TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN III

ĐỀ TÀI: SỬ DỤNG NODEJS VÀ WEBRTC ĐỂ TẠO LẬP PHẦN MỀM GỌI VIDEO CALL QUA INTERNET.

Giáo viên hướng dẫn: Nguyễn Tuấn Dũng
Sinh viên: Nguyễn Quốc Khoa
Lớp: INPG-15
MSSV: 20168278

Hà Nội, 28/1/2021

Mục Lục

1. Lời mở đầu

Ngày nay Internet đã trở thành một trong những dịch vụ phổ biến và thiết yếu, có ảnh hưởng lớn tới đời sống của con người. Một trong những nhu cầu đó chính là việc có thể liên lạc với nhau từ các khoảng cách xa. Chính vì thế, sau khi tham khảo với thầy giáo khi được chọn để làm đồ án về công nghệ NodeJS, em đã được thầy giáo giao cho mục tiêu tạo ra một trang web với khả năng gọi video thông qua công nghệ webRTC (và được thầy giáo khuyên dùng sử dụng API SipJS). Sau khi tham khảo qua các công nghệ tương tự, em thấy rằng PeerJS không chỉ thực hiện vai trò y hệt mà còn đơn giản hơn cho mục tiêu của mình là:

- Tạo một server phục vụ cho nhu cầu gọi video thông qua công nghệ NodeJS.
- Hiểu cách hoạt động của WebRTC và cách các API hoạt động với nhau.

Với chủ đề và mục tiêu của đồ án này, em xin chân thành cảm ơn thầy giáo Nguyễn Tuấn Dũng đã chỉ bảo và giúp đỡ em trong quá trình tìm hiểu và làm bài tập. Rất mong được thầy và các bạn đóng góp ý kiến để phần mềm ngày càng được hoàn thiện và đưa vào sử dụng.

2. Mục tiêu của đồ án

Sau khi tham khảo với thầy giáo khi được chọn để làm đồ án về công nghệ NodeJS, em đã được thầy giáo giao cho mục tiêu tạo ra một trang web với khả năng gọi video thông qua công nghệ webRTC (và được thầy giáo khuyên dùng sử dụng API SipJS). Sau khi tham khảo qua các công nghệ tương tự, em thấy rằng PeerJS không chỉ thực hiện vai trò y hệt mà còn đơn giản hơn cho mục tiêu của mình là:

+ Tạo một server phục vụ cho nhu cầu gọi video thông qua công nghệ NodeJS.
+ Hiểu cách hoạt động của WebRTC và cách các API hoạt động với nhau.
+ Tạo ra một website đơn giản nhưng dễ dùng, với các phòng riêng biệt và có các id động nhằm có thể phân biệt và chia phòng tùy theo người sử dụng.

Đối tượng sử dụng Website:

+ Những người sử dụng website là những cá nhân có nhu cầu liên lạc thông qua internet và sử dụng video call một cách an toàn và bảo mật.

Đặc điểm:

+ Hệ thống cần đạt được yêu cầu về việc sử dụng các api nhằm có thể kết nối qua WebRTC và tạo dựng các video call giữa những người dùng. Theo yêu cầu của thầy giáo, chức năng được ưu tiên thay vì ngoại hình của trang web để học sinh có thể hiểu được chức năng của các công nghệ được chọn lựa.

3. Các công nghệ cần thiết

- NodeJS:

NodeJS là một nền tảng được xây dựng trên V8 JavaScript Engine – trình thông dịch thực thi mã JavaScript, giúp xây dựng các ứng dụng web một cách đơn giản và dễ dàng mở rộng. NodeJS được phát triển bởi Ryan Dahl vào năm 2009 và có thể chạy trên nhiều hệ điều hành khác nhau: OS X, Microsoft Windows, Linux.
Một số ưu điểm của NodeJS là:

+ Nodejs là một nền tảng (Platform) phát triển độc lập được xây dựng ở trên Javascript Runtime của Chrome mà chúng ta có thể xây dựng được các ứng dụng mạng một cách nhanh chóng và dễ dàng mở rộng.

+ Phần Core bên dưới của Nodejs được viết hầu hết bằng C++ nên cho tốc độ xử lý và hiệu năng khá cao.

+ Nodejs tạo ra được các ứng dụng có tốc độ xử lý nhanh, realtime thời gian thực.

+ Nodejs áp dụng cho các sản phẩm có lượng truy cập lớn, cần mở rộng nhanh, cần đổi mới công nghệ, hoặc tạo ra các dự án Startup nhanh nhất có thể.

- WebRTC:

WebRTC là các API viết bằng javascript giúp giao tiếp theo thời gian thực mà không cần cài plugin hay phần mềm hỗ trợ. WebRTC có khả năng hỗ trợ trình duyệt giao tiếp thời gian thực thông qua Video Call, Voice Call hay transfer data P2P(peer-to-peer), không cần đến plugin, phần mềm khác. Tính năng nổi bật nhất của WebRTC chắc chắn chính là khả năng truyền tải video, âm thanh, gửi dữ liệu theo thời gian thực giữa hai hay nhiều thiết bị mà không qua trung gian, không cần cài thêm plugin. Ưu điểm của WebRTC:

+ Mã nguồn mở miễn phí: WebRTC là một dự án mã nguồn mở miễn phí. Google cho biết đây là một công cụ truyền thông thời gian thực hoàn toàn miễn phí và có sẵn trên mọi trình duyệt.
+ Hỗ trợ đa nền tảng: Mặc dù WebRTC vẫn trong giai đoạn phát triển nhưng nó đã hoạt động tốt trên hầu hết mọi trình duyệt của các hệ điều hành bất kì. Cho phép lập trình viên viết các đoạn mã HTML làm việc với máy tính hoặc thiết bị di động.
+ Bảo mật voice và video: Giao thức SRTP (Secure Real-Time Transport Protocol) được dùng để mã hóa và xác thực dữ liệu media. Chống lại các khả năng bị nghe trộm trong quá trình thực hiện tác vụ video hay voice.
+ Không cần plugin hay phần mềm hỗ trợ: Yếu tố quan trọng giúp WebRTC được đánh giá rất cao chính là khả năng hoạt động không cần đến plugin bên thứ ba mang đến sự tiện lợi, tối ưu tốc độ, tiết kiệm chi phí,…
+ Tương đối dễ sử dụng: WebRTC có thể được tích hợp trong các dịch vụ web bằng cách dùng JavaScript APIs, các Framework có sẵn.

+ Sử dụng bằng thông hiệu quả: Hỗ trợ nhiều kiểu media và các thiết bị đầu cuối khác nhau, WebRTC sử dụng băng thông hiệu quả hơn, hoạt động tốt trong mọi điều kiện đường truyền mạng.

- PeerJS:

PeerJS là một thư viện JavaScript hoạt động như một trình bao bọc cho WebRTC và cho phép bạn tạo các kết nối ngang hàng rất dễ dàng. Với PeerJS, ta có thể tạo kết nối như vậy chỉ trong 3 dòng mã. PeerJS giao dịch với bắt tay WebRTC và cho phép kết nối bằng ID ngang hàng. Tuy nhiên, để thiết lập kết nối, bạn cần có PeerServer. Điều này được viết bằng Node.js và ta cần thiết lập nó. Không có dữ liệu đi qua máy chủ - nó chỉ được sử dụng để môi giới kết nối giữa các client.

- UUID:

UUID (là viết tắt của Universally Unique IDentifier) là một api của Java nhằm có thể gắn được một id động cho một trang web, với các mục đích khắc phục các điểm yếu sau của một hệ thống thông thường:

+ Dữ liệu lớn, kiểu khóa chính auto imcrement cần nhiều byte để lưu hơn. Và khóa chính kiểu này không phù hợp khi mà hệ thống có nhiều server, nhiều client cùng lúc truy cập trên toàn thế giới.
+ Nếu dùng khóa chính kiểu auto imcrement, có thể dễ dàng truy ra được trong database có bao nhiêu record. Thường thấy ở đường dẫn kiểu "domain.com/user/12345".

- Socket.io:

Socketio sẽ giúp các bên ở những địa điểm khác nhau kết nối với nhau, truyền dữ liệu ngay lập tức thông qua server trung gian. Socketio có thể được sử dụng trong nhiều ứng dụng như chat, game online, cập nhật kết quả của một trận đấu đang xảy ra,... Socketio không phải là một ngôn ngữ, mà chỉ là 1 công cụ giúp thực hiện những ứng dụng realtime. Vì thế, không thể sử dụng socketio để thay thế hoàn toàn cho một ngôn ngữ, mà phải sử dụng kết hợp với một ngôn ngữ khác. Ngôn ngữ đó có thể là php, asp.net, nodejs,...

- Express:

Expressjs là một framework được xây dựng trên nền tảng của Nodejs. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile. Expressjs hỗ trợ các method HTTP và midleware tạo ra API vô cùng mạnh mẽ và dễ sử dụng. Các chức năng chính của nó là:

+ Thiết lập các lớp trung gian để trả về các HTTP request.
+ Define router cho phép sử dụng với các hành động khác nhau dựa trên phương thức HTTP và URL.
+ Cho phép trả về các trang HTML dựa vào các tham số.

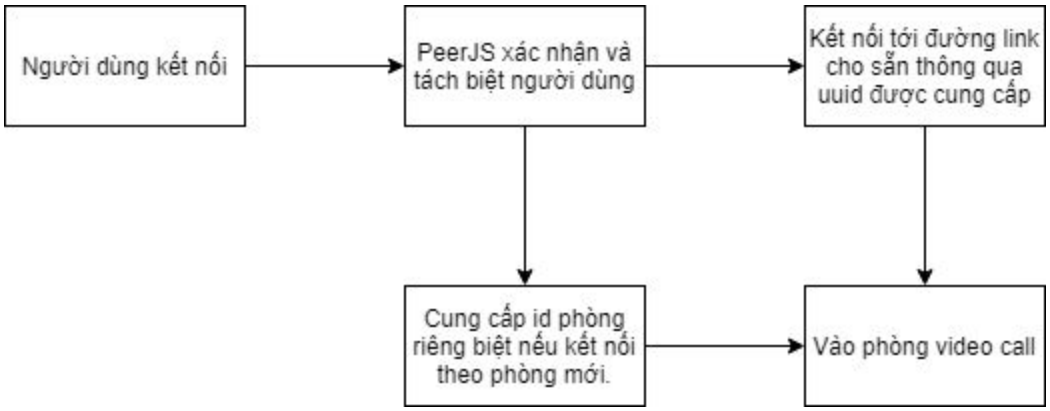
4. Phân tích hệ thống

- Cách hệ thống hoạt động:

Khi server được kích hoạt, mỗi client kết nối sẽ được gán cho một mã thông qua PeerJS nhằm để phân biệt giữa các người dùng. Sau đó, thông qua UUID, mỗi người khi kết nối sẽ được đưa vào một phòng với một tên đặc biệt luôn được tạo ra dưới dạng động nhờ bản chất của API này. Việc sử dụng UUID và Peer sẽ cho ta những lợi thế sau:

+ Phát hiện và sử dụng được những “phòng” riêng thông qua mỗi tên phòng được ngẫu nhiên hóa nhờ UUID. Hai người ở hai phòng khác nhau không thể kết nối.
+ Hiển thị dòng video stream đi và nhận của người dùng.

- Biểu đồ hoạt động:



5. Phân tích mã nguồn

Trước khi bắt đầu ta cần cài đặt và bổ sung các module cần thiết của node. Ngoài các module được nói trên ra chúng ta cần tổng cộng các module sau thông qua các mã sau bằng cmd của windows:

- Socket.io:
+ npm i socket.io -g

- Express:
+ npm i expresss -g

- Ejs:
+ npm i ejs -g
- Uuid:
+ npm i uuid -g

- nodemon:
+ npm i --save-dev nodemon -g

- Thư viện peerjs:
+ npm install peerjs -g

Đầu tiên, ta cần phải nhập các module này vào một file serverjs và sử dụng UUID để nó có thể gắn cho ta một roomid một cách ngẫu nhiên dưới dạng động:

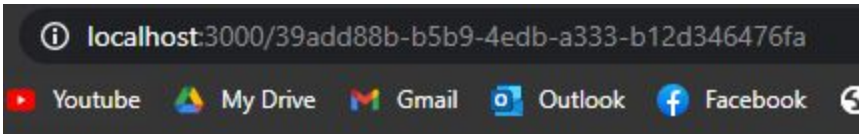
```
const express = require('express')
const app = express()
const server = require('http').Server(app)
const io = require('socket.io')(server)
const { v4: uuidV4 } = require('uuid')

app.set('view engine', 'ejs')
app.use(express.static('public'))

app.get('/', (req, res) => {
  res.redirect(`/${uuidV4()}`)
})

server.listen(3000)
```

Thông qua mã này, tại cổng 3000 của localhost chúng ta sẽ thấy được một roomid ngẫu nhiên được gắn vào url.



Chúng ta cần phải sử dụng lệnh Get nhằm có thể tải được một file ejs nhằm có thể hiển thị các thông tin này và đẩy lên front end của trang web với một cửa sổ window có kích thước 300x300px.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <script>
    const ROOM_ID = "<%= roomId %>"
  </script>
  <script defer
src="https://unpkg.com/peerjs@1.2.0/dist/peerjs.min.js"></script
>
  <script src="/socket.io/socket.io.js" defer></script>
  <script src="script.js" defer></script>
  <title>project 3</title>
  <style>
    #video-grid {
      display: grid;
      grid-template-columns: repeat(auto-fill, 300px);
      grid-auto-rows: 300px;
    }

    video {
      width: 100%;
      height: 100%;
      object-fit: cover;
    }
  </style>
</head>
```

```
<body>
  <div id="video-grid"></div>
</body>
</html>
```

Để có thể load được file ejs này, chúng ta cần thêm đoạn mã sau vào file js của server:

```
app.get('/:room', (req, res) => {
  res.render('room', { roomId: req.params.room })
})
```

Trong đoạn mã này chúng ta sẽ tạo ra file html cho thẻ video, đồng thời import các file socket.io.js và thư viện của peerjs vào cùng với file sript.js tự tạo. Cùng lúc dó, ta cũng lấy room_id động được đưa vào từ server.js vào trong file ejs. Ta sẽ tạo ra một đường dẫn công khai tĩnh và tạo ra script.js để viết các mã javascript dành riêng cho phần mềm.

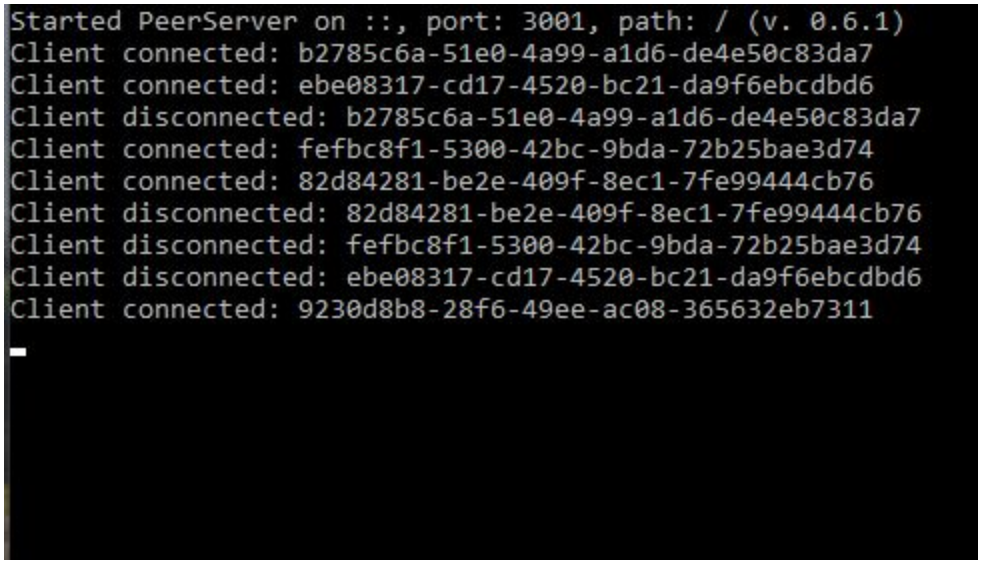
Trong server.js, chúng ta cần viết các đoạn mã socket.io để lúc một người dùng kết nối, chúng ta có thể truyền được roomid và userid liên quan tới người dùng đó.

```
io.on('connection', socket => {
  socket.on('join-room', (roomId, userId) => {
    socket.join(roomId)
    socket.to(roomId).broadcast.emit('user-connected', userId)

    socket.on('disconnect', () => {
      socket.to(roomId).broadcast.emit('user-disconnected',
userId)
    })
  })
})
```

Trong đoạn mã này chúng ta có thể thấy được hai sự kiện khi người dùng gia nhập phòng và nhận về được roomid và userid. Bước tiếp theo chúng ta sẽ truyền các thông tin này tới tất cả các người dùng khác trừ người đang truyền thông tin. Sự kiện user-connected sẽ được gửi đi và chúng ta cũng sẽ thông qua các userid của người đã kết nối. Cùng lúc đó, chúng ta cũng có một sự kiện tương tự cho lúc người dùng kết nối bằng các truyền userid của người đã ngắt kết nối.

Sau khi đã cài đặt thư viện peerjs một cách công khai thông qua đoạn mã `npm install peerjs -g`, ta có thể bắt đầu server của peer js trong cùng đường dẫn với server của mình thông qua mã `peerjs --port 3001` để mở cổng 3001 cho server js



Sau khi khởi động, chúng ta có thể thấy được từng người dùng kết nối và ngắt kết nối thông qua feedback từ chính cmd, đồng thời sẽ gán id cho người dùng tương ứng. Chính vì vậy, thông qua đoạn mã sau, ta sẽ gọi server socket.io và khởi tạo thư viện peerjs để truyền các tham số như host và số cổng. Khi đó mỗi lúc có người kết nối họ sẽ được gán tên tương ứng, và ta sẽ tạo một sự kiện truyền room_id và user_id hiện tại.

```
const socket = io('/')
const myPeer = new Peer(undefined, {
  host: '/',
  port: '3001'
})
myPeer.on('open', id => {
  socket.emit('join-room', ROOM_ID, id)
})
socket.on('user-connected', userId => {
    console.log("User Connected " + userId)
  })
```

Trong thẻ html của ta đang chứa một thẻ video, và chúng ta sẽ đặt vào trong scriptjs dòng mã sau:

```
const videoGrid = document.getElementById('video-grid')

const myVideo = document.createElement('video')
myVideo.muted = true

navigator.mediaDevices.getUserMedia({
  video: true,
```

```
  audio: true
}).then(stream => {
  addVideoStream(myVideo, stream)

})


function addVideoStream(video, stream) {
  video.srcObject = stream
  video.addEventListener('loadedmetadata', () => {
    video.play()
  })
  videoGrid.append(video)
}
```

Thông qua navigator api chúng ta có thể nhắc tới webcam để có thể hiển thị video và nối lại đường truyền này tới các yếu tố video.

Để có thể gọi các người dùng khác, chúng ta cần hiển thị được dòng video của người đó lên cửa sổ của mình thông qua mã:

```
socket.on('user-connected', userId => {
    console.log("User Connected " + userId)
    connectToNewUser(userId, stream)
  })

function connectToNewUser(userId, stream) {
  const call = myPeer.call(userId, stream)
  const video = document.createElement('video')
  call.on('stream', userVideoStream => {
    addVideoStream(video, userVideoStream)
  })
  call.on('close', () => {
    video.remove()
  })

  peers[userId] = call
}
```

Trong phương pháp này chúng ta dùng peerjs để gọi các người dùng khác với userid và đồng thời cũng truyền dòng video và tạo ra cửa sổ video mới khi đường truyền thành công.

Khi cuộc gọi đóng, ta sẽ xóa video khỏi cửa sổ hiện tại:

```
myPeer.on('call', call => {
    call.answer(stream)
    const video = document.createElement('video')
    call.on('stream', userVideoStream => {
      addVideoStream(video, userVideoStream)
    })
  })
```

Đoạn mã này sẽ cho ta nối một sự kiện trên đối tượng peer khi cuộc gọi đang hiện hữu và chúng ta cần trả lời và cho phép video của mình hiện lên tại cửa sổ của những người khác. Cuối cùng, để có thể tắt cửa sổ video của những người đã ngắt kết nối chúng ta cần sử dụng đoạn lệnh:

```
const peers = {}

function connectToNewUser(userId, stream) {
  const call = myPeer.call(userId, stream)
  const video = document.createElement('video')
  call.on('stream', userVideoStream => {
    addVideoStream(video, userVideoStream)
  })
  call.on('close', () => {
    video.remove()
  })

  peers[userId] = call
}
```

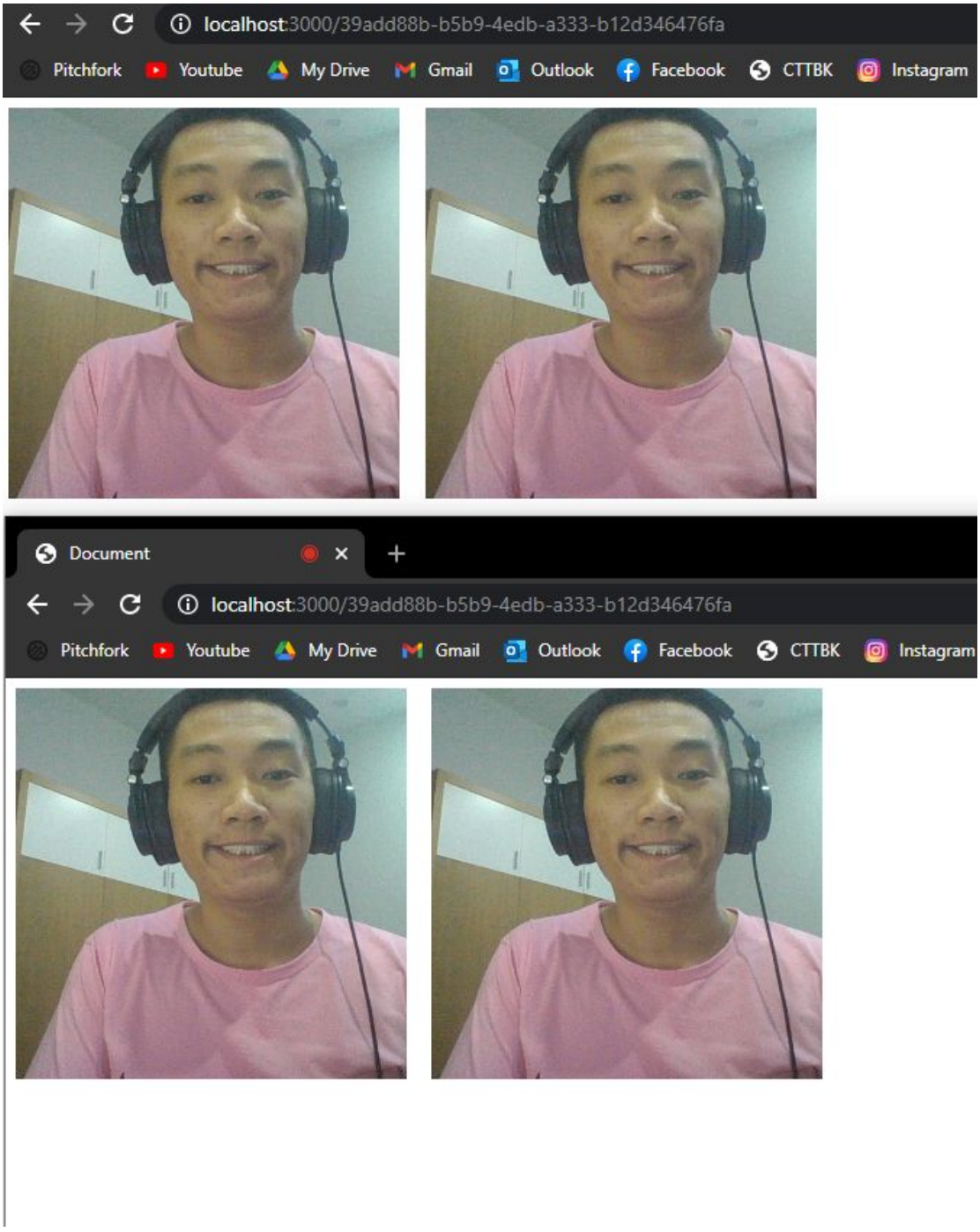
Khi chúng ta kết nối một người dùng mới chúng ta sẽ copy người dùng được nối vào trong đối tượng peers này

```
socket.on('user-disconnected', userId => {
  if (peers[userId]) peers[userId].close()
})
```

Cuối cùng khi người dùng rời cuộc gọi, sự kiện này sẽ được triển khai tự động bởi socket.io và đóng bằng peerjs.

6. Hình ảnh phần mềm

Do được yêu cầu rằng phần giao diện không cần phải làm quá kĩ càng, em đã viết chương trình dưới dạng cốt lõi nhất nhằm có thể hiểu được cách phần mềm hoạt động. Đối với em, nodejs và thiết kế web vẫn là một ngôn ngữ mới, và vẫn sẽ phải cố gắng để trau dồi kiến thức tạo dựng phần mềm.



7. Kết luận - Hướng phát triển

NodeJS vẫn là một ngôn ngữ mới với em. Việc phát triển được thành công một phần mềm thông qua các loại thư viện khác nhau đã khiến em được mở mang hơn về công nghệ này. Tuy được khuyên dùng sử dụng SIPJS nhưng do em vẫn còn cảm thấy cá nhân mình chưa đủ khả năng để triển khai nó một cách rõ ràng, đồng thời cũng muốn thử sức với một công nghệ dễ hơn nhằm có thể tiến từ từ. Cũng từ đây, phần mềm cũng có thể được tích hợp vào các trang web ổn định và có giao diện thân thiện hơn với người dùng nhằm giúp họ có thể giao tiếp trong một môi trường thích hợp. Em xin cảm ơn thầy giáo Nguyễn Tuấn Dũng đã hướng dẫn và giúp đỡ em trong quá trình học tập cũng như phát triển phần mềm này.