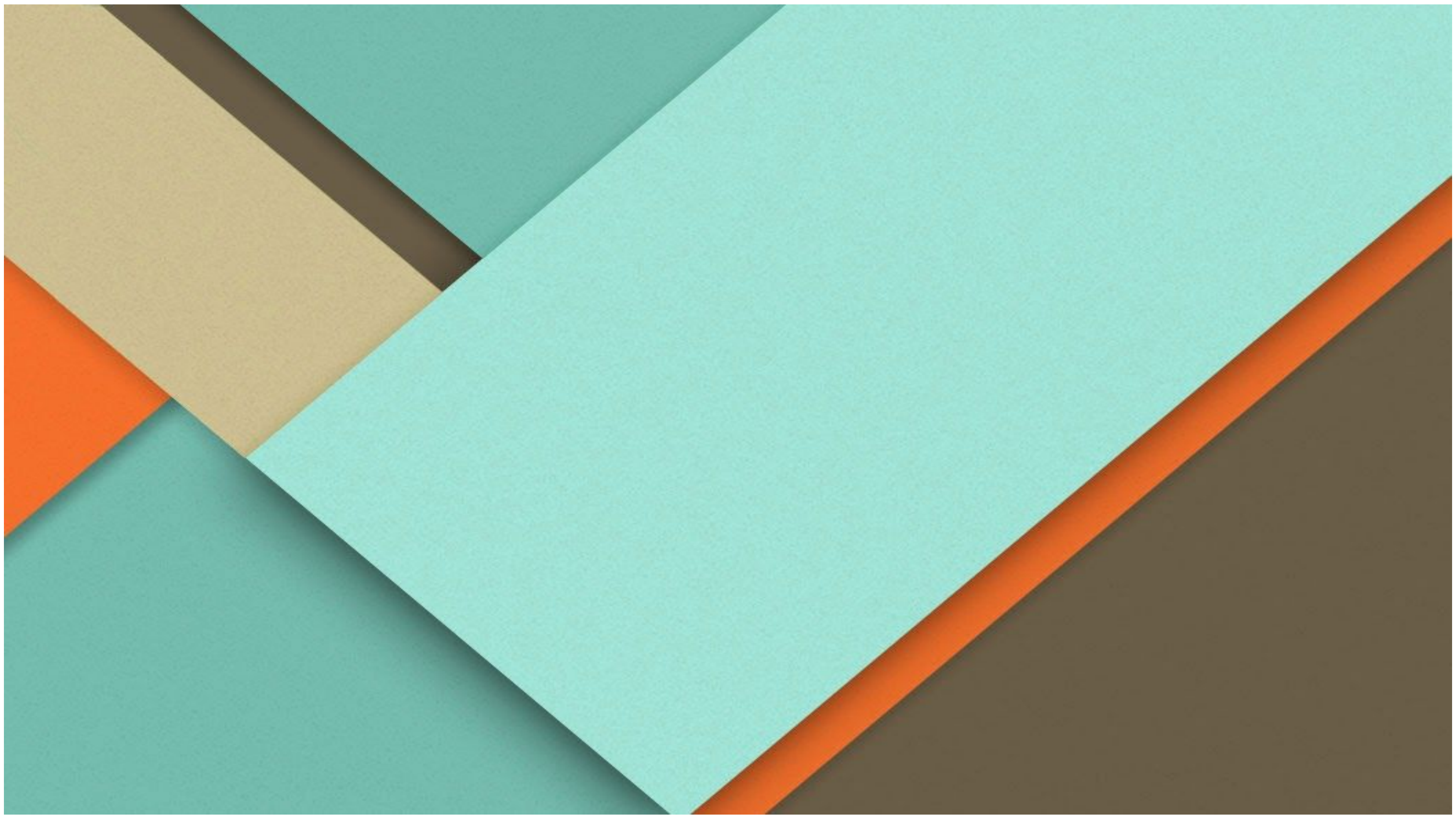


Group Chat App

Submitted By: Group 9

1. Abhishek Kumar - 17114005
2. Gagan Kumre - 17114028
3. Gandhi Ronnie - 17114029
4. Hemant Singh - 17114038
5. Kishan Kumar - 17114046
6. Siddhant Nayak - 17114071

Contribution Summary

I. Siddhant Nayak (17114071) Pg 2-3

Worked on the Backend Architecture for User Model and Database. Also integrated the user model to the public_chat page route.

II. Hemant Singh (17114038) Pg 3-4

Worked on the backend routing which included integration of socket.io with the rest of the backend of the app, the user model to the login page route and register page route, and also took care of logout on the public chat page.

III. Gagan Kumre (17114028) Pg 5-6

Worked on implementation of signup, authentication function, validation of college email address and hashing of users password with the use of passport.js and mongodb

IV. Abhishek Kumar (17114005) Pg 7-8

Worked on emitting events (send and receive any events with any data) from client to server and broadcasting (emit the event from the server to the rest of the users).

V. Kishan Kumar (17114046) Pg 9-10

Designed complete UI and some of the frontend backend integration.

VI. Gandhi Ronnie (17114029) Pg 11-14

Coding the front-end (UI) of the app and connecting the backend with the frontend in Express.JS.

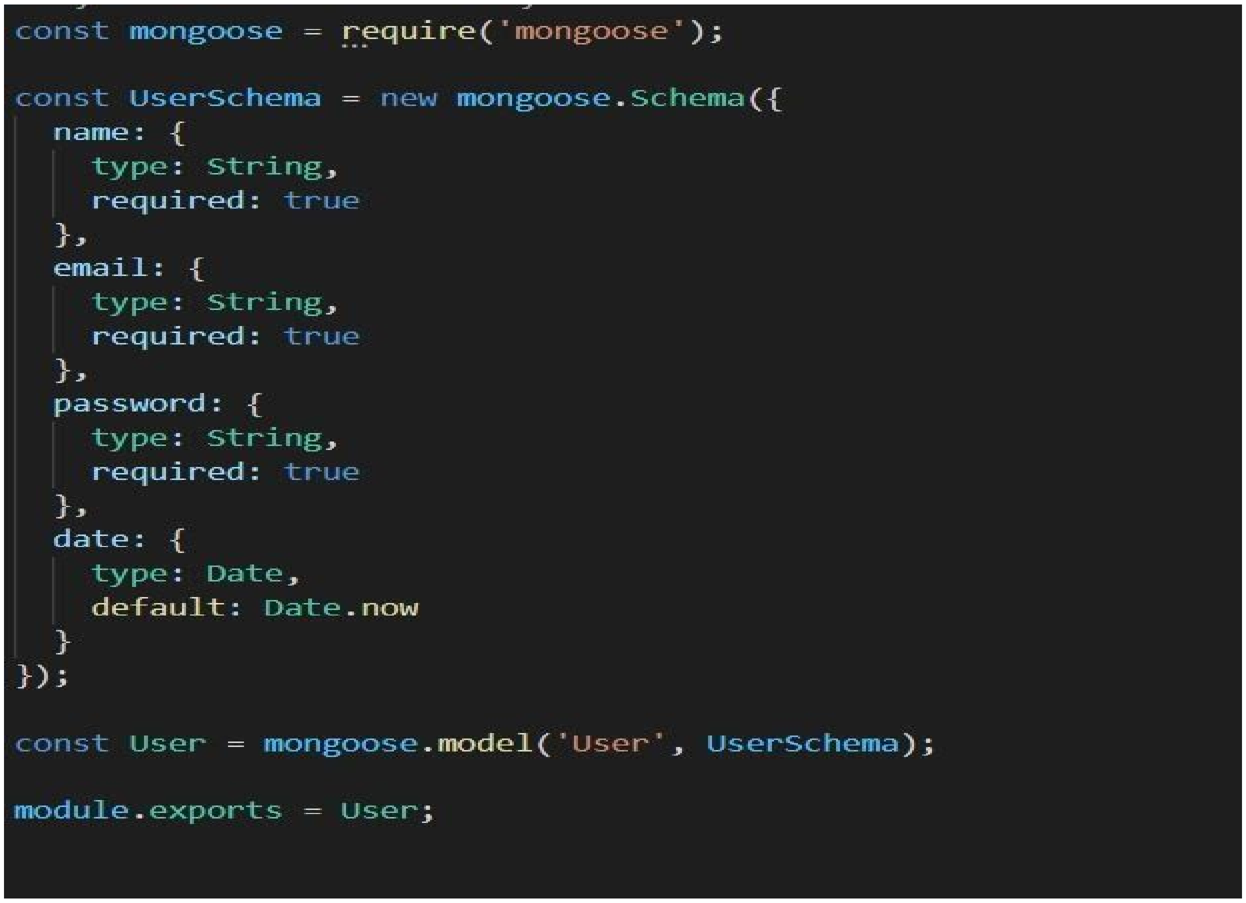
Source Code : https://github.com/gagankumre/ACN-Project

Introduction

We have used Socket programming to build a platform for reliable conveyance and smooth flow of information from the administration to the students alongside functionality of students being able to communicate with each other as well. The platform (Group chat app) has an authentication feature as well so only relevant people (Students of same college) can participate and use the platform.

Backend Architecture for User Model + Database

- The app has a User model that defines basic information of a user like name, email, password and date.
- The information for all users is stored in a MongoDB database

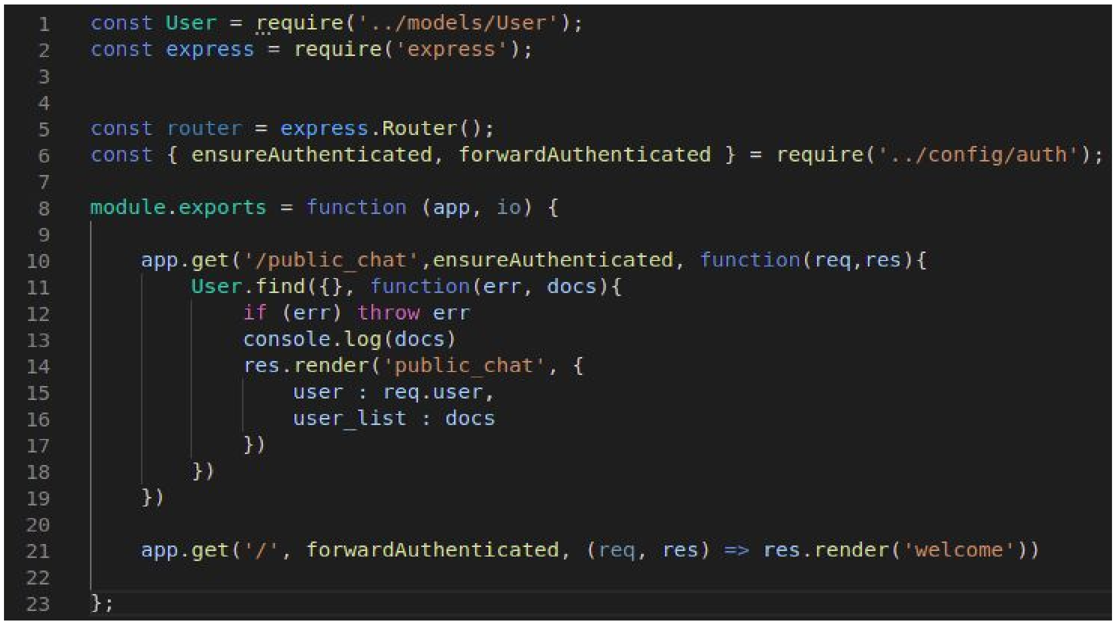


```
const mongoose = require('mongoose');

const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  email: {
    type: String,
    required: true
  },
  password: {
    type: String,
    required: true
  },
  date: {
    type: Date,
    default: Date.now
  }
});

const User = mongoose.model('User', UserSchema);

module.exports = User;
```

Backend Architecture for creating routes

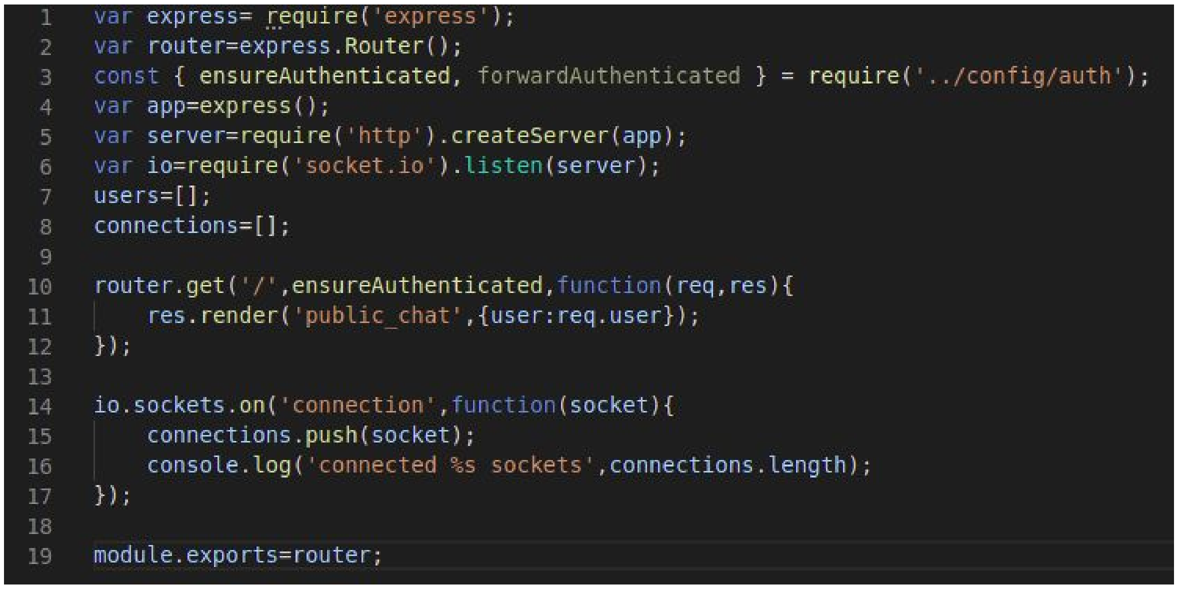
There are following route for each purpose:

- **Index.Js** : It integrates the user model to the public_chat page route.



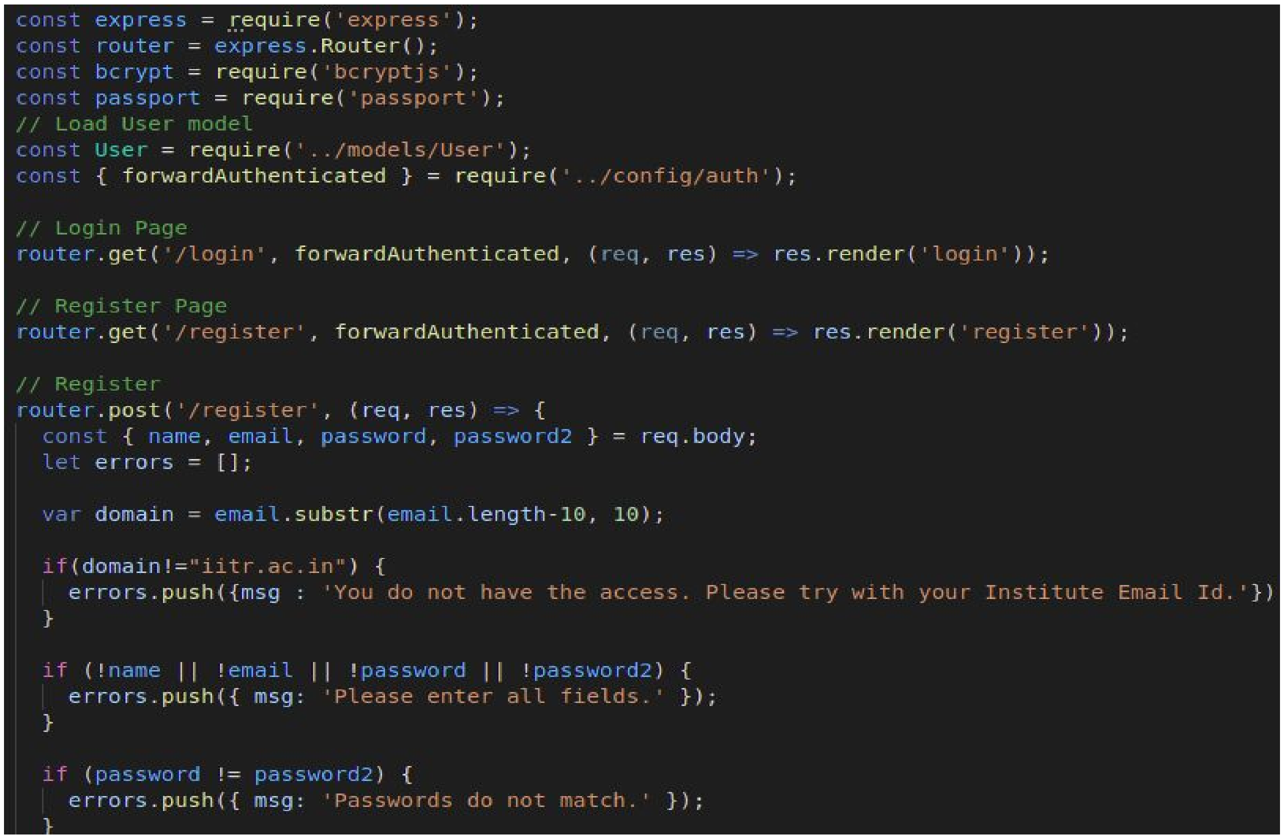
```
const User = require('../models/User');
const express = require('express');


const router = express.Router();
const { ensureAuthenticated, forwardAuthenticated } = require('../config/auth');

module.exports = function (app, io) {

    app.get('/public_chat',ensureAuthenticated, function(req,res){
        User.find({}, function(err, docs){
            if (err) throw err
            console.log(docs)
            res.render('public_chat', {
                user : req.user,
                user_list : docs
            })
        })
    })

    app.get('/', forwardAuthenticated, (req, res) => res.render('welcome'))

};
```

- **public_chat.Js**: It handles integration of socket.io with the rest of the backend of the app by building a connection.

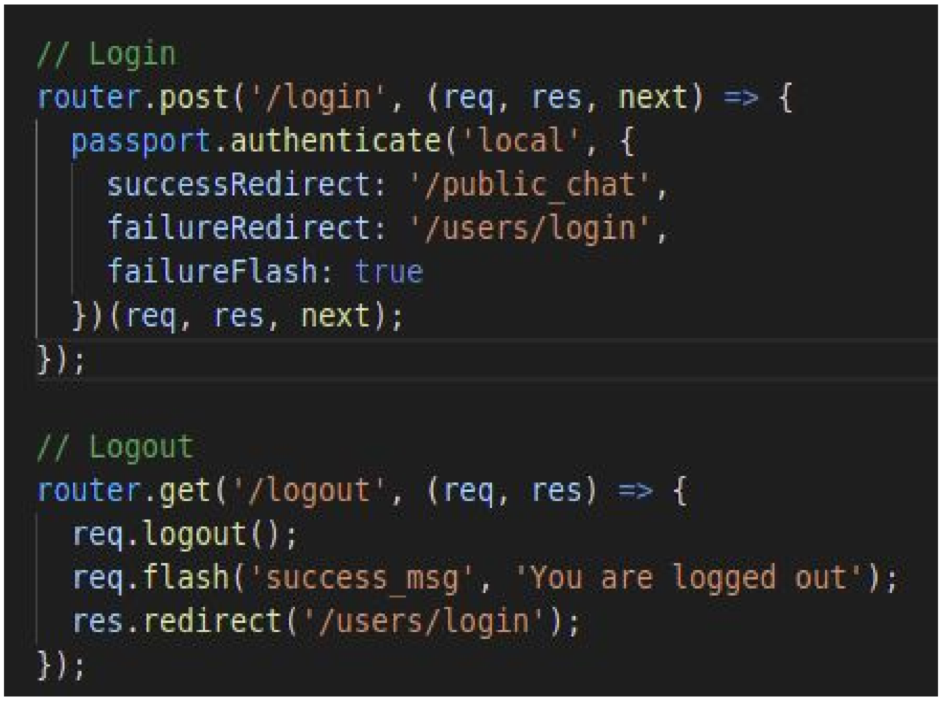


```
var express= require('express');
var router=express.Router();
const { ensureAuthenticated, forwardAuthenticated } = require('../config/auth');
var app=express();
var server=require('http').createServer(app);
var io=require('socket.io').listen(server);
users=[];
connections=[];

router.get('/',ensureAuthenticated,function(req,res){
    res.render('public_chat',{user:req.user});
});

io.sockets.on('connection',function(socket){
    connections.push(socket);
    console.log('connected %s sockets',connections.length);
});

module.exports=router;
```

- **user.js** : It integrates the user model to the login page route, register page route, and also takes care of logout on the public chat page.



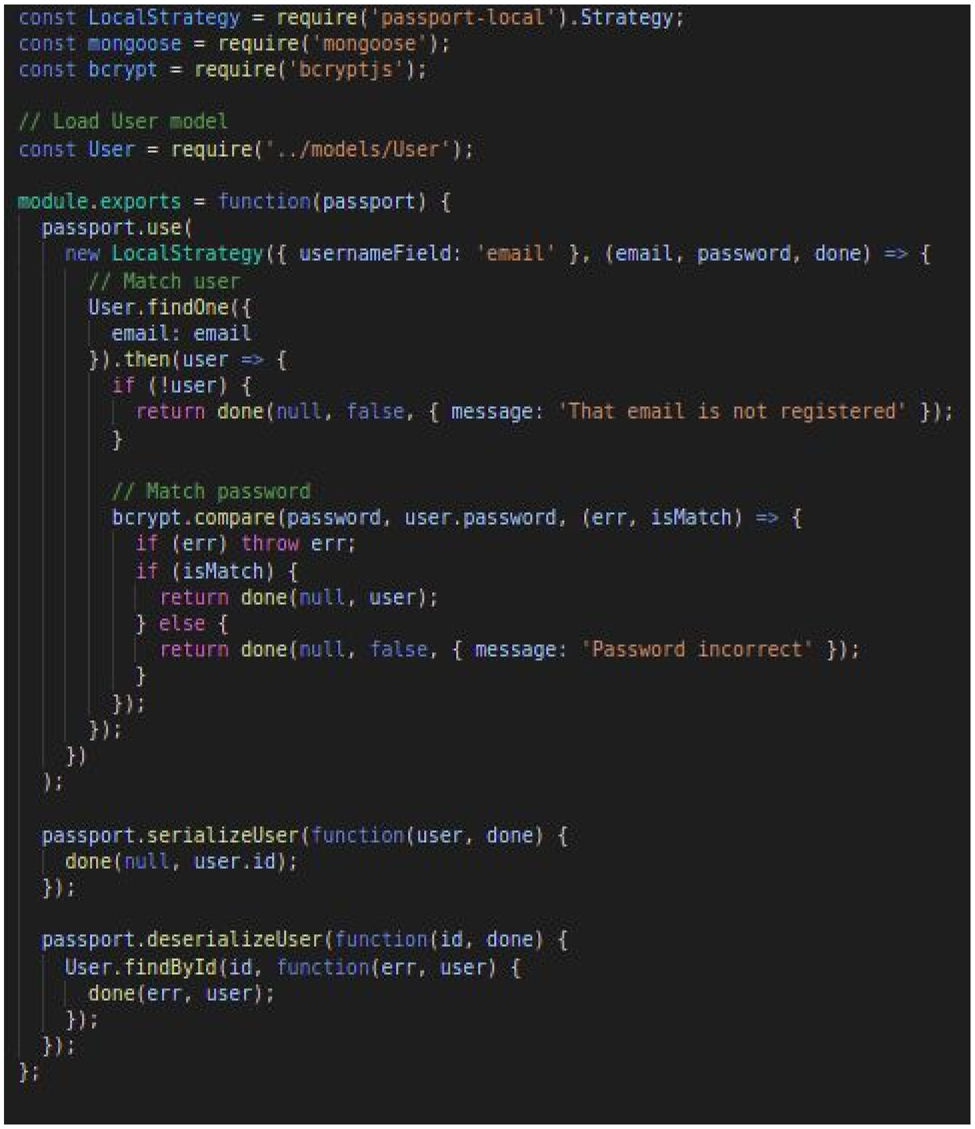
```
const express = require('express');
const router = express.Router();
const bcrypt = require('bcryptjs');
const passport = require('passport');
// Load User model
const User = require('../models/User');
const { forwardAuthenticated } = require('../config/auth');

// Login Page
router.get('/login', forwardAuthenticated, (req, res) => res.render('login'));

// Register Page
router.get('/register', forwardAuthenticated, (req, res) => res.render('register'));

// Register
router.post('/register', (req, res) => {
  const { name, email, password, password2 } = req.body;
  let errors = [];

  var domain = email.substr(email.length-10, 10);

  if(domain!="iitr.ac.in") {
    errors.push({msg : 'You do not have the access. Please try with your Institute Email Id.'})
  }

  if (!name || !email || !password || !password2) {
    errors.push({ msg: 'Please enter all fields.' });
  }

  if (password != password2) {
    errors.push({ msg: 'Passwords do not match.' });
  }
```



```
// Login
router.post('/login', (req, res, next) => {
  passport.authenticate('local', {
    successRedirect: '/public_chat',
    failureRedirect: '/users/login',
    failureFlash: true
  })(req, res, next);
});

// Logout
router.get('/logout', (req, res) => {
  req.logout();
  req.flash('success_msg', 'You are logged out');
  res.redirect('/users/login');
});
```

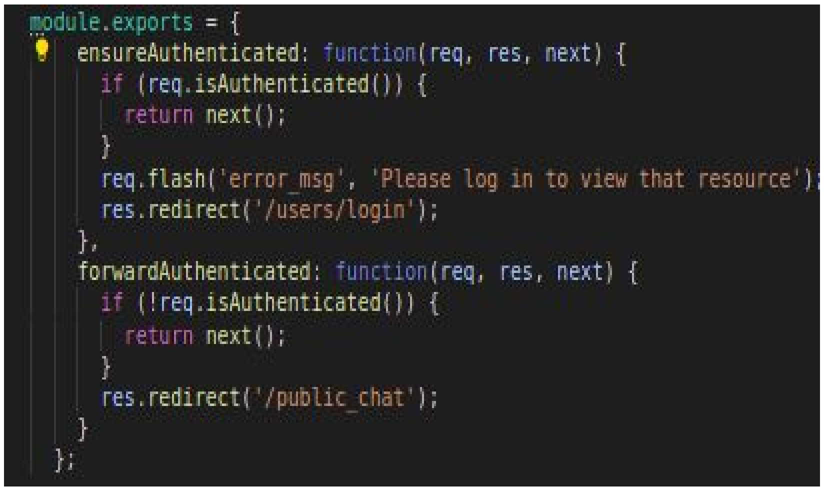
Authentication:

- Passport.js : provides encryption with serializers and deserializer functions and makes login a secure process for the users



```
const LocalStrategy = require('passport-local').Strategy;
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

// Load User model
const User = require('../models/User');

module.exports = function(passport) {
  passport.use(
    new LocalStrategy({ usernameField: 'email' }, (email, password, done) => {
      // Match user
      User.findOne({
        email: email
      }).then(user => {
        if (!user) {
          return done(null, false, { message: 'That email is not registered' });
        }

        // Match password
        bcrypt.compare(password, user.password, (err, isMatch) => {
          if (err) throw err;
          if (isMatch) {
            return done(null, user);
          } else {
            return done(null, false, { message: 'Password incorrect' });
          }
        });
      });
    })
  );

  passport.serializeUser(function(user, done) {
    done(null, user.id);
  });

  passport.deserializeUser(function(id, done) {
    User.findById(id, function(err, user) {
      done(err, user);
    });
  });
};
```

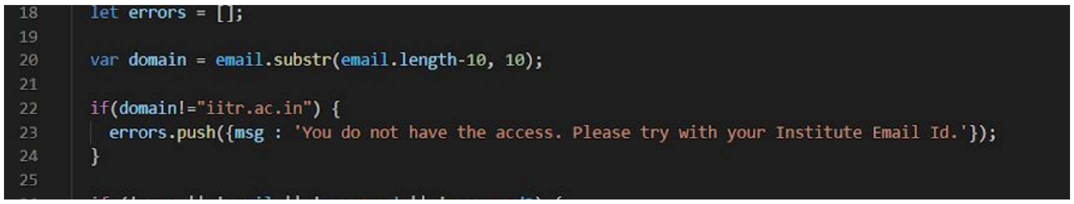
- auth.js: it makes sure only trusted and authenticated emails are allowed else it will show an error message.



```
module.exports = {
  ensureAuthenticated: function(req, res, next) {
    if (req.isAuthenticated()) {
      return next();
    }
    req.flash('error_msg', 'Please log in to view that resource');
    res.redirect('/users/login');
  },
  forwardAuthenticated: function(req, res, next) {
    if (!req.isAuthenticated()) {
      return next();
    }
    res.redirect('/public_chat');
  }
};
```

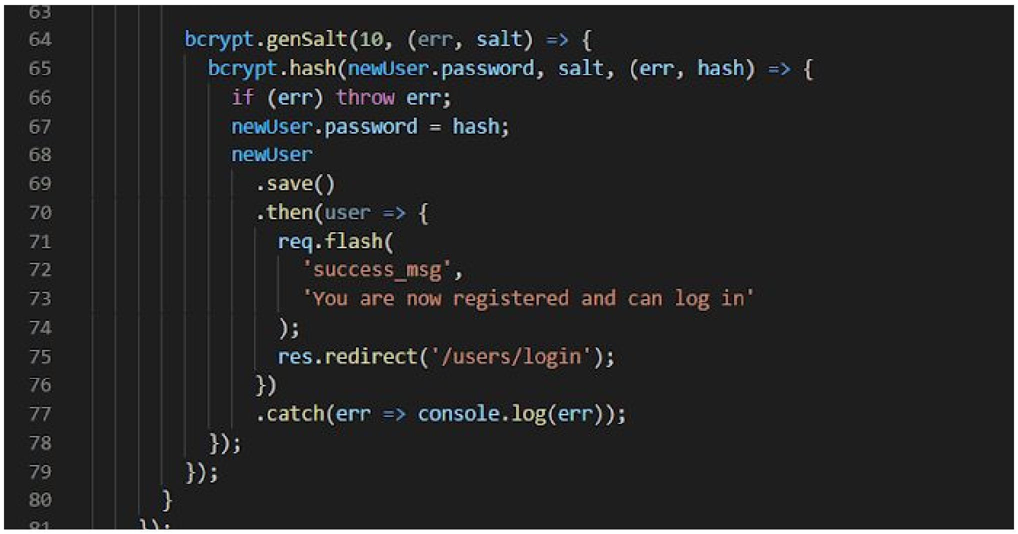
- users.js : used in registering the new user and validation of his details (checking of password length, checking the domain of email, comparison of confirm password and password, checking if user already exists etc.)

  Validation of domain-



```
let errors = [];

var domain = email.substr(email.length-10, 10);

if(domain!="iitr.ac.in") {
  errors.push({msg : 'You do not have the access. Please try with your Institute Email Id.'});
}

```

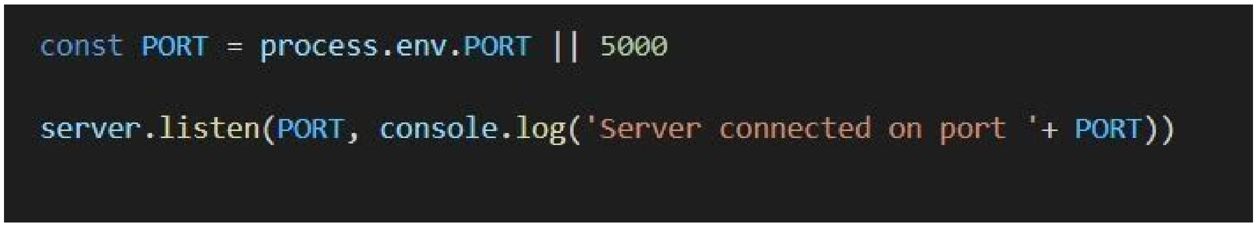
  Hashing of password-



```
          bcrypt.genSalt(10, (err, salt) => {
            bcrypt.hash(newUser.password, salt, (err, hash) => {
              if (err) throw err;
              newUser.password = hash;
              newUser
                .save()
                .then(user => {
                  req.flash(
                    'success_msg',
                    'You are now registered and can log in'
                  );
                  res.redirect('/users/login');
                })
                .catch(err => console.log(err));
            });
          });
        }
```

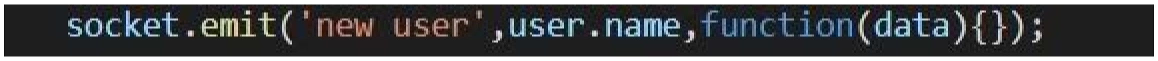
Socket Programming

1. Set the environment variable PORT to tell the web server what port to listen on.



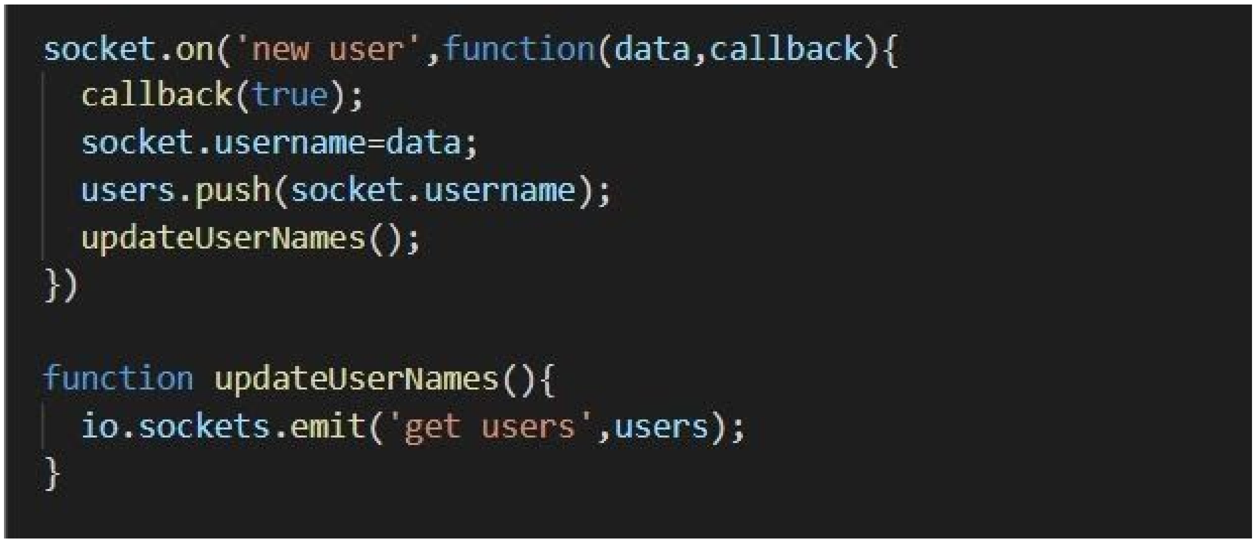
```
const PORT = process.env.PORT || 5000

server.listen(PORT, console.log('Server connected on port '+ PORT))
```

2. Created an event to send the username to the server.



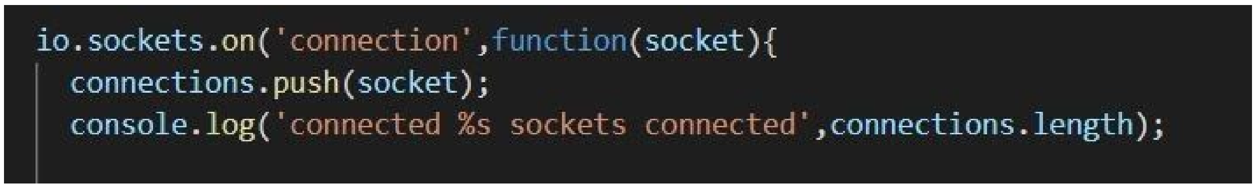
```
  socket.emit('new user',user.name,function(data){});
```

3. Created the server to accept the new username sent by client and update all users.



```
socket.on('new user',function(data,callback){
  callback(true);
  socket.username=data;
  users.push(socket.username);
  updateUserNames();
})

function updateUserNames(){
  io.sockets.emit('get users',users);
}
```

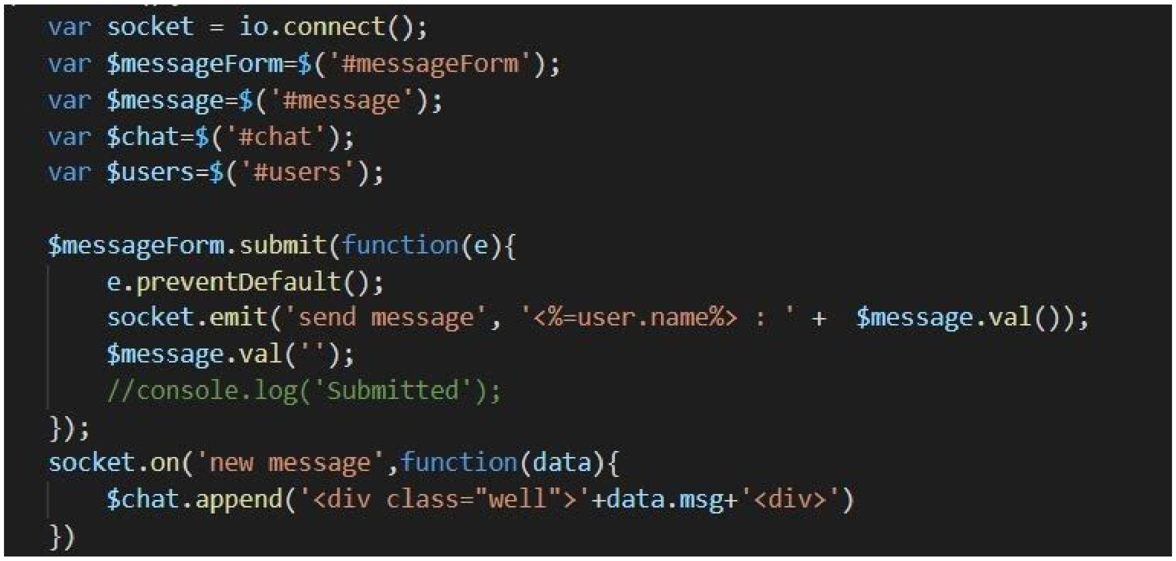
4. Created the server to accept connections from the client.



```
io.sockets.on('connection',function(socket){
  connections.push(socket);
  console.log('connected %s sockets connected',connections.length);
```

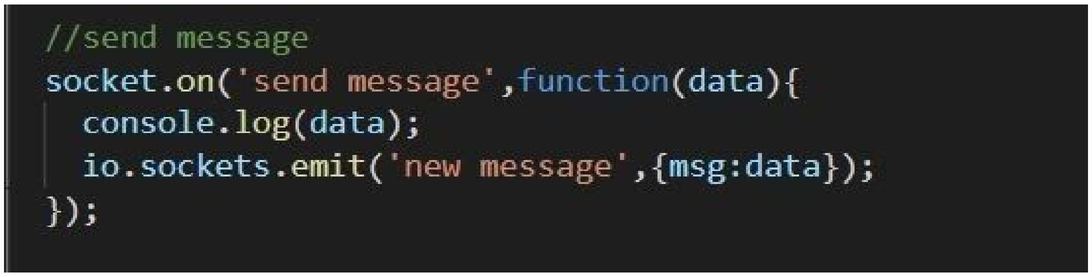
5. Created the link between server and client for messaging.

(i) Client Side for the messaging



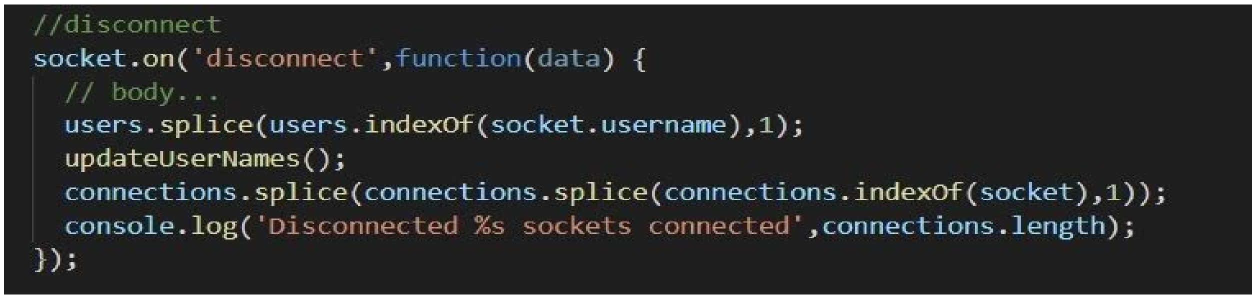
```
var socket = io.connect();
var $messageForm=$('#messageForm');
var $message=$('#message');
var $chat=$('#chat');
var $users=$('#users');

$messageForm.submit(function(e){
    e.preventDefault();
    socket.emit('send message', '<%=user.name%> : ' +  $message.val());
    $message.val('');
    //console.log('Submitted');
});
socket.on('new message',function(data){
    $chat.append('<div class="well">'+data.msg+'<div>')
})
```

(ii) Server Side of the messaging



```
//send message
socket.on('send message',function(data){
  console.log(data);
  io.sockets.emit('new message',{msg:data});
});
```

6. Created an event to disconnect clients from the server.



```
//disconnect
socket.on('disconnect',function(data) {
  // body...
  users.splice(users.indexOf(socket.username),1);
  updateUserNames();
  connections.splice(connections.splice(connections.indexOf(socket),1));
  console.log('Disconnected %s sockets connected',connections.length);
});
```

User Interface Design

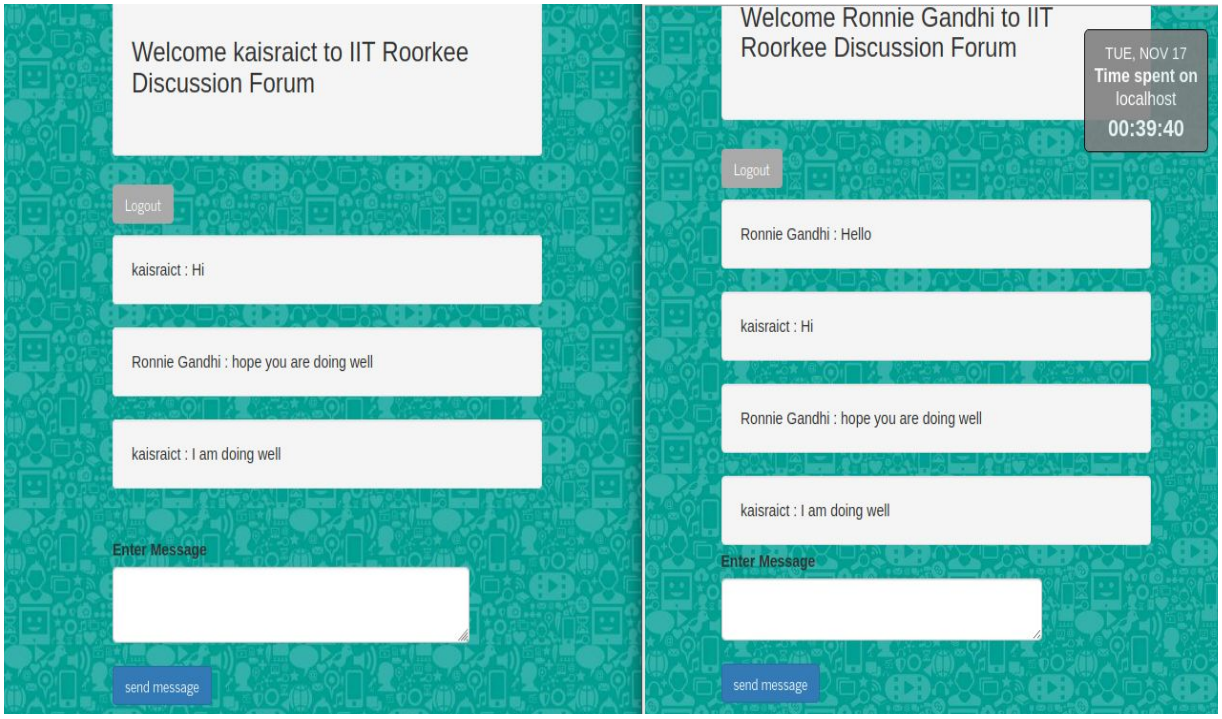
The UI for this app consists of:

1. A welcome page asking if you are a new user or want to login.

2. Then there is a register pager or login page based on what user chooses. In case user registers, he/she is forwarded to login

3. After the user logs in, the public chat page is shown. Here every user
   a. can see messages posted from other users
   b. post messages themselves also.

Design is as follows:



Welcome kaisraict to IIT Roorkee Discussion Forum

kaisraict : Hi

Ronnie Gandhi : hope you are doing well

kaisraict : I am doing well

Welcome Ronnie Gandhi to IIT Roorkee Discussion Forum

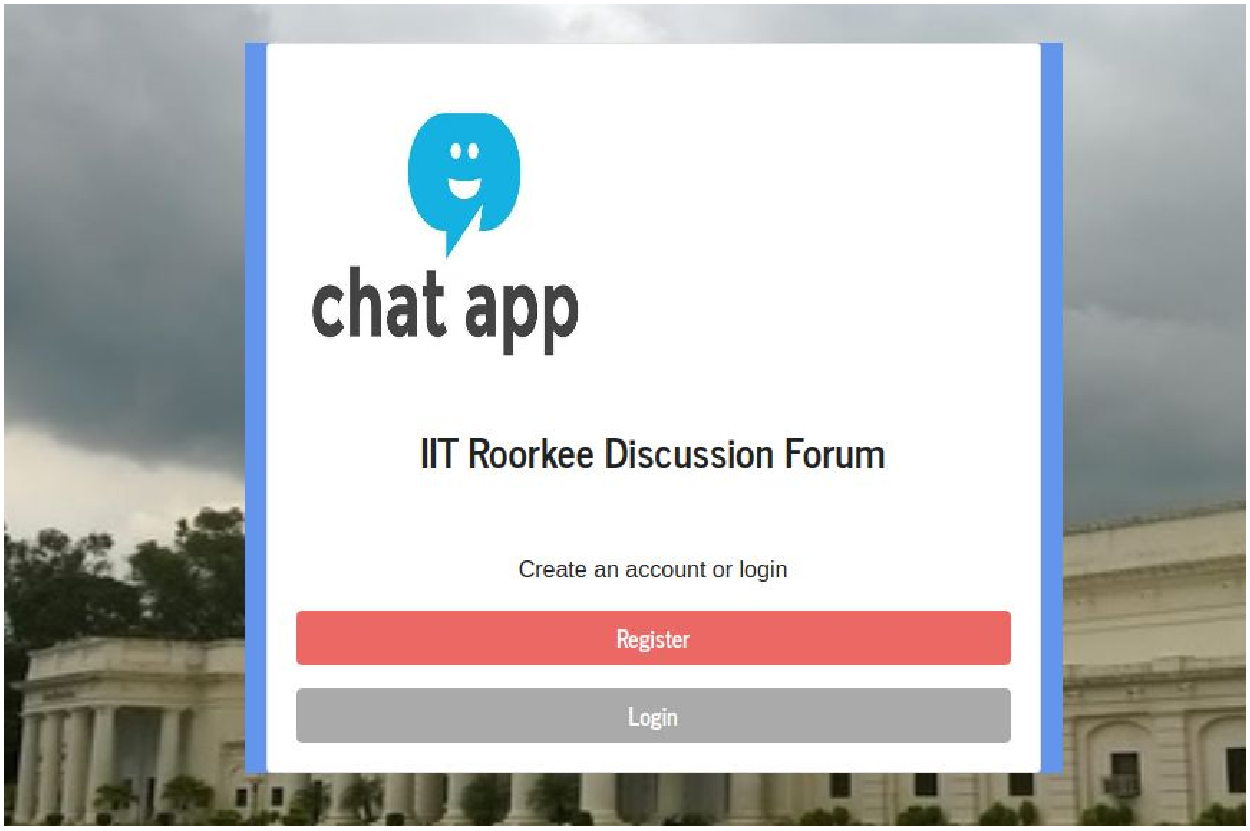
TUE, NOV 17 Time spent on localhost 00:39:40

Ronnie Gandhi : Hello

kaisraict : Hi

Ronnie Gandhi : hope you are doing well

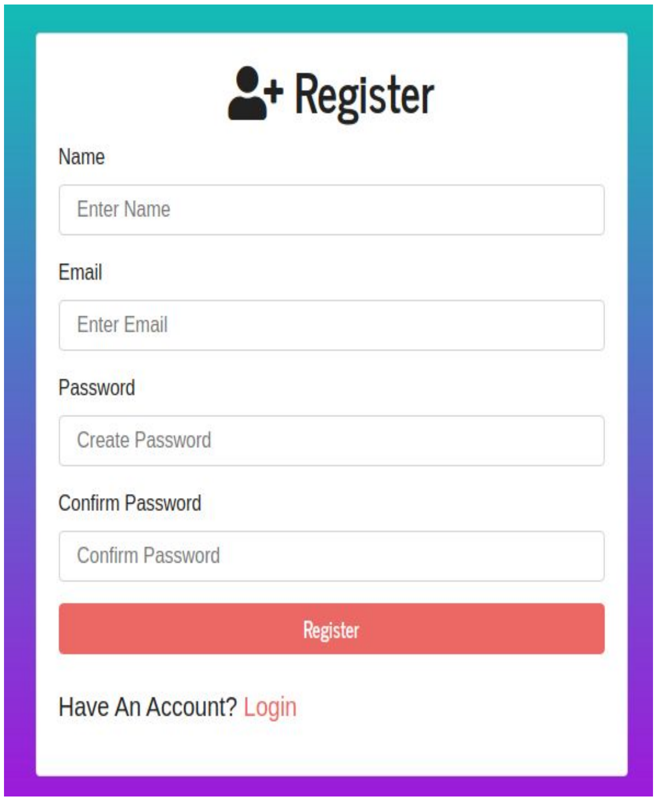
kaisraict : I am doing well



chat app

IIT Roorkee Discussion Forum

Create an account or login



Register

Name

Enter Name

Email

Enter Email

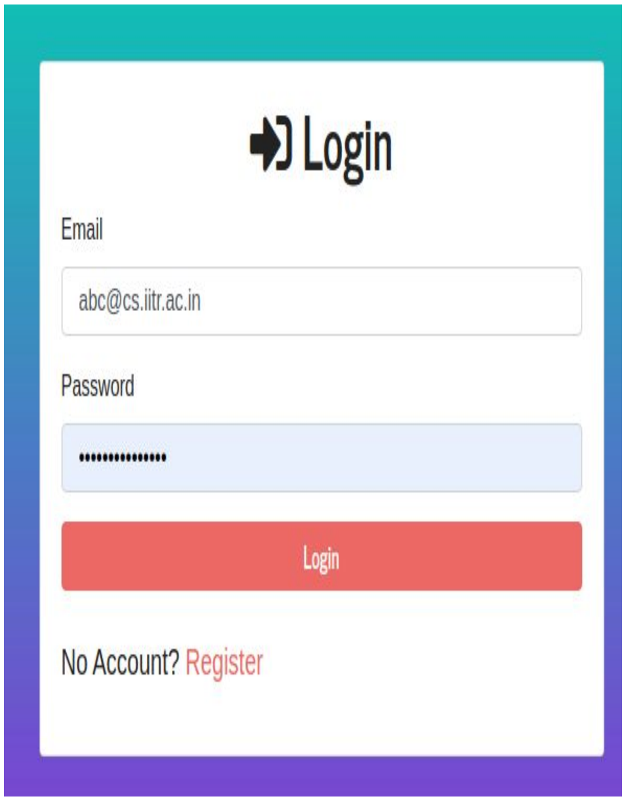
Password

Create Password

Confirm Password

Confirm Password

Have An Account? Login



Login

Email

abc@cs.iitr.ac.in

Password

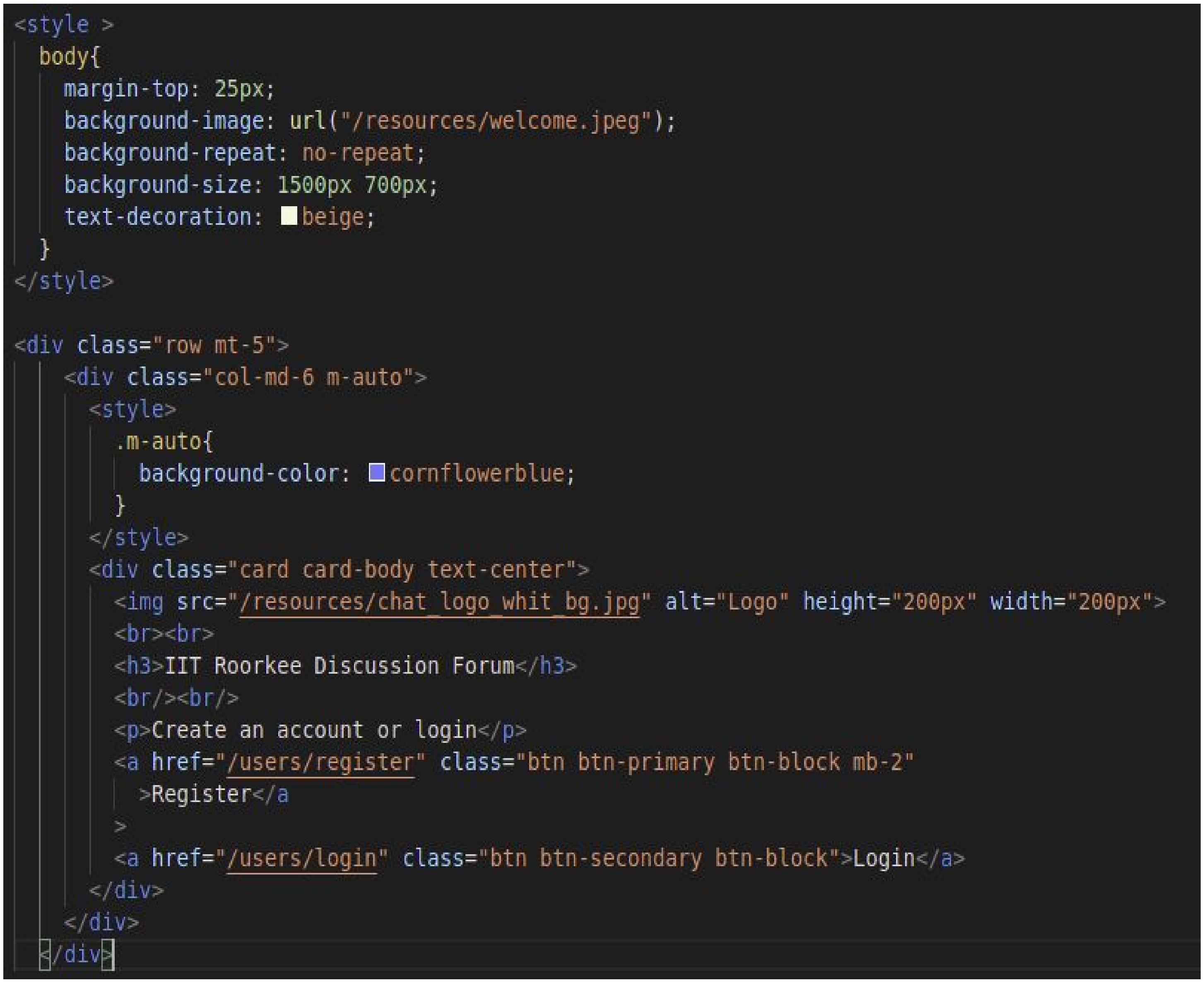
••••••••••••••

No Account? Register

User Interface Code(Frontend) + Some backend Integration

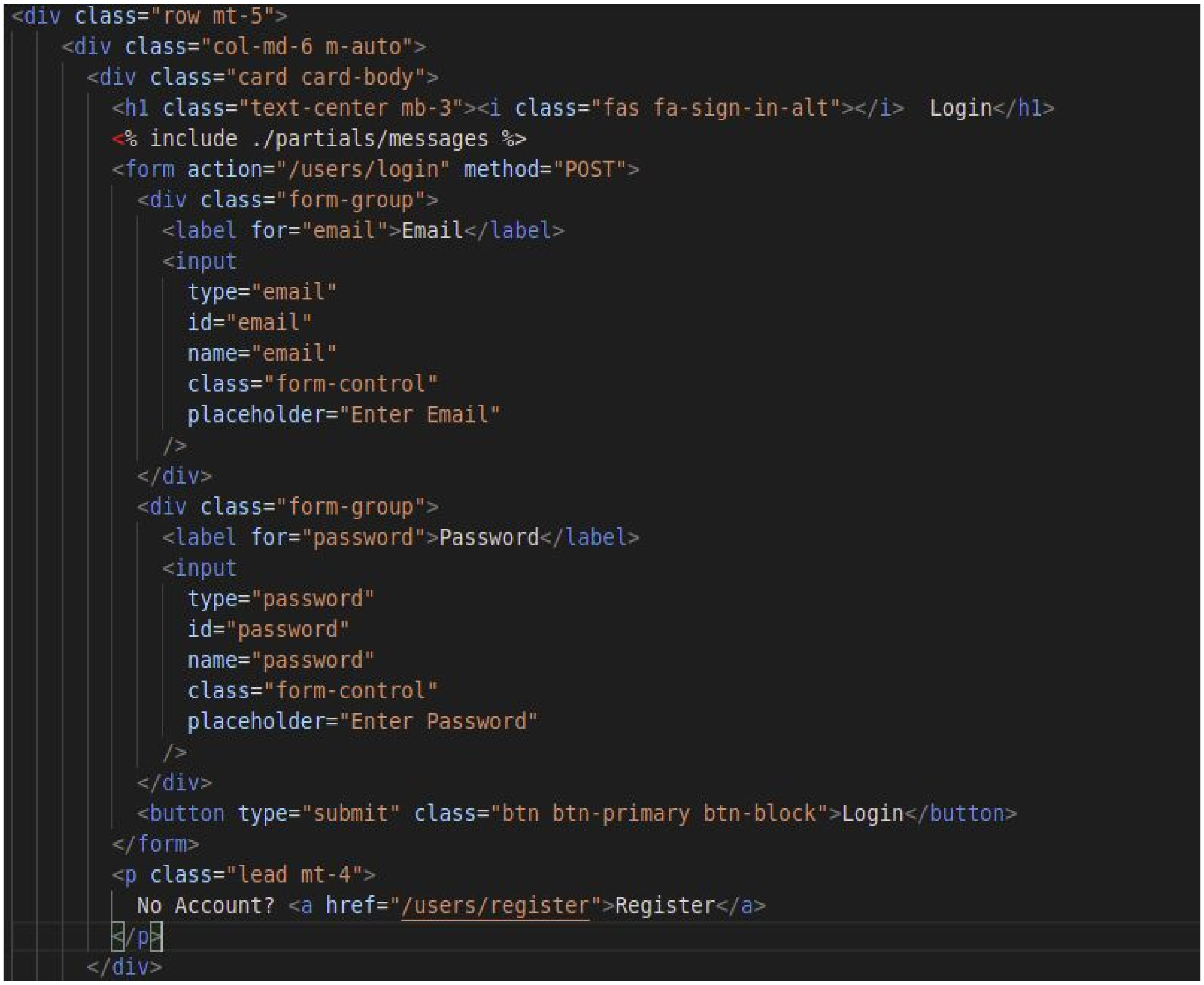
So the frontend contains following items:

- **welcome.ejs**: Created the welcome page with container having chat app logo + title + asking user to register or login as shown above.



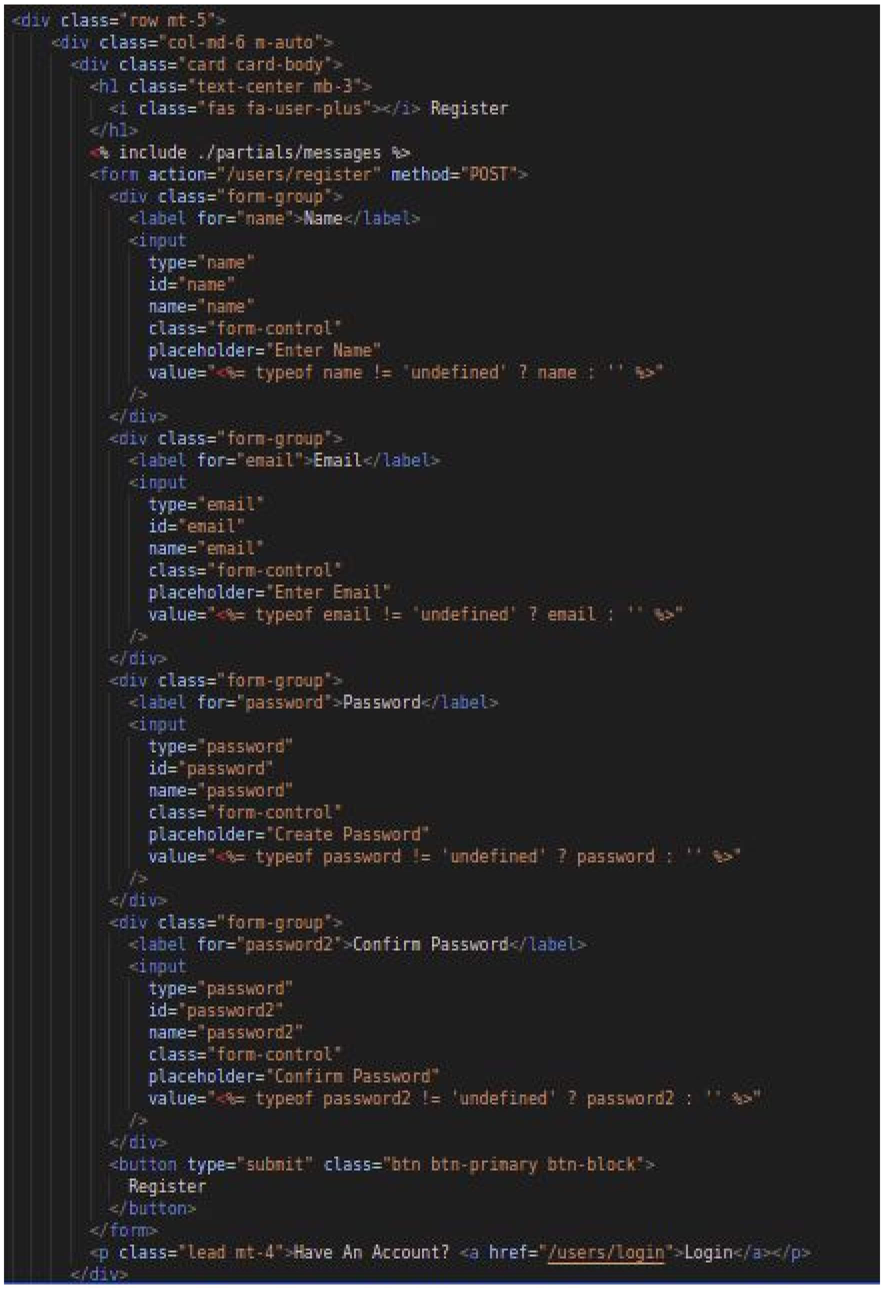
```
<style >
  body{
    margin-top: 25px;
    background-image: url("/resources/welcome.jpeg");
    background-repeat: no-repeat;
    background-size: 1500px 700px;
    text-decoration: beige;
  }
</style>

<div class="row mt-5">
    <div class="col-md-6 m-auto">
      <style>
        .m-auto{
          background-color: cornflowerblue;
        }
      </style>
      <div class="card card-body text-center">
        <img src="/resources/chat_logo_whit_bg.jpg" alt="Logo" height="200px" width="200px">
        <br><br>
        <h3>IIT Roorkee Discussion Forum</h3>
        <br/><br/>
        <p>Create an account or login</p>
        <a href="/users/register" class="btn btn-primary btn-block mb-2"
          >Register</a
        >
        <a href="/users/login" class="btn btn-secondary btn-block">Login</a>
      </div>
    </div>
  </div>
```

- **login.ejs: It asks already registered users to login with a username, i.e. their email and password and sends them to public_chat page after they press the login button. If username is not registered then ask the user for the same and send him to the registration page.**



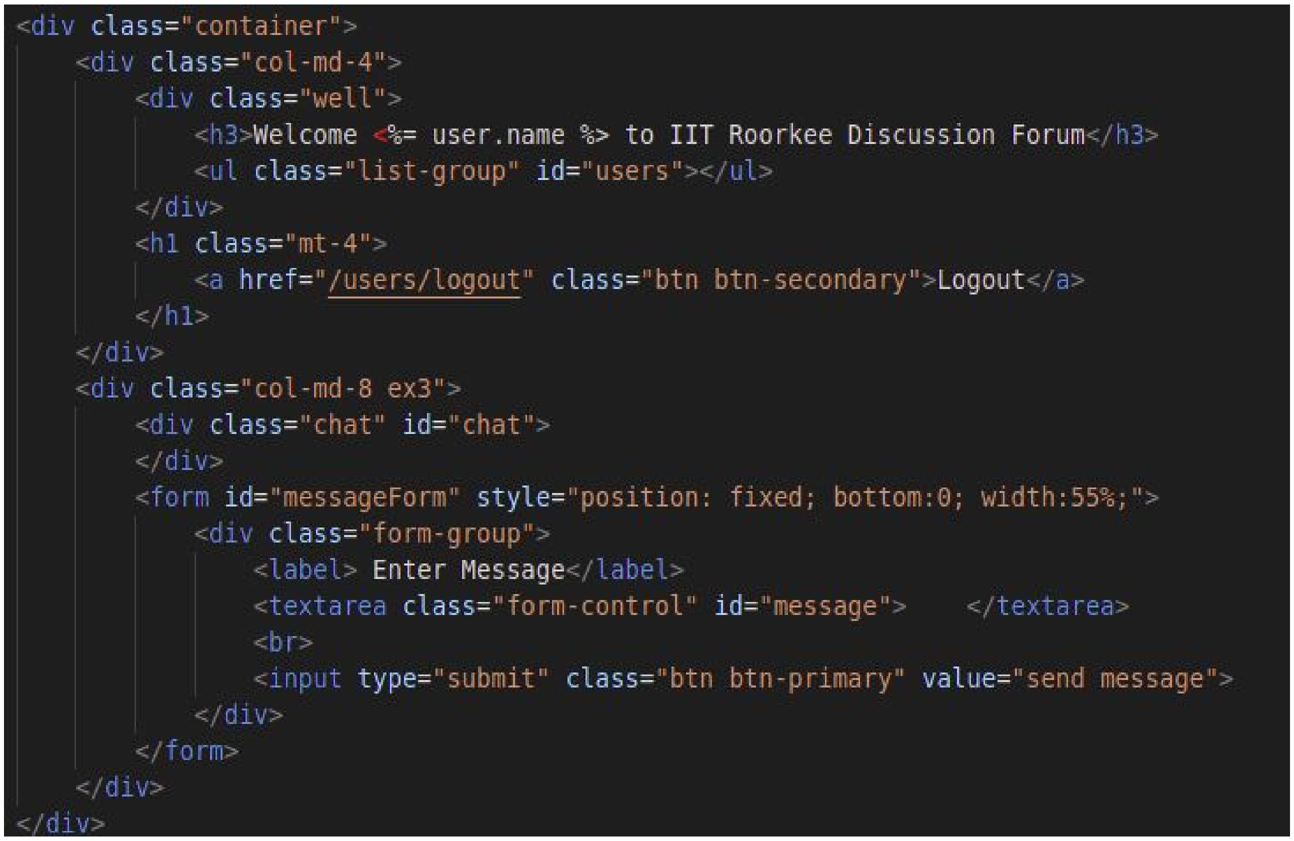
```
<div class="row mt-5">
    <div class="col-md-6 m-auto">
      <div class="card card-body">
        <h1 class="text-center mb-3"><i class="fas fa-sign-in-alt"></i>  Login</h1>
        <% include ./partials/messages %>
        <form action="/users/login" method="POST">
          <div class="form-group">
            <label for="email">Email</label>
            <input
              type="email"
              id="email"
              name="email"
              class="form-control"
              placeholder="Enter Email"
            />
          </div>
          <div class="form-group">
            <label for="password">Password</label>
            <input
              type="password"
              id="password"
              name="password"
              class="form-control"
              placeholder="Enter Password"
            />
          </div>
          <button type="submit" class="btn btn-primary btn-block">Login</button>
        </form>
        <p class="lead mt-4">
          No Account? <a href="/users/register">Register</a>
        </p>
      </div>
```

- **register.ejs: It takes care of registering only authenticated, with iitr email , users and stores their data in the database by connecting with the backend user model. If user already has an account, it asks user for the same and sends user to login page.**



```
<div class="row mt-5">
    <div class="col-md-6 m-auto">
      <div class="card card-body">
        <h1 class="text-center mb-3">
          <i class="fas fa-user-plus"></i> Register
        </h1>
        <% include ./partials/messages %>
        <form action="/users/register" method="POST">
          <div class="form-group">
            <label for="name">Name</label>
            <input
              type="name"
              id="name"
              name="name"
              class="form-control"
              placeholder="Enter Name"
              value="<%= typeof name != 'undefined' ? name : '' %>"
            />
          </div>
          <div class="form-group">
            <label for="email">Email</label>
            <input
              type="email"
              id="email"
              name="email"
              class="form-control"
              placeholder="Enter Email"
              value="<%= typeof email != 'undefined' ? email : '' %>"
            />
          </div>
          <div class="form-group">
            <label for="password">Password</label>
            <input
              type="password"
              id="password"
              name="password"
              class="form-control"
              placeholder="Create Password"
              value="<%= typeof password != 'undefined' ? password : '' %>"
            />
          </div>
          <div class="form-group">
            <label for="password2">Confirm Password</label>
            <input
              type="password"
              id="password2"
              name="password2"
              class="form-control"
              placeholder="Confirm Password"
              value="<%= typeof password2 != 'undefined' ? password2 : '' %>"
            />
          </div>
          <button type="submit" class="btn btn-primary btn-block">
            Register
          </button>
        </form>
        <p class="lead mt-4">Have An Account? <a href="/users/login">Login</a></p>
      </div>
```

- **public_chat.ejs: Creating the chatting area, with a welcome note for the user. It has a scrollable chat messages container and a "Enter Message" textbox with a send button and a logout button. The socket.io code to access the newly received message and send a new message to the client is applied here.**



```
<div class="container">
    <div class="col-md-4">
        <div class="well">
            <h3>Welcome <%= user.name %> to IIT Roorkee Discussion Forum</h3>
            <ul class="list-group" id="users"></ul>
        </div>
        <h1 class="mt-4">
            <a href="/users/logout" class="btn btn-secondary">Logout</a>
        </h1>
    </div>
    <div class="col-md-8 ex3">
        <div class="chat" id="chat">
        </div>
        <form id="messageForm" style="position: fixed; bottom:0; width:55%;">
            <div class="form-group">
                <label> Enter Message</label>
                <textarea class="form-control" id="message">    </textarea>
                <br>
                <input type="submit" class="btn btn-primary" value="send message">
            </div>
        </form>
    </div>
</div>
```