University of Brighton

Project Proposal

WhisPeerer – a WebRTC messaging application

05/01/2016

Dominic Rathbone
Student number: 12843140

CI360 – Mobile Application Development
School of Computing, Engineering and Mathematics

Summary

My android application will allow two users to message and voice/video chat directly with each other without the need for an intermediary server. To do so, my application will utilise WebRTC, a new technology aimed to provide peer-to-peer connectivity to the web. By avoiding the need for an intermediary server, it provides a service with greater anonymity and security which a lot of people find important in an age where digital privacy has become such a concern.

Rationale

On a shallow level, my android application aims to provide a free internet-based service to users who want to to avoid the cost of using mobile networks to message and video/voice chat with each other.

Nowadays, many consumers that use these types of services have to place their trust on the third party providers running them to securely store and use their data. However, there is inherent data privacy and security issues associated with client-server architectures used by these providers. These problems are growing larger in the domain of mobile applications as many deploy a server based infrastructure to take the stress of data processing away from client devices, this is shown by the current OWASP (Open Web Application Security Project) top 10 mobile security concerns with the first on list is “Weak server side controls” [1]. On a deeper level, my application is being created to provide a method of communication and data exchange that avoids these issues.

These issues have become more apparent in the mainstream media in recent years with consumer's application data not only being susceptible to these illegal and unethical attacks [2] but also becoming susceptible to legal, intentional access with bills such as the “Investigatory Powers Bill” being drafted by the UK government which aims to force mobile and internet companies to be able to decrypt user's data on request [3]. In this case, my application also serves users who are becoming more aware of these issues and want to avoid the possibility of third parties inappropriately using the data being stored on their servers.

Application

At the moment, the largest messaging platforms such as WhatsApp and Facebook Messenger both provide mobile applications for their services. From a normal user's perspective, it is hard to criticise these two applications as they have a great user interface and range of functionality, this being shown by their user bases of 900 million [4] and 700 million [5] respectively. However, they are liable to criticism from a technical perspective.

Although WhatsApp has implemented VoIP functionality through a library called PJSIP, it is not as advanced as WebRTC in terms of protocols used [6]. On top of this, the majority of it's functionality is based on a client-server architecture which, again, is vulnerable to some of the issues previously described with WhatsApp even being a target of a proposed ban by the government for their use of strong data encryption protocols [7].

In contrast to this, there is evidence Facebook Messenger implemented WebRTC for it's mobile video and voice calling early on in 2015. Although this is a good example of how WebRTC can be applied, Facebook Messenger has a security flaw in it's implementation in that it uses the SDES (Session Description Encryption Security Descriptions) encryption standard instead of DTLS (Datagram Transport Layer Security) which is the new standard encryption scheme for WebRTC. The problem with this is "*the encryption keys are sent via the signaling servers and can be used to retroactively decrypt traffic*" [8]. This could potentially mean that, for example, this service would be liable to access by the government in the case that the "Investigatory Powers Bill" is passed even though it is using technology which should avoid this.

One application that provides an alternative to these is Sicher which aims to provide a free end-to-end encrypted instant messaging service. From their FAQ, it states it uses "*point-to-point encryption, based on asymmetric cryptography. It means that only the recipient who owns the private key can decrypt the message. RSA cryptosystem is used with 2048 bit keys. Additionally all data exchange between mobile apps and Sicher servers is protected using SSL*" [9]. Although, it does exchange this encrypted data through a server, it reduces the risk of this by immediately deleting this data off their servers when the recipient receives it [10]. However, the level of encryption that it bases it's product on could again suffer if the "Investigatory Powers Bill" was passed and they were forced to downgrade. Another minor criticism of mine is that the user interface is relatively immature and it has only been designed for use with a mobile phone as opposed to larger devices such as tablets.

A criticism of all of these applications is that they lack transparency in terms of their source code due to the fact that they are proprietary. By using an application that is open source, the level of transparency almost guarantees that there is little room for inappropriate storage and usage of the data that goes through it. One messaging application like this is ChatSecure. An application that aims to provides symmetric OTR (off the record) encryption, which is a method of encryption specific to real time messaging, this works by using a new key for each and every message sent by an application.

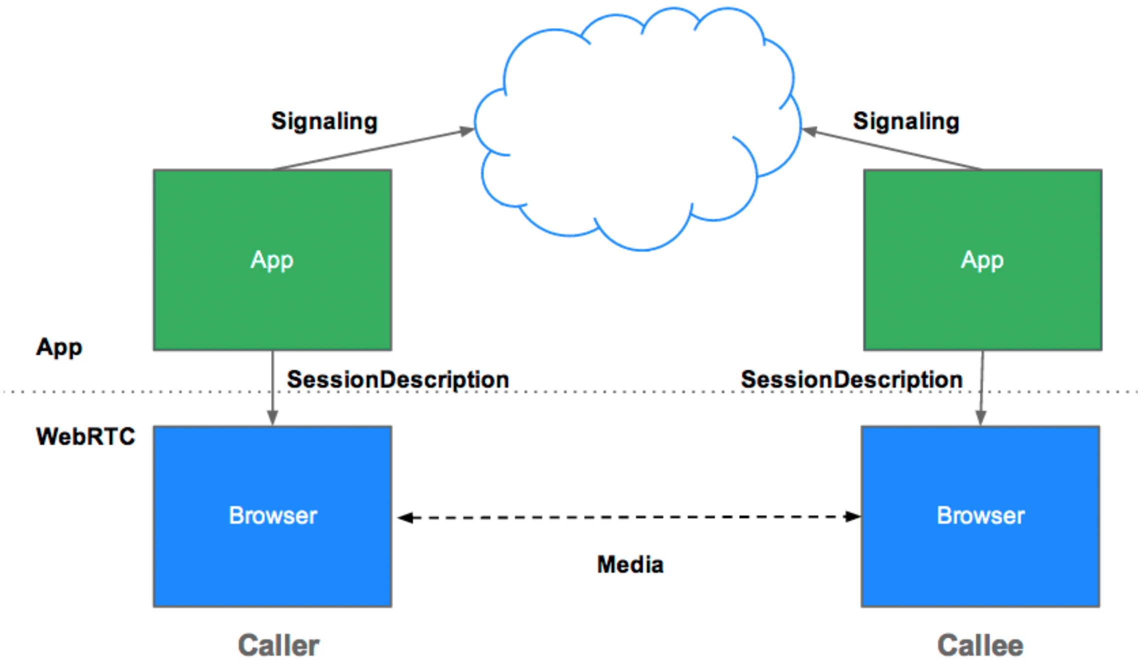
My application would provide an alternative to traditional instant messaging applications such as these by completely basing it around the WebRTC (Web Real Time Communication) native API for Android. This API allows the an instance of the application to form a peer to peer connection with another instance to directly exchange data over. From the user's perspective, this connection gives them the ability to stream data such as text, video and audio to and from one another. By using this alternative architecture, it avoids the data passing through a third party server and avoids some of the security and privacy issues associated with the applications mentioned previously.

Scenario of use

A use case for my application would be someone (referred to as user A) who wants to communicate with their friend (referred to as user B) anonymously. They would open up the application and enter a temporary name to identify themselves by. From the application, they would then share this name with user B by email (amongst the various other forms of sharing my application would allow for e.g. Text, Facebook Messenger). After user B receives this email, user B would go to the application, click “join chat” and enter user A's name, sending a chat request to User A. This would trigger a notification on user A's application with their request, once user A has accepted the request, they enter a new chat room and either of them could either trigger a video/voice chat or just send plain text messages to each other. After they are done and the users would disconnect from each other and the room joining them on the signalling server would be disposed of.

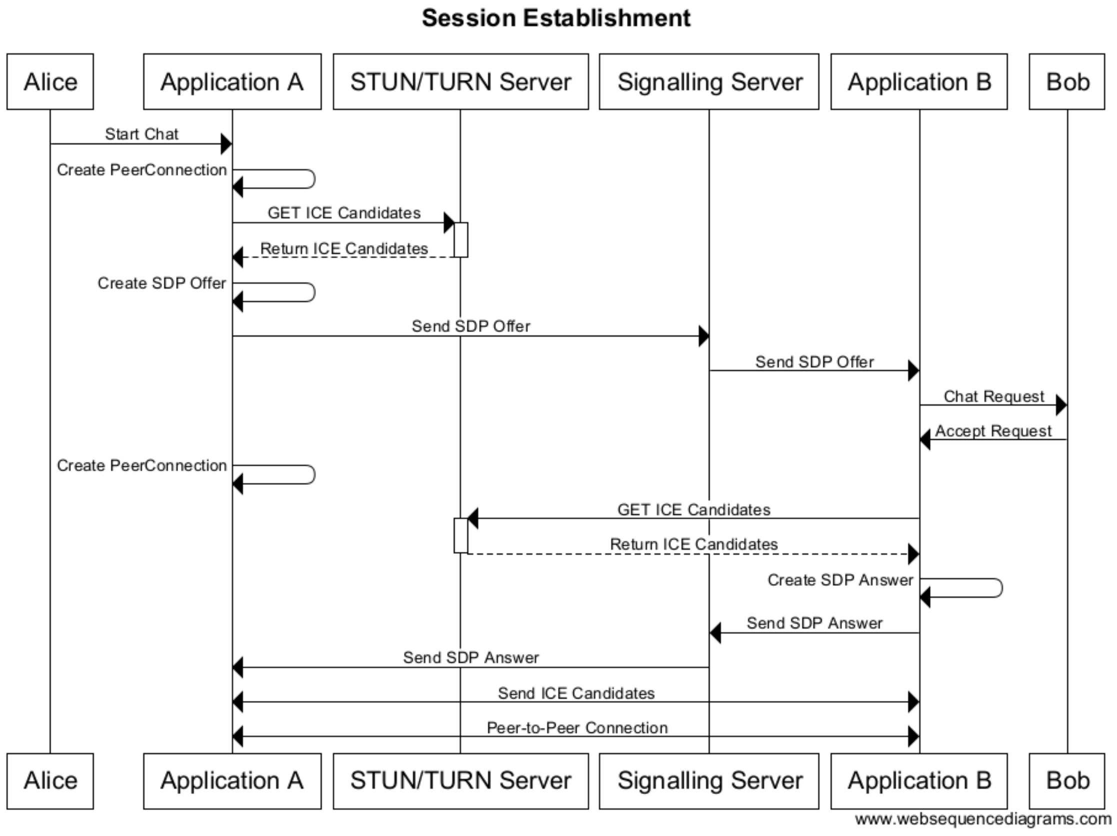
Technical overview

The basis of the application will be LibJingle, a C++ library that implements the W3C WebRTC API definition that can be compiled for use with Android, for this application, a precompiled version will be used. In order for a connection to be formed between two applications, it uses an architecture called JSEP (JavaScript Session Establishment Protocol). However, in this case with the LibJingle library, the browser would be replaced by the user's Android client.



As you can see in the diagram, the applications needs a method of signalling to establish a session, normally a server but this is purposefully left up to the developer so they can retroactively fit WebRTC into their current architecture if they already have a signalling solution. In our case, the method of signalling will be via a Node.js server. This signalling server is used to exchange client meta data in order to establish the session and will not touch any of the data sent via the peer-to-peer connection. To establish this session, this server needs to use a communications protocol that can send data bi-directionally between client and server such as WebSockets. To implement WebSockets, the Socket.io library for Node.js will be used.

To model the body of the messages exchanged during session establishment, SDP (Session Description Protocol) is used. The flow of session establishment is as follows:



Before everything, a PeerConnectionFactory object is instantiated, this will represent the connection between the two applications and from it, streams can be added, offers and answers can be created and various other things can be achieved. The offer and the answer contain meta data such as the media streams that have been added to it, a list of media constraints that describe what they have allowed access to (e.g. video & audio sources) as well as a list of ICE Candidates that have been retrieved before the offer was sent. ICE (**Interactive Connectivity Establishment)** candidates are a list of IP addresses and ports that the offer is potentially being sent from, once these have been exchanged, the receiving application will attempt to form a connection with each of these until it finds one that works. This is a technique that allows the application to avoid issues caused by network address translation. It gets these candidates by hitting an external server (STUN or TURN) that returns the client's external IP back to themselves. As this process is asynchronous, the applications can send new ICE candidates to each other after the offer or answer has been sent, in the case they weren't added beforehand (if the server takes too long to respond and the callback doesn't happen in time). For our application, we will use Google's public STUN server to retrieve potential candidates as it is free and reliable. The peer to peer connection uses the DTLS-SRTP protocol to encrypt the data traveling between the two users.

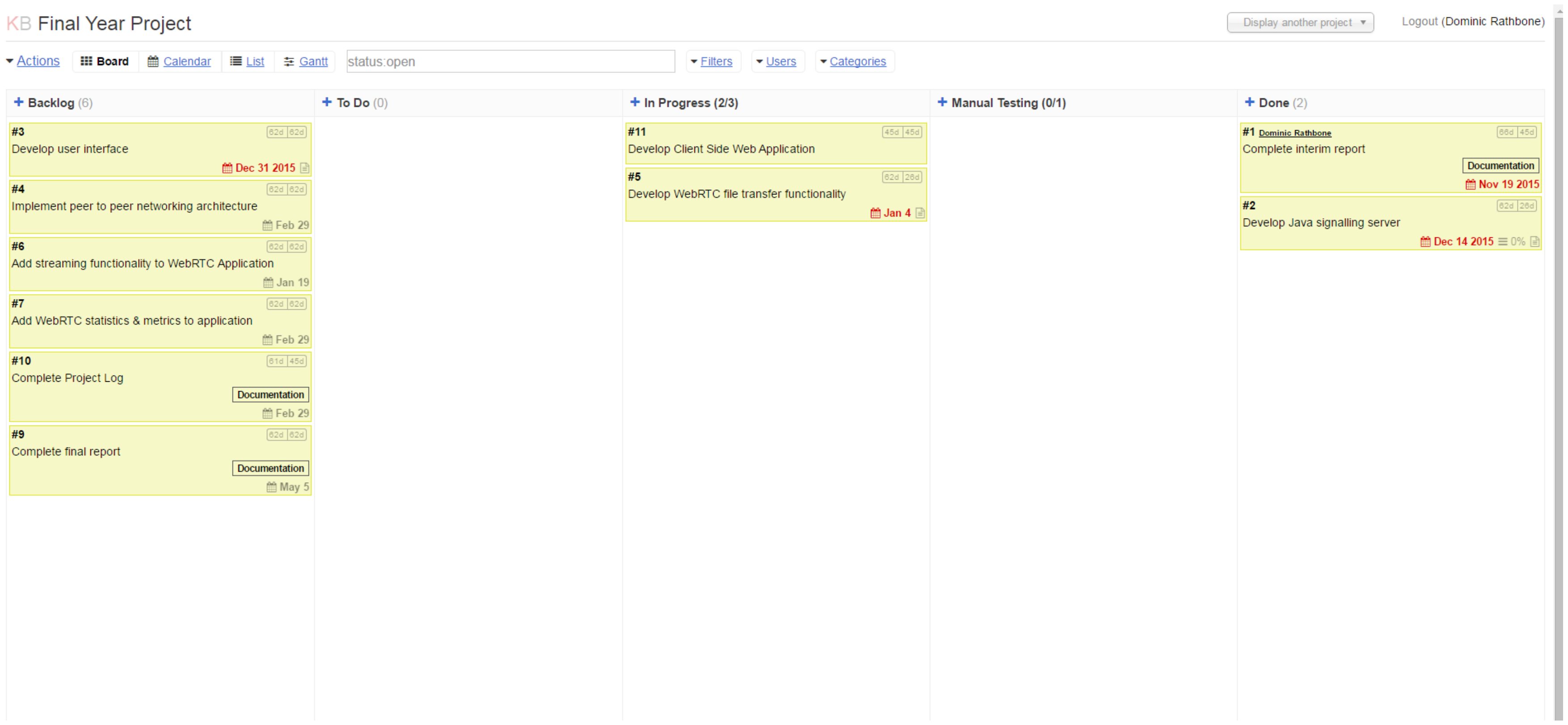
To get video streams from the camera, the API has a class called “VideoCapturerAndroid” that acts as a wrapper for the camera functionality on your phone, allowing the application to retrieve the camera input as a stream that can be added to the PeerConnectionFactory object. To do this, once a VideoCapturerAndroid object has been instantiated, you can instantiate a VideoSource or AudioSource object with the capturer as a parameter and then create a MediaStream object. To this MediaStream, the VideoSource and AudioSource can then be added. The source and stream objects are all created using PeerConnectionFactory methods. To output this video, the API again provides a class to do so called “VideoRendererGui” that uses OpenGL to render the stream to a view.

Technical challenges

The main technical challenge will be getting to grips with the WebRTC Native API for Android as it is a change from how I have worked with it before as a JavaScript API where it seems to be a lot more mature as a technology. Another problem I've found is sending data over WebRTC that isn't in the form of a stream is normally handled by the DataChannel API. However, I am struggling to find any documentation on this and it might not have been implemented yet, in this case I will have a challenge finding a solution for plain text messaging between two peers. On top of this, I am not familiar with OpenGL or video rendering techniques so handling this in conjunction with remote streaming might be complicated.

Work plan

To develop the project, I will use a methodology called Kanban. This focuses on using a task board to complete work iteratively. This work is tasked up before the project starts and belongs to a queue until it is pulled into the work flow, at this point it goes through an iteration of development, testing and stakeholder approval. This approach avoids the pitfalls of sequential approaches such as waterfall as each iteration through the work flow is relatively small and it is easy to reiterate through a step if it fails at some point whereas in waterfall, each step in the work flow takes months to complete and would cause a major delay if one failed. It also avoids overloading developers with too much work as it uses a WIP (work-in-progress) limit, indicating how many tasks they should have in the workflow at one time meaning the work will be to a higher quality. The kanban board is a big part of this process as it allows all the stakeholders of the team to visualise where the project is at, for my board I will use “Kanboard”, a free and open source web application that I have hosted on an Amazon Web Services server instance. I have been using this approach with this board in particular for my Final Year Project and it has been working very well.



KB Final Year Project

Display another project ▾ Logout (Dominic Rathbone)

▾ Actions | Board | Calendar | List | Gantt | status:open | ▾ Filters | ▾ Users | ▾ Categories

+ Backlog (6)

- **#3** Develop user interface — 62d 62d — Dec 31 2015
- **#4** Implement peer to peer networking architecture — 62d 62d — Feb 29
- **#6** Add streaming functionality to WebRTC Application — 62d 62d — Jan 19
- **#7** Add WebRTC statistics & metrics to application — 62d 62d — Feb 29
- **#10** Complete Project Log — 61d 45d — Documentation — Feb 29
- **#9** Complete final report — 62d 62d — Documentation — May 5

+ To Do (0)

+ In Progress (2/3)

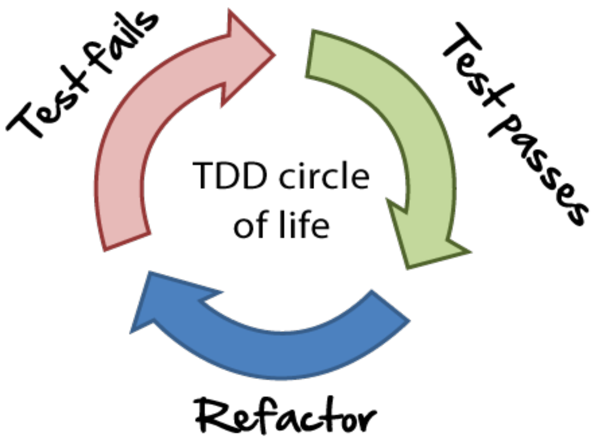
- **#11** Develop Client Side Web Application — 45d 45d
- **#5** Develop WebRTC file transfer functionality — 62d 26d — Jan 4

+ Manual Testing (0/1)

+ Done (2)

- **#1** Dominic Rathbone — Complete interim report — 66d 45d — Documentation — Nov 19 2015
- **#2** Develop Java signalling server — 62d 26d — Dec 14 2015 — 0%

For my development work flow, I will use a test-driven development methodology for the application logic in order to produce high quality, tested code. This focuses on writing unit tests first and writing the code to make the tests pass and can be represented by a traffic light system:

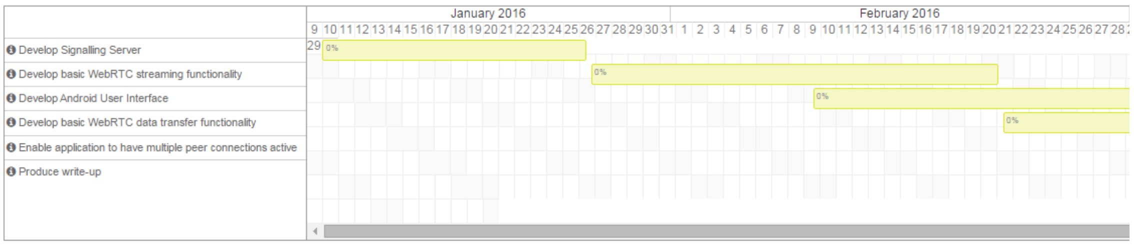


Once code has gone through this development process, it then goes to a manual testing phase in which I will test it from a user's perspective in order to make sure the flow throughout the application works and actually makes sense.

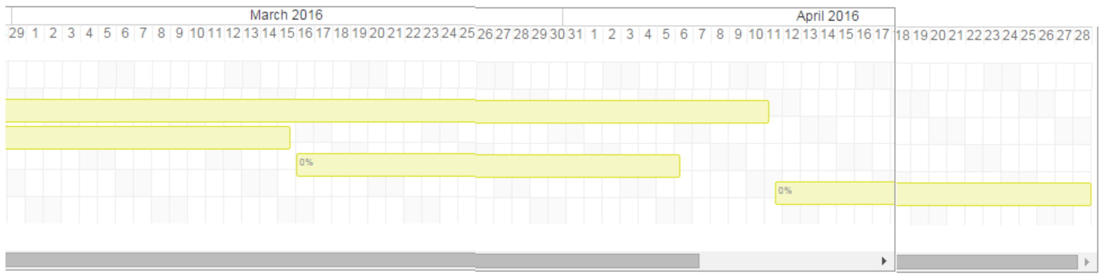
My application will be split up into several tasks but overall, there will be 4 deliverables, three end products and one intermediate. The first will be the Node.js signalling server as it needs to be completed to allow my application to form a connection between two applications. The intermediate product will be the bare bones application with working WebRTC streaming functionality, with this eventually becoming the second end product at which point it will be fully developed with a functioning user interface. The last product will be the write-up of my project, detailing how the project went.

Project Estimation

As an initial estimate for my project, I have produced a gantt chart:



Develop Signalling Server
Develop basic WebRTC streaming functionality
Develop Android User Interface
Develop basic WebRTC data transfer functionality
Enable application to have multiple peer connections active
Produce write-up



(Please, note: if the images are too small, there is a gantt chart within the zip file, along with larger version of all the images in this document).

Ethical considerations

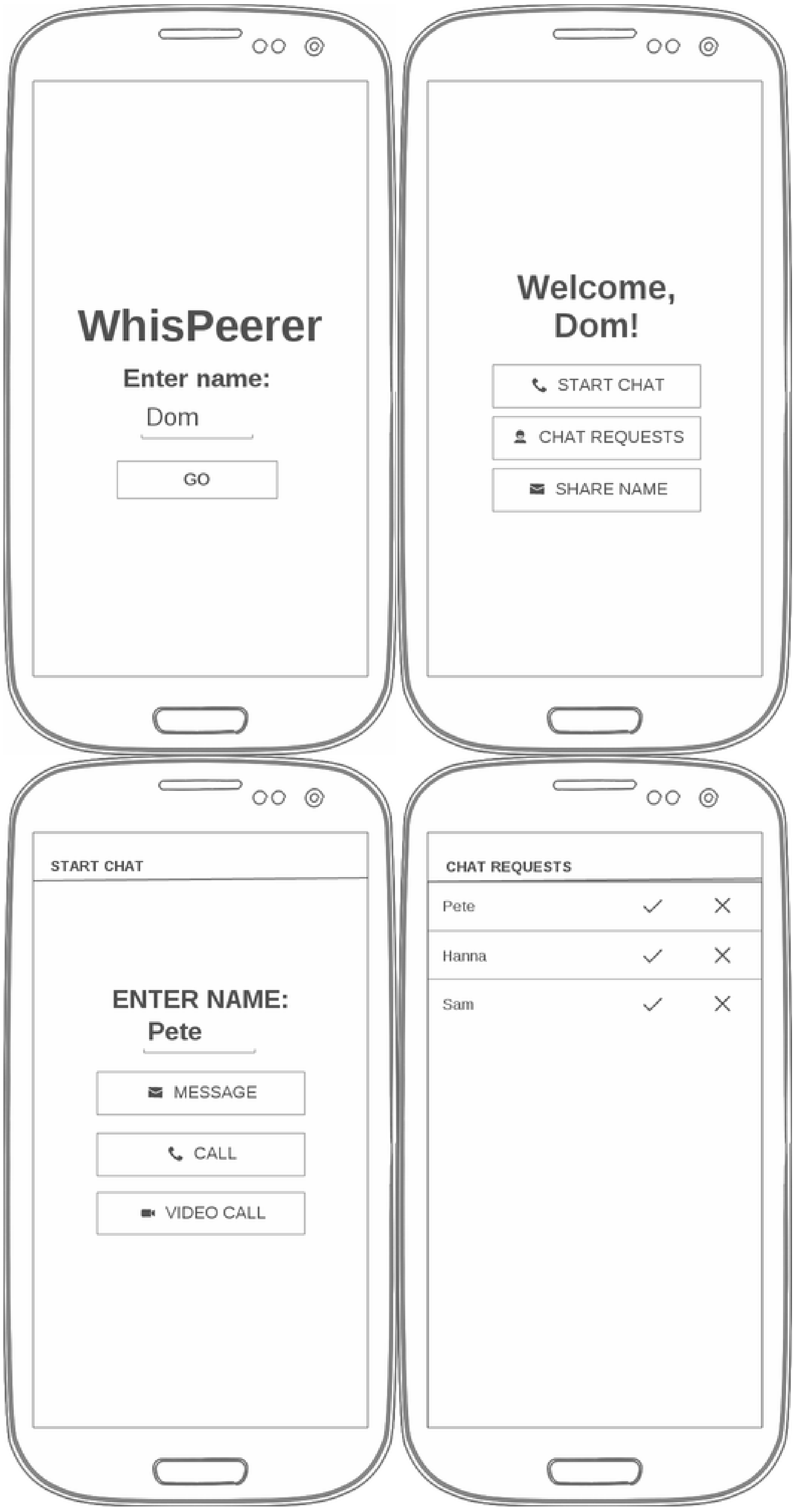
My application focuses on using peer-to-peer technology to transfer data directly between two instances. By doing this, it attempts to avoid the data privacy concerns associated with using servers to store information. In combination with this, it also avoids storing application data on the client by making all user information temporary. when the user starts the application, they create a completely disposable alias that is destroyed once they leave the application.

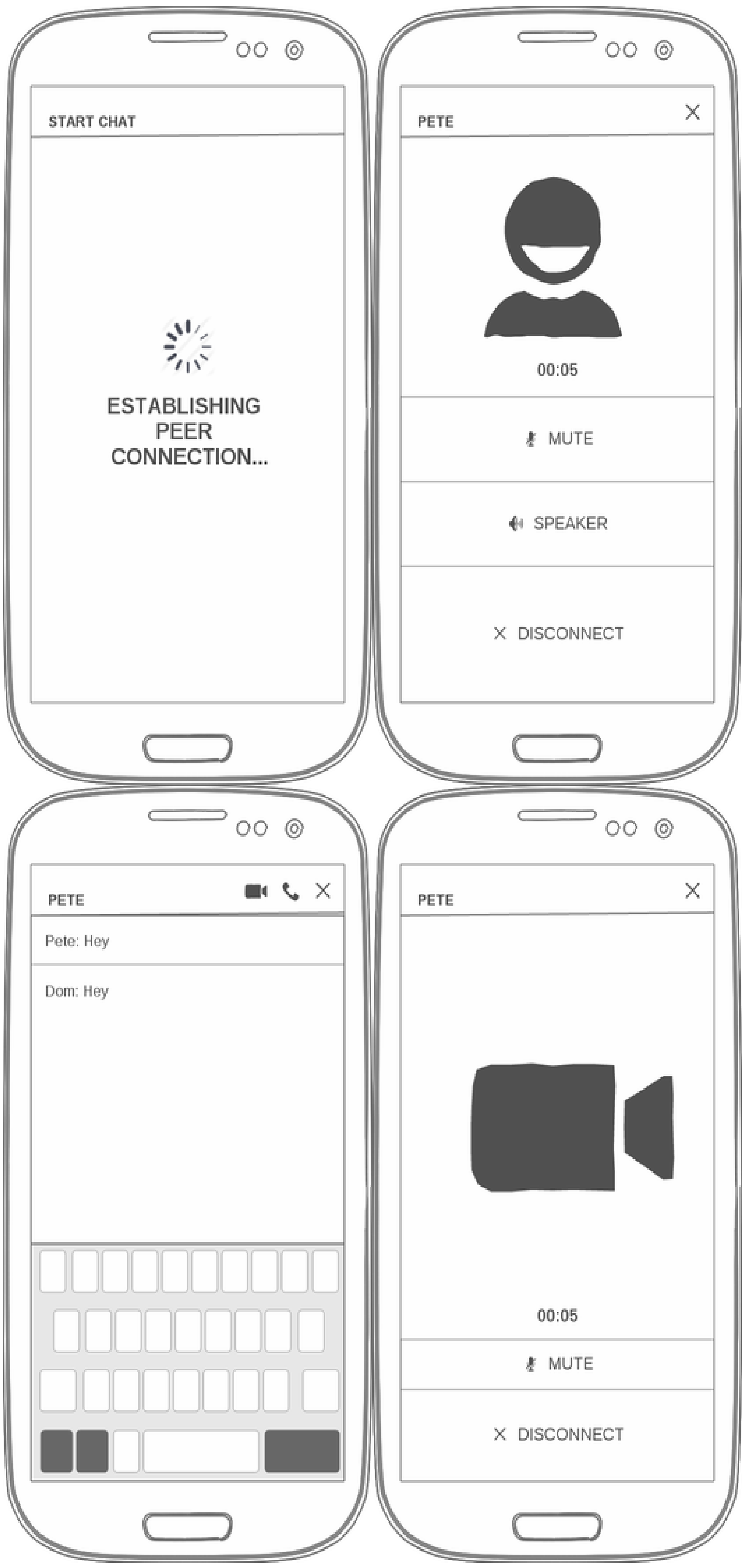
Due to the focus on data privacy, people may argue applications such as this may be exploited for discussion and engagement in illegal activities and consumers wouldn't use one if they hadn't done anything wrong. This argument is known as the “nothing to hide” argument and is a constant source of discussion, in particular since Edward Snowden disclosed evidence of government surveillance on citizens. In fact, Snowden makes a great point on the matter: “If you think privacy is unimportant for you because you have nothing to hide, you might as well say free speech is unimportant for you because you have nothing useful to say”. Outside of this, I don't believe my application would have any ethical issues to consider.

Resources

As it will be used for messaging, my application will require two android devices to test the streaming and data transfer functionality. To develop it, I will use the Android SDK along with a precompiled version of the LibJingle WebRTC library available on GitHub and Android Studio as my Java development Environment. For the signalling server, I will use Node.JS along with the socket.io library along with the GitHub Atom text editor as my development environment. To host the server, I will use a free amazon web services ec2 instance.

UI design draft





Pete: Hey

Dom: Hey

References

[1] OWASP. (2014). OWASP Mobile Security Project - Top Ten Mobile Risks.Available: https://www.owasp.org/index.php/Mobile_Top_10_2014-M1. Last accessed 06/01/2016.

[2] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. (N/A). A Study of Android Application Security. Available: http://www.cs.rice.edu/~sc40/pubs/enck-sec11.pdf. Last accessed 06/01/2016.

[3] Home Office. (2015). Draft Investigatory Powers Bill. Available: https://www.gov.uk/government/publications/draft-investigatory-powers-bill. Last accessed 06/01/2016.

[4] Leo Sun. (2015). Facebook Inc.'s WhatsApp Hits 900 Million Users: What Now?.Available: http://www.fool.com/investing/general/2015/09/11/facebook-incs-whatsapp-hits-900-million-users-what.aspx. Last accessed 06/01/2016.

[5] Alexei Oreskovic. (2015). Facebook Messenger added 100 million users in the last three months. Available: http://uk.businessinsider.com/facebook-messenger-has-700-million-users-2015-6?r=US&IR=T. Last accessed 06/01/2016.

[6] Philipp Hancke. (2015). What's up with WhatsApp and WebRTC?. Available: https://webrtchacks.com/whats-up-with-whatsapp-and-webrtc/. Last accessed 06/01/2016.

[7] Rory Cellan-Jones. (2015). Does the government really want to ban WhatsApp, iMessage and Skype?. Available: http://www.bbc.co.uk/news/technology-33737813. Last accessed 06/01/2016.

[8] Philipp Hancke. (2015). MESSENGER EXPOSED: Investigative Report. Available: https://cdn.andyet.com/webrtc-reports/messenger-report.pdf. Last accessed 06/01/2016.

[9] N/A. (N/A). Sicher FAQ. Available: http://www.shape.ag/en/faq/sections/sicher.php#why-is-sicher-secure. Last accessed 06/01/2016.

[10] N/A. (N/A). Sicher Privacy Policy. Available: http://www.shape.ag/en/privacy/sicher/. Last accessed 06/01/2016.

Justin Uberti. (2011). Javascript Session Establishment Protocol (JSEP).Available: https://lists.w3.org/Archives/Public/public-webrtc/2012Jan/att-0002/JavascriptSessionEstablishmentProtocol.pdf. Last accessed 09/01/2016.

Available: http://www.agilenutshell.com/assets/test-driven-development/tdd-circle-of-life.png. Lastaccessed 09/11/2015.