

# Assignment 2 Report - IDG2001

By students:

Navid Amiri,  
E-mail: [navida@stud.ntnu.no](mailto:navida@stud.ntnu.no) ,  
Student Number: 523993

and

Sivan Sabir Mahmud,  
E-mail: [sivansm@stud.ntnu.no](mailto:sivansm@stud.ntnu.no),  
Student Number: 542768

Heroku Link (not working – has 503 error – explanation within the report):

<https://assignment2-idg2001-group2.herokuapp.com/>

## 1.0 - Office Scenario

### Office temperature and humidity reader – “Smart Office”

A manager in a medium sized company is often getting complaints that the office is too cold or too warm. This problem has been going on for months and she would like to solve it somehow.

For the moment, she is measuring the office temperature based on the complaints and adjusting the temperature manually. The manager is aware that there is also humidity, as a second variable, that should be addressed if she really wants to solve the problem in the office. Based on her research, she is finding that a humidity level between 20 and 60 is best suited for office environment.

The problem that the manager is facing is not only time consuming but also stressful. She is often disturbed during her work and must check the temperature and fix it, often several times during a day. The problem is also decreasing productivity among the employees as they are not in a right temperature during their work-day, which is either too warm or too cold.

The solution for her problem is to get hold on sensors that not only checks the temperature- but also the humidity level. These can be placed on different sides of the office and hold the temperature fixed between a set of values. For example, that the temperature of the different rooms should be between 19 and 26 degrees Celsius, which according to research is the most suitable for an office environment.

Having the temperature and humidity level displayed on the screen would also enable all the employees and managers at the office to see it at any given time.

## 2.0 – MQTT JS

In the publisher(.js), we managed to generate random data with the help of the `math.random` function that was placed inside of our `getRandomInt` function. It takes two parameters: `min` and `max`. Illustrated below.

```
//using math.random to generate data every time it loops through the code
// Function from: https://www.geeksforgeeks.org/how-to-generate-random-number-in-given-range-using-javascript/
function getRandomInt(min, max) {
  return Math.round(Math.random() * (max - min) + min);
}
```

*Figure 1 – screenshot of part of the publisher.js*

With the help of our two parameters, we could set different values to help us with the creation of temperature and humidity data. For example, the ideal temperature was set between 19 and 26 (measured in Celsius). The ideal humidity, on the other hand, was set between 20 and 60 (measured in percentage) as it is the best humidity level for office environment.

The data is then set inside of the “setInterval function” that goes in loop every 10 seconds. Illustrated below.

```
//after the connection to the broker is created, the publisher is sending a message every 10sec - source: https://www.youtube.com/watch?v=8NgIdT\_0Bc&ab\_channel=LintangWisesa
client.on('connect', () => {
  setInterval(function() {
    //inspired by: https://localcoder.org/local-broker-mqtt-based-publish-subscribe-using-android-application
    //garbage data
    let message = {
      "sensorName": "sensor1",
      "sensorUnit1": "Cel",
      "tempValue": getRandomInt(19, 26), //random number between 19 and 26
      "sensorUnit2": "Percent",
      "humidityValue": getRandomInt(20, 60), //random percentage between 20 and 60
      "time": Date.now(),
    }
    message = JSON.stringify(message)
    client.publish(topic, message)
    console.log("message sent!", message)
  }, 10000) //prints this message with interval 10000ms (10sec.)
}
```

Figure 2– screenshot of publisher.js

With the help of topic, our broker recognizes the data, parsing the data and storing it inside of our mongoDB collection: “sensordatas”.

```
44 //message published
45 aedes.on('publish', async function (packet, client) {
46   console.log('Client \x1b[31m' + (client ? client.id : 'BROKER_' + aedes.id) + '\x1b[0m has published', packet.payload.toString(), 'on', packet.topic, 'to broker', aedes.id)
47   var message = packet.payload.toString()
48   //transforming to object before sending to db
49   var parseMessage = JSON.parse(message)
50   mongoose.connect(mongoURL, { useNewUrlParser: true, useUnifiedTopology: true })
51   const sensor = new SensorModel({
52     sensorName: parseMessage.sensorName,
53     sensorUnit1: parseMessage.sensorUnit1,
54     tempValue: parseMessage.tempValue,
55     sensorUnit2: parseMessage.sensorUnit2,
56     humidityValue: parseMessage.humidityValue,
57     time: parseMessage.time,
58   })
59   await sensor.save()
60   console.log("data saved to mongodb")
61 })
62
```

Figure 3 – screenshot of broker.js

Result of running publisher.js (running locally):

```
message sent! {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":25,"sensorUnit2":"Percent","humidityValue":58,"time":1652127593834}
message sent! {"sensorName":"sensor1","sensorUnit1":"Cel","tempValue":22,"sensorUnit2":"Percent","humidityValue":36,"time":1652127683829}
message sent! {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":23,"sensorUnit2":"Percent","humidityValue":24,"time":1652127683834}
message sent! {"sensorName":"sensor1","sensorUnit1":"Cel","tempValue":23,"sensorUnit2":"Percent","humidityValue":28,"time":1652127613829}
message sent! {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":24,"sensorUnit2":"Percent","humidityValue":28,"time":1652127613834}
message sent! {"sensorName":"sensor1","sensorUnit1":"Cel","tempValue":23,"sensorUnit2":"Percent","humidityValue":29,"time":1652127623830}
message sent! {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":19,"sensorUnit2":"Percent","humidityValue":24,"time":1652127623834}
message sent! {"sensorName":"sensor1","sensorUnit1":"Cel","tempValue":24,"sensorUnit2":"Percent","humidityValue":54,"time":1652127633830}
message sent! {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":19,"sensorUnit2":"Percent","humidityValue":48,"time":1652127633834}
message sent! {"sensorName":"sensor1","sensorUnit1":"Cel","tempValue":20,"sensorUnit2":"Percent","humidityValue":52,"time":1652127643832}
message sent! {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":20,"sensorUnit2":"Percent","humidityValue":29,"time":1652127643835}
message sent! {"sensorName":"sensor1","sensorUnit1":"Cel","tempValue":24,"sensorUnit2":"Percent","humidityValue":41,"time":1652127653833}
message sent! {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":21,"sensorUnit2":"Percent","humidityValue":34,"time":1652127653835}
message sent! {"sensorName":"sensor1","sensorUnit1":"Cel","tempValue":21,"sensorUnit2":"Percent","humidityValue":25,"time":1652127663834}
message sent! {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":19,"sensorUnit2":"Percent","humidityValue":35,"time":1652127663835}
```

Figure 4– screenshot of publisher terminal

Result of running broker.js (running locally):



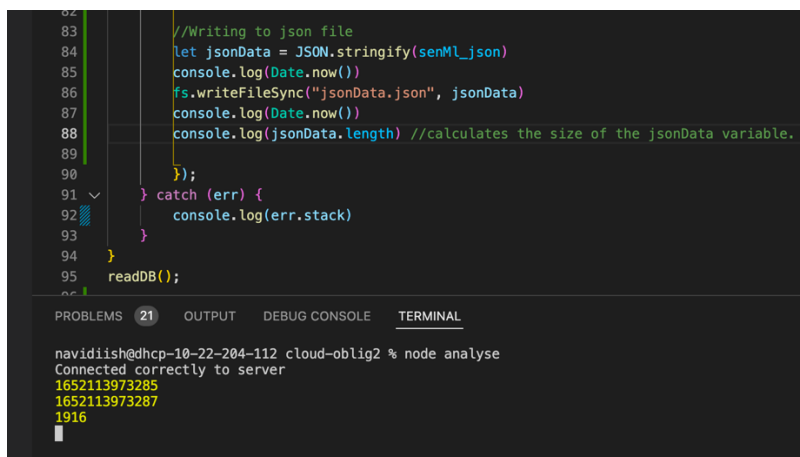
```
PROBLEMS 30 OUTPUT DEBUG CONSOLE TERMINAL
to broker 1e368598-608e-405a-beba-b32735ecdf8
data saved to mongodb
Client mqtt_569c487638a9 has published {"sensorName":"sensor1","sensorUnit1":"Cel","tempValue":24,"sensorUnit2":"Percent","humidityValue":54,"time":1652127633830} on office
to broker 1e368598-608e-405a-beba-b32735ecdf8
Client mqtt_569c487638a9 has published {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":19,"sensorUnit2":"Percent","humidityValue":48,"time":1652127633834} on office
to broker 1e368598-608e-405a-beba-b32735ecdf8
data saved to mongodb
Client mqtt_569c487638a9 has published {"sensorName":"sensor1","sensorUnit1":"Cel","tempValue":28,"sensorUnit2":"Percent","humidityValue":52,"time":1652127643832} on office
to broker 1e368598-608e-405a-beba-b32735ecdf8
Client mqtt_569c487638a9 has published {"sensorName":"sensor2","sensorUnit1":"Cel","tempValue":20,"sensorUnit2":"Percent","humidityValue":29,"time":1652127643835} on office
data saved to mongodb
data saved to mongodb
[]
```

Figure 5 – screenshot of broker terminal

## 3.0 - Encoding

Encoding to JSON, XML and CBOR done in the file “analyse.js”.

### 3.1 - Encoding to JSON file



```
82 //Writing to json file
83 let jsonData = JSON.stringify(senML_json)
84 console.log(Date.now())
85 fs.writeFileSync("jsonData.json", jsonData)
86 console.log(Date.now())
87 console.log(jsonData.length) //calculates the size of the jsonData variable.
88 });
89 } catch (err) {
90 console.log(err.stack)
91 }
92 readDB();
93
PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL
navidiish@dhcp-10-22-204-112 cloud-oblig2 % node analyse
Connected correctly to server
1652113973285
1652113973287
1916
```

Figure 6 – screenshot above illustrates our solution when encoding to JSON file

Date.now() shows the time since 1970 in ms. When console logged just before (1652113973285) and after (1652113973287) the “fs.writeFileSync(“jsonData.json”, jsonData)”, it measures the time it takes to encode the data to JSON file.

When calculating these two numbers we are getting that:

$$1652113973287\text{ms} - 1652113973285\text{ms} = 2\text{ms}$$

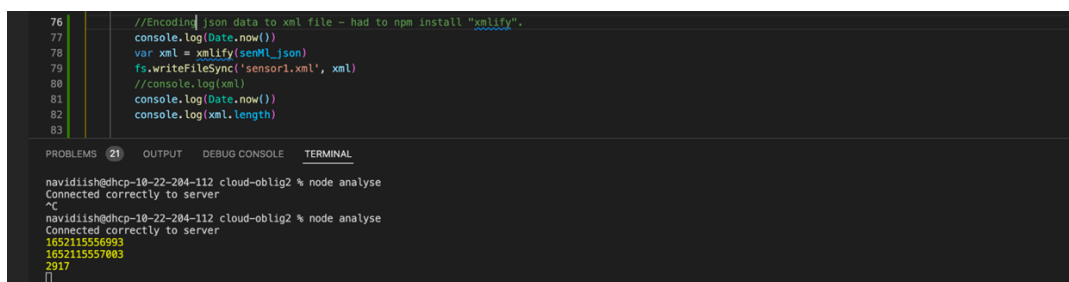
This means that the JSON file was created in 2ms.

Furthermore, the data is 1916 in length. We found this by taking the length of the stringified JSON data and console logging everything: `console.log(jsonData.length)`.

The data that was created is stored within a file called “jsonData.json” which is stored within our project.

### 3.2 - Encoding to XML file

Although the case description encouraged us to use another sensor to encode our data to XML file, due to lack of time, we had to use the same sensor (sensor1) to encode our data to an XML file. The result of our test and the method we used is displayed in our screenshot below.



```
76 //Encoding json data to xml file - had to npm install "xmlify".
77 console.log(Date.now())
78 var xml = xmlify(sensor1_json)
79 fs.writeFileSync('sensor1.xml', xml)
80 //console.log(xml)
81 console.log(Date.now())
82 console.log(xml.length)
83
```

PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL

```
navidiish@dhcp-10-22-204-112 cloud-oblig2 % node analyse
Connected correctly to server
^C
navidiish@dhcp-10-22-204-112 cloud-oblig2 % node analyse
Connected correctly to server
1652115556993
1652115557003
2917
{}

```

Figure 7 – Screenshot of the XML encoding

As the previous section, the Date.now() function has been used here by following the same principles. We are also measuring the file size using the same method. It takes 10ms (1652115557003- 1652115556993) to encode the XML format. The XML format has also 2917 in length. This was calculated by console logging “xml.length”.

### 3.3 - Encoding to CBOR file

As mentioned under “Encoding to JSON file”, the Date.now() function has been used here by following the same principles. We are also measuring the file size using the same method. It takes 8ms (1652116988464-1652116988456) to encode the CBOR format. The CBOR format has 1149 in length. This was calculated by console logging “serialisedBuffer.length”.

```
62 //speed of encoding data into cbor starts
63 console.log(Date.now())
64 //cbor senml
65 let serialisedBuffer = cbor.encode(senML_json)
66 fs.writeFileSync('cbor_senml.cbor', serialisedBuffer)
67 //speed of encoding data into cbor end
68 console.log(Date.now())
69 console.log(serialisedBuffer.length)
70
```

PROBLEMS 21 OUTPUT DEBUG CONSOLE TERMINAL

```
navidiish@dhcp-10-22-204-112 cloud-oblig2 % node analyse
Connected correctly to server
1652116988456
1652116988464
1149
```

Figure 8 – Screenshot of the CBOR encoding

### 3.4 - Conclusion:

JSON compared to XML is both faster and is smaller in length, which leads to a smaller file size. CBOR on the other hand is smaller than both XML and JSON, but when encoding, it is slower than JSON by 6ms and faster than XML by 4ms.

## 4.0 – display in frontend

We managed to display data in the frontend by creating a React app. The result is visible below:

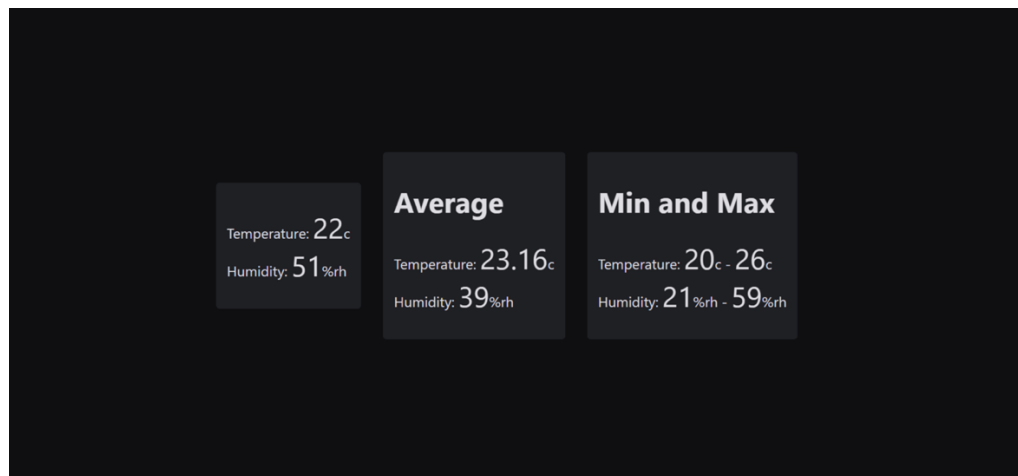


Figure 9 - Screenshot of our frontend (what client sees)

In our example here, we are fetching data that was stored on MongoDB sent by our publisher through our MQTT broker.

We set up a node.js server with express and made two routes that connected to the database. One for getting all the data in the database, and the other for getting highest and lowest value of each unit. Then we set up React to display 3 components we made. One for getting the current temperature and humidity, one for getting the average, and one for getting the highest and lowest values.

## 5.0 – Deployment

We did not manage to get the deployment right. The plan was to deploy the MQTT broker and publisher on its own server and Express (created with NODE JS) on its own server. And then having the react app that is deployed with the Express server fetching data from the Express server that queries data from MongoDB. We did not manage to finalize the plan as we ran into issues. Due to lack of time, we had to go on finishing the report. On our Heroku link provided, there is only the Express server.

### Sources:

We have tried to deliver all the sources used throughout our code.