❊ فریم ورک مورد استفاده در سرور:

در این پروژه از فریم ورک python flask و افزونه flask_sse برای هندل کردن ریکوئست های http و sse استفاده شده.

❊ توضیح کد:

در ابتدا با وارد کردن آدرس server_ip:listening_port صفحه ی login.html توسط سرور برگردانده می شود.

در این صفحه دو اینپوت موجود است. نفر اول با پر کردن فیلد creat chat به صفحه ی wait.html منتقل شده و بر روی ایونت سورسی با آدرس server_ip:listening_port/stream?channel={channel_name} شروع به گوش دادن به ایونت های سرور میکند که در آن channel_name همان مقداری است که در صفحه ی login برای فیلد creat chat پر کرده است (این مقدار توسط ریم ورک flask و به هنگام رندر کردن template به آن پاس داده می‌شود).

```
var source = new EventSource("/stream?channel="+"{{name}}");
```

سپس نفر دوم با پر کردن فیلد join chat و قرار دادن مقداری که نفر اول در creat chat قرار داده بود می تواند با نفر اول ارتباط برقرار کند.

فرایند به این صورت است که پس از زدن join chat فرد دوم به صفحه ی join.html منتقل میشود و او نیز بر روی همان ایونت سورس قبلی به گوش دادن به پیام های سرور می پردازد.

قبل از هر چیز در هر دو صفحه ابتدا مدیای خود فرد گرفته شده و در صورت موفقیت آمیز بودن آن، تابع getUserMediaSuccess فراخوانی می‌شود که در آن فیلد ویدئوی مربوط به خود شخص پر شده و این استریم به RtcConnection نیز افزوده می‌شود. همچنین تابع gotRemoteStream نیز به عنوان call back برای ایونت onaddstream ست می‌شود.

```
var constraints = {
    video: true
};

if(navigator.mediaDevices.getUserMedia) {
    navigator.mediaDevices.getUserMedia(constraints).then(getUserMediaSuccess).catch(errorHandler);
} else {
    alert('Your browser does not support getUserMedia API');
}


function getUserMediaSuccess(stream) {
    console.log("getUserMediaSuccess");
    localStream = stream;
    localVideo.src = window.URL.createObjectURL(stream);
    peerConnection.addStream(localStream);
```

برای نفر دوم در این تابع علاوه بر موارد ذکر شده اقدام به ایجاد sdp offer هم می‌شود:

```
var constraints = {
    video: true
};

if(navigator.mediaDevices.getUserMedia) {
    navigator.mediaDevices.getUserMedia(constraints).then(getUserMediaSuccess).catch(errorHandler);
} else {
    alert('Your browser does not support getUserMedia API');
}

function getUserMediaSuccess(stream) {
    localVideo.src = window.URL.createObjectURL(stream);
    peerConnection.onaddstream = gotRemoteStream;
    peerConnection.addStream(stream);
    peerConnection.createOffer().then(createdDescription).catch(errorHandler);

}
```

پس از ایجاد sdp offer تابع createdDescription فراخوانی میشود که در آن نفر دوم پیامی از نوع request که حاوی sdp offer میباشد بر روی کانالی که هر دو نفر به آن گوش می دهند توسط تابع send_message می‌فرستد. نحوه ی کار تابع send_message جلوتر توضیح داده شده.

```
function createdDescription(description) {
    console.log('got description');
    peerConnection.setLocalDescription(description).then(function() {
        send_massage(JSON.stringify({'sdp': peerConnection.localDescription}) , "{{name}}", 'request');
    }).catch(errorHandler);
}
```

نفر اول با دریافت پیامی از نوع request اقدام به ست کردن remote description خود با محتوای sdp پیام دریافتی و فرستادن پیامی از نوع response میکند که حاوی sdp answer میباشد.

```
source.addEventListener('request', gotRequest);

function gotRequest(event) {
    console.log("gotRequest");
    var data = JSON.parse(event.data);
    data =  JSON.parse(data.message);
    peerConnection.setRemoteDescription(new RTCSessionDescription(data.sdp)).then(function() {
        peerConnection.createAnswer().then(createdDescription).catch(errorHandler);
    }).catch(errorHandler);

}
```

سرانجام نفر دوم با دریافت پیامی از نوع response اقدام به ست کردن remote description خود میکند:

```
source.addEventListener('response', gotResponse);

function gotResponse(event) {
    console.log("gotResponse");
    var data = JSON.parse(event.data);
    data =  JSON.parse(data.message);
    peerConnection.setRemoteDescription(new RTCSessionDescription(data.sdp));
        }
```

سپس در هر دو صفحه تابع gotRemoteStream در اثر ایونت onaddstream فراخوانی می‌شود که منجر به پر شدن تگ ویدئوی مربوط به شخص مقابل با استریم دریافتی می‌گردد:

```
function gotRemoteStream(event) {
    console.log('gotRemoteStream');
    remoteVideo.src = window.URL.createObjectURL(event.stream);
}
```

باقی توابع موجود و ایونت لیسنر ها مربوط به ice agent می‌باشند. پیام هایی از نوع ice1 توسط نفر اول و به هنگام یافتن کاندیدا ها توسط ice agent به نفر دوم ارسال میشود و پیام های ice2 توسط نفر دوم به نفر اول ارسال میگردد.

اکنون به شرح تابع send_message که در نقش کانال سیگنالینگ ما عمل می‌کند می‌پردازیم.

❊ نحوه کار **send_message**:

این تابع ۳ ورودی میگیرد. ابتدا محتوای پیام ارسالی، سپس کانالی که سرور سیگنالینگ می‌بایست این پیام را روی آن کانال sse منتشر کند و سرانجام نوع پیام ارسالی (نوع ایونت).

همانطور که در بدنه‌ی این تابع مشخص است، با فراخوانی شدن آن یک ریکوئست xhr از نوع POST و به آدرس server_ip:listening_port/channel/send/{channel}/{type} فرستاده می‌شود.

```
<script>
    function send_massage(massage, id, type) {
        var xhr = new XMLHttpRequest();
        var inp = "/channel/send/"+id+"/"+type;
        xhr.open('POST', inp, true);
    // xhr.onload = function () {
    //     if (xhr.readyState === xhr.DONE) {
    //         if (xhr.status === 200) {
    //             console.log(xhr.response);
    //             console.log(xhr.responseText);
    //         }
    //     }
    // };
        xhr.send(massage);
    }
</script>
```

در سمت سرور از افزونه‌ی flask_sse استفاده شده که به سادگی محتوای ریکوئست دریافتی را روی کانال sse مورد نظر ارسال می‌کند:

```
from flask import Flask
from flask import Blueprint
from flask import request
from flask_sse import sse

channel = Blueprint('channel', __name__)


@channel.route("/send/<channel_name>/<massage_type>", methods=['POST'])
def send(channel_name, massage_type):
    print "Hello channel send!"
    sse.publish({"message": request.data}, type=massage_type, channel=channel_name)
    return "send", massage_type, "to", channel_name
```

❊ نحوه اجرای سرور و نیازمندی ها:

همانطور که اشاره شد، در این پروژه از فریم ورک python flask استفاده شده. پس در ابتدا با دستور

```
$sudo pip install Flask
```

این فریم ورک زبان پایتون را نصب می‌کنیم.

برای نصب افزونه flask_sse:

```
$sudo pip install sse
$sudo pip install flask_sse
```

همچنین این افزونه از دیتابیس Redis استفاده میکند:

```
$wget http://download.redis.io/redis-stable.tar.gz
$tar xvzf redis-stable.tar.gz
$cd redis-stable
$make install
```

علاوه بر موارد ذکر شده برای اجرای اسکریپت پایتون میبایست از gunicorn استفاده نمود (چرا که thread مربوط به app فلسک از thread مربوط به sse جداست):

```
$sudo apt-get install gunicorn
```

ابتدا میبایست از در حال اجرا بودن سرور redis اطمینان حاصل کرد:

```
$services redis-server start
```

سرانجام برای اجرای سرور سیگنالینگ از دستور زیر در دایرکتوری مربوط به فایل main.py استفاده میکنیم:

```
$gunicorn main:app --worker-class gevent --bind 0.0.0.0:8000
```

❊ نتیجه اجرا:

(سعی کردم دوتا لپتاپ رو بگیرم جلوی هم کیفیتش بد شد ☺)