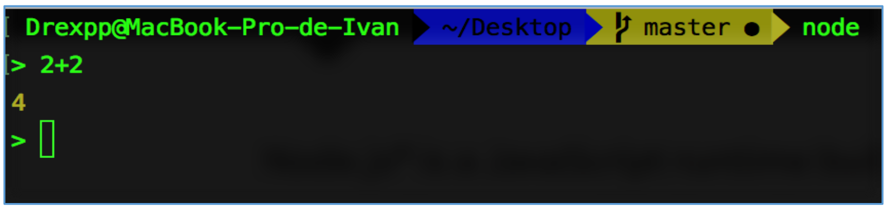
Development of a multiroom chat application using node.js and socket.io for its later deployment on a PaaS as Herkou

To start off with the development of this application we based on the book “Node.js in action” part one “Node fundamentals” chapter two “Building a multiroom chat application”. First we are using node since it is a new technology that let us to quickly build robust and fast server side applications using javascript which bring us the benefits of asynchronous code. Also Node.js offers good sinergy with emerging technologies in the client side like React.js, Vue.js,…

We first needed to download node from its official website (node). Then we can just go to our terminal and we will have avaliable the option to write “node” and open a CLI (Command Line Interface) for writing javascript code just as shown in the picture below.

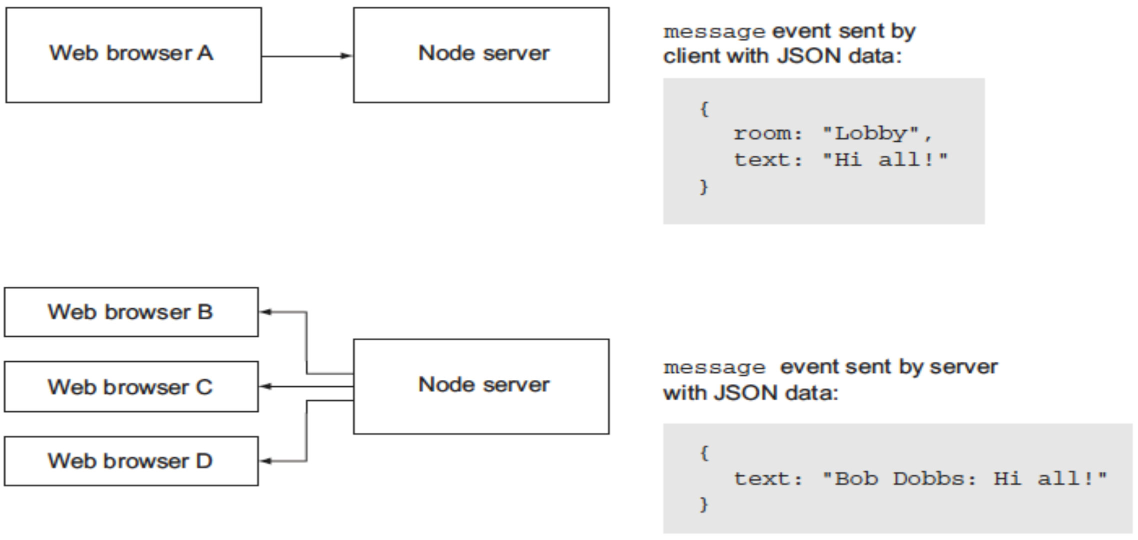


```
Drexpp@MacBook-Pro-de-Ivan  ~/Desktop  master ●  node
> 2+2
4
> 
```

We are sure we have node installed and working.

Since the application is covered in about 25 pages in the book, we will summarize why socket.io is chosen, what structure of folders and files has the Project and most important parts of it with screencaptures and comments about what the application in each file, all code is commented. Lastly we will go over how to deploy our application with heroku in few steps.

1. Why socket.io?
   Socket.io is just a javascript library that uses the websocket protocol avaliable in most web browsers. Socket-io enables bi-directional communication between web clients and servers. In our case it let us control when user connects to rooms, sending messages, changing nicknames, changing rooms and disconnection of users. We will emit a message to the server from the server side for example when sending a message. The server will be “waiting (on)” to those messages to proccess then and reply in the same to the web client which will be also waiting for similar messages from the server.

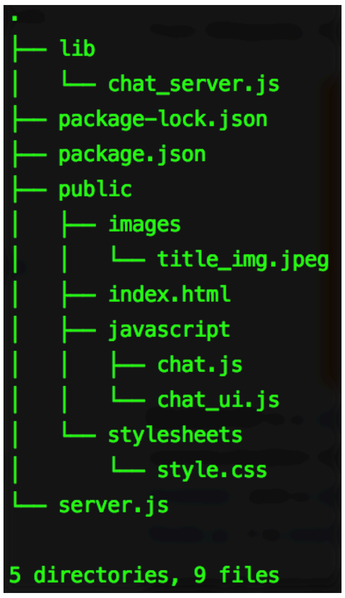

Web browser A
Node server
message event sent by client with JSON data:
Web browser B
Web browser C
Web browser D
Node server
message event sent by server with JSON data:

```
{
   room: "Lobby",
   text: "Hi all!"
}
```

```
{
   text: "Bob Dobbs: Hi all!"
}
```

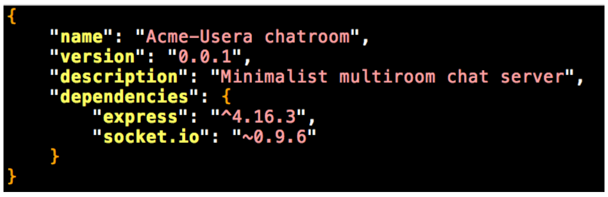
Figure 2.11 **Sending a chat message**

2. Structure of folders and files



```
.
├── lib
│   └── chat_server.js
├── package-lock.json
├── package.json
├── public
│   ├── images
│   │   └── title_img.jpeg
│   ├── index.html
│   ├── javascript
│   │   ├── chat.js
│   │   └── chat_ui.js
│   └── stylesheets
│       └── style.css
└── server.js

5 directories, 9 files
```

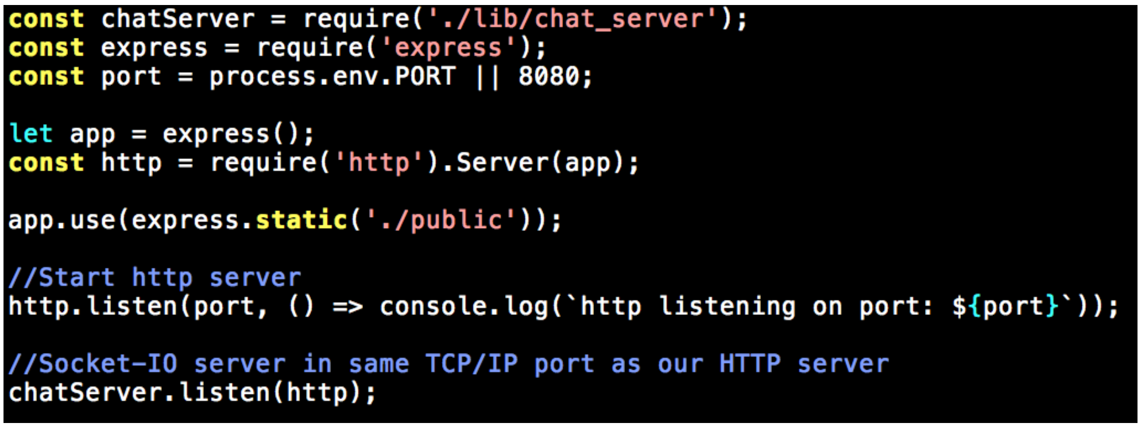
- package.json y package-lock.json: They are just dependency files which will enable to use "npm install" inside our directory file to install all needed dependency libraries, we will have node and express. (NPM is a package manager that will install the needed dependencies.)



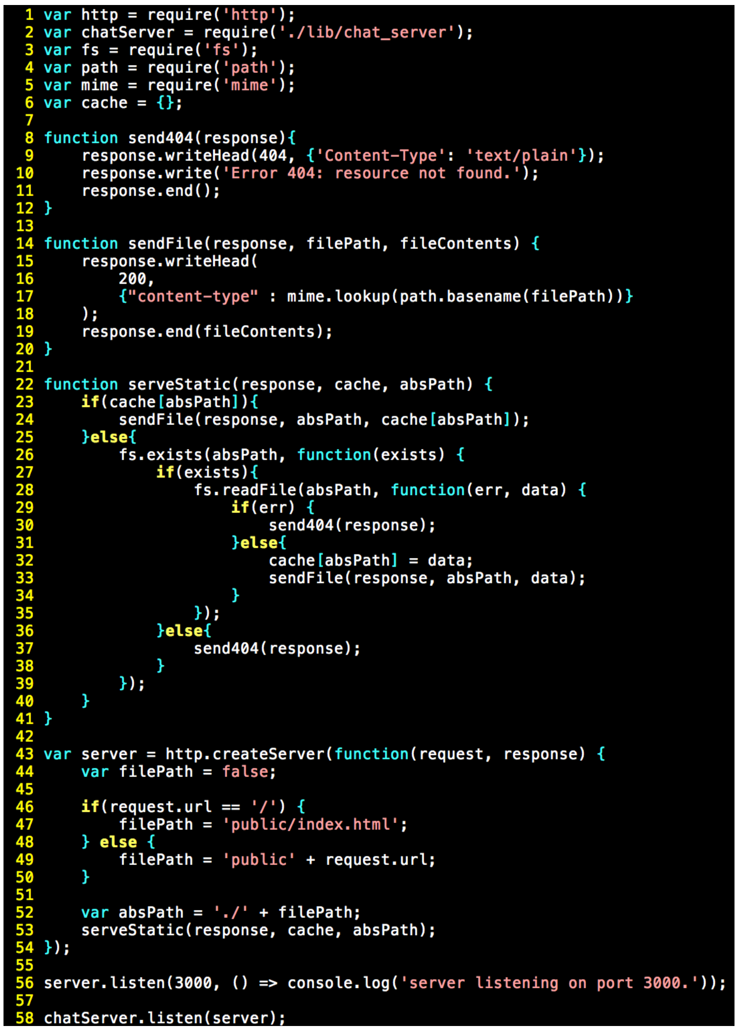
```
{
    "name": "Acme-Usera chatroom",
    "version": "0.0.1",
    "description": "Minimalist multiroom chat server",
    "dependencies": {
        "express": "^4.16.3",
        "socket.io": "~0.9.6"
    }
}
```

- server.js: Is the main file of our server, it will create the http server and will help socket.io to listen to our server. “app.use” will tell express where to look for client for serving those files, in “/public” folder (files which will be downloaded in client browser). By default index.html will be launched by express as our main page, we will only need and index.html file for the Project.

We changed this part a lot from the original book, also using express which the original application in the book didn’t use, we can see the difference between the original server.js file and our file which works the same way:

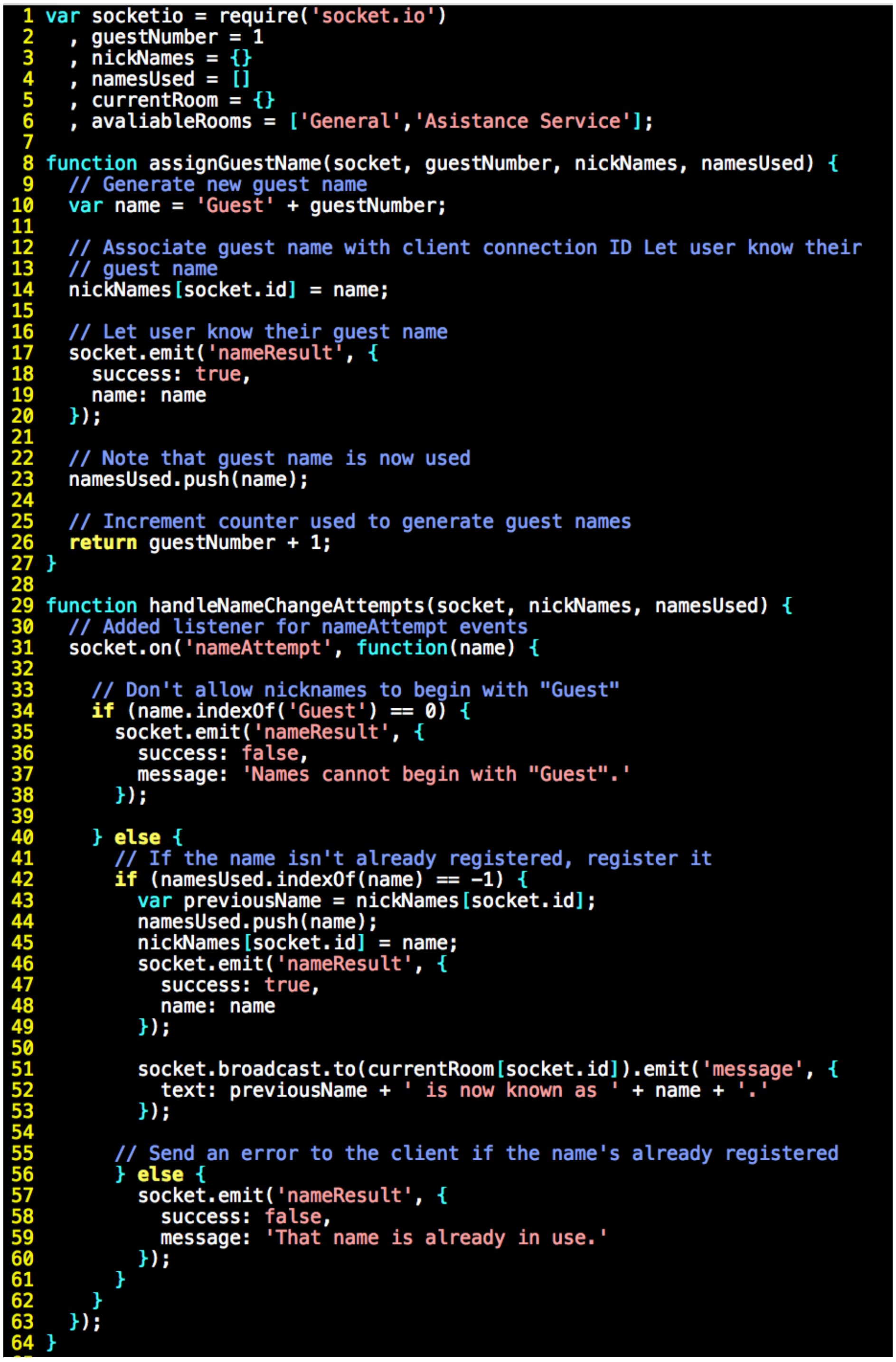


```
const chatServer = require('./lib/chat_server');
const express = require('express');
const port = process.env.PORT || 8080;

let app = express();
const http = require('http').Server(app);

app.use(express.static('./public'));

//Start http server
http.listen(port, () => console.log(`http listening on port: ${port}`));

//Socket-IO server in same TCP/IP port as our HTTP server
chatServer.listen(http);
```



```
var http = require('http');
var chatServer = require('./lib/chat_server');
var fs = require('fs');
var path = require('path');
var mime = require('mime');
var cache = {};

function send404(response){
    response.writeHead(404, {'Content-Type': 'text/plain'});
    response.write('Error 404: resource not found.');
    response.end();
}

function sendFile(response, filePath, fileContents) {
    response.writeHead(
        200,
        {"content-type" : mime.lookup(path.basename(filePath))}
    );
    response.end(fileContents);
}

function serveStatic(response, cache, absPath) {
    if(cache[absPath]){
        sendFile(response, absPath, cache[absPath]);
    }else{
        fs.exists(absPath, function(exists) {
            if(exists){
                fs.readFile(absPath, function(err, data) {
                    if(err) {
                        send404(response);
                    }else{
                        cache[absPath] = data;
                        sendFile(response, absPath, data);
                    }
                });
            }else{
                send404(response);
            }
        });
    }
}

var server = http.createServer(function(request, response) {
    var filePath = false;

    if(request.url == '/') {
        filePath = 'public/index.html';
    } else {
        filePath = 'public' + request.url;
    }

    var absPath = './' + filePath;
    serveStatic(response, cache, absPath);
});

server.listen(3000, () => console.log('server listening on port 3000.'));

chatServer.listen(server);
```

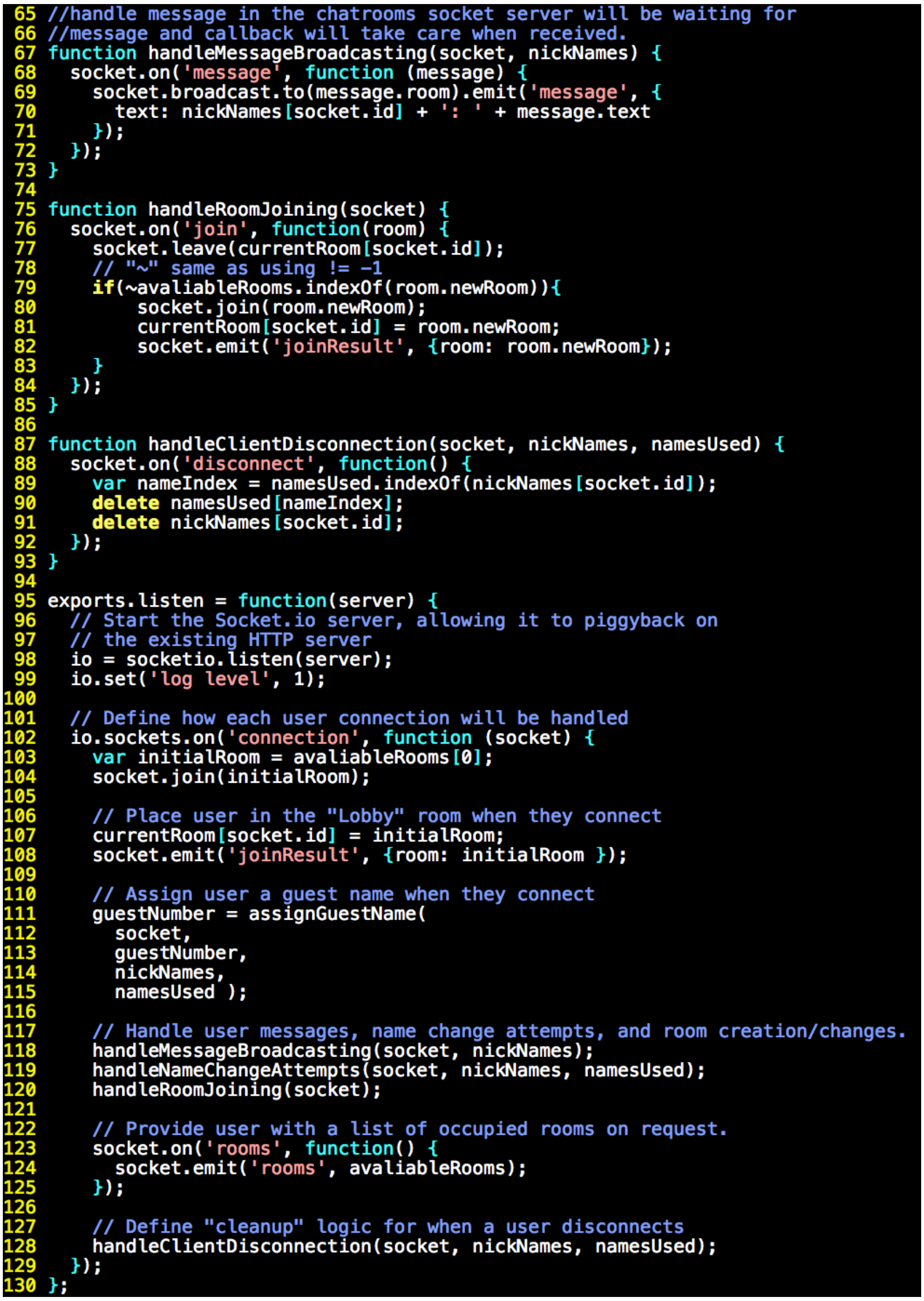
- lib/chat_server.js: This folder has the main socket.io functionality.



```
var socketio = require('socket.io')
  , guestNumber = 1
  , nickNames = {}
  , namesUsed = []
  , currentRoom = {}
  , avaliableRooms = ['General','Asistance Service'];

function assignGuestName(socket, guestNumber, nickNames, namesUsed) {
  // Generate new guest name
  var name = 'Guest' + guestNumber;

  // Associate guest name with client connection ID Let user know their
  // guest name
  nickNames[socket.id] = name;

  // Let user know their guest name
  socket.emit('nameResult', {
    success: true,
    name: name
  });

  // Note that guest name is now used
  namesUsed.push(name);

  // Increment counter used to generate guest names
  return guestNumber + 1;
}

function handleNameChangeAttempts(socket, nickNames, namesUsed) {
  // Added listener for nameAttempt events
  socket.on('nameAttempt', function(name) {

    // Don't allow nicknames to begin with "Guest"
    if (name.indexOf('Guest') == 0) {
      socket.emit('nameResult', {
        success: false,
        message: 'Names cannot begin with "Guest".'
      });

    } else {
      // If the name isn't already registered, register it
      if (namesUsed.indexOf(name) == -1) {
        var previousName = nickNames[socket.id];
        namesUsed.push(name);
        nickNames[socket.id] = name;
        socket.emit('nameResult', {
          success: true,
          name: name
        });

        socket.broadcast.to(currentRoom[socket.id]).emit('message', {
          text: previousName + ' is now known as ' + name + '.'
        });

      // Send an error to the client if the name's already registered
      } else {
        socket.emit('nameResult', {
          success: false,
          message: 'That name is already in use.'
        });
      }
    }
  });
}
```

I this first picture we defined required attributes for this file like guest number which will be useful for showing a message to everyone in the room with “GuestX” and a number which will be incremented when someone connects with a new socket.

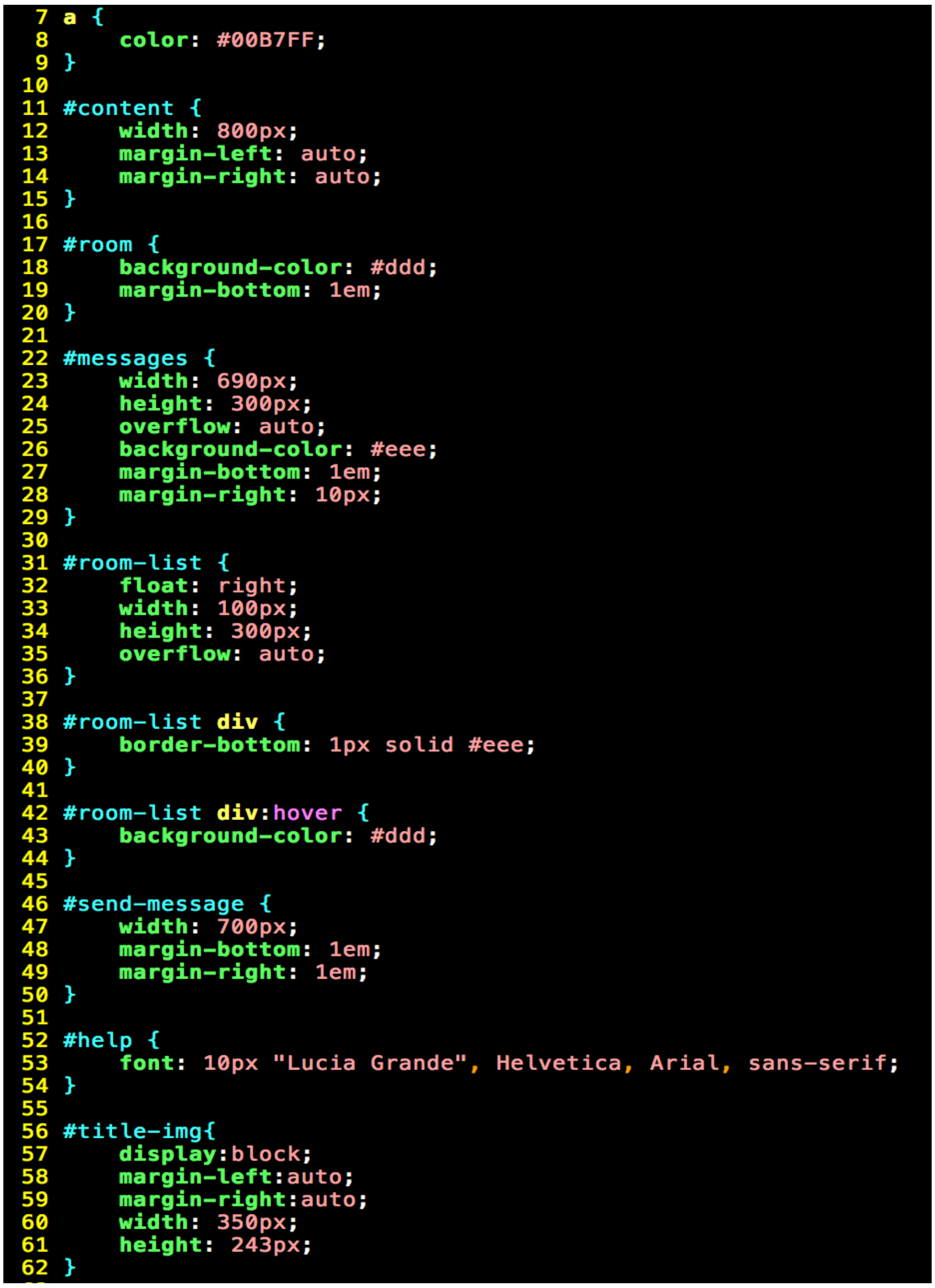
A function for assign the guest names and a function for taking care of name change attempts. (We can see socket.io emited a broadcast message which will be recived by every client web server in the currentRoom of the socket who just changed its name).



```
//handle message in the chatrooms socket server will be waiting for
//message and callback will take care when received.
function handleMessageBroadcasting(socket, nickNames) {
  socket.on('message', function (message) {
    socket.broadcast.to(message.room).emit('message', {
      text: nickNames[socket.id] + ': ' + message.text
    });
  });
}

function handleRoomJoining(socket) {
  socket.on('join', function(room) {
    socket.leave(currentRoom[socket.id]);
    // "~" same as using != -1
    if(~avaliableRooms.indexOf(room.newRoom)){
        socket.join(room.newRoom);
        currentRoom[socket.id] = room.newRoom;
        socket.emit('joinResult', {room: room.newRoom});
    }
  });
}

function handleClientDisconnection(socket, nickNames, namesUsed) {
  socket.on('disconnect', function() {
    var nameIndex = namesUsed.indexOf(nickNames[socket.id]);
    delete namesUsed[nameIndex];
    delete nickNames[socket.id];
  });
}

exports.listen = function(server) {
  // Start the Socket.io server, allowing it to piggyback on
  // the existing HTTP server
  io = socketio.listen(server);
  io.set('log level', 1);

  // Define how each user connection will be handled
  io.sockets.on('connection', function (socket) {
    var initialRoom = avaliableRooms[0];
    socket.join(initialRoom);

    // Place user in the "Lobby" room when they connect
    currentRoom[socket.id] = initialRoom;
    socket.emit('joinResult', {room: initialRoom });

    // Assign user a guest name when they connect
    guestNumber = assignGuestName(
      socket,
      guestNumber,
      nickNames,
      namesUsed );

    // Handle user messages, name change attempts, and room creation/changes.
    handleMessageBroadcasting(socket, nickNames);
    handleNameChangeAttempts(socket, nickNames, namesUsed);
    handleRoomJoining(socket);

    // Provide user with a list of occupied rooms on request.
    socket.on('rooms', function() {
      socket.emit('rooms', avaliableRooms);
    });

    // Define "cleanup" logic for when a user disconnects
    handleClientDisconnection(socket, nickNames, namesUsed);
  });
};
```

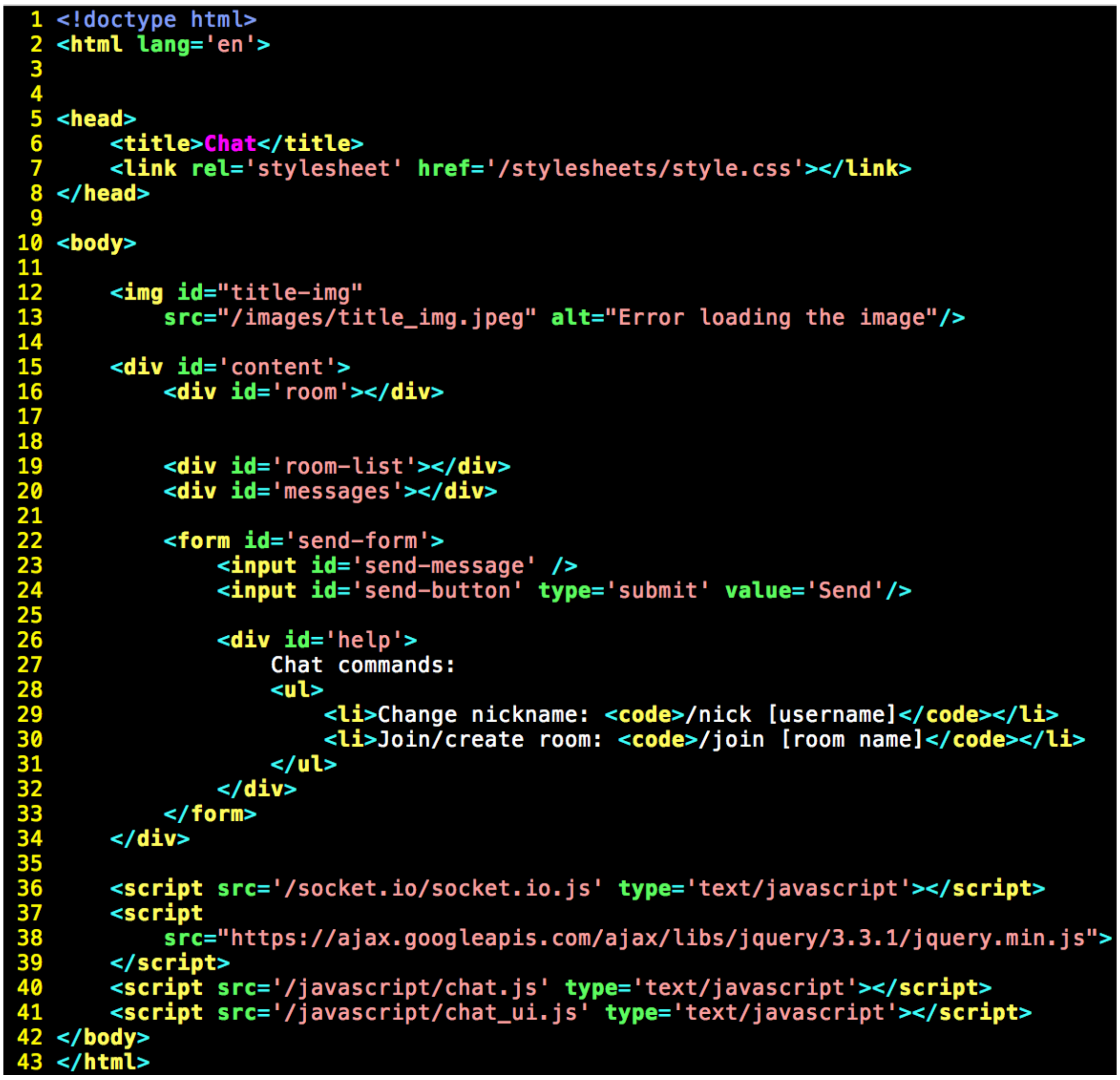
We defined the last three functions for handling sending messages exchanges in a room, joining a different room and disconnections. Finally we initialize the socket.io server.

- /Public:
    - /images: Contains the image of Acme-Usera over the chat
    - /stylesheets/style.css: A simple css file



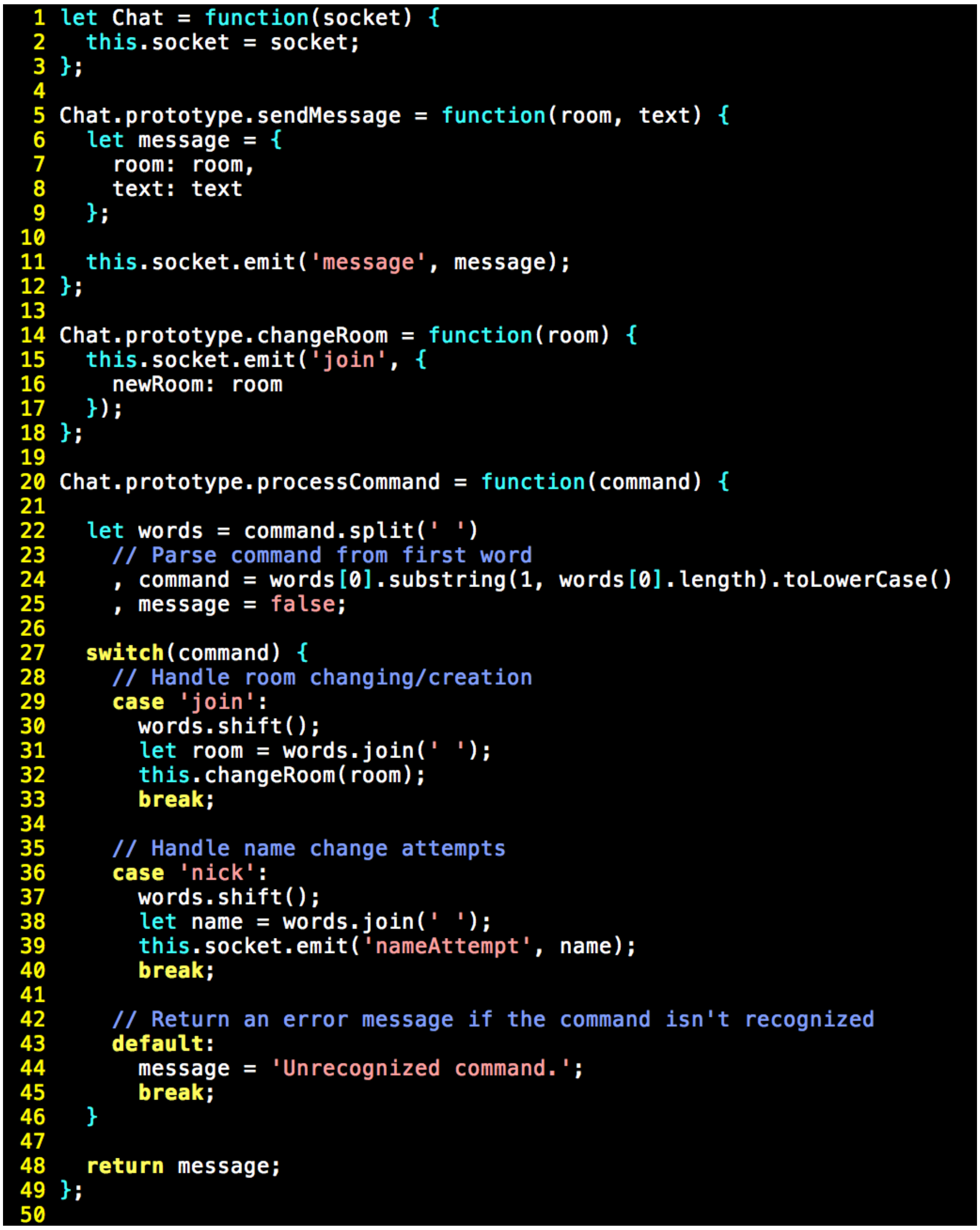
```css
a {
    color: #00B7FF;
}

#content {
    width: 800px;
    margin-left: auto;
    margin-right: auto;
}

#room {
    background-color: #ddd;
    margin-bottom: 1em;
}

#messages {
    width: 690px;
    height: 300px;
    overflow: auto;
    background-color: #eee;
    margin-bottom: 1em;
    margin-right: 10px;
}

#room-list {
    float: right;
    width: 100px;
    height: 300px;
    overflow: auto;
}

#room-list div {
    border-bottom: 1px solid #eee;
}

#room-list div:hover {
    background-color: #ddd;
}

#send-message {
    width: 700px;
    margin-bottom: 1em;
    margin-right: 1em;
}

#help {
    font: 10px "Lucia Grande", Helvetica, Arial, sans-serif;
}

#title-img{
    display:block;
    margin-left:auto;
    margin-right:auto;
    width: 350px;
    height: 243px;
}
```

- /index: Contains a basic HTML structure of the webpage.



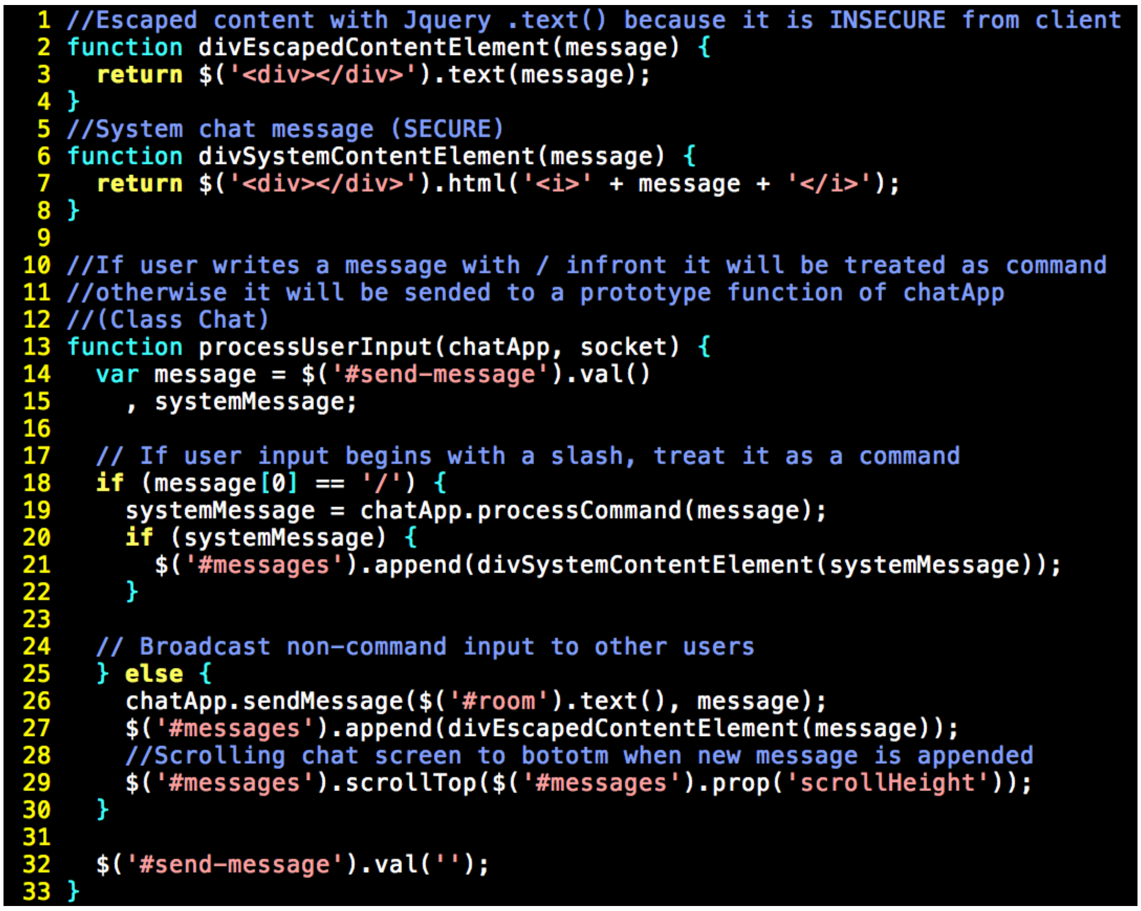
```html
<!doctype html>
<html lang='en'>


<head>
    <title>Chat</title>
    <link rel='stylesheet' href='/stylesheets/style.css'></link>
</head>

<body>

    <img id="title-img"
        src="/images/title_img.jpeg" alt="Error loading the image"/>

    <div id='content'>
        <div id='room'></div>


        <div id='room-list'></div>
        <div id='messages'></div>

        <form id='send-form'>
            <input id='send-message' />
            <input id='send-button' type='submit' value='Send'/>

            <div id='help'>
                Chat commands:
                <ul>
                    <li>Change nickname: <code>/nick [username]</code></li>
                    <li>Join/create room: <code>/join [room name]</code></li>
                </ul>
            </div>
        </form>
    </div>

    <script src='/socket.io/socket.io.js' type='text/javascript'></script>
    <script
        src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src='/javascript/chat.js' type='text/javascript'></script>
    <script src='/javascript/chat_ui.js' type='text/javascript'></script>
</body>
</html>
```

- javascript/
  - chat.js: In this file we create a Chat class and we then define prototypes for that class so we can use them later on in the chat_ui.js which takes care of showing the information and chat behaviour. We defined how the chat will process the commands of chat which are join for changing room and nick for swaping “GuestX” predefined name.



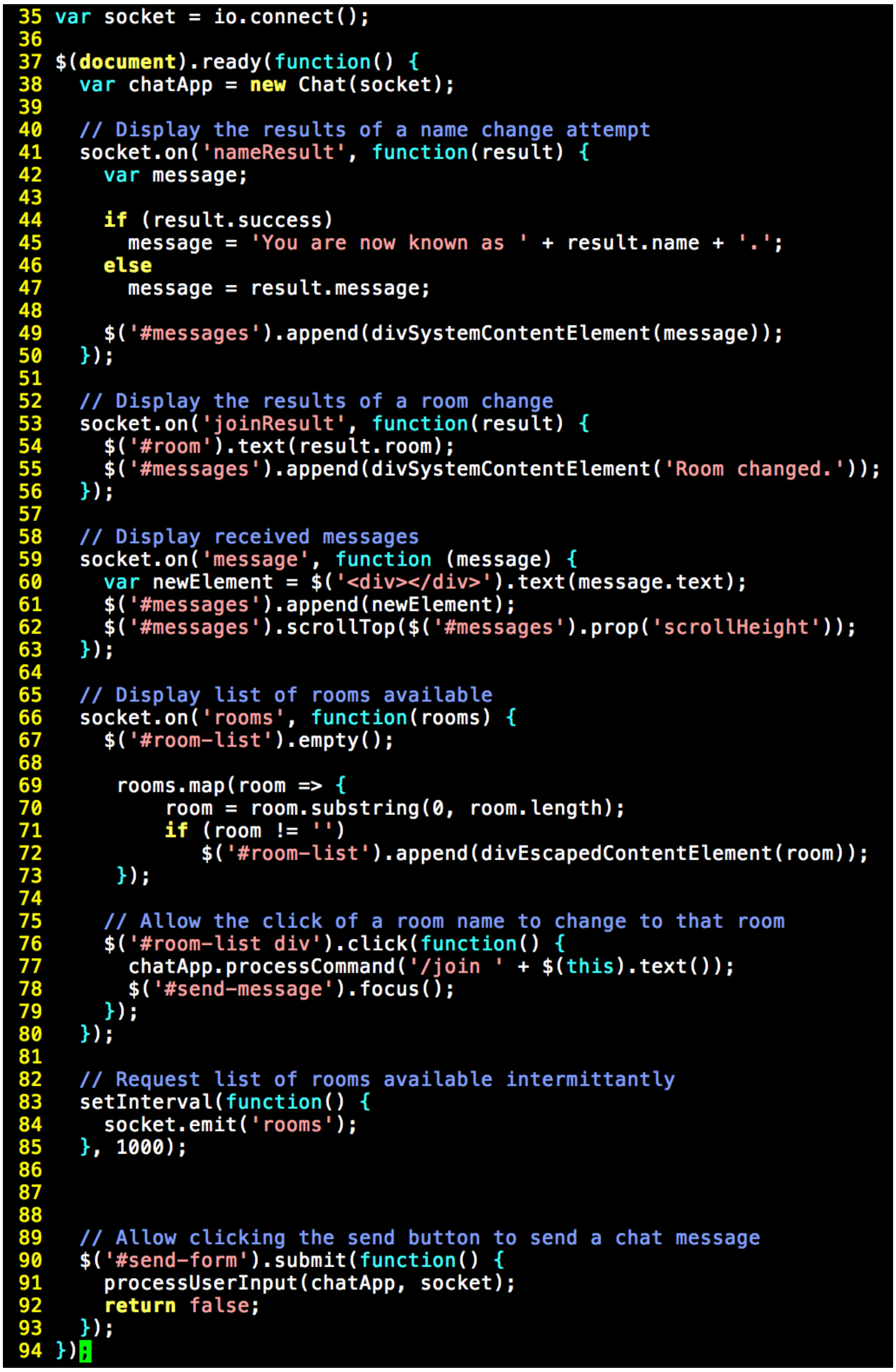
```javascript
let Chat = function(socket) {
  this.socket = socket;
};

Chat.prototype.sendMessage = function(room, text) {
  let message = {
    room: room,
    text: text
  };

  this.socket.emit('message', message);
};

Chat.prototype.changeRoom = function(room) {
  this.socket.emit('join', {
    newRoom: room
  });
};

Chat.prototype.processCommand = function(command) {

  let words = command.split(' ')
    // Parse command from first word
    , command = words[0].substring(1, words[0].length).toLowerCase()
    , message = false;

  switch(command) {
    // Handle room changing/creation
    case 'join':
      words.shift();
      let room = words.join(' ');
      this.changeRoom(room);
      break;

    // Handle name change attempts
    case 'nick':
      words.shift();
      let name = words.join(' ');
      this.socket.emit('nameAttempt', name);
      break;

    // Return an error message if the command isn't recognized
    default:
      message = 'Unrecognized command.';
      break;
  }

  return message;
};

```

- chat_ui.js: This file contains the main functionallity of the client side chat. We defined couple functions using Jquery for escaping insecure content and a different one for system messages which are secure.



```
//Escaped content with Jquery .text() because it is INSECURE from client
function divEscapedContentElement(message) {
  return $('<div></div>').text(message);
}
//System chat message (SECURE)
function divSystemContentElement(message) {
  return $('<div></div>').html('<i>' + message + '</i>');
}

//If user writes a message with / infront it will be treated as command
//otherwise it will be sended to a prototype function of chatApp
//(Class Chat)
function processUserInput(chatApp, socket) {
  var message = $('#send-message').val()
    , systemMessage;

  // If user input begins with a slash, treat it as a command
  if (message[0] == '/') {
    systemMessage = chatApp.processCommand(message);
    if (systemMessage) {
      $('#messages').append(divSystemContentElement(systemMessage));
    }

  // Broadcast non-command input to other users
  } else {
    chatApp.sendMessage($('#room').text(), message);
    $('#messages').append(divEscapedContentElement(message));
    //Scrolling chat screen to bototm when new message is appended
    $('#messages').scrollTop($('#messages').prop('scrollHeight'));
  }

  $('#send-message').val('');
}
```

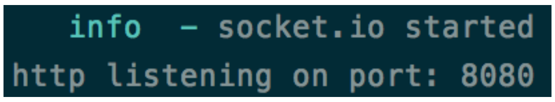
We define here the socket in the client side and the socket.on functions for waiting and processing server side emits. Sending a message calls the function defined in last page. There is an avaliable option of clicking in the name of chat room for changing room.



```javascript
var socket = io.connect();

$(document).ready(function() {
  var chatApp = new Chat(socket);

  // Display the results of a name change attempt
  socket.on('nameResult', function(result) {
    var message;

    if (result.success)
      message = 'You are now known as ' + result.name + '.';
    else
      message = result.message;

    $('#messages').append(divSystemContentElement(message));
  });

  // Display the results of a room change
  socket.on('joinResult', function(result) {
    $('#room').text(result.room);
    $('#messages').append(divSystemContentElement('Room changed.'));
  });

  // Display received messages
  socket.on('message', function (message) {
    var newElement = $('<div></div>').text(message.text);
    $('#messages').append(newElement);
    $('#messages').scrollTop($('#messages').prop('scrollHeight'));
  });

  // Display list of rooms available
  socket.on('rooms', function(rooms) {
    $('#room-list').empty();

     rooms.map(room => {
         room = room.substring(0, room.length);
         if (room != '')
            $('#room-list').append(divEscapedContentElement(room));
     });

    // Allow the click of a room name to change to that room
    $('#room-list div').click(function() {
      chatApp.processCommand('/join ' + $(this).text());
      $('#send-message').focus();
    });
  });

  // Request list of rooms available intermittantly
  setInterval(function() {
    socket.emit('rooms');
  }, 1000);



  // Allow clicking the send button to send a chat message
  $('#send-form').submit(function() {
    processUserInput(chatApp, socket);
    return false;
  });
});
```

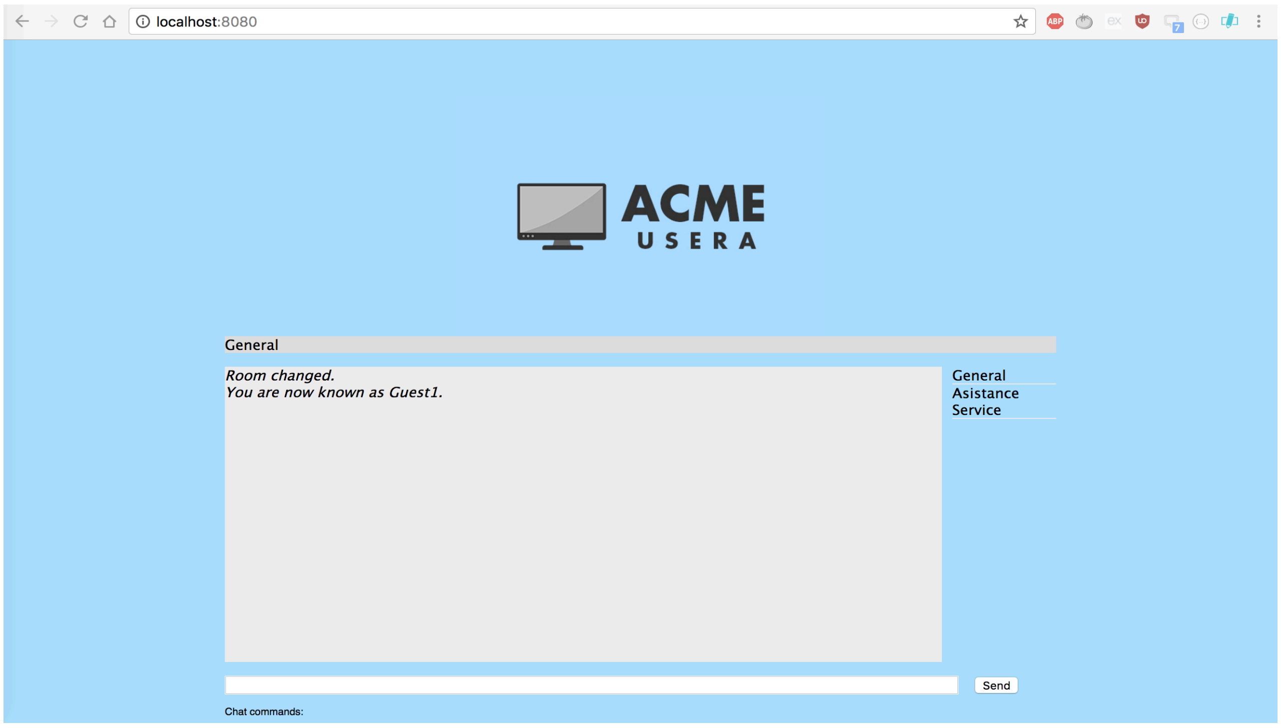
We lastly have to write in the root of our project's command line the instructions for node to deploy the application with our server.js file.

We type: "node server.js" and hopefully it shows this message



```
   info  - socket.io started
http listening on port: 8080
```

We can check our localhost port 8080



localhost:8080

General

Room changed.
You are now known as Guest1.

General
Asistance
Service

Send

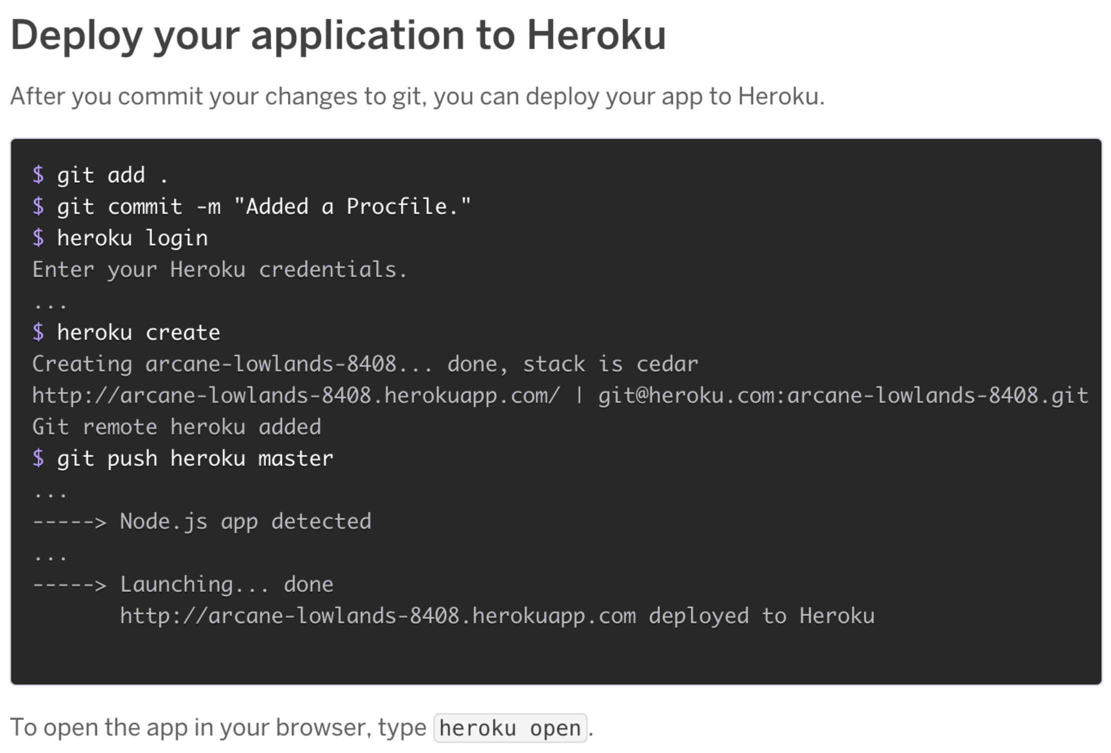
Chat commands:

We lastly have to deply our application with some platform so we don't have to deploy node in the virtual machine, which we tested it was possible using node version "v4.9.1-x86" which it is the last version which supports window XP node's use. We can't make use of arrow functions or let and const variables of javascript since it is part of ECMAscript 6 and it wasn't yet avaliable or probably even released for the V8 engine used by node.js v4.9.1 at the time, so to overcome this problem we decided to follow professor Rafael Corchuelo's advice of deploying it in a platform so we didn't have to worry about all this constraints.

Heroku makes this so easy. It has documentation avaliable about how to deploy applications of any type of language like java, pythong, go, javascript,…

We downloaded heroku CLI and type "heroku login" which enables linking our account (needed) to operate with heroku and git from command line.

We can use "git add ." and "git commit –m 'our message'" to create a local git repository, we write now "heroku create" to create our application and "git push heroku master" to deploy it in heroku platform and make it avaliable to the world.



Deploy your application to Heroku

After you commit your changes to git, you can deploy your app to Heroku.

```
$ git add .
$ git commit -m "Added a Procfile."
$ heroku login
Enter your Heroku credentials.
...
$ heroku create
Creating arcane-lowlands-8408... done, stack is cedar
http://arcane-lowlands-8408.herokuapp.com/ | git@heroku.com:arcane-lowlands-8408.git
Git remote heroku added
$ git push heroku master
...
-----> Node.js app detected
...
-----> Launching... done
       http://arcane-lowlands-8408.herokuapp.com deployed to Heroku
```

To open the app in your browser, type `heroku open`.

The link provided to us was https://safe-atoll-72115.herokuapp.com/ which can be used to chat in our hackathon project. We provide all the code in a code.zip file