LetsTalk -Online Chatting

Project Submitted in Partial Fulfillment of the Requirements for the

Degree of Bachelor of

Technology in the field of Computer Science and Engineering

BY

SHUBHANJAN BARAI (123180703083)

Department of Computer Science and Engineering
JIS College of Engineering

Block-A, Phase-III, Kalyani, Nadia, Pin-741235
West Bengal, India
May, 2021

List of Figures

Figure Description Page no

CONTENTS

ABSTRACT

Teleconferencing or Chatting, is a method of using technology to bring people and ideas together despite of the geographical barriers. The technology has been available for years but the acceptance it was quit recent. Our project is an example of a chat server. It is made up of 2 applications the client application, which runs on the users Android Device and server application, which runs on any Android Device on the network. To start chatting client should get connected to server where they can do private and group chat security measures were taken taken during the last one.

INTRODUCTION

Messaging apps now have more global users than traditional social networks—which means they will play an increasingly important role in the distribution of digital journalism in the future. Drawing upon our interviews and case studies, we identify a number of opportunities and challenges for organizations using—or hoping to use—messaging apps for news. We argue that to devise a successful messaging app strategy, publishers must understand regional strongholds, user demographics,

and popular features of each app. As happened after the early days of social media, before which a proliferation of services (some with regional strengths) led to intense competition for user attention, we expect to see some eventual consolidation among chat apps. Elsewhere, we conclude that issues around information, privacy, personal security, and mobile data penetration will unfold in different ways around the world; apps like Telegram and FireChat are among those at the forefront of addressing and solving these problems. In developing editorial strategies for some of these wide-ranging messaging platforms, news organizations are not just helping to future-proof themselves, they are also venturing into online spaces that could enable them to reach hundreds of millions of (often young) people with whom they have never engaged before.

The emergence of computer network and telecommunication technologies bears the same objective: to allow people to communicate. Chatting is a method of using technology to bring people and ideas together despite geographical barriers. The technology has been available for years but the acceptance of it was quite recent. Our project is an example of a chat server. It is made up of an application which runs on any pc connected to the network. To start chatting our client should get connected to server where they can do group chatting.

TECHNOLOGIES USED

Frontend Development:

The part of a website that the user interacts with directly is termed the front end. It is also referred to as the ‘client side’ of the application. It includes everything that users experience directly: text colors and styles, images, graphs and tables, buttons, colors, and navigation menu. HTML, CSS, and JavaScript are the languages used for Front End development. The structure, design, behavior, and content of everything seen on browser screens when websites, web applications, or mobile apps are opened up, is implemented by front End developers. Responsiveness and performance are two main objectives of the Front End. The developer must ensure that the site is responsive i.e., it appears correctly on devices of all sizes no part of the website should behave abnormally irrespective of the size of the screen.

We have used HTML, CSS, and JavaScript. Below are the reason for using these technologies.

Why HTML?

Hypertext Markup Language, or HTML, is a programming language used to describe the structure of information on a webpage. A web page can contain headings, paragraphs, images, videos, and many

other types of data. Front-end developers use the HTML element to specify what kind of information each item on a webpage contains. A look under the hood of any website would reveal a basic HTML code page, written with an HTML structure editor, providing structure for all the page's components, including its header element, footer element, main content, and other inline elements.

Why CSS ?

CSS(Cascading Style Sheet) is the language for describing the presentation of Web pages, including colors, layout, and fonts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based markup language. The separation of HTML from CSS, known as external CSS, makes it easier to maintain sites, share style sheets across pages, and tailor pages to different environments.

Why JavaScript?

The importance of JavaScript as a web technology can be determined from the fact that it is currently used by 94.5% of all websites. As a client-side programming language, JavaScript helps web developers to

make web pages dynamic and interactive by implementing custom client-side scripts. One can even combine JavaScript, HTML5 and CSS3 to create web pages that look good across browsers, platforms, and devices.

There are also a number of reasons why each modern web developer must know JS like-

- Implement Client-Side Scripts
- Write Server-Side Code
- Responsive Web Design
- Varying Libraries and Framework.

Backend Development:

Backend is the server-side of the website. It stores and arranges data, and also makes sure everything on the client-side of the website works fine. It is the part of the website that you cannot see and interact with. It is the portion of software that does not come in direct contact with the users. The parts and characteristics developed by backend designers are indirectly accessed by users through a front-end application. Activities, like writing APIs, creating libraries, and working with system components without user interfaces or even systems of scientific programming, are also included in the backend.

Why Nodejs?

Node (or more formally *Node.js*) is an open-source, cross platform runtime environment that allows developers to create all kinds of server-side tools and applications in JavaScript. The runtime is intended for use outside of a browser context (i.e., running directly on a computer or server OS). As such, the environment omits browser-specific JavaScript APIs and adds support for more traditional OS APIs including HTTP and file system libraries.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

For Database:

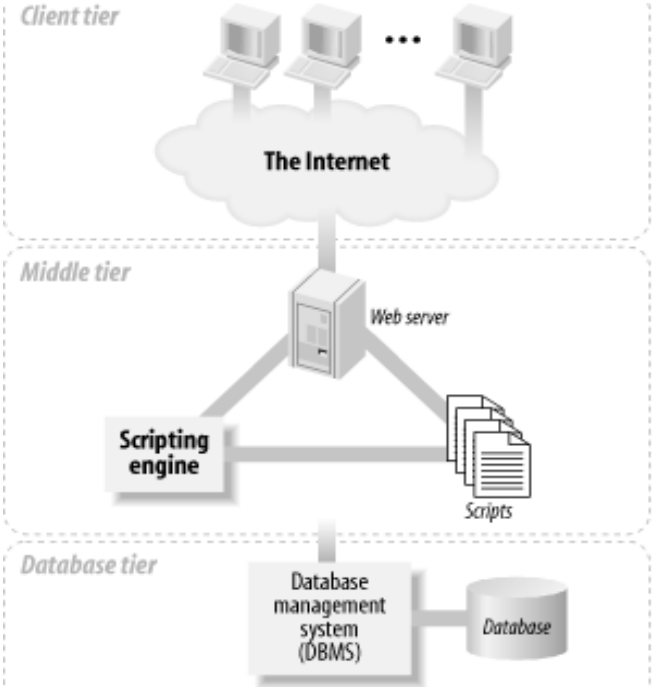

Figure 1: 3-Tier of Database

The three-tier architecture is conceptual. In practice, there are different implementations of web database applications that fit this architecture. The most common implementation has the web server (which includes the scripting engine that processes the scripts and carries out the actions they specify) and the database management system installed on one machine: it's the simplest to manage and secure, and it's our focus in this book. With this implementation on modern hardware, your applications can probably handle tens of thousands of requests every hour.

Why MongoDB?

In the simplest terms, MongoDB is a cross-platform document-oriented NoSQL database that uses JSON-like documents using dynamic schemas, called BSON documents, instead of following the conventional relational database (RDB) structure.

As a document database, MongoDB makes it easy for developers to store structured or unstructured data. It uses a JSON-like format to store documents. This format directly maps to native objects in most modern programming languages, making it a natural choice for developers, as they don't need to think about normalizing data. MongoDB can also handle high volume and can scale both vertically or horizontally to accommodate large data loads.

We will use MongoDB not for storing chats between the people, but MongoDB will be used to store the details of Users who are registering as first user and existing User.

What is Socket.IO?

Socket.IO is a library that enables low-latency**,** bidirectional and event-based communication between a client and a server.

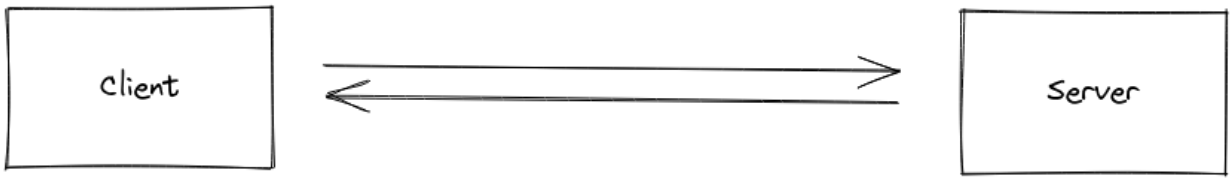

Figure 2: Explanation of Socket.IO

It is built on top of the WebSocket protocol and provides additional guarantees like fallback to HTTP long-polling or automatic reconnection. Socket.io has two parts: client-side and server-side. Both parts have an identical API.

- The server-side library runs in node.js.
- The client-side library runs in the browser.

Socket.IO is quite popular and companies like Trello, Yammer, Amazon, Zendesk, and several others use it to develop robust real-time applications. Socket.io is a very popular JavaScript framework on GitHub and is heavily dependent on the NPM plugin. It runs on every browser or platform and focuses equally on speed and reliability.

Real-time applications of the Socket.io

1. **Instant messengers:** Instant messengers are those applications in which you do not need to refresh your application and website to receive new messages, such as WhatsApp, Facebook Messenger, etc.

2. **Push Notifications:** Push notifications are those applications in which you will be notified when anyone tags you in a picture and story on Instagram and Facebook.
3. **Collaboration Applications:** Collaboration applications are those applications that allow updates to the same document at the same time by different users, such as Google Docs.
4. **Online Gaming:** Online gaming applications are those applications that help to establish bi-direction communication between multiple users. Examples of real-time online games are Pubg, Call of Duty, Among Us, Fortnite, etc.

Features of Socket.IO

1. **Reliability:** It relies on Engine.IO, which establishes a long-running polling connection before attempting to upgrade to better "testing" transports, such as WebSocket.
2. **Auto-reconnection support:** The disconnected client keeps trying to reconnect until the server is available again.
3. **Disconnection detection:** The heartbeat mechanism of the socket.io is engine.io. It lets both the server and the client know when no one else is responding.
4. **Binary support:** Any serializable data structures can be emitted, including:
5. ArrayBuffer and Blob in the browser

6. ArrayBuffer and Buffer in Node.js
7. **Multiplexing support:** Socket.IO allows you to create multiple namespaces to separate concerns within your application, which will act as separate communication channels but share the same underlying connection.

LITERATURE SURVEY

The online chat group is a small-scale multiuser social networking platform, in which users participate in the discussions and send and receive information. Online chat group service providers are concerned about the number of active members because more active members means more advertising revenues. For the group owners and members, efficiency of information acquisition is the concern. So, it is of great value to model these two indicators' impacting factors.

As we know the use of internet has increased greatly and internet has become one of the easiest and cheapest sources of communication there are many messaging and chatting applications coming up. There are already many applications available for communication. The oldest one we use is electronic mails. Other applications available are the various social websites, SMS, Mobile Chatting applications and much more.

we can see that the world is completely depended on the internet. We communicate with long distance friends through internet. We make use of various social media websites. We communicate with our friends and relatives through messages, online chats, calls, video chats and much more. Messaging and chatting with friends have become the easiest and cheapest way of communication. We can send images and audio, video clips on long distance within fraction of second. As the users of the internet are increasing day by day on a large scale the performance measure comes into pictures.

But to be successful, they need effective tools, resources and methodologies to provide the user with a high service level. Passing from LAN Client-Server(C/S) to Internet Web-based applications, your audience increases but your risks increase as well. So, now, application performances need greater attention. In the USA a survey has found that a user waits just 8 seconds for a page to download completely before leaving the site. In Italy, this limit may be higher, but anyway, providing high performance is a key factor in the success of the site.

The very first attempts at chatbots' implementation were rule-based. Rule-based models are usually easier to design and to implement, but are limited in terms of capabilities, since they have difficulties answering complex queries. Rule-based chatbots answer users' queries by looking

for patterns matches; hence, they are likely to produce inaccurate answers when they come across a sentence that does not contain any known pattern. Furthermore, manually encoding pattern matching rules can be difficult and time consuming. Furthermore, pattern matching rules are brittle, highly domain specific, and do not transfer well from one problem to the other.

In our quest to better understand network traffic dynamics, we examine Internet chat systems. Although chat as an application does not contribute huge amounts of traffic, chat systems are known to be habit-forming. This implies that catering to such users can be a promising way of attracting them, especially in low bandwidth environments such as wireless networks. Unfortunately, there is no common protocol base for chat systems. Rather there are a multitude of protocol variants whose specifications, with some exceptions, such as IRC and ICQ, are unavailable or ill defined. In addition, chat systems are often layered on top of other application protocols like HTTP. Therefore, there is no simple way of even identifying chat traffic.

METHODOLOGY

Methodology here is simply when the user enters the room he/she can talk over a topic and easily leave the room when done. Suppose a scenario that one student is not getting the answer from anywhere out of the crowd,

then he decides to talk to someone like any normal talk in a home so that his/her doubts get cleared and also, he/she can ask question which will crystal clear the doubt after talking.

This way our website helps in any students stuck with programming language.

IMPLEMENTATION

This way we are connecting the user to the chat application and giving the message accordingly to that user and other user present in the room.

```
io.on('connection', socket => {
    socket.on('joinRoom', ({ username, room }) => {
        const user = userJoin(socket.id, username, room);
        socket.join(user.room);
        socket.emit('message', formatMessage(botName, '<h2> Welcome to LetsTalk! </h2>'));
        //broadcast when user connects
```

```
        socket.broadcast.to(user.room).emit('message',
    formatMessage(botName, `<h3> ${user.username} has joined the
    Room </h3>`));

        //send users and room

        io.to(user.room).emit('roomUsers', {

            room: user.room,

            users: getRoomUser(user.room)

        });

    });
```

How the random messages are printing?

```
    socket.on('chatMessage', msg => {

        const user = getCurrentUser(socket.id);

        io.to(user.room).emit('message',
    formatMessage(user.username, msg));

    });
```

When a user disconnects or leave the room the following code is used-

```
socket.on('disconnect', () => {
        const user = userLeave(socket.id);
        if (user) {
            io.to(user.room).emit('message',   formatMessage(botName,
`<h3>${user.username} has left the Room </h3>`));
            //send users and room
            io.to(user.room).emit('roomUsers', {
                room: user.room,
                users: getRoomUser(user.room)
            });
        }
    });
});
```

Utilities Folder

Utilities Folder consists of 2 javascript files.

message.js – It is used to manage the message send by the user.

```
const moment = require("moment");
function formatMessage(username, text) {
    return {
        username,
        text,
        time: moment().format('h:mm a')
    };
}
module.exports = formatMessage;
```

user.js – It is used to manage the user's entering , staying or leaving the group.

```
const users = [];
//Join user to chat
function userJoin(id, username, room) {
    const user = { id, username, room };
    users.push(user);
```

```
    return user;
}
//get user
function getCurrentUser(id) {
    return users.find(user => user.id === id);
}
//user leaves chat
function userLeave(id) {
    const index = users.findIndex(user => user.id === id);
    if(index !== -1) {
        return users.splice(index, 1)[0]; }
}
//get users room
function getRoomUser(room) {
    return users.filter(user => user.room === room);
}
```

```
module.exports = {
    userJoin,
    getCurrentUser,
    userLeave,
    getRoomUser
};
```

Frontend part :

main.js file –

```
const chatForm = document.getElementById('chat-form');
const chatMessages = document.querySelector('.chat-messages');
const roomName = document.getElementById('room-name');
const userList = document.getElementById('users');
const {username, room} = Qs.parse(location.search, {
    ignoreQueryPrefix: true
});
```

```
const socket = io();

socket.emit('joinRoom', { username, room });

//get room users

socket.on('roomUsers', ({ room, users }) => {

    outputRoomName(room);

    outputUsers(users);});

socket.on('message', message => {

    // console.log(message);

    outputMessage(message);

    chatMessages.scrollTop = chatMessages.scrollHeight;});

chatForm.addEventListener('submit', e => {

    e.preventDefault();

    const msg = e.target.elements.msg.value;

    socket.emit('chatMessage', msg);

e.target.elements.msg.value = '';
```

```
e.target.elements.msg.focus();

});

function outputMessage(message) {

    const div = document.createElement('div');

    div.classList.add('message');

    div.innerHTML = `<p class="meta">${message.username}
<span>${message.time}</span></p>

    <p class="text">

        ${message.text}

    </p>`;

    document.querySelector('.chat-messages').appendChild(div);

}

//add room name to DOM

function outputRoomName(room) {

    roomName.innerText = room;

}
```

```
//add users to DOM
function outputUsers(users) {
    userList.innerHTML = `
        ${users.map(user => `<li>${user.username}</li>`).join('')}
    `;
}
//Prompt the user before leave chat room
document.getElementById('leave-btn').addEventListener('click', () => {
    const leaveRoom = confirm('Are you sure you want to leave the chatroom?');
    if (leaveRoom) {
      window.location = '../index.html';
    } else {
    }
  });
```

Result and Discussion

This the Home page or First page shown to the user after user clicks on the link. There is basically a normal form which consists of name and the room which that particular user wants to join.

The rooms are C, Java, Python and C++. User can get their doubt clear by joining the respective room in which they are getting error or any suggestion or any kind help.

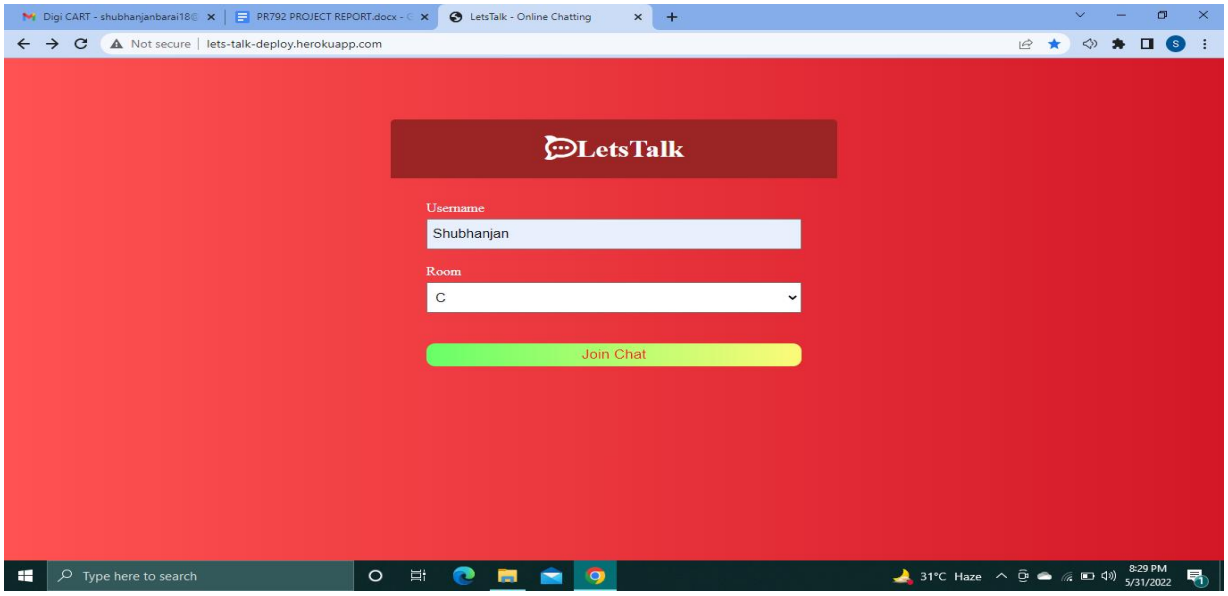

Figure 3: Home Page

Now as you can see one user is already in the room and this is the preview when someone enters the room. There is a "LetsTalk Assistant" to welcome or notify that user. As shown in the Figure 4.

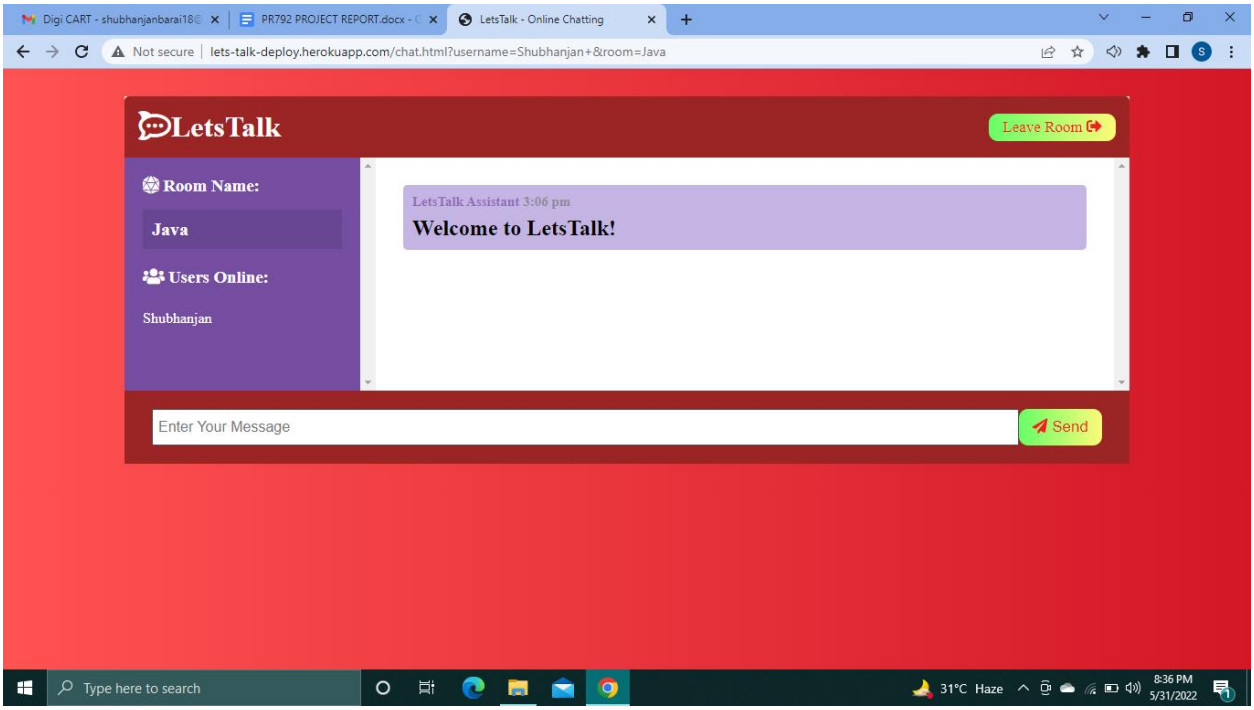

Figure 4 : Notification from LetsTalk Assistant after entering room

When the first user is already present, the second user enters the room for which the first user gets an alert in bold about the new user entrance into the website. It shows "{That User} has joined the room". Now they can continue their discussion on a topic.

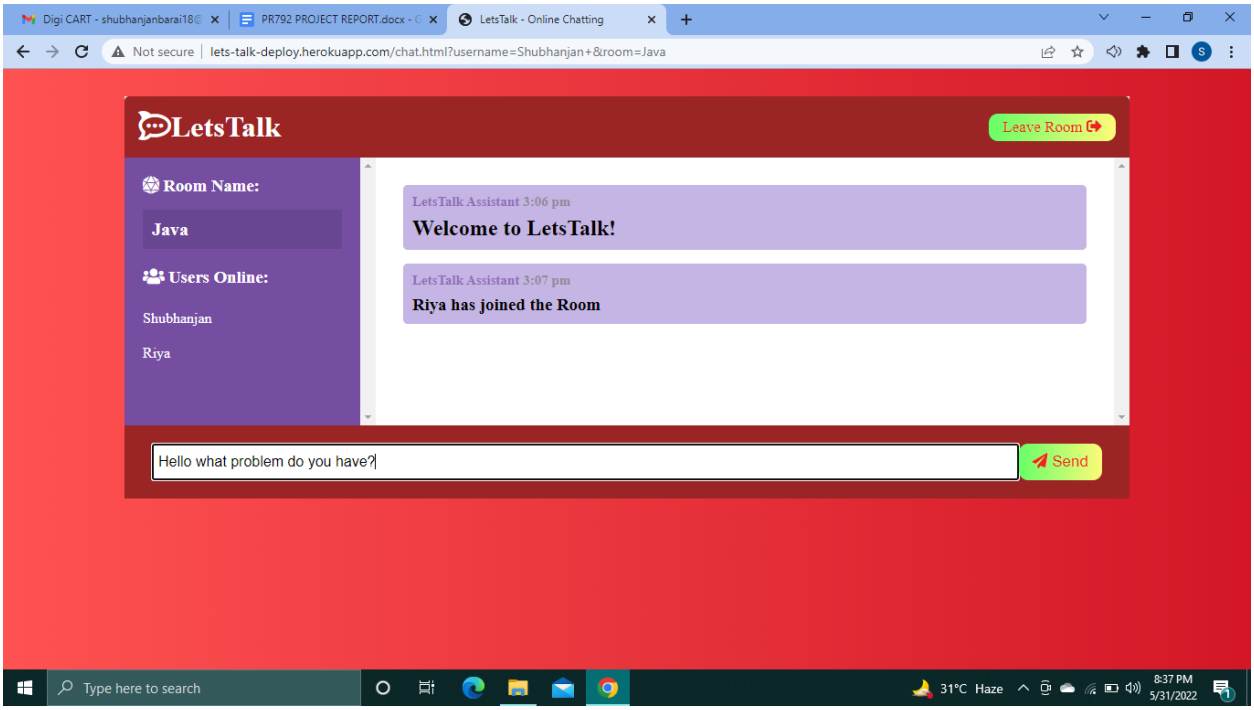

Figure 5: When Second User enters into the room

Now as you can see in figure 6, how easy it is to ask for a doubt from the person online, he/she can easily answer the query, and leave the room.

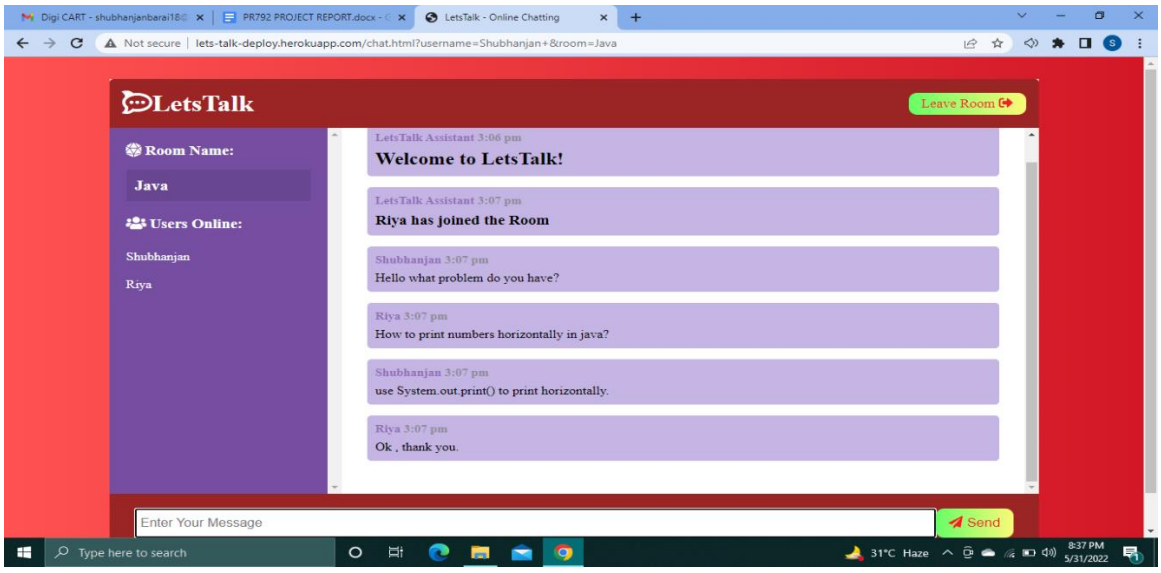

Figure 6: Doubt Clearing

You can also check the number of Users online in the left side of the Box or active members in that particular room. It's will ask confirmation on exit. Just to check whether the "Leave Room" button is pressed by mistakenly. You can cancel if pressed by mistakenly, or you can choose "Confirm" if you are sure you want to leave.

Whenever any user leaves the room, it will again show an alert in bold that "{any user} left the room". Also, the User gets removed from the "Users Online" section on the left pane. Just to let other users know about the active members in that room.

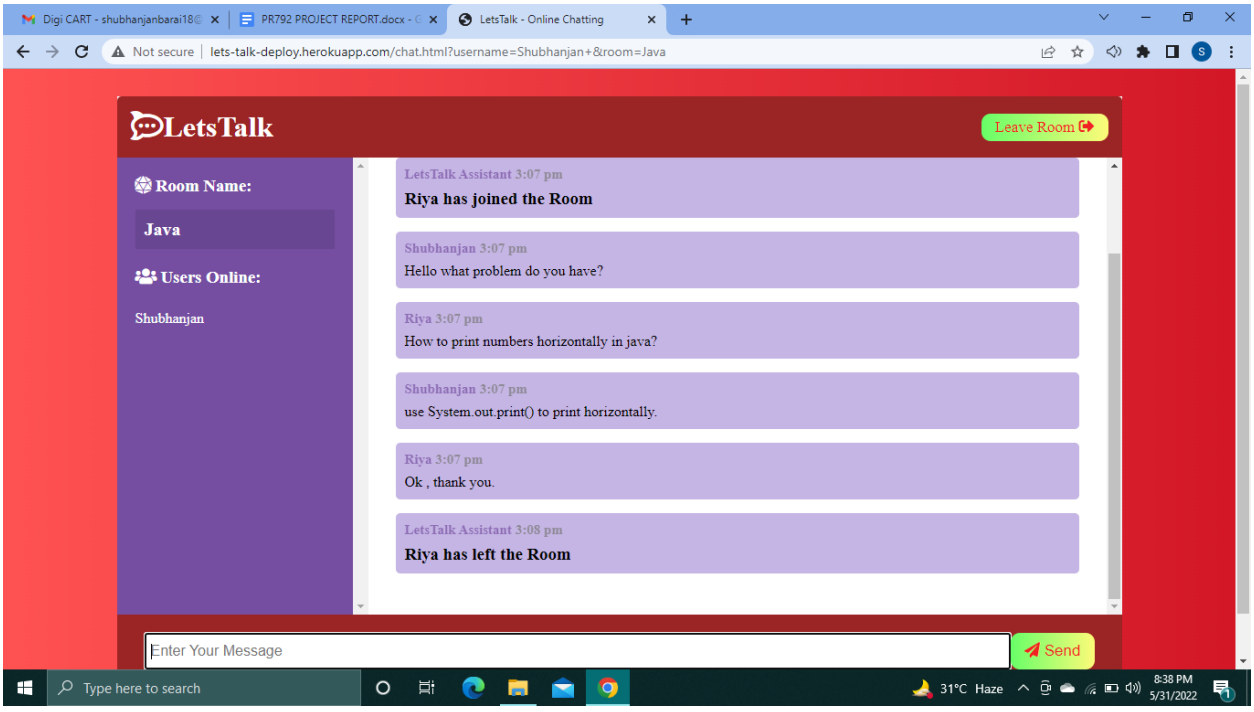

Figure 7: When second person left the room it gets notified

CONCLUSION

The main objective of the project is to develop a Secure Chat Application. I had taken a wide range of literature review in order to achieve all the tasks, where I came to know about some of the products that are existing in the market. I made a detailed research in that path to cover the loopholes that existing systems are facing and to eradicate them in our application. In the process of research I came to know about the latest technologies and different algorithms.

I analyzed various encryption algorithms (DES, AES, IDEA…), Integrity algorithms (MD5, SHA), key-exchange algorithms, authentication and I had implemented those functionalities in my application. I had done a detailed research on Certificate Authority and key tools for the generation of certificates.

FUTURE WORK

With the knowledge we have gained by developing this application, we are confident that in the future we can make the application more effectively by adding this services.

- Extending this application by providing Authorisation service (two ways authentication factor). We will introduce two ways

authentication factor so that only that user can enter into the room by using their credentials.

- Creating Database and maintaining users.
- Increasing the effectiveness of the application by providing Voice Chat.
- Extending it to Web Support.

REFERENCES

1. https://nodejs.org/en/docs/
2. https://devdocs.io/css/
3. https://www.w3schools.com/jsrEF/default.asp
4. https://socket.io/docs/v4/
5. https://stackoverflow.com/