Project report on

**RANGO RIDE HAILING ANDROID MOBILE APP**

Submitted by

**ABHINESH.V** (R2014002)

Under the Guidance of

**Mrs. MADHUPRIYA M.S.**

In partial fulfillment of the requirements

for the award of the degree -

**BACHELOR OF COMPUTER APPLICATION**

**BANGALORE NORTH UNIVERSITY**

**TAMAKA, KOLAR, BENGALURU – 560049**



**DEPARTMENT OF COMPUTER APPLICATION**

**EAST POINT COLLEGE OF HIGHER EDUCATION**

Jnana Prabha East Point campus, Virgonagar Post, Avalahalli,

Bengaluru – 560049, Karnataka, India.

**SEPTEMBER 2023**

**CERTIFICATE**

This is to certify that the project work entitled “**RANGO RIDE HAILING ANDROID MOBILE APP**” is a bonafide record of work done by **ABHINESH.V (R2014002*)*** submitted in partial fulfillment of the requirement for the award of the degree BACHELOR OF COMPUTER APPLICATION of Bangalore North University, Tamaka, Kolar, Bengaluru under the supervision of **Mrs.MADHUPRIYA.MS.**

Date: Signature of the Guide

Counter signed by

**GUIDE H.O.D PRINCIPAL**

External Viva-voce Conducted on ………………….

INTERNAL EXAMINER EXTERNAL EXAMINER

**DECLARATION**

I hereby declare that this project entitled “RANGO RIDE HAILING ANDROID MOBILE APP’ submitted to the Bangalore North University, Tamaka, Kolar, Bengaluru in partial fulfillment of the requirement for the award of the degree **BACHELOR OF COMPUTER APPLICATION** is a record of original work done by me during the period of study at **EAST POINT COLLEGE OF HIGHER EDUCATION** under the guidance of **Mrs.MADHUPRIYA.MS.**.

Date:  **ABHINESH.V (R2014002)**

**ACKNOWLEDGEMENT**

If words are considered as a symbol of approval and token of appreciation then let the words play the heralding role expressing my gratitude, The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose continuous cooperation made it possible, whose constant guidance and encouragement crown all efforts with success. We are grateful to teacher Mrs.MADHUPRIYA.M.S. for the guidance and constructive suggestions that helped us in the preparation of this project.

Date : ABHINESH.V (R2014002)

**SYNOPSIS**

The commune messenger is a real-time web based messenger. It is a peer to peer chatting application. The application would provide separate chat room for two individual people in private chat. Every user would be able to see and select between multiple contacts/users list to communicate in their own personalized dashboard.

JavaScript with Reactjs would be used to develop the client side UI along with HTML and CSS whereas the server-side will be coded with node.js with frameworks being express.js. The user data and other related data would be stored in a Mongodb database which is Document database. The database will be created and managed using Mongoose Dependency. The messenger would work on any modern browser that support ECMASCRIPT 5 and above versions of JavaScript.

**Features:**

* Realtime online messaging.
* Having multiple contacts list and friend to message in the

dashboard.

* previous messaging history.
* Setting unique Avatar for every user.

**CONTENT**

1. INTRODUCTION 01

1.1. Purpose 01

1.2. Scope 01

2. SYSTEM ANALYSIS AND Requirements 02

2.1. System Analysis 02

2.2. System Requirements. 02

3. SYSTEM DESIGN 06

3.1. Use Case 06

3.2. Data Flow 08

3.3. Entity Relationship diagram 11

3.4. Module Division 12

4. IMPLEMENTATION 14

4.1. Front-end 14

4.2. Back-end 46

4.3. Project Implementation 54

4.4. Project Snapshots 58

5. CONCLUSION 61

6. REFERENCES 62

**1. INTRODUCTION**

Communication is a mean for people to exchange messages. It has stated since the beginning of human creation. Distant communication began as early as 1800 with the introduction of television, telegraph and then telephone. Interestingly enough, telephone communication stands out as the fastest growing technology, from fixed line to mobile wireless, from voice call to data transfer, The emergence of computer network and telecommunication technologies bears the same objective that is to allow people to communicate. Chatting is a method using technology to bring people and ideas together despite of the geographical barriers. Our project is a example of a chat server. It is made up of client application which runs on the users browser and server application which runs on any computer on the network. To start chatting our client should get connected to server where they can do private chatting.

**1.1. Purpose**

The main aim of this project is to enable instant communication through a medium which is fast and easy to use. Providing users with personalized Dashboard to chat with friends and colleagues.

**1.2. Scope**

In the this Modern era instant communication is a norm for everyone, making instant chat or messaging applications like commune a ever growing daily life entity. The Scope of a web based chat application will always be on constant.

01

**2. SYSTEM ANALYSIS AND REQUIREMENTS**

**2.1. Analysis**

* The Commune Messenger is a peer to peer to communication web application that requires fast processing for immediate data transfer and requests.
* The Application is required to change the content of the page without refreshing the whole page which leads to smooth user experience.
* The User data and the Messages or chat Information need to be stored in a organized manner.

**2.2. Requirements**

***Hardware requirements:***

Any Computer that can run Modern Browsers that supports modern JavaScript i.e. ECMASCRIPT 5 and later.

Popular Browser that can be used are:

* Google Chrome
* Mozilla FireFox
* Brave Browser
* Microsoft Edge

02

***Software Tools or Requirements:***

JavaScript with MERN Stack:

**MERN** stack is a web development framework that consists of four open-source technologies: ***MongoDB, Express.js***, React, and ***Node.js***. It is a popular stack for building full-stack web applications and provides a complete solution for developing scalable and efficient web applications.

***Nodejs*** is a cross-platform, open-source server environment that can run on Windows, Linux, Unix, macOS, and more. Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.

Node.js lets developers use JavaScript to write command line tools and for server-side scripting. The functionality of running scripts server-side produces dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

Node.js has anevent-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games).

***MongoDB*** is a popular open-source NoSQL document-oriented database that stores data in a JSON-like format called BSON. It was first introduced in 2009 and has since become a widely-used database system, especially for web and mobile applications.

One of the key features of MongoDB is its flexibility and scalability. It allows for the storage of large amounts of data and can easily scale horizontally to handle increasing amounts of traffic and data. MongoDB also supports a variety of data types, including arrays, embedded documents, and binary data.

03

Another important feature of MongoDB is its ability to provide high availability and fault tolerance. This is achieved through replication, sharding, and automatic failover mechanisms.

MongoDB is also designed to work well with modern programming languages and frameworks such as Node.js, Ruby, Python, and Java. It provides a rich set of APIs and drivers that allow developers to easily integrate it into their applications.

***Express.js*** (or simply Express) is a popular open-source web application framework for Node.js. It provides a set of features for building web applications and APIs, including routing, middleware, and HTTP utilities.

Express allows developers to build web applications more easily by providing a simple, flexible, and minimalist approach to web development. It is built on top of Node.js and is designed to work well with other libraries and frameworks. It also provides a robust set of features such as error handling, session management, and templating engines.

One of the key benefits of using Express is its simplicity and flexibility. It provides a lightweight framework that allows developers to easily create web applications and APIs without being bogged down by excessive configuration and boilerplate code.

Express also has a large and active community, which means that developers can easily find documentation, tutorials, and support when needed. Additionally, it provides a rich set of third-party middleware and plugins, which can be used to extend its functionality even further.

04

***React***  (or simply React) is an open-source JavaScript library for building user interfaces. It was developed by Facebook and released in 2013. React allows developers to build complex and interactive user interfaces using a declarative syntax, which means that developers describe how the UI should look and React takes care of updating the DOM when changes are made.

React works by breaking the UI down into small, reusable components, which can be composed together to form more complex user interfaces. These components are written in JavaScript, and React provides a set of APIs for rendering and updating them.

One of the key benefits of using React is its performance. React uses a technique called Virtual DOM, which minimizes the number of changes that need to be made to the actual DOM. Instead, React updates a virtual representation of the DOM, compares it to the previous state, and makes only the necessary updates to the actual DOM. This approach allows React to update the UI more quickly and efficiently.

Another benefit of using React is its ecosystem. React has a large and active community, which means that developers can easily find documentation, tutorials, and support when needed. Additionally, there are many third-party libraries and tools available for React, which can be used to extend its functionality even further.

05

**3. SYSTEM DESIGN**

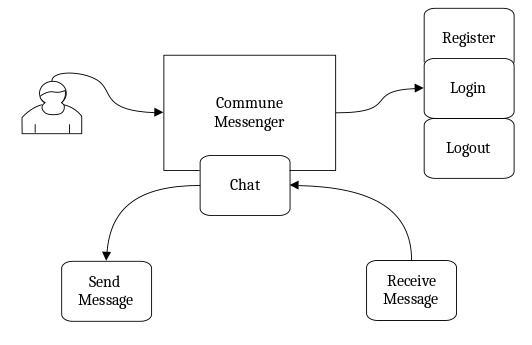
**3.1. Use Case**

A use case diagram is a type of behavioral diagram in the Unified Modeling Language (UML) that represents the interactions between a system and its external actors. It is a visual representation of the functional requirements of a system and illustrates the various use cases (i.e., scenarios or sequences of actions) that a system or a component of a system may have.

A use case diagram consists of several components, including actors, use cases, and relationships. Actors are external entities that interact with the system, such as users or other systems. Use cases represent the various functionalities or processes that the system provides to its actors. Relationships between actors and use cases show how the actors interact with the system and trigger the use cases.

Use case diagrams are useful for communicating the functional requirements of a system to stakeholders, including users, developers, and project managers. They can be used to identify the different features and functionalities of a system, as well as to identify any potential issues or gaps in the system's requirements. Use case diagrams can also be used to generate test cases and to validate the system's behavior.

06



Use Case scenario of the Commune Messenger has Multiple Users who can chat with other user of the application in real time.

The User will be able to Register with the application if they are new otherwise login if they are already a user of the application.

Users can Send and Receive messages from other users in text format and even send Emojis.

07

**3.2 Data Flow Diagram**

A data flow diagram (DFD) is a visual representation of the flow of data through a system or process. It is a type of diagram that shows how data is input, processed, and output in a system. DFDs are used to model the flow of data in a system, to identify the processes that are involved in handling the data, and to identify the data stores where the data is stored.

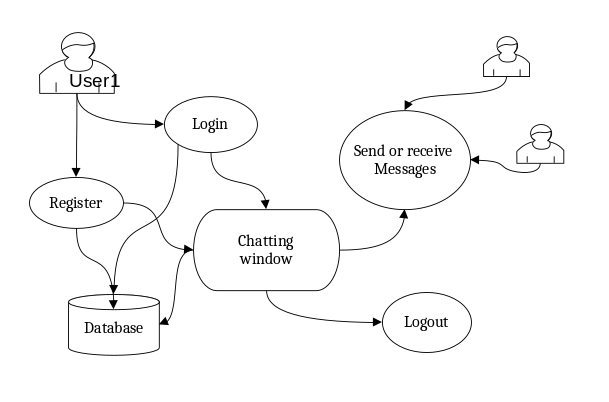
DFDs consist of several components, including processes, data stores, data flows, and external entities. Processes represent the various tasks or operations that are performed on the data. Data stores represent the locations where the data is stored or retrieved from. Data flows represent the movement of data between processes and data stores. External entities represent the sources or destinations of data outside of the system being modeled.

DFDs are useful for understanding and analyzing the flow of data in a system, and for identifying potential issues or areas for improvement. They are also useful for communicating the flow of data to stakeholders, including users, developers, and project managers. DFDs can be used to identify bottlenecks, redundancies, or inconsistencies in the data flow, and to design more efficient or effective processes for handling the data.

The application usage starts with User Logging into the application or registering if the user is new to the application. When the user logs in, the user is authenticated and redirected to the Chatting window or the Dashboard of the individual user. Through the chatting window the user can chat with other users in real-time. All the users data and messaging data are stored in the database instantly.

08

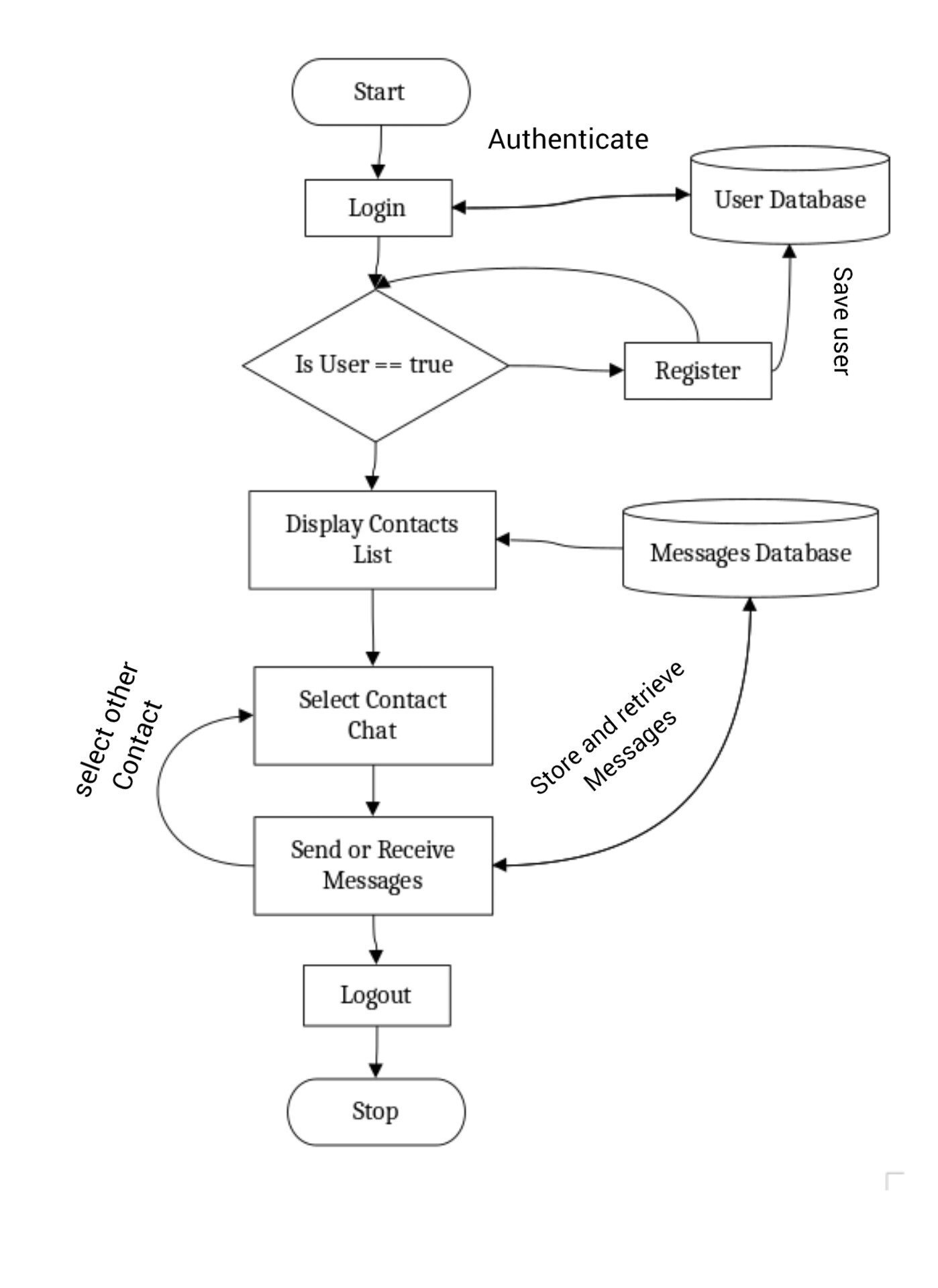
*Level One Data Flow Diagram:*



The Level one Data Flow Diagram of the Commune Messenger application represents the abstract view of the Data flow in the chat application.

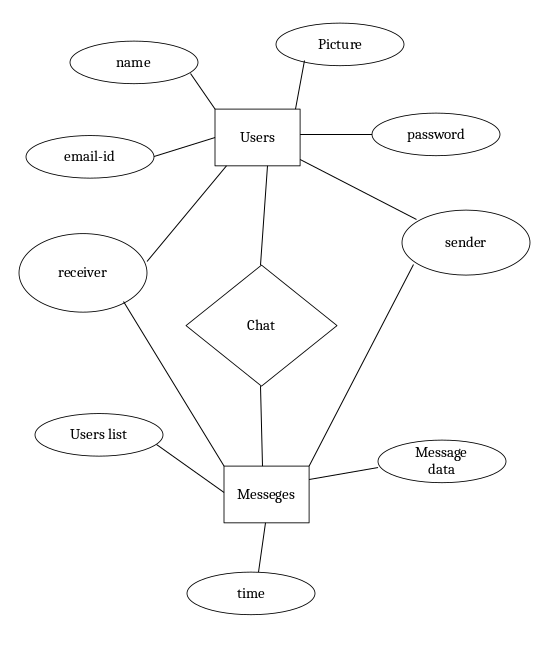
09

*Level Two Data Flow Diagram:*

**

10

**3.3 Entity Relationship Diagram**

****

The ER diagram of the commune Messenger application represents the Scheme or structure in which the data generated by the application is stored in the mongodb database. The User and Messages are the only two entities as a chat app has only user and ‘messages sent’. 11

**3.4 Module Division**

The Development process of the project is sub-divided into much smaller division or part so as to organize and Develop the Application in a transparent and understandable method. Each Module of the project development process were divided based on different feature or the functionality.

*The Module Division are as follows:*

Module 1. User Interface Design and Development.

Module 2. Database Schema Design

Module 3.Routes and API

Module 4.Login and Authentication Function

Module 5.Chat Function

*Module 1 : User Interface Design and Development*

The User Interface of the Commune Messenger was Developed with the help of the React js Framework , which is a JavaScript library that helps the developer to build interactive websites. React uses Virtual DOM which supports in creating single page applications.

The UI also uses many other dependencies like toastify which helps in providing pop-up notification features. ‘Socket.io client’ to communicate with the backend.

12

*Module 2: Database Schema Design*

MongoDB is used as the Database server for the Commune Application. In this Module we define the Database Model i.e Schema which specifies data storage structure.

We also Mongoose, a nodejs library which makes communicating with the database much easier.

*Module 3: Routes and API*

In this module we have coded the Routes and API for the project. Routes and API are the language through which the back-end communicates with front-end. To accomplish this task the CORS dependency was used to establish connection between the back-end and the front-end. ‘Socket.io’ was used to create a streamlined path between the back-end and the front-end.

*Module 4:Login and Authentication Function*

To save and recognize the users of the application we need a separate functionality called Authentication. We have made use the Bcrypt dependency in this module.

*Module 5: chat function*

In this Module we developed the chat functionality of the project which makes use of the Database and Routes and API’s to store and retrieve data i.e, messeges of the user.

13

**4. IMPLEMENTATION**

**4.1. Front-end Source Code:**

*client/src/index.js →*

import React from "react";

import ReactDOM from "react-dom";

import App from "./App";

import "./index.css";

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById("root")

);

*client/src/App.js*

import React from "react";

import { BrowserRouter, Routes, Route } from "react-router-dom";

import SetAvatar from "./components/SetAvatar";

import Chat from "./pages/Chat";

import Login from "./pages/Login";

import Register from "./pages/Register";

export default function App() {

return (

<BrowserRouter>

<Routes>

<Route path="/register" element={<Register />} />

<Route path="/login" element={<Login />} />

<Route path="/setAvatar" element={<SetAvatar />} />

<Route path="/" element={<Chat />} />

14

</Routes> /BrowserRouter>

);

}

*Client/src/pages/login.jsx*

import React, { useState, useEffect } from "react";

import axios from "axios";

import styled from "styled-components";

import { useNavigate, Link } from "react-router-dom";

import Logo from "../assets/logo.svg";

import { ToastContainer, toast } from "react-toastify";

import "react-toastify/dist/ReactToastify.css";

import { loginRoute } from "../utils/APIRoutes";

export default function Login() {

const navigate = useNavigate();

const [values, setValues] = useState({ username: "", password: "" });

const toastOptions = {

position: "bottom-right",

autoClose: 8000,

pauseOnHover: true,

draggable: true,

theme: "dark",

};

useEffect(() => {

if (localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)) {navigate("/");

}

}, []);

const handleChange = (event) => {

setValues({ ...values, [event.target.name]: event.target.value });

};

const validateForm = () => {

const { username, password } = values;

15

if (username === "") {

toast.error("Email and Password is required.", toastOptions);

return false;

} else if (password === "") {

toast.error("Email and Password is required.", toastOptions);

return false;

}

return true;

};

const handleSubmit = async (event) => {

event.preventDefault();

if (validateForm()) {

const { username, password } = values;

const { data } = await axios.post(loginRoute, {

username,

password,

});

if (data.status === false) {

toast.error(data.msg, toastOptions);

}

if (data.status === true) {

localStorage.setItem(

process.env.REACT\_APP\_LOCALHOST\_KEY,

JSON.stringify(data.user)

);

}

};

}

navigate("/");

return (

<>

<FormContainer>

<form action="" onSubmit={(event) => handleSubmit(event)}>

<div className="brand">

<img src={Logo} alt="logo" />

<h1>commune</h1>

</div>

<input

type="text"

placeholder="Username"}

16

name="username"

onChange={(e) => handleChange(e)}

min="3"

/>

<input

type="password"

placeholder="Password"

name="password"

onChange={(e) => handleChange(e)}

/>

<button type="submit">Log In</button>

<span>

Don't have an account ? <Link to="/register">Create One.</Link>

</span>

</form>

</FormContainer>

<ToastContainer />

</>

);

const FormContainer = styled.div`

height: 100vh;

width: 100vw;

display: flex;

flex-direction: column;

justify-content: center;

gap: 1rem;

align-items: center;

background-color: #131324;

.brand {

display: flex;

align-items: center;

gap: 1rem;

justify-content: center;

img {

height: 5rem;

}

h1 {

color: white;

text-transform: uppercase;

}

17

}

form {

display: flex;

flex-direction: column;

gap: 2rem;

background-color: #00000076;

border-radius: 2rem;

padding: 5rem;

}

input {background-color: transparent;

padding: 1rem;

border: 0.1rem solid #4e0eff;

border-radius: 0.4rem;

color: white;

width: 100%;

font-size: 1rem;

&:focus {

border: 0.1rem solid #997af0;

outline: none;

}

}

button {

background-color: #4e0eff;

color: white;

padding: 1rem 2rem;

border: none;

font-weight: bold;

cursor: pointer;

border-radius: 0.4rem;

font-size: 1rem;

text-transform: uppercase;

&:hover {

background-color: #4e0eff;

}

}

span {

color: white;

text-transform: uppercase;

a{

color: #4e0eff;

18

text-decoration: none;

font-weight: bold;

}

}

`;

*Client/src/pages/register.jsx*

import React, { useState, useEffect } from "react";

import axios from "axios";

import styled from "styled-components";

import { useNavigate, Link } from "react-router-dom";

import Logo from "../assets/logo.svg";

import { ToastContainer, toast } from "react-toastify";

import "react-toastify/dist/ReactToastify.css";

import { registerRoute } from "../utils/APIRoutes";

export default function Register() {

const navigate = useNavigate();const toastOptions = {

position: "bottom-right",

autoClose: 8000,

pauseOnHover: true,

draggable: true,

theme: "dark",

};

const [values, setValues] = useState({

username: "",

email: "",

password: "",

confirmPassword: "",

});

useEffect(() => {

if (localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)) {

navigate("/");

}

}, []);

19

const handleChange = (event) => {

setValues({ ...values, [event.target.name]: event.target.value });

};

const handleValidation = () => {

const { password, confirmPassword, username, email } = values;

if (password !== confirmPassword) {

toast.error(

"Password and confirm password should be same.",

toastOptions

);

return false;

} else if (username.length < 3) {

toast.error(

"Username should be greater than 3 characters.",

toastOptions

);

return false;

} else if (password.length < 8) {

toast.error(

"Password should be equal or greater than 8 characters.",

toastOptions

);

return false;

} else if (email === "") {

toast.error("Email is required.", toastOptions);

return false;

}

return true;

};

const handleSubmit = async (event) => {event.preventDefault();

if (handleValidation()) {

const { email, username, password } = values;

const { data } = await axios.post(registerRoute, {

username,

email,

password,

});

}

20

};

if (data.status === false) {

toast.error(data.msg, toastOptions);

}

if (data.status === true) {

localStorage.setItem(

process.env.REACT\_APP\_LOCALHOST\_KEY,

JSON.stringify(data.user)

);

navigate("/");

}

return (

<>

<FormContainer>

<form action="" onSubmit={(event) => handleSubmit(event)}>

<div className="brand">

<img src={Logo} alt="logo" />

<h1>commune</h1>

</div>

<input

type="text"

placeholder="Username"

name="username"

onChange={(e) => handleChange(e)}

/>

<input

type="email"

placeholder="Email"

name="email"

onChange={(e) => handleChange(e)}

/>

<input

type="password"

placeholder="Password"

name="password"

onChange={(e) => handleChange(e)}

/>

<input

type="password"

placeholder="Confirm Password"

name="confirmPassword"

21

onChange={(e) => handleChange(e)}}

/>

<button type="submit">Create User</button>

<span>

Already have an account ? <Link to="/login">Login.</Link>

</span>

</form>

</FormContainer>

<ToastContainer />

</>

);

const FormContainer = styled.div`

height: 100vh;

width: 100vw;

display: flex;

flex-direction: column;

justify-content: center;

gap: 1rem;

align-items: center;

background-color: #131324;

.brand {

display: flex;

align-items: center;

gap: 1rem;

justify-content: center;

img {

height: 5rem;

}

h1 {

color: white;

text-transform: uppercase;

}

}

form {

display: flex;

flex-direction: column;

gap: 2rem;

background-color: #00000076;

border-radius: 2rem;

padding: 3rem 5rem;

}

22

input {

background-color: transparent;

padding: 1rem;

border: 0.1rem solid #4e0eff;

border-radius: 0.4rem;

color: white;

width: 100%;

font-size: 1rem;

&:focus {

border: 0.1rem solid #997af0;}

outline: none;

}

button {

background-color: #4e0eff;

color: white;

padding: 1rem 2rem;

border: none;

font-weight: bold;

cursor: pointer;

border-radius: 0.4rem;

font-size: 1rem;

text-transform: uppercase;

&:hover {

background-color: #4e0eff;

}

}

span {

color: white;

text-transform: uppercase;

a{

color: #4e0eff;

text-decoration: none;

font-weight: bold;

}

}

`;

*23*

*Client/src/pages/chat.jsx*

import React, { useEffect, useState, useRef } from "react";

import axios from "axios";

import { useNavigate } from "react-router-dom";

import { io } from "socket.io-client";

import styled from "styled-components";

import { allUsersRoute, host } from "../utils/APIRoutes";

import ChatContainer from "../components/ChatContainer";

import Contacts from "../components/Contacts";

import Welcome from "../components/Welcome";

export default function Chat() {

const navigate = useNavigate();

const socket = useRef();

const [contacts, setContacts] = useState([]);

const [currentChat, setCurrentChat] = useState(undefined);

const [currentUser, setCurrentUser] = useState(undefined);

useEffect(async () => {

if (!localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)) {

navigate("/login");

} else {setCurrentUser(

await JSON.parse(

localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)

)

);

}

}, []);

useEffect(() => {

if (currentUser) {

socket.current = io(host);

socket.current.emit("add-user", currentUser.\_id);

}

}, [currentUser]);

}

useEffect(async () => {

if (currentUser) {

if (currentUser.isAvatarImageSet) {

const data = await axios.get(`${allUsersRoute}/${currentUser.\_id}`);

24

setContacts(data.data);

} else {

navigate("/setAvatar");

}}}, [currentUser]);

const handleChatChange = (chat) => {

setCurrentChat(chat);

};

return (

<>

<Container>

<div className="container">

<Contacts contacts={contacts} changeChat={handleChatChange} />

{currentChat === undefined ? (

<Welcome />

):(

<ChatContainer currentChat={currentChat} socket={socket} />

)}

</div>

</Container>

</>

);

const Container = styled.div`

height: 100vh;

width: 100vw;

display: flex;

flex-direction: column;

justify-content: center;

gap: 1rem;

align-items: center;

background-color: #131324;

.container {}

`;

height: 85vh;

width: 85vw;

background-color: #00000076;

display: grid;

grid-template-columns: 25% 75%;

@media screen and (min-width: 720px) and (max-width: 1080px){

grid-template-columns: 35% 65%;

}

25

*client/src/components/ChatContainer.jsx*

import React, { useState, useEffect, useRef } from "react";

import styled from "styled-components";

import ChatInput from "./ChatInput";

import Logout from "./Logout";

import { v4 as uuidv4 } from "uuid";

import axios from "axios";

import { sendMessageRoute, recieveMessageRoute } from "../utils/APIRoutes";

export default function ChatContainer({ currentChat, socket }) {

const [messages, setMessages] = useState([]);

const scrollRef = useRef();

const [arrivalMessage, setArrivalMessage] = useState(null);

useEffect(async () => {

const data = await JSON.parse(

localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)

);

const response = await axios.post(recieveMessageRoute, {

from: data.\_id,

to: currentChat.\_id,

});

setMessages(response.data);

}, [currentChat]);

useEffect(() => {

const getCurrentChat = async () => {

if (currentChat) {

await JSON.parse(

localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)

).\_id;

}

};getCurrentChat();

}, [currentChat]);

const handleSendMsg = async (msg) => {

const data = await JSON.parse(

26

localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)

);

socket.current.emit("send-msg", {

to: currentChat.\_id,

from: data.\_id,

msg,

});

await axios.post(sendMessageRoute, {

from: data.\_id,

to: currentChat.\_id,

message: msg,

});

const msgs = [...messages];

msgs.push({ fromSelf: true, message: msg });

setMessages(msgs);

};

useEffect(() => {

if (socket.current) {

socket.current.on("msg-recieve", (msg) => {

setArrivalMessage({ fromSelf: false, message: msg });

});

}

}, []);

useEffect(() => {

arrivalMessage && setMessages((prev) => [...prev, arrivalMessage]);

}, [arrivalMessage]);

useEffect(() => {

scrollRef.current?.scrollIntoView({ behavior: "smooth" });

}, [messages]);

return (

<Container>

<div className="chat-header">

<div className="user-details">

<div className="avatar">

<img

src={`data:image/svg+xml;base64,${currentChat.avatarImage}`}

alt=""

/>

</div>

<div className="username">

27

<h3>{currentChat.username}</h3>

</div>

</div>}

<Logout />

</div>

<div className="chat-messages">

{messages.map((message) => {

return (

<div ref={scrollRef} key={uuidv4()}>

<div

className={`message ${

message.fromSelf ? "sended" : "recieved"

}`}

>

<div className="content ">

<p>{message.message}</p>

</div>

</div>

</div>

);

})}

</div>

<ChatInput handleSendMsg={handleSendMsg} />

</Container>

);

const Container = styled.div`

display: grid;

grid-template-rows: 10% 80% 10%;

gap: 0.1rem;

overflow: hidden;

@media screen and (min-width: 720px) and (max-width: 1080px) {

grid-template-rows: 15% 70% 15%;

}

.chat-header {

display: flex;

justify-content: space-between;

align-items: center;

padding: 0 2rem;

.user-details {

display: flex;

28

align-items: center;

gap: 1rem;

.avatar {

img {

height: 3rem;

}

}

.username {

h3 {

color: white;

}

}

}

}.chat-messages {

padding: 1rem 2rem;

display: flex;

flex-direction: column;

gap: 1rem;

overflow: auto;

&::-webkit-scrollbar {

width: 0.2rem;

&-thumb {

background-color: #ffffff39;

width: 0.1rem;

border-radius: 1rem;

}

}

.message {

display: flex;

align-items: center;

.content {

max-width: 40%;

overflow-wrap: break-word;

padding: 1rem;

font-size: 1.1rem;

border-radius: 1rem;

color: #d1d1d1;

@media screen and (min-width: 720px) and (max-width: 1080px) {

max-width: 70%;

}

29

}

}

.sended {

justify-content: flex-end;

.content {

background-color: #4f04ff21;

}

}

.recieved {

justify-content: flex-start;

.content {

background-color: #9900ff20;

}

}

}

`;

30

*client/src/components//ChatInput.jsx*

import React, { useState } from "react";

import { BsEmojiSmileFill } from "react-icons/bs";

import { IoMdSend } from "react-icons/io";

import styled from "styled-components";import Picker from "emoji-picker-react";

export default function ChatInput({ handleSendMsg }) {

const [msg, setMsg] = useState("");

const [showEmojiPicker, setShowEmojiPicker] = useState(false);

const handleEmojiPickerhideShow = () => {

setShowEmojiPicker(!showEmojiPicker);

};

const handleEmojiClick = (event, emojiObject) => {

let message = msg;

message += emojiObject.emoji;

setMsg(message);

};

const sendChat = (event) => {

event.preventDefault();

if (msg.length > 0) {

handleSendMsg(msg);

setMsg("");

}

};

}

return (

<Container>

<div className="button-container">

<div className="emoji">

<BsEmojiSmileFill onClick={handleEmojiPickerhideShow} />

{showEmojiPicker && <Picker onEmojiClick={handleEmojiClick} />}

</div>

</div>

<form className="input-container" onSubmit={(event) => sendChat(event)}>

31

<input

type="text"

placeholder="type your message here"

onChange={(e) => setMsg(e.target.value)}

value={msg}

/>

<button type="submit">

<IoMdSend />

</button>

</form>

</Container>

);

const Container = styled.div`

display: grid;

align-items: center;

grid-template-columns: 5% 95%;

background-color: #080420;

padding: 0 2rem;

@media screen and (min-width: 720px) and (max-width: 1080px) {padding: 0 1rem;

gap: 1rem;

}

.button-container {

display: flex;

align-items: center;

color: white;

gap: 1rem;

.emoji {

position: relative;

svg {

font-size: 1.5rem;

color: #ffff00c8;

cursor: pointer;

}

.emoji-picker-react {

position: absolute;

top: -350px;

background-color: #080420;

box-shadow: 0 5px 10px #9a86f3;

border-color: #9a86f3;

32

.emoji-scroll-wrapper::-webkit-scrollbar {

background-color: #080420;

width: 5px;

&-thumb {

background-color: #9a86f3;

}

}

.emoji-categories {

button {

filter: contrast(0);

}

}

.emoji-search {

background-color: transparent;

border-color: #9a86f3;

}

.emoji-group:before {

background-color: #080420;

}

}

}

}

.input-container {

width: 100%;

border-radius: 2rem;

display: flex;

align-items: center;

gap: 2rem;

background-color: #ffffff34;

input {

width: 90%;

height: 60%;background-color: transparent;

color: white;

border: none;

padding-left: 1rem;

font-size: 1.2rem;

&::selection {

background-color: #9a86f3;

}

&:focus {

33

outline: none;

}

}

`;

}

button {

padding: 0.3rem 2rem;

border-radius: 2rem;

display: flex;

justify-content: center;

align-items: center;

background-color: #9a86f3;

border: none;

@media screen and (min-width: 720px) and (max-width: 1080px) {

padding: 0.3rem 1rem;

svg {

font-size: 1rem;

}

}

svg {

font-size: 2rem;

color: white;

}

}

34

*client/src/components/Contacts.jsx*

import React, { useState, useEffect } from "react";

import styled from "styled-components";

import Logo from "../assets/logo.svg";

export default function Contacts({ contacts, changeChat }) {

const [currentUserName, setCurrentUserName] = useState(undefined);

const [currentUserImage, setCurrentUserImage] = useState(undefined);

const [currentSelected, setCurrentSelected] = useState(undefined);

useEffect(async () => {

const data = await JSON.parse(

localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)

);

setCurrentUserName(data.username);}

setCurrentUserImage(data.avatarImage);

}, []);

const changeCurrentChat = (index, contact) => {

setCurrentSelected(index);

changeChat(contact);

};

return (

<>

{currentUserImage && currentUserImage && (

<Container>

<div className="brand">

<img src={Logo} alt="logo" />

<h3>commune</h3>

</div>

<div className="contacts">

{contacts.map((contact, index) => {

return (

<div

key={contact.\_id}

className={`contact ${

index === currentSelected ? "selected" : ""

}`}

onClick={() => changeCurrentChat(index, contact)}

> 35

<div className="avatar">

<img

src={`data:image/svg+xml;base64,${contact.avatarImage}`}

alt=""

/>

</div>

<div className="username">

<h3>{contact.username}</h3>

</div>

</div>

);

})}

</div>

<div className="current-user">

<div className="avatar">

<img

src={`data:image/svg+xml;base64,${currentUserImage}`}

alt="avatar"

/>

</div>

<div className="username">

<h2>{currentUserName}</h2>

</div>

</div>

</Container>

)}

</>

);const Container = styled.div`

display: grid;

grid-template-rows: 10% 75% 15%;

overflow: hidden;

background-color: #080420;

.brand {

display: flex;

align-items: center;

gap: 1rem;

justify-content: center;

img {

height: 2rem;

}

h3 {

36

color: white;

text-transform: uppercase;

}

}

.contacts {

display: flex;

flex-direction: column;

align-items: center;

overflow: auto;

gap: 0.8rem;

&::-webkit-scrollbar {

width: 0.2rem;

&-thumb {

background-color: #ffffff39;

width: 0.1rem;

border-radius: 1rem;

}

}

.contact {

background-color: #ffffff34;

min-height: 5rem;

cursor: pointer;

width: 90%;

border-radius: 0.2rem;

padding: 0.4rem;

display: flex;

gap: 1rem;

align-items: center;

transition: 0.5s ease-in-out;

.avatar {

img {

height: 3rem;

}

}

.username {

h3 {

color: white;

}

}}

}

37

.selected {

background-color: #9a86f3;

}

.current-user {

background-color: #0d0d30;

display: flex;

justify-content: center;

align-items: center;

gap: 2rem;

.avatar {

img {

height: 4rem;

max-inline-size: 100%;

}

}

.username {

h2 {

color: white;

}

}

@media screen and (min-width: 720px) and (max-width: 1080px) {

gap: 0.5rem;

.username {

h2 {

font-size: 1rem;

}

}

}

}

`;

38

*client/src/components/Logout.jsx*

import React from "react";

import { useNavigate } from "react-router-dom";

import { BiPowerOff } from "react-icons/bi";

import styled from "styled-components";

import axios from "axios";

import { logoutRoute } from "../utils/APIRoutes";

export default function Logout() {

const navigate = useNavigate();

const handleClick = async () => {

const id = await JSON.parse(

localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)

).\_id;

const data = await axios.get(`${logoutRoute}/${id}`);

if (data.status === 200) {localStorage.clear();

navigate("/login");

}

}

};

return (

<Button onClick={handleClick}>

<BiPowerOff />

</Button>

);

const Button = styled.button`

display: flex;

justify-content: center;

align-items: center;

padding: 0.5rem;

border-radius: 0.5rem;

background-color: #9a86f3;

border: none;

cursor: pointer;

svg {

font-size: 1.3rem;

color: #ebe7ff;

}

`;

39

*client/src/components/SetAvatar.jsx*

import React, { useEffect, useState } from "react";

import styled from "styled-components";

import axios from "axios";

import { Buffer } from "buffer";

import loader from "../assets/loader.gif";

import { ToastContainer, toast } from "react-toastify";

import "react-toastify/dist/ReactToastify.css";

import { useNavigate } from "react-router-dom";

import { setAvatarRoute } from "../utils/APIRoutes";

export default function SetAvatar() {

const api = `https://api.multiavatar.com/4645646`;

const navigate = useNavigate();

const [avatars, setAvatars] = useState([]);

const [isLoading, setIsLoading] = useState(true);

const [selectedAvatar, setSelectedAvatar] = useState(undefined);

const toastOptions = {

position: "bottom-right",

autoClose: 8000,

pauseOnHover: true,

draggable: true,

theme: "dark",

};useEffect(async () => {

if (!localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY))

navigate("/login");

}, []);

const setProfilePicture = async () => {

if (selectedAvatar === undefined) {

toast.error("Please select an avatar", toastOptions);

} else {

const user = await JSON.parse(

localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)

);

const { data } = await axios.post(`${setAvatarRoute}/${user.\_id}`, {

image: avatars[selectedAvatar],

});

40

}

};

if (data.isSet) {

user.isAvatarImageSet = true;

user.avatarImage = data.image;

localStorage.setItem(

process.env.REACT\_APP\_LOCALHOST\_KEY,

JSON.stringify(user)

);

navigate("/");

} else {

toast.error("Error setting avatar. Please try again.", toastOptions);

}

useEffect(async () => {

const data = [];

for (let i = 0; i < 4; i++) {

const image = await axios.get(

`${api}/${Math.round(Math.random() \* 1000)}`

);

const buffer = new Buffer(image.data);

data.push(buffer.toString("base64"));

}

setAvatars(data);

setIsLoading(false);

}, []);

return (

<>

{isLoading ? (

<Container>

<img src={loader} alt="loader" className="loader" />

</Container>

):(

<Container>

<div className="title-container">}

<h1>Pick an Avatar as your profile picture</h1>

</div>

<div className="avatars">

{avatars.map((avatar, index) => {

return (

<div

41

className={`avatar ${

selectedAvatar === index ? "selected" : ""

}`}

>

<img

src={`data:image/svg+xml;base64,${avatar}`}

alt="avatar"

key={avatar}

onClick={() => setSelectedAvatar(index)}

/>

</div>

);

})}

</div>

<button onClick={setProfilePicture} className="submit-btn">

Set as Profile Picture

</button>

<ToastContainer />

</Container>

)}

</>

);

const Container = styled.div`

display: flex;

justify-content: center;

align-items: center;

flex-direction: column;

gap: 3rem;

background-color: #131324;

height: 100vh;

width: 100vw;

.loader {

max-inline-size: 100%;

}

.title-container {

h1 {

color: white;

}

}

.avatars {

display: flex;

42

gap: 2rem;.avatar {

border: 0.4rem solid transparent;

padding: 0.4rem;

border-radius: 5rem;

display: flex;

justify-content: center;

align-items: center;

transition: 0.5s ease-in-out;

img {

height: 6rem;

transition: 0.5s ease-in-out;

}

}

.selected {

border: 0.4rem solid #4e0eff;

}

}

.submit-btn {

background-color: #4e0eff;

color: white;

padding: 1rem 2rem;

border: none;

font-weight: bold;

cursor: pointer;

border-radius: 0.4rem;

font-size: 1rem;

text-transform: uppercase;

&:hover {

background-color: #4e0eff;

}

}

`;

43

*client/src/components/Welcome.jsx*

import React, { useState, useEffect } from "react";

import styled from "styled-components";

import Robot from "../assets/robot.gif";

export default function Welcome() {

const [userName, setUserName] = useState("");

useEffect(async () => {

setUserName(

await JSON.parse(

localStorage.getItem(process.env.REACT\_APP\_LOCALHOST\_KEY)

).username

);

}, []);

return (

<Container>

<img src={Robot} alt="" />}

<h1>

Welcome, <span>{userName}!</span>

</h1>

<h3>Please select a chat to Start messaging.</h3>

</Container>

);

const Container = styled.div`

display: flex;

justify-content: center;

align-items: center;

color: white;

flex-direction: column;

img {

height: 20rem;

}

span {

color: #4e0eff;

}

`;

44

*client/src/utils/APIRoutes.js*

export const host = "http://localhost:5000";

export const loginRoute = `${host}/api/auth/login`;

export const registerRoute = `${host}/api/auth/register`;

export const logoutRoute = `${host}/api/auth/logout`;

export const allUsersRoute = `${host}/api/auth/allusers`;

export const sendMessageRoute = `${host}/api/messages/addmsg`;

export const recieveMessageRoute = `${host}/api/messages/getmsg`;

export const setAvatarRoute = `${host}/api/auth/setavatar`;

45

**4.2 Back-end Source Code:**

*server/index.js →*

const express = require("express");

const cors = require("cors");

const mongoose = require("mongoose");

const authRoutes = require("./routes/auth");

const messageRoutes = require("./routes/messages");

const app = express();

const socket = require("socket.io");

require("dotenv").config();

app.use(cors());

app.use(express.json());mongoose

.connect(process.env.MONGO\_URL, {

useNewUrlParser: true,

useUnifiedTopology: true,

})

.then(() => {

console.log("DB Connetion Successfull");

})

.catch((err) => {

console.log(err.message);

});

app.use("/api/auth", authRoutes);

app.use("/api/messages", messageRoutes);

const server = app.listen(process.env.PORT, () =>

console.log(`Server started on ${process.env.PORT}`)

);

const io = socket(server, {

cors: {

origin: "http://localhost:3000",

credentials: true,

},

});

global.onlineUsers = new Map();

io.on("connection", (socket) => {

global.chatSocket = socket;

46

socket.on("add-user", (userId) => {

onlineUsers.set(userId, socket.id);

});

socket.on("send-msg", (data) => {

const sendUserSocket = onlineUsers.get(data.to);

if (sendUserSocket) {

socket.to(sendUserSocket).emit("msg-recieve", data.msg);

}

});

});

*server/Controllers/messageController.js →*

const Messages = require("../models/messageModel");

module.exports.getMessages = async (req, res, next) => {

try {

const { from, to } = req.body;const messages = await Messages.find({

users: {

$all: [from, to],

},

}).sort({ updatedAt: 1 });

const projectedMessages = messages.map((msg) => {

return {

fromSelf: msg.sender.toString() === from,

message: msg.message.text,

};

});

res.json(projectedMessages);

} catch (ex) {

next(ex);

}

};

module.exports.addMessage = async (req, res, next) => {

try {

const { from, to, message } = req.body;

const data = await Messages.create({

message: { text: message },

users: [from, to],

47

sender: from,

});

if (data) return res.json({ msg: "Message added successfully." });

else return res.json({ msg: "Failed to add message to the database" });

} catch (ex) {

next(ex);

}

};

*server/Controllers/userController.js →*

const User = require("../models/userModel");

const bcrypt = require("bcrypt");

module.exports.login = async (req, res, next) => {

try {

const { username, password } = req.body;

const user = await User.findOne({ username });

if (!user)

return res.json({ msg: "Incorrect Username or Password", status: false });

const isPasswordValid = await bcrypt.compare(password, user.password);

if (!isPasswordValid)

return res.json({ msg: "Incorrect Username or Password", status: false });

delete user.password;

return res.json({ status: true, user });

} catch (ex) {

next(ex);}

};

module.exports.register = async (req, res, next) => {

try {

const { username, email, password } = req.body;

const usernameCheck = await User.findOne({ username });

if (usernameCheck)

return res.json({ msg: "Username already used", status: false });

48

const emailCheck = await User.findOne({ email });

if (emailCheck)

return res.json({ msg: "Email already used", status: false });

const hashedPassword = await bcrypt.hash(password, 10);

const user = await User.create({

email,

username,

password: hashedPassword,

});

delete user.password;

return res.json({ status: true, user });

} catch (ex) {

next(ex);

}

};

module.exports.getAllUsers = async (req, res, next) => {

try {

const users = await User.find({ \_id: { $ne: req.params.id } }).select([

"email",

"username",

"avatarImage",

"\_id",

]);

return res.json(users);

} catch (ex) {

next(ex);

}

};

module.exports.setAvatar = async (req, res, next) => {

try {

const userId = req.params.id;

const avatarImage = req.body.image;

const userData = await User.findByIdAndUpdate(

userId,

{

isAvatarImageSet: true,

avatarImage,

},

{ new: true }

49

);

return res.json({

isSet: userData.isAvatarImageSet,image: userData.avatarImage,

});

} catch (ex) {

next(ex);

}

};

module.exports.logOut = (req, res, next) => {

try {

if (!req.params.id) return res.json({ msg: "User id is required " });

onlineUsers.delete(req.params.id);

return res.status(200).send();

} catch (ex) {

next(ex);

}

};

50

*server/Routes/auth.js →*

const {

login,

register,

getAllUsers,

setAvatar,

logOut,

} = require("../controllers/userController");

const router = require("express").Router();

router.post("/login", login);

router.post("/register", register);

router.get("/allusers/:id", getAllUsers);

router.post("/setavatar/:id", setAvatar);

router.get("/logout/:id", logOut);

module.exports = router;

*server/Routes/messages.js →*

const { addMessage, getMessages } = require("../controllers/messageController");

const router = require("express").Router();

router.post("/addmsg/", addMessage);

router.post("/getmsg/", getMessages);

module.exports = router;

51

*server/Models/userModel.js*

const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({

username: {

type: String,

required: true,

min: 3,

max: 20,

unique: true,

},

email: {

type: String,

required: true,

unique: true,

max: 50,

},

password: {

type: String,

required: true,

min: 8,

},

isAvatarImageSet: {

type: Boolean,

default: false,

},

avatarImage: {

type: String,

default: "",

},

});module.exports = mongoose.model("Users", userSchema);

52

*server/Models/messageModel.js*

const mongoose = require("mongoose");

const MessageSchema = mongoose.Schema(

{

message: {

text: { type: String, required: true },

},

users: Array,

sender: {

type: mongoose.Schema.Types.ObjectId,

ref: "User",

required: true,

},

},

{

timestamps: true,

}

);

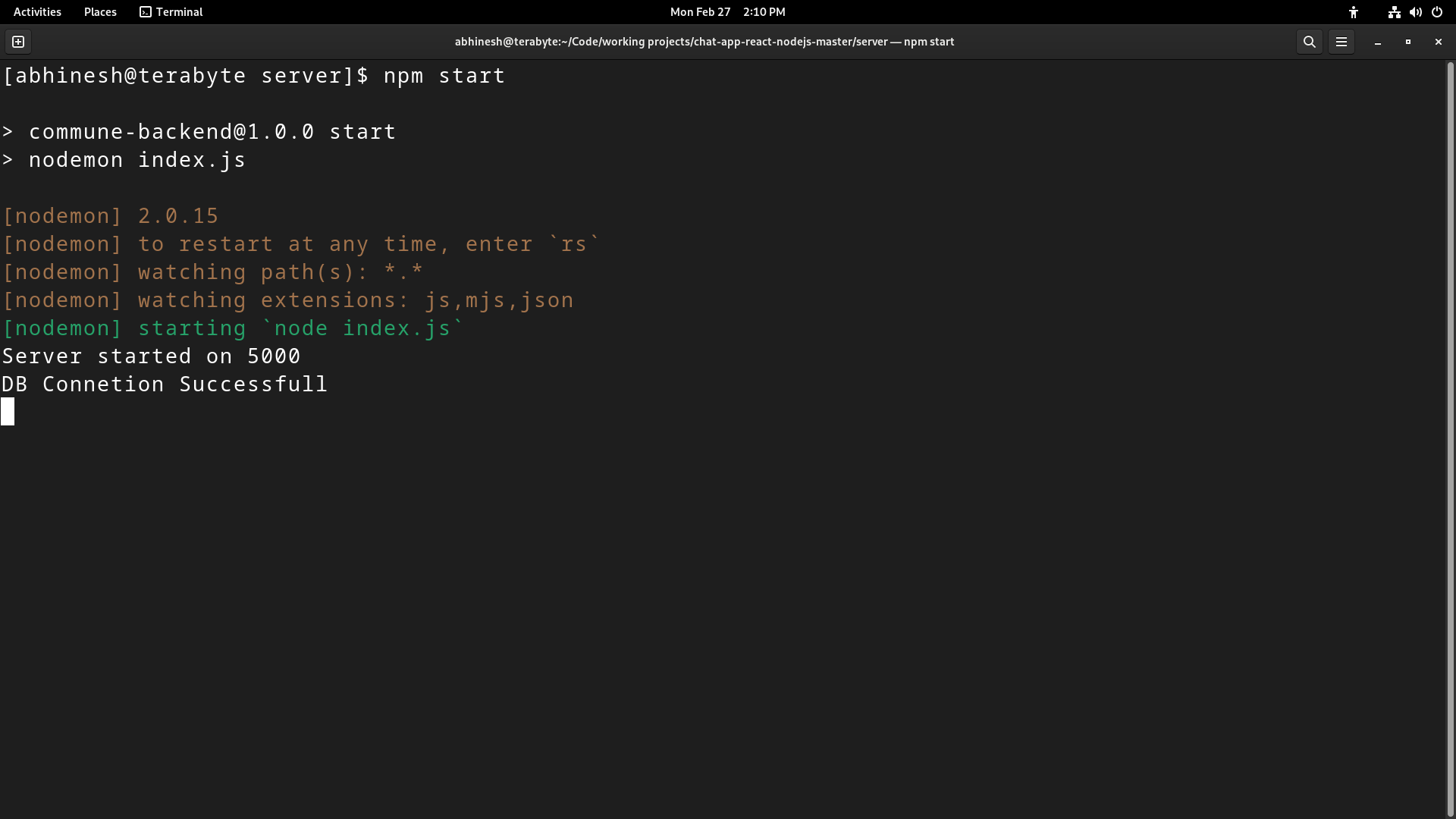
module.exports = mongoose.model("Messages", MessageSchema);

53

**4.3. PROJECT IMPLEMENTATION**

***Starting the Backend Server:***

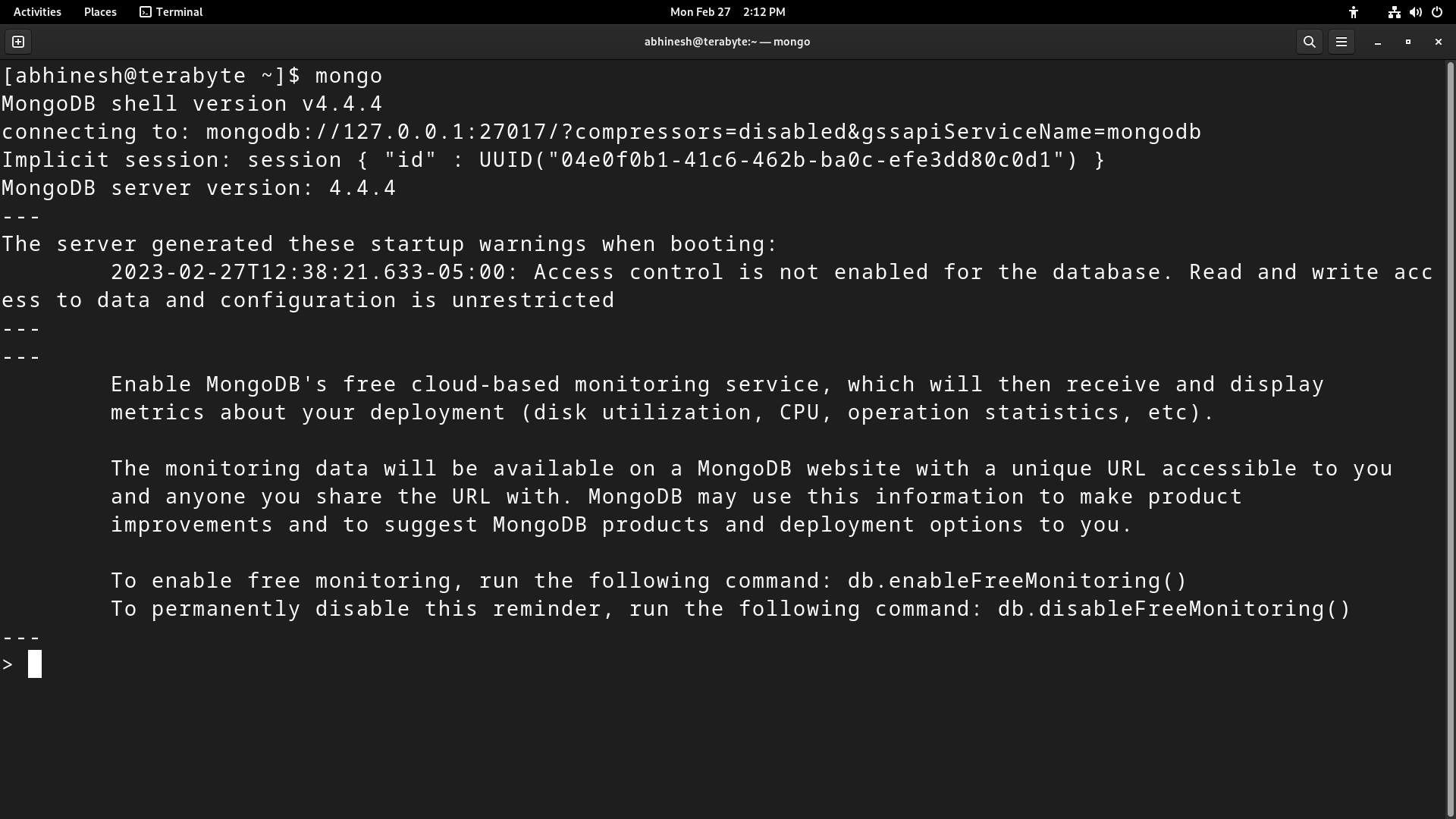
Run the npm start command in the root directory of the Server of the commune messenger through a terminal to Start the backend server.

****

54

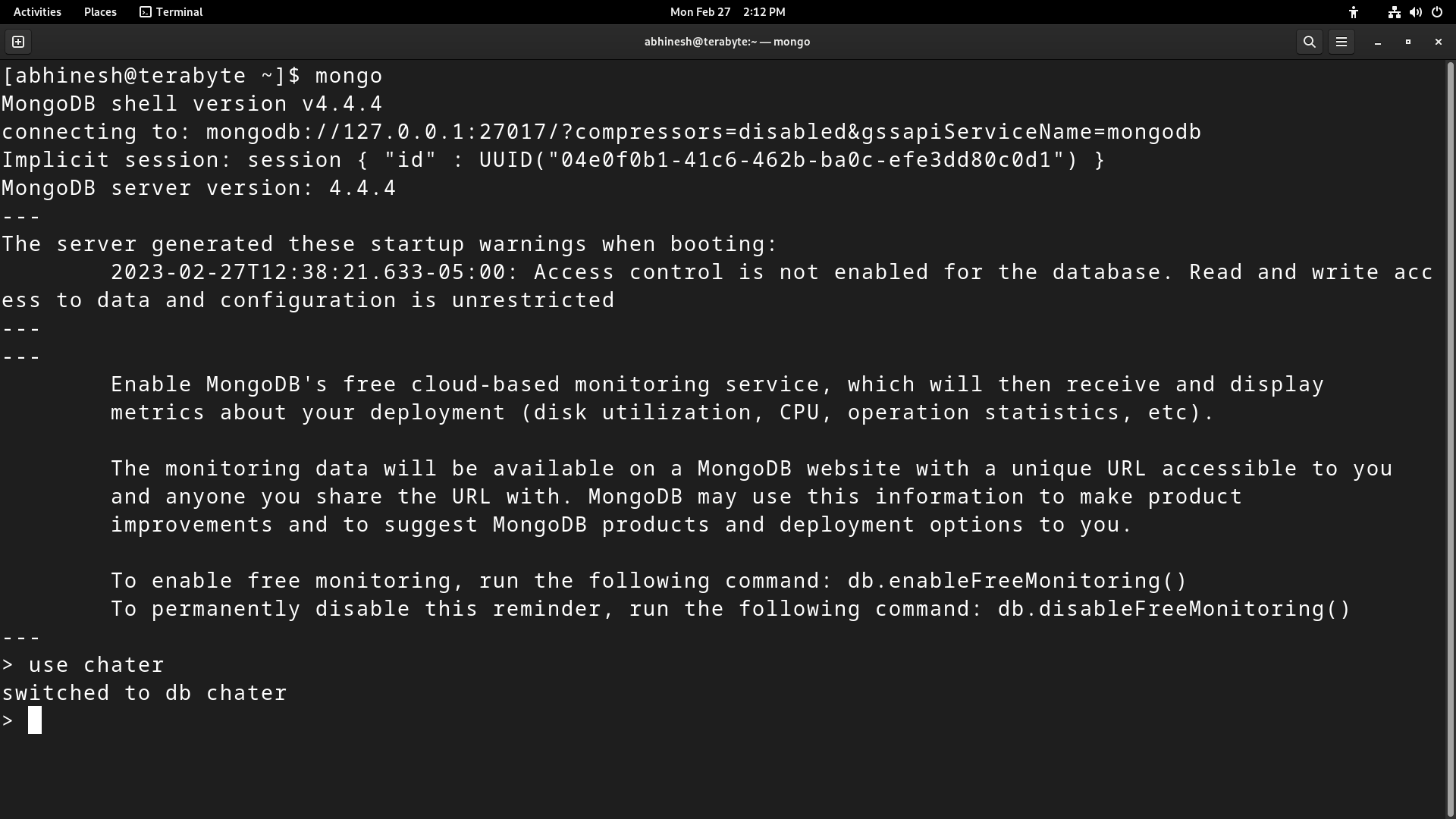
***Starting the MongoDB database :***

1. Run the ‘mongo’ command in the terminal to instantiated the mongoDB server which starts on "mongodb://127.0.0.1:27017”



55

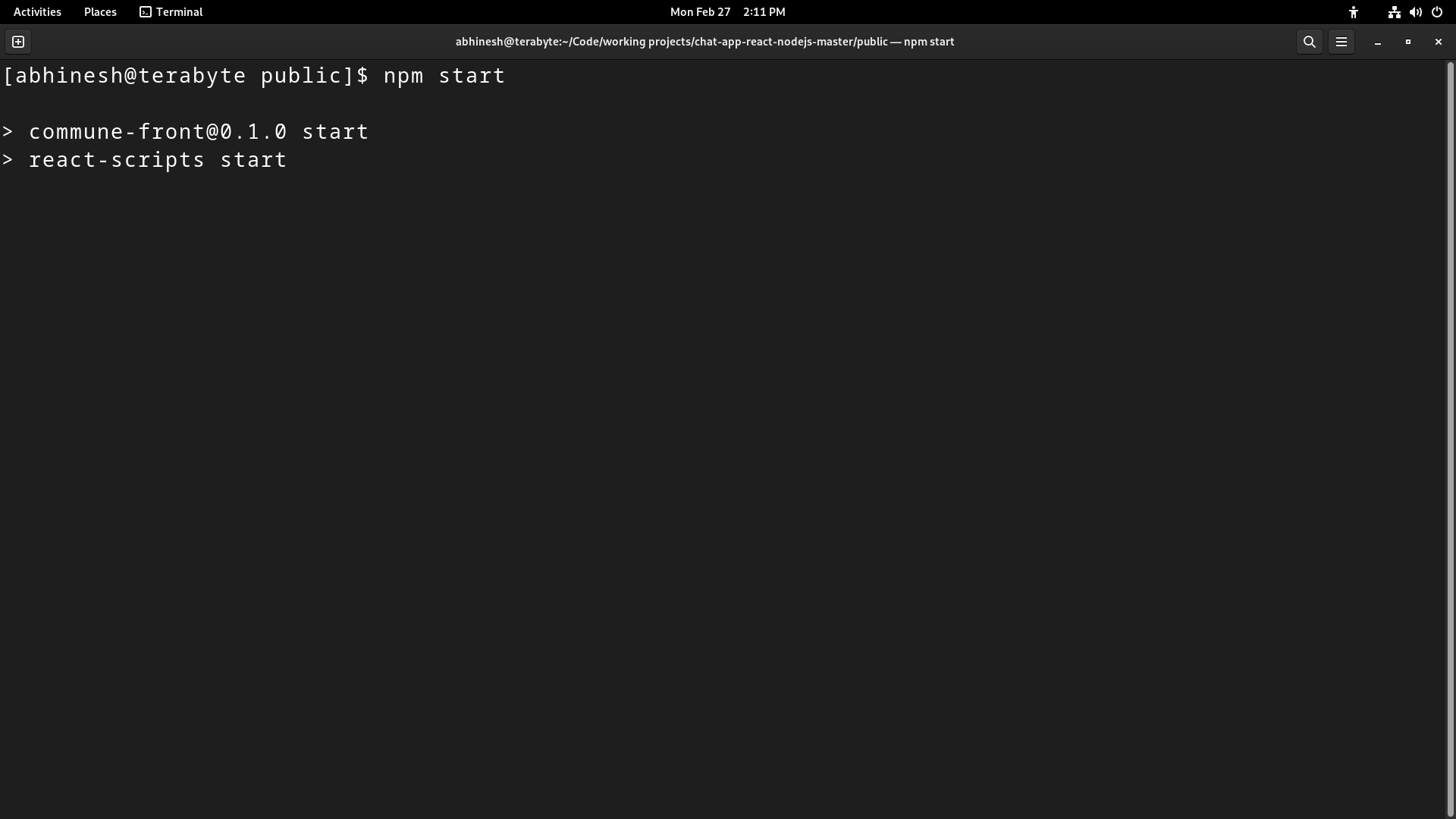
2. Enter the the ‘use chater’ command to specify the database name which makes the port directory “mongodn://127.0.0.1:27017/chater”

****

56

***Staring the Frontend server:***

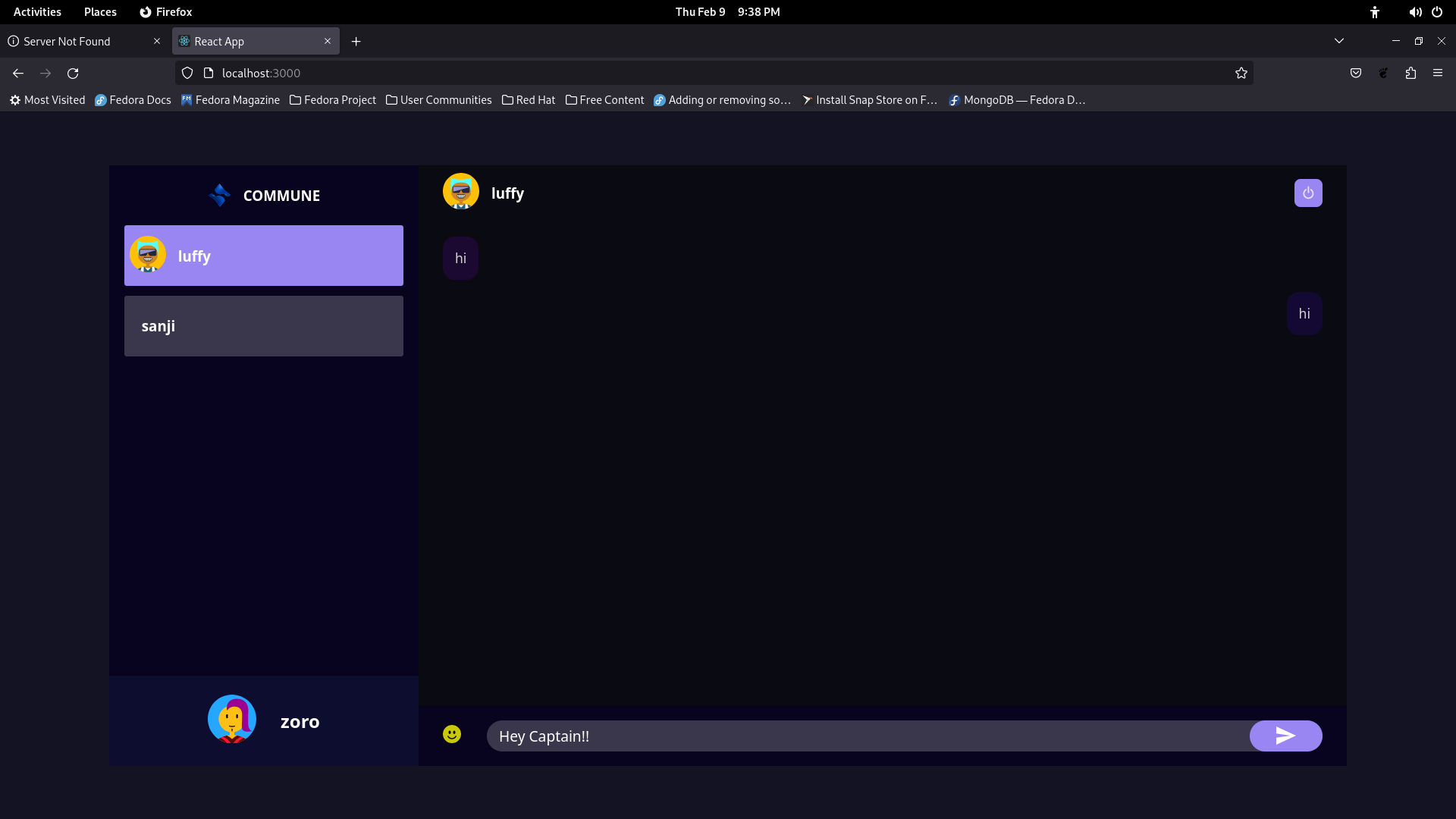
To start the front-end server run the ‘npm start’ command in the root directory of the frontend server which will start the react client on the browser at port “localhost://3000”.

****

57

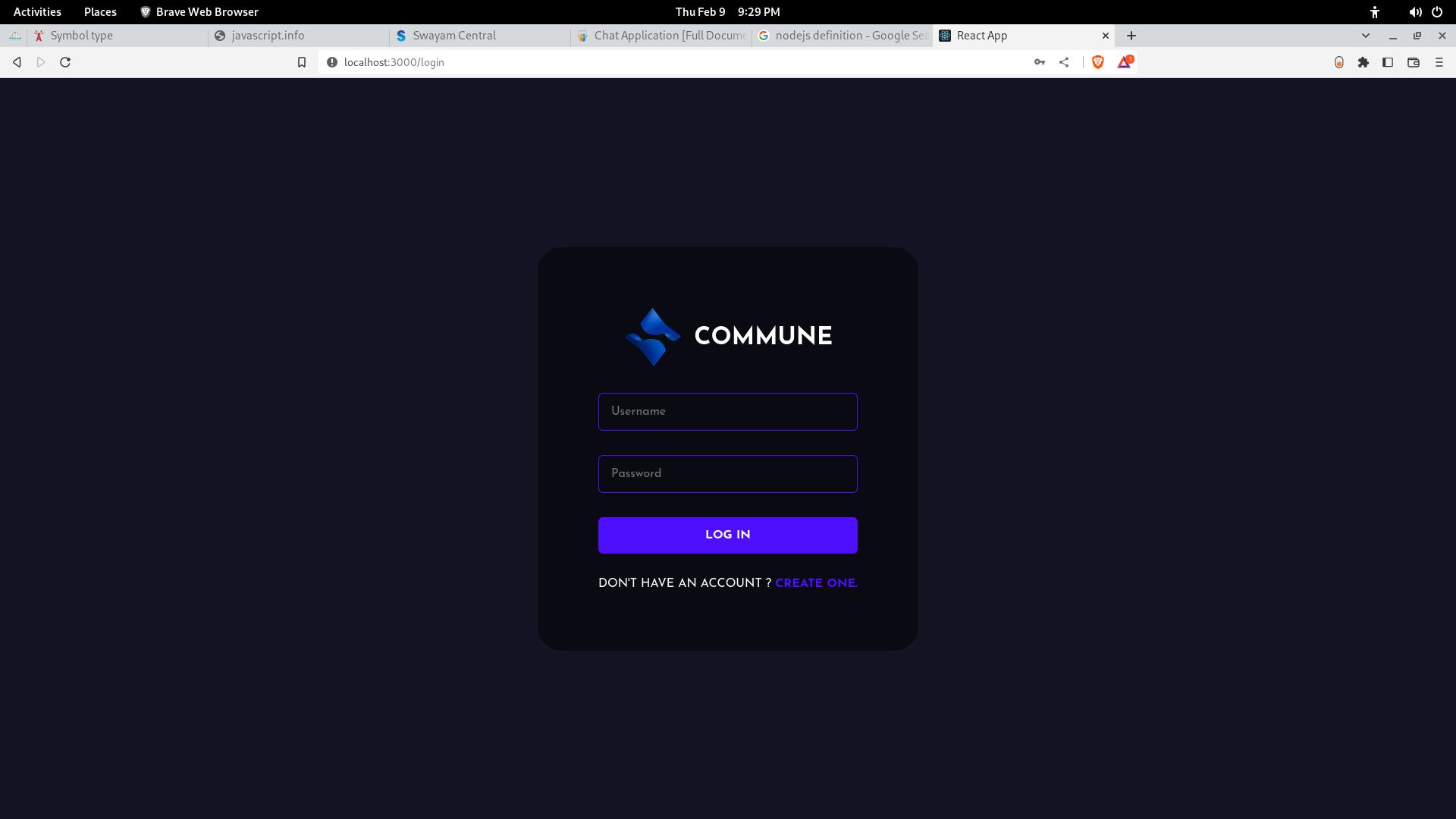
**4.4. PROJECT SNAPSHOTS**

*User’s Dashboard:*

**

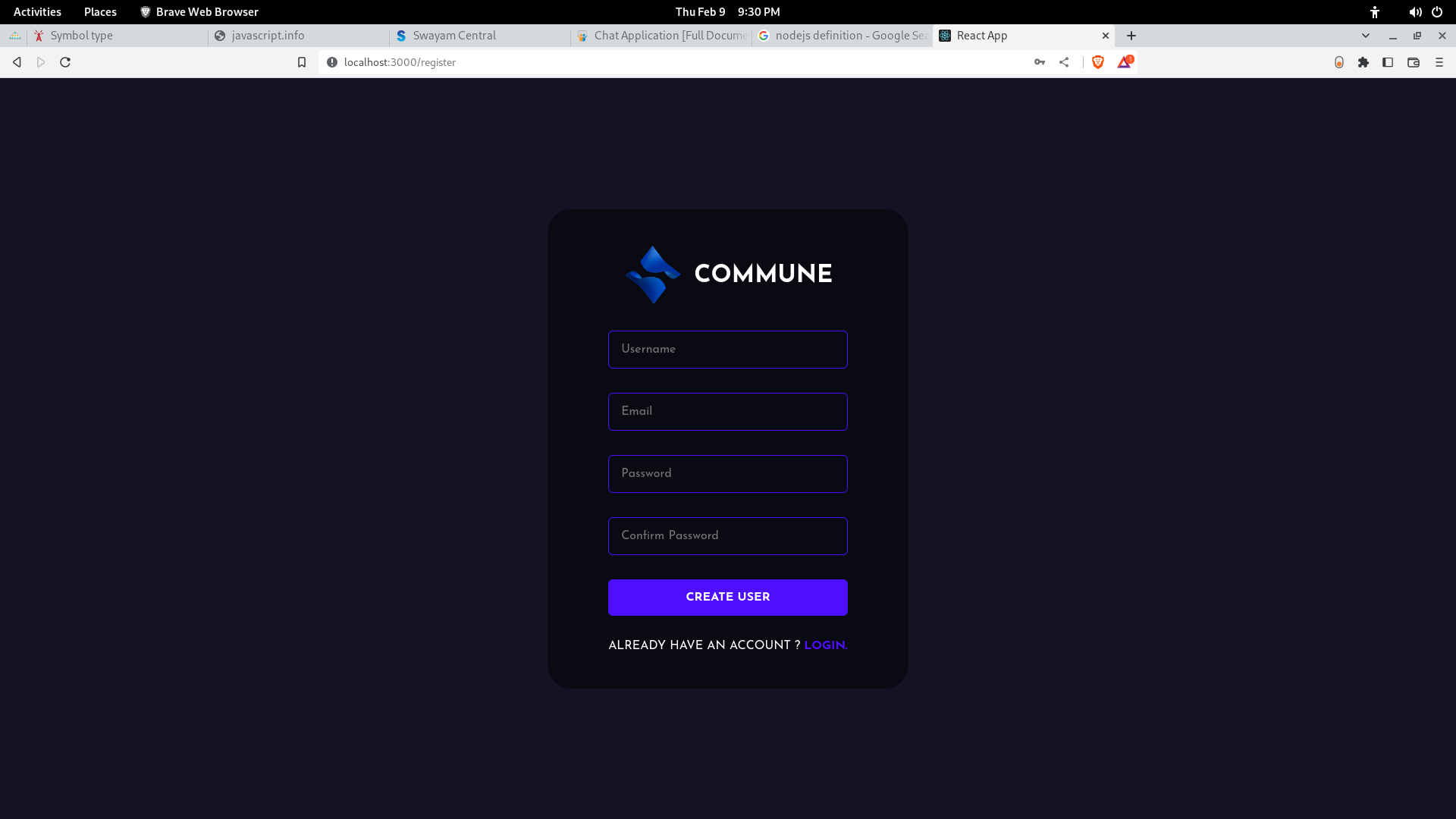
58

*Login Page:*



59

Register Page:

****

60

**5. CONCLUSION**

our chat application provides a simple and user-friendly interface for users to communicate with each other in real-time. It is built using a combination of technologies, including MongoDB, Express.js, React, and Node.js (MERN stack), and allows users to create accounts and send messages to other users.

Throughout this documentation, we have provided a detailed description of the application's features, as well as instructions for installing, configuring, and running the application on a local machine. We have also discussed the architecture of the application, including its server-side and client-side components, and provided an overview of the technologies used.

Overall, our chat application is a reliable and efficient tool for real-time communication, and can be customized and extended to meet the specific needs of users. In future we may be extending to include feature such as:

1. Group Chatting
2. File Transfering
3. Audio messages
4. Audio calls. Etc.

61

**6. REFERENCES**

[1] Marijn Haverbeke - “Eloquent JavaScript Third Edition .

[2] <https://www.w3schools.com/REACT/DEFAULT.ASP>

[3] <https://www.tutorialspoint.com/expressjs/index.htm>

[4] [https://www.w3schools.com/javascript](https://www.s3schools.com/javascript)

[5] <https://www.w3schools.com/mongodb>

62