Hochschule Konstanz
Department of Computer Science

Ubiquitous Computing WS 24

Exercises Documentation

Contents

List of Figures

List of Tables

Acronyms

1 Exercise Introduction

1.1 Exercise 0: Intro

The first or zero exercise is quite forward and can bee seen as a warm-up exercise. The goal is to get familiar with the tools and the environment and write a small program for a blinking LED (see following code).

```
1   const int ledPin =    LED_BUILTIN;  // The onboard LED pin
2   const int delayTime = 1000;         // Delay time in milliseconds (1 second)
3
4   void setup() {
5     pinMode(ledPin, OUTPUT);
6   }
7
8   void loop() {
9     digitalWrite(ledPin, HIGH);       // Turn the LED on
10    delay(delayTime);
11
12    digitalWrite(ledPin, LOW);        // Turn the LED off
13    delay(delayTime);
14  }
```

1.2 Exercise 1; Internal RGB

In the first exercise we are going to use the internal RGB LED to switch between the three colors red, green and blue. Cause there is also a template for this exercise, we will not go into detail here.

```
1   #include <WiFiNINA.h>
2
3   const int delayTime = 500; // Delay time in milliseconds (0.5 seconds)
4
```

```
5       void setup() {
6         pinMode(LEDR, OUTPUT);
7         pinMode(LEDG, OUTPUT);
8         pinMode(LEDB, OUTPUT);
9       }
10
11      void loop() {
12        blink_red();
13        delay(delayTime);
14        blink_blue();
15        delay(delayTime);
16        blink_green();
17        delay(delayTime);
18      }
19
20      void blink_red(){
21        Serial.print("Red\n");
22        digitalWrite(LEDR, HIGH);   // Red
23        digitalWrite(LEDB, LOW);    // Blue
24        digitalWrite(LEDG, LOW);    // Green
25      }
26
27      void blink_blue(){
28        Serial.print("Blue\n");
29        digitalWrite(LEDR, LOW);    // Red
30        digitalWrite(LEDB, HIGH);   // Blue
31        digitalWrite(LEDG, LOW);    // Green
32      }
33
34      void blink_green(){
35        Serial.print("Green\n");
36        digitalWrite(LEDR, LOW);    // Red
37        digitalWrite(LEDB, LOW);    // Blue
38        digitalWrite(LEDG, HIGH);   // Green
39      }
```

1.3 Exercise 2: Temperature Sensor

In the second exercise we are going to use the internal temperature sensor to measure the temperature of the Arduino board. Also we will use the LEDs from the task before to visualize the temperature like described in the task.

The following figures show the different states of the LEDs depending on the temperature. As it can be seen in the figures, the board was cooled down using a cooling pad and heated up using a hair dryer.
The code for this exercise can be seen on the github repository `https://github.com/Smokey95/AIN_Ubiquitous_Computing/blob/main/exercise/exercise2_temperature/exercise2_temperature.ino`.

Figure 1.1: Temperature Sensor between 20 and 36 degrees Celsius

Figure 1.2: Temperature Sensor under 20 degrees Celsius

Figure 1.3: Temperature Sensor above 36 degrees Celsius

1.4 Exercise 3: Microphone

In the third exercise we are going to use the internal microphone to measure the sound level. Therefore we used the link provided in the task description. Like seen in the figures below, the LED turn on/off when a certain sound level is reached.

Like before the code for this exercise can be seen on the github repository `https://github.com/Smokey95/AIN_Ubiquitous_Computing/blob/main/exercise/exercise3_microphone/exercise3_microphone.ino` It has to be noted that the `while-loop` for preventing the program from running till a serial monitor is connected will only work if all monitors are closed (even in other Arduino IDE instances).

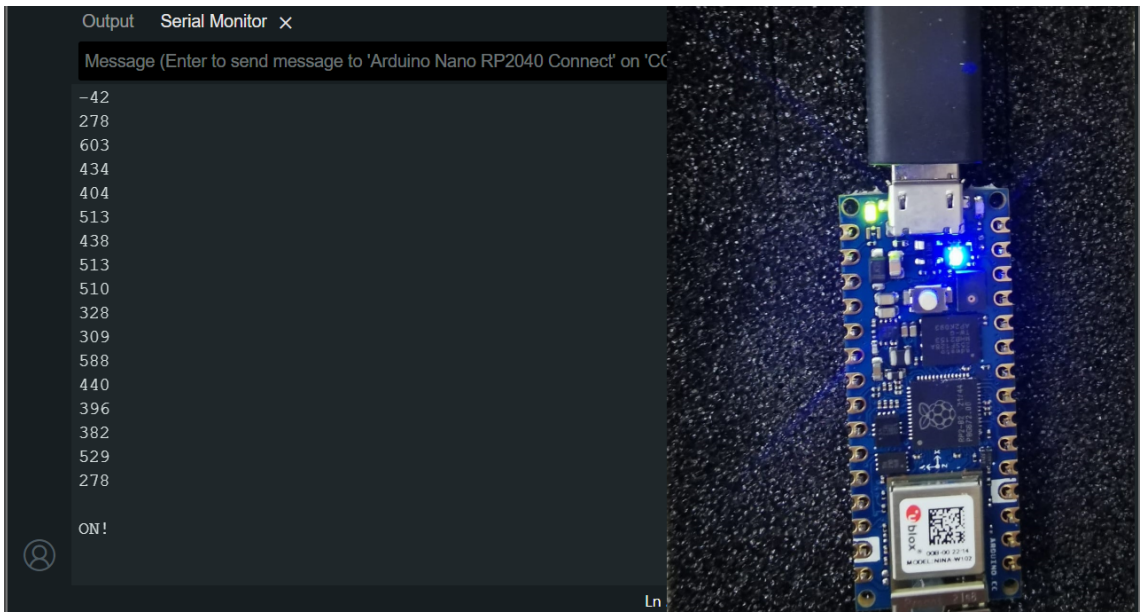

Figure 1.4: Sound detected, LED on

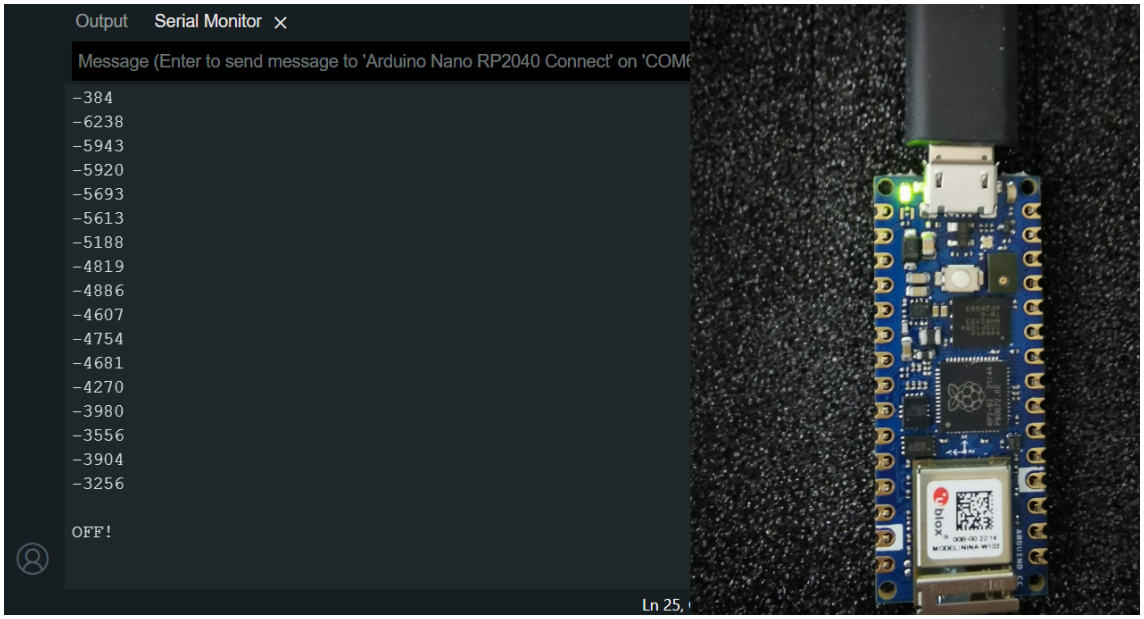

Figure 1.5: Sound detected, LED off

1.5 Exercise 4: Posture Detector. Accelerometer

In the last exercise we are going to use the internal accelerometer to detect the posture of the Arduino board. Therefore we used the link provided in the task description to realize the following code as well as the serial monitor output seen in figure 1.6. Like before the code can be seen on the github repository `https://github.com/Smokey95/AIN_Ubiquitous_Computing/blob/main/exercise/exercise4_posture/exercise4_posture.ino`.

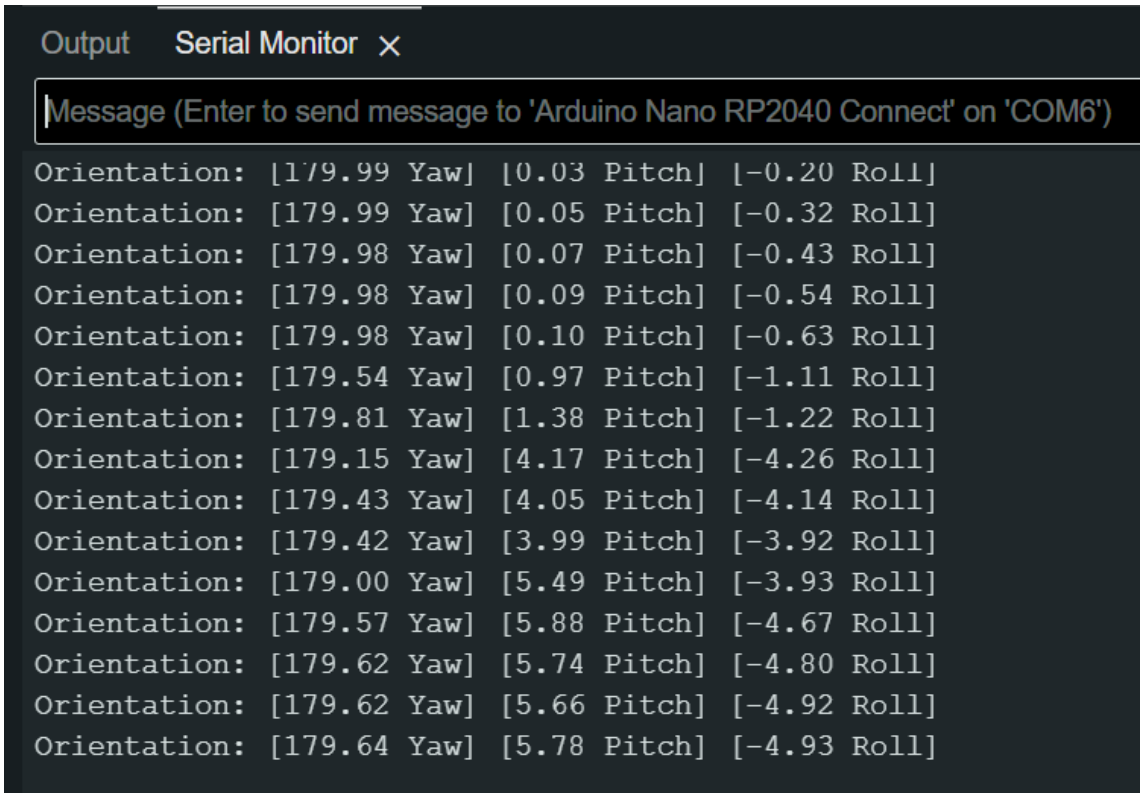

Figure 1.6: Posture Detector

2 Exercise 1: Node-RED

In this exercise we will get to know the programming tool Node-RED which will be used for wiring together hardware devices, APIs and online services in new and interesting ways.

As the tool is primary picture based, there will not be much code in this exercise. The exported `.json` of the Node-RED flow can be found in the sheet 2 folder of the `https://github.com/Smokey95/AIN_Ubiquitous_Computing` in the exercise section.

2.1 Exercise 1; Hello World

We will start with an basic example to get familiar with the tool bi just triggering a simple `Hello World` message.

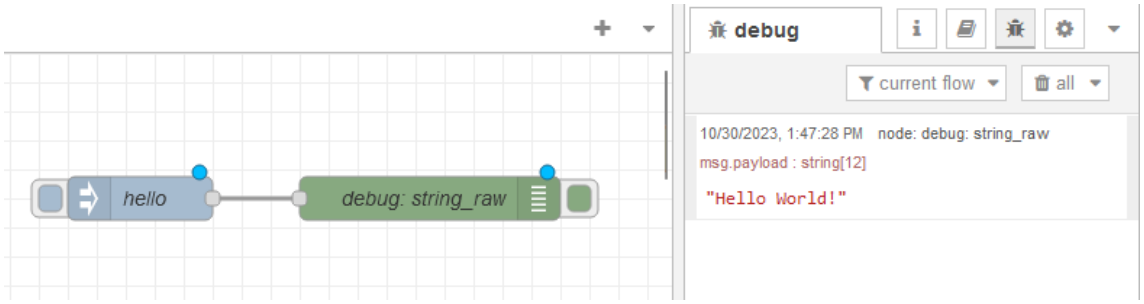

Figure 2.1: Hello World

2.2 Exercise 2: Change Message

In this exercise we will change the incoming `Hello World` message to `Hello Mars` and print it to the debug console.

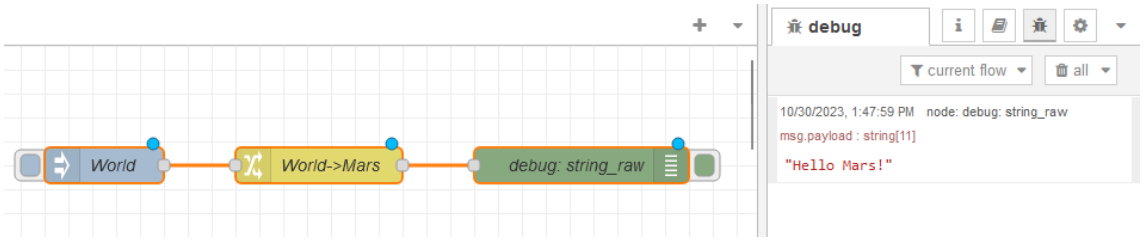

Figure 2.2: Change Message

2.3 Exercise 3: HTTP Request

In this exercise we will use the `http request` node to get the current earthquake data from the `https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/significant_month.csv`. It was decided to print the raw data to the debug console. Also the `change` node was used to change payload where the `mag` data was greater than 5.3 to `Panic!`.

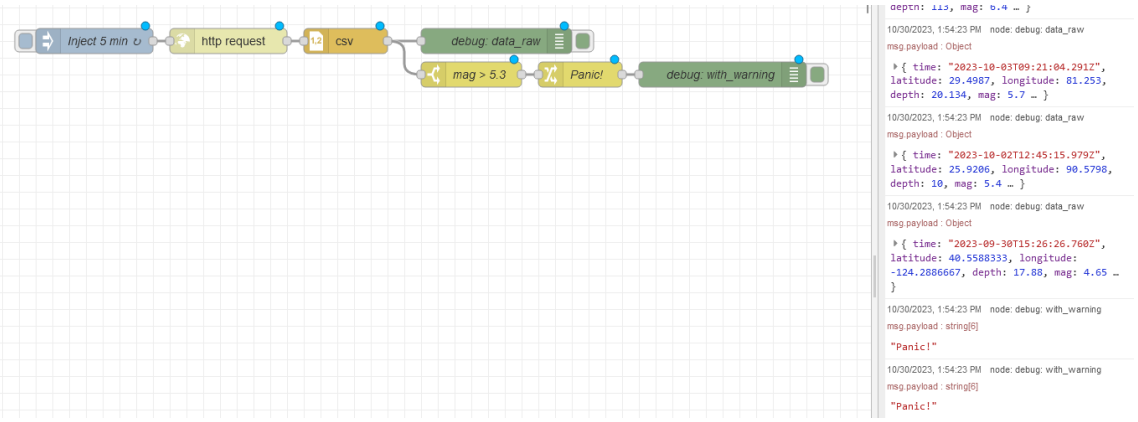

Figure 2.3: HTTP Request

2.4 Exercise 4: Dashboards

In this exercise we will use the `dashboard` node to create a simple dashboard which will show the current time in several different formats.

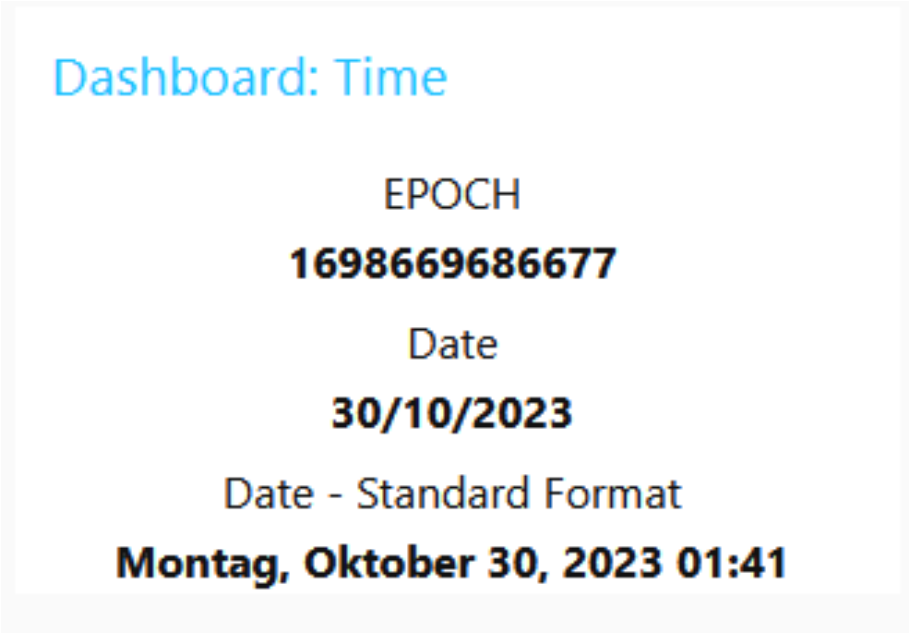

Figure 2.4: Dashboard

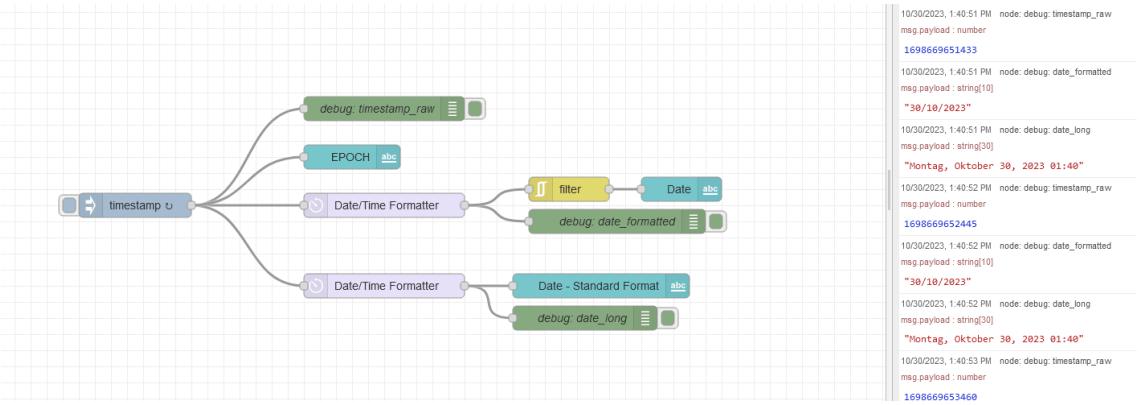

Figure 2.5: Nodes

To display the date with written day name and month some time were spent to find the correct formatting of `dddd, MMMM DD, YYYY hh:mm` to display the timestamp like wanted.

2.5 Exercise 5: LED Control

2.6 Exercise 6: Temperature

In this exercise we will use the gauge node to display the current temperature of the Arduino board. The temperature is measured using the internal temperature sensor of the Arduino. It was not a requirement but i tried to reuse the code from the first exercise with no changes leading to expanded node red flow.

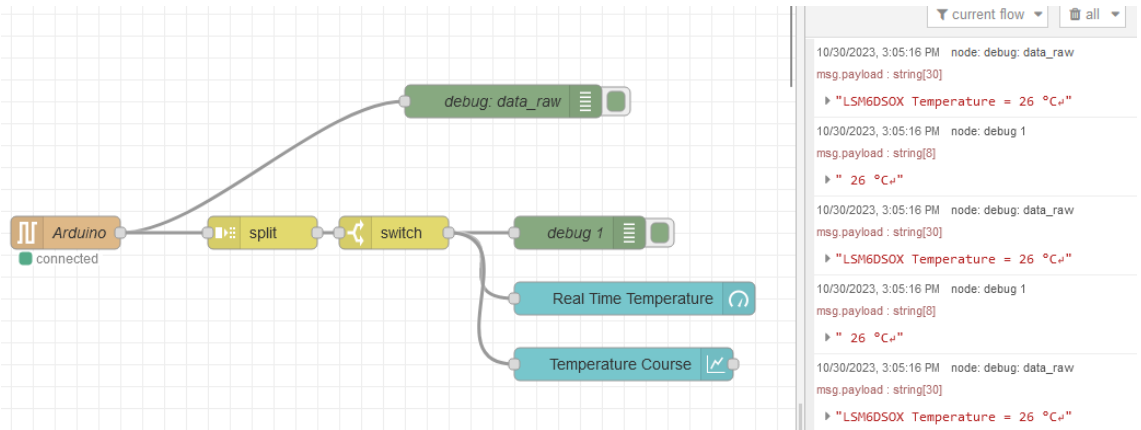

Figure 2.6: Temperature

Like seen in the figure above, the Arduino board prints the current temperature to the serial port like in the code from the first exercise. To process this data the `split` node was used to split the incoming data stream and a `switch` node which will compare incoming data if it `contains °C` and only forward this data.

The resulting dashboard can be seen in the figure below.

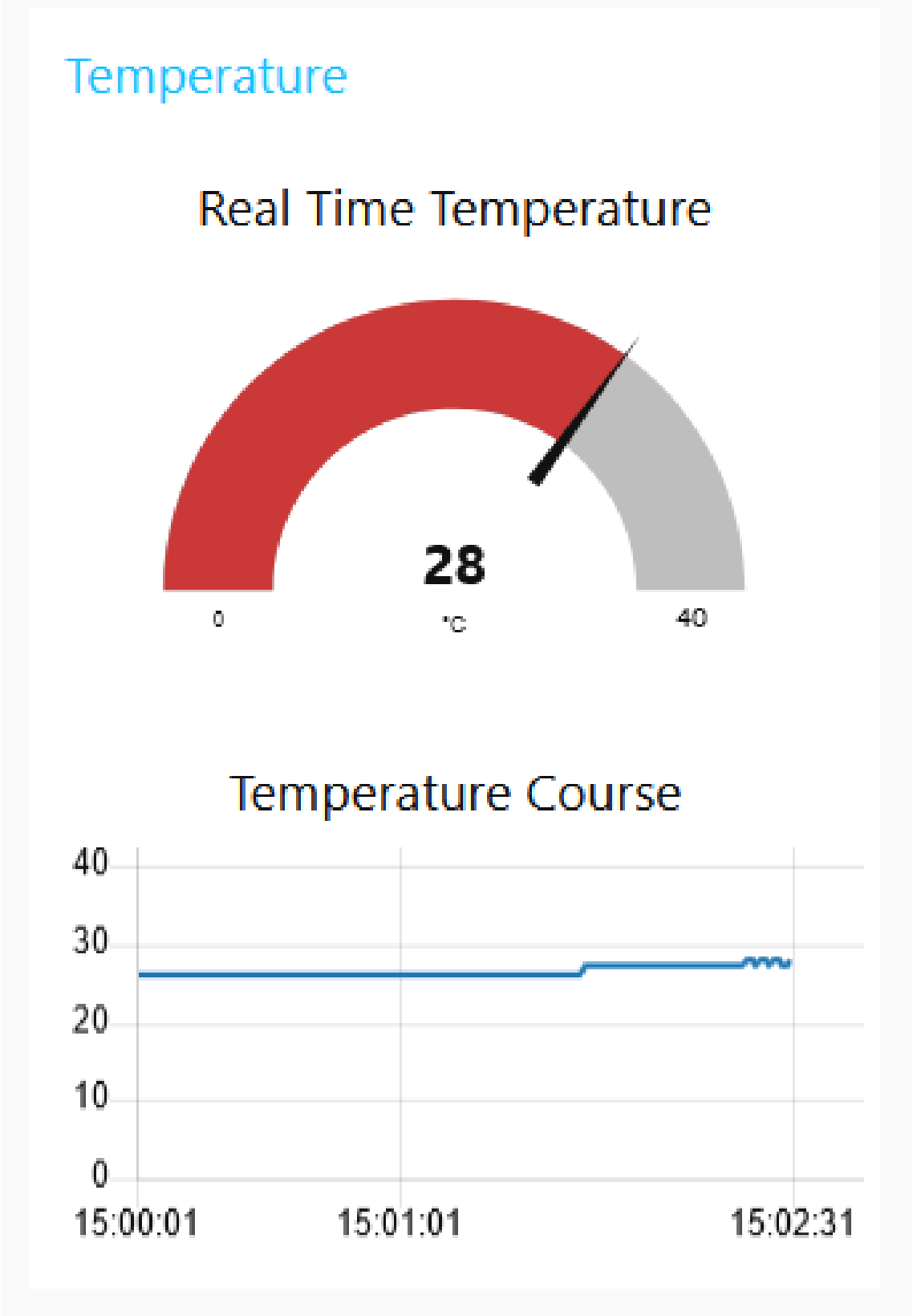

Figure 2.7: Temperature Dashboard

2.7 Exercise 7: MQTT

In this exercise we will use the `mqtt` node to send the current temperature to a MQTT broker (hosted on `https://www.hivemq.com/try-out/`) and display it on a dashboard on `https://www.datacake.de/`.

Most effort was spent on this exercise cause it was not clearly described (or at least for me) how important the `topic` field is. This field has to match either in Node-RED config as well as on the datacake settings. Even after figure this out the data was not processed right so i created a second flow which send the data to the broker where it will be returned as `raw_data`.

Also it has to be noted that this time the code of the Arduino was changed to send the temperature in a json styled format to the serial port. As most of the code was not changed only the relevant part is shown below.

```
1   ...
2   // Offset of ~30%
3   float temperature_normalized = temperature_deg / 1.3;
4
5   // Create a JSON-formatted string
6   String jsonString = "{\"temp\": " + String(temperature_normalized) + "}";
7
8   // Send JSON-formatted string over serial
9   Serial.println(jsonString);
10  ...
```

This data will then rooted from the serial port to the `mqtt` node which will send it to the broker. Notice that the data will be send to two different topics. One topic is used to send data to an raw channel and the other one is used to send data to a formatted channel (see figure below).

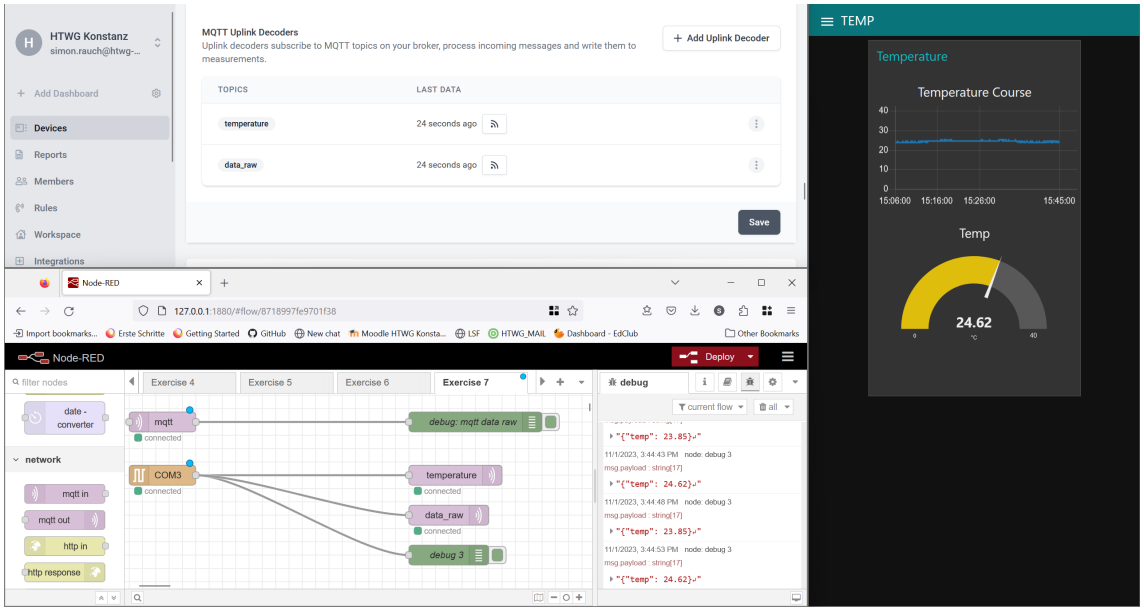

Figure 2.8: MQTT

I spent some more time cause i does not why but even if the same data was send to both topics only the raw data was seen on datacake even if both Uplink decoder were the same. To waste no more time the decoder from the temperature channel was moved to the raw channel which worked fine and data was displayed on the dashboard. Following you will see the java script to parse the data from the raw channel to the temperature field as well as the dashboard.

```
function Decoder(topic, payload) {
  // Transform incoming payload to JSON
  payload = JSON.parse(payload);

  // Extract Temperature from payload, do calculation
  var temperature = payload.temp;

  // Forward Data to Datacake Device API using Serial, Field-Identifier
  return [
      {
          device: "e8aec33d-a641-4d25-a235-786fd1291a7b", // Serial Number or Device ID
          field: "TEMPERATURE",
          value: temperature
      },
  ];
}
```

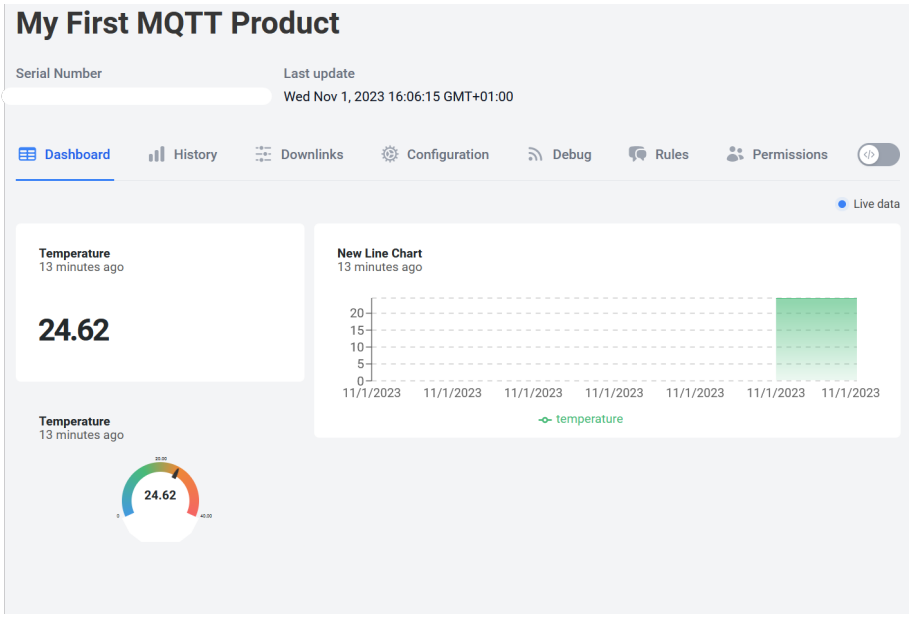

Figure 2.9: MQTT Dashboard

At the end of this exercise a downlink was setup to send a message to the Arduino board. This was also a little bit frustrating but the config as well as the result can be seen below.

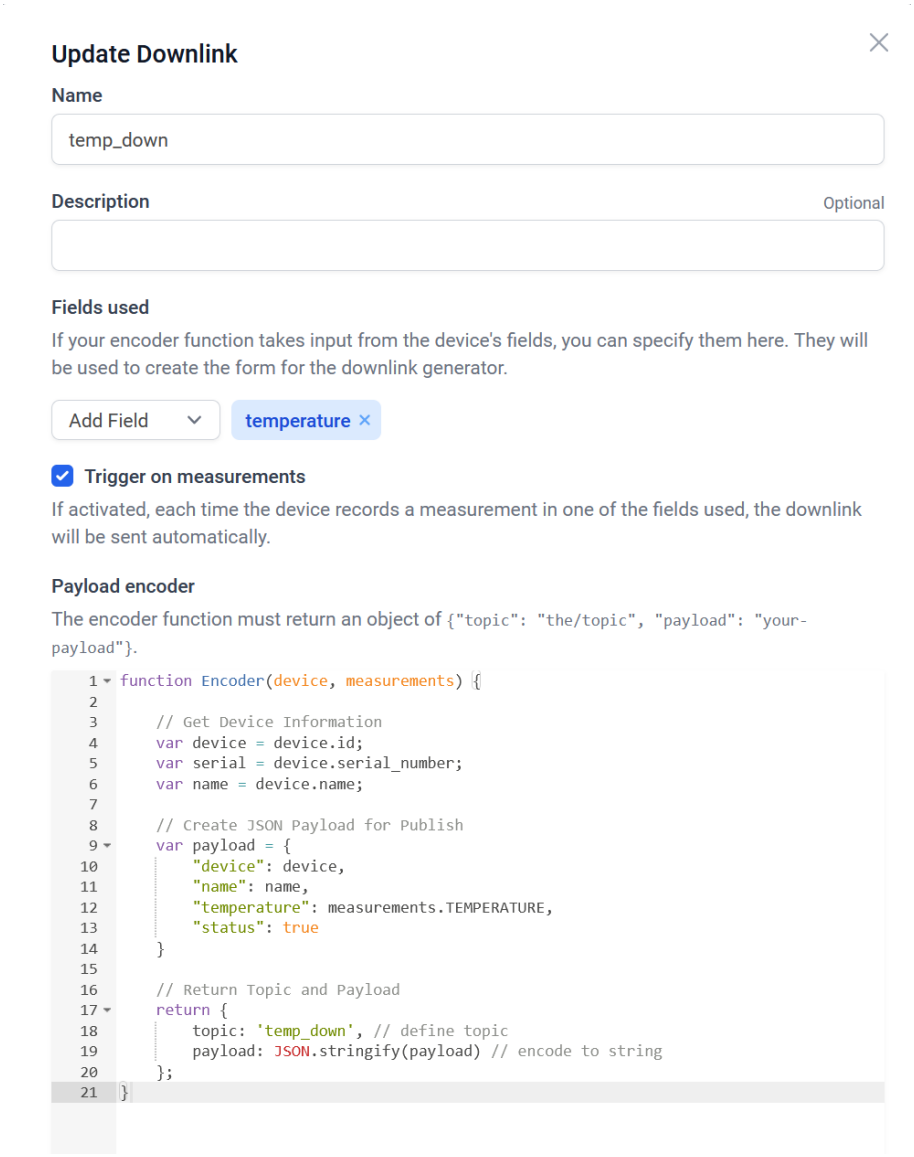

Figure 2.10: MQTT Downlink

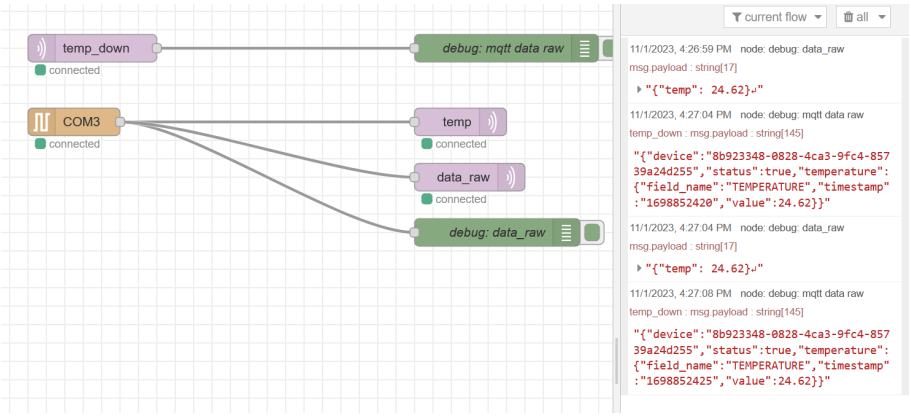

Figure 2.11: MQTT Downlink Result