

PROFESSIONAL TRAINING REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology Degree in

Information Technology

By

Mr. SALMAN (Reg. No. 37120062)



DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119. TAMIL NADU

SEPTEMBER 2020



SCHOOL OF COMPUTING

BONAFIDE CERTIFICATE

This is to certify that this Professional Training Report is the bonafide work of **Mr. SALMAN** (Reg. No. 37120062) who underwent the professional training in “**PRIVATE MEETING**” under our supervision from September 2020 to January 2021.

Internal Guide

DR. JEBERSON RETNA RAJ.R. M.Tech.,Ph.D,

Head of the Department

DR. R. SUBHASHINI M.E.,Ph.D,

Submitted for Viva voce Examination held on_____

Internal Examiner

External Examiner

DECLARATION

I, Mr. SALMAN (Reg. No. 37120062) hereby declare that the Professional Training Report on “**PRIVATE MEETING**” done by me under the guidance of DR. JEBERSON RETNA RAJ. R. M.Tech.,Ph.D, at Sathyabama Institute Of Science And Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D, Dean**, School of Computing , **Dr.SUBHASHINI M.E., Ph.D.** , Head of the Department of Information and Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. Jeberson Retna Raj M.Tech., Ph.D.**,for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

ABSTRACT

In this Project, we analyse the feasibility of implementing live video streaming protocols into web applications with the use of WebRTC. As a result of demand the distribution of video content requires ever increasing bandwidth. Although, specialised programs exist to distribute video content efficiently, web pages have up until recently not been able to leverage these technologies. WebRTC could serve as a solution by enabling peer to- peer communication directly between browsers without any need for a server as an intermediary. Video-on-demand (VoD) services, such as YouTube, generate most of the Internet traffic today, and the popularity of video services is growing.

Service and CDN providers have to invest more and more in distribution networks, which creates pressure to innovate novel approaches. Peer-to-peer (P2P) streaming is a viable alternative that is scalable and can meet the increasing demand. The emerging HTML5 standard introduces APIs that give web browsers an ability to communicate directly with each other in real time. New standards also enable a setup, where browsers can act as P2P nodes. This project reviews whether the new HTML5 and WebRTC standards are a fit for P2P video streaming, evaluates the performance challenges and proposes solutions. Preliminary analysis indicates that HTML5 can be applied to VoD, but there are concerns.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No
	ABSTRACT	v
	LIST OF TABLES	vi
	LIST OF ABBREVIATIONS	viii
1	INTRODUCTION	1
	1.1 OUTLINE OF THE PROJECT	1
	1.2 LITERATURE STATEMENT	1
2.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	3
	2.1 PROBLEM STATEMENT	3
	2.2 OBJECTIVES	3
	2.3 APPLICATION	3
3.	EXPERIMENTAL OR MATERIALS AND METHODS	5
	3.1 OVERVIEW	5
	3.2 BACKGROUND	5
	3.3 GENERAL REQUIREMENTS	6
	3.4 IMPLEMENTATION	
4.	RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS	8
	4.1 OVERVIEW OF PLATFORM	8

	4.2 PLATFORM USED	8
	4.2.1 PYTHON	8
	4.2.2 JAVASCRIPT	9
5	SUMMARY AND CONCLUSIONS	16
	REFERENCES	
	APPENDIX	
	A.SCREENSHOTS	
	B. SOURCE CODE	

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
--------------	-----------

WebRTC	- Web Real-Time Communication
--------	-------------------------------



EDU FABRICA

Certificate of Participation

This certificate is proudly presented to

SALMAN

for participating in the

EDUFABRICA WINTER CAMP 2020

on the topic

DATA SCIENCE WITH PYTHON

conducted by

ETHICAL EDUFABRICA PVT. LTD.

at their DELHI Center



Niraj Singh
Director

Ethical Edufabrica Pvt. Ltd.



Microsoft Technology Associate

Mr. SALMAN

has successfully completed the requirements to be recognized as a Microsoft Technology Associate for

Security Fundamentals

Date of achievement: January 12, 2020
verify.certipoint.com WLSDu-FMXc


Satya Nadella
Chief Executive Officer

Microsoft
Technology Associate

CHAPTER 1

INTRODUCTION

1.1 OUTLINE OF THE PROJECT

The interest towards HTML5 is strongly increasing. Besides new visualization possibilities HTML5 brings a number of new opportunities to run increasingly sophisticated web applications with novel features such as parallel processing, data storage and real-time communication (RTC), without any additional add-ons or plug-ins. These features are available through APIs, which browser applications can call from JavaScript. It is rather clear that simple standalone applications can be implemented with HTML5, but the situation is more challenging when we start to analyze diverse applications with high computation and communication needs. The main use case we study is video-on-demand (VoD) using peer-to-peer (P2P) approach. Unlike a video conferencing use case, the P2P VoD is a good example of a communication intensive application that requires complex local operations to download and upload streaming content, to manage the download schedule, and to play the content. Similar operations are typical of many distributed applications and therefore the experiences gained have a wider applicability beyond the video streaming use case. However, implementing P2P streaming with HTML5 has still challenges. Firstly, the standards are under preparation, and therefore browser implementations have issues with supported functions, stability and interoperability. Secondly, video streaming sets substantial requirements on CPU performance and bandwidth consumption. In mobile space these issues are even more critical. This research proposes an experimental system for P2P VoD service applying several new HTML5 APIs that have recently been standardized.

1.2 LITERATURE OVERVIEW

WebRTC is currently under development by the W3C [4] and the IETF [3], but parts of the draft specification have already been partially implemented in Google Chrome and Mozilla Firefox. Whereas, previous W3C web standards allowed communication only

between a browser and a server, WebRTC allows direct communication between browsers, without any server in the middle. This theoretically allows for the implementation of peer-to-peer algorithms such as Coolstreaming or GridMedia.

The WebRTC API currently by Google Chrome and Mozilla Firefox consists of the following data communication functions [2]:

- **RTCPeerConnection:** The **RTCPeerConnection** interface provides a connection between peers to exchange data.
- **MediaStream:** The **MediaStream** interface represents a stream of audio or video data.
- **RTCDataChannel:** The **RTCDataChannel** interface describes a full-duplex data connection between two nodes.

CHAPTER 2

AIM AND SCOPE OF THE PRESENT INVESTIGATION

2.1 PROBLEM STATEMENT

This project is implementing a simple video conference application, where you can do an online video conference between two persons opening the same html page. This is done via WebRTC and Websockets.

2.2 OBJECTIVES

There are a number of facets to peer-to-peer communications and video-conferencing in HTML covered by this specification:

- Connecting to remote peers using NAT-traversal technologies such as ICE, STUN, and TURN.
- Sending the locally-produced tracks to remote peers and receiving tracks from remote peers.
- Sending arbitrary data directly to remote peers.

2.3 Applications

WebRTC enables developers to facilitate audio, video, and data communication between two browsers without using any plugin. Hence, it becomes easier for developers to simplify browser communication and improve user experience. A web developer can use WebRTC to facilitate peer-to-peer communication between two browsers without putting extra time and effort. He can even use the HTML5 specification to facilitate audio, video, or data communication between two browsers.

WebRTC can be used anywhere a human is interacting with a computer system. These are the areas where the fastest adoption is occurring:

1. Google Meet and Hangouts
2. Facebook Messenger
3. Discord
4. Amazon Chime
5. Houseparty
6. Appear.in
7. Gotomeeting
8. Peer5

CHAPTER 3

EXPERIMENTAL OR MATERIALS AND METHODS USED

3.1 Overview

This project is implementing a simple video conference application, where you can do an online video conference between two persons opening the same html page. This is done via WebRTC and Websockets.

3.2 BACKGROUND

There are various ways to implement streaming with P2P technologies. In this work we focus on how to apply BitTorrent and HTML5 for P2P streaming [1]. However, regular BitTorrent is focused on the P2P file sharing rather than the VoD streaming [2]. For that reason a few modifications are required to guarantee the order and timeliness of video pieces and furthermore, to ensure an uninterrupted viewing experience. BitTorrent functionality can be added to web browsers via a few different methods. Plugins and in-application webservers are already available, but our proposed JavaScript based P2P/BitTorrent client is still to be realized. Having a platform independent P2P client, running on PCs and mobile devices and dealing with the issues related with the differing platforms is the goal. HTML5 Web Real Time Communication (WebRTC) standard plays a key role in browser-to-browser communication. The WebRTC standard enables real-time communication and also introduces UDP based communication to web browsers to complement the normally used TCP communication with HTTP. The standard enables three modes and APIs for browser-to-browser communication: Peer Connection API offers connection establishment, Media Stream API video streaming functions and Data Channel API arbitrary data sharing operations [3].

3.3 GENERAL REQUIREMENTS

a. SOFTWARE REQUIREMENTS

- Operating system : Windows XP/7
- Coding Language : Python, JavaScript
- Tool : XAMP or WAMP

b. HARDWARE REQUIREMENTS

- Processor: Intel® Core(TM) i5 2520M CPU @ 2.50Ghz (Dual core)
- RAM: 2048 MB-2.00 GB
- Operating System: Windows 7 Service Pack 1 64-bit

3.4 IMPLEMENTATION

The design mirrors the BitTorrent architecture and the same entities Tracker, Seeder and Peers - are used. Figure 1 illustrates the functionality. Initially a Peer creates a torrent file containing the metadata of the video, hashes of each piece of the video, and uploads the torrent file to the Tracker (1). The actual video is uploaded to the Seeder (2). When another Peer requests the same video, the Peer first obtains the respective tracker file and starts downloading the video pieces from the Seeder or the original Peer or both (3). Eventually, as the video becomes popular, missing pieces are shared by Peers, which decreases the load on the original Peer and Seeder. Periodic access to the Tracker for requesting the latest Peer information is needed (4).

The video publishing process begins when a user starts a web application and selects a video to share. This phase involves the creation of the meta-data of the video file for the Tracker, and moving the file to a location accessible to the web application. Part of the meta-data creation is the generation of cryptographic MD5 hash values.

The HTML5 File API is initially used to store the video file to be distributed, and once that process is complete, the file is moved to permanent storage using the IndexedDB API. Finally, once the file hashes have been generated, the newly created torrent file and the data file are sent to a Seeder for wider distribution. When a user requests a video, the meta-data of the video file (torrent file) is downloaded to the user's browser along with a list of Peers who currently have the video available.

Then the web application starts downloading the pieces of the video file from the Peers using the Data Channel API. When enough pieces for the beginning of the video have been downloaded and their MD5 checksum values have been successfully verified, the playback will start using HTML5's Video element. Offline Application Caching API can be used to load and execute a copy of the HTML5 JavaScript code stored on the client, even when the web browser is not connected to the Internet. This API allows playing the stored videos also when no connection to Internet is available.

CHAPTER 4

RESULTS AND DISCUSSION, PERFORMANCE ANALYSIS

4.1 OVERVIEW OF PLATFORM

This project is implementing a simple video conference application, where you can do an online video conference between two persons opening the same html page. This is done via WebRTC and Websockets.

4.2 PLATFORM USED

4.2.1 Python:

Python is a general-purpose interpreted, interactive, high level, object-oriented scripting language. Fully dynamic type system and automatic memory management. Python is designed to be highly readable. It uses English keywords frequently. It was created by Guido van Rossum during 1985- 1990. Open Source : Support all os like (Windows, Linux, Mac..). Minimal code, Clarity, Consistency, Easy to teach. Python is a general-purpose language, which means it can be used to build just about anything, which will be made easy with the right tools/libraries.

1. Beginner Friendliness:

Python was designed to be easy to understand and fun to use Python, many find coding in Python a satisfying experience. Thus, Python has gained popularity for being a beginner-friendly language.

2. Easy to understand

Python reads like English, which takes a lot of syntax-learning stress off

coding beginners. Python handles a lot of complexity for you, so it is very beginner-friendly.

3. Advantages of Python

- Extensive Support Libraries
- Integration Feature
- Improved Programmer's Productivity
- Productivity

4. Limitations or Disadvantages of Python

- Gets Slow in Speed
- Weak in Mobile Computing
- Run-time Errors
- Not Easy to Maintain

4.2.1 JavaScript :

Javascript is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Javascript:

- Javascript is the most popular programming language in the world and that makes it a programmer's great choice. Once you learnt Javascript, it helps you developing great front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.JS etc.
- Javascript is everywhere, it comes installed on every modern web browser and so to learn Javascript you really do not need any special environment setup. For example Chrome, Mozilla Firefox , Safari and every browser you know as of today, supports Javascript.

- Javascript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.
- JavaScript usage has now extended to mobile app development, desktop app development, and game development. This opens many opportunities for you as Javascript Programmer.
- Due to high demand, there is tons of job growth and high pay for those who know JavaScript. You can navigate over to different job sites to see what having JavaScript skills looks like in the job market.
- Great thing about Javascript is that you will find tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.

CHAPTER 5

SUMMARY AND CONCLUSIONS

5.1 SUMMARY

HTML5 can be used to perform P2P video streaming between browsers without any additional plug-ins. However, some limitations exist. Part of the challenges arise from limitations in current browser implementations, part of performance issues. The resources available through browsers are much more limited than for native applications. Further study would be needed to understand how much these limitations are fundamental in the browser technology or simply initial problems of premature implementations. The work does not end by implementing a P2P based solution for media streaming, but performance measurements over various platforms, including mobile devices, must be carried through. Mobile devices offer limited CPU and bandwidth capacity, and thus streaming applications must be well optimized for the scarce resources.

As we do not have a clear understanding of the load distribution between the different elements with an HTML5 based VoD service, the next step is to implement a working P2P VoD client on the latest Firefox versions, find out the relative importance of each part and locate the largest bottlenecks. The performance of the Data Channel API, file hashing and video decoding on the target browser are the main focus areas. The proposed P2P VoD design and implementation only works over a homogenous P2P network where all participants are using the HTML5 based solution, but for the end user it would be useful to get access to existing P2P networks such as BitTorrent. Allowing access to the vast number of existing content sharing and video streaming solutions is another avenue for future work.

REFERENCES

- [1] A. Bakker, R. Petrocco, M. Dale, G. J., G. V., R. D., and P. J., "Online Video Using BitTorrent and HTML5 Applied to Wikipedia," in Peer-toPeer Computing (P2P), 2010 IEEE Tenth International Conference on, Aug. 2010, pp. 1–2.

- [2] J. Mol, A. Bakker, J. Pouwelse, D. Epema, and H. Sips, "The Design and Deployment of a BitTorrent Live Video Streaming Solution," in Multimedia, 2009. ISM '09. 11th IEEE International Symposium on, Dec. 2009, pp. 342–349.

- [3] S. Loreto and S. Romano, "Real-time communications in the web: Issues, achievements, and ongoing standardization efforts," Internet Computing, IEEE, vol. 16, no. 5, pp. 68–73, Sept.-Oct. 2012.

- [4] M. Bynens, "jsPerf – MD5 Shootout Revision 14," October 2012. [Online]. Available: <http://jsperf.com/md5-shootout/14>

Source Code

```
// UI Manager for the Chat app
function myjsapp(peerClient) {
    var chatHistory = {};
    var chatPanel = {};

    var cookie = {
        // Read cookie
        get : function getCookie (name) {
            var cookies = {};
            var c = document.cookie.split('; ');
            for (i = c.length - 1; i >= 0; i--) {
                var C = c[i].split('=');
                cookies[C[0]] = C[1];
            }
            return cookies[name] || null;
        },

        // create cookie
        set : function createCookie (name, value, minutes) {
            if (minutes) {
                var date = new Date();
                date.setTime(date.getTime() + (minutes * 60 * 1000));
                var expires = "; expires=" + date.toGMTString();
            } else
                var expires = "";
            document.cookie = name + "=" + value + expires + "; path=/";
        },
```

```
remove : function deleteCookie (name) {  
    var date = new Date();  
    date.setTime(date.getTime() - 60 * 1000);  
    document.cookie = name + "="; expires=" + date.toGMTString() + "; path="/;  
}  
};
```

```
function EventListeners() {  
    $('#peer-id').tooltip()
```

```
function connectToPeer() {  
    var id = $('#inputPeerUserId').val().trim();  
    if(id) {  
        peerClient.connectTold(id.toLowerCase())  
        $('#inputPeerUserId').val("")  
    }  
}  
$('#connect-btn').click(function (event) {  
    connectToPeer()  
});
```

```
$('#inputPeerUserId').keypress(function(event) {  
    if (13 == event.which) {  
        connectToPeer()  
    }  
});
```

```
$(document).on('click', '.peeruser', function() {  
    var id = $(this).text()  
    $('#inputPeerUserId').val(id)  
    connectToPeer()  
});
```

```
Element.prototype.remove = function() {  
    this.parentElement.removeChild(this);  
}
```

```
$('#peer-id').click(function (event) {  
    var textArea = document.createElement("textarea");  
    // Avoid flash of white box if rendered for any reason.  
    textArea.style.background = 'transparent';  
    textArea.value = $(this).text();  
    document.body.appendChild(textArea);  
    textArea.select();
```

```
    try {  
        var successful = document.execCommand('copy');  
        var msg = successful ? 'successful' : 'unsuccessful';  
        console.log('Copying text command was ' + msg);  
        textArea.remove();  
    } catch (err) {  
        console.log('Oops, unable to copy');  
    }  
});
```

```
$('.end-call').click(function (event) {  
    // clear CSS for mute buttons  
    $('.mute-audio, .mute-video').removeClass('btn-success').addClass('btn-secondary')  
    // End established call  
    peerClient.endCall();  
})
```

```
$('#user-name').keypress(function (event) {  
    if (13 == event.which) {  
        var username = $('#user-name').val().trim();  
        $('#getUserModal').modal('hide')  
        if(cookie.get('username') != username)  
            startPeerClient(username)  
    }  
})
```

```
$('.username-done').click(function (event) {  
    var username = $('#user-name').val().trim();  
    if(cookie.get('username') != username)  
        startPeerClient(username)  
})
```

```
$('.accept-call').click(function (event) {  
    // End established call  
    peerClient.acceptIncomingCall();  
})
```

```
$('.reject-call').click(function (event) {
```

```

        // End established call
        peerClient.rejectIncomingCall();
    })

    $('.mute-audio').click(function (event) {
        if($(this).hasClass('btn-secondary')) {
            $(this).removeClass('btn-secondary').addClass('btn-success')
            // End established call
            peerClient.muteAudio(false);
        } else {
            $(this).removeClass('btn-success').addClass('btn-secondary')
            peerClient.muteAudio(true);
        }
    })

    $('.mute-video').click(function (event) {
        if($(this).hasClass('btn-secondary')) {
            $(this).removeClass('btn-secondary').addClass('btn-success')
            // End established call
            peerClient.muteVideo(false);
        } else {
            $(this).removeClass('btn-success').addClass('btn-secondary')
            peerClient.muteVideo(true);
        }
    })
}

function appendToHistory(id, message, isSent) {
    if(chatHistory[id]) {

```

```

        var hist = chatHistory[id];
        var fromTxt = isSent ? 'You' : id
        var msg = $('<li><b>' + fromTxt + ': </b></li>').append('<span>' + message +
'</span>')
        hist.append(msg)
        .scrollTop(hist[0].scrollHeight);
    }
}

```

```

function startPeerClient(username) {
    // TODO - Set title
    cookie.set('username', username);
    peerClient.connectToServerWithId(username);
}

```

```

// Show Username Modal
var username = cookie.get('username');
if(username) {
    $('#user-name').val(username)
    startPeerClient(username)
} else {
    $('#getUserNameModal').modal('show')
}

```

```

EventListeners();

```

```

return {
    setPeerId : function (id) {
        $('#peer-id').text(id);
    }
}

```

```
},
```

```
createChatWindow: function(id) {  
    var toPeerId = id;  
    var panel = $('<div class="panel panel-primary chat-div"><div class="panel-  
heading"></div>' +  
        '<div class="panel-body"></div><div class="panel-footer">' +  
        '<div class="form-inline"><div class="form-group">' +  
        '</div></div></div></div>')  
  
    var title = $('<span class="panel-title"></span>').text(toPeerId)  
    var history = $('<ul class="chatHistory"></ul>')  
    var message = $('<input type="text" class="form-control" placeholder="Enter  
Message">')  
    var sendBtn = $('<button type="button" class="btn btn-outline-  
primary">Send</button>')  
    var callButton = $('<a class="portfolio-link">');  
    var videoCall = $('<i class="fa fa-video-camera fa-2x call-icon" aria-  
hidden="true"></i>');  
    var audioCall = $('<i class="fa fa-phone fa-2x call-icon" aria-  
hidden="true"></i></a>');  
  
    callButton.append(audioCall).append(videoCall);  
  
    chatHistory[toPeerId] = history  
    chatPanel[toPeerId] = panel  
  
    $('<.panel-heading', panel).append(title).append(callButton)  
    $('<.panel-body', panel).append('<span class="text-primary">You can now start  
chatting</span>').append(history)
```

```
$('.form-group', panel).append(message).append(sendBtn)
```

```
$('.chat-container > div').append(panel);
```

```
$('.panel-heading', panel).click(function () {  
    var panelBody = $(".panel-body, .panel-footer", $(this).parent());  
    if(panelBody.hasClass("hide")) {  
        panelBody.removeClass("hide")  
        panel.removeClass('min')  
    } else {  
        panel.addClass('min')  
        panelBody.addClass("hide")  
    }  
})
```

```
message.keypress(function(event) {  
    if (13 == event.which) {  
        var msgText = $(this).val().trim()  
        if(msgText) {  
            peerClient.sendMessage(toPeerId, msgText)  
            appendToHistory(toPeerId, msgText, true)  
            $(this).val("")  
        }  
    }  
});
```

```
sendBtn.click(function(event) {  
    var msgText = message.val().trim()
```

```

    if(msgText) {
        peerClient.sendMessage(toPeerId, msgText)
        appendToHistory(toPeerId, msgText, true)
        message.val("").focus()
    }
});

```

```

audioCall.click(function (event) {
    // initializeLocalVideo()
    var isVideoCall = false;
    peerClient.makeCall(toPeerId, isVideoCall);
    return false
})

```

```

videoCall.click(function (event) {
    // initializeLocalVideo()
    var isVideoCall = true;
    peerClient.makeCall(toPeerId, isVideoCall);
    return false
})
// TODO - Hide panels if more than 3
},

```

```

appendHistory : appendToHistory,

```

```

closeChatWindow : function (id) {
    if(chatPanel[id]) {
        chatPanel[id].remove()
    }
}

```

```

        delete chatPanel[id]
        delete chatHistory[id]
    }
},
showVideoCall : function (options) {
    $('#videoCallPanel').modal('show')
    if(options['video'])
        $('#videoCallPanel .title').text('Video Call')
    else
        $('#videoCallPanel .title').text('Voice Call')
},
showIncomingCall : function (peerId, options) {
    $('#callConfirmationModal').modal('show')
    if(options['video'])
        var txt = "Incoming Video call from : " + peerId
    else
        var txt = "Incoming Voice call from : " + peerId
    $('#callConfirmationModal .peer-name').text(txt)
},
closeVideoCall : function () {
    $('.end-call').click()
},
setTheirVideo : function (stream) {
    var video = document.getElementById('their-video');
    if (typeof video.srcObject == "object") {
        video.srcObject = stream;
    } else {
        video.src = URL.createObjectURL(stream);
    }
}

```

```

    }
  },
  setMyVideo : function (stream) {
    // $('#my-video').prop('src', stream);
    var video = document.getElementById('my-video');
    if (typeof video.srcObject == "object") {
      video.srcObject = stream;
    } else {
      video.src = URL.createObjectURL(stream);
    }
  },
  showError : function (msg) {

},
  updateOnlineUsers : function (users) {
    var list = $('#onlinepeers')
    list.empty()
    if(users.length == 0) {
      var usr = '<li>Looks like no one is online</li>'
      list.append(usr);
      return
    }
    for (var i = 0; i < users.length; i++) {
      var usr = '<li class="peeruser">'+ users[i] + '</li>'
      list.append(usr);
    }
  }
};

```

```
}
```

```
var myapp, peerapp;
```

```
$(document).ready(function () {  
    myapp = myjsapp(peerapp);  
});
```

Peer-client.js

```
peerapp = (function() {  
    'use strict';
```

```
    console.log("Peer client started");
```

```
    var PEER_SERVER = 'my-peer.herokuapp.com';
```

```
    var PORT = 443;
```

```
    var connectedPeers = {};
```

```
    var myPeerID = generateRandomID(7);
```

```
    var peer;
```

```
    var peerIdAlreadyTakenCount = 3;
```

```
    // Compatibility shim
```

```
    navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia  
    || navigator.mozGetUserMedia;
```

```

// Connect to server
function connectToServerWithId(peerId) {
    myPeerID = peerId || myPeerID;
    myPeerID = myPeerID.toLowerCase();
    if(peer && peer.disconnected == false) {
        peer.disconnect()
    }
    peer = new Peer(myPeerID, { host: PEER_SERVER, port: PORT, path: '/', secure:
true });
    peerCallbacks(peer);
}

// var peer = new Peer({ host: 'my-peer.herokuapp.com', port: '443', path: '/', secure:
true });

// connectToServerWithId(myPeerID);
console.log(peer)

initializeLocalMedia({'audio': true, 'video': true});

// Generate random ID
function generateRandomID(length) {
    var chars = '123456789abcdefghijklmnopqrstuvwxyz'
    var result = "";
    for (var i = length; i > 0; --i) result += chars[Math.floor(Math.random() *
chars.length)];
    return result;
}

// Data channel
// Handle a Text and File connection objects

```

```

function connect(c) {
  console.log(c)
  // Handle a chat connection.
  if (c.label === 'chat') {
    // c.peer
    myapp.createChatWindow(c.peer)

    c.on('data', function(data) {
      console.log(c.peer + ' : ' + data);
      // Append data to chat history
      myapp.appendHistory(c.peer, data)
    });

    c.on('close', function() {
      delete connectedPeers[c.peer];
      myapp.closeChatWindow(c.peer)
    });
    connectedPeers[c.peer] = c;
  } else if (c.label === 'file') {
    c.on('data', function(data) {
      // If we're getting a file, create a URL for it.
      if (data.constructor === ArrayBuffer) {
        var dataView = new Uint8Array(data);
        var dataBlob = new Blob([dataView]);
        var url = window.URL.createObjectURL(dataBlob);
        // $('##' + c.peer).find('.messages').append('<div><span class="file">' +
        //   c.peer + ' has sent you a <a target="_blank" href="' + url +
        '>file</a>.</span></div>');
      }
    });
  }
}

```

```
    });  
  }  
}
```

// Handle Audio and Video Calls

```
function callConnect(call) {
```

```
  // Hang up on an existing call if present
```

```
  if (window.existingCall) {
```

```
    window.existingCall.close();
```

```
  }
```

```
  // Wait for stream on the call, then set peer video display
```

```
  call.on('stream', function(stream) {
```

```
    myapp.setTheirVideo(stream)
```

```
  });
```

```
  // UI stuff
```

```
  window.existingCall = call;
```

```
  call.on('close', function() {
```

```
    console.log("Call Ending")
```

```
    myapp.closeVideoCall()
```

```
  });
```

```
}
```

```
function peerCallbacks(peer) {
```

```
  peer.on('open', function(id) {
```

```
    console.log('My peer ID is: ' + id);
```

```

    console.log(new Date());
    myapp.setPeerId(id);
    fetchOnlinePeers();
  });

  peer.on('connection', connect);

  peer.on('call', function(call) {
    console.log("Receiving a call")
    console.log(call)
    // New call requests from users
    // Ask Confirm before accepting call
    // if(window.incomingCall) {
    //   window.incomingCall.answer()
    //   setTimeout(function () {
    //     window.incomingCall.close();
    //     window.incomingCall = call
    //     myapp.showIncomingCall(call.peer);
    //   }, 1000)
    // } else {
    if(window.existingCall) {
      // If already in a call, rejecting the new calls
      rejectIncomingCall(call)
    } else {
      window.incomingCall = call
      myapp.showIncomingCall(call.peer, call.options.metadata);
    }
  })
}

```

```
});
```

```
peer.on('close', function(conn) {  
    // New connection requests from users  
    console.log("Peer connection closed");  
});
```

```
peer.on('disconnected', function(conn) {  
    console.log("Peer connection disconnected : " + conn);  
    console.log(new Date());  
    if(conn !== myPeerID) {  
        return  
    }  
    setTimeout(function () {  
        connectToServerWithId();  
    }, 5000);  
  
    // peer.reconnect()  
});
```

```
peer.on('error', function(err) {  
    console.log(new Date());  
    console.log("Peer connection error:")  
    console.log(err)  
    if("unavailable-id" === err.type) {  
        // ID Already taken, so assigning random ID after 3 attempts  
        peerIdAlreadyTakenCount++;  
        if(peerIdAlreadyTakenCount >= 3) {
```

```

        peerIdAlreadyTakenCount = 0;
        myPeerID = generateRandomID(4);
    }
} else if ("peer-unavailable" == err.type) {
    myapp.showError(err.message)
}
});
};

```

```

function connectTold(id) {
    if(!id || peer.disconnected)
        return;
    var requestedPeer = id;
    if (!connectedPeers[requestedPeer]) {
        // Create 2 connections, one labelled chat and another labelled file.
        var c = peer.connect(requestedPeer, {
            label: 'chat',
            serialization: 'none',
            metadata: { message: 'hi i want to chat with you!' }
        });
        c.on('open', function() {
            connect(c);
        });
        c.on('error', function(err) { alert(err); });

        // File Sharing
        // var f = peer.connect(requestedPeer, { label: 'file', reliable: true });
        // f.on('open', function() {
    }
}

```

```

        // connect(f);
        // });
        // f.on('error', function(err) { alert(err); });
    }
}

```

```

function sendMessage(peerId, msgText) {

```

```

    if(connectedPeers[peerId]) {
        var conn = connectedPeers[peerId]
        conn.send(msgText)
    }
}

```

```

function initializeLocalMedia(options, callback) {

```

```

    if(options) {
        options['audio'] = true;
        if(options['video'])
            options['video'] = true;
    } else {
        options['audio'] = true;
        options['video'] = false;
    }
}

```

```

// Get audio/video stream

```

```

navigator.getUserMedia(options, function(stream) {
    // Set your video displays

```

```

    window.localStream = stream;
    myapp.setMyVideo(window.localStream)
    if(callback)
        callback();
}, function(err) {
    console.log("The following error occurred: " + err.name);
    alert('Unable to call ' + err.name)
});
}

```

```

function makeCall(callerID, isVideoCall) {
    console.log("Calling..." + callerID)

    var options = {audio: true};
    if(isVideoCall)
        options['video'] = true;

    initializeLocalMedia(options, function() {
        myapp.showVideoCall(options)
        var call = peer.call(callerID, window.localStream, { 'metadata' : options });
        callConnect(call)
    });
}

```

```

function acceptIncomingCall() {
    var call = window.incomingCall;
    var metadata = call.options.metadata;
    console.log(metadata);
}

```

```
initializeLocalMedia(metadata, function() {  
    call.answer(window.localStream);  
    myapp.showVideoCall(metadata);  
    callConnect(call)  
});  
}
```

```
function rejectIncomingCall(reCall) {  
    var call = reCall || window.incomingCall;  
    var metadata = call.options.metadata;  
    console.log(metadata);  
    console.log("Rejecting incomingCall")  
    call.answer();  
    setTimeout(function () {  
        call.close();  
    }, 1000)  
}
```

```
function endCall() {  
    if(window.existingCall)  
        window.existingCall.close();  
    window.existingCall = null  
}
```

```
function muteAudio(status) {  
    if(status == false)  
        status = false
```

```
else
    status = true
if(window.localStream) {
    var audioTracks = window.localStream.getAudioTracks()
    if(audioTracks && audioTracks[0])
        audioTracks[0].enabled = status;
    }
}
```

```
function muteVideo(status) {
    if(status == false)
        status = false
    else
        status = true
    if(window.localStream) {
        var videoTracks = window.localStream.getVideoTracks()
        if(videoTracks && videoTracks[0])
            videoTracks[0].enabled = status;
        }
    }
```

```
function closeConnection(id) {
    var conns = peer.connections[peerId];
    if(connectedPeers[peerId]) {
        var conn = connectedPeers[peerId]
        conn.send(msgText)
    }
}
```

// Make sure things clean up properly.

```
window.onunload = window.onbeforeunload = function(e) {  
  if (!!peer && !peer.destroyed) {  
    peer.destroy();  
  }  
};
```

```
function fetchOnlinePeers() {  
  $.ajax("https://" + PEER_SERVER + "/peerjs/" + myPeerID + "/onlineusers")  
  .done(function( data ) {  
    // console.log(data);  
    if(data.msg == 'Success') {  
      data.users.splice(data.users.indexOf(myPeerID), 1)  
      myapp.updateOnlineUsers(data.users)  
    }  
  });  
}
```

// Update Online users on every 5 seconds

```
setInterval(function () {  
  fetchOnlinePeers()  
}, 5000)
```

```
return {  
  makeCall : makeCall,  
  endCall : endCall,  
  sendMessage : sendMessage,
```

```
connectTold : connectTold,  
connectToServerWithId : connectToServerWithId,  
acceptIncomingCall : acceptIncomingCall,  
rejectIncomingCall : rejectIncomingCall,  
muteAudio : muteAudio,  
muteVideo : muteVideo  
}  
})();
```

Screenshots





