|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PANS w KROŚNIE** | | | | |
| **INFORMATYKA:** SSI | | | | **III ROK** |
| Imię i nazwisko | Michał Miksiewicz  Michał Pasieka | Tryb studiów | stacjonarne | |
| Temat projektu | Zewnętrzna stacja pogodowa na ESP32. | | | |

1. **Cele projektu**

Celem projektu było stworzenie zewnętrznej stacji pogodowej zbudowanej z wykorzystaniem ESP32, wysyłającej dane z czujników na serwer MQTT za pośrednictwem połączenia bezprzewodowego Wi-Fi. Przesyłanymi danymi są temperatura, wilgotność powietrza, ciśnienie, poziom zaćmienia, zawartość pyłów PM1.0,PM2.5 i PM10.0. Pomiary są również wyświetlane na wyświetlaczu OLED.

1. **Wykorzystany sprzęt i rozwiązania**
   1. **Sprzęt**

Do zrealizowania projektu wykorzystano następujący sprzęt:

- Płytka ESP32 WiFi

- czujnik pyłów zawieszonych PMS3003

- czujnik temperatury, wilgotności, ciśnienia BME680

- czujnik zaćmienia

- wyświetlacz OLED SH1106

- router MikroTik

- serwer

* 1. **Środowisko programowania**

Do programowania ESP32 użyto środowiska ArduinoIDE z zainstalowanymi wymaganymi bibliotekami.

* 1. **Zastosowane protokoły i technologie**
     1. **Łączność bezprzewodowa przez WiFi**

Do połączenia z serwerem MQTT utworzyliśmy sieć bezprzewodową przez MikroTik-57B99B, która umożliwia komunikację ESP32 z serwerem. ESP32 łączy się z siecią dzięki użyciu funkcji w kodzie podając jej SSID i hasło. Połączenie jest ustanawiane w setup().

* + 1. **Usługa serwera MQTT i komunikacja**

MQTT (Message Queuing Telemetry Transport) to lekki i otwarty protokół komunikacyjny stworzony do efektywnej wymiany informacji między urządzeniami w architekturze typu klient-serwer, zwłaszcza w kontekście Internetu Rzeczy (IoT). Komunikacja opiera się na modelu publish-subscribe, co oznacza, że urządzenia mogą publikować (wysyłać) wiadomości do tematów (topic) oraz subskrybować (odbierać) wiadomości z określonych tematów. Wiadomości są grupowane w tematy (topics), co umożliwia elastyczną organizację i filtrowanie komunikatów.

Serwer z systemem operacyjnym Debian 12 posiada zainstalowaną usługę MQTT Mosquitto na porcie 443. Połączenie z serwerem jest nawiązywane dzięki zastosowaniu funkcji w kodzie podając IP serwera, port oraz nazwę użytkownika.

Po wykonaniu pomiarów na brokerze MQTT publikowane są wyniki w topic’ach:

Wyniki z (czujnik zaćmienia):

- topic0 = "WeatherStation/Light"

Wyniki z BME680 (czujnik temp., wilgotności, ciśnienia):

- topic1 = "WeatherStation/Pressure\_hPa"

- topic2 = "WeatherStation/Humidity\_%"

- topic3 = "WeatherStation/Temp\_C"

Wyniki z PMS3003 (czujnik pyłów zawieszonych):

- topic4 = "WeatherStation/PM1.0"

- topic5 = "WeatherStation/PM2.5"

- topic6 = "WeatherStation/PM10.0"

* + 1. **Standard I²C – połączenie wyświetlacza i BME680**

Wyświetlacz OLED oraz BME680 są połączone do ESP32 za pomocą magistrali I²C. Jest to możliwe, ponieważ urządzenia podłączone do tej magistrali używają unikalnych adresów, co pozwala im współdziałać na wspólnym kanale komunikacyjnym. Oba urządzenia z ESP32 są połączone za pomocą dwóch linii magistrali - SDA (data)(na ESP pin 21) i SCL (zegar) (na ESP pin 22). W kodzie z biblioteką Wire.h jest możliwe ustanowienie pinów jako magistrala I²C. Służy do tego funkcja Wire.begin(21, 22);.

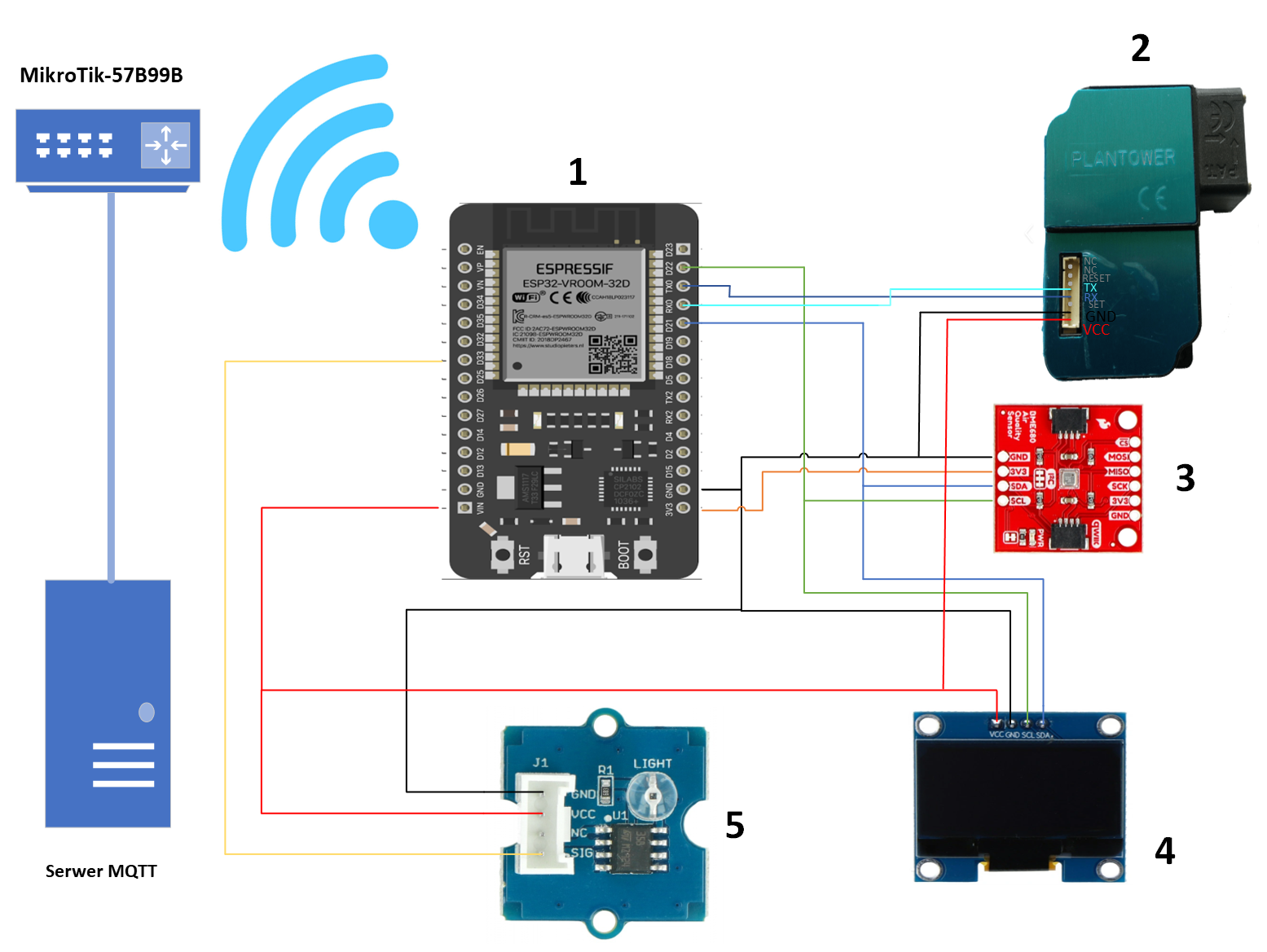
* + 1. **Kompilacja i programowanie ESP32 przez USB**

Przed podłączeniem ESP32 należy zainstalować sterowniki do *CP210x USB to UART bridge*. Bez nich środowisko ArduinoIDE nie będzie w stanie wykryć podłączonej płytki.

Program napisany w języku C, jest kompilowany w środowisku ArduinoIDE. Proces programowania ESP32 jest również dokonywany w tym środowisku, uprzednio podłączając płytkę do komputera za pośrednictwem USB. Aby programowanie przebiegło pomyślnie należy odłączyć piny TX0 i RX0.

* + 1. **Konfig**

nh

1. **Schemat**

1.Płytka ESP32 WiFi

2.Czujnik pyłów zawieszonych PMS3003

3.Czujnik temperatury, wilgotności, ciśnienia BME680

4.Wyświetlacz OLED SH1106

5.Czujnik zaćmienia

1. **Kod programu na ESP32**

//Zewnętrzna stacja pogodowa

//Autorzy: Michał Miksiewicz, Michał Pasieka

#include "PMS.h"

#include <SPI.h>

#include <Wire.h>

#include <Adafruit\_Sensor.h>

#include <Adafruit\_BME680.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SH1106.h>

#include <WiFi.h>

#include <PubSubClient.h>

// WiFi

WiFiClient espClient;

PubSubClient client(espClient);

const char \*ssid = "MikroTik-57B99B";

const char \*password = "IOT\_wifi";

// Wyswietlacz

#define OLED\_SDA 21

#define OLED\_SCL 22

Adafruit\_SH1106 display(21, 22); // definiujemy piny, do których podłączony został wyświetlacz

// Czujnik czastek stalych

PMS pms(Serial);

PMS::DATA data;

float avg\_pm10 = 0;

float avg\_pm25 = 0;

float avg\_pm100 = 0;

// Czujnik bme680

Adafruit\_BME680 bme;

float pres = 0;

int hum = 0;

float temp = 0;

// Czujnik zacmienia

int analogPin = 33;

int light\_val = 0;

// Flagi i zmienne odslugujace bledy

int i = 0, nd = 0;

int correct\_sleep = 1800, error\_sleep = 900; // w sekundach

bool PMS\_ERR = false, isCheckingPM = false, isErrorSleep = false;

// MQTT Broker

const char \*mqtt\_broker = "10.0.2.120";

const int   mqtt\_port = 443;

const char \*mqtt\_username = "esp32WeatherStation";

const char \*mqtt\_password = "public";

const char \*topic0 = "WeatherStation/Light";

const char \*topic1 = "WeatherStation/Pressure\_hPa";

const char \*topic2 = "WeatherStation/Humidity\_%";

const char \*topic3 = "WeatherStation/Temp\_C";

const char \*topic4 = "WeatherStation/PM1.0";

const char \*topic5 = "WeatherStation/PM2.5";

const char \*topic6 = "WeatherStation/PM10.0";

void callback(char \*topic, byte \*payload, unsigned int length)

{

    Serial.print("Wiadomość dotarła: ");

    Serial.println(topic);

    Serial.print("Wiadomosc:");

    for (int i = 0; i < length; i++)

    {

        Serial.print((char)payload[i]);

    }

    Serial.println();

    Serial.println("-----------------------");

}

void PomiarBme680(int seconds)

{

    for (int c = 0; c <= seconds; c++)

    {

        pres = bme.readPressure() / 100.0F;

        hum = bme.readHumidity();

        temp = bme.readTemperature();

        light\_val = analogRead(analogPin);

        light\_val = map(light\_val, 0, 4095, 0, 100);

        Serial.print("Zacmienie: ");

        Serial.println(light\_val);

        Serial.print("Temperature = ");

        Serial.print(temp);

        Serial.println(" \*C");

        Serial.print("Pressure = ");

        Serial.print((int)pres);

        Serial.println(" hPa");

        Serial.print("Humidity = ");

        Serial.print(hum);

        Serial.println(" %");

        if (!isCheckingPM)

        {

            display.clearDisplay();

            display.setTextColor(WHITE);

            display.setCursor(0, 0);

            if (!isErrorSleep)

                display.print("Pomiar PM: ");

            else

            {

                display.print("Czujnik nie odpowiada");

                display.print(error\_sleep);

                display.println(" s uspienia");

            }

            display.setCursor(0, 20);

            display.print("PM1.0:  ");

            display.println(avg\_pm10);

            display.print("PM2.5:  ");

            display.println(avg\_pm25);

            display.print("PM10.0: ");

            display.println(avg\_pm100);

            display.setCursor(0, 55);

            display.print("T:");

            display.print(temp);

            display.print(" P:");

            display.print((float)pres, 1);

            display.print(" H:");

            display.print(hum);

            display.display();

        }

        char msg\_out[20];

        sprintf(msg\_out, "%d", light\_val);

        client.publish(topic0, msg\_out);

        sprintf(msg\_out, "%f", pres);

        client.publish(topic1, msg\_out);

        sprintf(msg\_out, "%d", hum);

        client.publish(topic2, msg\_out);

        sprintf(msg\_out, "%f", temp);

        client.publish(topic3, msg\_out);

        client.loop();

        delay(1000);

    }

}

void setup()

{

    Serial.begin(9600);

    Wire.begin(21, 22);//inicjacja biblioteki wire (i2c)

    if (!bme.begin())//jesli

    {

        Serial.println("Could not find a valid BME680 sensor, check wiring!");

        while (1)

            ;

    }

    Serial.println("BME680 sensor found!");

    delay(1000);

    // wifi

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL\_CONNECTED)

    {

        delay(500);

        Serial.println("Łączenie z WiFi...");

    }

    // mqtt

    client.setServer(mqtt\_broker, mqtt\_port);

    client.setCallback(callback);

    display.begin(SH1106\_SWITCHCAPVCC, 0x3C); // definiujemy rodzaj użytego wyświetlacza oraz adres I2C

    pms.passiveMode();                        // Tryb pasywny w tym trybie czujnik wysyla dane tylko za żądaniem

    display.clearDisplay();

    display.display();

}

void loop()

{

  //poloczenie z serwerem MQTT

    while (!client.connected())

    {

        String client\_id = "esp32-klient-";

        client\_id += String(WiFi.macAddress());

        Serial.printf("Klient %s łączy się z publicznym brokerem MQTT\n", client\_id.c\_str());

        if (client.connect(client\_id.c\_str(), mqtt\_username, mqtt\_password))

        {

            Serial.println("Połączono z brokerem MQTT");

        }

        else

        {

            Serial.print("Błąd Sieci");

            Serial.print(client.state());

            delay(2000);

        }

    }

    Serial.println("Wybudzanie czujnika...");

    pms.wakeUp(); // Tryb operacyjny czujnika (wybudzenie)

    //definicja srednich odczutow z 10 pomiarów

    avg\_pm10 = 0;

    avg\_pm25 = 0;

    avg\_pm100 = 0;

    while (true)

    {

        PomiarBme680(30); //pomiar czujnika bme680 30 razy z 1s delay

        isCheckingPM = true; //status pobierania danych

        Serial.print("Numer pomiaru: ");

        Serial.println(i + 1);

        pms.requestRead(); //wyslanie żądania w trybie pasywnym do czujnika

        if (pms.readUntil(data)) //odczyt danych

        {

            Serial.print("\nPM 1.0 (ug/m3): ");

            Serial.println(data.PM\_AE\_UG\_1\_0);

            avg\_pm10 += data.PM\_AE\_UG\_1\_0;

            Serial.print("PM 2.5 (ug/m3): ");

            Serial.println(data.PM\_AE\_UG\_2\_5);

            avg\_pm25 += data.PM\_AE\_UG\_2\_5;

            Serial.print("PM 10.0 (ug/m3): ");

            Serial.println(data.PM\_AE\_UG\_10\_0);

            avg\_pm100 += data.PM\_AE\_UG\_10\_0;

            display.clearDisplay();

            display.setTextColor(WHITE);

            display.setCursor(0, 0);

            display.println("Wykonywanie 10 probek pomiarow");

            display.setCursor(0, 15);

            display.print("Numer pomiaru: ");

            display.println(i + 1);

            display.setCursor(0, 30);

            display.print("PM1.0 : ");

            display.println(data.PM\_AE\_UG\_1\_0);

            display.print("PM2.5 : ");

            display.println(data.PM\_AE\_UG\_2\_5);

            display.print("PM10.0: ");

            display.println(data.PM\_AE\_UG\_10\_0);

            display.setCursor(0, 55);

            display.print("T:");

            display.print(temp);

            display.print(" P:");

            display.print((float)pres, 1);

            display.print(" H:");

            display.print(hum);

            display.display();

            i++;

        }

        else //jesli dane nie istnieja wyswietl ostatnie poprawne wartosci

        {

            Serial.println("\nNo data.");

            display.clearDisplay();

            display.setTextColor(WHITE);

            display.setCursor(0, 0);

            display.println("Wykonywanie 10 probek pomiarow");

            display.setCursor(0, 15);

            display.setTextColor(WHITE);

            display.setCursor(0, 25);

            display.print("Brak pomiaru: ");

            display.println(nd + 1);

            display.setCursor(0, 55);

            display.print("T:");

            display.print(temp);

            display.print(" P:");

            display.print((float)pres, 1);

            display.print(" H:");

            display.print(hum);

            display.display();

            nd++;

        }

        if (nd >= 10) //jesli dane nie zostaly wyswietlone 10 raz z rzedu

        {

            PMS\_ERR = true; //ustaw flage bledu

            break;

        }

        if (i >= 10)

            break;

    }

    if (PMS\_ERR)

    {

        isCheckingPM = false;

        Serial.println("Czujnik nie odpowiada.");

        display.clearDisplay();

        display.setTextColor(WHITE);

        display.setCursor(0, 0);

        display.print("Czujnik nie odpowiada");

        display.print(error\_sleep);

        display.println(" s uspienia");

        display.display();

        pms.sleep();

        isErrorSleep = true;

        PomiarBme680(error\_sleep);

        isErrorSleep = false;

    }

    if (!PMS\_ERR)

    {

        isCheckingPM = false;

        avg\_pm10 /= 10;

        avg\_pm25 /= 10;

        avg\_pm100 /= 10;

        Serial.println("Średnia 10 pomiarow: ");

        Serial.println(avg\_pm10);

        Serial.println(avg\_pm25);

        Serial.println(avg\_pm100);

        // Wyswietlacz

        display.clearDisplay();

        display.setTextColor(WHITE);

        display.setCursor(0, 0);

        display.print("Pomiar PM: ");

        display.setCursor(0, 20);

        display.print("PM1.0:  ");

        display.println(avg\_pm10);

        display.print("PM2.5:  ");

        display.println(avg\_pm25);

        display.print("PM10.0: ");

        display.println(avg\_pm100);

        display.setCursor(0, 55);

        display.print("T:");

        display.print(temp);

        display.print(" P:");

        display.print(pres);

        display.print(" H:");

        display.print(hum);

        display.display();

        char msg\_out[20];

        sprintf(msg\_out, "%f", avg\_pm10);

        client.publish(topic4, msg\_out);

        sprintf(msg\_out, "%f", avg\_pm25);

        client.publish(topic5, msg\_out);

        sprintf(msg\_out, "%f", avg\_pm100);

        client.publish(topic6, msg\_out);

    }

    //Reset flag i zmiennych iteracyjnych

    i = 0;

    nd = 0;

    PMS\_ERR = false;

    Serial.println("[PMS3003] Going to sleep.");

    pms.sleep();  //uspij czujnik pms3003

    PomiarBme680(correct\_sleep); //oczytuj tylko czujnik bme680 przez "correct\_sleep" sekund czasu

}

Wykorzystane biblioteki:

#include "PMS.h” - biblioteka obsługi i komunikacji z czujnikami pyłów PMS

#include <SPI.h> - biblioteka komunikacji z wieloma urządzeniami

#include <Wire.h> - biblioteka obsługi i komunikacji z urządzeniami przez I²C

#include <Adafruit\_Sensor.h> - biblioteka zapewniająca jednolity interfejs dla różnych czujników

#include <Adafruit\_BME680.h> - bliblioteka obsługi i komunikacji z czujnikiem BME680

#include <Adafruit\_GFX.h> - biblioteka obsługi graficznej wyświetlaczy (ogólnie)

#include <Adafruit\_SH1106.h> - bliblioteka obsługi i komunikacji z wyświetlaczem OLED SH1106

#include <WiFi.h> - bilbioteka obsługi połączenia przez moduł WiFi w ESP32

#include <PubSubClient.h> - biblioteka obsługi połączenia z MQTT oraz akcji z nim związanych

1. **Działanie programu**

Na początku programu importowane są biblioteki, deklarowane zmienne oraz stałe (np. zmienne przechowujące wyniki pomiarów, stałe jak np. IP serwera MQTT, SSID sieci bezprzewodowej) czy flagi wykorzystane w kodzie. Inicjalizowane są tez funkcje z bibliotek (np. PMS, WiFiClient, PubSubClient). Ustawiane są też piny dla wyświetlacza.

Zadeklarowane są też funkcje:

callback(char \*topic, byte \*payload, unsigned int length) – wiadomość zwrotna do serwera MQTT

PomiarBme680(int seconds) – funkcja odczytująca dane z BME680 i wysyłająca je na serwer MQTT. Funkcja zawiera instrukcje dla wyświetlacza w celu reprezentacji danych. Argumentem funkcji jest liczba sekund, czyli czas trwania pomiarów (1 pomiar trwa ok. 1s)

Program rozpoczyna się wykonaniem funkcji **setup()**;.

Są w niej inicjalizowane piny do magistrali I²C, następnie następuje połączenie z BME680. Jeśli nie ma połączenia program nie wykona się dalej, gdyż nie będzie możliwe odczytywanie pomiarów. W dalszej kolejności następuje łączenie z siecią WiFi, a następnie z serwerem MQTT. Następnie zostaje zainicjalizowane połączenie z wyświetlaczem i wyczyszczenie go oraz przełączenie PMS3003 w tryb pasywny.

Po funkcji setup(); program zaczyna działanie w pętli **loop()**;

Na początku sprawdzany jest status połączenia z serwerem MQTT (jeśli nie połączony, następuje nawiązanie połączenia; jeśli połączony proces ten jest pomijany). Po tym następuje wybudzenie PMS3003 oraz wyzerowanie zmiennych przechowujących wyniki pomiarów pyłów zawieszonych. Następnie wykonywana jest pętla while(true), która zawiera logikę wykonywania pomiarów przez urządzenia oraz instrukcje dla wyświetlacza do reprezentacji wyników i stanu pomiarów (pominiemy omawianie ich).

Pomiary rozpoczynają się wykonaniem 30 pomiarów BME680 ( *PomiarBme680(30);* ) po nich następuje ustawienie flagi rozpoczęcia próbkowania i wykonanie 10 próbek pomiaru pyłów za pomocą PMS3003 (jeśli nie dostaniemy odpowiedzi od czujnika inkrementujemy zmienną *nd* przechowującą liczbę „pustych próbek”). Po wykonaniu 10 poprawnych pomiarów, te są uśredniane oraz wysyłane do brokera MQTT, po czym następuje uśpienie PMS3003 na 30 min (*PomiarBme680(correct\_sleep); ,gdzie correct\_sleep =1800 ,czyli 1800 pomiarów BME680 trwających łącznie około 1800s)* . Jeżeli natomiast 10 próbek pod rząd będzie „pustych” ustawiana jest flaga błędu. Pomiary pyłów nie są wysyłane do brokera MQTT, a PMS3003 zostaje uśpiony na 15min(*PomiarBme680(error\_sleep);, gdzie error\_sleep=900, czyli około 900s*),a następnie na 30 min (*PomiarBme680(correct\_sleep);*) czyli łącznie 45. Po zakończeniu obu przypadków flagi i zmienne iteracyjne są zerowane. Po tym pomiary są wykonywane ponownie.

1. **Zrzuty ekranu z działania**

Opis:

Zdj

Opis:

Zdj

Itd. todo