School of Electronic Engineering
and Computer Science

Final Report

Programme of study:

Multimedia & Arts Technology

Project Title:

Reforming Social Music Spaces: A human centered approach

Supervisor: Gyorgy Fazekas

Student Name:

Mark G Brown

Final Year
Undergraduate Project 2018/19

Date: 22nd March 2019

Blank Page

Table of Contents

Abstract

Eighty-six percent of surveyed participants state that there is always music playing at their social events, listening to music is often considered an individual experience but it has a profound effect on our ability to communicate and self-express. Observations of several groups in these social contexts shows that there are several aspects of the current solutions that hinder social qualities. This paper takes these hindrances into consideration and devises a solution that supports enriched social interaction.

The methodology tackles the findings in to a presentable product which clearly has a positive effect on major areas in the social space, making music queuing an increasingly collaborative task; resulting in greater music discovery and diversity.

Some initial observations made on group listening gave insights into some subconscious behaviours but more importantly displayed how the product can be used in further studies as an analytical tool for deeper analysis of music choice and its effects on the community and much more.

1. Introduction

Throughout the duration of this project, an extensive investigation into physical social spaces has been conducted in an attempt to understand the effects music has on both the individual and the community within these physical spaces.

Many other academics have contributed work in this area (Liu and Reimer, 2008), (O'Hara et al., 2004), (Danielsen, Hagen, 2015), of which, the devised solution takes their learnings into consideration. These academics conclude that music has a profound ability to bring people together but also has consequences. The theme of effecting ones esteem is consistent in both the academic work and primary investigations that were taken out.

To fully understand the affects these environments have. A human centred approach and mindset was used. This ensures that no feature or design choice is made without evidence to suggest it would supporting a need or behaviour. Understanding the human behaviour used multiple investigation techniques including: literature reviews, ethnography, interviews and surveys.

The methodology followed a d.school design thinking approach that seeks to find stories from users and engage with them on a personal level to truly understand their needs and feel how they feel.

The product of this work is a real-time collaborative playlist app which allows guests to see what is playing, search and queue music, add music to their library and save the playlist in Spotify, while also retaining playback controls for the organiser. Currently the product is built for iOS.

A strong case is produced from this extensive testing of the product to display how it supports existing and new human behaviour. The supporting evidence is shown as five metrics; request percentage change, ratio of requests played vs skipped, fraction of users and attendees, music discovery percentage change and average time difference in a common task for new vs old solution.

With a vibrant image on how this product can improve the social environment, future work is evitable and clear.

2. Background

2.1 Academic Works

Other academic works have attempted similar solutions and all show positive results. However, the focus of these products tends to be on creating an online platform. The Social Playlist (Liu and Reimer, 2008) prototype highlights how “listening can bring friends closer together” in a physical shared space. If physical spaces mixed with music improves our communication abilities, then we need to support these communication channels in the solution. The social playlist does not support any further interaction within the solution, and it only shows who has ‘chosen’ what song.

Solutions that are designed for the physical space such as Jukola (O’Hara et al., 2004) provides a fertile ground for these communication channels and uses itself to prompt many different interactions between users. However, it found that the voting solution to choosing the music dominated the music listening.

> *“it was sort of distracting because we cared more about what we were voting for rather than listening to the song that had won the vote” (O’Hara et al., 2004).*

Jukola provides devices placed around a social space which in turn restricts their accesses and prevents anonymity (something which seems detrimental in some social contexts given some observations).

Both Social Playlist and Jukola mention how their solutions provide a minimal collection of music, Jukola is restricted by copyright law, while the user’s digital music collection constrains Social Playlist. Modern streaming services now dominate music consumption, Spotify itself has over one hundred and forty-one million users as of September 2018 (Spotify 2018). The three characteristics of music streaming experience defined by Hagen (Danielsen, Hagen, 2015) demonstrates how a modern-day solution which integrates Spotify (or some other service) is the best approach.

Both the individual and social listening experience has fundamentally changed. Music discovery is now a daily occurrence due to the abundance of music on one individual service. Social sharing via online networks is now the default way of sharing new music. These social interactions are a form of self-expression used to fundamentally fulfil the need for “love and belonging” (Maslow, 1943, 1954). We can see the need for self-expression in the research performed by O’Hara. To ensure that their songs were played some groups with similar tastes would form alliances in order to gain control of the majority vote.

“There were even examples of within-group strategic voting in order for an individual to gain control over his/her group’s voting resource.”
(O’Hara et al., 2004)

Social Playlist (Liu and Reimer, 2008) is a solution which combines the individual and social listening over a network. While this solution works well as discussed in the conclusion section, there are many cases in which the social playlist fails to consider the differences in the individual’s context. Social Playlist works based on a user’s selected context and picks music from a collection of songs that is pre-defined for that context.

Liu and Reimer chose this approach as their initial observations show that the “music choice is itself related to the context”. If the user activity varying in context, due to the nature of the prototype being a “mobile experience that is not tied to any specific location”, then the collection of users would have to match in context for the solutions song choice to be suitable for all the listeners — for example, the gym vs working in an office. One context could be considered inappropriate for the other.

O’Hara’s research displays that “all types of music have certain connotations and physical characteristics that suggest appropriate ways to behave” (O’Hara et al. 2004). Therefore, the correct music choice for the context is crucial for creating and breaking a mood. The proposed solution in this paper will reflect this observation.

2.2 Non-Academic Competition

Subtv Playlister (SubtvU, 2018) is a non-academic solution to social music spaces, presents some useful features but focuses on being a multi-user music player while omitting social factors. Subtv does not seem to acknowledge the need to organise chosen music. Instead, subtv lets the music play in the order chosen, when testing, subtv presents a problem where one user could choose a whole album, and another user can then follow up and choose a single song.

The resulting order means that the second user has to listen to an entire album before they hear their choice. The initial observations made in the following chapter describe that music the audience does not enjoy resulting in them leaving the space, especially if the social area is public. Subtv’s play style increases the risk of the users disliking picked songs by potentially subjecting users to multiple disliked songs one after the other.

Spotify’s collaborative playlists are a popular choice for informal social events. These playlists are identical to Spotify’s playlist system, but multiple users can add to them. The issues with collaborative playlists originate when using these playlists in a real-time environment. Once playing, new songs added to the playlist will not be added to the queue. This observation is not surprising as the original purpose of the collaborative playlist was not for real-time social events. However, as Spotify users look for a solution

to real-time collaborative playlists, this workaround currently is the best solution within Spotify.

The supporting academic work talked about above displays some common hypotheses. Liu and Reimer (2008) conclude that “on-the-fly manipulation of the playlist is crucial. Users should be able to add new songs, use predefined playlists or change song associations”. This statement along with previous comments indicate that the context of an event controls the chosen music.

Reflecting on O’Hara’s work on Jukola, the designed solution should be entertaining but not intrusive to the user’s current activity. Ensuring that the system is available on each user’s device is essential to ensuring system accessibility.

Given Danielsen’s comment that “the given streaming service, via the smartphone, attaches itself to the listener, often literally, which makes the practice of using music as an accompaniment to daily life more flexible than ever” (Danielsen, 2015) we can assure that using users’ smartphones connected with a streaming service is an accessible and affordable solution.

2.3 Technical Background

2.3.1 Music Libraries Sources

Spotify is the chosen service for this product, it has a detailed and flexible web API (Spotify, 2018) which is used to find, play, control playback, create and add playlists plus much more. This API has a rich and detailed documentation with built in tools to test the API calls, this permits analysis of the response format which supports implementation planning.

Other Spotify Web API recourses include example code and demos and tutorials. This makes the Spotify Web API a perfect solution for the product. From reading the references alone a short list of reference end points can be made. For example, the player endpoints would allow control over the playback, volume and transport control (next, previous, seek), this would support the most basic of functionality in the product.

Additional useful API calls include: creating and following playlists, get recommendations (songs based on a defined set of attributes) and retrieving audio feature for a track (including tempo, key & time signature). With just these endpoints in mind a vast collection of feature can be derived. The recommendation call will allow the filtering of tracks for a particular context. Using the semantic markers such as ‘dancability’ and ‘energy’, a set of tracks can be derived that fit a certain context. These markers will be experimented with to see what attributes define a given context before this is implemented into the product.

The create and follow playlist endpoints can be used to simplify the social music sharing post event. By automatically logging each song that was played and creating a playlist from it, the guests can refer back to it in their Spotify app after the event.

Retrieving audio features opens up a huge pool or abilities, from the analysis of every song in the Spotify library an algorithm can be derived to select songs for an abundance of uses. Given an understanding of musical orientation preference we can manipulate the order of the chosen songs to create a progressive order.

Spotify is not the only accessible music streaming resource available. Apple Music, another popular streaming service, provides an Apple Music API (Apple, 2018) as well. This API is well documented as well but requires a developer account to fully use the features. To mitigate unnecessary costs this API will not be used.

Before discovering the Spotify audio features endpoint other API's were analysed. Superpowered Audio SDK (Superpowered, 2018) is a Music analysis tool for BMP Detection, Key Detection and more, this solution was a perfect candidate for developing features based on audio analysis but is no longer required as Spotify can provide all the functionality and reduce the computational needs of the product.

2.3.2 Storage and Mobile Infrastructure Management.

Spotify does not provide the product with everything it needs to succeed, maintaining what music is queued to be played is at the core of the product, a feature that Spotify API does not support. To manage this and other details such as the current session etc. the product will need to use a database that can store this data. Google's Firebase is a comprehensive solution that offers a vast range of products, including; Authentication, Cloud Storage and Real-time Database. These features allow comfortable setup of infrastructure within the product.

By testing the authentication and real-time database features a comprehensive understanding of the required knowledge was gathered. This solution is a perfect addition to the products collection of technical resources. Firebase also handles scalability, with an associated cost of course, which allows the product to be successful with not many repercussions.

The nature of an event allows us to remove un-useful data such as the Spotify Auth tokens (required for each use who wishes to access their Spotify account), this will reduce the amount of data that is stored overall making the product easier to maintain.

The choice of using firebase was made when comparing to common competitor Amazon Web Services (AWS), in contrast AWS was very complicated to grasp and offers a vast collection of tools, most of which wouldn't be necessary. Previous experience with Firebase makes it the most suitable solution.

2.3.3 Platform and Language

To ensure the best possible outcome, familiar technologies and knowledge should be exploited. However, the project is a learning experience which means that it should be used to improve a skillset. Swift is a programming language which is familiar enough to realistically achieve a goal but would require learning extra skills.

Swift is an Apple specific language reducing the audience to iPhone and iPad users. While this may seem restricting, it is the best solution to provide the best product. Other solutions which include iOS and Android devices were too unfamiliar and require extra learning.

React Native uses the React JavaScript library to simultaneously build native apps for multiple target OS's. This approach in the past has been confusing and troublesome taking up to a week to achieve the simplest of tasks. By avoiding this solution, a knowledgeable approach can be taken.

Swift has a comprehensive documentation of which is integrated into the IDE (XCode), this promotes rapid definitions of unfamiliar methods and classes. The Swift community is also very active, this is proven in the amount of open source example projects and help documents around the web, including tutorial videos. This will allow any troubles that occur during development to be overcome quickly. A collection of operations is available in the Swift standard library, set operators such as 'intersection' can be easily used in one line. For example;

```
let userSongs: Set = ["Stay Lost", "Why", "My Old Man"]
let otherUserSongs: Set = ["Did You See", "Stay Lost"]
let commonSongs = userSongs.intersection(otherUserSongs)
print(commonSongs)
// Prints "["Stay Lost"]"
```

Figure 2.1 *- Code sample for finding common items in an array.*

This example displays how we might be able to easily find relationships between users' music collection, which could be used for suggesting music to the session. This kind of functionality make Swift a good choice, adding experience of the language on top of this turns Swift in to the best choice.

2.3.4 Architecture & Structure

Swift libraries encourage the adoption of a variation of architectural patterns that are commonly found in well devised software solutions. The fundamental architecture pattern that will underpin the entire project is the model, view, controller pattern (or MVC).

This architectural pattern “separates the functional layer of the system (model) from the two aspects of the user interface, the view and the controller.” (Lethbridge, Langanière, 2005). This pattern increases component cohesion, reduces coupling and increase reusability. This pattern is also well known for its flexibility and testability, the reduced coupling also means that the UI can be tested independently from the rest of the application making testing methods such as unit testing much more accessible.

As common in most applications, the singleton object could be used in many cases where only a single instance of an object should exist at any one time. For example the sessions should only exist once within the app and the details should also be accessible from multiple parts of the application.

With the mention of real-time features above, it seems only fitting that the app made use of the observer pattern. With the observable object being the datastore, when this object gets updated it notifies the observer. Allowing us to effectively pass data between the model and the view sections of the solution.

Delegates and Protocols are another common occurrence throughout the swift library. A vast amount of the objects and interface elements implement similar behaviour (and therefore methods), it does not make sense to duplicate this behaviour nor does it make sense for these classes or UI elements to be subclasses of another class. In this situation a delegator can indicate a collection of methods and behaviour that is readily available to the delegate. A common example of this pattern in swift is when using UITextField. To manage the behaviour of the field and the associated keyboard the delegator has a wide range of methods that can be implemented. These range from UI events such as when the field is tapped to functions that notify the text field what to do when the text field reaches a certain length.

```
@available(iOS 2.0, *)
Optional func textFieldDidEndEditing(_ textField: UITextField)
```

Figure 2.2 *– An example of delegate method for UITextField.*

By making use of these patterns an intelligent solution to the problem can be produced that encourages code reusability and is prepared for further development beyond the project.

3. Methodology

The inherent human focused nature of this project indicated that human qualities have a huge impact in the way the solution is devised. This project undertakes the human centred design methodology along with more design centric approaches such as design thinking. As a result, the project followed these basic phases/steps:

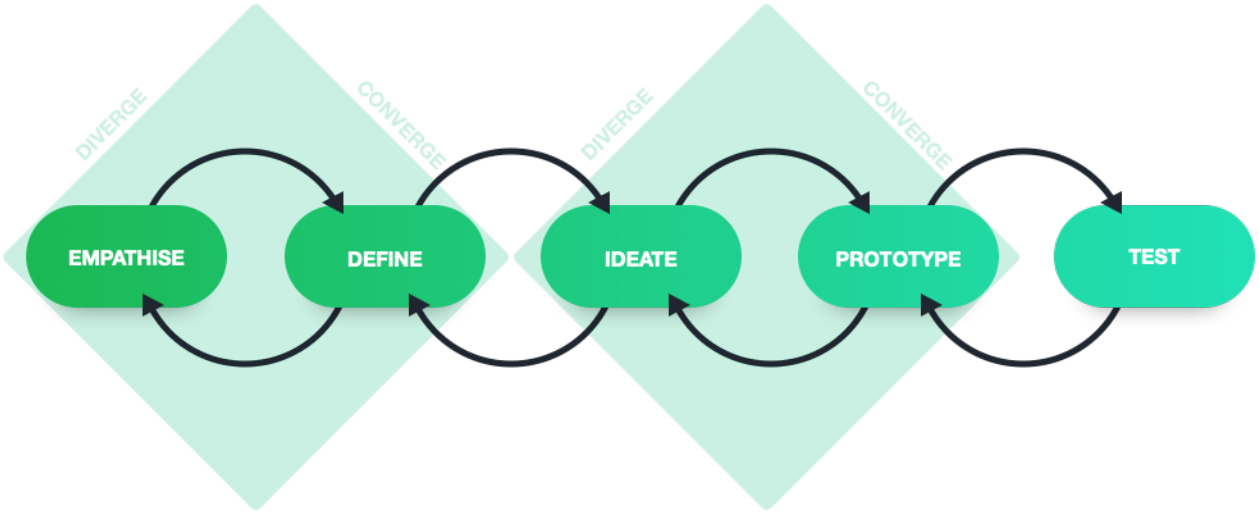

Figure 3.1 *Diagram illustrating the methodology*

3.1 Empathise

Beginning the project with an open mind is essential to this project. Learning what the solution should be based only on research and observations is a critical factor in creating a useful product. During this Empathise phase; Interviews, observations, surveys and background research was conducted to gain a holistic understanding of the problem. All of which each is spoken about independently elsewhere.

3.2 Define

Too much divergent thinking in this part of the process creates an abundance of data, of which can be difficult to manage in such a small project. To control this, the mindset switch from divergent to convergent thinking is critical. This part of the projects lifecycle collects all the data to form problem statements, journey maps, personas and how might we questions. These deliverables will stimulate creative thinking and present problems in multiple ways to ensure the problem is fully encapsulated.

3.3 Ideate

This next phase uses the deliverables from the define phase to devise solutions. As the problem has now become human centred with the how might we questions; solutions can be developed that reflects these needs. The ideation phase mainly consists of brainstorming sessions such as ‘space and saturate’ and sketching out ideas. We then look at these ideas and start narrowing down to features and some specification. At this stage no idea is a bad idea.

3.4 Prototype

The prototype phase took these sketches and transformed them into a testable prototype. Many iterations of this phase took place between testing and prototyping. The main three outcomes were:

- Low Fidelity Mock-up
- Minimal Viable Product
- Working Prototype

The deliverables from this phase include: visual designs, software specification documents and codebase milestones. Throughout the technical prototyping parts of the project, the software specification from earlier prototypes were carried across and from which, code tasks were identified. These were then tracked with a Kanban board tool such as Trello and Jira.

Tracking these tasks created a clear picture of what was needed to make each feature work. These features were referred to in Jira as ‘Epics’. Motivation and focus was easily maintainable using these ‘Epics’ as the large (and daunting) task of creating the designed product was broken into manageable pieces.

3.5 Testing

The final phase involved testing what was previously created. Testing of designs and code was conducted over the project lifecycle to ensure issues and bugs were found in a timely manner and managed accordingly. The general results of the tests were used to inform the next iteration of prototyping. The testing methods included:

- Usability Testing
- Black Box testing
- Coverage Testing
- Field Testing

Deciding which testing methods to use was a difficult decision. Preece, Rodgers & Sharp (2015) give a great insight into the benefits of using further user centred studies in the testing phases. Field or ‘in the wild’ testing often presents unexpected behaviour and factors, allowing data retrieval that would not be possible in a lab environment. However, field testing can be messy and difficult to record. For this reason focused usability testing was also used to create controlled and measurable tests with user.

This variation of methods ensures all levels of behaviour is captured. Black box and coverage testing also helps to reduce the amount of bugs and defects within the product.

This segmentation of mindsets allowed focus on the problem, the solution, the design and the implementation all independently of each other. This approach resulted in a product that understands the human experience and caters for it.

4. Findings & Requirements

The academic works discussed in section 2 give a thorough overview of what has already been achieved; however, some of the work that is undertaken in the physical, social space omits human behaviour. This factor has the potential to impact the success of the solution. To ensure that the project goals align with the users we must ensure we understand the needs and behaviour. In this project, a triangulation of methods is used to extract a source of truth. “Triangulation is the process of combining several different research methods to illuminate on area of study” (Visocky O’Grady 2017). This triangulation is achieved by collecting user data in three ways; Surveys, Observations and Interviews.

4.1 Survey

Initially, a survey was distributed to 100 people. The survey included questions such as;

- How often do you find yourself in a social situation where music is playing?
- How is music controlled?
- What is good and bad about this setup?

The full list of questions can be found in the supporting material. The response from this survey showed that social situations make use of mobile phones (78%) or tv’s (31.5%) to play music the most, alongside this, a majority 65% of these responses claim that Spotify is the preferred media/service.

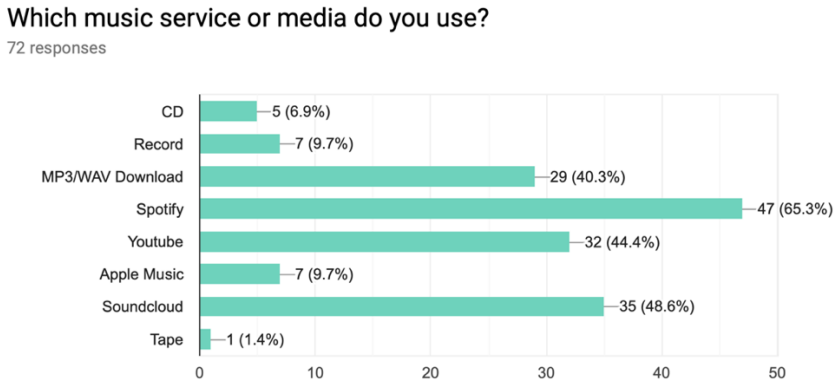

Figure 4.1 Graph displaying the distribution of preferred media/service

Demographic data such as age, gender, employment status and marital status was collected and analysed but showed no trends between the number of social situations and any demographic. The most popular age and occupation were 18-21 and student. This demographic is a reflection on the conventional communication channels of which the survey was distributed (amongst family and friends, social media and online communities).

Overall the data informs which media and platforms are widely used in social situations and most suitable for the solution, but no other relevant findings.

4.2 Observations

Finding insights into why these solutions are the most used required more descriptive data. To obtain such data ethnographic observations were performed. Several observations were made on different social groups. Each group used a different mechanism of music control; including: ‘event organiser controls music using a single device’ and ‘multiple guests using any available device’. Multiple observations were made about these groups, some of which allowed insight into problems that arise in the social music space and the attendee’s behaviour.

Coinciding with the observations made in O’Hara’s work, it was clear that some guests were using music as a form of self-expression. Several accounts of gaze were witnessed after a guest started playing a song; the guest would play a song and continuously look at an individual within the group awaiting a facial or verbal response.

The type of response was dependant on the song itself. Songs with ‘follow along’ vocals awaited a smile, while heavier, ‘dirtier’ songs awaited a look of disgust (the mark of approval in some genres). Verbal responses ranged from “Oh I love this one” to “Ah this track is sick!”. With these observations, it is clear that playing music to a group of people has some effect on an individual’s self-esteem.

An interesting alternative observation was disapproval upon playing a song or not knowing a song that has historical, popular or cultural meaning. The disapproval response would always result in an abundance of justifications for liking the song. Liu and Reimer found similar behaviour when testing The Social Playlist.

> *“He can select songs he likes moreover justify some selections which other people might interpret in an unintended way” (Liu & Reimer, 2008)*

Dependant on the willingness of self-expression, some listeners would prefer not to share their musical taste in fear of subjective criticism. This behaviour was observed by queuing music and then rearranging it to appear randomly in the queue. Again, the justification for the song was always apparent when the anonymous user was asked about their choice (on occasions where anonymity was unavailable).

One of the most exciting aspects of the observations was the actions that were taken to gain control of the music. In social situations, we usually see that using other people’s devices without permission is a violation of the non-explicit social rules. However, in these contexts, if the purpose is to change or control music, it is socially acceptable.

Other social norms that violate non-explicit social rules in non-music focused environments include; disrupting the currently playing and disconnecting devices from Bluetooth speakers, auxiliary connections or chargers to play music from another device, even without permission from the organiser. It seemed that once control of the music was acquired the attendee would try to maintain that control for as long as possible. These actions can be described by standing near the controls/device or locking/stowing away the device.

Some event organisers would distribute a playlist before the event to create a collection of people's music requests. In cases where this was observed playlists would alleviate some of the actions taken to gain control of the music. These lists were usually created with Spotify's collaborative playlists, allowing multiple Spotify users to add to a single playlist.

However, in some of the observations, the playlists were restarted during the event to refresh the contents as once playing new additions will not enter the queue unless restarted. As a result, a situation where songs were played multiple times occurred; this usually resulted in songs that had already been played once or twice being skipped by the controller.

During these events boredom was characterised as dull movements, frowning or soulless facial expressions, not engaging with other attendees or being sat down alone scrolling through their phone. This behaviour commonly appeared when the style of music changed in something other than upbeat music or music with personal/social reference.

An example of this was when an attendee changed the music halfway through a song; the song playing was an upbeat favourite house song that was currently in the dance charts to an indie pop band song that was also popular but had sad/depressing themes and a different overall tone. This particular scenario resulted in many people sitting down and pulling frowning faces. This resulted in the attendee who changed the music attempting to justify his choice of music with utterances like "wait until the chorus, it gets better" and "I know it is a sad song, but It is so good".

The most common situation that caused attendees to act as if they were bored was when the same style of music was played for extended periods; this seemed to be around 30 minutes of one single style. For example, a mixed group of guys were dancing while pop/rap music was playing. For the first 15-20 minutes they danced continuously, after 20 minutes they began to show signs of fatigue, and after about five minutes one of the males suggested getting a drink, of which the whole crowd agreed and followed him. They refreshed their drinks and continued to talk by the drinks area until the style of music changed.

This behaviour indicates that the activity of getting drinks was a result of the boredom that occurred from the consistent music style. Four to five other occurrences of this behaviour were observed, indicating that this behaviour is a common way to move on from repetitive music.

Behaviour classified here as ‘punching in’ was commonly observed. This action is where attendees play a song next or immediately (stopping the current song); usually, the song is somewhat relevant the previous song but occasionally is completely unrelated. The reasoning for this action seems to be because of the songs social significance or to fulfil a self-esteem need in showing off what they believe will be a ‘people pleaser’.

Throughout the observation process, music was seen frequently as the topic of conversation. It allowed for a common touchpoint to initiate a conversation between two people that were unfamiliar with each other. Utterances like “This song is great, have you heard it before?” or “Oh you like this song too?” were typical conversation starters. In some cases, this conversation could extend to find common interests.

An example of this was an observation of a male and female discussing the music. The male notices the girl enjoying the music and proceeds to ask her if she likes the artist (In this case the artist was Dillion Francis), the female responded positively stating that she saw him at a festival last year, which then formed into a conversation surrounding their festival plans for the coming summer.

For people with stronger relationships music could begin conversations that reflected on shared memories or experiences. Reflecting on times at school or last year’s freshers events were common themes. Some attendees displayed how they had associated a song with an entire holiday because the release coincided with the holiday dates.

Other common conversations observed during the observation revolved around: the artist playing or associated artists, artist or label social media activity, events and festivals. A vast amount of conversations were trivia based, for example, “did you know that artist x released an album with artist y?” or “when this band performs they use a prophet five synth from the 80’s”.

Services like Shazam were often observed to be used when attendees wanted to know the track metadata. This actions that followed usually resulting in the track being added to their music library.

In one case an observation was made where Shazam could not identify the song, attendees would then result in using google search to find the song, which was not successful. The use of Shazam would occupy the attendee for approximately one to two minutes; some attendees took their phone closer to the speakers to obtain the best results.

4.3 Interviews

The observations made, displays how the attendees behave. However, user interviews show insight into what attendees think. These interviews are useful to note whether they are aware of their behaviour. The responses can also support some of the conclusions made from observations which will help define problem statements and solution provoking questions.

The primary goal of the interviews was to understand how people experience music at events from a personal perspective and empathise with their pain points. The focus was specifically on problems with control in social music spaces. A few of the questions included:

- Can you tell me about the last time you had a good experience with music in a social space?
- How was the music that was playing chosen?
- Can you tell me about what you usually do directly after an event?

These interviews were semi-structured which allowed for discussions to vary and evolve; this also enables probing of particular responses. The 5 Why's method was commonly used to understand the motivation for a response. The initial questions were used to create a joint understanding of the purpose of the interview and understand how they use music in social spaces.

Following this, positive and negative experiences were discussed. When wrapping up, the participants were given the option to ask questions and discuss anything they thought would be useful to themselves or the project. Omitting terms such as app or streaming at the beginning of the interview ensured that there were no leading questions during the discussions.

Scheduling six interviews, three male, three female proved to provide an ample collection of findings. Due to scheduling issues, two of the female interviewees took place as a group. The group interview involved more discussion between participants, but some of the responses may have a been modified to fit in with the other participants perspective. Nevertheless, enough individual data was collected not to be affected by this side effect.

The participants were chosen based on their age and occupation. Using the results shown in the survey, the collection of participants displays qualities that coincide with the most popular age range and occupations (18 – 25 and a Student or Full-Time Employee). These participants all had a range of musical usage; some listened to music for one hour a day; some listened all day non-stop. However, they all were confident that music would be playing at any events. These events ranged from house parties and gatherings with friends (or hanging out) to working in the office with 15 – 20 other co-workers and dinner parties with family. This range of contexts displays a critical insight that music is almost essential in social environments.

Many of the interviews had common themes that supported earlier observations. An event usually started with a playlist; four participants noted that these playlists are sometimes collaborative playlists that are made before the events, similar to what was observed earlier in the project. During discussions with the two female participants, they reflected on times that they arrived late, and missed the songs they put on the playlist.

The discussed setups were consistent with the data found in the observations and the survey; the most prominent was the ‘event organiser controls music using a single device’ setup, this setup was also the most commonly observed.

Participants shared that the setup would be subject to certain downfalls. For example, one participant highlighted a memory when they requested the organiser to play a song for them, although the organiser agreed, the song never played. The cause of this was the 'play next option' this meant that once several people had requested a song, the original request was lost four or five songs back, i.e. last come first served.

This setup also incurred some other difficulties in requesting music; one participant noted that to have a song played, the organiser had to approve that request, or it would not be queued. This issue was not commonly a problem as the participant and organiser said they usually have a mutual likeness for the music.

Many participants noted how they feel good when they are in control of the music, and those feelings of confidence arise when attendees show positive responses to the chosen music. One participant who is a hobbyist DJ explained that “the feeling is similar when DJ’ing in a club or bar”.

When asked about times when music played that participants liked or did not like, they mentioned that disliked music would not affect them if they are in a conversation but would stop them dancing. One interviewee stated that “if the songs bad I want to get out the room”. Alternatively, the positive memories would entice the attendees to start dancing with the possibility of interrupting their conversation if the song was ‘that good’.

Reflecting on positive memories invoked some interesting comments. Participants mentioned that events and moments that come to mind revolve around who they were with and what music was playing. One interviewee spoke of a memory where their family gather for a party. A result of the diverse age range, music from a collection of eras was played. The participant referred to these songs as “old bangers.” To acquire the status of “banger” the songs usually had memories associated.

Another reflection from a different participant noted that they have fond memories of social events where a vast array of music was played from different genres and eras. They described this kind of music as; songs or albums released when they were young, the music their parents listened to and music that was popular on nights out when they

turned 18. They also noted that many conversations that happened at events revolve around music, specifically the music that's playing and relevant to them.

Discussing negative memories was much more detailed than positive memories. The participants seemed much more passionate about the things that went wrong than the positive memories they had just discussed. A theme that occurred with all six of the interviewees involved events which made use of auxiliary connections.

Problems involved attendees fighting to claim ownership of the aux cable, unplugging other people's devices and leavening them unattended. In one instance an interviewee spoke of an event in which her iPod was lost/stolen because of this setup, causing her to lose an extensive library of music and hundreds of pounds worth of technology.

Another problem that was mentioned with this setup was the persistent style of music. While each person held possession of the music playing device, the style of music would become monotonous as their music library and personal taste are finite.

The issue raised here meant that music styles would be static; participants described behaviour consistent with what was observed when music become uninteresting.

The social environment was prone to abrupt silences and interruptions when the music devices used auxiliary connections. This interruption was most apparent at parties with loud music. One response noted how this could have a devastating effect on a conversation if attendees are talking loudly to compensate for the music. The silence can make conversation awkward, and it can be challenging to continue when the music resumes.

Inconsistent music quality was another side effect of this approach; interviewees claimed that some attendees jump straight to YouTube when looking for music. This choice often provides lower quality audio that streaming services like Spotify and some uploads are generally low-quality rips or recordings of songs. While this may seem slightly insignificant legal and performance/recording rights issues occur when using rips and unofficial recordings, which means that artists do not receive royalties or benefits from being played in social spaces.

The reoccurring statement from a few participants referred to how these issues had different levels of impact depends on the social context and the number of attendees. An example used was car journeys with friends vs house parties. A small environment such as a car was more subjected to random interruptions but was less impactful than in a large environment.

The most emotional response from the interviewees was when post-event activities were discussed. Finding songs or reliving the order in which a few songs were played was almost impossible to find and recreate. Two participants spoke of how they had used Shazam at a couple of events to find details of the songs playing and then added them

to their library. To find songs post-event google search was queried using the lyrics that the guest could remember. However, they noted that this was not very effective, and they usually just asked the host if they remembered.

4.4 Convergence of Methods

Upon reflection on the interview results, we can see reoccurring themes. These themes identify areas for improvement of the social music space. In order to fully understand and appreciate these themes, point of view statements were defined to state the correct design challenge clearly. Point of view (or POV) statements are tasks associated with the third stage of the design thinking model. They provide goal orientated insights which allow for effective ideation for potential solutions.

These statements are a composite of the needs and insights that have been gathered in the discussed research. These statements are narrow enough to allow focused generation of high quantity and quality ideas. The point of view follows a three-part composite. The user, the need of the user and what insight/s we have into this need and user. This need is then formed into a POV Madlib. The Madlib combines all three elements into a problem statement which fuels the design process. For example;

User	Need	Insight
Party Attendees	Variation in music regularly	Attendees get bored when listening to one style of music for extended periods of time.

Figure 4.2 An example point of view template.

“Party attendees need variation in music regularly because attendees get bored when listening to one style of music for extended periods” – POV Madlib example

To generate ideas from these statements the design thinking model usually turns these statements into questions. These questions are designed to provide a theoretical ground for which unbiased ideas and designs can form, completely free of implementation constructs and constraints. One of the most widely adopted techniques is to covert these statements into “How might we” questions. These purposefully ambiguous questions reframe the point of view into multiple questions, for example:

- How Might We create an environment with a consistent variation of music?

- How Might We discover the boundaries of what music should and should not be played?
- How Might We decide when the music style has become stale?
- How Might We prevent people from collectively choosing too much of the same music?
- How Might We allow people to decide that they want to listen to one type of music all the time?

These simple examples came from one point of view statement, collectively looking at all of these questions spark imagination and ensure that designs incorporate user needs.

"A properly framed How Might We does not suggest a particular solution but gives you the perfect frame for innovative thinking." – IDEO (n.d.)

The point of view statements and how might we questions were eventually used to create a range of personas. The collection of findings allowed detailed accounts of what a potential user might look and think. These personas were used throughout the rest of the project to maintain the human focus. Asking questions like "would Lilly use this feature?" or "Would Grayson understand how to do this?" promoted the human qualities and constraints they might provide.

Journey maps were also devised for some of the complex tasks; this demonstrated the pain points, emotions and touchpoints in an existing activity. This activity constructed a holistic view of the task and identified patterns and reoccurring themes that had a negative impact. Journey maps also identify the highlights to take from user activities.

With the entire collection of findings and analytical activities a complete human centred prototype can be generated. The complete collection of Interview questions, Point of View statements and How Might We question can be found in the supporting materials research folder.

Figure 4.3 Image of a journey map produced from research

5. Design

The Methodology chapter outlines a five-phase method for designing a solution. The collection of ‘How Might We’ questions, personas and journey maps are used to compose a collection of screen designs for initial testing with our participants, which eventually are turned into software requirements.

5.1 Ideation and Visual Design

Brainstorming how to solve the questions from the previous chapter is the first step taken to generate solutions. This brainstorm produced a collection of post-it notes with ideas some of which shared themes or patterns. The similar items are placed next to each other and then summarised with a heading.

Figure 5.1 *Image of space saturate and group activity*

With these ideas and features a collection of preliminary sketches can be generated. The collection of sketches varies in complexity and realism. Exploring all of these possibilities ensures that the best solution is found. Doing some necessary testing with these paper sketches show early on which ideas are good and poor; more about how these tests were conducted can be found in chapter 7 & 8 - Testing and Evaluation.

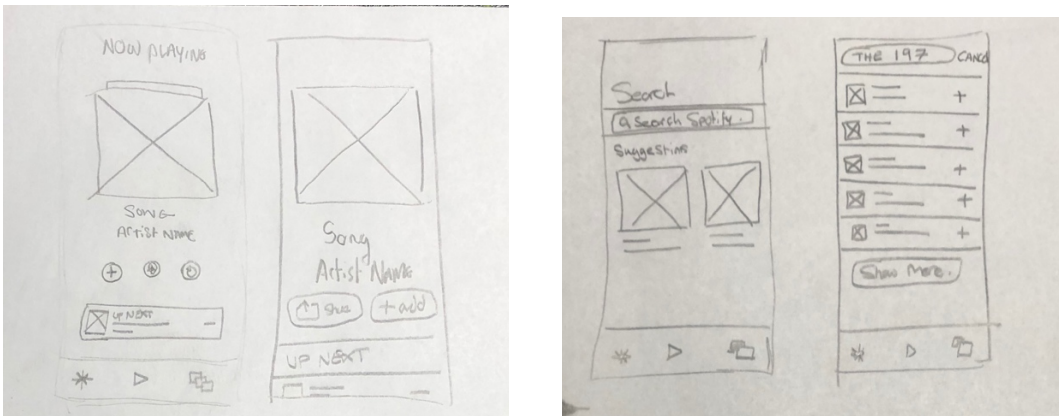

Figure 5.2 *Paper sketches for simple prototype*

Taking good ideas from these sketches, we can create some wireframes. These wireframes identify the layout and component sizing in the application, omitting details like content and colours brings focus to the elements mentioned above allowing precise and detailed observations when testing. From this, we also start to understand the technical software requirements.

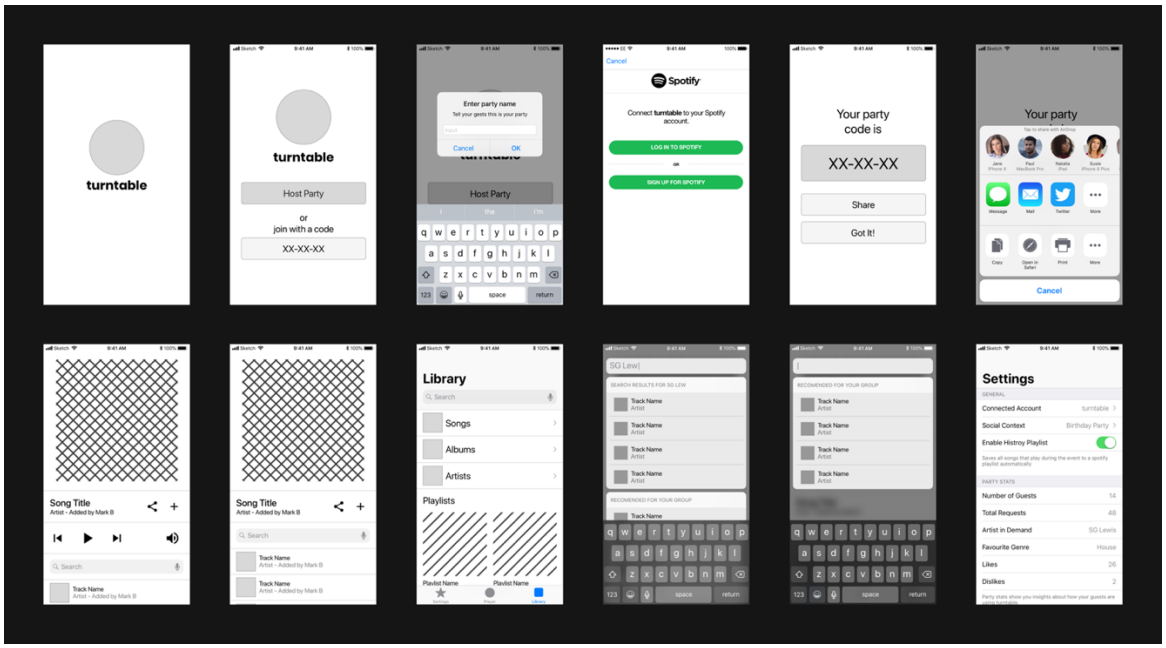

Figure 5.3 *Collection of wireframes created for testing*

To ensure that the design maintained consistency throughout the application; a visual language was produced. This language defined components such as colour, typography, button styles/states, iconography and image styles. As more high-fidelity screen designs were produced reusable components were also added to the visual language, this visual language is available with the complete collection of screen designs in the supporting material.

The design language was built with all three levels of design in mind (Norman, 2004). While the majority of products tend to apply one level more than the others; a genuine balance was attempted here. The visceral level leverages the users idea of beauty and attractiveness by using the contrast of dark backgrounds and vivid colours a strong visual attraction can be made. The gestalt laws are also employed throughout the design to ensure that the layout is even and visually stimulating.

The behavioural design aspect focused on the in app experience and the qualities of how it feels to use the app. What happens when you tap a button, how quickly does it respond. These were all elements that impacted this level of design.

Good reflective design invokes memories. Music is a large part of our daily lives and as seen in the earlier chapters, we have strong association between music and memories or parts of our lives. We can leverage these associations by strongly presenting artwork and the colours which they use. In this sense we can tap in to our users memory and present a design that promotes reflection and reminiscent.

While in the designing phase, further research was conducted about the Spotify API. This research displayed a few issues that made the wireframes and paper prototypes ineffective solutions. For example, Spotify requires each user to have a unique access token; sharing these tokens violates the terms of use. This limitation meant that the designs must ensure that all the users connect a Spotify account. The wireframes implied that users could join a session without a Spotify account; this is proved to be an unlikely scenario due to the terms of use constraints.

Taking these constraints into consideration a high-fidelity design was comprised. This design was built using Sketch and InVision. InVision allowed a quick and easy way to join the screen designs together and create a production quality mock-up of which could be used to test use cases with participants. These test cases included:

- Joining a session
- Setting up a session
- Searching for music and adding it to the queue
- Leaving a session
- Sharing a session code

These tests made it clear that some areas needed severe improvement and adaptations for different screen sizes. When reviewing the entire collection of designs and features, the product felt too ambitious. To reduce the feature set and increase the likelihood of success a, minimal viable product was defined. This idea, taken from the lean start-up methodology, is where a restricted feature set is introduced to validate the idea of the product by implementing the fundamental ‘core’ components of the product.

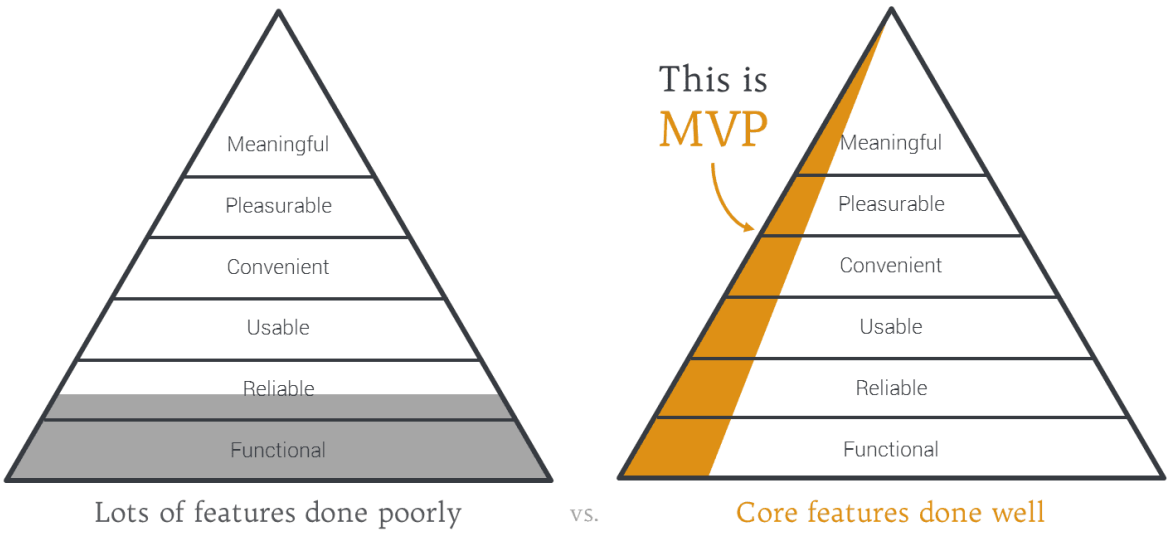

Figure 5.4 *MVP model as described by Brian Pagan (Pagan 2015)*

Gothelf & Seiden (2016) discuss the attributes of an MVP, noting that this approach requires a deep understanding of what needs to be learnt next. After the extensive research and design phases the hypothesis was clear, the next question was how do we test that hypothesis while minimising the resources needed.

This image (Pagan 2015) encapsulates the premise of MVP. To effectively figure out what features to include and exclude from this version the MoSCoW method was used. MoSCoW prioritises what the product; must have, should have, could have and will not have (at least this time). As there is no precise rationale for what constitutes a must-have feature, research from chapter four was consulted. Seeing which themes appeared commonly or seemed most important indicated which features should be considered core. Once this process was complete, the high-fidelity design was iterated on once more to produce the MVP.

5.2 Implementation & Structure

To convert the designs and visual language into a development tool, Zeplin was used. This app allowed the extraction of classes for colours and fonts directly from the sketch document. This feature made it easier to maintain consistency throughout the development process and also enable internal references to particular colours, such as seaFoamBlue (The repapering blue/green shade used for interactive screen elements). Another benefit was being able to see the layout properties such as padding and margins. The design document leads to producing a range of functional and non-functional technical requirements.

Req. No	Type	Description	Dependencies
1	Functional	Attendees must be able to access their Spotify account	Nil
2	Functional	Organisers must be able to create a session	1
3	Functional	Organisers must be able to generate a session code	2
4	Functional	Organisers must be able to share session codes	3
5	Functional	Guests must be able to enter session codes	3
6	Functional	Guests must be able to join a session	5
7	Functional	Attendees must be able to leave the session	6,2
8	Functional	Attendees must be able to browse music by artist, songs, playlists or albums	1

Figure 5.5 *A selection requirements, the entire collection can be found in supporting material*

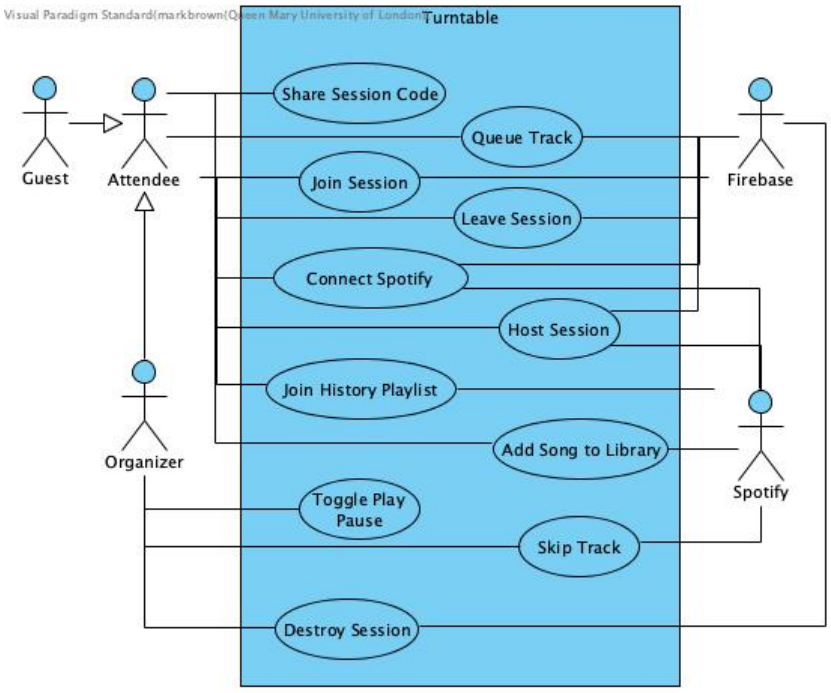

Figure 5.6 *Abstract use case diagram*

These requirements allowed the generation of a UML use case, class and sequence diagrams. These diagrams were used to lead the first portion of implementation.

The use case diagram (Figure 5.6) displays each action that a user could take and how they interact with other actors (in this case Spotify and Firebase). From this, we also begin to see functions that will be reused multiple times throughout a series of uses.

The class diagram shows where common software design patterns start to form. It also helps collect formal requirements that will make the application functional. The model objects in the full diagram (found in the supporting material) clearly describes the data, data type and any defaults values.

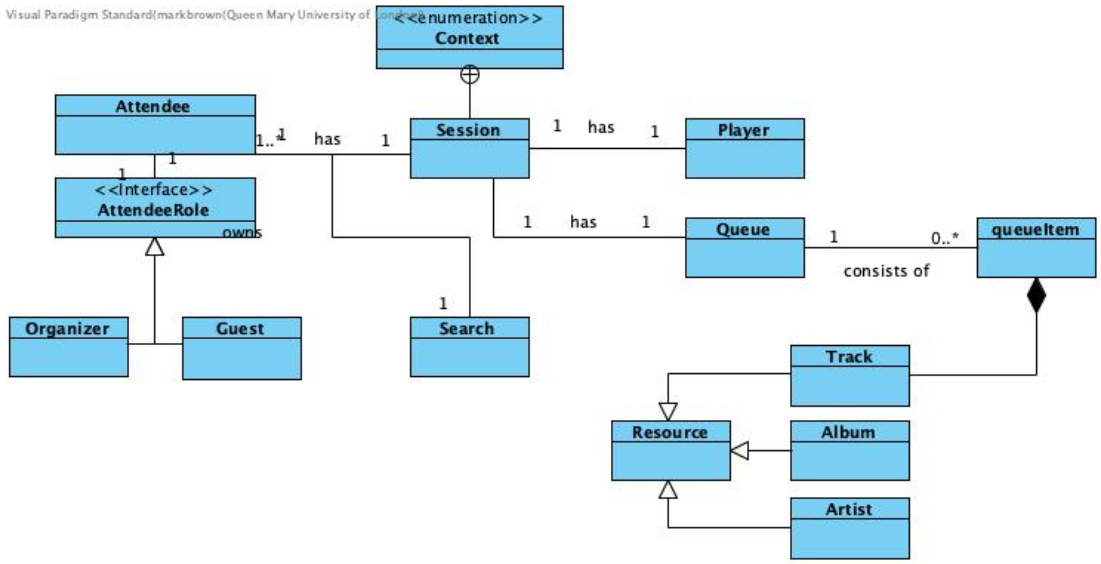

Figure 5.7 *Abstract class diagram*

This information will jump start the implementation process and allows us to start creating model classes straightaway.

As mentioned earlier, Cocoa applications promote the use of the MVC (Model-View-Controller) pattern to integrate with existing architecture; ensuring that the objects play one part of the MVC role makes the integration more manageable. The MVC architecture is implemented across the application; this ensures that objects communicate with other objects correctly.

The design process now consists of both visual and architectural components which are comprehensive enough to begin the implementation phase. A complete collection of design deliverables can be found in the supporting materials; please refer to the Readme file for more information.

6 Prototype & Implementation

The technical documentation and design can be iterated indefinitely, deciding when the implementation phase can commence, is a balance of project resources (such as time and scope). In this project, the implementation of core features began before the end of the design testing phase. This choice allowed for efficient use of time and iteration on some design elements before the complete implementation began. These core features were primarily mock implementations of Spotify API and Firebase features.

6.1 GitHub & Source Control

To effectively manage code and project resources, source control was used, before the project was created a git repository was initialised via GitHub. This repository can be found at:

https://github.com/mmarkgbrownn/publicTurntable

While GitHub was used to commit code that could be reverted if need be; other source control tools such as OneDrive cloud storage was used to allow cross-device development without the need to make insignificant commits.

These source control tools were also used to create necessary documentation of progress and activities within a timeframe by the use of commit messages. Through this, a comprehensive understanding of the implementation activities can be collected.

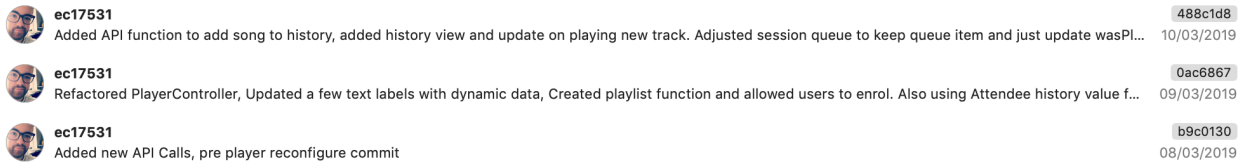

Figure 6.1 *Selection of commit messages*

XCode was the development environment used for this project. Its source control integration tools were the primary commit method; this allowed for comprehensive review and conflict management within the environment.

6.2 Resources

Throughout the implementation, documentation and other help resources were used to overcome challenges or apply API methods and functions. The main sources were:

- Apple Documentation (Apple, 2019)
- Spotify Web API Reference (Spotify, 2019)

- Firebase Documentation (Google, 2019)
- Stack Overflow
- Object Orientated Software Engineering book (Lethbridge et.al, 2005)

Collectively the Apple, Firebase and Spotify Documentations were used to see how to implement features of the corresponding API's. Spotify Documentation was used to identify the URL and response format for Web API calls. Firebase documentation included comprehensive details of how the Auth() objects work, how to refer to database children and observation methods. Apple Documentation was used more frequently to understand how to implement classes and protocols.

The documentation was very well presented but on occasion was not detailed enough to solve the problems that were being addressed. Stack Overflow is a community which provides support amongst software engineers and coders. In scenarios where the documentation was misleading or not sufficient; similar problems were browsed through, and their answers were used to help solve issues.

Another resource was the Object-Orientated Software Engineering textbook (Lethbridge et al. 2005). This book was used to identify components in a UML diagram and the uses of design patterns. The testing chapter was later used to aid black-box testing methods.

6.3 Implementation Tools

As indicated by the resources Swift, Spotify API and Firebase were used in the implementation to provide a range of capabilities. The entire list of tools used consists of:

- Swift 4.2 / 5
- Firebase 5
    - Firebase Auth
    - Firebase Database
- Spotify Streaming SDK
- Spotify Web API
- Swifty JSON
- LBTA Components

Swift 5 is the most recent (at the time of writing [April 2019]) release of the Apple-designed programming language. It is used explicitly for Apple built products and software such as; MacOS, iOS, WatchOS and TVOS.

This language includes features of which inspired clean code and sophisticated implementation. Closure blocks allowed some simple statements to be written quickly and semantically. Completion blocks were used frequently to notify methods when an API call or function had responded. This notification was particularly useful when URL

requests run asynchronously. The below example illustrates how a Web API call made use of the completion block.

```
SPTAudioStreamingController.playSpotifyURI(nowPlaying) { (error) in

    if error != nil { print(error!); return }

}
```

Figure 6.2 *Example of a completion block*

A common find in swift is an optional value; this is where a variable might be initialised with no value. Optional values are enumerations with either a value of nil or a wrapped value. To use an optional, the value must be unwrapped. There are several ways of achieving this; Optional binding is the most common way to do as such. These bindings result with a new local constant or variable values which can be used within the binding statement.

In many cases, optional binding is used to check for an optional value. This type of use can create multiple cases where the else statement is simply used to return and transfer control outside of the function, while the rest of the function lies within the if statement.

An ideal solution is guard statements. Guard statements are similar to the optional binding method above but provide a much cleaner syntax, the else condition of a guard statement must exit the current scope, or an error is thrown. When compared the implementation is much cleaner.

Another clean implementation method that prevents overuse of the if-else statement is ternary operators. These operators provide a clean and semantic way of saying if/else but arguably are more performance intensive. “By doing nothing more than replacing the ternary operator with an if else statement, the build time was reduced by 92.9%” (Gummesson, 2016). This information raises questions on whether readability is more important than build time. In this project, this issue is managed by applying these more explicit implementation methods when code becomes tricky to read.

Google Firebase was used to manage user auth and implement the real-time database. These libraries were managed using cocoa pods, the specific pods that were imported were the Firebase/Auth & Firebase/Database.

Firebase Auth used methods to sign users in or register them based on email and password data. Logged in sessions were also checked to see if user sessions were still active when reopening the app.

Firebase Database was used to add user details, session parameters and session queues to the firebase service. Observers were set up to monitor the session queue and session objects, looking for additions or changes in the data structure.

Spotify currently provides two API for authorising users and playing music. The remote API requires users have Spotify installed on the device to generate an authorisation token. As the application design focuses on being as independent from Spotify as possible, the Streaming API was the preferred resource. This API allows direct audio streaming from Spotify instead of acting as a remote for the main Spotify application.

The alternative remote API also uses the tracks played to later inform personal music suggestions based on the played music. This could result in disrupting the hosts personal music library.

To serve functionality such as search, adding to the queue, adding to the library and history playlist. The Spotify Web API was used. This implementation of the API performs URL calls which return JSON data that correlates to the requested data. To effectively extract the required data from the JSON responses, cocoa pod Swifty JSON was used.

In swift 4.2 the decodable object was given the functionality to map a JSON response to the custom NSObject class. This ability makes for very simple and easy to manage data collection from web services assuming the keys match up. In the case of the Spotify JSON responses not all the data from the response is required and, more often than not, the naming conventions do not match. As a result, API responses used the SwiftyJSON cocoa pod to extract data. This pod allowed for simple reference to objects or data within the tree structure of the JSON while also handling casting data to types such as String, Int or Array.

{
"birthdate":"1937-06-01",
"country":"SE",
"display_name":"JM Wizzler",
"email":"email@example.com",
}
	let username = $0["display_name"].string
	let email = $0["email"].string

Figure 6.3 *Sample JSON response from Spotify (Left) SwiftyJSON solution (Right)*

6.4 Process

The implementation process saw three distinct phases; the UI implementation and general app flow with mock-up data and assets, the Firebase Integration and finally Spotify integration. This process was necessary as the Spotify API could only be run on devices, not simulators. To maximise the flexibility of the development process the Spotify integration was left until the last moment. In the meantime function stubs were written to mock implementation.

6.4.1 UI

To start with, the data structure objects were made to begin identifying how the data would be displayed. These models included Track, Session, QueueItem, Attendee and SessionQueue objects. Following this, views were made to display the data from these objects. The UITabBarController was set as the AppDelegates rootViewController allowing TurntableTabBarController to be displayed first at launch. This controller has three child controllers within the hierarchy. These children are; SettingsController, PlayerController and SearchController. This would later be replaces by the rootViewController class to handle switching between app and user states.

The player controller was the most complicated of controllers to implement as it made use of UIView and UICollectionView which had to be displayed as sub-views in PlayerController. Ultimately the PlayerController became a subclass of the UICollectionViewController class and implemented three collection view sections. This decision is discussed in more detail later.

To add visual elements to the view hierarchy sub-classes of UIView (or in some cases UICollectionViewCell) were implemented and then referenced. These view classes setup each view element as a constant and added them to the view's hierarchy in that class. As more UIView subclasses were made it became clear that there is a cleaner way to implement these views. The UIView subclasses were implementing the init() and the required init() method. A class called BaseView was introduced and made the superclass of all the view elements with the init() and required init() methods implemented which prevented irrelevant code appearing in each class.

A function that was implemented in all of these classes was SetupView(). This class was used in each view to setup sub-view constraints. The Swift standard method to do this is to either use the visual format or anchor constraints. Both of which are difficult to read and debug.

To make this easier an extension class was implemented that allowed the anchor() function to be called. This function allowed the setup of all four anchor, padding and size constraints. Once these constraints were correctly setup, the view could be added to the hierarchy. Almost all of the sections within the app have view setups like this. However, there are a few that make use of UITableViews.

The UITableView setup is somewhat different from the other method. This method requires the use of both the UITableViewController and UITableViewDataSource protocol. UITableView was used for the host session and settings interfaces; these interfaces were the most suitable for UITableView due to the numerous rows of data each with a different accessory.

When implementing UITableView complications arose with the terminology. Each row in the TableView consitis of a title, subtitle and detail. Within the Apple Documentation, it is not very clear how to access these properties. Figure 6.4 displays where each of these components lies within the row. After extensive research through the documentation, it appears that tableViewRows have predefined styles which can be set when using the deuqueResuableCell method.

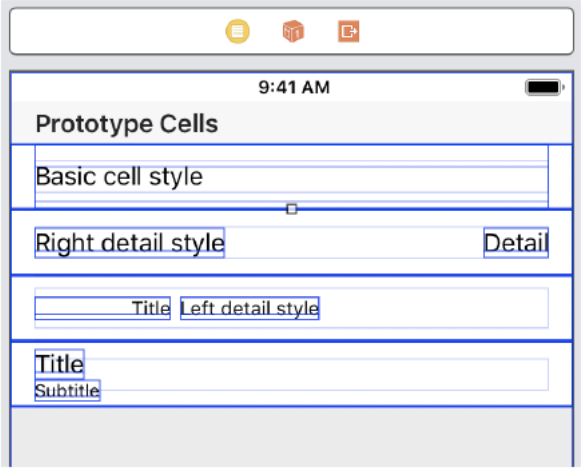

Figure 6.4 *– UITableViewCell structure (Apple, 2019)*

6.4.2 Firebase Integration

Once the player, home and setup views were created Firebase could be integrated. The first part of the integration was authorising the user. To ensure that the Spotify integration did not cause issues later on mock Spotify credentials were stored within the project. These were later removed when they were no longer required. The main issue with the firebase authentication was deciding how to provide the user details. The app design displays a single sign-on method where the user selects the ‘connect Spotify’ action and then authorises via Spotify in a web or app interface. The issue here was what to provide as a password to firebase.

As the firebase database does not hold sensitive personal data that is not available through Spotify the Spotify user id was used as a password.

Another issue was how to figure out which users had used the app before and required the signIn method instead of the createUser method. A function called fetchProviders(forEmail: String) returned the type of sign-in method for the provided email. If no email was found the function returns nil. This logic is applied to decide which function is necessary.

Firebase Database was used to add and observe details from the users' associated session as well as creating the session. The addition of data to the database was simple.

It required a database reference and child values (Figure 6.4). This process is another demonstration in which completion blocks were used to run code after the asynchronous activity was complete.

```
Database.database().reference().child("session").child(sessionKey)
```

Figure 6.5 *Firebase database reference example.*

Observing values was just as easy; using a similar reference to the data structure and calling observe or observeSingleEvent allows the observations of data within the reference. Using the DataEventType enumeration, different types of events can be observed. In the observeQueue function, the event type of '.childAdded' is used to notify the observer when data is added. Alternatively the '.childChanged' event is used if the sessions nowPlaying value is updated to reflect the organisers player change to a new track.

Once these features had been implemented the implementation was ready for Spotify Integration. Testing the current implementation on several devices in a single session was also a good idea at this point to assess the performance of the app and real-time features on both new and old devices (iPhone X & SE).

6.4.3 Spotify Integration

Similarly, to the firebase integration, Spotify authentication was the first feature to implement. This decision completed the authentication process and finalised the single sign-on feature.

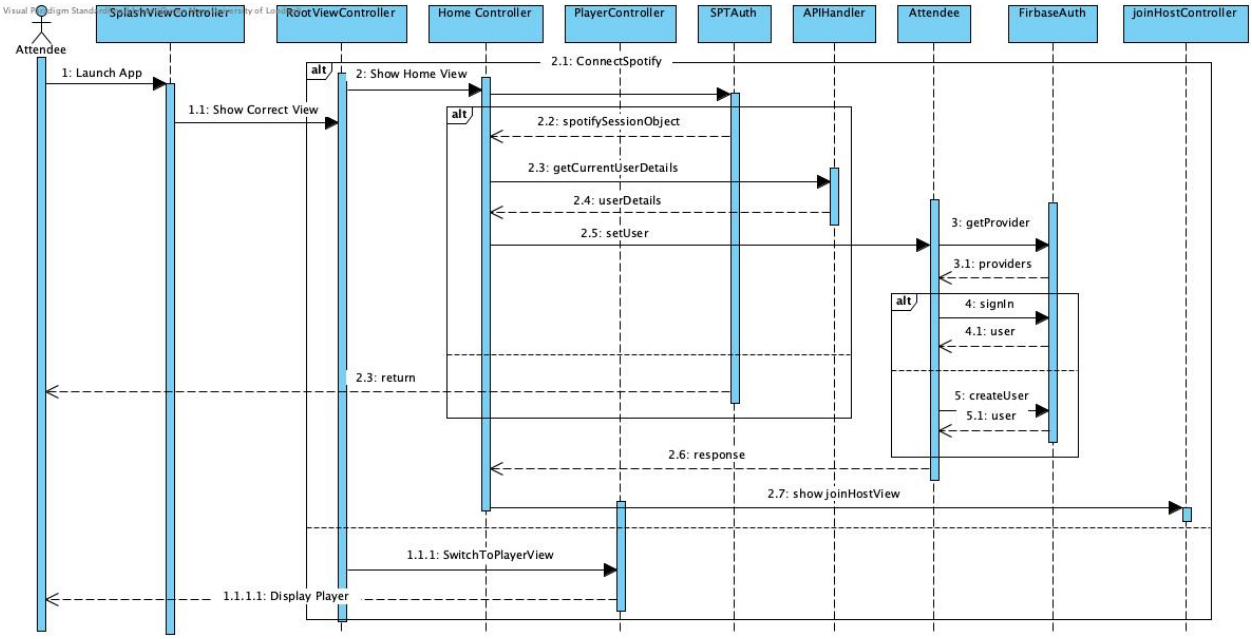

Figure 6.6 *Sequence diagram displaying the flow for authorising users.*

The authentication requires an array of scope objects that are available from the Spotify API. These scopes identify what kind of functionality the app wishes to use and more importantly enables the user to make informed decisions based on how their data and profile is accessed and used. The scopes used include; library read, library modify, read email and read private.

The authentication process provides the application with an access token that can be used to make actions on the user's profile. This access token has an expiration date of one hour after authorisation. Once the token has expired, a new token must be acquired. To do this an encrypted refresh token is provided, this token can be sent to a refresh service that is set up via Heroku.

In the Spotify documentation, the token refresh service is discussed at length. Spotify recommends using its One-Click-Deploy Heroku app to manage token swapping. As this is already set up, it is uncertain how this works; however, once the endpoint with the refresh token is provided and called, the service replies with a new access token.

To implement this without disturbing the users experience the API provides a function to check if the access token is still valid. This function is low cost and can be done before the Spotify service is used. The authentication process only returns a Spotify session object that contains the tokens and expiration date. To get the users full details for Firebase auth another API call must be made to acquire user details. This call is achieved via the v1/me endpoint. The response is JSON data with the user data including Spotify id, display name, email and date of birth. Once these details are returned, they are then sent to Firebase auth.

At this stage of product development, the only streaming functionality required is play/pause, and skip forwards. These features are simple to implement and can be done by setting up the SPTAudioStreamingController. The streaming controller delegate is implemented as a superclass of playerController, the sharedInstance() can be referred to from anywhere in the player. To initialise the player the login(withAccessToken: String) method must be called with the user access token. This function is called if the user is also the organiser of the session.

If the login is successful, the delegate function audioStreamingDidLogin is called. In here we begin playing the sessions nowPlaying track. This play function takes the Spotify URI of the track and also accepts a starting position. Currently, this is always set to 0 but could be used to continue playing the song from the point in which the app was closed in a future version.

To toggle the play/pause state the isSetPlaying value of SPTAudioStreaminController is set to true or false dependant on the state. To handle skipping songs or playing through the queue, several actions are taken. These actions ensure that the same song data is presented across all devices in the session.

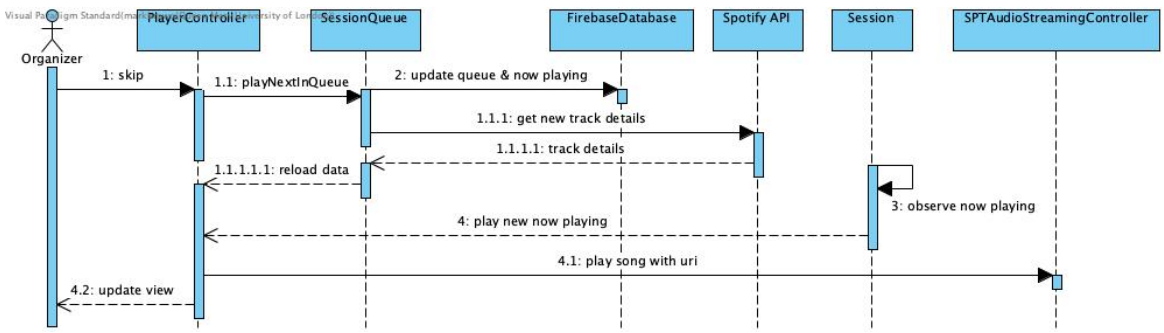

Figure 6.7 *Sequence diagram describing the flow for skipping a song*

The rest of the Spotify functions are provided by the Web API, this application implements the following endpoints:

- v1/me (get user details)
- v1/tracks/trackId/?market=GB (get track details)
- v1/me/tracks (add track to user library [track id passed as url data])
- v1/me/tracks/contains?ids (check if track is in user library)
- v1/users/userId/playlists (create playlist)
- v1/playlists/playlistId/followers (follow or unfollow playlist)
- v1/search?queryString&type=track (search tracks)
- v1/playlists/playlistId/tracks?uris=spotify:track:trackId (add track to playlist)

Endpoints such as ‘check if track is in user library’, ‘follow/unfollow playlist’ or ‘add track to playlist’ return simple JSON that indicate success. However, a more straightforward way to check for success in these cases is to check the HTML response code, 200 indicate success in these cases.

Other endpoints such as ‘get track details’ or ‘search tracks’ return track details. These track details can be extracted with the SwiftyJSON cocoa pod mentioned previously. To make the process simple a Track initialiser that takes a dictionary object of [String : Any] was created to allow for the JSON response to be converted into a dictionary which is then passed to Track objects.

A similar process is used for the ‘get user details’ endpoint. However, this function returns JSON data. The authentication method extracts the necessary data.

Endpoints do not always return useful data, ‘add track to playlist’ and ‘add track to user library’ return library snapshots, these are not useful in these cases. Similar to the ‘follow/unfollow playlist’ function the HTML response code is checked for success and the response data is discarded.

6.5 Challenges

Throughout this chapter implementation methods have been discussed at several points. Some of these points were challenges that were overcome by testing or research and could be solved within an hour. However, not all these challenges were as simple. This section will highlight the most challenging aspects of the implementation. Some of which are still issues in the current build.

6.5.1 App UI Architecture

As mentioned in 6.4.1 the player view was easy to implement visually but each solution seemed to raise issues with the interaction method. The first approach took was to implement both the now playing header and Up Next/History views as sub-views of the controller. However, this meant that one would overlay the other, to solve this problem the UIEdgeInsets was set on the collection with the height set to that of the header’s height constraint. This visually seemed like the perfect implementation and matched the design from section 5. This solution later raised issues; UIEdgeInsets manoeuvres content within the associated view which from a visual perspective has no impact. Similarly, to how CSS padding works this means that the edges of the view remain in the same location; it ultimately prevents users from interacting with the underneath view.

Amending this solution seemed straightforward, instead of using the UIEdgeInsets to move the content down try to attach the collection views top anchor to the headers bottom anchor. This solution again worked visually but prevented scrolling actions.

The final solution used was to implement the entire view as a collection view with each section of the collection view being sections of the display; i.e. Header, Up Next and History. This solution was much better both for the interaction and the code architecture. The previous methods implemented or altered the view inside of the controller which breaks the MVC design pattern and increases coupling. Figure 6.7 illustrates how these sections are implemented in the collection view.

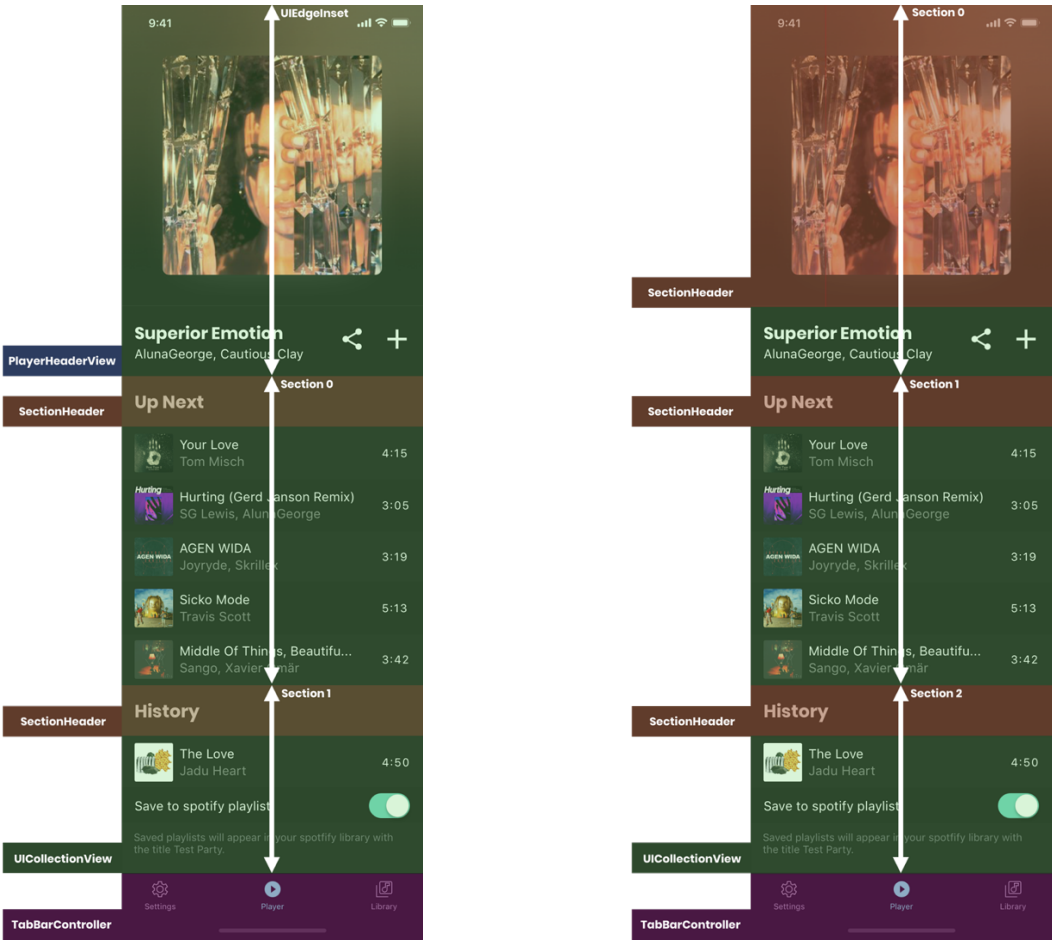

Figure 6.8 *player using EdgeInsets (Left), alternative player using collectionView sections (Right)*

One of the most common errors was the implementation of updating/refreshing UI components with new data. Numerous functions in the application are asynchronous for example; calls to Spotify API or Firebase observers. Both of these examples should update the UI more often than not. Apple Documentation advises that all UI updates should happen for various reasons. A few of them including:

- UIKit is initialised and runs on the main thread; all UI components are part of UIKit
- Asynchronous events can finish at any time; UI should be updated instantly.
- Touch events occur as part of UIKit and therefore are on the main thread.

These UI updates need to be carefully considered as what constitutes an update may not necessarily seem like a UI update. When calling collectionView.reloadData() UI is updated to propagate the data structure changes on to the view. Trying to run this

function on an asynchronous thread will result in a crash. This issue is simple to fix as dispatching instructions to the main thread can be done within most asynchronous functions by using the DispatchQueue class.

6.5.2 Working with Spotify

The second biggest challenge revolved around the use of Spotify. The authorisation method described above makes the implementation seem simple; however, when testing the application with multiple users, it is unclear whether Spotify has successfully authorised or not. Failure sometimes returns an error (of which is not informative) and other times does not.

These errors are sometimes justified, e.g. when there is no WIFI or 4g connection or when the authorisation is aborted, and correct error reporting needs to be implemented in the main application. The issue is when the authorisation fails for no apparent reason. These issues are yet to be solved and will be the main focus of the next build.

The streaming SDK makes use of the AVFoundation tool kit. This API is the method for which the audio is produced and controlled. A few issues arise here; the most prominent of which is the lack of background audio support. Currently, locking the device or sending the app to the background stops the audio. Apple documentation claim that the solution is to simply enable background AVFoundation activity in the app capabilities. However, there seem to be other issues causing this behaviour.

6.6 Using the Debugging Tools

During the entire development process, the use of the XCode debugging tools was vital to success. The most common of which was the breakpoints. Breakpoints were used when unexpected behaviour occurred, and the investigation of values and execution flow needed to be monitored. Breakpoints in XCode can be added and removed during the application running. This feature was particularly useful when testing a new function. In conjunction with breakpoints, the debug console was used to investigate values. The console allowed reverse engineering of the problems that were occurring.

The other primary debugging tool that was used was the 'view hierarchy inspector'. This tool allowed the visualisation of the structure of the views in real time. The tool also allows the manipulation of this by spreading apart the hierarchy allowed for a more unobstructed representation. The tools primary use was used for building new UI components.

To easily manage the bugs and tasks of the development and debugging process Jira tasks and bugs were created. This Kanban style presentation allowed monitoring of personal progress while also maintaining a detailed documentation of issues that existed and how they were fixed. This was useful in a few occasions where bugs reapered.

7. Testing

Several methods were used to test various aspects of the product to ensure that it was capable of the functionality outlined by the requirements set out in chapter five. Similar to how chapter four used multiple techniques to find as many points as possible; the same method is applied here. The testing activities used include:

- User Testing
- Black box testing
- Coverage Testing

This variation of testing activities returns findings from behaviour to implementation bugs and defects. By making improvements and fixes alongside development, a well-prepared evaluation can take place with little interruption from minor issues that could affect our ability to prove or disprove the hypothesis.

7.1 Testing Methods

User testing seeks to understand how the design and interface facilitates a specific task such as ‘joining a session’ or ‘queuing a track’. With a deep focus on each segmented task the findings present themselves as issues in the visibility of the systems features, the quality of the feedback and placement/mapping of elements.

Black-box testing focuses on revealing defects within the solution. Bugs and code defects have a huge impact in the users experience and can ultimately distract the users from their task resulting in unrealistic findings in regard to the hypothesis. This was learnt the hard way by attempting two field tests of which major errors caused basic features to fail in the collaborative space.

Black-box testing methods begin by testing out software features without any insight or consideration of the implementations details or architecture. Once the basic features are tested more sever interactions can take place to see if anything ‘breaks’ the product. This method found countless issues throughout the project.

Finally, coverage testing aims to uncover product defects across a range of OS versions and device models. This is critical to this solution due to its nature of being a social product. OS and device version/model are variables that cannot be controlled and are highly likely to vary with any group of individuals. Fortunately, resources were available which allowed testing of extremes including; screen size, model and software version.

7.2 Testing Tools

The tools that were used to record or facilitate testing in the project largely vary. Complex tools and analytics methods such as Firebase analytics were used to capture quantifiable behaviour while simple tools such as paper prototypes were used to evaluate the quality

of a design. Knowing what to test and how is a skill that has become more apparent in the projects lifecycle and can be a challenge.

Equipment such as video and audio recorders were used to capture ethnographic and usage data that could be revisited later for analysis.

TestFlight and the AppStore was used to distribute the product for testing; this provided an easy to access and reputable source for attendees to use. Formal review from the app store connect service is required by every app used publicly on apple devices, this approved the build of the product while also indicating a certain level of quality. This service also presented analytical information regarding the total installs and crashes as well as basic user information.

To make the application public the app store requires several documents and details that may be considered beyond the scope of the project. However, to test effectively these were absolutely necessary and helped solidify the products purpose and description, these items included:

- Privacy policy – helped formulate what data is being collected and how it's used
- Age Ratings – how might the app impact people of different ages and backgrounds
- Promotional Text – how to describe the end product and what it achieves
- App previews & screenshots – what are the most important parts of the product

7.3 Findings

Below are the findings from the testing activities. Some of the mentioned issues were detrimental to the evaluation phase and also discuss a solution that has been implemented while other talk about future work that would take place to solve the issue.

7.3.1 Design & Usability

When presented with the task of joining a session; several test users found it difficult to enter the session code. The design of the text field lacked the correct signifiers to indicate to the participants that it could be tapped on. A quick fix to this was to select the text field automatically when the view loaded. This was a very effective fix and solved the issue.

Asking users to host a session resulted in a straight forward and hassle free experience for the testers. However, some users were confused when they had completed the setup session process because their player screen did not play music even when the transport control mechanism indicated music was playing. When looking into this it seems that Spotify free users don't have access to the streaming features of the SDK, as a result the player simply doesn't play music. A fix for this would be to indicate to users when they connect a Spotify free account what they can and cannot do. While this couldn't be

implemented in this time frame; a solution for this might look something like figure 7.3.1, 7.3.2 and 7.3.3.

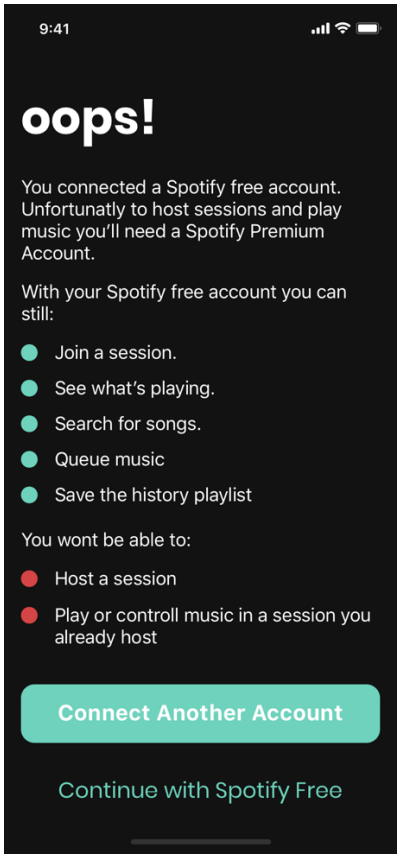

Figure 7.1 *– Display of available features for Spotify free users*

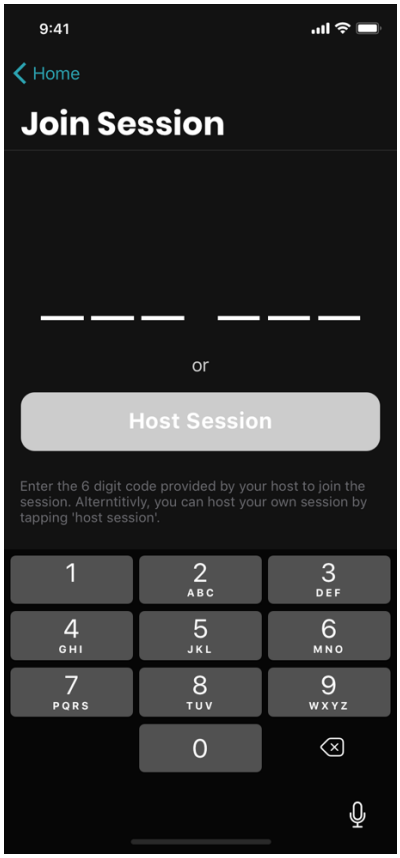

Figure 7.2 *– Disabled Host session button for free users*

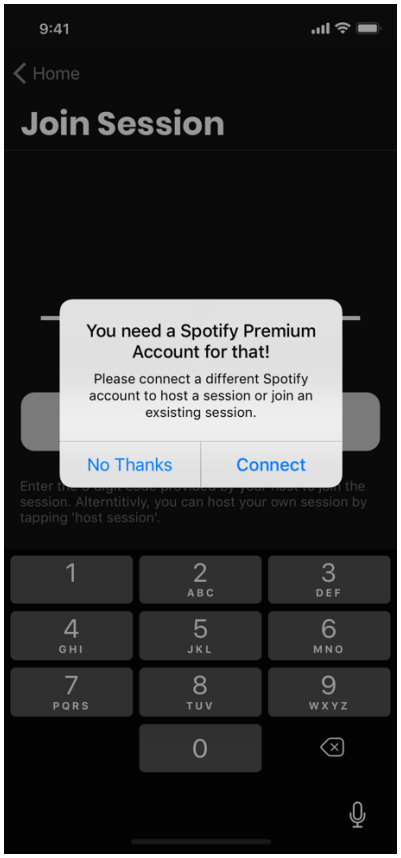

Figure 7.3 *– Prompt when free users tap host session button*

When asked to queue a track some users would tap on the 'add to library' button which resulted in them adding the song to their library, this incorrect action is a good example of a mistake. Their previous knowledge with the + symbol indicates that this has the addition action and their mental model of adding a track to the queue aligns well with the + signs perceived action. In the visual hierarchy of the player screen the + button is more prominent that the search tab below, this is most likely the main reason for this error. A simple fix would be to add a descriptive label to the + button or to make it less prominent visually.

The first couple of tests that involved asking the users to save the session as a Spotify playlist went well, visibility of the switch was clear and the switch promoted the correct affordance (tap to switch). However, once a large amount of tracks was added to the up next or history sections it was difficult to discover how to complete the task. Moving this action to the top of the history section would improve this issue but not fix it entirely. An alternative solution would be to modify the layout of the player screen to present the up next and history sections as tabs, as seen in figure 7.4 & 7.5.

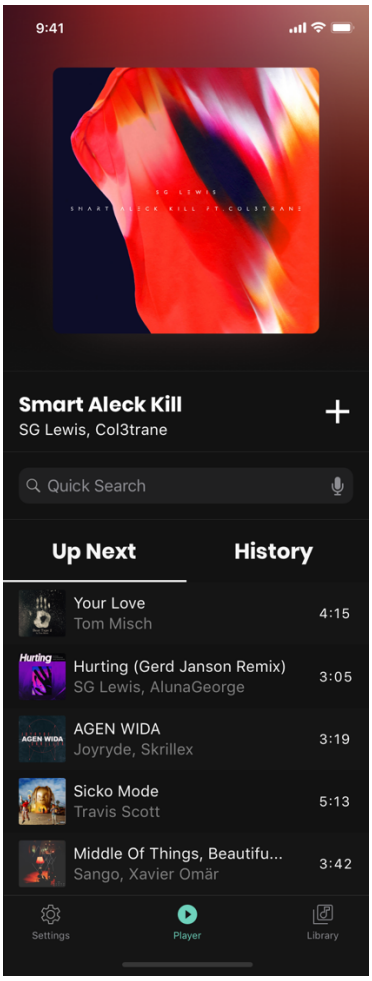

Figure 7.4 *– Alternative Up Next Design*

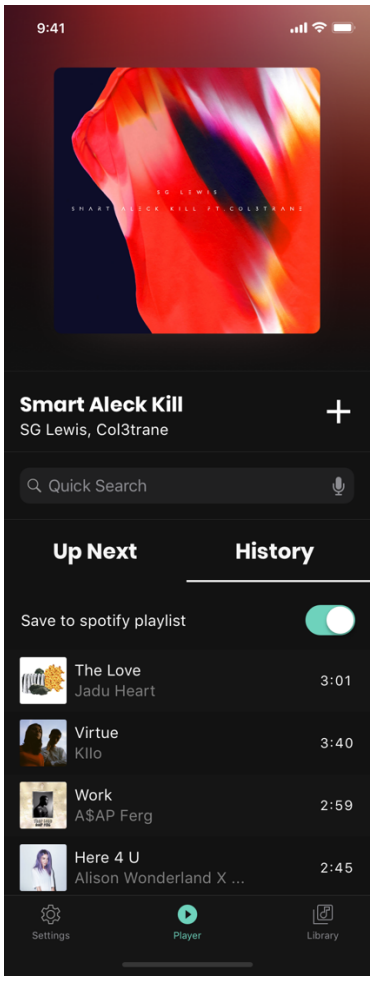

Figure 7.5 *– Alternative History Design*

The final issue that was observed in the user testing session was the lack of visibility of the share action. A large collection of testers actually asked what the button did. This response was a big surprise as the icon was derived from the Google Material Design icon collection and was thought to be a well-known symbol for the sharing action. However, it appears that this icon isn't very common among iOS apps which displays why users were unfamiliar with the icon.

Switching to a common apple designed icon would most likely improve awareness of the icons function.

7.3.2 Black Box Testing

When personally testing the application the most detrimental issue found was inaccurate search results. The search results would never give valuable or even relevant results. The search function updates the view after every keyboard tap, this means that search results might display the result the user is looking for without having to type in the whole name of a song or artist. When this issue was investigated it appeared that the view was updating the results with the key tap for the previous entry. For example a search for “diplo” would return results for the search string “dipl”. Moving the update view function call into a completion handler ensured that the results displayed were always the most recent.

Another issue within the search screen was that the ‘in queue’ indicator would display that tracks are in the queue when they weren’t. Again a simple fix solved the issue. It seemed that the string value for the image reference wasn’t being reset with every instance. Again moving code into a different location solved the issue.

The session code text field waits for exactly 6 characters and then attempts to join the session. This means that users can very quickly join session by just typing the number in. However, when an incorrect session code is entered there is currently no feedback. This is an issue as users might come confused when they think they have entered the correct code and wait for something to happen. Checking for an incorrect code and making some display change such as a red highlight or haptic feedback to prompt session code re-entry might improve this issue but is not likely to happen during the time frame of this project.

The application saves user data such as access tokens and session keys when the application closes. This means that when the app reopens they can jump straight back into the session. A fullback from this setup meant that in some cases the app would consistently crash when trying to reopen that app. After a thorough investigation it appeared that an escaped nil value was the cause of this. Refactoring the observeNowPlaying function to ensure that the history array only added an object when there had been more than one song played fixed this. This fix also introduced a new issue where some now playing tracks would appear in both the up next and in the now playing header, this issue is yet to be fixed but only appears occasionally.

The final two issues regarded layout and auto constraint settings. In the player some track names would run off the screen, this was simply fixed by changing the trailing anchor of the label and adapting the line break mode to truncate the tale of the text and add a ellipsis.

Similarly the session name in the setup session screen would also overrun. The same approach was taken to prevent this, however, an extra fix had to be made to ensure users don't put in unwanted long session names. A delegate method called shouldChangeCharectersIn was implemented to truncate any string over 20 characters long. This solved both issues regarding this text field.

7.3.3 Coverage Testing

This test made use of two physical iPhone devices (an iPhone X and SE) with different screen sizes, this allowed for extreme testing with the largest available screen size and the smallest. Throughout the testing both devices behaved in identical manors. The only variation was the display. These differences were usually a result of using constant values in auto constraint layouts.

The only display issue that prevented users to perform an action was the session code display after setting up a session. As the width of the label exceeded that available to the smaller device the label didn't appear on the screen. This caused the session organiser to not be able to verbally share the code as soon as they created the session.

The other two variations were cosmetic variations causing the layout to become uneven. The most prominent was the home screen layout. This sub view was being anchored to the top of the devices screen with a padding value of 60pt. Due to different pixel densities across devices 60pt appeared much larger on the smaller devices. Fixing this view to the centre of the super views y axis ensured that the content would always be displayed correctly on all devices. The same issue occurred with the now playing meta data view in the player. This issue was solved very similarly to the home screen.

Most recently, new issues have been uncovered when testing in various physical locations. Places where 4G is has to be used (in a park for example) there are several places where the application seem to crash. These issues do not occur when connected to WiFi and therefore seem to be related to the connectivity method. The most problematic function was searching, which also happens to be the most common.

As this is the most recent discovery, no solution for this issue have been implemented, however, on initial investigation it seems that the issue revolves around the extended period taken to return from a completion handler.

While coverage testing ensured that the application was correctly working across all the devices it did not display many issues.

8. Evaluation

Evaluation is the activity of presenting accumulated designs and ideas from prior findings to prove or disprove a hypothesis. In return this displays the amount of value provided by the solution. Providing evidence that the designed and implemented solution works is key at this stage of the development, it allows for reflection on how the solution performs and also presents new issues and defects caused from the designed solution that may not have been visible in the other testing methods above.

Throughout this evaluation process we can derive quantifiable metrics. These metrics can be used to quickly refer to the improvements made and act as evidence. When all the metrics are available, we can evaluate which parts of the system perform best and are the most/least effective. These metrics can also be used to direct future work and to generate interest from potential investors.

The more involved evaluation methods described here show data that clearly indicate areas for improvement and narrows the focus for future work in ways that activity focused testing cannot.

The key outcome from this process is justification of the products existence displaying that it provides value to the users.

8.1 Evaluation Methods

Several methods were used in order to obtain a holistic view of the products achievements and downfalls. These activates ranged from active analysis during field testing and analysing data that was collected automatically using various tools.

8.1.1 Observations

Field testing is a holistic approach which attempts to understand how the solution works in the environment in which it is designed for. This activity focuses on the behavior of the attendees and how the solution improves their social interaction and ability to request and play music within the social space.

Observations were focused on:

- What the user was doing generally during a session
- Their interactions with other people (both verbal and non-verbal)
- Their facial expressions (did they look happy/confused/angry etc)
- Their environment and how they interacted with it and the objects in it
- Artefacts/Tools that they used (e.g. their smartphone)

Initially observations were made by making notes while sat amongst the participants. It became apparent that this approach had several downfalls; behavior was not re-playable, presence was affecting user behavior and keeping up with users activity meant that notes were very basic. A better approach was to make video and audio recordings which could be revisited later. The camera could also be set up conspicuously to not intrude However, this setup required consent which may have effected their behavior as they were conscious of the recording but users soon forgot and behaved as normal.

These observational methods show us a lot of what the user does with the system and how; but doesn't provide feedback on the experience the user has.

8.1.2 Follow up interviews

Following up field testing with questions that investigates what the user thinks and feels during usage further evidences the value the product provides while also showing us what could be done better.

Questions that gaged what users thought and felt were compared to what they did using the recordings to see if these actions aligned, this data provides an account for their actions and utterances. The content followed a similar construct to the previous interviews from chapter 4, this repetition allowed comparisons to be made against the earlier findings.

Questions from these interviews included:

Table 8.1 Sample Interview Questions		
How did you find out how to queue music during the event.	Was it fun/easy/difficult to add music to the queue?	Did you experience any issues?
Did these issues affect your overall experience?	How did you find the music you wanted to queue?	Was there anything you would like to add or think we missed?

8.1.3 Analytical Data

Analytical data come in many forms. The most useful data however, was the counts of user actions. This count was achieved by closely monitoring video/audio recordings and comparing these with time stamps of songs added to the queue or other recorded data.

The second form of data used was user engagement. This was acquired using Firebase analytics, a tool that captures the average screen time for a given screen class. From this, a comprehensive analysis of how time spent in the app was distributed amongst the views can be made.

Spotify's developer dashboard was used to provide data about endpoint requests. This indicates the most popular/frequently used features of the app and give a specific amount for the number of actions made in the app within a given time frame.

Finally, metadata from the acquired Spotify playlist was analysed for to identify if trends and patterns emerge when using the product. This analysis highlighted:

- Track patterns
- Popular genres
- Release dates

8.2 Results

8.2.1 Observations

The ethnographic study proved detailed insights much like the observations made in chapter 4.2. The most apparent observation was attendee's inability to discover how to queue music. The actions followed a similar structure to the original system observed earlier where users would: identify who had control of the music, wait for an appropriate time to approach them and ask them to queue a song. The response from the organiser would direct them to the app. Once they had downloaded the app the attendee required almost no guidance to join the session and queue music.

From observing the music that was played it was clear that these sessions had a larger variation of music. Some music choices were chosen as 'Jokes', these were clearly identifiable as a vast majority of attendees laughed when the songs were played. These songs were often culturally known or related to 'meme' culture, for example Rick Astley's 'Never Gonna Give You Up'. These tracks were often skipped after a short duration.

Skipping tracks was often a negotiation process between the guests and the organiser. A guest would prompt the negotiation by asking 'can we skip this one?' While this utterance is clearly directed at the organiser; the guest would gaze towards the other guests prompting them to speak next. Gaze is a common communicative tool to signal turn taking (Nakano, Reinstein, Stocky and Cassel, 2003) and can be manipulated to select speakers when accompanied by an attributable silence (longer than 1.0 second).

The general variation of music and genres is a product of the collaborative nature of the solution. This variation increases the likelihood of abrupt stylistic and tempo changes

which was observed to have a negative impact if it occurs at the wrong time but does reduce the likelihood of the style becoming stale and uninteresting.

Out of place or generally disliked songs where very often met with justification in previous scenarios. The tested solution saw no justification for tracks that didn't fit the environment. This is mostly a result of the anonymity of queuing music in the app. Attendees also seemed more receptive with songs that fit this criteria, an analysis of this behaviour indicates that the queuing actions in the app scenario are much lower cost than that of the previous scenario. The queuing activity is much more accessible and repeatable, resulting in much less impact of a poorly chosen track.

One apparent side-effect of the app setup was the general increased usage of phones. While a numerical increase couldn't be derived from the data it was clear that people were using unrelated apps more frequently. While attendees were still capable of communicating with other guests'; utterances were not as relevant to the conversation and average feedback time would increase. In some cases, back channel responses such as "umm" and "yeah" were used when a detailed utterance was expected.

There was an apparent relationship between the age of songs and attendees. Sessions with a larger age range increased the range of release date. The initial hypothesis was that older people queued older songs. However, when analysing who added which songs; it was apparent that the entire range of ages queued more diverse music. This indicated further that audience attributes beyond personal taste affects the choice of music. Attendees are conscious of their environment and make adjustments accordingly, even in an anonymous setting. This variation of music was not apparent in the solution where one person had majority control of the music.

Attendees who didn't have access to the system (mainly android users as this version was built for iOS) still asked other users if they could queue music using their devices. Unlike the previous scenarios, where access was limited to the device playing the music, access was much easier for these users as the number of devices which controlled the music was larger.

Throughout all of the field tests a common new behaviour was a guessing game activity which involved attendees collaboratively guessing who queued which song based on their knowledge of the communities music tastes. This was often met with surprises as to who queued the songs resulting in a rise in the attendees self-esteem. In one scenario an attendee queued a song from 2012 which was seen by all the other attendees as a 'throwback'. When the guessing activity showed who queued the track, everyone was met by surprise as this choice did not fit the mental representation of that guest's music taste. This resulted in several other songs from a similar era to be queued and discussed, an activity of which wouldn't have occurred as easily in the non-anonymous, non-collaborative setting from before.

This behaviour displays a large increase in all of the attendees self-esteem and confidence in sharing music taste to others when compared to previous observations from chapter 4.

Some attendees were observed to use alternative music apps and sources to find music and then queue the music found through these alternative apps. This indicates that the search features aren't providing users with enough relevant results. Another account for this kind of behaviour would be that users find it difficult to think of a search query using the built in search features. Several times users were observed to blindly stare at the search screen for approximately 10-15 seconds before either typing in a search query or navigate to another app to use an alternative search function. These alternative apps that were used make suggestions for users as the search, for example recently searched and new suggestions are commonly found in Spotify and Apple Music.

Some queue items were inaccurately placed in the queue, this seemed to be a result of the search function requiring a single tap to add a song to the queue with no ability to remove an added item. Due to the increased acceptance of out of place tracks these slips and mistakes often resulted in attendees listening to music which was not intended to be played. These occurrences could be recognised when two songs of similar names was placed in the queue by the same user within one second of each other. These mistakes often contributed to the abrupt changes in style and tempo.

Conversation about music, artists and events were still as apparent as in previous setups and seemed to have little impact in the majority of group sizes, however it seemed that small groups (between two and five people) were more likely to discuss the music that each person was queuing as they queued it.

The 'punching in' behaviour described in chapter 4.2 wasn't possible in the new scenario at all. Guests would sometimes ask organisers if they can play a song now or next, the organiser would always respond saying they cannot. This was apparent in both situations where the organiser seemed interested and not interested in the song the guest was trying to play. The guest would usually respond by queueing the song like every other track. However, the song would often go unnoticed when it eventually played, it was apparent that the moment where the track was relevant had passed or was no longer applicable to the conversation at the time of playing.

Even after several fixes and builds some bugs still appeared during the sessions. However, these bugs didn't disable people's ability to achieve their goals but increases the time it took them.

8.2.2 Interviews

Five guest attendees were interviewed post event, three of these were male while the other two were female. Questions were asked directly after the event when possible to

gain the most detailed account. The interview structure shadowed the interviews from chapter 4. Both positive and negative experiences were investigated.

To begin with, questions surrounding their contributions to the music were asked. Of which all five participants used the app during the event. The amount of use per participant varied from users who queued 10+ songs to those who queued 2 or 3. Each of these participants described how their experience of downloading, joining and adding music was simple. However, participants who queued only a few songs noted that it was “quite a long process to go through if you only want to queue one songs and that’s it”. They added that “It can bring you back though, I only intended on queuing one song but went back for a few more”.

This behaviour shows some degree of attractiveness. People who found the initial queuing easy were likely to return to the app even when they had no intention to queue more music.

Themes occurred surrounding the inability to discover the app. Two participants sated that they didn’t know about the app at all until they asked the host if they could queue music. One of them went on further to say how they don’t usually queue music at events because they commonly aren’t that familiar enough to ask the host. This indicates that word of mouth is not sufficient enough to permit a good level of discoverability. These comments also line up with a few observations made earlier.

The most common topic discussed in the interviews was the search function. Participants shared that their experience finding songs took longer than expected. One participant further explained that on more than one occasion they paused for more than a minute trying to think of something to play. Another guest stated that they repeatedly referred to other music apps that they use too look for suggestions or their recently added. Again these comments are not surprising based on the observations made. Suggestions and library access should be a strong focus of any future work.

All five participants said that they had no issues getting the music they had queued to play, the only times songs they queued did not play was when they were queued as talking points that were no longer relevant because the song took too long to play or they were played as jokes. Again this behaviour was observed several times, adding to this observation, the participants who queued the songs had no issues with their songs being skipped as there was a sufficient rationale.

Overall, all five participants talked positively about their experience noting that they were making much more of a contribution than any other event they had been too in the past. One participant noted that this increased contribution from everyone made the community feel much more connected.

8.2.3 Analytics

Activity data was derived from notes, recordings and playlist data from both the new and old setups. To make comparable statistics, excerpts were taken from each test. The criteria was specified that the analysed excerpts must:

- Be an hour long each.
- Start from the beginning of the event.
- Have a recorded playlist of tracks that were played with the correct order maintained.
- Include six to eight guests.
- Exclude the songs played by the host (for accurate request data)

Two instances of both the original setup (where the organiser had control over the music) and the new solution were analysed. The data from each session was averaged across both setups and a percentage change was calculated to indicate an increase or decrease in activity. The algorithm shows how this was calculated where A is the average count of the session.

$$\frac{(A_1 - A_2)}{|A_1|} \times 100$$

Figure 8.1 *– Method for calculating percentage change.*

The results display a large increase in guest participation, this can be seen in both the number of requests made and the amount of people per session who request music. The data also shows a reduction in the number of requests that were skipped per session. Below is a comprehensive collection of graphs and tables to visualize the results.

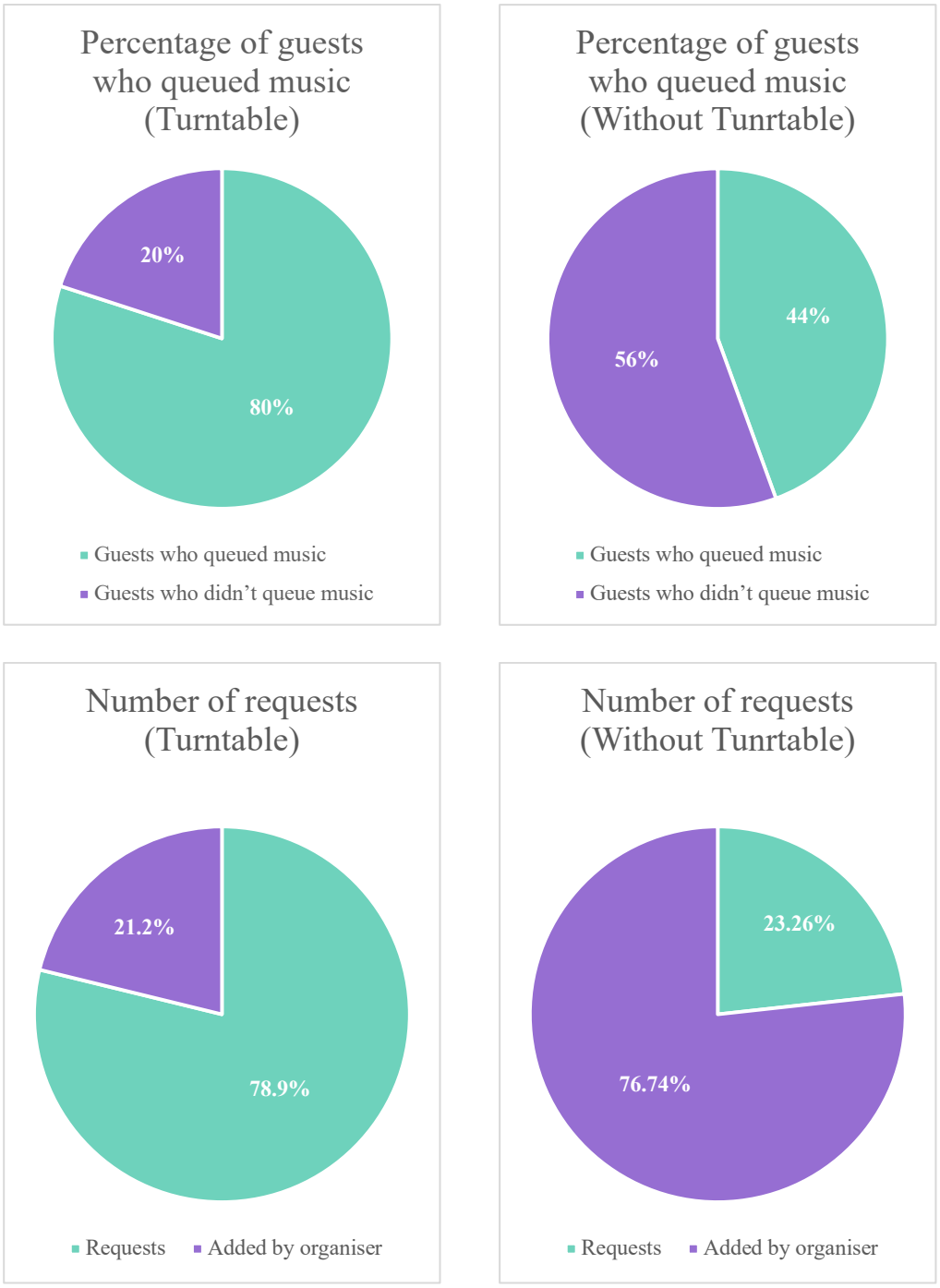

Figure 8.2 *– Percentage change for guests who queued music with and without the solution (top).*

Figure 8.3 *– Percentage change for the number of tracks that were requests (bottom).*

Percentage Change *

Number of Requests	Number of Requests Skipped
+239%	-88%
Number of Guests Who Queued Music	**Number of Songs Added To Library**
+50%	+500%
Avg Time to Add Music to Queue	
+26%	

**Rounded to the nearest whole percent*

***Table 8.2** – Collection of statistics derived from user behaviour and usage.*

From the above percentage change, we can see that the impact of the app on collaboration in the social space is mainly positive. The only downfall in the above statistics is the length of time taken to add a song to the queue. As observed from the recordings; attendees had trouble queuing music, this metric reflects this negative impact.

Spotify track objects include analysis of song 'features' these features describe the sound and themes of the track. Using this information, we can see what common track attributes appear during these events. This data could be useful for seeing if different social contexts have more prominent features. In the analysis on the session playlists two reoccurring attributes were 'happiness' and 'dancability'.

These popular attributes are not unexpected as these themes are usually associated with social events. However, this data will become more significant when testing in more varied environments such as dinner parties.

The most common genre for all of the sessions observed was never similar. The top five most popular genres however, always included electronic and pop music. Again this might be a reflection on the types of environments that were tested but once more data is collected we can start to hypothesise about user traits and their listening habits and weather this has any impact on other users music choice like the release date has.

The observations made note of the of the larger track variation when the age range was more diverse. Release date analysis also shows a diverse release date range with session of an increased age range. As mentioned before this larger variation is made by both the older and younger attendees.

	Session A	Session B
Number of attendees	6	8
Age range	22 – 61	20 - 25
Number of tracks played	18	18

Table 8.3 *– Table of data from each session used in below graphs*

This data gives insights into how the attendees are somewhat aware each other's musical preference and adapt to that. This also shows an awareness of the older users way of consuming music. As mentioned earlier, for majority of millennials and gen z music discovery can now be a daily activity made possible by streaming services.

Baby boomers and gen X are much more used to music discovery becoming a long term activity. Meaning that music most likely stayed popular for longer. This could be a reason why we see a much wider variation with older users. Of course these comments are merely theories. More research would need to be carried out to make a detailed account for this patterns appearance.

The below graphs display this variation is a graphical format while table 8.3 gives the attributes of the sessions and the audiences.

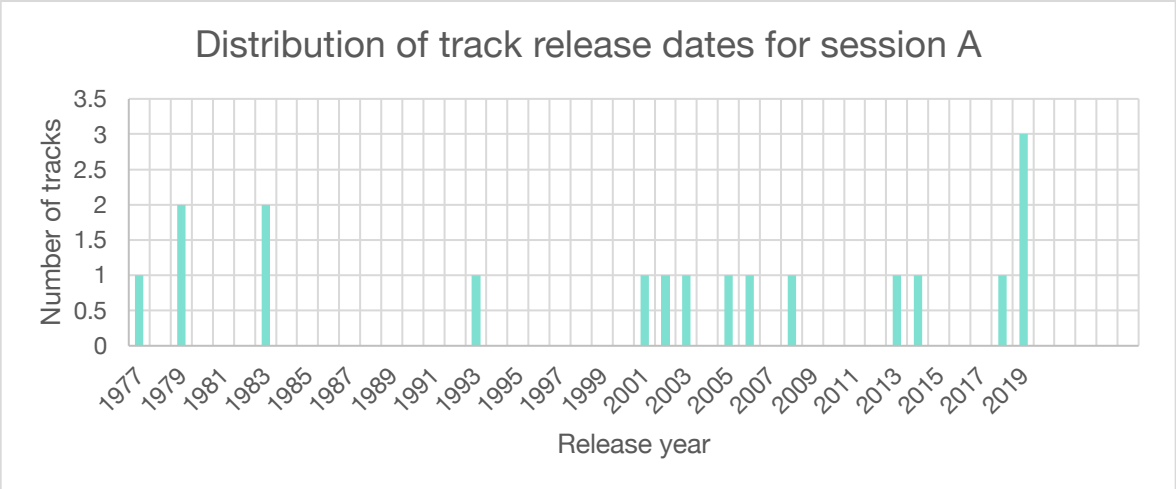

Figure 8.4 *- Distribution of track release dates for session A*

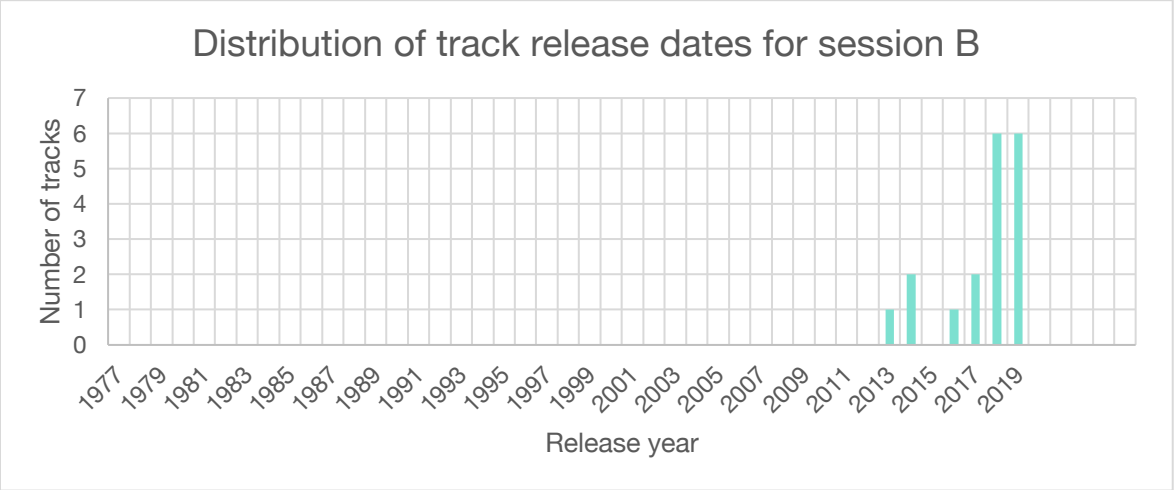

Figure 8.5 *- Distribution of track release dates for session B*

9. Discussion & Conclusion

The collection of data from the various evaluation techniques give a detailed insight into the solutions ability to improve collaboration in the social space. The product clearly facilitates the action of queuing music much more effectively than the original solution. However, there are still issues pivoting around product discoverability and the solutions ability to provide useful music suggestions and recommendations.

Much like what O'Hara et al (2004) found in their work with Jukola; we saw that some users could become distracted by the system. However, in this solution, turntable is not the distraction but acts a facilitator to move focus to other apps, specifically social media. It is unclear at this moment if this distraction has a negative impact on the collaboration and overall social experience.

The attendee's self-esteem is still large factor to the queuing process, turntable shows signs of positive effects on the individual queuing music. This is largely due to the increased discussion and gamification of the listening process, lower cost of queuing tracks and the anonymity of the action.

Users seem more experimental with their music choice with the new-found anonymity. Discussions about music choice and guessing games provide the queuing attendee with an option to take ownership of a queued track if the community is showing signs of acceptance.

The 'how might we questions' from chapter 4.4 sate that there is a need for a wide variation of music. When analysing the playlists from the recorded sessions we can see that without any restriction to the music that can be chosen by attendees, this solution meets this need. Interestingly, this issue is solved just by creating an open access queue. This asks the question of is the degree of music diversity is related to the number of contributors?

The above data clearly shows not only that more requests are made but also more attendees are making requests. As a result, it seems that music discoverability is also improved, with an increase of 500% in tracks added to individuals' libraries.

The development process of this solution had a quick turnaround with only 4 months spent on the main implementation. Release 1.1 is the most recent version at the time of this report and still has large areas for improvement and potential to address other issues discovered in the research chapters.

With the statistical results it is clear that this project has a positive impact on the social contexts in which it is used. This impact indicates that development shouldn't stop after this project, and could grow further into a commercial product. During the post event interviews many attendees shared their thoughts on the apps usefulness and how it could 'make money'.

New and interesting behaviour is also starting to occur even with the small amount contexts observed. This shows that the product has the ability to not only improve the social space but also innovate it to bring new abilities that were not as possible before. Continuing to study behaviour is essential to understand if this new behaviour has a lasting positive or negative impact to the overall social event.

With the small amount of data that was collected throughout the first few tests it seems that the product could also facilitate further studies into how people use music in social contexts. We've already discovered a relationship between age and release data that goes beyond a simple hypothesis of 'older people queue older music'.

The project as a whole was very comprehensive and took a holistic approach to understand the issues of the environment in which it studies, resulting in a human centred solution that solved real issues. Not all requirements derived from the research are met, however, this list was very ambitious, more so than the project was capable of achieving. The requirements that weren't met and findings from the evaluation give insight into potential future work.

Ideally this project inspires others with the value that can be derived from enhancing attendee experience and how HCI methodologies such as design thinking can truly attribute to a project's success or failure. In understanding the human problem through observation and empathy this project may have seen a vastly different outcome. The learnings from the early research were valuable tools throughout the project and were continuously referred back to.

To download and test this application directly from TestFlight please use the link below.

https://testflight.apple.com/join/BtaB5mjS

10. Further Work

As indicated by the findings and evaluation, much more work can be done in this area of research, both with the product and the analysis of the data from the sessions. Initially the bugs and errors that were still within the product should be addressed and tested.

The next logical steps for the product would include; making changes based from the ethnographic findings, submitting to the app store for public availability and to make the system more available for other OS's and devices.

Meanwhile, further testing should be taken out to collect more data to further support the hypothesis and discover more behaviour. These findings will direct the changes made to the system in updates.

To maximise the amount of data collected for analysis, promotion of the public app should be considered to entice users to use it in a variation of natural environments. This should be done via social media and word of mouth initially.

Continuing to develop features from the list of requirements would also provide more data about different aspects of the solutions ability to improve these social spaces. One major feature that should be researched further and implemented is a selection of ordering algorithms which sort music based on their audio features (key, tempo and time signature).

Alongside all of this future work, further research can include work focused on music phycology to further understand how music affects individuals and their phycological state. This research could indicate new areas for improvement of experimentation that might not be considered when observing from an outside perspective.

Acknowledgements

Thanks to the one-hundred and fourteen anonymous contributors throughout the observations, user testing, interviews and surveys. The academics and authors powerful and inspiring work mentioned throughout influenced this project immensely, together with the anonymous participants their contributions made this project insightful and enjoyable.

References

Apple. (2019). Apple Developer Documentation. Found at https://developer.apple.com/documentation/

Apple. (2018) Apple Music API [on-line]. Available from https://developer.apple.com/documentation/applemusicapi [Accessed 6/12/18]

Apple (2019) Configuring the Cells for your Table [on-line]. Available from https://developer.apple.com/documentation/uikit/views_and_controls/table_views/configuring_the_cells_for_your_table [Accessed 22/04/19]

Danielsen, A. (2017). Music, media and technological creativity in the digital age. Norge,
Institutt for musikkvitenskap, Universitetet I Oslo.

Google. (2019). Add Firebase to your iOS Project. Found at https://firebase.google.com/docs/ios/setup

Gothelf, J. Seiden, J (2016). Lean UX Designing Great Products with Agile Teams (Second edition). O'Reilly Media, Inc. California USA.

Gummerson, R. (2016) Regarding Swift build time optimizations. Found at https://tinyurl.com/yctnlplc

Hagen, A. N. (2015). *Using Music Streaming Services: Practices, Experiences and the Lifeworld of Musicking*. PhD dissertation. University of Oslo.

IDEO. (n.d.). How Might We. Found at http://www.designkit.org/methods/3

Lethbridge, T. Laganiere, R. (2005). Object-Orientated Software Engineering: Practical Software Development using UML and Java. McGraw-Hill Inc. New York USA

Liu, K. T. Reimer, R. A. (2008). 'Social Playlist: Enabling Touch Points and Enriching Ongoing Relationships Through Collaborative Mobile Music Listening', *MobileHCI '08 Proceedings of the 10th international conference on Human computer interaction with mobile devices and services.* Amsterdam, The Netherlands, September 02 - 05, 2008. pp. 403-406.

Maslow, A. H. (1943). A Theory of Human Motivation. Psychological Review, 50(4), 370-96.

Maslow, A. H. (1954). Motivation and Personality. New York: Harper and Row.

Nakano, Y, I. Reinstein, G. Stocky, T. Cassel, J. (2003). “Towards a Model of Face-to-Face Grounding”. Amsterdam The Netherlands, John Benjamin’s Publishing Co.

Norman, D. (2004). Emotional Design, Why We Love (or Hate) Everyday Things. Basic Books. Cambridge USA

O’Hara, K. Lipson, M. Jansne, M. Unger, A. Jeffries, H. Macer, P. (2004). ‘Jukola: democratic music choice in a public space’, *DIS '04 Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods* Psychological Review, 50*,, and techniques.* Cambridge, MA, USA, August 01 – 04, 2004. Pp. 145-154

Pagan, B. (2015). Lean Start-up MVP: How To Make Meaningful Products. Found at https://brianpagan.net/2015/lean-startup-mvp-how-to-make-meaningful-products/

Preece, J. Rogers, Y. Sharp, H. (2015). Interaction Design beyond human-computer interaction (Fourth edition). John Wilet & Sons Ltd. West Sussex United Kingdom.

Spotify. (2018). Company Info. Found at: https://newsroom.spotify.com/company-info/

Spotify. (2019). Platform Documentation. Found at https://developer.spotify.com/documentation/

SubtvU. (2018) Subtv Playlister – Available Now [on-line]. Available from *http://www.sub.tv/playlister/* [Accessed 6/12/18]

Superpowered. (2018). Music Analysis for Android, iOS, macOS, tvOS, Linux and Windows [on-line] Available from *https://superpowered.com/music-analysis-bpm-key-detection-waveform* [Accessed 6/12/18]

Visoky O’Grady, J. Visoky O’Grady, K. (2017). *A Designer’s Research Manual 2nd Edition.* Quarto Publishing Group USA Inc.

Blank Page