Web real time communication

Busnita Stefan

[https://github.com/stefanBusnita](#_top)

Table of Contents

[I.What is WebRTC ? 2](#_Toc462758302)

[1.1 Signaling 3](#_Toc462758303)

[1.2 RTCPeerConnection 3](#_Toc462758304)

[1.3 STUN 4](#_Toc462758305)

[1.4 TURN 4](#_Toc462758306)

[II. Purpose of the project 4](#_Toc462758307)

[III. Motivation 5](#_Toc462758308)

[IV. Technologies used 5](#_Toc462758309)

[V. Usage 5](#_Toc462758310)

[7](#_Toc462758311)

[VI. Acronyms 10](#_Toc462758312)

[VII. References 10](#_Toc462758313)

# 

# I.What is WebRTC ?

First of all I have to mention that **WebRTC** is a project supported by Google, Mozilla and Opera, amongst others, you can read more about this at the provided link [1]. If you want to read some more on the history of **WebRTC**, be sure to check *Harald Alvestrand’s* post on this topic [2], also a good article is written by *Sam Dutton* on the beginnings of real time web communication and early implementations with all the ups and downs [3].

In order to understand the concept of **WebRTC** first we must understand the peer to peer (also refered to as **P2P**) concept behind this new technology.

Basically peer to peer describes a type of connection from a client device (peer) to another device (other peer ) without the use of servers to send data between the two peers. Actually WebRTC requires some data exchange with some servers, named TURN and STUN, which can take care of Network Address Translators (NAT’s) and firewalls. We will mention these types of servers again when we’ll get into some more detail regarding the way in which connections are handled, created between the peers and destroyed. With the help of WebRTC a browser can directly exchange data with another browser, optimized for low latency, therefore it is a great choice for a range of applications which might include audio/video calling, file sharing between two peers and more. In our case it was used for a lightweight browser chat application, but we’ll get there soon enough.

As I mentioned before there are 2 servers that can take care of some aspects when using WebRTC:

* Clients need to exchange some data between them in order to coordinate communication with each other, sending their local configuration and receiving a remote configuration from the other peer. This process is called signaling.
* Peers have to be traceable and discoverable by each other even if hidden by NAT’s (Network Address Translators) or firewalls, this way communication is ensured between the peers.

## 1.1 Signaling

Signaling can be described as the process in which communication between two peers is established and coordinated by the exchange of small information regarding the nature of the peers. The process of signaling can take care of information exchange such as network data regarding IP addresses, ports, session control for establishing connections or closing them.

Setting up the signaling process is strictly in the hands of the application using **WebRTC**. As mentioned in the **Javascript Session Establishment Protocol** (JSEP), “*The thinking behind WebRTC call setup has been to fully specify and control the media plane, but to leave the signaling plane up to the application as much as possible. The rationale is that different may prefer to use different protocols…*”[5]. We can conclude that the signaling process must be in the hands of app using this technology.

But how is all this information exchanged ? The answer is **SDP** (**Session Description Protocol**). The information exchange works by sending your own description and receiving the other peer description. In other words my local description is set as the remote description of the receiver, and his local description sent towards me is set as the remote description.

## 1.2 RTCPeerConnection

**RTCPeerConnection** is used by **WebRTC** in order to create the connection between peers. It has two main tasks for starting the communication process. First discover the media plane that is going to be used between the peers for communication, and the second task is to find candidates for the connection. After these two tasks are completed, the signaling service comes into action and can start the exchange of data between the two peers.

The following stages are involved in the offer/answear mechanism. I’m going to refer to the two peers involved as caller and calee.

1. The caller creates **RTCPeerConnection** along with an offer (**SDP**) and a local description which is going to be sent by the Caller towards the calee.
2. The calee is going to set his remote description using the caller’s local description, and now is aware of the callers configuration.
3. The calee sends his own local description back to the caller, which in turn is going to set as the remote description of the caller by the caller.

The caller and the calee also have to exchange other information, regarding their own IP’s and ports configuration. This is done using the **Internet Connectivity Establishment** framework (ICE).

**ICE** is a technique used in computer networks used in order to find a way in which two machines can communicate over the network using peer to peer networking. The ICE framework was developed by the Internet Engineering Task Force. [7]

What about an **ICE** candidate in a **WebRTC**, how is he discovered and made known to the application ?

The following is a description from the JSEP draft: *“When a new ICE candidate is available, the ICE Agent will notify the application via a callback; these candidates will automatically be added to the local session description. When all candidates have been gathered, the callback will also be invoked to signal that the gathering process is complete.”*[6]

To summarize on our case, the caller creates an ***onicecandidate*** handler at the **RTCPeerConnection** object creation time. The handler is called when candidates become available. In the handler the caller sends his candidate information to the calee. The calee adds the information to the ice candidates list with the ***addIceCandidate***() method, therefore the remote peer description is set.

ICE candidate trickling is also available, meaning that the caller will incrementally provide candidates to the calee, and the calee will create connections without waiting for all candidates to arrive. As described in the JSEP draft : *“This process allows the callee to begin acting upon the call and setting up the ICE (and perhaps DTLS) connections immediately, without having to wait for the caller to gather all possible candidates. This results in faster call startup in cases where gathering is not performed prior to initating the call.”* [6]

The whole process mentioned above can be tricky when considering the existence of NAT’s and firewalls that is why we need the TURN and STUN servers. A simple explanation would be that the STUN server is used to get an external network address and the TURN server is used to pass on traffic if peer to peer connection fails. [3]

## 1.3 STUN

The WebRTC application uses the **STUN** server to discover IP’s and port’s of peers that are hidden behind a **NAT or firewall**. A peer can get a publicly accessible address ( accessible from the outside world ) for itself and create a link with another peer by passing this information. After establishing a public IP and port for the peer, the communication will continue with that peer using that **STUN** given address.

## 1.4 TURN

**TURN** servers are used to pass on streams of data between peers, not signaling data. The RTCPeerConnection starts by using **UDP** as a communication mean, if that fails it resorts to **TCP**, and if that doesn’t work **TURN** servers come into action. A **TURN** server provides its own public accessible address therefore proxies and firewalls don’t interfere in the process of peer communication via a **TURN** server.

# II. Purpose of the project

Implementing a lightweight standalone desktop-like web chat application, as a personal project. Recalling some DOM manipulation technologies like Jquery was a side objective. But why did I say desktop-like ? The vast majority of web chat apps don’t have the possibility to chat with multiple peers at once, or to minimize a chat window when the situation gets cluttered (before you jump to conclusions, I said the vast majority, not all other chat apps). Therefore implementing a standalone system that could do that turned out to be a fun personal project. The app is used at the company I currently work at, as a backup chat app for us (the development team).

# III. Motivation

Started to have a growing interest in web real time communication. I started reading different articles and I chose the best suited technologies for the job ( in my opinion ). My idea included some heavy DOM manipulation, a lightweight web server, easy to write and deploy, and also a plugin for the bidirectional real-time event based communication.

# IV. Technologies used

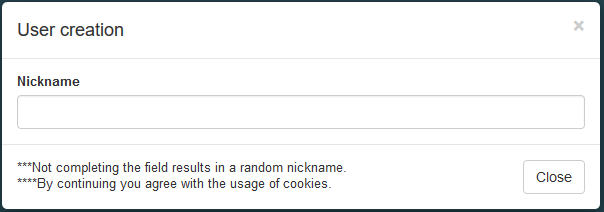
Heavy **DOM** manipulation like chat window creation, dragable chat windows, dynamic creation required a good **DOM** manipulation framework. Considering all this **JQUERY** turned out the perfect choice. The real-time communication is event drive and **socket.io** was the best choice. Due to the fact that **node.js** is used (socket.io and node.js go like peanut butter and jelly) a lightweight server was needed and **express.js** provided all the necessary requirements.

Using node led to the need to have all my **required(‘module’)** bundled up in a single script and sent as needed, and **browserify** came to the rescue.

Creating chat rooms require passwords to be kept on the server and encryption was necessary, this is where **cryptojs** helped, encryption for room passwords was done using AES and salt.

# V. Usage

On accessing the app, the user is prompted with a modal dialog in which the username must be completed:

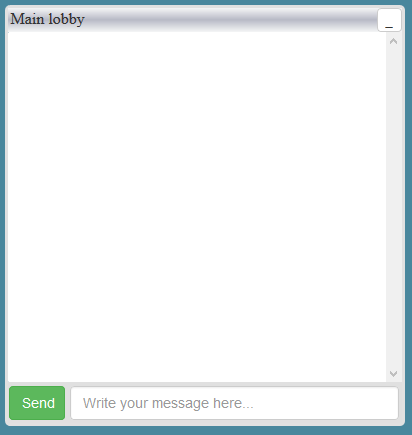


By continuing the user also accepts the usage of cookies, used for storing user preferences for the app.

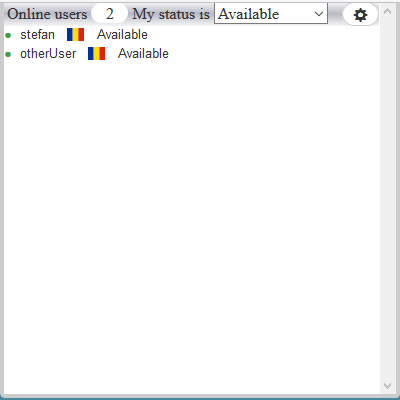
If the user closes the modal and no nickname is entered a random nickname is created. C:\Users\stefan.busnita\Desktop\SnipsForChatDocs\RandomNickname.PNG

On entering the actual nickname : C:\Users\stefan.busnita\Desktop\SnipsForChatDocs\ActualNickname.PNG

The first thing that the user sees is the main lobby. All users automatically join the main lobby. All messages written in the main lobby can be seen by all other users.

All chat windows created in the program will look similar to this one. A similarity between desktop windows and app windows can be seen here, namely in the shape and the window action area, the top right corner. There are 2 available actions : minimize and close (close not available in main lobby).

Upon minimization, a button with the window name will appear in the left bottom corner of the screen :

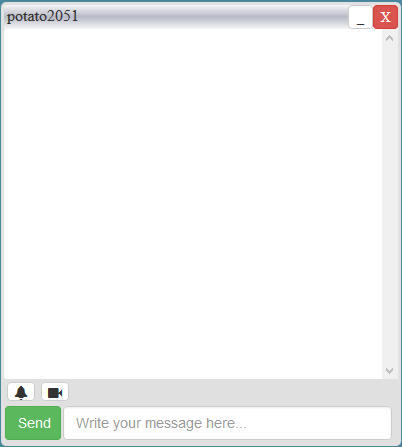


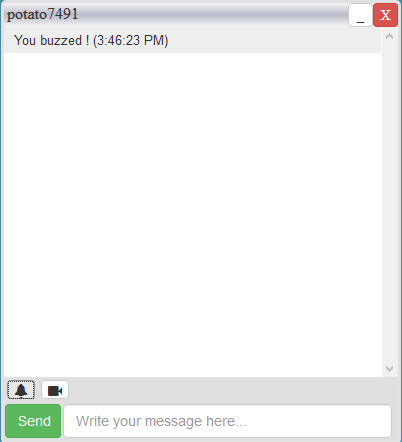
On the right side of the screen the list with all online users is available. You can change your status from the available statuses dropdown. A user is described by his/her name, the country flag, and the status. A green sign appears next to each user if his/her current status involves availability to chat. If not a red dot appears signifying that the user is busy, and can’t chat. Also there is an option to activate/deactivate buzz sound, and other options. A window for settings is available by pressing the cog-wheel icon button.

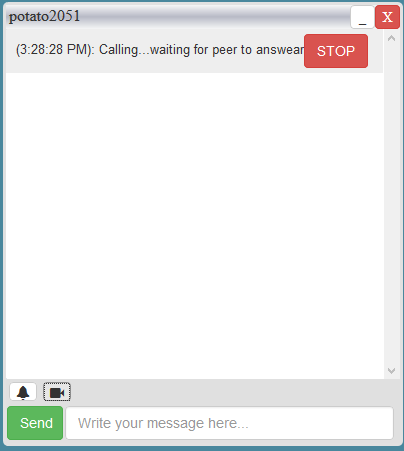
A chat screen with a user can be opened in 2 way:

* A message is received from a user, therefore a new window is created by the program and a chat can start.
* You can engage into a conversation by clicking the nickname of the user you wish to chat with.

The chat window might look like this:

As was discussed earlier, the minimize and close buttons are available in the top right corner. Two actions are also available for the particular chat window with a user, namely buzzing the user or video/audio calling the user.

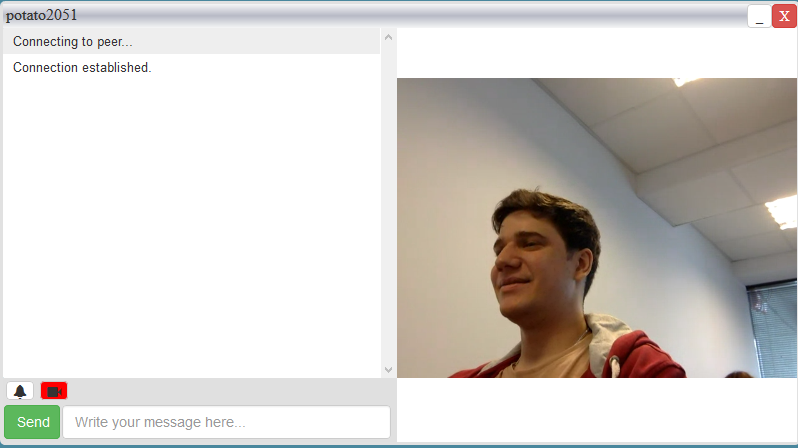
On buzzing the user :



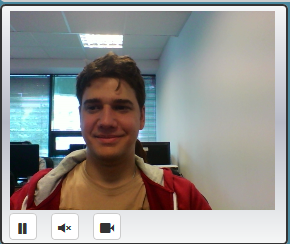
On calling the user:

# C:\Users\stefan.busnita\Desktop\SnipsForChatDocs\ToAnswear.PNG

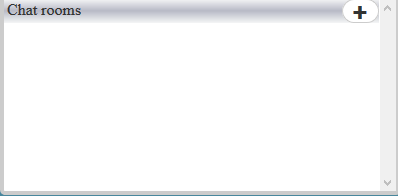
On the other end :

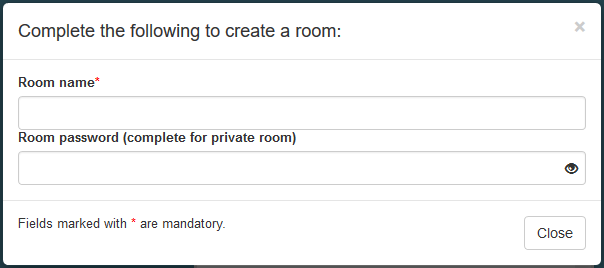
When video / audio connection is established :   


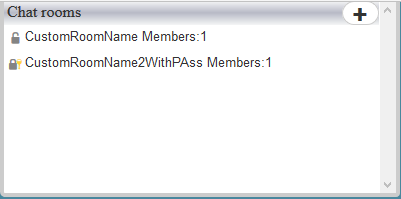
Your own video is available in the left part of the screen under room creation. Some controls for video/audio are also available.



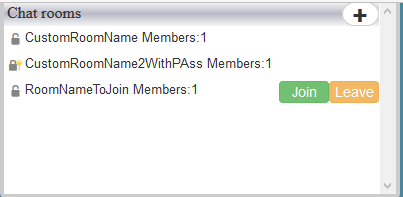
The left part of the screen also has a room creation section, with a list containing all the rooms. A room can be private or public. A private room has a certain password, and is signaled in the list with a special icon containing a key and a lock. Public rooms have a similar icon, with an open lock.



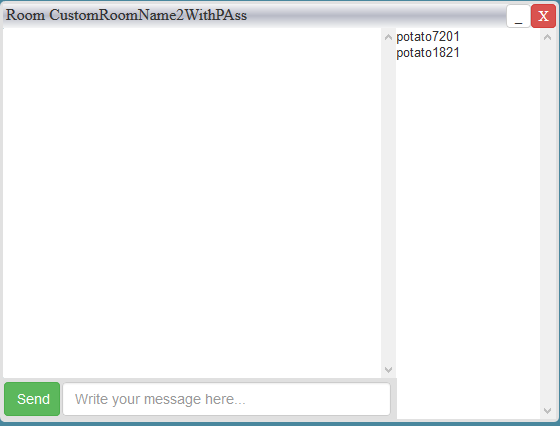




In order to join a room or leave a room, simply click a room name and 2 options will appear, obviously JOIN and LEAVE.



A room window contains a list with all users, and similar functionality to other windows.



# VI. Acronyms

SDP – Session Description Protocol

WebRTC – Web real time communication

UDP – User Datagram Protocol

TCP – Transmission Control Protocol

TURN – Traversal Using Relays around NAT

STUN – Session Traversal Utilities for NAT

ICE –Internet Connectivity Establishment

JSEP - Javascript Session Establishment Protocol

DOM – Document Object Model

HTTP - Hypertext Transfer Protocol

AES – Advanced Encryption Standard

# VII. References

[1]. <https://webrtc.org/>

[2]. [https://groups.google.com/forum/#!topic/discuss-webrtc/I0GqzwfKJfQ](#_VI._References)

[3]. [http://www.html5rocks.com/en/tutorials/webrtc/basics/](#_VI._References)

[4]. [http://www.html5rocks.com/en/tutorials/webrtc/infrastructure/](#_VI._References)

[5]. [https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03#section-1.1](#_VII._References)

[6]. [https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-03#section-3.4.1](#_VI._References)

[7]. [https://en.wikipedia.org/wiki/Interactive\_Connectivity\_Establishment](#_VI._References)