

**KIERUNEK: Informatyka**

**SPECJALNOŚĆ: Inżynieria oprogramowania**

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

**PRACA KWALIFIKACYJNA/PROJEKT INŻYNIERSKI**

**SZYMON UGLIS NR ALBUMU 16982**

**PROMOTOR/WYKŁADOWCA DR JĘDRZEJ UŁASIEWICZ**

Przyjmuję pracę i dopuszczam do egzaminu dyplomowego

...............................................................

*(data i podpis promotora)*

**Wrocław 2024**

**SPIS TRES´CI**

1. [Wste˛p 3](#_TOC_250036)
   1. [Wprowadzenie 3](#_TOC_250035)
   2. [Cel projektu oraz opis tworzonego rozwia˛zania 3](#_TOC_250034)
      1. [Funkcje urza˛dzenia 3](#_TOC_250033)
      2. [Doste˛p zdalny do urza˛dzenia 4](#_TOC_250032)
2. [Konfiguracja sprze˛towa 5](#_TOC_250031)
   1. [ESP-WROOM-32 6](#_TOC_250030)
      1. [Specyfikacja ESP-WROOM-32 7](#_TOC_250029)
   2. [Czujnik nate˛z˙enia s´wiatła - TSL25911 8](#_TOC_250028)
      1. [Specyfikacja TSL25911 8](#_TOC_250027)
      2. [Przykładowy program testu czujnika TSL25911 9](#_TOC_250026)
   3. [Czujnik temperatury i wilgotnos´ci powietrza - DHT22 9](#_TOC_250025)
      1. [Specyfikacja DHT22 10](#_TOC_250024)
      2. [Przykładowy program testu czujnika DHT22 11](#_TOC_250023)
   4. [Czujnik cis´nienia oraz temperatury - DPS310 11](#_TOC_250022)
      1. [Specyfikacja DPS310 12](#_TOC_250021)
      2. [Przykładowy program testu czujnika DPS310 13](#_TOC_250020)
   5. [Czujnik s´wiatła ultrafioletowego - LTR390 13](#_TOC_250019)
      1. [Specyfikacja LTR390 14](#_TOC_250018)
      2. [Przykładowy program testu czujnika LTR390 15](#_TOC_250017)
3. [S´rodowisko programowe 16](#_TOC_250016)
4. [Oprogramowanie 17](#_TOC_250015)
   1. [Protokół mDNS 17](#_TOC_250014)
   2. [Doste˛p do danych przez przegla˛darke˛ 19](#_TOC_250013)
   3. [Doste˛p do danych przez REST API 21](#_TOC_250012)
   4. [Doste˛p do danych diagnostycznych 21](#_TOC_250011)
   5. [Integracja z MQTT 23](#_TOC_250010)
5. [Testy 24](#_TOC_250009)
   1. [Interfejs WWW 24](#_TOC_250008)
   2. [Interfejs REST API 24](#_TOC_250007)
   3. [MQTT 25](#_TOC_250006)
   4. [Porównanie z komercyjnymi danymi pogodowymi 27](#_TOC_250005)
   5. [Doste˛p do danych za pomoca˛ MQTT i Home Assistant 28](#_TOC_250004)
6. [Wnioski 30](#_TOC_250003)
7. [Literatura 31](#_TOC_250002)
8. [Spisy programów, tabel, fotografii 32](#_TOC_250001)
9. [Streszczenie 33](#_TOC_250000)

# WSTE˛P

## Wprowadzenie

Dzie˛ki szerokiemu doste˛powi do internetu mamy łatwy doste˛p do danych pogodowych aktualnych jak i historycznych z całego s´wiata. Istnieje wiele serwisów, programów telewizyjnych, które aktualne dane pogodowe, prognozy pogody prezentuja˛ nam w przys- te˛pny sposób. Problemem z poleganiem na danych pogodowych z popularnych ser- wisów jest relatywnie niska dokładnos´c´ aktualnych warunków pogodowych jak i prog- noz pogody. Bowiem serwisy te nie moga˛ miec´ stacji pogowowych rozstawionych co

na przykład kilometr, aby dokładnos´c´ dla kaz˙dego potencjalnego zainteresowana była

wysoka. Dlatego polega sie˛ w duz˙ym stopniu na ogólnych danych z kilku, ba˛dz´ kilku-

natstu stacji w danym regionie, aby wykonac´

ekstrapolacje˛

dla aktualnych warunków

pogodowych dla całego regionu. Serwisy udoste˛pniaja˛ce dane pogodowe dla lokalizacji sa˛ tylko przybliz˙eniem faktycznych stanu jaki znajduje sie w danym miejscu.

## Cel projektu oraz opis tworzonego rozwia˛zania

Celem projektu jest zbadanie przydatnos´ci modułu ESP32 oraz kompatybilnych kom- ponentów do celów kolekcji danych pogodowych. Rezultatem be˛dzie utworzenie urza˛dzenia opartego na systemie wbudwanym pozwalaja˛cym na wielofunkcyjny pomiar parametrów pogodowych oraz udoste˛pnianie ich za pomoca˛ witryny [www.](http://www/)

Urza˛dzenie, be˛dzie udoste˛pniało równiez˙ interfejs programistyczny API REST, które be˛dzie umoz˙liwiało integracje oraz dalszy rozwój projektu (np. integracja z systemem home assitant czy innymi urza˛dzeniami Internetu rzeczy (IoT)).

### Funkcje urza˛dzenia

* + - * Pomiar oraz kalkulacja danych pogodowych na podstawie danych wejs´ciowych z czujników
      * Udoste˛pnienie i agregacja danych na stronie www
      * Udoste˛pnienie interfejsu programistycznego REST
      * Moz˙liwos´c´ zmiany parametrów odczytów sensorów

### Doste˛p zdalny do urza˛dzenia

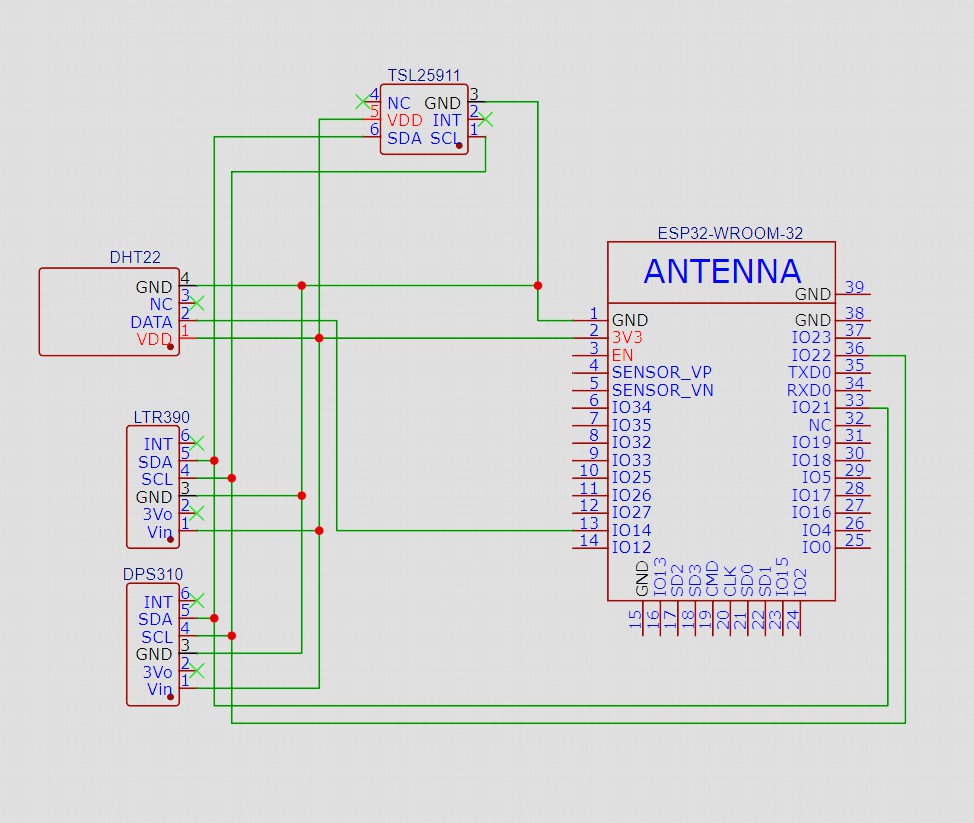
Urza˛dzenia ze s´wiata IoT sa˛ najcze˛s´ciej urza˛dzeniami lokalnymi - bez publicznego adresu IP i/lub bez moz˙liwos´ci zdalnego doste˛pu. Ze wzgle˛du na niska˛ moc takich urza˛dzen´, ubogie oprogramowanie, w które takie urza˛dzenia sa˛ wyposaz˙one bardzo cze˛sto nie zawieraja˛ wielu waz˙nych zabezpieczen´ przeciwko potencjalnym atakom. Dodatkowo dane udoste˛pnianie przez urza˛dzenia IoT z bardzo cze˛sto sa˛ danymi które nie powinny byc doste˛pne publicznie (np. dane o obecnos´ci osób w pomieszczeniu lub czy zamek od drzmi jest zamknie˛ty).

Zalecanym jest, aby doste˛p do urza˛dzen´ IoT był całkowicie lokalny (najlepiej do-

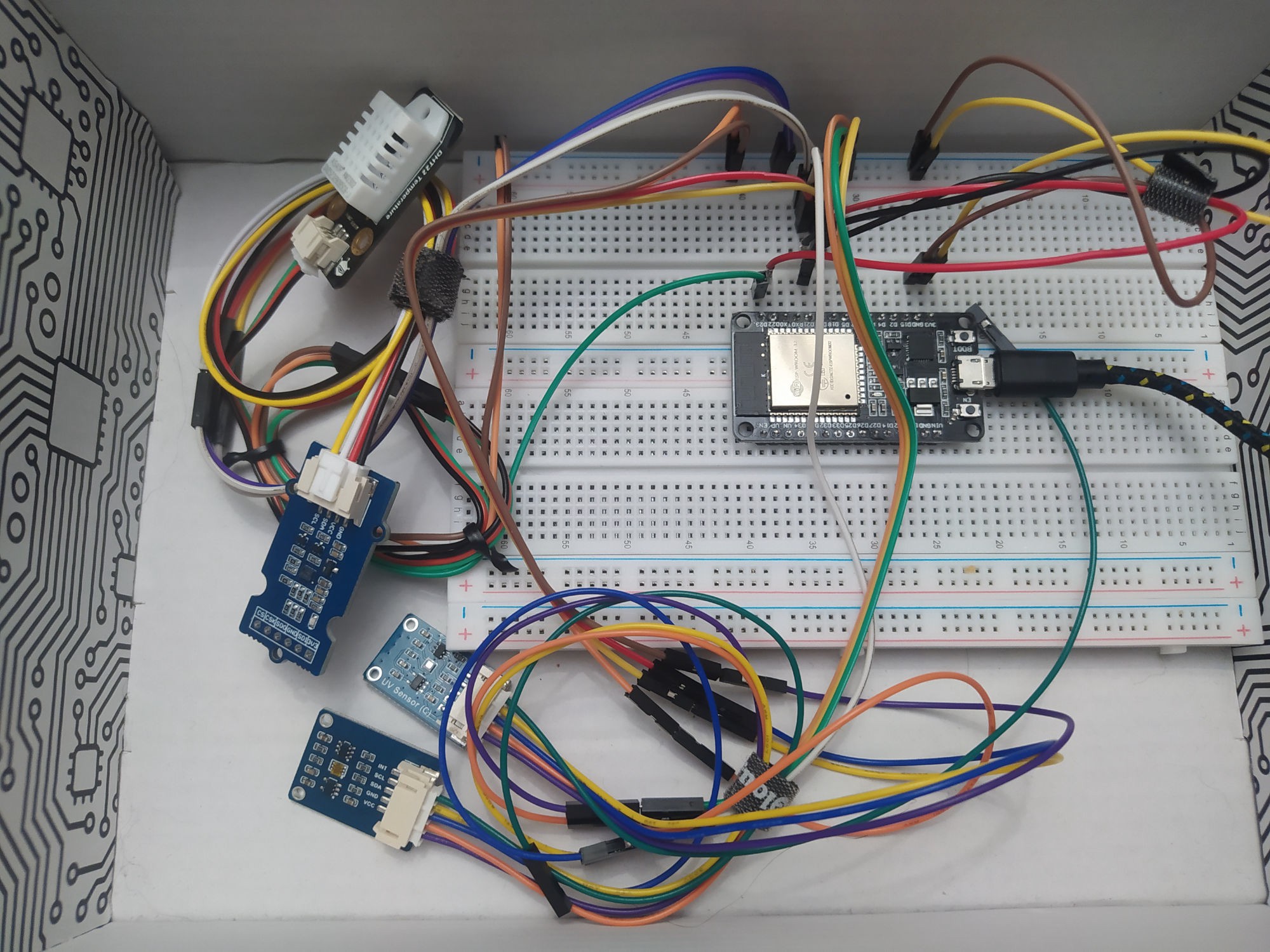
datkowo zablokowac´ doste˛p do internetu dla tych urza˛dzen´), a zbieraniem i udoste˛pni- aniem danych publiczych było realizowane przez oprogramowanie dedykowane do tego celu. Aby umoz˙liwic´ doste˛p do danych zbieranych przez tworzone urza˛dzenie be˛dzie im- plementowac´ moz˙liwos´c´ konfiguracji wysyłania danych za pomoca˛ MQTT. Dane zagre- gowane przez MQTT dalej moga˛ byc´ bezpiecznie filtrowane, przetwarzane i udoste˛pniane publicznie przez oprogramowanie do tego przystosowane.

# KONFIGURACJA SPRZE˛TOWA

* Mikrokontroler - ESP-WROOM-32
* Czujnik nate˛z˙enia s´wiatła - TSL25911
* Czujnik temperatury i wilgotnos´ci powietrza - DHT22
* Czujnik cis´nienia oraz temperatury - DPS310
* Czujnik s´wiatła ultrafioletowego - LTR390



Obrazek 1: Schemat urza˛dzenia



Obrazek 2: Fotografia prototypu urza˛dzenia

## ESP-WROOM-32

ESP-WROOM-32 (albo ESP32-WROOM-32) to mikrokontroler ze zintegrowanym WIFI oraz bluetooth. Nadaje sie˛ do szerokiej gamy zastosowan´: od kontrolera czujników z niskim poborem energii do zaawansowanych zadan´ enkodowania sygnałów głosowych

czy muzyki. Moduły bazuja˛ce na rdzeniach ESP32 zyskały popularnos´c´ ze wzkle˛du

na zaintegrowana˛ obsługe˛ WIFI, niskim kosztom oraz bogatej dokumentacji oraz wielu moz˙liwych integracji.

Mikrokontroler znajdzie zastosowanie w prostych jak i bardziej skomplikowanych projektach, dzie˛ki dwóm rdzenion, które moga˛ byc´ kontrolowane osobno, szerokich moz˙li- wos´ciach podła˛czenia urza˛dzen´ perfyferyjnych (wsparcie dla: I2C, UART, SPI oraz inne). ESP-WROOM-32 składa sie˛ z mikrokontrolera ESP32-D0WDQ6 posiadaja˛cego dwa rdzenie, które pozwaja˛ na prace od 80MHz do 240 MHz. Procesor równiez˙ posiada ko- procesor o niskim poborze mocy, który moz˙e zostac´ uz˙yty zamiast głównych rdzeni w

przypadku kiedy nie jest potrzebna duz˙a moc obliczeniowa.

Integracja WIFI, Bluetooth i Bluetooth LE do mikrokontrolera pozwala na rozszerze- nie moz˙liwych zastosowan´, gdzie nie jest moz˙liwe podła˛czenie do sieci w konwencjon- alny spoób za pomoca˛ kabla. Komunikacja bezprzewodowa równiez˙ umoz˙liwa całkowicie zdalne zastosowania z wykorzystaniem akumulatorów oraz paneli słonecznych do zasila- nia mikrokontrolera.

### Specyfikacja ESP-WROOM-32

|  |  |
| --- | --- |
| Element | Specyfikacja |
| Procesor | 2 rdzenie (80-240MHz) |
| WIFI | 802.11 b/g/n (802.11n: przepustowos´c´ 150 Mbps) |
| Bluetooth | Bluetooth v4.2 i Bluetooth LE |
| Pamie˛c´ SPI Flash | 4MB |
| Napie˛cie zasilania | 3.0 V - 3.6 V |
| Zuz˙ycie pra˛du | S´rednie: 80 mA |

Tabela 1: Najwaz˙niejsze specyfikacje ESP-WROOM-32

Do realizacji projektu zostata wybrany mikrokontroler ESP-WROOM-32 ze wzgle˛du

na:

* + - * bogate moz˙liwos´ci podła˛czenia urza˛dzen´ perfyferyjnych
      * wbudowane wspracie poła˛czen´ WIFI
      * duz˙a˛ ilos´c´ pamie˛ci flash
      * niewielka˛ cene˛ w porównaniu do oferowanych specyfikacji
      * łatwa˛ doste˛pnos´c´

Potencjalnymi alternatywami, które nie zostały wybrane do realizacji projektu sa˛:

* + - * Arduino Uno R3 (brak wbudowanej obsługi poła˛czen´ sieciowych)
      * ESP8266 (wolniejszy procesor, mniejsza ilos´c´ pamie˛ci flash, mniejsza ilos´c´ wyprowadzen´ do podła˛czenia urza˛dzen´ perfyferyjnych)
      * Olimex ESP32-POE[10] (wbudowana obsługa poła˛czenie przewodowego ethernet, mniejsza ilos´c´ wyprowadzen´ do podła˛czenia urza˛dzen´ perfyferyjnych)
      * LILYGO T-SIM7000G[3] (wbudowana obsługa LTE, mniejsza ilos´c´ wyprowadzen´ do podła˛czenia urza˛dzen´ perfyferyjnych, wysoka cena)

W finalnym produkcie w zalez˙nos´ci od przeznaczenia uz˙ycie ła˛cznos´ci bezprzewodowej moz˙e byc´ niewskazane (daleka odległos´c´ od punktu doste˛powego WIFI, duz˙a ilos´c´ przeszkód).

Uz˙ycie innych wersji ESP32 z wbudowana˛ obsługa˛ przewodowego poła˛czenia sieciowego lub moz˙liwos´cia poła˛czenia do sieci przez LTE, moz˙e byc´ wymagane - jednak w ramach realizacji zakresu projektu wybór padł na łatwiej doste˛pny moduł z ła˛cznos´cia˛ bezprze- wodowa˛ WIFI.

## Czujnik nate˛z˙enia s´wiatła - TSL25911

TSL25911 to czujnik s´wiatła z odpowiedzia˛ podobna˛ do ludzkiego oka. Posiada czułos´c´ 188 uLux oraz zakres dynamiczny 1 do 600,000,000. W porównaniu do innych tan´szych rozwia˛zan´ bazuja˛cych na fotokomórkach CdS, daje o wiele bardziej precyzujne rezultaty, z moz˙liwos´cia˛ zmiany czułos´ci i rozdzielczos´ci w locie.

### Specyfikacja TSL25911

|  |  |
| --- | --- |
| Element | Specyfikacja |
| Rozdzielczos´c´ 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) |

Tabela 2: Specyfikacja TSL25911

### Przykładowy program testu czujnika TSL25911

1 *#include <Wire.h>*

2

3 *#include "Adafruit\_TSL2591.h"*

4

5 Adafruit\_TSL2591 tsl = Adafruit\_TSL2591(2591);

6

7 **void** setup() {

8 Wire.begin();

9 Serial.begin(115200);

10

11 tsl.setGain(TSL2591\_GAIN\_MED);

12 tsl.setTiming(TSL2591\_INTEGRATIONTIME\_300MS);

13 }

14

15 **void** loop() {

16 **uint32\_t** lum = tsl.getFullLuminosity();

17 **uint16\_t** ir = lum >> 16;

18 **uint16\_t** full = lum & 0xFFFF;

19

20 Serial.print(F("TSL2591 sensor data - Calculated Lux:

*'→* '"));

21 Serial.print(tsl.calculateLux(full, ir));

22 Serial.print(F("' Full spectrum light: '"));

23 Serial.print(full);

24 Serial.print(F("' Infrared light: '"));

25 Serial.print(ir);

26 Serial.print(F("' Visible light: '"));

27 Serial.print(full - ir);

28 Serial.println(F("'"));

29 }

Program 1: Test czujnika TSL25911

## Czujnik temperatury i wilgotnos´ci powietrza - DHT22

Czujnik DHT22 składaja sie˛ z dwóch cze˛s´ci: pojemnos´ciowego czujnika wilgotnos´ci i termistora. Wewna˛trz znajduje sie˛ równiez˙ bardzo prosty chip, który dokonuje konwersji sygnału analogowego na cyfrowy i wysyła sygnał cyfrowy z temperatura˛ i wilgotnos´cia˛. Sygnał cyfrowy jest łatwy do odczytania za pomoca˛ dowolnego mikrokontrolera.

Czujnik ten cechuje sie bardzo niska˛ cena˛ oraz bardzo duz˙a˛ dokładnos´cia˛ w stosunku

do ceny.

Potencjalnymi alternatywami, które mogły byc´ wykorzystane to:

* DHT11 (niz˙sza dokładnos´c´)
* AHT20 (niz˙sza dokładnos´c´ odczytu temperatury oraz podobna dokładnos´c´ odczytu wilgotnos´ci, doste˛pny tylko jeden adres I2C)

Został wybrany DHT11 ze wzgle˛du na prostote podła˛czenia (jedno wyjs´cie cyfrowe) co sprawia, z˙e moz˙liwe jest podła˛czenie kilku czujników do jednego mikrokontrolera (co nie moz˙liwe jest w przypadku AHT20, bez wykorzystania na przykład drugigo interfejsu I2C). Czujnika DHT22 cechuje sie˛ równiez˙ niskim kosztem oraz wysoka˛ dokładnos´c´ w stosunku do ceny czujnika, co zadecydowało o finalnym wyborze tego produku.

### Specyfikacja DHT22

|  |  |
| --- | --- |
| Element | Specyfikacja |
| Zakres oraz dokładnos´c´ odczytu temperatury | -40 to 80 °C (dokładnos´c´: 0.5°C) |
| Zakres oraz dokładnos´c´ odczytu wilgotnos´ci | 0 to 100 RH% (dokładnos´c´: 2-5 %) |
| Maksymalna cze˛stotliwos´c´ odczytu danych | 0.5Hz |

Tabela 3: Specyfikacja DHT22

### Przykładowy program testu czujnika DHT22

|  |  |  |
| --- | --- | --- |
| 1 | *#include "DHT.h"* |  |
| 2 |  |  |
| 3 | *#define DHTPIN 23* |  |
| 4 | *#define DHTTYPE DHT22* |  |
| 5 |  |  |
| 6 | DHT dht(DHTPIN, DHTTYPE); |  |
| 7 |  |  |
| 8 | **void** setup() { |  |
| 9 | Serial.begin(115200); |  |
| 10 | dht.begin(); |  |
| 11 | } |  |
| 12 |  |  |
| 13 | **void** loop() { |  |
| 14 | **auto** humidity = dht.readHumidity(); |  |
| 15 | **auto** temperature = dht.readTemperature(); |  |
| 16 |  |  |
| 17 | **if** (isnan(humidity) || isnan(temperature)) | { |

18 Serial.println(F("Failed to read from DHT

*'→* sensor!"));

19 **return**;

20 }

21

22

23 Serial.print(F("DHT22 sensor data - Temp: '"));

24 Serial.print(temperature);

25 Serial.print(F("' Humidity: '"));

26 Serial.print(humidity);

27 Serial.print(F("' Heat index: '"));

28 Serial.print(dht.computeHeatIndex(temperature,

*'→* humidity, false));

29 Serial.println(F("'"));

30 }

Program 2: Test czujnika DHT22

## Czujnik cis´nienia oraz temperatury - DPS310

Czujnik DP310 słuz˙y do dokładnego pomiaru cis´nienia atmosferycznego oraz posi-

ada dodatkowy czujnik temperatury. Poasiada wysoka˛ dokładnos´c´ obsolutna˛ +-1 hPa

oraz pracuje w zakresie od 300 do 1200 hPa. Czujnik cechuje sie niska˛ cena˛, wysoka˛ dokładnos´cia˛ w stosunku do ceny, oraz posiada dodatkowy czujnik temperatury, który

moz˙e słuz˙yc´ jako preferowana metoda pomiaru temperatury, albo jako dodatkowy punkt w danych pomiarowych.

Potencjalnymi alternatywami, które mogły byc´ wykorzystane to:

* EE895 (niz˙sza dokładnos´c´ pomiaru cis´nienia (700-1100 hPa, wysoka cena ze wzgle˛du na obecnos´c´ czujnika C02))
* BMP180 (niz˙sza dokładnos´c´ pomiaru cis´nienia)

Czujnik został wybrany z powodu konkurencyjnej ceny, duz˙ej doste˛pnos´ci oraz wysok- iej dokładnos´ci pomiaru w stosunku do ceny produktu.

### Specyfikacja DPS310

|  |  |
| --- | --- |
| Element | Specyfikacja |
| Zakres oraz dokładnos´c´ odczytu temperatury | -40 to 85 °C (dokładnos´c´: 1°C) |
| Zakres odczytu cis´nienia atmosferycznego | 300 do 1200 hPa |
| Dokładnos´c´ absolutna odczytu cis´nienia atmosferycznego | 1 hPa |
| Dokładnos´c´ dynamiczna odczytu cis´nienia atmosferycznego | 0.002 hPa |

Tabela 4: Specyfikacja DPS310

### Przykładowy program testu czujnika DPS310

1 *#include <Wire.h>*

2

3 *#include "Adafruit\_DPS310.h"*

4

5 Adafruit\_DPS310 dps;

6

7 **void** setup() {

8 Serial.begin(115200);

9 Wire.begin();

10

11 dps.begin\_I2C();

12

13 dps.configurePressure(DPS310\_64HZ, DPS310\_64SAMPLES);

14 dps.configureTemperature(DPS310\_64HZ,

*'→*

15 }

16

DPS310\_64SAMPLES);

17 **void** loop() {

18 sensors\_event\_t pressure\_event;

19 sensors\_event\_t temperature\_event;

20 dps.getEvents(&temperature\_event, &pressure\_event);

21

22 Serial.print(F("DPS310 sensor data - Pressure: '"));

23 Serial.print(pressure\_event.pressure);

24 Serial.print(F("' Temperature: '"));

25 Serial.print(temperature\_event.temperature);

26 }

Program 3: Test czujnika DPS310

## Czujnik s´wiatła ultrafioletowego - LTR390

Czujnik LTR390 jest urza˛dzeniem o bardzo niskim koszcie oraz wysokiej dokład- nos´ci na s´wiatło w zakresie od 300nm do 350 nm. Czujnik posiada dwa sensory - je- den do s´wiatła UV, drugi do s´wiatła widzialnego co pozwala na precyzyjne okres´lenie ilos´ci padaja˛cego na czujnik s´wiatła UV. W porównaniu do innych czujników oferuje bezpos´redni odczyt wartos´ci s´wiatła UV z sensora (czujnik SI1145 oblicza ilos´c´ s´wiatła UV z całkowitego odczytu padaja˛cego s´wiatła˛ we wszystkich zakresach) oraz pozwala na odczyt dancych za pomoca˛ interfejsu I2C (bez koniecznos´ci posiadania interfejsów SPI

czy posiadania dodatkowych konwerterów analogowo-cyfrowcych) Potencjalnymi alternatywami, które mogły byc´ wykorzystane to:

* SI1145 (niz˙sza dokładnos´c´ wartos´c´ oblicznana)

pomiaru s´wiatła UV - brak bezpos´reniego odczytu -

* GUVA-S12SD (wymagany konwerter analogow-cyfrowy do odczytu danych)

Czujnik został wybranu z powodu wysokiej dokładnos´ci (bezpos´redni odczyt wartos´ci padaja˛cego s´wiatła UV) oraz prostoty podła˛czenia czujnika do mikrokontrolera (inter- fejs I2C). Bardzo niska cena oraz moz˙liwos´c´ odczytu nate˛z˙enia s´wiatła (w lux) jest do- datkowym plusem.

### Specyfikacja LTR390

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

Tabela 5: Specyfikacja LTR390

### Przykładowy program testu czujnika LTR390

1 *#include <Wire.h>*

2

3 *#include "Adafruit\_LTR390.h"*

4

5 Adafruit\_LTR390 ltr = Adafruit\_LTR390();

6

7 **void** setup() {

8 Serial.begin(115200);

9 Wire.begin();

10

11 ltr.begin();

12

13 ltr.setMode(LTR390\_MODE\_UVS);

14 ltr.setGain(LTR390\_GAIN\_3);

15 ltr.setResolution(LTR390\_RESOLUTION\_16BIT);

16 }

17

18 **void** loop() {

19 Serial.print(F("LTR390 sensor data - UVS: '"));

20 Serial.print(ltr.readUVS());

21 Serial.println(F("'"));

22 }

Program 4: Test czujnika LTR390

# S´RODOWISKO PROGRAMOWE

Do implementacji systemu wbudowanego wykorzystane zostana˛:

* Je˛zyk C do utworzenia oprogramowania mikrokontrolera
* Je˛zyk HTML oraz CSS do utworzenia interfejsu www
* S´rodowisko programistyczne Arduino Studio w wersji 2.2.1

# OPROGRAMOWANIE

## Protokół mDNS

Protokół mDNS słuz˙y do rozwia˛zywania nazw urza˛dzen´ sieciowych do adresów IP - dzie˛ki temu uz˙ytkownik ma moz˙liwos´c´ poła˛czenia z urza˛dzeniem nie znaja˛c jego adresu

- wystarczy nazwa urza˛dzenia sieciowego. mDNS nalez˙y do rodziny protkołów Zeroconf (lub zero-config)[12] - nie wymaga konfiguracja do pracy i jest protkół ten wspierany przez Windowsa (10+) oraz Linuxa (po instalacji programu Avahi).

W projekcie zostanie protokół ten uz˙yty, aby uz˙ytkownik nie musiał znac´ adresu

IP urza˛dzenia, aby sie z nim poła˛czyc´. Protokół jest konfigurowany za pomoca˛ pliku nagłówkowego ESPmDNS.h. Aby skonfigurowac´ usługe˛, aby mikrokontroler był moz˙liwy do odnalezienia w sieci poprzez mDNS nalezy wystartowac´ serwer mDNS oraz dodac´

nowa˛ usługe˛

usługe˛

z typem (w przykładzie http), protkołem (w przykładzie tcp)

oraz portem pod którym usługa jest doste˛pna (w przykładzie: 80 - port serwera www z danymi, które udoste˛pnia urza˛dzenie).

Po prawidłwej konfiguracji oraz wpisaniu adresu

<http://esp32-weather-station.local/>w przegla˛darce powinnis´my zobaczyc´ strone˛ www udoste˛pniania˛ przez mikrokontroler.

1 **void** setup()

2 {

3 *// esp32-weather-station.local*

4 **if** (!MDNS.begin("esp32-weather-station")) {

5 Serial.println("Error setting up MDNS

*'→* responder!");

6 **while** (1) {

7 delay(1000);

8 }

9 }

10

11 MDNS.addService("http", "tcp", 80);

12 }

Program 5: Przykład konfiguracji usługi mDNS

Do implementacji serwera http została uz˙yta biblioteka ESP32 HTTPS Server [1].

1 *#include <HTTPServer.hpp>*

2 *#include <HTTPRequest.hpp>*

3 *#include <HTTPResponse.hpp>*

4

5 httpsserver::HTTPServer http\_server =

*'→* httpsserver::HTTPServer();

6

7 **void** handle\_api\_request(

8 httpsserver::HTTPRequest \*req,

9 httpsserver::HTTPResponse \*res

10 );

11 **void** handle\_root\_request(

12 httpsserver::HTTPRequest \*req,

13 httpsserver::HTTPResponse \*res

14 );

15

|  |  |  |
| --- | --- | --- |
| 16 **void** setup(**void**) | |  |
| 17 { | |
| 18 httpsserver::ResourceNode \*nodeRoot  *'→* httpsserver::ResourceNode( | | = new |
| 19 "/", | |  |
| 20 "GET", | |  |
| 21 | &handle\_root\_request |  |
| 22 | ); |  |
| 23 | httpsserver::ResourceNode \*nodeApi =  *'→* httpsserver::ResourceNode( | new |
| 24 | "/api", |  |
| 25 | "GET", |  |
| 26 | &handle\_api\_request |  |
| 27 | ); |  |
| 28 | http\_server.registerNode(nodeRoot); |  |
| 29 | http\_server.registerNode(nodeApi); |  |
| 30 | http\_server.start(); |  |
| 31 | **if** (http\_server.isRunning()) { |  |
| 32 | Serial.println("Server ready."); |  |
| 33 | } |  |
| 34 | } |  |
| 35 |  |  |
| 36 | **void** loop(**void**) |  |
| 37 | { |  |
| 38 | http\_server.loop(); |  |
| 39 | } |  |

Program 6: Wycinek kodu konfiguruja˛cego serwer http

Biblioteka pozwala na proste definiowanie adresów podstron, wraz z przyjmowanym typem zapytania http. Gdy serwer odbierze poła˛czenie od klienta z prawidłowym adresem, wykonuje metode zarejestrowana˛ w obiekcie serwera, w przeciwnym wypadku zwraca odpowiedz´ z kodek 404 Not Found.

## Doste˛p do danych przez przegla˛darke˛

Urza˛dzenie udoste˛pnia strone˛ www z odczytami czujników w prostym formacie dla odczytu przez człowieka. Przy pomocy HTML oraz CSS prezentowana jest tabela ze wszystkimi danymi, które udoste˛pnia urza˛dzenie, informacja jaki czas temu dane były aktulizowane oraz jak długo urza˛dzenie pracuje.

Aktualne dane zebrane przez urza˛dzenie sa˛ wstrzykiwane do szablonu http za po- moca˛ 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 maja˛ byc´ wstrzyknie˛te. Naste˛pnie gotowy plik html jest wysyłany do uz˙ytkown- ika.

1 **void** handle\_root\_request(

2 httpsserver::HTTPRequest \*req,

3 httpsserver::HTTPResponse \*res

4 ) {

5 res->setHeader("Content-Type", "text/html");

6

7 **char** buffer[2000];

8 sprintf(

9 buffer,

10 html\_template,

11 mdns\_hostname,

12 mdns\_hostname,

13 current\_reading\_data.temperature,

14 current\_reading\_data.humidity,

15 current\_reading\_data.heat\_index,

16 current\_reading\_data.visible\_light,

17 current\_reading\_data.full\_spectrum\_light,

18 current\_reading\_data.infrared\_light,

19 current\_reading\_data.calculated\_lux,

20 current\_reading\_data.pressure,

21 current\_reading\_data.uvs,

22 sensor\_read\_delay,

23 calculate\_uptime()

24 );

25

26 res->print(buffer);

27 }

Program 7: Kod obsługuja˛cy interfejs uz˙ytkownika

Szablon pliku html jest przechowywany w pamie˛ci programowej programu w stałej html\_template. Zaleta˛ takiego rozwia˛zania jest prostota implementacji oraz szy- bkos´c´ działania - dane sa˛ zawsze gotowe do wykrzystania. Alternatwywnie moz˙na byłoby

wykorzystac´

pamie˛c´

SPIFFS urza˛dzenia do przechowywania szablonu html - byłoby

to bardziej elastyczne rozwia˛zanie, które nie wymagało by ponownej kompilacji pro- gramu, aby zmienic´ szablon html. Z drugiej strony rozwia˛zanie to mogłoby powodowac´ zwie˛kszony narzut przy generowaniu odpowiedzi przy kaz˙dym zapytaniu. Ze wzle˛du na prostote˛ pierwszego rozwia˛zania oraz potencjalne problemy z wydajnos´cia˛ drugiego rozwia˛zania zdecydowałem sie załadowac´ szablon do kodu programu w tej wersji opro- gramowania. Do deklaracji zawartos´ci zmiennej jest uz˙yta deklaracja literał cia˛gu znaków

(string literal), aby w prosty sposób było moz˙liwe zawarcie w s´rodku zmiennej znaków spcjalnych, znaków niedozwolonych oraz w celu Udoste˛pnienia moz˙liwos´ci deklaracji wieloniowego cia˛gu znaków.

1 **const char** html\_template[] = R"###(

2 <html lang="en">

3 <head>

4 ...

5 )###";

Program 8: Kod deklaracji szablonu html

Przyszłym usprawnieniem urza˛dzenia mogłoby byc´ sprawdznie wydajnos´ci pamie˛ci SPIFFS do przechowywania szablonów html, aby oszczedzic´ na pamie˛ci programowej.

## Doste˛p do danych przez REST API

Urza˛dzenie udoste˛pnia dane równiez˙ poprzez REST API w postaci JSON - umoz˙liwia to łatwa˛ integracje˛ urza˛dzenia z innymi narze˛dziami, albo umoz˙liwia stworzenie własnych integracji.

## Doste˛p do danych diagnostycznych

Urza˛dzenie udoste˛pnia dane, które moga˛ słuzyc´ do debugowania urza˛dzenia za po- moca˛ interfejsu szeregowego. Podczas procesu uruchamiania na port szeregowy wysyłane sa˛ dane informuja˛ce o przebiegu procesu uruchamiania urza˛dzenia, poła˛czeniu do siecie bezprzewodowej oraz po poprawnym uruchomieniu, aktualnie zebrane danye pogodowe.

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

1 **void** send\_debug\_info\_to\_serial()

2 {

3 Serial.print(F("DHT22 sensor data - Temp: '"));

4 Serial.print(current\_reading\_data.temperature);

5 Serial.print(F("' Humidity: '"));

6 Serial.print(current\_reading\_data.humidity);

7 Serial.print(F("' Heat index: '"));

8 Serial.print(current\_reading\_data.heat\_index);

9 Serial.println(F("'"));

10

11 Serial.print(F("TSL2591 sensor data - Calculated

*'→* Lux: '"));

12 Serial.print(current\_reading\_data.calculated\_lux);

13 Serial.print(F("' Full spectrum light: '"));

14

*'→* Serial.print(current\_reading\_data.full\_spectrum\_light);

15 Serial.print(F("' Infrared light: '"));

16 Serial.print(current\_reading\_data.infrared\_light);

17 Serial.print(F("' Visible light: '"));

18 Serial.print(current\_reading\_data.visible\_light);

19 Serial.println(F("'"));

20

21 Serial.print(F("DPS310 sensor data - Pressure: '"));

22 Serial.print(current\_reading\_data.pressure);

23 Serial.println(F("hPa'"));

24

25 Serial.print(F("LTR390 sensor data - UVS: '"));

26 Serial.print(current\_reading\_data.uvs);

27 Serial.println(F("'"));

28

29 Serial.println(F(""));

30 }

Program 9: Kod doste˛pu do danych diagnostycznych

Jako, z˙e powyz˙sza funkcja operuje na duz˙ej ilos´c´ tekstu w sposób cały, zostało zas- tosowana optymalizacja w postaci uz˙ycia funkcji F - funkcja ta sprawia, z˙e dany łan´cuch znaków nie jest kopiowany do pamie˛ci PSRAM mikrokontrolera przed wysłaniem na in- terfejs szeregowy, a dane sa˛ bez´pos´rednio kopiowane z programu do strumienia, który wysyłane dane na interfejs.

## Integracja z MQTT

Urza˛dzenie dodatkowo udoste˛pnia moz˙liwos´c´ poła˛czenia do serwera MQTT - co umoz˙li- wia bezpieczne udoste˛pniane danych do serwera zewne˛trzengo. Do implementacji poła˛czenia do serwera MQTT uz˙yta została biblioteka EspMQTTClient[11] - zarza˛dza poła˛cze- niem do serwera oraz wymiana˛ danych mie˛dzy klientem oraz serwerem.

Poniz˙szy kod umoz˙liwia podła˛czenie urza˛dzenia do lokalnego albo zewne˛trzego ser- wera MQTT:

EspMQTTClient mqtt\_client( "192.168.0.15", *// host*

1883, *// port* "username", *// username* "password", *// password*

mdns\_hostname *// device name*

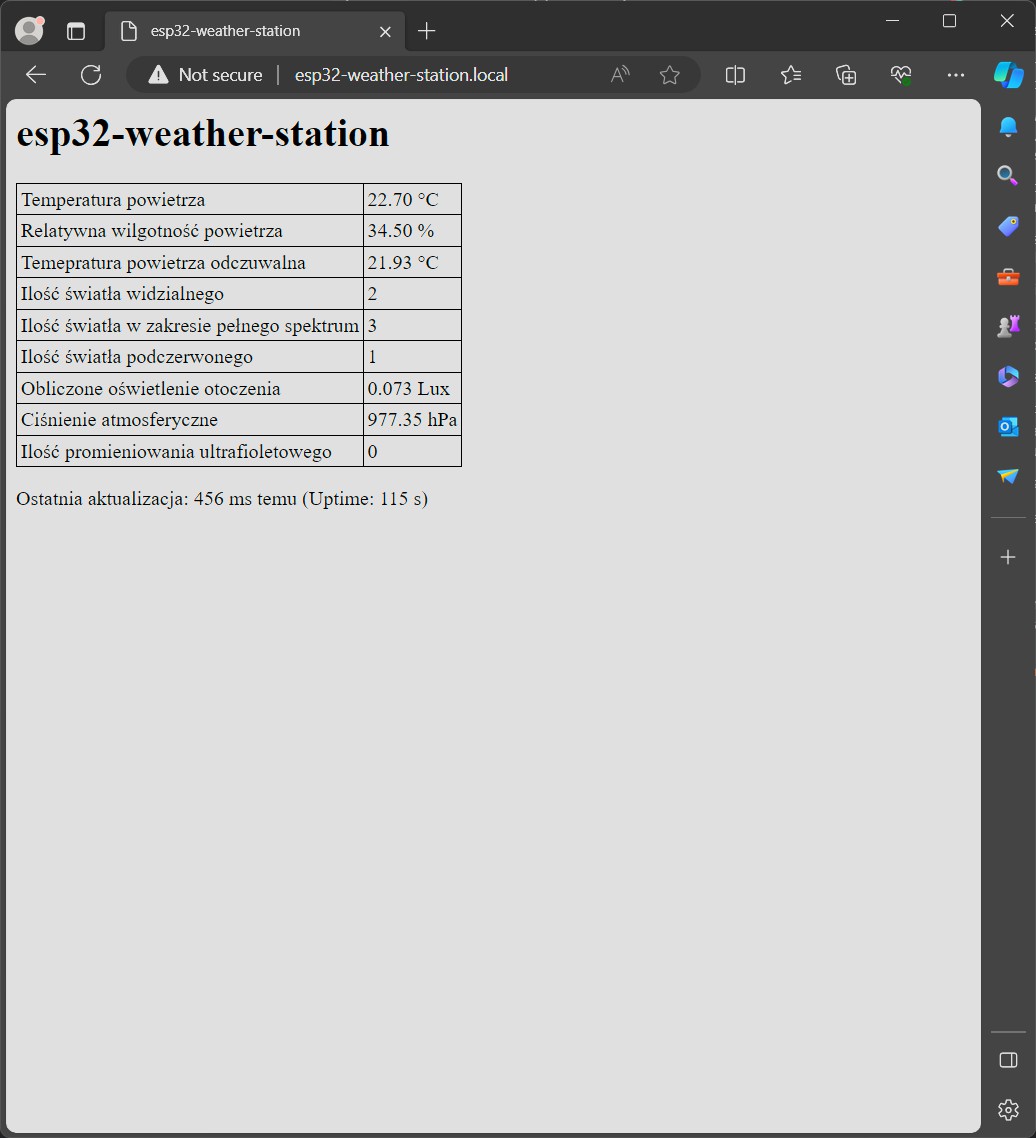
);

Program 10: Kod umozliwiaja˛cy poła˛czenie do serwera MQTT

# TESTY

## Interfejs WWW

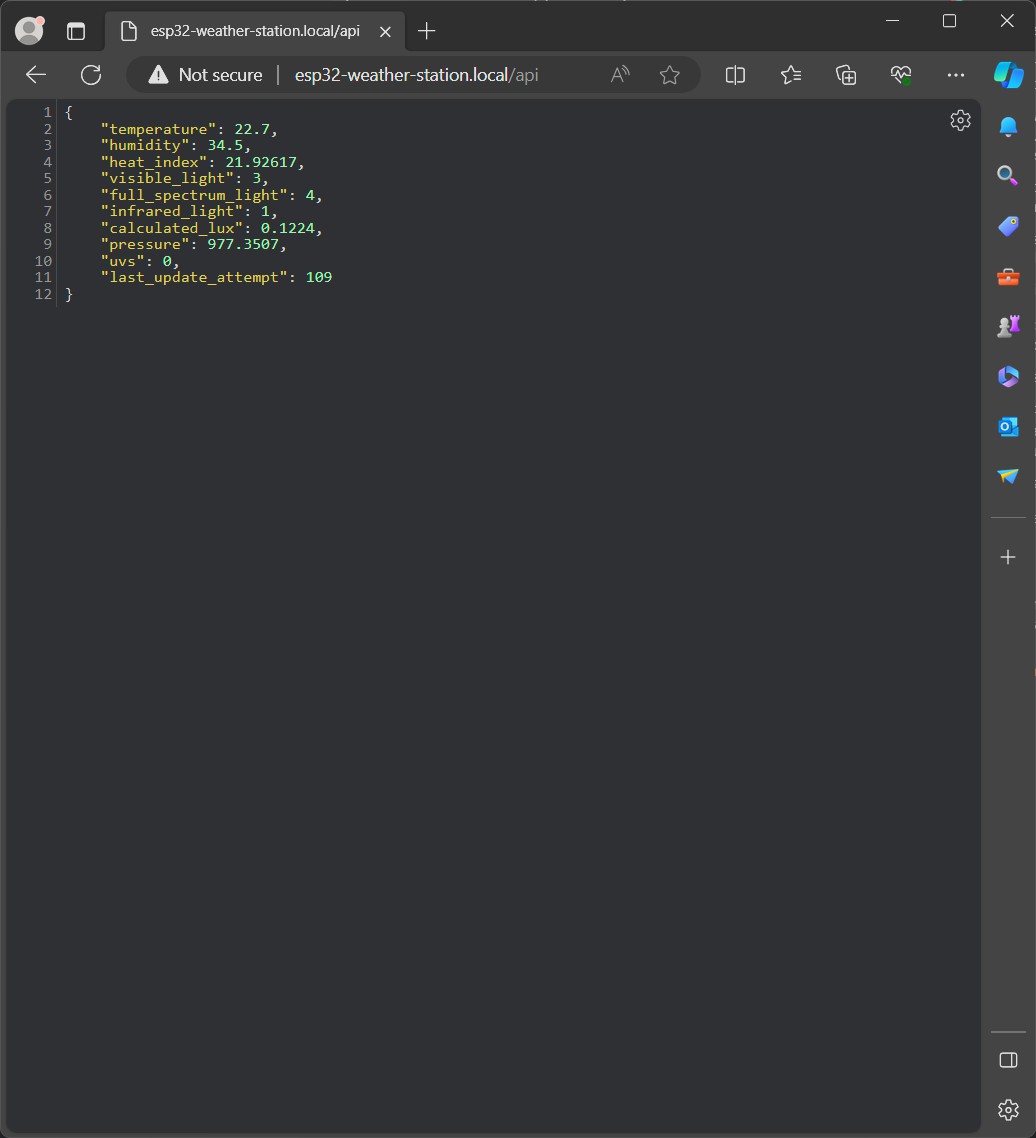
Interfejs WWW uz˙ytkownika zawiera komplet danych zbieranych przez urza˛dzenie oraz kiedy ostatni raz dane były aktualizowane. Ods´wiez˙enie przegla˛darki spodowuje pobranie nowych danych z urza˛dzenia.



Obrazek 3: Test interfejsu WWW

## Interfejs REST API

Interfejs REST dla programisty zwraca dane w formacie JSON.



Obrazek 4: Test interfejsu REST API

## MQTT

Urza˛dzenie dodatkowo wysyła dane z czujników do skonfigurowane serwera MQTT. Dane sa˛ wysyłane od razu po ich aktualizacji.

**Ci** MQTT Explorer D X

**'f 192.168.0.15**

* **zigbee2mq1t** 11;:; t:Jp1cs 133 1··1essages 1

**l' homeassistant 'f sensor**

* + **0x00124b002a4ef85a** 15 top cs 5 n°essages 1
  + **0x00124b002a4e3b99** , 5 :op1·:s 5 messages
  + **0x00124b002a4e3448** ·5 ;o:)1cs 5 messages
  + **DESKTOIP-6BA2R8D-satelllit**,**e**, :op1:,. 1·1essage
  + **szymon-pc** -1 tDp c- message \_1
  + **0x70b3d52b60040253** S :opics *8* ·11essages
  + **0xa4c 1389d192ee4bd** 3 :Jp ics 3 ·11essag es
  + **SZYMON-PC-satellliite** 1 :0D1c 13 messa;;,es
  + **SZYMON-PC** *(3* tJp cs 73 messages
  + **0xa4c13824ae796b8e** *i?,* ;o:)1cs 3 messages
  + **0xa4c138592be2d806** 3 :op1:=s 3 messa;;,es

**T esp32-weather-station temperature=** 22.70

**humidity** = 34.40

**heat\_index** = 21.92

**calc1.1lated\_lux** = 0.49

**full\_spectrnm\_light** = 16

**infrared\_light** = 4

**visible\_ligh1** = **12**

**pressure** = 977.32

**uvs** = 0

* **binary\_sensor** 16 tc:p cs c:.2 ·11essages)
* **switch**, , :op1c,. wessege
* **select** (2 tcp cs 2 n·essages 1
* **lock** 1 ,OD::. 1 1·1essege
* **update**-( top c- n'essage ·1
* **device\_automation** •15 topics 15 messages
* **button** ,6 :op1cs 33 wessages 1
* **hass.agent,**5 :opics 235 ·11essages
* **$SYS** (53 :op1cs 1c:.94 messages

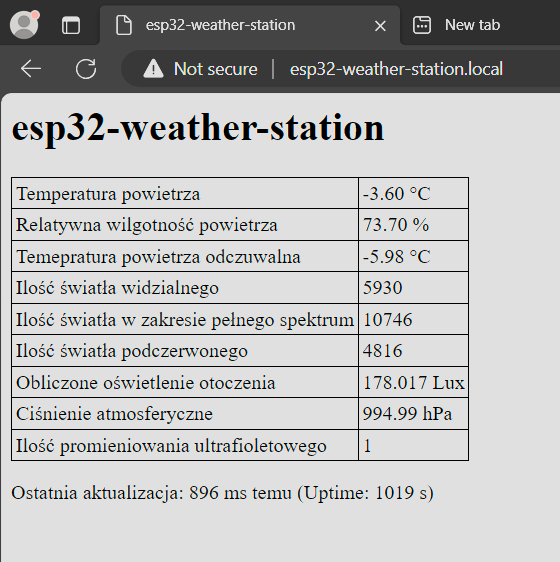
**Application Edit View**

= **Q. Search... C, DISCONNECT** & •

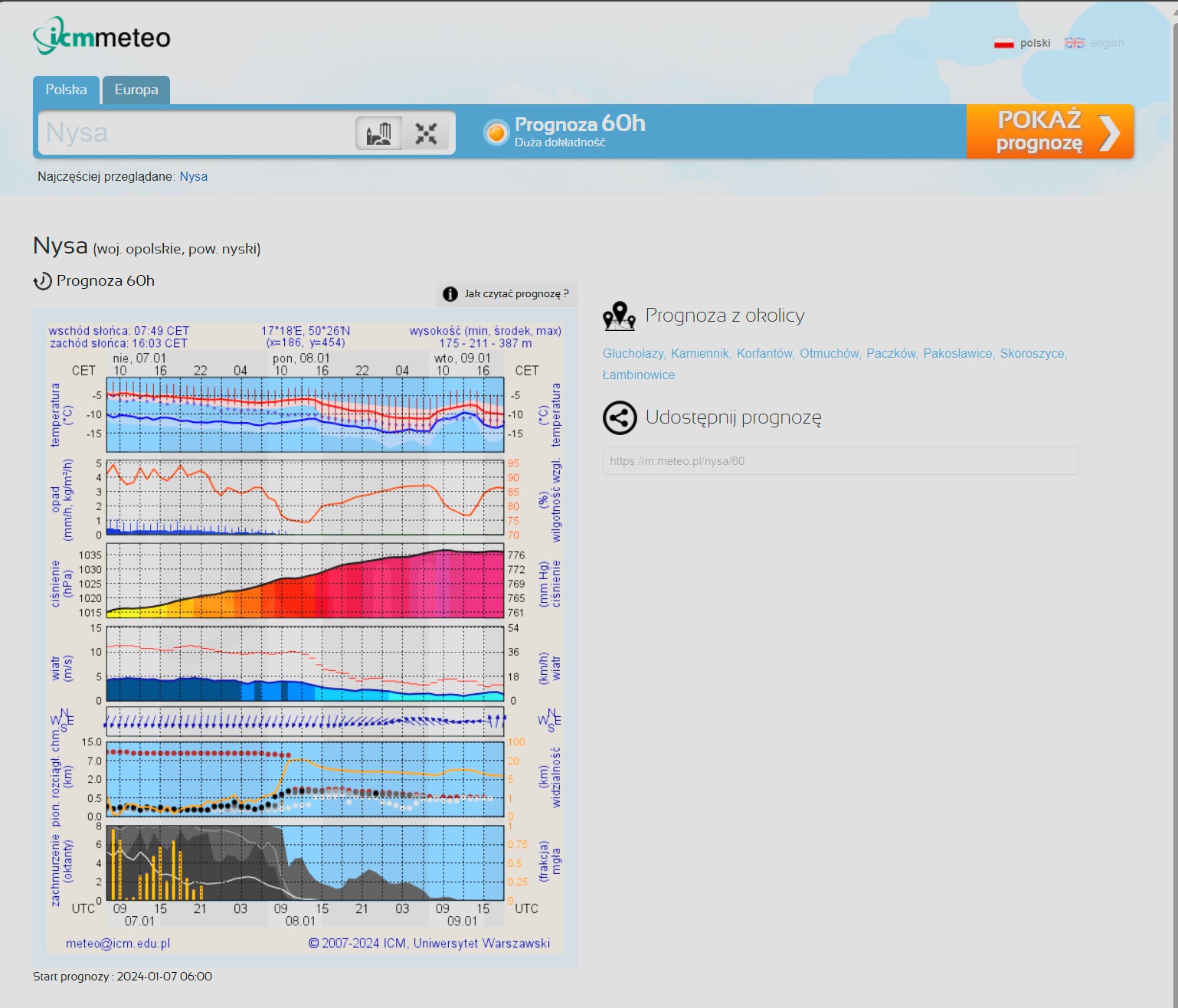
Obrazek 5: Test integracji z MQTT

## Porównanie z komercyjnymi danymi pogodowymi

Dane pogodowe dostarczane przez urza˛dzenie be˛da˛ bardziej dokładne lokalnie niz˙ prognoza pogody doste˛pna w komercyjnych serwisach. Obrazek 6 wykazuje, z˙e aktualna temperatura wynosi -3.6°C a wzgle˛dna wilgotnos´c´ powietrza wynosi 73.7%. Dane udoste˛pniane przez serwis meteo.pl [4] dla tej samej co lokalizacja urza˛dzenia wskazuje, z˙e temperatura wynosi -5°C a wzgle˛dna wilgotnos´c´ powietrza wynosi 85%.



Obrazek 6: Porównanie danych pogodowych - urza˛dzenie

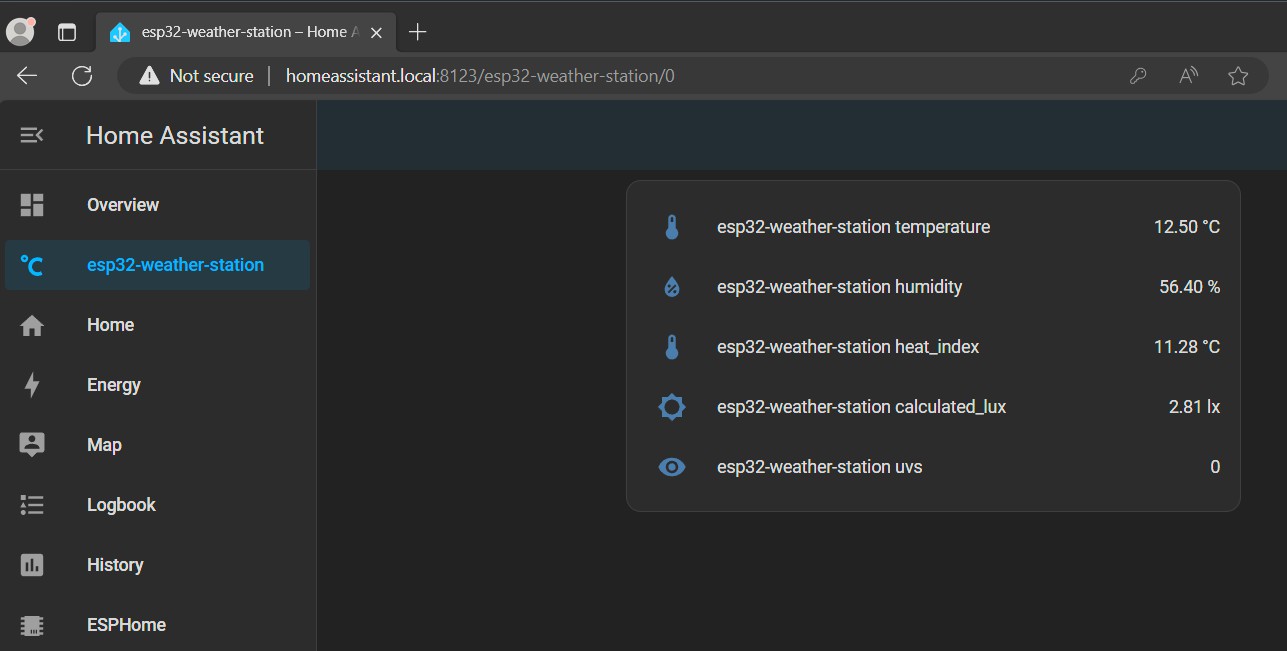


Obrazek 7: Porównanie danych pogodowych - meteo.pl

## Doste˛p do danych za pomoca˛ MQTT i Home Assistant

Dane przesyłane do MQTT moga˛ byc´ odczytane przez zewne˛trzny serwis w ceu

wys´wietlenia danych - w tym przypadku jest to aplikacja Home Assistant[2]. Za po- moca˛ wpisu w konfiguracji w je˛zyku YAML moz˙liwe jest odczytanie danych z serwera MQTT i wys´wietlenie danych tych na panelu.



Obrazek 8: Doste˛p do danych pogodowych przez Home Assistant

# WNIOSKI

Moduły bazuja˛ce na mikrokontrolerze ESP32 pozwalaja˛ na łatwa˛ realizacje˛ projek- tów, a przez duz˙a˛ róz˙norodnos´c´Integracja z MQT doste˛pnych na rynku modułów bazu- ja˛ych na tym mikrokontrolerze daje to ogromna˛ wszechstronnos´c´ w realizacji oraz doste˛p- nos´ci, jednoczes´nie zachowuja˛c kompatybilnos´c´ mie˛dzy modułami.

W tej pracy zrealizowana lokalna stacja pogodowa umoz˙liwia odczyt danych pogodowych w sieci lokalnej a takz˙e umoz˙liwia łatwa˛ integracje˛ z zewne˛trzymi systemami (interfejs REST API oraz integracja z MQTT).

Doste˛p do danych spoza sieci lokalnej moz˙liwy jest po podła˛czeniu do doste˛pnego publicznie serwera MQTT - urza˛dzenie zostaje doste˛pne tylko w sieci lokalnej i nie jest naraz˙one na potencjalne ataki w przypadku jez˙eli urza˛dzenie te byłoby doste˛pne pub- licznie. Lokalny doste˛p do urza˛dzenia jest jest bardzo prosty dzie˛ki protokołowi mDNS - aby poła˛czyc´ sie˛ z urza˛dzeniem nie jest potrzebny adres IP, a wystarczy nazwa urza˛dzenia w sieci, która jest statyczna.

Kolejnym etapem rozwoju projektu mogłabybyc´ kolejna wersja interfejsu uz˙ytkown- ika, dodatkowe integracje z innymi systemami zewne˛trzynymi lub integracja czujników oraz mikrokontrolera na jednej płytce PCB.

Podła˛czenie kolejnych czyjników do urza˛dzenia równiez˙ nie powinno sprawiac´ prob- lemu ze wzgle˛du na duz˙a˛ doste˛pnos´c´ dodatkowyc´ pinów WE/WYJ oraz otwartej nartury całego ekosystemu Arduino.

# LITERATURA

1. fhessel. *Biblioteka ESP32 HTTPS Server*. URL: https : / / github . com / fhessel/esp32\_https\_server/tree/master.
2. home-assistant.io. *Home Assistant*. URL: https://www.home-assistant. io/.
3. LILYGO. *LILYGO T-SIM7000G*. URL: [https://www.lilygo.cc/products/](http://www.lilygo.cc/products/) t-sim7000g.
4. meteo.pl. *Nysa - prognoza pogody 60h | ICM Meteo*. URL: https://m.meteo. pl/nysa/60.
5. n.a. *Adafruit DPS310 Precision Barometric Pressure and Altitude Sensor*. URL: https://learn.adafruit.com/adafruit- dps310- precision- barometric-pressure-sensor.
6. n.a. *Adafruit LTR390 UV Sensor*. URL: https://learn.adafruit.com/ adafruit-ltr390-uv-sensor.
7. n.a. *Adafruit TSL2591 High Dynamic Range Digital Light Sensor*. URL: https:

//learn.adafruit.com/adafruit-tsl2591.

1. n.a. *DHT11, DHT22 and AM2302 Sensors*. URL: https://learn.adafruit. com/dht.
2. n.a. *ESP32-WROOM-32 DATASHEET*. URL: https : / / www . espressif . com/sites/default/files/documentation/esp32- wroom- 32\_ datasheet\_en.pdf.
3. Olimex. *Olimex ESP32-POE*. URL: [https://www.olimex.com/Products/](http://www.olimex.com/Products/) IoT/ESP32/ESP32-POE/open-source-hardware.
4. plapointe6. *Biblioteka ESP32 EspMQTTClient*. URL: https://github.com/ plapointe6/EspMQTTClient.
5. Paweł Strajeczek. *Multicast DNS i Bonjour/Zeroconf*. URL: https://strejczek. com/multicast-dns-i-bonjourzeroconf/.

# SPISY PROGRAMÓW, TABEL, FOTOGRAFII

**Spis tabel**

1. Najwaz˙niejsze specyfikacje ESP-WROOM-32 7
2. Specyfikacja TSL25911 8
3. Specyfikacja DHT22 10
4. Specyfikacja DPS310 12
5. Specyfikacja LTR390 14

## Spis programów

1. Test czujnika TSL25911 9
2. Test czujnika DHT22 11
3. Test czujnika DPS310 13
4. Test czujnika LTR390 15
5. Przykład konfiguracji usługi mDNS 17
6. Wycinek kodu konfiguruja˛cego serwer http 18
7. Kod obsługuja˛cy interfejs uz˙ytkownika 20
8. Kod deklaracji szablonu html 21
9. Kod doste˛pu do danych diagnostycznych 22
10. Kod umozliwiaja˛cy poła˛czenie do serwera MQTT 23

## Spis obrazków, fotografii

1. Schemat urza˛dzenia 5
2. Fotografia prototypu urza˛dzenia 6
3. Test interfejsu WWW 24
4. Test interfejsu REST API 25
5. Test integracji z MQTT 26
6. Porównanie danych pogodowych - urza˛dzenie 27
7. Porównanie danych pogodowych - meteo.pl 28
8. Doste˛p do danych pogodowych przez Home Assistant 29

# STRESZCZENIE

Praca inz˙ynierska podejmuje temat "Konstrukcja stacji pogodowej opartej na mikrokon- trolerze ESP32 z interfejsem uz˙ytkownika oraz API REST". Głównym celem pracy było przygotowanie prototypu urza˛dzenia stacji pogodowej wraz z oprogramowaniem oraz test przydatnos´ci takiego urza˛dzenia do celów domowych. W ramach pracy został wybrany mikrokontroler oraz czujniki wykorzystane w urza˛dzeniu. Zostało przygotowane równiez˙ oprogramowanie, które steruje praca˛ mikrokontrolera - urza˛dzenia udoste˛pnia interfejs www dla uz˙ytkownika, interfejs programowy REST API oraz integracje z MQTT. Duz˙a róz˙norodnos´c´ czujników, wysoka doste˛pnos´c´ mikrokontrolerów oraz bogaty ekosystem programowy umoz˙liwia szybka˛, tania˛ i prosta˛ konstrukcje urza˛dzen´ bazuja˛cych na ESP32. Zaletami lokalnej stacji pogodowej jest wysoka dokładnos´c´ oraz wysoka przewidywal- nos´c´ odczytów w porównaniu do danych zbieranych przez inne komercyjne serwisy.