Jeżeli chcesz zacytować tą pracę to wejdź na zenodo.org:
Projekt systemu pomiarowo-kontrolnego na bazie Arduino

**Projekt Systemu pomiarowo-kontrolnego na bazie Arduino**

**Kacper Ostrowski**

# Wstęp



Przykład kontenera z osprzętem satelitarnym

W dobie dynamicznego rozwoju systemów automatyki i zdalnego nadzoru coraz większe znaczenie zyskują rozwiązania umożliwiające stały monitoring oraz kontrolę parametrów środowiskowych w różnego rodzaju obiektach technicznych. Jednym z takich obiektów są kontenery ze sprzętem elektronicznym, często instalowane w bezpośrednim sąsiedztwie anten komunikacyjnych, w których kluczowe znaczenie ma utrzymanie odpowiednich warunków temperaturowo-wilgotnościowych oraz nadzór nad stanem infrastruktury.



Płytka Arduino Mega 2560



Płytka Arduino Ethernet Shield

Celem niniejszej pracy jest przedstawienie projektu oraz realizacji systemu pomiarowo-kontrolnego opartego na platformie Arduino MEGA z wykorzystaniem modułu Ethernet Shield. System ten został

zaprojektowany do monitorowania i sterowania środowiskiem wewnątrz kontenera technicznego, w którym znajduje się sprzęt wspierający pracę anteny nadawczo-odbiorczej.

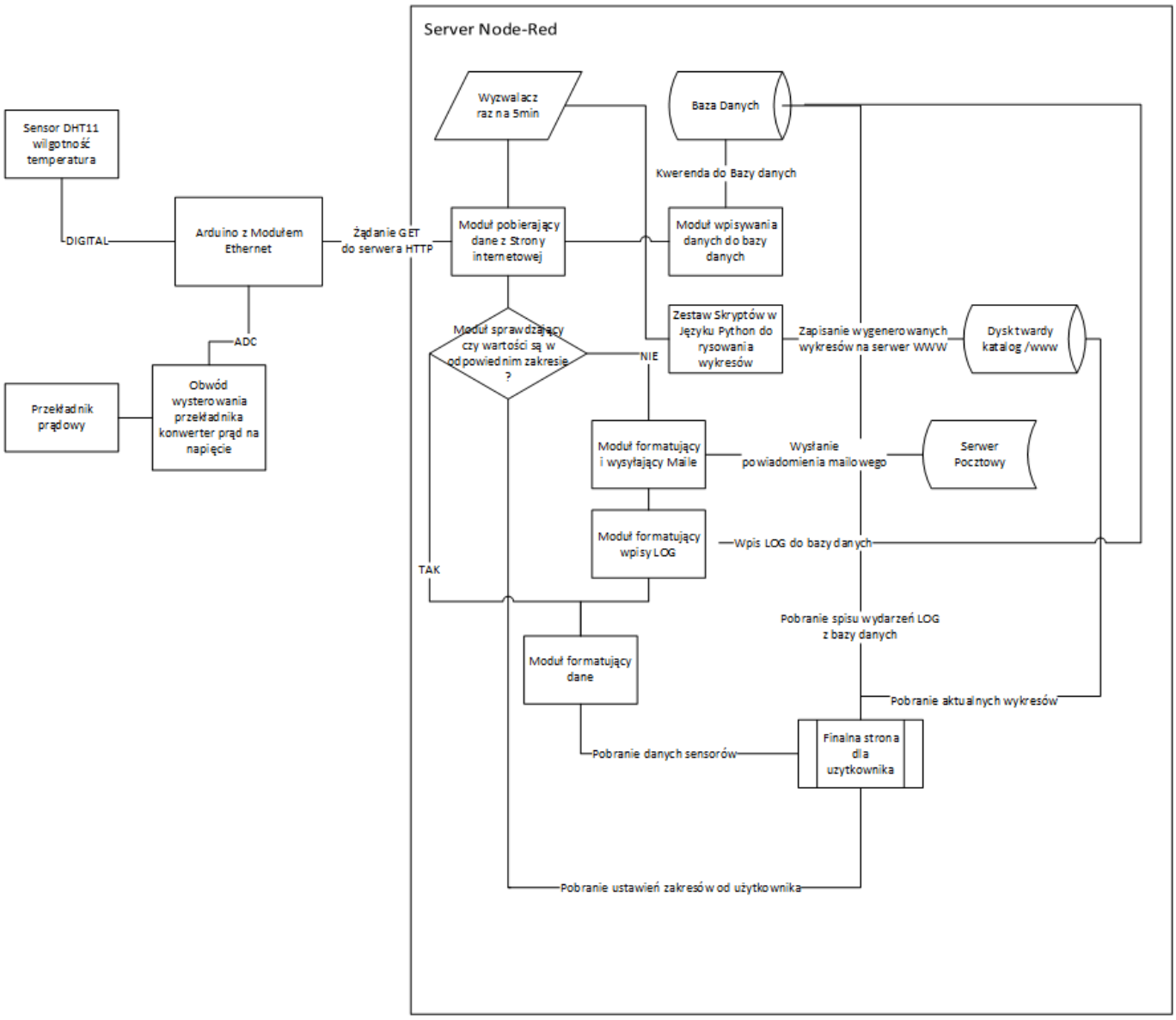
W skład funkcjonalności systemu wchodzi:

- pomiar temperatury i wilgotności powietrza,
- kontrola poboru prądu przez grzejnik znajdujący się wewnątrz kontenera,
- detekcja otwarcia i zamknięcia drzwi,
- monitorowanie obecności cieczy poprzez czujnik zalania.

Zebrane dane są przesyłane za pomocą interfejsu sieciowego, co umożliwia ich zdalny odczyt oraz integrację z zewnętrznymi systemami nadzorującymi. Rozwiązanie to ma na celu zwiększenie niezawodności pracy urządzeń znajdujących się w kontenerze poprzez wczesne wykrywanie nieprawidłowości oraz zapewnienie odpowiednich warunków środowiskowych.

# Opis ogólny działania systemu pomiarowo-kontrolnego

Na poniższym schemacie blokowym przedstawiono uproszczoną strukturę działania systemu pomiarowo-kontrolnego zbudowanego w oparciu o mikrokontroler Arduino MEGA oraz środowisko Node-RED pracujące na serwerze. Celem systemu jest pozyskiwanie danych środowiskowych, analiza ich poprawności, zapisywanie do bazy danych oraz prezentacja wyników użytkownikowi końcowemu.




Schemat blokowy systemu pomiarowo-kontrolnego

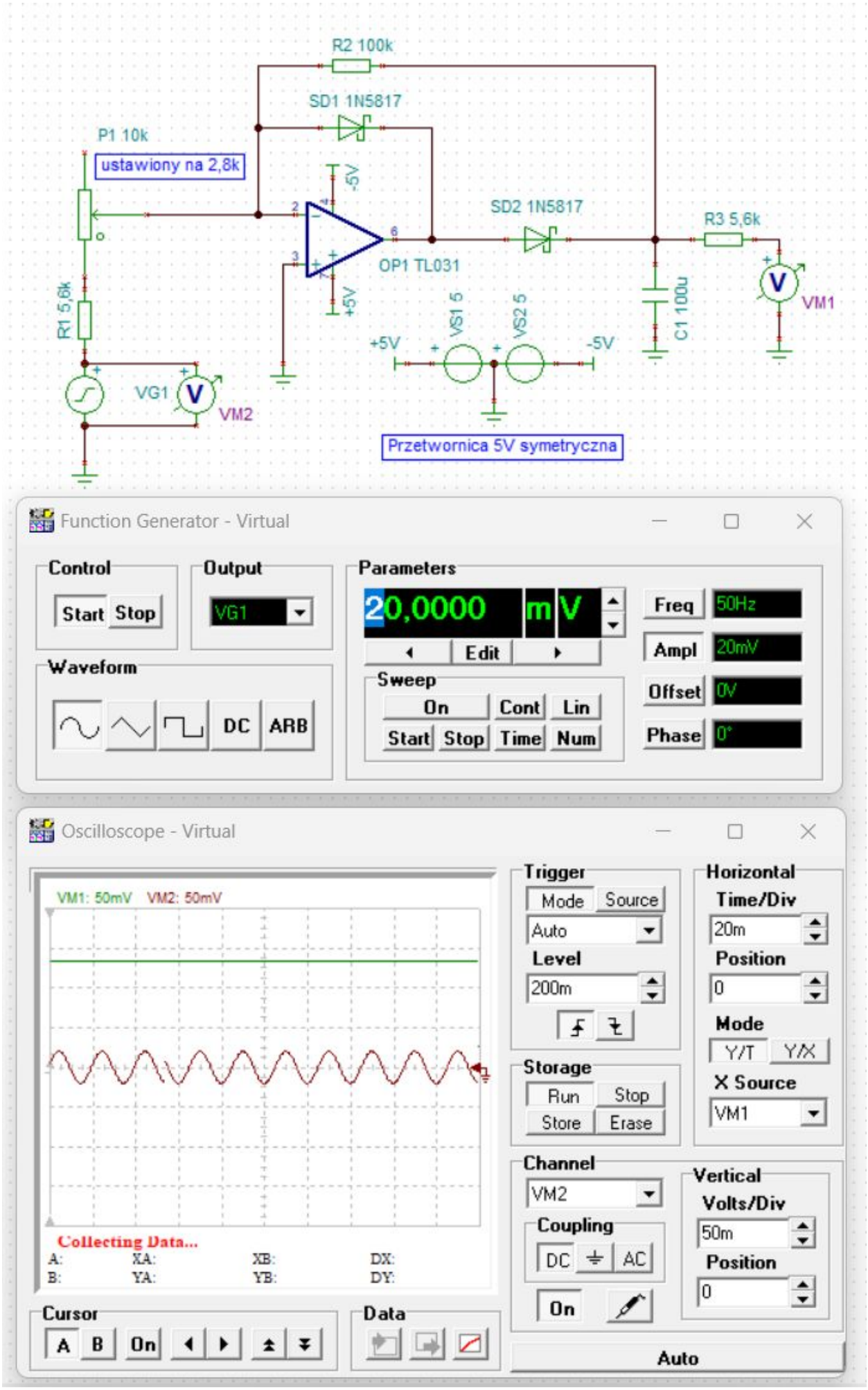
System składa się z następujących głównych części:

- **Moduł Arduino z Ethernet Shield** – odpowiada za pobieranie danych z czujników oraz ich przesyłanie do serwera za pomocą żądań HTTP.
    - *Sensor DHT11* – mierzy temperaturę i wilgotność powietrza.

    **Opis:** Zastosowano sensor w tej wersji ponieważ wahania temperatury wewnątrz kontenera nie spadają poniżej zera

    - *Przekładnik prądowy i tor pomiarowy* – umożliwia pomiar poboru prądu przez grzejnik, sygnał przekształcany jest na napięcie i odczytywany przez przetwornik ADC.

    **Opis:** przekładnik prądowy zastosowany tutaj to prosty transformator, który wymaga poniższego obwodu do poprawnego działania i konwersji napięć na poprawne takie które można odczytać za pomocą przetwornika analogowego Arduino.


R2 100k
SD1 1N5817
P1 10k
ustawiony na 2,8k
-5V
SD2 1N5817
R3 5,6k
OP1 TL031
+5V
VM1
R1 5,6k
C1 100u
VS1 5
VS2 5
+5V
-5V
VG1
VM2
Przetwornica 5V symetryczna
Function Generator - Virtual
Control
Start Stop
Output
VG1
Parameters
20,0000 mV
Edit
Freq 50Hz
Ampl 20mV
Offset 0V
Phase 0°
Waveform
DC ARB
Sweep
On Cont Lin
Start Stop Time Num
Oscilloscope - Virtual
VM1: 50mV VM2: 50mV
Collecting Data...
A: XA: XB: DX:
B: YA: YB: DY:
Trigger
Mode Source
Auto
Level
200m
Storage
Run Stop
Store Erase
Horizontal
Time/Div
20m
Position
0
Mode
Y/T Y/X
X Source
VM1
Channel
VM2
Coupling
DC AC
On
Vertical
Volts/Div
50m
Position
0
Cursor
A B On
Data
Auto

Schemat obwodu do obsługi przekładnika prądowego razem z pomiarami wykonanymi w symulatorze TINA-TI

- **Serwer z oprogramowaniem Node-RED** – realizuje główną logikę systemu:
  - cykliczne pobieranie danych z Arduino,
  - analiza zakresu poprawności danych,
  - zapis danych do bazy danych,
  - generowanie wykresów i raportów,
  - wysyłka powiadomień mailowych.

  **Opis poszczególnych modułów:**

  - *Moduł sprawdzający wartości danych* – decyduje, czy parametry środowiskowe mieszczą się w ustalonym zakresie.

  **Opis:** Składa się z kilku części jedna część pobiera informację z płytki Arduino następnie dane są weryfikowane czy nie przekraczają odpowiednich zakresów.

  - *Moduł formatujący i wysyłający maile* – wysyła ostrzeżenie w przypadku przekroczenia progów alarmowych.

  **Opis:** Pobiera informację jaki parametr został przekroczony na którym kontenerze i wysyła maila

  - *Moduł zapisu do bazy danych oraz rysowania wykresów* – dane są zapisywane i przedstawiane graficznie za pomocą zestawu skryptów w języku Python.

  **Opis:** Formatuje kwerendę INSERT do bazy danych a następnie wpisuje te dane. Druga część to zestaw skryptów w języku Python które łączą się do bazy danych a następnie pobierają i rysują dane które zostają zapisane w katalogu WWW serwera apache. Panel Dashboard w node-red potem się do nich odnosi.

  - *Finalna strona użytkownika* – strona WWW prezentująca aktualne wartości czujników, wykresy oraz umożliwiająca zmianę zakresów monitorowanych parametrów.

  **Opis:** Jest to dashboard zbudowany za pomocą narzędzia node-red prezentuje wykresy aktualne parametry pozwala przełączyć zasilanie do grzejnika itp.

- **Moduły komunikacyjne i użytkowe:**
  - *Serwer pocztowy* – obsługuje wysyłkę wiadomości email.

**Opis:** Jest to zewnętrzny serwer pocztowy na który program wysyła maile. Jest to element zewnętrzny nie będziemy omawiać jego implementacji.

- *Baza danych i katalog serwera WWW* – dane są przechowywane i udostępniane użytkownikowi w postaci wykresów i historii pomiarów.

**Opis:** Są to zewnętrzne komponenty które pozwalają na zbieranie danych oraz na udostępnianie wygenerowanych wykresów po sieci.

System został zaprojektowany z myślą o niezawodnym działaniu w trudnych warunkach środowiskowych, umożliwiając szybką reakcję w razie przekroczenia parametrów krytycznych.

# Omówienie zasady działania systemu Node-RED

Node-RED to środowisko programistyczne typu open-source, które umożliwia graficzne projektowanie przepływów danych (tzw. *flow*) przy pomocy połączeń pomiędzy tzw. węzłami (*nodes*). Środowisko to oparte jest na języku JavaScript (Node.js), a konfiguracja logiki systemu odbywa się z wykorzystaniem intuicyjnego interfejsu przeglądarkowego.

## Podstawowe pojęcia

Podstawową jednostką danych w systemie Node-RED jest obiekt `msg`, który zawiera dane przekazywane między węzłami. Każdy *node* przyjmuje dane wejściowe, wykonuje określoną operację, a następnie przesyła wynik do kolejnych węzłów w postaci zaktualizowanego obiektu `msg`. Obiekt ten ma strukturę podobną do JSON i może zawierać następujące właściwości:

- `msg.payload` – główna część wiadomości zawierająca dane przesyłane przez system.
- `msg.topic` – etykieta opisująca temat wiadomości (używana np. do filtrowania).
- `msg.timestamp` – znacznik czasu nadania wiadomości.
- inne niestandardowe właściwości – mogą być dodane przez użytkownika lub przez węzły (np. `msg.sensorType`, `msg.alert`, itp.).

## Przepływ danych między węzłami

Węzły (*nodes*) są specjalizowanymi blokami funkcyjnymi, które realizują konkretne operacje. Przykładowe typy węzłów:

- `inject` – inicjuje przepływ danych (np. co 5 minut).
- `http request` – pobiera dane z zewnętrznego źródła (np. Arduino).
- `function` – przetwarza dane przy pomocy kodu JavaScript.
- `switch` – sprawdza warunki logiczne i kieruje dane na odpowiednie ścieżki.
- `debug` – służy do testowania i podglądu zawartości wiadomości.
- `email` – wysyła wiadomości e-mail.
- `database` – zapisuje dane do bazy (np. MySQL, SQLite).

W momencie przesłania danych z jednego węzła do drugiego, obiekt `msg` zostaje przekazany dalej. Każdy węzeł może odczytać, zmodyfikować lub rozgałęzić wiadomość, co pozwala na budowanie złożonych procesów przetwarzania informacji.

## Zastosowanie w omawianym systemie

W omawianym systemie pomiarowo-kontrolnym Node-RED pełni rolę centralnej logiki zarządzania i przetwarzania danych. Przykładowy przepływ danych wygląda następująco:

1. **Wyzwalacz czasowy** (`inject`) uruchamia co 5 minut żądanie HTTP do Arduino.
2. **Węzeł HTTP** pobiera dane pomiarowe (np. temperatura, wilgotność, prąd).
3. **Węzeł funkcyjny** analizuje dane i zapisuje je w obiekcie `msg.payload`.
4. **Węzeł warunkowy** (`switch`) sprawdza, czy dane znajdują się w dopuszczalnym zakresie.
5. W zależności od wyniku, dane kierowane są do:
    - **Węzła bazy danych**, gdzie następuje ich zapis,
    - **Węzła logującego**, który tworzy wpis w historii systemu,
    - **Węzła wysyłającego e-mail**, w przypadku wystąpienia alarmu.
6. Dane są również przekazywane do modułu odpowiedzialnego za generowanie wykresów i prezentowane użytkownikowi w postaci graficznej na stronie WWW.

Dzięki modularnej budowie system Node-RED pozwala w prosty sposób modyfikować lub rozbudowywać logikę działania bez konieczności pisania dużej ilości kodu. Każdy z węzłów można edytować wizualnie, a debugowanie odbywa się w czasie rzeczywistym.

# Omówienie i analiza przepływów na platformie Node-RED

# System Pomiarowo-Kontrolny TTCOMM Sp.z.o.o.

**Opracowano przez: Kacper Ostrowski**

**Wersja z dnia: 11.06.2024**

## Analog Sensor Readings

| Sensor | Value |
|---|---|
| Temperature | 20 C |
| Humidity | 44 % |
| Electric Current | 0.00 A |

## Digital Input Readings

| Sensor | Value |
|---|---|
| Door Open/Closed | 1 |
| Flood sensor | 1 |
| Waveguide Position 1 | 1 |
| Waveguide Position 2 | 1 |
| Waveguide Power Supply | 1 |

## Control Buttons

Override Control Buttons:

Waveguide position 1: ON Please check afterwards the status of position in the above table

Waveguide position 2: ON Please check afterwards the status of position in the above table

Room Heater: ON OFF

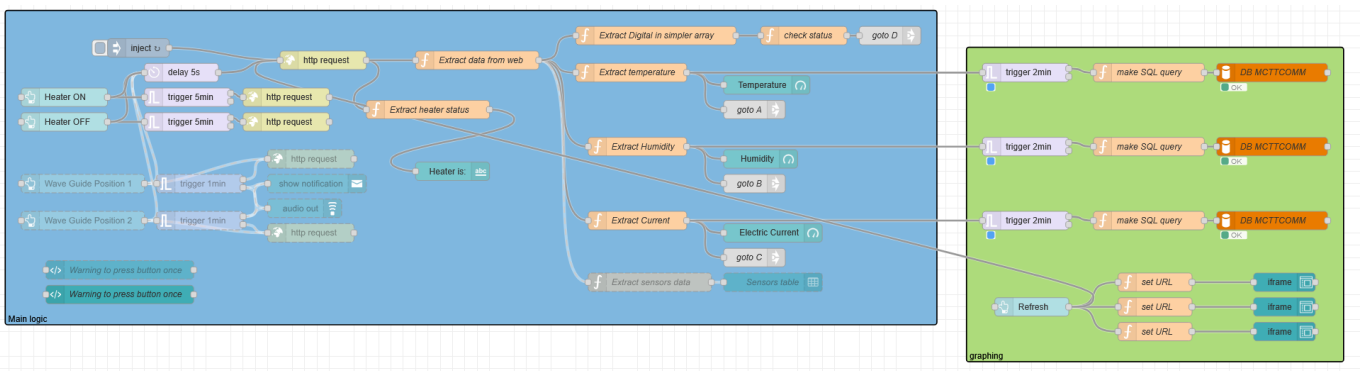
Status: ON

Przykład strony zwracanej przez Arduino

Środowisko Node-RED zostało wykorzystane jako centralny komponent systemu odpowiedzialny za komunikację z urządzeniem pomiarowym (Arduino), przetwarzanie danych oraz ich dalszą dystrybucję do pozostałych elementów infrastruktury informatycznej (baza danych, serwer WWW, system powiadamiania). Ze względu na swoją modularną budowę oraz intuicyjny interfejs graficzny, Node-RED doskonale sprawdza się w systemach automatyki, monitoringu oraz zbierania danych w czasie rzeczywistym.

Stworzony program został podzielony na zestaw logicznych bloków, z których każdy odpowiada za realizację konkretnego zadania w ramach systemu. Dzięki takiej strukturze możliwe było zapewnienie wysokiej przejrzystości działania oraz łatwości rozbudowy o dodatkowe funkcjonalności.

# Bloki: Main Logic i Graphing




Widok bloków z aplikacji Node-RED

## Listingi poszczególnych węzłów funkcyjnych

```
// Extract the payload
let payload = msg.payload;
msg.container = "S5";
// Initialize the result object
let result = {
    analog_sensors: {},
    digital_inputs: {}
};

// Regex patterns to match the sensor readings
let analogSensorPattern = /<tr><td>([^<]+)<\/td><td>([^<]+)<\/td><\/tr>/g;
let digitalSensorPattern = /<tr><td>([^<]+)<\/td><td>([^<]+)<\/td><\/tr>/g;

// Extract analog sensor readings
let match;
while (match = analogSensorPattern.exec(payload)) {
    let sensorName = match[1].trim();
    let sensorValue = match[2].trim();
    result.analog_sensors[sensorName] = sensorValue;
}

// Extract digital input readings
while (match = digitalSensorPattern.exec(payload)) {
    let sensorName = match[1].trim();
    let sensorValue = match[2].trim();
    result.digital_inputs[sensorName] = sensorValue;
}

// Return the result as a JSON object
msg.payload = result;
return msg;
```

```
// Extract the payload (HTML content)
```

```
let html = msg.payload;

// Initialize a variable to store the heater status
let heaterStatus = "Unknown";

// Use a regular expression to find the heater status
let statusMatch = html.match(/<p class='status-(on|off)'>Status:
(ON|OFF)/i);

if (statusMatch) {
    // Extract the status (ON or OFF) from the match
    heaterStatus = statusMatch[2];
}

// Set the heater status as the new payload
msg.payload = heaterStatus;

// Return the modified message
return msg;
```

```
// Initialize an empty array to store digital sensor data
let digitalSensorsArray = [];

// Extract the digital_inputs object from the payload
let digitalInputs = msg.payload.digital_inputs;

// Loop through each key-value pair in the digital_inputs object
for (let [sensorName, sensorValue] of Object.entries(digitalInputs)) {
    // Push each sensor's name and value as an object to the array
    digitalSensorsArray.push({
        name: sensorName,
        value: sensorValue
    });
}

// Set the output message payload to the array of digital sensors
msg.payload = digitalSensorsArray;

// Return the modified message
return msg;
```

```
// Extract the digital sensors array from the incoming message payload
let digitalSensorsArray = msg.payload;

// Initialize an empty array to store sensors with value "0"
let sensorsWithZeroValue = [];

// Loop through each sensor in the array
for (let sensor of digitalSensorsArray) {
    // Check if the sensor's name starts with "Waveguide Position"
    if (!sensor.name.startsWith("Waveguide Position")) {
```

```
        // Check if the sensor's value is "0"
        if (sensor.value === "0") {
            // Add the sensor to the array of sensors with value "0"
            sensorsWithZeroValue.push(sensor);
        }
    }
}

// Check if there are any sensors with value "0"
if (sensorsWithZeroValue.length > 0) {
    // Set the message payload to the array of sensors with value "0"
    msg.payload = sensorsWithZeroValue;
    // Return the message to the next node
    return msg;
} else {
    // If no sensors have value "0", return null (which means the message is
discarded)
    return null;
}
```

```
// Access the temperature value from the JSON object
msg.payload = parseInt(msg.payload.analog_sensors["Temperature"]);
msg.name = "Temperature"
msg.container = "S5";
// Return the modified message
return msg;
```

```
// Access the temperature value from the JSON object
msg.payload = parseInt(msg.payload.analog_sensors["Humidity"]);
msg.name = "Humidity";
msg.container = "S5";
// Return the modified message
return msg;
```

```
// Access the temperature value from the JSON object
msg.payload = msg.payload.analog_sensors["Electric Current"];
msg.payload = parseFloat(msg.payload.match(/[\d.]+/)[0]);
msg.name = "Electric Current";
msg.container = "S5";
// Return the modified message
return msg;
```

```
msg.topic = `INSERT INTO \`S5_Temperature\` (\`DATE\`, \`VALUE\`) VALUES
(now(), '${msg.payload}');`;

return msg;
```

```
//pierwszy
msg.url = "http://<ip serwer www>/graphs/S5_Temperature_1d.png";
return msg;
//drugi
```

```
msg.url = "http://<ip serwer www>/graphs/S5_Current_1d.png";
return msg;
//trzeci
msg.url = "http://<ip serwer www>/graphs/S5_Humidity_1d.png";
return msg;
```

## Wyjaśnienie zasady działania węzłów

1. Węzeł: Extract Data from Web Węzeł odpowiedzialny za przetwarzanie danych HTML pobranych ze strony WWW. W pierwszej kolejności przypisuje zawartość `msg.payload` do zmiennej `payload` oraz ustawia nazwę kontenera jako S5 (`msg.container`). Następnie tworzy pusty obiekt `result`, zawierający dwie sekcje: `analog_sensors` oraz `digital_inputs`. Za pomocą wyrażeń regularnych
`/ <tr><td>([^<]+)<\/td><td>([^<]+)<\/td><\/tr>/g`
przeszukuje dane HTML w celu wyodrębnienia nazw i wartości sensorów analogowych i cyfrowych. W każdej pętli `while`, odnalezione dane przypisywane są do odpowiednich pól w obiekcie `result`. Na końcu wynikowy obiekt JSON jest przypisywany do `msg.payload` i przekazywany dalej.

2. Węzeł: Extract heater status Węzeł służy do odczytu stanu grzejnika na podstawie treści HTML przekazanej w `msg.payload`. Za pomocą wyrażenia regularnego `/<p class='status-(on|off)'>
Status: (ON|OFF)/i` wyszukiwany jest fragment HTML zawierający informację o stanie urządzenia. Jeśli dopasowanie zakończy się powodzeniem (`if (statusMatch)`), z drugiego elementu tablicy wynikowej `statusMatch[2]` pobierany jest status grzejnika (np. `ON` lub `OFF`). Wartość ta przypisywana jest do `msg.payload` i przekazywana dalej w przepływie.

3. Węzeł: Extract Digital in simpler array Węzeł przekształca dane wejściowe z obiektu `msg.payload.digital_inputs` na prostszą formę — tablicę obiektów. Na początku tworzona jest pusta tablica `digitalSensorsArray`, do której kolejno dodawane są obiekty zawierające nazwę i wartość każdego czujnika cyfrowego (`name`, `value`). Dane są pozyskiwane za pomocą pętli `for (let [sensorName, sensorValue] of Object.entries(...))`, a wynik przypisywany do `msg.payload` jako nowa, uproszczona struktura danych.

4. Węzeł: check status Węzeł analizuje dane wejściowe zawarte w `msg.payload`, które stanowią tablicę czujników cyfrowych. Dla każdego czujnika wykonywana jest kontrola — jeśli jego nazwa nie zaczyna się od `"Waveguide Position"` oraz jego wartość wynosi `"0"`, to czujnik zostaje dodany do nowej tablicy `sensorsWithZeroValue`. Jeśli po zakończeniu pętli tablica zawiera jakiekolwiek elementy, zostaje ona ustawiona jako nowe `msg.payload` i przekazana dalej. W przeciwnym przypadku węzeł zwraca `null`, co powoduje przerwanie dalszego przetwarzania wiadomości.

5. Węzeł: Extract Temperature Węzeł pobiera wartość temperatury z pola `analog_sensors["Temperature"]` znajdującego się w `msg.payload`, a następnie konwertuje ją do liczby całkowitej za pomocą `parseInt(...)`, przypisując wynik z powrotem do `msg.payload`. Ustawia również identyfikatory `msg.name` oraz `msg.container`

odpowiednio na `"Temperature"` i `"S5"`.

6. Węzeł: Extract Humidity Analogicznie do poprzedniego węzła, pobiera wartość wilgotności z `analog_sensors["Humidity"]`, konwertuje ją na liczbę całkowitą funkcją `parseInt(...)` i przypisuje do `msg.payload`. Ustawia także nazwę parametru i kontenera w `msg.name` oraz `msg.container`.

7. Węzeł: Extract Current Węzeł odpowiedzialny za ekstrakcję wartości natężenia prądu z pola `analog_sensors["Electric Current"]`. Używa wyrażenia regularnego `msg.payload.match(/[.]+/)` do wyłuskania liczby zmiennoprzecinkowej z tekstu, a następnie przekształca ją do typu `float`. Dodaje również pola opisujące nazwę parametru i kontener.

8. Węzeł: make SQL query Generuje zapytanie SQL wstawiające wartość pomiaru temperatury do tabeli `S5_Temperature` z aktualnym znacznikiem czasu. Tworzy pole `msg.topic` zawierające zapytanie w formacie
`INSERT INTO ... VALUES (now(), '${msg.payload}')`.

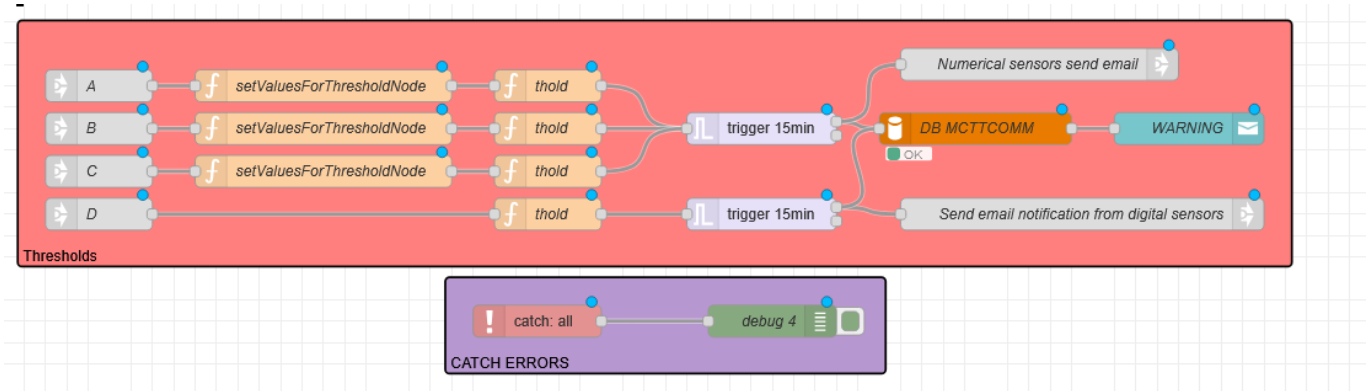
9. Węzły: set URL Każdy z węzłów ustawia odpowiedni adres URL wykresu parametru w polu `msg.url`:

   - Pierwszy węzeł: `S5_Temperature_1d.png`
   - Drugi węzeł: `S5_Current_1d.png`
   - Trzeci węzeł: `S5_Humidity_1d.png`

   Adresy odnoszą się do zasobów graficznych generowanych przez zewnętrzny serwer WWW.

# Bloki: Thresholds i CATCH ERRORS

-




Widok bloków z aplikacji Node-RED

## Listingi poszczególnych węzłów funkcyjnych

```
msg.lowerThreshold = global.get("temperatureLowerThold","file");
msg.upperThreshold = global.get("temperatureUpperThold","file");
```

```
return msg;
```

```
let value = msg.payload;  // The incoming numeric value
let name = msg.name;  // Variable name
let lowerThreshold = msg.lowerThreshold;  // Lower boundary
let upperThreshold = msg.upperThreshold;  // Upper boundary
let container = msg.container; // Name of the container
msg.url = "http://api.ttcomm.net/graphs/S5_Temperature_1d.png";
msg.value = value; //used for extracting it for email

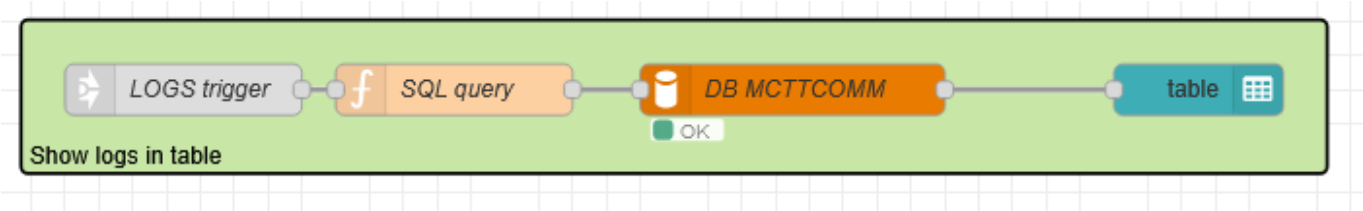
if (value < lowerThreshold || value > upperThreshold) {
    msg.topic = `INSERT INTO \`thold_log\` (\`TIME\`, \`CONTAINER\`,
\`LOG\`)
    VALUES (NOW(), '${container}',
    'Parameter ${name} in container ${container} has a value of ${value},
which is outside the set boundaries (${lowerThreshold} -
${upperThreshold}).');`;
    return msg;  // Send the message to the next node
} else {
    return null;
}
```

### Wyjaśnienie zasady działania węzłów

1. Węzeł: setValuesForThresholdNode Węzeł ten pobiera z pamięci globalnej zdefiniowane progi temperatury: dolny `temperatureLowerThold` oraz górny `temperatureUpperThold`, zapisane w pliku konfiguracyjnym (drugi parametr: `"file"`). Ustawia je w wiadomości jako pola `msg.lowerThreshold` oraz `msg.upperThreshold`, umożliwiając ich dalsze użycie w logice porównawczej.

2. Węzeł: thold Węzeł służy do porównania wartości pomiaru z zadanymi progami. Na podstawie pól `msg.payload` (wartość), `msg.lowerThreshold`, `msg.upperThreshold`, `msg.name` oraz `msg.container` wykonywana jest kontrola, czy wartość wychodzi poza zadane granice. Jeśli tak, to generowane jest zapytanie SQL (`msg.topic`) dodające rekord do tabeli `thold_log`, zawierający czas, nazwę kontenera oraz opis przekroczenia. Dodatkowo przypisywane są pola `msg.url` (link do wykresu) oraz `msg.value` (wykorzystywana np. do wiadomości e-mail). Jeśli wartość mieści się w dopuszczalnych granicach, węzeł zwraca `null`, zatrzymując dalszy przepływ wiadomości.

## Blok: Show logs in table




Widok bloków z aplikacji Node-RED

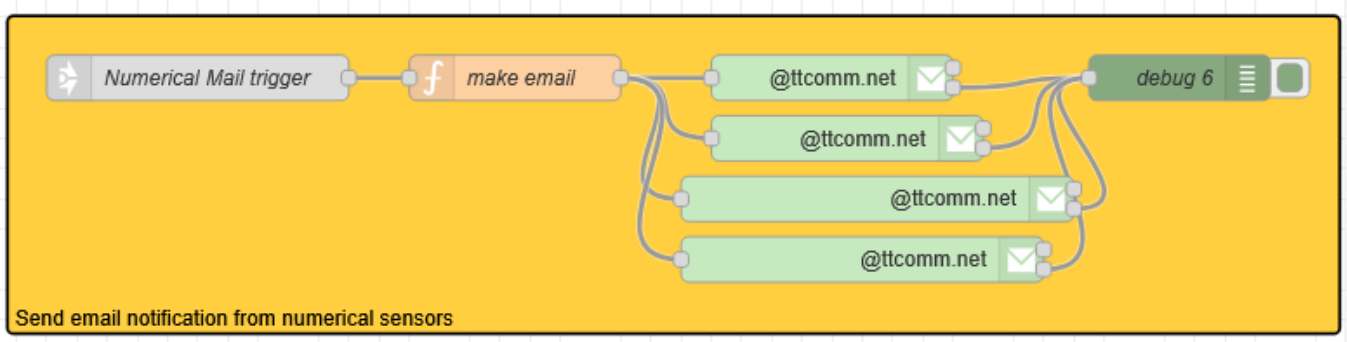
### Listingi poszczególnych węzłów funkcyjnych

```
msg.topic = `SELECT * FROM thold_log;`;
return msg;
```

### Zasada działania

Tutaj zasada działania jest bardzo prosta pobieramy wszystko co jest w tabeli `thold_log` a następnie wyświetlamy to w interfejsie w formie tabeli.

## Blok: Send Numerical notificaiton from numerical sensors




Widok bloków z aplikacji Node-RED

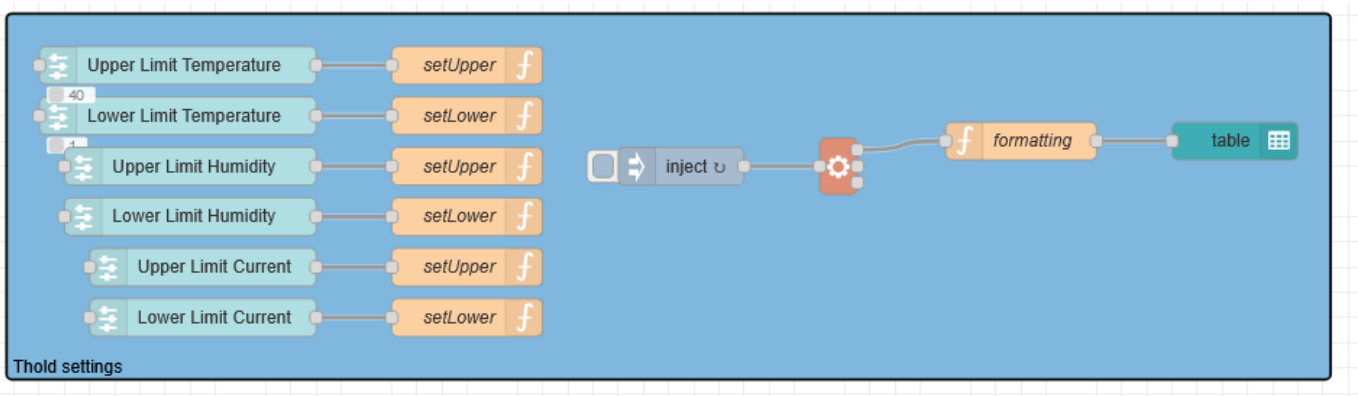
### Listingi poszczególnych węzłów funkcyjnych

```
msg.from = "node-red@ttcomm.net"
msg.topic = `Alert: ${msg.container} ${msg.name} is outside thresholds`;
msg.payload = `Parameter ${msg.name} in container ${msg.container} has a
value of ${msg.value}, which is outside the set boundaries
(${msg.lowerThreshold} - ${msg.upperThreshold})<br><a
href=\"${msg.url}\">Click here to view graph</a>`;
return msg;;
```

### Zasada działania

Tutaj pobieramy z obiektu `msg` potrzebne parametry żeby utworzyć wiadomośc email aby następnie wysłać ją do odpowiednich użytkowników

# Blok: Thold settings




Widok bloków z aplikacji Node-RED

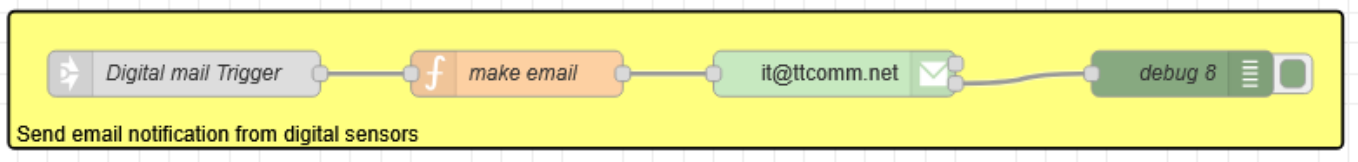
### Listingi poszczególnych węzłów funkcyjnych

```
//bloki lower
global.set("humidityLowerThold", msg.payload, "file");
//bloki upper
global.set("currentUpperThold", msg.payload, "file");
```

### Zasada działania

Tutaj ustawiamy wartości globalne dla poszczególnych thresholdów. Są one nastepnie wykorzystywane w poprzednich omawianych tutaj przepływach.

# Blok: Send email notification from digital sensors




Widok bloków z aplikacji Node-RED

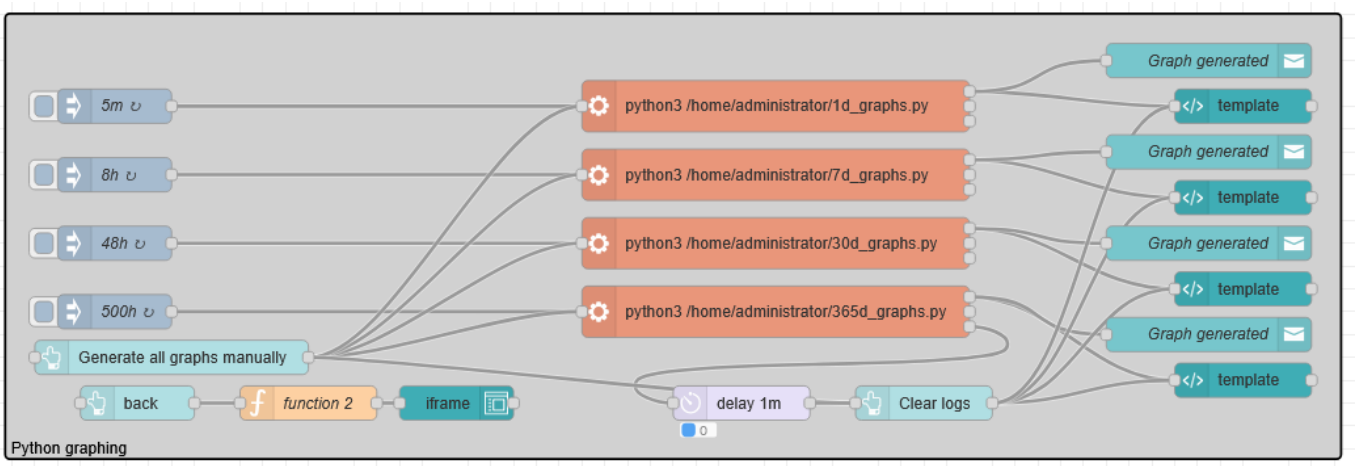
### Listingi poszczególnych węzłów funkcyjnych

```
msg.from = "node-red@ttcomm.net"
msg.topic = `Alert: ${msg.container} digital sensors changed status`;
msg.payload = `Sensor in container ${msg.container} status:
${msg.sensorStatusString}`;
return msg;
```

### Zasada działania

Ten blok jest potrzebny z racji trochę innej struktury maili dla sensorów z wartościami numerycznymi.

## Blok: Python graphing



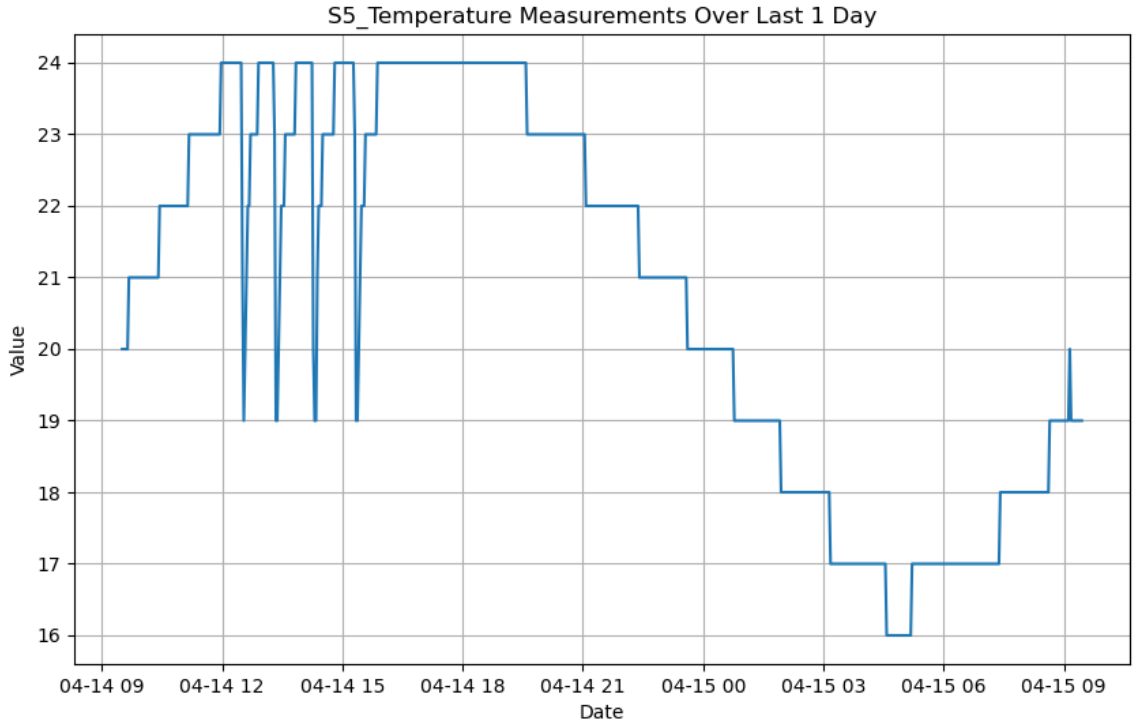

Widok bloków z aplikacji Node-RED

### Zasada działania

Przedstawiony przepływ Node-RED realizuje automatyczną oraz ręczną generację wykresów na podstawie skryptów Pythona. Cztery wyzwalacze czasowe (co 5 minut, 8 godzin, 48 godzin, oraz 500 godzin) inicjują wykonanie odpowiednich skryptów generujących wykresy dla okresów: 1, 7, 30 i 365 dni. Dodatkowo, zastosowany został przycisk umożliwiający ręczne wygenerowanie wszystkich wykresów jednocześnie oraz kolejny przycisk służący do powrotu do domyślnego widoku wykresów. Wyniki uruchamiania skryptów prezentowane są użytkownikowi na pulpicie Node-RED poprzez elementy UI, wyświetlające logi z wykonania skryptów oraz powiadomienia („toast”). Całość interfejsu dopełnia element typu iframe, w którym użytkownik może wygodnie przeglądać wygenerowane wykresy.

# Skrypt w Pythonie do generowania grafów

Skrypt odpowiedzialny jest za automatyczne generowanie wykresów parametrów środowiskowych takich jak temperatura, wilgotność oraz pobór prądu na podstawie danych zgromadzonych w bazie MySQL. Dane te są następnie przetwarzane i wizualizowane w formie wykresów PNG, które mogą być publikowane w sieci lokalnej lub w przeglądarce użytkownika końcowego.




Przykład grafu wygenerowanego przez skrypt

# Opis działania skryptu

Skrypt rozpoczyna działanie od zaimportowania niezbędnych bibliotek: `pymysql` do komunikacji z bazą danych, `matplotlib.pyplot` do generowania wykresów, `pandas` do analizy danych oraz `datetime` do obsługi czasu (`linia 1–5`).

Funkcja `plot_1d_graph(table_name)` przyjmuje jako argument nazwę tabeli z bazy danych, z której zostaną pobrane dane (`linia 7`). Następnie nawiązywane jest połączenie z bazą danych MySQL (`linia 9–13`), a dane z kolumn DATE oraz VALUE są pobierane do ramki danych `DataFrame` przy użyciu zapytania SQL (`linia 16`).

Po pobraniu danych połączenie zostaje zamknięte (`linia 19`), a kolumna DATE jest konwertowana do typu `datetime` (`linia 22`), co umożliwia filtrowanie danych w zadanym zakresie czasu.

Zakres czasowy ustalany jest na ostatnie 24 godziny, co realizuje fragment:

```
start_date = datetime.now() - timedelta(days=1)
    filtered_df = df[df['DATE'] >= start_date]
```

Następnie, za pomocą biblioteki `matplotlib`, generowany jest wykres typu liniowego (`linia 27–31`). Na wykresie umieszczane są podpisy osi oraz tytuł wykresu, który zawiera nazwę tabeli.

Wygenerowany wykres jest zapisywany jako plik PNG w katalogu `/var/www/html/graphs/` pod nazwą odpowiadającą nazwie tabeli z przyrostkiem `_1d` (`linia 34`).

Na końcu, skrypt wywołuje funkcję `plot_1d_graph` trzykrotnie dla różnych tabel:

- S5_Temperature
- S5_Humidity
- S5_Current

Każde z tych wywołań skutkuje wygenerowaniem osobnego wykresu z ostatnich 24 godzin dla danego parametru.

```
import pymysql
import matplotlib.pyplot as plt
import pandas as pd
from datetime import datetime, timedelta

def plot_1d_graph(table_name):
# Establish connection to the MySQL database
conn = pymysql.connect(
host='localhost',        # Your MySQL host
user='administrator',    # Your MySQL username
password='PASS',# Your MySQL password
database='mcTTcomm'      # Your database name
)

# Fetch data from the specified table
query = f"SELECT DATE, VALUE FROM {table_name}"
df = pd.read_sql(query, conn)

# Close the connection
conn.close()

# Convert 'DATE' column to datetime format
df['DATE'] = pd.to_datetime(df['DATE'])

# Define time range for 1 day
start_date = datetime.now() - timedelta(days=1)
filtered_df = df[df['DATE'] >= start_date]

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(filtered_df['DATE'], filtered_df['VALUE'], linestyle='-')
plt.title(f'{table_name} Measurements Over Last 1 Day')
plt.xlabel('Date')
plt.ylabel('Value')
plt.grid(True);

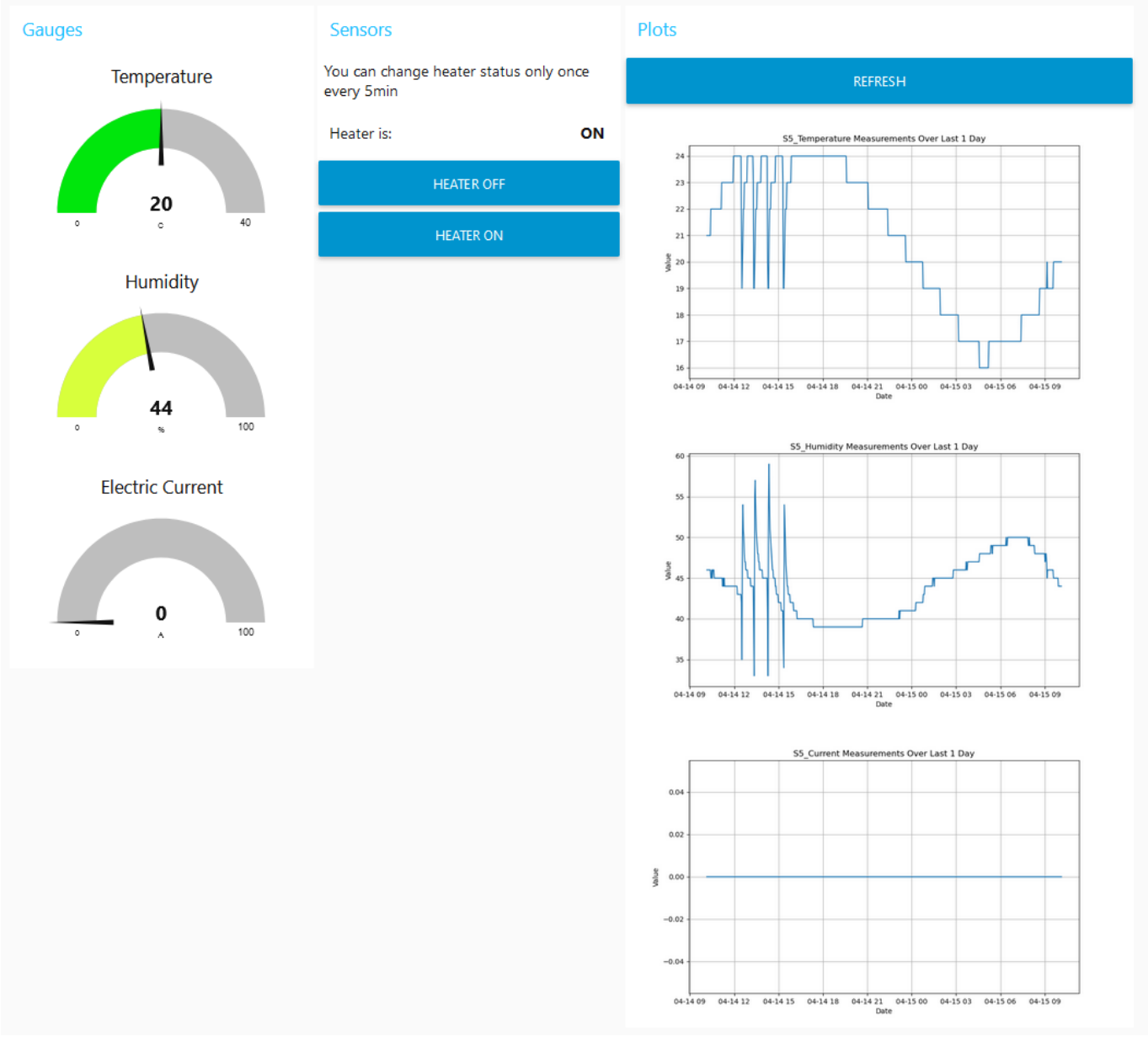
# Save the plot as a PNG file
output_path = f'/var/www/html/graphs/{table_name}_1d.png'
plt.savefig(output_path)
plt.close()

print(f"1-day graph saved to {output_path}")

# Example usage:
```

```
plot_1d_graph('S5_Temperature')
plot_1d_graph('S5_Humidity')
plot_1d_graph('S5_Current')
```

# Interfejs Graficzny



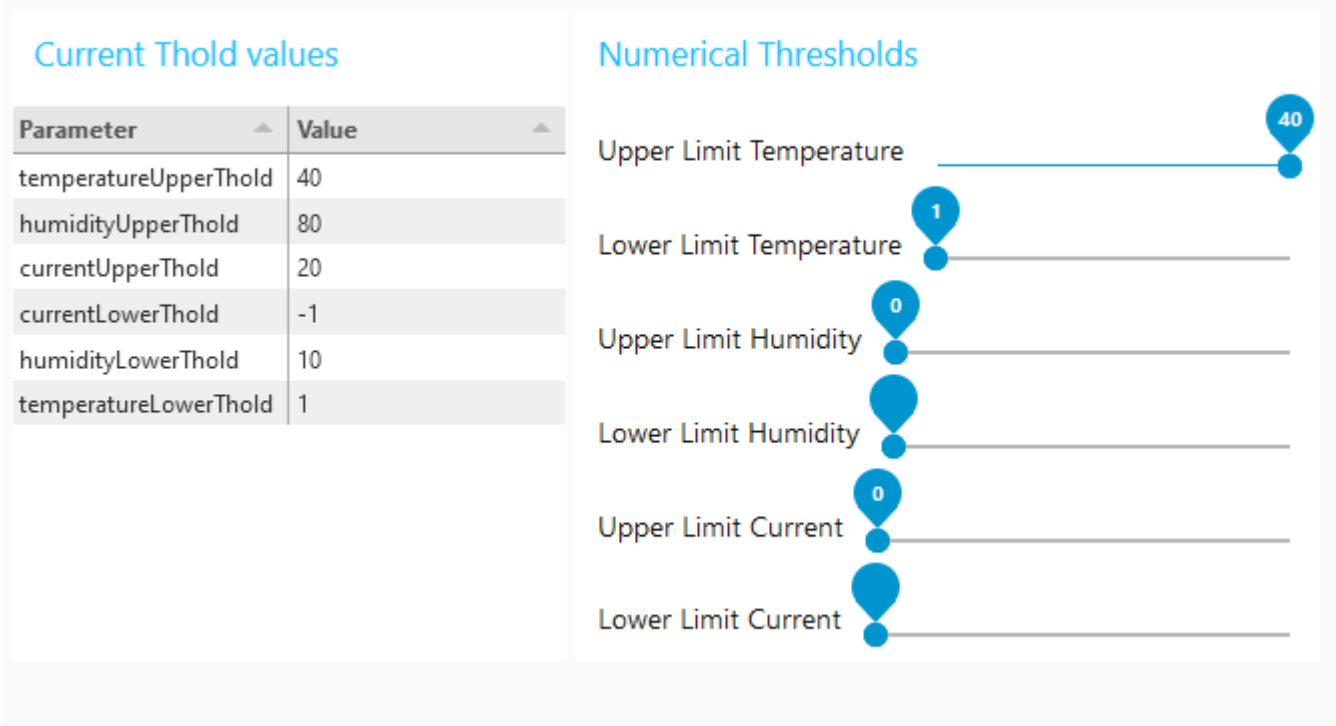

Główna strona interfejsu

Główna strona aplikacji prezentuje kluczowe informacje w czytelnej formie graficznej. Znajdują się tutaj trzy wskaźniki wskazówkowe (gauges), pokazujące odczyty bieżących parametrów środowiskowych: temperatury, wilgotności oraz natężenia prądu elektrycznego. Dodatkowo, użytkownik posiada możliwość manualnej kontroli urządzenia grzewczego, które można załączać lub wyłączać z częstotliwością maksymalnie raz na 5 minut. Po prawej stronie znajduje się przestrzeń do wyświetlania aktualnych wykresów przedstawiających historię pomiarów dla ostatniego dnia.

| Id | Time | Cont... | Notification |
|---|---|---|---|
| 390 | 2024-12-16T13:45:10.000Z | S5 | Parameter Temperature in container S5 has a value of 12, which is outside the set boundaries (10 - 0). |
| 391 | 2024-12-16T13:47:00.000Z | S5 | Parameter Temperature in container S5 has a value of 12, which is outside the set boundaries (10 - 0). |
| 392 | 2024-12-20T10:19:00.000Z | S5 | Parameter Temperature in container S5 has a value of 22, which is outside the set boundaries (10 - 21). |
| 393 | 2024-12-20T21:34:00.000Z | S5 | Parameter Temperature in container S5 has a value of 2, which is outside the set boundaries (10 - 21). |
| 394 | 2025-01-05T07:49:54.000Z | S5 | Parameter Temperature in container S5 has a value of 9, which is outside the set boundaries (10 - 21). |
| 395 | 2025-02-13T07:44:36.000Z | S5 | Parameter Temperature in container S5 has a value of 9, which is outside the set boundaries (10 - 21). |
| 396 | 2025-02-14T14:14:53.000Z | S5 | Parameter Temperature in container S5 has a value of 2, which is outside the set boundaries (10 - 21). |
| 397 | 2025-02-14T14:15:23.000Z | S5 | Parameter Temperature in container S5 has a value of 3, which is outside the set boundaries (10 - 21). |
| 398 | 2025-02-14T14:30:23.000Z | S5 | Parameter Temperature in container S5 has a value of 3, which is outside the set boundaries (10 - 21). |
| 399 | 2025-02-14T14:42:23.000Z | S5 | Parameter Temperature in container S5 has a value of 2, which is outside the set boundaries (10 - 21). |
| 400 | 2025-02-14T14:49:23.000Z | S5 | Parameter Temperature in container S5 has a value of 3, which is outside the set boundaries (10 - 21). |
| 401 | 2025-02-14T14:51:23.000Z | S5 | Parameter Temperature in container S5 has a value of 4, which is outside the set boundaries (10 - 21). |
| 402 | 2025-02-14T14:52:23.000Z | S5 | Parameter Temperature in container S5 has a value of 5, which is outside the set boundaries (10 - 21). |
| 403 | 2025-02-14T14:54:23.000Z | S5 | Parameter Temperature in container S5 has a value of 6, which is outside the set boundaries (10 - 21). |
| 404 | 2025-02-14T14:56:23.000Z | S5 | Parameter Temperature in container S5 has a value of 7, which is outside the set boundaries (10 - 21). |
| 405 | 2025-02-14T14:57:23.000Z | S5 | Parameter Temperature in container S5 has a value of 8, which is outside the set boundaries (10 - 21). |
| 406 | 2025-02-14T14:59:23.000Z | S5 | Parameter Temperature in container S5 has a value of 9, which is outside the set boundaries (10 - 21). |
| 407 | 2025-02-16T01:03:41.000Z | S5 | Parameter Temperature in container S5 has a value of 9, which is outside the set boundaries (10 - 21). |
| 408 | 2025-02-16T01:58:41.000Z | S5 | Parameter Temperature in container S5 has a value of 9, which is outside the set boundaries (10 - 21). |

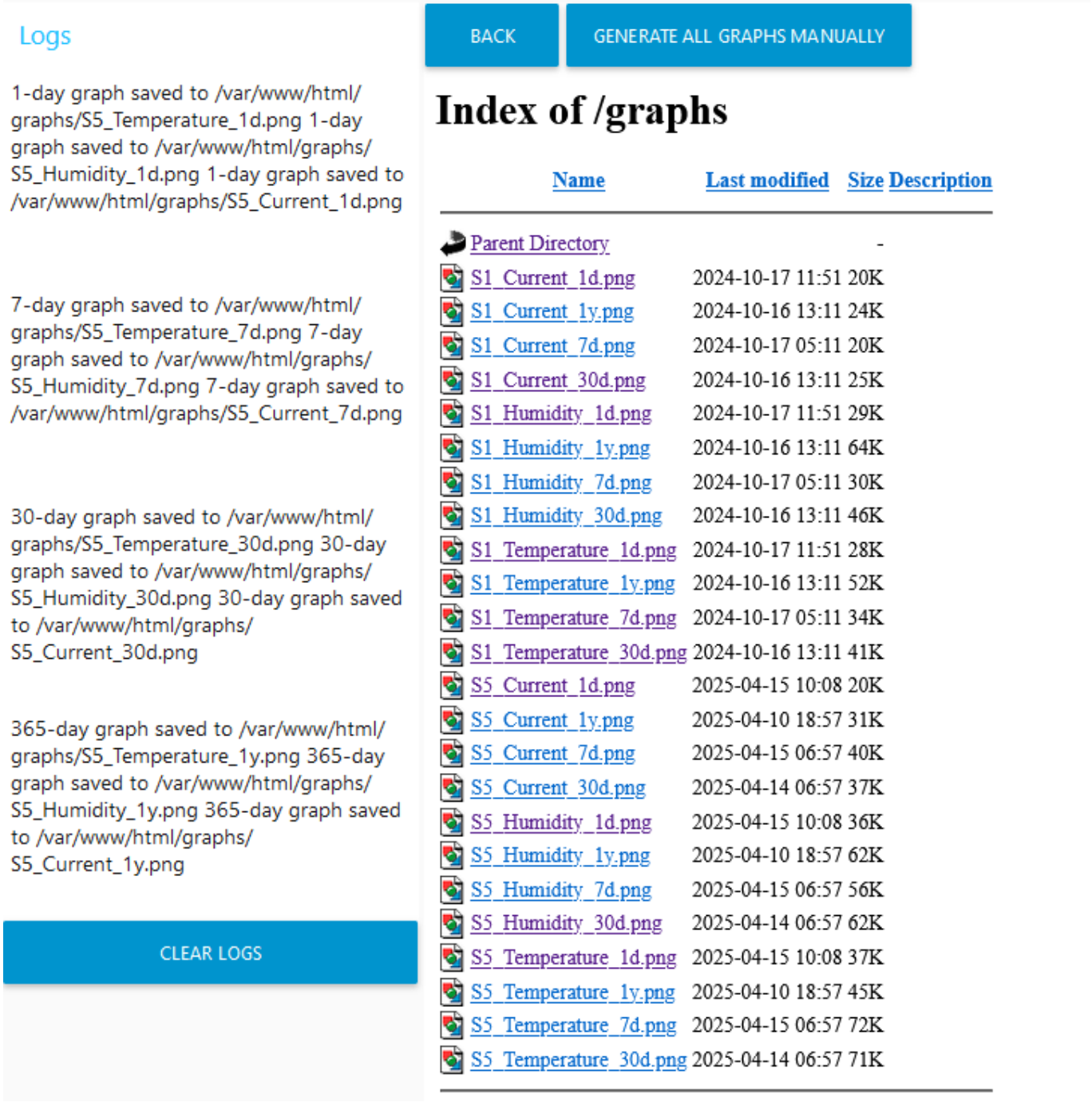
Tabela z logami

Zakładka tabeli z logami wyświetla szczegółowe informacje o zdarzeniach systemowych. Każdy rekord zawiera znacznik czasowy konkretnego zdarzenia, identyfikator kontenera, którego dotyczy zdarzenie oraz treść powiadomienia. Logi informują przede wszystkim o przekroczeniach zdefiniowanych progów parametrów środowiskowych (np. temperatura poza dopuszczalnym zakresem).




Ustawienia progów powiadomień

Interfejs ustawień pozwala użytkownikowi na konfigurację progów upozorowaniowych dla poszczególnych parametrów środowiskowych. Parametry te to górne i dolne limity temperatury, wilgotności oraz natężenia prądu elektrycznego. Po lewej stronie widoczna jest aktualna konfiguracja, natomiast po prawej stronie użytkownik może intuicyjnie dostosowywać te wartości przy użyciu suwaków.



Interfejs do przeglądania grafów

Sekcja przeglądania wykresów pozwala na generowanie oraz wizualizację wcześniej zapisanych plików graficznych prezentujących historyczne pomiary parametrów. Po lewej stronie widoczne są logi informujące użytkownika o statusie generacji wykresów wraz z możliwością ich wyczyszczenia. Po prawej stronie znajduje się widok katalogu z wygenerowanymi wykresami, posortowanymi według parametrów oraz zakresów czasowych (np. 1 dzień, 7 dni, 30 dni oraz 1 rok). Użytkownik może również ręcznie wymusić wygenerowanie wszystkich wykresów za pomocą dedykowanego przycisku.

# Spisy

# Wykaz załączników

1. Kod dla platformy Arduino MEGA
   arduino.zip
2. Skrypt bazy danych oraz dane które system zebrał
   db.zip