SANGAM

SOCIAL MEDIA APP

Tiwari Shaan

Introducing our groundbreaking social media platform, crafted to transform the way you connect and interact online! Our app provides an intuitive, innovative space designed for effortless communication and engagement.

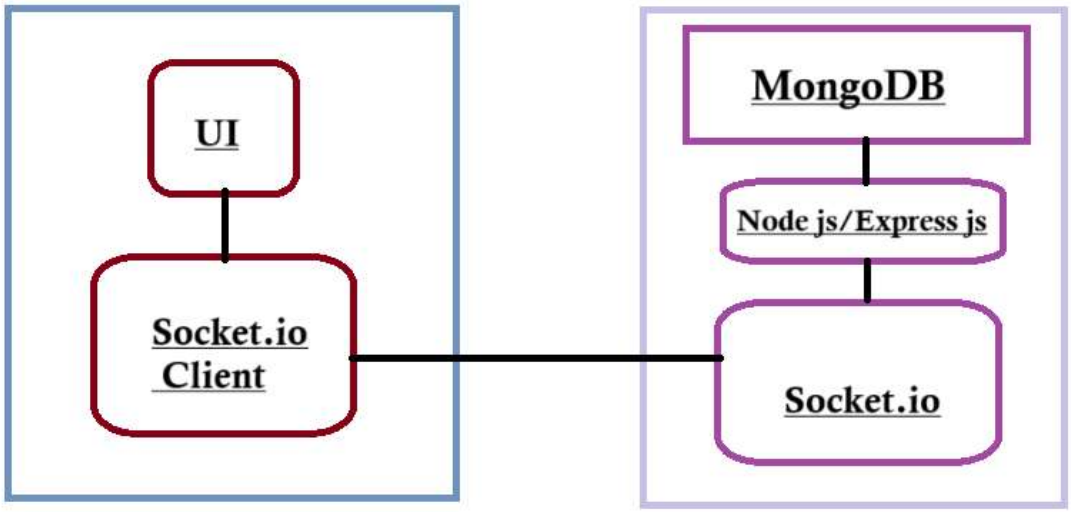
Break new ground with real-time messaging that makes collaboration and conversation smoother than ever. Whether you're brainstorming ideas, working on projects, or enjoying casual chats, our messaging system ensures seamless interaction and clarity.

Capture and keep every significant moment with our handy post-saving feature. Save important posts, insightful articles, or motivating content for future reference and share them with your network. With our app, nothing valuable gets lost in the ever-expanding social media universe.

Your privacy and security are our top priorities. We use cutting-edge encryption technology to protect your data, ensuring that all your communications stay confidential and shielded from unauthorized access.

Experience a new standard of online connectivity and communication. Our app combines easy messaging, instant notifications, and engaging stories in one transformative platform. Join us and redefine how you connect, collaborate, and explore together!

Technical Architecture



The technical architecture of our social media app is designed around a client-server model, utilizing a REST API for initial client-server interactions. The frontend, acting as the client, integrates socket.io-client to establish real-time communication with the backend server. On the backend, socket.io and the Express.js framework handle server-side logic and facilitate real-time functionalities such as messaging, post uploads, and story uploads.

The frontend encompasses the user interface and presentation layer, and includes socket.io-client for maintaining a persistent socket connection with the server. This enables bidirectional communication, allowing for immediate updates and seamless user interaction.

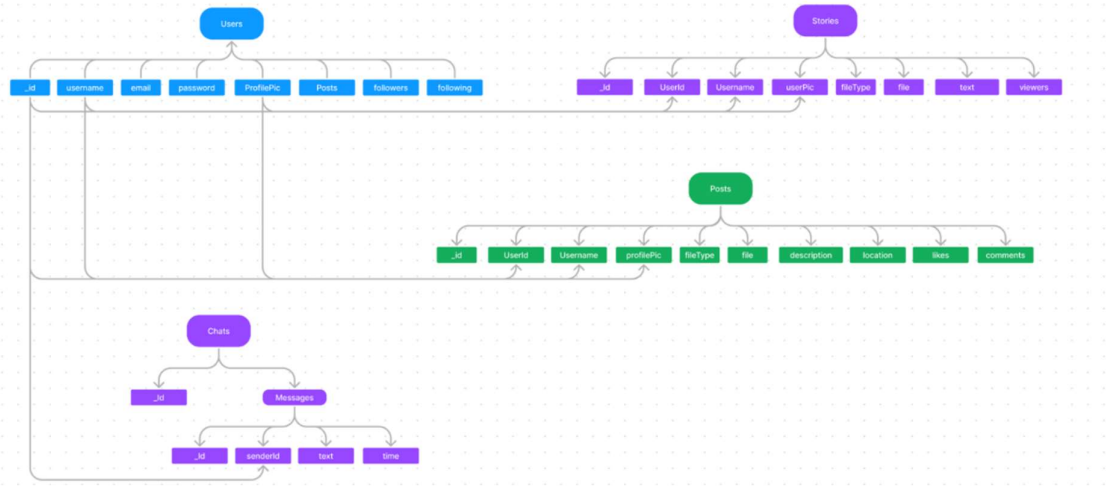
Authentication is managed through the REST API, which securely verifies user credentials and issues access tokens or session cookies. Once authenticated, the client uses a socket connection to access real-time features.

Real-time messaging is powered by the socket.io library, enabling instant exchange of messages, including text, emojis, and images, between users.

The app also supports real-time post and story uploads. Users can create and upload content—text, images, videos, or a mix of media—via the socket connection. The backend processes these uploads, ensuring they are linked to the correct user and made available for real-time viewing and interaction.

The backend employs Express.js to manage REST API endpoints, routing requests to appropriate controllers and services. User data, including profiles, posts, and stories, is stored and accessed using a database like MongoDB, ensuring efficient data management.

Overall, the integration of frontend, backend, REST API, socket.io, Express.js, and MongoDB creates a robust technical architecture. This setup supports real-time messaging, seamless content uploads, authentication, and secure data storage, delivering a dynamic and interactive social media experience.



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 JSON like 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.htm

✓ **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: Tshaan1104/SOCIAL_MEDIA_APP (github.com)

- **Install the required dependencies by running the following commands:**

1. **cd frontend**
2. **npm install**
3. **cd ../backend**
4. **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:6001**

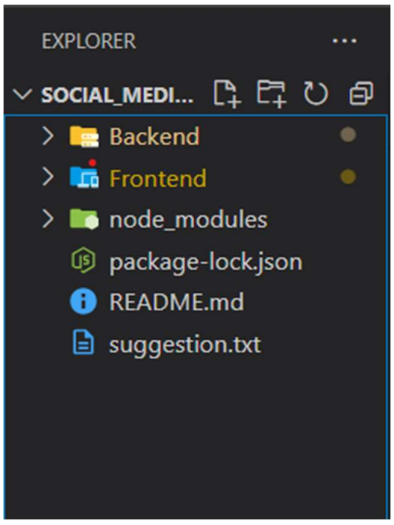
Roles & Responsibilities

User:

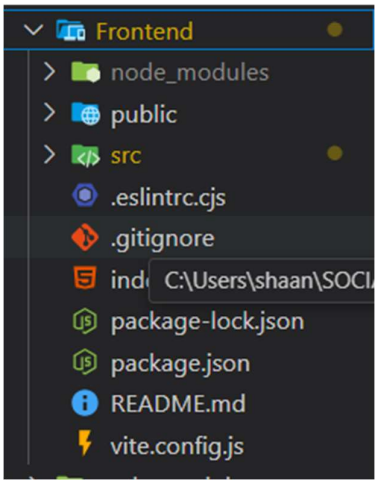
- **Create and manage a personal profile.**
- **Share posts, photos, videos, and stories with their network.**
- **Engage in conversations through comments, likes, and shares.**
- **Follow other users and discover new accounts, topics, and trends.**
- **Explore and discover new content, communities, 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:

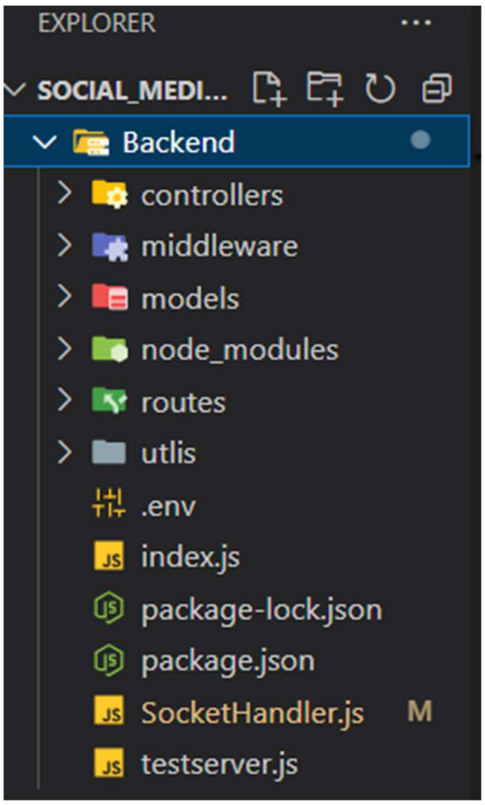
- **Inside the SANGAM(social media app) directory, we have the following folders**



- **Frontend Directory**



- **Backend Directory**



Project Flow:

Project demo:

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

Demo link: https://drive.google.com/file/d/1whsK69OTeUzy-6DI_cECQv3FteXRJLoc/view?usp=sharing

Use the code in: Tshaan1104/SOCIAL_MEDIA_APP (github.com)

Milestone 1: Project setup and configuration.

• Folder setup:

1. **Create frontend and backend folders:**
   - **Frontend**
   - **Backend**

• Installation of required tools:

Download the following to install necessary tools:

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

Open the backend folder to download necessary tools:

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

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.
   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

```
import express from "express";
import bodyParser from "body-parser";
import mongoose from "mongoose";
import cors from "cors";
import { Server } from "socket.io";
import http from "http";
import path from "path";
import dotenv from "dotenv";
import { fileURLToPath } from "url";
import SocketHandler from "./SocketHandler.js";
dotenv.config();

import authRoutes from "./routes/Route.js";
// import SocketHandler from "./SocketHandler.js";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const app = express();

app.use(express.json());

app.use(bodyParser.json({ limit: "30mb", extended: true }));
app.use(bodyParser.urlencoded({ limit: "30mb", extended: true }));
app.use(cors());

app.use("/api", authRoutes);
const server = http.createServer(app);

const io = new Server(server, {
  cors: {
    origin: "*",
    methods: ["GET", "POST", "DELETE", "PUT"],
  },
});

io.on("connection", (socket) => {
  console.log("User connected");
  SocketHandler(socket);
});
app.get("/", (req, res) => {
```

```
io.on("connection", (socket) => {
  console.log("User connected");
  SocketHandler(socket);
});
app.get("/", (req, res) => {
  res.send("Welcome to the AI");
});

console.log("JWT_SEeeCRET:", process.env.JWT_SECRET); // secret key

//MongoDB connection
//Mongoose Connection
const PORT = 6001;
mongoose
  .connect(
    "mongodb+srv://tshaan1104:Tiwari9536@mediaapp.tm4hd.mongodb.net/Social_Media_app",
    {}
  )
  .then(() => {
    server.listen(PORT, () => {
      console.log(`Server is running on port http://localhost:${PORT}/ `);
    });
  })
  .catch((err) => {
    console.log(`Error with DB connection: ${err}`);
  });
```

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.

- **Chats model**

```
import mongoose from 'mongoose';

const chatSchema=mongoose.Schema({
  _id:{
    type:String,
    require:true,
  },
  messages:{
    type:Array,
  }
});

const Chats = mongoose.model('chats', chatSchema);

export default Chats;
```

- **User Model**

```
import mongoose from 'mongoose';

// const mongoose = require('mongoose');

const userSchema = mongoose.Schema({
  username:{
    type: String,
    require:true
  },
  email:{
    type:String,
    require:true,
    unique:true
  },
  password:{
    type:String,
    require:true
  },
  profileImg:{
    type:String
  },
  about:{
    type:String
  },
  posts:{
    type:Array
  },
  followers:{
    type:Array
  },
  following:{
    type:Array
  }
});


const User = mongoose.model('users', userSchema);

export default User;
```

-**Stories Model**

```
import mongoose from 'mongoose';


// const mongoose = require('mongoose');

const storySchema = new mongoose.Schema({
  userId:{
    type: String
  },
  userName:{
    type:String,
  },
  userImg:{
    type:String
  },
  fileType:{
    type:String
  },
  file:{
    type:String
  },
  text:{
    type:String
  },
  viewers:{
    type:Array
  }
},{timestamps:true});


const Stories = mongoose.model('stories', storySchema);

export default Stories;
```

- **Posts Model**

```
import mongoose from 'mongoose';

// const mongoose = require('mongoose');

const postSchema = mongoose.Schema({
  userId:{
    type: String
  },
  userName:{
    type:String,
  },
  userImg:{
    type:String
  },
  fileType:{
    type:String
  },
  file:{
    type:String
  },
  description:{
    type:String
  },
  location:{
    type:String
  },
  likes:{
    type:Array
  },

  comments:{
    type:Array
  }
},{timestamps:true});


const Post = mongoose.model('posts', postSchema);

export default Post;
```

-

3. Authentication:

- **Backend:**

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

- **Register**

```
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import User from "../models/Users.js";
import dotenv from "dotenv";
import generateProfileImageURL from "../utlis/profileImage.js";

const generateToken = (id) => {
  const jwtSecret = process.env.JWT_SECRET;
  return jwt.sign({ id }, jwtSecret, {
    expiresIn: "30d",
  });
};

export const register = async (req, res) => {
  try {
    const { username, password, email, profileImg } = req.body;
    const salt = await bcrypt.genSalt();
    const passwordHash = await bcrypt.hash(password, salt);
    console.log("JWT_SECRET:", process.env.JWT_SECRET);

    const profileImage = profileImg || generateProfileImageURL(username);

    const newUser = new User({
      username,
      password: passwordHash,
      email,
      profileImg: profileImage,
    });

    const user = await newUser.save();
    const token = generateToken(user._id);

    const userdata = {
      _id: user._id,
      username: user.username,
      email: user.email,
      profileImg: user.profileImg,
      about: user.about,
      posts: user.posts,
      followers: user.followers,
      following: user.following,
```

- **Login**

```
    following: user.following,
    };

    res.status(200).json({ token, user: userdata });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

export const login = async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email: email });
    if (!user) return res.status(400).json({ msg: "User not found" });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ msg: "Invalid Credentials" });

    const token = generateToken(user._id);
    delete user.password;
    const userdata = {
      _id: user._id,
      username: user.username,
      email: user.email,
      profileImg: user.profileImg,
      about: user.about,
      posts: user.posts,
      followers: user.followers,
      following: user.following,
    };
    res.status(200).json({ token, user: userdata });

    console.log(token, userdata);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

- **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.

```jsx
import React, { useContext } from 'react'
import '../styles/login.css'
import { AuthenticationContext } from '../context/AuthenticationContextProvider';
const Register = ({setIsLoginBox}) => {
  const {setUsername, setEmail, setPassword, register} = useContext(AuthenticationContext);

  const handleRegister = async (e) =>{
    e.preventDefault();

    await register()
  }
  return (
    <form className="authForm">
        <h2>Register</h2>
        <div className="form-floating mb-3 authFormInputs">
            <input type="text" className="form-control" id="floatingInput"
            placeholder="username" onChange={(e)=> setUsername(e.target.value)} />
            <label htmlFor="floatingInput">Username</label>
        </div>
        <div className="form-floating mb-3 authFormInputs">
            <input type="email" className="form-control" id="floatingEmail"
            placeholder="name@example.com" onChange={(e)=> setEmail(e.target.value)} />
            <label htmlFor="floatingInput">Email address</label>
        </div>
        <div className="form-floating mb-3 authFormInputs">
            <input type="password" className="form-control" id="floatingPassword"
            placeholder="Password" onChange={(e)=> setPassword(e.target.value)} />
            <label htmlFor="floatingPassword">Password</label>
        </div>
        <button className="btn btn-primary" onClick={handleRegister}>
        <span className="btn-anim1"></span>
    <span className="btn-anim2"></span>
    <span className="btn-anim3"></span>
    <span className="btn-anim4"></span>Sign up</button>

        <p>Already registered? <span onClick={()=> setIsLoginBox(true)}>Login</span></p>
    </form>
  )
}
export default Register
```

- **Login**

```jsx
import React, { useContext } from 'react'
import '../styles/login.css'
import { AuthenticationContext } from '../context/AuthenticationContextProvider';

const Login = ({setIsLoginBox}) => {

  const {setEmail, setPassword, login} = useContext(AuthenticationContext);

  const handleLogin = async (e) =>{
    e.preventDefault();
    await login();
  }

  return (
    <form className="authForm">
        <h2>Login</h2>
        <div className="form-floating mb-3 authFormInputs">
            <input type="email" className="form-control" id="floatingInput"
            placeholder="name@example.com" onChange={(e) => setEmail(e.target.value)} />
            <label htmlFor="floatingInput">Email address</label>
        </div>
            <div className="form-floating mb-3 authFormInputs">
            <input type="password" className="form-control" id="floatingPassword"
            placeholder="Password" onChange={(e) => setPassword(e.target.value)} />
            <label htmlFor="floatingPassword">Password</label>
        </div>
        <button type="submit" className="btn btn-primary" onClick={handleLogin}>
        <span className="btn-anim1"></span>
    <span className="btn-anim2"></span>
    <span className="btn-anim3"></span>
    <span className="btn-anim4"></span>Sign in</button>

        <p>Not registered? <span onClick={()=> setIsLoginBox(false)}>Register</span></p>
    </form>
  )
}

export default Login
```

Context Api for Authentication:

```
import React, { createContext, useState } from 'react';
import axios from "axios";
import { useNavigate } from "react-router-dom";

export const AuthenticationContext = createContext();

const AuthenticationContextProvider = ({children}) => {

  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  const profileImg = 'https://cdn.pixabay.com/photo/2015/10/05/22/37/blank
  profile-picture-973460_960_720.png'


  const inputs = {username: username, email: email, password: password, profileImg: profileImg}


  const navigate = useNavigate();

  const login = async () =>{

    try{

      const loginInputs = {email: email, password: password}
        await axios.post('http://localhost:6001/api/login', loginInputs)
        .then( async (res)=>{
            console.log("holaads",res);
            localStorage.setItem('userToken', res.data.token);
            localStorage.setItem('userId', res.data.user._id);
            localStorage.setItem('username', res.data.user.username);
            localStorage.setItem('email', res.data.user.email);
            localStorage.setItem('profileImg', res.data.user.profileImg);
            localStorage.setItem('posts', res.data.user.posts);
            localStorage.setItem('followers', res.data.user.followers);
            localStorage.setItem('following', res.data.user.following);
            navigate('/');
        }).catch((err) =>{
            console.log(err);
        });
  }
}

const register = async () =>{

  try{
      await axios.post('http://localhost:6001/api/register', inputs)
      .then( async (res)=>{
        localStorage.setItem('userToken', res.data.token);
        localStorage.setItem('userId', res.data.user._id);
        localStorage.setItem('username', res.data.user.username);
        localStorage.setItem('email', res.data.user.email);
        localStorage.setItem('profileImg', res.data.user.profileImg);
        localStorage.setItem('posts', res.data.user.posts);
        localStorage.setItem('followers', res.data.user.followers);
        localStorage.setItem('following', res.data.user.following);
        navigate('/');
      }).catch((err) =>{
          console.log(err);
      });
```

```
  return (                                                    (property) register:
    <AuthenticationContext.Provider value={{login, register,
      logout, username, setUsername, email, setEmail, password,
      setPassword }} >{children}</AuthenticationContext.Provider>
  )
}

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.

```
import React, { createContext, useReducer, useState } from 'react'
import socketIoClient from 'socket.io-client';
export const GeneralContext = createContext();
const WS = 'http://localhost:6001';
const socket = socketIoClient(WS);
export const GeneralContextProvider = ({ children }) => {
    const [isCreatPostOpen, setIsCreatePostOpen] = useState(false);
    const [isCreateStoryOpen, setIsCreateStoryOpen] = useState(false);
    const [isNotificationsOpen, setNotificationsOpen] = useState(false);
    const [notifications, setNotifications] = useState([]);
    const [chatFirends, setChatFriends] = useState([]);
    const INITIAL_STATE = {
        chatId: 'null',
        user: {},
    };
    const userId = localStorage.getItem('userId');

    const chatReducer = (state, action) => {
        switch (action.type) {
            case "CHANGE_USER":
                return {
                    user: action.payload,                                                    any
                    chatId: userId > action.payload._id ? userId + action.payload._id : action.payload._id + userId
                }
            default:
                return state;
        }
    };
    const [state, dispatch] = useReducer(chatReducer, INITIAL_STATE);
    return (
        <GeneralContext.Provider value={{ socket, isCreatPostOpen, setIsCreatePostOpen,
            isCreateStoryOpen, setIsCreateStoryOpen, isNotificationsOpen, setNotificationsOpen,
            notifications, setNotifications, chatFirends, setChatFriends, chatData: state,
            dispatch }}>{children}</GeneralContext.Provider>
    )
}
```

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.

```jsx
import React, { useContext, useState, useEffect } from 'react';
import '../styles/CreatePosts.css';
import { RxCross2 } from 'react-icons/rx';
import { GeneralContext } from '../context/GeneralContextProvider';
import axios from 'axios';
import { ref, uploadBytesResumable, getDownloadURL } from 'firebase/storage';
import { storage } from '../firebase.js';
import { v4 as uuidv4 } from 'uuid';

const CreatePost = () => {
    const { isCreatPostOpen, setIsCreatePostOpen } = useContext(GeneralContext);

    const [postType, setPostType] = useState('photo');
    const [postDescription, setPostDescription] = useState('');
    const [postLocation, setPostLocation] = useState('');
    const [postFile, setPostFile] = useState(null);
    const [uploadProgress, setUploadProgress] = useState(0);

    // Use useEffect to handle state updates based on upload progress
    useEffect(() => {
        if (uploadProgress === 100) {
            // Reset form fields and close modal
            setPostDescription('');
            setPostLocation('');
            setPostFile(null);
            setIsCreatePostOpen(false);
            setUploadProgress(0); // Reset progress
        }
    }, [uploadProgress, setIsCreatePostOpen]);

    const handlePostUpload = async (e) => {
        e.preventDefault();

        if (!postFile) return; // Guard clause for no file

        const storageRef = ref(storage, uuidv4());
        const uploadTask = uploadBytesResumable(storageRef, postFile);

        uploadTask.on('state_changed',
            (snapshot) => {
                setUploadProgress((snapshot.bytesTransferred / snapshot.totalBytes) * 100);
            (error) => {
                console.log(error);
            },
            () => {
                getDownloadURL(uploadTask.snapshot.ref).then(async (downloadURL) => {
                    console.log('File available at', downloadURL);

                    try {
                        const inputs = {
                            userId: localStorage.getItem('userId'),
                            userName: localStorage.getItem('username'),
                            userPic: localStorage.getItem('profileImg'),
                            fileType: postType,
                            file: downloadURL,
                            description: postDescription,
                            location: postLocation,
                            comments: { "New user": "This is my first comment" }
                        };
                        await axios.post('http://localhost:6001/api/createpost', inputs);
                    } catch (err) {
                        console.log(err);
                    }
                });
            }
        );
    };

    return (
        <>
            <div className="createPostModalBg" style={{ display: isCreatPostOpen ? 'contents' : 'none' }}>
                <div className="createPostContainer">
                    <RxCross2 className='closeCreatePost' onClick={() => setIsCreatePostOpen(false)} />
                    <h2 className="createPostTitle">Create post</h2>
                    <hr className="createPostHr" />
                    <div className="createPostBody">
                        <form>
                            <select className="form-select" aria-label="Select Post Type"
                            onChange={(e) => setPostType(e.target.value)}>
                                <option defaultValue='photo'>Choose post type</option>
```

```
                <div className="createPostBody">
                    <form>
                        <select className="form-select" aria-label="Select Post Type"
                        onChange={(e) => setPostType(e.target.value)}>
                            <option defaultValue='photo'>Choose post type</option>
                            <option value="photo">Photo</option>
                            <option value="video">Video</option>
                        </select>
                        <div className="uploadBox">
                            <input type="file" name="PostFile" id="uploadPostFile" onChange={(e) => setPostFile(e.target.files[0])} />
                        </div>
                        <div className="form-floating mb-3 authFormInputs descriptionInput">
                            <input type="text" className="form-control descriptionInput" id="floatingDescription" placeholder="Description" onChange={(e)
                            => setPostDescription(e.target.value)} value={postDescription} />
                            <label htmlFor="floatingDescription">Description</label>
                        </div>
                        <div className="form-floating mb-3 authFormInputs postLocation">
                            <input type="text" className="form-control postLocation" id="floatingLocation" placeholder="Location" onChange={(e) =>
                            setPostLocation(e.target.value)} value={postLocation} />
                            <label htmlFor="floatingLocation">Location</label>
                        </div>
                        {uploadProgress ?
                            <button disabled>Uploading... {Math.round(uploadProgress)}%</button>
                            :
                            <button onClick={handlePostUpload}>Upload</button>
                        }
                    </form>
                </div>
            </div>
        </div>
    </>
  );
};

export default CreatePost;
```

- **Backend**

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

```
import Post from "../models/Post.js";

export const createpost=async (req, res) => {
    try {
        const newPost=new Post(req.body);
        const post =await newPost.save();
        res.status(200).json(post);

    }
    catch (err) {
        res.status(500).json({error:err});
}}
```

5.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.

```jsx
import React, { useContext, useEffect, useState } from 'react';
import '../styles/Posts.css';
import { AiOutlineHeart, AiTwotoneHeart } from "react-icons/ai";
import { BiCommentDetail } from "react-icons/bi";
import { FiSend } from "react-icons/fi";
import { FaGlobeAmericas } from "react-icons/fa";
import {IoIosPersonAdd} from 'react-icons/io'
import postImg from '../images/bicyle sample.jpg';
import axios from 'axios';
import { GeneralContext } from '../context/GeneralContextProvider';
import { useNavigate } from 'react-router-dom';

const Post = () => {

    const navigate = useNavigate();

    const {socket} = useContext(GeneralContext);


    const [posts, setPosts] = useState([]);

    useEffect(() => {
        fetchPosts();
      }, []);

      const fetchPosts = async () => {
        try {
          const response = await axios.get('http://localhost:6001/api/fetchAllPosts');
          const fetchedPosts = response.data;
          setPosts(fetchedPosts);
        } catch (error) {
          console.error(error);
        }
      };



    //   Like

    const handleLike = (userId, postId) =>{
        socket.emit('postLiked', {userId, postId});
```

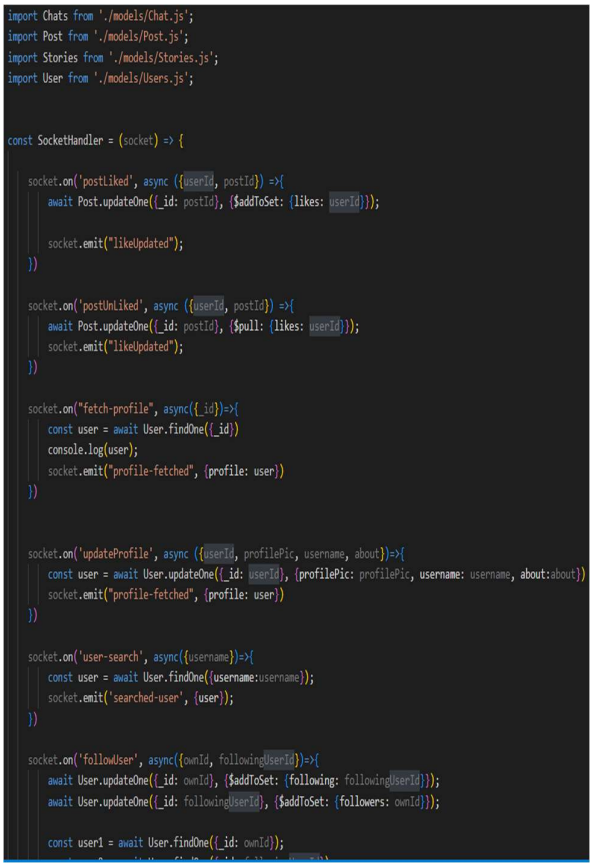
```jsx
    }

    const handleUnLike = (userId, postId) =>{
        socket.emit('postUnLiked', {userId, postId});

    }


    useEffect(()=>{
        socket.on("likeUpdated", ()=>{
            // alert("likedd");
        })

        socket.on('userFollowed', ({following})=>{

            localStorage.setItem('following', following);

        })

    },[socket])


    const handleFollow = async (userId) =>{
        // ownId = current user Id
        // followingUserId = user you want to follow
        socket.emit('followUser', {ownId: localStorage.getItem('userId'), followingUserId: userId});
    }



    const [comment, setComment] = useState('');

    const handleComment = (postId, username)=>{
        socket.emit('makeComment', {postId, username, comment});
    }
```

```
<div className="post" key={post._id}>

    <div className="postTop">
        <div className="postTopDetails">
            <img src={post.userPic} alt="" className="userpic" />
            <h3 className="usernameTop" onClick={()=> navigate(`/profile/${post.userId}`)}>{post.userName}</h3>
        </div>

        {localStorage.getItem('following').includes(post.userId) || localStorage.getItem('userId') === post.userId ?

        <></>
        :
        <IoIosPersonAdd style={{cursor: "pointer"}} id='addFriendInPost' onClick={()=>handleFollow(post.userId)} />
        }

    </div>

    { post.fileType === 'photo'?

            <img src={post.file} className='postimg' alt="" />

            :

            <video id="videoPlayer" className='postimg' controls autoPlay muted>
                <source src={post.file} />
            </video>

            }

    <div className="postReact">
        <div className="supliconcol">

            {
                post.likes.includes(localStorage.getItem('userId')) ?

                <AiTwotoneHeart className='support reactbtn'  onClick={() => handleUnLike(localStorage.getItem('userId'), post._id)}/>

                :

                <AiOutlineHeart className='support reactbtn'  onClick={() => handleLike(localStorage.getItem('userId'), post._id)}/>
```

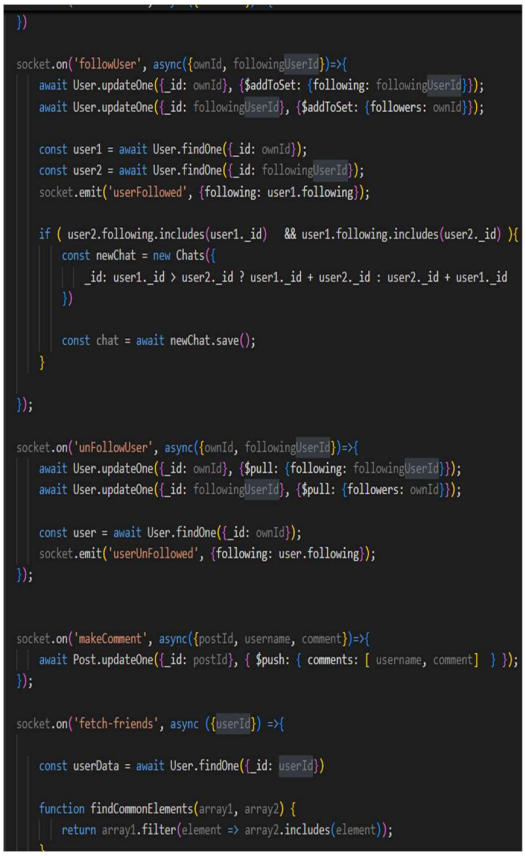
```
            <label htmlFor="support" className='supportCount'>{post.likes.length}</label>
        </div>
        {/* <BiCommentDetail className='comment reactbtn' /> */}
        {/* <FiSend className='share reactbtn' onClick={()=> {handleShare(post)}} /> */}
        <div className="placeiconcol">
            <FaGlobeAmericas className='placeicon reactbtn' name='place' />
            <label htmlFor="place" className='place'>{post.location}</label>
        </div>
    </div>



    <div className="detail">
        <div className='descdataWithBtn'>
            <label htmlFor='username' className="desc labeldata" id='desc'>
                <span style={{fontWeight: 'bold'}}>
                    {post.userName}
                </span>
                        {post.description}
            </label>
        </div>
    </div>
    <div className="commentsContainer">
        <div className="makeComment">
            <input type="text" placeholder='type something...' onChange={(e)=>setComment(e.target.value)}/>
            {comment.length === 0 ?
                <button className='btn btn-primary' disabled>comment</button>
            :
                <button className='btn btn-primary' onClick={()=>handleComment(post._id, localStorage.getItem('username'))} >comment</button>
            }
        </div>
        <div className="commentsBody">
            <div className="comments">
                {post.comments.map((comment,index)=>{
                    return(
```

```
                        <button className='btn btn-primary' onClick={()=>handleCommer
                    }
                </div>
                <div className="commentsBody">
                    <div className="comments">
                        {post.comments.map((comment,index)=>{
                            return(

                                <p key={index}><b>{comment[0]}</b> {comment[1]}</p>
                            )
                        })}
                    </div>
                </div>
            </div>
        </div>
        </div>
        )

    }) : <></>}

    </div>
  )
}

export default Post
```

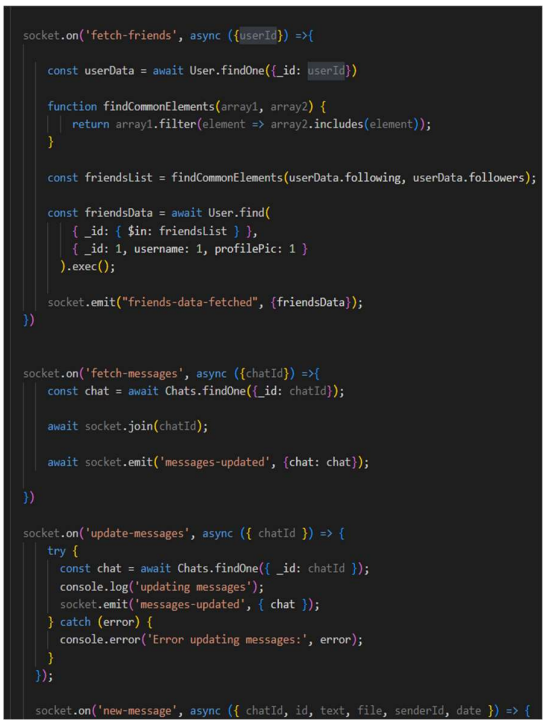
7. Socket Handling in backend:



```
import Chats from './models/Chat.js';
import Post from './models/Post.js';
import Stories from './models/Stories.js';
import User from './models/Users.js';


const SocketHandler = (socket) => {

    socket.on('postLiked', async ({userId, postId}) =>{
        await Post.updateOne({_id: postId}, {$addToSet: {likes: userId}});

        socket.emit("likeUpdated");
    })

    socket.on('postUnLiked', async ({userId, postId}) =>{
        await Post.updateOne({_id: postId}, {$pull: {likes: userId}});
        socket.emit("likeUpdated");
    })

    socket.on("fetch-profile", async({_id})=>{
        const user = await User.findOne({_id})
        console.log(user);
        socket.emit("profile-fetched", {profile: user})
    })


    socket.on('updateProfile', async ({userId, profilePic, username, about})=>{
        const user = await User.updateOne({_id: userId}, {profilePic: profilePic, username: username, about:about})
        socket.emit("profile-fetched", {profile: user})
    })

    socket.on('user-search', async({username})=>{
        const user = await User.findOne({username:username});
        socket.emit('searched-user', {user});
    })

    socket.on('followUser', async({ownId, followingUserId})=>{
        await User.updateOne({_id: ownId}, {$addToSet: {following: followingUserId}});
        await User.updateOne({_id: followingUserId}, {$addToSet: {followers: ownId}});

        const user1 = await User.findOne({_id: ownId});
```



```
    })

    socket.on('followUser', async({ownId, followingUserId})=>{
        await User.updateOne({_id: ownId}, {$addToSet: {following: followingUserId}});
        await User.updateOne({_id: followingUserId}, {$addToSet: {followers: ownId}});

        const user1 = await User.findOne({_id: ownId});
        const user2 = await User.findOne({_id: followingUserId});
        socket.emit('userFollowed', {following: user1.following});

        if ( user2.following.includes(user1._id)  && user1.following.includes(user2._id) ){
            const newChat = new Chats({
                _id: user1._id > user2._id ? user1._id + user2._id : user2._id + user1._id
            })

            const chat = await newChat.save();
        }

    });

    socket.on('unFollowUser', async({ownId, followingUserId})=>{
        await User.updateOne({_id: ownId}, {$pull: {following: followingUserId}});
        await User.updateOne({_id: followingUserId}, {$pull: {followers: ownId}});

        const user = await User.findOne({_id: ownId});
        socket.emit('userUnFollowed', {following: user.following});
    });


    socket.on('makeComment', async({postId, username, comment})=>{
        await Post.updateOne({_id: postId}, { $push: { comments: [ username, comment]  } });
    });

    socket.on('fetch-friends', async ({userId}) =>{

        const userData = await User.findOne({_id: userId})

        function findCommonElements(array1, array2) {
            return array1.filter(element => array2.includes(element));
```



```
    socket.on('fetch-friends', async ({userId}) =>{

        const userData = await User.findOne({_id: userId})

        function findCommonElements(array1, array2) {
            return array1.filter(element => array2.includes(element));
        }

        const friendsList = findCommonElements(userData.following, userData.followers);

        const friendsData = await User.find(
            { _id: { $in: friendsList } },
            { _id: 1, username: 1, profilePic: 1 }
          ).exec();

        socket.emit("friends-data-fetched", {friendsData});
    })


    socket.on('fetch-messages', async ({chatId}) =>{
        const chat = await Chats.findOne({_id: chatId});

        await socket.join(chatId);

        await socket.emit('messages-updated', {chat: chat});

    })

    socket.on('update-messages', async ({ chatId }) => {
        try {
          const chat = await Chats.findOne({ _id: chatId });
          console.log('updating messages');
          socket.emit('messages-updated', { chat });
        } catch (error) {
          console.error('Error updating messages:', error);
        }
      });

      socket.on('new-message', async ({ chatId, id, text, file, senderId, date }) => {
```

```
    socket.on('new-message', async ({ chatId, id, text, file, senderId, date }) => {
      try {
        await Chats.findOneAndUpdate(
          { _id: chatId },
          { $addToSet: { messages: { id, text, file, senderId, date } } },
          { new: true }
        );

        const chat = await Chats.findOne({ _id: chatId });
        console.log(chat);
        socket.emit('messages-updated', { chat });
        socket.broadcast.to(chatId).emit('message-from-user');
      } catch (error) {
        console.error('Error adding new message:', error);
      }
    });


    socket.on('chat-user-searched', async ({ownId, username})=>{
      const user = await User.findOne({username:username});
      if(user){
        if (user.followers.includes(ownId) && user.following.includes(ownId)){

          socket.emit('searched-chat-user', {user});

        }else{
          socket.emit('no-searched-chat-user');
        }
      }else{
        socket.emit('no-searched-chat-user');
      }
    });


    socket.on('fetch-all-posts', async()=>{
      const posts = await Post.find();
      socket.emit('all-posts-fetched', {posts});
    })
```

```
    socket.on('fetch-all-posts', async()=>{
      const posts = await Post.find();
      socket.emit('all-posts-fetched', {posts});
    })


    socket.on('delete-post', async ({postId}) =>{
      await Post.deleteOne({_id: postId});
      const posts = await Post.find();
      socket.emit('post-deleted', {posts});
    });


    socket.on('create-new-story', async({userId, username, userPic, fileType, file, text})=>{
      const newStory = new Stories({userId, username, userPic, fileType, file, text});
      await newStory.save();
    })

    socket.on('fetch-stories', async()=>{
      const stories = await Stories.find();
      socket.emit('stories-fetched', {stories});
    });

    socket.on('story-played', async ({storyId, userId})=>{
      await Stories.updateOne({_id: storyId}, {$addToSet: {viewers: userId}});

    })
}

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.

```
import User from "../models/Users.js";
import Stories from "../models/Stories.js";
import Post from "../models/Post.js";
export const fetchAllPosts=async (req, res, next) =>{
 try {
    const posts=await Post.find().sort({_id:-1});
    res.json(posts);
 }
 catch (err) {
    console.error(err);
    res.status(500).json({error:'Server error'});
 }}
export const fetchUserName=async (req, res) =>{
        try{
            const userId=req.body.userId;
            const user = await User.findById(userId);
            console.log(userId);
            res.status(200).json(user);
        }
        catch(error){
            console.log(error);
            res.status(500).json({error: error.message});
        }}
export const fetchUserImg=async(req,res)=>{
    try {
        const userId=req.body.userId;
        const user= await User.findOne({_id:userId});
        console.log(userId);
        res.status(200).json(user);
    }catch(error){
        console.error(error);
        res.status(500).json({error:'Server error'});
    }}
export const fetchAllStories =async(req,res)=>{
    try{
        const stories = await Stories.find();
        res.status(200).json(stories);
    }catch(error){
        console.log(error);
        res.status(500).json({error:'Server error'});
    }}
```

9. Create Stories:

Frontend:

Create New Story

```
import React, { useContext, useState, useEffect } from 'react';
import '../styles/CreatePosts.css';
import { GeneralContext } from '../context/GeneralContextProvider';
import { RxCross2 } from 'react-icons/rx';
import { ref, uploadBytesResumable, getDownloadURL } from "firebase/storage";
import { storage } from '../firebase.js';
import { v4 as uuidv4 } from 'uuid';

const CreateStory = () => {
    const { socket, isCreateStoryOpen, setIsCreateStoryOpen } = useContext(GeneralContext);

    const [storyType, setStoryType] = useState('photo');
    const [storyDescription, setStoryDescription] = useState('');
    const [storyFile, setStoryFile] = useState(null);
    const [uploadProgress, setUploadProgress] = useState(null);

    useEffect(() => {
        if (uploadProgress === 100) {
            setStoryDescription('');
            setStoryFile(null);
            setIsCreateStoryOpen(false);
            setUploadProgress(null);
        }
    }, [uploadProgress, setIsCreateStoryOpen]);

    const handleStoryUpload = async (e) => {
        e.preventDefault();

        const storageRef = ref(storage, uuidv4());
        const uploadTask = uploadBytesResumable(storageRef, storyFile);

        uploadTask.on('state_changed',
            (snapshot) => {
                setUploadProgress((snapshot.bytesTransferred / snapshot.totalBytes) * 100);
            },
            (error) => {
                console.log(error);
            },
            () => {
                getDownloadURL(uploadTask.snapshot.ref).then(async (downloadURL) => {
                    console.log('File available at', downloadURL);
```

```
            () => {
                getDownloadURL(uploadTask.snapshot.ref).then(async (downloadURL) => {
                    console.log('File available at', downloadURL);

                    try {
                        await socket.emit('create-new-story', {
                            userId: localStorage.getItem('userId'),
                            username: localStorage.getItem('username'),
                            userPic: localStorage.getItem('profileImg'),
                            fileType: storyType,
                            file: downloadURL,
                            text: storyDescription,
                        });
                        setUploadProgress(100);
                    } catch (err) {
                        console.log(err);
                    }
                });
            }
        );
    };
```

```
    return (
        <div className="createPostModalBg" style={isCreateStoryOpen ? { display: 'contents' } : { display: 'none' }}>
            <div className="createPostContainer">
                <RxCross2 className='closeCreatePost' onClick={() => setIsCreateStoryOpen(false)} />
                <h2 className="createPostTitle">Add new story</h2>
                <hr classNamе="createPostHr" />
                <div className="createPostBody">
                    <form>
                        <select className="form-select" aria-label="Select Post Type" onChange={(e) => setStoryType(e.target.value)}>
                            <option defaultValue='photo'>Choose post type</option>
                            <option value="photo">Photo</option>
                            <option value="video">Video</option>
                        </select>
                        <div className="uploadBox">
                            <input type="file" name="PostFile" id="uploadPostFile" onChange={(e) => setStoryFile(e.target.files[0])} />
                        </div>
                        <div className="form-floating mb-3 authFormInputs descriptionInput">
                            <input type="text" className="f                                                    ="Description" onChange={(e) =>
                            setStoryDescription(e.target.va
                            <label htmlFor="floatingDescrip
                        </div>
                        {uploadProgress !== null ?
                            <button disabled>Uploading... {Math.round(uploadProgress)}%</button>
                            :
                            <button onClick={handleStoryUpload}>Upload</button>
                        }
                    </form>
                </div>
            </div>
        </div>
    );
};

export default CreateStory;
```

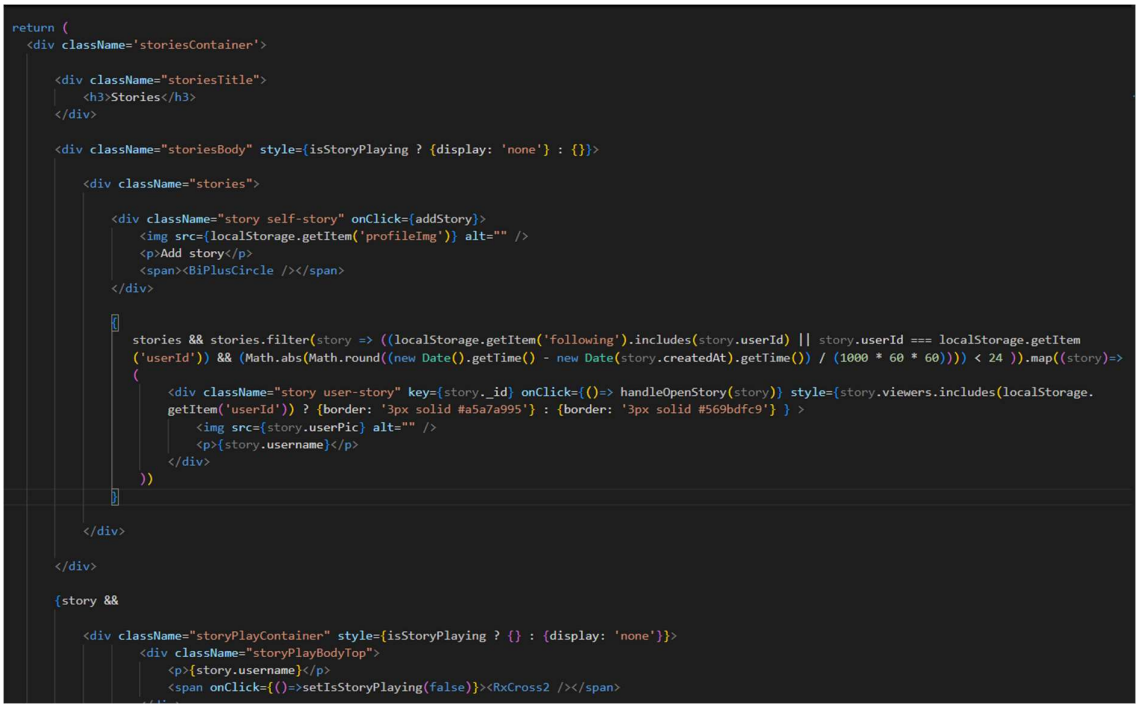
See Real World Examples From GitHub

var Math: Math

An intrinsic object that provides basic mathematics functionality and constants.

- **Stories**

```
import React, { useContext, useEffect, useState } from 'react'
import '../styles/Stories.css'
import { BiPlusCircle } from 'react-icons/bi'
import { GeneralContext } from '../context/GeneralContextProvider';
import axios from 'axios';
import {RxCross2} from 'react-icons/rx'

const Stories = () => {

    const {socket, isCreateStoryOpen, setIsCreateStoryOpen} = useContext(GeneralContext);

    const [stories, setStories] = useState([])
    const [isStoryPlaying, setIsStoryPlaying] = useState(false);

    const [story, setStory] = useState();

    const addStory = async () =>{
        setIsCreateStoryOpen(true)
    }

    useEffect(() => {
        fetchStories();
      }, []);

      const fetchStories = async () => {
        try {

            const response = await axios.get('http://localhost:6001/api/fetchAllStories');
            setStories(response.data)
            console.log(response.data[0])
        } catch (error) {
          console.error(error);
        }
      };

      const handleOpenStory = async (story) =>{

        setStory(story);
        await socket.emit('story-played', {storyId: story._id, userId: localStorage.getItem('userId')});
        setIsStoryPlaying(true);
```



```
return (
  <div className='storiesContainer'>

    <div className="storiesTitle">
        <h3>Stories</h3>
    </div>

    <div className="storiesBody" style={isStoryPlaying ? {display: 'none'} : {}}>

        <div className="stories">

            <div className="story self-story" onClick={addStory}>
                <img src={localStorage.getItem('profileImg')} alt="" />
                <p>Add story</p>
                <span><BiPlusCircle /></span>
            </div>

            {
              stories && stories.filter(story => ((localStorage.getItem('following').includes(story.userId) || story.userId === localStorage.getItem
              ('userId')) && (Math.abs(Math.round((new Date().getTime() - new Date(story.createdAt).getTime()) / (1000 * 60 * 60)))) < 24 )).map((story)=>
              (
                  <div className="story user-story" key={story._id} onClick={()=> handleOpenStory(story)} style={story.viewers.includes(localStorage.
                  getItem('userId')) ? {border: '3px solid #a5a7a995'} : {border: '3px solid #569bdfc9'} } >
                      <img src={story.userPic} alt="" />
                      <p>{story.username}</p>
                  </div>
              ))
            }

        </div>

    </div>

    {story &&

        <div className="storyPlayContainer" style={isStoryPlaying ? {} : {display: 'none'}}>
                <div className="storyPlayBodyTop">
                    <p>{story.username}</p>
                    <span onClick={()=>setIsStoryPlaying(false)}><RxCross2 /></span>
```

```
    {story &&

        <div className="storyPlayContainer" style={isStoryPlaying ? {} : {display: 'none'}}>
                <div className="storyPlayBodyTop">
                    <p>{story.username}</p>
                    <span onClick={()=>setIsStoryPlaying(false)}><RxCross2 /></span>
                </div>
                <div className="storyPlayBodyContent">
                    {story.fileType === 'photo' ?
                        <img src={story.file} alt="" />
                    :
                        <video id="videoPlayer" className='postimg' controls autoPlay muted>
                            <source src={story.file} />
                        </video>
                    }

                    <p>{story.text}</p>
                </div>
        </div>
    }



  </div>
  )
}

export default Stories
```

- **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):**

```
import React from 'react'
import '../styles/Chat.css'
import HomeLogo from '../components/HomeLogo'
import Navbar from '../components/Navbar'
import Sidebar from '../components/chat/Sidebar'
import UserChat from '../components/chat/Userchat'

const Chat = () => {
  return (
    <div className='chatPage'>
      {/* <HomeLogo /> */}
      <Navbar />

    <div className="home">

      <Sidebar  />
      <UserChat />

    </div>
    </div>
  )
}

export default Chat
```

- **Sidebar:**

```
import React from 'react'
import Search from './Search'
import Chats from './Chats'
// import Navbar from './'

const Sidebar = () => {
  return (
    <div className='sidebar'  >

      {/* <Navbar /> */}

      <Search />
      <Chats />

    </div>
  )
}

export default Sidebar
```

- **Chats**
  To display all the other chats

```
import React, { useContext, useEffect, useState } from 'react'
import {GeneralContext} from '../../context/GeneralContextProvider';
const Chats = () => {
  const {socket, chatFirends, setChatFriends, dispatch, chatData} = useContext(GeneralContext)
  const userId = localStorage.getItem('userId');
  // const [friendsData, setFriendsData] = useState([])
  useEffect(()=>{
    socket.emit('fetch-friends', {userId});
    socket.on("friends-data-fetched", ({friendsData})=>{
      setChatFriends(friendsData);
    });
  },[])
  const handleSelect = (data) =>{
    dispatch({type:"CHANGE_USER", payload: data});
    console.log(chatData);
  }
  useEffect(()=>{
    if(chatData.chatId !== null){
      socket.emit('fetch-messages', {chatId: chatData.chatId})

    }
  }, [chatData])
  return (
    <div className='chats'>
      {chatFirends.map((data)=>{
    return(
      <div className="userInfo" key={data._id} onClick={()=> handleSelect(data)} >
        <img src={data.profileImg} alt="" />
        <div className="userChatInfo">
          <span>{data.username}</span>
        </div>
      </div>
    )

  })}
      </div>
  )
}

export default Chats
```

- **UserChat:**

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

```
import {BiArrowBack} from 'react-icons/bi';
import Input from './Input';
import Messages from './Messages';
import { GeneralContext } from '../../context/GeneralContextProvider';

const UserChat = () => {
  const {chatData} = useContext(GeneralContext);
  return (
    <div className='chat'>
      {
        chatData.user &&
      <div className="chatInfo">
        <img src={chatData.user?.profileImg} alt="" />
        <span>{chatData.user.username}</span>

      </div>
    }
      <Messages />
      <Input />
    </div>
  )
}
export default UserChat
```

- **Input:**

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

```jsx
import React, { useContext, useState } from 'react'
import { BiImageAdd } from 'react-icons/bi'
import { GeneralContext } from '../../context/GeneralContextProvider'
import {v4 as uuid} from 'uuid';
import { getDownloadURL, ref, uploadBytesResumable } from 'firebase/storage';
import { storage } from '../../firebase';
import axios from 'axios';

const Input = () => {

    const {socket, chatData} = useContext(GeneralContext);

    const [text, setText] = useState('');
    const [file, setFile] = useState(null);

    const [uploadProgress, setUploadProgress] = useState();

    const userId = localStorage.getItem('userId');


    const handleSend = async () =>{

     if (file){

       const storageRef = ref(storage, uuid());
       const uploadTask = uploadBytesResumable(storageRef, file);

       uploadTask.on('state_changed',
       (snapshot) => {
           setUploadProgress((snapshot.bytesTransferred / snapshot.totalBytes) * 100);
       },
       (error) => {
           console.log(error);
       },
       () => {
           getDownloadURL(uploadTask.snapshot.ref).then( async (downloadURL) => {
           console.log('File available at', downloadURL);

           try{
             let date = new Date()
             await socket.emit('new-message', {chatId: chatData.chatId ,id: uuid(), text: text, file: downloadURL, senderId: userId, date: date});
```

```jsx
             setText('');
             setFile(null);
           }catch(err){
               console.log(err);
           }


           });
       }
       );

     }else{

       let date = new Date()
       await socket.emit('new-message', {chatId: chatData.chatId ,id: uuid(), text: text,file: '', senderId: userId, date: date});
       setText('');
     }

   }

 return (
   <div className='input' >
     <input type="text" placeholder='type something...' onChange={e => setText(e.target.value)} value={text} />
     <div className="send">
       <input type="file" style={{display : 'none'}} id='file' onChange={e=> setFile(e.target.files[0])} />
       <label htmlFor="file" style={{display:'flex'}}>
         <BiImageAdd />
         <p style={{fontSize: '12px'}}>{uploadProgress ? Math.floor(uploadProgress) + '%' : ''}</p>
       </label>
       <button onClick={handleSend} >Send</button>
     </div>
   </div>
 )
}

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.

```jsx
import React, { useContext, useEffect, useState } from 'react'
import Message from './Message'
import { GeneralContext } from '../../context/GeneralContextProvider';
const Messages = () => {
  const {socket} = useContext(GeneralContext)
  const [messages, setMessages] = useState([]);
  const {chatData} = useContext(GeneralContext);
  useEffect(() => {
    const handleMessagesUpdated = ({ chat }) => {
      console.log('chat', chat);
      if (chat) {
        setMessages(chat.messages);
      }};
    const handleNewMessage = async () => {
      if (chatData.chatId) {
        console.log('new message', chatData.chatId);
        socket.emit('update-messages', { chatId: chatData.chatId });
      } else {
        console.log('chatData.chatId is null or undefined');
      }};
    if (chatData.chatId) {
      socket.on('messages-updated', handleMessagesUpdated);
      socket.on('message-from-user', handleNewMessage);
    }
    return () => {
      // Clean up event listeners when the component unmounts
      socket.off('messages-updated', handleMessagesUpdated);
      socket.off('message-from-user', handleNewMessage);
    };
  }, [socket, chatData]);
  return (
    <div className='messages' >
      {messages.length > 0 &&  messages.map((message)=>(

        <Message message={message} key={message.id} />
      ))
      }
</div>
  )
}
export default Messages
```

- **Message:**

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

```jsx
import React, { useContext, useEffect, useRef } from 'react'
import { GeneralContext } from '../../context/GeneralContextProvider';

const Message = ({message}) => {

  const {chatData} = useContext(GeneralContext);

  const ref = useRef();

  let date = new Date(message.date);

  useEffect(() => {
    ref.current?.scrollIntoView({behavior:'smooth'})
  }, [message]);

  const userId = localStorage.getItem('userId');
  return (
    <div>
      <div ref={ref} className={`message ${message.senderId === userId ? "owner" : ""}`}>
      <div className="messageInfo">
        <img src={message.senderId === userId ? localStorage.getItem('profileImg') : chatData.user.profileImg} alt="" />
        <span>{ date.getHours() < 12 ?  date.getHours() + ':' + date.getMinutes() + ' AM' : date.getHours()-12 + ':' + date.getMinutes() + ' PM' }</span>
      </div>
      <div className="messageContent">
        <p>{message.text}</p>
        {message.file && <img src={message.file} alt="" />}

      </div>
    </div>
    </div>
  )
}

export default Message
```

- **Navbar**

```jsx
import React, { useContext, useEffect, useState } from 'react';
import "../styles/Navbar.css";
import { BiHomeAlt } from "react-icons/bi";
import { BsChatSquareText } from "react-icons/bs";
import { CgAddR } from "react-icons/cg";
import { TbNotification } from "react-icons/tb";
import navProfile from '../images/rmr block logo.png';
import { GeneralContext } from '../context/GeneralContextProvider';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';

const Navbar = () => {

  const {isCreatPostOpen, setIsCreatePostOpen, setIsCreateStoryOpen, isNotificationsOpen, setNotificationsOpen} = useContext(GeneralContext);

  const navigate = useNavigate();

  const profileImg = localStorage.getItem('profileImg');
  const userId = localStorage.getItem('userId');


  return (
    <>
    <div className="Navbar">
        <BiHomeAlt className="homebtn btns" onClick={()=> navigate('/')} />
        <BsChatSquareText  className="chatbtn btns" onClick={()=> navigate('/chat')} />
        <CgAddR className="createPostbtn btns" onClick={()=> {setIsCreatePostOpen(!isCreatPostOpen); setIsCreateStoryOpen(false)}} />
        <TbNotification className="Notifybtn btns" onClick={()=> setNotificationsOpen(!isNotificationsOpen)}/>
        <img  src={profileImg} alt="" onClick={()=> navigate(`/profile/${userId}`)} />
    </div>



    </>
  )
}

export default Navbar
```

Logo & User Search

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

```jsx
import React from 'react'
import '../styles/SearchContainer.css';
import { useNavigate } from 'react-router-dom';


const Search = ({searchedUser, setSearchedUser}) => {

    const navigate = useNavigate();
  return (
    <div className="searchContainer">

        {searchedUser &&  <div className="searchedUserInfo" onClick={()=> {navigate(`/profile/${searchedUser._id}`);  setSearchedUser();}} >
                    <img src={searchedUser.profileImg} alt="" />
                    <div className="searchedUserChatInfo">
                      <span>{searchedUser.username}</span>
                    </div>
                  </div>
                }

    </div>
  )
}

export default Search
```

Routes (Backend)

```
import express from 'express';
import {login,register} from '../controllers/Auth.js';
import {createpost} from '../controllers/createPost.js';
import {fetchAllPosts,fetchAllStories,fetchUserImg,fetchUserName} from '../controllers/Posts.js';

const router =express.Router();

router.post('/login', login);
router.post('/register', register);
router.post('/createpost', createpost);
router.get('/fetchAllPosts', fetchAllPosts);
router.get('/fetchAllStories', fetchAllStories);
router.get('/fetchUserImg', fetchUserImg);
router.get('/fetchUserName', fetchUserName);

export default router;
```

For More Information refer to the

Demo Link : https://drive.google.com/file/d/1whsK69OTeUzy-6DI_cECQv3FteXRJLoc/view?usp=sharing

Code : Tshaan1104/SOCIAL_MEDIA_APP (github.com)

Thank You!!!