WhisPeerer
A WebRTC communications application

Dominic Rathbone
Student Number: 12843140

Source Code & Documentation:
https://github.com/domr115/CI360-Mobile-App-Dev

May 17, 2016

Contents

Chapter 1

Introduction

The aim of this project was to provide an application enabling users to communicate using peer-to-peer technology. The motivation behind this was the increasing concern of security many consumers face in the modern age due to third parties, whether it be a corporation such as Facebook or an authority such as the UK Government, storing and analysing their data. By using a technology called WebRTC, it is possible for two users to form a direct peer to peer connection over the internet. Although a server is used for the initial signalling and negotiation process between the peers, the communication data sent over this communications channel is never disclosed to an external server. On top of this, the application aimed to provide a sense of anonymity and impermanence by only providing users with a temporary, random unique identifier for a user-name and only storing data within the lifespan of the application.

Chapter 2

Development

2.1 Node.js Signalling Server

2.1.1 API

In order to set up a space in which two users can exchange the meta-data needed for the negotiation of a peer connection, an API was created on the server using Express.js, a popular web application framework for Node.js. This consists of two endpoints, one endpoint to create a new user and one endpoint to check if the user exists. The former returns a unique, random session identifier to the application and the latter simply returns a status code of 200 or 404 dependent on whether the user exists or not. s. This API was designed using the Representation State Transfer (REST) architectural style in order to provide a uniform and consistent API to it's consumers. REST achieves this by modelling the endpoints in an API around the resources they represent with HTTP verbs representing the operations that are achievable on this resource. For example, in order to create a user on the server, the consumer send a POST request to a "/user" endpoint and in order to check if a user exists, the consumer sends a GET request to "/users/[userId]".

2.1.2 WebSockets

To provide a bi-direction communications channels for the two application instances to negotiate the peer connection over, a WebSockets implementation called Socket.io was used. Socket.io models communication using the concepts of name spaces and rooms where a name space is represented by the endpoint a socket connects to and rooms are channels within this that users can join and leave. In this case, each user is represented by their own namespace "/user/[userId]". When another user wants to communicate with a user, they join their namespace which puts it into a "busy" state where no other user can join. From this namespace, the users can

2.2 Android Application

2.2.1 Architecture

2.2.2 Design Patterns

2.2.3 Android Async HTTP Client

2.2.4 WebSockets

2.2.5 WebRTC

2.2.6 OpenGL

2.2.7 Android Services

2.3 External Technologies

2.4 Git

2.5 IDE

Chapter 3

Design

As the application used relative new technologies, the majority of the project was focused on developing around these to ensure they worked. Due to this, the design aspect of the project was considered a lower priority.

Chapter 4

Testing

4.1 Automated Testing

4.1.1 Unit Tests

4.2 Manual Testing

Chapter 5

Reflection & Review

5.1 Background Research

5.2 Methodology

5.3 Project Estimation

5.4 Development Process

5.4.1 Debugging

5.5 Application Improvements

5.5.1 Web to Native Application Communication

5.6 Conclusion