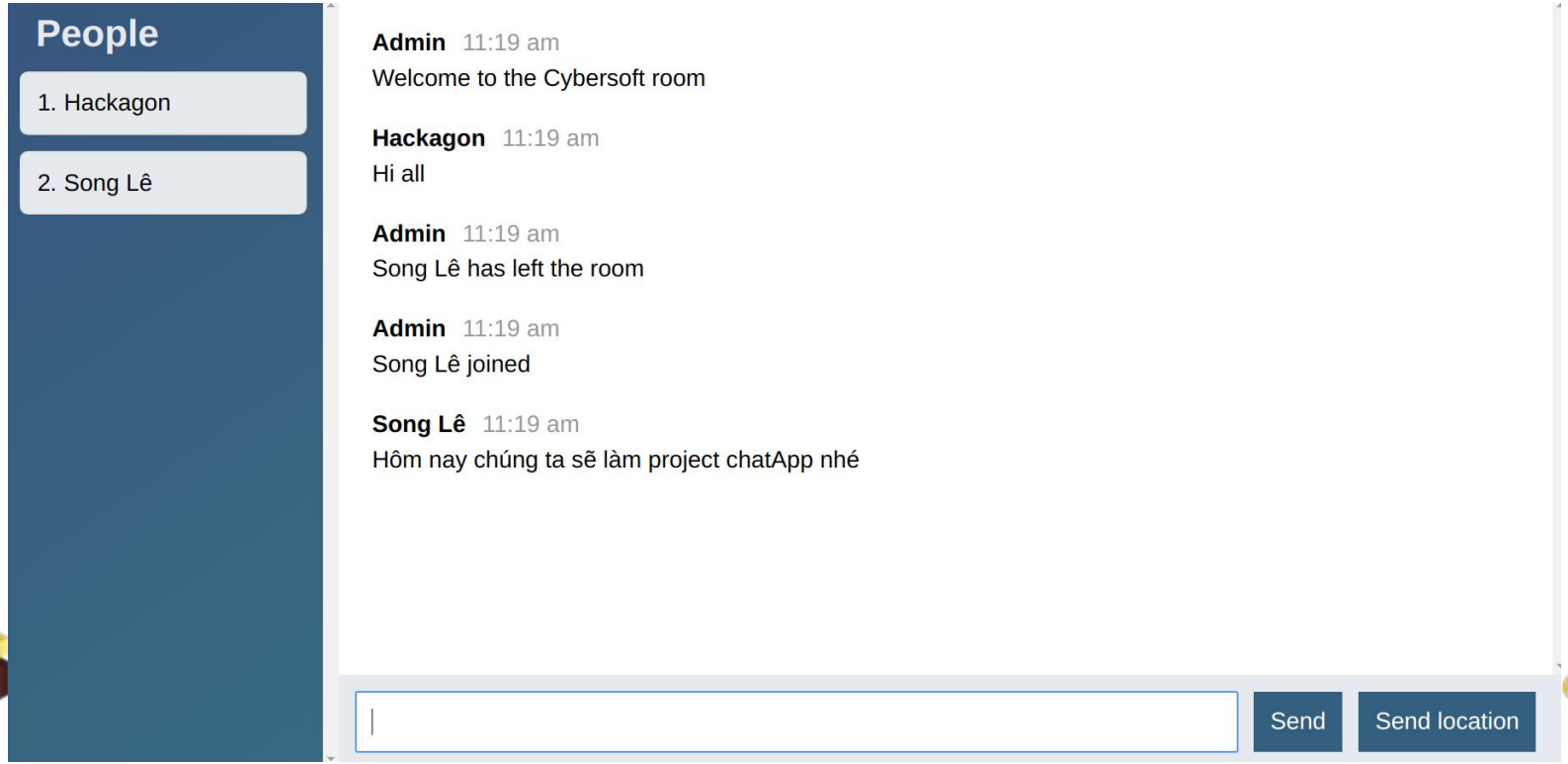
SOCKET IO - CHAT APP

PHÓ NGHĨA VĂN

PROJECT: CHAT APP



PROJECT: CHAT APP

- Server side: Terminal, folder server
- Client side: Browser và Console của browser, folder public

KIẾN THỨC

- Websocket và socket.io
- Sự kiện **connect**, **disconnect** ở phía server và client
- **Custom một sự kiện**
- Emiting và Listening một sự kiện
- Emiting một sự kiện đến một room riêng biệt
- Sử dụng **geolocation** để định vị
- Áp dụng Jquey để DOM đến các thành phần html
- Template engine **MUSTACHE**

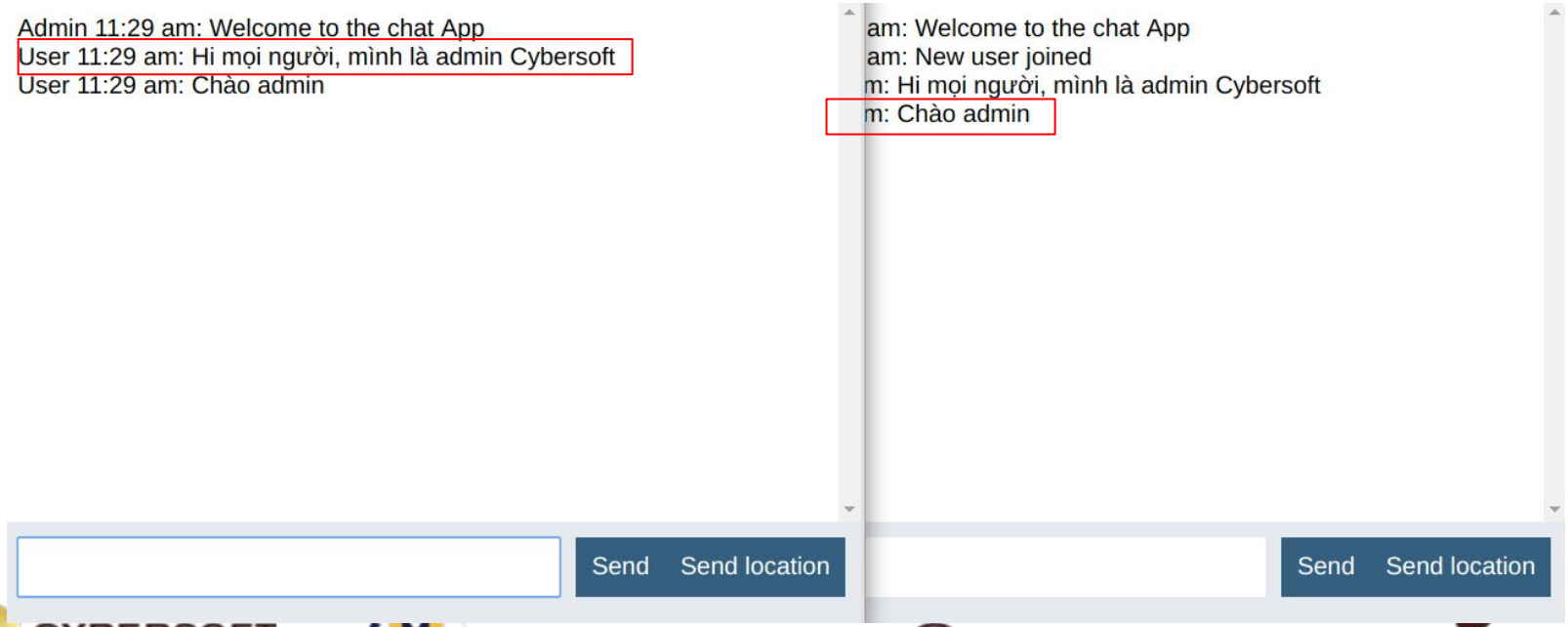
CÔNG CỤ

- Visual studio code
- Terminal
- NodeJS và các extension:
  - Bootstrap 4, Font awesome 4, Font awesome 5 Free & Pro Snippet
  - Auto close tag
  - Bracket pair colorizer
  - **Node.js module Intellisense**

WEBSOCKET LÀ GÌ?

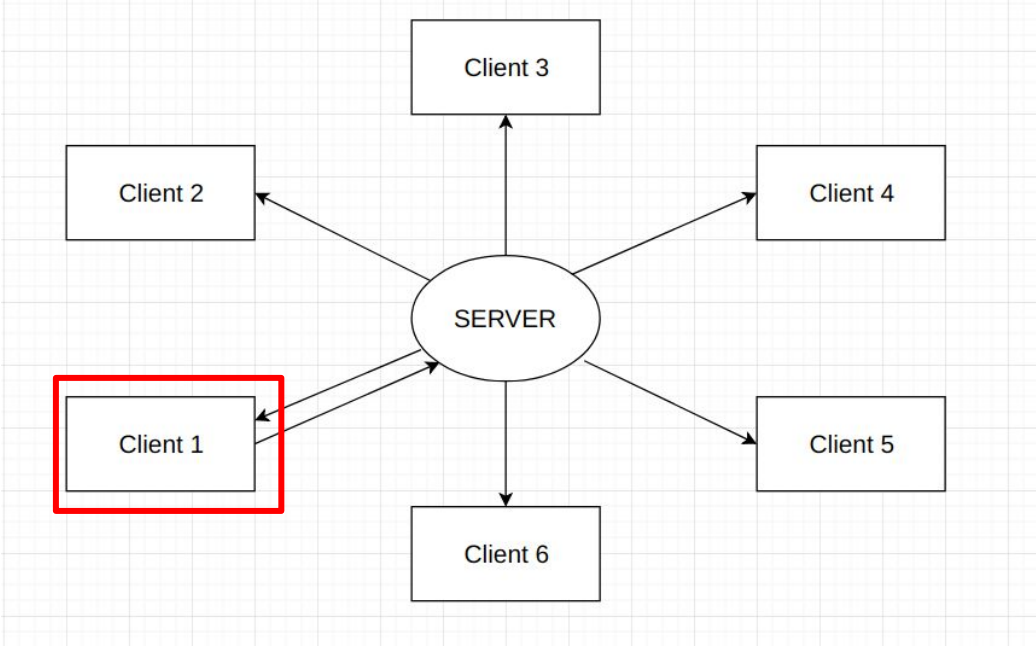
- Websocket là giao thức chuẩn cho trao đổi dữ liệu hai chiều giữa client và server. Giao thức WebSocket không chạy trên HTTP, thay vào đó nó thực hiện trên giao thức TCP.
- Người ta thường dùng Websocket thay vì HTTP cho những trường hợp yêu cầu real time (thời gian thực). Ví dụ bạn muốn hiển thị biểu đồ, chỉ số chứng khoán, web chat, game cờ caro,...
- Gói tin của WebSockets nhẹ hơn HTTP rất nhiều, giảm độ trễ của network lên đến 3 lần, không cần phải gửi request liên tiếp như HTTP.

CHAT-APP



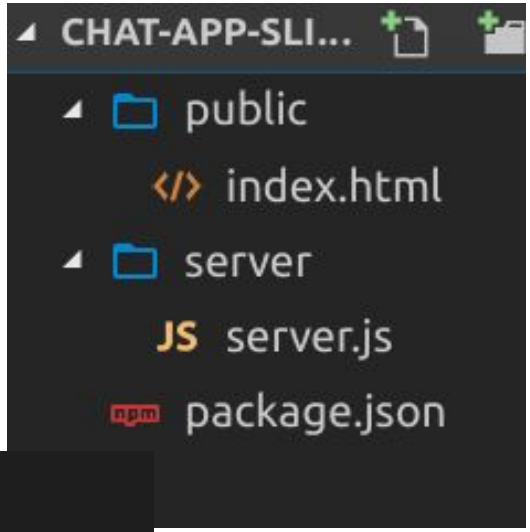
CHAT-APP

Khi Client 1, muốn gửi message đến cả nhóm, Client 1 sẽ **emit** (phát) message đến server, server **listen** (nhận) message này và sẽ **emit** đến các client còn lại
(bao gồm hoặc không Client 1
- client đang tương tác với server)



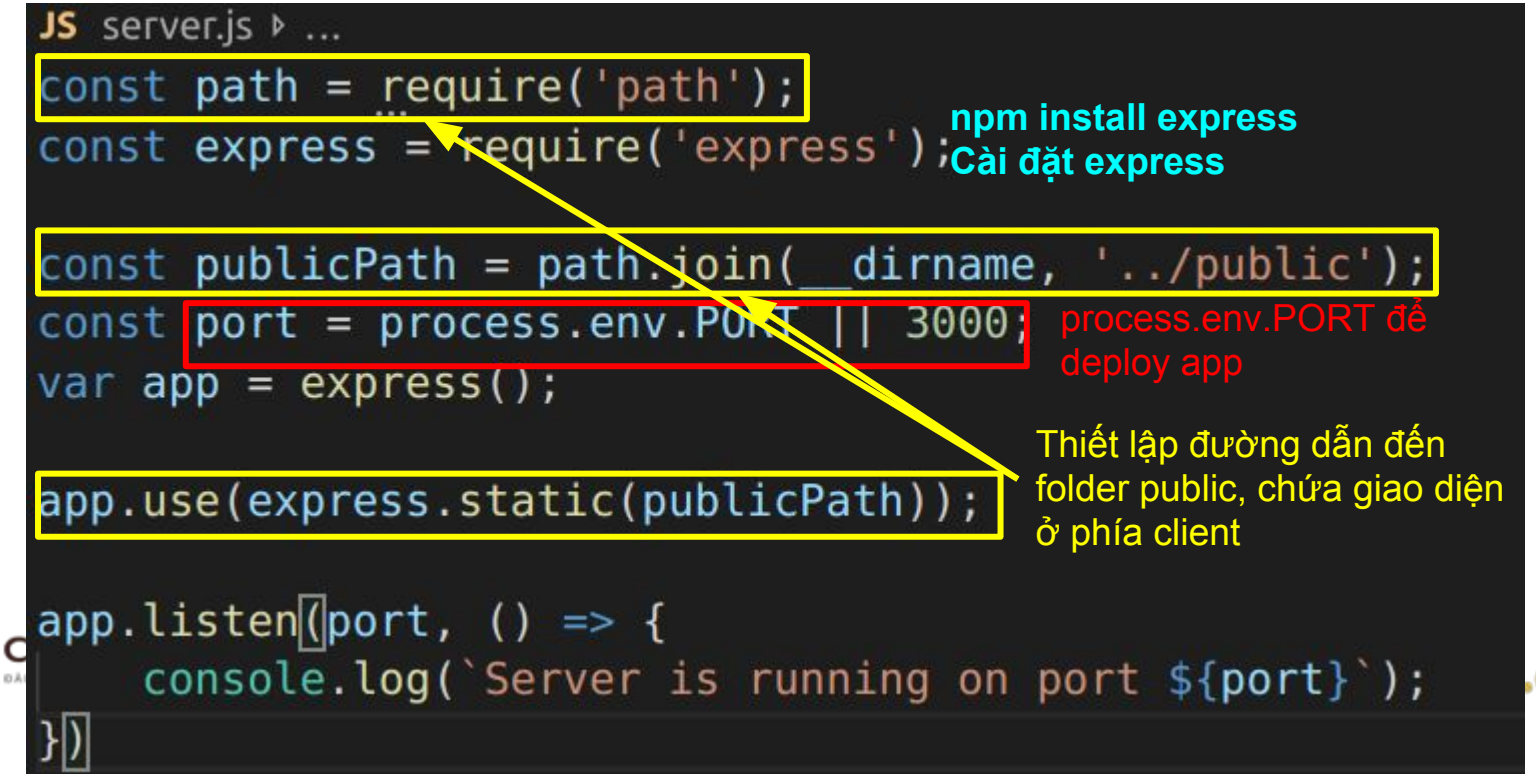
KHỞI TẠO PROJECT

- Vào terminal **npm init**
- Tạo folder server và file server.js
- Tạo folder public và file index.html
- Trong **index.html**, khởi tạo cấu trúc như sau




```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h1>WELCOME TO THE CHAT APP</h1>
</body>
</html>
```

CẤU HÌNH SERVER.JS



NODEMON

- **npm install nodemon**
- Trong package.json, thiết lập **start** để trỏ đến trỏ đến file **server.js**

```
{
  "name": "chat-app-slide",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon server/server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.4",
    "nodemon": "^1.18.4"
  }
}
```

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

LASS

KẾT QUẢ

- npm start
- Thu được kết quả như sau:

CLIENT SIDE

```
hackagon@meta:/media/hackagon/MetaData/CybersoftAcademy/NodeJS_Lecture/Project-chatApp/chat-app-slide$ npm start

> chat-app-slide@1.0.0 start /media/hackagon/MetaData/CybersoftAcademy/NodeJS_Lecture/Project-chatApp/chat-app-slide
> nodemon server/server.js

[nodemon] 1.18.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server/server.js`
Server is running on port 3000
```

SERVER SIDE

Socket.IO

Giới thiệu:

- Socket.IO là một thư viện javascript có mục đích tạo ra các ứng dụng realtime trên trình duyệt cũng như thiết bị di động
- Thư viện này hỗ trợ cả 2 phía: client và server

Cài đặt:

- **npm install socket.io**

TÍCH HỢP Socket.IO

Gọi thư viện **socket.io** và built-in **http**.

http giúp tạo một server

```
const path = require('path');
const express = require('express');
const http = require('http');
const socketIO = require('socket.io')

const publicPath = path.join(__dirname, '../public');
const port = process.env.PORT || 3000;
var app = express();

var server = http.createServer(app);
var io = socketIO(server);

app.use(express.static(publicPath));

server.listen(port, () => {
    console.log(`Server is running on port ${port}`);
})
```

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

TÍCH HỢP Socket.IO

Phía **server**:

- Sự kiện **connection** xảy ra khi có một truy cập đến localhost:3000
- Sự kiện **disconnect** xảy ra khi hủy truy cập trên

```
io.on('connection', (socket) => {
    console.log('New user connected')

    socket.on("disconnect", () => {
        console.log('User was disconnected');
    })
})
```

TÍCH HỢP Socket.IO

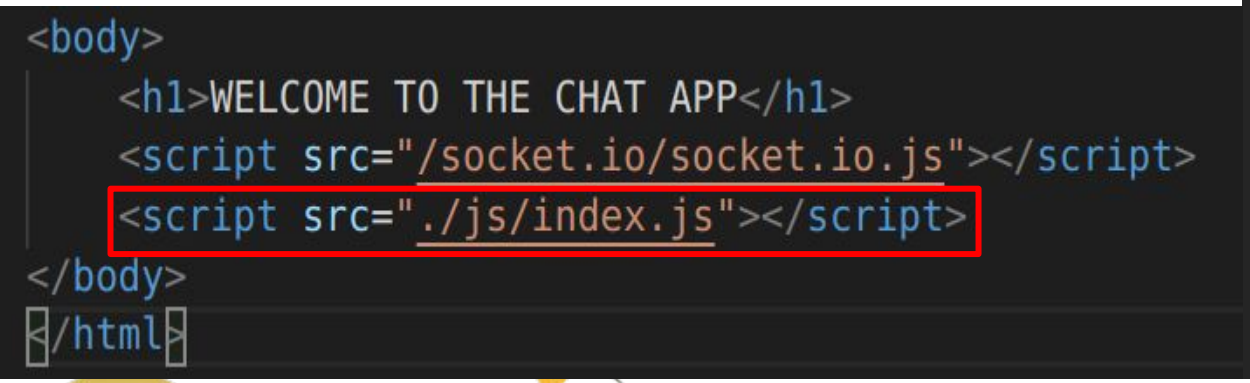
Phía **client**:

- Sự kiện **connect** xảy ra khi client có thể truy cập được đến server thành công
- Sự kiện **disconnect** xảy ra khi server bị stop, do đó client không thể truy cập đến server

```
<script src="/socket.io/socket.io.js"></script>
<script>
    var socket = io();

    socket.on("connect", () => {
        console.log('Connected to the server');
    })

    socket.on("disconnect", () => {
        console.log('Disconnected from the server');
    })
</script>
```

TÍCH HỢP Socket.IO

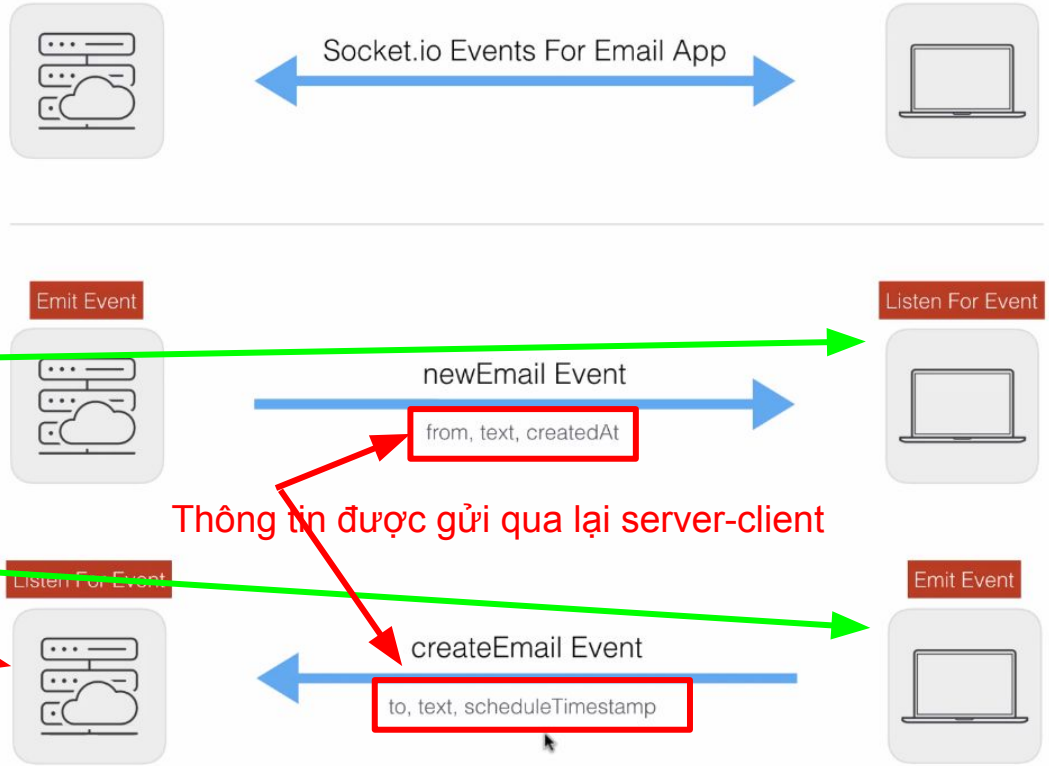
- Tạo file **./js/index.js**
- Đưa toàn bộ code trong <script> tag vào bên trong **index.js**, và gọi **index.js** trong **index.html**



EMITTING & LISTENING TO EVENTS

Mô hình emit-listen thông qua ví dụ về truyền nhận email.

- Khi server nhận được một email từ client A, nó sẽ **emit (phát)** đi đến client B. Client B sẽ **listen (nhận)** email từ server gửi đến.
- Tương tự, client B cũng có thể **emit** email đến server



EMITTING & LISTENING TO EVENTS

Truyền email từ **server** đến **client:**

- Phía server **emit** ra một sự kiện (**custom event**) "newEmail" với nội dung dạng json

```
io.on("connection", (socket) => {
    console.log("New user connected");

    socket.emit('newEmail', {
        from: 'hackagon@gmail.com',
        text: 'Hey, Go fishing today',
        createAt: new Date().getTime()
    })

    socket.on("disconnect", () => {
        console.log("User was disconnected");
    })
})
```

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

LASS

EMITTING & LISTENING TO EVENTS

Truyền email từ **server** đến **client:**

- Phía client **listen** sự kiện "newEmail" và in ra ngoài màn hình console của browser

```
var socket = io();

socket.on('connect', () => {
    console.log("Connected to the server");
})

socket.on('disconnect', () => {
    console.log("Disconnected from the server");
})

socket.on('newEmail', (email) => {
    console.log('New email: ', email);
})
```

Kết quả

```
Connected to the server                                    index.js:4
New email:                                                 index.js:17
▶ {from: "hackagon@gmail.com", text: "Hey, Go fishing to
  day", createAt: 1539915371787}
```

EMITTING & LISTENING TO EVENTS

Truyền email từ **client** đến **server**:

- Phía client sẽ **emit** một sự kiện với tên 'createEmail' và thông tin dạng json

```
var socket = io();

socket.on('connect', () => {
    console.log("Connected to the server");

    socket.emit('createEmail', {
        to: 'hackagon@gmail.com',
        text: 'No, I want to be at home'
    })
})

socket.on('disconnect', () => {
    console.log("Disconnected from the server");
})

socket.on('newEmail', (email) => {
    console.log('New email: ', email);
})
```

EMITTING & LISTENING TO EVENTS

Truyền email từ **client** đến **server**:

- Phía server sẽ **listen** event 'createEmail' được gửi từ phía client và in ra ngoài terminal

```
io.on("connection", (socket) => {
    console.log("New user connected");

    socket.emit('newEmail', {
        from: 'hackagon@gmail.com',
        text: 'Hey, Go fishing today',
        createAt: new Date().getTime()
    })

    socket.on('createEmail', (newEmail) => {
        console.log('Email from client: ', newEmail);
    })

    socket.on("disconnect", () => {
        console.log("User was disconnected");
```

Kết quả

```
New user connected
Email from client:  { to: 'hackagon@gmail.com', text: 'No, I want
 to be at home' }
```

EXERCISE

Trong 2 ví dụ trên, chúng ta đã thực hiện việc **emit (phát)** ở phía server, **listen (nhận)** ở phía client và ngược lại.

Trong exercise này, thay vì emit và listen event là một email, các bạn hãy emit và listen một event là message, như vậy sẽ phù hợp hơn với project mà chúng ta đang xây dựng (chat app).

SOLUTION

Server

```
io.on("connection", (socket) => {
    console.log("New user connected");

    socket.emit('newMessage', {
        from: 'Hackagon',
        text: 'Hey go shopping today',
        createAt: new Date().getTime()
    })

    socket.on('createMessage', (message) => {
        console.log('Message: ', message);
    })

    socket.on("disconnect", () => {
        console.log("User was disconnected");
    })
})
```

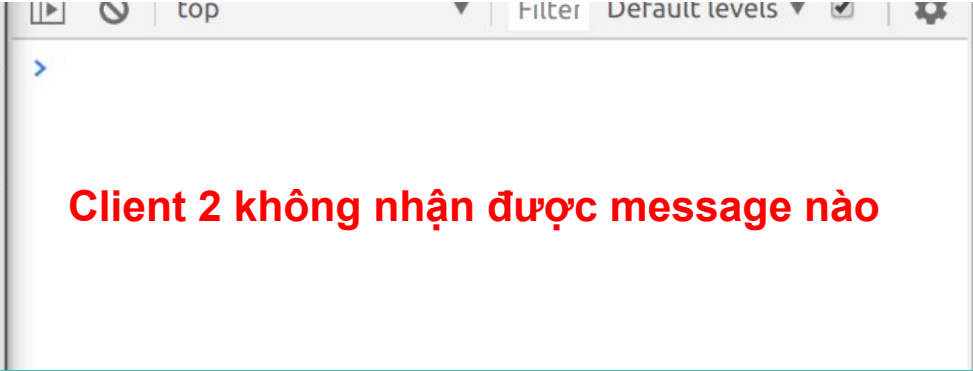
Client

```
var socket = io();

socket.on('connect', () => {
    console.log("Connected to the server");

    socket.emit('createMessage', {
        text: 'No, Today I am to busy',
        from: 'Ninja'
    })

})
socket.on('disconnect', () => {
    console.log("Disconnected from the server");
})

socket.on('newMessage', (message) => {
        console.log('New message: ', message)
    }
)
```

Socket.emit vs io.emit

- **Socket.emit**: server chỉ emit event đến 1 client mà server đó đang tương tác.

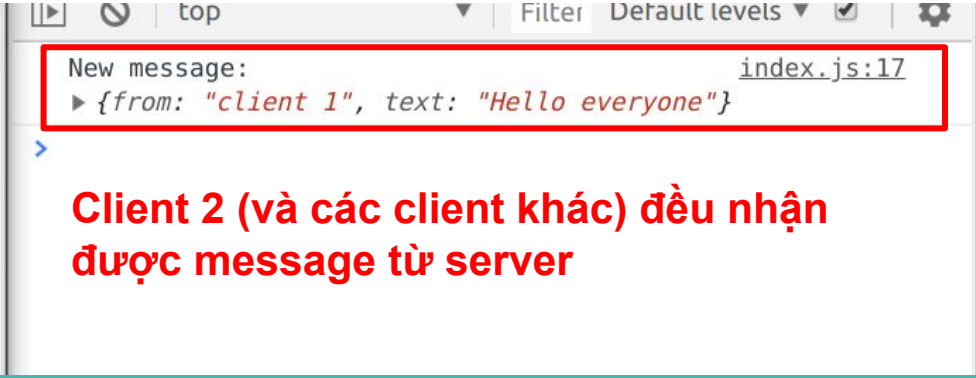
```
socket.on('createMessage', (message) => {
    console.log('Message: ', message);
    socket.emit('newMessage', {
        from: message.from,
        text: message.text
    })
})
```



Socket.emit vs io.emit

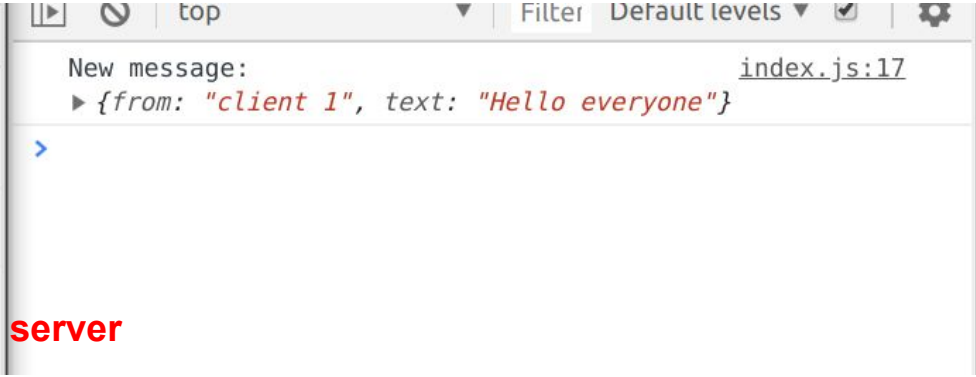
- **Io.emit**: server emit event đến tất cả clients đang kết nối với server.

```
socket.on('createMessage', (message) => {
    console.log('Message: ', message);
    io.emit('newMessage', {
        from: message.from,
        text: message.text
    })
})
```



BROADCASTING EVENTS

Vấn đề: khi sử dụng io.emit, thì client gửi message lên server lại nhận về chính message mà nó gửi, như vậy khá là "dị" khi làm một group chat



Client 1: nhận lại chính message mà nó gửi lên server

BROADCASTING EVENTS

Giải pháp: sự kiện broadcast giúp server gửi message đến những clients/ sockets đang không tương tác với server.

Các bạn có thể tự kiểm tra lại kết quả

```
socket.on('createMessage', (message) => {
    console.log('Message: ', message);
    // io.emit('newMessage', {
    //     from: message.from,
    //     text: message.text
    // })
    socket.broadcast.emit('newMessage', {
        from: message.from,
        text: message.text
    })
})
```

BROADCASTING EVENTS

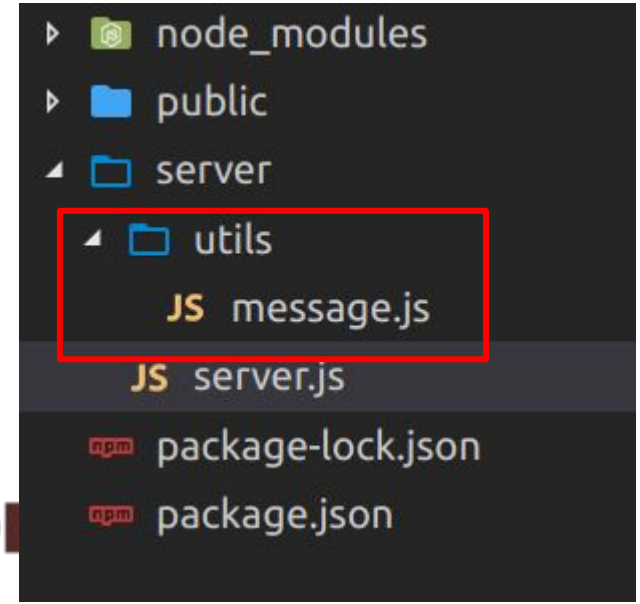
Vấn đề: chúng ta có thể sử dụng **broadcast** để thực hiện tính năng như sau: khi có một **client abc** mới truy cập vào hệ thống, server sẽ gửi thông báo "Welcome to the chat app" cho **clent abc** và gửi thông báo "New user join" cho các client còn lại (trừ **client abc**).

```
socket.emit('newMessage', {
    from: 'Admin',
    text: 'Welcome to the chat App',
    createAt: new Date().getTime()
})

socket.broadcast.emit('newMessage', {
    from: 'Admin',
    text: 'New user joined'
})
```

GENERATE MESSAGE

- Mục đích: tạo ra một hàm generateMessage() dùng để nhất quán cấu trúc của một message (bao gồm người gửi, nội dung, thời điểm lúc gửi). Tái sử dụng lại nhiều lần

```
const generateMessage = (from, text) => {
    return {
        from,
        text,
        createAt: new Date().getTime()
    }
}

module.exports = {
    generateMessage
}
```



GENERATE MESSAGE

```
io.on("connection", (socket) => {
    console.log("New user connected");

    socket.emit('newMessage', generateMessage('Admin', 'Welcome to the chat App'))

    socket.broadcast.emit('newMessage', generateMessage('Admin', 'New user joined'));

    socket.on('createMessage', (message) => {
        console.log('Message: ', message);
        // io.emit('newMessage', {
        //     from: message.from,
        //     text: message.text
        // })
        socket.broadcast.emit('newMessage', generateMessage(message.from, message.text));
    })

    socket.on("disconnect", () => {
        console.log("User was disconnected");
    })
})
```

ACKNOWLEDGEMENT EVENT

- Trong thực tế, làm sao để client biết được message của mình có thực sự được gửi đến server hay chưa, nếu chưa thì do lỗi gì? **Acknowledgement event** sẽ giúp tạo ra một thông báo, gửi từ phía server về lại client để thông báo message đã được nhận hay chưa.

```
socket.emit('createMessage', {
    from: 'Hieu',
    text: 'Ngay mai di sieu thi nhe'
}, (data) => {
    console.log('Sucess: ', data)
})
```

Phía client (index.js)

Callback function

```
socket.on('createMessage', (message, callback)
 => {
    console.log('Message: ', message);
    socket.broadcast.emit('newMessage',
    generateMessage(message.from,
    message.text)) ;
    callback('The message has ben sent');
})
```

Phía server

ACKNOWLEDGEMENT EVENT

- Test: Khi client gửi message đi thì server sẽ trả thông báo về

```
> socket.emit('createMessage',{text: 'Hello everyone', from:'client 1'},
(data) => console.log(data))
<· ▶ r {io: r, nsp: "/", json: r, ids: 2, acks: {…}, …}
  The message has ben sent                                    VM2223:1
>
```

Phía server gửi thông báo về

FORM & JQUERY

- Tạo form và sử dụng jquery để dom đến các thành phần của form

```
<!-- Vung hien thi noi dung chat -->
<ol id="messages"></ol>

<!-- O input cho nguoi dung nhap message -->
<form id="message-form">
    <input type="text" name="message" >
    <button>Send</button>
    <button type="button">Send Location</button>
</form>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
<script src="/socket.io/socket.io.js"></script>
<script src="./js/index.js"></script>
```

Vùng hiển thị nội dung chat

Form để người dùng nhập vào nội dung chat

JQuey CDN

FORM & JQUERY

- DOM đến button Send, thiết lập sự kiện cho button này khi người dùng click vào: tiến hành emit sự kiện 'createMessage' đến server, server sau khi nhận được sự kiện sẽ trả về cho tất cả client

```
$('#message-form').on('submit', (event) => {
    event.preventDefault();

    socket.emit('createMessage', {
        from: 'User',
        text: $('[name=message]').val()
    }, (data) => {
        console.log('Sucess: ', data)
    })
})
```

```
socket.on('createMessage', (message, callback) => {
    console.log('Message: ', message);
    io.emit('newMessage', generateMessage(message.from, message.text)) ;
    callback('The message has ben sent');
})
```

FORM & JQUERY

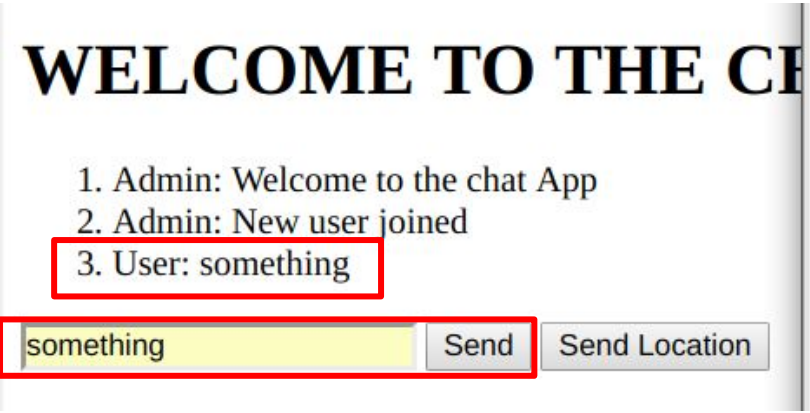
- Sau khi client emit event "createMessage" đến server, server ghi nhận và emit lại event "newMessage" đến tất cả client (bao gồm cả client gửi event)

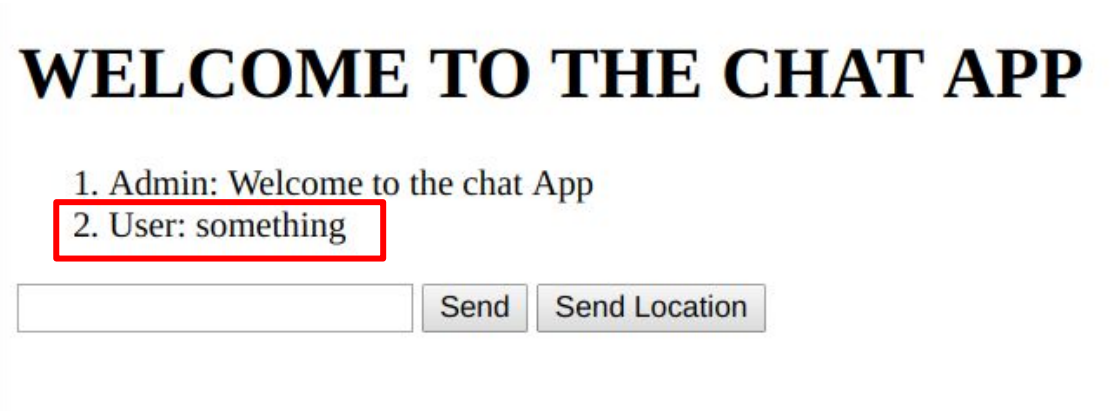
```
socket.on('newMessage', (message) => {
        console.log('New message: ', message)
        var li = $('<li></li>');
        li.text(`${message.from}: ${message.text}`);
        $('#messages').append(li);
    }
)
```

- Dom đến danh sách
- Tạo <li> với nội dung gồm người gửi (**message.from**) và nội dung tin nhắn (**message.text**) nhận <li> làm con

FORM & JQUERY

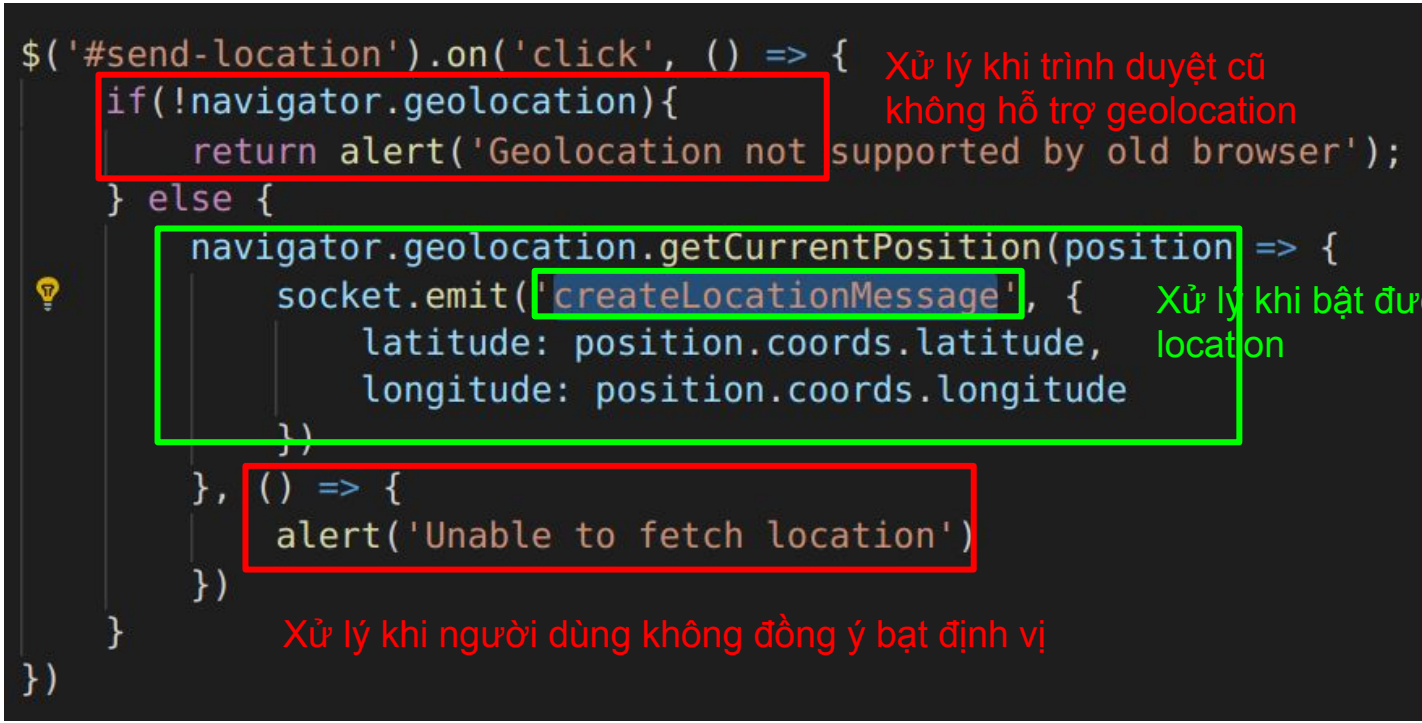
Test: Nội dung chat "something" được nhập từ client 1, đã hiển thị được trên khung chat của tất cả các clients.





GEOLOCATION

Tính năng: khi người dùng click vào button **Send location** thì thông tin về **latitude** và **longitude** sẽ được hiển thị ra khung chat

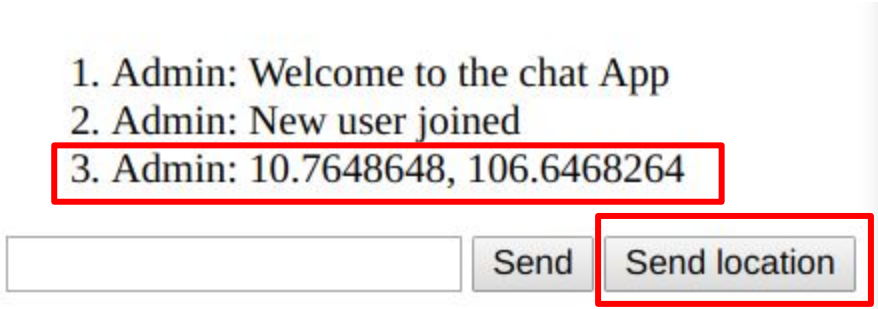


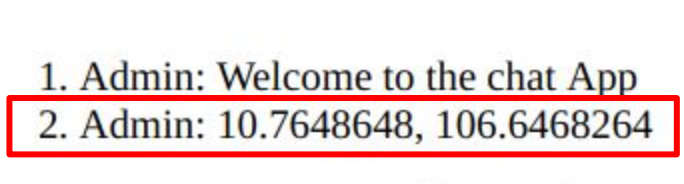
CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

GEOLOCATION

Server nhận được sự kiện "createLocationMessage" sau đó sẽ lấy thông tin này **emit** đến các client khác.

```
socket.on('createLocationMessage', (coords) => {
    io.emit('newMessage', generateMessage('Admin', `${coords.latitude},
    ${coords.longitude}`))
})
```





Khi click vào button **Send location** của client 1, thì thông tin về tọa độ của client 1 được hiển thị trên khung chat của tất cả clients

GEOLOCATION

Tạo thêm một cấu trúc message **generateLocationMessage** trong ./server/utils/message.js, export và require tại server.js

```
const generateLocationMessage = (from, latitude, longitude) => {
    return {
        from,
        url: `https://www.google.com/maps?q=${latitude},${longitude}`,
        createAt: new Date().getTime()
    }
}
```

```
socket.on('createLocationMessage', (message) => {
    io.emit('newLocationMessage', generateLocationMessage(message.from,
    message.latitude, message.longitude))
})
```

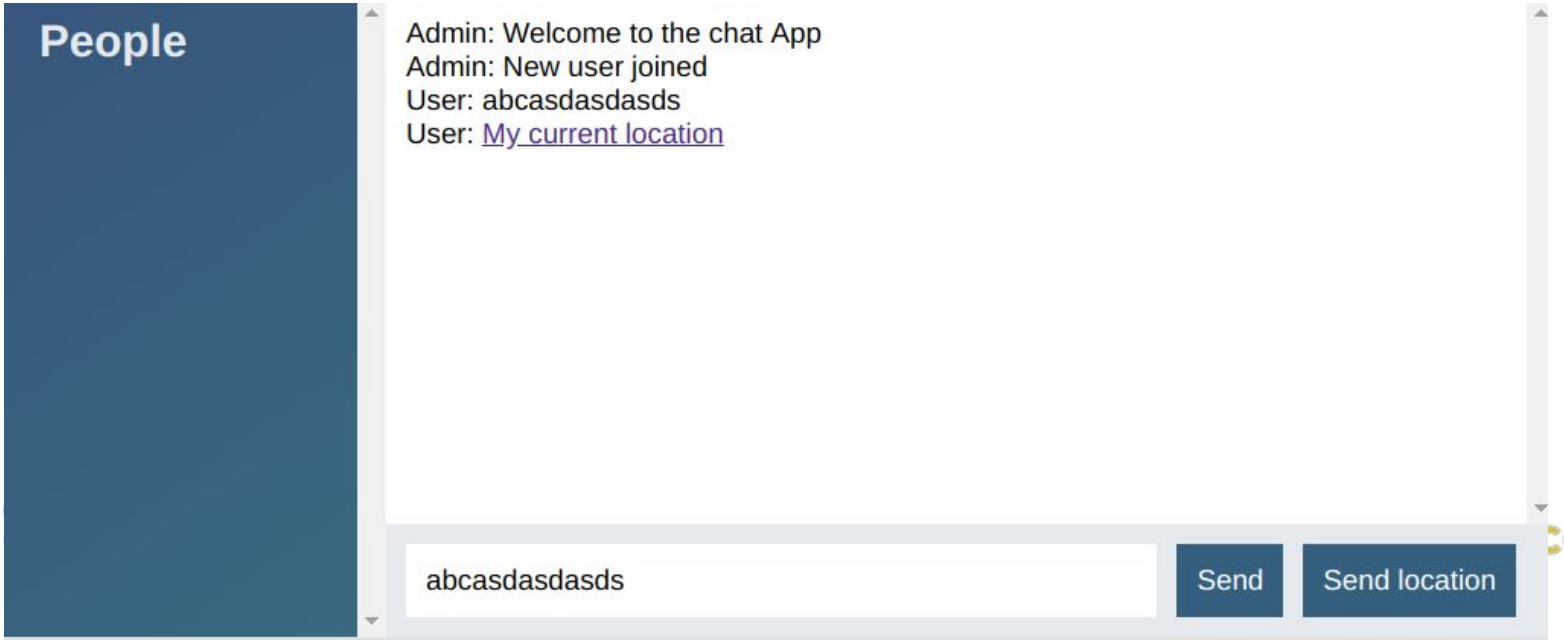
Thay đổi tại server

GEOLOCATION

Tại index.js, tiến hành nhận sự kiện newLocationMessage mà server emit, sau đó render ra khung chat với cấu trúc html như sau:

```
socket.on('newLocationMessage', (message) => {
    console.log(message)
    var li = $('<li></li>');
    var a = $('<a target="_blank">My current location</a>');
    li.text(`${message.from}: `) // User: My current location (a)
    a.attr('href', message.url);
    li.append(a);
    $('#messages').append(li);
})
```

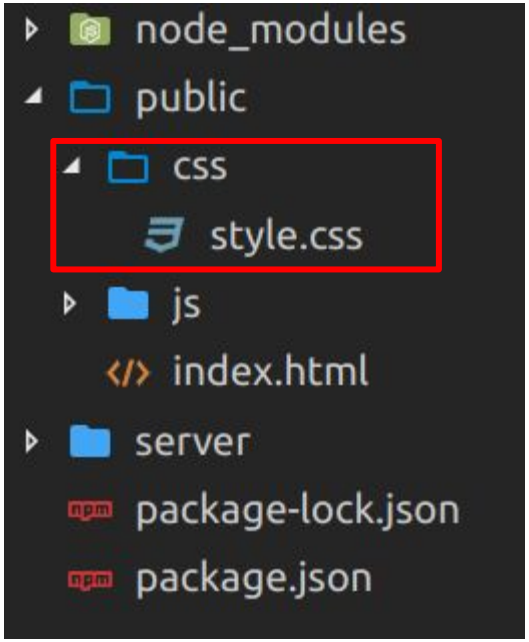
CSS chat app


People
Admin: Welcome to the chat App
Admin: New user joined
User: abcasdasdasds
User: My current location
abcasdasdasds
Send
Send location

CSS chat app

Vào trang: **http://links.mead.io/chat-css** lấy source css

```
    <link rel="stylesheet" href="./css/style.css">
</head>

<body class="chat">

    <div class="chat__sidebar">
        <h3>People</h3>
        <div id="users"></div>
    </div>

    <div class="chat__main">
        <ol id="messages" class="chat__messages"></ol>

        <div class="chat__footer">
            <form id="message-form">
                <input type="text" name="message" >
                <button>Send</button>
                <button type="button" id="send-location">Send location</button>
            </form>
        </div>
```



THAY ĐỔI NHỎ TĂNG UX

- Làm sao để autofocus vào ô input khi người dùng truy cập app?
- Làm sao để không hiển thị phần gợi ý, khi dùng focus vào ô input?
- Làm sao để xóa text trong ô input mỗi khi người dùng hit enter hoặc click submit?

Đây là những vấn đề cơ bản của html, js, các bạn có thể tự giải quyết, nên không trình bày chi tiết trong slide. Trong video trên codepro có hướng dẫn chi tiết.

MOMENT - FORMAT TIME

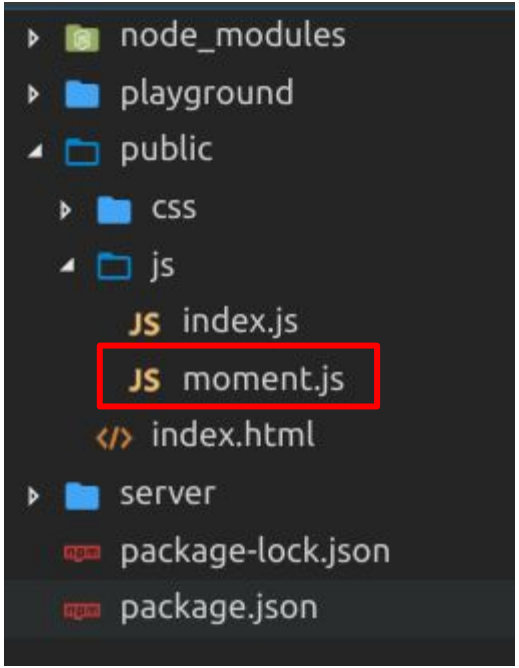
- **moment** là package giúp format time một cách hiệu quả, tiết kiệm thời gian
- Cài đặt: **npm install moment**
- moment().valueOf() cho kết quả giống hoàn toàn với new Date().getTime()

```
const generateMessage = (from, text) => {
    return {
        from,
        text,
        // createAt: new Date().getTime()
        createAt: moment().valueOf()
    }
}

const generateLocationMessage = (from, latitude, longitude) => {
    return {
        from,
        url: `https://www.google.com/maps?q=${latitude},${longitude}`,
        createAt: moment().valueOf()
    }
}
```

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

MOMENT - FORMAT TIME

- Copy file ./node_modules/moment/**moment.js** vào ./public/js `<script src="./js/moment.js"></script>`
- Import vào index.html
- Muốn thay đổi format thì sử dụng phương thức moment().**format()**
- Muốn thay đổi định dạng ngày giờ thành **2:30 PM** thì format theo **moment().format('h:mm a')**

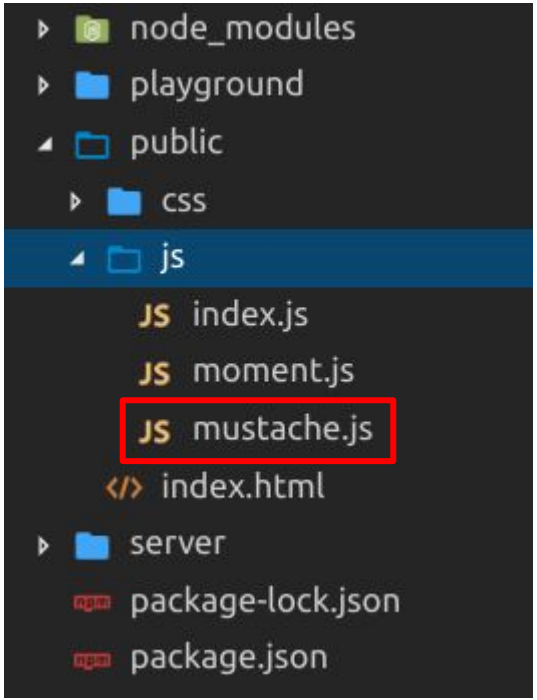


MOMENT - FORMAT TIME

```
socket.on('newMessage', (message) => {
        // console.log('New message: ', message)
        var formattedTime = moment(message.createAt).format('h:mm a');
        var li = $('<li></li>');
        li.text(`${message.from} ${formattedTime}: ${message.text}`);
        $('#messages').append(li);
    }
)

socket.on('newLocationMessage', (message) => {
    // console.log(message)
    var formattedTime = moment(message.createAt).format('h:mm a');
    var li = $('<li></li>');
    var a = $('<a target="_blank">My current location</a>');
    li.text(`${message.from} ${formattedTime}: `) // User: My current location (a)
    a.attr('href', message.url);
    li.append(a);
    $('#messages').append(li);
})
```

MUSTACHE

- Đây là một template engine khá phổ biến (tương tự như EJS)
- Trong bài này, sử dụng **mustache** tại client side
- Truy cập: https://github.com/janl/mustache.js
- Tìm và download file mustache.js vào trong thư mục ./public/js



MUSTACHE

Sử dụng **mustache** để render cấu trúc html cho message (khi người dùng click submit)

- Chèn thư viện mustache vào
- Tạo cấu trúc html trong <script> tag như hình bên
- Gía trị bên trong {{...}} là giá trị động, có được từ js gửi qua.

```
<script src="https://code.jquery.com/jquery-3.3
<script src="/socket.io/socket.io.js"></script>
<script src="./js/moment.js"></script>
<script src="./js/mustache.js"></script>
<script src="./js/index.js"></script>
```

```
<script id="message-template" type="text/template">
    <li class="message">
        <div class="message__title">
            <h4>{{from}}</h4>
            <span>{{createAt}}</span>
        </div>
        <div class="message__body">
            <p>{{text}}</p>
        </div>
    </li>
</script>
```

MUSTACHE

- DOM đến script **message-template**
- Tiến hành render với các liệu (JSON) cần thiết
- append

```
socket.on('newMessage', (message) => {
        var template = $('#message-template').html();
        var html = Mustache.render(template, {
            text: message.text,
            from: message.from,
            createAt: moment(message.createAt).format('h:mm a')
        })

        $('#messages').append(html)

        // var formattedTime = moment(message.createAt).format('h:mm a');
        // var li = $('<li></li>');
        // li.text(`${message.from} ${formattedTime}: ${message.text}`);
        // $('#messages').append(li);
    }
)
```

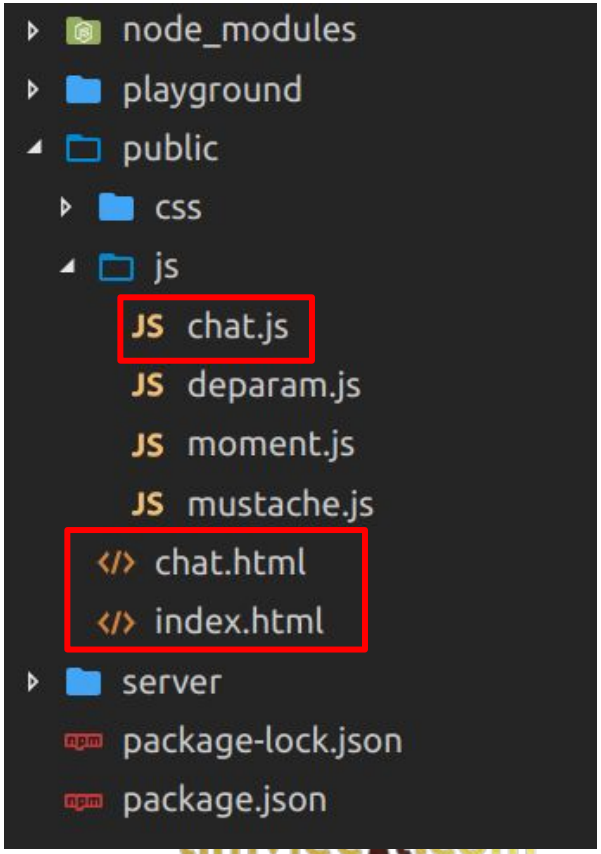
EXERCISE

Yêu cầu: Làm tương tự **message-template**, tạo một cấu trúc **location-message-template** để render ra cấu trúc html khi người dùng click **Send location**

Chi tiết bài giải có trong video trên CodePro, nên không trình bày trong slide này

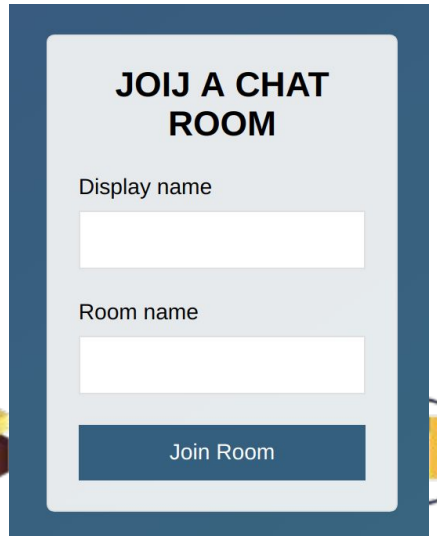
JOIN PAGE

- Rename index.html → chat.html, index.js → chat.js
- Tạo trang index.html mới



JOIN PAGE

Trong **index.html** sử dụng các class được định nghĩa sẵn (trong style.css)

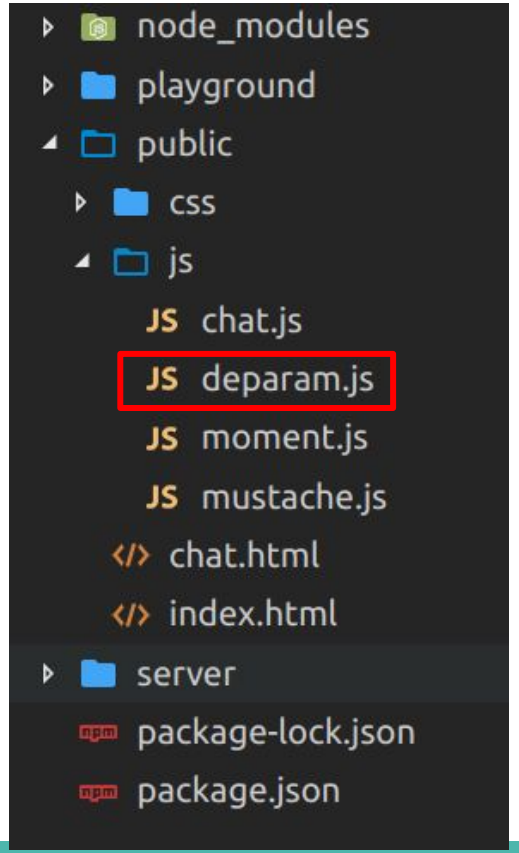



```
<body class="centered-form">
    <div class="centered-form__form">
        <form action="/chat.html">
            <div class="form-field">
                <h3>JOIJ A CHAT ROOM</h3>
            </div>

            <div class="form-field">
                <label for="name">Display name</label>
                <input type="text" name="name" autofocus id="name" />
            </div>

            <div class="form-field">
                    <label for="room">Room name</label>
                    <input type="text" name="room" id="room" />
            </div>

            <div class="form-field">
                <button>Join Room</button>
            </div>
        </form>
    </div>
</body>
```

WELCOME USER JOIN A ROOM

- Để parse được một queryString thành jsonObject, sử dụng thư viện deparam trong link sau: http://links.mead.io/deparam
- Trong ./public/js/chat.js, thực hiện việc emit sự kiện "join" với nội dung params là JSON object gồm **name** và **room**.

```
socket.on('connect', () => {
    console.log("Connected to the server");
    socket.emit('join', {
        params: $.deparam(window.location.search)
    })

})
```



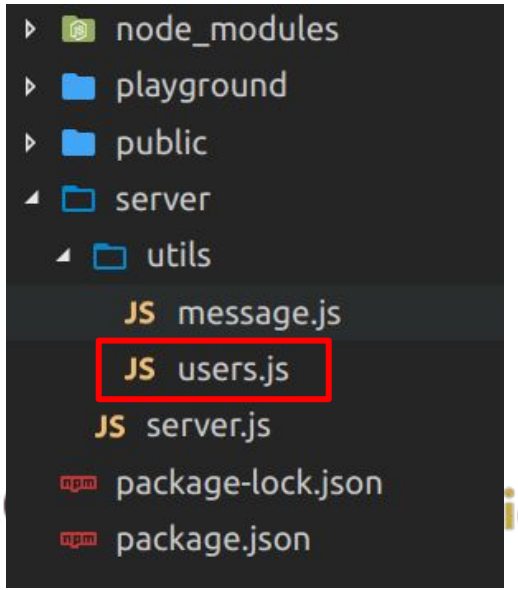
WELCOME USER JOIN A ROOM

- Tại server **listen** sự kiện "join", và **emit** sự kiện "newMessage".
- Chú ý: để gửi message chỉ đến với những người đang ở trong cùng 1 room, dùng phương thức socket.broadcast.**to(<TÊN ROOM>)**.emit hoặc io.**to(<TÊN ROOM>)**.emit

```
socket.on('join', (join) => {
    let { room, name } = join.params;
    socket.join(room);

    socket.emit('newMessage', generateMessage('Admin', `Welcome to the ${room}
    room`))
    socket.broadcast.to(room).emit('newMessage', generateMessage('Admin', `${name}
    joined`));
})
```

XÂY DỰNG LỚP ĐỐI TƯỢNG USERS

- Mục đích: quản lý việc một client truy cập vào phòng chat (thêm), client hủy kết nối đến phòng chat (xóa), lấy danh sách clients và hiển thị (đọc)



XÂY DỰNG LỚP ĐỐI TƯỢNG USERS

```
class Users{
    constructor(){
        this.listOfUsers = []
    }
    addUser(id, name, room){
        var user = {id, name, room};
        this.listOfUsers.push(user);
    }
    getUserById(id){
        var user = this.listOfUsers.filter(user => user.id === id)[0]
        return user
    }
    removeUser(id){
        var user = this.getUserById(id);
        var theList = this.listOfUsers.filter(user => user.id !== id);
        this.listOfUsers = theList;
        return user;
    }
    getListOfUsersInRoom(room){
        var theList = this.listOfUsers.filter(user => user.room === room);
        return theList;
    }
}
module.exports = { Users }
```

Thêm user

Tìm kiếm user theo id

Xóa user theo id, return về user bị xóa để có thể gửi thông báo đến các clients khác: user/client bị xóa đã rời phòng chat

Lấy danh sách user trong cùng một phòng chat

Xuất lớp đối tượng Users

THÊM USER

```
const { Users } = require('./utils/users');
var users = new Users(); Khởi tạo đối tượng users
```

Thêm user khi có một client truy cập vào phòng chat. Tại **server.js:**

```
socket.on('join', (join) => {
    let { room, name } = join.params;
    socket.join(room);

    users.addUser(socket.id, name, room);
    io.to(room).emit('usersInRoom', {
        usersInRoom: users.getListOfUsersInRoom(room)
    })

    socket.emit('newMessage', generateMessage('Admin', `
    socket.broadcast.to(room).emit('newMessage', generat
})
```

Thêm một user vào danh sách users

Emit sự kiện "usersInRoom" đến tất cả các client đang ở cùng phòng chat mà user mới vừa được add vào

XÓA USER

Xóa user khi có một client ngắt kết nối phòng chat. Tại **server.js:**

```
socket.on("disconnect", () => {
    var user = users.removeUser(socket.id);
    console.log(user);
    if(user){
        io.to(user.room).emit('usersInRoom', {
            usersInRoom: users.getListOfUsersInRoom(user.room)
        })
        io.to(user.room).emit('newMessage',
        generateMessage('Admin', `${user.name} has left the room`))
    }

    console.log("User was disconnected");
})
```

Phương thức removerUser return về user bị xóa

update lại danh sách user đang ở trong phòng

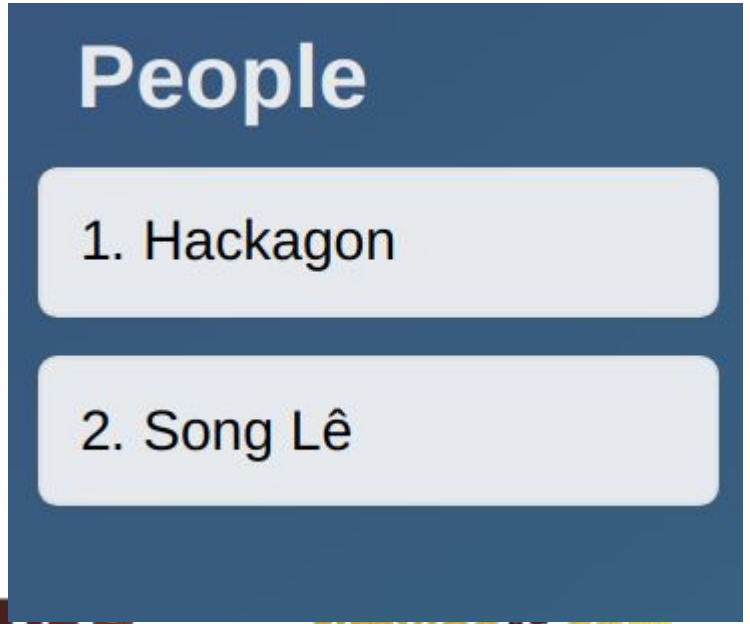
Gửi thông báo: đã có user nào đó rời khỏi phòng

HIỂN THỊ DANH SÁCH USER

Tại **chat.js**:

```
socket.on('usersInRoom', (message) => {
    var users = message.usersInRoom;
    console.log(users);
    var ol = $('<ol></ol>')
    users.forEach(user => {
        var li = $('<li></li>');
        li.text(user.name);
        ol.append(li);
    })
    $('#users').html(ol);
})
```

DOM đến thẻ ở chat.html



EXERCISE

Đến đây chúng ta còn 3 vấn đề cần giải quyết, 3 vấn đề này khá đơn giản và đã trình bày trong video trên CodePro nên không trình bày trong slide này. Các bạn có thể tự giải quyết.

1. Khi chat nhiều, xuất hiện scrollbar đứng, hãy dùng jQuery để scrollbar đứng luôn autoscroll xuống bottom.
2. Khi chat, hiển thị đúng tên người chat mà client cung cấp (không để "User")
3. Khi chat, chỉ những người trong cùng 1 phòng chat mới có thể chat với nhau, thông tin chat không gửi đến những người ở phòng chat khác.

TÓM TẮT

- Websocket và socket.io
- Sự kiện **connect**, **disconnect** ở phía server và client
- **Custom một sự kiện**
- Emiting và Listening một sự kiện
- Emiting một sự kiện đến một room riêng biệt
- Sử dụng **geolocation** để định vị
- Áp dụng Jquey để DOM đến các thành phần html
- Template engine **MUSTACHE**

CẢM ƠN CÁC BẠN ĐÃ THEO DÕI