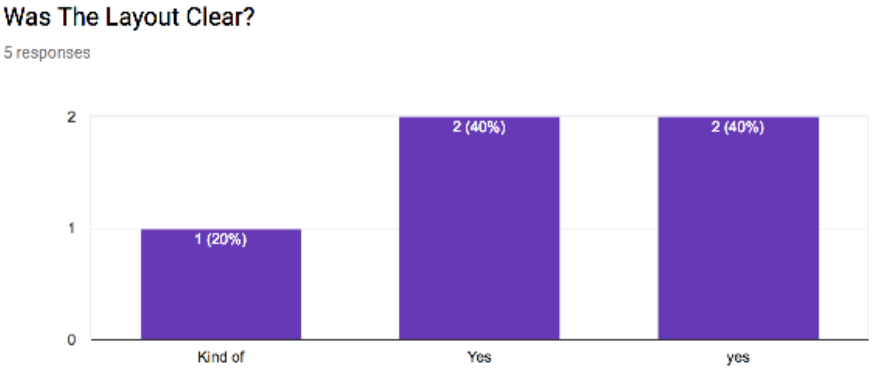

Figure 215: Was The Layout Clear Result

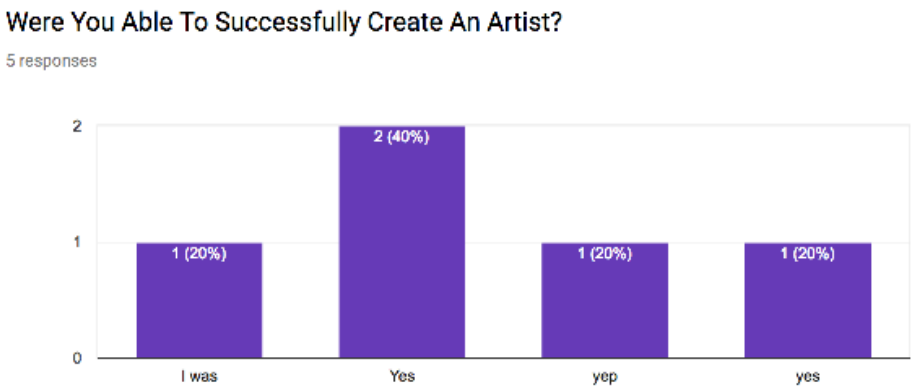

Figure 216: Were You Able To Successfully Create An Artist Result

Figure 217: Were You Able To Successfully Create An Album Result

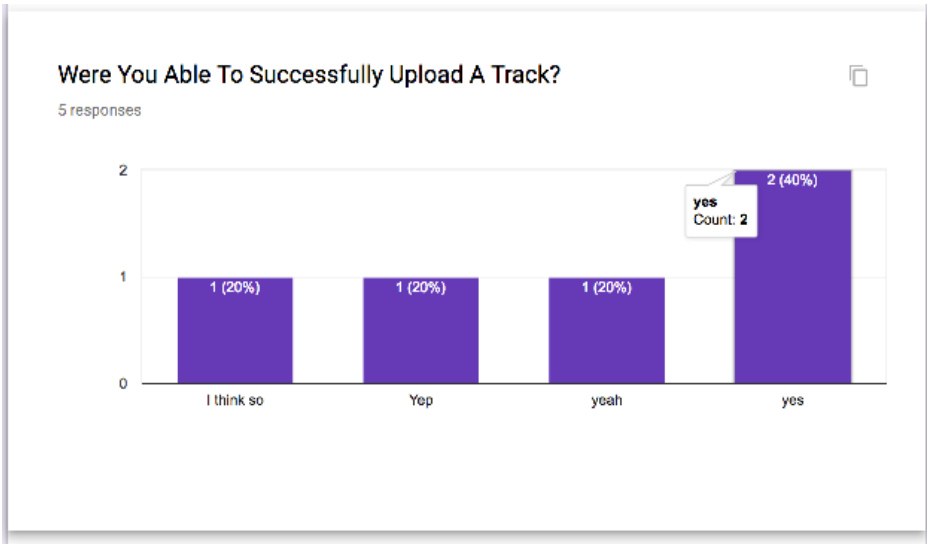

Figure 218: Were You Able To Successfully Upload A Track Result

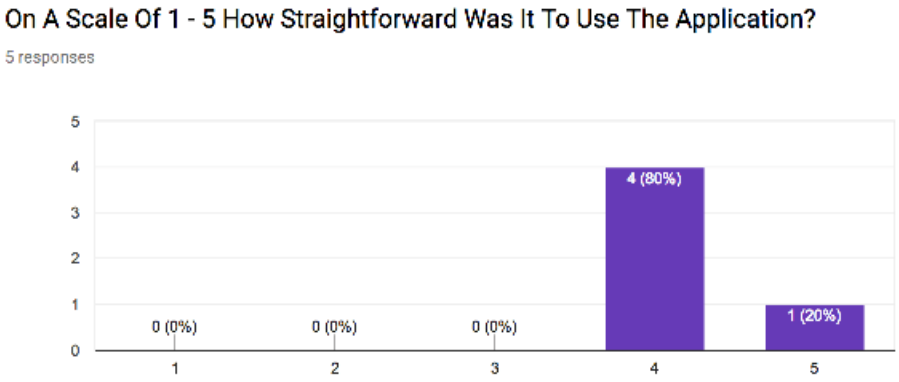

Figure 219: Did You Experience Any Errors Crashes Or Bugs Result

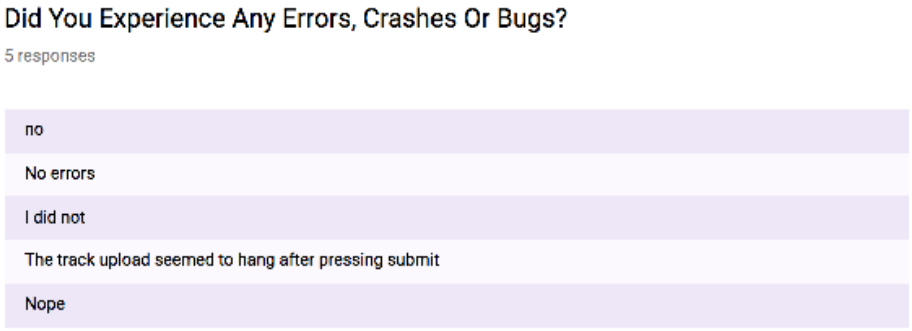

Figure 220: On A Scale Of 1 – 5 How Straightforward Was It To Use The Application Result

11.1.4 Beta Testing Results

As can be seen in the responses from the users, the design of the application is straightforward and relatively easy to use. The application also performs to a good standard and all inputs work within the application.

However, as can be seen in *figure 220* one user experienced a hang when uploading a track in the track form. After careful analysis by the author, replicating the issue was not possible. Therefore, it is understood that the error that occurred at the time was due to an internet connectivity issue, cause the web application to lose connection to the database.

11.2 Testing The Mobile Application

To ensure that the mobile application is working as expected and is useful to its users, some testing must be conducted. Some requirements were stated in the requirements specification earlier in this dissertation. Therefore, these will be used as the foundation for the following sections.

11.2.1 Functional Testing

The use case approach is going to be used to test the functional requirements of the mobile application. *Table 4* shows the use case scenario, the expected result and whether the use case performs accordingly.

Use Case	Expected Result	Pass/ Fail
User enters credentials and presses login button.	User is taken to home page.	Pass
User clicks sign up button.	HTML inputs dynamically change to allow user to sign up.	Pass
User enters sign up credentials and clicks sign up button.	User account is created on database and user is taken to home page.	Pass
User clicks menu button on home page.	User logout page is displayed.	Pass
User clicks logout button on the logout page.	User is logged out and taken to the login page.	Pass
User clicks track on the home page.	Mini player appears, and audio begins to play.	Pass
User clicks on search button in the tab bar.	Search page is displayed.	Pass
User enters search criteria into search bar.	Tracks matching search criteria are displayed.	Pass

User clicks on track within the returned search list.	Mini player is displayed, and audio begins to play.	Pass
Explore page tab bar button is clicked.	The artist selection page is displayed, if first time clicking the explore page.	Pass
User clicks artist within the artist list that is displayed.	Toast notification is displayed and behind the scene averaging algorithm occurs.	Pass
User enters search criteria into search bar on the artist selection page.	Results matching the search criteria are displayed in the list.	Pass
User attempts to close the artist selection page without selecting at least one artist.	An error alert is displayed, prompting user to make at least one selection.	Pass
User clicks preview button on card within the explore page.	Audio from the JSON file is begins playing.	Pass
User clicks pause button on card within the explore page.	Audio from the JSON file is paused.	Pass
User clicks dislike button on cards within the explore page.	Card is removed, and track attributes are ignored. Toast notification confirms action.	Pass
User clicks like button on cards within the explore page.	Card is removed, and track attributes are added to the averaging algorithm. Toast notification is displayed to confirm action.	Pass
User likes five tracks within the explore page.	User is taken to the recommendation page, which displays tracks of a similar nature based on the average tempo from the tracks liked on the previous page.	Pass
User clicks track within the recommendation list.	Mini player is displayed, and audio playback begins.	Pass
User clicks navigation bar button in the top right.	Action sheet is displayed allowing user to change the criteria use to sort tracks.	Pass
User clicks energy in action sheet.	Average energy is used to sort the list of recommended tracks.	Pass

User clicks dancability in action sheet.	Average dancability is used to sort the list of recommended tracks.	Pass
User clicks tempo in action sheet.	Average energy is used to sort the list of recommended tracks.	Pass

Table 4: Mobile Application Functional Use Case Testing

11.2.2 Non-Functional Testing

Some non-functional requirements were also stated in the requirement specification. This will be compared with the complete product to ensure they were achieved.

Stability: The application was tested within the web browser, as well as on an iOS device. Both platforms were stable, and the outputs were predictable each time. At no point did the final prototype crash on each of the devices.

Availability: As most of the data within the application is stored on the database the application is 26MB in total. This could be considered large for a mobile application. However, as some of the data for this proof of concept needs to be stored locally, this increases the size.

Portability: The author was able to test the application within the web browser as well as on an iOS device. However, testing on Android was not possible as the author did not have access to one of these devices. However, if the Android installation process is anything like the iOS process, the application should compile and install easily.

Usability: As the application only contains a few pages and follows models like that of Spotify and Soundcloud, users should be able to use the application effectively. However, this is proven by user feedback in the following section.

11.2.3 Beta Testing

Beta testing was conducted under the supervision of the author as code signing was required for the application to run, which some users may not have been familiar with. The testing was completed on an iOS device with five participants taking part.

The following questionnaire was created using Google Forms, *Google, 2018,* to capture the users' response after using the mobile application.

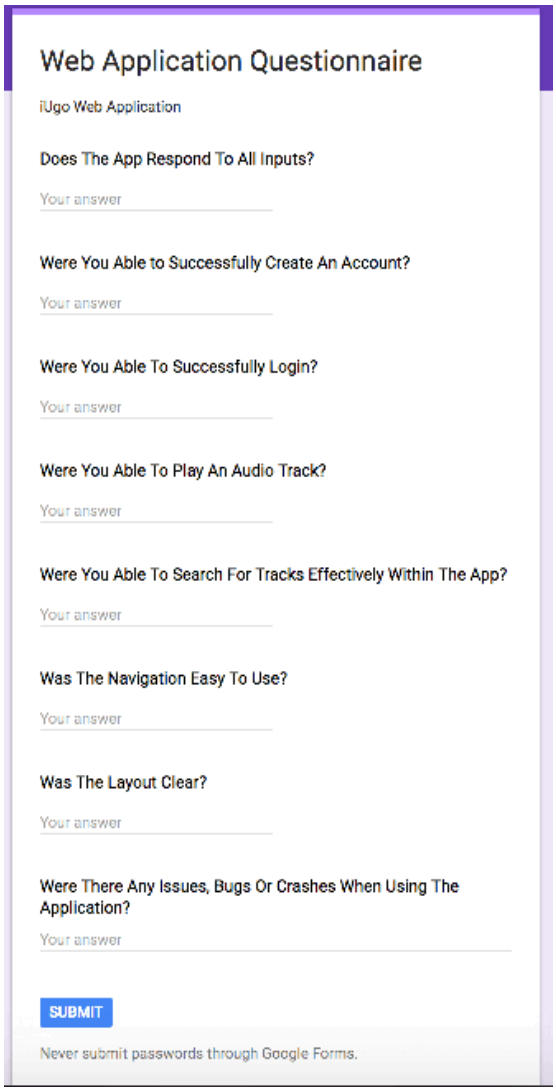

Figure 221: Mobile Application Questionnaire

The following *figures* show the users response to the questionnaire.

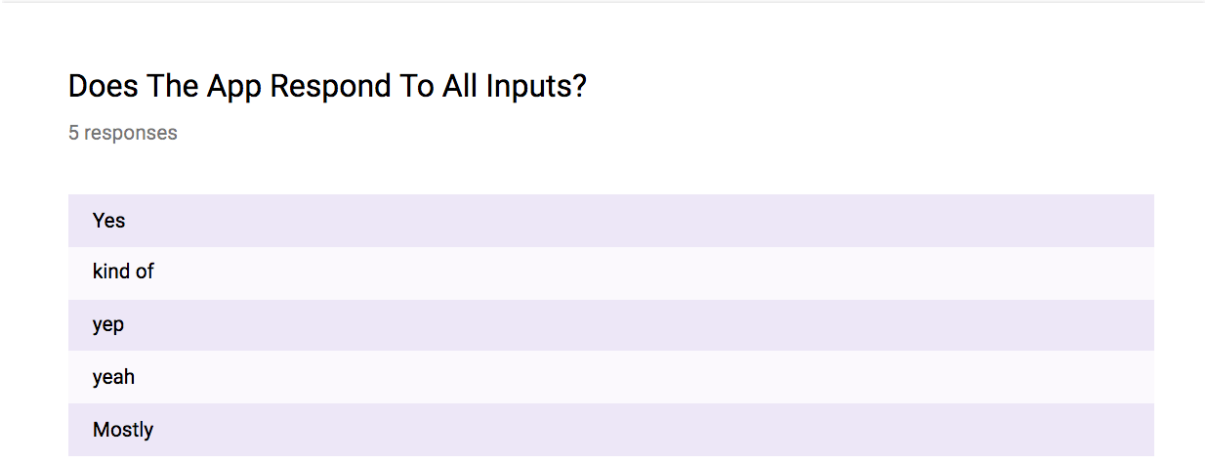

Figure 222: Does The App Respond To All Inputs Response

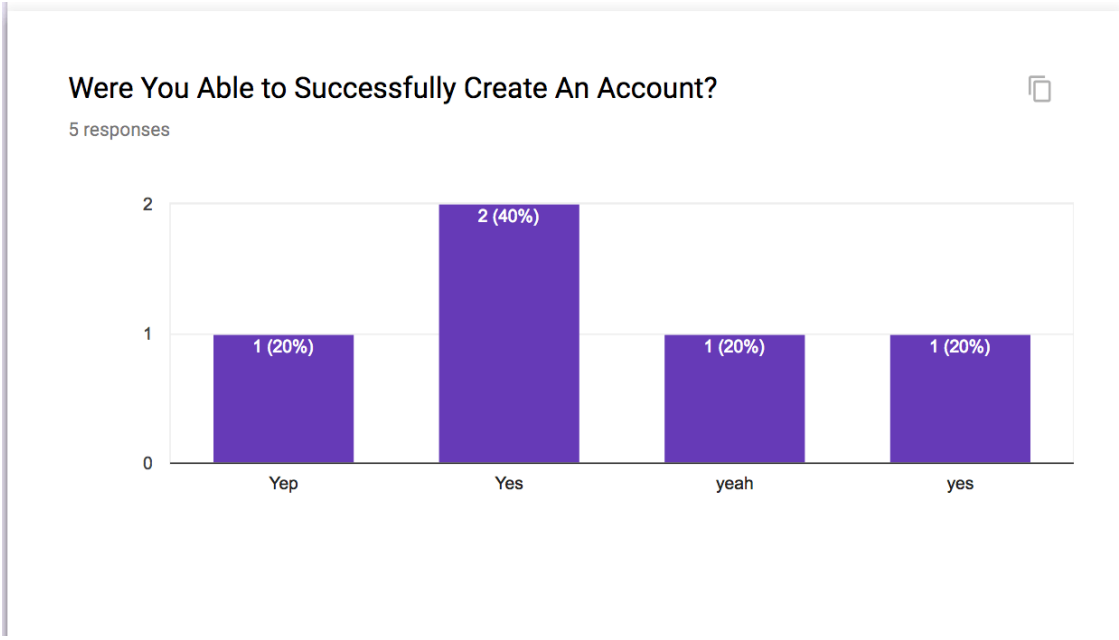

Figure 223: Were You Able To Successfully Create An Account Response

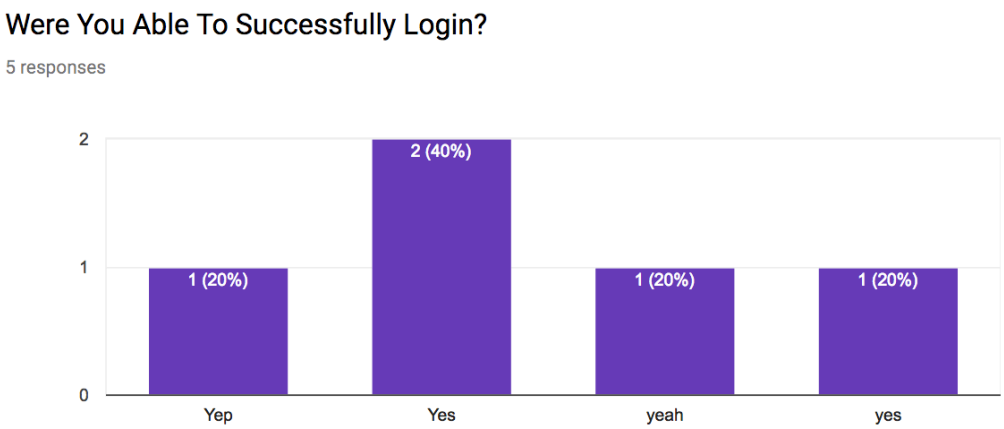

Figure 224: Were You Able To Successfully Login Response

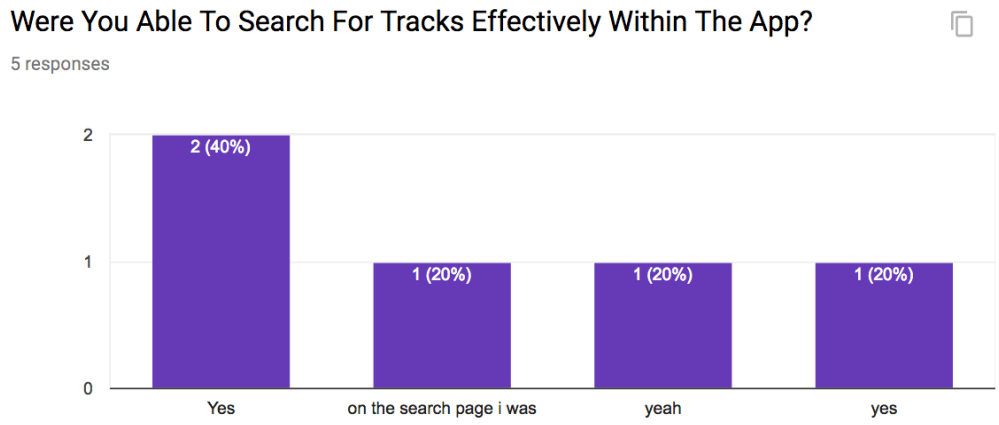

Figure 225: Were You Able To Search For Tracks Effectively Within The App Response

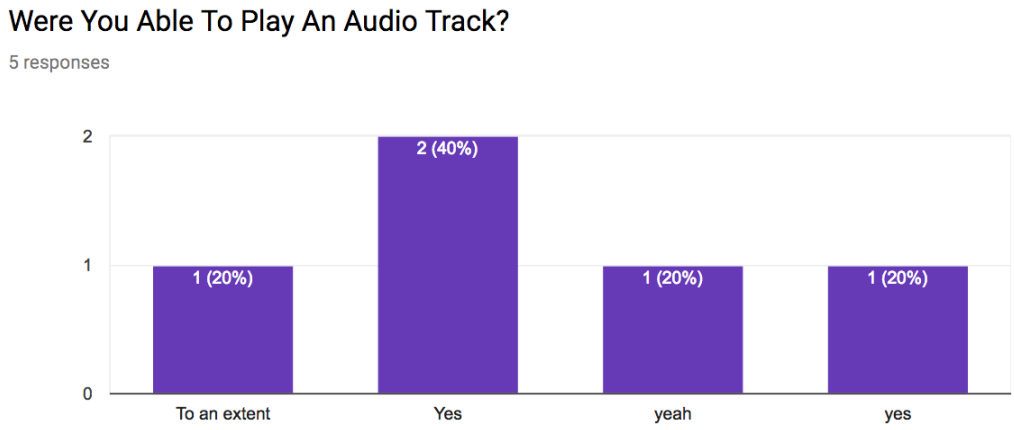

Figure 226: Were You Able To Play An Audio Track Response

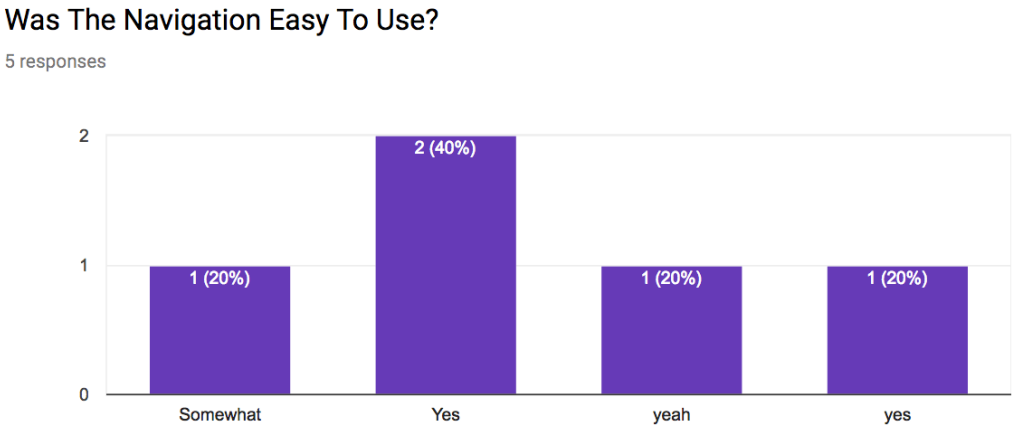

Figure 227: Was The Navigation Easy To Use Response

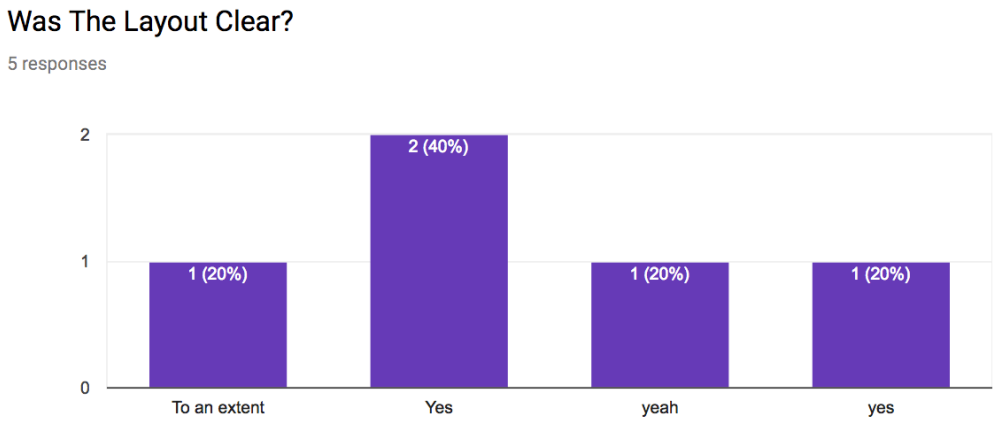

Figure 228: Was The Layout Clear Response

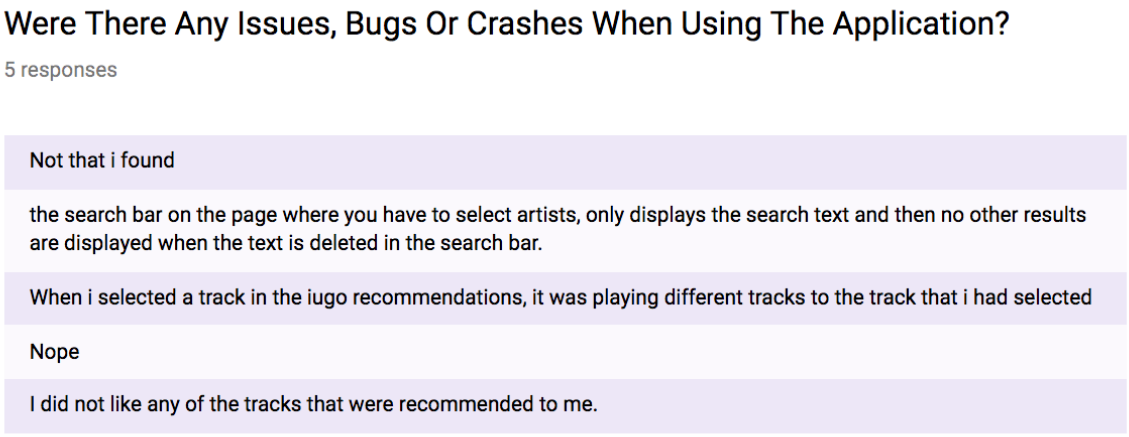

Figure 229: Were There Any Issue, Bugs Or Crashes When Using The Application Response

11.2.4 Beta Testing Results

As can be seen by the user response within the questionnaire, the application is easy to use and relatively straightforward. However, as can be seen if *figure 229* some users experiences some issues with certain features.

11.2.4.1 Fixing The Search Bar Issue

As explained by the user in the beta testing, the search bar was only displaying the first result and then not displaying any results after that. This was due to the array not being reset when a character was entered or removed from the search bar. Therefore, the data that was in the array was the previous search criteria. *Figure 230* shows the code needed within the search function to achieve this, which resets the JSON data within the array when a character is entered or removed.

```
searchQuery(param : any) : void {

let val : string = param;

this.dataFinder.getJSONDataAsync("./assets/data/artists.json").then(data => {
 this.SetQueryOptionsData(data);
```

Figure 230: Search Bar Character Rest Code

11.2.4.2 Fixing The Recommended Track Playlist

As explained by a user in *figure 229,* when clicking on a recommended track on the recommendation page, the track that began playback, was not the track that was displayed.

This was due to the tracks within the array, being taken from the tracks directory on the database. As the tracks directory on the database contains all the available tracks, all the available tracks were being used as the objects within the array. When the index for the object in the array was then called, it called the corresponding object, which would have been the first track in the tracks directory. Therefore, the tracks in the array, needed to be the tracks that were narrowed down by the query, and not the tracks in the tracks directory.

This was achieved by ensuring the query was performed first and the response was subscribed to. Once subscribed to, the response is stored in an array, along with its attributes. *Figure 231* shows the code used in the recommendations.ts file.

```
this.af.list('/tracks', {
  query: {
    orderByChild:'tempo',
  startAt: this.averageTempo
  }})
.subscribe(data => {
  this.tracksArr = [];

  data.forEach(pope => {
    this.allTracks.subscribe(all=>{
      all.forEach(alle=>{
        if(alle.$key==pope.$key){


   this.tracksArr.push(

    { src: alle.url,
      artist: alle.artistName,
      title: alle.name,
      art: alle.albumArt,
      preload: 'metadata',
      key:alle.$key,
      artistId:alle.artist,
      albumId:alle.album,
      album:alle.albumName,
      artistName:alle.artistName,
      tempo:alle.tempo,
      dancability:alle.dancability,
      energy:alle.energy
      }
```

Figure 231: Recommendation Audio Player Fix

12.0 CONCLUSION

The purpose of this project was to create a platform that allowed users to upload their own content, in the form of bootlegs, and listen to them within a mobile application. The purpose was also to be able to share bootlegs and find them easily using an easy to use and fun method. It was also important that this project would use an external source as a way of finding music that users of the service have uploaded. This was due to a high number of users using other services and not being too sure whether this service was right for them. To an extent, this was also designed as a way of being able to steal customers from the other services.

The literature review of bootleg material shows the want to listen to bootleg music, but major platforms seem to shy away from it. Although bootleg music is illegal, it is a form of expression for other artists, and should therefore be treated like an original track in the authors opinion. This in turn is used as the main justification for this project.

Evaluation of the mobile market showed that the Android platform has the largest share within the market. However, as the author is an iOS user, this is the preferred development platform. Without wanting to miss out on the potential for a large amount of growth and users, the author chose hybrid development which can cater for both platforms. The review of these technologies revealed the advantages and disadvantages.

The design chapter within this dissertation contains detailed wireframes of each section of the web application and mobile application. By doing so allowed the product to be effective and useful. This also helped with any issues that arose during the implementation process, as many of the features and designs had already been defined. As this was such a large project, it allowed the author to stay on track and focus on implementing the service rather than designs.

From the authors point of view, the entire project has been a success, and the product that has been created matches very closely to what the author set out to create. As the author has never used hybrid mobile application development, it was also a learning curve. The author does have experience with HTML, JavaScript and CSS, however, the Angular framework used within the hybrid development framework, turned out to be a lot more of a JavaScript derivative than previously thought. With that said, the author believes he has developed a full understanding on the Angular framework, which he hopes is demonstrated effectively within the implementation processes.

13.0 FUTURE WORK

One of the users in the testing mentioned that they did not like the recommended tracks, which can be seen in *figure 229.* As the author could only use a duplication of one of his original tracks, due to copyright reasons, this is naturally to be expected. Therefore, in the future the author hope to implement this with real music tracks that will hopefully match users with real tracks they may like.

Another feature the author would like to implement, is being able to use all three parameters within the query to the database. In the current application, the user is only able to sort the recommendation list by one of the three parameters (dancability, energy or tempo). This is due to the Firebase query only being able to query one item at a time, which in turn causes the restriction. It is thought that if all parameters can be used within a query to the database, recommended tracks that are return will better suit the user.

Another feature the author would like to implement, is the ability for the web application to automatically calculate the dancability, energy and tempo values. Currently, the user must specify the parameter values. This is due to the parameters not being able to be calculated without a complex algorithm being created, or the algorithm from Spotify. In the future however, the author would like to have created an algorithm for these values and have the web application calculate the values automatically. This in turn should lead to better recommendations made to the users, and stop users from miss-calculating the values maliciously.

Finally, the author would also like to implement a third-party service such as Spotify, so that the local JSON files are no longer needed. As the application is already set up to receive JSON data, this can simply be swapped for a JSON response from an API like Spotify. As explained previously, there was an issue with acquiring the correct authentication, when querying the Spotify API which led to the author using local JSON files instead. However, implementing the Spotify API would allow for cross platform recommendations, as first set out by the author to do.

References

Agile Business, 2018. *Moscow Prioritisation*. Agile Business, Agile Business. Available from: https://www.agilebusiness.org/content/moscow-prioritisation-0 [Accessed 30 April 2018].
Akveo, 2018. *Changing Color Scheme*. Ng2 Admin, Akveo. Available from: https://akveo.github.io/ng2-admin/articles/011-changing-color-scheme/ [Accessed 9 May 2018].
Akveo, 2018. *Create New Page*. Ng2 Admin, Akveo. Available from: https://akveo.github.io/ng2-admin/articles/013-create-new-page/ [Accessed 9 May 2018].
Akveo, 2018. *Ng2 Admin*. Akveo, Akveo. Available from: https://www.akveo.com/ng2-admin/ [Accessed 26 April 2018].
Akveo, 2018. *Sidebar*. Ng2 Admin, Akveo. Available from: https://akveo.github.io/ng2-admin/articles/015-sidebar/ [Accessed 9 May 2018].
Alex Moazed, 2016. *Why Soundcloud Will Be Worth More Than Spotify*. Tech Crunch, Tech Crunch. Available from: https://techcrunch.com/2016/01/24/why-soundcloud-will-be-worth-more-than-spotify/ [Accessed 25 April 2018].
Alexintosh, 2017. *ionic3-soundboard*. GitHub, GitHub. Available from: https://github.com/Alexintosh/ionic3-soundboard [Accessed 10 May 2018].
Amuse, 2018. *How to upload, a step-by-step guide to Amuse!*. Amuse, Amuse. Available from: https://support.amuse.io/hc/en-us/articles/115001794785-How-to-upload-a-step-by-step-guide-to-Amuse- [Accessed 24 April 2018].
Android Studio, 2018. *Android Studio*. Android Studio, Android Studio. Available from: https://developer.android.com/studio/index.html [Accessed 25 April 2018].
Angular, 2018. *Displaying Data*. Angular, Angular. Available from: https://angular.io/guide/displaying-data [Accessed 9 May 2018].
Angular, 2018. *NgForOf*. Angular, Angular. Available from: https://angular.io/api/common/NgForOf [Accessed 9 May 2018].
Angular, 2018. *NgModel*. Angular, Angular. Available from: https://angular.io/api/forms/NgModel [Accessed 9 May 2018].
Angular, 2018. *Template Syntax*. Angular, Angular. Available from: https://angular.io/guide/template-syntax [Accessed 9 May 2018].
AngularJS, 2018. *ngBindHtml*. AngularJS, AngularJS. Available from: https://docs.angularjs.org/api/ng/directive/ngBindHtml [Accessed 10 May 2018].
Anthony Bouchard, 2017. *Augment your Spotify Music app with Spodinhancer*. Angular, 2018. *Form Validation*. Angular, Angular. Available from: https://angular.io/guide/form-validation [Accessed 9 May 2018].
Apache, 2018. *Cordova*. Apache, Apache. Available from: http://cordova.apache.org [Accessed 25 April 2018].
Apple, 2018. *XCode*. Apple, Apple. Available from: https://developer.apple.com/xcode/ [Accessed 25 April 2018].
Arielfaur, 2017. *Ionic Audio*. NPM, NPM. Available from: https://www.npmjs.com/package/ionic-audio [Accessed 30 April 2018].
BesthookUpApps, 2018. *Tinder Review*. BestHookUpApps, BestHookUpApps. Available from: https://www.besthookupapps.net/tinder-review/ [Accessed 1 May 2018].
Bootstrap, 2018. *Bootstrap*. Bootstrap, Bootstrap. Available from: https://getbootstrap.com [Accessed 26 April 2018].

Braude, E. J. and Bernstein M. E., 2011. *Software Engineering: Modern Approaches*. 2nd ed. Hoboken, NJ: John Wiley & Sons, Inc.
Chris Tutorials, 2017. *Load Local JSON Files in AngularJS 4 w/ Typescript | Ionic Framework*. Youtube, Youtube. Available from: https://drive.google.com/file/d/1M5iC_4xeXQBmeVr1KwYmAaIq-7Lra3zJ/view [Accessed 10 May 2018].
Cory Rylan, 2015. *Angular ngFor syntax*. Cory Rylan, Cory Rylan. Available from: https://coryrylan.com/blog/angular-ng-for-syntax [Accessed 10 May 2018].
Creative Commons, 2018. *About The Licenses*. Creative Commons, Creative Commons. Available from: https://creativecommons.org/licenses/ [Accessed 24 April 2018].
Daniyalahmedkhan, 2017. *Android-Firebase-AudioPlayer*. Github, Github. Available from: https://github.com/daniyalahmedkhan/Android-Firebase-AudioPlayer [Accessed 10 May 2018].
F.Laurens, 2017. *Let's learn how to install and setup AngularFire2 4.0*. Medium, Medium. Available from: https://medium.com/letsboot/lets-learn-how-to-install-and-setup-angularfire2-4-0-135d72bb0a41 [Accessed 9 May 2018].
Google, 2018. *Create a Storage Reference on Web*. Firebase, Firebase. Available from: https://firebase.google.com/docs/storage/web/create-reference [Accessed 9 May 2018].
Google, 2018. *Firebase*. Firebase, Google. Available from: https://firebase.google.com [Accessed 26 April 2018].
Google, 2018. *firebase.database.Query*. Firebase, Firebase. Available from: https://firebase.google.com/docs/reference/js/firebase.database.Query [Accessed 10 May 2018].
Google, 2018. *Get Started with Firebase Authentication on Websites*. Firebase, Firebase. Available from: https://firebase.google.com/docs/auth/web/start [Accessed 9 May 2018].
Google, 2018. *Google Forms*. Google, Google. Available from: https://docs.google.com/forms/u/0/ [Accessed 10 May 2018].
Google, 2018. *Retrieving Data*. Firebase, Firebase. Available from: https://firebase.google.com/docs/database/admin/retrieve-data [Accessed 9 May 2018].
Google, 2018. *Saving Data*. Firebase, Firebase. Available from: https://firebase.google.com/docs/database/admin/save-data [Accessed 9 May 2018].
Google, 2018. *Upload Files on Web*. Firebase, Firebase. Available from: https://firebase.google.com/docs/storage/web/upload-files [Accessed 9 May 2018].
Google, 2018. *Use File Metadata on Web*. Firebase, Firebase. Available from: https://firebase.google.com/docs/storage/web/file-metadata [Accessed 9 May 2018].
Gov UK, 2018. *Intellectual property and your work*. gov.uk, gov.uk. Available from: https://www.gov.uk/intellectual-property-an-overview [Accessed 24 April 2018].
Gov UK, 2018. *Parody and pastiche*. Gov UK, Gov UK. Available from: https://www.gov.uk/government/publications/parody-and-pastiche [Accessed 24 April 2018].
Ian Sommerville and Pete Sawyer, 1997. *Requirements Engineering: A Good Practice Guide*. New York: John Wiley & Sons, Inc.
iDownloadBlog, iDownloadBlog. Available from: http://www.idownloadblog.com/2017/12/17/spodinhancer/ [Accessed 1 May 2018].
Ionic, 2018. *Deploying to a Device*. Ionic, Ionic. Available from: https://ionicframework.com/docs/intro/deploying/ [Accessed 9 May 2018].

Ionic, 2018. *Getting Started With Ionic*. Ionic, Ionic. Available from: https://ionicframework.com/getting-started/ [Accessed 1 May 2018].
Ionic, 2018. *Installing Ionic*. Ionic, Ionic. Available from: https://ionicframework.com/docs/intro/installation/ [Accessed 9 May 2018].
Ionic, 2018. *Ionic App Showcase*. Ionic, Ionic. Available from: https://showcase.ionicframework.com/apps/top [Accessed 25 April 2018].
Ionic, 2018. *Ionic Framework*. Ionic, Ionic. Available from: https://ionicframework.com [Accessed 25 April 2018].
Ionic, 2018. *Ionic Generate*. Ionic, Ionic. Available from: https://ionicframework.com/docs/cli/generate/ [Accessed 9 May 2018].
Ionic, 2018. *IonicPage*. Ionic, Ionic. Available from: https://ionicframework.com/docs/api/navigation/IonicPage/ [Accessed 9 May 2018].
Ionic, 2018. *ModalController*. Ionic, Ionic. Available from: https://ionicframework.com/docs/api/components/modal/ModalController/ [Accessed 10 May 2018].
Ionic, 2018. *NavController*. Ionic, Ionic. Available from: https://ionicframework.com/docs/api/navigation/NavController/ [Accessed 9 May 2018].
Ionic, 2018. *NavParams*. Ionic, Ionic. Available from: https://ionicframework.com/docs/api/navigation/NavParams/ [Accessed 10 May 2018].
Ionic, 2018. *Tabs*. Ionic, Ionic. Available from: https://ionicframework.com/docs/api/components/tabs/Tabs/ [Accessed 9 May 2018].
Ionic, 2018. *Toast Controller*. Ionic, Ionic. Available from: https://ionicframework.com/docs/api/components/toast/ToastController/ [Accessed 10 May 2018].
IT Knowledge Portal, 2018. *Software Development Methodologies*. IT Knowledge Portal, IT Knowledge Portal. Available from: http://www.itinfo.am/eng/software-development-methodologies/ [Accessed 30 April 2018].
Jave Bratt, 2016. *How to sync Lists from Firebase and display them in Ionic Apps*. Jave Bratt, Jave Bratt. Available from: https://javebratt.com/firebase-list-ionic-2/ [Accessed 9 May 2018].
Jeff Watts-Roy, 2018. *Agile Method Of Software Development*. Number 8, Number 8. Available from: https://number8.com/common-mistakes-using-agile-method/ [Accessed 30 April 2018].
Lory Gil, 2014. *The best Music apps of 2014*. iDownloadBlog, iDownloadBlog. Available from: http://www.idownloadblog.com/2014/12/21/the-best-music-apps-of-2014/ [Accessed 1 May 2018].
Mark Mulligan, 2017. *ANNOUNCING MIDIA'S STREAMING SERVICES MARKET SHARES REPORT*. Midia Research, Midia Research. Available from: https://www.midiaresearch.com/blog/announcing-midias-streaming-services-market-shares-report/ [Accessed 25 April 2018].
Methee Banjukeaw, 2017. *ionic MP3 music player*. Ionic, Ionic. Available from: https://market.ionicframework.com/starters/ionic-mp3-music-player [Accessed 10 May 2018].
Michaelangelo Matos, 2016. *A History of Music Bootlegs, Told Through 25 of the Most Significant Recordings*. Vulture, Vulture. Available from: http://www.vulture.com/2016/11/25-historically-significant-bootleg-recordings.html [Accessed 24 April 2018].

Microsoft, 2018. *Visual Studio Code*. Microsoft, Microsoft. Available from: http://code.visualstudio.com [Accessed 25 April 2018].
Mozilla, 2018. *Web Audio*. Mozilla, Mozilla. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Using_Web_Audio_API [Accessed 30 April 2018].
MySQL, 2018. *MySQL*. MySQL, MySQL. Available from: https://www.mysql.com [Accessed 26 April 2018].
Oxford Dictionaries, 2018. *Bootleg*. Oxford Dictionaries, Oxford Dictionaries. Available from: https://en.oxforddictionaries.com/definition/bootleg [Accessed 24 April 2018].
Paul Resnikoff, 2015. *YouTube Accounts for 40% of All Music Listening, and 4% of All Music Revenues*. Digital Music News, Digital Music News. Available from: https://www.digitalmusicnews.com/2015/10/09/youtube-accounts-for-40-of-all-music-listening-and-4-of-all-music-revenues/ [Accessed 25 April 2018].
PHP, 2018. *PHP*. PHP, PHP. Available from: http://php.net [Accessed 26 April 2018].
Raja Yogan, 2016. *Ionic 2 - Music App*. Youtube, Youtube. Available from: https://www.youtube.com/watch?v=S8NsG7jOgcc [Accessed 10 May 2018].
Shawn Lindsey, 2012. *The 'Chopped and Screwed' History of Houston Hip-Hop*. University Of Houston, University Of Houston. Available from: http://www.uh.edu/news-events/stories/2012/august/08282012UHMDJScrew.php [Accessed 24 April 2018].
Simon, 2017. *How to Build An Ionic App with Firebase and AngularFire 4*. Devdactic, Devdactic. Available from: https://devdactic.com/ionic-firebase-angularfire/ [Accessed 9 May 2018].
Statista, 2017. *Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017*. Statista, Statista. Available from: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/ [Accessed 25 April 2018].
Stavros Kounis, 2018. *How To Authenticate with Firebase and Ionic 3 — Email/Password and Google Sign-In*. Medium, Medium. Available from: https://medium.com/appseed-io/integrating-firebase-password-and-google-authentication-into-your-ionic-3-app-2421cee32db9 [Accessed 9 May 2018].
The Guardian, 2016. *Number-of-new-songs-since-last-month*. The Guardian, The Guardian. Available from: https://community.spotify.com/t5/Content-Questions/Number-of-new-songs-since-last-month/td-p/1455052 [Accessed 24 April 2018].
Thomas2017, 2016. *Ionic searcher with json array*. Ionic Forum, Ionic Forum. Available from: https://forum.ionicframework.com/t/ionic-searcher-with-json-array/42122 [Accessed 10 May 2018].
Thomasnu, 2014. *Calculate average from list of values in JSON*. Stackoverflow, Stackoverflow. Available from: https://stackoverflow.com/questions/25186243/calculate-average-from-list-of-values-in-json [Accessed 10 May 2018].
W3 Schools, 2018. *HTML 5 Audio*. W3 Schools, W3 Schools. Available from: https://www.w3schools.com/html/html5_audio.asp [Accessed 30 April 2018].
W3Schools, 2018. *Angular ng-if Directive*. W3Schools, W3Schools. Available from: https://www.w3schools.com/angular/ng_ng-if.asp [Accessed 9 May 2018].
W3Schools, 2018. *HTML Hidden Attribute*. W3Schools, W3Schools. Available from: https://www.w3schools.com/tags/att_global_hidden.asp [Accessed 9 May 2018].
W3Schools, 2018. *JavaScript JSON*. W3Schools, W3Schools. Available from: https://www.w3schools.com/js/js_json.asp [Accessed 10 May 2018].

W3Schools, 2018. *JSON*. W3Schools, W3Schools. Available from: https://www.w3schools.com/js/js_json_intro.asp [Accessed 26 April 2018].
Wave Maker, 2018. *Rapid Application Development Model*. Wave Maker, Wave Maker. Available from: https://www.wavemaker.com/rapid-application-development-model/ [Accessed 30 April 2018].
Will McCartney, 2017. *LENGOLAND: THE BUILDING BLOCKS OF BASSLINE*. Diss, Diss. Available from: http://diss.ltd/lengoland-the-building-blocks-of-bassline [Accessed 24 April 2018].
Wireframe.cc, 2018. *Wireframe.cc*. Wireframe.cc, Wireframe.cc. Available from: https://wireframe.cc [Accessed 1 May 2018].

Appendix A – Approved Ethics Checklist

Bournemouth University

Research Ethics Checklist

Reference Id	18111
Status	Submitted

Researcher Details

Name	Jake Gregory
Faculty	Faculty of Science & Technology
Status	Undergraduate (BA, BSc)
Course	BSc Music & Audio Technology
Have you received external funding to support this research project?	No

Project Details

Title	Final Year Project
Proposed Start Date of Data Collection	21/11/2017
Proposed End Date of Project	18/05/2018
Supervisor	Andrew Watson

Summary - no more than 500 words (including detail on background methodology, sample, outcomes, etc.)

The original idea for this project was to create a platform that allowed users to create bootlegged material and upload it to a platform. It would then be distributed to other users across the platform, for free, without the permission of the original artist or record label. The music would then be combined with music from Spotify, using their SDK, giving the user a larger library of music (official and unofficial), and the bootlegger a platform to share their talent. This would have been achieved using some very sophisticated copyright loopholes, such as the parody and pastiche legislation or creating a platform that simply pointed the users in the direction of the bootlegged material without any file hosting involved.Unfortunately, after some careful research and speaking with Andrew Kitchenham, it has been brought to my attention that the proposed idea is not feasible without breaching several copyright laws, which would in turn be breaching the ethical code of practise.As a result, i have decided to create this project as a proof of concept. This will mean that the project will be created for the sole purpose of proving that such a platform is technically possible. As a result the project will not be released publicly, or be made public, at any stage during the development process. The project will be created solely for the use within my final year project assignment as a proof of concept. Therefore i will be continuing with the original idea, but instead, i will be keeping everything localised. However, during the development stages, I will be using an online development tool that manages the server side operations, such as file storage, which i will be using in order to store and retrieve audio. As a result, this audio will be live on a remote server and by uploading bootlegged material, i would be breaching copyright laws. To avoid this, whilst using the online development tool, i will be using "dummy" audio files which contain no audio data and original pieces of music that i have the rights to distribute, such as my own original creations.

External Ethics Review

Does your research require external review through the NHS National Research Ethics Service (NRES) or through another external Ethics Committee?	No

Research Literature

Is your research solely literature based?	No

Human Participants

Will your research project involve interaction with human participants as primary sources of data (e.g. interview, observation, original survey)?	No

Final Review

Will you have access to personal data that allows you to identify individuals OR access to confidential corporate or company data (that is not covered by confidentiality terms within an agreement or by a separate confidentiality agreement)?	No
Will your research involve experimentation on any of the following: animals, animal tissue, genetically modified organisms?	No
Will your research take place outside the UK (including any and all stages of research: collection, storage, analysis, etc.)?	No

Please use the below text box to highlight any other ethical concerns or risks that may arise during your research that have not been covered in this form.
Copyright as explained at the start.