

# **REAL TIME CHAT APPLICATION**

## **MINOR PROJECT REPORT**

*Submitted in partial fulfilment of the requirements for V Semester of*

*the degree of*

## **BACHELOR OF TECHNOLOGY**

*in*

## **INFORMATION TECHNOLOGY**

*By*

**JYOTI KUMARI**

**ROLL No : 16219**



**DEPARTMENT OF INFTORMATION TECHNOLOGY**

**DRONACHARYA GROUP OF INSTITUTION, GREATER NOIDA  
(AFFILIATED BY DR. APJ Abdul Kalam Technical University)**

## **CANDIDATES' DECLARATION**

It is hereby certified that the work which is being presented in the B. Tech Minor Project Report entitled "**REAL TIME CHAT APPLICATION**" in partial fulfilment of the V Semester Practical exam of **Bachelor of Technology** and submitted in the **Department of INFORMATION TECHNOLOGY** of **Dronacharya Group of Institute of Technology & Management, New Delhi (Affiliated By Dr. APJ Abdul Kalam Technical University)** is an authentic record of our own work carried out during a period from **September 2022 to February 2023**

## **ABSTRACT**

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 was quite recent. Our project is an example of a chat server. It is made up of two applications - the client application, which runs on the user's web browser and server application, runs on any hosting servers on the network. To start chatting client should get connected to server where they can do private and group chat. Security measures were taken during the last one.

## **ACKNOWLEDGEMENT**

We express our deep gratitude to Dr. Rajat Kumar , Project Coordinator Department of Information Technology for her valuable guidance and suggestion throughout my project work.

We would like to extend our sincere thanks to HOD of IT Department ( Dr. Bipin Pandey) for his time to time suggestions to complete our project work. We are also thankful to for providing me the facilities to carry out my project work.

We would also like to thank our families and friends, who helped us in every possible way.

**Jyoti Kumari**  
**Roll No.: 16219**

## **TABLE OF CONTENT**

<b>CANDIDATES' DECLARATION .....</b>	<b>ii</b>
<b>ABSTRACT.....</b>	<b>iii</b>
<b>ACKNOWLEDGEMENT.....</b>	<b>iv</b>
<b>LIST OF FIGURES .....</b>	<b>vii</b>
<b>LIST OF TABLES .....</b>	<b>viii</b>
<b>CHAPTER 1: INTRODUCTION.....</b>	<b>8</b>
1.0 Introduction.....	8
1.1 Problem Statement.....	8
1.2 Innovative Ideas of Project .....	9
1.3 Project Objective.....	9
1.4 Scope of The Project.....	9
1.5 What is Node.js? .....	11
1.5.1 Features of Node.js .....	11
1.6 What is Socket.io? .....	11
1.6.1 Features of Node.js .....	11
 <b>CHAPTER 2: PRODUCT OVERVIEW .....</b>	 <b>12</b>
2.0 Users and Stakeholders .....	12
1.1 Project Perspective:.....	12
1.2 Interface: .....	12
1.3 Functional and Non-Functional Requirements: -.....	13
1.3.1 Functional Requirements .....	13
1.3.2 Non-Functional Requirements .....	13
1.4 Use Case Table .....	13
<b>CHAPTER 3: ARCHITECTURE .....</b>	<b>14</b>
3.0 Chat Architecture: How We Approach It .....	14
3.1 Chat App or Client Side.....	15
3.2 Chat Server Engine .....	16
<b>CHAPTER 4: SCREENSHOTS .....</b>	<b>17</b>
<b>CONCLUSION .....</b>	<b>19</b>
<b>REFERENCES.....</b>	<b>20</b>

## **LIST OF FIGURES**

FIGURE 1.1 MERN STACK .....	8
FIGURE 2.1 GUI OF APPLICATION .....	12
FIGURE 3.1 ARCHITECTURE .....	14
FIGURE 3.2 CLIENT-SIDE STRUCTURE TREE .....	15
FIGURE 3.3 PACKAGE.JSON FILE OF CLIENT-SIDE.....	15
FIGURE 3.4 SERVER-SIDE STRUCTURE TREE .....	16
FIGURE 3.5 DEPENDENCIES FOR SERVER .....	16
FIGURE 4.1 STARTING GUI AND LOGIN/SIGNUP.....	17
FIGURE 4.2 ADDING USER IN GROUP CHAT/ PERSONAL CHAT .....	17
FIGURE 4.3 CHATTING INTERFACE .....	18

## **LIST OF TABLES**

TABLE 1:USERS AND STAKEHOLDER .....	12
TABLE 2: USE CASE .....	13

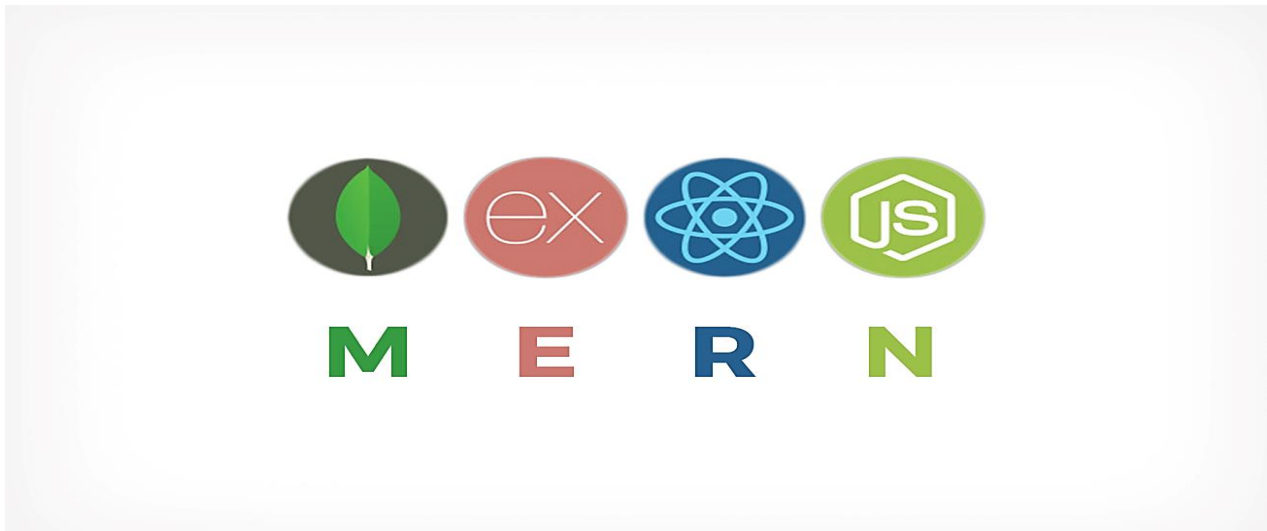
# CHAPTER 1: INTRODUCTION

## 1.0 Introduction

Today Developers around the world are making efforts to enhance user experience of using application as well as to enhance the developer's workflow of designing applications to deliver projects and rollout change requests under strict timeline. Stacks can be used to build web applications in the shortest span of time. The stacks used in web development are basically the response of software engineers to current demands. They have essentially adopted pre-existing frameworks (including JavaScript) to make their lives easier.

While there are many, MEAN and MERN are just two of the popular stacks that have evolved out of JavaScript. Both stacks are made up of open source components and offer an end-to-end framework for building comprehensive web apps that enable browsers to connect with databases. The common theme between the two is JavaScript and this is also the key benefit of using either stack. One can basically avoid any syntax errors or any confusion by just coding in one programming language, JavaScript. Another advantage of building web projects with MERN is the fact that one can benefit from its enhanced flexibility.

In order to understand MERN stack, we need to understand the four components that make up the MERN stack(fig.1), namely – MongoDB, Express.js, React and Node.js.



*Figure 1.1 MERN Stack*

## 1.1 Problem Statement

- This project is to create a chat application with a server and users to enable the users to chat with each other's.
- To develop an instant messaging solution to enable users to seamlessly communicate with each other.
- The project should be very easy to use enabling even a novice person to use it.
- This project can play an important role in organizational field where employees can connect through LAN.
- The main purpose of this project is to provide multi chatting functionality through network.



## 1.2 Innovative Ideas of Project

- **GUI:** Easy to use GUI (Graphical User Interface), hence any user with minimal knowledge of operating a system can use the software.
- **Platform independence:** The messenger operates on any system irrelevant of the underlying operating system.
- **Unlimited clients:** “N” number of users can be connected without any performance degradation of the server.

## 1.3 Project Objective

- **Communication:** To develop an instant messaging solution to enable users to seamlessly communicate with each other.
- **User friendliness:** The project should be very easy to use enabling even a novice person to use it.

## 1.4 Scope of The Project

- Broadcasting Chat Server Application is going to be a text communication software, it will be able to communicate between two computers using point to point communication.
- The limitation of Live Chat is it does not support audio conversations. To overcome this limitation, we are concurrently working on developing better technologies.
- Companies would like to have a communication software wherein they can communicate instantly within their organization.
- The fact that the software uses an internal network setup within the organization makes it very secure from outside attacks.



## 1.5 What is Node.js <sup>[5]</sup>?

- Node.js is a very powerful JavaScript-based platform built on Google Chrome's JavaScript V8 Engine. It is used to develop I/O intensive web applications like video streaming sites, single-page applications, and other web applications. Node.js is open source, completely free, and used by thousands of developers around the world.
- Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine). Node.js was developed by Ryan Dahl in 2009.
- Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.
- Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent.

### 1.5.1 Features of Node.js

1. **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
2. **I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous i.e. non-blocking. So, 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. It is also a reason that it is very fast.
3. **Single threaded:** Node.js follows a single threaded model with event looping.
4. **Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
5. **No buffering:** Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
6. **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js application.

## **1.6 What is Socket.io ?**

- **Socket.IO is a library that enables real-time, bidirectional and event-based communication between the browser and the server. It consists of:**
  - **a Node.js server: Source | API**
  - **a Javascript client library for the browser ( Which can be also run from Node js)**

### **1.6.1 Features of Socket.io**

- **reliability**
- **automatic reconnection.**
- **packet buffering.**
- **acknowledgments.**
- **broadcasting to all clients or to a subset of clients**
- **multiplexing (what we call "Namespace")**

## CHAPTER 2: PRODUCT OVERVIEW

The functionality of the chat application is to give the ability to chat with whoever is online on the application. The users and stakeholders will be a small group for now, the use cases will be what is available to the user, and the functional/nonfunctional requirements will be covered, as well as the milestones of the chat application.

### 2.0 Users and Stakeholders

*This section will deal with the users and stakeholders. The users will be using the chat application and the stakeholders will develop, maintain, and test the chat application.*

Team and I	We will be developing, maintaining, and testing the chat application through its phases of development.
Users	The users will be anyone who has the chat application and registers for it.

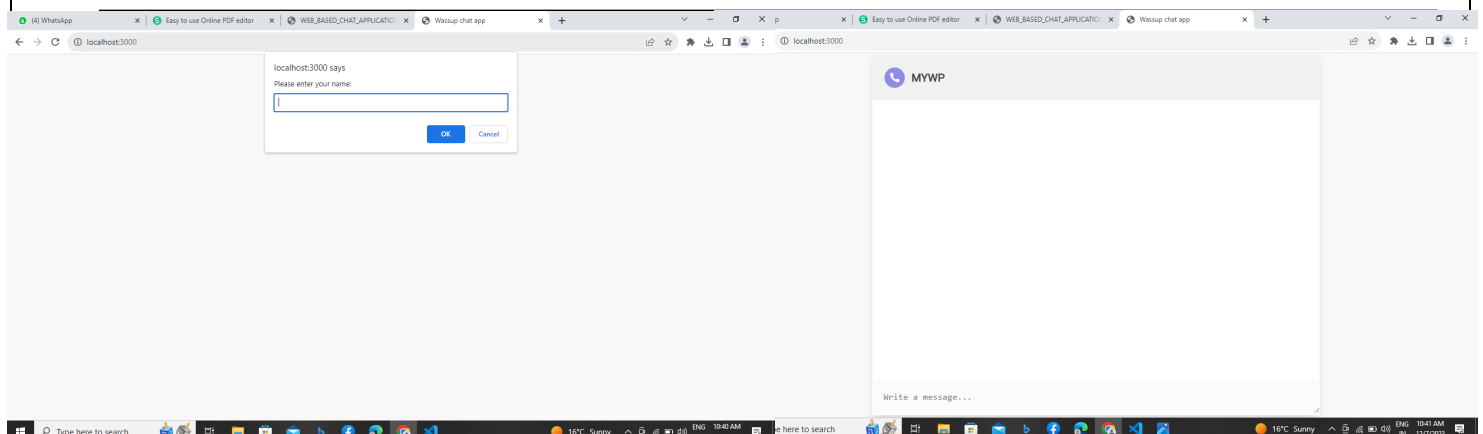
*Table 1:Users and Stakeholder*

### 1.1 Project Perspective:

- The system to be developed here is a Chat facility. It is a centralized system. It is Client-Server system with centralized database server. All local clients are connected to the centralized server via LAN.
- There is a two-way communication between different clients and server. This chat application can be used for group discussion. It allows users to find other logged in users.

### 1.2 Interface:

- This application interacts with the user through G.U.I. The interface is simple, easy to handle and self-explanatory.
- Once opened, user will easily come into the flow with the application and easily uses all interfaces properly. However, the basic interface available in our application(fig.2) is: -
  - Title panel
  - Message panel
  - Contact list panel
  - Status panel



*Figure 2.1 GUI of Application*

## 1.3 Functional and Non-Functional Requirements: -

### 1.3.1 Functional Requirements

- 2. User Registration:** User must be able to register for the application through an Email, Username and Password. On Opening the application, user must be able to register themselves or they can directly login if there have an account already. If user skips this step, user should able to chat. The user's email will be the unique identifier of his/her account on Chat Application.
- 3. Adding New Contacts:** The application should detect all contacts from the server database. If any of the contacts have user entered with Chat Application, those contacts must automatically be added to the users contact list on Chat Application.
- 4. Send Message:** User should be able to send instant message to any contact on his/her Chat Application contact list. User should be notified when message is successfully delivered to the recipient by coloring message.
- 5. Broadcast Message:** User should be able to create groups of contacts. User should be able to broadcast messages to these groups.

### 1.3.2 Non-Functional Requirements

- 1. Privacy:** Messages shared between users should be encrypted to maintain privacy.
- 2. Robustness:** In case user's system crashes, a backup of their chat history must be stored on remote database servers to enable recoverability.
- 3. Performance:** Application must be lightweight and must send messages instantly.

## 1.4 Use Case Table

<i>Level 0</i>	<i>Level 1</i>	<i>Level 2</i>	<i>Actor</i>
<b><i>Chat Application</i></b>	Authentication System	Registrar, Login, Logout	User and Admin
	Contact Form	Search/Find Users, Adding Users	User
	Chat Form	Send Message	User
	Monitor	Chat History	User and Admin

Table 2: Use Case

## CHAPTER 3: ARCHITECTURE

### 3.0 Chat Architecture: How We Approach It<sup>[6]</sup>

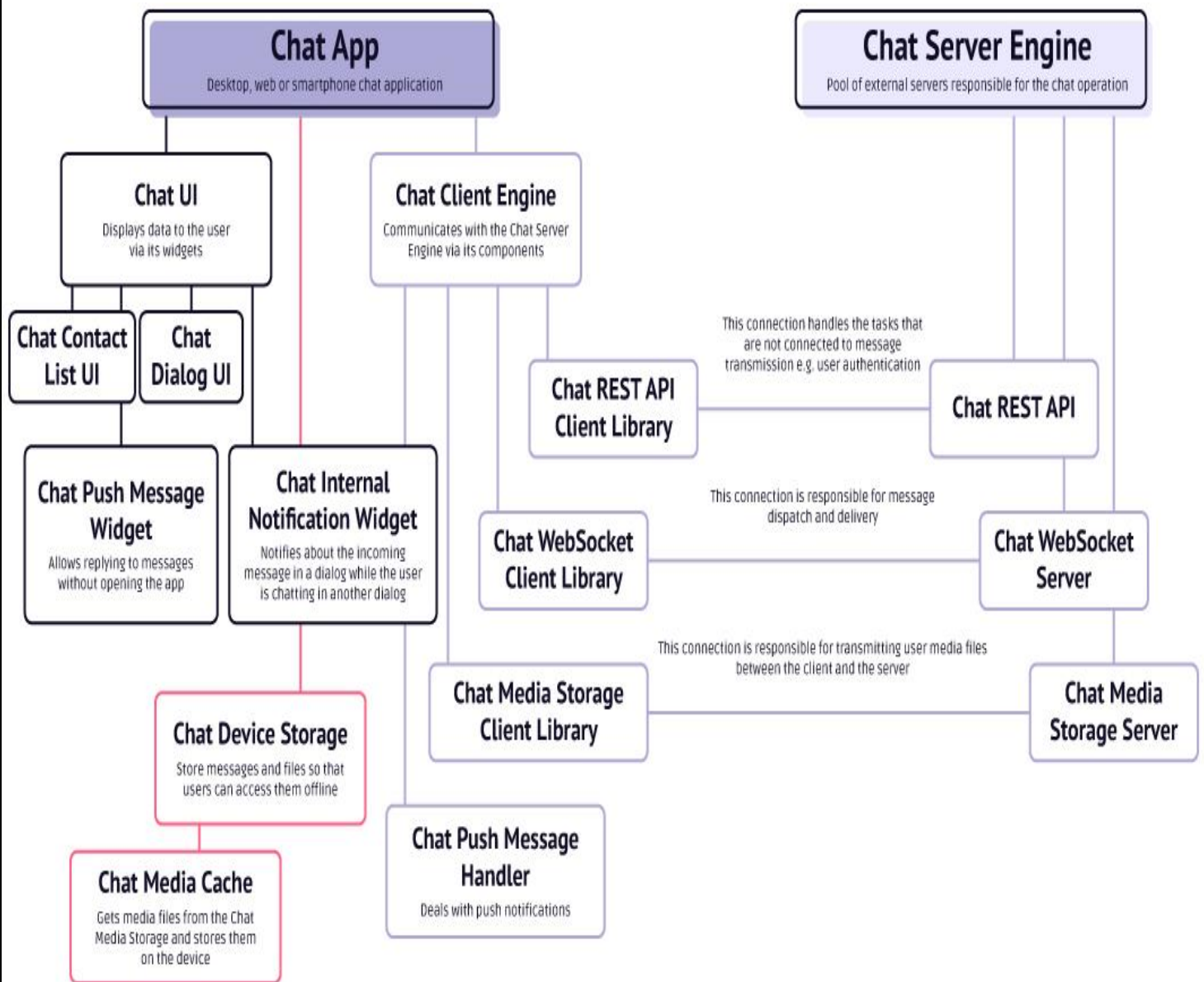


Figure 3.1 Architecture

A chat consists of two major parts(fig.3):

- **Chat App** or **client part**, which is a Web chat application. Build on Node js
- **Chat Server Engine** or **server part**, which is a pool of external servers responsible for the chat operation Using SOCKET.IO. This is the place where all the chat magic happens.

Both parts contain various components that communicate to each other and bring the chat into action

### 3.1 Chat App or Client Side

**Chat App** is the other major part of the chat architecture, the one that users directly interact with. It's split into two separate root components:

- **Chat Client Engine** (fig.3.2) handles all the communication with the Chat Server Engine via its internal components: Chat REST API Client Library and Chat WebSocket Client Library.
- **Chat UI** displays data to users: **Chat Contact List UI**, **Chat Dialog UI**

#### Component:

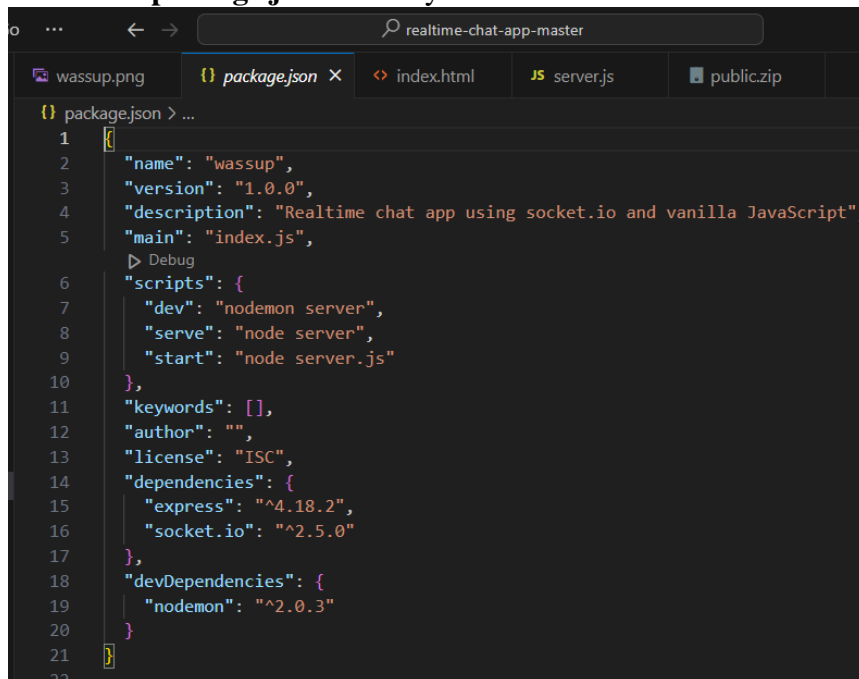
Components are the building blocks of any React app and a typical React app will have many of these. Simply put, a component is a JavaScript class or function that optionally accepts inputs i.e. properties(props) and returns a React element that describes how a section of the UI (User Interface) should appear.

App.js is the starting point of our React app.

A **package.json** file (fig.3.3):

- lists the packages your project depends on
- specifies versions of a package that your project can use.
- makes your build reproducible, and therefore easier to share with other developers.

A **package.json** file may look similar to this:



```
{
  "name": "wassup",
  "version": "1.0.0",
  "description": "Realtime chat app using socket.io and vanilla JavaScript",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon server",
    "serve": "node server",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2",
    "socket.io": "^2.5.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.3"
  }
}
```

Figure 3.3 Package.json file of Client-side

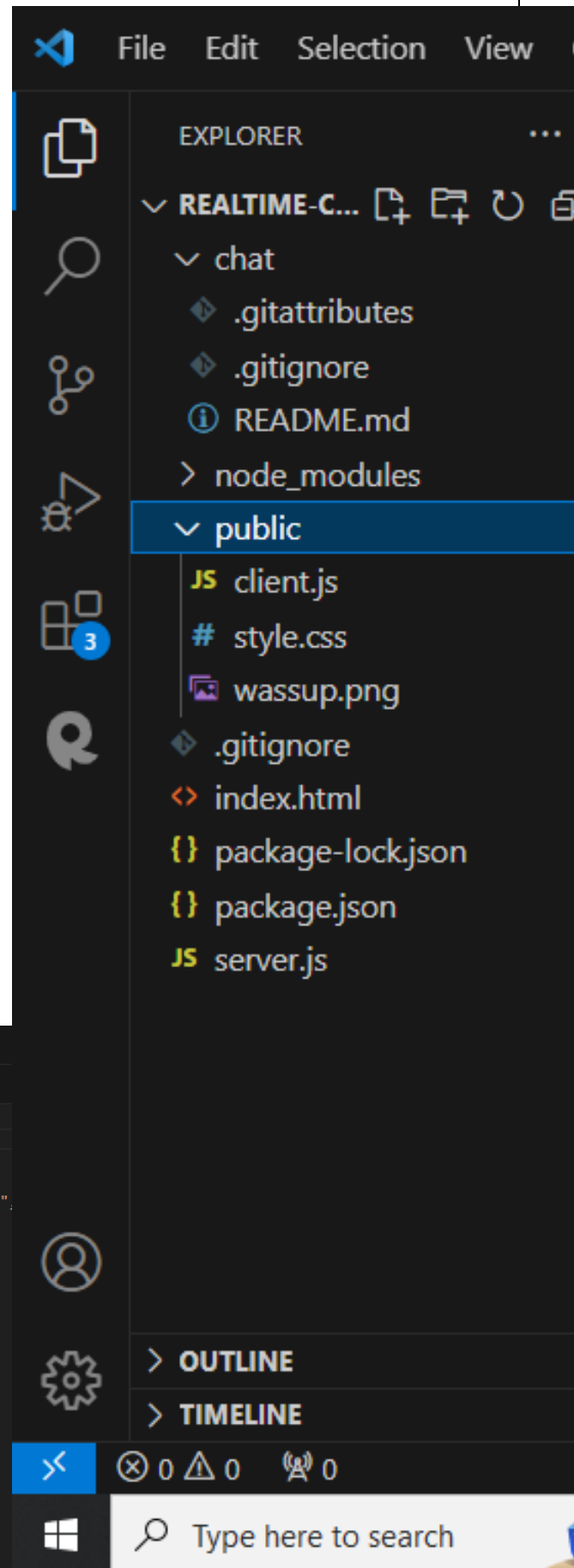
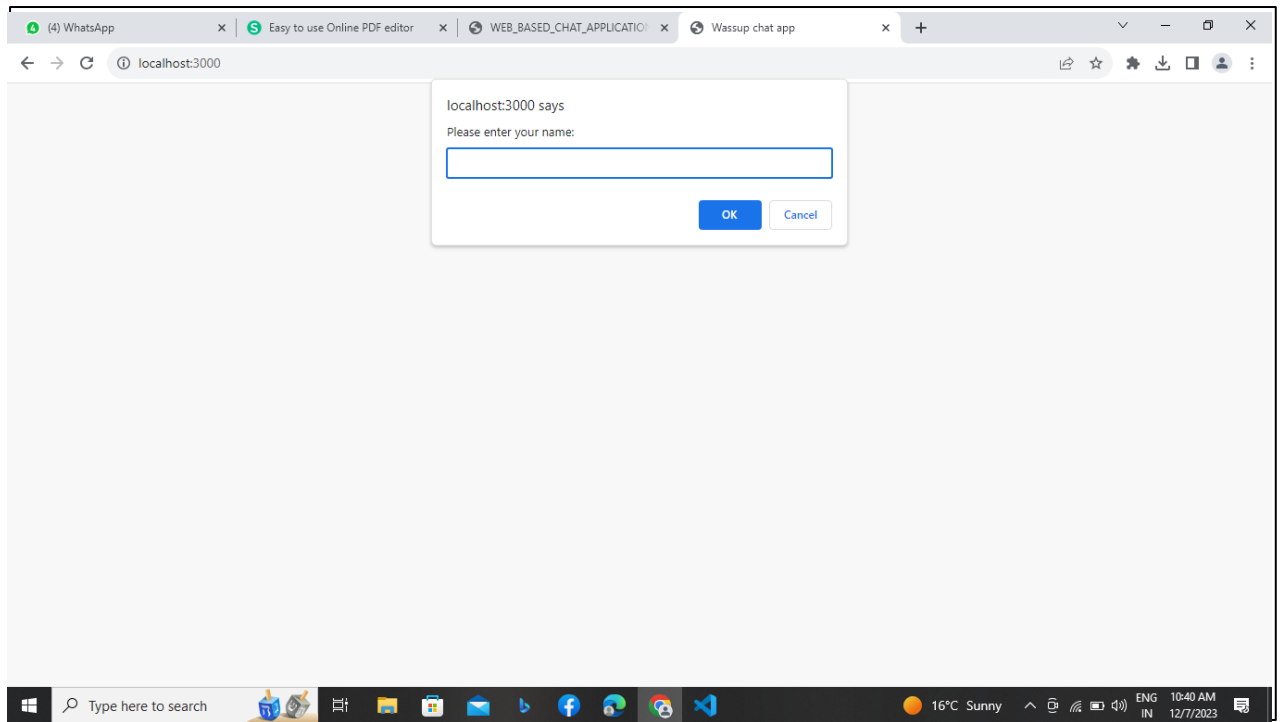


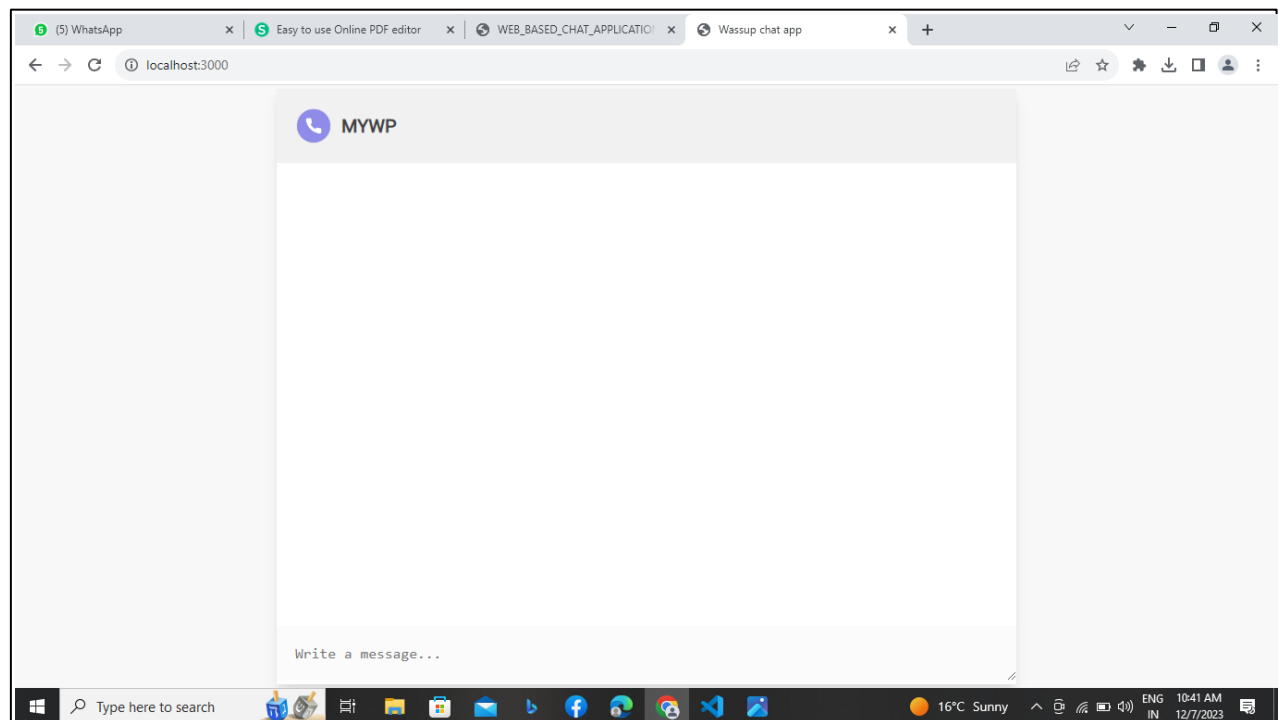
Figure 3.2 Client-Side Structure Tree



## **CHAPTER 4: SCREENSHOTS**



*Figure 4.1 Starting GUI and Login/Signup*



*Figure 4.2 Adding User in group chat/personal chat*

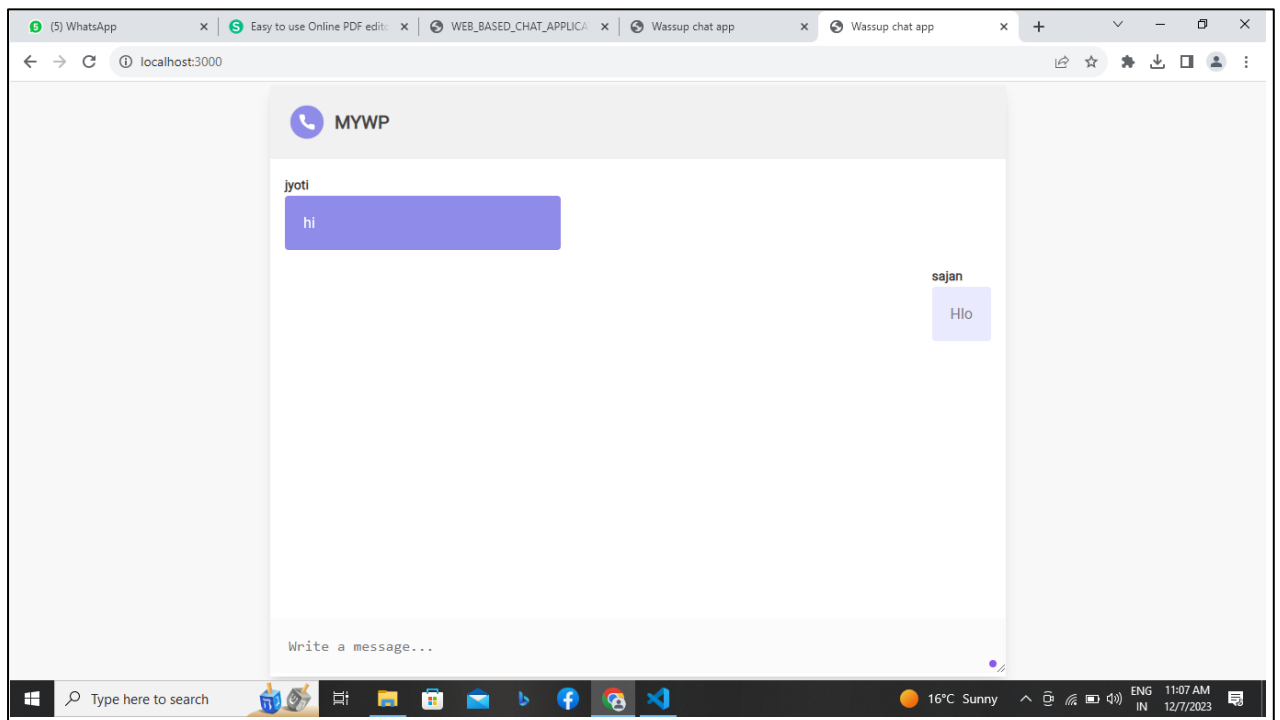


Figure 4.1 Chatting Interface

## **CONCLUSION**

There is always a room for improvements in any apps. Right now, we are just dealing with text communication. There are several chat apps which serve similar purpose as this project, but these apps were rather difficult to use and provide confusing interfaces. A positive first impression is essential in human relationship as well as in human computer interaction. This project hopes to develop a chat service Web app with high quality user interface. In future we may be extended to include features such as:

- File Transfer
- Voice Message
- Video Message
- Audio Call
- Video Call
- Group Call

## **REFERENCES**

- 
1. ExpressJS: <https://expressjs.com/en/guide/routing.html>,  
[2] <https://www.javatpoint.com/expressjs-tutorial>.
  2. Npm: <https://www.npmjs.com/>
  3. ReactJS: <https://reactjs.org/docs/getting-started.html>,  
[3] <https://www.javatpoint.com/reactjs-tutorial>,  
[4] [https://www.tutorialspoint.com/reactjs/reactjs\\_overview.htm](https://www.tutorialspoint.com/reactjs/reactjs_overview.htm).
  4. NodeJS: <https://nodejs.org/en/docs/>,  
[5] <https://www.javatpoint.com/nodejs-tutorial>.
  5. Socket.IO: [6] <https://yellow.systems/blog/guide-to-the-chat-architecture>
  6. REST API: [7] <https://www.toptal.com/nodejs/secure-rest-api-in-nodejs>