HackMe 2.0 - XperienZ

HKME-26 – Live Video Streaming Using WebRTC

Version 1.0

Published on 01/09/20177

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 01/09/2017 | 1.0 | First Draft | Saurabh Nutan Sohoni |

**References**

|  |  |
| --- | --- |
| **No.** | **Artefact Name** |
|  | <https://www.html5rocks.com/en/tutorials/webrtc/basics/> |
| 3. | <https://bitbucket.org/webrtc/codelab>  <https://w3c.github.io/webappsec-secure-contexts/> |

Contents

[1. Introduction 4](#_Toc471766803)

[1.1 Purpose 4](#_Toc471766804)

[1.2 Scope 4](#_Toc471766805)

[1.3 Hardware requirements 4](#_Toc471766806)

[1.4 Software requirements 4](#_Toc471766807)

[2. OVERVIEW OF CONCEPT AND METHODOLOGY USED 4](#_Toc471766808)

[2.1 What is socket.io? 4](#_Toc471766809)

[2.2 How to write a sample socket.io application? 5](#_Toc471766810)

[2.3 What is WebRTC? 6](#_Toc471766811)

[2.4 How to write a simple video stream application using WebRTC? 6](#_Toc471766812)

[3. VIDEO CHAT APPLICATION IN ACTION 8](#_Toc471766813)

[3.1 Access to media devices and HTTPS server 8](#_Toc471766814)

[3.2 Join the chat room 8](#_Toc471766815)

# Introduction

# Purpose

The purpose of this document is to guide developers in creating a sample Video Chat application using WebRTC for streaming of video and socket.io for signaling between WebRTC clients.

# Scope

Scope of this project is limited to development of a video chat application between a maximum of two clients at a given time. This application is targeted for mobile users on android and iOS. WebRTC standards are currently supported in Chrome (version >=7.0), Chrome for Android, Firefox (version >= 6.0) and Safari Mobile (version >=4.2). The application will make use of built in camera for capturing vide

# Hardware requirements

Since this project aims at extending the capabilities of a smartphone, the major hardware required will be a smartphone with built in camera (preferably front camera, as video chat will only be desirable when end users are viewing each other and recording their video at the same time).

# Software requirements

This project aims at creating a simple web app. Hence the software requirements are minimal and are given below:

1. Web Browser: Google Chrome (version >=7.0), Firefox (version >= 6.0)
2. Mobile Browser: Chrome for android (version >= 3.0), Firefox (version >= 6.0), Safari Mobile (version >= 4.2)
3. Text editor software such as Notepad++, Sublime Text, etc.

# OVERVIEW OF CONCEPT AND METHODOLOGY USED

This section lists some important concepts of socket.io and WebRTC protocol. It also provides an insight into essential steps not to be missed in order for the desired application to work smoothly.

# What is socket.io?

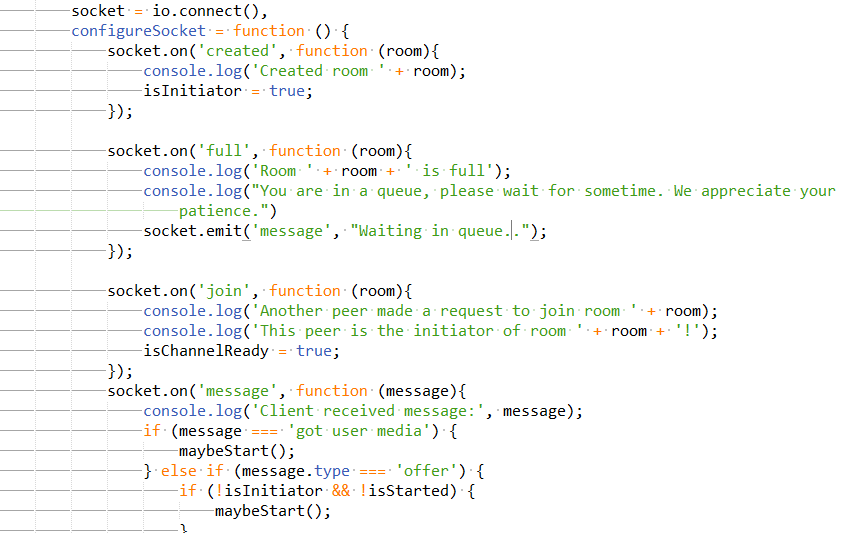
In a conventional web application, communication between a client and a server is driven by the client. Until and unless the client makes a request to the server, the server doesn’t interact or send any kind of data to any client machine. The role of server is to respond to client requests based on authorization, authentication and the type of data expected in response.

Socket.io enables real time bi-directional communication. In this type of application, even server can request for some kind of information from the client and client may choose to respond. This communication is event based.

# How to write a sample socket.io application?

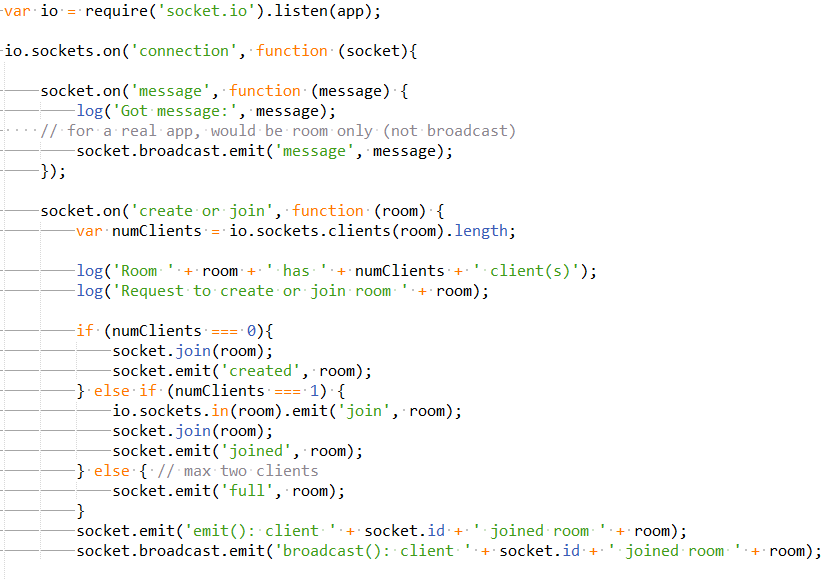
As stated earlier, socket.io application is based on communication via event driven model. Either of the client or server emits events (mostly, custom events are used). Listeners are created on either on client or on server for executing a piece of code whenever the event is captured. Following code snippet gives a brief on bi-directional communication using event driven model:

Code on client machine:



Socket is initiated by calling connect() method available on global object io. Client is subscribed to ‘created’, ‘full’, ‘join’ and ‘message’ events and performs respective operation on capturing them.

Code on server machine:



Socket connection is created once application is loaded in the browser. Server listens to events ‘message’ and ‘create or join’ and sends corresponding response either by using broadcast (send message to all clients connected) or emit (send response to only that client which requested a message).

# What is WebRTC?

WebRTC stands for Web Real Time Communication. This API is aimed at facilitating plug-in free video and data communication between client machines (most of the time without the interference of server using STUN protocol).

WebRTC makes use of MediaStream API and implements following:

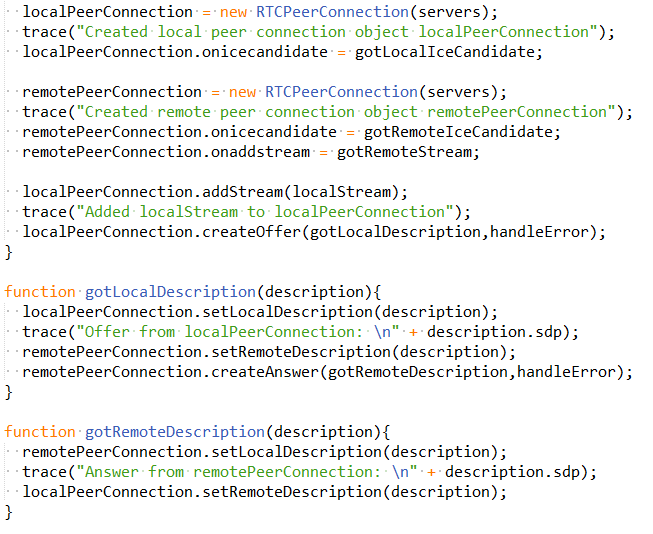
1. Media capture through MediaStream API
2. RTCPeerConnection API for real time media communication between clients with facilities of data encryption and bandwidth management
3. RTCDataChannel API for real time generic data communication between clients

# How to write a simple video stream application using WebRTC?

We will be creating an application which will capture from and stream video to same client machine. This will demonstrate the methodology used in order to establish RTC Peer Connection. Given below are the important steps to be followed in order in to create a communication channel between two peers for streaming video:

1. Create a mechanism for exchanging signaling information between clients. This exchange of information is necessary before a peer communication channel can be established. Using socket.io, this requirement can be achieved easily.
2. Create a media stream by accessing media devices, in our case the web camera, through HTML5 getUserMedia API.
3. On successful capture of media, first client sets its local description of media and sends an offer.
4. This offer is acknowledged by sending an answer by second client with its local description of media attached.
5. Once the local and remote description of media is set on both ends, information on available ICE candidates is shared.
6. Once this information is received, a peer connection is established and actual streaming takes place between clients.

Following code snippet demonstrates implementation of few of the steps mentioned above:



# VIDEO CHAT APPLICATION IN ACTION

We are all set to launch our application now. Sample code base is included in the root folder where this guide is found.

Few guidelines on usage are given below:

# Access to media devices and HTTPS server

Access to media inputs on mobile devices allows users to listen in to the private conversations of the other users. If it were enabled over unencrypted HTTP, would allow an attacker to inject code that listens in and sends the conversations to the attacker. As a result, HTML5’s getUserMedia API allows access to media devices only from secured contexts. Secure contexts include applications running on localhost and https server and not on http server. To create https server for test environment, self-signed certificate can be created by installing openssl and using secure keys on server machine.

In our application, https server is created by using pem node module. Server startup needs following steps to be executed:

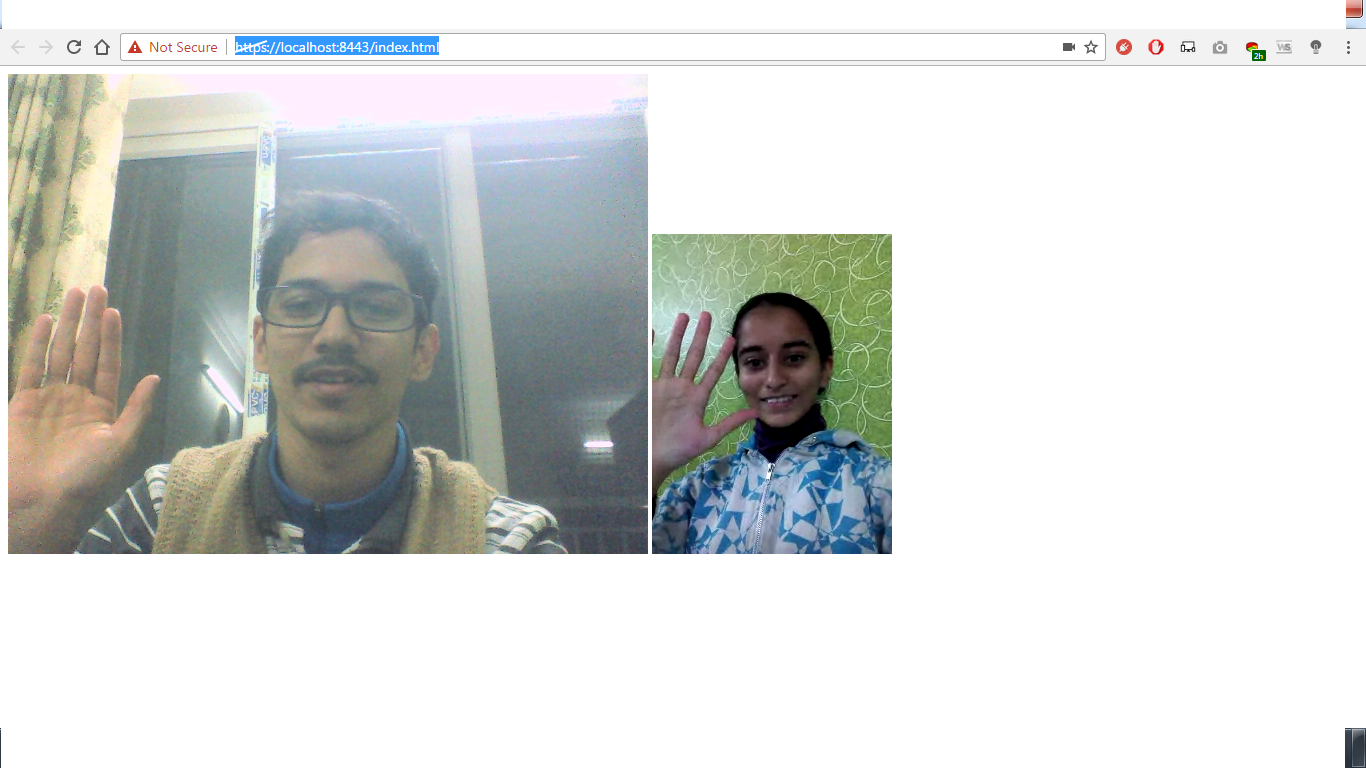
1. Install self-signed certificate on server machine
2. Navigate to root folder of video chat application and run command: npm install
3. Run server by using command: node server.js. Server will start on localhost on port no 8443.

# Join the chat room

Once server is up and running, open chat application on different client machines (preferably using Chrome browser) and voila! Nothing else is required. Since a self-signed certificate is being used for creating https server, browser will treat it as insecure. But for testing purposes and demo application, this error can be safely ignored.

Below are few snapshots from the chat application:

1. Snap taken on desktop



Video on the left is user’s own stream while the one on the right is remote user’s stream

1. Snap taken on android phone:



Due to screen height limitation, only remote user’s snap is captured.