**ỦY BAN NHÂN DÂN TP HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO CHUYÊN ĐỀ**

**ỨNG DỤNG WEB CHAT**

**SỬ DỤNG SOCKET.IO**

**Giảng viên hướng dẫn : Lê Minh Nhựt Triều**

**Nhóm : 12**

**Họ và tên sinh viên : Trần Minh Tâm**

**Trần Văn Phúc**

***TP. Hồ Chí Minh, tháng …11.. năm 2019***

**MỤC LỤC**

LỜI MỞ ĐẦU 4

CHƯƠNG 1: GIỚI THIỆU SOCKET.IO 5

1.1 Khái Niệm Real-Time 5

1.1.1 Sự ra đời của khái niệm realtime 5

1.1.2 Định nghĩa realtime 5

1.2 Khái niệm Socket 6

1.2.1 Tại sao người dùng lại cần đến socket? 7

1.3 Khái niệm WebSocket 7

1.3.1 Cơ chế hoạt động của WebSocket 7

1.4 Socket.IO 9

1.5 Các tính năng chính của Socket.IO 9

1.5.1 Độ tin cậy 9

1.5.2 Hỗ trợ tự động kết nối lại 9

1.5.3 Phát hiện mất kết nối 10

1.5.4 Hỗ trợ truyền dữ liệu dạng binary (nhị phân) 10

1.5.5 Hỗ trợ đa kết nối 10

1.5.6 Hỗ trợ ”room” 11

CHƯƠNG 2: USER GUIDE 12

     2.1   Cài đặt môi trường 12

     2.2   Khởi động ứng dụng 14

CHƯƠNG 3: DEMO ỨNG DỤNG WEB CHAT SỬ DỤNG CÔNG NGHỆ

SOCKET.IO 16

3.1 Register vào ứng dụng 16

3.1.1 Register thất bại 21

3.1.2 Register thành công 23

3.2 Private Message 24

3.3 Chat All(Broadcasting) 28

3.4 Rooms 30

3.5 Disconnect 36

3.6 Logout 37

CHƯƠNG 4: KẾT LUẬN 39

TÀI LIỆU THAM KHẢO 41

LỜI MỞ ĐẦU

Trong thế giới phát triển mạnh mẽ về công nghệ hiện nay đã mang lại nhiều ích lợi và thuận tiện hơn cho cuộc sống. Trong đó, đặc biệt phải kể đến các công nghệ và các ứng dụng realtime đã giúp cho người dùng được tương tác với thông tin, dữ liệu hoặc thậm chí giao tiếp giữa người và người một cách chính xác và nhanh chóng hơn bao giờ hết. Ví dụ như chức năng thông báo trên facebook, chức năng chat trên messenger , ...

Và trong thời kì mà khái niệm realtime còn mới mẻ thì để xây dựng một ứng dụng đáp ứng những chức năng theo thời gian thực là một khó khăn rất lớn. Nhưng với sự phát triển như hiện nay thì đã xuất hiện nhiều công cụ hỗ trợ cho việc triển khai ứng dụng realtime một cách dễ dàng hơn. Tiêu biểu nhất đó là thư việc Socket.IO hỗ trợ cho nền tảng web và có thể kết hợp linh hoạt với nhiều ngôn ngữ khác nhau. Socket.IO là công cụ rất thích hợp để nghiên cứu và tìm hiểu cho việc phát triển ứng dụng real time một cách nhanh chóng và dễ dàng.

Một ứng dụng realtime sẽ có rất nhiều bài toán cần giải quyết. Trong số đó, chúng tôi xin phép chỉ đi tìm hiểu sâu về phần cốt lõi của một ứng dụng real time, đó là phần tương tác thông tin giữa người dùng với nhau. Thông qua việc trình bày các khái niệm cơ bản và demo một ứng dụng chat có sử dụng thư viện Socket.IO. nhóm em sẽ trình bày một web chat sử dụng Socket.Io có các chức năng: gửi tin nhắn broadcast, gửi tin nhắn riêng, gửi tin nhắn theo nhóm để làm rõ các tính năng của Socket.IO.

**CHƯƠNG 1: GIỚI THIỆU SOCKET.IO**

* 1. **Khái niệm real-time**
     1. **Sự ra đời của khái niệm realtime**

Sự phát triên mạnh mẽ của công nghệ đã giúp cải thiện và nâng cao được trải nghiệm của người dùng hiện nay. Nếu trước đây, để có thể cập nhật thông tin hoặc dữ liệu trên một ứng dụng thì người dùng cần cập nhật lại ứng dụng bằng kết nối lại với máy chủ thì hiện nay bài toán đó đã được giải quyết một cách một cách hoàn toàn bằng khái niệm realtime.

Giả sử bạn đang trong một tình huống rất hay gặp trong công việc hiện nay là đợi một email từ đồng nghiệp hoặc khách hàng quan trọng. Đối với một ứng dụng truyền thống thông thường thì bạn phải liên tục refresh (làm mới) lại ứng dụng để có thể thấy được danh sách email mới nhất. Nguyên nhân là do mô hình sử dụng của các ứng dụng thường là request – response. Máy chủ chỉ trả kết quả khi được người dùng yêu cầu. Và trải nghiệm liên tục phải nhấn F5 hoặc mở lại ứng dụng liên tục không phải là trải nghiệm dễ chịu chút nào.

Và đó khi khái niệm realtime ra đời nhằm giải quyết vấn đề này. Khái niệm mang ý nghĩa là theo thời gian thực, tức sự phản hồi hoặc thông tin mới sẽ được ứng dụng cập nhật liên tục và hoàn toàn tự động. Như ở tình huống trên thì realtime sẽ thể hiện cụ thể qua tính năng hiện thông báo khi có email gửi tới, giúp người dùng hoàn toàn chủ động hơn trong công việc, không phải chầu chực chờ đợi email nữa.

Với việc mang lại nhiều ích lợi và giải quyết được nhiều bài toán thì realtime được xem là một trong những tính năng phải có của hầu hết các ứng dụng hiện nay.

* + 1. **Định nghĩa realtime**

Realtime là thuật ngữ chỉ việc phản hồi và tương tác của người dùng với ứng dụng được diễn ra trong một khoảng thời gian rất ngắn, gần như đạt đến trạng thái tức thời.

Chúng ta có thể nhìn thấy tính năng realtime ở hầu hết các ứng dụng hiện nay, mà tiêu biểu nhất chính là qua các ứng dụng trò chuyện trực tuyến hoặc các ứng dụng theo dõi thông tin,...

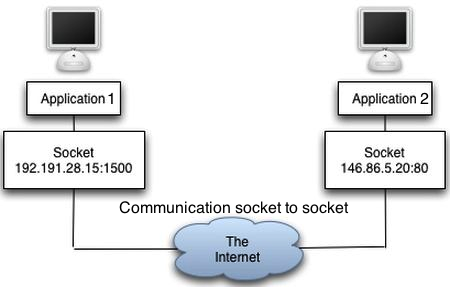
Một số ví dụ về các dạng ứng dụng realtime:

* **Ứng dụng chat** – Các ứng dụng trò chuyện trực tuyến như Whatsapp, Facebook Messenger, v.v. Tính năng realtime được thể hiện rõ ở việc bạn không cần phải làm mới ứng dụng/website của mình để nhận được các nội dung chat mới.
* **Ứng dụng theo dõi thông tin** – Các ứng dụng như theo dõi thông tin thời tiết, theo dõi thông tin chứng khoán,... giúp người dùng được tiếp cận với những thông tin hoặc dữ liệu mới nhất tính theo từng giây.
* **Game Online** – Các trò chơi như Liên Minh Huyền Thoại, PUBG, Đột Kích, FIFA ONLINE 4 v.v.., nhờ vào tính năng realtime mà nhiều người dùng được kết nối và tương tác trực tuyến với nhau trong một ván đấu hoặc trò chơi.

**1.2 Khái niệm Socket**

Hiện nay có nhiều cách và công nghệ để triển khai một ứng dụng hoặc tính realltime, mà trong đó phải kể đến Socket.

[Socket](https://en.wikipedia.org/wiki/Socket) được hiểu là một điểm cuối (end-point) của liên kết giao tiếp hai chiều (two-way communication) giữa hai chương trình chạy trên một môi trường mạng. Socket được ràng buộc bởi hai thông số: địa chỉ IP và số hiệu cổng (port) và thông qua các tầng TCP (TCP Layer) để định danh ứng dụng và dữ liệu được gửi đến.

Với địa chỉ IP sẽ giúp định danh cụ thể một máy tính và số hiệu cổng sẽ giúp phân biệt được các ứng dụng chạy trên cùng một máy tính.

***Hình 1.1***: Hình minh họa Socket

Như trong hình 1.1 đang minh họa cụ thể về một kết nối Socket giữa hai ứng dụng. Application 1 với Socket có địa chỉ IP (192.191.28.15) và số hiệu cổng (1500) kết nối với Application 2 với Socket có địa chỉ IP (146.86.5.20) và số hiệu cổng (80).

Ta có thể hiểu đơn giản Socket là một chiếc điện thoại giúp kết nối liên lạc giữa 2 người với nhau. Và để biết chính xác người cần tạo kết nối với nhau, ta cần đó là số điện thoại tương ứng với địa chỉ IP và số hiệu cổng.

### **Tại sao người dùng lại cần đến socket?**

Socket hiện là cách thức triển khai ứng dụng realtime đơn giản, nhanh, hiệu quả, ổn định và được nhiều nhà lập trình viên tin dùng. Hiện nay, Socket rất linh hoạt hỗ trợ ở hầu hết các hệ điều hành và cũng như các ngôn ngữ lập trình. Bên cạnh đó, trong quá trình làm việc, các bạn có thể chạy nhiều socket cùng một lúc để nâng cao hiệu suất của công việc.

* 1. **Khái niệm WebSocket**

**WebSoket** là công nghệ hỗ trợ giao tiếp hai chiều giữa client và server bằng cách sử dụng một TCP socket để kết nối. Ban đầu WebSocket được thiết kế để chỉ chuyên sử dụng cho nền tảng Web, nhưng hiện nay thì có thể sử dụng nó cho cả các nền tảng còn lại.

WebSockets chỉ mới xuất hiện trong HTML5 và là một kỹ thuật Reverse Ajax. Hiện nay WebSockets được hỗ trợ hầu hết trên các trình duyệt hiện đại như Chorme, Firefox, Safari, ...

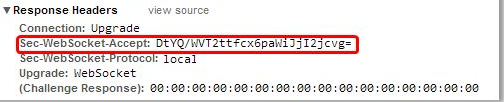
**1.3.1 Cơ chế hoạt động của WebSocket**

WebSocket hoạt động chủ yếu dựa trên cơ chế handshake. Vòng đời của một kết nối Websocket bắt đầu bằng một chuẩn HTTP đơn thuần. Và khi trình duyệt gửi một requyest (yêu cầu) lên máy chủ sẽ được kèm theo một yêu cầu mở kết nối WebSocket và kèm theo WebSocket-Key được thêm trong header của request (Hình 1.2).



***Hình 1.2***: Minh họa một request tạo kết nối WebSocket

Khi nhận được yêu cầu và đồng ý tạo kết nối, máy chủ gửi trả lại một response (phản hồi) có kèm theo một WebSocket-Accept được tạo và mã hóa dựa vào WebSocket-Key (Hình 1.3).



***Hình 1.3***: Minh họa một response tạo kết nối WebSocket

Ngay khi trình duyệt nhận được response đồng ý của máy chủ sẽ tạo ngay một kết nối handshake. Và bắt đầu trao đổi dữ liệu giữa trình duyệt và máy chủ thông qua kết nối này cho đến khi một bên đóng kết nối lại.

Hãy làm một phép so sánh nhanh qua lượng dữ liệu header mà giao thức HTTP và WebSocket trong mỗi giây. Một header request/response của HTTP có kích thước trung bình khoảng 871 byte, trong khi với WebSocket, kích thước này chỉ là 2 byte (sau khi kết nối thành công). Vậy nếu có 1000 người sử dụng cùng lúc trên ứng dụng của bạn thì:

* HTTP: 871 x 10,000 = 8,710,000 bytes = 69,680,000 bits mỗi giây (66 Mbps)
* WebSocket: 2 x 10,000 = 20,000 bytes = 160,000 bits mỗi giây (0.153 Kbps)

Ta dễ dàng thấy được điểm mạnh mà WebSocket mang lại, đó chính là tối ưu được dung lượng băng thông từ đó tăng tốc độ cho ứng dụng.

**1.4 Socket.IO**

Sau khi nắm được các khái niệm như Socket và WebSocket thì ta có thể dễ dàng định nghĩa Socket.IO là gì. Socket.IO chỉ đơn giản là một thư viện triển khai công nghệ WebSocket bằng ngôn ngữ JavaScript. Socket.IO không phải là một ngôn ngữ và khi sử dụng ta phải triển khai nó cùng với một ngôn ngữ lập trình khác như: php, c#, java, ...

Thư viện Socket.IO sẽ bao gồm 2 phần ứng với mô hình hoạt động của WebSocket là client và server:

* + - Client: gồm các thư viện giúp triển khai Socket.IO trên các ứng dụng. Và tùy vào ngôn ngữ đang sử dụng trên ứng dụng mà ta có thể tùy chọn thư viện Socket.IO hỗ trợ cho ngôn ngữ ấy. Ví dụ: như ứng dụng của bạn đang sử dụng c# thì bạn có thể tải và cài đặt SocketIoClientDotNet thông qua NuGet Package trên Visual Studio.
    - Server: gồm thư viện giúp triển khai một server hỗ trợ Socket.IO trên các máy chủ JavaScript.
  1. **Các tính năng chính của Socket.IO**

Do Socket.IO triển khai theo công nghệ WebSocket nên bản thân nó thừa hưởng được tất cả các ưu điểm cũng tiện lợi của WebSocket. Bên cạnh đó thư viện Socket.IO còn hỗ trợ nhiều tính năng và đồng thời khắc phục được các mặt hạn chế mà công nghệ WebSocket gặp phải. Trong đó phải kể đến các tính năng tiêu biểu sau:

* + 1. **Độ tin cậy**

Một hạn chế lớn trong việc triển khai một WebSocket đơn thuần đó là vấn đề kết nối WebSocket không thể hoạt động ổn định khi phía client có cấu hình liên quan đến Proxies hoặc Load Balancer (Cân bằng tải). Ngoài ra nếu client có cài đặt phần mềm diệt virus hoặc bật tính năng firewall trên máy tính thì cũng làm ảnh hưởng lớn đến kết nối WebSocket.

Tất cả vấn đề trên hoàn toàn được khắc phục trong thư viện Socket.IO giúp kết nối giữa client và server được tối ưu hơn được về mặt thời gian và cũng như cấu hình công nghệ WebSocket dễ dàng hơn.

### **Hỗ trợ tự động kết nối lại**

### Khi kết nối giữa client và server bị gián đoạn hoặc mất kết nối thì thư viện Socket.IO có hỗ trợ khôi phục và giúp kết nối trở lại. Ngoài ra người lập trình cũng có thể thiết lập các thông số cho cơ chế tự động kết nối lại này.

### Các tùy chỉnh cho việc kết nối lại:

### reconnectionAttempts : số lần thử kết nối lại.

### reconnectionDelay: thời gian giữa mỗi lần kết nối lại.

### **...**

### **Phát hiện mất kết nối**

### Với cơ chế heartbeat được triển khai trong thư viện Socket.IO, cho phép các máy phía client và máy phía server nhận biết khi nào máy còn lại không phản hồi nữa và từ đó có cách khắc phục và hướng xử lý tuy theo chức năng hoặc nghiệp vụ của ứng dụng.

Cơ chế heartbeat được thiết lập ngay khi kết nối handshake được tạo. Mỗi bên client lẫn server sẽ gửi một lệnh “ping” đi. Sau khi nhận được ping sẽ gửi trả lại một lệnh “pong”, cứ như vậy lệnh “ping” và “pong” được gửi và nhận sen kẽ. Từ đó hai bên sẽ đảm bảo kết nối luôn duy trì và xuyên suốt .

### **Hỗ trợ truyền dữ liệu dạng Binary (nhị phân)**

Với WebSocket ta chỉ có thể gửi dữ liệu dưới dạng string (Json string). Và chính điều này phần nào hạn chế đi khả năng mở rộng của ứng dụng hay đáp ứng của WebSocket đối với một số nghiệp vụ hoặc tính năng mong muốn. Ví dụ như: gửi file hoặc trò chuyện trực tuyến bằng video. Những dạng dữ liệu trên đều ở dạng binary nên không thể nào WebSocket có thể giải quyết được giúp ta những vấn đề như vậy.

Nhưng đến với thư viện Socket.IO đã có chức năng hỗ trợ gửi và nhận dữ liệu dạng Binary, như:

* + - Trên browser: ArrayBuffer và Blob (Binary Large Objects).
    - Trên Node.js: ArrayBuffer và Buffer.

### **Hỗ trợ đa kết nối**

### Socket.IO cho phép tạo cùng lúc nhiều kênh kết nối khác nhau nhưng vẫn sử dụng duy nhất một kết nối WebSocket duy nhất. Và Socket.IO dùng khái niệm namespace để dễ dàng quản lý và định nghĩa một kênh kết nối như vậy. Mỗi một namespace sẽ được định nghĩa và khởi tạo ở phía server. Khi một client muốn kết nối vào namespace sẽ gửi một request đến server.

Với việc tạo nhiều kênh kết nối, Socket.IO sẽ giúp nhà phát triển có thể phân tách các loại nghiệp vụ khác nhau. Ví dụ: authorization user (phân quyền người dùng) chẳng hạn. Tùy vào quyền của người dùng mà kết nối với một kênh khác nhau. (admin => namespace: “admin”, guest => namespace: “guest).

### **Hỗ trợ “Room”**

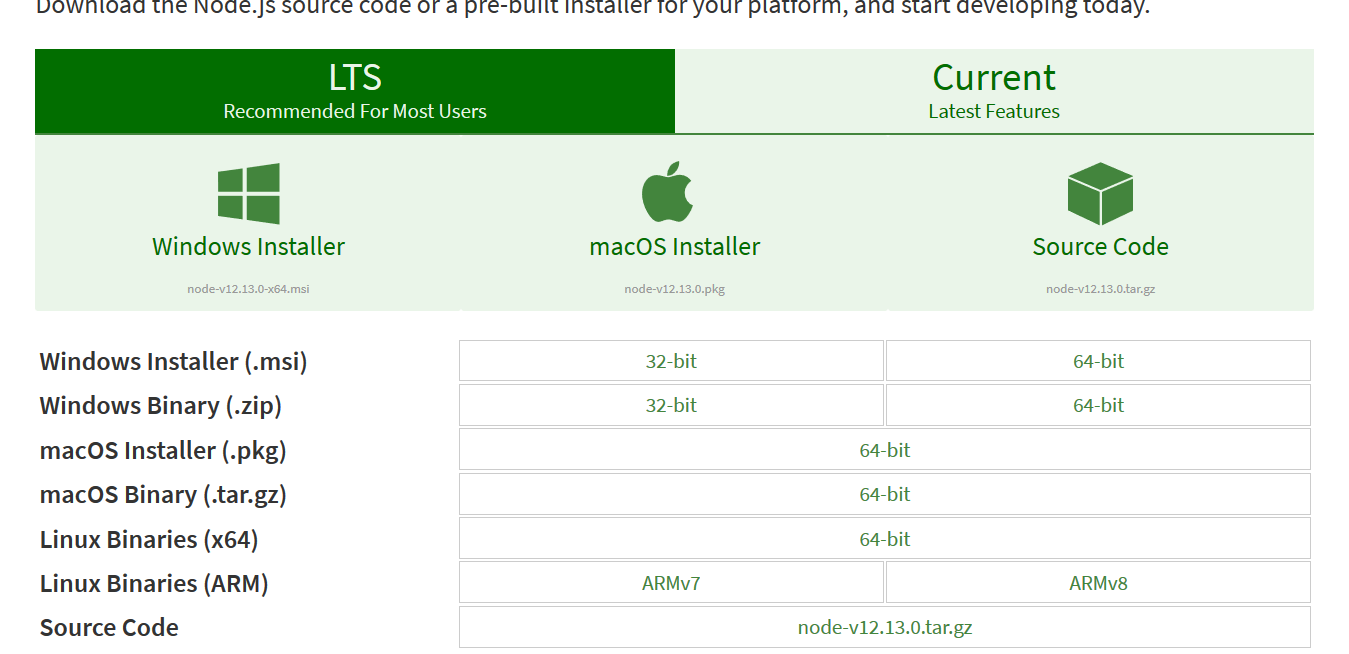
### Trong mỗi Namespace, Socket.IO lại cho phép chia nhỏ một kênh kết nối thành các “Room”. Và để tham gia Room thì phía client phải gửi lệnh join() và phải được sự chấp thuận của phía server. Trong trường hợp muốn thoát Room thì client gửi lệnh leave().

### Đây là một tính năng hữu ích trong các trường hợp nhử gửi thông báo cho một nhóm người dùng hoặc cho một người dùng nhất định được kết nối trên nhiều thiết bị khác nhau.

**CHƯƠNG 2: USER GUIDE**

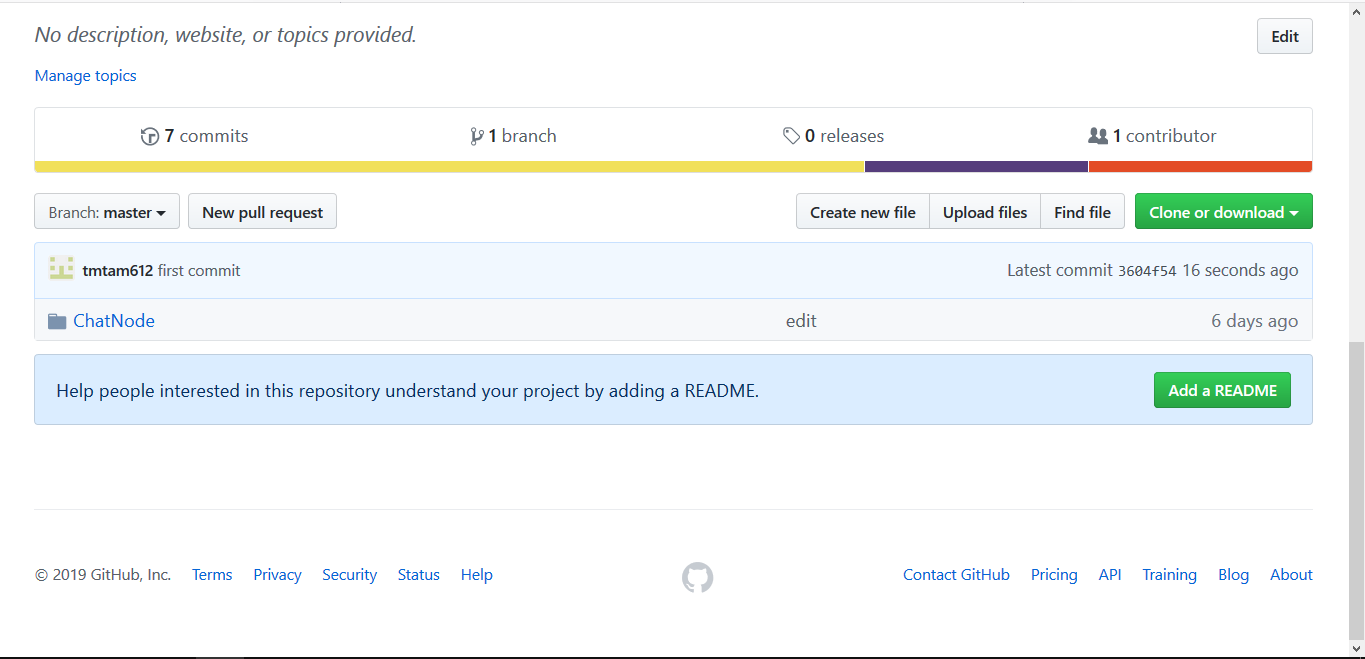
**2.1 Cài đặt môi trường**

Đầu tiên chúng ta cần phải cài môi trường nodejs, truy cập đến trang web trng chủ của nodejs để download <https://nodejs.org/en/download/>. Khi nhấn vào đường link cửa sổ sẽ hiện ra như thế này

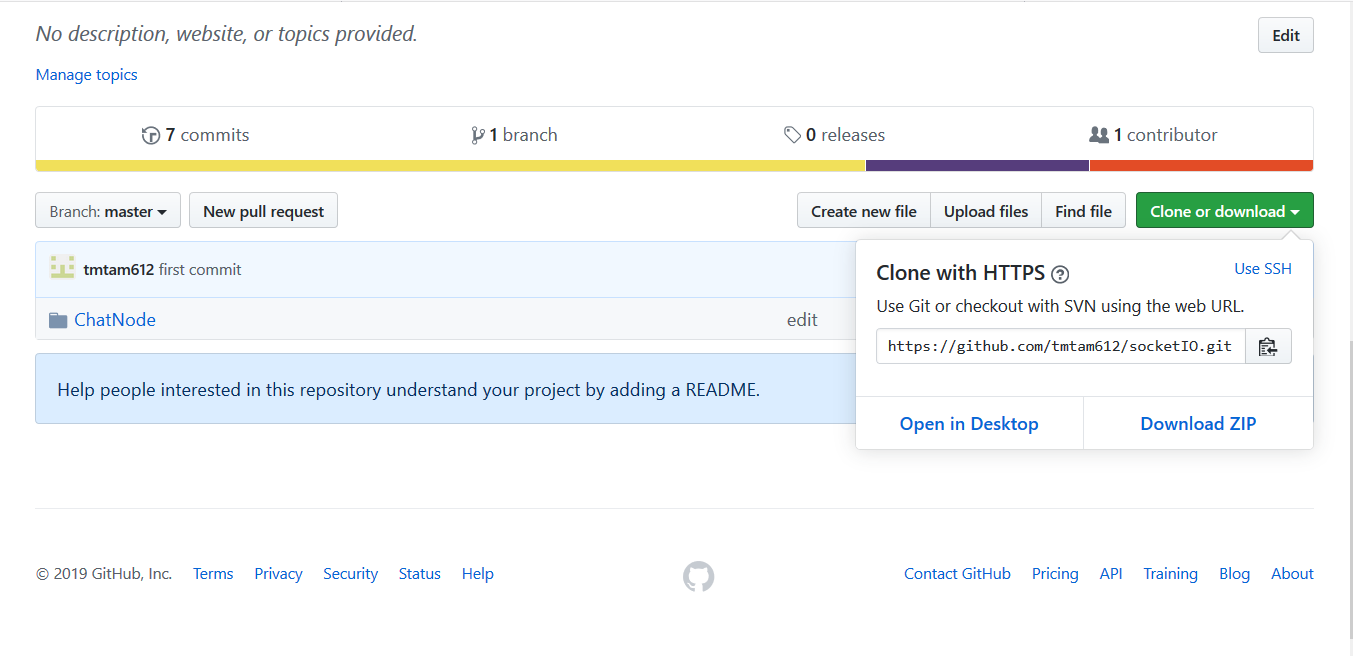


Download bản phù hợp với hệ điều hành của mình, và chạy cài đặt bình thường như những phần mềm ứng dụng khác.

Tiếp theo truy cập đến trang web git để download phần souce code về. Truy cập đến đường link <https://github.com/tmtam612/socketIO>, sẽ ra màn hình như sau.

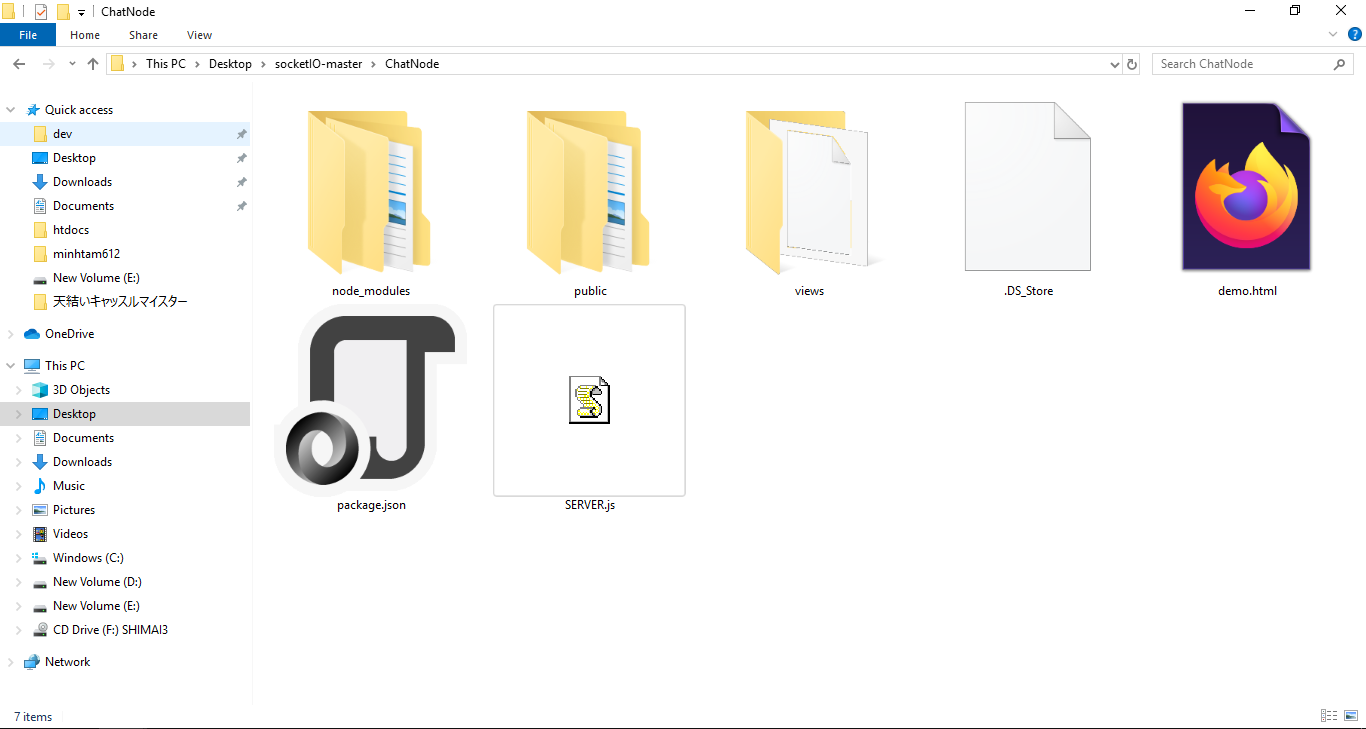


Nhấn vào nút clone or download, chọn tải về dưới dạng zip (Download Zip).

Sau khi tải về máy, giải nén file vừa mới tải về sẽ có những thư mục như sau.



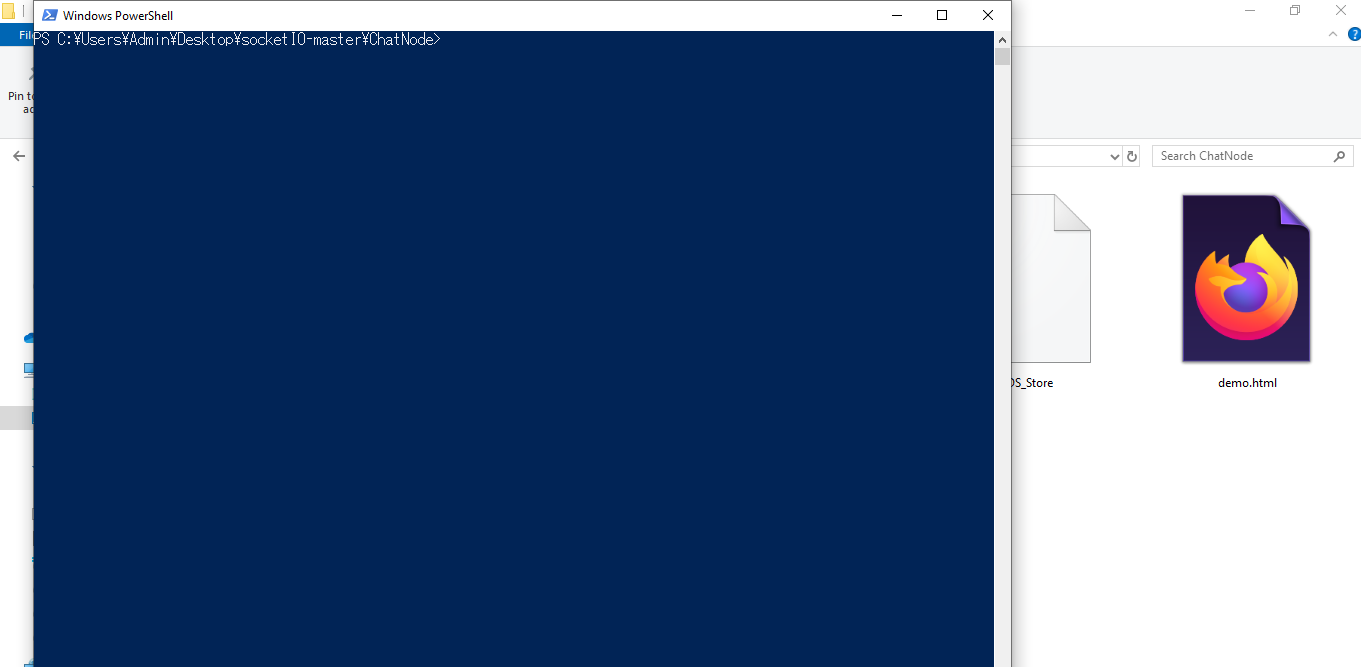
Truy cập vào thư mục ChatNode, có những file như sau là đã cài đặt môi trường thành công



**2.2 Khởi động ứng dụng**

Tại vị trí trong thư mục ChatNode, vào thanh đường dẫn gõ “cmd” và enter, hoặc mở cmd và dẫn vào thư mục bên trong ChatNode, hoặc bấm phím tắt shift+right click chọn open PowerShell window here.

Lúc này sẽ xuất hiện màn hình cmd/ powershell



Gõ cú pháp “node server.js” và giữ nguyên màn hình cmd đừng tắt chỉ thu nhỏ lại xuống thanh taskbar, ứng dụng web chat sẽ chạy dưới localhost với port 3000. Mở trình duyệt web lên và truy cập đường dẫn <http://localhost:3000/>, nếu hiện ra trang web như sau là thành công.

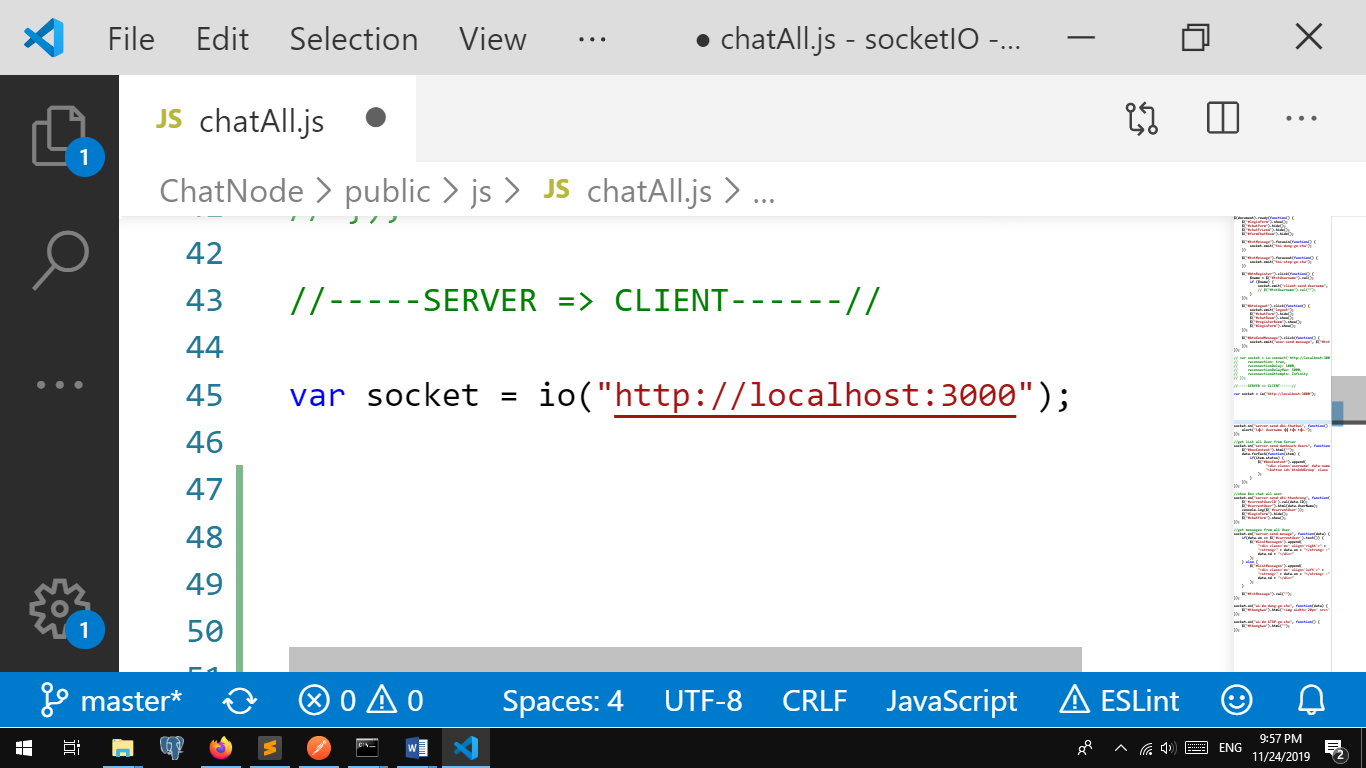


Nếu vẫn còn thắc mắc xin liên hệ gmail: [tmtam612@gmail.com](mailto:tmtam612@gmail.com)

**CHƯƠNG 3: DEMO ỨNG DỤNG WEB CHAT SỬ DỤNG CÔNG NGHỆ SOCKET.IO**

**3.1 Register vào ứng dụng**

Để tạo ứng dụng realtime socket.io, ở phần client, cần tạo biến socket để có thể phát hoặc nhận thông tin từ phía server. Vì mỗi trang web sẽ là một namespace ở dưới server, ta cần phải tạo một biến cục bộ socket cho trang web đó. Cú pháp khởi tạo biến socket cho namespace như sau



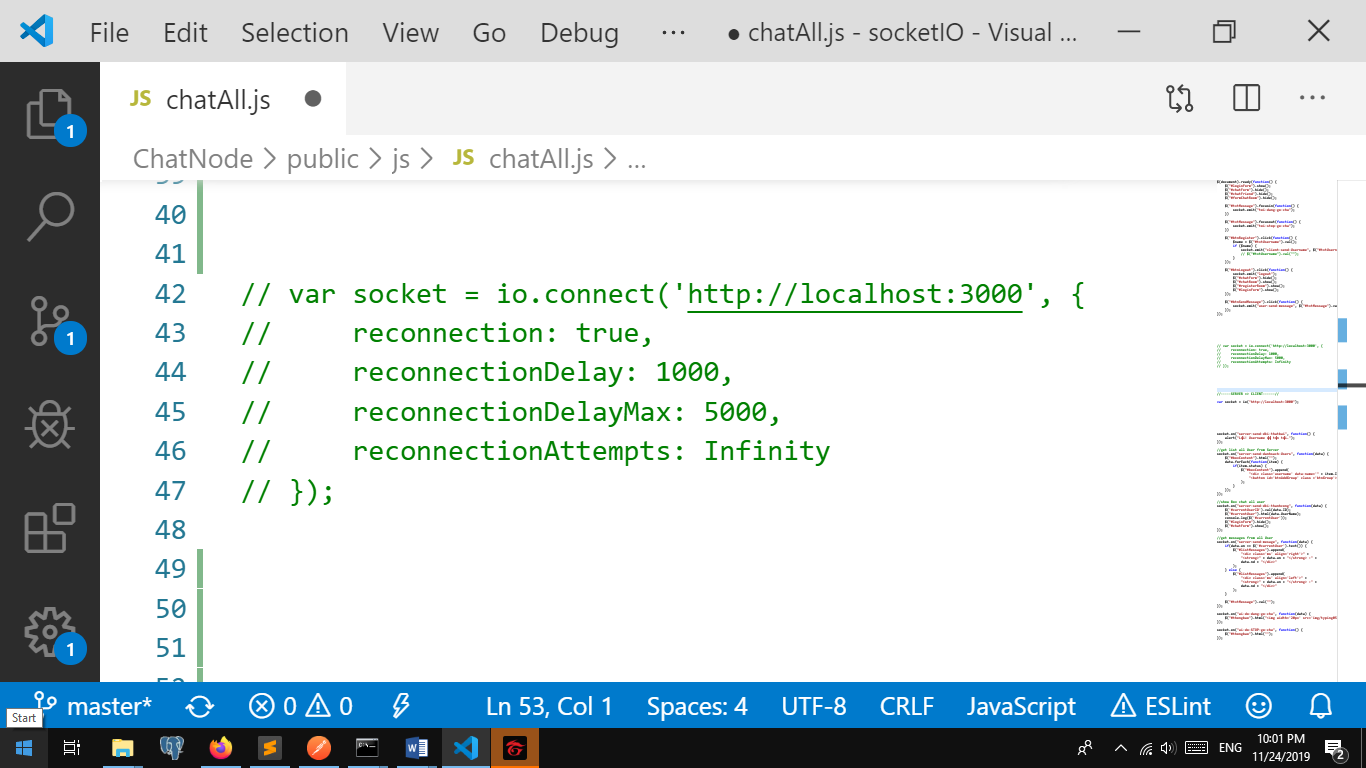
*Hình minh họa: tạo biến cục bộ socket*

Cú pháp io() truyền vào 1 tham số là tên đường dẫn đến server mà bạn cần liên lạc, để có thể gửi hoặc nhận thông tin với server, ở ví dụ trên là “http://localhost:3000”.

Cú pháp io() sẽ tự thuyết lập các tính năng tự động kết nối, số lần tự kết nối lại.

Có thể dùng cú pháp io.connect() để có thể thiết lập các tính năng khác của socket.io

Cú pháp io.connect() như sau



*Hình minh họa: tạo biến cục bộ socket*

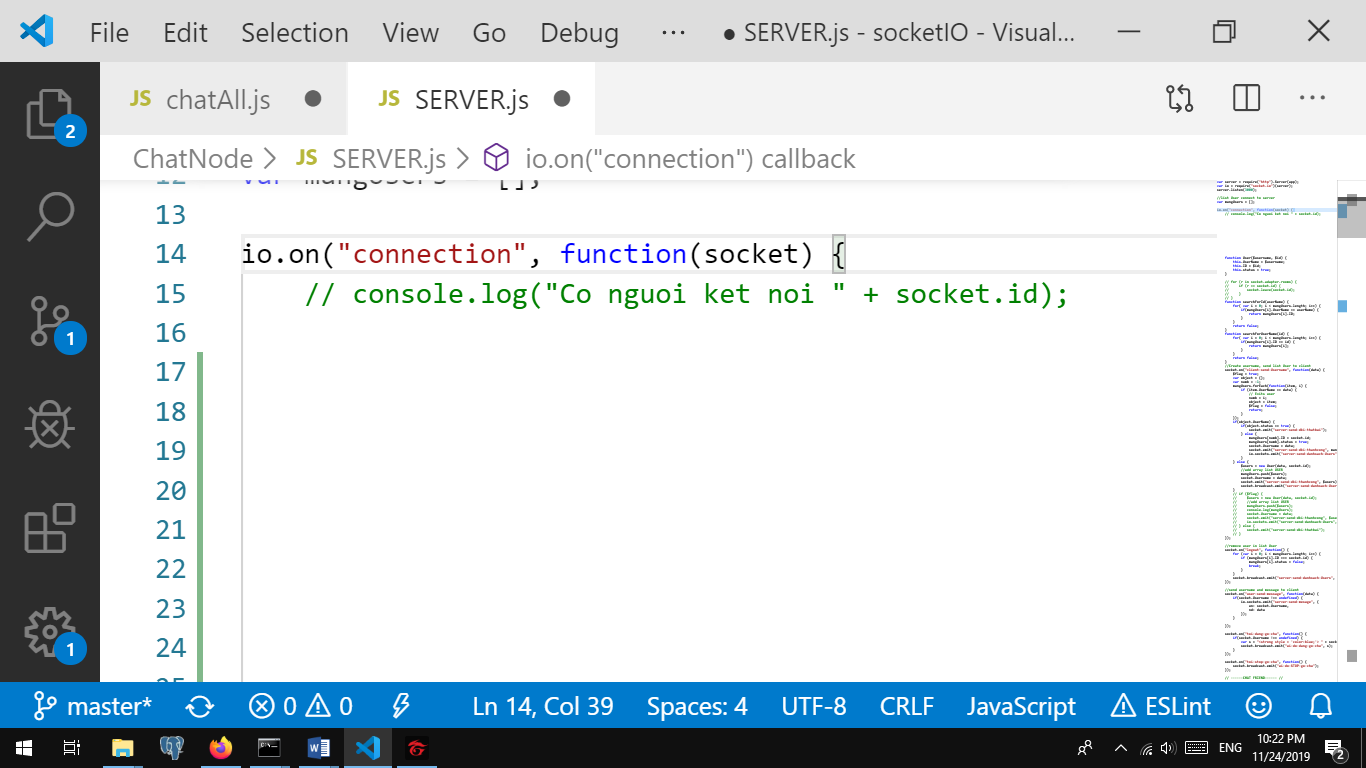
Cú pháp io.connect() gồm 2 phần:

* Phần thứ nhất là tên đường dẫn liên kết tới server, để client liên lạc với server.
* Phần thứ hai được thiết lập dưới dạng object, bao gồm các tính năng của socket mà người dùng cần thiết lập lại.

Ở cú pháp trên <http://localhost:3000> là đường dẫn tới server, phần đằng sau dấu phẩy là object các thuộc tính, bao gồm:

* reconnection: true, đây là thiết lập cho client có thể tự kết nối lại với server, nếu như ở 2 bên client, server có 1 bên gặp sự cố ngắt kết nối.
* reconnectionDelay: 1000, dùng để thiết lập thời gian liên lạc giữa client và server, ở đây thiết lập 1000 có nghĩa là liên lạc giữa client và server tốn thời gian là 1s.
* reconnectionDelayMax: 5000, là thiết lập thời gian tối đa mà một kết nối giữa client và server tốn, 5000 là số ms chỉ số giây tối đa mà kết nối đó có thể sử dụng.
* reconnectionAttempts: Infinity, dùng để thiết lập số lượng tự động kết nối lại đến server, nếu thiết lập 100, thì khi server chết, client sẽ tự động kết nối (ping) đến server 100 lần mỗi lần delay là 1s, nếu vượt quá số lần client sẽ không tự động kết nối nữa, ở đây thiết lập Infinity để client có thể kết nối(ping) tới server với số lần là vô hạn.

Khi đã tạo xong một biến socket ở client, client sẽ tự động gửi biến này tới server, server sẽ nhận được tín hiệu có một client đã kết nối đến.



*Hình minh họa: sự kiện connection được nhận ở phía server*

Cú pháp io.on(“connection”, function(socket) {}); tương ứng với sự kiện connection khi client gửi sự kiện connection tới server, Cú pháp .on() dùng để lắng nghe sự kiện được gửi tới.

Cú pháp on() gồm hai tham số:

* Tham số thứ nhất là tên của sự kiện được gửi tới (bắt buộc).
* Tham số thứ hai, có thể là một hàm dùng để xử lý sự kiện mà server nhận được (không bắt buộc).

Khi vừa truy cập đến trang web, nhập tên tham gia và nhấn nút register



*Hình minh họa: giao diện người dùng*

Nút register sẽ tạo một sự kiện gọi là đăng ký tài khoản với cú pháp như sau:



*Hình minh họa: sự kiện “client-send-Username” được gửi đi ở phía client*

Tất cả các phương thức ở phía client phát tới server đều được viết bằng ngôn ngữ javascript:

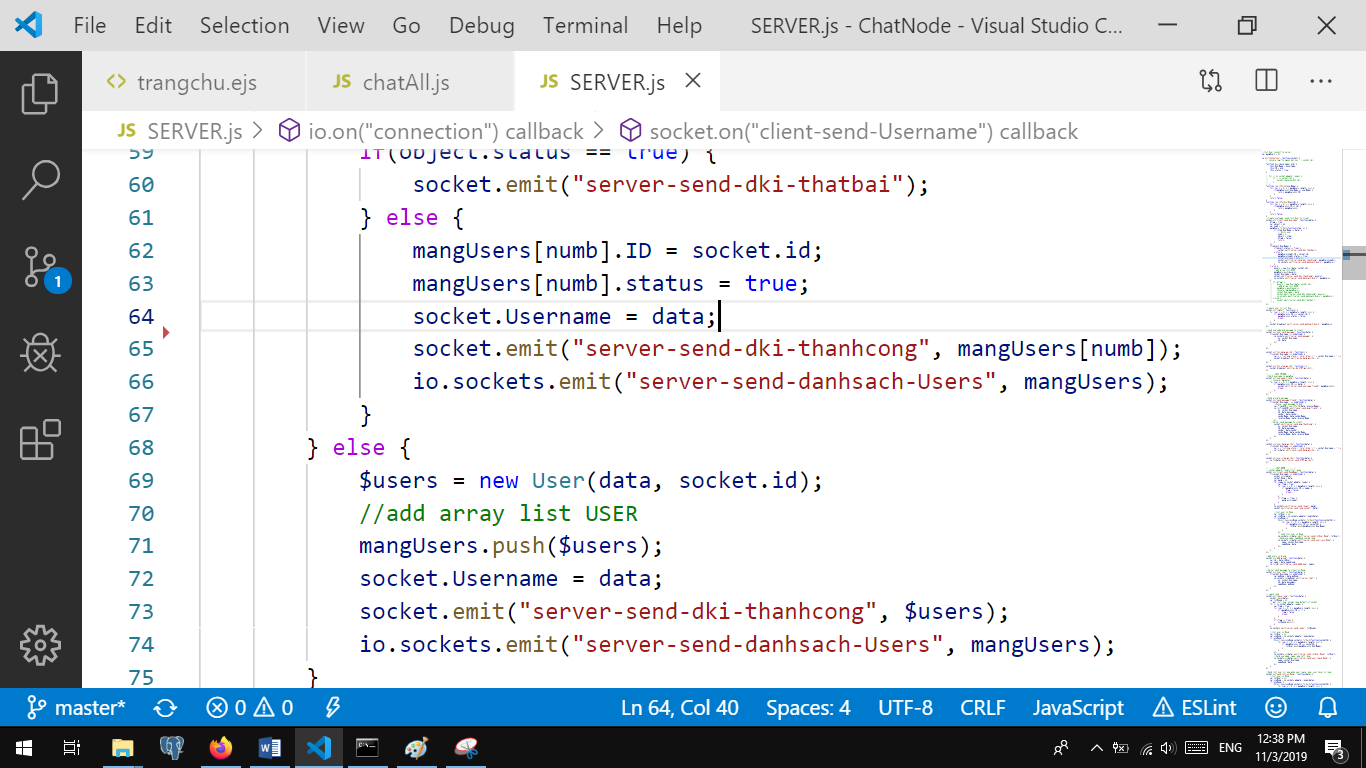
$(“#btnRegister”).click(): dùng để bắt sự kiện click của button register, khi người dùng nhấn nút button, client sẽ lấy name của người dùng bằng textbox có id là txtUsername qua cú pháp $(“#txtUsername”).val(), sau đó gửi đến server qua cú pháp emit().

Ở socket.io luôn luôn có 2 hàm gọi là on(), emit() dùng để nhận và gửi sự kiện, on() dùng để lắng nghe sự kiện, và emit() dùng để gửi sự kiện.

Cũng tương tự như cú pháp on(), cú pháp emit() gồm 2 tham số:

* Tham số thứ nhất là tên sự kiện sẽ gửi đi (bắt buộc).
* Tham số thứ hai là dữ liệu được gửi đi, để bên lắng nghe(on()), có thể nhận được dữ liệu và xử lý (không bắt buộc).





*Hình minh họa: hàm xử lý sự kiện ở phía server*

Như hình trên, client gửi sự kiện tên là client-send-Username bao gồm tên đăng nhập mà người dùng tạo đến server, server sẽ tạo một biến user có tên đăng nhập và socket.id đến mảng Users và gửi mảng Users này tới tất cả các socket để thông báo rằng đã có một user tham gia đến trang web qua cú pháp io.sockets.emit().

Cú pháp io.sockets.emit() gồm 2 tham số, cũng tương tự như cú pháp emit() thông thường:

* Tham số thứ nhất là tên sự kiện
* Tham số thứ hai là dữ liệu cần gửi.

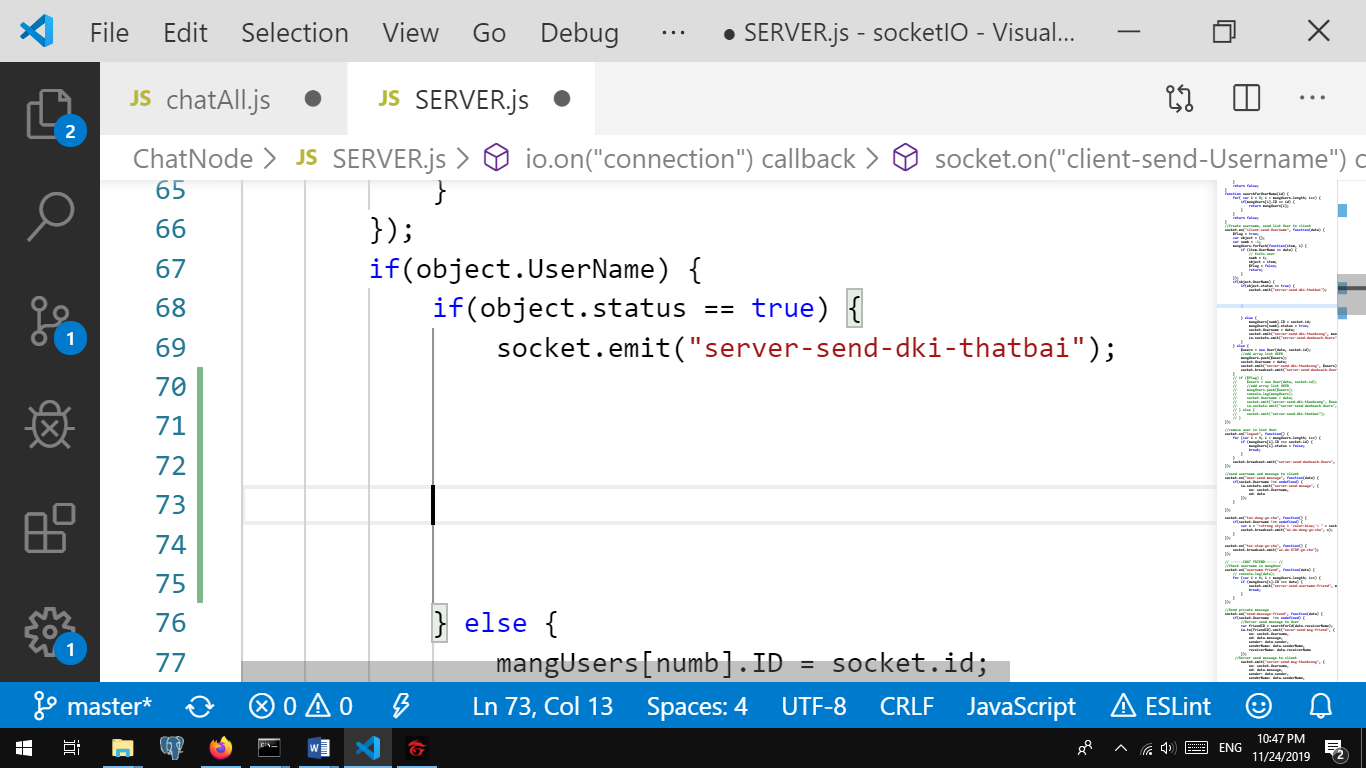
Nhưng khác với cú pháp emit() thông thường với đối tượng gọi là socket gửi đơn mục tiêu.

Io.socket.emit() sẽ gửi sự kiện đa mục tiêu, tức là nó sẽ gửi sự kiện tới tất cả client đang kết nối với server, ngoại trừ client mới tạo.

Biến socket.id rất quan trọng đối với mỗi client kết nối đến, nó tương tự như địa chỉ ip, giúp server phân biệt được client nào gửi đi và client nào sẽ được nhận. Tương tự như địa chỉ ip, nó có tính độc nhất, mỗi client chỉ có một socket.id và không có client nào trùng với client nào.

**6.1.1 Register thất bại.**

Trong sự kiện “client-send-Username”, server sẽ duyệt mảng Users, mảng Users ở đây chứa tất cả các tài khoản đã hoặc đang kết nối tới server, trong quá trình duyệt mảng nếu phát hiện tài khoản đang được kết nối với server, server sẽ gửi đến sự kiện có tên “server-send-dki-thatbai” để gửi thông báo đến client qua cú pháp gửi emit.

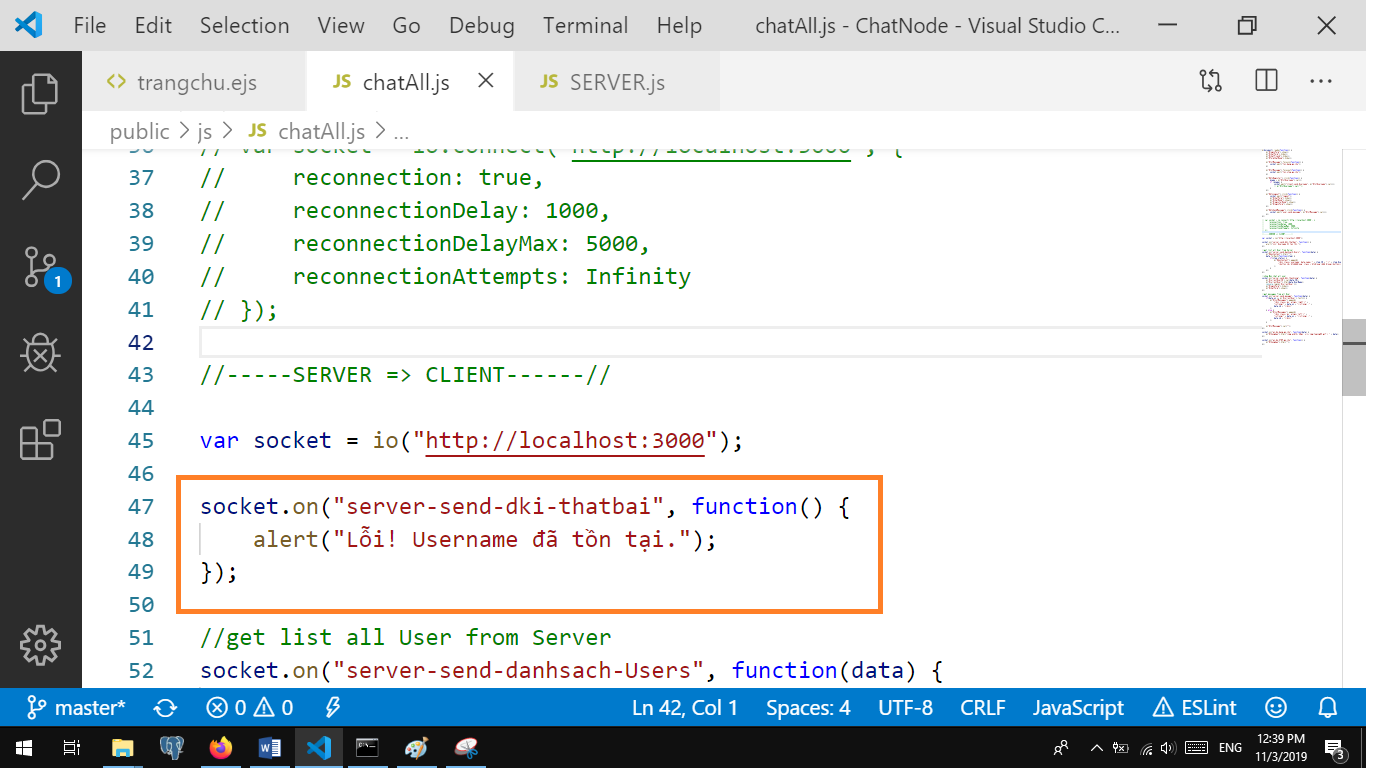


*Hình minh họa: sự kiện “server-send-dki-thatbai” gửi đi ở phía server*

Lần này hàm emit() sẽ gửi đi tên sự kiện, còn phần mở rộng dữ liệu gửi đi nếu không có, hàm on() sẽ thiết lập data được nhận tới mặc định là null.

Khi đó client sẽ có một sự kiện có sự kiện tên là “server-send-dki-thatbai” để hứng sự kiện được gửi từ phía server.

Thông qua sự kiện server-send-dki-thatbai, client sẽ nhận được sự kiện trên và tạo một message có thông tin đăng ký/ đăng nhập thất bại.



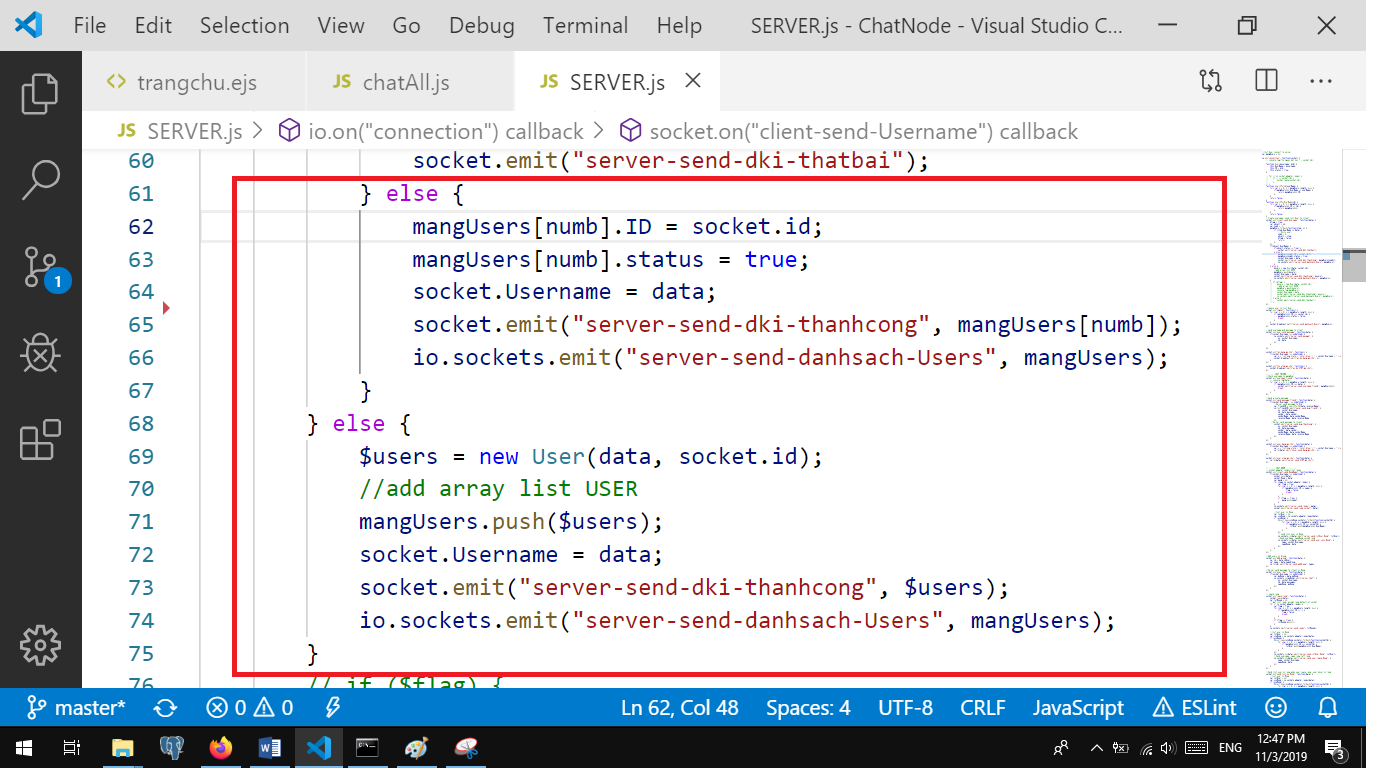
*Hình minh họa: sự kiện “server-send-dki-thatbai” nhận được ở phía client.*



*Hình minh họa giao diện người dùng*

**3.1.2 Register thành công**

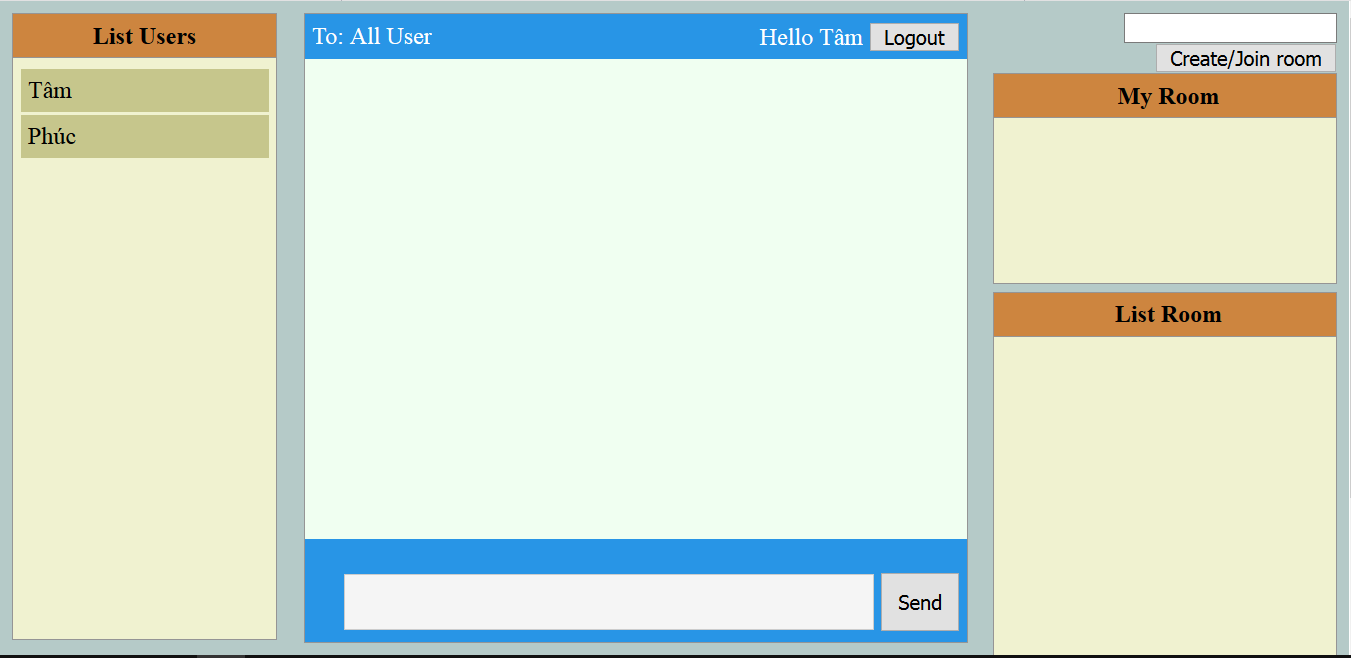
Nếu duyệt mảng, phát hiện tài khoản tồn tại nhưng đã offline, hoặc chưa tồn tại trong mảng Users, server sẽ tạo tài khoản, hoặc cập nhật socket.id của tài khoản.



*Hình ảnh minh họa: xử lý sự kiện ở phía sever*

Khi đó server sẽ phát đi một sự kiện “server-send-danhsach-Users” để cập nhật mảng Users đang online, và một sự kiện có tên là “server-send-dki-thanhcong” với tham số là tài khoản vừa được tạo hoặc vừa được cập nhật để vào trang chính của webchat.

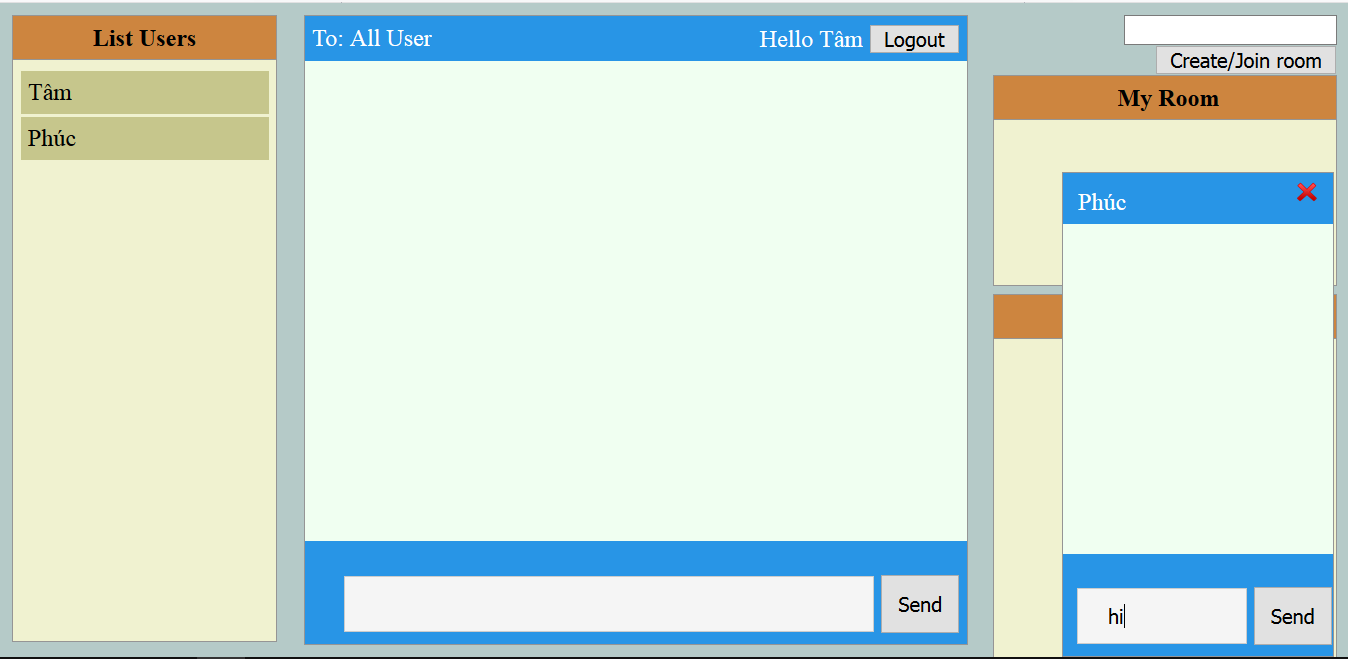
Các sự kiện này sẽ được client lắng nghe, và tạo giao diện cho người dùng.



*Hình ảnh minh họa: giao diện người dùng*

**3.2 Private Message**

Để gửi tin nhắn private, nhấn double click vào tên người dùng cần liên lạc, client sẽ tạo một hộp thoại private.



*Hình ảnh minh họa giao diện người dùng*

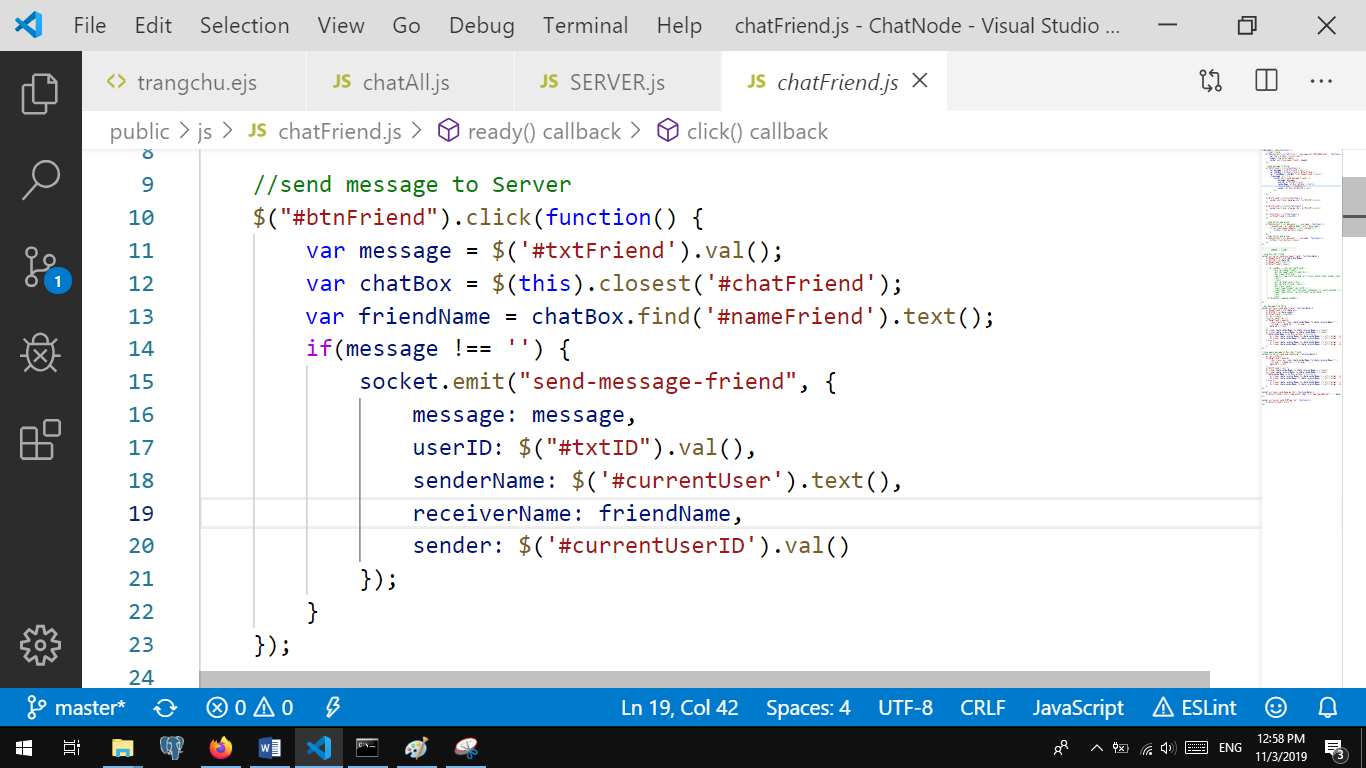
Khi nút Send được nhấn, hoặc nhấn nút enter, client sẽ tạo một sự kiện gửi tin nhắn, và phát đi đến server để giải quyết, server nhận được và giải quyết lập tức gửi lại cho người nhận tin nhắn.

Ở đây khi nút send được nhấn, sự kiện gửi sẽ lấy message là thứ người dùng nhập vào, và địa điểm gửi đi chính là socket#id của người nhận, và đẩy lên server với sự kiện là “send-message-friend”.

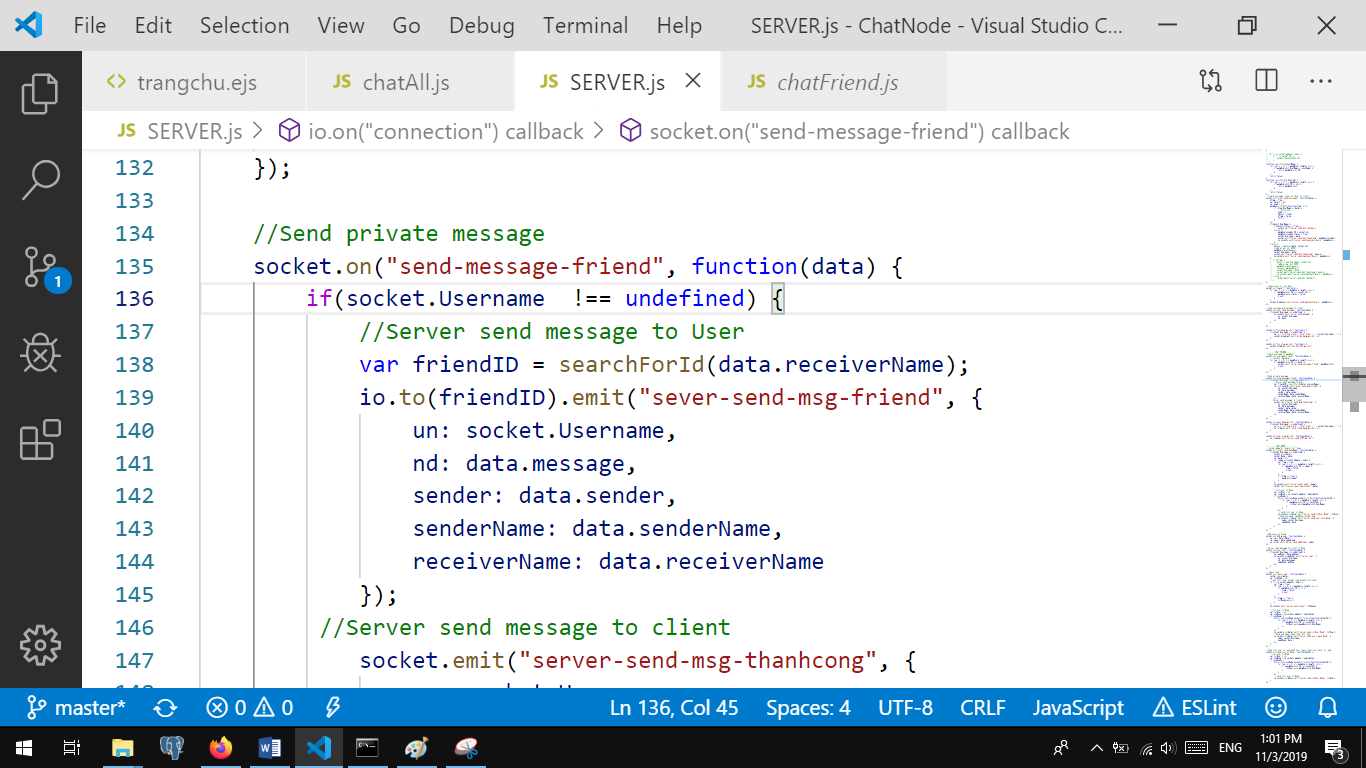
Với hàm emit(), client sẽ gửi đi sự kiện có tên là “send-message-friend”, và dữ liệu là một object bao gồm

* Message: tin nhắn cần gửi đi.
* userID: socket.id của người nhận.
* senderName: tên người gửi.
* receiverName: tên người nhận.
* sender: socket.id của người gửi.

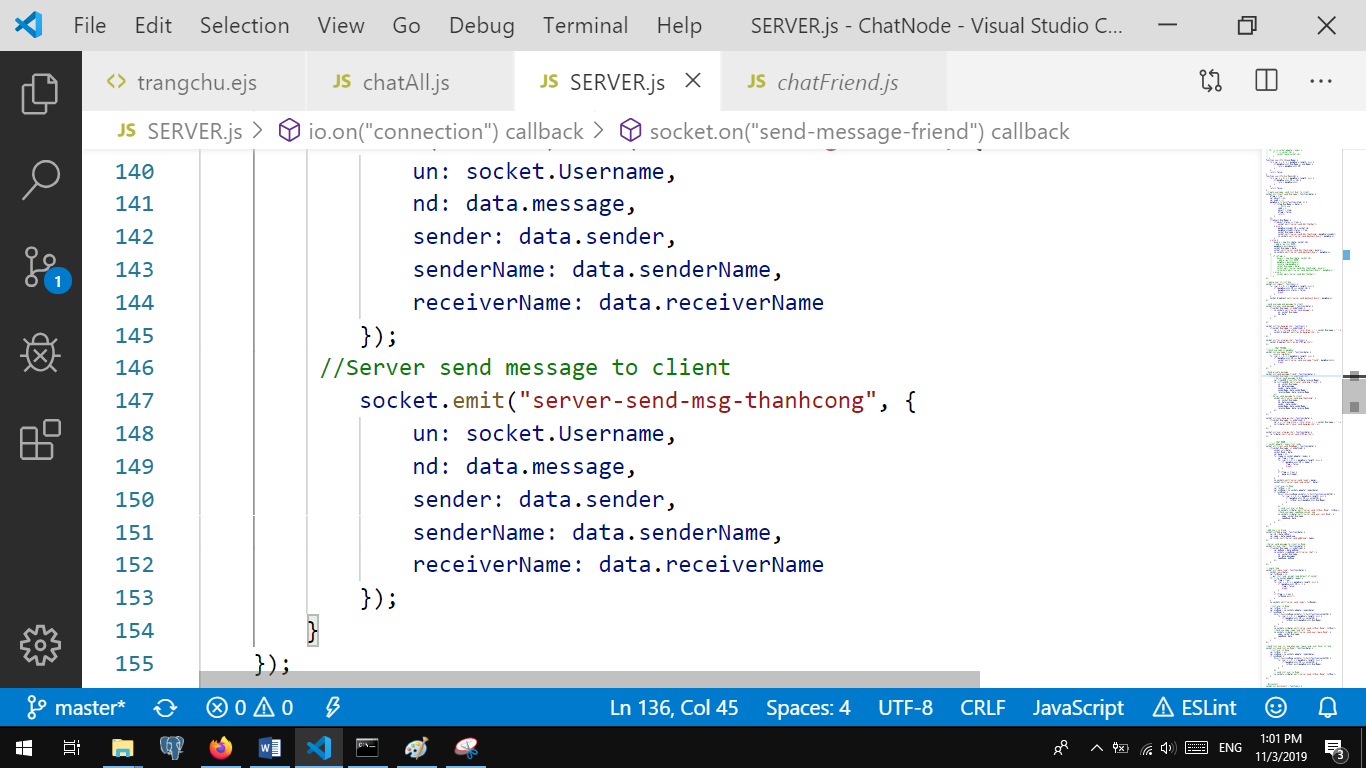
Khi đó ở phía server sẽ config một sự kiện tên là “send-message-friend” để hứng sự kiện được gửi tới.



*Hình ảnh minh họa: sự kiện “send-message-friend” được gửi đi ở phía client*



*Hình ảnh minh họa: sự kiện “send-message-friend” được nhận ở phía server.*



*Hình ảnh minh họa: sự kiện “server-send-msg-thanhcong” được gửi đi ở phía server.*

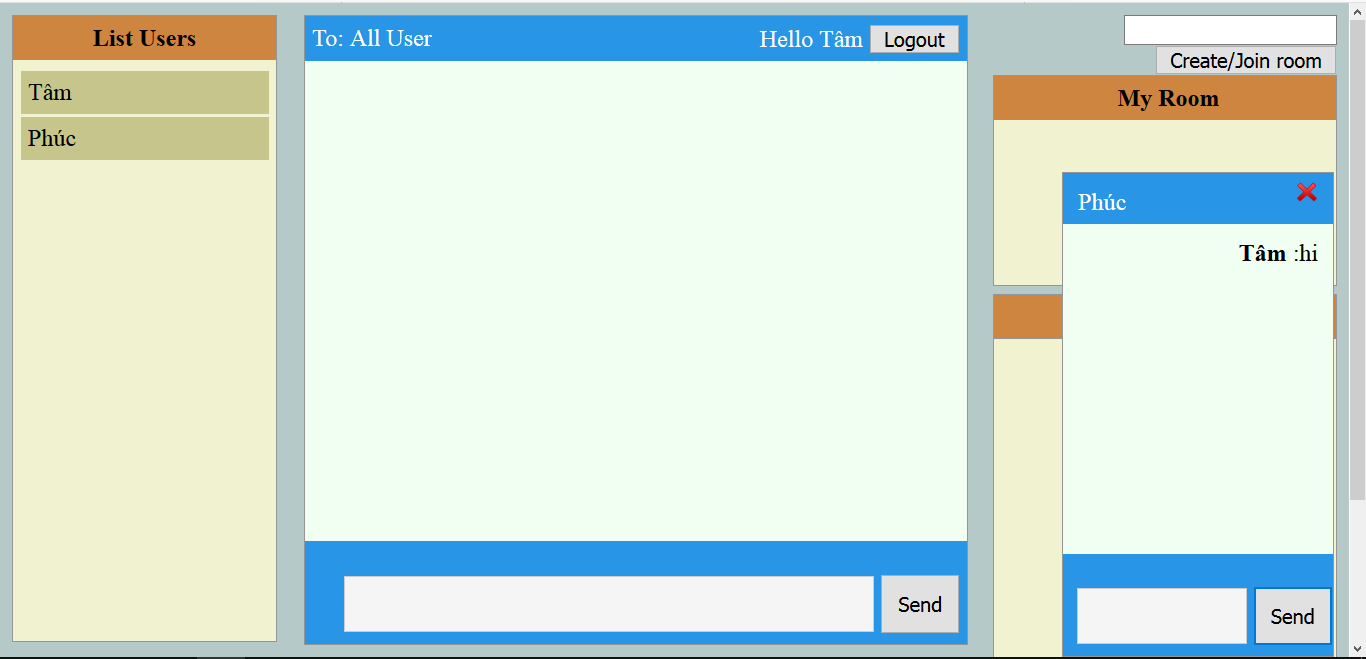
Trên server khi nhận được sự kiện “send-message-friend”, sẽ lập tức gửi đi 2 sự kiện “server-send-msg-friend”, sự kiện này sẽ gửi tin nhắn đến người nhận và “server-send-msg-thanhcong” để hiển thị cho chính người gửi thấy được tin nhắn mà mình đã gửi, client hứng và tạo giao diện, hiển thị lên cho cả người gửi và người nhận thông tin tin nhắn một cách tức thì.

Ở đây, để gửi từ client này để tới client khác dùng hàm to(), gồm 1 tham số truyền vào:

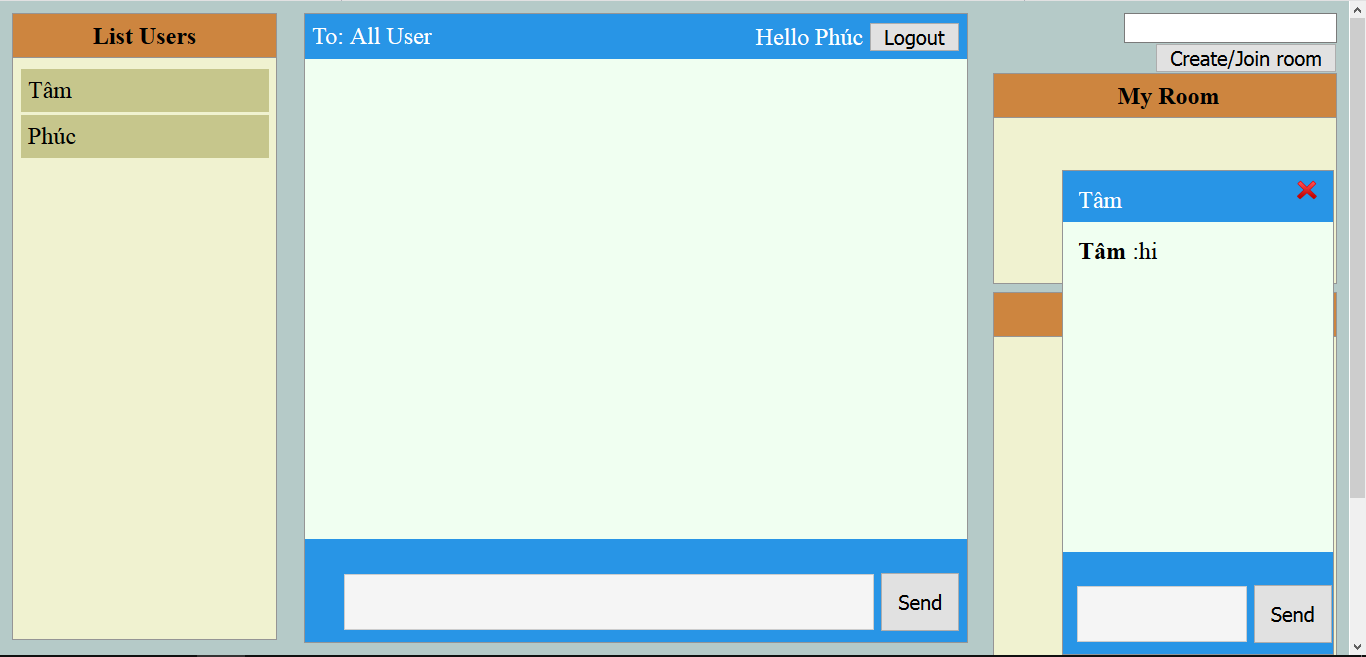
* Địa chỉ socket.id của socket cần liên lạc

Hàm to() dùng để gửi thông tin một cách private, khác với io.socket.emit() dùng để gửi tới tất cả client, hay client.emit() dùng để liên lạc tới server, to() dùng để liên lạc tới chỉ một client khác, và không được public đến mọi người.

**Lưu ý**: việc gửi tin nhắn private, phải thông qua socket#id của người nhận và người gửi, và socket#id này là của chính người dùng được tạo ra ngẫu nhiên bởi socket.io và không thể nào custom hay config được.



*Hình ảnh minh họa: giao diện người gửi*



*Hình ảnh minh họa: giao diện người nhận*

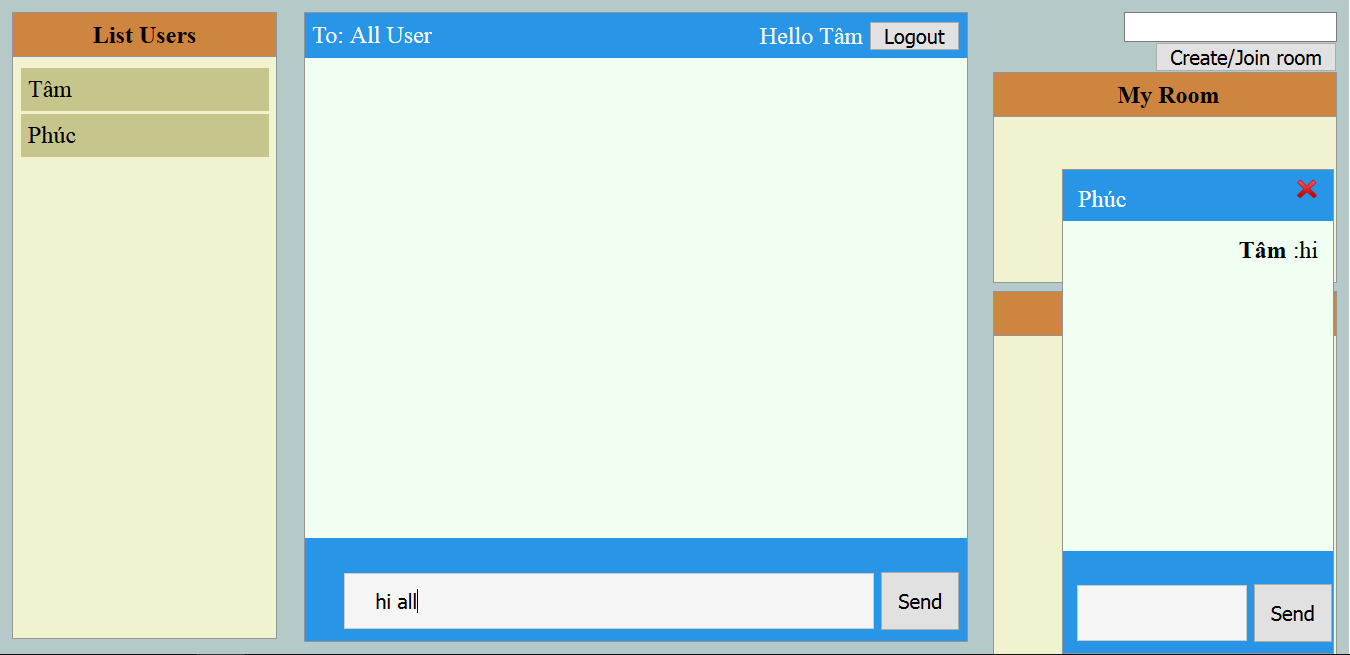
**3.3 Chat All(Broadcasting)**

Với socket.io, việc gửi tin nhắn broadcast là công việc đầu tiên cơ bản nhất, vì khi phát triển socket.io, trường hợp đầu tiên để phát triển đó chính là broadcast. Việc chat all, broadcast message đến mọi người dễ dàng hơn rất nhiều so với việc private message, hay chat group đến room được tạo.

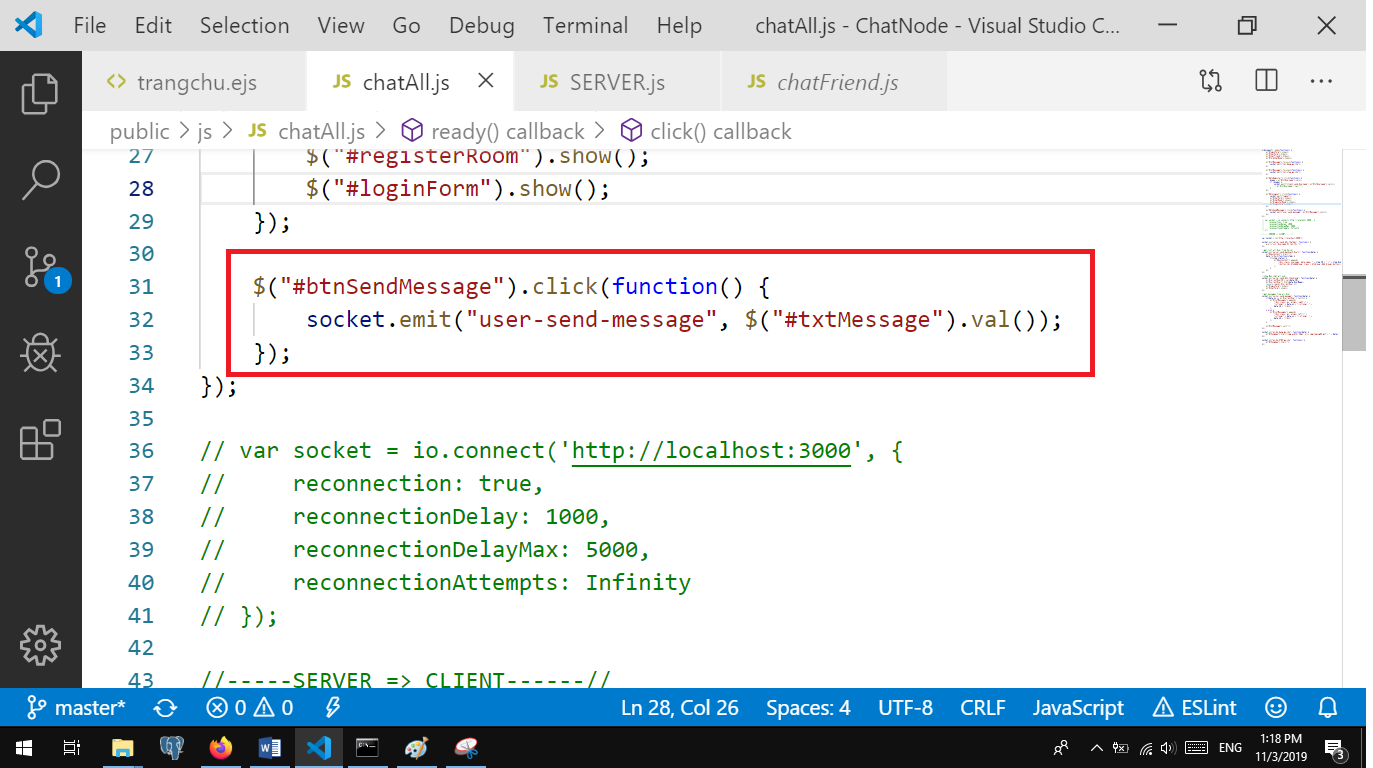
Khi gửi tin nhắn chat all đi, client chỉ làm một việc đó chính là gửi tin nhắn đến server, không cần thiết phải lọc tin nhắn xem gửi đến ai, hay gửi đến room cụ thể nào đó.

Server sẽ chỉ nhận được thông tin tin nhắn, và người gửi (ở đây chỉ cần socket#id hoặc username).

Mọi thứ trở nên dễ dàng hơn với phía server, server nhận và ngay tức thì gửi tin nhắn all tới tất cả user hiện tại trong trang web.



*Hình ảnh minh họa: giao diện người dùng*

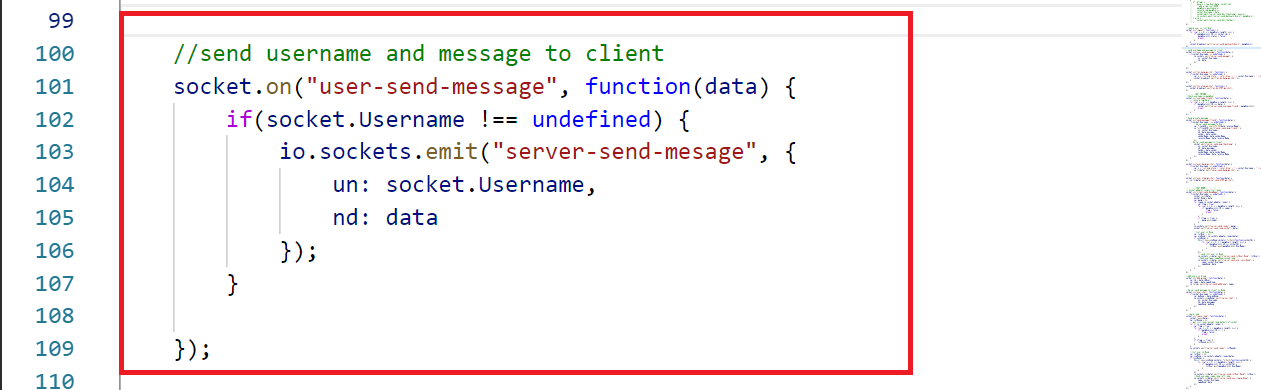


*Hình ảnh giao diện: sự kiện “server-send-message” được gửi đi ở phía client*

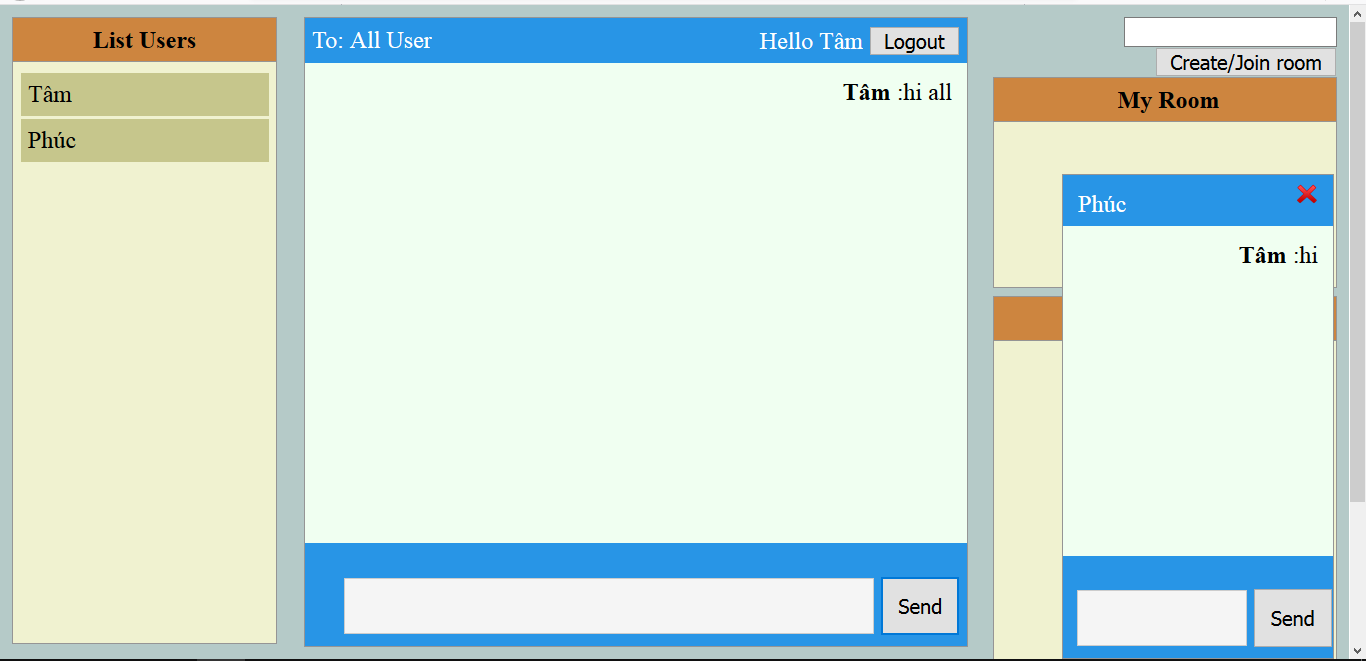
Khi server nhận được một sự kiện “user-send-message”, server ngay tức khắc gửi đi một sự kiện gọi là “server-send-message” để gửi tin nhắn tới mọi client đang kết nối.

Các client sẽ hứng sự kiện “server-send-message” và tạo giao diện đến tất cả người dùng.

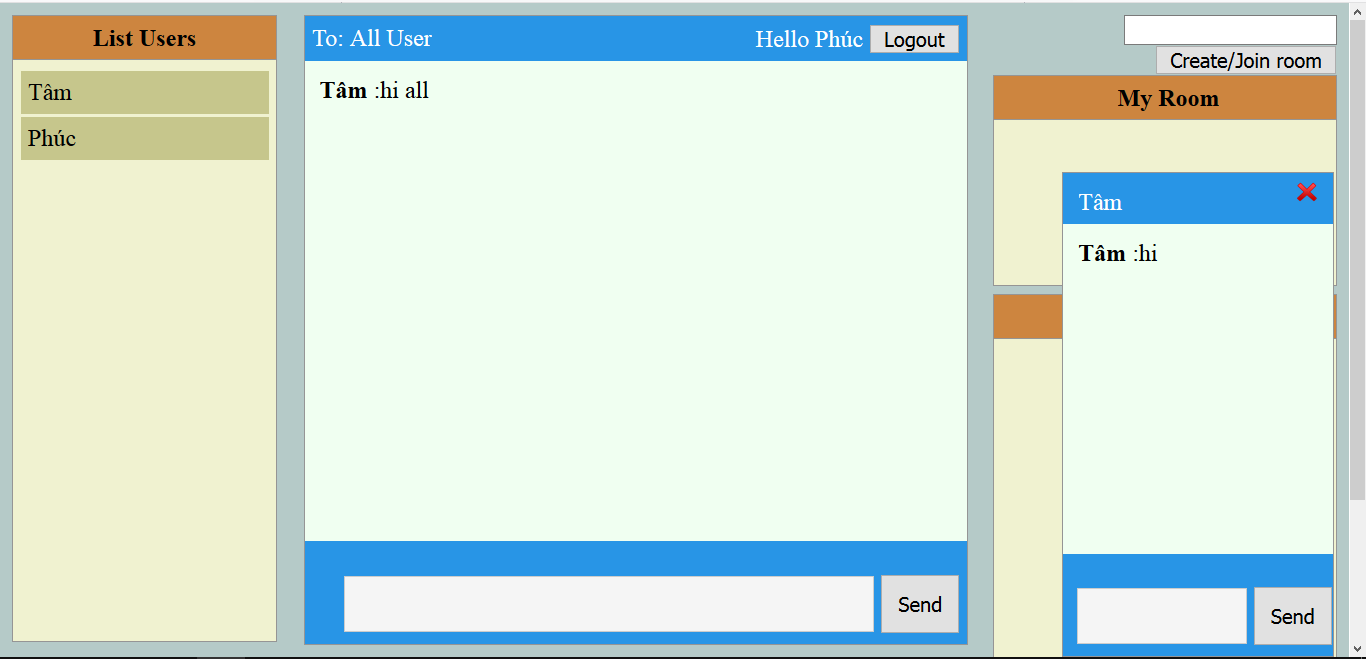
**Lưu ý**: việc gửi tin nhắn all này không cần quan tâm đến socket#id của tất cả người dùng, chỉ cần thông tin socket#id của người gửi để config giao diện cho tất cả người dùng. Vì thế không cần dùng tới hàm to(), chỉ cần dùng hàm io.sockets.emit().



*Code xử lý bên server*



*Hình ảnh minh họa: giao diện người gửi*

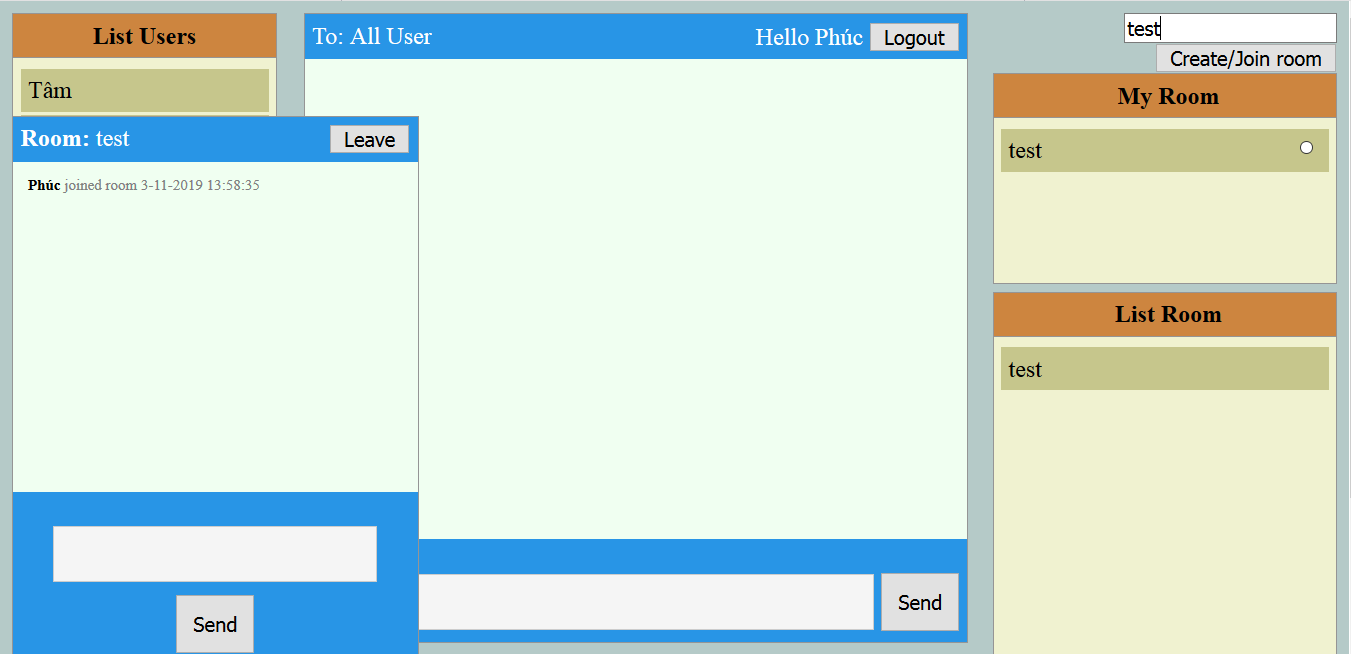


*Hình ảnh minh họa: giao diện người nhận*

**3.4 Rooms**

Để tạo room chúng ta nhập vào textbox tên room kế bên nút Create/Join room. Khi enter hay nhấn nút Create/Join room, client sẽ tạo một sự kiện bao gồm thông tin room cần tạo, người tạo đến server, khi đó chúng ta có thể config socket#id của room đã tạo.

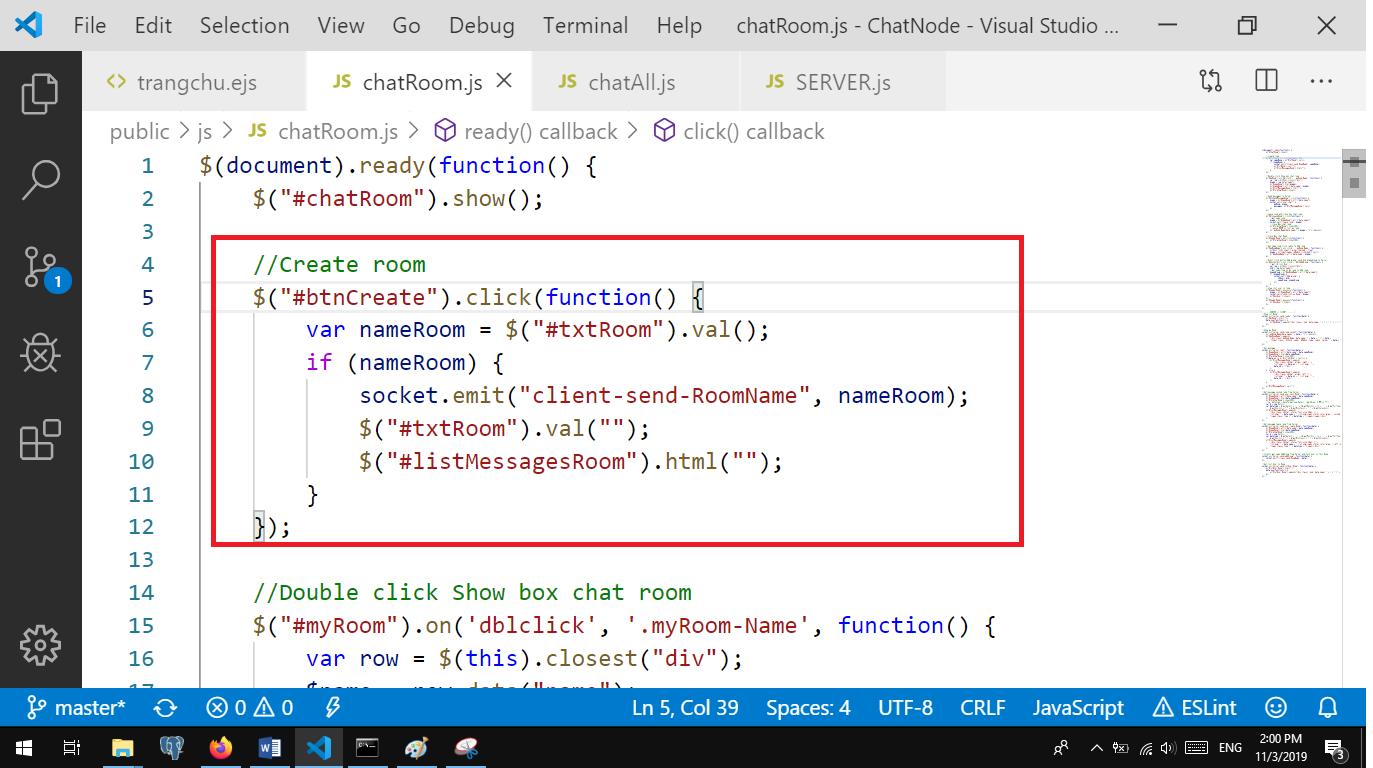
Ở ứng dụng này, config cho socket#id của room cần tạo chính là thông tin được nhập vào ô textbox kế bên nút Create/Join room, nên nhớ mọi chức năng gửi tin nhắn đều thông qua socket#id



*Hình ảnh minh họa: giao diện người dùng*

Khi tạo room, hay tham gia room, server sẽ broadcast một tin nhắn đến tất cả thành viên trong room là có người tham gia.

Khi rê chuột vào tên room, chúng ta có thể thấy tên của các thành viên có mặt trong room, người dùng bên ngoài có thể tham gia room khi được mời hoặc tự động gõ vào textbox tên room cần tham gia.



*Hình ảnh minh họa: sự kiện “client-send-RoomName” được gửi đi khi nhấn button create room*

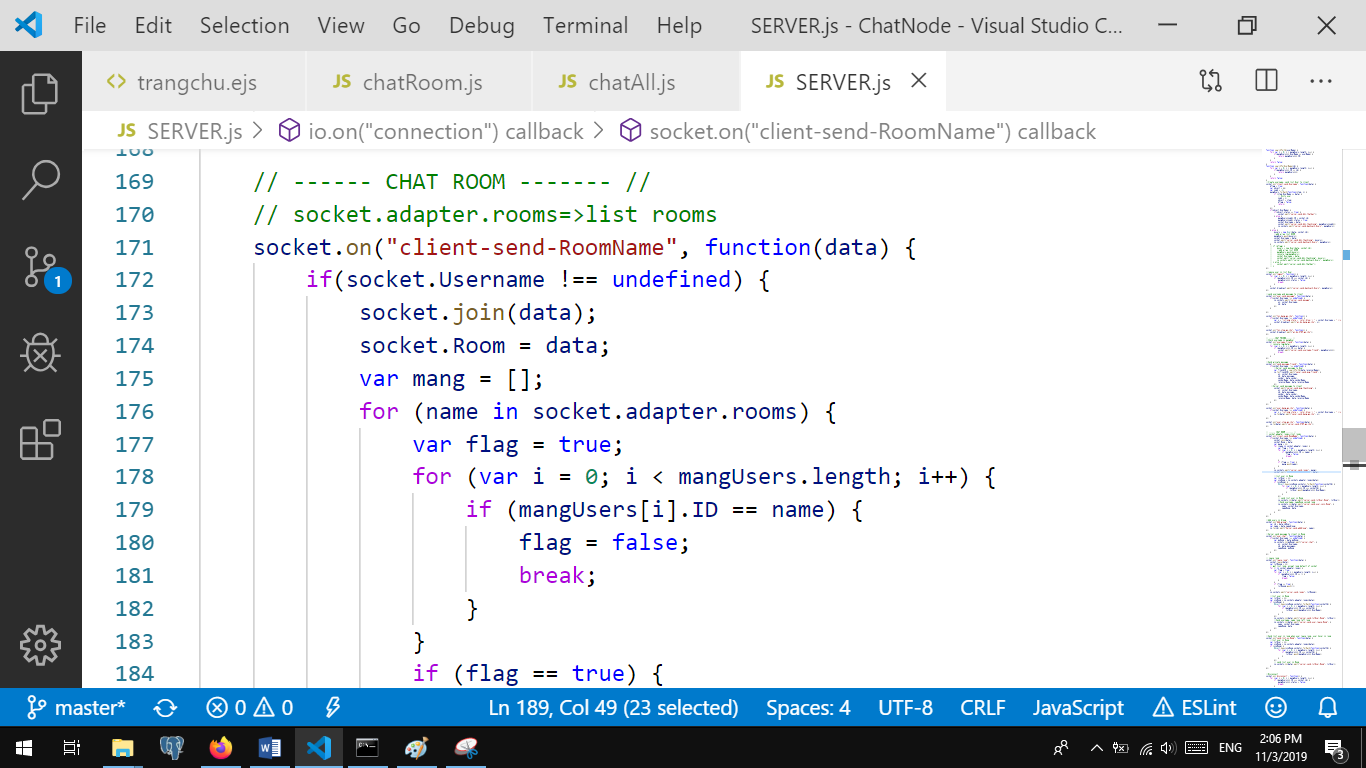
Ở đây khi tạo room, client sẽ gửi đến server qua hàm emit() với hai tham số, một là tên sự kiện “client-send-RoomName”, thứ hai là tên room được lấy textbox qua cú pháp $(“#txtRoom”).val();

Client gửi đến server một sự kiện có tên “client-send-RoomName”, server nhận được xử lý tương tự như tạo tài khoản register, kiểm tra xem tên phòng nhập đã tồn tại chưa. Nếu tồn tại thì auto join vào room, còn nếu chưa thì tạo mới một phòng có tên vừa được nhập.

Khi join vào room, socket.io hỗ trợ hàm có tên gọi là join(). Hàm join() chỉ có một tham số duy nhất

* đó là tên room cần tham gia: socket.join(“tên room”).

Khi đã tham gia mọi thông tin tin nhắn người dùng gửi sẽ không còn gửi đến socket.id riêng lẽ của từng người, mà gửi đến socket.id của room mà người dùng tham gia, từ đó mọi client tham gia đến room sẽ nhận được tin nhắn

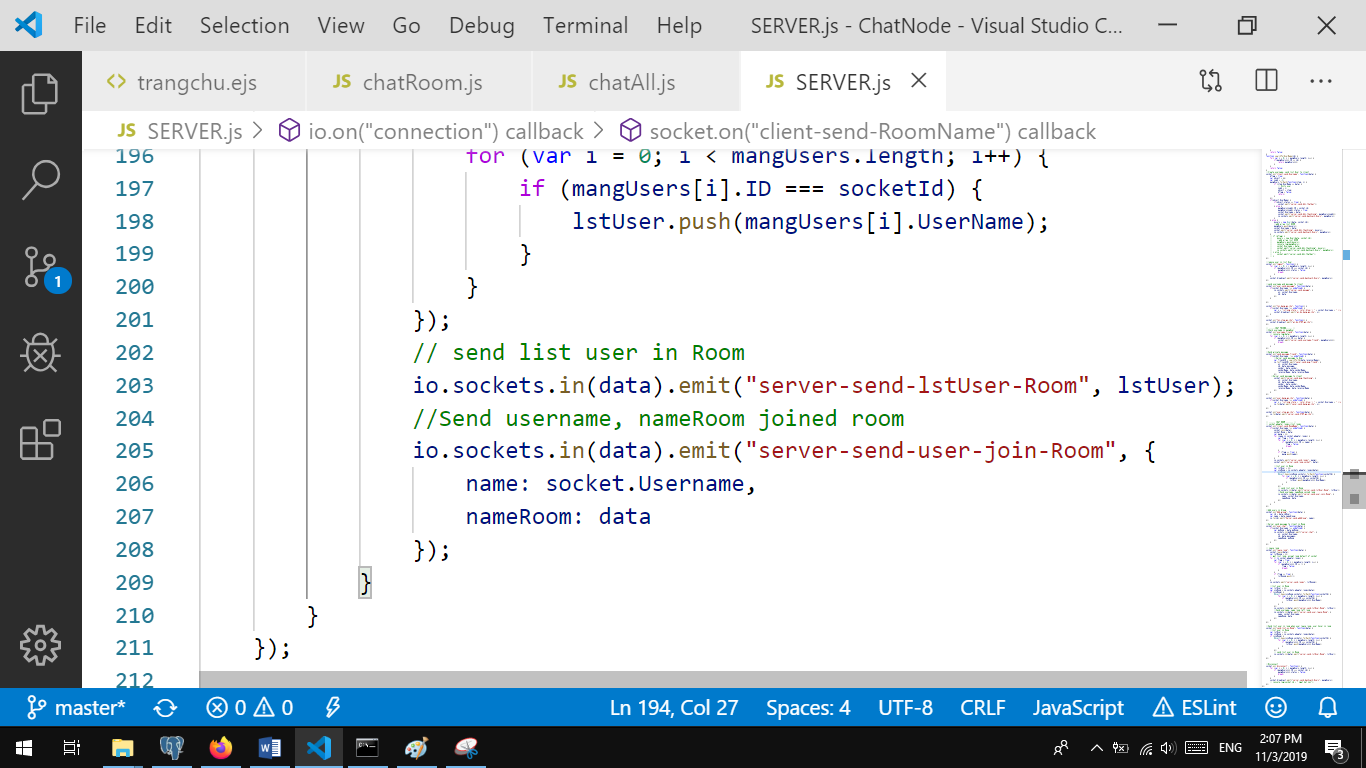


*Hình ảnh minh họa: xử lý sự kiện “client-send-RoomName” ở client*



*Hình ảnh minh họa: sự kiện server-send-room-socket*

Hàm io.sockets.adapter.rooms sẽ trả về kết quả dưới dạng mảng object, mỗi phần tử sẽ là một object chứa key là địa chỉ socket.id của room tồn tại, và value chính là tên room đã có sẵn.

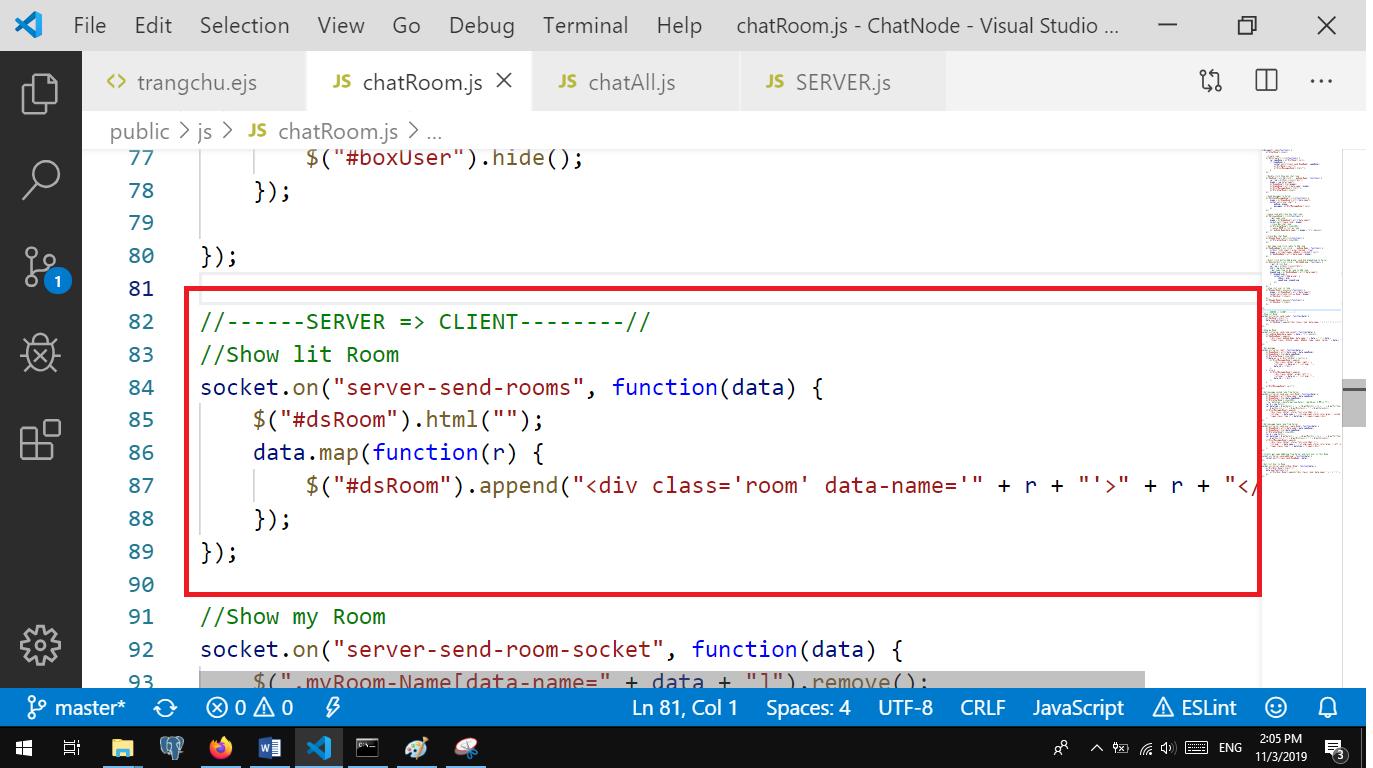


*Hình ảnh minh họa: Hàm in() ở phía server.*

Cũng giống như hàm to(), hàm in() gồm một tham số

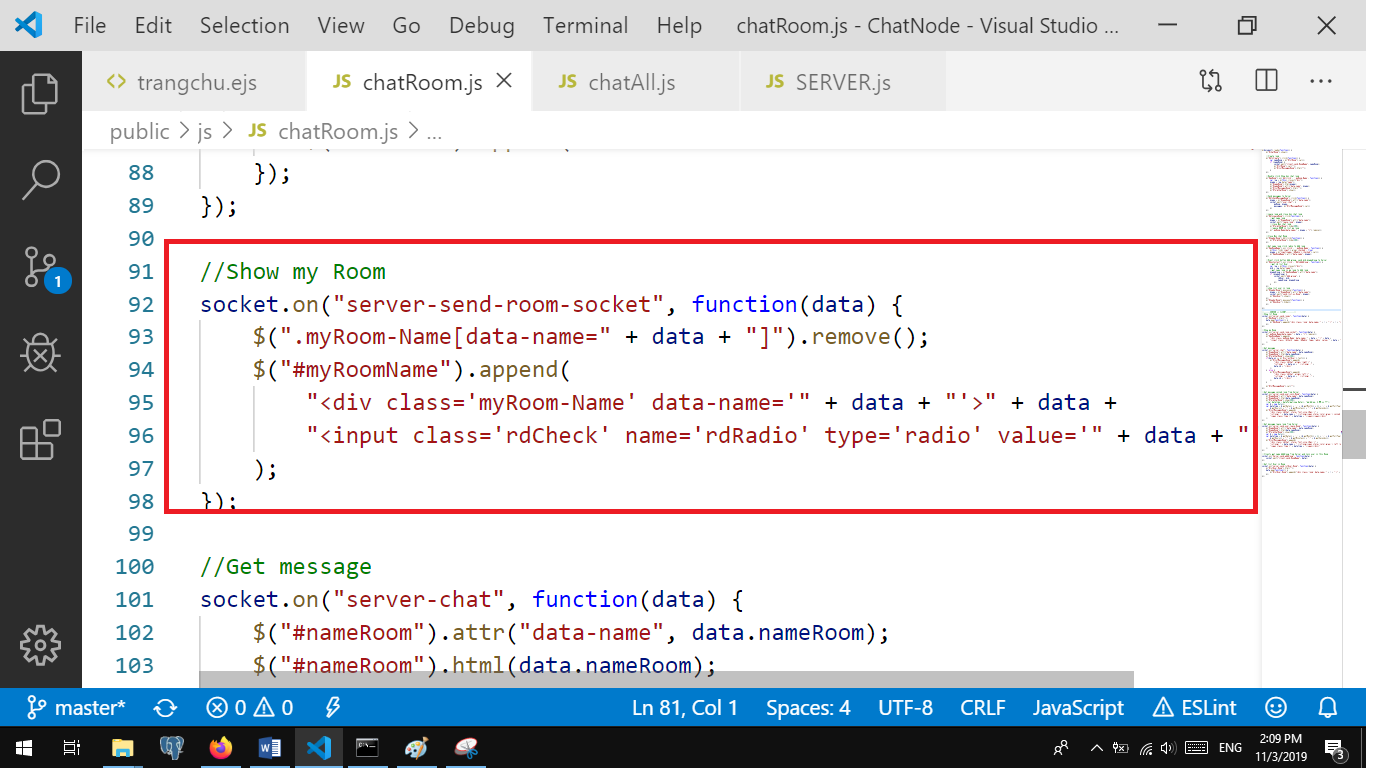
* đó là địa chỉ socket.id của tên room cần gửi

Tính năng của hàm in() cũng tương tự như hàm to() dùng để gửi tin nhắn riêng biệt đến đối tượng chỉ định mà không phải broadcast. Tuy nhiên, hàm to() là gửi tin nhắn tới duy nhất một đối tượng, nếu đối tượng đó không tồn tại, sẽ không có gì xảy ra, tin nhắn sẽ không được gửi, còn hàm in() khi đối tượng đó không tồn tại, nó sẽ gửi broadcast đến tất cả các đối tượng đang kết nối tới server, ngoại trừ người gửi.

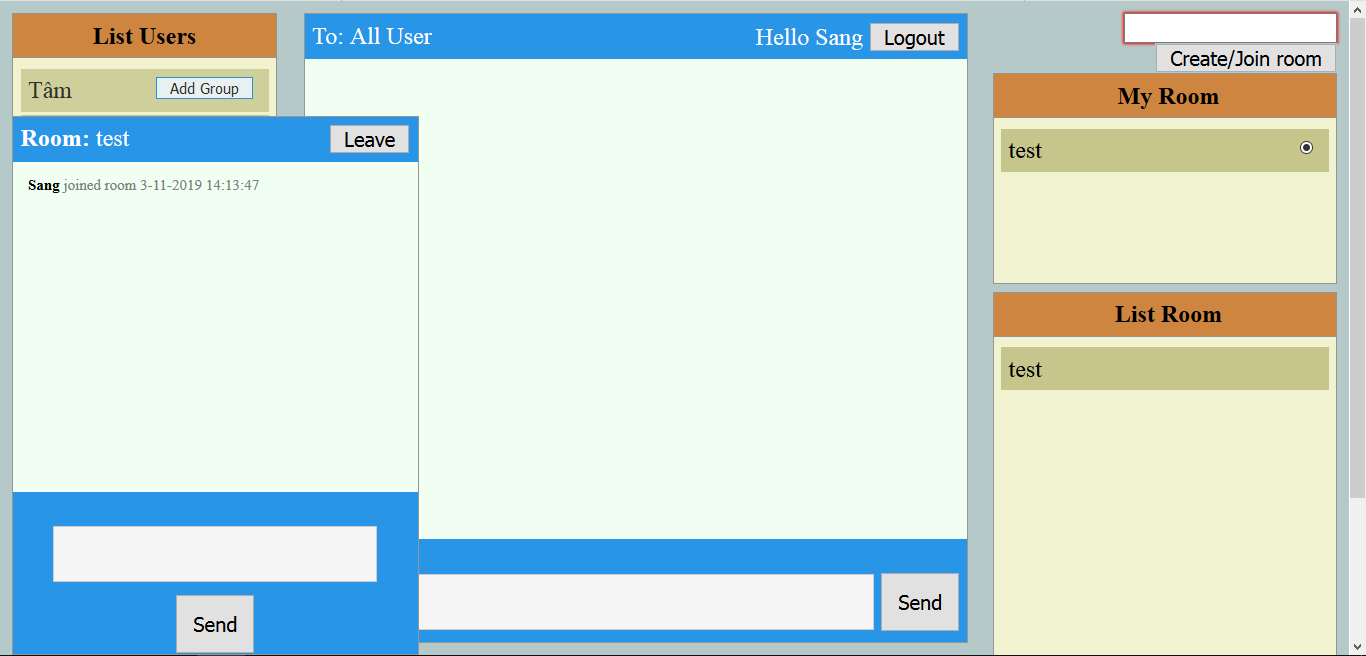


*Hình ảnh minh họa: sự kiện “server-send-rooms” ở client*

Khi người dùng tham gia vào room, server sẽ gửi đến client tín hiệu “server-send-room-socket” để hiện lên danh sách các room mà người dùng đó đang tham gia. Việc làm này sẽ giúp người dùng dễ quản lý các room của mình và cho phép mời các thành viên khác tham gia room.

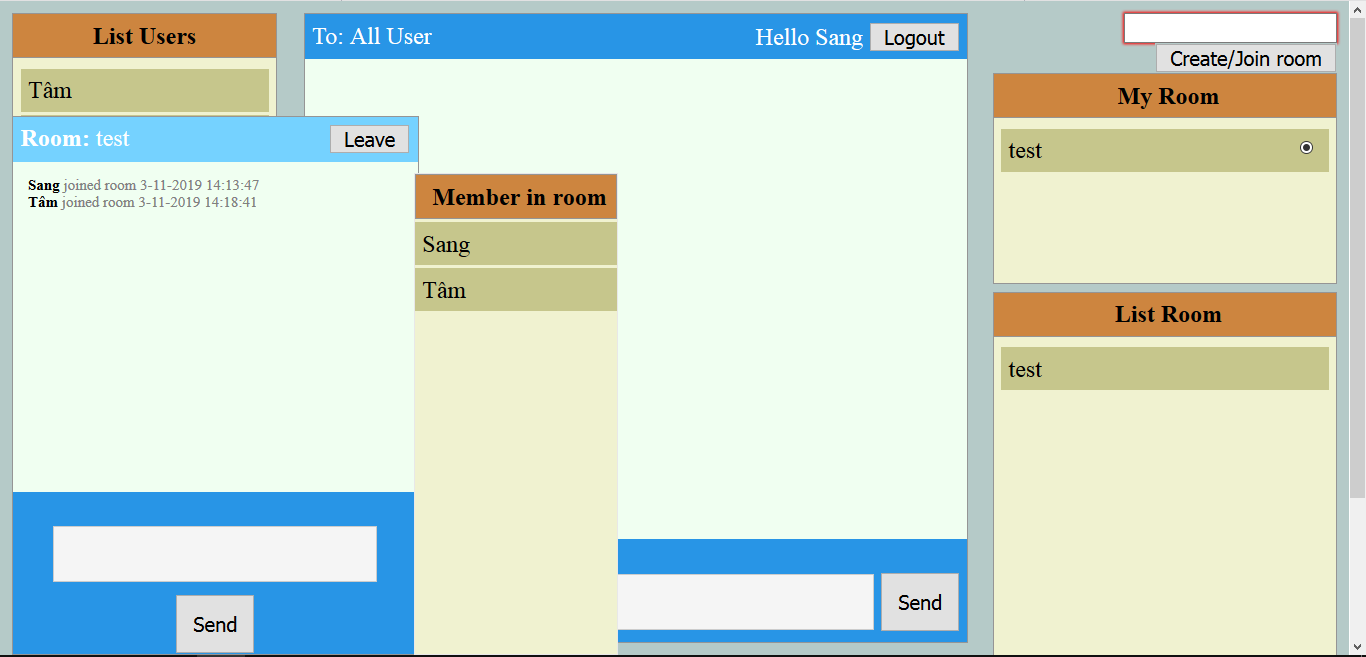


Để mời người khác tham gia room, chỉ cần rê chuột đến tên room mình tham gia trong my room, chọn vào radio button, sau đó rê chuột đến List User chọn user mà mình muốn người.



*Hình ảnh minh họa: giao diện người dùng.*

Khi mời người dùng, server sẽ xử lý xem người dùng đó có tồn tại trong group chưa, nếu chưa thì sẽ add người dùng vào group và gửi tin nhắn broadcast đến tất cả các thành viên trong nhóm là người dùng ấy đã tham gia group. Tính năng này giúp quản lý người dùng có trong group, cho các người dùng khác biết sự tham gia của người khác.



*Hình ảnh minh họa: giao diện người dùng*

Việc gửi tin nhắn đến group, được xử lý tương tự như gửi tin nhắn message, thay vì gửi tin nhắn đến socket#id của người nhận, ở đây sẽ gửi message đến socket#id của room được tạo.

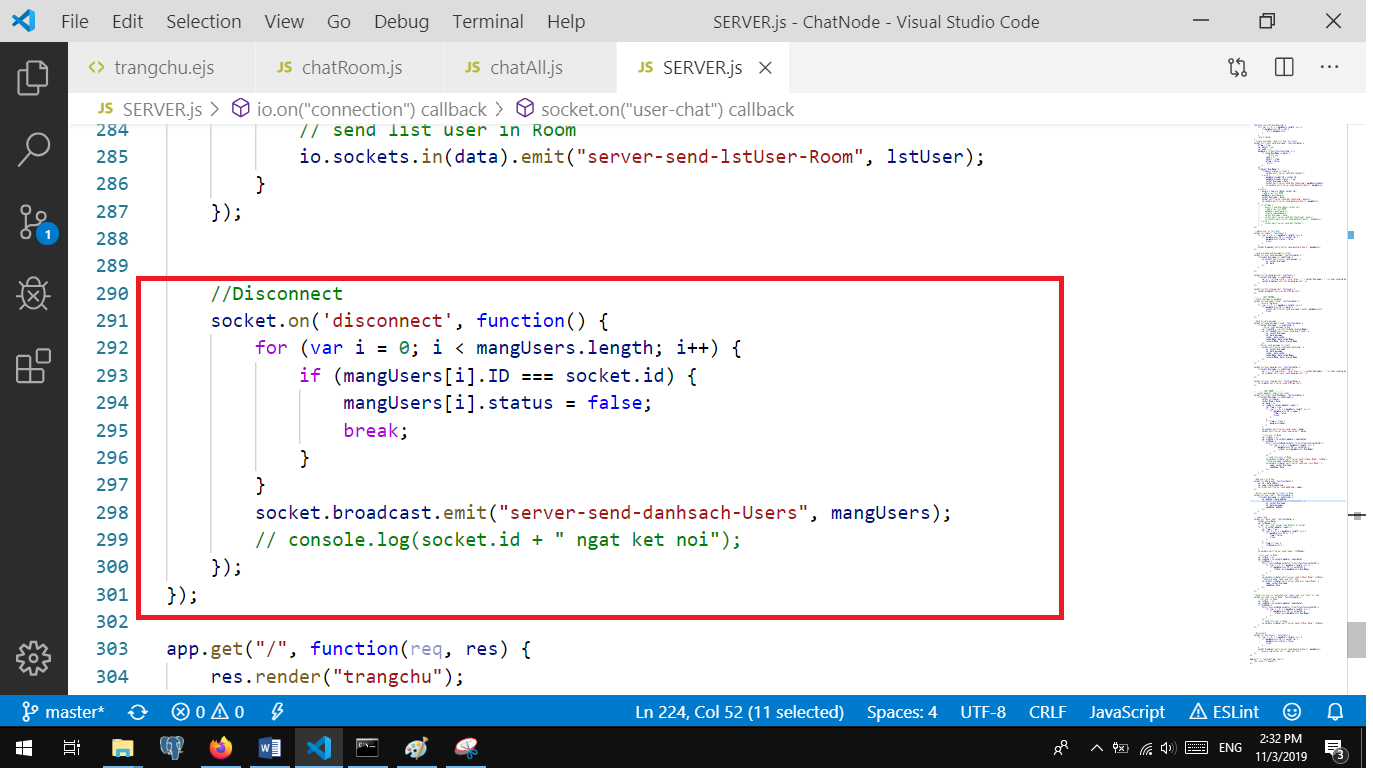
Server sẽ nhận được tin nhắn khi nút send trong room được nhấn, khi đó server sẽ kiểm tra xem người gửi có tồn tại không, nếu không tồn tại thì việc gửi tin nhắn sẽ không thành công, ngược lại sẽ gửi tin nhắn đến room cho các thành viên qua sự kiện “server-chat” với các tham số là tên người gửi, nội dung tin nhắn và tên room.

Client sẽ hứng event “server-chat” và tạo tin nhắn hiển thị đến tất cả người dùng có trong room, việc xử lý gửi tin nhắn ngay tức khắc đến tất cả các thành viên.

**3.5 Disconnect**

Khi người dùng refresh lại page, quá trình disconnection sẽ diễn ra, phía server sẽ ngay lập tức nhận ra sự kiện disconnect. Khi đó, server sẽ duyệt mảng Users tìm tài khoản vừa disconnect đó và chỉnh trạng thái của tài khoản đó là offline, để dành phiên làm việc lần sau tài khoản đó đăng nhập không cần phải tạo mới lần nữa.

Ngay lập tức server sẽ broadcast đến tất cả các client có thành viên out, để cập nhật ngay lập tức List User có trên màn hình hiển thị.

*Hình ảnh minh họa: sự kiện disconnect ở server.*

Khác với connection, khi client disconnect, nhờ vào cơ chết heartbeat, server sẽ tự động nhận được sự kiện có tên là disconnect và giải quyết.

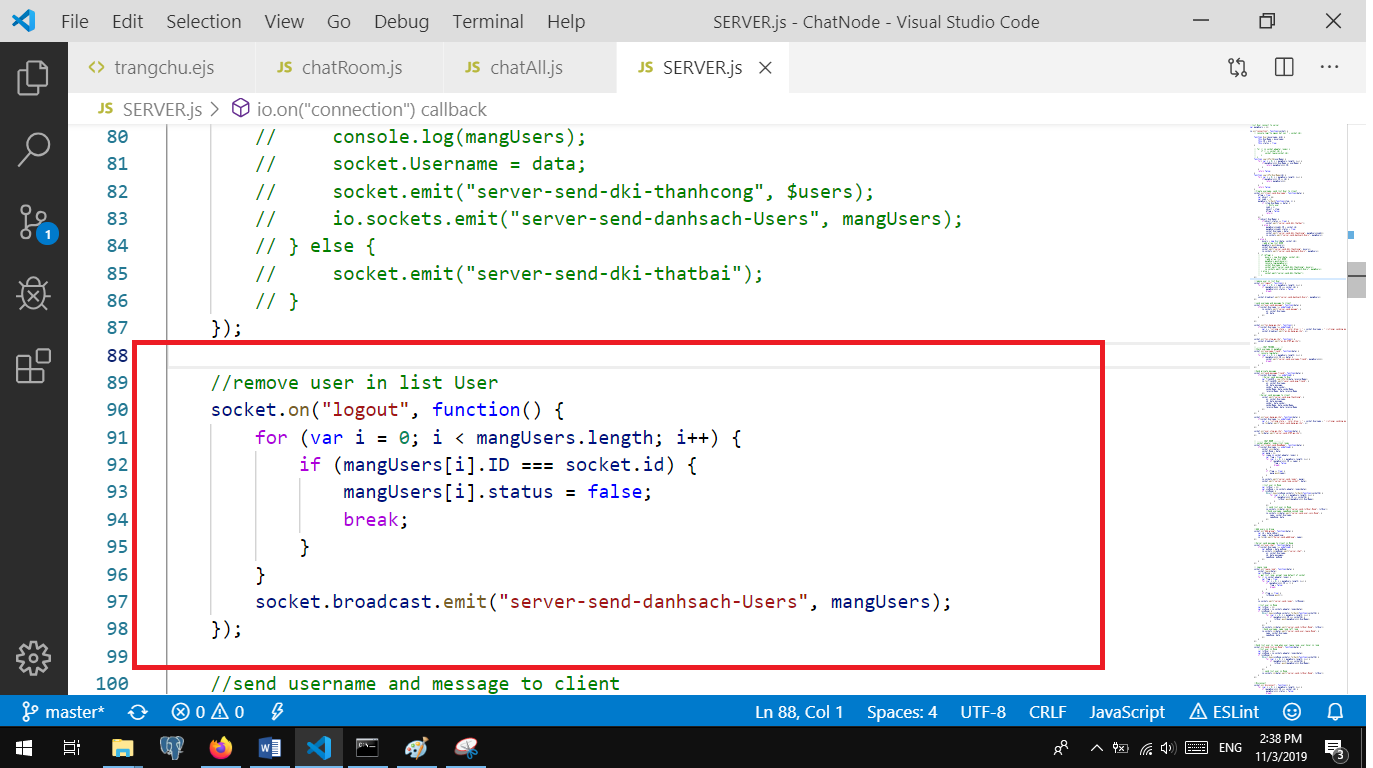
Hàm socket.broadcast.emit() có hai tham số bao gồm tên sự kiện và dữ liệu gửi đi tương tự như io.socket.emits() dùng để gửi sự kiện đến mọi client kết nối, ngoài ra socket.broadcast.emit() còn gửi tới những người dùng vừa mới tạo, hoặc chính client, kẻ tạo nên sự kiện mất kết nối.

**3.6 Logout**

Cũng tương tự như disconnection server sẽ duyệt mảng Users tìm tài khoản vừa disconnect đó và chỉnh trạng thái của tài khoản đó là offline, để dành phiên làm việc lần sau tài khoản đó đăng nhập không cần phải tạo mới lần nữa.

Ngay lập tức server sẽ broadcast đến tất cả các client có thành viên out, để cập nhật ngay lập tức List User có trên màn hình hiển thị.

**Lưu ý**: disconnection khác với logout, logout là người dùng chủ động nhấn nút logout, còn disconnect là khi có sự cố xảy ra khi ngắt kết nối bất ngờ với trang web (connection timeout, mất mạng,refresh page..), sự kiện logout được xử lý khi nút logout được nhấn, ngược lại disconnect diễn ra một cách bất ngờ.



*Hình ảnh minh họa: sự kiện logout ở phía server*

**CHƯƠNG 4: KẾT LUẬN**

Qua quá trình nghiên cứu và áp dụng tạo ứng dụng web chat, chúng em học được nhiều cái mới về hay của công nghệ real-time này. Nó xử lý rất nhanh, ngay lập tức hiển thị cho người dùng mà không cần phải đợi gửi yêu cầu và nhận về từ phía server, cũng không cần phải phụ thuộc vào số lượng khủng database.

Nhưng mọi thứ đều có mặt trái của nó, qua quá trình tìm hiểu và triển khai, chúng em đã tổng kết được một số ưu điểm và nhược điểm của nó:

Ưu điểm:

* Thứ nhất phải nói đến đó là nhanh, cực kì nhanh, thông tin vừa được gửi và nhận cực kì nhanh bất kể khoảng cách, hay số lượng lớn data.
* Thuận tiện cho người dùng, chỉ một cú click và gửi người dùng và người gửi đều nhận được thông tin một cách chính xác, giảm đáng kể sự chờ đợi kết quả nhận được, được áp dụng trên mọi nền tảng trình duyệt web.
* Áp dụng rộng rãi hiện nay, có thể thấy công nghệ real-time socket.io được áp dụng ở mọi nơi như stream, game online, liên lạc từ xa, từ quân sự tới sinh hoạt người dân là thứ bắt buộc phải có, chỉ cần một cú click căn cứ quân sự và người lính đều phát hiện cái gì đang diễn ra. Bạn không thể cứ gọi điện thoại và chờ 2 phút thì nhận được giọng nói của người đang liên lạc.
* Hỗ trợ cho các công nghệ khác như deep-learning,… Chỉ cần một lần thoáng qua hình ảnh hoặc giọng nói là có thể nhận diện.
* Hiệu năng cực kì cao, ở các công nghệ cũ nếu 1 người gửi request thì tầm 15s trả về kết quả từ phía server như vậy nếu 1.000.000 người nhấn cùng lúc thì lượng thời gian xử lý trả về sẽ là 15.000.000s tức 4.167 giờ có thể dẫn đến die server, đó là chưa kể đến số lượng database khổng lồ. Với công nghệ real-time, mọi thứ được diễn ra ngay lập tức chỉ bằng một cú click.

Nhược điểm:

* Phía server phải cực kì mạnh, tối ưu hóa để có tải trọng số lượng lớn người dùng tương tác liên tục.
* Tốn cực lớn chi phí để config phía server lưu thông tin database số lượng khủng khi mọi thứ được diễn ra bằng một cú click.
* Nếu mọi khách truy cập duy trì kết nối websocket liên tục đến máy chủ, điều đó gây lãng phí hơn so với thực hiện một vài yêu cầu ajax trong suốt phiên truy cập. Ví dụ một số ứng dụng nho chỉ cần một ít thông tin ngay tại thời điểm hiện tại mà không cần cập nhật liên tục, dẫn đến việc lãng phí băng thông

**TÀI LIỆU THAM KHẢO**

<https://socket.io/docs/>

<https://viblo.asia/p/co-ban-ve-socketio-bJzKm0kY59N>

<https://viblo.asia/p/websocket-la-gi-hieu-ro-ve-websocket-1Je5E4BmlnL>

<https://viblo.asia/p/websocket-la-gi-Ljy5VxkbZra>

<https://viblo.asia/p/co-ban-ve-giao-thuc-websocket-va-thu-vien-socketio-63vKjmmM52R>

<https://viblo.asia/p/nodejs-va-socketio-can-ban-jlA7GKxdvKZQ>

<https://techtalk.vn/xay-dung-app-don-gian-voi-nodejs-expressjs-va-socket-io.html>

<https://kipalog.com/posts/Tim-hieu-ve-lap-trinh-socket---Buoc-dau-lam-quen>

<https://techmaster.vn/posts/34415/hieu-ve-cong-nghe-real-time-server-sent-events-vs-websocket>

<https://www.it-swarm.net/vi/node.js/su-khac-nhau-giua-socket.io-va-websockets/1067384250/>

<http://itplus-academy.edu.vn/ket-noi-thoi-gian-thuc-la-gi-va-nhung-dac-diem-noi-bat-cua-chung-1954.html>

<https://www.educba.com/websocket-vs-socket-io/>

<https://stackoverflow.com/questions/10112178/differences-between-socket-io-and-websockets>

<https://blog.tinohost.com/socket-la-gi/>

<https://vietpro.net.vn/node-js-26-tong-quan-ve-socket-io.html>

<https://vietpro.net.vn/node-js-30-socket-io-namspace.html>

<https://vietpro.net.vn/node-js-29-socket-io-broadcasting.html>

<https://davidwalsh.name/websocket>