Akademia Techniczno-Informatyczna w Naukach Stosowanych we Wrocławiu

Skrypt do zajęć Projekt specjalnościowy platformy IoT opartej o Arduino (Projekt 2)

Paweł Dobrowolski

Wrocław 2024

Skrypt przygotowany w ramach projektu „Kształcenie przyszłości: internet rzeczy w zrównoważonej automatyce i robotyce" współfinansowanego ze środków Unii Europejskiej w ramach Programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027, Priorytet 1 Umiejętności, Działanie 01.05 Umiejętności w szkolnictwie wyższym, Typ projektu Dostosowanie oferty podmiotów systemu szkolnictwa wyższego do potrzeb rozwoju gospodarki oraz zielonej i cyfrowej transformacji.

Spis treści

1. Wstęp

Realizacja projektów z wykorzystaniem platformy Arduino pozwala na ich szybkie uruchomienie oraz testowanie różnych wariantów opracowanych rozwiązań. Dzięki dużej ilości bezpłatnie dostępnych bibliotek oraz licznych modułów elektronicznych kompatybilnych z Arduino zaprogramowanie urządzenia nie powinno stanowić większego problemu nawet dla niedoświadczonego użytkownika. Uniwersalność środowiska pozwala na przenoszenie oprogramowania pomiędzy różnymi platformami (STM, AVR, ESP32) bez żadnych jego zmian. Nie jest też wymagana znajomość architektury mikrokontrolera, na który implementowane jest oprogramowanie. Takie uproszczenie pozwala za pomocą kilku komend np. odczytać dane z czujnika lub uruchomić transmisję Wi-Fi i przesłać dane do innego urządzenia.

Popularność środowiska Arduino spowodowała, że stało się one zauważalne przez producentów oprogramowania. Wraz ze wzrostem popularności idei Internetu rzeczy dostawcy zaczęli udostępniać platformy do gromadzenia danych, tzw. chmury. Aby uprościć do nich dostęp opracowano i udostępniono skrypty / biblioteki umożliwiające korzystanie z nich przez użytkowników Arduino. Dzięki temu możliwe jest opracowanie aplikacji sterującej (np. na smartphone) bez znajomości języka JAVA.

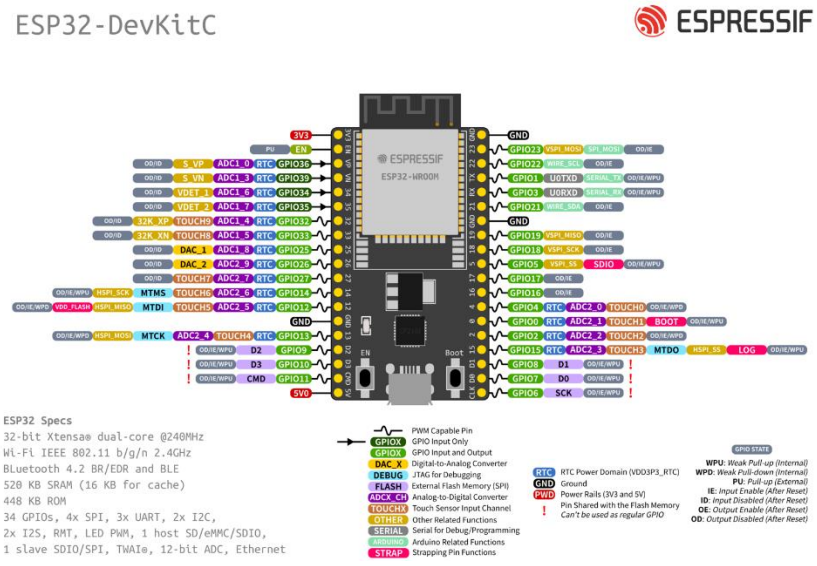
W niniejszej pracy zaprezentowano trzy projekty, które zbudowano w oparciu o platformę Arduino. Pierwszym z nich jest projekt stacji pogodowej, który do wizualizacji danych wykorzystuje narzędzie ThingSpeak firmy MathWorks. Drugi projekt – sterowanie ogrzewaniem i oświetleniem w pomieszczeniu wykorzystuje aplikację mobilną Blynk w wersji darmowej. Trzeci projekt – system monitorujący jakość powietrza korzysta z protokołu MQTT oraz oprogramowania Node-RED uruchomionego na lokalnym brokerze tj. Raspberry Pi 4B. Wszystkie z omówionych projektów korzystają z programowalnego układu ESP32-WROOM-32E z wbudowanym modułem Wi-Fi, który zapewnia programową obsługę wszystkich czujników i modułów wykonawczych oraz transmisję danych do chmury.

2. Platforma Arduino

Platforma Arduino pozwala na szybkie połączenie, uruchomienie i testowanie prototypów urządzeń elektronicznych. Dzięki uniwersalnemu językowi, który oparto na funkcjach i makrach możliwe jest szybkie uruchamianie modułów (np. sensorów) na dowolnych mikrokontrolerach. Duża ilość platform kompatybilnych ze środowiskiem Arduino IDE sprawia, że jest to bardzo popularne środowisko.

2.1. Arduino – płytki z mikrokontrolerem ESP

Na płytkach kompatybilnych z Arduino najczęściej znajdują się mikrokontrolery STM lub AVR. Wraz z rozwojem idei Internetu rzeczy dołączyły również moduły ESP32. Zawierają one programowalny układ (mikrokontroler) wraz z modułem Wi-Fi oraz zintegrowaną anteną (dostępne są również wersje ze złączem antenowym). Dzięki dużej ilości wyprowadzeń (GPIO) oraz różnorodności interfejsów komunikacyjnych moduł ESP32 jest bardzo dobrym wyborem dla urządzeń IoT.



Rysunek 1 Opis wyprowadzeń modułu ESP32-DevKitC V4 [8]

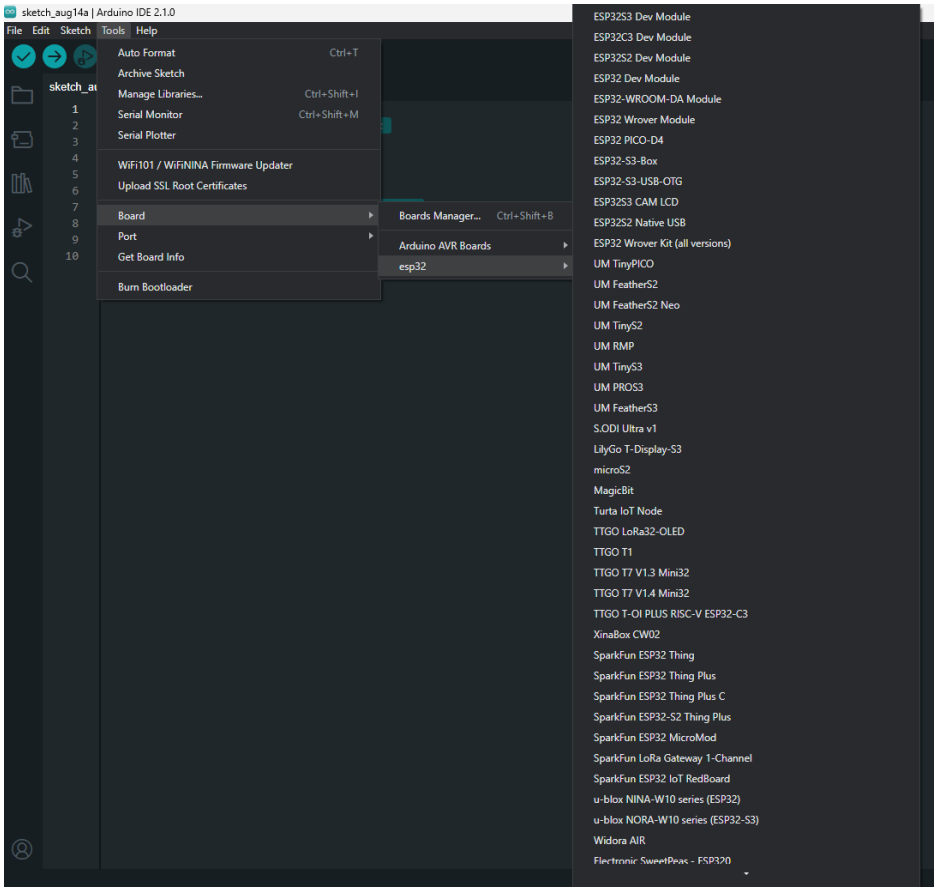
2.2. Arduino IDE

Arduino IDE to środowisko, w którym implementujemy oprogramowanie, kompilujemy je oraz wgrywamy do mikrokontrolera. Posiada ono wbudowany monitor portu szeregowego,

za którego pomocą możemy przesyłać dane z mikrokontrolera do komputera. Jest to szczególnie przydatne np. podczas weryfikacji poprawności działania napisanych programów.

Przed rozpoczęciem tworzenia oprogramowania należy wybrać platformę, na której będzie prowadzony projekt. W na pasku wyboru opcji w zakładce *Tools* → *Board* należy wskazać właściwą pozycję. W przypadku braku płytki należy wykorzystać *Boards Manager* w celu doinstalowania brakujących pakietów. W projektach opisanych w niniejszym dokumencie wykorzystano szablon płytki opisany jako *DOIT ESP32 DEVKIT V1.*

Aby móc w pełni korzystać z dostępnych bibliotek należy je doinstalować. Po otwarciu *Manage Libraries* w zakładce *Tools* pojawia się okno *Library Manager*. Za jago pomocą można znaleźć biblioteki do zastosowanych modułów elektronicznych.



Rysunek 2 Manager płytek Arduino

Po wyborze płytki w edytorze pojawia się poniższy kod, który stanowi zalążek programu.

```
void setup() {
  // put your setup code here, to run once:
}
void loop() {
  // put your main code here, to run repeatedly:
}
```

W funkcji *setup()* wpisuje się komendy, które wywołują się jednokrotnie przy starcie programu. W funkcji *loop()* wpisuje się kod programu, który wykonywany jest w nieskończonej pętli. Odpowiednikiem funkcji *loop()* w języku C / C++ jest *while(1){}*.

Aby móc odczytywać / zapisywać wartości cyfrowe / analogowe wykorzystuje się poniższe funkcje:

pinMode(13, OUTPUT) – pin 13 ustawiony w konfiguracji wyjścia,

digitalWrite(13, LOW) – pin 13 ustawiony na stan niski (parametr HIGH – stan wysoki) ,

pinMode(0, INPUT) – pin 0 ustawiony w konfiguracji wejścia,

pinMode(0, INPUT_PULLUP) – pin 0 ustawiony w konfiguracji wejścia z aktywnym wewnętrznym pull-up,

digitalRead(0) – odczyt stanu na wejściu cyfrowym,

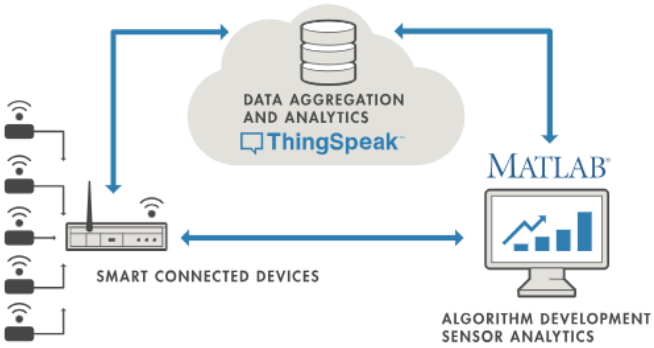
analogRead(A0) – odczyt wartości analogowej na pinie A0 przetwornika ADC,

delay(200) – opóźnienie 200 ms (0.2 sekundy).

Poza wyżej opisanymi funkcjami, Arduino IDE umożliwia programowanie mikrokontrolerów w języku C / C++ bazując na rejestrach programowanego mikrokontrolera. Ten sposób programowania wymaga znajomości architektury oraz rejestrów wewnętrznych zaimplementowanego układu.

3. Projekt stacji pogodowej z wykorzystaniem narzędzia ThingSpeak

Przedstawiony poniżej projekt stacji pogodowej wykorzystuje chmurę danych opracowaną przez MathWorks – ThingSpeak. Umożliwia ona zbieranie oraz analizę danych historycznych. Dzięki możliwości pobierania z niej danych do innych urządzeń możliwe jest sterowanie złożonym obiektem. Łącząc chmurę ThingSpeak z oprogramowaniem Matlab można analizować dane sensoryczne, opracowywać algorytmy sterowania czy dokonywać predykcji na podstawie zgromadzonych już danych.



Rysunek 3 Struktura połączeń ThingSpeak

Poniższy projekt stacji pogodowej zakłada pomiar temperatury, wilgotności oraz ciśnienia. Czas pomiaru (dzień, miesiąc, rok, godzina, minuta, sekunda) pobierany jest z zewnętrznego zegara czasu rzeczywistego (RTC). Układ ten posiada wbudowaną baterię CR2032, która zasila układ, gdy zaniknie zasilanie. Dzięki temu czas jest stale odmierzany bez względu na przerwy w dostawie energii elektrycznej do urządzenia czy jego wyłączeniu. Aby użytkownik mógł odczytać dane ze stacji pogodowej bez konieczności łączenia się z chmurą, zamontowano wyświetlacz, na którym wyświetlane są bieżące informacje o temperaturze, ciśnieniu, wilgotności oraz dane czasowe – data i godzina prezentowanego pomiaru.

Podczas działania stacji pogodowej co 2 sekundy dane przesyłane są do chmury ThingSpeak. Po zalogowaniu się na konto ThingSpeak dane reprezentowane są na wykresach (każdy wykres na osobnym polu). Istnieje również możliwość wyeksportowania wszystkich zebranych danych do pliku CSV.

3.1. Dobór elementów systemu

Właściwy dobór elementów systemu (modułów) oraz znajomość ich kluczowych parametrów znacznie ułatwia konstruowanie urządzeń prototypowych. Poniżej opisano kluczowe moduły elektroniczne wykorzystane do budowy systemu wraz z ich najważniejszymi parametrami. W poniższym zestawieniu nie uwzględniono przewodów połączeniowych czy płytek stykowych.

Moduł mikrokontrolera z Wi-Fi ESP32-DevKitC V4

- Komunikacja bezprzewodowa: Wi-Fi w paśmie 2,4 GHz i standardzie 802.11b/g/n; Bluetooth, BLE 4.2
- Ilość wyprowadzeń GPIO: 34
- Interfejsy komunikacyjne: UART, I2C, SPI, SDIO, PWM, I2S, ADC, DAC
- Taktowanie: do 240MHz
- Pamięć ROM: 448 KB
- Pamięć SRAM: 520 KB
- Pamięć Flash SPI: 4 MB
- Napięcie zasilania: od 3 V do 3,6 V

Moduł czujnika temperatury i wilgotności DHT11

- Interfejs komunikacyjny: Serial Interface (Single-Wire Two-Way)
- Zakres pomiaru temperatury: -0ºC : 55 ºC
- Dokładność pomiaru temperatury: ±2 ºC
- Zakres pomiaru wilgotności: 20% RH : 90 % RH
- Dokładność pomiaru wilgotności: ± 4 RH (dla 25 ºC)
- Rozdzielczość pomiarów: 8 bitów
- Zasilanie: 3.3 – 5.5V DC

Moduł czujnika ciśnienia BMP280

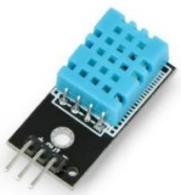
- Interfejs komunikacyjny: I2C
- Zakres pomiaru ciśnienia: 300 hPa : 1100 hPa
- Dokładność pomiaru ciśnienia: ±1 hPa (dla 950 hPa : 1050 hPa)
- Zasilanie: 1.8V – 3.6V DC
- Wyjście danych: zgodnie ze standardem I2C
- Adres: 0x77

Moduł zegara czasu rzeczywistego DS3231

- Interfejs komunikacyjny: I2C
- Odczyt czasu: godziny, minuty, sekundy
- Odczyt daty: miesiąc, dzień, rok
- Bateria podtrzymująca: CR2032
- Zasilanie: 3.3V – 5.0V DC
- Wyjście danych: zgodnie ze standardem I2C
- Adres: 0x57

Moduł wyświetlacza OLED SH1106

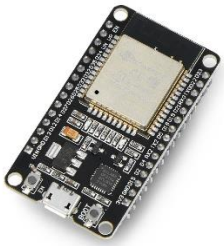
- Interfejs komunikacyjny: I2C
- Przekątna ekranu: 1.3”
- Rozdzielczość: 128 x 64 px
- Zasilanie: 3.3V – 5.0V DC
- Wyjście danych: zgodnie ze standardem I2C
- Adres: 0x3c



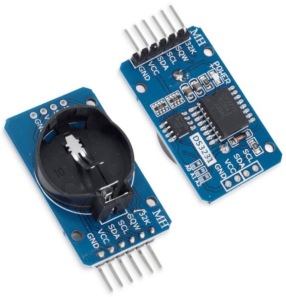
Rysunek 4 Moduł czujnika DHT11

Rysunek 5 Moduł czujnika ciśnienia BMP280

Rysunek 6 Moduł wyświetlacza OLED



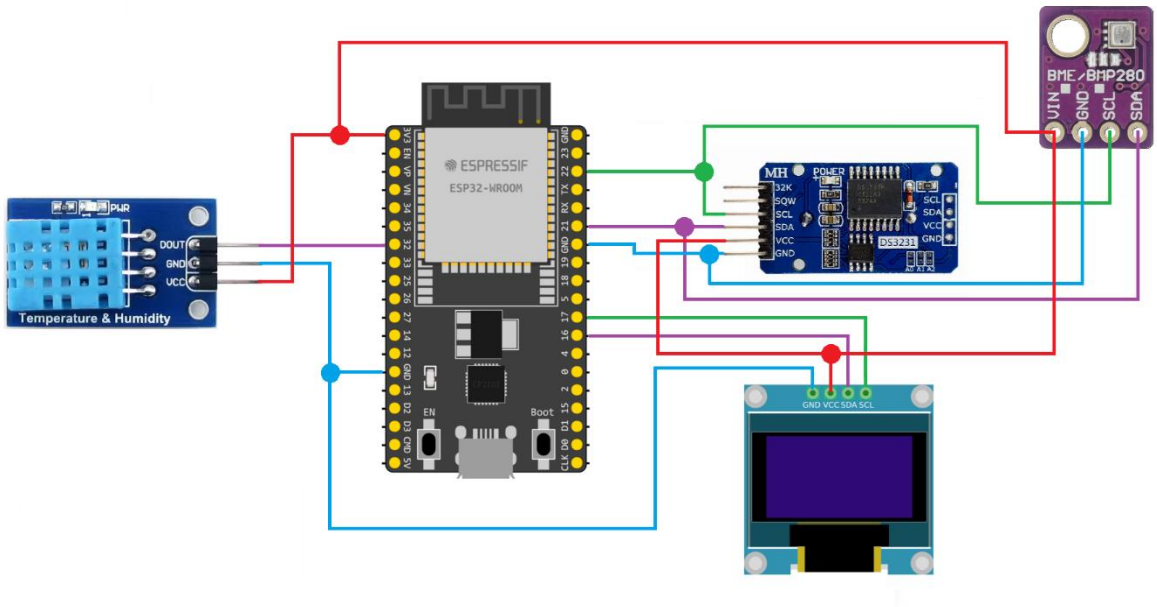
Rysunek 7 Moduł ESP32-DevKitC V4



Rysunek 8 Moduł zegara RTC DS3231

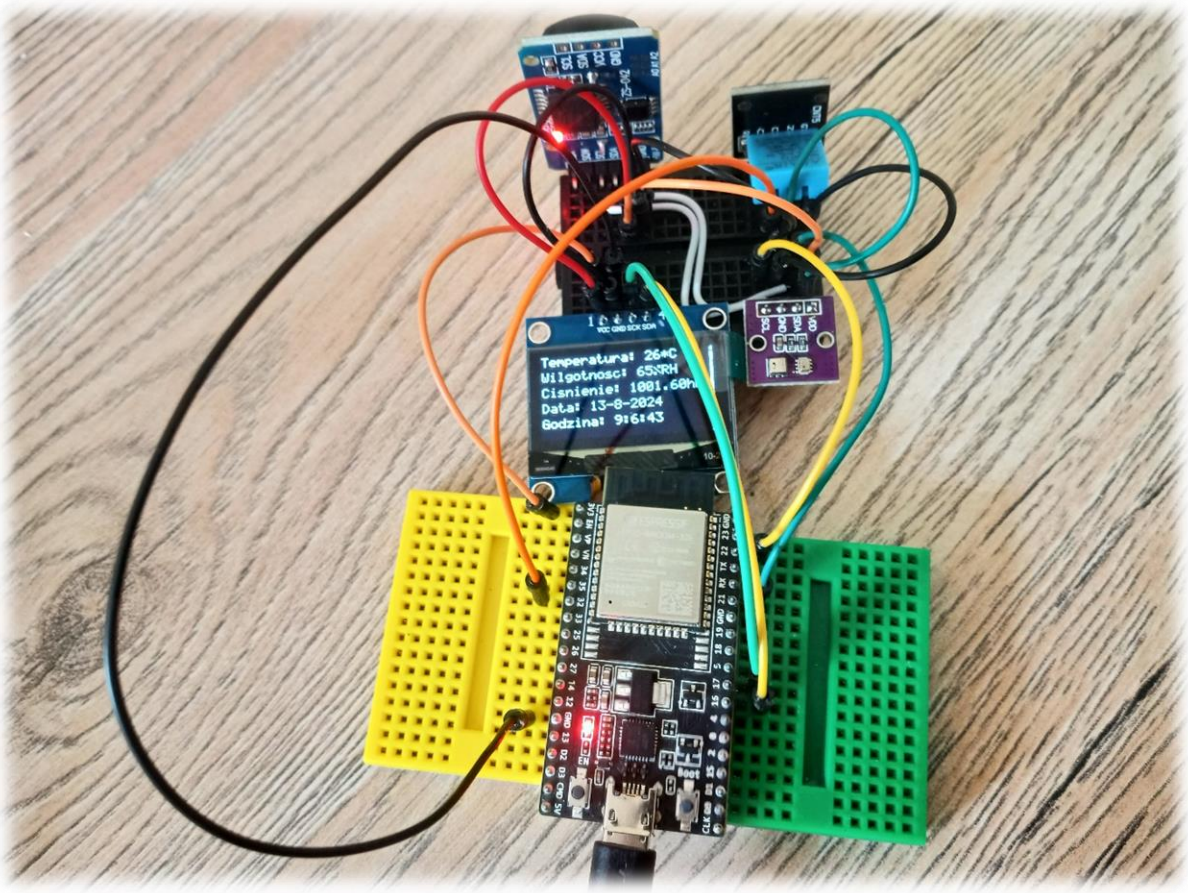
3.2.Schemat ideowy

Na szkicu poniżej zaprezentowano schemat połączeń modułów elektronicznych z modułem mikrokontrolera ESP. Poniższe połączenie pozwala na uruchomienie i poprawne działanie programu z punktu *3.4 Opis programu.* Wszystkie moduły zasilane są napięciem 3.3V (pin 3V3). Czujnik DHT11 podłączony jest do pinu 32, wyświetlacz OLED do pinów 16 i 17, natomiast zegar RTC oraz czujnik ciśnienia do pinów 21 i 22 (I2C hardware).



Rysunek 9 Schemat ideowy połączeń

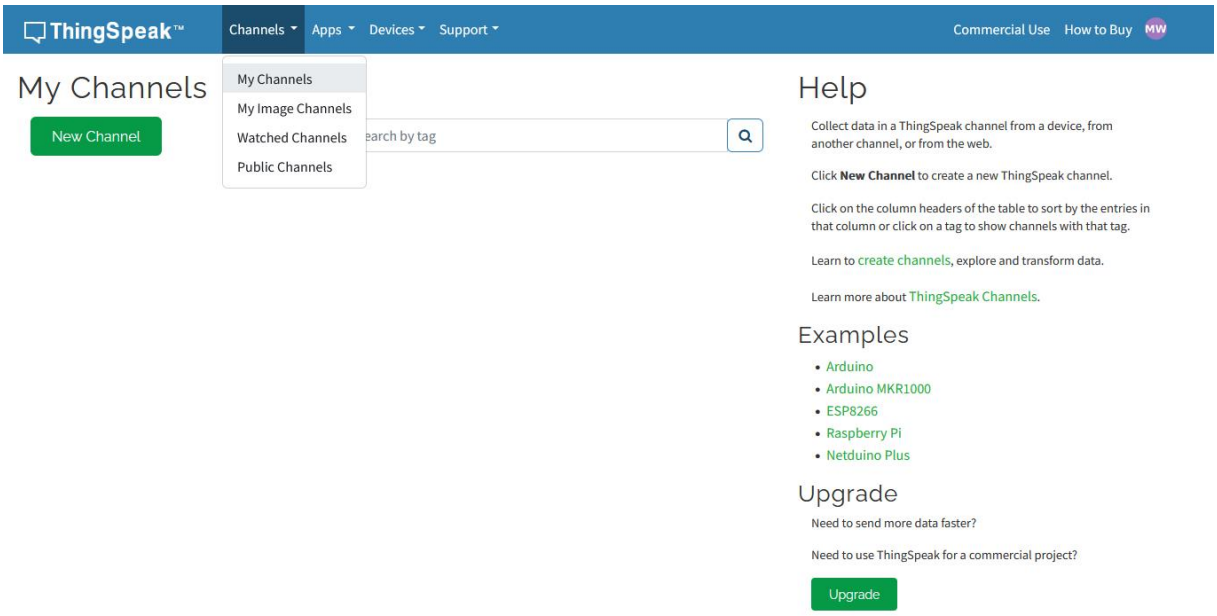
Po wykonaniu połączeń z wykorzystaniem przewodów oraz płytek stykowych powyższy układ może wyglądać jak na zdjęciu poniżej.



Rysunek 10 Przykład zmontowanej stacji pogodowej z wykorzystaniem płytek stykowych

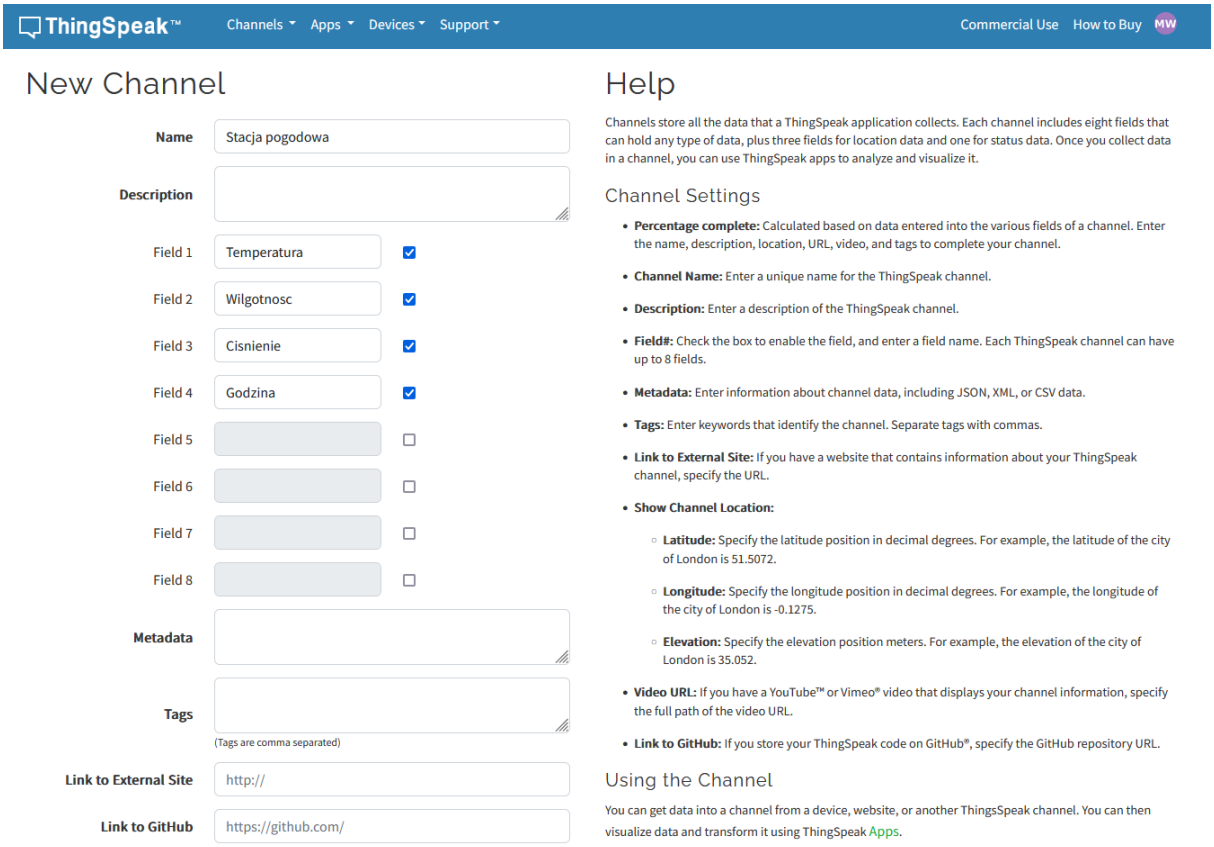
3.3. Konfiguracja aplikacji ThingSpeak

Po utworzeniu konta i zalogowaniu się do aplikacji ThingSpeak (https://thingspeak.com/) pojawia się okno, jak na grafice poniżej. Pierwszą czynność, którą należy wykonać to z zakładki *Channels* → *My Channels* wybrać opcję *New Channel* (zielony przycisk).



Rysunek 11 Stworzenie kanału transmisji danych ThingSpeak

Pojawi się wówczas okno tworzenia nowego kanału transmisji danych do chmury ThingSpeak. Należy wpisać nazwę kanału oraz zmienne, które będą przesyłane do chmury. W prezentowanej wersji (bezpłatnej) istnieje możliwość przesyłania 8 zmiennych do chmury (Field 1 – Field 8). Aby aktywować kolejne pola (Field) należy zaznaczyć checkbox znajdujący się po prawej stronie i wpisać własną nazwę. Nazwa ta stanowi identyfikator dla określonego typu danych (np. temperatury).



Rysunek 12 Tworzenie nowego kanału transmisji danych

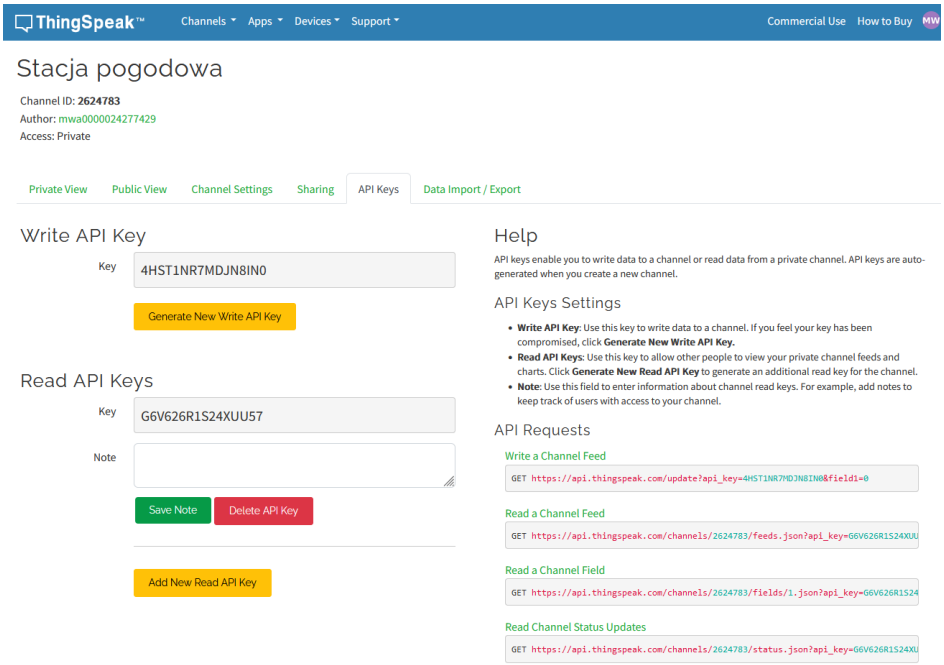
Po utworzeniu kanału należy go otworzyć i przejść do zakładki *API Keys*, w której określone zostały parametry oraz klucze połączenia:

ID kanału (channelID): 2624783

Klucz odczytu (readAPIKey): G6V626R1S24XUU57

Klucz zapisu (writeAPIKey): 4HST1NR7MDJN8IN0

Są one niezbędne do nawiązania połączenia oraz transmisji danych przez klienta tj. ESP32.



Rysunek 13 API Keys

Graficzną prezentację przesłanych danych z wykorzystaniem ThingSpeak oraz wbudowanych szablonów dla wykresów przedstawiono na rysunku poniżej.

Rysunek 14 Graficzna prezentacja przesłanych danych w ThingSpeak

3.4.Opis programu

Wymagane biblioteki:

```
#include <DHT11.h>              // biblioteka czujnika temperatury i wilgotności
#include <I2C_RTC.h>            // biblioteka zegara czasu rzeczywistego
#include <ThingSpeak.h>         // biblioteka chmury ThingSpeak
#include <WiFi.h>               // biblioteka połączenia Wi-Fi
#include <WiFiClient.h>
#include <oled.h>               // biblioteka wyświetlacza OLED
#include <Adafruit_BMP280.h>    // biblioteka czujnika ciśnienia
```

```
// UWAGA - piny wybrane do SDA i SCK muszą być inne niż magistrali I2C pozostałych urządzeń
// biblioteka OLED jest zaimplementowana w taki sposób, że emuluje I2C za pomocą GPIO
// (nie wykorzystuje I2C hardware'owego)
```

Definicja obiektów display (wyświetlacz OLED) oraz bmp (czujnik cośnienia):

```
OLED display(16,17,NO_RESET_PIN,OLED::W_128,OLED::H_64,OLED::CTRL_SH1106);
// 16 – pin danych;       17 – pin zegara;          NO_RESET_PIN – sterownik bez pinu reset
// OLED::W_128 – szerokość OLED (ilość pikseli);
// OLED::H_64 – wysokość OLED (ilość pikseli);
// OLED::CTRL_SH1106 – model sterownika wyświetlacza OLED
Adafruit_BMP280 bmp;
```

Dane połączenia sieciowego po Wi-Fi:

```
char ssid[] = "Moja_siec_WiFi";
char pass[] = "Moje_WiFi123";
```

Dane połączenia (dane z ThingSpeak):

```
char thingSpeakAddress[] = "api.thingspeak.com";
unsigned long channelID = 2624783;
char* readAPIKey = "G6V626R1S24XUU57";
char* writeAPIKey = "4HST1NR7MDJN8IN0";
```

Definicja obiektów RTC (zegara czasu rzeczywistego), dht11 (czujnika temperatury i wilgotności) oraz client (połączenia Wi-Fi):

```
static DS3231 RTC;
DHT11 dht11(32);
WiFiClient client;
```

Inicjalizacja OLED, RTC, BMP280:

```
void setup()
{
    Serial.begin(9600);
    display.begin();
    RTC.begin();
```

```
bmp.begin();                    // wykorzystuje magistralę I2C zainicjalizowaną przez RTC
```

Ustawienie zegara RCT przykładową datą: 13.08.2024 i godziną 09:03:59:

```
RTC.setHourMode(CLOCK_H12);          // tryb 12-godzinny AM / PM

RTC.setDay(13);
RTC.setMonth(8);
RTC.setYear(2024);

RTC.setHours(9);
RTC.setMinutes(3);
RTC.setSeconds(56);
RTC.setWeek(3);
```

Inicjalizacja Wi-Fi:

```
WiFi.begin(ssid, pass);
int wifi_ctr = 0;
while (WiFi.status() != WL_CONNECTED)
{
    delay(500);
    Serial.print(".");
}
```

Inicjalizacja ThingSpeak:

```
ThingSpeak.begin( client );
Serial.println("WiFi connected");
}

void loop()
{
```

Zmienne zapisujące wartość temperatury oraz wilgotności:

```
int temperature = 0;
int humidity = 0;
```

Odczyt temperatury, wilgotności oraz ciśnienia:

```
int result = dht11.readTemperatureHumidity(temperature, humidity);
int pressure = bmp.readPressure();
```

Odczyt daty i godziny z zegara RTC:

```
uint hour = RTC.getHours();
uint minute = RTC.getMinutes();
uint second = RTC.getSeconds();
uint day = RTC.getDay();
uint month = RTC.getMonth();
uint year = RTC.getYear();
```

Zapis danych do chmury:

```
ThingSpeak.setField( 1, temperature );
ThingSpeak.setField( 2, humidity );
ThingSpeak.setField( 3, pressure );
```

Zapis godziny i daty do chmury w formacie:

dla godziny: 09:10:23 tj. 091023 dla daty: 13.08.2024 tj. 13082024

```
ThingSpeak.setField( 4, (long int) (hour * 10000 + minute * 100 + second));
ThingSpeak.setField( 5, (long int) (day * 1000000 + month * 10000 + year));

int writeSuccess = ThingSpeak.writeFields( channelID, writeAPIKey );
```

Wyświetlanie danych o temperaturze, wilgotności, ciśnieniu, godzinie oraz dacie na wyświetlaczu OLED:

```
display_data(temperature, humidity, pressure, hour , minute, second, day, month, year);
```

Odświeżanie danych co 2000ms (2 sekundy) na wyświetlaczu OLED:

```
delay(2000);
}
```

Funkcja wyświetlająca dane o temperaturze, wilgotności, ciśnieniu, godzinie oraz dacie na wyświetlaczu OLED:

```
void display_data(int temperature, int humidity, int pressure, uint hours, uint minutes, uint seconds,
uint day, uint month, uint year)
{
```

Przygotowanie danych do wyświetlania:

```
String t_string = "Temperatura: " + String(temperature) + "*C";
String h_string = "Wilgotnosc: " + String(humidity) + "%RH";
String p_string = "Cisnienie: " + String((double) pressure / 100) + "hPa";
String d_string = "Data: " + String(day) + "-" + String(month) + "-" + String(year);
String c_string = "Godzina: " + String(hours) + ":" + String(minutes) + ":" + String(seconds);
```

Zapisanie danych do wyświetlania w pamięci wyświetlacza:

```
display.clear();
display.draw_string(2,0, t_string.c_str());
display.draw_string(2,12, h_string.c_str());
display.draw_string(2,24, p_string.c_str());
display.draw_string(2,36, d_string.c_str());
display.draw_string(2,48, c_string.c_str());
```

Wyświetlenie danych:

```
display.display();
}
```

4. Projekt sterowania oświetleniem oraz ogrzewaniem przez aplikację Blynk

Przedstawiony poniżej projekt sterowania oświetleniem oraz ogrzewaniem przez aplikację Blynk umożliwia sterowanie oświetleniem oraz ogrzewaniem dla jednego pokoju. Dzięki ciągłemu pomiarowi temperatury użytkownik zna jej aktualną wartość. Z poziomu aplikacji istnieje możliwość ustawienia temperatury zadanej w pomieszczeniu i w przypadku zbyt niskiej temperatury uruchomi się ogrzewanie (aktywuje się przekaźnik załączający np. grzejnik elektryczny). Użytkownik ma również możliwość sterowania oświetleniem na zasadzie włącz / wyłącz oraz sterowania jego jasnością. Umożliwia to sygnał PWM, który jest podłączony do diody LED symbolizującej oświetlenie w pokoju. W praktycznym zastosowaniu diodę LED należałoby zamienić na układ optotriaka z głównym triakiem załączającym (optotriak bez układu ZCD (zero-cross-detect)). Taki układ pozwoliłby na sterownie jasnością oświetlenia ze standardowych żarówek jak i energooszczędnych LEDowych.

Zakres funkcjonalności wyżej opisanego projektu dostosowano do ograniczeń bezpłatnej wersji aplikacji Blynk, która umożliwia przesyłanie do 5 zmiennych. Dane przesyłane są z / do chmury za pośrednictwem modułu Wi-Fi zintegrowanego z programowalnym modułem ESP32 – WROOM – 32E.

Blynk jest pakietem oprogramowania, który umożliwia prototypowanie i zdalne zarządzanie podłączonymi urządzeniami elektronicznymi. Pozwala na analizowanie danych w czasie rzeczywistym i danych historycznych, a także na zdalne sterowanie urządzeniami z dowolnego miejsca na świecie, gdzie zapewniony jest dostęp do internetu. Występuje w formie aplikacji przeglądarkowej, w której konfiguruje się parametry połączenia, a także można stworzyć webowy panel sterowania. Korzystając z aplikacji mobilnej zainstalowanej na Android lub iOS można stworzyć prosty interfejs, który umożliwi podgląd paramentów i sterowanie urządzeniem.

4.1. Dobór elementów systemu

Właściwy dobór elementów systemu (modułów) oraz znajomość ich kluczowych parametrów znacznie ułatwia konstruowanie urządzeń prototypowych. Poniżej opisano kluczowe moduły elektroniczne wykorzystane do budowy systemu wraz z ich najważniejszymi

parametrami. W poniższym zestawieniu nie uwzględniono przewodów połączeniowych czy płytek stykowych.

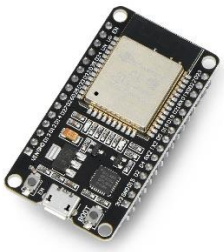
Moduł mikrokontrolera z Wi-Fi ESP32-DevKitC V4

- Komunikacja bezprzewodowa: Wi-Fi w paśmie 2,4 GHz i standardzie 802.11b/g/n; Bluetooth, BLE 4.2
- Ilość wyprowadzeń GPIO: 34
- Interfejsy komunikacyjne: UART, I2C, SPI, SDIO, PWM, I2S, ADC, DAC
- Taktowanie: do 240MHz
- Pamięć ROM: 448 KB
- Pamięć SRAM: 520 KB
- Pamięć Flash SPI: 4 MB
- Napięcie zasilania: od 3 V do 3,6 V

Czujnik temperatury DS18B20

- Interfejs komunikacyjny: 1-wire
- Zakres pomiaru temperatury: -55ºC : 125 ºC
- Dokładność pomiaru: ±0.5 ºC
- Rozdzielczość pomiaru: 9 – 12 bitów
- Zasilanie: 3.3 – 5V DC
- Obudowa: TO-92
- Wyjście danych: pin DQ typu open-collector

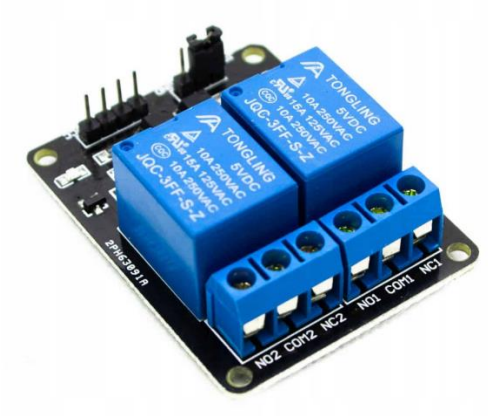
Rysunek 15 Czujnik temperatury DS18B20



Rysunek 16 Moduł ESP32-DevKitC V4

Moduł dwuprzekaźnikowy

- Napięcie zasilania cewki przekaźnika: 5V
- Aktywacja przekaźnika: stan niski (LOW)
- Maksymalne obciążenie styków: 10A, 250 VAC (obciążenie rezystancyjne)



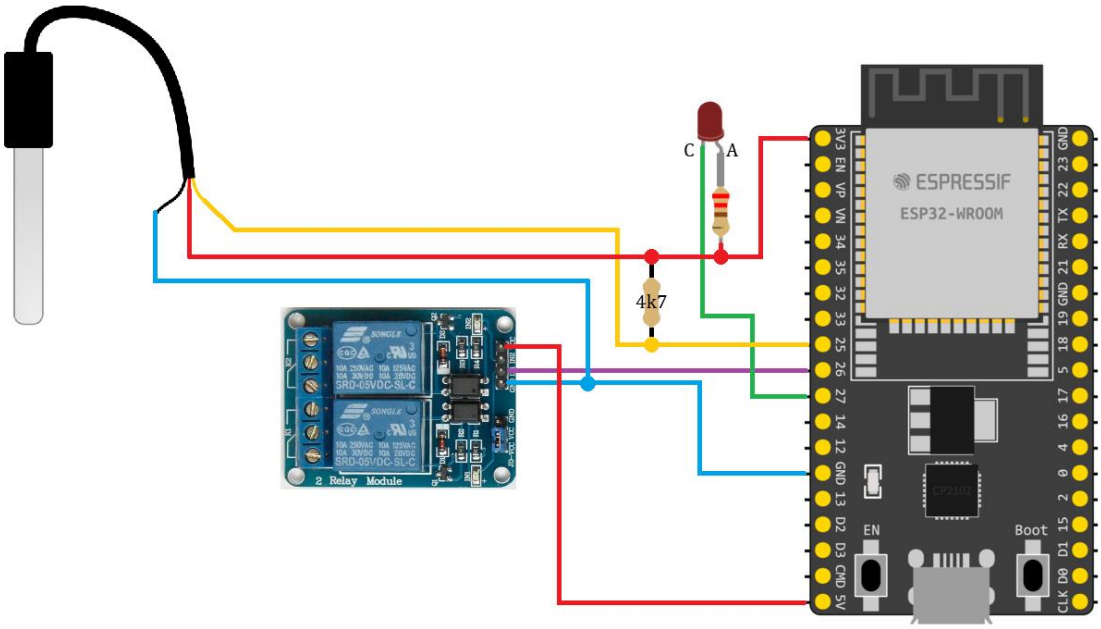
Rysunek 17 Moduł dwuprzekaźnikowy

Dioda LED z rezystancyjnym ograniczeniem prądowym

- Kolor: zielony
- Prąd: 11 mA
- Rezystancja szeregowa: 100 Ω

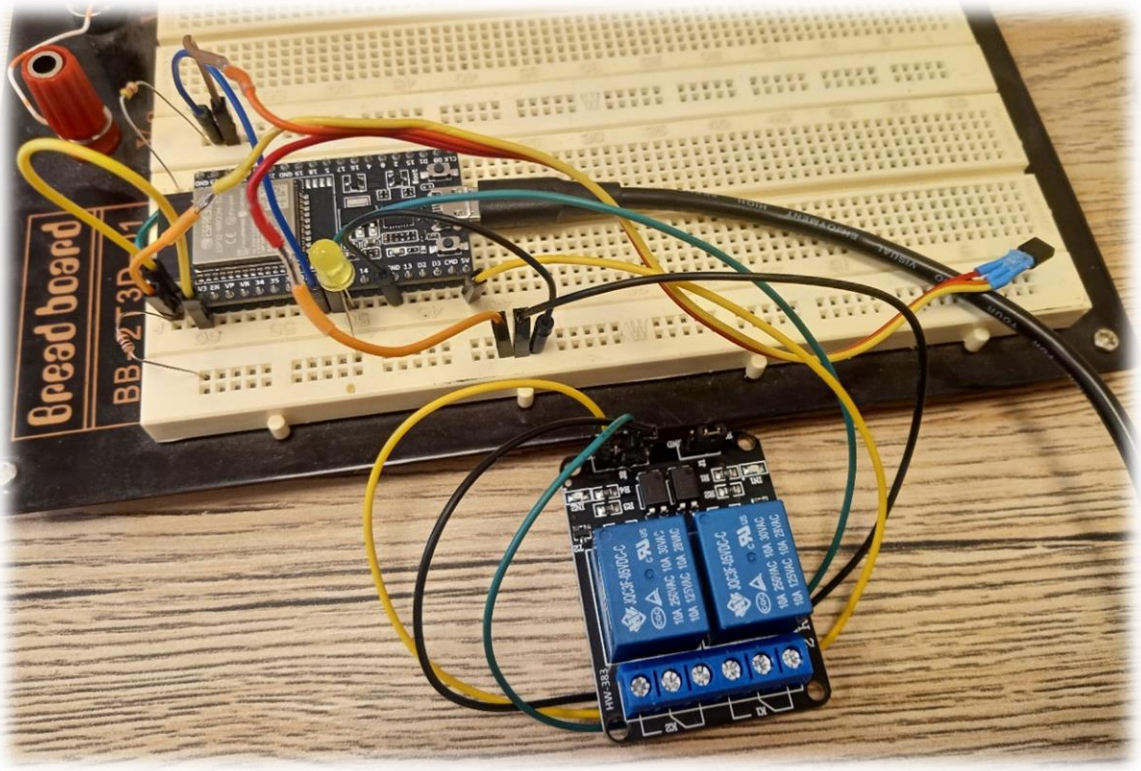
4.2.Schemat ideowy

Na szkicu poniżej zaprezentowano schemat połączeń modułów elektronicznych z modułem mikrokontrolera ESP. Poniższe połączenie pozwala na uruchomienie i poprawne działanie programu z punktu *4.4 Opis programu.* Wszystkie moduły zasilane są napięciem 3.3V (pin 3V3). Czujnik DS18B20 podłączony jest do pinu 25, dioda LED do pinu 27, natomiast moduł przekaźnikowy (wejście IN1) do pinu 26.



Rysunek 18 Schemat ideowy połączeń

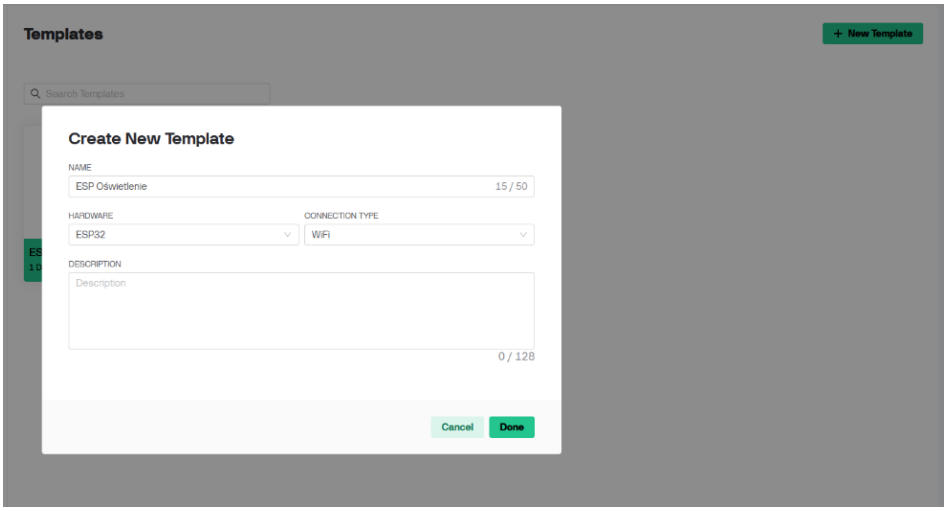
Po wykonaniu połączeń z wykorzystaniem przewodów oraz płytek stykowych powyższy układ może wyglądać jak na zdjęciu poniżej.



Rysunek 19 Przykład zmontowanego systemu sterowania oświetleniem oraz ogrzewaniem z wykorzystaniem płytek stykowych

4.3.Konfiguracja aplikacji Blynk

Po utworzeniu konta na stronie *https://blynk.io/* i zalogowaniu się należy stworzyć nowy szablon, który będzie ściśle powiązany z urządzeniem, z którego chcemy zbierać dane. Na etapie tworzenia szablonu należy wskazać platformę, która będzie łączyła się z chmurą (w przypadku omawianego projektu będzie to ESP) oraz sposób połączenia (Wi-Fi). Nadajemy również nazwę oraz opcjonalnie opis.

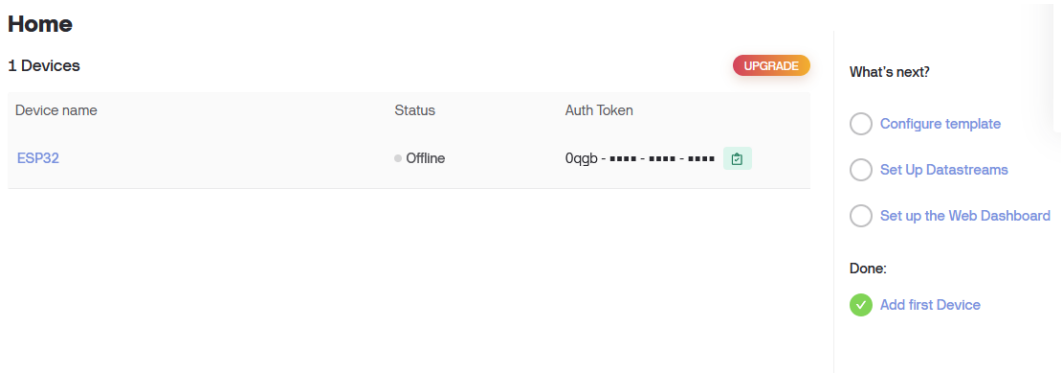


Rysunek 20 Blynk - tworzenie szablonu

Po utworzeniu szablonu w zakładce *Home* należy dodać urządzenie – opcja *Add first device*. Istotnym jest również, aby zapisać dane, które umożliwią połączenie z chmurą. Definicje *BLYNK_TEMPLET_ID* oraz *BLYNK_TEMPLATE_NAME* są wymagane do nawiązania połączenia.

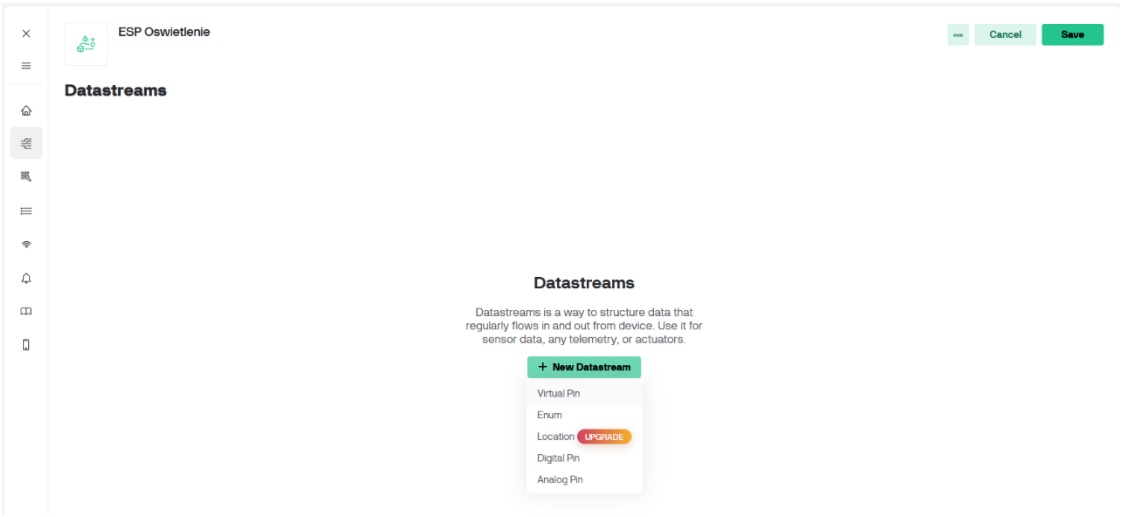
Rysunek 21 Blynk - konfiguracja urządzenia

Po dodaniu urządzenia w zakładce *Home* pojawi się zdefiniowane urządzenie, jego status oraz token autoryzacyjny. Token należy skopiować, gdyż jest on wymagany do autoryzacji połączenia urządzenia z chmurą. Jest on niepowtarzalny i ściśle związany ze skonfigurowanym urządzeniem. W sytuacji, gdy urządzenie nawiąże połączenie z chmurą status ulegnie zmianie – zmieni się na *Online*. ***Informacja*** - wersja bezpłatna oprogramowania umożliwia zdefiniowanie maksymalnie trzech urządzeń.



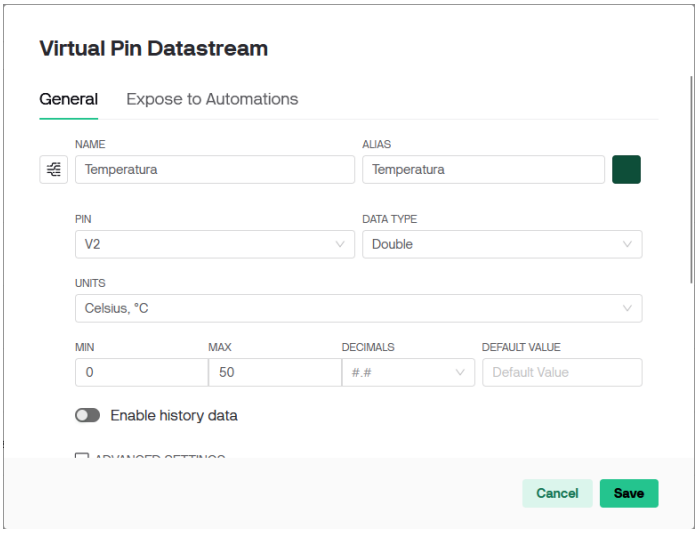
Rysunek 22 Blynk - zdefiniowane urządzenie ESP

Kolejnym etapem jest stworzenie *Datastreams* czyli strumieni danych. Jako strumień danych należy postrzegać każdą zmienną, jaką przesyłamy do / z chmury. Wybierając *New Datastream* należy z rozwijanej listy wybrać *Virtual Pin*. Taka konfiguracja umożliwia transmisję różnego typu danych (float, int, double, string) pomiędzy urządzeniem a chmurą. W wersji bezpłatnej możliwe jest stworzenie maksymalnie pięciu datasterams.



Rysunek 23 Blynk - konfiguracja datastreams

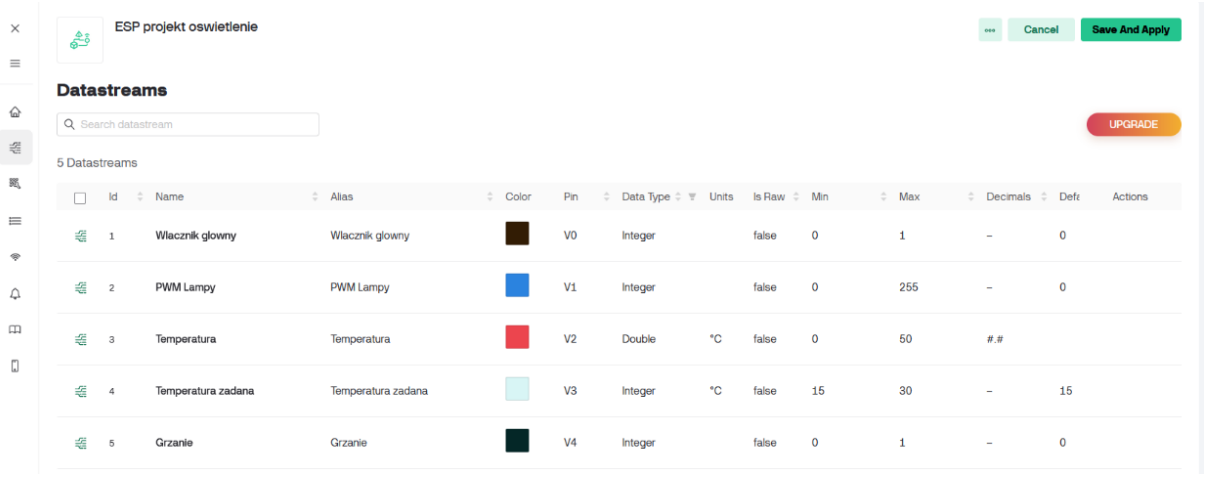
Po utworzeniu nowego datastream'u pojawia się jego okno konfiguracyjne. Poza nazwą należy wybrać typ zmiennej (*Data type*), jednostki, sposób wyświetlania wartości, jej zakres. Istotnym jest, aby właściwie wskazać w polu *PIN* numer wirtualnego pinu, gdyż to on odpowiada za mapowanie zmiennych z urządzenia do chmury.



Rysunek 24 Blynk - konfiguracja wirtualnego pinu VP

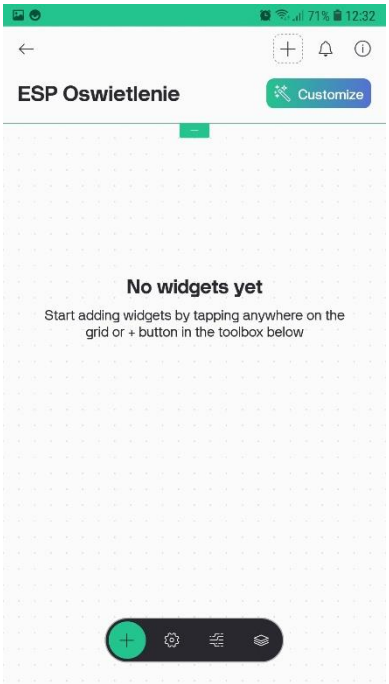
Lista wszystkich datastream'ów w przedstawionym projekcie:

- *Wlacznik glowny* – zmienna odpowiedzialna za włączanie / wyłączanie oświetlenia przyciskiem z poziomu aplikacji
- *PWM Lampy* – zmienna przesyłająca wartość od 0 do 255 co odpowiada wypełnieniu od 0 do 100% 8-bitowego PWM'a
- *Temperatura* – zmienna przesyłająca wartość temperatury w pomieszczeniu (odczyt z czujnika)
- *Temperatura zadana* – wartość temperatury oczekiwanej ustawiana przez użytkownika w aplikacji
- *Grzanie* – zmienna odpowiedzialna za włączanie / wyłączanie ogrzewania np. grzejnika elektrycznego

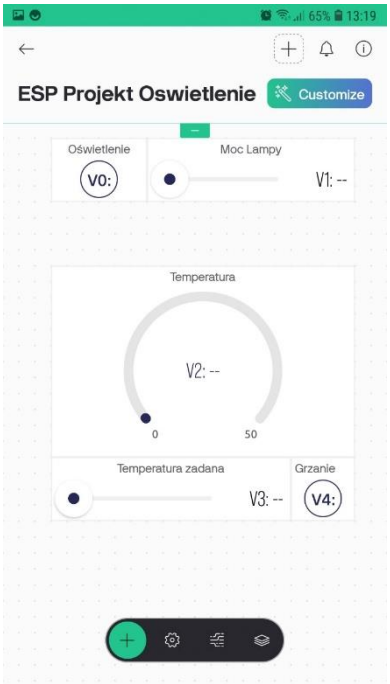


Rysunek 25 Blynk - zdefiniowane datastreams

Po zdefiniowaniu wszystkich datastream'ów można przystąpić do tworzenia aplikacji. Tworzenie aplikacji polega na przeciąganiu w przestrzeń roboczą widgetów, przyłączeniu do nich datastreamów oraz konfigurowaniu ich poprzez np. dodawanie tekstu, zakresów, zdefiniowanie kolorów, itp.



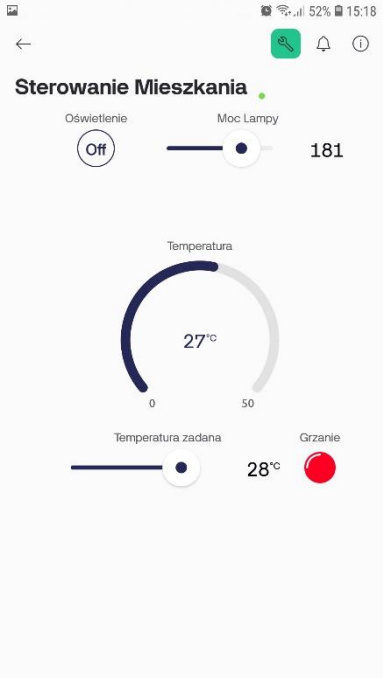
Rysunek 26 Blynk - okno aplikacji



Rysunek 27 Blynk - aplikacja z dodanymi widgetami

Oznaczenia V0 – V4 to piny wirtualne.

Rysunek 28 Blynk - uruchomiona aplikacja



Rysunek 29 Blynk - uruchomiona aplikacja z aktywnym grzaniem

4.4. Opis programu

Inicjalizacja pinów, do których podłączono elementy pomiarowe oraz wykonawcze:

```
#define GPIO27 27          // sterowanie jasnością oświetlenia (PWM)
#define GPIO26 26          // sterowanie włączeniem / wyłączeniem przekaźnika grzejnika
```

Dane połączenia (dane z Blynk):

```
#define BLYNK_PRINT Serial
#define BLYNK_TEMPLATE_ID "TMPL4ytAGpaRU"
#define BLYNK_TEMPLATE_NAME "ESP Oswietlenie"
```

Wymagane biblioteki:

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <DS18B20.h>
```

Token autoryzacji pobrany ze strony konfiguracyjnej Blynk:

```
char auth[] = "gJjI4Lo0loLNrgn0fR2roS4Icm2xw7MS";
```

Dane połączenia sieciowego po Wi-Fi:

```
char ssid[] = "Moja_siec_WiFi";
char pass[] = "Moje_WiFi123";
```

Inicjalizacja czujnika pomiaru temperatury podłączonego do pinu 25:

```
DS18B20 ds(25);
```

Inicjalizacja zmiennych globalnych:

```
int wlacznik_swiatla = 0;
int grzanie = 0;
int temperatura_zadana = 15;
double temperatura = 0.0;
int moc_lampy = 0;
```

Inicjalizacja pinów mikrokontrolera, połączenia Wi-Fi oraz Blynk:

```
void setup()
{
    pinMode(GPIO32, OUTPUT);
    pinMode(GPIO27, OUTPUT);
    pinMode(GPIO26, OUTPUT);

    analogWrite(GPIO27, 255);          // inicjalizacja PWM
    digitalWrite(GPIO26, HIGH);        // ustawienie pinu w stan wysoki

    Serial.begin(115200);              // uruchomienie portu szeregowego do debugowania
    delay(10);
    Serial.print("Connecting to ");
    Serial.println(ssid);
```

Inicjalizacja Wi-Fi:

```
    WiFi.begin(ssid, pass);
    int wifi_ctr = 0;

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.println("WiFi connected");
```

Inicjalizacja Blynk:

```
    Blynk.begin(auth, ssid, pass);
}
```

Funkcja główna programu:

```
void loop()
{
    Blynk.run();                        // wywołuje funkcje BLYNK_WRITE(Vx)
    temperatura = ds.getTempC();        // odczyt temperatury z czujnika
    Blynk.virtualWrite(V2, temperatura); // zapis danych do chmury Blynk – zmienna
                                        odpowiada właściwemu virtual pin
    Blynk.virtualWrite(V4, grzanie);    // zapis danych do chmury Blynk – zmienna
                                        odpowiada właściwemu virtual pin
```

Regulacja temperatury – regulator dwustawny z histerezą 2 ºC:

```
    if(grzanie)
    {
        if(temperatura > temperatura_zadana + 1)
        {
            grzanie = 0;
            digitalWrite(GPIO26, HIGH);
        }
    }
    else if(temperatura < temperatura_zadana - 1)
    {
        grzanie = 1;
        digitalWrite(GPIO26, LOW);
    }
}
```

Odczyt danych z aplikacji Blynk:

```
BLYNK_WRITE(V0)                 // odczyt przycisku włączającego / wyłączającego światło
{                               // funkcja wywoływana przez Blynk.run();
    int value = param.asInt();
    wlacznik_swiatla = value;
    if(wlacznik_swiatla)
        analogWrite(GPIO27, moc_lampy);
    else
        analogWrite(GPIO27, 255);
}

BLYNK_WRITE(V1)                 // odczyt wartości jasności oświetlenia z Blynk
{                               // funkcja wywoływana przez Blynk.run();
    int value = param.asInt();
    moc_lampy = 255 - value;
    if(wlacznik_swiatla)
        analogWrite(GPIO27, moc_lampy);
    else
        analogWrite(GPIO27, 255);
}
```

```
BLYNK_WRITE(V3)                    // odczyt wartości temperatury zadanej z Blynk
{                                  // funkcja wywoływana przez Blynk.run();
      int value = param.asInt();
      temperatura_zadana = value;
}
```

5. Projekt urządzenia mierzącego smog i jakość powietrza

Przedstawiony poniżej projekt urządzenia mierzącego smog i jakość powietrza wykorzystuje protokół MQTT do transmisji danych. Urządzenie pomiarowe zbudowane jest z modułu mikrokontrolera ESP, czujnika pomiaru jakości powietrza SPS30, który umożliwia pomiar cząsteczek PM1, PM2.5, PM4, PM10 oraz czujnika temperatury, wilgotności i ciśnienia atmosferycznego oraz zegara RTC. Dane zebrane od sensorów przesyłane są za pomocą Wi-Fi do brokera MQTT, którego funkcję pełni minikomputer Raspberry Pi 4B.

Zgodnie z ideą protokołu MQTT urządzenie przesyłające dane publikuje wiadomości (dane) ze zdefiniowanym tematem (*Topic*). Broker zapisuje te dane (ostatnio przesłane) i udostępnia je, gdy otrzyma żądanie z określonym tematem. W prezentowanym przykładzie dane z każdego sensora stanowiły odrębny temat, pod którym były publikowane.

Aby móc zwizualizować dane wykorzystano oprogramowanie Node-RED 4.0. Stworzono w nim diagram połączeń i przepływu danych, a otrzymane dane zaprezentowano na wykresach.

5.1. Dobór elementów systemu

Właściwy dobór elementów systemu (modułów) oraz znajomość ich kluczowych parametrów znacznie ułatwia konstruowanie urządzeń prototypowych. Poniżej opisano kluczowe moduły elektroniczne wykorzystane do budowy systemu wraz z ich najważniejszymi parametrami. W poniższym zestawieniu nie uwzględniono przewodów połączeniowych czy płytek stykowych.

Moduł mikrokontrolera z Wi-Fi ESP32-DevKitC V4

- Komunikacja bezprzewodowa: Wi-Fi w paśmie 2,4 GHz i standardzie 802.11b/g/n; Bluetooth, BLE 4.2
- Ilość wyprowadzeń GPIO: 34
- Interfejsy komunikacyjne: UART, I2C, SPI, SDIO, PWM, I2S, ADC, DAC
- Taktowanie: do 240MHz
- Pamięć ROM: 448 KB
- Pamięć SRAM: 520 KB

- Pamięć Flash SPI: 4 MB
- Napięcie zasilania: od 3 V do 3,6 V

Czujnik pomiaru jakości powietrza SPS30

- Interfejs komunikacyjny: I2C (dostępny również UART)
- Pomiar pyłów: PM1, PM2.5, PM4, PM10
- Zakres pomiaru: od 1 do 1000 $\mu g/m^3$
- Dokładność pomiaru: $\pm 10\ \mu g/m^3$ od 0 do 100 $\mu g/m^3$
  ± 10 % od 100 do 1000 $\mu g/m^3$
- Żywotność: powyżej 8 lat przy pracy 24/7
- Zasilanie: 4.5V – 5.5V DC
- Wyjście danych: zgodnie ze standardem I2C / UART
- Adres SPS30: 0x69

Moduł czujnika ciśnienia BMP280 oraz temperatury i wilgotności AHT20

- Interfejs komunikacyjny: I2C
- Zakres pomiaru ciśnienia: 300 hPa : 1100 hPa
- Dokładność pomiaru ciśnienia: ±1 hPa (dla 950 hPa : 1050 hPa)
- Zakres pomiaru temperatury: -40ºC : 80 ºC
- Dokładność pomiaru temperatury: ±0.3 ºC
- Zakres pomiaru wilgotności: 0 RH : 100% RH
- Dokładność pomiaru wilgotności: ±2%
- Zasilanie: 1.8V – 3.6V DC
- Wyjście danych: zgodnie ze standardem I2C
- Adres AHT20: 0x38
- Adres BMP280: 0x77

Minikomputer Raspberry Pi 4

- Komunikacja bezprzewodowa: Wi-Fi w paśmie 2.4 GHz / 5.0 GHz i standardzie 802.11b/g/n/ac; Bluetooth, BLE 5.0
- Ilość wyprowadzeń GPIO: 28
- Interfejsy komunikacyjne: UART, I2C, SPI, PWM
- Taktowanie: 4 x 1.5 GHz
- Pamięć RAM: 2 GB
- Napięcie zasilania: 5.2V / 3A

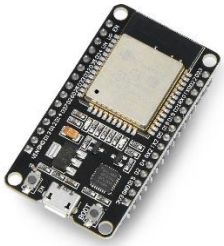
Moduł zegara czasu rzeczywistego DS3231

- Interfejs komunikacyjny: I2C
- Odczyt czasu: godziny, minuty, sekundy
- Odczyt daty: miesiąc, dzień, rok
- Bateria podtrzymująca: CR2032
- Zasilanie: 3.3V – 5.0V DC
- Wyjście danych: zgodnie ze standardem I2C
- Adres: 0x57

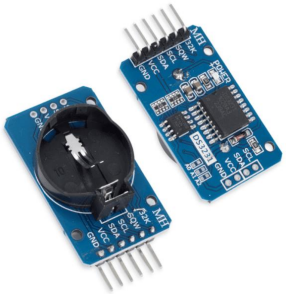
Rysunek 30 Czujnik jakości powietrza SPS30

Rysunek 31 Moduł czujnika BMP280 oraz AHT20

Rysunek 32 Raspberry Pi 4



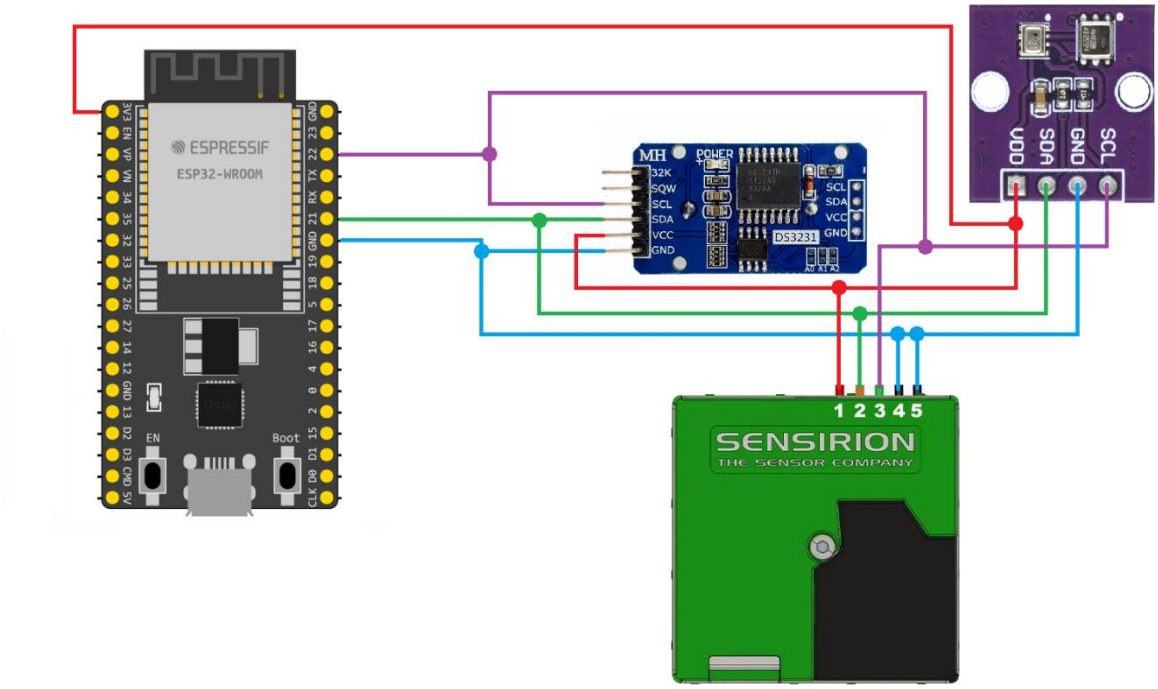
Rysunek 33 Moduł ESP32-DevKitC V4



Rysunek 34 Moduł zegara RTC DS3231

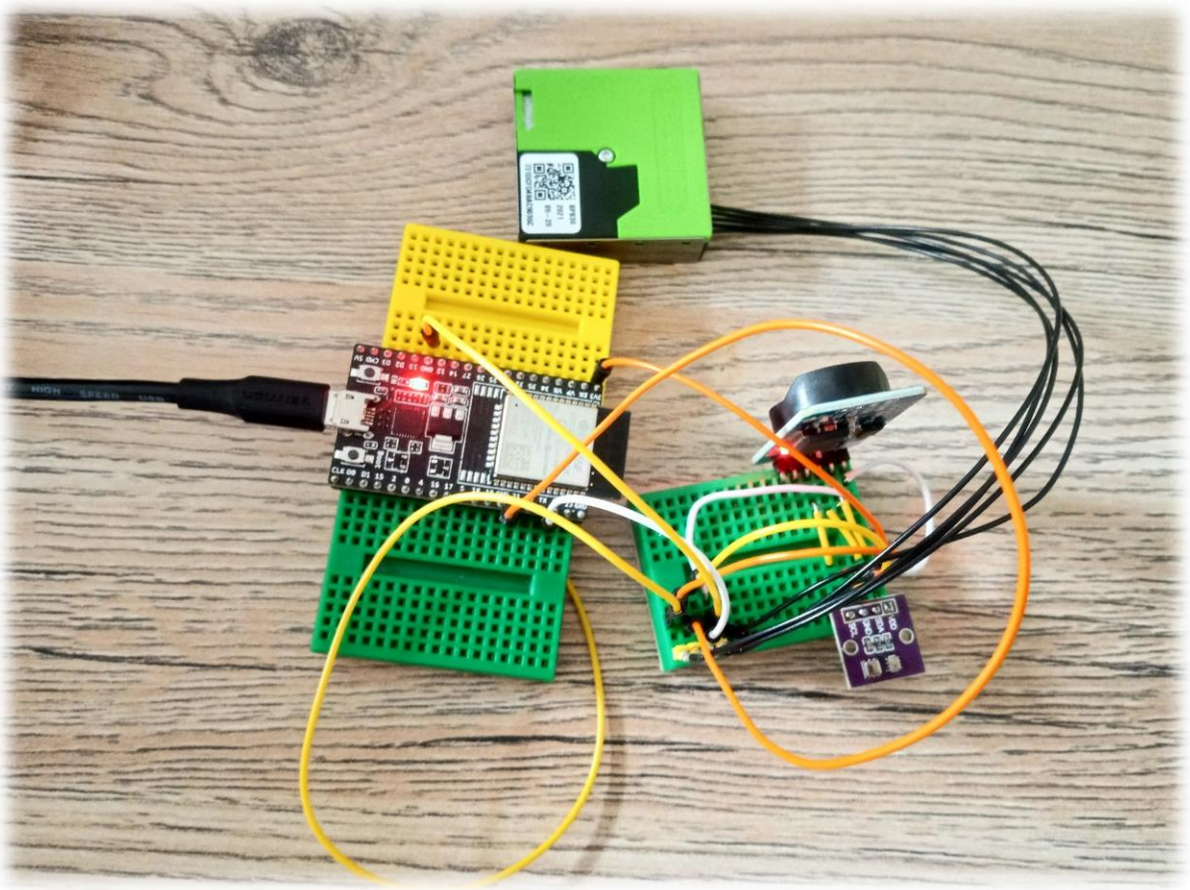
5.2.Schemat ideowy

Na szkicu poniżej zaprezentowano schemat połączeń modułów elektronicznych z modułem mikrokontrolera ESP. Poniższe połączenie pozwala na uruchomienie i poprawne działanie programu z punktu *5.4 Opis programu.* Wszystkie moduły zasilane są napięciem 3.3V (pin 3V3). Czujniki SPS30, BMP280, AHT20, DS3231 podłączone są do wspólnej magistrali I2C – piny 21 oraz 22.



Rysunek 35 Schemat ideowy połączeń

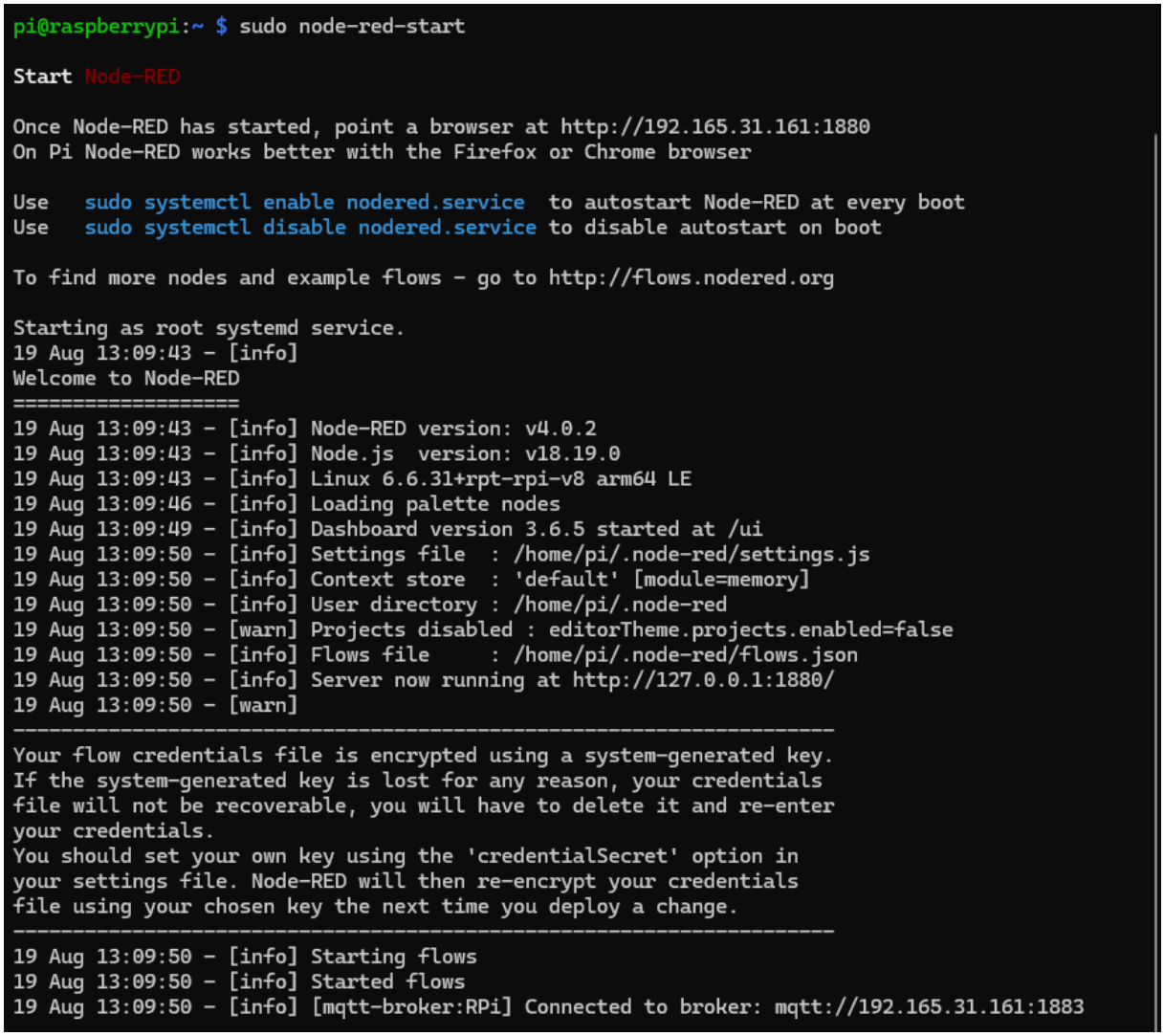
Po wykonaniu połączeń z wykorzystaniem przewodów oraz płytek stykowych powyższy układ może wyglądać jak na zdjęciu poniżej.



Rysunek 36 Przykład zmontowanego urządzenia mierzącego smog i jakość powietrza z wykorzystaniem płytek stykowych

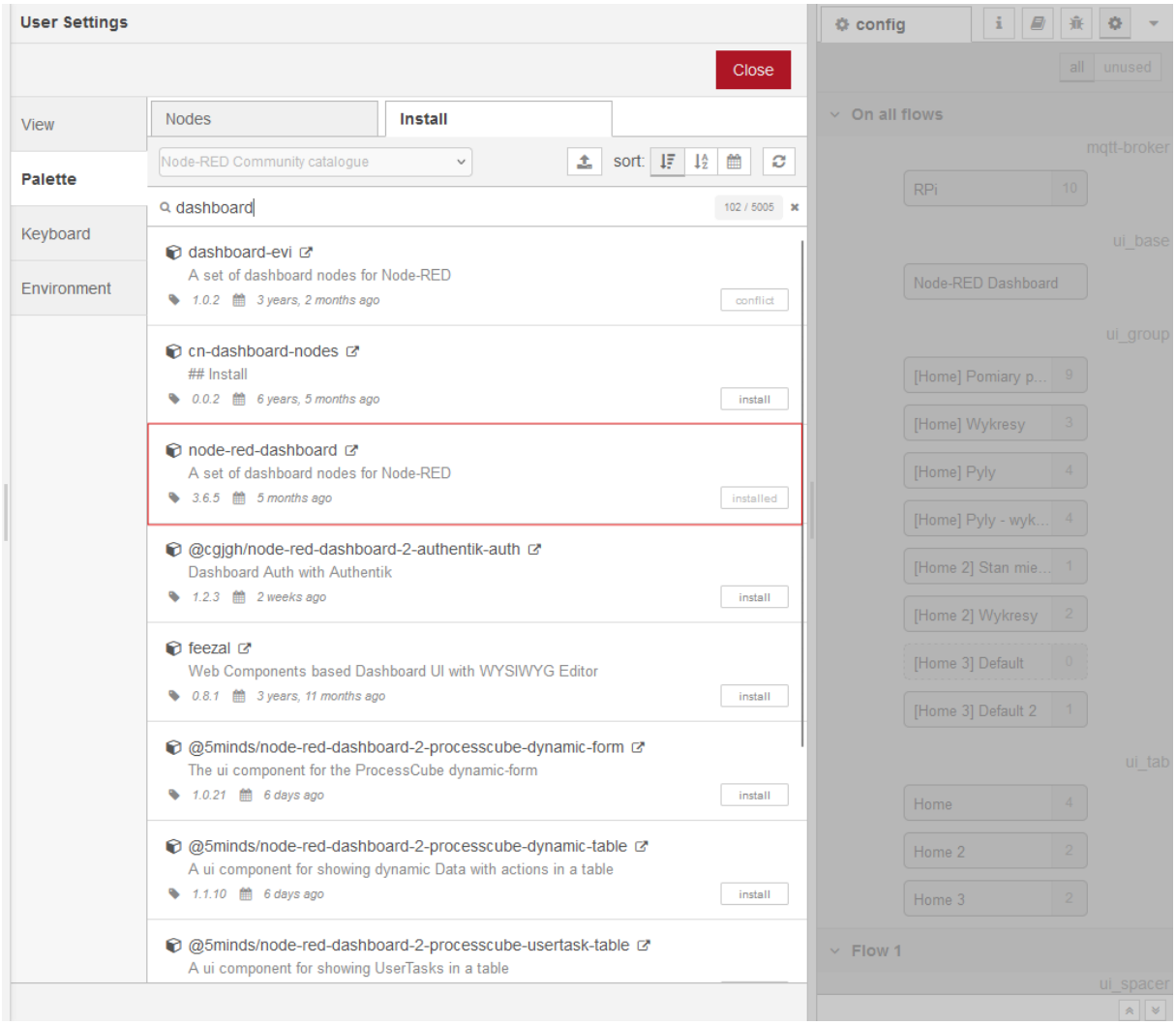
5.3.Konfiguracja brokera MQTT: Node-RED

Mając zainstalowany pakiet Mosquitto Broker oraz Node-RED na Raspberry Pi 4B (komendy instalacyjne) podano w skrypcie *(Skrypt do zajęć Projekt specjalnościowy platformy IoT opartej o Rasberry Pi (Projekt 1))* należy uruchomić oprogramowanie Node-RED. W tym celu należy w terminalu wpisać komendę: *sudo node-red-start.* Po jej uruchomieniu na terminalu pojawią się dane, za pomocą których można uruchomić wersję webową oprogramowania. W prezentowanym przykładzie będzie to *http://192.165.31.161:1880.*



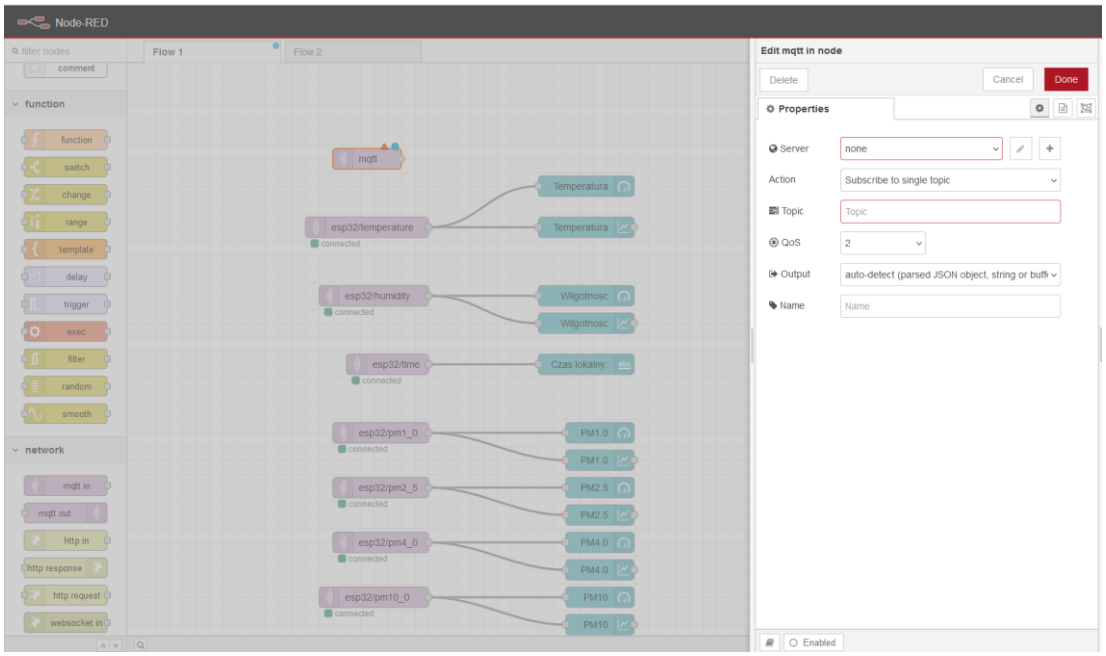
Rysunek 37 Uruchomienie Node-RED z terminala

Po wpisaniu adresu w przeglądarkę pojawia się okno konfiguracyjne, za pomocą którego można dodawać węzły oraz przekazywać dane pomiędzy publisher'em a subscriber'em. Aby móc prezentować dane w sposób liczbowy, a także graficzny (na wykresach) należy doinstalować pakiet *node-red-dashboard.* Instalację należy wykonać za pomocą narzędzia *Manage palette* (opcja dostępna po kliknięciu w trzy poziome kreski znajdujące się w prawym górnym rogu ekranu). Po uruchomieniu narzędzia należy wyszukać pakiet oraz go zainstalować.



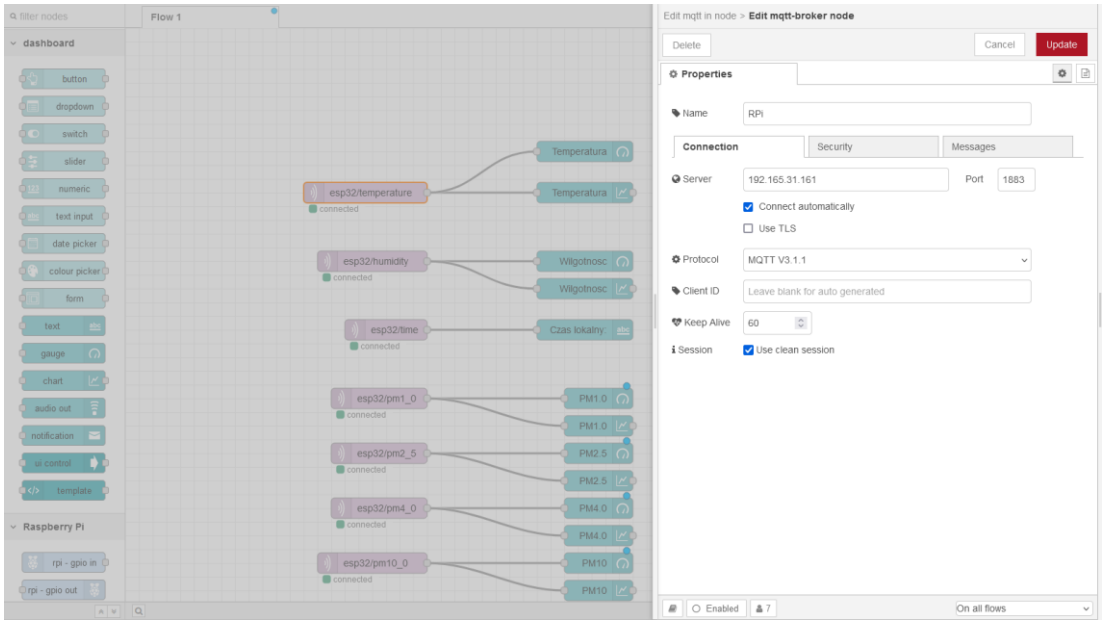
Rysunek 38 Instalacja pakietu Node-RED-dashboard

Po instalacji pakietu *node-red-dashboard* należy powrócić na stronę główną (*Flow 1*). Aby dodać węzeł MQTT z zakładki *network* należy wybrać opcję *mqtt in.* Pojawi się wówczas okno konfiguracyjne, w którym należy wybrać serwer (*Rapsberry Pi*) oraz wpisać temat (*Topic*).



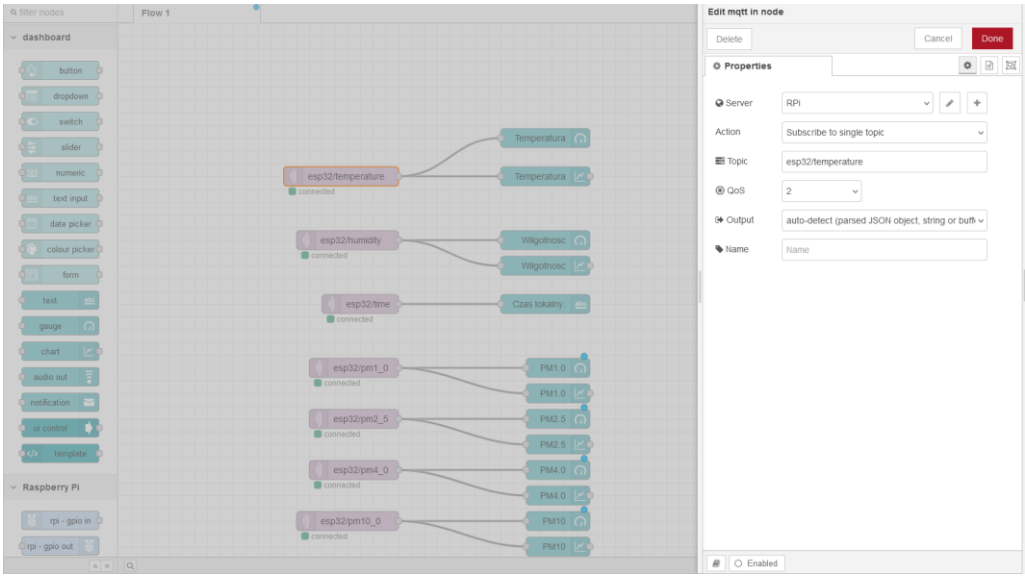
Rysunek 39 Dodanie węzła MQTT

W przypadku pierwszego uruchomienia lista serwerów będzie pusta. Należy wówczas skonfigurować nowy serwer klikając na symbol [+]. W nowo otwartym oknie *Właściwości* należy wpisać nazwę serwera, jego adres IP, wersję protokołu MQTT oraz ustawić parametr *Keep Alive,* który określa maksymalny czas (w sekundach) na odpowiedź, po upłynięciu którego urządzenie jest automatycznie rozłączane (w przypadku braku odpowiedzi).



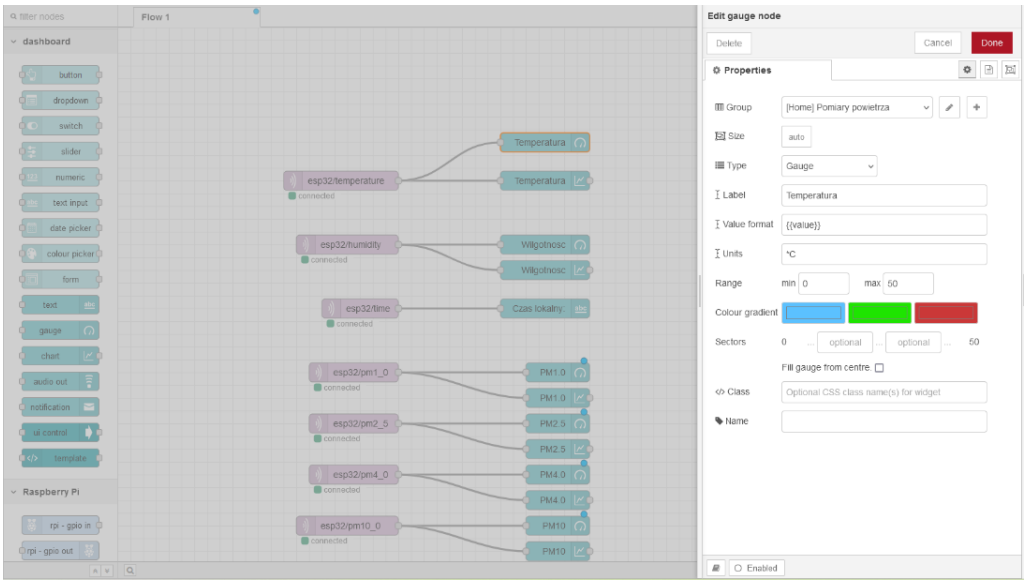
Rysunek 40 Ustawienia brokera MQTT

Po ustawieniu wszystkich parametrów węzła przykładową konfigurację przedstawiono na rysunku poniżej.



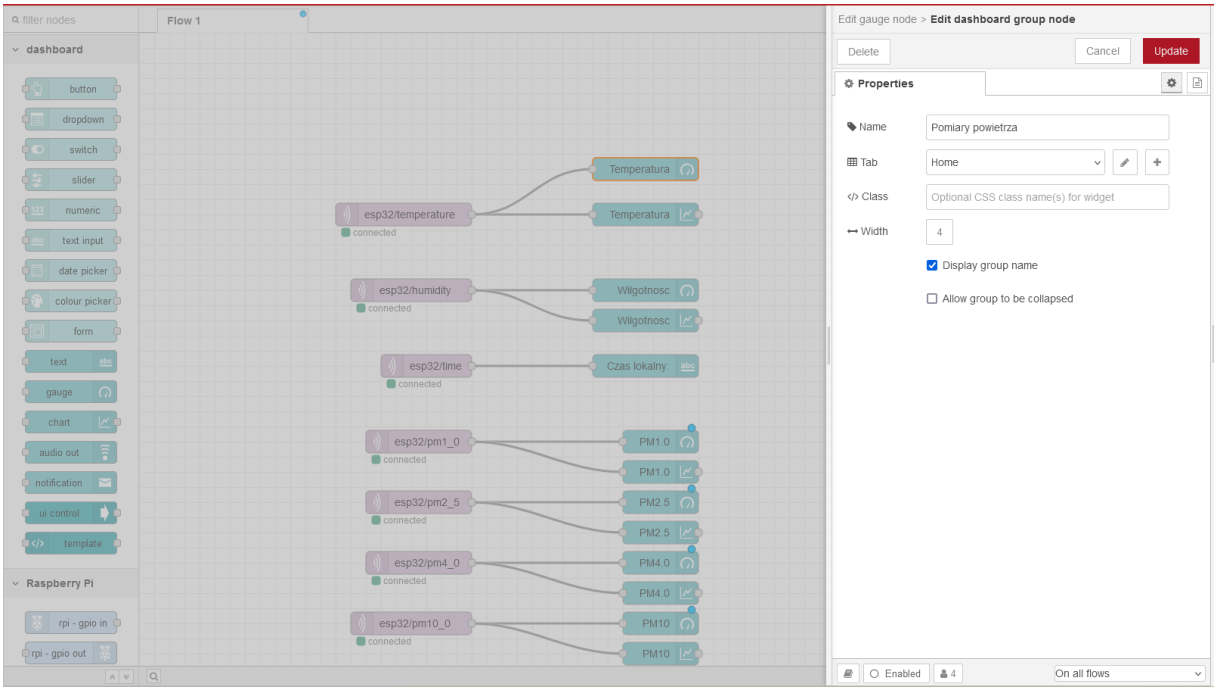
Rysunek 41 Konfiguracja węzła MQTT

Wyświetlanie danych w sposób graficzny jest możliwe z wykorzystaniem pól *text, guage* lub *chart* w zakładce *dashboard.* Po dodaniu pola oraz wykonaniu wirtualnego połączenia z danymi, które chcemy zobrazować na wykresie, należy uruchomić jego okno konfiguracyjne. Za jego pomocą można skonfigurować osie, kolorystykę i jednostki. Znajduje się tam również pole *Group,* które określa w jakiej grupie (kolumnie) wykres będzie się znajdował.

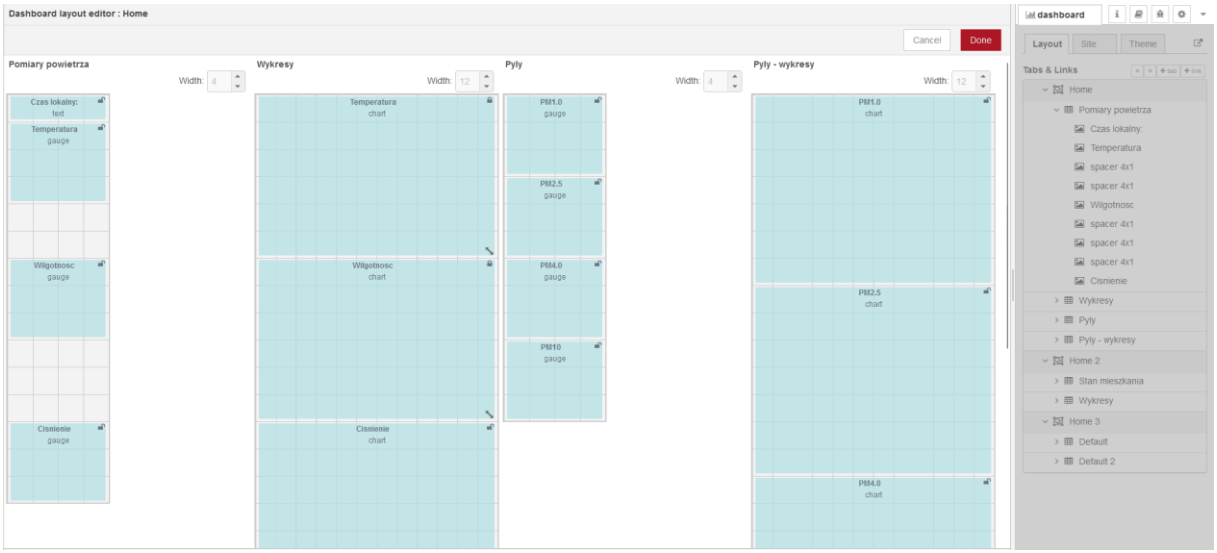


Rysunek 42 Edycja wskaźników graficznych

Jeżeli chcielibyśmy dodać kolumnę (prezentacja graficzna) należy kliknąć na symbol [+] znajdujący się obok pola *Group*. Wówczas pojawi się konfigurator nowej grupy (kolumny). W sytuacji, gdy prezentacja danych nie jest możliwa na jednej planszy (tablicy / ekranie) należy dodać nową tablicę przyciskiem [+] znajdującym się obok pola *Tab*.



Rysunek 43 Tworzenie nowej grupy



Rysunek 44 Przykładowe grupy wraz z rozmieszczeniem elementów prezentacji graficznej

Poniżej zaprezentowano finalny wygląd aplikacji. Wizualizacja danych jest możliwa wyłącznie po wgraniu oprogramowania na ESP32 i przesłaniu danych z czujników do brokera MQTT. Aplikacja webowa prezentująca dane jest dostępna pod adresem *http://192.165.31.161:1880/ui*

Rysunek 45 Aplikacja webowa – Node-Red

5.4.Opis programu

Wymagane biblioteki:

```
#include <DHT11.h>
#include <I2C_RTC.h>
#include <WiFi.h>
#include <WiFiClient.h>
#include <sps30.h>
#include <Adafruit_BMP280.h>
#include <AHT20.h>
#include <ArduinoMqttClient.h>
```

Dane połączenia sieciowego po Wi-Fi:

```
char ssid[] = "Moja_siec_WiFi";
char pass[] = "Moje_WiFi123";
```

Dane brokera MQTT:

```
const char broker[] = "192.165.31.161";
int  port = 1883;
```

Tematy publikowane do brokera MQTT:

```
const char topic[]  = "esp32/temperature";
const char topic2[]  = "esp32/humidity";
const char topic3[]  = "esp32/time";
const char topic4[]  = "esp32/pm1_0";
const char topic5[]  = "esp32/pm2_5";
const char topic6[]  = "esp32/pm4_0";
const char topic7[]  = "esp32/pm10_0";
const char topic8[]  = "esp32/pressure";
```

Definicja obiektów RTC (zegara czasu rzeczywistego), aht20 (czujnika temperatury i wilgotności), bmp (czujnika ciśnienia), client (połączenia Wi-Fi) oraz mqttClient:

```
static DS3231 RTC;
Adafruit_BMP280 bmp;
AHT20 aht20;
WiFiClient client;
MqttClient mqttClient(client);

unsigned long previousMillis = 0;
```

Inicjalizacja RTC, BMP280, AHT20:

```
void setup()
{
    int16_t ret;
    Serial.begin(9600);
    RTC.begin();
    aht20.begin();
    bmp.begin();
```

Ustawienie zegara RCT przykładową datą: 13.08.2024 i godziną 14:06:56:

```
    RTC.setHourMode(CLOCK_H12);

    RTC.setDay(13);
    RTC.setMonth(8);
    RTC.setYear(2024);

    RTC.setHours(14);
    RTC.setMinutes(6);
    RTC.setSeconds(56);

    RTC.setWeek(3);
```

Inicjalizacja czujnika jakości powietrza SPS30:

```
sensirion_i2c_init();

while (sps30_probe() != 0)
{
    Serial.print("SPS sensor probing failed\n");
    delay(500);
}
```

Aktywacja automatycznego przedmuchiwania czujnika SPS30:

```
ret = sps30_set_fan_auto_cleaning_interval_days(4);
if (ret)
{
    Serial.print("error setting the auto-clean interval: ");
    Serial.println(ret);
}

ret = sps30_start_measurement();
if (ret < 0)
{
    Serial.print("error starting measurement\n");
}
```

Inicjalizacja Wi-Fi:

```
WiFi.begin(ssid, pass);
int wifi_ctr = 0;
while (WiFi.status() != WL_CONNECTED)
{
    delay(500);
    Serial.print(".");
}

Serial.println("WiFi connected");
```

Inicjalizacja połączenia jako klient MQTT:

```
if (!mqttClient.connect(broker, port))
{
    Serial.print("MQTT connection failed! Error code = ");
    Serial.println(mqttClient.connectError());
    while (1);
}

Serial.println("You're connected to the MQTT broker!");
Serial.println();
}
```

```
void loop()
{
```

Zmienne zapisujące wartość temperatury, wilgotności oraz jakości powietrza:

```
int temperature = 0;
int humidity = 0;
mqttClient.poll();
struct sps30_measurement m;
uint16_t data_ready;
int16_t ret;
double pm1;
double pm25;
double pm4;
double pm10;
double pressure;
```

Zmienna pobierająca aktualną wartość czasu od uruchomienia programu:

```
unsigned long currentMillis = millis();
```

Odczyt danych z czujników do 2 sekundy (2000 ms):

```
if (currentMillis - previousMillis >= 2000L)
{
    float temperature = aht20.getTemperature();
    float humidity = aht20.getHumidity();
    pressure = bmp.readPressure() / 100.0;
    do
    {
        ret = sps30_read_data_ready(&data_ready);
        if (ret < 0)
        {
            Serial.print("error reading data-ready flag: ");
            Serial.println(ret);
        }
        else if (!data_ready)
        Serial.print("data not ready, no new measurement available\n");

        else
        break;

        delay(100); /* retry in 100ms */
    } while (1);
```

Odczyt danych z SPS30:

```
    ret = sps30_read_measurement(&m);
    if (ret < 0)
```

```
{
Serial.print("error reading measurement\n");
}
else
{
    pm1 = m.mc_1p0;
    pm25 = m.mc_2p5;
    pm4 = m.mc_4p0;
    pm10 = m.mc_10p0;
}
```

Odczyt aktualnej godziny i daty:

```
String hours = String(RTC.getHours());
String minutes = String(RTC.getMinutes());
String seconds = String(RTC.getSeconds());
String time = String(hours + ":" + minutes + ":" + seconds);

previousMillis = currentMillis;
```

Przesyłanie danych do brokera MQTT wiadomości z ustalonymi tematami:

```
mqttClient.beginMessage(topic);
mqttClient.print(temperature);
mqttClient.endMessage();

mqttClient.beginMessage(topic2);
mqttClient.print(humidity);
mqttClient.endMessage();

mqttClient.beginMessage(topic3);
mqttClient.print(time);
mqttClient.endMessage();

mqttClient.beginMessage(topic4);
mqttClient.print(pm1);
mqttClient.endMessage();

mqttClient.beginMessage(topic5);
mqttClient.print(pm25);
mqttClient.endMessage();

mqttClient.beginMessage(topic6);
mqttClient.print(pm4);
mqttClient.endMessage();

mqttClient.beginMessage(topic7);
mqttClient.print(pm10);
```

```
        mqttClient.endMessage();

        mqttClient.beginMessage(topic8);
        mqttClient.print(pressure);
        mqttClient.endMessage();
    }
}
```

6. Wykaz literatury

[1]. Andy King, „Programowanie Internetu rzeczy", APN PROMISE, 2021

[2]. Jeremy Blum, „Odkrywanie Arduino. Narzędzia i techniki inżynierii", Wiley, 2020

[3]. Udo Brandes, „ESP32 steuert Roboterauto", Elektor Verlag, 2022

[4]. Alexandru Radovici, Cristian Rusu, Ioana Culic, „Komercyjne i przemysłowe aplikacje Internetu rzeczy na Raspberry Pi", Apress, 2020

[5]. Markus Edenhauser, „Node-RED: IoT Programmierung mit ESP32 & MQTT", pixeledi.eu, 2023

[6]. Dhairya Parikh, „Raspberry Pi and MQTT Essentials. A complete guide to helping you build innovative full-scale prototype projects using Raspberry Pi and MQTT protocol", Packt Publishing, 2022

[7]. Dokumentacje techniczne modułów elektronicznych użytych w projektach pobrane ze stron producentów lub dostawców

[8]. Źródło internetowe, dostęp 19.08.2024r. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/hw-reference/esp32/get-started-devkitc.html

Spis rysunków