WEBRTC mini Application for video and audio call

**ABSTRACT**:

In this project, an app is developed for real time audio and video communication using WebRTC API. Firstly, two different browsers obtain the IP addresses of each other through the process called signaling i.e for the peer to peer connection. The web browsers register to backend server running at 8000 port and then they register to WEBRTC API using API like MediaStream, RTCPeerConnection and RTCDataChannel.

**WebRTC API:**

It is open source API which provides browsers and mobile applications to have Real Time Communications by using API’s. Applications based on WebRTC get network information like IP addresses and ports and prefer to exchange this all information with the other client which is also using the WebRTC application to make peer to peer connection through NAT, STUN or TURN and various firewalls. Moreover, it coordinates communication and report errors, initiate request or close sessions to the client at proper time without any delay. Thus, all streaming audio, video or data like media information, client capability like resolution and codecs are also shared through peer to peer connection using WebRTC.

**Peer-To-Peer Communication:**

To communicate with another person, each person’s web browser must have to be agreeing to begin communication. They must know how to locate one another,“bypass security and firewall protections and transmit all multimedia communications in real”time.

Here, one of the biggest challenges to have bidirectional transmit multimedia data is“how to locate and establish a network socket connection with another client’s browser. This can be handled by making an HTTP request to a known address and easily locatable DNS server to get back response”from another client.

But, if one of the client is not a web server then in this case WebRTC plays significant role in sending audio and video data directly to another client without going through any external server.

**Firewall and NAT Traversal:**

There are Firewalls and NAT in local networks, due to this reason they are not assigned a public IP address to communicate with another client for the valid communication. Behind the node, inside a firewall private IP addresses are given to the computer and NAT translates that IP addresses to public IP addresses for communication purposes. Suppose you are at university and you want to connect your laptop to university’s wifi, here behind the NAT some private addresses are assigned to your laptop which is not known to another clients available in the world. The connections with the outside world is made with the public IP addresses of the computer or laptop. Only NAT knows that the request and responses are coming from which private IP addresses located behind the NAT. Two clients know the public IP addresses with the help of STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT) servers. In WebRTC applications, both the clients firstly know their public IP addresses through STUN/TURN servers. If the IP addresses are not found using STUN server then TURN server initiates requests to find the actual public IP addresses to make peer to peer connection for secure audio and video communication.

**Different APIs of WebRTC:**

Communication with two different clients is done using APIs described below:

1. MediaStream : The main task of this API is to get access to data streams like from the client’s camera and microphone.
2. RTCPeerConnection: “It handles audio or video calling, with facilities like encryption and bandwidth management.”
3. RTCDataChannel: It mainly handles communication between two clients for the proper exchange of information between the two different clients.

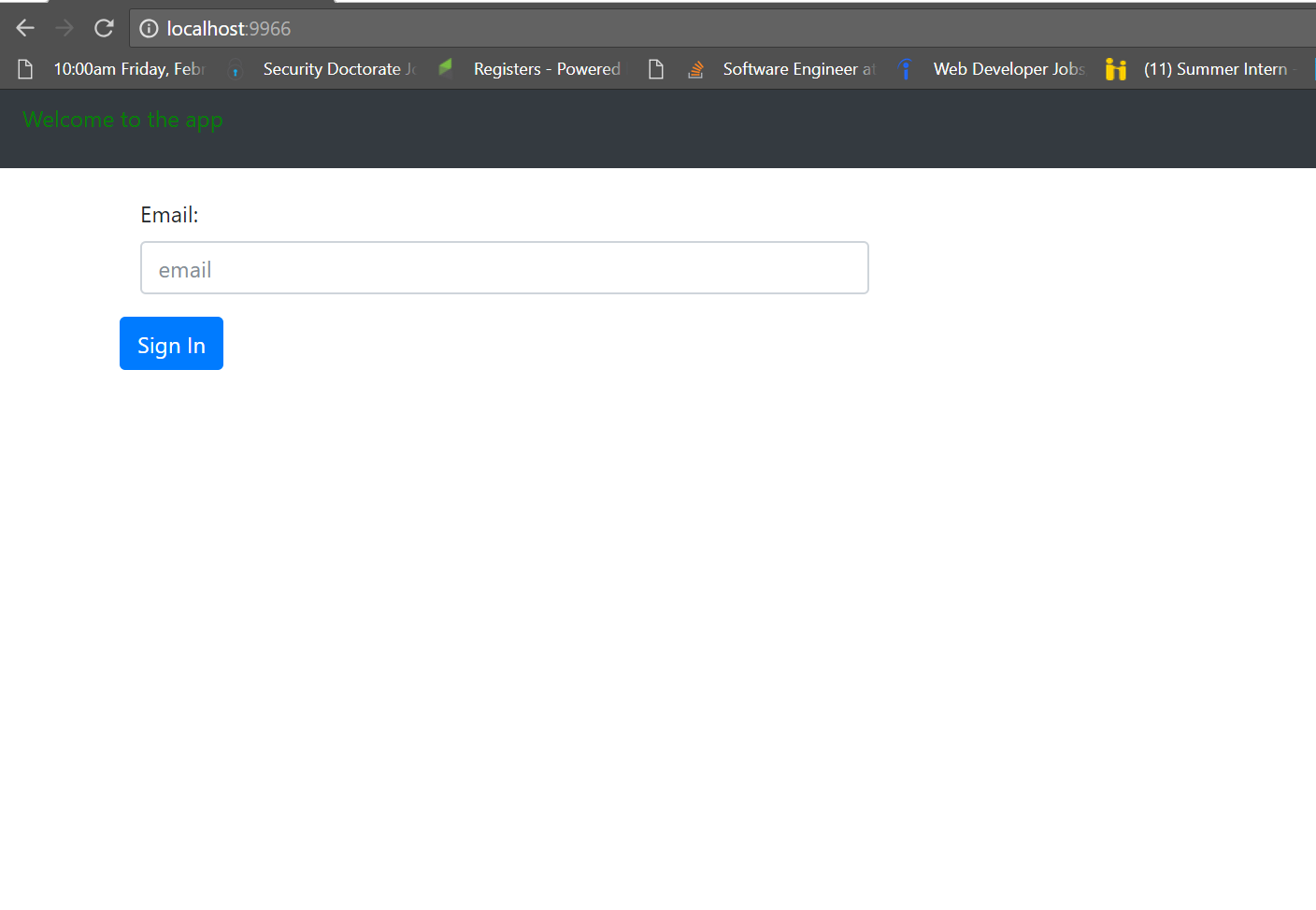
**Application Architecture-**

The complete source code of project can be found at-

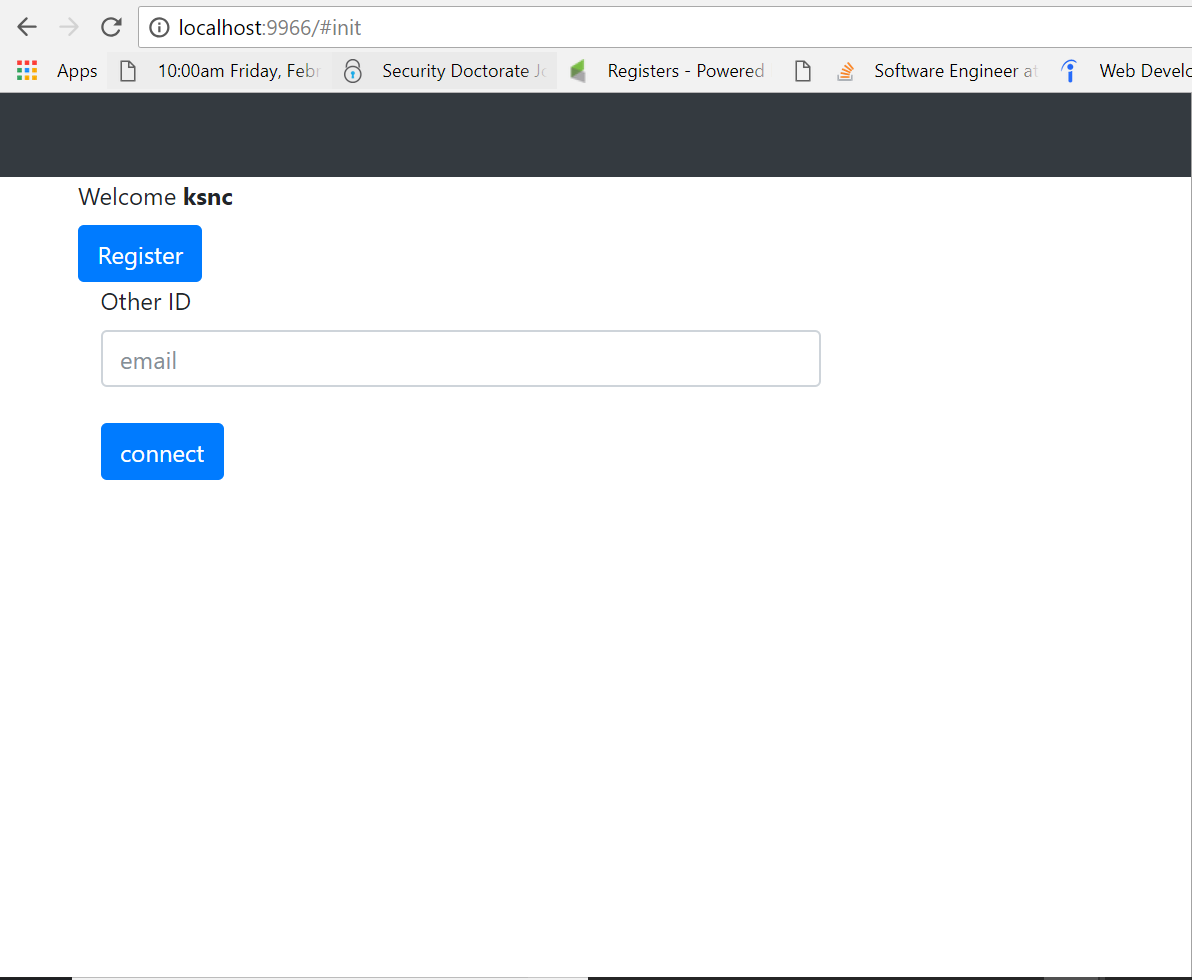
<https://github.com/komaljit/WebRTC-video-app>

1. **User sign in-**

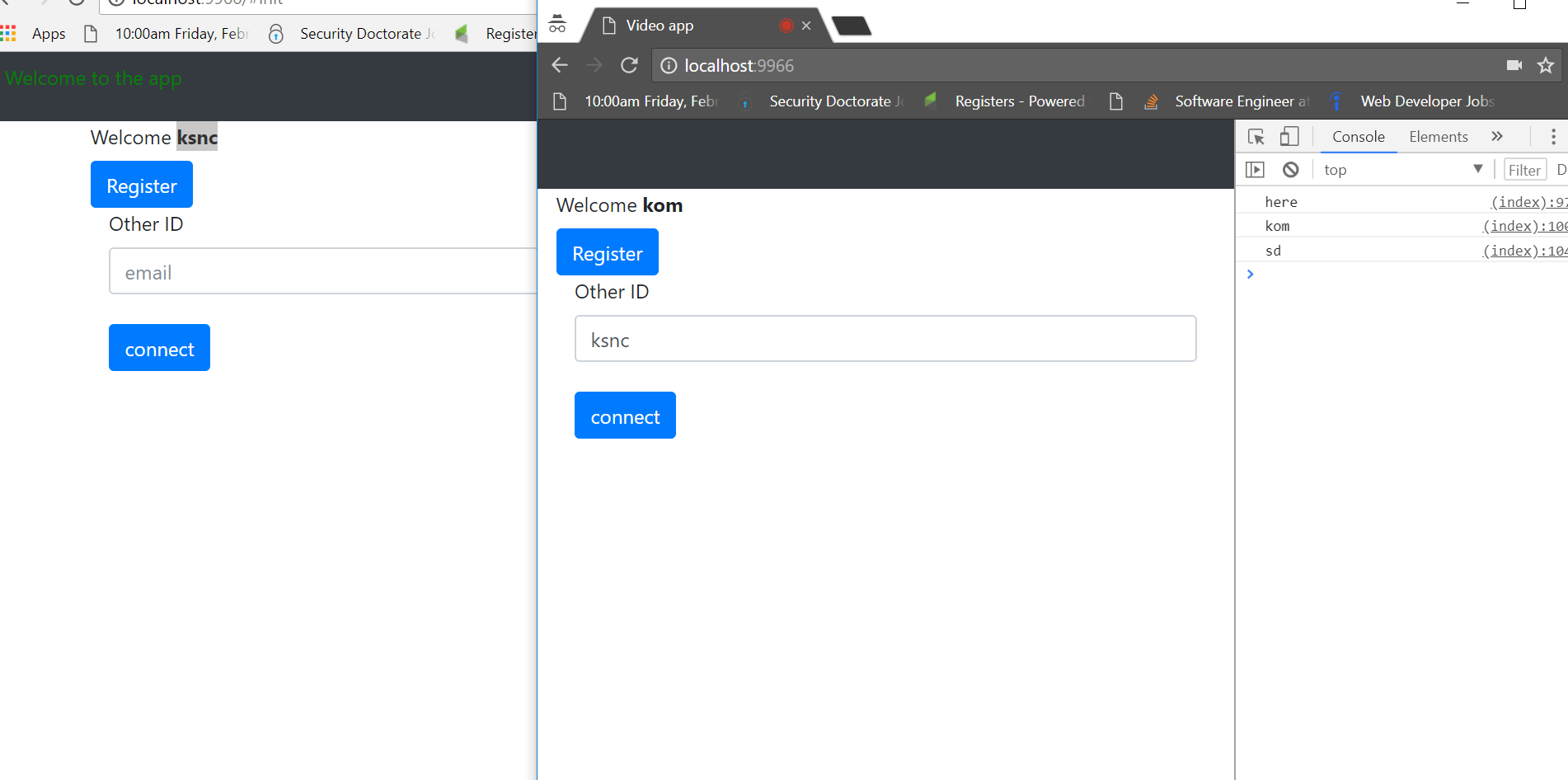
When a user first time opens the web app, user see this page. He will provide a unique email to sign in, which will uniquely identify him.



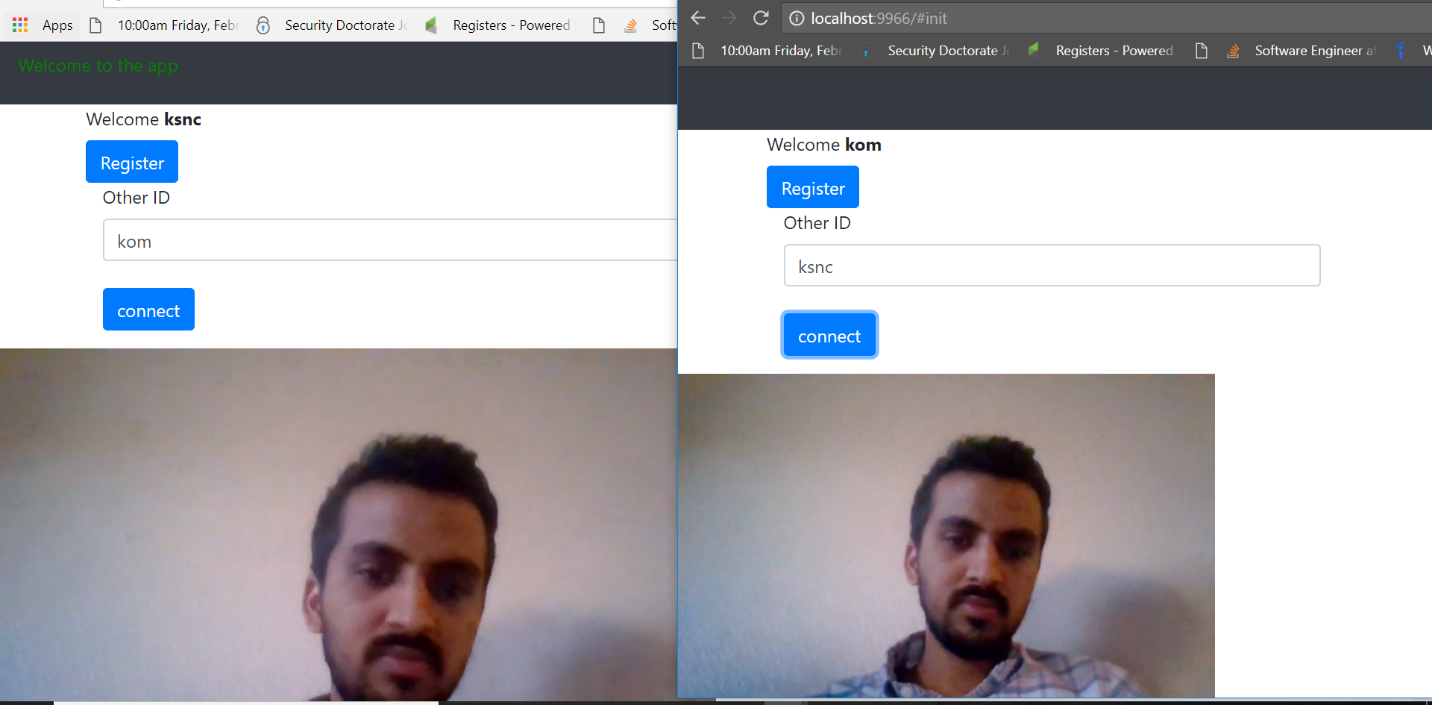
User registerations with WEBRTC API- After signing up, the user will register with web rtc api to get ID.



In the following screen, the two users- KOM, KOMS who are registered, will call each other by clicking connect button and they also have t provide the ID of the other person (Note- that ID is not which WEBRTC provides the user, it is the email which user gets from our application)



**Calling each other-** In the following screen shot, the two users are calling each other from different windows.



Our applications are running on two servers-

Localhost:800 Django server with which users register with email

Localhost:9966 Node server which is acting as front-end server for registering with

WEBRTC.

**Some core parts of Source code-**

*// api call to django aerver***var** API = **'http://127.0.0.1:8000'**;  
  
**function** *doRegister*(payload){  
 fetch(API+**'/register/'**, {  
 **method**: **'POST'**,  
 **headers**: {  
 **'Content-Type'**: **'application/json'** },  
 **body**: ***JSON***.stringify(payload)  
 }).then(**function**(res){  
 ***console***.log(**"inside api file"**);  
 **return** res;  
 });  
}  
  
**function** *getPeer*(payload, callback){  
 fetch(API+**'/getpeer/'**, {  
 **method**: **'POST'**,  
 **headers**: {  
 **'Content-Type'**: **'application/json'** },  
 **body**: ***JSON***.stringify(payload)  
 }).then(**function**(res){  
 ***console***.log(**"inside getpeer api"**);  
 **return** res.json();  
 }).then(**function**(res){  
 ***console***.log(**"inside api file"**);  
 callback(res);  
 });  
}  
  
exports.*getPeer* = *getPeer*;  
exports.*doRegister* = *doRegister*;

Javascript file for streaming video and audio-

**var** userMedia = *require*(**'getusermedia'**);  
**var** API = *require*(**'./api'**);  
  
  
userMedia({ **video**: **true**, **audio**: **true** }, **function** (err, stream) {  
 **if** (err) **return *console***.error(err);  
  
 **var** Peer = *require*(**'simple-peer'**);  
 **var** peer = **new** Peer({  
 **initiator**: **location**.**hash** === **'#init'**,  
 **trickle**: **false**,  
 **stream**: stream  
 });  
  
 peer.on(**'signal'**, **function** (data) {  
 ***console***.log(**'here'**);  
 **var** payload = {};  
 **localStorage**.setItem(**"yourId"**, data);  
 payload [**'rtc\_id'**] = ***JSON***.stringify(data);  
 payload [**'email'**] = **localStorage**.getItem(**'user'**);  
 ***console***.log(payload.**email**);  
 API.*doRegister*(payload)  
 *// document.getElementById('yourId').value = JSON.stringify(data);* });  
  
 **document**.getElementById(**'connect'**).addEventListener(**'click'**, **function** () {  
 **var** payload = {};  
 payload[**'email'**] = **document**.getElementById(**'otherId'**).**value**;  
 API.*getPeer*(payload,**function**(res){  
 ***console***.log(**"inside api file"**);  
 ***console***.log(res);  
 *// var otherId = JSON.parse(document.getElementById('otherId').value);* **var** otherId = ***JSON***.parse(res.**rtc\_id**);  
 peer.signal(otherId);  
 });  
 });  
  
  
 peer.on(**'stream'**, **function** (stream) {  
 **var** video = **document**.createElement(**'video'**);  
 **document**.**body**.appendChild(video);  
  
 video.**src** = **window**.URL.createObjectURL(stream);  
 video.play()  
 })  
});

**from** rest\_framework.views **import** APIView  
**from** rest\_framework.generics **import** RetrieveAPIView  
**from** django.shortcuts **import** get\_object\_or\_404  
**from** .serializer **import** RegisterSerializer, GetPeerSerializer  
**from** rest\_framework.response **import** Response  
**from** .models **import** Peer  
  
  
**class RegisterApi**(APIView):  
 # authentication\_classes = (SessionAuthentication,)  
 serializer\_class = RegisterSerializer  
  
 **def post**(self, request):  
 **return** self.create(request)  
  
 **def create**(self, request):  
 print(request.data)  
 print("here")  
 serializer = self.serializer\_class(data=request.data)  
 print(serializer.is\_valid())  
 **if** serializer.is\_valid(raise\_exception=Response(status=400)):  
 serializer.save()  
 **return** Response(serializer.data,status=204)  
 **else**:  
 **return** Response(status=400)  
  
  
**class GetPeerId**(APIView):  
 # authentication\_classes = (SessionAuthentication,)  
 serializer\_class = GetPeerSerializer  
  
 **def post**(self, request):  
 print(self.request.data)  
 **return** self.retrieve(request)  
  
 **def retrieve**(self, request):  
 instance = self.get\_object()  
 serializer = self.serializer\_class  
 serializer\_data = serializer(instance)  
 print(serializer\_data.data)  
 **return** Response(serializer\_data.data)  
  
 **def get\_object**(self):  
 **return** get\_object\_or\_404(Peer, email=self.request.data.get('email'))

**References-**

1. <https://webrtc.org/>
2. <https://github.com/shama/letswritecode/blob/master/p2p-video-chat-webrtc>
3. <https://en.wikipedia.org/wiki/WebRTC>