Tampere University of Technology

Industrial Internet Project

Web-based Room Occupancy Checker

Salman Azim
Student ID: 245038
salman.azim@tut.fi

Prasun Biswas
Student ID: 267948
prasun.biswas@student.tut.fi

Submitted on 05/09/2018

Introduction

We are living in an age where everything is becoming digital. We can't find a place where we can survive without computers. In this data driven world, the huge amount of data produced, have posed a question, that how we can use this data for the betterment of our life. For the ease of connectivity, it has become easy to connect everything together. Miniaturization of electronics and invention of sensors has also paved the way for using it to connect our household items to the internet and control them remotely. Internet of things is kind of a blessing which is making our life easier which nobody thought could be possible or needed in the first place.

The Industrial Internet project course has given us the opportunity to brainstorm innovative ideas and work with the modern ways of connectivity. We are doing the project named "**Web-based Room Occupancy Checker**" for the completion of the course **Industrial Internet Project**. In this project we are using modern web technology to check any room occupancy real-time remotely from a computer browser.

Need For this Project

While looking for ideas for this project, our focus was to work with problems we face in everyday life, specially at the university. An idea which students face in a regular basis and a solution that can make their life a bit easier. With that in mind, we came up with the problem that we face all the times that is to find empty rooms for studying or doing group assignments or finding a place for holding a group meeting. We literally need to check all the study rooms in all the buildings in the university to find an empty room. So, in this project we tried to solve this problem by coming up with a solution which is web based, so that students can check the room occupancy from the web browser real-time and that could save some potential time.

Hardware Used

There are several hardwares used for this project. The list is given below:

1. **Raspberry Pi:** Raspberry Pi is a cheap yet powerful little computer which is an essential part in handling small scale projects. It has its own operating system which is Linux based. In our case we used the Raspbian operating system. The most important feature in Raspberry Pi would be the dedicated GPIO pins on its board. These pins consist of input output pins which can connect sensors and

actuators, provide 3V or 5V power supply and ground pins. We are using the Raspberry Pi 3 Model B for this project.

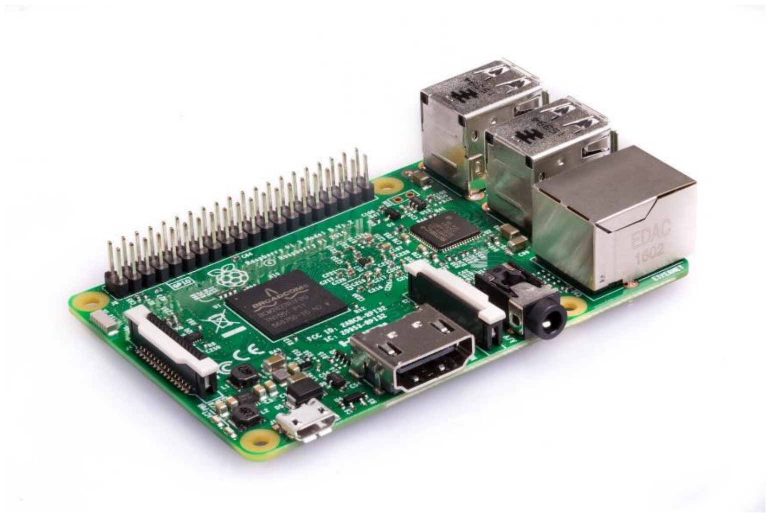


Fig 1: Raspberry Pi 3 Model B (Source: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/)

2. **PIR Sensor:** PIR stands for Passive InfraRed sensors that can detect human motion. It is a very common yet special type sensor which can detect human movement from infrared emission. This sensor can work with 3.3V or 5V supply and has a certain range. In modern PIR sensor the time for which it will hold the output signal and the sensitivity of the sensor can be adjusted by two potentiometers on the sensor. It has 3 pins, one for Input voltage, one for ground and one as Output pin. If a human movement is present in its range then it can send binary high or low signal to the connected device, in our case which is Raspberry Pi.

Fig 2: PIR Motion Sensor (Source: https://www.lelong.com.my/pir-motion-sensor-module-hc-sr501-w-adjustable-delay-time-output-si-fun4u-191168626-2018-04-Sale-P.htm)

3. **Laptop:** A laptop was used for creating local server where the data will be received from the Raspberry Pi and can be used to show in the web-based solution.

Tools Used

Lots of tools were used for this project. The list is given below:

1. Python for creating Raspberry Pi server
2. Node.js for backend programming of the main server
3. Express.js for creating HTTP server
4. HTML, CSS, Bootstrap and JavaScript for frontend programming
5. WebStorm IDE as a programming interface
6. Socket programming for sending and receiving data from python server to node.js server.

Procedure

Connecting the Sensor

First, we connected the sensor with the Raspberry Pi GPIO to get the sensor data to the Pi. We used two PIR sensors for this project. We used GPIO pin 16 and 18 for connecting the output pin of the sensors. The supplied voltage was 5V.

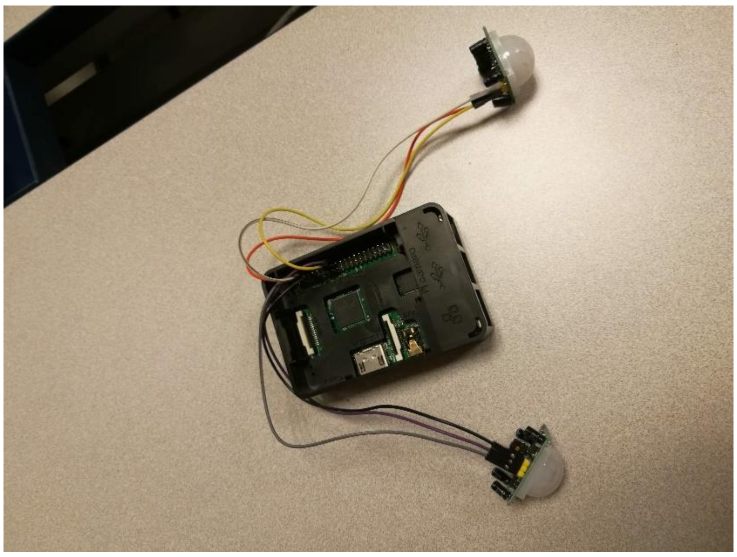
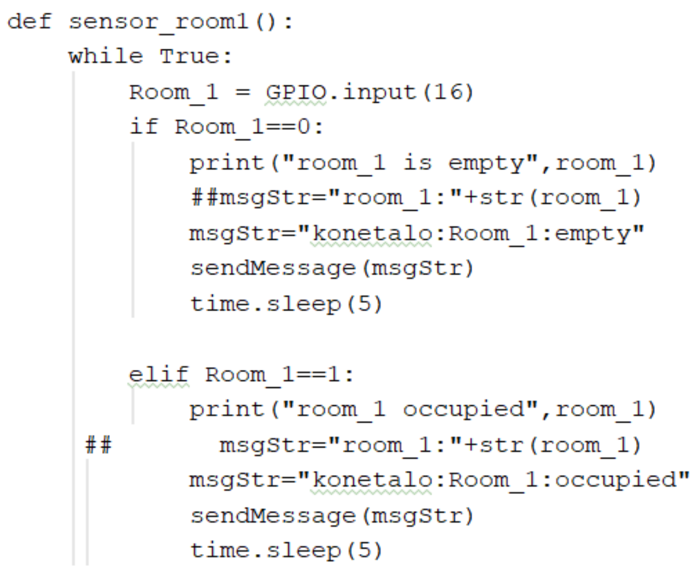


Fig3: PIR sensors connected with Raspberry PI

Creating RPI Server

After connecting the sensor, the main challenge was to get the data in a form that can be read. So, a file was created with python language as it is the most used language for projects in RPI. This file could read data from the GPIO pins by using the library RPI.GPIO. After successfully connecting the sensor and running this file, the sensor data started to appear on the command prompt of RPI as form of 0 or 1. The part of the program for collecting data from the pin 16 for sensor 1 is provided below.



```
def sensor_room1():
    while True:
        Room_1 = GPIO.input(16)
        if Room_1==0:
            print("room_1 is empty",room_1)
            ##msgStr="room_1:"+str(room_1)
            msgStr="konetalo:Room_1:empty"
            sendMessage(msgStr)
            time.sleep(5)

        elif Room_1==1:
            print("room_1 occupied",room_1)
     ##       msgStr="room_1:"+str(room_1)
            msgStr="konetalo:Room_1:occupied"
            sendMessage(msgStr)
            time.sleep(5)
```

Main server(app.js)

After setting up the RPI to get the sensor data from the PIR sensors, two immediate steps are to set up a server in RPI from which the data will be sent and to setting up a server where the data will be received on the local server or the laptop we are using in this case. For main server where the data will be received from connected RPI is a EXPRESS server run with node.js. This server will also render the room status to the user with a web view.

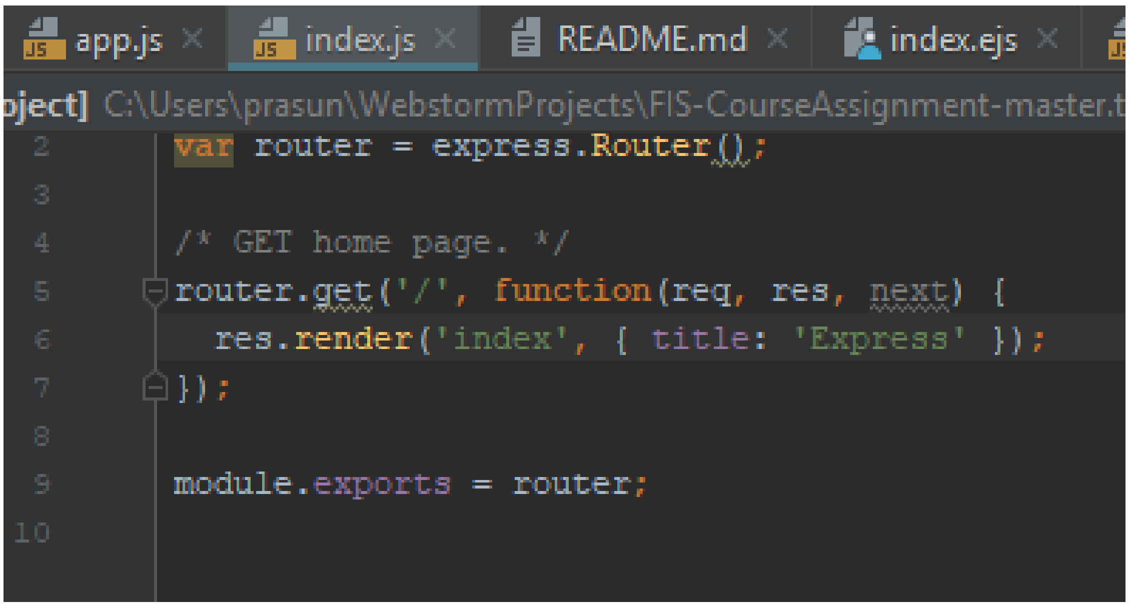
Express 4.16.3

Fast, unopinionated, minimalist web framework for Node.js

```
$ npm install express --save
```

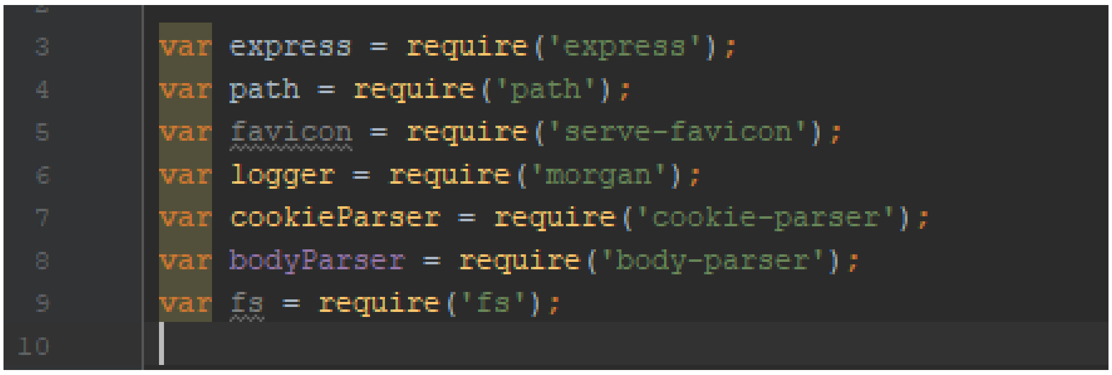
Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. With a myriad of HTTP utility methods and middleware at disposal, creating a robust API is quick and easy. Express provides a thin layer of fundamental web application features, without obscuring Node.js features and many popular frameworks are based on Express.

This express framework is very convenient for routing of webpage. Routing refers to how an application's endpoints (URIs) respond to client requests. I defined routing using methods of the Express app object that correspond to HTTP methods; for example, app.get() to handle GET requests and app.post to handle POST requests. I can also use app.all() to handle all HTTP methods and app.use() to specify middleware as the callback function (See Using middleware for details). These routing methods specify a callback function (sometimes called "handler functions") called when the application receives a request to the specified route (endpoint) and HTTP method. In other words, the application "listens" for requests that match the specified route(s) and method(s), and when it detects a match, it calls the specified callback function.In fact, the routing methods can have more than one callback function as arguments. With multiple callback functions, it is important to provide next as an argument to the callback function and then call next() within the body of the function to hand off control to the next callback. The following snippet shows the routing in our application.


app.js
index.js
README.md
index.ejs
oject] C:\Users\prasun\WebstormProjects\FIS-CourseAssignment-master.t

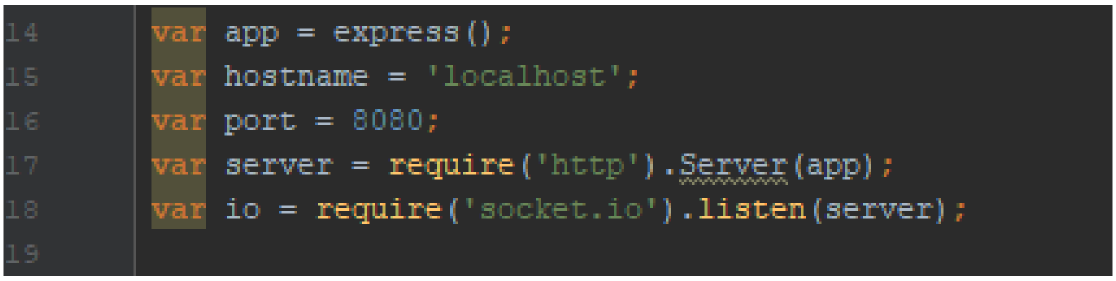
```
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});

module.exports = router;
```

The rich package library of node.js is extremely helpful during project work. This following snippet shows the installed packages that has been used in this project in the main server app named app.js.



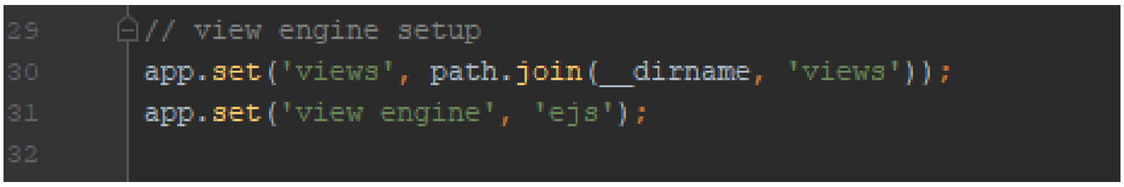
```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var fs = require('fs');

```

Using http, server.js and express we made the main server named "app". The following snippet shows the lines of codes we used to set up the server hosted in local host and the port number where the user can hit to get the room status.



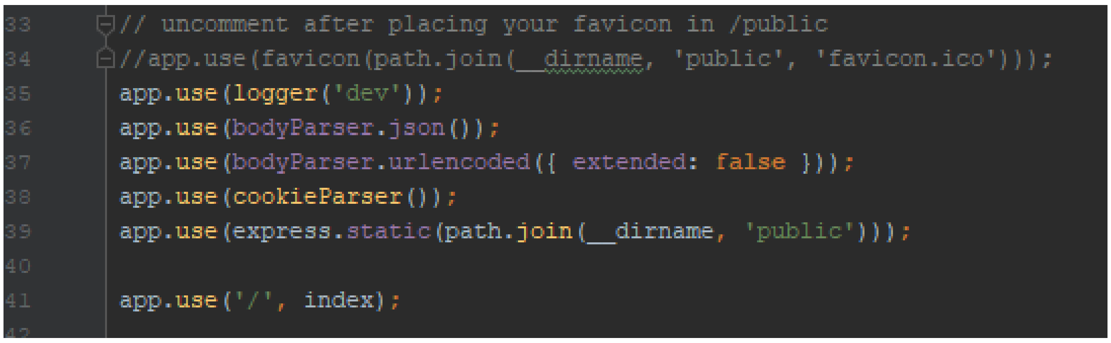
```
var app = express();
var hostname = 'localhost';
var port = 8080;
var server = require('http').Server(app);
var io = require('socket.io').listen(server);

```

To set up the view for used we used index.ejs file which located in the view folder of this express app. "ejs" stands for embedded javascript which is a simple templating language that lets us generate HTML markup with plain JavaScript. No religiousness about how to organize things and reinvention of iteration and control-flow but just plain JavaScript. Following snippet shows the code used to set up index.ejs page for this project.



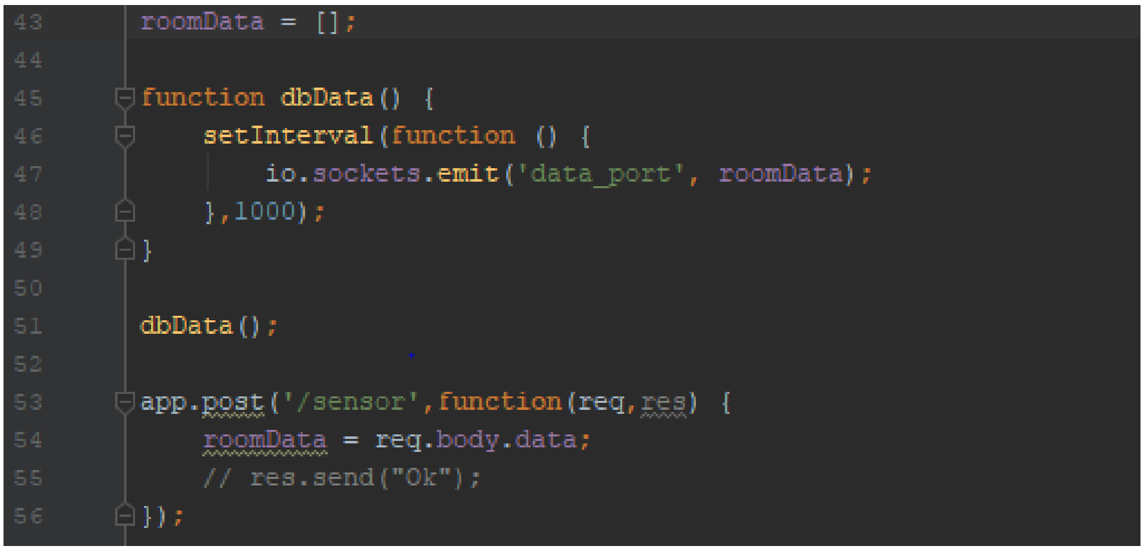
```
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

```

The final data that we receive in app.js server from a middleware server is a json formatted data. To process json formatted data we used npm package "bodyparser". To set up the path and directories for the files that are in public folder to easily accessed by the app we used "express.static.(path__)



```
// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', index);
```

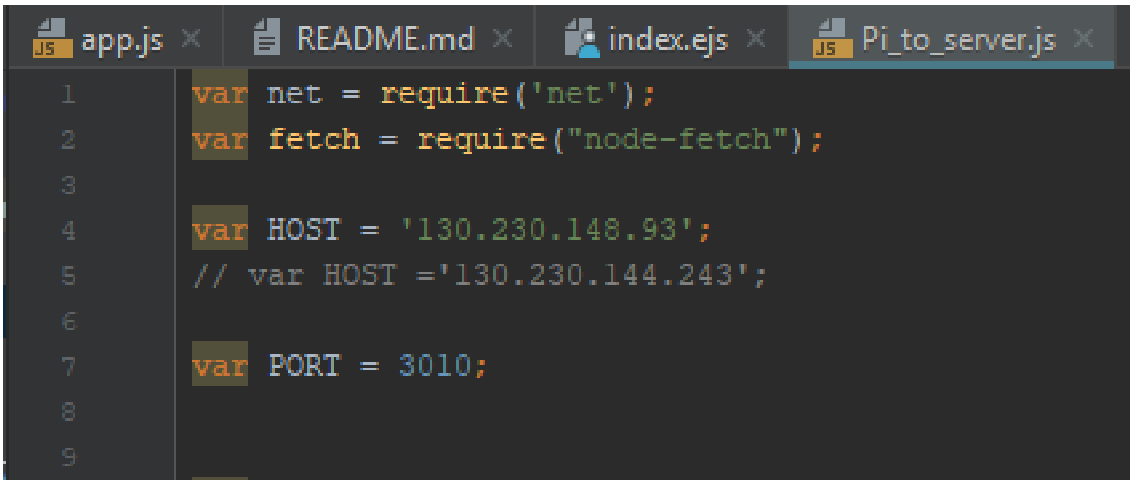
The temporary data storage in app.js server is an array named "roomdata" where the data received from middleware server as a json formatted data is stored. Name of the middleware server is "Pi_to_server.js". This application is a real-time application which is accomplished by using socket.io . The main function is "dbData()". This function includes io.socket.emit("data_port",roomdata) function inside. The "data_port" parameter is a namespace which can be used by the socket clients(in this case the client is in index.ejs file) connected with the same namespace "data_port" to receive the data "roomdata" use it to show data to user. The data in "roomdata" array get updated when "Pi_to_server" middleware server sends data by http post method after processing to build and bind in specific json data format to be used by app.js server. The data is posted to http://localhost:8080/sensor/ url which belongs to a app.post method that updates the data to roomdata array instantly. A setInterval function always sends the data to the socket clients using emit method.



```
roomData = [];

function dbData() {
    setInterval(function () {
        io.sockets.emit('data_port', roomData);
    },1000);
}

dbData();

app.post('/sensor',function(req,res) {
    roomData = req.body.data;
    // res.send("Ok");
});
```

Middle server (PI_to_server.js)

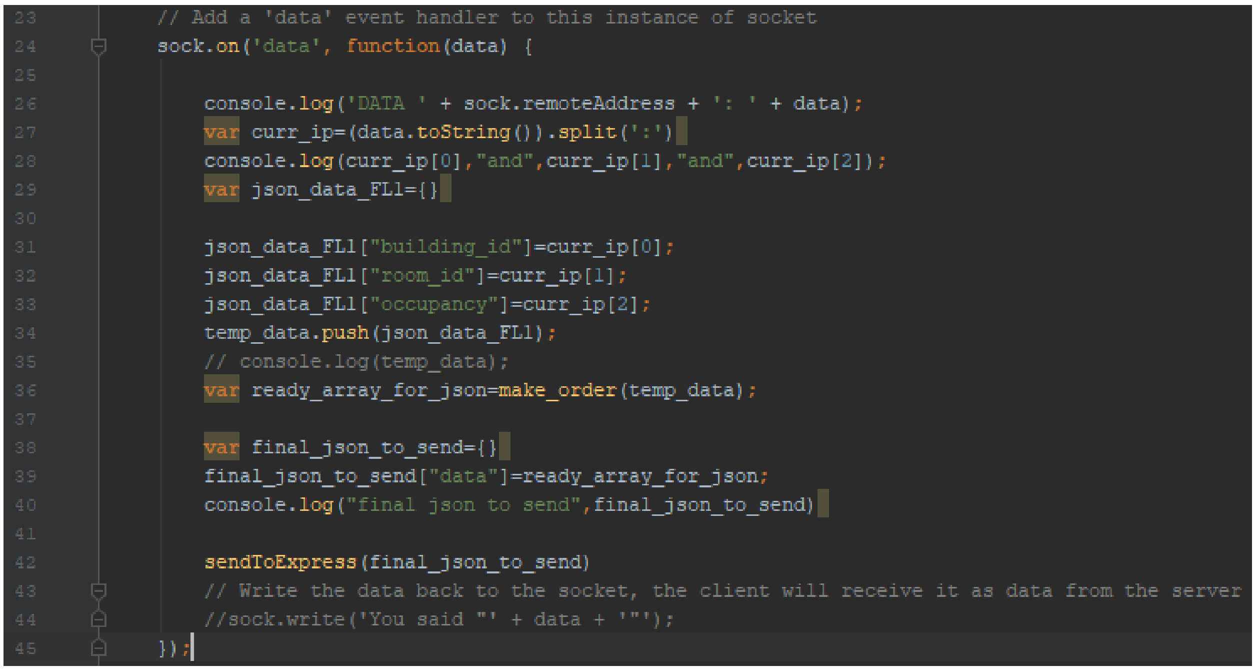
In this file we used two module named "net" and "node-fetch". Node.js net module is used to create both servers and clients. This module provides an asynchronous network wrapper and it can be imported. Using “net” we created this middleware server on the localhost IP(in this case IP 130.230.148.93 which changes depending on different network it is connected to) on port 3010.



```
var net = require('net');
var fetch = require("node-fetch");

var HOST = '130.230.148.93';
// var HOST ='130.230.144.243';

var PORT = 3010;
```

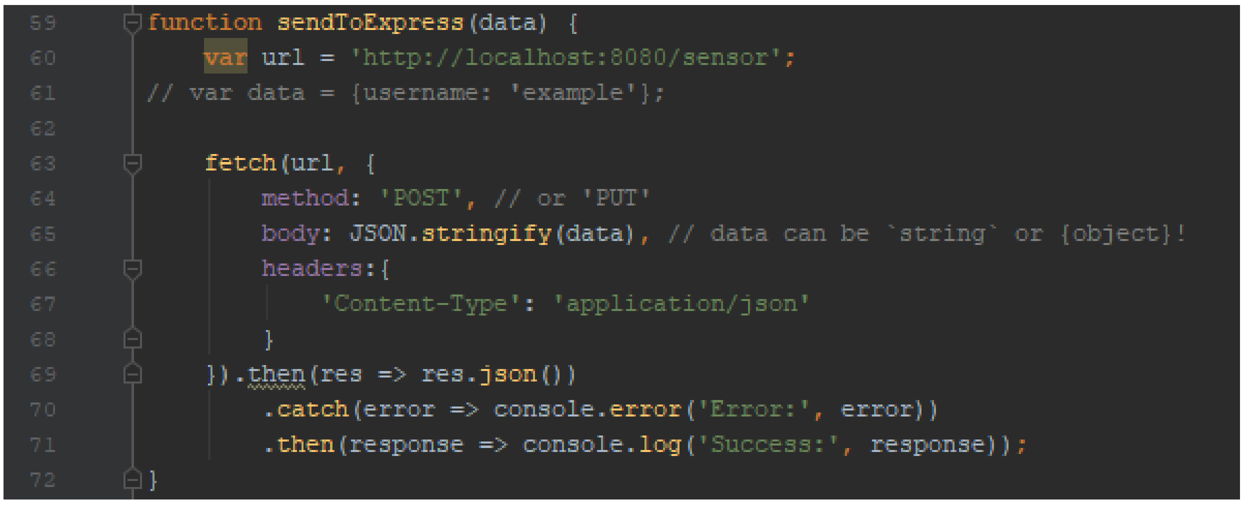
“net.createServer([options][, connectionListener])” Creates a new TCP server. The connectionListener argument is automatically set as a listener for the 'connection' event. In our program we used “net.createServer(function(sock)) to be connected with another socket client(in this case RPI that sends the sensor data to Pi_to_server.js). the “sock” object that passed inside is unique socket object created for individual connected socket client. Inside this function block we kept some function and variable. The first stored variable is an array of json formatted data. This json data is structured to split and format the received data from connected RPI server(which sends the data in string format).

The code in the following snippet is included inside server created by “net.createServer(…….)” and this is a socket connection. This specific socket is connected with the socket running in RPI server and receives the data sent by RPI server. The received data is in string format which is then split in parts(the parts represents building_id, room_id, occupance) and then inserted into a json variable. The variable is then pushed to the temp_data array. The temp_data array is then passed into a function named make_order() which returns an array of unique json data for all the rooms in different building. Then the data is inserted into a json variable named “final_json_to_sent” and sent to the main express server with a function “sendToExpress()”.



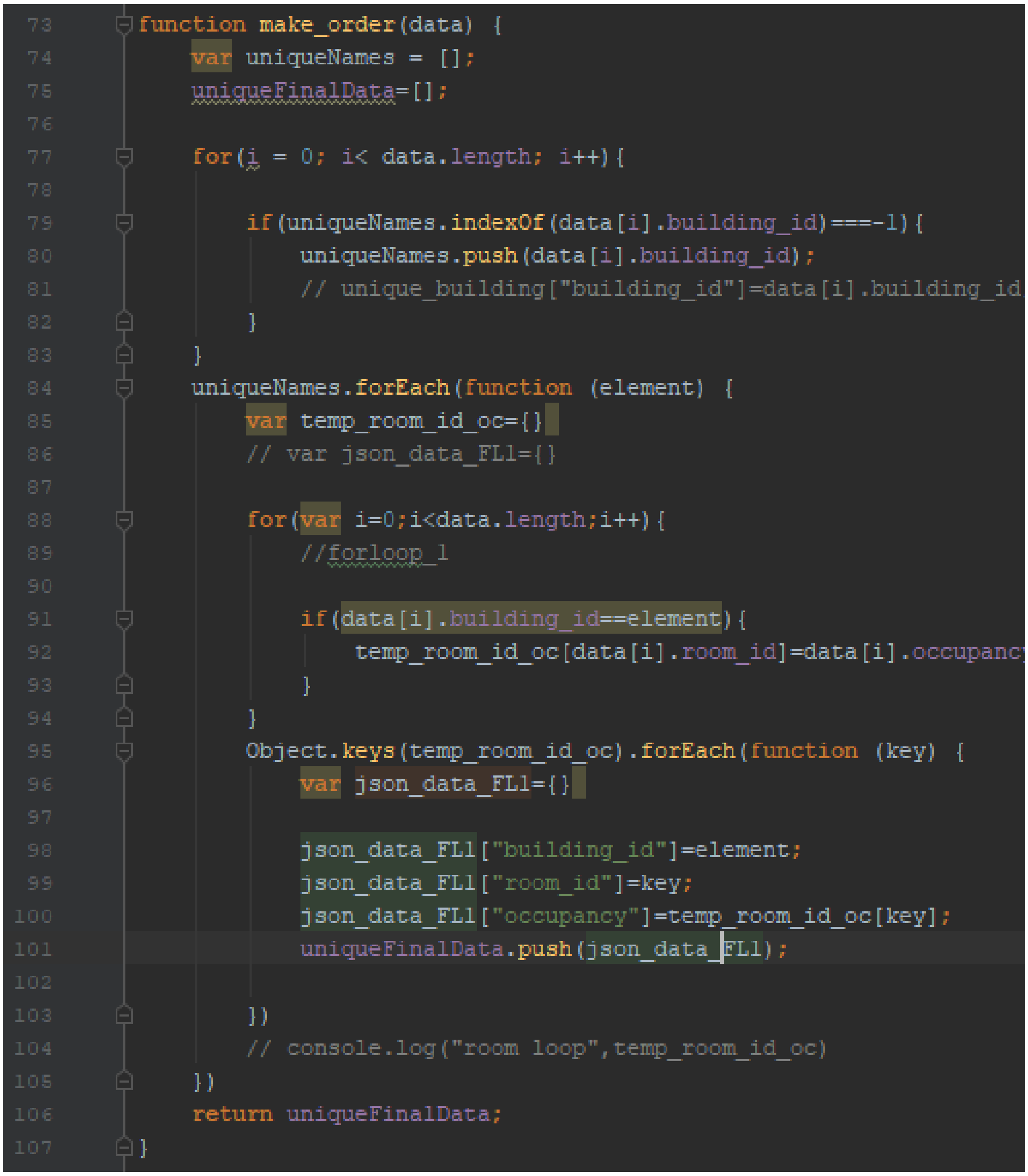
```
    // Add a 'data' event handler to this instance of socket
    sock.on('data', function(data) {

        console.log('DATA ' + sock.remoteAddress + ': ' + data);
        var curr_ip=(data.toString()).split(':')
        console.log(curr_ip[0],"and",curr_ip[1],"and",curr_ip[2]);
        var json_data_FL1={}

        json_data_FL1["building_id"]=curr_ip[0];
        json_data_FL1["room_id"]=curr_ip[1];
        json_data_FL1["occupancy"]=curr_ip[2];
        temp_data.push(json_data_FL1);
        // console.log(temp_data);
        var ready_array_for_json=make_order(temp_data);

        var final_json_to_send={}
        final_json_to_send["data"]=ready_array_for_json;
        console.log("final json to send",final_json_to_send)

        sendToExpress(final_json_to_send)
        // Write the data back to the socket, the client will receive it as data from the server
        //sock.write('You said "' + data + '"');
    });
```

sendToExpress(): This following snippet represents code that uses the “node-fetch” module to send data to specific url using http post method.



```
function sendToExpress(data) {
    var url = 'http://localhost:8080/sensor';
// var data = {username: 'example'};

    fetch(url, {
        method: 'POST', // or 'PUT'
        body: JSON.stringify(data), // data can be `string` or {object}!
        headers:{
            'Content-Type': 'application/json'
        }
    }).then(res => res.json())
        .catch(error => console.error('Error:', error))
        .then(response => console.log('Success:', response));
}
```

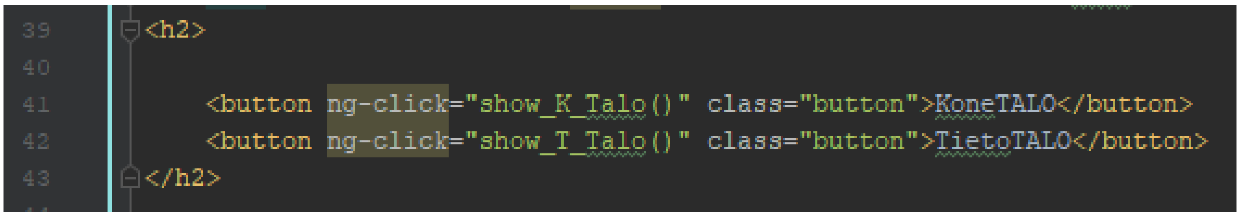
Make_order(): The following code takes array of json data as argument and remove the duplicate data in that array and returns a array of unique json data. Thus, we made sure that the data about the room status in every building is unique.



```
function make_order(data) {
    var uniqueNames = [];
    uniqueFinalData=[];

    for(i = 0; i< data.length; i++){

        if(uniqueNames.indexOf(data[i].building_id)===-1){
            uniqueNames.push(data[i].building_id);
            // unique_building["building_id"]=data[i].building_id
        }
    }
    uniqueNames.forEach(function (element) {
        var temp_room_id_oc={}
        // var json_data_FL1={}

        for(var i=0;i<data.length;i++){
            //forloop_1

            if(data[i].building_id==element){
                temp_room_id_oc[data[i].room_id]=data[i].occupanc
            }
        }
        Object.keys(temp_room_id_oc).forEach(function (key) {
            var json_data_FL1={}

            json_data_FL1["building_id"]=element;
            json_data_FL1["room_id"]=key;
            json_data_FL1["occupancy"]=temp_room_id_oc[key];
            uniqueFinalData.push(json_data_FL1);

        })
        // console.log("room loop",temp_room_id_oc)
    })
    return uniqueFinalData;
}
```

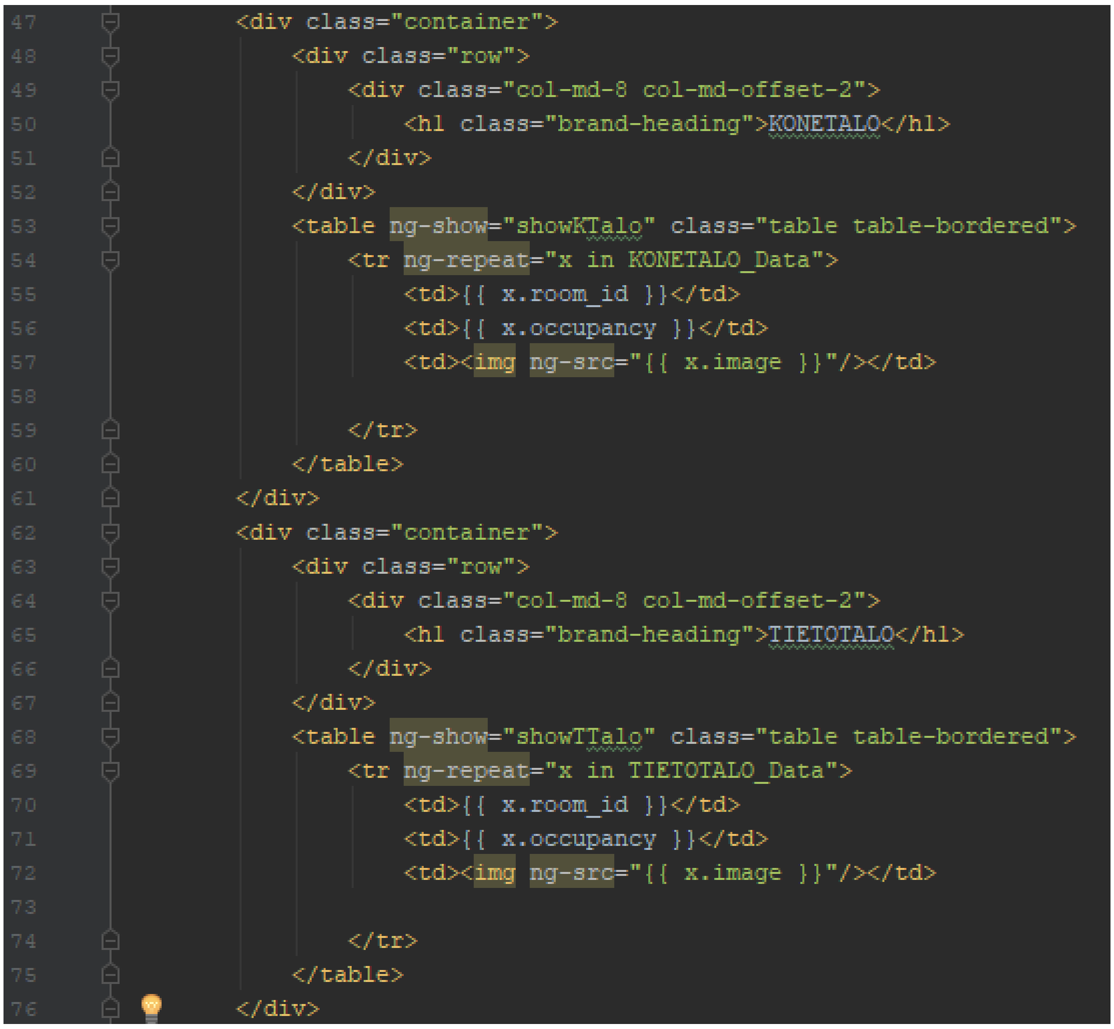
Index.ejs

This file contains all the modules that will show the data to the user. it contains technology and modules such as HTML, CSS reference, AngularJS, SocketIO, Bootstrap and plain JavaScript. The view contains three button and two data table. The two buttons represents two building named konetalo and tietotalo, where with button click the data of the corresponding building is displayed and other data tables are minimized. This button click function is controlled with AngularJS.



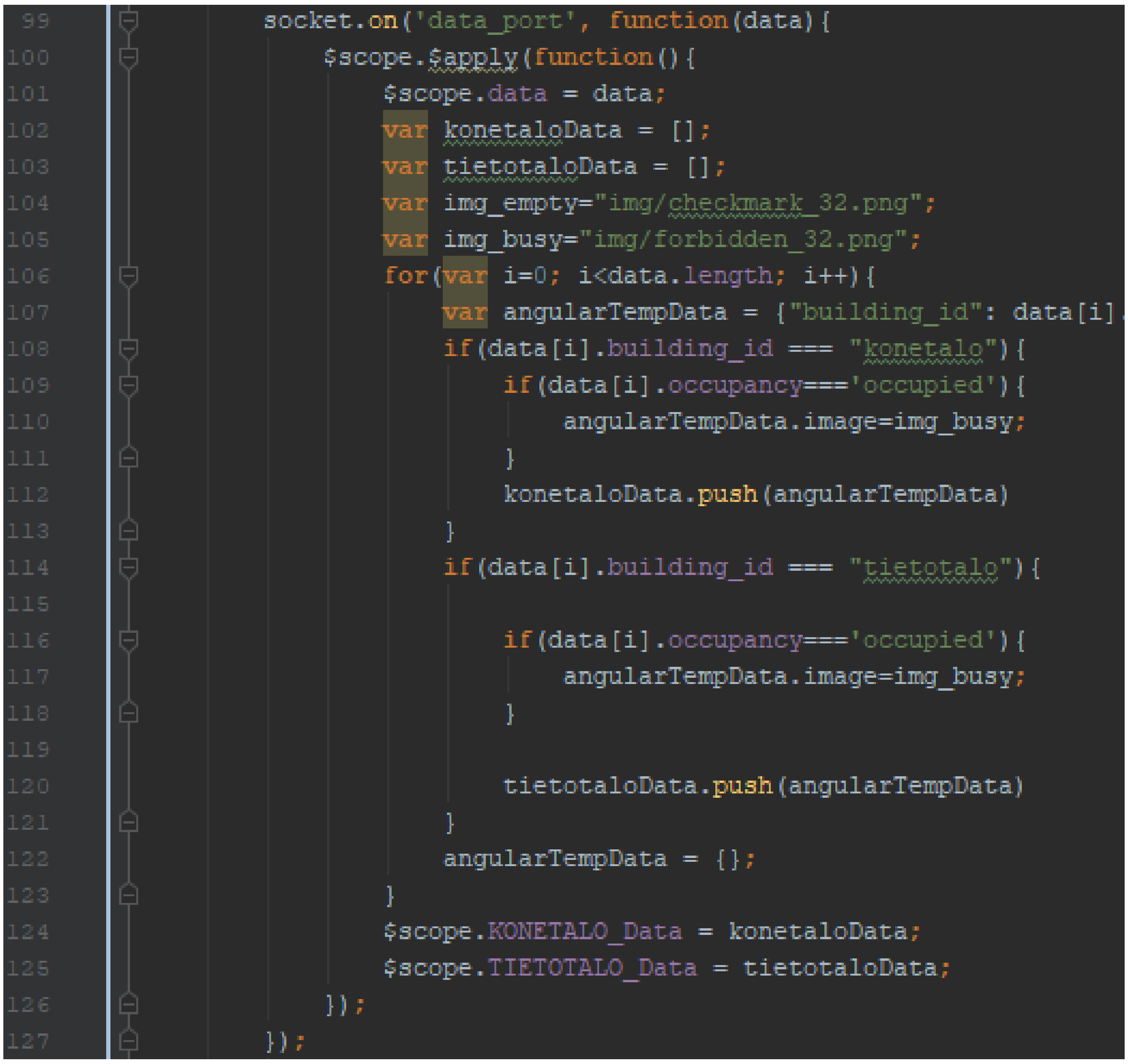
```
<h2>

    <button ng-click="show_K_Talo()" class="button">KoneTALO</button>
    <button ng-click="show_T_Talo()" class="button">TietoTALO</button>
</h2>
```

After button click the table data is displayed of minimized. The following snippet shows the code for the tables where the data fields are controlled by AngularJS. Corresponding data is stored in arrays and then displayed in tables. Where the left field shows room number, middle field shows status, the right field shows valid(green tick mark) or invalid (red dash) image depending on room status.



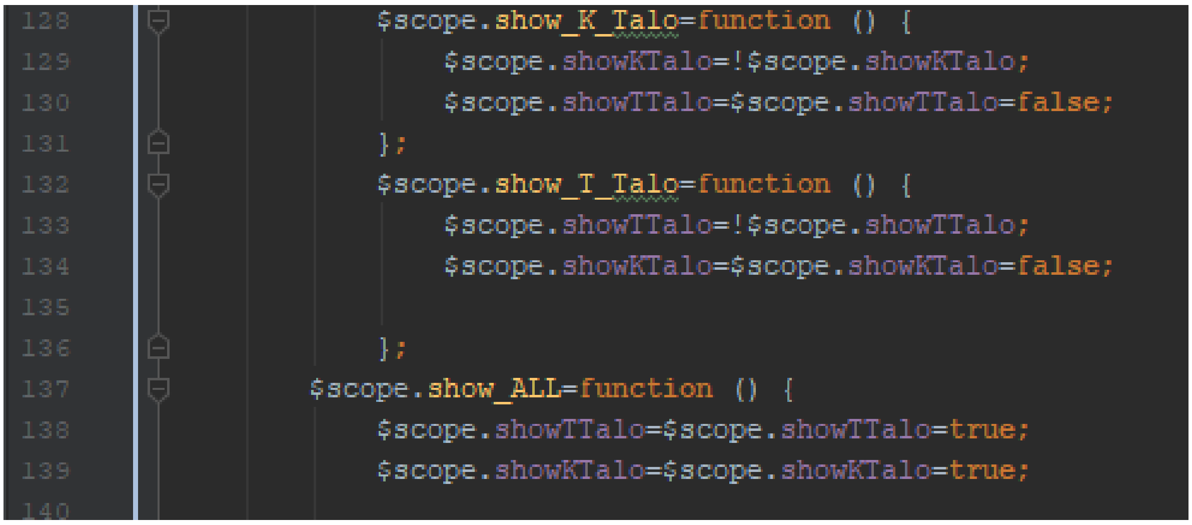
```
        <div class="container">
            <div class="row">
                <div class="col-md-8 col-md-offset-2">
                    <h1 class="brand-heading">KONETALO</h1>
                </div>
            </div>
            <table ng-show="showKTalo" class="table table-bordered">
                <tr ng-repeat="x in KONETALO_Data">
                    <td>{{ x.room_id }}</td>
                    <td>{{ x.occupancy }}</td>
                    <td><img ng-src="{{ x.image }}"/></td>

                </tr>
            </table>
        </div>
        <div class="container">
            <div class="row">
                <div class="col-md-8 col-md-offset-2">
                    <h1 class="brand-heading">TIETOTALO</h1>
                </div>
            </div>
            <table ng-show="showTTalo" class="table table-bordered">
                <tr ng-repeat="x in TIETOTALO_Data">
                    <td>{{ x.room_id }}</td>
                    <td>{{ x.occupancy }}</td>
                    <td><img ng-src="{{ x.image }}"/></td>

                </tr>
            </table>
        </div>
```

Inside socket the data is received from main server is inserted into two different array for two different building. Those array is used to display data in tables.



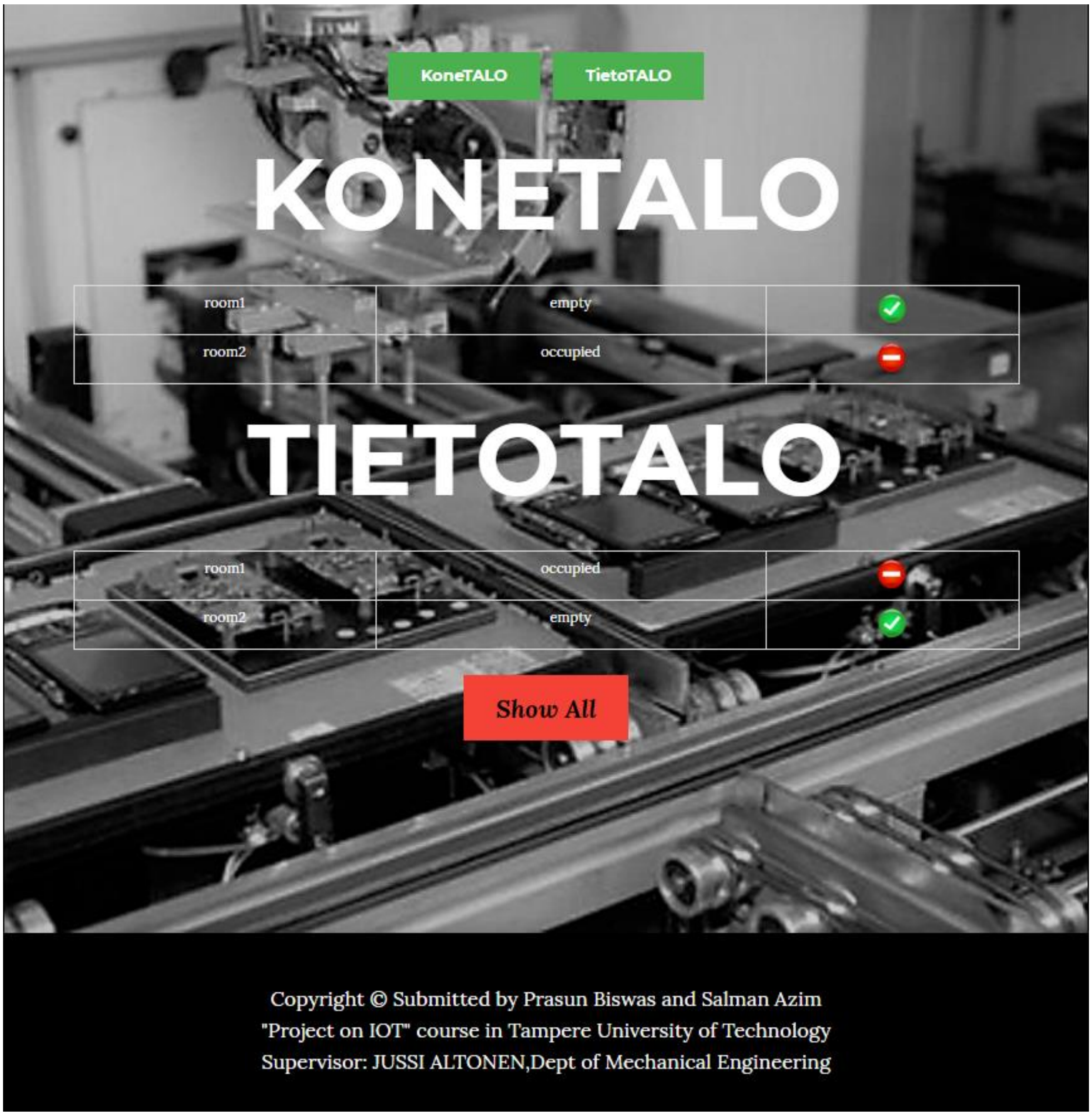
```
socket.on('data_port', function(data){
    $scope.$apply(function(){
        $scope.data = data;
        var konetaloData = [];
        var tietotaloData = [];
        var img_empty="img/checkmark_32.png";
        var img_busy="img/forbidden_32.png";
        for(var i=0; i<data.length; i++){
            var angularTempData = {"building_id": data[i]
            if(data[i].building_id === "konetalo"){
                if(data[i].occupancy==='occupied'){
                    angularTempData.image=img_busy;
                }
                konetaloData.push(angularTempData)
            }
            if(data[i].building_id === "tietotalo"){

                if(data[i].occupancy==='occupied'){
                    angularTempData.image=img_busy;
                }

                tietotaloData.push(angularTempData)
            }
            angularTempData = {};
        }
        $scope.KONETALO_Data = konetaloData;
        $scope.TIETOTALO_Data = tietotaloData;
    });
});
```

The following snippet contains code for all button control function. These functions are executed for all three buttons in the user interface.



```
    $scope.show_K_Talo=function () {
        $scope.showKTalo=!$scope.showKTalo;
        $scope.showTTalo=$scope.showTTalo=false;
    };
    $scope.show_T_Talo=function () {
        $scope.showTTalo=!$scope.showTTalo;
        $scope.showKTalo=$scope.showKTalo=false;

    };
$scope.show_ALL=function () {
    $scope.showTTalo=$scope.showTTalo=true;
    $scope.showKTalo=$scope.showKTalo=true;
```

An image of the user interface that shows the room status as tables is included below.



KONETALO

room1	empty	✓
room2	occupied	⛔

TIETOTALO

room1	occupied	⛔
room2	empty	✓

Copyright © Submitted by Prasun Biswas and Salman Azim
"Project on IOT" course in Tampere University of Technology
Supervisor: JUSSI ALTONEN,Dept of Mechanical Engineering

Fig4: User Interface for Room occupancy checking

Video Link

From the link below, the video of this project can be checked which we made.

https://photos.app.goo.gl/xB5S2zyYfYCAwwxCA

Further Development

This solution can be further developed for adding more useful features. A sign in page can be added with which students can sign in with their student ID and password. This solution can assure to usability only remains for the provided clients and make the solution secure.

The room booking system can also be added where clients can book any room for certain time from the system. For providing these two solutions we have to use database technology.

Conclusion

With the technological innovation, people are coming up with solutions that can help to do our work more efficiently. There are lots of take away from this project. First of all, this project led us to think how we can use technology to make the our easier. Secondly, we learned plenty of new technologies and used them hands on by working with them. Thirdly, it has helped us to brainstorm useful technological ideas which can be built to save our time to become more efficient every day. Lastly, we would like to thank our teacher **Professor Jussi Aaltonen** for giving us this opportunity to work for this project and completing the course.