



Tapes - A music application that utilizes the multi-sensory qualities of mainstream music services to create an immersive user experience.

Final Year Project Report

DT228

BSc in Computer Science

Dylan Kieran

Cindy Liu

School of Computing
Technological University Dublin

10/04/2019

Abstract

With the ever-growing popularity of music streaming services grows a need for a service that provides an all in one library of music, instead of having to switch service for a song that doesn't exist on one platform.

Every service is different, they provide a different user interface, user experience and music library. *Tapes* will provide an all-inclusive library for users, so they can stream every and any song they want, all while providing a seamless, coherent user interface and user experience. *Tapes* users can benefit from music videos from YouTube and higher song quality from Spotify all in one place. They can also browse and listen to local radio stations and discover music events that are upcoming in their area.

The *Tapes* application will be developed using Node.js, HTML and CSS. Data will be retrieved from several APIs including YouTube, Spotify and Ticketmaster. The data will then be manipulated and output to the users. Playback will be implemented for both Spotify and YouTube allowing users to both play and queue songs/videos as well as listen to playlists and view recommendations.

Declaration

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

Signed:

A handwritten signature in blue ink that reads "Dylan Kieran". The signature is written in a cursive style. Below the signature is a horizontal line.

Dylan Kieran

10/04/2019

Acknowledgements

I'd like to thank my project supervisor Cindy Liu for taking the time to meet with me each week. I would also like to thank Damian Gordon for his help and advice throughout the project process over the last few months. Lastly, I'd like to thank my family and friends for their continued support.

Table of Contents

Abstract	2
Declaration	3
Acknowledgements	4
Table of Contents	5
Table of Figures	8
1. Introduction.....	10
1.1 Project Background	10
1.2 Project Description	10
1.3 Aims & Objectives.....	11
1.4 Project Scope	11
1.5 Structure of Document	12
2. Research	14
2.1 Background Research	14
2.2 Alternative Existing Solutions	17
2.2.1 Last.fm	17
2.2.2 Sonos	18
2.2.3 Autobeat	19
2.3 Music Sources Researched	20
2.3.1 YouTube.....	20
2.3.2 Spotify.....	20
2.3.3 Apple Music	21
2.3.4 Soundcloud	22
2.4 User Survey.....	22
2.5 Literature Researched	27
2.6 Resultant Findings	28
2.7 Technologies Researched	29
2.8 Technology Review	32
2.9 Conclusions.....	35
3. System Design.....	36
3.1 Introduction.....	36
3.2 Methodologies.....	36

3.3 Review Methodologies	39
3.4 Technical Architecture.....	40
3.5 Source Code Layout	42
3.6 Functionality Overview	45
3.6.1 Tapes Functionality.....	45
3.6.2 YouTube Functionality.....	46
3.6.3 Spotify Functionality.....	48
3.7 User Interface and User Experience	48
3.7.1 Design Iterations.....	49
3.7.2 Low Fidelity Prototype.....	51
3.7.3 Horizontal Prototype	52
3.8 Conclusions.....	55
4. System Development	56
4.1 Introduction.....	56
4.2 Front-End Development	56
4.2.1 Spotify.....	56
4.2.2 YouTube.....	63
4.2.3 Radio.....	69
4.2.4 Events	69
4.3 Back-End Development	72
4.3.1 Authentication Server.....	72
4.4 Challenges.....	73
4.4.1 New Technologies	73
4.4.2 API Limitations.....	74
4.4.3 Time Management	75
4.4.4 User Acceptance.....	75
4.5 Conclusions.....	75
5. Testing and Evaluation	76
5.1 Introduction.....	76
5.2 System Testing.....	76
5.2.1 Test Levels	76
5.2.2 Test Methods.....	78
5.2.3 Test Plans.....	79

5.2.4 API Testing	84
5.2.5 User Testing	85
5.3 Prototype	86
5.4 User Evaluation	88
5.5 Conclusions	91
6. Project Plan	92
6.1 Introduction	92
6.2 Gantt Chart	92
6.3 Requirements	93
6.4 Initial Proposal	96
6.5 Changes to the Proposal	97
6.6 Conclusions	98
7. Conclusions and Future Work	99
7.1 Introduction	99
7.2 Conclusions	99
7.3 Future Work	101
8. Bibliography	105
9. Appendix	109

Table of Figures

Figure 1: RIAA U.S. Music Industry Revenue 2018	15
Figure 2: Last.fm Charts Screen [47]	18
Figure 3: Sonos Application Home Screen [48]	19
Figure 4: Autobeat Desktop Home Screen and Mobile Application [15]	20
Figure 5: User Survey Results Q1.....	22
Figure 6: User Survey Results Q2.....	23
Figure 7: User Survey Results Q3.....	23
Figure 8: User Survey Results Q4.....	24
Figure 9: User Survey Results Q5.....	24
Figure 10: User Survey Results Q6	25
Figure 11: User Survey Results Q7	25
Figure 12: User Survey Results Q8	26
Figure 13: User Survey Results Q9	26
Figure 14: Waterfall Methodology	36
Figure 15: Test Driven Development.....	38
Figure 16: Scrum Framework	38
Figure 17: Technical Architecture.....	41
Figure 18: Tapes Server Source Code Layout	42
Figure 19: Tapes Client Source Code Layout	42
Figure 20: Use Case 1 – Tapes Functionality	46
Figure 21: Use Case 2: YouTube Functionality	47
Figure 22: Use Case 3 – Spotify Functionality	48
Figure 23: Tapes Design First Iteration.....	49
Figure 24: Tapes Design Second Iteration	50
Figure 25: Tapes Design Final Iteration	50
Figure 26: Low Fidelity Prototype Screen 1.....	51
Figure 27: Low Fidelity Prototype Screen 2.....	51
Figure 28: Horizontal Prototype - Discovery	52
Figure 29: Horizontal Prototype - Spotify.....	52
Figure 30: Horizontal Prototype - YouTube.....	53
Figure 31: Horizontal Prototype - Library.....	53
Figure 32: Horizontal Prototype – Spotify Player	54
Figure 33: Horizontal Prototype – YouTube Player	54
Figure 34: Initialisation of Spotify Player Source Code	57
Figure 35: Spotify Playback JSON Data.....	58
Figure 36: Tapes Player	58
Figure 37: Parsing JSON data and inserting into DOM Source Code.....	59
Figure 38: Tapes Spotify Search and Results	60
Figure 39: Play Source Code	61
Figure 40: Tapes Spotify Playlists	62
Figure 41: Tapes Spotify Recommendations	63
Figure 42: Tapes YouTube	64
Figure 43: Tapes YouTube Search Source Code	65
Figure 44: Tapes Queue a Video.....	66
Figure 45: Queue Source	67
Figure 46: Tapes Video Queue	68
Figure 47: Radio Source Code.....	69
Figure 48: Events Source Code	71
Figure 49: Tapes Events	71

Figure 50: Authentication Server	73
Figure 51: Postman - YouTube Top Music Videos	85
Figure 52: Tapes Prototype - Spotify	87
Figure 53: Tapes Prototype - YouTube	87
Figure 54: Tapes Prototype - YouTube Player	88
Figure 55: User Evaluation - Q1	88
Figure 56: User Evaluation - Q2	89
Figure 57: User Evaluation - Q3	90
Figure 58: User Evaluation - Q5	90
Figure 59: Project Plan - Gantt Chart	93
Figure 60: Tapes Database	102
Figure 61: Tapes Spotify Top Songs	109
Figure 62: Last.fm Top Songs	110
Figure 63: Last.fm Top Artists	111

1. Introduction

1.1 Project Background

Music streaming now makes more money than physical CDs, digital downloads, and licensing deals combined. Based on the “Mid-Year 2018 RIAA Music Revenues Report” music streaming accounts for 75% of the revenue for the US music industry. [39]

As streaming has surged in popularity, so has the number of companies that have stepped in to offer their own services. Spotify, YouTube Music, Apple Music, Pandora, Amazon Music, Tidal, Deezer, Napster, Slacker Radio and Google Play Music are only some of the many companies that are taking over the music streaming industry. [40] These services offer largely the same core functions. While some focus on radio and others on-demand streaming, they offer the ability to stream and listen to all music on any device. Most services work off a freemium business model meaning the service offers both free and premium services. The freemium business model works by offering simple and basic services for free for the user to try, and advanced or additional features at a premium. [36]

1.2 Project Description

With the ever-growing popularity of music streaming services grows a need for a service that provides an all in one collection for a new age music library. Every service is different, they provide different music selection, user interface and user experience. Current platforms offer either music playback or video playback, *Tapes* will combine these services by offering the user the choice of both. Current platforms offer limited music choice due to licensing, *Tapes* will provide an all-inclusive library by combining these services, so users can stream every and any song they want. The focus of *Tapes* is to provide the user with a seamless, coherent user interface and user experience.

This project will attempt to utilize the multi-sensory qualities of mainstream music services to create an immersive experience. Similar to how festivals use both audio and visuals to stimulate a person’s senses, *Tapes* will employ the use of various streaming service APIs to provide the same stimulation.

Tapes users can benefit from music videos from YouTube and higher song quality from Spotify all in one place. They can also browse and listen to local radio stations and discover music events that are upcoming in their area.

The *Tapes* application will be developed using Node.js, HTML and CSS. Data will be retrieved from several APIs including YouTube, Spotify and Ticketmaster. The data will then be manipulated and output to the users. Playback will be implemented for both Spotify and YouTube allowing users to both play and queue songs/videos as well as create and listen to playlists and view recommendations.

1.3 Aims & Objectives

The primary aim of this project is to design and develop a rich web application that incorporates the multi-sensory qualities of mainstream music streaming services (Spotify and YouTube) while maximising the users' experience. The application will feature the most prominent features of both services in order to give users a cohesive application. The design of the application will be user focused in order to provide the best user experience. Some of the main objectives of this project are as follows:

- Research and locate available music streaming APIs and determine if they can be incorporated into the project.
- Research areas of rich web technology and determine what technologies would be best suited for the project.
- Explore the area of music streaming in order to include successful features of these sites into the project.
- Carry out extensive user testing to incorporate user feedback and suggestions into the project.
- Design and develop the application, considering all research.
- Plan and execute usability testing of the application to get feedback on the user interface and user experience.

1.4 Project Scope

Tapes acts as an alternative application to view and listen to music and music videos. The goal of *Tapes* is not to replace the current music streaming services but instead incorporate these services into the one application for ease of use. By incorporating these different services into the one application it allows the users to have an even larger music library with alternative ways to both listen and view music. The application will aim to improve the user's experience by providing a simple yet effective user interface with easy to use

functionality. The application will focus on user experience, therefore extensive user testing will be carried out throughout the project's lifecycle.

1.5 Structure of Document

The following is a summarization of the sequential chapters of the dissertation. Each chapter will focus on a specific area of the project.

Chapter 2: Research

Chapter two outlines the research undertaken throughout the lifecycle of the project. The concept of the application is formed, and initial ideas are established. Alternative existing solutions to the application are discovered and critiqued. Finally, the technologies necessary to achieve these requirements are researched.

Chapter 3: Design

This chapter introduces the project management approach for the application. The applications main use cases are identified and described. The application's user interface components are designed in detail with an emphasis on providing a better user experience over existing solutions.

Chapter 4: System Development

Chapter four focuses on the development of the system. The implementation of the applications main features is discussed in detail, outlining how and why certain features were implemented in such a way.

Chapter 5: Testing and Evaluation

This chapter focuses on the testing and evaluation of the project. The various levels and methods of testing are outlined and explained. The application evaluation is outlined and discussed.

Chapter 6: Project Plan

Chapter six contains the project plan. This section focuses on the time management and lifecycle of the project. The project Gantt chart is included as well as how the project changed from the initial proposal to the final product.

Chapter 7: Conclusion

Chapter seven is the conclusion of the dissertation. This chapter rounds out the dissertation and discusses possible future work that could be carried out on the project. Future work areas include aspects of user suggestions that could not be implemented in the given time constraints of the project.

Chapter 8: Bibliography

Chapter eight contains the bibliography for the project and lists several websites, books, journals and tutorials that were researched over the course of the project.

Chapter 9: Appendix

The appendix contains features that were implemented but not incorporated into the final build.

2. Research

2.1 Background Research

Initial background research was conducted in the areas of music streaming and rich web applications.

Music Streaming

With the advent of the internet, broadband and individual smart phone and digital devices the traditional format of music distribution has changed. From the purchase of vinyl albums and singles, music distribution and acquisition transitioned through tapes, CDs and finally digital files. With this came the introduction of music streaming services.

The first source of digital music came in the form of MP3 audio files. As more and more people ventured on to the internet in the late 1990s the popularity of MP3 files grew. [36] MP3 files became the new source for music. It didn't take long for people to exploit the power of sharing these MP3 files. In 1999 Napster was founded, a website that allowed peer to peer file sharing, which gave users access to millions of songs for free. The Recording Industry Association of America (RIAA) took legal action against Napster. They filed a lawsuit against it for the unauthorized distribution of copyrighted material. After a long court battle, the RIAA obtained an injunction from the courts that forced Napster to shut down its network in 2001. [37]

Two years after the release of Napster, Apple announced the launch of the iTunes music store. This allowed users to purchase music online and burn CDs to MP3 files and add these files to their iTunes library. Apple's release of iTunes came hand in hand with the release of the first iPod, capable of playing and holding a vast library of music on a small handset device. [38]

Between the release of Napster and iTunes music store, another key aspect of audio streaming was released, Last.fm. Last.fm used machine learning principles that tracked listening habits, made comparisons to other user data, and gave music recommendations. [36]

In 2005, Pandora was released. Pandora used similar algorithms to Last.fm however they used them as a way to reinvent radio. Instead of listening to a radio station helmed by a DJ, Pandora allowed users to listen to their own custom station. Pandora was one of the first to introduce a freemium service by which users can pay a subscription fee to remove ads and endless streaming of music. [36]

In 2008, Spotify was released. Spotify put all these things together. It provided the ability to stream music no matter the bandwidth, a massive online music library and music recommendations for the price of \$10 per month. [36] Spotify opened a new door for larger companies to create their own streaming services. Since the release of Spotify, Apple, Google and Amazon have all developed their own streaming services.

Music streaming now makes more money than physical CDs, digital downloads, and licensing deals combined. Based on the “Mid-Year 2018 RIAA Music Revenues Report” music streaming accounts for 75% of the revenue for the US music industry. [39]

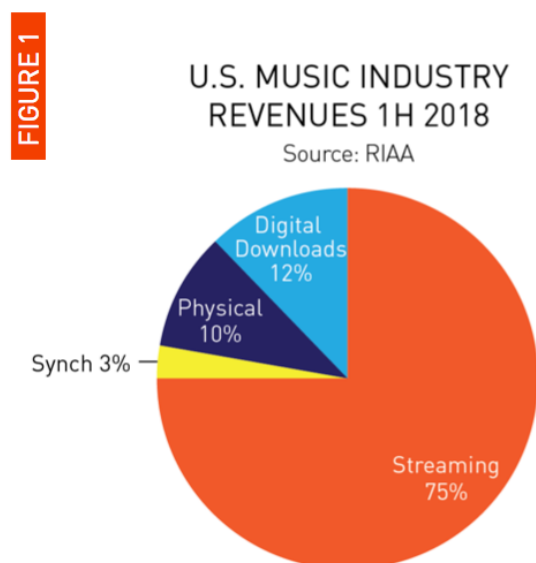


Figure 1: RIAA U.S. Music Industry Revenue 2018

Rich Web Application

A web application is a software program that runs on a web server. The end user interacts with the server application through a user interface on a remote device, usually a browser.

There are many possible architectural approaches used to divide the business and presentation logic responsibilities between client and server. These have evolved over time as client devices and networks have become more powerful and as users have demanded a more intuitive user experience.[45]

Originally the web was a document management system with one language before diverging into three separate languages over time, those three separate languages being HTML (Hyper Text Mark-up Language), CSS (Cascading Style Sheets) and JS (JavaScript). These three separate languages are responsible for providing different functionality for a web application, namely that of structure, styling and behaviour. This division of responsibility was considered to be a separation of concerns, a design principle of having distinct responsibilities in each part of the design with as little overlap as possible. [13] However, there are a few obvious problems with this separation of concerns principle, namely it being not strict by language and that it does not offer a useful level of abstraction.

The first criticism, not strict by language, meaning that all three languages can perform some or all of the functions of the two others, so this separation is not strictly enforced. HTML can embed asynchronous behaviour (with aid of JavaScript), CSS can load and unload DOM content and animate elements asynchronously and JavaScript DOM API allows JavaScript developers to completely replace HTML and CSS with a pure JavaScript approach. Another argument against separating by structure/style or behaviour is that it does not really offer a useful level of abstraction. [13] For example, in an input form with multiple input fields, ideally the developer would like to include all of the supporting logic within that form element. However, separation of concerns dictates the developer needs to split up this functionality which does not offer the cleanest most maintainable abstraction for this.

A rich web application is the latest evolution in a series of iterative design changes that web applications have undergone in the decades since the web first appeared. A rich web application loosely follows a separation of concerns principle but instead separates the structure of the application into a client server paradigm. [45] A rich web application is a web application where the client has most, if not all, of the responsibility for implementing the presentation layer logic, i.e. the user interface and user experience. Rich web applications see the significant use of JavaScript programming in the client browser often employing a lean data transfer interface with the server.

2.2 Alternative Existing Solutions

The concept of combining multiple music streaming services into one application is something that hasn't been explored to a large extent, only three main applications have done so.

2.2.1 Last.fm

Last.fm is a music website founded in the United Kingdom in 2002. The service started out as a radio streaming service but was discontinued in 2014. The service then expanded and introduced the ability to access a large catalogue of stored music this was later removed entirely and replaced by links to YouTube and Spotify. [25] Last.fm implements a music recommender system called "Audioscrobbler", this system is used to build a detailed profile of each user's music taste by recording details of the tracks the user listens to. These details are then transferred to the Last.fm's database either via the music player itself (such as Spotify, Deezer, Tidal or MusicBee) or via a plug-in installed into the user's music player. This data is then compiled and displayed on the user's profile page. [26]

The recommender system Last.fm uses is quite unique and performs very well however, it can seem a bit invasive for some users as the tool has to be downloaded and run in the background of the user's system to provide results. With this in mind, an alternative solution will be taken for the *Tapes* application.

Last.fm includes a subscriber feature, meaning for €3 per month users have access to ad-free browsing and a range of additional listening reports and recommendations. [41] While free users have access to the majority of the same features as subscribed users, they have a large number of advertisements that are displayed on screen, that cannot be blocked by common ad blockers. This leads for a cluttered and overwhelming user interface.

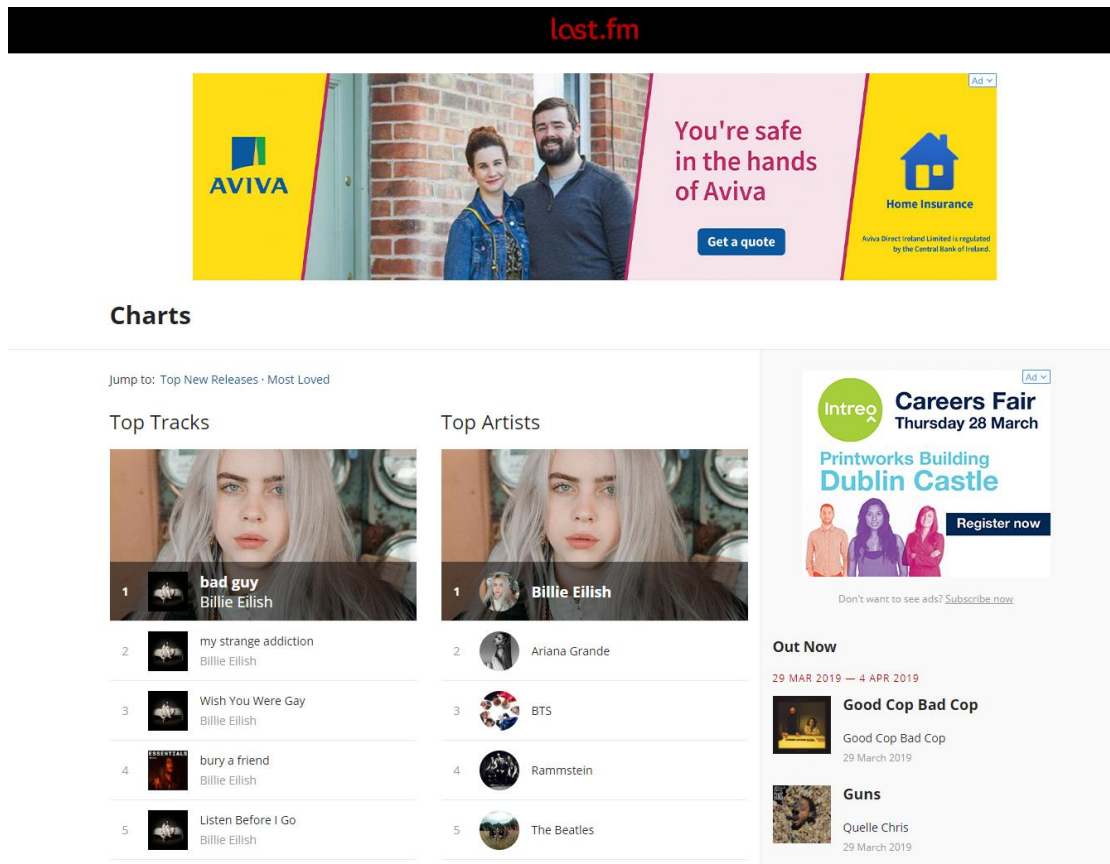


Figure 2: Last.fm Charts Screen [47]

The Spotify player on Last.fm requires a user to connect a device to the application. Once a song is selected it will begin to play on the device, not in the browser on the Last.fm website. This can confuse and frustrate users that want to listen to the music within the browser not on their device. A large portion of songs listed on the website aren't playable even though the songs are available on the Spotify application.

2.2.2 Sonos

Sonos is an American electronics company that produce home sound systems. They have developed a cross platform music player application for all devices that provides the ability to stream music directly to their Sonos devices. The Sonos app allows users to listen to music from the most popular streaming services such as Apple Music, Spotify, Soundcloud, Google Play Music, Deezer and Tidal. The app allows for simple navigation, easy room control, quick access to the music and the ability to search all services at once [14]. At a glance, the Sonos app sounds a near perfect example of what this project should be. However, the Sonos

application is only available to users who have purchased a Sonos sound system and only provides the ability to stream music to that device.

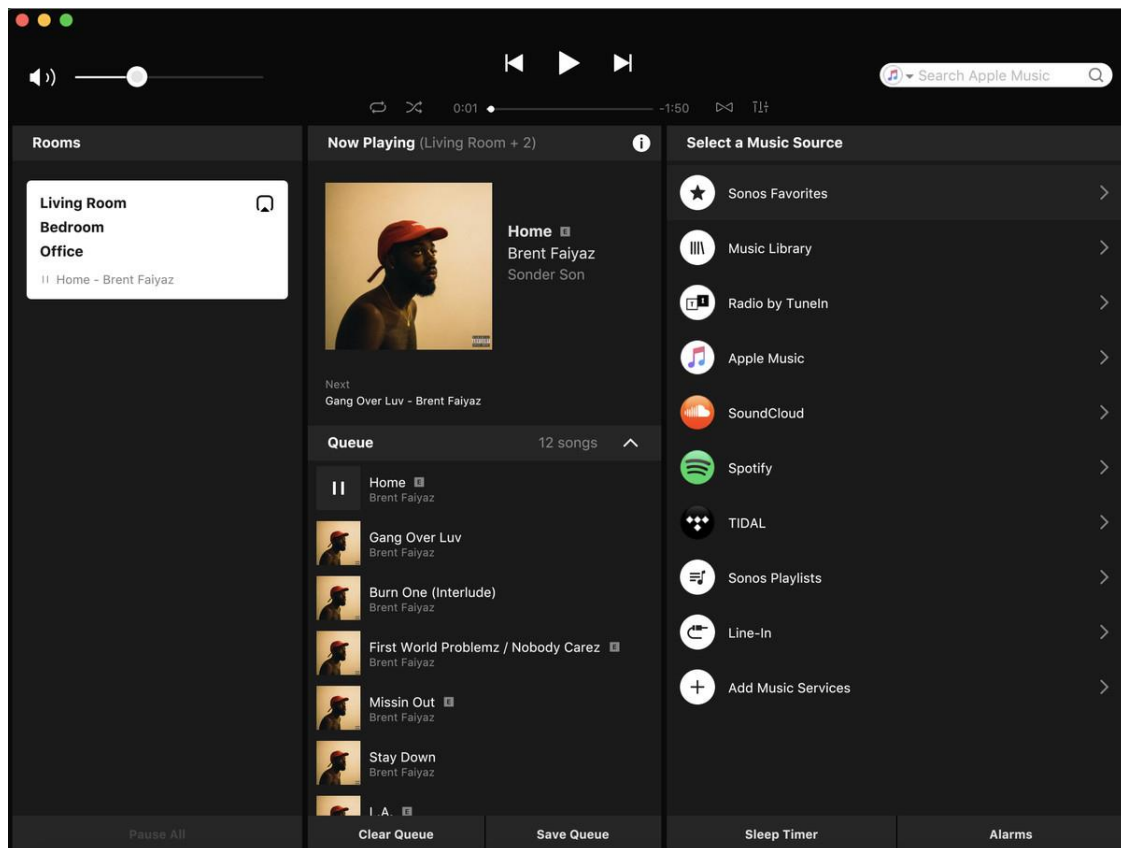


Figure 3: Sonos Application Home Screen [48]

2.2.3 Autobeat

Autobeat is a music player that unifies music from YouTube, SoundCloud and a user's system library into a single application. Autobeat's goal was to create a unified music player that allowed users to search for songs everywhere with one unified search, create mixed playlists and add songs to the user's library with one click [15]. The Autobeat player had a lot of potential to fill the gap in the market for a unified music player however, development of the application seems to have halted as they have removed the application from the Google Play store and have not released their desktop application.

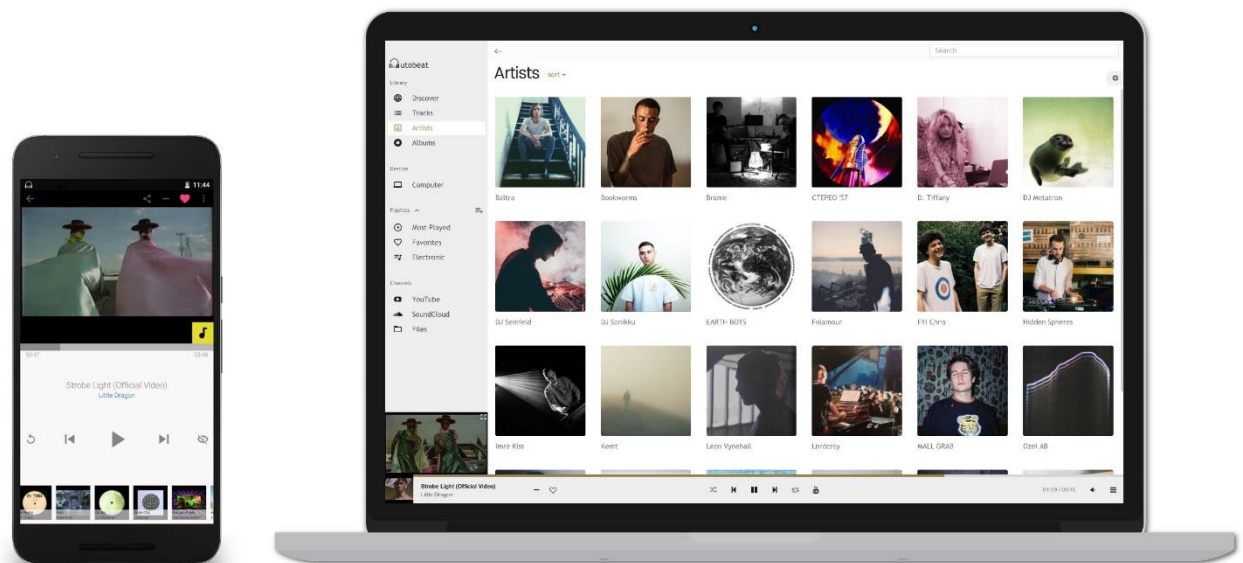


Figure 4: Autobeat Desktop Home Screen and Mobile Application [15]

2.3 Music Sources Researched

2.3.1 YouTube

YouTube is the world's largest video sharing platform. YouTube provides the ability for content creators such as music artists and producers to release their music videos on the service. Since release YouTube has grown and has become recognized as one of the largest sources of free music streaming. YouTube recognized this and in 2015 released YouTube Music. YouTube Music is a separate subscription-based service (positioned as a more direct competitor to Apple Music and Spotify), offering ad-free and background/audio only streaming, and downloading for offline playback, for music content on YouTube. [27]

YouTube offer several APIs including a Video Player API and an Analytics API. The Video Player API allows for the use of an embedded player that allows for video playback directly in an application. The Analytics API allows for the retrieval of information such as viewing statistics, popularity metrics and more from YouTube videos and channels. [1]

2.3.2 Spotify

Spotify is an online music streaming service that gives you access to millions of songs and podcasts from artists all around the world. Spotify is available in 65 countries and has over 207 million active users. [30] Spotify has a library of over 30 million songs because of this

and the number of users actively using Spotify as their primary source of music streaming means that as the market leader it was an obvious choice to include the ability to listen to Spotify music in this research. [28]

Spotify acts on a freemium service meaning it has two tiers: free and premium. The free service limits the users access to certain features. [30] Spotify has experimented with their free service in the past. As of April 18th, 2018, Spotify's free service allows users to listen to on-demand music to whatever song they want an unlimited number of times, as long as the song is on one of the user's 15 personalized discovery playlists. This therefore limits the user's access to Spotify's features; users don't have the ability to search and play any song they want or create their own music playlists. [31]

The Spotify API and SDK provide the ability for a user to search for songs, playlists and artists and view the metadata they contain. [2] The SDK allows users to use the application as alternate device, meaning users can connect their device to another device and control the playback on the secondary device, similar to how you can connect your phone to a speaker. [29] Only users that have a Spotify premium account can access features provided by the API and SDK.

2.3.3 Apple Music

Apple Music is a music and video streaming service developed by Apple Inc. Users can stream music to their devices on demand. [33] Apple Music is available in over 110 countries and has 56 million active users. [32] However, the service is strictly premium, meaning users are charged in order to access its features. [33] Apple music boasts a large music library of over 45 million songs along with a large number of music videos and concerts that are coupled with the music. [32]

MusicKit JS is an Apple Music SDK that allows users to play songs from Apple Music and their Cloud Library inside a developers JavaScript application. [4] In order to access and use its features a developer must obtain a developer key which can be obtained by signing up to Apple's developer program which is a yearly subscription of €99.

2.3.4 Soundcloud

SoundCloud is an online audio distribution platform and music sharing platform. Soundcloud allows users to upload and share their own music. [34] Soundcloud has over 190 million tracks from over 20 million creators. [35] Soundcloud is very popular for upcoming music artists and producers as it provides a free service that allows creators to share their work. Soundcloud contains millions of tracks that aren't available on any other service making it an important resource for any new application.

The Soundcloud API and SDK allow users to retrieve metadata information on songs and artists and provides the ability for audio playback.

2.4 User Survey

An initial survey was carried out in order provide necessary information for the design of the application. The survey consisted of nine questions about how and where people listen to music, what devices they use to listen to music, do they use a music streaming service, and what features do they like and dislike about these services. The survey was carried out on a sample size of 22 people. The results impacted the design of the project.

Question: Do you listen to music?

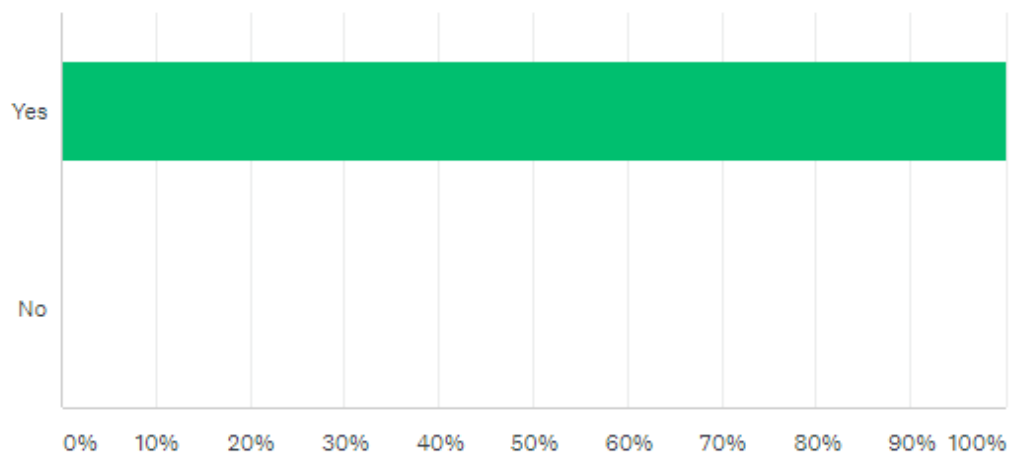


Figure 5: User Survey Results Q1

Question: How much time do you spend on average listening to music in a day?

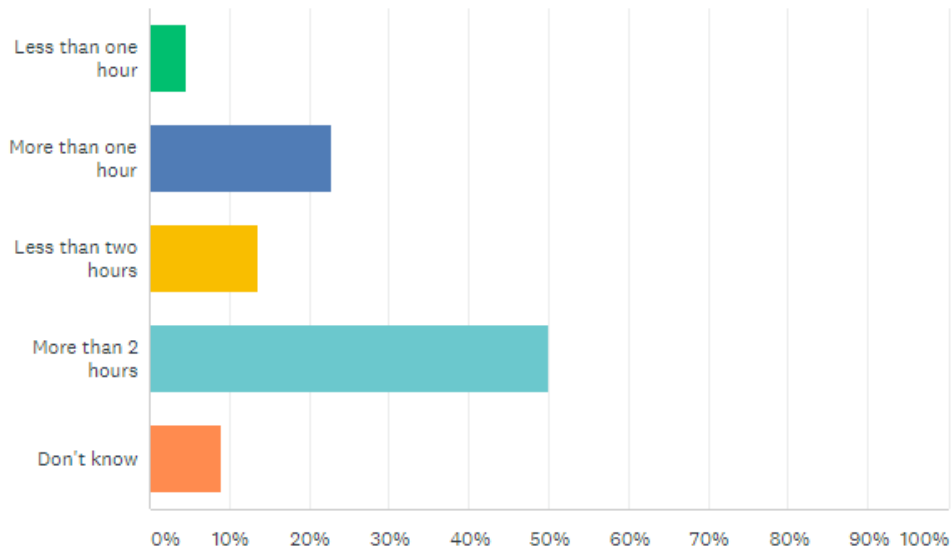


Figure 6: User Survey Results Q2

Question: In what place(s) do you listen to music?

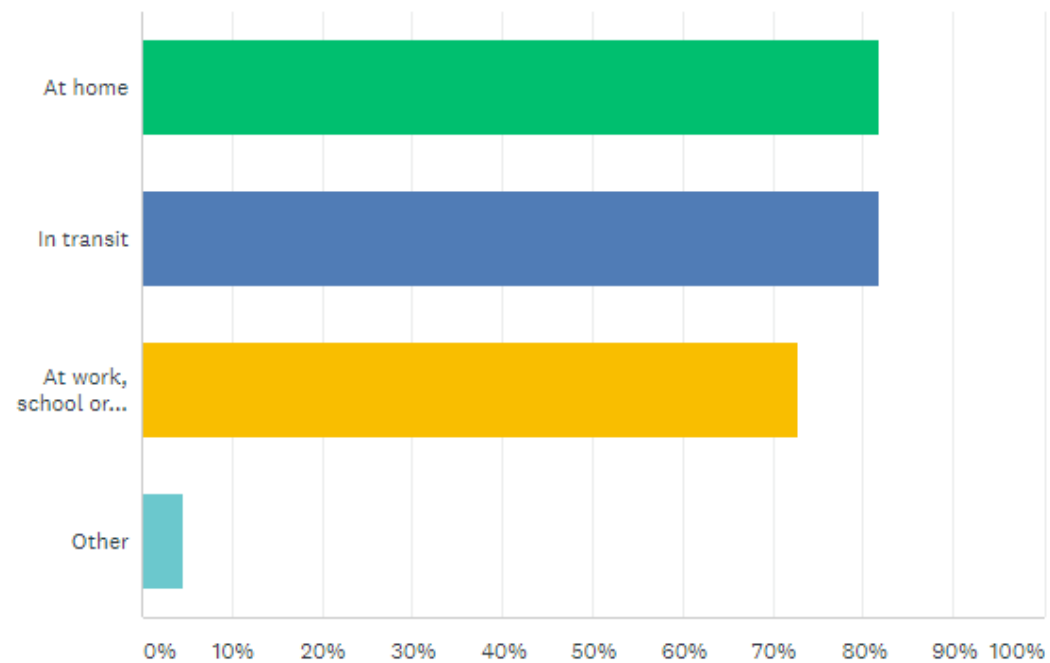


Figure 7: User Survey Results Q3

Question: On what device(s) do you listen to your music on?

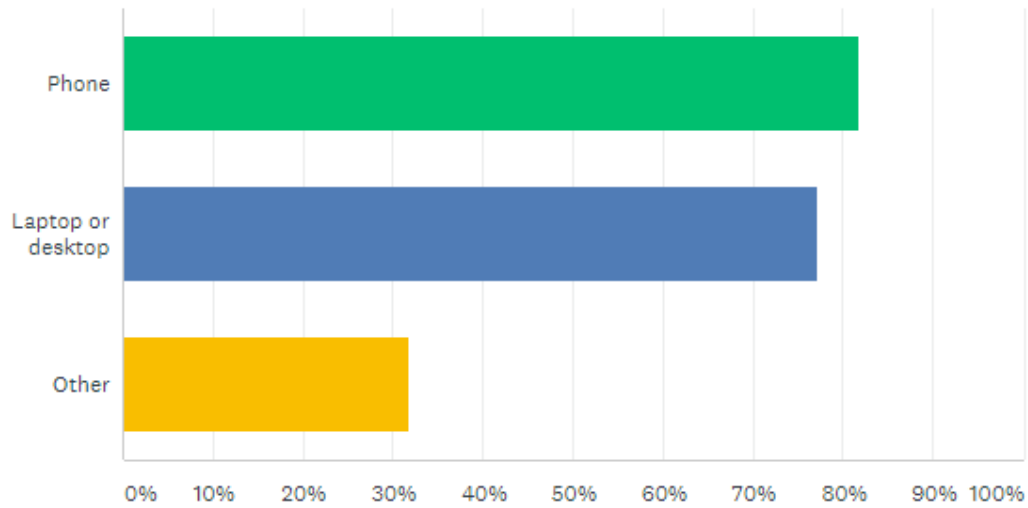


Figure 8: User Survey Results Q4

Question: Do you use a music streaming service for your music?

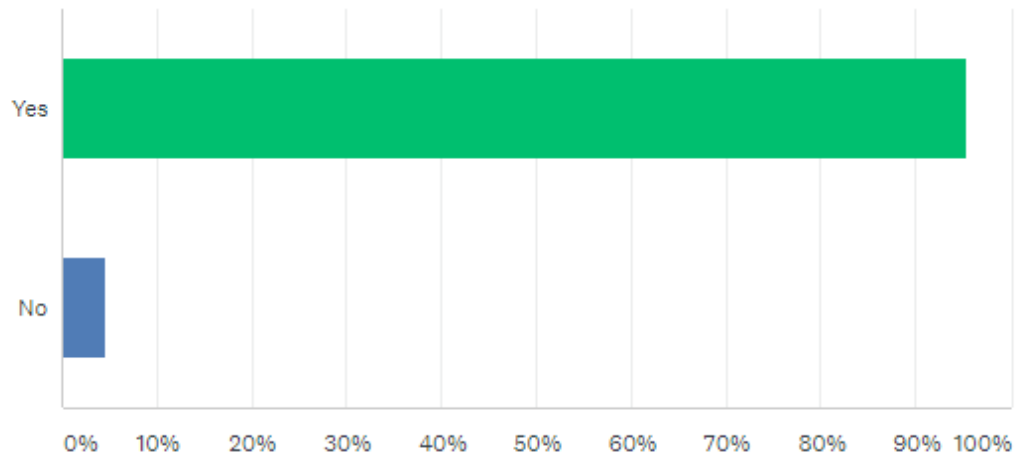


Figure 9: User Survey Results Q5

Question: What music streaming service(s) do you use?

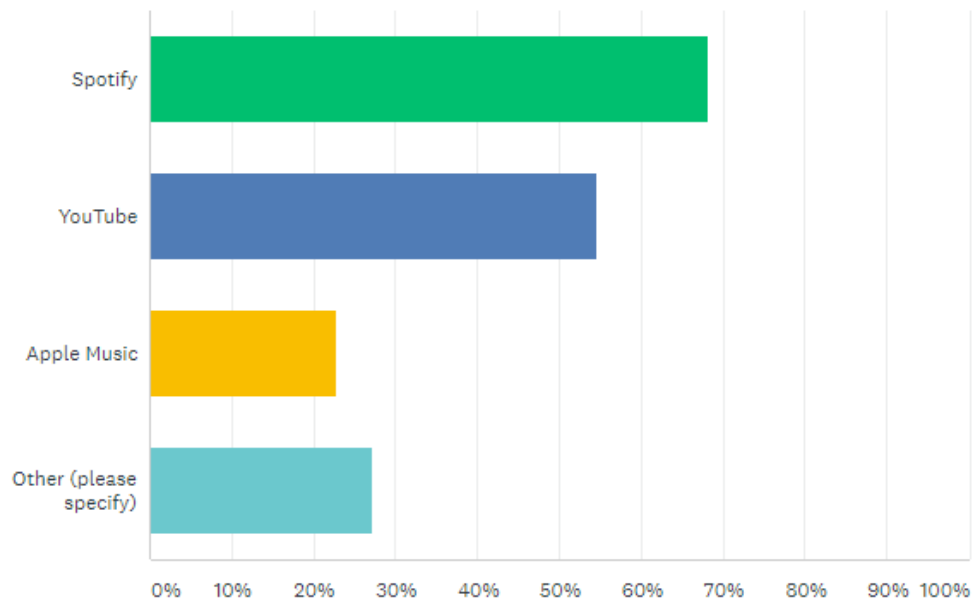


Figure 10: User Survey Results Q6

Question: What feature(s) of these services appeal to you?

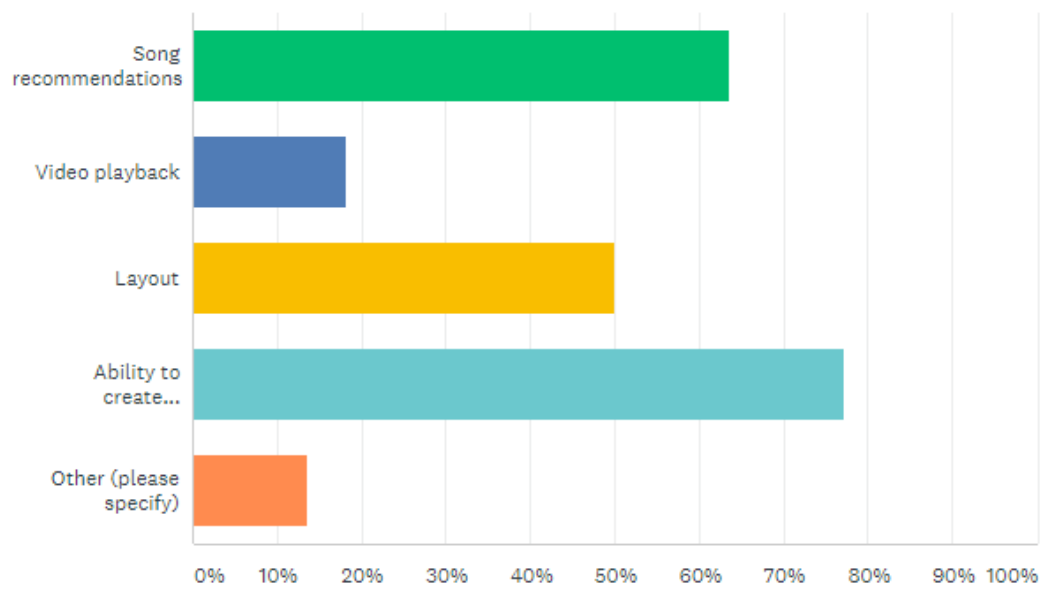


Figure 11: User Survey Results Q7

Question: What feature(s) of these services do you dislike?

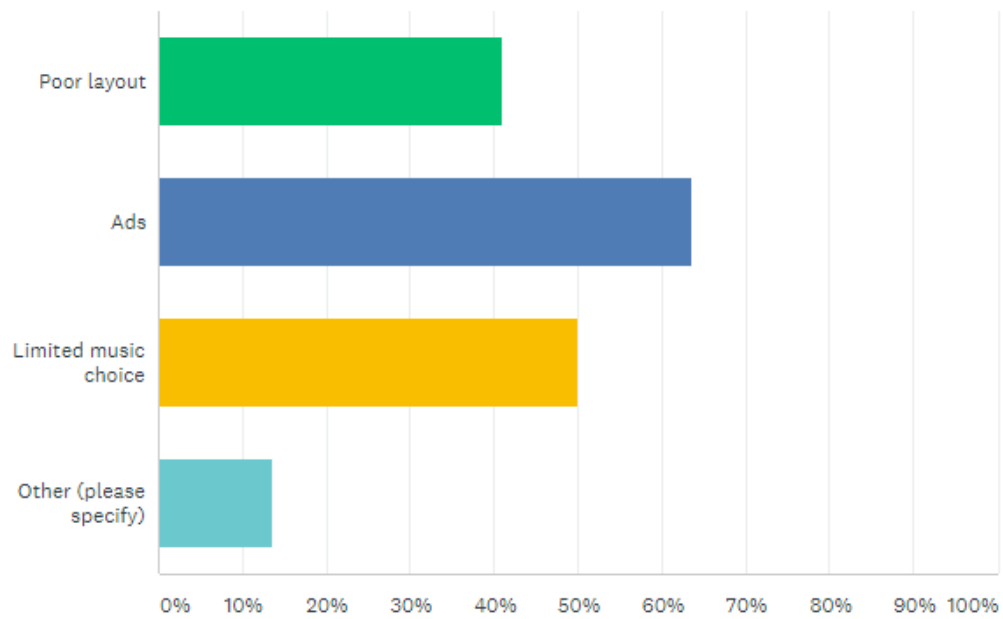


Figure 12: User Survey Results Q8

Question: Would you use an application that combines these music services?

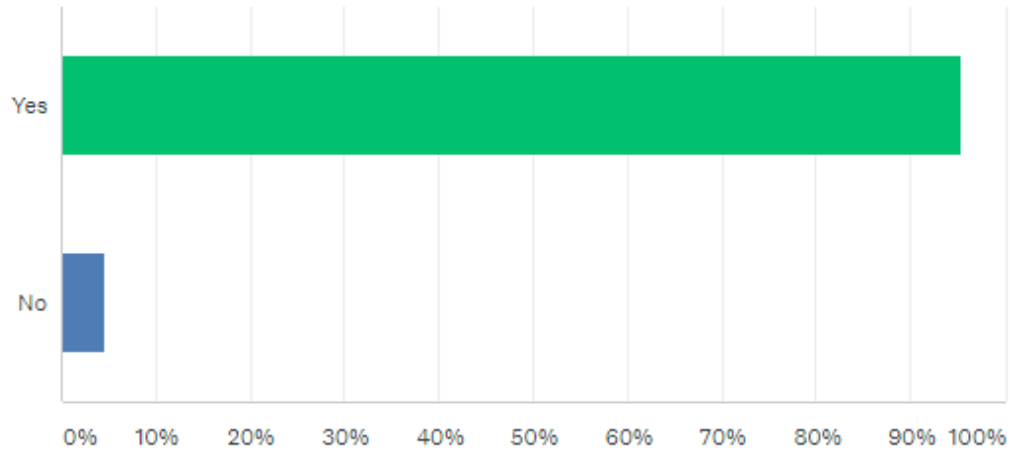


Figure 13: User Survey Results Q9

From the results of the survey, it was clear that majority of people listen to music via a music streaming service with 95%. Spotify and YouTube came back as the most popular streaming services. 82% of people use their phones as their primary device to listen to music while 77% also use their laptop or desktop machines. 95% of people said they would use an application that combines music streaming services.

One of the goals of the survey was to gather a list of important features for the *Tapes* application. It was found that song recommendations, layout, video playback, song queues and radio were some of the key features that appealed to users. Some features that users disliked about streaming services were poor layout, ads, limited music choice, premium services and lack of content.

The idea of creating *Tapes* as a web application was derived from the user survey. It was found from the survey that users listen to music in a variety of locations on multiple different devices. A web application would allow a user to access the application on a range of devices at home, school or work. With a desktop application the user would have to download the application meaning users at work or school may be unable to download the software due to network restrictions, but they could access a web application.

2.5 Literature Researched

Lee and Maiman-Waterman (2012) looked at understanding user requirements for music information services. They discovered an increase in the popularity of streaming services for music. With their surveys they found that YouTube and Spotify were in the top three most popular music services. They also found that users found that getting song recommendations and discovering new music is a key aspect of a good music streaming service. [6]

Liikkanen and Pirkka (2015) looked at current trends in the interacting with digital music and they also found when 700 Finish students were surveyed that YouTube and Spotify were the most popular services for on demand music and that many users utilized both actively. They found YouTube to be more popular because it's more accessible, more socially connected and satisfies their music needs with the bonus of visual content. [7]

Krastel, Bassellier and Ramaprasad (2015) undertook a study into how social features can play a key role in the development of online social connectedness with a music application. They show that implementing social features can create a more engaging environment for users, which can potentially increase the extent they can use the site and their willingness to pay for the site. [8]

Kachkach (2016) looked at user behaviour and sentiment in music streaming services. He wanted to explore the under-exploited aspects of user behaviour in music streaming services and show how they can be used to improve these services and be used to more accurately depict user preferences and sentiments. He found that monitoring specific aspects of the streaming service such as: change in user satisfaction, the evolution of user attention and engagement with the introduction of new features, finding patterns in streaming behaviour to improve user experiences, lead to a more enhanced user experience. [9]

2.6 Resultant Findings

It was clear from the research on alternative existing solutions that there is a gap in the market for an application that combines a range of music streaming services. Competitive applications either cater for a small audience, Sonos, or have flaws to their design, Last.fm. It was concluded that the main challenge of the *Tapes* application was creating an efficient yet effective user interface which would allow for an engaging user experience.

Based on the findings from the user survey and researched literature it was made clear that *Tapes* should focus on the implementation of both Spotify and YouTube as they are the most popular services. It should also incorporate enhanced user experience by providing additional features suggested in the user survey such as song suggestions, playlists, recommendations and a visual player.

YouTube APIs offer several important pieces of functionality to the *Tapes* application such as video playback, video playlists and the ability to search the site for a requested video and then play that video. [1] Video playback is a unique factor that only YouTube can provide for the application. Music videos add to the user experience of the music, often giving an insight into the meaning of the lyrics or provide an interpretation that may not occur to the listener without the visual cues.

The Spotify API and SDK allow *Tapes* users to play and listen to songs within the *Tapes* application, search, get recommendations and play playlists. However, the Spotify API and SDK come with some limitations, a user must have a Spotify premium account and in order to access the features on the *Tapes* application and a secondary device must be used in order to connect playback. [2] The limitations of Spotify do provide a challenge, however

due to the popularity of the premium service, with over 87 million premium users, Spotify premium is still one of the most popular streaming services and will be incorporated into the application, even with limited access. [30]

The decision was made not to include Apple Music into the application. A number of factors came into consideration when making this decision, the cost being a primary factor. The yearly subscription of €99, is a steep price to pay for a service that is less popular than other music streaming services that have the same functionality and provide a free API/SDK. By providing users with access to both YouTube and Spotify assures that all users can make use of the *Tapes* application. Users that have an Apple Music account over a Spotify account can make use of the free features that the YouTube implementation offers.

Soundcloud functionality could not be implemented into the *Tapes* application as the API is currently unavailable due to a high number of requests and has been under a re-evaluation period for a number of months.

2.7 Technologies Researched

The following technologies were researched for the development of the application. The benefits and drawbacks of some technologies were obvious from the background research; however, it was necessary to test and refine the use of certain technologies in order to best suit this design process.

Electron

Electron is an open source framework developed by GitHub. Electron allows for the development of desktop GUI applications using front and back end components originally developed for web applications. Electron is built on NodeJS for its backend and Chromium for its frontend. Several desktop applications are built with Electron including GitHub's Atom, Microsoft's Visual Studio Code, Skype, Discord and WhatsApp. [5]

Node.js

Node.js is an open source, cross platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript and can be used on OS X, Microsoft Windows and Linux.

Node.js also provides a rich library of JavaScript modules which simplifies the development of web applications using Node.js. [16]

AngularJS

AngularJS is a client-side JavaScript MVC framework used to develop web-based applications. AngularJS was originally started as a project in Google but now is an open source framework. Angular provides the ability to change static HTML into dynamic HTML by providing built in attributes and components. It also provides an ability to create custom attributes using JavaScript. [17]

React

React is a JavaScript library created by Facebook used for building user interfaces. React provides the ability to create reusable UI components. React allows for large web applications which data can change, without reloading the page. The main purpose of React is to be fast, scalable and simple. It can be used in conjunction with JavaScript framework libraries such as Node.js and Angular JS. [18]

MySQL

MySQL is an open source relational database management system (RDBMS) based on Structured Query Language (SQL) meaning that the data is split up between tables. MySQL is a cross platform database meaning it supports almost all platforms including Linux, Unix and Windows. [19]

MongoDB

MongoDB is a free open source document orientated database. MongoDB is a NoSQL database meaning it doesn't follow the relational database model. MongoDB is a cross platform database meaning it supports almost all platforms including Linux, Unix and Windows. MongoDB uses imbedded JSON like documents instead of using joins like in a relational database. MongoDB uses collections which are like tables in relational databases. A collection in MongoDB is made up of different fields and structures. [20]

Bootstrap

Bootstrap is an open source front-end framework for creating web application UIs. It contains HTML and CSS based design templates for text, forms, buttons, navigation and other interface components as well as optional JavaScript components. [21]

Materialize

Materialize is a CSS framework used to create responsive webpages created by Google. Google's goal when creating Materialize was to create a system of design that allows for a unified user experience across all their products on any platform. [22]

Spotify API

Spotify is a music streaming service that allows users to listen to millions of songs and other content like podcasts from artists all around the world. The Spotify web API and SDK allow a developer to use Spotify features and data in an application. The Web API allows users to search for artists, playlists and songs. While the SDK allows for music playback of all songs on their service. [2]

YouTube API

YouTube is a video sharing service where users can watch, like, share, comment and upload their own videos. YouTube provides a huge platform for artists to share their work. Many music artists choose to upload their music videos and songs to YouTube for free for their audiences, making YouTube one of the largest locations for music streaming. YouTube provide many APIs' for their service. The main ones of interest for this project being their Video Player API and their Analytics API. The Video Player API allows for the use of an embedded player to play videos directly in an application. The Analytics API allows for the retrieval of information such as viewing statistics, popularity metrics and more from YouTube videos and channels. [1]

SoundCloud API

SoundCloud is an online audio distribution platform and music sharing website. It enables users to upload, promote and share their sounds across the web. The SoundCloud API makes the features of SoundCloud available to app developers. [3]

MusicKit JS

MusicKit JS is an SDK created by Apple that allows users to play songs from Apple Music and their Cloud library as well as create playlists and add songs to their library. [23]

Musixmatch API

Musixmatch is a platform for users to search and share music lyrics. It is the largest platform of its kind having 73 million users and 14 million lyrics. Musixmatch displays lyrics synchronised with music being played. [24]

Last.fm API

The Last.fm API allows anyone to build their own programs using Last.fm data. This API allows developers to retrieve information on tracks, artists and chart information based on information gathered by their “Scrobbler” extension, and other music streaming services. [42]

Songkick API

Songkick provides concert discovery services and ticket sales for live music events. Its API gives developers access to its database which contains over 6 million upcoming and past events. [43]

Ticketmaster API

Ticketmaster is Ireland’s largest ticketing retailer for concerts, theatre, sports and attractions across the country. Ticketmaster’s discovery API allows developers to retrieve information for events, attractions, or venues. [44]

2.8 Technology Review

Rich Web Application vs Electron Desktop Application

Originally the application was to be developed as a desktop application using Electron, however when testing and creating a prototype some of the limitations of Electron became evident. Security and code protection being the main issues. With the use of various APIs in the application the user is required to sign in to a service in order to retrieve information, once signed in an authentication token is sent back to the application for that user to be identified by the API and use its features. A client id and client secret id are required in order

to identify the application to the service. This client secret id is used for many grant types and should not be available to the public. This makes it necessary to create a server that stores this information that acts as a mediator between the client and the service API. While using Electron, it was not possible to find a way to successfully interact with the server and retrieve an authentication key for the user, without storing the client secret key in the code, while this is acceptable for development, it would not be satisfactory for a fully functional application.

As well as being able to interact with a server for authentication purposes a rich web application has additional benefits that were not considered until after a user survey was performed. With a web application, users can use the application on any system such as Linux, Mac and Windows, the developer does not have to worry about incorporating additional features to cater for a specific system. A web application also allows users to access the application while at work or school where application downloads maybe blocked.

Node.js or AngularJS

In terms of delivering a software system in due time, it was important that the technologies chosen were suitable and were sufficiently familiar to the developer. Given these requirements Node would have been the perfect choice for creating a desktop application with Electron as it is a cross-platform runtime library and environment for running JavaScript applications outside the browser and Electron is built on top of it. With the change to a rich web application the use of Node is put to question. However, the benefits of using Node are that it has an extensive library of modules that help with certain features, it has extensive support documentation and it allows for JavaScript fetch queries that will be used extensively throughout the application in order to retrieve data from the various APIs used.

Choice of APIs

The Ticketmaster API was incorporated into the application in order to allow users to view upcoming music events in their area. The Ticketmaster API was chosen over alternative APIs such as Songkick and PredictHQ. Initially Songkick was implemented for the events feature. Songkick returns a list of upcoming music events in the user's area in order of the date. However, after some consideration and testing it was found that Ticketmaster retrieves a list of music events that are most relevant, filtering out unadmirable events. Songkick also does not provide cover art for events which is a design feature that matches the theme of the

application. PredictHQ was also considered however the API is a paid service with the same features that the Ticketmaster API offers.

The Musixmatch API and Genius API were considered for incorporating on screen song lyrics that users can look at while listening to their music. However due to licensing agreements the Genius API cannot provide free access to song lyrics. Musixmatch has combated this issue by providing a paid subscription service that allows developers to implement song lyrics into their applications, however price plans start at \$1000 per month. Due to the licensing agreements in place for song lyric APIs it was not possible to find an affordable or convenient solution.

The YouTube and Spotify APIs were chosen based on the findings previously mentioned.

Database

Initially MongoDB was chosen as the database technology for the application because it allows for the storage of unstructured data, which is similar to the data that is being retrieved from the APIs used in the application. The initial purpose of the database was to store user information such as username, password, playlists and metadata information for song recommendations. As development progressed the need for a database became less suitable.

The Spotify API requires a user to log into the service in order to access its features because of this, it would be excessive for a user to have to log in twice for the service. A login would again be unfavourable with the addition of the YouTube API as this feature does not require users to login and therefore can be accessed by all users.

The Spotify API has an advanced recommendation system that allows for specific tweaks in areas such as danceability, energy and loudness and can be adjusted by interacting with the API allowing for specific user recommendations without having to save data within a database.

CSS Framework

In order to choose the most suitable CSS framework for the application both Bootstrap and Materialize were tested. Both offered a lot of features that assisted with the design of the

application. The two frameworks are very similar in terms of what they have to offer. However, after some testing and deliberation the decision was made to use Materialize. One feature that Materialize provides that Bootstrap lacked was collections, namely avatar collections. An avatar collection allows you to display a list of objects on the screen and with them a circular image that relates to the object in the list. This feature was used prominently throughout the application for features such as Events and YouTube's Video Queue and playlists.

2.9 Conclusions

In conclusion, it was clear from the background research that there's a need for a multi service streaming application that combines a host of different music streaming services into the one application. Alternative existing solutions either don't provide the service to the majority of users or don't provide an enticing user experience. It was clear the *Tapes* application should focus on both YouTube and Spotify based off an initial user survey and researched literature. Finally, from the technologies researched it was made clear why certain decisions were made in terms of what technologies were being used to develop the *Tapes* application.

3. System Design

3.1 Introduction

The system design was an important aspect of the project. The system design outlines the chosen project management approach, the technical architecture of the system, an overview of the functionality of the system and the user interface and user experience of the system.

3.2 Methodologies

There are many different project management approaches that may be taken in the development process. The following approaches were analysed in order to decide which was the most suitable one for this project.

Waterfall Method

Waterfall model is a traditional project management approach. It is called waterfall because it is linear and sequential. It has several discrete phases. Each phase has strictly defined goals and take place one after another. No phase begins before the previous phase is complete. The only way to revisit a phase is to start again at the beginning of the model. The goal of the waterfall model is to gather and make clear the requirements at the beginning of development in order to avoid issues further on, without the possibility of making changes.

[11]

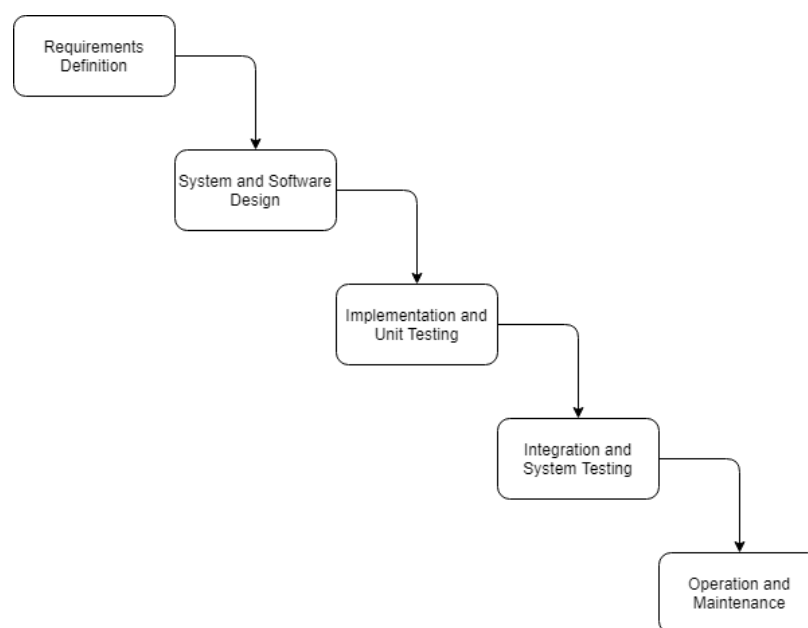


Figure 14: Waterfall Methodology

1. Requirements Definition

Detailed requirements are gathered, analysed and plans are made for the whole development process.

2. System and Software Design

Using the established requirements, a system design is created. No coding takes place at this phase, but specifications are established such as programming language, hardware requirements etc.

3. Implementation and Unit Testing

Coding takes place at this phase. Information is used from the previous stages and a functional product is created. Typically, small pieces of code are created and then integrated at the end of this phase.

4. Integration and System Testing

Once all the coding is complete, the product is then tested. Problems are found and reported. If any serious issues arise the project may need to return to phase one for revaluation. If the product passes testing the product is then deployed to be released.

5. Operation and Maintenance

Once the product has been delivered to the users, if issues arise patches and updates may need to be made in order to address them. If any big issues arise at this phase, again the project may need to return to phase one.

Test Driven Development

Test Driven Development is an approach to program development in which testing, and code development is interleaved. Code is developed incrementally, along with a set of tests for that increment. The next increment doesn't commence until that code has passed all its tests. Test driven development was introduced as part of the extreme programming agile development method. It has developed a mainstream acceptance and is used in both agile and plan based processes. [12]

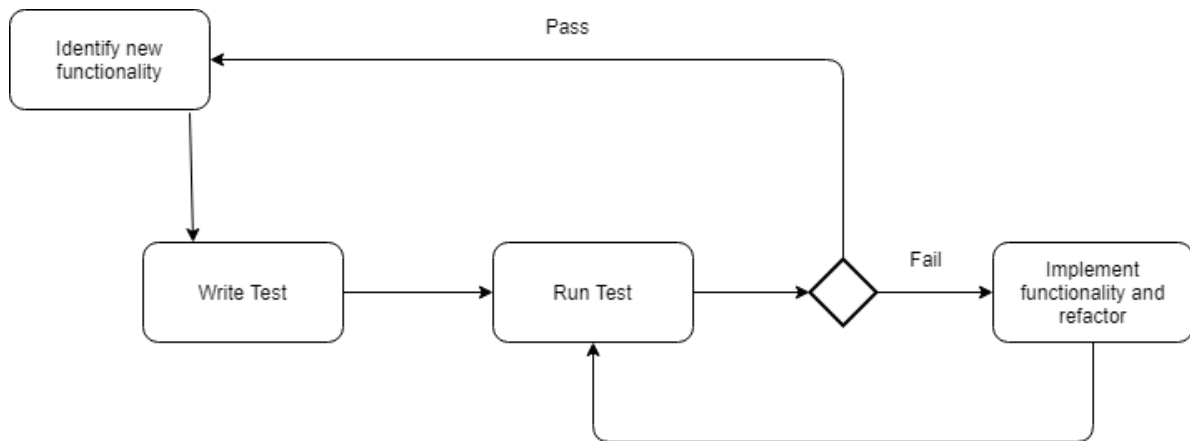


Figure 15: Test Driven Development

The fundamental TDD process is shown above. The steps of the process are as follows.

1. Identify the increment of functionality that is required. This should be small and be able to be implemented in a few lines of code.
2. Test scripts are written that will test for the intended functionality of the code.
3. Run the code and see does is work in accordance to the test script.
4. If the test fails, then refactor the code and implement the functionality again and re run the test script.
5. Once all tests are run successfully, a new implementation of a new increment of functionality can be carried out.

Scrum Framework

Scrum is a framework within which people can address complex adaptive problems while effectively delivering products. The Scrum model suggests that a project progresses in a series of sprints. Sprints are timed to usually two weeks but no longer than a month.

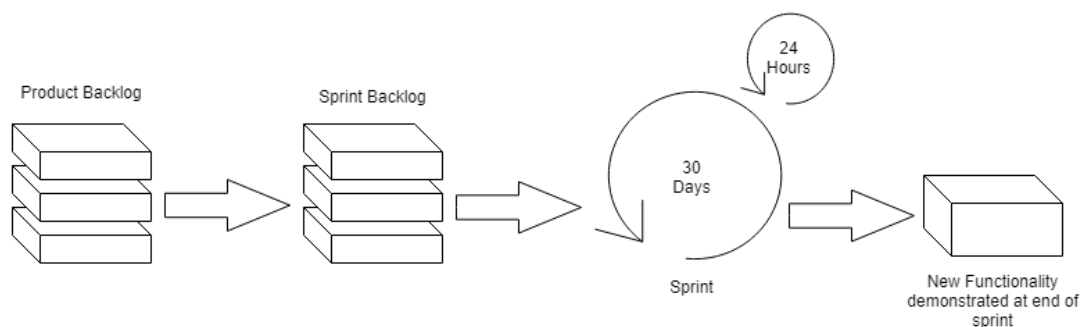


Figure 16: Scrum Framework

Sprints start with a planning meeting, where team members form a prioritized list of features which contains short descriptions of all functionality desired in the product, this is called the product backlog.

Team members are then delegated items to develop and create a sprint backlog, a list of tasks to perform during the sprint. A typical Scrum backlog comprises of features, bugs, technical work and knowledge acquisition.

Each day of the sprint, there should be a team meeting where team members share what they worked on the prior day, will work on that day and identify any issues with further progress. At the end of a sprint, the team conducts a sprint review during which the team demonstrates the new functionality to the stakeholders of the project, who then provide feedback. This feedback loop within Scrum may result in changes to the project which can result in revising or adding items to the product backlog. [12]

3.3 Review Methodologies

All the above methodologies have their own advantages and disadvantages. The linear nature of the Waterfall Model makes a project easy to manage, the sequential system makes it known where the project is at any given time and if it's where it should be. Waterfall model makes it simple to show the progress of the project because of the discrete phases. However, the drawback of the Waterfall Model is the difficulty of accommodating change after the process is underway. [11] This factor makes the Waterfall Model unsuitable for this project because of the unpredictable behaviour of the multiple APIs intended to be used in this project. APIs can be updated without warning, which could force changes in design. If a change had to be made in the design the project would have to go back to phase one and the process would have to be repeated which could cause delays in the progress of the project.

The scrum framework enables projects where the requirements are hard to quantify to be successfully developed due to daily meetings and updates. Scrum is iterative which allows for changes to be made throughout the progression of the project, short sprints also makes it easier to cope with these changes. [12] Scrum is very beneficial in a team environment or in a business environment, however for this project it wouldn't be beneficial as the project is

an individual project rather than a team project. Aspects of the scrum framework will be taken and implemented in the approach for weekly meetings with stakeholders (Project Supervisor and Academic Mentors).

Test Driven Development allows for code to be more modular because of the creation of small tests, this also helps to understand and internalise key principles of good design. Test Driven Development offers well defined progress tracking and reporting capabilities, it helps a developer to understand their code, it encourages small steps which improves the design by cutting unnecessary dependencies and it helps clarify requirements because it makes it necessary to figure out concretely what inputs must be fed into the system and what outputs are expected. [12]

Test Driven Development is the approach that will be taken for this project as it puts emphasis on the key aspects of the Final Year Project, documentation and understanding. It allows for continuous documentation of the development process which in turn enhances the understanding of the complexity of the project.

3.4 Technical Architecture

Technical Architecture design is the first stage in the software design process. It is the critical link between design and requirements. It identifies the main structural components in a system and the relationships between them. The goal of creating the technical architecture is to establish a model that describes how the system is organized as a set of communicating components. [11]

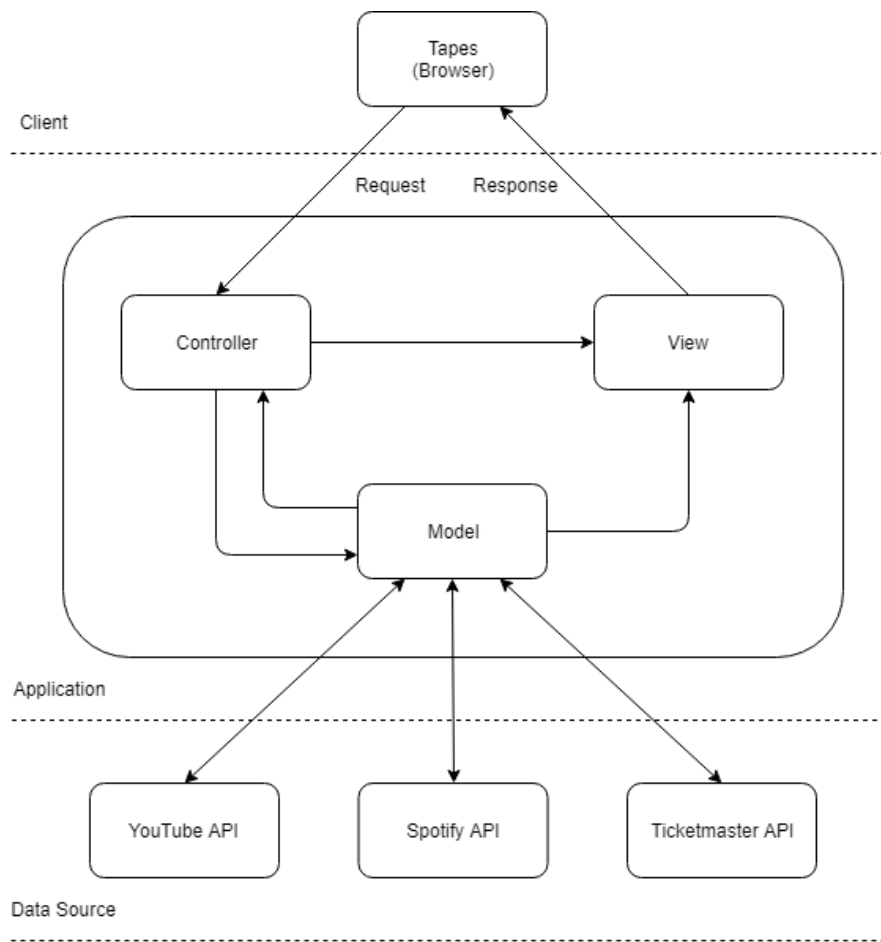


Figure 17: Technical Architecture

Model View Controller

Model View Controller architecture separates presentation and interaction from system data. The system is separated into three distinct components that interact with each other. The Model component manages the system data and associated operations on that data. The View component represents the visual layer of how the data is presented to the user. The Controller component manages user interaction (e.g. key presses, mouse clicks etc.) and passes these interactions to the View and the Model. The advantage of using the Model View Controller pattern is that it separates the elements of the system allowing them to change independently. [12]

3.5 Source Code Layout

Tapes Server

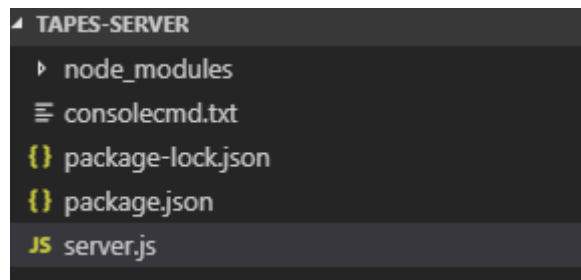


Figure 18: Tapes Server Source Code Layout

The server application has one main JavaScript file called `server.js`. The server is used for Spotify authentication. Once a user has requested to log in, they will be redirected to the server. Once authenticated the server retrieves an access token for that user and it is passed in the address bar back to the client, the user is also redirected back to the client.

`node_modules`, `package-lock.json` and `package.json` are all part of Node.js and are used to configure the dependencies for the project.

Tapes Client

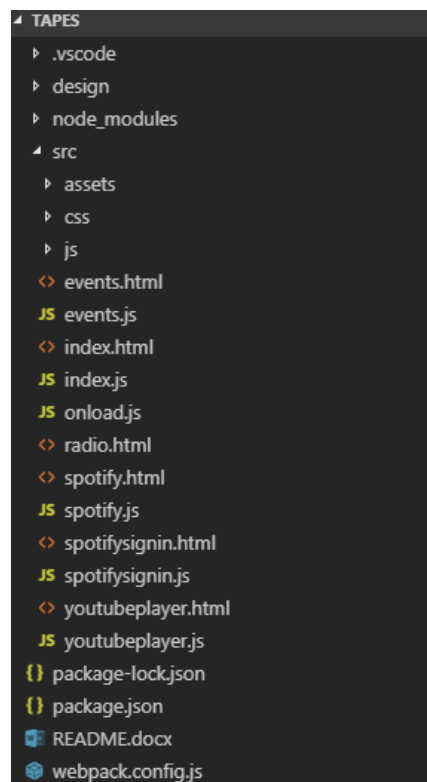


Figure 19: Tapes Client Source Code Layout

The *Tapes* client code contains the same Node.js dependency files as the server application. It also has two additional folders: `design` and `src`. The `design` folder stores information regarding the design of the application including screenshots included in this report. The `src` folder contains all the source code for the client application. The contents of the `src` folder is as follows:

Assets

This folder contains image assets that are used throughout the application.

CSS

Contains Materialize CSS files that are used to style the user interface. These CSS files were modified in order to fit with the theme of the *Tapes* application.

JS

The JS folder holds Materialize JavaScript files that are used throughout the application for interactive elements.

Events.html

The `events.html` file contains the layout of the Events page of the application. This page gives the users a list of upcoming events in their area. The relevant CSS and JavaScript files are also embedded within the HTML file.

Events.js

`Event.js` provides the functionality for the `events.html` page. It connects to the Ticketmaster API and retrieves information on upcoming events in the user's area.

Index.html

The `index.html` file contains the layout of the Home page of the application. This page displays the top songs according to Last.fm, YouTube Ireland and YouTube US. The relevant CSS and JavaScript files are also embedded within the HTML file.

Index.js

`Index.js` provides the functionality for the `index.html` page. It connects to the Last.fm and YouTube APIs and performs a variety of transformations and outputs the results to the user.

Onload.js

The onload.js file is embedded in most of the HTML files. This file ensures that on load of the DOM, certain features of Materialize are initialised and functional.

Radio.html

The Radio.html file contains the layout of the Radio page of the application. This page displays a list of popular Irish radio stations and allows users to listen to said radio station within their browser.

Spotify.html

Spotify.html contains the layout of the Spotify Player page of the application. The user is redirected here once they have logged in and authorized the *Tapes* application. The page then displays users with instructions on how to connect their device to the *Tapes* application and play their Spotify tracks on the *Tapes* player within the browser. Once their device is connected the page is updated with JavaScript to display the *Tapes* player along with Spotify search, recommendations and playlists. This page embeds the Spotify SDK and API in order to allow Spotify functionality.

Spotify.js

Spotify.js provides the functionality of the *Tapes* Spotify player, search, recommendations and playlists. It manipulates the DOM in accordance with the users input. Once the user is logged in to Spotify the page is updated through JavaScript to an instructions page and then when the user enables playback the page transforms to the *Tapes* player allowing users to skip, pause, play and adjust audio levels.

Spotifysignin.html

This page provides users with the ability to log into Spotify and authorize the *Tapes* application.

Spotifysignin.js

Spotifysignin.js redirects users to the authentication server once a log in button is clicked on the Spotifysignin.html page.

Youtubeplayer.html

The youtubeplayer.html contains the layout of the YouTube page of the application. This page allows users to search and queue for music videos on YouTube. Users can make use of songs on suggested playlists by adding them to the song queue. Users can then remove the songs from the queue. The relevant CSS and JavaScript files are also embedded within the HTML file.

Youtubeplayer.js

YouTube.js facilitates the functionality of the YouTube page. It connects to the YouTube player API allowing videos to be played within a player that can be manipulated unlike an iFrame. The player does not facilitate queuing, so this functionality had to be implemented here. Search functionality for both videos and playlists is also implemented here.

3.6 Functionality Overview

3.6.1 Tapes Functionality

The following use case shows the basic user interaction with the application. The user has the ability to navigate through the different webpages provided. The user can view the current most popular songs and artists on the home page, access the YouTube player provided on the YouTube page, sign in to Spotify and make use of the interactive Spotify player and view song recommendations on the Spotify page, view and listen to the most popular radio stations on the radio page and browse events that are upcoming in the user's area.

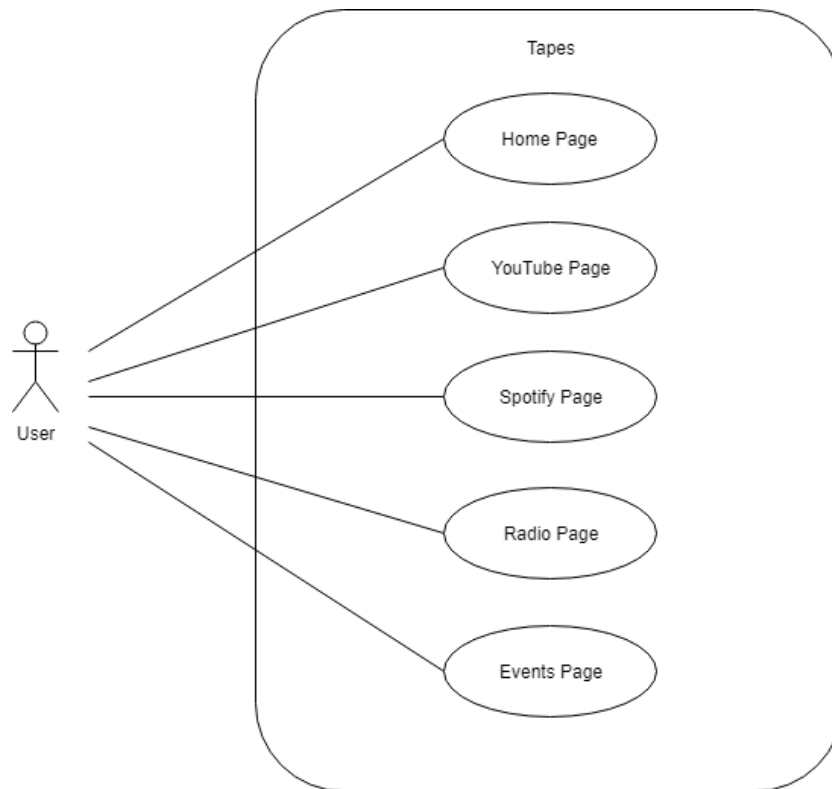


Figure 20: Use Case 1 – Tapes Functionality

3.6.2 YouTube Functionality

The following use case shows the functionality of the YouTube page. The user is not required to log into YouTube in order to provide a more fluid user experience.

Search Songs

The user has the ability to search YouTube for any video on the platform. The user can then add the video to the video queue. If the user clicks the thumbnail of the video the video will be added to the top of the queue and play immediately. If the add to queue icon is clicked the video will be added to the end of the queue.

Song Queue

The user can remove a video from the current video queue by clicking on the thumbnail or the remove queue icon.

Search Playlist

Due to the limitations of the YouTube API, the user can search for an existing playlist by entering the id of the playlist. The playlist can then be added to the video queue.

Explore Suggested Playlists

The *Tapes* application provides the user with suggested playlists in the genres of rap and pop as well as the top music videos on the YouTube platform. The user can add videos from the playlist to the song queue.

Control Player

The user has the ability to control playback via the YouTube player that is embedded into the *Tapes* application. Users can skip or play the previous video once videos exist in the video queue. Users can also control the audio levels and seek through the video within the player.

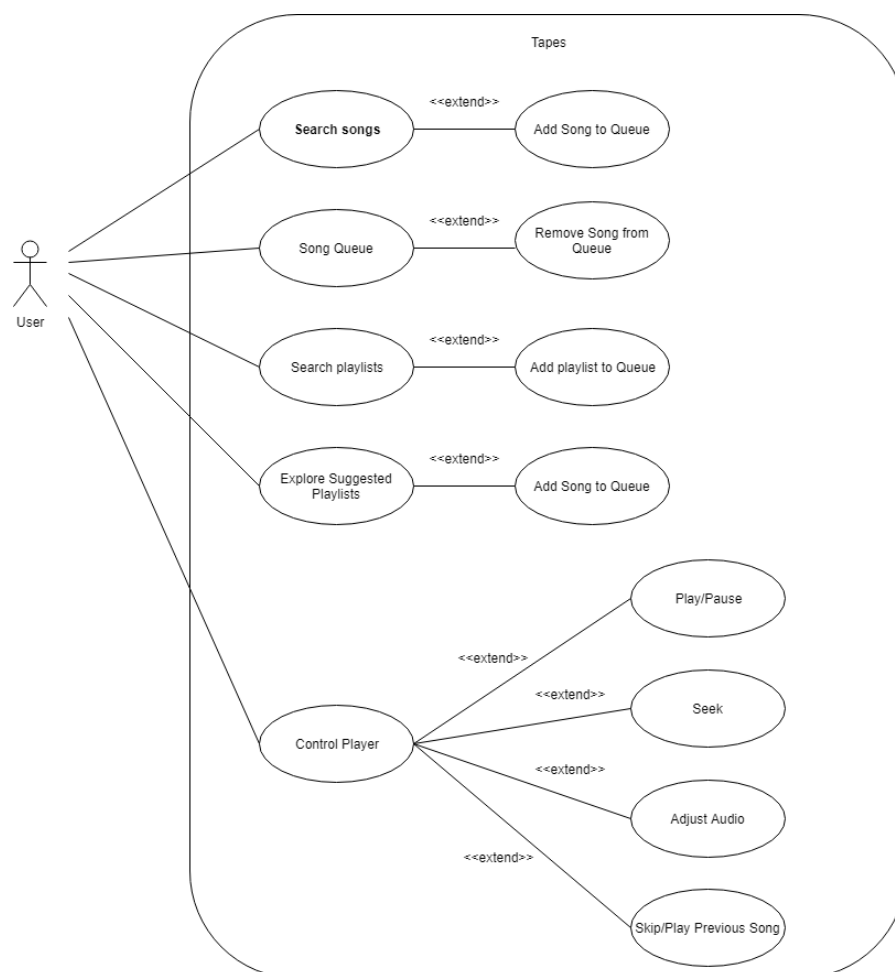


Figure 21: Use Case 2: YouTube Functionality

3.6.3 Spotify Functionality

The following use case shows a situation where the user signs into Spotify. In order to access the features that the Spotify API provides, the user must have a premium Spotify account, log into the service and add *Tapes* as an approved application. Once the user has signed in, they can follow the on-screen instructions to connect their device to *Tapes*. When a user has signed in and connected their device, they can stream tracks from their device to the *Tapes* player. The users can skip or play previous songs and adjust the audio levels.

User can search for specific tracks or artists using a search menu, *Tapes* will then provide a list of results which the user can select their chosen track and it will begin to play in the *Tapes* player.

Users are provided with song recommendations and playlists from which they can also select and play within the *Tapes* player. Recommendations are based off the current track that is playing. The featured playlists are all playlists the user has most recently been listening too.

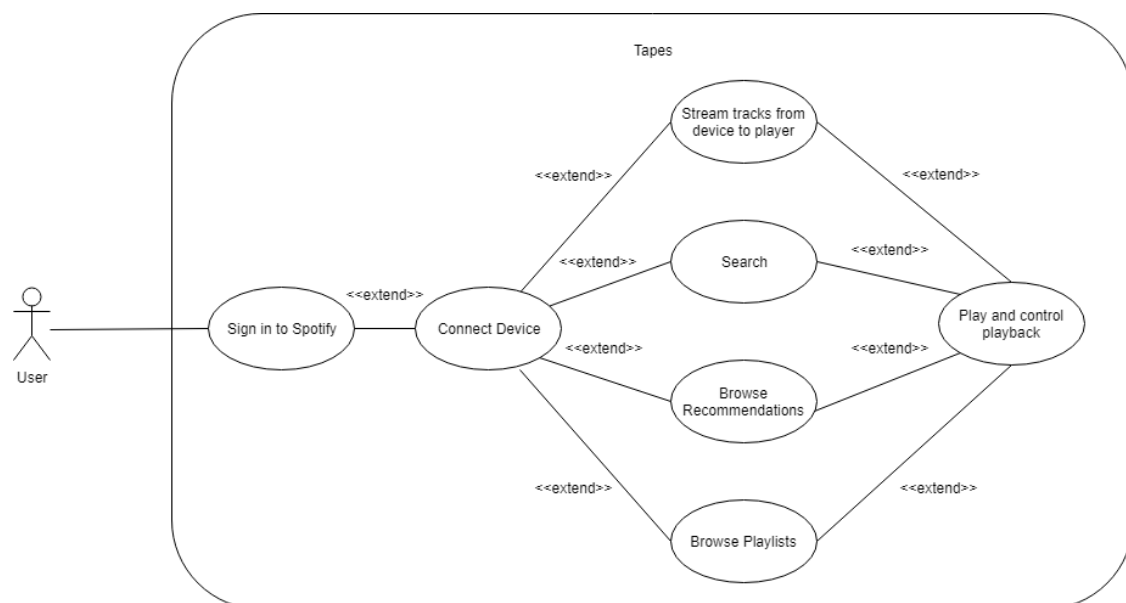


Figure 22: Use Case 3 – Spotify Functionality

3.7 User Interface and User Experience

User interface (UI) design refers to the visual elements of an application which includes the look and feel of the application and the presentation and the interactivity of the application.

It's the interface the user interacts with, which makes the overall experience of an application aesthetically pleasing.

User experience (UX) design is the process of enhancing the user's satisfaction and loyalty by improving the usability, ease of use and pleasure provided in the interaction between the user and the application. User experience can be gaged by user engagement such as user surveys.

3.7.1 Design Iterations

Being that the *Tapes* application is user focused a large amount of time went into designing the user interface and creating a satisfactory user experience. User experience can be gained through the usability of the user interface. It is important to have a coherent navigable layout that makes it easy for users to find what they are looking for. The user interface went through a number of design iterations. An initial low fidelity prototype was created, the project prototype was based off its design. An additional horizontal prototype was created which formed the base of the final design. The final solution follows the Z pattern, which mimics the route the human eye travels when reading – left to right, zigzagging top to bottom. Using the Z pattern focuses the user's attention on elements of the page persuading them to move closer to the goal of the page.

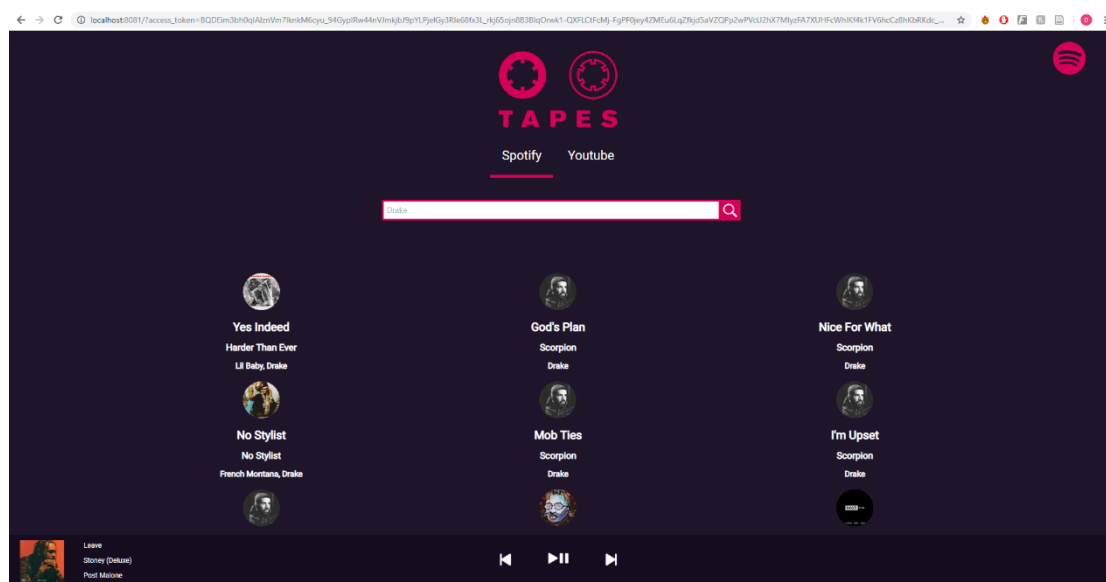


Figure 23: Tapes Design First Iteration

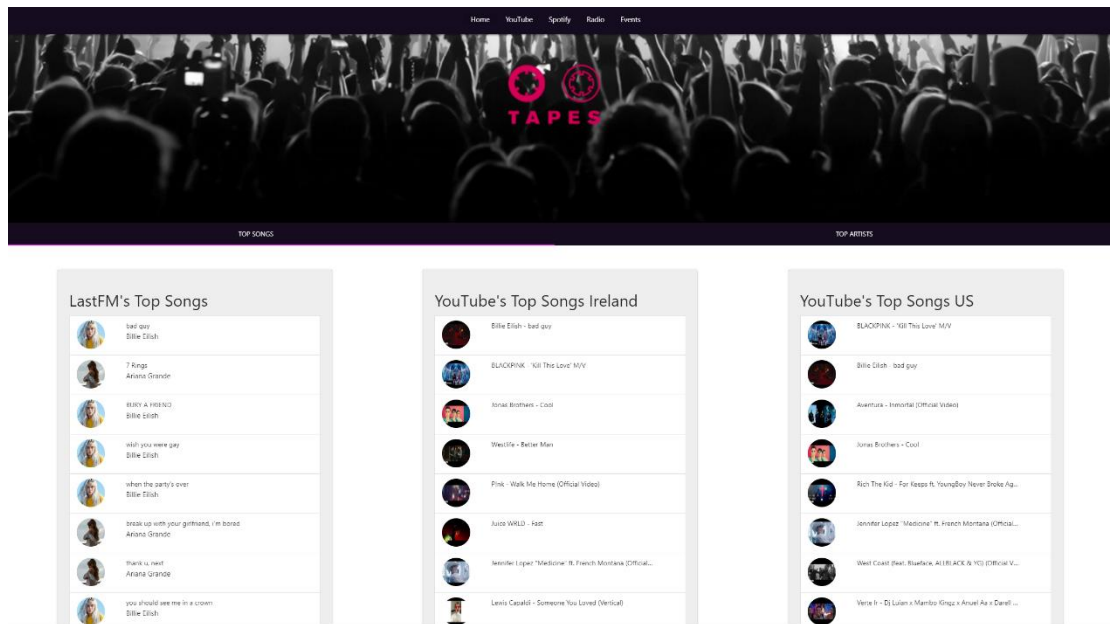


Figure 24: Tapes Design Second Iteration

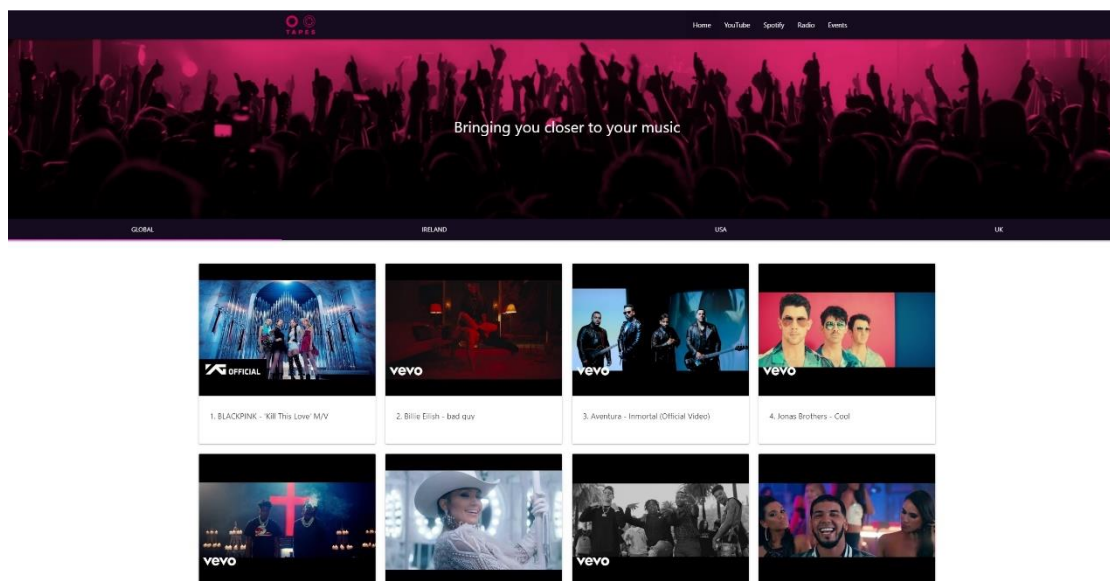


Figure 25: Tapes Design Final Iteration

Aspects of the user experience were improved after user testing was concluded. Feedback elements such as toasts were introduced to improve the experience of certain pages. After user evaluation was concluded layout changes were made to the home page provide a more satisfactory user experience. The use of Materialize CSS framework allowed for a consistent design throughout the application.

3.7.2 Low Fidelity Prototype

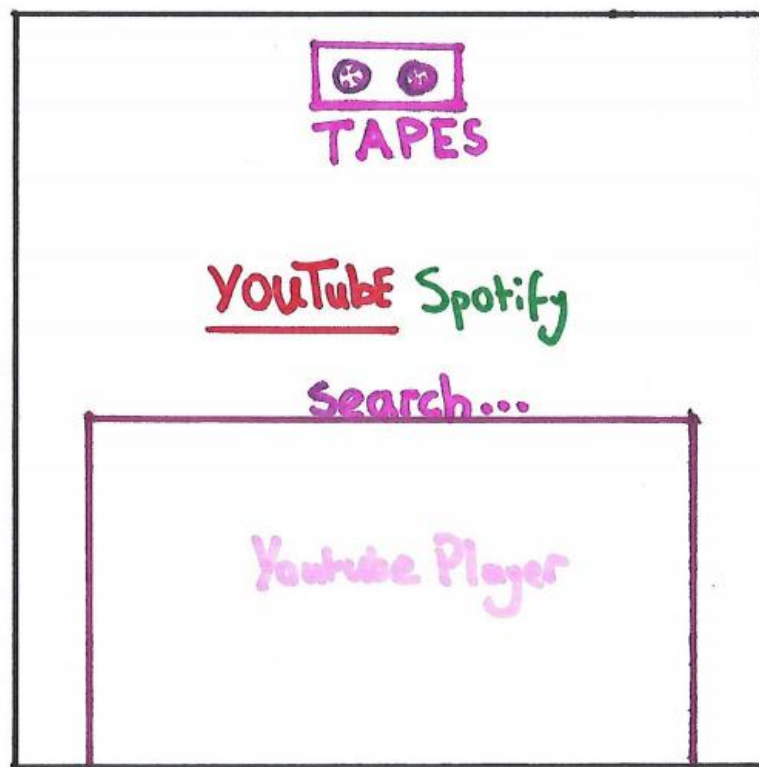


Figure 26: Low Fidelity Prototype Screen 1

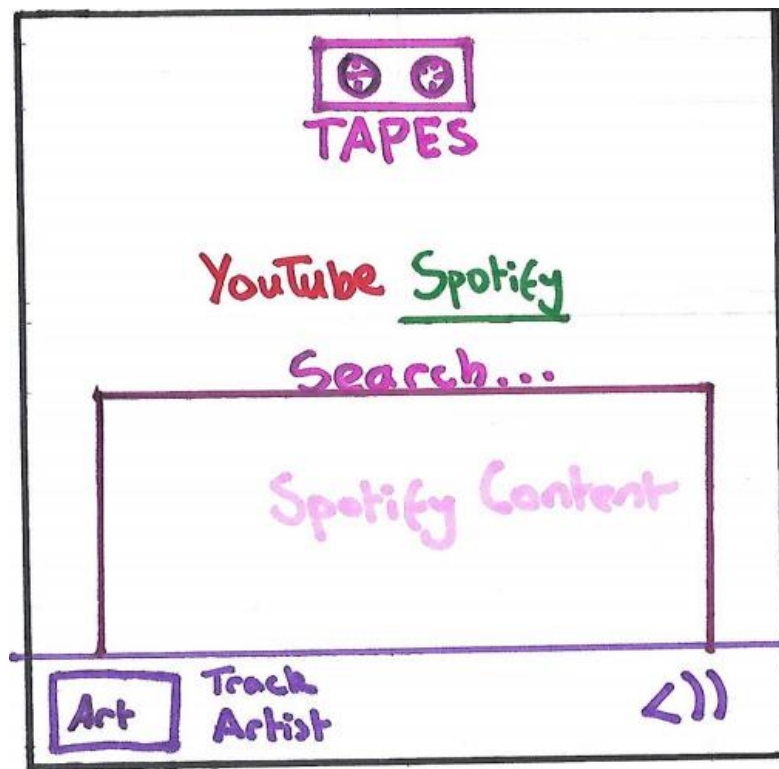


Figure 27: Low Fidelity Prototype Screen 2

3.7.3 Horizontal Prototype

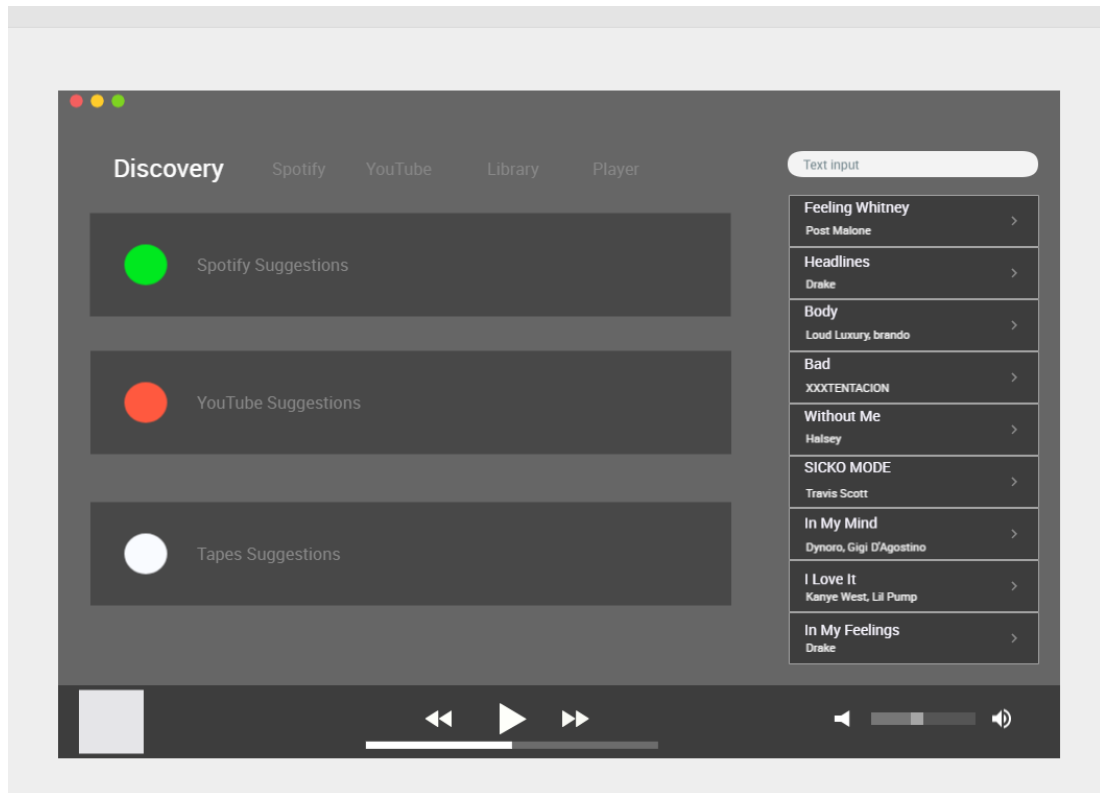


Figure 28: Horizontal Prototype - Discovery

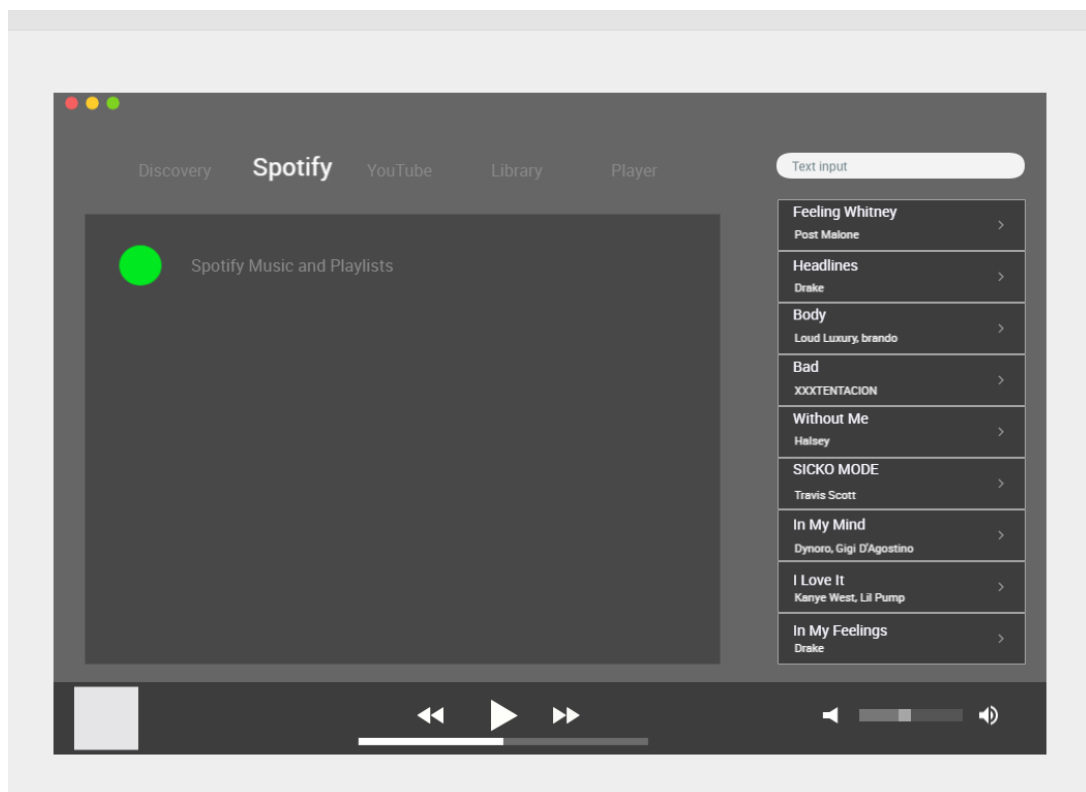


Figure 29: Horizontal Prototype - Spotify

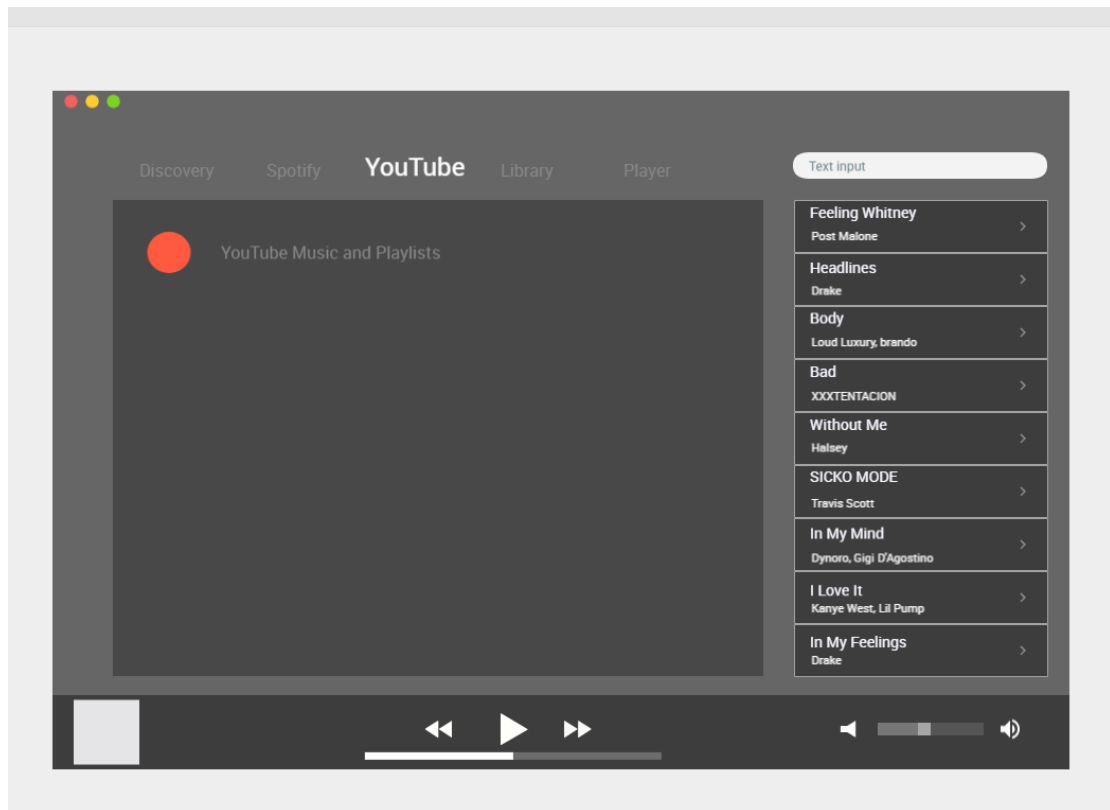


Figure 30: Horizontal Prototype - YouTube

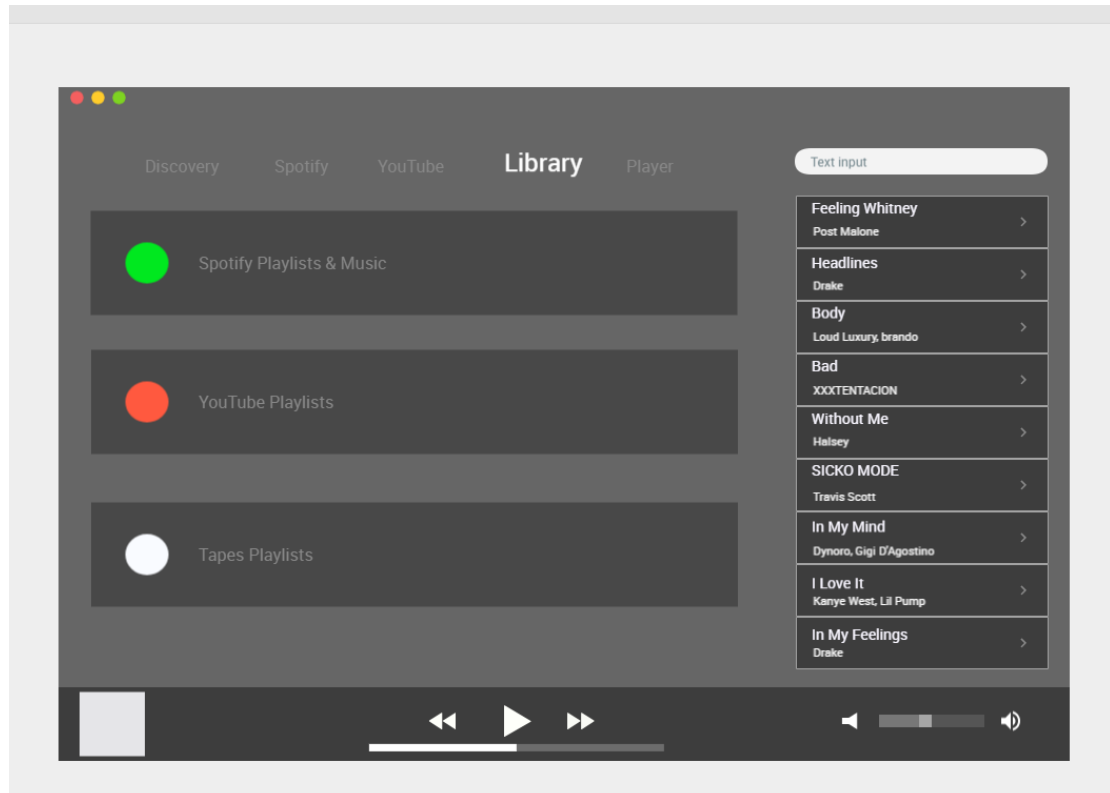


Figure 31: Horizontal Prototype - Library

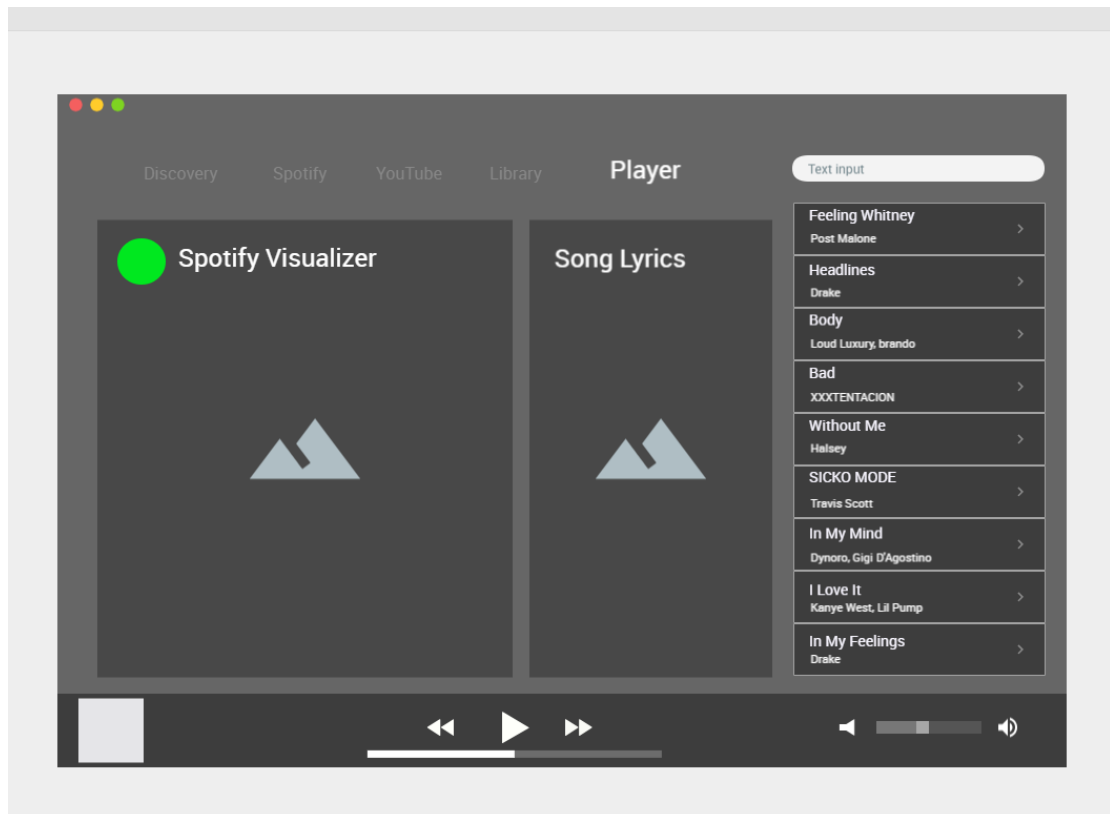


Figure 32: Horizontal Prototype – Spotify Player

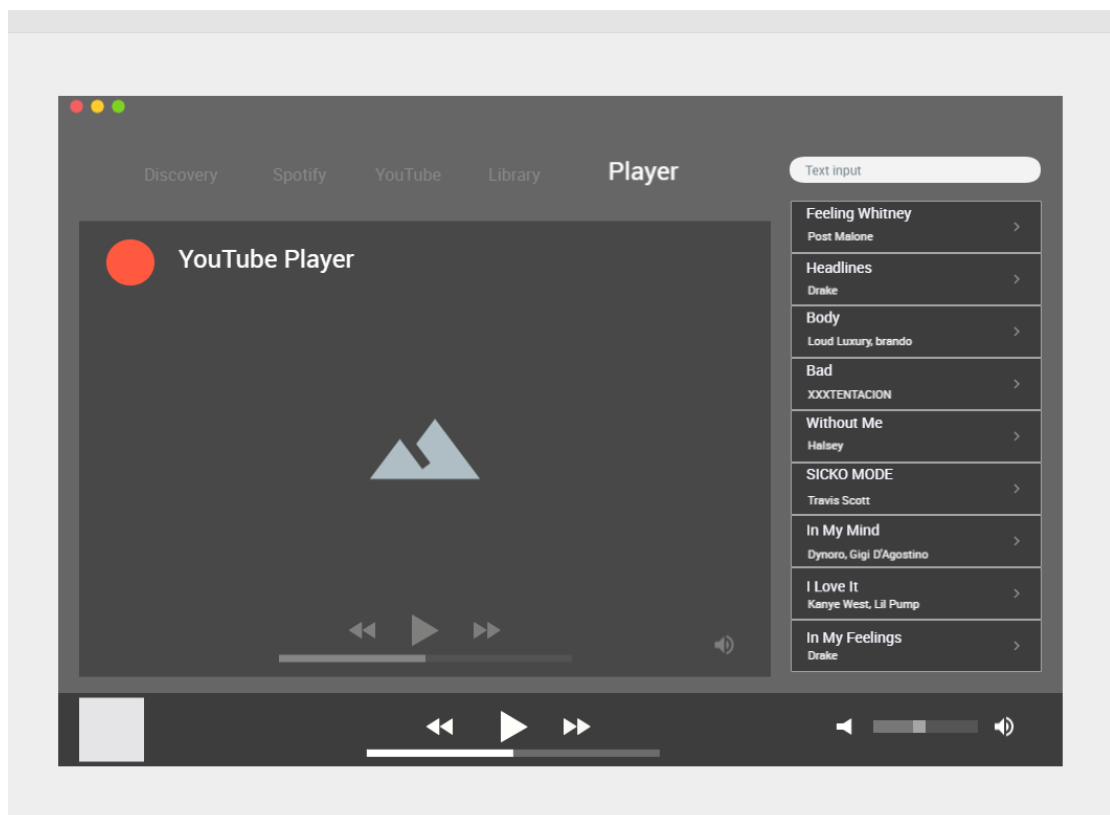


Figure 33: Horizontal Prototype – YouTube Player

3.8 Conclusions

In conclusion, Test Driven Development was chosen as the project management approach for the project as it puts emphasis on documentation and understanding. The technical architecture of the system was defined with emphasis on the Model View Controller principle. The source code of the document was presented, and its contents was outlined. The functionality of the application was prescribed with the aid of use case diagrams. Finally, the progression of the user interface and user experience was detailed.

4. System Development

4.1 Introduction

The development of the *Tapes* application took place over a number of months. An initial prototype was created where fundamental features were tested and implemented. The prototype provided a base for future developments. The chosen methodology of Test Driven Development allowed for testing to be carried out throughout the development cycle. Test cases were created for each component, if a test failed the component was refactored and tested again until it passed. This iteration ensured the development of high quality code. The following section will look at the development of the *Tapes* application and the imposed challenges.

4.2 Front-End Development

4.2.1 Spotify

Spotify consists of two resources for developers an API and an SDK. The API provides JSON metadata about music artists, albums and tracks as well as user information such as playlists. The SDK allows developers to retrieve the current users track list and enables audio playback from within the browser.

Both resources require the user to authorize the *Tapes* application. It is also necessary for the user to have Spotify premium in order to access features used by the API or SDK.

A backend authentication server was created to facilitate user login. Once the user is logged in the access token is generated for that user and is passed from the server to the client via the browser's address bar. The token is then parsed in the client and used in future requests to the API or SDK.

The user is initially greeted with a login screen when they navigate to the Spotify page of the application. The user then clicks the log in button which redirects them to the authorization server.

Once the user is logged in, the page will dynamically load the instruction screen. The instruction screen provides on screen assistance to the user. In order to facilitate Spotify playback, the user must change their current device to *Tapes*.

Tapes Player

Once the user has connected their device to *Tapes*, playback can be initialised. Users can then control audio playback through the *Tapes* player.

```
//Call Spotify Player and initiate player
//Allows user playback in web browser from any device
window.onSpotifyWebPlaybackSDKReady = () => {
  const player = new Spotify.Player({
    name: 'Tapes',
    getOAuthToken: cb => { cb(access_token); }
  });

  //Error handling
  player.addListener('initialization_error', ({ message }) => { console.error(message); });
  player.addListener('authentication_error', ({ message }) => { console.error(message); });
  player.addListener('account_error', ({ message }) => { console.error(message); });
  player.addListener('playback_error', ({ message }) => { console.error(message); });

  //Playback status updates
  player.addListener('player_state_changed', state => {

    console.log(state);

    //Change DOM and display current track information
    let trackName = state.track_window.current_track.name;
    let albumName = state.track_window.current_track.album.name;
    let artistName = state.track_window.current_track.artists
      .map(artist => artist.name)
      .join(", ");
    let albumArt = state.track_window.current_track.album.images[2].url;
    let trackid = state.track_window.current_track.id;

    document.getElementById("playback").innerHTML = playback;

    document.getElementById("albumArt").src = albumArt;
    document.getElementById("trackName").innerHTML = trackName;
    document.getElementById("albumName").innerHTML = albumName;
    document.getElementById("artistName").innerHTML = artistName;

    playerEvents();
  });
}
```

Figure 34: Initialisation of Spotify Player Source Code

The above code shows the initialisation of the Spotify connect player. The name of the application and the access token that was generated when the user logged in are passed in as parameters in order to authorize the application. Once the player's state has changed to ready the DOM is dynamically updated to the *Tapes* player.

When the player is initialised, it retrieves JSON information regarding the current, next and previous tracks.

spotify.js:179

```

▼ Object
  bitrate: 256000
  context: {uri: "spotify:user:1157975653:playlist:2073900fgEbJxsM90kGkFT", metadata: {}}
  disallows: {pausing: true}
  duration: 184904
  paused: true
  position: 2143
  repeat_mode: 0
  restrictions: {disallow_pausing_reasons: Array(1)}
  shuffle: true
  timestamp: 1554128619230
  track_window:
    ▼ current_track:
      ▶ album: {uri: "spotify:album:5SUsCi66DN0gxvH2ujplhj", name: "Grace", images: Array(3)}
      ▶ artists: [{-}]
      duration_ms: 184904
      id: "4eQBL0cuF3ismYCzBff2Ii"
      is_playable: true
      linked_from: {uri: null, id: null}
      linked_from_uri: null
      media_type: "audio"
      name: "Grace"
      type: "track"
      uri: "spotify:track:4eQBL0cuF3ismYCzBff2Ii"
      __proto__: Object
    ▼ next_tracks: Array(2)
      ▼ 0:
        ▶ album: {uri: "spotify:album:3pP8bp1N19n1AM9xFpdKtZ", name: "For Once In My Life", images: Array(3)}
        ▶ artists: [{-}]
        duration_ms: 169800
        id: "4kP69y3GKH19tXckfgp4bK"
        is_playable: true
        linked_from: {uri: "spotify:track:2yMmwardt8VzlpNBWrGYD6", id: "2yMmwardt8VzlpNBWrGYD6"}
        linked_from_uri: "spotify:track:2yMmwardt8VzlpNBWrGYD6"
        media_type: "audio"
        name: "For Once In My Life"
        type: "track"
        uri: "spotify:track:4kP69y3GKH19tXckfgp4bK"
        __proto__: Object
      ▶ 1: {id: "6pTg4H8juKV9iSzrcVUbM3", uri: "spotify:track:6pTg4H8juKV9iSzrcVUbM3", type: "track", linked_from_uri:
        length: 2
        __proto__: Array(0)}
    ▼ previous_tracks: Array(2)
      ▼ 0:
        ▶ album: {uri: "spotify:album:40GMAhrivY3R01rsY4YdrZb", name: "Views", images: Array(3)}
        ▶ artists: (2) [{-}, {-}]
        duration_ms: 263373
        id: "38tuIIRq1kuJKPwHF2B85z"
        is_playable: true
        linked_from: {uri: "spotify:track:11KJ5RSgaDxqydKYiD2Jew", id: "11KJ5RSgaDxqydKYiD2Jew"}
        linked_from_uri: "spotify:track:11KJ5RSgaDxqydKYiD2Jew"
        media_type: "audio"
        name: "Too Good"
        type: "track"
        uri: "spotify:track:38tuIIRq1kuJKPwHF2B85z"
        __proto__: Object
      ▶ 1: {id: "1r3M82sqImOu3AUwRdELRT", uri: "spotify:track:1r3M82sqImOu3AUwRdELRT", type: "track", linked_from_uri:
        length: 2
        __proto__: Array(0)}
        __proto__: Object
        __proto__: Object
  
```

Figure 35: Spotify Playback JSON Data

This information is then parsed in order to be displayed to the user. The track window provides information relating to the current track, next track and previous track. The track name, album name, artists and album art are all retrieved from here and displayed to the user within the *Tapes* player.

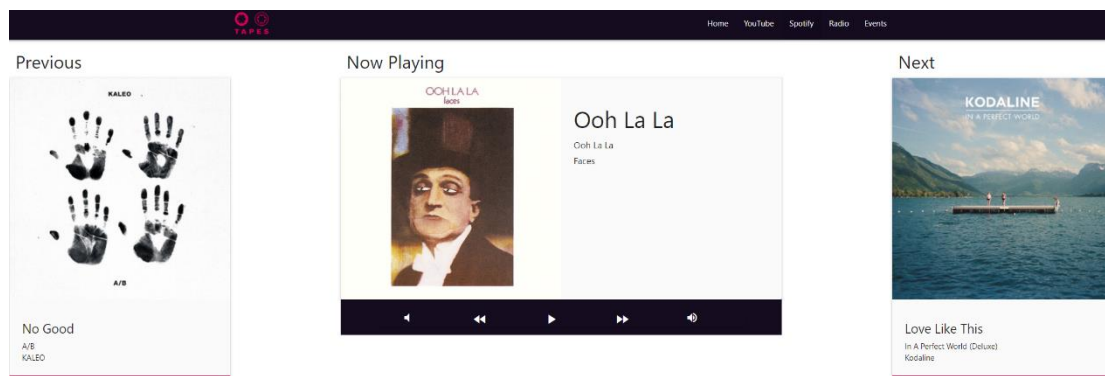


Figure 36: Tapes Player

Users can control the player with on screen buttons that are clear and easy to see. These are controlled by onclick events that trigger independent functionality depending on the button pressed. Users can adjust the volume, play or pause the current track, and skip or play previous tracks.

In order to incorporate the next and previous tracks into the player the DOM must be dynamically updated. In order to do this, JavaScript node elements were created and updated based on ids.

```
//Change DOM and display next track information
//If next track exists add it
if(state.track_window.next_tracks[0] == null)
{
    document.getElementById("nexttrack").innerHTML = null;
}
else
{
    let nexttrackName = state.track_window.next_tracks[0].name;
    let nextalbumName = state.track_window.next_tracks[0].album.name;
    let nextartistName = state.track_window.next_tracks[0].artists
        .map(artist => artist.name)
        .join(", ");
    let nextalbumArt = state.track_window.next_tracks[0].album.images[0].url;

    insertNext();

    document.getElementById("nextalbumArt").src = nextalbumArt;
    document.getElementById("nexttrackName").innerHTML = nexttrackName;
    document.getElementById("nextalbumName").innerHTML = nextalbumName;
    document.getElementById("nextartistName").innerHTML = nextartistName;
}

//Change DOM and display previous track information
//If previous track exists add it
if(state.track_window.previous_tracks[0] == null)
{
    document.getElementById("previous").innerHTML = null;
}
else if(state.track_window.previous_tracks[1] != null)
{
    let prevtrackName = state.track_window.previous_tracks[1].name;
    let prealbumName = state.track_window.previous_tracks[1].album.name;
    let prevartistName = state.track_window.previous_tracks[1].artists
        .map(artist => artist.name)
        .join(", ");
    let prealbumArt = state.track_window.previous_tracks[1].album.images[0].url;

    insertPrev();

    document.getElementById("prealbumArt").src = prealbumArt;
    document.getElementById("prevtrackName").innerHTML = prevtrackName;
    document.getElementById("prealbumName").innerHTML = prealbumName;
    document.getElementById("prevartistName").innerHTML = prevartistName;
}
else
{
    let prevtrackName = state.track_window.previous_tracks[0].name;
    let prealbumName = state.track_window.previous_tracks[0].album.name;
    let prevartistName = state.track_window.previous_tracks[0].artists
        .map(artist => artist.name)
        .join(", ");
    let prealbumArt = state.track_window.previous_tracks[0].album.images[0].url;

    insertPrev();

    document.getElementById("prealbumArt").src = prealbumArt;
    document.getElementById("prevtrackName").innerHTML = prevtrackName;
    document.getElementById("prealbumName").innerHTML = prealbumName;
    document.getElementById("prevartistName").innerHTML = prevartistName;
}
```

Figure 37: Parsing JSON data and inserting into DOM Source Code

Spotify Search

Tapes allows users to search for individual tracks or artists. Once the user has entered and selected the search icon, the query they entered is passed to a search function. The search function parses this query value and carries out a fetch call to the Spotify API. The query value is passed into this fetch along with the Spotify search endpoint URL and additional query parameters. Once the results are retrieved, they are parsed and output to the user.

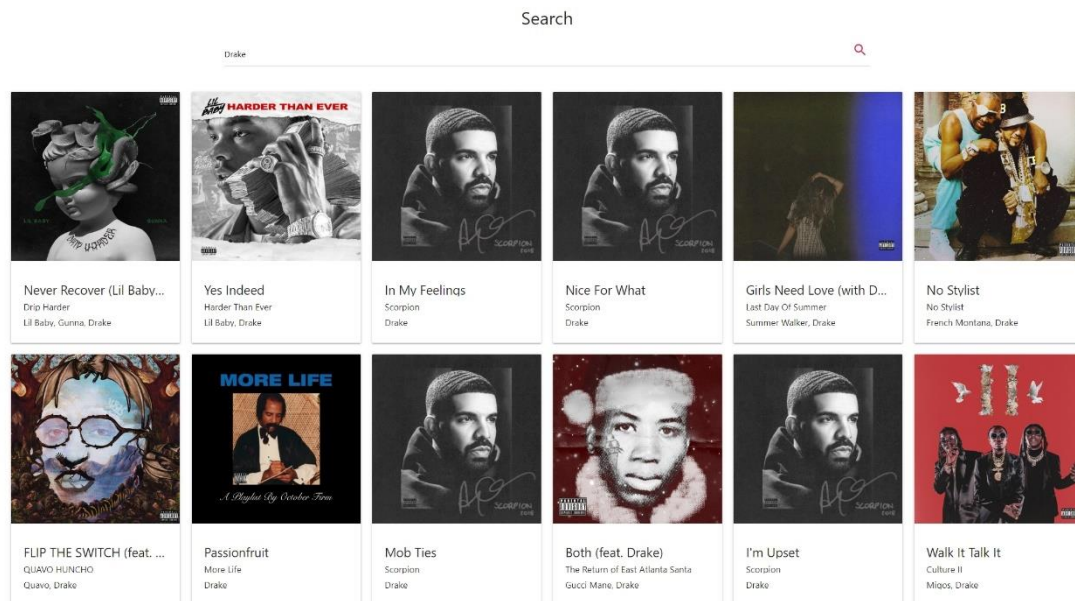


Figure 38: *Tapes* Spotify Search and Results

The user is then able to select a track and it will begin to play in the player. The ability to play a specific track from *Tapes* into the player proved challenging due to the player working off a specific device. In order to play a track, the users current active device had to be found first, this was done with a fetch call to the device endpoint. The device endpoint retrieves a JSON list of the user's devices. This data then needed to be searched in order to find the device id of the current active device connected to the *Tapes* application. Once the device id is found it can be passed along with the selected URI of the track the user would like to play to a fetch call. This then plays the selected track in the *Tapes* player.

```

function getDevice()
{
    return fetch("https://api.spotify.com/v1/me/player/devices",{
        method: 'get',
        headers: { 'Authorization': 'Bearer ' + accessToken}
    })
    .then(res => res.json())
    .catch(error => console.log(error))
}

function playSong(uri)
{
    getDevice().then(function(res){
        //console.log(res);
        let i = 0;
        let deviceid = "";

        do{
            i++;
            if(res.devices[i].is_active == true)
            {
                deviceid = res.devices[i].id;
            }
        }while(res.devices[i].is_active != true)

        url = "https://api.spotify.com/v1/me/player/play?device_id=" + deviceid;
        data = {"uris": [uri]};

        fetch(url,{
            method: 'put',
            headers: { 'Authorization': 'Bearer ' + accessToken},
            body: JSON.stringify(data)
        })
        .then(res => res.json())
        .catch(error => console.log(error))
    })
}

```

Figure 39: Play Source Code

Spotify Playlists

Tapes retrieves the user's current playlists and displays them onscreen to the user; this allows them to easily navigate and switch playlists. In order to retrieve the user's playlists a fetch call was made to the Spotify playlist endpoint. The results were passed back in JSON and parsed in order to be displayed to the user. Similar to how individual songs are played on the *Tapes* player the playlist URI is passed to a play playlist function which retrieves the current active device and a fetch call is made in order to play the playlist. However, the query parameter that the playlist is attached to is context_uri instead of uri which was used to play individual songs.

Playlists

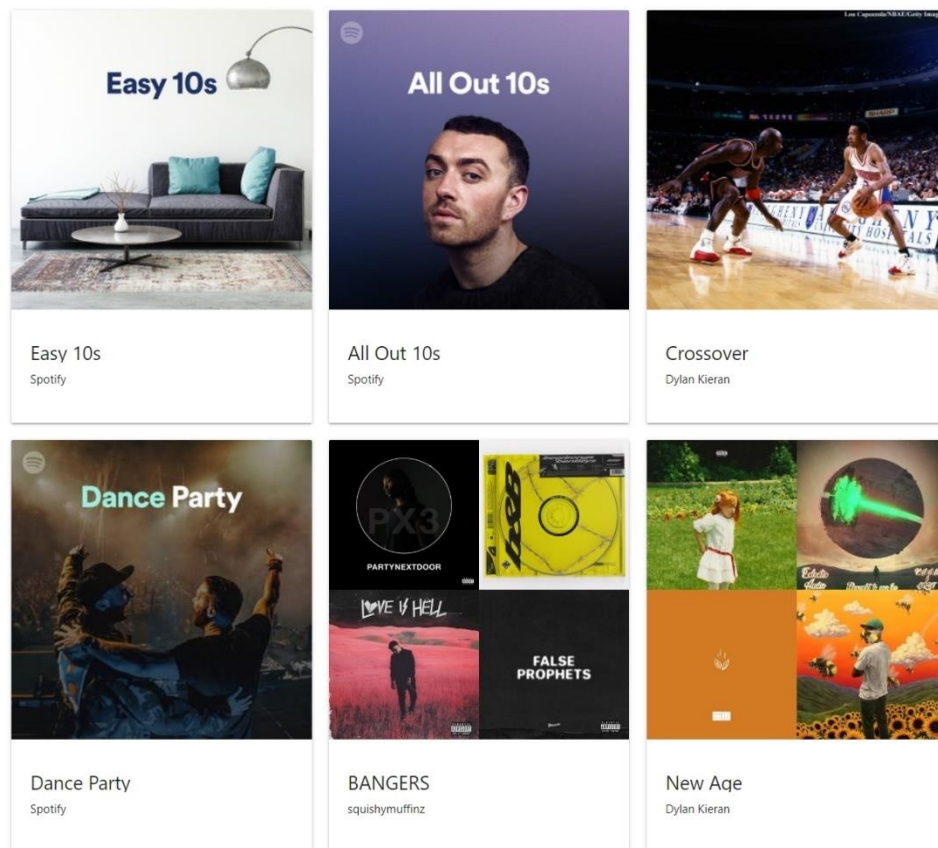


Figure 40: Tapes Spotify Playlists

Spotify Recommendations

The Spotify recommendation system works off two key methods collaborative filtering and a taste profile. Using collaborative filtering, Spotify finds playlists that are similar to the user's and then recommends songs from those playlists. Spotify also builds a taste profile, which can predict genres and micro genres the user likes based on songs, artists and playlists that the user repeatedly listens to and then recommends songs from similar categories. Spotify also learns which songs, artists and playlists the user doesn't like based on songs the user skips.

The *Tapes* application utilises the Spotify recommendation system. The current track id is passed into a fetch call along with the recommendation endpoint. This then returns a list of recommendations that are parsed and displayed to the user. The recommendations are based on the current track that is playing but also takes into consideration the user's taste profile. Recommendations can then be selected and played like the search results.

Song Recommendations

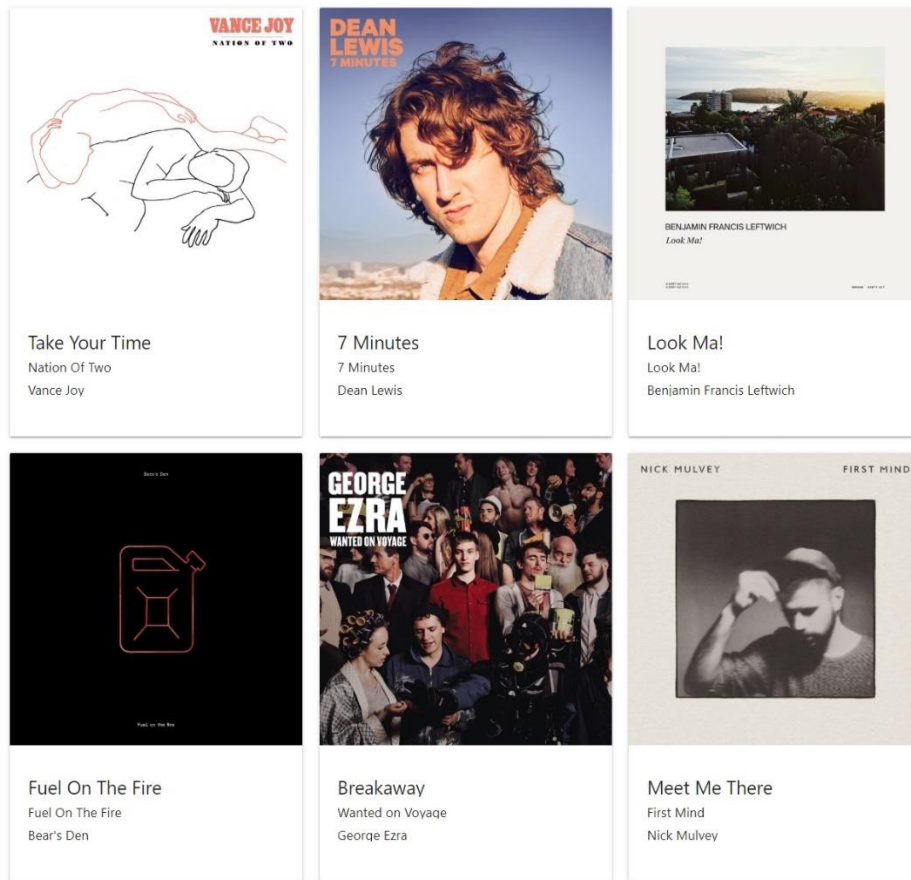


Figure 41: Tapes Spotify Recommendations

4.2.2 YouTube

YouTube provide developers with several different APIs. For the purposes of this application the YouTube iFrame Player API and the YouTube Data API were used.

The iFrame player API allows a developer to embed YouTube videos in their websites and control the player using JavaScript. The APIs JavaScript functions provide the ability to play, pause or stop videos, adjust the player volume or retrieve information about the video being played. Event listeners can also be added for certain player events such as change in player state or video playback quality change. The YouTube Data API provides JSON data on resources such as channel, video, playlist and subscription information.

Unlike the Spotify API, the user does not have to be logged into YouTube in order to access its features. Because of this, the authentication server was not needed, instead a client authorization code generated in the google developer console is used to authorize requests to the API.

YouTube Player

Once the YouTube page is selected the YouTube player is initialised loading a pre-selected video for the users. The user can also load a video from the home screen of the *Tapes* application. The user is displayed with a list of options to the left of the page. The user can search for a video, browse the video queue, queue a playlist, browse top trending music videos, or browse recommended playlists for genres such as pop or rap.

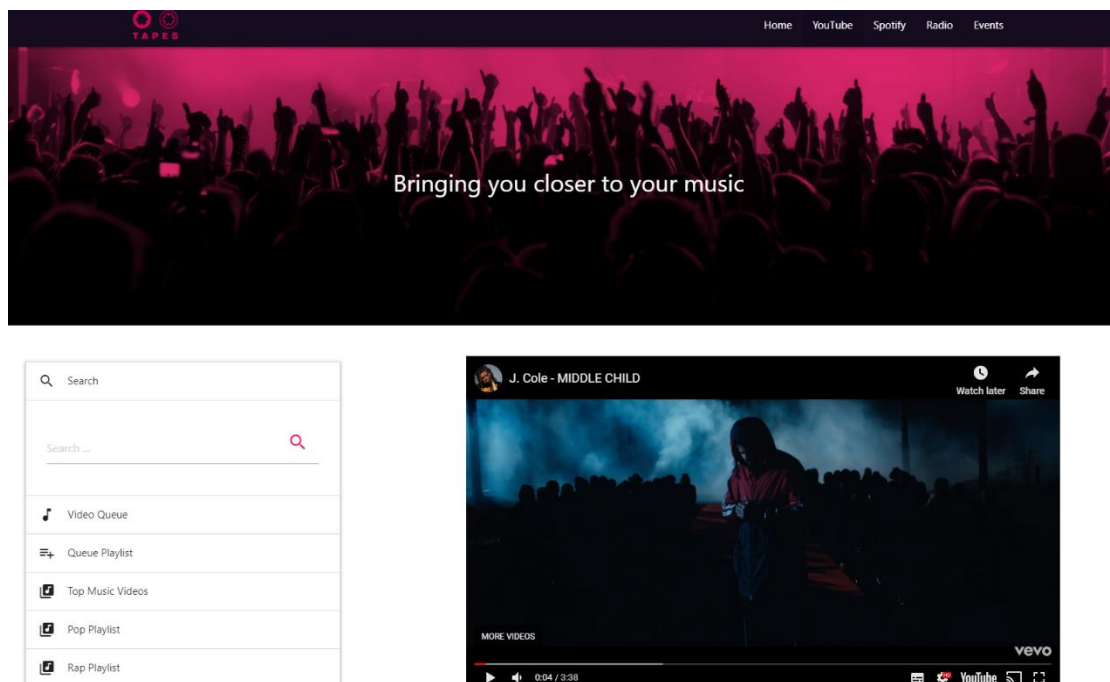


Figure 42: Tapes YouTube

YouTube Search

In order to search for a video, the user must input their query and click the search button. This query value is retrieved and passed as a parameter along with the YouTube key and the max number of results to a search function. The search function removes the spaces in the query and adds a plus for every space in order to query the YouTube API. The query returns a set of JSON data. The data is then parsed, and specific information is taken and displayed to the user. The thumbnail and video title are displayed to the user along with an add to

queue button. The thumbnail can also be selected to add to the queue; however, this will add the video to the beginning of the queue rather than the end, for instant playback.

```
//YouTube Search Function
function youtubeSearch(YouTubeKey, youtubeQuery, maxResults)
{
    let q = youtubeQuery.split(' ').join('+');

    return fetch('https://www.googleapis.com/youtube/v3/search?part=snippet'
+ '&maxResults=' + maxResults
+ '&q=' + q
+ '&key=' + YouTubeKey,{
    method: 'get'
})
    .then(res => res.json())
    .catch(error => console.log)
}

//Populate div with search results
document.getElementById("searchYoutube").onclick = function()
{
    let youtubeQuery = document.getElementById("youtubeQuery").value;
    console.log(youtubeQuery);

    youtubeSearch(YouTubeKey, youtubeQuery, 15).then(function(res) {

        let id = 0;
        console.log(res);
        document.getElementById("search-results").innerHTML = null;

        for(let i=0; i<15; i++)
        {
            if(res.items[i].id.kind === "youtube#video")
            {
                document.getElementById("search-results").innerHTML +=
                "<li id=\"" + res.items[i].id.videoId + "\" class=\"collection-item avatar\">" +
                "<img src=\"" + res.items[i].snippet.thumbnails.high.url + "\" alt=\"" +
                "class=\"circle\" onclick=\"addtoqueue('" + res.items[i].id.videoId + "')\">" +
                "<div class=\"col s10 offset-s1\">" +
                "<p class=\"title\">" + res.items[i].snippet.title + "</p>" +
                "<i class=\"secondary-content material-icons small pink-text\">" +
                "onclick=\"addtoqueue('" + res.items[i].id.videoId + "')\">add_to_queue</i>" +
                "</div>" +
                "</li>";
                id++;
            }
        }
    })
};
```

Figure 43: Tapes YouTube Search Source Code

YouTube Queue

One challenge faced with the YouTube iFrame Player API is that it did not provide functionality for queuing videos. It allowed for a list of videos to be played but did not provide queuing functionality so this needed to be implemented.

For an item to be queued a video must be selected. Videos can be queued by selecting the thumbnail or the add to queue button. If a thumbnail is selected the video will be added to the top of the queue to be played instantly. If the add to queue button is selected the video will be added to the end of the queue.

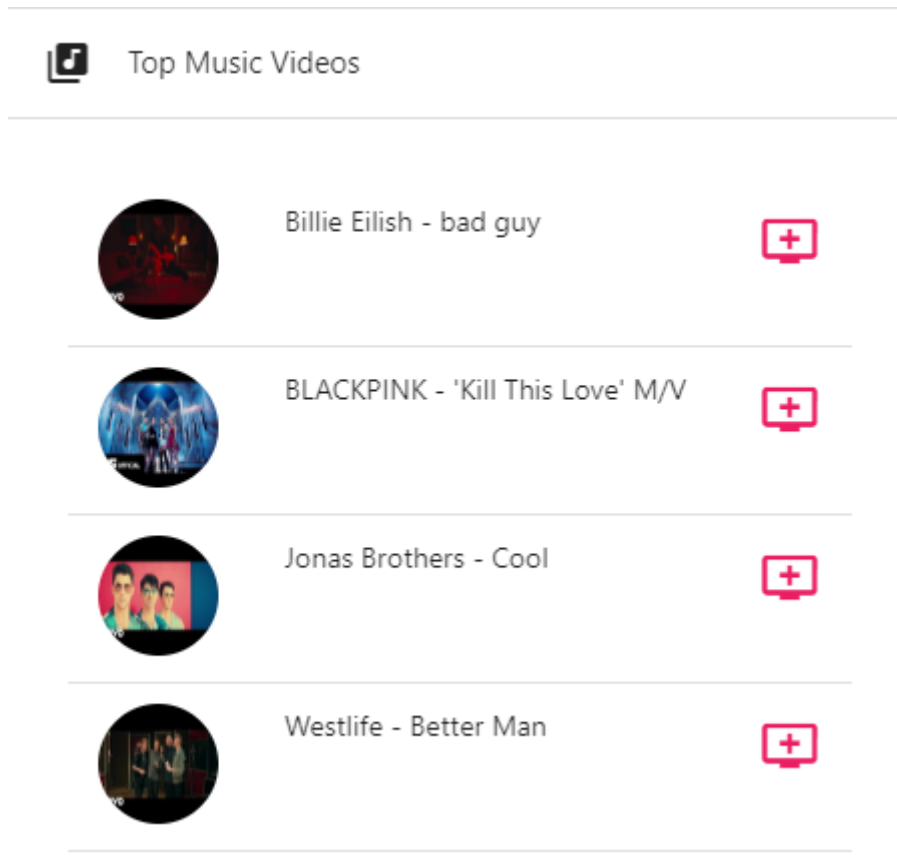


Figure 44: Tapes Queue a Video

If a video is selected by the add to queue button the video id will be passed to an add to queue function. By using the video id this function retrieves the JSON data associated with it, parses it, and creates a DOM element for the video in the video queue. The video id is then pushed to a queue array. Once the video is added to the array the YouTube player queue needs to be re initialised in order to be able to skip videos.

```

//Queue
let Queue = [];

// Add item to queue
function addToQueue(videoId)
{
    //Add song to bottom of Queue
    retrieveVideoInfo(videoId).then(function(res)
    {
        console.log(res.items);
        insertQueue(res);
    })

    Queue.push(videoId);
    console.log("Video added:" + Queue);
    addQueue(Queue);
}

// Retrieve Youtube video information
function retrieveVideoInfo(videoId)
{
    return fetch('https://www.googleapis.com/youtube/v3/videos?part=snippet'
    + '&id=' + videoId
    + '&key=' + YouTubeKey,{
        method: 'get'
    })
    .then(res => res.json())
    .catch(error => console.log(error))
}

//Populate Queue list with added song
function insertQueue(res)
{
    console.log(res);

    document.getElementById("queue").innerHTML +=
    "<li id=\"" + res.items[0].id + "\" class=\"collection-item avatar\">" +
    "<img src=\"" + res.items[0].snippet.thumbnails.high.url + "\" alt=\"" + res.items[0].id + "\" class=\"circle\" onclick=\"addToQueue('" + res.items[0].id + "')\">" +
    "<div class=\"col s10 offset-s1\">" +
    "<p class=\"title\">" + res.items[0].snippet.title + "</p>" +
    "<i class=\"secondary-content material-icons small pink-text\" onclick=\"removefromQueue('" + res.items[0].id + "')\">remove_from_queue</i>" +
    "</div>" +
    "</li>";

    queueToast(res.items[0].snippet.title);
}

```

Figure 45: Queue Source

When a video is selected by its thumbnail, the video id is passed to an alternate add to queue function. This function again retrieves the videos information via JSON. This retrieved data is then passed to a function that parses the videos data and adds it to the top of the video queue list by creating a DOM element. The video id is then passed to an unshift function that adds the new video id to the top of the queue array. Once the video is added to the array the YouTube player queue again needs to be re initialised in order to be able to skip videos.

Once the video queue is populated with items, users have the ability to remove items from the queue by selecting the remove from queue button. Once selected the button passes the video id to the remove from queue function. This function gets the position of the video id from the queue array and removes it by using a splice function. The video is then removed from the DOM by finding the element by its video id and removing it. Once the video is removed from the array the YouTube player queue again needs to be re initialised in order to be able to skip videos.

For usability, toasts were implemented for the queue functions. Toasts appear on screen when a video is added or removed from the video queue or when a playlist is added to the queue.

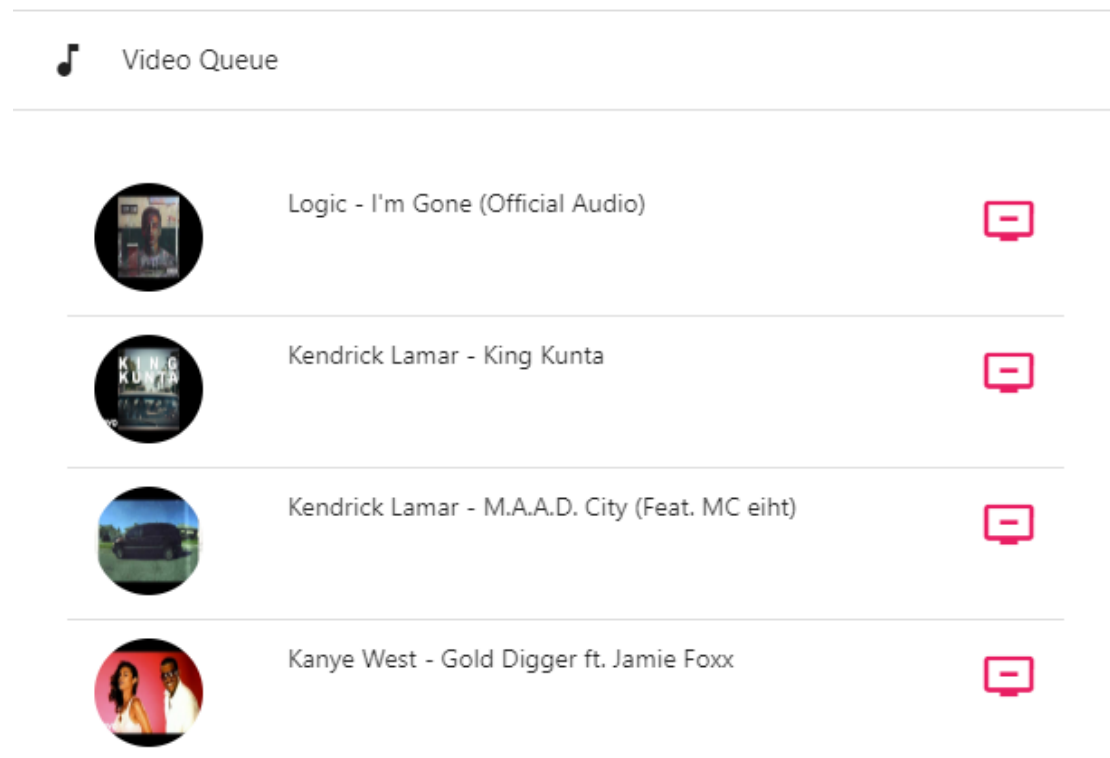


Figure 46: Tapes Video Queue

YouTube Playlists

Users can search and queue a playlist in the *Tapes* application. However due to limitations with the YouTube API users can only search by the playlist id, not by name.

When the user has input the playlist id and pressed the search button, a function is called that retrieves the playlists information. The DOM is then updated displaying the results from the playlist search to the user. The user can then add the playlist to the video queue by pressing the add to queue button. The playlist id is then passed to an add playlist function. This function uses a fetch function to search for the videos contained within that playlist via the YouTube Data API. The videos are then added to the video queue as previously mentioned.

The user is provided with three pre-determined recommended playlists that they can also choose music from.

4.2.3 Radio

In order to listen to a radio station on a webpage, one must obtain an audio streaming link for that station. Several solutions were tested in order to obtain these streaming links. The first solution was to retrieve the links through an API. The Dribble API was tested but failed to provide relevant streaming links. The Shoutcast API was also tested here but too failed to provide streaming links. Instead a manual solution was taken. Streaming links were taken from the listenlive.eu website.

This enabled the application to have several popular radio stations listed and allowed users to listen in on any they chose.

Streaming links were embedded within audio tags as the source of the audio. A control parameter was also included that allows the user to adjust audio levels and play/pause the stream.

```
<div class="col s3">
  <div class="card">
    <div class="card-image">
      
      <span class="card-title">RTE 2fm</span>
    </div>
    <div class="card-content">
      <audio controls src="http://icecast1.rte.ie/2fm" type="audio"></audio>
    </div>
  </div>
</div>
```

Figure 47: Radio Source Code

4.2.4 Events

Several different solutions were tested for the events feature of the application. The Ticketmaster API was chosen over alternative APIs as it allows events to be sorted by relevancy which provides an immediate list of relevant upcoming events over an endless list of niche events.

The current location of the user is gathered by use of the geolocation API. The get current position function initiates an asynchronous request to detect the user's position and queries

the positioning hardware to get up-to-date information. Once the latitude and longitude of the user is found it's passed into a get location function.

The get location function makes a fetch call to the Geonames API, passing with it the user's current latitude and longitude. The Geonames API response with the ISO code or country code related to the latitude and longitude.

The ISO code is then passed to a return events function, this function carries out a fetch call to the Ticketmaster API which includes a set of parameters. The parameters consist of the classification name, the country code, sorting information, size and the API key. The gathered ISO code is passed in as the country code, the classification name is set to music, the sorting code is set to relevance and the API key is passed in.

A response is then received from the API, this response consists of all ticket types for every relevant music event. The tickets are then filtered to show only general admission tickets. Once this is complete the JSON data is parsed and output to the user within a Materialize collection. A button allows users to be redirected to the events page on Ticketmaster where they can choose to book tickets to the event.

```

navigator.geolocation.getCurrentPosition(function(location) {
  const lat = location.coords.latitude
  const lng = location.coords.longitude

  getLocation(lat, lng).then(function(iso)
  {
    returnEvents(iso).then(function(res)
    {
      console.log(res);
      document.getElementById("event-list").innerHTML = null;
      let id=0;
      let date1 = "";

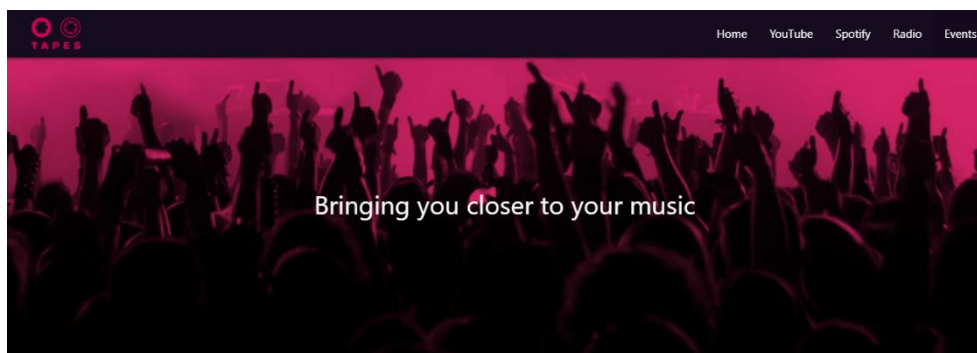
      for(let i=0; i<50; i++)
      {
        if(res._embedded.events[i].name.includes('Vip') ||
           res._embedded.events[i].name.includes('Platinum') ||
           res._embedded.events[i].name.includes('VIP'))
        {
          id++;
        }
        else
        {
          console.log(res._embedded.events[i].name);

          date1 = new Date(res._embedded.events[i].dates.start.dateTime);
          console.log(res._embedded.events[i].dates.start.dateTime)

          document.getElementById("event-list").innerHTML +=
            "<li class='collection-item avatar'>" +
            "<img src='\" + res._embedded.events[i].images[7].url + \"' alt='\" class='circle'>" +
            "<div class='col s8 offset-s1'>" +
            "<span class='title'>" + res._embedded.events[i].name + "</span>" +
            "<p class='truncate title'>" + date1 + "<br></p>" +
            "<p>" + res._embedded.events[i]._embedded.venues[0].name +
            "<a href='\" + res._embedded.events[i].url +
            "\" class='secondary-content pink waves-effect waves-light btn'><i class='material-icons left'>event</i>Book</a>" +
            "</div>" +
            "</li>";
          id++;
        }
      }
    })
  })
});

```

Figure 48: Events Source Code

















Events Near You		
	Eagles Sat Jul 06 2019 18:30:00 GMT+0100 (Irish Standard Time) 3Arena	 BOOK
	Eagles Mon Jul 08 2019 18:30:00 GMT+0100 (Irish Standard Time) 3Arena	 BOOK
	PINK - Beautiful Trauma World Tour Tue Jun 18 2019 18:00:00 GMT+0100 (Irish Standard Time) RDS Arena	 BOOK
	Metallica - Worldwired Tour Sat Jun 08 2019 14:00:00 GMT+0100 (Irish Standard Time) SLANE CASTLE	 BOOK
	Jimmy Buffett Sat Sep 21 2019 20:00:00 GMT+0100 (Irish Standard Time) Olympia Theatre	 BOOK
	Cher - Here We Go Again Fri Nov 01 2019 20:00:00 GMT+0000 (Greenwich Mean Time) 3Arena	 BOOK
	Cher - Here We Go Again Sun Nov 03 2019 20:00:00 GMT+0000 (Greenwich Mean Time) The SSE Arena, Belfast	 BOOK

Figure 49: Tapes Events

4.3 Back-End Development

It was discovered through the development of the prototype that an authorization server would need to be developed. This authorization server would act as a medium of communication to Spotify when the user attempts to login.

4.3.1 Authentication Server

An authentication server was created in order to process Spotify login and authorization. The server makes use of Node.js modules express, request and querystring.

When the user clicks on the sign in button on the client it sends the user to the authentication server which redirects them to the Spotify authorization page. The user can log into the Spotify service directly or via Facebook. A scope is established within the server and passed to Spotify; this outlines what features the *Tapes* application has access to. The user can then decide to approve or deny access.

For the user to approve access to the application the server must pass the client id and client secret id. These ids are obtained from the Spotify developer dashboard and need to be passed to the Spotify authorization page. The client secret key should not be made public, as it can be used for malicious intent. In order to keep this confidential both the client id and client secret id are passed to the server as environmental variables.

Once the user has authorized access, an access token is generated for that user by Spotify and the authentication server passes the token to the client via the address bar. This access token can be used to query the Spotify API endpoints and gain access to the Spotify SDK for audio playback.


```

//Redirect uri that needs to be added to Spotify developer console
let redirect_uri = 'http://localhost:8888/callback'

//Login functionality that send user to Spotify authorization along with the scope
app.get('/login', function(req, res) {
  res.redirect('https://accounts.spotify.com/authorize?' +
    querystring.stringify({
      response_type: 'code',
      client_id: process.env.SPOTIFY_CLIENT_ID,
      scope: 'user-read-recently-played user-follow-read user-modify-playback-state user-library-read',
      redirect_uri
    }))
})

//Upon callback retrieve api access token, callback to application and add it to address bar
app.get('/callback', function(req, res) {
  let code = req.query.code || null
  let authOptions = {
    url: 'https://accounts.spotify.com/api/token',
    form: {
      code: code,
      redirect_uri,
      grant_type: 'authorization_code'
    },
    headers: {
      'Authorization': 'Basic ' + (new Buffer(
        process.env.SPOTIFY_CLIENT_ID + ':' + process.env.SPOTIFY_CLIENT_SECRET
      ).toString('base64'))
    },
    json: true
  }
  request.post(authOptions, function(error, response, body) {
    var access_token = body.access_token
    let uri = process.env.FRONTEND_URI || 'http://localhost:8080/spotify.html'
    res.redirect(uri + '?access_token=' + access_token)
  })
})
}

```

Figure 50: Authentication Server

4.4 Challenges

With any development project there can be several challenges that must be overcome in order to produce a final product on time. The following are challenges that were faced when creating the *Tapes* application.

4.4.1 New Technologies

With any project the introduction of new technologies can be an issue. Unforeseen complications can arise when using unfamiliar technologies. Such issues were encountered at the beginning of the project with the development of the prototype with the Electron framework, as previously mentioned. Additional complication arose with the introduction of additional APIs.

In order to overcome these issues additional testing and research were undertaken in order to come up with more appropriate solutions. Integration testing was performed on several APIs until appropriate solutions were discovered. Test Driven Development methodology aided with the incorporation of new technologies into the application.

4.4.2 API Limitations

Spotify

Whilst the Spotify API and SDK provided some important features to the *Tapes* applications, but it's not without its limitations.

One of the main limitations of the Spotify API is that the users are required to be logged and have a premium account to have access to any of the features the API and SDK provide. Whilst it's understandable to incorporate this feature for audio playback, in order to prevent misuse of the service e.g. increasing play counts, static information such as track, and playlist information would be extremely beneficial to developers and allow them to incorporate some Spotify statistics into their applications without the need for a user to be logged in.

Ticketmaster

The Ticketmaster API was chosen over alternative APIs such as Songkick, due to Songkick's limitations. Songkick was initially tested for the events feature of the application. However, after some consideration and testing it was found that Songkick does not return event images, which was an important design feature for the events page. An additional benefit of using Ticketmaster was that it filtered events that were relevant, giving a more concise list of upcoming music events for the user.

YouTube

YouTube also provided important features to the *Tapes* application, but it too had some limitations.

The YouTube player API does not provide queueing functionality for videos like conventional music streaming services. Queueing functionality had to instead be developed for the *Tapes* application.

An additional limitation of the YouTube API is that it does not provide the ability to search for playlists by name. Instead the playlist must be searched by id and then individual items of the playlist can be extracted and then added to the video queue.

4.4.3 Time Management

Time management is a challenge for any project, with this project especially as there were several phases, the initial proposal, interim report and final report where deadlines needed to be met. It can be difficult to meet deadlines with large projects that have several features and iterations. In order to maintain consistent progress with *Tapes* application, a project plan was created and followed. This will be looked at in detail in the coming chapters.

4.4.4 User Acceptance

Another challenge of the *Tapes* application is user acceptance. As users are the main stakeholder of this application. There is always the risk that the users don't like the application.

This is combated with extensive user testing that was carried out throughout the lifecycle of the application. Initial user testing was carried out in order to gather the fundamental features that should be included into the application. Further user input was taken throughout the course of development and a final user evaluation was conducted towards the end of the development process.

4.5 Conclusions

In conclusion, the development of the application consisted of both front and back end developments. The front-end provided the core functionality of the application whilst the backend provided Spotify authentication. Spotify functionality allows *Tapes* users to play and listen to songs within the *Tapes* application, search, get recommendations and play playlists. YouTube functionality allows user to search and play videos, search and queue playlists and the ability to queue videos, a feature that YouTube does not currently provide. Additional *Tapes* functionality includes the capability to play and listen to radio and find upcoming music events in the user's area.

5. Testing and Evaluation

5.1 Introduction

Testing and evaluating an application are key components in the project lifecycle. Evaluation of an application ensures that it meets the needs of the user whilst testing ensures the quality of the end product. The following section will look at how the *Tapes* application was tested and evaluated.

5.2 System Testing

As previously mentioned, Test Driven Development was the chosen methodology for the *Tapes* application. This allowed for the continuous testing of individual components. With Test Driven Development features of the application were identified and then a test case was created. The feature would then be developed, and a test would be run. If a test passed, development could begin on the next component, if a test failed the component would be refactored and tested again.

The test methods performed varied from Black Box testing to White Box testing. Testing was performed at all levels, unit testing was performed when testing the individual components of the test cases, integration testing was performed when integrating APIs and when configuring separate application components together, system testing was performed when development was complete.

One key aspect of testing of the *Tapes* application was user testing. The *Tapes* application focuses heavily on the user experience because of this several user tests were carried out.

5.2.1 Test Levels

A level of system testing is a process where every unit or component of a system is tested. The goal of system testing is to evaluate the system's compliance with the specified needs. There are many different testing levels which help to check behaviour and performance. The following are the testing levels used to test the *Tapes* application.

Unit Testing

Unit Testing is a testing approach where individual units/components of a software are tested. The aim of the approach is to validate that each individual unit of the software works the way it is intended to be, based on the design. A unit is the smallest part of any software. It tends to have one or few inputs and a single output. It could be an individual program, function or procedure etc. White Box Testing is usually used to test this approach. [10]

Unit testing came hand in hand with the test-driven development approach. Test cases were created for each component of the application and tests were carried out to ensure they worked as expected.

Integration Testing

Integration Testing is a software testing approach where individual units are combined and tested as a group. The aim of this approach is to expose defects in the interfaces and interaction between integrated components. Methods to test this approach include Black Box Testing, White Box Testing and Grey Box Testing. [10]

Once individual components were working as expected they had to be integrated with already working features of the *Tapes* application. A number of tests were conducted to ensure components worked in tandem with each other. Integration testing was also carried out with the various APIs used to ensure their functionality met with expectations.

System Testing

System Testing is the process of testing a completed and integrated software to verify that it meets the specified requirements. Black Box Testing is usually the method used to test this process. [10] Final testing was conducted before user evaluation. Small issues were discovered and subsequently fixed.

Acceptance Testing

Acceptance Testing is testing with respect to the user needs, requirements and business processes conducted to determine whether a system satisfies the acceptance criteria. Black Box testing method is used in Acceptance Testing.

User evaluation of the system was carried out once development was complete. Feedback was given on the application and changes were made with the retrieved feedback.

5.2.2 Test Methods

Black Box Testing

Black Box Testing is a test strategy that involves viewing the program as a black box where internal behaviour and structure of the program are ignored. Test data is created by examining the specifications of the program. The aim of Black Box testing is to find circumstances in which the program does not behave according to its specifications. These can range from incorrect or missing functions, interface errors or errors in data structures or external database access. [10]

Black Box testing was carried out throughout the lifecycle of the project. Frequent testing of the user interface was carried out to ensure features were working as expected. Black Box testing was also carried out by the users of the application during the user evaluation period.

White Box Testing

White Box Testing is a testing strategy that permits the examination of the internal structures of the program. Test data is created from examining the program's logic, often features that are left out or forgotten about in the program's specifications. The aim of White Box Testing is to execute every statement in the program at least once. With this approach the tester chooses inputs to exercise paths through the code and determines the appropriate outputs. [10]

White Box testing was performed throughout development process. The internal workings of the application were tested throughout. White Box testing was particularly useful when implementing the YouTube queue functionality. This was combined with various test plans to ensure features were fully functioning.

User Testing

User Testing is necessary in order to capture user requirements and identify problems with the design or prototype. User testing is a key phase of this project as the users are a key stakeholder in the application. An initial survey was carried out in order to gather the

necessary requirements to start development. The survey consisted of a variety of questions about music and music streaming services. The survey provided necessary requirements that aided in the design of the project. A user evaluation was conducted towards the end of the project, this provided useful feedback on the application, some of which were included into the final developments.

5.2.3 Test Plans

The following test plans were used to carry out testing for key components of the development of the *Tapes* application.

Set Up Environment

Test Increment	Test Description	Test Result	Pass?
1	Create project folders	Success	Yes
2	Install Node.js	Success	Yes
3	Install dependencies	Success	Yes
4	Configure dependencies	Success	Yes
5	Set up GitHub repository for both client and server	Success	Yes
6	Initialise Git repo in project folder	Success	Yes
7	Push to GitHub	Success	Yes

Home Screen

Test Increment	Test Description	Test Result	Pass?
1	Create an improved User Interface using Materialize components	Success	Yes
2	Add navigation menu	Success	Yes
3	Add parallax image of logo to interface	Success	Yes
4	Add tabs for “Top Songs” and “Top Artists”	Success	Yes
5	Create Card containers within a collection list to display “Top Songs” and “Top Artists”	Success	Yes

6	Apply for Last.fm developer key and retrieve the key	Success	Yes
7	Create function to retrieve Top Songs from Last.fm	Success	Yes
8	Create function to retrieve Top Artists from Last.fm	Success	Yes
9	Create function to retrieve Top Songs from Ireland via the YouTube API	Success	Yes
10	Create function to retrieve Top Songs from USA via the YouTube API	Success	Yes
11	Create function to populate the contents of “Top Songs from Last.fm” with the retrieved JSON data	Success	Yes
12	Create function to populate the contents of “Top Artists” with the retrieved JSON data from the Last.fm API	Success	Yes
13	Create function to populate the contents of “YouTube’s Top Songs Ireland” with the retrieved JSON data	Success	Yes
14	Create function to populate the contents of “YouTube’s Top Songs US” with the retrieved JSON data	Success	Yes

Authentication Server

Test Increment	Test Description	Test Result	Pass?
1	Import dependencies	Success	Yes
2	Create an application for Spotify developer	Success	Yes
3	Retrieve Client ID and Client Secret ID from Spotify developer console	Success	Yes
4	Create a call back URL in Spotify developer console	Success	Yes
5	Create a login endpoint for the server	Success	Yes

6	Redirect login to Spotify authorization	Success	Yes
7	Pass Spotify client id and client secret id as environmental variables to authorization	Success	Yes
8	Provide users scope to authorization	Success	Yes
9	Create a call back endpoint which has the same URL as in the Spotify developer console	Success	Yes
10	Retrieve authorization code for user from Spotify	Success	Yes
11	Redirect user to client	Success	Yes
12	Pass authorization token to client via address bar	Success	Yes

Spotify Client

Test Increment	Test Description	Test Result	Pass?
1	Create a sign in button	Success	Yes
2	Create Spotify sign in page	Success	Yes
3	Redirect user to login endpoint on server	Success	Yes
4	Once user has signed in and authorized retrieve access token by parsing the contents of the address bar	Success	Yes
5	Does access token exist	Success	Yes
6	If access token exists – Dynamically update contents of page to show playback instructions	Success	Yes
7	Create playback instruction contents	Success	Yes
8	Create Spotify player once access token is retrieved and device connects to application	Success	Yes
9	Create <i>Tapes</i> player contents	Success	Yes
10	Dynamically update contents of page to show <i>Tapes</i> player	Success	Yes
11	Retrieve JSON for current track list	Success	Yes
12	Update on screen visuals with the contents of the JSON for the current track	Success	Yes
13	Check if the track list has a next track	Success	Yes

14	If track list has next track – Update on screen visuals with the contents of the JSON for the next track	Success	Yes
15	Check if the track list has a previous track	Success	Yes
16	If track list has a previous track – Update on screen visuals with the contents of the JSON for the previous track	Success	Yes
17	Create player events for play, pause, next track, previous track, and audio levels	Success	Yes
18	Create player event HTML and CSS	Success	Yes
19	Create the ability to dynamically add the next track and previous track HTML and CSS if exists	Success	Yes
20	Create next track HTML and CSS by creating DOM elements within JavaScript	Success	Yes
21	Create previous track HTML and CSS by creating DOM elements within JavaScript	Success	Yes
22	Create a get recommendations function	Success	Yes
23	Retrieve and pass in to the function the current track id and access token	Success	Yes
24	Make fetch call to the Spotify recommendation endpoint passing in the track id and access token	Success	Yes
25	Dynamically create and update the contents of the recommendation sub section	Success	Yes

YouTube

Test Increment	Test Description	Test Result	Pass?
1	Create application in Google developer	Success	Yes
2	Retrieve client id	Success	Yes
3	Create and initiate YouTube player	Success	Yes
4	Create YouTube HTML and CSS	Success	Yes
5	Create search function	Success	Yes

6	Pass into search the search query, YouTube key and the max results	Success	Yes
7	Retrieve results and display to the user by updating the search result list	Success	Yes
8	Create a function to create a queue	Success	Yes
9	Create HTML and CSS for queue list	Success	Yes
10	Create a button that allows user to add a video from the search to the video queue	Success	Yes
11	Create function to add video to start of the queue	Success	Yes
12	Ensure item gets added to top of queue in JavaScript and in HTML	Success	Yes
13	Create function to remove item from the queue	Success	Yes
14	Add remove button to items in the queue	Success	Yes
15	Ensure item gets removed from queue in JavaScript and in HTML	Success	Yes
16	Create HTML and CSS for Top Songs	Success	Yes
17	Retrieve Top Songs from YouTube by filtering the search to video category id	Success	Yes
18	Populate Top Songs list with results	Success	Yes
19	Add function that allows user to search for playlist	Success	Yes
20	Retrieve playlist information	Success	Yes
21	Add button that allows users to add the playlist to the queue	Success	Yes
22	Add suggested playlists for different genres (Pop, Rap)	Success	Yes
23	Create CSS that incorporates all elements as a list	Success	Yes
24	Create Toasts that display if the user add/removes items to the video queue	Success	Yes
25	Ensure users can interact with player and video queue	Success	Yes

Radio

Test Increment	Test Description	Test Result	Pass?
1	Include navigation bar	Success	Yes
2	Create HTML/CSS card holders for stations	Success	Yes
3	Download and add appropriate Radio logo's	Success	Yes
4	Find streaming sources	Success	Yes
5	Create playback controls	Success	Yes
6	Add streaming sources as source for audio	Success	Yes
7	Ensure streaming source works	Success	Yes

Events

Test Increment	Test Description	Test Result	Pass?
1	Include navigation bar	Success	Yes
2	Create HTML/CSS card collection to hold events	Success	Yes
3	Set up application for Ticketmaster developer	Success	Yes
4	Retrieve API key	Success	Yes
5	Create a function that retrieves Music events based on relevance and location	Success	Yes
6	Retrieve and manipulate JSON data	Success	Yes
7	Create a function that populates event list with the JSON data	Success	Yes

5.2.4 API Testing

API testing is a type of software testing that involves testing application program interfaces (APIs) directly and as part of integration testing to determine if they meet expectations for functionality, reliability, performance and security.

Extensive testing was carried out on the APIs that were used in the development of the *Tapes* application. API endpoints needed to be tested to ensure that the resulting JSON data contained the necessary information for the application. The JSON data also needed to be examined in order to correctly parse its data to the front end of the application. Since APIs lack a GUI, API testing is performed at the message layer.

Postman is an application used for interacting with HTTP APIs. Postman was used extensively throughout the testing of the application. Postman provides users with a graphical user interface which enables the user to construct requests and read responses.

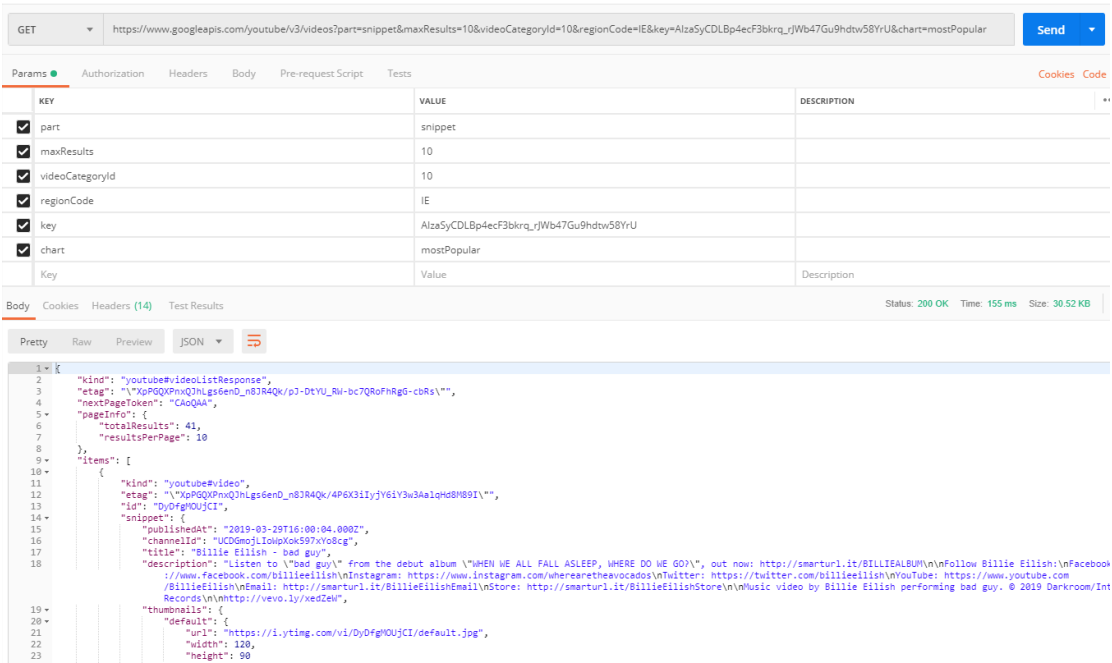


Figure 51: Postman - YouTube Top Music Videos

5.2.5 User Testing

The most important part of the *Tapes* application is the user experience. In order to provide this, user input was needed throughout the project lifecycle. An initial user survey was conducted, it was used to gather information relating to the development and design of the application. The results of the survey were previously mentioned in the research section. The following are the list of questions asked to the users.

Question
Do you listen to music?
How much time do you spend listening to music in a day?
In what place(s) do you listen to music?
On what device(s) do you listen to music on?
Do you use a music streaming service for your music?

What music service(s) do you use?
What feature(s) of these services appeal to you?
What feature(s) of these services do you dislike?
Would you use an application that combines these music services?

A user evaluation was performed towards the end of development. This was used to gather user feedback on aspects of the application such as usability, user interface and user experience. A number of suggestions were made over the course of the evaluation, some of which were incorporated into the last phase of development. The results of the user evaluation will be looked at in detail in the subsequent system evaluation. The following are the list of questions asked to the users.

Question
What features (if any) did you like about the application?
What features (if any) did you dislike about the application?
Would you agree or disagree with the following statements? <ul style="list-style-type: none"> - <i>Tapes</i> was easy to navigate - Spotify functionality was easy to use - YouTube functionality was easy to use - <i>Tapes</i> is a useful resource for combining my favourite streaming services - Functionality worked as expected
How can <i>Tapes</i> be improved?
How likely are you to recommend <i>Tapes</i> to a friend?

5.3 Prototype

An initial prototype was developed for the application. This was used as a base for future developments. The prototype was vital for testing functionality of the main APIs, Spotify and YouTube. The prototype consisted of a backend server and frontend client. The prototype allowed for the testing of pivotal features of the application. Some of the features implemented were:

- An authentication server which initiates user login to Spotify and retrieves user access token which is passed to client.

- Search functionality for Spotify API which allows users to retrieve search results for songs, artists or albums.
- Playback functionality which allows users to use *Tapes* as a Spotify player to play, pause, skip or play previous songs within the application.
- Search functionality for YouTube which allows users to search for music videos from YouTube.
- Playback functionality for YouTube that allows users to watch YouTube videos within the *Tapes* application and control playback.

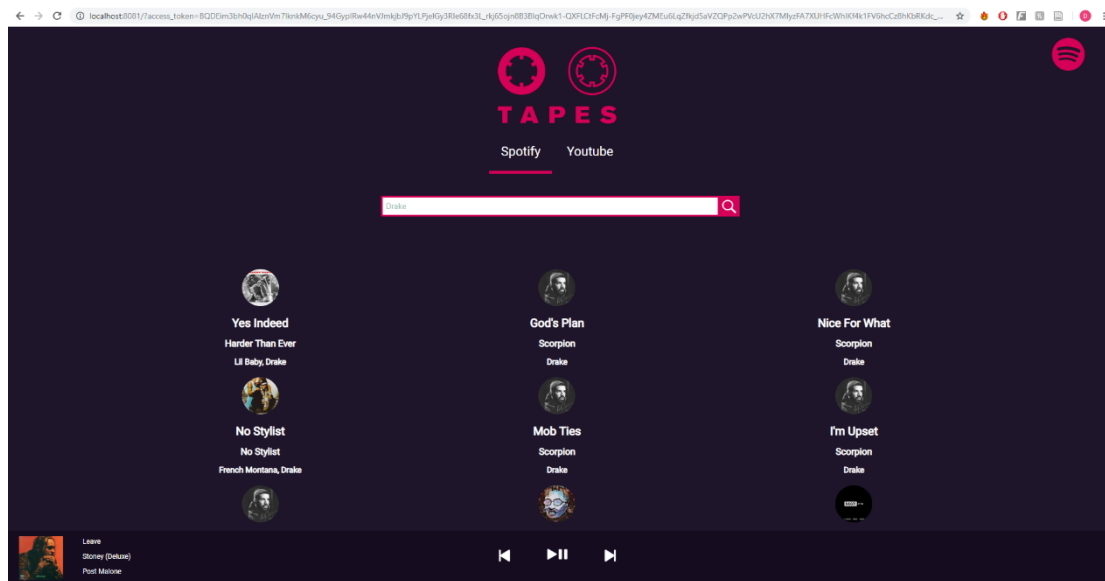


Figure 52: Tapes Prototype - Spotify

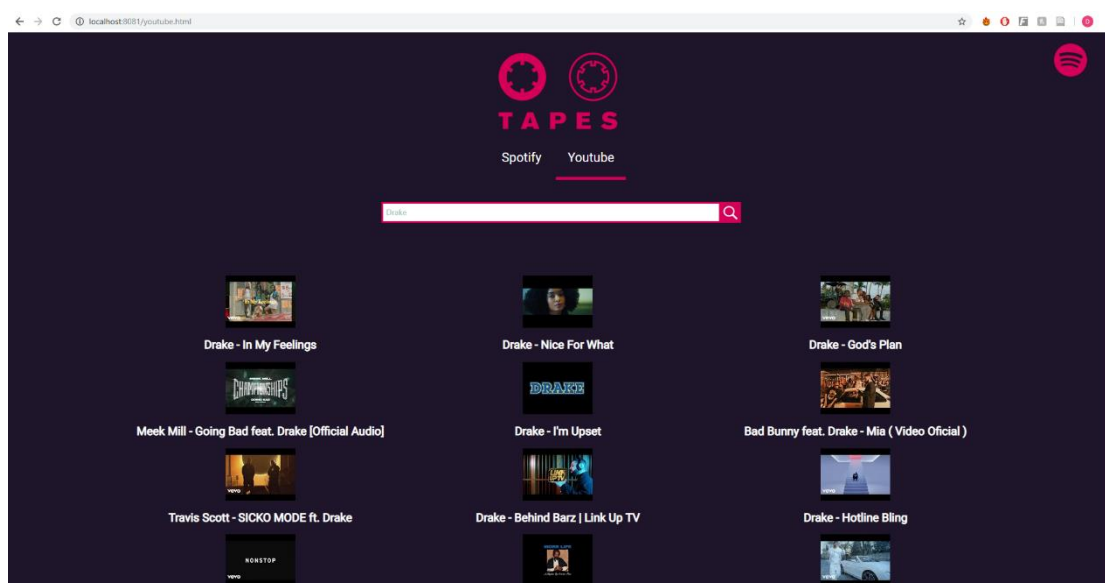


Figure 53: Tapes Prototype - YouTube

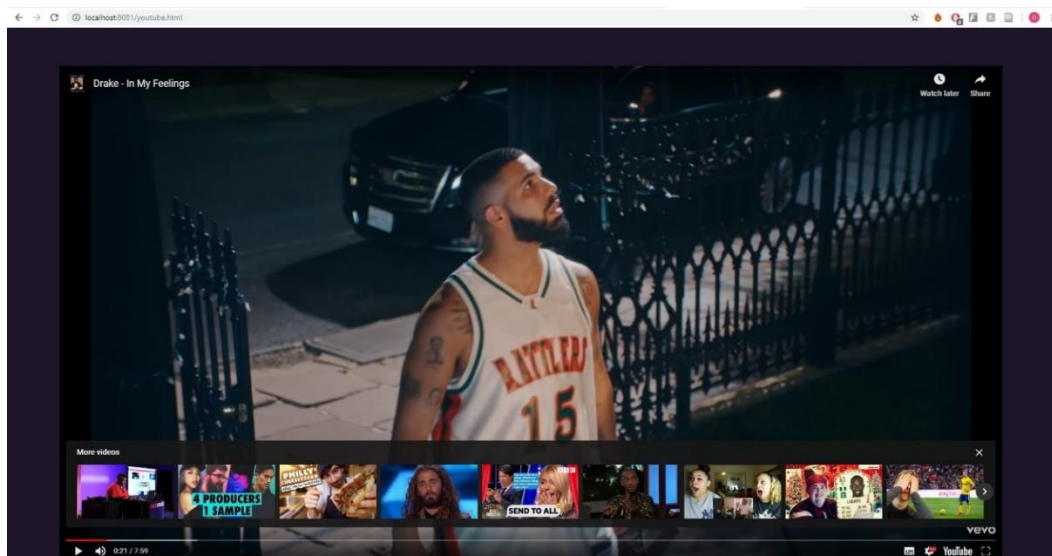


Figure 54: Tapes Prototype - YouTube Player

5.4 User Evaluation

Being that *Tapes* is user focused the application, it was evaluated by users. The user evaluation took place on March 27th, 2019. Candidates were provided with the application and a set of headphones. The think aloud protocol was implemented whereby participants interact and use the application while continuously thinking out loud. Think aloud protocol was very beneficial as it allowed users to voice their misconceptions, which turned into redesign recommendations. Once a user had completed testing the system, they filled in a user survey. The following are the list of the questions asked to the users and their results.

Q. What features (if any) did you like about the application?

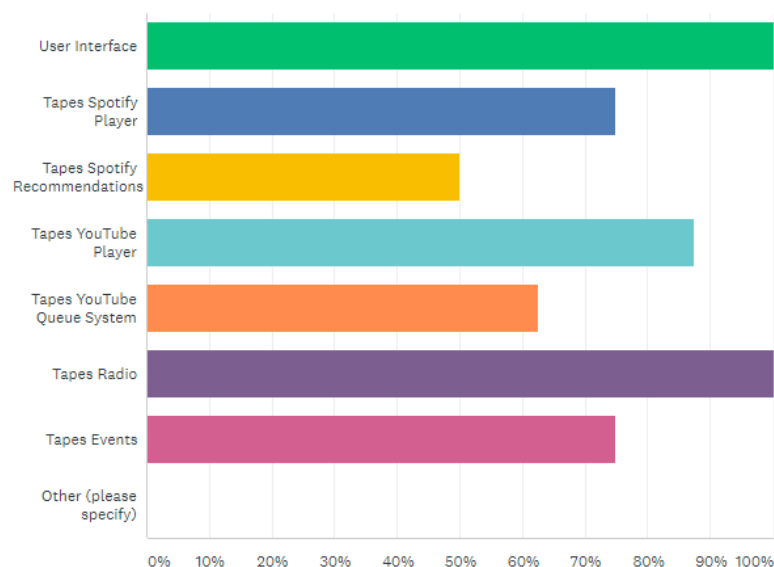


Figure 55: User Evaluation - Q1

Q. What features (if any) did you dislike about the application?

- Spotify Recommendations (Could not play through browser)
- Was unable to access Spotify features as I didn't have a Spotify premium account
- Having to reload Spotify player when loading back into the page
- N/A (x5)

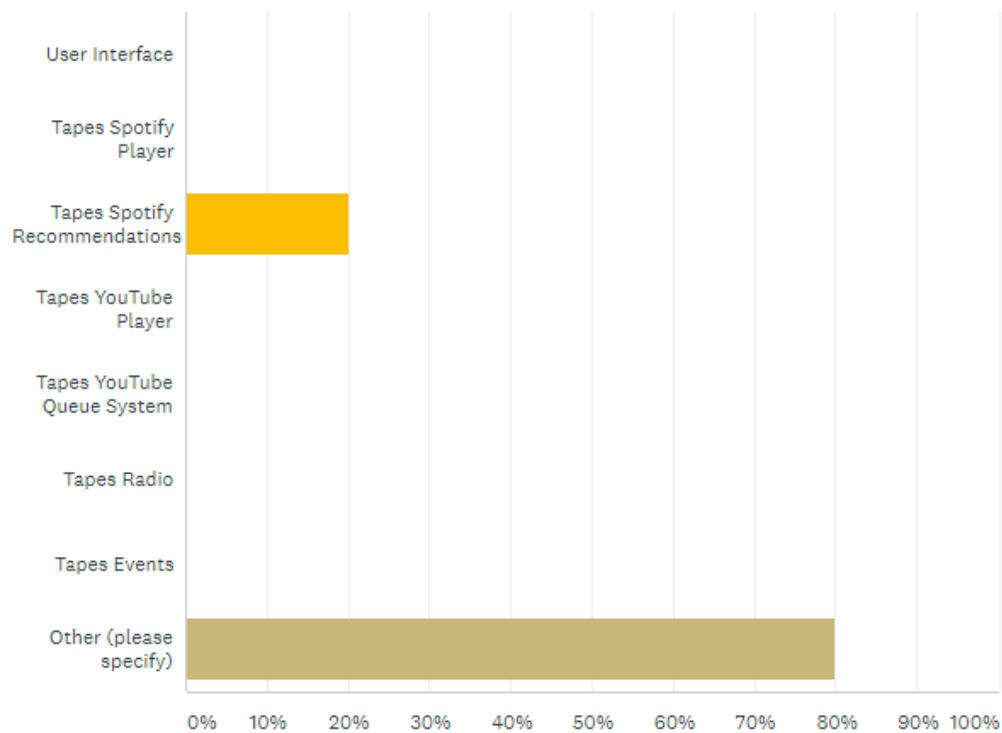


Figure 56: User Evaluation - Q2

Q. Would you agree or disagree with the following statements?

- *Tapes* was easy to navigate (100% reacted positively)
- Spotify functionality was easy to use (62.5% positive, 37.5% neutral)
- YouTube functionality was easy to use (100% positive)
- *Tapes* is a useful resource for combining my favourite streaming services (12.5% negative, 87.5% positive)
- Functionality worked as expected (62.5% positive, 37.5% neutral)

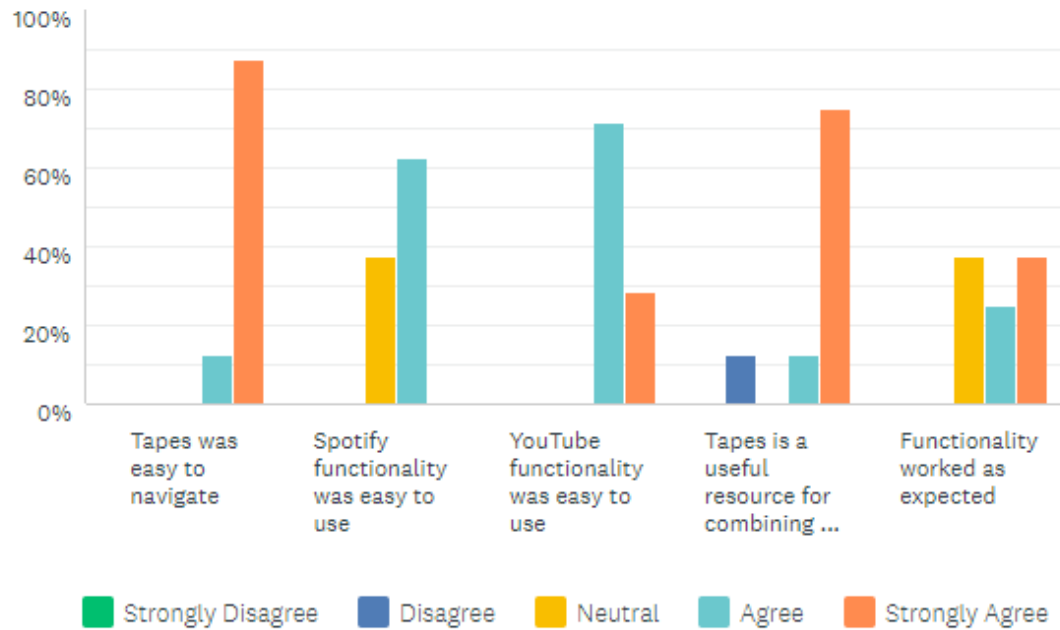


Figure 57: User Evaluation - Q3

Q. How can *Tapes* be improved?

- Links from homepage for top songs and top artists
- Added YouTube functionality
- Apple Music
- Top playlists recommendations
- Incorporate Spotify free feature

Q. How likely are you to recommend *Tapes* to a friend?

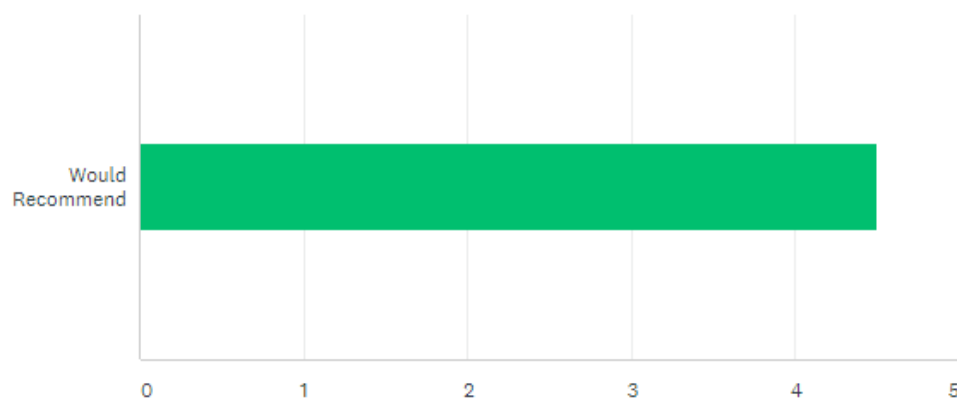


Figure 58: User Evaluation - Q5

Feedback from the user evaluation was overwhelmingly positive users found *Tapes* to be easy to navigate, easy to use, aesthetically pleasing and very useful. Every major component of the application was well received by the users. There were very few features that users disliked. Features disliked were subsequently changed in order to improve the user experience. 90% of users would recommend the *Tapes* application to a friend again reinstating the positive feedback from the users.

Based on recommendations made during the think aloud protocol and from the user survey, changes were made to the application to create a better overall user experience.

Adjustments were made to the home page, users felt that it lacked purpose as it only displayed music charts and top artists. Users can now view the top music videos depending on country, if a video is selected users are redirected to the YouTube player within *Tapes* and the video is played. Additional features were developed after the user evaluation which negated some user feedback such as the ability to select and listen to playlists and recommendations via the *Tapes* Spotify player.

5.5 Conclusions

In conclusion, various methods of testing were carried out to ensure the application was working as necessary and without error. Testing at all levels ensured a complete application that met the needs of the users. Black and White Box Testing ensured the application responded correctly to all kinds of inputs, was sufficiently usable, could perform functions within an acceptable time and ensured interoperability of components. The choice of Test-Driven Development allowed for the creation of test scripts that were used for necessary component testing throughout the development process. User testing ensured the application was designed to meet the needs of the user. This was confirmed with the later user evaluation of the system where feedback was overwhelmingly positive.

6. Project Plan

6.1 Introduction

Project planning is an important phase for any project. Proper planning ensures realistic expectations are set for the project, which further ensures the project maintains steady progress, so deadlines are met. The project plan acts as a roadmap for the project, which allows the project manager to take the project step by step. Changes can be made to the project plan to ensure the necessary requirements are met on time.

For this project there was a set time period of 34 weeks and was broken down into three key phases, the initial proposal, interim submission and final submission. The initial proposal involved background research and a proposal write up which outlined the initial project description and objectives. A project report and prototype were created for the interim phase and the final submission consisted of a final report and the finished *Tapes* application.

A large amount of time was put into the research portion of the project. Background research was carried out in the areas of literature, development technologies, design and alternative existing solutions. The project is user focused therefore a number of user tests were performed including an initial user survey and a final user evaluation.

The following section will focus on the project plan, the established requirements and the changes made to the initial proposal.

6.2 Gantt Chart

In order to meet deadlines and focus the work carried out on the project, a Gantt chart was created. A Gantt chart is a horizontal bar chart that visually represents a project plan over time. The Gantt chart acted as a great visualization and prioritization tool for the project as it provided a total overview of the project. It added clarity to the project as it provided a map of how the project was unfolding. It also improved time management as time periods were allocated for specific aspects of the project. The following is the Gantt chart that was used throughout the lifecycle of the project.

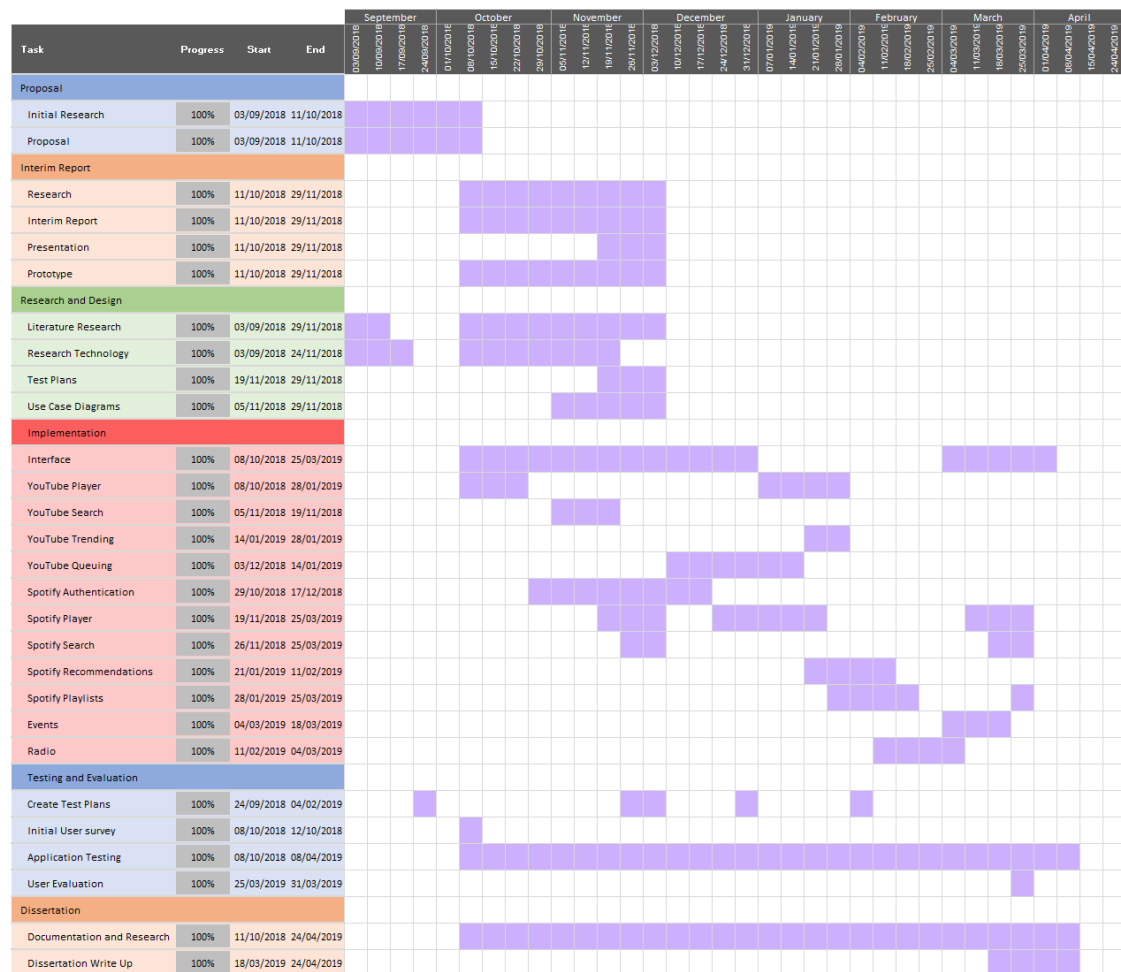


Figure 59: Project Plan - Gantt Chart

6.3 Requirements

Initial Requirements

From the interim report an initial set of requirements was formulated. These requirements formed the base for future developments; however, a number of features could not be implemented for reasons that have been previously mentioned or that will be addressed in the subsequent section.

Requirement ID	Requirement Name	Description	Priority
1	Login	Login to music services	High
2	Search for Spotify Music	Ability to search for music with the Spotify API	Very High

3	Search for YouTube Music	Ability to search for music with the YouTube API	Very High
4	Combine Searches	Create a unified search	High
5	Music Player	The ability to control music playback with a music player displayed at the bottom of the application	Very High
6	Import Playlists from music services	Ability to import users' playlists that they have created on each music service	Medium
7	Creating Combined Playlists	Ability for the user to create a playlist that has music from each service	Medium
8	Song Queue	Create a queue display that allows users to see the order in which songs will play	Medium
9	Suggested Songs	Display songs suggestions based on users listening habits on services	High
10	Trending Songs	Display current trending songs on services	Low
11	Charts	Display current songs on top of the charts for each service	Low
12	Audio Visualizer	Create and display an audio visualizer	High
13	Song Lyrics	Display song lyrics on screen that links to the song the user is currently listening to	Low

Updated Requirements

After further testing and development a new set of requirements were defined. These requirements were based on user testing and feedback and additional developments made after the initial prototype.

Requirement ID	Requirement Name	Description	Priority
1	Display Trending Songs	Display current trending songs	Low
2	Display Trending Music Videos	Display current trending music videos	Low
3	Display Trending Artists	Display current trending music artists	Low
4	YouTube Playback	Allow YouTube playback with the ability to control playback	High
5	YouTube Search	Ability to search and play videos from YouTube	High
6	YouTube Queuing	Ability to queue videos from YouTube	Medium
7	YouTube Playlists	Ability to retrieve and play playlists from YouTube	Medium
8	YouTube Recommendations	YouTube Music recommendations	Medium
9	Spotify Login	Ability to log into Spotify and authenticate the <i>Tapes</i> application	High
10	Spotify Player	Spotify playback within the browser on the <i>Tapes</i> application and provide the ability to control playback	High
11	Spotify Search	Search functionality for Spotify and ability to play selected songs	High
12	Spotify Recommendations	Song recommendations for Spotify	Medium
13	Radio	Ability to play and listen to radio stations	Medium
14	Events	View and book relevant music events that are upcoming in a user's area	Low

6.4 Initial Proposal

The initial proposal for this project is very different to the outcome of the project. The initial proposal was deemed not complex enough and other functionality needed to be proposed. The summary of the initial proposal can be seen below:

The goal of this project is to create a desktop music application that combines a range of music streaming services (Spotify, Soundcloud, YouTube) into one place, all while maintaining a strong user experience (UX) using an intelligent logging and prediction system.

Because of the combination of different service APIs, the idea is to make the application as seamless and transparent as possible in order not to hinder the users experience. To effectively do this the application will take data recordings of common trends in the users listening habits and develop playlists and song suggestions for that user.

The application will include an audio visualizer that changes depending on the beats and notes of the music and will be customizable by the user. The user will have a profile that allows them to see statistics about themselves such as hours listened, favourite artists and recommended artists etc. Each song will have a sidebar that will show the song lyrics, this will be taken from the Genius API.

There are a lot of similarities between the initial proposal and the project outcome, with the core functionality and premise being the same, to create a music application that combines a range of music streaming services into one place, all while maintain a strong user experience. Additional functionality was proposed that did not make it to the final product.

The first additional functionality proposed was a recommender system. The recommender system would provide users with song suggestions based on a user's taste profile. This taste profile would be created by taking data recordings of common trends in the user's listening habits and performing both contents based and collaborative filtering methods to retrieve song recommendations.

Another proposal was an audio visualizer, this feature would have added to the overall user experience of the application. It was intended that an audio stream would be passed from Spotify into an audio visualizer made in JavaScript and the output would be displayed to the user. This would act as an additional feature rather than just listening to a song.

As well as an audio visualizer the user could also view the song lyrics. This would be displayed in a side menu of the audio player.

6.5 Changes to the Proposal

Early into the project, it became apparent that the core functionality was more complex than originally thought. Learning the different elements of each API and incorporating them all into the one application proved challenging due to familiarity and lack of documentation available. Therefore, some of the features suggested in the project proposal could not be implemented due to them being out of the project scope among them a recommendation system and an audio visualizer.

Having a recommendation system for *Tapes* would have been a nice addition to the application but not a necessary one. Each streaming service provides their own unique recommendation service with Spotify and YouTube at the forefront for these recommendation systems. Both APIs provide the ability to make use of their respective recommendation system and output recommendations to the user based off song or video choices.

The inclusion of an audio visualizer would be a welcome addition to the *Tapes* application however, it would be far beyond the scope of the project and would warrant being the sole focus of the project. In place of the audio visualizer *Tapes* has its own Spotify player that provides its own visual stimulus by displaying the current, previous and next track information and album art.

The additional feature of song lyrics, which would be displayed to the user when a video or song is being played, could not be implemented due to licensing restrictions. APIs that provide the ability to retrieve song lyrics cost in excess of €1000 per month. Users however can still view the song lyrics for YouTube videos by enabling the captions on the player.

Another change to the application was the shift from a desktop application to a web application. This transition was made as it was discovered that Electron was not suitable for an application depending on the ability to log into several APIs. After an initial user survey, the idea of a web application arose. A web application made more sense for the *Tapes* application as it would allow users to access the application on several devices at home, school or work without having to develop specific OS versions of the application.

6.6 Conclusions

In conclusion, the Gantt chart used to map the project plan proved beneficial for planning and time management aspects of the project. It allowed for work to be focused on individual development features and other areas of the project such as testing and reporting. The initial list of requirements that were created during the proposal phase of the project were changed based on user feedback and testing. These requirements were then the base of future developments that were carried out on the application. Whilst providing a solid foundation to the project the initial proposal was changed as aspects were beyond the scope of the project.

7. Conclusions and Future Work

7.1 Introduction

This chapter presents the conclusions made for each chapter of the report, as well as the overall project conclusions. The chapter concludes by exploring possible future work ideas.

7.2 Conclusions

Research

Background research was conducted in the areas of Music Streaming and Rich Web Applications. From this, it was made clear that music streaming is the new way forward for the music industry. Music streaming has almost put an end to music piracy as users favour paying small monthly subscriptions to support their favourite artists. Rich web applications are the latest iteration in a series of changes that web applications have undergone in the decades since the web first appeared. A rich web application sees a significant use of JavaScript programming in the client browser which employs a lean data transfer interface with APIs or servers making it the ideal choice for the *Tapes* application. Last.fm, Sonos and Autobeat were looked at as alternative existing solutions. It was found that these solutions either have limited or no access or don't provide an enticing user experience. Based on the findings from the user survey and researched literature it was made clear that *Tapes* should focus on the implementation of both Spotify and YouTube as they are the most popular services. It should also incorporate enhanced user experience by providing additional features suggested in the user survey such as song suggestions, playlists, recommendations and a visual player. Finally, the technologies used for creating the *Tapes* application were selected. Node.js was chosen as the JavaScript framework for the application as it was more suitable and familiar to the developer over alternative frameworks. The Spotify API and SDK, YouTube's Data and Player APIs and Ticketmaster's API were all selected over alternative solutions as they best suited the project. Materialize was then chosen as the CSS framework over Bootstrap as it provided several different features that better matched the design of the user interface.

System Design

In this chapter the system design features were outlined. Test Driven Development was chosen as the project management approach for the project as it puts emphasis on documentation and understanding. It allows for continuous documentation of the

development process which in turn enhances the understanding of the complexity of the project. The technical architecture of the system was defined with emphasis on the Model View Controller principle. The source code of the document was presented, and its contents was outlined. A functional overview was given for the overall *Tapes* application and the Spotify and YouTube features with the aid of use case diagrams. Finally, the different user interface design iterations were presented, and user experience design features were outlined.

System Development

The development of the *Tapes* application consisted of both front and back end developments. The front-end provided the core functionality of the application whilst the backend provided Spotify authentication. Spotify functionality allows *Tapes* users to play and listen to songs within the *Tapes* application, search, get recommendations and play playlists. YouTube functionality allows user to search and play videos, search and queue playlists and the ability to queue videos, a feature that YouTube does not currently provide. Additional *Tapes* functionality includes the capability to play and listen to radio and find upcoming music events in the user's area.

Testing and Evaluation

Several methods of testing were carried out to ensure the application was working as necessary and did not hinder the user's experience. Testing was carried out at all levels with a variety of methods such as White and Black Box Testing. This ensured the application responded correctly to all kinds of inputs, was sufficiently usable, could perform functions within an acceptable time and ensured interoperability of components. Test Driven Development ensured continuous documentation and testing throughout the development cycle as it requires you to create test plans for individual components. Test Driven Development proved beneficial in the testing and development of the application as it ensured continuous progress tracking and reporting. User testing ensured the application was designed to meet the needs of the user. The *Tapes* application is a user focused application therefore a user evaluation was conducted in order to evaluate the application. The user evaluation feedback was overwhelmingly positive which in turn proved the applications success.

Project Plan

A Gantt chart was used to map the plan and track the progress of the project. It proved beneficial in guiding the development of the application and assisted with meeting deadlines as it allowed for work to be focused on individual development features and other areas of the project such as testing and reporting when needed. The initial list of requirements that were created during the proposal phase of the project were changed based on user feedback and testing. These requirements were then the base of future developments that were carried out on the application. Whilst providing a solid foundation to the project the initial proposal was changed as aspects were beyond the scope of the project.

Overall Conclusions

The objective of this project was to design and develop a rich web application that incorporated the multi-sensory qualities of mainstream music streaming services, while maximising the users' experience. Based on the user evaluation feedback, the overall objective of the project has been achieved. Whilst there is always room for improvement the core functionality and project aims are met in the *Tapes* application. The initial user survey and user evaluation allowed development to be focused on the user's needs which in turn aided to the success of the project.

If the project was to be carried out again, several changes would be made. With the knowledge gained from the development of the application the JavaScript framework would be changed from Node.js to React. Whilst Node.js worked well for the purposes of the application, React would increase performance and provide easier functionality for creating and updating DOM elements which was used extensively throughout the application. The original reason for using Node.js was due to familiarity, with no deadlines, an advanced understanding of React could be gathered over a longer time period.

7.3 Future Work

Whilst the *Tapes* application achieved the goals set out at the start of the project there is still a lot of potential future work that can be carried out. Additional functionality and improvements can be made to the existing application as there are no deadlines in place. Features mentioned in the initial proposal can be implemented over a longer period.

Database

For the initial application there was no need for a database as the data was being provided by a variety of APIs. However, if *Tapes* was to grow as an application the curation of a database and most importantly an independent dataset would be of value. The dataset would allow for the creation of joint playlists and a universal player for any streaming service added to the platform. For this to work a song table would store the unique streaming service identifier for that song. In the case of YouTube this would be the video id and for Spotify it would be the URI of the song. If additional services are added to the application their own identifiers for each song would need to be added. This is no easy task; the gathering and cleaning of data would take a substantial amount of time. A provisional ERD can be seen below.

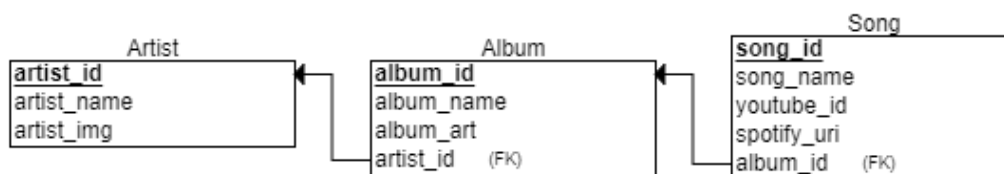


Figure 60: Tapes Database

Recommendations

The addition of a database which allows for the interconnection of the different streaming services, provides the opportunity for a *Tapes* recommendation system. The *Tapes* recommendation system would gather user data based on all services they use on the *Tapes* platform and would offer a joint list of recommendations based on the users listening habits. This recommender system could take a similar approach to Spotify's recommendation system by incorporating both collaborative filtering and content-based filtering methods.

Soundcloud

Soundcloud could not be implemented into the initial application due to their developer service being under re-evaluation. Soundcloud is very popular service for upcoming music artists and producers as it provides a free service that allows creators to share their work. Soundcloud contains millions of tracks that aren't available on any other service. Because of this once their re-evaluation has concluded Soundcloud would be incorporated into the *Tapes* application.

Apple Music

Apple Music was not incorporated into the initial *Tapes* application due to its developer subscription cost of €99 a year. However, if *Tapes* was to have a revenue stream once released, then the addition of Apple Music would be reconsidered and added to the *Tapes* application. By including Apple music *Tapes*' popularity would grow due to the influx of new users that have Apple music subscriptions. Therefore, the cost would be justified.

Audio Visualizer

As previously mentioned, an audio visualizer was included in the initial proposal but was not incorporated into the *Tapes* application as it was out of the projects scope. An audio visualizer would consume a lot of time and require a considerable amount of additional research and development. However, without time constraints this feature could be incorporated into the *Tapes* application. This would add to the multisensory goals of the application. The audio visualizer would work off Spotify data which is provided by the Spotify API. Initial research was conducted on Spotify audio visualizers, the image below is a sample kaleidoscope visualizer that works off the Spotify API.

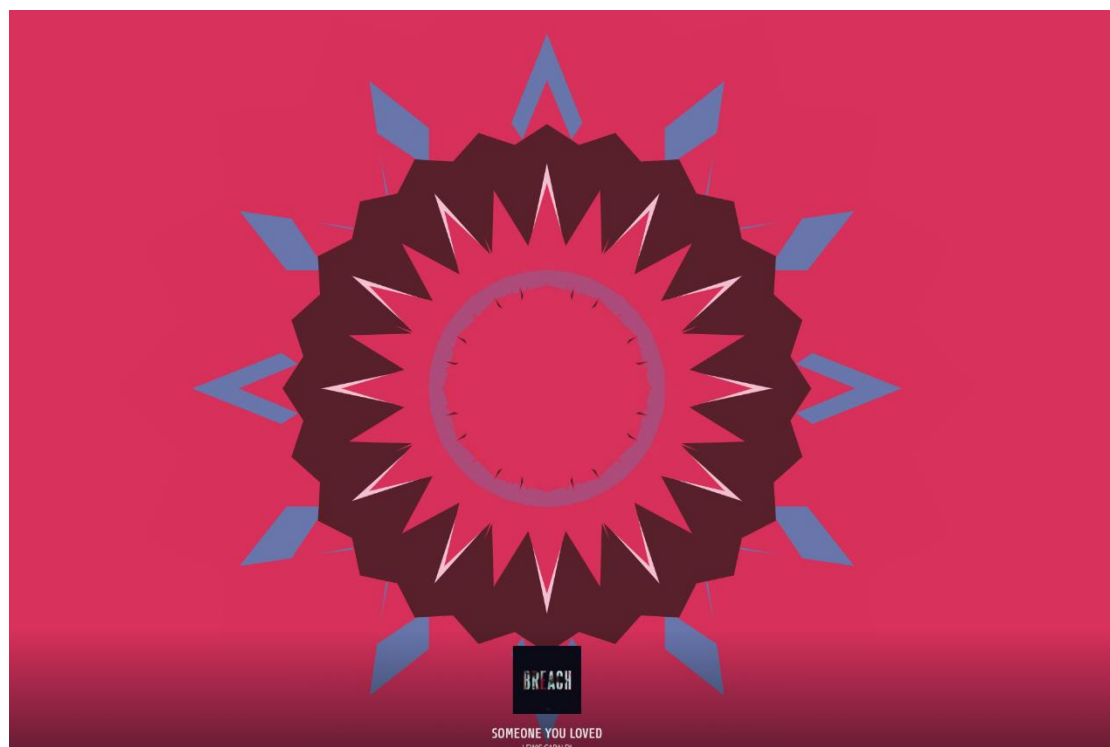


Figure 53: Spotify Kaleidoscope Audio Visualizer [46]

Radio

An improved radio solution could be implemented that allows users to search for specific radio stations. This could also incorporate trending radio stations, a list of stations based on current listener numbers.

Podcasts

With the growing popularity of podcasts, the Tapes application could incorporate a service for podcast listeners which allows them to browse and listen to podcasts.

8. Bibliography

- [1] YouTube, (-), YouTube API, <https://developers.google.com/youtube/> , Date Accessed: October 2018.
- [2] Spotify, (-), Spotify API, <https://developer.spotify.com/documentation/web-api/> , Date Accessed: October 2018.
- [3] Soundcloud, (-), Soundcloud API, <https://developers.soundcloud.com/> , Date Accessed: October 2018.
- [4] Apple Music, (-), MusicKit JS, <https://developer.apple.com/documentation/musickitjs> , Date Accessed: October 2018.
- [5] Electron, (-), Electron Framework, <https://electronjs.org/> , Date Accessed: October 2018.
- [6] Jin Ha Lee & Nichole Maiman-Waterman. (2012) Understanding User Requirements for Music Information Services, http://ismir2012.ismir.net/event/papers/253_ISMIR_2012.pdf , Date Accessed: October 2018.
- [7] Liikkanen, Lassi A. and Åman, Pirkka. (2015) Shuffling services: Current trends in interacting with digital music, https://l.kryptoniitti.com/lassial/files/publications/150420-Shuffling_services_music_interaction_trends.pdf , Date Accessed: October 2018.
- [8] Zachary Krastel, Geneviève Bassellier and Jui Ramaprasad. (2015) Music is Social: From Online Social Features to Online Social Connectedness, <https://pdfs.semanticscholar.org/8d4d/05201efd01bb612a8a14f6593d9ea11a3bb9.pdf> , Date Accessed: October 2018.
- [9] Ahmed Kachkach (2016) Analyzing user behaviour and sentiment in music streaming services, <https://pdfs.semanticscholar.org/e745/027e21705c9219dd1272297a082c5270c9c8.pdf> , Date Accessed: November 2018.
- [10] Glenford J. Myers, Corey Sandler, Tom Badgett (2011) The Art of Software Testing 3rd Edition, Wiley Publishing.
- [11] Roger S. Pressman (2010) Software Engineering: A Practitioners Approach 7th Edition, McGraw Hill.
- [12] Ian Sommerville (2016) Software Engineering 10th Edition, Pearson Education.
- [13] Leon Shklar and Richard Rosen (2006) Web Application Architecture: Principles, Protocols and Practices, John Wiley & Sons, Ltd.
- [14] Sonos, (-), Sonos App, <https://www.sonos.com/en-ie/controller-app> , Date Accessed: October 2018.

- [15] Autobeat, (-), Autobeat Player, <http://www.autobeatplayer.com/>, Date Accessed: October 2018.
- [16] Node.js, (-), Node.js about, <https://nodejs.org/en/about/>, Date Accessed: November 2018.
- [17] AngularJS, (-), AngularJS FAQ, <https://docs.angularjs.org/misc/faq>, Date Accessed: November 2018.
- [18] React, (-), ReactJS Getting Started, <https://reactjs.org/docs/getting-started.html>, Date Accessed: November 2018.
- [19] Oracle, (-), MySQL Documentation, <https://dev.mysql.com/doc/refman/8.0/en/>, Date Accessed: November 2018.
- [20] MongoDB, (-), What is MongoDB, <https://www.mongodb.com/what-is-mongodb>, Date Accessed: November 2018.
- [21] Bootstrap, (-), Bootstrap 3 Documentation, <https://getbootstrap.com/docs/3.3/>, Date Accessed: November 2018.
- [22] Materialize, (-), Materialize About, <https://materializecss.com/about.html>, Date Accessed: November 2018.
- [23] Apple, (-), MusicKit JS, <https://developer.apple.com/documentation/musickitjs>, Date Accessed: November 2018.
- [24] Musixmatch, (-), Musixmatch API, <https://developer.musixmatch.com>, Date Accessed: November 2018.
- [25] Last.fm Music Team. (2015), Artists and Labels: How to make the most of Last.fm, <https://getsatisfaction.com/lastfm/topics/artists-and-labels-how-to-make-the-most-of-last-fm>, Date Accessed: January 2019.
- [26] Last.fm, (-), Last.fm About, <https://www.last.fm/about>, Date Accessed: January 2019.
- [27] Ben Popper, (November 12th 2015), YouTube Music is here, and it's a game changer, <https://www.theverge.com/2015/11/12/9723496/youtube-music-app-offline-background>, Date Accessed: January 2019.
- [28] Spotify, (-), Spotify About, https://support.spotify.com/us/?_ga=2.103916736.1741291046.1554295734-93746253.1529264109, Date accessed: February 2019.
- [29] Spotify, (-), Spotify SDK, <https://developer.spotify.com/documentation/web-playback-sdk/>, Date Accessed: October 2018.
- [30] Mansoor Iqbal (February 27th 2019), Spotify Usage and Revenue Statistics, <http://www.businessofapps.com/data/spotify-statistics/>, Date Accessed: March 2019.

- [31] Raymond Wong (April 24th 2018), Spotify gives its iOS and Android apps a complete overhaul, <https://mashable.com/2018/04/24/spotify-mobile-app-redesign-free-tier-on-demand-playlists/?europa=true#2Gp56tvaHqqF>, Date Accessed: February 2019.
- [32] Craig Smith (March 24th 2019), Interesting Apple Music Statistics and Facts (2019) | By the Numbers, <https://expandedramblings.com/index.php/apple-music-statistics-and-facts/>, Date Accessed: March 2019.
- [33] Apple, (-), Apple Music, <https://www.apple.com/apple-music/>, Date Accessed: January 2019.
- [34] Soundcloud, (-), Soundcloud Contact, <https://soundcloud.com/pages/contact>, Date Accessed: January 2019.
- [35] Craig Smith (March 28th 2019), 16 Amazing Soundcloud statistics and Facts (2019) | By the Numbers, <https://expandedramblings.com/index.php/soundcloud-statistics/>, Date Accessed: March 2019.
- [36] Craig Granell (May 16th 2018), A history of music streaming, <https://www.dynaudio.com/dynaudio-academy/2018/may/a-history-of-music-streaming>, Date Accessed: March 2019.
- [37] Mark Harris (January 23rd 2019), The History of Napster, <https://www.lifewire.com/history-of-napster-2438592>, Date Accessed: March 2019.
- [38] Apple (April 28th 2003), Apple launches the iTunes Music Store, <https://www.apple.com/newsroom/2003/04/28Apple-Launches-the-iTunes-Music-Store/>, Date Accessed: March 2019.
- [39] RIAA (Recording Industry Association of America)(June 2018), Mid-Year 2018 RIAA Music Revenues Report, <http://www.riaa.com/wp-content/uploads/2018/09/RIAA-Mid-Year-2018-Revenue-Report-News-Notes.pdf>, Date Accessed: March 2019.
- [40] Neil Howe (January 16th 2019), How Music Streaming Won Over Millennials, <https://www.forbes.com/sites/neilhowe/2019/01/16/how-music-streaming-won-over-millennials/#2b2269ad25c7>, Date Accessed: April 2019.
- [41] Last.fm, (-), Last.fm Subscribe, <https://www.last.fm/subscribe>, Date Accessed: March 2019.
- [42] Last.fm, (-), Last.fm API, <https://www.last.fm/api>, Date Accessed: November 2018.
- [43] Songkick, Songkick Developer, <https://www.songkick.com/developer>, Date Accessed: January 2019
- [44] Ticketmaster, (-), Ticketmaster Developer, <https://developer.ticketmaster.com/>, Date Accessed: January 2019.

- [45] Fraternali P, Rossi G, Sánchez-Figueroa F. May 2010, Rich internet applications. <https://ieeexplore.ieee.org/abstract/document/5481362>. Date Accessed: October 2018.
- [46] Zach Winter, Kaleidosync, <https://www.kaleidosync.com/>, Date Accessed: February 2019.
- [47] Last.fm, (-), Last.fm Charts, <https://www.last.fm/charts>, Date Accessed: November 2018.
- [48] The Verge (October 9th 2018), Sonos updates its desktop app and adds maximum volume limiter, <https://www.theverge.com/2018/10/9/17956980/sonos-desktop-app-redesign-maximum-volume-limit-feature>, Date Accessed: November 2018.

9. Appendix

Spotify Top Songs

The functionality to display the current Spotify charts was created, however was not implemented into the final application. A simple function was created that fetches the “Global Top 50” playlist from Spotify, this data would have then been parsed and displayed to the user on the Spotify page of the application.

```
// Retrieve Spotify topSongs
function spotify_topSongs(accessToken)
{
  url="https://api.spotify.com/v1/playlists/37i9dQZEVXbMDoHDwVN2tF"

  return fetch(url,{
    method: 'get',
    headers: { 'Authorization': 'Bearer ' + accessToken}})
    .then(res => res.json())
    .catch(error => console.log(error))
}
```

Figure 61: Tapes Spotify Top Songs

Last.fm Top Songs and Artists

The Last.fm API was included into the application. It provides the ability to retrieve the top songs in the current global charts. It can also retrieve the current trending artists on the platform. Functionality was implemented for both features on the initial design of the *Tapes* application. However, following user evaluation these features were removed from the application as they provided little to no purpose only to provide information to the user. They were instead replaced with YouTube charts for trending music videos in different location that could be selected and played in the *Tapes* YouTube player.

LastFM's Top Songs



bad guy
Billie Eilish



7 Rings
Ariana Grande



BURY A FRIEND
Billie Eilish



wish you were gay
Billie Eilish



when the party's over
Billie Eilish



break up with your girlfriend, i'm bored
Ariana Grande



thank u, next
Ariana Grande



you should see me in a crown
Billie Eilish



Sucker
Jonas Brothers



all the good girls go to hell
Billie Eilish

Figure 62: Last.fm Top Songs

Top Artists











	Billie Eilish Playcount: 18246014 1
	Ariana Grande Playcount: 106295362 2
	Queen Playcount: 191515341 3
	Post Malone Playcount: 25498388 4
	Kanye West Playcount: 238435480 5
	Lady Gaga Playcount: 285349057 6
	Radiohead Playcount: 499350957 7
	Kendrick Lamar Playcount: 103051382 8
	Arctic Monkeys Playcount: 332172624 9
	The Beatles Playcount: 516845569 10

Figure 63: Last.fm Top Artists