**THE UNIVERSITY OF DANANG**

**UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**INFORMATION TECHNOLOGY FACULTY**

**GRADUATION PROJECT**

**MAJOR: INFORMATION TECHNOLOGY**

**SPECIALIZATION: SOFTWARE ENGINEERING**

**PROJECT NAME:**

**ONLINE CONFERENCE APPLICATION ON ANDROID PLATFORM**

Student: **VU NGOC SON**

Class: **12T4**

Student ID: **102120252**

Instructor: **M.S.DO THI TUYET HOA**

**Danang, 05/2017**

**INSTRUCTOR’S COMMENTS**

**REVIEWER’S COMMENTS**

# ABSTRACT

Subject name: Online conference application on Android platformStudent: Vu Ngoc Son  
Student ID: 102120252  
Class: 12T4  
Preview:

This subject will help the developer create your own videoconferencing easily which can work on web browser platform and mobile platform. It will also introduce you to WebRTC technology which is a strong open source.  
Video conference is an application help people create a meeting room to communicate by video or audio. The user just input username and room name, it will create a new one or join into an existing room. The vision of this application is communication multi-platform. In this subject, we will stop in communicating between a web browser and Android mobile.

|  |  |
| --- | --- |
| THE UNIVERSITY OF DANANG  **UNIVERSITY OF SCIENCE AND TECHNOLOGY**  INFORMATION TECHNOLOGY FALCUTY | **SOCIALIST REPUBLIC OF VIET NAM**  Independence - Freedom - Happiness |

**GRADUATION PROJECT REQUIREMENTS**

Student Name: Vu Ngoc Son Student ID: 102120252

Class:12T4 Department: Information Technology Major: Software Engineering

1. *Name of project:*

Online conference application on Android platform

1. *This project’s results:* ☐ *Are protected by an intellectual property agreement*
2. *Initial data:*

……………………………………..……………………………………………..……......……………………………………………………………………………………………..………………………………….…..………………………..………………………

1. *Project contents:*

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

1. *Charts and drawings (please specify the drawing type and size):*

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

1. *Supervisor:*…………………………………..……………………
2. *Date of assignment:*  *…./…/2017*
3. *Date of completion: …./…/2017*

|  |  |
| --- | --- |
|  | *Danang, 05/ 2017* |
| **Head of Division**…………………….. | **Supervisor** |

# ACKNOWLEDGEMENTS

This project would not have been possible without the help of many people. Hereafter, I would like to thank them for their help, support, and/or advice they have given me during the semester.

First, I would like to thank my project supervisor, M.S. Do Thi Tuyet Hoa for her guidance, help, and the many discussion sessions that gave me courage and enthusiasm to design and build this application.

Secondly, I would like to thank my family and my friends for their constant support and amazing ideas they provided me with.

Finally, I would like to thank University of Science and Technology, The University of Danang for this opportunity to work on this project.

Da Nang, May 2017

Sincerely,

*Vu Ngoc Son*

**GUARANTY**

I guarantee:

1. The contents of this senior project are performed by myself following the guidance of **M.S. Do Thi Tuyet Hoa**.
2. All references which used this senior project thesis, are quoted with author’s name, project’s name, time and location to publish clearly and faithfully.
3. All invalid copies, educated statute violation or cheating will be born the full responsibility by myself.

Student,

*Vu Ngoc Son*

# TABLE OF CONTENTS

[INSTRUCTOR’S COMMENTS i](#_Toc483056152)

[REVIEWER’S COMMENTS ii](#_Toc483056153)

[ABSTRACT iii](#_Toc483056154)

[GRADUATION PROJECT REQUIREMENTS iv](#_Toc483056155)

[ACKNOWLEDGEMENTS i](#_Toc483056156)

[GUARANTY ii](#_Toc483056157)

[TABLE OF CONTENTS iii](#_Toc483056158)

[LIST OF FIGURES v](#_Toc483056159)

[LIST OF TABLES vii](#_Toc483056160)

[LIST OF ABBREVIATIONS viii](#_Toc483056161)

[INTRODUCTION 1](#_Toc483056162)

[Chapter 1 THEORIES & TECHNOLOGIES 3](#_Toc483056163)

[1.1. WebRTC 3](#_Toc483056164)

[***1.1.1.* *Introducing* 3**](#_Toc483056165)

[***1.1.2.* *Peer to Peer* 5**](#_Toc483056166)

[***1.1.3.* *NAT, STUN Server, and TURN Server* 5**](#_Toc483056167)

[1.2. Media server - Kurento - SFU technology 7](#_Toc483056168)

[***1.2.1.* *Media Server* 7**](#_Toc483056169)

[***1.2.2.* *Kurento Media Server* 8**](#_Toc483056170)

[***1.2.3.* *Full Mesh - MCU - SFU* 9**](#_Toc483056171)

[1.3. Signaling Server – Node.js 13](#_Toc483056172)

[***1.3.1.* *Signaling* 13**](#_Toc483056173)

[***1.3.2.* *Node.js* 13**](#_Toc483056174)

[1.4. Android – Web SocketIO 15](#_Toc483056175)

[***1.4.1.* *Android - WebRTC SDK* 15**](#_Toc483056176)

[***1.4.2.* *SocketIO* 19**](#_Toc483056177)

[Chapter 2 Analysis & Design 20](#_Toc483056178)

[2.1. Video call one to one WebRTC 20](#_Toc483056179)

[***2.1.1.* *Introducing* 20**](#_Toc483056180)

[***2.1.2.* *Sequence diagram* 21**](#_Toc483056181)

[***2.1.3.* *Implementation* 22**](#_Toc483056182)

[2.2. Integrate KMS – Videoconferencing 27](#_Toc483056183)

[***2.2.1.* *Introducing* 27**](#_Toc483056184)

[***2.2.2.* *Install Kurento Media Server* 27**](#_Toc483056185)

[2.3. Use case diagram 30](#_Toc483056186)

[2.4. Screen mockups 33](#_Toc483056187)

[2.5. Sequence diagram 40](#_Toc483056188)

[Chapter 3 DEPLOYMENT AND TESTING 42](#_Toc483056189)

[3.1. Developing environment 42](#_Toc483056190)

[***3.1.1. Android Studio* 42**](#_Toc483056191)

[***3.1.2. Bitbucket* 42**](#_Toc483056192)

[3.2. Deployment 43](#_Toc483056193)

[***3.2.1. Deploy for user* 43**](#_Toc483056194)

[***3.2.2. Deploy for developer* 43**](#_Toc483056195)

[3.4. Testing 43](#_Toc483056196)

[3.5. Result evaluation 49](#_Toc483056197)

[REFERENCES 51](#_Toc483056198)

# LIST OF FIGURES

[Figure 1.1: WebRTC with all devices 3](#_Toc483056687)

[Figure 1.2: STUN WebRTC 6](#_Toc483056688)

[Figure 1.3: TURN Server with WebRTC 7](#_Toc483056689)

[Figure 1.4: Introducing Kurento 9](#_Toc483056690)

[Figure 1.5: Mesh Architecture 10](#_Toc483056691)

[Figure 1.6: MCU Architecture 11](#_Toc483056692)

[Figure 1.7: SFU Architecture 12](#_Toc483056693)

[Figure 1.8: Node.js diagram 14](#_Toc483056694)

[Figure 2.1: One to one diagram 21](#_Toc483056695)

[Figure 2.2: Get media stream demo 23](#_Toc483056696)

[Figure 2.3: WebRTC architecture 25](#_Toc483056697)

[Figure 2.4: One to one demo 27](#_Toc483056698)

[Figure 2.5: Use case diagram 30](#_Toc483056699)

[Figure 2.6: Screen mockup - Join room browser 33](#_Toc483056700)

[Figure 2.7: Screen mockup - Room browser 34](#_Toc483056701)

[Figure 2.9: Screen mockup - Display on remote when turning off video 35](#_Toc483056702)

[Figure 2.10: Screen mockup - Join room & Show room 36](#_Toc483056703)

[Figure 2.11: Screen mockup – Display functions 38](#_Toc483056704)

[Figure 2.12: KMS sequence diagram 40](#_Toc483056705)

[Figure 3.1. Initial project on Android Studio 42](#_Toc483056706)

[Figure 3.2.Manage source code on Bitbucket 43](#_Toc483056707)

[Figure 3.3: Demo - Join room 44](#_Toc483056708)

[Figure 3.4: Demo – Room 45](#_Toc483056709)

[Figure 3.5: Demo - functions 45](#_Toc483056710)

[Figure 3.6: Demo - Click on participant 46](#_Toc483056711)

[Figure 3.7: Demo - Turn your video 46](#_Toc483056712)

[Figure 3.8: Demo - Turn off video - participant's screen 47](#_Toc483056713)

[Figure 3.9: Demo - Join room - mobile 48](#_Toc483056714)

[Figure 3.10: Demo - Room mobile 48](#_Toc483056715)

[Figure 3.11: Demo - Room - Participants 49](#_Toc483056716)

[Figure 3.12: Demo - Room - Participants 49](#_Toc483056717)

# LIST OF TABLES

[Table 2.1: Use case - Join room 40](#_Toc483054335)

[Table 2.2: Use case - Switch video stream 41](#_Toc483054336)

[Table 2.3: Use case - Switch audio stream 41](#_Toc483054337)

[Table 2.4: Use case - Leave room 42](#_Toc483054338)

[Table 2.5: Description mockup - Join room browser screen 43](#_Toc483054339)

[Table 2.6: Description mockup - Show room browser screen 44](#_Toc483054340)

[Table 2.7: Description mockup - Turn off video screen 45](#_Toc483054341)

[Table 2.8: Description mockup - Join room mobile screen 46](#_Toc483054342)

[Table 2.9: Description mockup - Show room mobile screen 48](#_Toc483054343)

# LIST OF ABBREVIATIONS

|  |  |  |
| --- | --- | --- |
| **No.** | **Items** | **Description** |
| 1 | WebRTC | Web real-time communication |
| 2 | NAT | Network address translation |
| 3 | STUN | Session Traversal Utilities for NAT |
| 4 | TURN | Traversal Using Relay Around NAT |
| 5 | API | Application Programming Interface |
| 6 | KMS | Kurento Media Server |
| 7 | SDP | Session Description Protocol |
| 8 | P2P | Peer to Peer |
| 9 | SFU | Selective Forward Unit |
| 10 | MCU | Multi Connection Unit |
| 11 | HTML | HyperText Markup Language |
| 12 | CSS | Cascading Style Sheets |
| 13 | Socket IO | Socket Input Output |
| 14 | SDK | Software Development Kit |
| 15 | NPM | Node Package Manager |
| 16 | TV | Television |

# INTRODUCTION

Currently, various agencies like companies, enterprise, schools, families as well as universities are increasingly opting for available videoconferencing systems to allow interaction with people in geographically distant locations, since communication through video brings beneﬁts that voice or messages cannot meet themselves alone, such as body language, gestures, and demonstration of emotions, ﬁnally resulting in a more clear and concise communication that achieves the objectives of the meetings with the same effectiveness as a classroom event.   
Popular examples include Skype, Firefox Hello, Talky …

Skype is owned by Microsoft and uses a proprietary protocol for transmission of multimedia streams, plus it requires the installation of a mobile application or desktop to access the service. It also allows making video calls on the web platform, but the maximum for a video conference is ten but Skype recommends five for the best quality.

Firefox Hello is an application for video calls exclusively. It is based on WebRTC and is used from the Mozilla Firefox browser only. After the emergence of WebRTC, some companies have designed solutions based solely on this technology.

A notable example is Talky, which allows the user to create video conferencing rooms in a simple way, as the URL is created anonymously and sent to those who want to participate.

UberConference is another example of integral solutions, as consisting on a voice-only conference in which the user can participate from a web browser or making a phone call to a number assigned by the system to log in.  
Almost videoconferencing in the market, they do not public their source, so:  
**HOW CAN WE IMPLEMENT A VIDEOCONFERENCE APPLICATION?**

Prior to the existence of WebRTC, there were already many videoconferencing systems available on the market. One of the most comprehensive and focused on the business sector is Cisco WebEx, which is a robust remote collaboration solution composed of various services such as video conferencing rooms with file sharing features, desktop, presentation, and dissemination. Unlike this system, the project in this document is based on WebRTC, so the users will not need to download any software to run it in the web browser. In addition, we also provide a mobile app for mobile user Outside of the corporative usage, many applications allow video conferencing.Let’s imagine the world where your phone, TV, and computer could all communicate on a common platform. Imagine it was easy to add video chat and peer-to-peer data sharing to your web application. You can implement your own video conference easily. That's the vision of WebRTC.

# THEORIES & TECHNOLOGIES

## WebRTC

### *Introducing*

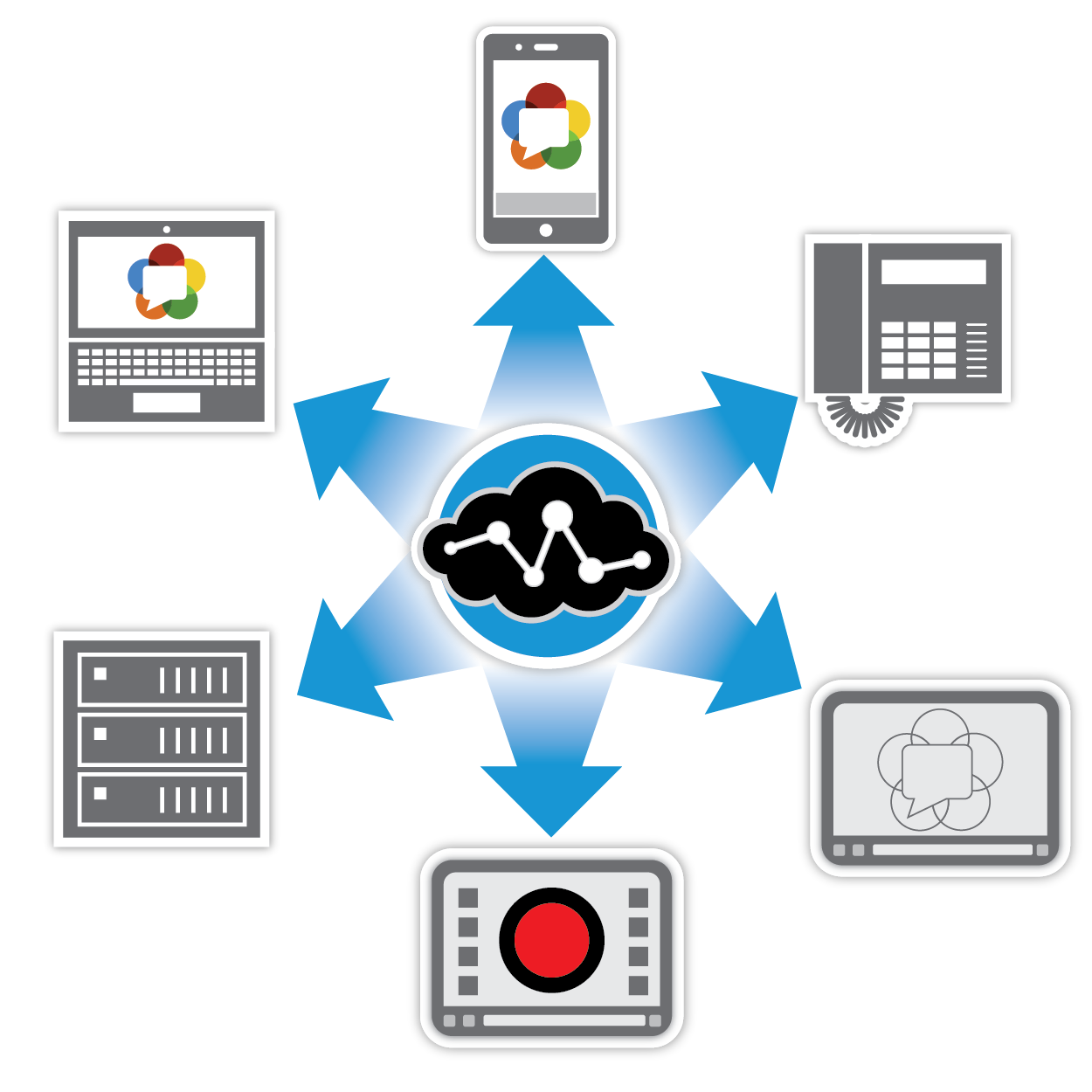
**

Figure 1.1: WebRTC with all devices

WebRTC is an open source project to enable real-time communication of audio, video, and data in The Web and native apps.  
WebRTC has several JavaScript APIs — click the links to see demos.

* getUserMedia (): capture audio and video.
* MediaRecorder: record audio and video.
* [RTCPeerConnection](https://webrtc.github.io/samples/src/content/peerconnection/pc1/): stream audio and video between users.
* RTCDataChannel: stream data between users.

**Where can I use WebRTC?**

In Firefox, Opera and in Chrome on desktop and Android. WebRTC is also available for native apps on iOS and Android.

**What is signaling?**

WebRTC uses RTCPeerConnection to communicate streaming data between browsers but also needs a mechanism to coordinate communication and to send control messages, a process known as signaling. Signaling methods and protocols are not specified by WebRTC. In this code lab, we use Node, but there are many alternatives.

**What are STUN and TURN?**

WebRTC is designed to work peer-to-peer, so users can connect by the most direct route possible. However, WebRTC is built to cope with real-world networking: client applications need to traverse NAT gateways and firewalls and peer to peer networking needs fallbacks in case direct connection fails. As part of this process, the WebRTC APIs use STUN servers to get the IP address of your computer, and TURN servers to function as relay servers in case peer-to-peer communication fails. We’ll go to detail in the next part.

**Is WebRTC secure?**

Encryption is mandatory for all WebRTC components, and its JavaScript APIs can only be used from secure origins (HTTPS or localhost). Signaling mechanisms aren't defined by WebRTC standards, so it's up to you make sure to use secure protocols.

### *Peer to Peer*

Peer-to-Peer (P2P) is a distributed network architecture in which nodes share a part of their resources to contribute to the service and content offered by the network. With the advent of WebRTC on browsers, P2P is presented as a promising technique for the scalability problem in video streaming over the Internet.

### *NAT, STUN Server, and TURN Server*

The Network Address Translator (NAT) (RFC1631) has been standardized to alleviate the scarcity and depletion of IPv4 addresses.  
A NAT device at the edge of a private local network is responsible for maintaining a table mapping of private local IP and port tuples to one or more globally unique public IP and port tuples. This allows the local IP addresses behind a NAT to be reused among many different networks, thus tackling the IPv4 address depletion issue.

***SESSION TRAVERSAL UTILITIES FOR NAT (STUN)***NATs provide a device with an IP address for use within a private local network, but this address can't be used externally. Without a public address, there's no way for WebRTC peers to communicate. To get around this problem WebRTC uses STUN.

STUN servers live on the public internet and have one simple task: check the IP: port address of an incoming request (from an application running behind a NAT) and send that address back as a response. In other words, the application uses a STUN server to discover its IP: port from a public perspective. This process enables a WebRTC peer to get a publicly accessible address for itself, and then pass that on to another peer via a signaling mechanism, in order to set up a direct link. (In practice, different NATs work in different ways, and there may be multiple NAT layers, but the principle is still the same.)

STUN servers don't have to do much or remember much, so relatively low-spec STUN servers can handle a large number of requests.

Most WebRTC calls successfully make a connection using STUN: 86%, according to webrtcstats.com, though this can be less for calls between peers behind firewalls and complex NAT configurations.

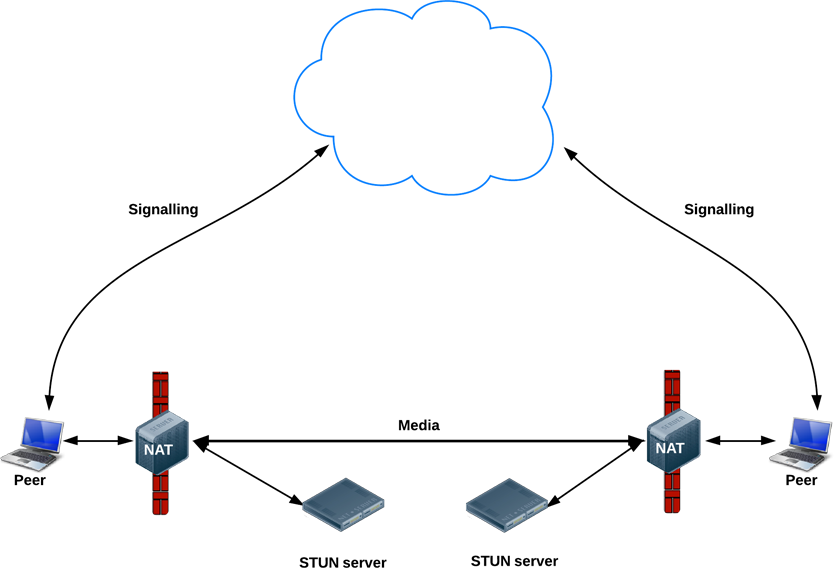


Figure 1.2: STUN WebRTC

***TRAVERSAL USING RELAY AROUND NAT (TURN)***  
If there is a firewall to be traversed, it may not allow stun based traffic to the other user. It happens in enterprises where port randomization techniques are used. In such cases, TURN is used. This works by adding relay between clients that acts as a peer to peer connection on behalf of the client. The client gets information from TURN server. It is similar to streaming videos from one server by making a request to another server.

In most cases, we will use a STUN server; it helps perform a NAT/firewall traversal and establish a direct connection between the peers. In other words, the STUN server is utilized only during the stage of establishing a connection. After the connection has been established, peers will transfer the media data directly between them.

In some cases (unfortunately, they are not so rare), the STUN server won't help you get through a firewall or NAT, and establishing a direct connection between the peers will be impossible, for example, if both peers are behind a symmetric NAT.  
In this case, the TURN server can help you.  
A TURN server works as a retransmitter between the peers. Using the TURN server, all the media data between the peers will be transmitted through the TURN server.

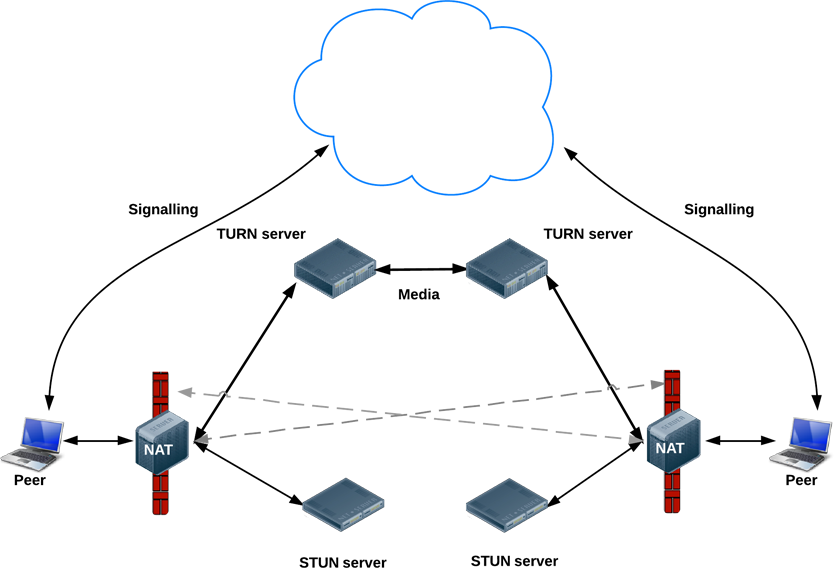


Figure 1.3: TURN Server with WebRTC

If your application gives a list of several STUN/TURN servers to a WebRTC API, then the web browser will try to use STUN servers first; in case the connection failed, it will try to use the TURN servers automatically.

## Media server - Kurento - SFU technology

### *Media Server*

WebRTC is currently under standardization at the IETF and W3C and has the support of the most important companies in the area of Internet and telecommunications. WebRTC has been conceived as a peer-to-peer technology where browsers can directly communicate without the mediation of any kind of infrastructure. This model is enough for creating basic applications but features such as group communications, media stream recording, media broadcasting or media transcoding are difficult to implement on top of it. For this reason, many applications require using a media server.

Conceptually, a WebRTC media server is just a kind of “multimedia middleware” (it is in the middle of the communicating peers) where media traffic pass through when moving from source to destinations. Media servers are capable of processing media streams and offering different types including groups communications (distributing the media stream one peer generates among several receivers, i.e. acting as Multi-Conference Unit), mixing (transforming several incoming streams into one single composite stream), transcoding (adapting codecs and formats between incompatible clients), recording (storing in a persistent way the media exchanged among peers), etc.

### *Kurento Media Server*

Kurento is a WebRTC media server and a set of client APIs making simple the development of advanced video applications for WWW and smartphone platforms. Kurento features include group communications, transcoding, recording, mixing, broadcasting and routing of audiovisual flows.

Kurento also provides advanced media processing capabilities involving computer vision, video indexing, augmented reality and speech analysis. Kurento modular architecture makes simple the integration of third party media processing algorithms (i.e. speech recognition, sentiment analysis, face recognition, etc.), which can be transparently used by application developers as the rest of Kurento built-in features.

Kurento’s core element is Kurento Media Server, responsible for media transmission, processing, loading, and recording. It is implemented in low-level technologies based on GStreamer to optimize the resource consumption. It provides the following features:

• Networked streaming protocols, including HTTP, RTP, and WebRTC.

• Group communications (MCUs and SFUs functionality) supporting both media mixing and media routing/dispatching.

• Generic support for computational vision and augmented reality filters.

• Media storage supporting writing operations for WebM and MP4 and playing in all formats supported byGStreamer.

• Automatic media trans codification between any of the codecs supported by GStreamer including VP8, H.264, H.263, AMR, OPUS, Speex, G.711, etc.

There are available Kurento Client libraries in Java and Javascript to control Kurento Media Server from applications. And in this application, we will integrate Kurento with NodeJs based on WebSocket and JSON-RPC.

At the heart of the Kurento architecture, there is a media server called the Kurento Media Server (KMS). Kurento Media Server is based on pluggable media processing capabilities meaning that any of its provided features is a pluggable module that can be activated or deactivated. Moreover, developers can seamlessly create additional modules extending Kurento Media Server with new functionalities which can be plugged dynamically.

Kurento Media Server provides, out of the box, group communications, mixing, transcoding, recording, and playing.

In addition, it also provides advanced modules for media processing including computer vision, augmented reality, alpha blending and much more.



Figure 1.4: Introducing Kurento

### *Full Mesh - MCU - SFU*

There are in general 3 main models of deploying a multiparty video conference: Full Mesh, MCU, SFU. And now we will go to figure out what are they? And what are advantages such as disadvantages of each these architectures?

Mesh is an architecture for multipoint where every participant sends and receives its media to all other participants.A mesh is a very common technique that is used in WebRTC to build multipoint conferences. It can usually scale to 4-6 participants for video sessions at most.

You can image that, If we have n participants want to make a video call together. We have to create n\*(n-1) peer connections to store and display media stream for each participant.



Figure 1.5: Mesh Architecture

Advantages of mesh architecture

* Simple to implement in WebRTC
* Requires very little backend infrastructure, keeping the resulting service cheap to operate

Disadvantages of mesh architecture

* Can’t scale to a large number of participants
* Requires a lot of uplink bandwidth from the participants

MCU stands for Multipoint Conferencing Unit.|  
MCU - Where a participant is “speaking” to a central entity who mixes all inputs and sends out a single stream towards each participant  
MCU generally implement the mixing architecture and are expensive due to their need for a lot of processing power per session.



Figure 1.6: MCU Architecture

Advantages of mesh architecture

* Simple at client side: single audio/video mixed stream from server
* Interoperability: the server can transcode
* Low bandwidth required

Disadvantages of mesh architecture

* CPU expensive decoding/encoding in server side (high latency)
* Nonflexible client side application

SFU stands for Selective Forwarding Unit.  
At times, the term is used to describe a type of video routing device, while at other times it will be used to indicate the support of routing technology and not a specific device.  
SFU - Where a participant sends his media to a central entity, who routes all incoming media as he sees fit to participants – each one of them receiving usually more than a single stream



Figure 1.7: SFU Architecture

**Advantages of mesh architecture**

* High throughput, low latency
* Low CPU at server side (no decoding/encoding)
* Good uplink bandwidth usage
* The client-side application can render each remote video stream as desired

**Disadvantages of mesh architecture**

* Simulcast/SVC required for real scenarios

## Signaling Server – Node.js

### *Signaling*

Signaling is the process of coordinating communication. In order for a WebRTC application to set up a 'call', its clients need to exchange information:

* Session control messages used to open or close communication.
* Error messages.
* Media metadata such as codecs and codec settings, bandwidth and media types.
* Key data used to establish secure connections.
* Network data, such as a host's IP address and port as seen by the outside world

This signaling process needs a way for clients to pass messages back and forth. That mechanism is not implemented by the WebRTC APIs.  
And in this thesis, we will use NodeJs to build our signaling server.

### *Node.js*

Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine)for easily building fast and scalable network applications. It was developed by Ryan Dahl in 2009 and its latest version is v0.10.36. It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices  
Node.js is an open source, a cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.  
Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

Following are some of the important features that make Node.js the first choice of software architects.

* **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
* **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
* **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.
* **License** − Node.js is released under the MIT license.

Following is the link on Github wiki containing an exhaustive list of projects, application, and companies which are using Node.js. This list includes eBay, General Electric, GoDaddy, Microsoft, PayPal, Uber, Wiki pics, Yahoo!, and Yammer to name a few.

The following diagram depicts some important parts of Node.js.

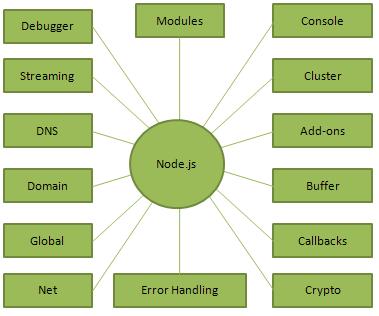


Figure 1.8: Node.js diagram

Following are the areas where Node.js is proving itself as a perfect technology partner.

* I/O bound Applications
* Data Streaming Applications
* Data Intensive Real-time Applications (DIRT)
* JSON APIs based Applications
* Single Page Applications

Node.js is not advisable to use Node.js for CPU intensive applications.

## Android – SocketIO

### *Android - WebRTC SDK*

* + - 1. ***Introducing Android***

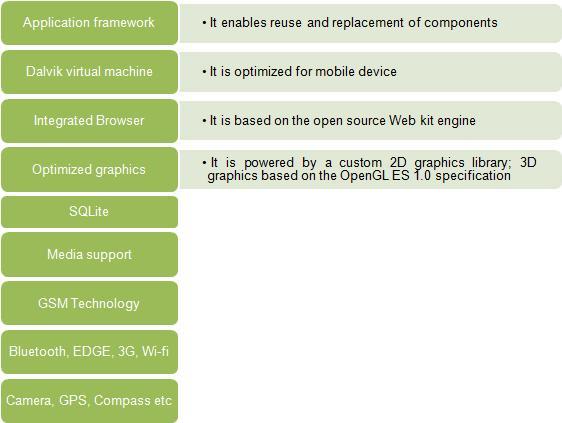
Operating Systems have developed a lot in last 15 years. Starting from black and white phones to recent smartphones or mini computers, mobile OS has come far away. Especially for smartphones, Mobile OS has greatly evolved from Palm OS in 1996 to Windows pocket PC in 2000 then to Blackberry OS and Android.

One of the most widely used mobile OS these days is **ANDROID**. **Android** is a software bunch comprising not only the operating system but also middleware and key applications. Android Inc was founded in Palo Alto of California, U.S. by Andy Rubin, Rich miner, Nick sears and Chris White in 2003. Later Android Inc. was acquired by Google in 2005. After original release, there have been a number of updates in the original version of Android.



**Features & Specifications**

**Android** is a powerful Operating System supporting a large number of applications in Smart Phones. These applications make life more comfortable and advanced for the users. Hardware that supports Android is mainly based on ARM architecture platform. Some of the current features and specifications of android are:



Android comes with an Android market which is an online software store. It was developed by Google. It allows Android users to select, and download applications developed by third party developers and use them. There are around 2.0 lack+ games, application, and widgets available on the market for users.

Android applications are written in java programming language. Android is available as open source for developers to develop applications which can be further used for selling in the android market. There are around 200000 applications developed for android with over 3 billion+ downloads. Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. For software development, Android provides **Android SDK** (Software development kit). Read more about open source software.

**Applications**

These are the basics of Android applications:

•      Android applications are composed of one or more application components (activities, services, content providers, and broadcast receivers)

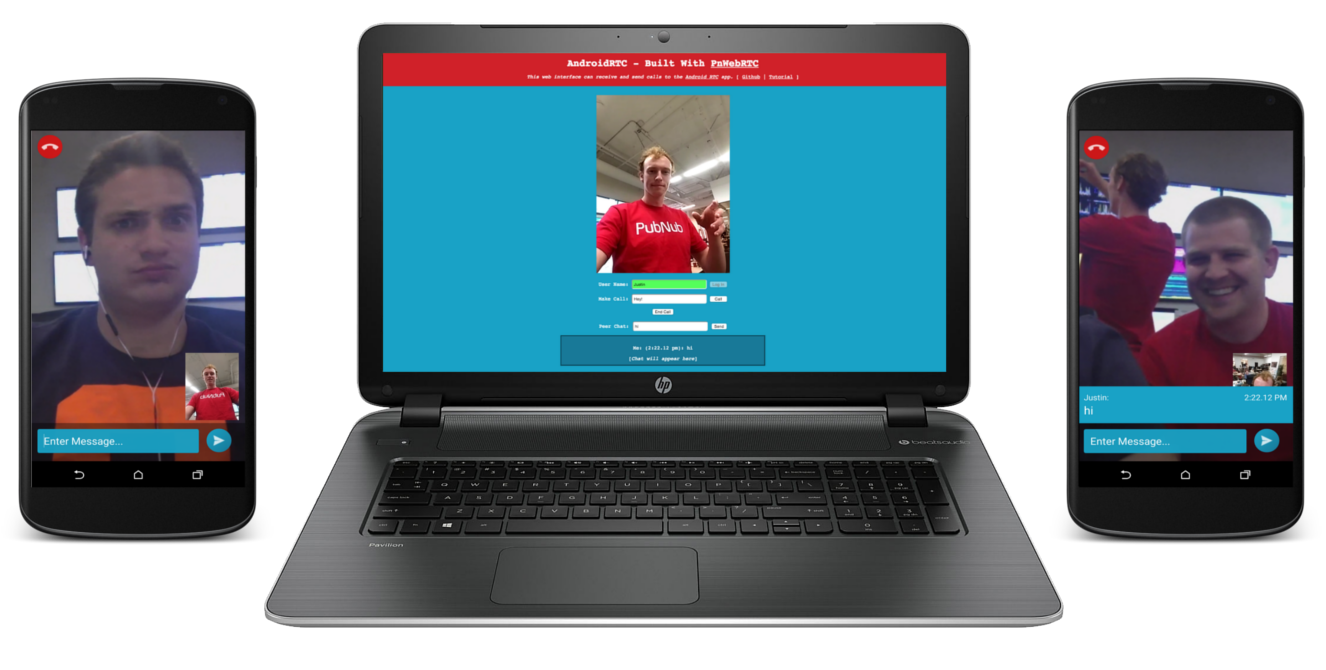
•      Each component performs a different role in the overall application behavior, and each one can be activated individually (even by other applications)

•      The manifest file must declare all components in the application and should also declare all application requirements, such as the minimum version of Android required and any hardware configurations required

•      Non-code application resources (images, strings, layout files, etc.) should include alternatives for different device configurations (such as different strings for different languages)

Google, for software development and application development, had launched two competitions ADC1 and ADC2 for the most innovative applications for Android. It offered prizes of USD 10 million combined in ADC1 and 2. ADC1 was launched in January 2008 and ADC 2 was launched in May 2009. These competitions helped Google a lot in making Android better, more user-friendly, advanced and interactive.

* + - 1. ***WebRTC Android SDK***



Google also provide WebRTC SDK for Android devices.The Android WebRTC SDK is fully compatible with our JavaScript WebRTC SDK, making our Android app cross-platform. No additional technology is required for mobile-to-web and web-to-mobile video and voice communication.

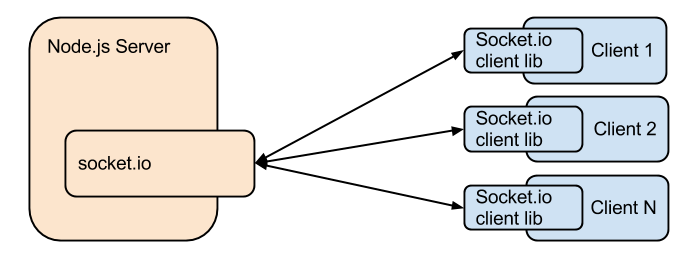
WebRTC is already a flexible, lightweight API for web-based communication, so when combined with our new Android SDK, you have the makings for a powerful communication app for the web and mobile users alike.

### *SocketIO*

How to client side can communicate with server real-time? Using socketIO you can easily build real-time applications and even multi-connections.  
Socket.IO finally reached version 1.0 on the 28th of May, 2014. The Socket.IO project contained two parts before 1.0: a transport handling implementation, and a high-level API. Transport handling has been moved out into a separate, framework-agnostic project: Engine.IO. This allows other developers to build new APIs and projects for the real-time web without reinventing the wheel.  
Apart from architectural changes, Socket.IO 1.0 introduces many user-facing changes, including:

* Binary streaming support
* Improved support for horizontal scaling
* Removal of cluttered debug messages in the console by default
* Support for Node.js streams via the socket.io - stream module

In this article, we'll take a quick look at how Socket.io can be used to send and receive data in real-time.  
Socket.IO provides both server-side and client-side components with similar APIs.



And in this article, we will integrate SocketIO with Node.js, as you can image we will have a strong real-time application.

# Analysis & Design

We’re going to build an application base on WebRTC architecture which is technology will help you implement your own video conference easily. With a vision, your application will be worked in almost kinds of devices like web, mobile … It will help on useful communication.

* 1. **Video call one to one WebRTC**
     1. ***Introducing***

Build an app to get video from your camera and share with another one and receive simultaneous video from peer-to-peer via WebRTC. Along the way, we'll learn how to use the core WebRTC APIs and set up a messaging server using Node.

**What we'll learn**

* Get video from your webcam
* Stream video with RTCPeerConnection
* Stream data with RTCDataChannel
* Set up a signaling service to exchange messages
* Combine peer connection and signaling

**What we'll need**

* Chrome 47 or above
* Web Server for Chrome, or use your own web server of choice.
* The sample code
* A text editor
* Basic knowledge of HTML, CSS, and JavaScript
  + 1. ***Sequence diagram***

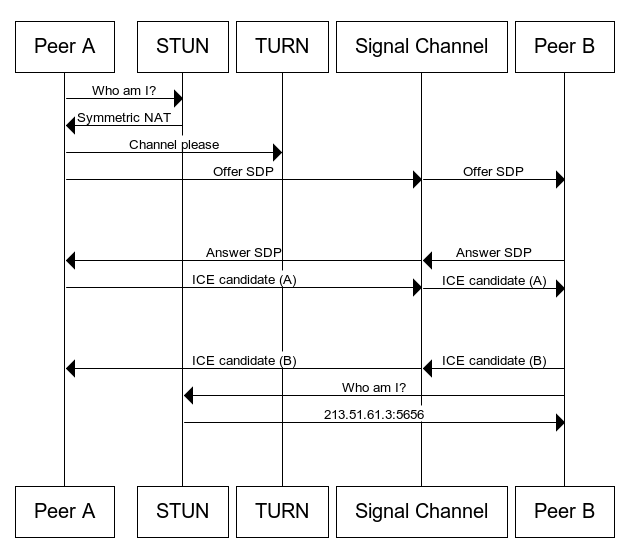


Figure 2.1: One to one diagram

* Peer A will use APIs of WebRTC to create his media stream. Media stream will include video and audio, and then send a signal to peer B to announce that A wants to create a connection with B.
* Peer B receive a signal from A and reply announce A that he accept to create a connection with A, and B also turns on his media stream.
* After receive agree on a signal from B, A will create RTCPeerConnection and call createOffer() to create sdpOffer. If it’s successful, it will return RTCPeerDescription. A will send RTCPeerDescription and sdpOffer to B via the server.
* B receive data from A and then set remote stream from RTCPeerDescription of A and create SDP answer. After creating SDP answer, B also have RTCPeerDescription and SDP answer. Now, B will send them to A via the server.
* A will receive data from B and set remote stream from RTCPeerDescription of B.
* After that, A and B connected peer to peer and they can see media each other.
  + 1. ***Implementation***

**Step 1: Get media stream and turn on camera:**

For more basic, we will try it on Javascript library first.

1. 'use strict';
2. navigator.getUserMedia = navigator.getUserMedia ||
3. navigator.webkitGetUserMedia || navigator.mozGetUserMedia;
4. var constraints = {
5. audio: true,
6. video: true
7. };
8. var video = document.querySelector('video');
9. function successCallback(stream) {
10. window.stream = stream; *// stream available to console*
11. if (window.URL) {
12. video.src = window.URL.createObjectURL(stream);
13. } else {
14. video.src = stream;
15. }
16. }
17. function errorCallback(error) {
18. console.log('navigator.getUserMedia error: ', error);
19. }
20. navigator.getUserMedia(constraints, successCallback, errorCallback);

Comment:  
Line 5, 6: Check platform to run WebRTC  
Line 8: Open stream video and audio  
Line 12: Get video tag to display media stream  
Line 14-21: Adding media stream to video tag  
Line 23-25: Callback error function  
Line 27: Run program.

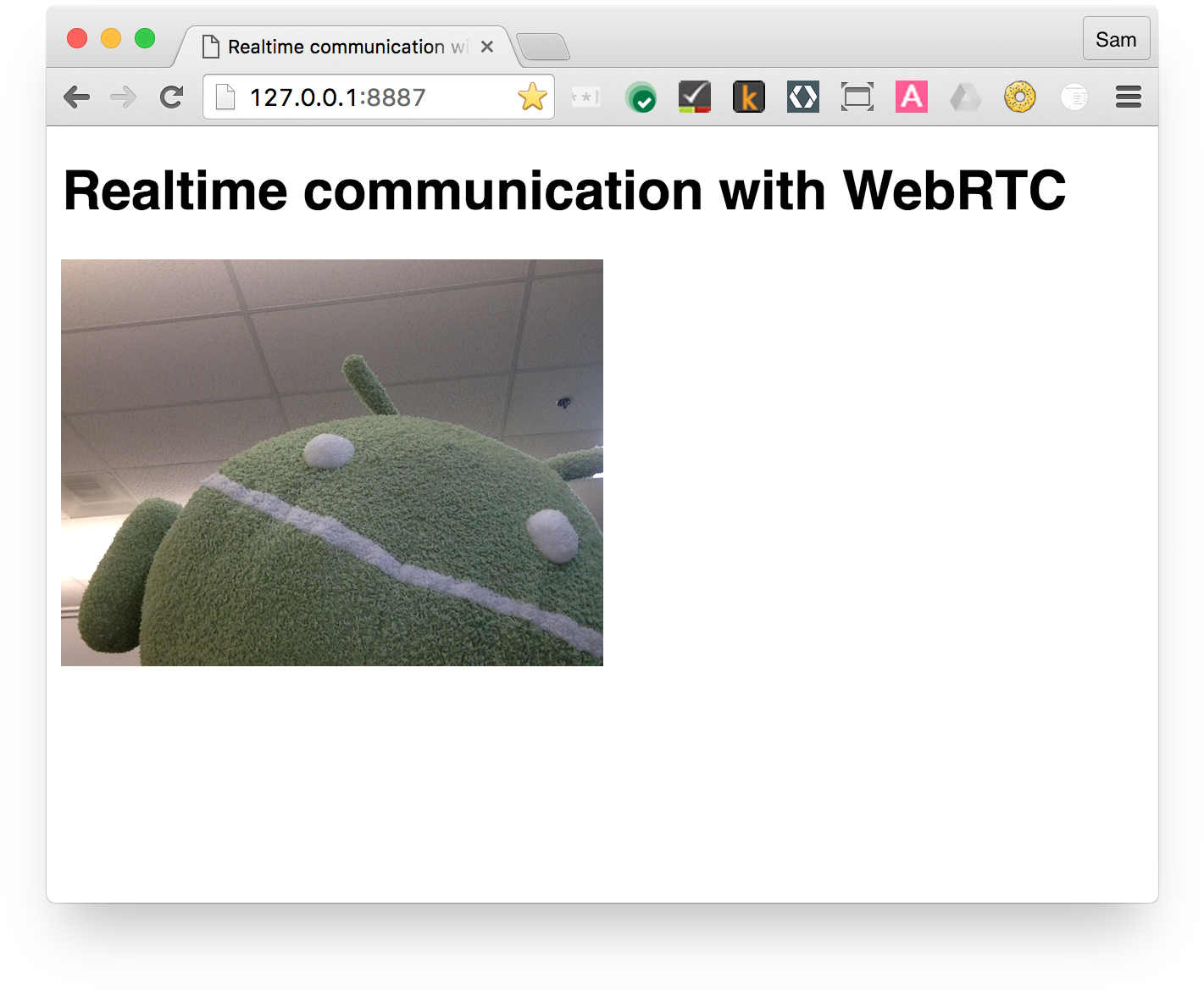


Figure 2.2: Get media stream demo

**Step 2: Setting signaling**

The Node application in this step has two tasks.  
First, it acts as a message relay:  
This function will listen client’s signal on node “message”. After having signal, the server will send back to the client on “message” node too.  
socket.on() is function listen event.  
socket.broadcast.emit() is function send back event to client.

socket.on('message', function (message) {

log('Got message: ', message);

socket.broadcast.emit('message', message);

});

Second, it manages WebRTC video chat 'rooms':  
Firstly, for each participant who joins to the room, the server will check a number of participants already. If a number of participants equal to 1, the server will create a room. If a number of participants == 2, the server will have that participant join to the room.   
If we have already 2 people in the room, the server will return room full for people who joins to room.

if (numClients === 1) {

socket.join(room);

socket.emit('created', room, socket.id);

} elseif (numClients === 2) {

socket.join(room);

socket.emit('joined', room, socket.id);

io.sockets.in(room).emit('ready');

} else { *// max two clients*

socket.emit('full', room);}

**Step 3: RTCPeerConnection – send and receive via server**

RTCPeerConnection is the WebRTC component that handles stable and efficient communication of streaming data between peers.

Below is a WebRTC architecture diagram showing the role of RTCPeerConnection. As you will notice, the green parts are complex!

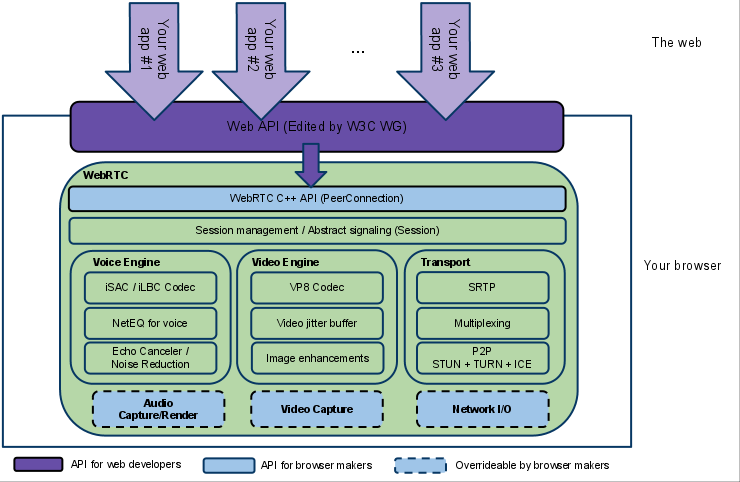
****

Figure 2.3: WebRTC architecture

From a JavaScript perspective, the main thing to understand from this diagram is that RTCPeerConnection shields web developers from the myriad complexities that lurk beneath. This part shows a simplified example of WebRTC from a signaling perspective.  
Create an offer and set it as the local description to the server using socket.io :

pc1 =new webkitRTCPeerConnection(servers);  
pc1.addStream(localStream);  
pc1.createOffer(gotDescription1);  
function gotDescription1(desc){  
 pc1.setLocalDescription(desc);  
 trace("Offer from pc1 \n"+ desc.sdp);  
io.socket.emit(‘message’, desc);  
}

After caller sends SDF offer to the server, the server will send it to the callee. After callee receives SDP offer, callee will create new peer connection and add this stream, it also creates SDP answer and sends to the caller via the server.

pc2 =new webkitRTCPeerConnection(servers);  
function gotRemoteStream(e){  
pc2.setRemoteDescription(desc);  
pc2.createAnswer(function(sdpAnswer) {  
 io.socket.emit(‘message’, sdpAnswer);  
 });  
}

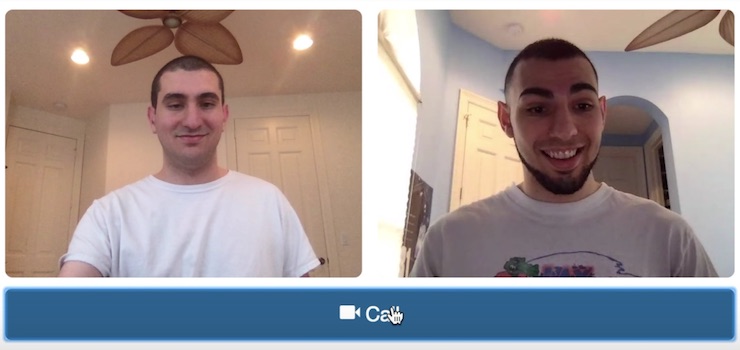


Figure 2.4: One to one demo

* 1. **Integrate KMS – Videoconferencing**
     1. ***Introducing***

We’ve got how data flow when we use WebRTC via the basic application above. So in this part, we will find down how to implement Kurento for creating multi-video conference as well use SFU architecture.

As you know, by SFU architecture, we will have server get all media stream from clients, and then server will execute some process like create SDP answer, ice candidate … and send back to client. In order to do it, we must have a media server to exchange stream. Kurento Media Server is a strong media server and we probably use it as well.

Videoconferencing is a mobile application have user can create room for meeting, discussion … Additional, we can use WebRTC for communication multi devices, it means videoconferencing can have people communicate between mobile and web browser. We can develop to more device in the future.

For more basic, we will go on communication between 2 user first and then multi people we can implement similar completely.

* + 1. ***Install Kurento Media Server***

Step 1: Kurento Media Server Installation

Kurento Media Server (KMS) has to be installed on Ubuntu 14.04 LTS (64 bits).  
In order to install the latest stable Kurento Media Server version (6.6.0), we have to type the following commands, one at a time and in the same order as listed here. When asked for any kind of confirmation, reply affirmatively.

echo “deb <https://ubuntu.kurento.org> trusty kms6” | do tee /etc/apt/sources.list.d/kurento.list

wget -0 – <http://ubuntu.kurento.org/kurento.gpg.key> | sudo apt-key add -  
sudo apt-get update

sudo apt-get install kurento-media-server-6.0

**Step 2: Configure STUN and TURN server**

If Kurento Media Server is located behind a [NAT](http://doc-kurento.readthedocs.io/en/stable/glossary.html#term-nat) we need to use a [STUN](http://doc-kurento.readthedocs.io/en/stable/glossary.html#term-stun) or [TURN](http://doc-kurento.readthedocs.io/en/stable/glossary.html#term-turn) in order to achieve [NAT traversal](http://doc-kurento.readthedocs.io/en/stable/glossary.html#term-nat-traversal). In most of the cases, a STUN server will do the trick. A TURN server is only necessary when the NAT is symmetric.

In order to setup a STUN server you should uncomment the following lines in the Kurento Media Server configuration file located on at /etc/kurento/modules/kurento/WebRtcEndpoint.conf.ini:

stunServerAddress**=<**stun\_ip\_address**>**  
stunServerPort**=<**stun\_port**>**  
stunServerAddress**=<**stun\_ip\_address**>**stunServerPort**=<**stun\_port**>**

The parameter stunServerAddress should be an IP address (not the domain name). There is plenty of public STUN servers available, for example, 173.194.66.127:19302

In order to setup a TURN server you should uncomment the following lines in the Kurento Media Server configuration file located on at /etc/kurento/modules/kurento/WebRtcEndpoint.conf.ini:  
turnURL=user:password@address:port;  
As before, TURN address should be an IP address (not the domain name). See some examples of TURN configuration below:  
turnURL=kurento:kurento@193.147.51.36:3478;

**Step 3: Integrate Kurento with signaling server Node.js**

After installation and configure Kurento Media Server, we have to integrate Kurento with Node.js server to call and use it. Node package management provides kurento-client for access and uses APIs of Kurento.  
Easily, we just run the command: npm install kurento-client.  
All of the libraries will be installed automatically for you.

After installing kurento-client for the server. We will have some function which is provided for Kurento to create RTCPeer, generate SDP, the ICE candidate ...

* 1. **Use case diagram**

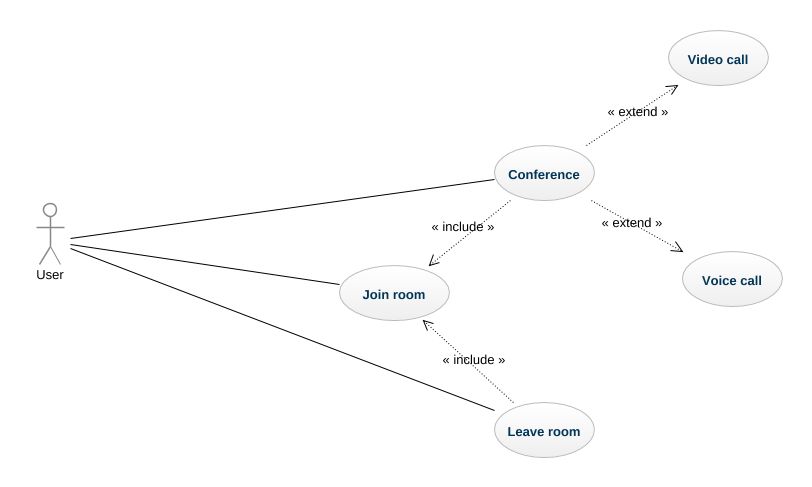
****

Figure 2.5: Use case diagram

Table 2.1: Use case - Join room

|  | |
| --- | --- |
| ID | UC - 1 |
| Title | Join room |
| Description | The user creates a room if that room exists, the user can join into that room. |
| Primary Actor | User |
| Preconditions | User is joined into application |
| Postconditions | Seeing video and audio stream. |
| Flow of events | Firstly, the user needs to input name and room that user wants to join or create. Secondly, the user will receive an announcement if the user is using web platform. That announcement has user turn media stream. The third, after user agrees on the announcement, the user will see the video on the screen, if that room has already a user, the user will also see these video from them. |
| Success points | The user will see media own stream and the other’s media stream. |
| False points | The user can listen audio but can not see the video. The user can not see or listen to any stream. |

Table 2.2: Use case - Switch video stream

|  | |
| --- | --- |
| ID | UC - 2 |
| Title | Switch video stream |
| Description | User can open or close own video stream |
| Primary Actor | User |
| Preconditions | User is joined into application successfully |
| Post-conditions | No display own video. |
| Flow of events | Firstly, the user needs to join room successfully. Secondly, Click on icon video stream The third, if currently, the video is opening, the application will close video. |
| Success points | The application will close the video, and in the participants' machine will not see user’s video who clicked turn off video. |
| False points | User clicks on switch video but it doesn’t work. The application still shows video and participants still see the video stream. |

Table 2.3: Use case - Switch audio stream

|  | |
| --- | --- |
| ID | UC - 3 |
| Title | Switch audio stream |
| Description | User can open or close own audio stream |
| Primary Actor | User |
| Preconditions | User is joined into application successfully |
| Post-conditions | Mute audio. |
| Flow of events | Firstly, the user needs to join room successfully. Secondly, Click on icon audio stream The third, if currently, audio is opening, the application will mute audio. |
| Success points | The participants' machine will not listen to user’s audio who clicked turn off audio. |
| False points | The participants' machine still listens to user’s audio who clicked turn off audio. |

Table 2.4: Use case - Leave room

|  | |
| --- | --- |
| ID | UC - 3 |
| Title | Leave room |
| Description | User leave room to finish |
| Primary Actor | User |
| Preconditions | User is joined into application successfully |
| Post-conditions | Back to join room screen |
| Flow of events | Firstly, the user needs to join room successfully. Secondly, Click on icon leave room button. |
| Success points | The application will back to join room screen. |
| False points | The application still in room screen. |

* 1. **Screen mockups**

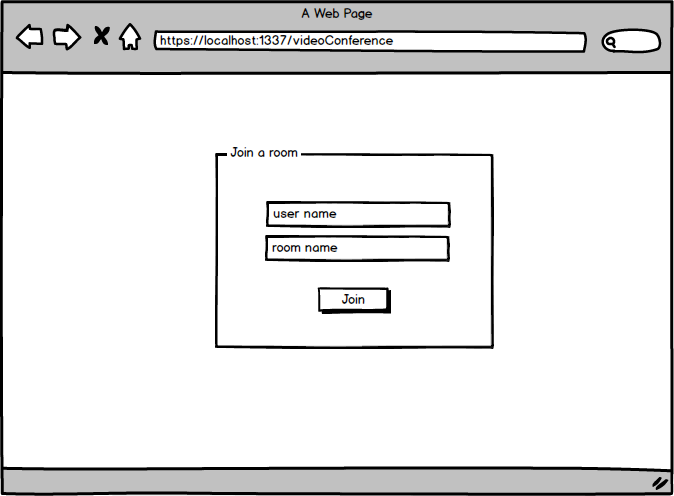
****

Figure 2.6: Screen mockup - Join room browser

Table 2.5: Description mockup - Join room browser screen

|  |  |  |
| --- | --- | --- |
| **Screen** | Screen mockup - Join room browser | |
| **Description** | This screen will be displayed a form for user input information to create or create room | |
| **Screen Access** | After navigating to videoconference on browser URL | |
| **Screen Content** | | |
| **Item** | **Type** | **Description** |
| username | TextInput | Input username |
| roomname | TextInput | Input roomname |
| Join | Button | Join room action |

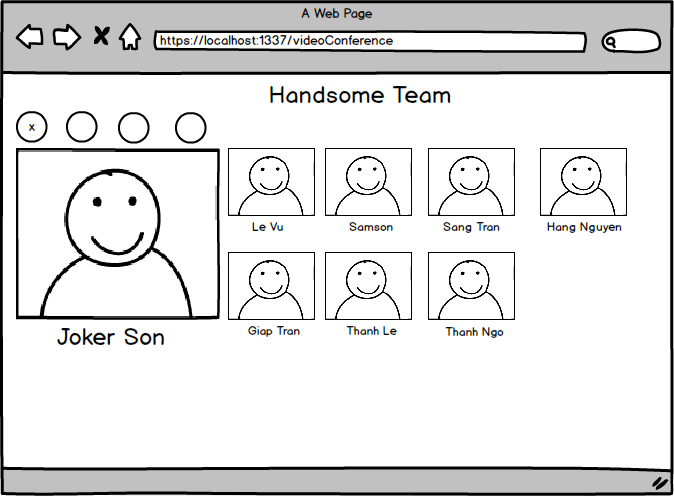


Figure 2.7: Screen mockup - Room browser

Table 2.6: Description mockup - Show room browser screen

|  |  |  |
| --- | --- | --- |
| **Screen** | Screen mockup - Room browser | |
| **Description** | This screen will be displayed inside the room. It will show your video and video of participants who are joining that room. | |
| **Screen Access** | After create or join room successfully. | |
| **Screen Content** | | |
| **Item** | **Type** | **Description** |
| leaveRoom | Button | Click to leave current room. |
| Turn off video | Button | Click to turn off your video, all of the participants in that room will not see your video. |
| Turn off audio | Button | Click to turn off your audio, all of the participants in that room will not see your audio. |
| Room name | Text | Show room name |
| Video | Video element | Display video of people who are joining the room. |
| User name | Text | Display name of the user who is joining the room. |

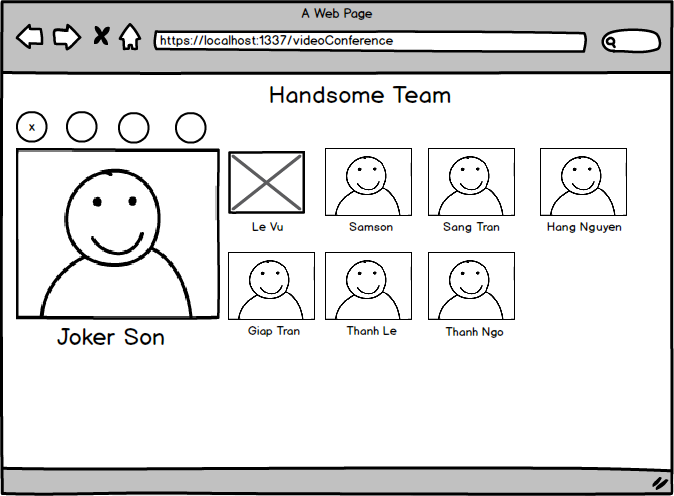
****

Figure 2.9: Screen mockup - Display on remote when turning off video

Table 2.7: Description mockup - Turn off video screen

|  |  |  |
| --- | --- | --- |
| **Screen** | Screen mockup - Display on remote when turning off video | |
| **Description** | This screen will be displayed inside the room. It will show your video and video of participants who are joining that room. If have any user turn off video, it will hide video of that user. | |
| **Screen Access** | After access room screen successful and one user turn off video. | |
| **Screen Content** | | |
| **Item** | **Type** | **Description** |
| leaveRoom | Button | Click to leave current room. |
| Turn off video | Button | Click to turn off your video, all of the participants in that room will not see your video. |
| Turn off audio | Button | Click to turn off your audio, all of the participants in that room will not see your audio. |
| Room name | Text | Show room name |
| Video | Video element | Display video of people who are joining the room. |
| Username | Text | Display name of the user who is joining the room. |

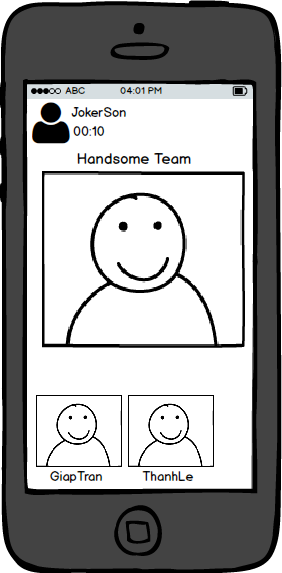
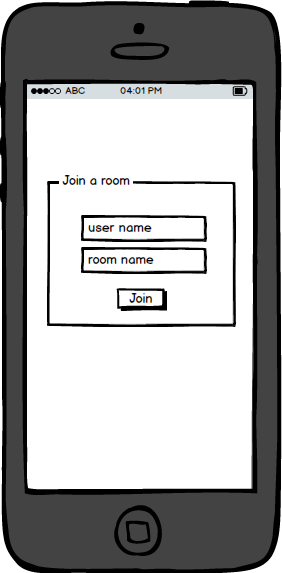
****

Figure 2.10: Screen mockup - Join room & Show room

Table 2.8: Description mockup - Join room mobile screen

|  |  |  |
| --- | --- | --- |
| **Screen** | Screen mockup - Join room & Show room | |
| **Description** | This screen will be displayed a form for user input information to create or create room | |
| **Screen Access** | After navigating to videoconference on browser URL | |
| **Screen Content** | | |
| **Item** | **Type** | **Description** |
| username | TextInput | Input username |
| roomname | TextInput | Input roomname |
| Join | Button | Join room action |

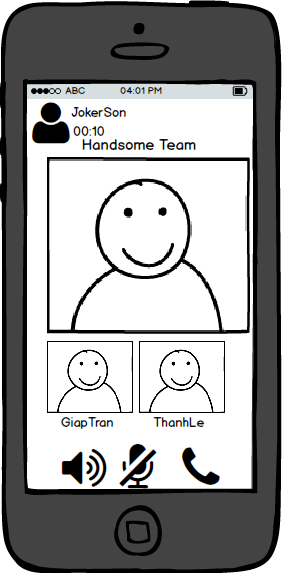
****

Figure 2.11: Screen mockup – Display functions

Table 2.9: Description mockup - Show room mobile screen

|  |  |  |
| --- | --- | --- |
| **Screen** | Screen mockup – Display functions | |
| **Description** | This screen will be displayed inside the room. It will show your video and video of participants who are joining that room. | |
| **Screen Access** | After create or join room successfully. | |
| **Screen Content** | | |
| **Item** | **Type** | **Description** |
| leaveRoom | Button | Click to leave current room. |
| Turn off video | Button | Click to turn off your video, all of the participants in that room will not see your video. |
| Turn off audio | Button | Click to turn off your audio, all of the participants in that room will not see your audio. |
| Room name | Text | Show room name |
| Video | Video element | Display video of people who are joining the room. |
| Username | Text | Display name of the user who is joining the room. |

* 1. **Sequence diagram**

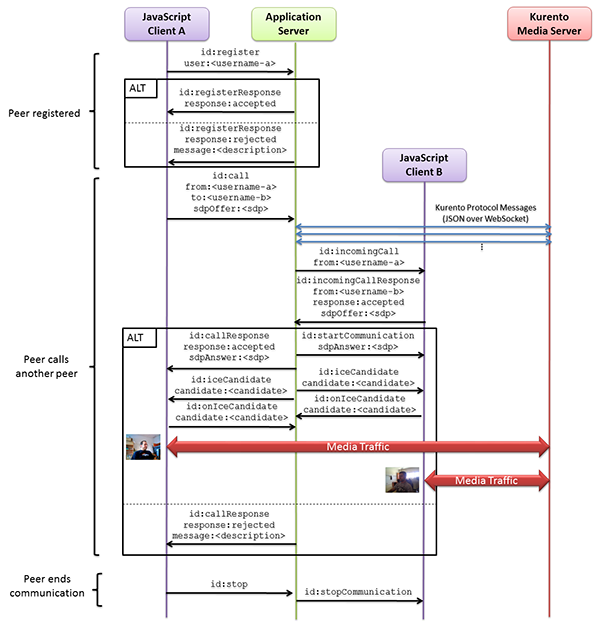
****

Figure 2.12: KMS sequence diagram

The client and the server communicate through a signaling protocol based on JSON messages over WebSocket ‘s. The normal sequence between client and application server logic is as follows:

* 1. User A is registered in the application server with his name
  2. User B is registered in the application server with her name
  3. User A issues a call to User B
  4. User B accepts the incoming call

The communication is established and media flows between User A and User B

One of the users finishes the video communication

As you can see in the diagram, SDP and ICE candidates need to be exchanged between client and server to establish the WebRTC connection between the Kurento client and server.

# DEPLOYMENT AND TESTING

## 3.1. Developing environment

### *3.1.1. Android Studio*

My application was developed on Android Studio that is official IDE of Android development.

I used Android Studio 2.3.1, built on May 20, 2017.

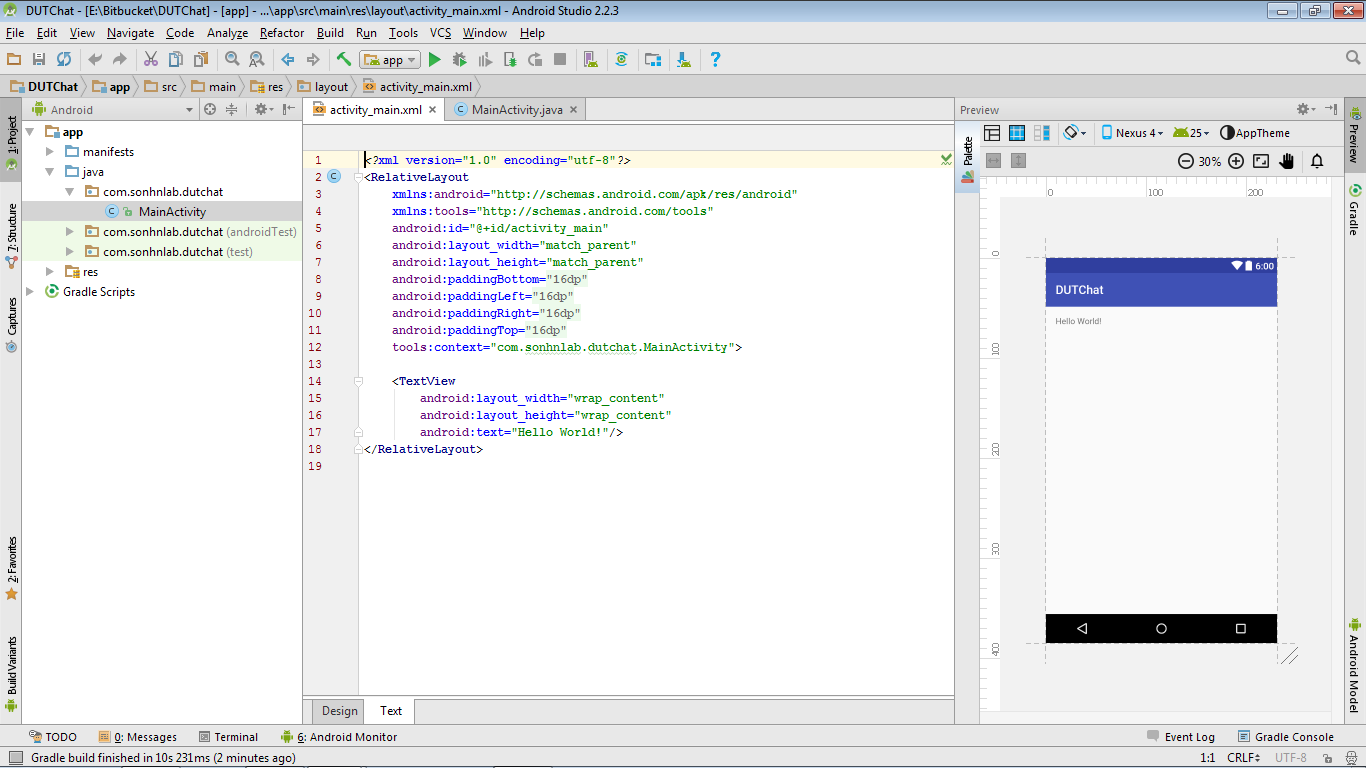


Figure 3.1. Initial project on Android Studio

Besides that, I used Virtual Devices of Android Studio as an emulator for testing in the development progress. Virtual Devices are regarded as the fastest android emulator.

### *3.1.2. Bitbucket*

Bitbucket is a web-based hosting service for source code and development projects that use either Mercurial or Git revision control systems that are owned by Atlassian. Not like similar Github, it allows developers to store private repositories. I prefer using Bitbucket because it can keep my source code privately and it supports issue tracking, too.

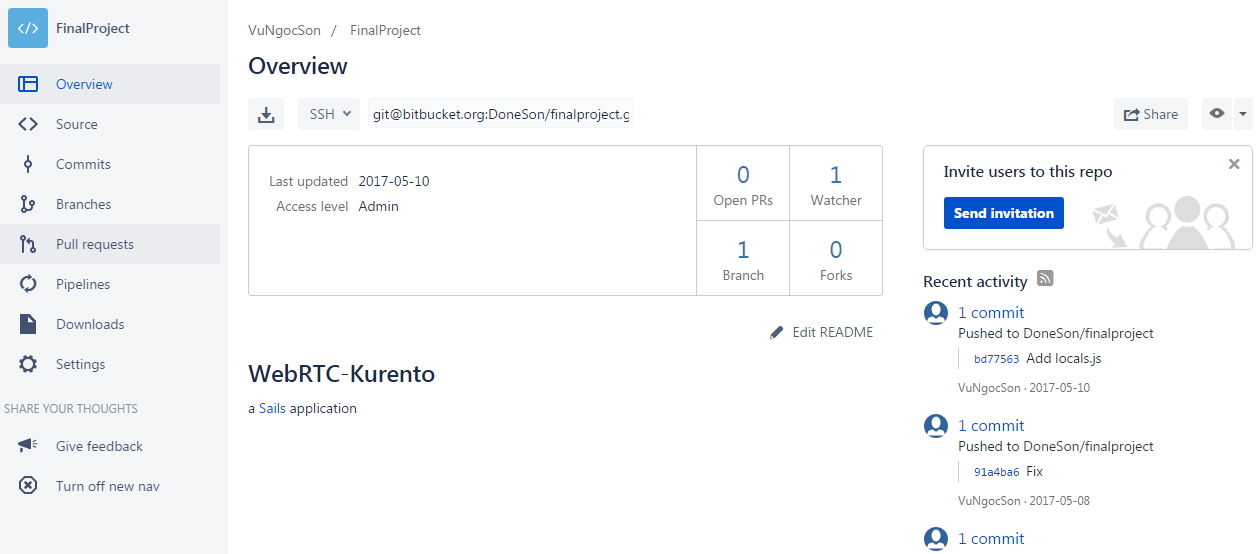


Figure 3.2.Manage source code on Bitbucket

## 3.2. Deployment

### *3.2.1. Deploy for user*

Now user can use this application to install APK file that I provide. Once I put the app on the Google Play Store, the user can download it conveniently.

### *3.2.2. Deploy for developer*

* ***Step 1: Clone source code***

Clone source code on bitbucket:

*https://DoneSon@bitbucket.org/DoneSon/finalproject.git*

* ***Step 2: Install 3rd party***
* Install KMS: *http://dockurento.readthedocs.io/en/stable/installation\_guide.html*
* Install Node.js, NPM:  
  [*https://nodejs.org/en/download/package-manager/*](https://nodejs.org/en/download/package-manager/)
* ***Step 3: Run***

Access directory and run by command: *node app.js*

* ***Step 4: Ready to use***

Access android application for mobile.  
Navigate to: *https://localhost:1337/videoConference*

Enjoy working.

## 3.4. Testing

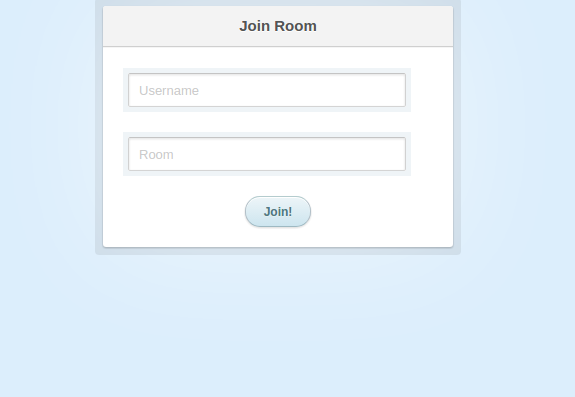
****

Figure 3.3: Demo - Join room

Join room is screen have user input username and room name to create or join the room.

After entering your name, room name that you want to join, let’s click join button to join the room. The system will get information to register and create new room on the server or join to an existing room.

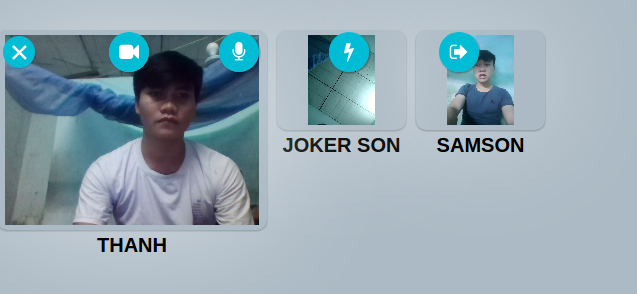
****

Figure 3.4: Demo – Room

This screen is displayed after joining a room. In this screen, you will see the video as well as listen to audio from participants who are joining in the room.

It will also have some button to click: Leave the room, Turn off/on video, Turn off/on audio.

For each participant, it will show their name at the bottom.

******

Figure 3.5: Demo - functions

****

Figure 3.6: Demo - Click on participant

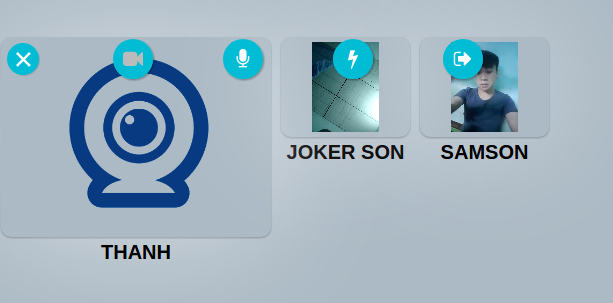
****

Figure 3.7: Demo - Turn your video

**

**

Figure 3.8: Demo - Turn off video - participant's screen

|  |  |
| --- | --- |
| 18470751_1372928632794618_119958844_n  Figure 3.9: Demo - Join room - mobile | 18492397_1372929079461240_1187669914_n  Figure 3.10: Demo - Room mobile |

|  |  |
| --- | --- |
| C:\Users\h\Desktop\18555157_1120270978116804_773246429_n.png  Figure 3.11: Demo - Room - Participants | C:\Users\h\Desktop\18623148_2279769632248129_501866863_n.png  Figure 3.12: Demo - Room - Participants |

## 3.5. Result evaluation

The application works well when internet connection good. It’s still unstable when multi-participant join to application

**CONCLUSION**

By the time of working on this project, I also improve my self-studying skills, research skills, and English skill, which is very useful to my future career.

During the research, theoretical research and deployment of technology application, the project has achieved the following results.

* Achieve theoretical results
* The application uses advanced technologies such as KMS, WebRTC APIs.
* The way to build an application from requirement to purchase to store
* Practical application
* The application permit multi participants join to the room.
* The application permit turns on/off media from local.

However, the project still has the following problems such as communicate multi devices.

* Drawbacks
* The system is simple with basic functions and difficult to apply in a large place.
* The UI is not good at some screen.
* Still not smooth at some screen.

In this context, I will survey some of the previously mentioned results which can be improved or extended further. This section also briefly describes some interesting ideas, which are worth investigating further. Here are these points:

* Future work.
* Support multi-platform: I will learn and develop this app on iOS.
* Config deeply to media stream to add more function.
* Improve the quality of video and audio stream.

# REFERENCES

[1] What is media server?  
*https://www.fiware.org/devguides/real-time-processing-of-media streams/whats-webrtc-and-whats-a-media-server/*

[2] Native code Android SDK.  
[*https://webrtc.org/native-code/android/#using-the-bundled-android-sdkndk*](https://webrtc.org/native-code/android/#using-the-bundled-android-sdkndk)

[3] What is WebRTC  
[*https://en.wikipedia.org/wiki/WebRTC*](https://en.wikipedia.org/wiki/WebRTC)

[4] WebRTC overview.  
[*https://webrtc.org/*](https://webrtc.org/)

[5] WebRTC code implement.  
[*https://www.html5rocks.com/*](https://www.html5rocks.com/)

[6] Kurento overview  
[*https://www.kurento.org/*](https://www.kurento.org/)

[7] What is SFU?  
 *https://webrtcglossary.com/sfu/*

[8] Compare SFU, MCU and Mesh solution?  
*https://testrtc.com/different-multiparty-video-conferencing/*