

Social Media App (MERN)

Introduction:

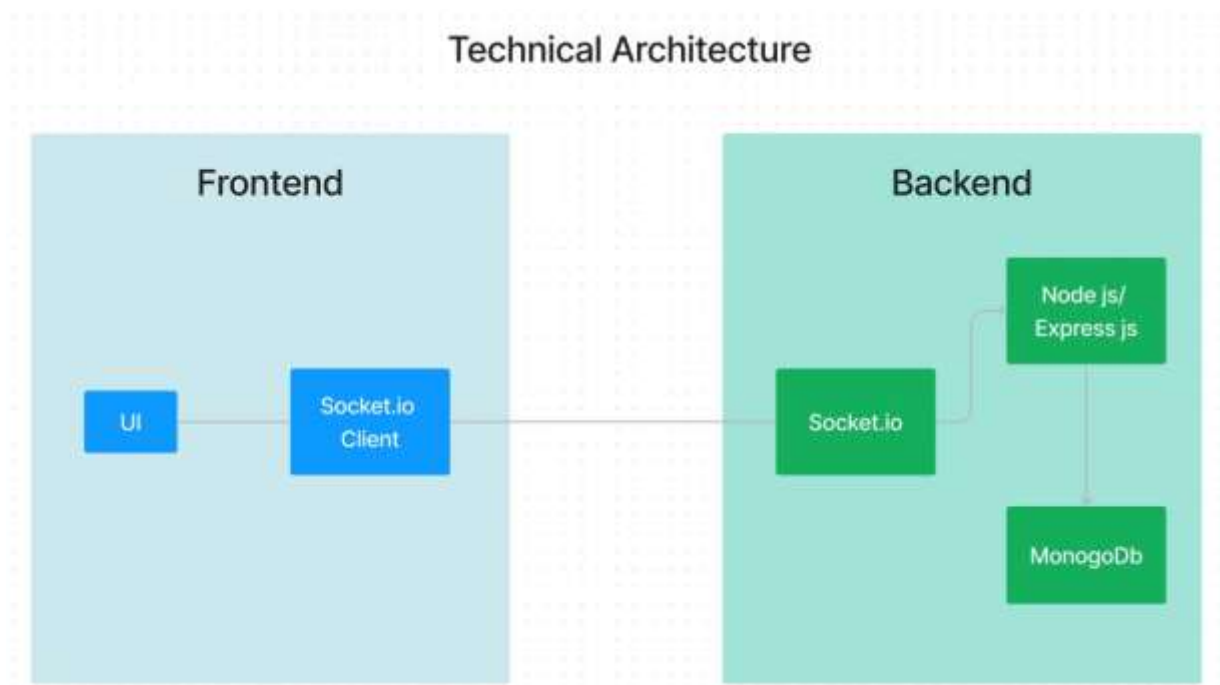
Introducing our revolutionary social media app! Designed to redefine online connections, our platform offers an intuitive and innovative space for seamless communication and engagement.

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.

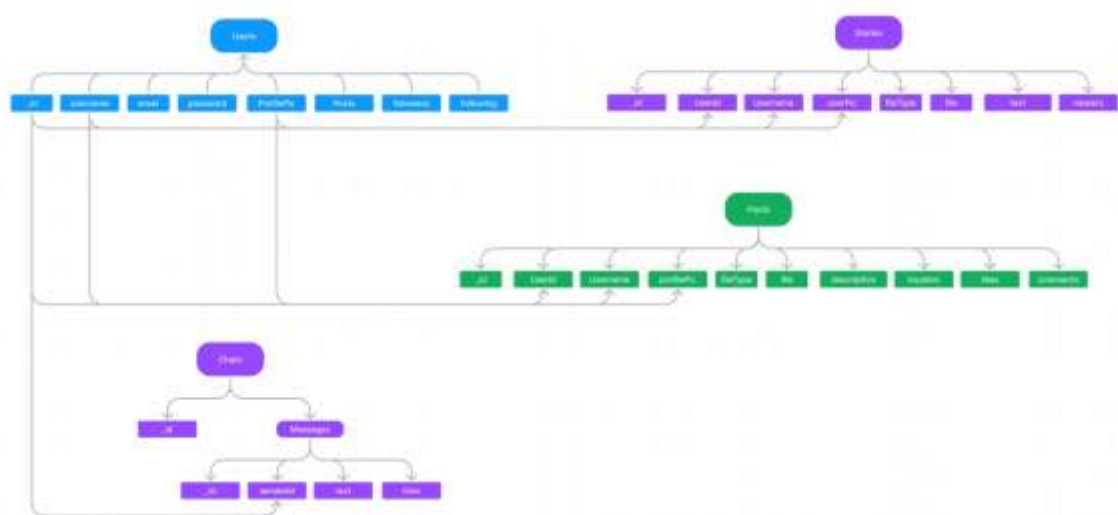
Real-time messaging is facilitated through the socket.io library, enabling instant exchange of messages between users. Users can engage in chats with their friends, sharing text, emojis, images, etc., in real-time.

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/parvinshaik/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.

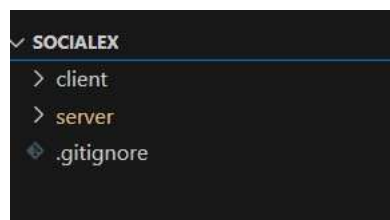
## **Roles & Responsibilities:**

### **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

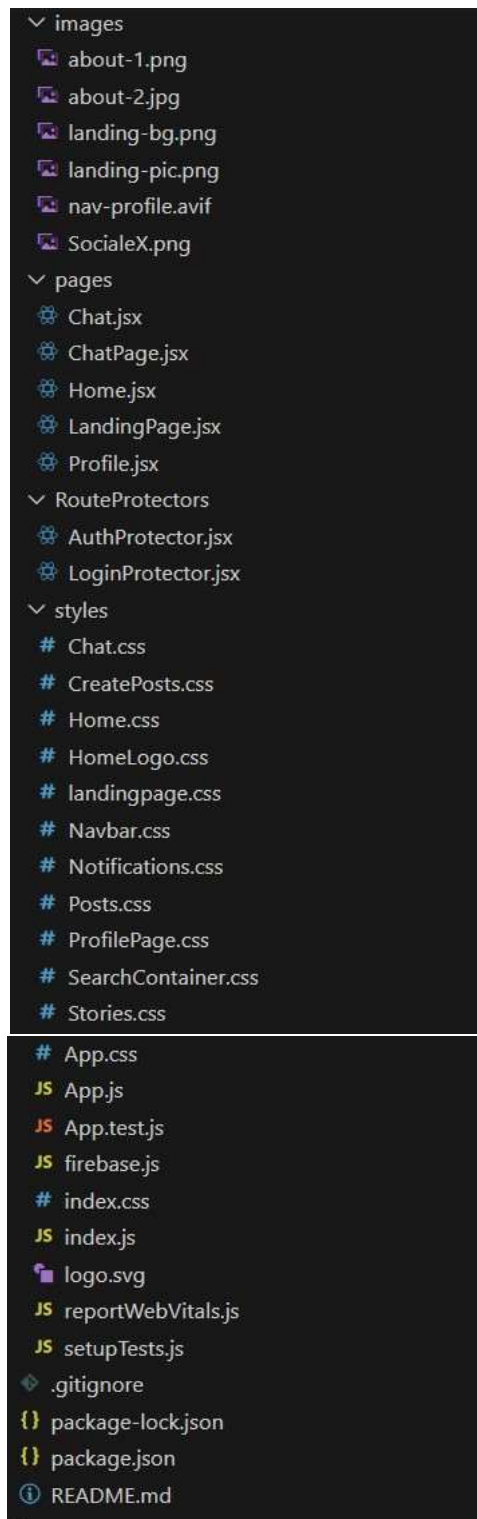


### **Client directory:**

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.

```

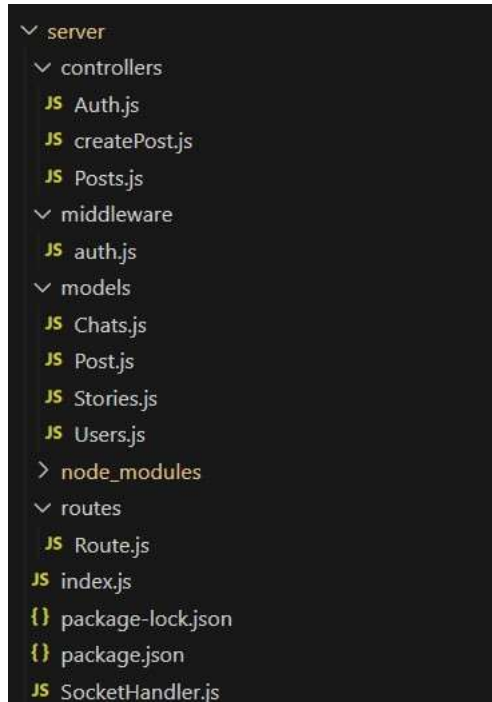
 client
 > node_modules
 > public
 src
 components
 chat
 Chats.jsx
 Input.jsx
 Message.jsx
 Messages.jsx
 Search.jsx
 Sidebar.jsx
 UserChat.jsx
 CreatePost.jsx
 CreateStory.jsx
 HomeLogo.jsx
 Login.jsx
 Navbar.jsx
 Notifications.jsx
 Post.jsx
 Register.jsx
 Search.jsx
 Stories.jsx
 context
 AuthenticationContextProvider.jsx
 GeneralContextProvider.jsx
 SocketContextProvider.jsx
```



### 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/1mZve78StQGyPyZZoD0j0CKmDeKRgRT5t/view?usp=sharing>

Use the code in: <https://github.com/parvinshaik/Socialex.git>

## 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.

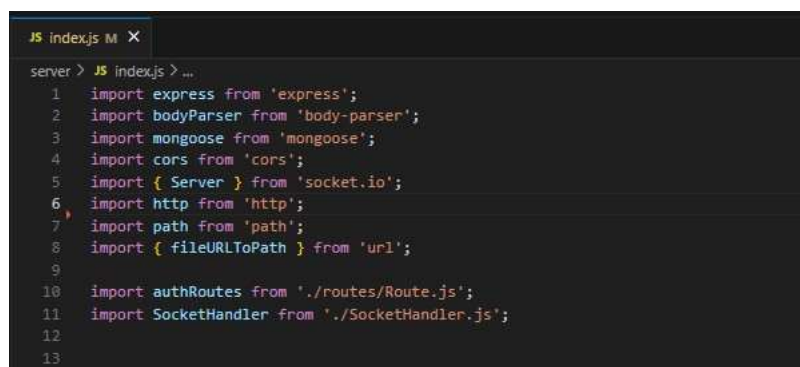
## Add notifications

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.

A screenshot of a code editor window titled 'JS index.js M X'. The editor shows the following code:

```
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 databa

```
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
59).catch((e)=> console.log(`Error in db connection ${e}`));
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 const Stories = mongoose.model('stories', storySchema);
28 export default Stories;
```

## Posts model

```
JS Post.js X
server > models > JS Post.js > [0] 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:



```

Auth.js M X
server > controllers > # Auth.js > @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 = 'thisIsTheSecretCodeForTheJWTToken';
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 = {
38 _id: user._id, username: user.username,
39 email: user.email, profilePic: user.profilePic,
40 about: user.about, posts: user.posts,
41 followers: user.followers, following: user.following };
42
43 res.status(200).json({token, user:userData});
44
45 }catch(err){
46 res.status(500).json({error: err.message});
47 }
48 }

```

Login:

```

Auth.js M X
server > controllers > # Auth.js > @login
49
50 export const login = async (req, res) =>{
51 try{
52 const {email, password} = req.body;
53 const user = await Users.findOne({email:email});
54 if(!user) return res.status(400).json({msg: "User does not exist"});
55
56 const isMatch = await bcrypt.compare(password, user.password);
57 if(!isMatch) return res.status(400).json({msg: "Invalid credentials"});
58
59 // generate jwt token using function we defined at top of the page
60 const token = generateToken(user._id);
61 delete user.password;
62 const userData = {
63 _id: user._id, username: user.username, email: user.email,
64 profilePic: user.profilePic, about: user.about, posts: user.posts,
65 followers: user.followers, following: user.following };
66
67 res.status(200).json({token, user:userData});
68 console.log(token, userData);
69 }catch(err){
70 res.status(500).json({error: err.message});
71 }
72 }

```

## 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:

```
src > components > @ Register.js > |@getedit
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 <span already registered? setIsLoginBox(true)}>Login
32 </form>
33)
34 }
35
36 export default Register
```

### Login UI:

```
src > components > @ login.js > |@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 <span Not registered? setIsLoginBox(false)}>Register
28 </form>
29)
30 }
31
32 export default login
```

## Context Api for Authentication:

```
AuthenticationContextProvider.jsx M X
client > src > context > AuthenticationContextProvider.jsx > 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 > 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
```

```

11 AuthenticationContextProvider.js
12
13 client > src > context > @ AuthenticationContextProvider.js @ AuthenticationContextProvider
14
15 11
16 12
17 13
18 14
19 15
20 16
21 17
22 18
23 19
24 20
25 21
26 22
27 23
28 24
29 25
30 26
31 27
32 28
33 29
34 30
35 31
36 32
37 33
38 34
39 35
40 36
41 37
42 38
43 39
44 40
45 41
46 42
47 43
48 44
49 45
50 46
51 47
52 48
53 49
54 50
55 51
56 52
57 53
58 54
59 55
60 56
61 57
62 58
63 59
64 60
65 61
66 62
67 63
68 64
69 65
70 66
71 67
72 68
73 69
74 70
75 71
76 72
77 73
78 74
79 75
80 76
81 77
82 78
83 79
84 80
85 81
86 82
87 83
88 84
89 85
90 86
91 87
92 88
93 89
94 90
95 91
96 92
97 93
98 94
99 95
100 96
101 97
102 98
103 99
104 100
105 101
106 102
107 103
108 104
109 105
110 106
111 107
112 108
113 109
114 110
115 111
116 112
117 113
118 114
119 115
120 116
121 117
122 118
123 119
124 120
125 121
126 122
127 123
128 124
129 125
130 126
131 127
132 128
133 129
134 130
135 131
136 132
137 133
138 134
139 135
140 136
141 137
142 138
143 139
144 140
145 141
146 142
147 143
148 144
149 145
150 146
151 147
152 148
153 149
154 150
155 151
156 152
157 153
158 154
159 155
160 156
161 157
162 158
163 159
164 160
165 161
166 162
167 163
168 164
169 165
170 166
171 167
172 168
173 169
174 170
175 171
176 172
177 173
178 174
179 175
180 176
181 177
182 178
183 179
184 180
185 181
186 182
187 183
188 184
189 185
190 186
191 187
192 188
193 189
194 190
195 191
196 192
197 193
198 194
199 195
200 196
201 197
202 198
203 199
204 200
205 201
206 202
207 203
208 204
209 205
210 206
211 207
212 208
213 209
214 210
215 211
216 212
217 213
218 214
219 215
220 216
221 217
222 218
223 219
224 220
225 221
226 222
227 223
228 224
229 225
230 226
231 227
232 228
233 229
234 230
235 231
236 232
237 233
238 234
239 235
240 236
241 237
242 238
243 239
244 240
245 241
246 242
247 243
248 244
249 245
250 246
251 247
252 248
253 249
254 250
255 251
256 252
257 253
258 254
259 255
260 256
261 257
262 258
263 259
264 260
265 261
266 262
267 263
268 264
269 265
270 266
271 267
272 268
273 269
274 270
275 271
276 272
277 273
278 274
279 275
280 276
281 277
282 278
283 279
284 280
285 281
286 282
287 283
288 284
289 285
290 286
291 287
292 288
293 289
294 290
295 291
296 292
297 293
298 294
299 295
300 296
301 297
302 298
303 299
304 300
305 301
306 302
307 303
308 304
309 305
310 306
311 307
312 308
313 309
314 310
315 311
316 312
317 313
318 314
319 315
320 316
321 317
322 318
323 319
324 320
325 321
326 322
327 323
328 324
329 325
330 326
331 327
332 328
333 329
334 330
335 331
336 332
337 333
338 334
339 335
340 336
341 337
342 338
343 339
344 340
345 341
346 342
347 343
348 344
349 345
350 346
351 347
352 348
353 349
354 350
355 351
356 352
357 353
358 354
359 355
360 356
361 357
362 358
363 359
364 360
365 361
366 362
367 363
368 364
369 365
370 366
371 367
372 368
373 369
374 370
375 371
376 372
377 373
378 374
379 375
380 376
381 377
382 378
383 379
384 380
385 381
386 382
387 383
388 384
389 385
390 386
391 387
392 388
393 389
394 390
395 391
396 392
397 393
398 394
399 395
400 396
401 397
402 398
403 399
404 400
405 401
406 402
407 403
408 404
409 405
410 406
411 407
412 408
413 409
414 410
415 411
416 412
417 413
418 414
419 415
420 416
421 417
422 418
423 419
424 420
425 421
426 422
427 423
428 424
429 425
430 426
431 427
432 428
433 429
434 430
435 431
436 432
437 433
438 434
439 435
440 436
441 437
442 438
443 439
444 440
445 441
446 442
447 443
448 444
449 445
450 446
451 447
452 448
453 449
454 450
455 451
456 452
457 453
458 454
459 455
460 456
461 457
462 458
463 459
464 460
465 461
466 462
467 463
468 464
469 465
470 466
471 467
472 468
473 469
474 470
475 471
476 472
477 473
478 474
479 475
480 476
481 477
482 478
483 479
484 480
485 481
486 482
487 483
488 484
489 485
490 486
491 487
492 488
493 489
494 490
495 491
496 492
497 493
498 494
499 495
500 496
501 497
502 498
503 499
504 500
505 501
506 502
507 503
508 504
509 505
510 506
511 507
512 508
513 509
514 510
515 511
516 512
517 513
518 514
519 515
520 516
521 517
522 518
523 519
524 520
525 521
526 522
527 523
528 524
529 525
530 526
531 527
532 528
533 529
534 530
535 531
536 532
537 533
538 534
539 535
540 536
541 537
542 538
543 539
544 540
545 541
546 542
547 543
548 544
549 545
550 546
551 547
552 548
553 549
554 550
555 551
556 552
557 553
558 554
559 555
560 556
561 557
562 558
563 559
564 560
565 561
566 562
567 563
568 564
569 565
570 566
571 567
572 568
573 569
574 570
575 571
576 572
577 573
578 574
579 575
580 576
581 577
582 578
583 579
584 580
585 581
586 582
587 583
588 584
589 585
590 586
591 587
592 588
593 589
594 590
595 591
596 592
597 593
598 594
599 595
600 596
601 597
602 598
603 599
604 600
605 601
606 602
607 603
608 604
609 605
610 606
611 607
612 608
613 609
614 610
615 611
616 612
617 613
618 614
619 615
620 616
621 617
622 618
623 619
624 620
625 621
626 622
627 623
628 624
629 625
630 626
631 627
632 628
633 629
634 630
635 631
636 632
637 633
638 634
639 635
640 636
641 637
642 638
643 639
644 640
645 641
646 642
647 643
648 644
649 645
650 646
651 647
652 648
653 649
654 650
655 651
656 652
657 653
658 654
659 655
660 656
661 657
662 658
663 659
664 660
665 661
666 662
667 663
668 664
669 665
670 666
671 667
672 668
673 669
674 670
675 671
676 672
677 673
678 674
679 675
680 676
681 677
682 678
683 679
684 680
685 681
686 682
687 683
688 684
689 685
690 686
691 687
692 688
693 689
694 690
695 691
696 692
697 693
698 694
699 695
700 696
701 697
702 698
703 699
704 700
705 701
706 702
707 703
708 704
709 705
710 706
711 707
712 708
713 709
714 710
715 711
716 712
717 713
718 714
719 715
720 716
721 717
722 718
723 719
724 720
725 721
726 722
727 723
728 724
729 725
730 726
731 727
732 728
733 729
734 730
735 731
736 732
737 733
738 734
739 735
740 736
741 737
742 738
743 739
744 740
745 741
746 742
747 743
748 744
749 745
750 746
751 747
752 748
753 749
754 750
755 751
756 752
757 753
758 754
759 755
760 756
761 757
762 758
763 759
764 760
765 761
766 762
767 763
768 764
769 765
770 766
771 767
772 768
773 769
774 770
775 771
776 772
777 773
778 774
779 775
780 776
781 777
782 778
783 779
784 780
785 781
786 782
787 783
788 784
789 785
790 786
791 787
792 788
793 789
794 790
795 791
796 792
797 793
798 794
799 795
800 796
801 797
802 798
803 799
804 800
805 801
806 802
807 803
808 804
809 805
810 806
811 807
812 808
813 809
814 810
815 811
816 812
817 813
818 814
819 815
820 816
821 817
822 818
823 819
824 820
825 821
826 822
827 823
828 824
829 825
830 826
831 827
832 828
833 829
834 830
835 831
836 832
837 833
838 834
839 835
840 836
841 837
842 838
843 839
844 840
845 841
846 842
847 843
848 844
849 845
850 846
851 847
852 848
853 849
854 850
855 851
856 852
857 853
858 854
859 855
860 856
861 857
862 858
863 859
864 860
865 861
866 862
867 863
868 864
869 865
870 866
871 867
872 868
873 869
874 870
875 871
876 872
877 873
878 874
879 875
880 876
881 877
882 878
883 879
884 880
885 881
886 882
887 883
888 884
889 885
890 886
891 887
892 888
893 889
894 890
895 891
896 892
897 893
898 894
899 895
900 896
901 897
902 898
903 899
904 900
905 901
906 902
907 903
908 904
909 905
910 906
911 907
912 908
913 909
914 910
915 911
916 912
917 913
918 914
919 915
920 916
921 917
922 918
923 919
924 920
925 921
926 922
927 923
928 924
929 925
930 926
931 927
932 928
933 929
934 930
935 931
936 932
937 933
938 934
939 935
940 936
941 937
942 938
943 939
944 940
945 941
946 942
947 943
948 944
949 945
950 946
951 947
952 948
953 949
954 950
955 951
956 952
957 953
958 954
959 955
960 956
961 957
962 958
963 959
964 960
965 961
966 962
967 963
968 964
969 965
970 966
971 967
972 968
973 969
974 970
975 971
976 972
977 973
978 974
979 975
980 976
981 977
982 978
983 979
984 980
985 981
986 982
987 983
988 984
989 985
990 986
991 987
992 988
993 989
994 990
995 991
996 992
997 993
998 994
999 995
1000 996

```

#### 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:

```

1 GeneralContextProvider.js
2
3 client > src > context > @ GeneralContextProvider.js @ GeneralContextProvider
4
5 1
6 2
7 3
8 4
9 5
10 6
11 7
12 8
13 9
14 10
15 11
16 12
17 13
18 14
19 15
20 16
21 17
22 18
23 19
24 20
25 21
26 22
27 23
28 24
29 25
30 26
31 27
32 28
33 29
34 30
35 31
36 32
37 33
38 34
39 35
40 36
41 37
42 38
43 39
44 40
45 41
46 42
47 43
48 44
49 45
50 46
51 47
52 48
53 49
54 50
55 51
56 52
57 53
58 54
59 55
60 56
61 57
62 58
63 59
64 60
65 61
66 62
67 63
68 64
69 65
70 66
71 67
72 68
73 69
74 70
75 71
76 72
77 73
78 74
79 75
80 76
81 77
82 78
83 79
84 80
85 81
86 82
87 83
88 84
89 85
90 86
91 87
92 88
93 89
94 90
95 91
96 92
97 93
98 94
99 95
100 96
101 97
102 98
103 99
104 100
105 101
106 102
107 103
108 104
109 105
110 106
111 107
112 108
113 109
114 110
115 111
116 112
117 113
118 114
119 115
120 116
121 117
122 118
123 119
124 120
125 121
126 122
127 123
128 124
129 125
130 126
131 127
132 128
133 129
134 130
135 131
136 132
137 133
138 134
139 135
140 136
141 137
142 138
143 139
144 140
145 141
146 142
147 143
148 144
149 145
150 146
151 147
152 148
153 149
154 150
155 151
156 152
157 153
158 154
159 155
160 156
161 157
162 158
163 159
164 160
165 161
166 162
167 163
168 164
169 165
170 166
171 167
172 168
173 169
174 170
175 171
176 172
177 173
178 174
179 175
180 176
181 177
182 178
183 179
184 180
185 181
186 182
187 183
188 184
189 185
190 186
191 187
192 188
193 189
194 190
195 191
196 192
197 193
198 194
199 195
200 196
201 197
202 198
203 199
204 200
205 201
206 202
207 203
208 204
209 205
210 206
211 207
212 208
213 209
214 210
215 211
216 212
217 213
218 214
219 215
220 216
221 217
222 218
223 219
224 220
225 221
226 222
227 223
228 224
229 225
230 226
231 227
232 228
233 229
234 230
235 231
236 232
237 233
238 234
239 235
240 236
241 237
242 238
243 239
244 240
245 241
246 242
247 243
248 244
249 245
250 246
251 247
252 248
253 249
254 250
255 251
256 252
257 253
258 254
259 255
260 256
261 257
262 258
263 259
264 260
265 261
266 262
267 263
268 264
269 265
270 266
271 267
272 268
273 269
274 270
275 271
276 272
277 273
278 274
279 275
280 276
281 277
282 278
283 279
284 280
285 281
286 282
287 283
288 284
289 285
290 286
291 287
292 288
293 289
294 290
295 291
296 292
297 293
298 294
299 295
300 296
301 297
302 298
303 299
304 300
305 301
306 302
307 303
308 304
309 305
310 306
311 307
312 308
313 309
314 310
315 311
316 312
317 313
318 314
319 315
320 316
321 317
322 318
323 319
324 320
325 321
326 322
327 323
328 324
329 325
330 326
331 327
332 328
333 329
334 330
335 331
336 332
337 333
338 334
339 335
340 336
341 337
342 338
343 339
344 340
345 341
346 342
347 343
348 344
349 345
350 346
351 347
352 348
353 349
354 350
355 351
356 352
357 353
358 354
359 355
360 356
361 357
362 358
363 359
364 360
365 361
366 362
367 363
368 364
369 365
370 366
371 367
372 368
373 369
374 370
375 371
376 372
377 373
378 374
379 375
380 376
381 377
382 378
383 379
384 380
385 381
386 382
387 383
388 384
389 385
390 386
391 387
392 388
393 389
394 390
395 391
396 392
397 393
398 394
399 395
400 396
401 397
402 398
403 399
404 400
405 401
406 402
407 403
408 404
409 405
410 406
411 407
412 408
413 409
414 410
415 411
416 412
417 413
418 414
419 415
420 416
421 417
422 418
423 419
424 420
425 421
426 422
427 423
428 424
429 425
430 426
431 427
432 428
433 429
434 430
435 431
436 432
437 433
438 434
439 435
440 436
441 437
442 438
443 439
444 440
445 441
446 442
447 443
448 444
449 445
450 446
451 447
452 448
453 449
454 450
455 451
456 452
457 453
458 454
459 455
460 456
461 457
462 458
463 459
464 460
465 461
466 462
467 463
468 464
469 465
470 466
471 467
472 468
473 469
474 470
475 471
476 472
477 473
478 474
479 475
480 476
481 477
482 478
483 479
484 480
485 481
486 482
487 483
488 484
489 485
490 486
491 487
492 488
493 489
494 490
495 491
496 492
497 493
498 494
499 495
500 496
501 497
502 498
503 499
504 500
505 501
506 502
507 503
508 504
509 505
510 506
511 507
512 508
513 509
514 510
515 511
516 512
517 513
518 514
519 515
520 516
521 517
522 518
523 519
524 520
525 521
526 522
527 523
528 524
529 525
530 526
531 527
532 528
533 529
534 530
535 531
536 532
537 533
538 534
539 535
540 536
541 537
542 538
543 539
544 540
545 541
546 542
547 543
548 544
549 545
550 546
551 547
552 548
553 549
554 550
555 551
556 552
557 553
558 554
559 555
560 556
561 557
562 558
563 559
564 560
565 561
566 562
567 563
568 564
569 565
570 566
571 567
572 568
573 569
574 570
575 571
57
```



## 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.

```

1 // CreatePost.jsx
2 import React, { useState } from 'react';
3 import './CreatePostForm.css';
4 import { BrowserRouter } from 'react-router-dom';
5 import { useSelector } from 'redux';
6 import axios from 'axios';
7 import { ref, uploadBytesResumable, getDownloadURL } from 'firebase/storage';
8 import { storage } from '../firebase.js';
9 export { ref as addRef } from 'react-redux';
10
11 const CreatePost = () => {
12 const [createPostForm, setCreatePostForm] = useState({});
13 const [postType, setPostType] = useState('text');
14 const [postDescription, setPostDescription] = useState('');
15 const [postLocation, setPostLocation] = useState('');
16 const [postId, setPostId] = useState(null);
17 const [uploadProgress, setUploadProgress] = useState(0);
18
19 if (uploadProgress === 100) {
20 setPostDescription('');
21 setPostLocation('');
22 setPostFile(null);
23 setCreatePostForm(false);
24 setUploadProgress(0);
25 }
26
27 const handlePostClick = async () => {
28 e.preventDefault();
29 const formData = new FormData();
30 const uploadTask = uploadBytesResumable(storageRef, postFile);
31 uploadTask.on('state_changed',
32 (snapshot) => {
33 setUploadProgress((snapshot.bytesTransferred / snapshot.totalBytes) * 100);
34 },
35 error => {}
36);
37 try {
38 getDownloadURL(uploadTask.snapshot.ref).then(() => setDownloadURL());
39 console.log('File available at', downloadURL);
40 } catch {
41 console.error('Error uploading file');
42 }
43 const inputs = Object.values(createPostForm).filter(input => input !== '');
44 const postData = {
45 postId: Date.now().toString(),
46 postType: postType,
47 postDescription: postDescription,
48 postLocation: postLocation,
49 comments: []
50 };
51 axios.post('http://localhost:3001/api/posts', postData)
52 .then(response => {
53 console.log('Post created successfully');
54 })
55 .catch(error => {
56 console.error('Error creating post');
57 });
58 };
59
60 return (
61 <div>
62 <h2>Create Post</h2>
63 <form>
64 <div>
65 <label>Post Type:</label>
66 <select>
67 <option value='text'>text</option>
68 <option value='image'>image</option>
69 <option value='video'>video</option>
70 </select>
71 </div>
72 <div>
73 <label>Post Description:</label>
74 <input type='text' value={postDescription}/>
75 </div>
76 <div>
77 <label>Post Location:</label>
78 <input type='text' value={postLocation}/>
79 </div>
80 <div>
81 <label>Post File:</label>
82 <input type='file' value={postFile}/>
83 </div>
84 <button type='button' onClick={handlePostClick}>Create Post</button>
85 </form>
86 </div>
87);
88 };
89
90 export default CreatePost;

```

[illegible]

## Backend:

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

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

```
client > src > components > Posts > 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 { IoPersonAdd } 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:8001/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('likeUpdated')
40 })
41 socket.on('userfollowed', ({following}) => {
42 localStorage.setItem('following', following);
43 })
44 }, [socket])
45
46 const handleFollow = async (userId) => {
47 socket.emit('followUser', {userId: 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 }
```



## 7. Socket Handling in backend:

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

```
SocketHandler.js X
server > 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 socket.on('updateProfile', async ({userId, profilePic, username, about})=>{
27 const user = await User.updateOne({_id: userId}, {profilePic: profilePic, username: username, about: about});
28 socket.emit('profile-fetched', {profile: user})
29 })
30
31 socket.on('user-search', async ({username})=>{
32 const user = await User.findOne({username: username});
33 socket.emit('searched-user', {user});
34 })
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 + user1._id
48 })
49
50 const chat = await newChat.save();
51 }
52 })
53
54 }
```

```
SocketHandler.js X
server > 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 }
```



```

SocketHandler.js X
server > 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

```

```

SocketHandler.js X
server > 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.

```
Stories.jsx M X
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
28 const response = await axios.get('http://localhost:6001/fetchAllStories');
29 setStories(response.data)
30 console.log(response.data[0])
31 } catch (error) {
32 console.error(error);
33 }
34 };
35
36 const handleOpenStory = async (story) =>{
37
38 setStory(story);
39 await socket.emit('story-played', {storyId: story._id, userId: localStorage.getItem('userId')});
40 setIsStoryPlaying(true);
41
42 }
43 }
```

```
Stories.jsx M X
client > src > components > Stories.jsx > Stories
44
45 <div className="storiesContainer">
46
47 <div className="storiesHeader">
48 <div>Stories / 10</div>
49 </div>
50
51 <div className="storiesBody" style={isStoryPlaying ? {display: 'none'} : {}}>
52
53 <div className="story">
54
55 <div className="story user-story" onClick={handleOpenStory}>
56
57 <div>Add story</div>
58 <div>@Poojapada </div>
59 </div>
60
61 <div>
62 <div>
63 <div>
64 <div>
65 <div>
66 <div>
67 <div>
68 <div>
69 <div>
70 <div>
71 <div>
72 <div>
73 <div>
74 <div>
75 <div>
76 <div>
77 <div>
78 <div>
79 <div>
80 <div>
81 </div>
82 </div>
83 </div>
84 </div>
85 </div>
86 </div>
87 </div>
88 </div>
89 </div>
90 </div>
91 </div>
92 </div>
93 </div>
94 </div>
95 </div>
96 </div>
97 </div>
98 </div>
99 </div>
100 </div>
101 </div>
102
103 <div>
104 <div>
105 <div>
106 <div>
107 <div>
108 <div>
109 <div>
110 <div>
111 <div>
112 <div>
113 <div>
114 <div>
115 <div>
116 <div>
117 <div>
118 <div>
119 <div>
120 <div>
121 <div>
122 <div>
123 <div>
124 </div>
125 </div>
126 </div>
127 </div>
128 </div>
129 </div>
130 </div>
131 </div>
132 </div>
133 </div>
134 </div>
135 </div>
136 </div>
137 </div>
138 </div>
139 </div>
140 </div>
141 </div>
142 </div>
143 </div>
144 </div>
145 </div>
146 </div>
```



```
@GatsbyPage() <X>
plant {
 components ? @ GatsbyPage > HOC(Gatsby) > HOC(renderAllPages)

 return () =>
 <div className="createStoryForm">{style}</createStoryForm> {display: "contents"} : {display: "none"}}>
 <div className="createFormContainer">

 <@Gatsby className="CreatePostForm" onClick={()=> setHistoryOpen(true)} />
 <div className="createFormTitle">Add new story</div>
 <div className="createForm">

 <div className="createFormBody">
 <form>

 <select className="form-select" aria-label="Select Post Type" onChange={e=> setStoryType(e.target.value)}>
 <option defaultSelected="" value="photo">Choose post type</option>
 <option value="photo">Photo</option>
 <option value="video">Video</option>
 </select>

 <div className="uploadArea">
 <input type="file" name="postData" id="uploadPostFile" onChange={e=> setStoryFile(e.target.files[0])} />
 <p></p>
 </div>
 <div className="form-floating no-1 autoWidthInput descriptionInput">
 <input type="text" className="form-control descriptionInput" id="floatingDescription" placeholder="description"
 onChange={e=> setDescription(e.target.value)} value=storyDescription />
 <label htmlFor="floatingDescription">Text</label>
 </div>
 <@loadProgress>
 <button disabled=isLoading... {Math.round(loadProgress)}%</button>
 </div>
 <button onClick=GatsbyStoryUpload>Upload</button>
 </form>
 </div>
 </div>
 </div>
 }
 }
}
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.

```

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
15 <Sidebar />
16 <UserChat />
17
18 </div>
19 </div>
20)
21 }
22
23 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 './Navbar'
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.

```
client > src > components > chat > Search.jsx > Search > Search.jsx
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', { ownerId: 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 && No User Found!}
46
47 {user && <div className="userInfo" onClick={() => handleSelect(user)} >
48
49 <div className="userChatInfo"> {user.username} </div>
50 </div>
51 </div>
52)
53 }
54 export default Search
```

## Chats:

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

```
client > src > components > chat > @ Chats.js > @ Chat > @ chatFriends.map() callback
1 import React, { useContext, useEffect, useState } from 'react'
2 import { GeneralContext } from '../context/GeneralContextProvider';
3 const Chats = () => {
4
5 const {socket, chatFriends, 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
22 useEffect(()=>{
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 <chatFriends.map[(data)]=>{
32 return(
33 <div className='userInfo' key={data._id} onClick={()=> handleSelect(data)} >
34
35 <div className='userChatInfo'>
36 {data.Username}
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.

```
client > src > components > chat > UserChat.js > @ 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
16 {chatData.user.username}
17 </div>
18 </div>
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.

```
client > src > components > chat > Input.jsx > 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 }
```

```
client > src > components > chat > Input.jsx > Input
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
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('chatou', 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
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 data = new Date(message.createdAt);
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' >
18 <div className='message-header' >
19
20 <div>
21 <div>
22 {data.getHours() < 12 ? 'data.getHours()' : 'AM'} <div> {data.getMinutes() < 10 ? '0' : ''} {data.getMinutes()}
23 </div>
24 <div>
25 {message.text}
26 </div>
27 <div>
28 {message.fileName &&
29 </div>
30 </div>
31 </div>
32 </div>
33 </div>
34)
35 }
36
37 export default Message
```

## 11. Navbar:

Now let's look into the navbar component.

[illegible]

## 12. Logo & User Search:

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

```

1 HomeLogo.jsx | X
2
3 client > src > components > @ HomeLogo.js | @ HomeLogo
4
5 1 import React, { useContext, useEffect, useState } from 'react';
6 2 import logging from '../images/socialA.png';
7 3 import './styles/HomeLogo.css';
8 4 import { TSearch } from 'react-icons/tb';
9 5 import { GeneralContext } from '../context/GeneralContextProvider';
10 6 import Search from './Search';
11
12 7
13 8
14 9
15 10
16 11
17 12
18 13
19 14
20 15
21 16
22 17
23 18
24 19
25 20
26 21
27 22
28 23
29 24
30 25
31 26
32 27
33 28
34 29
35 30
36 31
37 32
38 33
39 34
40 35
41 36
42 37
43 38
44 39
45 40
46 41
47 42
48 43
49 44
50 45
51 46
52 47
53 48
54 49
55 50
56 51
57 52
58 53
59 54
60 55
61 56
62 57
63 58
64 59
65 60
66 61
67 62
68 63
69 64
70 65
71 66
72 67
73 68
74 69
75 70
76 71
77 72
78 73
79 74
80 75
81 76
82 77
83 78
84 79
85 80
86 81
87 82
88 83
89 84
90 85
91 86
92 87
93 88
94 89
95 90
96 91
97 92
98 93
99 94
100 95
101 96
102 97
103 98
104 99
105 100
106 101
107 102
108 103
109 104
110 105
111 106
112 107
113 108
114 109
115 110
116 111
117 112
118 113
119 114
120 115
121 116
122 117
123 118
124 119
125 120
126 121
127 122
128 123
129 124
130 125
131 126
132 127
133 128
134 129
135 130
136 131
137 132
138 133
139 134
140 135
141 136
142 137
143 138
144 139
145 140
146 141
147 142
148 143
149 144
150 145
151 146
152 147
153 148
154 149
155 150
156 151
157 152
158 153
159 154
160 155
161 156
162 157
163 158
164 159
165 160
166 161
167 162
168 163
169 164
170 165
171 166
172 167
173 168
174 169
175 170
176 171
177 172
178 173
179 174
180 175
181 176
182 177
183 178
184 179
185 180
186 181
187 182
188 183
189 184
190 185
191 186
192 187
193 188
194 189
195 190
196 191
197 192
198 193
199 194
200 195
201 196
202 197
203 198
204 199
205 200
206 201
207 202
208 203
209 204
210 205
211 206
212 207
213 208
214 209
215 210
216 211
217 212
218 213
219 214
220 215
221 216
222 217
223 218
224 219
225 220
226 221
227 222
228 223
229 224
230 225
231 226
232 227
233 228
234 229
235 230
236 231
237 232
238 233
239 234
240 235
241 236
242 237
243 238
244 239
245 240
246 241
247 242
248 243
249 244
250 245
251 246
252 247
253 248
254 249
255 250
256 251
257 252
258 253
259 254
260 255
261 256
262 257
263 258
264 259
265 260
266 261
267 262
268 263
269 264
270 265
271 266
272 267
273 268
274 269
275 270
276 271
277 272
278 273
279 274
280 275
281 276
282 277
283 278
284 279
285 280
286 281
287 282
288 283
289 284
290 285
291 286
292 287
293 288
294 289
295 290
296 291
297 292
298 293
299 294
300 295
301 296
302 297
303 298
304 299
305 300
306 301
307 302
308 303
309 304
310 305
311 306
312 307
313 308
314 309
315 310
316 311
317 312
318 313
319 314
320 315
321 316
322 317
323 318
324 319
325 320
326 321
327 322
328 323
329 324
330 325
331 326
332 327
333 328
334 329
335 330
336 331
337 332
338 333
339 334
340 335
341 336
342 337
343 338
344 339
345 340
346 341
347 342
348 343
349 344
350 345
351 346
352 347
353 348
354 349
355 350
356 351
357 352
358 353
359 354
360 355
361 356
362 357
363 358
364 359
365 360
366 361
367 362
368 363
369 364
370 365
371 366
372 367
373 368
374 369
375 370
376 371
377 372
378 373
379 374
380 375
381 376
382 377
383 378
384 379
385 380
386 381
387 382
388 383
389 384
390 385
391 386
392 387
393 388
394 389
395 390
396 391
397 392
398 393
399 394
400 395
401 396
402 397
403 398
404 399
405 400
406 401
407 402
408 403
409 404
410 405
411 406
412 407
413 408
414 409
415 410
416 411
417 412
418 413
419 414
420 415
421 416
422 417
423 418
424 419
425 420
426 421
427 422
428 423
429 424
430 425
431 426
432 427
433 428
434 429
435 430
436 431
437 432
438 433
439 434
440 435
441 436
442 437
443 438
444 439
445 440
446 441
447 442
448 443
449 444
450 445
451 446
452 447
453 448
454 449
455 450
456 451
457 452
458 453
459 454
460 455
461 456
462 457
463 458
464 459
465 460
466 461
467 462
468 463
469 464
470 465
471 466
472 467
473 468
474 469
475 470
476 471
477 472
478 473
479 474
480 475
481 476
482 477
483 478
484 479
485 480
486 481
487 482
488 483
489 484
490 485
491 486
492 487
493 488
494 489
495 490
496 491
497 492
498 493
499 494
500 495
501 496
502 497
503 498
504 499
505 500
506 501
507 502
508 503
509 504
510 505
511 506
512 507
513 508
514 509
515 510
516 511
517 512
518 513
519 514
520 515
521 516
522 517
523 518
524 519
525 520
526 521
527 522
528 523
529 524
530
```

**Search Component:**

```

10 Search.js M X
11
12 client > src > components > @ Search.js | M Search
13
14 1 import React from 'react'
15 2 import './styles/SearchContainer.css'
16 3 import { useNavigate } from 'react-router-dom'
17 4
18 5 const Search = ({searchUser, setSearchUser}) => {
19 6 const navigate = useNavigate()
20 7 return (
21 8 <div className="searchContainer">
22 9
23 10 <searchUser AS <div className="searchUserInfo" onClick={() => navigate(`/profile/${searchUser._id}`)} setSearchUser({}) >
24 11
25 12 <div className="searchUserContainer">
26 13 <div>{searchUser.username}</div>
27 14 <div>{searchUser.email}</div>
28 15 </div>
29 16 </div>
30 17 </div>
31 18)
32 19 }
33
34 export default Search
35

```

### 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 > [0] 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;
```

**Demo link:**

<https://drive.google.com/file/d/1mZve78StQGyPyZZoD0j0CKmDeKRgRT5t/view?usp=sharing>

**Github Link :** <https://github.com/parvinshaik/Socialex.git>