WEB BASED CHAT APPLICATION

MINOR PROJECT REPORT

Submitted in partial fulfilment of the requirements for the award of

the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

By

Amit Shekhawat	**Aniket Kumar Sharma**
Enrollment No.: 01015602716	**Enrollment No.: 0115602716**
Ashish Anurag	**Jagdish Chandra**
Enrollment No.: 0175602716	**Enrollment No.: 03015602716**

Guided by

Mrs. Uma Tomar,
Assistant Professor (CSE)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
Dr. AKHILESH DAS GUPTA INSTITUTE OF TECHNOLOGY & MANAGEMENT (AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI) NEW DELHI – 110053
NOV 2019

CANDIDATES' DECLARATION

It is hereby certified that the work which is being presented in the B. Tech Minor Project Report entitled **"WEB BASED CHAT APPLICATION"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** and submitted in the **Department of Computer Science & Engineering** of **Dr. Akhilesh Das Gupta Institute of Technology & Management, New Delhi (Affiliated to Guru Gobind Singh Indraprastha University, Delhi)** is an authentic record of our own work carried out during a period from **August 2019 to November 2019** under the guidance of **Mrs. Uma Tomar, Assistant Professor (CSE).**

The matter presented in the B. Tech Major Project Report has not been submitted by us for the award of any other degree of this or any other Institute.

Amit Shekhawat **Enrollment No.: 01015602716**	**Aniket Kumar Sharma** **Enrollment No.: 01115602716**
Ashish Anurag **Enrollment No.: 01715602716**	**Jagdish Chandra** **Enrollment No.: 03015602716**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge. They are permitted to appear in the External Major Project Examination

Mrs. Uma Tomar **Asst. Prof., CSE**	**Prof. (Dr.) Saurabh Gupta** **Head, CSE**

The B. Tech Minor Project Viva-Voce Examination of **AMIT SHEKHAWAT, ANIKET KUMAR SHARMA, ASHISH ANURAG, JAGDISH CHANDRA (01015602716, 01115602716, 01715602716, 03015602716)** has been held on

Prof. (Dr.) Anupam Kumar Sharma **Project Coordinator**	**(Signature of External Examiner)**

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.

ACKNOWLEGEMENT

We express our deep gratitude to **Mrs. Uma Tomar**, Assistant Professor, Department of Computer Science & Engineering for her valuable guidance and suggestion throughout my project work. We are thankful to **Prof. (Dr.) Anupam Kumar Sharma,** Project Coordinator for his valuable guidance.

We would like to extend our sincere thanks to **Prof**. (**Dr.) Saurabh Gupta, Head of Department,** for his time to time suggestions to complete our project work. We are also thankful to **Prof. (Dr.) Sanjay Kumar, Director** 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.

Amit Shekhawat **Enrollment No.: 01015602716**	**Aniket Kumar Sharma** **Enrollment No.: 01115602716**
Ashish Anurag **Enrollment No.: 01715602716**	**Jagdish Chandra** **Enrollment No.: 03015602716**

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

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.

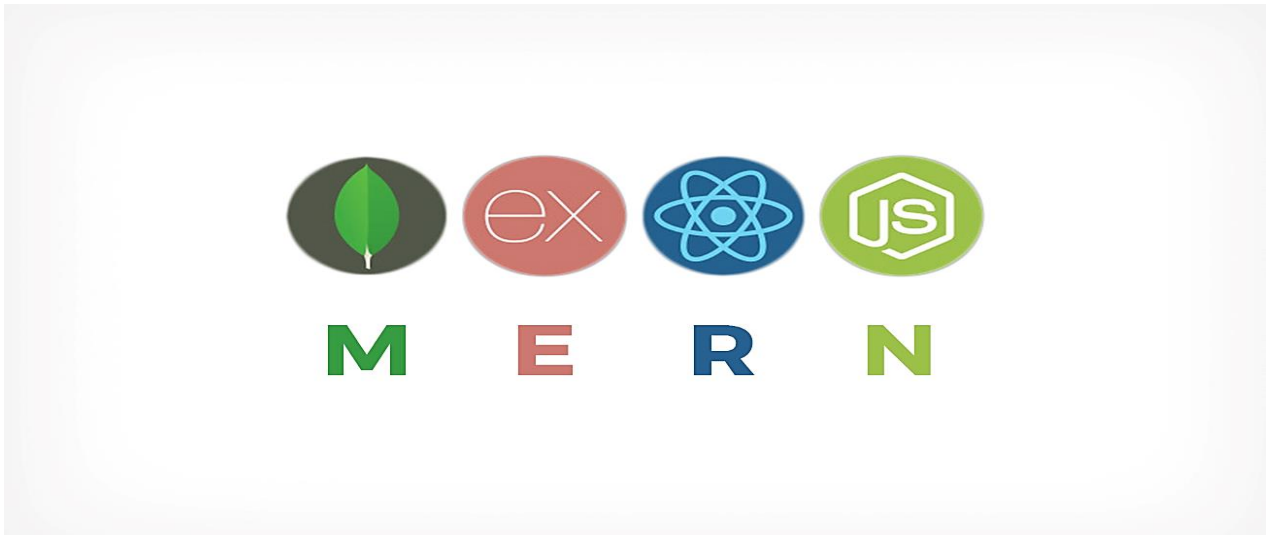


Figure 1.1 MERN Stack

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 MongoDB[1]?

- MongoDB is a cross-platform document-oriented NoSQL database used for high volume data storage that provides high performance, high availability and easy scalability.
- MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time. The document model maps to the objects in the application code, making data easy to work with.
- The data model available within MongoDB allows users to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
- MongoDB works on concept of collections and documents. Each database contains collections which in turn contains documents. Each document can have varying number of fields. The size and content of each document can also be different from each other.

1.5.1 Key Components of MongoDB Architecture

1. **_id** – This is a 24-digit unique identifier field required in every MongoDB document in the collection. The _id field is like the document's primary key. If the user creates a new document without an _id field, MongoDB will automatically create the field.
2. **Collection -** Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Typically, all documents in a collection are of similar or related purpose.
3. **Document -** A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.
4. **Database -** Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.
5. **Field -** A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.

1.6 What is Express.js[2]?

- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is an open source framework developed and maintained by the Node.js foundation.
- Express provides us the tools that are required to build our app, be it single-page, multi-page or hybrid web applications. It is flexible as there are numerous modules available on **npm(Node Package Manager)**, which can be directly plugged into Express.
- Unlike its competitors like Rails and Django, which have an opinionated way of building applications, Express has no "best way" to do something. It is very flexible and pluggable.
- Pug (earlier known as Jade) is a terse language for writing HTML templates. It produces HTML, supports dynamic code and code reusability (DRY). It is one of the most popular template languages used with Express.
- Express can be thought of as a layer built on the top of the Node.js that helps manage a server and routes. It allows users to setup middleware to respond to HTTP Requests and defines a routing table which is used to perform different actions based on HTTP method and URL.
- Express allows to dynamically render HTML Pages based on passing arguments to templates.
- Express is asynchronous and single threaded and performs I/O operations quickly.

1.6.1 Why use Express?

- Ultra-fast I/O.
- Asynchronous and single threaded.
- MVC like structure.
- Robust API makes routing easy.

1.7 What is React[3]?

- ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front-end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.
- A ReactJS application is made up of multiple components, each component responsible for outputting a small, reusable piece of HTML code. The components are the heart of all React applications. These Components can be nested with other components to allow complex applications to be built of simple building blocks. ReactJS uses virtual DOM based mechanism to fill data in HTML DOM. The virtual DOM works fast as it only changes individual DOM elements instead of reloading complete DOM every time.
- Instead of using regular JavaScript, React codes are written in something called JSX (JavaScript Syntax Extension). JSX is basically a syntax extension of regular JavaScript and is used to create React elements. These elements are then rendered to the React DOM. JSX is faster than normal JavaScript as it performs optimizations while translating to regular JavaScript.

1.7.1 Why use React[4]?

- Uses virtual DOM which is a JavaScript object. This will improve apps performance, since JavaScript virtual DOM is faster than the regular DOM.
- Can be used on client and server side as well as with other frameworks.
- Component and data patterns improve readability, which helps to maintain larger apps.

1.8 What is Node.js[5]?

- 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.

1.8.1 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.

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.

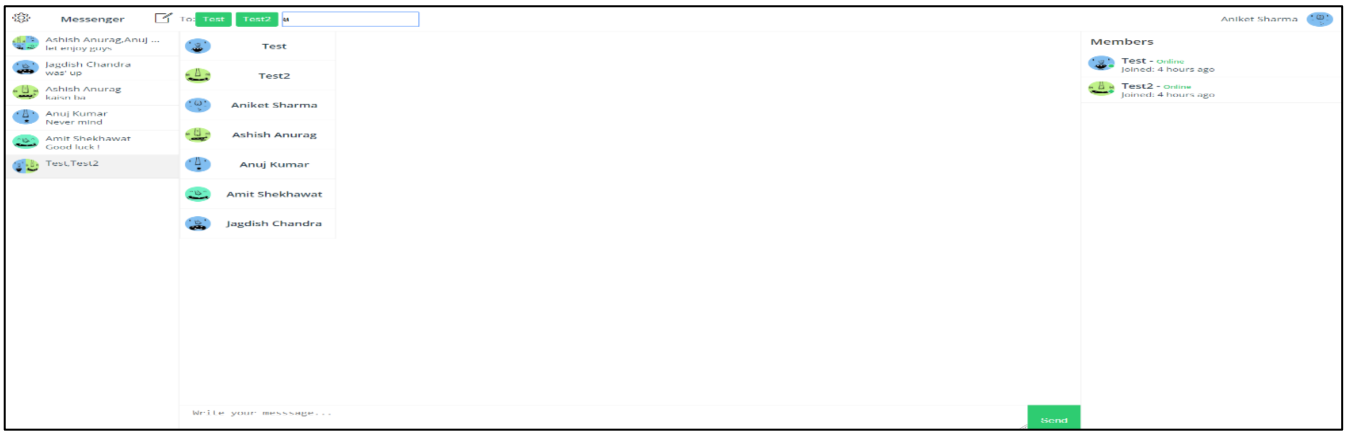
Table 1:Users and Stakeholder

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



Ashish Anurag,Anuj ...
let enjoy guys
Jagdish Chandra
was' up
Ashish Anurag
kaisn ba
Anuj Kumar
Never mind
Amit Shekhawat
Good luck !
Test,Test2

Test - Online
Joined: 4 hours ago
Test2 - Online
Joined: 4 hours ago

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.

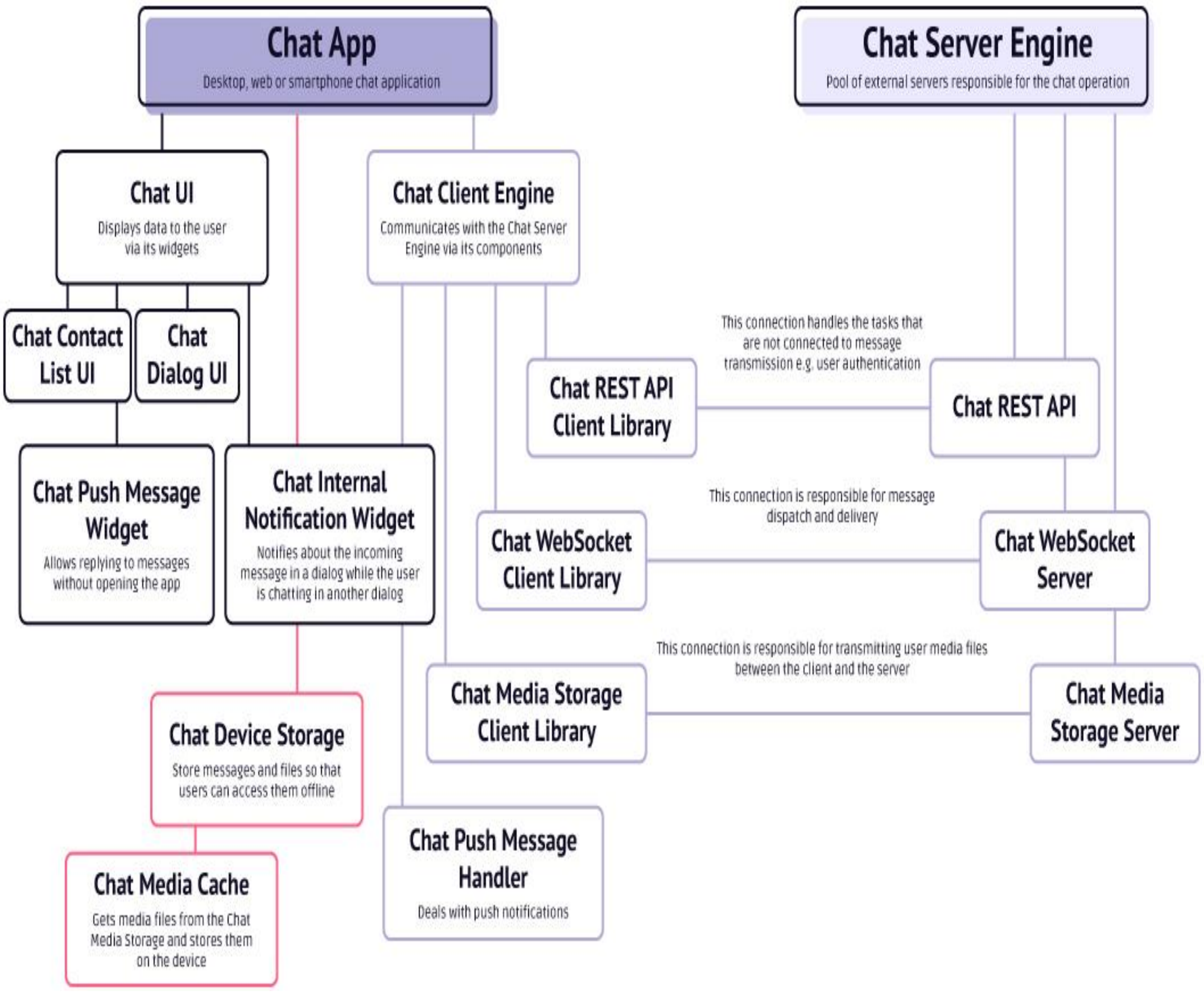
1.4 Use Case Table

Level 0	*Level 1*	*Level 2*	*Actor*
Chat Application	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[6]



Chat App
Desktop, web or smartphone chat application
Chat Server Engine
Pool of external servers responsible for the chat operation
Chat UI
Displays data to the user via its widgets
Chat Client Engine
Communicates with the Chat Server Engine via its components
Chat Contact List UI
Chat Dialog UI
This connection handles the tasks that are not connected to message transmission e.g. user authentication
Chat REST API Client Library
Chat REST API
Chat Push Message Widget
Allows replying to messages without opening the app
Chat Internal Notification Widget
Notifies about the incoming message in a dialog while the user is chatting in another dialog
This connection is responsible for message dispatch and delivery
Chat WebSocket Client Library
Chat WebSocket Server
This connection is responsible for transmitting user media files between the client and the server
Chat Media Storage Client Library
Chat Media Storage Server
Chat Device Storage
Store messages and files so that users can access them offline
Chat Media Cache
Gets media files from the Chat Media Storage and stores them on the device
Chat Push Message Handler
Deals with push notifications

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 React
- **Chat Server Engine** or **server part**, which is a pool of external servers responsible for the chat operation. 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:

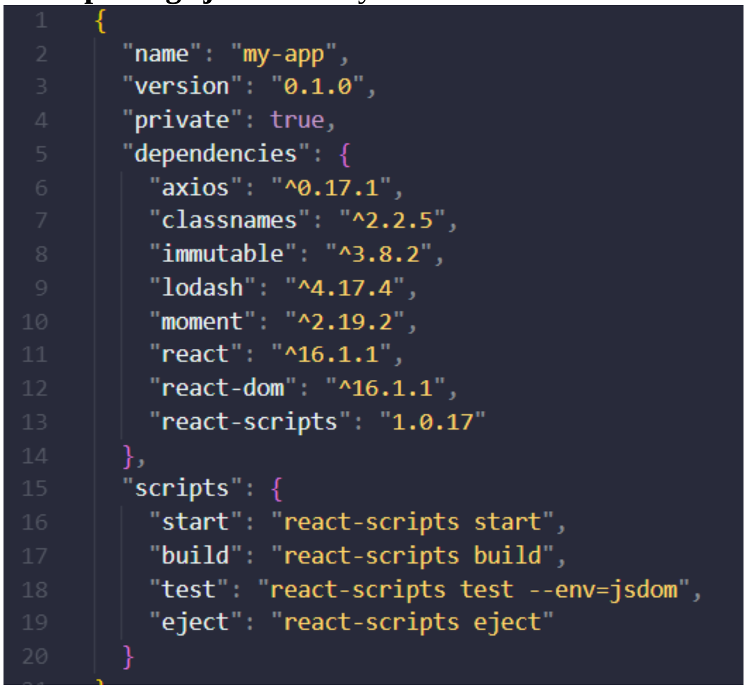
- **Chat Client Engine** (fig.3.2) handles all the communication with the Chat Server Engine via its internal components: Chat REST API Client Library and Chat WebSocket Client Library.
- **Chat UI** displays data to users: **Chat Contact List UI**, **Chat Dialog UI**

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.

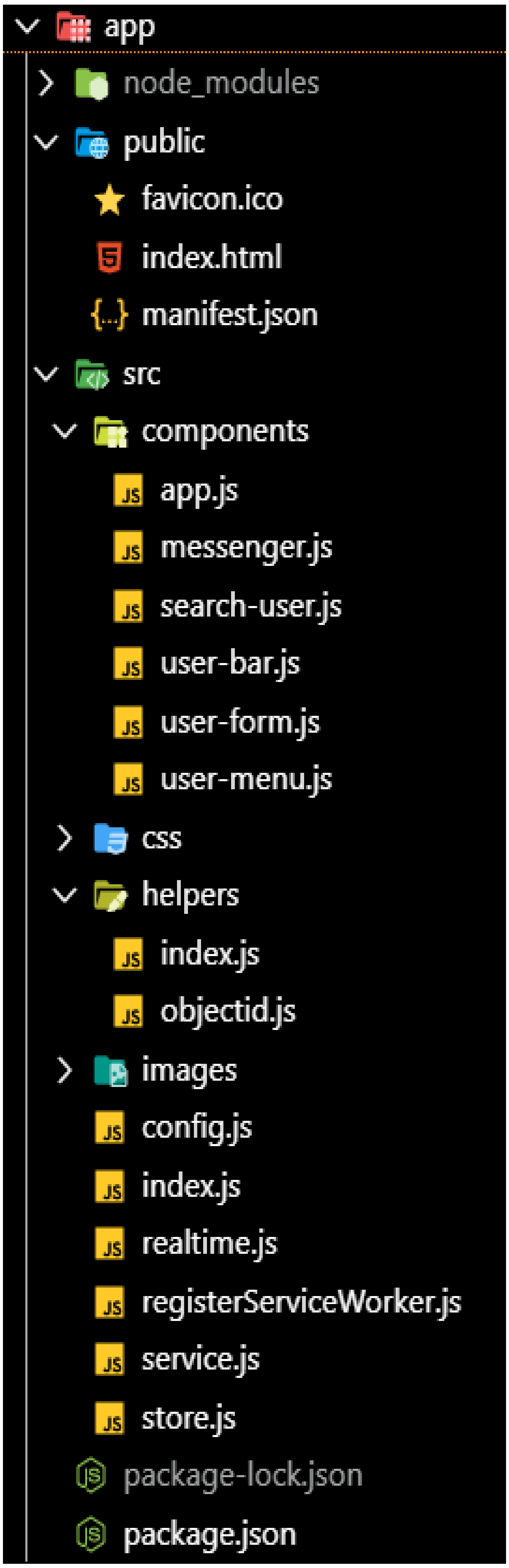
A **package.json** file(fig.3.3):
- lists the packages your project depends on
- specifies versions of a package that your project can use.
- makes your build reproducible, and therefore easier to share with other developers.

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



```json
{
  "name": "my-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "axios": "^0.17.1",
    "classnames": "^2.2.5",
    "immutable": "^3.8.2",
    "lodash": "^4.17.4",
    "moment": "^2.19.2",
    "react": "^16.1.1",
    "react-dom": "^16.1.1",
    "react-scripts": "1.0.17"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
```

Figure 3.3 Package.json file of Client-side



```
app
  node_modules
  public
    favicon.ico
    index.html
    manifest.json
  src
    components
      app.js
      messenger.js
      search-user.js
      user-bar.js
      user-form.js
      user-menu.js
    css
    helpers
      index.js
      objectid.js
    images
    config.js
    index.js
    realtime.js
    registerServiceWorker.js
    service.js
    store.js
  package-lock.json
  package.json
```

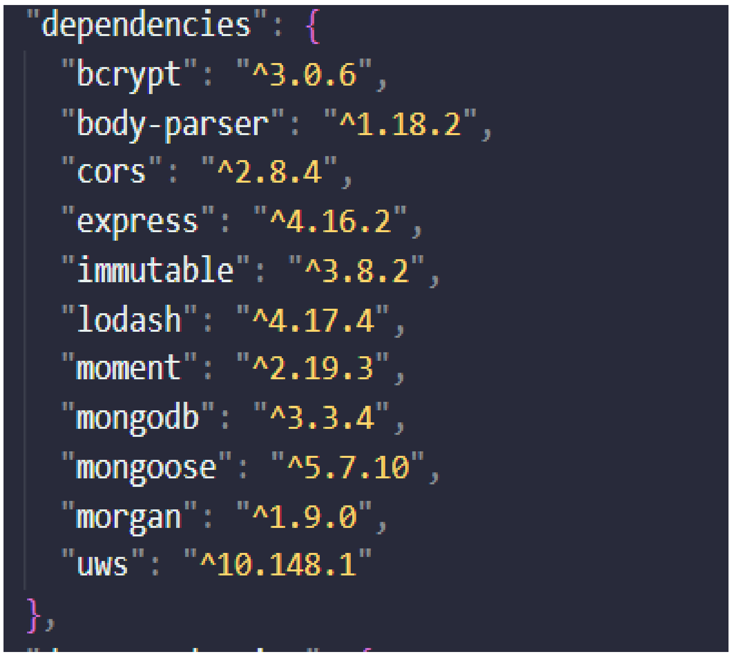
Figure 3.2 Client-Side Structure Tree

3.2 Chat Server Engine

This is a core of the chat architecture that handles message delivery and dispatch. In our version of chat architecture, it includes the following components(fig.3.5):

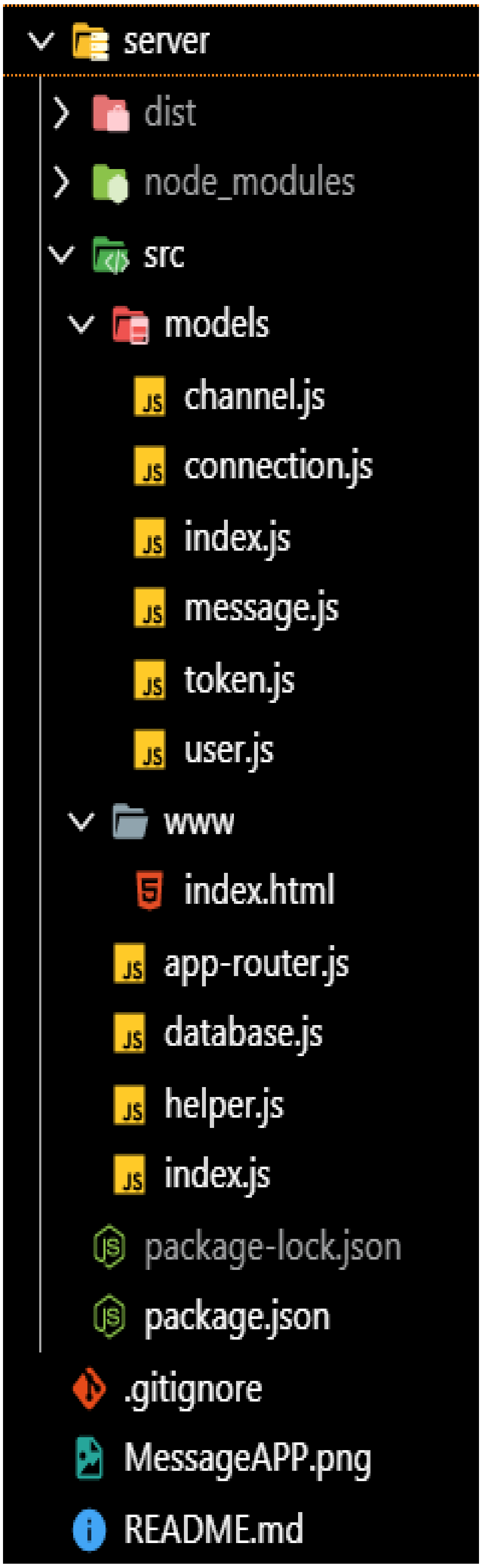
- **Chat REST API** handles the tasks that are not connected directly to message dispatch and delivery, such as user authentication, changing of user settings, friends invitation, downloading sticker packs, etc. The Chat App (the chat client part) communicates with the Chat REST API via the **Chat REST API[7] Client Library**.
- **Chat WebSocket Server** is responsible for transmitting messages between users. The Chat App communicates with the Chat WebSocket Server via the **Chat WebSocket Client Library**. This connection is open two ways; that means users don't have to make requests to the server if there are any messages for them, they just get them right away.

Dependencies(fig.3.6):
The third-party package or modules installed using **npm** are specified in this segment.
The **package.json** file is the heart of Node.js system. It is the manifest file of any Node.js project and contains the metadata of the project. The package.json file is the essential part to understand, learn and work with the Node.js. It is the first step to learn about development in Node.js.



```
"dependencies": {
  "bcrypt": "^3.0.6",
  "body-parser": "^1.18.2",
  "cors": "^2.8.4",
  "express": "^4.16.2",
  "immutable": "^3.8.2",
  "lodash": "^4.17.4",
  "moment": "^2.19.3",
  "mongodb": "^3.3.4",
  "mongoose": "^5.7.10",
  "morgan": "^1.9.0",
  "uws": "^10.148.1"
},
```

Figure 3.5 Dependencies for Server



```
server
  dist
  node_modules
  src
    models
      channel.js
      connection.js
      index.js
      message.js
      token.js
      user.js
    www
      index.html
    app-router.js
    database.js
    helper.js
    index.js
  package-lock.json
  package.json
.gitignore
MessageAPP.png
README.md
```

Figure 3.4 Server-Side Structure tree

CHAPTER 4: SCREENSHOTS

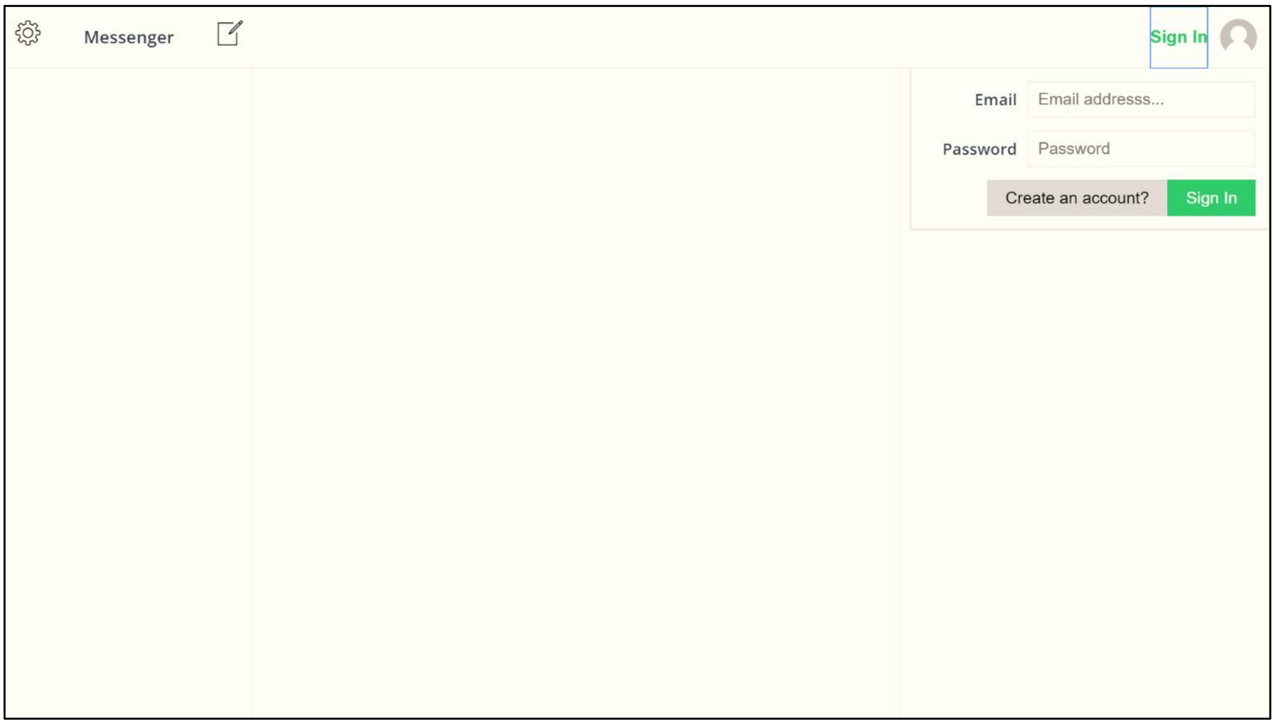


Figure 4.1 Starting GUI and Login/Signup

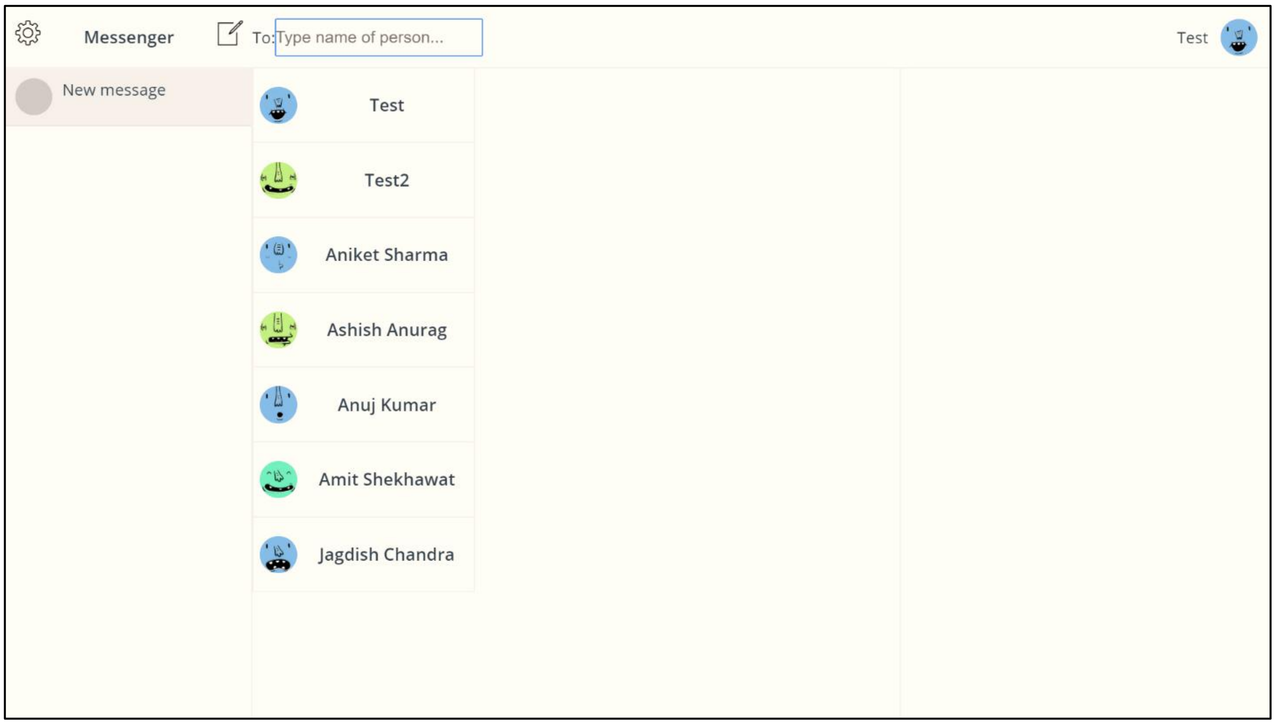


Figure 4.2 Adding User in group chat/ personal chat

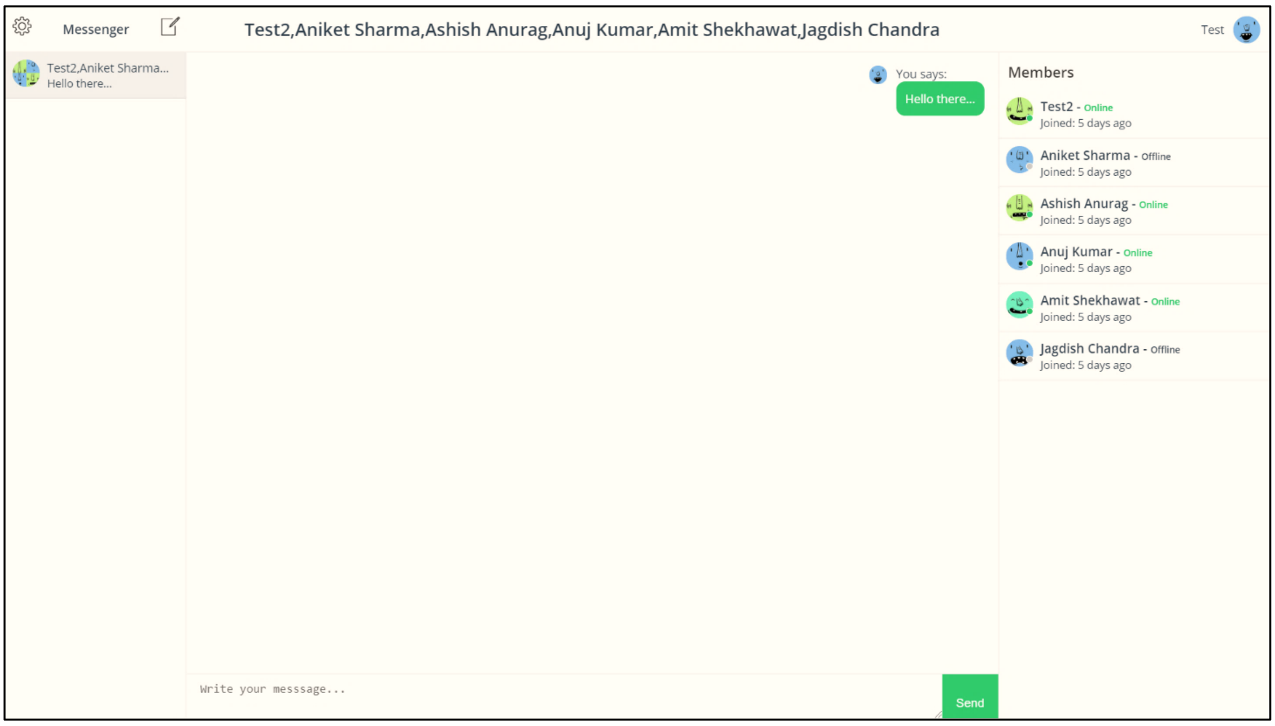


Figure 4.1 Chatting Interface

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. MongoDB: https://docs.mongodb.com/ecosystem/drivers/ ,
   [1] https://www.guru99.com/what-is-mongodb.html.
2. ExpressJS: https://expressjs.com/en/guide/routing.html,
   [2] https://www.javatpoint.com/expressjs-tutorial.
3. Npm: https://www.npmjs.com/
4. ReactJS: https://reactjs.org/docs/getting-started.html,
   [3] https://www.javatpoint.com/reactjs-tutorial,
   [4] https://www.tutorialspoint.com/reactjs/reactjs_overview.htm.
5. NodeJS: https://nodejs.org/en/docs/,
   [5] https://www.javatpoint.com/nodejs-tutorial.
6. WebSocket: [6]https://yellow.systems/blog/guide-to-the-chat-architecture
7. REST API: [7] https://www.toptal.com/nodejs/secure-rest-api-in-nodejs