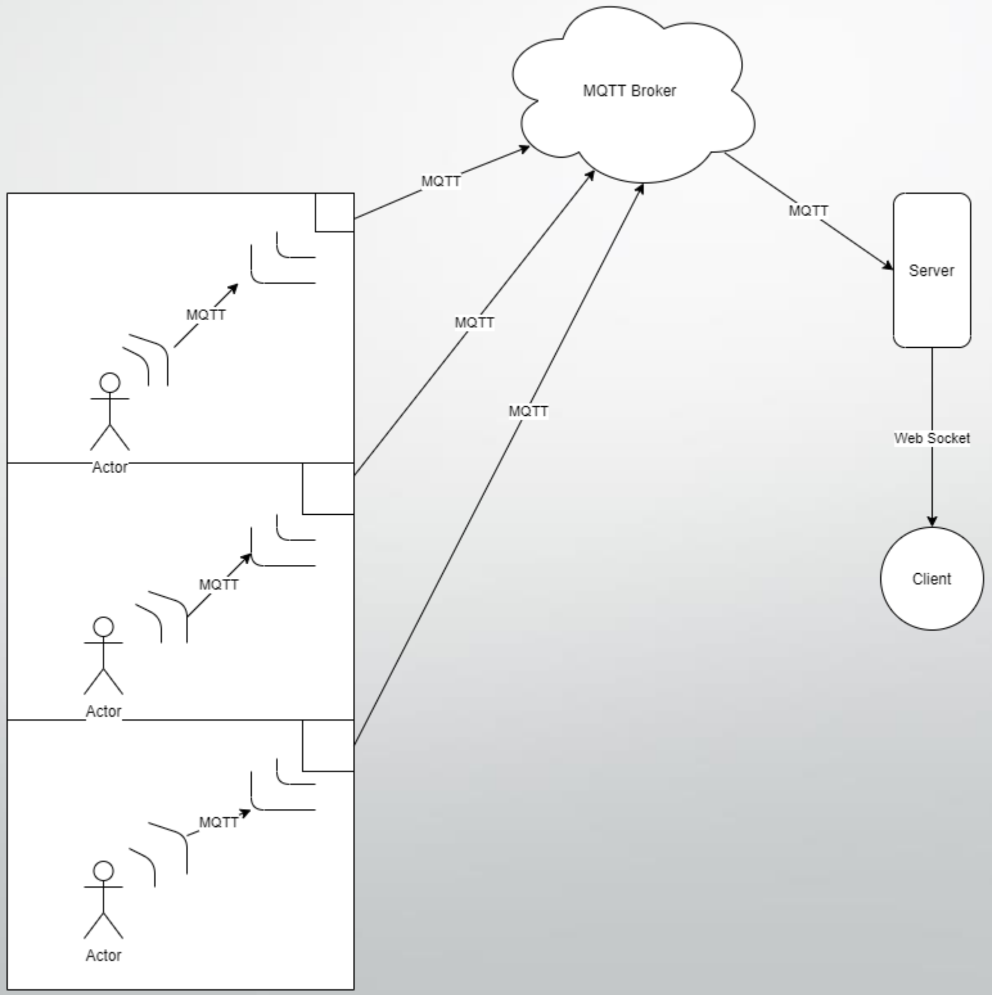
IoT Final project

Francesco Stucci

The architecture



Back-end

- **Emitter:** bluetooth device assigned to each employees
- **Scanner:** bluetooth scanner installed on each floor/room we need to monitor
- **Server:** server to handle the scanners data
  - **Database:** to store data collected from scanners

Protocol

All backend devices communicates among them via **MQTT** protocol

Framework

Server, scanners and emitters are implemented by using the **NodeJs** framework

DBMS

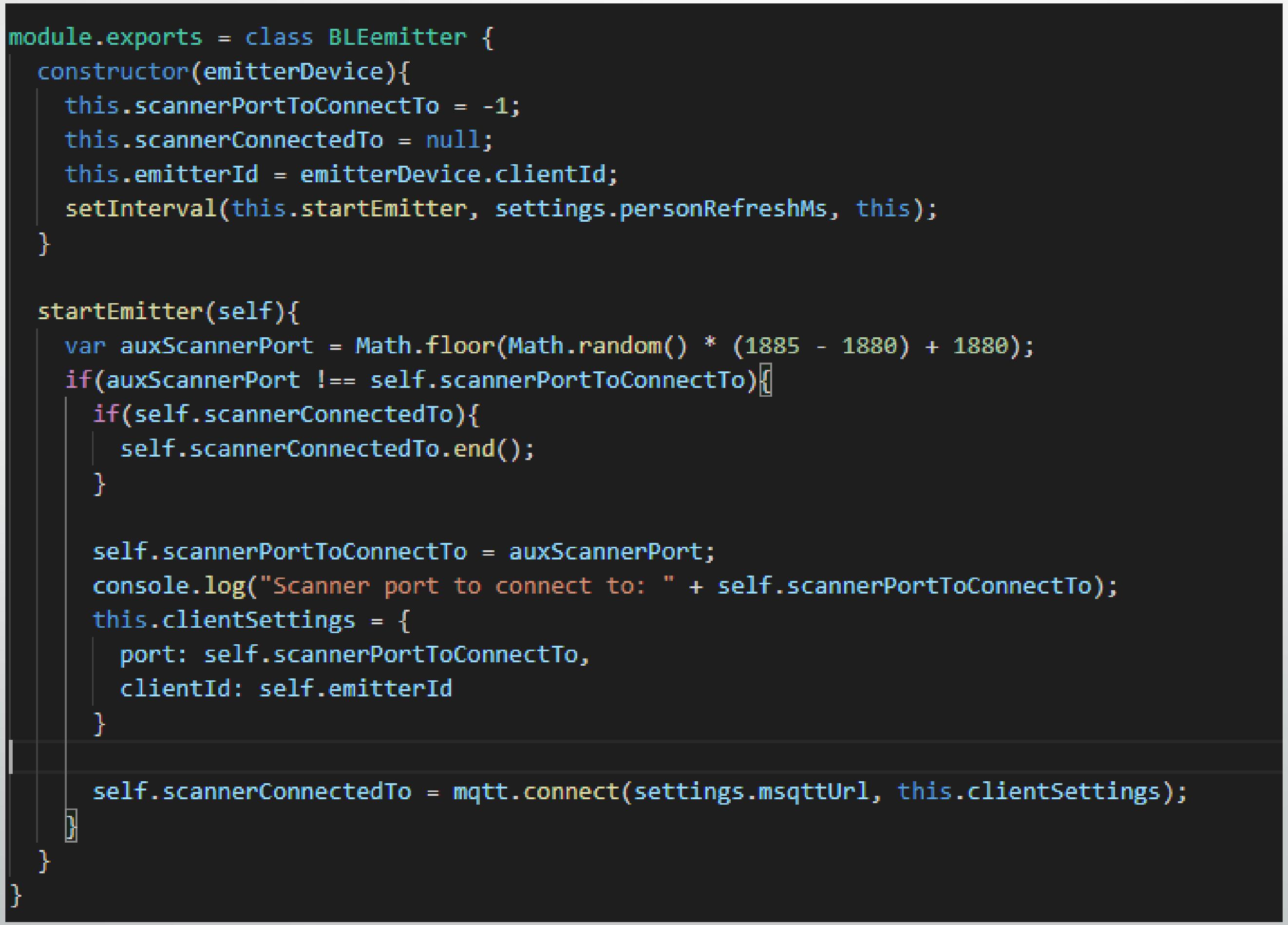
All data are stored with **MongoDB**, a NoSQL DBMS.

Stored data

<u>Data</u>

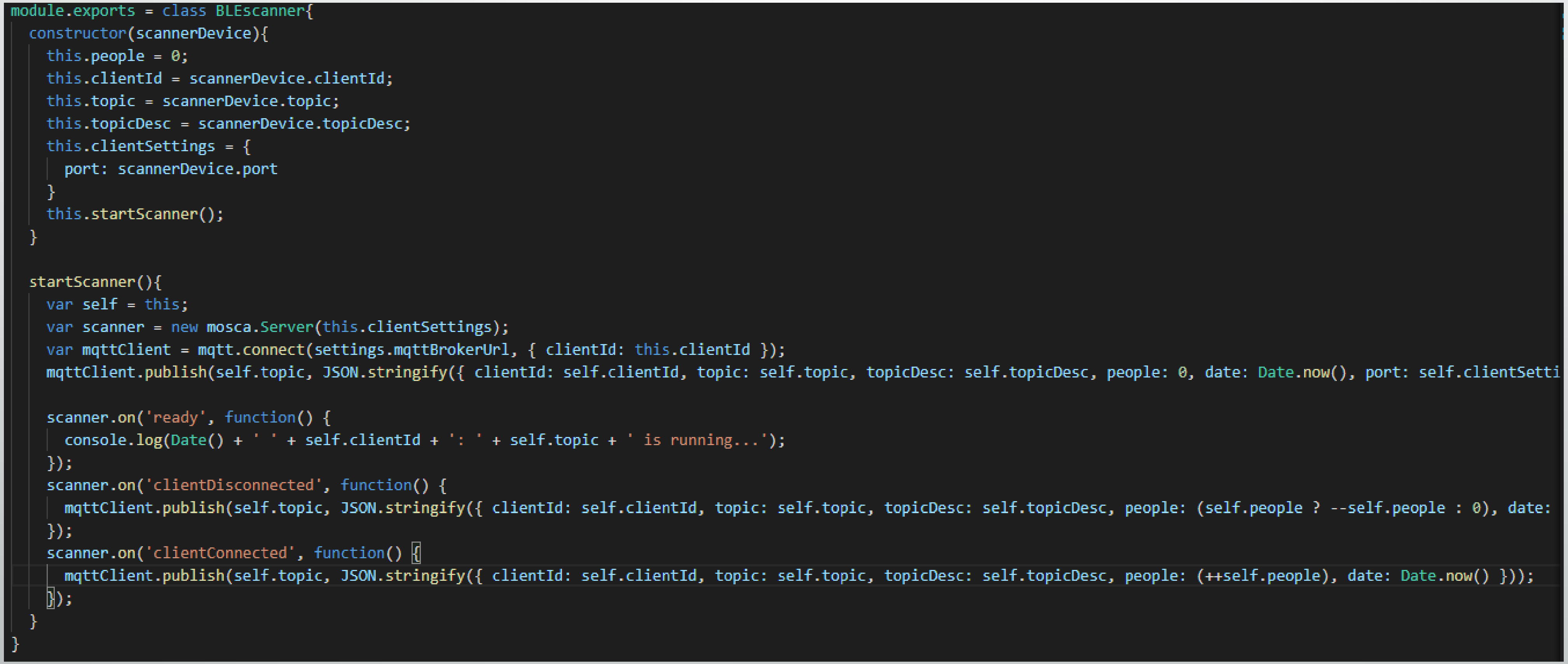
```
{
	clientId: int,
	topis: string,
	topicDesc: string,
	people: int,
	date: Date
}
```

Emitter



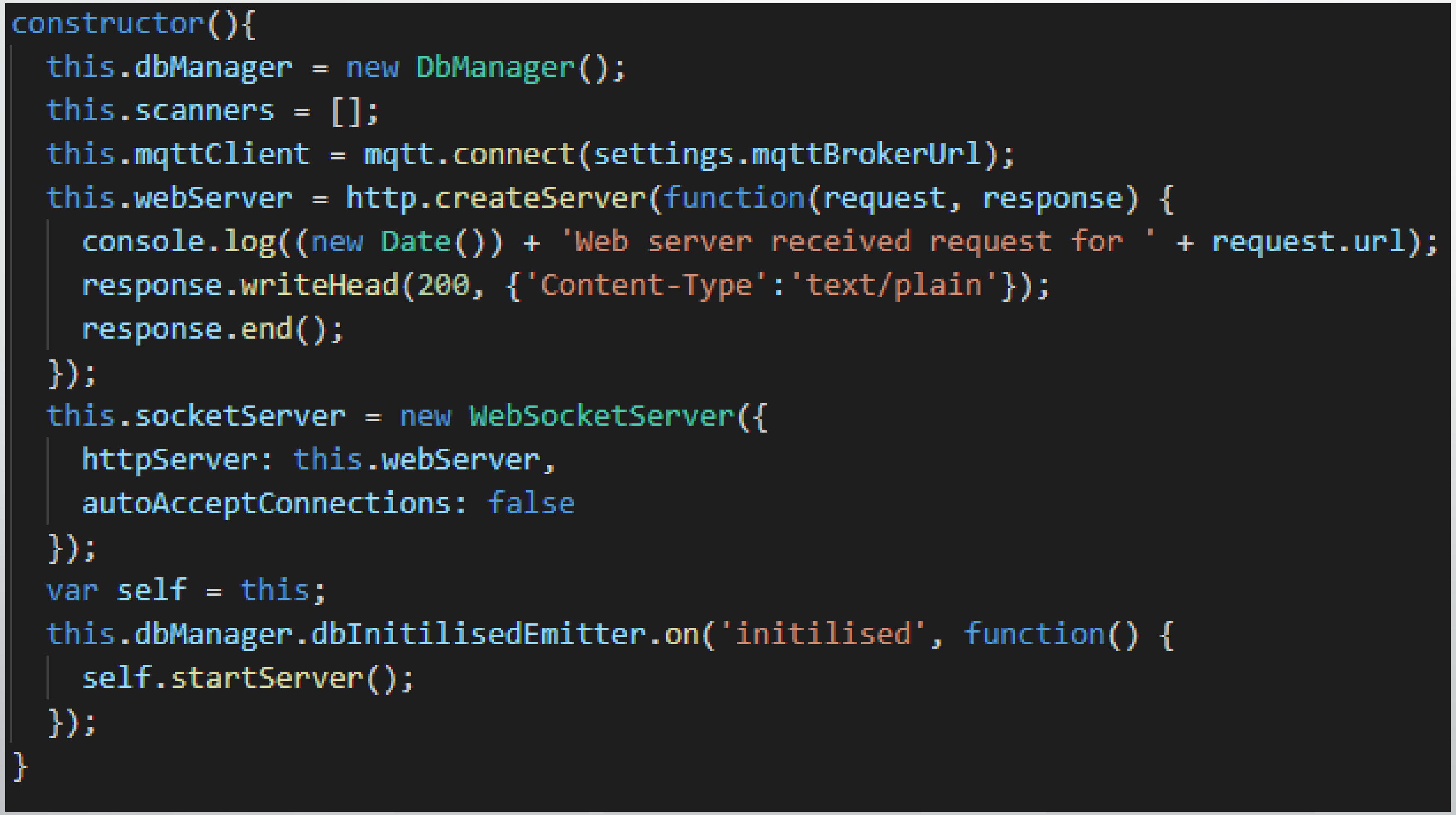
```
module.exports = class BLEemitter {
  constructor(emitterDevice){
    this.scannerPortToConnectTo = -1;
    this.scannerConnectedTo = null;
    this.emitterId = emitterDevice.clientId;
    setInterval(this.startEmitter, settings.personRefreshMs, this);
  }

  startEmitter(self){
    var auxScannerPort = Math.floor(Math.random() * (1885 - 1880) + 1880);
    if(auxScannerPort !== self.scannerPortToConnectTo){
      if(self.scannerConnectedTo){
        self.scannerConnectedTo.end();
      }

      self.scannerPortToConnectTo = auxScannerPort;
      console.log("Scanner port to connect to: " + self.scannerPortToConnectTo);
      this.clientSettings = {
        port: self.scannerPortToConnectTo,
        clientId: self.emitterId
      }

      self.scannerConnectedTo = mqtt.connect(settings.msqttUrl, this.clientSettings);
    }
  }
}
```

Scanner

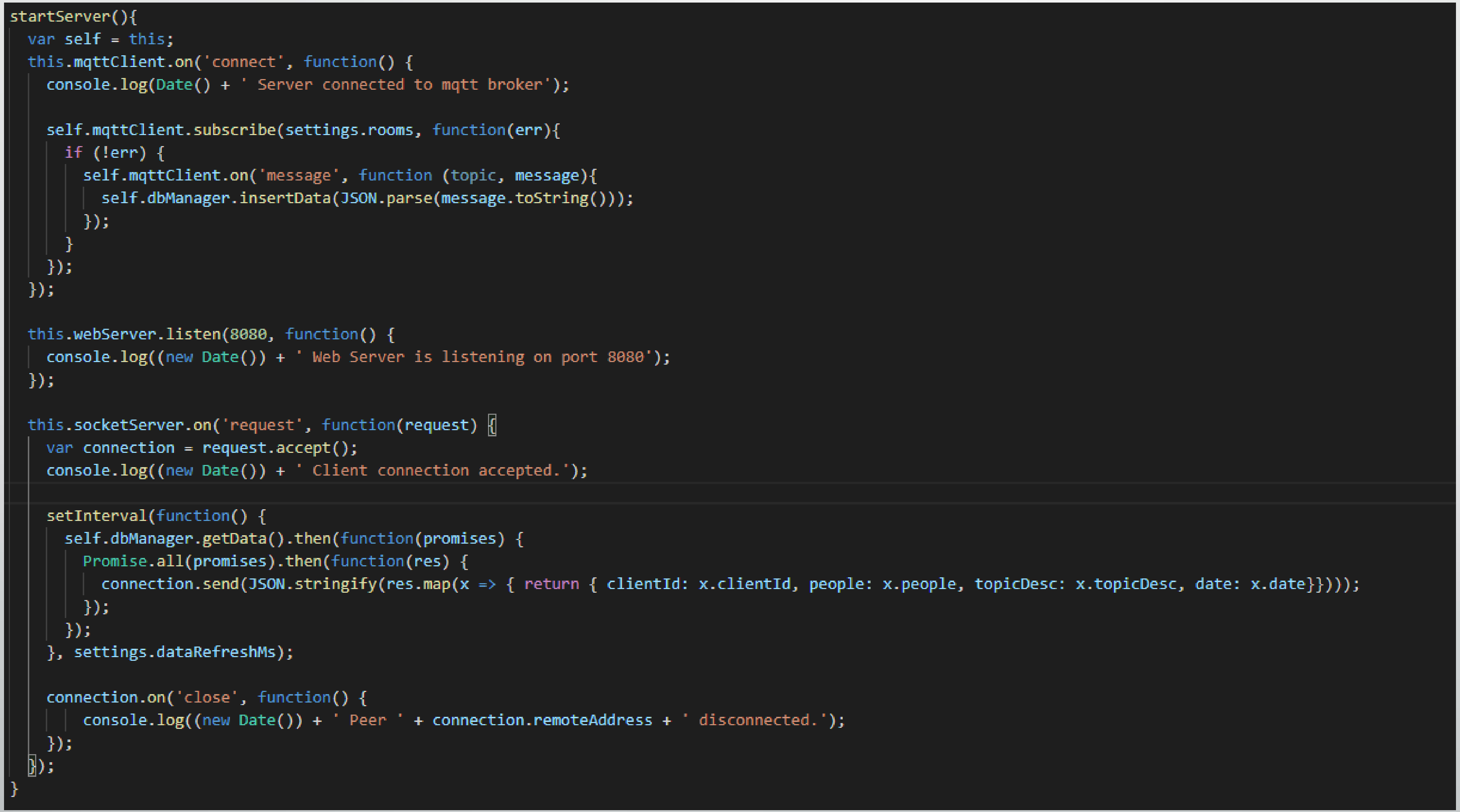


```
module.exports = class BLEscanner{
  constructor(scannerDevice){
    this.people = 0;
    this.clientId = scannerDevice.clientId;
    this.topic = scannerDevice.topic;
    this.topicDesc = scannerDevice.topicDesc;
    this.clientSettings = {
      port: scannerDevice.port
    }
    this.startScanner();
  }

  startScanner(){
    var self = this;
    var scanner = new mosca.Server(this.clientSettings);
    var mqttClient = mqtt.connect(settings.mqttBrokerUrl, { clientId: this.clientId });
    mqttClient.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic, topicDesc: self.topicDesc, people: 0, date: Date.now(), port: self.clientSetti

    scanner.on('ready', function() {
      console.log(Date() + ' ' + self.clientId + ': ' + self.topic + ' is running...');
    });
    scanner.on('clientDisconnected', function() {
      mqttClient.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic, topicDesc: self.topicDesc, people: (self.people ? --self.people : 0), date:
    });
    scanner.on('clientConnected', function() {
      mqttClient.publish(self.topic, JSON.stringify({ clientId: self.clientId, topic: self.topic, topicDesc: self.topicDesc, people: (++self.people), date: Date.now() }));
    });
  }
}
```

Server



```
constructor(){
  this.dbManager = new DbManager();
  this.scanners = [];
  this.mqttClient = mqtt.connect(settings.mqttBrokerUrl);
  this.webServer = http.createServer(function(request, response) {
    console.log((new Date()) + 'Web server received request for ' + request.url);
    response.writeHead(200, {'Content-Type':'text/plain'});
    response.end();
  });
  this.socketServer = new WebSocketServer({
    httpServer: this.webServer,
    autoAcceptConnections: false
  });
  var self = this;
  this.dbManager.dbInitilisedEmitter.on('initilised', function() {
    self.startServer();
  });
}
```



```
startServer(){
  var self = this;
  this.mqttClient.on('connect', function() {
    console.log(Date() + ' Server connected to mqtt broker');

    self.mqttClient.subscribe(settings.rooms, function(err){
      if (!err) {
        self.mqttClient.on('message', function (topic, message){
          self.dbManager.insertData(JSON.parse(message.toString()));
        });
      }
    });
  });

  this.webServer.listen(8080, function() {
    console.log((new Date()) + ' Web Server is listening on port 8080');
  });

  this.socketServer.on('request', function(request) {
    var connection = request.accept();
    console.log((new Date()) + ' Client connection accepted.');

    setInterval(function() {
      self.dbManager.getData().then(function(promises) {
        Promise.all(promises).then(function(res) {
          connection.send(JSON.stringify(res.map(x => { return { clientId: x.clientId, people: x.people, topicDesc: x.topicDesc, date: x.date}})));
        });
      });
    }, settings.dataRefreshMs);

    connection.on('close', function() {
        console.log((new Date()) + ' Peer ' + connection.remoteAddress + ' disconnected.');
    });
  });
}
```

Front-end

- **Client:** remote desktop pc which is supposed to monitor the building

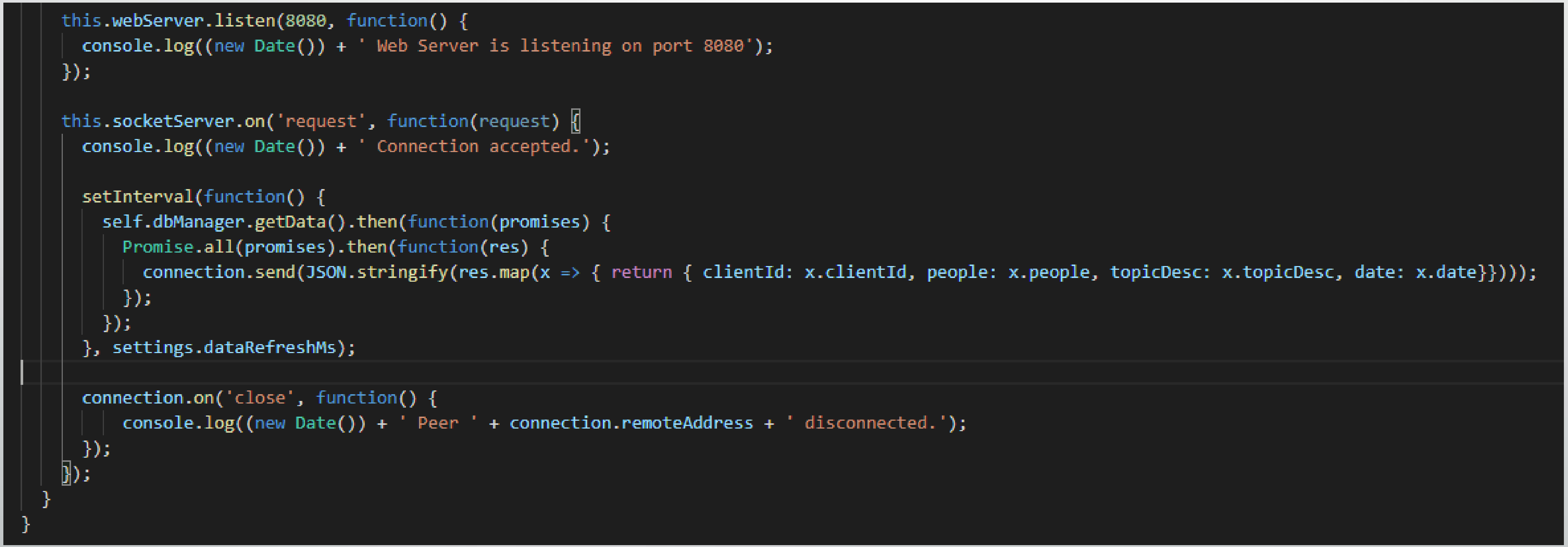
Protocol

Client and server transfer data through **WebSocket** protocol

Framework

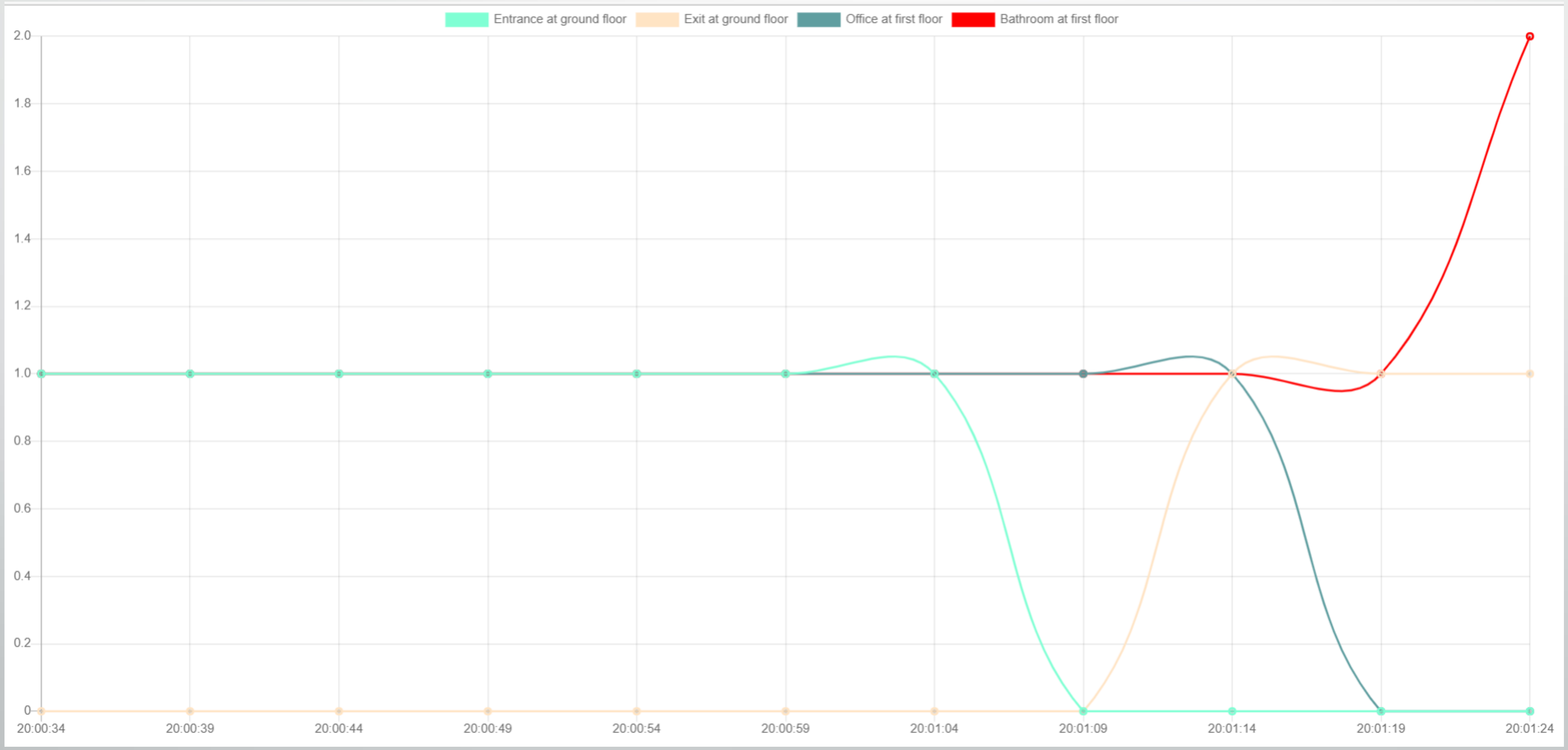
The implementation of the frontend has been developed by using **Angular** framework

Server



```
    this.webServer.listen(8080, function() {
      console.log((new Date()) + ' Web Server is listening on port 8080');
    });

    this.socketServer.on('request', function(request) {
      console.log((new Date()) + ' Connection accepted.');

      setInterval(function() {
        self.dbManager.getData().then(function(promises) {
          Promise.all(promises).then(function(res) {
            connection.send(JSON.stringify(res.map(x => { return { clientId: x.clientId, people: x.people, topicDesc: x.topicDesc, date: x.date}})));
          });
        });
      }, settings.dataRefreshMs);

      connection.on('close', function() {
          console.log((new Date()) + ' Peer ' + connection.remoteAddress + ' disconnected.');
      });
    });
  }
}
```

Client


Entrance at ground floor
Exit at ground floor
Office at first floor
Bathroom at first floor
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
20:00:34
20:00:39
20:00:44
20:00:49
20:00:54
20:00:59
20:01:04
20:01:09
20:01:14
20:01:19
20:01:24