

**INTERNSHIP REPORT** on  
**FULL STACK DEVELOPMENT WITH MERN**

A REPORT SUBMITTED TO JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY,  
KAKINADA IN PARTIAL FULFILLMENT OF AWARD OF DEGREE

**BACHELOR OF TECHNOLOGY**  
**IN**  
**ELECTRICAL AND ELECTRONICS ENGINEERING**



SUBMITTED BY

TOMPALA CHARAN KUMAR	(21HU5A0208)
INTURI SRAVANTHI	(21HU5A0204)
BIRUDU DHEERAJ	(21HU5A0202)
AREMANDA GOPICHANDU	(21HU5A0201)

UNDER ESTEEMED GUIDENCE OF

**Mr. K. MANOJ PAVAN KUMAR** M.TECH, Asst.professor

**CHEBROLU ENGINEERING COLLEGE**

(Approved by AICTE New Delhi, Affiliated to JNTU-Kakinada)

ACCREDITED NAAC 'A' GRADE,UGC-AUTONOMOUS

Door No:2B, Main Road, Chebrolu, Chebrolu, Guntur

522212

# CHEBROLU ENGINEERING COLLEGE

(Approved by AICTE, New Delhi & Affiliated to JNTU Kakinada, A.P)

CHEBROLU (Village & Mandal), GUNTUR-522212



## CERTIFICATE

This is to certify that the Internship report entitled “**FULL STACK DEVELOPMENT (MERN)**” is a piece of project work done by **TOMPALA CHARAN KUMAR (21HU5A0208)**, **INTURI SRAVANTHI (21HU5A0204)**, **BIRUDU DHEERAJ (21HU5A0202)** and **AREMANDA GOPICHANDU (21HU5A0201)**. Under the guidance of **Mr. K. MANOJ PAVAN KUMAR** M.Tech, Asst.professor. at the **DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING** for the degree of Bachelor of Technology of JNTU- KAKINADA, AP,INDIA.

The project viva-voice Exam is held on \_\_\_\_\_ of \_\_\_\_\_, 2023/2024.

Project Guide

Head of the Department

External Exam

## DECLARATION

We hereby declare that the project report entitled “**FULL STACK DEVELOPMENT (MERN)**” with reference to JNTU KAKINADA is carried out by us under the guidance of **Mr. K. MANOJ PAVAN KUMAR** M. Tech , **Asst.professor**. We also declare that this project report is a result of our own effort and were not submitted to any other completion of bachelor of technology in **ELECTRICAL AND ELECTRONICS ENGINEERING**.

## SIGNATURES

TOMPALA CHARAN KUMAR	(21HU5A0208)
INTURI SRAVANTHI	(21HU5A0204)
BIRUDU DHEERAJ	(21HU5A0202)
AREMANDA GOPI CHANDU	(21HU5A0201)

# ACKNOWLEDGEMENT

We express our profound gratitude to the principal **Dr.R.V. KRISHNAIAHM.Tech, PhD** & to the Director **M. SRINIVAS, MSC, M.Phil.** for holistic support in project work.

We would like to express our sincere thanks to **CH. HARI BABU<sub>M.Tech</sub>**, Head of the department of **ELECTRICAL AND ELECTRONICS ENGINEERING** for support in project work.

We express our gratitude and acknowledgement our deep indebtedness to our project guide **Mr. K. MANOJ PAVAN KUMAR<sub>M. Tech, Asst.professor</sub>**. Her valuable suggestions and guidance helped us a lot in completing and presenting on “**FULL STACK DEVELOPMENT (MERN)**”.

Finally, we wish to express our whole gratitude to our **PARENTS** and all our **FRIENDS** without whose encouragement would not have been able to complete this work.

## WITH REGARDS

TOMPALA CHARAN KUMAR	(21HU5A0208)
INTURI SRAVANTHI	(21HU5A0204)
BIRUDU DHEERAJ	(21HU5A0202)
AREMANDA GOPI CHANDU	(21HU5A0201)

## **ABSTRACT**

This abstract explores the role of full stack developers in the dynamic landscape of social media applications. With the exponential growth of social media platforms, the demand for versatile developers capable of handling both front-end and back-end development has surged. Full stack developers are integral in crafting seamless user experiences while ensuring robust server-side functionalities.

This paper delves into the essential skills and technologies required for full stack development, including proficiency in languages such as JavaScript, Python, and SQL, as well as frameworks like React and Django. Additionally, it examines the unique challenges faced by full stack developers in the realm of social media, such as scalability, security, and real-time data processing.

By employing agile methodologies and continuous integration practices, full stack developers contribute to the rapid iteration and evolution of social media applications, enabling them to stay ahead in a competitive market. Through case studies and industry insights, this abstract sheds light on the pivotal role of full stack developers in shaping the future of social media platforms.

# Social Media App

## Introduction:

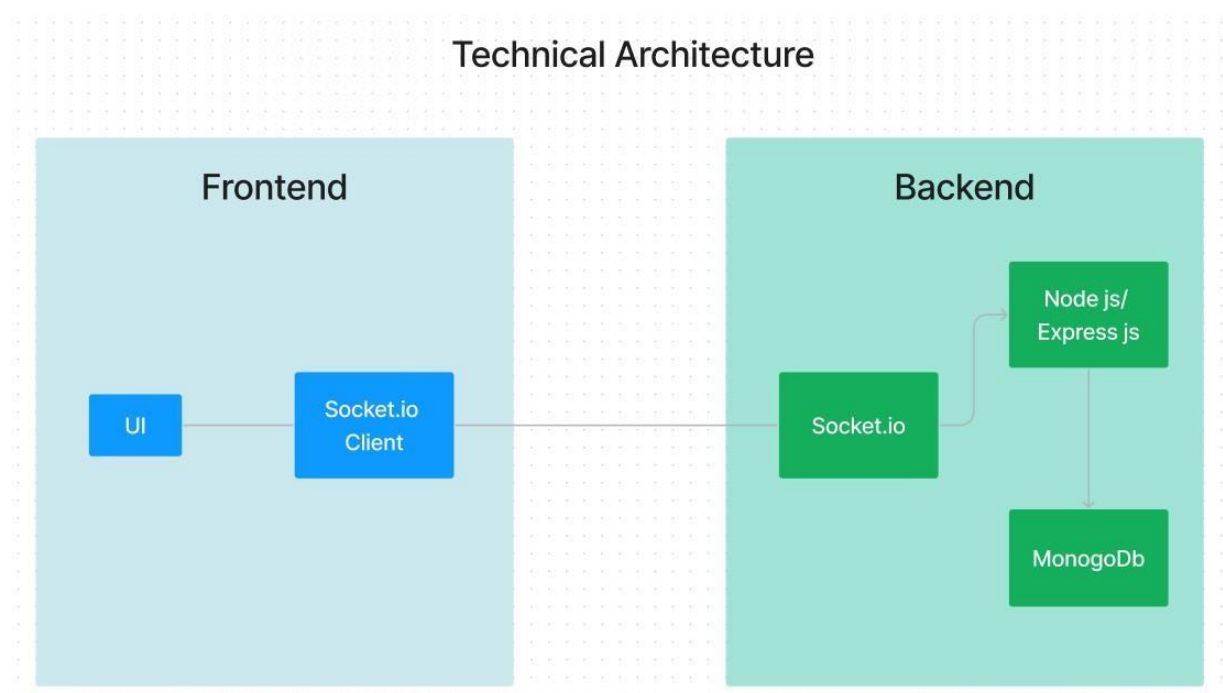
Introducing our revolutionary social media app! Designed to redefine online connections, our platform offers an intuitive and innovative space for seamless communication and engagement.

Break down barriers and enhance collaboration with real-time messaging. Whether you're sharing ideas, collaborating on projects, or simply having fun conversations, our messaging feature allows for seamless interaction and understanding.

Never miss a moment with our convenient post saving functionality. Capture important posts, articles, or inspiring content for later access and sharing with your followers. Your valuable discoveries are preserved, ensuring nothing gets lost in the vast social media landscape.

Your privacy and security are paramount. Our app employs robust encryption to safeguard your data, ensuring all communications remain confidential and protected from unauthorized access.

Step into a new era of online connections and communication. Discover the unmatched convenience of seamless messaging, in-app notifications, stories— all within our gamechanging social media app. Connect, engage, and explore more together!



The technical architecture of our social media app follows a client-server model, with a REST API used for the initial client-server connection. The frontend serves as the client and incorporates `socket.io-client` for establishing real-time communication with the backend server. The backend utilizes `socket.io` and `Express.js` frameworks to handle server-side logic and facilitate real-time messaging, post uploading, story uploading, and more.

The frontend includes the user interface and presentation layer, as well as the `socket.io-client` for establishing a persistent socket connection with the server. This enables real-time bidirectional communication, allowing for instant updates and seamless interaction between users.

Authentication is handled through the REST API, which securely verifies user credentials and provides access tokens or session cookies for subsequent requests. Once authenticated, the client establishes a socket connection with the backend to enable real-time features.

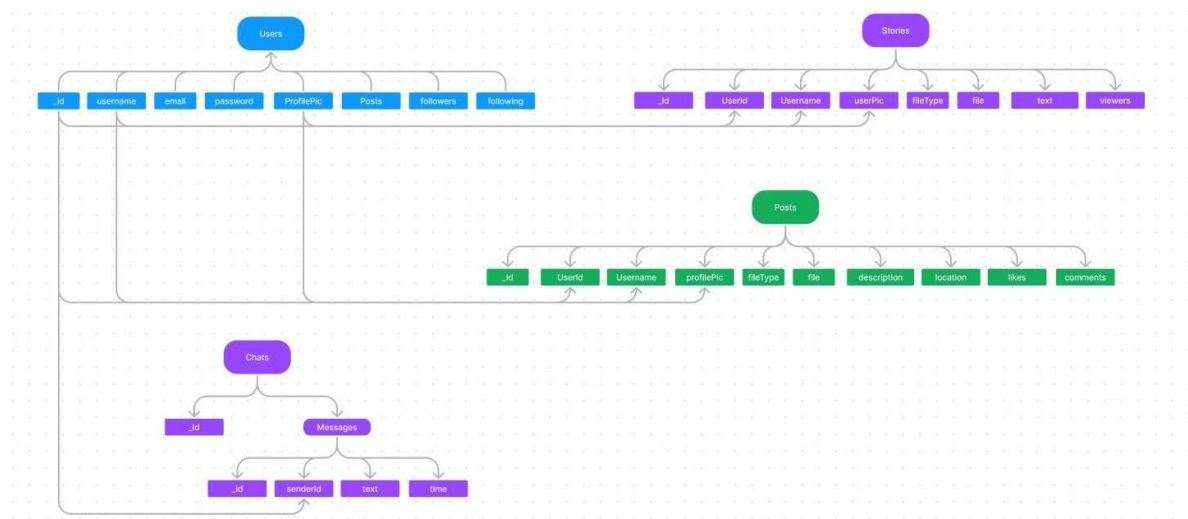
Real-time messaging is facilitated through the `socket.io` library, enabling instant exchange of messages between users. Users can engage in chats with their friends, sharing text, emojis, images, etc., in real-time.

Uploading posts and stories is also supported through the socket connection. Users can create and upload posts or stories, including text, images, videos, or a combination of media. The server receives and processes these uploads, ensuring they are associated with the correct user and made available for other users to view and interact with in real-time.

The backend utilizes `Express.js` to handle the REST API endpoints, routing requests to the appropriate controllers and services. User data, including profiles, posts, stories, and other relevant information, is stored and retrieved from a database such as `MongoDB`, ensuring efficient storage and retrieval of data.

Together, the frontend, backend, REST API, `socket.io`, `Express.js`, and database (e.g., `MongoDB`) form a comprehensive technical architecture for our social media app. This architecture enables real-time messaging, seamless post and story uploading, authentication, and secure data storage, providing users with a dynamic and interactive social media experience.

## ER Diagram:



In our social media app, the ER diagram showcases entities such as users, posts, and interactions. It illustrates how these entities relate to each other, helping us understand the underlying database structure and the flow of information within the app.

The ER diagram represents the relationship between users and posts, highlighting how users can create, share, and interact with posts within the app. It also captures the relationships between users and their followers, indicating the ability for users to follow and be followed by others.

Additionally, the diagram represents interactions such as likes, comments, etc., showcasing how users can engage with posts and interact with each other's content. These interactions contribute to the overall user experience and social engagement within the app.

By visualizing the relationships between entities, the ER diagram helps us understand the overall structure of the database and the interconnectedness of different components within the social media app. It provides a valuable tool for designing and optimizing the app's functionality and data management.

## Key features:

- **Real-time Updates:** Stay up to date with the latest activities and posts from your connections. Receive instant notifications for likes, comments, and mentions, ensuring you never miss out on important interactions.
- **Explore & Discover:** Explore a vast world of content and discover new ideas, trends, and communities. Engage with trending posts, discover new accounts, and connect with like-minded individuals.



- **Messaging and Chat:** Engage in private conversations and group chats with friends and followers. Share messages, emojis, photos, and videos, fostering real-time communication and connection.
- **Interactive Features:** Interact with posts through likes, comments, and shares. Express your thoughts, provide feedback, and engage in lively discussions with your network.
- **Follow and Connect:** Follow your favourite accounts and connect with influencers, brands, and individuals who inspire you. Build a vibrant network of connections and discover new opportunities.
- **Data privacy and Security:** We prioritize the protection of your personal information and data. Our app employs robust security measures, ensuring that your interactions, posts, and personal details remain secure and confidential.

These key features collectively enhance your social media experience, providing a dynamic and interactive platform for real-time communication, discovery, and connection with others.

## PRE-REQUISITES:

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, React.js, Socket.io:

- **Node.js and npm:**

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

- Download: <https://nodejs.org/en/download/>
- Installation instructions: <https://nodejs.org/en/download/package-manager/>

- **Express.js:**

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture. Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.

Installation: Open your command prompt or terminal and run the following command:

```
npm install express
```

## □ MongoDB:

MongoDB is a flexible and scalable NoSQL database that stores data in a JSONlike format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

## □ React.js:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

Follow the installation guide: <https://reactjs.org/docs/create-a-new-react-app.html>

## □ Socket.io:

Socket.io is a real-time bidirectional communication library that enables seamless communication between the server and clients. It allows for real-time data exchange, event-based messaging, and facilitates the development of real-time applications such as chat, collaboration, and gaming platforms.

Install Socket.io, a real-time bidirectional communication library for web applications.

Installation:

- Open your command prompt or terminal of server and run the following command: **npm install socket.io**
- Open your command prompt or terminal of client and run the following command: **npm install socket.io-client**

## □ HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

- **Database Connectivity:** Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations. To Connect the Database with Node JS go through the below provided link:
  - <https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/>
- **Front-end Framework:** Utilize Angular to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.
- **Version Control:** Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.
  - Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>
- **Development Environment:** Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.
  - Visual Studio Code: Download from <https://code.visualstudio.com/download>
  - Sublime Text: Download from <https://www.sublimetext.com/download>
  - WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

To run the existing Video Conference App project downloaded from GitHub:

Follow below steps:

- **Clone the Repository:**
    - Open your terminal or command prompt.
    - Navigate to the directory where you want to store the e-commerce app.
    - Execute the following command to clone the repository:  

```
git clone https://github.com/harsha-varadhan-reddy-07/SocialeX.git
```
- Install Dependencies:**
- Navigate into the cloned repository directory:

```
cd SocialeX
```

- Install the required dependencies by running the following commands:

```
cd client
```

```
npm install
```

```
cd ../server
```

```
npm install
```

#### □ **Start the Development Server:**

- To start the development server, execute the following command:

```
npm start
```

- The video conference app will be accessible at <http://localhost:3000>

#### □ **Access the App:**

- Open your web browser and navigate to <http://localhost:3000>.
- You should see the video conference app's homepage, indicating that the installation and setup were successful.

You have successfully installed and set up the e-commerce app on your local machine. You can now proceed with further customization, development, and testing as needed.

## **Roles & Responsibilities:**

### **User:**

- Create and manage a personal profile.
- Share posts, photos, videos, and stories with their network.
- Engage in conversations through comments, likes, and shares.
- Follow other users and discover new accounts, topics, and trends.
- Explore and discover new content, communities, and opportunities.
- Interact with notifications and stay updated with the activities of their connections.
- Utilize messaging and chat features to communicate with friends and followers.

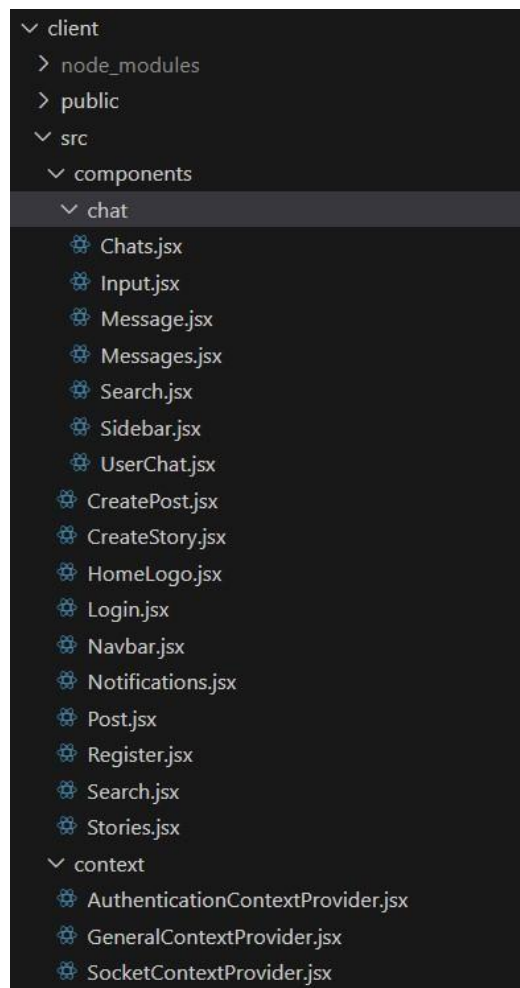
## Project structure:

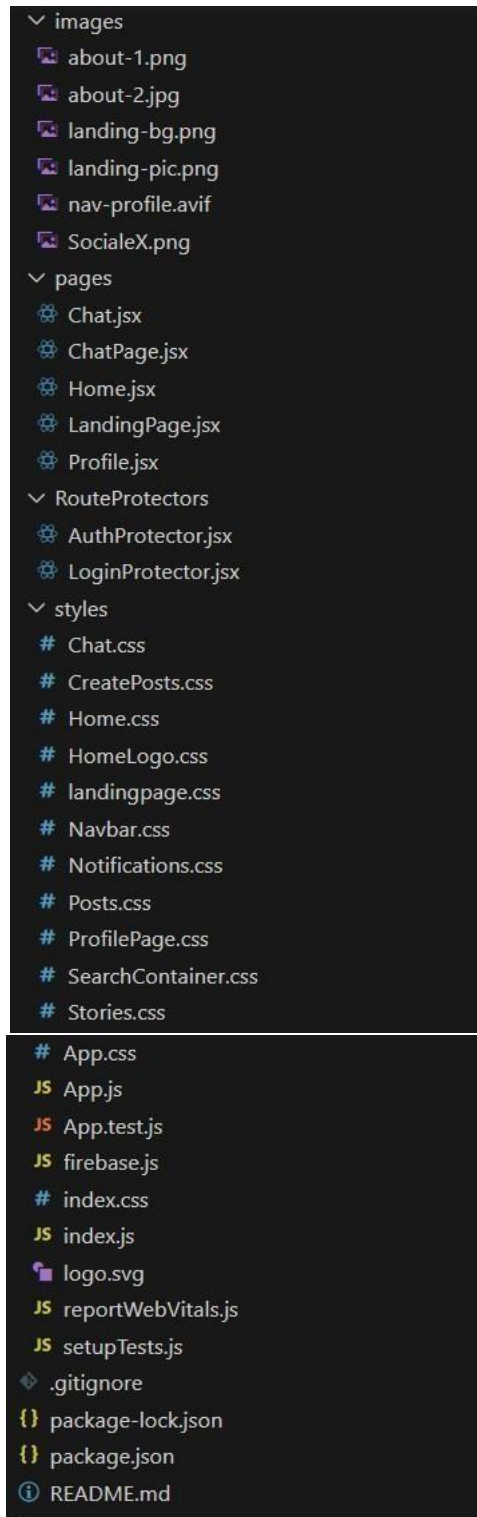
- Inside the SocialeX (social media app) directory, we have the following folders



- **Client directory:**

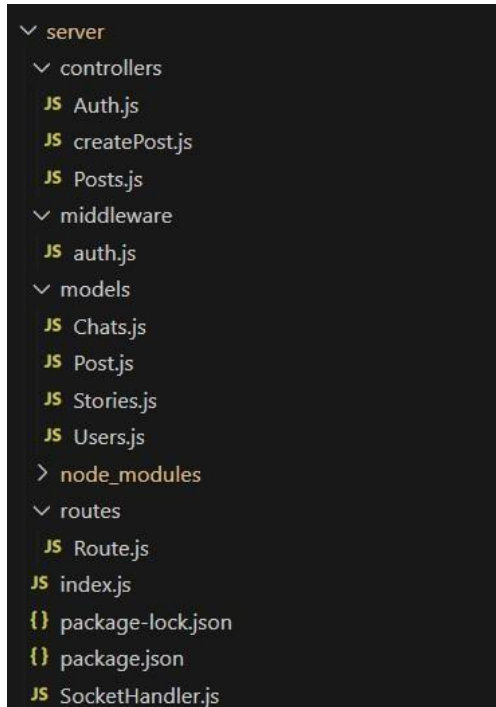
The below directory structure represents the directories and files in the client folder (front end) where, react Js is used along with Api's such as socket.io.





- **Server directory:**

The below directory structure represents the directories and files in the server folder (back end) where, node js, express js and mongodb are used along with socket.io.



## Project Flow:

### Project demo:

Before starting to work on this project, let's see the demo.

Demo link:

<https://drive.google.com/file/d/15JCHvQTNjOBfsxkzomK6s4YkhQBHE18o/view?usp=sharing>

Use the code in: <https://github.com/harsha-varadhan-reddy-07/SocialeX>

## Milestone 1: Project setup and configuration.

- **Folder setup:**

1. Create frontend and backend folders

- ✦ client
- ✦ server

- 

### Installation of required tools:

Open the frontend folder to install necessary tools. For frontend (client), we use:

Tools/libraries	Installation command
React Js	<code>npx create-react-app .</code>
Socket.io-client	<code>npm install socket.io-client</code>
Bootstrap	<code>npm install bootstrap</code>
Axios	<code>npm install axios</code>
Firebase	<code>npm install firebase</code>
uuid	<code>npm install uuid</code>

Open the backend folder to install necessary tools. For backend (server), we use:

Tools/libraries	Installation command
Express Js	<code>npm install express</code>
Mongoose	<code>npm install mongoose</code>
Bcrypt	<code>npm install bcrypt</code>
Body-parser	<code>npm install body-parser</code>
Cors	<code>npm install cors</code>
Dotenv	<code>npm install dotenv</code>
Http	<code>npm install http</code>
Socket.io	<code>npm install socket.io</code>

## Milestone 2: User Authentication & landing page

- **Setup express server**

1. Create index.js file in the server (backend folder).
2. Create a .env file and define port number to access it globally.
3. Configure the server by adding cors, body-parser.

- **Configure MongoDB**



- 

1. Import mongoose.
2. Add Database URL to the .env file.
3. Connect the database to the server.
4. Create a 'models' folder in the server to store all the DB models.

### **Develop Ui (landing & login)**

1. Develop the UI for landing page of the application.
2. Add the login & registration components to it or create new pages for the.
3. Collect the data from forms and send a request to backend along with that data.
4. Use axios to communicate with the server.

- **Add authentication in the server**

1. Create the "User" model for the MongoDB.
2. Create auth controller file to control the authentication actions.
3. Import "bcrypt" – used to hash(encode) the password to make it secure.
4. Define registration & login activities in the server.
5. Using Axios library, make request from the frontend.
6. Configure frontend & backend for authentication and store authenticated data in Context API in frontend.
7. On successful authentication, redirect to the home page.

### **Milestone 3: Web application development**

- **Create socket.io connection**

1. After the successful authentication, establish a socket connection between the client and the server.
2. Use socket.io connection to update data on user events seamlessly.
3. Use socket.io in chat feature as it helps to retrieve data in real-time.

- **Add create post feature**

1. Allow the user to create a post (photo/video).

- 2. Upload the media file to firebase storage or any other cloud platform and store the data and file link in the MongoDB.
  3. Retrieve the posts and display to all the users.
- **Add profile management**
  1. Add a profile page for every individual user.
  2. Display the user details and posts created by the user.
  3. Allow user to update details such as profile pic, username and about.

#### **Add chat feature**

1. Create an in-app chat feature.
  2. Use socket.io for real-time updates.
  3. Allow users to share media files in the chat.
- **Add stories/feed**
    1. Stories became one of the popular features of a social media app nowadays.
    2. Create the UI to display the stories.
    3. Allow users to add stories.
    4. Delete stories automatically when the uploaded time reaches 24hrs.
    5. Display stories to the followers.
  - **Add notifications**
    1. Use notifications feature to notify about new followers or new chats.

### **Code Explanation:**

#### **1. Server setup:**

Firstly, let's setup the server (backend). In the index.js file, import the required libraries and tools.

```
JS index.js M X
server > JS index.js > ...
1  import express from 'express';
2  import bodyParser from 'body-parser';
3  import mongoose from 'mongoose';
4  import cors from 'cors';
5  import { Server } from 'socket.io';
6  import http from 'http';
7  import path from 'path';
8  import { fileURLToPath } from 'url';
9
10 import authRoutes from './routes/Route.js';
11 import SocketHandler from './SocketHandler.js';
12
13
```

After importing all the libraries, setup the server with express.js and add “cors” as the middleware. Also define the socket connection for the future use. Here, the routes in the code below will be defined later. Also it’s important to connect the mongo database.

JS index.js M X

server > JS index.js > ...

```
13
14 // config
15 const __filename = fileURLToPath(import.meta.url);
16 const __dirname = path.dirname(__filename);
17
18 const app = express();
19
20 app.use(express.json());
21
22 app.use(bodyParser.json({limit: "30mb", extended: true}))
23 app.use(bodyParser.urlencoded({limit: "30mb", extended: true}));
24 app.use(cors());
25
26
27 app.use('', authRoutes);
28
29 const server = http.createServer(app);
30
31 const io = new Server(server, {
32   cors: {
33     origin: '*',
34     methods: ['GET', 'POST', 'PUT', 'DELETE']
35   }
36 });
37
38 io.on("connection", (socket) =>{
39   console.log("User connected");
40
41   SocketHandler(socket);
42 })
43
44
45 // mongoose setup
46
47 const PORT = 6001;
48
49 mongoose.connect('mongodb://localhost:27017/socialeX', {
50   useNewUrlParser: true,
51   useUnifiedTopology: true,
52 })
53 .then(()=>{
54   server.listen(PORT, ()=>{
55     console.log(`Running @ ${PORT}`);
56   });
57 })
58 .catch((e)=> console.log(`Error in db connection ${e}`));
59
60
```

## 2. Create Database models:

Now let's define all the required models for database. Initially, let's create a models folder and add separate files for each model.

- Users model

```
JS Users.js X
server > models > JS Users.js > userSchema > profilePic
1  import mongoose from 'mongoose';
2
3  const userSchema = mongoose.Schema({
4    username: {
5      type: String,
6      require: true
7    },
8    email: {
9      type: String,
10     require: true,
11     unique: true
12   },
13   password: {
14     type: String,
15     require: true
16   },
17   profilePic: {
18     type: String
19   },
20   about: {
21     type: String
22   },
23   posts: {
24     type: Array
25   },
26   followers: {
27     type: Array
28   },
29   following: {
30     type: Array
31   }
32 });
33
34 const User = mongoose.model("users", userSchema);
35 export default User;
```

- Chats model

```
JS Chats.js X
server > models > JS Chats.js > ...
1  import mongoose from "mongoose";
2
3  const chatSchema = mongoose.Schema({
4    // _id = u1 _id + u2 _id (u1 < u2 - compare both and arrange in order)
5    _id: {
6      type: String,
7      require: true
8    },
9    messages: {
10     type: Array
11   }
12 });
13
14 const Chats = mongoose.model("chats", chatSchema);
15 export default Chats;
```

- Stories model

```
JS Stories.js X
server > models > JS Stories.js > [e] storySchema
1  import mongoose from 'mongoose';
2
3  const storySchema = new mongoose.Schema({
4    userId: {
5      type: String
6    },
7    username: {
8      type: String
9    },
10   userPic: {
11     type: String
12   },
13   fileType: {
14     type: String
15   },
16   file: {
17     type: String
18   },
19   text: {
20     type: String
21   },
22   viewers: {
23     type: Array
24   }
25 }, {timestamps: true});
26
27
28 const Stories = mongoose.model('stories', storySchema);
29 export default Stories;
```

- **Posts model**

```
JS Post.js X
server > models > JS Post.js > [e] postSchema
1  import mongoose from "mongoose";
2
3  const postSchema = mongoose.Schema({
4    userId: {
5      type: String
6    },
7    userName: {
8      type: String
9    },
10   userPic: {
11     type: String
12   },
13   fileType: {
14     type: String
15   },
16   file : {
17     type: String
18   },
19   description: {
20     type: String
21   },
22   location: {
23     type: String
24   },
25   likes: {
26     type: Array
27   },
28   comments: {
29     type: Array
30   }
31 }, {timestamps: true});
32
33 const Post = mongoose.model("posts", postSchema);
34 export default Post;
```

### 3. Authentication:

- **Backend:**

In the backend(server), let's define functions for registration and login

## Register:

```
JS Authjs M X
server > controllers > JS Authjs > login
1  import bcrypt from 'bcrypt';
2  import jwt from 'jsonwebtoken';
3  import Users from '../models/Users.js';
4
5
6
7
8  const generateToken =(id) =>{
9
10     const jwtSecret = 'thisIsTheSceretCodeForTheJWTToken';
11
12     return jwt.sign((id), jwtSecret, {
13       expiresIn: '30d',
14     })
15   }
16
17   export const register = async (req, res) =>{
18     try{
19
20       const {username, email, password, profilePic} = req.body;
21
22       const salt = await bcrypt.genSalt();
23       const passwordHash = await bcrypt.hash(password, salt);
24
25       const newUser = new Users({
26         username,
27         email,
28         password: passwordHash,
29         profilePic
30       });
31
32       const user = await newUser.save();
33
34       // generate jwt token using function we defined at top of the page
35       const token = generateToken(user._id);
36
37       const userData = { _id: user._id, username: user.username,
38                         email:user.email, profilePic:user.profilePic,
39                         about: user.about, posts: user.posts,
40                         followers: user.followers, following:user.following };
41
42       res.status(200).json({token, user:userData});
43
44     }catch(err){
45       res.status(500).json({error: err.message});
46     }
47   };
48
```

## Login:

```
JS Authjs M X
server > controllers > JS Authjs > login
48
49   export const login = async (req, res) =>{
50     try{
51       const {email, password} = req.body;
52       const user = await Users.findOne({email:email});
53       if(!user) return res.status(400).json({msg: "User does not exist"});
54
55       const isMatch = await bcrypt.compare(password, user.password);
56       if(!isMatch) return res.status(400).json({msg: "Invalid credentials"});
57
58       // generate jwt token using function we defined at top of the page
59       const token = generateToken(user._id);
60       delete user.password;
61       const userData = { _id: user._id, username: user.username, email:user.email,
62                         profilePic:user.profilePic, about: user.about, posts: user.posts,
63                         followers: user.followers, following:user.following };
64
65       res.status(200).json({token, user:userData});
66       console.log(token, userData);
67     }catch(err){
68       res.status(500).json({error: err.message});
69     }
70   };
71
```



## Frontend:

In the frontend, first we need to create UI for the authentication and the with the data collected in the auth for, we need to perform actions accordingly. In our case, we used separate components for login, register. Then we used context Api to store the authentication data

### Register UI:

```
Register.jsx X
client > src > components > Register.jsx > (0) default
1  import React, { useContext } from 'react'
2  import { AuthenticationContext } from '../context/AuthenticationContextProvider'
3
4  const Register = ({ setIsLoginBox }) => {
5
6      const { setUsername, setEmail, setPassword, register } = useContext(AuthenticationContext);
7
8      const handleRegister = async (e) => {
9          e.preventDefault();
10
11          await register()
12      }
13
14      return (
15          <form className="authForm">
16              <h2>Register</h2>
17              <div className="form-floating mb-3 authFormInputs">
18                  <input type="text" className="form-control" id="floatingInput" placeholder="username" onChange={(e) => setUsername(e.target.value)} />
19                  <label htmlFor="floatingInput">Username</label>
20              </div>
21              <div className="form-floating mb-3 authFormInputs">
22                  <input type="email" className="form-control" id="floatingEmail" placeholder="name@example.com" onChange={(e) => setEmail(e.target.value)} />
23                  <label htmlFor="floatingInput">Email address</label>
24              </div>
25              <div className="form-floating mb-3 authFormInputs">
26                  <input type="password" className="form-control" id="floatingPassword" placeholder="Password" onChange={(e) => setPassword(e.target.value)} />
27                  <label htmlFor="floatingPassword">Password</label>
28              </div>
29              <button className="btn btn-primary" onClick={handleRegister}>Sign up</button>
30
31              <p>Already registered? <span onClick={() => setIsLoginBox(true)}>Login</span></p>
32          </form>
33      )
34  }
35
36  export default Register
```

### Login UI:

```
Login.jsx X
client > src > components > Login.jsx > (0) Login
1
2  import React, { useContext } from 'react'
3  import { AuthenticationContext } from '../context/AuthenticationContextProvider';
4
5  const Login = ({ setIsLoginBox }) => {
6
7      const { setEmail, setPassword, login } = useContext(AuthenticationContext);
8
9      const handleLogin = async (e) => {
10          e.preventDefault();
11          await login();
12      }
13
14      return (
15          <form className="authForm">
16              <h2>Login</h2>
17              <div className="form-floating mb-3 authFormInputs">
18                  <input type="email" className="form-control" id="floatingInput" placeholder="name@example.com" onChange={(e) => setEmail(e.target.value)} />
19                  <label htmlFor="floatingInput">Email address</label>
20              </div>
21              <div className="form-floating mb-3 authFormInputs">
22                  <input type="password" className="form-control" id="floatingPassword" placeholder="Password" onChange={(e) => setPassword(e.target.value)} />
23                  <label htmlFor="floatingPassword">Password</label>
24              </div>
25              <button type="submit" className="btn btn-primary" onClick={handleLogin}>Sign in</button>
26
27              <p>Not registered? <span onClick={() => setIsLoginBox(false)}>Register</span></p>
28          </form>
29      )
30  }
31
32  export default Login
33
```

## Context Api for Authentication:

```
AuthenticationContextProvider.jsx M X
client > src > context > AuthenticationContextProvider.jsx > [0] AuthenticationContextProvider
1  import React, { createContext, useState } from 'react';
2  import axios from "axios";
3  import { useNavigate } from "react-router-dom";
4
5  export const AuthenticationContext = createContext();
6
7  const AuthenticationContextProvider = ({children}) => {
8
9      const [username, setUsername] = useState('');
10     const [email, setEmail] = useState('');
11     const [password, setPassword] = useState('');
12
13     // const profilePic = 'https://images.unsplash.com/photo-1593085512500-5d55148d6f0d?ixlib=rb-4.0.
14
15     const profilePic = '';
16
17     const inputs = {username: username, email: email, password: password, profilePic: profilePic};
18
19
20     const navigate = useNavigate();
21
```

```
AuthenticationContextProvider.jsx M X
client > src > context > AuthenticationContextProvider.jsx > [0] AuthenticationContextProvider
19
20     const navigate = useNavigate();
21
22     const login = async () =>{
23
24         try{
25
26             const loginInputs = {email: email, password: password}
27             await axios.post('http://localhost:6001/login', loginInputs)
28             .then( async (res)=>{
29                 console.log("holaads",res);
30                 localStorage.setItem('userToken', res.data.token);
31                 localStorage.setItem('userId', res.data.user._id);
32                 localStorage.setItem('username', res.data.user.username);
33                 localStorage.setItem('email', res.data.user.email);
34                 localStorage.setItem('profilePic', res.data.user.profilePic);
35                 localStorage.setItem('posts', res.data.user.posts);
36                 localStorage.setItem('followers', res.data.user.followers);
37                 localStorage.setItem('following', res.data.user.following);
38                 navigate('/');
39             }).catch((err) =>{
40                 console.log(err);
41             });
42
43         }catch(err){
44             console.log(err);
45         }
46     }
47
48     const register = async () =>{
49
50         try{
51             await axios.post('http://localhost:6001/register', inputs)
52             .then( async (res)=>{
53                 localStorage.setItem('userToken', res.data.token);
54                 localStorage.setItem('userId', res.data.user._id);
55                 localStorage.setItem('username', res.data.user.username);
56                 localStorage.setItem('email', res.data.user.email);
57                 localStorage.setItem('profilePic', res.data.user.profilePic);
58                 localStorage.setItem('posts', res.data.user.posts);
59                 localStorage.setItem('followers', res.data.user.followers);
60                 localStorage.setItem('following', res.data.user.following);
61                 navigate('/');
62             }).catch((err) =>{
63                 console.log(err);
64             });
65
66         }catch(err){
67             console.log(err);
68         }
69     }
70
```

```

AuthenticationContextProvider.jsx M X
client > src > context > AuthenticationContextProvider.jsx > AuthenticationContextProvider
71
72
73
74
75   const logout = async () =>{
76     for (let key in localStorage) {
77       if (localStorage.hasOwnProperty(key)) {
78         localStorage.removeItem(key);
79       }
80     }
81     navigate('/landing');
82   }
83
84
85
86   return (
87     <AuthenticationContext.Provider value={{login, register, logout, username, setUsername, email, setEmail, password, setPassword}} >{children}</AuthenticationContext.Provider>
88   )
89 }
90
91 export default AuthenticationContextProvider

```

#### 4. Set Socket client:

After successful authentication, redirect to the home page. Along with that, we establish a socket connection at client side. Now let's create a general context file to pass required info to all the children files.

General context:

```

GeneralContextProvider.jsx M X
client > src > context > GeneralContextProvider.jsx > GeneralContextProvider
1   import React, { createContext, useReducer, useState } from 'react'
2   import socketIoClient from 'socket.io-client';
3
4   export const GeneralContext = createContext();
5
6
7   const WS = 'http://localhost:6001';
8
9   const socket = socketIoClient(WS);
10
11  export const GeneralContextProvider = ({children}) => {
12
13    const [isCreatePostOpen, setIsCreatePostOpen] = useState(false);
14    const [isCreateStoryOpen, setIsCreateStoryOpen] = useState(false);
15    const [isNotificationsOpen, setIsNotificationsOpen] = useState(false);
16
17    const [notifications, setNotifications] = useState([]);
18
19    const [chatFirends, setChatFriends] = useState([]);
20
21    const INITIAL_STATE = {
22      chatId: 'null',
23      user: {},
24    };
25  };
26
27  const userId = localStorage.getItem('userId');
28
29  const chatReducer = (state, action) => {
30    switch (action.type) {
31      case "CHANGE_USER":
32        return {
33          user: action.payload,
34          chatId: userId > action.payload._id ? userId + action.payload._id : action.payload._id + userId
35        }
36      default:
37        return state;
38    }
39  };
40
41  const [state, dispatch] = useReducer(chatReducer, INITIAL_STATE);
42
43
44
45
46  return (
47    <GeneralContext.Provider value={{socket, isCreatPostOpen, setIsCreatePostOpen, isCreateStoryOpen,
48      setIsCreateStoryOpen, isNotificationsOpen, setNotificationsOpen, notifications,
49      setNotifications, chatFirends, setChatFriends, chatData:state, dispatch}}>
50      {children}
51    </GeneralContext.Provider>
52  )
53
54 }

```

## 5. Create Posts:

### Frontend:

Here, on creating posts, we use firebase storage to store the files in the cloud. So, first we get the uploaded file URL from firebase and then update it to the MongoDB.

```
client > src > components > CreatePost.jsx > handlePostUpload > uploadTask.on('state_changed') callback > then() callback
1 import React, { useContext, useState } from 'react'
2 import './styles/CreatePosts.css'
3 import { RxCross2 } from 'react-icons/rx'
4 import { GeneralContext } from '../context/GeneralContextProvider'
5 import axios from 'axios'
6 import { ref, uploadBytesResumable, getDownloadURL } from 'firebase/storage'
7 import { storage } from '../firebase.js'
8 import { v4 as uuidv4 } from 'uuid'
9
10 const CreatePost = () => {
11
12   const { setIsCreatePostOpen, setIsCreatePostOpen } = useContext(GeneralContext);
13   const [postType, setPostType] = useState('photo');
14   const [postDescription, setPostDescription] = useState('');
15   const [postLocation, setPostLocation] = useState('');
16   const [postFile, setPostFile] = useState(null);
17   const [uploadProgress, setUploadProgress] = useState();
18
19   if (uploadProgress === 100) {
20     setPostDescription('');
21     setPostLocation('');
22     setPostFile(null);
23     setIsCreatePostOpen(false);
24     setUploadProgress();
25   }
26   const handlePostUpload = async (e) => {
27     e.preventDefault();
28     const storageRef = ref(storage, uuidv4());
29     const uploadTask = uploadBytesResumable(storageRef, postFile);
30     uploadTask.on('state_changed',
31       (snapshot) => {
32         setUploadProgress((snapshot.bytesTransferred / snapshot.totalBytes) * 100);
33       },
34       (error) => {
35         console.log(error);
36       },
37       () => {
38         getDownloadURL(uploadTask.snapshot.ref).then(async (downloadURL) => {
39           console.log('File available at', downloadURL);
40           try {
41             const inputs = {
42               userId: localStorage.getItem('userId'),
43               username: localStorage.getItem('username'),
44               userPic: localStorage.getItem('profilePic'),
45               fileType: postType,
46               file: downloadURL,
47               description: postDescription,
48               location: postLocation,
49               comments: ["New user: " + "this is my first comment"]
50             };
51             await axios.post('http://localhost:6001/createPost', inputs)
52               .then(async (res) => {
53                 console.log(err);
54               })
55               .catch((err) => {
56                 console.log(err);
57               });
58           } catch (err) {
59             console.log(err);
60           }
61         });
62       }
63     );
64   };
65 }
```

```
client > src > components > CreatePost.jsx > handlePostUpload > uploadTask.on('state_changed') callback > then() callback
57
58 return (
59   <div className="createPostModalBg" style={isCreatePostOpen ? { display: 'contents' } : { display: 'none' }} >
60     <div className="createPostContainer">
61       <RxCross2 className="closeCreatePost" onClick={() => setIsCreatePostOpen(false)} />
62       <h2 className="createPostTitle">Create post</h2>
63       <hr className="createPostIn" />
64       <div className="createPostBody">
65         <form>
66           <select className="form-select" aria-label="Select Post Type" onChange={(e) => setPostType(e.target.value)} >
67             <option defaultValues="photo">Choose post type</option>
68             <option value="photo">Photo</option>
69             <option value="video">Video</option>
70           </select>
71           <div className="uploadBox">
72             <input type="file" name="PostFile" id="uploadPostFile" onChange={(e) => setPostFile(e.target.files[0])} />
73           </div>
74           <div className="form-floating mb-3 authFormInputs descriptionInput">
75             <input type="text" className="form-control descriptionInput" id="floatingDescription" placeholder="Description"
76               onChange={(e) => setPostDescription(e.target.value)} value={postDescription} />
77             <label htmlFor="floatingDescription">Description</label>
78           </div>
79           <div className="form-floating mb-3 authFormInputs postLocation">
80             <input type="text" className="form-control postLocation" id="floatingLocation" placeholder="Location"
81               onChange={(e) => setPostLocation(e.target.value)} value={postLocation} />
82             <label htmlFor="floatingLocation">Location</label>
83           </div>
84           <div>
85             <button disabled="Uploading..." {Math.round(uploadProgress)}%</button>
86             <button onClick={handlePostUpload}>Upload</button>
87           </div>
88         </form>
89       </div>
90     </div>
91   </div>
92 )
93
94 export default CreatePost
```



- **Backend:**

In the backend, we create a separate file in controllers folder and define a function to create new post.

```
JS createPost.js X
server > controllers > JS createPost.js > createPost
1 import Post from '../models/Post.js';
2
3 export const createPost = async (req, res) =>{
4   try{
5
6     const newPost = new Post(req.body);
7
8     const post = await newPost.save();
9
10  }catch(e){
11    res.status(500).json({error:e});
12  }
13 }
```

## 6. Display Posts:

Now we need to fetch all the posts from the database and display to users depending on the rules we define. In this case, posts will be displayed to all the users and a follow button will be displayed on the top of the post, if the user is not following the user of that post.

```
Post.jsx M X
client > src > components > Post.jsx > Post
1 import React, { useContext, useEffect, useState } from 'react';
2 import '../styles/Posts.css';
3 import { AiOutlineHeart, AiTwotoneHeart } from "react-icons/ai";
4 import { FaGlobeAmericas } from "react-icons/fa";
5 import { IoIosPersonAdd } from 'react-icons/io'
6 import axios from 'axios';
7 import { GeneralContext } from '../context/GeneralContextProvider';
8 import { useNavigate } from 'react-router-dom';
9
10 const Post = () => {
11
12   const navigate = useNavigate();
13   const {socket} = useContext(GeneralContext);
14   const [posts, setPosts] = useState([]);
15
16   useEffect(() => {
17     fetchPosts();
18   }, []);
19   const fetchPosts = async () => {
20     try {
21       const response = await axios.get('http://localhost:6001/fetchAllPosts');
22       const fetchedPosts = response.data;
23       setPosts(fetchedPosts);
24     } catch (error) {
25       console.error(error);
26     }
27   };
28
29   // Like
30
31   const handleLike = (userId, postId) =>{
32     socket.emit('postLiked', {userId, postId});
33   }
34   const handleUnlike = (userId, postId) =>{
35     socket.emit('postUnliked', {userId, postId});
36   }
37   useEffect(()=>{
38     socket.on("likeUpdated", ()=>{
39       // alert("liked");
40     })
41     socket.on('userFollowed', ({following})=>{
42       localStorage.setItem('following', following);
43     })
44   },[socket])
45
46   const handleFollow = async (userId) =>{
47     socket.emit('followUser', {ownId: localStorage.getItem('userId'), followingUserId: userId});
48   }
49
50   const [comment, setComment] = useState('');
51   const handleComment = (postId, username) =>{
52     socket.emit('makeComment', {postId, username, comment});
53   }
54 }
```

```

client > src > components > Post.jsx > Post > posts.map() callback
56
57 return (
58   <div className='postsContainer'>
59
60     {posts ? posts.map((post) => {
61
62       return(
63
64         <div className="Post" key={post._id}>
65
66           <div className="postTop">
67             <div className="postTopDetails">
68               <img src={post.userPic} alt="" className="userpic" />
69               <h3 className="usernameTop" onClick={() => navigate(`/profile/${post.userId}`)}>{post.userName}</h3>
70             </div>
71
72             {localStorage.getItem('following').includes(post.userId) || localStorage.getItem('userId') === post.userId ?
73
74               <></>
75               :
76               <IoIosPersonAdd style={{cursor: "pointer"}} id='addFriendInPost' onClick={() => handleFollow(post.userId)} />
77             }
78
79           </div>
80
81           { post.fileType === 'photo'?
82             <img src={post.file} className='posting' alt="" />
83             :
84             <video id="videoPlayer" className='posting' controls autoPlay muted>
85               <source src={post.file} />
86             </video>
87           }
88
89           <div className="postReact">
90             <div className="supliconcol">
91
92               {
93                 post.likes.includes(localStorage.getItem('userId')) ?
94                 <AiTwotoneHeart className='support reactbtn' onClick={() => handleUnlike(localStorage.getItem('userId'), post._id)} />
95                 :
96                 <AiOutlineHeart className='support reactbtn' onClick={() => handleLike(localStorage.getItem('userId'), post._id)} />
97               }
98
99               <label htmlFor="support" className='supportCount'>{post.likes.length}</label>
100             </div>
101             </* <BiCommentDetail className='comment reactbtn' /> */>
102             </* <FiSend className='share reactbtn' onClick={() => {handleShare(post)}} /> */>
103             <div className="placeiconcol">
104               <FaGlobeAmericas className='placeicon reactbtn' name='place' />
105               <label htmlFor="place" className='place'>{post.location}</label>
106             </div>
107           </div>
108

```

```
client > src > components > Post.jsx > @Post > posts.map() callback
```

```
108  
109  
110     <div className="detail">  
111       <div className="descdataWithBtn">  
112         <label htmlFor="username" className="desc labeldata" id="desc">  
113           <span style={{fontWeight: 'bold'}}>  
114             {post.userName}  
115           </span>  
116           &nbsp;    {post.description}  
117         </label>  
118       </div>  
119     <div className="commentsContainer">  
120       <div className="makeComment">  
121         <input type="text" placeholder="type something..." onChange={(e)=>setComment(e.target.value)}/>  
122         {comment.length === 0 ?  
123           <button className="btn btn-primary" disabled>comment</button>  
124           :  
125           <button className="btn btn-primary" onClick={()=>handleComment(post._id, localStorage.getItem('username'))} >comment</button>  
126         }  
127       </div>  
128       <div className="commentsBody">  
129         <div className="comments">  
130           {post.comments.map((comment)=>{  
131             return(  
132               <p><b>{comment[0]}</b> {comment[1]}</p>  
133             )  
134           })}  
135         </div>  
136       </div>  
137     </div>  
138   </div>  
139 </div>  
140 )  
141  
142 }} : <></>  
143  
144 </div>  
145 }  
146  
147  
148 export default Post
```

## 7. Socket Handling in backend:

Let's create a file to handle the socket actions in the backend.

```
JS SocketHandler.js X
server > JS SocketHandler.js > SocketHandler > socket.on('story-played') callback
1
2 import Chats from './models/Chats.js';
3 import Post from './models/Post.js';
4 import Stories from './models/Stories.js';
5 import User from './models/Users.js';
6
7 const SocketHandler = (socket) => {
8
9   socket.on('postLiked', async ({userId, postId}) =>{
10     await Post.updateOne({_id: postId}, {$addToSet: {likes: userId}});
11     socket.emit("likeUpdated");
12   })
13
14   socket.on('postUnLiked', async ({userId, postId}) =>{
15     await Post.updateOne({_id: postId}, {$pull: {likes: userId}});
16     socket.emit("likeUpdated");
17   })
18
19   socket.on('fetch-profile', async ({_id})=>{
20     const user = await User.findOne({_id})
21     console.log(user);
22     socket.emit("profile-fetched", {profile: user})
23   })
24
25
26
27   socket.on('updateProfile', async ({userId, profilePic, username, about})=>{
28     const user = await User.updateOne({_id: userId}, {profilePic: profilePic, username: username, about:about})
29     socket.emit("profile-fetched", {profile: user})
30   })
31
32   socket.on('user-search', async({username})=>{
33     const user = await User.findOne({username:username});
34     socket.emit('searched-user', {user});
35   })
36
37   socket.on('followUser', async({ownId, followingUserId})=>{
38     await User.updateOne({_id: ownId}, {$addToSet: {following: followingUserId}});
39     await User.updateOne({_id: followingUserId}, {$addToSet: {followers: ownId}});
40
41     const user1 = await User.findOne({_id: ownId});
42     const user2 = await User.findOne({_id: followingUserId});
43     socket.emit('userFollowed', {following: user1.following});
44
45     if ( user2.following.includes(user1._id) && user1.following.includes(user2._id) ){
46       const newChat = new Chats({
47         _id: user1._id > user2._id ? user1._id + user2._id : user2._id + user1._id
48       })
49
50       const chat = await newChat.save();
51     }
52
53   });
54
```

```
JS SocketHandler.js X
server > JS SocketHandler.js > SocketHandler > socket.on('story-played') callback
54
55   socket.on('unFollowUser', async({ownId, followingUserId})=>{
56     await User.updateOne({_id: ownId}, {$pull: {following: followingUserId}});
57     await User.updateOne({_id: followingUserId}, {$pull: {followers: ownId}});
58
59     const user = await User.findOne({_id: ownId});
60     socket.emit('userUnFollowed', {following: user.following});
61   });
62
63
64   socket.on('makeComment', async({postId, username, comment})=>{
65     await Post.updateOne({_id: postId}, { $push: { comments: [ username, comment] } });
66   });
67
68   socket.on('fetch-friends', async ({userId}) =>{
69
70     const userData = await User.findOne({_id: userId})
71
72     function findCommonElements(array1, array2) {
73       return array1.filter(element => array2.includes(element));
74     }
75
76     const friendsList = findCommonElements(userData.following, userData.followers);
77
78     const friendsData = await User.find(
79       { _id: { $in: friendsList } },
80       { _id: 1, username: 1, profilePic: 1 }
81     ).exec();
82
83     socket.emit("friends-data-fetched", {friendsData});
84   })
85
```

```

JS SocketHandler.js X
server > JS SocketHandler.js > SocketHandler > socket.on('story-played') callback
86
87   socket.on('fetch-messages', async ({chatId}) =>{
88     const chat = await Chats.findOne({_id: chatId});
89
90     await socket.join(chatId);
91
92     await socket.emit('messages-updated', {chat: chat});
93
94   })
95
96   socket.on('update-messages', async ({ chatId }) => {
97     try {
98       const chat = await Chats.findOne({ _id: chatId });
99       console.log('updating messages');
100       socket.emit('messages-updated', { chat });
101     } catch (error) {
102       console.error('Error updating messages:', error);
103     }
104   });
105
106   socket.on('new-message', async ({ chatId, id, text, file, senderId, date }) => {
107     try {
108       await Chats.findOneAndUpdate(
109         { _id: chatId },
110         { $addToSet: { messages: { id, text, file, senderId, date } } },
111         { new: true }
112       );
113
114       const chat = await Chats.findOne({ _id: chatId });
115       console.log(chat);
116       socket.emit('messages-updated', { chat });
117       socket.broadcast.to(chatId).emit('message-from-user');
118     } catch (error) {
119       console.error('Error adding new message:', error);
120     }
121   });
122
123
124   socket.on('chat-user-searched', async ({ownId, username})=>{
125     const user = await User.findOne({username:username});
126     if(user){
127       if (user.followers.includes(ownId) && user.following.includes(ownId)){
128
129         socket.emit('searched-chat-user', {user});
130
131       }else{
132         socket.emit('no-searched-chat-user');
133       }
134     }else{
135       socket.emit('no-searched-chat-user');
136     }
137   });
138

```

```

JS SocketHandler.js X
server > JS SocketHandler.js > SocketHandler > socket.on('story-played') callback
139
140   socket.on('fetch-all-posts', async())=>{
141     const posts = await Post.find();
142     socket.emit('all-posts-fetched', {posts});
143   })
144
145
146   socket.on('delete-post', async ({postId}) =>{
147     await Post.deleteOne({_id: postId});
148     const posts = await Post.find();
149     socket.emit('post-deleted', {posts});
150   });
151
152
153   socket.on('create-new-story', async({userId, username, userPic, fileType, file, text})=>{
154     const newStory = new Stories({userId, username, userPic, fileType, file, text});
155     await newStory.save();
156   })
157
158   socket.on('fetch-stories', async())=>{
159     const stories = await Stories.find();
160     socket.emit('stories-fetched', {stories});
161   });
162
163   socket.on('story-played', async ({storyId, userId})=>{
164     await Stories.updateOne({_id: storyId}, {$addToSet: {viewers: userId}});
165   })
166
167 }
168
169 export default SocketHandler;

```



## 8. Fetch Posts:

As we used axios to fetch posts, let's define code for that. Along with fetching posts, we also included code for fetching stories.

```
JS Posts.js X
server > controllers > JS Posts.js > [⌘] fetchAllStories
1  import Post from '../models/Post.js';
2  import Stories from '../models/Stories.js';
3  import User from '../models/Users.js'
4
5  export const fetchAllPosts = async (req, res) =>{
6      try {
7          const posts = await Post.find().sort({ _id: -1 });
8
9          res.json(posts);
10     } catch (error) {
11         console.error(error);
12         res.status(500).json({ error: 'Server error' });
13     }
14 }
15
16 export const fetchUserName = async (req, res) =>{
17     try {
18         const userId = req.body.userId;
19         const user = await User.findById(userId);
20         console.log(userId);
21         res.status(200).json(user);
22     } catch (error) {
23         console.error(error);
24         res.status(500).json({ error: 'Server error' });
25     }
26 }
27
28 export const fetchUserImg = async (req, res) =>{
29     try {
30         const userId = req.body.userId;
31         const user = await User.findOne({ _id: userId });
32         console.log(userId);
33         res.status(200).json(user);
34     } catch (error) {
35         console.error(error);
36         res.status(500).json({ error: 'Server error' });
37     }
38 }
39
40 export const fetchAllStories = async (req, res) =>{
41     try {
42         const stories = await Stories.find();
43
44         res.status(200).json(stories);
45     } catch (error) {
46         console.error(error);
47         res.status(500).json({ error: 'Server error' });
48     }
49 }
```

## 9. Create Stories:

- **Frontend:**

Let's design the posts display UI and then we create a pop-up modal to create new story.

```

client > src > components > Stories.jsx > Stories
1 import React, { useContext, useEffect, useState } from 'react'
2 import '../styles/Stories.css'
3 import { BiPlusCircle } from 'react-icons/bi'
4 import { GeneralContext } from '../context/GeneralContextProvider';
5 import axios from 'axios';
6 import { RxCross2 } from 'react-icons/rx'
7
8 const Stories = () => {
9
10   const {socket, setIsCreateStoryOpen} = useContext(GeneralContext);
11
12   const [stories, setStories] = useState([])
13   const [isStoryPlaying, setIsStoryPlaying] = useState(false);
14
15   const [story, setStory] = useState();
16
17   const addStory = async () =>{
18     setIsCreateStoryOpen(true)
19   }
20
21   useEffect(() => {
22     fetchStories();
23   }, []);
24
25   const fetchStories = async () => {
26     try {
27       const response = await axios.get('http://localhost:6001/fetchAllStories');
28       setStories(response.data)
29       console.log(response.data[0])
30     } catch (error) {
31       console.error(error);
32     }
33   };
34
35   const handleOpenStory = async (story) =>{
36
37     setStory(story);
38     await socket.emit('story-played', {storyId: story._id, userId: localStorage.getItem('userId')});
39     setIsStoryPlaying(true);
40
41   }
42

```

```

43  return (
44    <div className="storiesContainer">
45      <div className="storiesTitle">
46        <h3>Stories</h3>
47      </div>
48      <div className="storiesBody" style={isStoryPlaying ? {display: 'none'} : {}}>
49        <div className="stories">
50          <div className="story self-story" onClick={addStory}>
51            <img src={localStorage.getItem('profilePic')} alt="" />
52            <p>Add story</p>
53            <span><BiPlusCircle /></span>
54          </div>
55          {
56            stories && stories.filter(story => ({localStorage.getItem('following').includes(story.userId)
57              || story.userId === localStorage.getItem('userId')
58              && (Math.abs(Math.round((new Date()).getTime() - new Date(story.createdAt).getTime()) / (1000 * 60 * 60))) < 24 ))
59            ).map((story) => (
60              <div className="story user-story" key={story_id}
61                onClick={() => handleOpenStory(story)}
62                style={story.viewers.includes(localStorage.getItem('userId'))
63                  ? {border: '3px solid #a5a7a995'}
64                  : {border: '3px solid #569bdcf9'}
65                } >
66                <img src={story.userPic} alt="" />
67                <p>{story.username}</p>
68              </div>
69            ))
70          }
71        </div>
72      </div>
73    </div>
74  )
75
76  </div>
77
78  </div>
79
80  </div>
81
82  </div>
83
84  </div>
85
86  </div>
87
88  </div>
89
90  </div>
91
92  </div>
93
94  </div>
95
96  </div>
97
98  </div>
99
100 </div>
101
102 </div>
103
104 </div>
105
106 </div>
107
108 </div>
109
110 </div>
111
112 </div>
113
114 </div>
115
116 </div>
117
118 </div>
119
120 </div>
121
122 </div>
123
124 </div>
125
126 </div>
127
128 </div>
129
130 </div>
131
132 </div>
133
134 </div>
135
136 </div>
137
138 </div>
139
140 </div>
141
142 </div>
143
144 </div>
145
146 </div>
147
148 </div>
149
150 </div>
151
152 </div>
153
154 </div>
155
156 </div>
157
158 </div>
159
160 </div>
161
162 </div>
163
164 </div>
165
166 </div>
167
168 </div>
169
170 </div>
171
172 </div>
173
174 </div>
175
176 </div>
177
178 </div>
179
180 </div>
181
182 </div>
183
184 </div>
185
186 </div>
187
188 </div>
189
190 </div>
191
192 </div>
193
194 </div>
195
196 </div>
197
198 </div>
199
200 </div>
201
202 </div>
203
204 </div>
205
206 </div>
207
208 </div>
209
210 </div>
211
212 </div>
213
214 </div>
215
216 </div>
217
218 </div>
219
220 </div>
221
222 </div>
223
224 </div>
225
226 </div>
227
228 </div>
229
230 </div>
231
232 </div>
233
234 </div>
235
236 </div>
237
238 </div>
239
240 </div>
241
242 </div>
243
244 </div>
245
246 </div>
247
248 </div>
249
250 </div>
251
252 </div>
253
254 </div>
255
256 </div>
257
258 </div>
259
260 </div>
261
262 </div>
263
264 </div>
265
266 </div>
267
268 </div>
269
270 </div>
271
272 </div>
273
274 </div>
275
276 </div>
277
278 </div>
279
280 </div>
281
282 </div>
283
284 </div>
285
286 </div>
287
288 </div>
289
290 </div>
291
292 </div>
293
294 </div>
295
296 </div>
297
298 </div>
299
300 </div>
301
302 </div>
303
304 </div>
305
306 </div>
307
308 </div>
309
310 </div>
311
312 </div>
313
314 </div>
315
316 </div>
317
318 </div>
319
320 </div>
321
322 </div>
323
324 </div>
325
326 </div>
327
328 </div>
329
330 </div>
331
332 </div>
333
334 </div>
335
336 </div>
337
338 </div>
339
340 </div>
341
342 </div>
343
344 </div>
345
346 </div>
347
348 </div>
349
350 </div>
351
352 </div>
353
354 </div>
355
356 </div>
357
358 </div>
359
360 </div>
361
362 </div>
363
364 </div>
365
366 </div>
367
368 </div>
369
370 </div>
371
372 </div>
373
374 </div>
375
376 </div>
377
378 </div>
379
380 </div>
381
382 </div>
383
384 </div>
385
386 </div>
387
388 </div>
389
390 </div>
391
392 </div>
393
394 </div>
395
396 </div>
397
398 </div>
399
400 </div>
401
402 </div>
403
404 </div>
405
406 </div>
407
408 </div>
409
410 </div>
411
412 </div>
413
414 </div>
415
416 </div>
417
418 </div>
419
420 </div>
421
422 </div>
423
424 </div>
425
426 </div>
427
428 </div>
429
430 </div>
431
432 </div>
433
434 </div>
435
436 </div>
437
438 </div>
439
440 </div>
441
442 </div>
443
444 </div>
445
446 </div>
447
448 </div>
449
450 </div>
451
452 </div>
453
454 </div>
455
456 </div>
457
458 </div>
459
460 </div>
461
462 </div>
463
464 </div>
465
466 </div>
467
468 </div>
469
470 </div>
471
472 </div>
473
474 </div>
475
476 </div>
477
478 </div>
479
480 </div>
481
482 </div>
483
484 </div>
485
486 </div>
487
488 </div>
489
490 </div>
491
492 </div>
493
494 </div>
495
496 </div>
497
498 </div>
499
500 </div>
501
502 </div>
503
504 </div>
505
506 </div>
507
508 </div>
509
510 </div>
511
512 </div>
513
514 </div>
515
516 </div>
517
518 </div>
519
520 </div>
521
522 </div>
523
524 </div>
525
526 </div>
527
528 </div>
529
530 </div>
531
532 </div>
533
534 </div>
535
536 </div>
537
538 </div>
539
540 </div>
541
542 </div>
543
544 </div>
545
546 </div>
547
548 </div>
549
550 </div>
551
552 </div>
553
554 </div>
555
556 </div>
557
558 </div>
559
560 </div>
561
562 </div>
563
564 </div>
565
566 </div>
567
568 </div>
569
570 </div>
571
572 </div>
573
574 </div>
575
576 </div>
577
578 </div>
579
580 </div>
581
582 </div>
583
584 </div>
585
586 </div>
587
588 </div>
589
```

```

Stories.jsx M X
client > src > components > Stories.jsx > Stories
83
84 {story &&
85
86   <div className="storyPlayContainer" style={isStoryPlaying ? {} : {display: 'none'}}>
87     <div className="storyPlayBodyTop">
88       <p>{story.username}</p>
89       <span onClick={()=>setIsStoryPlaying(false)}><RxCross2 /></span>
90     </div>
91     <div className="storyPlayBodyContent">
92       {story.fileType === 'photo' ?
93         <img src={story.file} alt="" />
94         :
95         <video id="videoPlayer" className='posting' controls autoPlay muted>
96           <source src={story.file} />
97         </video>
98       }
99       <p>{story.text}</p>
100     </div>
101   </div>
102 }
103
104
105
106 </div>
107 )
108 }
109
110 export default Stories

```

Create new story:

```

CreateStory.jsx M X
client > src > components > CreateStory.jsx > CreateStory
1 import React, { useContext, useState } from 'react';
2 import '../styles/CreatePosts.css'
3 import { GeneralContext } from '../context/GeneralContextProvider';
4 import { RxCross2 } from 'react-icons/rx';
5 import {ref, uploadBytesResumable, getDownloadURL} from "firebase/storage";
6 import {storage} from '../firebase.js';
7 import { v4 as uuidv4 } from 'uuid';
8
9 const CreateStory = () => {
10
11   const {socket, isCreateStoryOpen, setIsCreateStoryOpen} = useContext(GeneralContext);
12
13   const [storyType, setStoryType] = useState('photo');
14   const [storyDescription, setStoryDescription] = useState('');
15   const [storyFile, setStoryFile] = useState(null);
16
17   const [uploadProgress, setUploadProgress] = useState();
18
19   if (uploadProgress === 100){
20     setStoryDescription('');
21     setStoryFile(null);
22     setIsCreateStoryOpen(false);
23     setUploadProgress();
24   }
25

```

```

CreateStory.jsx M X
client > src > components > CreateStory.jsx > CreateStory > handleStoryUpload
27
28 const handleStoryUpload = async (e) =>{
29   e.preventDefault();
30   const storageRef = ref(storage, uuidv4());
31   const uploadTask = uploadBytesResumable(storageRef, storyFile);
32
33   uploadTask.on('state_changed',
34     (snapshot) => {
35       setUploadProgress((snapshot.bytesTransferred / snapshot.totalBytes) * 100);
36     },
37     (error) => {
38       console.log(error);
39     },
40     () => {
41       getDownloadURL(uploadTask.snapshot.ref).then( async (downloadURL) => {
42         console.log('File available at', downloadURL);
43
44         try{
45           await socket.emit('create-new-story', {userId: localStorage.getItem('userId'), username: localStorage.getItem('username'),
46             userPic: localStorage.getItem('profilePic'), fileType: storyType, file: downloadURL,
47             text: storyDescription});
48
49           setIsCreateStoryOpen(false);
50           setStoryDescription('');
51           setStoryFile(null);
52           setIsCreateStoryOpen(false);
53           setUploadProgress();
54         }catch(err){
55           console.log(err);
56         }
57       });
58     }
59   );
60 }

```

```

CreateStory.jsx M X
client > src > components > CreateStory.jsx > | CreateStory > | handleStoryUpload
57
58   return (
59     <div className="createPostModalBg" style={isCreateStoryOpen ? {display: 'contents'} : {display: 'none'}} >
60       <div className="createPostContainer">
61         <RxCross2 className="closeCreatePost" onClick={() => setIsCreateStoryOpen(false)} />
62         <h2 className="createPostTitle">Add new story</h2>
63         <hr className="createPostHr" />
64         <div className="createPostBody">
65           <form>
66             <select className="form-select" aria-label="Select Post Type" onChange={(e) => setStoryType(e.target.value)} >
67               <option defaultValue="photo">Choose post type</option>
68               <option value="photo">Photo</option>
69               <option value="video">Video</option>
70             </select>
71             <div className="uploadBox">
72               <input type="file" name="PostFile" id="uploadPostFile" onChange={(e) => setStoryFile(e.target.files[0])} />
73             </div>
74             <div className="form-floating mb-3" authFormInputs descriptionInput">
75               <input type="text" className="form-control descriptionInput" id="floatingDescription" placeholder="Description"
76                 onChange={(e) => setStoryDescription(e.target.value)} value={storyDescription} />
77               <label htmlFor="floatingDescription">Text</label>
78             </div>
79             {uploadProgress ?
80               <button disabled=Uploading... {Math.round(uploadProgress)}%</button>
81             :
82               <button onClick={handleStoryUpload}>Upload</button>
83             }
84           </form>
85         </div>
86       </div>
87     </div>
88   )
89 }
90
91 export default CreateStory

```

- **Backend:**

The backend for stories is already covered in socket handling file and posts file in server.

## 10. Chat feature:

The backend for the chat is already covered in socket handling. In frontend, we use multiple components. Let's go through each of them.

- **Chat page(main):**

Let's create a new file in pages folder for chat feature.

```

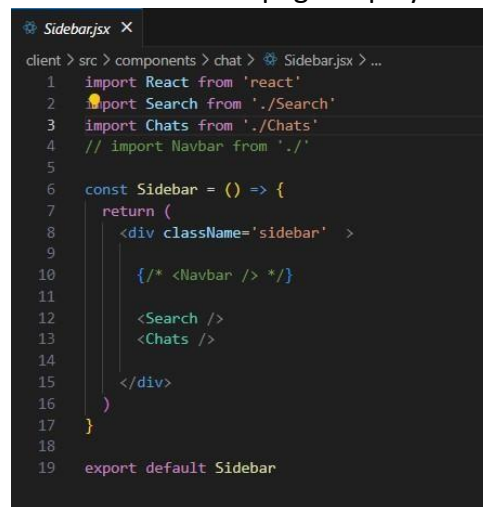
Chat.jsx M X
client > src > pages > Chat.jsx > ...
1   import React from 'react'
2   import '../styles/Chat.css'
3   import Navbar from '../components/Navbar'
4   import Sidebar from '../components/chat/Sidebar'
5   import UserChat from '../components/chat/UserChat'
6
7   const Chat = () => {
8     return (
9       <div className='chatPage'>
10        </* <HomeLogo /> */>
11        <Navbar />
12
13        <div className="home">
14          <Sidebar />
15          <UserChat />
16        </div>
17      </div>
18    )
19  }
20
21  export default Chat

```

- 

### Sidebar:

The sidebar in chat page displays the search component and the users list.



```
client > src > components > chat > Sidebar.jsx > ...
1  import React from 'react'
2  import Search from './Search'
3  import Chats from './Chats'
4  // import Navbar from './'
5
6  const Sidebar = () => {
7    return (
8      <div className='sidebar' >
9
10       {/* <Navbar /> */}
11
12       <Search />
13       <Chats />
14
15     </div>
16   )
17 }
18
19 export default Sidebar
```

- **Search:**

The search feature helps to search for users to chat with them.

```

Search.jsx M X
client > src > components > chat > Search.jsx > Search > handleSelect
1 import React, { useContext, useEffect, useState } from 'react'
2 import { TbSearch } from 'react-icons/tb'
3 import { GeneralContext } from '../../context/GeneralContextProvider';
4
5 const Search = () => {
6
7   const {dispatch, socket} = useContext(GeneralContext)
8   const [search, setSearch] = useState('');
9   const userId = localStorage.getItem('userId');
10  const [user, setUser] = useState();
11  const [err, setErr] = useState(false);
12
13  const handleSearch = async (e) =>{
14    e.preventDefault();
15    setErr(false);
16    setUser();
17    await socket.emit('chat-user-searched', {ownId: userId, username: search});
18    setSearch('')
19  }
20
21  useEffect(()=>{
22    socket.on('searched-chat-user', async ({user})=>{
23      setUser(user);
24    });
25    socket.on('no-searched-chat-user', async ()=>{
26      setErr(true);
27    });
28  },[socket])
29
30  const handleSelect = async (user) =>{
31    await dispatch({type:"CHANGE_USER", payload: user});
32    setUser();
33  }
34
35  return (
36    <div className='search'>
37      <div className="searchform">
38        <input type="text" placeholder='Search'
39          onChange={(e)=> {setSearch(e.target.value)}} value={search} />
40        <div className="s-icon" onClick={handleSearch}>
41          <TbSearch />
42        </div>
43      </div>
44
45      {err && <span>No User Found!!</span>}
46
47      {user && <div className="userInfo" onClick={() => handleSelect(user)} >
48        <img src={user.profilePic} alt="" />
49        <div className="userChatInfo"> <span>{user.username}</span> </div>
50      </div>
51    </div>
52  )
53 }
54 export default Search

```

## Chats:

In chats component, we display the list of users available to chat.



```

Chats.jsx 1. M X
client > src > components > chat > Chats.jsx > Chats > chatFirends.map() callback
1 import React, { useContext, useEffect, useState } from 'react'
2 import { GeneralContext } from '../../context/GeneralContextProvider';
3 const Chats = () => {
4
5     const {socket, chatFirends, setChatFriends, dispatch, chatData} = useContext(GeneralContext)
6     const userId = localStorage.getItem('userId');
7
8     useEffect(()=>{
9
10         socket.emit('fetch-friends', {userId});
11
12         socket.on("friends-data-fetched", ({friendsData})=>{
13             setChatFriends(friendsData);
14         });
15     },[])
16
17     const handleSelect = (data) =>{
18         dispatch({type:"CHANGE_USER", payload: data});
19         console.log(chatData);
20     }
21     useEffect(()=>{
22
23         if(chatData.chatId !== null){
24             socket.emit('fetch-messages', {chatId: chatData.chatId})
25         }
26     }, [chatData])
27
28     return (
29         <div className='chats'>
30
31             {chatFirends.map((data)=>{
32                 return(
33                     <div className="userInfo" key={data._id} onClick={()=> handleSelect(data)} >
34                         <img src={data.profilePic} alt="" />
35                         <div className="userChatInfo">
36                             <span>{data.username}</span>
37                         </div>
38                     </div>
39                 )
40             })}
41         </div>
42     )
43 }
44 export default Chats

```

- **UserChat:**

UserChat contains components such as messages, inputs, etc., that let's users interact.

```

UserChat.jsx M X
client > src > components > chat > UserChat.jsx > UserChat
1 import React, { useContext } from 'react'
2 import Input from './Input';
3 import Messages from './Messages';
4 import { GeneralContext } from '../../context/GeneralContextProvider';
5
6 const UserChat = () => {
7
8     const {chatData} = useContext(GeneralContext);
9
10    return (
11        <div className='chat'>
12            { chatData.user &&
13
14                <div className="chatInfo">
15                    <img src={chatData.user?.profilePic} alt="" />
16                    <span>{chatData.user.username}</span>
17                </div>
18            }
19
20            <Messages />
21            <Input />
22
23        </div>
24    )
25 }
26 export default UserChat

```

## Input:

Input component helps to type and send the message to the user at other end.

```
Input.jsx M X
client > src > components > chat > Input.jsx > Input > handleSend
1  import React, { useContext, useState } from 'react'
2  import { BiImageAdd } from 'react-icons/bi'
3  import { GeneralContext } from '../../context/GeneralContextProvider'
4  import { v4 as uuid } from 'uuid';
5  import { getDownloadURL, ref, uploadBytesResumable } from 'firebase/storage';
6  import { storage } from '../../firebase';
7
8  const Input = () => {
9
10     const {socket, chatData} = useContext(GeneralContext);
11     const [text, setText] = useState('');
12     const [file, setFile] = useState(null);
13     const [uploadProgress, setUploadProgress] = useState();
14     const userId = localStorage.getItem('userId');
15
16     const handleSend = async () =>{
17
18         if (file){
19             const storageRef = ref(storage, uuid());
20             const uploadTask = uploadBytesResumable(storageRef, file);
21             uploadTask.on('state_changed',
22                 (snapshot) => {
23                     setUploadProgress((snapshot.bytesTransferred / snapshot.totalBytes) * 100);
24                 },
25                 (error) => {
26                     console.log(error);
27                 },
28                 () => {
29                     getDownloadURL(uploadTask.snapshot.ref).then( async (downloadURL) => {
30                         console.log('File available at', downloadURL);
31
32                         try{
33                             let date = new Date()
34                             await socket.emit('new-message', {chatId: chatData.chatId ,id: uuid(),
35                                                                 text: text, file: downloadURL,
36                                                                 senderId: userId, date: date});
37                             setUploadProgress();
38                             setText('');
39                             setFile(null);
40                         }catch(err){
41                             console.log(err);
42                         }
43                     });
44                 });
45         }else{
46             let date = new Date()
47             await socket.emit('new-message', {chatId: chatData.chatId ,id: uuid(),
48                                             text: text,file: '', senderId: userId, date: date});
49             setText('');
50         }
51     }
52 }
```

```
Input.jsx M X
client > src > components > chat > Input.jsx > Input > handleSend
53
54     return (
55         <div className='input' >
56             <input type="text" placeholder='type something...' onChange={e => setText(e.target.value)} value={text} />
57             <div className="send">
58                 <input type="file" style={{display : 'none'}} id='file' onChange={e=> setFile(e.target.files[0])} />
59                 <label htmlFor="file" style={{display:'flex'}}>
60                     <BiImageAdd />
61                     <p style={{fontSize: '12px'}}><uploadProgress ? Math.floor(uploadProgress) + '% : ' /></p>
62                 </label>
63                 <button onClick={handleSend} >Send</button>
64             </div>
65         </div>
66     )
67
68     export default Input
```



## Messages:

The messages component is a group of all the messages in the chat. Each message will further be considered as a separate component.

```
Messages.jsx M X
client > src > components > chat > Messages.jsx > Messages
1 import React, { useContext, useEffect, useState } from 'react'
2 import Message from './Message'
3 import { GeneralContext } from '../../../context/GeneralContextProvider';
4
5 const Messages = () => {
6
7   const {socket} = useContext(GeneralContext)
8   const [messages, setMessages] = useState([]);
9   const {chatData} = useContext(GeneralContext);
10
11   useEffect(()=>{
12     const handleMessagesUpdated = ({ chat }) => {
13       console.log('chatuu', chat);
14       if (chat) {
15         setMessages(chat.messages);
16       }
17     };
18
19     const handleNewMessage = async () => {
20       console.log('new message', chatData.chatId);
21       socket.emit('update-messages', { chatId: chatData.chatId });
22     };
23
24     socket.on('messages-updated', handleMessagesUpdated);
25     socket.on('message-from-user', handleNewMessage);
26
27     return () => {
28       // Clean up event listeners when the component unmounts
29       socket.off('messages-updated', handleMessagesUpdated);
30       socket.off('message-from-user', handleNewMessage);
31     };
32   },[socket, chatData])
33
34   return (
35     <div className='messages' >
36
37       {messages.length > 0 && messages.map((message)=>{
38         <Message message={message} key={message.id} />
39       })}
40     </div>
41   )
42 }
43
44 export default Messages
```

## Message:

The message component is an individual component for each message in the chat.

```
Message.jsx M X
client > src > components > chat > Message.jsx > Message
1 import React, { useContext, useEffect, useRef } from 'react'
2 import { GeneralContext } from '../../../context/GeneralContextProvider';
3
4 const Message = ({message}) => {
5
6   const {chatData} = useContext(GeneralContext);
7   const ref = useRef();
8   let date = new Date(message.date);
9
10   useEffect(() => {
11     ref.current?.scrollIntoView({behavior:'smooth'})
12   }, [message]);
13
14   const userId = localStorage.getItem('userId');
15   return (
16     <div>
17       <div ref={ref} className={`message ${message.senderId === userId ? "owner" : ""}`>
18         <div className="messageInfo">
19           <img src={message.senderId === userId ? localStorage.getItem('profilePic') : chatData.user.profilePic} alt="" />
20           <span>{ date.getHours() < 12 ? date.getHours() + ':' + date.getMinutes() + ' AM' : date.getHours()-12 + ':' + date.getMinutes() + ' PM' }</span>
21         </div>
22         <div className="messageContent">
23           <p>{message.text}</p>
24           {message.file && <img src={message.file} alt="" />}
25         </div>
26       </div>
27     </div>
28   )
29 }
30
31 export default Message
```

## 11. Navbar:

Now let's look into the navbar component.

```
client > src > components > Navbar.jsx > @Navbar
8 import { GeneralContext } from '../context/GeneralContextProvider';
9 import { useNavigate } from 'react-router-dom';
10
11 const Navbar = () => {
12
13   const {isCreatPostOpen, setIsCreatPostOpen, setIsCreateStoryOpen, isNotificationsOpen, setNotificationsOpen} = useContext(GeneralContext);
14
15   const navigate = useNavigate();
16   const profilePic = localStorage.getItem('profilePic');
17   const userId = localStorage.getItem('userId');
18
19   return (
20     <div className="Navbar">
21       <BiHomeAlt className="homebtn btns" onClick={()=> navigate('/')} />
22       <BsChatSquareText className="chatbtn btns" onClick={()=> navigate('/chat')} />
23       <CgAddR className="createPostbtn btns" onClick={()=> {setIsCreatPostOpen(!isCreatPostOpen); setIsCreateStoryOpen(false)}} />
24       <TbNotification className="Notifybtn btns" onClick={()=> setNotificationsOpen(!isNotificationsOpen)} />
25       <img className="profile" src={profilePic} alt="" onClick={()=> navigate(`/profile/${userId}`)} />
26     </div>
27   )
28 }
29 export default Navbar
```

## 12. Logo & User Search:

The logo and search components are implemented together. A separate Search component is created to make it better understandable.

```
client > src > components > HomeLogo.jsx > @HomeLogo
1 import React, { useContext, useEffect, useState } from 'react';
2 import logoimg from '../images/SocialeX.png';
3 import '../styles/HomeLogo.css';
4 import { TbSearch } from 'react-icons/tb';
5 import { GeneralContext } from '../context/GeneralContextProvider';
6 import Search from './Search';
7
8 const HomeLogo = () => {
9
10   const {socket} = useContext(GeneralContext);
11   const [search, setSearch] = useState('');
12   const [searchedUser, setSearchedUser] = useState();
13
14   const handleSearch = async ()=>{
15     await socket.emit('user-search', {username: search});
16     setSearch('');
17   }
18   useEffect(()=>{
19     socket.on('searched-user', ({user})=>{
20       setSearchedUser(user);
21     });
22   },[socket])
23
24   return (
25     <div className="LogoSearch">
26       <img className="logoimg" src={logoimg} alt="" />
27       <div className="Search">
28         <input type="text" placeholder="Search" onChange={(e)=> {setSearch(e.target.value)}} value={search} />
29         <div className="s-icon" onClick={handleSearch}>
30           <TbSearch />
31         </div>
32       </div>
33       <Search searchedUser={searchedUser} setSearchedUser={setSearchedUser} />
34     </div>
35   )
36 }
37
38 export default HomeLogo
```

Search Component:

```

Search.jsx M X
client > src > components > Search.jsx > Search
1 import React from 'react';
2 import '../styles/SearchContainer.css';
3 import { useNavigate } from 'react-router-dom';
4
5 const Search = ({searchedUser, setSearchUser}) => {
6   const navigate = useNavigate();
7   return (
8     <div className="searchContainer">
9
10       <div className="searchedUser" && <div className="searchedUserInfo" onClick={()=> {navigate(`/profile/${searchedUser._id}`); setSearchUser();}} >
11         <img src={searchedUser.profilePic} alt="" />
12         <div className="searchedUserChatInfo">
13           <span>{searchedUser.username}</span>
14         </div>
15       </div>
16     </div>
17   );
18 }
19
20 export default Search

```

### 13. Routes in Backend:

As we used a separate routing file for routing at server side, let's implement it.

```

JS Route.js X
server > routes > JS Route.js > default
1 import express from 'express';
2 import { login, register } from '../controllers/Auth.js';
3 import { createPost } from '../controllers/createPost.js';
4 import { fetchAllPosts, fetchAllStories, fetchUserImg, fetchUserName } from '../controllers/Posts.js';
5
6 const router = express.Router();
7
8 router.post('/register', register);
9 router.post('/login', login);
10 router.post('/createPost', createPost);
11 router.get('/fetchAllPosts', fetchAllPosts);
12 router.get('/fetchUserName', fetchUserName);
13 router.get('/fetchUserImg', fetchUserImg);
14 router.get('/fetchAllStories', fetchAllStories);
15
16 export default router;

```

Finally, for any further assistance, use the links below:

Demo link:

<https://drive.google.com/file/d/15JCHvQTNjOBfsxkzomK6s4YkhQBHE18o/view?usp=sharing>

Use the code in: <https://github.com/harsha-varadhan-reddy-07/SocialeX>

\*\*\* Happy coding!! \*\*\*