Konstrukcja stacji pogodowej opartej na mikrokontrolerze ESP32 z interfejsem użytkownika oraz API REST

Szymon Uglis

# Wstęp

## Wprowadzenie

Dzięki szerokiemu dostępowi do internetu mamy łatwy dostęp do danych pogodowych aktualnych jak i historycznych z całego świata. Istnieje wiele serwisów, programów telewizyjnych, które aktualne dane pogodowe, prognozy pogody prezentują nam w przystępny sposób. Problemem z poleganiem na danych pogodowych z popularnych serwisów jest relatywnie niska dokładność aktualnych warunków pogodowych jak i prognoz pogody. Bowiem serwisy te nie mogą mieć stacji pogodowowych rozstawionych co na przykład kilometr, aby dokładność dla każdego potencjalnego zainteresowana była wysoka. Dlatego polega się w dużym stopniu na ogólnych danych z kilku, bądź kilkunastu stacji w danym regionie, aby wykonać ekstrapolację dla aktualnych warunków pogodowych dla całego regionu. Serwisy udostępniające dane pogodowe dla lokalizacji są tylko przybliżeniem faktycznych stanu jaki znajduje sie w danym miejscu.

## Cel projektu oraz opis tworzonego rozwiązania

Celem projektu jest zbadanie przydatności modułu ESP32 oraz kompatybilnych komponentów do celów kolekcji danych pogodowych. Rezultatem będzie utworzenie urządzenia opartego na systemie wbudowanym pozwalającym na wielofunkcyjny pomiar parametrów pogodowych oraz udostępnianie ich za pomocą witryny www.

Urządzenie, będzie udostępniało również interfejs programistyczny API REST, które będzie umożliwiało integracje oraz dalszy rozwój projektu (np. integracja z systemem home assistant czy innymi urządzeniami Internetu rzeczy (IoT)).

### Funkcje urządzenia

* Pomiar oraz kalkulacja danych pogodowych na podstawie danych wejściowych z czujników
* Udostępnienie i agregacja danych na stronie www
* Udostępnienie interfejsu programistycznego REST
* Możliwość zmiany parametrów odczytów sensorów

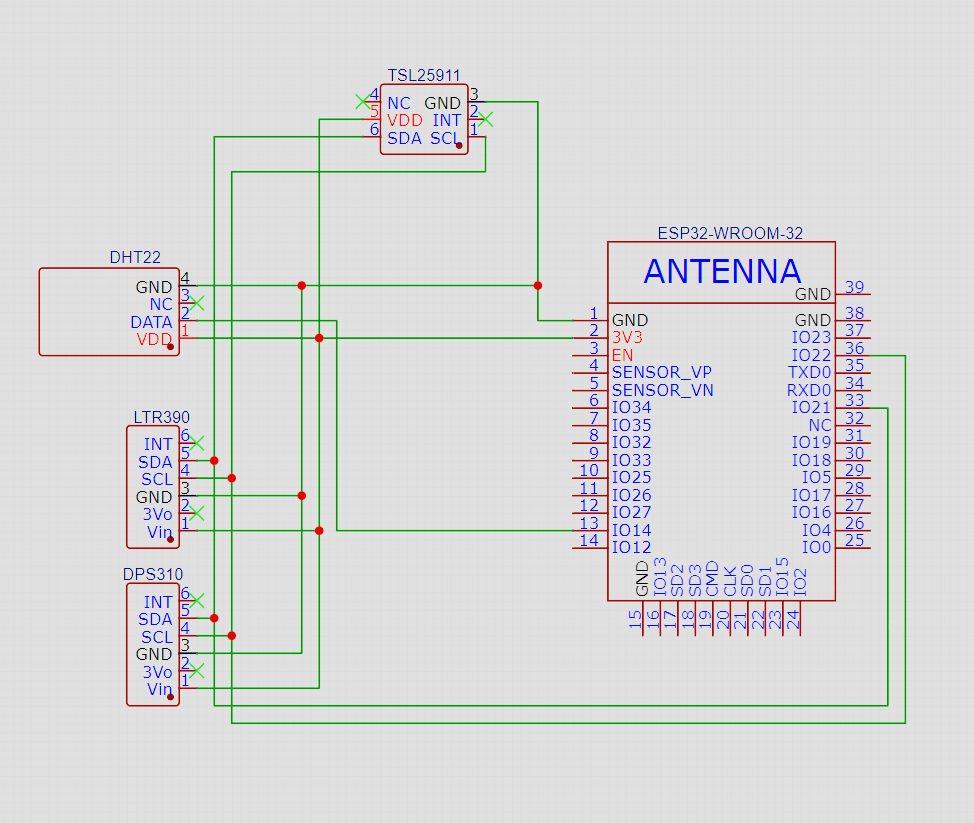
### Dostęp zdalny do urządzenia

Urządzenia ze świata IoT są najczęściej urządzeniami lokalnymi - bez publicznego adresu IP i/lub bez możliwości zdalnego dostępu. Ze względu na niską moc takich urządzeń, ubogie oprogramowanie, w które takie urządzenia są wyposażone bardzo często nie zawierają wielu ważnych zabezpieczeń przeciwko potencjalnym atakom. Dodatkowo dane udostępnianie przez urządzenia IoT z bardzo często są danymi które nie powinny byc dostępne publicznie (np. dane o obecności osób w pomieszczeniu lub czy zamek od drzmi jest zamknięty).

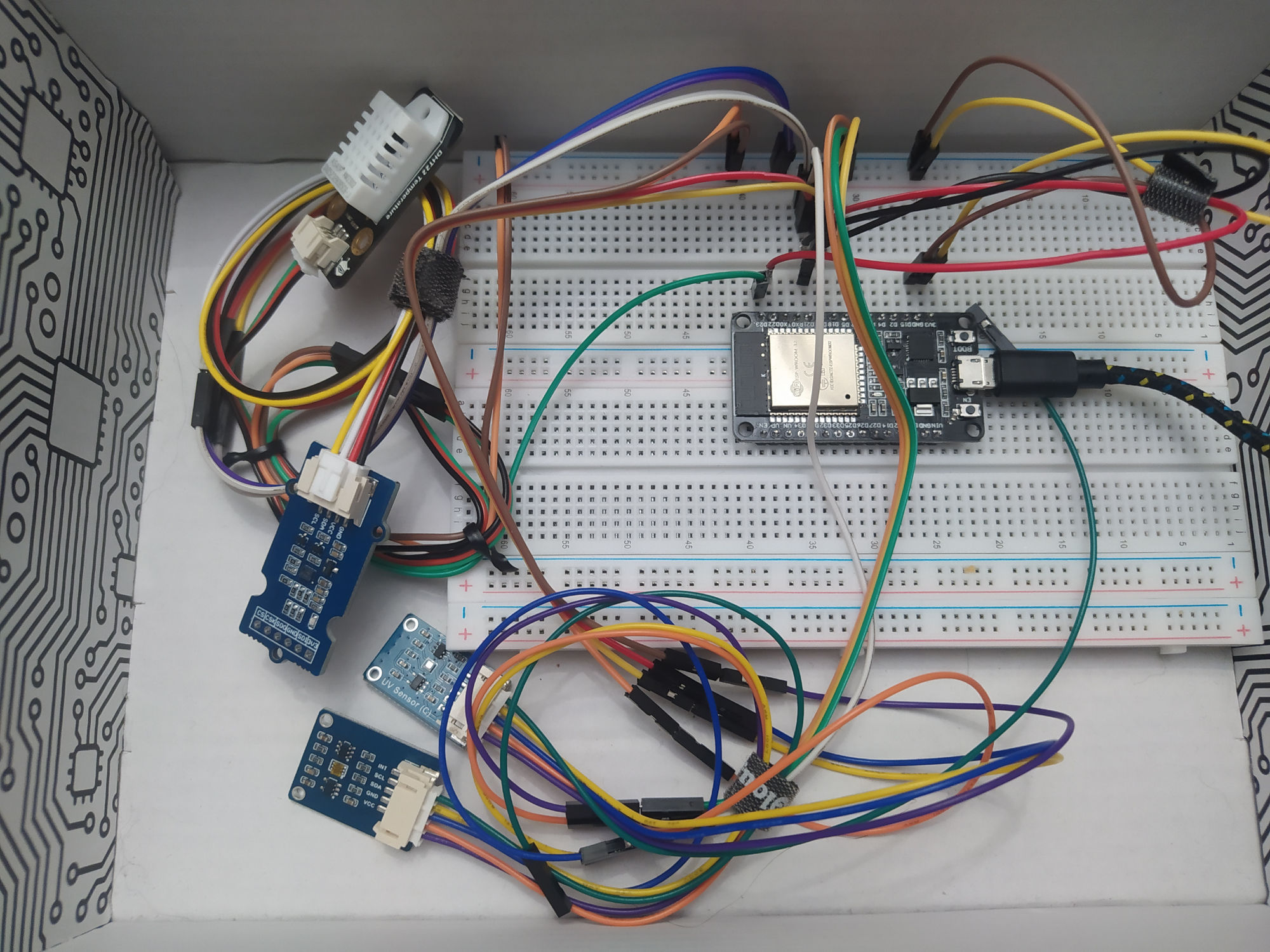
Zalecanym jest, aby dostęp do urządzeń IoT był całkowicie lokalny (najlepiej dodatkowo zablokować dostęp do internetu dla tych urządzeń), a zbieraniem i udostępnianiem danych publicznych było realizowane przez oprogramowanie dedykowane do tego celu. Aby umożliwić dostęp do danych zbieranych przez tworzone urządzenie będzie implementować możliwość konfiguracji wysyłania danych za pomocą MQTT. Dane zaagregowane przez MQTT dalej mogą być bezpiecznie filtrowane, przetwarzane i udostępniane publicznie przez oprogramowanie do tego przystosowane.

# Konfiguracja sprzętowa

* Mikrokontroler - ESP-WROOM-32
* Czujnik natężenia światła - TSL25911
* Czujnik temperatury i wilgotności powietrza - DHT22
* Czujnik ciśnienia oraz temperatury - DPS310
* Czujnik światła ultrafioletowego - LTR390



Schemat urządzenia



Fotografia prototypu urządzenia

## ESP-WROOM-32

ESP-WROOM-32 (albo ESP32-WROOM-32) to mikrokontroler ze zintegrowanym WIFI oraz bluetooth. Nadaje się do szerokiej gamy zastosowań: od kontrolera czujników z niskim poborem energii do zaawansowanych zadań enkodowania sygnałów głosowych czy muzyki. Moduły bazujące na rdzeniach ESP32 zyskały popularność ze wzklędu na zaintegrowaną obsługę WIFI, niskim kosztom oraz bogatej dokumentacji oraz wielu możliwych integracji.

Mikrokontroler znajdzie zastosowanie w prostych jak i bardziej skomplikowanych projektach, dzięki dwóm rdzenion, które mogą być kontrolowane osobno, szerokich możliwościach podłączenia urządzeń peryferyjnych (wsparcie dla: I2C, UART, SPI oraz inne).

ESP-WROOM-32 składa się z mikrokontrolera ESP32-D0WDQ6 posiadającego dwa rdzenie, które pozwają na prace od 80MHz do 240 MHz. Procesor również posiada ko-procesor o niskim poborze mocy, który może zostać użyty zamiast głównych rdzeni w przypadku kiedy nie jest potrzebna duża moc obliczeniowa.

Integracja WIFI, Bluetooth i Bluetooth LE do mikrokontrolera pozwala na rozszerzenie możliwych zastosowań, gdzie nie jest możliwe podłączenie do sieci w konwencjonalny sposób za pomocą kabla. Komunikacja bezprzewodowa również umożliwia całkowicie zdalne zastosowania z wykorzystaniem akumulatorów oraz paneli słonecznych do zasilania mikrokontrolera.

### Specyfikacja ESP-WROOM-32

Najważniejsze specyfikacje ESP-WROOM-32

| Element | Specyfikacja |
| --- | --- |
| Procesor | 2 rdzenie (80-240MHz) |
| WIFI | 802.11 b/g/n (802.11n: przepustowość 150 Mbps) |
| Bluetooth | Bluetooth v4.2 i Bluetooth LE |
| Pamięć SPI Flash | 4MB |
| Napięcie zasilania | 3.0 V - 3.6 V |
| Zużycie prądu | Średnie: 80 mA |

Do realizacji projektu została wybrany mikrokontroler ESP-WROOM-32 ze względu na:

* bogate możliwości podłączenia urządzeń peryferyjnych
* wbudowane wsparcie połączeń WIFI
* dużą ilość pamięci flash
* niewielką cenę w porównaniu do oferowanych specyfikacji
* łatwą dostępność

Potencjalnymi alternatywami, które nie zostały wybrane do realizacji projektu są:

* Arduino Uno R3 (brak wbudowanej obsługi połączeń sieciowych)
* ESP8266 (wolniejszy procesor, mniejsza ilość pamięci flash, mniejsza ilość wyprowadzeń do podłączenia urządzeń peryferyjnych)
* Olimex ESP32-POE (wbudowana obsługa połączenie przewodowego ethernet, mniejsza ilość wyprowadzeń do podłączenia urządzeń peryferyjnych)
* LILYGO T-SIM7000G (wbudowana obsługa LTE, mniejsza ilość wyprowadzeń do podłączenia urządzeń peryferyjnych, wysoka cena)

W finalnym produkcie w zależności od przeznaczenia użycie łączności bezprzewodowej może być niewskazane (daleka odległość od punktu dostępowego WIFI, duża ilość przeszkód). Użycie innych wersji ESP32 z wbudowaną obsługą przewodowego połączenia sieciowego lub możliwością połączenia do sieci przez LTE, może być wymagane - jednak w ramach realizacji zakresu projektu wybór padł na łatwiej dostępny moduł z łącznością bezprzewodową WIFI.

## Czujnik natężenia światła - TSL25911

TSL25911 to czujnik światła z odpowiedzią podobną do ludzkiego oka. Posiada czułość 188 uLux oraz zakres dynamiczny 1 do 600,000,000. W porównaniu do innych tańszych rozwiązań bazujących na fotokomórkach CdS, daje o wiele bardziej precyzyjne rezultaty, z możliwością zmiany czułości i rozdzielczości w locie.

### Specyfikacja TSL25911

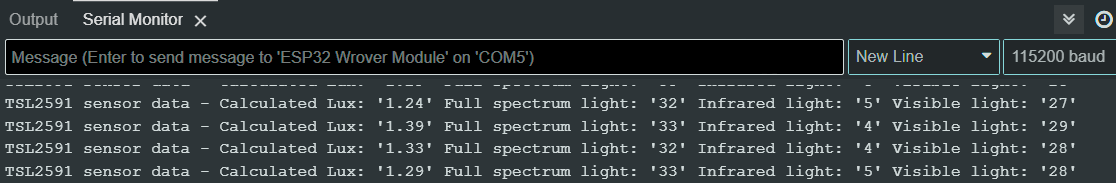
Specyfikacja TSL25911

| Element | Specyfikacja |
| --- | --- |
| Rozdzielczość dynamiczna | 1 do 600,000,000 |
| Zakres pomiaru | 188 uLux - 88,000 Lux |
| Zakres temperatury pracy | -30 - 80 °C |
| Interfejs | I2C (adres: 0x29 i 0x28) |

### Przykładowy program testu czujnika TSL25911

#include <Wire.h>  
  
#include "Adafruit\_TSL2591.h"  
  
Adafruit\_TSL2591 tsl = Adafruit\_TSL2591(2591);  
  
void setup() {  
 Wire.begin();  
 Serial.begin(115200);  
  
 tsl.setGain(TSL2591\_GAIN\_MED);  
 tsl.setTiming(TSL2591\_INTEGRATIONTIME\_300MS);  
}  
  
void loop() {  
 uint32\_t lum = tsl.getFullLuminosity();  
 uint16\_t ir = lum >> 16;  
 uint16\_t full = lum & 0xFFFF;  
  
 Serial.print(F("TSL2591 sensor data - Calculated Lux: '"));  
 Serial.print(tsl.calculateLux(full, ir));  
 Serial.print(F("' Full spectrum light: '"));  
 Serial.print(full);  
 Serial.print(F("' Infrared light: '"));  
 Serial.print(ir);  
 Serial.print(F("' Visible light: '"));  
 Serial.print(full - ir);  
 Serial.println(F("'"));  
}

### Test czujnika TSL25911



Rezultat testu czujnika TSL25911

## Czujnik temperatury i wilgotności powietrza - DHT22

Czujnik DHT22 składaja się z dwóch części: pojemnościowego czujnika wilgotności i termistora. Wewnątrz znajduje się również bardzo prosty chip, który dokonuje konwersji sygnału analogowego na cyfrowy i wysyła sygnał cyfrowy z temperaturą i wilgotnością. Sygnał cyfrowy jest łatwy do odczytania za pomocą dowolnego mikrokontrolera.

Czujnik ten cechuje sie bardzo niską ceną oraz bardzo dużą dokładnością w stosunku do ceny.

Potencjalnymi alternatywami, które mogły być wykorzystane to:

* DHT11 (niższa dokładność)
* AHT20 (niższa dokładność odczytu temperatury oraz podobna dokładność odczytu wilgotności, dostępny tylko jeden adres I2C)

Został wybrany DHT11 ze względu na prostote podłączenia (jedno wyjście cyfrowe) co sprawia, że możliwe jest podłączenie kilku czujników do jednego mikrokontrolera (co nie możliwe jest w przypadku AHT20, bez wykorzystania na przykład drugiego interfejsu I2C). Czujnika DHT22 cechuje się również niskim kosztem oraz wysoką dokładność w stosunku do ceny czujnika, co zadecydowało o finalnym wyborze tego produktu.

### Specyfikacja DHT22

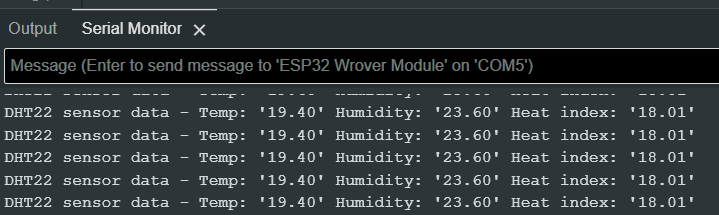
Specyfikacja DHT22

| Element | Specyfikacja |
| --- | --- |
| Zakres oraz dokładność odczytu temperatury | -40 to 80 °C (dokładność: 0.5°C) |
| Zakres oraz dokładność odczytu wilgotności | 0 to 100 RH% (dokładność: 2-5 %) |
| Maksymalna częstotliwość odczytu danych | 0.5Hz |
| Interfejs | One Wire (jeden przewód) |

### Przykładowy program testu czujnika DHT22

#include <DHT.h>  
  
#define DHTPIN 23  
#define DHTTYPE DHT22  
  
DHT dht(DHTPIN, DHTTYPE);  
  
void setup() {  
 Serial.begin(115200);  
   
 dht.begin();  
 Serial.print(F("Works"));  
}  
  
void loop() {  
 auto humidity = dht.readHumidity();  
 auto temperature = dht.readTemperature();  
  
 if (isnan(humidity) || isnan(temperature)) {  
 Serial.println(F("Failed to read from DHT sensor!"));  
 return;  
 }  
  
 auto heat\_index = dht.computeHeatIndex(temperature, humidity, false);  
   
 Serial.print(F("DHT22 sensor data - Temp: '"));  
 Serial.print(temperature);  
 Serial.print(F("' Humidity: '"));  
 Serial.print(humidity);  
 Serial.print(F("' Heat index: '"));  
 Serial.print(heat\_index);  
 Serial.println(F("'"));  
}

### Test czujnika DHT22



Rezultat testu czujnika DHT22

## Czujnik ciśnienia oraz temperatury - DPS310

Czujnik DP310 służy do dokładnego pomiaru ciśnienia atmosferycznego oraz posiada dodatkowy czujnik temperatury. Posiada wysoką dokładność obsolutną +-1 hPa oraz pracuje w zakresie od 300 do 1200 hPa. Czujnik cechuje sie niską ceną, wysoką dokładnością w stosunku do ceny, oraz posiada dodatkowy czujnik temperatury, który może służyć jako preferowana metoda pomiaru temperatury, albo jako dodatkowy punkt w danych pomiarowych.

Potencjalnymi alternatywami, które mogły być wykorzystane to:

* EE895 (niższa dokładność pomiaru ciśnienia (700-1100 hPa, wysoka cena ze względu na obecność czujnika C02))
* BMP180 (niższa dokładność pomiaru ciśnienia)

Czujnik został wybrany z powodu konkurencyjnej ceny, dużej dostępności oraz wysokiej dokładności pomiaru w stosunku do ceny produktu.

### Specyfikacja DPS310

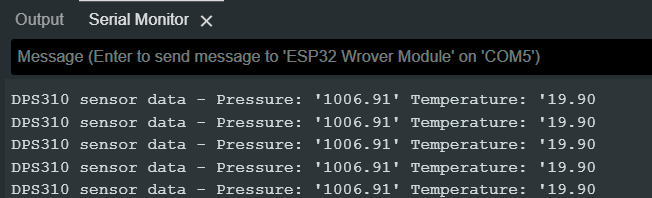
Specyfikacja DPS310

| Element | Specyfikacja |
| --- | --- |
| Zakres oraz dokładność odczytu temperatury | -40 to 85 °C (dokładność: 1°C) |
| Zakres odczytu ciśnienia atmosferycznego | 300 do 1200 hPa |
| Dokładność absolutna odczytu ciśnienia atmosferycznego | 1 hPa |
| Dokładność dynamiczna odczytu ciśnienia atmosferycznego | 0.002 hPa |
| Interfejs | I2C (domyślny adres: 0x77) |

### Przykładowy program testu czujnika DPS310

#include <Wire.h>  
  
#include "Adafruit\_DPS310.h"  
  
Adafruit\_DPS310 dps;  
  
void setup() {  
 Serial.begin(115200);  
 Wire.begin();  
  
 dps.begin\_I2C();  
  
 dps.configurePressure(DPS310\_64HZ, DPS310\_64SAMPLES);  
 dps.configureTemperature(DPS310\_64HZ, DPS310\_64SAMPLES);  
}  
  
void loop() {  
 sensors\_event\_t pressure\_event;  
 sensors\_event\_t temperature\_event;  
 dps.getEvents(&temperature\_event, &pressure\_event);   
  
 Serial.print(F("DPS310 sensor data - Pressure: '"));  
 Serial.print(pressure\_event.pressure);  
 Serial.print(F("' Temperature: '"));  
 Serial.print(temperature\_event.temperature);  
 Serial.println(F(""));  
}

### Test czujnika DPS310



Rezultat testu czujnika DPS310

## Czujnik światła ultrafioletowego - LTR390

Czujnik LTR390 jest urządzeniem o bardzo niskim koszcie oraz wysokiej dokładności na światło w zakresie od 300nm do 350 nm. Czujnik posiada dwa sensory - jeden do światła UV, drugi do światła widzialnego co pozwala na precyzyjne określenie ilości padającego na czujnik światła UV. W porównaniu do innych czujników oferuje bezpośredni odczyt wartości światła UV z sensora (czujnik SI1145 oblicza ilość światła UV z całkowitego odczytu padającego światłą we wszystkich zakresach) oraz pozwala na odczyt danych za pomocą interfejsu I2C (bez konieczności posiadania interfejsów SPI czy posiadania dodatkowych konwerterów analogowo-cyfrowych)

Potencjalnymi alternatywami, które mogły być wykorzystane to:

* SI1145 (niższa dokładność pomiaru światła UV - brak bezpośredniego odczytu - wartość obliczana)
* GUVA-S12SD (wymagany konwerter analogow-cyfrowy do odczytu danych)

Czujnik został wybrany z powodu wysokiej dokładności (bezpośredni odczyt wartości padającego światła UV) oraz prostoty podłączenia czujnika do mikrokontrolera (interfejs I2C). Bardzo niska cena oraz możliwość odczytu natężenia światła (w lux) jest dodatkowym plusem.

### Specyfikacja LTR390

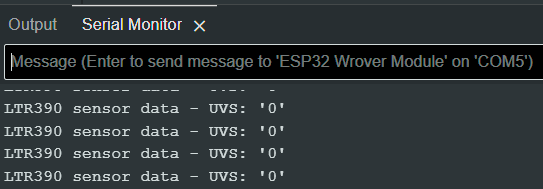
Specyfikacja LTR390

| Element | Specyfikacja |
| --- | --- |
| Zakres odczytu światła UV | od 300nm do 350nm |
| Zakres odczytu światła widzialnego | od 0 do 524287 (20 bitowy zakres) |
| Interfejs | I2C (adres: 0x53) |

### Przykładowy program testu czujnika LTR390

#include <Wire.h>  
  
#include "Adafruit\_LTR390.h"  
  
Adafruit\_LTR390 ltr = Adafruit\_LTR390();  
  
void setup() {  
 Serial.begin(115200);  
 Wire.begin();  
  
 ltr.begin();  
  
 ltr.setMode(LTR390\_MODE\_UVS);  
 ltr.setGain(LTR390\_GAIN\_3);  
 ltr.setResolution(LTR390\_RESOLUTION\_16BIT);  
}  
  
void loop() {  
 Serial.print(F("LTR390 sensor data - UVS: '"));  
 Serial.print(ltr.readUVS());  
 Serial.println(F("'"));  
}

### Test czujnika LTR390



Rezultat testu czujnika LTR390

# Środowisko programowe

Do implementacji systemu wbudowanego wykorzystane zostaną:

* Język C++ do utworzenia oprogramowania mikrokontrolera
* Język HTML oraz CSS do utworzenia interfejsu www
* Środowisko programistyczne Arduino Studio w wersji 2.2.1

# Oprogramowanie

Do implementacji serwera http została użyta biblioteka ESP32 HTTPS Server .

#include <HTTPServer.hpp>  
#include <HTTPRequest.hpp>  
#include <HTTPResponse.hpp>  
  
httpsserver::HTTPServer http\_server = httpsserver::HTTPServer();  
  
void handle\_api\_request(httpsserver::HTTPRequest \*req, httpsserver::HTTPResponse \*res);  
void handle\_root\_request(httpsserver::HTTPRequest \*req, httpsserver::HTTPResponse \*res);  
  
void setup(void)  
{  
 httpsserver::ResourceNode \*nodeRoot = new httpsserver::ResourceNode(  
 "/",   
 "GET",   
 &handle\_root\_request  
 );  
 httpsserver::ResourceNode \*nodeApi = new httpsserver::ResourceNode(  
 "/api",   
 "GET",   
 &handle\_api\_request  
 );  
 http\_server.registerNode(nodeRoot);  
 http\_server.registerNode(nodeApi);  
 http\_server.start();  
 if (http\_server.isRunning()) {  
 Serial.println("Server ready.");  
 }  
}  
  
void loop(void)  
{  
 http\_server.loop();  
}

Biblioteka pozwala na proste definiowanie adresów podstron, wraz z przyjmowanym typem zapytania http. Gdy serwer odbierze połączenie od klienta z prawidłowym adresem, wykonuje metode zarejestrowaną w obiekcie serwera, w przeciwnym wypadku zwraca odpowiedź z kodek 404 Not Found.

## Dostęp do danych przez przeglądarkę

Urządzenie udostępnia stronę www z odczytami czujników w prostym formacie dla odczytu przez człowieka. Przy pomocy HTML oraz CSS prezentowana jest tabela ze wszystkimi danymi, które udostępnia urządzenie, informacja jaki czas temu dane były aktualizowane oraz jak długo urządzenie pracuje.

Aktualne dane zebrane przez urządzenie są wstrzykiwane do szablonu http za pomocą funkcji sprintf. W pierwszym parametrze podana jest zmienna do bufora na przetwarzane, w drugim zmienna z szablonem html a reszta parametrów to poszczególne dane, które mają być wstrzyknięte. Następnie gotowy plik html jest wysyłany do użytkownika.

void handle\_root\_request(  
 httpsserver::HTTPRequest \*req,   
 httpsserver::HTTPResponse \*res  
) {  
 res->setHeader("Content-Type", "text/html");  
  
 char buffer[2000];  
 sprintf(  
 buffer,   
 html\_template,  
 mdns\_hostname,  
 mdns\_hostname,  
 current\_reading\_data.temperature,  
 current\_reading\_data.humidity,  
 current\_reading\_data.heat\_index,  
 current\_reading\_data.visible\_light,  
 current\_reading\_data.full\_spectrum\_light,  
 current\_reading\_data.infrared\_light,  
 current\_reading\_data.calculated\_lux,  
 current\_reading\_data.pressure,  
 current\_reading\_data.uvs,  
 sensor\_read\_delay,  
 calculate\_uptime()  
 );  
  
 res->print(buffer);  
}

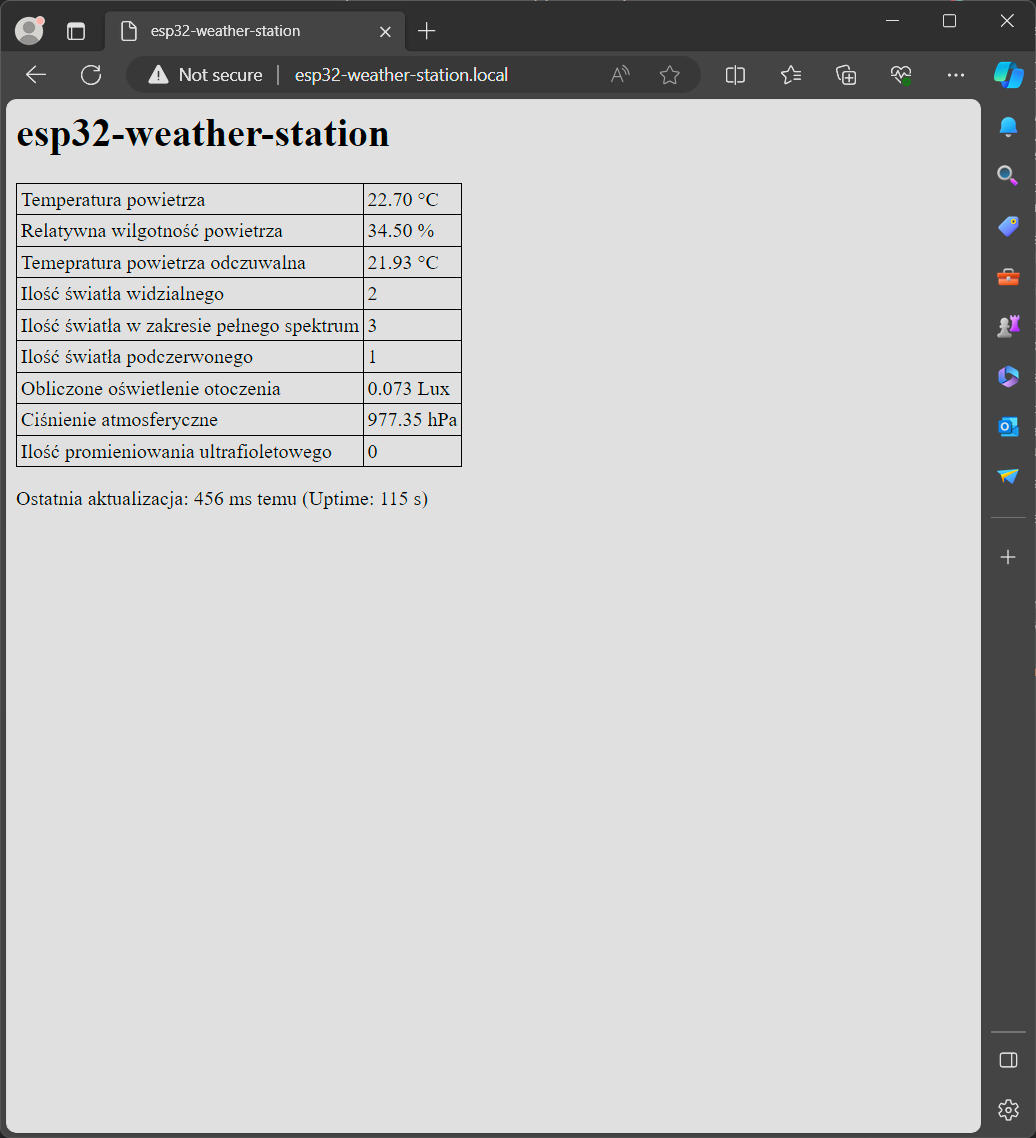
Szablon pliku html jest przechowywany w pamięci programowej programu w stałej html\_template. Zaletą takiego rozwiązania jest prostota implementacji oraz szybkość działania - dane są zawsze gotowe do wykorzystania. Alternatywnie można byłoby wykorzystać pamięć SPIFFS urządzenia do przechowywania szablonu html - byłoby to bardziej elastyczne rozwiązanie, które nie wymagało by ponownej kompilacji programu, aby zmienić szablon html. Z drugiej strony rozwiązanie to mogłoby powodować zwiększony narzut przy generowaniu odpowiedzi przy każdym zapytaniu. Ze względu na prostotę pierwszego rozwiązania oraz potencjalne problemy z wydajnością drugiego rozwiązania zdecydowałem sie załadować szablon do kodu programu w tej wersji oprogramowania. Do deklaracji zawartości zmiennej jest użyta deklaracja literał ciągu znaków (string literal), aby w prosty sposób było możliwe zawarcie w środku zmiennej znaków specjalnych, znaków niedozwolonych oraz w celu Udostępnienia możliwości deklaracji wieloliniowego ciągu znaków.

const char html\_template[] = R"###(  
 <html lang="en">  
 <head>  
 ...  
)###";

Przyszłym usprawnieniem urządzenia mogłoby być sprawdzanie przydatności pamięci SPIFFS do przechowywania szablonów html, aby oszczędzić na pamięci programowej.

### Testy interfejsu WWW

Interfejs WWW użytkownika zawiera komplet danych zbieranych przez urządzenie oraz kiedy ostatni raz dane były aktualizowane. Odświeżenie przeglądarki spowoduje pobranie nowych danych z urządzenia.

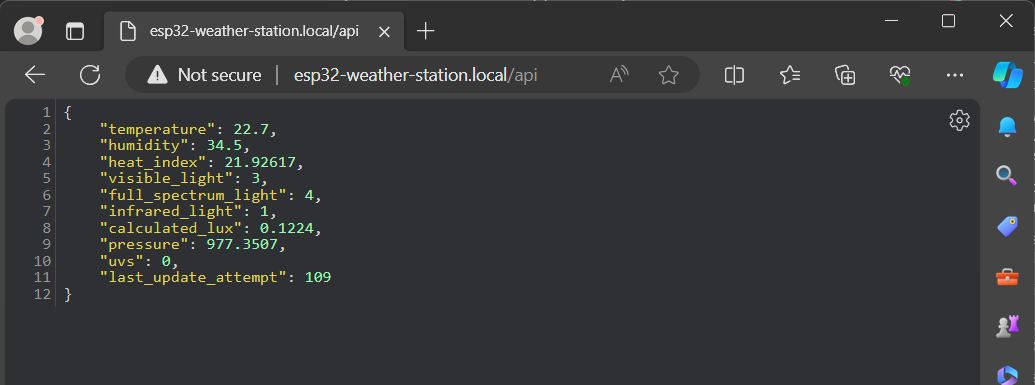


Test interfejsu WWW

## Dostęp do danych przez REST API

Urządzenie udostępnia dane również poprzez REST API w postaci JSON - umożliwia to łatwą integrację urządzenia z innymi narzędziami, albo umożliwia stworzenie własnych integracji.

### Testy REST API



Test interfejsu REST API

## Dostęp do danych diagnostycznych

Urządzenie udostępnia dane, które mogą służyć do debugowania urządzenia za pomocą interfejsu szeregowego. Podczas procesu uruchamiania na port szeregowy wysyłane są dane informujące o przebiegu procesu uruchamiania urządzenia, połączeniu do sieci bezprzewodowej oraz po poprawnym uruchomieniu, aktualnie zebrane dane pogodowe.

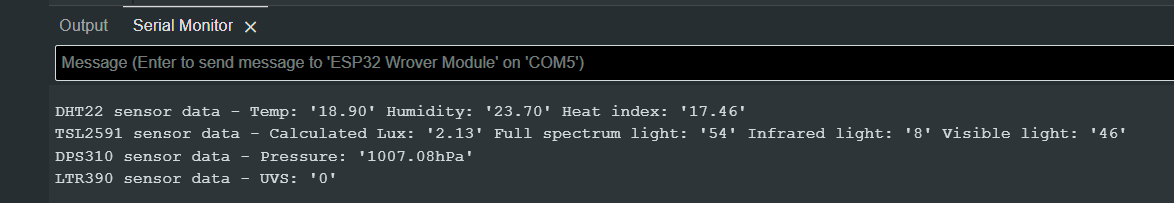
Funkcja send\_debug\_info\_to\_serial wysyła dane na interfejs szeregowy w postaci łatwej do przeczytania dla człowieka:

void send\_debug\_info\_to\_serial()  
{  
 Serial.print(F("DHT22 sensor data - Temp: '"));  
 Serial.print(current\_reading\_data.temperature);  
 Serial.print(F("' Humidity: '"));  
 Serial.print(current\_reading\_data.humidity);  
 Serial.print(F("' Heat index: '"));  
 Serial.print(current\_reading\_data.heat\_index);  
 Serial.println(F("'"));  
  
 Serial.print(F("TSL2591 sensor data - Calculated Lux: '"));  
 Serial.print(current\_reading\_data.calculated\_lux);  
 Serial.print(F("' Full spectrum light: '"));  
 Serial.print(current\_reading\_data.full\_spectrum\_light);  
 Serial.print(F("' Infrared light: '"));  
 Serial.print(current\_reading\_data.infrared\_light);  
 Serial.print(F("' Visible light: '"));  
 Serial.print(current\_reading\_data.visible\_light);  
 Serial.println(F("'"));  
  
 Serial.print(F("DPS310 sensor data - Pressure: '"));  
 Serial.print(current\_reading\_data.pressure);  
 Serial.println(F("hPa'"));  
  
 Serial.print(F("LTR390 sensor data - UVS: '"));  
 Serial.print(current\_reading\_data.uvs);  
 Serial.println(F("'"));  
  
 Serial.println(F(""));  
}

Jako, że powyższa funkcja operuje na dużej ilość tekstu w sposób cały, zostało zastosowana optymalizacja w postaci użycia funkcji F - funkcja ta sprawia, że dany łańcuch znaków nie jest kopiowany do pamięci PSRAM mikrokontrolera przed wysłaniem na interfejs szeregowy, a dane są bezpośrednio kopiowane z programu do strumienia, który wysyłane dane na interfejs.

## Test danych diagnostycznych

To przeprowadzenia testy zostało użyte narządzie Serial Monitor z Arduino Studio. Mikrokontroler wysyła dane na port szeregowy w prędkości 115200 baud.



Test danych diagnostycznych

## Integracja z MQTT

Urządzenie dodatkowo udostępnia możliwość połączenia do brokera MQTT - co umożliwia bezpieczne udostępniane danych do serwera zewnętrznego.

MQTT to prosty oraz lekki protokół wymiany danych oparty o wzorzec publikacja/subskrypcja. Przeznaczony do zastosowań nie wymagających dużej przepustowości. Ze względu na prostotę oraz lekkość jest bardzo niezawodny oraz niewymagający skomplikowanej implementacji co czyni go idealnym dla zastosowań Internetu Rzeczy.

Do implementacji połączenia do serwera MQTT użyta została biblioteka EspMQTTClient - zarządza połączeniem do serwera oraz wymianą danych między klientem oraz serwerem. Poniższy kod umożliwia podłączenie urządzenia do lokalnego albo zewnętrznego serwera MQTT:

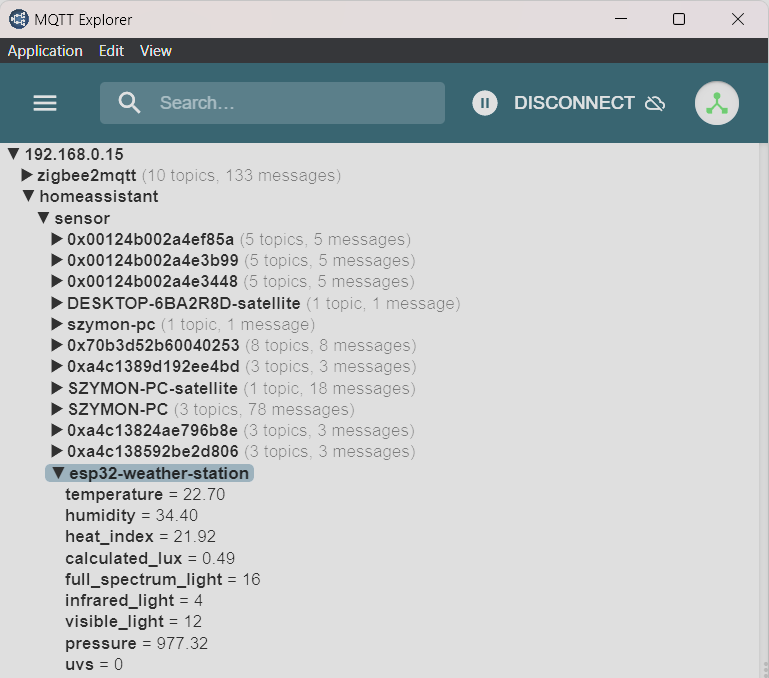
EspMQTTClient mqtt\_client(  
 "192.168.0.15", // host  
 1883, // port  
 "username", // username  
 "password", // password  
 mdns\_hostname // device name  
);

Do wysyłki danych do brokera MQTT potrzebny jest temat, na który informacja ma być wysłana oraz sama dana w formacie String - ciągu znaków. W przykładzie zmienna z typem float zmieniania jest na ciąg znaków String:

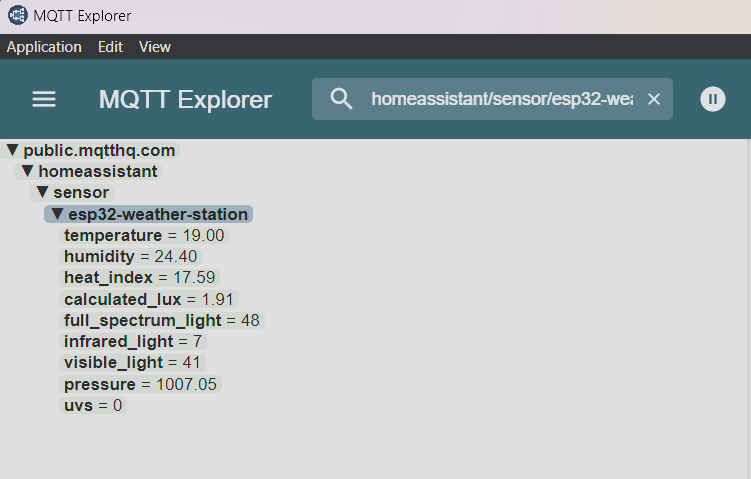
void send\_data\_to\_mqtt()  
{  
 String topic = "homeassistant/sensor/esp32-weather-station/temperature";  
 mqtt\_client.publish(topic, float\_to\_string(current\_reading\_data.temperature));  
}

### Testy połączenia MQTT

Możliwe jest połączenie sie do lokalnego serwera MQTT jeżeli urządzenie przetwarza danych delikatne lub dane mogą zostać wysyłane do publicznego serwera MQTT, co pozwoli na zdalny dostęp do danych, spoza sieci lokalnej.

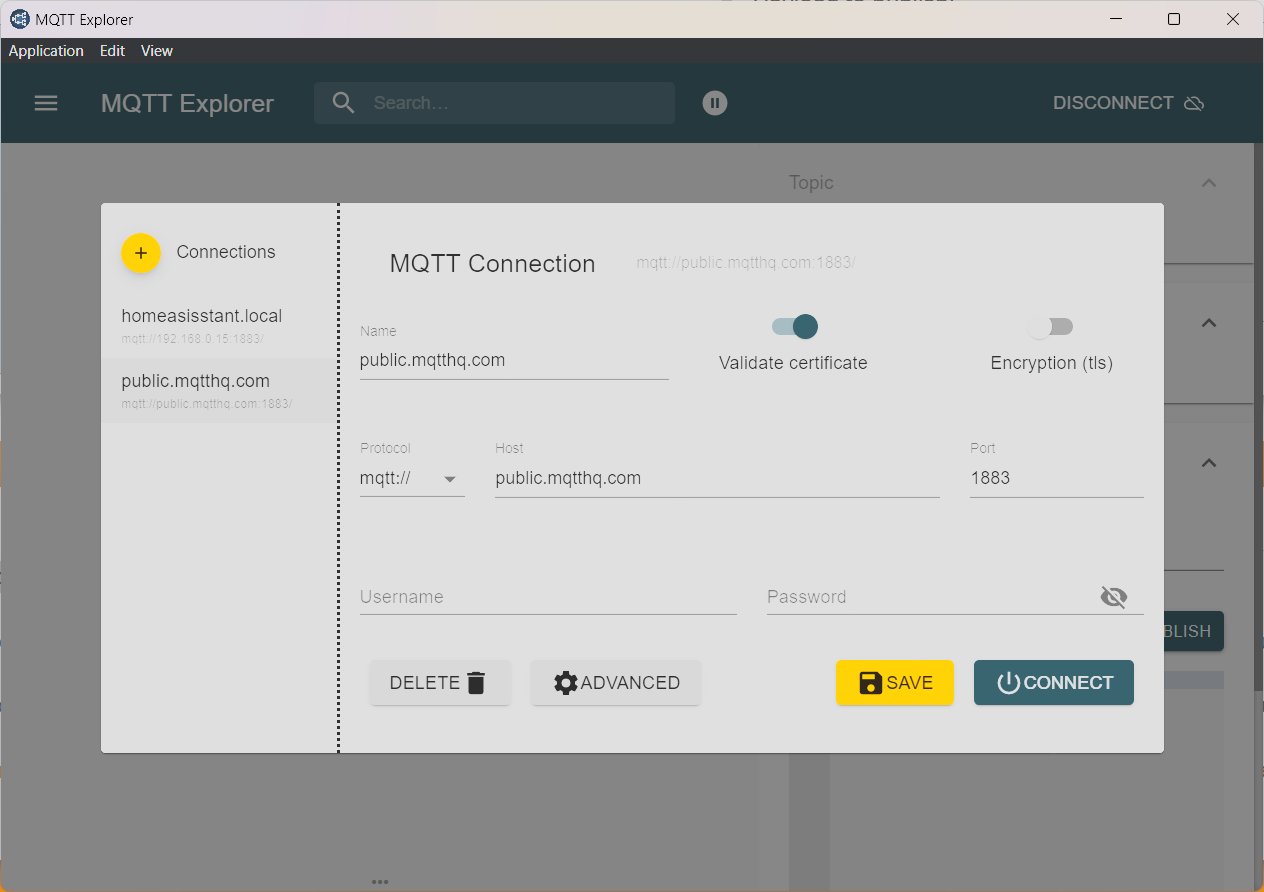


Test integracji z MQTT z serwerem lokalnym



Test integracji z MQTT z serwerem publicznym

Do podejrzenia danych przesyłanych na serwer MQTT został użyty program MQTT Explorer - jest to darmowy program pełniący rolę klienta MQTT, pozwalający na odczyt danych z tematów serwera MQTT oraz na nadawanie nowych wiadomości w zadanym temacie. Program jest dostępny na systemy operacyjne Windows, Linux oraz Mac



MQTT Explorer - podłączenie do serwera mqtthq.com

## Protokół mDNS

Protokół mDNS służy do rozwiązywania nazw urządzeń sieciowych do adresów IP - dzięki temu użytkownik ma możliwość połączenia z urządzeniem nie znając jego adresu - wystarczy nazwa urządzenia sieciowego. mDNS należy do rodziny protokołów Zeroconf (lub zero-config) - nie wymaga konfiguracja do pracy i jest protokół ten wspierany przez Windowsa (10+) oraz Linuxa (po instalacji programu Avahi).

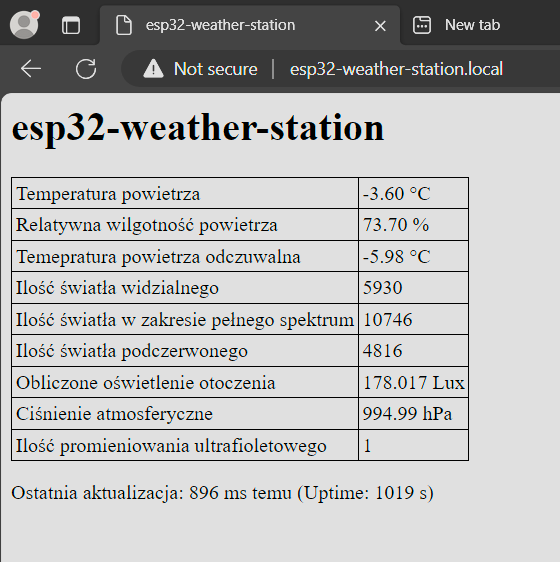
W projekcie zostanie protokół ten użyty, aby użytkownik nie musiał znać adresu IP urządzenia, aby sie z nim połączyć. Protokół jest konfigurowany za pomocą pliku nagłówkowego ESPmDNS.h. Aby skonfigurować usługę, aby mikrokontroler był możliwy do odnalezienia w sieci poprzez mDNS nalezy wystartować serwer mDNS oraz dodać nową usługę usługę z typem (w przykładzie http), protokołem (w przykładzie tcp) oraz portem pod którym usługa jest dostępna (w przykładzie: 80 - port serwera www z danymi, które udostępnia urządzenie).

Po prawidłowej konfiguracji oraz wpisaniu adresu  
http://esp32-weather-station.local/ w przeglądarce powinniśmy zobaczyć stronę www udostępnianą przez mikrokontroler.

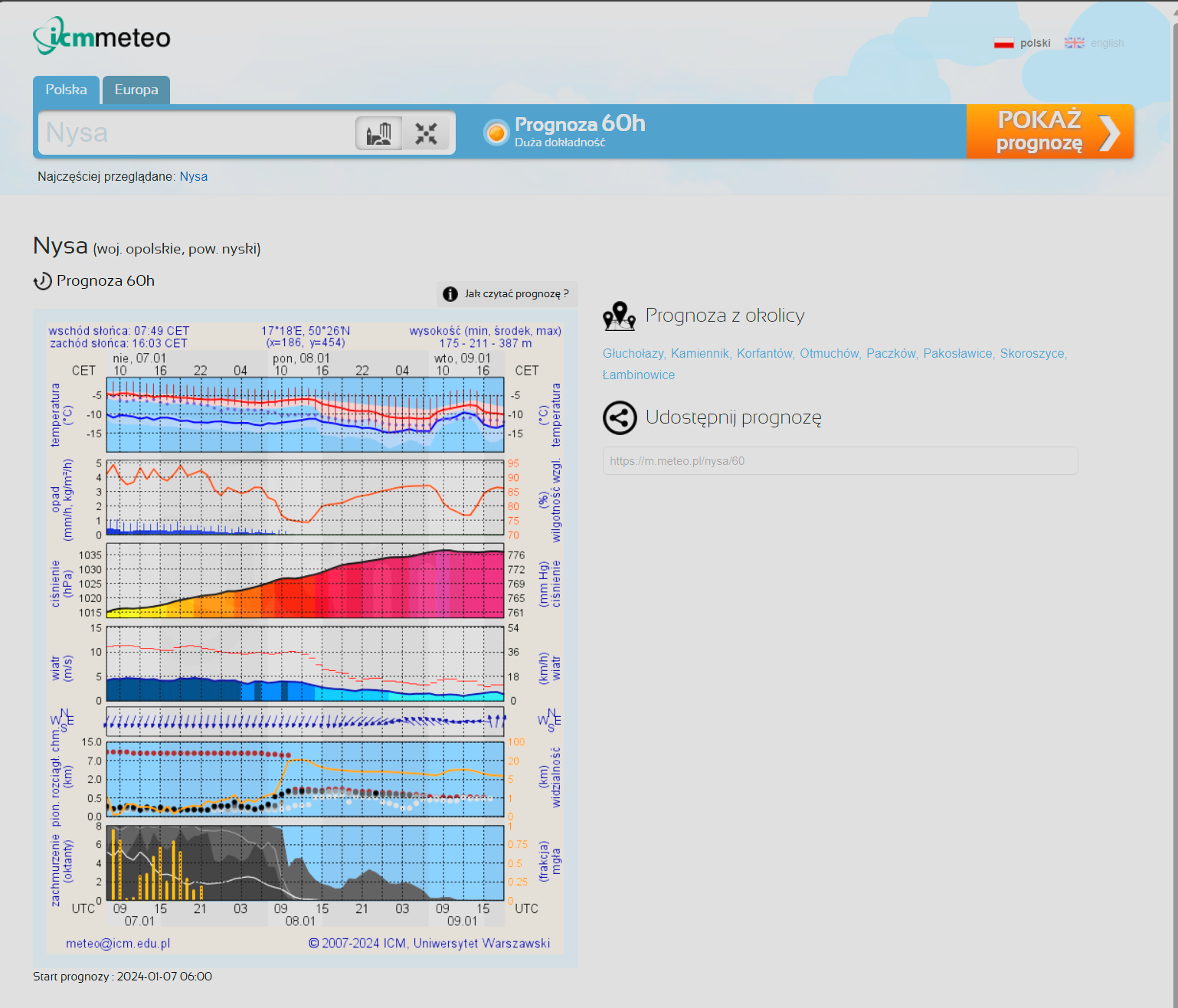
void setup()  
{  
 // esp32-weather-station.local  
 if (!MDNS.begin("esp32-weather-station")) {  
 Serial.println("Error setting up MDNS responder!");  
 while (1) {  
 delay(1000);  
 }  
 }  
  
 MDNS.addService("http", "tcp", 80);  
}

# Testy - porównanie z komercyjnymi danymi pogodowymi

Dane pogodowe dostarczane przez urządzenie będą bardziej dokładne lokalnie niż prognoza pogody dostępna w komercyjnych serwisach. Obrazek [3](#test-outside-site) wykazuje, że aktualna temperatura wynosi -3.6°C a względna wilgotność powietrza wynosi 73.7%. Dane udostępniane przez serwis meteo.pl dla tej samej co lokalizacja urządzenia wskazuje, że temperatura wynosi -5°C a względna wilgotność powietrza wynosi 85%.



Porównanie danych pogodowych - urządzenie



Porównanie danych pogodowych - meteo.pl

# Wnioski

Moduły bazujące na mikrokontrolerze ESP32 pozwalają na łatwą realizację projektów, a przez dużą różnorodność ekosystemu Arduino integracja z innymi wspieranymi komponentami jest prosta oraz daje ogromną wszechstronność w realizacji projektów.

W tej pracy zrealizowana została lokalna stacja pogodowa umożliwiająca odczyt danych pogodowych w sieci lokalnej za pomocą interfejsu WWW, interfejsu REST API. Dostęp do danych spoza sieci lokalnej możliwy jest po podłączeniu do dostępnego publicznie serwera MQTT - urządzenie zostaje bezpośrednio dostępne tylko w sieci lokalnej, a dane mogą zostać pobrane z brokera MQTT (na przykład: mqtthq.com). Lokalny dostęp do urządzenia jest jest bardzo prosty dzięki protokołowi mDNS - aby połączyć się z urządzeniem nie jest potrzebny adres IP, a wystarczy nazwa urządzenia w sieci, która jest statyczna.

Urządzenie zostało uruchomione w środowisko lokalnej sieci LAN - interfejs WWW, interfejs programistyczny dostępne są wyłączenie w sieci lokalnej, a dzięki możliwości połączenia do serwera MQTT możliwe jest odczytanie danych, które zbiera urządzenie poza lokalną siecią. Dane zostały poprawnie odczytane za pomocą interfejsu WWW, przez interfejs programistyczny REST API w formacie JSON oraz zostały poprawnie odczytane z tematu w publicznym brokerze MQTT.

Kolejnym etapem rozwoju projektu mogłaby być kolejna wersja interfejsu użytkownika, dodatkowe integracje z innymi systemami zewnętrznymi lub integracja czujników oraz mikrokontrolera na jednej płytce PCB. Podłączenie kolejnych czujników do urządzenia również nie powinno sprawiać problemu ze względu na dużą dostępność dodatkowych pinów WE/WYJ oraz otwartej natury całego ekosystemu Arduino. Kolejnym usprawnieniem projektu mogłoby być przystosowanie urządzenia do uruchomienia bez lokalnej sieci bezprzewodowej - do połączenia z Internetem mogłyby zostać wykorzystane modemy sieci GSM lub LoRA lub połączenie z siecią przewodową (sieć Ethernet).

# Literatura

# Spisy programów, tabel, fotografii

## Spis tabel

## Spis obrazków, fotografii

# Streszczenie

Praca inżynierska podejmuje temat "Konstrukcja stacji pogodowej opartej na mikrokontrolerze ESP32 z interfejsem użytkownika oraz API REST". Głównym celem pracy było przygotowanie prototypu urządzenia stacji pogodowej wraz z oprogramowaniem oraz test przydatności takiego urządzenia do celów domowych. Zaletami lokalnej stacji pogodowej jest wysoka dokładność oraz wysoka przewidywalność odczytów w porównaniu do danych zbieranych przez inne komercyjne serwisy. W ramach pracy został wybrany mikrokontroler oraz czujniki wykorzystane w urządzeniu, zostało skonstruowany prototyp urządzenia bazujący na wybranych komponentatch oraz zostało przygotowane oprogramowanie sterujące pracją mikrokontrolera. Duża różnorodność czujników, wysoka dostępność mikrokontrolerów oraz bogaty ekosystem programowy umożliwia szybką, tanią i prostą konstrukcje urządzeń bazujących na ESP32. Oprogrwamowanie urządzenia udostępnia interfejs www dla użytkownika, interfejs programowy REST API oraz integracje z brokerem MQTT. Urządzenie zostało uruchomione w środowisko lokalnej sieci LAN - interfejs WWW, interfejs programistyczny dostępne są wyłączenie w sieci lokalnej, a dzięki możliwości połączenia do serwera MQTT możliwe jest odczytanie danych, które zbiera urządzenie poza lokalną siecią. Przyszłymi usprawnieniami projektu mogłyby być: integracja z wiekszą ilością czujników, implementacja przewodowego połączenia sieciowego, połączeń GSM lub LoRa oraz usprawnienia w interfejsie użytkownika.