**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**

CHƯƠNG 1: GIỚI THIỆU SOCKET.IO 1

1.1 Khái Niệm Real-Time 1

1.2 Khái niệm Socket 2

1.2.1 Tại sao người dùng lại cần đến socket? 2

1.2.2 Socket hoạt động như thế nào? 2

1.2.3 Phân loại Socket 3

a) Stream Socket 3

b) Datagram Socket 4

1.3 Khái niệm WebSocket 4

1.4 Socket.IO 6

1.5 Các tính năng chính của Socket.IO 7

1.5.1 Reliability 7

1.5.2 Auto-reconnection support 7

1.5.3 Disconnection detection 7

1.5.4 Binary support 7

1.5.5 Multiplexing support 7

1.5.6 Room support 8

1.6 Cấu hình Config để sử dụng Socket.IO 8

1.7 Tại sao nên chọn Socket.IO 9

CHƯƠNG 2: CƠ CHẾ HOẠT ĐỘNG CỦA SOCKET.IO 10

2.1 Khai báo sử dụng socketio 10

2.2 Cơ chế lắng nghe, truyền dữ liệu của Socket.IO 11

2.3 Socket.IO Broadcasting 14

CHƯƠNG 3: SO SÁNH SOCKET.IO VỚI WEBSOCKET 16

CHƯƠNG 4: ROOMS AND NAMESPACES 18

4.1 Default namspace 18

4.2 Custom namespace 19

4.3 Rooms 19

4.3.1 Joining and Leaving 19

4.4 Default Room 20

4.5 Disconnection 20

4.6 Gửi tin nhắn từ bên ngoài 20

TÀI LIỆU THAM KHẢO 21

**CHƯƠNG 1: GIỚI THIỆU SOCKET.IO**

* 1. **Khái niệm real-time**

Hãy thử tưởng tượng bạn phải đợi đồng nghiệp gửi một e-mail và phải liên tục tải lại trang để kiểm tra thư đã đến hay chưa. Chả phải tốt hơn khi hệ thống sẽ tự động thông báo có thư mới cho bạn khi bạn đã nhận được thư hay sao? Hiện nay, với sự trợ giúp của công nghệ kết nối thời gian thực(real-time), những thông tin mới luôn được cập nhật gần như ngay lập tức và các lập trình viên, nhất là lập trình ứng dụng đang cần tự update cho mình những thông tin về cách xây dựng hệ thống này cho các dự án của mình.

Hầu hết các website hiện nay đang sử dụng HTTP, giúp xử lý các yêu cầu khi có một người dùng truy cập vào URL và gửi đi các lệnh đến server. Server sau đó nhập yêu cầu và chuyển dữ liệu tới người dùng. Và ngay sau khi vận chuyển hết dữ liệu tới người dùng, server sẽ ngắt kết nối.

Đây chính là vấn đề chúng ta cần bàn tới. Mặc dù nó vận hành khá ổn với một vài loại website, nhưng thuật toán này lại gây ra vấn đề về trải nghiệm người dùng khi sử dụng các website cần sự tương tác liên tục như chatbox, trao đổi công việc hay truyền tải các file tới nhiều bộ phận khác nhau trong cùng một server. Chính vì vậy nhu cầu cần tạo  ra một nền tảng giúp việc vận chuyển dữ liệu được diễn ra liên tục khiến cho các nhà phát triển công nghệ tạo ra một loại kết nối giúp các lập trình viên ứng dụng vào công việc của mình: Kết nối thời gian thực(real-time).

Realtime ám chỉ răng website có thể phản hồi và tương tác người dùng một cách tức thì mà người dùng không cần chờ đợi lâu hoặc refresh lại ứng dụng hoặc trình duyệt. Chúng ta có thể nhìn thấy realtime ở khắp mọi nơi: thực tế nhất chính là qua các ứng dụng nhắn tin, hoặc bảng tin newsfeed trên Facebook.

Một ứng dụng thời gian thực (RealTime Application) là một ứng dụng hoạt động trong một khoản thời gian mà người dùng có thể cảm nhận được ngay lập tức thời.

Một số ví dụ về cá ứng dụng thời gian thực:

* **Instant messengers** – Các ứng dụng chat, trò chuyện như Whatsapp, Facebook Messenger, v.v. Bạn sẽ không cần phải làm mới ứng dụng/website của mình để nhận được các nội dung chat mới.
* **Push Notifications** – Khi một ai đó tags bạn vào một bức ảnh trên Facebook, ban sẽ nhận được a thông báo ngay lập tức.
* **Collaboration Applications** – Các ứng dụng như Google docs, cho phép nhiều người cập nhật cùng một tài liệu cùng lúc và áp dụng thay đổi cho tất cả các trường hợp của mọi người.
* **Game Online** – Các trò chơi như Liên Minh Huyền Thoại, PUBG, Đột Kích, FIFA ONLINE 4 v.v , cũng là một số ví dụ về ứng dụng thời gian thực.
  1. **Khái niệm Socket**

[Socket](https://en.wikipedia.org/wiki/Socket) là một điểm cuối (end-point) của liên kết truyền thông hai chiều (two-way communication) giữa hai chương trình chạy trên một môi trường mạng. Các lớp Socket được sử dụng để biểu diễn kết nối giữa client và [server](https://blog.tinohost.com/server-va-cac-loai-may-chu-pho-bien/),  được ràng buộc với một cổng port (thể hiện là một con số cụ thể) để các tầng TCP (TCP Layer) có thể định danh ứng dụng mà dữ liệu sẽ được gửi tới.

Lập trình socket là lập trình cho phép người dùng kết nối các máy tính truyền tải và nhận dữ liệu từ máy tính thông qua mạng

Hiểu đơn giản, socket là thiết bị truyền thông hai chiều gửi và nhận dữ liệu từ máy khác.

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

Trong quá trình làm việc các bạn có thể chạy nhiều socket cùng một lúc nên công việc của bạn sẽ nhanh hơn, nâng cao hiệu suất làm việc.

Ngày nay, Socket được hỗ trợ trong hầu hết các hệ điều hành như MS Windows, Linux và được sử dụng trong nhiều ngôn ngữ lập trình khác nhau: như C, C++, Java, Visual Basic, Visual C++,…

### **1.2.2 Socket hoạt động như thế nào?**

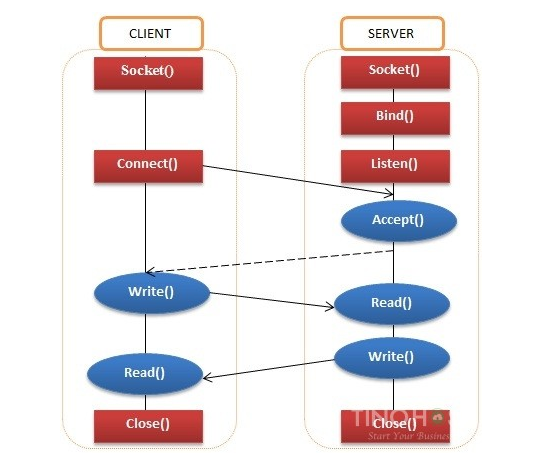
Là một giao diện lập trình ứng dụng mạng, socket giúp các bạn lập trình kết nối các ứng dụng để truyền và nhận giữ liệu trong môi trường có kết nối Internet bằng cách sử dụng phương thức TCPIP và UDP.

Khi cần trao đổi dữ liệu cho nhau thì 2 ứng dụng cần phải biết thông tin tối thiểu là IP và số hiểu cổng của ứng dụng kia.  
+ 2 ứng dụng có thể nằm cùng trên một máy  
+ 2 ứng dụng cùng nằm trên một máy không được cùng số hiệu cổng

### **1.2.3 Phân loại Socket**

#### **a) Stream Socket**

Dựa trên giao thức TCP( Tranmission Control Protocol), việc truyền dữ liệu chỉ thực hiện giữa 2 quá trình đã thiết lập kết nối. Do đó, hình thức này được gọi là **socket hướng kết nối.**

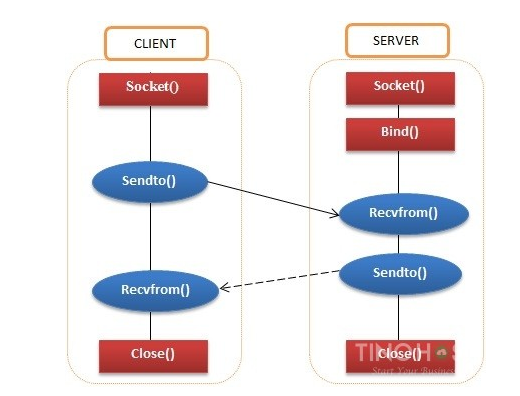


**Ưu điểm:** Có thể dùng để liên lạc theo mô hình client và sever. Nếu là mô hình client /sever thì sever lắng nghe và chấp nhận từ client. Giao thức này đảm bảo dữ liệu được truyền đến nơi nhận một cách đáng tin cậy, đúng thứ tự nhờ vào cơ chế quản lý luồng lưu thông trên mạng và cơ chế chống tắc nghẽn. Đồng thời, mỗi thông điệp gửi phải có xác nhận trả về và các gói tin chuyển đi tuần tự.

**Hạn chế**: Có một đường kết nối (địa chỉ IP) giữa 2 tiến trình nên 1 trong 2 tiến trình kia phải đợi tiến trình kia yêu cầu kết nối.

#### **b)  Datagram Socket**

Dựa trên giao thức UDP( User Datagram Protocol) việc truyền dữ liệu không yêu cầu có sự thiết lập kết nối giữa 2 quá trình. Do đó, hình thức này được gọi là **socket không hướng kết nối.**



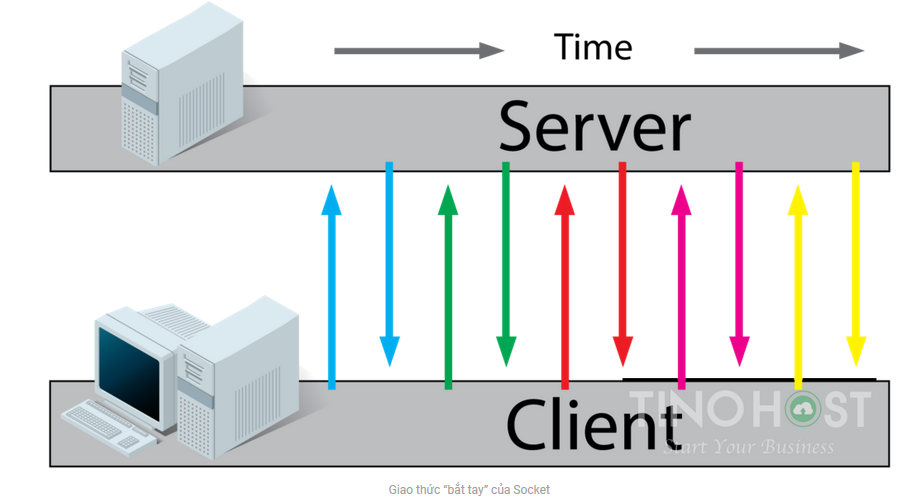
**Ưu điểm:** Do không yêu cầu thiết lập kết nối, không phải có những cơ chế phức tạp nên tốc độ giao thức khá nhanh, thuận tiện cho  các ứng dụng truyền dữ liệu nhanh như chat, game…..

**Hạn chế:** Ngược lại với giao thức TCP thì dữ liệu được truyền theo giao thức UDP không được tin cậy, có thế không đúng trình tự và lặp lại.

* 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 tạo một kết nối hiệu quả và ít tốn kém. Mặc dù được thiết kế để chuyên sử dụng cho các ứng dụng web, lập trình viên vẫn có thể đưa chúng vào bất kì loại ứng dụng nào.

* WebSockets mới xuất hiện trong HTML5, là một kỹ thuật Reverse Ajax. WebSockets cho phép các kênh giao tiếp song song hai chiều và hiện đã được hỗ trợ trong nhiều trình duyệt (Firefox, Google Chrome và Safari). Kết nối được mở thông qua một HTTP request (yêu cầu HTTP), được gọi là liên kết WebSockets với những header đặc biệt. Kết nối được duy trì để bạn có thể viết và nhận dữ liệu bằng JavaScript như khi bạn đang sử dụng một TCP socket đơn thuần.
* Dữ liệu truyền tải thông qua giao thức HTTP (thường dùng với kĩ thuật Ajax) chứa nhiều dữ liệu không cần thiết trong phần header. Một header request/response của HTTP có kích thước khoảng 871 byte, trong khi với WebSocket, kích thước này chỉ là 2 byte (sau khi đã kết nối). Vậy giả sử bạn làm một ứng dụng game có thể tới 10,000 người chơi đăng nhập cùng lúc, và mỗi giây họ sẽ gửi/nhận dữ liệu từ server. Hãy so sánh lượng dữ liệu header mà giao thức HTTP và WebSocket trong mỗi giây:
* HTTP: 871 x 10,000 = 8,710,000 bytes = 69,680,000 bits per second (66 Mbps)
* WebSocket: 2 x 10,000 = 20,000 bytes = 160,000 bits per second (0.153 Kbps) Như bạn thấy chỉ riêng phần header thôi cũng đã chiếm một phần lưu lượng đáng kể với giao thức HTTP truyền thống.



### **Ưu điểm**

WebSockets cung cấp khả năng giao tiếp hai chiều mạnh mẽ, có độ trễ thấp và dễ xử lý lỗi.

API cũng rất dễ sử dụng trực tiếp mà không cần bất kỳ các tầng bổ sung nào, so với Comet, thường đòi hỏi một thư viện tốt để xử lý kết nối lại, thời gian chờ timeout, các Ajax request (yêu cầu Ajax), các tin báo nhận và các dạng truyền tải tùy chọn khác nhau (Ajax long-polling và jsonp polling).

Không cần phải có nhiều kết nối như phương pháp Comet long-polling.

### **Hạn chế**

Là một đặc tả mới của HTML5, WebSocket vẫn chưa được tất cả các trình duyệt hỗ trợ.  
Do WebSocket là một TCP socket chứ không phải là HTTP request, nên không dễ sử dụng các dịch vụ có phạm vi-yêu cầu, như SessionInViewFilter của Hibernate.

* 1. **Socket.IO**

Trước tiên, Socket là 1 công nghệ. Đừng nhầm lẫn giữa [Socket.IO](http://Socket.IO) và Socket. [Socket.IO](http://Socket.IO) không phải là mô hình Socket duy nhất hiện nay, và cũng không phải là mô hình web socket duy nhất hiện nay. Socket là cách bạn tổ chức mô hình client-server để một trong 2 bên luôn trong tình trạng sẵn sàng trả lời bên kia và ngược lại. Để đảm bảo việc này, kết nối giữa Client và Server phải ở trạng thái “keep-alive” và phải luôn xảy ra quá trình đồng bộ giữa Client-Server. Socket sẽ mang lại khả năng trả lời tức thì từ một trong 2 bên khi bên kia đưa ra một sự kiện, thay vì phải thực thi lại một loạt các thủ tục kết nối phức tạp như trước, và ứng dụng của bạn sẽ trở thành ứng dụng thời gian thực ví dụ: Yahoo Messenger, Skype v.v… đều là các ứng dụng được xây dựng theo mô hình Socket.

Socket.IO là một thư viện Javascript cho các ứng dụng web realtime (web thời gian thực).

Nó cho phép giao tiếp hai chiều theo thời gian thực giữa các máy khách và máy chủ web.

Nó có hai phần client-side library hoạt động trên trình duyệt, và server-side library cho node.js.

Socket.io 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](http://asp.net), nodejs,...

* 1. **Các tính năng chính của Socket.IO**
     1. **Reliability**

Dựa vào Engine.IO. Engine.IO thiết lập kết nối bỏ phiếu dài, sau đó cố gắng nâng cấp lên các băng thông vận chuyển dữ liệu tốt hơn nhanh hơn.

Ưu điểm:

* + Cải thiện performance phía server
  + Cải thiện trải nghiệm của người dùng

Nhược điểm:

* Một số proxies block websocket
* Personal firewall và các phần mềm diệt virus đôi khi block websocket

### **Auto-reconnection support**

### Khi máy chủ server chết, các máy phía client sẽ cố gắng kết nối tới máy chủ không ngừng nghỉ, cho tới khi máy chủ server hoạt động trở lại bình thường.

### Các tùy chỉnh cho việc reconnection:

### reconnectionAttempts

### reconnectionDelay

### **Disconnection detection**

### Một cơ chế heartbeat được triển khai ở tầng Engine.IO, cho phép các máy phía client, và máy phía server biết khi nào máy kia không phản hồi nữa.

### Chức năng đó được tính bằng các bộ định thời gian được đặt trên cả máy chủ và máy khách, với các giá trị timeout (tham số pingInterval và pingTimeout) được chia sẻ trong quá trình kết nối handshake.

### **Binary support**

### Bất kỳ cấu trúc dữ liệu serializable nào đều có thể được hỗ trợ, bao gồm:

### [ArrayBuffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer) và [Blob](https://developer.mozilla.org/en-US/docs/Web/API/Blob) trong trình duyệt.

### [ArrayBuffer](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/ArrayBuffer) và [Buffer](https://nodejs.org/api/buffer.html) trong nodejs.

### **Multiplexing support**

### Socket.IO cho phép tạo ra nhiều Namespaces, hoạt động như các kệnh liên lạc riêng biệt(private), nhưng chia sẻ cùng một kết nối bên dưới.

### **Room support**

### Trong mỗi Namespace, có nhiều kênh khác nhau được gọi là Rooms, mà các sockets có thể tham gia và rời đi. Các Socket khi kết nối có thể gửi tin broadcast đến bất kỳ phòng nào, đến mọi sockets khác đã tham gia.

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

### Các tính năng này đi kèm với API đơn giản và thuận tiện, trông giống như sau:

### 

# Cấu hình Config để sử dụng Socket.IO

Cấu trúc một ứng dụng realtime sử dụng socket bao gồm 2 phần: phía server, phía client.

1. Phía server Đây là nơi sẽ cài đặt socket io. Ngôn ngữ để dựng server có thể là php, [asp.net](http://asp.net), nodejs,... Tuy nhiên, tùy vào ngôn ngữ lựa chọn mà cách cấu trúc server khác nhau. Ở đây, nếu được thì khuyến khích sử dụng nodejs để dựng server, vì như vậy có thể cài trực tiếp socketio vào cùng một server. Nếu sử dụng php thì phải cài thêm những package khác, hoặc phải chuẩn bị riêng server để chạy socketio.
2. Phía client: Ở phía client sẽ xây dựng giao diện người dùng. Ở đây có thể sử dụng js, hoặc các thư viện của js như jquery,... Nói chung là ngôn ngữ gì cũng được.
3. Những thứ cần chuẩn bị để sử dụng socketio

* PC: Mac hoặc Win đều được
* Download phần mềm socketio

1. Cài đặt socketio trên server nodejs Tạo thư mục Demo (Có thể đặt tên bất kỳ) Mở màn hình terminal (Mac)/ cmd (Win), cd đến thư mục Demo. Tại màn hình cmd, gõ câu lệnh: npm init; rồi nhấn enter. Khi đó hệ thống sẽ chạy và yêu cầu nhập tên dự án, nhập tên dự án bất kỳ. Ở các setting khác thì enter bỏ qua, khi hệ thống hiển thị Yes or No? thì gõ Yes để cài đặt.

Sau khi hoàn thành bước cài đặt ở trên, hệ thống sẽ tạo ra file package.json. Đây là file dùng để cài đặt cấu hình server. Tiếp theo, cần đặt những package cần thiết trên server để có thể làm web và ứng dụng realtime. Để cài những package này mở màn hình cmd > cd đến thư mục Demo > gõ dòng lệnh *npm install express ejs socketio* Khi cài đặt thành công, hệ thống sẽ tự tạo thư mục *nodemodules*

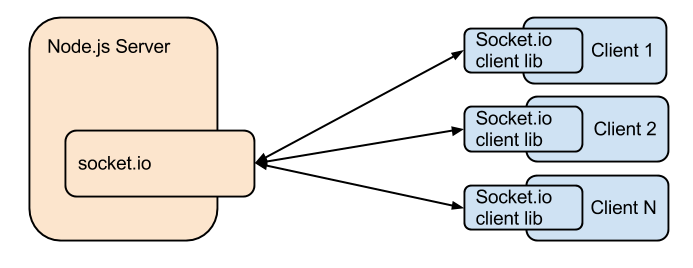
## 1.7 Tại sao nên chọn Socket.IO

* Socket.IO khá phổ biến, nó được sử dụng bới **Microsoft Office**, **Yammer**, **Zendesk**, **Trello** và nhiều tổ chức khác để xây dựng hệ thống thời gian thực mãnh mẽ.
* Nó là một trong những framework Javascript mãnh mẽ trên Github, và hầu hết phụ thuộc vào module NPM.
* Socket.IO cũng có một cộng đồng lớn, có nghĩa là việc tìm kiếm sự giúp đỡ rất dễ dàng.

# CHƯƠNG 2: CƠ CHẾ HOẠT ĐỘNG CỦA SOCKET.IO

### **2.1 Khai báo sử dụng Socket.IO**

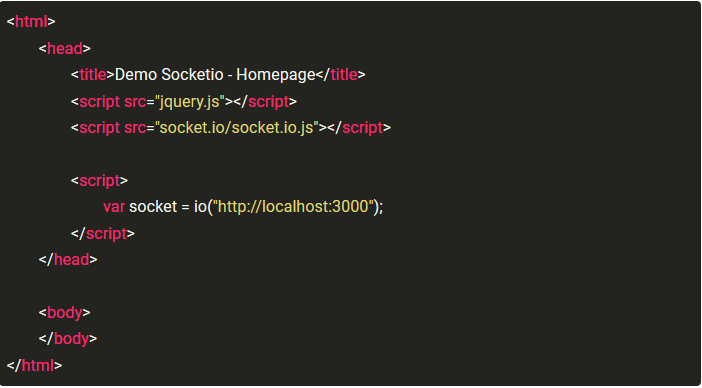
Cơ chế hoạt động của một ứng dụng realtime đó là thông qua server để lắng nghe (listen) data và truyền data về các máy client. Vì vậy cần cài khai báo sử dụng socketio ở cả phía server và client.



Code khai báo sử dụng socketio ở server:

### 

Code khai báo sử dụng socketio ở phía client



### **2.2 Cơ chế lắng nghe, truyền dữ liệu của Socket.IO**

Để lắng nghe data, ta sử dụng câu lệnh socket.on(), để phát dữ liệu thì sử dụng lệnh socket.emit() .

Ví dụ:

Client gửi 1 đoạn chat đi, thì khi đó ở phía server cần viết code để nhận dữ liệu đoạn code đó và truyền dữ liệu chat đó đi đến các server khác. Đồng thời ở ở phía client cũng cần viết code để gửi và nhận dữ liệu từ server.

Client sẽ nhận được dữ liệu thông thường là json khi server trả dữ liệu về.

Chú ý: socket.emit và socket.on luôn đi theo cặp

Code phía server

### 



### Code phía client

### 

### 

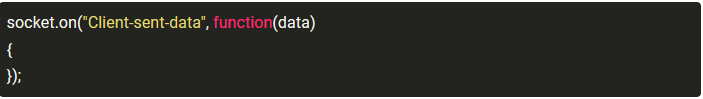
Lưu ý: socket.on và socket.emit có parameter thứ 1 là tên đường truyền. Tên đường truyền có thể là tên bất ký, tuy nhiên đễ truyền và nhận dữ liệu của chung 1 đường truyền thì tên đường truyền phải giống nhau.

Ví dụ: client gửi data bằng đường truyền có tên là Client-Sent-data, thì để nhận được data của đường truyền này thì server của phải khai báo tên đường truyền là Client-Sent-data trong lệnh socket.on.

Client send

### 

Server listen



Lưu ý: Khi sử dụng callback option, nên truyền data dưới dạng json, hoặc object, hoặc string

Khi emit một sự kiện ở phía client, không cần phương thức form action, tất cả đều được xử lý dưới js được viết theo kiểu callback function.

# 2.3 Socket.IO Broadcasting

Broadcasting nghĩa là gửi một tin nhắn đến tất cả người dùng đang kết nối. Broadcasting có thể thực hiện ở nhiều cấp độ. Chúng ta có thể gửi tin nhắn đến tất cả người dùng đang kết nối, khách hàng có tên bất kỳ và cả khách hàng ở trong một phòng cụ thể. Để phát một sự kiện cho tất cả các khách hàng, chúng ta có thể sử dụng phương thức io.sockets.emit

**Lưu ý**: Thao tác này sẽ phát ra sự kiện cho TẤT CẢ máy khách đang được kết nối.

Trong ví dụ này tôi sẽ phát ra sự kiện cho tất cả máy khách đang kết nối.



Về phía máy khách, chúng ta tiến hành xử lý sự kiện broadcast.



Bây giờ khi có người mới kết nối sẽ có thông điệp chào mừng và những người đã kết nối trước đó sẽ nhận được có bao nhiêu người đang kết nối với máy chủ.

**CHƯƠNG 3: SO SÁNH SOCKET.IO VỚI WEBSOCKET**

Websocket là một kỹ thuật Reverse Ajax, dữ liệu truyền tải thông qua giao thức HTTP (thường dùng với kĩ thuật Ajax) chứa nhiều dữ liệu không cần thiết trong phần header. Một header request/response của HTTP có kích thước khoảng 871 byte, trong khi với WebSocket, kích thước này chỉ là 2 byte (sau khi đã kết nối).  
Vậy giả sử bạn làm một ứng dụng game có thể tới 10,000 người chơi đăng nhập cùng lúc, và mỗi giây họ sẽ gửi/nhận dữ liệu từ server. Hãy so sánh lượng dữ liệu header mà giao thức HTTP và WebSocket trong mỗi giây:

- HTTP: 871 x 10,000 = 8,710,000 bytes = 69,680,000 bits per second (66 Mbps)

- WebSocket: 2 x 10,000 = 20,000 bytes = 160,000 bits per second (0.153 Kbps)

Websocket dựa trên html5 nên không hỗ trợ hầu hết các trình duyệt web.

Socket.IO là một WebSocket API được tạo bởi Guillermo Rauch, CTO của LearnBoost và nhà khoa học hàng đầu của LearnBoost Labs. Socket.IO sẽ sử dụng tính năng detection xem kết nối có được thiết lập với WebSocket, AJAX long polling, Flash, v.v., giúp tạo các ứng dụng real-time thực hoạt động ở mọi nơi một cách nhanh chóng. Socket.IO cũng cung cấp API cho Node.js trông rất giống API phía máy khách.

Socket.IO đơn giản hóa API WebSocket và thống nhất các API của các transports.. Các transports bao gồm:

* WebSocket
* Flash Socket
* AJAX long-polling
* AJAX multipart streaming
* IFrame
* JSONP polling

Socket.IO cũng cung cấp các sự kiện kết nối, ngắt kết nối và thông báo thông thường mà API WebSocket gốc cung cấp. Socket cũng cung cấp một phương thức bao bọc từng loại sự kiện, giống như cách mà Node thực hiện.

Socket.IO có những chức năng mà websocket không có:

* Reliability
* Auto-reconnection support
* Disconnection detection
* Binary support
* Multiplexing support
* Room support

Kết luận: Nhìn chung so sánh giữa websocket và socket.IO có thể tổng hợp thành bảng dưới đây  
Giống nhau: Đều là công nghệ real-time, hoạt động trong một khoản thời gian mà người dùng có thể cảm nhận được ngay lập tức thời.

|  |  |
| --- | --- |
| WebSocket | Socket.IO |
| Là giao thức được thiết lập qua kết nối TCP | Là thư viện được dùng để hoạt động với websocket |
| Cung cấp hoàn chỉnh giao tiếp song song trên kết nối TCP | Cung cấp các sự kiện dựa trên giao tiếp giữa browser và server |
| Proxy và load balancer không được hỗ trợ | Có thể thiết lập proxy và load balancers |
| Không hỗ trợ broadcasting | Có hỗ trợ broadcasting |
| Không có fallback option | Có hỗ trợ fallback options |

**CHƯƠNG 4: ROOMS AND NAMESPACES**

Socket.IO cho phép “namespace” sockets, nghĩa là cho phép chỉ định các điểm đến khác nhau(endpoints)hoặc đường dẫn path.

Đây là một tính năng hữu ích để giảm thiểu số lượng tài nguyên (kết nối TCP) và đồng thời tách biệt các services trong ứng dụng bằng cách khởi tạo các kệnh tách biệt lẫn nhau.

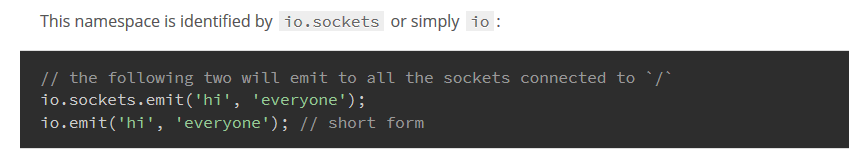
Nhiều namespace thực sự chia sẻ cùng một kết nối Websocket do đó tiết kiệm cho chúng ta các cổng trên máy chủ.

Namespace được tạo ra ở phía máy chủ. Tuy nhiên, chúng được máy khách tham chiếu đến bằng cách gửi yêu cầu đến máy chủ.

## 4.1 Default namespace

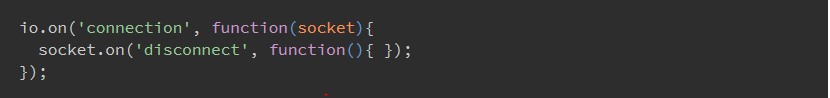
Namespace gốc ‘/’ là namespace mặc định, được kết nối với máy khách nếu namespace không được chỉ định bởi client trong khi kết nối với máy chủ. Tất cả các kết nối đến máy chủ sử dụng phía máy khách ở đối tượng socket được thực hiện với namespace mặc định.

Ví dụ:



Mỗi Namespace sẽ gửi(emit) một sự kiện kết nối đến server, server sẽ nhận mỗi đối tượng socket gửi đến với dạng tham số

Ví dụ



**4.2 Custom namespace**

Chúng ta có thể tạo ra các namespace tùy chỉnh của riêng mình. Để thiết lập namespace tùy chỉnh, chúng ta có thể gọi phương thức of ở phía máy chủ. 

Tiếp theo để kết nối một máy khách với namespace này, bạn cần cung cấp namespace làm tham số cho lời gọi hàm tạo io để tạo một kết nối và một đối tượng socket ở phía máy khách client.



Mỗi khi có một ai đó kết nối họ sẽ nhận được sự kiện “chao”

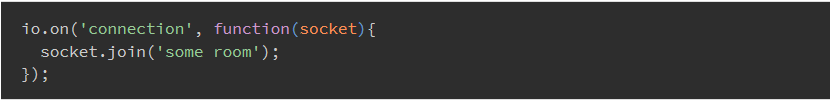
**4.3 Rooms**

Với mỗi namespace, chúng ta có thể khai báo kênh để socket có thể join hoặc leave.

**4.3.1 Joining and Leaving**

Dùng hàm join() để liên lạc tới một room xác định, sử dụng hàm to() hoặc hàm in() để broadcasting hoặc emitting.

Ví dụ:





Để thoát khỏi room, sử dụng hàm leave(), với cú pháp tương tự join().

\***Lưu ý**: hàm to() tham số truyền vào là một socket.id và socket.id này không thể custom. Nó được khởi tạo ngẫu nhiên khi có một client kết nối tới server.

**4.4 Default Room**

Mỗi socket trong Socket.IO được xác định bởi một mã định danh, duy nhất và không thể đoán trước được, socket.id. Để thuận tiện cho người dùng, mỗi socket sẽ tự động join một room bằng chính id đó, nói cách khác mỗi socket khi kết nối vào đều tạo ra một room của riêng chính mình.

Điều này dẫn đến việc dễ dàng broadcast tin nhắn tới các socket khác rằng ta đã kết nối.



**4.5 Disconnection**

Sau khi ngắt kết nối, các sockets sẽ leave tất cả các room mà chúng đã kết nối đến, điều này dẫn đến việc các client đang làm việc sẽ không phát hiện rằng máy chủ đã die, và sẽ vẫn tiếp tục cố gắng kết nối tới máy chủ.

Lưu ý: Khi bị ngắt kết nối đột xuất, các sockets sẽ mất đi địa chị socket.id hiện tại và khi kết nối lại được với máy chủ. Các sockets sẽ được cung cấp một socket.id mới, việc gửi tin nhắn tới các socket.id sẽ không còn thực hiện được.

**4.6 Gửi tin nhắn từ bên ngoài**

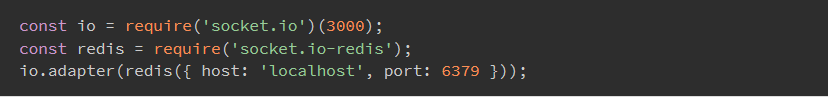
Trong một số trường hợp, chúng ta có thể muốn emit các events đến các sockets trong namespace/ rooms từ bên ngoài của socket.IO

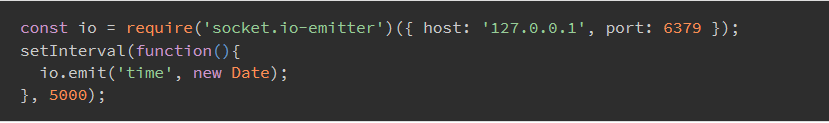
Có rất nhiều cách để giải quyết vấn đề này, như triển khai một kênh cụ thể để gửi tin nhắn vào.

Để tạo điều kiện cho sử dụng này, tạo hai modules

* Socket.io-redis
* Socket.io-emitter

Bằng cách config redis adapter, chúng ta có thể emit message từ bất kỳ ứng dụng nào tới kênh.

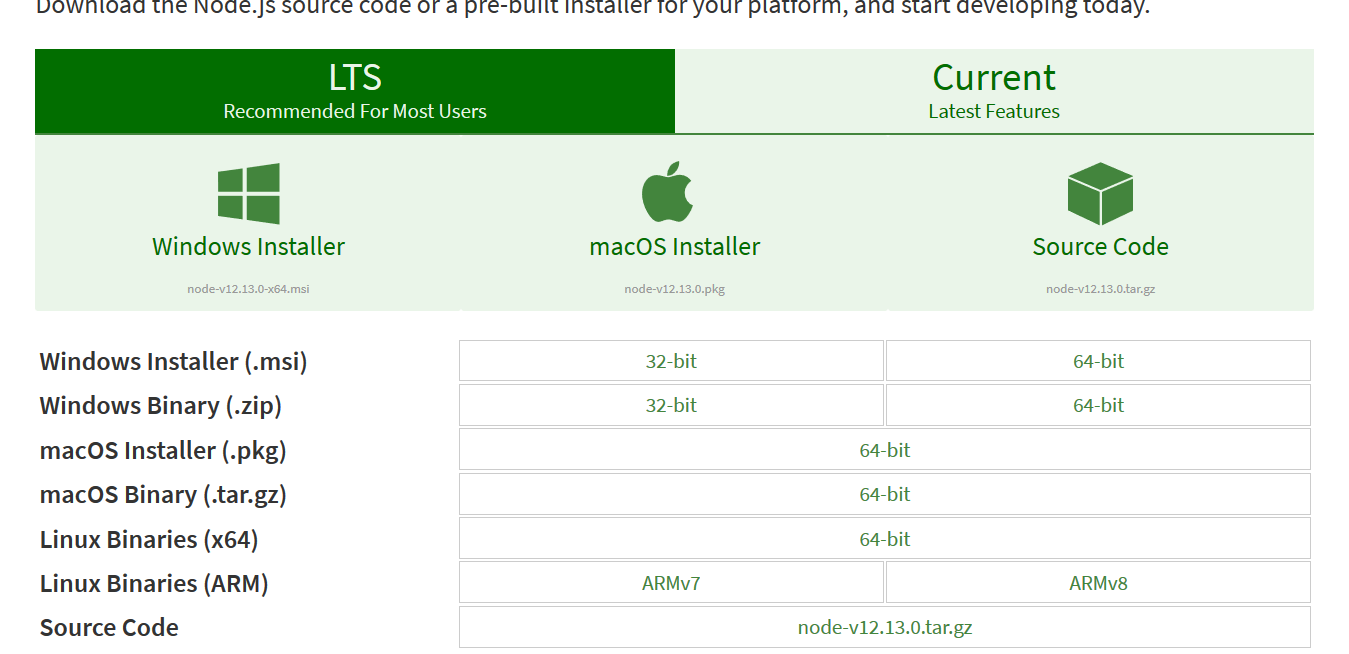




**CHƯƠNG 5: USER GUIDE**

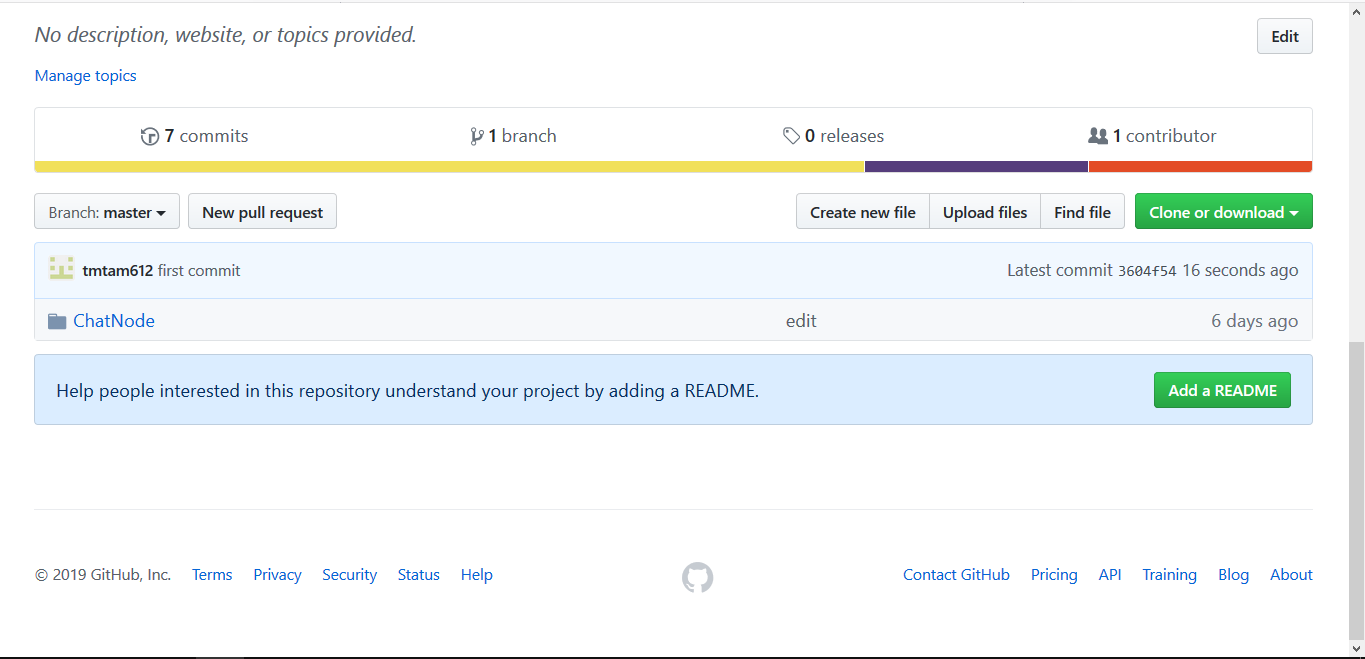
**5.1 Cài đặt môi trường**

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

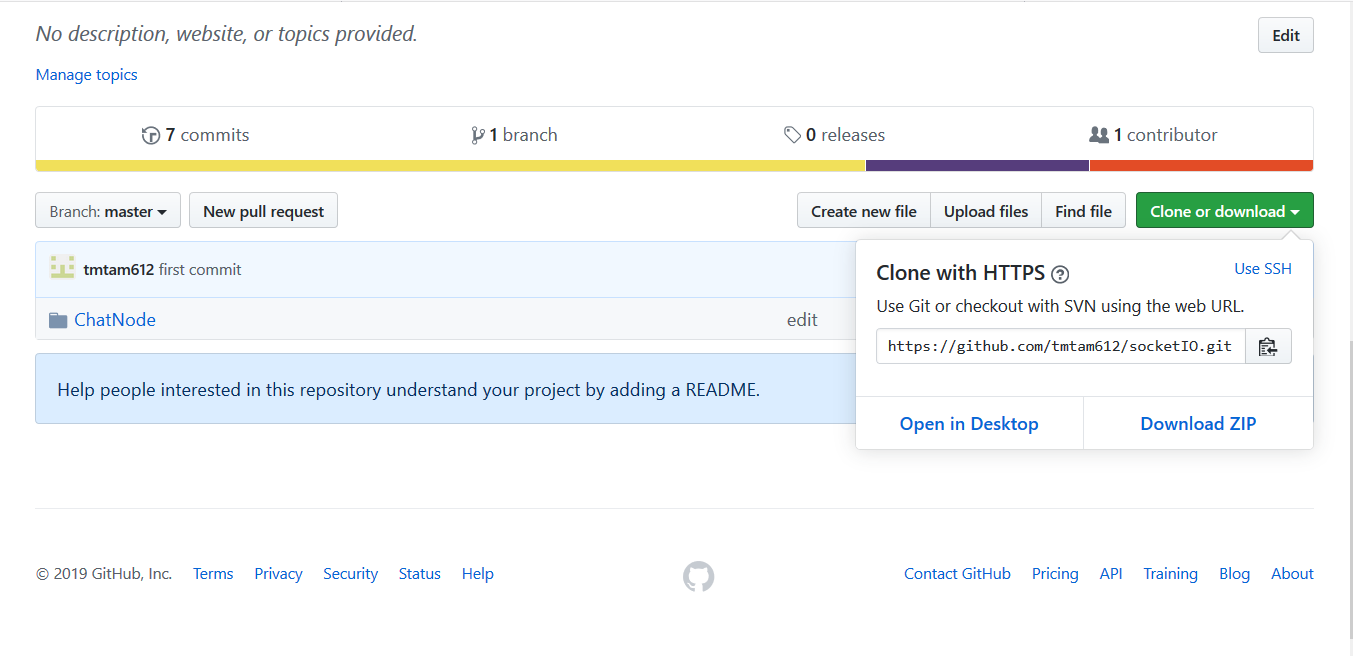


Download bản phù hợp với hệ điều hành của mình, và chạy cài đặt bình thường như những phần mềm ứng dụng khác.

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.

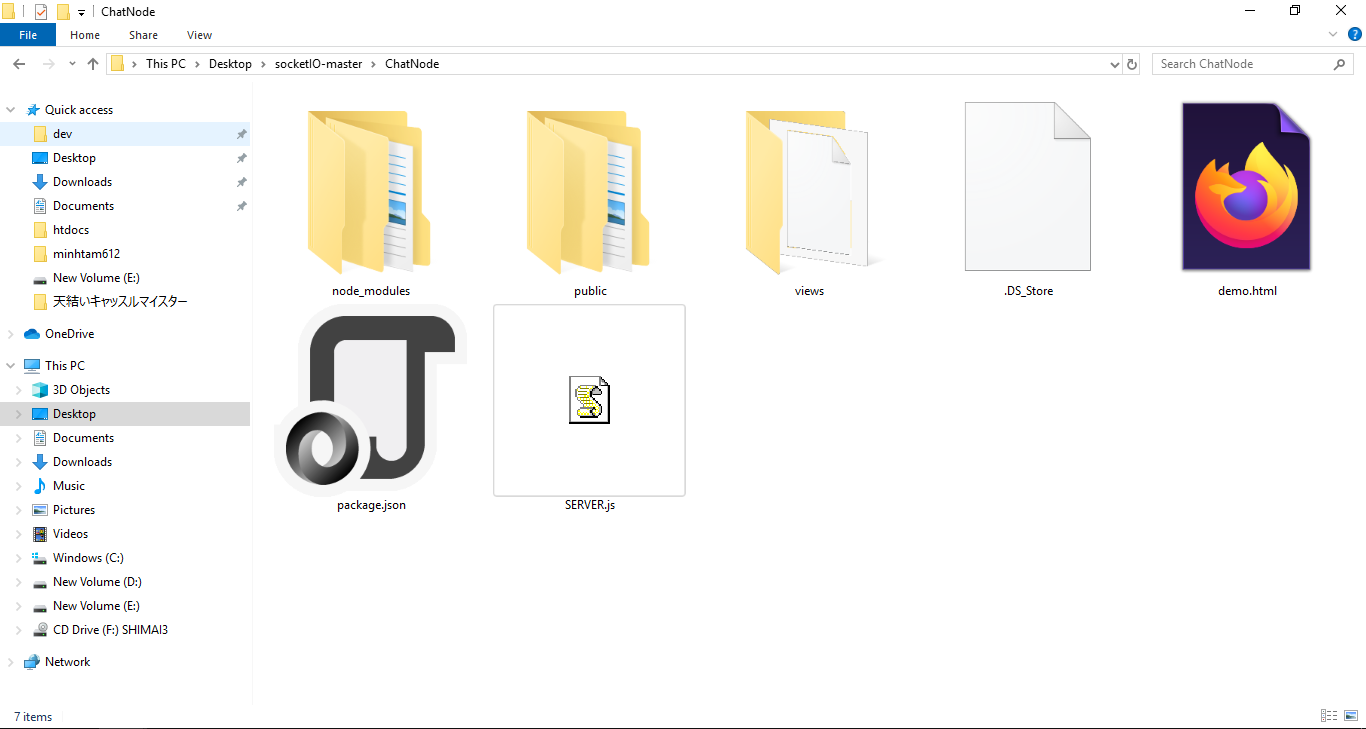


Nhấn vào nút clone or download, chọn tải về dưới dạng zip (Download Zip).

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

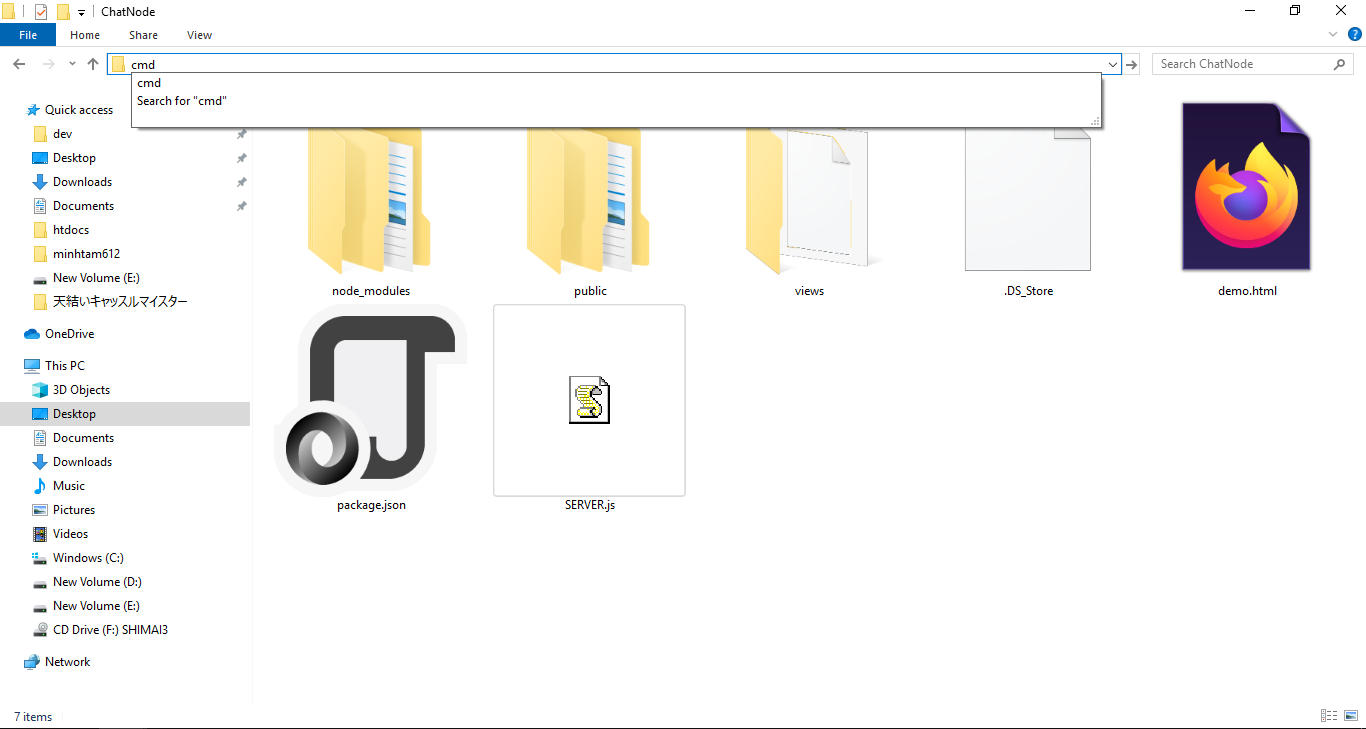


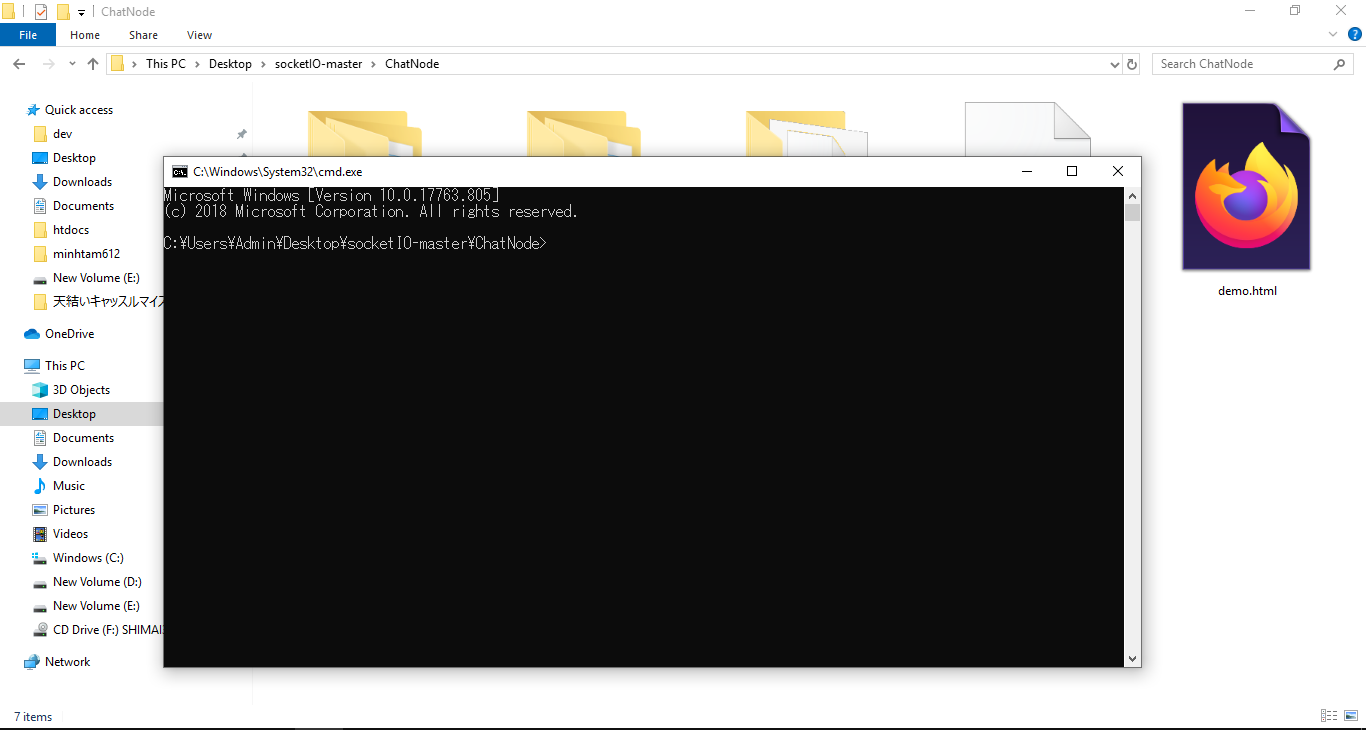
Truy cập vào thư mục ChatNode, có những file như sau là đã cài đặt môi trường thành công



**5.2 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.



Lúc này sẽ xuất hiện màn hình cmd, 

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

**6.1 Register vào ứng dụng**

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



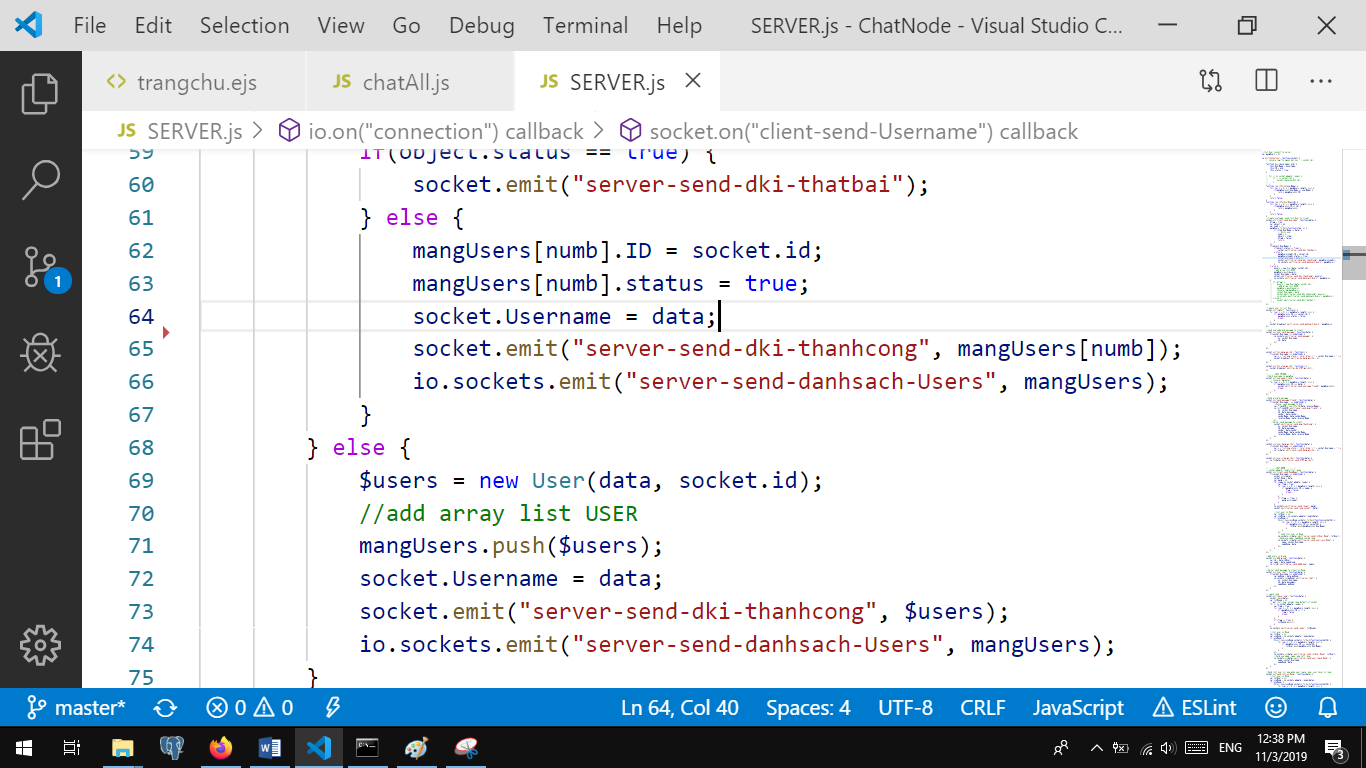
Nút register sẽ tạo một sự kiện gọi là đăng ký tài khoản với socket.io,



$name là tên đăng nhập, khi $name khác rỗng và khác khác null, khác undefined, nó sẽ gửi đến server có namespace tên là “client-send-Username” với data là tên của người đăng nhập bằng câu lệnh emit().

Trên server lúc này sẽ config một namespace tên là “client-send-Username” để hứng sự kiện được nhận từ phía client.

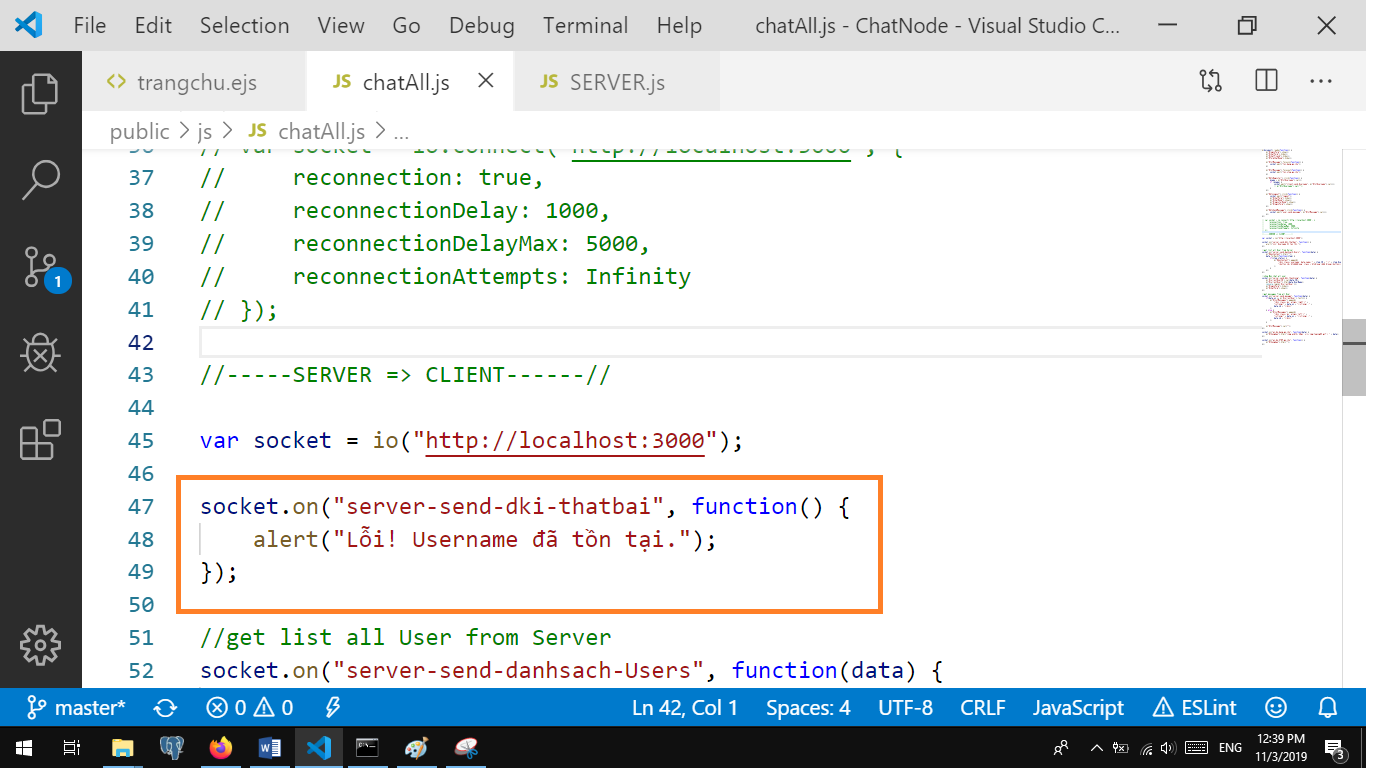




**6.1.1 Register thất bại.**

Trong namespace “client-send-Username”, server sẽ duyệt mảng mangUsers, mangUsers 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 namespace có tên “server-send-dki-thatbai” để gửi thông báo đến client.

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

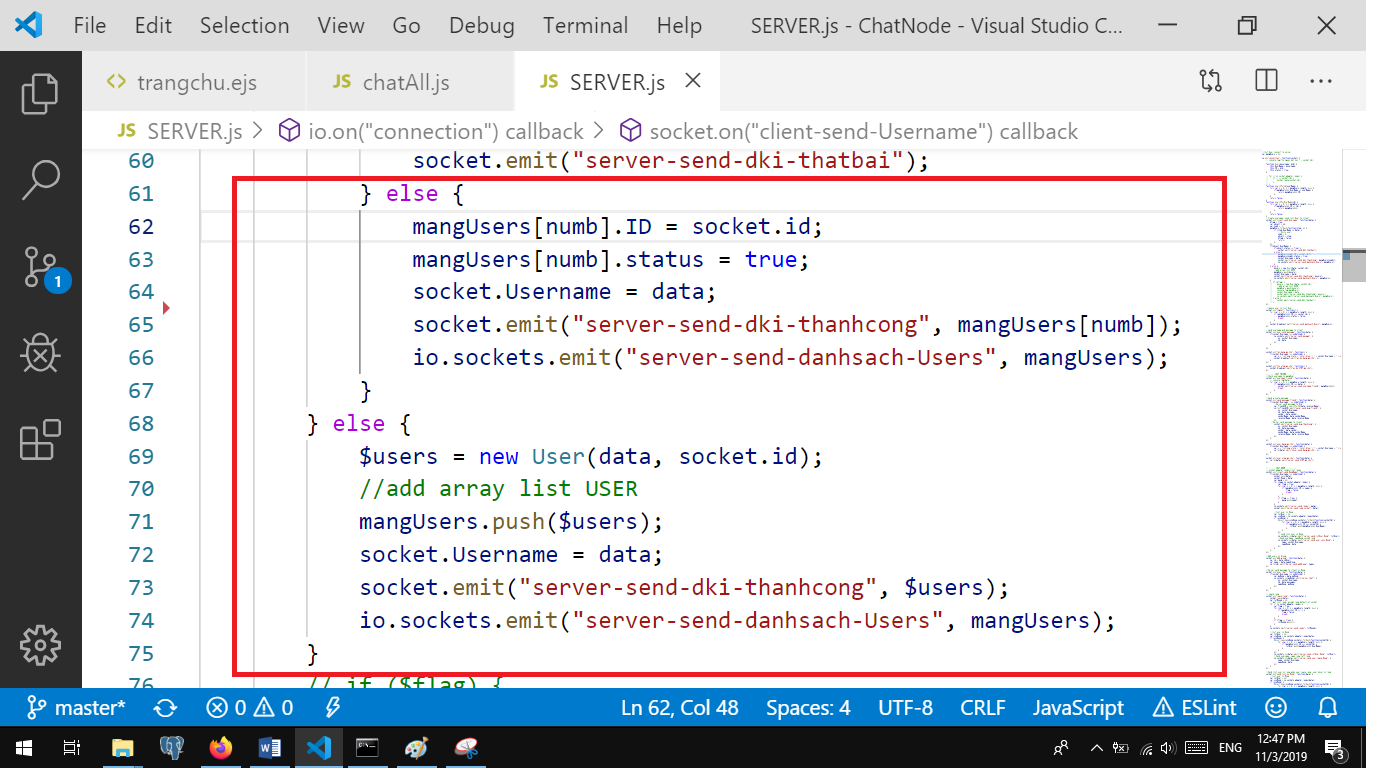




Khi nhận được sự kiện, client sẽ hiện lên message alert Username đã tồn tại.

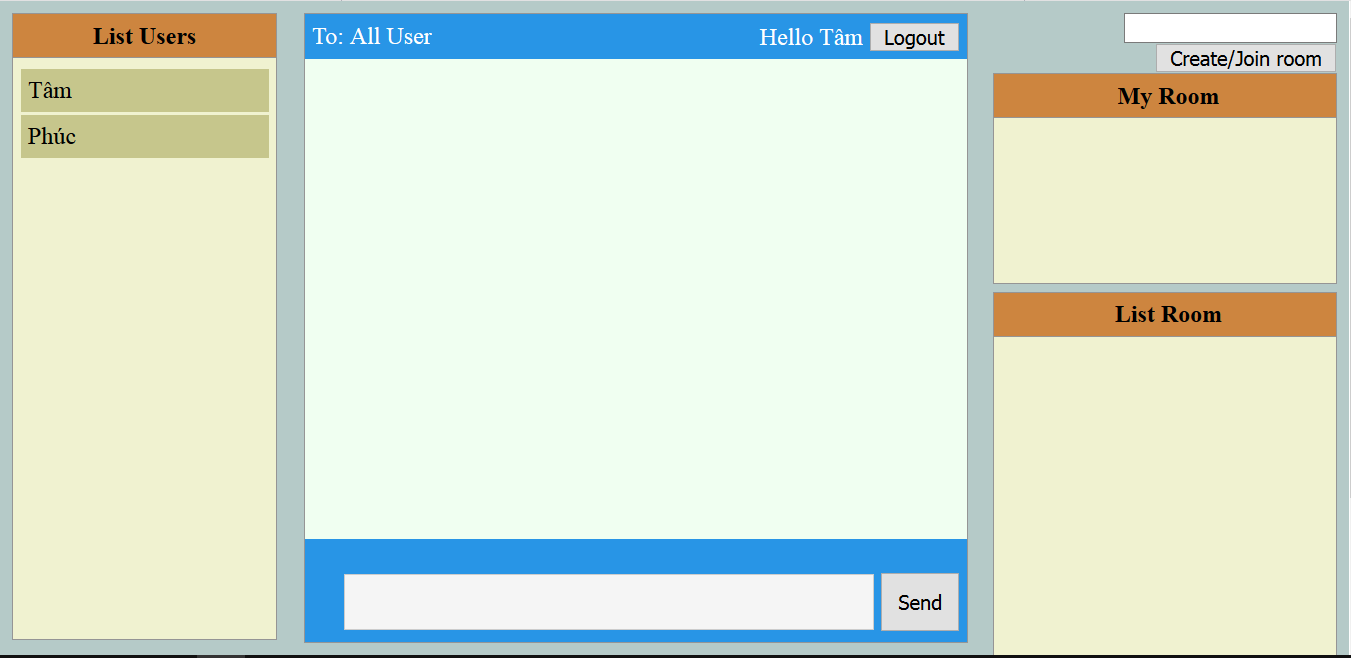
**6.1.2 Register thành công**

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



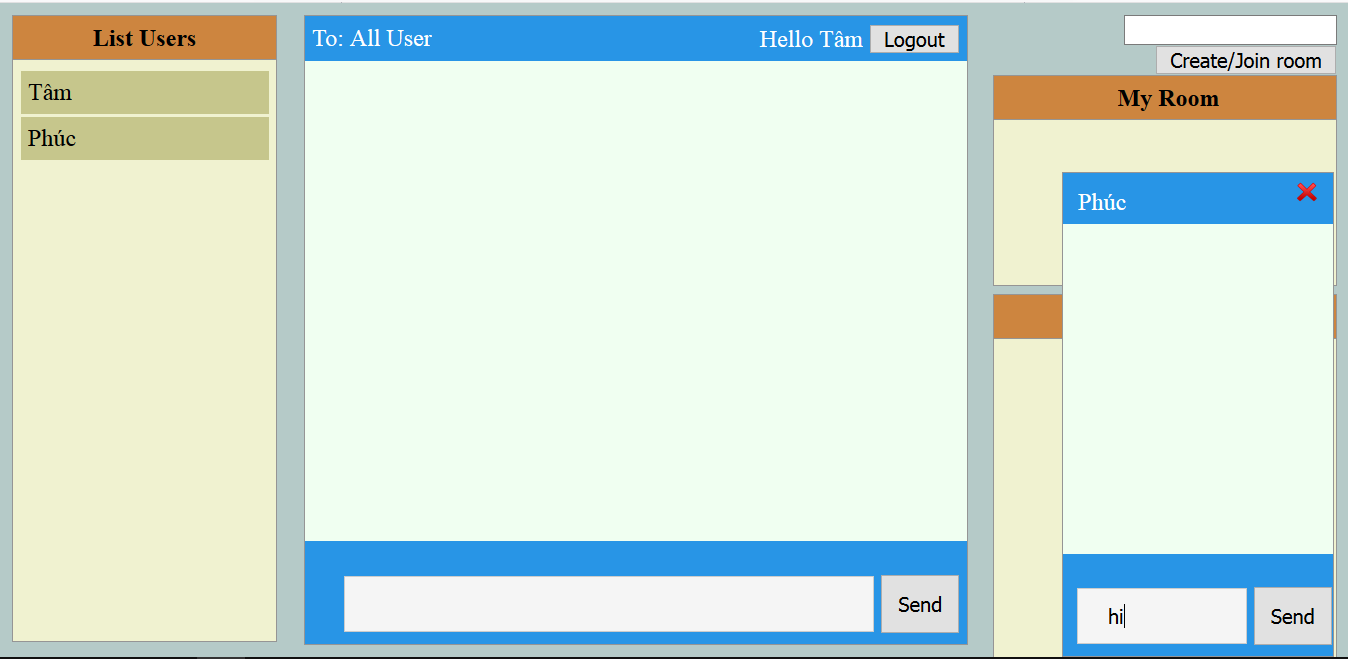
Khi đó server sẽ phát đi một namespace “server-send-danhsach-Users” để cập nhật mangUser đang online, và một namespace có tên là “server-send-dki-thanhcong” với tham số là tài khoản vừa được tạo hoặc vừa được cập nhật để vào trang chính của webchat.

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



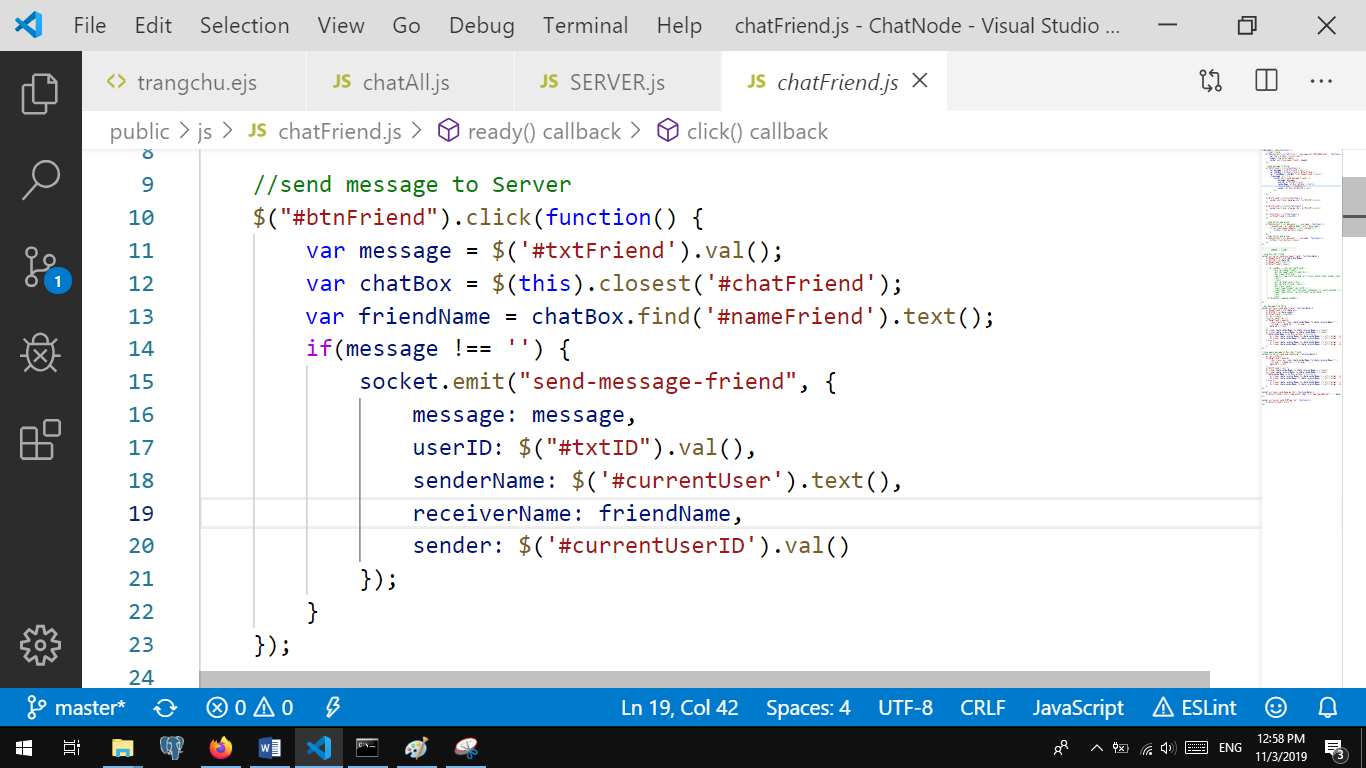
**6.2 Private Message**

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

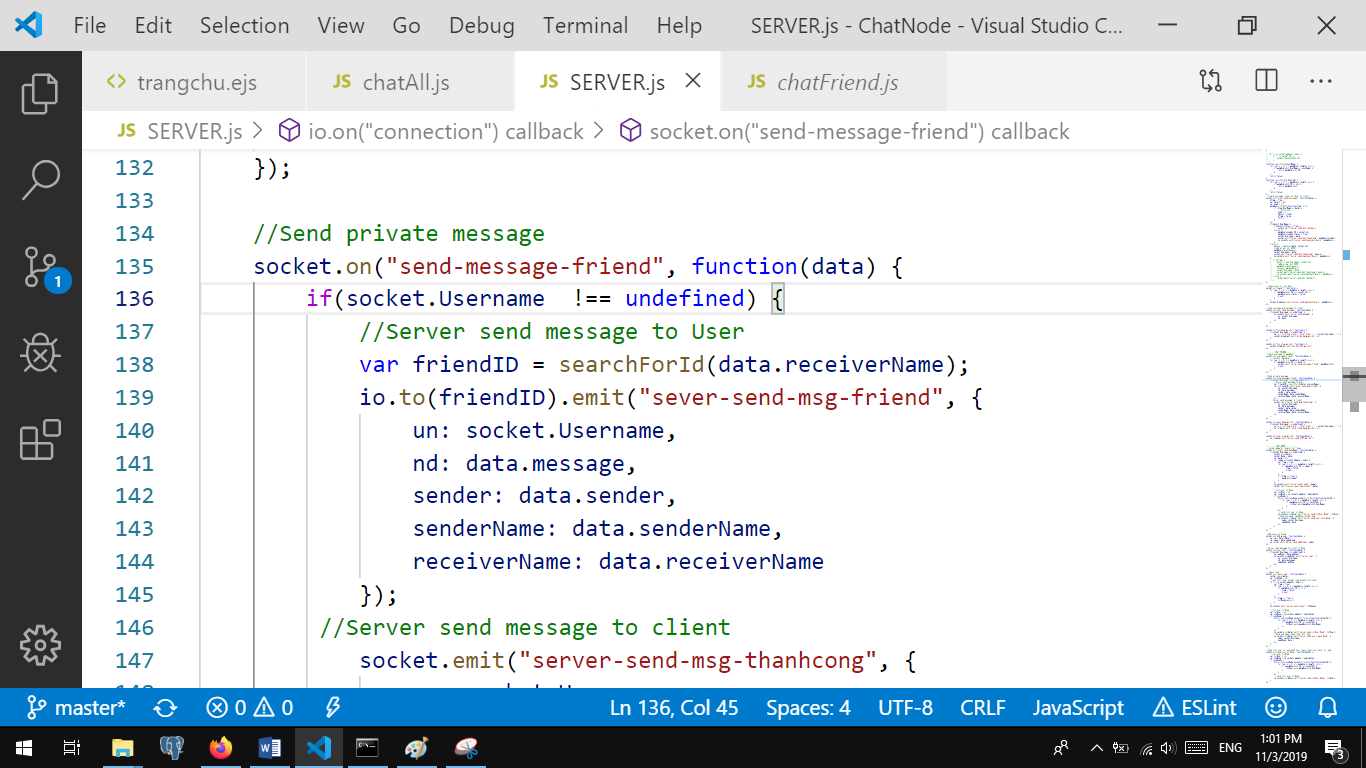


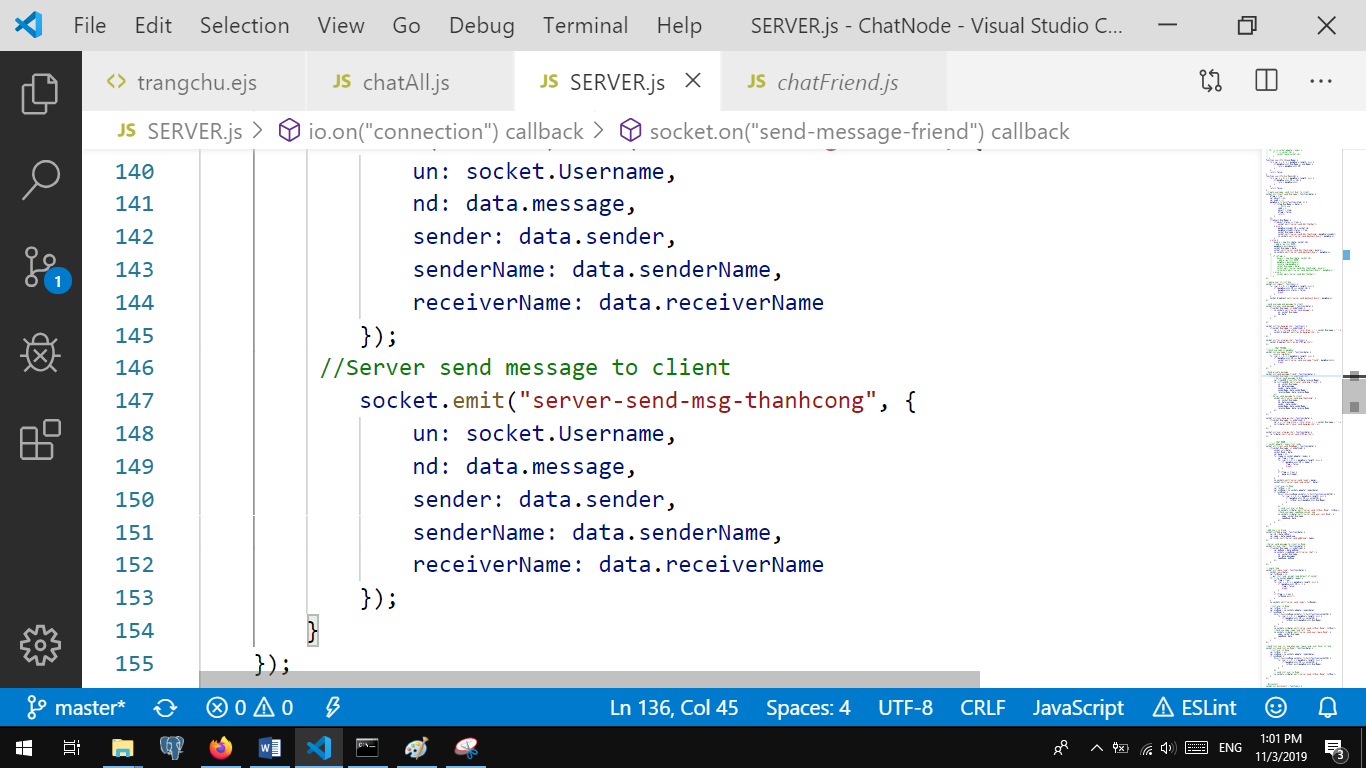
Khi nút send được gửi đi, hoặc nhấn nút enter, client sẽ tạo một sự kiện gửi tin nhắn, và phát đi đến server để giải quyết, server nhận được và giải quyết lập tức gửi lại cho người nhận tin nhắn.

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 namespace là “send-message-friend”



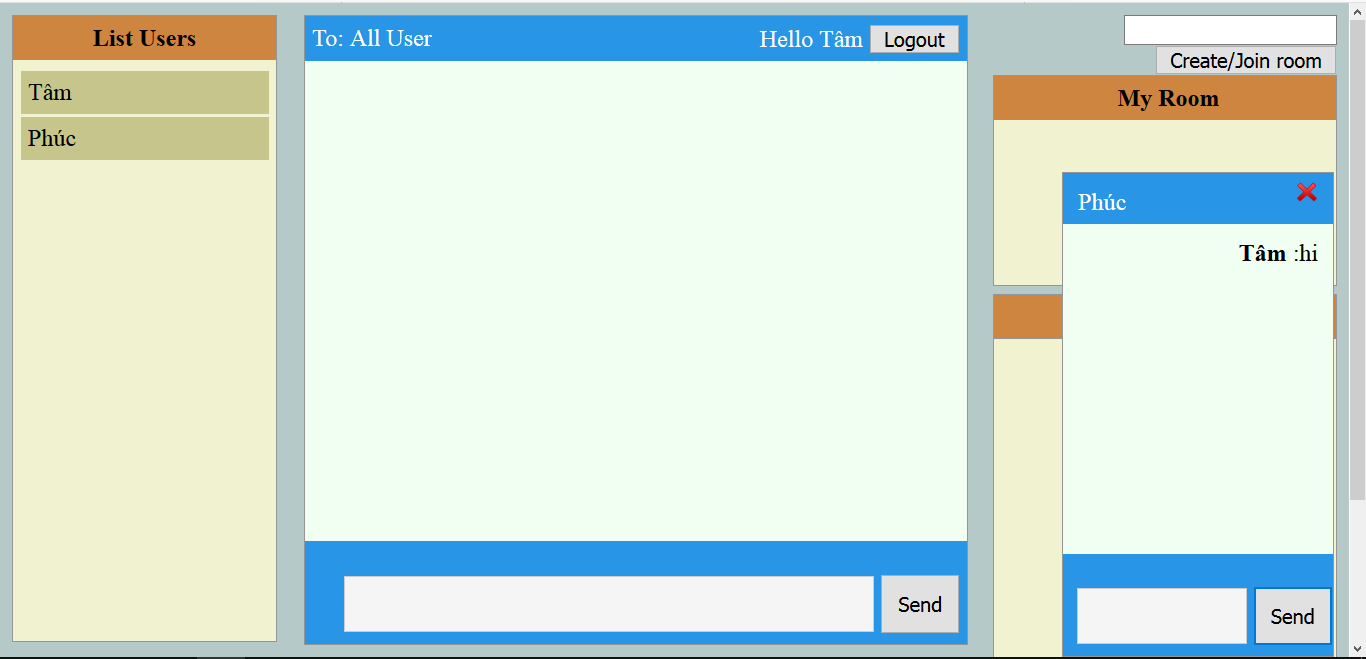
Khi đó ở phía server sẽ config một namespace tên là “send-message-friend” để hứng sự kiện được gửi tới.

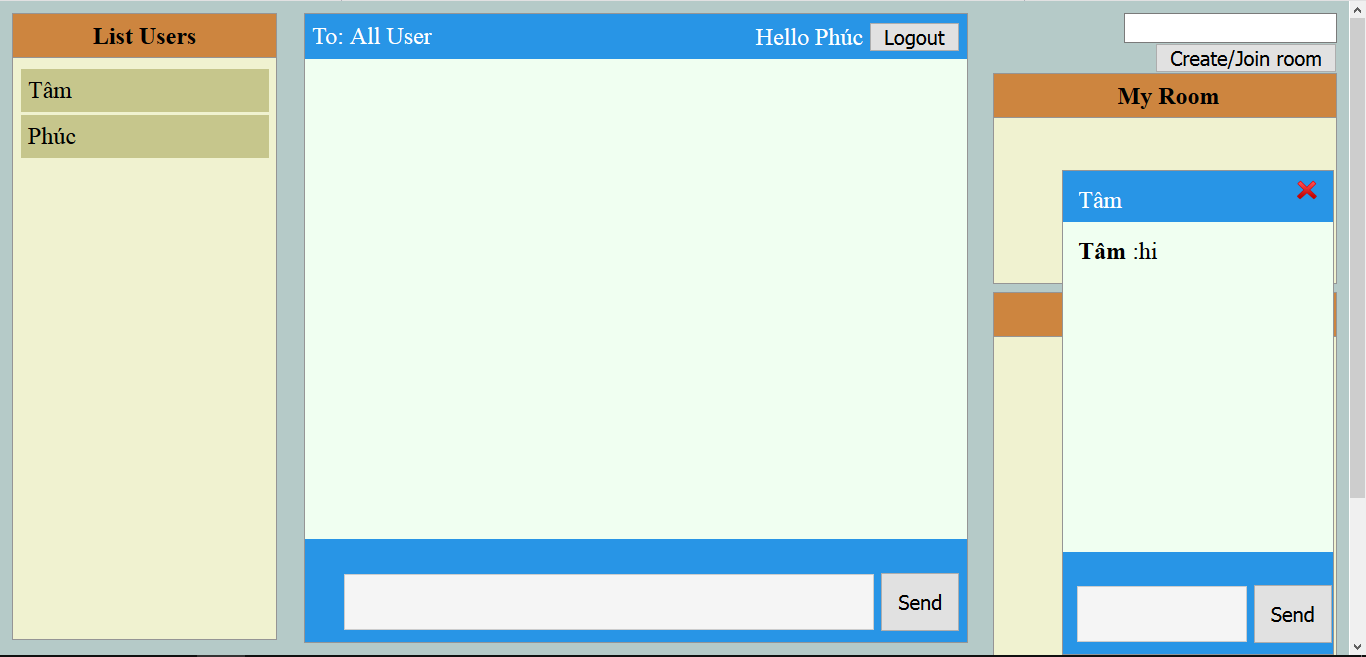




Trên server khi nhận được sự kiện “send-message-friend”, sẽ lập tức gửi đi 2 sự kiện “server-send-msg-friend”, sự kiện này sẽ gửi tin nhắn đến người nhận và “server-send-msg-thanhcong” để hiển thị cho chính người gửi thấy được tin nhắn mà mình đã gửi, client hứng và tạo giao diện, hiển thị lên cho cả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

**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.





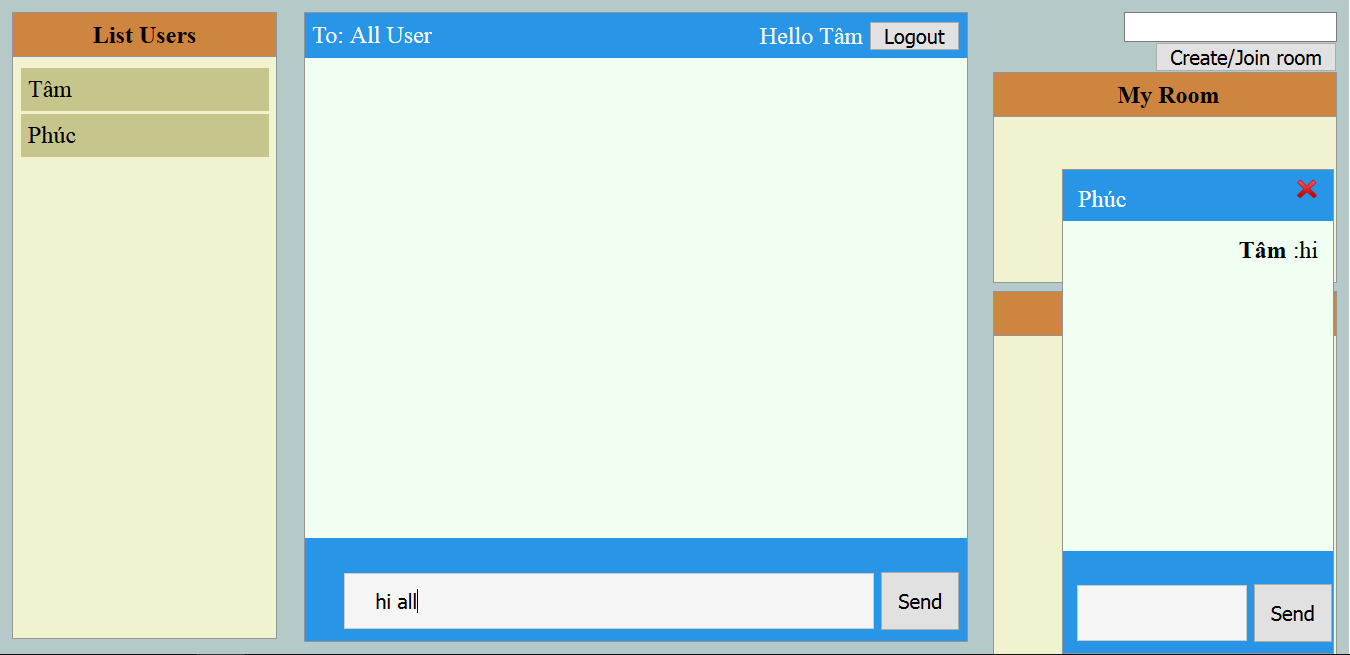
**6.3 Chat All(Broadcasting)**

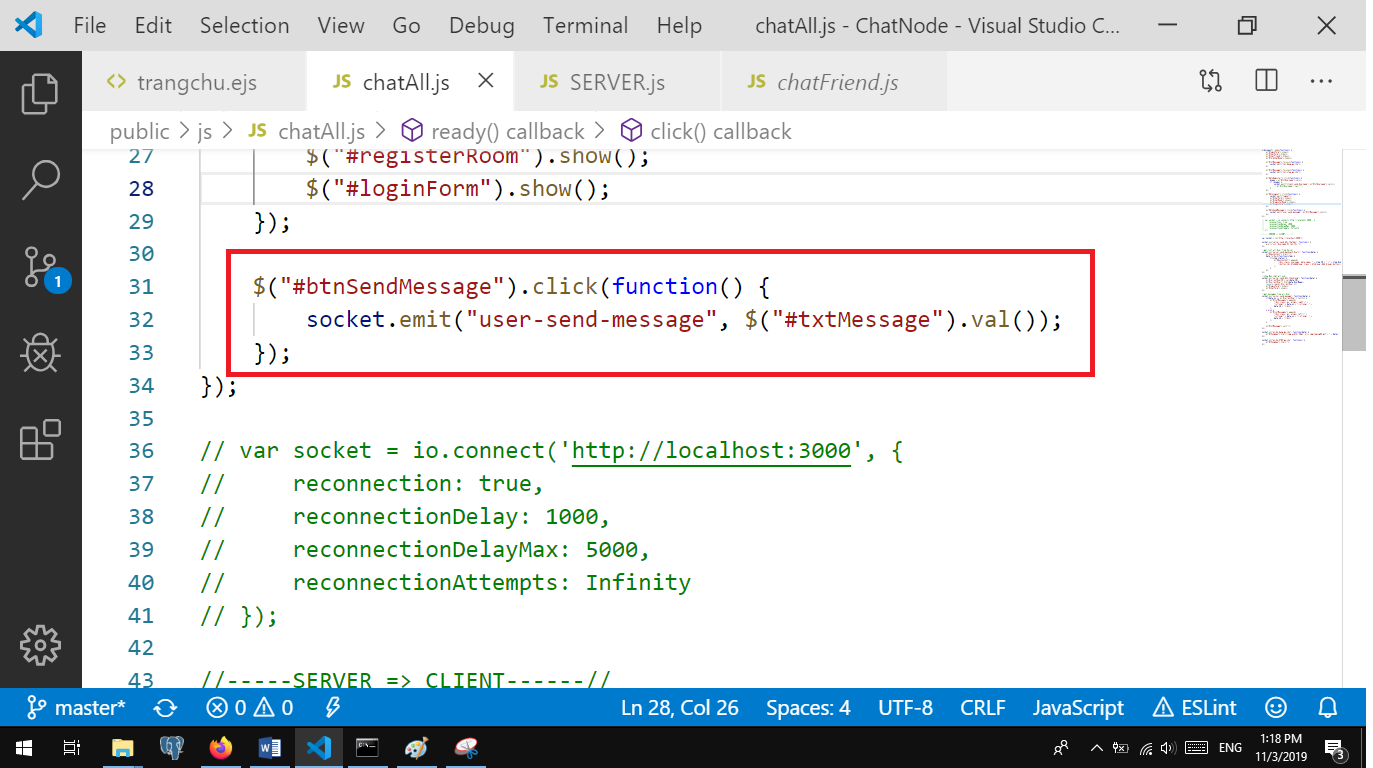
Với socket.io, việc gửi tin nhắn broadcast là công việc đầu tiên cơ bản nhất, vì khi phát triển socket.io, trường hợp đầu tiên để phát triển đó chính là broadcast. Việc chat all, broadcast message đến mọi người dễ dàng hơn rất nhiều so với việc private message, hay chat group đến room được tạo.

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.

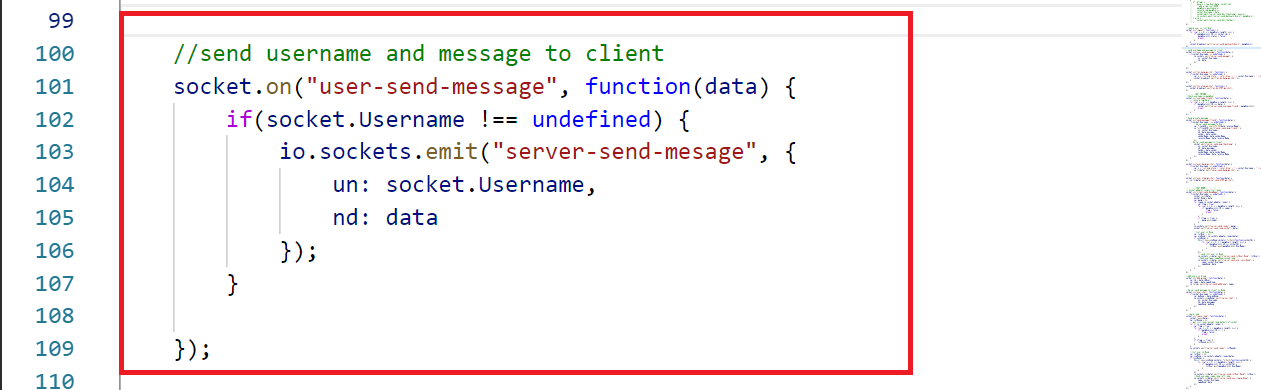




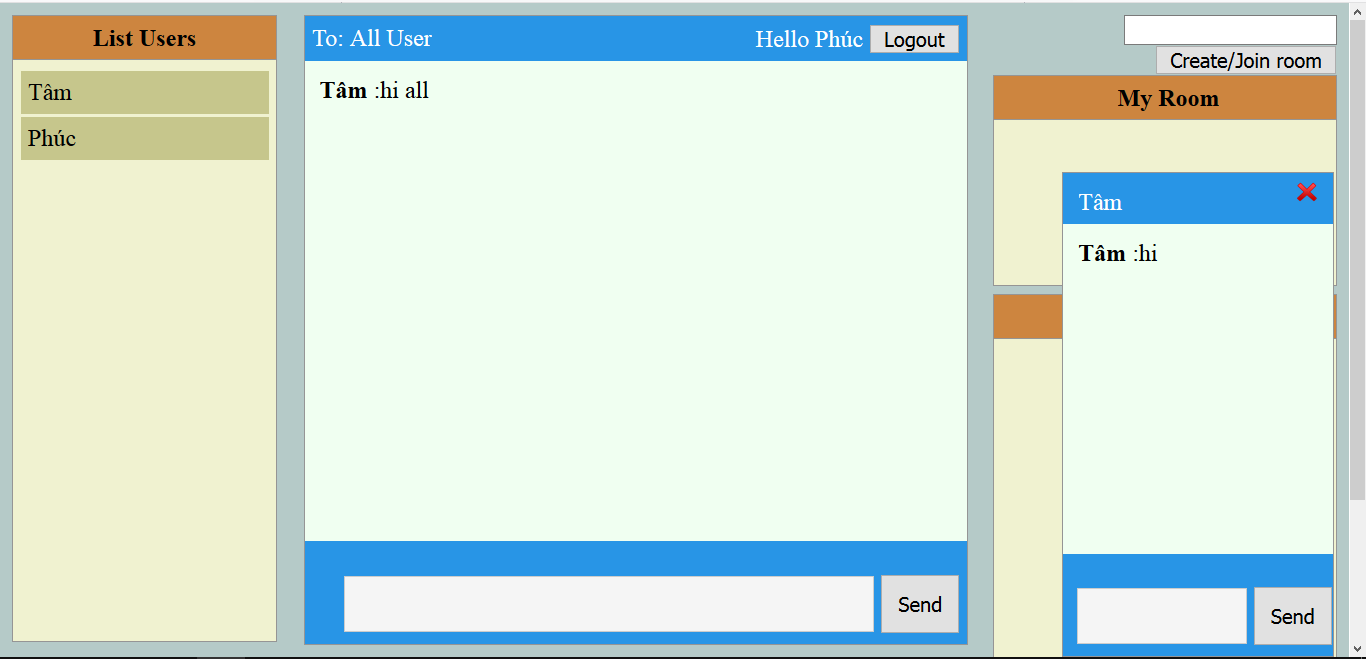
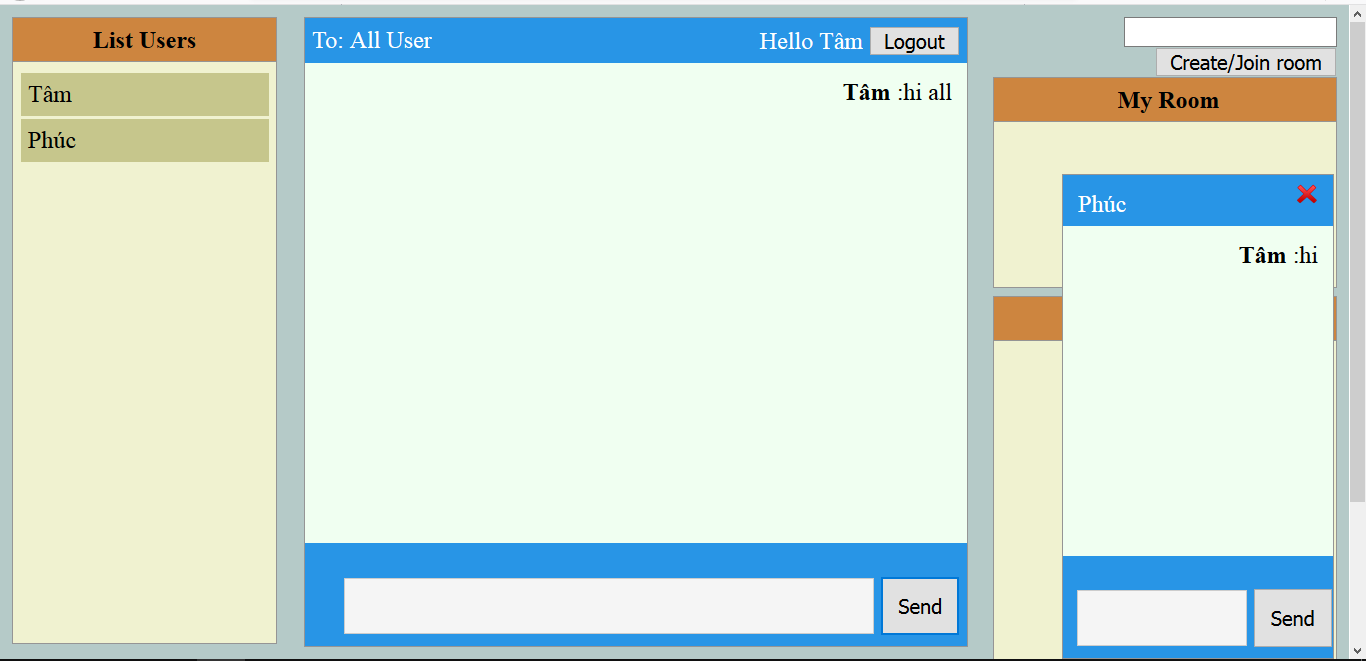
Khi server nhận được một sự kiện “user-send-message”, server ngay tức khắc gửi đi một sự kiện gọi là “server-send-message” để gửi tin nhắn đới mọi client đang kết nối.

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.



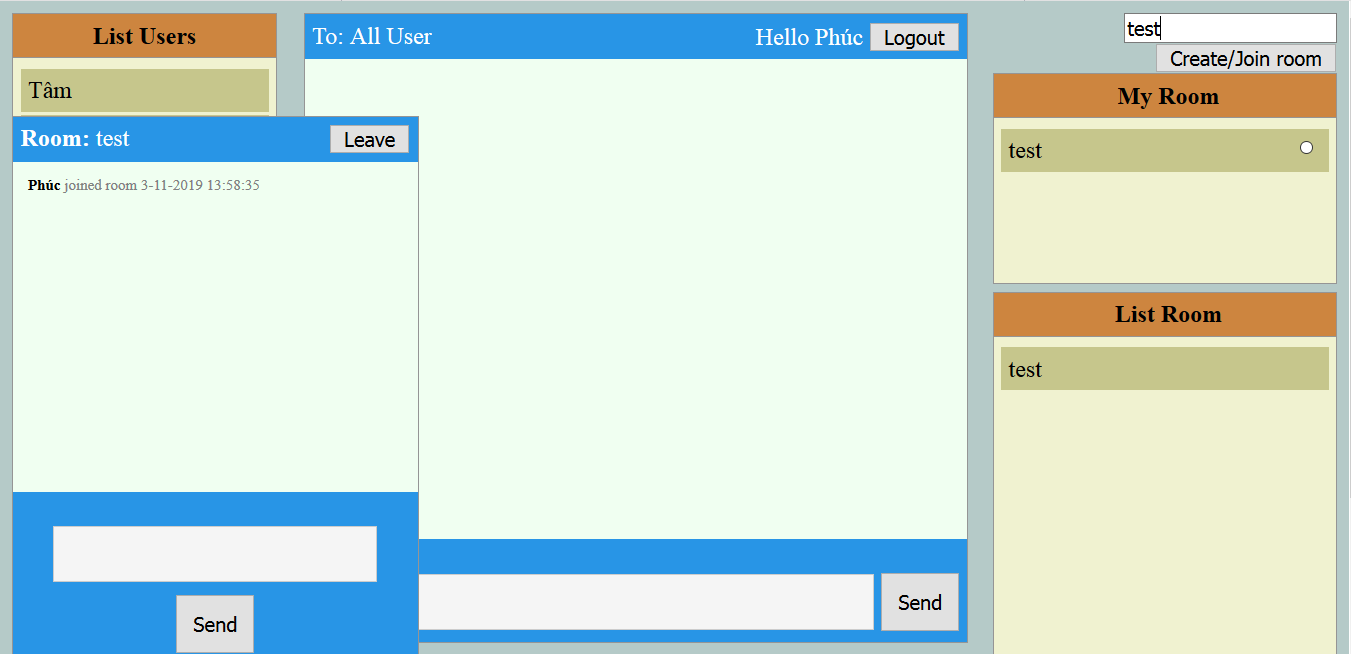
*Code xử lý bên server*



**6.4 Rooms**

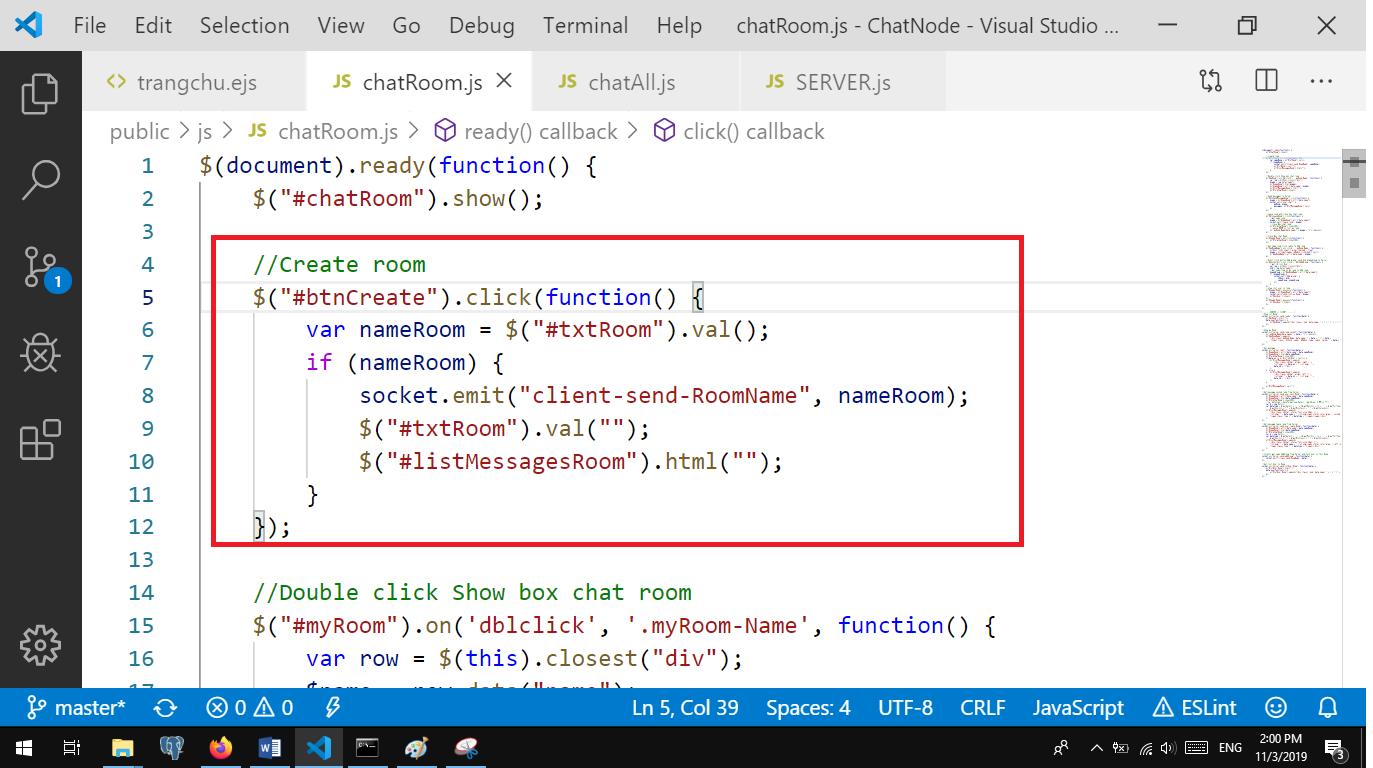
Để tạo room chúng ta nhập vào textbox tên room kế bên nút Create/Join room. Khi enter hay nhấn nút Create/Join room, client sẽ tạo một sự kiện bao gồm thông tin room cần tạo, người tạo đến server, khi đó chúng ta có thể config socket#id của room đã tạo.

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

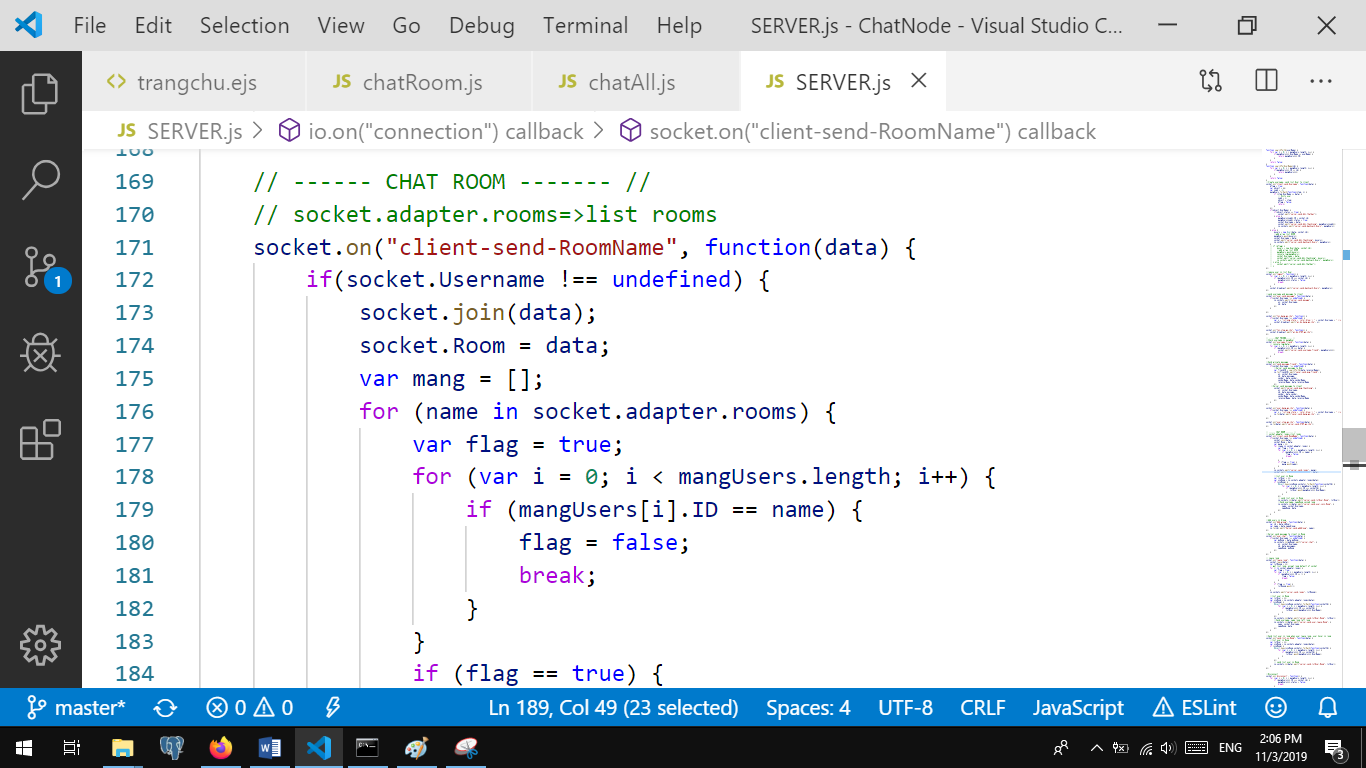


Khi tạo room, hay tham gia room, server sẽ broadcast một tin nhắn đến tất cả thành viên trong room là có người tham gia.

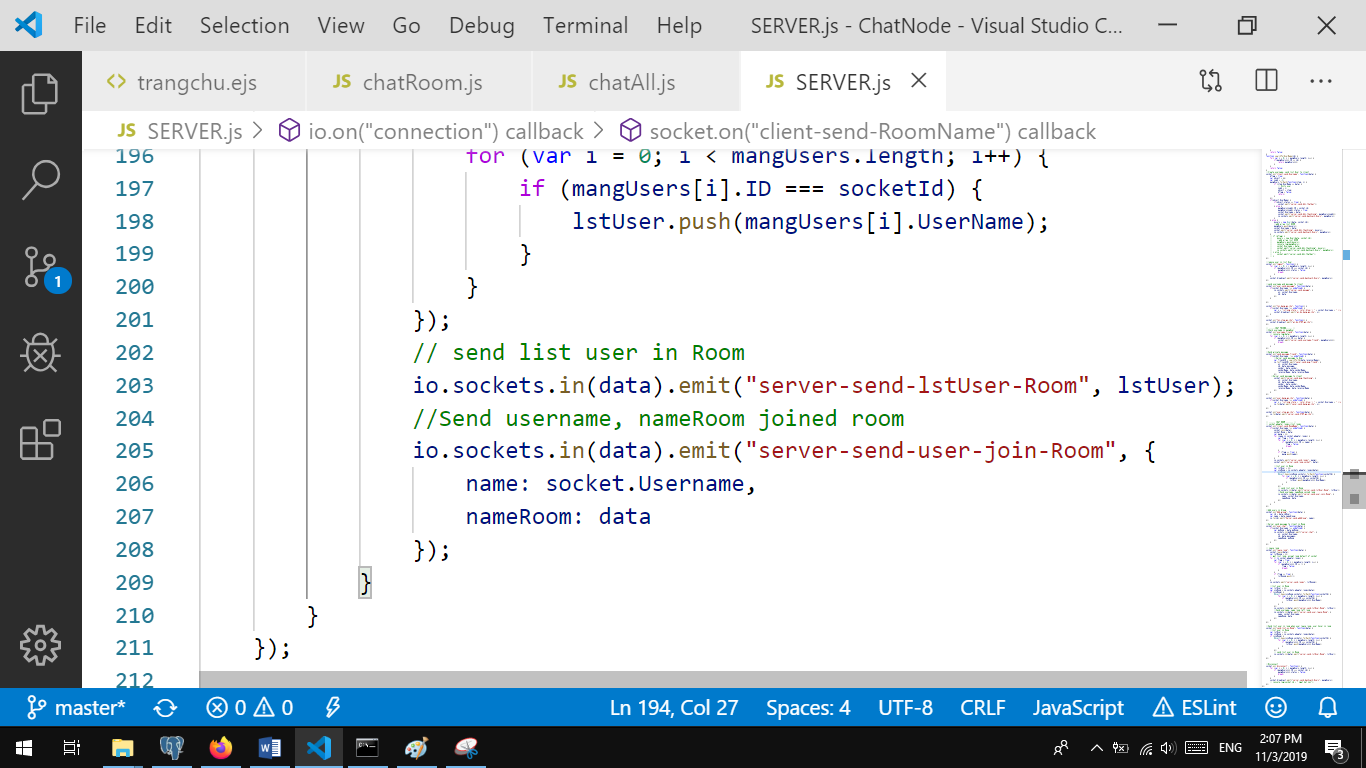
Khi rê chuột vào tên room, chúng ta có thể thấy tên của các thành viên có mặt trong room, người dùng bên ngoài có thể tham gia room khi được mời hoặc tự động gõ vào textbox tên room cần tham gia.



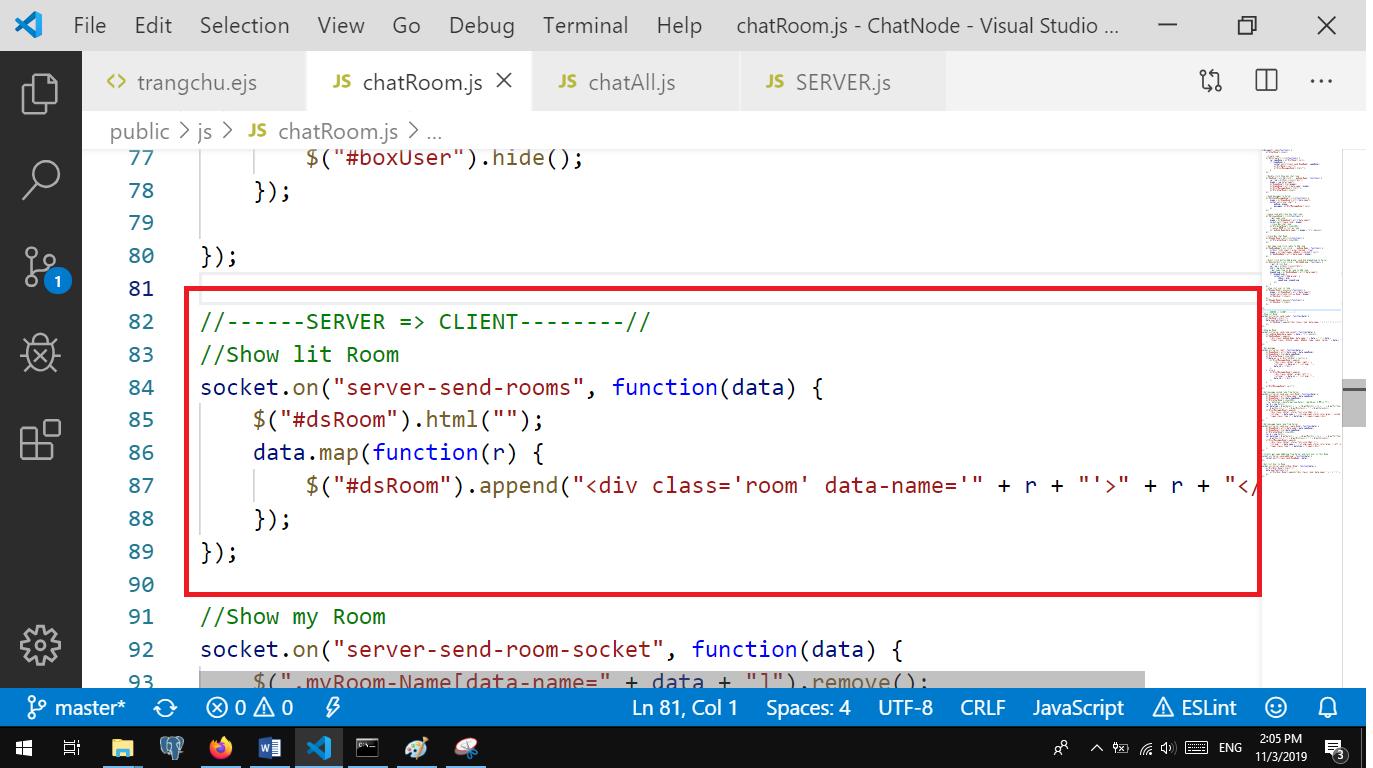
Client gửi đến server một sự kiện có tên “client-send-RoomName”, server nhận được xử lý tương tự như tạo tài khoản register, kiểm tra xem tên phòng nhập đã tồn tại chưa. Nếu tồn tại thì auto join vào room, còn nếu chưa thì tạo mới một phòng có tên vừa được nhập.



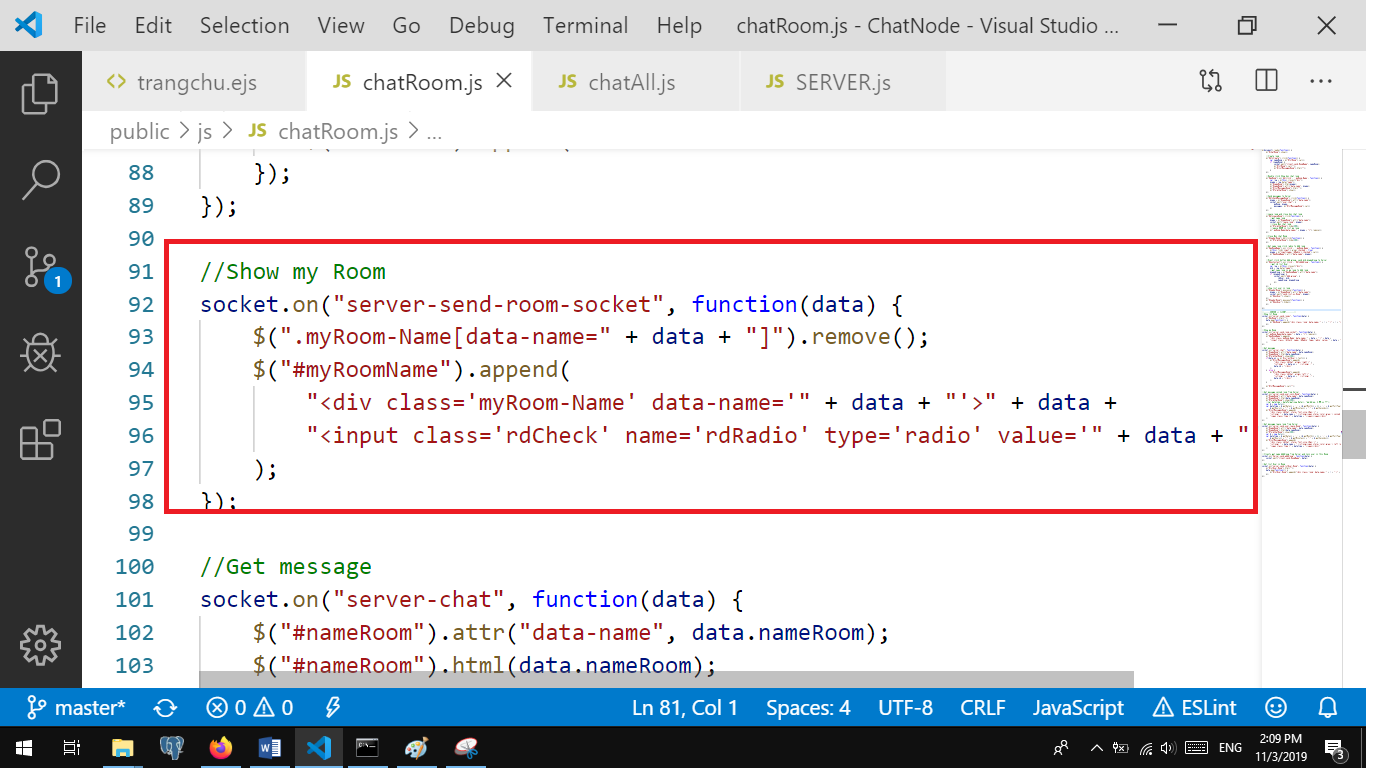




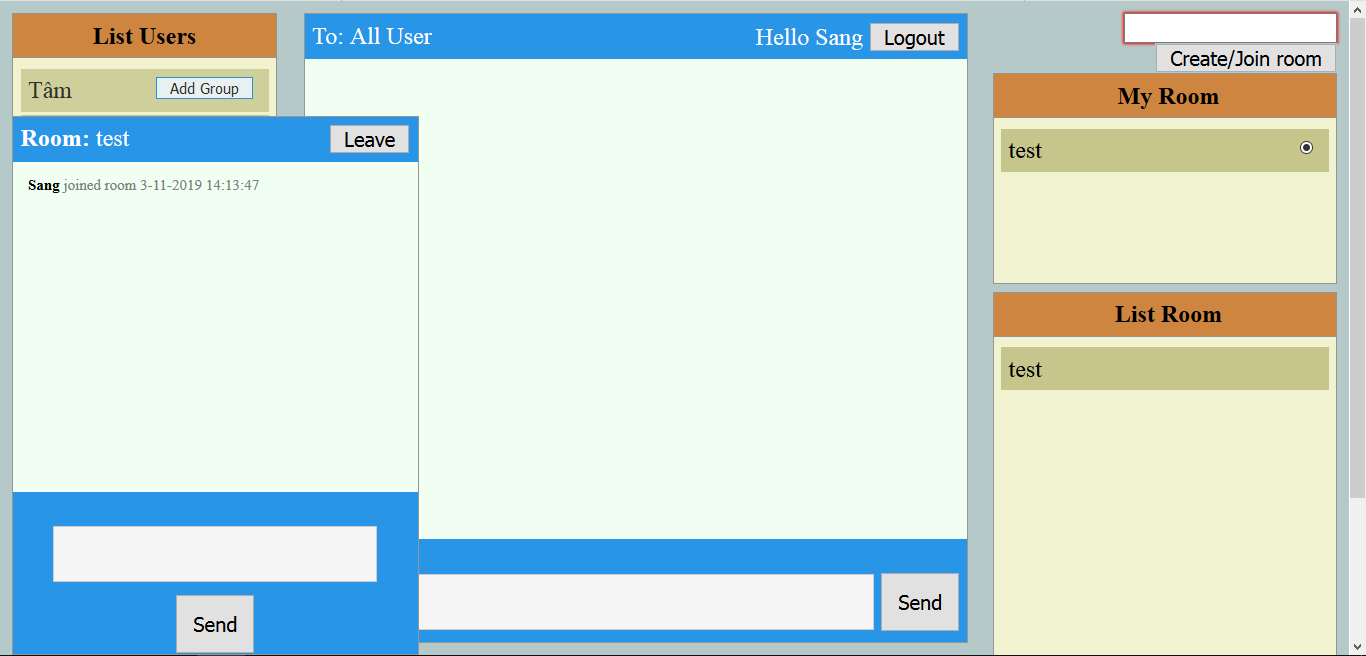
Ngoài ra server sẽ báo về client những room đã được tạo ra trên màn hình “List room” bằng việc phát đi một tín hiệu “server-send-rooms” đến client



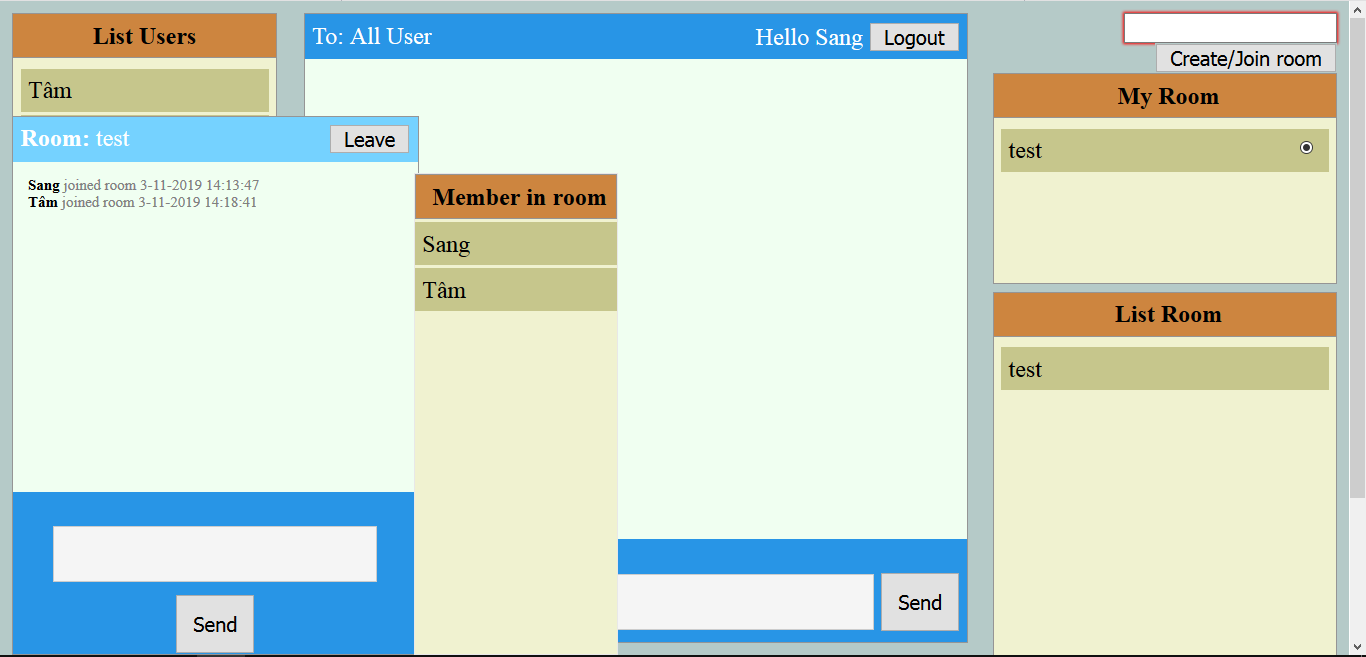
Khi người dùng tham gia vào room, server sẽ gửi đến client tín hiệu “server-send-room-socket” để hiện lên danh sách các room mà người dùng đó đang tham gia. Việc làm này sẽ giúp người dùng dễ quản lý các room của mình và cho phép mời các thành viên khác tham gia room.



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.



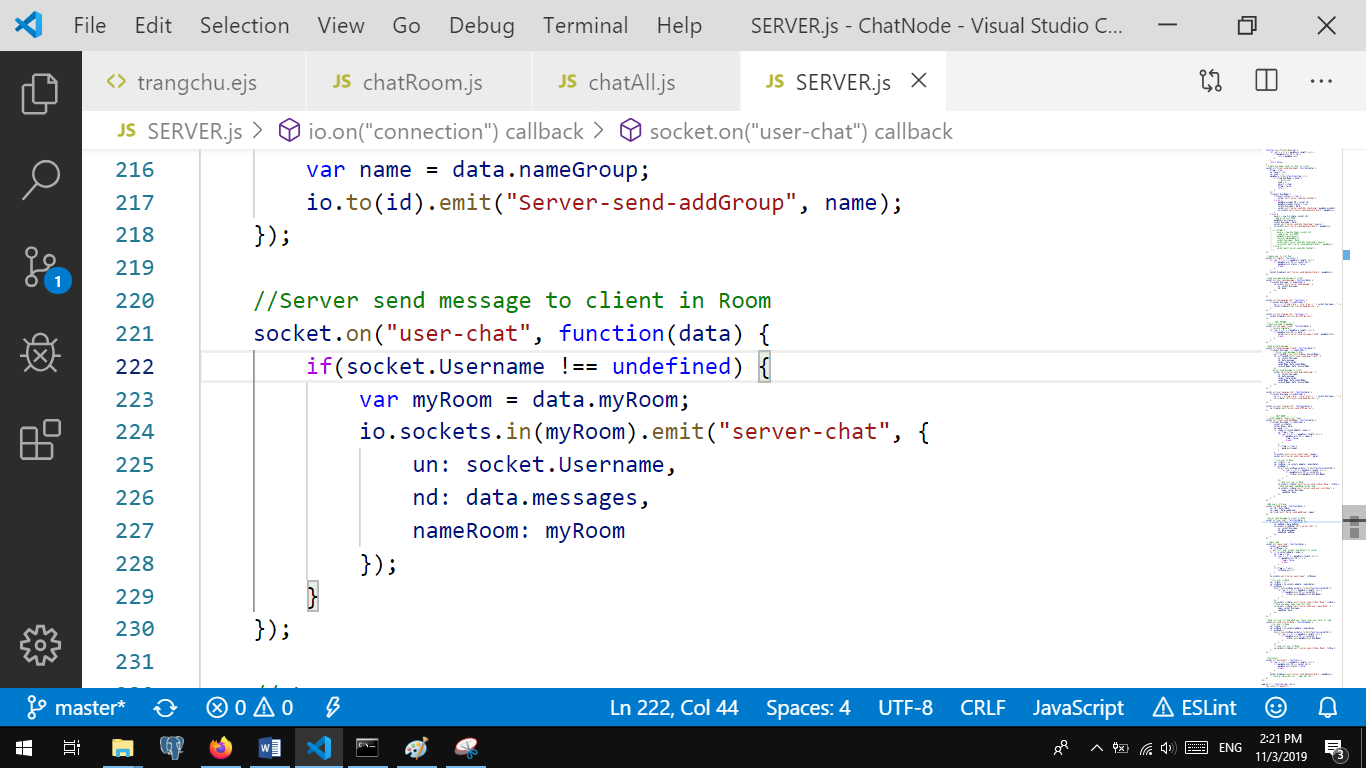
Khi mời người dùng, server sẽ xử lý xem người dùng đó có tồn tại trong group chưa, nếu chưa thì sẽ add người dùng vào group và gửi tin nhắn broadcast đến tất cả các thành viên trong nhóm là người dùng ấy đã tham gia group. Tính năng này giúp quản lý người dùng có trong group, cho các người dùng khác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

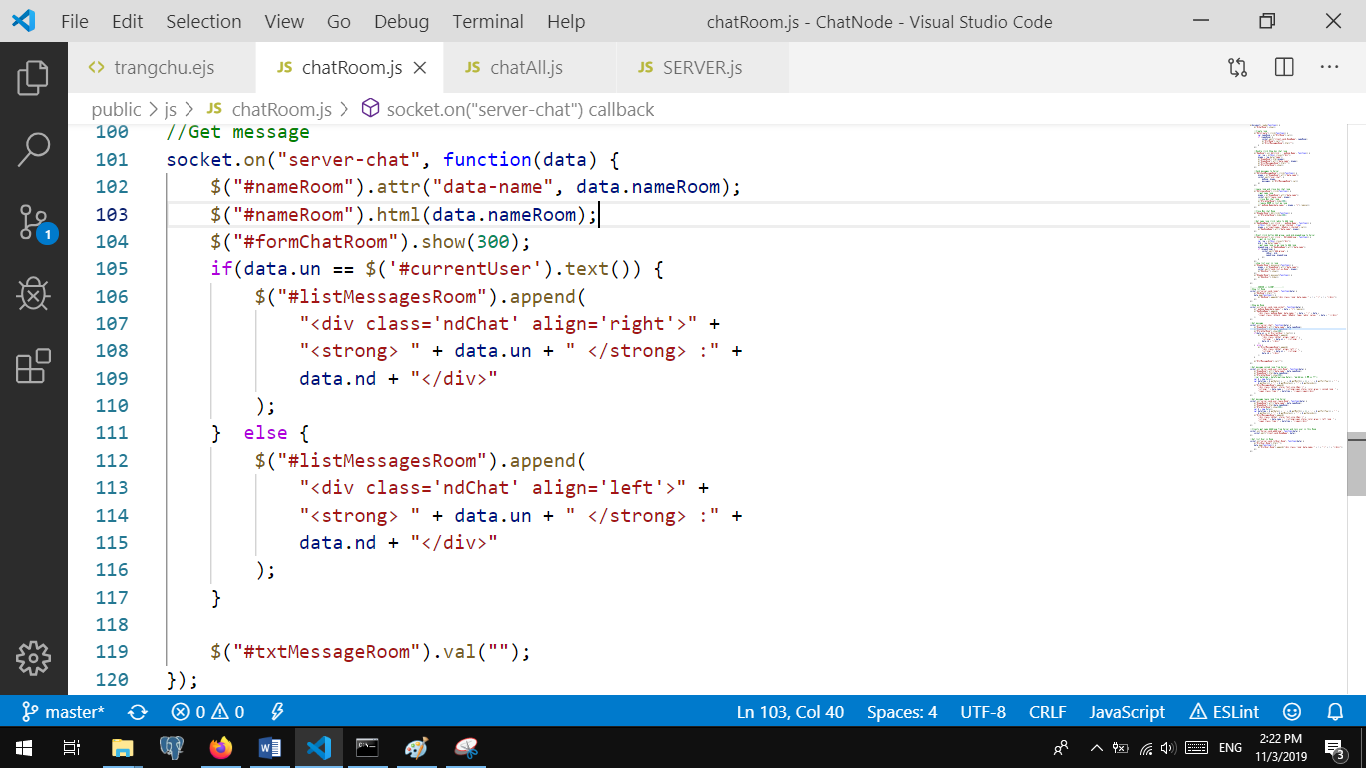


Việc gửi tin nhắn đến group, được xử lý tương tự như gửi tin nhắn message, thay vì gửi tin nhắn đến socket#id của người nhận, ở đây sẽ gửi message đến socket#id của room được tạo.

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 namespace “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.

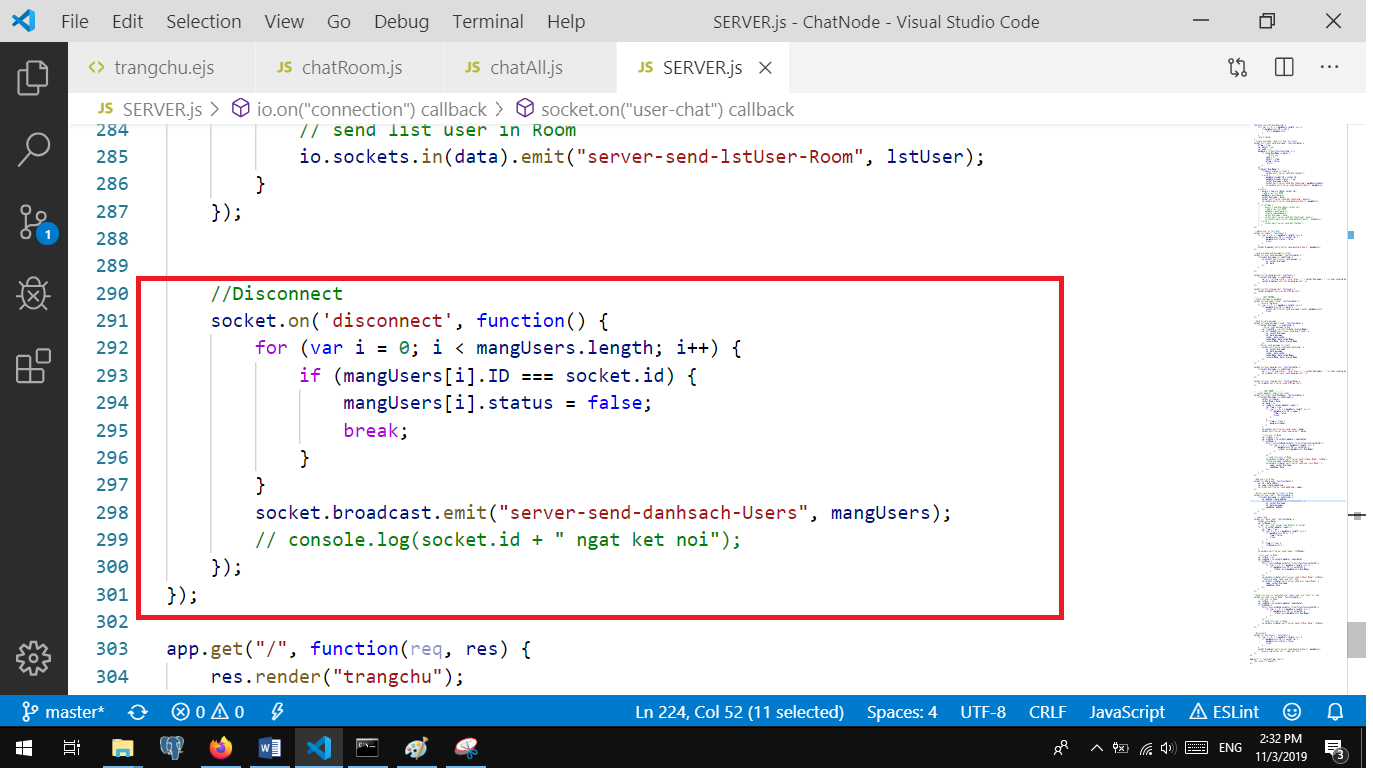




**6.5 Disconnect**

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 mangUsers 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.

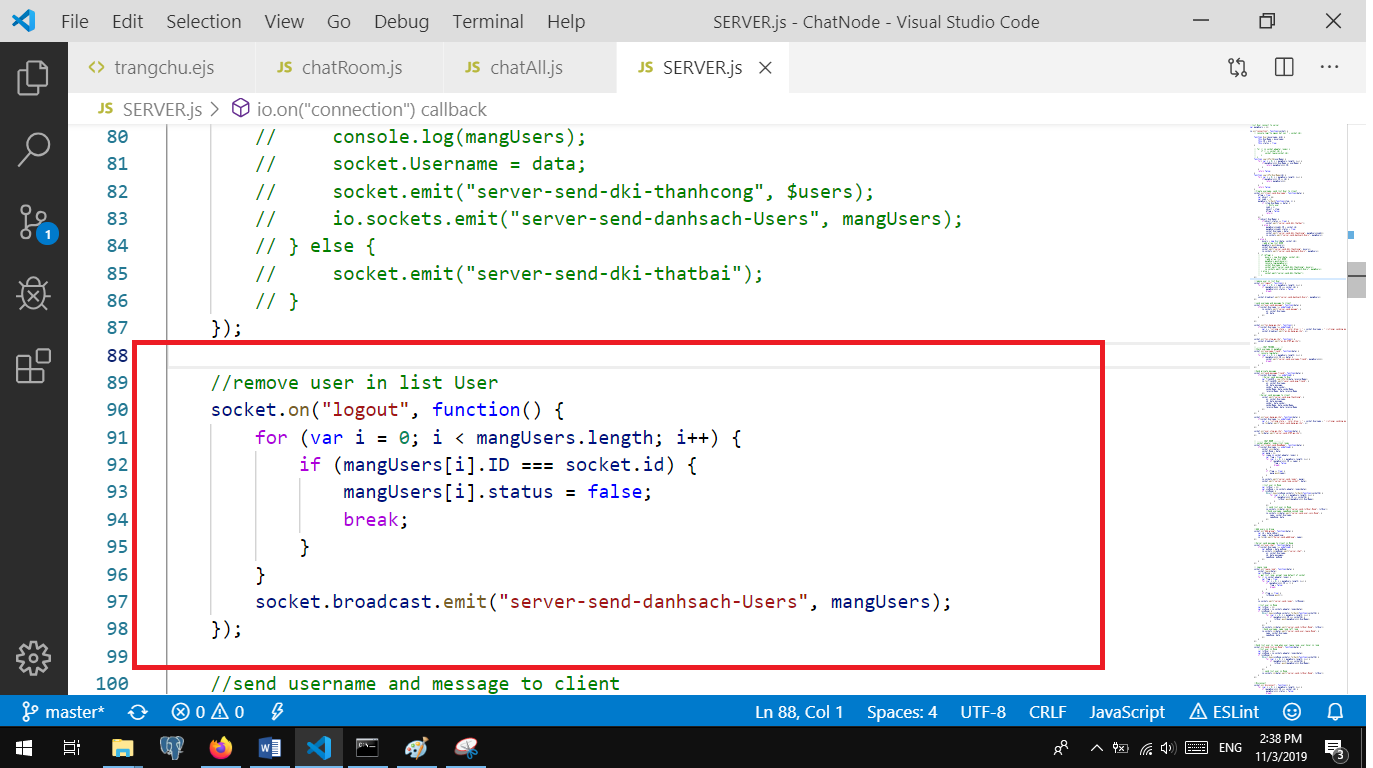


**6.6 Logout**

Cũng tương tự như disconnection server sẽ duyệt mangUsers 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.



**CHƯƠNG 7: 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 nhỏ chỉ cần một ít thông tin ngay tại thời điểm đó mà không cần phải 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>