University of South Wales – Faculty of Computing, Engineering and Science

*Collaborative*

*Audio*

*System*

Milestone 1 – Sub-Report

Research & Experimentation

by

Jack Holmes

10016600

|  |  |
| --- | --- |
| 1st Assessor | 2nd Assessor |
| Keith Verheyden | Colin Morris |

Computing Individual Project

Table of Contents

Introduction 3

Research Approach 3

Project Planning 5

Waterfall Method 6

Incremental Method 6

Summary 7

System Overview 8

Initial Proposal 8

Initial Server Setup 8

Basic System Requirements 9

Functional requirements 9

Non-functional requirements 9

Evaluation of Existing Technologies & Techniques 11

Collaborative Services 11

Kompoz 11

Splice 11

Jamly 12

Version Control Systems 14

Conflict Resolution 14

Open-Source Audio Tools 14

Audacity 15

Sound Exchange – SoX 15

Summary 16

Server Development 18

Server-side Languages 18

Python 19

Prototype Implementation 20

Python IDE’s 21

Client Development 23

Audio Recording & Playback 23

GUI 23

Mobile Application Development 24

Web, Native, or Hybrid 24

Intel XDK 25

File Formats, Storage & Transfer 26

Audio File formats 26

Data Storage 27

Data Exchange & Data Interchange Formats 27

Python & JSON 28

FTP 29

Concurrency Considerations 31

Processes & Threads 31

Issues Regarding the Implementation of Concurrency 32

Synchronisation Issues 33

Potential System Design 36

Conclusion 37

References 39

# Introduction

This sub-report documents the efforts made by myself in researching appropriate material for the design and development of the final deliverable. To show the full extent of the research performed, all of the stages involved have been included, even aspects that won’t make it into the final design of the end deliverable.

The purpose of this report is to not only to show what research has been carried out, but to also show how the research will ultimately affect the end deliverable. The main contents of this report will discuss:

* Software evaluation
* Platforms and technologies
* Methodologies
* Academic research
* Evaluation of commercial alternatives

The project I have undertaken is fairly technical in nature, therefore, I have split the report into themes that are common in computer science and other relevant areas.

## Research Approach

For the majority of my research I follow an approach consisting of a review of relevant literature and practical application. Prior knowledge has given me a basic idea of how to implement the deliverable. The purpose of this research is to refine my knowledge and help me make more informed decisions when designing the system.

A literature review will serve as the method to discover new, more efficient implementation options. I will be reviewing sources mainly from books, however I will also be citing scholarly articles and the web.

To make sure that certain technical aspects of the system can be implemented, I will be trialling some of the basic functionality. Throughout this report I will experiment with various technologies and platforms to find the most suitable method for my development.

# Project Planning

At the beginning of the project all of the tasks that needed to be completed were detailed in a Gantt chart. A of the activities that needed to completed were placed in a sequential fashion, where one task led on to the next. Research unveiled that a waterfall process model had been implemented. Hughes and Cotterell (2006, pp.283) explain how students can easily misjudge time allocation of tasks, simply due to the fact that they will be using tools that they are unfamiliar with. This point will play a major factor in the chosen planning approach, as most of the tools, software and techniques will have never been used in prior development.

|  |
| --- |
|  |

## Waterfall Method

The waterfall method is seen as one of the more basic models of system development. It typically follows the pattern as shown below.

|  |
| --- |
| Requirements  Design  Development  Validation  Maintenance |

As Hughes and Cotterall (2006, pp. 75-76) explain, the waterfalls main strength is also its downfall - its lack of flexibility. The clear-cut nature of the waterfall model promotes strict adherence to deadlines, however there is great importance in defining the exact deliverables early on as there is very little room for change later on in the development process.

For very straightforward projects with low risk of change, the sequential nature of the waterfall model is ideal. This project will most likely go through unpredictable iterations of changes and experimentation thus making this method less than ideal.

## Incremental Method

With the limitations of a waterfall model in mind, a process model more suited to this type of project was needed. The spiral model is an iterative process model. As Hughes and Cotterall (2006, pp. 76-77) explain – aspects of a system that are hard to specify could be better developed using an incremental approach. The spiral model’s key strength is the fact that it is able to cope if various phases of the project encounter complications or unforeseen issues that hinder development.

Incremental development is well suited to experimentation and prototyping. Sommerville (2010, pp.32-33) states that incremental development achieves a final system implementation by progressively validating new versions. This will be ideal for this system as the intention is to implement at least one prototype.

As this is such an early stage in the project it is impossible to pin down a time allocation for every single task that will be performed. With greater experience in risk assessment, or if the system was very clear to produce, then a waterfall model would be the better choice to base this system upon. The fact that an incremental approach, such as the spiral method, accommodates possible contingencies in development makes it an ideal approach to use.

An incremental approach is almost ideal for this project, so what are the disadvantages? Is there a limit to the amount of iterations a system can cycle through? Sommerville (2010, pp.34-35) explains that an incremental approach may degrade a system’s structure, as functionality is constantly being ‘tacked on’, and resources can be used up by constantly documenting new implementations.

## Summary

There are a lot of factors to consider when implementing a project plan. Every process model has its pros and cons, however there is flexibility when it comes to choosing. A process model is simply an abstraction for a certain application, which means it doesn’t have to be strictly adhered to. The most advantageous approach to use for this project is most likely incremental development. With the amount of uncertainty within this project, the most valuable asset needed in a plan is flexibility. Iterative development excels when it comes to trying out various features through prototyping – a task that will feature heavily during the development of the system.

The disadvantages of a waterfall development outweigh those of a typical incremental based approach. This project is small compared to the large projects that these process models have been based upon. There aren’t other team members to manage, no customers are involved and it is unlikely that the system will grow so large that it will become unmanageable. These points alone should lessen the impact of any disadvantages that an incremental approach may have.

# System Overview

The main objective is to develop a working prototype of the system. As stated in the introduction, the current focus is to develop a server-side environment that provides collaborative functionality to multiple client devices. This section discusses the purpose of the system and what the system’s functionality is. This is by no means the final layout of the system, as continued research and experimentation continue to show other avenues of development.

## Initial Proposal

The initial proposal for the system specified the development of an environment that would allow users to upload and interact with each other’s content (audio files) via a server. This proposal was heavily centred on a mobile application, with the mentality that the GUI and audio manipulation techniques would be the main technical hurdles to overcome. However as research continued, it became clear that the opposite was true. The research unearthed that there are many frameworks and open-source aids that can create a basic client application for you on many different platforms with ease. Servers on the other hand, or at least frameworks for collaborative audio specifically, were virtually non-existent. For this reason the main focus shifted to researching how the server will operate.

### Initial Server Setup

This diagram was created at a very early research stage to better define what was being created. It shows, very basically, what a client will typically upload or download and how the server will process these files.

|  |
| --- |
|  |

The diagram shows how the server processes files, however it doesn’t indicate the general structure in which users will be able to collaborate.

## Basic System Requirements

The system requirements haven’t changed drastically during my research, apart from the focus changing to a server-side system. My research was based on fulfilling the requirements outlined below.

### Functional requirements

Functional requirements are determined by the outcome of individual functionalities within the system itself. From an early point it was established what actions the system would need to perform. Functional requirements are typically generated from a use case diagram.

### Non-functional requirements

A non-functional requirement can be outlined as a characteristic that affects the broader elements of the system. Sommerville (2010, pp.87-91) explains that the non-functional aspects of a system are often the most critical. A user can typically avoid an encounter with abnormal system functionality, whereas a single non-functional feature that isn’t meeting requirements can often cripple the entire system. The non-functional requirements that affect my system include:

#### Implementation

As the system currently stands, all programming for the server will be carried out with the Python programming language and will be executed on a Linux-based platform. The client prototype will be implemented using Python’s TkInter GUI library. If the project is then ahead of schedule then a final client for mobile will be developed using Intel’s XDK platform using JavaScript, HTML5 and CSS.

#### Performance

The nature of the system will mean that the server will have significant overhead, especially if there are many clients interacting with the server at once. This will mean techniques, such as concurrent programming, may need to be employed to achieve a more efficient system.

The client implementation will most likely require slightly less computation. Audio file formats will need to be used efficiently in order to get quick transmission between client and server.

# Evaluation of Existing Technologies & Techniques

The initial steps of research involved finding similar systems that already existed and how they achieved a collaborative environment. This research was vital as it uncovered solutions to implementation issues that were being faced. In this section, existing solutions and technologies related to the end deliverable will be discussed, specifically, how they handle multiple users, how they implement a collaborative environment and the various tools and techniques used.

## Collaborative Services

### Kompoz

Kompoz (Kompoz, 2015) could be defined as collaboration as a service, it allows musicians to upload projects from their computer and download other user’s projects via the website.

#### Collaborative Environment

To collaborate with other musicians, the user selects a session from the site’s ‘Collaborations’ page. A user can then submit a recording to the session by uploading it from their computer. However the creator of a session must accept a submission for it to become part of the final track, otherwise it is labelled as an ‘idea’.

In terms of version control, this implementation may be the least technical in the list. Essentially a ‘creator’ user controls all submissions, so there is no risk of conflicting versions as only a single user makes decisions about the tracks.

### Splice

Splice (Splice, 2015) is essentially another cloud collaboration service. The main target audience seems to be musicians creating electronic dance music. This is assumed due to the content on the site, however it would seem there is nothing stopping other genres from using the service. A Splice client application must be downloaded to the user’s computer in order to upload and download projects. Projects must be made in a compatible digital audio workstation (DAW).

#### Collaborative Environment

The service’s main feature is providing access to other user’s projects, allowing you to create alterations via your personal DAW. Once a user has made adjustments to the project and uploads it, it becomes a new ‘version’ alongside the original project.

In this case no merging of files actually happen unless you consider the operations that the user performs in their DAW. This software highlights a different approach to collaboration - all users can create their own versions, which effectively make version control unnecessary.

### Jamly

Jamly (Jamly.co, 2015) is another musical collaboration service, however all functionality is provided via its website. Jamly uses JavaScript within the browser’s website to record both audio and video from the computer’s default camera and microphone. To collaborate with other musicians, the user must select a session from the home page. The user is then allowed to watch the selected session. If they like what they hear, the user can go on to record their own section within the session. Once on the recording page, the user records via their device whilst listening to the original version to keep in time. Finally the user can listen/watch a preview of the session, correct any latency in the recording and upload.

#### Collaborative Environment

Jamly operates quite differently to the other two services listed. Firstly, the service records your audio directly, whereas the other two services required you to upload a file. Secondly, it records video, the benefits of this are debatable. It may be beneficial as a musician to observe other musicians for visual cues when recording (e.g. watching a drummer to help you keep in sync), However the musicians within the video are quite small and the video is often of poor quality depending on each user’s capture device.

Jamly’s approach to version control is also unique in this list. I tested the version control system within Jamly by creating my own session and then recording two sections simultaneously on separate tabs within the browser.

|  |  |
| --- | --- |
| Macintosh HD:Users:jackholmes:Documents:Development:Resources:Images:No Conflict.tiff  (Jamly.co, 2015) | Macintosh HD:Users:jackholmes:Documents:Development:Resources:Images:Conflict.tiff  (Jamly.co, 2015) |
| Default  A typical merge operation – a recording is made and saved to the session. | conflict  The session forks if a concurrent recording is made during another recording. |

Jamly effectively forks a session in two if a version conflict is detected. This implementation of version control avoids the need for a ‘controlling’ user to manually choose what to do with the separate versions. To record a new section, a user must select which version that they want to collaborate with.

|  |
| --- |
| Macintosh HD:Users:jackholmes:Documents:Development:Resources:Images:Post Conflict.tiff |
| Post-conflict  Additional recordings can be added to either end of the fork. |

Any new user can now collaborate with the new version or the old version. One criticism of this system, however, is that there is no warning to any user that a concurrent recording is being made. Concurrency issues are discussed later in the report.

(Jamly.co, 2015)

Another note to make about this system is that it allows you to compensate for recording latency and change the mixing volume for the recording. This highlights some possible issues with this type of system. Firstly all client devices could be different, this will mean that they will all have different recording equipment, different processing power (latency issues), and they could all output a mixture of audio formats.

## Version Control Systems

Due to the collaborative nature of this system, I looked for collaborative services or open-source frameworks that may provide some insight to how to deal with multiple users uploading and editing shared content. There are many systems that exist for version control, however most of them are built for software development. They all deal with the same issue - providing a system to manage change in documents or files between multiple users. This section highlights the techniques used by these systems.

### Conflict Resolution

There are typically two methods to avoiding conflicts: merging and locking.

#### Merging

Git (Git-scm.com, 2015) is a version control system. The main feature of Git that is most relevant to this project is how it handles conflict resolution. Git uses a merging concurrency model. This basically means that a user has to physically manage how data from two files will create a new file. Users are allowed to edit the same document as each other at a local level.

#### Locking

Locking may be considered an old-fashioned technique, however it is still a valid option to consider. Locking is still used by version control systems, however, in most cases it is used in conjunction with a merging approach. In systems where users are unable to communicate, a locking mechanism may be necessary. It is known that binary files, such as audio, are often impossible to merge back into each other in version control (Collins-Sussman, Fitzpatrick and Michael Pilato, 2015).

However it shouldn’t be assumed that locking prevents conflicts, as no system can do this perfectly. Where merging allows many users to work on the same file, locking explicitly allows only a single user to make changes at a time, often hindering productivity. This indicates that a hybrid approach may be the best approach for this system.

## Open-Source Audio Tools

A key technicality in this system is manipulating audio files. Research started with open-source solutions. Using an open-source implementation was beneficial as it could provide the necessary techniques involved in editing audio files.

### Audacity

Audacity (Audacityteam.org, 2015) is a program that allows users to record and edit audio files. An advantage of Audacity is that it’s cross-platform and the source code is available for download. The initial reaction to the source code was that it was far too large to gain any useful information from. The majority of Audacity is written in C++, trying to understand such a large program would consume far too much research time. By inspecting the various libraries that Audacity used, promising libraries were discovered – ‘libsoxr’ and ‘WxWidgets’.

### Sound Exchange – SoX

Within Audacity, ‘libsoxr’ is a library used by SoX (Sox.sourceforge.net, 2015), an audio file manipulation program. SoX is a command line utility. This was a benefit, as the functionality of the program could be accessed without delving into the ‘libsoxr’ library. SoX is actually constructed from a collection of other open-source libraries, giving it a huge amount of functionality. SoX is also open-source and is written in C, making it cross-platform, ideal for prototype development.

Following experimentation with SoX and Python, a SoX command can be invoked using Python’s ‘os.system()’ call. Below two audio files are merged into one called ‘111111.wav’. The command’s exit code is also captured, to check if the program executed successfully.

|  |
| --- |
|  |
|  |

## Summary

As seen above, collaboration can be implemented using many different techniques. In terms of collaborative functionality between users, two patterns seem to emerge in all of these services and tools.

#### User-Dependent Collaboration

Within Kompoz and Git is typically a hierarchical structure of users that governs how files and content are authorised into a final ‘mix’. These collaborative services keep the collaboration process ‘on rails’ - a linear approach, however one that could be deemed suitable for its application. Git is slightly different to Kompoz as it gives users full reign of what they can change and upload. However, if conflicting files have been made, or multiple users try to upload to the same branch, there can be great complications when trying to merge it back into the original ‘master branch’. Git’s strength and weakness is that it needs micro-management of every single change.

#### User-Independent Collaboration

Services such as Splice and Jamly use a system where users don’t have to control how collaboration flows. A single user doesn’t control iterations of collaboration, instead whenever a user uploads an idea, a new ‘session’ is effectively created. Jamly manages conflicts by simply ‘forking’ into two separate versions. Splice doesn’t even consider version control, it simply saves files as new ‘versions’ for each user.

#### Collaborative Approach For This System

The chosen collaborative approach for the system will essentially sculpt how some of the key functionality of the system is implemented (e.g. the server). Whichever approach is chosen will still result in a collaborative environment, it will just change the experience for the user. In this case, neither approach holds advantages over the other, they are simply different takes on the same problem. For the sake of this system, characteristics will be taken from both types of collaboration. It is intended that a user driven ‘session’ structure (similar to Kompoz) will be implemented into the final system. However, a ‘forking’ mechanism (inspired by Jamly and Git), will be implemented where clients conflict.

# Server Development

The server will perform tasks essential to the operations of the system. It will have to manage clients and will have to perform data transfer tasks.

### Server-side Languages

In order for a language to be considered for the development of a server, it needs to be able to interact with key aspects of typical server functionality, such as sockets and protocols. As already stated, this project will most likely change, as experimentation with various features and implementations is carried out. This means that the language chosen will need to be flexible, cross-platform and high-level. The two languages this report will focus on will be PHP and Python. This was a decision based upon research findings and personal familiarity. The system could have been written in C, however low-level program development is known for complexity and time consumption. A study between various languages (Prechelt, 2000) shows that productivity while using low-level languages is much lower compared to using a high-level language such as Python.

When initially experimenting with code, PHP seemed an obvious choice. It’s known for web-based applications. PHP is a general-purpose programming language, like C, however it is considered to be high-level, and is interpreted. PHP’s main purpose is for developing dynamic web pages, however it’s also known for its flexibility in being able to develop a much broader scope of applications.

Python is again another high-level interpreted language. Its main purpose is for improving productivity (Rossum, 2015). Python’s syntax and structure is centred on human-readable code making it ideal for beginners. Python is also highly flexible due to a large standard library (Piotrowski, 2006).

Both languages seemed ideal for this system, however only a single language would be used in the final deliverable. By carrying out various experiments with both languages, issues were discovered with PHP that made it less suited for this project.

### Python

Python was chosen as the primary server-side language. Python is ideal for this project. The language is centred on fast deployment and supports iterative development through its extendibility. The decision was made by putting the Python language against the most important requirements that were needed for the development of the system.

#### Flexibility

Flexibility is one of the most important characteristics that the project will require. Flexibility will ensure that the language can adapt to the changing needs of the project specification. As already stated, Python’s main strength, or perhaps its weakness, is its large standard library. Python makes use of modules, which are basically libraries that can extend the functionality of a program. If a user wants to program socket functionality, they can, by simply using the import keyword.

If the project will make use of prototyping, then Python is ideal. Python itself can execute at a similar speed at compared to other languages of the same type. If extra functionality, or optimisation is still needed, Python supports extension through C and C++ code. C/C++ code can be written as modules and simply imported into a Python script. Python’s ability to balance high-level and low-level functionality makes it ideal for this project.

#### Usability

One of the most noticeable features of a Python program compared to other languages is its clarity. Python was created with readability of code as one of its main features, this was done to make code more reusable (Rossum, 2015). Python’s clarity also helps in learning the program. As stated at the beginning of the report, learning any tools is important to consider, as it will have a knock-on affect with any set deadlines.

PHP uses C style syntax, which can differentiate between each coder. Python’s code structure means there is usually only a single standard way of laying out code. Below is a prime example of what Python avoids.

|  |
| --- |
|  |

#### Efficiency

Due to Python’s high-level nature, it can often achieve the same outcome in fewer lines of code than a similar program written in a low-level language. Code written in Python can sometimes be up to 5-10 times shorter than the same implementation in a lower level language, such as C++ (Python.org, 2015).

Efficiency in writing short concise code shouldn’t be confused with efficient execution of code. Python runs relatively slow compared to a low-level programming language such as C. Speed and efficiency rely greatly on how code is implemented, so judging Python on these factors is rather subjective.

As said earlier, optimisation can be added later in development through module extension. This just shows that Python doesn’t excel when developing performance sensitive applications, however, it is capable of very fast deployment. As already discussed, Python deployment is incredibly fast. Python is a great ‘glue’ language; it provides the basic structure of the program, but can provide faster execution by letting the user implement modular low-level code.

### Prototype Implementation

To prototype my system I will need access to a server environment. This will allow me to test both the client and server aspects together. I will be using XAMPP (Apachefriends.org, 2015), which is built upon standard server tools, such as Apache HTTP server, proFTPD server and MySQL Database. Apache HTTP server is the most popular web server in the world. It is typically used with a UNIX/Linux OS because they are generally free and multi-user oriented. This is useful as all of my development will be in OSX, a derivative of UNIX. A possible issue with XAMPP is that it is quite general use and doesn’t support Python exclusively. To get around this, mod\_python (Modpython.org, 2015) would have to be downloaded. mod\_python is an Apache module that embeds a Python interpreter within the server application. Using mod\_python would allow Python to access Apache HTTP related functionality and low-level features such as sockets within the Apache server.

An alternative to using Apache is implementing the server directly through Python. Python already has full server functionality in its wide selection of modules. Below shows some basic functionality through the ‘socket’ module, a simple message is sent and received using both the client and the server.

|  |
| --- |
|  |

Choosing this approach would give me true control over server functionality, although it may consume considerable development time implementing a full-fledged server.

There are also frameworks for server development, such as CherryPy (Cherrypy.org, 2015). These frameworks provide a huge amount of functionality and make system deployment even faster by wrapping existing standard library code with an even more efficient API. CherryPy is a viable option for my development, especially when used in conjunction with the standard Python library.

### Python IDE’s

#### IDLE

Python on OSX comes equipped with its default IDE – ‘IDLE’, a simple application that provides access to the Python shell. It gives users all of the standard features of an IDE, such as a debugger and syntax highlighting. This application is suitable for smaller applications. Efforts were then made to look for other platforms that could help me obtain maximum productivity when developing Python code.

#### PyCharm 5

After some basic research, the PyCharm (JetBrains, 2015) IDE was discovered. PyCharm 5 allows you to create ‘projects’, similar to other IDE’s, such as Visual Studio. In IDLE the user has to manually create directory structures themselves in order to make sure all of their code can co-interact. PyCharm 5 ensures that all the correct resources are available and provides easy navigational access to them.

Another feature that make PyCharm 5 favourable over IDLE is the fact it has code completion. PyCharm 5 will display suggestions as you type, increasing productivity even further. As concurrency may feature in my system, PyCharm 5 can visualise the state of threads running in the program, this will aid me greatly during development, as developing concurrent systems can often be quite a complex task.

# Client Development

The client side of development will involve implementing support for key elements, such as, file transfer, audio recording, audio playback, GUI and user keyboard input. Tools and techniques have been considered for both the prototype and the final implementation.

The first implementation of the system will be a prototype, this will require a different approach to a full-fledged implementation. Prototype development is centred on quick deployment, so that the core functionality of a system can be tested. This means the tools that are chosen to develop the system will need to reflect this.

## Audio Recording & Playback

I will be using PyAudio (PyAudio, 2015) to perform the core functionality of the client. PyAudio is a cross-platform audio I/O library, based on the PortAudio library. It will be used in conjunction with wxPython (Wxpython.org, 2014) to implement the client functionality. The pyAudio module provides record, play and wire (record then playback instantly) functionality.

## GUI

The default choice for a GUI API was Tkinter, a GUI package that is part of the Python standard library. The main issue however is its number of widgets, which are somewhat limited compared to other GUI libraries. As a result, another GUI library was chosen – wxWidgets. Using Tkinter could result in development being undertaken and then realising that it can’t implement the features needed.

wxWidgets library was first encountered when searching through the source code for Audacity. wxWidgets is a cross-platform GUI library. The implementation for Python that will be used is called wxPython. This implementation isn’t part of the Python standard library, however it has more widgets than Tkinter, making it far more flexible.

Existing audio tools have already been developed using wxPython and after viewing the source code it was evident that it is relatively low-hassle to implement.

|  |
| --- |
|  |
|  |
| (Driscoll, 2015) |

## Mobile Application Development

Originally, the client side of the system was planned to be built on a mobile platform, however as shown in this report, research has changed the path of development. If time permits then a mobile deliverable may also be developed. Here is the research exploring this area.

### Web, Native, or Hybrid

Applications for mobile can be developed using a variety of methods, each with its own strengths and weaknesses.

Web applications are the easiest type to develop out of the three. All smart mobile devices now feature a web browsing functionality so that they can display web pages. Web applications use a devices web functionality to display information in the same way. Web apps are essentially web pages packaged as an application. By making use of CSS and HTML, web apps are very quick to develop, using web development packages such as Dreamweaver. The downside to web applications is they have very limited access to the devices native functionality (microphone, camera, file management, etc), however, with the emergence of HTML5, web applications continue to grow more and more powerful, and can access native functionality (Mobilehtml5.org, 2015).

Native Applications are on the other end of the spectrum of mobile application development. These applications require much more time to develop and require an IDE such as Visual Studio or xCode to develop. Development of a native app gives the developer much more control. All of the device’s functionality is available through a native API, giving access to low-level features. Native development requires much more specialised knowledge on specific mobile architecture.

Hybrid applications obtain the best features of both web and native development. Using a development tool such as Intel XDK or PhoneGap, you can achieve native functionality, whilst implementing the GUI using a web-based approach.

### Intel XDK

Intel XDK (Intel, 2015) is an IDE for hybrid mobile app development. It uses the Cordova (Cordova.Apache.org, 2015) API to encapsulate the native functionality of major phone OS’s. This effectively lets you write an application once, that will work on multiple devices (this is impossible in native applications development). Intel XDK uses JavaScript, HTML, and CSS, yet still is able to deliver an app almost as flexible as one built natively.

|  |
| --- |
| A sample project provided by Intel XDK. |

# File Formats, Storage & Transfer

We know that data that is going to have be stored on both the client and the server. At some point some of this data will need to be transferred over a network. This section focuses on file related issues that may affect the operations of the system.

## Audio File formats

The largest type of file that the system will have to handle will be audio. Audio formats can range in size quite drastically and can be either lossless or lossy in their compression. When choosing which audio format the system was most likely to use, four main points were considered:

* How will it be created?
* What operations will be performed on it?
* What is its purpose?
* How will it be stored?

Audio files are going to be created on the client device. Already, this raises the question: What formats does the client support? The prototype will be developed on a typical desktop operating system (most likely OSX), these systems generally support a very large range of audio formats by default. However if development was undertaken for a more specialised platform such as mobile, then certain distributions will have limitations on what formats they support. SoX supports an extensive amount of formats. If client audio files differ too much then SoX will be able convert the audio into a different format once they’re uploaded to the server.

Audio files will undergo modification in the system, such as merge operations. These operations will typically take an audio file, convert it into a modifiable format, modify it, and then reformat it into the original format. If this is applied again and again to a lossy format, then quality of the recording will degrade with every modification. To avoid this situation a lossless audio type such as .WAV should be used.

For the prototype, PyAudio will be used to record and output the actual audio files. This uses the standard python audio modules. Python supports a small range of audio formats, including .WAV. .WAV files can be included in a program by simply using ‘import wave’. Other formats can be imported into Python, however the focus is to simply implement the core functionality in the form of a prototype, so .WAV will be sufficient. If a final version of the system was being developed, where handling audio files was time-critical, then a lossy format such as .MP3, may have been picked. The system is being implemented to aid collaboration through audio, meaning that audio quality needs to be guarded to an extent.

## Data Storage

There two types of data that the system will have to store - text and audio. This section discusses the techniques and tools that help implement effective storage solutions.

The system is going to handle multiple users, each user will have an account, they may have a session associated with them, they may be involved in another user’s session, they will have recordings associated to them, etc. Complex and relational data is best stored in a database. A database, such as MySQL sets up the initial infrastructure so that data can simply be organised into tables. User details, session data, associated audio file locations can all be accessed in a structured manner.

Depending upon the implementation of the prototype, there may be no need for a database as the amount of users may be so low. In this case data could simply read in from text-based formats such as JSON.

In either case the server will need to also store audio files. The method that will be implemented will make use of the default file system. This makes sense as the file systems found on modern operating systems have already been extensively optimised.

## Data Exchange & Data Interchange Formats

Data interchange formats provide a structured method of reading and writing data. If a structured collection of information needs to be passed around a system then ideally a data interchange format will be used. A data interchange format, such as JSON, essentially provides a standard method of structuring data. Simple textual data could be used to pass simple information, however problems could soon arise if the amount of information grows.

Consider this scenario in a system such as this one. A client device wishes to view or download a ‘session’ (a collection of recordings made by various users). This will mean that some form of information will need to be passed from the server to the client device. Information such as the session name, duration of the session, users in the session, etc. will need to be transferred. If we implemented a system where plain textual information is exchanged between the client and the server, then we have to establish a common layout of information so that A) the server can create it, and B) the client can read it. This is exactly what data interchange formats provide, a scheme, so that textual data can be written and read efficiently.

### Python & JSON

As Python will feature heavily, it makes sense to use JSON, as Python uses similar conventions (Json.org, n.d.). JSON data handling is part of the standard Python library, providing access to essential decode and encode functionality. JSON is extremely readable, especially compared to other data interchange formats. Another reason why using JSON makes sense is because it basically follows the same syntax as some of the data structures used within Python. JSON entries can easily be accessed as Python lists and dictionaries, as shown below.

|  |
| --- |
|  |
|  |
| Python & JSON |

### FTP

As stated earlier XAMPP provides an FTP server. It makes sense to use this as the default method of transferring files within this system. Python comes with a FTP library that allows you to perform a variety of operations. In the proposed setup an ftp server that is packaged in XAMPP is used - proFTPD. Below shows how a file can be transferred via FTP using IDLE and the ftplib module in Python.

|  |  |
| --- | --- |
|  |  |
|  |  |
| Before File Transfer | After File Transfer |
|  | |
|  | |
| IDLE Commands | |

# Concurrency Considerations

This section explores concurrency, the issues associated and how it may be implemented into this system. This is relevant to the development of the end deliverable as server/client functionality is often implemented using concurrency, through the use of processes and threads.

The system potentially requires a lot of computing and it needs results within a time constraint. These are the traits of a system that would benefit from concurrent implementation. Bacon (1997) describes that the applications that benefit most from concurrency require a large amount of computing and have a need for a result to be completed within a window of time.

A potential concurrent implementation can be identified before the system is developed, as processes that run side by side independently. Rumbaugh et al (1991) describe concurrency as two objects that receive events simultaneously, but don’t interact with each other. A possible element within the system that could apply concurrency could be how the server handles client sessions. Each client could run concurrently, possibly creating a more practical and efficient system.

## Processes & Threads

Operating systems can concurrently execute programs through the use of processes and their lightweight counterparts – threads. A process can be loosely defined as a program in execution. A processor can only handle a single process at a time, this is partly why concurrency is implemented. Processes have three typical states that control their execution:

|  |  |  |
| --- | --- | --- |
|  | |  |
| RUNNING | Currently being executed by the processor. |
| RUNNABLE | Waiting for scheduling so a processor can run it. |
| BLOCKED | Unable to run until a certain event happens. |
|  | |

These states allow us to control when certain processes are running. More specifically, they provide the ability to ‘pause’ a process so that higher priority processes can be run instead. This is exactly what this system would benefit from - if multiple clients were being handled in this fashion then even distribution of resources could be applied to every client.

Threads could be viewed as lightweight processes. A process can create multiple threads, allowing it to breakdown its execution into small concurrent sections. This is also known as multithreading. Multithreading can be used for servers that need to handle clients simultaneously. Silberschatz (2012) explains that a server could utilise multithreading by creating a thread for listening to client requests and then create extra threads for handling requests for each client. A potential idea is shown below.

Client

Server

New Thread

1. New client makes a request

2. New thread is created to deal with request

3. Server continues listening for other clients

## Issues Regarding the Implementation of Concurrency

With a slow, steady stream of client requests the system above may be a satisfactory method of handling client threads, however in reality, client requests can fluctuate. Implementing such a system makes the server quite inflexible, as it isn’t able to scale for increased client requests.

Web servers use resource pools to accommodate for unpredicted surges of traffic. Holden (2002) explains that web servers create child processes beforehand to avoid real-time start up overhead. This is a viable option, however separate processes are unnecessary for this system’s application, a more efficient system would make use of threads, as they are quicker to create. Silberschatz (2012) describes processes as more time-consuming and harder to manage than threads. To get around this problem, Holden (2002) continues to explain that a server could be implemented as a single process that handles each client within individual threads. This would lower the overhead for per-request operations, however coupled with a pooling technique could become even more efficient.

Client

Server

Thread

1. New client makes a request

2. Thread is retrieved from pool to deal with request.

3. Server continues listening for other clients

A multithreaded approach would bring the most efficiency however there are other issues that come with threads and processes, such as synchronisation.

### Synchronisation Issues

Synchronisation is needed in the system to manage how multiple clients will be accessing shared data. Consider the diagram below, it shows how a lack of synchronisation may affect the system if a shared audio file was altered while another client was unaware.

|  |  |  |
| --- | --- | --- |
| Client 1 | Shared audio file on server | Client 2 |
| Download file | file - unaltered |  |
| Make x changes to local file | file - unaltered |  |
| Upload filex | file - unaltered | Download file: file |
|  | filex - 1 alteration | Make y changes to local file |
|  | filex - 1 alteration | Upload filey |
|  | filexy? - 2 alterations – client 2 only expected 1 alteration |  |

The problem with the scenario above is that client 2 is editing a version of the file that no longer exists on the server (the orange section). By the time client 2 uploads their modified file, client 1 has already made changes to the original file. This is an issue because client 2 may now not want to apply their changes in addition to the changes made by client 1. In the system this could result in undesirable outcomes for users, or even worse, corrupted data.

This situation could be described as a race condition - two threads of control (clients) that are trying to access and manipulate shared data. A way of avoiding this situation has already been discussed earlier in the report (see Jamly). Alternatively I could use approaches regarding critical sections. A critical section can be identified as a part of the code that could potentially create race conditions. According to Silberschatz (2012, pp.206), a critical section must be handled so that no two processes can be executing code within their respective critical sections at the same time. This basically means that before a process/thread enters its critical section, there must be a check so that no other process/thread is in its critical section.

To implement such a system, locking mechanisms such as semaphores, and mutex’s must be used. Locking mechanisms generally solve synchronisation issues in the same way, by protecting critical regions with controlled access between processes. Semaphores and mutex’s provide mutual exclusion between processes, allowing only a single process to be within their critical section at a time. A mutex lock has two atomic states - locked and unlocked. A mutex lock could be shown as follows:

|  |
| --- |
| do{  acquire lock  critical section  release lock  rest of code…  } while (true) |

The problem with the implementation shown above is that it creates a busy waiting scenario. While other processes wait to gain access to the critical section, they are constantly cycling through a loop, using system resources. The disadvantages of using this method are concerned as negligible in systems where the waiting period is very small. The proposed system that is being developed is likely to be dependant upon user input; therefore the waiting period could range vastly.

Another issue with this system is that there is no control over waiting processes. Semaphores can be implemented as either binary or counting semaphores. Binary semaphores are very similar to mutex’s, however counting semaphores are used to control how many processes have access to a critical region. Semaphores typically use wait() and signal() to control access.

Semaphores are capable of placing waiting processes into a FIFO queue, this could help alleviate the busy-waiting issues. Processes put into the waiting queue would be put into a blocked state, until they are ejected from the queue and put into a ready state. Processes would no longer be constantly checking when the critical region is accessible. This still isn’t a perfect solution, Silberschatz (2012, pp.217) explains that semaphores, even with a waiting queue, could put processes in a position where they are waiting indefinitely. This is also known as deadlock. An example of this is shown below, where two processes have to access two separate semaphores.

|  |  |
| --- | --- |
| Process 1 | Process 2 |
| wait(A) | wait(B) |
| wait(B) | wait(A) |
|  |  |
|  |  |
|  |  |
| signal(A) | signal(B) |
| signal(B) | signal(A) |

The diagram shows how neither process will get to the ‘signal’ instructions, as they are trying to gain access to a semaphore that is being used by the other process. Deadlock avoidance and deadlock detection are methods of handling deadlocks.

Deadlock avoidance works by controlling the how resources are requested. It is difficult to implement, using algorithms such as the Bankers algorithm.

Deadlock detection works on the basis of allowing deadlocks and then recovering from them. Burns and Davies (1993) explain that deadlock detection permits processes to get into deadlock. According to Silberschatz (2012) in deadlock detection there are two main algorithms, an algorithm to check if a deadlock has occurred and an algorithm to recover from the deadlock.

An algorithm has to be used periodically to detect deadlock in a system. Deadlock recovery can then be achieved by either killing every single deadlocked process, or killing deadlocked processes one-by-one until the deadlock dissipates.

# Potential System Design

Taking all of the research into account a possible implementation method has been reached. It makes sense that the first prototype of the system will not make use of a GUI, and will instead provide functionality through a CLI. The focus will be on the core functionality of the system – essentially server development. In later prototypes, existing functionality of the system can be built upon, by adding less technical elements such as a GUI.

The next main task is implementing a server that can handle clients and sessions concurrently. Research has already shown that this is possible and the server framework, ‘cherryPy’, supports multithreading.

Another task that will need completing is the underlying code. Collaborative systems and ‘sessions’ are all necessary to implementing an initial prototype.

The diagram below gives a brief overview of how the system will operate.

|  |
| --- |
| User  Client  Actions  Server  Actions  Create Session  Upload Session  Select Session  Record Audio to session  Process  & Store  Retrieve Session  Store |

# Conclusion

Throughout this report various technologies and platforms related to the implementation of the system have been researched. The research has been quite effective in informing certain design decisions, as the consequences of implementing almost every proposed solution has been explored.

With the current knowledge gained from this research, a basic prototype of the system can now be developed. This report shows the effect that the research has had on the design of the system. Key technologies would never have been considered without this research and the benefits of certain technologies, such as, concurrency or Python, would never have been discovered.

The potential pitfalls in development have also been found. An iterative development approach has been chosen – which as already discussed, is ideal for this type of project. However, the project would have continued using the waterfall process, if research hadn’t discovered iterative development, potentially causing undesirable results.

An unexpected discovery made, is that high-level implementation doesn’t mean a lack of flexibility. In terms of Python, this couldn’t be further from the truth. A large amount of time will hopefully be saved on implementing low-level code.

Overall, a more efficient design and development approach has been gained from this research. Key findings include:

* Periods of development should be a lot quicker, thanks to the use of high-level tools such as Python.
* Iterative development should make the final deliverable more robust.
* Implementation of concurrency should make the deliverable more efficient.
* Open source technologies such as SoX will provide some core functionality.
* Development environments will increase development productivity.

This is by no means the end of research, just merely the end of the initial phase. There are many topics that will still need to be researched in order to implement a full-fledged system. Human computer interaction and GUI development are just two topics that will need to be researched as development moves into the realms of client development. As the project progresses, there may be unforeseen complications, which may hinder development. This means there will always be opportunities for more research. The contents of this report provide a solid starting-point to create an initial prototype design and begin development.

# References

Apachefriends.org, (2015). *XAMPP Installers and Downloads for Apache Friends*. [online] Available at: https://www.Apachefriends.org/index.html [Accessed 13 Nov. 2015].

Audacityteam.org, (2015). *Audacity: Free Audio Editor and Recorder*. [online] Available at: http://audacityteam.org/ [Accessed 7 Nov. 2015].

Bacon, J. (1997). *Concurrent systems*. 2nd ed. Harlow, England: Addison-Wesley.

Beazley, D. and Jones, B. (2013). *Python cookbook*. 3rd ed. Sebastopol: O'Reilly Media.

Burns, A. and Davies, G. (1993). *Concurrent programming*. Wokingham, Eng.: Addison-Wesley Pub. Co.

Cherrypy.org, (2015). *CherryPy — A Minimalist Python Web Framework*. [online] Available at: http://cherrypy.org/ [Accessed 12 Nov. 2015].

Collins-Sussman, B., Fitzpatrick, B. and Michael Pilato, C. (2015). *Version Control Basics*. [online] Svnbook.red-bean.com. Available at: http://svnbook.red-bean.com/en/1.6/svn.basic.version-control-basics.html [Accessed 3 Nov. 2015].

Cordova.Apache.org, (2015). *Apache Cordova*. [online] Available at: https://cordova.Apache.org// [Accessed 4 Nov. 2015].

Cotterell, M. and Hughes, B. (2006). *Software project management*. 4th ed. London: McGraw-Hill Education.

Driscoll, M. (2015). *Python music player*. [image] Available at: http://www.blog.pythonlibrary.org/2010/04/20/wxpython-creating-a-simple-mp3-player/ [Accessed 11 Nov. 2015].

Git, (2015). *Build software better, together*. [online] Available at: https://Git.com/ [Accessed 6 Nov. 2015].

Git-scm.com, (2015). *Git*. [online] Available at: https://git-scm.com/ [Accessed 18 Nov. 2015].

Holden, S. and Beazley, D. (2002). *Python Web programming*. Indianapolis, Ind.: New Riders.

Jamly.co, (2015). *Jamly - Free Online Music Sharing & Collaboration*. [online] Available at: http://www.jamly.co/ [Accessed 4 Nov. 2015].

JetBrains, h. (2015). *Python IDE & Django IDE for Web developers : JetBrains PyCharm*. [online] Jetbrains.com. Available at: https://www.jetbrains.com/pycharm/ [Accessed 15 Nov. 2015].

Json.org, (n.d.). *JSON*. [online] Available at: http://www.json.org/ [Accessed 4 Nov. 2015].

Koepke, H. (2015). *10 Reasons Python Rocks for Research (And a Few Reasons it Doesn’t) — Hoyt Koepke*. [online] Stat.washington.edu. Available at: https://www.stat.washington.edu/~hoytak/blog/whypython.html [Accessed 9 Nov. 2015].

Kompoz, (2015). *Kompoz Music Collaboration*. [online] Available at: http://www.kompoz.com/music/home [Accessed 3 Nov. 2015].

Lutz, M. (2011). *Programming Python*. 4th ed. Sebastopol, CA: O'Reilly.

Mobilehtml5.org, (2015). *Mobile HTML5 compatibility on iPhone, Android, Windows Phone, BlackBerry, Firefox OS and other mobile devices*. [online] Available at: http://mobilehtml5.org/ [Accessed 5 Nov. 2015].

Modpython.org, (2015). *mod\_python - Apache / Python Integration*. [online] Available at: http://modpython.org/ [Accessed 11 Nov. 2015].

People.csail.mit.edu, (2015). *PyAudio: PortAudio v19 Python Bindings*. [online] Available at: https://people.csail.mit.edu/hubert/pyaudio/ [Accessed 12 Nov. 2015].

Phonegap.com, (2015). *PhoneGap | Home*. [online] Available at: http://phonegap.com/ [Accessed 5 Nov. 2015].

Piotrowski, P. (2006). *Build a Rapid Web Development Environment for Python Server Pages and Oracle Database 10g*. [online] Oracle.com. Available at: http://www.oracle.com/technetwork/articles/piotrowski-pythoncore-084049.html [Accessed 7 Nov. 2015].

Portaudio.com, (2015). *PortAudio - an Open-Source Cross-Platform Audio API*. [online] Available at: http://www.portaudio.com/ [Accessed 4 Nov. 2015].

Prechelt, L. (2000). An empirical comparison of seven programming languages. *Computer*, 33(10), pp.23-29.

Python.org, (2015). *Welcome to Python.org*. [online] Available at: https://www.python.org/doc/essays/comparisons/ [Accessed 9 Nov. 2015].

Rossum, G. (2015). *Foreword for "Programming Python" (1st ed.)*. [online] Python.org. Available at: https://www.python.org/doc/essays/foreword/ [Accessed 13 Nov. 2015].

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991). *Object-oriented modeling and design*. Englewood Cliffs, N.J.: Prentice Hall.

Silberschatz, A., Galvin, P. and Gagne, G. (2012). *Operating system concepts*. 9th ed. USA: John Wiley & Sons.

Software.intel.com, (2015). *Intel® XDK | Intel® Developer Zone*. [online] Available at: https://software.intel.com/en-us/intel-xdk [Accessed 6 Nov. 2015].

Sommerville, I. (2011). *Software engineering*. Boston: Pearson.

Sox.sourceforge.net, (2015). *SoX - Sound eXchange | HomePage*. [online] Available at: http://sox.sourceforge.net/ [Accessed 4 Nov. 2015].

Splice, (2015). *Splice*. [online] Available at: https://splice.com/ [Accessed 5 Nov. 2015].

Surina.net, (2015). *SoundTouch Sound Processing Library*. [online] Available at: http://www.surina.net/soundtouch/ [Accessed 5 Nov. 2015].

Wxpython.org, (2014). *wxPython*. [online] Available at: http://www.wxpython.org/ [Accessed 10 Nov. 2015].