Internet Technology Lab Report

Aneek Roy

BCSE – IV

Roll Number – 001410501072

Jadavpur University

Problem Statement:

Write multi-client chat application consisting of both client and server programs.

In this chat application simultaneously, several clients can communicate with each

other. For this you need a single server program that clients connect to. The client

programs send the chat text (input) to the server and then the server distributes

that chat text to all the other clients. Each client then displays the text sent to it

by the server. The server should be able to handle several clients concurrently. It

should work fine as clients come and go. This can be implemented in two ways

1. The server can handle multiple clients simultaneously by forking a separate

process for each client.

2. The server can create separate threads (instead of processes) to handle

separate clients.

In general, the server program:

* Accepts connection requests from clients
* For each accepted connection starts a process/thread
* Each process/thread reads data from the client and sends it to all other

clients or selected clients

* When a process/thread detects that a client has disconnected it should free

allotted resources and stop processing for that client

Prepare a detailed report of the experiments you have done, and your

observations.

Dependencies :

node.js : **Node.js** is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side. Historically, JavaScript was used primarily for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML, to be run client-side by a JavaScript engine in the user's web browser. Node.js enables JavaScript to be used for server-side scripting, and runs scripts server-side to produce dynamic web page content *before* the page is sent to the user's web browser.

socket.io : Socket.IO is a JavaScript library for realtime web applications. It enables realtime, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for Node.js. Both components have a nearly identical API. Like Node.js, it is event-driven.

Socket.IO primarily uses the WebSocket protocol with polling as a fallback option,[3] while providing the same interface. Although it can be used as simply a wrapper for WebSocket, it provides many more features, including broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O.

express.oi : This node.js library seeks to combine express and socket.io into one cohesive library. This project started as a fork of express.io.

Mongodb :

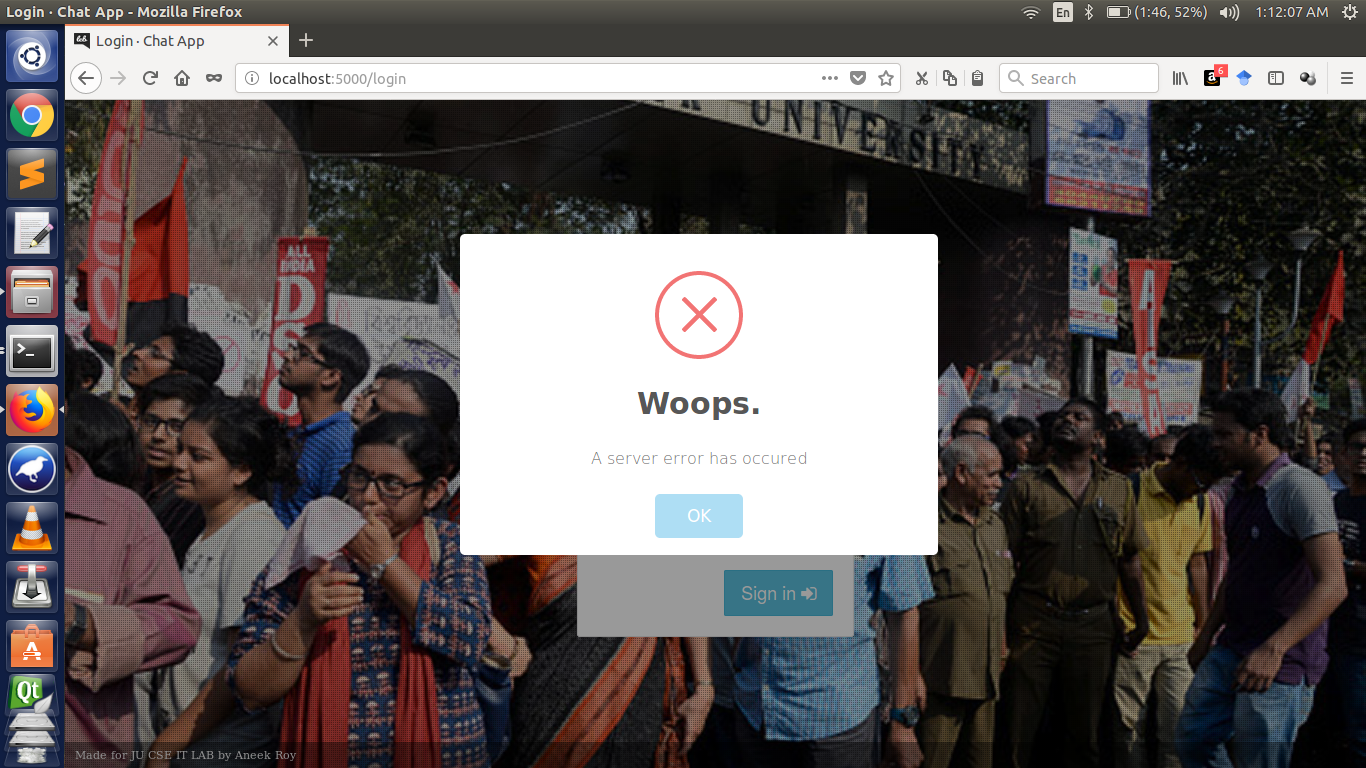
node-xmppserver :

XMPP is an open technology for real-time communication, which powers a wide range of applications including instant messaging, presence, multi-party chat, voice and video calls, collaboration, lightweight middleware, content syndication, and generalized routing of XML data.

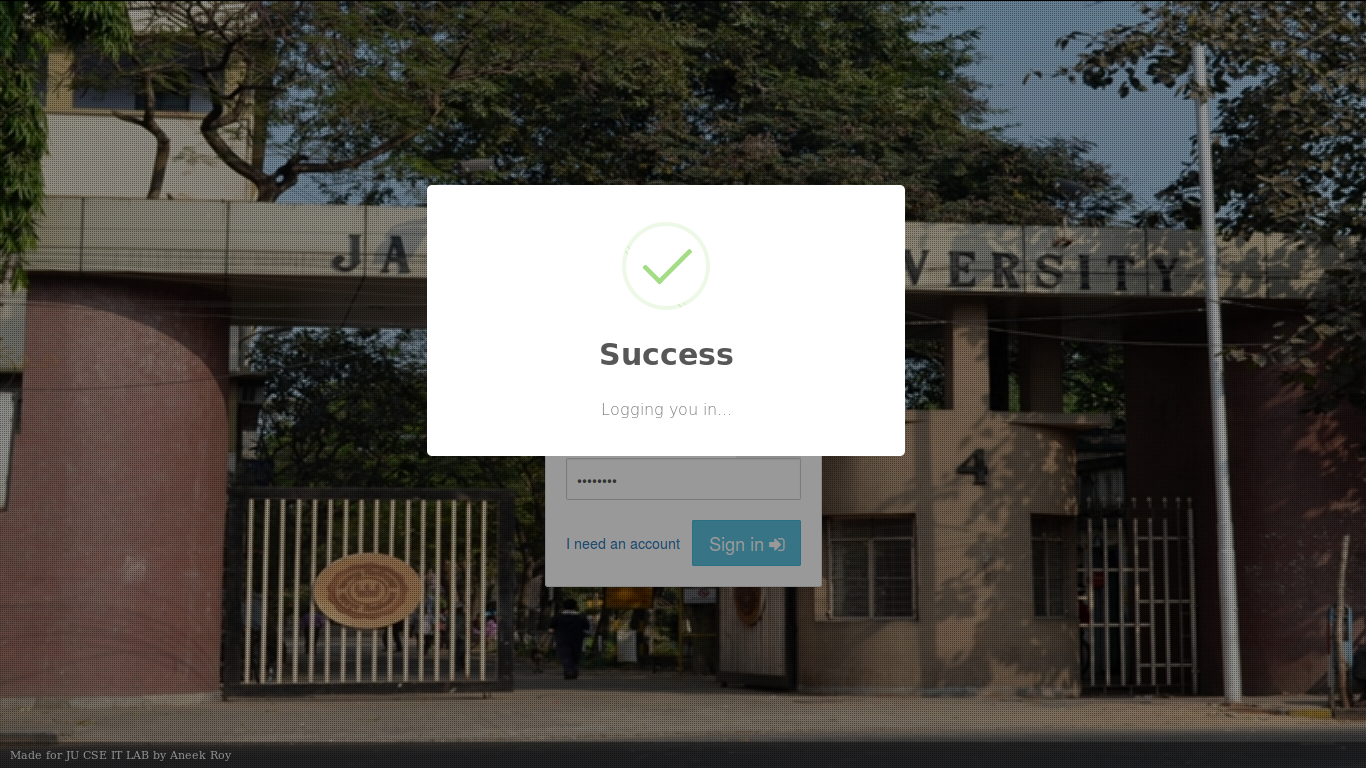
Output Screenshots :



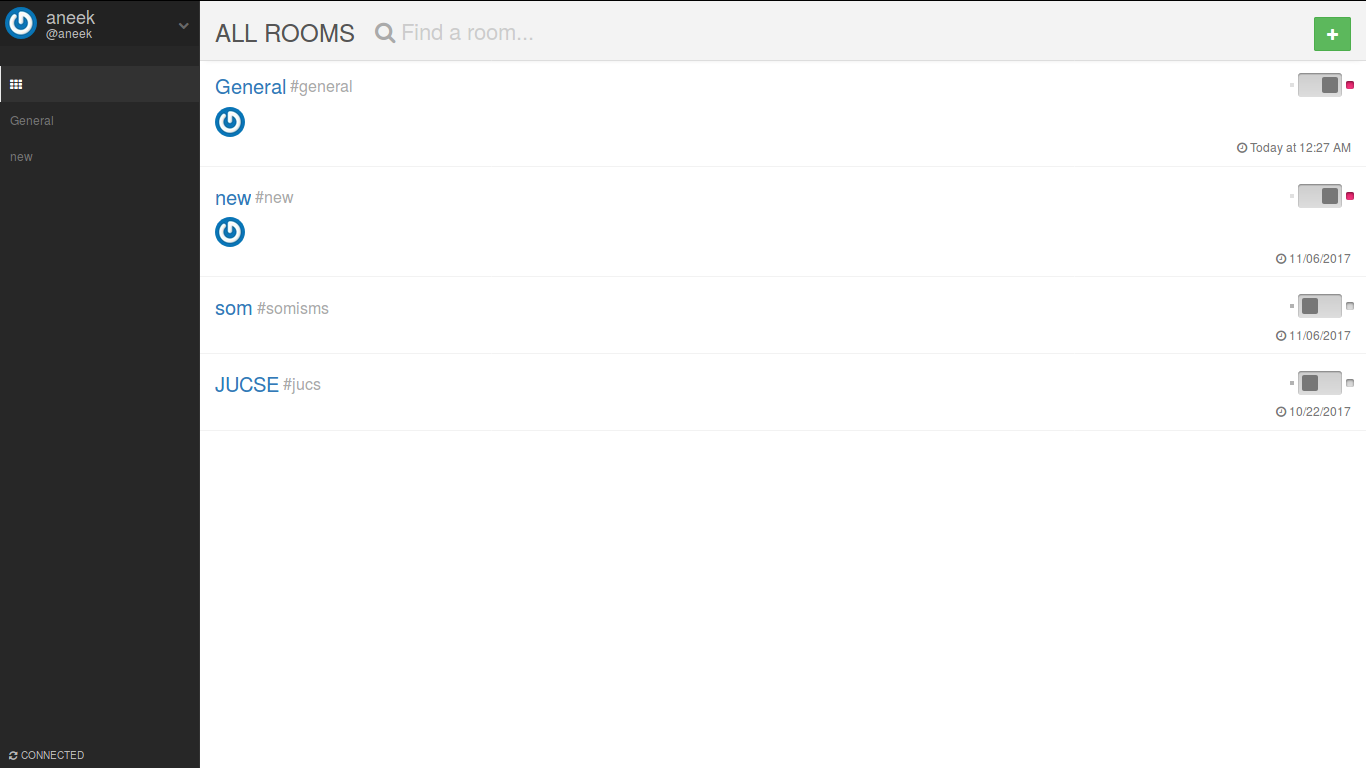
Login Window



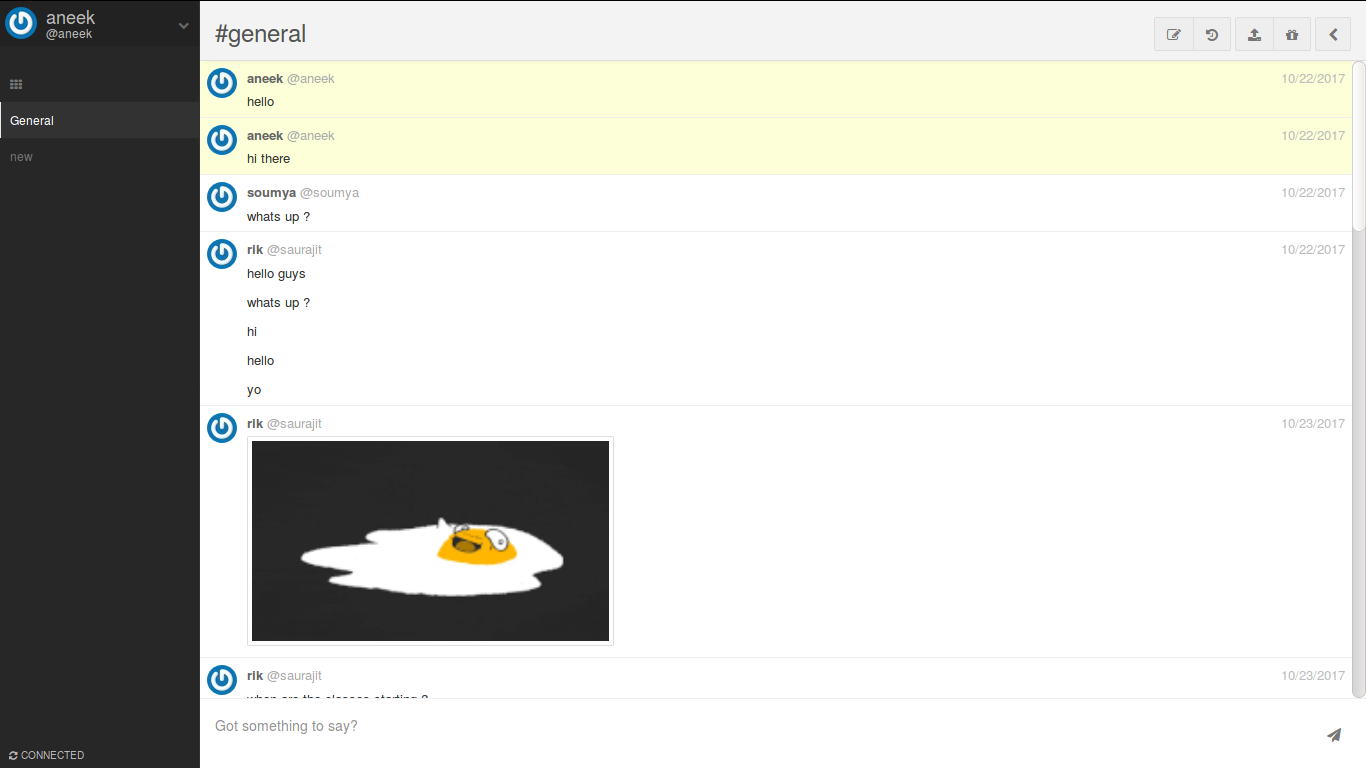
Error in Login due server connection error



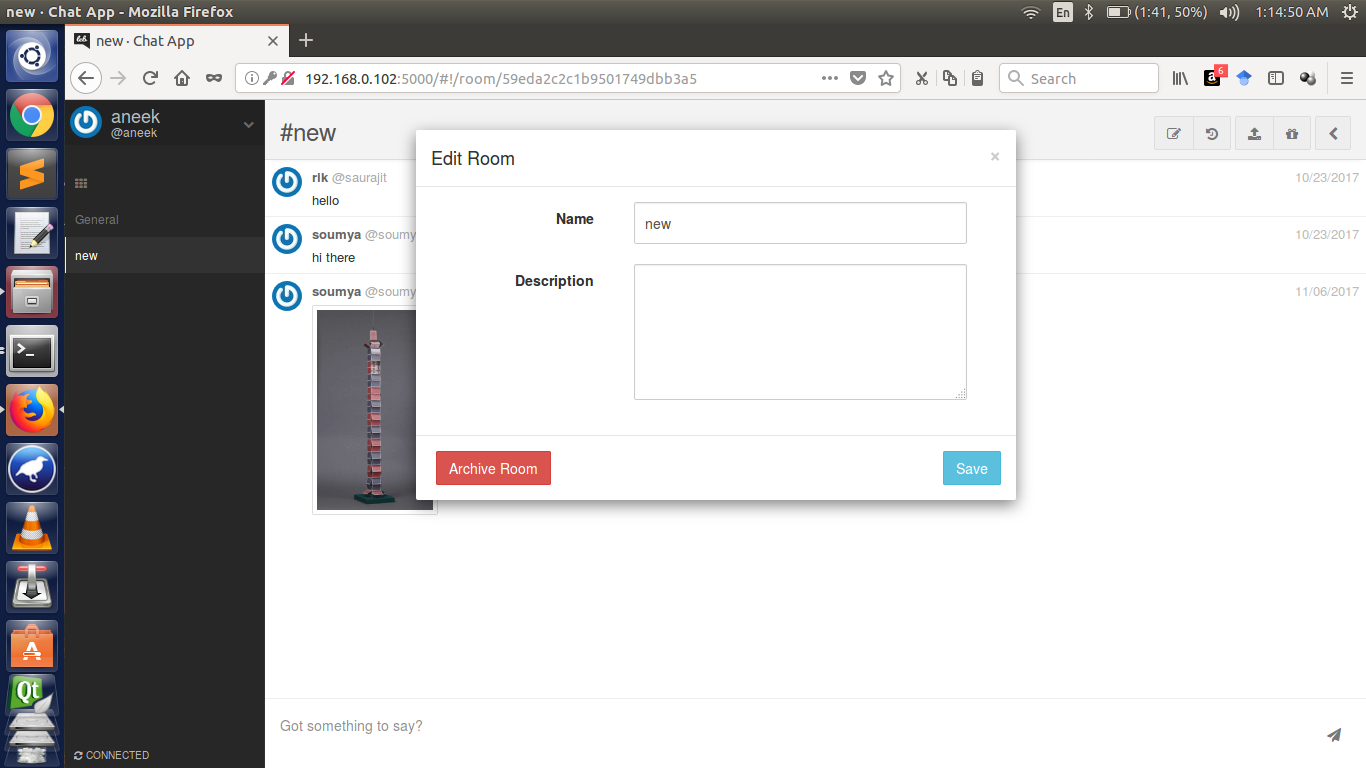
Successful login after authentication of username password through Mongodb database



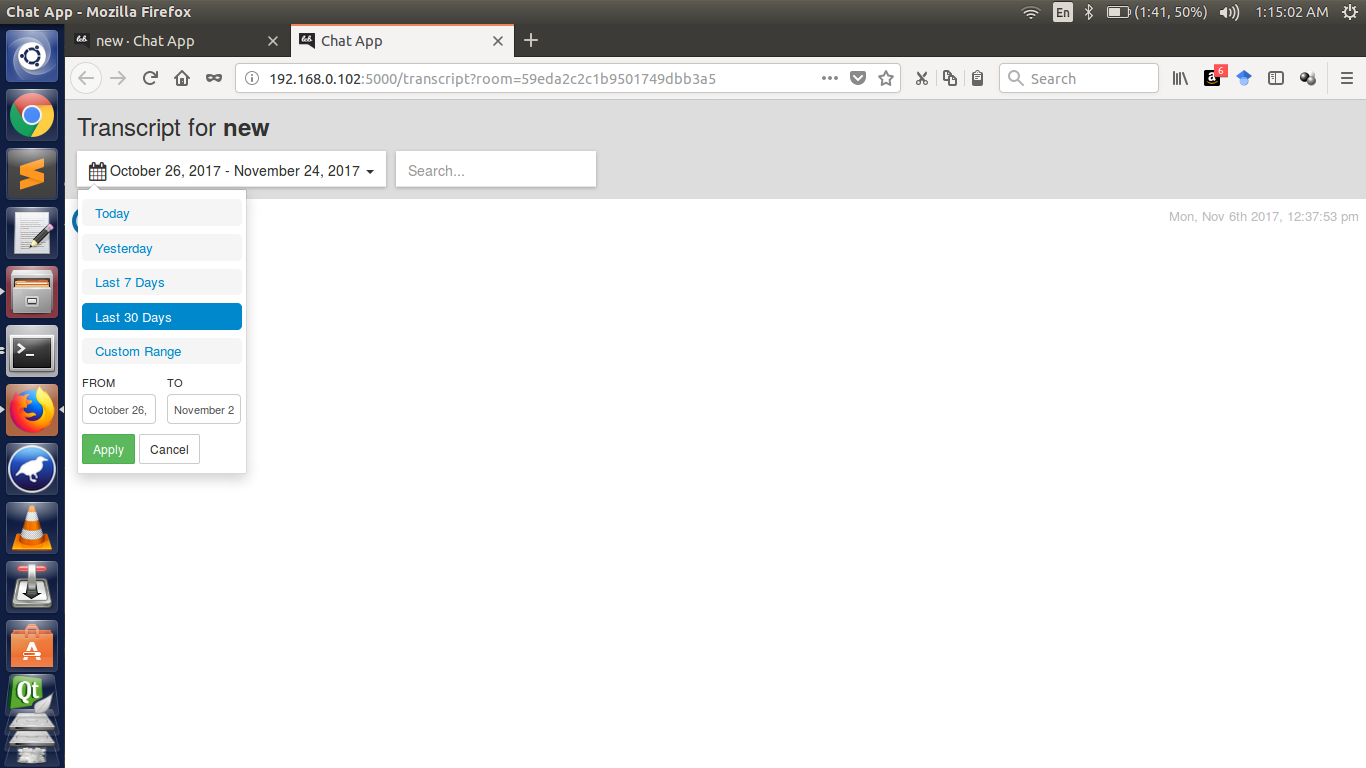
Chat Rooms interface with all the users showing online in respective chatrooms

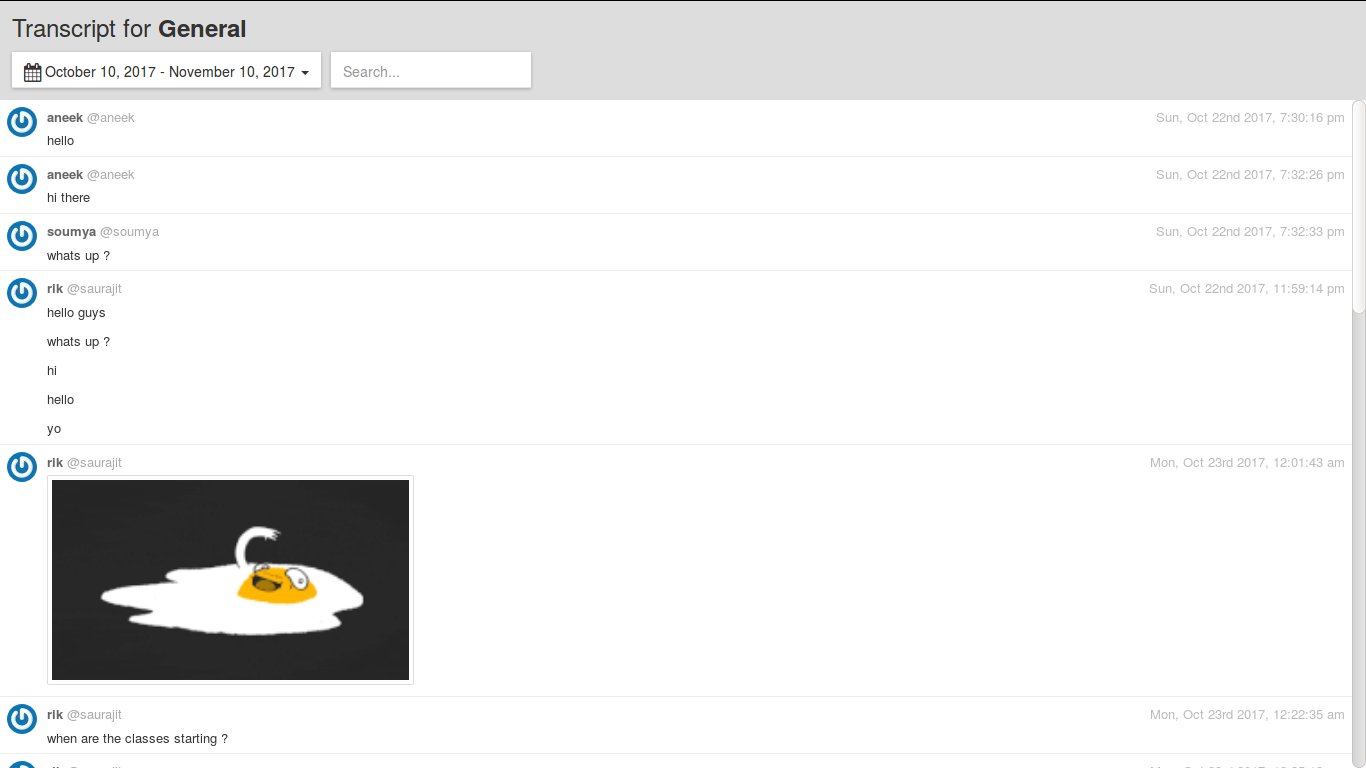


Chat in a given chatroom with time stamps



Interface to add a new chatroom

Chat History Transcripts range query implementation using mongodb



**App.js:**

'use strict';

process.title = 'letschat';

require('colors');

var \_ = require('lodash'),

path = require('path'),

fs = require('fs'),

express = require('express.oi'),

i18n = require('i18n'),

bodyParser = require('body-parser'),

cookieParser = require('cookie-parser'),

compression = require('compression'),

helmet = require('helmet'),

http = require('http'),

nunjucks = require('nunjucks'),

mongoose = require('mongoose'),

connectMongo = require('connect-mongo/es5'),

all = require('require-tree'),

psjon = require('./package.json'),

settings = require('./app/config'),

auth = require('./app/auth/index'),

core = require('./app/core/index');

var MongoStore = connectMongo(express.session),

httpEnabled = settings.http && settings.http.enable,

httpsEnabled = settings.https && settings.https.enable,

models = all(path.resolve('./app/models')),

middlewares = all(path.resolve('./app/middlewares')),

controllers = all(path.resolve('./app/controllers')),

app;

//

// express.oi Setup

//

if (httpsEnabled) {

app = express().https({

key: fs.readFileSync(settings.https.key),

cert: fs.readFileSync(settings.https.cert),

passphrase: settings.https.passphrase

}).io();

} else {

app = express().http().io();

}

if (settings.env === 'production') {

app.set('env', settings.env);

app.set('json spaces', undefined);

app.enable('view cache');

}

// Session

var sessionStore = new MongoStore({

url: settings.database.uri,

autoReconnect: true

});

// Session

var session = {

key: 'connect.sid',

secret: settings.secrets.cookie,

store: sessionStore,

cookie: { secure: httpsEnabled },

resave: false,

saveUninitialized: true

};

// Set compression before any routes

app.use(compression({ threshold: 512 }));

app.use(cookieParser());

app.io.session(session);

auth.setup(app, session, core);

// Security protections

app.use(helmet.frameguard());

app.use(helmet.hidePoweredBy());

app.use(helmet.ieNoOpen());

app.use(helmet.noSniff());

app.use(helmet.xssFilter());

app.use(helmet.hsts({

maxAge: 31536000,

includeSubdomains: true,

force: httpsEnabled,

preload: true

}));

app.use(helmet.contentSecurityPolicy({

defaultSrc: ['\'none\''],

connectSrc: ['\*'],

scriptSrc: ['\'self\'', '\'unsafe-eval\''],

styleSrc: ['\'self\'', 'fonts.googleapis.com', '\'unsafe-inline\''],

fontSrc: ['\'self\'', 'fonts.gstatic.com'],

mediaSrc: ['\'self\''],

objectSrc: ['\'self\''],

imgSrc: ['\* data:']

}));

var bundles = {};

app.use(require('connect-assets')({

paths: [

'media/js',

'media/less'

],

helperContext: bundles,

build: settings.env === 'production',

fingerprinting: settings.env === 'production',

servePath: 'media/dist'

}));

// Public

app.use('/media', express.static(\_\_dirname + '/media', {

maxAge: '364d'

}));

// Templates

var nun = nunjucks.configure('templates', {

autoescape: true,

express: app,

tags: {

blockStart: '<%',

blockEnd: '%>',

variableStart: '<$',

variableEnd: '$>',

commentStart: '<#',

commentEnd: '#>'

}

});

function wrapBundler(func) {

// This method ensures all assets paths start with "./"

// Making them relative, and not absolute

return function() {

return func.apply(func, arguments)

.replace(/href="\//g, 'href="./')

.replace(/src="\//g, 'src="./');

};

}

nun.addFilter('js', wrapBundler(bundles.js));

nun.addFilter('css', wrapBundler(bundles.css));

nun.addGlobal('text\_search', false);

// i18n

i18n.configure({

directory: path.resolve(\_\_dirname, './locales'),

locales: settings.i18n.locales || settings.i18n.locale,

defaultLocale: settings.i18n.locale

});

app.use(i18n.init);

// HTTP Middlewares

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({

extended: true

}));

// IE header

app.use(function(req, res, next) {

res.setHeader('X-UA-Compatible', 'IE=Edge,chrome=1');

next();

});

//

// Controllers

//

\_.each(controllers, function(controller) {

controller.apply({

app: app,

core: core,

settings: settings,

middlewares: middlewares,

models: models,

controllers: controllers

});

});

//

// Mongo

//

mongoose.connection.on('error', function (err) {

throw new Error(err);

});

mongoose.connection.on('disconnected', function() {

throw new Error('Could not connect to database');

});

//

// Go Time

//

function startApp() {

var port = httpsEnabled && settings.https.port ||

httpEnabled && settings.http.port;

var host = httpsEnabled && settings.https.host ||

httpEnabled && settings.http.host || '0.0.0.0';

if (httpsEnabled && httpEnabled) {

// Create an HTTP -> HTTPS redirect server

var redirectServer = express();

redirectServer.get('\*', function(req, res) {

var urlPort = port === 80 ? '' : ':' + port;

res.redirect('https://' + req.hostname + urlPort + req.path);

});

http.createServer(redirectServer)

.listen(settings.http.port || 5000, host);

}

app.listen(port, host);

//

// XMPP

//

if (settings.xmpp.enable) {

var xmpp = require('./app/xmpp/index');

xmpp(core);

}

var art = fs.readFileSync('./app/misc/art.txt', 'utf8');

console.log('\n' + art + '\n\n' + 'Release ' + psjon.version.yellow + '\n');

}

function checkForMongoTextSearch() {

if (!mongoose.mongo || !mongoose.mongo.Admin) {

// MongoDB API has changed, assume text search is enabled

nun.addGlobal('text\_search', true);

return;

}

var admin = new mongoose.mongo.Admin(mongoose.connection.db);

admin.buildInfo(function (err, info) {

if (err || !info) {

return;

}

var version = info.version.split('.');

if (version.length < 2) {

return;

}

if(version[0] < 2) {

return;

}

if(version[0] === '2' && version[1] < 6) {

return;

}

nun.addGlobal('text\_search', true);

});

}

mongoose.connect(settings.database.uri, function(err) {

if (err) {

throw err;

}

checkForMongoTextSearch();

startApp();

});

**settings.yml:**

http:

enable: true

host: 192.168.0.102

port: 5000

https:

enable: false

port: 5001

key: key.pem

cert: certificate.pem

files:

enable: true

provider: local

maxFileSize: 100000000

restrictTypes: true

allowedTypes:

- 'image/jpeg'

- 'image/png'

- 'image/gif'

local:

dir: uploads

xmpp:

enable: false

port: 5222

domain: example.com

database:

uri: mongodb://localhost/chatter

secrets:

cookie: secretsauce

**loginauthorisation/index.js**

'use strict';

var \_ = require('lodash'),

async = require('async'),

cookieParser = require('cookie-parser'),

mongoose = require('mongoose'),

passport = require('passport'),

passportSocketIo = require('passport.socketio'),

BearerStrategy = require('passport-http-bearer'),

BasicStrategy = require('passport-http').BasicStrategy,

settings = require('./../config'),

plugins = require('./../plugins');

var providerSettings = {},

MAX\_AUTH\_DELAY\_TIME = 24 \* 60 \* 60 \* 1000,

loginAttempts = {},

enabledProviders = [];

function getProviders(core) {

return settings.auth.providers.map(function(key) {

var Provider;

if (key === 'local') {

Provider = require('./local');

} else {

Provider = plugins.getPlugin(key, 'auth');

}

return {

key: key,

provider: new Provider(settings.auth[key], core)

};

});

}

function setup(app, session, core) {

enabledProviders = getProviders(core);

enabledProviders.forEach(function(p) {

p.provider.setup();

providerSettings[p.key] = p.provider.options;

});

function tokenAuth(username, password, done) {

if (!done) {

done = password;

}

var User = mongoose.model('User');

User.findByToken(username, function(err, user) {

if (err) { return done(err); }

if (!user) { return done(null, false); }

return done(null, user);

});

}

passport.use(new BearerStrategy(tokenAuth));

passport.use(new BasicStrategy(tokenAuth));

passport.serializeUser(function(user, done) {

done(null, user.\_id);

});

passport.deserializeUser(function(id, done) {

var User = mongoose.model('User');

User.findOne({ \_id: id }, function(err, user) {

done(err, user);

});

});

app.use(passport.initialize());

app.use(passport.session());

session = \_.extend(session, {

cookieParser: cookieParser,

passport: passport

});

var psiAuth = passportSocketIo.authorize(session);

app.io.use(function (socket, next) {

var User = mongoose.model('User');

if (socket.request.\_query && socket.request.\_query.token) {

User.findByToken(socket.request.\_query.token, function(err, user) {

if (err || !user) {

return next('Fail');

}

socket.request.user = user;

socket.request.user.loggedIn = true;

socket.request.user.usingToken = true;

next();

});

} else {

psiAuth(socket, next);

}

});

}

function checkIfAccountLocked(username, cb) {

var attempt = loginAttempts[username];

var isLocked = attempt &&

attempt.lockedUntil &&

attempt.lockedUntil > Date.now();

cb(isLocked);

}

function wrapAuthCallback(username, cb) {

return function(err, user, info) {

if (!err && !user) {

if(!loginAttempts[username]) {

loginAttempts[username] = {

attempts: 0,

lockedUntil: null

};

}

var attempt = loginAttempts[username];

attempt.attempts++;

if (attempt.attempts >= settings.auth.throttling.threshold) {

var lock = Math.min(5000 \* Math.pow(2, (attempt.attempts - settings.auth.throttling.threshold), MAX\_AUTH\_DELAY\_TIME));

attempt.lockedUntil = Date.now() + lock;

return cb(err, user, {

locked: true,

message: 'Account is locked.'

});

}

return cb(err, user, info);

} else {

if(loginAttempts[username]) {

delete loginAttempts[username];

}

cb(err, user, info);

}

};

}

function authenticate() {

var req, username, cb;

if (arguments.length === 4) {

username = arguments[1];

} else if (arguments.length === 3) {

username = arguments[0];

} else {

username = arguments[0].body.username;

}

username = username.toLowerCase();

if (arguments.length === 4) {

req = \_.extend({}, arguments[0], {

body: {

username: username,

password: arguments[2]

}

});

cb = arguments[3];

} else if (arguments.length === 3) {

req = {

body: {

username: username,

password: arguments[1]

}

};

cb = arguments[2];

} else {

req = \_.extend({}, arguments[0]);

req.body.username = username;

cb = arguments[1];

}

checkIfAccountLocked(username, function(locked) {

if (locked) {

return cb(null, null, {

locked: true,

message: 'Account is locked.'

});

}

if (settings.auth.throttling &&

settings.auth.throttling.enable) {

cb = wrapAuthCallback(username, cb);

}

var series = enabledProviders.map(function(p) {

var provider = p.provider;

return function() {

var args = Array.prototype.slice.call(arguments);

var callback = args.slice(args.length - 1)[0];

if (args.length > 1 && args[0]) {

return callback(null, args[0]);

}

provider.authenticate(req, function(err, user) {

if (err) {

return callback(err);

}

return callback(null, user);

});

};

});

async.waterfall(series, function(err, user) {

cb(err, user);

});

});

}

module.exports = {

setup: setup,

authenticate: authenticate,

providers: providerSettings

};

**chatrooms.js**

'use strict';

var mongoose = require('mongoose'),

\_ = require('lodash'),

helpers = require('./helpers');

var getParticipants = function(room, options, cb) {

if (!room.private || !options.participants) {

return cb(null, []);

}

var participants = [];

if (Array.isArray(options.participants)) {

participants = options.participants;

}

if (typeof options.participants === 'string') {

participants = options.participants.replace(/@/g, '')

.split(',').map(function(username) {

return username.trim();

});

}

participants = \_.chain(participants)

.map(function(username) {

return username && username.replace(/@,\s/g, '').trim();

})

.filter(function(username) { return !!username; })

.uniq()

.value();

var User = mongoose.model('User');

User.find({username: { $in: participants } }, cb);

};

function RoomManager(options) {

this.core = options.core;

}

RoomManager.prototype.canJoin = function(options, cb) {

var method = options.id ? 'get' : 'slug',

roomId = options.id ? options.id : options.slug;

this[method](roomId, function(err, room) {

if (err) {

return cb(err);

}

if (!room) {

return cb();

}

room.canJoin(options, function(err, canJoin) {

cb(err, room, canJoin);

});

});

};

RoomManager.prototype.create = function(options, cb) {

var Room = mongoose.model('Room');

Room.create(options, function(err, room) {

if (err) {

return cb(err);

}

if (cb) {

room = room;

cb(null, room);

this.core.emit('rooms:new', room);

}

}.bind(this));

};

RoomManager.prototype.update = function(roomId, options, cb) {

var Room = mongoose.model('Room');

Room.findById(roomId, function(err, room) {

if (err) {

// Oh noes, a bad thing happened!

console.error(err);

return cb(err);

}

if (!room) {

return cb('Room does not exist.');

}

if(room.private && !room.owner.equals(options.user.id)) {

return cb('Only owner can change private room.');

}

getParticipants(room, options, function(err, participants) {

if (err) {

// Oh noes, a bad thing happened!

console.error(err);

return cb(err);

}

room.name = options.name;

// DO NOT UPDATE SLUG

// room.slug = options.slug;

room.description = options.description;

if (room.private) {

room.password = options.password;

room.participants = participants;

}

room.save(function(err, room) {

if (err) {

console.error(err);

return cb(err);

}

room = room;

cb(null, room);

this.core.emit('rooms:update', room);

}.bind(this));

}.bind(this));

}.bind(this));

};

RoomManager.prototype.archive = function(roomId, cb) {

var Room = mongoose.model('Room');

Room.findById(roomId, function(err, room) {

if (err) {

// Oh noes, a bad thing happened!

console.error(err);

return cb(err);

}

if (!room) {

return cb('Room does not exist.');

}

room.archived = true;

room.save(function(err, room) {

if (err) {

console.error(err);

return cb(err);

}

cb(null, room);

this.core.emit('rooms:archive', room);

}.bind(this));

}.bind(this));

};

RoomManager.prototype.list = function(options, cb) {

options = options || {};

options = helpers.sanitizeQuery(options, {

defaults: {

take: 500

},

maxTake: 5000

});

var Room = mongoose.model('Room');

var find = Room.find({

archived: { $ne: true },

$or: [

{private: {$exists: false}},

{private: false},

{owner: options.userId},

{participants: options.userId},

{password: {$exists: true, $ne: ''}}

]

});

if (options.skip) {

find.skip(options.skip);

}

if (options.take) {

find.limit(options.take);

}

if (options.sort) {

var sort = options.sort.replace(',', ' ');

find.sort(sort);

} else {

find.sort('-lastActive');

}

find.populate('participants');

find.exec(function(err, rooms) {

if (err) {

return cb(err);

}

\_.each(rooms, function(room) {

this.sanitizeRoom(options, room);

}.bind(this));

if (options.users && !options.sort) {

rooms = \_.sortBy(rooms, ['userCount', 'lastActive'])

.reverse();

}

cb(null, rooms);

}.bind(this));

};

RoomManager.prototype.sanitizeRoom = function(options, room) {

var authorized = options.userId && room.isAuthorized(options.userId);

if (options.users) {

if (authorized) {

room.users = this.core.presence

.getUsersForRoom(room.id.toString());

} else {

room.users = [];

}

}

};

RoomManager.prototype.findOne = function(options, cb) {

var Room = mongoose.model('Room');

Room.findOne(options.criteria)

.populate('participants').exec(function(err, room) {

if (err) {

return cb(err);

}

this.sanitizeRoom(options, room);

cb(err, room);

}.bind(this));

};

RoomManager.prototype.get = function(options, cb) {

var identifier;

if (typeof options === 'string') {

identifier = options;

options = {};

options.identifier = identifier;

} else {

identifier = options.identifier;

}

options.criteria = {

\_id: identifier,

archived: { $ne: true }

};

this.findOne(options, cb);

};

RoomManager.prototype.slug = function(options, cb) {

var identifier;

if (typeof options === 'string') {

identifier = options;

options = {};

options.identifier = identifier;

} else {

identifier = options.identifier;

}

options.criteria = {

slug: identifier,

archived: { $ne: true }

};

this.findOne(options, cb);

};

module.exports = RoomManager;

**messages.js**

var \_ = require('lodash'),

mongoose = require('mongoose'),

helpers = require('./helpers');

function MessageManager(options) {

this.core = options.core;

}

MessageManager.prototype.create = function(options, cb) {

var Message = mongoose.model('Message'),

Room = mongoose.model('Room'),

User = mongoose.model('User');

if (typeof cb !== 'function') {

cb = function() {};

}

Room.findById(options.room, function(err, room) {

if (err) {

console.error(err);

return cb(err);

}

if (!room) {

return cb('Room does not exist.');

}

if (room.archived) {

return cb('Room is archived.');

}

if (!room.isAuthorized(options.owner)) {

return cb('Not authorized.');

}

Message.create(options, function(err, message) {

if (err) {

console.error(err);

return cb(err);

}

// Touch Room's lastActive

room.lastActive = message.posted;

room.save();

// Temporary workaround for \_id until populate can do aliasing

User.findOne(message.owner, function(err, user) {

if (err) {

console.error(err);

return cb(err);

}

cb(null, message, room, user);

this.core.emit('messages:new', message, room, user, options.data);

}.bind(this));

}.bind(this));

}.bind(this));

};

MessageManager.prototype.list = function(options, cb) {

var Room = mongoose.model('Room');

options = options || {};

if (!options.room) {

return cb(null, []);

}

options = helpers.sanitizeQuery(options, {

defaults: {

reverse: true,

take: 500

},

maxTake: 5000

});

var Message = mongoose.model('Message');

var find = Message.find({

room: options.room

});

if (options.since\_id) {

find.where('\_id').gt(options.since\_id);

}

if (options.from) {

find.where('posted').gt(options.from);

}

if (options.to) {

find.where('posted').lte(options.to);

}

if (options.query) {

find = find.find({$text: {$search: options.query}});

}

if (options.expand) {

var includes = options.expand.replace(/\s/, '').split(',');

if (\_.includes(includes, 'owner')) {

find.populate('owner', 'id username displayName email avatar');

}

if (\_.includes(includes, 'room')) {

find.populate('room', 'id name');

}

}

if (options.skip) {

find.skip(options.skip);

}

if (options.reverse) {

find.sort({ 'posted': -1 });

} else {

find.sort({ 'posted': 1 });

}

Room.findById(options.room, function(err, room) {

if (err) {

console.error(err);

return cb(err);

}

var opts = {

userId: options.userId,

password: options.password

};

room.canJoin(opts, function(err, canJoin) {

if (err) {

console.error(err);

return cb(err);

}

if (!canJoin) {

return cb(null, []);

}

find.limit(options.take)

.exec(function(err, messages) {

if (err) {

console.error(err);

return cb(err);

}

cb(null, messages);

});

});

});

};

module.exports = MessageManager;

login.html :

<!-- <span class="flavour">'</span>s -->

<% block body %>

<section class="lcb-login-main">

<h1 class="lcb-login-logo"> Chatter</h1>

<div class="lcb-login-boxes">

<form class="lcb-login-box lcb-login-box-login validate" action="./account/login" method="post" data-refresh="true">

<h2 class="lcb-login-box-heading"><$ \_\_('Sign in') $></h2>

<img class="lcb-login-avatar" style="display: none;" />

<div class="form-group">

<input class="required form-control" placeholder="<$ \_\_('Username or Email') $>" name="username" type="text" autofocus />

</div>

<div class="form-group">

<input class="required form-control" placeholder="<$ \_\_('Password') $>" name="password" type="password" autocomplete="off" />

</div>

<div class="lcb-login-box-bottom">

<% if auth.local.enableRegistration %>

<div class="links pull-left">

<a href="#" class="lcb-show-box" data-target=".lcb-login-box-registration">

<$ \_\_('I need an account') $>

</a>

</div>

<% endif %>

<button class="btn-info btn btn-lg pull-right" type="submit">

<$ \_\_('Sign in') $>

<i class="fa fa-sign-in"></i>

</button>

</div>

</form>

<% if auth.local.enableRegistration %>

<form class="lcb-login-box lcb-login-box-registration validate" action="./account/register" method="post" style="display: none;">

<h2 class="lcb-login-box-heading"><$ \_\_('Register') $></h2>

<div class="form-group">

<div class="input-group">

<span class="input-group-addon">@</span>

<input type="text" name="username"

class="required form-control"

placeholder="<$ \_\_('Username') $>" maxlength="48" >

</div>

</div>

<div class="form-group">

<input class="required email form-control" name="email" type="text" placeholder="<$ \_\_('Email') $>" />

</div>

<div class="form-group">

<input class="required form-control" name="display-name" type="text" placeholder="<$ \_\_('Display Name') $>" maxlength="48" />

</div>

<div class="row">

<div class="form-group col-sm-9">

<input class="required form-control" name="first-name" type="text" placeholder="<$ \_\_('First Name') $>" maxlength="48" />

</div>

<div class="form-group col-sm-9">

<input class="required form-control" name="last-name" type="text" placeholder="<$ \_\_('Last Name') $>" maxlength="48" />

</div>

</div>

<div class="row">

<div class="form-group col-sm-9">

<input class="required form-control" name="password" type="password"

placeholder="<$ \_\_('Password') $>" minlength="8" maxlength="64" autocomplete="off" />

</div>

<div class="form-group col-sm-9">

<input class="required form-control" name="password-confirm" type="password"

placeholder="<$ \_\_('Confirm Password') $>" minlength="8" maxlength="64" autocomplete="off" />

</div>

</div>

<div class="lcb-login-box-bottom">

<div class="links pull-left">

<a href="#" class="lcb-show-box" data-target=".lcb-login-box-login">

<$ \_\_('I already have an account') $>

</a>

</div>

<button class="btn-info btn btn-lg pull-right" type="submit">

<$ \_\_('Register') $>

</button>

</div>

</form>

<% endif %>

<footer class="lcb-login-footer">

<p class="lcb-login-footer-heart">

<$ \_\_('Made for JU CSE IT LAB by Aneek Roy') $> <span class="dash"></span>

</p>

</footer>

</div>

</section>

<% endblock %>

**chat.html :**

<div id="lcb-client" class="lcb-client">

<div class="lcb-header">

<button type="button" class="btn lcb-header-toggle">

<i class="fa fa-bars" aria-label="<$ \_\_('Toggle Navigation') $>"></i>

</button>

<div class="lcb-header-logo">Chat App</div>

</div>

<section class="lcb-sidebar">

<div class="dropdown">

<a class="lcb-account-button dropdown-toggle" data-toggle="dropdown" href="3">

<img class="lcb-account-button-avatar lcb-avatar" src="https://www.gravatar.com/avatar/<$ account.avatar $>?s=50" />

<span class="lcb-account-button-name"><$ account.displayName $></span>

<span class="lcb-account-button-username">@<$ account.username $></span>

<i class="lcb-account-button-chevron fa fa-chevron-down"></i>

</a>

<ul class="dropdown-menu dropdown-menu-right">

<li>

<a data-toggle="modal" href="#lcb-profile">

<i class="fa fa-fw fa-edit"></i> <$ \_\_('Edit Profile') $>

</a>

</li>

<li>

<a data-toggle="modal" href="#lcb-account">

<i class="fa fa-fw fa-cogs"></i> <$ \_\_('Account Settings') $>

</a>

</li>

<li>

<a data-toggle="modal" href="#lcb-notifications">

<i class="fa fa-fw fa-bell"></i> <$ \_\_('Notifications') $>

</a>

</li>

<% if settings.xmpp.enable %>

<li>

<a data-toggle="modal" href="#lcb-xmpp">

<i class="fa fa-fw fa-comments"></i> <$ \_\_('XMPP/Jabber') $>

</a>

</li>

<% endif %>

<li>

<a data-toggle="modal" href="#lcb-tokens">

<i class="fa fa-fw fa-lock"></i> <$ \_\_('Auth Tokens') $>

</a>

</li>

<li class="divider"></li>

<li><a href="./logout"><i class="fa fa-fw fa-sign-out"></i> <$ \_\_('Logout') $></a></li>

</ul>

</div>

<div class="lcb-status-indicators">

<span class="lcb-status-indicator lcb-status-indicator-error lcb-status-connection" data-status="disconnected" style="display: none;">

<i class="fa fa-plug"></i> <$ \_\_('Disconnected') $>

</span>

<span class="lcb-status-indicator lcb-status-connection" data-status="connected" style="display: none;">

<i class="fa fa-refresh"></i> <$ \_\_('Connected') $>

</span>

<span class="lcb-status-indicator lcb-status-connection" style="display: none;">

<i class="fa fa-bell-slash"></i>

</span>

</div>

<div class="lcb-version">

<!-- v <$ version $> -->

</div>

<div class="lcb-tabs-outer">

<ul class="lcb-tabs">

<li data-id="list" class="lcb-tab lcb-tab-home">

<a href="#!/">

<i class="fa fa-th" aria-label="<$ \_\_('Home') $>"></i>

</a>

</li>

</ul>

</div>

</section>

<section class="lcb-panes">

<div class="lcb-rooms-browser lcb-pane hide" data-id="list">

<header class="lcb-room-header">

<div class="lcb-room-meta">

<h2 class="lcb-room-heading lcb-room-heading-loud"><$ \_\_('All Rooms') $></h2>

</div>

<div class="lcb-rooms-browser-filter">

<label class="lcb-rooms-browser-filter-label" for="lcb-rooms-browser-filter-input">

<i class="fa fa-search" aria-label="<$ \_\_('Search') $>"></i>

</label>

<input id="lcb-rooms-browser-filter-input" class="lcb-rooms-browser-filter-input" type="search" autofocus placeholder="<$ \_\_('Find a room...') $>" />

</div>

<div class="lcb-room-header-actions">

<a class="btn btn-success lcb-rooms-browser-new" data-toggle="modal" data-target="#lcb-add-room" href="#!/rooms/add">

<i class="fa fa-plus" aria-label="<$ \_\_('Add Room') $>"></i>

</a>

</div>

</header>

<header class="lcb-rooms-browser-header hide">

<h2 class="lcb-rooms-browser-heading">

<$ \_\_('All Rooms') $>

</h2>

</header>

<ul class="lcb-rooms-list"></ul>

<% include 'includes/modals/add-room.html' %>

<% include 'includes/modals/password.html' %>

</div>

</section>

<div class="lcb-loading lcb-client-loading">

<img class="lcb-loading-indicator" src="./media/img/loading.svg" alt="<$ \_\_('Loading') $>..." />

</div>

**Transcript.html**

<div class="lcb-transcript">

<input type="hidden" name="room-id" value="<$ room.id $>" />

<input type="hidden" name="room-name" value="<$ room.name $>" />

<div class="lcb-transcript-header">

<h1 class="lcb-transcript-heading"><$ \_\_('Transcript for') $> <strong><$ room.name $></strong></h1>

<div class="lcb-transcript-filters clearfix">

<div class="lcb-transcript-daterange">

<i class="fa fa-calendar fa-lg"></i>

<span class="lcb-transcript-daterange-range"></span>

<b class="caret"></b>

</div>

<% if text\_search %>

<input type="search" class="lcb-search-entry form-control" placeholder="<$ \_\_('Search') $>..." />

<% endif %>

</div>

</div>

<ul class="lcb-messages lcb-transcript-messages"></ul>

</div>