Thomas More Kempen

Individual Project

IT essentials 1ITF

Joris Van Puyenbroeck
r-number: m9407046

CAMPUS

Geel

Technology
Elektronics-ICT / Applied informatics

IT essentials

Course unit: IT essentials

Educational activity: IT essentials

First tier

ASSOCIATIE K.U.LEUVEN

Academiejaar 2020-2021

Word in advance

In this paper I would like to present a DIY small scale ‘weather station’, based on the ESP32 platform.

Before this assignment I had some, but albeit little experience with development boards like the ESP. I recently built a WiFi scanner with an ESP8286, but that project was mainly just uploading a sketch and using it.

I have some experience in building simple Arduino circuits, so I was already familiar with the Arduino IDE. I also had some extra ESP32’s and some cheap spare Chinese sensors laying around. One of my personal pet projects “in the works” is an IoT Power consumption wireless network, with Ampere power sensors, hooked to very small ESP8286’s. I’ve not yet started that project, due to lack of time, and not enough skills to progress.

This assignment improved my skills significantly: I learned the basics of C and was able to customize the scripts that I found. I have to admit I’m not a C programmer yet. It’s more that I’m able now to get the right headers, functions and adapt them for my project. This is just the beginning, but it seems the IoT virus has bitten me!

I used the Arduino IDE for this project due to some early troubles with PlatformIO and VSCode. During one of the early steps I deleted some headers and configuration settings, and for a long period of time, I was not able to get a stable compiling environment, especially after installing ‘vcpkg’ . Thankfully, I got it working again.

I think I succeeded in reaching the highest difficult level for this assignment. In this paper I’ll describe and demonstrate all steps I took before completing the final two sketches. I’ll demonstrate how the two boards work and what technologies were used.

The ‘making-of video’ can be found here: https://youtu.be/C1oz8z5ogCc

Table of contents

Content

List of illustrations

List of abbreviations and symbols used

I2C	Inter Integrated Circuit
MCU	micro controller unit
MQTT	Message Queuing Telemetry Transport
NAS	Network attached Storage
SCL	Serial Clock Line
SDA	Serial Data Line
SQL	Structured Query Language
PHP	Hypertext Preprcocessor

1 Description of the project

1.1 Target

We were tasked to build a weather station based on an ESP32 WSROOM micro controller unit (MCU) board (Espressif) and add two or more I²C sensors.

1.2 Preliminary steps

- During class
  - I installed PlatformIo on VSCode
  - I connected the wires as shown on the provided breadboard
  - I got the blink led sketch working.
- At home
  - I downloaded and read all the available documentation on Canvas
  - I downloaded the suggested scripts and saved them as 'building blocks' for the adapted, working scripts
  - I got all kinds of materials from my workshop like 'jumper wires'

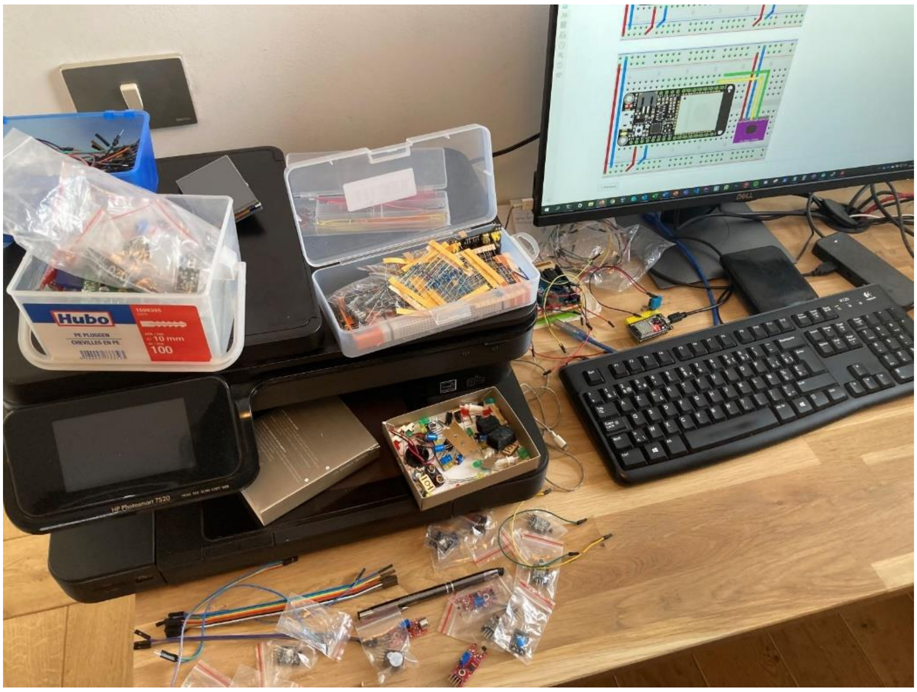


Figure 1 My workbench

1.3 Wiring

ESP32 Espressif with BMP280 temperature and pressor sensor and BH1750 light sensor.

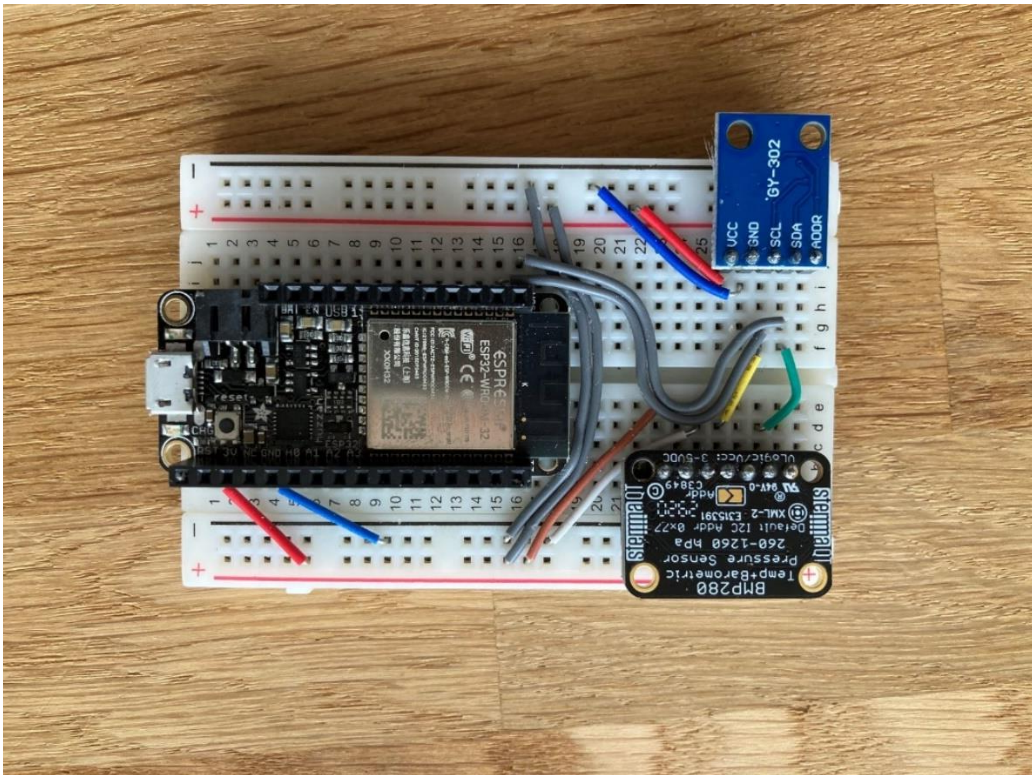


Figure 2 Jumper wiring

Figure 2 shows the wiring of this setup. Aside from 3,3v and GND, we connect the SCL and SDA of both sensors to the correct pins of the I2C bus on the ESP32 board (top right). The jumper wires have a fixed length, so they had to be bent. But jumper wires still look more 'tidy' than 'off the shelf' breadboard wires.

Due to some alignment issues in Fritzing, the wiring doesn't reflect the true wiring as can be seen in Figure 3. The pins of the BMP280 sensor didn't exactly line up with the breadboard holes after a 180 rotation, so some extra wires were added in the diagram (not on the board itself).

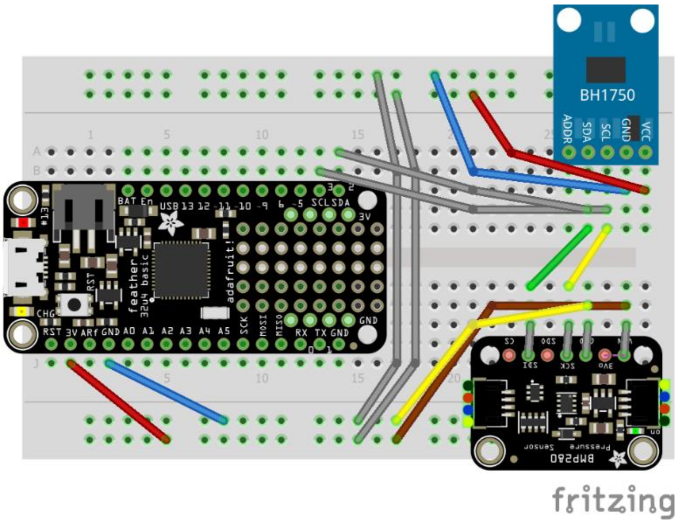


Figure 3 Fritzing breadboard wire diagram

1.4 Block diagram

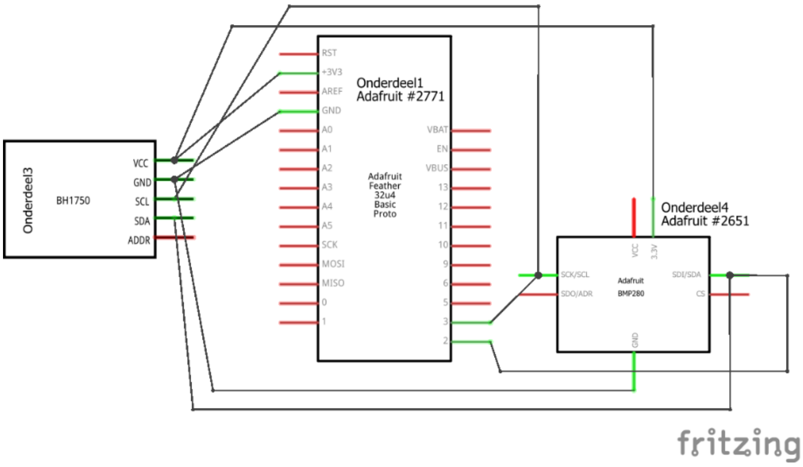


Figure 4 Breadbord block diagram setup 1

1.5 Code

We provide the code for “Visualization of current and historical data by communication over MQTT” in paragraph 2. Before we do that, we give an oversight of the different coding steps I took, to get to that point.

1.5.1 Building blocks

Table 1 Some building blocks for the code

Sketch	Tested functionality	Technology, headers, snippets of important code	
sketch_esp32_blink	- Serial Com - Upload - Building environment - Led blinking	- Arduino.h	☑
sketch_bmp280_BH1750_serial	- Two sensors connected - Serial output 3 sensor readings in void loop	- Wire.h - Adafruit_bmp280.h - BH1750.h	☑
sketch_esp_dash_BMP280_BH1750 (local webserver on ESP32)	- WiFi ESP32 - Asynch webserver - Dash webpage in void loop	- Wifi.h - asyncTCP.h - ESPAsyncWebServer.h - ESPdash.h	☑
sketch_esp_dash_3sensors_deepsleep	- Deep sleep and wake up by timer	- #define TIME_TO_SLEEP - esp_deep_sleep_start()	☑
sketch_SQl_sinners_3sensors_deepsleep	- httpconnection - PHP code to write to MySQL database on db.sinners.be - https://jorisvp.sinners.be/ITessentials/esp_chart.php	- HTTPClient.h - http.POST(httpRequestData) - (PHP) $conn = new mysqli($servername, $username, $password, $dbname) - $conn->query($sql)	☑
sketch_mqtt_light	- MQTT with one sensor - Output to Thingspeak -	- Timelib.h - Pubsubclient.h - client.setServer(mqtt_server, 1883 - client.publish(outTopic, attributes)	☑
sketch_station2_DHT11_BH1750_MQTT	- second board to safely test the MQTT approach - DHT11 Aliexpress sensor (not i2c) - Output to local and remote MQTT broker - First tests with local MQTT server on Docker NAS Synology 918+	- DHT.h - Mosquitto (local MQTT) Synology DSM package	☑
sketch_weatherstation_wittenberg	- Integration of all steps - Connection to sinners.be - Connection to MQTT broker - Node Red on local server (Synology Docker) o Output to Thingspeak (remote) https://thingspeak.com/channels/1318816 o Output to local Node red chart (local) o Output to local InfluxDB - Deep sleep set to 30mins	- ArduinoJson.h - Docker (LAN) - Node Red (local webserver) - Msg.payload = () - JSON and Javascript objects - InfluxDB on Docker - Grafana on Docker - Telegraf (experiments)	☑

1.5.2 Visualization of current and historical data by communication over MQTT

The code can be found in paragraph 2. The idea is to centralize the ESP32 output in 3 MQTT channels, capture it with Node Red, and distribute it from there to various outputs.

```
client.publish("esp32_wittenberg/temperature", tempString);
client.publish("esp32_wittenberg/pressure", presString);
client.publish("esp32_wittenberg/light", lightString);
```

Those channels or subscribed to on a local Node Red server (Docker, port 1880, Synology NAS).

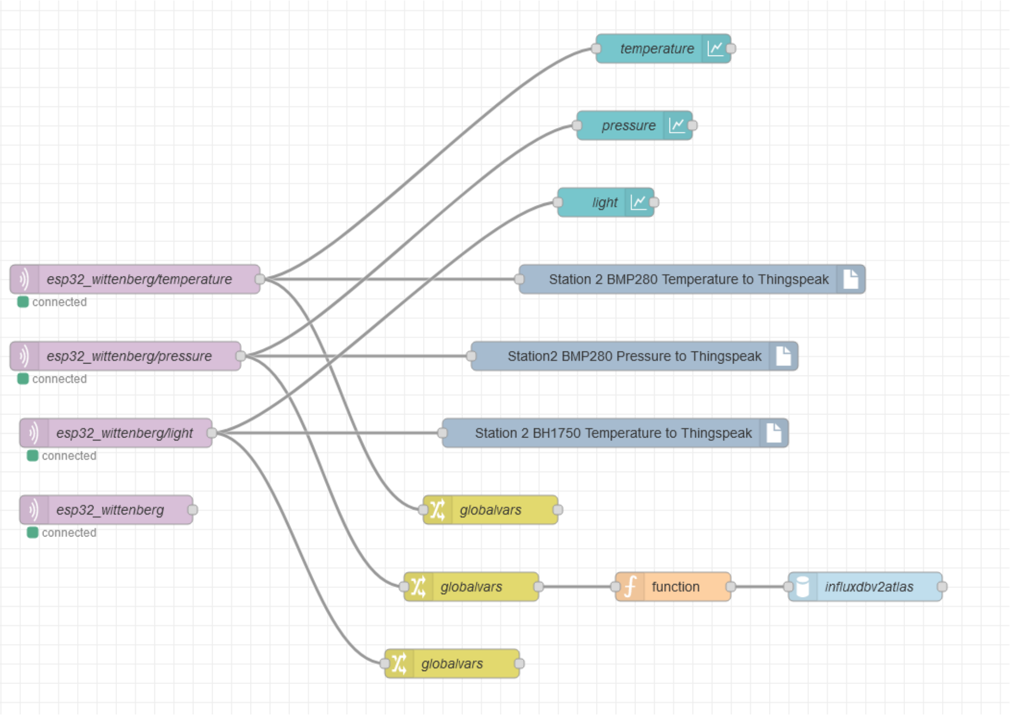
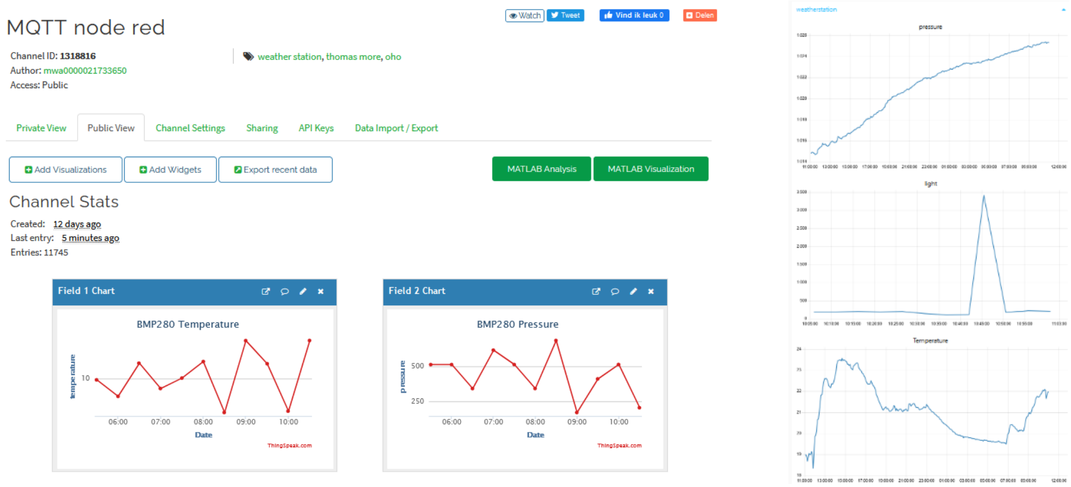

temperature
pressure
light
esp32_wittenberg/temperature
connected
Station 2 BMP280 Temperature to Thingspeak
esp32_wittenberg/pressure
connected
Station2 BMP280 Pressure to Thingspeak
esp32_wittenberg/light
connected
Station 2 BH1750 Temperature to Thingspeak
esp32_wittenberg
connected
globalvars
globalvars
function
influxdbv2atlas
globalvars

Figure 5 Node Red Flow Chart

The readings (msg.payload) are forwarded as numbers to Thingspeak (https://thingspeak.com/channels/1318816)



MQTT node red

Channel ID: 1318816

Author: mwa0000021733650

Access: Public

weather station, thomas more, oho

Channel Stats

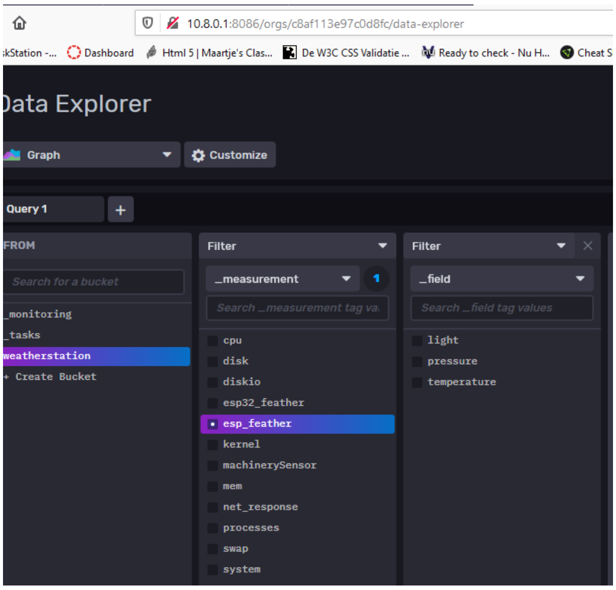
Created: 12 days ago

Last entry: 5 minutes ago

Entries: 11745

Figure 6 Output to Thingspeak and Node Red

The data is also sent to a Node Red Dashboard and to a local InfluxDB server (Docker, port 8086) in the bucket "weatherstation". The client and server operate on a VPN (10.8.0.1/24) network.



10.8.0.1:8086/orgs/c8af113e97c0d8fc/data-explorer

Figure 7 Data source local InfluxDB

I choose a local instance of InfluxDB because then I could experiment with all MQTT plugins , Telegraf and Grafana output. I took me quite a while to learn how to bridge MQTT to Flux. But now my Node Red code converts the msg.payload correctly to the InfluxDB format.

```
msg.payload = {
  bucket: 'weatherstation',
  precision: 'ms',
  data: [
    {
      measurement: 'esp_feather',

      tags: {
        deviceId: 'esp32_espressif'
      },

      fields: {
        temperature: global.get("globtemp"),
        pressure: global.get("globpres"),
        light: global.get("globlight")
      },

      timestamp: Date.now()
    },
  ]
};

return msg;
```

Meanwhile, the HTTP server on the ESP32 still outputs directly to https://jorisvp.sinners.be/ITessentials/esp_chart.php .

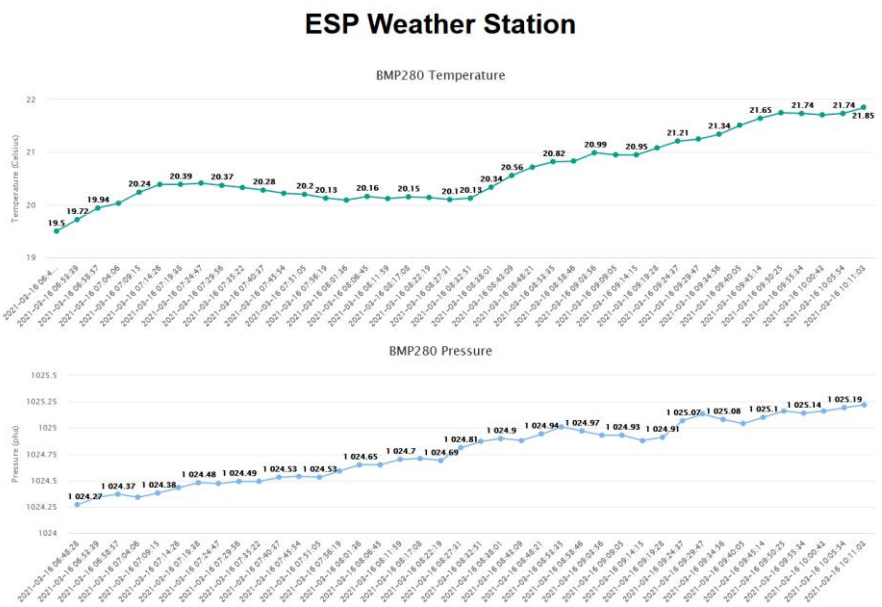


Figure 8 SQL-PHP generated output on sinners.be

Finally the output options are endless with Node Red and InfluxDB. To finish, I present an InfluxDB dashboard. This can be made publicly available as well, as I will show in the video.

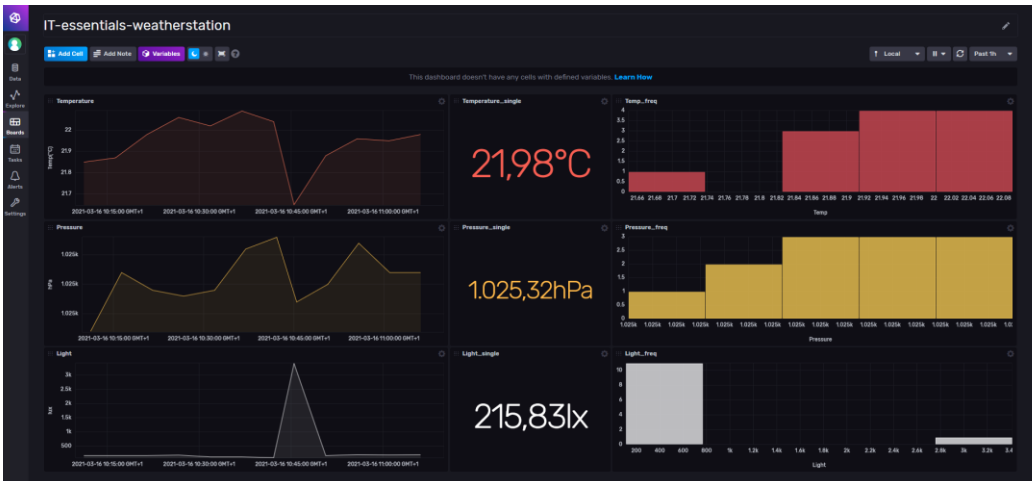


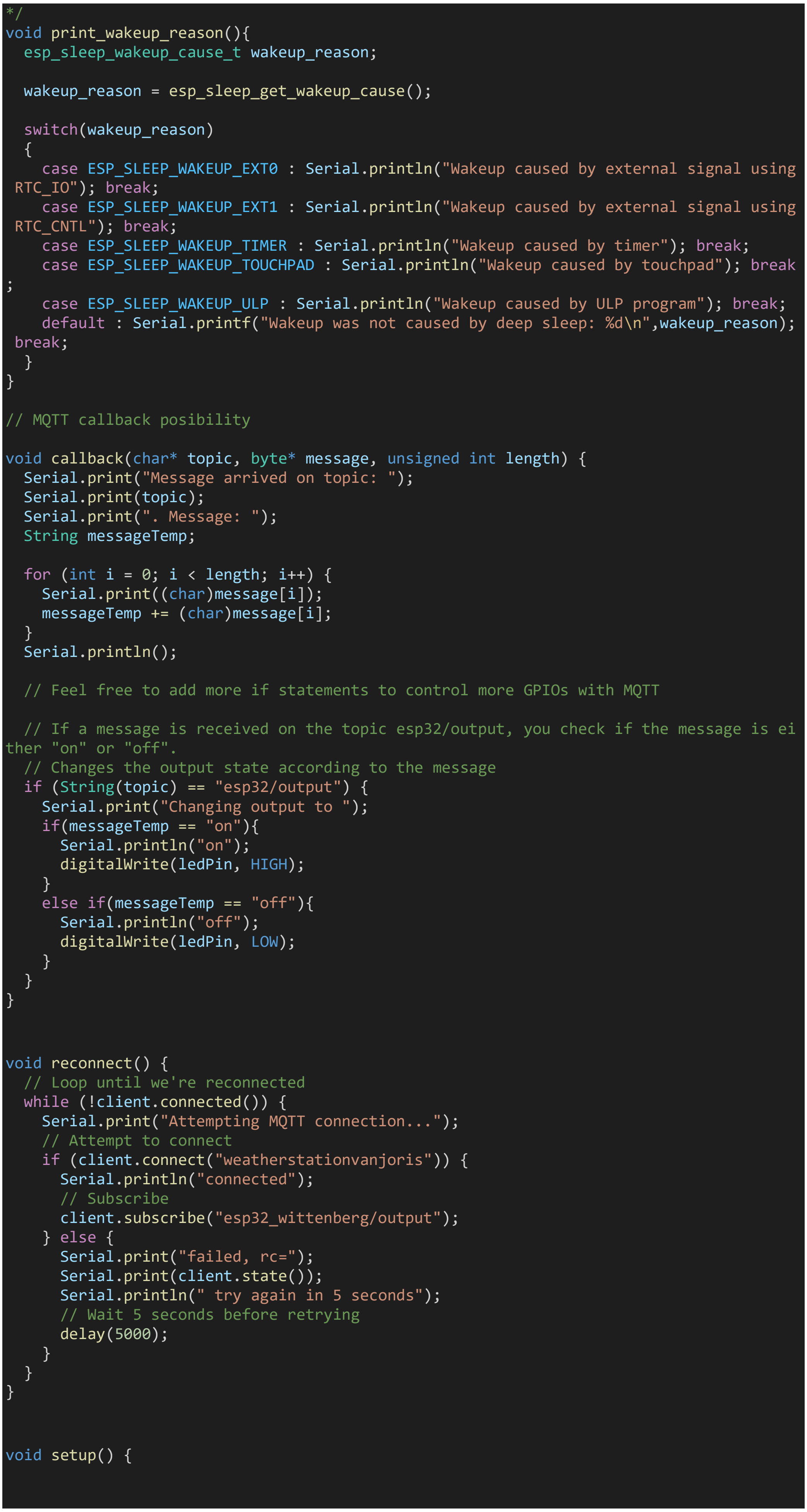
Figure 9 Output on InfluxDB

```
Boot number: 10
Wakeup caused by timer
Connecting
...
Connected to WiFi network with IP Address: 192.168.0.194
Temperature: 22.18 *C
Pressure: 1025.25 hPa
Light : 310.83 lx
httpRequestData: api_key=tPmAT5Ab3j7F9&value1=22.18&value2=1025.25&value3=310.83
HTTP Response code: 200
Attempting MQTT connection...connected
Setup ESP32 to sleep for every 300 Seconds
Going to sleep now
```

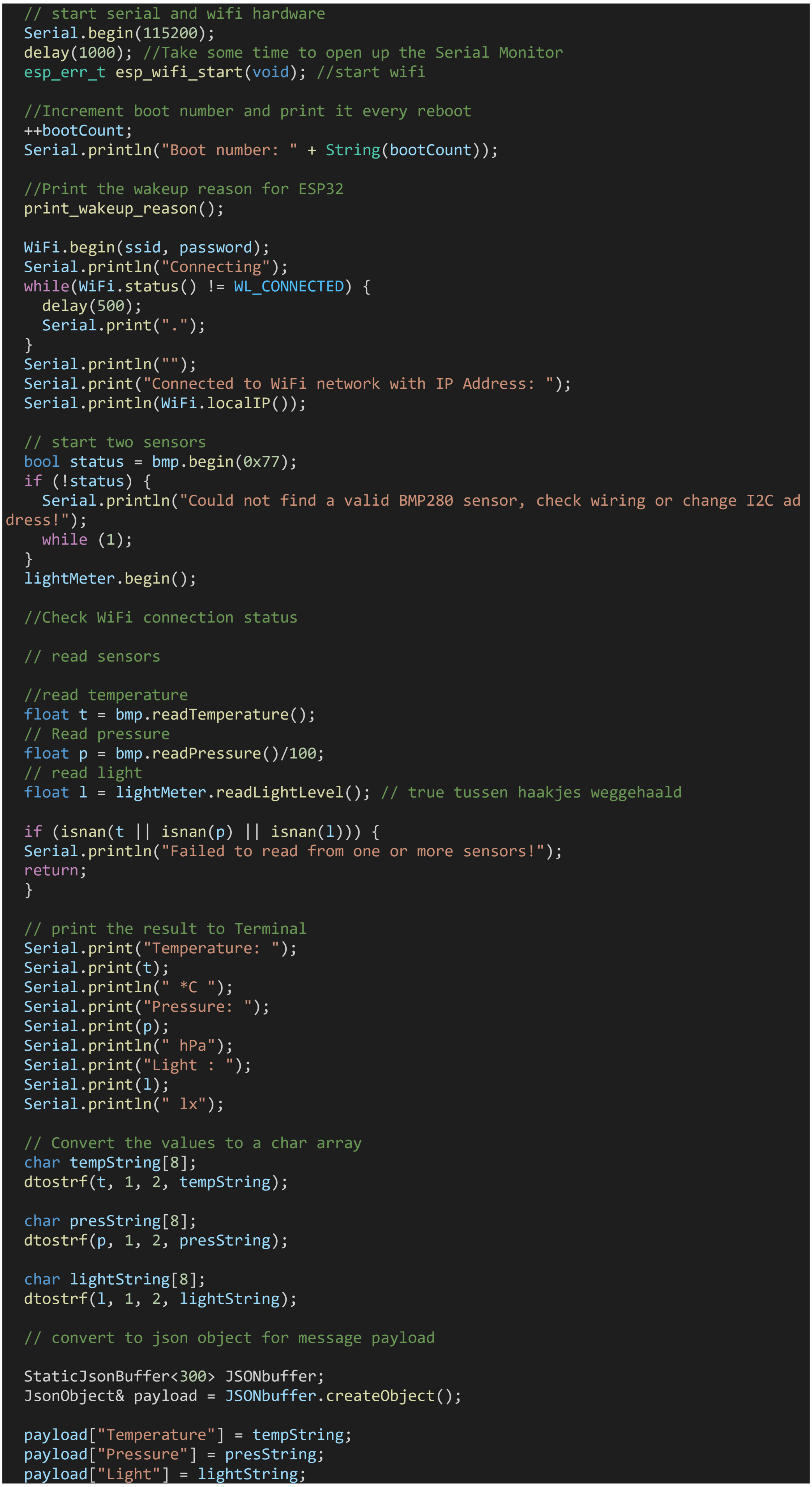
Figure 10 Serial output

2 Code (16/3/’21)

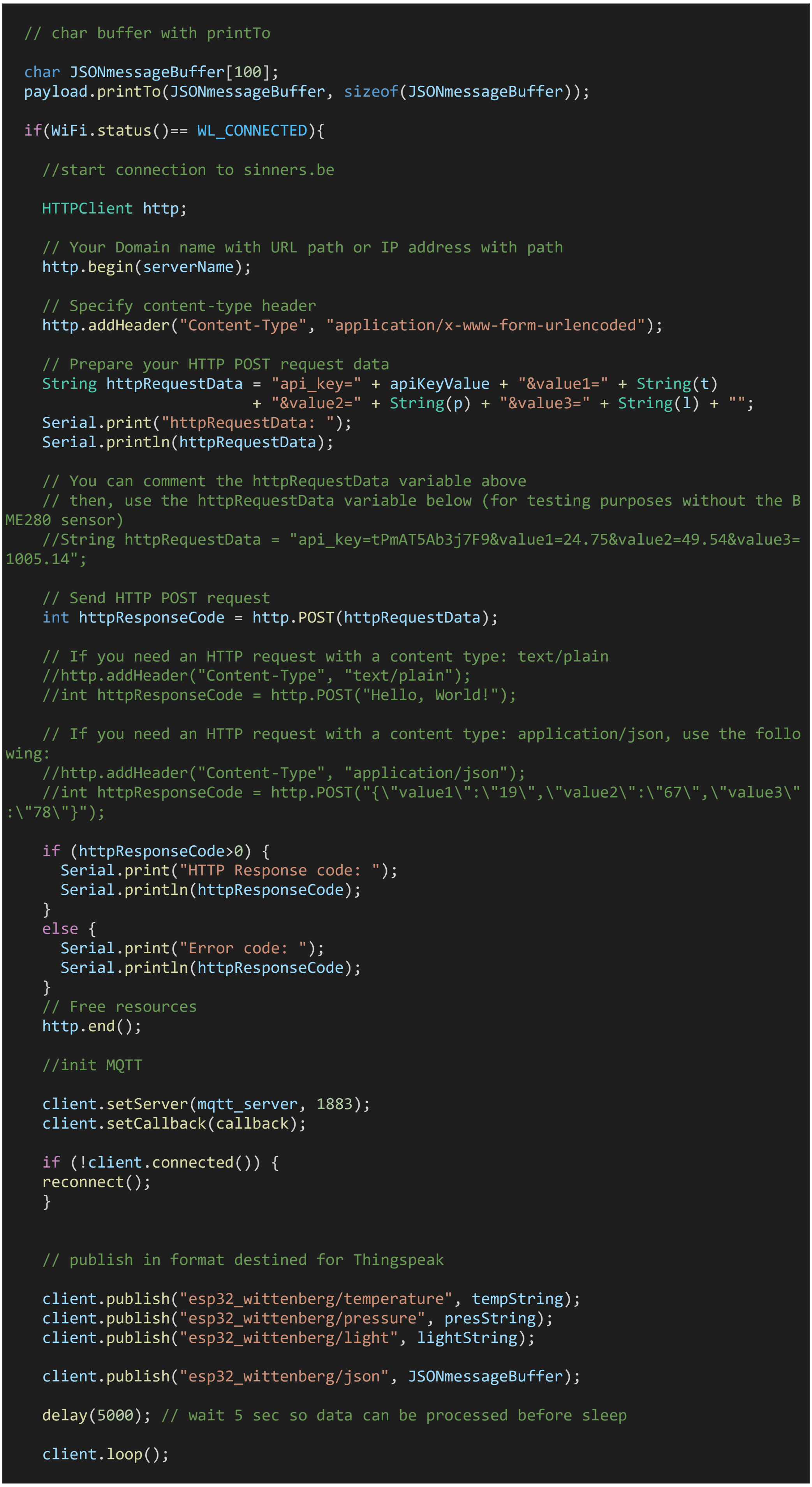
```
/*
  Joris Van Puyenbroeck

  Based on Rui Santos and many other examples
  Complete project details at https://RandomNerdTutorials.com

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files.

  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.

*/


#include <WiFi.h>
#include <HTTPClient.h>
#include <PubSubClient.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include <BH1750.h>
#include <ArduinoJson.h>
// #define LEDPIN 13
// LED Pin
const int ledPin = 13;

// Replace with your network credentials
const char* ssid     = "telenet-2CDDA_EXT";
const char* password = "xxxxxxx";

// const char* ssid     = "iPhone van joris";
// const char* password = "martje99";

WiFiClient espClient;

// MQTT data

const char* mqtt_server = "broker.emqx.io";
// const char* user = "xxx";
// const char* pwd = "xxxx";

PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;


// REPLACE with your Domain name and URL path or IP address with path
const char* serverName = "https://jorisvp.sinners.be/ITessentials/post_data.php";

// Keep this API Key value to be compatible with the PHP code provided in the project p
age.
// If you change the apiKeyValue value, the PHP file /post-
data.php also needs to have the same key
String apiKeyValue = "xxxxxxxx";

Adafruit_BMP280 bmp;  // I2C
BH1750 lightMeter;


// deep sleep

#define uS_TO_S_FACTOR 1000000  /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP  1800        /* Time ESP32 will go to sleep (in seconds) */
RTC_DATA_ATTR int bootCount = 0;

/*
Method to print the reason by which ESP32
has been awaken from sleep
```



```
*/
void print_wakeup_reason(){
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch(wakeup_reason)
  {
    case ESP_SLEEP_WAKEUP_EXT0 : Serial.println("Wakeup caused by external signal using
 RTC_IO"); break;
    case ESP_SLEEP_WAKEUP_EXT1 : Serial.println("Wakeup caused by external signal using
 RTC_CNTL"); break;
    case ESP_SLEEP_WAKEUP_TIMER : Serial.println("Wakeup caused by timer"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD : Serial.println("Wakeup caused by touchpad"); break
;
    case ESP_SLEEP_WAKEUP_ULP : Serial.println("Wakeup caused by ULP program"); break;
    default : Serial.printf("Wakeup was not caused by deep sleep: %d\n",wakeup_reason);
 break;
  }
}

// MQTT callback posibility

void callback(char* topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  Serial.println();

  // Feel free to add more if statements to control more GPIOs with MQTT

  // If a message is received on the topic esp32/output, you check if the message is ei
ther "on" or "off".
  // Changes the output state according to the message
  if (String(topic) == "esp32/output") {
    Serial.print("Changing output to ");
    if(messageTemp == "on"){
      Serial.println("on");
      digitalWrite(ledPin, HIGH);
    }
    else if(messageTemp == "off"){
      Serial.println("off");
      digitalWrite(ledPin, LOW);
    }
  }
}


void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("weatherstationvanjoris")) {
      Serial.println("connected");
      // Subscribe
      client.subscribe("esp32_wittenberg/output");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}


void setup() {
```



```
  // start serial and wifi hardware
  Serial.begin(115200);
  delay(1000); //Take some time to open up the Serial Monitor
  esp_err_t esp_wifi_start(void); //start wifi

  //Increment boot number and print it every reboot
  ++bootCount;
  Serial.println("Boot number: " + String(bootCount));

  //Print the wakeup reason for ESP32
  print_wakeup_reason();

  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());

  // start two sensors
  bool status = bmp.begin(0x77);
  if (!status) {
    Serial.println("Could not find a valid BMP280 sensor, check wiring or change I2C ad
dress!");
    while (1);
  }
  lightMeter.begin();

  //Check WiFi connection status

  // read sensors

  //read temperature
  float t = bmp.readTemperature();
  // Read pressure
  float p = bmp.readPressure()/100;
  // read light
  float l = lightMeter.readLightLevel(); // true tussen haakjes weggehaald

  if (isnan(t || isnan(p) || isnan(l))) {
  Serial.println("Failed to read from one or more sensors!");
  return;
  }

  // print the result to Terminal
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.println(" *C ");
  Serial.print("Pressure: ");
  Serial.print(p);
  Serial.println(" hPa");
  Serial.print("Light : ");
  Serial.print(l);
  Serial.println(" lx");

  // Convert the values to a char array
  char tempString[8];
  dtostrf(t, 1, 2, tempString);

  char presString[8];
  dtostrf(p, 1, 2, presString);

  char lightString[8];
  dtostrf(l, 1, 2, lightString);

  // convert to json object for message payload

  StaticJsonBuffer<300> JSONbuffer;
  JsonObject& payload = JSONbuffer.createObject();

  payload["Temperature"] = tempString;
  payload["Pressure"] = presString;
  payload["Light"] = lightString;
```



```
  // char buffer with printTo

  char JSONmessageBuffer[100];
  payload.printTo(JSONmessageBuffer, sizeof(JSONmessageBuffer));

  if(WiFi.status()== WL_CONNECTED){

    //start connection to sinners.be

    HTTPClient http;

    // Your Domain name with URL path or IP address with path
    http.begin(serverName);

    // Specify content-type header
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    // Prepare your HTTP POST request data
    String httpRequestData = "api_key=" + apiKeyValue + "&value1=" + String(t)
                           + "&value2=" + String(p) + "&value3=" + String(l) + "";
    Serial.print("httpRequestData: ");
    Serial.println(httpRequestData);

    // You can comment the httpRequestData variable above
    // then, use the httpRequestData variable below (for testing purposes without the B
ME280 sensor)
    //String httpRequestData = "api_key=tPmAT5Ab3j7F9&value1=24.75&value2=49.54&value3=
1005.14";

    // Send HTTP POST request
    int httpResponseCode = http.POST(httpRequestData);

    // If you need an HTTP request with a content type: text/plain
    //http.addHeader("Content-Type", "text/plain");
    //int httpResponseCode = http.POST("Hello, World!");

    // If you need an HTTP request with a content type: application/json, use the follo
wing:
    //http.addHeader("Content-Type", "application/json");
    //int httpResponseCode = http.POST("{\"value1\":\"19\",\"value2\":\"67\",\"value3\"
:\"78\"}");

    if (httpResponseCode>0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
    }
    else {
      Serial.print("Error code: ");
      Serial.println(httpResponseCode);
    }
    // Free resources
    http.end();

    //init MQTT

    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);

    if (!client.connected()) {
    reconnect();
    }


    // publish in format destined for Thingspeak

    client.publish("esp32_wittenberg/temperature", tempString);
    client.publish("esp32_wittenberg/pressure", presString);
    client.publish("esp32_wittenberg/light", lightString);

    client.publish("esp32_wittenberg/json", JSONmessageBuffer);

    delay(5000); // wait 5 sec so data can be processed before sleep

    client.loop();
  }
  else {
    Serial.println("WiFi Disconnected");
  }


// deep sleep

  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  esp_err_t esp_wifi_stop(void);
  Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) +
  " Seconds");
  Serial.println("Going to sleep now");
  delay(1000);
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("This will never be printed");

}

void loop() {

}
```