CHAT APPLICATION

A multi-client chat application using Node.js

Abstract

In this document we implement a real-time multi client chat application with node.js and socket.io. Html and CSS was used to develop the front-end..

Amartya Choudhury
JU-BCSE IV
Roll – 001610501026
Internet Technology Lab

Problem Statement:

Write a multi-client chat application consisting of both client and server programs. In this chat application simultaneously several clients can communicate with each other. For this you need a single server program that clients connect to. The client programs send the chat text or image (input) to the server and then the server distributes that message (text or image) to all the other clients. Each client then displays the message sent to it by the server. The server should be able to handle several clients concurrently. It should work fine as clients come and go.Develop the application using a framework based on Node.JS. How are messages handled concurrently? Which web application framework(s) did you follow?

Prepare a detailed report of the experiments you have done, and your observations on performance of the system.

Node.js

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices

Following are some of the important features that make Node.js the first choice of software architects.

- **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
- **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.

Node.js has proved to be useful for I/O intensive applications. However it should not be used for CPU intensive applications.

Callbacks

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

Socket.io

Socket.IO is a JavaScript library for **real-time web applications**. It enables real-time, bi-directional communication between web clients and servers. It has two parts: a **client-side library** that runs in the browser, and a **server-side library** for node.js. Both components have an identical API.

Real-time applications

A real-time application (RTA) is an application that functions within a period that the user senses as immediate or current.

Some examples of real-time applications are −

- **Instant messengers** − Chat apps like Whatsapp, Facebook Messenger, etc. You need not refresh your app/website to receive new messages.
- **Push Notifications** − When someone tags you in a picture on Facebook, you receive a notification instantly.
- **Collaboration Applications** − Apps like google docs, which allow multiple people to update same documents simultaneously and apply changes to all people's instances.
- **Online Gaming** − Games like Counter Strike, Call of Duty, etc., are also some examples of real-time applications.

Why Socket.IO?

Writing a real-time application with popular web applications stacks like LAMP (PHP) has traditionally been very hard. It involves polling the server for changes, keeping track of timestamps, and it is a lot slower than it should be.

Sockets have traditionally been the solution around which most real-time systems are architected, providing a bi-directional communication channel between a client and a server. This means that the server can push messages to clients. Whenever an event occurs, the idea is that the server will get it and push it to the concerned connected clients.

ExpressJS

We will be using express to build the web server that Socket.IO will work with. Any other node-server-side framework or even node HTTP server can be used.

Server side Design:

package.json

```
{
  "name": "chat-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.4",
    "mongodb": "^3.3.2",
  }
}
```

messages.js

```
const generateMessage = (username,text) =>{
    return {
        username,
        text ,
        createdAt : new Date().getTime()
    }
}
const generateLocationMessage = (username,url) =>{
    return {
        username,
        url,
        createdAt : new Date().getTime()
    }
}
const generateImageMessage = (username,dataUrl) =>{
    return {
        username,
        dataUrl,
        createdAt : new Date().getTime()
    }
}
```

```
module.exports ={
    generateMessage ,
    generateLocationMessage,
    generateImageMessage
}
```

users.js

```
const users = []

const addUser = ({ id, username, room }) => {
    // Clean the data
    username = username.trim().toLowerCase()
    room = room.trim().toLowerCase()

    // Validate the data
    if (!username || !room) {
        return {
            error: 'Username and room are required!'
        }
    }

    // Check for existing user
    const existingUser = users.find((user) => {
        return user.room === room && user.username === username
    })

    // Validate username
    if (existingUser) {
        return {
            error: 'Username is in use!'
        }
    }

    // Store user
    const user = { id, username, room }
    users.push(user)
    return { user }
}

const removeUser = (id) => {
    const index = users.findIndex((user) => user.id === id)
```

```
    if (index !== -1) {
        return users.splice(index, 1)[0]
    }
}

const getUser = (id) => {
    return users.find((user) => user.id === id)
}

const getUsersInRoom = (room) => {
    room = room.trim().toLowerCase()
    return users.filter((user) => user.room === room)
}

module.exports = {
    addUser,
    removeUser,
    getUser,
    getUsersInRoom
}
```

index.js

```
const path = require('path')
const http = require('http')
const express = require('express')
const socketio = require('socket.io')
const { generateMessage , generateLocationMessage,generateImageMessage} = require
('./utils/messages.js')
const {addUser,removeUser,getUser,getUsersInRoom} = require('./utils/users.js')
const app = express()
const server = http.createServer(app)
const io = socketio(server)

const port = process.env.PORT || 3000
const publicDirectoryPath = path.join(__dirname,'../public')

app.use(express.static(publicDirectoryPath))

io.on('connection',connect)
function connect(socket){
    console.log('New websocket connection')
    socket.on('join',(options,callback) =>{
```

```
        const {error,user} = addUser( {id:socket.id,...options})
        if(error){
            return callback(error)
        }
        socket.join(user.room)
        socket.emit('message',generateMessage('Admin','Welcome!'))
        socket.broadcast.to(user.room).emit('message',generateMessage('Admin',use
r.username + 'has joined'))    // send joining notifcation to all other users
        io.to(user.room).emit('roomData',{
            room :user.room,
            users:getUsersInRoom(user.room)
        })
        callback()
    })
    socket.on('sendMessage',(message,callback)=>{
        const user = getUser(socket.id)
        io.to(user.room).emit('message',generateMessage(user.username,message))
        callback('Delivered')
    })

    socket.on('sendLocation',(coords,callback) =>{
        const user = getUser(socket.id)
        io.to(user.room).emit('locationMessage',generateLocationMessage(user.user
name,'https://www.google.com/maps?q=' + coords.latitude + ',' + coords.longitude)
)
        callback()
    })

    socket.on('sendImage',(imageUrl,callback) =>{
        const user = getUser(socket.id)
        io.to(user.room).emit('imageMessage',generateImageMessage(user.username,i
mageUrl))
        callback()
    })
    socket.on('disconnect',() =>{
        const user = removeUser(socket.id)
        if(user){
            io.to(user.room).emit('message',generateMessage(user.username + 'A us
er has left'))
            io.to(user.room).emit('roomData',{
                room :user.room,
                users:getUsersInRoom(user.room)
            })
        }
    })
```

```
}

server.listen(port,() =>{
    console.log('Server is up and running on port '+ port)
})
```

We set up the express application to run on port 3000. The application will serve an “index.html” file on the root route . Now we require Socket.IO and will log "A user connected", every time a user goes to this page and "A user disconnected", every time someone navigates away/closes this page.

The require('socket.io')(http) creates a new socket.io instance attached to the http server. The **io.on event handler** handles connection, disconnection, etc., events in it, using the socket object.

We have set up our server to log messages on connections and disconnections. We now have to include the client script and initialize the socket object there, so that clients can establish connections when required. The script is served by our **io server** at **'/socket.io/socket.io.js'**.

When a new user joins a room, the use is added to the user list in the server and the server **broadcasts** the information to all clients in the room.Except the new joinee . When client sends a message , an image , or his location the server **broadcasts the message to everyone** in the room.

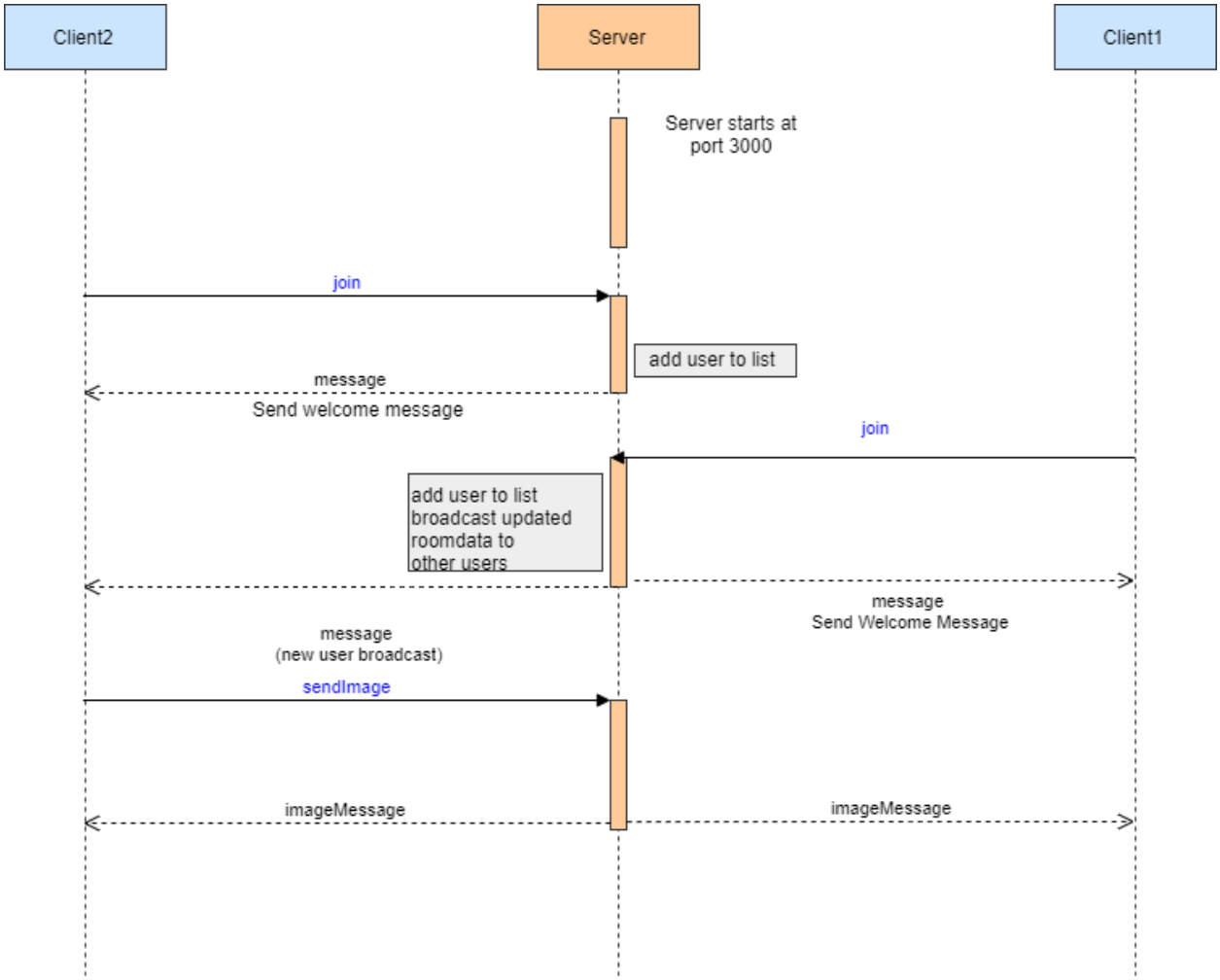

Fig 1 : Sequence diagram showing chat using websockets .

Client-side design :

index.html

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Chat App</title>
        <link rel="icon" href="/img/favicon.png">
        <link rel="stylesheet" href="/css/styles.min.css">
    </head>
    <body>
        <div class="centered-form">
            <div class="centered-form__box">
```

```
                <h1>Join</h1>
                <form action="/chat.html">
                <label>Display Name</label>
                <input type="text" name="username" placeholder="Display Name" req
uired>
                <label>Room</label>
                <input type="text" name="room" placeholder="room" required>
                <button>Join</button>
                </form>
            </div>
        </div>
    </body>
</html>
```

A user who wants to enter a room to chat enters his username and room name .The user is redirected to chat.html.

chat.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Chat App</title>
        <link rel="icon" href="/img/favicon.png">
        <link rel="stylesheet" href="/css/styles.min.css">
    </head>
    <body>
        <div class="chat">
            <div id = "sidebar" class = "chat__sidebar">
            </div>
            <div class = "chat__main">

                <div id=past-messages class ="chat__messages"></div>

                <div class="compose">
                    <form id="message-form">
                        <input name="message" placeholder="message">
                        <button>Send</button>
                    </form>
                    <button id="send-location">Send Location</button>
                    <input type='file' accept='image/*' onchange='openFile(event)
'>
                    <img id='output'>
                    <script>
```

```
                    var openFile = (event) => {
                        var input = event.target;
                        var reader = new FileReader();
                        reader.onload = function(){
                            var dataURL = reader.result;
                            socket.emit("sendImage",dataURL,(error) =>{
                                console.log(error)
                            })
                            };
                    reader.readAsDataURL(input.files[0]);
                    };
                    </script>
                </div>
            </div>
        </div>

        <script id=message-template type=text/html>
            <div class="message">
                <p>
                    <span class = "message__name">{{username}}</span>
                    <span class = "message__meta">{{createdAt}}</span>
                </p>
                <p>{{message}}</p>
            </div>
        </script>

        <script id="location-template" type =text/html>
            <div class="message">
                <p>
                    <span class = "message__name">{{username}}</span>
                    <span class = "message__meta">{{createdAt}}</span>
                </p>
                <p><a href ="{{url}}" target = "_blank">My Current Location</a></
p>
            </div>
        </script>
        <script id="image-template" type =text/html>
            <div class="message">
                    <p>
                        <span class = "message__name">{{username}}</span>
                        <span class = "message__meta">{{createdAt}}</span>
                    </p>
                    <img src = "{{url}}">
            </div>
        </script>
```

```
        <script id="sidebar-template" type="text/html">
            <h2 class ="room-title">{{room}}</h2>
            <h3 class ="list-title">Users</h3>
            <ul class="users">
                {{#users}}
                    <li>{{username}}</li>
                {{/users}}
            </ul>
        </script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/mustache.js/3.0.1/mustache.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.22.2/moment.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/qs/6.6.0/qs.min.js"></script>
        <script src="/socket.io/socket.io.js"></script>
        <script src="/js/chat.js"></script>

    </body>
</html>
```

chat.js

```
const socket = io()

//elements
const $messageform = document.querySelector('#message-form')
const $messageformInput = $messageform.querySelector('input')
const $messageformButton = $messageform.querySelector('button')
const $sendLocationButton =document.querySelector('#send-location')
const $messages = document.querySelector('#past-messages')

//template
const messageTemplate = document.querySelector('#message-template').innerHTML
const locationTemplate = document.querySelector('#location-template').innerHTML
const sidebarTemplate = document.querySelector('#sidebar-template').innerHTML
const imageTemplate = document.querySelector('#image-template').innerHTML
const { username , room } = Qs.parse(location.search,{ignoreQueryPrefix:true})

const autoscroll = () =>{
    const $newMessage = $messages.lastElementChild
```

```
    const newMessageStyles = getComputedStyle($newMessage)
    const newMessageMargin = parseInt(newMessageStyles.marginBottom)
    const newMessageHeight = $newMessage.offsetHeight + newMessageMargin
    const visibleHeight = $messages.offsetHeight
    const containerHeight = $messages.scrollHeight
    const scrollOffset = $messages.scrollTop + visibleHeight
    if(containerHeight - newMessageHeight <= scrollOffset){
        $messages.scrollTop = $messages.scrollHeight
    }
}
socket.emit('join',{username,room},(error)=>{
    if(error){
        alert(error)
        location.href = '/'
    }
})
socket.on('message',(message)=>{
    console.log(message)
    const html =Mustache.render(messageTemplate,{
       username : message.username,
        message: message.text,
       createdAt: moment(message.createdAt).format('h:mm a')
    })
    console.log(html)
    $messages.insertAdjacentHTML('beforeend',html)
    autoscroll()
})

socket.on('locationMessage',(message)=>{
    console.log(message)
    const html =Mustache.render(locationTemplate,{
       username:message.username,
        url: message.url,
       createdAt: moment(message.createdAt).format('h:mm a')
    })
    console.log(html)
    $messages.insertAdjacentHTML('beforeend',html)
    autoscroll()
})

socket.on('imageMessage',(message) =>{
    console.log(message)
    const html = Mustache.render(imageTemplate,{
        username:message.username,
        url:message.dataUrl,
```

```
        createdAt : moment(message.createdAt).format('h:mm a')
    })
    console.log(html)
    $messages.insertAdjacentHTML('beforeend',html)
    autoscroll()
})
if($messageform){
    $messageform.addEventListener('submit',(e)=>{
        e.preventDefault()
        $messageformInput.setAttribute('disabled','disabled')
        message = e.target.elements.message.value
        socket.emit('sendMessage',message,(error) =>{
            $messageformInput.removeAttribute('disabled')
            $messageformInput.value = ''
            $messageformInput.focus()
            if(error){
                console.log(error)
                return
            }
            console.log('Message Delivered')
        })
    })
}

if($sendLocationButton){
    $sendLocationButton.addEventListener('click',()=>{
    if(!navigator.geolocation){
        console.log("geolocation is not supported by the browser")
    }

    else{
        $sendLocationButton.setAttribute('disabled','disabled')
        navigator.geolocation.getCurrentPosition((position)=>{
            console.log(position)
            socket.emit('sendLocation',{
                latitude:  position.coords.latitude,
                longitude: position.coords.longitude
            },()=>{
                $sendLocationButton.removeAttribute('disabled')
                console.log('Location shared')
            })
        })
    }
})
```

We initialize the socket object so that client can initialize the connection with the server when the server . On different button clicks the socket emits different events with their corresponding callbacks .The events are handled by the server using event handlers and responses are broadcasted to all the users connected to the room .Client side event handlers display the messages sent from the server .

Output:

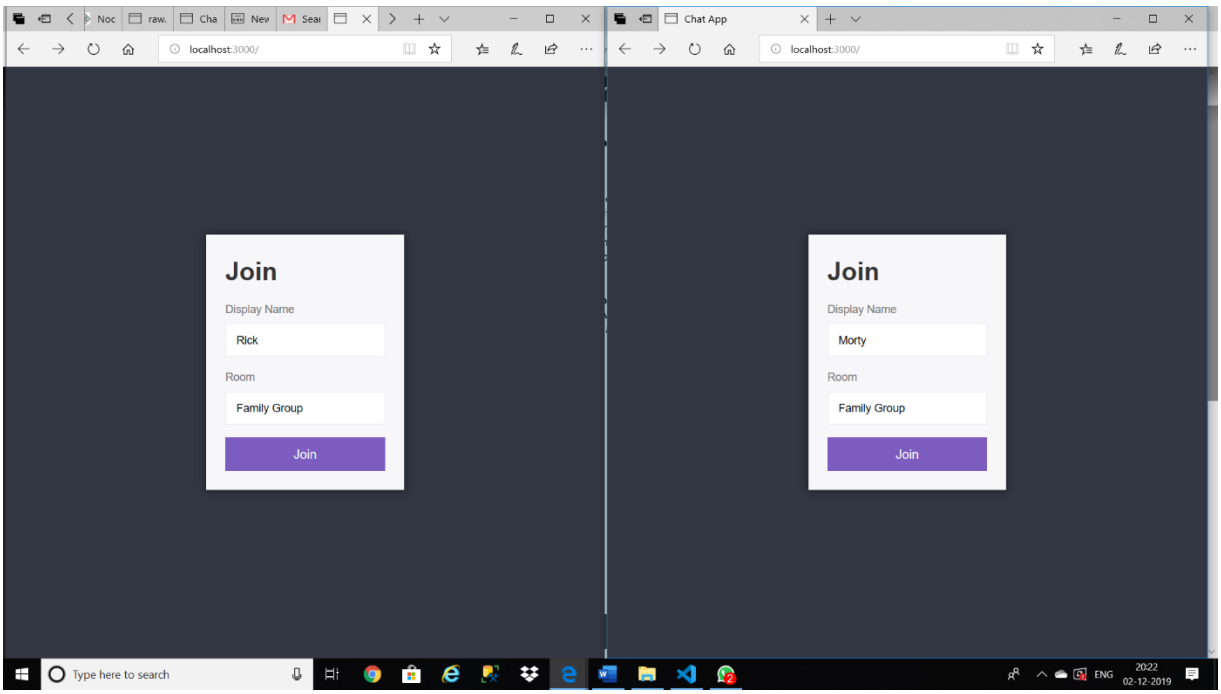

Fig 2: Both Rick and Morty enter the room “Family Group”

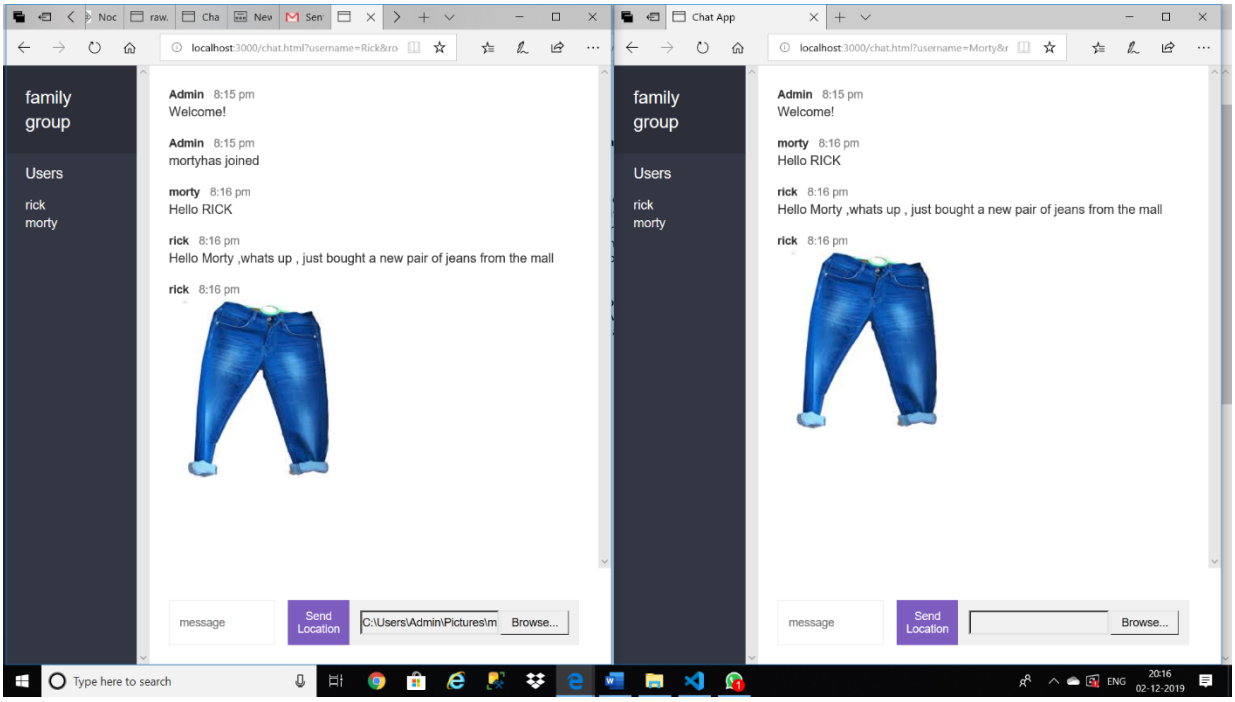

Fig 3: Conversation

References:

- https://socket.io/
- https://nodejs.org/en/about/
- https://expressjs.com/