Social Media App (MERN)

Introduction:

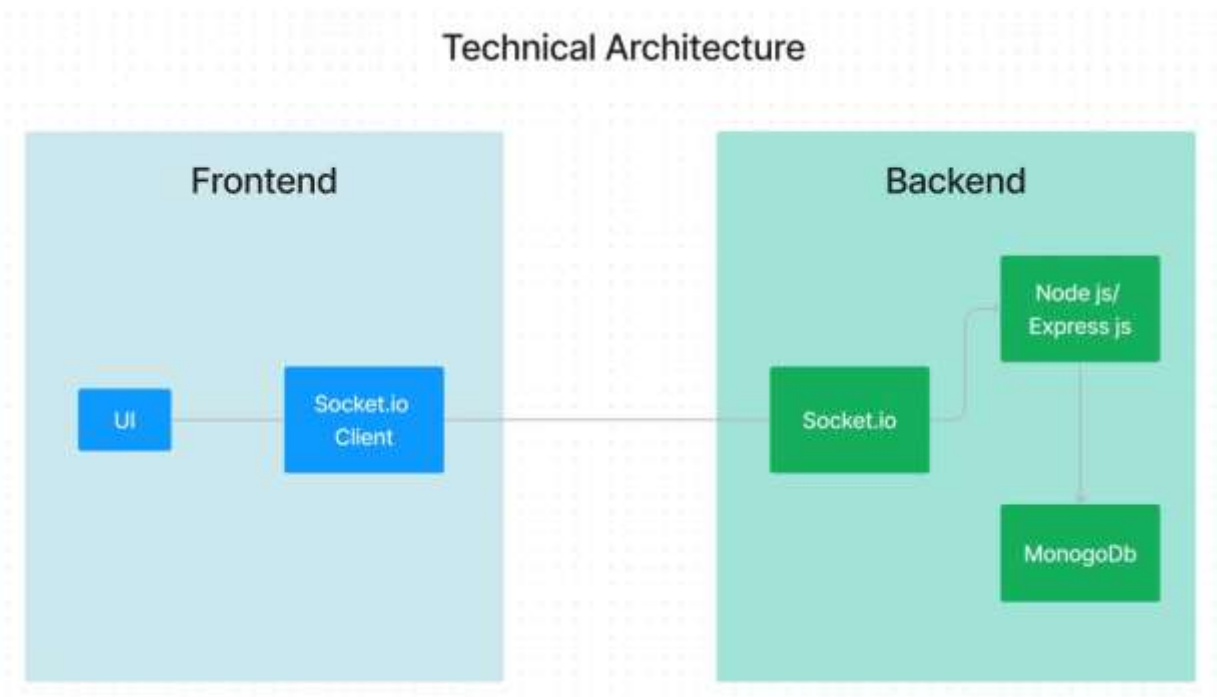
Introducing our revolutionary social media app! Designed to redefine online connections, our platform offers an intuitive and innovative space for seamless communication and engagement.

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.

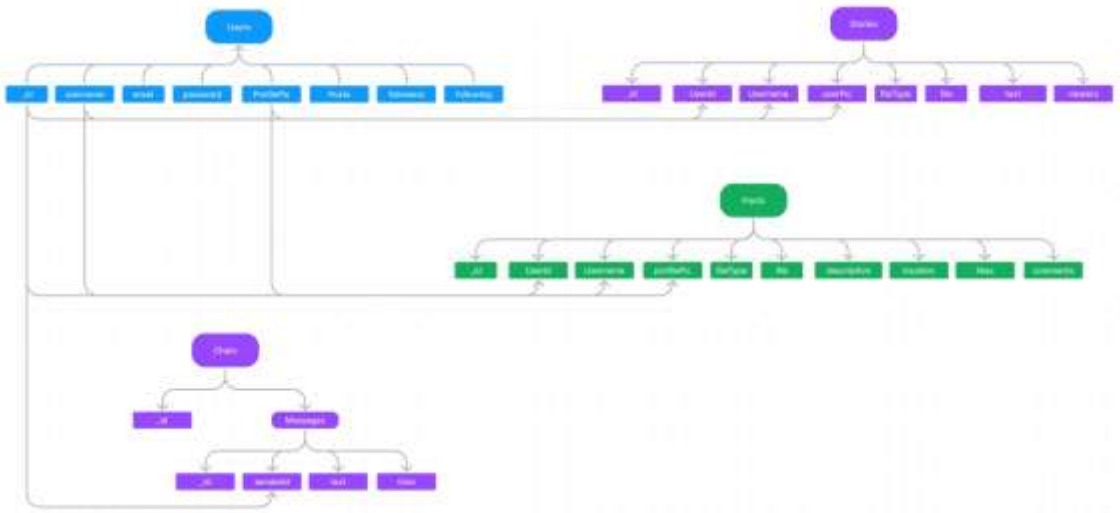
Real-time messaging is facilitated through the socket.io library, enabling instant exchange of messages between users. Users can engage in chats with there friends, sharing text, emojis, images, etc., in real-time.

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://gitscm.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:

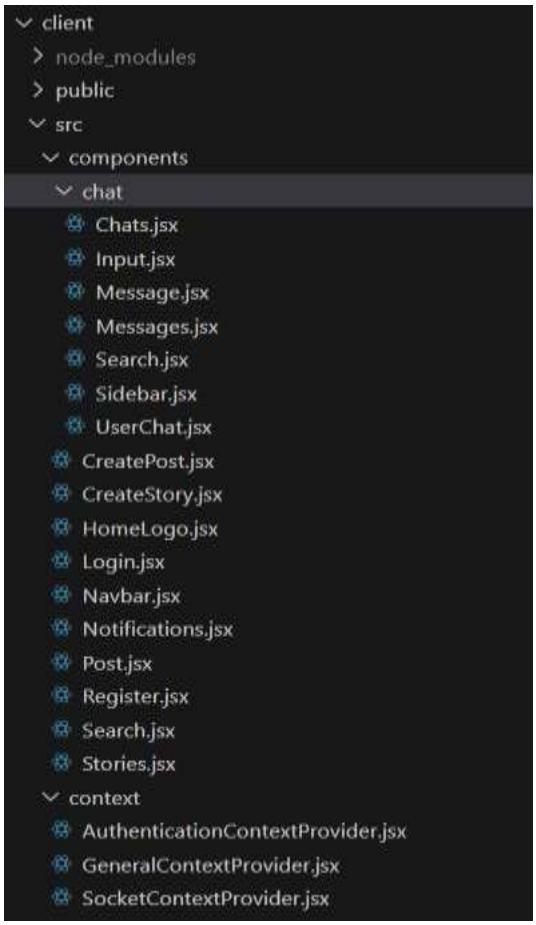
- Create and manage a personal profile.
- Share posts, photos, videos, and stories with their network.
- Engage in conversations through comments, likes, and shares.
- Follow other users and discover new accounts, topics, and trends.
- Explore and discover new content, communities, and opportunities.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

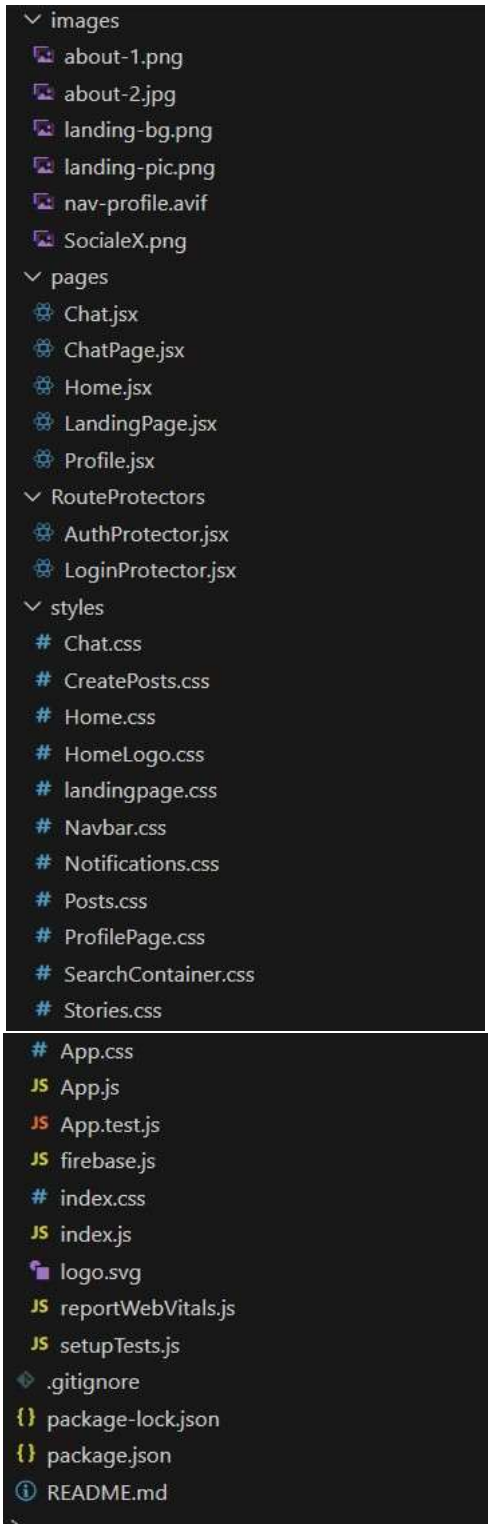
Project structure:

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

Client directory:

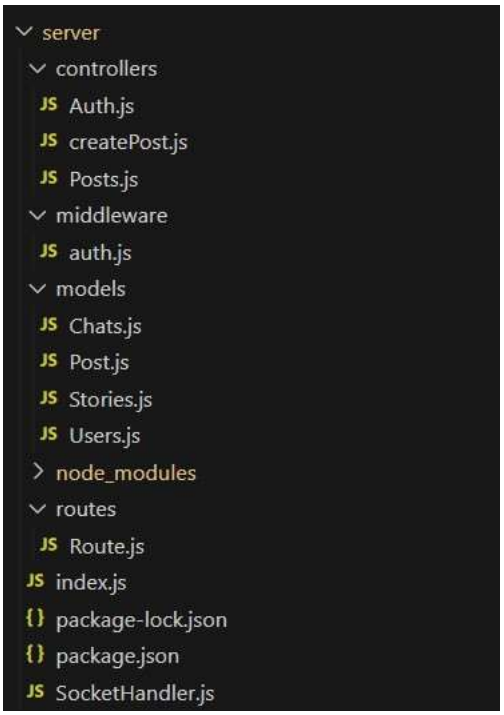
The below directory structure represents the directories and files in the client folder (front end) where, react Js is used along with Api's such as socket.io.


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	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 install necessary tools. For backend (server), we use:

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.

```
JS index.js M ×
server > JS index.js > ...
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 { fileURLToPath } from 'url';

import authRoutes from './routes/Route.js';
import SocketHandler from './SocketHandler.js';
```

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 ×

server > JS index.js > ...

```
// config
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('', authRoutes);

const server = http.createServer(app);

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

io.on("connection", (socket) =>{
    console.log("User connected");

    SocketHandler(socket);
})


// mongoose setup

const PORT = 6001;

mongoose.connect('mongodb://localhost:27017/socialeX', {
        useNewUrlParser: true,
        useUnifiedTopology: true,
    }
).then(()=>{

        server.listen(PORT, ()=>{
            console.log(`Running @ ${PORT}`);
        });
    }
).catch((e)=> console.log(`Error in db connection ${e}`));

```

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

Users.js

server > models > Users.js > userSchema > profilePic

```javascript
import mongoose from 'mongoose';

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

const User = mongoose.model("users", userSchema);
export default User;
```

Chats model

```
Chats.js
server > models > Chats.js > ...
import mongoose from "mongoose";

const chatSchema = mongoose.Schema({
    // _id = u1 _id + u2 _id  (u1 < u2 - compare both and arrange in order)
    _id: {
        type: String,
        require: true
    },
    messages: {
        type: Array
    }
});

const Chats = mongoose.model("chats", chatSchema);
export default Chats;
```

Stories model

```
Stories.js
server > models > Stories.js > storySchema
import mongoose from 'mongoose';

const storySchema = new mongoose.Schema({
    userId:{
        type: String
    },
    username: {
        type: String
    },
    userPic:{
        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

```
JS Post.js ×
server > models > JS Post.js > postSchema
import mongoose from "mongoose";

const postSchema  = mongoose.Schema({
    userId: {
        type: String
    },
    userName:{
        type: String
    },
    userPic:{
        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:

```
JS Auth.js M ×
server > controllers > JS Auth.js > [0] login
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import Users from '../models/Users.js';



const generateToken =(id) =>{

    const jwtSecret = 'thisIsTheSceretCodeForTheJWTToken';

    return jwt.sign({id}, jwtSecret, {
        expiresIn: '30d',
    })
}

export const register = async (req, res) =>{
    try{

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

        const salt = await bcrypt.genSalt();
        const passwordHash = await bcrypt.hash(password, salt);

        const newUser = new Users({
            username,
            email,
            password: passwordHash,
            profilePic
        });

        const user = await newUser.save();

        // generate jwt token using function we defined at top of the page
        const token = generateToken(user._id);

        const userData = {_id: user._id, username: user.username,
                          email:user.email, profilePic:user.profilePic,
                          about: user.about, posts: user.posts,
                          followers: user.followers, following:user.following };

        res.status(200).json({token, user:userData});

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

Login:

```
JS Auth.js M ×
server > controllers > JS Auth.js > [0] login

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

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

        // generate jwt token using function we defined at top of the page
        const token = generateToken(user._id);
        delete user.password;
        const userData = {_id: user._id, username: user.username, email:user.email,
                          profilePic:user.profilePic, 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.

Register UI:

```jsx
Register.jsx
client > src > components > Register.jsx > default
import React, { useContext } from 'react'
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 class="authForm">
        <h2>Register</h2>
        <div class="form-floating mb-3 authFormInputs">
            <input type="text" class="form-control" id="floatingInput" placeholder="username" onChange={(e)=> setUsername(e.target.value)} />
            <label htmlFor="floatingInput">Username</label>
        </div>
        <div class="form-floating mb-3 authFormInputs">
            <input type="email" class="form-control" id="floatingEmail" placeholder="name@example.com" onChange={(e)=> setEmail(e.target.value)} />
            <label htmlFor="floatingInput">Email address</label>
        </div>
        <div class="form-floating mb-3 authFormInputs">
            <input type="password" class="form-control" id="floatingPassword" placeholder="Password" onChange={(e)=> setPassword(e.target.value)} />
            <label htmlFor="floatingPassword">Password</label>
        </div>
        <button class="btn btn-primary" onClick={handleRegister}>Sign up</button>

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

export default Register
```

Login UI:

```jsx
Login.jsx
client > src > components > Login.jsx > Login
import React, { useContext } from 'react'
import { AuthenticationContext } from '../context/AuthenticationContextProvider';

const Login = ({setIsLoginBox}) => {

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

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

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

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

export default Login
```

Context Api for Authentication:

AuthenticationContextProvider.jsx M

client > src > context > AuthenticationContextProvider.jsx > AuthenticationContextProvider

```jsx
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 profilePic = 'https://images.unsplash.com/photo-1593085512500-5d55148d6f0d?ixlib=rb-4.0.

  const profilePic ='';

  const inputs = {username: username, email: email, password: password, profilePic: profilePic};


  const navigate = useNavigate();
```

AuthenticationContextProvider.jsx M

client > src > context > AuthenticationContextProvider.jsx > AuthenticationContextProvider

```jsx

  const navigate = useNavigate();

  const login = async () =>{

    try{

      const loginInputs = {email: email, password: password}
        await axios.post('http://localhost:6001/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('profilePic', res.data.user.profilePic);
            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);
        });

    }catch(err){
        console.log(err);
    }
  }

  const register = async () =>{

    try{
        await axios.post('http://localhost:6001/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('profilePic', res.data.user.profilePic);
          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);
        });

    }catch(err){
        console.log(err);
    }
  }

```

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

  const logout = async () =>{

    for (let key in localStorage) {
      if (localStorage.hasOwnProperty(key)) {
        localStorage.removeItem(key);
      }
    }

    navigate('/landing');
  }



  return (
    <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.

General context:

```
GeneralContextProvider.jsx
client > src > context > GeneralContextProvider.jsx > GeneralContextProvider
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 [chatFriends, 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,
                  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, chatFriends, setChatFriends, chatData:state, dispatch}}>

              {children}
    </GeneralContext.Provider>
  )
}
```

5. **Create Posts:**

Frontend:

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

```
CreatePost.js ×
client > src > components > CreatePost.js > CreatePost > handlePostUpload > uploadTask.on('state_changed').catch > then() callback
import React, { useContext, useState } from 'react'
import '../Styles/CreatePosts.css'
import {ReCross2} from 'react-icons/re'
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 [isCreatePostOpen, setIsCreatePostOpen] = useContext(GeneralContext);
    const [postType, setPostType] = useState('Posts');
    const [postDescription, setPostDescription] = useState('');
    const [postLocation, setPostLocation] = useState('');
    const [postFile, setPostFile] = useState(null);
    const [uploadProgress, setUploadProgress] = useState();

    if (uploadProgress === 100){
        setPostDescription('');
        setPostLocation('');
        setPostFile(null);
        setIsCreatePostOpen(false);
        setUploadProgress();
    }
    const HandlePostUpload = async (e) =>{
        e.preventDefault();
        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('profilePic'), fileType: postType, file: downloadURL,
                                description: postDescription, location: postLocation, comments:{"New user": "This is my first comment"}}

                axios.post('http://localhost:5001/createPost', inputs)
                .then( async (res)=>{
                }).catch(err=> {
                    console.log(err);
                });

            }catch(err){
                console.log(err);
            }});
        });
    }
```

```
CreatePost.js ×
client > src > components > CreatePost.js > CreatePost > handlePostUpload > uploadTask.on('state_changed').catch > then() callback

    return (
        <>
            <div className="createPostModalBg" style={isCreatePostOpen? {display: 'contents'} : {display: 'none'}} />
                <div class="createPostContainer">

                    <ReCross2 className="closeCreatePost" onClick={()=> setIsCreatePostOpen(false)} />
                    <p> className="createPostTitle">Create post</p>
                    <br> className="createPostBr" />

                    <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 autoforalinputs 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 autoforalinputs 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.

```
createPost.js ×
server > controllers > createPost.js > createPost
import Post from '../models/Post.js';

export const createPost = async (req, res) =>{
    try{

        const newPost = new Post(req.body);

        const post = await newPost.save();

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

6. Display Posts:

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

```
Post.jsx M ×
client > src > components > Post.jsx > Post
import React, { useContext, useEffect, useState } from 'react';
import '../styles/Posts.css';
import { AiOutlineHeart, AiTwotoneHeart } from "react-icons/ai";
import { FaGlobeAmericas } from "react-icons/fa";
import {IoIosPersonAdd} from 'react-icons/io'
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/FetchAllPosts');
          const fetchedPosts = response.data;
          setPosts(fetchedPosts);
        } catch (error) {
          console.error(error);
        }
      };

    //  like

    const handleLike = (userId, postId) =>{
        socket.emit('postLiked', {userId, postId});
    }
    const handleUnLike = (userId, postId) =>{
        socket.emit('postUnLiked', {userId, postId});
    }
    useEffect(()=>{
        socket.on("likeUpdated", ()=>{
            // alert("liked");
        })
        socket.on('userfollowed', ({following})=>{
            localStorage.setItem('following', following);
        })
    },[socket])

    const handleFollow = async (userId) =>{
        socket.emit('FollowUser', {ownId: localStorage.getItem('userId'), FollowingUserId: userId});
    }

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

Post.jsx M ×

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

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

  {posts ? posts.map((post) => {

      return(

      <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" name='place'>{post.location}</label>
          </div>
      </div>
```

Post.jsx M ×

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

```
          <div className="detail">
              <div className='descdatawithBtn'>
                  <label htmlFor='username' className="desc labelbtn" 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 class="comments">
                      {post.comments.map((comment)=>{
                          return(

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

      }) : <></>}

      </div>
  )
}

export default Post
```

7. Socket Handling in backend:

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

```
SocketHandler.js
server > SocketHandler.js > SocketHandler > socket.on('story-played') callback

import Chats from "./models/Chats.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});
        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();
        }

    });
```

```
SocketHandler.js
server > SocketHandler.js > SocketHandler > socket.on('story-played') callback

    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));
        }

        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});
    })
```

SocketHandler.js

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

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 ×
server > controllers > JS Posts.js > fetchAllStories
import Post from '../models/Post.js';
import Stories from '../models/Stories.js';
import User from '../models/Users.js'

export const fetchAllPosts = async (req, res) =>{
    try {
        const posts = await Post.find().sort({ _id: -1 });

        res.json(posts);
      } catch (error) {
        console.error(error);
        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.error(error);
      res.status(500).json({ error: 'Server error' });
    }
}

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.error(error);
      res.status(500).json({ error: 'Server error' });
    }
}
```

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 ×

client > src > components > Stories.jsx > Stories

```react_stories_1.jsx
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, 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/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);

      }
```

```react_stories_2.jsx
    return (
      <div className="storiesContainer">

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

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

          <div className="stories">

            <div className="story self-story" onClick={addStory}>
              <image src={localStorage.getItem('profilePic')} alt="" />
              <p>Add story</p>
              <BiPlusCircle />
            </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: '1px solid #e4e4e4'}
                                : {border: '1px solid #1d9bf0'}
                                }
                  <image src={story.userPic} alt="" />
                  <p>{story.username}</p>
                </div>
                ))
            }

          </div>

        </div>
      )
```

```
Stories.jsx M ×
client > src > components > Stories.jsx > Stories
            {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='posting' controls autoPlay muted>
                                <source src={story.file} />
                            </video>
                        }

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



        </div>
    )
}

export default Stories
```

Create new story:

```
CreateStory.jsx M ×
client > src > components > CreateStory.jsx > CreateStory
import React, { useContext, useState } 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();

    if (uploadProgress === 100){
        setStoryDescription('');
        setStoryFile(null);
        setIsCreateStoryOpen(false);
        setUploadProgress();
    }

```

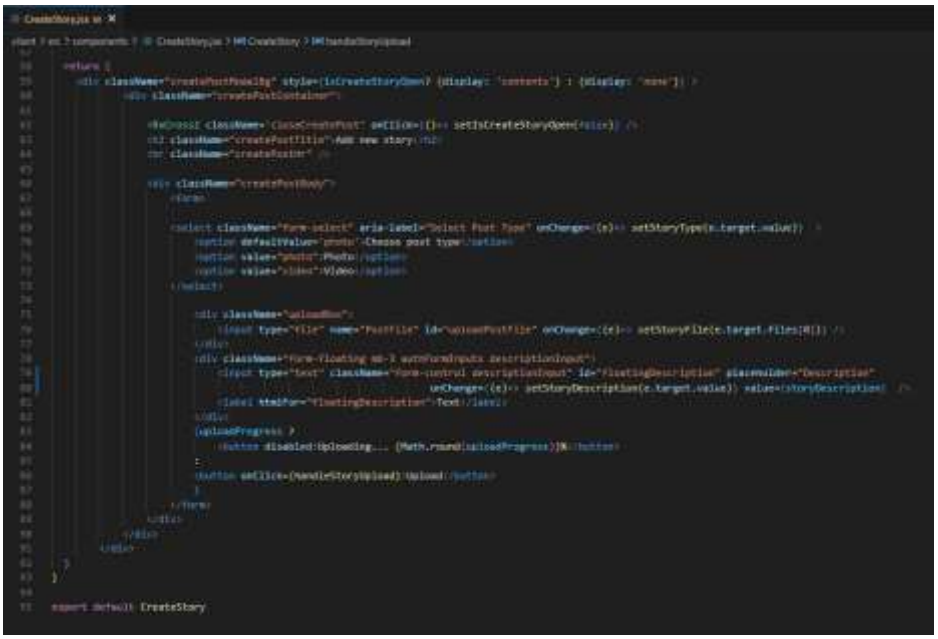
```
CreateStory.jsx M ×
client > src > components > CreateStory.jsx > CreateStory > handleStoryUpload
    const handleStoryUpload = async (e) =>{
        e.preventDefault();
        const storageRef = ref(storage, uuidv4());
        const uploadTask = uploadBytesResumable(storageRef, storyFile);

        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);

            try{
                await socket.emit('create-new-story', {userId: localStorage.getItem('userId'), username: localStorage.getItem('username'),
                                                       userPic: localStorage.getItem('profilePic'), fileType: storyType, file: downloadURL,
                                                       text: storyDescription});

                setIsCreateStoryOpen(false);
                setStoryDescription('');
                setStoryFile(null);
                setIsCreateStoryOpen(false);
                setUploadProgress();

            }catch(err){
                console.log(err);
            } });
        } );
    }
```



Backend:

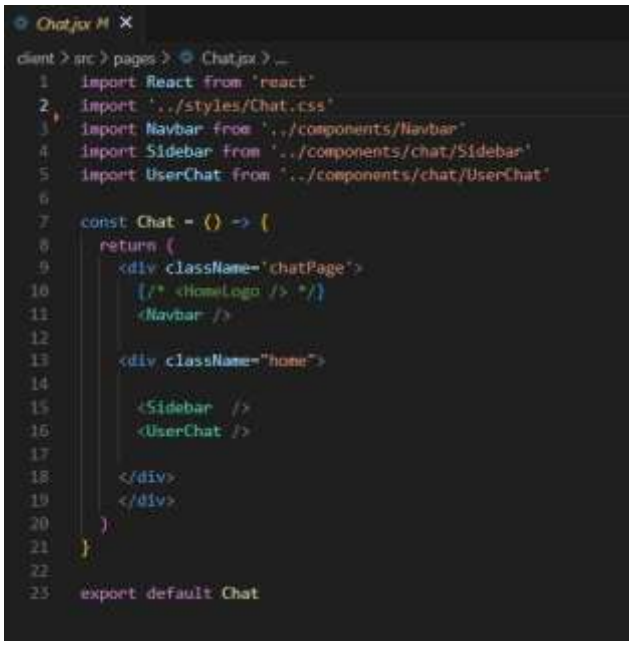
The backend for stories is already covered in socket handling file and posts file in server.

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.



Sidebar:

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

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

const Sidebar = () => {
  return (
    <div class="sidebar" >

      {/* <Navbar /> */}

      <Search />
      <Chats />

    </div>
  )
}

export default Sidebar
```

Search:

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

```jsx
import React, { useContext, useEffect, useState } from 'react'
import { ThSearch } from 'react-icons/th'
import { GeneralContext } from '../../context/GeneralContextProvider';

const Search = () => {

  const [dispatch, socket] = useContext(GeneralContext)
  const [search, setSearch] = useState('');
  const userId = localStorage.getItem('userId');
  const [user, setUser] = useState();
  const [err, setErr] = useState(false);

  const handleSearch = async (e) =>{
    e.preventDefault();
    setErr(false);
    setUser();
    await socket.emit('chat-user-searched', {ownId: userId,username: search});
    setSearch('')
  }

  useEffect(()=>{
    socket.on('searched-chat-user', async ({user})=>{
      setUser(user);
    });
    socket.on('no-searched-chat-user', async ()=>{
      setErr(true);
    });
  },[socket])

  const handleSelect = async (user) =>{
    await dispatch({type:"CHANGE_USER", payload: user});
    setUser();
  }

  return (
    <div class="search">
      <div class="searchform">
        <input type="text" placeholder="Search"
                onChange={(e)=> {setSearch(e.target.value)}} value={search} />
        <div class="s-icon" onClick={handleSearch}>
          <ThSearch />
        </div>
      </div>

      {err && <span>No User Found!!</span>}

      {user &&  <div class="userInfo" onClick={() => handleSelect(user)} >
                  <img src={user.profilePic} alt="" />
                  <div class="userChatInfo">  <span>{user.username}</span> </div>
                </div>
              }
    </div>
)}
export default Search
```

Chats:

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

```
Chats.jsx 1, M
client > src > components > chat > Chats.jsx > Chats > chatFriends.map() callback
import React, { useContext, useEffect, useState } from 'react'
import {GeneralContext} from '../../context/GeneralContextProvider';
const Chats = () => {

  const {socket, chatfriends, setChatFriends, dispatch, chatData} = useContext(GeneralContext)
  const userId = localStorage.getItem('userId');

  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 class="chats">

    {chatFriends.map((data)=>{
      return(
        <div class="userInfo" key={data._id} onClick={()=> handleSelect(data)} >
          <img src={data.profilePic} alt="" />
          <div class="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.

```
UserChat.jsx M
client > src > components > chat > UserChat.jsx > UserChat
import React, { useContext } from 'react'
import Input from './Input';
import Messages from './Messages';
import { GeneralContext } from '../../context/GeneralContextProvider';

const UserChat = () => {

  const {chatData} = useContext(GeneralContext);

  return (
    <div class="chat">
      { chatData.user &&

      <div class="chatInfo">
        <img src={chatData.user?.profilePic} 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.

```
Input.jsx M ×
client > src > components > chat > Input.jsx > Input > handleSend
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';

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});
              setUploadProgress();
              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('');
      }
    }

```

```
Input.jsx M ×
client > src > components > chat > Input.jsx > Input > handleSend
    return (
      <div class="input" >
        <input type="text" placeholder="type something..." onChange={e => setText(e.target.value)} value={text} />
        <div class="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.

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

Message:

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

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

11. Navbar:

Now let's look into the navbar component.

```
Navbar.jsx 1, M ×
client > src > components > Navbar.jsx > Navbar
import { GeneralContext } from '../context/GeneralContextProvider';
import { useNavigate } from 'react-router-dom';

const Navbar = () => {

  const [isCreatPostOpen, setIsCreatePostOpen, setIsCreateStoryOpen, isNotificationsOpen, setNotificationsOpen] = useContext(GeneralContext);

  const navigate = useNavigate();
  const profilePic = localStorage.getItem('profilePic');
  const userId = localStorage.getItem('userId');

  return (
    <div class="Navbar">
      <BiHomeSmile class="homebtn btns" onClick={()=> navigate('/')} />
      <BsChatSquareText  class="chatbtn btns" onClick={()=> navigate('/chat')} />
      <CgAddR class="createPostbtn btns" onClick={()=> (setIsCreatePostOpen(!isCreatPostOpen), setIsCreateStoryOpen(false))} />
      <TbNotification class="Notibtn btns" onClick={()=> setNotificationsOpen(!isNotificationsOpen)}/>
      <img class="profile" src={profilePic} alt="" onClick={()=> navigate(`/profile/${userId}`)} />
    </div>
  )
}
export default Navbar
```

12. Logo & User Search:

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

```
HomeLogo.jsx M ×
client > src > components > HomeLogo.jsx > HomeLogo
import React, { useContext, useEffect, useState } from 'react';
import logoImg from '../Images/SocialeX.png';
import '../styles/HomeLogo.css';
import { TbSearch } from 'react-icons/tb';
import { GeneralContext } from '../context/GeneralContextProvider';
import Search from './Search';

const HomeLogo = () => {

  const [socket] = useContext(GeneralContext);
  const [search, setSearch] = useState('');
  const [searchedUser, setSearchedUser] = useState[];

  const handleSearch = async ()=>{
    await socket.emit('user-search', {username: search});
    setSearch('')
  }
  useEffect(()=>{
    socket.on('searched-user', ({user})=>{
      setSearchedUser(user);
    });
  },[socket])

  return (
    <div class="logoSearch">
      <img class="logoImg" src={logoImg} alt="" />
      <div class="Search">
        <input type="text" placeholder='Search' onChange={(e)=> (setSearch(e.target.value))} value={search} />
        <div class="s-icon" onClick={handleSearch}>
          <TbSearch />
        </div>
      </div>
      <Search searchedUser={searchedUser} setSearchedUser={setSearchedUser} />
    </div>
  )
}

export default HomeLogo
```

Search Component:

```
Search.jsx M ×
client > src > components > Search.jsx > Search
import React from 'react'
import '../styles/SearchContainer.css';
import { useNavigate } from 'react-router-dom';

const Search = ({searchedUser, setSearchedUser}) => {
    const navigate = useNavigate();
  return (
    <div class="searchContainer">

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

export default Search
```

13. Routes in Backend:

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

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

Demo link:

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

Github Link : https://github.com/parvinshaik/Socialex.git