Name :- Dibyendu Mukhopadhyay
Class :- BCSE-IV Group :- A3 **RollNo.** :- 001710501077
Subject - Internet Technology Lab

Assignment No. :- **02**

Problem Statement

Assignment 2: Implement a multi client chat application

Submission due: 02-04 November 2020

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 the protocol layers thus created.

Solution Approach:-

The aim of this lab is to create a collaborative chat for multiple users with the help of using node.js (using packages like socket.io for managing websockets and express for deploying the server) and basic HTML and javascript (using DOM HTML) for the client side.

In order to deploy, you will need to install both node.js and npm (node package manager). After installing these, you will need to download the following packages by using npm :- express, nodemon and socket.io.

1. Express is the micro web framework for nodejs.
2. Nodemon helps to detect the changes and restart the server.
3. Socketio is a powerful package which manages the websocket.

I have developed two sides to make the chat application running :- client side and server side. The node.js handles the server side to solve the backend task, while the client side will be loaded directly by the user's device.

For implementation, first create the npm project using command “npm init” and then install the required dependencies for running the application.

Server side :-

It is implemented in node.js, socket.io and express which handles the messages concurrently. Responsible for the following objectives:-

1. When a user joins the chat, he/she is greeted.
2. All connected users are notified who enters the chat room.
3. Users can broadcast the message.
4. List of the active users are shown.

```
const express = require('express');
const app = express();
const http = require('http').createServer(app)
const PORT = process.env.PORT || 3000

const users = {};

var usernames = [] ; //list of the usernames
var connections = []; // connection by usernames

http.listen(PORT, () =>
{
    console.log(`Listening on port ${PORT}`);
});

app.use(express.static(__dirname + "/utils"));
app.get('/',(reg,res) =>
{
    res.sendFile(__dirname + "/utils/index.html");
});

const io = require('socket.io')(http)
io.on('connection', socket =>
{
    //check if the connection is successful
    console.log("Connection Success...");
    // countClients++;
    connections.push(socket);
    console.log("Connected : %s sockets are connected",
connections.length);

    //when new user connected
    socket.on('new-user', user_name =>
    {
        usernames.push(user_name);
        updateUserNames();
        users[socket.id] = user_name

        console.log(user_name + " joins");
        console.log("Current Active users :-" + usernames);
        // console.log(countClients + " User(s) Online");

        socket.broadcast.emit('user-connected', user_name);
    });

    //updating usernames when connection and disconnecting
    function updateUserNames(){
        io.sockets.emit('usernames',usernames);
    }

    //when user send message, server then broadcasted it those who
are connected
    socket.on("message", (msg) =>
    {
        console.log(msg);
        socket.broadcast.emit("message", msg);

    });

    //when user send the image, server then bradcasted it those
who are connected
    socket.on("image", (imgData) =>
    {
        console.log("broadcasting image...");
        socket.broadcast.emit("image",imgData);
    });

    //when user left from the server, server let them know that
user left
    socket.on('disconnect', () =>
    {

        usernames.splice(usernames.indexOf(users[socket.id]), 1);
        connections.splice(connections.indexOf(socket), 1);

        console.log("Connected : %s sockets are connected",
connections.length);
        updateUserNames();

        console.log(users[socket.id] + " left");
        socket.broadcast.emit('user-disconnected',
users[socket.id]);
        // delete users[socket.id];
        // updateUserNames();

        console.log("Current Active users :-" + usernames);
        // console.log(`${--countClients} User(s) Online`);
```

```
     });
});
```

The first event that our server will receive on a new client connection is connection. connection events handlers, pass along the socket that was just established. The socket handles the following events:

- **socket.on('message', callback)** – callback is called when a new message is received. message can be any type of data, depending on what was sent.
- **socket.on('disconnect', callback)** – callback is called when the socket disconnects.
- **socket.emit('message', args)** – Send message over the socket. *[UNICAST]*
- **socket.broadcast.send('message', args)** – Broadcasts message to all sockets except the sender. *[MULTICAST]*
- **io.sockets.emit('message', args)** – Broadcasts message to all clients, even the sender too. *[BROADCAST]*

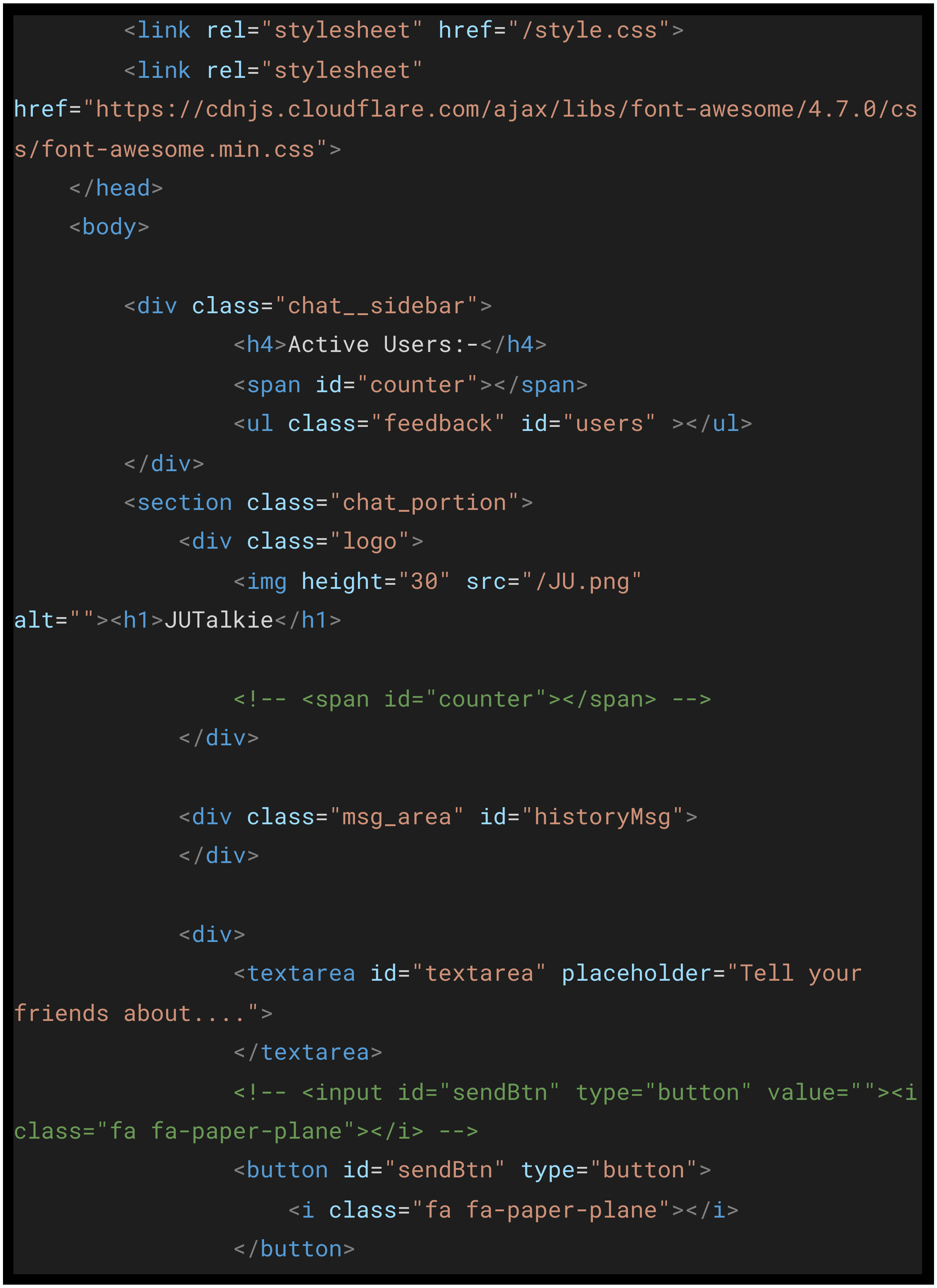
Client Side:-

HTML and CSS files :-

The html file is used to build the structure of the websites and its contents. The CSS file is here to make the web page more user friendly and good looking which will improve the user experience.

Index.html:-

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>
            JUTalkie
        </title>
```



```html
        <link rel="stylesheet" href="/style.css">
        <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/cs
s/font-awesome.min.css">
    </head>
    <body>

        <div class="chat__sidebar">
                <h4>Active Users:-</h4>
                <span id="counter"></span>
                <ul class="feedback" id="users" ></ul>
        </div>
        <section class="chat_portion">
            <div class="logo">
                <img height="30" src="/JU.png"
alt=""><h1>JUTalkie</h1>

                <!-- <span id="counter"></span> -->
            </div>

            <div class="msg_area" id="historyMsg">
            </div>

            <div>
                <textarea id="textarea" placeholder="Tell your
friends about....">
                </textarea>
                <!-- <input id="sendBtn" type="button" value=""><i
class="fa fa-paper-plane"></i> -->
                <button id="sendBtn" type="button">
                    <i class="fa fa-paper-plane"></i>
                </button>
                <label for="sendImage" class="imageLable">
                    <input class="uploadFile" type="button"
value="image"  />
                    <!-- <i class="fa fa-paperclip"></i> -->
                    <input id="sendImage" type="file"
value="image"  />

                </label>
            </div>
        </section>

        <script
src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
        <script
src="https://cdnjs.cloudflare.com/ajax/libs/mustache.js/3.0.1/must
ache.min.js"></script>
        <script src="/socket.io/socket.io.js"></script>
        <script src="/client.js"></script>

    </body>
</html>
```

Client Chat Script

It communicates with the server by sending messages to the server and listening for notifications.

```
const socket = io()

let user_name ;
let textarea = document.querySelector('#textarea')
let messageArea = document.querySelector('.msg_area')
// let userArea = document.querySelector('#users')
var $users = $('#users')
let sendImage = document.querySelector('#sendImage')
let sendBtn = document.querySelector('#sendBtn')
//Enter user name
do
{
    user_name = prompt("Enter your username");
}while(!user_name);

appendMessage('You joined the JUTalkies....Share your idea!');
socket.emit('new-user', user_name); //send to the server to
broadcast your name
// addToUsersBox(user_name);
document.title = 'JUTalkie | ' + user_name ; //title of the
website

//After broadcasting, those who are connected will be able to see
socket.on('user-connected', user_name => {
    appendMessage(`${user_name} connected... Say Hi :)`) //other
users will be able to see this

});

//if user left, other users will know whether that user left from
the server
socket.on('user-disconnected', user_name => {
    appendMessage(`${user_name} left the fun...`) //only for those
who are connected to see this message
});
```

```
socket.on('usernames',data =>
{
    // let divMain = document.createElement("li");
    var listName="";

    for(i=0;i<data.length;i++)
    {
        listName += '<li class="list-group-item">'+
data[i]+'</li>' ;
    }
    // document.getElementById("users").appendChild(listName);
    $users.html(listName);
});

//for enter key to send message
textarea.addEventListener('keyup', (e) =>
{
    if(e.key == "Enter") //Enter value
        sndMsg(e.target.value); //message send function
});

//send button click function to send textarea message
sendBtn.addEventListener('click' ,() =>
{
    var messageInput = document.getElementById('textarea'),
    msg = messageInput.value;

    messageInput.value = '';
    messageInput.focus();
    if (msg.trim().length != 0) //if message has some value, then
send message
```

```
    {
        sndMsg(msg);
    };
}, false );

//send message
function sndMsg(message)
{
    let msg = {
        user: user_name,
        message: message.trim()
    }
    appendMsg(msg, 'out'); //outgoing text message
    textarea.value = ""; //after sending, textarea becomes
null/cleared
    autoScrollDown(); //scrolling will be done automatically

    socket.emit("message", msg) //send message to the server to
broadcast
                                //so that other users can see the
message
}

//message in the message area(Chat area)
function appendMsg(msg, type)
{
    let divMain = document.createElement("div");
    let className = type ;
    divMain.classList.add(className, "msg");

    let sending =
    ` <h4>${msg.user}</h4>
```

```
      <p>${msg.message}</p>`

    divMain.innerHTML = sending;
    messageArea.appendChild(divMain);
}

//send image function onchange
sendImage.addEventListener('change', () =>
{

    var filesSelected =
document.getElementById("sendImage").files;
    if (filesSelected.length > 0) {
      var fileToLoad = filesSelected[0];

      var fileReader = new FileReader();

      fileReader.onload = function(fileLoadedEvent) {

        var srcData = fileLoadedEvent.target.result; //  data:
base64

        var newImage = document.createElement('img');
        newImage.src = srcData;

        let imgData = {
            user: user_name,
            message: srcData
        }

        displayImage( `${user_name}` , srcData); //display image
in the message area
```

```
    socket.emit('image',imgData); //send image to the server
to broadcast it so that users can see image
   }
   fileReader.readAsDataURL(fileToLoad);
  }
 });

 //display image in the message area
 function displayImage(user , srcData ) {
  var newImage = document.createElement('img');
    newImage.src = srcData;

    // document.getElementById("historyMsg").innerHTML = user
+ newImage.outerHTML;

    messageArea.append(newImage);
    appendMessage(`${user} send image...`);
    autoScrollDown();
    // alert("Converted Base64 version is " +
document.getElementById("historyMsg").innerHTML);

};

//incoming text message
socket.on("message", (msg) =>
{
  appendMsg(msg, 'in');
  autoScrollDown();
});

//incoming image message
```

```
socket.on("image", (imgData) =>
{
    displayImage(imgData.user, imgData.message);
    // autoScrollDown();
});

//append message
function appendMessage(message)
{
    const messageElement = document.createElement('div');
    let className = "announce" ;
    messageElement.classList.add(className, "msg");

    messageElement.innerText = message;
    messageArea.append(messageElement);
}

//auto scrolling down
function autoScrollDown()
{
    messageArea.scrollTop = messageArea.scrollHeight ;
}
```

Socket.io plays a very important role for dealing with the message to send or receive between client and server.

- **socket.emit(‘message’, [callback])** – Used to send messages to the server.
- **socket.on(‘message’, callback)** – Used to receive messages from the server. callback is invoked on reception.

Salient Features:-

The 5 basic features are :-

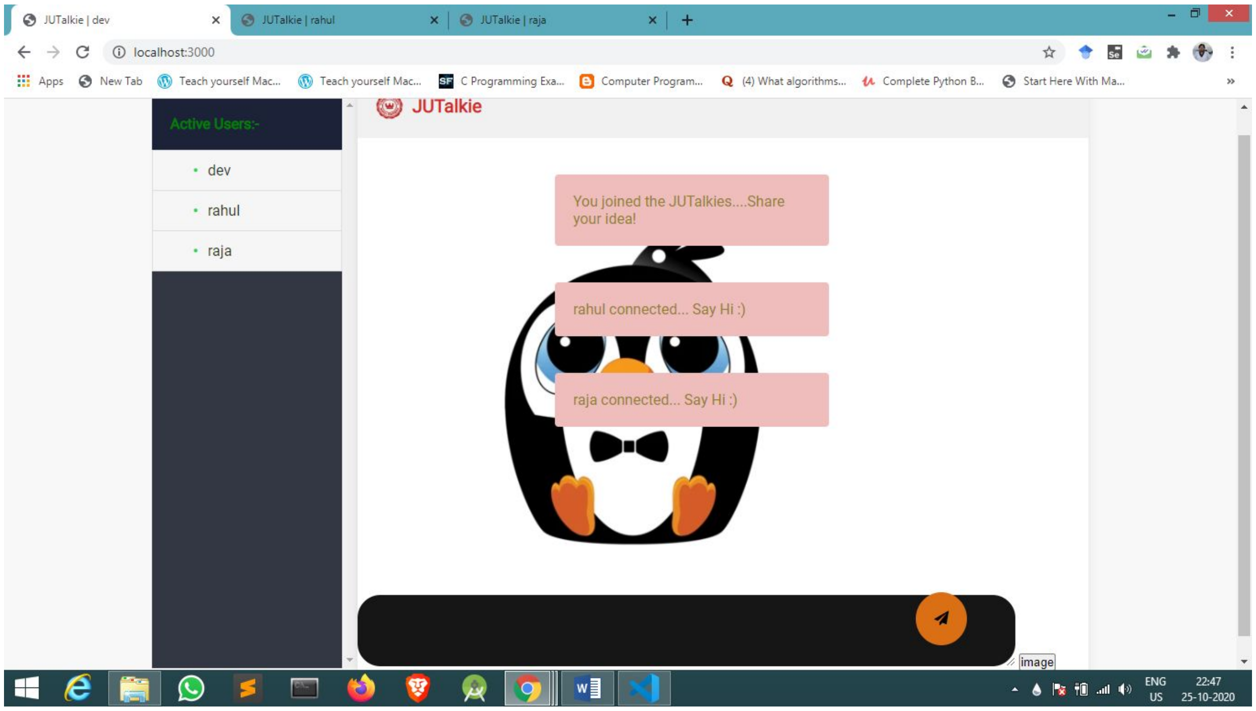
1. Multiple users can chat together in real-time.
2. Users can join and leave if they wish.
3. All users will get notified when the user joins or leaves the current chat.
4. The list of active users are provided.
5. Concept of unicast, multicast and broadcast is provided.
6. The user can send both text as well as images.
7. The user can send the voice message too. (Not including part of question, but done for myself for learning)

Input/Output:-

The following images contain the output of the proposed approach.

- Three user connections are done.

●



JUTalkie

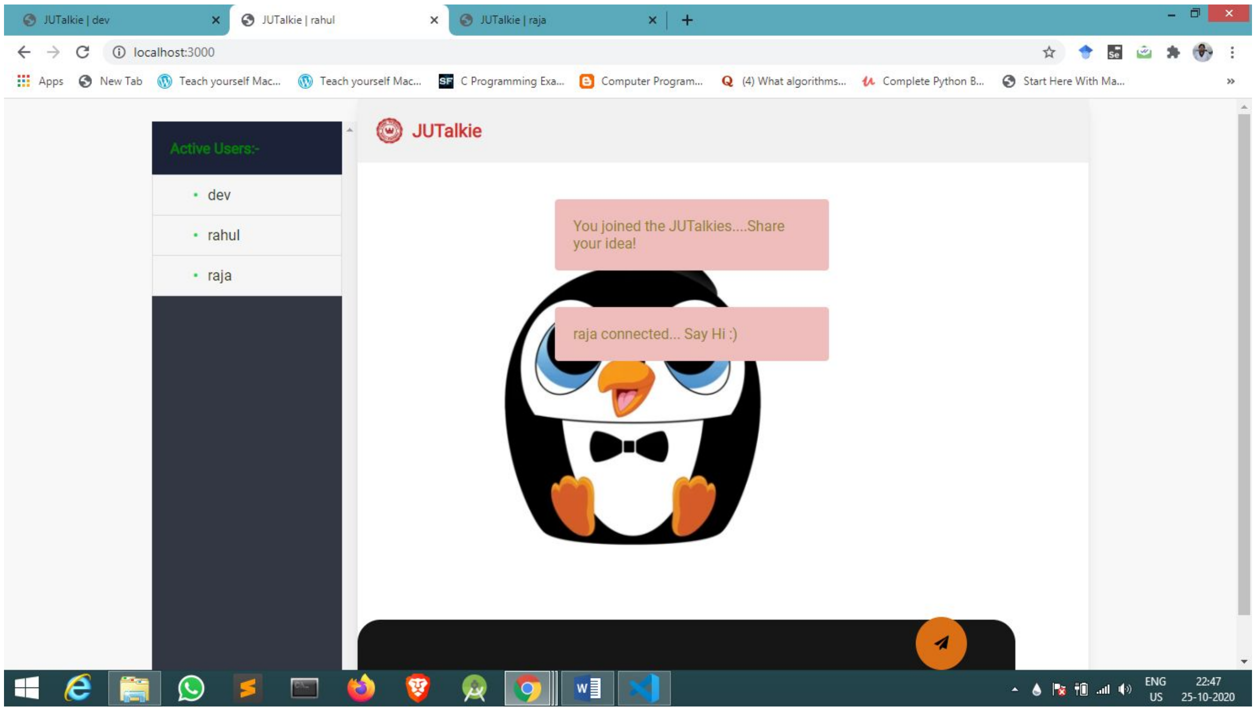
Active Users:-

- dev
- rahul
- raja

You joined the JUTalkies....Share your idea!

rahul connected... Say Hi :)

raja connected... Say Hi :)



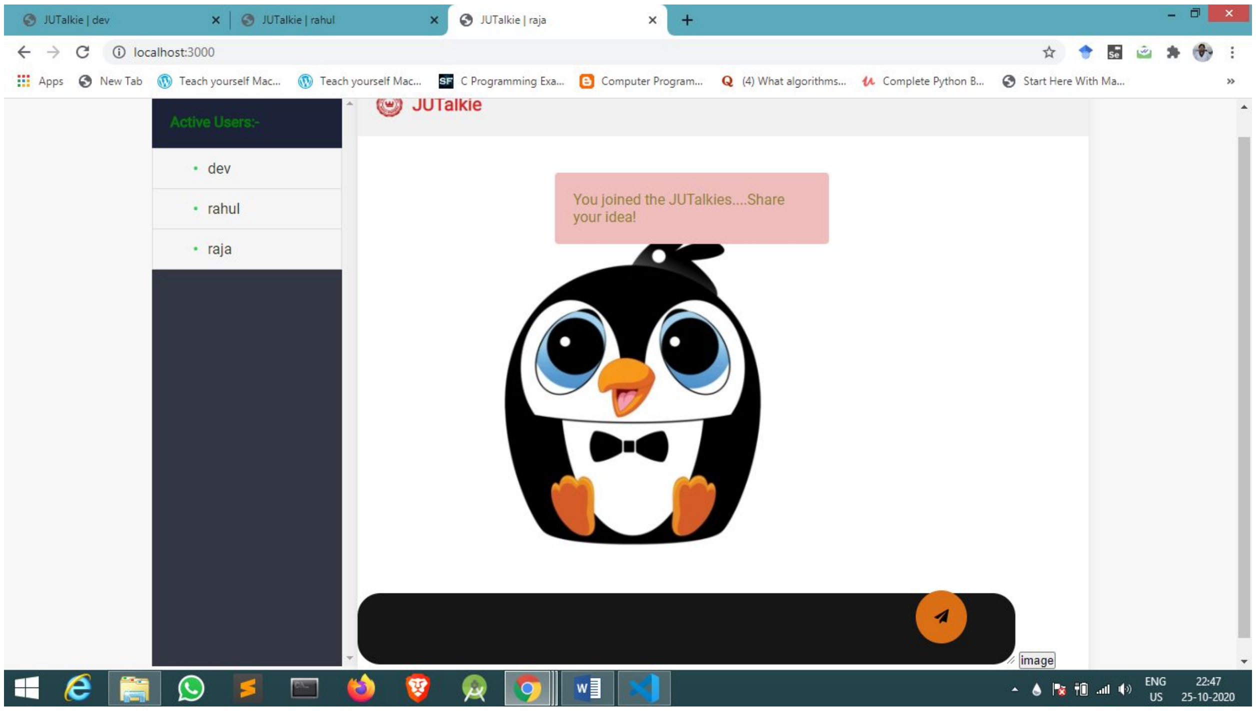
JUTalkie

Active Users:-

- dev
- rahul
- raja

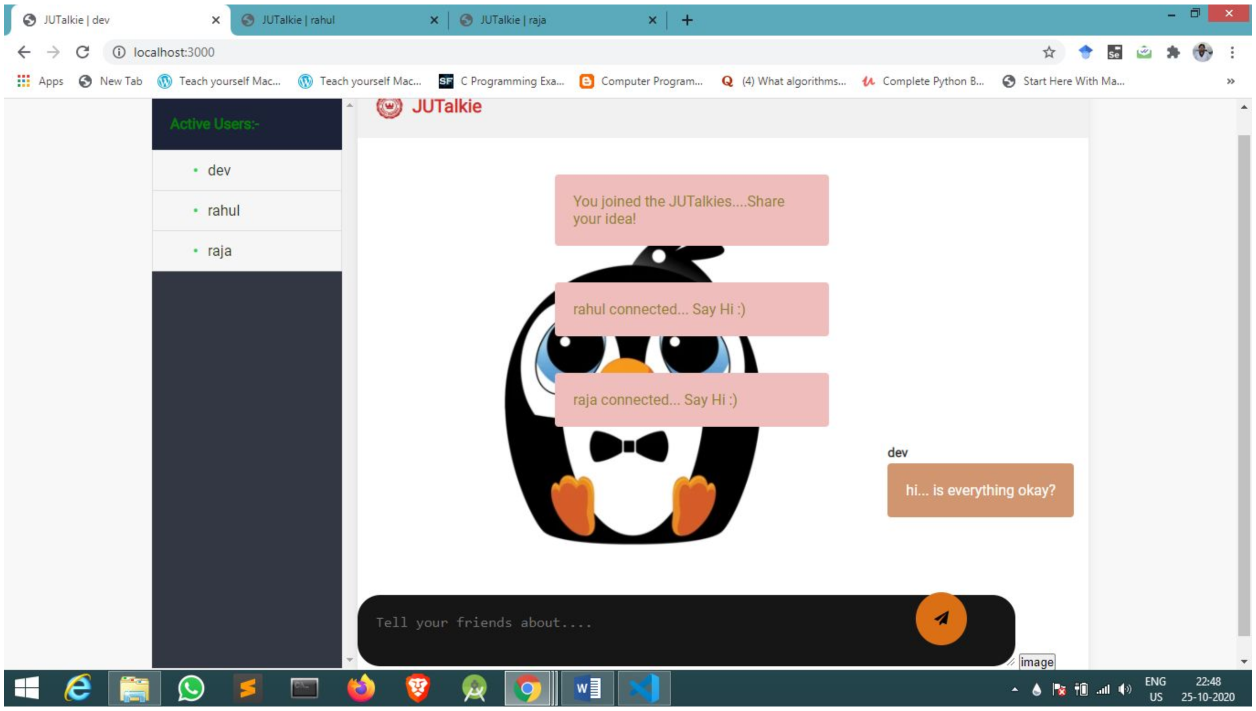
You joined the JUTalkies....Share your idea!

raja connected... Say Hi :)



You joined the JUTalkies....Share your idea!

- Sending text message



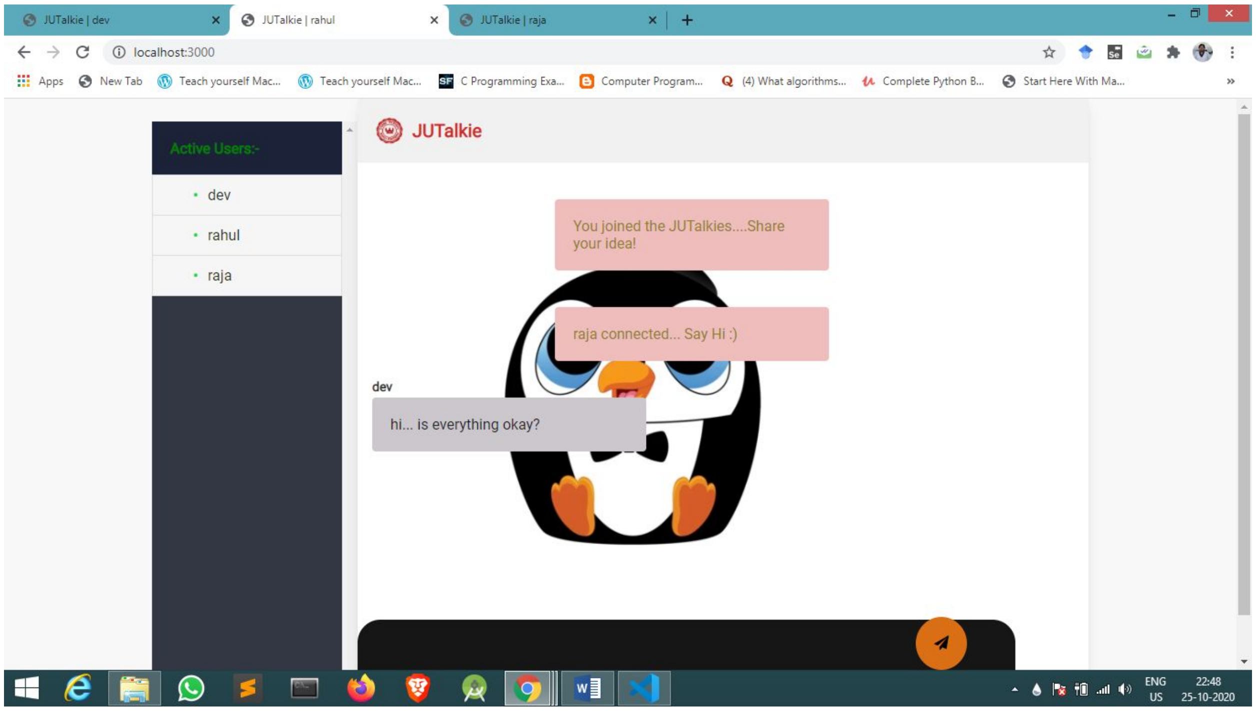
You joined the JUTalkies....Share your idea!

rahul connected... Say Hi :)

raja connected... Say Hi :)

dev

hi... is everything okay?



JUTalkie | dev
JUTalkie | rahul
JUTalkie | raja

localhost:3000

Active Users:-

- dev
- rahul
- raja

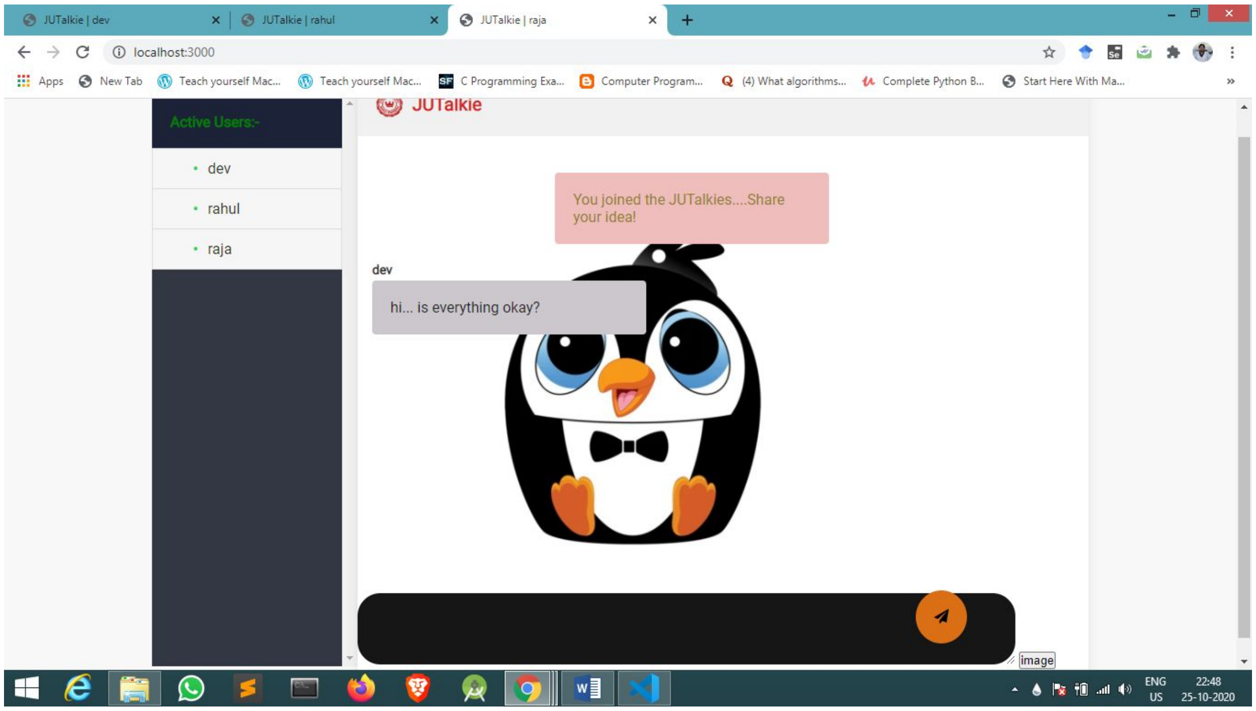
JUTalkie

You joined the JUTalkies....Share your idea!

raja connected... Say Hi :)

dev

hi... is everything okay?



JUTalkie | dev
JUTalkie | rahul
JUTalkie | raja

localhost:3000

Active Users:-

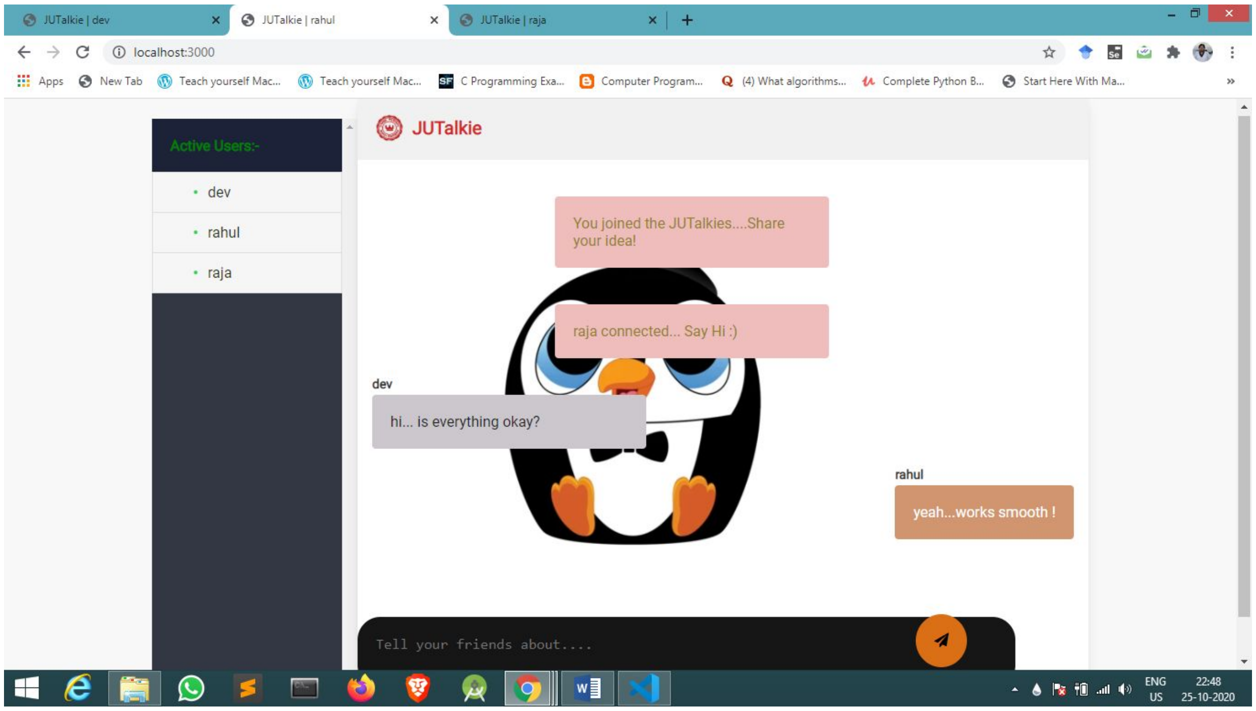
- dev
- rahul
- raja

JUTalkie

You joined the JUTalkies....Share your idea!

dev

hi... is everything okay?



JUTalkie | dev
JUTalkie | rahul
JUTalkie | raja
localhost:3000

Active Users:-

- dev
- rahul
- raja

JUTalkie

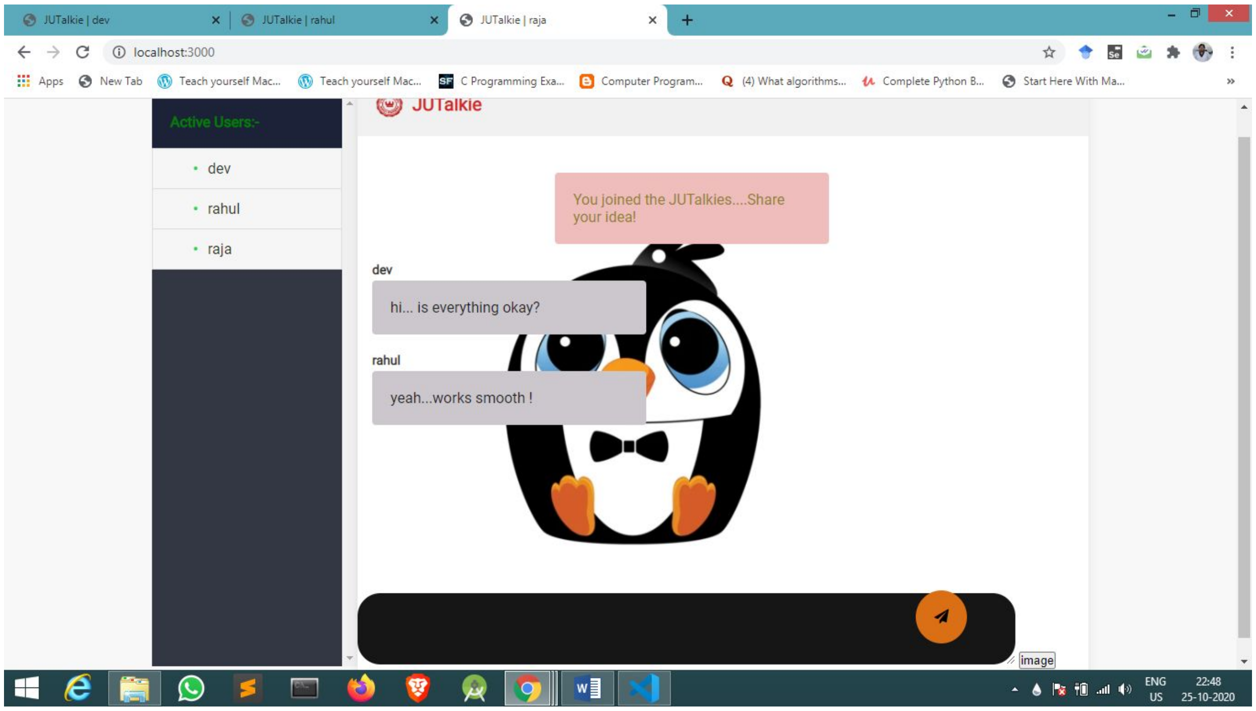
You joined the JUTalkies....Share your idea!

raja connected... Say Hi :)

dev
hi... is everything okay?

rahul
yeah...works smooth !

Tell your friends about....



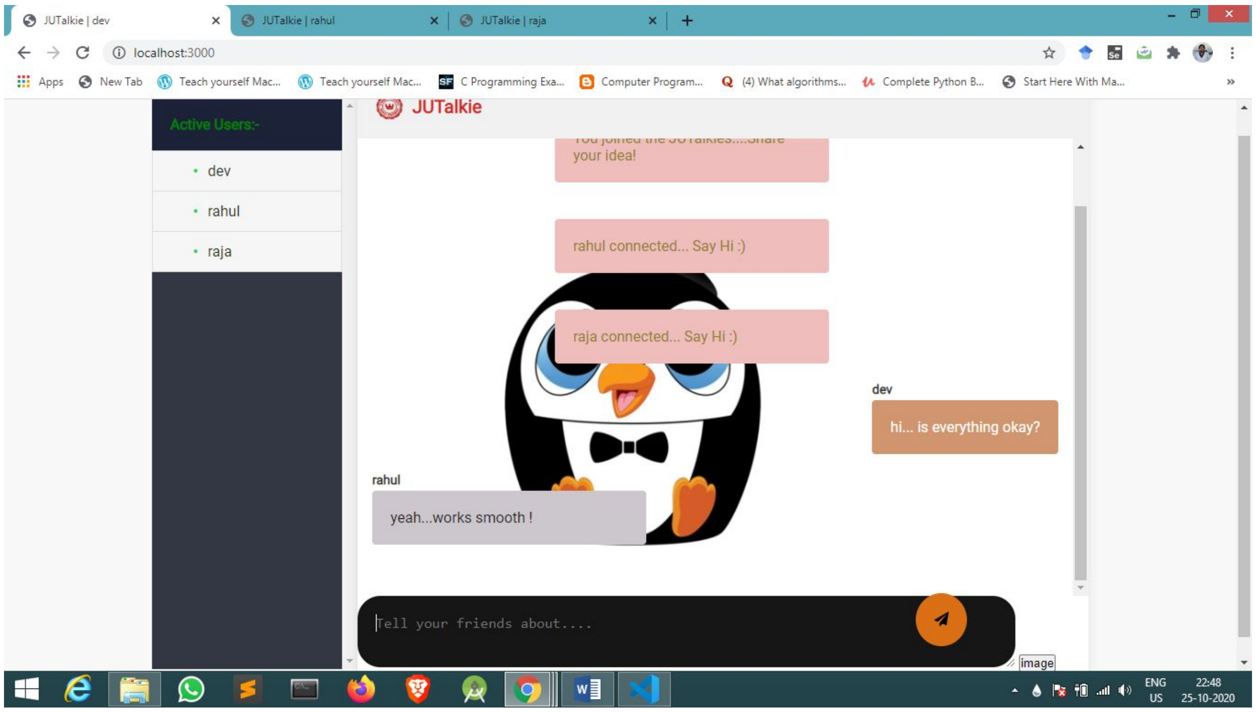
You joined the JUTalkies....Share your idea!

dev

hi... is everything okay?

rahul

yeah...works smooth !



your idea!

rahul connected... Say Hi :)

raja connected... Say Hi :)

dev

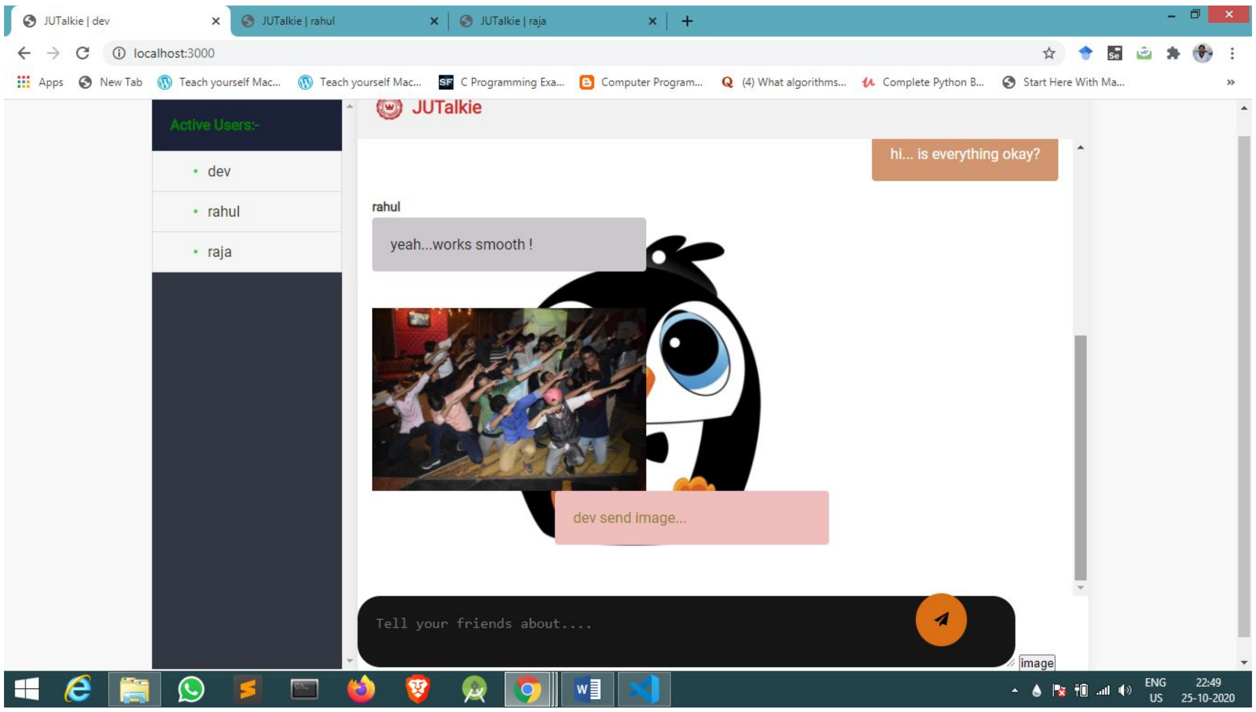
hi... is everything okay?

rahul

yeah...works smooth !

Tell your friends about....

- Sending image message



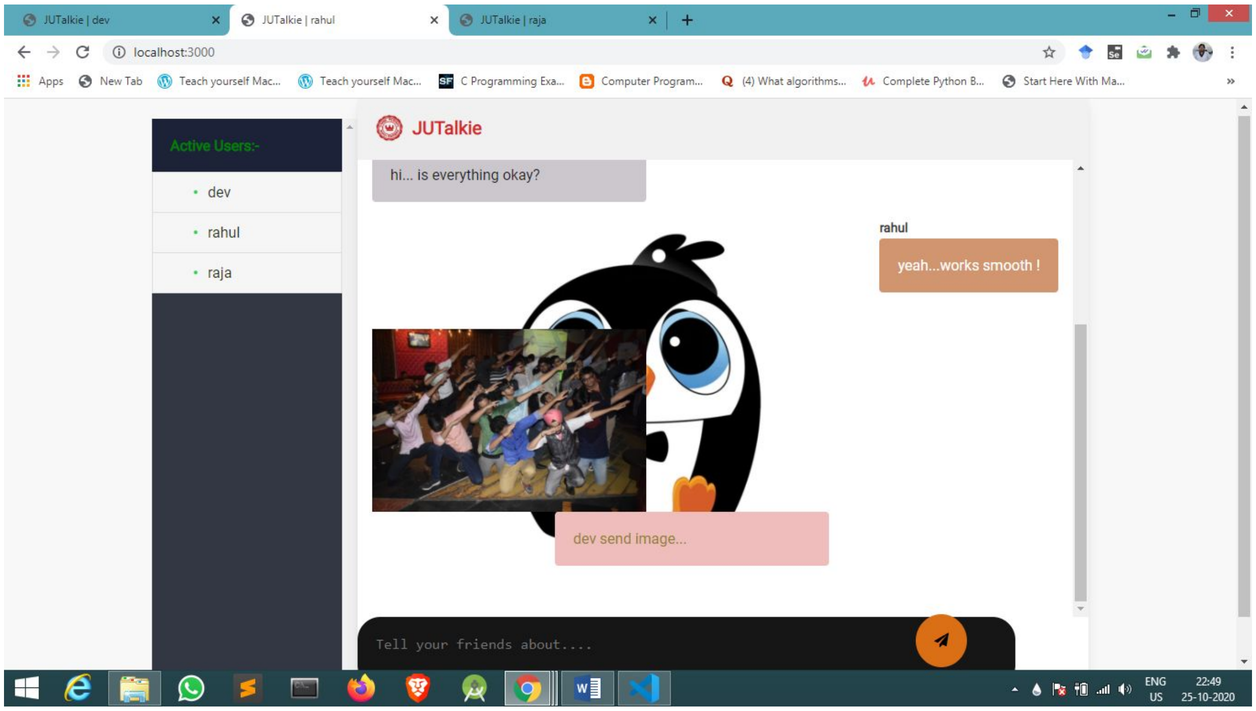
hi... is everything okay?

rahul

yeah...works smooth !

dev send image...

Tell your friends about....

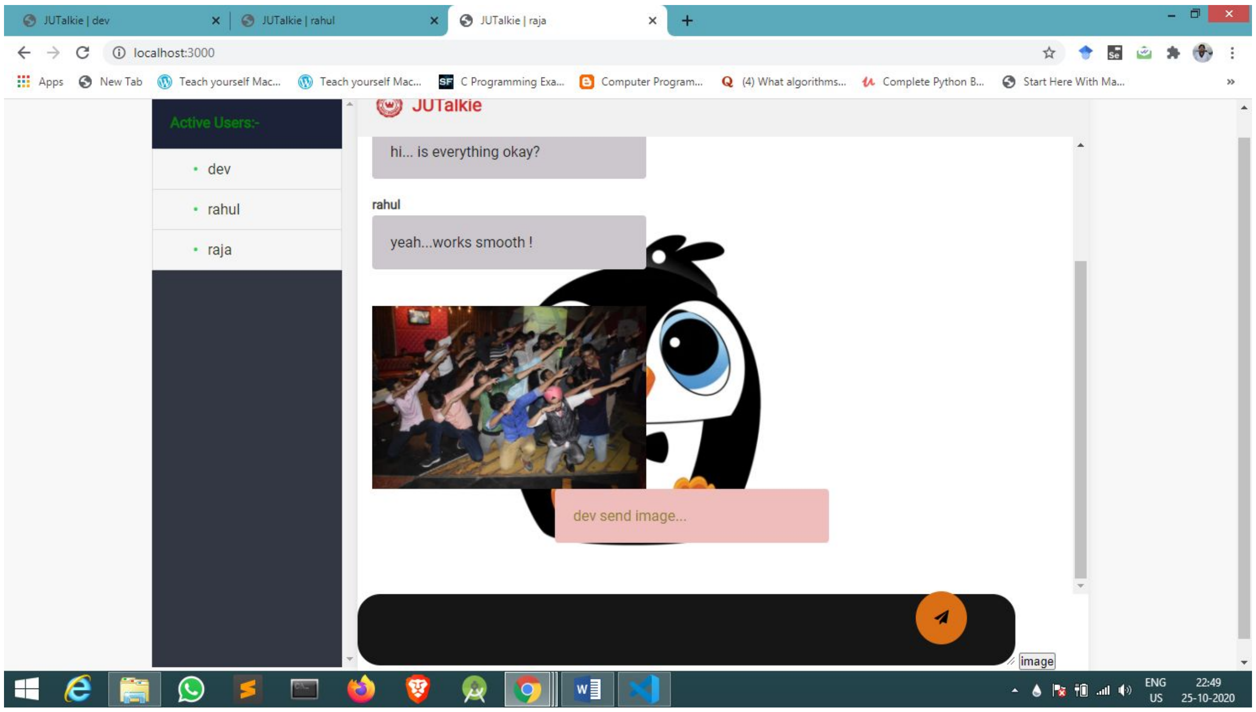


hi... is everything okay?

rahul

yeah...works smooth !

dev send image...



hi... is everything okay?

rahul

yeah...works smooth !

dev send image...