

# Javascript-Socket.io

Building Portfolio page with Parallax effect, Using http-server, Saving codebase using GIT

## Objectives of the Session:

- Usage Express and Socket.io

## Prerequisites:

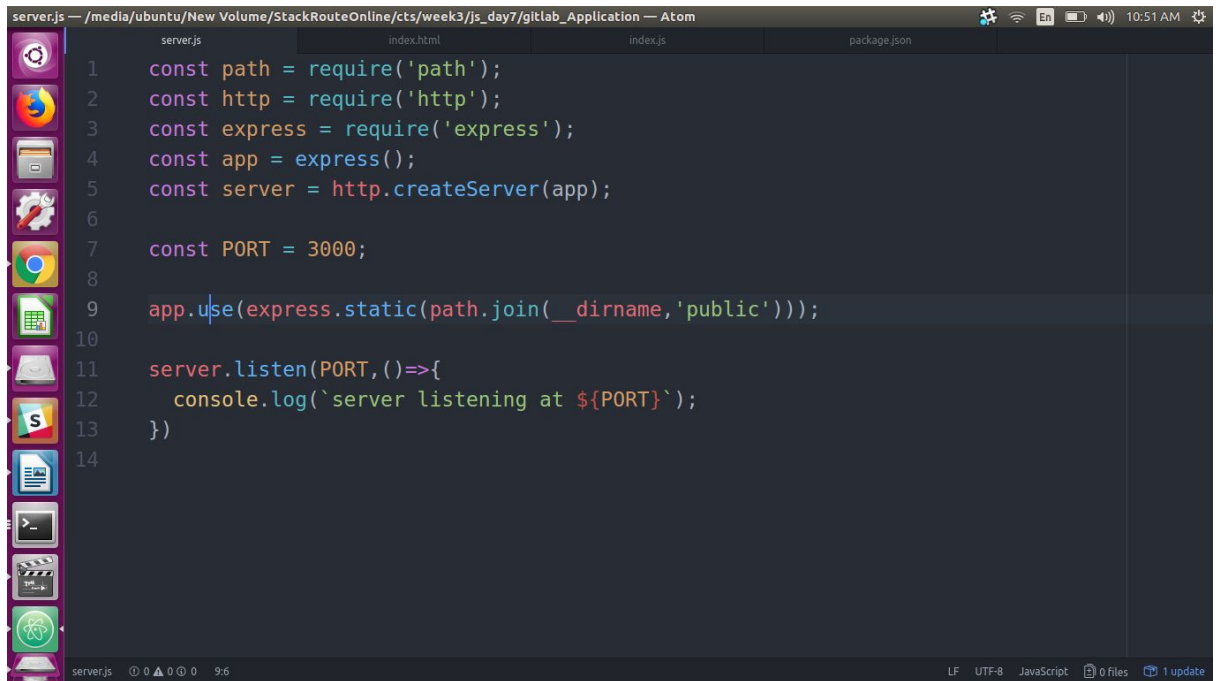
Basic understanding of html, css, bootstrap and JavaScript.

## Learning:

In this session we will learn how to add chat in gitlab.

- 1) Create a public folder
- 2) Shift all the existing code in this folder.
- 3) Change the access key
- 4) Create a package.json
- 5) Add express using `npm install express --save`

6) Create the server.js with following code

A screenshot of the Atom text editor. The title bar shows the file path: /media/ubuntu/New Volume/StackRouteOnline/cts/week3/Js\_day7/gitlab\_Application — Atom. The editor has tabs for server.js, index.html, index.js, and package.json. The server.js file is open and contains the following JavaScript code:

```
1  const path = require('path');
2  const http = require('http');
3  const express = require('express');
4  const app = express();
5  const server = http.createServer(app);
6
7  const PORT = 3000;
8
9  app.use(express.static(path.join(__dirname, 'public')));
10
11 server.listen(PORT, () => {
12   console.log(`server listening at ${PORT}`);
13 })
14
```

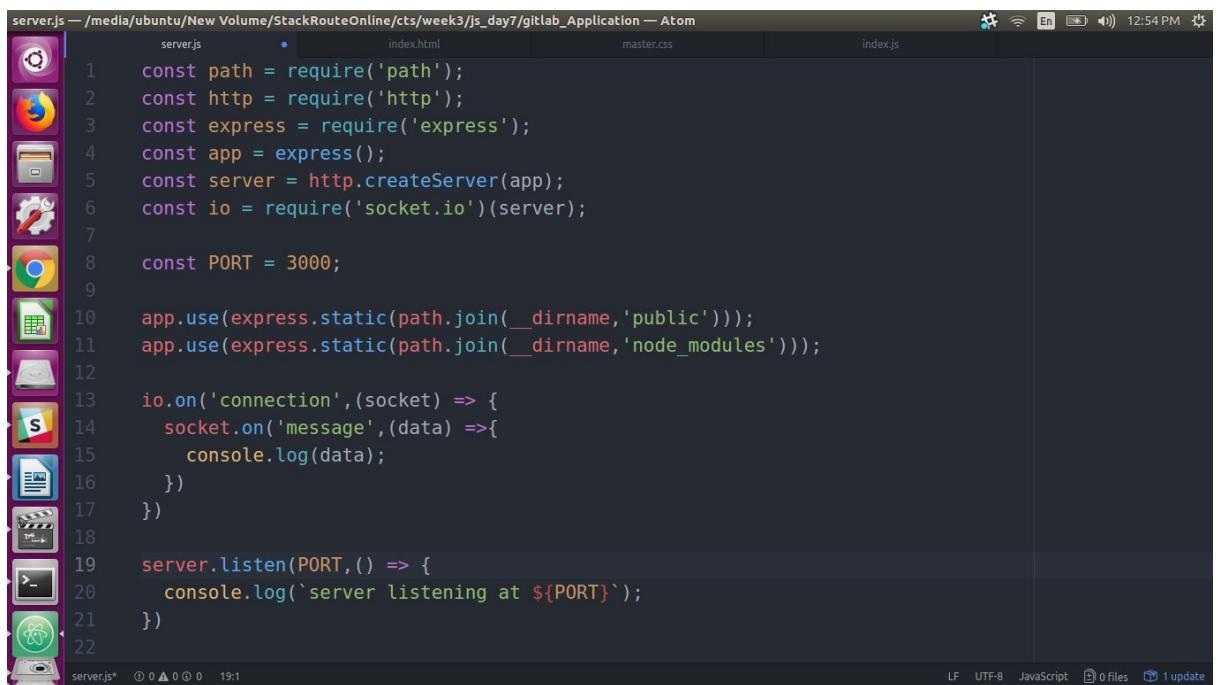
The status bar at the bottom indicates the file is server.js, encoding is UTF-8, language is JavaScript, and there are 0 files and 1 update.

7) Add the start script.

8) Run the application using npm start.

9) Install bootstrap and use it.

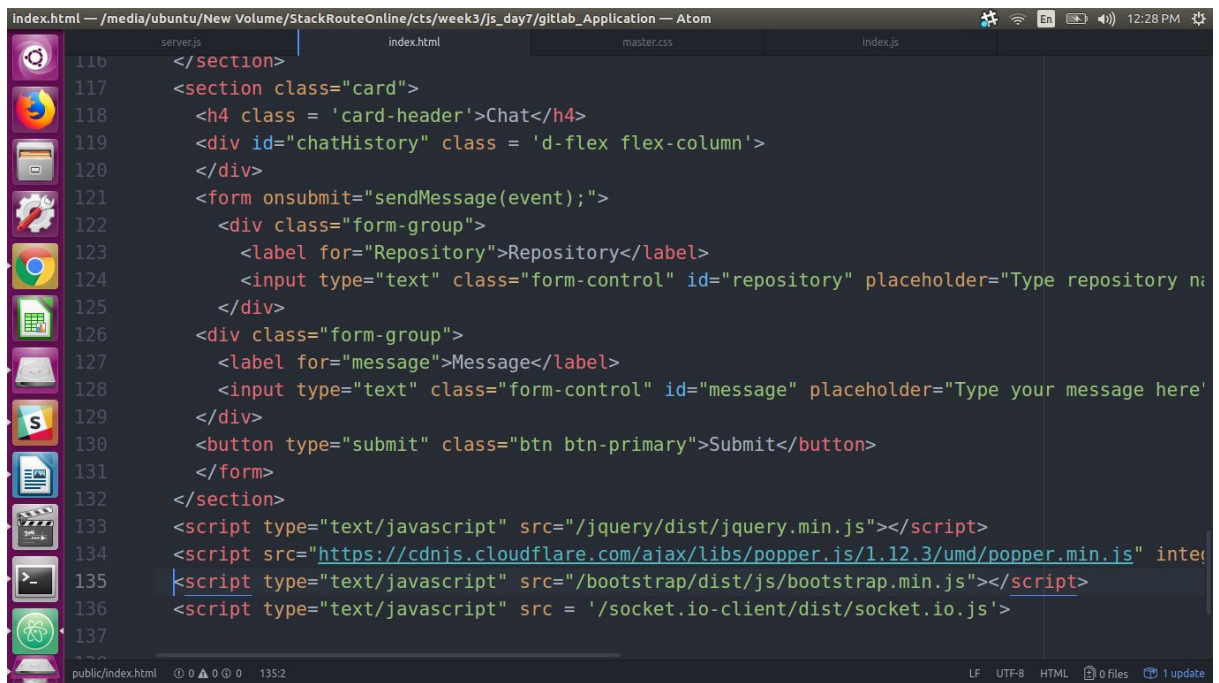
10) Add the socket code.

A screenshot of the Atom text editor. The title bar shows the file path: /media/ubuntu/New Volume/StackRouteOnline/cts/week3/Js\_day7/gitlab\_Application — Atom. The editor has tabs for server.js, index.html, master.css, and index.js. The server.js file is open and contains the following JavaScript code:

```
1  const path = require('path');
2  const http = require('http');
3  const express = require('express');
4  const app = express();
5  const server = http.createServer(app);
6  const io = require('socket.io')(server);
7
8  const PORT = 3000;
9
10 app.use(express.static(path.join(__dirname, 'public')));
11 app.use(express.static(path.join(__dirname, 'node_modules')));
12
13 io.on('connection', (socket) => {
14   socket.on('message', (data) => {
15     console.log(data);
16   })
17 })
18
19 server.listen(PORT, () => {
20   console.log(`server listening at ${PORT}`);
21 })
22
```

The status bar at the bottom indicates the file is server.js\*, encoding is UTF-8, language is JavaScript, and there are 0 files and 1 update.

11) Add Chat UI.



The screenshot shows the Atom editor with a file named 'index.html' open. The code is HTML and defines a chat interface. It includes a section with a class 'card' containing a header 'Chat', a history container 'chatHistory', and a form. The form has two input fields: 'repository' and 'message', and a 'Submit' button. The 'repository' input has a placeholder 'Type repository name' and the 'message' input has a placeholder 'Type your message here'. The form's onsubmit event is 'sendMessage(event);'. Below the form, there are four script tags: jQuery, Popper.js, Bootstrap, and Socket.io-client.

```
116 </section>
117 <section class="card">
118   <h4 class = 'card-header'>Chat</h4>
119   <div id="chatHistory" class = 'd-flex flex-column'>
120   </div>
121   <form onsubmit="sendMessage(event);">
122     <div class="form-group">
123       <label for="Repository">Repository</label>
124       <input type="text" class="form-control" id="repository" placeholder="Type repository name" />
125     </div>
126     <div class="form-group">
127       <label for="message">Message</label>
128       <input type="text" class="form-control" id="message" placeholder="Type your message here" />
129     </div>
130     <button type="submit" class="btn btn-primary">Submit</button>
131   </form>
132 </section>
133 <script type="text/javascript" src="/jquery/dist/jquery.min.js"></script>
134 <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.3/umd/popper.min.js" integrity="sha384-VeJcnSOLf8ZtrVP96MAsOJk+QIzKz0QZpRwWYq/g5VVGRG/p53GhRkn68RcXofJ" crossorigin="anonymous"></script>
135 <script type="text/javascript" src="/bootstrap/dist/js/bootstrap.min.js"></script>
136 <script type="text/javascript" src = '/socket.io-client/dist/socket.io.js'></script>
137
```

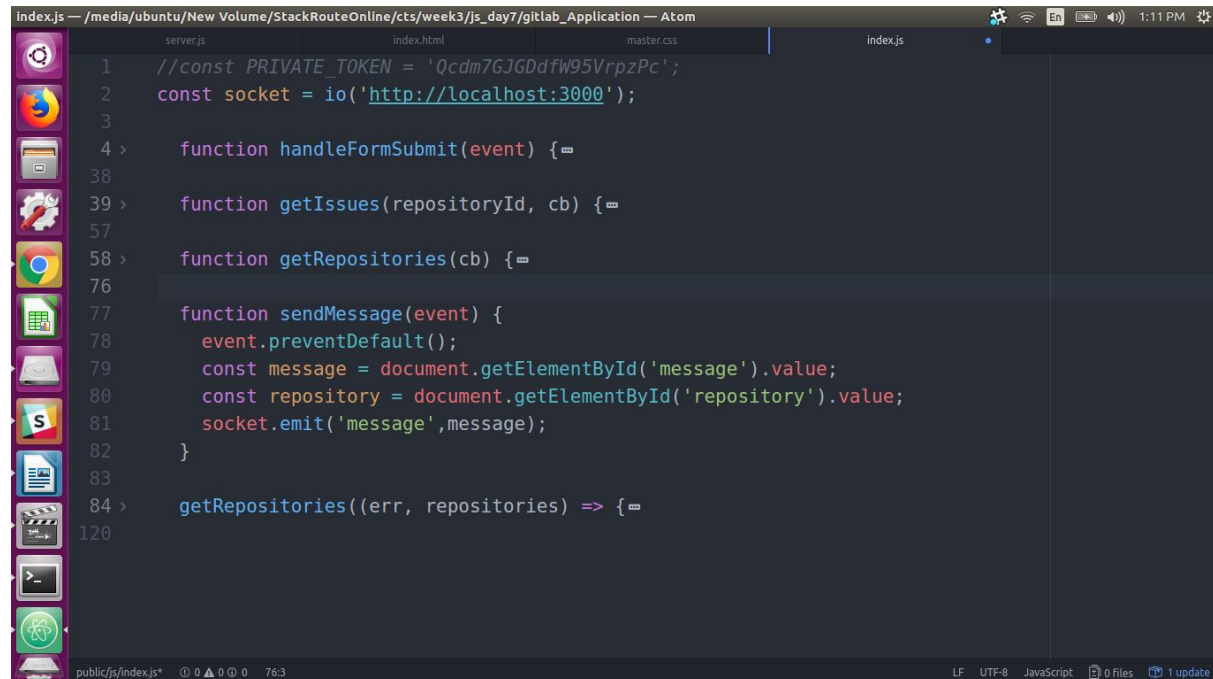
12) Add socket lib. <script type="text/javascript" src = '/socket.io-client/dist/socket.io.js'>

13) </script>

14) Add the client side code

a) Add const socket = io('http://localhost:3000'); at the top

b) Add sendMessage Method with following code.

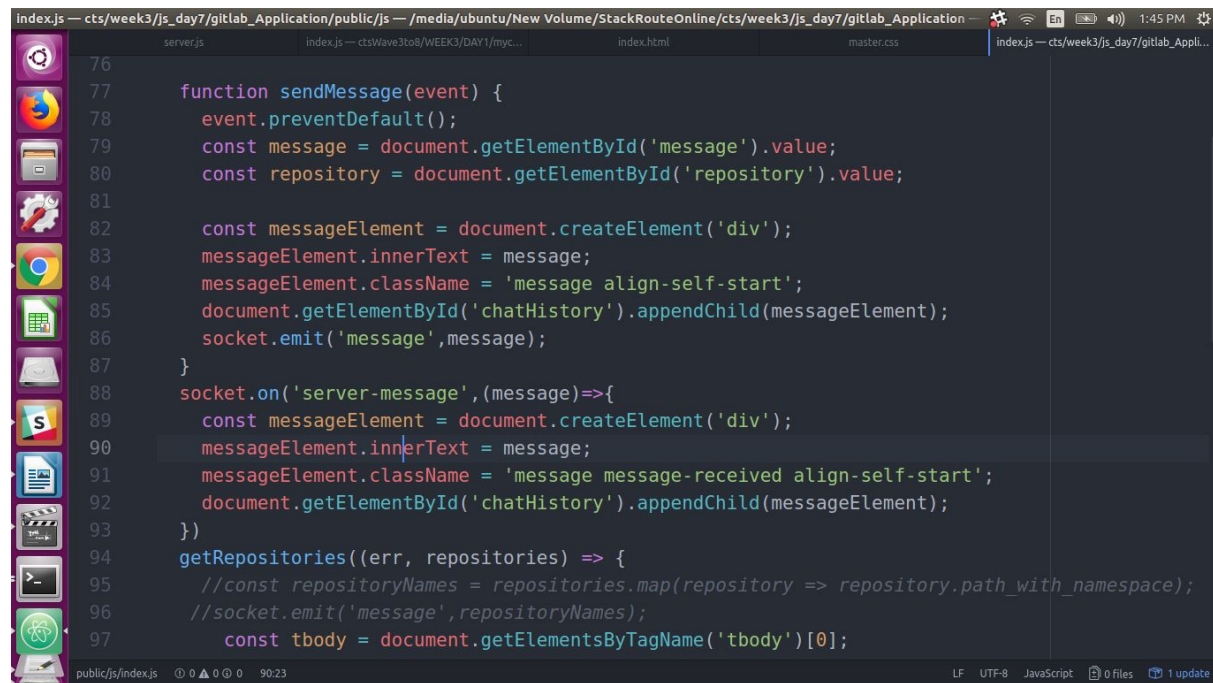


The screenshot shows the Atom editor with a file named 'index.js' open. The code is JavaScript and defines the client-side logic for the chat application. It starts with a comment and a const declaration for 'PRIVATE\_TOKEN'. Then, it defines a const 'socket' using 'io' and the URL 'http://localhost:3000'. There are three functions: 'handleFormSubmit', 'getIssues', and 'getRepositories'. The 'sendMessage' function is defined with a comment and a function body that prevents default behavior, gets the values of the 'message' and 'repository' inputs, and emits a 'message' event to the socket. The 'getRepositories' function is also defined with a comment and a function body that calls the 'getRepositories' method of the socket.

```
index.js — /media/ubuntu/New Volume/StackRouteOnline/cts/week3/js_day7/gitlab_Application — Atom
1 //const PRIVATE_TOKEN = 'Qcdm7GJGDdfw95VrpzPc';
2 const socket = io('http://localhost:3000');
3
4 > function handleFormSubmit(event) {=
38
39 > function getIssues(repositoryId, cb) {=
57
58 > function getRepositories(cb) {=
76
77 function sendMessage(event) {
78   event.preventDefault();
79   const message = document.getElementById('message').value;
80   const repository = document.getElementById('repository').value;
81   socket.emit('message',message);
82 }
83
84 > getRepositories((err, repositories) => {=
120
```

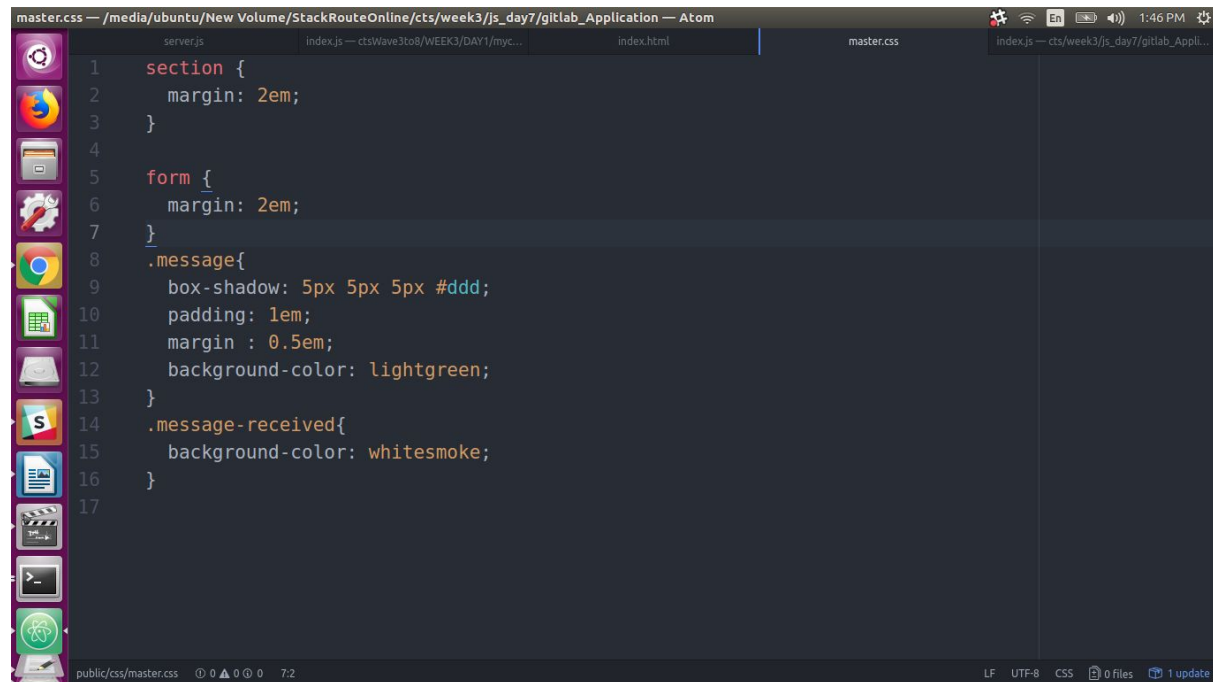
c) See the result.

d) Print the chat on ui



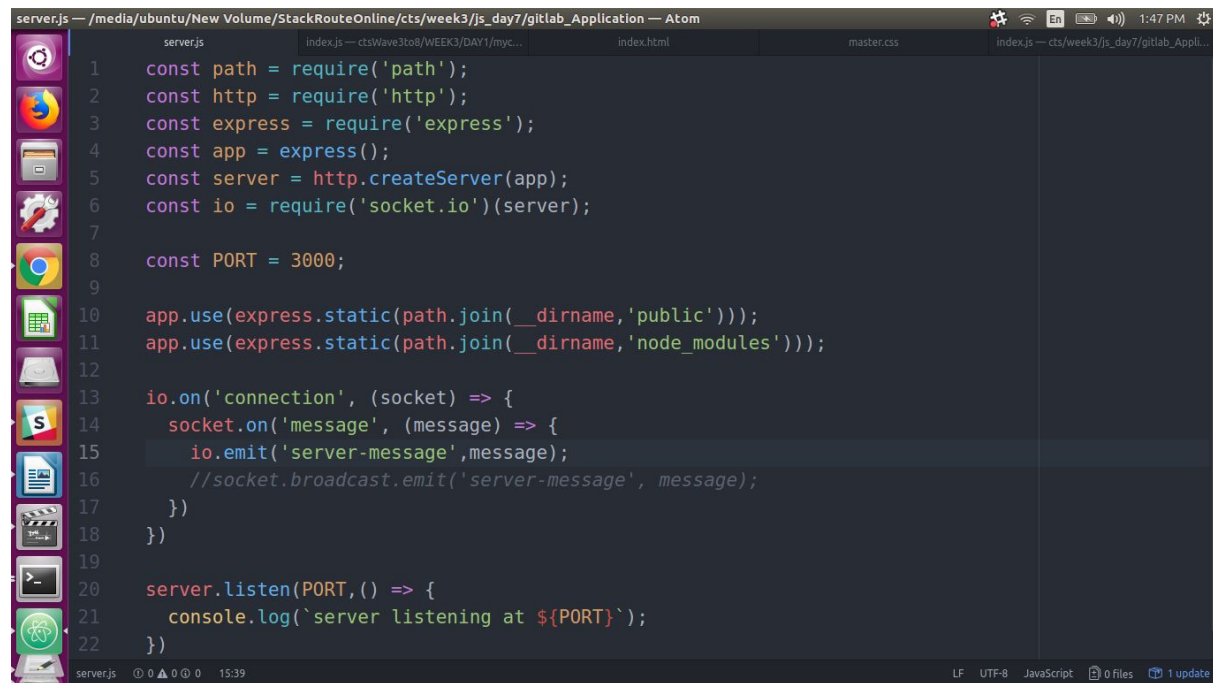
```
76
77 function sendMessage(event) {
78   event.preventDefault();
79   const message = document.getElementById('message').value;
80   const repository = document.getElementById('repository').value;
81
82   const messageElement = document.createElement('div');
83   messageElement.innerText = message;
84   messageElement.className = 'message align-self-start';
85   document.getElementById('chatHistory').appendChild(messageElement);
86   socket.emit('message', message);
87 }
88 socket.on('server-message', (message) => {
89   const messageElement = document.createElement('div');
90   messageElement.innerText = message;
91   messageElement.className = 'message message-received align-self-start';
92   document.getElementById('chatHistory').appendChild(messageElement);
93 })
94 getRepositories((err, repositories) => {
95   //const repositoryNames = repositories.map(repository => repository.path_with_namespace);
96   //socket.emit('message', repositoryNames);
97   const tbody = document.getElementsByTagName('tbody')[0];
```

e) Css



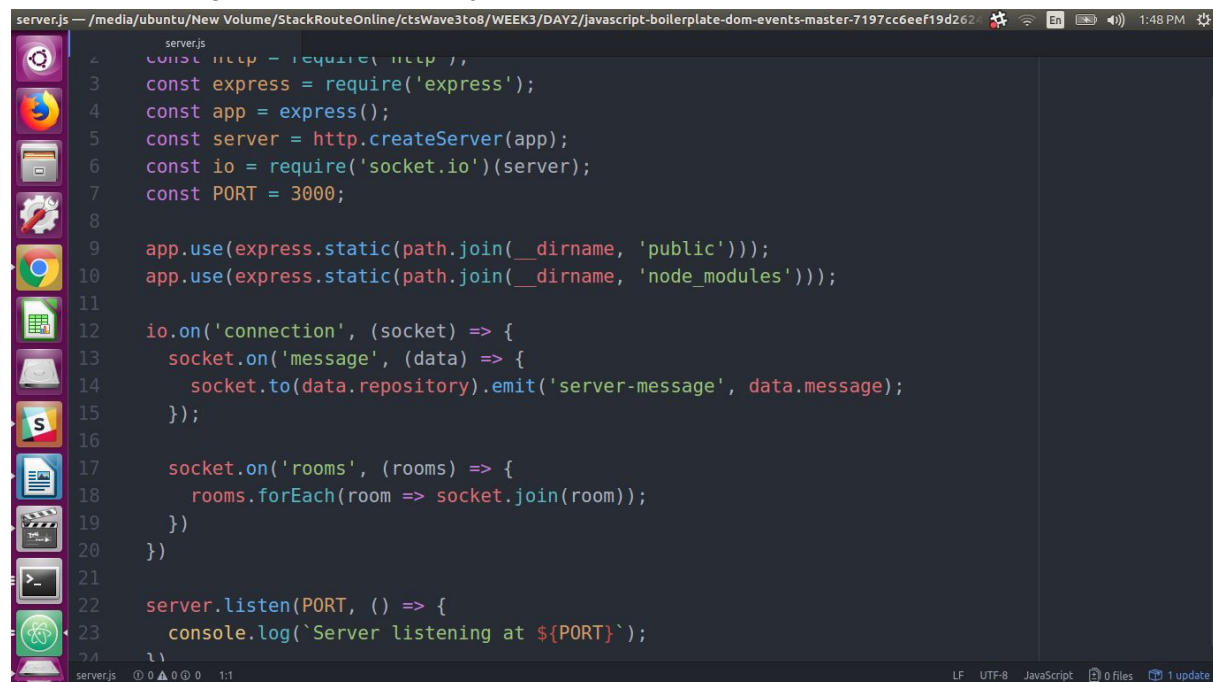
```
1 section {
2   margin: 2em;
3 }
4
5 form {
6   margin: 2em;
7 }
8 .message{
9   box-shadow: 5px 5px 5px #ddd;
10  padding: 1em;
11  margin : 0.5em;
12  background-color: lightgreen;
13 }
14 .message-received{
15   background-color: whitesmoke;
16 }
17
```

f) server.js

A screenshot of the Atom text editor interface. The title bar shows the file path: /media/ubuntu/New Volume/StackRouteOnline/cts/week3/js\_day7/gitlab\_Application - Atom. The editor displays the content of server.js. The code includes imports for path, http, express, and socket.io, sets a PORT of 3000, uses express.static for public and node\_modules, and sets up socket.io for connection and message handling. The server listens on the specified port and logs the listening status.

```
1  const path = require('path');
2  const http = require('http');
3  const express = require('express');
4  const app = express();
5  const server = http.createServer(app);
6  const io = require('socket.io')(server);
7
8  const PORT = 3000;
9
10 app.use(express.static(path.join(__dirname, 'public')));
11 app.use(express.static(path.join(__dirname, 'node_modules')));
12
13 io.on('connection', (socket) => {
14   socket.on('message', (message) => {
15     io.emit('server-message', message);
16     //socket.broadcast.emit('server-message', message);
17   })
18 })
19
20 server.listen(PORT, () => {
21   console.log(`server listening at ${PORT}`);
22 })
```

g) To send message to a room(server.js):-

A screenshot of the Atom text editor interface. The title bar shows the file path: /media/ubuntu/New Volume/StackRouteOnline/ctsWave3to8/WEEK3/DAY2/javascript-boilerplate-dom-events-master-7197cc6eef19d262 - Atom. The editor displays the content of server.js. The code includes imports for http, express, and socket.io, sets a PORT of 3000, uses express.static for public and node\_modules, and sets up socket.io for connection, message handling, and joining rooms. The server listens on the specified port and logs the listening status.

```
1  const http = require('http');
2  const express = require('express');
3  const app = express();
4  const server = http.createServer(app);
5  const io = require('socket.io')(server);
6  const PORT = 3000;
7
8
9  app.use(express.static(path.join(__dirname, 'public')));
10 app.use(express.static(path.join(__dirname, 'node_modules')));
11
12 io.on('connection', (socket) => {
13   socket.on('message', (data) => {
14     socket.to(data.repository).emit('server-message', data.message);
15   });
16
17   socket.on('rooms', (rooms) => {
18     rooms.forEach(room => socket.join(room));
19   })
20 })
21
22 server.listen(PORT, () => {
23   console.log(`Server listening at ${PORT}`);
24 })
```



```
index.html — /media/ubuntu/New Volume/StackRouteOnline/ctsWave3to8/WEEK3/DAY2/javascript-bollerplate-dom-events-master-7197cc6eef19d2
server.js index.html
131 > function handleFormSubmit(event) {=
165
166 > function getIssues(repositoryId, cb) {=
184
185 > function getRepositories(cb) {=
203
204 getRepositories((err, repositories) => {
205
206     const repositoryNames = repositories.map(repository => repository.path_with_namespace);
207
208     socket.emit('rooms', repositoryNames);
209
210     const tbody = document.getElementsByTagName('tbody')[0];
211
212     let tbodyInnerHTML = '';
213
214     repositories.forEach((repository) => {
215         getIssues(repository.id, (err, issues) => {
216             let countsInitial = {
217                 total: 0,
218                 my_issues: 0
219             };
220         });
221     });
222 }
```

h)