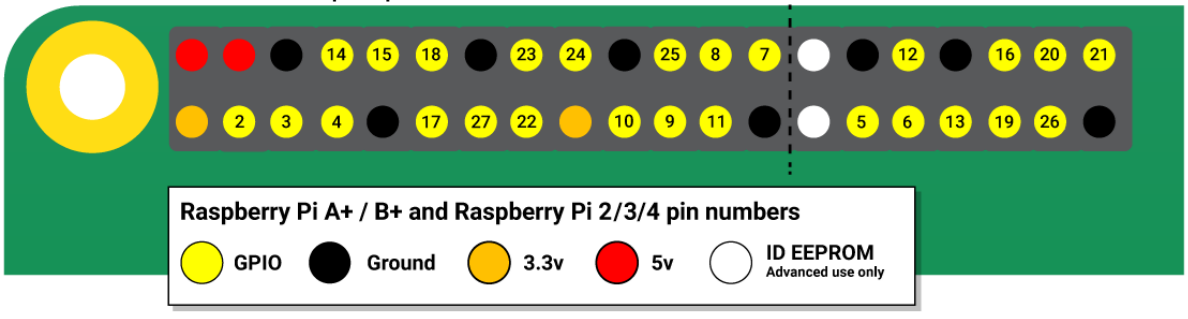
Sprawozdanie Laboratoria nr.4

Jako że nie mogłem się połączyć z Onion, lub występowały błędy skali czujnika (-255), postanowiłem spróbować sił na własną rękę i zamówić DHT11. Podłączyłem go do własnego RaspberryPi3B+. Żeby wykonać zadanie musiałem skonfigurować urządzenie w sieci domowej. Starałem się mierzyć temp i wilgotność na podstawie wiedzy zdobytej na laboratoriach w celu napisania niniejszego sprawozdania, które zawiera wszystkie założenia narzucone przez prowadzącego.

Czujnik zakupiony ze strony:
https://botland.com.pl/czujniki-multifunkcyjne/1886-czujnik-temperatury-i-wilgotnosci-dht11-modul-przewody-5903351242448.html

Po wczytaniu się w dokumentację Rpi, ponieważ każdy model może mieć różne sekcje zasilania GPIO znalazłem podpowiedź:



Montaż czujnika i skonfigurowanie wykonałem po zapoznaniu się z danymi technicznymi zasilania oraz poszukałem trochę w intrenecie. Artykuły, który spełniły moje oczekiwania w najlepszym stopniu, załączam w linku poniżej. Opisują one podłączenie i konfiguracje czujnika.

https://kostrzewinki.pl/1-schemat-podlaczenia-czujnika-temperatury-wilgotnosci-dht-11/
https://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-the-raspberry-pi/

Ja postanowiłem wykonać zadanie na systemie operacyjnym Arch. Na decyzję moją wpłyną fakt, że znam trochę Linuxa na jego podstawie. Wiem, jak poradzić sobie z ewentualnymi problemami oraz ze względu na dużą bazę repozytoriów i społeczność. Ponad to system ten nie instaluje nic ponad to co jest wymagane, co pozwala oszczędzić zasoby oraz nie marnuje ich zbędne konfiguracje. A dla mnie to kolejna możliwość na lepsze poznanie tego systemu.

1. Na wstępie należy przygotować kartę micro SD przed montażem w malince (konfiguracja karty również zrobiona z poziomu linuxa – najlepiej poniższe kroki wykonać jako administrator):
   - lsblk
   - fdisk /dev/sdX – tu wpisujemy adres naszej SD (mmcblk….)
     następnie w programie fdisk:
     - **o** – wyczyści dysk
     - **p** – wyświetli listę partycji (powinna być pusta)
     - **n** następnie **p** później **1** i **ENTER** jako domyślny sektor początkowy, wpisujemy **+200M** jako ostatni sektor partycji
     - **t** następnie **c** aby ustawić typ *FAT32* (to umożliwi rozruch systemu operacyjnego)
     - **n** następnie **p** później **2** i same **ENTER**Y (chyba że chcemy wprowadzić inny rozmiar dysku – domyślnie zajmie resztę)
     - **w** – zapisuje poczynione zmiany i wychodzi z programu
   - mkfs.vfat /dev/sdX1 — formatowanie pierwszej partycji
   - mkdir boot — oraz montowanie katalogu
   - mount /dev/sdX1 boot — rozruchowego
   - mkfs.ext4 /dev/sdX2 — formatowanie drugiej partycji
   - mkdir root — tworzenie głównego drzewa
   - mount /dev/sdX2 /root — katalogu root
   - wget *http://os.archlinuxarm.org/os/ArchLinuxARM-rpi-armv7-latest.tar.gz* - pobieramy odpowiednie do generacji ARM oprogramowanie
   - bsdtar -xpf ArchLinuxARM-rpi-armv7-latest.tar.gz -C root – tu może wystąpić problem i należy wpisać scieżkę do zamontowanego root (w takiej sytuacji pomoże – **lsblk**)
   - sync
   - mv root/boot/* boot – tu również może wystąpić błąd ścieżki – jw.
   - umount -R boot root
2. System gotowy. Kartę, kabel Ethernet montuje w malince i uruchamiam (login *root* hasło *root*, użytkownik domyślny *alarm* hasło *alarm*). Następnie z komputera przy pomocy programu nmap przeszukuję własną sieć i łączę się z Rpi i własnym kontem utworzonym przy którejś z prób przez SSH:

```
[markiel@archlinux ~]$ su -
Hasło:
[root@archlinux ~]# nmap -sn 192.168.100.0/24
Starting Nmap 7.93 ( https://nmap.org ) at 2023-04-06 14:31 CEST
Nmap scan report for 192.168.100.1
Host is up (0.0019s latency).
MAC Address: [redacted] (Huawei Technologies)
Nmap scan report for 192.168.100.5
Host is up (0.094s latency).
MAC Address: [redacted] (Xiaomi Communications)
Nmap scan report for 192.168.100.161
Host is up (1.1s latency).
MAC Address: [redacted] (InPro Comm)
Nmap scan report for 192.168.100.164
Host is up (0.062s latency).
MAC Address: [redacted] (Raspberry Pi Foundation)
Nmap scan report for 192.168.100.163
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 5.67 seconds
[root@archlinux ~]# ssh markiel@192.168.100.164
The authenticity of host '192.168.100.164 (192.168.100.164)' can't be established.
ED25519 key fingerprint is SHA256:ht5Pl1NaZ8VvxT/O49TSVq4bFmJh/nQcKtSX6MpgBQY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.100.164' (ED25519) to the list of known hosts.
markiel@192.168.100.164's password:
Linux raspberrypi 5.15.84-v7+ #1613 SMP Thu Jan 5 11:59:48 GMT 2023 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Apr  4 11:39:08 2023 from 192.168.100.163
markiel@raspberrypi:~ $
```

3. Zalecane jest po pierwszym uruchomieniu wykonać aktualizację kluczy repozytoriów dla wszystkich architektur, które są podpisywane przez system kompilacji opiekunów projektu. Następnie przez to, że system jest czysty potrzebować będziemy:
   - Postępuje wg instrukcji *https://github.com/adafruit/Adafruit_Python_DHT*
   - pacman -S base-devel sudo wget python pip
   - sudo pip install Adafruit-DHT ; sudo pip install gpiozero
   - sudo python3 -m pip install --upgrade pip setuptools wheel
4. Zainstalowałem wszystkie niezbędne składniki, więc można sprawdzić działanie czujnika. W moim przypadku czujnik jest podłączony do pinu 14.

```
markiel@raspberrypi:~/Adafruit_Python_DHT/examples $ sudo ./AdafruitDHT.py 11 14
Temp=26.0*  Humidity=42.0%
```

```
markiel@raspberrypi:~/Adafruit_Python_DHT/examples $ python3 AdafruitDHT.py 11 14
Temp=27.0*  Humidity=38.0%
```

5. Kiedy już wiem, że komunikacja pomiędzy malinką, a czujnikiem jest prawidłowa przystąpiłem do rejestracji na platformie ubidot.com oraz wstępnych konfiguracji skryptu, oto różnice wprowadzone w kodzie dostarczonym przez ubidots.

```
0a1,2
> #!/usr/bin/env python
>
4a7,8
> import sys
> import Adafruit_DHT
6,7c10,11
< TOKEN = "..."  # Put your TOKEN here
< DEVICE_LABEL = "machine"  # Put your device label here
---
> TOKEN = "BBFF-oon7f2jN3cyw66OsGD8SQroQehblOZ"  # Put your TOKEN here
> DEVICE_LABEL = "rpi2"  # Put your device label here
15,16c19
<     value_1 = random.randint(-10, 50)
<     value_2 = random.randint(0, 85)
---
>     value_2, value_1 = Adafruit_DHT.read_retry(Adafruit_DHT.DHT11, 14)
19,22c22,23
<     lat = random.randrange(34, 36, 1) + \
<         random.randrange(1, 1000, 1) / 1000.0
<     lng = random.randrange(-83, -87, -1) + \
<         random.randrange(1, 1000, 1) / 1000.0
---
>     lat = 53.4261129
>     lng = 14.529831,15
```

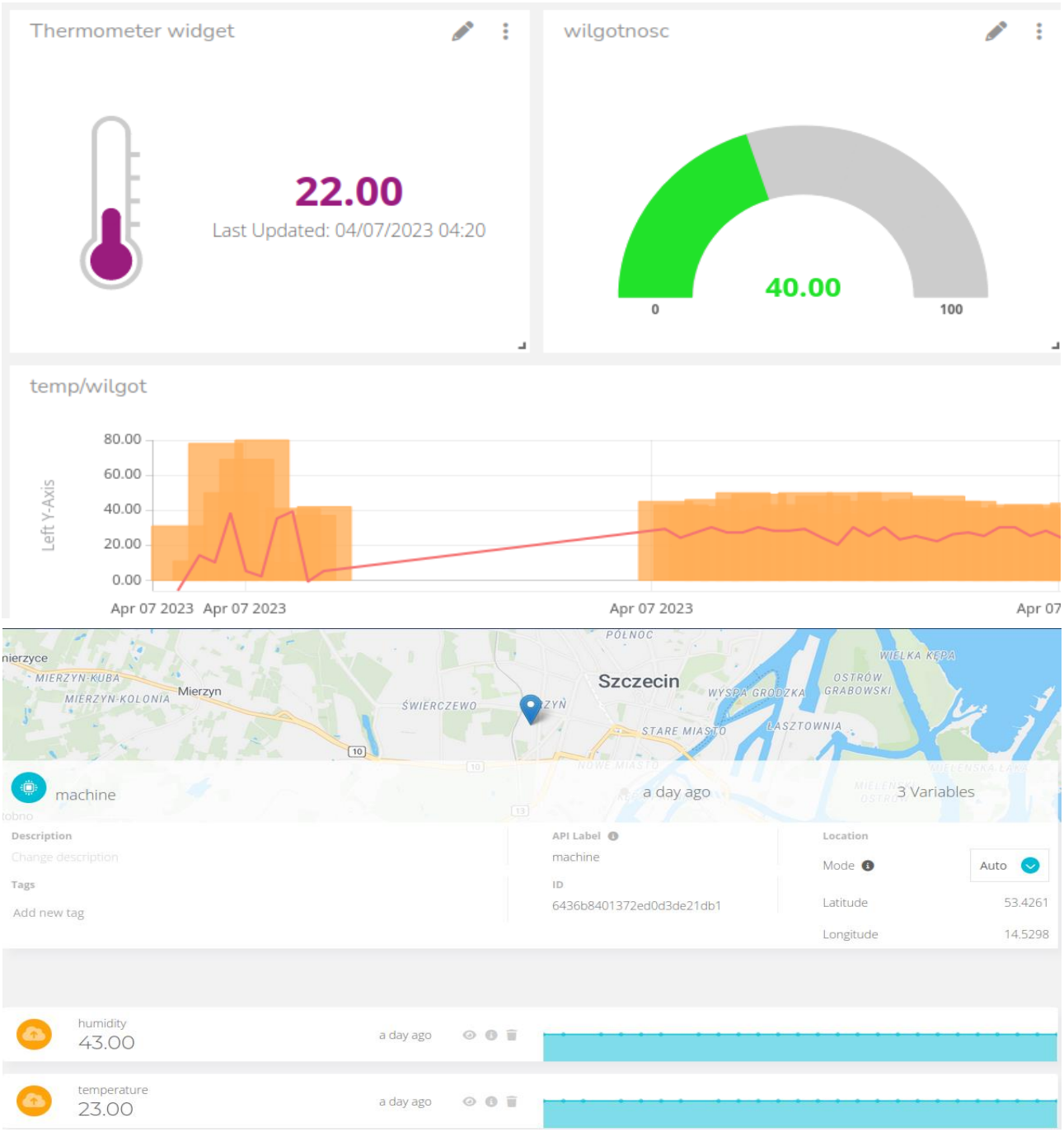
skrypt z ubidots z wprowadzonym tokenem

```
#!/usr/bin/env python

import time
import requests
import math
import random
import sys
import Adafruit_DHT

TOKEN = "BBFF-oon7f2jN3cyw66OsGD8SQroQehblOZ"  # Put your TOKEN here
DEVICE_LABEL = "rpi2"  # Put your device label here
VARIABLE_LABEL_1 = "temperature"  # Put your first variable label here
VARIABLE_LABEL_2 = "humidity"  # Put your second variable label here
VARIABLE_LABEL_3 = "position"  # Put your second variable label here


def build_payload(variable_1, variable_2, variable_3):
    # Creates two random values for sending data
    value_2, value_1 = Adafruit_DHT.read_retry(Adafruit_DHT.DHT11, 14)

    # Creates a random gps coordinates
    lat = 53.4261129
    lng = 14.529831,15
    payload = {variable_1: value_1,
               variable_2: value_2,
               variable_3: {"value": 1, "context": {"lat": lat, "lng": lng}}}

    return payload


def post_request(payload):
    # Creates the headers for the HTTP requests
    url = "http://industrial.api.ubidots.com"
    url = "{}/api/v1.6/devices/{}".format(url, DEVICE_LABEL)
    headers = {"X-Auth-Token": TOKEN, "Content-Type": "application/json"}
```

```
import sys

import Adafruit_DHT


# Parse command line parameters.
sensor_args = { '11': Adafruit_DHT.DHT11,
                '22': Adafruit_DHT.DHT22,
                '2302': Adafruit_DHT.AM2302 }
if len(sys.argv) == 3 and sys.argv[1] in sensor_args:
    sensor = sensor_args[sys.argv[1]]
    pin = sys.argv[2]
else:
    print('Usage: sudo ./Adafruit_DHT.py [11|22|2302] <GPIO pin number>')
    print('Example: sudo ./Adafruit_DHT.py 2302 4 - Read from an AM2302 connected to GPIO pin #4')
    sys.exit(1)

# Try to grab a sensor reading.  Use the read_retry method which will retry up
# to 15 times to get a sensor reading (waiting 2 seconds between each retry).
humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
```

dokumentacja z Adafruit

6. A oto efekty końcowe

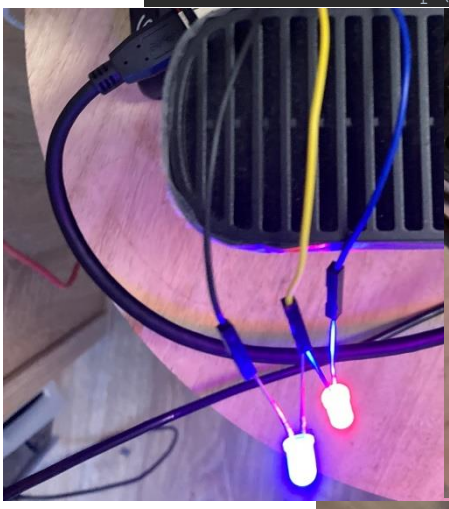
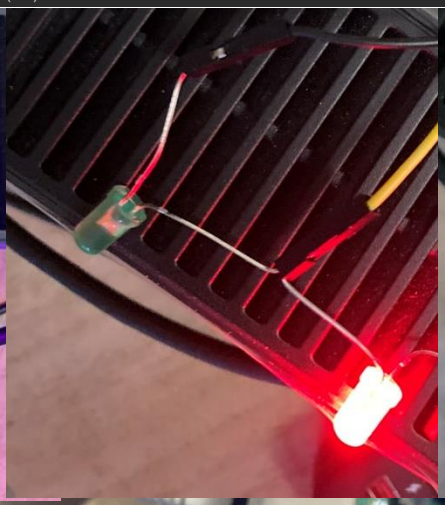
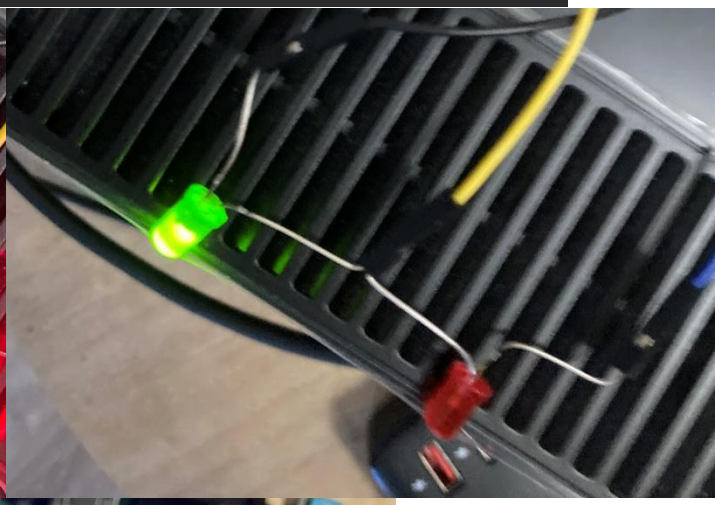
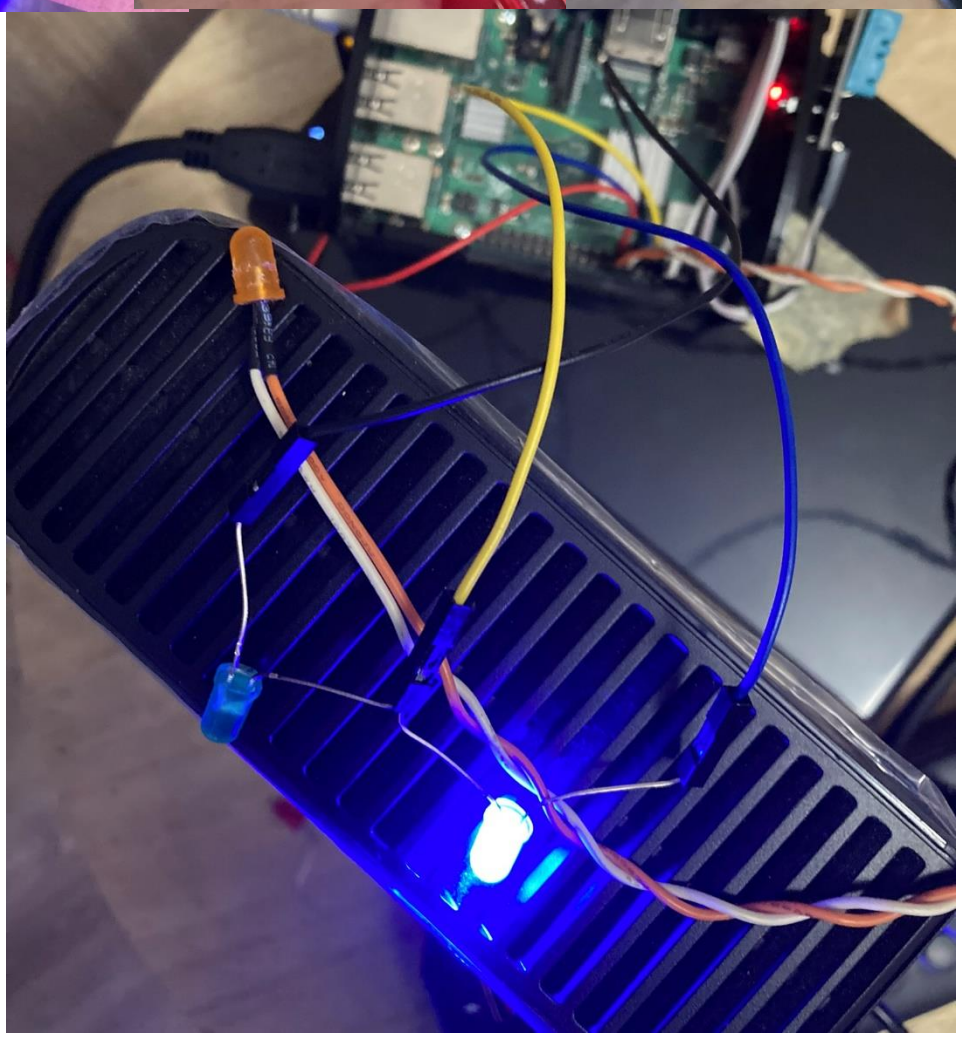


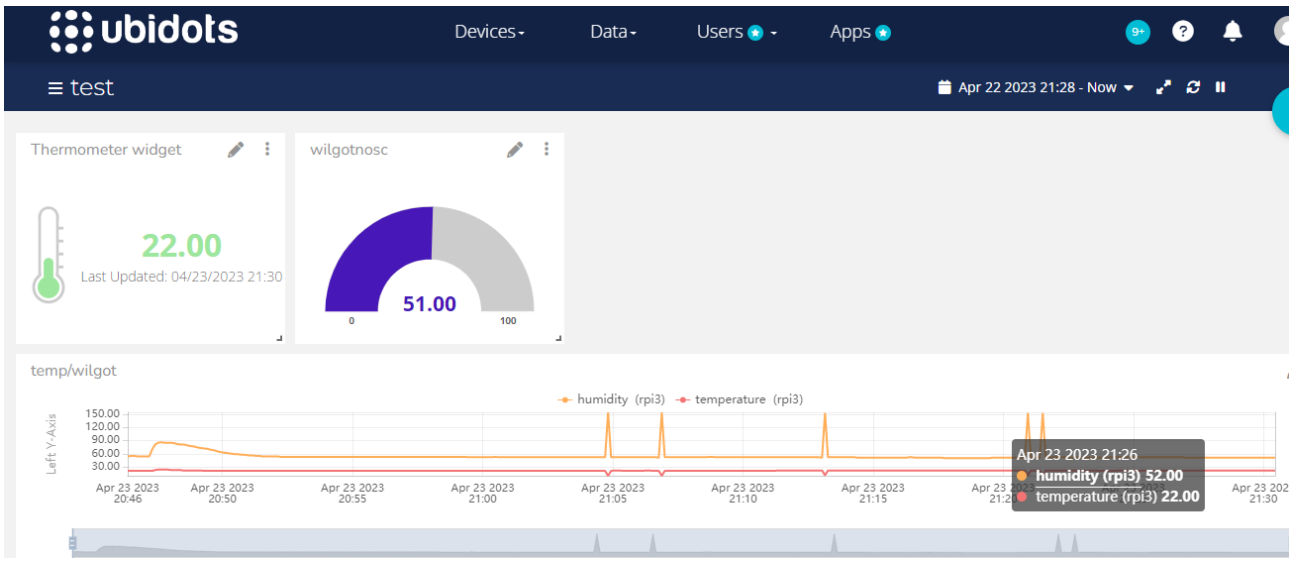
7. Po uwzględnieniu zająć z 23.04.2023 zaktualizowałem kod; wszystko działa:

```
#!/usr/bin/env python

import time
import requests
import math
import random
import sys
import Adafruit_DHT
from gpiozero import LED

TOKEN = "..."  # Put your TOKEN here
DEVICE_LABEL = "rpi3"  # Put your device label here
VARIABLE_LABEL_1 = "temperature"  # Put your first variable label here
VARIABLE_LABEL_2 = "humidity"  # Put your second variable label here
VARIABLE_LABEL_3 = "position"  # Put your second variable label here

blue = LED(17)
```

```
green = LED(27)
red = LED(23)

def build_payload(variable_1, variable_2, variable_3):
    # Creates two random values for sending data
    humidity, temperature = Adafruit_DHT.read_retry(Adafruit_DHT.DHT11, 14)
    print(f'temp: {temperature}, hum:{humidity}')
    # Handle temperature LED status
    if temperature > 23:
        green.off()
        blue.off()
        red.on()
    elif temperature < 22:
        green.off()
        red.off()
        blue.on()
    else:
        green.on()
        red.off()
        blue.off()

    # Creates a random gps coordinates
    lat = 53.4261129
    lng = 14.529831,15
    payload = {variable_1: temperature,
               variable_2: humidity,
               variable_3: {"value": 1, "context": {"lat": lat, "lng": lng}}}

    return payload


def post_request(payload):
    # Creates the headers for the HTTP requests
    url = "http://industrial.api.ubidots.com"
    url = "{}/api/v1.6/devices/{}".format(url, DEVICE_LABEL)
    headers = {"X-Auth-Token": TOKEN, "Content-Type": "application/json"}

    # Makes the HTTP requests
    status = 400
    attempts = 0
    while status >= 400 and attempts <= 5:
        req = requests.post(url=url, headers=headers, json=payload)
        status = req.status_code
        attempts += 1
        time.sleep(1)

    # Processes results
    print(req.status_code, req.json())
    if status >= 400:
        print("[ERROR] Could not send data after 5 attempts, please check \
            your token credentials and internet connection")
        return False

    print("[INFO] request made properly, your device is updated")
    return True


def main():
    payload = build_payload(
        VARIABLE_LABEL_1, VARIABLE_LABEL_2, VARIABLE_LABEL_3)

    print("[INFO] Attemping to send data")
    post_request(payload)
    print("[INFO] finished")
```

```
if __name__ == '__main__':
    while (True):
        main()
        time.sleep(5)
```



8. Jeśli chodzi o blynk.io to zamknęło darmową współpracę z Rpi, ale znalazłem kilka rozwiązań. W owe rozwiązania na pewno się bardziej wczytam w wolnej chwili, ponieważ zaintrygował mnie temat IoT i cloud oraz możliwości wyświetlania czy to na komputerze czy urządzeniach mobilnych. Jak wynika z dokumentacji z GitHub Rpi wspiera „Dashoabrd blynk" przez: *https://github.com/blynkkk/blynk-library*

Najprawdopodobniej najlepszym rozwiązaniem będzie sprawdzenie nowego systemu poprzez wgranie na kartę microSD systemu od OperWRT, oto dokumentacja i instrucja, którą wykorzystam w późniejszym czasie.

https://git.openwrt.org/?p=openwrt%2Fopenwrt.git&a=search&h=HEAD&st=commit&s=Raspberry%20Pi%203
https://openwrt.org/toh/hwdata/raspberry_pi_foundation/raspberry_pi_3_bplus
https://openwrt.org/toh/raspberry_pi_foundation/raspberry_pi#installation