

Projekt

Analizator powietrza

Wykonał:

Dzieciuch Hubert

Spis treści

Wstęp	3
Opis użytych komponentów.....	3
ESP32-DevKitC	3
DHT11	4
Fotorezystor	5
Silnik	6
Sterownik silnika L298N	6
Analizator	7
Technologia bazy danych	8
Program - ESP32	8
Użyte biblioteki	8
Nawiązanie połączenia Wi-Fi.....	8
Połączenie z bazą danych	9
Odczyt czujników.....	9
Odczyt czasu z serwera NTP	9
Przesyłanie danych do bazy	10
Utworzona baza danych.....	10
Kod aplikacji	11
Wykorzystane biblioteki.....	11
Połączenie z bazą danych	11
Utworzenie GUI	11
Generowanie wykresów.....	12
Ukazanie działania systemu	13
Podsumowanie	15

Wstęp

Wykonany projekt jest to rezultat realizacji zadania projektowego z przedmiotu Systemy Wbudowane i Wieloagentowe.

Realizacja projektu polega na wykonaniu analizatora powietrza, czyli urządzenia składającego się z płytki ESP32, odpowiednich czujników oraz silnika. Zastosowane czujniki pozwalają na odczyt nasłonecznienia, wilgotności powietrza oraz temperatury. Dane z odczytów będą wysyłane do bazy danych znajdującej się w chmurze. Drugą częścią jest napisanie aplikacji desktopowej w języku Python, która będzie umożliwiać ukazanie historycznych odczytów danych oraz na ich podstawie generować wykres. Dodatkowo z poziomu aplikacji możemy sterować silnikiem imitującym pracę wentylatora. W aplikacji mamy możliwość wyboru kiedy ma załączyć się wentylator (wybieramy temperaturę otoczenia powyżej której załącza się silnik) oraz cztery poziomy prędkości.

Zakres prac jakie były realizowane w celu wykonania projektu:

- Zrealizowanie połączeń na płytce prototypowej
- Utworzenie bazy danych w chmurze
- Napisanie kodu programu dla mikrokontrolera
- Napisanie programu aplikacji desktopowej
- Testowanie działania
- Zbieranie odczytów w celu umożliwienia pokazania działania aplikacji

Opis użytych komponentów

ESP32-DevKitC



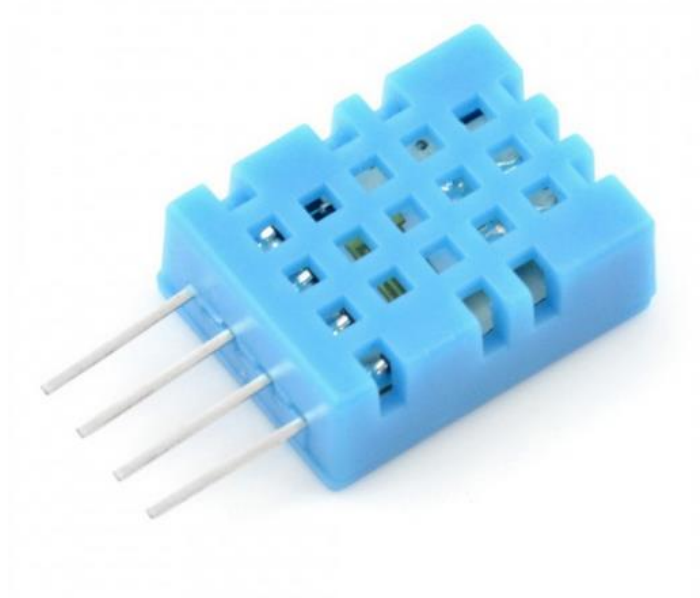
Rysunek 1 ESP32-DevKitC

Sercem elektronicznej części jest płytka ESP32-DevKitC z wbudowanym modulem ESP-WROOM-32.

Główne cechy platformy z modulem ESP-WROOM-32:

- Napięcie zasilania: 5 V - z microUSB
- Mikrokontroler Dual Core Tensilica LX6 240 MHz
- Pamięć SRAM 520 KB
- Pamięć Flash: 4 MB
- Wbudowany układ WiFi 802.11BGN HT40
- Zabezpieczenia WiFi: WEP, WPA/WPA2, PSK/Enterprise, AES / SHA2 / Elliptical Curve Cryptography / RSA-4096
- Wbudowany moduł Bluetooth BLE/v4.2
- Zintegrowany czujnik Halla oraz interfejs dotykowy
- 30 wyprowadzeń GPIO w tym:
 - 3x UART
 - 3x SPI
 - 2x I2C (2x I2S)
 - 12- kanałowy przetwornik ADC
 - 2- kanałowy przetwornik DAC
 - Wyjścia PWM
 - Interfejs kart SD
- Wymiary: 55 x 28 x 8 mm

DHT11



Rysunek 2 DHT11

DHT11 jest to popularny czujnik temperatury i wilgotności powietrza z interfejsem cyfrowym, jednoprzewodowym.

Parametry czujnika są następujące:

- Napięcie zasilania: 3,3 V do 5,5 V
- Średni pobór prądu: 0,2mA
- Temperatura
 - Zakres pomiarowy: do - 20°C do +60°C
 - Rozdzielczość: 8-bitów (1°C)
 - Dokładność: 2°C
 - Czas odpowiedzi: 6s - 15s (typowo 10s)
- Wilgotność:
 - Zakres pomiarowy: od 5% do 95% RH
 - Rozdzielczość: 8-bitów ($\pm 1\%$ RH*)
 - Dokładność ± 4 RH* (przy 25°C)
 - Zakres pomiarowy: 6s - 30 s

*RH – jest to wilgotność względna wyrażana procentach. Jest to stosunek rzeczywistej wilgoci w powietrzu do maksymalnej jej ilości, którą może utrzymać powietrze w danej temperaturze.

Fotorezystor

Fotoopornik służy do uzyskania odczytu natężenia światła. Jego rezystancja ulega zmianie pod wpływem padającego światła.



Rysunek 3 Fotorezystor

Silnik

Pracę wentylatora w projekcie imituje silnik DC prądu stałego bez przekładni.

Specyfikacja silnika:

- Napięcie zasilania: od 2 V do 5,5 V
- Prąd na biegu jałowym: 66 mA
- Obroty: ok. 13800 obr/min
- Długość wału: 7 mm
- Średnica wału: 1,5 mm
- Wymiary: 26,3 x 15,5 mm



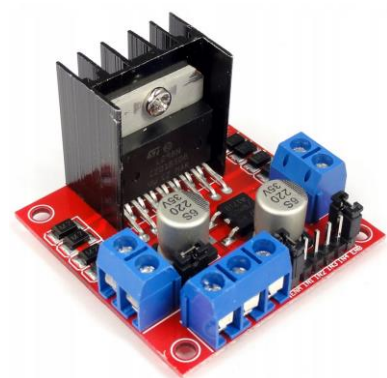
Rysunek 4 Silnik(wentylator)

Sterownik silnika L298N

Do sterowania silnikiem zastosowany został bardzo popularny moduł L298N. Jest to dwukanałowy sterownik silników(mostek H).

Specyfikacja sterownika:

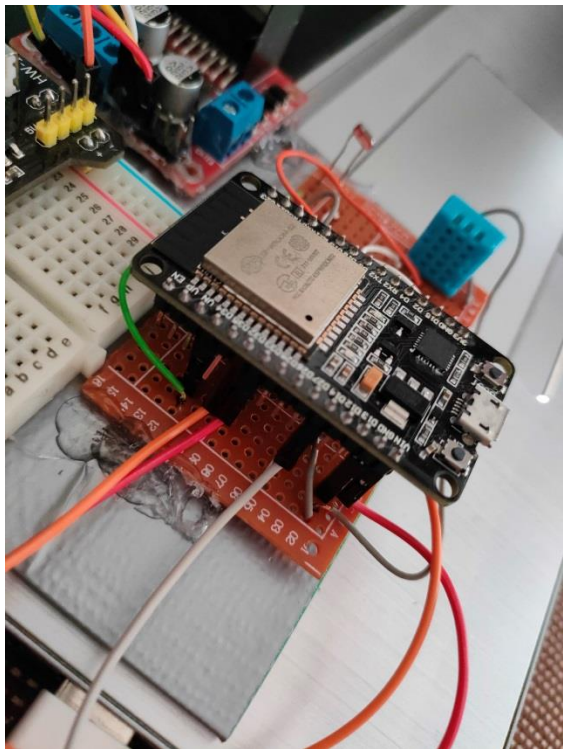
- Maksymalna moc: 25 W
- Zasilanie części logicznej 6...12 V
- Zasilanie silników V_s : 4,8...46 V
- Pobór prądu części logicznej I_{ss} : 36 mA
- Pobór prądu na kanał I_o : 2 A
- Wymiary: 47 × 53 mm, waga ok.29g



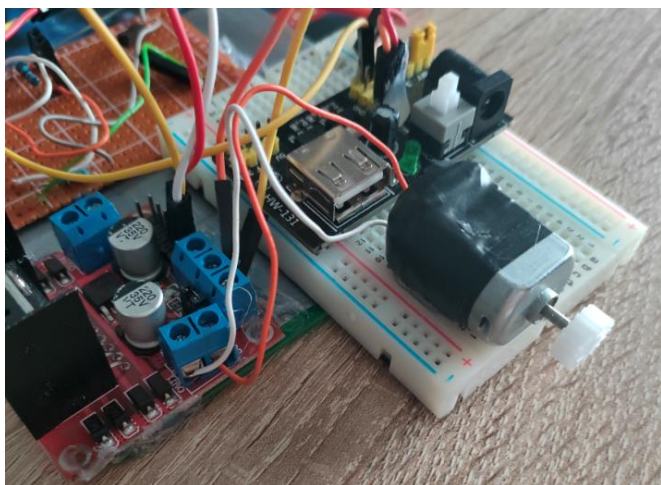
Rysunek 5 Sterownik silnika L298N

Analizator

Realne wykonanie połączeń zostało wykonane na płytce uniwersalnej, ze względu na fakt że połączeń było na tyle niewiele, nieopłacalne okazało się wykonanie dedykowanej płytki PCB w celach projektowych. Płytką umożliwiła szybkie wykonanie trwalszego połączenia w porównaniu jakie zostałyby uzyskane przy zastosowaniu płytki prototypowej. Do ESP32 podpięty został fotorezystor, czujnik dht11 oraz sterownik silnika L298N. Do sterownika natomiast został podpięty silnik. Silnikiem sterujemy za pomocą wyjść cyfrowych mikrokontrolera, natomiast poziomem jego prędkości sterujemy przy pomocy wyjścia PWM. Na Rysunku 6 widzimy ESP32 wraz z czujnikami, na Rysunku 7 znajduje się natomiast silnik wraz z źródłem napięcia.



Rysunek 6 Analizator zdj1



Rysunek 7 Analizator zdj2

Technologia bazy danych

Do przechowywania danych projektowych wybrana została baza danych czasu rzeczywistego Firebase typu NoSQL. Dane przechowywane w niej są w formacie JSON, cała zawartość jest hostowana w chmurze. Synchronizacja następuje ze wszystkimi klientami w czasie rzeczywistym. Firebase jest to flagowe rozwiązanie oferowane przez firmę Google do tworzenia aplikacji mobilnych. Ponadto kolejną zaletą tego rozwiązania jest fakt, że w początkowym użytku prywatnym jest ona darmowa, oczywiście w ograniczonym zakresie. Cena rośnie natomiast proporcjonalnie do wykorzystywanego przez nas pakietu danych.



Rysunek 8 Baza danych - logo

Program - ESP32

Głównym zadaniem ESP32 jest pobieranie odczytów z podłączonych czujników, dzięki temu uzyskujemy informacje o: nasłonecznieniu, wilgotności powietrza oraz temperaturze. Dane te są zapisywane w chmurze z ustalonym przez nas odstępem czasowym. Ważnym elementem programu jest pogrupowanie ich odpowiednio według daty oraz godziny. Poniżej znajdują się najważniejsze fragmenty kodu wraz z koniecznymi opisami.

Użyte biblioteki

Użyte zostało pięć bibliotek. „WiFi.h” służy do nawiązania połączenia z lokalną siecią przez ESP32, „FirebaseESP32.h” jest konieczna w celu umożliwienia wysyłania danych do bazy. „string.h” jest to wbudowana baza danych w ArduinoIDE, umożliwia korzystanie z dynamicznych tablic typu string. „DHT.h” to biblioteka dzięki której o wiele łatwiejsze jest zbieranie odczytów z czujnika DHT11. Ostatnia biblioteka jest przydatna podczas konwertowania wartości czasu.

```
#include <WiFi.h>
#include <FirebaseESP32.h>
#include <string.h>
#include "DHT.h"
#include "time.h"
```

Nawiązanie połączenia Wi-Fi

Nawiązanie połączenia Wi-Fi jest oczywiście bardzo ważne, gdyż dzięki temu dane mogą być zapisywane do bazy bez naszej ingerencji. Po zdefiniowaniu hasła na początku programu w pętli głównej dokonujemy połączenia z siecią lokalną dzięki wyżej załączonemu fragmentowi kodu.

```
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
Serial.print("Connecting to Wi-Fi");

while (WiFi.status() != WL_CONNECTED)
{
    Serial.print(".");
    delay(500);
}
Serial.println();
Serial.print("Connected with IP: ");
Serial.println(WiFi.localIP());
Serial.println();
```


Połączenie z bazą danych

Jak w poprzednim przypadku na początku programu zdefiniowane zostały nazwa hosta oraz klucz autoryzacji do bazy danych. Powyższy fragment dokonuje połączenia z bazą oraz dodatkowo ustawiamy timeout w razie problemów z połączeniem.

```
Firestore.begin(FIREBASE_HOST, FIREBASE_AUTH);
Firestore.reconnectWiFi(true);
Firestore.setReadTimeout(firebaseData, 1000 * 60);
Firestore.setWriteSizeLimit(firebaseData, "tiny");
```

Odczyt czujników

Poniższy fragment znajduje się w pętli setup i służy on do rozpoczęcia połączenia odczytywania informacji z czujnika DHT11.

```
dht.begin();
```

Poniższy fragment odpowiada za przypisanie odpowiednim zmiennym wartości, które zostały odczytane z czujników.

```
h = dht.readHumidity();
t = dht.readTemperature();
s=analogRead(34);
```

Odczyt czasu z serwera NTP

Aby odczytywane dane mogły służyć do jako dane historyczne, konieczne było zapisywanie ich wraz z datą oraz godziną. W tym celu posłużyliśmy się odczytywaniem czasu z serwera NTP. Poniżej widzimy fragment w którym inicjujemy połączenie oraz ustawiamy przesunięcie czasowe.

```
const char* ntpServer = "pool.ntp.org";
const long  gmtoffset_sec = 3600;
const int   daylightOffset_sec = 3600;
```

Kolejny fragment przedstawia konfigurację czasu na podstawie zainicjowanego połączenia.

```
configTime(gmtoffset_sec, daylightOffset_sec, ntpServer);
```

Zdefiniowana została funkcja, która korzystając z biblioteki „time.h”, zwraca strukturę zawierającą informacje o czasie, następnie przy użyciu funkcji „asctime” konwertujemy ją do odpowiedniej formy.

```
String printLocalTime()
{
    String temp;
    struct tm timeinfo;
    if(!getLocalTime(&timeinfo)){
        Serial.println("Failed to obtain time");
    }
    temp=asctime(&timeinfo);

    return temp;
}
```

Przesyłanie danych do bazy

Ważne było odpowiednie dobranie konfiguracji pliku JSON, tak aby była możliwość odczytu danych w bazie w takiej formie umożliwiającej wygenerowanie wykresy. Ważnym aspektem było również utworzenie zapisu w taki sposób, aby każdy odczyt tworzył nowy wiersz, a nie nadpisywał zapis poprzedni.

```
json.set("naslonecznienie", s);  
json.set("temperatura", t);  
json.set("wilgotnosc", h);  
Firebase.pushJSON(firebaseData, "/" + dataa + "/" + godzina, json);
```

Utworzona baza danych

Poniżej załączony został zrzut ekranu, który ukazuje w jakiej formie prezentuje się utworzona baza danych. Dane pogrupowane są według dat, następnie każdej godzinie przypisana jest pieczętka czasowa automatycznie wygenerowana przed Firebase. Każda godzina przechowuje natomiast odczyty czujników. Dodatkowo w bazie znajduje się zakładka Properties, w której przechowywane są informacje na temat temperatury powyżej której ma załączyć się silnik oraz poziom jego prędkości.



Rysunek 9 Widok bazy danych

Kod aplikacji

Aplikacja została napisana w języku Python. Jest to aplikacja desktopowa, która generuje okno GUI. W oknie znajduje się możliwość wybrania zakresu dat dla jakich ma zostać wygenerowany wykres, możemy również wybrać parametr. Aplikacja posiada zabezpieczenia w przypadku podania złego zakresu lub błędnej daty. W dalszej części opisane zostały kluczowe krótkie fragmenty kodu.

Wykorzystane biblioteki

W programie wykorzystane została biblioteki: „Pyrebase” – do zarządzania bazą danych, „tkinter” – GUI, „matplotlib” – do generowania wykresów oraz „datetime” – biblioteka pozwalająca używać zmiennych przechowujących informacje o dacie i czasie.

```
from pyrebase import *
import matplotlib.pyplot as plt
from tkinter import *
import datetime
```

Połączenie z bazą danych

Przy wykorzystaniu jednej z wcześniej opisanych bibliotek nawiązane zostało połączenie z bazą danych, w postaci klienta po wcześniejszym zdefiniowaniu e-maila posiadającego dostęp do bazy.

```
firebaseConfig={'apiKey': "****",
                'authDomain': "****",
                'databaseURL': "****",
                'projectId': "****",
                'storageBucket': "****",
                'messagingSenderId': "****",
                'appId': "****",
                'measurementId': "****"}

firebase=pyrebase.initialize_app(firebaseConfig)
db=firebase.database()

auth=firebase.auth()

email="****"
password="****"

auth.sign_in_with_email_and_password(email, password)
```

Utworzenie GUI

Korzystając z biblioteki „tkinter” utworzone zostało GUI pozwalające zarządzanie generowanymi wykresami za pomocą graficznej aplikacji. Poniżej znajdują się krótkie fragmenty, ukazujące najważniejsze elementy części programu odpowiadającego za GUI:

- Tworzenie etykiet. Są to elementy wyświetlające tekst na aplikacji

```
mylabel_dzien = Label(window, text="Wybierz Dzień:", font=("Arial Bold",18))
mylabel_dzien.place(x=100, y=50)
```

- Tworzenie rozwijalnych opcjonalnych menu

```
lista_dzien=OptionMenu(window, var_dzien, *var_dzien_all)
lista_dzien.config(font=('arial', (16)),bg="#DCDCDC",activebackground="#96B4B4",width=6)
lista_dzien.place(x=130, y=100)
```

- Utworzenie przycisku

```
myButton = Button(window, text="Wygeneruj wykres", font=("Arial Bold",20, 'underline italic'), padx=40, pady=20, command=dane_pobieranie, bg="#1ec81e",activebackground="#336633")
myButton.place(x=700, y=320)
```

- Ukazywanie błędów, przejawia się ono wyświetleniem okna zawierającego komentarz o błędzie

```
window_error3 = Tk()
window_error3.title("Error")
window_error3.geometry('350x50')

mylabel_blad3 = Label(window_error3, text="Komentarz: Podana data jest błędna.", font=("Arial Bold", 14), fg="red")
mylabel_blad3.place(x=10, y=10)
```

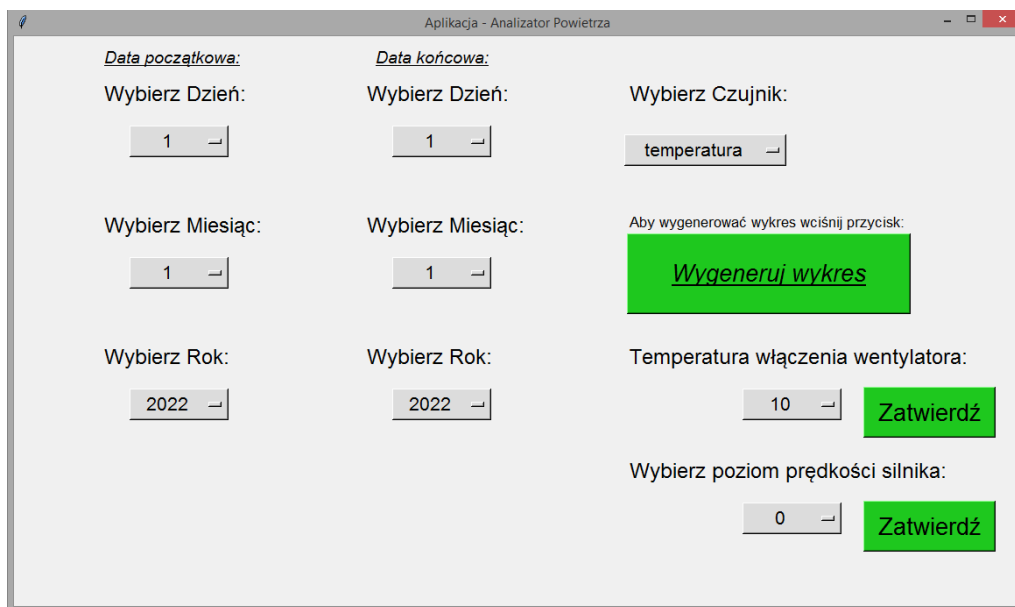
Generowanie wykresów

Wykresy generowane są przy użyciu funkcji plot. Tablica „y” zawiera pobrane z bazy danych wartości odczytu odpowiedniego czujnika. Natomiast „x” jest to tablica zawierająca informacje o dacie oraz czasie odczytu.

```
plt.plot(x, y, marker='o', color="red", linewidth=3)
plt.grid(True)
plt.legend([czujnik])
plt.xlabel("Data oraz godzina odczytu")
```

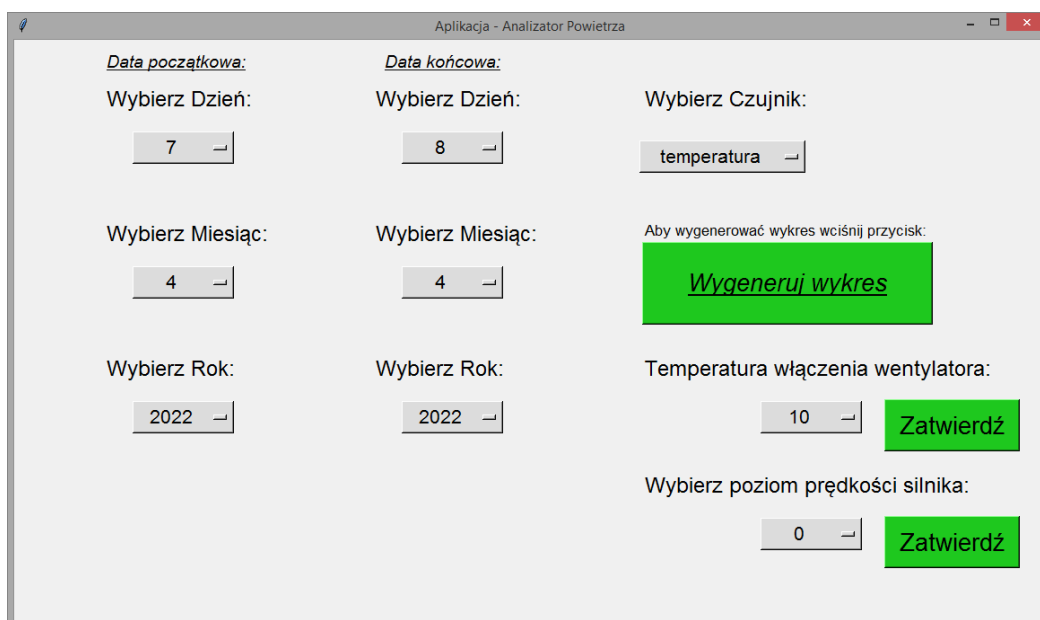
Ukazanie działania systemu

Tak prezentuje się główne okno aplikacji po jej uruchomieniu. Widzimy tu 3 główne sekcje. Po lewej znajduje się sekcja gdzie w menu opcjonalnym wybieramy datę początkową, w środkowej sekcji datę końcową przedziału dla którego ma zostać wygenerowany wykres. Prawa sekcja zawiera menu opcjonalne, gdzie możemy wybrać jaki odczyt chcemy sprawdzić, oraz przycisk do wygenerowania wykresu.

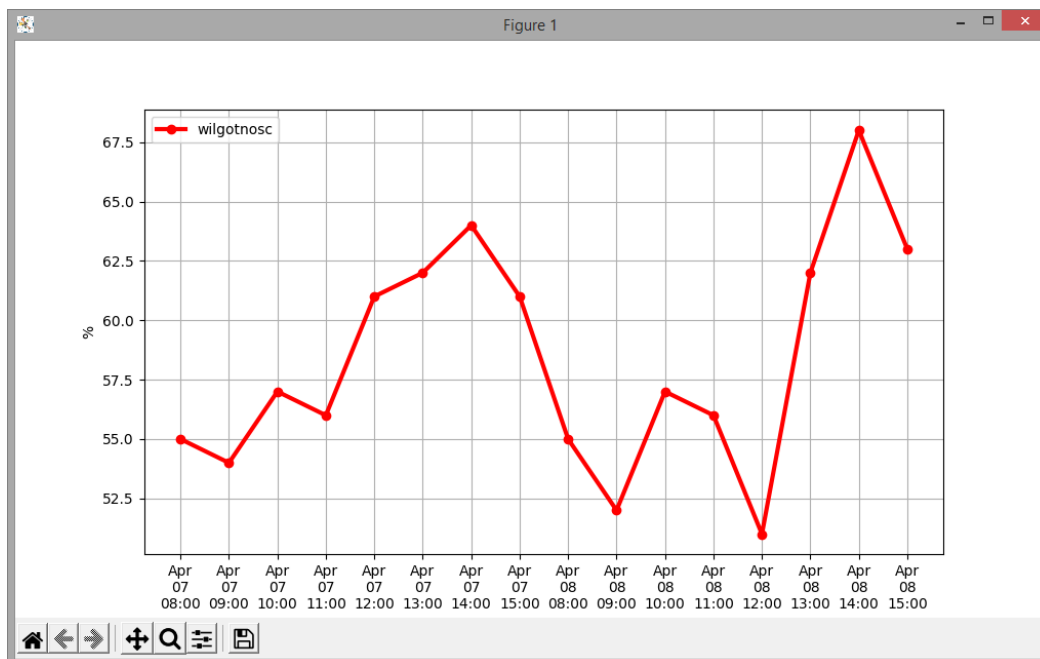


Rysunek 10 Aplikacja - widok główny

Po wybraniu zakresu wartości oraz odczytu aplikacja generuje wykres dla tych dat, dla których znajdują się odczyty w bazie danych. Znajduję się tam również legenda, gdyby użytkownik zapomniał jaki odczyt chciał wygenerować.

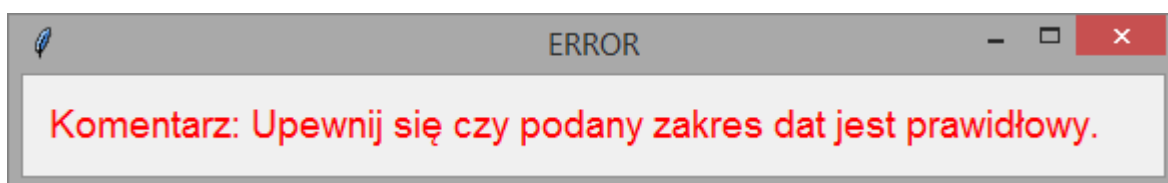


Rysunek 11 Aplikacja - ustawienia użytkownika



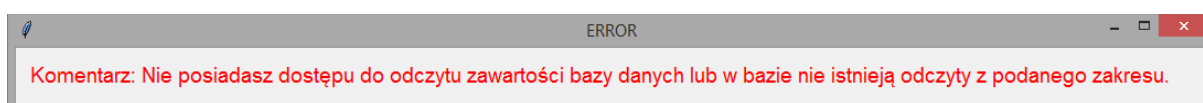
Rysunek 12 Wygenerowany wykres

Aplikacja posiada już wcześniej wspomniany układ zabezpieczeń dla błędów. W przypadku, gdy użytkownik wybierze zły zakres dat, czyli data początkowa jest „większa” niż data końcowa generowany jest następujący komunikat.



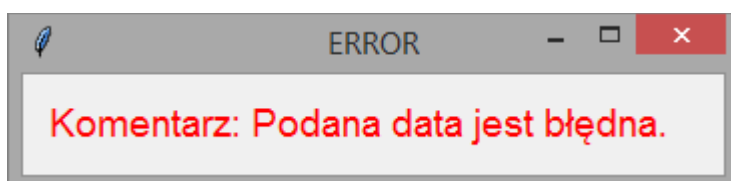
Rysunek 13 Błąd - zły zakres dat

W sytuacji, gdy nie mamy dostępu do bazy danych lub pobrane przez nas dane są puste (czyli baza nie posiada odczytów z danego zakresu) generowane jest następujące okno.



Rysunek 14 Błąd - brak dostępu, brak danych

Ostatnim zabezpieczeniem jest sytuacja, gdy użytkownik poda błędną datę np. 30.02.2022.



Rysunek 15 Błąd - błędna data

Podsumowanie

Podczas testowania cały system sprawuje się naprawdę dobrze. Aplikacja dzięki zabezpieczeniom nie wyłącza się z powodu błędów. Wszystko jest przejrzyste oraz proste w użytkowaniu.

Podczas wykonywania projektu nie natkałem zbyt wiele błędów. Jedynym problemem okazał się transport projektu w przypadku wykonania go na płytce stykowej. Podczas przenoszenia analizatora doszło prawdopodobnie do styku pomiędzy złączami i finalnie doszło do spalenia mikrokontrolera. Prowadzi to do refleksji że nawet małe projektu warto umieszczać na płytkach drukowanych oraz dokumentować pracę na bieżąco.