# Report for Assignment 2

**Group members:**
Daniel Björk
daniepbj@stud.ntnu.no
548137

Ole Ragnar Randen
olerr@stud.ntnu.no
539663

**Heroku Link:**
https://cloud-oblig-2-mqtt.herokuapp.com/

**Github repo:**
https://github.com/oleragnar-boop/Cloud-assignment-2

***Explanation of the selected IoT scenario, and what type of data used.***
We decided to choose a smart greenhouse as our Iot Scenario. By simulating sensors to read humidity, temperature and light we can determine if the data matches optimal growing conditions for a specific plant type. In this iteration of the project we decided to hard code values for an optimal span of humidity, temperature and light for tomatoes. In future iterations outside of this class it could be expanded into a wider set of sensors such as when to water the plants and simulation of climate cycles. The data used is first sent as raw data from each sensor, to later be parsed as json before sending it to the database

**Explanations of how the system functions**
The system consists of multiple parts and we decided to explain each part by going through the data flow from sensor to the data presenter page. Explanations of code and the code snippets contains pseudocode / snippets for demonstration purposes. Please see code on github for actual functionality. The data analyzer and the sensors must be run locally for the system to work.

**Javascript files simulates sensor data and sends it to the broker**

- A set of javascript files that generates JSON data.
    - humidity_sensor.js
    - light_sensor.js
    - temperature_sensor.js

The first part consists of three sensors simulated as independent javascript files generating random data according to each sensor. First it connects to the broker as a client, it then sends data to the topic it's providing data to. Below is an example of how temperature is generated in **temperature_sensor.js**

```
const generateTemperatureData = () => {
    min = Math.ceil(35);
    max = Math.floor(0);
    let = TemperatureData = [{
        name: client_name,
        time: Date.now(),
        temperature: Math.floor(Math.random() * (max - min) + min)
    }]
    payload = JSON.stringify(TemperatureData)
}
```

Function: connectToBroker

This function creates the topic, connects to the broker with said topic and publishes on an interval.

```
const connectToBroker = () => {

    const client = mqtt.connect(connectUrl, {
(...)
        })
    const topic = 'humidity';
    client.on('connect', () => {
        console.log('Successfully connected to the broker')
        //publishing on a set interval, uses global variable payload
          setInterval(() => {
            client.publish(topic, payload, { qos: 0, retain: false },
(error) => {
                if (error) {
                    console.error(error)
                } else {
                    console.log(payload)
                }
```

Verify_client() checks to see if a client's name and password is in the MongoDb collection of clients that are approved before they are allowed to connect to the broker.

```
const verify_client = async () => {
    Clients.findOne({name: client_name, password: client_pass}, function (err, obj){
        if(err || obj == null ) {
            console.log("Wrong credentials, client was not connected.")
        } else{
            connectToBroker()
        }
    })
    }
```

On each update (set interval for publish ) the data is published to the broker.
**The broker, data analyzer** and **results presenter** are all hosted on the heruko platform

**The MQTT broker**

Our broker is based on examples in exercise 9. It acts as a middleman between the subscribers and publishers. Fired messages as data is sent from the sensors. This data is subscribed to by the data analyser and on every new entry the data is passed along.

Aedes.on connects/ disconnects clients, and passes data according to who is subscribed

```
ws.createServer({ server: httpServer }, aedes.handle);

httpServer.listen(port, function () {
    console.log('Aedes listening on port:', port)
    aedes.publish({ topic: 'aedes/hello', payload: "I'm broker " +
aedes.id })
})

//when a client connects
aedes.on('client', function (client) {
    console.log('Client Connected: \x1b[33m' + (client ? client.id :
client) + '\x1b[0m',
        'to broker', aedes.id)
})

//when a client disconnects
```

```
aedes.on('clientDisconnect', function (client) {
    console.log('Client Disconnected: \x1b[31m' + (client ? client.id
: client) +
        '\x1b[0m', 'to broker', aedes.id)
})

//when a topic is subscribed to
aedes.on('subscribe', function (subscriptions, client) {
    console.log('MQTT client \x1b[32m' + (client ? client.id :
client) +
        '\x1b[0m subscribed to topics: ' + subscriptions.map(s =>
s.topic).join('\n'),
        'from broker', aedes.id)
})

//when a topic is unsubscribed from
aedes.on('unsubscribe', function (subscriptions, client) {
    console.log('MQTT client \x1b[32m' + (client ? client.id :
client) +
        '\x1b[0m unsubscribed to topics: ' +
subscriptions.join('\n'), 'from broker',
        aedes.id)
})

// fired when a message is published
aedes.on('publish', function (packet, client) {
    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)
})
```

**The data analyzer**

**subscribers/data_analyzer**

The data analyser subscribes to the topic of each sensor. When a message is received from the broker it identifies what sensor made the latest update. When identified, the data is sorted according to the matching schema so that it can be added to the database.

The following functions are run before the database entry:

```
mongoose.connect
connectToBroker
const topicHumidity = 'humidity'
```

- Connects to database so that data can be pushed
- Connects to the broker in order to receive data
- Define the topics to subscribe to

When data is fired from the sensor through the broker, the data analyzer checks what topic got an update and adds the correct topic to the matching collection

```
//When a message is received, the data is parsed
client.on('message', async (topic, payload) => {
    const payloadJSON = JSON.parse(payload)
    let checkString = JSON.stringify(payloadJSON)
    // If the name = light, do light shit and if name = humidity, do humidity shit
    if (checkString.includes("Light_sensor")) {
        console.log('Received Message from Light sensor:', topicLight, payload.toString())
        LightSensorModel.create(payloadJSON, async function (err, entry) {
            if (err) return console.error(err);
            console.log("Added data to database")

        })
    } else if (checkString.includes("Humidity_sensor")) {
        console.log('Received Message from Humidity sensor:', topicHumidity, payload.toString())
        Humidity.create(JSON.parse(payload), async function (err, entry) {
            if (err) return console.error(err);
            console.log("Added data to database")
        })
    } else if (checkString.includes("Temperature_sensor")) {
        console.log('Received Message from Temperature sensor:', topicTemperature, payload.toString())
        TemperatureSensorModel.create(JSON.parse(payload), async function (err, entry) {
            if (err) return console.error(err);
            console.log("Added data to database")
        })
    }
})
}
```

**The results presenter**

**/views/index.ejs**

The results presenter is served by app.js. It uses variables from the queries in app.js to display the following data from MongoDb:

- The lowest/highest temperature/humidity/light entry
- The average value for, humidity, temperature and light
- Displays all entries in graphs made in chart.js

**Result of analysis:**

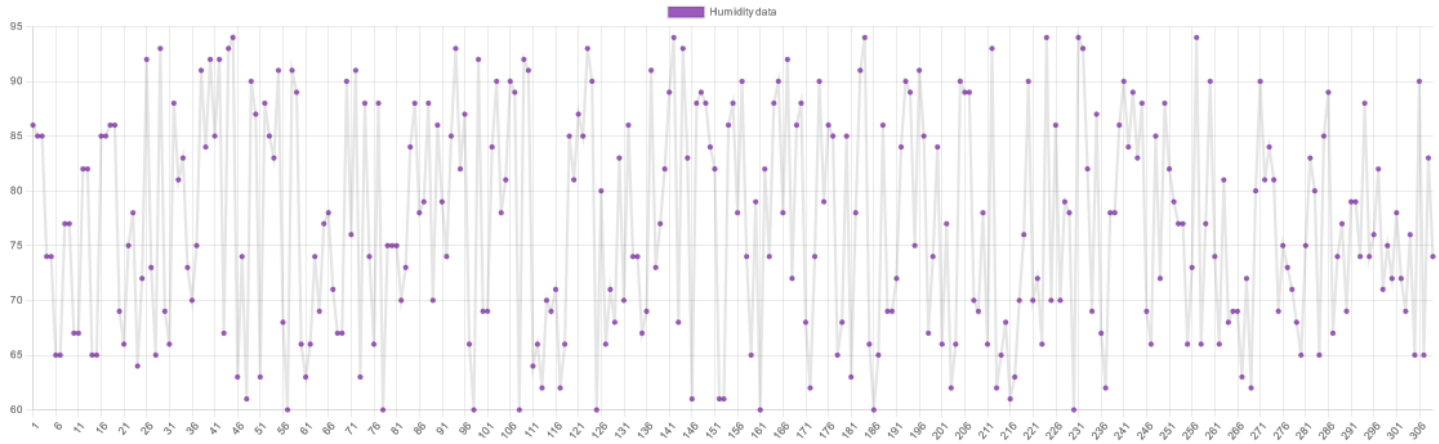
# Data Renderer

Average relative humidity: 76.98%

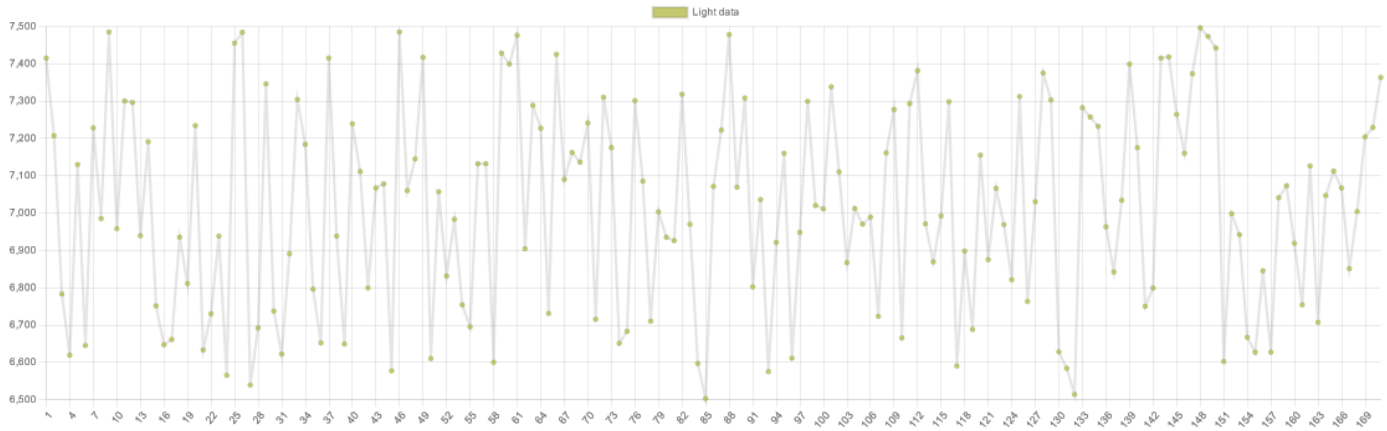
Average lumen: 7014.90 lm

Average temperature: 13.83 celsius

## Humidity Chart




## Light Chart




## Temperature Chart