If you want to cite this work then use the DOI indetifier:
Ostrowski, K. (2025). Design of a measurement and control system based on Arduino (Version 1). Zenodo. https://doi.org/10.5281/zenodo.15270122

Arduino: Measurement and control system project

Kacper Ostrowski

Introduction

Example of a container with satellite equipment

In an era of dynamic development of automation and remote monitoring systems, solutions enabling continuous monitoring and control of environmental parameters in various types of technical facilities are becoming increasingly important. One such facility is containers with electronic equipment, often installed in the immediate vicinity of communication antennas, where it is crucial to maintain appropriate temperature and humidity conditions and supervise the condition of the infrastructure.

Arduino Mega 2560 board

Arduino Ethernet Shield board

The purpose of this thesis is to present the design and implementation of a measurement and control system based on the Arduino MEGA platform using the Ethernet Shield module. This system is designed to monitor and control the environment inside a technical container, which houses the equipment supporting the operation of a transmit/receive antenna.

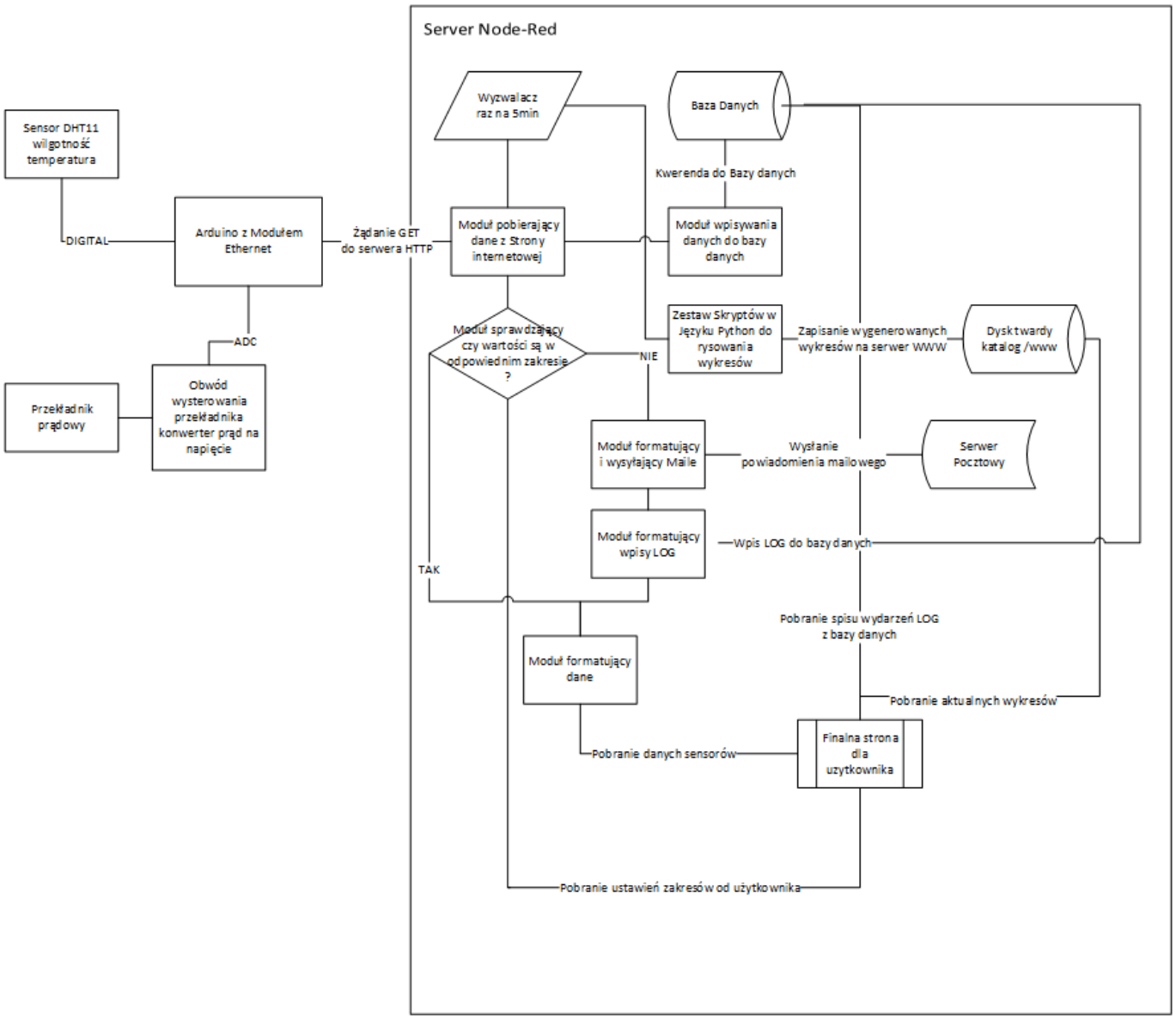
The functionality of the system includes:

- measurement of air temperature and humidity,
- control of current consumption by the heater located inside the container,
- detection of door opening and closing,
- monitoring the presence of liquids via a flood sensor.

The collected data is transmitted via a network interface, allowing it to be read remotely and integrated into external monitoring systems. The solution is designed to increase the reliability of the equipment inside the container by detecting abnormalities at an early stage and ensuring appropriate environmental conditions.

General description of the operation of the measurement and control system

The block diagram below shows a simplified structure of the operation of the measurement and control system based on the Arduino MEGA microcontroller and the Node-RED environment running on the server. The purpose of the system is to acquire environmental data, analyse its correctness, save it to a database and present the results to the end user.



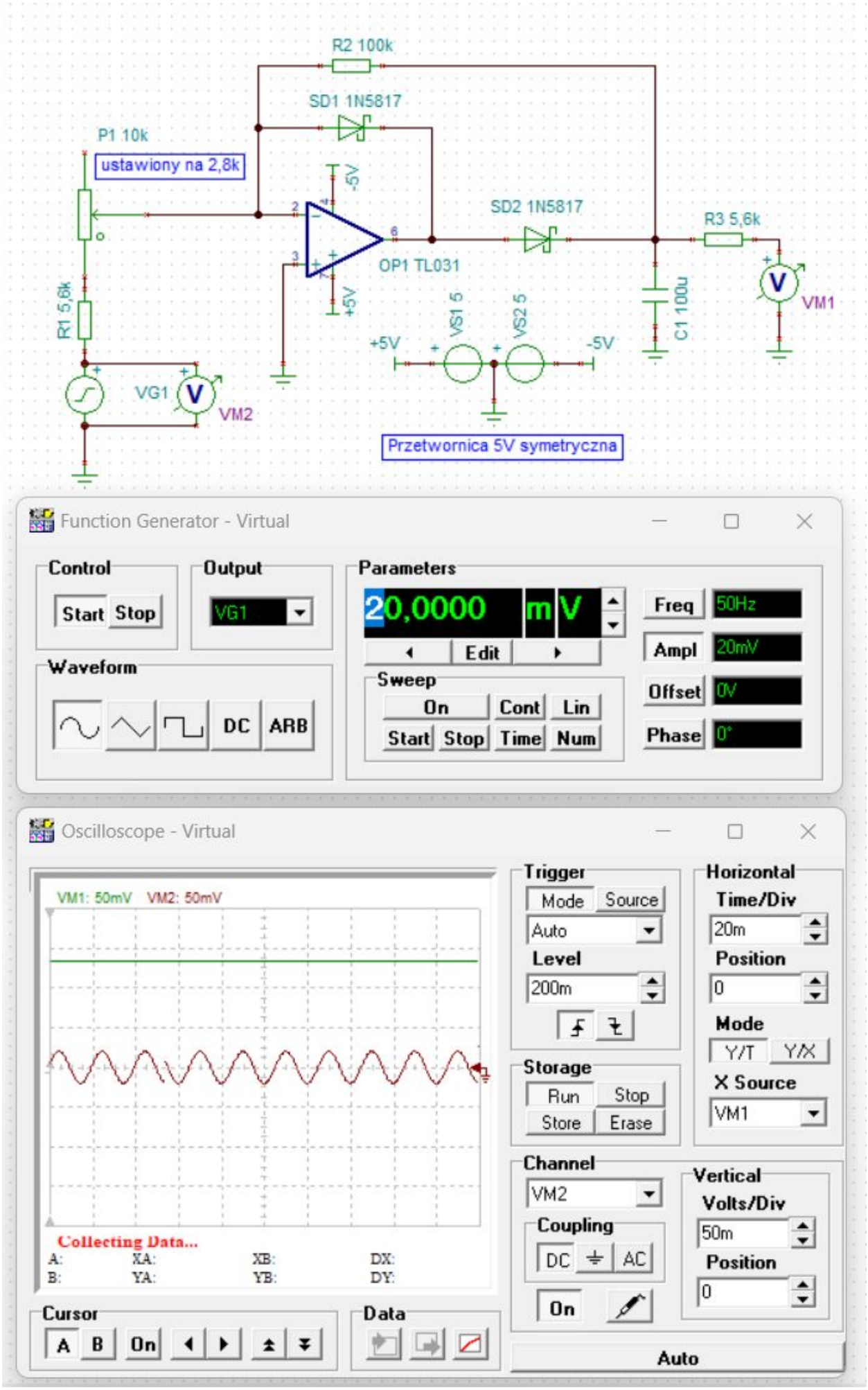
Block diagram of the measurement and control system

The system consists of the following main parts:

- **Arduino module with Ethernet Shield** - is responsible for collecting data from the sensors and sending it to the server via HTTP requests.
    - *DHT11 sensor* - measures air temperature and humidity.

  Description: The sensor used in this version is because the temperature fluctuations inside the container do not fall below zero
    - *Current transformer and measuring circuit* - allows the current consumption of the heater to be measured, the signal is converted into voltage and read out by an ADC.

  Description: the current transformer used here is a simple transformer, which requires the circuit below to function correctly and convert the voltages to the correct ones that can be read by the Arduino ADC.


R2 100k
SD1 1N5817
P1 10k
ustawiony na 2,8k
-5V
SD2 1N5817
R3 5,6k
OP1 TL031
+5V
R1 5,6k
VS1 5
VS2 5
C1 100u
VM1
+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

Schematic of the circuit to operate the current transformer together with measurements taken in the TINA-TI simulator

- **Server with Node-RED software** - implements the main logic of the system:
    - cyclic data retrieval from the Arduino,
    - analysis of the range of correctness of the data,
    - writing data to the database,
    - generation of graphs and reports,
    - sending email notifications.

  Description of individual modules:

    - *Module checking data values* - decides whether the environmental parameters are within the set range.

  Description: Consists of several parts one part takes information from the Arduino board then the data is verified whether it is within the relevant ranges.

    - *The formatting and sending module* - sends a warning when alarm thresholds are exceeded.

  Description: Retrieves what parameter has been exceeded on which container and sends an email

    - *Database recording and graph drawing module* - The data is saved and represented graphically using a set of Python scripts.

  Description: Formats an INSERT query into a database and then writes that data. The second part is a set of Python scripts that connect to the database and then retrieve and draw the data which is saved in the web directory of the apache server. The Dashboard in node-red then refers to these.

    - *Final user page* - A web page presenting the current sensor values, graphs and allowing the ranges of monitored parameters to be changed.

  Description: This is a dashboard built using the node-red tool presents graphs of current parameters allows you to switch power to the heater etc.

- **Communication and utility modules:**
    - *Mail server* - handles the sending of email messages.

Description: This is the external mail server to which the program sends emails. This is an external component we will not discuss its implementation.

- *Database and web server catalogue* - data is stored and made available to the user in the form of graphs and measurement history.

Description: These are external components that allow data to be collected and the generated graphs to be made available over the web.

The system is designed to operate reliably in harsh environmental conditions, enabling a rapid response if critical parameters are exceeded.

Overview of the Node-RED system principle

Node-RED is an open-source development environment that enables the graphical design of data flows (so-called 'data flows'). *flow*flow) by means of connections between so-called nodes (*nodes*). The environment is based on JavaScript (Node.js) and the configuration of the system logic is done using an intuitive browser interface.

Basic concepts

The basic unit of data in the Node-RED system is the object `msg`object, which contains the data transferred between nodes. Each *node* accepts input, performs the specified operation and then sends the result to the subsequent nodes as an updated object `msg`. This object has a JSON-like structure and can contain the following properties:

- `msg.payload` - the main part of the message containing the data sent by the system.
- `msg.topic` - label describing the subject of the message (used e.g. for filtering).
- `msg.timestamp` - timestamp of the message.
- other custom properties - can be added by the user or by nodes (e.g. `msg.sensorType`, `msg.alert`, etc.).

Data flow between nodes

Nodes (*nodes*) are specialised functional blocks that perform specific operations. Examples of node types:

- `inject` - initiates a data flow (e.g. every 5 minutes).
- `http request` - fetches data from an external source (e.g. Arduino).
- `function` - processes data using JavaScript code.
- `switch` - checks logical conditions and directs data to the appropriate paths.
- `debug` - is used to test and preview message content.
- `email` - sends email messages.
- `database` - writes data to a database (e.g. MySQL, SQLite).

When data is transferred from one node to another, the object `msg` is passed on. Each node can read, modify or branch the message, allowing complex information processing to be built.

Application in this system

In the measurement and control system under discussion, Node-RED acts as the central logic for data management and processing. An example data flow is as follows:

1. **Time trigger** (`inject`) launches an HTTP request to the Arduino every 5 minutes.
2. **HTTP node** retrieves measurement data (e.g. temperature, humidity, current).
3. **Function node** analyses the data and saves it in an object `msg.payload`.

4. **Conditional node** (`switch`) checks whether the data is within an acceptable range.
5. Depending on the result, the data is directed to:
    - **database node**where it is written,
    - **Logging node**which creates an entry in the system history,
    - **an e-mail sending node**Node which sends an e-mail when an alarm occurs.
6. The data is also transferred to the module responsible for generating graphs and presented to the user in graphical form on the website.

Thanks to its modular design, the Node-RED system allows the operating logic to be easily modified or extended without having to write a lot of code. Each node can be edited visually and debugging takes place in real time.

Discussing and analysing flows on the Node-RED platform

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

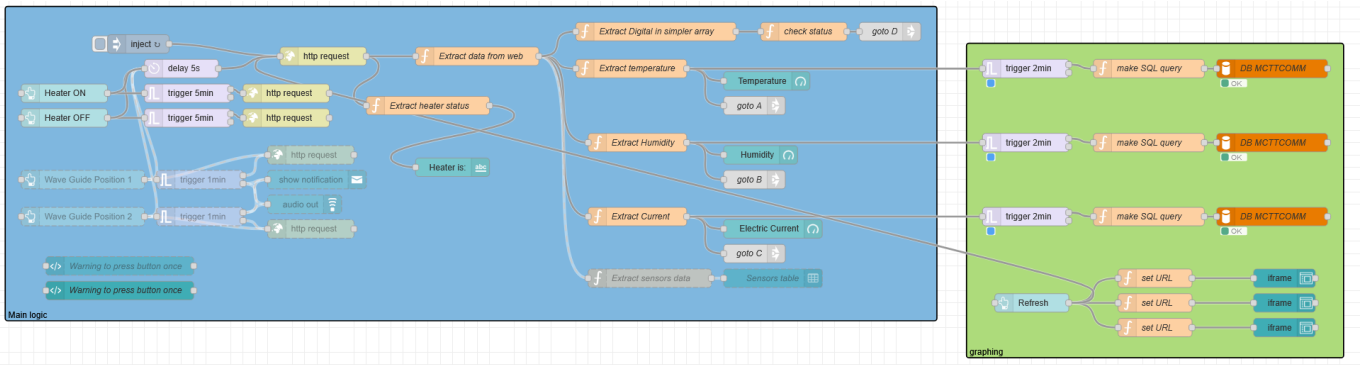
Status: ON

Example of a page returned by Arduino

The Node-RED environment was used as the central component of the system responsible for communication with the measuring device (Arduino), data processing and its further distribution to other elements of the IT infrastructure (database, web server, notification system). Due to its modular design and intuitive graphical interface, Node-RED is ideal for automation, monitoring and real-time data collection systems.

The programme created was divided into a set of logical blocks, each of which is responsible for performing a specific task within the system. Thanks to this structure, it was possible to ensure high transparency of operation and ease of expansion with additional functionalities.

Blocks: Main Logic and Graphing



View of the blocks from the Node-RED application

Listings of individual function nodes

```
// 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;
```

Explanation of the principle of nodes

1. Node: Extract Data from Web The node responsible for processing HTML data extracted from a web page. It first assigns the content `msg.payload` to the variable `payload` and sets the container name as S5 (`msg.container`). It then creates an empty object `result`containing two sections: `analog_sensors` and `digital_inputs`. Using regular expressions `/ <tr><td>([^<]+)<\/td><td>([^<]+)<\/td> <\/tr> /g` searches the HTML data to extract the names and values of the analogue and digital sensors. In each loop `while`the found data is assigned to the corresponding fields in the object `result`. Finally, the resulting JSON object is assigned to the `msg.payload` and passed on.

2. Node: Extract heater status This node is used to read the heater status based on the HTML content passed in the `msg.payload`. Using the regular expression `/<p class='status-(on|off)'>Status: (ON|OFF)/i` an HTML fragment containing information about the status of the device is searched. If the match is successful (`if (statusMatch)`), the second element of the resulting array `statusMatch[2]` the status of the heater is retrieved (e.g. `ON` or `OFF`). This value is assigned to `msg.payload` and passed on in the flow.

3. Node: Extract Digital in simpler array The node transforms the input data from the object `msg.payload.digital_inputs` into a simpler form - an array of objects. At first, an empty array is created `digitalSensorsArray`to which objects containing the name and value of each digital sensor are successively added (`name`, `value`). The data is retrieved using a loop `for (let [sensorName, sensorValue] of Object.entries(...))`and the result is assigned to `msg.payload` as a new, simplified data structure.

4. Node: check status The node analyses the input data contained in `msg.payload`, which constitute an array of digital sensors. A check is performed for each sensor - if its name does not start with `"Waveguide Position"` and its value is `"0"`then the sensor is added to a new array `sensorsWithZeroValue`. If the array contains any elements at the end of the loop, it is set as a new `msg.payload` and passed on. Otherwise, the node returns `null`, which interrupts further message processing.

5. Node: Extract Temperature The node extracts the temperature value from the field `analog_sensors["Temperature"]` contained in `msg.payload`and then converts it to an integer using `parseInt(...)`, assigning the result back to `msg.payload`. It also sets the identifiers `msg.name` and `msg.container` to `"Temperature"` i `"S5"`.

6. Node: Extract Humidity Analogous to the previous node, it extracts the humidity value from the `analog_sensors["Humidity"]`, converts it to an integer with the function `parseInt(...)`

and assigns it to `msg.payload`. It also sets the name of the parameter and the container in `msg.name` and `msg.container`.

7. Node: Extract Current Node responsible for extracting current values from the field `analog_sensors["Electric Current"].`. Uses a regular expression `msg.payload.match(/[.]+/)` to extract a floating point number from the text, and then converts it to type `float`. It also adds fields to describe the parameter name and container.

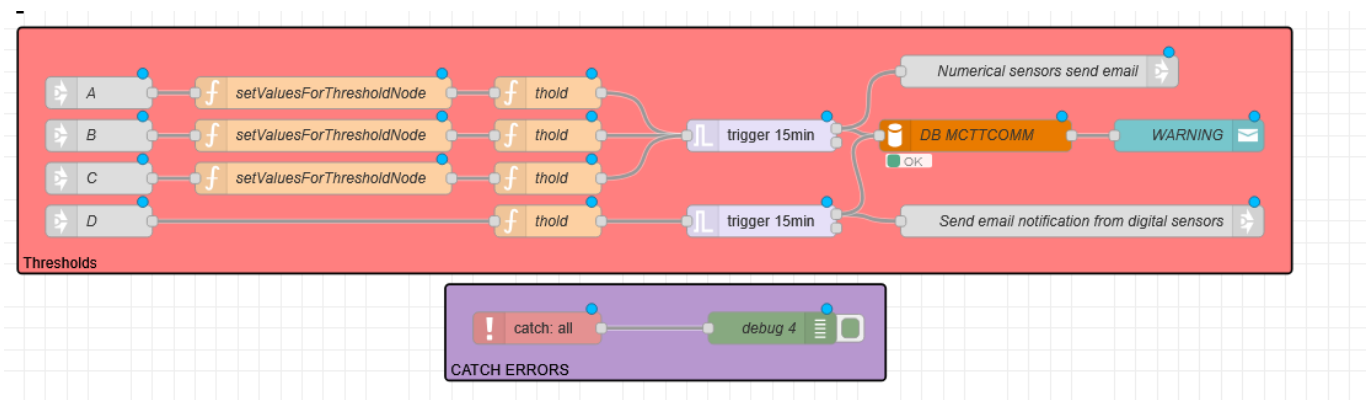
8. Node: make SQL query Generates a SQL query that inserts a temperature measurement value into the table `S5_Temperature` with the current timestamp. Creates a field `msg.topic` containing a query in the format
`INSERT INTO ... VALUES (now(), '${msg.payload}')%.`.

9. Nodes: set URL Each node sets the corresponding URL of the parameter graph in the field `msg.url`:

   - First node: `S5_Temperature_1d.png`
   - Second node: `S5_Current_1d.png`
   - Third node: `S5_Humidity_1d.png`

   The addresses refer to image resources generated by an external web server.

Blocks: Thresholds and CATCH ERRORS



View of blocks from the Node-RED application

Listings of individual function nodes

```
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
```

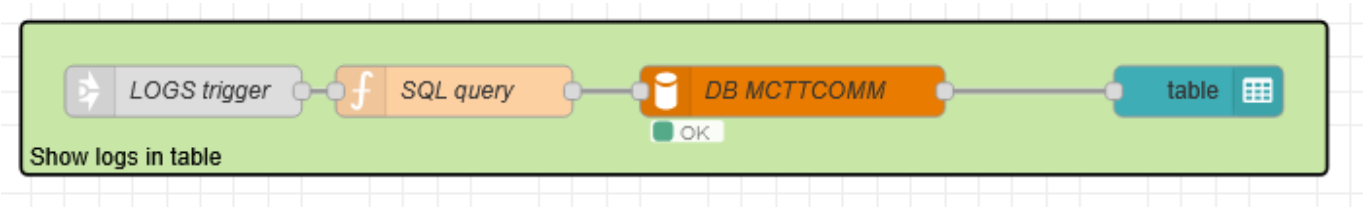
```
let upperThreshold = msg.upperThreshold;  // Upper boundary
let container = msg.container; // Name of the container
msg.url = "http://api.ttcomm.net/graphs/S5_Temperature_1d.png";
msg.value = value; //used for extracting it for email

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;
}
```

Explanation of the principle of nodes

1. Node: setValuesForThresholdNode This node retrieves the defined temperature thresholds from global memory: lower `temperatureLowerThold` and upper `temperatureUpperThold`, stored in the configuration file (second parameter: `"file"`). It sets them in the message as fields `msg.lowerThreshold` and `msg.upperThreshold`, allowing their further use in comparative logic.

2. Node: thold The node is used to compare the measurement values with the set thresholds. Based on the fields `msg.payload` (value), `msg.lowerThreshold`, `msg.upperThreshold`, `msg.name` and `msg.container` a check is performed to see if the value falls outside the specified limits. If so, an SQL query is generated (`msg.topic`) that adds the record to the table `thold_log`containing the time, container name and description of the overrun. Additionally, the fields are assigned `msg.url` (link to the chart) and `msg.value` (used, for example, for an email message). If the value is within acceptable limits, the node returns `null`, stopping further message flow.

Block: Show logs in table



View of blocks from the Node-RED application

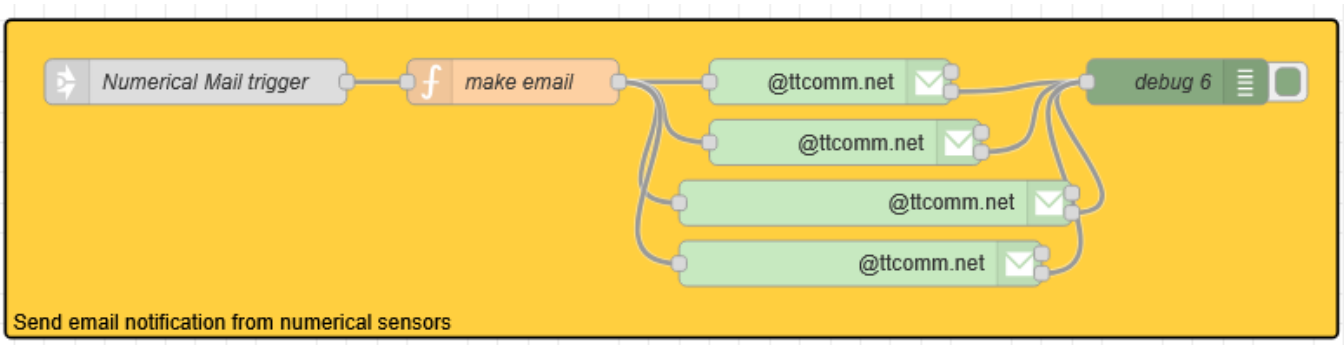
Listings of individual function nodes

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

Operating principle

Here the working principle is very simple we retrieve everything that is in the table `thold_log` and then display it in the interface as a table.

Block: Send Numerical notificaiton from numerical sensors



View of blocks from the Node-RED application

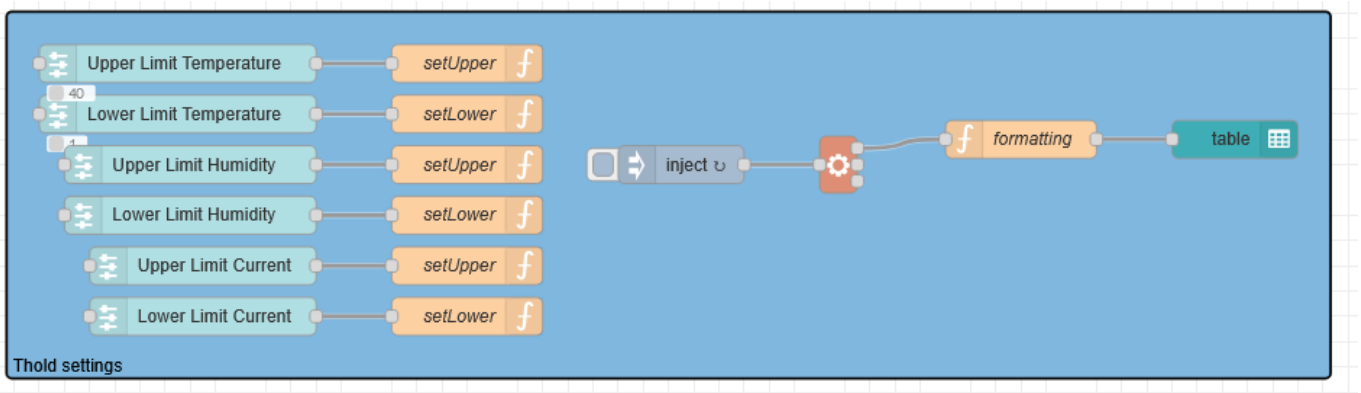
Listings of individual function nodes

```
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;;
```

Operating principle

Here we retrieve from the object `msg` object in order to create the email message and then send it to the appropriate users

Block: Thold settings



View of blocks from the Node-RED application

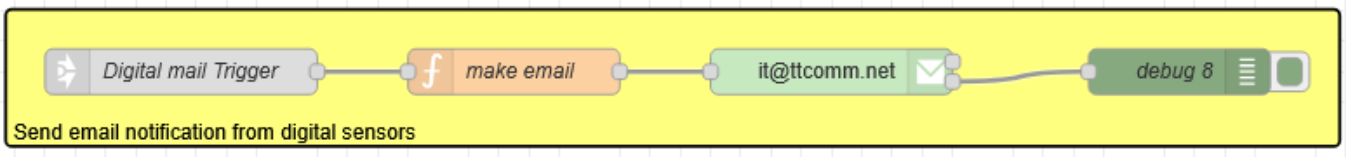
Listings of individual function nodes

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

Operating principle

Here, we set the global values for the individual thresholds. These are then used in the previous flows discussed here.

Block: Send email notification from digital sensors



View of blocks from the Node-RED application

Listings of individual function nodes

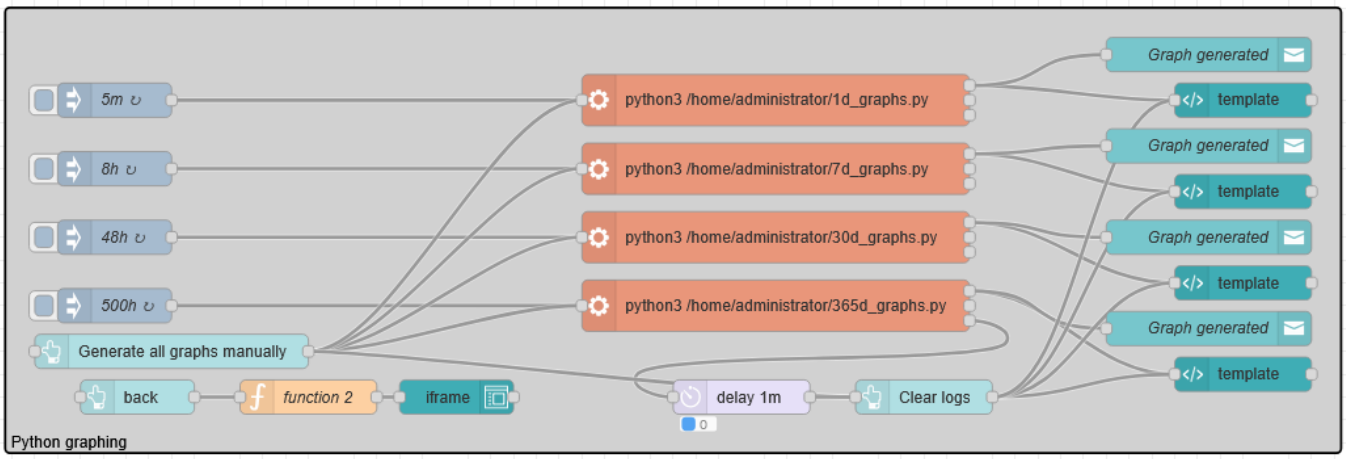
```
msg.from = "node-red@ttcomm.net"
msg.topic = `Alert: ${msg.container} digital sensors changed status`;
msg.payload = `Sensor in container ${msg.container} status:
${msg.sensorStatusString}`;
return msg;
```

Operating principle

This block is needed because of the slightly different structure of the emails for sensors with

numerical values.

Block: Python graphing



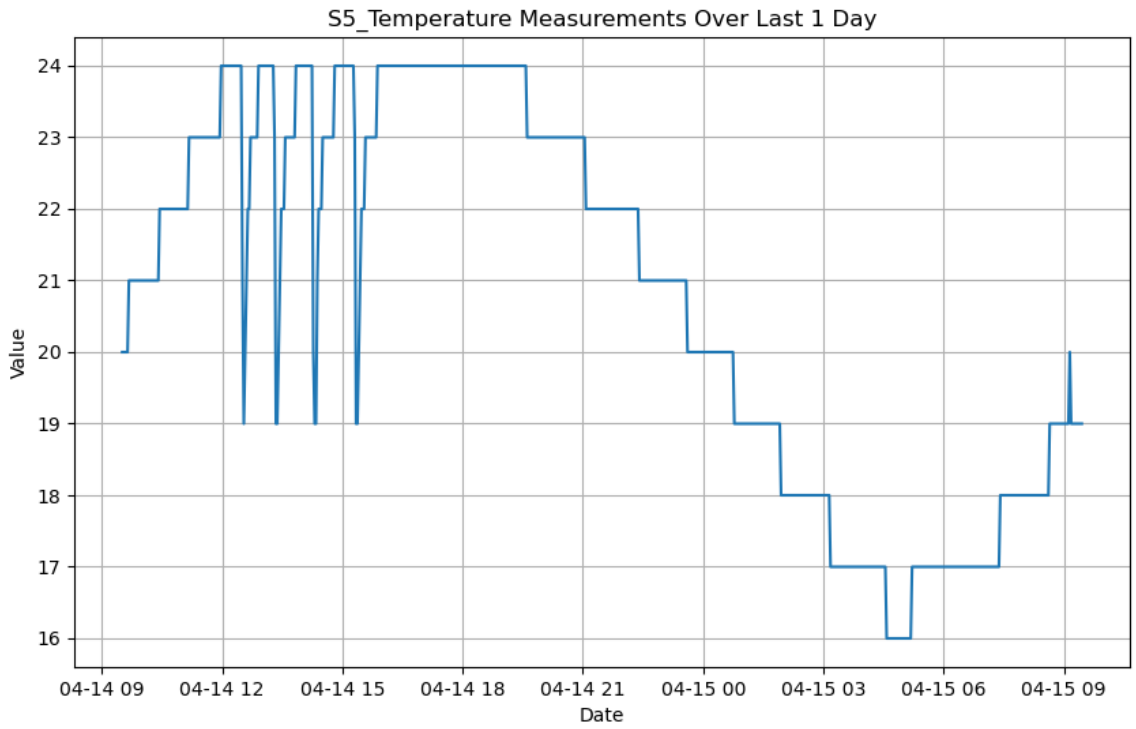
Block view from the Node-RED application

Operating principle

The Node-RED flow presented here implements the automatic and manual generation of graphs based on Python scripts. Four time triggers (every 5 minutes, 8 hours, 48 hours, and 500 hours) initiate the execution of the corresponding scripts that generate graphs for periods of: 1, 7, 30 and 365 days. In addition, a button was used to manually generate all the charts simultaneously and another button to return to the default view of the charts. The results of running scripts are presented to the user on the Node-RED dashboard through UI elements displaying script execution logs and notifications ('toast'). The entire interface is complemented by an iframe element in which the user can conveniently view the generated charts.

Python script for generating graphs

The script is responsible for automatically generating graphs of environmental parameters such as temperature, humidity and power consumption based on data stored in a MySQL database. This data is then processed and visualised in the form of PNG graphs, which can be published on the local network or in the end user's browser.



Example of a graph generated by the script

Description of the script

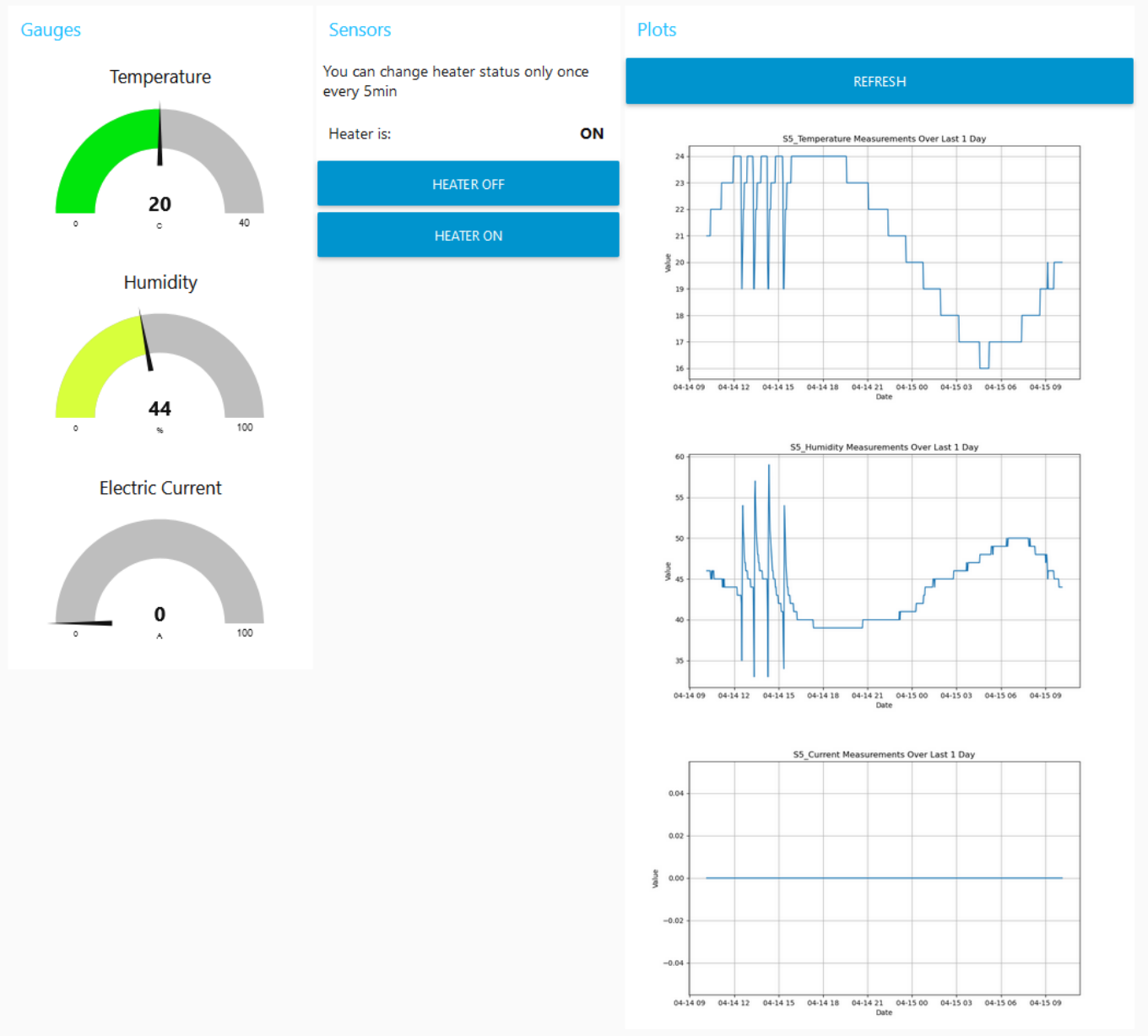
The script starts by importing the necessary libraries: `pymysql` for communication with the database, `matplotlib.pyplot` for graph generation, `pandas` for data analysis and `datetime` for handling time (`line`).

Function `plot_1d_graph(table_name)` takes as argument the name of the database table from which the data will be extracted (`line`). A connection to the MySQL database is then established (`line9-13%''), and the data from the columns ''DATE'' and ''VALUE'' are downloaded into the data frame ''DataFrame'' using an SQL query (''line''). After retrieving the data, the connection is closed (''line 19%`), and the column DATE is converted to type `datetime (line 22%''), which allows the data to be filtered over a specified time range. The time range is set to the last 24 hours, which implements the extract: <code python> start_date = datetime.now() - timedelta(days=1) filtered_df = df[df['DATE'] >= start_date] </code> Then, using the library ''matplotlib''library, a line graph is generated (''line 27-31''). Axis captions are placed on the graph, as well as the title of the graph, which includes the name of the table. The generated chart is saved as a PNG file in the directory ''/var/www/html/graphs/'' under the name corresponding to the table name with the suffix ''_1d'' (''line34''). Finally, the script calls the function ''plot_1d_graph'' three times for different tables: * ''S5_Temperature'' * ''S5_Humidity'' * ''S5_Current%%`

Each of these calls results in the generation of a separate graph for the last 24 hours for a given parameter.

```
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')
```

Graphical interface



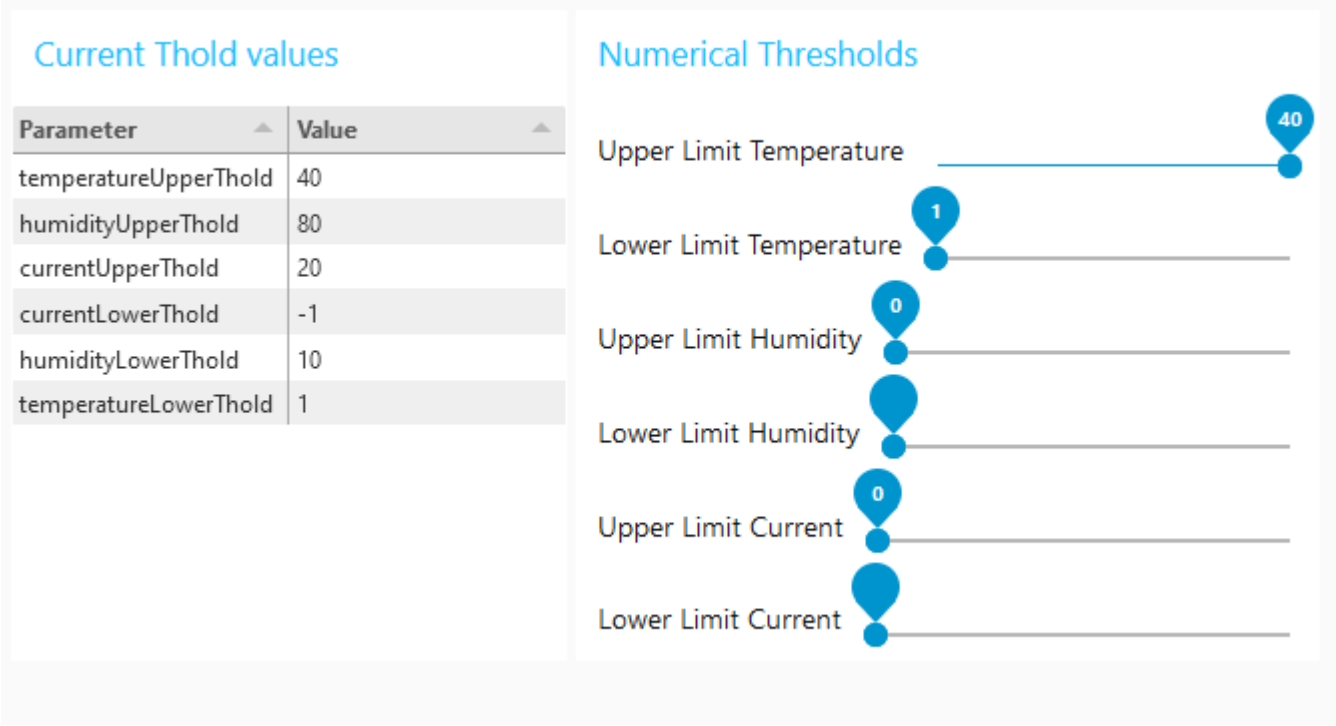
Main page of the interface

The main page of the application presents key information in an easy-to-read graphical form. It features three pointer indicators (gauges), showing readings of current environmental parameters: temperature, humidity and electric current. In addition, the user has the option of manually controlling the heater, which can be switched on or off at a frequency of up to once every five minutes. To the right is a space for displaying current graphs showing the history of measurements for the last day.

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).

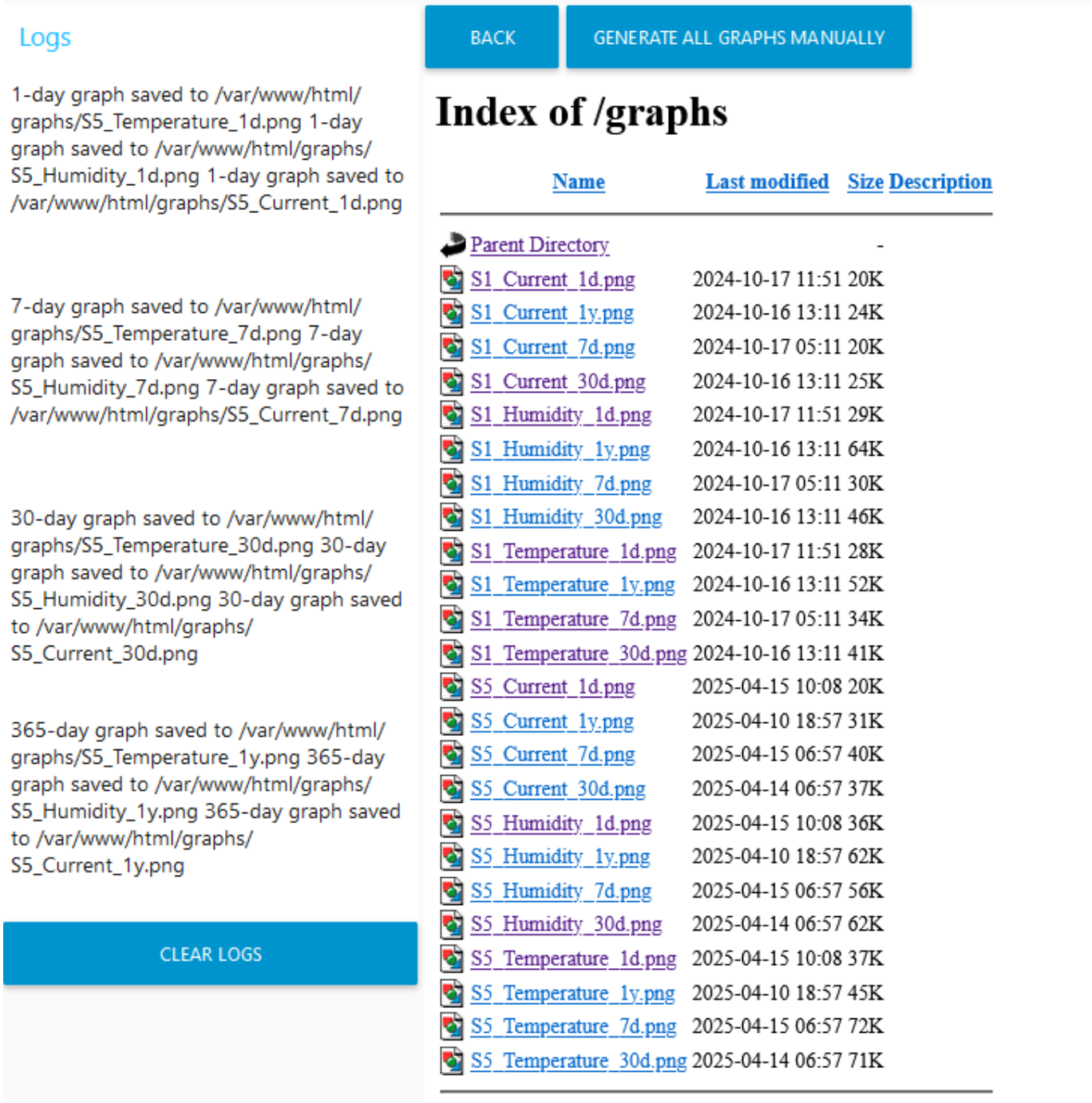
Table with logs

The log table tab displays detailed information about system events. Each record contains the timestamp of the specific event, the identifier of the container affected by the event and the content of the notification. The logs mainly report on exceedances of defined thresholds of environmental parameters (e.g. temperature outside the permissible range).



Settings for notification thresholds

The settings interface allows the user to configure the upozone thresholds for specific environmental parameters. These parameters are the upper and lower limits for temperature, humidity and electrical current. On the left, the current configuration is visible, while on the right the user can intuitively adjust these values using the sliders.



Interface for viewing graphs

The graph viewing section allows the generation and visualisation of previously saved graph files showing historical parameter measurements. On the left-hand side, logs are displayed informing the user of the status of the graph generation, together with the possibility of clearing them. On the right is a directory view of the generated graphs, sorted by parameter and time ranges (e.g. 1 day, 7 days, 30 days and 1 year). The user can also manually force the generation of all charts using a dedicated button.

Clips

List of annexes

1. Database script and data which the system has collected
   db.zip
2. Code for the Arduino MEGA platform

arduino.ino

```
#include <Streaming.h>
#include <SimpleDHT.h>
#include <SPI.h>
#include <Ethernet.h>
#include <C:\Users\kostrowski\Desktop\program\AVT5636lib.h>
#include <Servo.h>
#include <C:\Users\kostrowski\Desktop\program\MemoryFree.h>


AVT5636 myBoard;
Servo myServo;

SimpleDHT11 dht11;

byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 80, 53);

// Initialize the Ethernet server library
EthernetServer server(8080);

float temperature;
float humidity;

int term2=0;
int  term3=0;
int term4=0;
int  term5=0;

uint32_t prevMillis = millis();


void setup() {
  pinMode(22, INPUT_PULLUP);
  pinMode(24, INPUT_PULLUP);
  pinMode(26, INPUT_PULLUP);
  pinMode(28, INPUT_PULLUP);
  pinMode(30, INPUT_PULLUP);
  myBoard.init();
  myServo.attach(PULSE2_PIN);
  //delay(3000);
```

```
  //Serial.begin(9600);
  //while (!Serial) {
  //  ; // wait for serial port to connect. Needed for native USB port
only
  //}

  Ethernet.begin(mac, ip);
  server.begin();
  //Serial.print("server is at ");
  //Serial.println(Ethernet.localIP());

}

String HTTP_req;
bool LED_status4 = 0;
bool LED_status3 = 0;
bool LED_status2 = 0;

int IntTemperature;
int IntHumidity;

unsigned long previousMillis = 0;

float CurrentSensor = 0;

void loop() {
  CurrentSensor = ((5.0/1024.0)*analogRead(10))*10;
  //Serial.println(CurrentSensor);

  LED_status4 = 0;
  LED_status3 = 0;

  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= 60000) {

    previousMillis = currentMillis;
    delay(2000);

    byte temperature = 0;
    byte humidity = 0;
    byte err = SimpleDHTErrSuccess;

    if ((err = dht11.read(31, &temperature, &humidity, NULL)) !=
SimpleDHTErrSuccess) {
      //Serial.print("Failed to read from DHT sensor, err=");
      //Serial.println(err);
      return;
    }
```

```
    IntTemperature = temperature;
    IntHumidity = humidity;
    term2 = temperature;
    term3 = humidity;
  }


  term4 = 444;
  term5 = 555;

  EthernetClient a = server.available();
  serveWebsite(a);


}

EthernetClient serveWebsite(EthernetClient client){
  if (client) {
    //Serial.println("new client");
    bool currentLineIsBlank = true;
    while (client.connected()) {
      //Serial.println("test1");
      if (client.available()) {
        //Serial.println("test2");
        char c = client.read();
        //Serial.write(c);
        HTTP_req += c;
        if (c == '\n' && currentLineIsBlank) {
          //Serial.println("test3");
          if (HTTP_req.indexOf("LED4=On") > -1) { LED_status4 = 1; }
          if (HTTP_req.indexOf("LED4=Off") > -1) { LED_status4 = 0; }
          if (HTTP_req.indexOf("LED3=On") > -1) { LED_status3 = 1; }
          if (HTTP_req.indexOf("LED3=Off") > -1) { LED_status3 = 0; }
          if (HTTP_req.indexOf("LED2=On") > -1) { LED_status2 = 1; }
          if (HTTP_req.indexOf("LED2=Off") > -1) { LED_status2 = 0; }

          client.println(F("HTTP/1.1 200 OK"));
          client.println("Content-Type: text/html");
          client.println("Connection: close");
          //client.println("Refresh: 10; URL=/");
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
          client.println("<head>");
          client.println("<meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
          // client.println("<style>");
          // client.println("body { font-family: Arial, sans-serif;
margin: 0; padding: 0; font-size: 12px; }");
          // client.println("h2 { color: #333; }");
          // client.println(".container { width: 80%; margin: 0 auto;
```

```
padding: 20px; }");
          // client.println("table { width: 100%; border-collapse:
collapse; margin-bottom: 20px; }");
          // client.println("th, td { padding: 8px; text-align: left;
border-bottom: 1px solid #ddd; }");
          // client.println("tr:hover { background-color: #f5f5f5; }");
          // client.println(".btn { padding: 8px 16px; margin: 5px;
text-decoration: none; color: white; border: none; display: inline-
block; font-size: 12px; }");
          // client.println(".btn-on { background-color: #4CAF50; }");
          // client.println(".btn-off { background-color: #f44336; }");
          // client.println(".status-on { background-color: #39FF14;
}"); // Neon green
          // client.println(".status-off { background-color: #FF073A;
}"); // Neon red
          // client.println("</style>");
          client.println("</head>");
          client.println("<body>");
          client.println("<div class=\"container\">");
          client.println("<h2>System Pomiarowo-Kontrolny TTCOMM
Sp.z.o.o.</h2>");
          client.println("<h4>Opracowano przez: Kacper
Ostrowski</h4>");
          client.println("<h4>Wersja z dnia: 11.06.2024</h4>");
          client.println("<h2>Analog Sensor Readings</h2>");
          client.println("<table>");
          client.println("<tr><th>Sensor</th><th>Value</th></tr>");

          client.println("<tr><td>Temperature</td><td>" +
String(IntTemperature) + " C</td></tr>");
          client.println("<tr><td>Humidity</td><td>" +
String(IntHumidity) + " %</td></tr>");

          client.println("<tr><td>Electric Current</td><td>
"+String(CurrentSensor)+"     A</td></tr>");
          client.println("</table>");
          client.println("<h2>Digital Input Readings</h2>");
          client.println("<table>");
          client.println("<tr><th>Sensor</th><th>Value</th></tr>");
          client.print("<tr><td>Door Open/Closed</td><td>");
          client.print(digitalRead(22));
          client.println("</td></tr>");
          client.print("<tr><td>Flood sensor</td><td>");
          client.print(digitalRead(24));
          client.println("</td></tr>");
          client.print("<tr><td>Waveguide Position 1</td><td>");
          client.print(digitalRead(26));
          client.println("</td></tr>");
          client.print("<tr><td>Waveguide Position 2</td><td>");
          client.print(digitalRead(28));
          client.println("</td></tr>");
```

```
          client.print("<tr><td>Waveguide Power Supply</td><td>");
          client.print(digitalRead(30));
          client.println("</td></tr>");
          client.println("</table>");
          client.println("<h2>Control Buttons</h2>");
          client.println("Override Control Buttons:");
          client.println("<form method='get'>");

          // LED4 control and mode selection
          client.println("<p>Waveguide position 1: <button name='LED4'
type='submit' class='btn btn-on' value='On'>ON</button> Please check
afterwards the status of position in the above table");
          //client.println("<button name='LED4' type='submit'
class='btn btn-off' value='Off'>OFF</button>");

          if (LED_status4) {
            myBoard.ledOn(4);
            delay(3000);
            myBoard.ledOff(4);
            delay(3000);
          } else {
            myBoard.ledOff(4);
          }

          // LED3 control and mode selection
          client.println("<p>Waveguide position 2: <button name='LED3'
type='submit' class='btn btn-on' value='On'>ON</button> Please check
afterwards the status of position in the above table");
          //client.println("<button name='LED3' type='submit'
class='btn btn-off' value='Off'>OFF</button>");


          if (LED_status3) {
            myBoard.ledOn(3);
            delay(3000);
            myBoard.ledOff(3);
            delay(3000);
          } else {
            myBoard.ledOff(3);
          }

          // LED2 control and mode selection
          client.println("<p>Room Heater: <button name='LED2'
type='submit' class='btn btn-on' value='On'>ON</button>");
          client.println("<button name='LED2' type='submit' class='btn
btn-off' value='Off'>OFF</button>");


          if (LED_status2) {
            myBoard.ledOn(2);
            client.println("<p class='status-on'>Status: ON");
```

```
          } else {
            myBoard.ledOff(2);
            client.println("<p class='status-off'>Status: OFF");
          }

          client.println("</form>");
          client.println("</div>");
          client.println("</body>");
          client.println("</html>");

          HTTP_req = "";
          break;
        }
        if (c == '\n') {
          currentLineIsBlank = true;
        } else if (c != '\r') {
          currentLineIsBlank = false;
        }
      }
    }
    client.stop();

    //Serial.println("client disconnected");
  }
}
```