



Ubiquitous Computing WS 24

Exercises Documentation



Contents

1	Exercise 1: Introduction	1
1.1	Exercise 1: Intro	1
1.2	Exercise 1.1; Internal RGB	1
1.3	Exercise 1.2: Temperature Sensor	2
1.4	Exercise 1.3: Microphone	4
1.5	Exercise 1.4: Posture Detector. Accelerometer	5
2	Exercise 2: Node-RED	7
2.1	Exercise 2.1; Hello World	7
2.2	Exercise 2.2: Change Message	7
2.3	Exercise 2.3: HTTP Request	8
2.4	Exercise 2.4: Dashboards	8
2.5	Exercise 2.5: LED Control	9
2.6	Exercise 2.6: Temperature	10
2.7	Exercise 2.7: MQTT	12
2.8	Exercise 2.8: Cat door	15
3	Exercise 3: MicroPython	20
3.1	Exercise 3.1: Creating a Cumulocity IoT Student Account	20
3.2	Exercise 3.2 Setup	20
3.2.1	Problem with <code>network</code> module	22
3.2.2	Flashing firmware manually	23
3.2.3	Updating Arduino Firmware	24
3.2.4	Using WiFi with <code>micropython</code>	25
3.3	Exercise 3.3: Sending data from the Arduino to Cumulocity IoT	25
4	Exercise 4: Machine Learning	27
4.1	Exercise 4: Machine Learning	27
4.1.1	Setup	27
4.1.2	Documentation	27

5 Exercise 5: Cumulocity	28
5.1 Exercise 5: Cumulocity	28
5.1.1 Task 1: Dashboards	28
5.1.2 Task 2: Smart Rules	29
5.1.3 Task 3: Analytics Builder	30
References	30

List of Figures

1.1	Temperature Sensor between 20 and 36 degrees Celsius	3
1.2	Temperature Sensor under 20 degrees Celsius	3
1.3	Temperature Sensor above 36 degrees Celsius	4
1.4	Sound detected, LED on	4
1.5	Sound detected, LED off	5
1.6	Posture Detector	6
2.1	Hello World	7
2.2	Change Message	7
2.3	HTTP Request	8
2.4	Dashboard	8
2.5	Nodes	9
2.6	LED Control	10
2.7	Temperature	11
2.8	Temperature Dashboard	11
2.9	MQTT	13
2.10	MQTT Dashboard	14
2.11	MQTT Downlink	15
2.12	MQTT Downlink Result	15
2.13	Cat Door Note-RED Flow	18
2.14	Cat Door Datacake	19
3.1	Thonny - Configure Interpreter	20
3.2	Thonny - Select Interpreter	21
3.3	Thonny - Install MicroPython	21
3.4	Error: Network Modul	23
3.5	Firmware Versions	23
3.6	Bootloader-Mode	24
3.7	Bootloader-Mode	24
3.8	Update Firmware	25
3.9	WiFi Connected	25
3.10	Send Data	26

5.1	Cumulocity Dashboard	28
5.2	Cumulocity Smart Rule	29
5.3	Cumulocity Alarm	29
5.4	Cumulocity Analytics Builder	30
5.5	Cumulocity Alarm	30

List of Tables

Acronyms

1 Exercise 1: Introduction

1.1 Exercise 1: 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 1.1: 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(LED_R, OUTPUT);
7         pinMode(LED_G, OUTPUT);
8         pinMode(LED_B, 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(LED_R, HIGH);      // Red
23        digitalWrite(LED_B, LOW);      // Blue
24        digitalWrite(LED_G, LOW);      // Green
25    }
26
27    void blink_blue(){
28        Serial.print("Blue\n");
29        digitalWrite(LED_R, LOW);      // Red
30        digitalWrite(LED_B, HIGH);      // Blue
31        digitalWrite(LED_G, LOW);      // Green
32    }
33
34    void blink_green(){
35        Serial.print("Green\n");
36        digitalWrite(LED_R, LOW);      // Red
37        digitalWrite(LED_B, LOW);      // Blue
38        digitalWrite(LED_G, HIGH);      // Green
39    }

```

1.3 Exercise 1.2: 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_Ubiqutous_Computing/blob/main/exercise/exercise2_temperature/exercise2_temperature.ino.

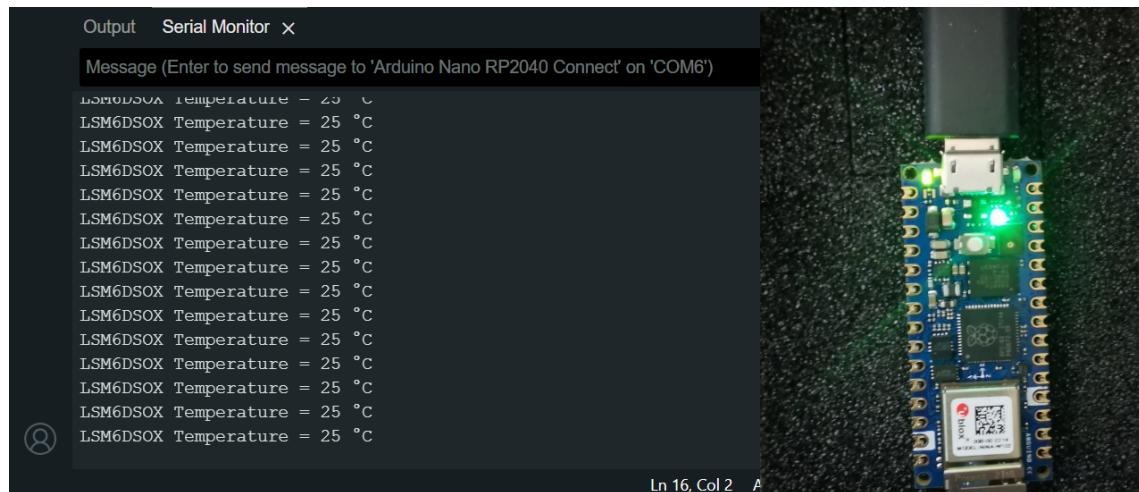


Figure 1.1: Temperature Sensor between 20 and 36 degrees Celsius

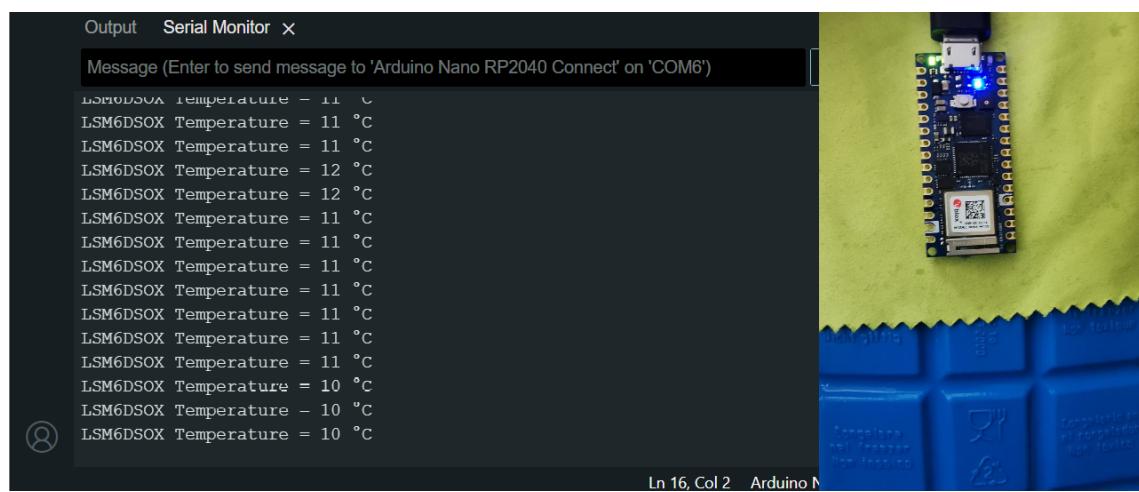


Figure 1.2: Temperature Sensor under 20 degrees Celsius



Figure 1.3: Temperature Sensor above 36 degrees Celsius

1.4 Exercise 1.3: 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).

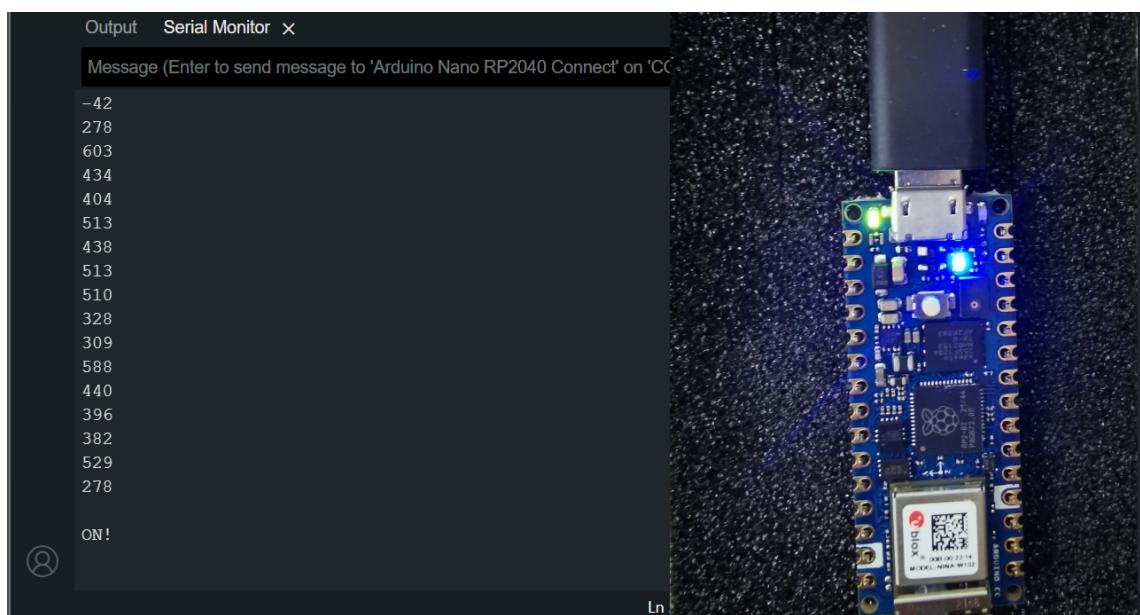


Figure 1.4: Sound detected, LED on

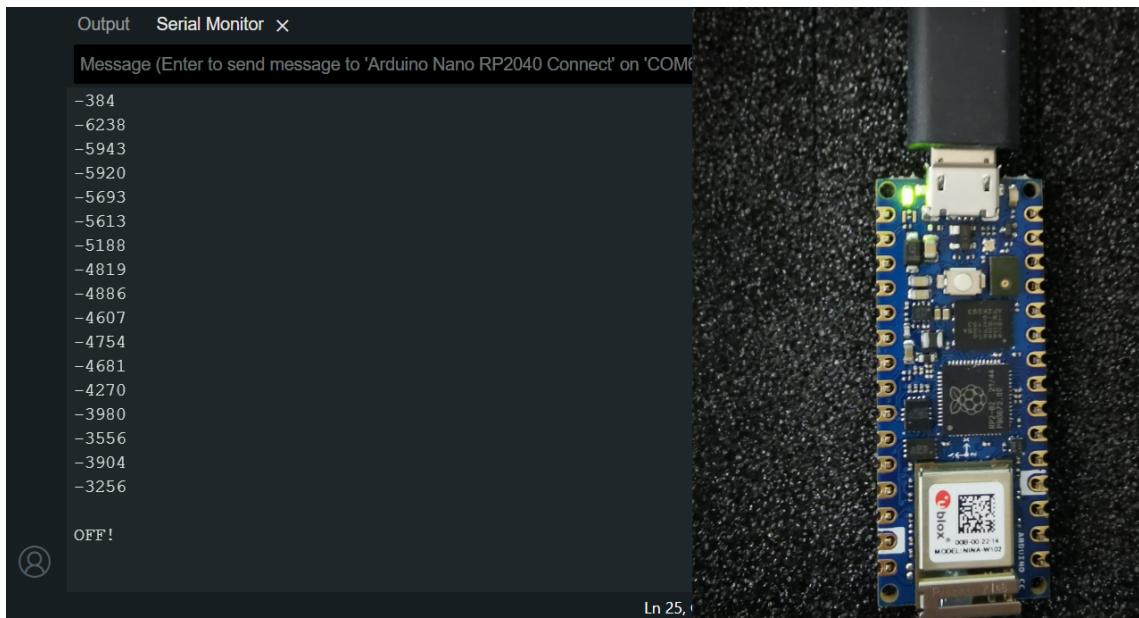


Figure 1.5: Sound detected, LED off

1.5 Exercise 1.4: 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.

Output Serial Monitor X

Message (Enter to send message to 'Arduino Nano RP2040 Connect' on 'COM6')

```
Orientation: [179.99 Yaw] [0.03 Pitch] [-0.20 Roll]
Orientation: [179.99 Yaw] [0.05 Pitch] [-0.32 Roll]
Orientation: [179.98 Yaw] [0.07 Pitch] [-0.43 Roll]
Orientation: [179.98 Yaw] [0.09 Pitch] [-0.54 Roll]
Orientation: [179.98 Yaw] [0.10 Pitch] [-0.63 Roll]
Orientation: [179.54 Yaw] [0.97 Pitch] [-1.11 Roll]
Orientation: [179.81 Yaw] [1.38 Pitch] [-1.22 Roll]
Orientation: [179.15 Yaw] [4.17 Pitch] [-4.26 Roll]
Orientation: [179.43 Yaw] [4.05 Pitch] [-4.14 Roll]
Orientation: [179.42 Yaw] [3.99 Pitch] [-3.92 Roll]
Orientation: [179.00 Yaw] [5.49 Pitch] [-3.93 Roll]
Orientation: [179.57 Yaw] [5.88 Pitch] [-4.67 Roll]
Orientation: [179.62 Yaw] [5.74 Pitch] [-4.80 Roll]
Orientation: [179.62 Yaw] [5.66 Pitch] [-4.92 Roll]
Orientation: [179.64 Yaw] [5.78 Pitch] [-4.93 Roll]
```

Figure 1.6: Posture Detector

2 Exercise 2: 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_Ubiqutous_Computing in the exercise section.

2.1 Exercise 2.1; Hello World

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

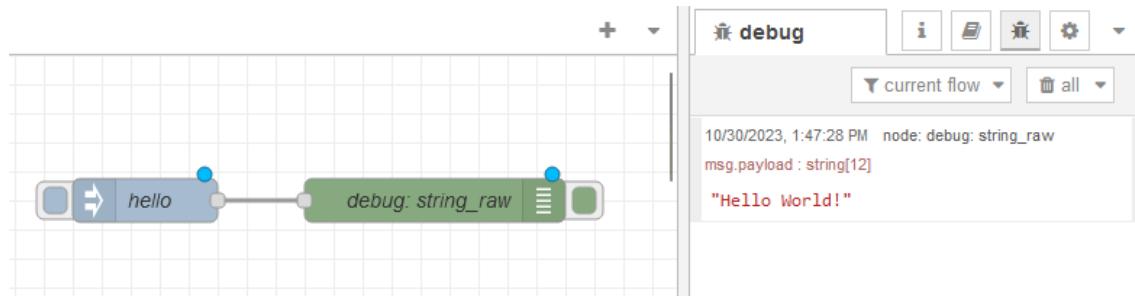


Figure 2.1: Hello World

2.2 Exercise 2.2: Change Message

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

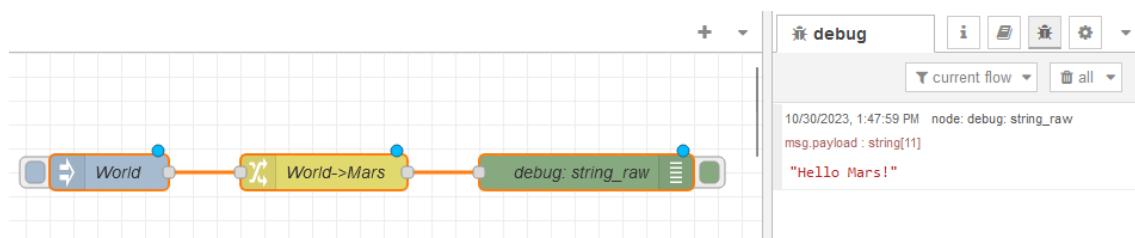


Figure 2.2: Change Message

2.3 Exercise 2.3: 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!`.

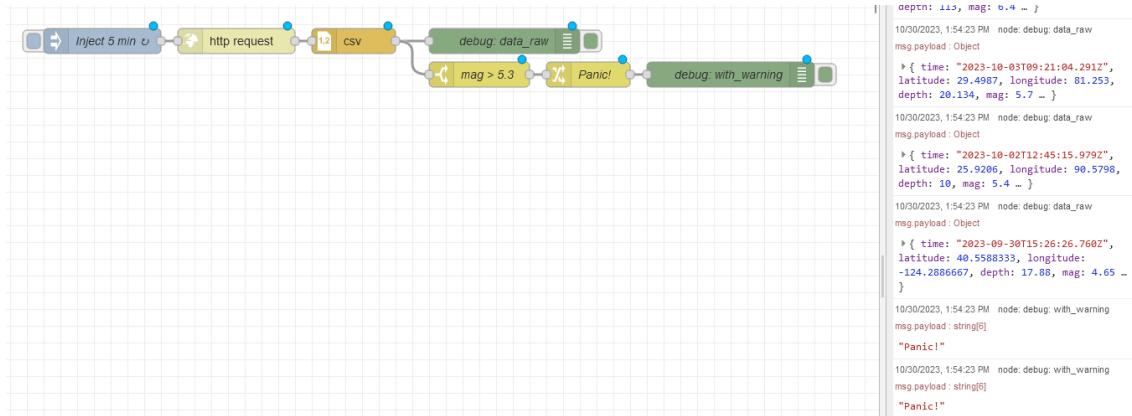


Figure 2.3: HTTP Request

2.4 Exercise 2.4: 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

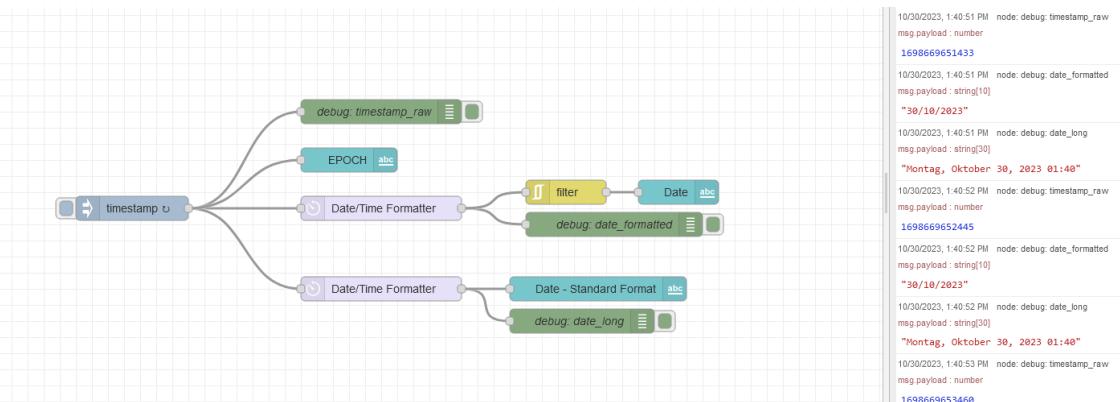


Figure 2.5: Nodes

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 2.5: LED Control

In the fifth exercise we will use the led node to control the LED on the Arduino board as well as a hardware setup like seen [here](<https://docs.arduino.cc/built-in-examples/digital/Button>).

Notable is that instead of using the 5V pin to power the LED, the 3.3V pin was used cause even when operating the circuit with an pull down resistor the 5V signal was bouncing. Instead working with the 3.3V signal was like expected.

The code below running on the Arduino board is as simple as expected.

```

1 #include <WiFiNINA.h>
2
3 const int button_pin = 4;
4
5 void setup() {
6     // Initialize LEDs
7     pinMode(LED_R, OUTPUT);
8
9     // initialize the pushbutton pin
10    pinMode(button_pin, INPUT);
11
12    Serial.begin(9600);
13}
14
15 void loop() {
16
17    static int debounce = 0;
18    static String msg = "";

```

```

19
20 // put main code here, to run repeatedly:
21
22 if (Serial.available() > 0) {
23     msg = Serial.readString();
24 }
25
26 if (digitalRead(button_pin) == HIGH ||
27     msg.compareTo("1") == false){
28     digitalWrite(LED_R, HIGH);           // Red LED on
29     Serial.println(1);                  // Writeback status for dashboard
30 } else {
31     digitalWrite(LED_R, LOW);          // Red LED off
32     Serial.println(0);
33 }
34
35 delay(100);
36 }

```

The figure below shows the Node-RED flow with the hardware button on the breadboard pushed (notable cause software button is not pushed but LED is on) and the dashboard showing the correct status.

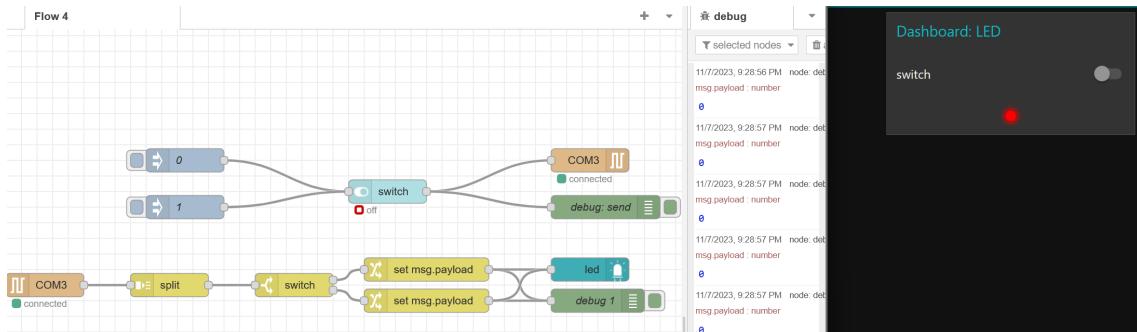


Figure 2.6: LED Control

2.6 Exercise 2.6: 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.

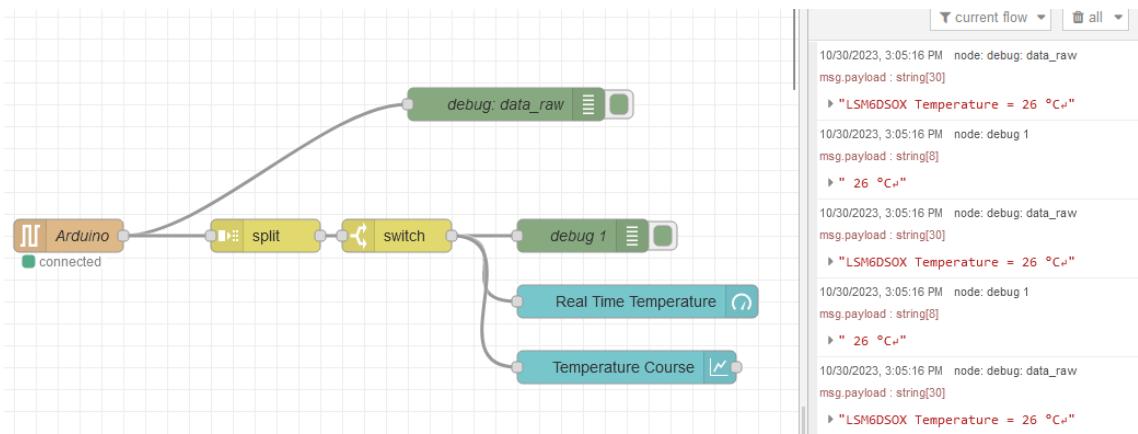


Figure 2.7: Temperature

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.

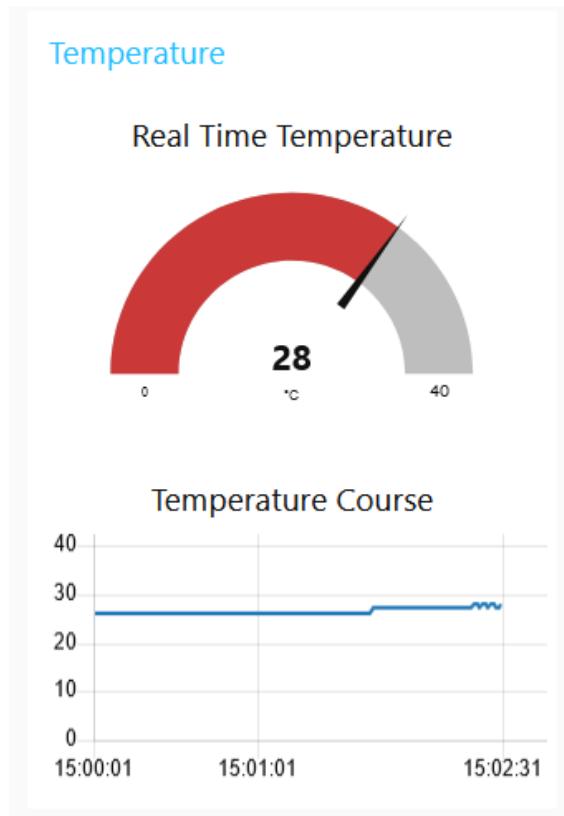


Figure 2.8: Temperature Dashboard

2.7 Exercise 2.7: 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).

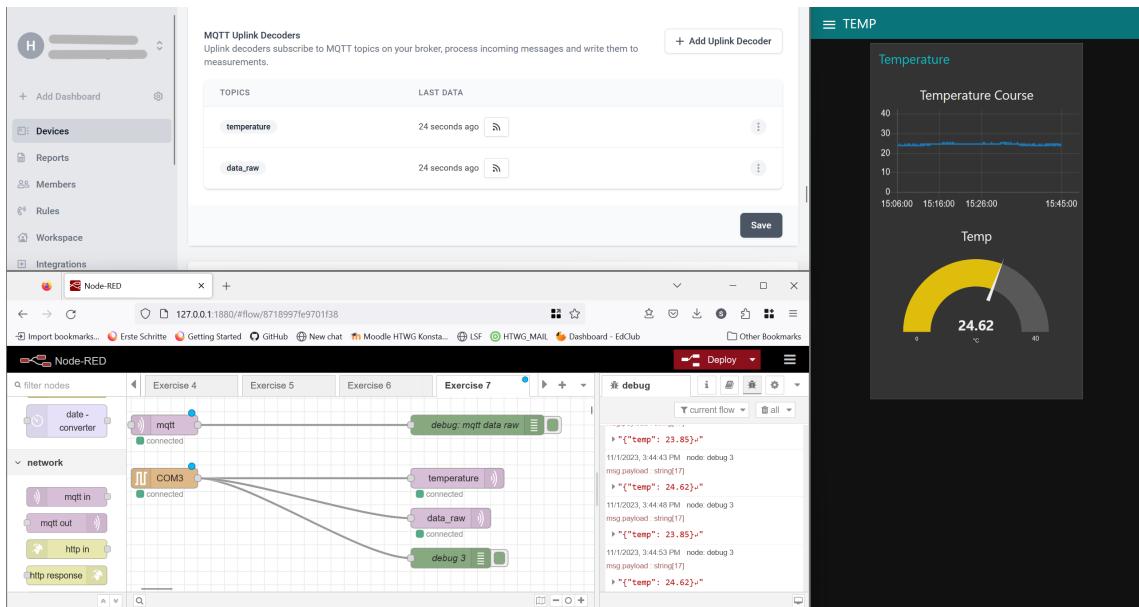


Figure 2.9: MQTT

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.

```

1  function Decoder(topic, payload) {
2      // Transform incoming payload to JSON
3      payload = JSON.parse(payload);
4
5      // Extract Temperature from payload, do calculation
6      var temperature = payload.temp;
7
8      // Forward Data to Datacake Device API using Serial, Field-Identifier
9      return [
10         {
11             device: "e8aec33d-a641-4d25-a235-786fd1291a7b", // Serial Number or Device ID
12             field: "TEMPERATURE",
13             value: temperature
14         },
15     ];
16 }
```

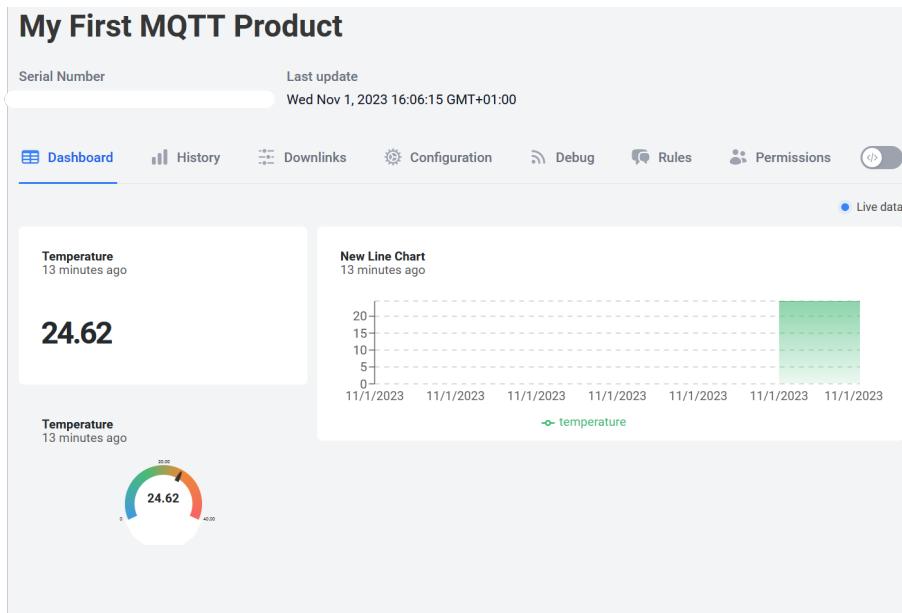


Figure 2.10: MQTT Dashboard

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.

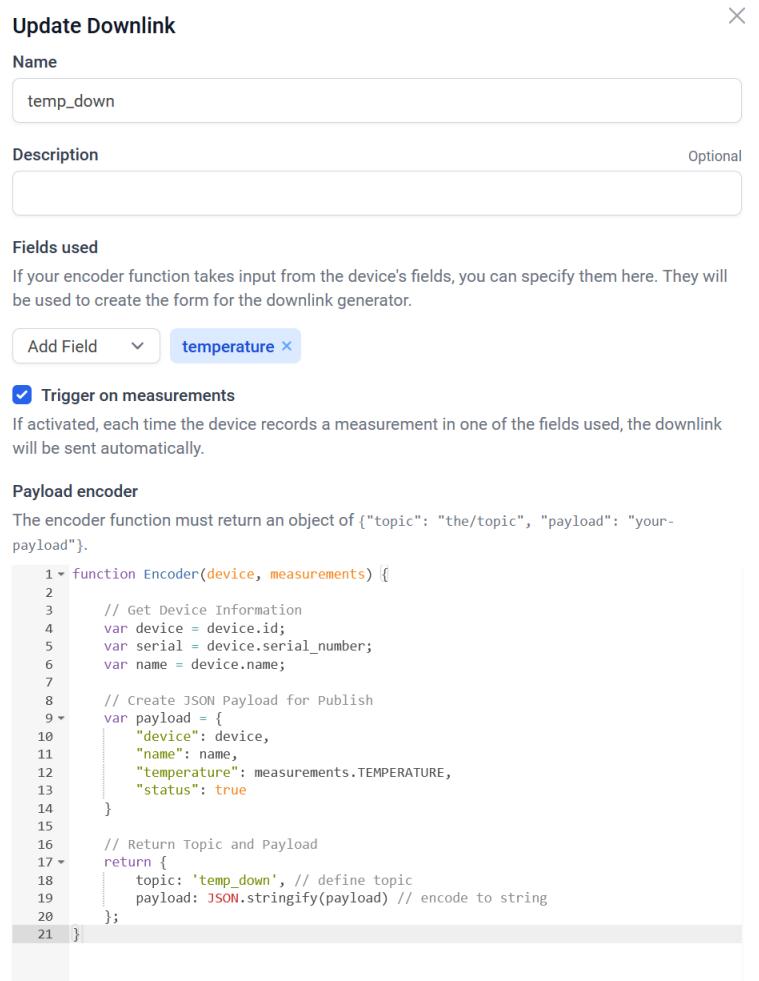


Figure 2.11: MQTT Downlink

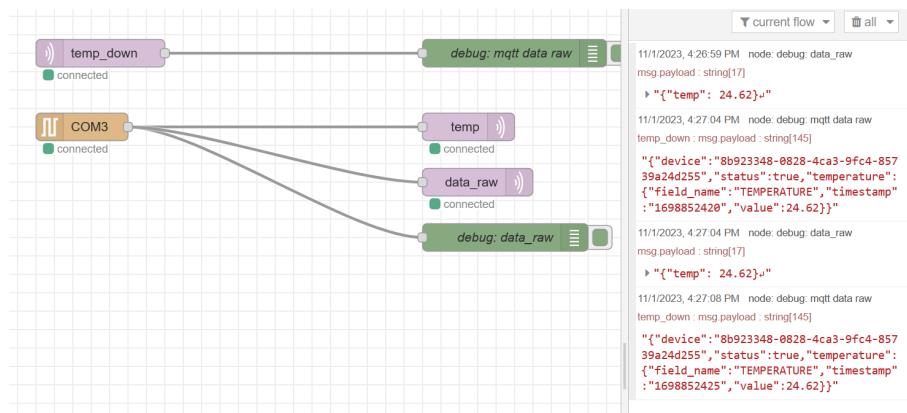


Figure 2.12: MQTT Downlink Result

2.8 Exercise 2.8: Cat door

In the last exercise we will use the `mqtt` as well as the arduinos gyroscope to detect if the cat door is open or closed. The gyroscope is used to detect the angle of the door as well as

the direction it is opened (detecting if cat is indoor or outdoor).

The code running on the arduino board is shown below.

```
1 #include <WiFiNINA.h>
2 #include <Arduino_LSM6DSOX.h>
3 #include <MadgwickAHRS.h>
4
5 Madgwick filter;
6
7 void setup() {
8     // Initialize serial communication and wait for port to open:
9     Serial.begin(9600);
10    while (!Serial) {
11        ; // wait for serial port to connect. Needed for native USB port only
12    }
13
14    // Initialize the LSM6DSOX
15    if (!IMU.begin()) {
16        Serial.println("Failed to initialize IMU!");
17        while (1);
18    }
19
20    filter.begin(104.00);
21
22    // Initialize LEDs
23    pinMode(LED_R, OUTPUT);
24    pinMode(LED_G, OUTPUT);
25    pinMode(LED_B, OUTPUT);
26
27    digitalWrite(LED_R, LOW);      // Red
28 }
29
30 void loop() {
31     float ax, ay, az;
32     float gx, gy, gz;
33     static int counter = 0;
34     static bool state;
35     static float temp = 0;
36
37     // check if data is available every loop
38     if(IMU.accelerationAvailable() &&
```

```

39     IMU.gyroscopeAvailable()) {
40
41     IMU.readAcceleration(ax, ay, az);
42     IMU.readGyroscope(gx, gy, gz);
43
44     // Update the filter with the latest data
45     filter.updateIMU(gx, gy, gz, ax, ay, az);
46
47     // Check the filter's roll for door movement
48     if (filter.getRoll() > 5){
49         state = true;
50         digitalWrite(LED_R, HIGH); // Turn on LED if door is opened
51     } else if (filter.getRoll() < -5) {
52         state = false;
53         digitalWrite(LED_R, HIGH); // Turn on LED if door is opened
54     } else {
55         digitalWrite(LED_R, LOW); // Turn off LED if door is closed
56     }
57 }
58
59 if(!(counter % 100)){
60     temp = get_temp();
61     send_data(temp, state);
62 }
63
64 counter > 100 ? counter = 1 : counter++;
65 delay(100);
66 }
67
68 float get_temp() {
69     // Check if temperature is available
70     if (IMU.temperatureAvailable())
71     {
72         int temperature_deg = 0;
73         IMU.readTemperature(temperature_deg);
74
75         // Offset of ~30%
76         float temperature_normalized = temperature_deg / 1.3;
77
78         return temperature_normalized;
79     }
80 }
```

```

81
82 void send_data(float temp, bool in_out){
83     // Construct JSON-style string
84     String jsonString = "{\"temperature\": " + String(temp) + ", \"state\": \"" + (in_out
85     Serial.println(jsonString);
86 }

```

Note that the sensor is checked in every iteration but the data is only send every 100th iteration to reduce the amount of data send to the broker. Also the data is send in a json styled format to the serial port.

Like seen in the figure below the data is then send to the broker using NodeRED where it is collected and displayed from datacake.

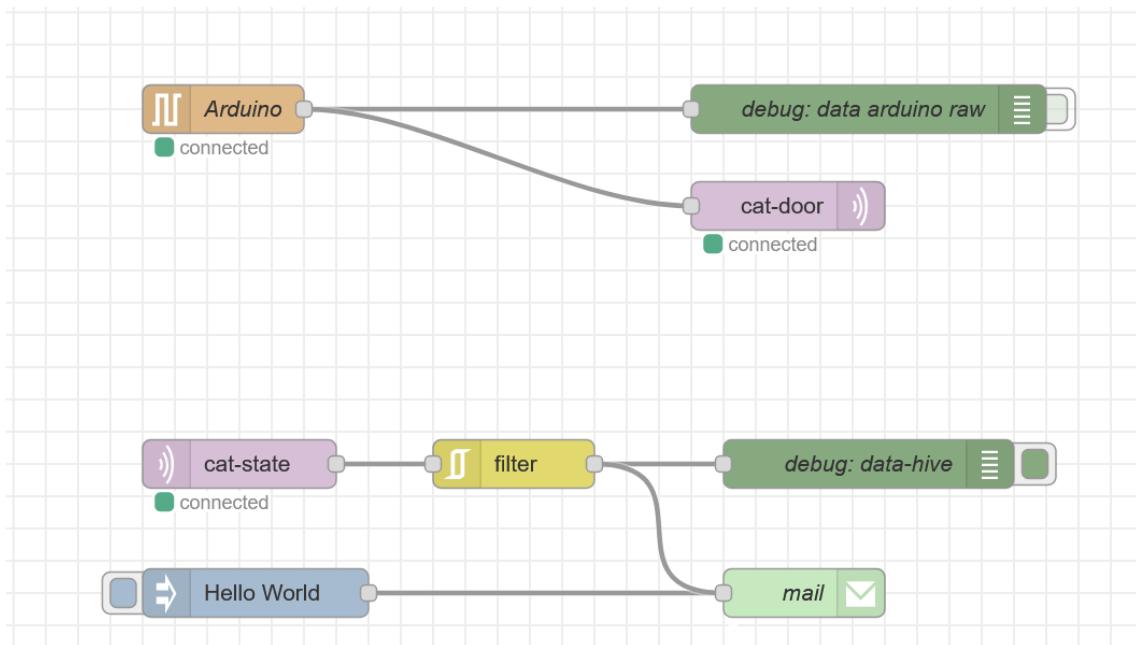


Figure 2.13: Cat Door Note-RED Flow

Also the data is send back from datacake to the NoteRED flow using the Downlink feature which was introduced in the last exercise. This will trigger a Mail node which will send an email to the configured email address. (Cause of privacy reasons the .json file is not included in the repository cause it contains the email address.)

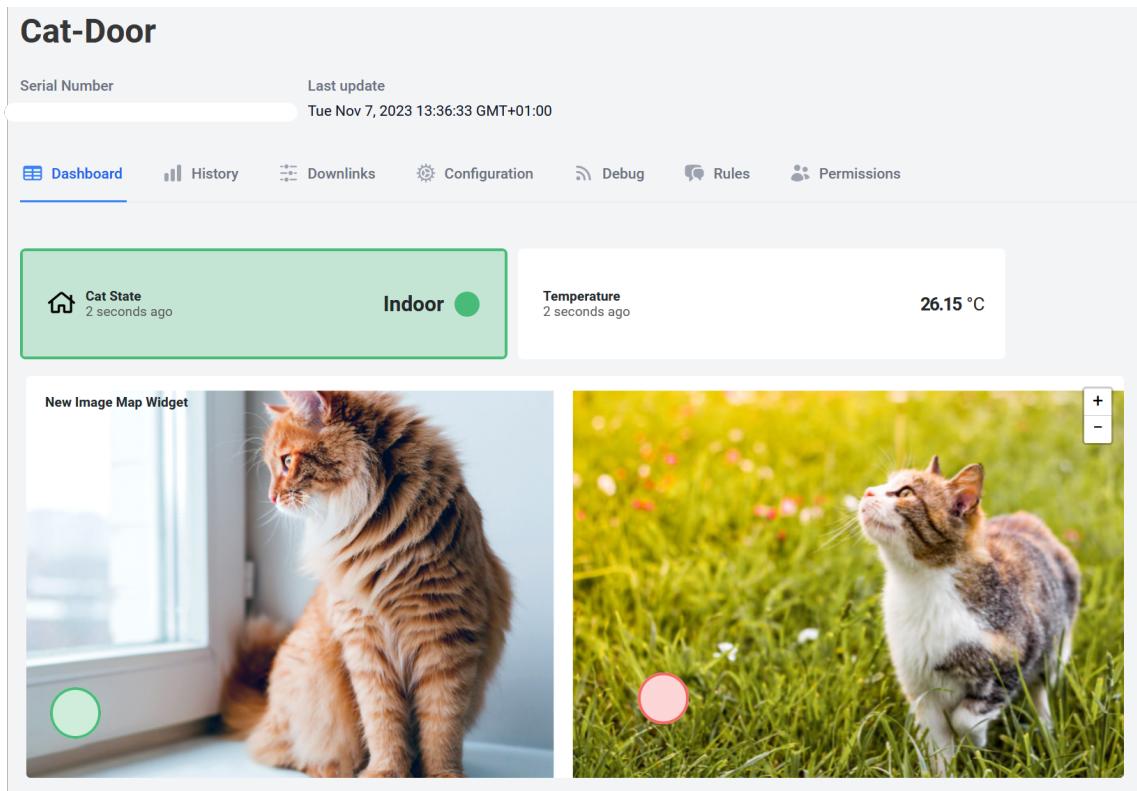


Figure 2.14: Cat Door Datacake

3 Exercise 3: MicroPython

3.1 Exercise 3.1: Creating a Cumulocity IoT Student Account

So the first was to create a Cumulocity IoT Student Account which might be straight forward but due to a outdated exercise description it was not that easy cause the link to the Cumulocity platform was deprecated. Anyway after finding the way to the Cumulocity IoT page the default dashboard was shown. At this point I am not completely sure if there is a need to do this exercise using a new platform or why it is not possible to use datacake which was used in the previous exercise.

3.2 Exercise 3.2 Setup

So setup for this exercise was a real pain an unfortunately the exercise description did not help at all. The first problem was to even get a working connection to the board via USB to install micropython. After installing Thonny the device were not shown like described further more the suggested Arduino Micropython Tools from Arduino which can be found *here* did not work at all. After some research I found a working solution with Thonny (see pictures below). It is highly recommended from my side to update this exercise description and give hints about setting device in Bootloader-Mode etc.

So the following steps were done to get a working connection to the board:

1. Download and install Thonny from [here](<https://thonny.org/>)
2. Open Thonny and connect the board via USB
3. Click on Local Python 3 in the upper right corner and select Configure Interpreter

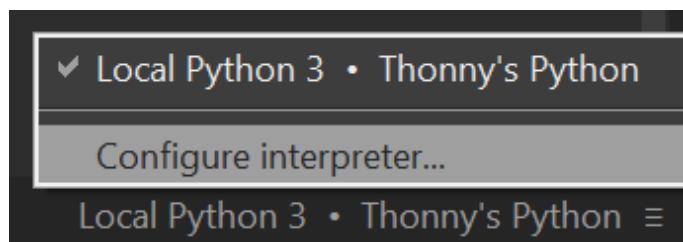


Figure 3.1: Thonny - Configure Interpreter

4. Select MicroPython(RP2040) and select the correct port

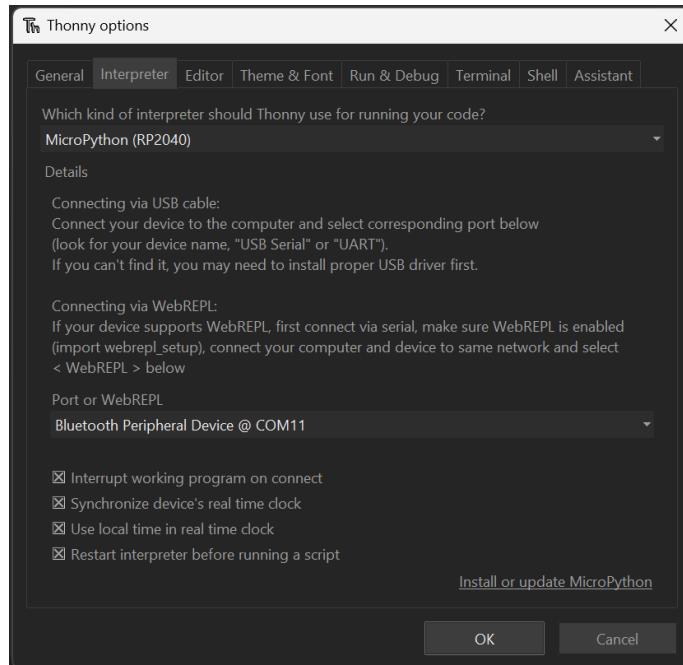


Figure 3.2: Thonny - Select Interpreter

5. Click on Install or update MicroPython

6. Select the target settings like shown in the picture below

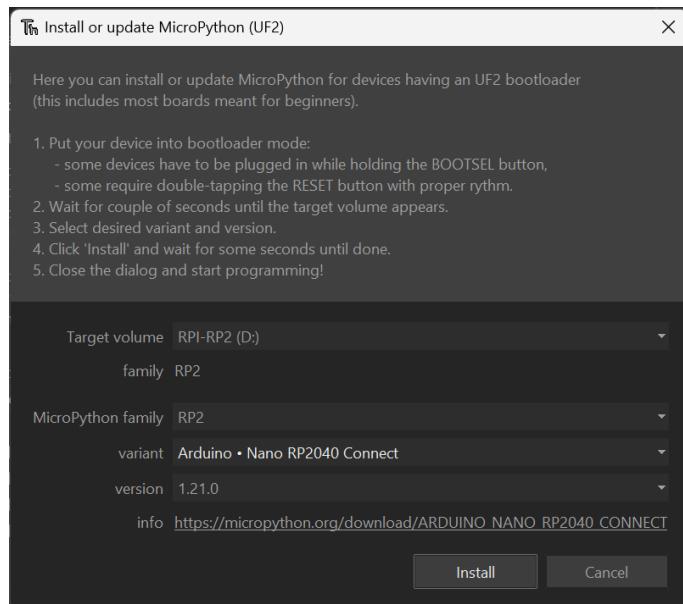


Figure 3.3: Thonny - Install MicroPython

7. NOTE: In some cases it is necessary to set the board in Bootloader-Mode by pressing Reset button twice

To test if the installation was successful I used the following code to blink the LED on the board which could be found [here](#): As this example is very huge I wrote a small script inspired by Arduino page [here](#). The code could be seen bellow, it just blinks the LED connected to PIN 25 on the board.

```
1 from machine import Pin
2 import time
3
4 myLED = Pin(25, Pin.OUT) #Nano RP2040 Connect
5 #myLED = Pin(10, Pin.OUT) #Nano 33 BLE / Nano 33 BLE Sense
6 #myLED = Pin(2, Pin.OUT) #Portenta H7
7
8
9 while True:
10     print("LED ON")
11     myLED.value(0)
12     time.sleep(1)
13     print("LED OFF")
14     myLED.value(1)
15     time.sleep(1)
```

3.2.1 Problem with network module

After installing micropython on the board I tried to connect to the WiFi using the following code which could be found [here](#):

```
1 import network
2
3 WIFI_NETWORK='YOUR_NETWORK_NAME'
4 WIFI_PASSWORD='YOUR_NETWORK_PASSWORD'
5
6 wlan = network.WLAN(network.STA_IF)
7 wlan.active(True)
8 wlan.connect(WIFI_NETWORK, WIFI_PASSWORD)
9
10 print()
11 print("Connected to ",WIFI_NETWORK)
```

This code is also part of the exercise but running this will lead to the following error:

```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Traceback (most recent call last):
  File "<stdin>", line 7, in <module>
OSError: Firmware version mismatch. Minimum supported firmware is v1.5.0 found v1.4.5
```

Figure 3.4: Error: Network Modul

After some research I found out that the network module is part of the `micropython` installation and this error is more or less a bug in the `micropython` installation. Research showed that there is an GitHub issue for this problem which could be found [here](#). which is still open and the workaround described there is not working for me. There were several attempts to get this running by flashing the firmware using several tools. It was flashed using Thonny and MicroPython Installer which is an Arduino tool which could be found [here](#). Also it was tested using OpenMV like described [here](#). Unfortunately none of these attempts were successful and the error still occurs. Even joining `micropythons` Discord channel showed that there is not even a question about this problem.

At the end I decided to flash firmware manual to the device like described [here](#). This site provides several firmware versions like shown in the image bellow:

Firmware

Releases

[v1.21.0 \(2023-10-05\) .uf2 / \[Release notes\] \(latest\)](#)
[v1.20.0 \(2023-04-26\) .uf2 / \[Release notes\]](#)
[v1.19.1 \(2022-06-18\) .uf2 / \[Release notes\]](#)
[v1.18 \(2022-01-17\) .uf2 / \[Release notes\]](#)

Preview builds

[v1.22.0-preview.164.gfce8d9fd5 \(2023-11-20\) .uf2](#)
[v1.22.0-preview.163.g57cce79a4 \(2023-11-20\) .uf2](#)
[v1.22.0-preview.162.g6117aa686 \(2023-11-17\) .uf2](#)
[v1.22.0-preview.152.g37c1c5fa4 \(2023-11-17\) .uf2](#)
(These are automatic builds of the development branch for the next release)

Figure 3.5: Firmware Versions

3.2.2 Flashing firmware manually

To flash the firmware manually the following steps were done:

1. Download the latest firmware version from [here](#)

- Set device in Bootloader-Mode by pressing Reset button twice. If the device is in Bootloader-Mode the device will open as a new drive like seen bellow:

Name	Date modified
System Volume Information	21.11.2023 14:58
INDEX.HTM	05.09.2008 17:20
INFO_UF2.TXT	05.09.2008 17:20

Figure 3.6: Bootloader-Mode

- Note:** If the device is open as a new drive as shown bellow it is **NOT** in Bootloader-Mode

Name	Date modified
System Volume Information	21.11.2023 14:42

Figure 3.7: Bootloader-Mode

- Copy the downloaded firmware to the devices root folder
- After copying the firmware the device will reboot and the firmware will be flashed
- After flashing the device will open as a new drive again like seen in figure 3.7
- When opening the device in e.g. Thonny the current firmware version will be shown in the shell

All released firmware were tested as well as one preview version. When flashing **v1.18** the device will not show up as a drive and there is still a warning about firmware mismatching but no error when running. This was the only version which did not show the error.

3.2.3 Updating Arduino Firmware

After more testing i figured out that there are more errors when using micropython on an Arudino board which has a firmware version lower than **1.4.8** installed (so unfortunately this is no official information but was posted on Arduino forum and after struggling with it for a few hours, I can confirm this).

So updating the firmware would be a easy one cause it can be done using the Arduino IDE. Guess what, this was not working as well. Cause there was micropython flashed to the board the Arduino IDE was not able to upload scripts or update the firmware. Don't know exactly what i did but after pressing BOOTLOADER button and flashing several micropython versions once it was possible to upload sketches from Arduino IDE again. After that it was possible to update the firmware using the Arduino IDE which can be seen in the picture bellow:



Figure 3.8: Update Firmware

3.2.4 Using WiFi with micropython

After this long journey (in which I unfortunately did not fight a dragon in a mountain full of gold and thus helped a dwarf tribe to freedom). It was possible to connect to the WiFi using the code shown above resulting in the following output:

```
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Try to connected to network [Smokeys-Hotspot] ...
Connnected to network [config: 192.168.43.87 ]
```

Figure 3.9: WiFi Connected

3.3 Exercise 3.3: Sending data from the Arduino to Cumulocity IoT

The last exercise was to send data from the Arduino to Cumulocity IoT. This was done using the code shown in `main.py` of the repository which could be found [here](#). Cause the code is very huge I will not post it here but it is well documented and should be self-explanatory.

For this exercise it would also be required to access the sensors on the board. Access the boards sensors was done using the `LSM6DSOX` library and the code provided [here](#). **Note:** The SDA and SCL pins are different on the Arduino board and has to be changed like shown [here](#). To import the library i wrote a small script `import_lsm6dsox.py` which imports the library using `mip`.

After setting up everything it was possible to send data to Cumulocity IoT resulting in the following output:

Devices									No filters	1 of 1 items		Configure columns		Reload	<input type="text" value="Search..."/>	
Status	Name	Model	Serial number	Group	Registration date	System ID	IMEI	Alarms								
	Smokeys_nano				23 Nov 2023, 21:40:06	19496										

Figure 3.10: Send Data

4 Exercise 4: Machine Learning

4.1 Exercise 4: Machine Learning

Exercise number four is about machine learning. Therefore we will use *Python* and *Tensorflow* to analyze and work on Smarhome data captured in a *.csv* file. As this is done in an *Jupyter Notebook* the task could be done local or in *Google Colab*.

4.1.1 Setup

As i prefer to work on my local machine and the *Jupyter Notebook* file which was handed out worked with *Google Colab* i had to do some changes to the code. These sections are marked with *Requirements WSL* in the *Jupyter Notebook* file which can be found in the *exercise/sheet 4* folder.

4.1.2 Documentation

As the exercises were done in the *Jupyter Notebook* file, the documentation of the two exercises is also done there as markdown text and code comments. Please see the *Jupyter Notebook* file in the *exercise/sheet 4* folder at https://github.com/Smokey95/AIN_Ubiquitous_Computing.

5 Exercise 5: Cumulocity

5.1 Exercise 5: Cumulocity

Exercise number five is about *Cumulocity*. There are three tasks defined in the sheet about how to use some features of *Cumulocity* portal.

5.1.1 Task 1: Dashboards

The first task is about creating a dashboard in *Cumulocity* portal. Therefore it was required to connect the smartphone to *Cumulocity* and install the *Cumulocity* app on the smartphone. After this the smartphone could be used to collect data like the battery level, the current location gyroscope and accelerometer data. This data was then visualized in a dashboard in the *Cumulocity* portal.

On the picture bellow you can see the dashboard as well as my location during the time i was working on this exercise.

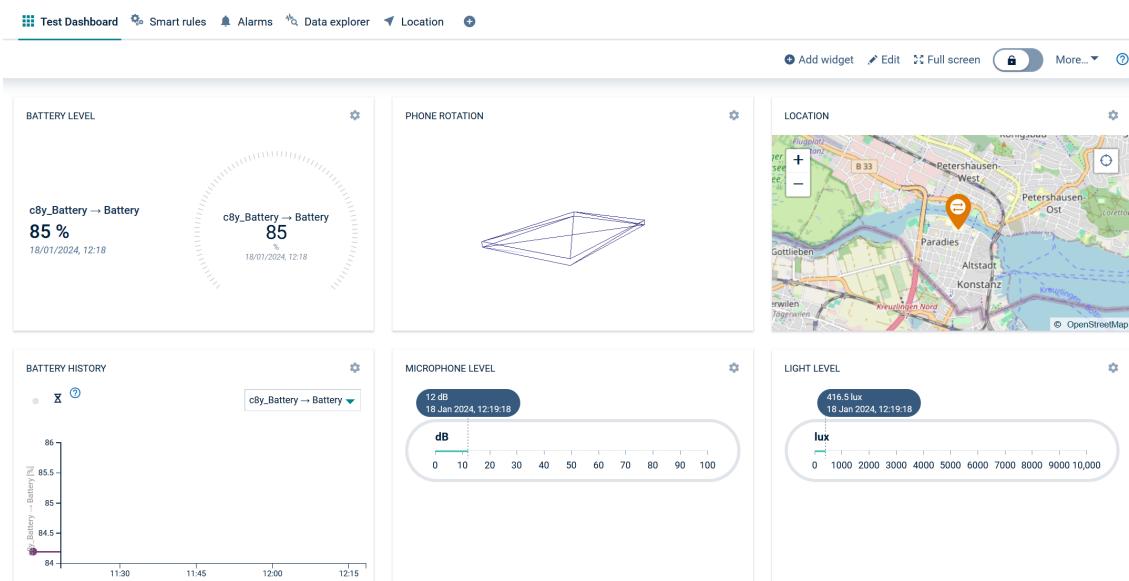


Figure 5.1: Cumulocity Dashboard

5.1.2 Task 2: Smart Rules

The second task is about creating a smart rule in *Cumulocity* portal. There we will use our already connected smartphone to trigger an alarm when it is upright or level. Like in the first task this was straight forward following the instructions on the tutorial page. As i did not read further at the task where the vibration alarm was set up to the next section, where it was described how to stop the vibration alarm again, my phone turned into a device we do not want to go to deep into detail here. Anyway bellow you will see the crated smart rule and the alarm which was triggered.

The screenshot shows the Cumulocity Smart rules dashboard with four active smart rules listed:

- Create a alarm when phone is upright**: On measurement explicit threshold create alarm. Status: Active.
- Create alarm when phone is level**: On measurement explicit threshold create alarm. Status: Active.
- Stop vibrating when level**: On alarm execute operation. Status: Active.
- Vibrate phone when upright**: On alarm execute operation. Status: Active.

Figure 5.2: Cumulocity Smart Rule

The screenshot shows the Cumulocity Alarms dashboard with the following categories and details:

- Critical**: 1 alarm for "Phone is level" (18 Jan 2024, 18:31:28, P30 Pro).
- Major**: 0 alarms.
- Minor**: 0 alarms.
- Warning**: 0 alarms.

Figure 5.3: Cumulocity Alarm

5.1.3 Task 3: Analytics Builder

The third task is about creating a analytics builder which could be seen as a more advanced version of the smart rules. With the analytics builder you can create more complex rules and also use the data from multiple devices and react on this data on a more complex way. The task was to implement a builder which will detect if the smartphone is shaken and then trigger an alarm. This was done by using the accelerometer data from the smartphone and a rule which will trigger an alarm if the acceleration is above a certain threshold. As before this was straight forward following the instructions on the tutorial page.

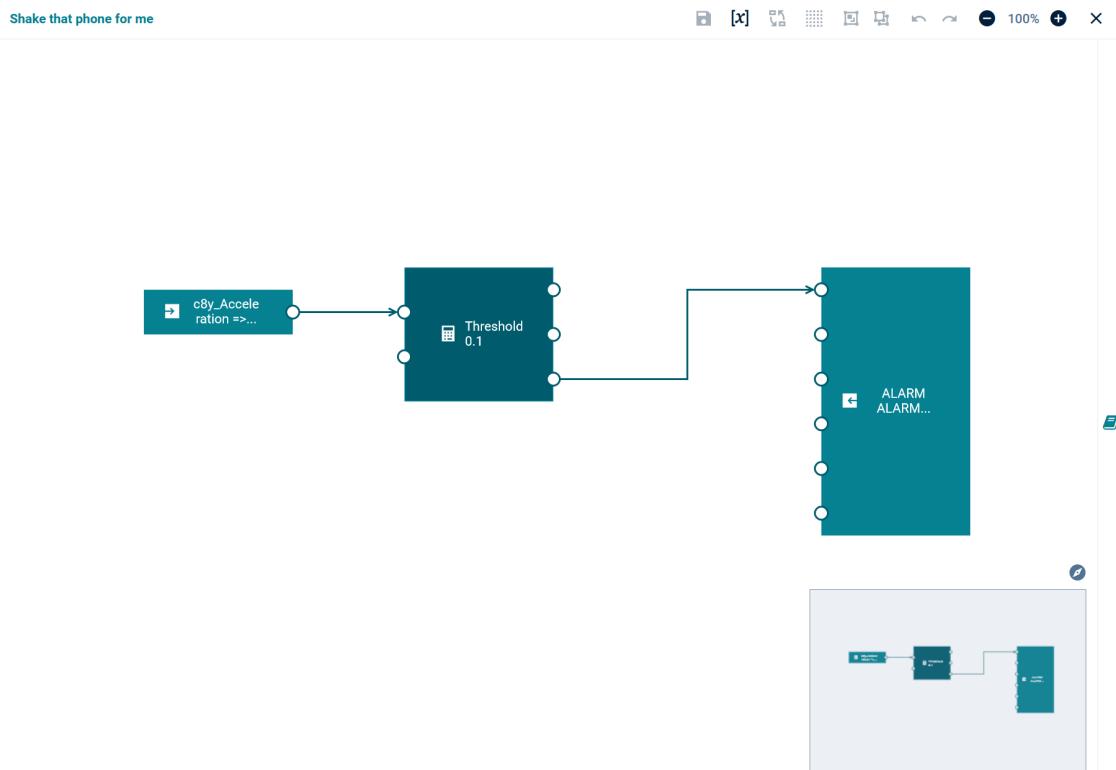


Figure 5.4: Cumulocity Analytics Builder

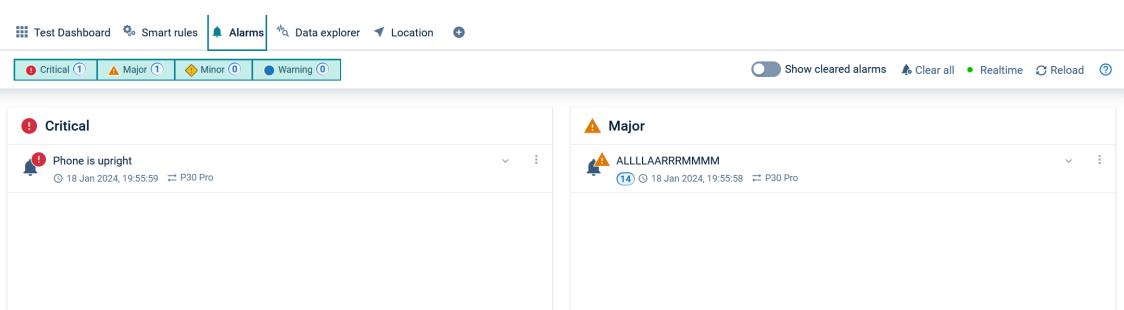


Figure 5.5: Cumulocity Alarm