JASLiN

Alan Lovell

3 May 2020

A project report submitted in partial fulfilment for the degree of

**Bachelor of Science in Computing**

**School of Physical Sciences and Computing**

**University of Central Lancashire**

Abstract

**Problem:**

Problem: What you tackled, and why this needed a solution

**Objectives:**

Objectives: What you set out to achieve, and how this addressed the problem

**Methodology:**

Methodology: How you went about solving the problem

**Achievements:**

Achievements: What you managed to achieve, and how far it meets your objectives.

Attestation

I understand the nature of plagiarism, and I am aware of the University’s

policy on this.

I certify that this document reports original work by me during my University project.

**Signature** **Date**

Acknowledgements

Acknowledge anyone who has helped you in your work such as your supervisor, technical support staff, fellow students or external organisations. Acknowledge the source of any work that is not your own.

Table of Contents

[Abstract i](#_Toc37594145)

[Attestation ii](#_Toc37594146)

[Acknowledgements iii](#_Toc37594147)

[Table of Contents iv](#_Toc37594148)

[List of Figures vi](#_Toc37594149)

[List of Tables vii](#_Toc37594150)

[List of Listings viii](#_Toc37594151)

[1 Introduction 1](#_Toc37594152)

[2 Background and Related Work 2](#_Toc37594153)

[3 Project Planning 3](#_Toc37594154)

[3.1 Introduction 3](#_Toc37594155)

[3.2 Methodology 3](#_Toc37594156)

[3.3 Requirements 3](#_Toc37594157)

[3.4 Potential Solutions 3](#_Toc37594158)

[3.5 Tools and Techniques 3](#_Toc37594159)

[3.6 Legal, Social and Ethical Issues 3](#_Toc37594160)

[3.6.1 First Subsection 3](#_Toc37594161)

[3.6.2 Second Subsection 3](#_Toc37594162)

[3.7 Another Section 3](#_Toc37594163)

[3.7.1 Summary 4](#_Toc37594164)

[4 Design 5](#_Toc37594165)

[4.1 Introduction 5](#_Toc37594166)

[4.2 System Design 5](#_Toc37594167)

[4.3 User Interface Design 5](#_Toc37594168)

[4.4 Summary 5](#_Toc37594169)

[5 Implementation 6](#_Toc37594170)

[5.1 Introduction 6](#_Toc37594171)

[5.2 Section 6](#_Toc37594172)

[5.2.1 Subsection 1 6](#_Toc37594173)

[5.2.2 Subsection 2 7](#_Toc37594174)

[5.3 Section 7](#_Toc37594175)

[5.4 Summary 7](#_Toc37594176)

[6 Test Strategy and Evaluation 8](#_Toc37594177)

[6.1 Introduction 8](#_Toc37594178)

[6.2 Functional Testing 8](#_Toc37594179)

[6.3 Non-Functional Testing 8](#_Toc37594180)

[6.4 User Testing 8](#_Toc37594181)

[6.5 Evaluation 8](#_Toc37594182)

[6.6 Summary 8](#_Toc37594183)

[7 Evaluation, Conclusions and Future Work 9](#_Toc37594184)

[7.1 Project Objectives 9](#_Toc37594185)

[7.2 Self-Evaluation 9](#_Toc37594186)

[7.3 Project Evaluation 9](#_Toc37594187)

[7.4 Applicability of Findings to the Commercial World 9](#_Toc37594188)

[7.5 Conclusions 9](#_Toc37594189)

[7.6 Future Work 10](#_Toc37594190)

[References 11](#_Toc37594191)

[Appendix 1 – Project Proposal 13](#_Toc37594192)

[Appendix 2 – Technical Plan 14](#_Toc37594193)

[Appendix 3 – Title of Appendix 15](#_Toc37594194)

List of Figures

[Figure 1 - Highly Technical Diagram 3](#_Toc476373580)

List of Tables

[Table 1 - Test Results 6](#_Toc476373581)

List of Listings

This list only applies to you if you use code snippets in your report. If you don’t have any listings, remove this whole section including the heading ‘List of Listings’.

[Listing 1 - [Main.java] The main class of the program 5](#_Toc476373582)

# Introduction

For editorial consistency, it is important to use Word styles properly. Word 2003 onwards has so-called ‘quick styles’. If the styles referred to below are not visible on the Home ribbon in the Styles category, choose ‘Apply Styles’ from the down arrow at the bottom right of the Styles category. Styles can then be applied from the drop-down box. To make a style visible as a quick style, choose Apply Styles, then click Styles (the AA icon) or use ‘Alt + Ctrl + Shift + S’, then right-click on the style and then ‘Add to Style Gallery’.

Chapters are entered using the ‘Heading 1’ paragraph style. The Heading 1 style automatically moves to the start of a new page and supplies the next chapter number. Pressing enter on a ‘Heading 1’ heading automatically inserts a ‘Heading 2’ heading underneath.

There should not be any text between a parent heading and its first sub-heading. For example, when you want to write an introductory section for the following sections, give that introductory section an own ‘Introduction’ heading instead of writing it between the heading preceding these sections and the first sub-heading.

As an example: This comment text is incorrectly placed between a ‘Heading 1’ (‘Introduction’) and a ‘Heading 2’ (‘Background and Context’).

Most text uses the ‘Normal Project Body’ paragraph style with 10-point Arial, 1.5-line spacing, single-sided pages.

In general, use the default spacing that headings and paragraphs give you. Avoid using new-lines or spaces to format text. If you need to use quotes, preferably use single curly quotes ‘…’. If you wish to emphasise something, use the ‘Emphasis’ style. In addition, also a ‘**Strong’** style is preconfigured.

Remember to Save frequently while you are working! Check that AutoSaving is enabled under options -> save -> 'Save AutoRecover information every 5 minutes'.

# Background and Related Work

## Project Context

Multi-Room audio is typically an expensive platform to run; a premium product, with a price tag to match. SONOS, the market leader in multi-room audio setups, has several different networked speaker options – however I have some issues with them.

SONOS pricing starts at £179 for their basic speakers, and £399 for just the streamer unit if you want to use your own. Along with this, usage of the SONOS platform is limited to their app on iOS and Android, with a desktop application available for Windows. The system is *not* platform agnostic.

And while alternative solutions exist, such as Naim’s *Mu-so* system or Audio Pro’s C-series, all sport the same critical issues that plague SONOS speakers. More cost-effective alternatives include Amazon Echo or Google Home, however these also have drawbacks for my specific application. Both providers offer average audio quality with limited customization options for audio playback, but a larger issue for my application is the always-on voice monitoring. This may be a convenience for some, but due to privacy concerns that surround such systems it isn’t a desirable feature.

Even alternative “DIY” solutions such as Max2Play have their drawbacks. The only platforms that are capable of multi-room audio are limited by an entry fee, and there aren’t any viable open source alternatives. As an advocate for free and open-source software, this is something I want to change.

I believe it would be feasible to develop my own solution capable of streaming high-quality audio within a local network, without the previously stated drawbacks. The cost per-node can be as low as £25 (plus speakers), which is significantly less than the £399 speaker-less SONOS configuration.

JASLiN will be a platform-agnostic web-controlled LAN-based multi-room audio solution without the cost or gimmicks associated with either Sonos or Echo type systems. After the duration of this assessment, JASLiN will be released as an open-source project –free to anyone on the internet to modify, update, and upgrade.

## Introduction

This project suggests the development of a client-server application controlled from a web-frontend to be ran on raspberry pi hardware, and this chapter serves to review available literature on surrounding technologies that could potentially be integrated into, or utilised to support the product.

This chapter will focus on three key elements – data transmission technologies, backend implementation technologies (including transcoding and handling communication between the UI and the hardware elements), and client playback and audio synchronisation strategies.

## Approach Overview

Broadcasting audio over a local-area network is technically possible, and has been implemented several times already by standalone manufacturers such as SONOS and Google. From the top level, the system is fairly simple and can be approximated with the diagram as below, fig. 1; however understanding how any of these individual components will operate with each-other requires a deeper understanding of the technologies involved.

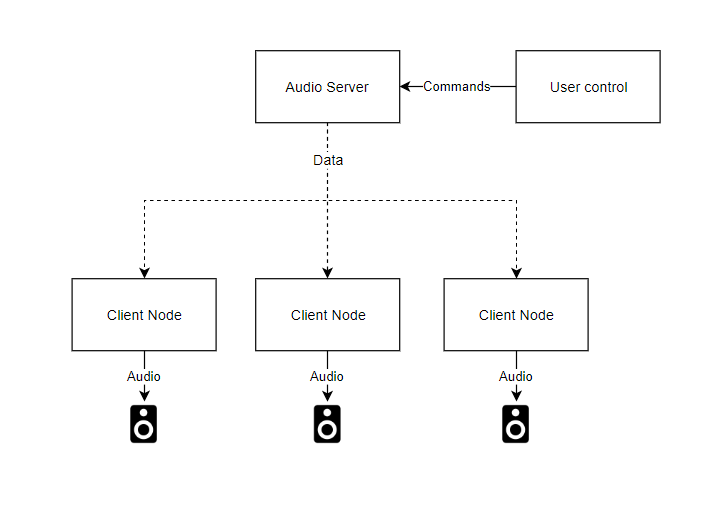


Figure 1: Approximate architecture

## Audio Server

Fundamentally, the audio server takes an audio stream and broadcasts it to client nodes, and can be controlled by an external user application. This brings up four primary questions for the audio server itself;

1. How will I get audio into the system?
2. How will I broadcast audio to the client nodes?
3. How will the user control the system?
4. What language will be used to develop this?

### Providing Audio

There are a lot of different audio formats which could be provided to the System, and very few of them are a mystery to me. As a hobbyist audiophile I have a deep personal understanding of different formats and their benefits and drawbacks. For the purpose of this assignment, I will be sure to limit my file compatibility testing to just MP3 and FLAC, as both of these formats offer free encoders and decoders for commercial use such as LAME for MP3 and Xiph for FLAC, along with being among the most popular audio formats worldwide.

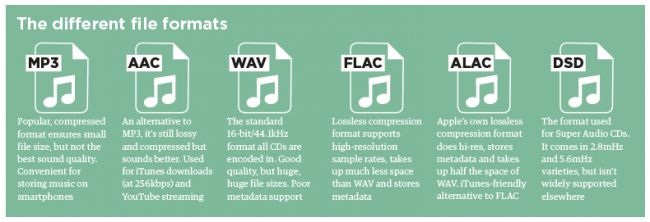


Figure 2: A summary of formats from WhatHifi (2019)

While I cannot upload copyrighted music within my code as file tests, I can generate my own audio to export a variety of formats to ensure compatibility. Legally, I can use my personal collection for testing without breaching any legislation.

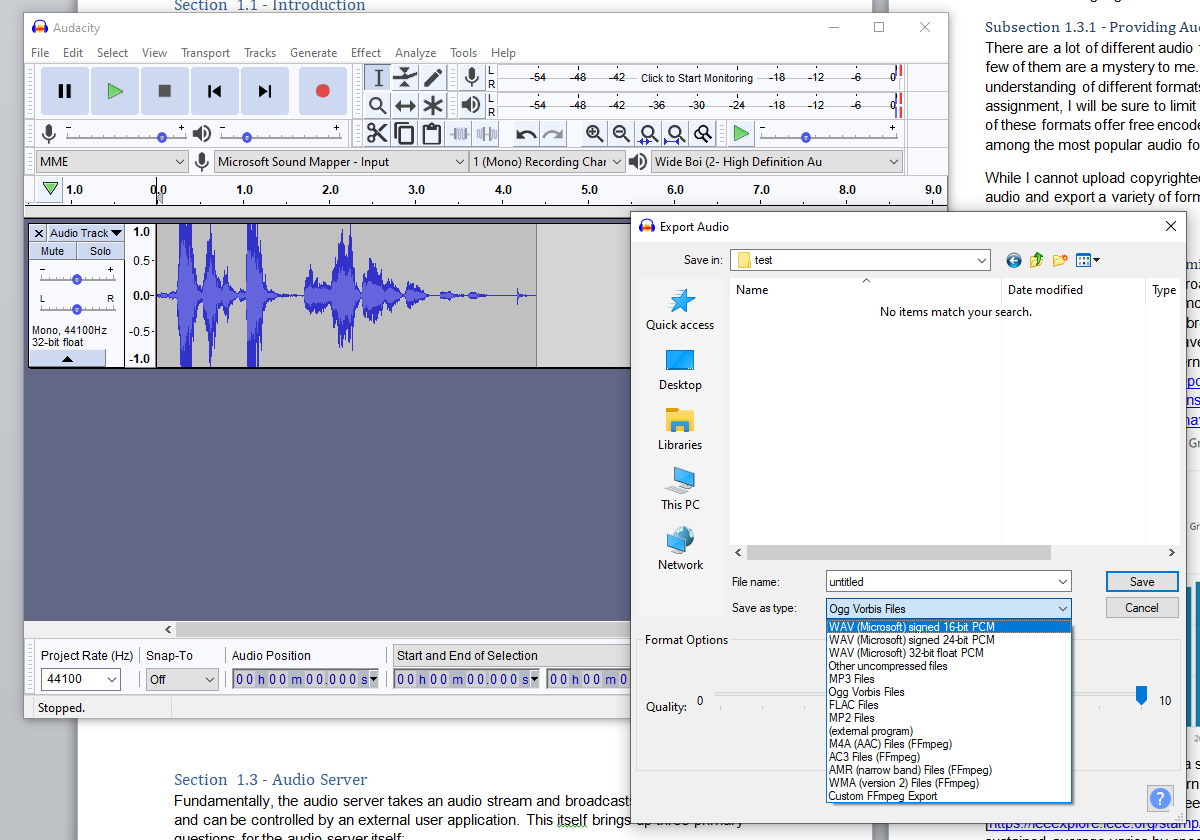
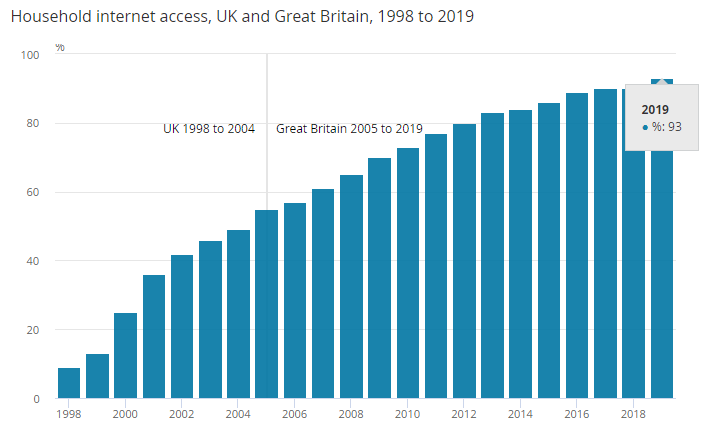


Figure 3: Generating my own audio

Unless I only provide an implementation for digital data files, Analogue to Digital conversion is an issue, and requires consideration for both hardware and software components. In this regard, either the hardware could drive the software choices, or the software could drive the hardware choices.

### Data Transmission Technologies

There are several different approaches which I could take to transmitting the audio data - Digital Audio Broadcast—commonly referred to as “DAB radios”—have been around for a while, and offer one method of broadcasting audio. This does, however, have drawbacks in the fact that very few people have direct access to DAB radio hardware. In contrast, 93% of households in the UK have internet access (Prescott, 2019).  
We can assume from this that a similar percentage of households host their own local-area network and WiFi router. Modern WiFi protocols, such as 802.11ac, allow for theoretical limits of 866.7Mbps transfer speeds according to the IEEE protocol specification (IEEE Standard for Information Technology, 2013); however testing sustained average varies by specific application. One test conducted by Kaewkiriya (2017) produces stable sustained bandwidths of 16.6 Mb/s, however Dolińska (et al, 2019) tested 802.11ac successfully at 93.7 Mb/s. While it is a broad stroke to assume that approximately 90% of the UK have access to a LAN with 802.11ac WiFi connectivity, it should be fairly reasonable to assume the types of people interested in an esoteric open-source audio streaming platform would have this. An alternative would be Bluetooth – this does accommodate the master/slave architecture required of the proposed system, however there are limitations to this – including bandwidth, which is limited to 2 Mb/s according to the Bluetooth Core Specifications (Bluetooth.com, 2019). While this would be acceptable for a single CD-quality audio stream, as this can be compressed to FLAC using LAME to approximately 800 Kb/s, it may struggle with higher resolution audio streams such as “studio masters” at 24-bit 96khz, or direct analogue streaming (such as from a turntable). This would make Bluetooth a poor choice for real-time data transfer.

When using internet protocol to transmit data within a local network, there are several different transmission methods that could be utilised within an audio streaming context; User Datagram Protocol (UDP) and Transmission Control Protocol (TCP/IP) are two such examples. TCP embeds metadata within packets in order to ensure that all the packets are received in the order they are sent – for this to occur, especially on a wireless network which cannot guarantee perfect transmissions, timeouts and retransmission strategies are required. This adds overhead to the data transmissions, both in terms of time and the size of data being sent. For this reason, TCP is cited as a cumbersome protocol, and Vincent (2018) describes the UDP protocol as “*more efficient than TCP and used for real-time communication (audio, video) […]”* and is, as a result, “*[…] is preferable to the overhead of a TCP connection*”. This is backed up by section 6.3 of the AES Standards document for *High-Performance streaming Audio-over-IP interoperability* document (AES, 2015) which states that devices shall use UDP in order to implement AES67 successfully.

### User Controls

As technology grows, so does the number of available platforms; Windows, Mac, iOS, Android, and Linux to list just a few such platforms. If I want this application to be platform agnostic, there are two routes I could take.

1. Develop a cross-platform application using a service such as Xamarin, which allows the developer to produce a single application capable of deployment for different platforms
2. Develop an application that runs in a container, virtual machine, or service. This could be along the lines of Java (running in a host JVM), Docker (running in a hosted container) or a Web Application, which can be accessed from any standard web browser.

The former has the advantage of being able to interface directly with applications through types of remote method invocation; however Xamarin applications aren’t truly platform *independent*. They do, however, help abstract away platform-specific functionality and allow the developer to generate code that isn’t tethered to specific platforms, and allows installable applications to be created much more quickly. An added benefit of this is performance – Xamarin compiles to code native to the target platform, which is always better at leveraging hardware than running through a virtualised host, such as Java or Docker.

The latter offers a greater level of flexibility, however may also result in functionality being stripped away – a web application in particular would be the ideal solution to this, as there is

### Language Choices

Considering platform agnosticism as the ultimate goal for this, it would only make sense to pick a language that is inherently platform agnostic – Java. Java runs in a virtual environment, and the Java code I produce won’t rely on the functionality of the host operating system. Along with this, it also offers the benefit of familiarity, as I already have some experience with Java. While it isn’t the most optimised platform, it does mean that I can develop and test the server software on Windows before deployment to the final hardware, such as a raspberry pi or a other Linux-based client.

## Client Nodes

Once the audio has been broadcast from the server, the client nodes have the role to interpret this information and play it back. This will, essentially, be comprised of three elements.

1. Receiving/decoding the audio data
2. Playing the audio data
3. Ensuring the audio is synchronised between nodes

### Playing back Audio

Streaming audio data and playing it back in realtime would require a transcoder, and while my knowledge on how to produce one of these is limited, thankfully such technologies already exist – FFMPEG, the free audiovisual codec library, provides functionality for this already. (source: https://trac.ffmpeg.org/wiki/StreamingGuide) There may be alternatives to this; however I will not consider them. FFMPEG is a free and open-source codec, which is implemented by both freewares like MusicBee, and high-end audio playback software such as Jriver (Jriver 2019). The hardware requirements for FFMPEG are minimal – while I cannot find any “minimum specification” officially published by the group, Whitmore (2011) comments on this issue with “*I can run ffmpeg on my wireless router so it can really run on anything*”. This alleviates my concern for minimum running requirements for the codec in question.

### Playback synchronisation

While playing audio over a network, there are some considerations to be made as network latency may vary between devices, as a result of network congestion or other factors. There are several patented technologies and designs that aim to mitigate this issue, including patent US7333519B2 for “*manually fine tuning audio synchronization of a home network*.” While it would be illegal to replicate such designs, I believe am allowed to take inspiration from such works.

## Conclusion

I struggled to find resources that may explicitly assist in the development of this project, and spent a lot of time trying to find relevant resources in order to make informed decisions as to how to progress the development of this project. Unfortunately, this area – despite being widely used – appears to lack any significant publications that I could find online.

What I did discover, however, was that UDP streaming over a WiFi LAN while using FFMPEG as the encoding/decoding engine is likely the best route forward for reasonable performance and practical implementation, and adhering to the AES67 specification will give this project a better chance of succeeding from an audio streaming perspective.

# Project Planning

## Introduction

Each of your chapters should have an introduction to tell your readers what they will find in the chapter.

## Methodology

## Requirements

## Potential Solutions

## Tools and Techniques

## Legal, Social and Ethical Issues

### First Subsection

If necessary, also use subsections. Subsections are entered using the ‘Heading 3’ paragraph style (all these heading styles are self-numbering). Do not go lower than Heading 3. Try to avoid if possible.

### Second Subsection

## Another Section

As an example of a figure, consider Figure 1.

To place a figure, insert the picture/diagram/etc. where you want it to be, make sure it is selected and then apply the ‘Project Figure Title’ style which centres the figure horizontally.

Captions are entered through the ribbon menu under ‘References’ -> ‘Insert Caption’ or through right-clicking an image and selecting ‘Insert Caption’. Add the caption text in the box, separated with a dash as the example below shows.

Each figure is numbered automatically, and it is possible to make cross-references to figures.

diss-fig

Figure 1 - Highly Technical Diagram

### Summary

Write a short summary at the end of each chapter. Do not use the words ‘In summary’, we know what it is from the title.

# Design

## Introduction

Each of your chapters should have an introduction to tell your readers what they will find in the chapter. Remember to change the sub chapter headings to some that are suitable to your project.

## System Design

Text goes here.

## User Interface Design

Text goes here.

## Summary

Write a short summary at the end of each chapter. Do not use the words ‘In summary’, we know what it is from the title.

# Implementation

## Introduction

Each of your chapters should have an introduction to tell your readers what they will find in the chapter. Depending on what type of project you are doing, you may name this chapter ‘Investigation’.

## Section

### Subsection 1

Code can be formatted using the ‘Code’ style. An example is shown below. It can be a little bit tricky to keep the formatting when pasting from an IDE but the following works for most IDEs: Copy the text from the IDE, paste it in Word, select the pasted code and change the style to ‘code’. It is worth noting that spell checking is deactivated for the ‘Code’ style.

public class Main {  
  
 public static void main(String[] args) {  
 System.*out*.println("Hello World!");  
 }  
}

Listing 1 - [Main.java] The main class of the program

Captions are entered through the ribbon menu under ‘References’ -> ‘Insert Caption’. Select ‘Listing’ (or add a new Label called ‘Listing’ if it does not already exist) and add the caption text in the white box, separated with a dash as the example above shows. Think about a naming convention for listings and stick to it throughout the report. For example, as seen above,  
‘[ClassName or Filename] Description’.

In case you are mixing multiple programming languages: Consider stating the language name in the caption if it is not obvious from the file name or when there is no file name to refer to. For example, when you use XML and HTML, JavaScript and TypeScript or other languages with similar syntax. A suggestion might be to add the language in parenthesis at the end.

It is also possible to use the ‘code’ style “inline” to highlight commands in normal text by selecting the words to highlight and choosing the ‘code’ style. For example:

This example demonstrates the ping 127.0.0.1 command.

Make sure to write the whole text first and select the part you want to highlight afterwards. When there is no selection, Word applies the selected style to the whole paragraph.

### Subsection 2

## Section

## Summary

Write a short summary at the end of each chapter. Do not use the words ‘In summary’, we know what it is from the title.

# Test Strategy and Evaluation

## Introduction

Each of your chapters should have an introduction to tell your readers what they will find in the chapter. In this chapter you should introduce your test strategy – how have you tested your artefact. You should also talk about user testing. How did you test with real people? How did you select them? What did you ask them to do? What ethical considerations did you adhere to? In this chapter you will also discuss how you have carried out an evaluation of your artefact. This is not the same thing as a total project evaluation.

## Functional Testing

To add a caption to a table, either select the whole table (e.g. by clicking on the + symbol in the upper left corner of the table), right-click it and choose ‘Insert Caption’ or click in any table cell and select ‘References’ -> ‘Insert Caption’ from the ribbon menu. Choose ‘Table’ as label and ‘above the item’ as position. Add the caption text in the box, separated with a dash as the example below shows.

Table 1 - Test Results

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Non-Functional Testing

Text goes here.

## User Testing

Text goes here.

## Evaluation

Text goes here.

## Summary

Write a short summary at the end of each chapter. Do not use the words ‘In summary’, we know what it is from the title.

# Evaluation, Conclusions and Future Work

## Project Objectives

Summarise what you have achieved.

## Self-Evaluation

This section is about yourself. Be honest. Look at where you were situated at the beginning of the project and where you are now. What have you learnt on a personal level, what have you found out about yourself? Try to reflect upon individual goals, experiences, and incidents. No one is perfect, and it is very likely that you will recall both good and bad experiences.

The purpose of the evaluation process is to highlight strengths, correct performance weaknesses, and develop unused skills and abilities. To do this, you must be willing to recognise areas that need improvement or development.

## Project Evaluation

Stand back and evaluate what you have achieved and how well you have met the objectives. Evaluate your achievements against your objectives in section 1.2. Demonstrate that you have tackled the project in a professional manner.

(The previous paragraph demonstrates the use of automatic cross-references: The ‘1.2’ is a Cross-reference to the text in a numbered item of the document, it is not literal text but a field. The number that appears here will change automatically if the number on the referred-to section is altered, for example if a chapter or section is added or deleted before it. Cross-references are entered using Word's **Insert** or **References** menu. Cross-references are set to update automatically when printed but may not do so on-screen beforehand; you can update a field manually on-screen by right-clicking on it and selecting Update field from the pop-up menu or by selecting the whole document and pressing F9.)

## Applicability of Findings to the Commercial World

Summarise what you have achieved and how it can apply to the commercial world.

## Conclusions

Summarise what you have achieved. Do not use the words ‘In conclusion’ or ‘to conclude’ or any derivative of those. We know this is the conclusions from the title.

## Future Work

Explain any limitations in your results and how things might be improved. Discuss how your work might be developed further. Reflect on your results in isolation and in relation to what others have achieved in the same field. This self-analysis is particularly important. You should give a critical evaluation of what went well, and what might be improved.

References

Cuevas, R. et al. (2010a) 'A collaborative P2P scheme for NAT Traversal server discovery based on topological information', *Computer Networks,* 54(12), pp. 120-122.

Cuevas, R. et al. (2010b) 'Book Section', in Kramer, J. D. (ed) *This is a another Book released in the same year,* London: PressAwesome, pp. 150-180.

Greene, D. & Williams, P. C. (1997) *'Linear Accelerators for Radiation Therapy',* 2nd edition, Bristol and Philadelphia: IOP Publishing Ltd.

Jacobson, J. & Andersen, O. (1997) 'Software Controlled Medical Devices', in *SP Report 1997:11,* Sweden: Swedish National Testing and Research Institute.

Ji, H. & Turner, K. J. (1999) 'Specification and Verification of Synchronous Hardware using LOTOS', in Wu, J., Chanson, S. T. & Gao, Q. (eds) *Formal Methods for Protocol Engineering and Distributed Systems (FORTE XII/PSTV XIX),* London: Kluwer Academic Publishers, pp. 295-312.

Turner, K. J. & Jennings, M. A. (2002) 'The Rules of Sailing Races for Hand-Held Devices', *J. Navigation,* 23(5), pp. 114-240.

University of Stirling (2002) *Computing Science and Mathematics Research,*   
http://www.cs.stir.ac.uk/research  
(accessed 01 March 2017).

The reference list above is generated by Word. Under ‘References’ -> ‘Manage Sources’ it is possible to add new references which can later be inserted in the report with ‘Insert Citation’. Example: (Cuevas, et al., 2010a)

Alternatively, you can add a new reference and immediately insert it as a citation by moving the cursor to the place where you want to insert the citation and then choosing ‘References’ -> ‘Insert Citation’ -> ‘Insert New Source’ from the ribbon menu.

You can hide parts of a citation by right-clicking on it, selecting ‘Edit Citation’ and ticking the checkboxes under ‘Suppress’. In the same dialog a reference to specific pages of the cited source can be added (which won’t be displayed in the references list). Example (same as above but with ‘Author’ suppressed and page 120 added): (2010a, p. 120)

Keep in mind that uncited sources will still appear in the references list above. Go to ‘References’ -> ‘Manage Sources’ to see which sources are cited and which aren’t. Sources in ‘Current List’ which have a checkmark are cited.

A custom style has been created for you to use to help with this. Download the file ‘HarvardUCLan2017.xsl’ and copy it to: %appdata%\Microsoft\Bibliography\Style.

Keep Word closed when doing this. To select the style click on ‘References’ -> ‘Style’ and select ‘Harvard – UCLan (2017)’. Tested with Word 2016 (Windows).

There might be some cases the Word bibliography function can’t handle. If you have a tool that suits you better such as ‘RefWorks’, ‘Citethisforme’ or ‘RefMe’ use that instead, then remove the list above and copy the references over. Don’t forget to use the correct notation style.

It is important that you refernece correctly, if you are not sure, ASK! You must be consistent, check your work!

Bluetooth.com. (2019). *211*. [online] Available at: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc\_id=457080 [Accessed 16 Nov. 2019].

Dolińska, I., Jakubowski, M. and Masiukiewicz, A. (2019). *2019 International Conference on Information and Digital Technologies (IDT)*. IEEE, p.122.

IEEE standard for information technology. (2013). IEEE, p.332.

Kaewkiriya, T. (2017). *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*. [S.l.]: IEEE, p.238.

Prescott, C. (2019). *Internet access – households and individuals, Great Britain - Office for National Statistics*. [online] Ons.gov.uk. Available at: https://www.ons.gov.uk/peoplepopulationandcommunity/householdcharacteristics/homeinternetandsocialmediausage/bulletins/internetaccesshouseholdsandindividuals/2019#almost-two-thirds-of-households-now-have-mobile-broadband-access [Accessed 16 Nov. 2019].

Scarrott, B. (2019). *MP3, AAC, WAV, FLAC: all the audio file formats explained*. [online] whathifi. Available at: https://www.whathifi.com/advice/mp3-aac-wav-flac-all-the-audio-file-formats-explained [Accessed 16 Nov. 2019].

Vincent, W. (2018). *TCP vs UDP - William Vincent*. [online] Wsvincent.com. Available at: https://wsvincent.com/tcp-vs-udp/ [Accessed 16 Nov. 2019].

Whitmore, C. (2011). *[FFmpeg-user] Min Hardware Requirements*. [online] Ffmpeg.org. Available at: https://ffmpeg.org/pipermail/ffmpeg-user/2011-March/000094.html [Accessed 16 Nov. 2019].

Wiki.jriver.com. (2019). *Release Notes MC24 - JRiverWiki*. [online] Available at: https://wiki.jriver.com/index.php/Release\_Notes\_MC24 [Accessed 16 Nov. 2019].

AES, 2015. *AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability*, AES, viewed 01 October 2019,<http://www.aes.org/standards/comments/drafts/aes67-r-171107-draft-rev-cfc.pdf>

Appendix 1 – Project Proposal

## The Project Title



[en: /jæs-lɪn/]

**J**ust **A**nother **S**peaker **Li**nk **N**etwork

## Project Context

Multi-Room audio is typically an expensive platform to run; a premium product, with a price tag to match. SONOS, the market leader in multi-room audio setups, has several different networked speaker options – however I have some issues with them.

SONOS pricing starts at £179 for their basic speakers, and £399 for just the streamer unit if you want to use your own. Along with this, usage of the SONOS platform is limited to their app on iOS and Android, with a desktop application available for Windows. The system is *not* platform agnostic.

And while alternative solutions exist, such as Naim’s *Mu-so* system or Audio Pro’s C-series, all sport the same critical issues that plague SONOS speakers. More cost-effective alternatives include Amazon Echo or Google Home, however these also have drawbacks for my specific application. Both providers offer average audio quality with limited customization options for audio playback, but a larger issue for my application is the always-on voice monitoring. This may be a convenience for some, but due to privacy concerns that surround such systems it isn’t a desirable feature.

Even alternative “DIY” solutions such as Max2Play have their drawbacks. The only platforms that are capable of multi-room audio are limited by an entry fee, and there aren’t any viable open source alternatives. As an advocate for free and open-source software, this is something I want to change.

I believe it would be feasible to develop my own solution capable of streaming high-quality audio within a local network, without the previously stated drawbacks. The cost per-node can be as low as £25 (plus speakers), which is significantly less than the £399 speaker-less SONOS configuration.

JASLiN will be a platform-agnostic web-controlled LAN-based multi-room audio solution without the cost or gimmicks associated with either Sonos or Echo type systems. After the duration of this assessment, JASLiN will be released as an open-source project –free to anyone on the internet to modify, update, and upgrade.

## Specific Objectives

* To create a Raspberry Pi-based network audio streaming solution
* To develop a server system for controlling the audio stream
* To control the system from a platform-agnostic web-frontend
* To release the source-code openly for community development post-assessment

**References**

*Technical Documents and Papers*

Viewed 01 October 2019,<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.142.6102&rep=rep1&type=pdf>

AES, 2015. *AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability*, AES, viewed 01 October 2019,<http://www.aes.org/standards/comments/drafts/aes67-r-171107-draft-rev-cfc.pdf>

Cox, J., 2018. *Multi-room audio: everything you need to know | What Hi-fi?,* What Hi-fi?, viewed 01 October 2019. *<*<https://www.whathifi.com/advice/multi-room-audio-everything-you-need-to-know>>

Gstreamer, 2019. *GStreamer*, Gstreamer, viewed 01 October 2019,<<https://gstreamer.freedesktop.org/documentation/?gi-language=c>>

Humfrey, N. 2007. *MAST - Multicast Audio Streaming Toolkit*, Aelius, viewed 01 October 2019, *<*<https://www.aelius.com/njh/mast/>>

IANA, *Real-Time Transport Protocol (RTP) Parameters*, IANA, viewed 01 October 2019,<<https://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml>>

Kuang, T., Williamson, C., 2002. *RealMedia Streaming Performance on an IEEE 802.11b Wireless LAN*, University of Calgary, viewed 01 October 2019,<http://pages.cpsc.ucalgary.ca/~carey/papers/2002/woc2002.pdf>

Locqueneux ,C., 2015. *How To Build A Multiroom Audio System Based On Raspberry Pi And Hifiberry*, HifiBerry, viewed 01 October 2019,<https://support.hifiberry.com/hc/en-us/articles/205699981-How-to-build-a-multiroom-audio-system-based-on-Raspberry-Pi-and-Hifiberry>

Majumba, A., et al., 2002. *Multicast and unicast real-time video streaming over wireless LANs*, viewed 01 October 2019,<https://ieeexplore.ieee.org/abstract/document/1013857>

Smith, N. & Farkas, L., 2019. GitHub - gstreamer-java/gst1-java-core: Java bindings for GStreamer 1.x, Codelerity Ltd, viewed 01 October 2019, *<*<https://github.com/gstreamer-java/gst1-java-core>>

Sonos, 2019, *Sonos,* Sonos, viewed 01 October 2019,<<https://www.sonos.com/en-gb/home>>

## Potential Ethical or Legal Issues

Legal issues may primarily concern licensing agreements with third-party libraries (EG: GStreamer). If this product were to go to market, this would be of even greater consideration, as there may be potential issues if the deliverables were to include the required hardware to operate. If system logs are to be generated by the server or client nodes as a result of the program, the data collected would need to be handled carefully and anonymised to ensure that data regulations within the EU (and the customer’s country) are adhered to. This applies to customer details too, assuming any are stored. Many of these issues can be mitigated by ensuring the licenses of any integrated libraries are adhered to – as JASLiN is planned to be freeware, many of the concerns regarding resale of software packages are alleviated.

Ethically speaking, there are several similar products that operate within a similar domain, for example Sonos systems, and this may pose an issue by means of cutting into their market share. This can be mitigated by not using or implementing any resources published by Sonos or their competitors, and by ensuring any designs, features, or concepts are not lifted from their work. The modern free market allows and encourages competition by design.

The system and source code, once finished, will be published under a presently undecided open-source license which should pose no ethical issues, however may present legal ones depending on the libraries and resources implemented within the project.

## Resources

*Documentation Resources*

GStreamer - <https://gstreamer.freedesktop.org/documentation/tutorials/index.html?gi-language=c>

GStreamer Java - <https://github.com/gstreamer-java/gst1-java-core>

Debian Java - <https://wiki.debian.org/Java/>

Multicast Audio Streaming Toolkit - <https://github.com/njh/mast>

MAST - <https://erg.abdn.ac.uk/users/gorry/course/intro-pages/uni-b-mcast.html>

FFMPEG - <https://www.ffmpeg.org/documentation.html>

Java UDP - <https://docs.oracle.com/javase/tutorial/networking/datagrams/index.html>

Python UDP - <https://wiki.python.org/moin/UdpCommunication>

NodeJS - <https://nodejs.org/en/docs/>

React - <https://reactjs.org/docs/getting-started.html>

Material UI - <https://material-ui.com/>

AnimeJS - <https://animejs.com/documentation/>

*Online Libraries/Lectures*

LinkedIn Learning, formerly Lynda.com - <https://www.linkedin.com/learning/>

Udemy - <https://www.udemy.com/>

Skillshare - <https://skillshare.com/>

*Tutorial Resources*

Running C on Pi - <https://www.raspberrypi.org/magpi/learn-code-c/>

Running Java on Pi - <https://javatutorial.net/raspberry-pi-java-tutorial>

Developing IoT applications for Pi - <https://dzone.com/refcardz/iot-applications-with-java-and-raspberry-pi?chapter=1>

Java Media Framework - <https://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html>

Java UDP Tutorial - <https://www.codejava.net/java-se/networking/java-udp-client-server-program-example>

ReactJS Tutorial - <https://reactjs.org/tutorial/tutorial.html>

pHAT DAC Tutorial - <https://learn.pimoroni.com/tutorial/phat/raspberry-pi-phat-dac-install>

*Software Resources*

Jetbrains IntelliJ IDEA - <https://www.jetbrains.com/idea/download/#section=windows>

Visual Studio Code - <https://code.visualstudio.com/>

Draw.io - [http://draw.io](http://draw.io/)

GIMP - <https://www.gimp.org/>

Inkscape - <https://inkscape.org/>

Git Fork - <https://git-fork.com/>

Github - <https://github.com/>

*Other Resources*

Red5 Media Server - <http://red5.org/>

Icecast - <http://icecast.org/>

Ampache - <https://github.com/ampache/ampache>

## Potential Commercial Considerations - Estimated costs and benefits

There are several different factors for consideration regarding the potential commercial and financial viability of the project. Below is a tabular compilation of the hardware and software requirements for the project endeavour; as the project is intended to be freeware, the cost of demonstration hardware will be provided by myself – however, if I were to charge a customer for a bespoke project such as this, they would provide both the cost of the hardware and the cost of my time, as approximated below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item** | **Purpose** | **Cost** | **Quantity** | **Measure** | **Total** |
| **Hardware** | | | | | |
| Raspberry Pi 4 – 2GB | To act as the Server for the audio stream and web control interface | £45 | 1 | Unit | £45 |
| Raspberry Pi Zero W | To act as Client node(s) for receiving the audio stream | £15 | 1 | Unit | £15 |
| PHat DAC | To output audio from the client nodes | £15 | 1 | Unit | £15 |
| 5v Power Supplies | To provide power to Server and Client node(s) | £5 | 2 | Unit | £10 |
| Cables (assorted) | For connecting devices together | £3 | 5 | Unit | £15 |
| 8GB Micro SD Cards | To store the Operating Systems for the Server and Client node(s) | £5 | 2 | Unit | £10 |
|  |  |  |  |  |  |
| *Hardware Cost* |  | | | |  |
| **Software** | | | | | |
| IntelliJ IDEA Community | To develop Java applications | £0 | 2 | Installation | £0 |
| Visual Studio Code | To develop web applications | £0 | 2 | Installation | £0 |
| Draw.io | To design charts and diagrams | £0 | 2 | Installation | £0 |
| GIMP | To create raster graphic assets | £0 | 2 | Installation | £0 |
| Inkscape | To create vector graphic assets | £0 | 2 | Installation | £0 |
| Git Fork | To manage git repositories | £0 | 2 | Installation | £0 |
| *Software Cost* |  | | | | *£0* |
| **Services** | | | | | |
| Development | Time invested in developing the software | £15 | 400 | Hours | £6,000 |
|  |  |  |  |  |  |
| *Service Cost* |  | | | | *£6,000* |
| Total Cost |  | | | |  |

While the software has potential marketability, I want to undertake this project to release an open-source alternative. This means that there will be no expected return on investment for the project, unless a benefactor voluntarily donates. By doing this, it will benefit the software community by offering an alternative to the current paid-for platforms, and those who want an inherently extensible audio streaming platform. The deliverable artefacts produced throughout this project are not inherently time-bound, thus there is no incentive to rush the development process.

## Proposed Approach

The project will be tackled in five primary stages in order to achieve the core functionality of the software. The lifecycle methodology I will use to implement this will be Agile-based.

Core Functionality is the Minimum Viable Product (MVP) for the project. This will be defined as the following features: Server node capable of broadcasting audio over the network; Multiple client nodes capable of simultaneous network audio playback; Server node capable of controlling the broadcast audio (Change track, play/pause, etc.); User capable of controlling the broadcast audio via a web frontend

20 hours will be invested into designing the solution architecture. 60 hours will be allotted for casting a constant looping audio stream from the server, and another 60 will be allotted for receiving the stream on the client nodes. Afterwards, another 60 hours of work will be put into creating the server backend to control the audio stream, and a further 50 hours for developing the web UI and frontend control mechanisms for the user. After this, I expect testing and system integration to take approximately 50 hours, for a cumulative estimate of 280 hours for the core functionality. This leaves a bucket of approximately 100 hours as either overhead in case of unexpected difficulties in implementation, or to be used for developing and implementing additional functionality to the system.

As of writing this, there are 31 weeks until May 2020 and the System will require approximately 9 hours of work per week to achieve core functionality, not accounting for additional time from the overhead.

Additional Functionalities are those that are outside the scope of the MVP, and will be later analysed and categorised by their desirability and ETA for implementation within the system. In the immediate term, it will be assumed that each additional feature will require between 15 and 20 hours of work to implement successfully. Examples of such additional features include the following:

Room grouping; Channel separation; Per-node volume; User authentication; Playlist management; Display for current playing track; Play/pause/skip (hardware based); Upload songs via web interface; Upload songs via SSH; Upload songs via USB; Cache library of songs; Spotify integration; Calculate delay playback; DSP/EQ; Bluetooth input; and Stereo/Surround capabilities.

Appendix 2 – Technical Plan

**Title**

****

[en: /jæs-l**ɪ**n/]

**J**​ust​**A**​nother​**S**​peaker​**Li**​nk​**N**​etwork

**Summary**

The project will involve managing network connections, broadcasting data via UDP, receiving and playing back audio data, and allowing this to be controlled from a web-based frontend. There will be a database used to manage the node configuration from the server. The primary challenges JASLiN poses will be managing the streamed audio data effectively, and ensuring the end result is listenable, with minimal packet loss over a local network. JASLiN will be undertaken using an agile-based methodology, as there are several complex components, and each can be taken as a different sprint within the project lifecycle.

*Internet technology has matured to where it can support high-quality audio.*

Chafe, Chisholm, et al (2000)

Project Summary: Creating a configurable open-source client-server platform capable of high-fidelity local-area network (LAN) audio broadcast controlled via a platform-agnostic web frontend.

**Deliverables**

A raspberry-pi compatible server and client application capable of providing the speaker link and web frontend functionality. This can be split into three distinct deliverable components:

1. The audio broadcast server software
2. The web frontend and control facility
3. The audio receiver software

**Constraints**

The primary constraints to consider are the deadlines provided by UCLan, time limitations due to working alongside university, and the hardware requirement for the project. The hardware may limit the development of the applications to at home only, as I may not always have access to the required components.

**Key Problems**

Development of JASLiN brings up three notable issues:

1. Audio broadcast
2. Audio decoding
3. Audio playback

However additional issues may arise through the implementation of UDP protocols and synchronisation between the client nodes utilised within this project. Along with this, the web interface will require investigation to design an effective and usable frontend for individuals to easily utilise the System. Fortunately, the scope of the project is well-defined with a set expectation for the Minimum Viable Product for the duration of this project.

**System and Work Outline**

The first steps to be taken are to flesh out the design documentation and create a list of components, split down further into class diagrams (and other UML plans) in order to understand the theoretical operational behaviour of the Software. These will also show relationships between the classes for the server, the clients, and the web-server and will be produced within the *Background and Related Work* phase of the project. During this phase, the UI for the web-servershall be planned and opinions gathered from members of the public, and I will use social media platforms to generate this data.

Next I will install and configure any required software, along with my repositories and the hardware for the project by installing the base operating system onto an SD card.

I will write the Server initially with a single looping audio file - this test file will be included in my repository, and will need to be sourced from a public media library such as WikiMedia. I can utilise a network monitor tool such as Wireshark to ensure the packets are being transmitted as expected. Once this component operates as intended, I will develop the initial client application to play back the audio stream. This should be the core of JASLiN, and from there the process can be refined and optimised until it works fully, then more features can be added. Unit tests will be designed and created early within the development stages of the Software.

This process should be finished by the beginning of January, with the web server and control mechanisms completed and integrated by the end of January.

In order to undertake this project, I need to gain a broader understanding of networked applications and how to implement broadcast/multicast transmissions over a protocol such as UDP in order to stream the audio. I have a reasonable idea of how to encode and decode the audio, however the streaming element - the core reason for undertaking this project - is the primary area I need to research in order to better understand.

I already have some skills required, however this project will develop on them including:

* Java development
* Web application development
* React development
* Web application development
* Object-oriented development
* UML diagrams

There are, as mentioned previously, areas which I have some understanding of and will benefit greatly from practical implementation of:

* Audio encoding and decoding
* Material UI and other web-frameworks

And there are some elements I have no experience in, and will require me to acquire new skills to complete:

* Streaming data
* UDP/Broadcast/multicast in Java
* Server/client configurations in Java

These topics appear to have a lot of information surrounding them, which will assist my project greatly.

**Project Activities**

There are a number of activities and phases required for a successful project, however these can fundamentally be broken down into Planning, Designing, Developing, Testing, and Documentation. The planning and design phases are, in this timeline, merged into the same category. The rest however have fairly distinct phases.

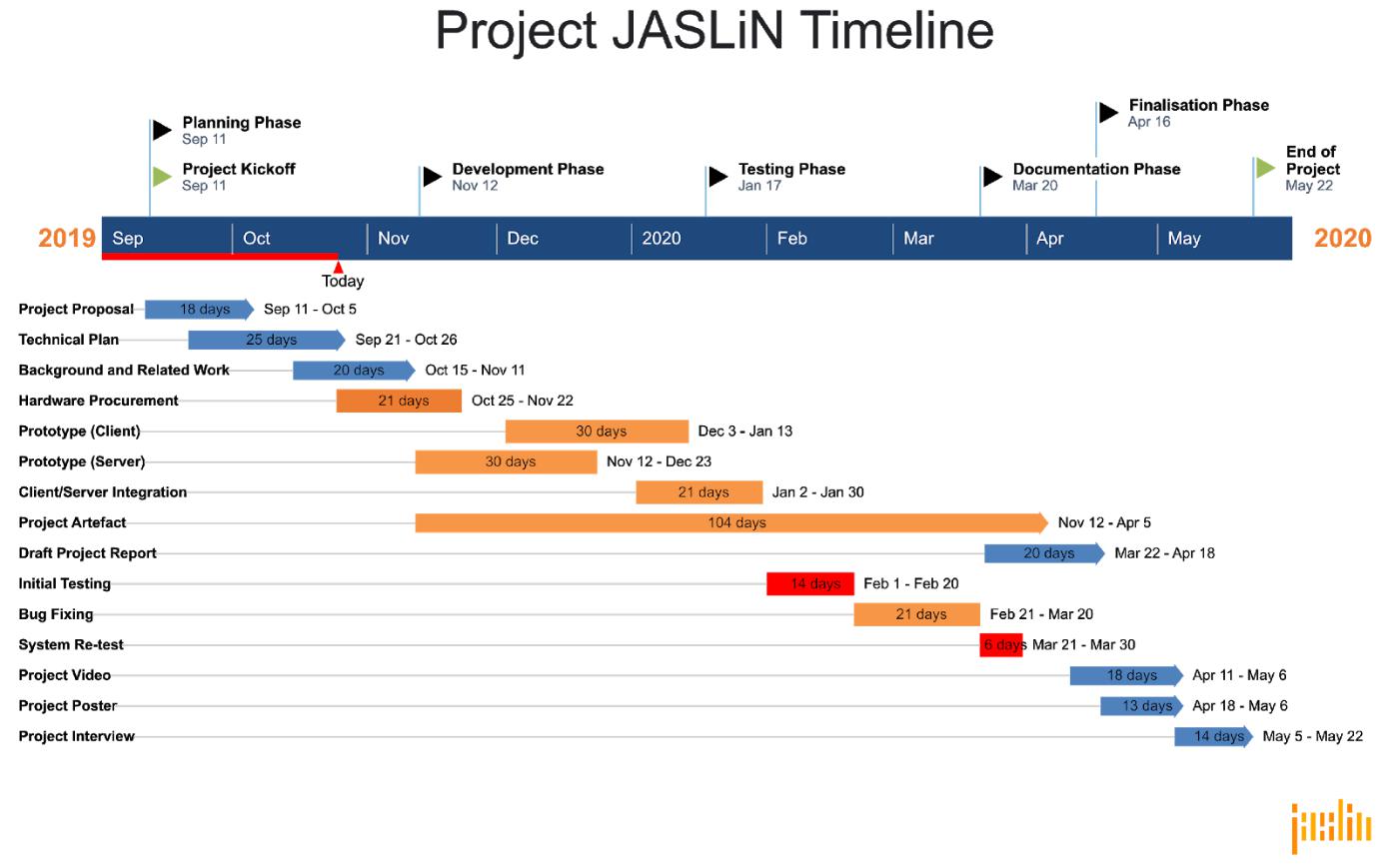


Figure 1: Gantt chart timeline of Project JASLiN

|  |  |
| --- | --- |
| Key | Description |
|  | Colours |
| Green | Project start and end |
| Orange | Stage produces a Software artefact |
| Blue | Stage produces a Documentation artefact |
| Red | Stage produces a Testing artefact |
|  | Shapes |
| Arrow | Product with UCLan hand-in deadline |
| Square | Project goal without explicit hand-in deadline |
|  | Table 1: Gantt chart key |

In my initial proposal document, I listed several potential features to be included in JASLiN, however due to time constraints it will not be possible to include all of these features. To prioritise these features, I have organised them using a MOSCOW analysis. While I expect the Must Haves to have been completed, I can’t be certain of the features beyond this as I have little experience dealing with network audio and UDP transmissions. I have made approximations for the hours required to achieve a minimum viable product, and there are some hours allotted outside of that for additional development and testing of features outside of the MVP scope.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Feature* |  |  |  | *Description* |
|  | The System​​***Must****​*​be capable of | | | |
| Audio server backend |  | The System must feature a backend server for broadcasting the | | |
|  |  | audio data across the network | | |

|  |  |  |
| --- | --- | --- |
| Web server frontend |  | The System must feature a server to host the web frontend and |
|  |  | handle the control interface between the user and the Audio Server |
|  |  | Backend |
| Play/Pause functionality |  | The user must, at a minimum, be able to play and pause the |
|  |  | currently playing music through the Web frontend |
| Node playback |  | The client nodes must automatically play back any audio broadcast |
|  |  | to them |
| Room grouping |  | Individual speaker nodes must be able to be grouped into a single |
|  |  | room |
|  | The System​​***Should****​*​be capable of | |
| Channel separation |  | Each node should be able to be configured in software to only output |
|  |  | left or right audio data |
| Analogue input |  | JASLiN should be capable of taking an analogue input (e.g.: from a |
|  |  | turntable) and using that stream for the broadcast |
| Per-node volume |  | The user should be capable of adjusting the volume of each speaker |
|  |  | node from within the web interface |
|  | The System​​***Could****​*​be capable of | |
| Calculate delay playback |  | The System could allow the user to set a delay for each node to |
|  |  | account for latency caused by spatial and network differences |
| Upload songs via SSH |  | The System could allow the user to upload media files via SSH to the |
|  |  | server node using an SFTP client |
| Upload songs via web interface |  | The System could allow the user to upload media files stored on their |
|  |  | device via the web interface to the server node |
| Upload songs via USB |  | The System could allow the user to upload media files via USB to the |
|  |  | server node |
| Cache library of songs |  | The System could provide a way of caching libraries for faster |
|  |  | access to audio files |
| User authentication |  | The System could provide a method of authenticating users within |
|  |  | the web interface |
| Playlist management |  | The System could provide a way of managing playlists |
|  | The System​​***Will not****​*​be capable of | |
| Play/pause/skip |  | As the nodes are Pi-based, it would be possible to add custom |
|  |  | hardware to them in order to add additional functionality |
|  |  | (play/pause/skip) without using the web interface. However, they will |
|  |  | not be included in this version of JASLiN |
| Bluetooth input |  | The System could utilise Bluetooth communication to receive audio |
|  |  | data for broadcast across the LAN, but this feature is very low priority |
|  |  | and as such will not be included in this version of JASLiN |
| Spotify integration |  | Integration to external audio playback services such as Spotify will |
|  |  | not be included in this version of JASLiN |
| DSP/EQ |  | Digital Signal Processing and Equalisation are advanced audio |
|  |  | manipulation features that will not be included in this version of |
|  |  | JASLiN |
| Display for current playing track |  | As the nodes are Pi-based, it would be possible to add custom |
|  |  | hardware to them in order to add additional functionality (a display |
|  |  | for currently playing track) without using the web interface. However, |
|  |  | they will not be included in this version of JASLiN |
| Stereo/Surround capabilities |  | A type of “room grouping”, speakers could be grouped into stereo or |
|  |  | surround configurations from the Web UI, however this feature will |
|  |  | not be included in this version of JASLiN |

Table 2: MOSCOW-prioritised list of features

Core Functionality is the Minimum Viable Product (MVP) for the project. This is defined as-above within the ​*Must Have​*section of Table 2.

20 hours will be invested into designing the solution architecture. 60 hours will be allotted for casting a constant looping audio stream from the server, and another 60 will be allotted for receiving the stream on the client nodes. Afterwards, another 60 hours of work will be put into creating the server backend to control the audio stream, and a further 50 hours for developing the

web UI and frontend control mechanisms for the user. After this, I expect testing and system integration to take approximately 50 hours, for a cumulative estimate of 280 hours for the core functionality. Assuming the Project requires 400 hours, this will leave a “bucket” of approximately 100 hours as either overhead in case of unexpected difficulties in implementation, or to be used for developing and implementing additional functionality to the System.

As of writing this, there are 26 weeks until May 2020 and the System will require approximately 10 hours of work per week to achieve core functionality, not accounting for additional time from the overhead.

**Risk Analysis**

Any software project has potential risks associated with it, and JASLiN is no exception to this. Below is a list of potential risks to the project (Table 3), and how I intend to mitigate them in order to increase the chance of JASLiN succeeding as a project.

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Severity** | **Likelihood** | **Mitigation Strategy** |
| Loss of data | High | Low | Utilising a remote repository for all code |
|  |  |  | elements will allow for redundancy in the |
|  |  |  | event of local data loss. |
| Loss of working | Fatal | Low | I will ensure access to a secondary |
| machine(s) |  |  | machine to try and mitigate this risk, |
|  |  |  | however if my primary working machine |
|  |  |  | fails it would likely be fatal to the project. |
| Lack of time to work on the | High | Medium | Prioritise effectively and allocate the |
| Project |  |  | required time for the Project in advance |
| Underestimating the work | High | Medium | Read documentation for components and |
| required |  |  | APIs in advance to ensure a basic |
|  |  |  | understanding of how to use the tools I |
|  |  |  | plan to |
| Limited understanding | Medium | High | I have a limited understanding of some |
|  |  |  | areas of this project, including UDP and |
|  |  |  | RTC. These are critical elements to the |
|  |  |  | project, however this can be mitigated by |
|  |  |  | ensuring I do thorough research and |
|  |  |  | planning before implementation. |
| Loss of JASLiN hardware | Medium | Minimal | If either of the Raspberry Pis required for |
|  |  |  | the project fail, it could be a significant |
|  |  |  | setback for ensuring the platform operates |
|  |  |  | as intended - however, it won’t be fatal, as |
|  |  |  | I could simply order replacement units |
|  |  |  | within the week. This would, however, |
|  |  |  | stretch the budget of the project more |
|  |  |  | than anticipated. |
| Failure to meet deadlines | Severe | Low | By prioritising and logging my progress |
|  |  |  | regularly, I should be able to mitigate this |
|  |  |  | by maintaining momentum throughout the |
|  |  |  | project. |

Table 3: Risk analysis

**Options**

While I want to take an agile approach for this, I believe a waterfall timeline best describes the workflow described previously, and will likely reflect the route I take. This does present some level of risk, as projects that follow a waterfall lifecycle tend to fail without a rigid set of requirements and expectations, however, I believe the requirements for this project are concrete enough to utilise a waterfall methodology.

The target environment doesn’t strictly have to be a Raspberry Pi - I could simply host it on a Windows-based server (x86 or x64 based architectures), however by using a Pi it reduces the cost of entry, which is a primary concern for the project.

The Raspberry Pis themselves require an operating system, and there are several different distributions of Linux, other \*nix, and Windows systems available. The exact choice is presently undecided, though it will almost certainly be a CLI-based OS.

**Potential Ethical or Legal Issues**

Legal issues may primarily concern licensing agreements with third-party libraries (EG: GStreamer). If this product were to go to market, this would be of even greater consideration, as there may be potential issues if the deliverables were to include the required hardware to operate. If system logs are to be generated by the server or client nodes as a result of the program, the data collected would need to be handled carefully and anonymised to ensure that data regulations within the EU (and the customer’s country) are adhered to. This applies to customer details too, assuming any are stored. However if I implement log-in functionality within the System, it would be paramount to ensure these details are stored securely so as to adhere to the Data Protection Act.

Licensing issues can be mitigated by ensuring the licenses of any integrated libraries are adhered to – as JASLiN is planned to be freeware, many of the concerns regarding resale of software packages are alleviated. This also applies to the software IDEs being used to develop the System.

|  |  |  |
| --- | --- | --- |
| **Software** | **License** | **Excerpt** ​(if applicable) |
| IntelliJ IDEA Community | Apache 2.0 - | “It can also be used for commercial |
|  |  | development.” - Jetbrains, n.d. |
| Visual Studio Code |  | “Is VS Code free? |
|  |  | Yes, VS Code is free for private or |
|  |  | commercial use. See the product |
|  |  | license for details” - Microsoft, n.d. |
| Draw.io |  | “You own the content you produce |
|  |  | with draw.io and may use it for any |
|  |  | purpose, including commercially” - |
|  |  | Atlassian, n.d. |
|  |  |  |
| GIMP | GPL v3 | “Can I use GIMP commercially? |
|  |  | Yes, you can. GIMP is free software, |
|  |  | it doesn’t put restrictions on the kind |
|  |  | of work you produce with it.” - GIMP, |
|  |  | n.d. |
| Inkscape | GPL v2 |  |
| Git Fork |  | “Commercial as well as |
|  |  | non-commercial use is allowed, free |
|  |  | of charge.” - Pristupov, n.d. |

Table 4: Software licenses

While some licenses could not be found, I have located explicit excerpts from the software providers current domains as of October 2019, declaring their applications acceptable for use within commercial settings - refer to Table 4.

Ethically speaking, there are several similar products that operate within a similar domain, for example Sonos systems, and this may pose an issue by means of cutting into their market share. This can be mitigated by not using or implementing any resources published by Sonos or their competitors, and by ensuring any designs, features, or concepts are not lifted from their work. The modern free market allows and encourages competition by design.

The System and source code, once finished, will be published under a presently undecided open-source license which should pose no ethical issues, however may present legal ones

depending on the libraries and resources implemented within the project. Along with this, I will ensure to abide by the Ethical Guidelines established by UCLan.

**Commercial Analysis**

While the software has potential marketability, I want to undertake this project to release an open-source alternative. This means that there will be no expected return on investment for the project, unless a benefactor voluntarily donates. By doing this, it will benefit the software community by offering an alternative to the current paid-for platforms, and those who want an inherently extensible audio streaming platform.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item** |  | **Purpose** |  | **Cost** | **Quantity** | **Measure** | **Total** |
|  |  | **Hardware Costs** | | |  |  |  |
| Raspberry Pi | 4 – | To act as the Server for the audio |  | £45 | 1 | Unit | £45 |
| 2GB |  | stream and web control interface |  |  |  |  |  |
| Raspberry Pi Zero W | | To act as Client node(s) for |  | £15 | 1 | Unit | £15 |
|  |  | receiving the audio stream |  |  |  |  |  |
| PHat DAC |  | To output audio from the client |  | £15 | 1 | Unit | £15 |
|  |  | nodes |  |  |  |  |  |
| 5v Power Supplies | | To provide power to Server and |  | £5 | 2 | Unit | £10 |
|  |  | Client node(s) |  |  |  |  |  |
| Cables (assorted) | | For connecting devices together |  | £3 | 5 | Unit | £15 |
| 8GB Micro SD Cards | | To store the Operating Systems for |  | £5 | 2 | Unit | £10 |
|  |  | the Server and Client node(s) |  |  |  |  |  |
| *Hardware Cost* |  |  |  |  |  |  | *£110* |
|  |  | **Software** | |  |  |  |  |
| IntelliJ | IDEA | To develop Java applications |  | £0 | 2 | Installation | £0 |
| Community |  |  |  |  |  |  |  |
| Visual Studio Code | | To develop web applications |  | £0 | 2 | Installation | £0 |
| Draw.io |  | To design charts and diagrams |  | £0 | 2 | Installation | £0 |
| GIMP |  | To create raster graphic assets |  | £0 | 2 | Installation | £0 |
| Inkscape |  | To create vector graphic assets |  | £0 | 2 | Installation | £0 |
| Git Fork |  | To manage git repositories |  | £0 | 2 | Installation | £0 |
| *Software Cost* |  |  |  |  |  |  | *£0* |
|  |  | **Benefits/Income** | | |  |  |  |
| Development |  | Time invested in developing the |  | £15 | 400 | Hours | £6,000 |
|  |  | software |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| *Service Cost* |  |  |  |  |  |  | *£6,000* |
| Total Cost |  |  |  |  |  |  | £6,110 |

Table 5: Costs and benefits of Project JASLiN

The software chosen to develop all the assets for this project were all selected due to their free licensing for commercial usage – while several of them do offer paid-for versions, the majority of them do not tend to offer tangible benefits to me as a developer.

As I would be developing this project as an open-source alternative to the present marketed options, I would not make any money from this – the cost cited below for my time is a minimum viable price to be able to develop the solution full-time, and as JASLiN is not for sale, would have to be funded by donations from supporters. This could be done by crowd-funding like Kickstarter or IndieGoGo, subscription-based platforms like Patreon, or simply posting PayPal (or other e-payment method) details online. Assuming I were to undertake this project full-time, payments would be collected monthly during development, in order to cover my living costs while making this.

The total cost to me would be simply investing in hardware – however, as I already own many of the items listed above, the immediate cost to me should be around a third of that cited. The hardware does, however, require purchase early in the lifecycle of the product. The System cannot be fully developed or tested without it.

**Employability Contribution**

For my final-year project at University, I chose to develop a pair of client and server applications capable of streaming audio over a LAN, with the aim to offer an open-source alternative to SONOS-type multi-room audio solutions. Throughout the lifecycle of this project, I improved my project management skills and timekeeping, along with gaining a deeper understanding of how RTC UDP transmissions are handled within a client/server configuration. I also gained experience with Java, and several web-technologies such as React and Material UI, along with digital audio encoding and decoding strategies.

**References**

Atlassian, n.d. ​*Pricing – draw.io​*[online] Atlassian, Available at: ​<https://about.draw.io/pricing/>​[Accessed​ 26 October 2019]

Chafe, Chisholm, Leistikow, Scavone Wilson, 2000. ​*A SIMPLIFIED APPROACH TO HIGH QUALITY* *MUSIC AND SOUND OVER IP​*, Penn State University,<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.142.6102&rep=rep1&type=pdf>

GIMP, n.d. ​*GIMP - Frequently Asked Questions​*, GIMP, Available at:

<https://www.gimp.org/docs/userfaq.html#can-i-use-gimp-commercially>​[Accessed 26 October 2019]

Jetbrains, n.d. ​*Download IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains​*, Jetbrains

[online] Available at: [​https://www.jetbrains.com/idea/download/#section=windows](https://www.jetbrains.com/idea/download/#section=windows)​[Accessed​ 26 October 2019]

Microsoft, n.d. ​*Visual Studio Code Frequently Asked Questions​*[online] Microsoft, Available at:​<https://code.visualstudio.com/docs/supporting/FAQ>​[Accessed 26 October 2019]

Pristupov, n.d. ​*Fork - a fast and friendly git client for Mac and Windows​*[online] Git Fork, Available at:

<https://git-fork.com/license>​[Accessed 26 October 2019]

Appendix 3 – Title of Appendix

You may have one or more appendices containing detail, bulky or reference material that is relevant though supplementary to the main text: perhaps additional specifications, tables or diagrams that would distract the reader if placed in the main part of the report. Make sure that you place appropriate cross-references in the main text to direct the reader to the relevant appendices.

Do not blindly include all of your code in the appendix or the body. Only include the parts you refer to in the report. You can put those parts either in the appendix or in the body (e.g. in the “Implementation” part).