SYMBIOSIS INTERNATIONAL (DEEMED UNIVERSITY)
(Established under Section 3 of the UGC Act, 1956)
Re-accredited by NAAC with 'A++' Grade | Awarded Category – I by UGC
Founder: Prof. Dr. S. B. Mujumdar, M.Sc., Ph.D. (Awarded Padma Bhushan and Padma Shri by President of India)

Name- Abhinav Kumar

PRN- 21070126006

Branch- AIML_A1

Batch- 2021-2025

Subject- Full Stack Development

Topic – Chat App Using MERN Stack

ESE Project Report

Introduction

Introducing my full-stack project utilizing the renowned MERN stack—a robust combination of MongoDB, Express.js, Node.js, and React.js. The MERN stack has gained widespread recognition for its simplicity and versatility, making it a preferred choice for developing dynamic web applications. As the JavaScript ecosystem continues to flourish, the MERN stack stands out as a go-to solution for both seasoned developers and newcomers alike.

This full-stack chat application boasts seamless functionality, easily deployable with minimal setup requirements. Leveraging Material UI for the frontend, powered by React, and fortified by Express.js and Node.js for the backend, it delivers real-time message broadcasting through the innovative use of Socket.IO.

Objectives

1. **Secure Authentication:** Implement JWT token-based authentication to ensure secure access to the application, allowing users to authenticate securely and access their accounts with confidence.
2. **Global Chat Integration:** Enable a global chat feature allowing users to broadcast messages to all other users of the application, fostering community engagement and communication among users.
3. **Private Chat Implementation**: Develop a private chat functionality enabling users to have one-on-one conversations securely, enhancing user interaction and facilitating personalized communication.
4. **Real-time Updates:** Implement real-time updates for user lists, conversation lists, and conversation messages, utilizing technologies like WebSockets or Socket.IO to ensure instantaneous updates and seamless user experience.
5. **User-friendly Interface:** Design an intuitive and visually appealing user interface to enhance user experience, ensuring ease of navigation and accessibility for users across different devices and platforms.

H/W and S/W requirements

Hardware Requirements:

Server Infrastructure: You'll need a server to host your application. This could be a physical server or a cloud-based solution like AWS, Azure, or DigitalOcean.

Memory and Storage: Ensure sufficient RAM and storage space on your server to accommodate the application's data and handle concurrent user requests efficiently.

Network Infrastructure: Reliable internet connectivity and sufficient bandwidth are essential for seamless communication between clients and the server.

Client Devices: Users will access the application from various devices such as desktops, laptops, tablets, and smartphones. Ensure compatibility with common devices and screen sizes.

Software Requirements:

Operating System: Your server should run a compatible operating system. Common choices include Linux distributions like Ubuntu, CentOS, or Debian.

Web Server: Install and configure a web server such as Nginx or Apache to serve your application to users and handle HTTP requests.

Database: Set up MongoDB to store application data. Ensure compatibility with the version of MongoDB supported by your application.

Node.js and npm: Install Node.js and npm (Node Package Manager) to run server-side JavaScript code and manage application dependencies.

Express.js: Install Express.js, a web application framework for Node.js, to handle routing, middleware, and other server-side functionality.

React.js: Set up React.js for the frontend to build interactive user interfaces and manage client-side application logic.

Socket.IO: Install Socket.IO to enable real-time communication between clients and the server, facilitating features like real-time chat updates.

Material UI: If you're using Material UI for the frontend, install the necessary dependencies to incorporate Material Design components and styles into your React application.

JWT (JSON Web Tokens): Implement JWT for authentication. Install relevant packages to generate, verify, and manage JWT tokens securely.

Code Snippet

client:

1) **Build**

   ➢ **Manifest.json**

```
➢ {
➢   "short_name": "React App",
➢   "name": "Create React App Sample",
➢   "icons": [
➢     {
➢       "src": "favicon.ico",
➢       "sizes": "64x64 32x32 24x24 16x16",
➢       "type": "image/x-icon"
➢     }
➢   ],
➢   "start_url": ".",
➢   "display": "standalone",
➢   "theme_color": "#000000",
➢   "background_color": "#ffffff"
➢ }
➢
```

2) **Public**

   ➢ **Index.html**

```
➢ <!DOCTYPE html>
➢ <html lang="en">
➢     <head>
➢         <meta charset="utf-8" />
➢         <link rel="shortcut icon"
  href="%PUBLIC_URL%/favicon.png" />
➢         <meta name="viewport" content="width=device-width,
  initial-scale=1" />
➢         <meta name="theme-color" content="#000000" />
➢
➢         <link rel="manifest" href="%PUBLIC_URL%/manifest.json"
  />
➢
➢         <title>Chat App</title>
➢     </head>
➢     <body>
➢         <noscript>You need to enable JavaScript to run this
  app.</noscript>
➢         <div id="root"></div>
➢      </body>
➢ </html>
➢
```

3) Src

➢ Chat.jsx

```
import React, { useState, useEffect } from 'react';
import { makeStyles } from '@material-ui/core/styles';
import Grid from '@material-ui/core/Grid';
import Paper from '@material-ui/core/Paper';
import List from '@material-ui/core/List';
import Tabs from '@material-ui/core/Tabs';
import Tab from '@material-ui/core/Tab';

import Header from '../Layout/Header';
import ChatBox from './ChatBox';
import Conversations from './Conversations';
import Users from './Users';

const useStyles = makeStyles(theme => ({
    paper: {
        minHeight: 'calc(100vh - 64px)',
        borderRadius: 0,
        background: 'linear-gradient(45deg, #ADD8E6 30%, #87CEEB 90%)', // Gradient light blue background
    },
    sidebar: {
        zIndex: 8,
    },
    subheader: {
        display: 'flex',
        alignItems: 'center',
        cursor: 'pointer',
    },
    globe: {
        backgroundColor: theme.palette.primary.dark,
    },
    subheaderText: {
        color: theme.palette.primary.dark,
    },
}));

const Chat = () => {
    const [scope, setScope] = useState('Global Chat');
    const [tab, setTab] = useState(0);
    const [user, setUser] = useState(null);
    const classes = useStyles();

    const handleChange = (e, newVal) => {
        setTab(newVal);
    };

    return (
        <React.Fragment>
            <Header />
            <Grid container>
                <Grid item md={4} className={classes.sidebar}>
                    <Paper className={classes.paper} square elevation={5}>
                        <Paper square>
                            <Tabs
                                onChange={handleChange}
                                variant="fullWidth"
                                value={tab}
                                indicatorColor="primary"
                                textColor="primary"
                            >
                                <Tab label="Chats" />
                                <Tab label="Users" />
                            </Tabs>
                        </Paper>
                        {tab === 0 && (
                            <Conversations
                                setUser={setUser}
                                setScope={setScope}
                            />
                        )}
                        {tab === 1 && (
                            <Users setUser={setUser} setScope={setScope} />
                        )}
                    </Paper>
                </Grid>
                <Grid item md={8}>
                    <ChatBox scope={scope} user={user} />
                </Grid>
            </Grid>
        </React.Fragment>
    );
};

export default Chat;

```

➢ **ChatBox.jsx**

```
import React, { useState, useEffect, useRef } from "react";
import { makeStyles } from "@material-ui/core/styles";
import Grid from "@material-ui/core/Grid";
import Typography from "@material-ui/core/Typography";
import TextField from "@material-ui/core/TextField";
import IconButton from "@material-ui/core/IconButton";
import SendIcon from "@material-ui/icons/Send";
import List from "@material-ui/core/List";
import ListItem from "@material-ui/core/ListItem";
import ListItemText from "@material-ui/core/ListItemText";
import ListItemAvatar from "@material-ui/core/ListItemAvatar";
import Avatar from "@material-ui/core/Avatar";
import Paper from "@material-ui/core/Paper";
import socketIOClient from "socket.io-client";
import classnames from "classnames";
import commonUtilites from "../Utilities/common";
import {
  useGetGlobalMessages,
  useSendGlobalMessage,
  useGetConversationMessages,
  useSendConversationMessage,
} from "../Services/chatService";
import { authenticationService } from
"../Services/authenticationService";

const useStyles = makeStyles((theme) => ({
  root: {
    height: "100%",
  },
  headerRow: {
    maxHeight: 60,
    zIndex: 5,
  },
  paper: {
    display: "flex",
    alignItems: "center",
    justifyContent: "center",
    height: "100%",
    color: theme.palette.primary.dark,
  },
  messageContainer: {
    height: "100%",
    display: "flex",
    alignContent: "flex-end",
  },
  messagesRow: {
    maxHeight: "calc(100vh - 184px)",
    overflowY: "auto",
  },
  newMessageRow: {
    width: "100%",
    padding: theme.spacing(0, 2, 1),
  },
  messageBubble: {
    padding: 10,
    border: "1px solid white",
    backgroundColor: "white",
    borderRadius: "0 10px 10px 10px",
    boxShadow: "-3px 4px 4px 0px rgba(0,0,0,0.08)",
    marginTop: 8,
    maxWidth: "40em",
  },
  messageBubbleRight: {
    borderRadius: "10px 0 10px 10px",
  },
  inputRow: {
    display: "flex",
    alignItems: "flex-end",
  },
  form: {
    width: "100%",
  },
  avatar: {
    margin: theme.spacing(1, 1.5),
  },
  listItem: {
    display: "flex",
    width: "100%",
  },
  listItemRight: {
    flexDirection: "row-reverse",
  },
}));

const ChatBox = (props) => {
  const [currentUserId] = useState(
    authenticationService.currentUserValue.userId
  );
  const [newMessage, setNewMessage] = useState("");
  const [messages, setMessages] = useState([]);
  const [lastMessage, setLastMessage] = useState(null);

  const getGlobalMessages = useGetGlobalMessages();
  const sendGlobalMessage = useSendGlobalMessage();
  const getConversationMessages = useGetConversationMessages();
  const sendConversationMessage = useSendConversationMessage();

  let chatBottom = useRef(null);
```

```
➢   const classes = useStyles();
➢ 
➢   useEffect(() => {
➢     reloadMessages();
➢     scrollToBottom();
➢   }, [lastMessage, props.scope, props.conversationId]);
➢ 
➢   useEffect(() => {
➢     const socket = socketIOClient(process.env.REACT_APP_API_URL);
➢     socket.on("messages", (data) => setLastMessage(data));
➢   }, []);
➢ 
➢   const reloadMessages = () => {
➢     if (props.scope === "Global Chat") {
➢       getGlobalMessages().then((res) => {
➢         setMessages(res);
➢       });
➢     } else if (props.scope !== null && props.conversationId !==
  null) {
➢       getConversationMessages(props.user._id).then((res) =>
  setMessages(res));
➢     } else {
➢       setMessages([]);
➢     }
➢   };
➢ 
➢   const scrollToBottom = () => {
➢     chatBottom.current.scrollIntoView({ behavior: "smooth" });
➢   };
➢ 
➢   useEffect(scrollToBottom, [messages]);
➢ 
➢   const handleSubmit = (e) => {
➢     e.preventDefault();
➢     if (props.scope === "Global Chat") {
➢       sendGlobalMessage(newMessage).then(() => {
➢         setNewMessage("");
➢       });
➢     } else {
➢       sendConversationMessage(props.user._id,
  newMessage).then((res) => {
➢         setNewMessage("");
➢       });
➢     }
➢   };
➢ 
➢   return (
➢     <Grid container className={classes.root}>
```

```
      <Grid item xs={12} className={classes.headerRow}>
        <Paper className={classes.paper} square elevation={2}>
          <Typography color="inherit" variant="h6">
            {props.scope}
          </Typography>
        </Paper>
      </Grid>
      <Grid item xs={12}>
        <Grid container className={classes.messageContainer}>
          <Grid item xs={12} className={classes.messagesRow}>
            {messages && (
              <List>
                {messages.map((m) => (
                  <ListItem
                    key={m._id}
                    className={classnames(classes.listItem, {
                      [`${classes.listItemRight}`]:
                        m.fromObj[0]._id === currentUserId,
                    })}
                    alignItems="flex-start"
                  >
                    <ListItemAvatar className={classes.avatar}>
                      <Avatar>
                        {commonUtilites.getInitialsFromName(m.fro
mObj[0].name)}
                      </Avatar>
                    </ListItemAvatar>
                    <ListItemText
                      classes={{
                        root: classnames(classes.messageBubble, {
                          [`${classes.messageBubbleRight}`]:
                            m.fromObj[0]._id === currentUserId,
                        }),
                      }}
                      primary={m.fromObj[0] && m.fromObj[0].name}
                      secondary={<React.Fragment>{m.body}</React.
Fragment>}
                    />
                  </ListItem>
                ))}
              </List>
            )}
            <div ref={chatBottom} />
          </Grid>
          <Grid item xs={12} className={classes.inputRow}>
            <form onSubmit={handleSubmit}
className={classes.form}>
              <Grid
```

```
                container
                className={classes.newMessageRow}
                alignItems="flex-end"
              >
                <Grid item xs={11}>
                  <TextField
                    id="message"
                    label="Message"
                    variant="outlined"
                    margin="dense"
                    fullWidth
                    value={newMessage}
                    onChange={(e) =>
setNewMessage(e.target.value)}
                  />
                </Grid>
                <Grid item xs={1}>
                  <IconButton type="submit">
                    <SendIcon />
                  </IconButton>
                </Grid>
              </Grid>
            </form>
          </Grid>
        </Grid>
      </Grid>
    </Grid>
  );
};

export default ChatBox;

```

➢ **Conversations.jsx**

```
import React, { useState, useEffect } from "react";
import List from "@material-ui/core/List";
import ListItem from "@material-ui/core/ListItem";
import ListItemText from "@material-ui/core/ListItemText";
import ListItemAvatar from "@material-ui/core/ListItemAvatar";
import Avatar from "@material-ui/core/Avatar";
import LanguageIcon from "@material-ui/icons/Language";
import Divider from "@material-ui/core/Divider";
import { makeStyles } from "@material-ui/core/styles";
import socketIOClient from "socket.io-client";

import { useGetConversations } from "../Services/chatService";
import { authenticationService } from "../Services/authenticationService";
import commonUtilites from "../Utilities/common";

const useStyles = makeStyles((theme) => ({
  subheader: {
    display: "flex",
    alignItems: "center",
    cursor: "pointer",
  },
  globe: {
    backgroundColor: theme.palette.primary.dark,
  },
  subheaderText: {
    color: theme.palette.primary.dark,
    fontWeight: "bold",
  },
  list: {
    maxHeight: "calc(100vh - 112px)",
    overflowY: "auto",
    backgroundColor: theme.palette.background.paper,
    borderRadius: theme.shape.borderRadius,
  },
  listItem: {
    cursor: "pointer",
    "&:hover": {
      backgroundColor: theme.palette.action.hover,
    },
  },
}));

const Conversations = (props) => {
  const classes = useStyles();
  const [conversations, setConversations] = useState([]);
  const [newConversation, setNewConversation] = useState(null);
  const getConversations = useGetConversations();

  // Returns the recipient name that does not
  // belong to the current user.
  const handleRecipient = (recipients) => {
    for (let i = 0; i < recipients.length; i++) {
      if (
        recipients[i].username !==
        authenticationService.currentUserValue.username
      ) {
        return recipients[i];
      }
    }
    return null;
  };

  useEffect(() => {
    getConversations().then((res) => setConversations(res));
  }, [newConversation]);

  useEffect(() => {
    let socket = socketIOClient(process.env.REACT_APP_API_URL);
    socket.on("messages", (data) => setNewConversation(data));

    return () => {
      socket.removeListener("messages");
    };
  }, []);

  return (
    <List className={classes.list}>
      <ListItem
        classes={{ root: classes.subheader }}
        onClick={() => {
          props.setScope("Global Chat");
        }}
      >
        <ListItemAvatar>
          <Avatar className={classes.globe}>
            <LanguageIcon />
          </Avatar>
        </ListItemAvatar>
        <ListItemText
          className={classes.subheaderText}
          primary="Global Chat"
        />
      </ListItem>
      <Divider />

      {conversations && (
        <>
          {conversations.map((c) => (
            <ListItem
              className={classes.listItem}
              key={c._id}
              button
              onClick={() => {
                props.setUser(handleRecipient(c.recipientObj));
                props.setScope(handleRecipient(c.recipientObj).na
me);
              }}
            >
              <ListItemAvatar>
                <Avatar>
                  {commonUtilites.getInitialsFromName(
                    handleRecipient(c.recipientObj).name
                  )}
                </Avatar>
              </ListItemAvatar>
              <ListItemText
                primary={handleRecipient(c.recipientObj).name}
                secondary={c.lastMessage}
              />
            </ListItem>
          ))}
        </>
      )}
    </List>
  );
};

export default Conversations;

```

- **Users.jsx**

```
import React, { useState, useEffect } from "react";
import List from "@material-ui/core/List";
import ListItem from "@material-ui/core/ListItem";
import ListItemText from "@material-ui/core/ListItemText";
import ListItemAvatar from "@material-ui/core/ListItemAvatar";
import Avatar from "@material-ui/core/Avatar";
import { makeStyles } from "@material-ui/core/styles";
import socketIOClient from "socket.io-client";

import { useGetUsers } from "../Services/userService";
import commonUtilites from "../Utilities/common";

const useStyles = makeStyles((theme) => ({
  subheader: {
    display: "flex",
    alignItems: "center",
    cursor: "pointer",
  },
  globe: {
    backgroundColor: theme.palette.primary.dark,
  },
  subheaderText: {
    color: theme.palette.primary.dark,
  },
  list: {
    maxHeight: "calc(100vh - 112px)",
    overflowY: "auto",
  },
  avatar: {
    margin: theme.spacing(0, 3, 0, 1),
  },
}));

const Users = (props) => {
  const classes = useStyles();
  const [users, setUsers] = useState([]);
  const [newUser, setNewUser] = useState(null);
  const getUsers = useGetUsers();

  useEffect(() => {
    getUsers().then((res) => setUsers(res));
  }, [newUser]);

  useEffect(() => {
    const socket = socketIOClient(process.env.REACT_APP_API_URL);
    socket.on("users", (data) => {
      setNewUser(data);
    });
  }, []);

  return (
    <List className={classes.list}>
      {users && (
        <React.Fragment>
          {users.map((u) => (
            <ListItem
              className={classes.listItem}
              key={u._id}
              onClick={() => {
                props.setUser(u);
                props.setScope(u.name);
              }}
              button
            >
              <ListItemAvatar className={classes.avatar}>
                <Avatar>{commonUtilites.getInitialsFromName(u.nam
e)}</Avatar>
              </ListItemAvatar>
              <ListItemText primary={u.name} />
            </ListItem>
          ))}
```

```
            </React.Fragment>
          )}
        </List>
    );
};

export default Users;

```

4) Home

➢ **Home.jsx**

```
import React, { useState, useEffect } from 'react';
import Container from '@material-ui/core/Container';

import history from '../Utilities/history';
import Login from './Login';
import Register from './Register';
import { authenticationService } from '../Services/authenticationService';

const Home = () => {
    const [page, setPage] = useState('login');

    useEffect(() => {
        if (authenticationService.currentUserValue) {
            history.push('/chat');
        }
    }, []);

    const handleClick = location => {
        setPage(location);
    };

    let Content;

    if (page === 'login') {
        Content = <Login handleClick={handleClick} />;
    } else {
        Content = <Register handleClick={handleClick} />;
    }

    return (
        <Container component="main" maxWidth="xs">
            {Content}
        </Container>
    );
};
```

```

export default Home;

```

➢ **Login.jsx**

```
import React from 'react';
import Button from '@material-ui/core/Button';
import { Formik } from 'formik';
import Grid from '@material-ui/core/Grid';
import Link from '@material-ui/core/Link';
import TextField from '@material-ui/core/TextField';
import Typography from '@material-ui/core/Typography';
import { makeStyles } from '@material-ui/core/styles';
import * as Yup from 'yup';

import history from '../Utilities/history';
import { useLogin } from '../Services/authenticationService';

const useStyles = makeStyles(theme => ({
    paper: {
        marginTop: theme.spacing(8),
        display: 'flex',
        flexDirection: 'column',
        alignItems: 'center',
        backgroundColor: '#ffffcc', // Yellow background color
        padding: theme.spacing(6), // Increased padding
        borderRadius: theme.spacing(3), // Rounded corners
    },
    form: {
        width: '80%', // Increased width of the form
        marginTop: theme.spacing(3),
    },
    submit: {
        margin: theme.spacing(4, 0, 2), // Adjusted margin
        backgroundColor: '#4caf50', // Green color for the button
        '&:hover': {
            backgroundColor: '#388e3c', // Darker shade of green on hover
        },
    },
    textField: {
        marginBottom: theme.spacing(4), // Increased space between fields
        fontSize: '1.2rem', // Increased font size
    },
    heading: {
        fontSize: '3rem', // Increased font size
        marginBottom: theme.spacing(4), // Increased space below heading
    },
}));

const Login = props => {
    const login = useLogin();
    const classes = useStyles();

    return (
        <div className={classes.paper}>
            <Typography component="h1" variant="h2" className={classes.heading} align="center">
                Sign in
            </Typography>
            <Formik
                initialValues={{
                    username: '',
                    password: '',
                }}
                validationSchema={Yup.object().shape({
                    username: Yup.string()
                        .required('Username is required')
                        .max(40, 'Username is too long'),
                    password: Yup.string()
                        .required('Password is required')
                        .max(100, 'Password is too long')
                        .min(6, 'Password too short'),
                })}
                onSubmit={(
                    { username, password },
                    { setStatus, setSubmitting }
                ) => {
                    setStatus();
                    login(username, password).then(
                        () => {
                            const { from } = history.location.state || {
                                from: { pathname: '/chat' },
                            };
                            history.push(from);
                        },
                        error => {
                            setSubmitting(false);
                            setStatus(error);
                        }
                    );
                }}
            >
                {({
                    handleSubmit,
                    handleChange,
                    values,
                    touched,
                    errors,
                }) => (
                    <form onSubmit={handleSubmit} className={classes.form}>
                        <TextField
                            id="username"
                            className={classes.textField}
                            name="username"
                            label="Username"
                            fullWidth
                            variant="outlined"
                            required
                            helperText={touched.username ? errors.username : ''}
                            error={touched.username && Boolean(errors.username)}
                            value={values.username}
                            onChange={handleChange}
                        />
                        <TextField
                            id="password"
                            className={classes.textField}
                            name="password"
                            label="Password"
                            fullWidth
                            variant="outlined"
                            required
                            helperText={touched.password ? errors.password : ''}
                            error={touched.password && Boolean(errors.password)}
                            value={values.password}
                            onChange={handleChange}
                            type="password"
                        />
                        <Button
                            type="submit"
                            fullWidth
                            variant="contained"
                            color="primary"
                            className={classes.submit}
                        >
```

```
                            Login
                        </Button>
                    </form>
                )}
            </Formik>
            <Grid container justify="flex-end">
                <Grid item>
                    <Typography>
                        <Link onClick={() =>
props.handleClick('register')} href="#">
                            Don't have an account?
                        </Link>
                    </Typography>
                </Grid>
            </Grid>
        </div>
    );
};

export default Login;

```

- **Register.jsx**

```
import React from 'react';
import { makeStyles } from '@material-ui/styles';
import Button from '@material-ui/core/Button';
import Grid from '@material-ui/core/Grid';
import Typography from '@material-ui/core/Typography';
import Link from '@material-ui/core/Link';
import TextField from '@material-ui/core/TextField';
import { Formik } from 'formik';
import * as Yup from 'yup';

import history from '../Utilities/history';
import { useRegister } from '../Services/authenticationService';

const useStyles = makeStyles(theme => ({
    paper: {
        marginTop: theme.spacing(8),
        display: 'flex',
        flexDirection: 'column',
        alignItems: 'center',
        backgroundColor: '#cce4f6', // Background color blue
        padding: theme.spacing(4), // Padding added
        borderRadius: theme.spacing(2), // Rounded corners
    },
    form: {
```

```
➢         width: '100%', // Full width
➢         marginTop: theme.spacing(1),
➢     },
➢     submit: {
➢         margin: theme.spacing(3, 0, 2),
➢     },
➢     textField: {
➢         marginBottom: theme.spacing(2), // Increased space
between fields
➢     },
➢ }));
➢
➢ const Register = props => {
➢     const register = useRegister();
➢     const classes = useStyles();
➢
➢     return (
➢         <div className={classes.paper}>
➢             <Grid container>
➢                 <Grid item>
➢                     <Typography component="h1" variant="h5"
align="center">
➢                         Register
➢                     </Typography>
➢                     <Formik
➢                         initialValues={{
➢                             name: '',
➢                             username: '',
➢                             password: '',
➢                             password2: '',
➢                         }}
➢                         validationSchema={Yup.object().shape({
➢                             name: Yup.string()
➢                                 .required('Name is required')
➢                                 .max(40, 'Too Long!'),
➢                             username: Yup.string()
➢                                 .required('Username is required')
➢                                 .max(40, 'Username address too
long'),
➢                             password: Yup.string()
➢                                 .required('Password is Required')
➢                                 .max(100, 'Password too long')
➢                                 .min(
➢                                     6,
➢                                     'Password should be at least
6 characters long'
➢                                 ),
➢                             password2: Yup.string().oneOf(
➢                                 [Yup.ref('password'), null],
➢                                 'Passwords do not match'
➢                             ),
➢                         })}
➢                         onSubmit={(
➢                             { name, username, password, password2
  },
➢                             { setStatus, setSubmitting }
➢                         ) => {
➢                             setStatus();
➢                             register(name, username, password,
  password2).then(
➢                                 user => {
➢                                     const { from } =
  history.location.state || {
➢                                         from: { pathname: '/chat'
  },
➢                                     };
➢                                     history.push(from);
➢                                 },
➢                                 error => {
➢                                     setSubmitting(false);
➢                                     setStatus(error);
➢                                 }
➢                             );
➢                         }}
➢                         validateOnChange={false}
➢                         validateOnBlur={false}
➢                     >
➢                         {({
➢                             handleSubmit,
➢                             handleChange,
➢                             values,
➢                             touched,
➢                             isValid,
➢                             errors,
➢                         }) => (
➢                             <form
➢                                 onSubmit={handleSubmit}
➢                                 className={classes.form}
➢                             >
➢                                 <TextField
➢                                     id="name"
➢                                     className={classes.textField}
➢                                     name="name"
➢                                     label="Name"
➢                                     fullWidth={true}
➢                                     variant="outlined"
➢                                        margin="normal"
➢                                        required={true}
➢                                        helperText={touched.name ?
   errors.name : ''}
➢                                        error={touched.name &&
   Boolean(errors.name)}
➢                                        value={values.name}
➢                                        onChange={handleChange}
➢                                    />
➢
➢                                    <TextField
➢                                        id="username"
➢                                        className={classes.textField}
➢                                        name="username"
➢                                        label="Username"
➢                                        fullWidth={true}
➢                                        variant="outlined"
➢                                        margin="normal"
➢                                        required={true}
➢                                        helperText={
➢                                            touched.username ?
   errors.username : ''
➢                                        }
➢                                        error={
➢                                            touched.username &&
➢                                            Boolean(errors.username)
➢                                        }
➢                                        value={values.username}
➢                                        onChange={handleChange}
➢                                    />
➢
➢                                    <TextField
➢                                        id="password"
➢                                        className={classes.textField}
➢                                        name="password"
➢                                        label="Password"
➢                                        fullWidth={true}
➢                                        variant="outlined"
➢                                        margin="normal"
➢                                        required={true}
➢                                        helperText={
➢                                            touched.password ?
   errors.password : ''
➢                                        }
➢                                        error={
➢                                            touched.password &&
➢                                            Boolean(errors.password)
➢                                        }
```

```
                                    value={values.password}
                                    onChange={handleChange}
                                    type="password"
                                />

                                <TextField
                                    id="password2"
                                    className={classes.textField}
                                    name="password2"
                                    label="Confirm Password"
                                    fullWidth={true}
                                    variant="outlined"
                                    margin="normal"
                                    required={true}
                                    helperText={
                                        touched.password2
                                            ? errors.password2
                                            : ''
                                    }
                                    error={
                                        touched.password2 &&
                                        Boolean(errors.password2)
                                    }
                                    value={values.password2}
                                    onChange={handleChange}
                                    type="password"
                                />

                                <Button
                                    type="submit"
                                    fullWidth={true}
                                    variant="contained"
                                    color="primary"
                                    className={classes.submit}
                                >
                                    Register
                                </Button>
                            </form>
                        )}
                    </Formik>
                </Grid>
                <Grid item xs={9}>
                    <Typography>
                        <Link
                            onClick={() =>
props.handleClick('login')}
                            href="#"
                        >
```

```
➢               Already have an account?
➢             </Link>
➢           </Typography>
➢         </Grid>
➢       </Grid>
➢     </div>
➢   );
➢ };
➢
➢ export default Register;
➢
```

5) Layout

➢ Header.jsx

```
➢ import React, { useState } from 'react';
➢ import { makeStyles } from '@material-ui/core/styles';
➢ import AppBar from '@material-ui/core/AppBar';
➢ import Toolbar from '@material-ui/core/Toolbar';
➢ import Button from '@material-ui/core/Button';
➢ import Menu from '@material-ui/core/Menu';
➢ import MenuItem from '@material-ui/core/MenuItem';
➢ import Link from '@material-ui/core/Link';
➢ import ArrowDropDownIcon from '@material-ui/icons/ArrowDropDown';
➢ import ArrowDropUpIcon from '@material-ui/icons/ArrowDropUp';
➢
➢ import { authenticationService } from
  '../Services/authenticationService';
➢ import history from '../Utilities/history';
➢ import logo from './logo.png';
➢
➢ const useStyles = makeStyles(theme => ({
➢   root: {
➢     flexGrow: 1,
➢   },
➢   title: {
➢     flexGrow: 1,
➢     display: 'flex',
➢   },
➢   userDropdown: {
➢     marginLeft: theme.spacing(2),
➢     padding: theme.spacing(1),
➢     [theme.breakpoints.down('xs')]: {
➢       marginLeft: 'auto',
➢     },
➢   },
➢ }));
➢
```

```
const Header = () => {
    const [currentUser] =
useState(authenticationService.currentUserValue);
    const [anchorEl, setAnchorEl] = useState(null);
    const [dropdownOpen, setDropdownOpen] = useState(false);

    const handleDropClose = () => {
        setDropdownOpen(false);
        setAnchorEl(null);
    };

    const handleDropOpen = event => {
        setDropdownOpen(true);
        setAnchorEl(event.currentTarget);
    };

    const handleLogout = () => {
        authenticationService.logout();
        history.push('/');
    };

    const arrowIcon = () => {
        if (dropdownOpen) {
            return <ArrowDropUpIcon />;
        }
        return <ArrowDropDownIcon />;
    };

    const classes = useStyles();

    return (
        <div className={classes.root}>
            <AppBar position="static">
                <Toolbar>
                    <Link href="/" className={classes.title}>
                        <img src={logo} alt="Logo" />
                    </Link>
                    <Button
                        aria-owns={anchorEl ? 'simple-menu' :
undefined}
                        aria-haspopup="true"
                        onClick={handleDropOpen}
                        className={classes.userDropdown}
                        color="inherit"
                    >
                        {currentUser.name}
                        {arrowIcon()}
                    </Button>
➢                       <Menu
➢                           id="simple-menu"
➢                           anchorEl={anchorEl}
➢                           open={Boolean(anchorEl)}
➢                           onClose={handleDropClose}
➢                           getContentAnchorEl={null}
➢                           anchorOrigin={{
➢                               vertical: 'bottom',
➢                               horizontal: 'right',
➢                           }}
➢                           transformOrigin={{
➢                               vertical: 'top',
➢                               horizontal: 'right',
➢                           }}
➢                       >
➢                           <MenuItem
   onClick={handleLogout}>Logout</MenuItem>
➢                       </Menu>
➢                   </Toolbar>
➢               </AppBar>
➢           </div>
➢       );
➢   };
➢
➢   export default Header;
➢
```

6) Services

➢ authenticationService.js

```
➢   import { BehaviorSubject } from 'rxjs';
➢   import { useSnackbar } from 'notistack';
➢
➢   import useHandleResponse from '../Utilities/handle-response';
➢
➢   const currentUserSubject = new BehaviorSubject(
➢       JSON.parse(localStorage.getItem('currentUser'))
➢   );
➢
➢   export const authenticationService = {
➢       logout,
➢       currentUser: currentUserSubject.asObservable(),
➢       get currentUserValue() {
➢           return currentUserSubject.value;
➢       },
➢   };
➢
➢   export function useLogin() {
    const { enqueueSnackbar } = useSnackbar();
    const handleResponse = useHandleResponse();

    const login = (username, password) => {
        const requestOptions = {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ username, password }),
        };

        return fetch(
            `${process.env.REACT_APP_API_URL}/api/users/login`,
            requestOptions
        )
            .then(handleResponse)
            .then(user => {
                localStorage.setItem('currentUser', JSON.stringify(user));
                currentUserSubject.next(user);
                return user;
            })
            .catch(function() {
                enqueueSnackbar('Failed to Login', {
                    variant: 'error',
                });
            });
    };

    return login;
}

export function useRegister() {
    const { enqueueSnackbar } = useSnackbar();
    const handleResponse = useHandleResponse();

    const register = (name, username, password, password2) => {
        const requestOptions = {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ name, username, password, password2 }),
        };

        return fetch(
            `${process.env.REACT_APP_API_URL}/api/users/register`,
            requestOptions
        )
```

```
            .then(handleResponse)
            .then(user => {
                localStorage.setItem('currentUser',
JSON.stringify(user));
                currentUserSubject.next(user);

                return user;
            })
            .catch(function(response) {
                if (response) {
                    enqueueSnackbar(response, {
                        variant: 'error',
                    });
                } else {
                    enqueueSnackbar('Failed to Register', {
                        variant: 'error',
                    });
                }
            });
    };

    return register;
}

function logout() {
    localStorage.removeItem('currentUser');
    currentUserSubject.next(null);
}

```

- **chatService.js**

```
import useHandleResponse from '../Utilities/handle-response';
import authHeader from '../Utilities/auth-header';
import { useSnackbar } from 'notistack';

// Receive global messages
export function useGetGlobalMessages() {
    const { enqueueSnackbar } = useSnackbar();
    const handleResponse = useHandleResponse();
    const requestOptions = {
        method: 'GET',
        headers: authHeader(),
    };

    const getGlobalMessages = () => {
        return fetch(
            `${process.env.REACT_APP_API_URL}/api/messages/global
`,
            requestOptions
        )
            .then(handleResponse)
            .catch(() =>
                enqueueSnackbar('Could not load Global Chat', {
                    variant: 'error',
                })
            );
    };

    return getGlobalMessages;
}

// Send a global message
export function useSendGlobalMessage() {
    const { enqueueSnackbar } = useSnackbar();
    const handleResponse = useHandleResponse();

    const sendGlobalMessage = body => {
        const requestOptions = {
            method: 'POST',
            headers: authHeader(),
            body: JSON.stringify({ body: body, global: true }),
        };

        return fetch(
            `${process.env.REACT_APP_API_URL}/api/messages/global`,
            requestOptions
        )
            .then(handleResponse)
            .catch(err => {
                console.log(err);
                enqueueSnackbar('Could send message', {
                    variant: 'error',
                });
            });
    };

    return sendGlobalMessage;
}

// Get list of users conversations
export function useGetConversations() {
    const { enqueueSnackbar } = useSnackbar();
    const handleResponse = useHandleResponse();
    const requestOptions = {
        method: 'GET',
        headers: authHeader(),
    };

    const getConversations = () => {
        return fetch(
            `${process.env.REACT_APP_API_URL}/api/messages/conversations`,
            requestOptions
        )
            .then(handleResponse)
            .catch(() =>
                enqueueSnackbar('Could not load chats', {
                    variant: 'error',
                })
            );
    };

    return getConversations;
}

// get conversation messages based on
// to and from id's
export function useGetConversationMessages() {
    const { enqueueSnackbar } = useSnackbar();
    const handleResponse = useHandleResponse();
    const requestOptions = {
        method: 'GET',
        headers: authHeader(),
    };

    const getConversationMessages = id => {
        return fetch(
            `${
                process.env.REACT_APP_API_URL
            }/api/messages/conversations/query?userId=${id}`,
            requestOptions
        )
            .then(handleResponse)
            .catch(() =>
                enqueueSnackbar('Could not load chats', {
                    variant: 'error',
                })
            );
    };

    return getConversationMessages;
}

```

```
export function useSendConversationMessage() {
    const { enqueueSnackbar } = useSnackbar();
    const handleResponse = useHandleResponse();

    const sendConversationMessage = (id, body) => {
        const requestOptions = {
            method: 'POST',
            headers: authHeader(),
            body: JSON.stringify({ to: id, body: body }),
        };

        return fetch(
            `${process.env.REACT_APP_API_URL}/api/messages/`,
            requestOptions
        )
            .then(handleResponse)
            .catch(err => {
                console.log(err);
                enqueueSnackbar('Could send message', {
                    variant: 'error',
                });
            });
    };

    return sendConversationMessage;
}

```

- **userService.js**

```
import useHandleResponse from '../Utilities/handle-response';
import authHeader from '../Utilities/auth-header';
import { useSnackbar } from 'notistack';

export function useGetUsers() {
    const { enqueueSnackbar } = useSnackbar();
    const handleResponse = useHandleResponse();
    const requestOptions = {
        method: 'GET',
        headers: authHeader(),
    };

    const getUsers = () => {
        return fetch(
            `${process.env.REACT_APP_API_URL}/api/users`,
            requestOptions
        )
            .then(handleResponse)
            .catch(() =>
➢                 enqueueSnackbar('Could not load Users', {
➢                     variant: 'error',
➢                 })
➢             );
➢     };
➢
➢     return getUsers;
➢ }
➢
```

7) Utilities

➢ auth-header.js

```
➢ import { authenticationService } from
  '../Services/authenticationService';
➢
➢ function authHeader() {
➢     const currentUser = authenticationService.currentUserValue;
➢     if (currentUser && currentUser.token) {
➢         return {
➢             Authorization: `${currentUser.token}`,
➢             'Content-Type': 'application/json',
➢         };
➢     } else {
➢         return {};
➢     }
➢ }
➢
➢ export default authHeader;
➢
```

➢ common.js

```
➢ export default {
➢   getInitialsFromName: (name) => {
➢     const letters = String(name)
➢       .split(" ")
➢       .map((i) => i.charAt(0));
➢     return letters.join("");
➢   },
➢ };
➢
```

➢ handle-response.js

```
➢ import { authenticationService } from
  '../Services/authenticationService';
➢ import { useSnackbar } from 'notistack';
➢
➢ const useHandleResponse = () => {
➢     const { enqueueSnackbar } = useSnackbar();
➢
```

```
    const handleResponse = response => {
        return response.text().then(text => {
            const data = text && JSON.parse(text);
            if (!response.ok) {
                if ([401, 403].indexOf(response.status) !== -1) {
                    authenticationService.logout();
                    enqueueSnackbar('User Unauthorized', {
                        variant: 'error',
                    });
                }

                const error = (data && data.message) || response.statusText;
                return Promise.reject(error);
            }

            return data;
        });
    };

    return handleResponse;
};

export default useHandleResponse;

```

- **history.js**

```
import { createBrowserHistory } from 'history';

const history = createBrowserHistory();

export default history;

```

- **private-route.js**

```
import React from 'react';
import { Route, Redirect } from 'react-router-dom';

import { authenticationService } from '../Services/authenticationService';

const PrivateRoute = ({ component: Component, ...rest }) => (
    <Route
        {...rest}
        render={props => {
            const currentUser = authenticationService.currentUserValue;
            if (!currentUser) {
                // not logged in so redirect to login page with the return url
                return (
                    <Redirect
                        to={{ pathname: '/', state: { from:
props.location } }}
                    />
                );
            }

            // authorised so return component
            return <Component {...props} />;
        }}
    />
);

export default PrivateRoute;

```

➢ **App.jss**

```
import React from 'react';
import { Router, Route } from 'react-router-dom';
import { createMuiTheme } from '@material-ui/core/styles';
import CssBaseline from '@material-ui/core/CssBaseline';
import { ThemeProvider } from '@material-ui/styles';
import { SnackbarProvider } from 'notistack';

import history from './Utilities/history';
import PrivateRoute from './Utilities/private-route';
import Home from './Home/Home';
import Chat from './Chat/Chat';

const theme = createMuiTheme({
    palette: {
        primary: {
            light: '#58a5f0',
            main: '#0277bd',
            dark: '#004c8c',
        },
        secondary: {
            light: '#ffd95a',
            main: '#f9a825',
            dark: '#c17900',
            contrastText: '#212121',
        },
        background: {
            default: '#f0f0f0',
        },
    },
    typography: {
        useNextVariants: true,
➢       },
➢   });
➢
➢   function App() {
➢       return (
➢           <ThemeProvider theme={theme}>
➢               <CssBaseline />
➢               <SnackbarProvider maxSnack={3}
    autoHideDuration={3000}>
➢                   <Router history={history}>
➢                       <Route path="/" exact component={Home} />
➢                       <PrivateRoute path="/chat" component={Chat}
    />
➢                   </Router>
➢               </SnackbarProvider>
➢           </ThemeProvider>
➢       );
➢   }
➢
➢   export default App;
➢
```

➢ **Index.css**

```
➢   @import
    url('https://fonts.googleapis.com/css?family=Roboto:300,400,500&d
    isplay=swap');
➢
➢   body {
➢       margin: 0;
➢   }
➢
➢   import React from 'react';
➢   import ReactDOM from 'react-dom';
➢   import './index.css';
➢   import App from './App';
➢   import * as serviceWorker from './serviceWorker';
➢
➢   ReactDOM.render(<App />, document.getElementById('root'));
➢
➢   // If you want your app to work offline and load faster, you can
    change
➢   // unregister() to register() below. Note this comes with some
    pitfalls.
➢   // Learn more about service workers: https://bit.ly/CRA-PWA
➢   serviceWorker.unregister();
➢
```

8) config

- **keys.js**

```
module.exports = {
  mongoURI: "mongodb://127.0.0.1:27017/chat-app",
  secretOrKey: "secret",
};

```

- **passport.js**

```
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
const mongoose = require('mongoose');
const User = mongoose.model('users');
const keys = require('../config/keys');
const opts = {};
opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
opts.secretOrKey = keys.secretOrKey;
module.exports = passport => {
    passport.use(
        new JwtStrategy(opts, (jwt_payload, done) => {
            User.findById(jwt_payload.id)
                .then(user => {
                    if (user) {
                        return done(null, user);
                    }
                    return done(null, false);
                })
                .catch(err => console.log(err));
        })
    );
};

```

9) Models

- **Conversation.js**

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

// Create Schema for Users
const ConversationSchema = new Schema({
    recipients: [{ type: Schema.Types.ObjectId, ref: 'users' }],
    lastMessage: {
        type: String,
    },
    date: {
        type: String,
        default: Date.now,
    },
});

module.exports = Conversation = mongoose.model(
    'conversations',
    ConversationSchema
);

const mongoose = require('mongoose');
const Schema = mongoose.Schema;

// Create Schema for Users
const GlobalMessageSchema = new Schema({
    from: {
        type: Schema.Types.ObjectId,
        ref: 'users',
    },
    body: {
        type: String,
        required: true,
    },
    date: {
        type: String,
        default: Date.now,
    },
});

module.exports = GlobalMessage = mongoose.model(
    'global_messages',
    GlobalMessageSchema
);

```

- **Message.js**

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

// Create Schema for Users
const MessageSchema = new Schema({
    conversation: {
        type: Schema.Types.ObjectId,
        ref: 'conversations',
    },
    to: {
        type: Schema.Types.ObjectId,
        ref: 'users',
    },
    from: {
        type: Schema.Types.ObjectId,
        ref: 'users',
    },
    body: {
        type: String,
        required: true,
    },
    date: {
        type: String,
        default: Date.now,
    },
});

module.exports = Message = mongoose.model('messages',
MessageSchema);

```

- **User.js**

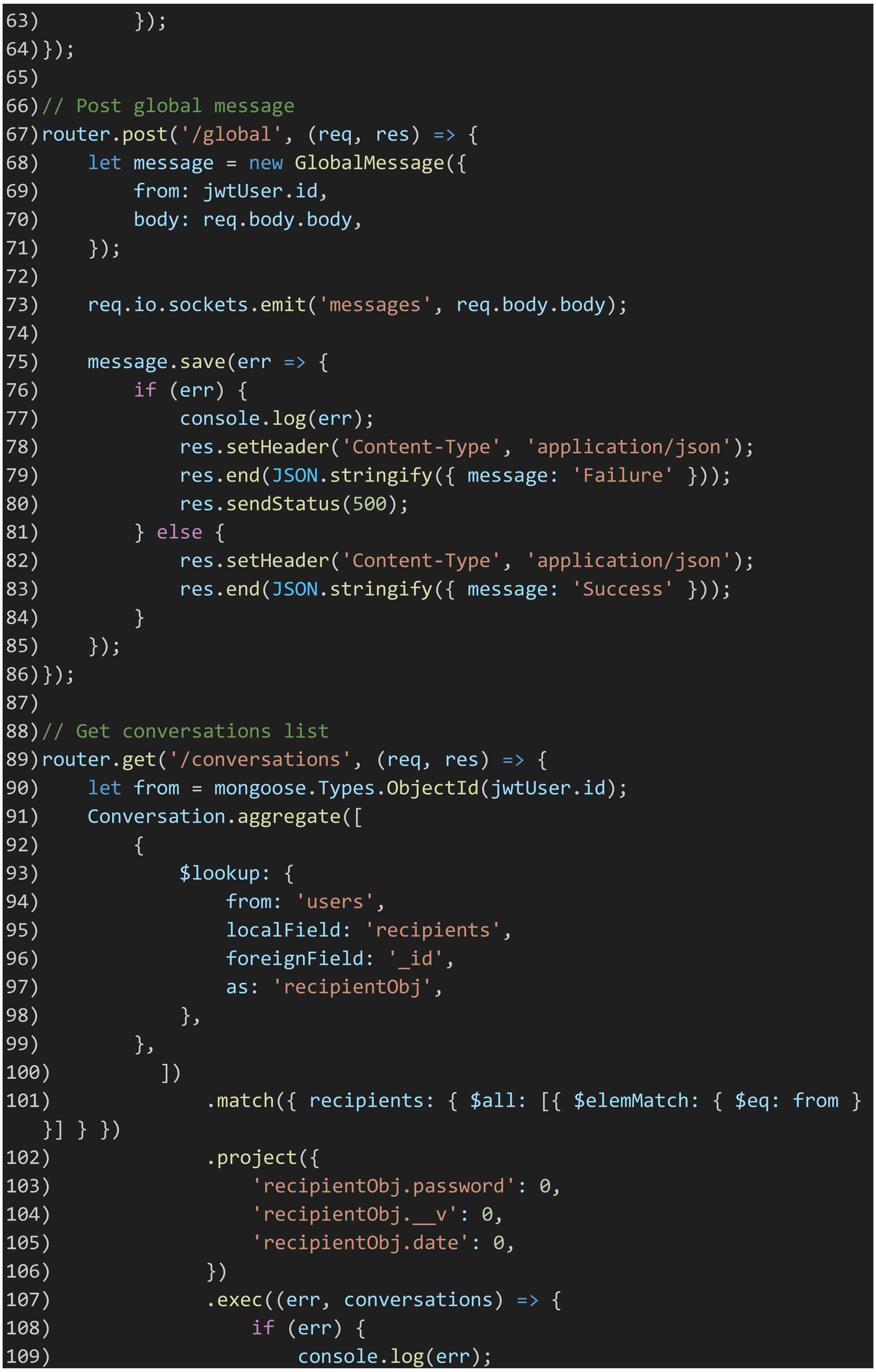
```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

// Create Schema for Users
const UserSchema = new Schema({
    name: {
        type: String,
        required: true,
    },
    username: {
        type: String,
        required: true,
    },
    password: {
        type: String,
        required: true,
    },
    date: {
        type: String,
        default: Date.now,
    },
});

module.exports = User = mongoose.model('users', UserSchema);

```

10) Routes\api

- **Messages.js**

```
11)const express = require('express');
12)const jwt = require('jsonwebtoken');
13)const mongoose = require('mongoose');
14)const router = express.Router();
```

```

const keys = require('../../config/keys');
const verify = require('../../utilities/verify-token');
const Message = require('../../models/Message');
const Conversation = require('../../models/Conversation');
const GlobalMessage = require('../../models/GlobalMessage');

let jwtUser = null;

// Token verfication middleware
router.use(function(req, res, next) {
    try {
        jwtUser = jwt.verify(verify(req), keys.secretOrKey);
        next();
    } catch (err) {
        console.log(err);
        res.setHeader('Content-Type', 'application/json');
        res.end(JSON.stringify({ message: 'Unauthorized' }));
        res.sendStatus(401);
    }
});

// Get global messages
router.get('/global', (req, res) => {
    GlobalMessage.aggregate([
        {
            $lookup: {
                from: 'users',
                localField: 'from',
                foreignField: '_id',
                as: 'fromObj',
            },
        },
    ])
        .project({
            'fromObj.password': 0,
            'fromObj.__v': 0,
            'fromObj.date': 0,
        })
        .exec((err, messages) => {
            if (err) {
                console.log(err);
                res.setHeader('Content-Type', 'application/json');
                res.end(JSON.stringify({ message: 'Failure' }));
                res.sendStatus(500);
            } else {
                res.send(messages);
            }
```



```
        });
});

// Post global message
router.post('/global', (req, res) => {
    let message = new GlobalMessage({
        from: jwtUser.id,
        body: req.body.body,
    });

    req.io.sockets.emit('messages', req.body.body);

    message.save(err => {
        if (err) {
            console.log(err);
            res.setHeader('Content-Type', 'application/json');
            res.end(JSON.stringify({ message: 'Failure' }));
            res.sendStatus(500);
        } else {
            res.setHeader('Content-Type', 'application/json');
            res.end(JSON.stringify({ message: 'Success' }));
        }
    });
});

// Get conversations list
router.get('/conversations', (req, res) => {
    let from = mongoose.Types.ObjectId(jwtUser.id);
    Conversation.aggregate([
        {
            $lookup: {
                from: 'users',
                localField: 'recipients',
                foreignField: '_id',
                as: 'recipientObj',
            },
        },
        ])
            .match({ recipients: { $all: [{ $elemMatch: { $eq: from } }] } })
            .project({
                'recipientObj.password': 0,
                'recipientObj.__v': 0,
                'recipientObj.date': 0,
            })
            .exec((err, conversations) => {
                if (err) {
                    console.log(err);
                res.setHeader('Content-Type', 'application/json');
                res.end(JSON.stringify({ message: 'Failure' }));
                res.sendStatus(500);
            } else {
                res.send(conversations);
            }
        });
});

// Get messages from conversation
// based on to & from
router.get('/conversations/query', (req, res) => {
    let user1 = mongoose.Types.ObjectId(jwtUser.id);
    let user2 = mongoose.Types.ObjectId(req.query.userId);
    Message.aggregate([
        {
            $lookup: {
                from: 'users',
                localField: 'to',
                foreignField: '_id',
                as: 'toObj',
            },
        },
        {
            $lookup: {
                from: 'users',
                localField: 'from',
                foreignField: '_id',
                as: 'fromObj',
            },
        },
    ])
        .match({
            $or: [
                { $and: [{ to: user1 }, { from: user2 }] },
                { $and: [{ to: user2 }, { from: user1 }] },
            ],
        })
        .project({
            'toObj.password': 0,
            'toObj.__v': 0,
            'toObj.date': 0,
            'fromObj.password': 0,
            'fromObj.__v': 0,
            'fromObj.date': 0,
        })
        .exec((err, messages) => {
```

```javascript
                if (err) {
                    console.log(err);
                    res.setHeader('Content-Type', 'application/json');
                    res.end(JSON.stringify({ message: 'Failure' }));
                    res.sendStatus(500);
                } else {
                    res.send(messages);
                }
            });
    });

    // Post private message
    router.post('/', (req, res) => {
        let from = mongoose.Types.ObjectId(jwtUser.id);
        let to = mongoose.Types.ObjectId(req.body.to);

        Conversation.findOneAndUpdate(
            {
                recipients: {
                    $all: [
                        { $elemMatch: { $eq: from } },
                        { $elemMatch: { $eq: to } },
                    ],
                },
            },
            {
                recipients: [jwtUser.id, req.body.to],
                lastMessage: req.body.body,
                date: Date.now(),
            },
            { upsert: true, new: true, setDefaultsOnInsert: true },
            function(err, conversation) {
                if (err) {
                    console.log(err);
                    res.setHeader('Content-Type', 'application/json');
                    res.end(JSON.stringify({ message: 'Failure' }));
                    res.sendStatus(500);
                } else {
                    let message = new Message({
                        conversation: conversation._id,
                        to: req.body.to,
                        from: jwtUser.id,
                        body: req.body.body,
                    });

                    req.io.sockets.emit('messages', req.body.body);
                message.save(err => {
                    if (err) {
                        console.log(err);
                        res.setHeader('Content-Type', 'application/json');
                        res.end(JSON.stringify({ message: 'Failure' }));
                        res.sendStatus(500);
                    } else {
                        res.setHeader('Content-Type', 'application/json');
                        res.end(
                            JSON.stringify({
                                message: 'Success',
                                conversationId: conversation._id,
                            })
                        );
                    }
                });
            }
        }
    );
});

module.exports = router;
```

➢ **User.js**

```
const express = require("express");
const router = express.Router();
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const mongoose = require("mongoose");

const keys = require("../../config/keys");
const verify = require("../../utilities/verify-token");
const validateRegisterInput = require("../../validation/register");
const validateLoginInput = require("../../validation/login");
const User = require("../../models/User");

router.get("/", (req, res) => {
  try {
    let jwtUser = jwt.verify(verify(req), keys.secretOrKey);
    let id = mongoose.Types.ObjectId(jwtUser.id);

    User.aggregate()
      .match({ _id: { $not: { $eq: id } } })
      .project({
        password: 0,
        __v: 0,
        date: 0,
      })
      .exec((err, users) => {
        if (err) {
          console.log(err);
          res.setHeader("Content-Type", "application/json");
          res.end(JSON.stringify({ message: "Failure" }));
          res.sendStatus(500);
        } else {
          res.send(users);
        }
      });
  } catch (err) {
    console.log(err);
    res.setHeader("Content-Type", "application/json");
    res.end(JSON.stringify({ message: "Unauthorized" }));
    res.sendStatus(401);
  }
});

router.post("/register", (req, res) => {
  // Form validation
  const { errors, isValid } = validateRegisterInput(req.body);
  // Check validation
  if (!isValid) {
    return res.status(400).json(errors);
  }
  User.findOne({ username: req.body.username }).then((user) => {
    if (user) {
      return res.status(400).json({ message: "Username already exists" });
    } else {
      const newUser = new User({
        name: req.body.name,
        username: req.body.username,
        password: req.body.password,
      });
      // Hash password before saving in database
      bcrypt.genSalt(10, (err, salt) => {
        bcrypt.hash(newUser.password, salt, (err, hash) => {
          if (err) throw err;
          newUser.password = hash;
          newUser
            .save()
            .then((user) => {
              const payload = {
                id: user.id,
                name: user.name,
              };
              // Sign token
              jwt.sign(
                payload,
                keys.secretOrKey,
                {
                  expiresIn: 31556926, // 1 year in seconds
                },
                (err, token) => {
                  if (err) {
                    console.log(err);
                  } else {
                    req.io.sockets.emit("users", user.username);
                    res.json({
                      success: true,
                      token: "Bearer " + token,
                      name: user.name,
                    });
                  }
                }
              );
            })
            .catch((err) => console.log(err));
        });
      });
    }
  });
});

router.post("/login", (req, res) => {
  // Form validation
  const { errors, isValid } = validateLoginInput(req.body);
  // Check validation
  if (!isValid) {
    return res.status(400).json(errors);
  }
  const username = req.body.username;
  const password = req.body.password;
  // Find user by username
  User.findOne({ username }).then((user) => {
    // Check if user exists
    if (!user) {
      return res.status(404).json({ usernamenotfound: "Username
not found" });
    }
```

```
➢     // Check password
➢     bcrypt.compare(password, user.password).then((isMatch) => {
➢       if (isMatch) {
➢         // User matched
➢         // Create JWT Payload
➢         const payload = {
➢           id: user.id,
➢           name: user.name,
➢         };
➢         // Sign token
➢         jwt.sign(
➢           payload,
➢           keys.secretOrKey,
➢           {
➢             expiresIn: 31556926, // 1 year in seconds
➢           },
➢           (err, token) => {
➢             res.json({
➢               success: true,
➢               token: "Bearer " + token,
➢               name: user.name,
➢               username: user.username,
➢               userId: user._id,
➢             });
➢           }
➢         );
➢       } else {
➢         return res
➢           .status(400)
➢           .json({ passwordincorrect: "Password incorrect" });
➢       }
➢     });
➢   });
➢ });
➢
➢ module.exports = router;
➢
```

11) utilities

- **Verify-token.js**

```
➢ const verify = req => {
➢     if (
➢         req.headers.authorization &&
➢         req.headers.authorization.split(' ')[0] === 'Bearer'
➢     )
➢         return req.headers.authorization.split(' ')[1];
➢     return null;
};

module.exports = verify;

```

12) validation

➢ **Login.js**

```
const Validator = require('validator');
const isEmpty = require('is-empty');
module.exports = function validateLoginInput(data) {
    let errors = {};

    // Converts empty fields to String in order to validate them
    data.username = !isEmpty(data.username) ? data.username : '';
    data.password = !isEmpty(data.password) ? data.password : '';

    if (Validator.isEmpty(data.username)) {
        errors.username = 'Username field is required';
    }

    if (Validator.isEmpty(data.password)) {
        errors.password = 'Password field is required';
    }
    return {
        errors,
        isValid: isEmpty(errors),
    };
};

```

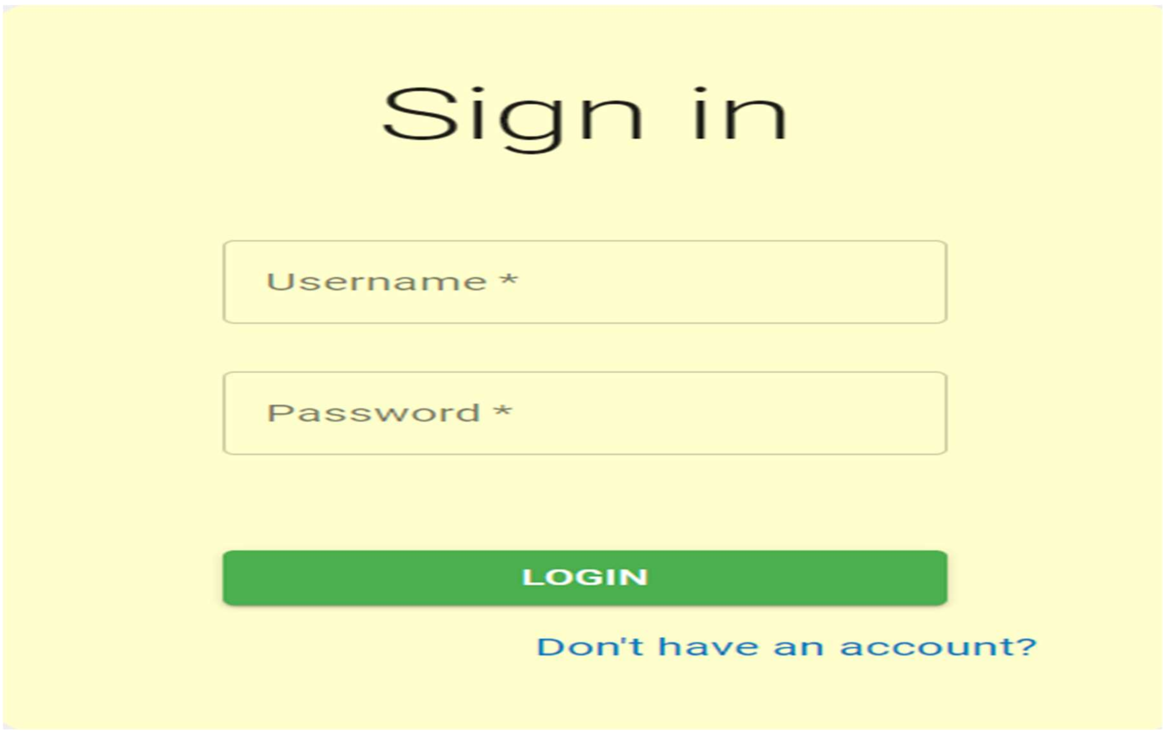
➢ **Register.js**

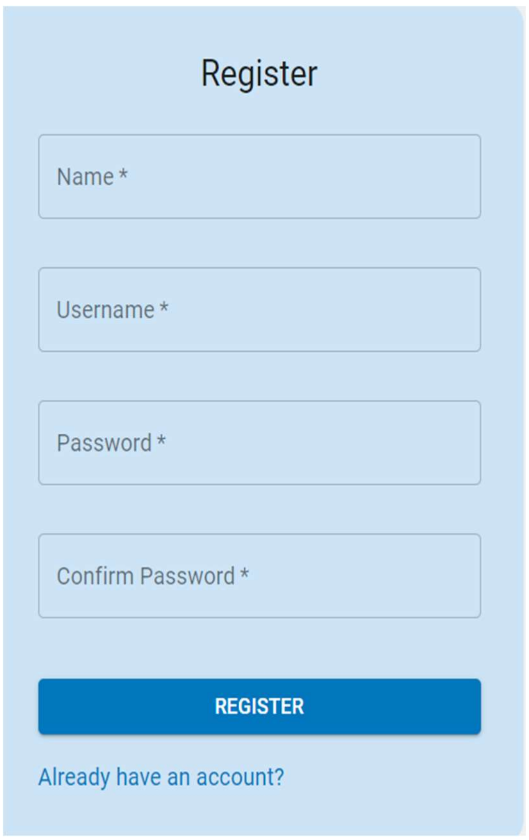
```
const Validator = require('validator');
const isEmpty = require('is-empty');

module.exports = function validateRegisterInput(data) {
    let errors = {};

    // Converts empty fields to String in order to validate them
    data.name = !isEmpty(data.name) ? data.name : '';
    data.username = !isEmpty(data.username) ? data.username : '';
    data.password = !isEmpty(data.password) ? data.password : '';
    data.password2 = !isEmpty(data.password2) ? data.password2 : '';

    if (Validator.isEmpty(data.name)) {
        errors.name = 'Name field is required';
    }

➢     if (Validator.isEmpty(data.username)) {
➢         errors.username = 'Username field is required';
➢     }
➢
➢     if (Validator.isEmpty(data.password)) {
➢         errors.password = 'Password field is required';
➢     }
➢     if (Validator.isEmpty(data.password2)) {
➢         errors.password2 = 'Confirm password field is required';
➢     }
➢     if (!Validator.isLength(data.password, { min: 6, max: 30 })) {
➢         errors.password = 'Password must be at least 6 characters';
➢     }
➢     if (!Validator.equals(data.password, data.password2)) {
➢         errors.password2 = 'Passwords must match';
➢     }
➢
➢     return {
➢         errors,
➢         isValid: isEmpty(errors),
➢     };
➢ };
➢
```

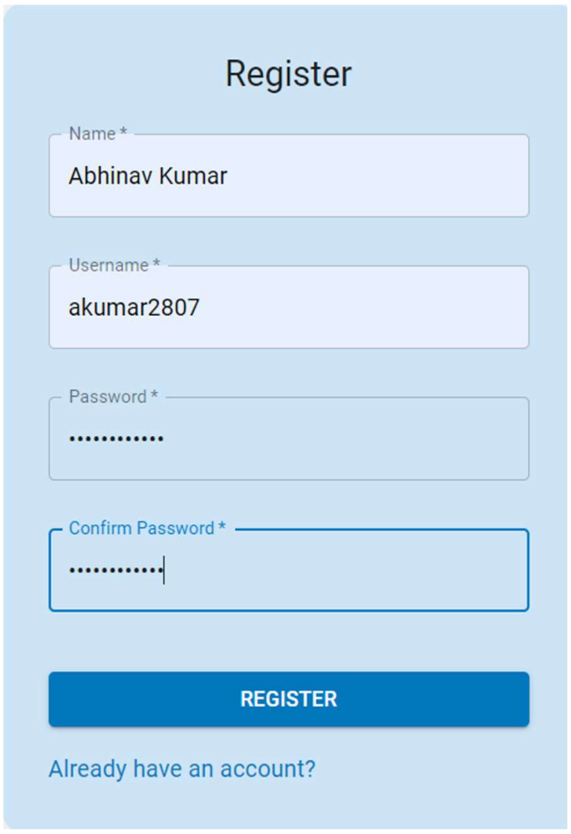
13) server.js

```
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const passport = require("passport");
const cors = require("cors");

const users = require("./routes/api/users");
const messages = require("./routes/api/messages");

const app = express();

// Port that the webserver listens to
const port = process.env.PORT || 5000;

const server = app.listen(port, () =>
  console.log(`Server running on port ${port}`)
);

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

// Body Parser middleware to parse request bodies
app.use(
  bodyParser.urlencoded({
    extended: false,
  })
);
app.use(bodyParser.json());

// CORS middleware
app.use(cors());

// Database configuration
const db = require("./config/keys").mongoURI;

mongoose
  .connect(db, {
    useNewUrlParser: true,
    useFindAndModify: false,
    useUnifiedTopology: true,
  })
  .then(() => console.log("MongoDB Successfully Connected"))
  .catch((err) => console.log(err));

// Passport middleware
app.use(passport.initialize());
// Passport config
require("./config/passport")(passport);

// Assign socket object to every request
app.use(function (req, res, next) {
  req.io = io;
  next();
});

// Routes
app.use("/api/users", users);
app.use("/api/messages", messages);
```

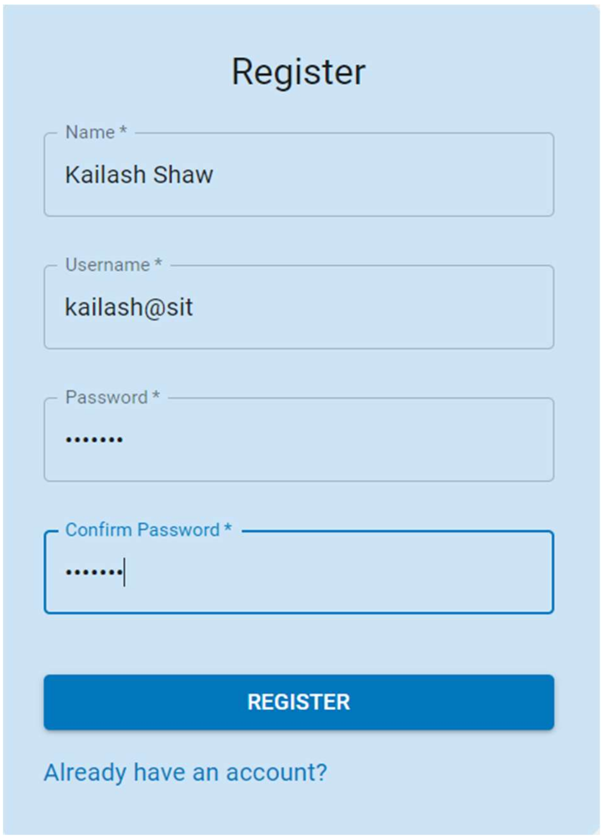
Output







Name *
Abhinav Kumar
Username *
akumar2807
Password *
••••••••••••
Confirm Password *
••••••••••••



Register

Name *

Kailash Shaw

Username *

kailash@sit

Password *

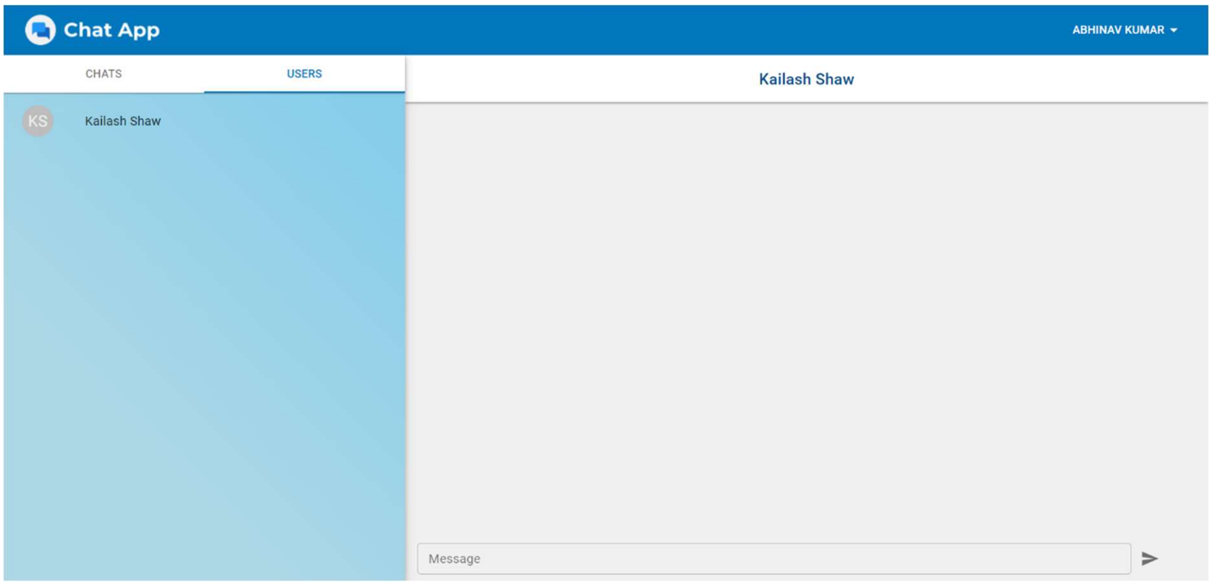
•••••••

Confirm Password *

•••••••

REGISTER

Already have an account?



Chat App

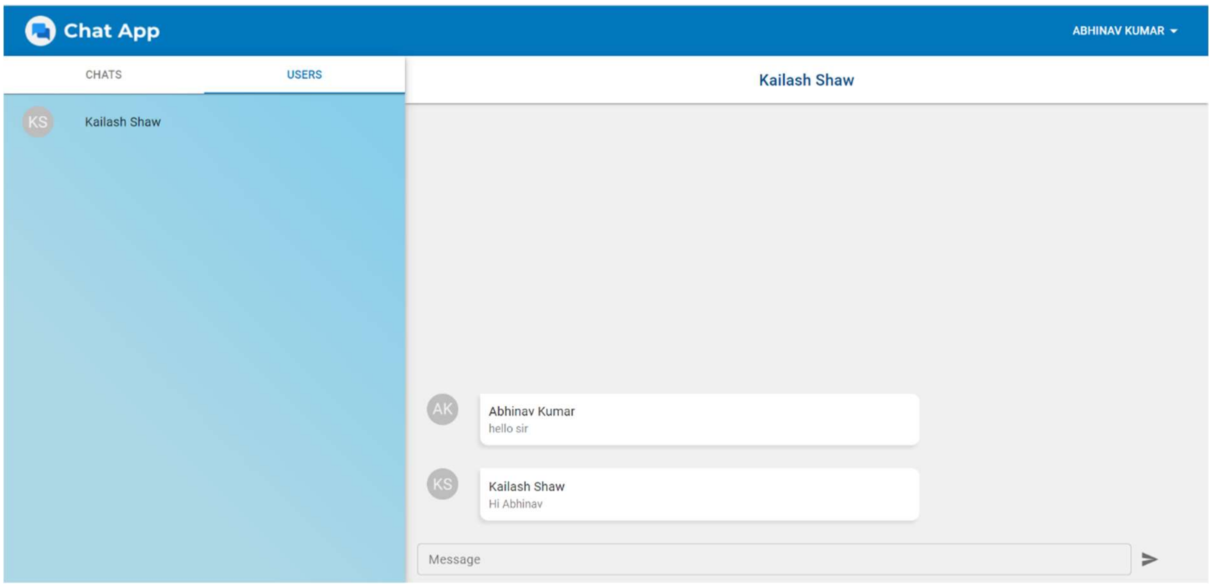
ABHINAV KUMAR

CHATS | USERS

KS Kailash Shaw

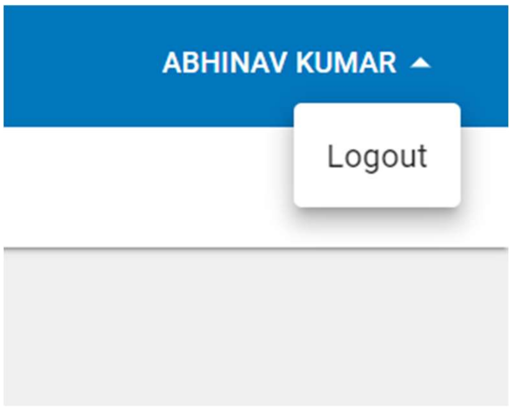
Kailash Shaw

Message



Abhinav Kumar
hello sir

Kailash Shaw
Hi Abhinav



Conclusion

In conclusion, this project showcases the power and versatility of the MERN stack in building a full-fledged chat application. By leveraging MongoDB for data storage, Express.js and Node.js for server-side logic, and React.js for dynamic user interfaces, we've created a robust platform for real-time communication. The integration of authentication using JWT tokens ensures secure access to the application, while features like global and private chat functionalities enhance user interaction. With real-time updates for user lists, conversations, and messages, users can enjoy seamless communication experiences. This project not only demonstrates the capabilities of the MERN stack but also serves as a valuable resource for developers looking to explore and master full-stack web development.

References

- ➢ **ChatGPT:**

  Website: OpenAI ChatGPT
- ➢ **GitHub:**

  Website: GitHub
- ➢ **GeeksforGeeks:**

  Website: GeeksforGeeks
- ➢ **Stack Overflow:**

  Website: Stack Overflow
- ➢ **College Material: Provided by college faculty**