Akademia Techniczno-Informatyczna w Naukach Stosowanych we Wrocławiu

Skrypt do zajęć Projekt zespołowy (Hackaton IoT)

Paweł Dobrowolski

Wrocław 2024

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

Spis treści

1. Wstęp

Podczas opracowywania, konstruowania oraz oprogramowywania rzeczywistych układów działających w jednym spójnym systemie wymieniającym informacje można napotkać wiele problemów związanych z jego uruchomieniem. Działając na różnych platformach wymagana jest znajomość środowiska programistycznego, języka programowania czy systemu operacyjnego, na którym dokonuje się uruchomienia. Zdobycie wspomnianych umiejętności wymaga czasu oraz stałego doskonalenia się. Z pomocą przychodzi dokumentacja techniczna oraz różnego typu fora branżowe, których członkowie wzajemnie sobie pomagają.

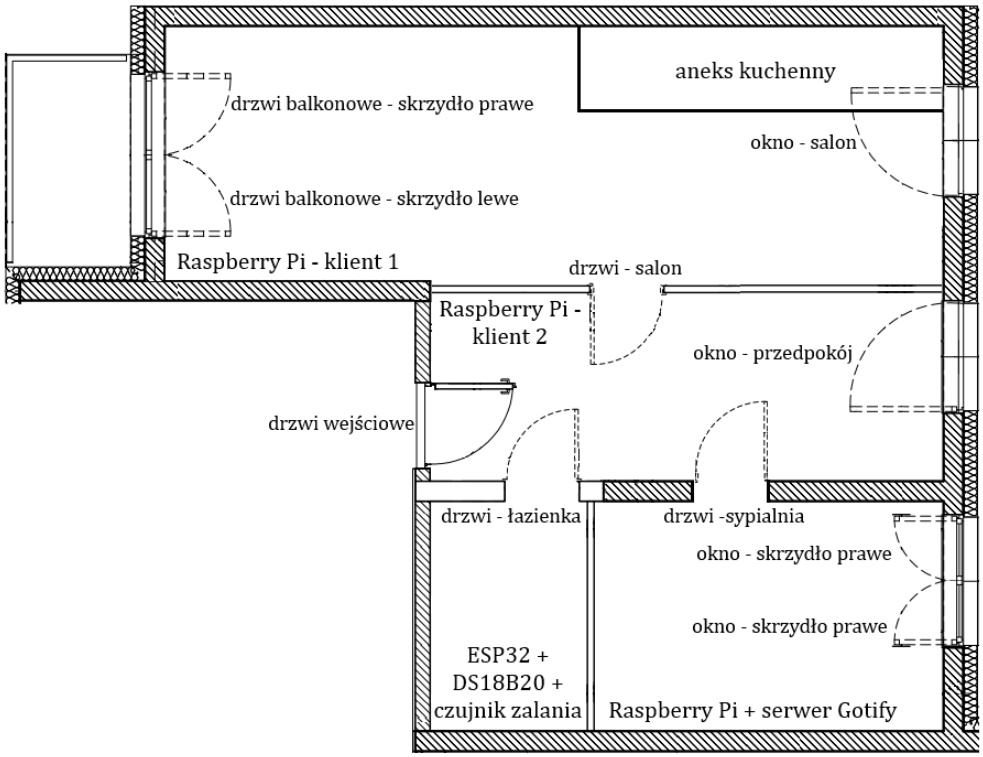
Popularność wybranych rozwiązań oraz szeroka dostępność bibliotek pomaga w uruchamianiu skomplikowanych systemów. W przypadku Raspberry Pi OS konieczna jest podstawowa znajomość obsługi systemu Linux oraz jego komend, ponieważ większość operacji wykonywanych jest z poziomu terminala (połączenie po protokole SSH).

W niniejszym opracowaniu przedstawiono projekt oraz konfigurację systemu alarmowego w mieszkaniu dwupokojowym z wykorzystaniem serwera Gotify rozsyłającego wiadomości typu push. Skonfigurowano jedno urządzenie Raspberry Pi jako serwer Gotify. Urządzenie to jednocześnie monitoruje stan otwarcia drzwi oraz okien w jednym z pomieszczeń. Dodatkowo w dwóch pozostałych pomieszczeniach skonfigurowano dwóch klientów (po jednym na pomieszczenie), którzy monitorują stan otwarcia drzwi oraz okien. W łazience skonfigurowano czujnik zalania oraz zbyt wysokiej temperatury w oparciu o moduł ESP32-WROOM-32 oraz platformę Arduino. Cały system nadzorowany jest z poziomu aplikacji mobilnej, za pomocą której można uzbroić lub rozbroić alarm.

2. Projekt systemu alarmowego z wykorzystaniem serwera Gotify

Omawiany z niniejszym opracowaniu projekt systemu alarmowego z wykorzystaniem serwera Gotify został dostosowany do przykładowego mieszkania dwupokojowego składającego się z sypialni, łazienki, salonu z aneksem kuchennym oraz balkonu. W każdym z pomieszczeń, tj. salon, sypialnia, przedpokój oraz kuchnia zamontowano urządzenia sygnalizujące otwarcie okien lub drzwi. Dla zwiększenia bezpieczeństwa oczujnikowano każde skrzydło, które można otworzyć.

System oparto na trzech minikomputerach Raspberry Pi 4B oraz jednym module ESP-WROOM-32. Minikomputery umieszczono w sypialni, przedpokoju oraz salonie. Do każdego podłączono czujniki otwarcia drzwi oraz okien. Dodatkowo na minikomputerze znajdującym się w sypialni zainstalowano i uruchomiono serwer Gotify, który obsługuje komunikację oraz jest odpowiedzialny za wysyłanie i odbieranie powiadomień. Moduł ESP-WROOM-32umieszczono w łazience. Jego zadaniem jest sygnalizacja zalania mieszkania oraz podwyższonej temperatury powietrza, co może świadczyć o potencjalnym źródle ognia lub pożarze.



Rysunek 1 Przykładowy układ mieszkania

Urządzenia współpracujące ze sobą wymieniają przez serwer Gotify informacje stosując metodę *jeden do wielu.* Oznacza to, że wiadomość przesłana przez *klienta 1* dociera do serwera oraz pozostałych klientów, a także aplikacji na smartphone. Z poziomu aplikacji mobilnej można uzbroić alarm lub go rozbroić wysyłając wiadomość push do wszystkich urządzeń o poniższej treści:

Tytuł wiadomości: *Ochrona*

Zawartość wiadomości (content): *ON* lub *OFF*

Zależnie od przesłanego argumentu (ON lub OFF) alarm uzbraja się lub jest rozbrajany. W przypadku rozbrojenia alarmu powiadomienia o otwarciu drzwi oraz okien nie są rozsyłane. Alarmy dotyczące zalania oraz zbyt wysokiej temperatury są aktywne bez względu na status ochrony i nie można ich wyłączyć.

2.1. Dobór elementów systemu

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

Minikomputer Raspberry Pi 4

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

Moduł mikrokontrolera z Wi-Fi ESP32-DevKitC V4

- Komunikacja bezprzewodowa: Wi-Fi w paśmie 2,4 GHz i standardzie 802.11b/g/n; Bluetooth, BLE 4.2
- Ilość wyprowadzeń GPIO: 34
- Interfejsy komunikacyjne: UART, I2C, SPI, SDIO, PWM, I2S, ADC, DAC
- Taktowanie: do 240MHz
- Pamięć ROM: 448 KB
- Pamięć SRAM: 520 KB
- Pamięć Flash SPI: 4 MB
- Napięcie zasilania: 3 – 3.6V DC

Czujnik temperatury DS18B20

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

Kontaktron

- Zasięg działania: 23 mm
- Konfiguracja styków: NO
- Prąd przełączany: < 250 mA

Detektor wody FC-37

- Zasilanie: 3.0 – 5V DC
- Wyjście cyfrowe: TTL
- Powierzchnia czujnika: 5 x 4 [cm]

Rysunek 2 Czujnik temperatury DS18B20

Rysunek 3 Moduł ESP32-DevKitC V4

Rysunek 4 Raspberry Pi 4

Rysunek 5 Kontaktron

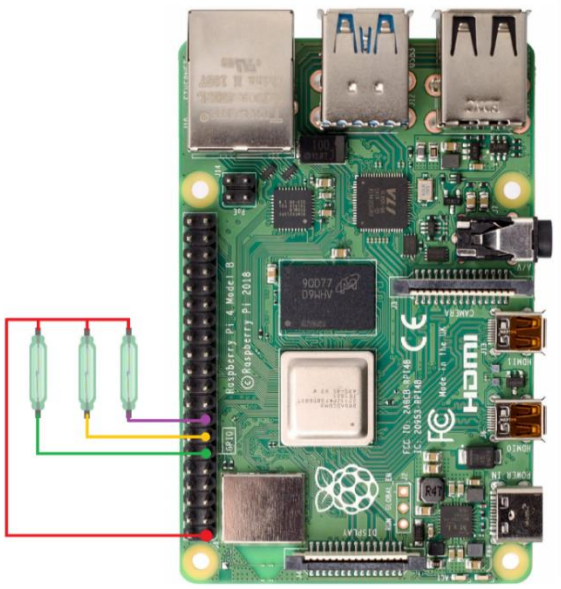
Rysunek 6 Detektor wody

2.2. Schemat ideowy

Na szkicu poniżej zaprezentowano schemat połączeń czujników otwarcia (kontaktronów) z minikomputerem Raspberry Pi 4B dla pomieszczeń: sypialnia i przedpokój. Kontaktrony zostały podłączone w następujący sposób:

GPIO17 – okno – część lewa – sypialnia
GPIO27 – okno – część prawa – sypialnia
GPIO22 – drzwi – sypialnia

GPIO17 – okno – przedpokój
GPIO27 – drzwi – łazienka
GPIO22 – drzwi – wejście do mieszkania



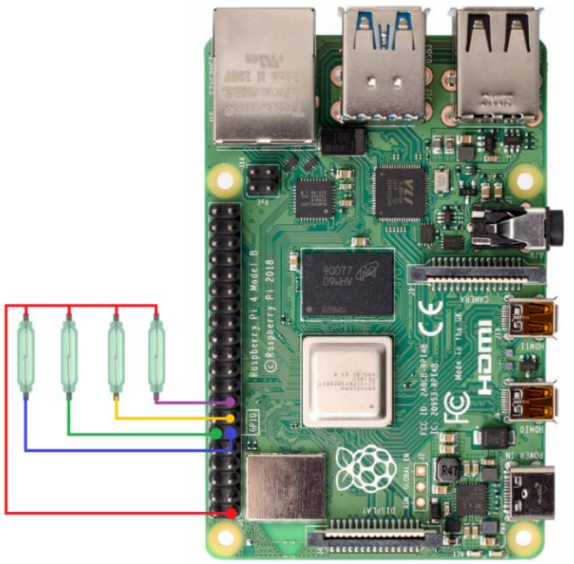
Rysunek 7 Schemat podłączenia Raspberry Pi

Na szkicu poniżej zaprezentowano schemat połączeń czujników otwarcia (kontaktronów) z minikomputerem Raspberry Pi 4 dla salonu. Kontaktrony zostały podłączone w następujący sposób:

GPIO17 – drzwi balkonowe – część lewa – salon
GPIO27 – drzwi balkonowe – część prawa – salon

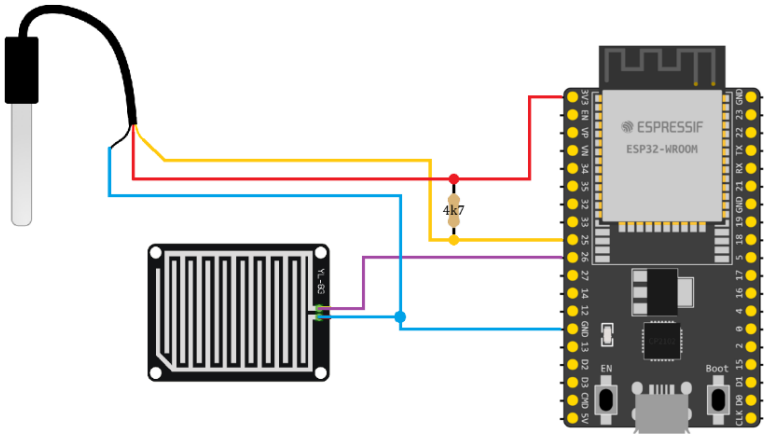
GPIO18 – okno – salon

GPIO22 – drzwi – salon



Rysunek 8 Schemat podłączenia Raspberry Pi - salon

Na szkicu poniżej zaprezentowano schemat połączeń modułów elektronicznych z modułem mikrokontrolera ESP-WROOM-32. Wszystkie moduły zasilane są napięciem 3.3V (pin 3V3). Czujnik DS18B20 podłączony jest do pinu 25, a czujnik zalania (obecności wody) do pinu 26.



Rysunek 9 Detektor zalania i pożaru

2.3. Konfiguracja serwera Gotify

Aby uruchomić serwer Gotify należy przeprowadzić jego instalację. W tym celu po uruchomieniu i zalogowaniu się na Raspberry Pi w terminalu należy wpisać poniższą komendę:

wget https://github.com/gotify/server/releases/download/v2.5.0/gotify-linux-arm-7.zip

Po jej wykonaniu pliki instalatora zostaną pobrane do folderu *opt,* skąd przebiegać będzie dalsza instalacja. Poleceniem:

cd /opt

wchodzimy do folderu, a wykonując polecenie *ls* upewniamy się, czy w danym katalogu znajduje się pobrany instalator (rozszerzenie.zip).

```
pi@raspberrypi:~ $ cd /opt
pi@raspberrypi:/opt $ ls
gotify  gotify-linux-arm-7.zip  pigpio  WidevineCdm
pi@raspberrypi:/opt $
```

Rysunek 10 Spakowane pliki instalatora Gotify

Aby rozpakować folder *gotify-linux-arm-7.zip* należy w terminal wpisać poniższą komendę:

sudo unzip gotify-linux-arm-7.zip -d gotify

oraz dodać uprawnienia dla użytkowników poleceniem:

sudo chmod +x gotify-linux-arm-7

Następnie należy uruchomić autostart usługi na serwerze. W tym celu należy w pliku konfiguracyjnym usługi */lib/systemd/system/gotify.service* zmodyfikować kod jak na rysunku poniżej. Aby móc edytować plik należy otworzyć go w edytorze np. nano za pomocą poniższej komendy:

nano /lib/systemd/system/gotify.service

```
 GNU nano 7.2                          /lib/systemd/system/gotify.service
[Unit]
#Gotify Service (LetsFOSS.com)
Description=Gotify Starter
After=network-online.target
Requires=network-online.target

[Service]
WorkingDirectory=/opt/gotify/
ExecStart=/opt/gotify/gotify-linux-arm-7
Restart=always

[Install]
WantedBy=multi-user.target
```

Rysunek 11 Modyfikacja pliku gotify.service

Po modyfikacji pliku i zapisaniu zmian należy przeładować usługę za pomocą komendy:

systemctl daemon-reload

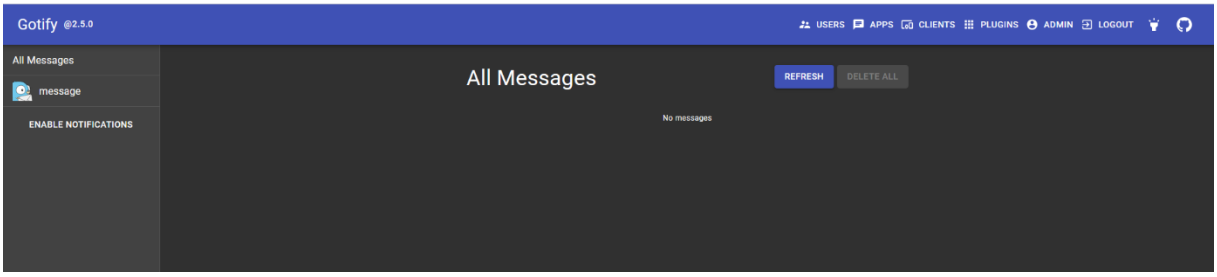
Kolejnym etapem jest uruchomienie usługi komendą:

systemctl start gotify

Aby uruchomić wersję graficzną serwera Gotify należy uruchomić przeglądarkę i wpisać poniższy adres:

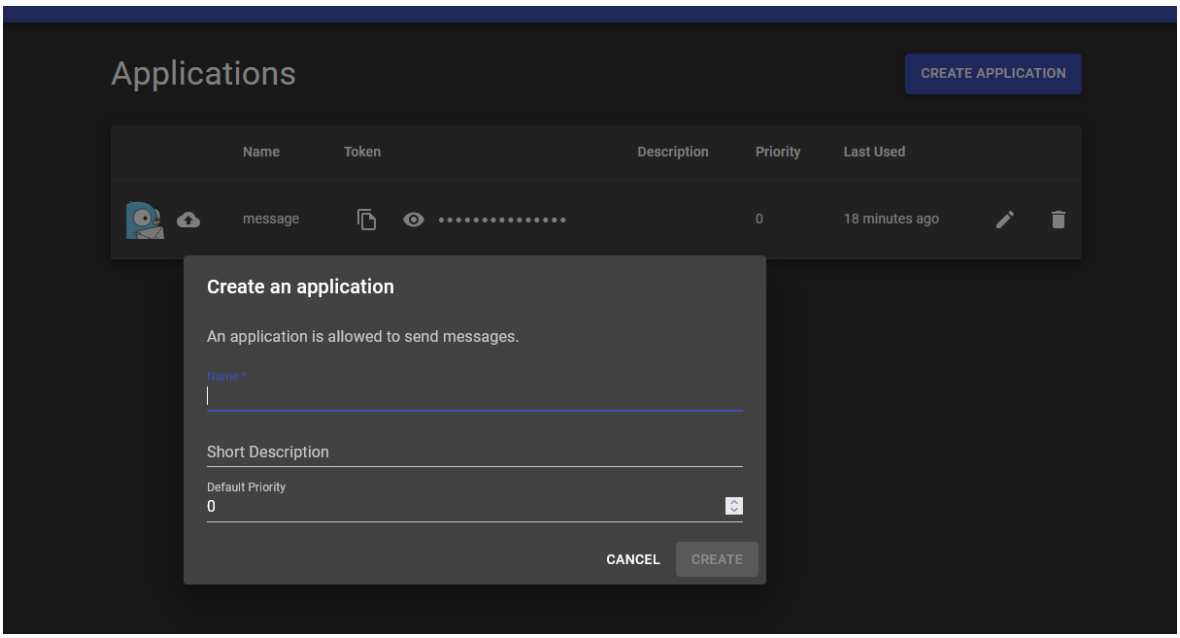
http://192.165.31.161:80

Wówczas pojawi się okno serwera, które zaprezentowano na poniższym rysunku.



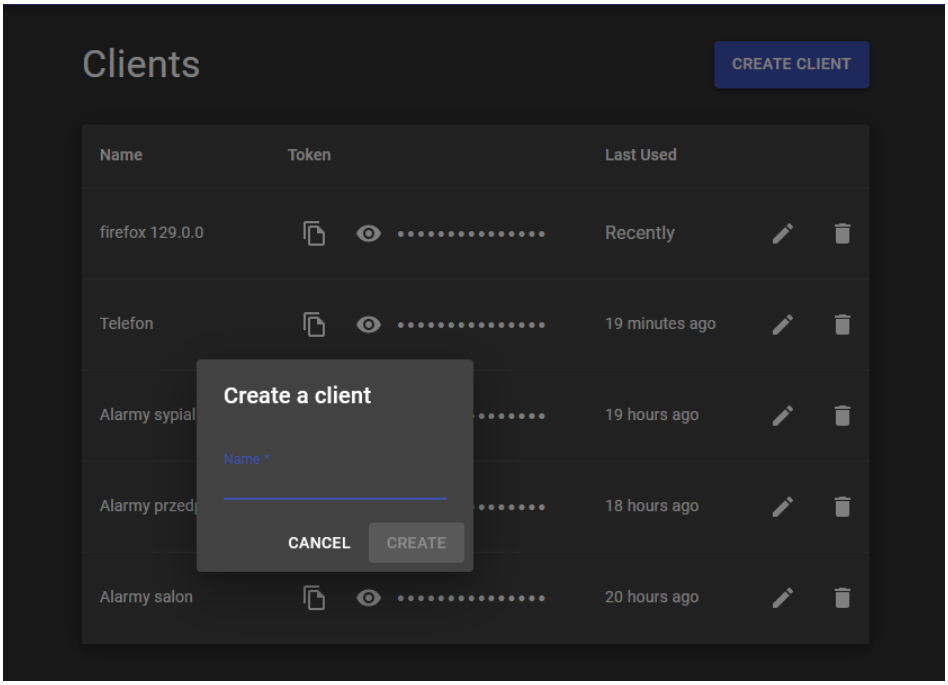
Rysunek 12 Serwer Gotify

Po uruchomieniu serwera należy stworzyć nową aplikację. W tym celu należy kliknąć na pole *Apps* znajdujące się na niebieskim pasku u góry ekranu. Pojawi się wówczas okno *Applications*, które umożliwia tworzenie nowych aplikacji oraz podgląd już istniejących. W celu stworzenia nowej aplikacji wybieramy opcję *Create application,* nadajemy jej własną nazwę oraz klikamy przycisk *Create.*



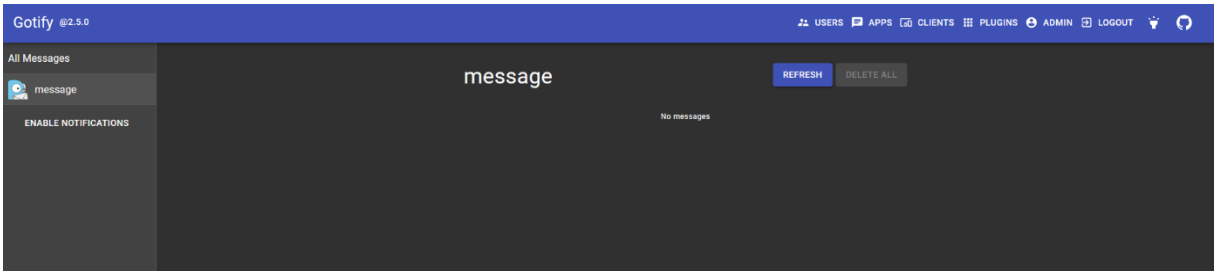
Rysunek 13 Tworzenie aplikacji w Gotify

Po stworzeniu aplikacji należy przejść do zakładki *Clients*. W tym celu należy kliknąć na pole *Clients* znajdujące się na niebieskim pasku u góry ekranu. Pojawi się wówczas okno *Clients*, które umożliwia tworzenie nowych klientów oraz podgląd już istniejących. W celu stworzenia nowego klienta wybieramy opcję *Create a client,* nadajemy jego własną nazwę oraz klikamy przycisk *Create.* W celu poprawnego działania systemu należy stworzyć tylu klientów, ile jest urządzeń w systemie. Dla każdego z nich zostanie wygenerowany indywidualny i niepowtarzalny token, dzięki któremu możliwe będzie połączenie z serwerem i przesyłanie lub odbieranie danych.

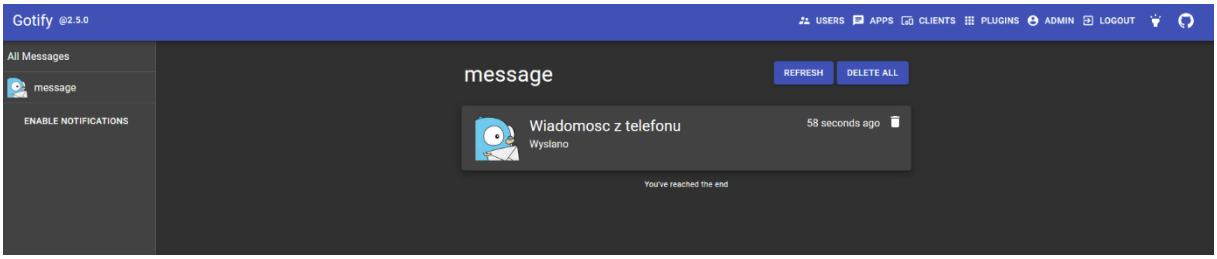


Rysunek 14 Dodawanie klientów w Gotify

Po skonfigurowaniu serwera należy powrócić na stronę główną, na której wyświetlane są wszystkie otrzymane dane w formie powiadomień.



Rysunek 15 Okno powiadomień



Rysunek 16 Powiadomienia na serwerze Gotify

2.4. Aplikacja mobilna

Do przesyłania i odbierania informacji ze smartphona wymagana jest instalacja aplikacji mobilnej Gotify (w przypadku systemu Android darmowa aplikacja jest dostępna w sklepie Play). Po jej uruchomieniu należy podać adres URL serwera:

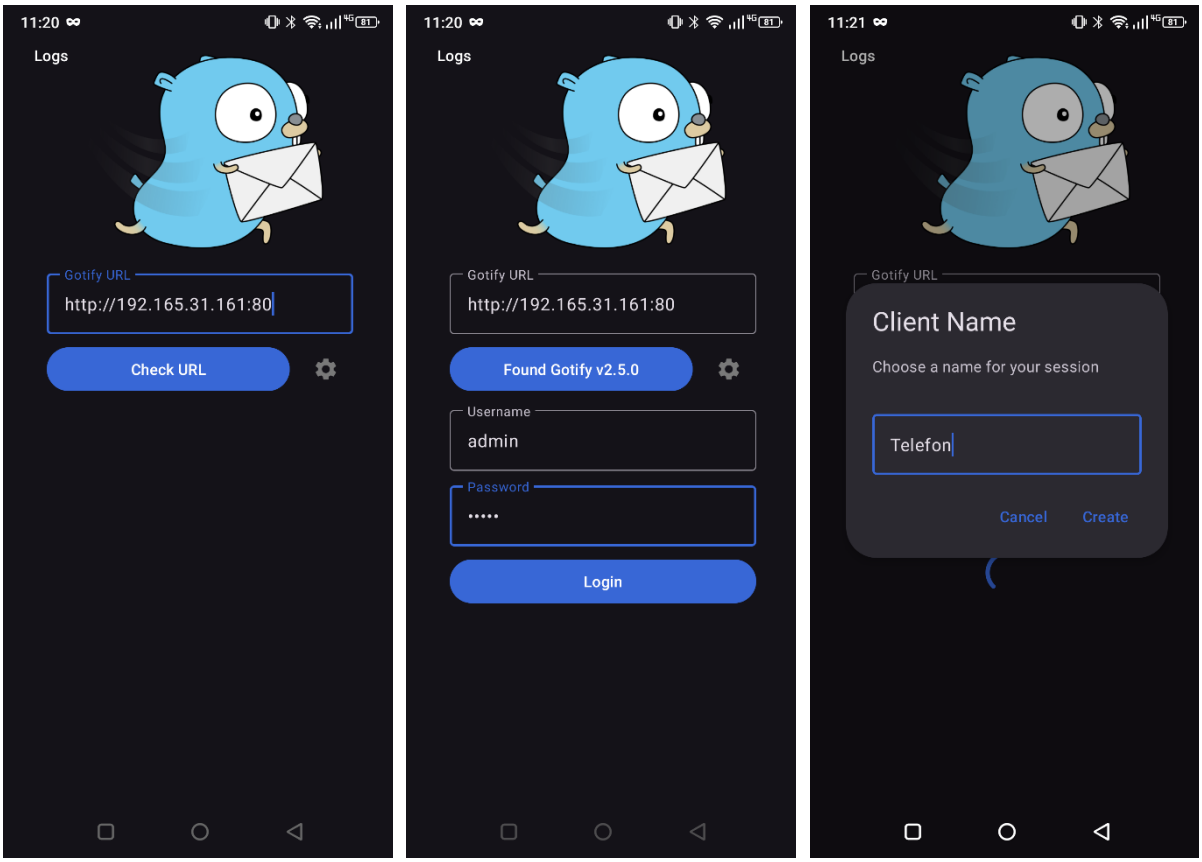
http://192.165.31.161:80

i go zweryfikować klikając *Check URL.* Po odnalezieniu serwera Gotify w sieci należy podać dane logowania, tj.

Username: admin

Password: admin

Po zalogowaniu należy ustawić nazwę klienta *Client name* i ją zatwierdzić przyciskiem *Create.*



Rysunek 17 Połączenie i konfiguracja klienta w aplikacji mobilnej

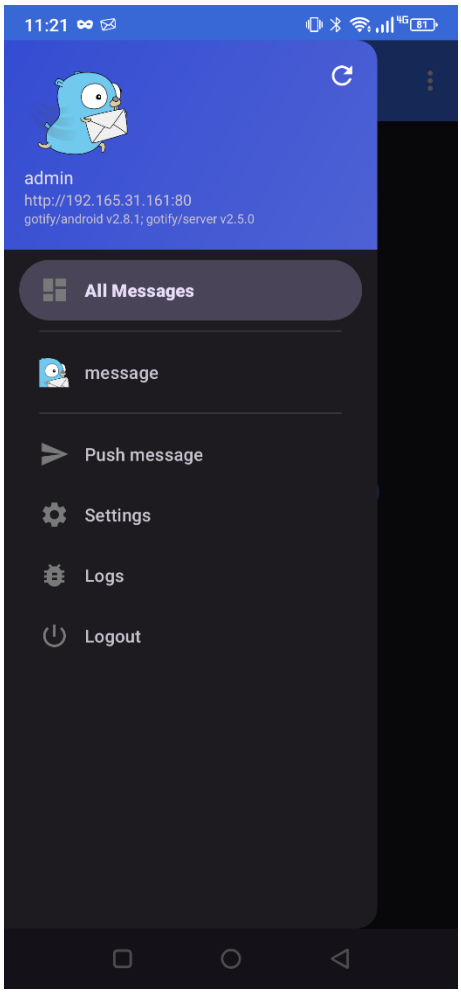
Po konfiguracji klienta mamy dostęp do panelu powiadomień (zakładka *All Messages*) lub możemy wysłać własne powiadomienie. Aby wysłać powiadomienie na serwer należy kliknąć na pole *Push message.* Pojawi się wówczas okienko, które umożliwia wysłanie wiadomości na

serwer. Żeby było to możliwe w wiadomości należy podać tytuł, content oraz priorytet. Aby uzbroić alarm lub go rozbroić należy wysłać wiadomość push do wszystkich urządzeń o poniższej treści:

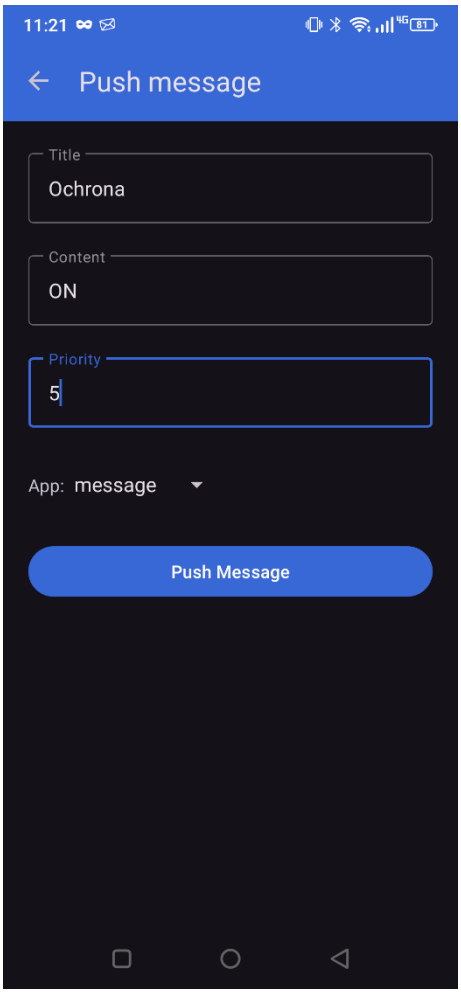
Tytuł wiadomości: *Ochrona*

Zawartość wiadomości (content): *ON* lub *OFF*

Zależnie od przesłanego argumentu (ON lub OFF) alarm uzbraja się lub jest rozbrajany. W przypadku rozbrojenia alarmu powiadomienia o otwarciu drzwi oraz okien nie są rozsyłane.



Rysunek 18 Opcja Push message

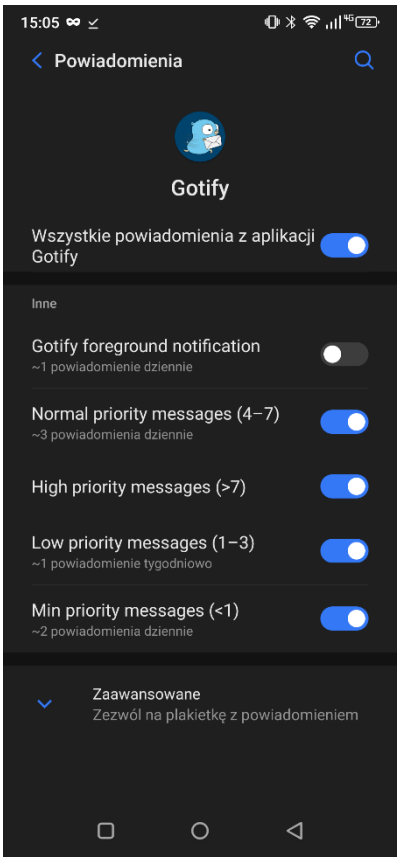


Rysunek 19 Przesłanie wiadomości push

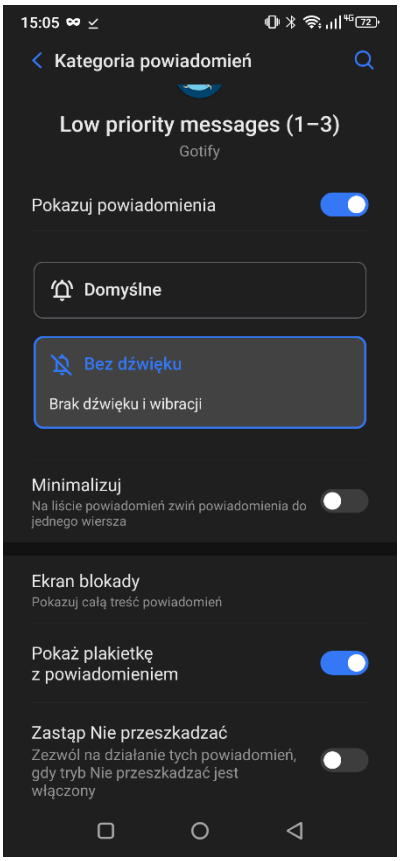
Jednym z przesyłanych parametrów w każdej przesyłanej wiadomości jest jej priorytet. W zależności od jego wartości powiadomienia pojawiają się na pasku powiadomień systemu Android lub tylko w samej aplikacji mobilnej Gotify. Wyróżnia się trzy rodzaje priorytetów:

- niski – wartość 1-3,
- normalny – wartość 4-7,
- wysoki – wartość powyżej 7.

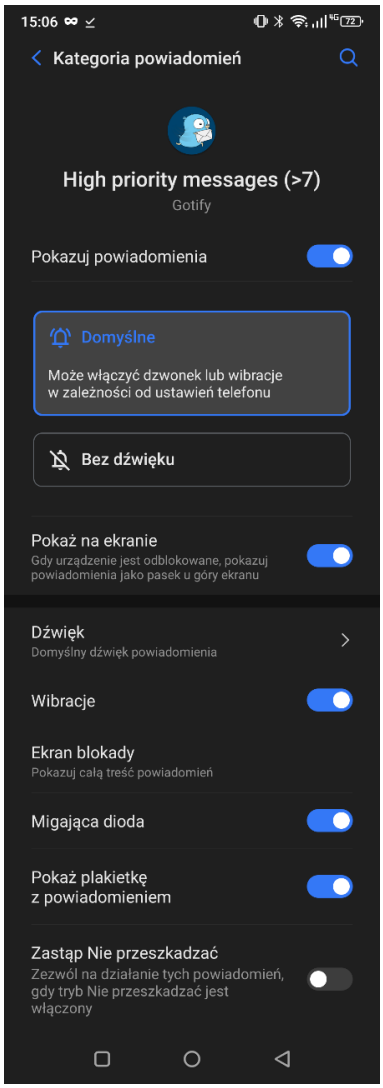
Dodatkowo aplikacja umożliwia dostosowanie zachowania telefonu w zależności od określonego priorytety, co przedstawiono na rysunkach poniżej.



Rysunek 20 Priorytety w aplikacji



Rysunek 21 Ustawienia niskiego priorytetu



Rysunek 22 Ustawienia wysokiego priorytetu

2.5. Opis programu

Przed uruchomieniem programów przedstawionych w niniejszym punkcie należy zainstalować wymagane biblioteki dla języka python. W tym celu należy doinstalować bibliotekę *gotify* poniższą komendą:

pip install gotify

---------- Oprogramowanie – serwer Gotify na Raspberry Pi ----------

Import bibliotek obsługujących serwer Gotify oraz sterowanie pinami GPIO:

```
import asyncio
import RPi.GPIO as GPIO
from gotify import AsyncGotify
```

Konfiguracja pinów obsługujących czujniki otwarcia drzwi i okien:

```
window1_pin = 17
window2_pin = 27
door_pin = 22
alarm_state = 1
```

Definicja klasy oraz metod odpowiedzialnych za połączenie z serwerem. Token został wygenerowany przez serwer Gotify i tutaj należy go wpisać (każde urządzenie ma swój indywidualny token):

```
class GotifyConnectionHandler:
    def __init__(self):
        self.async_gotify = AsyncGotify(base_url="http://192.165.31.161",
client_token="Cmhgr10nrQrgA_S", app_token="AKIXR957wFVsLz6")
```

Definicja funkcji odbierającej wiadomości z serwera. W przypadku odebrania wiadomości ochrona sprawdzany jest jej status (ON lub OFF). Warunkuje to o załączeniu lub wyłączeniu alarmu (zmienna *alarm_state*):

```
async def log_push_messages(self):
    async for msg in self.async_gotify.stream():
        if msg["title"] == "Ochrona":
            global alarm_state
            if msg["message"] == "ON":
                alarm_state = 1
            elif msg["message"] == "OFF":
                alarm_state = 0
```

Definicja funkcji wysyłającej wiadomości:

```
async def send_push_message(self, title, message):
    await self.async_gotify.create_message(message, title=title, priority=5)
```

Definicja funkcji przerwania w przypadku otwarcie drzwi lub okna:

```
def button_pressed_callback(self, channel):
    if alarm_state == 1:
        source = ""
        if channel == window1_pin:
            source = "Okno lewe"
        if channel == window2_pin:
            source = "Okno prawe"
        if channel == door_pin:
            source = "Drzwi od sypialni"
```

Wysłanie wiadomości z informacją o otwartych drzwiach lub oknie:

```
        asyncio.run(self.send_push_message("Alarm Sypialnia!", source))
```

Konfiguracja przerwań na pinach GPIO oraz ich wywołanie:

```
handler = GotifyConnectionHandler()
GPIO.setmode(GPIO.BCM)
GPIO.setup(window1_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(window2_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(door_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.add_event_detect(window1_pin, GPIO.FALLING, callback=handler.button_pressed_callback, bouncetime=100)
GPIO.add_event_detect(window2_pin, GPIO.FALLING, callback=handler.button_pressed_callback, bouncetime=100)
GPIO.add_event_detect(door_pin, GPIO.FALLING, callback=handler.button_pressed_callback, bouncetime=100)
```

Oczekiwanie na nowe wiadomości:

```
asyncio.run(handler.log_push_messages())
```

---------- Oprogramowanie - klient 1 Raspberry Pi ----------

Import bibliotek obsługujących serwer Gotify oraz sterowanie pinami GPIO:

```
import asyncio
import RPi.GPIO as GPIO
from gotify import AsyncGotify
```

Konfiguracja pinów obsługujących czujniki otwarcia drzwi i okien:

```
window1_pin = 17
window2_pin = 27
window3_pin = 18
door_pin = 22
alarm_state = 1
```

Definicja klasy oraz metod odpowiedzialnych za połączenie z serwerem. Token został wygenerowany przez serwer Gotify i tutaj należy go wpisać (każde urządzenie ma swój indywidualny token):

```
class GotifyConnectionHandler:
    def __init__(self):
        self.async_gotify = AsyncGotify(base_url="http://192.165.31.161",
client_token="C7.8T0s14i94afq", app_token="AKIXR957wFVsLz6")
```

Definicja funkcji odbierającej wiadomości z serwera. W przypadku odebrania wiadomości ochrona sprawdzany jest jej status (ON lub OFF). Warunkuje to o załączeniu lub wyłączeniu alarmu (zmienna *alarm_state*):

```
    async def log_push_messages(self):
        async for msg in self.async_gotify.stream():
            if msg["title"] == "Ochrona":
                global alarm_state
```

```
        if msg["message"] == "ON":
            alarm_state = 1
        elif msg["message"] == "OFF":
            alarm_state = 0
```

Definicja funkcji wysyłającej wiadomości:

```
    async def send_push_message(self, title, message):
        await self.async_gotify.create_message(message, title=title, priority=5)
```

Definicja funkcji przerwania w przypadku otwarcie drzwi lub okna:

```
    def button_pressed_callback(self, channel):
        if alarm_state == 1:
            source = ""
            if channel == window1_pin:
                source = "Okno tylne"
            if channel == window2_pin:
                source = "Okno balkonowe prawe"
            if channel == window3_pin:
                source = "Okno balkonowe lewe"
            if channel == door_pin:
                source = "Drzwi od salonu"
```

Wysłanie wiadomości z informacją o otwartych drzwiach lub oknie:

```
            asyncio.run(self.send_push_message("Alarm Przedpokoj!", source))
```

Konfiguracja przerwań na pinach GPIO oraz ich wywołanie:

```
handler = GotifyConnectionHandler()
GPIO.setmode(GPIO.BCM)
GPIO.setup(window1_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

GPIO.setup(window2_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(window3_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(door_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.add_event_detect(window1_pin, GPIO.FALLING, callback=handler.button_pressed_callback, bouncetime=100)
GPIO.add_event_detect(window2_pin, GPIO.FALLING, callback=handler.button_pressed_callback, bouncetime=100)
GPIO.add_event_detect(window3_pin, GPIO.FALLING, callback=handler.button_pressed_callback, bouncetime=100)
GPIO.add_event_detect(door_pin, GPIO.FALLING, callback=handler.button_pressed_callback, bouncetime=100)

Oczekiwanie na nowe wiadomości:

asyncio.run(handler.log_push_messages())

---------- Oprogramowanie - klient 2 Raspberry Pi ----------

Import bibliotek obsługujących serwer Gotify oraz sterowanie pinami GPIO:

```
import asyncio
import RPi.GPIO as GPIO
from gotify import AsyncGotify
```

Konfiguracja pinów obsługujących czujniki otwarcia drzwi i okien:

```
window_pin = 17
door1_pin = 27
door2_pin = 22
alarm_state = 1
```

Definicja klasy oraz metod odpowiedzialnych za połączenie z serwerem. Token został wygenerowany przez serwer Gotify i tutaj należy go wpisać (każde urządzenie ma swój indywidualny token):

```
class GotifyConnectionHandler:
    def __init__(self):
        self.async_gotify = AsyncGotify(base_url="http://192.165.31.161", client_token="CKuE-
HnnYVwT-OA", app_token="AKIXR957wFVsLz6")
```

Definicja funkcji odbierającej wiadomości z serwera. W przypadku odebrania wiadomości ochrona sprawdzany jest jej status (ON lub OFF). Warunkuje to o załączeniu lub wyłączeniu alarmu (zmienna *alarm_state*):

```
    async def log_push_messages(self):
        async for msg in self.async_gotify.stream():
            if msg["title"] == "Ochrona":
                global alarm_state
                if msg["message"] == "ON":
                    alarm_state = 1
```

```
        elif msg["message"] == "OFF":
            alarm_state = 0
```

Definicja funkcji wysyłającej wiadomości:

```
    async def send_push_message(self, title, message):
        await self.async_gotify.create_message(message, title=title, priority=5)
```

Definicja funkcji przerwania w przypadku otwarcie drzwi lub okna:

```
    def button_pressed_callback(self, channel):
        if alarm_state == 1:
            source = ""
            if channel == window_pin:
                source = "Okno"
            if channel == door1_pin:
                source = "Drzwi od lazienki"
            if channel == door2_pin:
                source = "Drzwi wejsciowe"
```

Wysłanie wiadomości z informacją o otwartych drzwiach lub oknie:

```
            asyncio.run(self.send_push_message("Alarm Przedpokoj!", source))
```

Konfiguracja przerwań na pinach GPIO oraz ich wywołanie:

```
handler = GotifyConnectionHandler()
GPIO.setmode(GPIO.BCM)
GPIO.setup(window_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(door2_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(door1_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.add_event_detect(window_pin, GPIO.FALLING, callback=handler.button_pressed_callback,
bouncetime=100)
```

GPIO.add_event_detect(door1_pin, GPIO.FALLING, callback=handler.button_pressed_callback, bouncetime=100)
GPIO.add_event_detect(door2_pin, GPIO.FALLING, callback=handler.button_pressed_callback, bouncetime=100)

Oczekiwanie na nowe wiadomości:

asyncio.run(handler.log_push_messages())

---------- *Oprogramowanie - klient ESP32* ----------

Wymagane biblioteki:

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

#define FLOOD_SENSOR_PIN 26
```

Dane połączenia sieciowego po Wi-Fi:

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

Konfiguracja czujnika temperatury:

```
DS18B20 ds(25);
int flooding_detected = 0;
int flooding_alert_sent = 0;
int temperature_alert_sent = 0;
unsigned long previousMillis = 0;
```

Funkcja przerwania od sensora zalania:

```
void IRAM_ATTR flood_interrupt_handler()
{
    if (digitalRead(FLOOD_SENSOR_PIN) == LOW)
        flooding_detected = 1;
    else
    {
        flooding_detected = 0;
```

```
        flooding_alert_sent = 0;
    }
}

void setup()
{
```

Inicjalizacja pinu zalania oraz funkcji przerwania:

```
    Serial.begin(115200);
    pinMode(FLOOD_SENSOR_PIN, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(FLOOD_SENSOR_PIN), flood_interrupt_handler,
    CHANGE);
```

Inicjalizacja Wi-Fi:

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

Pomiar temperatury co 2 sekundy. W przypadku zbyt wysokiej temperatury (ustawione 30 ºC) wysyłane jest powiadomienie o zbyt wysokiej temperaturze:

```
    if (currentMillis - previousMillis >= 2000L)
    {
```

```
        int temperature = ds.getTempC();
        if(temperature > 30 && !temperature_alert_sent)
        {
            post_notification("Wysoka temperatura");
            temperature_alert_sent = 1;
        }

        if(temperature < 29)
        temperature_alert_sent = 0;
```

Detekcja zalania mieszkania. W przypadku wykrycia zalania mieszkania wysyłane jest powiadomienie:

```
        if(flooding_detected && !flooding_alert_sent)
        {
            flooding_alert_sent = 1;
            post_notification("Zalanie mieszkania");
        }
        previousMillis = currentMillis;
    }
}
```

Konfiguracja połączenia z serwerem Gotify przez HTTP Client:

```
void post_notification(String msg)
{
    HTTPClient http;

    http.begin("http://192.165.31.161:80/message?token=AKIXR957wFVsLz6");

    http.addHeader("Content-Type", "application/json");
```

Scalenie i wysłanie wiadomości zgodnie z formatem serwera:

```
String message = "{\"message\": \"" + msg +  "\",\"title\": \"Alarm mieszkania!\",\"priority\": 5}";
Serial.println(message);

uint16_t httpResponseCode = http.POST(message.c_str());
```

Potwierdzenie odebrania wiadomości:

```
if(httpResponseCode>0)
{
    String response = http.getString();
    Serial.println(httpResponseCode);
    Serial.println(response);
}
else
{
    Serial.print("Error on sending POST: ");
    Serial.println(httpResponseCode);
}
http.end();
}
```

3. Wykaz literatury

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

[2]. Adam Jurkiewicz, „Python 3. Projekty dla początkujących i pasjonatów", Helion, 2021

[3]. Eric Matthes, „Python. Instrukcje dla programisty", Helion, 2021

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

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

[6]. Richardson Matt, Donat Wolfram, Wallace Shawn, „Wprowadzenie do Raspberry Pi", APN Promise, 2022

[7]. Monk Simon, Wallace Shawn, „Raspberry Pi. Przewodnik dla programistów Pythona", Helion, 2021

[8]. Negus Christopher, „Linux. Biblia", Helion, 2021

[9]. Sosna Łukasz, „Linux. Komendy i polecenia", Helion, 2022

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

[11]. Dokumentacja Gotify, dostęp 22.08.2024 https://lyz-code.github.io/blue-book/gotify/

[12]. Dokumentacja Gotify, dostęp 22.08.2024 https://gotify.net/

Spis rysunków