



**UMCS**

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ  
W LUBLINIE  
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **informatyka**

**Jan Bylina**

nr albumu: 303827

**Projekt oraz implementacja systemu gromadzenia rozproszonych danych z wykorzystaniem technologii LoRa**

Design and implementation of the distributed data collection system using LoRa technology

Praca licencjacka  
napisana w Katedrze Oprogramowania Systemów Informatycznych  
Instytutu Informatyki UMCS  
pod kierunkiem **dr hab. Przemysława Stpiczyńskiego**

**Lublin 2023**



# Spis treści

<b>Wstęp</b>	<b>7</b>
<b>1 Wykorzystane narzędzia, technologie i protokoły</b>	<b>9</b>
1.1 Urządzenia wykorzystywane w projekcie . . . . .	9
1.1.1 ESP32 . . . . .	9
1.1.2 Rasberry Pi Pico . . . . .	10
1.1.3 STM32 . . . . .	11
1.2 Języki programowania i technologie . . . . .	11
1.2.1 C++ for Arduio . . . . .	11
1.2.2 C for STM32 . . . . .	11
1.2.3 MicroPython for Rasberry Pi Pico . . . . .	11
1.3 Bazy danych i pozostałe technologie . . . . .	11
1.3.1 InfluxDB 2 . . . . .	11
1.3.2 Python for MQTT . . . . .	12
1.4 Protokoły komunikacyjne . . . . .	12
1.4.1 MQTT . . . . .	12
1.4.2 LoRa . . . . .	12
1.4.3 HTTP . . . . .	12
1.5 Bazy danych i pozostałe technologie . . . . .	12
1.5.1 InfluxDB 2 . . . . .	12
1.5.2 Docker . . . . .	12
1.5.3 PlatformIO . . . . .	12
<b>2 Istniejące rozwiązania</b>	<b>13</b>
2.1 LoRaWAN . . . . .	13
2.1.1 The Things Network ? . . . . .	13
2.1.2 ChirpStack ? . . . . .	13
2.1.3 Loriot ? . . . . .	13
2.2 Artykuły . . . . .	13
2.3 Wpisy w sieci i blogach . . . . .	13

<b>3 Założenie i Implementacja</b>	<b>15</b>
3.1 Podstawowe cele sieci . . . . .	15
3.2 Części sieci . . . . .	15
3.2.1 Węzły sieci . . . . .	16
3.2.2 Stacja przekaźnikowa . . . . .	16
3.2.3 Broker wiadomości . . . . .	16
3.2.4 Baza danych . . . . .	16
3.3 Wiadomości . . . . .	18
3.4 Obsługa protokołu . . . . .	18
3.4.1 Działanie węzłów . . . . .	18
3.4.2 Działanie przekaźnika . . . . .	19
3.4.3 Działanie bazy danych i programu zapisującego dane . . . . .	19
3.5 Implementacja . . . . .	19
3.5.1 Implementacja węzłów sieci . . . . .	19
3.5.2 Implementacja stacji przekaźnikowej . . . . .	20
3.5.3 Implementacja brokera wiadomości . . . . .	21
3.5.4 Baza danych . . . . .	21
3.5.5 Program zapisujący dane . . . . .	21
<b>4 Wdrożenie i testy</b>	<b>23</b>
4.1 Wdrożenie . . . . .	23
4.1.1 Węzły sieci . . . . .	23
4.1.2 Stacja przekaźnikowa . . . . .	23
4.1.3 Serwer testowy . . . . .	23
4.1.4 Broker wiadomości . . . . .	25
4.1.5 Baza danych i program zapisujący dane . . . . .	26
4.1.6 Redis . . . . .	26
4.2 Testy . . . . .	26
4.2.1 Poprawność działania . . . . .	26
4.2.2 Wydajność . . . . .	28
4.2.3 Zauważone problemy . . . . .	29
<b>5 Wnioski i perspektywy rozwoju</b>	<b>31</b>
<b>Spis listingów</b>	<b>33</b>
<b>Spis tabel</b>	<b>35</b>
<b>Spis rysunków</b>	<b>37</b>

Bibliografia

40



# Wstęp

Tu treść wstępu WSTĘP WSTEP ——



# Rozdział 1

## Wykorzystane narzędzia, technologie i protokoły

### 1.1 Urządzenia wykorzystywane w projekcie

#### 1.1.1 ESP32

ESP32 to jednoukładowy mikrokontroler, zaprojektowany i produkowany przez firmę Espressif Systems. Jego najważniejsze cechy to:

- energooszczędny procesor RISC o częstotliwości do 240 MHz
- 520 kB pamięci SRAM
- WiFi 802.11 b/g/n
- Bluetooth
- liczne interfejsy cyfrowe i analogowe, w tym:
  - UART
  - I2C
  - SPI
  - I2S
  - CAN
  - ADC
  - DAC
  - PWM
  - Ethernet MAC

- USB 2.0
- ...

[22]

Powstało wiele wersji tego układu, różniące się m.in. szybkością procesora, ilością pamięci flash, ilością pinów, ilością interfejsów cyfrowych i analogowych, a także możliwością pracy w trybie bezprzewodowym (WiFi) lub przewodowym (Ethernet)[23]. Najczęściej układ te wykorzystywane różnych projektach IoT, zarówno jako czujniki, jak i serwery.[22]

W projekcie ESP32 zostało wykorzystane w dwóch płytach TTGO T3 V1.6.1. Jedna z płyt została wykorzystana jako przekaźnik danych pomiędzy siecią LoRa a siecią WiFi, a druga jako część systemu zbierania danych z wykorzystaniem LoRa

### 1.1.2 Raspberry Pi Pico

Raspberry Pi Pico to płytki z mikrokontrolerem RP2040, zaprojektowana i produkowana przez firmę Raspberry Pi Foundation. Charakteryzuje się ona dwurdzeniowym procesorem ARM Cortex-M0+ o częstotliwości 133 MHz, 264 kB pamięci SRAM oraz 2 MB pamięci flash. Płytki posiada również wiele interfejsów cyfrowych i analogowych, w tym:

- UART
- I2C
- SPI
- I2S
- ADC
- DAC
- PWM
- USB 1.1

[13, 12] Płytki ta jest często wykorzystywana przez hobbyistów do różnych projektów IoT, a także jako sterownik silników, czy kontroler robotów.[12]

W projekcie dwie płytki zostały wykorzystane jako część systemu zbierania danych.

### 1.1.3 STM32

STM32 to rodzina 32 bitowych mikrokontrolerów produkowanych przez firmę STMicroelectronics. Bazują one na architekturze ARM Cortex-M, oferują wysoką wydajność i energooszczędność. Cztery główne rodzaje mikrokontrolerów STM32 to:

- Rodzina płyt z częścią kodu F(4/5) i H — oferują największą wydajność
- Rodzina płyt z częścią kodu L — oferują największą energooszczędność
- Rodzina płyt z częścią kodu G/C/F(1/3) — do zastosowań ogólnych
- Rodzina płyt z częścią kodu W(L/B/BA) — do zastosowań bezprzewodowych.  
[SRPAWDZIĆ]

[21] Głównym zastosowaniem STM32 są urządzenia wbudowane, w tym urządzenia medyczne, roboty, samochody, a także urządzenia IoT[NEED CITE]

W projekcie wykorzystano dwie płytki STM32WL55, które zostały wykorzystane jako część systemu zbierania danych.

## 1.2 Języki programowania i technologie

### 1.2.1 C++ for Arduino

---

### 1.2.2 C for STM32

---

### 1.2.3 MicroPython for Raspberry Pi Pico

MicroPython jest językiem

## 1.3 Bazy danych i pozostałe technologie

### 1.3.1 InfluxDB 2

---

### 1.3.2 Python for MQTT

## 1.4 Protokoły komunikacyjne

---

### 1.4.1 MQTT

---

### 1.4.2 LoRa

---

### 1.4.3 HTTP

---

## 1.5 Bazy danych i pozostałe technologie

### 1.5.1 InfluxDB 2

---

### 1.5.2 Docker

---

### 1.5.3 PlatformIO

---

## Rozdział 2

### Istniejące rozwiązania

#### 2.1 LoRaWAN

##### 2.1.1 The Things Network ?

##### 2.1.2 ChirpStack ?

##### 2.1.3 Loriot ?

#### 2.2 Artykuły

#### 2.3 Wpisy w sieci i blogach



# Rozdział 3

## Założenie i Implementacja

W poniższym rozdziale zaprocentowano założenia, potrzeby i projekt projektu Sieci Mesh na bazie LoRa

### 3.1 Podstawowe cele sieci

System ma kilka podstawowych założeń:

- Jeden centralny punkt gromadzenia danych
- Zapisywanie danych do bazy danych szeregów czasowych, w celu ich dalszego przetwarzania
- Zbieranie danych z dużego obszaru
- Niezależność od istniejących metod przesyłu danych (WiFi, Sieci komórkowe, Łączność satelitarna)
- Niezależność od platformy sprzętowej
- Zapewniać możliwie dużą dostarczalność pakietów
- Dane czasowe nie muszą być super dokładne (dopuszczalne są drobne opóźnienia)
- Węzły sieci tylko wysyłają dane, same nie konsumują przychodzących wiadomości

### 3.2 Części sieci

W celu uzyskania wyżej wspomnianych założeń, zaproponowano system złożonych z kilku części:

- Węzły sieci
- Stacje przekaźnikowe
- Broker wiadomości
- Baza danych

Na rysunku 3.1 przedstawiono schemat ideowy systemu.

### 3.2.1 Węzły sieci

Węzły sieci to urządzenia wyposażone w moduł LoRa i odpowiednie oprogramowanie pozwalające na pełną obsługę sieci. Gdy węzeł odbierze wiadomość, sprawdza jej poprawność i rozsyła ją dalej w celu zapewnienia jak największego zasięgu i dostarczalności.

Urządzenie to może być również wyposażone w różnego rodzaju czujniki, które dostarczają danych bazy danych.

### 3.2.2 Stacja przekaźnikowa

Stacja przekaźnikowa to urządzenie wyposażone zarówno w moduł LoRa jak i moduł umożliwiający komunikację z siecią Internet (np. moduł WiFi lub moduł bazy Ethernet).

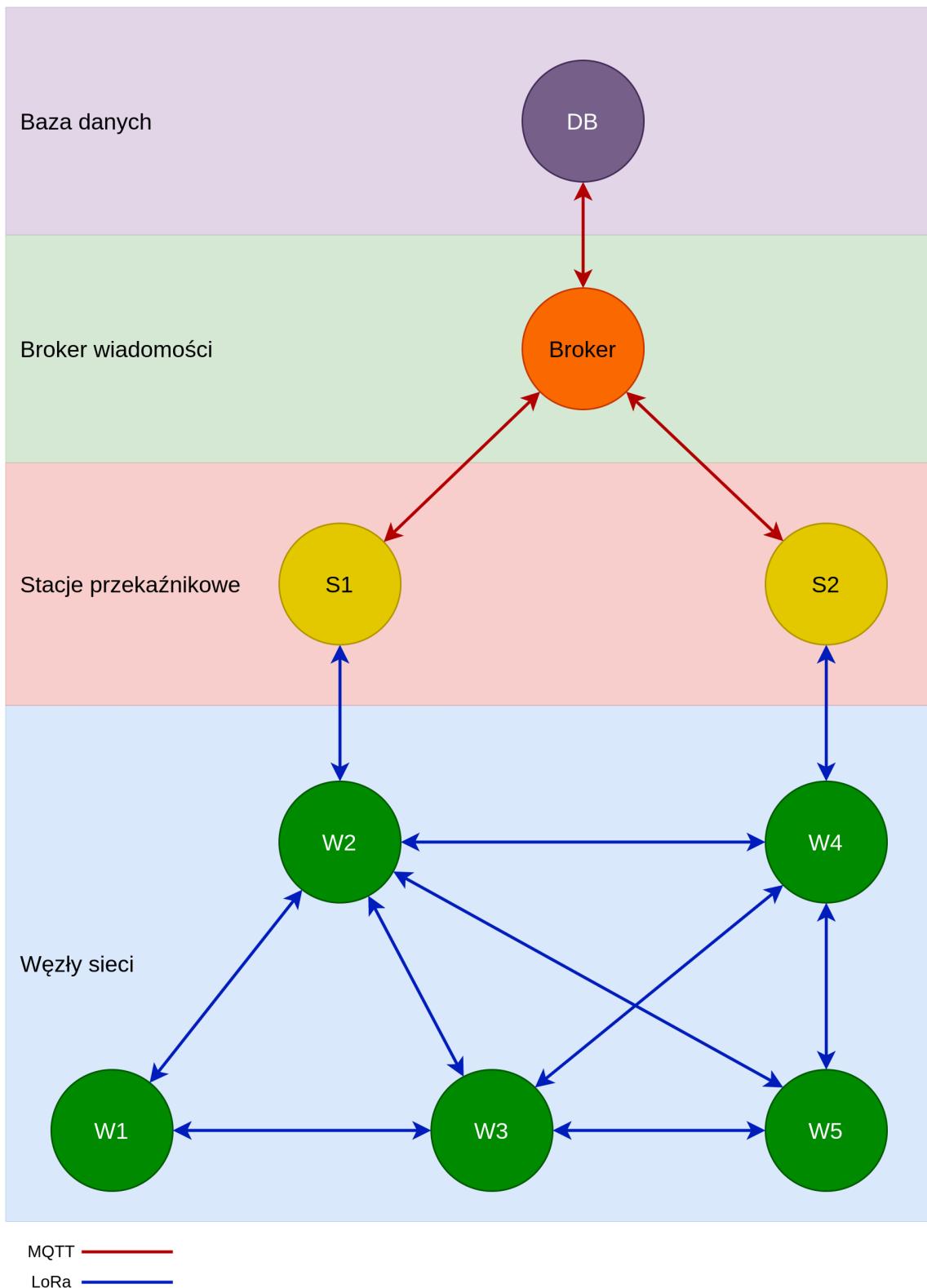
Urządzenie to odbiera przychodzące wiadomości LoRa i przesyła je do brokera wiadomości

### 3.2.3 Broker wiadomości

Broker wiadomości to program, działający na komputerze mającym dostęp do sieci, umożliwia on wydajną komunikację pomiędzy Stacją Przekaźnikową a Bazą Danych

### 3.2.4 Baza danych

Baza Danych umożliwiająca zapisywanie sporej ilości danych, uwzględniając również ich czas (baza danych szeregow czasowych). O zapisy danych z brokera wiadomości do bazy danych dba osobny program, który powinien sprawdzać również poprawność tych wiadomości, jak i dbać o to by nie zapisywać powtórzonych wiadomości



Rysunek 3.1: Diagram prezentujący schemat ideowy systemu

### 3.3 Wiadomości

Każda z wiadomości przesyłanych za pomocą tych sieci powinna mieć formę jak zaprezentowano na listingu 3.1

Zawiera ona pola:

- **ttl** — [Ang. time to live — czas życia] Wartość określająca maksymalną liczbę skoków pomiędzy węzłami sieci. Domyslnie wynosi 10, może zostać wydłużona w zależności od wielkości planowanej sieci
- **m\_id** — UUID [1] wiadomości, gwarantujący niepowtarzalność tej wiadomości. Ułatwia również jej dalsze przetwarzanie
- **d\_id** — numer identyfikacyjny urządzenia z którego pochodzi wiadomość
- **values** — słownik zawierający dane z urządzenia, do zapisania w bazie

```

1  {
2      "d_id": "id_233",
3      "values": {
4          "temp": "21",
5          "hum": "50",
6          "press": "1000",
7          "light": "100",
8          "co2": "1000",
9          "pm25": "10",
10         "pm10": "20"
11     },
12     "ttl": 10,
13     "m_id": "eaa17a7b-9388-43b6-9310-731c942fc6b9"
14 }
```

**Listing 3.1:** Przykładowa wiadomość przesyłana przez system

### 3.4 Obsługa protokołu

#### 3.4.1 Działanie węzłów

Wiadomości generowane są przez węzły sieci, zawierając wszystkie niezbędne pola (wymienione wyżej) i odczyty z czujników zamieszczonych na węźle. Następnie zostaje ona rozesłana do wszystkich węzłów w zasięgu (broadcasting).

Węzeł odbierając wiadomość, sprawdza jej poprawność (czy jest odpowiednio sformatowana, czy zawiera wszystkie potrzebne pola), i jeżeli wiadomość jest poprawna, a pole `ttl` jest większe od 0 rozsyła wiadomość dalej. Sprawdzanie wiadomości odbywa się by wyeliminować wiadomości niepoprawne z sieci.

### 3.4.2 Działanie przekaźnika

Przekaźnik odbiera wiadomości, i przesyła je do brokera wiadomości. Nie sprawdza poprawności wiadomości by zapewnić maksymalną wydajność i niezawodność.

### 3.4.3 Działanie bazy danych i programu zapisującego dane

Program pobiera kolejne wiadomości od brokera i przetwarza je w kolejności:

1. Sprawdzenie poprawności wiadomości
2. Sprawdzenie czy wiadomość nie została już sprawdzona (na podstawie pola `m_id`)
3. Zapisanie danych ze słownika `values` do bazy danych i przyporządkowanie ich do `d_id`, oznaczenie ich znacznikiem czasowym.
4. Zapisanie w pamięci operacyjnej `m_id`, potem do wykorzystania w kroku 2

Baza danych powinna przechowywać dane, takie jak:

- `d_id` — identyfikator urządzenia
- znacznik czasowy
- wartość pomiaru
- nazwa pomiaru

## 3.5 Implementacja

### 3.5.1 Implementacja węzłów sieci

W wyniku pracy nad systemem zbierania danych przygotowano implementację węzłów sieci w wykorzystaniem 2 platform sprzętowych:

- Raspberry Pi Pico
- TTGO LoRa32

## Raspberry Pi Pico

W wyniku pracy nad systemem zostały przygotowane dwa, identyczne urządzenia oparte o Raspberry Pi Pico. Urządzenie zostało wyposażone w moduł LoRa SX1262 [2] (z wykorzystaniem płytki rozwojowej Waveshare SX1262 LoRa Node Module [3]) oraz czujnik temperatury i wilgotności DHT11. Urządzenie zostało zaprogramowane w języku MicroPython z wykorzystaniem biblioteki `micropySX126X` [10]. Urządzenie wysyła wiadomości zawierające odczyty z czujnika co 5 minut. Wysyłanie wiadomości odbywa się w sposób asynchroniczny, dzięki czemu urządzenie może wykonywać inne operacje w tym czasie.

## TTGO LoRa32

W wyniku pracy nad systemem zostało przygotowane jedno urządzenie oparte o TTGO LoRa32. Urządzenie zostało wyposażone w moduł LoRa SX1276 [5]. Urządzenie zostało zaprogramowane w języku C++ z użyciem PlatformIO, ekosystemu do programowania urządzeń IoT. [18]. W programie została wykorzystane biblioteki:

- `arduino-LoRa` — biblioteka do obsługi modułu LoRa [20]
- `ArduinoJson` — biblioteka do obsługi formatu JSON [7]
- `Adafruit-SSD1306` — biblioteka do obsługi wyświetlacza [6] OLED
- `ESPRandom` — biblioteka do obsługi sprzętowego generatora liczb pseudolosowych [16]

Urządzenie wysyła wiadomości zawierające odczyty z czujnika co 5 minut. Wiadomości zawierają wartości losowe, wygenerowane za pomocą sprzętowego generatora liczb pseudolosowych zintegrowanego z mikrokontrolerem ESP32. Wysyłanie wiadomości odbywa się w sposób asynchroniczny, dzięki czemu urządzenie może wykonywać inne operacje w tym czasie.

### 3.5.2 Implementacja stacji przekaźnikowej

W wyniku pracy nad systemem zostało przygotowane jedno urządzenie oparte o płytę TTGO LoRa32. Urządzenie zostało wyposażone w moduł LoRa SX1276 [5] oraz moduł WiFi ESP32 [22]. Urządzenie zostało zaprogramowane w języku C++ z wykorzystaniem PlatformIO [18]. W programie została wykorzystane biblioteki:

- `arduino-LoRa` — biblioteka do obsługi modułu LoRa [20]
- `Adafruit-SSD1306` — biblioteka do obsługi wyświetlacza OLED [6]

- WiFi — biblioteka do obsługi modułu WiFi ESP32, zawarta w pakiecie arduino-esp32 [11]
- PubSubClient — biblioteka do obsługi protokołu MQTT [4]

Urządzenie odbiera wiadomości LoRa i przesyła je do brokera wiadomości za pomocą protokołu MQTT. Wysyłanie wiadomości odbywa się w sposób asynchroniczny, dzięki czemu urządzenie może wykonywać inne operacje w tym czasie. Urządzenie wyświetla na ekranie OLED informacje o stanie sieci LoRa, oraz otrzymanych wiadomościach. Urządzenie nie sprawdza poprawności wiadomości, by zapewnić jak największą wydajność i niezawodność.

### 3.5.3 Implementacja brokera wiadomości

Aby zapewnić niezawodność i wysoką wydajność komunikacji zdecydowano użyć otwartoźródłowego brokera wiadomości Mosquitto [17]. Jest to sprawdzony, wydajny i niezawodny program, który jest szeroko stosowany w systemach IoT, również w przemyśle. [17]

### 3.5.4 Baza danych

W celu zapisywania danych zdecydowano użyć bazy danych InfluxDB 2.0 [14]. Jest to baza danych szeregow czasowych, która jest wydajna i niezawodna. [14]

### 3.5.5 Program zapisujący dane

W celu zapisywania danych zdecydowano użyć programu napisanego w języku Python. Program został napisany w języku Python, z wykorzystaniem bibliotek:

- paho-mqtt — biblioteka do obsługi protokołu MQTT [9]
- influxdb-client — biblioteka do obsługi bazy danych InfluxDB [15]
- redis-py — biblioteka do obsługi bazy danych Redis [19]

Ostania, wymieniona biblioteka została użyta do połączenia się z bazą danych w pamięci w celu przechowywania identyfikatorów wiadomości, które zostały już sprawdzone. Dzięki temu program nie zapisuje powtórzonych wiadomości do bazy danych.



# Rozdział 4

## Wdrożenie i testy

### 4.1 Wdrożenie

System został wdrożony 1 maja 2023 roku. Wdrożenie polegało na uruchomieniu serwera z bazą danych, jedną stację przekaźnikową oraz uruchomieniu 3 urządzeń pomiarowych.

#### 4.1.1 Węzły sieci

Trzy węzły sieci zostały rozmieszczone na małym obszarze, około 30 metrów od siebie, w dwóch budynkach. Dwa urządzenia (jedno Raspberry Pi Pico i jedno TTGO LoRa32) zasilane są z sieci, jedno za pomocą baterii. Urządzenia zostały połączone z czujnikami za pomocą zwykłych kabli prototypowych, jak na zdjęciu 4.1. Przykład urządzenia wykorzystującego wbudowane czujniki został zaprezentowany na zdjęciu 4.2

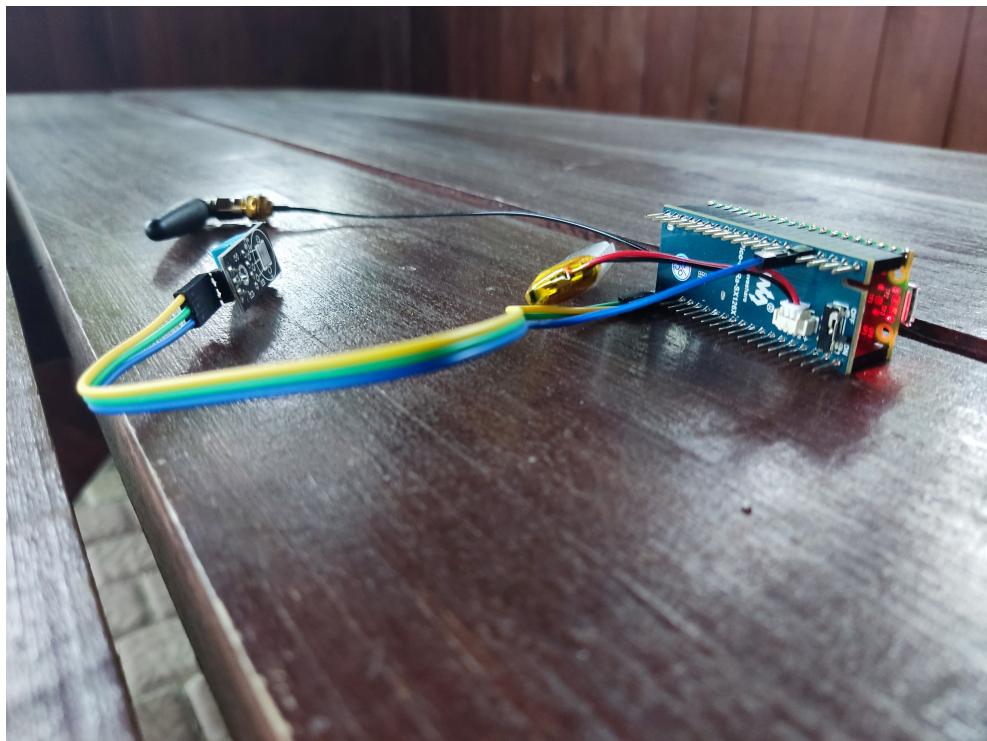
#### 4.1.2 Stacja przekaźnikowa

Stacja przekaźnikowa została umieszczona w pobliżu głównego serwera, w zasięgu sieci WiFi, wewnątrz budynku. Urządzenie zostało zasilone z sieci elektrycznej. Na ekranie urządzenia wyświetlane są informacje o stanie urządzenia, jak na zdjęciu 4.3.

#### 4.1.3 Serwer testowy

Na serwerze testowy zostały uruchomione wszystkie usługi, które zostały wykorzystane w projekcie, w szczególności:

- Mosquitto
- InfluxDB 2



Rysunek 4.1: Zdjęcie węzła sieci opartego o Raspberry Pi Pico wraz z baterią



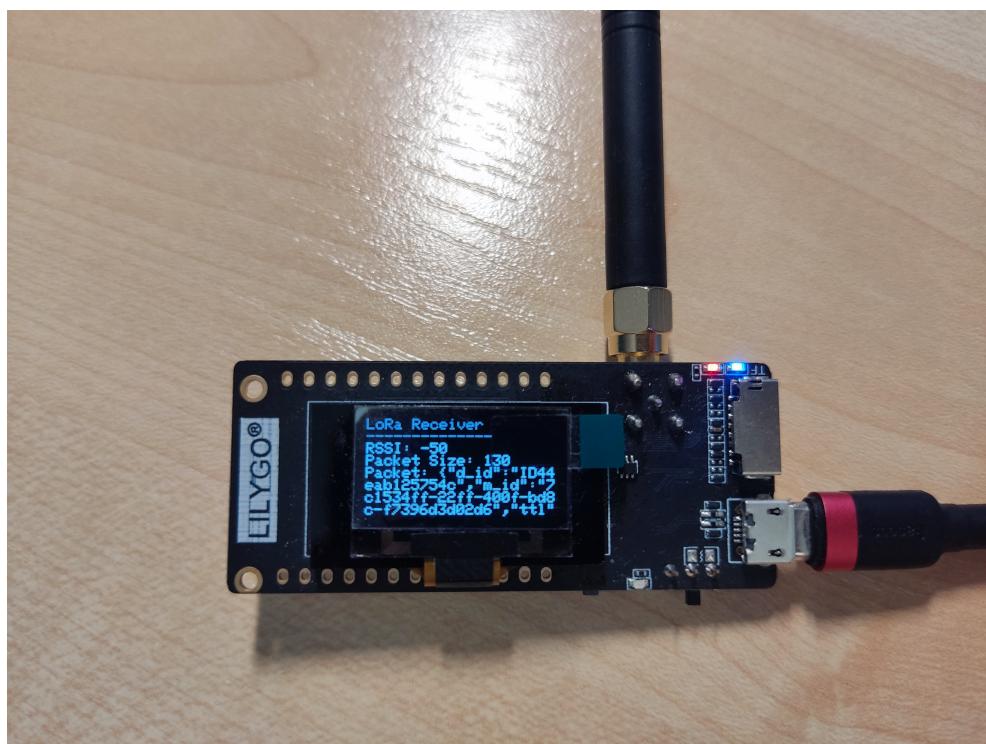
Rysunek 4.2: Zdjęcie węzła sieci opartego o TTGO LoRa32

- Program zapisujący do InfluxDB 2
- Redis

Serwer został uruchomiony na maszynie wewnętrz budynku, w sieci lokalnej. Na serwerze został uruchomiony system operacyjny Debian 11. Maszyna została wyposażona w procesor Intel Core i5-480M, 2 GB pamięci RAM oraz dysk SSD o pojemności 256 GB. Wszystkie usługi zostały uruchomione w kontenerach Dockera [8]. Wszystkie usługi zostały uruchomione w jednej sieci Dockera, w celu zapewnienia komunikacji między nimi.

#### 4.1.4 Broker wiadomości

Broker wiadomości został uruchomiony na wspólnym serwerze wewnętrz budynku. Broker został zabezpieczony hasłem, które zostało zapisane w pliku konfiguracyjnym. Wszystkie urządzenia zostały skonfigurowane tak, aby połączyć się z brokerem za pomocą tego hasła. Niestety, nie zostało użyte szyfrowanie TLS.



Rysunek 4.3: Zdjęcie stacji przekaźnikowej opartej o TTGO LoRa32

#### 4.1.5 Baza danych i program zapisujący dane

Baza danych wraz z programem zapisującym dane zostały uruchomione na wspólnym serwerze wewnętrz budynku. Baza danych została zabezpieczona tokenem dostępu, który został zapisany w pliku konfiguracyjnym. Wszystkie urządzenia zostały skonfigurowane tak, aby połączyć się z bazą danych za pomocą tego tokenu. Niestety, nie zostało użyte szyfrowanie TLS. Zarówno baza danych, jak i program zapisujący dane zostały uruchomione w kontenerach Dockera, w celu zapewnienia izolacji i łatwości konfiguracji.

#### 4.1.6 Redis

Redis został uruchomiony na wspólnym serwerze wewnętrz budynku. Redis pozostał niezabezpieczony, ponieważ został udostępniony w sieci wewnętrznej. Wszystkie urządzenia zostały skonfigurowane tak, aby połączyć się z Redisem za pomocą tego hasła. Niestety, nie zostało użyte szyfrowanie TLS. Redis został uruchomiony w kontenerze Dockera, w celu zapewnienia izolacji i łatwości konfiguracji.

### 4.2 Testy

System został przetestowany pod kątem poprawności działania, wydajności oraz niezawodności. Testy zostały przeprowadzone w dniach od 1 do 14 maja 2023 roku.

#### 4.2.1 Poprawność działania

Poprawność działania była weryfikowana na kilka sposobów:

##### Weryfikacja poprawności działania urządzeń pomiarowych

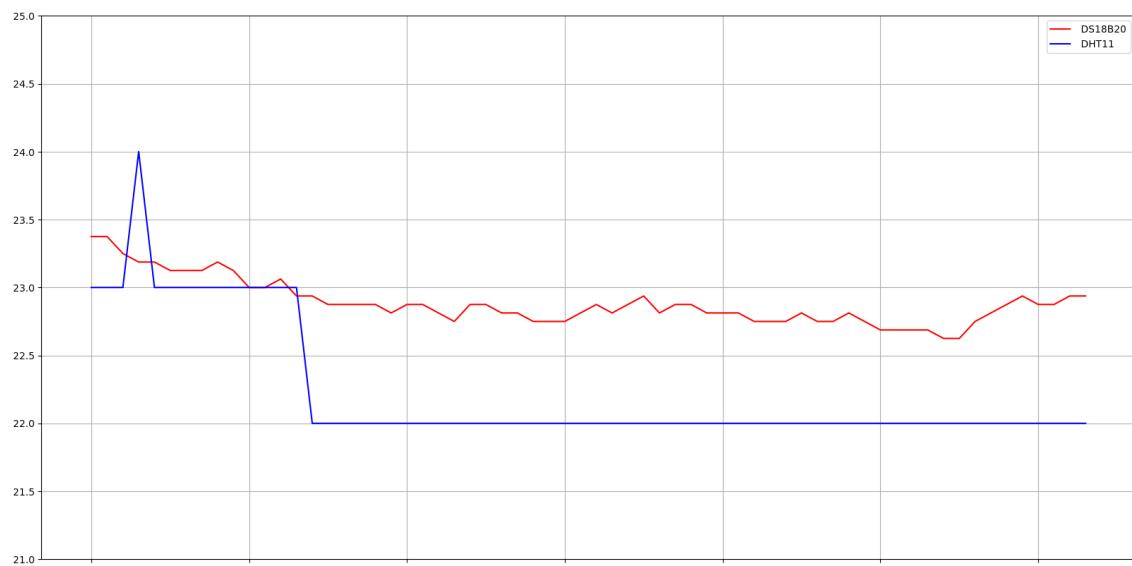
Weryfikacja poprawności działania urządzeń pomiarowych została przeprowadzona dla odczytów temperatury. Weryfikacja polegała na porównaniu odczytów z dwóch różnych czujników. Porównania dokonano pomiędzy czujnikiem DHT11 a czujnikiem DS18B20. Odczyty zostały zapisane do pliku CSV a następnie porównane. Przykładowe wyniki testów zostały przedstawione na rysunku 4.4.

Jak widać na załączonym rysunku, wyniki odczytów są bardzo zbliżone(maksymalna różnica pomiarów to 1°C), co świadczy o poprawności działania urządzeń pomiarowych.

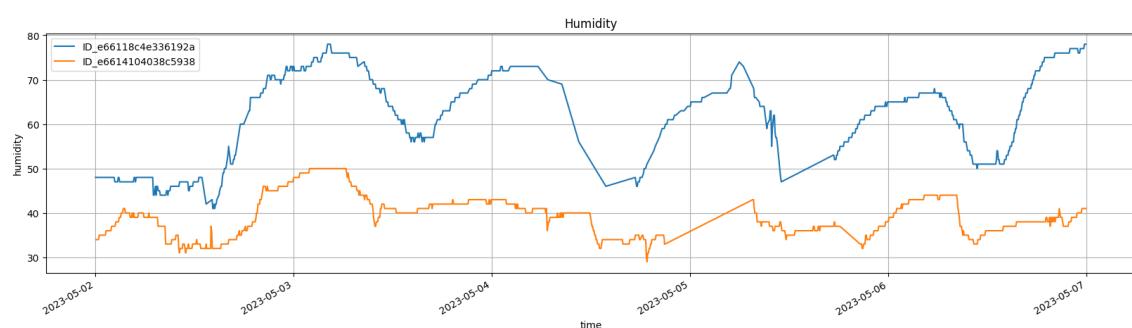
## Weryfikacja poprawności działania całego systemu

Weryfikacja poprawności działania całego systemu została przeprowadzona poprzez porównywanie pakietów wysyłanych przez urządzenia pomiarowe z pakietami zapisanymi w bazie danych. Weryfikacja polegała na porównaniu wszystkich pól pakietów. Porównanie to odbywało się ręcznie i dotyczyło tylko kilku pakietów. Dodatkowo weryfikowano poprawność działania poprzez sprawdzenie zmian we wskazaniach czujników. Przejścia dla temperatury i wilgotności były bardzo płynne,(co widać na rysunku) co świadczy o poprawności działania systemu. Przejścia dla liczb losowych były bardzo chaotyczne, co również świadczy o poprawności działania systemu. Przykładowe wyniki testów zostały przedstawione na rysunku 4.5.

Dzięki wyżej wymienionym testom, udało się stwierdzić, że system działa poprawnie.



Rysunek 4.4: Wykres temperatury od czasu dla czujników DHT11 i DS18B20



Rysunek 4.5: Wykres wilgotności od czasu(w okresie od 2 Maja 2023 do 7 Maja 2023)

## 4.2.2 Wydajność

Testy wydajności zostały przeprowadzone w celu sprawdzenia, czy system jest w stanie obsłużyć wszystkie pakiety wysyłane przez urządzenia pomiarowe. Testy zostały przeprowadzone w dwóch etapach:

- Testy wydajności stacji przekaźnikowej
- Testy wydajności węzłów sieci

### Testy wydajności stacji przekaźnikowej

Testy zostały przeprowadzone w następujący sposób: z 3 węzłów sieci były emitowane pakiety w bardzo krótkich odstępach czasu (co 10 sekund). Wszystkie pakiety były wysyłane do stacji przekaźnikowej. Następnie przekazywała ona te pakiety do brokera wiadomości. Wszystkie pakiety były zapisywane w bazie danych.

W wyniku testów zostało stwierdzone, że stacja przekaźnikowa jest w stanie obsługiwać wszystkie pakiety wysyłane przez węzły sieci poprawnie.

### Testy wydajności węzłów sieci

Testy zostały przeprowadzone w następujący sposób: z 3 węzłów sieci były emitowane pakiety w bardzo krótkich odstępach czasu(10 s). Weryfikacji podlegało to, czy wszystkie pakiety zostały wysłane ponownie rozgłoszone przez węzły sieci. Wszystkie pakiety były odbierane równie przez stację przekaźnikową.

W wyniku testów zostało stwierdzone, że węzły sieci są w stanie obsługiwać wszystkie pakiety wysyłane przez sieci poprawnie. Przykładowe wyniki testów zostały przedstawione na rysunku 4.6.

```

INFO: Message received: {"ttl":9,"values":{"humidity":79,"temperature":13}},{"m_id":"1710aa99-16ac-4783-84cb-9c0a9e01f402"},"d_id":"ID_e66118c4e336192a"}}
INFO: Message forwarded: b'{"d_id": "ID_e66118c4e336192a", "values": {"humidity": 79, "temperature": 13}, "ttl": 8, "m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402"}'
DEBUG: TX done.
INFO: Message received: {"ttl":9,"values":{"humidity":79,"temperature":13}},{"m_id":"1710aa99-16ac-4783-84cb-9c0a9e01f402"},"d_id":"ID_e66118c4e336192a"}}
INFO: Message forwarded: b'{"d_id": "ID_e66118c4e336192a", "values": {"humidity": 79, "temperature": 13}, "ttl": 6, "m_id": "ID_e66118c4e336192a"}'
DEBUG: TX done.
INFO: Message received: {"m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402"},"values": {"humidity":79,"temperature":13}},{"m_id":"1710aa99-16ac-4783-84cb-9c0a9e01f402"},"d_id":"ID_e66118c4e336192a"}}
INFO: Message forwarded: b'{"d_id": "ID_e66118c4e336192a", "values": {"humidity": 79, "temperature": 13}, "ttl": 5, "m_id": "ID_e66118c4e336192a"}'
INFO: Message received: {"m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402"},"values": {"humidity":79,"temperature":13}},{"m_id":"1710aa99-16ac-4783-84cb-9c0a9e01f402"},"d_id":"ID_e66118c4e336192a"}}
INFO: Message forwarded: b'{"d_id": "ID_e66118c4e336192a", "values": {"humidity": 79, "temperature": 13}, "ttl": 4, "m_id": "ID_e66118c4e336192a"}'
DEBUG: TX done.
INFO: Message received: {"ttl":1,"values": {"humidity":79,"temperature":13}},{"m_id":"1710aa99-16ac-4783-84cb-9c0a9e01f402"},"d_id":"ID_e66118c4e336192a"}}
INFO: Message forwarded: b'{"d_id": "ID_e66118c4e336192a", "values": {"humidity": 79, "temperature": 13}, "m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402"},"d_id": "ID_e66118c4e336192a"}'
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44ebab125754c", "m_id": "60de50d6-5086-4d83-820f-955c41877562"}, "ttl":18,"values": {"random_values":32,"random_values2":68}}
INFO: Message forwarded: b'{"values": {"random_values": 68, "random_values2": 68}, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44ebab125754c"}'
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44ebab125754c", "ttl": 8, "values": {"random_values": 32, "random_values2": 68}, "m_id": "60de50d6-5086-4d83-820f-955c41877562"}, "ttl": 0, "m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402"}
INFO: Message forwarded: b'{"values": {"random_values": 32, "random_values2": 68}, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44ebab125754c"}'
INFO: Message received: {"d_id": "ID44ebab125754c", "ttl": 4, "values": {"random_values": 32, "random_values2": 68}, "m_id": "60de50d6-5086-4d83-820f-955c41877562"}, "ttl": 3, "values": {"random_values": 68, "random_values2": 32}, "d_id": "ID44ebab125754c"}
INFO: Message forwarded: b'{"m_id": "60de50d6-5086-4d83-820f-955c41877562", "ttl": 3, "values": {"random_values": 68, "random_values2": 32}, "d_id": "ID44ebab125754c"}'
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44ebab125754c", "ttl": 2, "m_id": "60de50d6-5086-4d83-820f-955c41877562"}, "values": {"random_values": 32, "random_values2": 68}, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44ebab125754c"}
INFO: Message forwarded: b'{"values": {"random_values": 68, "random_values2": 32}, "ttl": 1, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44ebab125754c"}'
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44ebab125754c", "ttl": 0, "values": {"random_values": 32, "random_values2": 68}, "m_id": "60de50d6-5086-4d83-820f-955c41877562"}, "ttl": 0, "values": {"random_values": 68, "random_values2": 32}, "d_id": "ID44ebab125754c", TTL is 0
WARNING: Message dropped: {"m_id": "60de50d6-5086-4d83-820f-955c41877562", "ttl": 0, "values": {"random_values": 68, "random_values2": 32}, "d_id": "ID44ebab125754c"}, TTL is 0

```

Rysunek 4.6: Przykładowy błędny pakiet odebrany przez jeden z węzłów sieci

### 4.2.3 Zauważone problemy

W trakcie testów zostały zauważone następujące problemy:

- Węzły sieci czasami odbierały błędne pakiety, jak na rysunku 4.7. Problem ten został rozwiązyany przez sprawdzanie czy odebrany pakiet jest poprawny, a w przypadku błędu, pakiet jest odrzucany.
- Przekaźnik sieci czasami przekazuje błędne pakiety do brokera. Problem ten został rozwiązyany przez sprawdzanie czy odebrany pakiet jest poprawny na poziomie programu zapisującego do bazy danych, a w przypadku błędu, pakiet jest odrzucany. Można byłoby sprawdzać poprawność pakietu na poziomie przekaźnika, ale obniżyłoby to wydajność systemu.

Na wykresie ?? widać

```
2023-05-06 11:45:25 INFO      Message already processed
2023-05-06 11:46:45 WARNING   Received message: 23RXF2W>kuxkU>RZ#
2023-05-06 11:46:45 ERROR     Invalid json message
```

Rysunek 4.7: Przykładowy błędny pakiet odebrany przez jeden z węzłów sieci



## Rozdział 5

### Wnioski i perspektywy rozwoju



# Spis listingów

3.1 Przykładowa wiadomość przesyłana przez system . . . . .	18
---	----



# Spis tabel



# Spis rysункów

3.1	Diagram prezentujący schemat ideowy systemu . . . . .	17
4.1	Zdjęcie węzła sieci opartego o Raspberry Pi Pico wraz z baterią . . . . .	24
4.2	Zdjęcie węzła sieci opartego o TTGO LoRa32 . . . . .	24
4.3	Zdjęcie stacji przekaźnikowej opartej o TTGO LoRa32 . . . . .	25
4.4	Wykres temperatury od czasu dla czujników DHT11 i DS18B20 . . . . .	27
4.5	Wykres wilgotności od czasu(w okresie od 2 Maja 2023 do 7 Maja 2023)	27
4.6	Przykładowy błędny pakiet odebrany przez jeden z węzłów sieci . . . . .	28
4.7	Przykładowy błędny pakiet odebrany przez jeden z węzłów sieci . . . . .	29



# Bibliography

- [1] ietf. *A Universally Unique IDentifier (UUID) URN Namespace*. 2005. (Visited on 04/23/2023).
- [2] Semtech. *SX1261/2 Low Power Long Range Transceiver*. 2019. URL: [https://www.waveshare.com/w/upload/e/e1/DS\\_SX1261-2\\_V1.2.pdf](https://www.waveshare.com/w/upload/e/e1/DS_SX1261-2_V1.2.pdf) (visited on 04/16/2023).
- [3] Waveshare. *Waveshare SX1262 LoRa Node Module 868MHz*. 2019. URL: <https://www.waveshare.com/wiki/Pico-LoRa-SX1262> (visited on 04/16/2023).
- [4] knolleary. *pubsubclient*. 2020. URL: <https://github.com/knolleary/pubsubclient> (visited on 04/16/2023).
- [5] Semtech. *SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver*. 2020. URL: <https://semtech.my.salesforce.com/sfc/p/#E0000000Je1G/a/2R0000001Rbr/6EfVZUorrpoKFfvaF-Fkpgp5kzjiNyiaBqcpqh9qSjE> (visited on 04/16/2023).
- [6] adafruit. *Adafruit\_SSD1306*. 2023. URL: [https://github.com/adafruit/Adafruit\\_SSD1306](https://github.com/adafruit/Adafruit_SSD1306) (visited on 04/16/2023).
- [7] bblanchon. *ArduinoJson*. 2023. URL: <https://github.com/bblanchon/ArduinoJson> (visited on 04/16/2023).
- [8] Docker. *Docker Documentation*. 2023. URL: <https://docs.docker.com/> (visited on 04/23/2023).
- [9] Eclipse. *Paho MQTT Python Client*. 2023. URL: <https://www.eclipse.org/paho/> (visited on 04/23/2023).
- [10] ehong-tl. *micropySX126X*. 2023. URL: <https://github.com/ehong-tl/micropySX126X> (visited on 04/16/2023).
- [11] espressif. *arduino-esp32*. 2023. URL: <https://github.com/espressif/arduino-esp32> (visited on 04/16/2023).

- [12] Raspberry Pi Foundation. *Raspberry Pi Documentation*. 2023. URL: <https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html> (visited on 04/16/2023).
- [13] Raspberry Pi Foundation. *Raspberry Pi Pico Datasheet*. 2023. URL: <https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf> (visited on 04/16/2023).
- [14] InfluxData. *InfluxDB Documentation*. 2023. URL: <https://docs.influxdata.com/influxdb/v2.7/> (visited on 04/23/2023).
- [15] influxdata. *InfluxDB Python client library*. 2023. URL: <https://docs.influxdata.com/influxdb/v2.7/api-guide/client-libraries/python/> (visited on 04/23/2023).
- [16] moritz89. *ESPRandom*. 2023. URL: <https://github.com/moritz89/ESPRandom> (visited on 04/16/2023).
- [17] Eclipse Mosquitto. *Eclipse Mosquitto*. 2023. URL: <https://mosquitto.org/> (visited on 04/23/2023).
- [18] PlatformIO. *PlatformIO Documentation*. 2023. URL: <https://docs.platformio.org/en/latest/> (visited on 04/23/2023).
- [19] Redis. *redis-py*. 2023. URL: <https://redis.io/docs/clients/python/> (visited on 04/23/2023).
- [20] sandeepmistry. *arduino-LoRa*. 2023. URL: <https://github.com/sandeepmistry/arduino-LoRa> (visited on 04/16/2023).
- [21] STMicroelectronics. *STM32 Overview*. 2023. (Visited on 04/23/2023).
- [22] Espressif Systems. *ESP32 Datasheet*. 2023. URL: [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf) (visited on 04/16/2023).
- [23] Espressif Systems. *ESP32 SoCs*. 2023. URL: <https://www.espressif.com/en/products/socs> (visited on 04/16/2023).