System monitorowania jakości powietrza

AirGuard ESP32

Dokumentacja techniczna

Wykonał: Filip Skop
183952
1EF-DI

Rzeszów 2026

Spis treści

Część I

1. Założenie projektowe

W otaczającym nas środowisku z dnia na dzień pojawia się coraz więcej zanieczyszczeń, co negatywnie wpływa na zdrowie i komfort życia. W przestrzeni publicznej instaluje się coraz więcej stacji monitorujących, jednak ich zagęszczenie bywa niewystarczające – w sytuacji, gdy najbliższy punkt pomiarowy oddalony jest o kilka kilometrów, prezentowane dane nie odzwierciedlają realnego stanu powietrza w naszym bezpośrednim otoczeniu. Z pomocą przychodzi autonomiczny, własnoręcznie zbudowany system monitorowania jakości powietrza. Urządzenie zostało zaprojektowane w sposób umożliwiający jego łatwy montaż na balkonie, parapecie lub wewnątrz domu, dostarczając precyzyjnych i w pełni lokalnych odczytów.

2. Projekt na tle innych rozwiązań

Na rynku konsumenckim dostępnych jest wiele gotowych czujników oraz zintegrowanych stacji pogodowych. W dużej mierze systemy te charakteryzują się jednak zamkniętą architekturą – przechowują jedynie ostatnią zmierzoną wartość i prezentują ją wyłącznie na wbudowanym wyświetlaczu, co uniemożliwia długoterminową agregację danych i analizę trendów. Główną zaletą projektu jest jego modułowość oraz możliwość łatwej rozbudowy o kolejne sensory. Dodatkowo urządzenie nie ogranicza się do odczytu lokalnego; dane są bezprzewodowo przesyłane do bazy danych (serwer Proxmox/InfluxDB) hostowanej w sieci lokalnej lub na zewnętrznych serwerach. Zapewnia to pełną elastyczność w doborze miejsca pracy stacji, a zebrane pomiary są prezentowane użytkownikowi w formie czytelnych, dynamicznych wykresów w czasie rzeczywistym.

Część II

3. Wykorzystane elementy

3.1. ESP32 DevKit v1

Układ SoC stanowiący jednostkę centralną stacji pomiarowej. Wybrany ze względu na taktowanie do 240 Mhz, wbudowany moduł Wi-Fi oraz fizyczne wsparcie dla wielu interfejsów szeregowych (UART, I2C), co czyni go idealną platformą IoT.

Rys. 1: ESP 32 DevKit v1

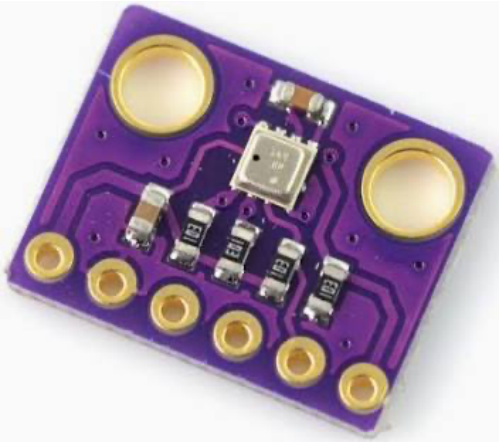
3.2. Laserowy czujnik pyłu PMS5003

Zaawansowany sensor działający na zasadzie rozpraszania światła laserowego na cząstkach stałych zawieszonych w powietrzu. Pozwala na precyzyjny pomiar frakcji PM2.5 oraz PM10.

Rys. 2: laserowy czujnik pyłu PMS5003

3.3. Cyfrowy sensor środowiskowy BME280

Zintegrowany czujnik połączony magistralą cyfrową, dostarczający dokładne dane o temperaturze, wilgotności względnej oraz ciśnieniu atmosferycznym.



Rys. 3: sensor BME280

3.4. Wyświetlacz graficzny SSD1306 OLED 0.96”

Miniaturowy wyświetlacz o rozdzielczości 128x64 pikseli, służący do prezentacji tekstowej i graficznej lokalnych odczytów pomiarowych w czasie rzeczywistym.

Rys. 4: wyświetlacz SSD1306 OLED

3.5. Dioda sygnalizacyjna LED RGB

Trójkolorowa dioda ze wspólną katodą, wysterowana sygnałem PWM z mikrokontrolera w celu płynnej zmiany barw sygnalizujących jakość powietrza.

Rys. 5: dioda RED RGB

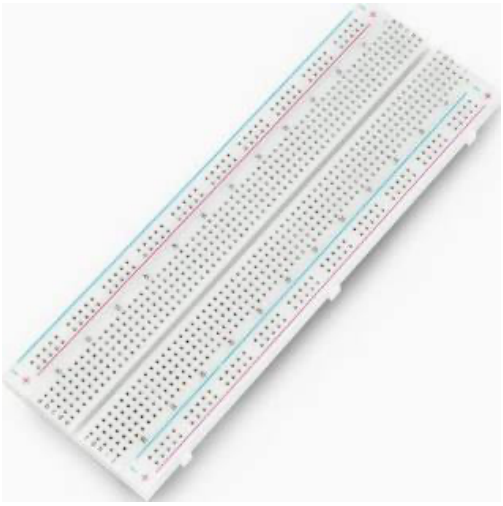
3.6. Przycisk tact-switch

Fizyczny przycisk monostabily podpięty do pinu gpio z wewnętrznym podciągnięciem, realizujący funkcję wybudzania układu ze stanu uśpienia i wywoływania natychmiastowej procedury pomiarowej.

Rys. 6: przycisk back-switch

3.7. Elementy prototypowe

Płytka stykowa oraz przewody połączeniowe męsko-męskie i żeńsko-męskie, umożliwiające szybkie i bezlutowe zestawienie całego obwodu elektrycznego na etapie testów laboratoryjnych.



Rys. 7: płytka stykowa

Rys. 8: przewody połączeniowe

4. Konfiguracja ESP32

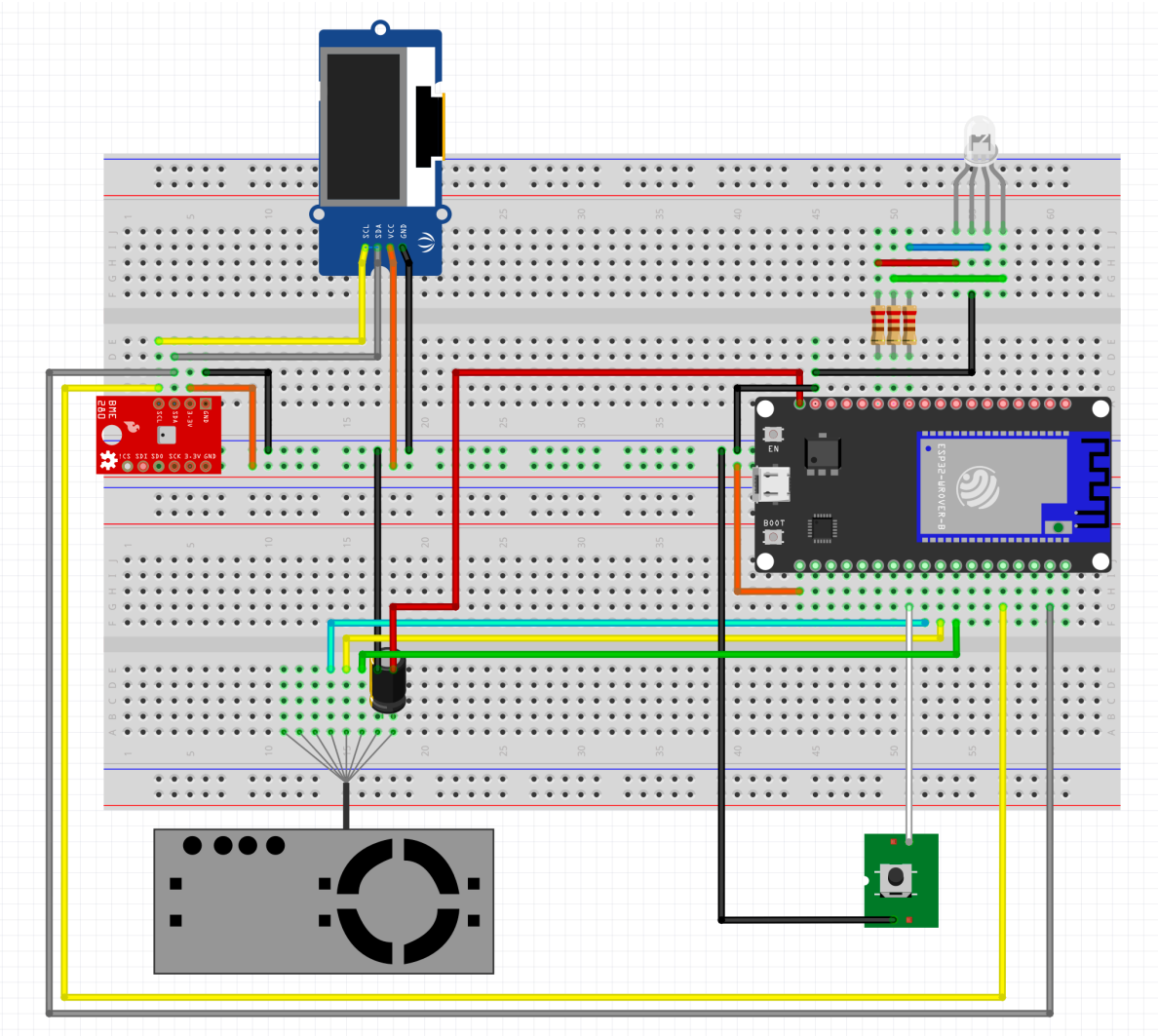
4.1. Połączenie fizyczne czujników

Połączenia zostały zaprojektowane i zweryfikowane w oprogramowaniu Fritzing. Architektura połączeń wykorzystuje następujące przypisania pinów ESP32:

- Magistrala I2C: Pin GPIO21 (SDA) oraz GPIO22 (SCL) współdzielone równolegle przez czujnik BME280 oraz wyświetlacz OLED SSD1306.
- Interfejs UART: Port sprzętowy Serial2 (GPIO16 jako RX2, GPIO17 jako TX2) podłączony dedykowanie do wyprowadzeń komunikacyjnych czujnika PMS5003.
- Sygnalizacja RGB: Trzy piny cyfrowe GPIO przypisane do poszczególnych struktur diody (Czerwona, Zielona, Niebieska).

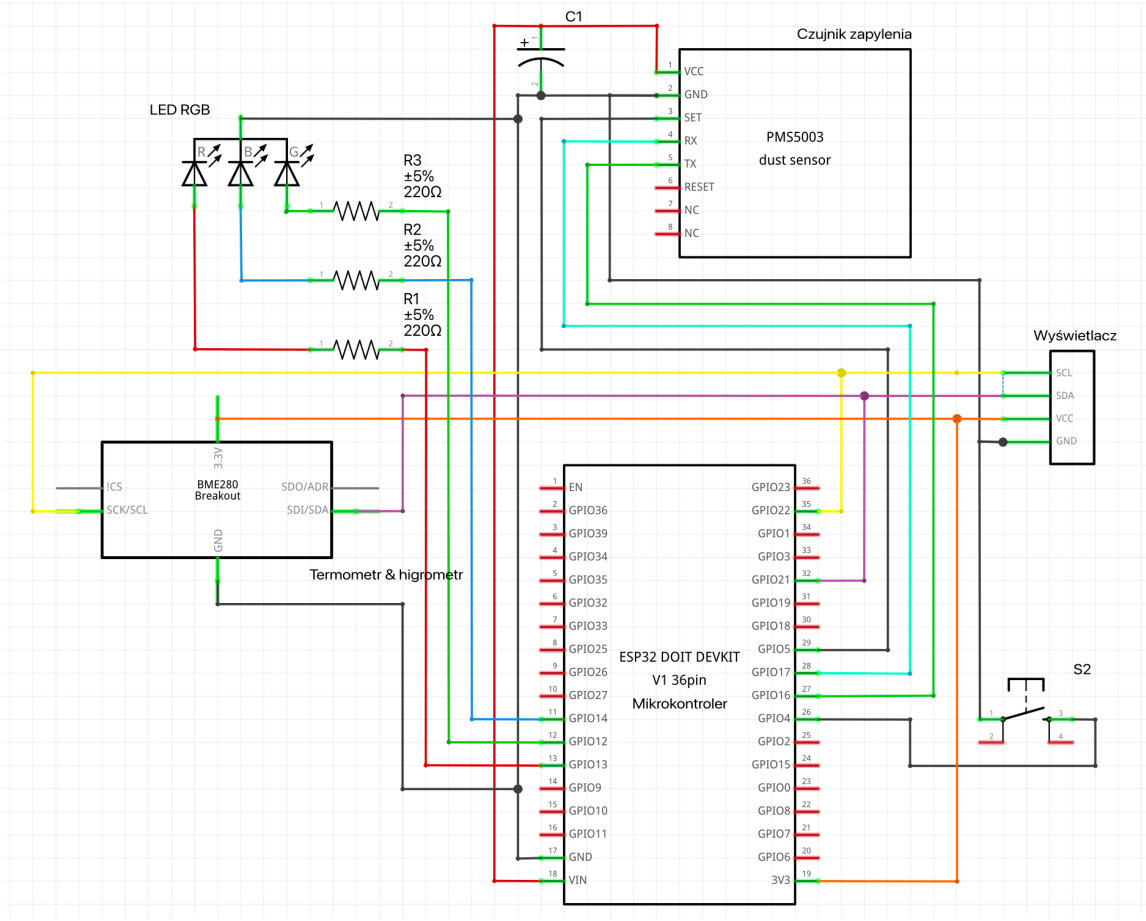
- Przycisk wybudzania: Pin GPIO ze wsparciem dla przerwań zewnętrznych podpięty do przycisku Tact-Switch.

Układ połączeń na płytce prototypowej w programie Fritzing:



Rys. 9: układ połączeń w programie Fritzing

Schemat połączeń w programie Fritzing:



Rys. 10: schemat połączeń w programie Firtzing

4.2. Pobieranie danych i obsługa magistral

Połączenia zostały zaprojektowane i zweryfikowane w oprogramowaniu Fritzing. Architektura połączeń wykorzystuje następujące przypisania pinów ESP32:

- **Czas rozgrzewania (Warm-up):** Program uwzględnia opóźnienie ok. 30 sekund na ustabilizowanie pracy wbudowanego wentylatora w czujniku PMS5003 przed pobraniem właściwej próbki pomiarowej, co zapobiega fałszowaniu wyników.
- **Zarządzanie energią:** Implementacja trybów niskoprądowych (Sleep oraz WakeUp) dla sensora laserowego kontrolowanych bezpośrednio z poziomu kodu programu za pomocą linii sterującej czujnika.

- **Interpretacja jakości:** Zebrane surowe wartości w µg/m3 są programowo analizowane i mapowane na progi jakości powietrza (Dobry, Umiarkowany, Zły) zgodnie z wytycznymi GIOS.

4.3. Komunikacja sieciowa MQTT

Po zestawieniu połączenia z lokalnym punktem Wi-Fi, mikrokontroler uruchamia klienta protokołu MQTT. Pomiary są formatowane w ciąg tekstowy i asynchronicznie publikowane do brokera MQTT, co zapewnia minimalny narzut sieciowy i odporność na zrywanie połączeń.

W zależności od wyboru miejsca mamy kilka możliwości połączenia z bazą danych:

- **Sieć lokalna (LAN)** - Klasyczny scenariusz, w którym serwer bazodanowy oraz stacja pomiarowa AirGuard operują w obrębie tej samej podsieci fizycznej. W tym układzie transmisja pakietów odbywa się bezpośrednio i nie są wymagane żadne dodatkowe kroki konfiguracyjne.
- **Wirtualna sieć prywatna (VPN)** - Bezpieczny tunel szyfrowany ustanowiony bezpośrednio między lokalizacją stacji a serwerem. Rozwiązanie to dedykowane jest dla miejsc, w których nie ma możliwości fizycznego dopowiedzenia jednej sieci wspólnej z głównym serwerem. W celu realizacji tego scenariusza wykorzystano infrastrukturę opartą na dwóch routerach sieciowych:

- Mikrotik HEX S zlokalizowany w domu rodzinnym, stanowiący węzeł centralny, do którego podpięty jest serwer główny.
- Mikrotik Hap AC^2, lokalizowany w akademiku, realizujący połączenie dla sieci LAN, w której pracowała stacja pogodowa.

Po wdrożeniu odpowiednich reguł pakietów filtrowania (Firewall) na obu urządzeniach brzegowych oraz właściwym skonfigurowaniu routingu

pomiędzy odizolowanymi sieciami wirtualnymi (VLAN) uzyskano stabilność i czasy odpowiedzi tożsame z pracą w bezpośredniej sieci lokalnej.

- **Zewnętrzny broker MQTT** - Scenariusz chmurowy, w którym stacja AirGuard – znajdując się w dowolnej sieci Wi-Fi z dostępem do Internetu – łączy się z zewnętrzną platformą pośredniczącą (w projekcie wykorzystano platformu *cloud.shiftr.io*), która następnie przekazuje pakiety danych na serwer domowy. Jest to rozwiązanie najbardziej elastyczne z punktu widzenia wdrożenia, lecz generujące opłaty abonamentowe w wysokości od kilku do kilkunastu złotych miesięcznie (w zależności od wybranego dostawcy usług SaaS). Pod względem technicznym transmisja charakteryzuje się taką samą stabilnością jak w przypadku LAN i VPN

4.4. Kompletny program sterujący dla ESP32

Poniższy program stanowi kompletny kod źródłowy oprogramowania układowego stacji AirGuard, podzielony na logiczne bloki funkcjonalne:

- Konfiguracja sieci bezprzewodowej (wykorzystanie zewnętrznego brokera MQTT):

```
const char* ssid = "iPhone (Filip)";
const char* password = "SKOpiasty1";
const char* mqtt_server = "airguard-v1-0.cloud.shiftr.io";
const char* mqtt_user = "airguard-v1-0";
const char* mqtt_pass = "tmRFwytUbkcxyAPF";
```

Rys. 11: konfiguracja sieci bezprzewodowej

- **Obsługa podsystemu sieciowego i synchronizacji czasu**

Procedura `setup_wifi()` nawiązuje połączenie radiowe i inicjuje synchronizację czasu wewnętrznego RTC z serwerami NTP (`pool.ntp.org`). Funkcja `reconnect()` realizuje bezpieczne, nieblokujące wznawianie sesji z brokerem MQTT nie częściej niż raz na 7 sekund. Dodatkowo w przypadku braku połączenia z siecią Wi-Fi stacja nie próbuje łączyć się z brokerem MQTT.

```
void setup_wifi() {
  display.clearDisplay();
  display.setCursor(0,0);
  display.println("Siec: " + String(ssid));
  display.println("Status: Laczenie...");
  display.display();

  WiFi.begin(ssid, password);
  int counter = 0;
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    counter++;
    if(counter > 10) {
      Serial.println("Blad WiFi");
      break;
    }
  }
  //ustawienie czasu jak mamy wifi
  if(WiFi.status() == WL_CONNECTED) {
    configTime(3600, 3600, "pool.ntp.org");
    Serial.println("Polaczono");
  }
}
```

Rys. 12: funkcja łącząca z Wi-Fi

```
void reconnect() {
  static unsigned long lastReconnectAttempt = 0;
  if (WiFi.status() != WL_CONNECTED) return;

  if (!client.connected()) {
    unsigned long now = millis();
    // Próbuj łączyć się nie częściej niż co 7 sekund
    if (now - lastReconnectAttempt > 7000) {
      lastReconnectAttempt = now;
      Serial.print("Proba MQTT...");
      if (client.connect("AirGuard_ESP32", mqtt_user, mqtt_pass)) {
        Serial.println("polaczono");
      } else {
        Serial.print("blad, rc=");
        Serial.print(client.state());
      }
    }
  }
}
```

Rys. 13: Funkcja łącząca z brokerem MQTT

- **Sprzętowo - programowa sekwencja budzenia sensora laserowego, odczyt i wysyłka danych do bazy, wyświetlania wyników na ekranie oraz ponowne uśpienie**

Ten fragment kodu bezpośrednio kontroluje zużycie energii oraz dokładność pomiaru. Zamiast utrzymywać laser i wentylator czujnika PMS5003 w ciągłej pracy (co doprowadziłoby do szybkiego zużycia komory optycznej i zafałszowania wyników przez osiadający kurz), procesor steruje fizycznym pinem `PMS_SET_PIN`.

Wybudza on sensor, odlicza asynchronicznie dokładnie 30 sekund na ustabilizowanie przepływu powietrza, pobiera próbkę, a po wysłaniu danych natychmiast odcina zasilanie lasera, wprowadzając go z powrotem w tryb głębokiego uśpienia.

```
digitalWrite(PMS_SET_PIN, HIGH);

for(int i = 30; i > 0; i--) {
  display.clearDisplay();
  display.setCursor(0,10);
  display.setTextSize(1);
  display.println("Pomiar za:");
  display.setTextSize(3);
  display.setCursor(40, 30);
  display.print(i);
  display.display();
  delay(1000);
}
```

Rys. 14: wybudzenie sensora i odliczanie do pomiaru

```
if (pms.readUntil(data)) {
  int pm25 = data.PM_AE_UG_2_5;
  int pm10 = data.PM_AE_UG_10_0;
  float temp = bme.readTemperature();
  float hum = bme.readHumidity();
  float pres = bme.readPressure() / 100.0F;

  //wysylka do influxa
  if (client.connected()) {
    client.publish("airguard/pm25", String(pm25).c_str());
    client.publish("airguard/pm10", String(pm10).c_str());
    client.publish("airguard/temp", String(temp, 1).c_str());
    client.publish("airguard/hum", String(hum, 1).c_str());
    client.publish("airguard/pres", String(pres, 1).c_str());
  }
```

Rys. 15: odczyt danych z czujników i wysyła za pomocą brokera MQTT do bazy danych influxDB

```
display.clearDisplay();
display.setTextSize(2);
display.setCursor(0, 0);
display.setTextColor(SSD1306_WHITE);
display.print("PM2.5:"); display.println(pm25);

display.setTextSize(1);
display.println("----------------------");
display.print("PM10: "); display.print(pm10); display.println(" ug/m3");
display.print("Temp: "); display.print(temp, 1); display.println(" C");
display.print("Wilg: "); display.print(hum, 0); display.println(" %");
display.print("Cisn: "); display.print(pres, 0); display.println(" hPa");
display.display();

delay(10000); // Wynik widoczny przez 10s
```

Rys. 16: Wyświetlanie wyników na ekranie

```
digitalWrite(PMS_SET_PIN, LOW);
display.clearDisplay();
display.setTextSize(1);
display.setCursor(0, 10);
display.println("Sensor uspiony.");
display.println("");
display.println("Wcisnij by wykonac ponowny pomiar");
display.display();
```

Rys. 17: uśpienie sensora

- **Zabezpieczenie przed zamrażaniem pętli głównej**

W konfiguracji sprzętowej zaimplementowano krytyczną dla stabilności funkcję sieciową. Domyślnie klient MQTT potrafi zablokować działanie całego mikrokontrolera na kilkanaście sekund, jeśli utraci łączność z serwerem zdalnym i czeka na odpowiedź.

```
setup_wifi();
client.setServer(mqtt_server, 1883);
client.setSocketTimeout(2);
```

Rys. 18: timeout dla gniazda sieciowego

Wprowadzenie ograniczenia timeoutu gniazda sieciowego do 2 sekund gwarantuje, że w przypadku utraty sygnału Wi-Fi (lub opóźnień sieciowych), urządzenie nie ulegnie „zawieszeniu” i zachowa pełną responsywność pętli głównej.

- **Hybrydowy mechanizm harmonogramowania pomiarów**

Najważniejsza część logiczna pętli `loop()`, która decyduje o autonomii stacji AirGuard. Algorytm weryfikuje status połączenia i dynamicznie przełącza źródło wyzwalania zdarzeń.

W przypadku stabilnego połączenia, odczytuje strukturę czasu z serwerów NTP i uruchamia procedurę pomiarową dokładnie w równych kwadransach czasu rzeczywistego. Jeśli sieć przestanie działać, układ automatycznie przechodzi na niezależne, wewnętrzne odmierzanie interwału za pomocą sprzętowego licznika milisekund procesora.

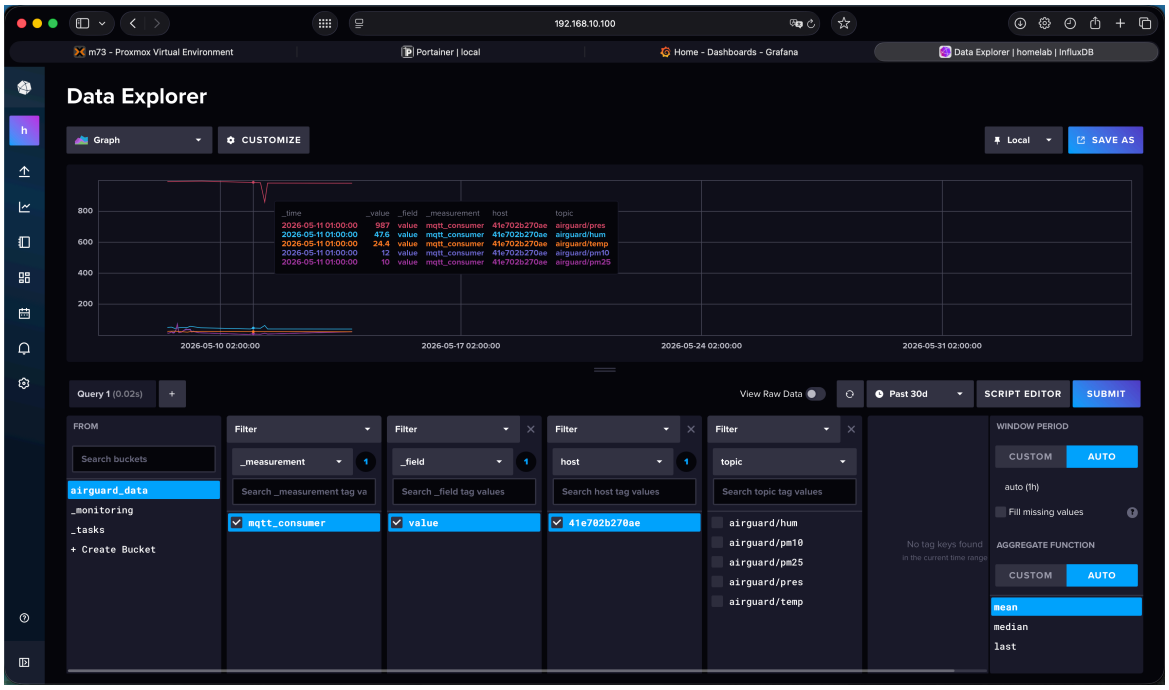
```
//auto pomary zegar
  if (WiFi.status() == WL_CONNECTED) {
    struct tm timeinfo;
    if (getLocalTime(&timeinfo)) {
      int m = timeinfo.tm_min;
      //rowne godziny
      if ((m == 0 || m == 15 || m == 30 || m == 45) && m != lastAutoMinute) {
        lastAutoMinute = m;
        performMeasurement();
        lastOfflineMillis = millis();
      }
    }
  } else {
    //zwykle oldiczanie jak nie ma wifi
    if (millis() - lastOfflineMillis >= OFFLINE_INTERVAL) {
      performMeasurement();
      lastOfflineMillis = millis();
    }
  }
```

Rys. 19: odliczanie 15 minut między pomiarami automatycznymi

5. Konfiguracja infrastruktury sieciowej i bazy danych

5.1. Baza danych serii czasowych InfluxDB

W przeciwieństwie do tradycyjnych relacyjnych baz danych (takich jak MySQL), w projekcie AirGuard zastosowano bazę danych **InfluxDB** dedykowaną dla serii czasowych (Time Series Database). Zapewnia ona najwyższą wydajność przy ciągłym zapisie strumieni pomiarowych oraz automatyczne indeksowanie danych znacznikiem czasu systemu operacyjnego.



Rys. 20: Widok w panelu InfluxDB

5.2. Serwer Proxmox, docker i wizualizacja Grafana

Całość infrastruktury serwerowej odpowiedzialnej za przetwarzanie, agregację oraz wizualizację danych pomiarowych została osadzona w środowisku chmury prywatnej. Warstwę abstrakcji sprzętowej realizuje system bare-metal hypervisor **Proxmox VE (Virtual Environment)** uruchomiony na serwerze domowym, co pozwala na optymalne zarządzanie zasobami obliczeniowymi i pełną izolację uruchomionych usług sieciowych.

W celu zapewnienia wysokiej skalowalności, łatwości migracji oraz minimalnego narzutu na pamięć RAM i procesor, w strukturze Proxmox wydzielono dedykowany kontener **LXC (Linux Containers)** pracujący pod kontrolą systemu operacyjnego Linux. Kontener ten pełni funkcję mikroserwera aplikacyjnego, wewnątrz którego zainstalowano silnik konteneryzacji **Docker**. Logiczna struktura przetwarzania danych wewnątrz środowiska Docker opiera się na czterech ściśle współpracujących ze sobą kontenerach aplikacyjnych (stos technologiczny TIG+M):

1. **Broker MQTT (Eclipse Mosquitto):** Lekki broker komunikatów implementujący protokół MQTT, stanowiący punkt dostępowy dla stacji pomiarowej AirGuard. Odpowiada za autoryzację urządzenia, odbieranie asynchronicznych pakietów sieciowych oraz zarządzanie kolejkami tematów (topics), na które publikowane są surowe odczyty pyłów zawieszonych i parametrów mikroklimatycznych.

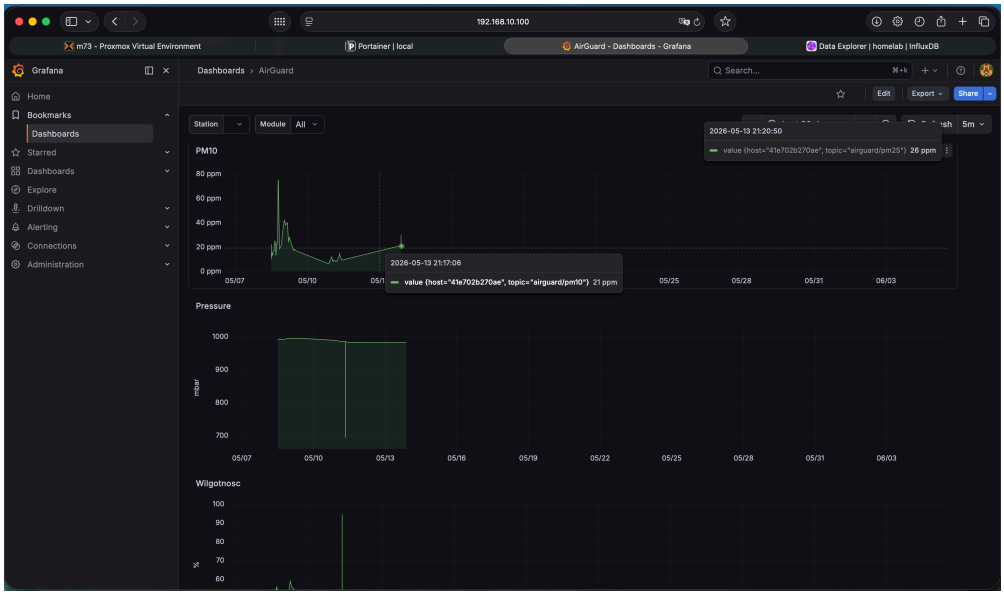
2. **Agent zbierający dane (Telegraf):** Autonomiczna usługa systemowa skonfigurowana jako subskrybent tematów na brokerze Mosquitto. Telegraf w czasie rzeczywistym przechwytuje napływające ciągi tekstowe, dokonuje ich wstępnej strukturyzacji, a następnie za pomocą dedykowanej wtyczki wyjściowej (output plugin) parsuje i strumieniuje dane bezpośrednio do silnika bazy danych.

3. **Baza danych serii czasowych (InfluxDB):** Wysoko wydajna baza danych zoptymalizowana pod kątem przechowywania i indeksowania danych pomiarowych powiązanych ze znacznikiem czasu (Time Series Database). Przechowuje ona pełną historię pomiarów ze stacji AirGuard, umożliwiając wykonywanie szybkich zapytań agregujących.

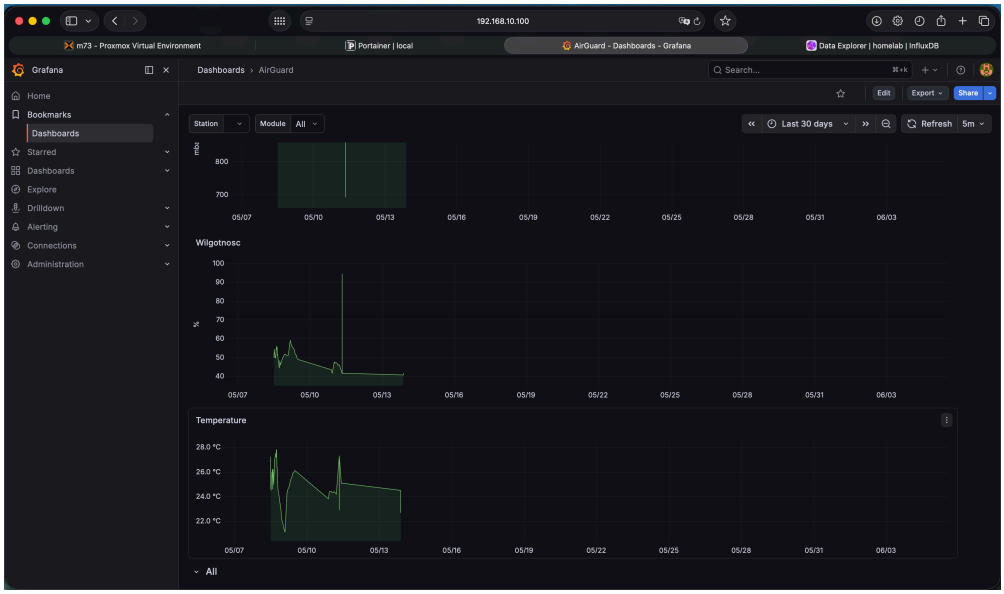
4. **Platforma analityczno-wizualizacyjna (Grafana):** Narzędzie podłączone bezpośrednio do bazy InfluxDB jako źródło danych (Data Source).

Odpowiada za warstwę prezentacji danych dla użytkownika końcowego. Pozwala na renderowanie dynamicznych paneli kontrolnych (Dashboards), generowanie dobowych oraz tygodniowych wykresów trendów, a także prowadzenie zaawansowanej analizy korelacji między wahaniami zanieczyszczeń powietrza a parametrami meteorologicznymi (takimi jak wilgotność czy ciśnienie) w czasie rzeczywistym.

Rys. 21



Rys. 22



Rys. 23

Rys. 21, Rys. 22 oraz Rys. 23 przedstawiają interfejs w Grafanie gdzie wyświetlane są wyniki pomiarów

Część III

Podsumowanie

Konfiguracja projektu nie odbyłaby się bez bibliotek open-source, które znacząco upraszczają komunikację między czujnikami a portami GPIO mikrokontrolera ESP32. Szeroka dostępność dokumentacji technicznej oraz przykładów aplikacyjnych ułatwiła szybkie poznanie tajników magistral cyfrowych, takich jak I2C oraz UART.

Wdrożenie warstwy sieciowo-serwerowej wymagało ugruntowanej wiedzy z zakresu inżynierii sieci oraz administracji systemami. Zamiast przestarzałych monolitów, zastosowano nowoczesny, skonteneryzowany stos technologiczny (Mosquitto, Telegraf, InfluxDB, Grafana) uruchomiony w silniku Docker wewnątrz kontenera LXC na hypervisorze Proxmox VE.

Zapewnienie bezpiecznej i stabilnej telemetryki wymusiło również zaawansowaną konfigurację urządzeń brzegowych MikroTik. Dzięki wdrożeniu szyfrowanych tuneli VPN, izolacji sieci VLAN oraz reguł zapory sieciowej (Firewall), stacja uzyskała niezawodne i odporne na awarie medium transmisyjne. Dopiero synergia elektroniki systemów wbudowanych, administracji sieciami oraz konteneryzacji pozwoliła na stworzenie w pełni autonomicznego systemu IoT.