



LOUGHBOROUGH UNIVERSITY

GROUP PROJECT

Final Report

Team Hydra:
Jay VAGHARIA
Oliver WOODINGS
Simon KERR

May 2015

Contents

1	Introduction	4
1.1	Background	4
1.2	The Customer	4
1.3	The Team	4
1.4	The Idea	5
2	Project Plan	6
2.1	Gantt Chart	7
2.2	Team Roles	7
2.3	Project Management	7
2.3.1	Team Communication	8
2.4	Resources	8
2.5	Software Development Approach	9
2.6	Technical Considerations	11
2.6.1	Mobile Application	11
2.6.2	Server-Side Systems	12
3	Idea Analysis	14
3.1	The Idea	14
3.2	User Value Proposition	16
4	Literature Review	18
4.1	User Interface Design	18
4.2	App frameworks	19
4.3	Effects of Music	20
4.4	Competitors	21
4.5	Music Sources	22
4.6	Legal Issues	23
4.6.1	Public Performance License	23
4.6.2	Entertainment Licensing	24
4.6.3	Music	25
4.7	Geolocation	25
4.8	NFC	26
4.8.1	Viability	26
5	Requirements	27
6	Design	33
6.1	System Architecture	33

7	Implementation	35
7.1	Inter-Service Communication	36
7.2	Audio Streaming	38
7.3	API	39
7.4	Authentication	40
7.5	Speaker Adapter	41
7.6	UI	42
8	Evaluation	45
9	Conclusion	46
	References	47

List of Figures

1	Phases in the James Martin approach to RAD	10
2	The pricing scheme [5]	24
3	Proposed Choona system architecture	33
4	Prototype architecture alongside original proposed architecture	35
5	Inter-service communication using HTTP APIs	36
6	Inter-service communication using Redis	37
7	Inter-service communication using Waterway	38
8	Buffered audio streaming through the Choona audio pipeline	38
9	Auth0 user management	40
10	Choona authentication process	41
11	PhoneGap/Cordova build process ¹	42
12	Choona’s application state/scope tree	43
13	Data lifecycle of the entire Choona ecosystem	44

¹Adobe Systems Inc, *PhoneGap build process*.

1 Introduction

1.1 Background

As part of this module, we were entered into an International IT Challenge run by *Atos*. Atos publicise this competition to many universities where students will attempt to create a solution to the requirements set by Atos. This year the challenge revolved around *Connected Living*. There are many definitions for connected living. Some definitions define it as something constrained to the home. Others describe it as a world where customers use different devices to experience connection anytime, anywhere. This definition covers connected homes, connected work and connected city. Atos believe that connected living involves bringing the home, workspace and city seamlessly together through smart devices providing connectivity anytime, anywhere. Consumers are always wanting to feel more connected to their workspace, their homes and their cities.

Originally, there were a few objectives set by Atos that they wished to be fulfilled:

- Easy Connection of Products
- Big Data Analytics
- Create an Audience
- Client Facing App

We tried to think outside the box for this project and decided not to create an app that would deal with the “connected home” but something that has never been created before and something we feel still meets the criteria laid out by Atos. As a team, we wanted to create something that would enhance peoples social lives and provide a service both to the customers and the businesses involved.

1.2 The Customer

From **Atos**, we are assigned a mentor who acts as a customer. We have been allocated **Mike Smith**; a Chief Technology Officer within Atos. Mike has extensive knowledge of all things technical and has recently written a paper about *Connected Train*. This makes him an ideal customer as he has a good knowledge of the *Connected World*. We aim to tap into this knowledge and gain advice and feedback in order to improve and enhance our idea from the beginning until the end of this module.

On another level, when we speak of customers thorough this project documentation, we will not only be referring to Mike, but also the thousands/millions of potential users for this system if and when it is launched.

1.3 The Team

We have decided to call our team **Hydra**. As all three of us our MCU (Marvel Cinematic Universe fans), we looked there for inspiration. At the heart of Marvel is an organisation

striving for world domination. We aim to do the same with our idea and therefore we have decided to call our team after the name of this organisation; Team **Hydra**. The team includes three Computer Science Masters students; Jay Vagharia, Oliver Woodings and Simon Kerr. Being computer scientists, we have a high interest in technology (with a small geek streak) and a passion for music. To combine both together would allow us to be involved in a project that we are all passionate about and will ensure we are completely motivated to succeed. We have all successfully completed industrial placements, with one team member already working for another company. The experience we have between the three of us should provide us with the necessary skills to tackle and complete this project.

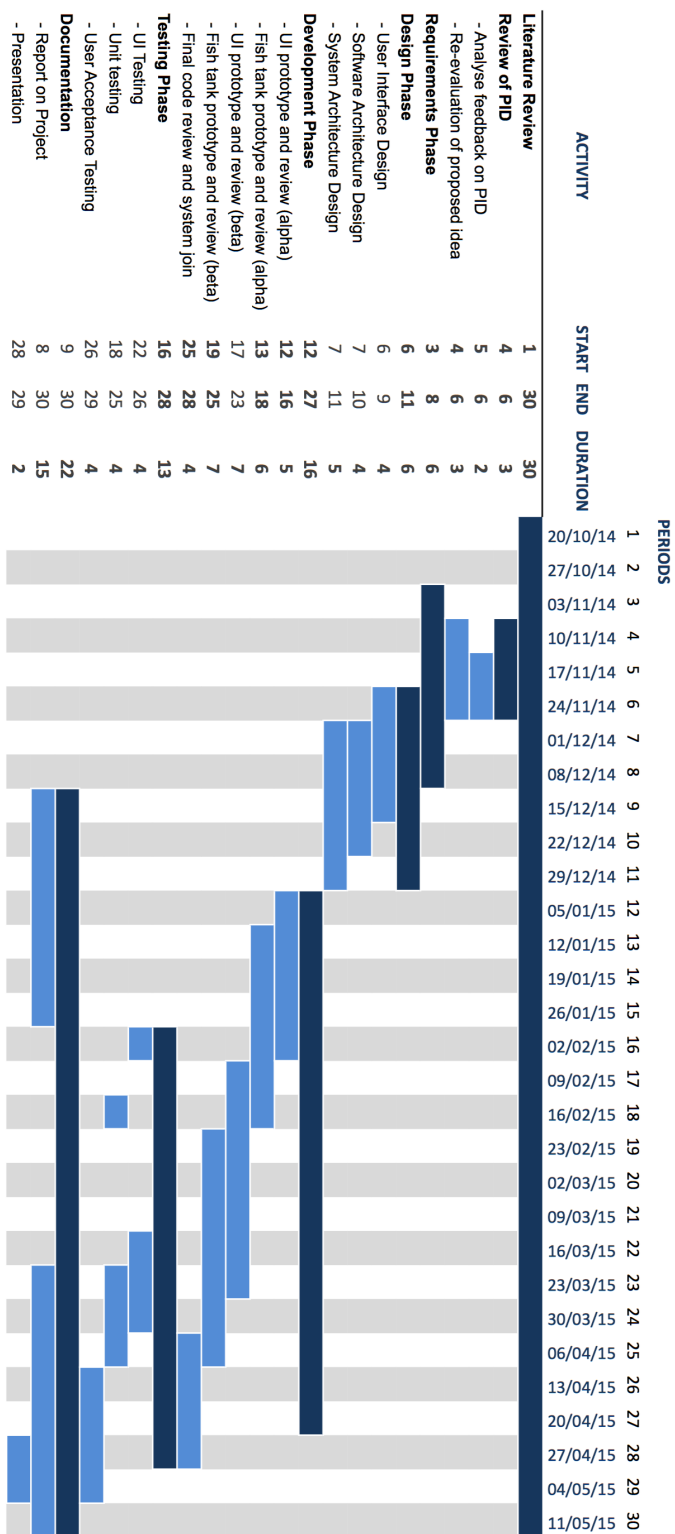
1.4 The Idea

In order to create something new, we steered away from the connected home. We wanted something different, something unique and that took us towards music. So we started off with trying to connect people with music. We want to provide the ability to connect to and control the music in your surroundings, anywhere. We want users to walk in to a coffee shop, library, bar, restaurant etc. and have the ability to suggest their favourite songs that are then played through the public audio media. **Choona** is an app focused around the ‘Connected Music’. Our research has shown there is no music sharing concept that allows people to collaboratively listen, interact and suggest music to people in your surrounding area, home or business through an intelligent, cloud-driven playlist system.

Choona is a public music player that lets you have a say in what you listen to in public, allowing you to suggest songs that you want to be played at your location. It provides you with the option to like/dislike songs suggested by others, where increased likes on a song will push that song further up the order allowing it to be played sooner. Furthermore, you can connect through your mobile device allowing you to listen to the music privately (via your headphones).

Choose provides many different features for the businesses involved. Advertisements can be added in two forms; visual or audio thus improving product/promotional awareness. Furthermore, gives the customer the opportunity of selecting what they want to hear thus keeping them happy.

2 Project Plan



2.1 Gantt Chart

The gantt chart we have provided illustrates how we will aim to complete this project on time. It includes a break up of all the tasks and the proposed start and finish times. We will use this gantt chart to track our progress throughout the project and there may be cases whenever we have had to adjust it due to time constraints and other activities outside the project.

2.2 Team Roles

We have tried to arrange the team according to our strengths and weaknesses. We have tried to consider the different tasks involved in the project and identify the roles that an individual is most likely suited to.

According to Belbin, there are 9 different team roles; three action roles, three people roles and three thought roles. Our team cover the majority of roles between them; we have shapers, implementers, completers, plants specialists, monitor and team-workers. Overall this provides us with problems solvers and passionate specialists that have a wealth of knowledge as well as individuals that have a motivation to complete this task on time and to a high standard. Below, we have defined our roles within the team to best suit our skills and to suit the type of role we are stronger at.

Name	Role	Responsibilities
Jay	Project Secretary, App UI	Organising meetings and following the project timeline. Assisting with development where possible.
Oliver	Project Manager, Technical Lead	Leading the technical aspects of the projects forward. Managing cloud infrastructure for both documentation and code.
Simon	Business Development, App UI	Carrying the app forward according to the business model in terms of legal and technological factors. UI development

2.3 Project Management

Responsibilities will be shared throughout the project as there are only three people in the team. The roles will change throughout the project as it may suit someone else to take on a responsibility at certain stages. There will also be times when experience will become a

factor; we have considered this and tried to align our team roles to match. During different stages of the project, it may be that we have different activities going on outside of the project that may cause slight readjustments to the gantt chart but this will not effect the overall outcome. The stronger programmer(s) in the team will have a better understanding of what is involved at the development stage of the project therefore they will be more inclined to take responsibility and delegate tasks.

Every week, we will meet up to discuss any feedback we have based on our previous weeks work and if any decisions need to be taken based on this, these will be made clear and a decision taken before the meeting finishes. If a decision cannot be taken because of lack of information, we will go away and research the area further and bring the relevant issue(s) up at the next meeting. We will also use these meetings to determine whether or not we are currently on schedule with the project and to make proactive decisions based upon our current rate of progress. During each meeting, we will decide what each members tasks for the next week thus allowing us to allocate the right amount of time to complete them.

2.3.1 Team Communication

The main communication method for all team members will be through a “WhatsApp” group. All member currently use this as a social messaging service therefore we felt it was a good idea to use this as we always have access to it. The main idea of this is to send short messages to each other whether it be the whole group or just one individual. This keeps everyone in the loop. We can effectively use this to organise meetings and communicate our progress and problems. We also have gmail accounts that allow us to send communications to our customer (Mike).

Further to this, we all have access to “Google Drive”. We have used this to create a folder that each of us have access to. Here we can post any documents we want the other team members to view and they can instantly access them and make any changes they want; these changes will be globally updated for all users to see.

Initially a “Trello” discussion board was used to throw ideas around and discuss potential features of the app as well as a broad business model.

We will also be using GitHub. GitHub is a web based repository that offers Source Code Management (SCM). SCM is a necessity in a team orientated software development project. A Version Control System (VCS) allows the developers to make revisions to the code seamlessly and without overwriting someone else changes. It also allows you to revert to previous versions or examine them where need be; to help identify issues with newer version(s). This is a very useful tool to have especially when the project involves such a large code base and multiple developers.

2.4 Resources

In order to complete this project, there will be several different resources we will need to have access to. This can usually be split into three different categories; personnel, funds

and non-personnel. However for this project, the only personnel we have are the three team members and this cannot be expanded on. Funds is also non-negotiable; standing at 0. We do have several non-personnel resources - these are resources that we need in order to produce the prototype and documentation. These are listed below with an explanation as to why these are needed.

- Raspberry Pi - There is the potential for some software to be developed and put on a hardware device in order to connect with the cloud and stream audio (music). For prototyping it is likely that pre-existing devices such as the Raspberry Pi will be used.
- Speakers - To enable us to listen to the output from the Raspberry Pi
- Laptops - these enable us to work as a team in any location at any time and for use in the presentation.
- Television - to present our idea and prototype at the end of the year.

Having the above items will make sure we can complete this project to a good standard and be able to give a presentation at the end.

2.5 Software Development Approach

Software Development Approaches have existed since the 1970s. They range from rigid, long-term methods such as *Waterfall*, through to experimental and risky approaches like *Extreme Programming*. Factors such as team size, business goals and clients all contribute to choosing the correct approach.

To identify the correct software development process to use we must first decide on some attributes of the project of which we can compare across various approaches. The following have been derived from the project ideas and plan:

1. **Team Size** - Most of the development throughout this project will be done by one person. This removes the need for large-scale approaches such as *Waterfall* and *Spiral* and instead opens up the possibility of a faster, looser process such as *Agile*.
2. **Client Interaction** - If the project idea is accepted we will be allowed to work very closely with a representative from Atos. This means our development approach needs to incorporate this throughout the entire lifecycle, making methodologies such as *Waterfall* less relevant since they do not encourage client interaction during the development stage.
3. **Timeline** - The entire development and testing cycle for the project is only a few months, so whatever approach is chosen needs to be dynamic and allow for quick iterations without the need to go through the entire process again. This requirement makes approaches such as *Incremental Development* inappropriate because they add in repeated requirements analysis and architecture design steps that may not be necessary.

4. **Deliverable** - The client is expecting a working prototype to be delivered at the end of the project. The technology must be built to a satisfactory standard, but does not need to be production-ready nor be immaculately written. This allows for the testing, integration and implementation steps in the software development approach to be removed as a priority and opens up the possibility of using less formal approaches such as *Prototyping* and *Extreme Programming*.

These requirements narrow the scope down to three relevant approaches. The first one is **Agile**. Agile development promotes principles such as continuous improvement, early delivery, adaptive planning and evolutionary development. It breaks tasks down into small increments, with the goal of each increment being a new release. There is a big focus on quick communication and easy feedback, normally achieved by daily standup meetings and regular planning sessions. Whilst the communication principles would be useful for this project, the approach also tends to have a reliance on multiple teams tackling the project which is not relevant.

The second approach we could take would be **Prototyping**. Software prototyping is perfect for projects where a production-ready deliverable is not required. Each iteration of the project has no requirement to fully work or meet the users' requirements. Instead, the software goes through an iterative modification process until user acceptance is met or the prototype is discarded. The downfall of prototyping is that it is a very relaxed approach. Client interaction is suggested, not enforced, and often the developers may lose sight of the completed project by getting too involved in a limited prototype which ultimately is only a partial representation of the desired product.

Finally, we have **Rapid Application Development (RAD)**. RAD is used to describe alternatives to the *Waterfall* approach that focus on iterative development rather than formal, structured planning cycles. The most common definition of RAD, created by James Martin, involves an initial requirements planning stage followed by an iterative construction and user feedback stage.

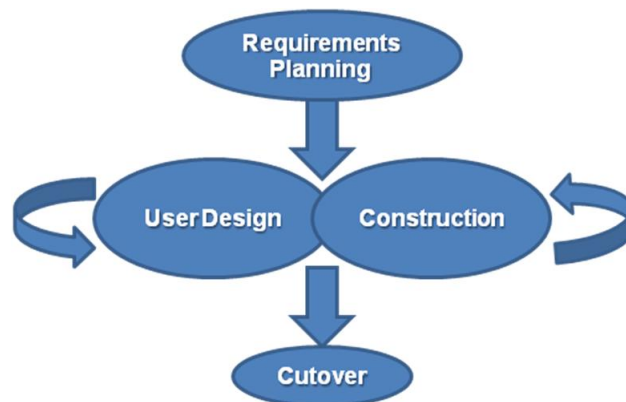


Figure 1: Phases in the James Martin approach to RAD

RAD fits in nicely with the four primary project attributes that we defined above. RAD has no mandatory definition for team size or interactions. It requires that development,

client feedback and testing go hand in hand. For this type of project this is ideal. RAD does not stipulate a restriction on the length of the project thus allowing for the development cycle to be as long as possible before the deadline is reached without the need to stipulate iterations during the planning stages. Finally, there are no requirements for repetitive testing and quality control. This can be implemented independently from the software development approach, allowing for testing to be suited to the individual project.

The analysis above clearly identifies **Rapid Application Development** as the most appropriate software development approach for this project.

2.6 Technical Considerations

The technical considerations for this project can be divided into two sections: mobile application and server-side systems.

2.6.1 Mobile Application

One of the basic top-level requirements for this project is that the end product is centered around a mobile application. Therefore it is important to ensure that all the technology used or idealised is compatible with as many mobile devices as possible. If the app starts to have a dependency on niche features, such as NFC and 4G, there is a risk of excluding large quantities of users.

For a mobile application, we must consider the platform. There are currently three major mobile platforms that support apps: Android, iOS and Windows Phone. Others like BlackBerry and Firefox OS either do not have a thriving application community or have a very small user-base. This project should ideally support all three of the major platforms. The problem with this is that each one requires applications to be written in different languages. In order to come to a decision, we need to consider the following:

- *Native Applications* - A native application is one that is written to work directly with the device's operating system. For example on Android this would mean that the application is written in the programming language Java and that it directly uses the device's native libraries. The advantage this approach is mainly performance, but there are also certain features that are only available for native applications. The disadvantage is that the application has to be written separately for each platform you want to support.
- *Framework Applications* - To solve the difficulties of supporting multiple platforms, frameworks such as Apache Cordova and Adobe Phonegap have been created. These allow developers to write their applications once and have it automatically work across all major platforms. The downside here is that the application normally exists inside a 'wrapper' which can have a negative impact on performance.

The **Technology** behind the mobile device is another area of consideration. Features such as NFC are starting to become prevalent in many high-end smartphones yet there are still

devices that block the use of such technology (such as the iPhone range - the iPhone 6 does have NFC capability but this is only available for apple pay) and there are devices that don't have this capability purely because they are on the lower-end of the market.

2.6.2 Server-Side Systems

Many ideas and concepts in this project revolve around connected data and technology. There are likely to be many different services and systems required in order to complete the project. Designing and architecting will be an important part of ensuring the project is a success.

There are many different styles of **Software Architecture**. For this project we will be using the *Microservices Architecture*. This is centered around separating out the application into individual services, each one with its own unique responsibility. This fits in nicely with the chosen Software Development Approach - *Rapid Application Development* - since it allows for entire services to be completed in a single development cycle.

Advantages

- Easier debugging
- Better fault isolation
- Independent service development and deployment
- Removes commitment to a specific technology stack
- Avoids monolithic applications and systems

Disadvantages

- Requires inter-service communication
- Testing can be more complicated
- Multi-service deployment can be hard to orchestrate
- Multiple services means more memory usage

A solid **Infrastructure** is essential in order to adequately support the numerous software services and systems required for the project. There are several options available when it comes to selecting infrastructure components (servers, database management systems etc):

- **Cloud Computing** - Cloud-based systems such as *Amazon Web Services* (AWS) and *IBM SoftLayer* allow systems engineers to easily configure and deploy scalable services across the globe. They often support many different types of services (application servers, databases, data processors, queues etc) and can be configured quickly and easily through user interfaces, rather than having to manually set up each service.

The downside to cloud-based infrastructure is usually the cost and also the added abstraction; the systems engineer is no longer in direct control of each moving part which can sometimes make fault finding a more laborious process.

- **Dedicated Servers** - Dedicated servers are the more traditional approach to supporting software systems. They give the systems engineer absolute control over the entire infrastructure, however this goes hand-in-hand with the additional responsibility involved in configuring everything by hand. Another concern is redundancy. In a cloud-based system it is very easy to spin up copies of a service in the event of failure, however with a dedicated system you normally need to have servers on standby all the time. Compared to cloud services, dedicated servers can often work out cheaper.

3 Idea Analysis

In November, we put forward the idea of Choona in the PID. Over time, details have changed and in the section we want to analyse the original idea, the components that make it up and discuss whether or not they remain the same and if not, why have they changed.

3.1 The Idea

We briefly mentioned what Choona was in the introduction. Its important that we provide a more detailed explanation of the idea and the system behind it. In order to do this, we will break each section of the project down and briefly explain it.

Choona is the name of our crowd controlled juke box. It allows users with a mobile device to connect to a Choona location and decide what music they want to listen to. **Login** - the user will be asked to sign into Choona. There will different options available to the user; they can log in via a username and password or they can login via a social media account (Facebook, Google+, Twitter etc.). In essence, this opens up the option for more users to connect quickly and easily with Choona and the opportunity to post on their social media account that they are using this app.

The main reason behind this app is to play the music you want. Therefore in order to do this, we need a playlist. This playlist will have several pieces of functionality. First of all, this will contain the list of songs that have been currently added to the playlist. The user will be able to scroll this page and look at the different songs. They can then **up-vote/down-vote them**. The idea of the up-vote/down-vote is to push songs further up the playlist so they are played quicker. Lets consider the example; there is a song that I really like and many other users within the facility also like that song. If we all decide to up-vote that song, then it will move further up the list because it will have received more votes than other songs on the playlist. The same is true for the other way; if we down-vote the song, it will move further down the playlist. This page will allow the user to add songs to the playlist. There will be **search** functionality; this explores any connected music source(s) and looks for the song name, artist or album depending on the users input. Any results will be displayed and the user can add the song they want. As this is a public system, there will need to be considerations taken on the available music. There may be occasions when we do not want explicit versions of songs being played (especially when there is a risk of children being present - nightclubs may wish to allow for explicit versions as their customers will be over the age of 18). Depending on the business and their choice; this will be reflected in the search results and the songs played.

In order to stop abuse of the system, there will be a limit on the number of songs a user can add to a playlist at any one time. There are different options available for this at but we are leaning towards a time-limited based approach. This would mean that a user can only add e.g. one song every fifteen minutes. It has been suggested that we add a points system to this facility that allows you to gain points through the number of up-votes any of your previously added songs receive. Once you get over a certain amount of points, the number

of songs you can listen to is bumped up. There may be instances when no songs have been added to the playlist by users. If this is the case, there will be a **default playlist** available that kicks in when the user playlist becomes empty. This means that there won't be any stage when there is no music being played. This default playlist is created and maintained by the business through their Choona account. Now we have the playlist, how do we connect to it?

Different locations will have different playlists. In order to differentiate between these distinct locations, we will make use of **Geolocation**. There are two sides to this; the business will have to create a boundary - this would be their shop floor area. We call this a *Geofence* and it acts like a virtual barrier. When inside the barrier, the user will have access/connection but when they leave, the user loses their access/connection. The second part to this is the mobile device. Using the locations services available on mobile devices, we can then determine whether they are inside the geofence. This therefore means we can allow customers to connect to that location's playlist where they can then add their own song choices and up-vote/down-vote other song choices on the playlist. Using this functionality, we can also make sure the state of the playlist is ideal for the current users. If a user leaves the geofence; any song(s) they have added to the list can be removed if that song(s) has not received any up-votes from any other users.

As we mentioned earlier, Choona allows for social media login but we take this one step further. We want to use social media as a way of socialising with friend. We have decided to use a notification system where users can add posts from Choona onto their social media account. A post will contain location (through geolocation), the song currently being played and a timestamp; something like "Chris is listening to Happy by Pharrell Williams at Loughborough Students Union - 10 minutes ago". In the app itself, any notifications posted by friends that are Choona users will be displayed. This is a small simple but fun aspect to the app and has been introduced because teenagers and young adults in today's society are heavy social media users with a large social influence.

Within the app, there will be a **history** page. Within this history page, there will be a list of different Choona locations that you have connected to. Behind each of these locations, there will be a list of songs that were played from when you entered that geofence to when you left the geofence. With this functionality, we are trying to provide the user with the ability to check back and find a song that they really liked but do not know the name of. It is a regular occurrence they we listen to a song but do not know what it is. With background music, it is hard to use an app like "Shazam" or "Soundhound" to identify the song for two reasons; the volume is too low or there is too much noise to identify the song. Therefore the history feature on Choona can allow the user to find out what that song is. Further to this, there will be functionality to **purchase or play** that song. Choona will look at the different music providers on the mobile device and then offer the option for the user to connect to that provider and play/buy the song e.g. if running Spotify, the app will offer

the option to “Play in Spotify” thus allowing the user to add that song to their own private music collection.

There may be environments when you want to listen to the music but there is too much background noise. This can be common for customers in coffee shops who are there to work or for people in an office environment. Choona will enable the user to listen to the playlist privately; through the use of headphones. Very simple and could be very effective.

There would be two different types of Choona accounts; standard user and admin. The customer will sign up for the standard user account, providing them with the functionality described above. The admin account is the account provided to the business. This is used for the purposes of identifying the music sources. The admin account will link up with the different available music sources to that business. This is linked to the search functionality in the customer app so the search results that appear for the customer relate to the music available from the connected source. Furthermore, the admin account will deal with adverts. The Choona app allows for advertising in two forms. The first within the app; at the top of the playlist, there will be an accordion. Within this accordion, the admin can either have one single image; depicting the business or they could utilise it for the purposes of adverts. They can place multiple different images that scroll through with each one depicting whatever they want; a new product or a special offer. The second form of advertising comes in sound bites. These sound bites would be positioned between songs (this is handled automatically by the Choona system). The admin account will just have to add the sound bites they want to placed between the songs.

3.2 User Value Proposition

There are four small sections that curtail a value proposition. These have been outlined below:

- (1) Headline - one short sentence about the end-benefit we are offering
- (2) Small paragraph - explanation of what we offer, for whom and why its useful
- (3) Key benefits and features of the app
- (4) One visual - pictures paint a thousand words

Based on the criteria above, we have written the follow proposition.

(1) Choona is a music-sharing app allowing users to collaboratively listen, interact and suggest music in public areas, businesses or homes through an intelligent, cloud-driven system.

(2) This app is for anyone who wants a say in the music they listen to; users can access music playlists through geolocation and add the songs (from different sources) they want.

Songs move further up the playlist the more “up-votes” they receive or further down the queue the more “down-votes” they receive. Choona offers social media interaction and provides a catalog of history to help find the song(s) that’s name you can’t remember.

(3) There are several key benefits to Choona for the user:

- Listen to your preferred music - In too many situations, we are forced to listen to music that doesn’t interest us. Though the majority of times the music is in the background, we somehow know it’s there. If we enjoy that music or can relate to that music in any way, it has a positive effect on our attitude and what we see.
- You do not need to sign up; you can log in using a social media account.
- Adverts can help highlight a new range of products or identify any special offers that you may be interested in.
- Helps identify songs that you cannot remember the name of or the artist that sings it.

(4) This image is a...

4 Literature Review

4.1 User Interface Design

Having read different articles relating to Human Computer Interaction components, several factors have been identified that directly link to improving interfaces of mobile applications. Some of these are the use of graphics, colour, font and other effects.

Colour is one of the main attributes and it covers many different areas including background, text colour and visual effects. Lets first consider background and text colour. There is a lot of different opinions on this topic; a lot of people will lean towards a light background with dark text. Dark backgrounds (dark designs) are becoming very popular now and add a creative and elegant appeal to the app. This is not something that can be left to preference. There are situations when dark backgrounds suit the app and there are situations when a light background is preferred. An app with lots of reading is better off having a light background with dark text. A recent survey was taken on this area and 47% prefer light background because it aids with *readability*. Another 10% said they prefer dark backgrounds with 36% saying it just depends on the function of the app [0]. With our app in mind, we are not so hung up on readability but eye fatigue. We want the user to be able to use the app in any conditions whether it be during the day, night, in unilluminated rooms (bars and nightclubs) or illuminated rooms. With this criteria in mind, we need to consider something that has not too high-contrast. We do not want a full white on full black or vice versa. We can use dark grey with off white and this will prevent the eyes burning out. One final note to make; darker backgrounds tend to use less battery. A test was carried out on an AMOLED screen and the results showed that ‘mostly’ white background use 1/3 more battery than black backgrounds [1]. Another test was carried out on a Nokia Lumia 720 (with WVGA IPS screen) and this used 6.37% more battery having a white background compared to dark background [2].

Now if we have the dark grey background, we need to consider other colours to use with the app. Blue is usually a popular colour of choice (the world’s favourite) [3]. Blue is the colour of the intellect and the mind. However, blue can be difficult to see on certain backgrounds as it tends to blend in. This is actually to do with our eyes. There are fewer photoreceptors that react to blue compared to other colours and they are not in the centre of the retina because it is sometimes hard to distinguish.

Lets consider another colour; orange is included in Ubuntu’s colour palette as they believe it signifies a community feeling. There are many other descriptions of orange - it is classified as a warm colour, it radiates warmth and is often associated with energy, happiness, attraction, stimulation and comfort [4]. We want our app to have a community like feeling. This is an app for the public where they can talk through the language of music. Two other interesting words here are *attraction* and *stimulation*. Ultimately we want users to be attracted to our app, enjoy their experience and to keep using it. Although colour won’t have too big an emphasis on this, a well designed app can and colour plays a part in this. Therefore orange seems to be a good colour choice. Colour can also used to convey information through visual recognition. A colour like green means on, safe, valid etc while red means off, danger, invalid

etc. We can make use of these colours in our app to provide information to the user. Talking of colour, one of the most important thing to remember is to not use too many different colours. We want to keep the colour scheme minimal. A busy colour scheme will obscure the dark background as the contrast will be too sharp. Therefore we shall stick with 2 different colours and a background colour.

Font is another important consideration. It is important to make sure the selected font is clean, crisp and works well with colours we have chosen. There are several areas to consider here; the first being weights. If we want to try and create contrast and a visual hierarchy within the app, we can consider different weights such as light, normal, italics, bold and extra bold. Legibility is also important due to the number of small screen devices there are currently on the market. If we use a font that is hard to read when it is smaller, then our app design has a serious flaw. Therefore it is important to use a *sans serif font*. This type of font does work well with a dark background. The trick, though, is to put only larger text in serif fonts, so that the extra white space floods around each character and makes the text very legible.

There are several other key considerations. Consistency; the app should be as consistent as possible with commands and menus containing the same content and format. If there are buttons or features designed on one part of the app that act in a certain way, then if this button/feature appears somewhere else, it should act the same way. The app must be navigable in the sense that we can manoeuvre between the different pages. Any messages that are displayed (toast notifications etc) should be descriptive and helpful. Simplicity is also very important and ensures the user will keep coming back to the app. Many useful apps have been created over the years yet people do not come back because they find it awkward to use or they find it complicated to use. An example of this is Bump. When this was first released, it had features to transfer music, photos, contact information other documents and recommended apps. But due to its complicated nature, it was slated and users abandoned it. When the creators re-developed the app with simpler page layouts, the app became easy to navigate and simple to understand. That is when its potential was realised and now has been downloaded millions of times.

4.2 App frameworks

There are numerous different frameworks available for us to use in order to create the client side app that users will interact with.

First of all, there is **PhoneGap**. PhoneGap allows users to develop hybrid mobile device applications using HTML, CSS and JavaScript. A hybrid application is one where it is neither a native mobile app (as layout rendering is done via web views instead of a platform native UI framework) nor a pure web-based app (because they are packaged as apps for distribution and have access to native device APIs). PhoneGap works across multiple platforms (iOS, Android, Windows etc.). However, any app created using this can be slightly unresponsive compared to fully native ones. There has also been reports that bugs can pop up on specific devices, something we don't want whenever we consider the UK operating system split is

49.7% android and 42.5% iOS with the remaining percentage covering Windows, Blackberry and others [4].

Two other frameworks we could use would be **Eclipse** and textbfX-code. Eclipse creates apps that look and feel professional with the help of the components contained in the SDK. However, this only produced android apps, thus leaving out all iOS users. Eclipse is also Java and not JavaScript therefore the skills required are slightly different. X-code has an amazing UI editor alongside other different development tools that make it easy to create effective and well connected interfaces. The environment also allows for easy testing and easy release of the app. However, this produces only iOS app's therefore we are eliminating the android section of the market as well as the others. X-code is also only available on Mac, therefore we cannot develop it using windows or Linux. Finally, X-code doesn't use Java or JavaScript, but Objective-C and at the moment, none of us have developed with this.

Ionic is one of the newer frameworks for app development. Essentially, Ionic is a wrapper around the Cordova framework that comes with a bunch of very powerful CLI (command-line interface) tools - suits some of us more than others. It has been built on top of AngularJS (from Google) where angular provides an application structure with Ionic providing the User Interface. The two are harmonious with Ionic providing angular directives for its own components. This means that certain features can be created with very simple HTML code. Ionic also makes use of Bower and NPM and having all these popular tools and frameworks surrounding it, makes it a popular choice. The creators of Ionic said they have focused on performance. In doing so, they have brought about simplicity - it keeps a flat, clean simple and powerful UI without unnecessary rendering of rounded corners etc. This fits in with our idea of the app; we want something simple and clean as this will result in a better user experience which will entice the user to use it again.

4.3 Effects of Music

Music comes in very different forms and covers many different genres. This leaves it very hard to play music that suits most or all kinds of customers. According to "Which", its not actually the music we like that grabs our attention but its the music we don't like that we notice most. Therefore, we don't want people to be listening to music they don't like for one main reason; it will annoy them and potentially make them leave the facility.

Music has a lot of power; it moves people of all cultures. Unfortunately, it is not understood as to why listening to music triggers such a rewarding experience but through brain scans, it seems songs trigger the same brain flooding (with dopamine) as food and sex. This reaction then causes the Nucleus Accumbens to communicate with the temporal gyrus [7]. This essentially causes us to register memories; we have a likeness to remember the fond memories with links of music and sounds. Having this knowledge, we can essentially use this to stimulate customers; happy customers drawing on happy memories will result in many different activities and these should be mainly positive ones for the shop, whether it be more purchases or decisive purchases.

How music is played also has an effect on our activities within a shop. There have been several different studies carried out over the years linked to how volume, speed and type of music effects our behaviour. For instance, the louder the music, the more likely it is that people will spend less time in that shop [8]. There are cases when this is a good thing; both for the customer and for the business. Lets consider a supermarket. Before we make our way to the supermarket, we have a fairly good idea of what we want. Therefore we are not actually going to spend any less money if we are fast and efficient compared with spending more time in the store. Its good for the customer because we will have more time to spend on other activities and its good for the business because their stores do not become bunged and they should in essence have a steadier stream of customers.

Slower music will result in customers spending more time in store and hopefully an increase in purchases [8]. This type of scenario is what businesses want when customers come for a look around e.g. cloths, furniture, jewellery and coffee shops etc. The customer may not have had the intention of making any purchases but the slow music changed their brain activity that resulted in a purchase they didn't intend to make. Even in coffee shops, it may result in the customer(s) grabbing another coffee before leaving.

Listening to the wrong type of music can make people believe they have been somewhere longer than they have. Something like this will force the customer to leave the store, maybe even before they have purchased anything. Therefore having the right type of music playing is essential to keep the customers in the store long enough to make sure they makes purchase. There is no better way of doing this than allowing the customer to have a say in the music they listen to.

4.4 Competitors

We have taken a look at other apps and services on the market and feel that nothing matches our idea. The closest service available is from **Sonos**. Sonos is a smart system of speakers and audio components that unite your digital music collection in one app and can then be controlled from any device. The rise of digital music has allowed for us to bring our music wherever we go, through media such as iPods, MP3 players etc. However, there hasn't been the same advancements in terms of systems that don't move around. 'Wireless' is the word that comes to mind when thinking of a solution. It is now possible to stream audio to a wireless device (speaker) and without compromising on the sound quality. Sonos provides the user with the ability to play music from a device wirelessly anywhere in the **home**. However, there are two issues with this; it is only available for the **home** and the consumer needs to purchase expensive **hardware**. The cheapest speaker available for purchase is £169. If you are a music-orientated person, you may wish to purchase their high-end hardware which can cost up to £1200. Unfortunately, Sonos do not support other wireless speakers. An adaptor can be purchased to allow these to be connected to their system, the *Connect* device, but this costs £279.

Pure have also moved into this market where they provide wireless speakers and hardware for *wireless music* in the **home**. They also allow you to purchase hardware to link your current speakers with their system at £69. This is much cheaper than Sonos but is still

quite expensive. It allows the consumer to wirelessly play their music from any music app or streaming service they want. **Bose** also provide a very similar service to Pure, but the hardware costs are more expensive.

From this, we can see that there are no services available for the wireless sharing of music outside the home. Our app would unite music into anybody's daily routine, whether this is at the office, coffee shop, restaurant as well as the home. We also want to make sure that no expensive costs are applied. Competitors can appear at anytime during the development of a project, so it is important that we keep looking for emerging competitors and that we can identify how our product is unique to theirs.

4.5 Music Sources

In today's market, there are many music sources available to an individual. We have virtual music from services such as 'Spotify', 'Google Music' and 'iTunes' as well as physical music on 'iPods', 'MP3 players' and other hardware devices.

Spotify is a music streaming service that offers access to a library of over 20 million music tracks with over 40 million active users. It is available across 58 markets including the UK, USA, France, Germany, Hong Kong and Argentina. It is available on iOS, Android, Windows phone as well as PC and Mac. One chain that is affiliated with Spotify is Costa. Costa have their own playlist that people can access from their device.

Google Music is another streaming service and offers the same service as Spotify. Again, they have a large library of songs (around 18 million) and the service is available in over 57 countries on all Android devices as well as web browsers.

Auracle Music is a custom streaming service that delivers the best background music through the internet. This allows you to tailor the music you have playing to suit your needs. At the moment, this system contains over 30 different licensed music channels with a whole host of music making it compatible with this project. The top advantage of this is actually the fact that it is for commercial use. This therefore means we can use it alongside Choona and not require any further licenses.

A year ago, Apple's **iTunes** accounted for 75% of the digital music market and with a huge 575 million active users. Although this may have decreased slightly in the last year, that is still a large user base. As well as general users, Starbucks is affiliated with iTunes and use this service to hand-pick and play music throughout their stores. The idea of allowing customers to put forward their music preference may be of interest to a chain like Starbucks amongst others. The issue with iTunes is that all the music has to be purchased before it can be listened to. There is no monthly subscription fee where you can listen to as much or as little music as possible. Apple are however bringing out their own music streaming service; called **Beats Music**. The above figures suggest that the music streaming industry is vast and that music is a part of many people's lives. Coffee shops have integrated these sources and music into their environment, but without the customer interaction. Choona would provide this interaction. Over the course of this project, we shall identify more sources because having more sources creates a larger user base as well as a better music library. We

shall look at how the different sources work to try and make sure we have adaptors in place that can cover the wide variety of sources available.

4.6 Legal Issues

With this type of app, there are certain laws and legislation in place that have to be adhered to in order to play music in public.

4.6.1 Public Performance License

Music playing for customers or staff through media such as radio, MP3, TV etc. is considered a *public performance*. The *Copyright, Designs and Patents Act 1988* means that an agreement is needed from the copyright owner before the material can be played in public. A music license (PPL) will grant this agreement. In most cases, a license is required but there are a few instances when one is not required. One example of this is where PRS artists have waived their rights. PRS for Music represents the rights of over 100,000 artists in the UK. It provides licensing to organisations to allow the playing, performing and availability of copyright music on behalf of the artists and overseas societies. The royalties are distributed fairly and efficiently. Another example is a hotel, guest house or B&B that has fewer than 25 rooms with no areas open to non-residents. Any business such as a coffee shop, bar or gym that plays recorded music in public will legally require a PPL. The likelihood of our service being used in places that don't have a PPL and require one is small. Most coffee shops, restaurants, gyms etc. will already have the license in place. It will be work places deciding to implement our service that will have to go about retrieving a PPL.

The costs vary depending on the facility and how the music is used. Cafe's, restaurants, pubs and bars are charged based on the area size, so the smaller the area, the smaller the fee. This is an annual fee and ranges from between around 130 to 325 per year [6]. The reason the fee is not too large is because the music is just background noise and therefore it is not the main attraction. In some establishments, they may not play background music and so if they wanted Choona, they will have to apply for the license. Ultimately, it will be their decision as to whether they decide to take on this extra annual fee; maybe Choona can promote them enough to gain extra sales thus warranting its purchase.

Retail shops too have to pay a fee and again this is based upon the area size of the store. These range anywhere from around 130 to 220. A lot of retailer space does not have music in the background and so they will need to judge whether or not they want to take on Choona and the required licenses they would need. It is all about whether or not Choona as a service will provide enough benefits over the costs needed. If the store already has a license in agreement, then the integration of Choona should be simple.

Nightclubs have a different payment scheme; they have a standard fee to pay and on top of that, they pay an additional cost (for each night of entertainment) that is calculated depending on the number of customers (population size) and the number of hours the music will be played [6]. Therefore fees can escalate enormously over a year long period. This though however would not make any difference for them; they will have to pay these fees

regardless of whether Choona is integrated or not therefore integration should be simple and straightforward. There are tables highlighting these costs for the different premises and these are available in the Appendix.

4.6.2 Entertainment Licensing

Introduced on 6 April 2015, this licence may need to be acquired by businesses, organisations and even individuals who want to provide entertainment. This entertainment can cover more than music but for this project, we only want to consider the instances when the entertainment revolves around the playing of music or the performance of music. This license will apply to places such as night clubs, live music venues (concerts) and large indoor arena's.

Below are the set of conditions where if any are met, a license is required:

- Entertainment is provided between 11pm and 8am
- Amplified live/recorded music performed to an audience greater than 500 people
- Recorded music played to an audience on premises where the sale of alcohol is not licensed.

Essentially this covers bars, nightclubs, concerts and other different use cases for Choona. The fees have been laid out in figure 2.

License	Temporary	1 year	2 Years	3 years
Commercial operation with capacity < 5000 people	£597	£597	£1194	£1792
Commercial operation with capacity > 5000 but < 10000 people	£3810	£1238	£2484	£3810
Commercial operation with capacity > 10000 people	£7520	£2513	£5007	£7520
Commercial event held in designated stadium	£597	£597	£1194	£1792
Other commercial events (including festivals)	£248	£248	£447	£597

Figure 2: The pricing scheme [5]

The prices vary somewhat and for the larger events and locations, these prices may not be too much of a concern. However, when it comes to smaller venues, they may be reluctant to pay the fee. The process itself in order to get a license, an application is to be submitted. The application can be considered for a period of 6 months before a decision needs to be made. This is not a short process and therefore if Choona is to be implemented in certain establishments, the appropriate amount of notice needs to be given so the license can be granted.

4.6.3 Music

At this current moment in time, Spotify is for personal use only. Therefore we cannot make use of Spotify as a commercial source with it stating in their terms and conditions that “Anywhere you need a license to play music, you are not allowed to use Spotify”. Obviously this is a big issue and something that has to be considered deeply. There are some small services out there that do allow for commercial use of music streaming (something like Auricle Music) but these are not on the same level as Spotify, Google Music etc.

Beats Music is also for personal/private use only. There is slight leeway here because unlike Spotify, they allow for commercial use if the user is a curator. In that case, a curator is someone who has a customised profile page that contains authentic postings and curated playlists. This will have to be verified with Beats Music. This could actually allow Choona to create this type of profile and eventually make use of this streaming service as the source.

4.7 Geolocation

Geolocation is a technology solution used to identify the real-world geographic location of an object. Geolocation makes it possible, from a device connected to the Internet, to obtain various types of information in real time and locate it on the map with high accuracy at a given point in time. Many different methods can be used to collect this data but for the purposes of this app, it will be through mobile phones (users device containing the app) and IP addresses (for the Choona service end).

The idea behind geolocation within this project would be to connect the user to the Choona system within their location automatically or show the different Choona locations within their geofence so they can choose which one they want to connect to. In doing so, we are highlighting Choona locations that are within range only thus limiting the search area and making life simpler for the user. There are several advantages to Geolocation. First of all, we can have **targeted adverts**. For both the customer and the business running Choona, more locally-targeted adverts can create a better app experience. Customers will get adverts related to their location and not just any old adverts thus the experience will feel less spammy. For the businesses, they can use this space to advertise new products, special offers in order to improve sales and custom.

Secondly, Geolocation can help improve user profiles. We can understand exactly where a user is in terms of activity and thus we use this to promote new app features or create different features based on user preferences. Things to consider here would be algorithms that automatically indicate songs you can play based on previous songs you have added to playlists. We can also use this to send push notifications; the sending of messages that could be linked with Choona in general or a specific Choona location. It is reasonably simple to implement the basic functionality of Geolocation on the app side with use HTML and JavaScript needed but to get it working with the backend will require considerable work and time to make it effective and consistent.

4.8 NFC

Near-Field Communication (NFC) is a form of short range wireless communication (4cm or less to launch a connection) allowing for radio communication and the passing of data packets between two devices, or smart tags that work with NFC. NFC is an advancement to RFID systems because it allows for two-way communication. This two-way communication can then be used for authentication purposes or data exchange.

At the moment, not all devices support NFC and it is suggested around 20% of phones worldwide will have NFC capability by the end of 2014. This figure is set to increase dramatically over the next five years meaning more devices will have the capability and more users will be familiar with the technology. This widespread reach of NFC phones could mean one day that NFC tags become as common as bar codes, so it makes sense to make use of this technology.

Using NFC tags with a mobile device, a user could access the playlist at their current location without the need to search for it. The idea is very durable as NFC tags are small and cheap enough to integrate anywhere. They do not need a power source but instead draw power from the device that reads them.

4.8.1 Viability

With NFC, it is unfortunate that apple devices cannot use it. Some do have NFC functionality but this is just for the purpose of ‘Apple Pay’ and nothing else. Almost all android devices currently have NFC functionality that can be used by the apps themselves. NFC tags or stickers are extremely cheap; around 1 per tag. In bulk purchases, this will decrease dramatically. They are extremely easy to encode through a NFC enabled device and allows for protection to stop it being changed by unauthorised user. If we were to adapt the use of NFC, the business themselves should be able to encode the tags themselves. It may be an idea to have a tutorial available where the business just needs to follow the small number of steps involved.

5 Requirements

Global

Requirement:	REQ.1
Type:	Functional
Description:	The user must be able to connect to the geolocation configured by the client.
Rationale:	When the system is initially set up, the client (coffee shop, office etc) will set up a geolocation and geofence for their customers using choona. The user will then be able to connect to this geo-area via their phone and then be able to suggest songs for the public in that geofence. This must be paired with REQ.2 for the user to get full access to the app.
Dependencies:	REQ.2
MoSCoW Rating:	Must

Requirement:	REQ.2
Type:	Functional
Description:	The user must be able to log on using social networks/by email.
Rationale:	An account is required for the user to suggest songs. This must be paired with REQ.1 for the user to get full access to the app.
Dependencies:	REQ.1
MoSCoW Rating:	Must

Requirement:	REQ.3
Type:	Functional
Description:	The client must be able to configure their choona configuration for their customers through a management system.
Rationale:	<p>The client must be able to configure different music options for their business and their geofence. This would include things like:</p> <ul style="list-style-type: none"> • Subscription options. • Genre restrictions. • Be able to configure any ad services for the location; including sound bytes and carousel images. • Default playlists if no songs are suggested. • Geofence/Geolocation options. • Override the queue if needed.
Dependencies:	REQ.1
MoSCoW Rating:	Must

Playlist Page

Requirement:	REQ.4
Type:	Functional
Description:	The user must be able to suggest a song using search.
Rationale:	For the dynamic playlist to work, the user must be able suggest a song they want to be played in public. This song will then be added to the dynamic playlist ordered by the number of votes. If the song already belongs on the playlist then it will not be added again.
Dependencies:	REQ.1, REQ.2
MoSCoW Rating:	Must

Requirement:	REQ.5
Type:	Functional
Description:	The user must be able to see all the songs suggested by others ordered by a intelligent algorithm for the geolocation they are connected to.
Rationale:	The user should be able to see what songs are suggests so they can vote for songs if they wish, these songs must be sorted by an intelligent algorithm where the amount of votes will be a big factor with elements to make the order of songs fairer (eliminating up voting too much, down voting too much, ghost voting).
Dependencies:	REQ.1, REQ.2
MoSCoW Rating:	Must

Requirement:	REQ.6
Type:	Functional
Description:	The user must be able to vote on the songs suggested by others for the geolocation they are connected to.
Rationale:	Songs that are already on the playlist can be upvoted/downvoted by the user. The dynamic playlist is dependent upon this where song are ordered by a amount of votes.
Dependencies:	REQ.1, REQ.2
MoSCoW Rating:	Must

Requirement:	REQ.7
Type:	Functional
Description:	For each song, the user must be able to see the album cover, song title and album title.
Rationale:	This is needed for the user to identify the song, the album cover adds a visual factor to all the songs.
Dependencies:	REQ.1, REQ.2
MoSCoW Rating:	Must

Requirement:	REQ.8
Type:	Functional
Description:	The user must be able to see what song is currently playing in the geolocation they are connected to.
Rationale:	This is to inform the user what is being played currently. Information such as album art, song title, album title and time elapsed will be shown.
Dependencies:	REQ.1, REQ.2
MoSCoW Rating:	Must

Requirement:	REQ.9
Type:	Functional
Description:	The user should be able to share the song currently playing.
Rationale:	To add a social side to the application, the user should be able to share what song they are listening to and where on various social networks. Their friends will be able to see this shared information not only on the social networks itself but also on their activity page. They should also be able to add a message to their activity.
Dependencies:	REQ.1, REQ.2, REQ.8
MoSCoW Rating:	Should

Requirement:	REQ.10
Type:	Functional
Description:	The user should be able to hear the currently playing song through their headphones in the geolocation they are connected to.
Rationale:	This is for the convenience to the user, they should be able to listen to the music privately if they wish. In some use cases this would be ideal such as in an office environment.
Dependencies:	REQ.1, REQ.2
MoSCoW Rating:	Should

Requirement:	REQ.11
Type:	Functional
Description:	The user should have the option to buy the currently playing song.
Rationale:	If the user likes the song, they should have the option to buy it through various services such as google play, itunes etc. This would also be another technique to monetize the app by promoting the artist and their song.
Dependencies:	REQ.1, REQ.2, REQ.8
MoSCoW Rating:	Should

Activity Page

Requirement:	REQ.12
Type:	Functional
Description:	The user should be able to see a list of shared songs.
Rationale:	To add a social side to the application, the user should be able to share what song they are listening to and where on various social networks. Their friends will be able to see this shared information not only on the social networks itself but also on their activity page.
Dependencies:	REQ.1, REQ.2, REQ.9
MoSCoW Rating:	Must

Requirement:	REQ.13
Type:	Functional
Description:	For each activity the user should be able to see the name of person, the song name and the location the person is listening at together with any additional message and a time stamp.
Rationale:	This information is required for the user to identify each activity and so they can differentiate between their friends.
Dependencies:	REQ.1, REQ.2, REQ.9
MoSCoW Rating:	Must

History Page

Requirement:	REQ.14
Type:	Functional
Description:	The user should be able to see the places they visited and the songs that were playing in the timeframe they were there.
Rationale:	In the likelihood of the user wondering what song was playing at a certain place they visited and not being able to remember the name, they can use the history page to look up that song.
Dependencies:	REQ.1, REQ.2, REQ.16
MoSCoW Rating:	Must

Requirement:	REQ.15
Type:	Non-Functional
Description:	The user should only be able to see history for the last week.
Rationale:	To minimize the data overload on storing all history information for all users. The app will only display history in the last week.
Dependencies:	REQ.1, REQ.2, REQ.16
MoSCoW Rating:	Must

Settings Page

Requirement:	REQ.16
Type:	Functional
Description:	The user must be able to toggle whether their history will be logged or not.
Rationale:	If the user wishes not to log all the songs played at the places they visited, they should be able to turn this feature off. This would be in the settings page.
Dependencies:	REQ.1, REQ.2
MoSCoW Rating:	Must

Requirement:	REQ.17
Type:	Functional
Description:	The user must be able to log out from their choona account.
Rationale:	There should be a button for the user to log out of their choona account if they wish to maybe log in with an alternative account.
Dependencies:	REQ.1, REQ.2
MoSCoW Rating:	Must

6 Design

6.1 System Architecture

As discussed during the planning stages, Choona will be built using the microservices architecture. Figure 3 shows the proposed system design of the Choona, split into four layers.

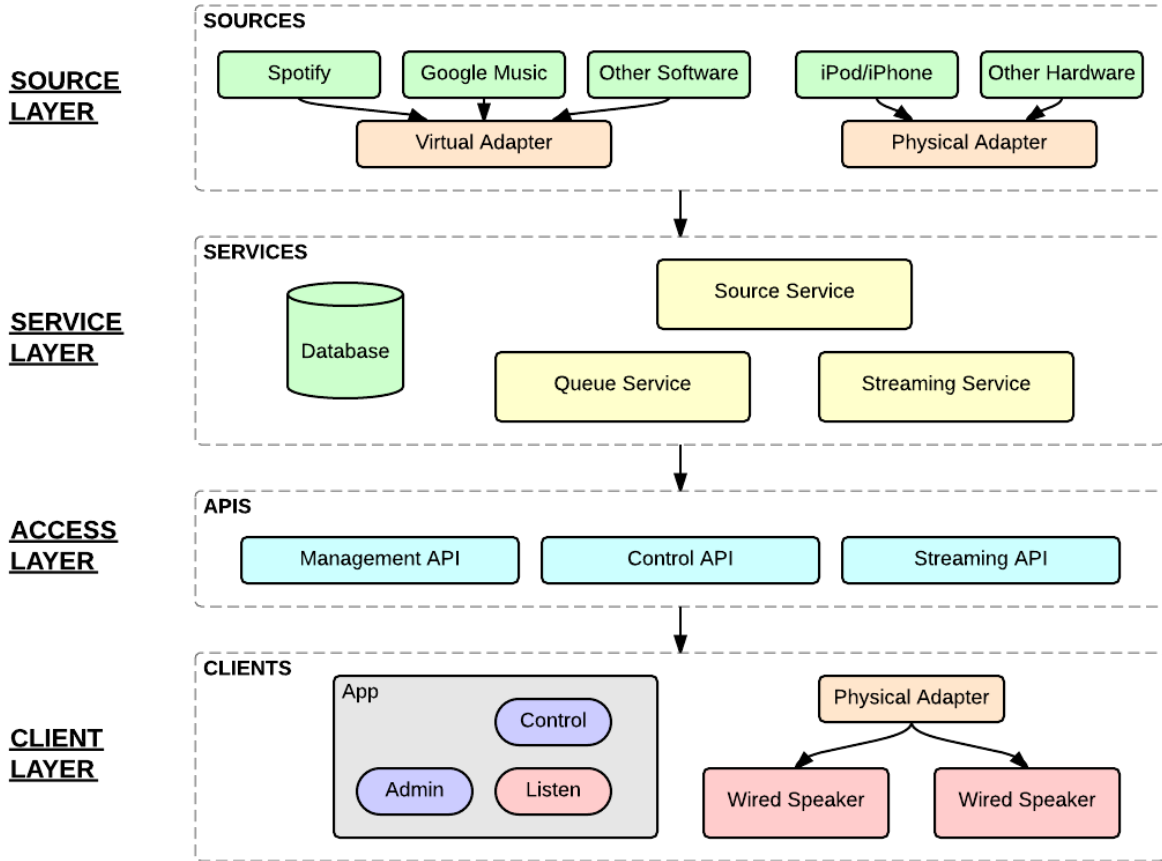


Figure 3: Proposed Choona system architecture

- **Source Layer**

Sources, such as Spotify and Google Music, all have their own unique APIs. For them to be consumed by Choona they need to present a common interface otherwise other services will need to know how to interact directly with each individual API. Therefore each source will be exposed through either a virtual or physical adapter. Each adapter understands the same set of commands and outputs data in the same format. This means that other services in Choona only need to implement the standard adapter interface in order to communicate with any type of audio source. For the first version of Choona the sources will only need to understand a basic set of instructions:

- `search [searchString]` - search the source's track database using the supplied search string
- `get [trackId]` - retrieve track information from the source's database for the specified track ID
- `play [trackId]` - stream the audio for the specified track ID

- **Service Layer**

The service layer contains the majority of the *business logic* of the Choona server-side infrastructure. There will be three main services at this layer. The queue service is responsible for maintaining the running playlist of tracks for a particular Choona location. The source service acts as an interface to all the different types of audio sources (instructions documented above). The streaming service takes tracks from the queue service, retrieves the audio via source service and sequences it into a single stream that the APIs can then send to clients. This layer has a more extensive set of instructions:

- `queue join [locationId] [clientId]` - make a client join a particular location
- `queue add [locationId] [trackId]` - add a song to the queue of a particular location
- `queue upvote [locationId] [trackId]` - upvote a song at a particular location
- `queue downvote [locationId] [trackId]` - upvote a song at a particular location
- `stream add [locationId] [trackId]` - instruct the streaming service to play a particular track next at a certain location
- `stream play [locationId]` - broadcast the stream to all listening clients of a location

- **Access Layer**

The access layer contains APIs that clients can use to interact with the Choona infrastructure. These APIs act as proxies, taking requests from clients and passing them through to the right services in the service layer. The only special instruction the access layer makes available is user authentication verification.

- **Client Layer**

The client layer contains all the consumers of Choona, such as the mobile app and speaker adapter. Clients use the access layer to authenticate and interact with the rest of the Choona infrastructure; they do not need to know about all the individual services, instead they only need to understand the access APIs.

7 Implementation

The system architecture proposed during the design stages has been divided into three distinct sections for development; services, API and UI. Figure 4 shows how the original architecture has been adapted for the development of the prototype.

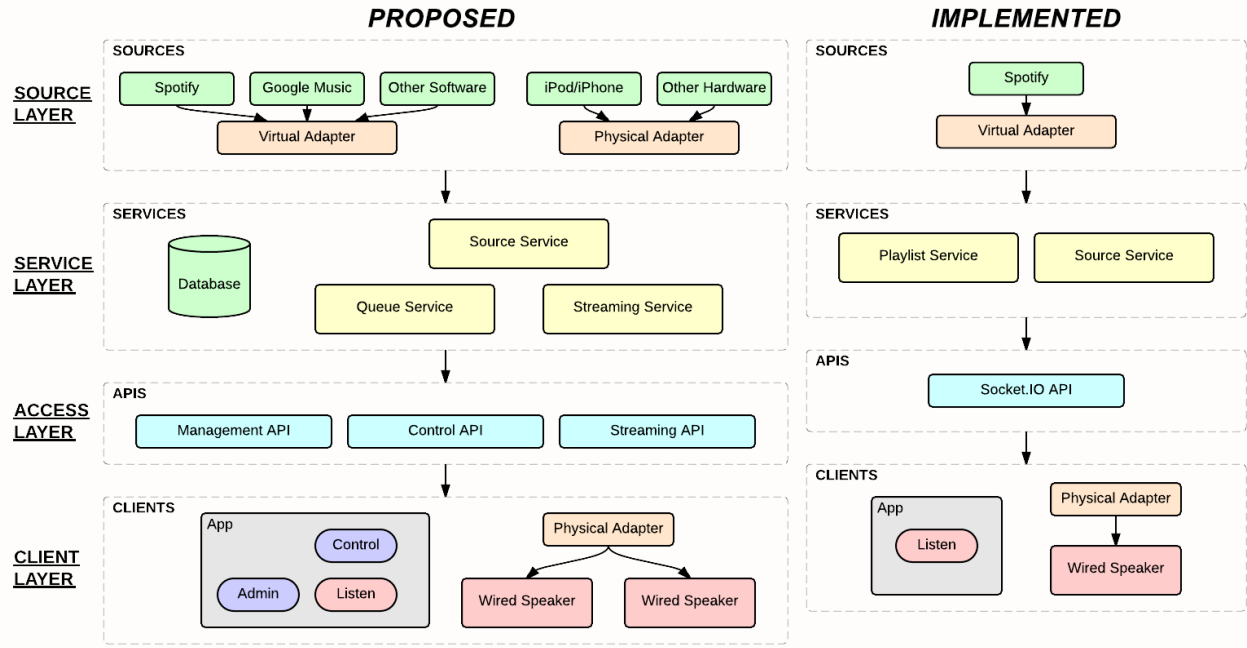


Figure 4: Prototype architecture alongside original proposed architecture

Spotify has been chosen for use as the first type of audio source due to its well documented and accessible API, along with its popularity as a personal music service. No physical audio sources, such as iPods, will be available in the prototype due to the added complexity of interfacing with a physical device.

The queue and streaming services have been joined together to create a single playlist service. This is because the contexts of the two services are very similar and can be regarded as tightly-coupled in functionality; the micro-services architecture specifies² that services should be de-coupled in order to promote scaling and flexibility.

All three types of APIs have been implemented as one singular Socket.IO API for the purposes of the prototype in an effort to make authentication simpler.

Finally, the management and control part of the mobile app has not been implemented due to time constraints; users will only be able to add songs, upvote/downvote tracks and listen to audio.

²Chris Richardson, *Microservice architecture patterns and best practices*.

7.1 Inter-Service Communication

As discussed during the design stages, the micro services architecture offers many advantages over traditional monolithic software. By having everything in the system running as a service it is possible to scale the application in every direction with relative ease. It does, however, introduce the complex problem of inter-service communication. Early implementations of the architecture made use of HTTP APIs³ which work well in simple applications. Relying on HTTP, however, can be seen as a restriction since it is a one-to-one client-to-server connection. In large scale applications it is often necessary to have one-to-many messaging between many different services. HTTP also requires each service to know how to connect to all other services it wants to communicate with, as shown in figure 5.

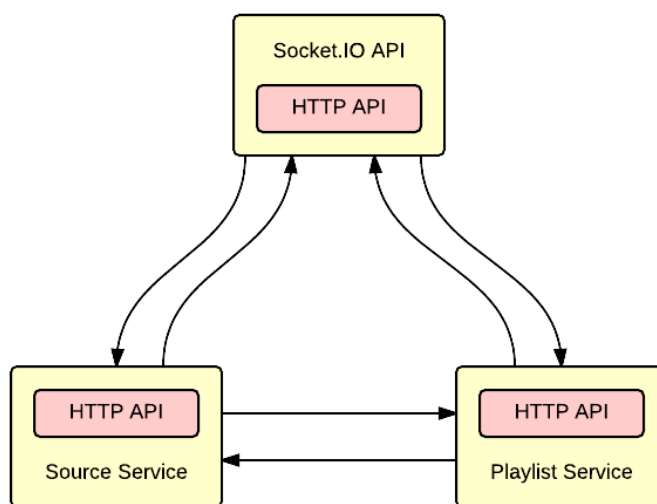


Figure 5: Inter-service communication using HTTP APIs

The most popular alternative to HTTP APIs for inter-service communication is a message broker, such as RabbitMQ⁴ or Redis⁵. In this kind of system, all nodes connect a central broker that is responsible for distributing messages. This has the advantage of a node only needing to know the location of the broker, rather than all other nodes. It also means that messages can be distributed to multiple nodes, removing the one-to-one restriction found in an HTTP-based communication system. Figure 6 shows how a redis-based communication system can be composed.

Messaging systems solve the problem of distributed inter-service communication, however they do introduce their own restrictions. Messages are very low-level; they are simply a single piece of data being transferred between two points. They are entirely stateless and do not

³John Mueller, *Deliving Into the Microservices Architecture*.

⁴Dejan Glozic, *Micro-Service Fun Node.js + Messaging + Clustering Combo*.

⁵Erin Swenson-Healey, *Micromessaging: Connecting Heroku Microservices w/Redis and RabbitMQ*.

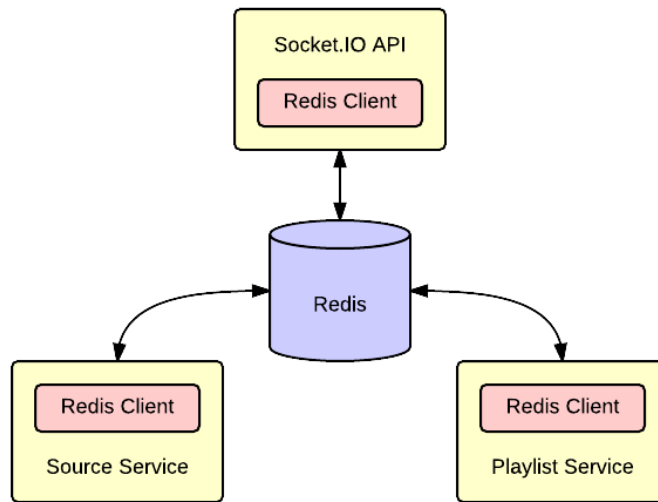


Figure 6: Inter-service communication using Redis

support any kind of meta data. It is therefore necessary to treat a service like Redis as a transport and implement a higher-level protocol on top.

This has been achieved in Choonaa by creating a new communication library called Waterway⁶, built on top of Redis. The roles for both Waterway and Redis can be defined as follows:

Waterway:

- Expose an API for three types of high-level messages:
 - Streams: a continuous flow of individual messages (such as audio data)
 - Requests: a single message with a guaranteed response (synonymous to an HTTP request)
 - Events: a single message with no response (such as a status report)
- Translate between high-level messages and low-level redis messages, identified with keys

Redis:

- Provide an open transport for any type of data to be sent as a message
- Distribute messages to nodes by pattern matching the message keys

As shown by figure 7, Waterway replaces the role of the standard Redis client in a service. This is where the translation between high-level Waterway messages and low-level Redis messages occurs.

⁶Team Hydra, *A Node.js microservice communication platform*.

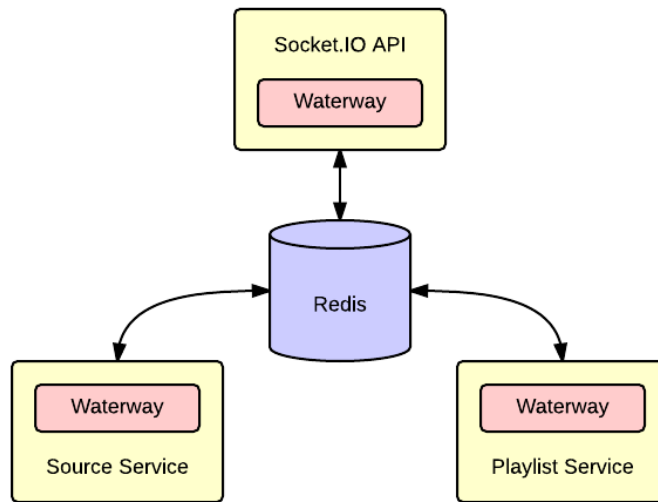


Figure 7: Inter-service communication using Waterway

7.2 Audio Streaming

Choona allows multiple clients to connect and listen to the same audio stream. This means that each client needs to be listening to the same audio at the same time, otherwise the whole system will become out of sync. The easiest way to achieve this is to always ensure the audio is being broadcast live, with a very small buffer window (in the same way that when watching television you always see what everyone else does). Waterway makes this process relatively simple due to its support of streams. As soon as the audio data is received from an audio source such as Spotify it is buffered and streamed out to other services at the exact bitrate of the song. This ensures that all services in Choona see the exact same audio data and the exact same time. It also has the added advantage of allowing the playlist service to know precisely when a track starts and stops; it can just wait for the buffered audio stream to end. Choona's streaming pipeline is described in figure 8.

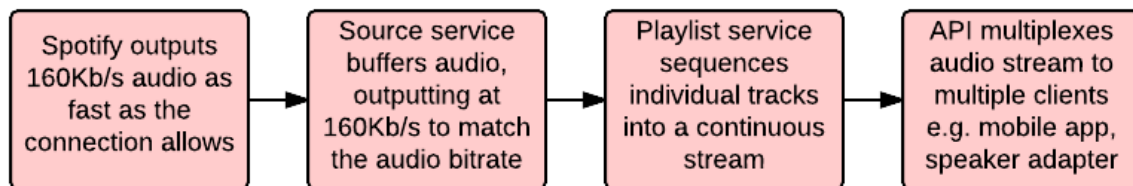


Figure 8: Buffered audio streaming through the Choona audio pipeline

Clients can then choose to handle the audio in whatever method they prefer; for example,

in a browser-based environment the MP3 stream can be converted into raw PCM audio data and played out of the device using the HTML5 Audio API.

7.3 API

Socket.IO is a realtime bi-directional communication protocol and library, designed to replace standard HTTP-based web communication. For Choona, Socket.IO offers the following benefits:

- **Bi-directional communication:** HTTP is a client-server model, where only the client can initiate requests. This is fine for regular websites, however it does not work for scenarios where the server needs to send requests without the client initiating them, such as a realtime application like Choona. To counter this there are now several different strategies and protocols for handling bi-directional communication on the web, many of which are supported by Socket.IO such as WebSockets and long polling.
- **Device support:** Socket.IO will select the best available bi-directional protocol supported by both the client and server when setting up connections, resulting in almost all devices being supported.
- **Binary data support:** As specified in the requirements, the Choona app will need to be able to play audio out of the phone speakers/headphones. This means the audio data will need to be streamed into the app. Socket.IO has built-in support for correctly transferring binary data which means the audio can be streamed using the same communication and authentication methods as the rest of the app.

Choona's Socket.IO API acts as a simple gateway between clients (such as the mobile app) and the many services that make up the backend infrastructure. It has three main objectives:

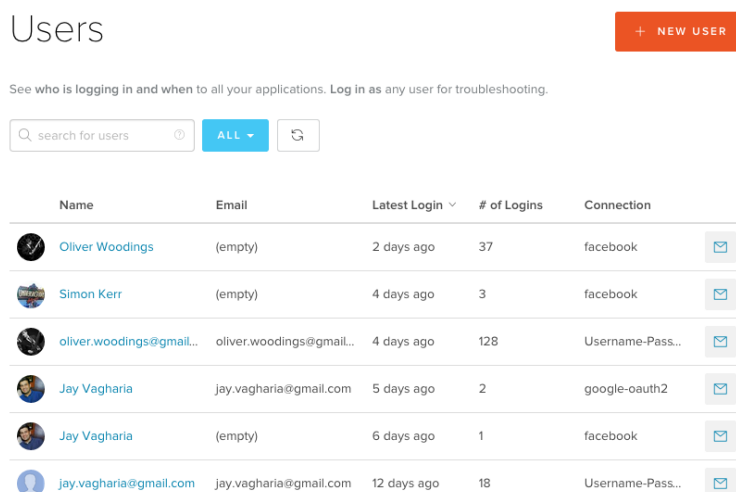
- authenticate users
- ensure requests are authorised
- proxy requests to other services in the system

The API can almost be interpreted as a firewall; it protects the internal services from being unauthorised access and ensures users are authenticated. It also means that clients do not and cannot know about the internal structure of Choona's architecture which adds an element of security by secrecy.

7.4 Authentication

Choona's user authentication is managed by the SaaS platform Auth0. It is now commonplace for applications to allow users to authenticate with many different third parties such as Facebook and Twitter. Developing and maintaining support for even one type of authentication can often prove to be fairly involved, let alone 3 or 4. Auth0 helps developers with this problem by abstracting all the different authentication providers into a single API, giving many advantages⁷:

- Only need to implement one authentication system
- Quickly build embedded login forms using the Auth0 libraries
- Manage all users from a central management interface (Figure 9)
- Easily maintain a single, common view of the user's data
- Integrate with existing user databases















Name	Email	Latest Login	# of Logins	Connection	
 Oliver Woodings	(empty)	2 days ago	37	facebook	
 Simon Kerr	(empty)	4 days ago	3	facebook	
 oliver.woodings@gmail..	oliver.woodings@gmail..	4 days ago	128	Username-Pass..	
 Jay Vagharia	jay.vagharia@gmail.com	5 days ago	2	google-oauth2	
 Jay Vagharia	(empty)	6 days ago	1	facebook	
 jay.vagharia@gmail.com	jay.vagharia@gmail.com	12 days ago	18	Username-Pass..	

Figure 9: Auth0 user management

Auth0 handles the entire login and authentication process without needing any changes to an application's server-side systems. This is achieved by using JSON Web Tokens; a unified way of structuring and encoding authentication data into a JSON object. This object is then encrypted using a private key, allowing it's integrity to be easily validated by third parties using the corresponding public key. JSON Web Tokens are used by the Choona API to ensure a user is authenticated and also to extract information about them for use in other services. This process is shown in figure 10.

⁷Auth0, *Auth0 - Identity made simple*.

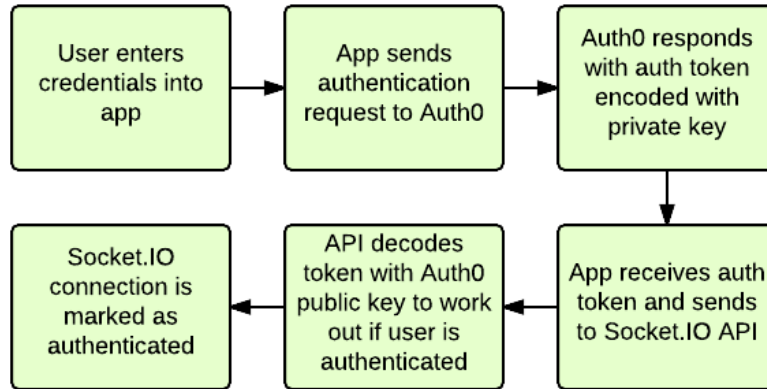


Figure 10: Choona authentication process

7.5 Speaker Adapter

As part of this first iteration of development a prototype of a Choona speaker adapter has been made. The hardware used is a Raspberry Pi 2 Model B running the Raspbian operating system. The software itself is designed to be as autonomous as possible; once the device has an internet connection it will automatically connect to the Choona API, authenticate and start streaming to music from the pre-configured playlist. This is then outputted from the jack socket of the Raspberry Pi into whatever speakers or headphones are connected.

7.6 UI

The Choona mobile app is made using the Ionic framework, which itself is built on top of AngularJS. As discussed in the design stages, building a mobile app using web-based technology allows it to easily be deployed to any platform. Ionic makes this even easier by making available a set of mobile-friendly UI components⁸. Figure 11 shows how PhoneGap/Cordova is used to take the Choona sourcecode and compile it into ready-to-deploy applications for each mobile platform.

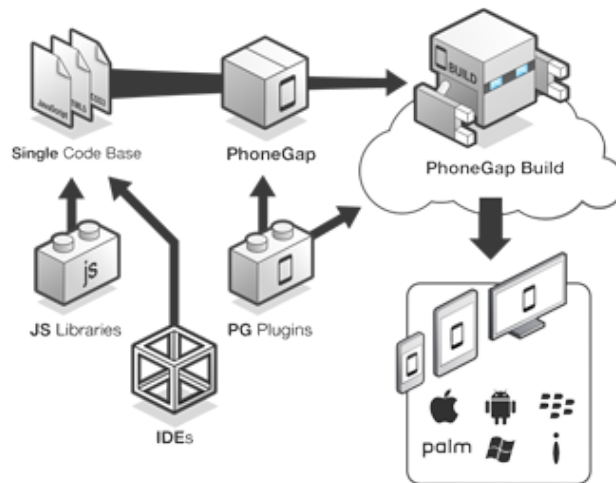


Figure 11: PhoneGap/Cordova build process⁹

The app itself is structured like a typical AngularJS application, using factories, states, controllers, scopes and views:

- **Factory:** a constructor that instantiates or prepares an object for use in the rest of the app
- **State:** a component of the routing state machine used to navigate around the application and render different views¹⁰
- **Controller:** responsible for interfacing between views and the rest of the app
- **Scope:** each controller/state has it's own scope for containing variables and methods, which is inherited and extended from parent scopes
- **View:** HTML mixed in with Angular's data binding syntax

Figure 12 shows Choona's application state/scope tree. Children inherit the scopes of their parents, which makes it possible to pass properties and methods from the root scope

⁸Drify Co, *Ionic: Advanced HTML5 Hybrid Mobile App Framework*.

¹⁰Angular UI, *The de-facto solution to flexible routing with nested views in AngularJS*.

all the way down to nodes at the bottom of the tree. This inheritance also makes it simple to implement authentication restrictions; by applying the restriction in the App state the application will also inherently prevent access to all the child states.

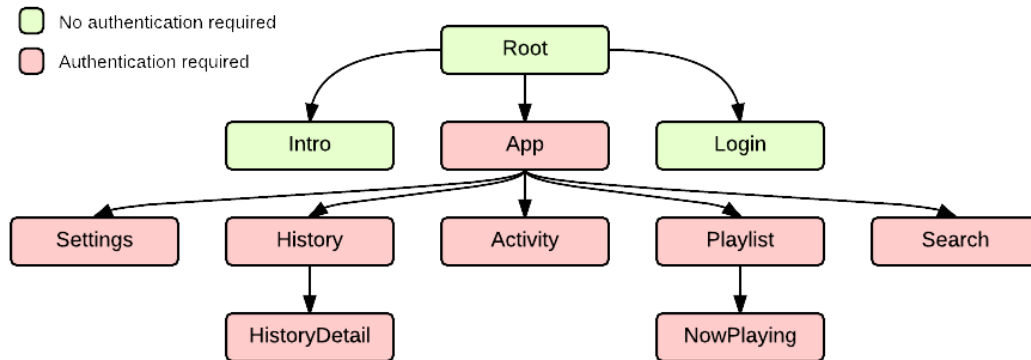


Figure 12: Choona’s application state/scope tree

Scope inheritance is used in Choona to share data between controllers and views, creating a *single source of truth*. This is achieved by storing application-wide data on the highest possible node in the tree; in this case it is the App state. For example, the current playlist information is stores on the App scope since it is required by both the Playlist and History states. If the data was stored on each node individually it would create two different sources of truth that would both need to be maintained separately. All data received from the Socket.IO API is stored at the application level in order to preserve this single source of truth. When data needs to be sent back to the Socket.IO API it is the responsibility of the closest controller/scope to the source of the action. For example if a user upvotes a song on the playlist, it is the role of the playlist controller to inform the API about the change. This whole process is the *data lifecycle* of Choona, and is visualised in figure 13.

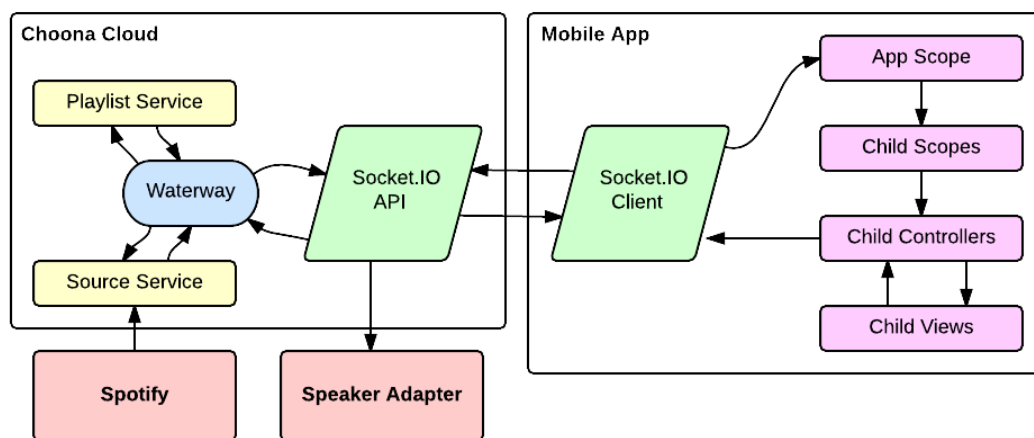


Figure 13: Data lifecycle of the entire Choona ecosystem

8 Evaluation

- User testing
- Client feedback

9 Conclusion

- Learn stuff
- Market readiness

References

- [1] Adobe Systems Inc. *PhoneGap build process*. <http://phonegap.com/uploads/artwork/Build-Diagram-2-Preview.png>. [Online; accessed 11-May-2015].
- [2] Angular UI. *The de-facto solution to flexible routing with nested views in AngularJS*. <https://github.com/angular-ui/ui-router>. [Online; accessed 8-May-2015].
- [3] Auth0. *Auth0 - Identity made simple*. <https://auth0.com/>. [Online; accessed 10-May-2015].
- [4] Chris Richardson. *Microservice architecture patterns and best practices*. <http://microservices.io/>. [Online; accessed 10-May-2015].
- [5] Dejan Glozic. *Micro-Service Fun Node.js + Messaging + Clustering Combo*. <http://dejanglozic.com/2014/04/21/micro-service-fun-node-js-messaging-clustering-combo/>. [Online; accessed 9-May-2015].
- [6] Drify Co. *Ionic: Advanced HTML5 Hybrid Mobile App Framework*. <http://ionicframework.com/>. [Online; accessed 9-May-2015].
- [7] Erin Swenson-Healey. *Micromessaging: Connecting Heroku Microservices w/Redis and RabbitMQ*. <http://blog.carbonfive.com/2014/04/28/micromessaging-connecting-heroku-microservices-wredis-and-rabbitmq/>. [Online; accessed 10-May-2015].
- [8] John Mueller. *Deliving Into the Microservices Architecture*. <http://blog.smartbear.com/microservices/delving-into-the-microservices-architecture/>. [Online; accessed 9-May-2015].
- [9] Team Hydra. *A Node.js microservice communication platform*. <https://github.com/HeilHydra/waterway>. [Online; accessed 11-May-2015].