ENVIRONMENTAL MONITORING

PHASE 3:DEVELOPMENT PART 1 **ABSTRACT:**

* We had some Environment click sensors handy, so we decided to hook them up to the Arduino MKR1000 and visualise them on WolkAbout IoT Platform.
* The idea was to do a measurement every minute and publish the results once every 15 minutes.
* If the publishing of sensor readings fails (due to a busy network or some other issue), then the results should be persisted in the Flash memory of the device, With a potential maximum of 96 writes a day.

**PYTHON SCRIPT:**

**import time import random def read\_sensor\_data():**

**temperature = random.uniform(10, 40) humidity = random.uniform(20, 80) return temperature, humidity def process\_data(temperature, humidity): print(f"Temperature: {temperature}°C") print(f"Humidity: {humidity}%") while True:**

**temperature, humidity = read\_sensor\_data() process\_data(temperature, humidity) time.sleep(10)**

**ARDUINO UNO R3:**

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BME680.h>

#include <bme680\_defs.h>

#include <bme680.h>

#include <WiFi101.h>

#include <RTCZero.h>

#include <FlashStorage.h>

#include "WolkConn.h"

#include "MQTTClient.h"

/\*Number of outbound\_message\_t to store\*/

#define STORAGE\_SIZE 32

#define SEALEVELPRESSURE\_HPA (1013.25)

/\*Circular buffer to store outbound messages to persist\*/ typedef struct

{ boolean valid;

outbound\_message\_t outbound\_messages[STORAGE\_SIZE]; uint32\_t head; uint32\_t tail; boolean empty; boolean full; } Messages; static Messages data; /\*Connection details\*/ const char\* ssid = "ssid"; const char\* wifi\_pass = "wifi\_pass"; const char \*device\_key = "device\_key"; const char \*device\_password = "device\_password"; const char\* hostname = "api-demo.wolkabout.com";

int portno = 1883; WiFiClient espClient;

PubSubClient client(espClient);

/\* WolkConnect-Arduino Connector context \*/ static wolk\_ctx\_t wolk; /\* Init flash storage \*/

FlashStorage(flash\_store, Messages);

/\*Init i2c sensor communication\*/

Adafruit\_BME680 bme; RTCZero rtc; bool read;

/\*Read sensor every minute. If you change this parameter make sure that it's <60\*/ const byte readEvery = 1; bool publish;

/\*Publish every 10 minutes. If you change this parameter make sure that it's <60\*/ const byte publishEvery = 10; byte publishMin;

/\*Flash storage and custom persistence implementation\*/ void \_flash\_store()

{ data.valid = true; flash\_store.write(data);

}

void increase\_pointer(uint32\_t\* pointer)

{

if ((\*pointer) == (STORAGE\_SIZE - 1))

{

(\*pointer) = 0;

} else

{

(\*pointer)++;

} }

void \_init()

{ data = flash\_store.read(); if (data.valid == false)

{ data.head = 0; data.tail = 0; data.empty = true; data.full = false;

} }

bool \_push(outbound\_message\_t\* outbound\_message)

{

if(data.full)

{

increase\_pointer(&data.head);

}

memcpy(&data.outbound\_messages[data.tail], outbound\_message, sizeof(outbound\_message\_t)); increase\_pointer(&data.tail); data.empty = false;

data.full = (data.tail == data.head); return true;

}

bool \_peek(outbound\_message\_t\* outbound\_message)

{

memcpy(outbound\_message, &data.outbound\_messages[data.head], sizeof(outbound\_message\_t)); return true;

}

bool \_pop(outbound\_message\_t\* outbound\_message)

{

memcpy(outbound\_message, &data.outbound\_messages[data.head], sizeof(outbound\_message\_t)); increase\_pointer(&data.head); data.full = false;

data.empty = (data.tail == data.head); return true;

}

bool \_is\_empty()

{

return data.empty;

}

void init\_wifi()

{

if ( WiFi.status() != WL\_CONNECTED) {

while (WiFi.begin(ssid, wifi\_pass) != WL\_CONNECTED) { delay(1000);

}

} }

void setup\_wifi()

{ delay(10);

if ( WiFi.status() != WL\_CONNECTED) { int numAttempts = 0;

while (WiFi.begin(ssid, wifi\_pass) != WL\_CONNECTED) { numAttempts++; if(numAttempts == 10){

Serial.println("Couldn't reach WiFi!"); break; }

delay(1000);

}

}

} void setup() {

pinMode(LED\_BUILTIN, OUTPUT); digitalWrite(LED\_BUILTIN, LOW);

/\*Initialize the circular buffer structure\*/

\_init(); init\_wifi();

wolk\_init(&wolk, NULL, NULL, NULL, NULL,

device\_key, device\_password, &client, hostname, portno, PROTOCOL\_JSON\_SINGLE, NULL, NULL);

wolk\_init\_custom\_persistence(&wolk, \_push, \_peek, \_pop,

\_is\_empty);

/\*The on board LED will turn on if something went wrong\*/ if(!bme.begin())

{

digitalWrite(LED\_BUILTIN, HIGH); }

/\*Sensor init\*/

bme.setTemperatureOversampling(BME680\_OS\_8X); bme.setHumidityOversampling(BME680\_OS\_2X); bme.setPressureOversampling(BME680\_OS\_4X); bme.setIIRFilterSize(BME680\_FILTER\_SIZE\_3); bme.setGasHeater(320, 150); // 320\*C for 150 ms delay(200); read = true; publish = true;

/\*Get current epoch from server\*/ wolk\_connect(&wolk); delay(100); wolk\_update\_epoch(&wolk); while (!(wolk.pong\_received)) { wolk\_process(&wolk); digitalWrite(LED\_BUILTIN, HIGH); delay(1000); } digitalWrite(LED\_BUILTIN, LOW); wolk\_disconnect(&wolk); rtc.begin();

rtc.setEpoch(wolk.epoch\_time);

rtc.setAlarmTime(rtc.getHours(), (rtc.getMinutes() + readEvery) % 60, rtc.getSeconds()); rtc.enableAlarm(rtc.MATCH\_MMSS); rtc.attachInterrupt(alarmMatch);

publishMin = (rtc.getMinutes() + publishEvery) % 60;

WiFi.lowPowerMode();

} void loop() { if(read) { read = false; if (!bme.performReading()) { digitalWrite(LED\_BUILTIN, HIGH); } wolk\_add\_numeric\_sensor\_reading(&wolk, "T", bme.temperature, rtc.getEpoch());

wolk\_add\_numeric\_sensor\_reading(&wolk, "H", bme.humidity, rtc.getEpoch());

wolk\_add\_numeric\_sensor\_reading(&wolk, "P", bme.pressure /

100.0, rtc.getEpoch());

wolk\_add\_numeric\_sensor\_reading(&wolk, "GR", bme.gas\_resistance, rtc.getEpoch()); wolk\_add\_numeric\_sensor\_reading(&wolk, "A",

bme.readAltitude(SEALEVELPRESSURE\_HPA), rtc.getEpoch());

/\*set new alarm\*/

int alarmMin = (rtc.getMinutes() + readEvery) % 60; rtc.setAlarmMinutes(alarmMin); delay(100);

} if(publish)

{ publish = false; setup\_wifi(); wolk\_connect(&wolk); if(!wolk.is\_connected)

{

flash\_store();

} delay(100);

if(wolk\_publish(&wolk) == W\_TRUE)

{

\_flash\_store();

}

/\*set new publish time\*/

publishMin = (rtc.getMinutes() + publishEvery) % 60; delay(100); wolk\_disconnect(&wolk); delay(100);

} delay(100);

/\*Timed interrupt routine\*/ void alarmMatch()

{

read = true;

if(publishMin == rtc.getMinutes())

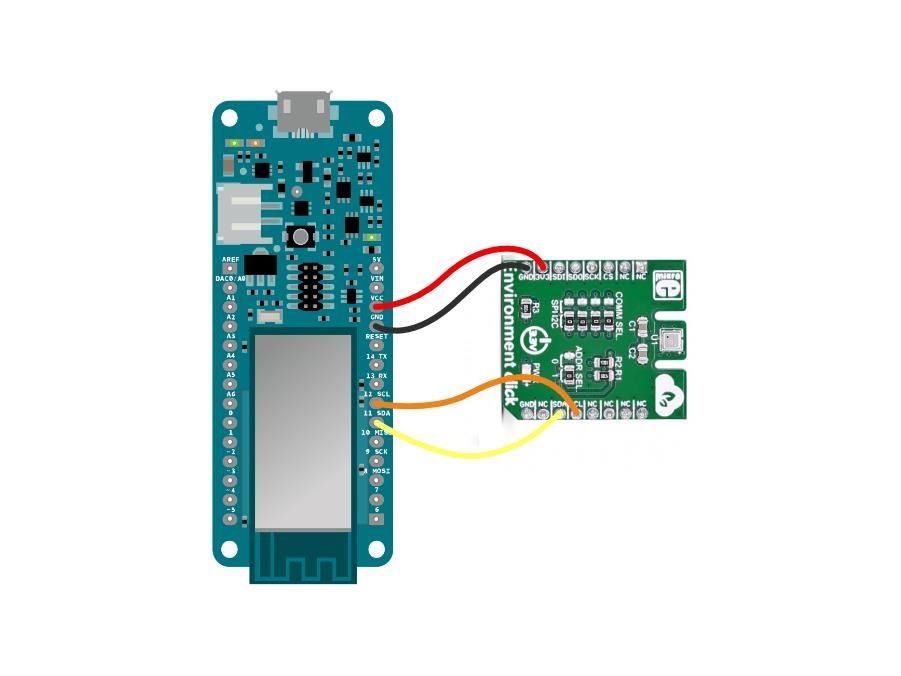
{

publish = true;

}

}

IOT DEVICE:



THESE CODE ARE THEROY ARE INCLUDED IN

PHASE3:ENVIRONMENTAL MONITORING

**BY :**

**V.Velmurugan (422621104301)**