UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA
Physics, Computer Science and Mathematics Department

Master Degree in Computer Science, IOT Course

FreshAirIot Report

Student:

Lorenzo Stigliano
No. 185534

Academic Year 2023-2024

Contents

1 Requirements hardware and software

Here it is the list of components (hardware and software) that are needed to build FreshAirIot project:

- esp8266 board (NodeMcu V3 CH340) no. 4
- esp32 board no. 1 (Wemos D1 R32)
- bme280 sensors no. 2
- bme680 sensors no. 1
- rgb led no. 1
- buzzer no. 1
- light sensor no. 1
- Arduino IDE: for each bme sensors you need to install the right libraries:
  - bme280 library
  - bme680 library

  Is also important to install Universal Telegram Bot library and OpenWeather library. You also need to check for mqtt inside of Arduino libraries. More information about libraries will be given in **Software** chapter.

2 Introduction

The air pollution is becoming more and more dangerous for the public health. Is important to avoid breathing unhealthy air . FreshAirIot aims to check the air's quality of the area in which the user lives and suggest if is the case, or not, to open the window. The esp boards are scattered inside of the house and outside (front yard/back yard, windows) and they collect data about the environment. Once they have enough data they suggest the user what to do. There are 3 type of notifications that the final user receive:

- telegram notification
- specific rgb light that are visible on the board (green = open the window, yellow = you may consider to open the window, red = do not open the window)
- specific music theme that are played by the board that are related with color of the rgb light (zelda's theme = green, pacman's theme = yellow, doom's theme = red)

For what concerns the telegram notification there is a telegram bot that is always available and can provide the user some information about the current weather condition and forecasting up to 2 days.

The project is built using two types of boards:

- ESP 8266
- ESP 32

All the sensors are available online and I suggest the reader to use Aliexpress website, to get the cheaper price. As I said in the **Prerequisites** section, you need to install some additional libraries, but this will be more clear in the **Software** chapter.

3 How to Build it

3.1 Hardware

3.1.1 Prerequisites

In order to build the project I suggest to use a solder. Often the sensor's pins they don't come pre-soldered to the pcb, so you need to connect them by yuorself.

3.1.2 Board esp 8266 no. 1

The board number one has the following sensors attached:

- bme680
- light sensor

Here it is the pins configuration:

- light sensors (sensor's pin to board's pin):
  - S →A0
  - - (minus) →GND
  - pin in the middle (the remaining one) →3V

  S pin is for transmitting data to the board. Light sensor works as a resistance and output a value that represent the light condition. If the value is low then it means that the environment is full of light, vice versa if it is high. In the software part of this report is shown how it is implemented.

- bme 680
  - VCC →3V
  - GND →GND
  - SCL →D1
  - SDA →D2

  The **SCL** is the clock line and **SDA** is for data. This sensors can detect: humidity, altitude, pressure, temperature and gas. For what concerns gas, it works in a similar way to the light sensors. It behaves as resistance that is sensible to VOC (CO2), acetone, alcohol, ... if the data registered is high then the air is clean, vice versa is unhealthy.

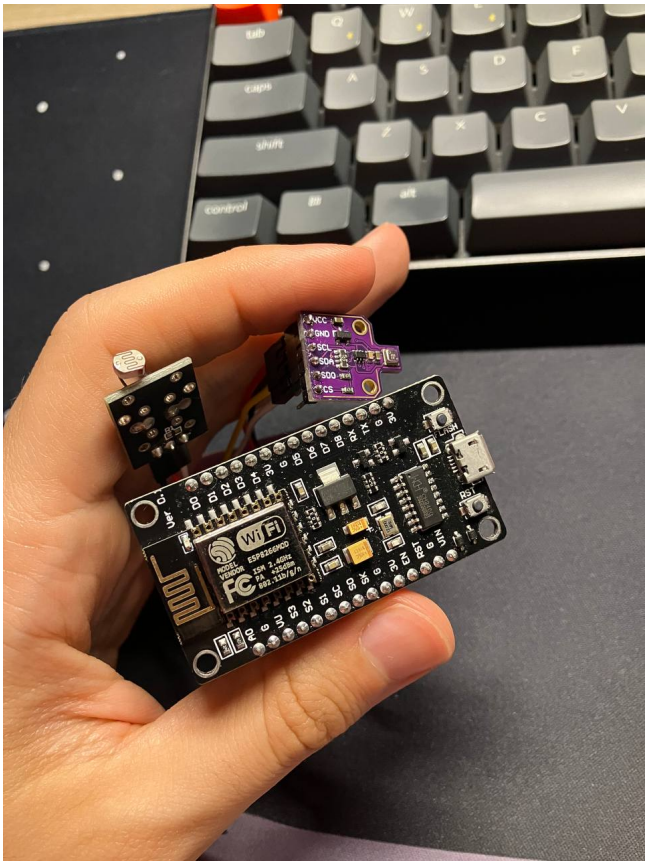

Figure 1: ESP8266 with bme680 and light sensor

3.1.3 Board esp 8266 no. 2

The board number two has just one sensor:

- bme 280
  - VCC →3V
  - GND →GND
  - SCL →D1
  - SDA →D2

  The **SCL** is the clock line and **SDA** is for data. This sensors can detect: altitude, pressure and temperature.

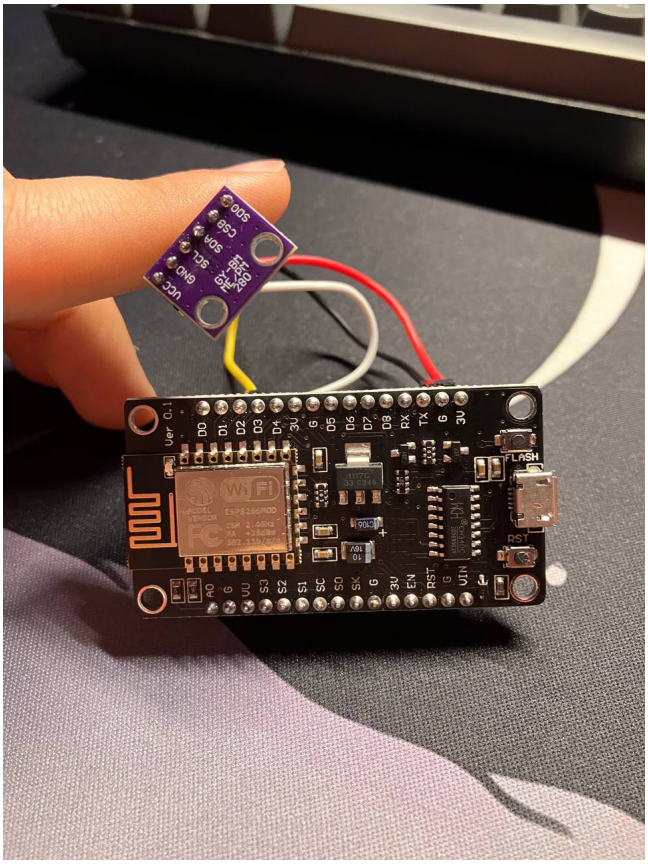

Figure 2: ESP 8266 with bme 280 sensor

3.1.4 Board esp 8266 no. 3

The board number two has just one sensor:

- bme 280
  - VCC →3V
  - GND →GND
  - SCL →D1
  - SDA →D2

  The **SCL** is the clock line and **SDA** is for data. This sensors can detect: altitude, pressure and temperature.

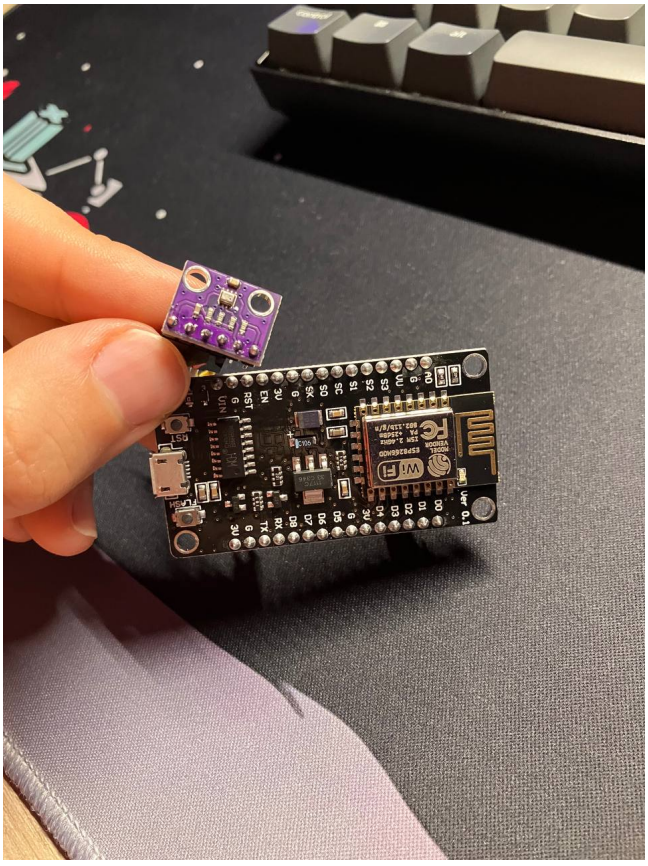

Figure 3: ESP 8266 with bme 280 sensor

3.1.5 Board esp 8266 no. 4

The board number four has not sensors attached, but it provides other services that will be explained in the software section.

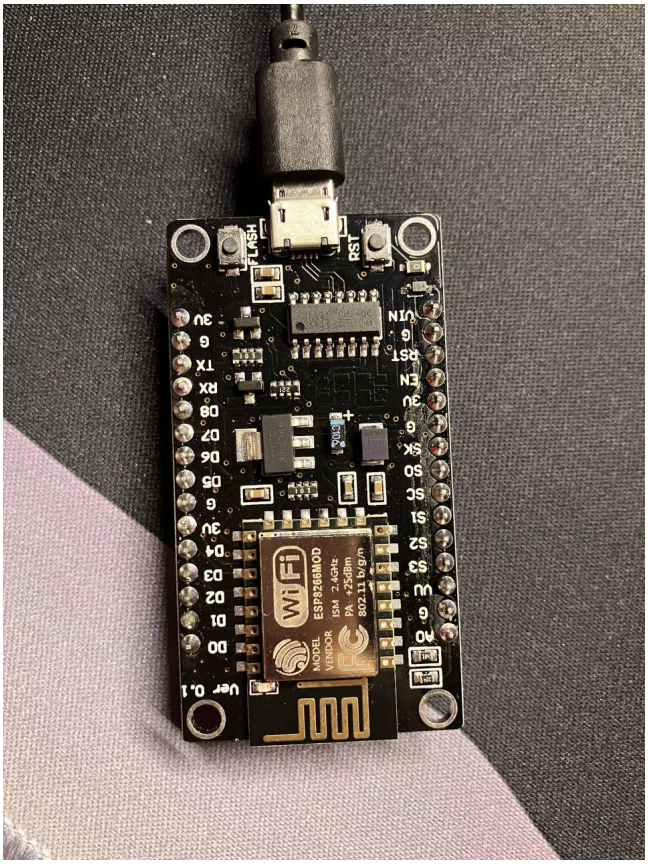

Figure 4: ESP 8266 Telegram Bot

3.1.6 Board esp 32 no. 1

The following board is used for actuators. It has two type of actuators:

- rgb led
  - R →IO19
  - G →IO23
  - B →IO18
  - - (minus) →GND
- buzzer
  - S →IO05
  - - (minus) →GND

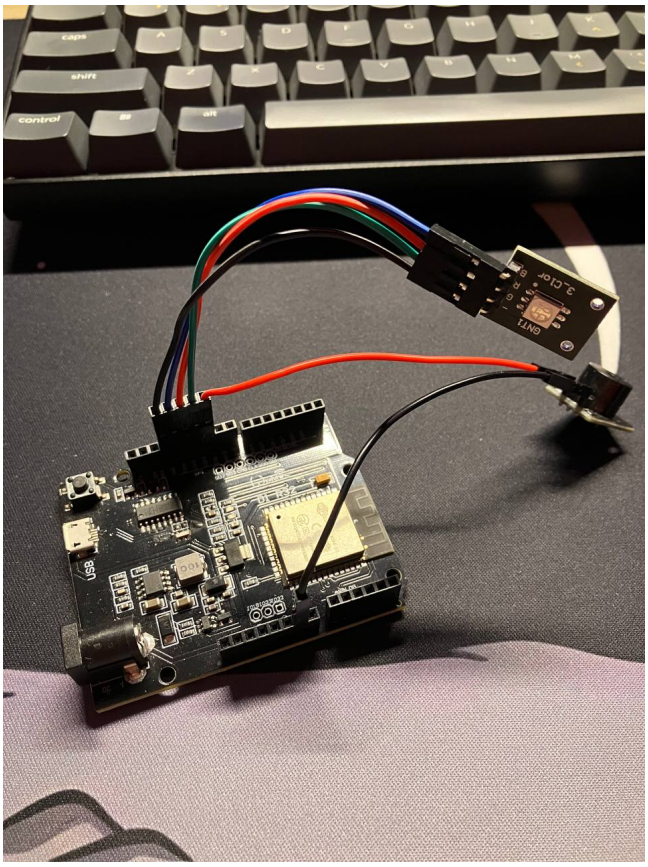

Figure 5: ESP32 with rbg light and buzzer actuators

3.2 Software

3.2.1 Prerequisites

The first thing it to have installed Arduino ide and install the followings libraries:

- ArduinoHttpClient
- Arduino_ESP32_OTA
- Adafruit BME280 Library
- Adafruit BME680 Library
- BSEC Software Library
- ArduinoJson
- ESP8266 Weather Station
- EspMQTTClient
- OpenWeather
- UniversalTelegramBot

All the libraries are available in Arduino ide: *Sketch →Inlcude Libraris →Manage Libaries*. From manage libraries option you can download and manage them directly from the ide. You also need to add the following boards manager URLs to have full accesso to all board libraries:

- *http://arduino.esp8266.com/stable/package_esp8266com_index.json*
- *https://dl.espressif.com/dl/package_esp32_index.json,https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json*

To do that you need to add those links in the text field inside of *file →preferences →Additional boards manager URLs*.

Delay and Deep Sleep

For testing purposes you will not find any deep sleep state, this is because I needed all boards to be reactive (maximum 10 seconds delay). In the final product I imagine at least 30 minutes of deep sleep time for each boards.

MQTT Configuration For what concerns MQTT I have used an online broker that offers its services for free. The broker is EQMX. In term of topic I have defined one topic for each type of data that the sensors can collect, also considering their position:

- home/hall/temperature
- home/hall/height
- home/hall/pressure
- home/room/temperature
- home/room/pressure
- home/room/height/
- home/room/window/temperature
- home/room/window/humidity
- home/room/window/pressure
- home/room/window/airquality
- home/room/window/height
- home/room/window/light
- home/actuators
- extern/
- weather/

Except the last two, all of them have a name that define clearly what is their purpose. The last one, `weather/`, is used to publish weather forecasting descriptions that are collected from esp 8266 board no. 2. (this board works also as a weather station). The esp 32 board then listens on `weather` topic and has a specific behavior if the next day is going to rain (it has a flashing blue light).

`extern/` is used by esp 8266 board no. 1 to publish the IAQ (Index of Air Quality). This data is used by esp 32 board and if there is the chance of raining then the rgb led will be blue, alternating the rgb color status related with *opening window* possibility. Actually, also `home/room/window/airquality` has information about air quality, but they are more fine grained, compare the the ones in `extern/`. This choice was done because in order to decide if is the case to open or not the window is enough to have that information, along with the data collected by the boards that are inside of the house.

Is possible to use a client toolbox, MQTTX, to visualize with a GUI, all the messages inside of each topics. This is really useful for debugging purposes, because is also possible to publish messages and check if the esp 32 board behaves correctly.

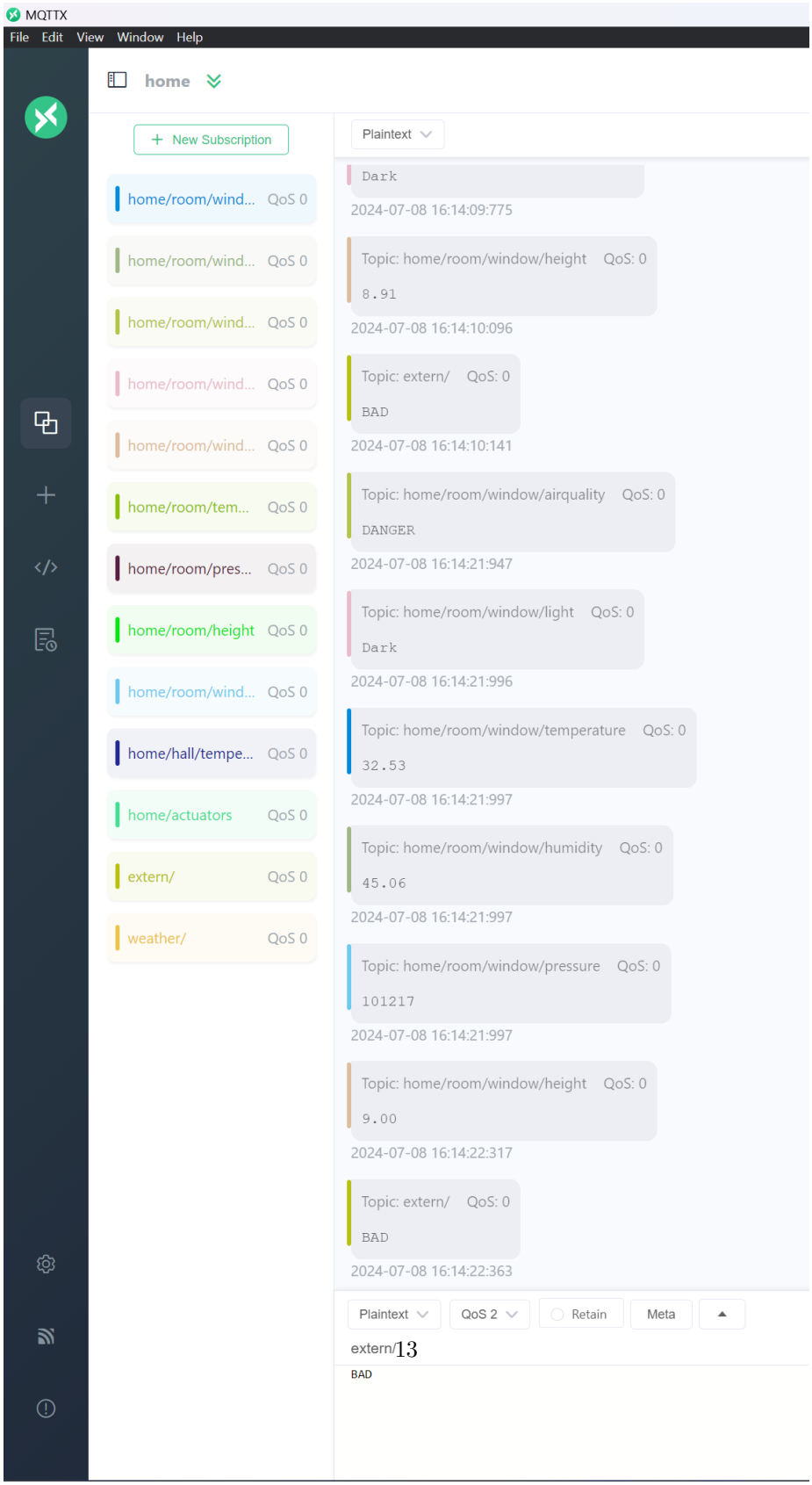

Figure 6: MQTTX Client

3.2.2 Board esp 8266 no. 1

This board is programmed for using bme680 and light sensors. It is placed typically outside of the house. In order to communicate all data that it senses to the other boards it leverages on mqtt protocol. It uses an online broker to deliver all the messages. This service is delivered my EMQX provider.

For what concerns the bme680 sensors there are some configuration values related with gas and humidity. Those values are used to calculate the quality of the air. In theory there is a pre-compiled library by Bosch, that gives us access to fine grain analysis of the data collected but the sensor. The only problem is that Bosch is not supporting the library actively and it is actually not compatible with the latest versions of Arduino IDE. On GitHub page are suggested some tricks that the developer can do in order to make it work, but they don't work anymore.

This is why I have implemented Bosch sensors with Adafruit library instead and there is a whole calculation to decide about the the air quality, leveraging on the basic sensing of gas that is available. This calculation is inspired by the consideration of another developer that has used the exact same sensors and has implemented the IAQ (Index of Air Quality) manually: github repo. The following image briefly summarize the following idea: as you can see, the ideal condition is to have 21 degree, 40% of humidity and 50k of gas resistance. The contribution of each of those elements is: humidity 10%, temperature 10% and gas 80%.

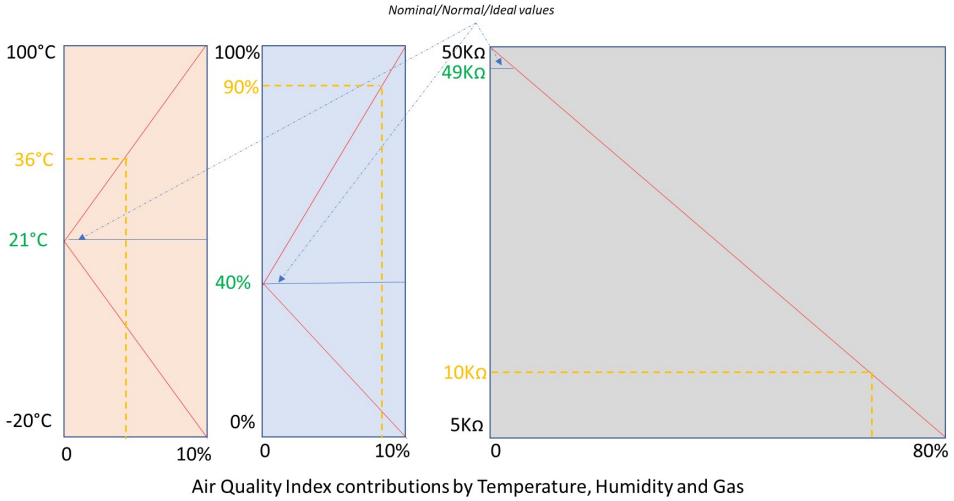

Figure 7: IAQ Scheme

Considering all that, this is the begging of the `.ino` file related with the first exp 8266 board:

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"
#include "WifiPassword.h"
#include "mqttCredentials.h"
#include <PubSubClient.h>
#include <ESP8266WiFi.h>
#include <stdio.h>

#define SEALEVELPRESSURE_HPA (1013.25)
#define MIN_VALUE 25
#define MAX_VALUE 800
#define REPORT_INTERVAL 1
#define MAX 100

Adafruit_BME680 bme; // I2C
//Adafruit_BME680 bme(BME_CS); // hardware SPI
//Adafruit_BME680 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK);

#define LIGHT_SENSOR_PIN A0 // The ESP8266 pin ADC0

float hum_weighting = 0.25; // so hum effect is 25% of the total air quality score
float gas_weighting = 0.75; // so gas effect is 75% of the total air quality score

float hum_score, gas_score;
float gas_reference = 250000;
float hum_reference = 40;
int   getgasreference_count = 0;

// MQTT Broker Configuration
const char *mqtt_broker = "broker.emqx.io";  // EMQX broker endpoint

// MQTT Topics
const char *mqtt_topic_temperature = "home/room/window/temperature";
const char *mqtt_topic_pressure = "home/room/window/pressure";
const char *mqtt_topic_humidity = "home/room/window/humidity";
const char *mqtt_topic_airquality = "home/room/window/airquality";
const char *mqtt_topic_light = "home/room/window/light";
const char *mqtt_topic_height = "home/room/window/height";
const char *mqtt_topic_extern = "extern/";

const char *mqtt_username = username;  // MQTT username for authentication
```

```
44 const char *mqtt_password = password_mqtt;  // MQTT password for authentication
45 const int mqtt_port = 1883;  // MQTT port (TCP)
46
47 WiFiClient espClient;
48 PubSubClient mqtt_client(espClient);
49
50 void connectToWiFi();
51
52 void connectToMQTTBroker();
53
54 void mqttCallback(char *topic, byte *payload, unsigned int length);
55
56 void setup() {
57   Serial.begin(9600);
58   while (!Serial);
59   Serial.println(F("BME680 async test"));
60
61   if (!bme.begin()) {
62     Serial.println(F("Could not find a valid BME680 sensor, check wiring!"));
63     while (1);
64   }
65
66   // Set up oversampling and filter initialization
67   bme.setTemperatureOversampling(BME680_OS_8X);
68   bme.setHumidityOversampling(BME680_OS_2X);
69   bme.setPressureOversampling(BME680_OS_4X);
70   bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
71   bme.setGasHeater(320, 150); // 320*C for 150 ms
72   GetGasReference();
73
74   // MQTT Configuration
75   connectToWiFi();
76   mqtt_client.setServer(mqtt_broker, mqtt_port);
77   mqtt_client.setCallback(mqttCallback);
78   connectToMQTTBroker();
79 }
80
81 void connectToWiFi() {
82     WiFi.begin(ssid, password);
83     Serial.print("Connecting to WiFi");
84     while (WiFi.status() != WL_CONNECTED) {
85         delay(500);
86         Serial.print(".");
87     }
88     Serial.println("\nConnected to the WiFi network");
89 }
```

```
90
91  void connectToMQTTBroker() {
92      while (!mqtt_client.connected()) {
93          String client_id = "esp8266-client-" + String(WiFi.macAddress());
94          Serial.printf("Connecting to MQTT Broker as %s.....\n", client_id.c_str());
95          if (mqtt_client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
96              Serial.println("Connected to MQTT broker");
97
98              mqtt_client.subscribe(mqtt_topic_temperature);
99              mqtt_client.subscribe(mqtt_topic_pressure);
100             mqtt_client.subscribe(mqtt_topic_humidity);
101             mqtt_client.subscribe(mqtt_topic_airquality);
102             mqtt_client.subscribe(mqtt_topic_light);
103             mqtt_client.subscribe(mqtt_topic_height);
104             mqtt_client.subscribe(mqtt_topic_extern);
105
106             // Publish message upon successful connection
107             // mqtt_client.publish(mqtt_topic, "Hi EMQX I'm ESP8266 Window Room ^^");
108         } else {
109             Serial.print("Failed to connect to MQTT broker, rc=");
110             Serial.print(mqtt_client.state());
111             Serial.println(" try again in 5 seconds");
112             delay(5000);
113         }
114     }
115 }
116
117 void mqttCallback(char *topic, byte *payload, unsigned int length) {
118     Serial.print("Message received on topic: ");
119     Serial.println(topic);
120     Serial.print("Message:");
121     for (unsigned int i = 0; i < length; i++) {
122         Serial.print((char) payload[i]);
123     }
124     Serial.println();
125     Serial.println("-----------------------");
126 }
```

As you can see in the beginning there is the configuration of three components for this board:

- WiFi
- MQTT
- BME 680 and light sensors

The WiFi and MQTT credentials are stored in two different files that need to be created by the user:

- WifiPassword.h
- mqttCredentials.h

In the second part of the file there is the loop in which data are collected and calculated by the sensors and finally published on different topics:

- home/room/window/temperature
- home/room/window/pressure
- home/room/window/humidity
- home/room/window/airquality
- home/room/window/light
- home/room/window/height
- extern/

Here it is the code related to the main loop, in which is also contained the IAQ algorithm. How it has been shown previously 7 this is based on three factors:

- humidity 10%
- temperature 10%
- bad gad level 80%

For each of them there is a score, that is based on their contribution to their final result and the fact that their actual value is in an optimal range. For humidity is between 38% and 41%, while for temperature is between 14 and 30 degrees Celsius. For gas the score is based upon the resistance of the sensor, so the higher is the value the better it is, in a range that is from 5000 to 50000 Ohms.

Combining all that scores we obtain a final result that represents the quality of the air. This is further adjusted in the `CalculateIAQ` method, but this is just for prettiness reason, it does no change the mathematical meaning of the number after the first calculation.

```
void loop() {
  if (!mqtt_client.connected()) {
    connectToMQTTBroker();
  }

  mqtt_client.loop();

  // Tell BME680 to begin measurement.
  unsigned long endTime = bme.beginReading();
  if (endTime == 0) {
    Serial.println(F("Failed to begin reading :("));
    return;
  }
  Serial.print(F("Reading started at "));
  Serial.print(millis());
  Serial.print(F(" and will finish at "));
  Serial.println(endTime);

  Serial.println(F("You can do other work during BME680 measurement."));
  delay(50); // This represents parallel work.
  // There's no need to delay() until millis() >= endTime: bme.endReading()
  // takes care of that. It's okay for parallel work to take longer than
  // BME680's measurement time.

  // Obtain measurement results from BME680. Note that this operation isn't
  // instantaneous even if milli() >= endTime due to I2C/SPI latency.
  if (!bme.endReading()) {
    Serial.println(F("Failed to complete reading :("));
    return;
  }
  Serial.print(F("Reading completed at "));
  Serial.println(millis());

  Serial.print(F("Temperature = "));
  Serial.print(bme.temperature);
  Serial.println(F(" *C"));

  Serial.print(F("Pressure = "));
  Serial.print(bme.pressure);
  Serial.println(F(" Pa"));

  Serial.print(F("Humidity = "));
  Serial.print(bme.humidity);
  Serial.println(F(" %"));

  Serial.print(F("Gas = "));
```

```
47    Serial.print(bme.gas_resistance);
48    Serial.println(F(" Ohms"));
49
50    Serial.print(F("Approx. Altitude = "));
51    Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
52    Serial.println(F(" m"));
53
54    Serial.println();
55
56    float current_humidity = bme.readHumidity();
57    if (current_humidity >= 38 && current_humidity <= 42)
58      hum_score = 0.10*100; // Humidity +/-5% around optimum
59    else
60    { //sub-optimal
61      if (current_humidity < 38)
62        hum_score = 0.10/hum_reference*current_humidity*100;
63      else
64      {
65        hum_score = ((-0.10/(100-hum_reference)*current_humidity)+0.10)*100;
66      }
67    }
68
69    float current_temperature = bme.readTemperature();
70    if (current_temperature >= 14 && current_temperature <= 30)
71      temp_score = 0.10*100; // Temperature +/-5% around optimum
72    else
73    { //sub-optimal
74      if (current_temperature < 14)
75        temp_score = 0.10/temp_reference*current_temperature*100;
76      else
77      {
78        temp_score = ((-0.10/(100-temp_reference)*current_temperature)+0.10)*100;
79      }
80    }
81
82    //Calculate gas contribution to IAQ index
83    float gas_lower_limit = 5000;   // Bad air quality limit
84    float gas_upper_limit = 50000;  // Good air quality limit
85    if (gas_reference > gas_upper_limit) gas_reference = gas_upper_limit;
86    if (gas_reference < gas_lower_limit) gas_reference = gas_lower_limit;
87    gas_score = (0.80/(gas_upper_limit-gas_lower_limit)*gas_reference -
88    (gas_lower_limit*(0.80/(gas_upper_limit-gas_lower_limit))))*100;
89
90    //Combine results for the final IAQ index value (0-100% where 100% is good quality air)
91
92    float air_quality_score = hum_score + gas_score + temp_score;
```

```
93
94      Serial.println("Air Quality = "+String(air_quality_score,1)+"% derived
95      from 10% of Humidity reading, 10% Temperature reading and 80%
96      of Gas reading - 100% is good quality air");
97      Serial.println("Humidity element was : "+String(hum_score/100)+" of 0.10");
98      Serial.println("Temperature element was : "+String(hum_score/100)+" of 0.10");
99      Serial.println("Gas element was : "+String(gas_score/100)+" of 0.80");
100     if (bme.readGas() < 120000) Serial.println("***** Poor air quality *****");
101     Serial.println();
102     if ((getgasreference_count++)%10==0) GetGasReference();
103     String aqs = CalculateIAQ(air_quality_score);
104     Serial.println(aqs);
105     Serial.println("----------------------------------------------");
106
107
108     int sensorValue = analogRead(LIGHT_SENSOR_PIN);
109     float level = 100 - ((sensorValue - MIN_VALUE) * 100 / (MAX_VALUE
110     - MIN_VALUE)); //normalised value
111
112     sensorValue = level;
113     Serial.println("LUX = ");
114     Serial.println(sensorValue);
115
116
117     // Light Data
118     String light_text = "";
119     if (sensorValue < 40) {
120       Serial.println(" => Dark");
121       light_text += "Dark";
122
123     } else if (sensorValue < 60) {
124       Serial.println(" => Dim");
125       light_text += "Dim";
126
127     } else if (sensorValue < 70) {
128       Serial.println(" => Light");
129       light_text += "Light";
130
131     } else if (sensorValue < 80) {
132       Serial.println(" => Bright");
133       light_text += "Bright";
134
135     } else {
136       Serial.println(" => Very bright");
137       light_text += "Very bright";
138
```

```
139     }
140
141     // External Condition Calculation
142     String extern_message = "BAD";
143
144     if (strcmp (aqs.c_str(), "Good") == 0 || strcmp (aqs.c_str(), "Moderate") == 0){
145       if (bme.pressure > 100000) {
146           if (bme.temperature > 12 && bme.temperature < 29){
147             if  (bme.humidity < 60){
148               extern_message = "GOOD";
149             } else {
150               extern_message = "MAYBE";
151             }
152           } else if (bme.temperature < 30 && bme.humidity < 60){
153             extern_message = "MAYBE";
154           }
155       } else {
156         extern_message = "BAD";
157       }
158     } else {
159       extern_message = "BAD";
160     }
161
162     // MQTT Publish Messages
163     mqtt_client.publish(mqtt_topic_airquality, aqs.c_str());
164     mqtt_client.publish(mqtt_topic_light, light_text.c_str());
165     mqtt_client.publish(mqtt_topic_pressure, String(bme.pressure).c_str());
166     mqtt_client.publish(mqtt_topic_humidity, String(bme.humidity).c_str());
167     mqtt_client.publish(mqtt_topic_temperature, String(bme.temperature).c_str());
168     mqtt_client.publish(mqtt_topic_height,
169     String(bme.readAltitude(SEALEVELPRESSURE_HPA)).c_str());
170     mqtt_client.publish(mqtt_topic_extern, extern_message.c_str());
171
172
173     delay(10000);
174
175     }
176
177     void GetGasReference(){
178     // Now run the sensor for a burn-in period, then use combination of relative
179     // humidity and gas resistance to estimate indoor air quality as a percentage.
180     Serial.println("Getting a new gas reference value");
181     int readings = 10;
182     for (int i = 1; i <= readings; i++){ // read gas for 10 x 0.150mS = 1.5secs
183       gas_reference += bme.readGas();
184     }
```

```
185   gas_reference = gas_reference / readings;
186 }
187
188 String CalculateIAQ(float score){
189   String IAQ_text = "";
190   score = (100-score)*5;
191   if      (score >= 301)                  IAQ_text += "Hazardous";
192   else if (score >= 201 && score <= 300 ) IAQ_text += "Very Unhealthy";
193   else if (score >= 176 && score <= 200 ) IAQ_text += "Unhealthy";
194   else if (score >= 151 && score <= 175 ) IAQ_text += "Unhealthy for Sensitive Groups";
195   else if (score >=  51 && score <= 150 ) IAQ_text += "Moderate";
196   else if (score >=  00 && score <=  50 ) IAQ_text += "Good";
197   else IAQ_text += "DANGER";
198   return IAQ_text;
199 }
200
```

The final result of the IAQ algorithm could be either GOOD, MAYBE or BAD. This result in then published on `extern/` and the esp 32 board (that has all the actuators) listens on that topic and behave on different ways. The value published on `extern/` is not the only one that defines the esp 32 board's behavior. That board also listens on other topics that tell the state of the house. Combining the data related with the house and with the external environment, is then able to suggest the user properly about the idea of opening the windows.

3.2.3 Board esp 8266 no. 2

The second esp 8266 board is designed for collecting data about the temperature of the living room and it gathers data about the weather, leveraging Open Weather API. The initial part of the code is similar to the esp 8266 no. 1 board, but in this case there is the configuration of the bme 280 sensor and Open Weather/Telegram API. As for WiFi and MQTT protocol, also for those other API there are configuration files that needed to be done:

- openweather_api_key.h
- telegrambotCredentials.h

MQTT topics are slightly different, because the data are related with the living room environment and not the external one.

```
#include <Arduino.h>
#if defined(ESP8266)
#include <ESP8266WiFi.h>
#else
#include <WiFi.h>
#endif
#include <JsonListener.h>
#include <time.h>
#include "OpenWeatherMapForecast.h"
#include <PubSubClient.h>
#include "mqttCredentials.h"
#include "WifiPassword.h"
#include "openweather_api_key.h"
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include "telegrambotCredentials.h"

// MQTT Broker and topics Configuration
const char *mqtt_broker = "broker.emqx.io";  // EMQX broker endpoint
const char *mqtt_topic_temperature = "home/hall/temperature";
const char *mqtt_topic_pressure = "home/hall/pressure";
const char *mqtt_topic_height = "home/hall/height";
const char *mqtt_topic_weather = "weather/";

const char *mqtt_username = username;  // MQTT username for authentication
const char *mqtt_password = password_mqtt;  // MQTT password for authentication
const int mqtt_port = 1883;  // MQTT port (TCP)
const char *bottoken = BotToken;
const char *chatidbot = ChatID;
```

```
// BMP280 Sensor Settings
#define BMP_SCK 13
#define BMP_MISO 12
#define BMP_MOSI 11
#define BMP_CS 10

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BMP280 bme;

// initiate the client
OpenWeatherMapForecast client;
String OPEN_WEATHER_MAP_APP_ID = open_weather_api_key;
String OPEN_WEATHER_MAP_LOCATION_ID = "3181903"; // Modena = 3173331, Bomporto = 3181903
String OPEN_WEATHER_MAP_LANGUAGE = "en";
boolean IS_METRIC = false;
uint8_t MAX_FORECASTS = 3;
String rain_notification = "not done";

/**
 * WiFi Settings
 */
#if defined(ESP8266)
const char* ESP_HOST_NAME = "esp-" + ESP.getFlashChipId();
#else
const char* ESP_HOST_NAME = "esp-" + ESP.getEfuseMac();
#endif
const char* WIFI_SSID     = ssid;
const char* WIFI_PASSWORD = password;

// initiate the WifiClient
WiFiClient wifiClient;
WiFiClientSecure wifiClientSecure;
PubSubClient mqtt_client(wifiClient);

// X509List cert(TELEGRAM_CERTIFICATE_ROOT);
UniversalTelegramBot bot(bottoken, wifiClientSecure);

/**
 * SETUP
 */
void setup() {
  // Connecting to wifi
  Serial.begin(115200);
  delay(500);
  connectWifi();
```

```
79
80    //configTime(0, 0, "pool.ntp.org");      // get UTC time via NTP
81    //wifiClient.setTrustAnchors(&cert); // Add root certificate for api.telegram.org
82
83    // Starting mqtt
84    mqtt_client.setServer(mqtt_broker, mqtt_port);
85    mqtt_client.setCallback(mqttCallback);
86    connectToMQTTBroker();
87
88    // Checking if the sensor is connected properly
89    if (!bme.begin(0x76)) {
90      Serial.println("Could not find a valid BMP280 sensor, check wiring!");
91      while (1);
92    }
93
94    wifiClientSecure.setInsecure();
95
96    Serial.println();
97  }
98
99
100 /**
101  * Helping funtions
102  */
103 void connectWifi() {
104   Serial.begin(9600);
105   WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
106   Serial.print("Connecting to ");
107   Serial.println(WIFI_SSID);
108   while (WiFi.status() != WL_CONNECTED) {
109     delay(500);
110     Serial.print(".");
111   }
112   Serial.println("");
113   Serial.println("WiFi connected!");
114   Serial.println(WiFi.localIP());
115   Serial.println();
116 }
117
118
119 void connectToMQTTBroker() {
120     while (!mqtt_client.connected()) {
121         String client_id = "esp8266-client-weather-station " + String(WiFi.macAddress());
122         Serial.printf("Connecting to MQTT Broker as %s.....\n", client_id.c_str());
123         if (mqtt_client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
124             Serial.println("Connected to MQTT broker");
```

```
125             mqtt_client.subscribe(mqtt_topic_weather);
126         } else {
127             Serial.print("Failed to connect to MQTT broker, rc=");
128             Serial.print(mqtt_client.state());
129             Serial.println(" try again in 5 seconds");
130             delay(5000);
131         }
132     }
133 }
134 
135 void mqttCallback(char *topic, byte *payload, unsigned int length) {
136     Serial.print("Message received on topic: ");
137     Serial.println(topic);
138     Serial.print("Message:");
139     for (unsigned int i = 0; i < length; i++) {
140         Serial.print((char) payload[i]);
141     }
142     Serial.println();
143     Serial.println("-----------------------");
144 }
145 
146 /**
147  * LOOP
148  */
149 void loop() {
150   if (!mqtt_client.connected()) {
151     connectToMQTTBroker();
152   }
153 
154   mqtt_client.loop();
155 
156   Serial.println("\n\nNext Loop-Step: " + String(millis()) + ":");
157 
158   OpenWeatherMapForecastData data[MAX_FORECASTS];
159   client.setMetric(IS_METRIC);
160   client.setLanguage(OPEN_WEATHER_MAP_LANGUAGE);
161   uint8_t allowedHours[] = {0,12};
162   client.setAllowedHours(allowedHours, 2);
163   uint8_t foundForecasts = client.updateForecastsById(data,
164   OPEN_WEATHER_MAP_APP_ID, OPEN_WEATHER_MAP_LOCATION_ID, MAX_FORECASTS);
165   Serial.printf("Found %d forecasts in this call\n", foundForecasts);
166   Serial.println("---------------------------------");
167   time_t time;
168   const char* description_forecast[foundForecasts];
169   for (uint8_t i = 0; i < foundForecasts; i++) {
170     Serial.printf("---\nForecast number: %d\n", i);
```

```
171       // {"dt":1527066000, uint32_t observationTime;
172       time = data[i].observationTime;
173       Serial.printf("observationTime: %d, full date: %s",
174       data[i].observationTime, ctime(&time));
175       // "main":{
176       //   "temp":17.35, float temp;
177       Serial.printf("temp: %f\n", data[i].temp);
178       //   "feels_like": 16.99, float feelsLike;
179       Serial.printf("feels-like temp: %f\n", data[i].feelsLike);
180       //   "temp_min":16.89, float tempMin;
181       Serial.printf("tempMin: %f\n", data[i].tempMin);
182       //   "temp_max":17.35, float tempMax;
183       Serial.printf("tempMax: %f\n", data[i].tempMax);
184       //   "pressure":970.8, float pressure;
185       Serial.printf("pressure: %f\n", data[i].pressure);
186       //   "sea_level":1030.62, float pressureSeaLevel;
187       Serial.printf("pressureSeaLevel: %f\n", data[i].pressureSeaLevel);
188       //   "grnd_level":970.8, float pressureGroundLevel;
189       Serial.printf("pressureGroundLevel: %f\n", data[i].pressureGroundLevel);
190       //   "humidity":97, uint8_t humidity;
191       Serial.printf("humidity: %d\n", data[i].humidity);
192       //   "temp_kf":0.46
193       // },"weather":[{
194       //   "id":802, uint16_t weatherId;
195       Serial.printf("weatherId: %d\n", data[i].weatherId);
196       //   "main":"Clouds", String main;
197       Serial.printf("main: %s\n", data[i].main.c_str());
198       //   "description":"scattered clouds", String description;
199       Serial.printf("description: %s\n", data[i].description.c_str());
200       description_forecast[i] = data[i].description.c_str();
201
202       //   "icon":"03d" String icon; String iconMeteoCon;
203       Serial.printf("icon: %s\n", data[i].icon.c_str());
204       Serial.printf("iconMeteoCon: %s\n", data[i].iconMeteoCon.c_str());
205       // }],"clouds":{"all":44}, uint8_t clouds;
206       Serial.printf("clouds: %d\n", data[i].clouds);
207       // "wind":{
208       //   "speed":1.77, float windSpeed;
209       Serial.printf("windSpeed: %f\n", data[i].windSpeed);
210       //   "deg":207.501 float windDeg;
211       Serial.printf("windDeg: %f\n", data[i].windDeg);
212       // rain: {3h: 0.055}, float rain;
213       Serial.printf("rain: %f\n", data[i].rain);
214       // },"sys":{"pod":"d"}
215       // dt_txt: "2018-05-23 09:00:00"   String observationTimeText;
216       Serial.printf("observationTimeText: %s\n",
```

```
217       data[i].observationTimeText.c_str());
218     }
219
220     mqtt_client.publish(mqtt_topic_weather, description_forecast[0]);
221     //Serial.printf("\nSono fuori da ifelse: %s",rain_notification);
222
223     if (strstr(description_forecast[0], "rain") != NULL
224     && strcmp(rain_notification.c_str(), "not done") == 0) {
225         rain_notification = "done";
226         bot.sendMessage(chatidbot, "Hey I just wanted to tell you that tomorrow
227         is going to rain, so the air will be better! :3","Markdown");
228         //Serial.printf("\nSono dentro ad if: %s",rain_notification);
229     } else if (strstr(description_forecast[0], "rain") == NULL) {
230       rain_notification = "not done";
231       //Serial.printf("\nSono dentro ad else: %s",rain_notification);
232     }
233
234     Serial.println();
235     Serial.println("-----------------------------------------------------/\n");
236
237
238     Serial.print("Hall Temperature = ");
239     Serial.print(bme.readTemperature());
240     Serial.println(" *C");
241
242     Serial.print("Hall Pressure = ");
243     Serial.print(bme.readPressure());
244     Serial.println(" Pa");
245
246     Serial.print("Hall Approx altitude = ");
247     Serial.print(bme.readAltitude(1013.25));
248     // this should be adjusted to your local forcase
249     Serial.println(" m");
250
251     Serial.println();
252     Serial.println("-----------------------------------------------------/\n");
253
254     Serial.println();
255     mqtt_client.publish(mqtt_topic_pressure, String(bme.readPressure()).c_str());
256     mqtt_client.publish(mqtt_topic_temperature,
257     String(bme.readTemperature()).c_str());
258     mqtt_client.publish(mqtt_topic_height,
259     String(bme.readAltitude(SEALEVELPRESSURE_HPA)).c_str());
260
261     delay(10000);
262   }
```

```
263 }
```

The data that are collected from Open Weather API also include 2 days forecasting. This information is used for creating notification about rain conditions. This is important because when is raining the air is more clean, so it could be convenient to open the window after that!

The data collected about forecasting are published on a specific MQTT topic that is call `weather`. The actuators board listen on that topic and when there is the chance of rain it notifies the user. This board has a *physical* notification to the user (blue light flashing on rgb led), but there is also a Telegram notification.

The telegram bot API is not only used for notify the user whenever is going to rain, but the can also works as a weather station for the current weather and forecasting the following two days! This is technically done in esp 8266 board no. 4, the no. 2 use Telegram Bot API just to notify the user if tomorrow is going to rain. For now the data collected are just referred to Bomporto, small town close to Modena, but I am still working on a system to change dynamically the location. The only problem is that: each location is mapped with a digit and to get that number you need to check on Open Weather website. There is no method that can do this directly into the code, so I need to create a small scraper that can retrieve this number automatically whenever the user writes the city that he/she is looking for in the following format: *city*, *state* (e.g. Modena, IT).

3.2.4 Board esp 8266 no. 3

The esp 8266 board number 3 is configured similarly to the board esp 8266 no. 2, but there is not Open Weather/Telegram Bot API. This board is just used to sense data, in particular the temperature, inside of one of the bedroom.

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include "WifiPassword.h"
#include "mqttCredentials.h"
#include <PubSubClient.h>
#include <ESP8266WiFi.h>

#define BMP_SCK 13
#define BMP_MISO 12
#define BMP_MOSI 11
#define BMP_CS 10

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BMP280 bme; // I2C
//Adafruit_BMP280 bme(BMP_CS); // hardware SPI
//Adafruit_BMP280 bme(BMP_CS, BMP_MOSI, BMP_MISO, BMP_SCK);

// MQTT Broker Configuration
const char *mqtt_broker = "broker.emqx.io";  // EMQX broker endpoint
const char *mqtt_topic_temperature = "home/room/temperature";
const char *mqtt_topic_pressure = "home/room/pressure";
const char *mqtt_topic_height = "home/room/height";

const char *mqtt_username = username;  // MQTT username for authentication
const char *mqtt_password = password_mqtt;  // MQTT password for authentication
const int mqtt_port = 1883;  // MQTT port (TCP)

WiFiClient espClient;
PubSubClient mqtt_client(espClient);

void connectToWiFi();

void connectToMQTTBroker();

void mqttCallback(char *topic, byte *payload, unsigned int length);


void setup() {
```

```
  Serial.begin(9600);
  Serial.println(F("BMP280 test"));

  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BMP280 sensor, check wiring!");
    while (1);
  }

  connectToWiFi();
  mqtt_client.setServer(mqtt_broker, mqtt_port);
  mqtt_client.setCallback(mqttCallback);
  connectToMQTTBroker();

}

void connectToWiFi() {
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConnected to the WiFi network");
}

void connectToMQTTBroker() {
    while (!mqtt_client.connected()) {
        String client_id = "esp8266-client-" + String(WiFi.macAddress());
        Serial.printf("Connecting to MQTT Broker as %s.....\n", client_id.c_str());
        if (mqtt_client.connect(client_id.c_str(), mqtt_username, mqtt_password)) {
            Serial.println("Connected to MQTT broker");

            mqtt_client.subscribe(mqtt_topic_temperature);
            mqtt_client.subscribe(mqtt_topic_pressure);
            mqtt_client.subscribe(mqtt_topic_height);

            // Publish message upon successful connection
            // mqtt_client.publish(mqtt_topic, "Hi EMQX I'm ESP8266 ^^");
        } else {
            Serial.print("Failed to connect to MQTT broker, rc=");
            Serial.print(mqtt_client.state());
            Serial.println(" try again in 5 seconds");
            delay(5000);
        }
    }
}
```

```
88
89  void mqttCallback(char *topic, byte *payload, unsigned int length) {
90      Serial.print("Message received on topic: ");
91      Serial.println(topic);
92      Serial.print("Message:");
93      for (unsigned int i = 0; i < length; i++) {
94          Serial.print((char) payload[i]);
95      }
96      Serial.println();
97      Serial.println("-----------------------");
98  }
99
100 void loop() {
101   if (!mqtt_client.connected()) {
102     connectToMQTTBroker();
103   }
104
105   mqtt_client.loop();
106   // mqtt_client.publish(mqtt_topic, "Hi EMQX I'm ESP8266 from Lorenzo's Room");
107
108   Serial.println();
109   Serial.println("-----------------------------------------------------/\n");
110
111   Serial.print("Room Temperature = ");
112   Serial.print(bme.readTemperature());
113   Serial.println(" *C");
114
115   Serial.print("Room Pressure = ");
116   Serial.print(bme.readPressure());
117   Serial.println(" Pa");
118
119   Serial.print("Room Approx altitude = ");
120   Serial.print(bme.readAltitude(1013.25));
121   // this should be adjusted to your local forcase
122   Serial.println(" m");
123
124   Serial.println();
125   Serial.println("-----------------------------------------------------/\n");
126
127   Serial.println();
128
129   mqtt_client.publish(mqtt_topic_pressure, String(bme.readPressure()).c_str());
130   mqtt_client.publish(mqtt_topic_temperature,
131   String(bme.readTemperature()).c_str());
132   mqtt_client.publish(mqtt_topic_height,
133   String(bme.readAltitude(SEALEVELPRESSURE_HPA)).c_str());
```

```
134
135     delay(10000);
136   }
```

3.2.5 Board esp 8266 no. 4

The final esp 8266 board is designed for running 24/7 Telegram Bot. This implementation is not sending any notification of rain, but it waits for user commands:

- todayweather: tells the user today's weather (actual temperature, minimum temperature, maximum temperature, humidity, description)
- forecastweather: tells the user following two day's weather (actual temperature, minimum temperature, maximum temperature, humidity, description)
- commands: list of the available commands
- status: tells the bot's status
- help: general information

The setup is similar to the esp 8266 no. 2, but there is a loop for the bot, because it has to be always online in order to respond quickly to the user.

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include <JsonListener.h>
#include "OpenWeatherMapForecast.h"
#include "OpenWeatherMapCurrent.h"
#include "WifiPassword.h"
#include "openweather_api_key.h"
#include "telegrambotCredentials.h"

// Wifi network station credentials
#define WIFI_SSID ssid
#define WIFI_PASSWORD password
// Telegram BOT Token (Get from Botfather)
#define BOT_TOKEN BotToken

OpenWeatherMapCurrent client;
OpenWeatherMapForecast client_forecast;

String OPEN_WEATHER_MAP_APP_ID = open_weather_api_key;
String OPEN_WEATHER_MAP_LOCATION_ID = "3181903"; // Modena = 3173331, Bomporto = 3181903
String OPEN_WEATHER_MAP_LANGUAGE = "en";
boolean IS_METRIC = true;
uint8_t MAX_FORECASTS = 2;

const unsigned long BOT_MTBS = 1000; // mean time between scan messages

```

```
unsigned long bot_lasttime; // last time messages' scan has been done
X509List cert(TELEGRAM_CERTIFICATE_ROOT);
WiFiClientSecure secured_client;
UniversalTelegramBot bot(BOT_TOKEN, secured_client);

void handleNewMessages(int numNewMessages)
{
  Serial.print("handleNewMessages ");
  Serial.println(numNewMessages);

  String answer;
  for (int i = 0; i < numNewMessages; i++)
  {
    telegramMessage &msg = bot.messages[i];
    Serial.println("Received " + msg.text);

    if (msg.text == "/help")
      answer = "So you need _help_, uh? me too! use /commands or /status";

    else if (msg.text == "/commands")
      answer = "Welcome my new friend! You are the first *" + msg.from_name + "* I've ever m

    else if (msg.text == "/status")
      answer = "All is good here, thanks for asking!";

    else if (msg.text == "/todayweather"){
      OpenWeatherMapCurrentData data;
      client.setLanguage(OPEN_WEATHER_MAP_LANGUAGE);
      client.setMetric(IS_METRIC);
      client.updateCurrentById(&data,
      OPEN_WEATHER_MAP_APP_ID, OPEN_WEATHER_MAP_LOCATION_ID);

      String msg_today = "*Forecast Number* " + String(i) + "\n" + "Temperature:
      " + String(data.temp) +"\n"
      + "Minimum Temperature: " + String(data.tempMin) + "\n" + "Maximum
      Temperature: " + String(data.tempMax) +
      "\n" + "Humidity: " + String(data.humidity) + "\n" + "Description: " +
      String(data.description) + "\n";

      bot.sendMessage(msg.chat_id, msg_today, "Markdown");

    } else if (msg.text == "/forecastweather"){

      OpenWeatherMapForecastData data[MAX_FORECASTS];
      client_forecast.setMetric(IS_METRIC);
      client_forecast.setLanguage(OPEN_WEATHER_MAP_LANGUAGE);
```

```
74        uint8_t allowedHours[] = {0,12};
75        client_forecast.setAllowedHours(allowedHours, 2);
76        uint8_t foundForecasts = client_forecast.updateForecastsById(data,
77        OPEN_WEATHER_MAP_APP_ID, OPEN_WEATHER_MAP_LOCATION_ID, MAX_FORECASTS);
78        const char* description_forecast[foundForecasts];
79
80        for (uint8_t i = 0; i < foundForecasts; i++) {
81            String msg_forecast = "*Forecast Number* " + String(i) + "\n" +
82            "Temperature: " + String(data[i].temp) +"\n"
83            + "Minimum Temperature: " + String(data[i].tempMin) + "\n" + "Maximum
84            Temperature: " + String(data[i].tempMax) +
85            "\n" + "Humidity: " + String(data[i].humidity) + "\n" + "Description: "
86            + String(data[i].description) + "\n";
87            bot.sendMessage(msg.chat_id, msg_forecast, "Markdown");
88        }
89      } else
90        answer = "Say what?";
91
92      bot.sendMessage(msg.chat_id, answer, "Markdown");
93    }
94  }
95
96  void bot_setup()
97  {
98    const String commands = F("["
99                              "{\"command\":\"todayweather\",\"description\":
100                             \"Shows Today's Weather\"},"
101                             "{\"command\":\"forecastweather\",\"description\":
102                             \"Shows Forecast Weather up to 15 days\"},"
103                             "{\"command\":\"commands\", \"description\":\"Message
104                             sent when you open a chat with a bot\"},"
105                             "{\"command\":\"status\",\"description\":\"Answer
106                             device current status\"},"
107                             "{\"command\":\"help\",  \"description\":\"Get bot
108                             usage help\"}" // no comma on last command
109                             "]");
110
111   bot.setMyCommands(commands);
112   //bot.sendMessage("25235518", "Hola amigo!", "Markdown");
113 }
114
115 void setup()
116 {
117   Serial.begin(9600);
118   Serial.println();
119
```

```
  // attempt to connect to Wifi network:
  configTime(0, 0, "pool.ntp.org");      // get UTC time via NTP
  secured_client.setTrustAnchors(&cert); // Add root certificate for api.telegram.org
  Serial.print("Connecting to Wifi SSID ");
  Serial.print(WIFI_SSID);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(500);
  }
  Serial.print("\nWiFi connected. IP address: ");
  Serial.println(WiFi.localIP());

  // Check NTP/Time, usually it is instantaneous and you can delete the code below.
  Serial.print("Retrieving time: ");
  time_t now = time(nullptr);
  while (now < 24 * 3600)
  {
    Serial.print(".");
    delay(100);
    now = time(nullptr);
  }
  Serial.println(now);

  bot_setup();
}

void loop()
{
  if (millis() - bot_lasttime > BOT_MTBS)
  {
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1);

    while (numNewMessages)
    {
      Serial.println("got response");
      handleNewMessages(numNewMessages);
      numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    }

    bot_lasttime = millis();
  }
}
```

3.2.6 Board esp 32 no. 1

This board is programmed to listen on the following MQTT topic and based on the data that are published it can suggest the user if it is the case or not to open the windows. This are the topic that it is subscribed to:

- home/room/temperature
- home/hall/temperature
- home/actuators
- extern/
- weather/

Thanks to `extern` the board knows about the external IAQ and with the data of the housem it can decide properly about opening or not the windows. The logic behind that is represented in the following scheme:

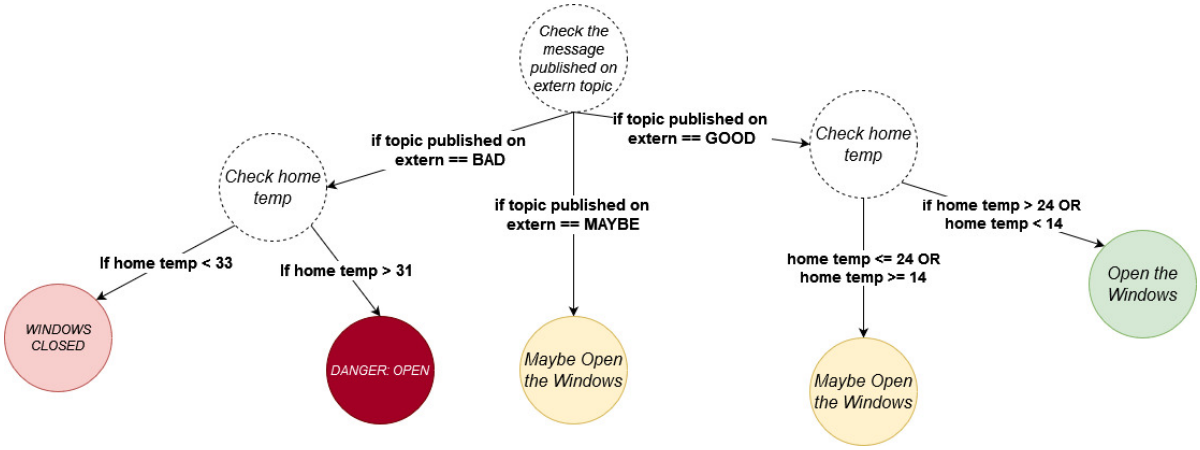

Figure 8: ESP 32 Actuators Scheme

So the board can have three basic states:

- open the window (green light, zelda theme song)
- Maybe open the window (yellow light, pacman theme song)
- do not open the window (red light, doom theme song)

If there is an intermittent blue light it means that it knows, thanks to the data published in `weather/` topic, that is going to rain the next day. There is also a corner case: if the temperature inside of the building is greater than 33 degree Celsius and the IAQ is BAD, there is a white light and a danger alarm. Each song is provided by Arduino song projects. The project provide all the information to play each song properly: tempo, notes and pitches. Every first time that the status of the actuators change, a specific song is played.

```
#include "pitches.h"
#include "WifiPassword.h"
#include "mqttCredentials.h"
#include <PubSubClient.h>
#include <WiFi.h>

#define BUZZER_PIN  5

#define PIN_RED    19 // GPIO23
#define PIN_GREEN  23 // GPIO22
#define PIN_BLUE   18 // GPIO21

int melody_danger[] = {
  NOTE_AS4,4,  NOTE_F4,-4, NOTE_AS4,4,  NOTE_F4,-4,
  NOTE_AS4,4,  NOTE_F4,-4, NOTE_AS4,4,  NOTE_F4,-4,
};

int melody_good[] = {

  NOTE_AS4,4,  NOTE_F4,-4,  NOTE_AS4,8,  NOTE_AS4,16,  NOTE_C5,16,
  NOTE_D5,16, NOTE_DS5,16,//7
  NOTE_F5,2,  NOTE_F5,8,  NOTE_F5,8,  NOTE_F5,8,  NOTE_FS5,16,
  NOTE_GS5,16,
  NOTE_AS5,-2,  NOTE_AS5,8,  NOTE_AS5,8,  NOTE_GS5,8,  NOTE_FS5,16,
  NOTE_GS5,-8,  NOTE_FS5,16,  NOTE_F5,2,  NOTE_F5,4,

  NOTE_DS5,-8, NOTE_F5,16, NOTE_FS5,2, NOTE_F5,8, NOTE_DS5,8, //11
  NOTE_CS5,-8, NOTE_DS5,16, NOTE_F5,2, NOTE_DS5,8, NOTE_CS5,8,
  NOTE_C5,-8, NOTE_D5,16, NOTE_E5,2, NOTE_G5,8,
  NOTE_F5,16, NOTE_F4,16, NOTE_F4,16,
  NOTE_F4,16,NOTE_F4,16,NOTE_F4,16,NOTE_F4,16,NOTE_F4,16,NOTE_F4,8,
  NOTE_F4,16,NOTE_F4,8,

};

int melody_maybe[] = {

  // Pacman
  // Score available at https://musescore.com/user/85429/scores/107109
  NOTE_B4, 16, NOTE_B5, 16, NOTE_FS5, 16, NOTE_DS5, 16, //1
  NOTE_B5, 32, NOTE_FS5, -16, NOTE_DS5, 8, NOTE_C5, 16,
  NOTE_C6, 16, NOTE_G6, 16, NOTE_E6, 16, NOTE_C6, 32, NOTE_G6, -16,
  NOTE_E6, 8,

  NOTE_B4, 16,  NOTE_B5, 16,  NOTE_FS5, 16,   NOTE_DS5, 16,  NOTE_B5,
  32,  //2
```

```
  NOTE_FS5, -16, NOTE_DS5, 8,  NOTE_DS5, 32, NOTE_E5, 32,  NOTE_F5,
  32,
  NOTE_F5, 32,  NOTE_FS5, 32,  NOTE_G5, 32,  NOTE_G5, 32, NOTE_GS5,
  32,  NOTE_A5, 16, NOTE_B5, 8

};

const int melody_bad[] PROGMEM = {

  // At Doom's Gate (E1M1)
  // Score available at https://musescore.com/pieridot/doom

  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8,
  NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8, //1
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, 8, NOTE_E2, 8,
  NOTE_E2, 8, NOTE_B2, 8, NOTE_C3, 8,
  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8,
  NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8,
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, -2,

  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8,
  NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8, //13
  NOTE_C3, 8, NOTE_E2, 8, NOTE_E2, 8, NOTE_AS2, 8, NOTE_E2, 8,
  NOTE_E2, 8, NOTE_B2, 8, NOTE_C3, 8,
  NOTE_E2, 8, NOTE_E2, 8, NOTE_E3, 8, NOTE_E2, 8, NOTE_E2, 8,
  NOTE_D3, 8, NOTE_E2, 8, NOTE_E2, 8,
  NOTE_FS3, -16, NOTE_D3, -16, NOTE_B2, -16, NOTE_A3, -16, NOTE_FS3,
  -16, NOTE_B2, -16, NOTE_D3, -16, NOTE_FS3, -16, NOTE_A3, -16,
  NOTE_FS3, -16, NOTE_D3, -16, NOTE_B2, -16,
};

const char *mqtt_broker = "broker.emqx.io";  // EMQX broker endpoint

const char *mqtt_topic_temperature_room = "home/room/temperature";
const char *mqtt_topic_temperature_hall = "home/hall/temperature";
const char *mqtt_topic_actuators = "home/actuators";
const char *mqtt_topic_extern = "extern/";
const char *mqtt_topic_weather = "weather/";


const char *mqtt_username = username;  // MQTT username for
// authentication
const char *mqtt_password = password_mqtt;  // MQTT password for
/ authentication
const int mqtt_port = 1883;  // MQTT port (TCP)

```

```
String mqttpayload_extern = "";
String mqttpayload_room_temp = "";
String mqttpayload_hall_temp = "";
String mqttpayload_weather = "";

WiFiClient espClient;
PubSubClient mqtt_client(espClient);

bool good = 0;
bool maybe = 0;
bool bad = 0;

void connectToWiFi();

void connectToMQTTBroker();

void mqttCallback(char *topic, byte *payload, unsigned int length);

void setup() {
  Serial.begin(9600);

  pinMode(PIN_RED,   OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);
  pinMode(PIN_BLUE,  OUTPUT);

  connectToWiFi();
  mqtt_client.setServer(mqtt_broker, mqtt_port);
  mqtt_client.setCallback(mqttCallback);
  connectToMQTTBroker();
}

void connectToWiFi() {
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nConnected to the WiFi network");
}

void connectToMQTTBroker() {
    while (!mqtt_client.connected()) {
        String client_id = "esp32 actuators board " +
        String(WiFi.macAddress());
        Serial.printf("Connecting to MQTT Broker as %s.....\n",
```

```
139           client_id.c_str());
140           if (mqtt_client.connect(client_id.c_str(), mqtt_username,
141           mqtt_password)) {
142               Serial.println("Connected to MQTT broker");
143               mqtt_client.subscribe(mqtt_topic_temperature_room);
144               mqtt_client.subscribe(mqtt_topic_temperature_hall);
145               mqtt_client.subscribe(mqtt_topic_actuators);
146               mqtt_client.subscribe(mqtt_topic_extern);
147               mqtt_client.subscribe(mqtt_topic_weather);
148
149
150               // Publish message upon successful connection
151               // mqtt_client.publish(mqtt_topic, "Hi EMQX I'm ESP32
152               Actuators ^^");
153           } else {
154               Serial.print("Failed to connect to MQTT broker, rc=");
155               Serial.print(mqtt_client.state());
156               Serial.println(" try again in 5 seconds");
157               delay(5000);
158           }
159       }
160   }
161
162   void mqttCallback(char *topic, byte *payload, unsigned int length) {
163       // mqttpayload_room_temp [mqttpayloadSize] = {'\8'};
164       // mqttpayload_hall_temp [mqttpayloadSize] = {'\8'};
165       // mqttpayload_extern [mqttpayloadSize] = {'\0'};
166       // mqttpayload_weather [mqttpayloadSize] = {'\0'};
167
168       // Debuging Messages
169       Serial.print("Message received on topic: ");
170       Serial.println(topic);
171       Serial.print("Message:");
172
173       if (strcmp(topic, "extern/") == 0){
174         mqttpayload_extern = "";
175         //Serial.println("Sono dentro extern\n");
176         for (unsigned int i = 0; i < length; i++) {
177           Serial.print((char) payload[i]);
178           mqttpayload_extern.concat((char) payload[i]);
179         }
180       }
181
182       if (strcmp(topic, "home/room/temperature")== 0){
183         mqttpayload_room_temp = "";
184         //Serial.println("Sono dentro room temp\n");
```

```
185         for (unsigned int i = 0; i < length; i++) {
186           Serial.print((char) payload[i]);
187           mqttpayload_room_temp.concat((char) payload[i]);
188 
189         }
190       }
191       Serial.println();
192 
193       if (strcmp(topic, "home/hall/temperature")== 0){
194         mqttpayload_hall_temp = "";
195         //Serial.println("Sono dentro hall temp\n");
196         for (unsigned int i = 0; i < length; i++) {
197           Serial.print((char) payload[i]);
198           mqttpayload_hall_temp.concat((char) payload[i]);
199         }
200       }
201     Serial.println();
202 
203       if (strcmp(topic, "weather/")== 0){
204         mqttpayload_weather = "";
205         //Serial.println("Sono dentro weather\n");
206         for (unsigned int i = 0; i < length; i++) {
207           Serial.print((char) payload[i]);
208           mqttpayload_weather.concat((char) payload[i]);
209         }
210       }
211 
212       Serial.println();
213 
214       if (strcmp(topic, "home/actuators")== 0){
215         //Serial.println("Sono dentro home/actuators\n");
216         for (unsigned int i = 0; i < length; i++) {
217           Serial.print((char) payload[i]);
218         }
219       }
220 
221       Serial.println();
222       Serial.println("-----------------------");
223   }
224 
225   // Song Parameters
226   int notes = 0;
227   int tempo = 0;
228   int wholenote = 0;
229   int divider = 0, noteDuration = 0;
230 
```

```
231 void loop() {
232   if (!mqtt_client.connected()) {
233     connectToMQTTBroker();
234   }
235 
236   mqtt_client.loop();
237 
238   float home_temp = (mqttpayload_room_temp.toFloat() +
239   mqttpayload_hall_temp.toFloat())/2;
240 
241   // Debugging Print
242   // Serial.println("Final values recorded (extern, room temp, hall
243   tempo, weather forecast 1 day): \n");
244   // Serial.println(mqttpayload_extern);
245   // Serial.println();
246   // Serial.println(mqttpayload_room_temp);
247   // Serial.println();
248   // Serial.println(mqttpayload_hall_temp);
249   // Serial.println();
250   // Serial.println(mqttpayload_weather);
251   // Serial.println();
252   // Serial.println("------------------------");
253 
254   if (strstr(mqttpayload_weather.c_str(), "rain")){
255       //Serial.println("Tomorrow is goin to rain: ");
256       for (int i = 0; i < 3; i++){
257         setColor(0,0,255);
258         delay(500);
259         setColor(255,255,255);
260        }
261   }
262 
263 if (strstr(mqttpayload_extern.c_str(), "GOOD") != NULL) {
264       if (home_temp > 24 || home_temp < 14){
265 
266 
267       if (good == 0){
268         good = 1;
269         maybe = 0;
270         bad = 0;
271 
272         setColor(0,255,0);
273         mqtt_client.publish(mqtt_topic_actuators, "WINDOWS OPEN");
274 
275         tempo = 88;
276         notes = sizeof(melody_good) / sizeof(melody_good[0]) / 2;
```

```
277           wholenote = (60000 * 4) / tempo;
278           divider = 0, noteDuration = 0;
279           for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {
280
281               // calculates the duration of each note
282               divider = melody_good[thisNote + 1];
283               if (divider > 0) {
284                 // regular note, just proceed
285                 noteDuration = (wholenote) / divider;
286               } else if (divider < 0) {
287                 // dotted notes are represented with negative durations!!
288                 noteDuration = (wholenote) / abs(divider);
289                 noteDuration *= 1.5; // increases the duration in half for dotted notes
290               }
291
292               // we only play the note for 90% of the duration, leaving 10% as a pause
293               tone(BUZZER_PIN, melody_good[thisNote], noteDuration*0.9);
294
295               // Wait for the specief duration before playing the next note.
296               delay(noteDuration);
297
298               // stop the waveform generation before the next note.
299               noTone(BUZZER_PIN);
300             }
301         }
302       } else {
303
304           if (maybe == 0){
305             maybe = 1;
306             good = 0;
307             bad = 0;
308             setColor(255,255,0);
309             mqtt_client.publish(mqtt_topic_actuators, "MAYBE WINDOWS OPEN");
310             tempo = 105;
311             notes = sizeof(melody_maybe) / sizeof(melody_maybe[0]) / 2;
312             wholenote = (60000 * 4) / tempo;
313             divider = 0, noteDuration = 0;
314             for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {
315
316                 // calculates the duration of each note
317                 divider = melody_maybe[thisNote + 1];
318                 if (divider > 0) {
319                   // regular note, just proceed
320                   noteDuration = (wholenote) / divider;
321                 } else if (divider < 0) {
322                   // dotted notes are represented with negative durations!!
```

```
                noteDuration = (wholenote) / abs(divider);
                noteDuration *= 1.5; // increases the duration in half for dotted notes
              }

              // we only play the note for 90% of the duration, leaving 10% as a pause
              tone(BUZZER_PIN, melody_maybe[thisNote], noteDuration * 0.9);

              // Wait for the specief duration before playing the next note.
              delay(noteDuration);

              // stop the waveform generation before the next note.
              noTone(BUZZER_PIN);
            }
        }
    }
  } else if (strstr(mqttpayload_extern.c_str(), "MAYBE") != NULL) {


        if (maybe == 0){
          maybe = 1;
          good = 0;
          bad = 0;
          setColor(255,255,0);
          mqtt_client.publish(mqtt_topic_actuators, "MAYBE WINDOWS OPEN");
          tempo = 105;
          notes = sizeof(melody_maybe) / sizeof(melody_maybe[0]) / 2;
          wholenote = (60000 * 4) / tempo;
          divider = 0, noteDuration = 0;
          for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

              // calculates the duration of each note
              divider = melody_maybe[thisNote + 1];
              if (divider > 0) {
                // regular note, just proceed
                noteDuration = (wholenote) / divider;
              } else if (divider < 0) {
                // dotted notes are represented with negative durations!!
                noteDuration = (wholenote) / abs(divider);
                noteDuration *= 1.5; // increases the duration in half for dotted notes
              }

              // we only play the note for 90% of the duration, leaving 10% as a pause
              tone(BUZZER_PIN, melody_maybe[thisNote], noteDuration * 0.9);

              // Wait for the specief duration before playing the next note.
              delay(noteDuration);
```

```
369
370                 // stop the waveform generation before the next note.
371                 noTone(BUZZER_PIN);
372               }
373           }
374     } else if (strstr(mqttpayload_extern.c_str(), "BAD") != NULL ) {
375         if (home_temp < 33){
376
377
378           if (bad == 0) {
379             setColor(255,0,0);
380             mqtt_client.publish(mqtt_topic_actuators, "WINDOWS CLOSED");
381             good = 0;
382             maybe = 0;
383             bad = 1;
384
385             tempo = 225;
386             notes = sizeof(melody_bad) / sizeof(melody_bad[0]) / 2;
387             wholenote = (60000 * 4) / tempo;
388             divider = 0, noteDuration = 0;
389
390             for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {
391
392                 // calculates the duration of each note
393                 divider = pgm_read_word_near(melody_bad+thisNote + 1);
394                 if (divider > 0) {
395                   // regular note, just proceed
396                   noteDuration = (wholenote) / divider;
397                 } else if (divider < 0) {
398                   // dotted notes are represented with negative durations!!
399                   noteDuration = (wholenote) / abs(divider);
400                   noteDuration *= 1.5; // increases the duration in half for dotted notes
401                 }
402
403                 // we only play the note for 90% of the duration, leaving 10% as a pause
404                 tone(BUZZER_PIN, pgm_read_word_near(melody_bad+thisNote), noteDuration * 0.9);
405
406                 // Wait for the specief duration before playing the next note.
407                 delay(noteDuration);
408
409                 // stop the waveform generation before the next note.
410                 noTone(BUZZER_PIN);
411             }
412           }
413       } else {
414             setColor(255,255,255);
```

```
415           mqtt_client.publish(mqtt_topic_actuators, "DANGER");
416
417           tempo = 88;
418           notes = sizeof(melody_danger) / sizeof(melody_danger[0]) / 2;
419           wholenote = (60000 * 4) / tempo;
420           divider = 0, noteDuration = 0;
421           for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {
422
423             // calculates the duration of each note
424             divider = melody_danger[thisNote + 1];
425             if (divider > 0) {
426               // regular note, just proceed
427               noteDuration = (wholenote) / divider;
428             } else if (divider < 0) {
429               // dotted notes are represented with negative durations!!
430               noteDuration = (wholenote) / abs(divider);
431               noteDuration *= 1.5; // increases the duration in half for dotted notes
432             }
433
434             // we only play the note for 90% of the duration, leaving 10% as a pause
435             tone(BUZZER_PIN, melody_danger[thisNote], noteDuration*0.9);
436
437             // Wait for the specief duration before playing the next note.
438             delay(noteDuration);
439
440             // stop the waveform generation before the next note.
441             noTone(BUZZER_PIN);
442           }
443       }
444     } else {
445       setColor(0,0,0);
446     }
447     delay(100);
448   }
449
450   void setColor(int R, int G, int B) {
451     analogWrite(PIN_RED,   R);
452     analogWrite(PIN_GREEN, G);
453     analogWrite(PIN_BLUE,  B);
454   }
```

3.2.7 NodeRed Configuration

I have configured NodeRed along with dashboard plugin to visualize all the data that are published in each different topics. I gather all the information in four views:

- window
- room
- hall
- home

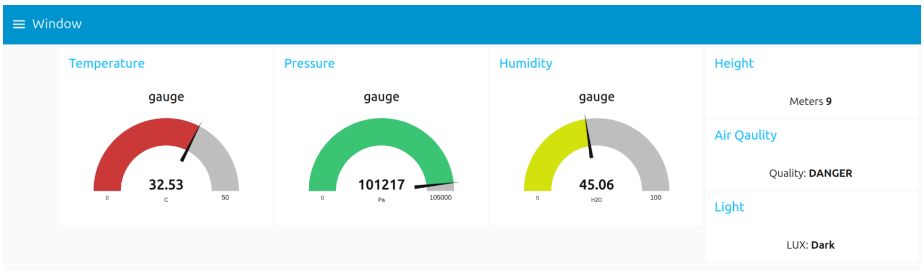

Figure 9: NodeRed window view

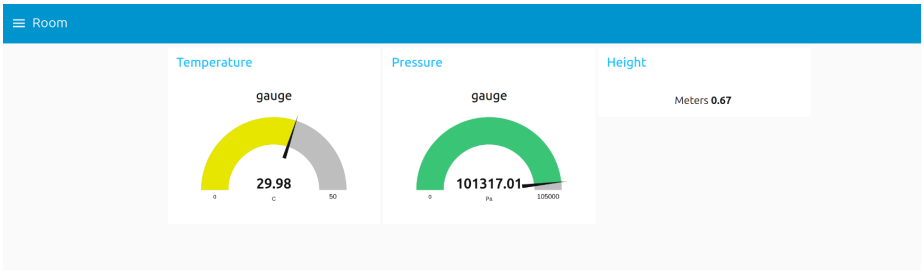

Figure 10: NodeRed room view

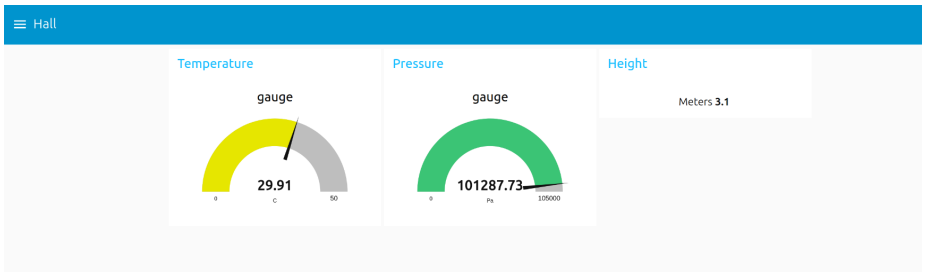

Figure 11: NodeRed hall view

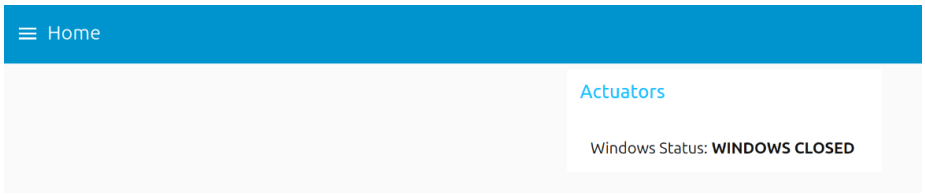

Figure 12: NodeRed actuators view

Thanks to that the user can see in real time what is going on inside and outside of the house. I have installed NodeRed into a ThinkPad T14, but is possible to install it also on small board, like Raspberry Pi/Orange Pi.

4 Testing

If everything is soldered and set up correctly when you connect the board the first time they immediately start to collect data and publish them into MQTT topics. After a minute or so, the actuators will start to show some specific color and it will start play melodies. It plays those song just once, every time its state changes. So, for example, with a nice weather and good IAQ, is likely that the actuators board will suggest the user to open the windows (green light + zelda's song). But if after a couple of hours the temperature will raise, along with the humidity and the IAQ will get worse it will suggest the user to close the window (red light + doom's song). If it is going to rain the next day the user will receive a message on telegram and the light status will blink with blue color.

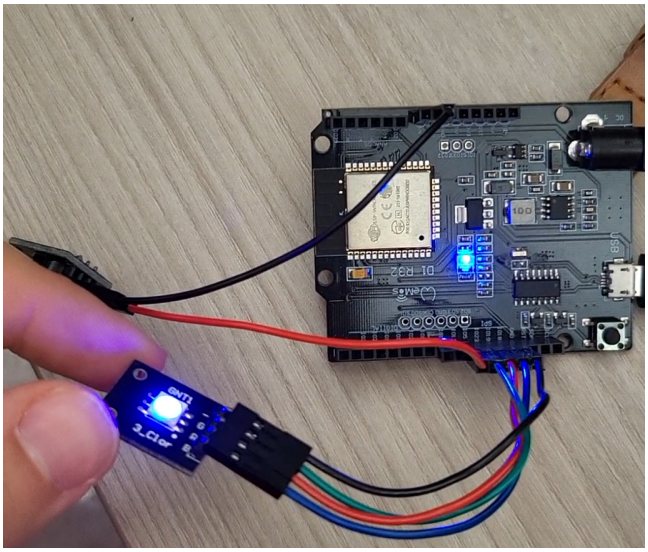

Figure 13: ESP32 Actuators suggesting to open the windows tomorrow

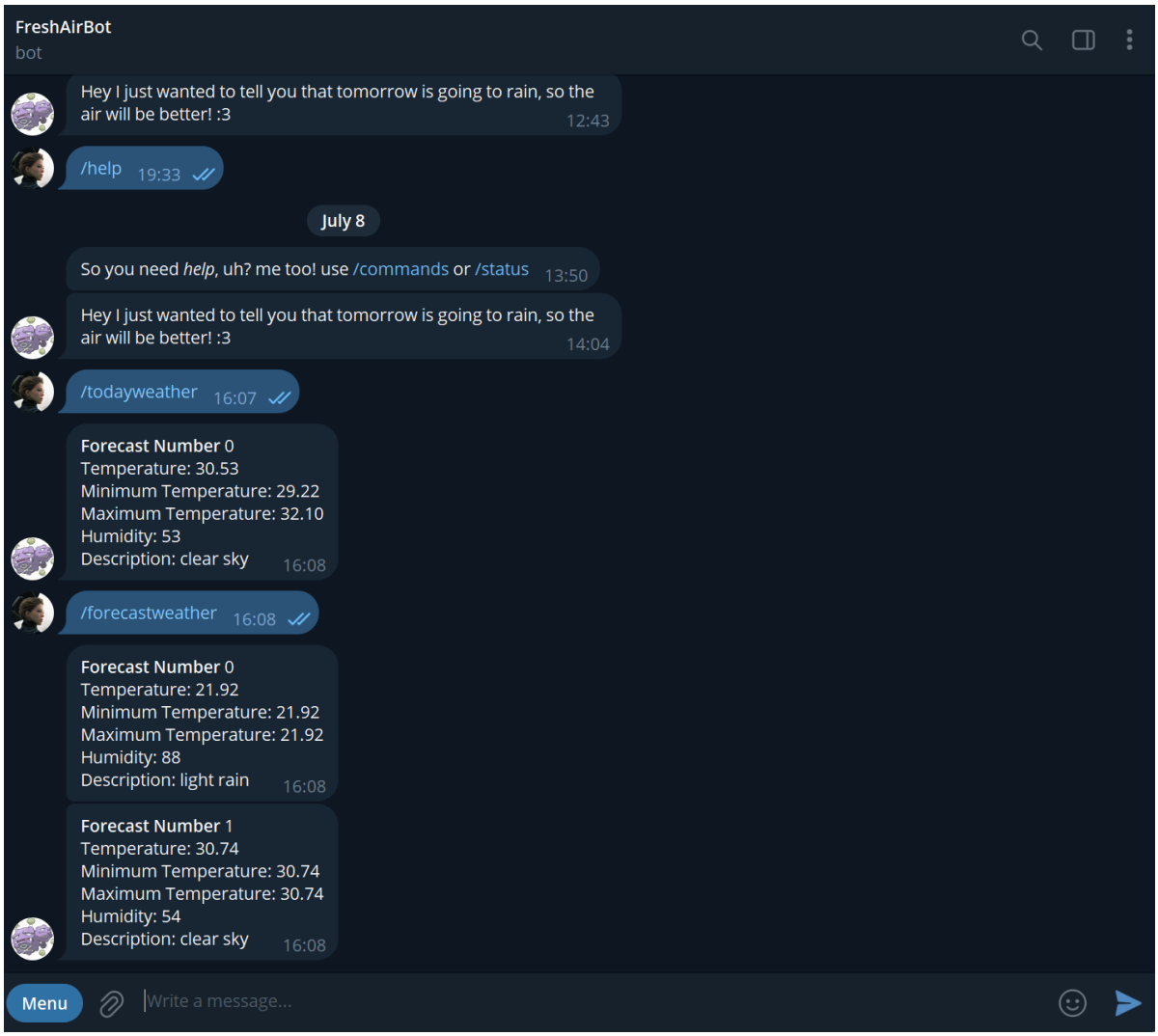

Figure 14: Interaction with Telegram Bot

5 Future Developments

There are several problems that need to be addressed. The sensors cannot be placed far away from an internet access point, so it could be interesting to do could implement LORA communication protocol.

Another aspect is related with the bot. As it has been said in the report, there should be an option that can modify the location of the weather forecasting directly in the telegram bot.

It would be interesting to try to implement a local MQTT broker with another esp board and doing the same also for NodeRed. No online broke and no computers, just boards.

For the forecasting air quality (that is related with the weather) it would be interesting if there is any API or developing one from scratch that can do some predictions, based on the vehicles traffic and the factories activity around my area. I think it should be possible to gather some data to consider also the human activities as an important factor for the calculus of IAQ in real time.