**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? 6

1.2.2 Socket hoạt động như thế nào? 7

1.2.3 Phân loại Socket 7

a) Stream Socket 7

b) Datagram Socket 8

1.3 Khái niệm WebSocket 9

1.4 Socket.IO 10

1.5 Các tính năng chính của Socket.IO 11

1.5.1 Reliability 11

1.5.2 Auto-reconnection support 11

1.5.3 Disconnection detection 11

1.5.4 Binary support 12

1.5.5 Multiplexing support 12

1.5.6 Room support 12

1.6 Cấu hình triển khai để sử dụng Socket.IO 13

1.7 Tại sao nên chọn Socket.IO 13

CHƯƠNG 2: CƠ CHẾ HOẠT ĐỘNG CỦA SOCKET.IO 14

2.1 Khai báo sử dụng socketio 14

2.2 Cơ chế lắng nghe, truyền dữ liệu của Socket.IO 15

2.3 Socket.IO Broadcasting 18

CHƯƠNG 3: ROOMS AND NAMESPACES 20

3.1 Default namspace 20

3.2 Custom namespace 20

3.3 Rooms 21

3.3.1 Joining and Leaving 21

3.4 Default Room 22

3.5 Disconnection 22

3.6 Gửi tin nhắn từ bên ngoài 22

CHƯƠNG 4: SO SÁNH SOCKET.IO VỚI WEBSOCKET 24

CHƯƠNG 5: USER GUIDE 26

     5.1   Cài đặt môi trường 26

     5.2   Khởi động ứng dụng 28

CHƯƠNG 6: DEMO ỨNG DỤNG WEB CHAT SỬ DỤNG CÔNG NGHỆ

SOCKET.IO 30

6.1 Register vào ứng dụng 30

6.1.1 Register thất bại 31

6.1.2 Register thành công 32

6.2 Private Message 33

6.3 Chat All(Broadcasting) 36

6.4 Rooms 38

6.5 Disconnect 44

6.6 Logout 45

CHƯƠNG 7: KẾT LUẬN 46

TÀI LIỆU THAM KHẢO 48

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**

1. 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.
2. 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.
3. 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.
4. 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 thuật ngữ dành cho ứng dụng, đặc biệt là 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 tính năng realtime ở hầu hết các ứng dụng hiện nay, mà tiêu biểu nhất chính là qua các ứng dụng nhắn tin, hoặc bảng tin newsfeed trên Facebook.

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**

Hiện nay có nhiều cách và công nghệ để triển khai một ứng dụng realltime, mà trong đó Socket đang được xem là công nghệ tốt nhất hiện nay.

Khái niện [Socket](https://en.wikipedia.org/wiki/Socket) 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. 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 sẽ cho phép người dùng kết nối các máy tính truyền tải và nhận dữ liệu từ máy tính thông qua mạng mà vẫn duy trì kết nối với nhau. Hiểu đơn giản, socket là thiết bị truyền thông hai chiều gửi và nhận dữ liệu từ máy khác.

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

Socket hiện là cách thức triển khai ứng dụng realtime đơn giản, nhanh, hiệu quả và tốt nhất hiện nay. 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 hiệu cổng của ứng dụng còn lại.

Trong đó :

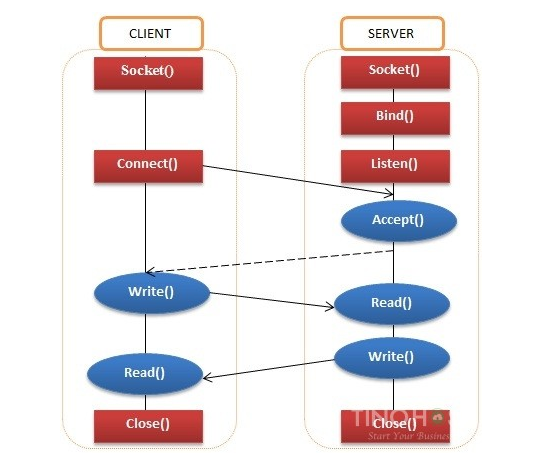
+ 2 ứng dụng có thể nằm trên cùng một máy tính.

+ 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.**

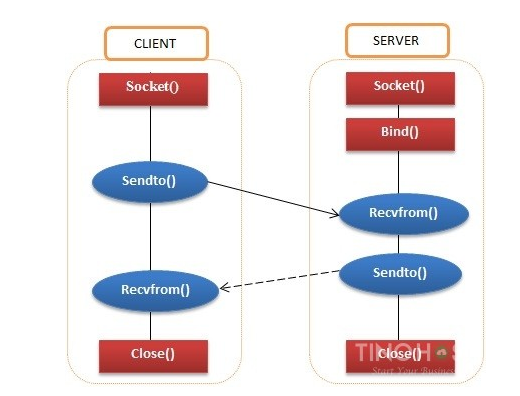


**Ưu điểm:** Có thể dùng để liên lạc theo mô hình client và sever. Nếu là mô hình client /sever thì sever lắng nghe và chấp nhận từ client. Giao thức này đảm bảo dữ liệu được truyền đến nơi nhận một cách đáng tin cậy, đúng thứ tự nhờ vào cơ chế quản lý luồng lưu thông trên mạng và cơ chế chống tắc nghẽn.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



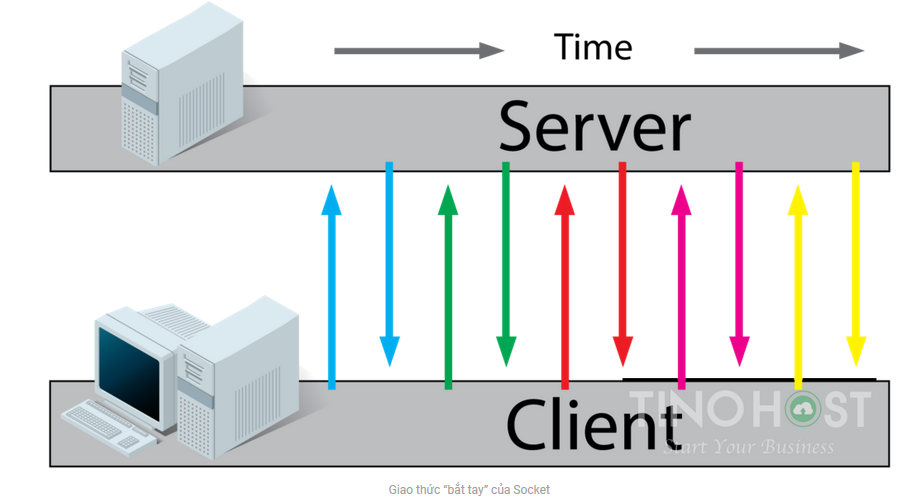
**Ưu điểm:** Do không yêu cầu thiết lập kết nối, không phải có những cơ chế phức tạp nên tốc độ giao thức khá nhanh, thuận tiện cho  các ứng dụng truyền dữ liệu nhanh như chat, game…..

**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 sẽ thiết lập kết nối long polling, sau đó nâng cấp lên các băng thông vận chuyển dữ liệu tốt hơn nhanh hơn.

Ưu điểm:

* + - Cải thiện performance phía server về mặt thời gian, tối ưu được băng thông.

Nhược điểm:

* + - Một số proxies chặn các kết nối websocket.
    - Tính năng firewall trên các máy tính và các phần mềm diệt virus đôi khi chặn kết nối của websocket.

### **Auto-reconnection support**

### Khi máy chủ server bị mất kết nối, 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 : tùy chỉnh số lần thử kết nối lại.

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

### **Disconnection detection**

### Với cơ chế heartbeat được triển khai ở tầng Engine.IO, cho phép các máy phía client, và máy phía server nhận 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**

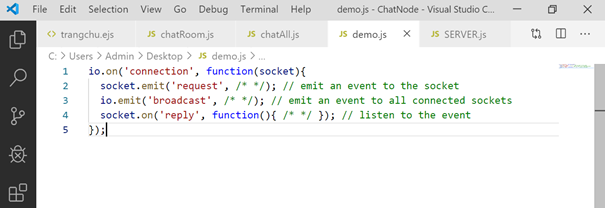
### Socket.IO cho phép tạo ra nhiều Namespaces, hoạt động như các kệnh liên lạc riêng biệt(private), nhưng chia sẻ cùng một kết nối bên dưới.

### **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 có thể dàng sử dụng thông qua việc gọi các api có sẵn của socket.io:



### 

# Cấu hình triển khai 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 nhằm quản lý các request nhận được từ phía các client thông qua một port nhất định. 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 socket.io vào server bằng lệnh của nodejs. Nếu sử dụng php thì phải cài thêm những package khác, hoặc phải chuẩn bị riêng server để chạy socket.io.
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 được nhúng trên phần giao diện người dùng để triển khai socket.io nhằm việc quản lý việc gửi và nhận request đến server cũng thông qua một port nhất định.

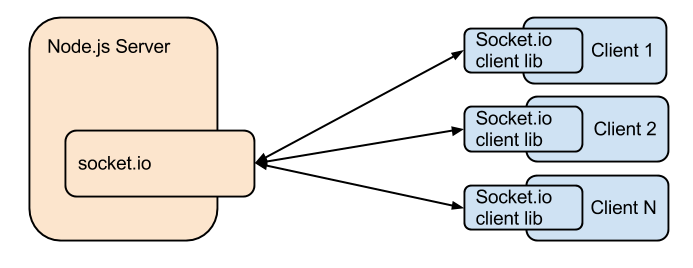
## 1.7 Tại sao nên chọn Socket.IO

* 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.
* Socket.IO khắc phục được hầu hết các nhược điểm của websocket như:
  + Socket.IO có thể hoạt động tốt cả trong môi trường có cài đặt Firewall, Antivirus, Proxy và Load Balancer.
  + Socket.IO có cơ chế phát hiện và khôi phục lại khi các client bị mất kết nối với server.
  + Socket.IO hỗ trợ đa kết nối và cả hỗ trợ gửi dữ liệu dạng binary như: hình ảnh, âm thanh,....

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

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

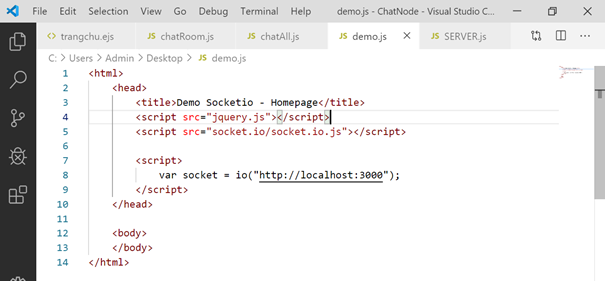
Cơ chế hoạt động của một ứng dụng realtime đó là thông qua server để lắng nghe (listen) các sự kiện, mà tiêu biểu nhất sự kiện gửi một request từ phía client và gửi trả lại responsive thích hợp. Vì vậy cần cài khai báo sử dụng socketio ở cả phía server và client.



Code khai báo sử dụng socketio ở server:

### 

Code khai báo sử dụng socketio ở phía client (được nhúng vào giao diện html ở client)

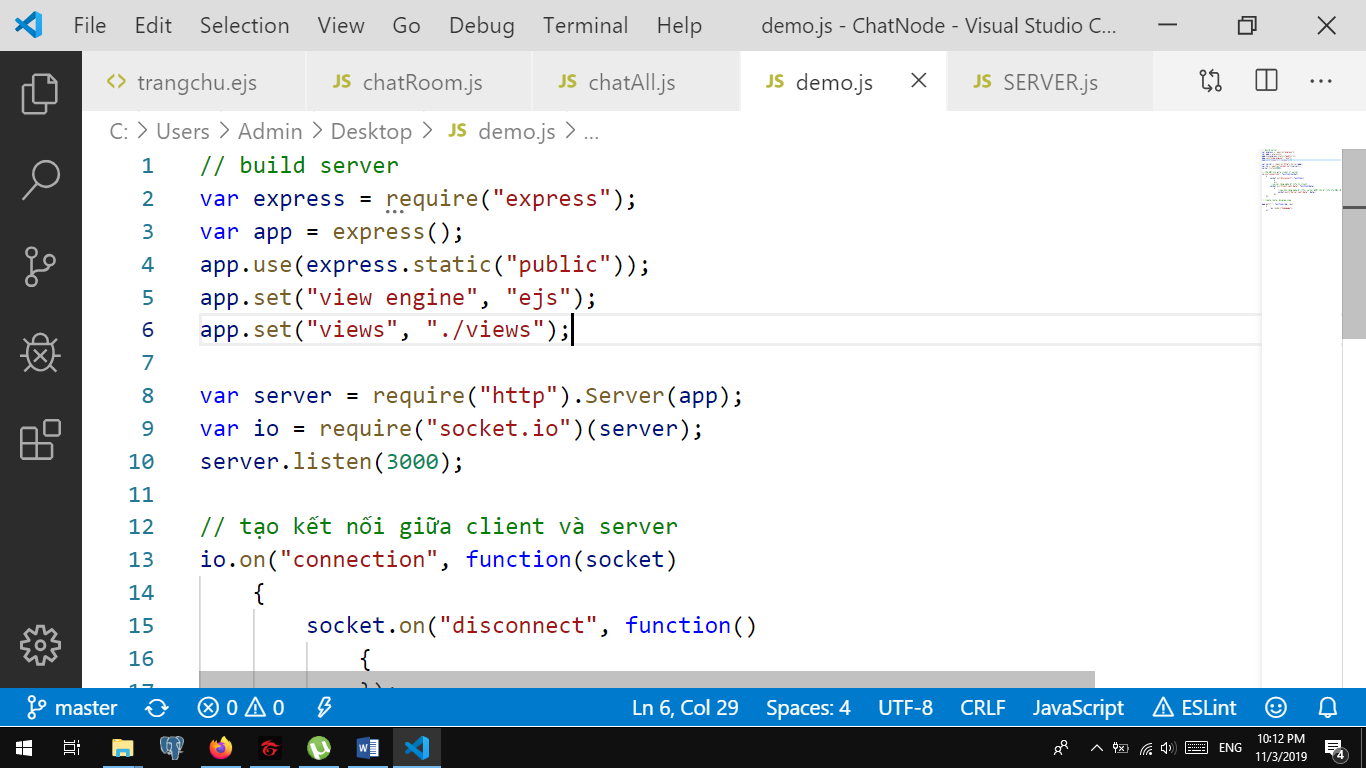


### **2.2 Cơ chế lắng nghe, truyền dữ liệu của Socket.IO**

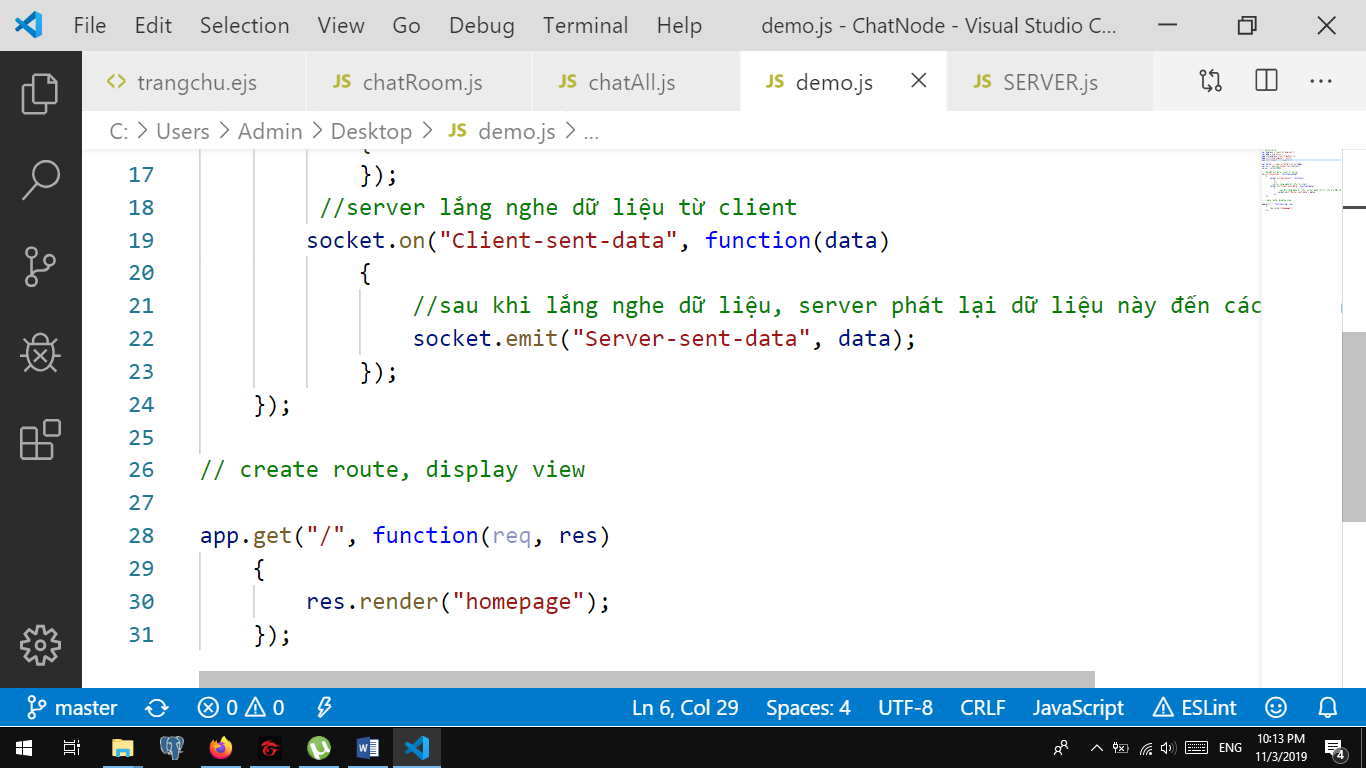
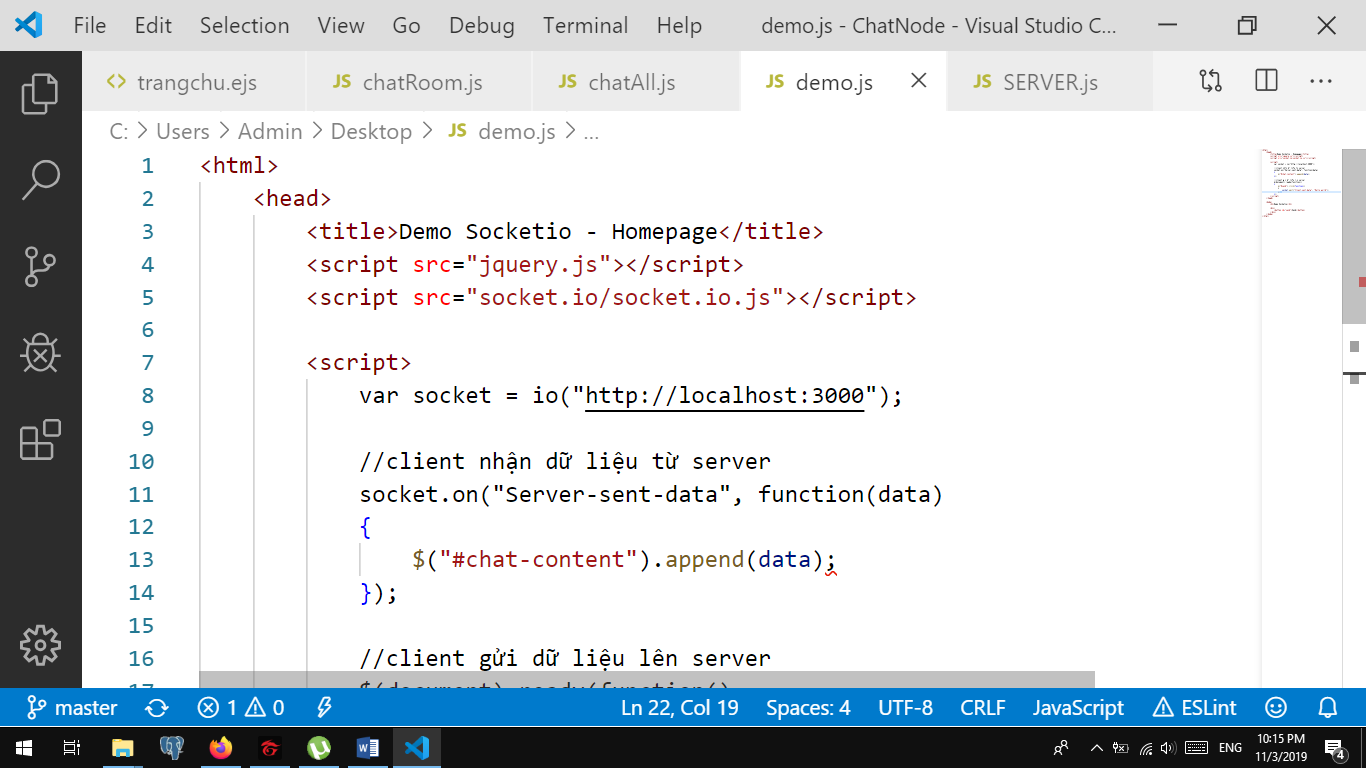
Để lắng nghe data, ta sử dụng câu lệnh socket.on(), để phát dữ liệu thì sử dụng lệnh socket.emit() .

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 và được ví như việc lắng nghe và hồi đáp luôn đi kèm với nhau.

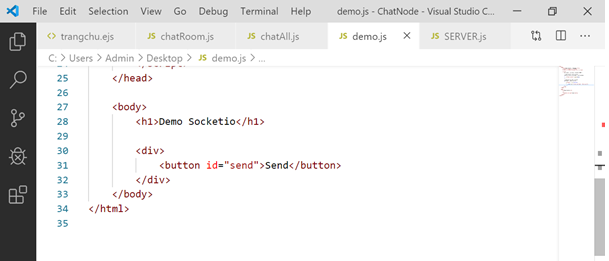
Code phía server



Tạo

Code phía client

### 



Lưu ý: socket.on và socket.emit có parameter thứ 1 là tên của sự kiên. Tên của sự kiện có thể là tên bất ký, tuy nhiên đễ truyền và nhận dữ liệu đúng và chính xác thì tên sự kiện cần phải giống nhau.

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

Client send

### 

Server listen



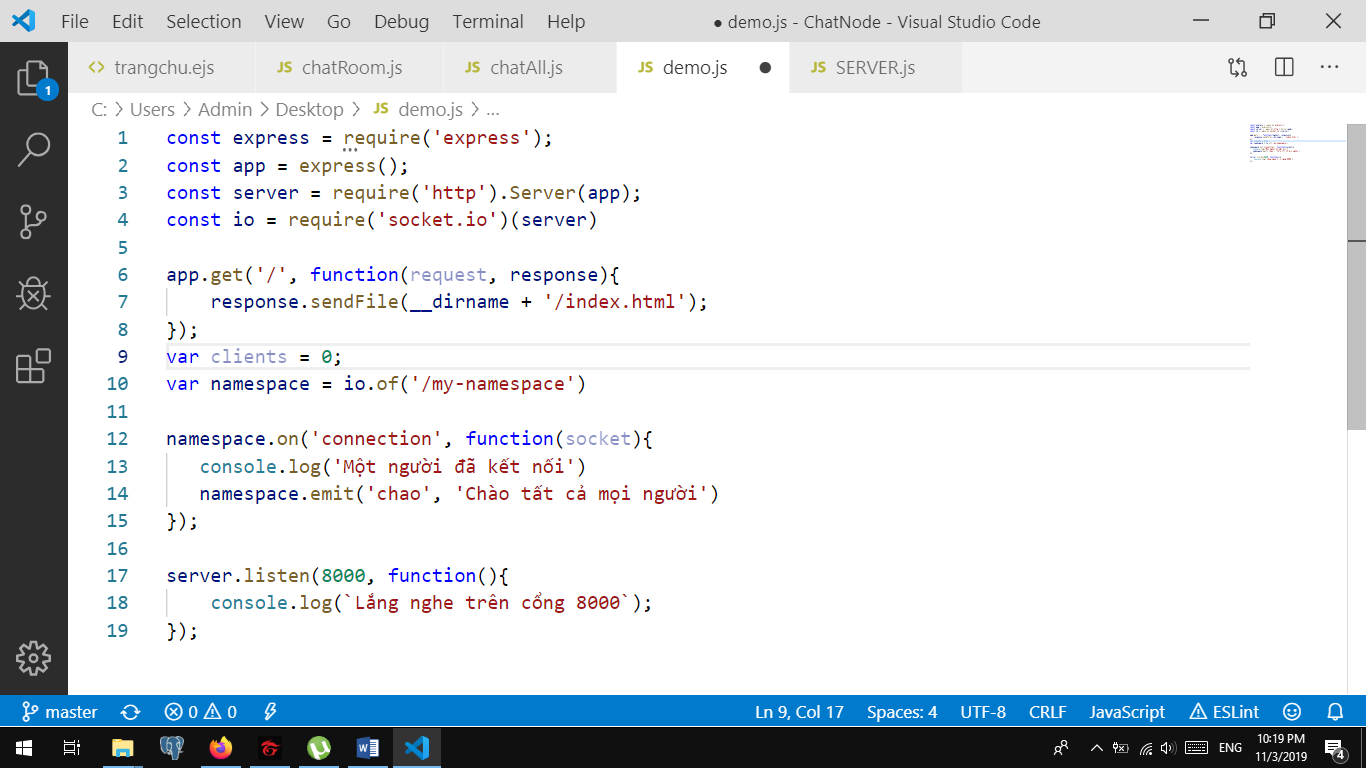
Lưu ý: Khi sử dụng callback option, nên truyền data dưới dạng json, hoặc object, hoặc string. Khi emit một sự kiện ở phía client, không cần phương thức form action để gửi một request lên server nữa, tất cả đều được xử lý dưới js được viết theo kiểu callback function và tích hợp sẵn trong câu lệnh socket.emit().

# 2.3 Socket.IO Broadcasting

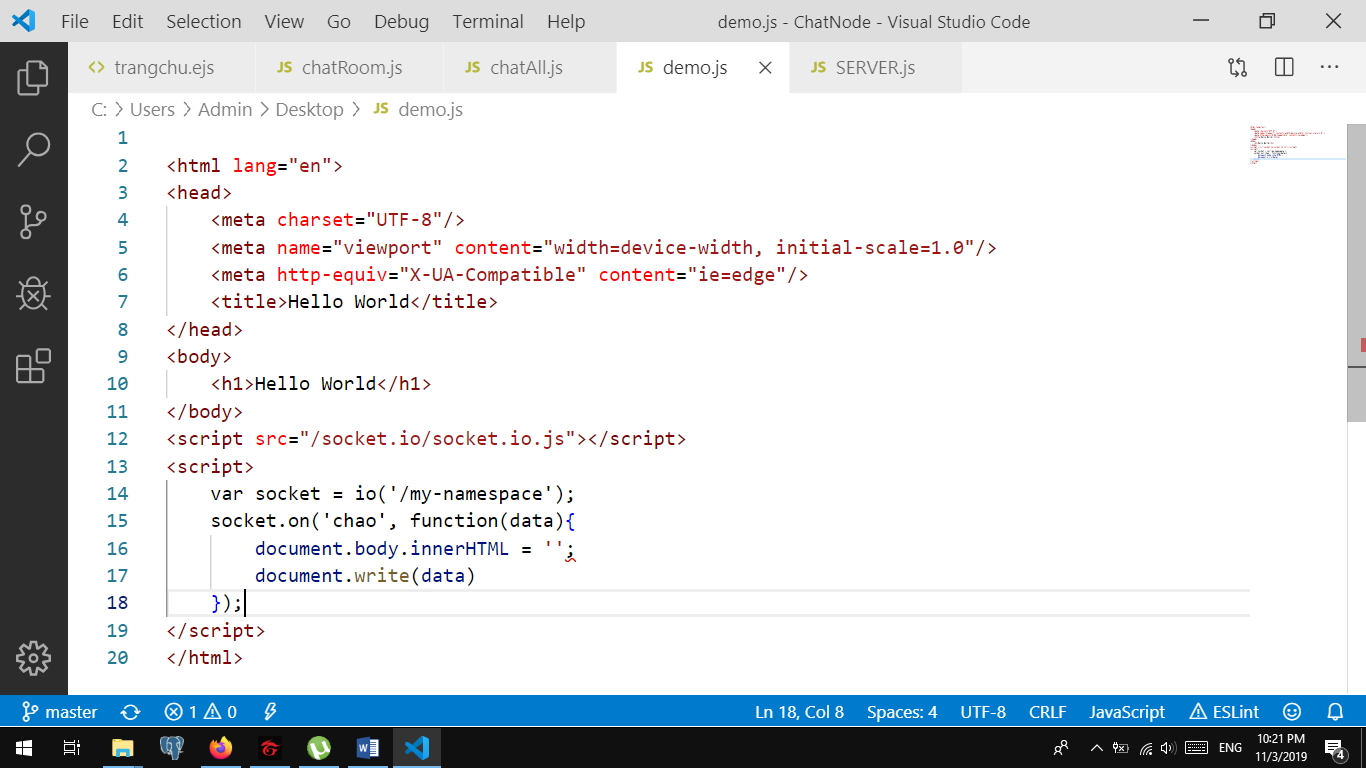
Broadcasting nghĩa là gửi một tin nhắn đến tất cả người dùng đang kết nối. Broadcasting có thể thực hiện ở nhiều cấp độ. Chúng ta có thể gửi tin nhắn đến tất cả người dùng đang kết nối, khách hàng có tên bất kỳ và cả khách hàng ở trong một phòng cụ thể. Để phát một sự kiện cho tất cả các khách hàng,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

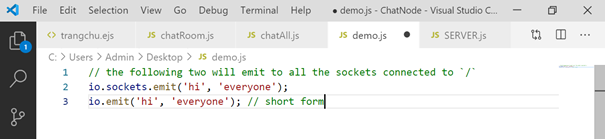
**CHƯƠNG 3: ROOMS AND NAMESPACES**

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.

## 3.1 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:



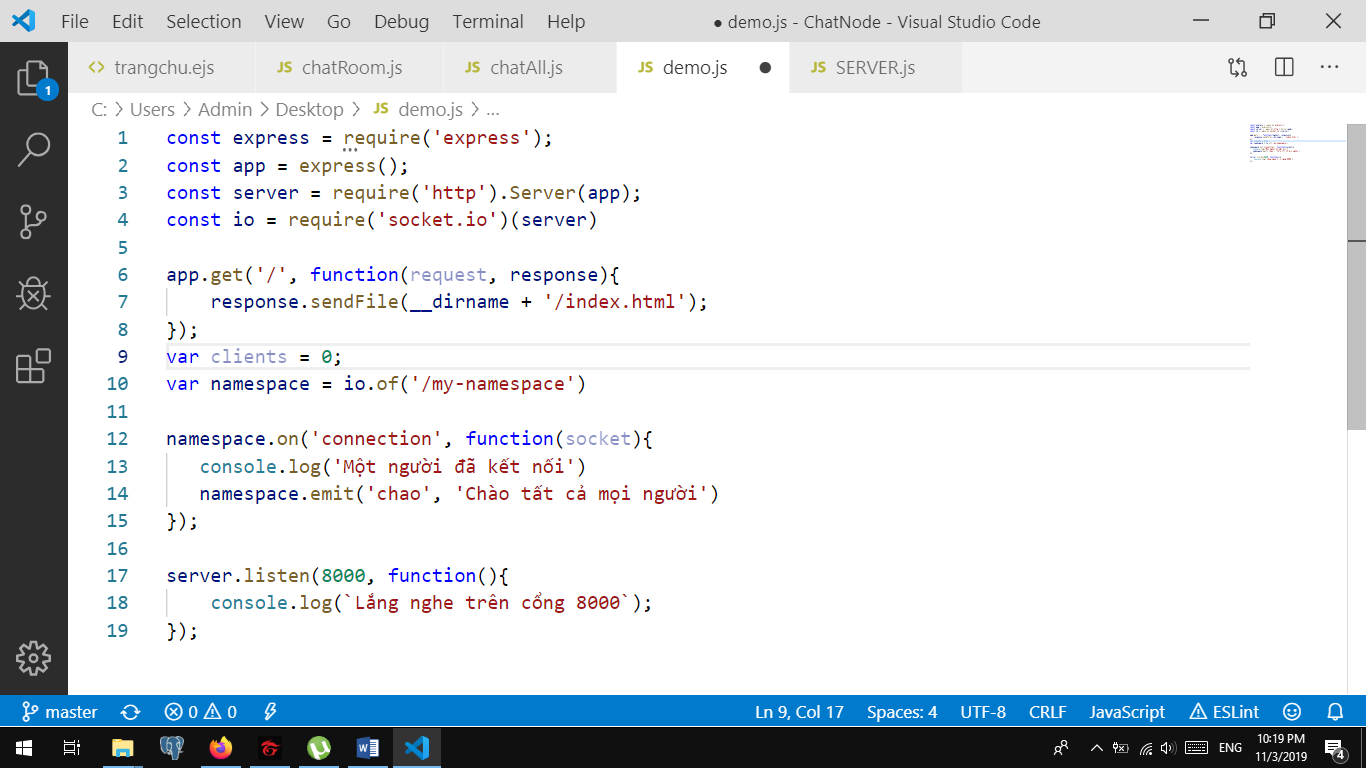
Mỗi Namespace sẽ gửi(emit) một sự kiện kết nối đến server, server sẽ nhận mỗi đối tượng socket gửi đến với dạng tham số.

Ví dụ

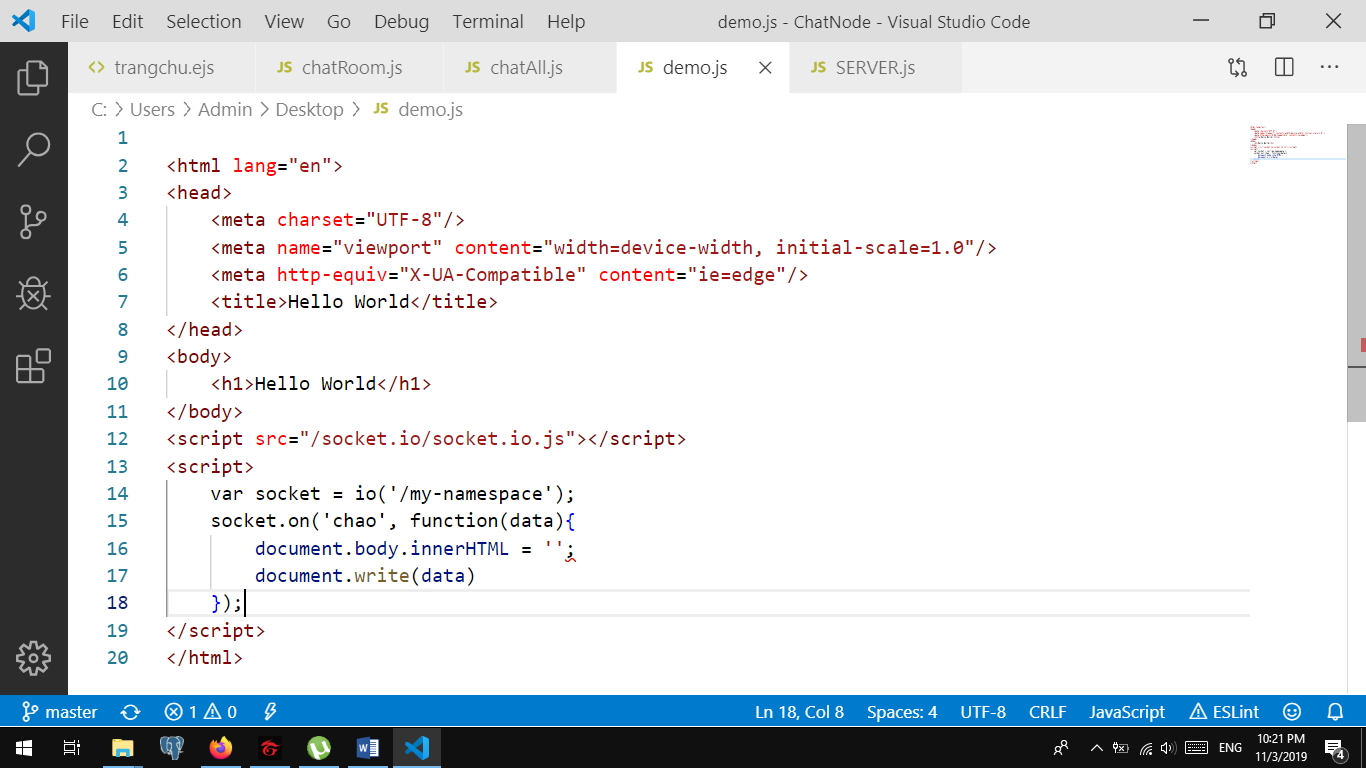
****

**3.2 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”

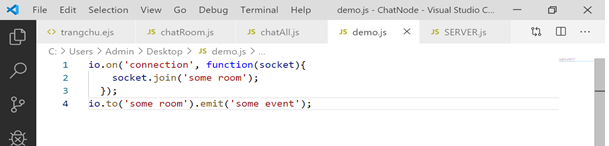
**3.3 Rooms**

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

**3.3.1 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:



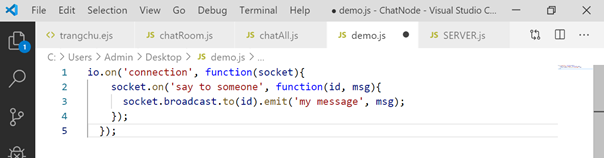
Để thoát khỏi room, sử dụng hàm leave(), với cú pháp tương tự join().

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

**3.4 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.



**3.5 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.

**3.6 Gửi tin nhắn từ bên ngoài**

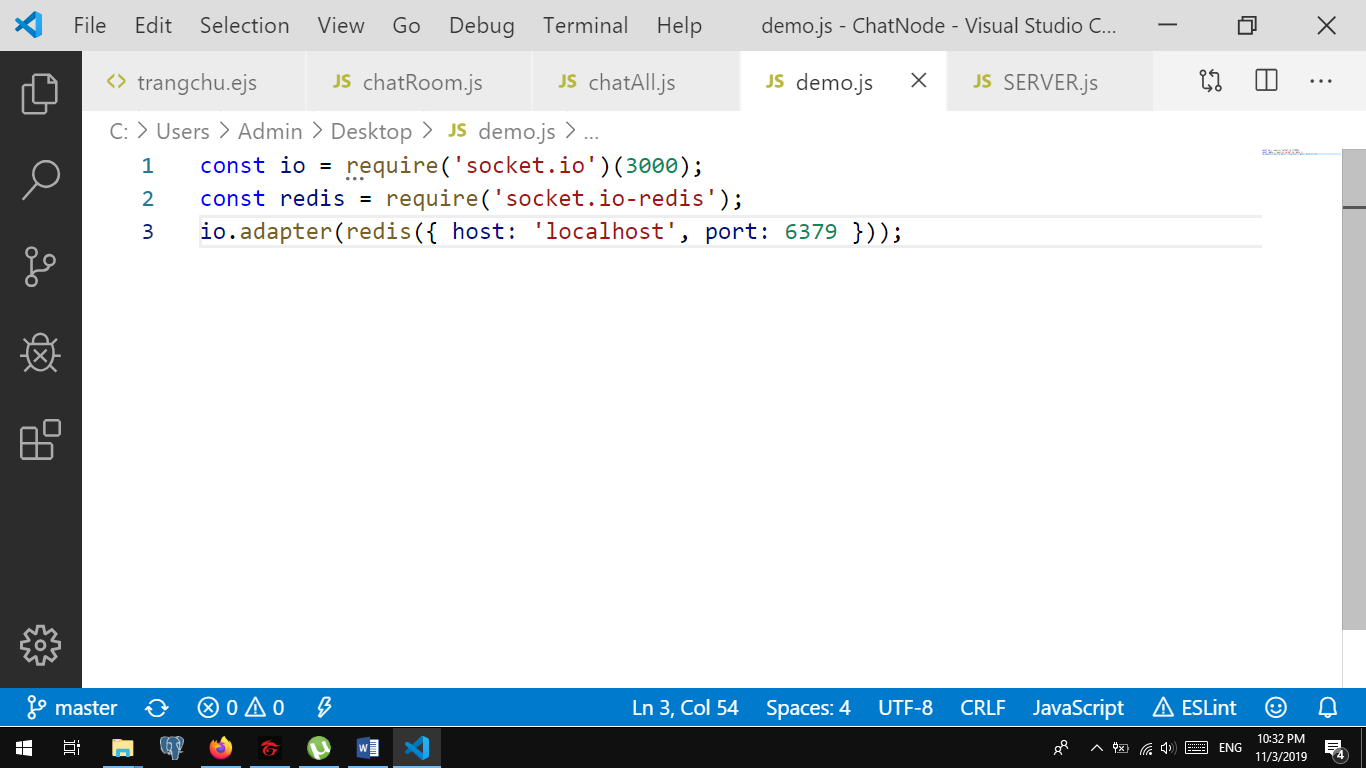
Trong một số trường hợp, chúng ta có thể muốn emit các events đến các sockets trong namespace/ rooms từ bên ngoài của socket.IO

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 4: 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:

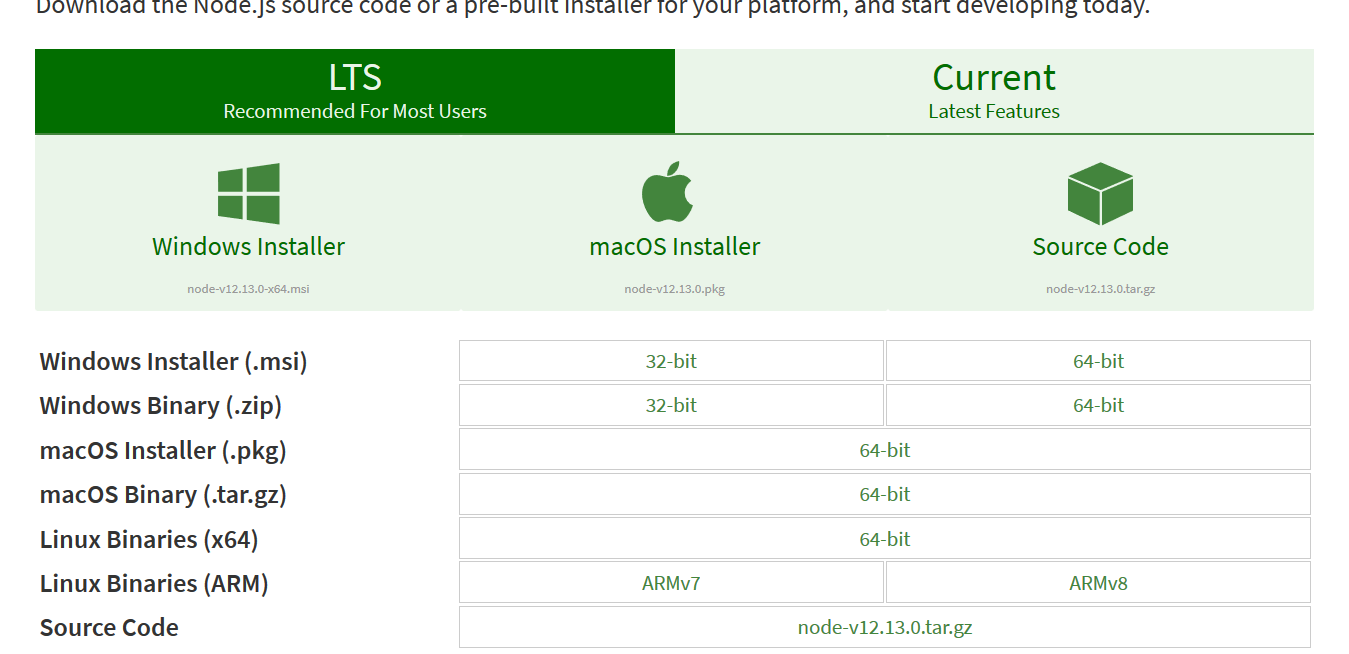
Giống nhau: Đều là công nghệ real-time, hoạt động trong một khoản thời gian mà người dùng có thể thao tác được ngay lập tức thời.

|  |  |
| --- | --- |
| WebSocket | Socket.IO |
| Là giao thức được thiết lập qua kết nối TCP | Là thư viện được dùng để hoạt động với websocket và các thuật toán dự phòng để tạo một kết nối realtime. |
| 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 5: USER GUIDE**

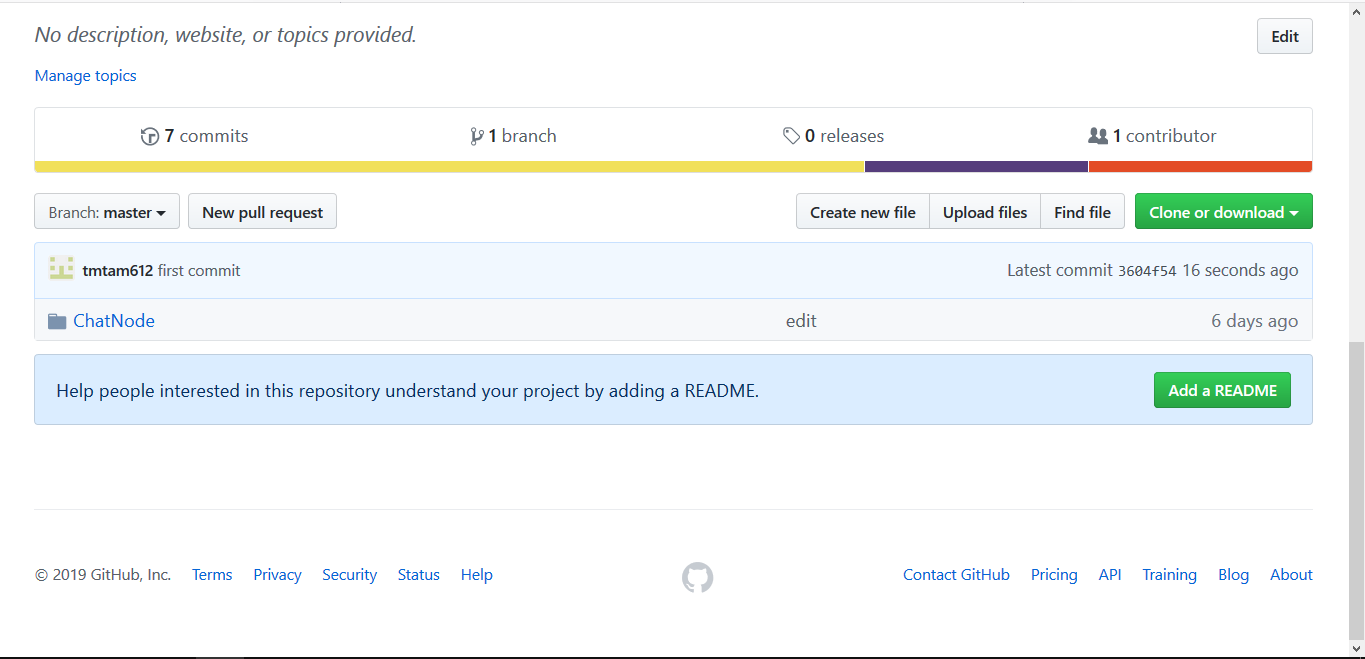
**5.1 Cài đặt môi trường**

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

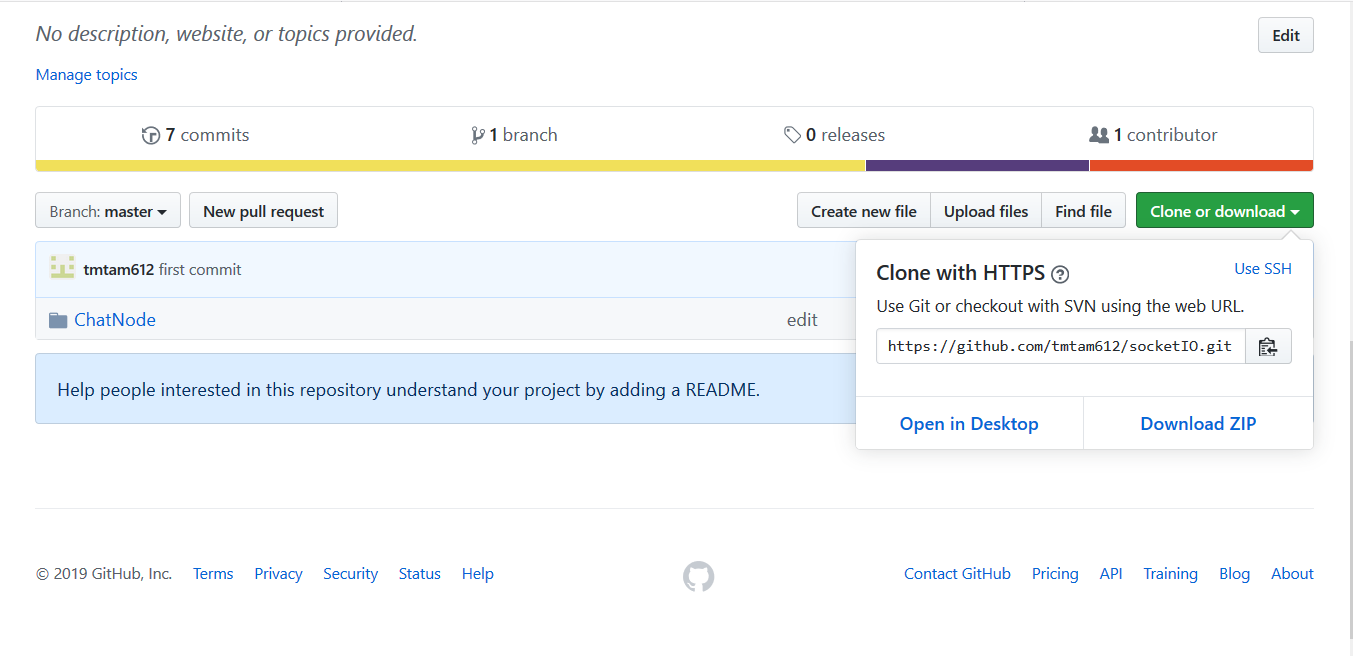


Download bản phù hợp với hệ điều hành của mình, và chạy cài đặt bình thường như những phần mềm ứng dụng khác.

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.

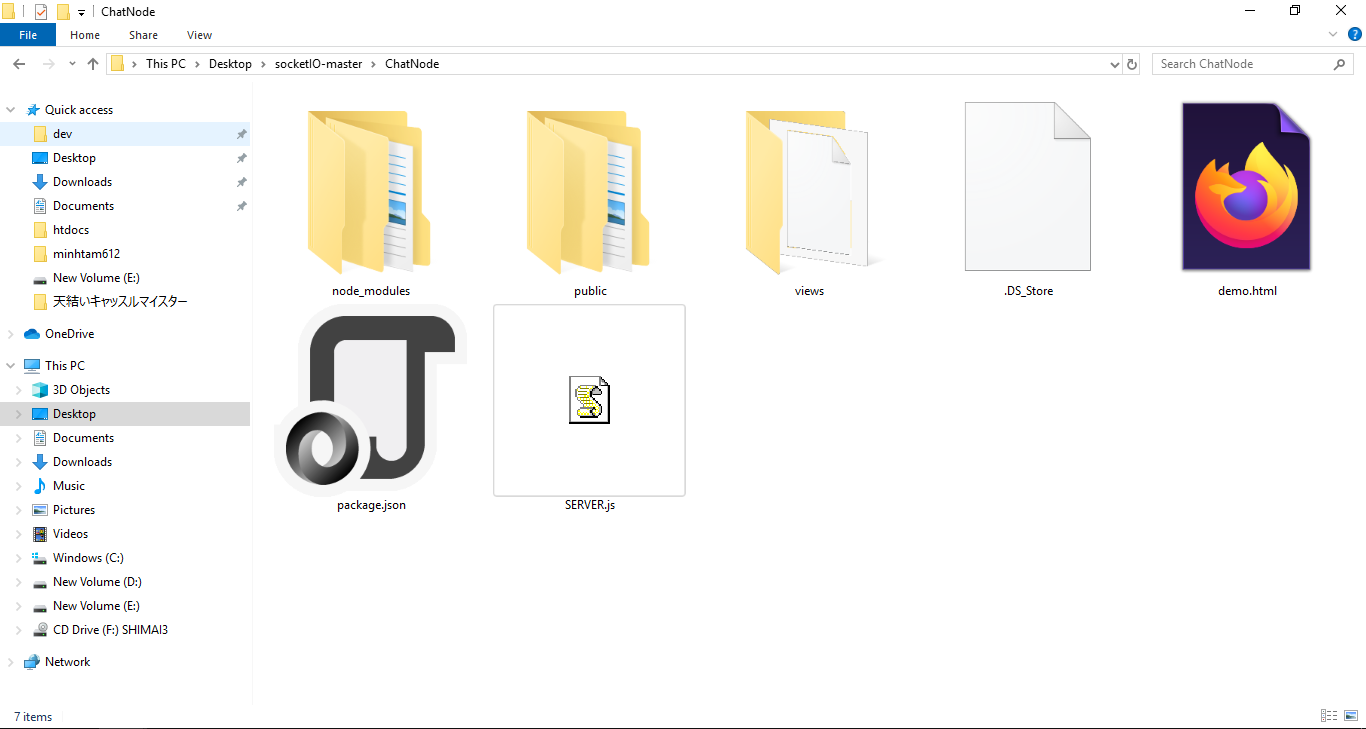


Nhấn vào nút clone or download, chọn tải về dưới dạng zip (Download Zip).

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



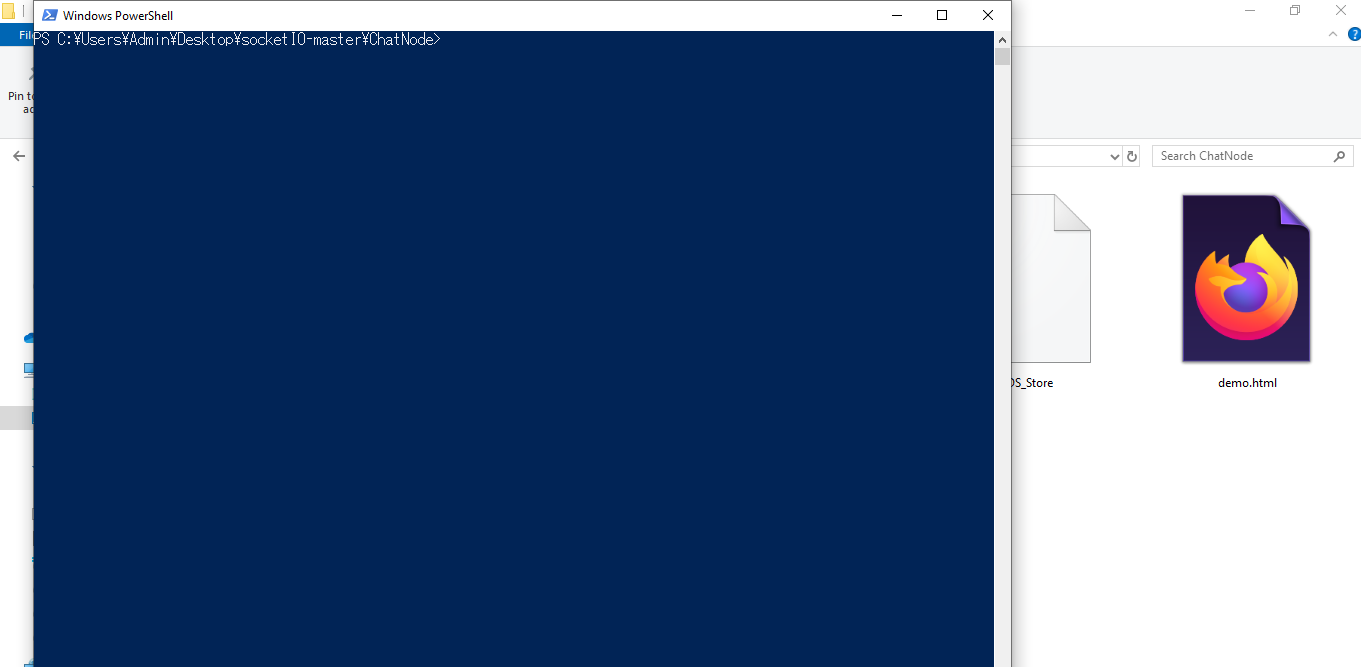
Truy cập vào thư mục ChatNode, có những file như sau là đã cài đặt môi trường thành công



**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, 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.

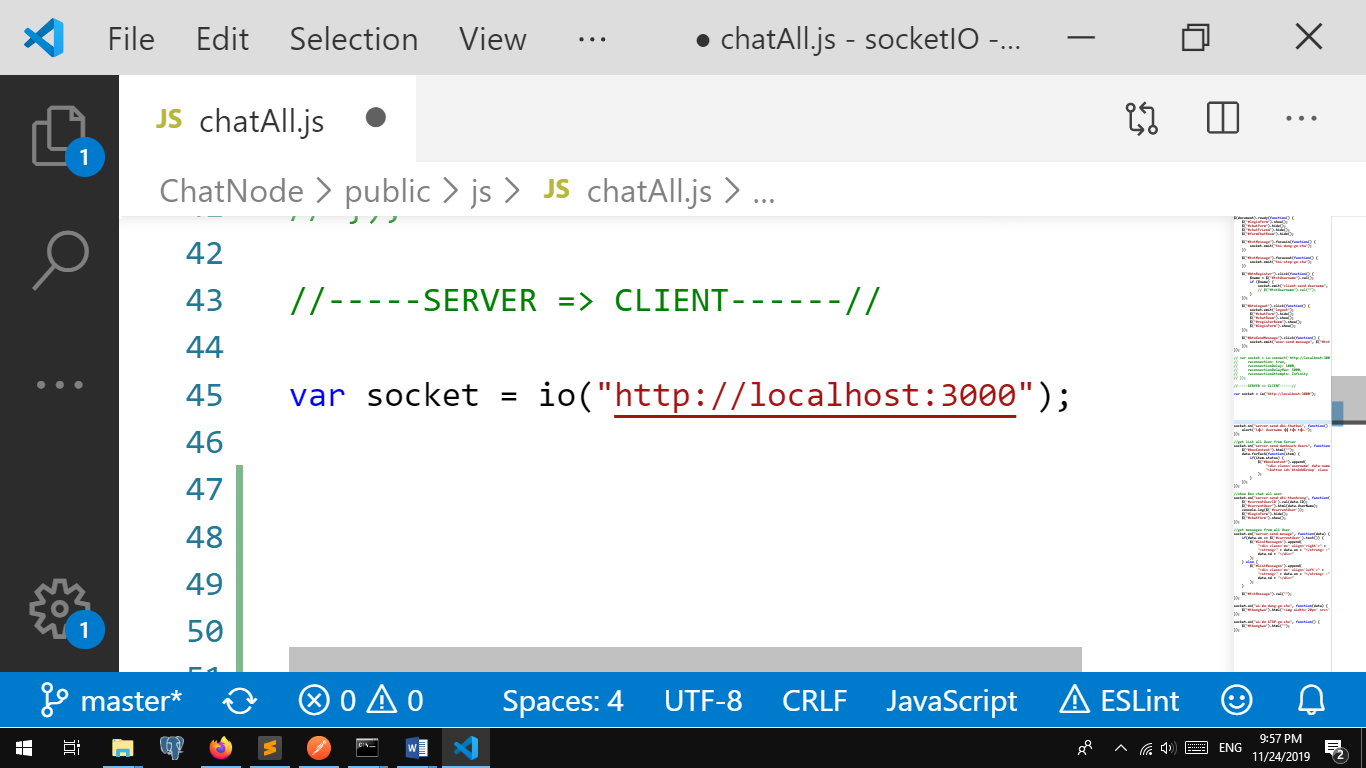


Nếu vẫn còn thắc mắc xin liên hệ gmail: [tmtam612@gmail.com](mailto:tmtam612@gmail.com)

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

**6.1 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



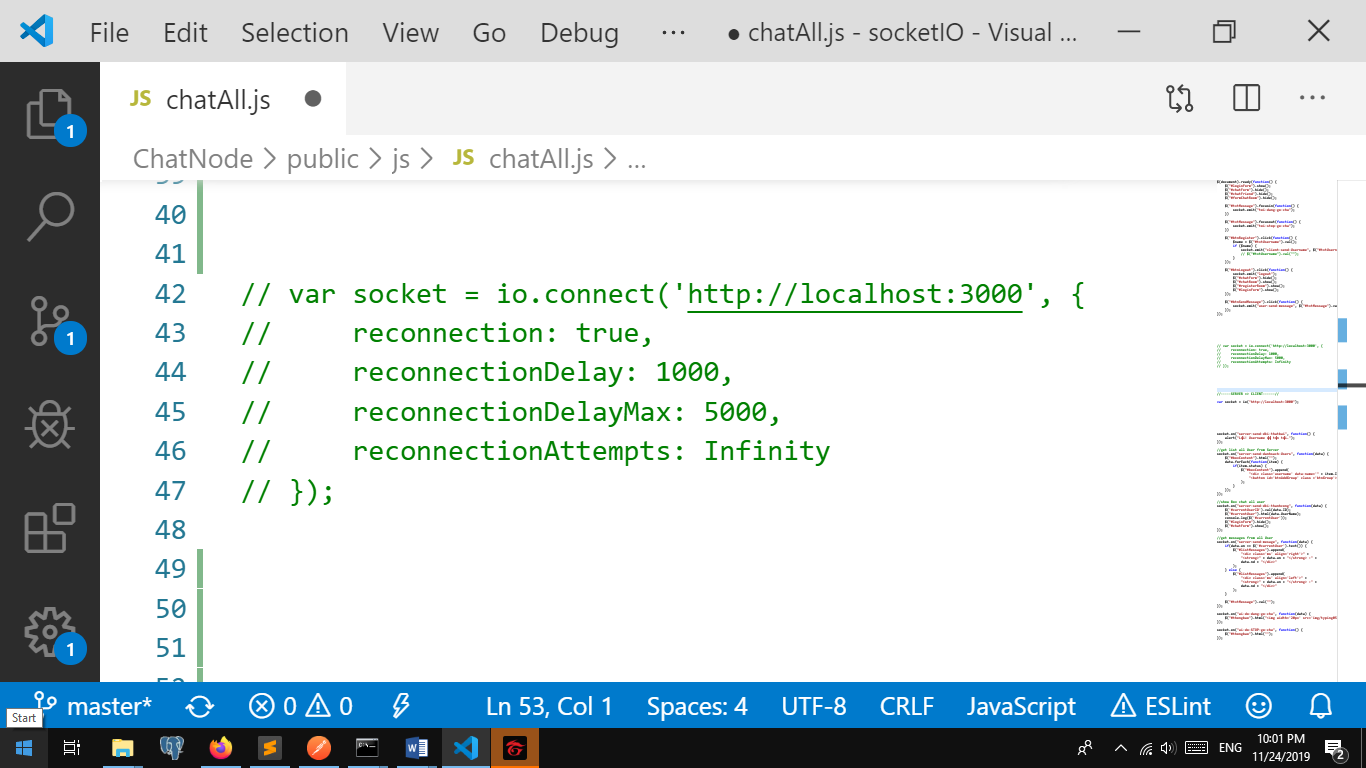
*Hình minh họa: tạo biến cục bộ socket*

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*

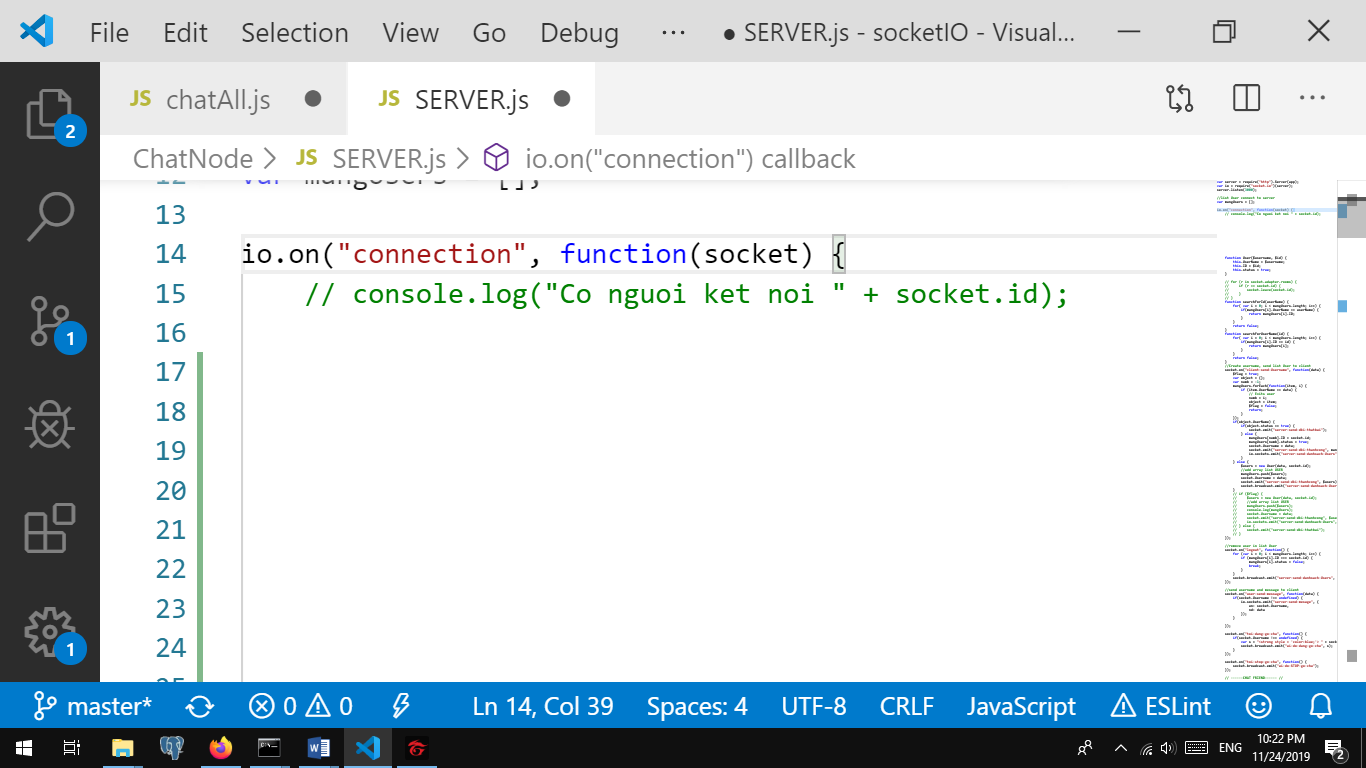
Cú pháp io.connect() gồm 2 phần:

* 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:

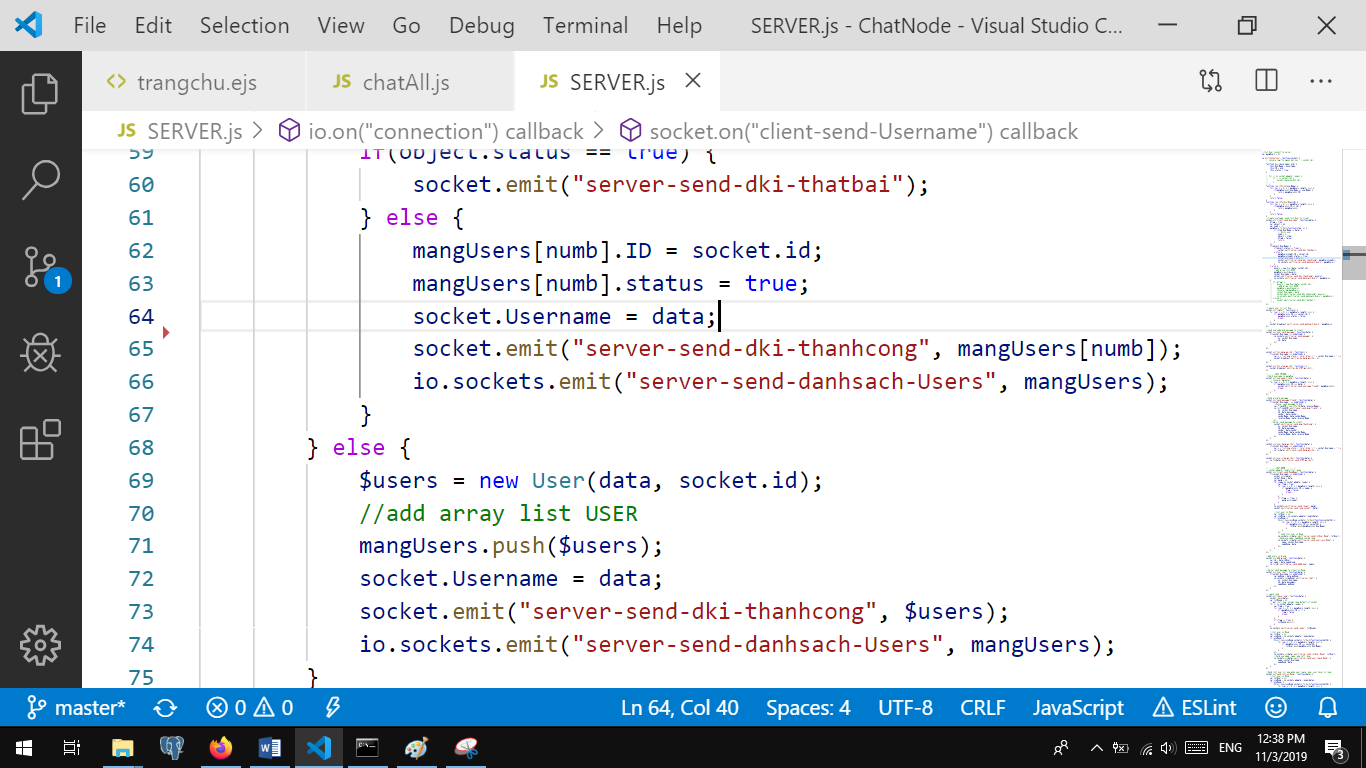
$(“#btnRegister”).click(): dùng để bắt sự kiện click của button register, khi người dùng nhấn nút button, client sẽ lấy name của người dùng bằng textbox có id là txtUsername qua cú pháp $(“#txtUsername”).val(), sau đó gửi đến server qua cú pháp emit().

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 mangUsers[] và gửi mangUser[] 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().

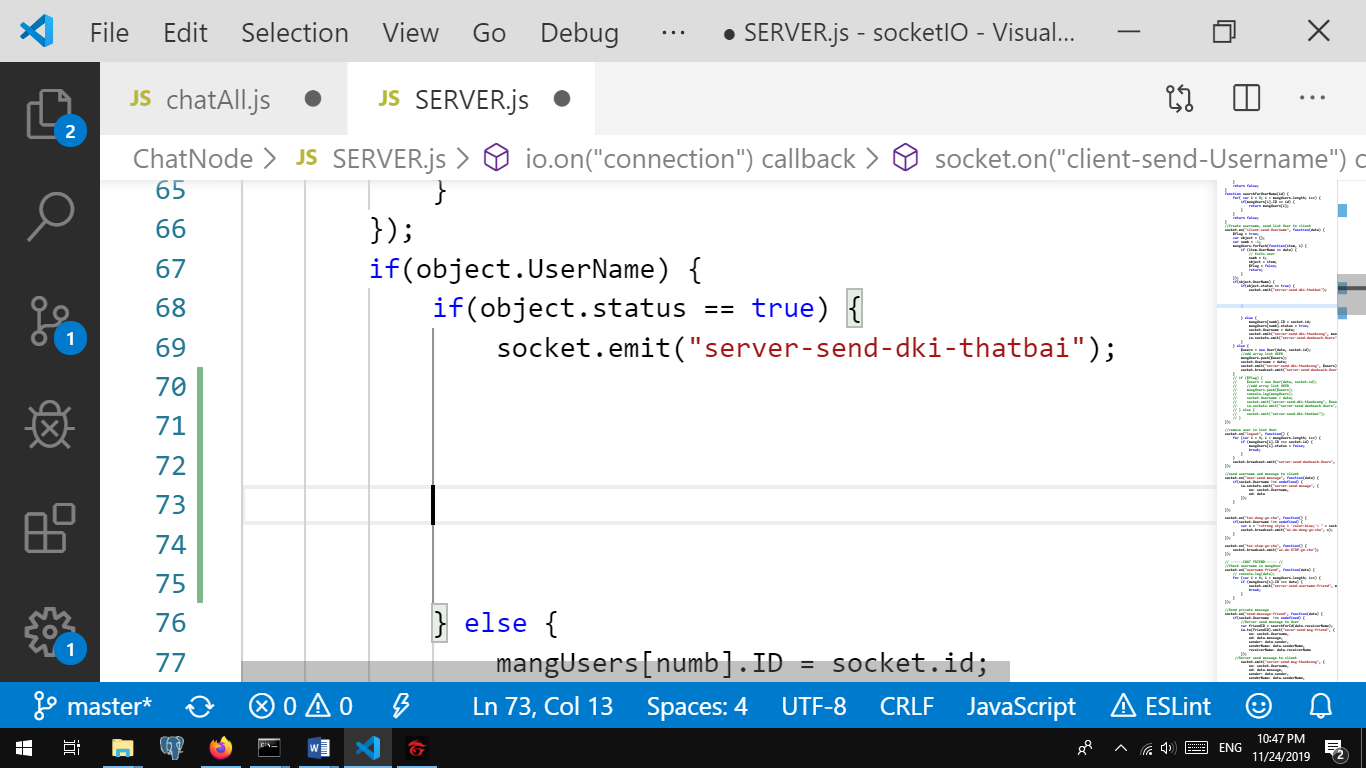
Cú pháp io.sockets.emit() gồm 2 tham số, cũng tương tự như cú pháp emit() thông thường, gồm 2 tham số, tham số thứ nhất là tên sự kiện, thứ hai là dữ liệu cần gửi. Nhưng khác với cú pháp emit() thông thường với đối tượng gọi là socket gửi đơn mục tiêu.

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 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 sự kiện có tên “server-send-dki-thatbai” để gửi thông báo đến client qua cú pháp gửi emit.

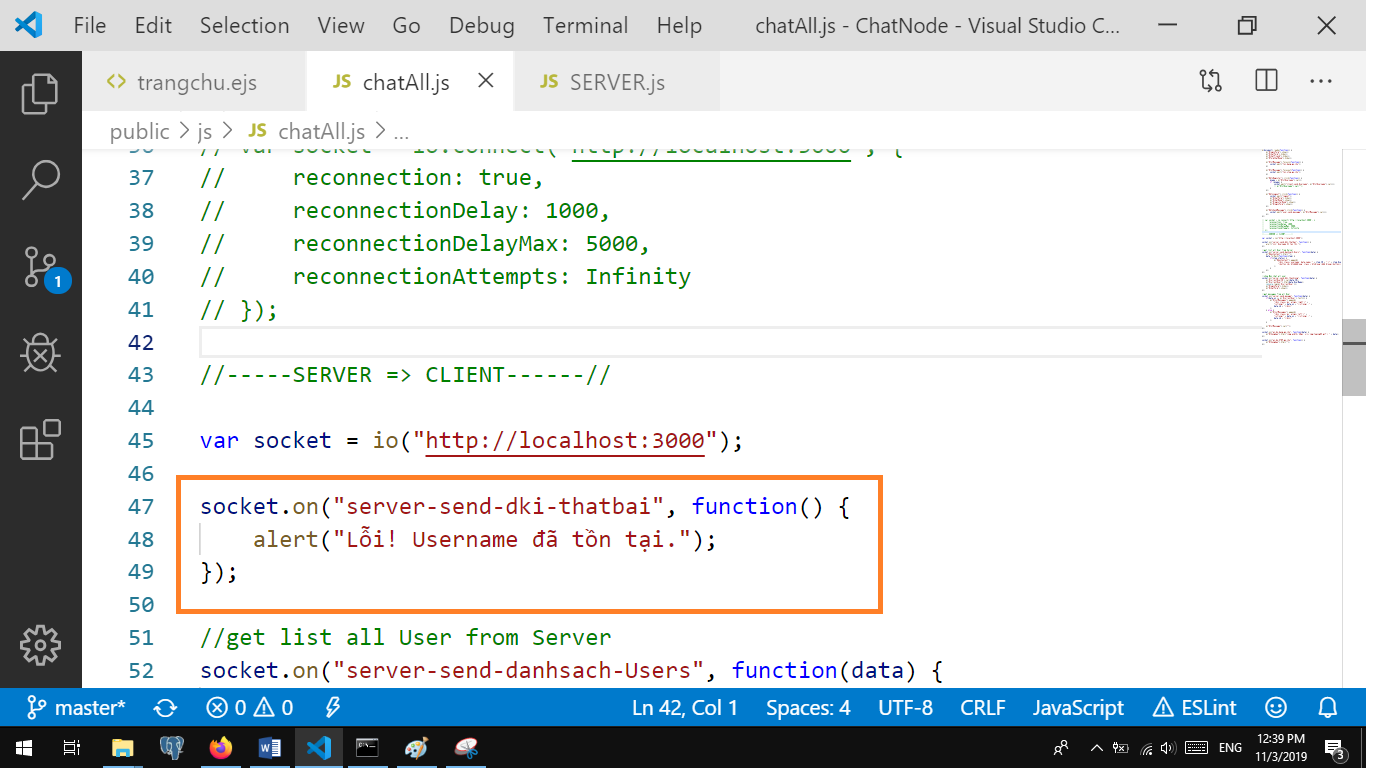


*Hình minh họa: sự kiện “server-send-dki-thatbai” gửi đi ở phía server*

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.



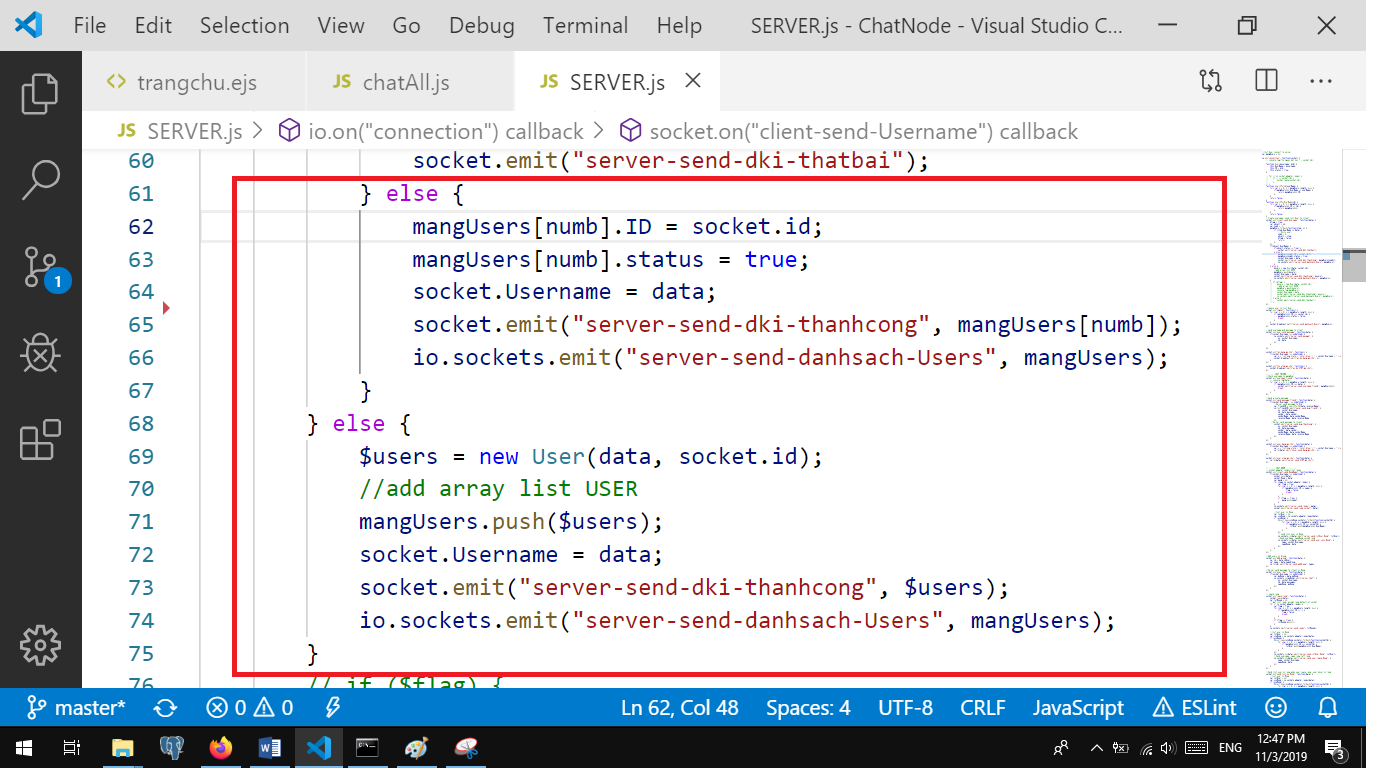
*Hình minh họa: sự kiện “server-send-dki-thatbai” nhận được ở phía client.*



*Hình minh họa giao diện người dùng*

**6.1.2 Register thành công**

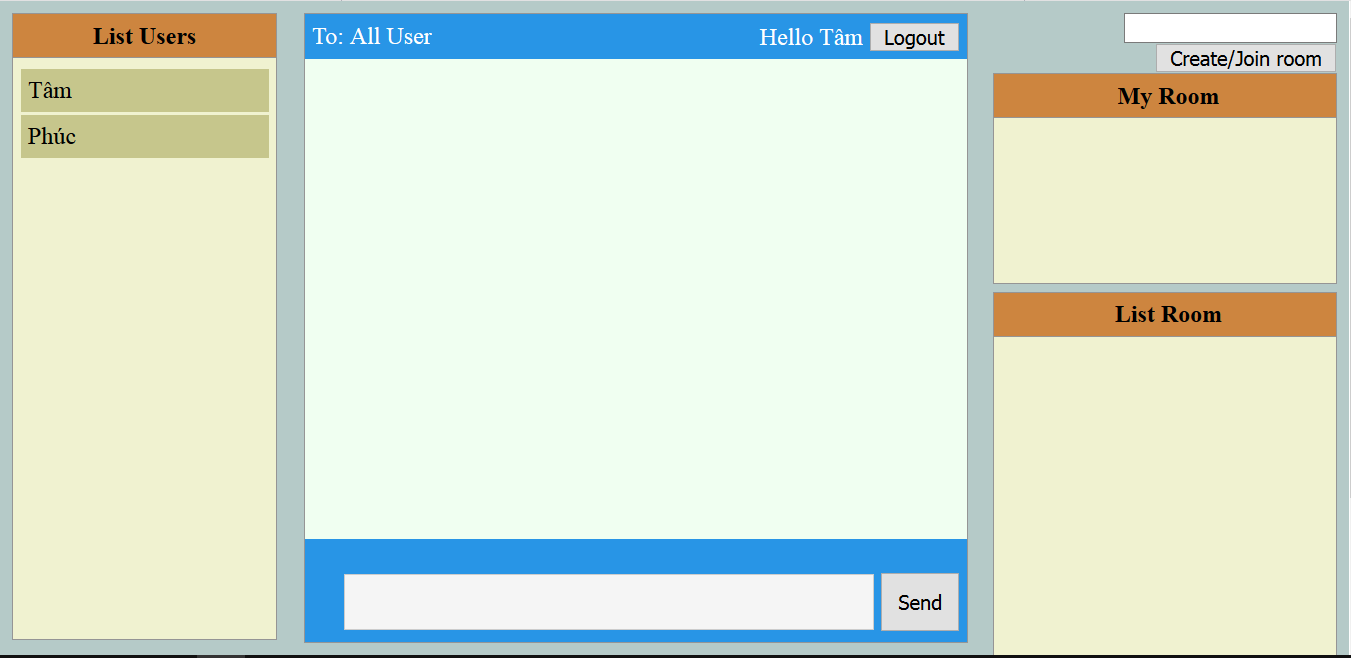
Nếu duyệt mảng, phát hiện tài khoản tồn tại nhưng đã offline, hoặc chưa tồn tại trong mangUsers, server sẽ tạo tài khoản, hoặc cập nhật socket.id của tài khoản.



*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 mangUser 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.

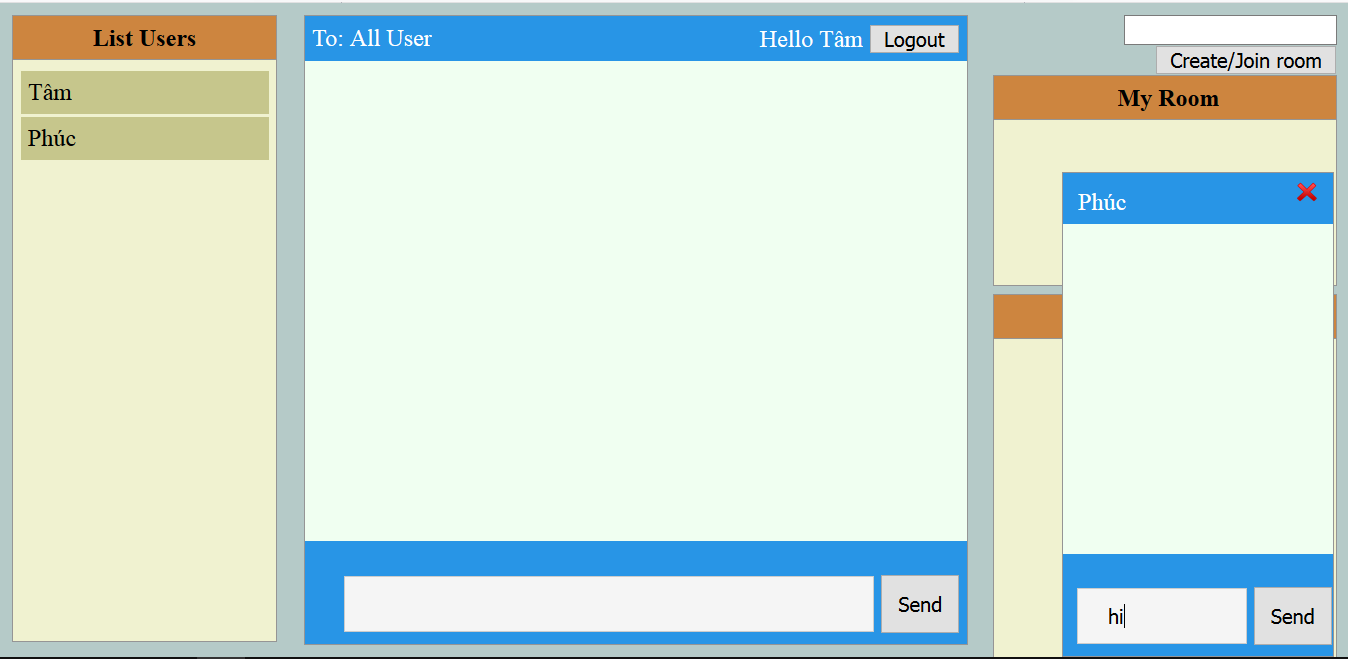
Các sự kiện này sẽ được client lắng nghe, và tạo giao diện cho người dùng.



*Hình ảnh minh họa: giao diện người dùng*

**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.



*Hình ảnh minh họa giao diện người dùng*

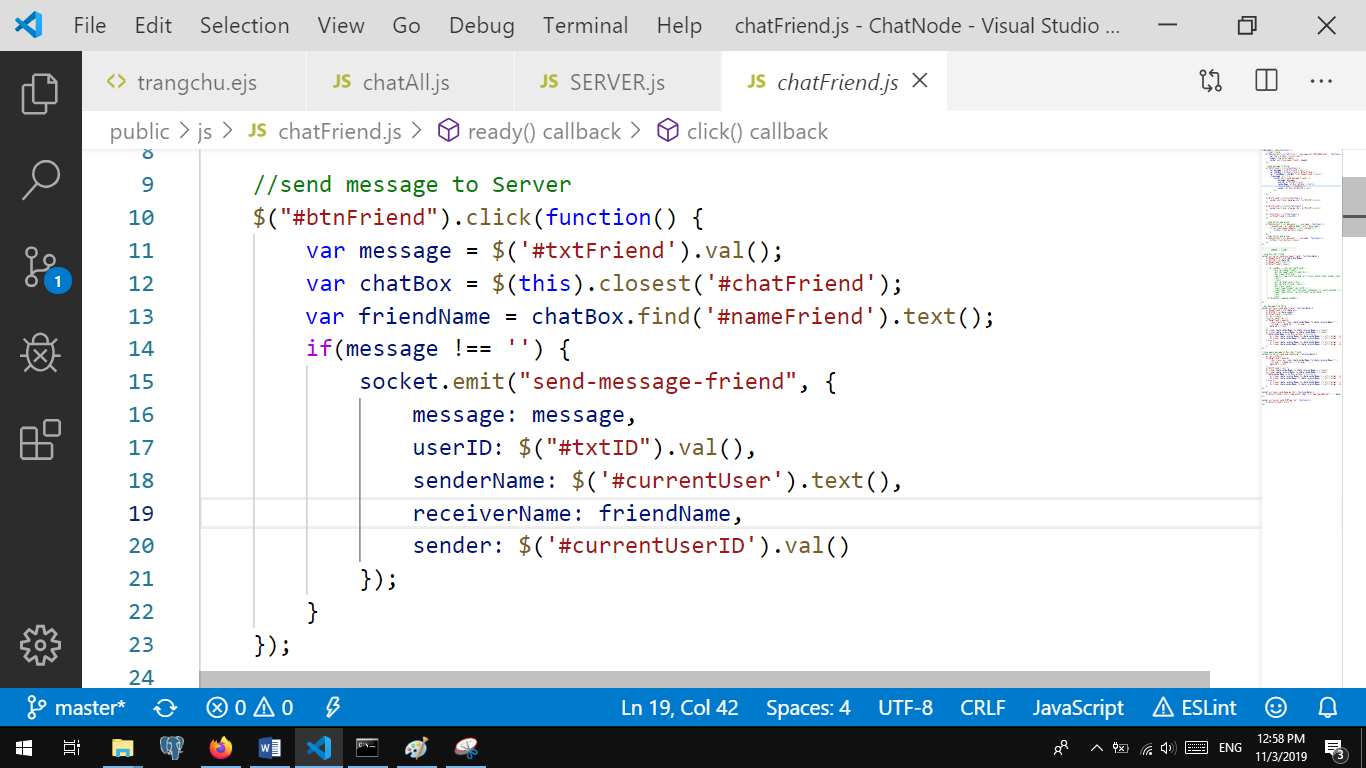
Khi nút Send được nhấn, hoặc nhấn nút enter, client sẽ tạo một sự kiện gửi tin nhắn, và phát đi đến server để giải quyết, server nhận được và giải quyết lập tức gửi lại cho người nhận tin nhắn.

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”.

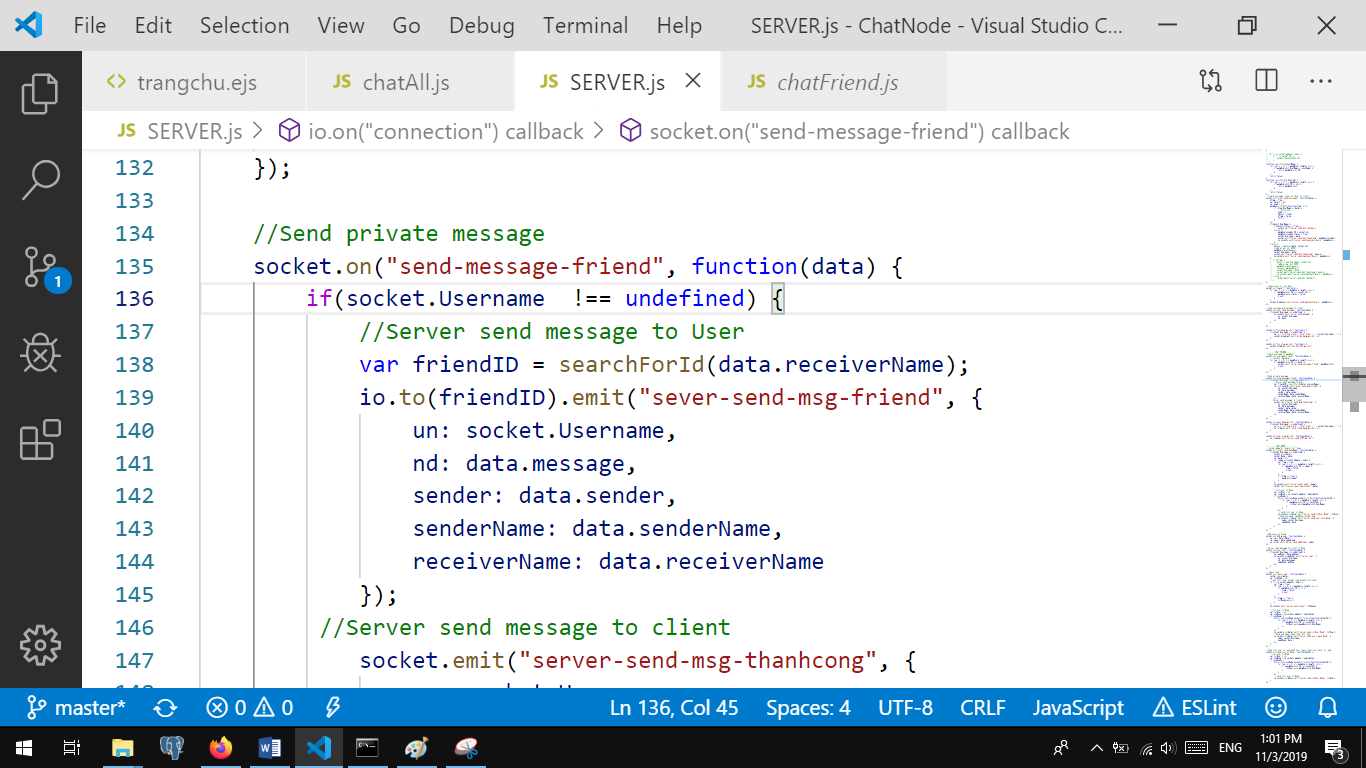
Với hàm emit(), client sẽ gửi đi sự kiện có tên là “send-message-friend”, và dữ liệu là một object bao gồm

* 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.

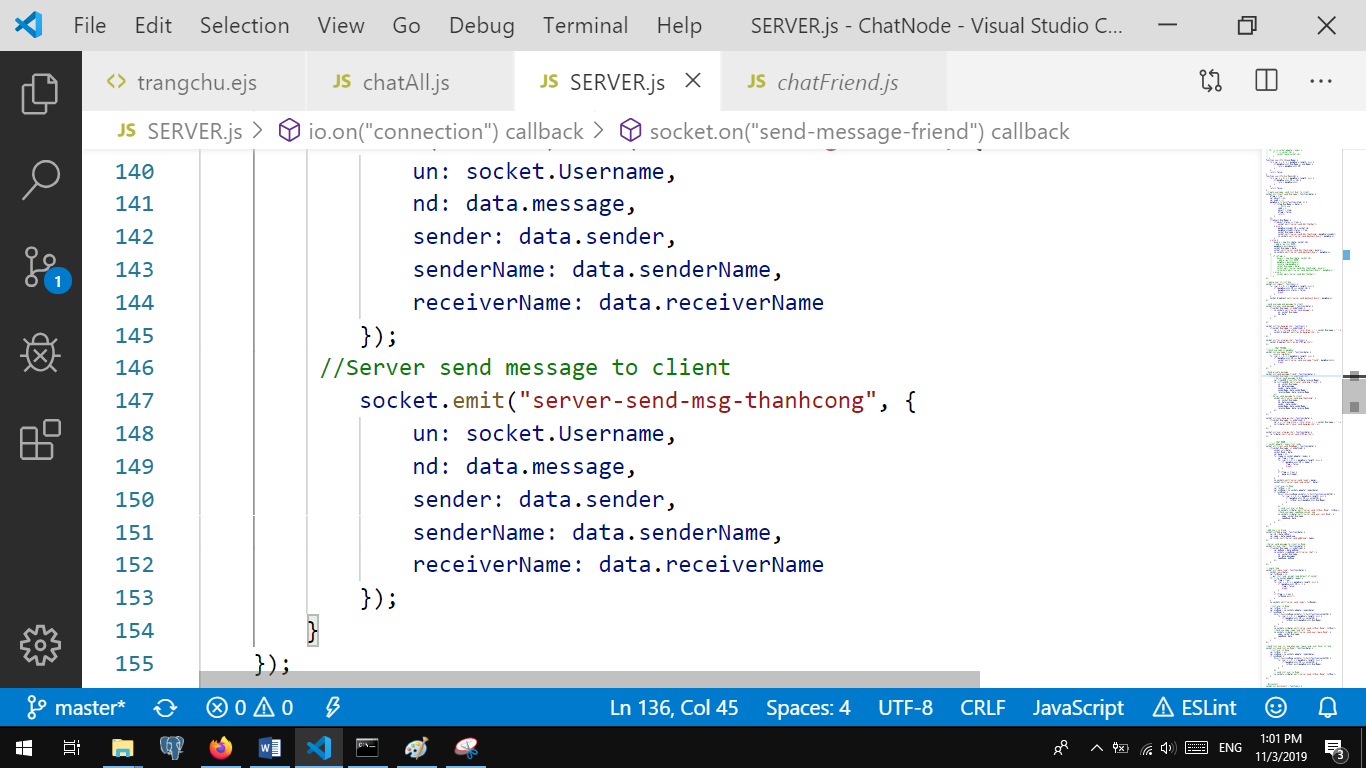
Khi đó ở phía server sẽ config một sự kiện tên là “send-message-friend” để hứng sự kiện được gửi tới.



*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.*



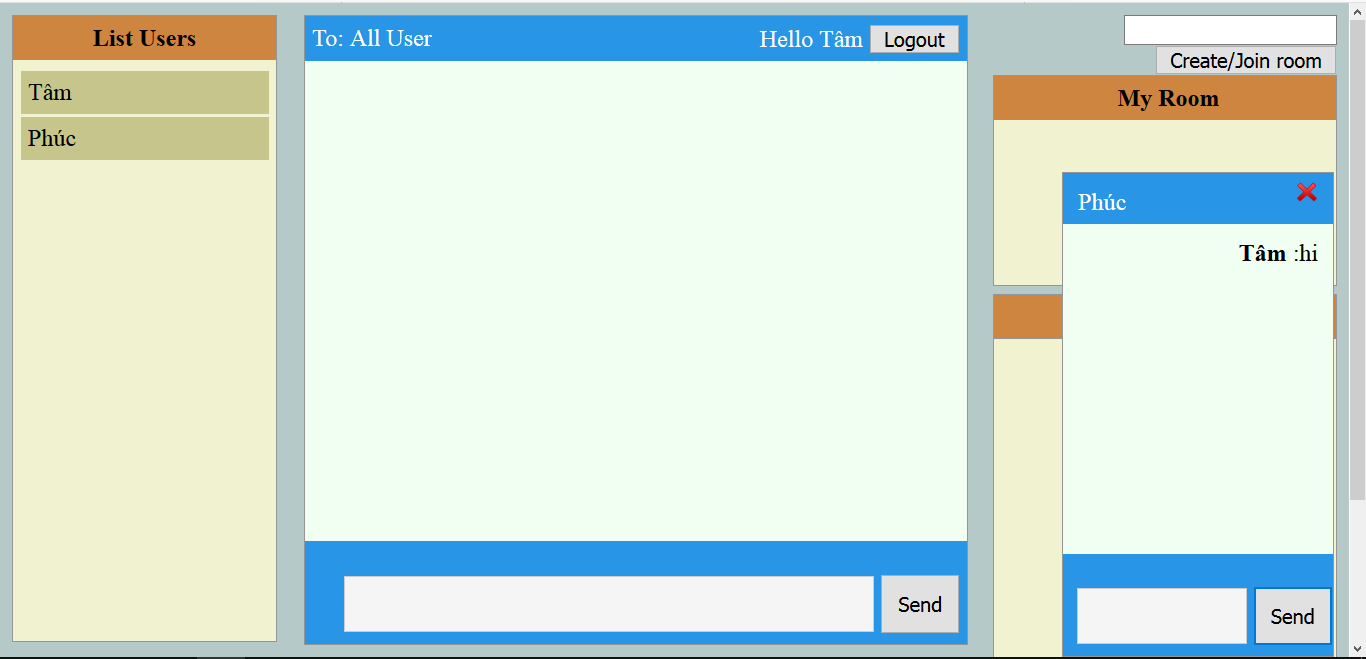
*Hình ảnh minh họa: sự kiện “server-send-msg-thanhcong” được gửi đi ở phía server.*

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.

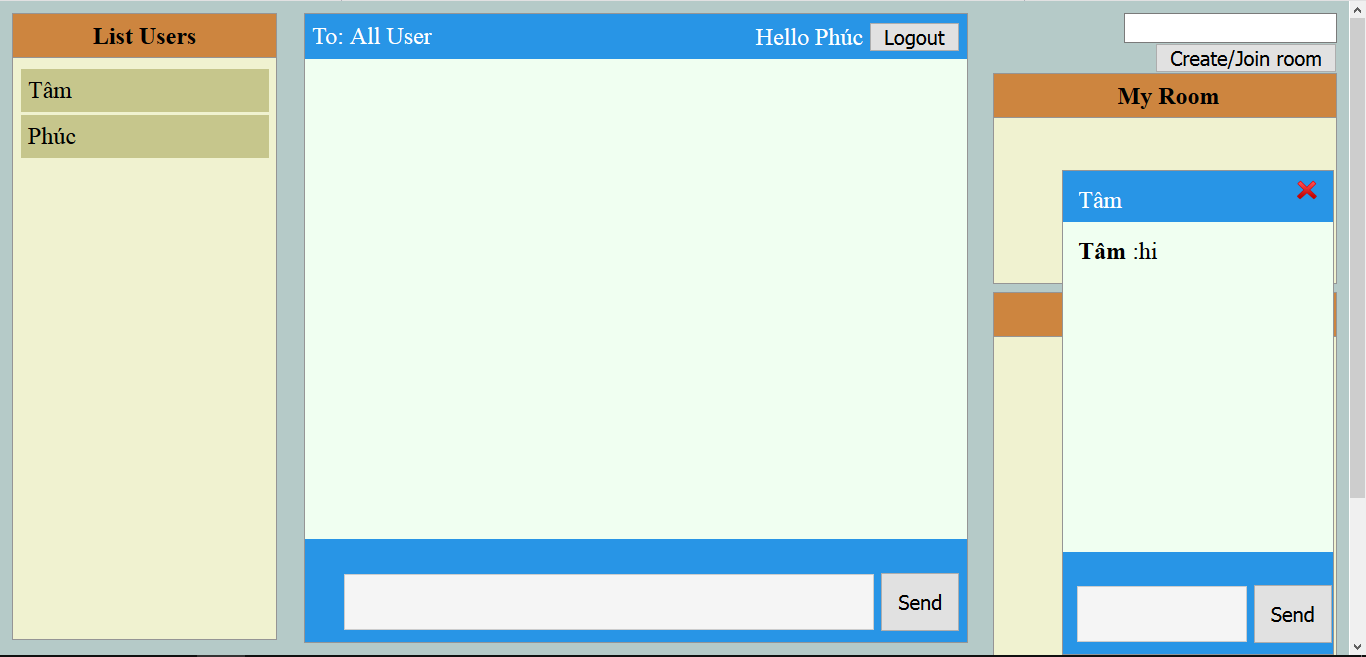
Ở đây, để gửi từ client này để tới client khác dùng hàm to(), dùng để gửi thông tin một cách private, khác với io.socket.emit() dùng để gửi tới tất cả client, hay client.emit() dùng để liên lạc tới server, to() dùng để liên lạc tới chỉ một client khác, và không được public đến mọi người.

Hàm to() gồm 1 tham số duy nhất, đó là địa chỉ socket.id của người cần nhận

**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*

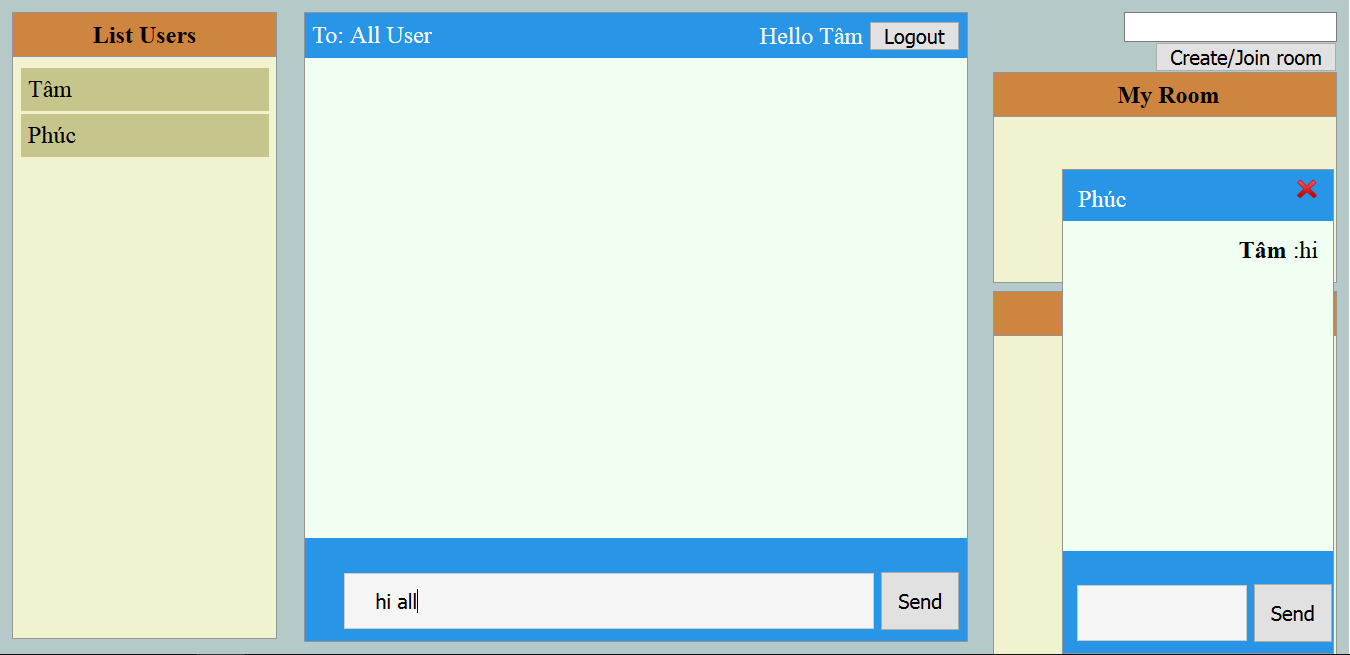
**6.3 Chat All(Broadcasting)**

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.

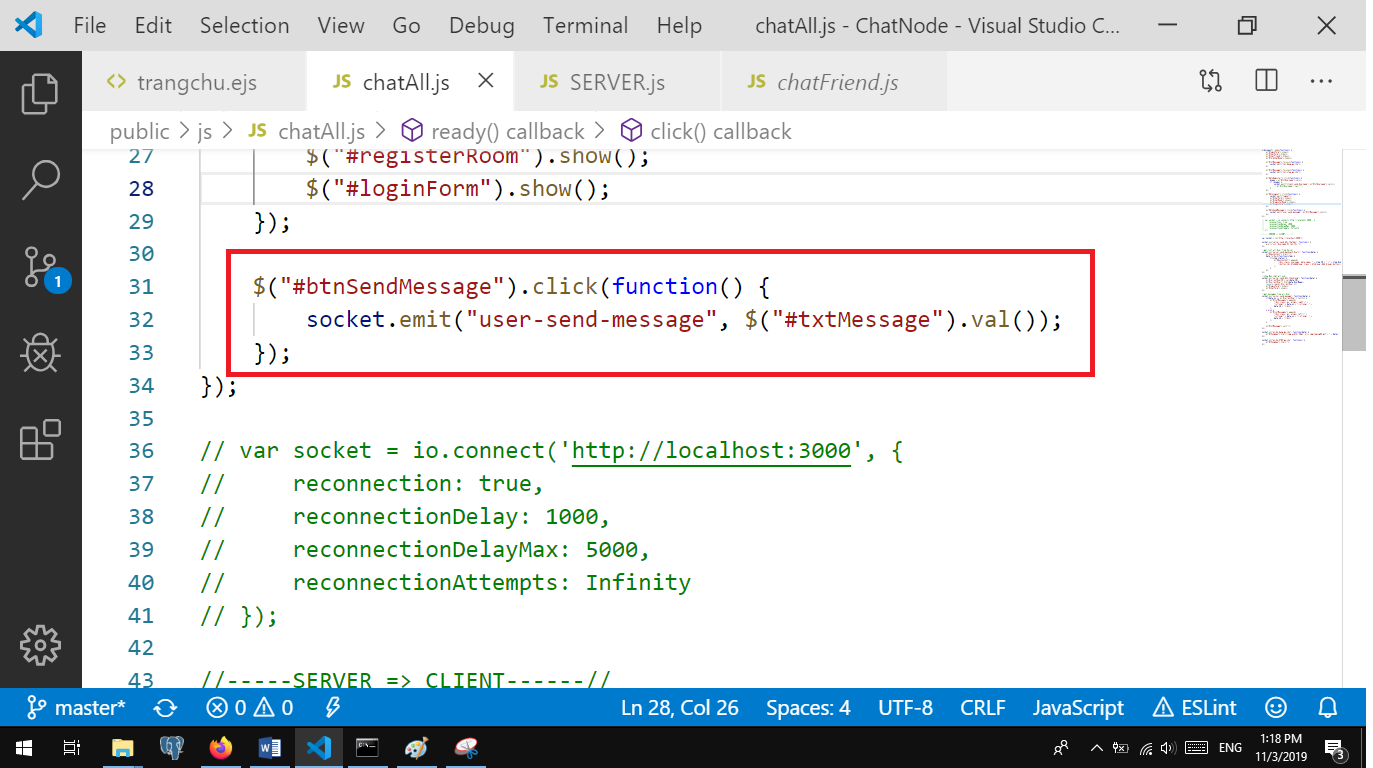
Khi gửi tin nhắn chat all đi, client chỉ làm một việc đó chính là gửi tin nhắn đến server, không cần thiết phải lọc tin nhắn xem gửi đến ai, hay gửi đến room cụ thể nào đó.

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.



*Hình ảnh minh họa: giao diện người dùng*

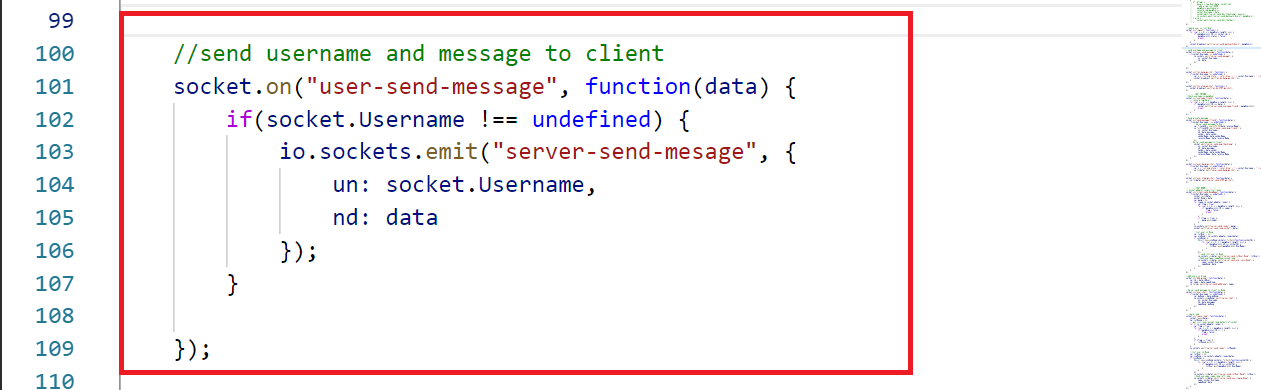


*Hình ảnh giao diện: sự kiện “server-send-message” được gửi đi ở phía client*

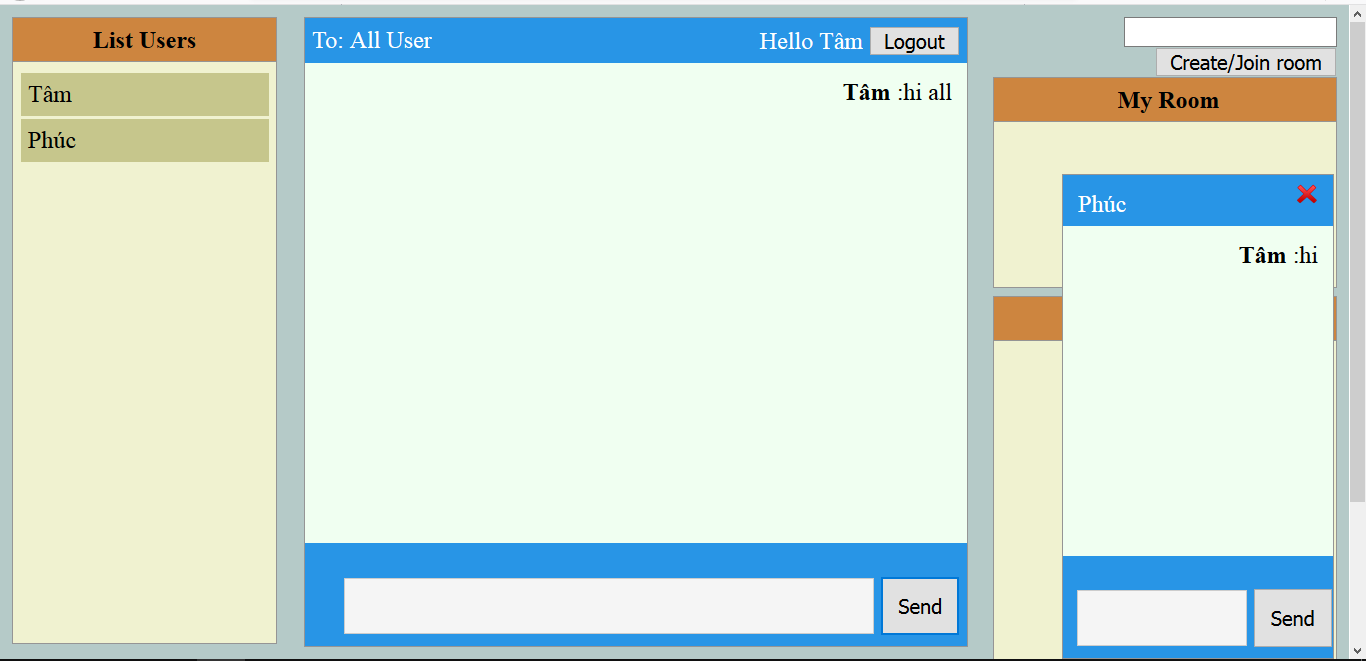
Khi server nhận được một sự kiện “user-send-message”, server ngay tức khắc gửi đi một sự kiện gọi là “server-send-message” để gửi tin nhắn tới mọi client đang kết nối.

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

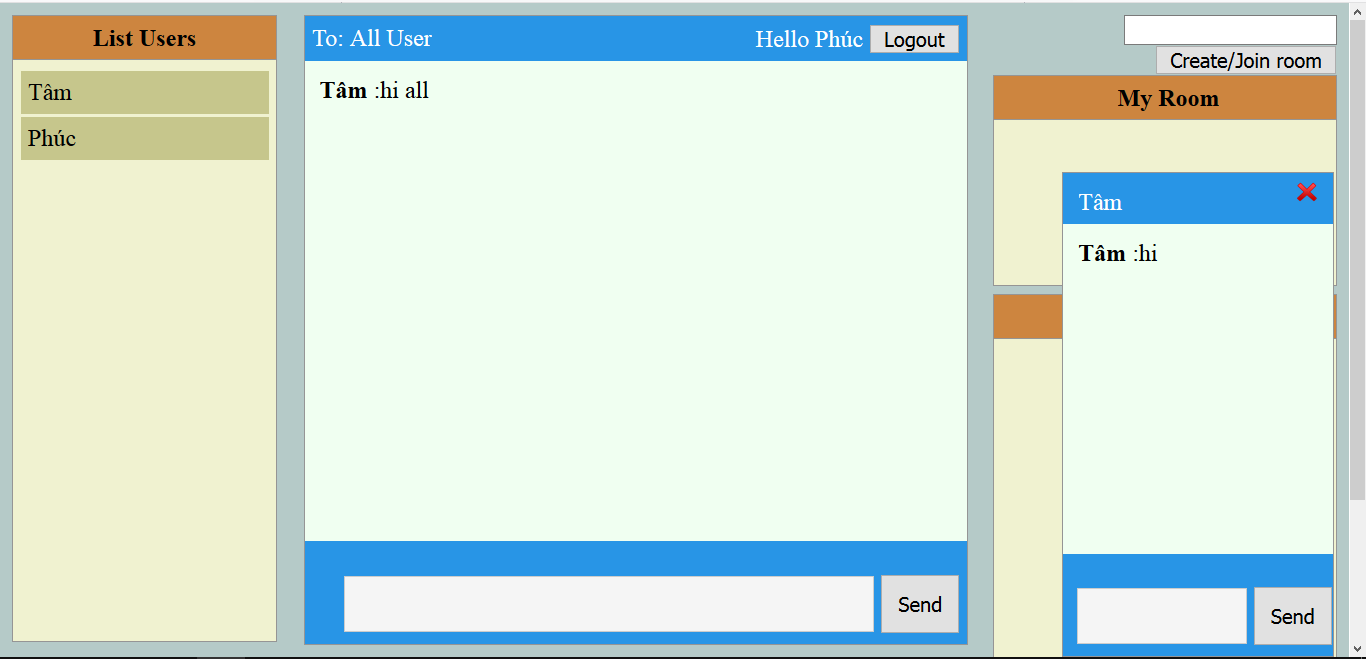
**Lưu ý**: việc gửi tin nhắn all này không cần quan tâm đến socket#id của tất cả người dùng, chỉ cần thông tin socket#id của người gửi để config giao diện cho tất cả người dùng.



*Code xử lý bên server*



*Hình ảnh minh họa: giao diện người gửi*

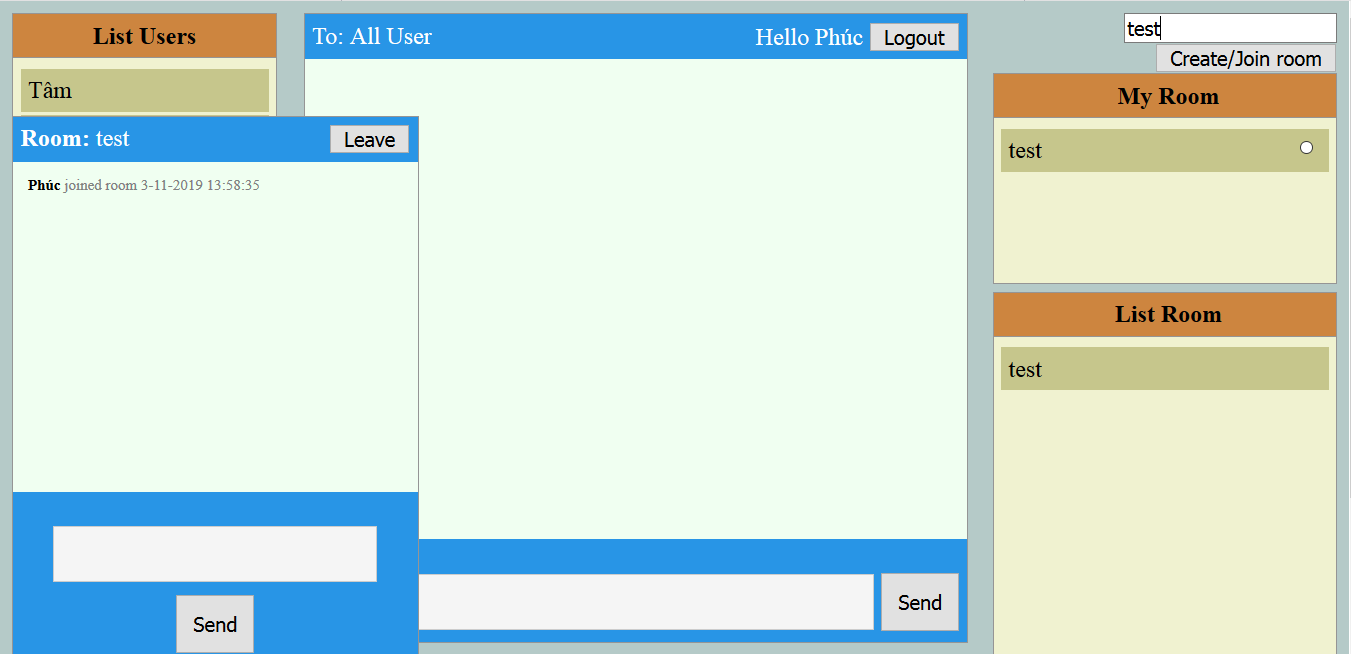


*Hình ảnh minh họa: giao diện người nhận*

**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.

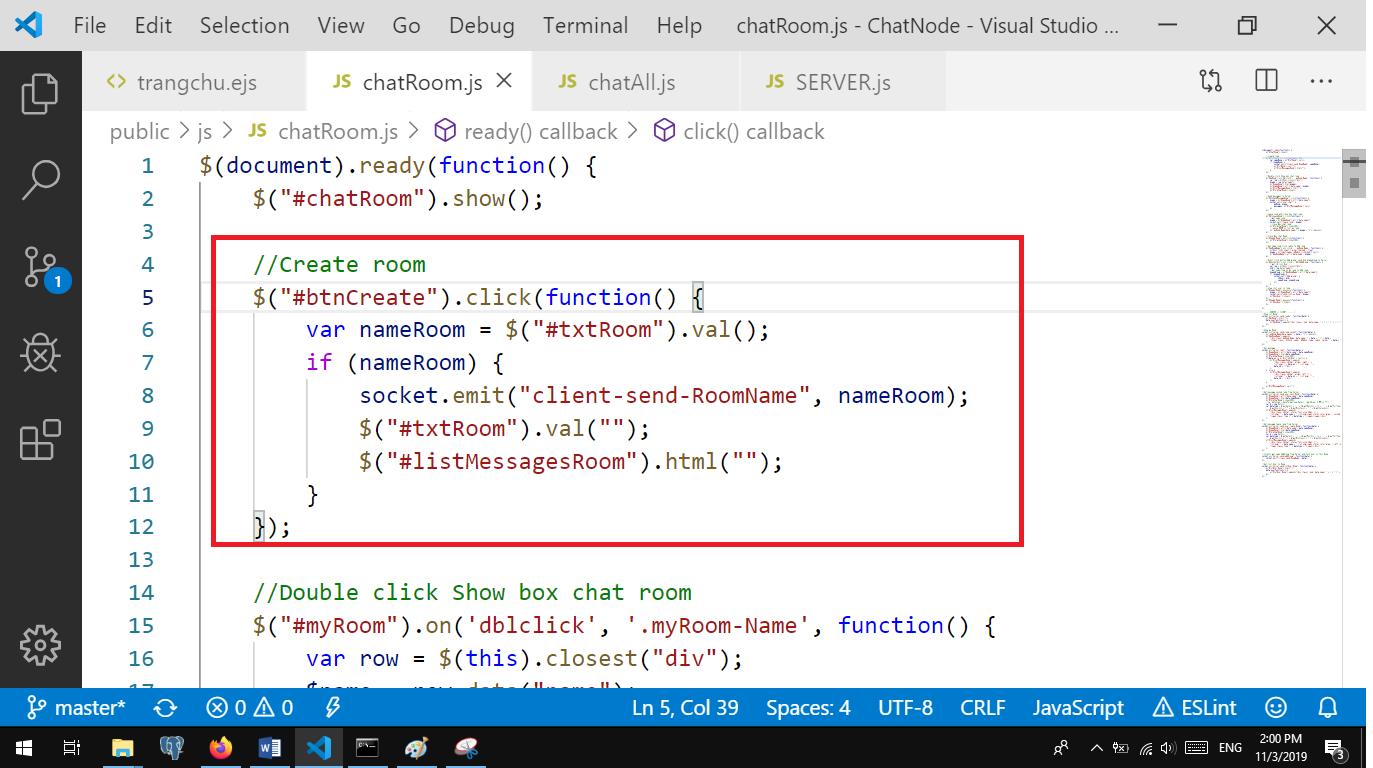
Ở ứng dụng này, config cho socket#id của room cần tạo chính là thông tin được nhập vào ô textbox kế bên nút Create/Join room, nên nhớ mọi chức năng gửi tin nhắn đều thông qua socket#id



*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.

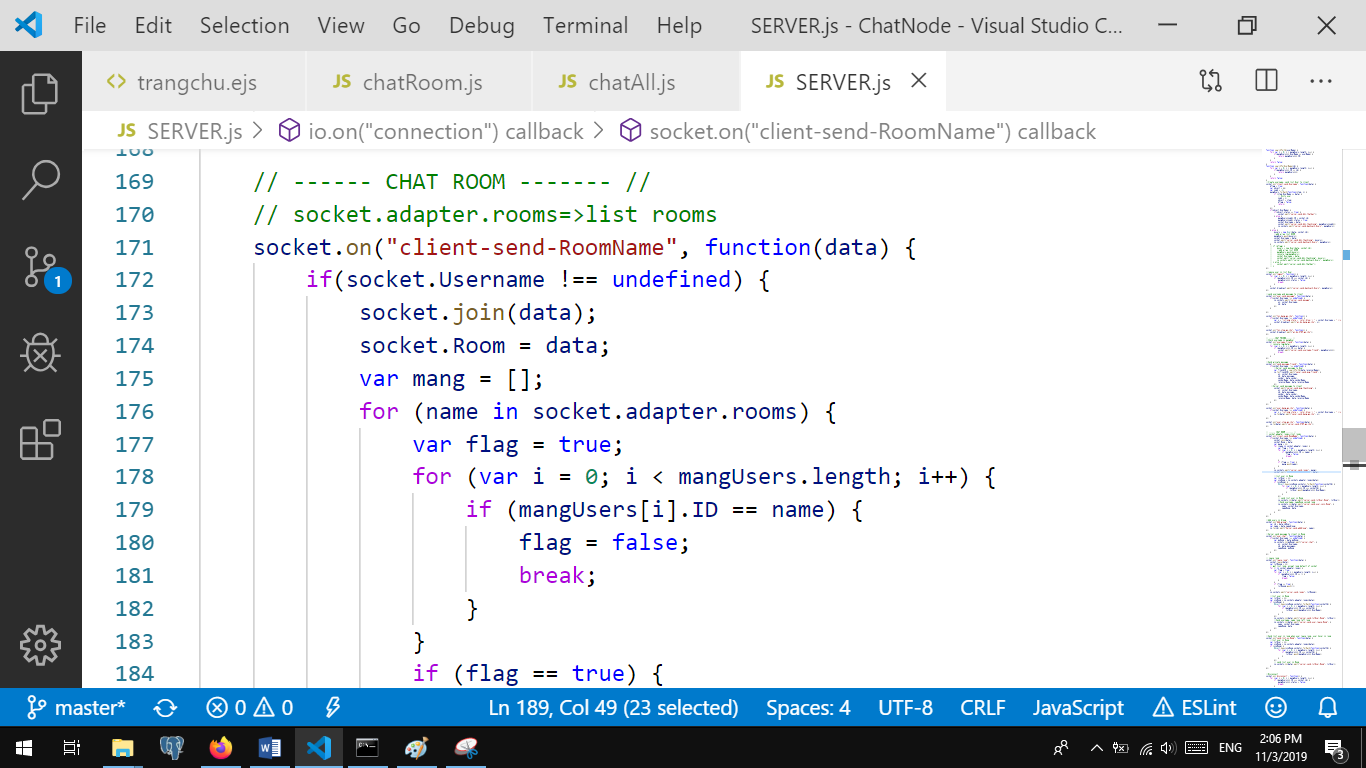


*Hình ảnh minh họa: sự kiện “client-send-RoomName” được gửi đi khi nhấn button create room*

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

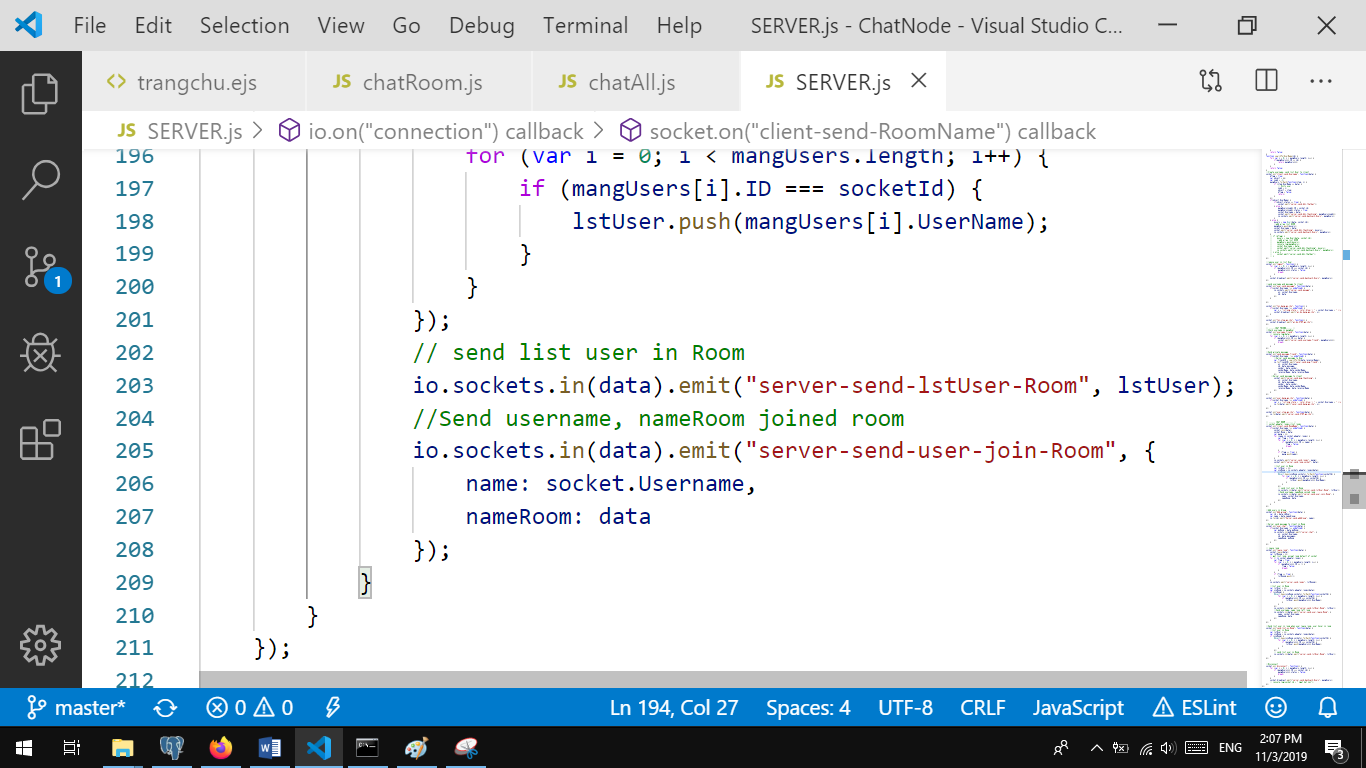


*Hình ảnh minh họa: xử lý sự kiện “client-send-RoomName” ở client*



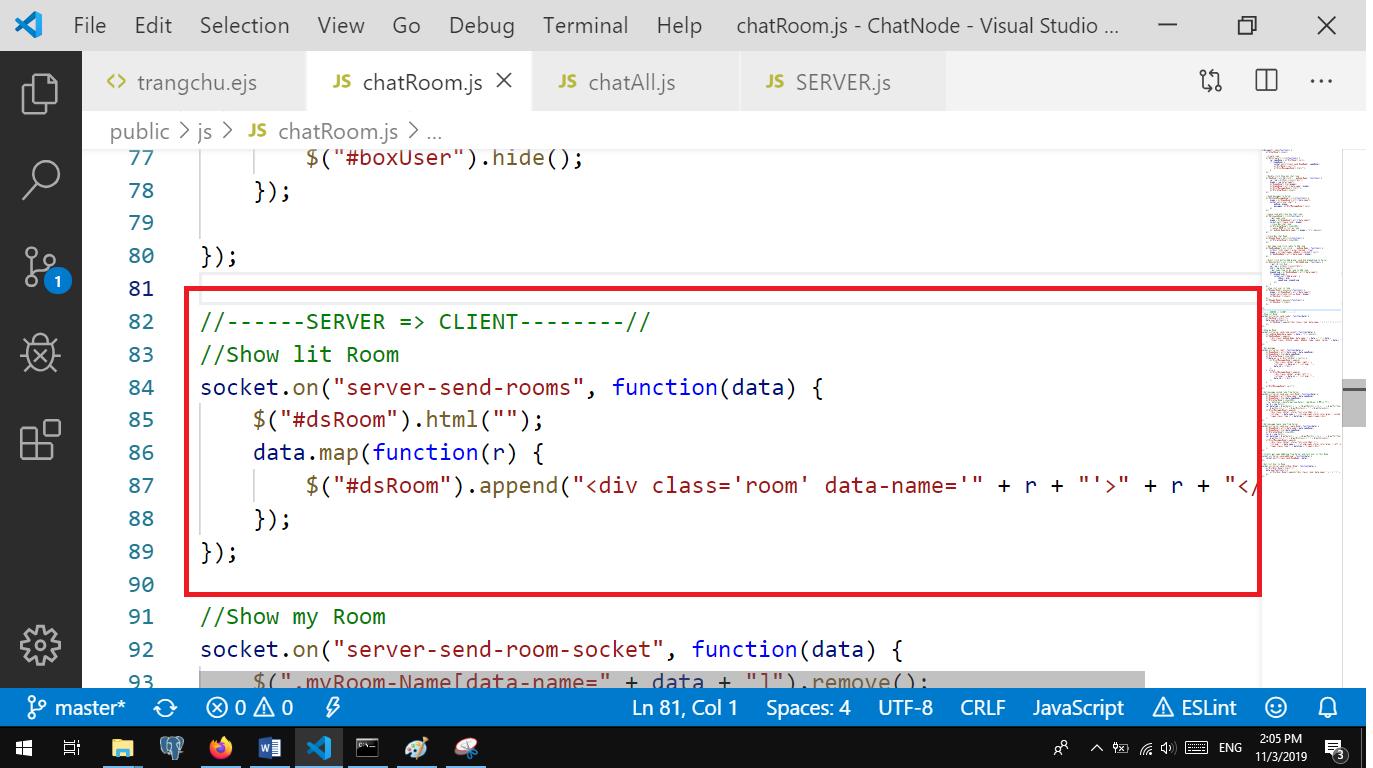
*Hình ảnh minh họa: sự kiện server-send-room-socket*

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.



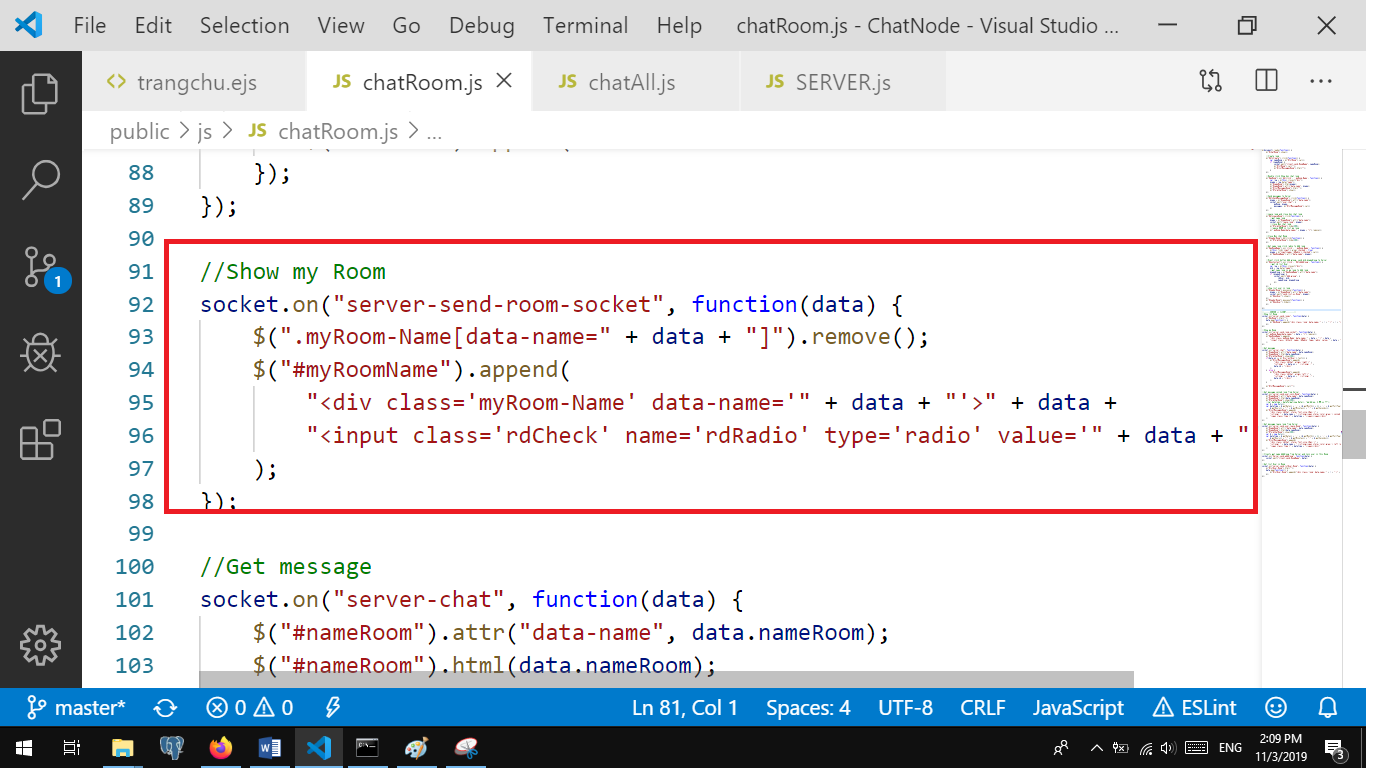
*Hình ảnh minh họa: Hàm in() ở phía server.*

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.

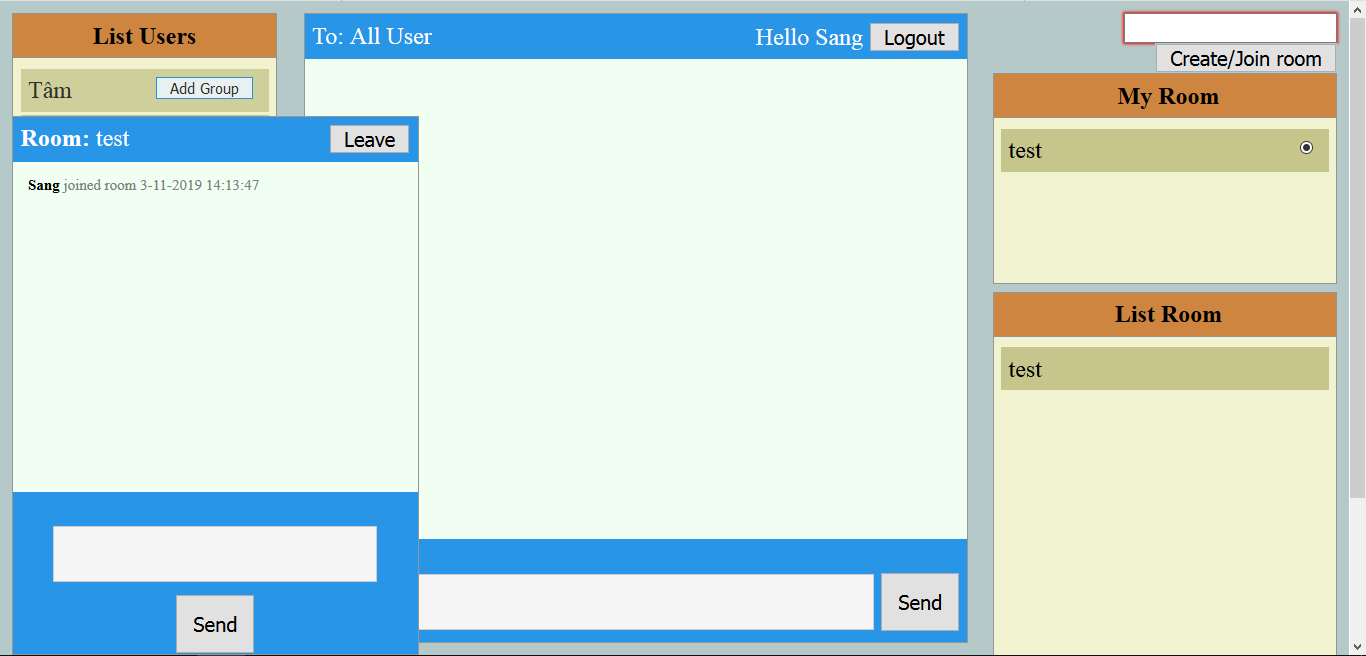


*Hình ảnh minh họa: sự kiện “server-send-rooms” ở client*

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.

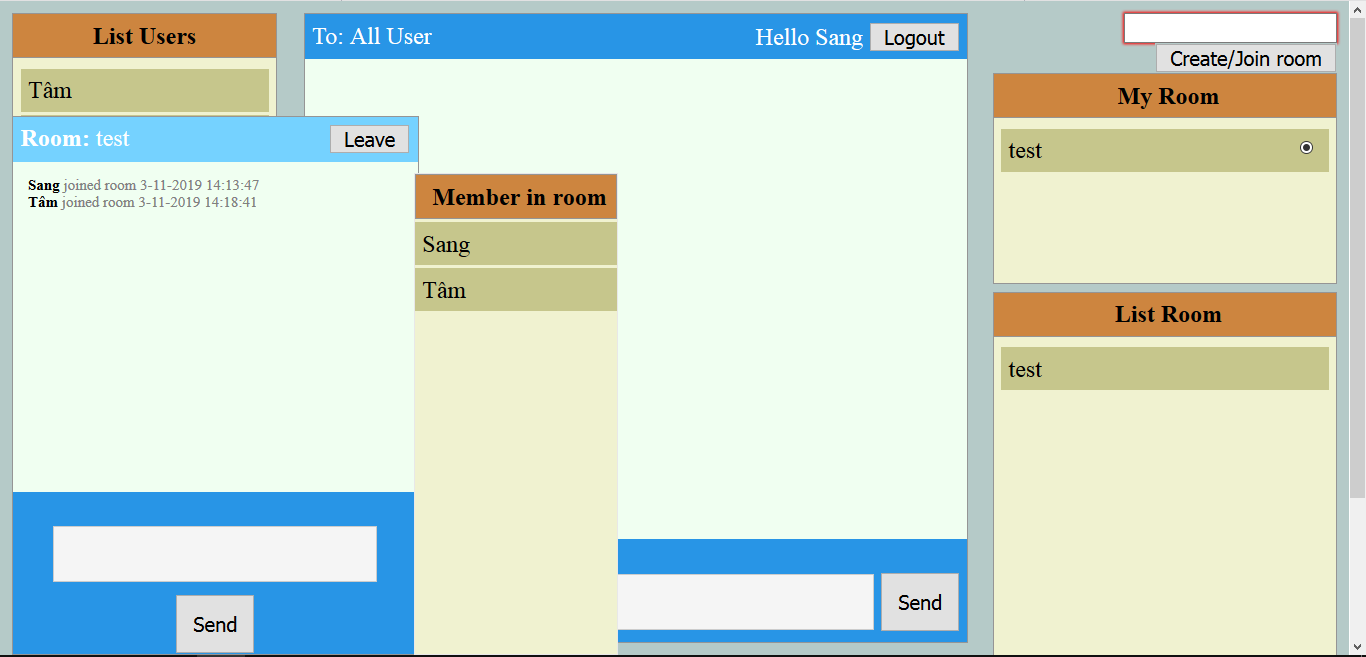


Để mời người khác tham gia room, chỉ cần rê chuột đến tên room mình tham gia trong my room, chọn vào radio button, sau đó rê chuột đến List User chọn user mà mình muốn người.



*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.

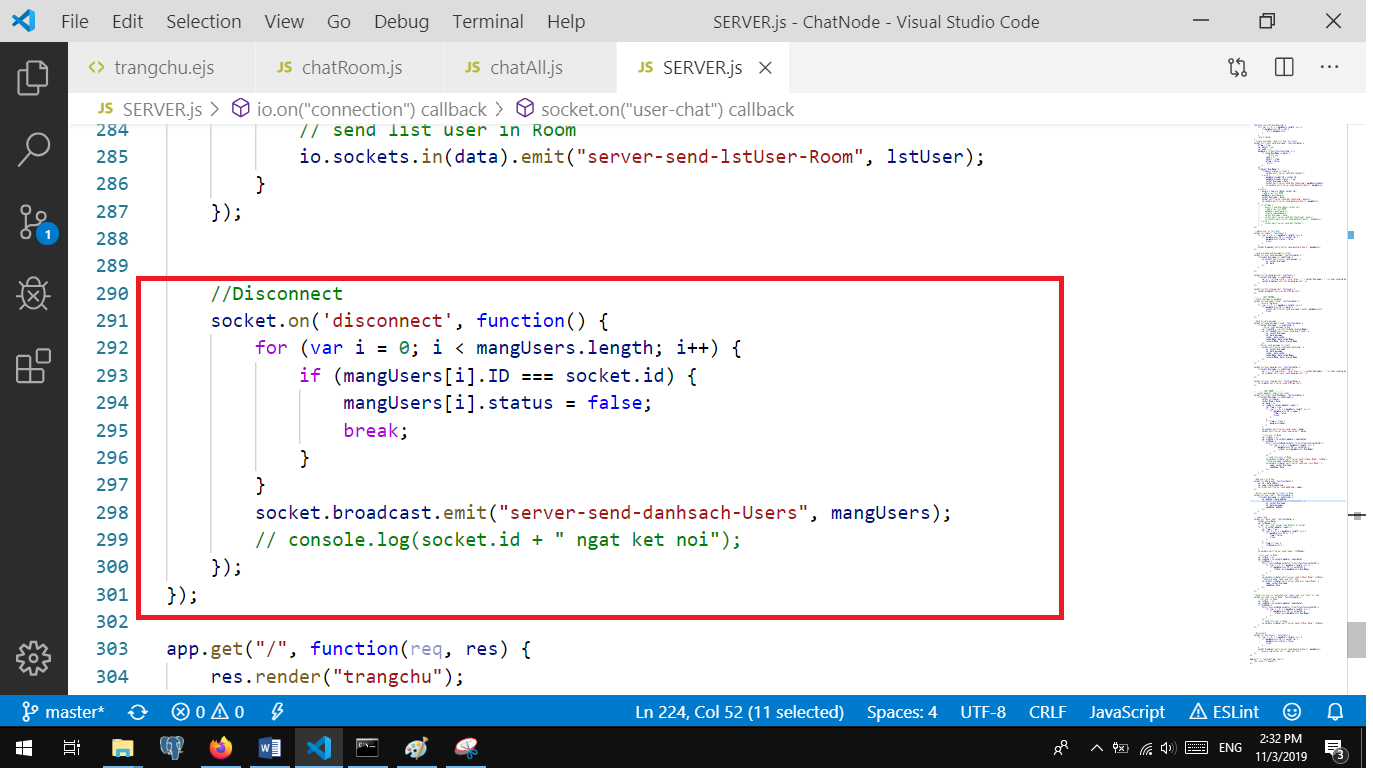
Server sẽ nhận được tin nhắn khi nút send trong room được nhấn, khi đó server sẽ kiểm tra xem người gửi có tồn tại không, nếu không tồn tại thì việc gửi tin nhắn sẽ không thành công, ngược lại sẽ gửi tin nhắn đến room cho các thành viên qua sự kiện “server-chat” với các tham số là tên người gửi, nội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

**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.

*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.

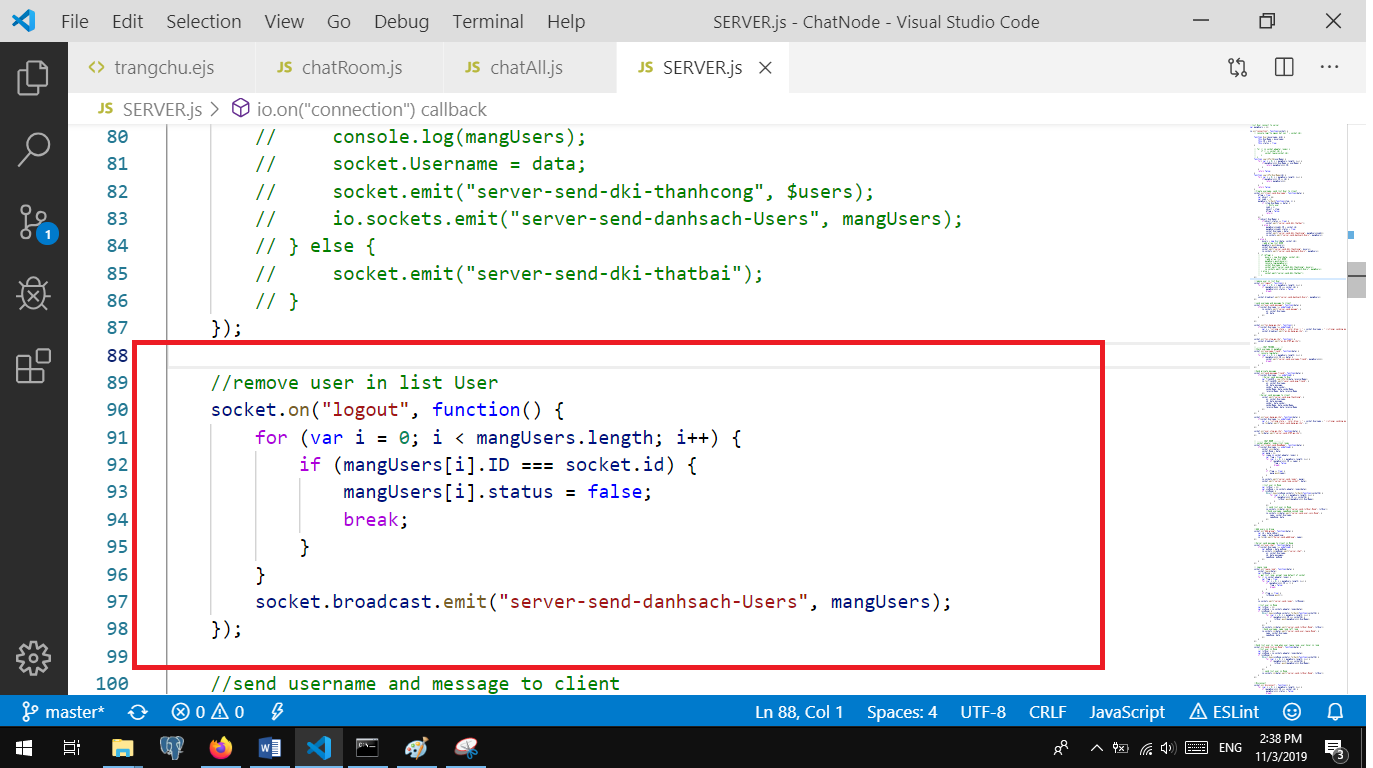
Hàm socket.broadcast.emit() có hai tham số bao gồm tên sự kiện và dữ liệu gửi đi tương tự như io.socket.emits() dùng để gửi sự kiện đến mọi client kết nối, ngoài ra socket.broadcast.emit() còn gửi tới những người dùng vừa mới tạo, hoặc chính client, kẻ tạo nên sự kiện mất kết nối.

**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.



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

**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 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>