Multi Sensor Air Pollution and Weather Station

June 2022 Version 3

Jalil Mirza, Stefan Poslad, Fesal Toosy

**Executive Summary**

This document gives **the technical description of a two-part low-cost IoT (Internet of Things) driven: Air Pollution (AP) or Air Quality (AQ) station and a Weather Station (WS)**. Either of these stations can be used independently of each other or used together. Currently, the AQ+WS system is built as at least 5 devices: a AQ sensor hub, a WS sensor hub, a local (edge) server, a remote cloud server for data analytics and visualization, an end-user access device, currently a laptop to view the data. The current physical forms of each of these stations use a wired interface to the edge server. There is a wireless (WiFi) interface between the edge and optional cloud server and end-user access device. Note the AQ and WS sensor hubs do share a common local ‘edge’ server so these must currently be co-located.

**In terms of the current status** (June 2022), a work prototype of this 4-5-device system has been demonstrated and powered via a laptop. It is noted that the current system protype design has some important limitations. Although it seems simple to create such a prototype, there were a number of non-trivial tech problems to achieve this including a worldwide shortage of Raspberry Pis as the ‘brains’ of the edge servers. If used outdoors, the AQ,WS hubs & edge server require use of a non-mains, battery pack, instead of a laptop – this has not yet been developed. Also note so far, no scientific AQ or WS studies or data has been collected outdoors.

**The next steps**: it is planned to focus on the AQ sensor hub only data and explore what kind of spatial-temporal variation distribution AQ data can be collected using a single sensor hub. We need to research including how developing how to power the sensor hub(s) and edge device outdoors. It also needs to be considered if we can re-design, partition and distribute the edge server into multiple lower-resource edge servers (based upon a Raspberry Pi zero) with one edge server per sensor hub that communicates wireless not wired so that can create a wireless network of several AQ and/or WS nodes.

**Future and Final Goal of the project:** is to involve QMUL students in the scientific experimentation of a small network of AQ and WS sensor hubs as this is a requirement by the (Wesfield) funder.

# Introduction

a two-part low-cost IoT (Internet of Things) driven, Air Pollution (AP) or Air Quality (AQ) station and Weather Station (WS) has been designed and implemented Either of these stations can be used independently of each other or used together. Currently, the AQ+WS system is built as 4-5 sub-systems of devices: a AQ sensor hub, a WS sensor hub, a local (edge) server, **a remote cloud server for data analytics and visualization [check is this done?]**, an end-user access device, currently a laptop to view the data. The current physical forms of each of these sensor hubs use a wired interface to the edge server. There is a wireless (WiFi) interface between the edge and cloud server. The AQ and WS sensor hubs do share a common local ‘edge’ server so these must currently be co-located. This design has limitations as a scalable or distributed design to build a network of multiple AQ+WS nodes to cover an area.  
  
Each of the AQ+WS sub-systems are described in turn. Note that there maybe some overlap between sub-systems, e.g., temperature, humidity, light sensors are needed for both AQ and a WS. These will be dealt with on a case by case basis.

# AQ Sensor Hub

## Dust Sensor (Empty)

## Temperature, Humidity, Pressure, Altitude, VOC (BME680)

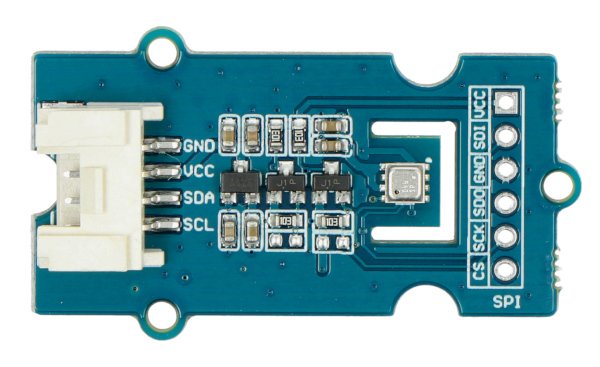
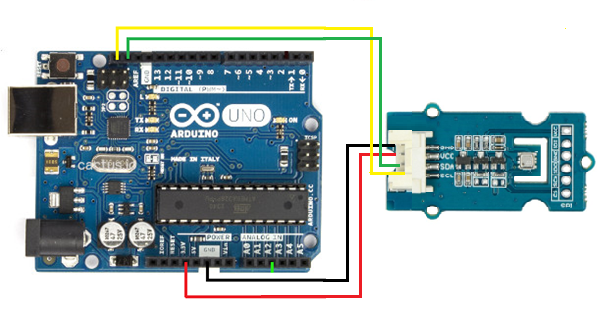


Figure 2.1 (BME680 Bosch Sensor Tec, I2C address: 0x76 (default).)

### Sensor specification BME680

* Supply voltage: 3.3 V to 5 V
* Measurement ranges:
  + temperature: from -40°C to 85°C - resolution 0.01°C
  + humidity: from 0% to 100% RH - resolution: 0.008% RH
  + pressure: 300 hPa to 1100 hPa - resolution: 0,18 hPa
  + IAQ (Indoor Air Quality): 0 to 500 - resolution 1
* Accuracy of measurements:
  + temperature: ± 0.5°C (at 25°C), ± 1°C (from 0°C to 60°C)
  + humidity: ± 3%RH
  + pressure: ± 0,6 hPa
  + IAQ: 0 to 3
* Gas measurement:
  + ethane
  + methyl
  + ethanol
  + acetone
  + carbon monoxide
* Communication: I2C / SPI
* I2C address: 0x76 (default). 0x77 (optional)
* Dimensions: 40 x 20 x 7 mm
* Weight: 9,1 g



## Multi Gas Sensor MiCS-6814 (CO+Others)

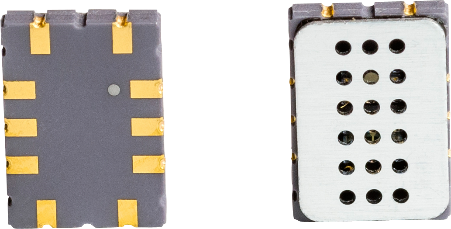
 

Figure 1.2.3 (Multi GAS Sensor MiCS-6814)

Module with multi-channel MiCS-6814 sensor, which can detect many harmful gases. It can measure three values simultaneously. The sensor can measure gases such as carbon dioxide CO, nitrogen dioxide NO2, ethanol, hydrogen H2, ammonia NO3, methane CH4, propane C3H8, and isobutane C4H10. The module is based on the ATmega168PA chip and communicates through the I2C interface. It is powered from 3.3 V to 5 V.

### Features

•Three fully independent sensing elements on one package

•Built with ATmega168PA

•I2C interface with programmable address

•Heating power can be shut down for low power

•Detectable gases

•Carbon monoxide CO 1 – 1000ppm

•Nitrogen dioxide NO2 0.05 – 10ppm

•Ethanol C2H5OH 10 – 500ppm

•Hydrogen H2 1 – 1000ppm

•Ammonia NH3 1 – 500ppm

•Methane CH4 >1000ppm

•Propane C3H8 >1000ppm

•Iso-butane C4H10 >1000ppm

### Gases Sensed

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item** | **Condition** | **Min.** | **Typ.** | **Max.** | **Unit** |
| Voltage | - | 3.1 | 3.3 | 5.25 | V |
| Ripple | @Max Power | - | 80 | 100 | mV |
| Heating Power | - | - | - | 88 | mW |
| Max Power | - | - | - | 150 | mW |
| ADC Precision | - | - | 10 | - | Bits |
| I2C Rate | - | - | 100 | 400 | kHz |
| VIL | @I2C | -0.5 | - | 0.99 | V |
| VIH | @I2C | 2.31 | - | 5.25 | V |

Interfacing

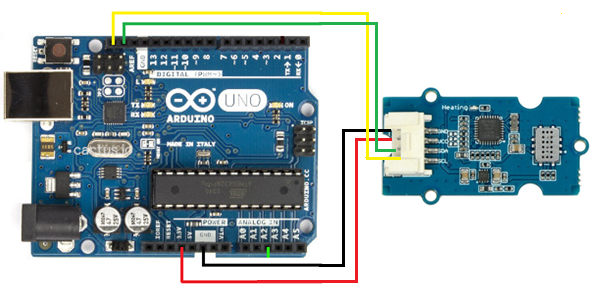


Figure 1.2.3.3 Interfacing with Arduino

### Programming

## Missing sensors NO2, O3 (Todo)

## AQ Sensor Hub Physical Design (Todo)

# WS Sensor Hub

A close-up of a microscope

Description automatically generated with medium confidence

A weather station is a facility, either on land or sea, with instruments and equipment for measuring atmospheric conditions to provide information for weather forecasts and to study the weather and climate. The measurements taken include temperature, relative humidity, barometric pressure, altitude, volatile gas components, wind speed, wind direction, precipitation amounts, and Sun radiation. Our design is based on IoT technology, automated measurements are taken every 30 seconds. The system provides real-time monitoring 24/7 and has the capability to store the data in local and cloud-based storage.

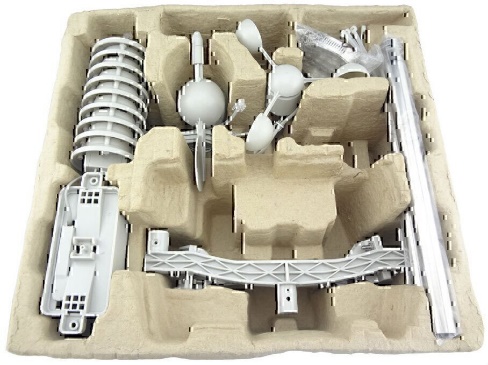


Figure 02.3 (Weather Meter Kit)

## WS sensors list

The following sensors are included in our design.

1. Temperature sensor for measuring the ambient temperature.
2. Relative humidity sensor for measuring relative humidity with a fast response time.

* Accuracy tolerance (+/- 3% r.H)
* Hysteresis (+/- 1.5% r.H)

1. Pressure sensor to measure the barometric pressure and altitude.
2. Gas LPG sensor to measure the Volatile Organic Components (VOC).
3. Anemometer for measuring the wind speed.
4. Wind vane for wind direction.
5. Rain gauge for measuring liquid precipitation over a set period of time.
6. Pyranometer for measuring the solar radiations.

### Solar Irradiation Shield



Figure 2.4 (c) (Solar Irradiation shield)

### Anemometer

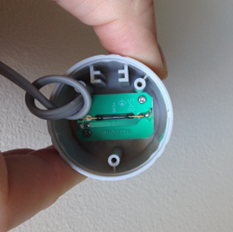
 

Figure 2.6 (a)

The cup-type anemometer part of the weather meter kit measures wind speed by closing a contact as a magnet moves past a switch. A wind speed of 1.492 MPH (2.4 km/h) causes the switch to close once per second. The anemometer switch is connected to the inner two conductors of the RJ11 cable shared by the anemometer and wind vane (pins 2 and 3.)

### Wind Vane

The wind vane is also part of the Weather meter kit and the most complicated of the three sensors. It has eight switches, each connected to a different resistor. The vane’s magnet may close two switches at once, allowing up to 16 different positions to be indicated. An external resistor can be used to form a voltage divider, producing a voltage output that can be measured with an analog to digital converter by using a microcontroller, as shown below.



Figure 2.13 (a) Wind Vane

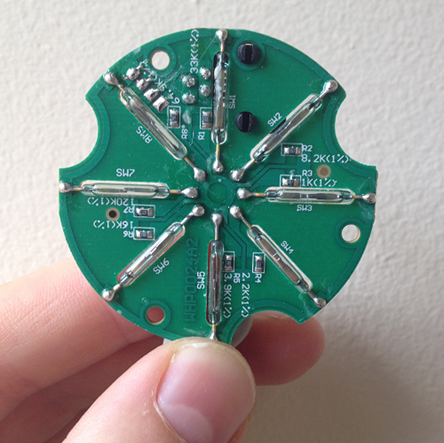
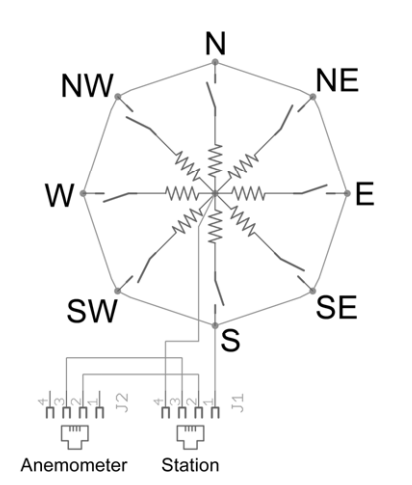


Figure 2.13 (b) (Wind Vane)

The switch and resistor arrangement are shown in the diagram to the right. Resistance values for all 16 possible positions are given in the table. Resistance values for positions between those shown in the diagram are the result of two adjacent resistors connected in parallel when the vane’s magnet activates two switches simultaneously.

|  |  |  |  |
| --- | --- | --- | --- |
| **Direction** | **Angle** | **Range** | |
| N | 0 | 0.24 | 0.69 |
| NE | 45 | 0.66 | 1.17 |
| E | 90 | 0.58 | 2.26 |
| SE | 135 | 2.34 | 3.85 |
| S | 180 | 4.49 | 4.77 |
| SW | 225 | 4.11 | 4.60 |
| W | 270 | 3.07 | 4.66 |
| NW | 315 | 1.37 | 3.18 |

Table 2.13

### Rain Gauge

The rain gauge is a self-emptying tipping bucket-type part of a weather meter kit. Each 0.011” (0.2794 mm) of rain causes one momentary contact closure that we connect with the microcontroller to interrupt input. The gauge’s switch is connected to the two center conductors of the attached RJ11-terminated cable.

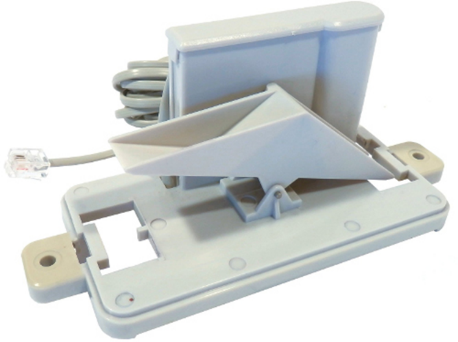
 

Figure 2.14 (Rain Gauge)

### Interfacing

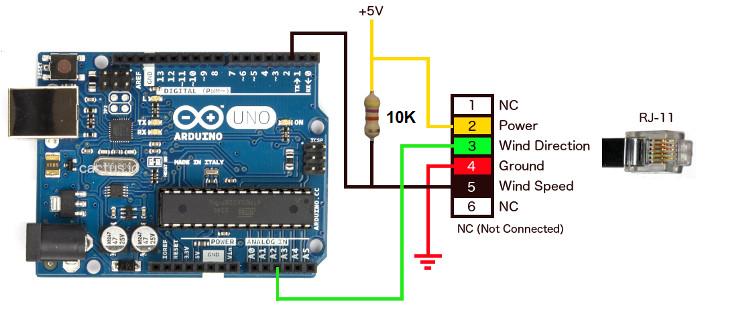
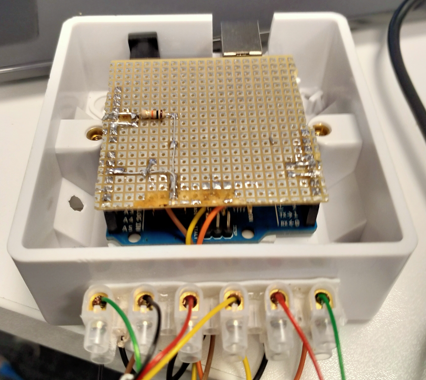


Figure 2.15 (c) Wind vane interface circuit. Voltage readings for a 5-volt supply and a resistor value of 10k ohms are given in the table.)

A close-up of a computer chip

Description automatically generated with low confidence

**Figure 3.1 a**



**Figure 3.1 b**

A picture containing electronics

Description automatically generated

**Figure 3.1 c**

* Port A2 (Wind Vane)
* Port 3 and 2 (Wind Speed and Rain Gauge)
* I2C for BME680 and VEML6075
* USB (Raspberry Pi)
* Port 1 and 0 Tx/Rx serial Port (Radio Module)

## Light Sensor (UV, IR, and Visible Light)

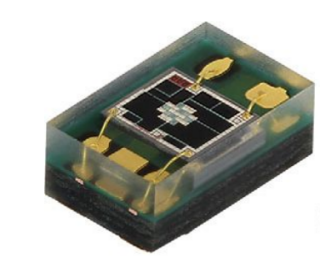
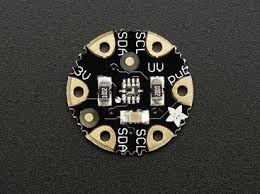
 

Figure 2.4 (a) VEML6075 Vishay Light Sensor and Adafruit VEML6075 module

### Descriptions

The SI1145 is a new sensor from SiLabs with a calibrated light sensing algorithm that can calculate UV Index. It doesn't contain an actual UV sensing element, instead it approximates it based on visible & IR light from the sun. It's a digital sensor that works over I2C so just about any microcontroller can use it. The sensor also has individual visible and IR sensing elements so you can measure just about any kind of light.

UV Index

UV light is light in the "ultra violet" spectrum, beyond the violet part of visible light. Even though UV is not visible to human eyes it does effect the eyes and skin! UV light is what lets us produce vitamin D, which is essential for humans. But it can also burn skin - what we call a sunburn.

We can measure UV in mW per area, but that's a little tough to keep in your head, especially if the area value is in cm or m or ft. To make it easier for people to track UV safety, the WHO came up with the UV index.

The SI1145 is calibrated to transmit the UV index based on the visible & IR light received, but what do those numbers mean?

As you're probably aware of - if you have lighter skin, you are more likely to get burned. So it's important to watch the UV index and wear sunscreen, hats, and sunglasses when the UV index gets above 4 or so!

**(UV index text below from epa.gov)**

0 ~ 2: Low

A UV Index reading of 0 to 2 means low danger from the sun's UV rays for the average person. Wear sunglasses on bright days. If you burn easily, cover up and use broad spectrum SPF 30+ sunscreen. Watch out for bright surfaces, like sand, water and snow, which reflect UV and increase exposure.

3 ~ 5: Moderate

A UV Index reading of 3 to 5 means moderate risk of harm from unprotected sun exposure. Stay in shade near midday when the sun is strongest. If outdoors, wear protective clothing, a wide-brimmed hat, and UV-blocking sunglasses. Generously apply broad spectrum SPF 30+ sunscreen every 2 hours, even on cloudy days, and after swimming or sweating. Watch out for bright surfaces, like sand, water and snow, which reflect UV and increase exposure.

6 ~ 7: High

A UV Index reading of 6 to 7 means high risk of harm from unprotected sun exposure. Protection against skin and eye damage is needed. Reduce time in the sun between 10 a.m. and 4 p.m. If outdoors, seek shade and wear protective clothing, a wide-brimmed hat, and UV-blocking sunglasses. Generously apply broad spectrum SPF 30+ sunscreen every 2 hours, even on cloudy days, and after swimming or sweating. Watch out for bright surfaces, like sand, water and snow, which reflect UV and increase exposure.

8 ~ 10: Very high

A UV Index reading of 8 to 10 means very high risk of harm from unprotected sun exposure. Take extra precautions because unprotected skin and eyes will be damaged and can burn quickly. Minimize sun exposure between 10 a.m. and 4 p.m. If outdoors, seek shade and wear protective clothing, a wide-brimmed hat, and UV-blocking sunglasses. Generously apply broad spectrum SPF 30+ sunscreen every 2 hours, even on cloudy days, and after swimming or sweating. Watch out for bright surfaces, like sand, water and snow, which reflect UV and increase exposure.

11 or more: Extreme

A UV Index reading of 11 or more means extreme risk of harm from unprotected sun exposure. Take all precautions because unprotected skin and eyes can burn in minutes. Try to avoid sun exposure between 10 a.m. and 4 p.m. If outdoors, seek shade and wear protective clothing, a wide-brimmed hat, and UV-blocking sunglasses. Generously apply broad spectrum SPF 30+ sunscreen every 2 hours, even on cloudy days, and after swimming or sweating. Watch out for bright surfaces, like sand, water and snow, which reflect UV and increase exposure.

### Technical details

SI1145 Details:

* IR Sensor Spectrum: Wavelength: 550nm-1000nm (centered on 800)
* Visible Light Sensor Spectrum: Wavelength: 400nm-800nm (centered on 530)
* Voltage Supply: Power with 3 VDC
* I2C address 0x60 (7-bit)
* Operating Temperature: -40°C ~ 85°C

Breakout Board details:

* Diameter: 14mm / 0.6"
* Height: 2mm / 0.08"
* Weight: 0.3g

### Interfacing:

I2C Pins

So you want to actually read the sensor data, these are the pins to use! This sensor uses classic I2C. There's 10K pullups on both pins. You can share the i2c pins with many other sensors as long as they do not use the same address - the SI1145 uses 7-bit address 0x60

SCL - this is the i2c clock pin, connect to your Flora I2C clock master pin.

SDA - this is the i2c data pin, connect to your Flora I2C data master pin.

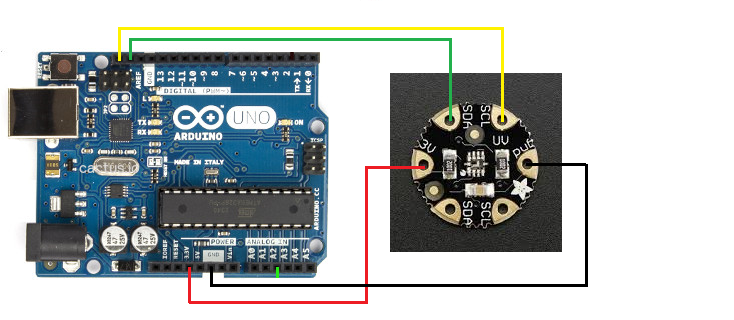


Figure 2.4 (b) VEML6075 module I2C Interfacing with Arduino Uno, I2C address 0x60, 7-bit

### Code:

## Weather Station Installation

A picture containing text, clock, sign

Description automatically generatedIcon

Description automatically generatedIcon

Description automatically generated

Diagram

Description automatically generated

# **Edge Server**

## Raspberry Pi 3.0 Bootup

Connect power and check the red power LED on the board is on.

On your connectd HDMI display (if not check if you selected correct HDMI input on TV) you should see the boot sequence:

•Stage 1 boot is in the on-chip ROM. Loads Stage 2 in the L2 cache

•Stage 2 is bootcode.bin. Enables SDRAM and loads Stage 3

•Stage 3 is loader.bin. It knows about the. elf format and loads start.elf

After first boot the Raspberry Pi boots up in setup mode. If it does not boot into setup mode you can type the following command to get there: sudo raspi-config.

## **Data Acquisition Service (todo)**

Figure 3.1.d

## Data Processing Service (todo)

### Software Installation and Configuration (details lacking)



**Figure 4.2**

The following software is installed on the Edge Server (Pi 3)

* Linux OS – the operating system
* Apache Web Server: TomCat
* MySql database is used to store the sensor data
* Node.js is used to
* Node-RED to visualise the sensor data in real-time: [https://nodered.org](https://nodered.org/)
* Python **is used to ????**
* Grafana to visualise and process the off-line sensor data

**Hardware Additions and Configuration**

Before installation of any software the Raspberry Pi that lacks an encased power unit should be equipped with a heat sink kit other vs the processor temperature increased and Raspberry Pi will be halted.

## Networking Topology (details lacking)

AQ (and WS) hubs are connected by wire to USB connectors in the Pi edge server. N.B. This is not a scalable design as we need 1 edge server to be wired to each sensor hub if we network these. We need users’ access devices to be remote to the sensor hub location so wireless comms is needed between these. There are different options: WiFi, LoRa, proprietary RF, etc.

### Wifi for IoT @QMUL Overview

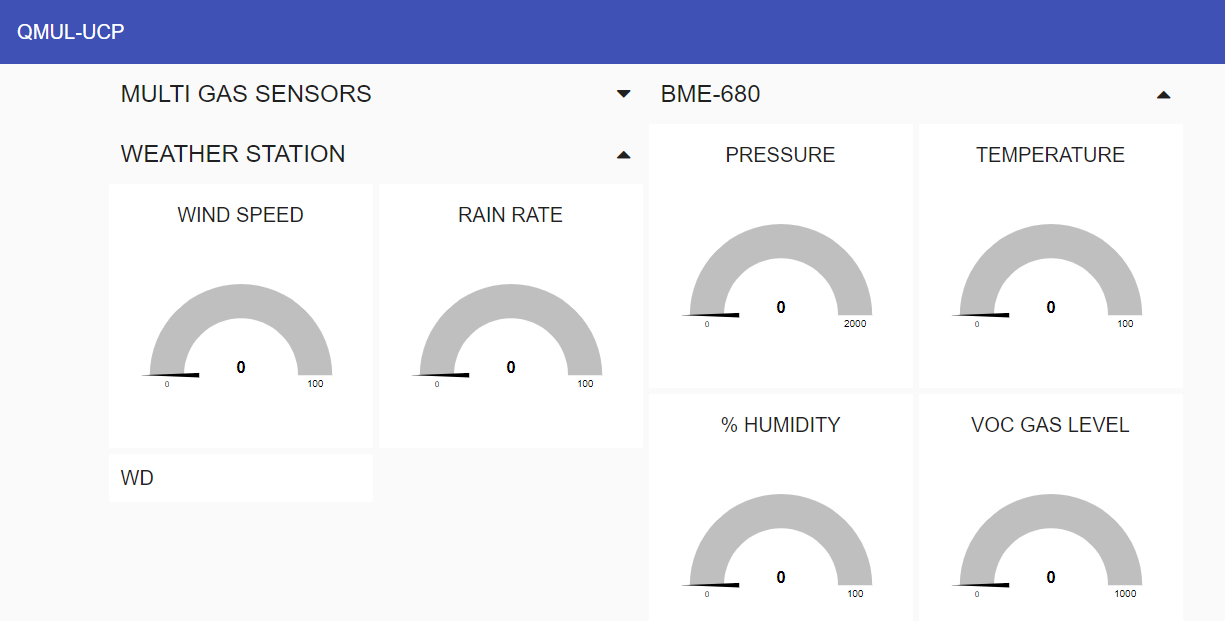
At QMUL there are 3 different WiFi networks available, each with their pros & cons: EduRoam, Guest, Events. The main common problem across all 3 different WiFi networks is that QMUL sets up WiFi to connect people not Things because WiFi access is linked to a person’s secure access credentials. There is no or little support or policy to link Things securely via WiFi @QMUL.

The principal issue with the QMUL events wifi, is that it requires a password change 1-2 times weekly, a big overhead to have to stop & reconfigure wifi connected things periodically over much longer periods.

The principal 2 issues with EduRoam to connect things via WiFi is that access is linked to a person’s logon credentials (LC) but a person’s LC should not be used to configure things with personal EduRoam credentials as these are usually plain coded in software and therefore can be easily discovered and then falsely used. In addition, in some low ICT resource IoT devices have an insufficient computation capability to support the strong encryption needed by EduRoam, e.g., not Arduino but possible some but not all types of Raspberry Pi. It’s unclear if we can request and require a guest account for multiple Things.

Hence the QMUL Guest wifi network is used.

## Sensor data Dashboard (no details)



**Figure 4.3 (Dashboard for AQ+WS)**

The dashboard is created using Node Red and is available at the following locations:  
[Node-RED Dashboard](http://161.23.55.170:1880/ui/#!/0?socketid=dGxK_m36Z1MjoQtoAAAF)

[Node-RED Dashboard](http://161.23.55.170:1880/ui/#!/0?socketid=4n95UOut_rIHn2--AAAF)

[Node-RED : 161.23.55.170](http://161.23.55.170:1880/)

[Node-RED : 161.23.55.170](http://161.23.55.170:1880/#flow/0f0f78d31b20a7de)

# Appendix 1.0 (code? Which?)

**Weather Station Arduino Code:**

/\*

AUTHOR: JALIL MIRZA (Assistant Professor) UCP

Project: QMUL-LL Project

Name: Weather Station Arduino Code 1

Date: 12-Apr-2022 23:30

Location: IoT Lab QMUL (Dr. Stefan Lab)

Features:

- Wind Direction

- Wind Speed

- Rain Fall

\*/

#define SKETCH\_NAME "Weather Station Part 1"

#define INTERVAL 30

const byte rainRatePin = 2;

const byte windSpeedPin = 3;

const byte windvoltage\_divider = A2;

unsigned long lastSend;

unsigned int windcnt = 0;

unsigned int raincnt = 0;

float WS\_WIND\_SPD = 0;

float WS\_RAIN\_RATE = 0;

String WS\_WIND\_DIR = "";

String wind\_direction = "";

float wind\_speed = 0;

float rain\_rate = 0;

void cntWindSpeed()

{

windcnt++;

}

void cntRain()

{

raincnt++;

}

void getWindSpeedWindDirectionAndRainRateData()

{

wind\_speed = (windcnt / INTERVAL) \* 2.4;

rain\_rate = (raincnt / 2) \* 0.2794;

windcnt = 0;

raincnt = 0;

WS\_WIND\_SPD = wind\_speed;

WS\_RAIN\_RATE = rain\_rate;

wind\_speed = 0;

rain\_rate = 0;

float voltage\_divider = analogRead(windvoltage\_divider) \* (5 / 1023.0);

// Serial.print("V="); Serial.println(voltage\_divider);

if (voltage\_divider > 0.24 && voltage\_divider < 0.69 ) {

wind\_direction = "N";

WS\_WIND\_DIR = wind\_direction;

wind\_direction = "";

delay(30);

}

else if (voltage\_divider > 0.66 && voltage\_divider < 1.17 ) {

wind\_direction = "NE";

WS\_WIND\_DIR = wind\_direction;

wind\_direction = "";

delay(30);

}

else if (voltage\_divider > 0.58 && voltage\_divider < 2.26 ) {

wind\_direction = "E";

WS\_WIND\_DIR = wind\_direction;

wind\_direction = "";

delay(30);

}

else if (voltage\_divider > 2.34 && voltage\_divider < 3.85 ) {

wind\_direction = "SE";

WS\_WIND\_DIR = wind\_direction;

wind\_direction = "";

delay(30);

}

else if (voltage\_divider > 4.49 && voltage\_divider < 4.77 ) {

wind\_direction = "S";

WS\_WIND\_DIR = wind\_direction;

wind\_direction = "";

delay(30);

}

else if (voltage\_divider > 4.11 && voltage\_divider < 4.60 ) {

wind\_direction = "SW";

WS\_WIND\_DIR = wind\_direction;

wind\_direction = "";

delay(30);

}

else if (voltage\_divider > 3.07 && voltage\_divider < 4.66 ) {

wind\_direction = "W";

WS\_WIND\_DIR = wind\_direction;

wind\_direction = "";

delay(30);

}

else if (voltage\_divider > 1.37 && voltage\_divider < 3.18 ) {

wind\_direction = "NW";

WS\_WIND\_DIR = wind\_direction;

wind\_direction = "";

delay(30);

}

String str\_Payload;

str\_Payload += String(WS\_RAIN\_RATE); //1

str\_Payload += "," + String(WS\_WIND\_SPD);//2

str\_Payload += "," + (WS\_WIND\_DIR); //3

str\_Payload += ","; + "\r\n";

byte Payload[str\_Payload.length()];

int LengthFrameAPI = 18 + sizeof(Payload);

int LengthPayload = sizeof(Payload);

Serial.print(str\_Payload + "\r\n");

lastSend = millis();

}

void setup()

{

Serial.begin(9600); // start serial for output

pinMode(rainRatePin, INPUT\_PULLUP);

pinMode(windSpeedPin, INPUT\_PULLUP);

attachInterrupt(digitalPinToInterrupt(rainRatePin), cntRain, RISING);

attachInterrupt(digitalPinToInterrupt(windSpeedPin), cntWindSpeed, RISING);

}

void loop()

{

// of ignore first reading

if ( millis() - lastSend > INTERVAL \* 1000 )

{

getWindSpeedWindDirectionAndRainRateData(); // Update and send only after delay

}

}

**Appendix 1.1**

**Environmental Sensors Arduino Code:**

/\*

AUTHOR: JALIL MIRZA (Assistant Professor) UCP

Project: QMUL-LL Project

Name: Environmental Sensors Arduino Code 2

Date: 12-Apr-2022 23:30

Location: IoT Lab QMUL (Dr. Stefan Lab)

Features:

- Temperature

- Humidity

- Barometric pressure

- Gas: Volatile Organic Compounds (VOC) like ethanol and carbon monoxide

- Altitude

- Dewpoint \*

- Heat Index \*

- UV Index

- IR Radiation

- Visible Light

- Gas NH3

- Gas CO

- Gas NO2

- Gas C3H8

- Gas CH4

- Gas C2H5OH

- Gas Dust Particle 2.5

The BME680 contains a MOX (Metal-oxide) sensor that detects VOCs in the air.

This sensor gives you a qualitative idea of the sum of VOCs/contaminants in

the surrounding air – it is not specific for a specific gas molecule.

MOX sensors are composed of a metal-oxide surface, a sensing chip to measure

changes in conductivity, and a heater. It detects VOCs by adsorption of oxygen

molecules on its sensitive layer.

The BME680 reacts to most VOCs polluting indoor air (except CO2). When the

sensor comes into contact with the reducing gases, the oxygen molecules

react and increase the conductivity across the surface. As a raw signal,

the BME680 outputs resistance values. These values change due to variations

in VOC concentrations:

\*/

#define SKETCH\_NAME "WeatherStation Part 2"

#define INTERVAL 30

#include <Wire.h>

#include "MutichannelGasSensor.h"

#include "Zanshin\_BME680.h"

#include "Adafruit\_SI1145.h"

Adafruit\_SI1145 uv = Adafruit\_SI1145();

//< Set the baud rate for Serial I/O

//const uint32\_t SERIAL\_SPEED {

// 9600

//};

// Include the BME680 Sensor library

BME680\_Class BME680;

float altitude(const int32\_t press, const float seaLevel = 1019);

float altitude(const int32\_t press, const float seaLevel) {

static float Altitude;

Altitude = 44330.0 \* (1.0 - pow(((float)press / 100.0) / seaLevel, 0.1903)); // Convert into meters

return (Altitude);

} // of method altitude()

float GAS\_SENSOR\_NH3 = 0;

float GAS\_SENSOR\_CO = 0;

float GAS\_SENSOR\_NO2 = 0;

float GAS\_SENSOR\_C3H8 = 0;

float GAS\_SENSOR\_C4H10 = 0;

float GAS\_SENSOR\_CH4 = 0;

float GAS\_SENSOR\_H2 = 0;

float GAS\_SENSOR\_C2H5OH = 0;

float GAS\_SENSOR\_Value = 0;

float DUST\_SENSOR\_CONS = 0;

float UV\_SENSOR\_VISIBLE = 0;

float UV\_SENSOR\_IR = 0;

float UV\_SENSOR\_UV = 0;

//temperatureCompensatedAltitude(int32\_t pressure, float temp =21.0, const float seaLevel=1013.25);

const byte LightSensorPin = A0;

unsigned long lastSend;

// Dust Sensor start

int pin = 8;

unsigned long duration;

unsigned long starttime;

unsigned long sampletime\_ms = 2000;

unsigned long lowpulseoccupancy = 0;

float ratio = 0;

float concentration = 0;

// Dust Sensor end

void getAndSendTemperatureAndHumidityData()

{

static int32\_t temp, humidity, pressure, gas; // BME readings

static char buf[16]; // sprintf text buffer

static float alt; // Temporary variable

float BME680D\_TEMP = 0;

float BME680D\_HUMI = 0;

float BME680D\_PRES = 0;

float BME680D\_VOCG = 0;

float BME680C\_ALT = 0;

float WS\_LDR\_LS = 0;

//float Rsensor;

//Resistance of sensor in K

//Calculate Wind Speed (klicks/interval \* 2,4 kmh)

// Get readings

int value = analogRead(A0);

BME680.getSensorData(temp, humidity, pressure, gas);

BME680D\_TEMP = temp / 100.0;

BME680D\_HUMI = humidity / 1000.0;

BME680D\_PRES = pressure / 100.0;

// Resistance milliohms

BME680D\_VOCG = gas / 100.0;

WS\_LDR\_LS = (float)(1023 - value) \* 5 / value;

alt = altitude(pressure); // temp altitude

BME680C\_ALT = (temperatureCompensatedAltitude(pressure, temp/100.0/\*, 1022.0\*/),2);

//////////////////// 2022-04-25 /////////////////

String str\_Payload;

str\_Payload += String(BME680D\_TEMP); //0

str\_Payload += "," + String(BME680D\_HUMI); //1

str\_Payload += "," + String(BME680D\_PRES); //2

str\_Payload += "," + String(BME680D\_VOCG); //3

str\_Payload += "," + String(WS\_LDR\_LS); //4

str\_Payload += "," + String(GAS\_SENSOR\_NH3); //5

str\_Payload += "," + String(GAS\_SENSOR\_CO); //6

str\_Payload += "," + String(GAS\_SENSOR\_NO2); //7

str\_Payload += "," + String(GAS\_SENSOR\_C3H8); //8

str\_Payload += "," + String(GAS\_SENSOR\_C4H10); //9

str\_Payload += "," + String(GAS\_SENSOR\_CH4); //10

str\_Payload += "," + String(GAS\_SENSOR\_H2); //11

str\_Payload += "," + String(GAS\_SENSOR\_C2H5OH); //12

str\_Payload += "," + String(DUST\_SENSOR\_CONS); //13

str\_Payload += "," + String(UV\_SENSOR\_VISIBLE); //14

str\_Payload += "," + String(UV\_SENSOR\_IR); //15

str\_Payload += "," + String(UV\_SENSOR\_UV); //16

str\_Payload += ","; + "\r\n"; //17

byte Payload[str\_Payload.length()];

int LengthFrameAPI = 18 + sizeof(Payload);

int LengthPayload = sizeof(Payload);

Serial.print(str\_Payload + "\r\n");

lastSend = millis();

}

void setup() {

Serial.begin(9600); // start serial for output

gas.begin(0x04);//the default I2C address of the slave is 0x04

gas.powerOn();

if (! uv.begin()) {

Serial.println("Didn't find Si1145");

while (1);

}

pinMode(8, INPUT);

starttime = millis();

// Start BME680 using I2C, use first device found

while (!BME680.begin(I2C\_STANDARD\_MODE)) {

// Serial.print(F("- Unable to find BME680. Trying again in 5 seconds.\n"));

delay(5000);

}

// of loop until device is located

BME680.setOversampling(TemperatureSensor, Oversample16); // Use enumerated type values

BME680.setOversampling(HumiditySensor, Oversample16); // Use numerated type values

BME680.setOversampling(PressureSensor, Oversample16); // Use enumerated type values

// Serial.print(F("- Setting IIR filter to a value of 4 samples\n"));

BME680.setIIRFilter(IIR4); // Use enumerated type values

// Serial.print(F("- Setting gas measurement to 320\xC2\xB0\x43 for 150ms\n")); // "�C" symbols

BME680.setGas(320, 150); // 320�c for 150 milliseconds

} // of method setup()

void loop()

{

// of ignore first reading

if ( millis() - lastSend > INTERVAL \* 1000 ) { // Update and send only after delay

getAndSendTemperatureAndHumidityData();

}

////////////////////////////////////////////////////////////

float c = 0;

c = gas.measure\_NH3();

GAS\_SENSOR\_NH3 = c;

c = gas.measure\_CO();

GAS\_SENSOR\_CO = c;

c = gas.measure\_NO2();

GAS\_SENSOR\_NO2 = c;

c = gas.measure\_C3H8();

GAS\_SENSOR\_C3H8 = c;

c = gas.measure\_C4H10();

GAS\_SENSOR\_C4H10 = c;

c = gas.measure\_CH4();

GAS\_SENSOR\_CH4 = c;

c = gas.measure\_H2();

GAS\_SENSOR\_H2 = c;

c = gas.measure\_C2H5OH();

GAS\_SENSOR\_C2H5OH = c;

////////////////////////////////////////////////////////////

duration = pulseIn(pin, LOW);

lowpulseoccupancy = lowpulseoccupancy + duration;

if ((millis() - starttime) >= sampletime\_ms) //if the sampel time = = 30s

{

ratio = lowpulseoccupancy / (sampletime\_ms \* 10.0);

concentration = 1.1 \* pow(ratio, 3) - 3.8 \* pow(ratio, 2) + 520 \* ratio + 0.62;

DUST\_SENSOR\_CONS = concentration;

lowpulseoccupancy = 0;

starttime = millis();

}

float UVindex = uv.readUV();

// the index is multiplied by 100 so to get the

// integer index, divide by 100!

UVindex /= 100.0;

UV\_SENSOR\_VISIBLE = (uv.readVisible());

UV\_SENSOR\_IR = (uv.readIR());

UV\_SENSOR\_UV = (UVindex);

delay(5000);

}

**Appendix 1.2**

**Weather Station Dashboard Node-Red Code:**

[

{

"id": "28e25aef1a915684",

"type": "tab",

"label": "Weather Station",

"disabled": false,

"info": "",

"env": []

},

{

"id": "3659f57cc450c3d4",

"type": "split",

"z": "28e25aef1a915684",

"name": "",

"splt": ",",

"spltType": "str",

"arraySplt": 1,

"arraySpltType": "len",

"stream": false,

"addname": "",

"x": 230,

"y": 300,

"wires": [

[

"37bfca132fe586ee"

]

]

},

{

"id": "1d7b91c9a6c84feb",

"type": "change",

"z": "28e25aef1a915684",

"name": "to number",

"rules": [

{

"t": "set",

"p": "payload",

"pt": "msg",

"to": "$number(msg.payload)",

"tot": "num"

},

{

"t": "set",

"p": "payload",

"pt": "msg",

"to": "$String(msg.payload)",

"tot": "str"

}

],

"action": "",

"property": "",

"from": "",

"to": "",

"reg": false,

"x": 380,

"y": 300,

"wires": [

[

"c4cc708d6c780bb4"

]

]

},

{

"id": "37bfca132fe586ee",

"type": "switch",

"z": "28e25aef1a915684",

"name": "Message Splitter SN",

"property": "parts.index",

"propertyType": "msg",

"rules": [

{

"t": "eq",

"v": "0",

"vt": "num"

},

{

"t": "eq",

"v": "1",

"vt": "str"

},

{

"t": "eq",

"v": "2",

"vt": "str"

},

{

"t": "eq",

"v": "3",

"vt": "str"

},

{

"t": "eq",

"v": "4",

"vt": "str"

},

{

"t": "eq",

"v": "5",

"vt": "str"

},

{

"t": "eq",

"v": "6",

"vt": "str"

},

{

"t": "eq",

"v": "7",

"vt": "str"

},

{

"t": "eq",

"v": "8",

"vt": "str"

},

{

"t": "eq",

"v": "9",

"vt": "str"

},

{

"t": "eq",

"v": "10",

"vt": "str"

},

{

"t": "eq",

"v": "11",

"vt": "str"

},

{

"t": "eq",

"v": "12",

"vt": "str"

},

{

"t": "eq",

"v": "13",

"vt": "str"

},

{

"t": "eq",

"v": "14",

"vt": "str"

},

{

"t": "eq",

"v": "15",

"vt": "str"

},

{

"t": "eq",

"v": "16",

"vt": "str"

}

],

"checkall": "true",

"repair": true,

"outputs": 17,

"x": 580,

"y": 300,

"wires": [

[

"5086ed1d68667a33",

"846449f6db9fecf4"

],

[

"37b5da1392cda604",

"2837cbfc94dd2a46"

],

[

"cab2a80c168d96ab",

"fadcac93409af35b"

],

[

"6494cf0ffedf304b",

"477f0df469a58a8f"

],

[

"ca50dae7ae6be9fe"

],

[

"0b276660d331ed05",

"99cf44cd9e5b4e80"

],

[

"1ffb91f73aa41ff8",

"d6a708f715d5612a"

],

[

"8eccd04884396363",

"a726ccf8fbcdde4a"

],

[

"1098f535d7d3acb7"

],

[

"e1717b3423bdc6ba"

],

[

"e97a3fe91ed291a3"

],

[

"15d1c0b922b3a3c7"

],

[

"3f52d7901f0964de"

],

[

"e6c27e211599cfe6"

],

[

"43e9629ca443f408"

],

[

"f3845eaa1c090471"

],

[

"58607cd131887da7"

]

]

},

{

"id": "0327d66081fa0564",

"type": "comment",

"z": "28e25aef1a915684",

"name": "split",

"info": "Split comma separated string into separated messages.\nValue placed to msg.payload\nValue type will be still string.\n\nThere will be 4 messages out of this split node\n\nFor each msg also given a property \"parts\". \nIt will be used later on.",

"x": 230,

"y": 240,

"wires": []

},

{

"id": "5f89fed288a21008",

"type": "comment",

"z": "28e25aef1a915684",

"name": "convert",

"info": "Convert the value of msg.payload to the number using JSONata expression.\n",

"x": 370,

"y": 240,

"wires": []

},

{

"id": "a1c309d94997e473",

"type": "comment",

"z": "28e25aef1a915684",

"name": "route",

"info": "Route every message to differet output by using switch node.\nAs property \"parts\" has been given to each message, we can use its \n\"index\" property to find out proper output\n\n",

"x": 630,

"y": 140,

"wires": []

},

{

"id": "846449f6db9fecf4",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "cdb1e6d07c5ef848",

"order": 2,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "TEMPERATURE",

"label": "°C",

"format": "{{value}}",

"min": 0,

"max": "100",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "25",

"seg2": "50",

"className": "",

"x": 1090,

"y": 260,

"wires": []

},

{

"id": "37b5da1392cda604",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "cdb1e6d07c5ef848",

"order": 3,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "% HUMIDITY",

"label": "%RH",

"format": "{{value}}",

"min": 0,

"max": "100",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "50",

"seg2": "75",

"className": "",

"x": 1080,

"y": 300,

"wires": []

},

{

"id": "cab2a80c168d96ab",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "cdb1e6d07c5ef848",

"order": 1,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "PRESSURE",

"label": "Pa",

"format": "{{value}}",

"min": 0,

"max": "2000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "500",

"seg2": "1000",

"className": "",

"x": 1070,

"y": 340,

"wires": []

},

{

"id": "477f0df469a58a8f",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "cdb1e6d07c5ef848",

"order": 4,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "VOC GAS LEVEL",

"label": "Ohms",

"format": "{{value}}",

"min": 0,

"max": "1000",

"colors": [

"#ff0000",

"#e6e600",

"#1ff702"

],

"seg1": "150",

"seg2": "300",

"className": "",

"x": 1090,

"y": 380,

"wires": []

},

{

"id": "9b9d83f69cc48521",

"type": "debug",

"z": "28e25aef1a915684",

"name": "",

"active": false,

"tosidebar": true,

"console": false,

"tostatus": false,

"complete": "false",

"statusVal": "",

"statusType": "auto",

"x": 1330,

"y": 680,

"wires": []

},

{

"id": "5086ed1d68667a33",

"type": "function",

"z": "28e25aef1a915684",

"name": "db Temp",

"func": "msg.temp = msg.payload;\nmsg.topic = \"INSERT INTO sensordata (sensorid,topic,value) VALUES ('QMUL-WS-02','/QMUL/CSEE/B312/TEMP',\"+ msg.temp +\");\"\nreturn msg;",

"outputs": 1,

"noerr": 0,

"initialize": "",

"finalize": "",

"libs": [],

"x": 1060,

"y": 80,

"wires": [

[

"31f9671db406cad8"

]

]

},

{

"id": "2837cbfc94dd2a46",

"type": "function",

"z": "28e25aef1a915684",

"name": "db humidity",

"func": "msg.temp = msg.payload;\nmsg.topic = \"INSERT INTO sensordata (sensorid,topic,value) VALUES ('QMUL-WS-02','/QMUL/CSEE/B312/HUMI',\"+ msg.temp +\");\"\nreturn msg;",

"outputs": 1,

"noerr": 0,

"initialize": "",

"finalize": "",

"libs": [],

"x": 1070,

"y": 120,

"wires": [

[

"31f9671db406cad8"

]

]

},

{

"id": "fadcac93409af35b",

"type": "function",

"z": "28e25aef1a915684",

"name": "db pressure",

"func": "msg.temp = msg.payload;\nmsg.topic = \"INSERT INTO sensordata (sensorid,topic,value) VALUES ('QMUL-WS-02','/QMUL/CSEE/B312/PRES',\"+ msg.temp +\");\"\nreturn msg;",

"outputs": 1,

"noerr": 0,

"initialize": "",

"finalize": "",

"libs": [],

"x": 1070,

"y": 160,

"wires": [

[

"31f9671db406cad8"

]

]

},

{

"id": "6494cf0ffedf304b",

"type": "function",

"z": "28e25aef1a915684",

"name": "db gas ",

"func": "msg.temp = msg.payload;\nmsg.topic = \"INSERT INTO sensordata (sensorid,topic,value) VALUES ('QMUL-WS-02','/QMUL/CSEE/B312/VOCG',\"+ msg.temp +\");\"\nreturn msg;",

"outputs": 1,

"noerr": 0,

"initialize": "",

"finalize": "",

"libs": [],

"x": 1050,

"y": 200,

"wires": [

[

"31f9671db406cad8"

]

]

},

{

"id": "79f258695d9dbe25",

"type": "function",

"z": "28e25aef1a915684",

"name": "db wind speed",

"func": "msg.temp = msg.payload;\nmsg.topic = \"INSERT INTO sensordata (sensorid,topic,value) VALUES ('QMUL-WS-02','/QMUL/CSEE/B312/WS',\"+ msg.temp +\");\"\nreturn msg;",

"outputs": 1,

"noerr": 0,

"initialize": "",

"finalize": "",

"libs": [],

"x": 1340,

"y": 860,

"wires": [

[]

]

},

{

"id": "c881850a3edfc153",

"type": "function",

"z": "28e25aef1a915684",

"name": "db wind direction",

"func": "msg.temp = msg.payload;\nmsg.topic = \"INSERT INTO sensordata (sensorid,topic,value) VALUES ('QMUL-WS-02','/QMUL/CSEE/B312/WD',\"+ msg.temp +\");\"\nreturn msg;",

"outputs": 1,

"noerr": 0,

"initialize": "",

"finalize": "",

"libs": [],

"x": 1350,

"y": 780,

"wires": [

[]

]

},

{

"id": "31f9671db406cad8",

"type": "mysql",

"z": "28e25aef1a915684",

"d": true,

"mydb": "6fa0bf3246bf4240",

"name": "db",

"x": 1290,

"y": 140,

"wires": [

[]

]

},

{

"id": "c4cc708d6c780bb4",

"type": "debug",

"z": "28e25aef1a915684",

"name": "",

"active": false,

"tosidebar": true,

"console": false,

"tostatus": false,

"complete": "false",

"statusVal": "",

"statusType": "auto",

"x": 570,

"y": 440,

"wires": []

},

{

"id": "ca50dae7ae6be9fe",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 5,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "LIGHT SENSOR LDR",

"label": "mm",

"format": "{{value}}",

"min": 0,

"max": "500",

"colors": [

"#ffffff",

"#b7b7b4",

"#000000"

],

"seg1": "",

"seg2": "",

"className": "",

"x": 1100,

"y": 1020,

"wires": []

},

{

"id": "0b276660d331ed05",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 6,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "NH3",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "10000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "500",

"seg2": "1000",

"className": "",

"x": 1050,

"y": 420,

"wires": []

},

{

"id": "75258836bf47e266",

"type": "function",

"z": "28e25aef1a915684",

"name": "db rain",

"func": "msg.temp = msg.payload;\nmsg.topic = \"INSERT INTO sensordata (sensorid,topic,value) VALUES ('QMUL-WS-02','/QMUL/CSEE/B312/RR',\"+ msg.temp +\");\"\nreturn msg;",

"outputs": 1,

"noerr": 0,

"initialize": "",

"finalize": "",

"libs": [],

"x": 1310,

"y": 820,

"wires": [

[]

]

},

{

"id": "1ffb91f73aa41ff8",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 7,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "CO",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "5000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "500",

"seg2": "1000",

"className": "",

"x": 1050,

"y": 500,

"wires": []

},

{

"id": "8eccd04884396363",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 8,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "NO2",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "2000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "500",

"seg2": "1000",

"className": "",

"x": 1050,

"y": 580,

"wires": []

},

{

"id": "1098f535d7d3acb7",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 12,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "C3H8",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "100000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "500",

"seg2": "1000",

"className": "",

"x": 1050,

"y": 660,

"wires": []

},

{

"id": "e1717b3423bdc6ba",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 16,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "C4H10",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "100000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "500",

"seg2": "1000",

"className": "",

"x": 1060,

"y": 940,

"wires": []

},

{

"id": "e97a3fe91ed291a3",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 10,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "CH4",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "1000000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "500",

"seg2": "1000",

"className": "",

"x": 1050,

"y": 700,

"wires": []

},

{

"id": "15d1c0b922b3a3c7",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 9,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "H2",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "10000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "2500",

"seg2": "5000",

"className": "",

"x": 1050,

"y": 740,

"wires": []

},

{

"id": "3f52d7901f0964de",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 11,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "C2H5OH",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "100000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "10000",

"seg2": "50000",

"className": "",

"x": 1060,

"y": 860,

"wires": []

},

{

"id": "80bdacd875c36bde",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "dbfc4352185bf301",

"order": 1,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "WIND SPEED",

"label": "KM/H",

"format": "{{value}}",

"min": 0,

"max": "100",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "25",

"seg2": "50",

"className": "",

"x": 1080,

"y": 1100,

"wires": []

},

{

"id": "2e5260d5c88d94df",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "",

"group": "dbfc4352185bf301",

"order": 2,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "RAIN RATE",

"label": "mm",

"format": "{{value}}",

"min": 0,

"max": "100",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "25",

"seg2": "50",

"className": "",

"x": 1070,

"y": 1140,

"wires": []

},

{

"id": "b85baaab3b5a3686",

"type": "serial in",

"z": "28e25aef1a915684",

"d": true,

"name": "",

"serial": "a8e3dcb738349c22",

"x": 90,

"y": 700,

"wires": [

[

"0d26401aecc07fbf"

]

]

},

{

"id": "89ec97a15d475c17",

"type": "ui\_text",

"z": "28e25aef1a915684",

"group": "dbfc4352185bf301",

"order": 3,

"width": "5",

"height": "1",

"name": "",

"label": "WD",

"format": "{{msg.payload}}",

"layout": "row-spread",

"className": "",

"x": 1050,

"y": 1180,

"wires": []

},

{

"id": "42719b42f62f706b",

"type": "serial in",

"z": "28e25aef1a915684",

"d": true,

"name": "",

"serial": "7dcbac4abf604cad",

"x": 90,

"y": 300,

"wires": [

[

"3659f57cc450c3d4"

]

]

},

{

"id": "7f40901e29151bc2",

"type": "debug",

"z": "28e25aef1a915684",

"name": "",

"active": false,

"tosidebar": true,

"console": false,

"tostatus": false,

"complete": "false",

"statusVal": "",

"statusType": "auto",

"x": 1070,

"y": 1220,

"wires": []

},

{

"id": "d8db4057889d19cc",

"type": "debug",

"z": "28e25aef1a915684",

"name": "",

"active": false,

"tosidebar": true,

"console": false,

"tostatus": false,

"complete": "false",

"statusVal": "",

"statusType": "auto",

"x": 1070,

"y": 1060,

"wires": []

},

{

"id": "0d26401aecc07fbf",

"type": "split",

"z": "28e25aef1a915684",

"name": "",

"splt": ",",

"spltType": "str",

"arraySplt": 1,

"arraySpltType": "len",

"stream": false,

"addname": "",

"x": 230,

"y": 700,

"wires": [

[

"a930ea0d562d2d6c"

]

]

},

{

"id": "c860d430dd0b0279",

"type": "change",

"z": "28e25aef1a915684",

"name": "to number",

"rules": [

{

"t": "set",

"p": "payload",

"pt": "msg",

"to": "$number(msg.payload)",

"tot": "num"

},

{

"t": "set",

"p": "payload",

"pt": "msg",

"to": "$String(msg.payload)",

"tot": "str"

}

],

"action": "",

"property": "",

"from": "",

"to": "",

"reg": false,

"x": 380,

"y": 700,

"wires": [

[

"6f481210fd4a2bcd"

]

]

},

{

"id": "a930ea0d562d2d6c",

"type": "switch",

"z": "28e25aef1a915684",

"name": "Message Splitter SN",

"property": "parts.index",

"propertyType": "msg",

"rules": [

{

"t": "eq",

"v": "0",

"vt": "num"

},

{

"t": "eq",

"v": "1",

"vt": "str"

},

{

"t": "eq",

"v": "2",

"vt": "str"

},

{

"t": "eq",

"v": "3",

"vt": "str"

},

{

"t": "eq",

"v": "4",

"vt": "str"

},

{

"t": "eq",

"v": "5",

"vt": "str"

},

{

"t": "eq",

"v": "6",

"vt": "str"

},

{

"t": "eq",

"v": "7",

"vt": "str"

},

{

"t": "eq",

"v": "8",

"vt": "str"

},

{

"t": "eq",

"v": "9",

"vt": "str"

},

{

"t": "eq",

"v": "10",

"vt": "str"

},

{

"t": "eq",

"v": "11",

"vt": "str"

},

{

"t": "eq",

"v": "12",

"vt": "str"

}

],

"checkall": "true",

"repair": true,

"outputs": 13,

"x": 580,

"y": 700,

"wires": [

[

"d8db4057889d19cc",

"2e5260d5c88d94df"

],

[

"d8db4057889d19cc",

"80bdacd875c36bde"

],

[

"89ec97a15d475c17",

"d8db4057889d19cc"

],

[],

[],

[],

[],

[],

[],

[],

[],

[],

[]

]

},

{

"id": "e2d5c61516032c70",

"type": "comment",

"z": "28e25aef1a915684",

"name": "split",

"info": "Split comma separated string into separated messages.\nValue placed to msg.payload\nValue type will be still string.\n\nThere will be 4 messages out of this split node\n\nFor each msg also given a property \"parts\". \nIt will be used later on.",

"x": 230,

"y": 640,

"wires": []

},

{

"id": "1bd4f278d3533aef",

"type": "comment",

"z": "28e25aef1a915684",

"name": "convert",

"info": "Convert the value of msg.payload to the number using JSONata expression.\n",

"x": 370,

"y": 640,

"wires": []

},

{

"id": "a25b15bb744ef2c0",

"type": "comment",

"z": "28e25aef1a915684",

"name": "route",

"info": "Route every message to differet output by using switch node.\nAs property \"parts\" has been given to each message, we can use its \n\"index\" property to find out proper output\n\n",

"x": 630,

"y": 560,

"wires": []

},

{

"id": "6f481210fd4a2bcd",

"type": "debug",

"z": "28e25aef1a915684",

"name": "",

"active": true,

"tosidebar": true,

"console": false,

"tostatus": false,

"complete": "false",

"statusVal": "",

"statusType": "auto",

"x": 570,

"y": 840,

"wires": []

},

{

"id": "99cf44cd9e5b4e80",

"type": "ui\_chart",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 13,

"width": 0,

"height": 0,

"label": "NH3",

"chartType": "line",

"legend": "false",

"xformat": "HH:mm:ss",

"interpolate": "linear",

"nodata": "",

"dot": false,

"ymin": "",

"ymax": "",

"removeOlder": 1,

"removeOlderPoints": "",

"removeOlderUnit": "86400",

"cutout": 0,

"useOneColor": false,

"useUTC": false,

"colors": [

"#1f77b4",

"#aec7e8",

"#ff7f0e",

"#2ca02c",

"#98df8a",

"#d62728",

"#ff9896",

"#9467bd",

"#c5b0d5"

],

"outputs": 1,

"useDifferentColor": false,

"className": "",

"x": 1050,

"y": 460,

"wires": [

[]

]

},

{

"id": "d6a708f715d5612a",

"type": "ui\_chart",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 14,

"width": 0,

"height": 0,

"label": "CO",

"chartType": "line",

"legend": "false",

"xformat": "HH:mm:ss",

"interpolate": "linear",

"nodata": "",

"dot": false,

"ymin": "",

"ymax": "",

"removeOlder": 1,

"removeOlderPoints": "",

"removeOlderUnit": "86400",

"cutout": 0,

"useOneColor": false,

"useUTC": false,

"colors": [

"#1f77b4",

"#aec7e8",

"#ff7f0e",

"#2ca02c",

"#98df8a",

"#d62728",

"#ff9896",

"#9467bd",

"#c5b0d5"

],

"outputs": 1,

"useDifferentColor": false,

"className": "",

"x": 1050,

"y": 540,

"wires": [

[]

]

},

{

"id": "a726ccf8fbcdde4a",

"type": "ui\_chart",

"z": "28e25aef1a915684",

"name": "",

"group": "4fa11ce7444b6cb0",

"order": 15,

"width": 0,

"height": 0,

"label": "NO2",

"chartType": "line",

"legend": "false",

"xformat": "HH:mm:ss",

"interpolate": "linear",

"nodata": "",

"dot": false,

"ymin": "",

"ymax": "",

"removeOlder": 1,

"removeOlderPoints": "",

"removeOlderUnit": "86400",

"cutout": 0,

"useOneColor": false,

"useUTC": false,

"colors": [

"#1f77b4",

"#aec7e8",

"#ff7f0e",

"#2ca02c",

"#98df8a",

"#d62728",

"#ff9896",

"#9467bd",

"#c5b0d5"

],

"outputs": 1,

"useDifferentColor": false,

"className": "",

"x": 1050,

"y": 620,

"wires": [

[]

]

},

{

"id": "e6c27e211599cfe6",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "Dust Sensor (PPD42NS) ",

"group": "4fa11ce7444b6cb0",

"order": 1,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "Dust Sensor (PPD42NS) ",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "10000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "10000",

"seg2": "50000",

"className": "",

"x": 1110,

"y": 980,

"wires": []

},

{

"id": "58607cd131887da7",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "UV Index (SI1145)",

"group": "4fa11ce7444b6cb0",

"order": 2,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "UV Index (SI1145)",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "15",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "10000",

"seg2": "50000",

"className": "",

"x": 1090,

"y": 820,

"wires": []

},

{

"id": "43e9629ca443f408",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "Visible Light (SI1145)",

"group": "4fa11ce7444b6cb0",

"order": 4,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "Visible Light (SI1145)",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "5000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "1000",

"seg2": "2500",

"className": "",

"x": 1100,

"y": 780,

"wires": []

},

{

"id": "f3845eaa1c090471",

"type": "ui\_gauge",

"z": "28e25aef1a915684",

"name": "IR Sensor (SI1145)",

"group": "4fa11ce7444b6cb0",

"order": 3,

"width": "5",

"height": "5",

"gtype": "gage",

"title": "IR Sensor (SI1145)",

"label": "ppm",

"format": "{{value}}",

"min": 0,

"max": "5000",

"colors": [

"#00b500",

"#e6e600",

"#ca3838"

],

"seg1": "1000",

"seg2": "2500",

"className": "",

"x": 1090,

"y": 900,

"wires": []

},

{

"id": "cdb1e6d07c5ef848",

"type": "ui\_group",

"name": "BME-680",

"tab": "58d5fa7d25fe03f8",

"order": 2,

"disp": true,

"width": "10",

"collapse": true,

"className": ""

},

{

"id": "6fa0bf3246bf4240",

"type": "MySQLdatabase",

"name": "Local Database",

"host": "127.0.0.1",

"port": "3306",

"db": "iotdata\_db",

"tz": "00:00",

"charset": "UTF8"

},

{

"id": "4fa11ce7444b6cb0",

"type": "ui\_group",

"name": "MULTI GAS SENSORS",

"tab": "58d5fa7d25fe03f8",

"order": 1,

"disp": true,

"width": "10",

"collapse": true,

"className": ""

},

{

"id": "dbfc4352185bf301",

"type": "ui\_group",

"name": "WEATHER STATION",

"tab": "58d5fa7d25fe03f8",

"order": 3,

"disp": true,

"width": "10",

"collapse": true,

"className": ""

},

{

"id": "a8e3dcb738349c22",

"type": "serial-port",

"serialport": "COM14",

"serialbaud": "9600",

"databits": "8",

"parity": "none",

"stopbits": "1",

"waitfor": "",

"dtr": "none",

"rts": "none",

"cts": "none",

"dsr": "none",

"newline": "\\n",

"bin": "false",

"out": "char",

"addchar": "",

"responsetimeout": "10000"

},

{

"id": "7dcbac4abf604cad",

"type": "serial-port",

"serialport": "COM18",

"serialbaud": "9600",

"databits": "8",

"parity": "none",

"stopbits": "1",

"waitfor": "",

"dtr": "none",

"rts": "none",

"cts": "none",

"dsr": "none",

"newline": "\\n",

"bin": "false",

"out": "char",

"addchar": "",

"responsetimeout": "10000"

},

{

"id": "58d5fa7d25fe03f8",

"type": "ui\_tab",

"name": "QMUL LL PROJECT (WEATHER STATION)",

"icon": "dashboard",

"order": 3,

"disabled": false,

"hidden": false

}

]