SCHOOL OF COMPUTING
Faculty of Engineering


# Google Glass for Training and Education

By Gurpreet Paul

2015/2016

Submitted in accordance with the requirements for the degree of Computer Science (BSc).

The candidate confirms that the following have been submitted:

| Deliverable | Recipient | Date |
|---|---|---|
| Report | SSO | 11/05/2016 |
| Glass app code | Supervisor + Assessor | 11/05/2016 |
| Android app code | Supervisor + Assessor | 11/05/2016 |
| Server code | Supervisor + Assessor | 11/05/2016 |
| Video file | Supervisor + Assessor | 11/05/2016 |

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

Type of project **Exploratory Software** Signature

# Summary

This report will discuss the development of the architecture behind a unique telemedicine system that uses Google Glass. Furthermore, this report will outline where Google Glass fits into telemedicine and the drawbacks that it has, providing the reader with a glimpse into the world of medicine and a more informed look at Google Glass.

This report shall have a focus on medical uses of Google Glass' technology, however the reader must note that the topics discussed in this report are applicable to any field.

# Acknowledgements

I would like to thank my project supervisor, Andy Bulpitt, for on-going support and valuable advice throughout the project.

A thank you also goes to Taj Hassan and Victoria Ward, both doctors, whose time and feedback provided me with a lot of insight into the world of medicine.

A big thank you goes out to my family and friends for putting up with my lack of presence while writing this report!

# Table of Contents

# 1. Introduction

Google Glass (depicted in Figure 1.1.0), from Google Inc., was released in February 2013. Since its inception, there have been numerous applications of its hardware and software, particularly in a medical context and particularly within the United States of America (Armstrong, 2014; Glauser, 2013).

*Figure 1.1.0 - Google Glass running XE22*

Google Glass (this report shall refer to Google Glass interchangeably as Glass henceforth) was officially intended for use as a consumer device and it had been marketed as such by Google themselves but its main benefits were found in an enterprise setting. Google Glass found its way into the public through the "Explorer" programme. The "Explorer" programme was a group of select people called "Explorers" who were to trial Google Glass in advance of its release (Hashimoto, Phitayakorn, & Castillo, 2015). It is unknown exactly how many Google Glasses went into medicine related activities.

Google Glass has voice recognition built into it, providing hands-free interactivity. Using voice commands like, "Okay Glass, take a picture", or, "Okay Glass, record a video", one can take a picture or record a video. The camera is partly where issues of privacy arose as another person would not be aware if a person wearing Glass was recording them or not. As such, groups like "Stop the Cyborgs" were vocal against Glass because of the privacy issues surrounding Glass (Stop the Cyborgs, 2016). See section entitled Privacy and Security (2.3.3) for more information.

Google Glass has a touchpad which allows a user to swipe through different panels called 'cards'. These cards can show you the weather, your recent emails and even show you directions around a city. All of this is fed back to the user through a small display in the upper right corner of the user's field of vision.

On January 19th 2015, Google officially stopped selling Google Glass and closed the Explorer programme (BBC, 2015; McGee, 2015). Google announced that it was working on an

"Enterprise Edition" of Google Glass under the name "Project Aura" (Swanner, 2015; Khan, 2016; Charlton, 2016). Although this means that initial version of Glass is no longer actively maintained by Google, it is still in use by people all over the world. An example of one such use is within a medical context, namely telemedicine.

Telemedicine, in its most general form, is defined to be medicine over the Internet. There are many benefits of using Glass to facilitate telemedicine: it's unobtrusive, it's hands-free, it can help people in remote locations, etc. (TD & TL, 2015).

In the medical field Google Glass has been adopted widely because of its technology. The hands-free aspect, combined with its camera and WiFi capabilities, make it a powerful tool to train and educate people. For example, a surgeon could theoretically record an operation and a medical student could watch that recording all the whilst taking notes in preparation for medical examinations.

Those reasons are why this report will discuss the development of a unique telemedicine application with a focus on Google Glass.

## 1.1. Problem definition

The core problem to solve is implementing a system that delivers audio and video from a person who is located remotely to Glass via the Internet, and for the Glass user to deliver a stream of Glass' camera and audio back to the observer.

To simplify, the problem to be solved is to make **bidirectional communication** between Glass and another 'smart device' **work over the Internet**.

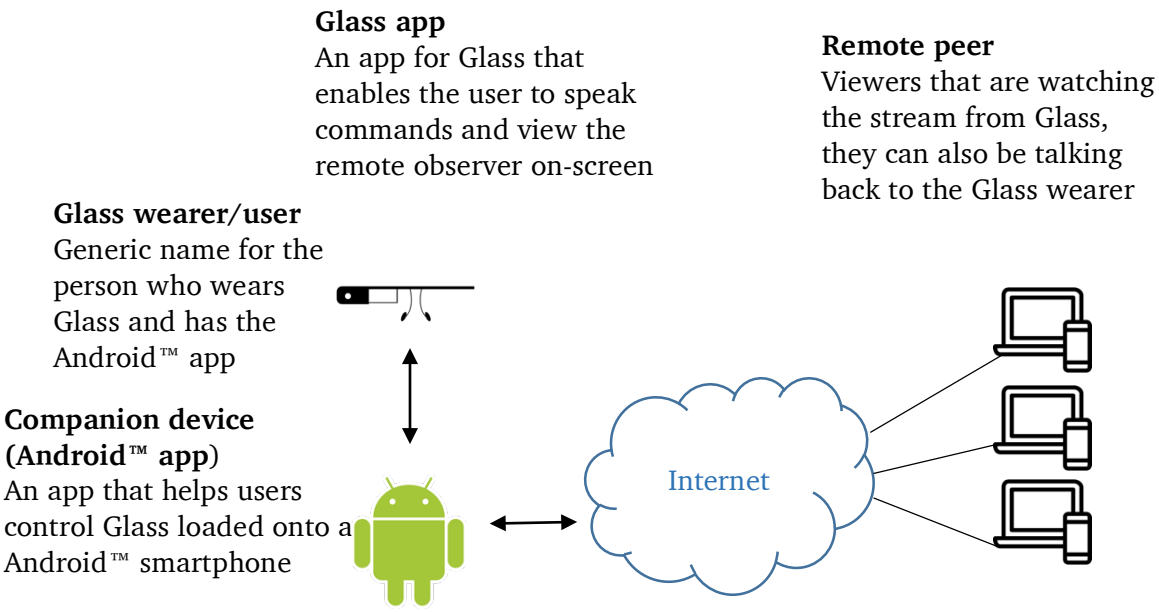



*Figure 1.1.1*

A quick summary of the diagram: A **remote peer** will have a 'smart device', such as a smartphone or tablet, that can stream a video capture of herself through the Internet to the **Glass user**, and the **Glass user** will have a specially developed **Glass app** for him to stream the video capture from Glass back to the remote peer.

The **companion device** is the bridge to the Internet. The **companion device** essentially makes it easier for Glass to communicate with the **remote peer** and also to control Glass' stream.

The above diagram will be referred to throughout the report.

In order to solve the problem, this report shall focus on creating an Android app[1] for the Glass user to stream through and a Glass app for the Glass user to view a stream in its prism display. This report shall also discuss the steps that led up to the final solution and why the stream would have to be **confidential, secure and speedy**. This would transform Glass into an invaluable tool in providing training and education to all sorts of people.

## 1.2. Problem clarification

In order to avoid confusion this report will define what it means to be a remote peer and define what it means to be a Glass user. The remote person may be a medical consultant providing advice to the Glass user and the Glass user may be a student listening to and watching the feed from the medical consultant in Glass' prism display. Or, due to the flexibility of the project, the roles may be reversed: The Glass user may be the medical consultant providing a live feed of what she is doing and the remote peer may be a student diligently taking notes while he is watching the feed from the medical consultant on his smartphone.

On a similar note, to "provide training", "to train" or "to educate" can mean a variety of things within this report and can cause confusion if not clarified. The following list provides various scenarios to aid clarification:

- A medical consultant guiding a Glass wearer through voice and/or video on the actions to perform. For example, the consultant may say, "Check if the patient is breathing," then the Glass wearer promptly performs the action. In this instance the Glass wearer learns by doing.
- The Glass wearer could be in a live surgical operation and therefore detailing what he or she is doing to a patient. This would be a great learning tool for the remote peer(s) as they will learn from the actions of a professional.
- The Glass wearer could simply record what he or she is doing ready for playback at a later date. This is a superb educational tool in itself as the Glass wearer would be able to go over what they did and critically evaluate every aspect of their actions.

## 1.3. Objectives

Combining the ideas from the above, this report can summarise the requirements into 5 objectives:

1. Develop a Glass app to communicate with an Android app. This app will have to successfully display a stream of a remote peer in its display.
2. The Android app mentioned in the above will also need to send streams to a remote peer.

[1] Due to time constraints this report will focus on creating an Android app, but any application on any operating system can be used, as discussed later in this report.

3. Create a server that can broker the connections from observers to Glass users.
4. Test the system.
5. Receive feedback on the system and outline the ways it can be improved.

If the reader is confused on the terminology above then Figure 1.1.1 is recommended to be reviewed. Also for meeting notes regarding the objectives see Appendix E: Initial Objectives.

## 1.4. Motivations

There are two motives for this project:

1. One motivation stems from a need to deliver training to medical students in a secure and confidential manner. To give more context surrounding this, a study was conducted involving 27 plastic surgeons who were viewing a live stream from Glass, 96.3% of plastic surgeons thought that the live transmission from Glass was a good or excellent learning experience (Denis Souto Valente, 2015). The study concludes by stating that, "[telemedicine] can positively impact health care delivery, medical documentation, surgical training and patient safety," which means that using Glass within telemedicine will make a worthwhile contribution to the training of future medical workers, of whom, consist of the following: student or trainee doctors, physicians and surgeons.

2. The other motivation for this project is to see if the streaming of Glass to another device (a computer, a tablet or a smartphone) can actually be done. As such, similar implementations of this project already exist, however exact details of those implementations are difficult to find. Furthermore, this project has a focus on mentoring which is different from most of the implementations anyway. Plus, they only focus on one way broadcasting of audio and video and not in a bidirectional manner like this project aims to do.

There have been some studies in the US that have developed similar sorts of software with Glass and students generally agree that using Glass is a much needed benefit (P Russel, 2014; Benninger, 2015). In one study in 2013 (Tully, Dameff, & Kaib, 2015), 30 second year medical students used Google Glass to record a first person standardised patient encounter (individuals trained to act like a real patient). In a survey, given at the end of the standardised patient encounter, 77% students stated that Glass had a neutral to positive experience. In a follow up survey, 70% of students responded that Glass is "worth including in the [clinical skills program]." Additionally, there have been cases where use of similar sorts of software would be of benefit to senior physicians, not just students (Chai, Babu, & Boyer, The Feasibility and Acceptability of Google Glass for Teletoxicology Consults, 2015; Chai, et al., 2015).

## 1.5. Potential impact

The potential impact of Google Glass for use as a training or education tool is huge. The biggest demonstration of Glass' impact was performed by Mr. Ahmed, a colorectal consultant, who broadcasted live an extended right hemi-colectomy and partial liver resection to over 13,000 surgical trainees and students in over 115 countries (TD & TL, 2015). Questions were also sent to Mr. Ahmed live. He could view them in the bottom left hand side of Glass' screen which allowed him to provide answers to them in real-time (Lee, 2014).

In a separate study at the University of Massachusetts Medical School, ER (Emergency Room) physicians wearing Glass evaluated poisoned patients while a live video feed was sent securely to a toxicology supervising consultant (Chai, Babu, & Boyer, The Feasibility and Acceptability of Google Glass for Teletoxicology Consults, 2015). Toxicology, is a branch of science concerned with studying a poison's nature and effect (Merriam Webster, 2016). The ER physicians received text-based messages from the consultant in Glass' display which guided them on what to do. The consultant could also ask for pictures of medication bottles, electrocardiograms and other pertinent information if allowed. As a result of this, the ER physicians reported greater confidence in diagnosing people with poison. What's really excellent about this is that **6 of those patients received antidotes they wouldn't have got if the physicians weren't using Glass**. 89% of the cases that the ER physician showed to the consultant toxicologist were considered success.

As the reader can imagine, the potential of Glass' impact in medicine and as a telemedicine tool is massive.

## 1.6. Degree relevance

This project requires an understanding of these concepts in Computer Science:

- There is much discussion on networking and protocols. TCP, UDP, RTSP, HTTP, WebSockets and video streaming are some examples of topic of discussion. These have relevancy with Networks and Scalable Architectures (COMP2444) and Computer Systems (COMP1440).
- Distributed Systems (COMP3900) gave much needed insight to client-server architectures and also cloud computing. These topics were important when choosing the right server type for this project.
- There are also concepts from Graphical User Interfaces (COMP2542) and Mobile Application Development (COMP3222) when designing the Android apps.
- Software Engineering (COMP2541) played a huge role in the development of this project. This project used a Scrum/Kanban style methodology.

# 2. Background Research

This section explains, in detail, what research has been conducted prior to this report being written. It will give the reader an insight into the significant underlying complexities of Glass and how this research project aims to combat them. There's much discussion on the benefits and drawbacks of using Glass. Moreover, before the final implementation, a look at other existing systems of a comparative nature will be taken into consideration.

Due to the nature of this project there are many names being used to define certain aspects of the whole system. So, to make it easier for the reader, this report will follow the naming in the diagram given in Figure 1.1.1.

## 2.1. Hardware

Google Glass has a head mounted display. The display is, as Google describes, "equivalent [to looking at] a 25-inch high definition screen from eight feet away." (Google, 2014). With a camera providing up to 720p (high definition) quality and photos that can manage 5 megapixels, Glass can record and take pictures in great detail.

Glass has a WiFi module capable of 802.11 b/g 2.4GHz connectivity. This is ample enough speeds for transference of data. However, the wireless standard **n** has been subsequently released and then later on, **ac** was released. Both **n** (up to 450mbs) and **ac** (up to 1300mbs on 5GHz) provide much faster speeds than that of **b** (11mbs) or **g** (up to 54mbs). This means that Glass has room for improvement on its WiFi speeds, however due to its size, Glass may not be able to fit the antennas **n** or **ac** require.

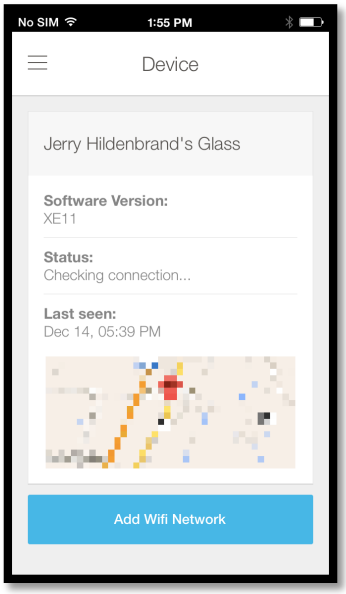



*Figure 2.1.1 – The MyGlass app on iOS (Hildenbrand, 2013)*

Glass has a Bluetooth 4.0 LE module that can be used to share files to/from Glass. It is with the Bluetooth module that the MyGlass app communicates with. The MyGlass app, developed by Google, provides anyone who has purchased Glass a way of configuring Glass.

This is useful because Glass does not have a keyboard for manual entry. To use the MyGlass app users need a Google account. Once signed into MyGlass, setting up Glass is a few taps. MyGlass also is available online where a user can enable or disable various services like Google Now, Google Books, YouTube, etc. One of many things that the MyGlass app can do is enable Glass to connect to a WiFi network (see
Figure 2.1.1 – The MyGlass app on iOS). MyGlass can also unlock, reset Glass and backup media such as photos and videos (Google, 2014).

Audio is built into Glass and can be emitted from Glass too. Glass has a storage size of 12GB which is useful for short clips of high definition recording but for longer recordings this space becomes easily filled. Glass' media is synced to Google Cloud however users require a valid Google+ account to use this feature.

The battery is stated to last a typical day with light usage. With medium WiFi usage battery life depletes quicker.

Glass has a touchpad that can be used to navigate its menu screens. Common actions on the touchpad include: swiping left, right and down.

## 2.2. Software

Glass runs on a special add-on to the Android SDK called the Glass Development Kit, or GDK (Glass Developers, 2016). Software developers who have developed with Android also know that there is also an API number which corresponds to the Android version number. It is generally the case that as the API level increases so does the number of features in that version of Android.

Shown below in Table 2.2.1 is a list of relevant Android names, API numbers and version numbers.

| Android Version | Released | API Level | Name |
|---|---|---|---|
| **Android 6.0** | Aug-15 | 23 | Marshmallow |
| **Android 5.1** | Mar-15 | 22 | Lollipop |
| **Android 5.0** | Nov-14 | 21 | Lollipop |
| **Android 4.4 - 4.4.2, GDK** | Oct-13 | 19 | KitKat |
| **Android 4.3** | Jul-13 | 18 | Jelly Bean |
| **Android 4.2 - 4.2.2** | Nov-12 | 17 | Jelly Bean |
| **Android 4.1 - 4.1.1** | Jun-12 | 16 | Jelly Bean |
| **Android 4.0.3 - 4.0.4** | Dec-11 | 15 | Ice Cream Sandwich |

*Table 2.2.1*
*Modified from Xamarin (Xamarin, 2016)*

The GDK has special modifications that enable software engineers to become productive with Glass. It has a special Gesture Detector that detects swipes and gestures accordingly. It has pre-built Live Card layouts (see Figure 2.1.1 and Figure 2.2.2) that provide consistency to the design of Glass apps. Aside from those modifications, developing for Glass is the same as developing for Android.


*Figure 2.2.1*
*Main menu items (Glass Developers, 2016)*

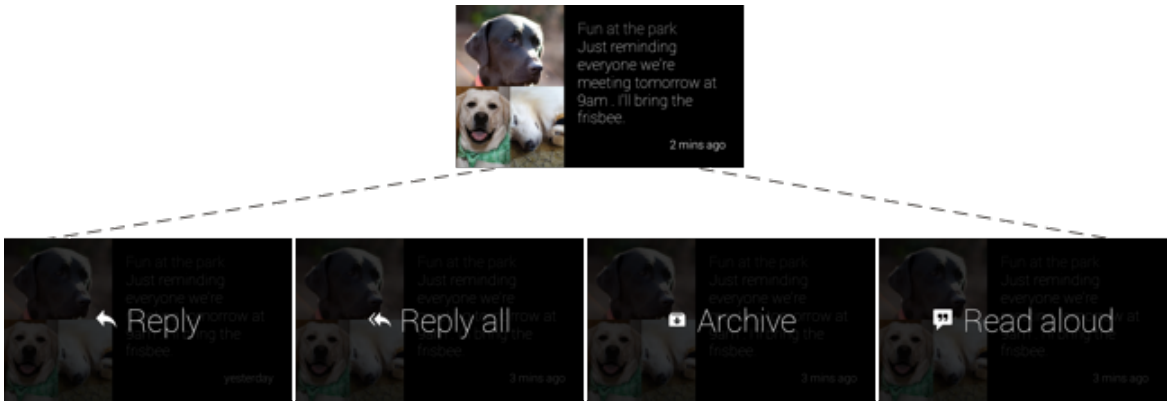



*Figure 2.2.2*
*Menu selection (Glass Developers, 2016)*

Google Glass' updates were publically known as XE (short for "Explorer Edition"). Over the course of about two years, the team behind Google Glass shipped many updates, the bulk of which were done in 2014 (Glass Almanac, 2016). The latest version update that was rolled out before Glass was decommissioned was XE22.

## 2.3. Limitations

The project had many difficulties to contend with. For the reader to understand why this project was a technical difficulty, each of Glass' difficulties, or limitations, will be discussed in this subsection. In the section titled Implementation this report will aim to develop software that will get around these problems.

### 2.3.1. General

The battery life is one of the major limiting factors when using Glass, especially when doing highly intensive work such as video streaming. It has a battery capacity of 570mAh.

A journal entry points out that Glass' battery life span is approximately 45 minutes when recording (Glauser, 2013). This raises concern over Glass' ability to stream for extended periods of time. That figure, put within the contexts of this report, implies that battery optimisation needs to take place when developing for Glass. As discussed in section 2.2,

Glass' operating system primarily works on a software level so memory management needs to be looked into when developing a video application for Glass.

The price is also a limitation as Glass has a starting price of £1000 in the UK and $1500 for the US (Martin, 2015). The high price tag is a barrier of entry for many would-be users of Glass.

Another limitation exists in terms of software. As Glass essentially runs on Android API 19 and Glass has not been updated since, software engineers working on Glass will not have the features of API 21 and above to work with. When implementing the project (see section 4), some features such as the updated WebView would have helped saved time rather than going to the effort of integrating a third-party library into the implementation.

### 2.3.2. Hardware

The first ever study on the potential of Glass as a telementoring device recorded that 50% of 34 surgeons rated the video quality as bad to poor when compared to Apple's iPhone 5 (Hashimoto, Phitayakorn, & Castillo, 2015). These findings may not be a useful comparison as the iPhone had a video quality of 1080p versus the Glass with a quality of 720p. Glass used Hangouts and the iPhone used FaceTime which meant that both pieces of software could have performed optimisations to the video quality that were undiscovered during the study.

A more useful comparison would have been to have similar kinds of devices – the iPhone 5 has 1GB of RAM whilst the Glass has 2GB. This may seem unfair to the iPhone but fundamentally Apple has optimised the hardware of its iPhone to work efficiently with that 1GB, Apple even has its own processor to work with that 1GB. On the other hand, Google's Android works with a massive variety of devices and processors which means the Android OS has had to do the majority of its optimisation on a software level rather than on a hardware level. Comparatively the iPhone has iOS running on it which targets the iPhone range only. It has a larger battery meaning more power can be routed to components of the system.

Not to discredit the study however, as it does show that quality in a telemedicine tool is something that people do focus on.

Glass has been known to overheat with many studies reporting that Glass becomes uncomfortably hot after heavy processing (Tanveer & Hoque, 2014).

In one study, two doctors used Glass to display vital signs in Glass' screen during three procedures. The users reported increased concentration on the task but also reported short battery life and Glass force closures due to overheating (Vorraber, 2014).

Glass has limitations in terms of WiFi. Glass has no support for enterprise networks. So places like large scale businesses, libraries and universities Glass would struggle to connect to a network without programmer intervention.

### 2.3.3. Privacy and Security

One article states that having state of the art technology like Glass would need a cultural/mind shift and that people would need to, "see it, feel it, touch it to understand

how it works and how it will change the game." Further in that article it states that there are invasion of privacy concerns with security being, "top-of-mind for many organisations" (Campeau, 2014).

Another article puts the issues of privacy in terms of facial recognition (Sarpu, 2014). In the article Glass was used for facial recognition for travellers trying to enter the USA. As travellers enter the USA, airport security can use a Glass app to identify people. Though a beneficial tool, it raises numerous issues in terms of privacy. Glass was initially banned in many places including places such as hospitals, banks, concert venues, locker rooms, etc. and still may be banned in many of those places. The film and move industry was very outspoken regarding Glass' entry into cinemas and, they too, banned it (Sherwin, 2014).

As privacy is such a big concern Google has stated that:

> *"Google won't add facial recognition features to our products without having strong privacy protections in place. With that in mind, we won't be approving any facial recognition Glassware at this time."*

This only takes into account Glass apps that go through Google's verification processes. Glass apps can be installed through development methods, meaning that Glass apps with facial recognition (or any other unwanted feature) can be installed without Google's approval.

There are also remarks that there are constraints to Glass' adoption rates in medicine because of privacy regulations and a reluctance by some hospital administrative staff to adopt new technologies (Glauser, 2013).

### 2.3.4. Health and User Experience

One study measured how the right eye's contrast sensitivity was affected whilst using Glass. The purpose was to report how much Glass interferes with normal visual function. It was concluded in the report that Glass produced a significant reduction of visual clarity in the right eye, with a result of greater than 0.5 log units (Longley & Whitaker, 2016). Clarity in the left eye remained unaffected. This is important as attention to detail is needed in many specialisations of surgery. Example specialisations in surgery where visual clarity is needed are: Cardiologists (specialise in dealing with the heart); Pathologists (specialise in studying body tissue for abnormalities); and Ophthalmologists (specialise in the eye).

Previously, this report mentioned a surgical procedure by Mr. Ahmed that was broadcasted to a large amount of people. One of the few limitations that existed when broadcasting was the fact that the glare from the surgical lights sometimes was shown across the viewer's field of vision. Another limitation was that Glass had to be correctly placed in order for viewers to see precisely what was going on. This adds cognitive load to the surgeon's thought process as the surgeon would also have to think about the positioning of Glass. Similar reports were concluded by another study (Hashimoto, Phitayakorn, & Castillo, 2015).

There was also some mention of the fact that the comments shown in Glass were distracting. A minor distraction could lead to some obviously life threatening consequences for the patient. Mr. Ahmed's head movements were also "disorienting for viewers." (Lee, 2014)

The problems identified above did prompt Mr. Ahmed to change subsequent live broadcasts however some problems still remained. A noteworthy quote, especially as it concerns the objectives of this report, was:

> *"Two-way connectivity between users is missing, and there are questions about whether live streamed information will ever be secure enough for routine medical use."*

Doubts expressed as directly as this are a cause for concern. Doubly so when security is involved. Echoing the above quote, Mr. Ahmed said the following about the broadcasted operation:

> *"I had to pull the feed. I didn't know who was watching it."*

This gives support to the fact that anything streamed from Glass would have to be secure, which requires extra computational power to do so. Streaming over the Internet would also mean that there are many points at which a malicious hacker could gain access to the stream. It is true that there are protocols and software to prevent such attacks from happening but Glass may not be able to computationally handle them, especially as Glass is running on an older Android operating system.

Another pain point when using Glass is that a smartphone, or external device needs to communicate with Glass for Glass to be used effectively. For example, the MyGlass app needs to set Glass' WiFi and enable/disable certain features. As Glass does not have a keyboard this reliance on another app to send it data will have to be a necessity.

## 2.4. Alternatives

Google Glass was chosen for this project, but alternatives do exist. To gauge what the alternatives offer, discussion on them will take place in this section.

### 2.4.1. Sony Smart EyeGlass


*Figure 2.4.1*
*(Williams, 2014)*

The Sony Smart EyeGlass (shown above in Figure 2.4.2) was released as a developer edition on March the 27th in 2015 (Sony, 2015). Sony has stated that the Smart EyeGlass' primary use would be within a business context as well as for use as a consumer product. So, unlike

Google Glass which was released as a consumer product, Sony is targeting the enterprise market early on in the product's lifecycle. Sony also wants to get into the AR (Augmented Reality) market by publicising examples of where Smart EyeGlass can be used. One example is that the EyeGlass can provide step by step engine maintenance instructions for workers at a repair shop.

The technology used in the lenses of Smart EyeGlass is monochromatic. This means that the Smart EyeGlass shows text on screen and nothing much else. Its uses are more for pop-up hints and notifications in the screen itself. For that reason, the Sony Smart EyeGlass could not be used for this project as the technology would not be able to meet the initial requirements.

### 2.4.2. Vuzix M100

*Figure 2.4.2*
*(Vuzix, 2016)*

The Vuzix M100 (Figure 2.4.2) was released in late 2013 and offers more features than Google Glass. Whilst Glass offers WiFi a/b which is ample enough, Vuzix offers a greater theoretical transfer speed by having support for WiFi a/b/n. The Vuzix is not fixed to the eyeglass frame like Glass but instead Vuzix opted for a mountable smart glass archetype, which essentially means that it can be mounted on any eyeglass frame.

Other than telemedicine, Vuzix also have a stake in aviation. Through an app called Glass4Flight they provide data insight on pilots and also provide a "Virtual Glass Cockpit" (NewsRx, 2015).

The M100 offers a camera quality of 1080p and 5 megapixel photos, a step higher than the Glass' 720p high definition video quality. The M100 otherwise has all of the features in Glass but at a cheaper price starting at £799.99 (Vuzix, 2016).

The one thing that Vuzix lacked was that it arrived on the wearable market after Glass and so lost out on the marketing advantage that Glass had. So, as a result, Glass became more widely adopted. To compound this further, Google offered Glass for free to a select group of people called "Explorers" which likely further increased the purchasing rate of Glass.

### 2.4.3. Vuzix M300


*Figure 2.4.3*
*(Vuzix, 2016)*

The Vuzix M300 (shown in Figure 2.4.3) is the next version up from the Vuzix M100 (Vuzix, 2016). It offers:

- Improved ergonomics.
- Hot swappable batteries.
- Improved battery life over the M100 (2 – 12 hours).
- Improved display over the M100.
- Intel Atom processor.
- Android 6.0.
- 2GB RAM.
- 1080p quality, 13 megapixel stills.
- Bluetooth 4.1.
- 4 Android buttons.
- WiFi specification and specs of **ac** at 2.4Ghz or 5Ghz.
- Left or right eye support.

It's definitely more technically developed than Glass but readers have to note that Glass was released in 2013 and this newer M300 will be released in the Summer of 2016 at a price of $1499. Vuzix offer an SDK at a price of $499 (Wearable Tech, 2016).

When the M300 is released they will provide fierce competition in the wearable head tech market. In addition to the above, the mere fact that Vuzix are offering an upgrade to the M100 proves that they see potential in smart glasses.

### 2.4.4. ODG R-7 Smart Glasses

*Figure 2.4.4*
*(Osterhout Group, 2016)*

The R-7 targets the enterprise, unlike Glass which was targeted at consumers (Osterhout Group, 2016). The R-7 (depicted above Figure 2.4.4) has a Qualcomm® Snapdragon™ 805 processor which is a powerful 2.7GHz processor aimed at smartphones and tablets (Qualcomm, 2016). The processor is advertised to support 4K video (ultra high definition). ODG have designed their own operating system called ReticleOS™ which is based on KitKat to run on the R-7.

The R-7 has dual screens instead of one. It also seems geared more towards 3D Augmented Reality (AR) with stereoscopic see through displays. Both screens display 720p quality at 80fps.

The camera is positioned in the center of the frame rather than up towards the right, so it has the benefit of effortless correct product placement. The camera records up to a quality of 1080p at 60fps, a step up from Glass' camera quality. Another thing it benefits from is its dual 650mAh battery which means more theoretical run time, but in reality, as the R-7 has power intensive components the actual run time would differ.

On par with the Vuzix M300, the R-7 has Bluetooth 4.1, WiFi speeds up to **ac** and all the relevant accelerometers and gyroscopes.

The price is one of the largest of the smart glasses mentioned here, which as of January 2016, is $2750. This is a much higher price than Glass'.

### 2.4.5. Recon Jet

*Figure 2.4.5*
*(Recon Instruments, 2016)*

Recon Jet (shown above in Figure 2.4.5) is a pair of fitness focused smart glasses aimed at athletes (Recon Instruments, 2016). It has a display equivalent of looking at a 30-inch screen at 7 feet away. The display and the camera are located at the bottom right of the wearer's face, compared to Glass' positioning of the top right. Recon have patented Glance Detection which means that when a Recon user glances at the screen, the screen turns on. Glass has a variation of this built in.

The Recon Jet otherwise has a more aerodynamic design than Glass. Recon have also design the Jet with modularity in mind – the batteries located on either side of the frame can be clipped on and off. The modularity comes at a cost of extra weight.

The Jet is otherwise a direct rival to Glass providing almost all the same features at a lower price of $499. A big caveat within the context of this report, is the display is at the bottom of the wearer's face and this would obstruct a doctor's view.

### 2.4.6. Optinvent Ora 2

Focused on the AR market, the Ora 2 has the technical specifications of Google Glass but with a few notable differences:

- The Ora 2 is capable of WiFi speeds up to **n**.
- It can be worn over "most" eyeglasses.

The price is cheaper than Glass at €699. (Optinvent, 2016)

### 2.4.7. ChipSiP SiME Smart Glasses

Another cheaper alternative to Glass running on KitKat, with the price reportedly being $550 (Andronico, 2015). The features of the SiME Glasses are almost equivalent to Google Glass but record video at up to 1080p quality. (ChipSiP, 2016)

### 2.4.8. Epson Moverio BT-200

Epson have launched into the head wearable market with their Epson Moverio BT-100 for both consumers and businesses. Mentioned here is the BT-200 which is 60% lighter than the BT-100 (Epson, 2016).

Comparing it to Glass there are some notable differences. One difference is that it uses two of the lenses on its frame to deliver HD content, browse the web and utilize smart devices. That is, the displays merge together to become one. Another difference is the BT-200 uses an external attachment to control it. The starting price is $699.99 (Epson, 2016).

The BT-300 is now in the pre-ordering stage and is claimed to have the following:

- 20% lighter than BT-200.
- OLED display.
- HD binocular display at 720p quality.
- 5 megapixel stills.

### 2.4.9. GoPro Hero

The GoPro deserves a mention in this report. The GoPro is an action camera that captures video at a very high quality. The quality of the GoPro (Hero 4 Silver) is 1080p and offers a framerate of 60. Due to this high framerate, the GoPro is suitable for slow motion shots. The GoPro and Google Glass were compared by a group of medical students and faculty members at Stanford University (Paro, Nazareli, Gurjala, & Lee, 2015). After wearing both of them, they found Glass to be more comfortable and easy to use than the GoPro, whilst the GoPro, although cumbersome, offered better battery life and better quality. In the context of this report, a further negative is that live streaming via WiFi from the GoPro seems to be difficult.

### 2.4.10. Virtual Reality headsets

AR (Augmented Reality) and VR (Virtual Reality) have taken the world by storm. With products like the Oculus Rift, Microsoft's HoloLens and the Samsung Gear VR, enthusiasts, particularly gamers, are being treated to immersive worlds and realities. AR/VR headsets typically include greater processing power, especially on the graphics side of the system. This is boon, particularly as this report is concerned with streaming images and data.

The largest drawback is the fact that AR/VR headsets are typically heavy and make the user look like an astronaut. Some headsets also require plugging into a computer which limits the freedom of movement that a Glass user can have.

Until headsets get smaller in size and have a battery life longer than 3 hours, this report will rule out AR/VR headsets as being a viable alternative to Glass.

### 2.4.11. Choice

Due to my supervisor having a pair of Google Glasses handy, this report chose to go with Google Glass. However, after critically evaluating the above smart glasses this report will now turn to deciding which smart glasses would be the most suitable for this report aside from Google Glass.

The choice depends on the time that a person will be purchasing.

- If the purchase date is within 10 years, AR/VR headsets would have reduced in size but would have increased in computational allowance so they would be the best educational tool.
- If the purchase date is within the next 5 years, one of the top smart glasses would be the best choice.
- If the purchase date is within the next year, the Vuzix M300 would be the best choice. The reason for this choice is because Vuzix have improved upon the first iteration of their smart glasses to make this second iteration a very good head wearable.

See the Evaluation section for more information on the choices presented here.

## 2.5. Possible solutions

Before implementing the project, research into similar sorts of existing systems and architectures was undertaken. Taking a look into pre-existing implementations at other businesses proved to be a fundamental exercise as it gave the project technologies to consider.

### 2.5.1. Pristine

Pristine, founded in May 2013, is a USA based startup focused on bringing HD video communication from wearable devices, such as Google Glass, into any kind of field (Pristine, 2016). Pristine had a primary focus in medicine but seem to have transitioned to other fields not related to medicine. This show's the viability of Glass in other fields.

On May 21st in 2014 Pristine received an initial investment of $750,000. Then, on September the 29th in 2014, Pristine received an investment from Amazon S3 Ventures for $5,400,000. The large figures of investment show that there is interest in Glass as a telemedicine tool and its impact.

In their implementation, Pristine took steps to remove the email, text and phone capabilities from Glass to ensure that patient information could not be sent to people outside the system (Strickland, 2014). In Pristine's implementation, Glass contacts a local server to retrieve patient information. This server is on the same LAN (Local Area Network) as Glass so patient's data is never transferred over the Internet and therefore is inherently more secure than implementations that do use the Internet (pioneers.io, 2013). In that same article, Kyle Samani the former CEO further highlights the need for such a telemedicine application by stating that:

> *"We have a lot of demand, far more than we can support at this stage. We've had 10 hospitals reach out to us, and nine or 10 individual surgeons and anaesthesiologists contact us about piloting the apps. I can't run 10 pilots. I simply don't have the manpower."*

In 2014 Pristine implemented another architecture that does use the Internet, "[Glass] streams [audio and video] live through encrypted channels to a remote server, allowing a medical specialist to view them using any computer." This statement alone is useful as it shows that encryption can be achieved with Glass and highlights that the stream from the server can be viewed from any computer. (Strickland, 2014). It also makes true the fact that Pristine eventually had to find secure methods to communicate what was shown in Glass to another device on the Internet.

The same article goes on to mention that in October 2013 Pristine signed up UC Irvine Medical Center as its first pilot customer. This means that Pristine was developing this telemedicine system, with Glass as the focal point, for four months before they started testing it. This provides an insight to the time frame that this project may need.

Further research into Pristine shows that they did a presentation on using WebRTC with Google Glass (Alaniz & Behera, 2014). WebRTC (Web Real Time Communications) is an initiative to enable encrypted sessions between two or more people using simple APIs (Application Programming Interfaces). For more information on WebRTC see section 2.7.5. During the presentation, the two speakers showed their implementation working. The video and audio quality was very good. Thus, WebRTC will definitely be tested within this project to see if the results of Pristine's one year of development can be emulated in less than three months. Or, alternatively, if impractical given those time constraints, other options will be considered.

This report discusses different implementations in the section entitled Implementation.

### 2.5.2. Augmedix

Augmedix is a startup based in San Francisco which launched in 2012 providing health care solutions that maintain health care records without losing time spent with patients (Chandler, 2015). Augmedix has raised a huge $40 million from over 9 investors; this reiterates how Glass is being valued in medicine.

Currently, doctors type as much information that a patient provides when they visit them. This is time consuming and detracts the focus of the doctor from the patient to the computer that they are typing on. What Augmedix provides is a scribe that notes down all the information for you. This scribe watches the feed from Glass that the doctor is wearing and writes down all the important information freeing up time for the doctor to speak to the patient. This improves healthcare as the doctor no longer has to focus on two things at once. (Augmedix, 2016)

Through some research online, Augmedix has gone down the WebRTC route (Augmedix Stack, 2016; Augmedix GitHub, 2013). See section 4.1.2 for more information on WebRTC. However, rather than using media streams that handle the audio and video, it looks to be the case that they have used data channels to achieve the same effect. A big caveat of WebRTC is its inability to stream 1-to-many. It can only stream 1-to-1.

### 2.5.3. Vital Enterprises

Vital was started in August 2013 at a hackathon (Vital, 2016). Vital was born out of the need for surgeons to see vital patient information in real time. Vital uses the display in Glass to meet that need. Up until 15th August 2015, Vital Enterprises was called VitalMedics, showing that they were targeting medicine early on. Since then, Vital have transitioned to targeting the following industries: laboratory, automobile, aerospace, logistical and field service industries - anywhere where a user's hands are busy – and therefore changed their name to Vital Enterprises.

Vital has built a patent-pending technology called ZeroTouch™ which enables, "hands-free interaction with [their] data display and communication capabilities." Users simply tilt their head to navigate Glass' screens (Vital Enterprises, 2014). This is very beneficial to surgeons who are unlikely to have hands free to operate Glass. Vital are offering a cross-platform solution, targeting Android and iOS. Likewise, Vital's technology works with other smart glasses such as the ODG R-7.

In one video (Vital Enterprises, 2015), Vital's technology is shown to be displaying pictures on screen in Glass. Video from Glass is also being streamed out to a remote observer. Audio is shown to be received as well as streamed outwards. There is no mention of Glass being able to receive video. No mention of the technology used could be found.

### 2.5.4. Third Eye Health (TEH)

Third Eye gives healthcare providers an online platform for the sharing of audio, video and data in real-time. Third Eye's objectives closely align with this project's objectives. All data is passed through "secure cloud servers so no protected health information is ever stored." (Third Eye Health, 2016)

TEH works with many head wearables including Google Glass. The exact technology that Third Eye Health are using could not be found.

### 2.5.5. Xpert Eye

Founded in 2004, Xpert Eye (from AMA™) is another telemedicine provider that focuses on delivering teleassistance and telementoring (AMA, 2015). Teleassistance is when a person receives help remotely.

The Xpert Eye system uses USB for the transference of video streams, as seen in some demo videos (AMA, 2016). A user attaches Glass to a smartphone that has the Xpert Eye app, then Glass can send data to a remote peer. Although Glass does benefit from using the smartphone's battery, the smartphone has to now have a bigger battery to cope with providing the electricity for two operating systems.

With that being said, Xpert Eye was used wirelessly via WiFi for video transmission with a remote observer from France to Japan, a distance that is over 10,000km (AMA, 2015; Rennes Atlanta, 2014). This was a 1-to-1 session which does not meet the objective of being able to stream to as many people as possible in a broadcast style format, but nonetheless a useful point to know is that 1-to-1 sessions can be done over that distance. No mention of technical difficulties and limitations were found.

Xpert Eye works with a number of smart glasses including the R-7, Vuzix and Google Glass.

### 2.5.6. Findings

All the possible solutions above are indirectly competing with each other. They are all using Glass to stream peer-to-peer information over the Internet. Pristine and Augmedix have relatively similar types of implementations. Vital has the most unique implementation by far as it uses head movements rather than voice commands. This is one area that future work should definitely take place.

Xpert Eye has a complicated but reliable implementation. Xpert Eye uses USB to transfer data to a companion device, very much similar to this report. However, problems exist with Glass and USB, see section 4.1.5.

## 2.6. Interviews

During planning it was realised that knowledge on the terminology and processes in medicine were lacking. Interviews were setup to remedy this lack of knowledge and also to plan how Glass would fit into those processes. Within this subsection the interviews are reformatted as part of this report. Where the interviews had knowledge gaps, this report did due diligence and researched those gaps.

### 2.6.1. Medical student

In order to obtain more information surrounding how medical students are trained and educated, a meeting with a medical student, who studies at the University of Leeds, was arranged. The insight she provided helped with many of the terminology and practices used within medicine. This can help the reader, who may not have a medical background, assess whether or not Glass can be used as a telementoring tool.

There are several steps that one must take in order to become a full doctor within the UK.

The first step is medical student. Medicine is typically a five year course in which a student is rigorously examined and assessed through regular exams. A student is assessed on a variety

of topics, some of the assessments include, but not limited to, participating in mockup scenarios whereby a student is told to perform a task on a dummy in the form of a person. Glass would be welcome in this type of assessment environment where an examiner would be able to record how students perform in different types of mockup scenarios. Also, for the medical student, who may be diligently revising for such an assessment, it would provide an idea of what to expect.

The second step is attending a two year Foundation Programme (Medical Schools Council, 2016) where students learn skills through placements at hospitals. It is at this point medical students officially become Junior Doctors and start to earn a salary (BMA, 2016). During those two years on a Foundation Programme, Junior Doctors will continue their general training in medicine, applying knowledge learnt at university. Glass would be used greatly here because Junior Doctors have the capabilities of asking senior doctors for help. This is where a project like this could prove to be very useful in the training and education in a medical context.

The third step is specialty and general practice training. This is a four to six year stage, after which, Junior Doctors can apply for more senior positions such as medical consultant and GP principal roles. Again, Glass can be used in a similar sense to the second step, helping where needs be.

After the specialty and general practice training, there is ongoing continued professional development, whereby the, now senior, doctor is required to undertake further learning every year through perhaps online courses and modules.

### 2.6.2. IT manager

For more knowledge on the infrastructure in hospitals a brief interview with an ICT Manager was arranged. The manager had knowledge on the IT infrastructure used within NHS hospitals.

The manager pointed out that UDP (the most typical protocol used for video streaming) would unlikely be permitted into hospitals. TCP was more likely to be accepted. This was disappointing because so far most of the businesses (more information in section 2.5) that implemented video streaming was through WebRTC which can use UDP. He also pointed out that the majority of Trusts (providers of secondary health care) are, “behind on technology,” and that some Trusts are more averse to improvements than others. For example, Barts and The London School of Medicine and Dentistry has better WiFi than hospitals in any of the Leeds Trusts. To backup this claim, Mr Ahmed’s surgery (mentioned in 1.5) would have not happened if it was not for the above average WiFi at Barts. This report has not been able to validate the WiFi claims at any of the Leeds Trust hospitals as access to the WiFi network is restricted.

The conversation then turned to the underlying security of the NHS. All Trusts use what is known as the N3 Network to communicate with anyone outside the network (N3, 2016). Picture the N3 as a tunnel in which all outside communication going into the tunnel is scrutinised and filtered. This is a great security measure to block a malicious hacker who wants to gain access into private information. The manager clarified that data going out of Trusts is put under much less scrutiny. This means that UDP could be used to connect to outside networks but this would restrict the peer-to-peer nature of WebRTC as all data is

routed through N3 anyway. This causes a great number of concerns in terms of speed and connectivity.

## 2.7. Protocols

Now that the reader is aware of the limitations of Glass, the alternatives to Glass and how other people have implemented a similar type of system using Glass, the report will turn to researching how an implementation may be built, paying particular attention to the protocols that may be used.

### 2.7.1. TCP

TCP is based around guaranteed delivery. To ensure that packets are delivered, TCP receives an acknowledgement packet for every packet was delivered. Therefore, waiting for this acknowledgement packet can cause frames to be delayed. This acknowledgement aspect of TCP makes video streaming over TCP slow and largely ineffective. An upside to using TCP is that it is almost always accepted through firewalls.

### 2.7.2. UDP

UDP is the opposite of TCP. UDP does not guarantee packet delivery nor does UDP guarantee that packets arrive in the same order that they were sent. UDP is used in gaming and streaming applications which make it a perfect protocol for use in this report. UDP's downside is that enterprise firewalls have a tendency to block it.

### 2.7.3. RTSP

RTSP (Real Time Streaming Protocol) is "analogous to the remote control of the streaming protocols" (Syme & Goldie, 2004). This is because it controls whether or not content can stream over networks such as the Internet. It does not deliver that content but instead it delivers control information over HTTP and TCP.

For example, it sends commands like DESCRIBE, SETUP and PLAY to a host that's listening on the address 14.98.292.1 on the Internet. As a result of these commands, the host will then send back a description of itself (DESCRIBE), will setup the stream (SETUP) and then start the stream (PLAY).

### 2.7.4. RTP

RTP is a protocol that is commonly used with RTSP. The benefit of using RTP instead of just UDP or TCP is that RTP is mainly used over UDP but can fall back to TCP if UDP is blocked. Another benefit is that RTP can use any audio or video codec at any time, so rather than just sending one over UDP, RTP can transition between many.

As such, RTP comes with its own control protocol to obtain periodic updates on the data transmission (Durresi & Jain, 2005). This protocol is called RTCP.

### 2.7.5. WebRTC

WebRTC was mentioned when looking at comparative systems in other businesses (in section 2.5).

WebRTC can be used to stream video, data and audio to another peer securely and speedily (Castrounis, 2015). Audio and video get streamed via Media Streams (MDN, 2016) and data gets streamed through Data Channels (Ristic, 2014). WebRTC can use TCP or UDP depending on which one is not blocked (see above for more information on the two protocols). WebRTC's intended use-case is to be used browser-to-browser, but other systems do have the capability for it such as Android and iOS (WebRTC, 2016).

WebRTC offers many distinct benefits. One such benefit is ease of use. Ease of use in the sense that no additional plugins need to be downloaded. Further to that, no additional applications need to be downloaded because it works in all the latest browsers. At the time of this report's writing, Chrome, Chrome Canary, Opera, Firefox, Firefox Nightly and Microsoft's Edge all have partial support for it (Is WebRTC ready yet?, 2016). Note that partial does not mean that they don't support it at all, partial means that some WebRTC features have to be 'enabled' by the developer for them to work.

WebRTC is easy to use because oftentimes a program built upon WebRTC is structured in such a way that enables a video session to be made over the Internet at the touch of a button.

Security is also a benefit as the WebRTC protocol uses encryption by default and so every stream is secure. WebRTC has a further, more smaller, security feature which is that it requires the user to permit the use of the camera and microphone (netscan, 2015).

For enterprises WebRTC offers monetary savings because it does not need a telephone number to make a call. Plus, as it's peer-to-peer, enterprises do not need to invest in very high-end servers to keep the connection alive. The last statement does not imply that WebRTC can run without servers, in fact, WebRTC needs servers to broker the initial connection between peers – a process known as "signalling". Signalling is required because often peers are located behind complex network infrastructures and WebRTC needs to know how to reach a peer. Signalling is left up to the developer who implements WebRTC which means that a developer is free to use any communication framework to signal to another peer that a call is being made (Respoke, 2016). After signalling done, the two peers can connect to each other.

The process of setting up WebRTC is shown in Figure 2.7.1:

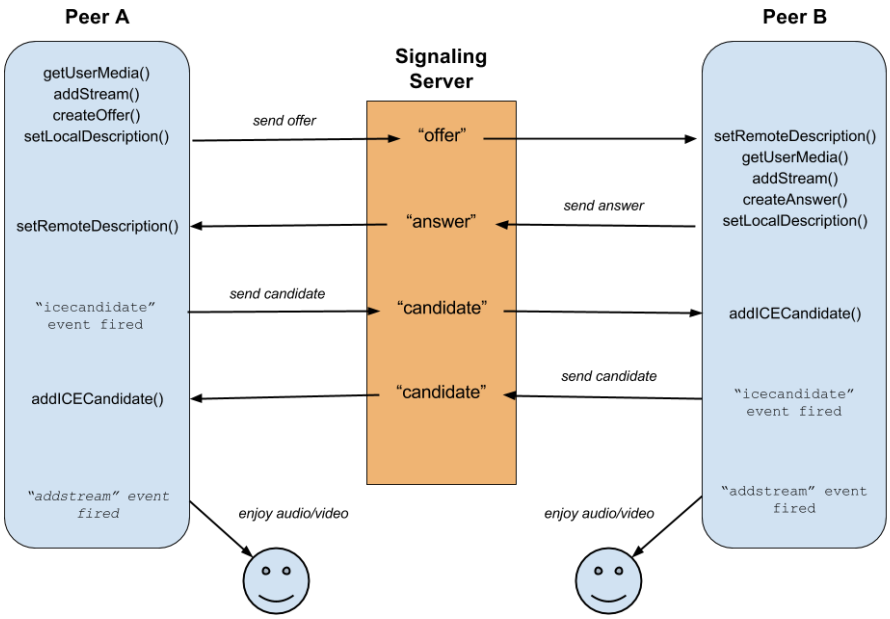



*Figure 2.7.1*
*(Alvarez, 2014)*

Within the confines of this report, WebRTC is a promising protocol which should, at the very least, be trialled as an implementation. However, referring back to Figure 1.1.1, the companion device may need to be dropped for the trial implementation to be developed. The reason for this is simple: WebRTC is peer-to-peer and not peer-through-device-to-peer. The companion device could instead be used to send Glass the information of the remote peer (Glass does not have a keyboard so this is a good solution) and let Glass connect to the remote peer by itself. This raises some concern as to whether or not Glass will be able to handle that amount of data being sent and received.

An implementation of WebRTC and a discussion on it can be found in section 4.1.

### 2.7.6. WebSockets

WebSockets are full-duplex, bidirectional communication sockets which can send anything from text to binary with low latency over the web (Pusher, 2016). WebSockets run through a server. For video streaming between peer-to-peer this is a big disadvantage, more so if more than two peers are hosting a 1-to-1 session. Another limitation is that WebSockets use TCP to maintain this connection.

A big advantage over WebRTC is that WebSockets can be streamed to more than one peer (broadcasting) whereas WebRTC does not support it unless extra modifications are made.

Even so, this report will provide an implementation using WebSockets that may be useful for comparison.

### 2.7.7. Bluetooth

Perhaps Bluetooth could also be integrated somewhere. Glass has a Bluetooth module which can be used to transfer files and photos to the MyGlass app (for more information on Glass'

Bluetooth see section 2.1). Could it be possible that Glass' Bluetooth LE (low energy) module can stream video and audio too?

Bluetooth can transfer with speeds of up to 25MB/s and therefore could be suitable for the purposes of this report (Paul, 2010). Bluetooth LE also reportedly consumes less battery which is ideal because Glass has a limited battery life.

### 2.7.8. Conclusions

WebRTC would be first choice for live streaming from peer-to-peer. However, this report is not just concerned with streaming peer-to-peer, it would be one of the aims to stream to as many people as possible.

Out of all the protocols RTP and RTSP sound like protocols aimed at live streaming. Their built-in frame checking and ordering relieve the software engineer of implementing it from scratch on top of UDP, for example.

The next protocol would be to use UDP. It is unreliable and more programming logic needs to handle the correct display of it, but if done correctly, UDP will give enhanced speed.

The last protocol of choice would be TCP because this report is targeting the use-case of live streaming.

## 2.8. Platforms

Previous research indicates that iOS is the dominant mobile operating system within the UK (Watson, 2015). Figures from 2015 show that iOS has a 58% market share and Android has a market share of 33%.

However, this report did not take into consideration the market share, or even previous research, because the end goal was to make Glass stream to another device. As Glass ran on Android, it seemed a suitable fit that the rest of the system would also be done on Android. It would save the time needed for context switching between Android and Swift/Objective-C.

Later in the report, a discussion on the choice of platform takes place in section 7.2.7.

# 3. Design and Planning

To mitigate risks and to make sure the project went as expected proper planning took place. A discussion on the methodology and how it will and has been used throughout the project will be presented here.

## 3.1. Methodology

The methodology used for this project actually turned out to have been two: Scrum and Kanban. Methodologies can either add rigid structure to a research project or add a more lenient structure. Scrum and Kanban was chosen as a dual methodology for this project because it added a lenient structure to this project that kept the project on schedule and on target.

### 3.1.1. Gantt chart

Giving out accurate timings for software is an arduous task as typically projects overrun in terms of cost and time by 30% (Jorgensen, 2014). For the most part, it's due to client's favorability for lower timings that software based projects overrun, as lower timings mean less money to be paid. As this project was not done for a client but for research purposes this pressure is alleviated.

Gantt charts help mitigate the overrun of cost and time. Gantt charts provide an overview of what the project will encompass and the time it will take. In Appendix D: Gantt Chart this project's Gantt chart is viewable for reading. This Gantt chart was done at the start of the project so some to-do items have been modified.

The reader will note that the project was broken down into four iterations, each four weeks long. Each iteration had a list of to-do items. The first sprint is where the Kanban structure was integrated into Scrum.

### 3.1.2. Scrum and Kanban justification

Scrum has a list of items to complete as part of a list called the "Sprint Backlog" that the Scrum Team completes. But typically within each sprint there is significant lack of visualisation into how the flow of work is progressing. Sure, a Scrum Team could keep on top of things by emailing back and forth, but this becomes a chore.

Kanban advocates the use of continuous improvement by purposefully keeping the project in a state of movement. It is due to this constant movement that Kanban increased productivity in the first sprint as it granted time to experiment and prototype with speed. Moreover, Kanban was used because it gave the Scrum Team a separate version of the Sprint Backlog that could be reset when each prototype was discarded. Kanban was used to create lists of to-do items that could be updated from "To do", "In progress" and "Done". The higher the item on the "To do" list, the more priority it got and the more "Done" items a prototype scored, the more likely it would be forwarded into the next sprint. This is why Kanban was integrated into this project's methodology in the first sprint.

Scrum was chosen as the overarching methodology for this project because the weekly meetings and meetings with stakeholders merged with Scrum perfectly. This project's

supervisor acted as the Scrum Master and the project stakeholders acted as the Product Owners. Every week the Scrum Master guided the Scrum Team on what to do and the team, with the aid of Kanban, did the rest.

Scrum sets up a framework for the stakeholders and the team to synchronise with each other. That is, the meetings gave everyone an opportunity to update one another.

Particularly effective was the use of time boxing within Scrum. Time boxing enabled the project to head in the right direction in four week sprints. Without time boxing, the project had an increased possibility of overrunning.

At the end of each time boxed sprint was a feedback meeting with the Product Owners/stakeholders.

## 3.2. Prototypes, designs and contingency plan

The first sprint was where rapid prototyping took place. As Scrum is an Agile methodology, it allowed the flexibility of throwaway prototyping, which is a type of prototyping where the previous prototype could be discarded (Campbell, 2014). With Kanban also being used, the project could select which prototypes marked most of the to-do items as done. This prototype could then be marked for forwarding into the second sprint.

Using concepts from the module GUI, some sketches of a prototype were drawn before the actual implementation starts (Appendix G: Low Fidelity Prototype). This enabled clarity and focus in the pursuit of the final implementation.

If everything did not go to plan, the contingency plan provided in Appendix F: Contingency Plan would be a good fail safe backup.

# 4. Implementation

Now that the reader has had a chance to understand the limitations of Google Glass (if not, then section 2.3 is recommended reading), this report will outline a telemedicine system that will meet the objectives outlined in Objectives (section 1.3) and hopefully combat those limitations.

## 4.1. Iteration 0

This iteration marks the initial phase that was performed prior to the official start date of the project and then continued after the official start date had past (see Appendix D: Gantt Chart for project planning information).

In this iteration a wealth of background research was performed (see section 2). Various prototypes were tested in this stage as part of a Kanban style format which fortunately granted the flexibility to do so. For a detailed explanation regarding the choice of methodology please view the section on Methodology (3.1).

Out of all of the four sprints, this sprint proved to be the one that was the most intensive. The items listed below are the implementations that were all trialled.

### 4.1.1. WebRTC (Hybrid)

The first implementation in this project was done using WebRTC because it was one technology that was mentioned in the section when looking at other possible solutions (in 2.5) and explained in section 2.7.

As Glass was running an older version of Android, CrossWalk's Embed API was used. CrossWalk is a project that brings newer APIs to older devices (CrossWalk, 2016). Using CrossWalk on Glass meant that Glass could use JavaScript functions that were not yet implemented in Glass' in-app browser implementation called WebView. This subsequently meant that WebRTC would then be available in Glass.

To make it easier to get started with WebRTC, PeerJS (a library for easy WebRTC) was used within CrossWalk. This meant that external JavaScript files written for PeerJS would have to be bundled in the final app, which was an extra step when compiling. Though, this meant less code needed to be written for each platform.

Once the JavaScript was coded and bundled, the app was tested on a **smartphone initially**. The result was that the smartphone could stream reliably to a remote peer at 30fps (frames per second).

Comparatively, when the app was tested on Glass, the stream was 2fps and out of the five times it was tested the app force closed itself three times. The CPU showed that CrossWalk was using a lot of CPU cycles to stream and so was running at above 60% when Glass was streaming. Comparatively Glass ran at 10%-30% when Glass was idle. Glass also generated a lot of heat the five times it was used to stream. Even after lowering the framerate and the quality, all of the issues still remained.

A diagram of this implementation looks like the one in Figure 4.1.1:


*Figure 4.1.1*

### 4.1.2. WebRTC (Native)

In the above subsection, the implementation was done through a WebView which meant it was a hybrid implementation. With a native implementation, the app is free to use GDK's (Glass Development Kit, a variation of the Android Development Kit) APIs directly which meant the frames per second should increase.

After implementing a Glass app and an Android app that it could connect to, Glass rendered 2fps to 5fps. This speed was still not feasible in practice and so this implementation was discarded.

Another major negative was that the native WebRTC library lacked sufficient documentation so implementing WebRTC on Android was a demanding endeavour. The WebRTC SDK integration for Android is also an in-depth process. Luckily, Pristine (mentioned in section 2.5.1) has open sourced the build so that the library is available for downloading and integrating into an Android app. However, the last update to it was in December 2015 (Maven, 2016).

Out of all the implementations this was the one with the most surprising result. Mainly because other people were using WebRTC successfully. It could be the case that the implementation was configured wrongly and so led to a speed dysfunction. Nonetheless, the Evaluation section outlines future work surrounding what could be done to make it work.

### 4.1.3. Airtube


*Figure 4.1.2*

Airtube is a library that enables direct communication between two Android devices (Airtube, 2016). In this case, Glass was one device and the other device was a local Android smartphone. Airtube required that users download another app for peer discovery in a Service Oriented Architecture. The fact that users had to download another app for it to work stymied work on this implementation, but the implementation proceeded to be built just to understand if it could be modified to work with Glass.

Upon implementing an app backed by Airtube, the conclusory findings were:

- It was impractical for users to download another app for service/peer discovery.
- Users had to install the service discovery app through the command line.
- The media codecs were outdated resulting in a lot of time spent overriding classes.
- Did not work on a smartphone (specifically a Samsung Galaxy S4) resulting in “Error 1001” and “incorrect colour”.
- The library had not been updated for over a year.

All of this meant that Airtube was certainly not an implementation that would be suitable for Glass in this context. A diagram of the implementation that Airtube would have had is shown in Figure 4.1.2.

### 4.1.4. Libstreaming

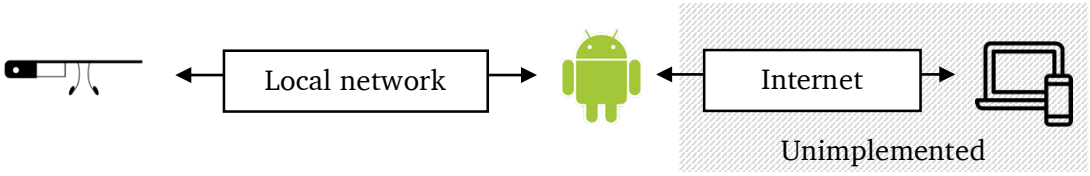



*Figure 4.1.3*

Libstreaming is a library that has support built-in for Google Glass (Libstreaming, 2016). It is very easy to configure and setup. It has support for RTSP, which is a protocol used for audio and and video streaming. RTSP is used in security cameras to stream captured frames to a WiFi enabled storage box.

Using Libstreaming’s configuration options an Android smartphone, the Android smartphone could successfully stream to a tablet. Testing this on Glass resulted in a stable stream. The advantages and disadvantages are outlined below. A diagram of the successful implementation looked like the one in Figure 4.1.3.

Advantages:

- Very little image distortion (no artefacts) but cloaked in a green hue.
- Many configuration options.
- Uses an industrial standard protocol RTSP.

Disadvantages:

- Bluetooth needed to be turned off for smoother playback.
- There was a 2 to 3 second delay in streaming video from Glass to another device.
- Glass ran hot after a short time.
- Streaming was unidirectional only. Complicated setup for a bidirectional implementation.
- The setup of Libstreaming was unclear and manually checking the source code was done many times.
- A resolution of 240 x 160 worked only with Glass.
- The framerate was 10fps.

It is worth highlighting a few key disadvantages to further explain why this implementation was not taken forward.

A key disadvantage in the above list was the fact that Libstreaming was unidirectional. Libstreaming is setup in such a way that a client has to connect to the RTSP stream in order to view it. To view a stream a client usually has to connect via IP address. This is a complication because a user simply cannot type in an IP address and obtain a stream. The main reason for this is NAT (Network Address Translation). Unless a user is aware of the complications of NAT and understands how to bypass it, it makes connections between two peers over the Internet very difficult.

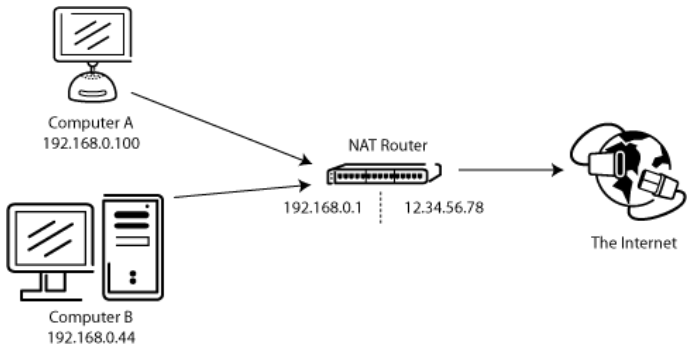



*Figure 4.1.4*
*(FileZilla Wiki, 2016)*

Typically, NAT is done by a router that is at the centre of the network. The router assigns what is known as public IP addresses. Public IP addresses are accessible by anyone over the Internet. If a device is behind NAT then they will also have what is known as a private IP address, i.e. one identifiable on the Local Area Network. In diagram Figure 4.1.4 above, the public IP address of both Computer A and Computer B is 12.34.56.78 and the private IP address is the one attached to the Computer A and Computer B. Both Computer A and B are aware of each other's private IP address because they are connected to the same network.

Now imagine if Computer A was Google Glass and it was waiting for connections to stream its video stream. How would a remote observer connect to it? Surely via its public IP address (12.34.56.78)? That's correct but what if Glass and the remote observer did not know Glass' public IP address? The answer is that a connection would not happen. The two devices don't know where the other is on the Internet. This scenario is typically the case when a router has a dynamic IP address. This makes it even more difficult to connect to a remote peer.

In section 2.7.5, it was explained that WebRTC uses signalling to find two peers on the Internet. The signalling process is the stage that sorts out the complications of NAT so WebRTC benefits from the lack of the complications of NAT traversal.

This report most likely presumes that the network infrastructure in hospitals is using the IPv4 scheme because the interview with the IT manager (section 2.6.2) mentioned that the infrastructure in some hospitals was poor.

Due to the limitations above, this implementation was discarded.

### 4.1.5. USB

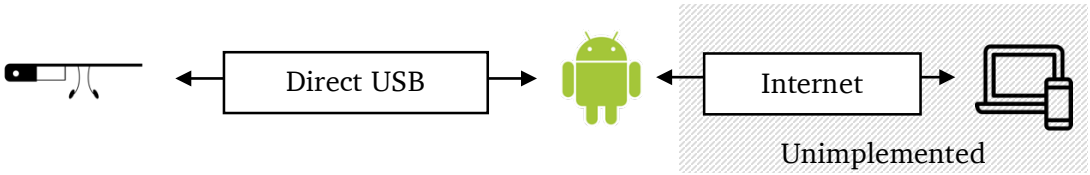



*Figure 4.1.5*

The next implementation was in the form of USB and it had the resemblance of the diagram in Figure 4.1.5. This was the most promising implementation by far as it solved many of Glass' limitations.

As the reader may be aware, Glass can be connected to a computer by a USB cable which allows the computer to communicate with Glass. USB operates on what is known as an accessory-host architecture, which is similar to a non-threaded client-server architecture. At any one time one device can act as a host and one device can act as an accessory. Computers almost always act as hosts whereas smartphones almost always act as accessories. Hosts provide power to the accessory and perform initial contact. At this point, the reader is prompted to imagine a keyboard plugged into a computer and note that the keyboard operates on the same protocol as this accessory-host architecture.

In this implementation, the plan was to make Glass the accessory and the companion smartphone the host. This meant that Glass was powered by the smartphone, resolving the limitation of Glass' meagre battery life. It also meant that the streams to/from Glass were theoretically lag free as the streams would travel through the USB cable directly to/from Glass. We could then stream from the companion smartphone to a remote device using the smartphone's power to facilitate the stream across networks. This resolved a lot of WiFi and security issues that existed in Glass. To visualise this, Figure 1.1.1 is a good reference.

Fortunately, to enable a smartphone to act as the host, a solution already existed in the form of an adapter called "USB OTG (On The Go)". These USB OTG adapters cost a small amount (£3 to £10) so a person wishing to stream from Glass could easily purchase one. The key feature of USB OTG, within the context of this project, was the fact that USB OTG can make a smartphone act as a host and Glass an accessory.

The implementation of a USB OTG app was time-consuming, a week was spent learning the protocols of USB and developing test apps for Glass and a smartphone. Even then, the implementation kept failing, Glass did not respond to any USB initiation requests.

It was at this point that some further research into the USB specification was needed and it revealed that USB 2.0 (the specification that Glass used) was half-duplex, meaning that only one device can be communicating at any one time. It was, in fact, USB 3.0 that was full-duplex. This finding stopped this implementation from going any further because the initial requirements were to enable streaming to and from Glass in a full-duplex fashion, which could only be done with USB 3.0.

Moreover, as USB 2.0 is half-duplex, Glass would be sending video and audio to the companion device but the companion device would not send anything back, not without great difficulty anyway. So as a result, a Glass wearer would not see the remote person in Glass' display, which is one of the requirements of this report.

### 4.1.6. Findings

Many implementations were tested in this first sprint. The most promising solutions were discarded at this point. Though, inspired by the USB implementation, the primary idea now changed from Glass being a standalone device that could send and receive, to one that included a companion device that would alleviate the computation necessary of the sending and receiving of streams.

Below, in Table 4.1.1 there are the various implementations and how fast they were.

| Implementation | FPS | Notes |
|---|---|---|
| WebRTC (Hybrid) | 2 | Overheating, force closes |
| WebRTC (Native) | 2 - 5 | Slow stream |
| Airtube | - | Unable to setup |
| Libstreaming | 8 - 10 | Poor image quality |
| USB OTG | - | Half-duplex only |

*Table 4.1.1*

The fastest and most configurable implementation in this iteration was Libstreaming but it suffered from poor image quality and a time-demanding library that was hard to get running. The slowest and most surprising implementation was WebRTC but it remains to be concluded whether the implementation was implemented incorrectly or WebRTC is not able to work on Glass. USB was the most promising solution but it was not bidirectional and so was discarded.

## 4.2. Iteration 1

Picking up where Iteration 0 left off, this iteration aimed to test some more solutions and finalise the prototype idea.

### 4.2.1. WiFi Direct

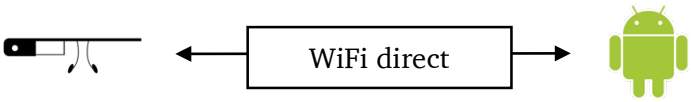



*Figure 4.2.1*

Perhaps a connection could be made from Glass to the companion device via WiFi Direct? WiFi direct connects one device to another device directly.

Inspired by the USB implementation, it was theorised that if we could achieve a connection to a smartphone from Glass directly, then we would be able to send the stream through the smartphone (the companion device) to the remote peer.

This implementation took a day to develop. However, once it was tested on Glass, Glass reported an error of "WiFi Direct unsupported" which halted further investment into this implementation. The implementation had an architecture shown in Figure 4.2.1.

### 4.2.2. MJPEG streaming


*Figure 4.2.2*
*Sending JPEG frames to an Android smartphone.*

MJPEG (Motion JPEG) streaming was implemented after reading through an article online regarding MJPEG on Android (Peepers, 2013). It is shown diagrammatically in Figure 4.2.2.

MJPEG is a good choice for Glass as MJPEG does not use a lot of computing power in comparison to say H.264 (another compression standard). The disadvantage is that MJPEG does not achieve as high a quality as H.264. While this report is discussing the H.263/264 codecs, it would have been unwise to have included them in an implementation as they suffer from crippling patents by some of the top companies (Panasonic, Dolby, Sony, Microsoft, Apple, to name a few – there are 26 companies in total) that stipulate that royalties have to be paid if a company were to use H.264 for commercial purposes (Patel, 2010). On that note, even using MPEG4 would cause a conflict of interest, as that too would mean that royalties would have to be paid.

Once developed to a reasonable standard, the app was initially tested on a smartphone and achieved 20fps. Comparatively, when tested on Glass, Glass and achieved a slower, but not

impractical, 10fps approximately at a quality of 720p. Surprisingly, this speed was achieved by using TCP.

What's even more surprising is that this implementation was using no complicated library to stream. This implementation was done using Java TCP sockets to sends frames across devices. Use of TCP sockets was learnt in the module Networks and Scalable Architectures. The simplicity made this implementation more maintainable and easily extendable than other implementations thus far.

In some previous implementations Glass was mentioned to become very hot during use. In this implementation, Glass did get hot but not as hot. Concordantly, Glass had a better time in working throughout the stream and not shutting down than some previous implementations.

The quality of the stream was also more than acceptable and more importantly, there was no artefacts in the resulting stream whatsoever.

The reader must note that this implementation did not include streaming to a remote peer. This implementation focused on transferring data to a companion device, such as a smartphone which was located on the same network as Glass. The sections below outline how MJPEG was modified so that streaming to a remote observer became a reality.

### 4.2.3. Findings

This report found that, so far, MJPEG streaming could successfully stream frames from Glass to an Android smartphone. The MJPEG implementation was taken into the third sprint for further improvement.

| Implementation | FPS | Notes |
|---|---|---|
| WiFi Direct | - | Unsupported |
| MJPEG Streaming | 8 - 13 | Garbage collection issues |

*Table 4.2.1*

Table 4.2.1 shows the significance of this finding. Comparing it to Table 4.1.1 in the previous implementation, the MJPEG solution is very fast but suffers from garbage collection. In the next section this report will try to incorporate the NDK into the implementation to see if that has any effect on the implementation in terms of speed. Use of widely available profiling tools will be used on the app so that further debugging in the garbage collection shall take place.

A quizzical notion this report found is that Glass has the capability (e.g. supported WiFi module and system capabilities) to run WiFi Direct. It is unlikely but possible that Google may release an update to enable WiFi Direct capabilities.

## 4.3. Iteration 2

In this subsection the following headings show each step of the improvement of the MJPEG implementation. Some screens such as the “Setup” screens were, by now, implemented. See Figure 4.3.1 below:

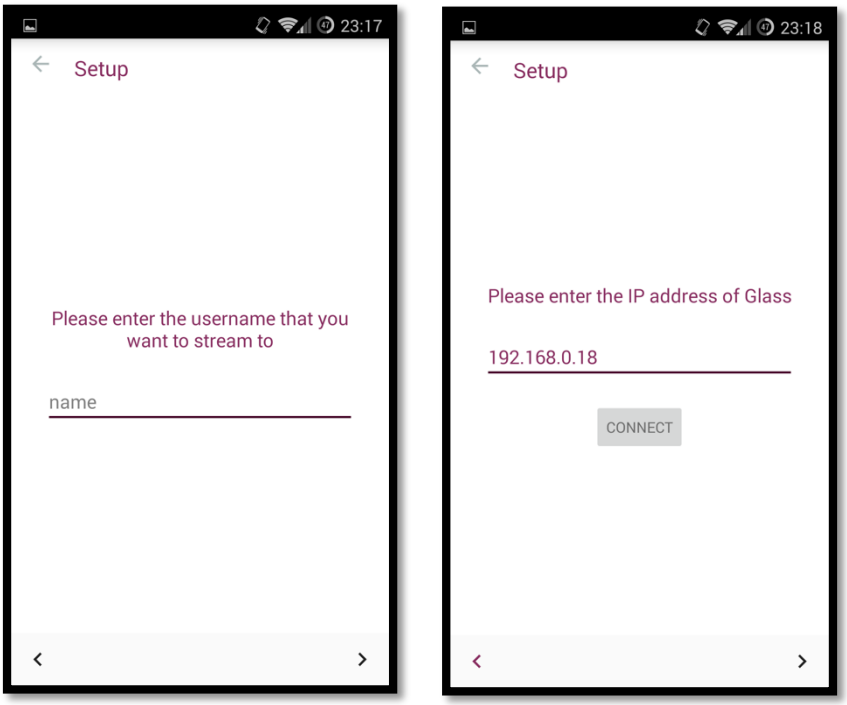



*Figure 4.3.1*
*Setup screens*

### 4.3.1. MJPEG and WebRTC

Once MJPEG was implemented and was streaming to a local Android smartphone, it was then time to display and forward that stream from the smartphone to a remote peer. One hypothetical solution was found in the form of WebRTC (explained in section 2.7.5 and tested in 4.1).

The reader may notice that WebRTC had already been implemented and declared to be unsuccessful. That is a fair observation but that was Glass streaming to a remote device only. There was no mention of a companion device handling the computational intensity of packing the stream and sending it over the Internet, which is exactly what this implementation will report the findings of.

Using the WebRTC (Native) implementation as the foundation for this implementation, this report found that WebRTC will not accept another stream as an input. That is, to reiterate, WebRTC would only work if the stream was from a camera source and audio source on the device itself. So, in effect, streaming the video and audio from Glass to the Android smartphone in order to send the streams to another device via WebRTC could not be done.

For a more technical perspective on this finding, when Glass would stream to the companion Android smartphone, the stream would be displayed on the smartphone's screen. The way it was displayed was through a SurfaceView, which is an object in the Android library that handles graphics, or in this case, streamed frames. However, this report found that there was no way that the SurfaceView (along with the rendered frames) could be fed into WebRTC so that it could send them to the remote peer.

So, due to this finding, another method had to be researched. The implementation below outlines one such method.

### 4.3.2. MJPEG and WebSockets

WebSockets (explained in section 2.7.6) was another protocol that was capable of streaming data over the Internet. The first point to make about WebSockets is that it requires a server to stream through. This is almost always going to be slower than using a direct peer-to-peer connection like WebRTC. Please see section 4.6 to find out more about the underlying technologies used in the server.

When the WebSocket server was run locally the delay of sending a stream from Glass (through the companion Android smartphone) to the server and the server sending it on to the remote peer, was negligible. As WebSockets are TCP based, this delay can be partly attributed to the guaranteed delivery mechanism of TCP and partly due to the low technical specifications of the server. Figure 4.3.2 shows a working stream:

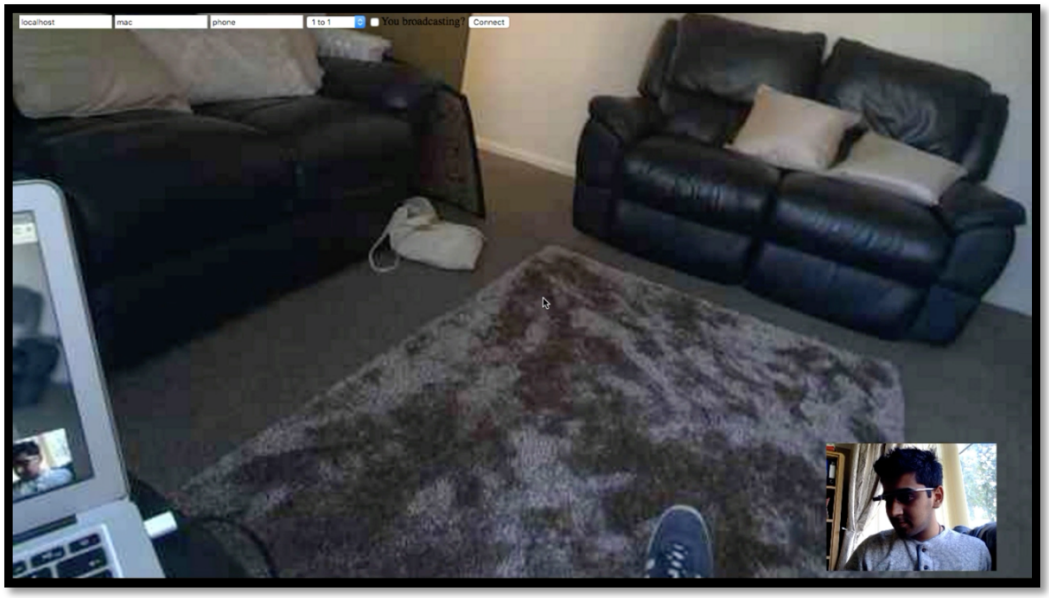



*Figure 4.3.2*
*Remote peer viewing the stream from Glass*

The screenshot above shows a recording of a working video stream from the remote peer's screen. In the bottom right hand corner the remote peer can see him/herself reflected back as is common with video calls. The larger screen shows the feed from Glass. At the top of the screenshot are some input boxes. In the above screenshot, the astute reader may notice that it is a locally hosted Chrome page that the implementation is running on. The scope of the creation of that web page is out of scope for this paper and will not be discussed. However, this shows that the implementation is cross-platform.

Comparatively, Figure 4.3.3 shows the screen on Glass.

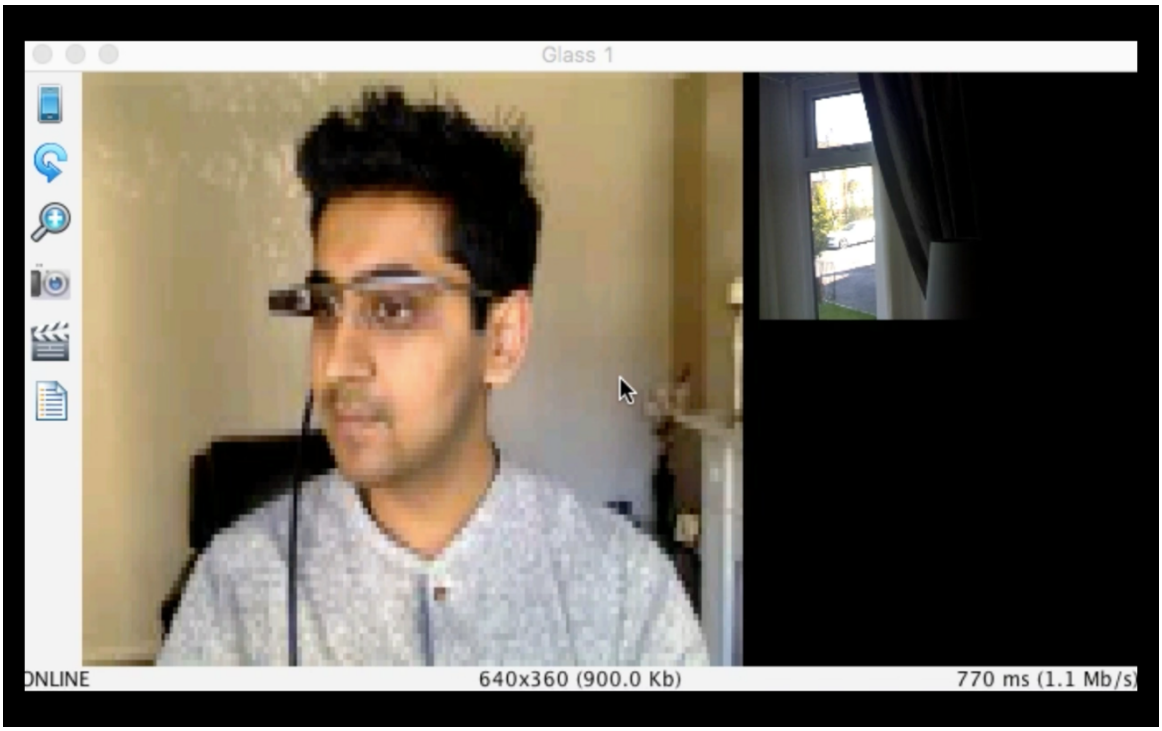



*Figure 4.3.3*
*Glass' screen*

On the left in the above screenshot is the stream from the remote peer. Contrary to what is shown in the screenshot, the remote peer would most likely **not** be wearing Glass. The top right is what Glass captures. It is useful to see what Glass is capturing because then the Glass wearer can position Glass in a correct way. The bottom right hand side of the screen is reserved for future work.

Audio segments were also sent over WebSockets. Please note that coherent audio could only be rendered when sending from the remote peer to the companion device. When audio was streamed from Glass to the remote peer, a lot of static noise was emitted.

When this project tested the implementation on a server located in London with 512MB of RAM and unlimited bandwidth rate, the video stream had a delay of approximately 8 seconds and a total distance of 338 miles. This delay is unacceptable when applied to the objectives of this report. Further improvement would be needed, see 4.3.3 below.

Finally, the implementation looked as follows in Figure 4.3.4:

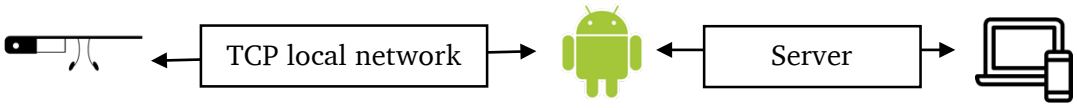



*Figure 4.3.4*
*Sending frames to a remote peer through an Android smartphone with TCP.*

### 4.3.3. Optimised JPEG streaming

The implementation mentioned in section 4.3.2 was modified so that any extraneous information, such as the headers of the stream and the headers of the images, were removed. This improved speed marginally.

There was also the issue of garbage collection occurring at 4 to 10 second intervals which led the Glass app to freeze somewhat. To clarify 'freezing', the Glass app was halting progress of the stream, i.e. the remote peer was no longer shown to be moving in Glass but Glass' preview was not affected. This shows that it was something to do with the network rather than Glass because Glass' preview was still moving.

In order to mitigate the freezing, this implementation focused on further optimisations in the form of correct buffer allocations and consuming only what was needed increased the speed further. These improvements increased the speed from 10fps to 17fps.

Other improvements were made such as the use of threads, increasing the buffer size and also modifying the app for Glass further increased the speed. More improvements came in the form of setting objects to null and recycling bitmaps. Setting objects to null means that the garbage collector is more likely to collect that object and so freeing up memory space.

Another improvement was that frame limiting was used. As frames were being sent through the smartphone, the smartphone could effectively limit the rate at which frames were being sent to Glass. As a result, the garbage collector ran less.

However, at random times, the issue of freezing still persisted. When testing with another Glass, the findings were that the issue of freezing streams was not present at all. However, after about a week of working with this new Google Glass, the issue of freezing became apparent once again. After more work with both the old pair and the new pair of Google Glasses, it was concluded that the main contributing factor to the stalling was the protocol TCP.

### 4.3.4. RenderScript (C++)

Building upon the optimization of the JPEG streaming implementation above, another implementation was done using a RenderScript. RenderScripts are highly performant C++ scripts that interface with Android through the NDK (Native Development Kit – allows C++ to be used with Android). (Sams, 2011)

Once the YUV RenderScript was implemented, tests were done to see if the Java garbage collector ran less. What occurred was the opposite, the garbage collector ran more frequently because bytes were being allocated a lot more.

Subsequently, this implementation was discarded, but serves as useful reference point.

### 4.3.5. Findings

In this iteration, this report found that streaming with Glass could be done. Many optimisations were done to improve the framerate of Glass. This report concluded that WebRTC was slow and that performing raw TCP socket transfer and then transferring over WebSockets was the path that brought the first workings of a prototype. The

implementation, now having been developed to a reasonable state, was in a position for improvement.

| Implementation | FPS | Notes |
|---|---|---|
| MJPEG + WebRTC | - | Cannot ‘fake’ a stream |
| MJPEG + WebSockets | - | Working but stalling |
| RenderScript (C++) | - | High garbage collection frequency |
| Optimised JPEG | 16 - 17 | WiFi issues |

*Table 4.3.1*

The report found the majority of the implementations in this iteration to be unsuccessful (see Table 4.3.1). In theory WebRTC would have been very useful for 1-to-1 streaming but it leaves the question open for broadcasting as WebRTC.

This report would like to highlight that the results above work only with a quality of 640x360px. When tested with 720 quality, the stream was very slow and kept stalling every 3 to 6 seconds for 5 seconds at a time. The profiling tools showed that Android was allocating many bytes within a short period of time and so the garbage collector ran more often. It also seemed that the WiFi network had to have low load on it for it to be of any usable use at a quality of 720p.

## 4.4. Iteration 3

In this sprint/iteration some other prototypes were tested with different protocols. All of this was done using the Optimised JPEG streaming implementation from the last sprint.

### 4.4.1. UDP

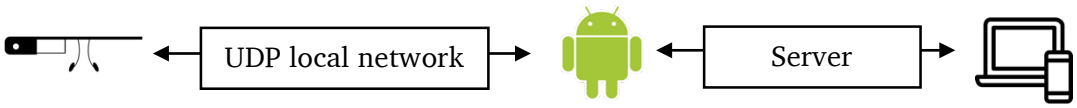



*Figure 4.4.1*
*UDP to an Android smartphone to a server.*

As the implementation so far was using TCP sockets to stream frames from Glass to the Android smartphone (and vice-versa), this report sought to build an implementation with UDP instead (see section 2.7.2 for more information on this).

Even though UDP is not accepted from outside a hospital's local area network, UDP would be able to work between two devices if they were on the same local network within the hospital.

Once the TCP sockets were swapped with UDP sockets, the implementation was tested with Glass and the Android smartphone on the same network. The result was that it was very fast, perhaps the fastest stream so far but this report **cannot** accurately say so for sure. The reason is that the stream was fragmented and cut in places. Some frames in the stream were out of order.

The implementation is shown in Figure 4.4.1 but further work on UDP can be done. See section 7.2.3.

### 4.4.2. Bluetooth

Another implementation was done using Bluetooth which was explained in section 2.7.7. The Bluetooth implementation was more reliable than the UDP implementation but there was one big finding when using Bluetooth: the video stream was either being sent or received at any one time.

There were two implementations that were tested:

1. Two Bluetooth sockets. One on Glass and one on the Android smartphone.
2. One Bluetooth socket for sending Glass' video. One TCP socket for receiving video.

In (1) and (2) it was found that Glass would either be sending or receiving, but not both at the time. Deductions from the above suggest that Glass can either use its Bluetooth module or its WiFi module at any one time, i.e. Bluetooth is serial.

The diagram for implementation (1) looks like the one in Figure 4.4.2:

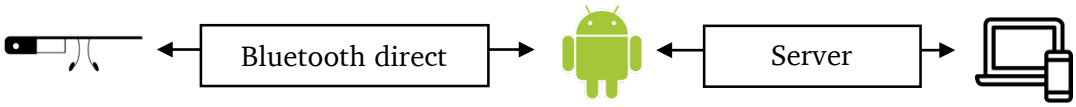



*Figure 4.4.2*
*Bluetooth implementation direct to/from companion device.*

The diagram for implementation (2) looks like the one shown in Figure 4.4.3:

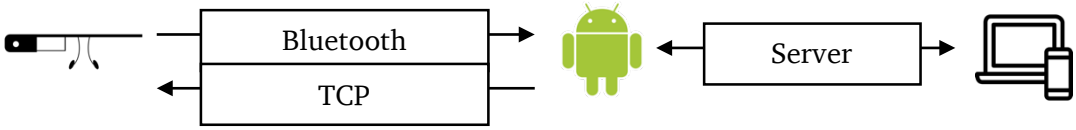



*Figure 4.4.3*
*Bluetooth sent to Android smartphone. TCP received from Android smartphone.*

### 4.4.3. Broadcasting

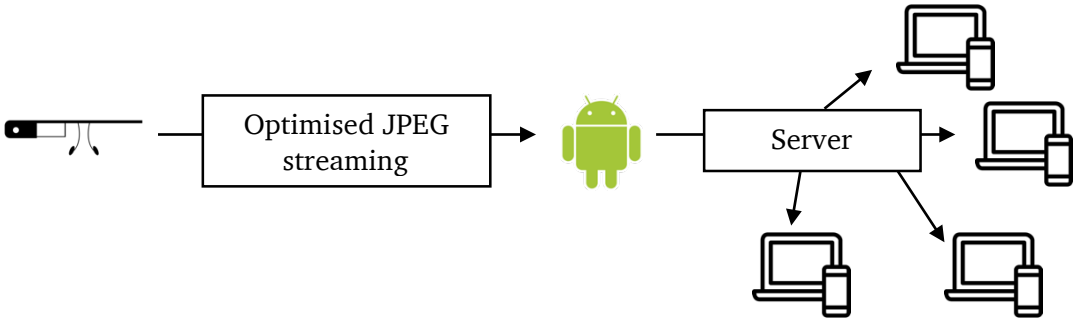



*Figure 4.4.4*
*Broadcasting to n number of clients.*

After the Bluetooth implementation, the project turned to spending time adding fixes to the code so it was as robust as possible. Similarly, the server was amended to add another mode: Broadcasting.

Broadcasting, unlike 1-to-1, was sending Glass' stream to as many people connected to the server as possible. In the section titled Server (below), it is explained that the server can handle many concurrent connections – as a result server issues hardly troubled the project.

As the reader may or may not be aware, Figure 4.4.4 above fully resembles Figure 1.1.1 and thus the conceptual vision for this project has been successfully implemented.

### 4.4.4. Audio

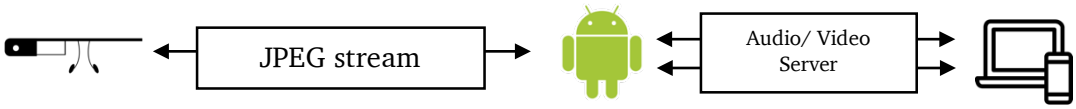



*Figure 4.4.5*
*Audio and video alongside each other in a 1-to-1 implementation.*

Following on from the implementation of Broadcasting, audio was implemented. The server was setup for streaming audio and so was the Android app. A new WebSocket had to be created for the audio to stream over. The renewed architecture is shown above in Figure 4.4.5.

### 4.4.5. Stakeholder meeting

Partway through the sprint/iteration, a (delayed) meeting was arranged between Product Owners, Scrum Master and Scrum Team. In that meeting many suggestions and improvements were provided.

1. The stakeholders were very pleased with the streaming aspect of the system. One additional feature that they requested was that the stakeholders wanted to record interactions between patients and doctors.
2. They felt that audio one way was okay but, if possible, text displayed in Glass was a feature that could be more of benefit.
3. They explained to me that the WiFi in hospitals varied, backing up the claims made by the IT Manager in section 2.6.2.
4. A simpler interface of the Android app was requested.

For feedback on the system as a whole, see section 6.1.2.

### 4.4.6. Findings

UDP and Bluetooth were not feasible solutions to the problems that this report set out to solve:

- UDP needs extending so that it can correctly order frames.
- Bluetooth was found to be serial so streaming both ways was impossible.

| Implementation | FPS | Notes |
|---|---|---|
| UDP | - | Stream broken |
| Bluetooth | 10 - 15 | 640x360, low res but reliable. Low battery consumption |

*Table 4.4.1*

At the end of this sprint, video and audio streaming was implemented successfully. Broadcasting to n number of clients was also successfully implemented. The stakeholder

meeting was performed as part of this project's methodology and many suggestions and improvements were discussed.

One more finding was the distance of the companion device and Glass from a WiFi router impacted the framerate massively. This finding was not realised before as the implementations were all done in the same position. But after moving positions to one that was on the fringes of the WiFi routers reach, the stream was unusable and, by all reckoning, broken. This report understands that it is possible that a user may be operating the implementation on the edge of the WiFi router's range, but in enterprise WiFi networks, the routers are positioned close to each other for the exact reason. In practice therefore, this should not be a problem.

## 4.5. Iteration 4

In this iteration mainly cleanup and refactoring occurred. Implementation of the stakeholder's/Product Owner's requests were incorporated into the app.

### 4.5.1. Refactoring

During this last sprint, a lot of refactoring took place. The networking code was implemented in a haphazard manner that led to TCP sockets being passed from one object to another causing what is known as spaghetti code (Computer Hope, 2016). This was illogical so had to be refactored into one file for it to become easier to read. Not only did this illogical code increase the amount of repeat code, but it caused additional complexity when implementing other features such as broadcasting. The DRY (Don't Repeat Yourself) principle was taught in the module Software Engineering and served as a useful measure of keeping code readable.

Refactoring was no small effort. By using knowledge gained from the module Software Engineering, refactoring took place slowly but surely.

### 4.5.2. Stakeholder features

Whilst the app was being refactored, this project took the chance to implement the improvements suggested at the stakeholder's meeting (see section 4.4.5).

The app was designed simpler, removing some screens such as the "View yourself screen" on the "Join a stream" page.

### 4.5.3. Recording

The recording feature has not been implemented due to lack of time. In the end, the screens had been implemented for it. The code is 99% complete. A future software engineer need only figure out how to record the audio and video stream to file.

To reiterate, the recording feature was to record to another smartphone/tablet directly as Glass has a limited amount of storage.

### 4.5.4. Stakeholder trial

In this final sprint, another meeting was arranged. This time it was with two A&E doctors. In this meeting, two scenarios took place. A scenario is when trained or untrained actors mimic a patient and a doctor performs diagnostics on that actor.

In one case, one of the A&E doctors mimicked a patient and the other performed diagnostics on that 'patient' whilst wearing Glass.

In another case, one of the A&E doctors assumed the position of Team Lead and the other assumed the role of patient. A Team Lead is a senior doctor who watches over many operating rooms.

In both cases, the non-patient was wearing Glass. Glass was using its built-in recording feature to record the interaction. Once the recording was over, the video was reviewed and findings provided in Evaluation.

### 4.5.5. Findings

As this is the last sprint, further findings for this sprint are given in the section titled Evaluation.

| Implementation | FPS | Notes |
|---|---|---|
| Optimised JPEG + WebSockets | 16 - 17 | WebSockets are TCP based, therefore slow and stalling |

*Table 4.5.1*

The final implementation (Table 4.5.1) was speedy over TCP but a protocol such as UDP would definitely be a worthwhile consideration. Of all the implementations this seemed to be the best and most of the issues were resolved. The garbage collector ran less but still the stream stalled from time to time.

**It is highly recommended that the reader view the video file which shows the working implementation. Also,** Appendix I: Final implementation screenshots **for the final screenshots of the Android app.**

## 4.6. Server

The server uses a range of libraries to help it deliver a stream in a bidirectional manner. The base of the server was implemented using NodeJS®. NodeJS is a JavaScript runtime built on Google Chrome's V8 engine. For the purposes of this report NodeJS will be used a server and wherever the word 'server' is used, it refers to the NodeJS environment.

The reason why NodeJS was chosen as a server was because it is very fast, comparable to Java and C in some cases (Kraus, 2013), and it is written in JavaScript - a language that many developers understand, so developers in the future would understand it. A basic yet functional server can be written in just 30 lines of code, proving how easy it is to get started (McLaughlin, 2011).

NodeJS has a dependency manager called npm that hosts a large number of open-source, ready to implement, libraries. By typing in a few lines of code, these libraries are instantly embedded in an application.

It is through npm that a server-side WebSocket library was found called ws (ws, 2016). The library ws claims to be, "the fastest RFC-6455 WebSocket implementation for Node.js," (Primus, 2016). Not only is it fast, it's also performant as it was tested to work with up to 600,000 concurrent connections on a server with 4 CPUs and 15GB worth of RAM (Kleveros, 2015).

Note that ws was actually installed to work with Primus, a WebSocket abstraction layer to prevent lock-in to one specific WebSocket library, purely so that WebSocket libraries could be swapped in and out. The other WebSocket library that was tested was Socket.IO, but the transfer speed was slow when compared to ws and so ws was used.

Once ws was installed on the server, settings were changed so that it would stream binary instead of other forms of messages like Base64, which a long string of characters that would denote an image in this case.

The server was built with knowledge from the modules Distributed Systems and Computer Systems.

# 5. Testing

The system was tested on an iteration-by-iteration basis to see if it had any obvious errors that would prohibit a user in using the system. The testing was imposed by the Scrum methodology.

## 5.1. Iteration 0

As there was no prototype in form at this point, no testing was done.

## 5.2. Iteration 1

As the project was still improving on the prototype at this point, testing was done on the prototype that was in use. Here is a testing matrix that measured if the apps were working as intended.

| Problem/Question | Expected | Fix |
|---|---|---|
| Can Glass stream to the Android device without force closing? | Glass should be able to maintain its connection and not force close for any reason | No fix needed |
| Can the server accept connections? | Server is able to stream to/from each device | Not implemented as of yet |
| Is Glass able to stream to a remote peer? | Glass is able to stream to a remote peer | Not implemented as of yet |
| Can Glass stream with reasonable quality? | Optimal quality should be 720p | Glass cannot stream in 720p yet but only. Will focus on quality as prototype improves |
| Glass crashes when Android closes stream abruptly | Glass should handle the error gracefully | Added fix for Glass app that handles reconnects |
| Android app crashes when no Glass is available | The Android 'companion' device should handle cases when Glass is unavailable | Android app now shows dialog box if Glass is not found |
| Android app crashes when orientation is changed | Android app should handle all cases of orientation | Locked Android app into landscape mode |

## 5.3. Iteration 2

With a viable solution implemented testing was done at the end of the Iteration 2 to assess if it was in full working order.

| Problem/Question | Expected | Fix |
|---|---|---|
| Can the server accept connections? | Server is able to stream to/from each device | Server can accept connections |
| Is Glass able to stream to a remote peer? | Glass is able to stream to a remote peer | Server can stream to remote peer from Glass |

| | | |
|---|---|---|
| Can Glass stream with reasonable quality? | Optimal quality should be 720p | Glass can handle 640x360 only without stalling |
| Android app crashes when setting up Glass | Android app should make a connection to Glass and send data such as type of stream and name of user | Increase buffer size when sending data |
| Glass app has unnecessary slides | Glass wearer should be able to understand the flow of the Glass app naturally | Removed unnecessary slides in Glass app |
| User ends up on setup screens in Android app when stream has ended | User should be brought back to the homepage when finished streaming | Fixed issue by removing the setup screens from the back stack |
| User should not progress to streaming screen if Glass is not on the network | If Glass is not found, show a dialog stating exactly that | Added/fixed dialogs to many aspects of the setup process |

## 5.4. Iteration 3

Mainly tests stemming from broadcasting.

| Problem/Question | Expected | Fix |
|---|---|---|
| Is the broadcasting feature on the stream type menu selections? | Broadcasting mode should be available on the stream type menus | Added broadcasting mode |
| How many clients can connect to the server whilst broadcasting? | The server should be able to handle more than one client | Server was able to handle more than one client |
| Broadcasting mode does not follow the same setup process as the rest of the app | All stream types should follow the same process as each other | Note: Fixed in the next iteration. |

## 5.5. Iteration 4

Prototype testing.

| Problem/Question | Expected | Fix |
|---|---|---|
| Can Glass stream 1-to-1 and broadcast one after the other? | No matter what the type of stream a user selects, they should be able to switch to a different type of stream without the app crashing | No fix needed |
| Does the server wait for both peers to connect in a 1-to-1 stream? | The server should be able to leave a connection open for all parties to connect | Server has a concept of ‘rooms’ that enable the server to wait for all parties to connect |
| Glass wearer cannot launch Glass app without using touchpad | Glass app should launch on voice command | Glass app launches on a predefined voice command from Google |

| | | |
|---|---|---|
| Glass app does not have an IP address when Glass has not connected to network | Show user IP address of Glass and show prompt if user not connected to a WiFi network | Added prompt asking user to connect to a network to obtain an IP address |
| Glass app shows multiple IP addresses – this confuses user | Glass app should show one IP address that they enter into the Android app to connect to | Network interfaces can have more than one IP address so issue seems largely unavoidable |
| Is streaming to the server secure? | Streaming to/from the server should be secure | This can only be fixed with a valid HTTPS certificate |

# 6. Evaluation

To measure if the project was a success, evaluation took place. For a more personal evaluation please see Appendix A: Personal Reflection.

## 6.1. Feedback

The first step in evaluating if the project was successful is to seek feedback by the people who would most likely use the apps.

### 6.1.1. Medical student

As students will be using the app and the Glass app, feedback was conducted with the medical student in order to find out what she thought of the app.

1. In one part of the app, Glass' IP address needs to be entered. She felt that users would get confused with what an IP address is.
2. She thought that the whole idea was great. She could understand what Glass' role was.
3. She felt that using something like the GoPro would be too bulky.

### 6.1.2. Stakeholders

There were two opportunities where feedback was given by various stakeholders.

The first opportunity was delayed into partway through the third sprint. The delay in the meeting did not disrupt the project as the delay allowed time to test new features. See section 4.4.5.

When demoing the two apps for the first time, the feedback was overall very positive. They found the experience of streaming using Glass great. A questionnaire was given at the end of the demo session and it is provided in this report for viewing in Appendix H: Questionnaire.

To summarise the feedback, the stakeholders felt that the app was easy to use, rendered good quality on both Glass and the remote observer's screen and had a good design (one stakeholder particularly liked the colour scheme of the Android app). The speed was given the highest rating but "setup of the apps" was given a low score. Work was done to improve the setup process making it simpler in sprint 4 (section 4.5.2). The design of the apps was also reconsidered as there were some redundant pages in places.

The second opportunity of feedback was provided with two A&E doctors within LGI. No questionnaire was given but one doctor said he could see the system being used in emergency situations such as remote assistance where people would need the most help, especially if there were no doctors around.

They also thought that Glass would be best worn by the Team Lead (see section 4.5.4) rather than the surgeon because the surgeon was liable to positioning Glass incorrectly, as they found out during the scenario.

## 6.2. Glass' position

Where would Glass fit into the processes in an operating theatre? Would the surgeons have time to spend setting up Glass?

When researching how doctors would utilise Glass (in section 4.5.4 and 6.1.2), this report found that Glass had exactly one opportunity to be worn in the operating theatre. When the surgeon gets ready for an operation, they clean their hands and arms thoroughly. Ideally Glass would be worn and setup before this process of cleaning happens. The surgeon is already under pressure of the upcoming operation Glass would be unlikely to be actively worn and thought about. This report agrees with the sentiments found in section 6.1.2 where the Team Lead would be the best person to wear Glass as he/she has an overall perspective of what is occurring in each operating theatre. Outside the operating theatre, Glass can be worn at any point and so is suitable to be worn around on the Team Lead.

Inside the operating theatre there is a computer and various medical monitoring equipment. Glass can be streamed to the computer in the operating theater provided it and the computer are on the same network. The computer can then record or send on the stream depending on what the user has selected in the app.

## 6.3. Usability

This section aims to answer, given if the Scrum Team were to be unavailable, would the stakeholders be able to setup the system themselves?

The answer is a firm no. The final implementation is setup in such a way that the user needs to run the server on a computer in their own network. That involves setting the prerequisites and also running from the command line. Moreover, the user then needs to enter in the server's IP address into the settings page of the Android app. The remote peer also needs to know the server's IP address. All of which may be out of the user's technical knowledge to perform.

In practice, the user would not have to enter the server's IP address as it would have been predefined and hardcoded into the app.

## 6.4. Methodology

The methodology of Scrum helped the project along tremendously. The use of checklists and cards was a definite positive. As Scrum is an Agile methodology, it afforded the project with the room to change requirements into smaller forms. For instance, when designing the 1-to-1 feature of the app, many prototypes had to be developed to meet the requirements of 1-to-1. The project also used Kanban in the first sprint; Kanban allowed the Scrum Team to break down the requirements given by the Scrum Master into smaller ones. For example, the smaller requirements of broadcasting were targeted first with Kanban then allowed the project to focus on 1-to-1 later on.

The first sprint was one of high intensity. In total over 6 prototypes were discarded within the first four week sprint. As a result of incorporating Kanban into the first sprint, the project could mark everything that one prototype could do that the others could not. So for example, the USB implementation could "provide Glass with more battery life" but it could

not “provide bidirectional streaming.” If a prototype did not provide whatever the Sprint Backlog needed, then the prototype was discarded and the Kanban list was reset. It was Kanban that made it very simple to ‘reset’ the project.

# 7. Conclusion

This section discusses overall findings and further work that may be needed if another person was to continue work on this project.

## 7.1. Findings

Google Glass has limitations in terms of raw computing power. As is the norm with the wearable tech industry, Glass has slowly become outdated compared to some rival smart glasses, even though at the time of its release it had some good hardware. It seems Glass has lost its first-mover advantage through a combination of privacy and technical issues. With that being said, Google is rumoured to be working on an upgraded “Enterprise Edition” but exact dates for its release are unknown.

All in all, the implementation that this report developed has successfully met all the objectives as outlined at the start of this report. The apps work smoothly but improvements can definitely be made.

If this report were to choose a different pair of smart glasses to work with, it would be the Vuzix M300 as it seems Vuzix have iterated upon their designs and listened to feedback given by its customers.

If this report were to choose any device to work with on the market, it wouldn’t be any of them. A better alternative would be to create a pair of smart glasses afresh with the sole purpose of being a telemedicine device. Through the creation of this device, the creators can address all the limitations of Glass outlined in this report and position themselves as the gold standard of telemedicine. This is a very costly endeavour which is why smart glasses are not appearing on the market every other day.

## 7.2. Future work

The end implementation can be improved. This report will now theorise some possible work that could be done on the system that would help it to perform better.

### 7.2.1. Code optimisations

A person working on this project in the future will need a great depth of knowledge on how to optimise code so that the Android/Java garbage collector runs less often. The garbage collector is an automated tool that runs every so often to free memory. On Android, if a developer allocates too many bytes within a given period of time, the garbage collector will run and more than likely freeze the app. A person must understand when to allocate the right amount of bytes whilst also achieving maximum speed.

This report mentioned that RenderScript was used (see section 4.3.4) and it increased the number of times the garbage collector ran. This finding could be explored more in the future. Investing time into exploring the NDK would be of huge benefit as OpenCV could be integrated into the app. OpenCV is an open source computer vision library that has more than 2,500 algorithms (OpenCV, 2016).

### 7.2.2. Multithreading optimisations

Android gives developers the freedom of managing their own threads and use of threads within this project has been liberal. In high performance mobile apps, knowledge and use of threads would be a necessity, purely so that the app is able to do two things at once.

Although the apps are designed to be multithreaded, further time could be invested into making sure that the asynchronous threads are safe and robust.

### 7.2.3. Revisit UDP

Even though the implementation with UDP was discarded (section 2.7.2), it is worth noting some improvements to make for future work.

One improvement that could be made is to discard incomplete frames. To do this frames could be fixed in size and so the devices could successfully parse each frame. Failing this, use of RTSP and RTP would be another viable path. RTSP and RTP was tested with Libstreaming but a standalone implementation could be developed.

### 7.2.4. Revisit WebRTC

In sections 4.1.1 and 4.1.2, WebRTC was tested and found too slow and unusable. However, it was implemented by other people just fine and so needs further work confirming that WebRTC is unusable in Glass. One avenue of research is to use Data Channels (see section 2.7.5) instead of Media Streams. Data Channels can be used to transfer data directly to another peer. The fact that this report has already implemented a WebRTC and a WebSocket implementation and found them to work means that this should not be a time consuming venture.

As explained previously in this report, the way that the WebSocket implementation works is by sending frames and audio to a WebSocket server. Instead of transferring the data to a WebSocket server, transfer the data through WebRTC's Data Channels instead. This way, when a 1-to-1 session is called, the data does not flow through a server, but the data flows directly from each peer, meaning less time spent going to a server and then to the peer.

One major thing this report wants to highlight is that using WebRTC's Data Channels one can use UDP or TCP. It was explained earlier in this report that use of TCP is slow and UDP would be more preferential. This is why using Data Channels would be so beneficial because it offers that flexibility.

### 7.2.5. Check P2P

Although WebRTC enables peer-to-peer communication, it is not designed for Glass' limited capabilities. Another implementation of peer-to-peer could be to create a unique protocol that works specifically for Glass. This would be a time intensive implementation as the peer-to-peer implementation would also have to work on other platforms.

### 7.2.6. Improve server

The server currently has two implementations of a server that are identical. Namely, the audio and video server are almost identical. As the focus was on the apps rather than the

server (which can be swapped for another server type) the report did not improve the server enough. So one avenue of further work is to make sure that the server is brought up to the utmost correct standards of software engineering.

### 7.2.7. Cross platform considerations

The end implementation was done with Android being used on both the two devices: Glass and the companion device. It is not the case that Android is the only operating system the end implementation can run on. The companion app can be built atop of any platform. The end implementation is built in such a way that any (mobile) operating system can communicate with the Glass app and still see the streams[2].

Future work may be conducted to expand the number of devices that the Glass app can communicate with. Currently only Android was implemented but there is nothing restricting an implementation to be done on iOS, OSX or Windows.

### 7.2.8. Improve usability

This report found that there was a lack of usability in section 7.2.8. Further work needs to be done on the usability which can only take place if other items in this section are completed. For example, improving the server before placing it on the Internet for worldwide access, should be top on the list of priorities before improving the usability.

Usability can also be improved when streaming by adding menu buttons to mute/unmute a stream, to zoom in when using Glass, etc. The basics are there, all that needs doing is adding more features.

[2] Provided, of course, that the mobile operating system has the correct APIs.

# 8. References


Airtube. (2016, Feb 10). *Airtube*. Retrieved Feb 10, 2016, from GitHub: https://github.com/thinktube-kobe/airtube

Alaniz, A., & Behera, R. (2014, June 30). *Google Glass and WebRTC*. Retrieved March 20, 2016, from YouTube: https://www.youtube.com/watch?v=1w95PJLr8V4

Alvarez, W. (2014, Aug 09). *simple-signaling-server*. Retrieved May 01, 2016, from GitHub - satanas: https://github.com/satanas/simple-signaling-server

AMA. (2016, April 28). *AMA*. Retrieved April 28, 2016, from Xpert Eye: http://www.ama.bzh/en/kit-pc/

AMA. (2015). *Company Presentation*. Retrieved April 28, 2016, from AMA: http://www.ama.bzh/en/presentation-societe/

AMA. (2015, December 15). *Rennes : a world first and medical with Google Glass*. Retrieved April 28, 2016, from AMA: http://www.ama.bzh/en/rennes-une-premiere-mondiale-et-medicale-avec-des-google-glass/

Andronico, M. (2015, March 06). *New Android-Powered Smartglasses Now Available for $550*. Retrieved April 04, 2016, from tom's guide: http://www.tomsguide.com/us/sime-smartglasses-price,news-20642.html

Armstrong, D. (2014). *A Heads-Up Display for Diabetic Limb Salvage Surgery: A View Through the Google Looking Glass.* Arizona, US: Ovid Medline Embase Web of Science.

Augmedix. (2016, April 03). *Augmedix features*. Retrieved April 03, 2016, from Augmedix: http://www.augmedix.com/features/

Augmedix GitHub. (2013, June 03). *Augmedix GitHub*. Retrieved April 04, 2016, from GitHub: https://github.com/saiful-sdsl/WevRtcAndroidClient/

Augmedix Stack. (2016, April 04). *WebRTC Stack & API Overview - Augmedix*. Retrieved April 04, 2016, from Gliffy: https://www.gliffy.com/publish/5748292/

Baset, S. A., & Schulzrinne, H. G. (2004). *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol.* Department of Computer Science. New York: Columbia University.

BBC. (2015, January 15). *Google Glass sales halted but firm says kit is not dead*. Retrieved April 28, 2016, from BBC News: http://www.bbc.co.uk/news/technology-30831128

Benninger, B. (2015). *Google Glass, Ultrasound and Palpation: The Anatomy Teacher of he Future?* Clinical Anatomy.

BMA. (2016, March 19). *Life as a doctor*. Retrieved March 19, 2016, from BMA: http://www.bma.org.uk/developing-your-career/medical-student/how-to-become-a-doctor/life-as-a-doctor

Campbell, A. (2014). *Minimising Mobile Application Data Usage.* Leeds: Leeds University.

Campeau, M. (2014). *Through the LOOKING glass.* Canadian HR Reporter.

Castrounis, A. (2015, Feb 02). *What Is WebRTC and How Does It Work?* Retrieved April 28, 2016, from innoarchitect: http://www.innoarchitech.com/what-is-webrtc-and-how-does-it-work/

Chai, P., Babu, K., & Boyer, E. (2015). *The Feasibility and Acceptability of Google Glass for Teletoxicology Consults.* Journal of Medical Toxicology.

Chai, P., Wu, R., Ranney, M., Bird, J., Chai, S., Zink, B., et al. (2015). *The Feastibility and Acceptability of Google Glass for Emergency Department Dermatology Consultations.* JAMA Dermatology.

Chandler, N. (2015). *Google glass finds success in medical community.* New Orleans City.

Charlton, A. (2016, January 25). *Google Glass consumer brand name killed off as social-media accounts are shut down*. Retrieved February 2016, 2016, from IB Times: http://www.ibtimes.co.uk/google-glass-consumer-brand-name-killed-off-social-media-accounts-are-shut-down-1539894

ChipSiP. (2016, April 05). *SiME Smart Glasses*. Retrieved April 06, 2016, from ChipSiP: http://www.chipsip.com/computing/index.php?mode=data&id=126&top=60

Computer Hope. (2016, May 06). *Spaghetti code*. Retrieved May 06, 2016, from Computer Hope: http://www.computerhope.com/jargon/s/spaghett.htm

CrossWalk. (2016, February 10). *CrossWalk Homepage*. Retrieved from CrossWalk: https://crosswalk-project.org/

Denis Souto Valente, L. S. (2015, October 21). *Telemedicine and Plastic Surgery: A Pilot Study.* Retrieved February 8, 2016, from Plastic Surgery International: http://www.hindawi.com/journals/psi/2015/187505/

Durresi, A., & Jain, R. (2005). *RTP¸ RTCP and RTSP for Real Time Applications.* (R. Zurawski, Ed.) CRC Press.

epic.org. (2016). *Google Glass and Privacy*. Retrieved February 16, 2016, from EPIC: https://epic.org/privacy/google/glass/

Epson. (2016, April 06). *Epson AR*. Retrieved April 06, 2016, from Epson: http://www.epson.com/cgi-bin/Store/jsp/Landing/moverio-augmented-reality-smart-glasses.do?UseCookie=yes

Epson. (2016, April 05). *Moverio BT-200*. Retrieved April 05, 2016, from Epson: http://www.epson.ae/ae/en/viewcon/corporatesite/products/mainunits/overview/12411

FileZilla Wiki. (2016, April 29). *Network Configuration.* Retrieved April 29, 2016, from FileZilla Wiki: https://wiki.filezilla-project.org/Network_Configuration

Glass Almanac. (2016, April 20). *A History of Google Glass XE Software Updates*. Retrieved April 20, 2016, from Glass Almanac: http://glassalmanac.com/history-google-glass-xe-software-updates/

Glass Developers. (2016, April 21). *GDK Quick Start*. Retrieved April 21, 2016, from Google Developers: https://developers.google.com/glass/develop/gdk/quick-start#for_android_experts

Glass Developers. (2016, April 03). *User interface*. Retrieved April 03, 2016, from Google Developers: https://developers.google.com/glass/design/ui

Glauser, W. (2013). Doctors among early adopters of Google Glass. *CMAJ ¸ 185* (16), 1385.

Google. (2014, November 20). *MyGlass app*. Retrieved December 29, 2015, from Play Store: https://play.google.com/store/apps/details?id=com.google.glass.companion&hl=en

Google. (2014). *Tech specs*. Retrieved April 26, 2016, from Google Glass Help: https://support.google.com/glass/answer/3064128?hl=en

Hashimoto, D. A., Phitayakorn, R., & Castillo, C. F.-d. (2015). *A blinded assessment of video quality in wearable technology for telementoring in open surgery: the Google Glass experience.* Springer Science.

Heiting, G. (2016, Jan 01). *Contrast Sensitivity Testing*. Retrieved April 04, 2016, from All About Vision: http://www.allaboutvision.com/eye-exam/contrast-sensitivity.htm

Hildenbrand, J. (2013, December 17). *MyGlass iOS.* Retrieved April 20, 2016, from Android Central: http://www.androidcentral.com/my-glass-app-ios-7-appears-apple-s-appstore-quickly-pulls-it

Is WebRTC ready yet? (2016, March 19). *Is WebRTC ready yet?* Retrieved March 19, 2016, from Is WebRTC ready yet?: http://iswebrtcreadyyet.com/

Jorgensen, M. (2014, August 29). *What We Do and Don't Know about Software Development Effort Estimation*. Retrieved April 15, 2016, from InfoQ: http://www.infoq.com/articles/software-development-effort-estimation

Khan, F. (2016, January 25). *More evidence Google Glass for consumers is really dead*. Retrieved February 15, 2016, from Tech Radar: http://www.techradar.com/news/wearables/here-s-more-evidence-google-glass-for-consumers-is-really-dead-1313714

Kleveros, D. (2015, April 13). *600k concurrent websocket connections on AWS using Node.js*. Retrieved May 03, 2016, from JayWay: https://www.jayway.com/2015/04/13/600k-concurrent-websocket-connections-on-aws-using-node-js/

Kraus, S. (2013, May 01). *C, JAVA AND JAVASCRIPT NUMERIC BENCHMARK AND A BIG SURPRISE*. Retrieved May 01, 2016, from Stefan Kraus: http://www.stefankrause.net/wp/?p=144

Lee, N. (2014). *Surgical training through the looking Glass.* The Lancet Technology.

Libstreaming. (2016, February 09). *LibStreaming*. Retrieved from GitHub: https://github.com/fyhertz/libstreaming

Longley, C., & Whitaker, D. (2016). Google Glass Glare: disability glare produced by ahead-mounted visual display. *Ophthalmic and Physiological Optics , 36* (2), 167-173.

Martin, C. (2015, Jan 17). *Google Glass UK release date, price and specs: Google Glass will go offsale on 19 January; how to buy Google Glass* . Retrieved April 03, 2016, from PcAdvisor: http://www.pcadvisor.co.uk/feature/gadget/google-glass-release-date-uk-price-specs-3436249/

Maven. (2016, April 28). *Libjingle Peerconnection*. Retrieved April 28, 2016, from Maven Repository: http://mvnrepository.com/artifact/io.pristine/libjingle

McGee, M. (2015, Jan 15). *Google Glass Is Dead? Google Glass Is Reborn? Yes. Both.* Retrieved April 04, 2016, from Marketing Land: http://marketingland.com/google-glass-dead-google-glass-reborn-yes-114522

McLaughlin, B. (2011, July 06). *What is Node.js?* Retrieved May 02, 2016, from Radar: http://radar.oreilly.com/2011/07/what-is-node.html

MDN. (2016, March 25). *MediaStream API*. Retrieved March 25, 2016, from Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API

Medical Schools Council. (2016, January 1). *Career pathways*. Retrieved March 19, 2016, from Med Schools: http://www.medschools.ac.uk/Students/careers/Pages/Career-Pathway.aspx

Merriam Webster. (2016, March 10). *toxicology*. Retrieved March 10, 2016, from Merriam Webster: http://www.merriam-webster.com/dictionary/toxicology

Mississippi State University. (2015). *USB Lecture*. Retrieved 2016, from Mississippi State University: http://my.ece.msstate.edu/faculty/reese/EE3724/lectures/usb/usb.pdf

N3. (2016, April 25). *N3*. Retrieved April 25, 2016, from N3: http://n3.nhs.uk/

netscan. (2015, Dec 09). *WebRTC Security: Just How Secure is Web Real Time Communication?* Retrieved March 19, 2016, from netscan: https://www.netscan.co/blog/webrtc-security-just-how-secure-is-web-real-time-communication/

NewsRx. (2015). *Vuzix Corporation; Vuzix Partners With HeadApp To Launch Pilot Program For Its Award-Winning M100 Smart Glasses.* Atlanta: NewsRx.

OpenCV. (2016, May 03). *About OpenCV*. Retrieved May 03, 2016, from OpenCV: http://opencv.org/about.html

Optinvent. (2016, April 05). *ORA-2*. Retrieved April 05, 2016, from Optinvent: http://www.optinvent.com/produit/ora-2

Osterhout Group. (2016, April 05). *R-7 Smartglasses*. Retrieved April 05, 2016, from Osterhout: http://www.osterhoutgroup.com/products-r7-glasses

P Russel, M. M.-H. (2014). *First "Glass" Education: Telementored Cardiac Ultrasonography Using Google Glass - A Pilot Study.* Academic Emergency Medicine.

Paro, J. A., Nazareli, R., Gurjala, A., & Lee, G. K. (2015). *Video-based self-review: comparing Google Glass and GoPro technologies.* Comparative, Stanford University, Division of Plastic Surgery.

Patel, N. (2010, April 05). *Know Your Rights: H.264, patent licensing, and you*. Retrieved April 29, 2016, from Engadget: http://www.engadget.com/2010/05/04/know-your-rights-h-264-patent-licensing-and-you/

Paul, I. (2010, Oct 26). *Wi-Fi Direct vs. Bluetooth 4.0: A Battle for Supremacy*. Retrieved May 03, 2016, from PCWorld: http://www.pcworld.com/article/208778/Wi_Fi_Direct_vs_Bluetooth_4_0_A_Battle_for_Supremacy.html

Peepers. (2013, February 13). *Peepers: realtime video streaming from Android*. Retrieved March 10, 2016, from Foxdog Studios: https://foxdogstudios.com/peepers

pioneers.io. (2013, August 27). *Google Glass and Healthcare – an Interview with Pristine.io*. Retrieved March 20, 2016, from Pioneers: https://pioneers.io/blog/2013/08/27/google-glass-and-healthcare-an-interview-with-pristine-io/

Primus. (2016, May 02). *Primus*. Retrieved May 02, 2016, from GitHub - Primus: https://github.com/primus/primus

Pristine. (2016, March 20). *About Pristine*. Retrieved March 20, 2016, from Pristine: https://pristine.io/about-pristine/

Pusher. (2016, March 20). *What are websockets?* Retrieved March 20, 2016, from Pusher: https://pusher.com/websockets

Qualcomm. (2016, April 05). *Snapdragon 805 Processor*. Retrieved April 05, 2016, from Qualcomm: https://www.qualcomm.com/products/snapdragon/processors/805

Recon Instruments. (2016, April 06). *Recon Jet*. Retrieved April 06, 2016, from Recon Instruments: http://www.reconinstruments.com/products/jet/

Rennes Atlanta. (2014, March 07). *A world 1st in Rennes: a surgeon connected to Japan via Google Glass*. Retrieved April 28, 2016, from Rennes Atlanta: http://www.rennes-atalante.fr/en/science-park-news/science-park-news/news-details/a-world-1st-in-rennes-a-surgeon-connected-to-japan-via-google-glass.html

Respoke. (2016, March 18). *What is WebRTC?* Retrieved March 18, 2016, from Respoke: https://www.respoke.io/webrtc/

Ristic, D. (2014, Feb 04). *WebRTC data channels*. Retrieved March 25, 2016, from Html5Rocks: http://www.html5rocks.com/en/tutorials/webrtc/datachannels/

Sams, R. J. (2011, Feb 09). *Introducing Renderscript*. Retrieved May 01, 2016, from Android Developers Blog: http://android-developers.blogspot.co.uk/2011/02/introducing-renderscript.html

Sarpu, B. A. (2014). Google: The Endemic Threat to Privacy. *Journal of High Technology Law , 15* (1), 97-134.

Sherwin, A. (2014, June 28). *Google Glass to be banned from all UK cinemas*. Retrieved April 28, 2016, from Independent: http://www.independent.co.uk/life-style/gadgets-and-tech/news/google-glass-to-be-banned-from-all-uk-cinemas-9570686.html

Sony. (2015, February 17). *Sony Releases the Transparent Lens Eyewear "SmartEyeglass Developer Edition"*. Retrieved March 20, 2016, from Sony News: http://www.sony.net/SonyInfo/News/Press/201502/15-016E/

SparkFun. (2016, February 7). *Bluetooth Basics*. Retrieved from SparkFun: https://learn.sparkfun.com/tutorials/bluetooth-basics/common-versions

Stop the Cyborgs. (2016, April 04). *Google Glass Ban*. Retrieved April 04, 2016, from Stop the Cyborgs: https://stopthecyborgs.org/google-glass-ban-signs/

Strickland, E. (2014, August 19). *Start-up Profile: Pristine Is Bringing Google Glass to the Hospital*. Retrieved March 20, 2016, from IEEE Spectrum: http://spectrum.ieee.org/at-work/start-ups/startup-profile-pristine-is-bringing-google-glass-to-the-hospital

Swanner, N. (2015, July 13). *Google Glass may finally end its embarrassing consumer life and get a job in enterprise*. Retrieved February 15, 2016, from The Next Web: http://thenextweb.com/google/2015/07/13/google-glass-may-finally-end-its-embarrassing-consumer-life-and-get-a-job-in-enterprise/#gref

Syme, M., & Goldie, P. (2004, March 05). *Understanding Application Layer Protocols*. Retrieved May 01, 2016, from informIT: http://www.informit.com/articles/article.aspx?p=169578&seqNum=3

Tanveer, M. I., & Hoque, M. E. (2014). *A google glass app to help the blind in small talk.* Proceedings of the 16th international ACM SIGACCESS conference on Computers & accessibility. New York: ACM.

TD, A., & TL, L. (2015). *Potential uses of wearable technology in medicine: lessons learnt from Google Glass.* International Journal of Clinical Practice. Wiley Subscription Services, Inc.

Thinktube. (2016, February 8). *Airtube*. Retrieved from GitHub: https://github.com/thinktube-kobe/airtube

Third Eye Health. (2016, April 28). *Third Eye Health*. Retrieved April 28, 2016, from Third Eye Health: https://thirdeyehealth.net/marketing/

Tully, J., Dameff, C., & Kaib, S. (2015). *Recording Medical Student's Encounters with Standardized Patients using Google Glass: Providing end-of-life Clinical Evaluation.* Academic Medicine.

Vital. (2016, April 28). *About Vital*. Retrieved April 28, 2016, from Vital: https://www.vital.enterprises/about

Vital Enterprises. (2014, April 2014). Hands-free vital sign monitoring on Google Glass.

Vital Enterprises. (2015, September 14). *Vital Enterprises - Introduction to Workflow & Team Support*. Retrieved April 28, 2016, from YouTube: https://www.youtube.com/watch?v=YNkbcfF43-U

Vorraber, W. e. (2014). Medical applications of near-eye display devices: An exploratory study. *International Journal of Surgery , 12* (12), 1266-1272.

Vuzix. (2016, April 20). *M100 Smart Glasses*. Retrieved April 20, 2016, from Vuzix: https://www.vuzix.com/Products/m100-smart-glasses

Vuzix. (2016, Jan 01). *Vuzix M300 Smart Glasses.* Retrieved April 14, 2016, from Vuzix: https://www.vuzix.com/Content/pdfs/Vuzix-M300-Product-Sheet-01-01-2016.pdf

Watson, M. S. (2015). *The Leeds Method of Management Antibiotic App.* School of Computing. Leeds: University of Leeds.

Wearable Tech. (2016, April 04). *Vuzix rolls out SDK for M300 Smart Glasses to foster app development*. Retrieved April 14, 2016, from Wearable Tech: http://www.wearabletechnology-news.com/news/2016/apr/04/vuzix-rolls-out-sdk-m300-smart-glasses-foster-app-development/

WebRTC. (2016, April 29). *Android - WebRTC*. Retrieved April 29, 2016, from WebRTC: https://webrtc.org/native-code/android/

WebRTC. (2016, February 10). *WebRTC homepage*. Retrieved from WebRTC: https://webrtc.org/

Williams, M. (2014, Sep 04). *Sony shows off latest SmartEyeglasses prototype*. Retrieved May 01, 2016, from PcWorld: http://www.pcworld.com/article/2603340/sony-shows-off-latest-smarteyeglasses-prototype.html

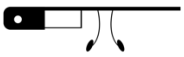
ws. (2016, May 03). *ws*. Retrieved May 03, 2016, from GitHub - ws: https://github.com/websockets/ws

Xamarin. (2016, April 05). *Understanding Android API Levels*. Retrieved April 05, 2016, from Xamarin: https://developer.xamarin.com/guides/android/application_fundamentals/understanding_android_api_levels/

# 9. Glossary

| | |
|---|---|
| **Telementoring** | The remote guidance of an inexperienced individual during an operation or procedure involving live, two-way audio-visual communication. The goal of telementoring is to recreate face-to-face encounters with a digital presence. |
| **Telemedicine** | Telementoring within the context of medicine. |
| **Contrast Sensitivity** | How the eye distinguishes between objects and their background, especially in environments of low visibility, fog or glare (Heiting, 2016). |
| **NHS Trusts** | Provides secondary health care within the National Health Service. |
| **Secondary health care** | Primary health care is the first point of contact between a person and doctors e.g. GP Surgeries. Secondary health care are places that primary health care doctors send people to for more specialised knowledge e.g. Hospitals. |
| **WebRTC** | Stands for Web Real Time Communication. Allows secure encrypted peer-to-peer connections to be made. Audio, data and video can all be streamed. |
| **USB OTG** | USB On The Go enables smartphones to power external devices. |
| **Scenarios** | A trained actor (or someone else) will act like a patient so that doctor's are adequately trained. |

# 10. Attributions



Google Glass image provided under the Creative Commons license. No changes were made. Created by Damion.

The Android robot is reproduced from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.

Laptop and smartphone image provided under the IconsMind licence. No changes were made. Created by IconsMind.com.

| | |
|---|---|
| **Sony Smart EyeGlass** | No infringement intended. |
| **Vuzix®** | No infringement intended. |
| **ODG R-7** | No infringement intended. |
| **Apple®** | No infringement intended. |
| **GoPro™** | No infringement intended. |

# 11. Appendix A: Personal Reflection

Overall, I had loads of fun working on this project. There were no major setbacks but the project itself was very challenging technically. It required in-depth knowledge on Java, Android, the protocols used on the web and also a huge amount of patience.

No doubt that the project was a lot of work but I never thought that the write-up would take so long to complete. If you're planning for your final year project definitely make sure that you've taken into account the write-up of the report. Likewise, make sure to write-up accurate estimations at the start of the project, because those will be used to give you an idea on how much you've got left to do.

There were times when I thought I would not be able to find a solution, but remaining optimistic, I kept searching until I got one. If all did not go to plan, I wrote a contingency plan. Having this written down provided me with a sense of security that gave me an idea that I could still populate this report even if I didn't get the main objectives done.

I found that starting early helped me understand the project a lot more. Starting early gave me time to understand the terminology in medicine and helped me gain a deeper understanding of the requirements before I started implementing a prototype. Having an understanding of the terminology then translated into being able to talk to the stakeholders in meetings, which then made it easier for both me and them to build rapport. With that in mind, if your project does have stakeholders involved then take a notepad to all your meetings and write tonnes of notes! Those notes will save you time and time again when it comes to writing your report.

I think the worst thing a student could do when starting their final year project is to instantly become negative about it. Sure, it's a lot of work and sure it would take a long time but it's your chance show the world what you have learnt. It's your opportunity to learn something you've never learnt before. In that sense, this project is definitely a positive.

In the end, all the objectives that were given to me at the start of this report have been implemented and I am happy that I have gained loads of knowledge in the process, met some brilliant people and did my best. So, to sum up, there are three things that I would advise a student do:

1. Work hard.
2. Learn a lot.
3. Be optimistic.

# 12. Appendix B: Materials Used

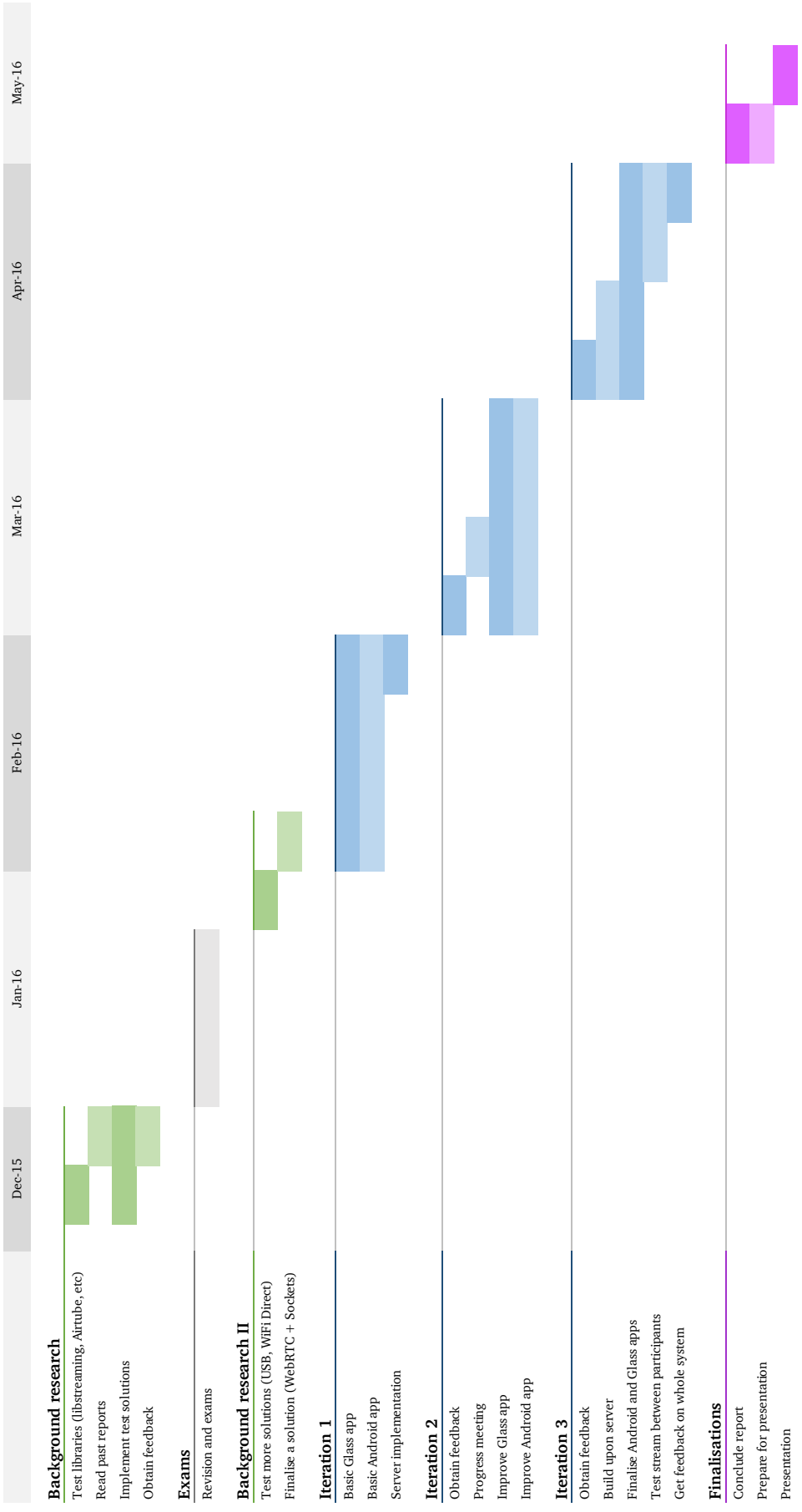
No materials were used prior to this report's writing.

# 13. Appendix C: Ethical Issues

All participants agreed verbally that they would like to be a part of the report. Where possible, agreements were made in writing.

In the future, people who would be recorded (in a hospital for example) whilst a prototype like the one in this report is in use, would most likely have signed a consent form or at the very least been made aware of a recording in progress.

# 14. Appendix D: Gantt Chart



# 15. Appendix E: Initial Objectives

The first meeting was given before this project's official start date. Below are some rough notes that were written in that meeting.

Requirements

- Record a patient interaction – download that interaction and analyse it to examiners – quality assure health care pro is trained. Linear process driven steps, human factors with patient interaction.
- Remote decision support – get expert advice on a patient interaction.
- More important to have a camera connected to an earpiece that may be connected to a phone. It's a good thing for the other person to speak to you and them to see you and you need to hear them.
- Be able to send stream to many people from Glass.
- Be able to send stream to one person from Glass.
- In a major incident people give action cards – decision support – latest support timed actions have you done this and this shown on the screen.

Tool on the laptop that may become an educational tool. Tablet connected to Glass maybe.

# 16. Appendix F: Contingency Plan

Below is the contingency plan.

1. **Are there people that could point you in the right direction?**
   This is important to note. Not every research paper has the answer. Ask around for advice on common technologies.

2. **Work on collating a list of technologies that are beneficial to the context of this report.**
   If there is no progress after two sprints then make sure you compare and contrast technologies that could be useful. Outline where they could fit into the procedures in a hospital.

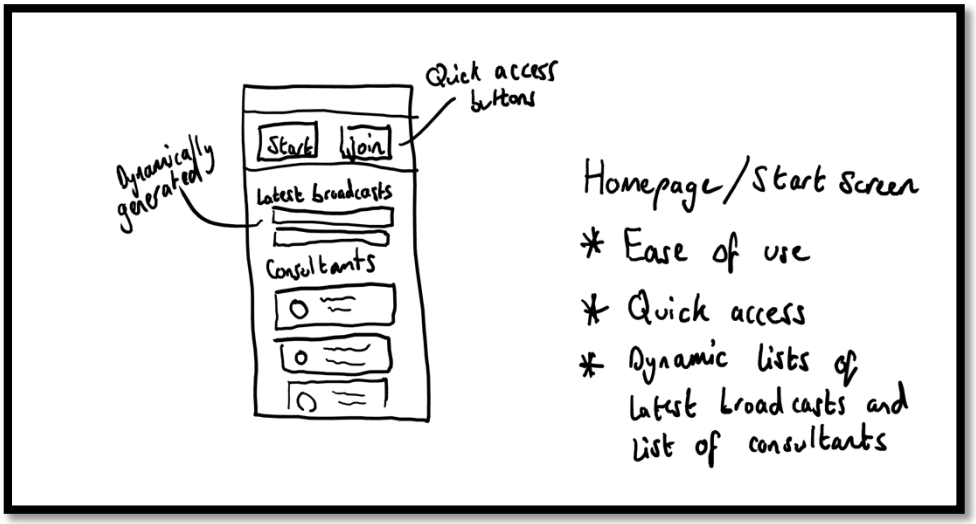
3. **Book sessions with doctors and general practitioners to see if this concept is beneficial to them.**
   Research exactly where Glass can fit into the procedures at a surgery. This process is not well outlined especially in technology based papers that. Readers and future reviewers can see the the benefit therefore.

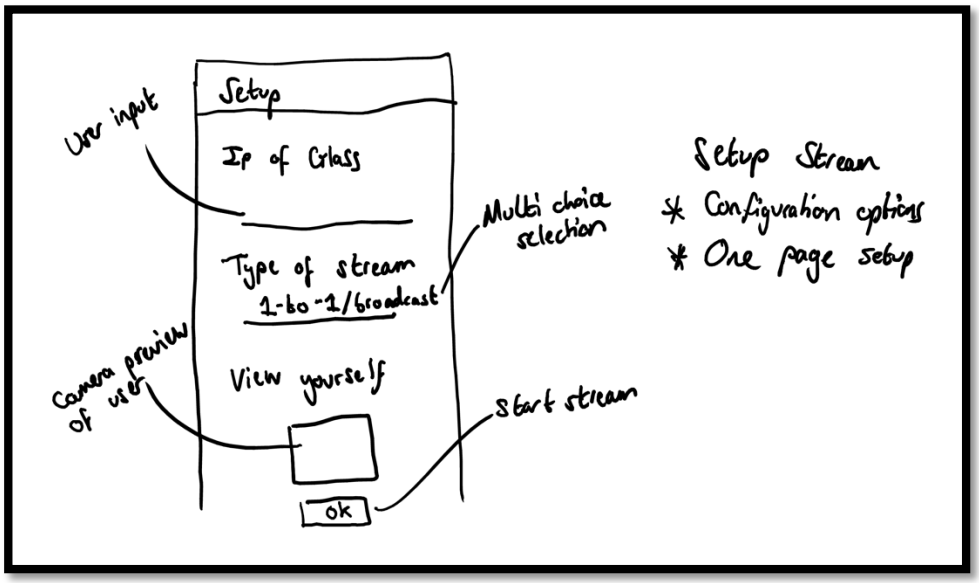
4. **Research existing businesses that are using Glass.**

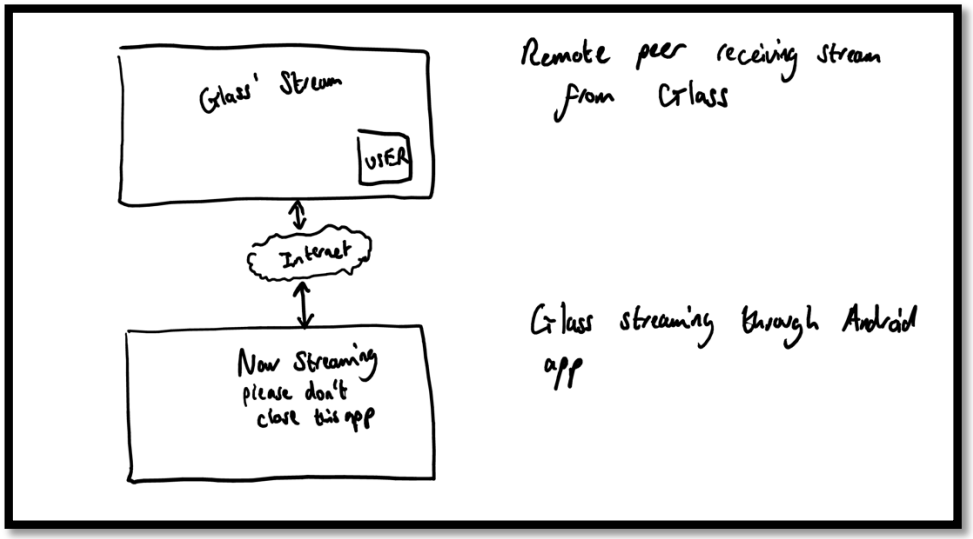
5. **Find alternatives to Glass.**
   Glass is not the sole player in the head wearable market. Find some other smart glasses that are comparable and try them to see if they can work to meet the objectives of this report.

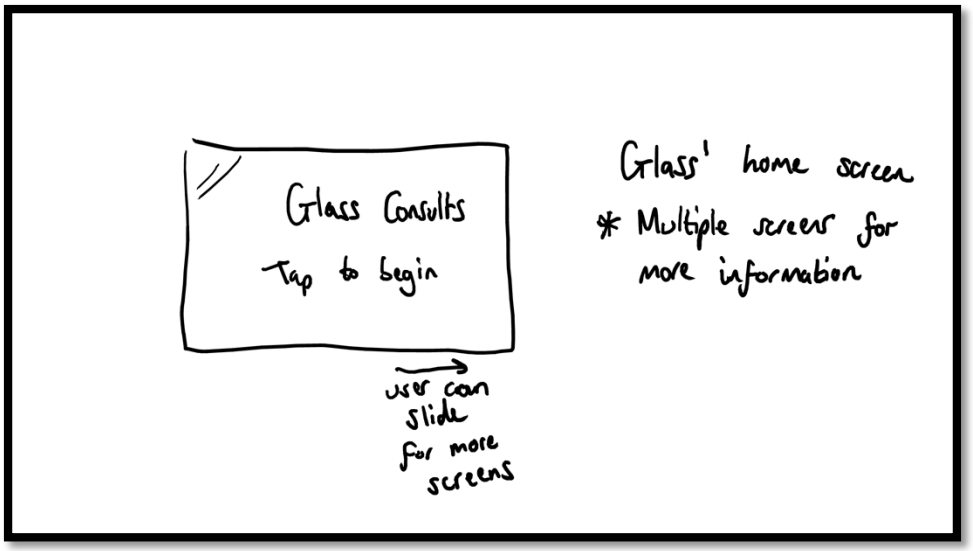
6. **See how Glass' built-in recording feature can help people in the professions unrelated to medicine.**
   Glass can record video but how useful is it for non-medicine based applications? This would give an indication as to how Glass would fare in markets wear hands-free recording would be advantageous.

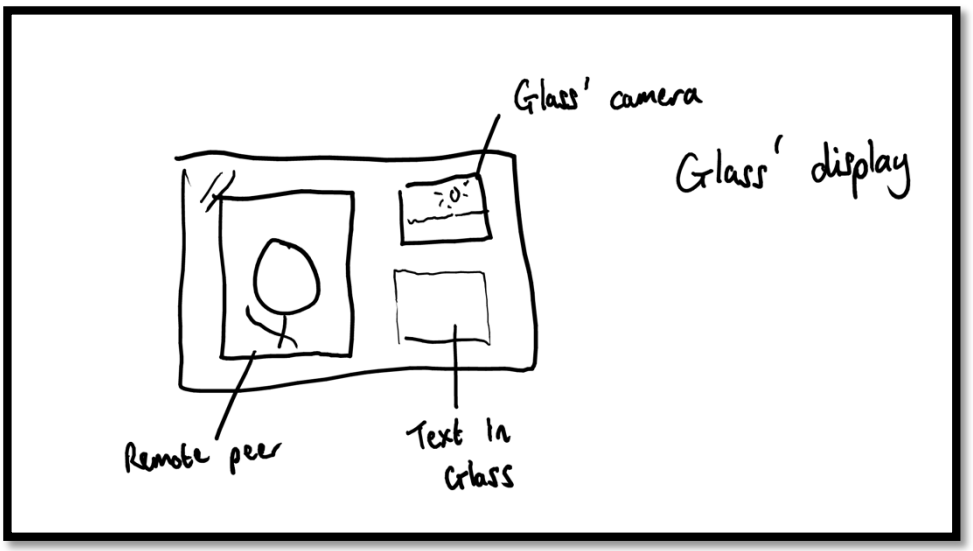
# 17. Appendix G: Low Fidelity Prototype




Glass' Stream
USER
Internet
Now Streaming
please don't
close this app
Remote peer receiving stream
from Glass
Glass streaming through Android
app


Glass Consults
Tap to begin
user can
slide
for more
screens
Glass' home screen
* Multiple screens for
more information


Glass' camera
Glass' display
Remote peer
Text in
Glass

# 18. Appendix H: Questionnaire

The following is a questionnaire that was filled in by stakeholders to gauge how functional the system is. The underlined number indicates their preference.

1. How easy was the Glass app and the Android app to use? Choosing 1 indicates that the apps were difficult to use and choosing 5 indicates that they were easy to use.

   1 2 3 <u>4</u> 5

2. How was the quality of the image in Glass? 1 being very bad and 5 being very good.

   1 2 3 <u>4</u> 5

3. How was the quality of the image of the remote peer? 1 being very bad and 5 being very good.

   1 2 3 <u>4</u> 5

4. How fast was the stream? 1 being very slow and 5 being very fast.

   1 2 3 4 <u>5</u>

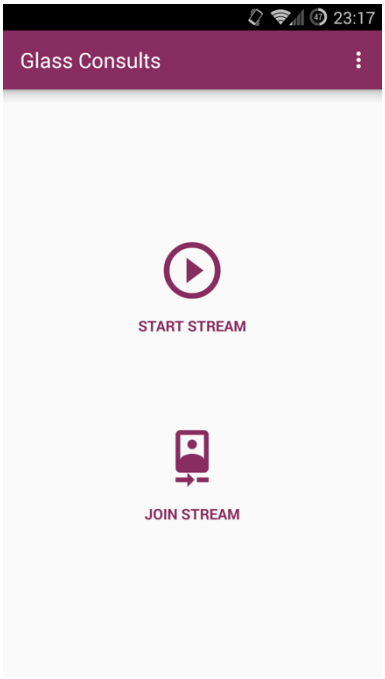
5. How easy was it to setup? 1 being difficult, 5 being easy.

   1 2 <u>3</u> 4 5

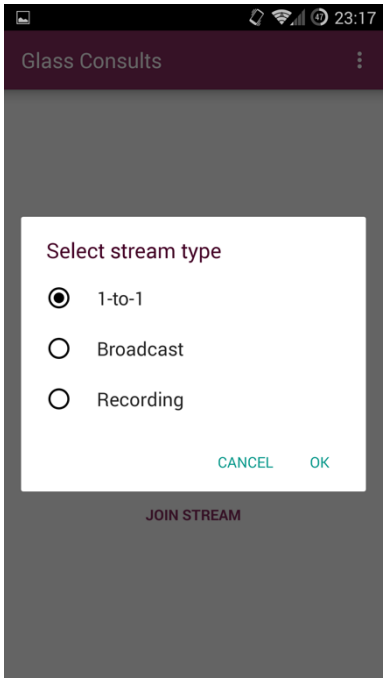
6. In your opinion, was the app well designed? 1 being not well designed at all and 5 being excellent design.

   1 2 <u>3</u> 4 5

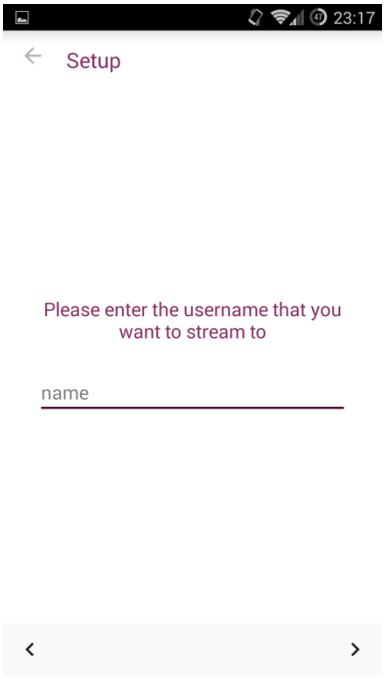
# 19. Appendix I: Final implementation screenshots




*Homepage*

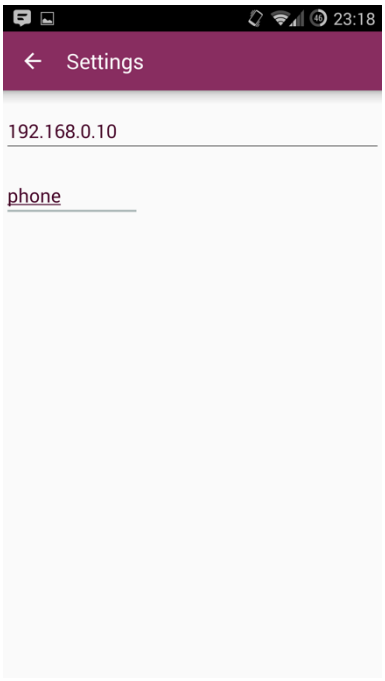



*Stream type*




*Setup stream I*


*Setup stream II*




*Settings*