Developing an IoT System

Uli Raich

Two lectures on Hardware and Software for
the Internet of Things applied to Air Quality measurement

Lecture 2: Accessing the “things” through the Internet

Presented online at the Seminar – Air Quality and
IoT based Air Sensors Nov. 2023

The “I” in IoT

To communicate with the IoT node over we Internet we must

- Connect the Node to the WiFi network
- Provide a TCP or WEB server (which is a particular type of TCP server!)
- We may need additional protocols like “server side events” or WEB sockets accessed through JavaScript

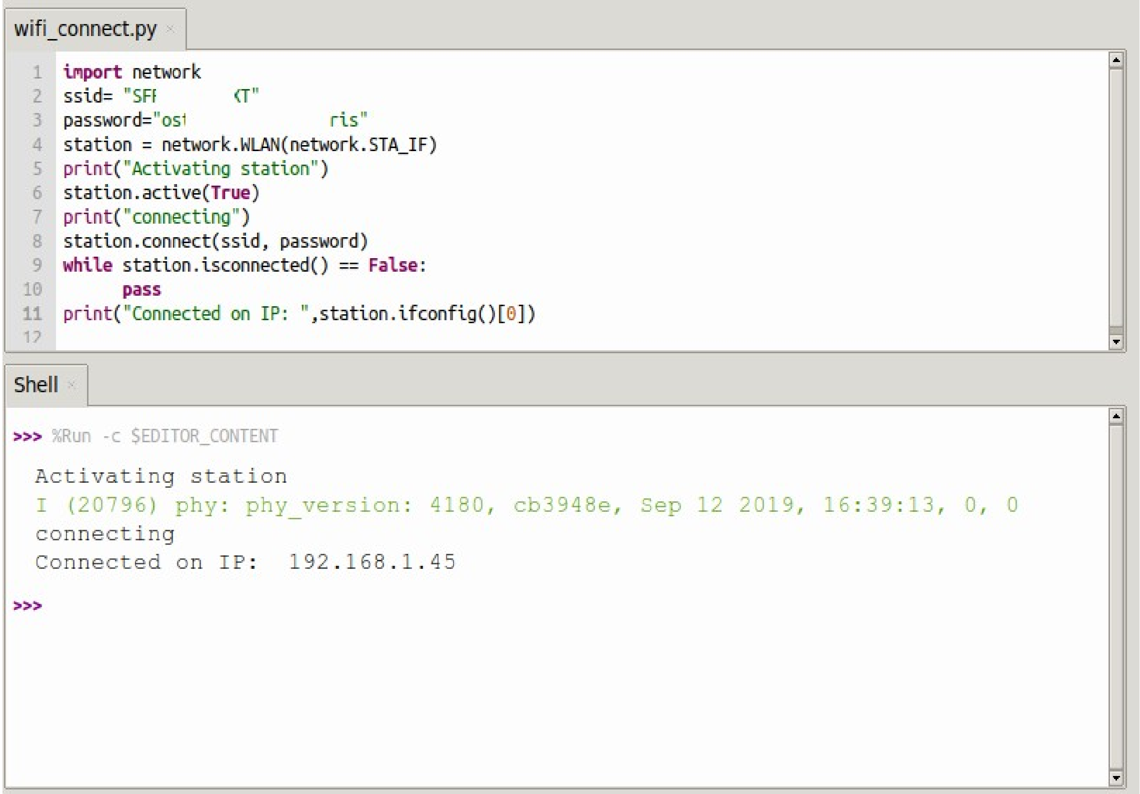
  or

- Communicate to an MQTT broker, which in turn sends or receives data from an MQTT publish or subscribe client

Connecting to the WiFi network

Micropython provides the WLAN class giving access to methods to

- Activate the WiFi station and check the activation
- Scan for nodes in the neighborhood
- connect/disconnect
- Get the network status
- Get/set IP level information



```
import network
ssid= "[redacted]"
password="[redacted]"
station = network.WLAN(network.STA_IF)
print("Activating station")
station.active(True)
print("connecting")
station.connect(ssid, password)
while station.isconnected() == False:
      pass
print("Connected on IP: ",station.ifconfig()[0])
```

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

  Activating station
  I (20796) phy: phy_version: 4180, cb3948e, Sep 12 2019, 16:39:13, 0, 0
  connecting
  Connected on IP:  192.168.1.45

>>>
```

A WiFi connection class

When working on IoT you must connect to the network very often.

I therefore wrote and integrated a module named *wifi_connect*

This makes connecting to the network super-simple:

```
from wifi_connect import *

connect()
```

connect also gets the current time from ntp and sets the real time clock on the ESP32

You can get the IP address with

getIPAddress()

or the current time with

gmtTime() or

cetTime()

d

A simple TCP server/client example

The server:

- create a socket
- bind the host address to a port
- listen for connection requests
- accept the connection
- receive data from the connection
- send data to the connection
- Close the connection

A simple TCP server/client example

The client:

- create a socket
- connect to the server
- send data
- receive data

Let’s try it on the PC first!

d

TCP server on the ESP32

There is no difference with respect to the code on the PC

Except prior connection to the WiFi network.

Now we can create a TCP server on the ESP32 that reads some sensors and sends the results to the PC

d

TCP server on the ESP32

There is no difference with respect to the code on the PC

Except prior connection to the WiFi network.

Now we can create a TCP server on the ESP32 that reads some sensors and sends the results to the PC

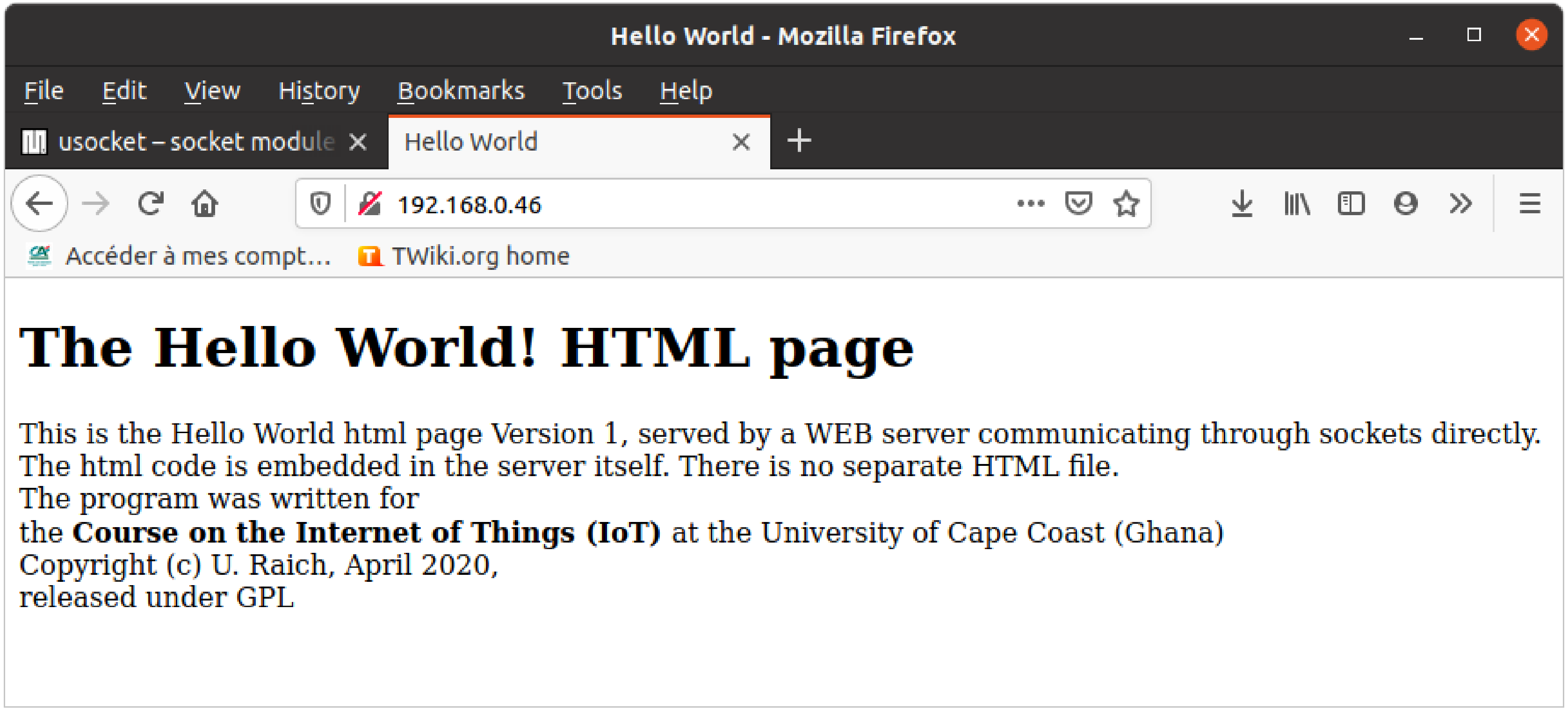
d

A simple WEB server

As we have seen, MicroPython contains a socket class for network access and that is all that is needed to implement a simple WEB server.

To make things even simpler a basic framework name *picoweb* is available on github.
I integrated this framework into the MicroPython binary to make it globally accessible

The first WEB page



The Hello World! HTML page

This is the Hello World html page Version 1, served by a WEB server communicating through sockets directly.
The html code is embedded in the server itself. There is no separate HTML file.
The program was written for
the **Course on the Internet of Things (IoT)** at the University of Cape Coast (Ghana)
Copyright (c) U. Raich, April 2020,
released under GPL

d

picoweb

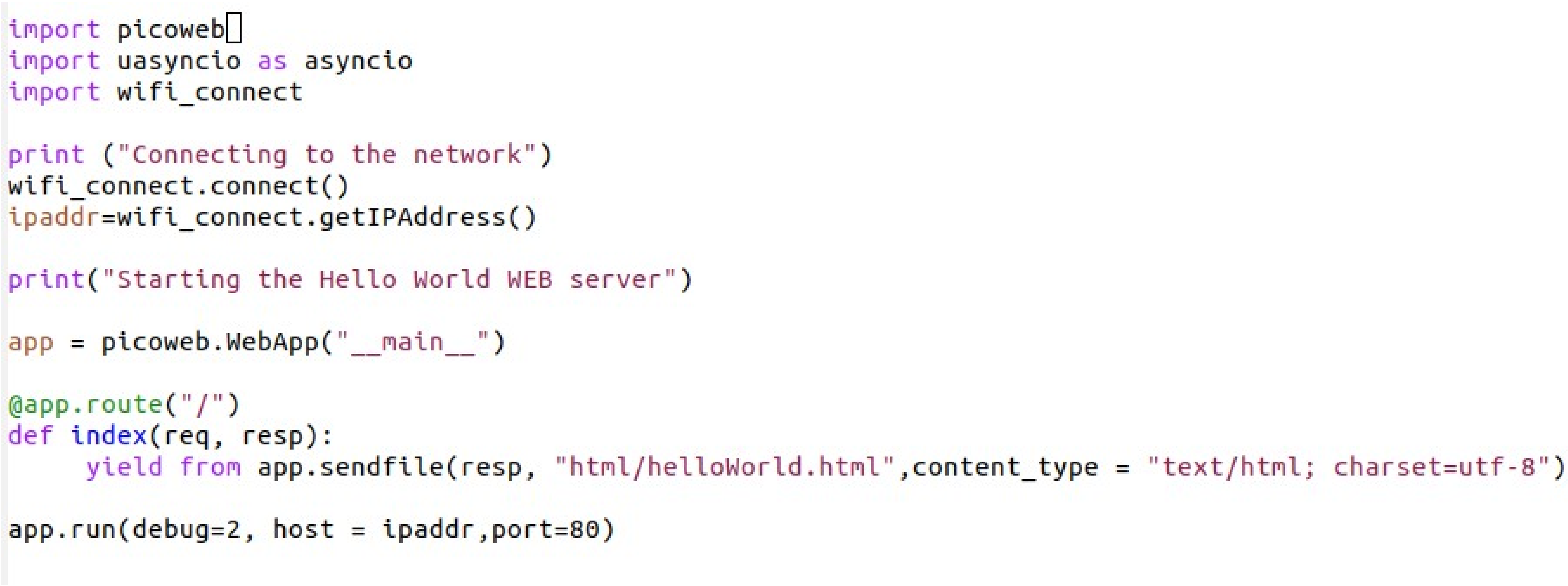
The picoweb module is a framework for writing WEB servers

If contains functionality to

- Create and listen to HTTP requests on a socket
- Handling routing
- Parse HTTP requests
- Prepare HTTP responses by sending the necessary header
- Send HTTP pages stored in files
- Handle templates

d

Hello World WEB server



```
import picoweb
import uasyncio as asyncio
import wifi_connect

print ("Connecting to the network")
wifi_connect.connect()
ipaddr=wifi_connect.getIPAddress()

print("Starting the Hello World WEB server")

app = picoweb.WebApp("__main__")

@app.route("/")
def index(req, resp):
     yield from app.sendfile(resp, "html/helloWorld.html",content_type = "text/html; charset=utf-8")

app.run(debug=2, host = ipaddr,port=80)
```

Integrating measurements into the WEB page

That is already not too bad!

However, we want to integrate measurements into the WEB page.

This can be done through templates

We define a HTML table and fill the entries with measurements made by the PMS5003 and the SHT30

A dictionary with the fields to be replaced as keys and the values to be put into the fields as values

picoweb’s method *render_template* will do the job.

d

Server Side events

This is still not perfect because we have to update the whole HTML page if we want to get new measurements

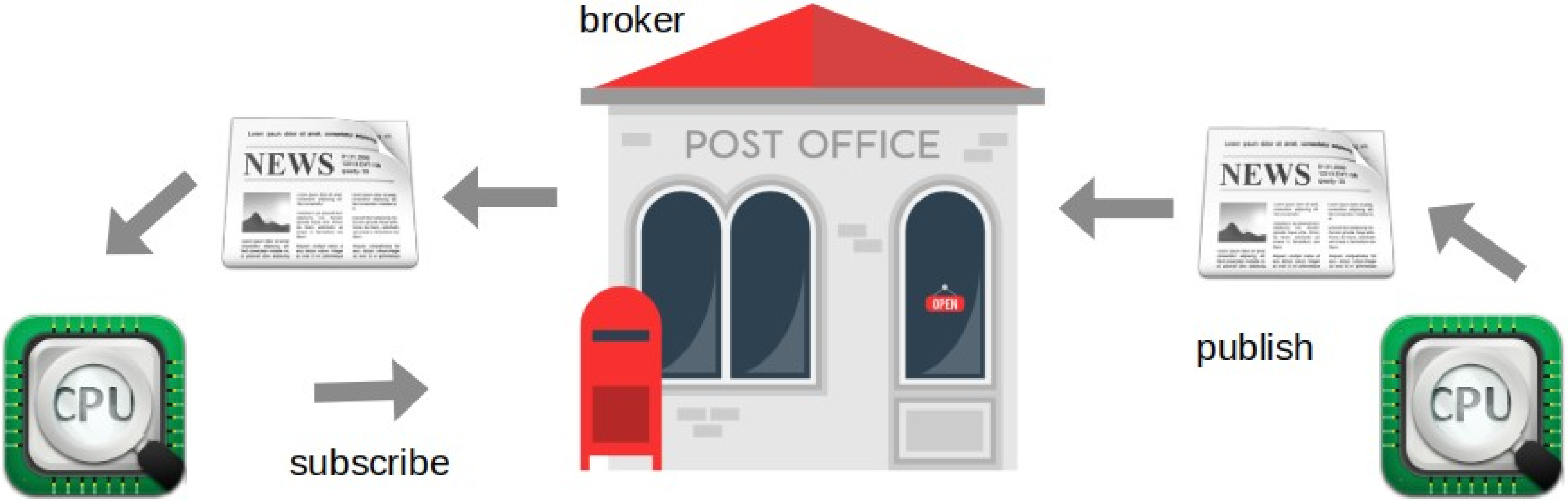
We would like the WEB server make periodic measurements, which update the page on the browser (client) side whenever they are sent

This can be achieved through server side events

d

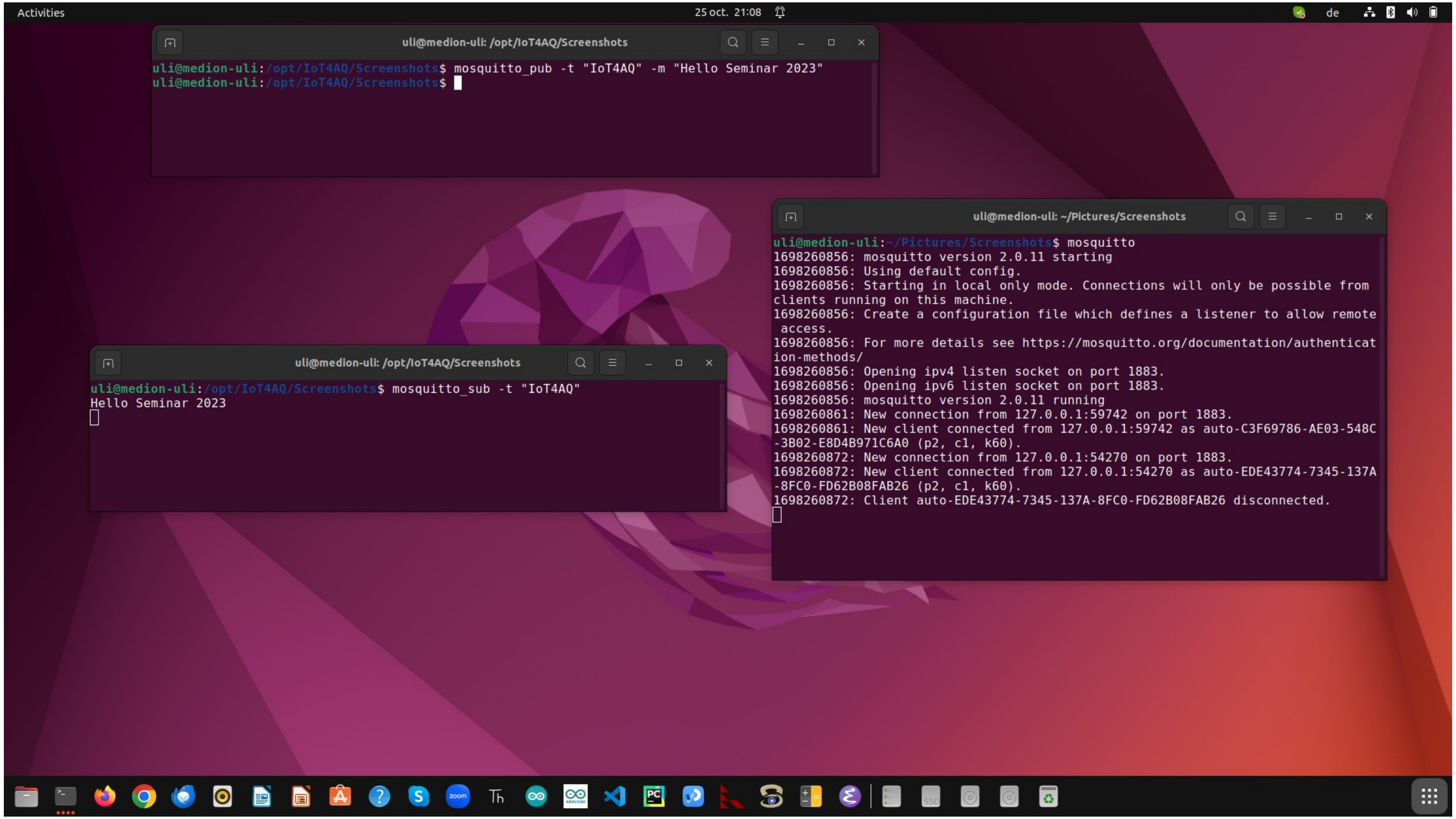
MQTT, another way to go online

Message Queuing Telemetry Transport: a publish-subscribe Protocol for IoT



The mosquitto broker

This works only on the local machine


Activities
25 oct. 21:08
de
uli@medion-uli: /opt/IoT4AQ/Screenshots

```
uli@medion-uli:/opt/IoT4AQ/Screenshots$ mosquitto_pub -t "IoT4AQ" -m "Hello Seminar 2023"
uli@medion-uli:/opt/IoT4AQ/Screenshots$ 
```

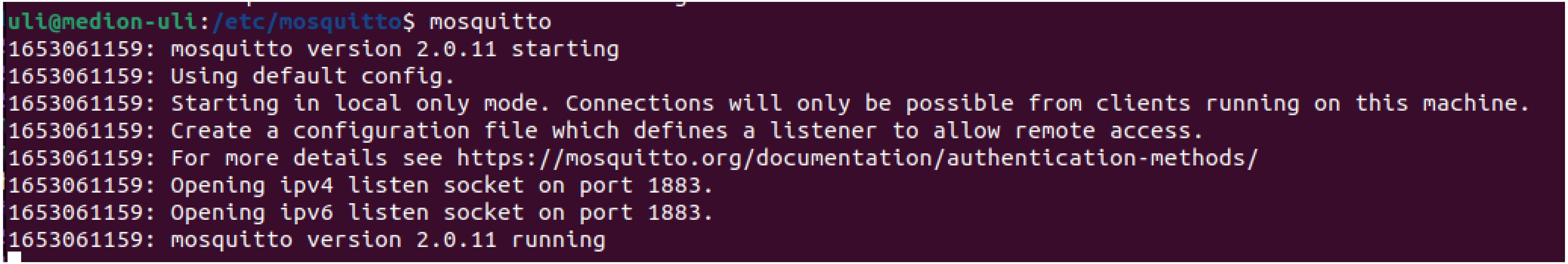
uli@medion-uli: ~/Pictures/Screenshots

```
uli@medion-uli:~/Pictures/Screenshots$ mosquitto
1698260856: mosquitto version 2.0.11 starting
1698260856: Using default config.
1698260856: Starting in local only mode. Connections will only be possible from
clients running on this machine.
1698260856: Create a configuration file which defines a listener to allow remote
 access.
1698260856: For more details see https://mosquitto.org/documentation/authenticat
ion-methods/
1698260856: Opening ipv4 listen socket on port 1883.
1698260856: Opening ipv6 listen socket on port 1883.
1698260856: mosquitto version 2.0.11 running
1698260861: New connection from 127.0.0.1:59742 on port 1883.
1698260861: New client connected from 127.0.0.1:59742 as auto-C3F69786-AE03-548C
-3B02-E8D4B971C6A0 (p2, c1, k60).
1698260872: New connection from 127.0.0.1:54270 on port 1883.
1698260872: New client connected from 127.0.0.1:54270 as auto-EDE43774-7345-137A
-8FC0-FD62B08FAB26 (p2, c1, k60).
1698260872: Client auto-EDE43774-7345-137A-8FC0-FD62B08FAB26 disconnected.
```

```
uli@medion-uli:/opt/IoT4AQ/Screenshots$ mosquitto_sub -t "IoT4AQ"
Hello Seminar 2023
```

d

Setting up mosquitto



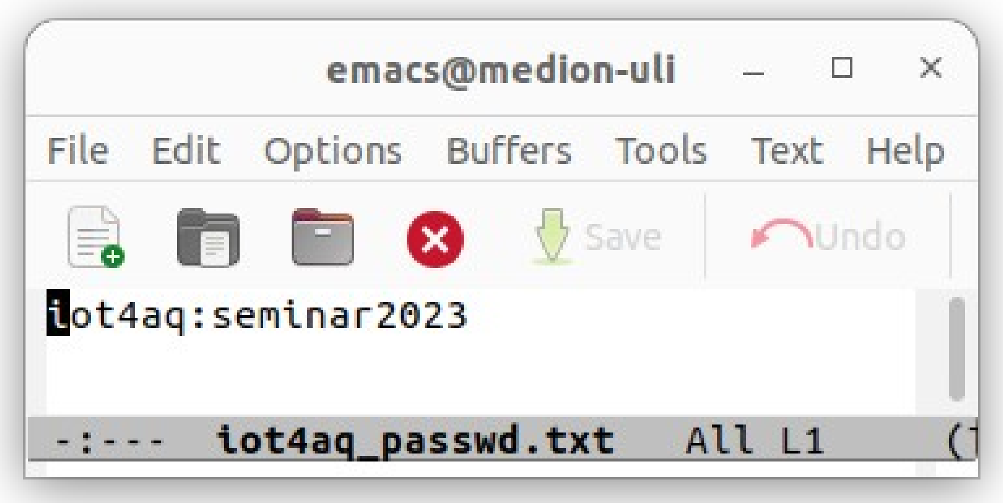
```
uli@medion-uli:/etc/mosquitto$ mosquitto
1653061159: mosquitto version 2.0.11 starting
1653061159: Using default config.
1653061159: Starting in local only mode. Connections will only be possible from clients running on this machine.
1653061159: Create a configuration file which defines a listener to allow remote access.
1653061159: For more details see https://mosquitto.org/documentation/authentication-methods/
1653061159: Opening ipv4 listen socket on port 1883.
1653061159: Opening ipv6 listen socket on port 1883.
1653061159: mosquitto version 2.0.11 running
```

When starting mosquitto you see that we have to define a listener and an authentication method if we want to access the broker from a remote machine like the ESP32

The easiest way to accomplish this is a password file

mosquitto password file

First we create a simple text file with a user name (iot4aq) and a password (seminar2023)

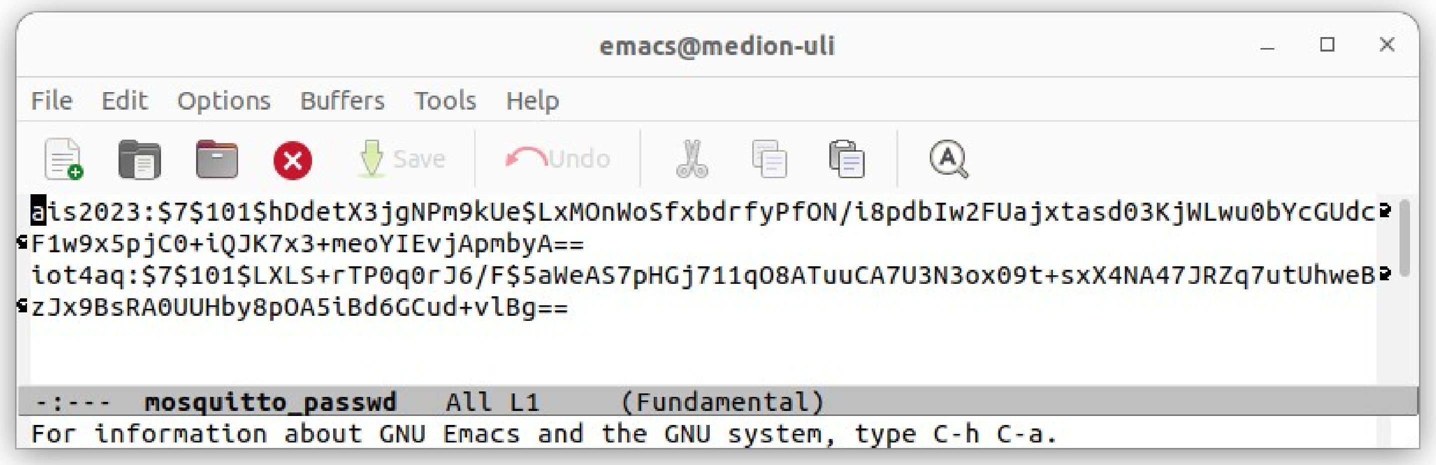


```
iot4aq:seminar2023
```

Encode password file

Then we encode it with mosquitto_passwd:

cp iot4aq_passwd.txt iot4aq_passwd # the file will be overwritten by the encoded password file
mosquitto_passwd -U iot4aq_passwd



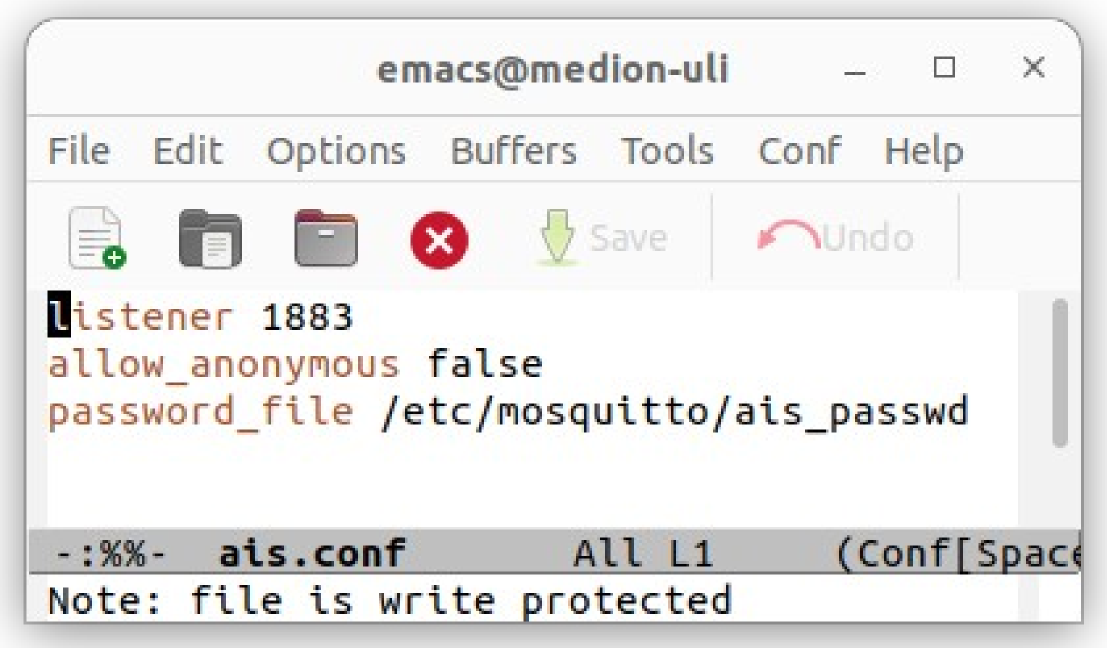
```
ais2023:$7$101$hDdetX3jgNPm9kUe$LxMOnWoSfxbdrfyPfON/i8pdbIw2FUajxtasd03KjWLwu0bYcGUdc
F1w9x5pjC0+iQJK7x3+meoYIEvjApmbyA==
iot4aq:$7$101$LXLS+rTP0q0rJ6/F$5aWeAS7pHGj711qO8ATuuCA7U3N3ox09t+sxX4NA47JRZq7utUhweB
zJx9BsRA0UUHby8pOA5iBd6GCud+vlBg==
```

```
-:---  mosquitto_passwd   All L1     (Fundamental)
For information about GNU Emacs and the GNU system, type C-h C-a.
```

… and we copy it to /etc/mosquitto/

Adapting the config file

Finally we create a custom mosquitto config file, which is located in /etc/mosquitto/conf.d enabling the password file.

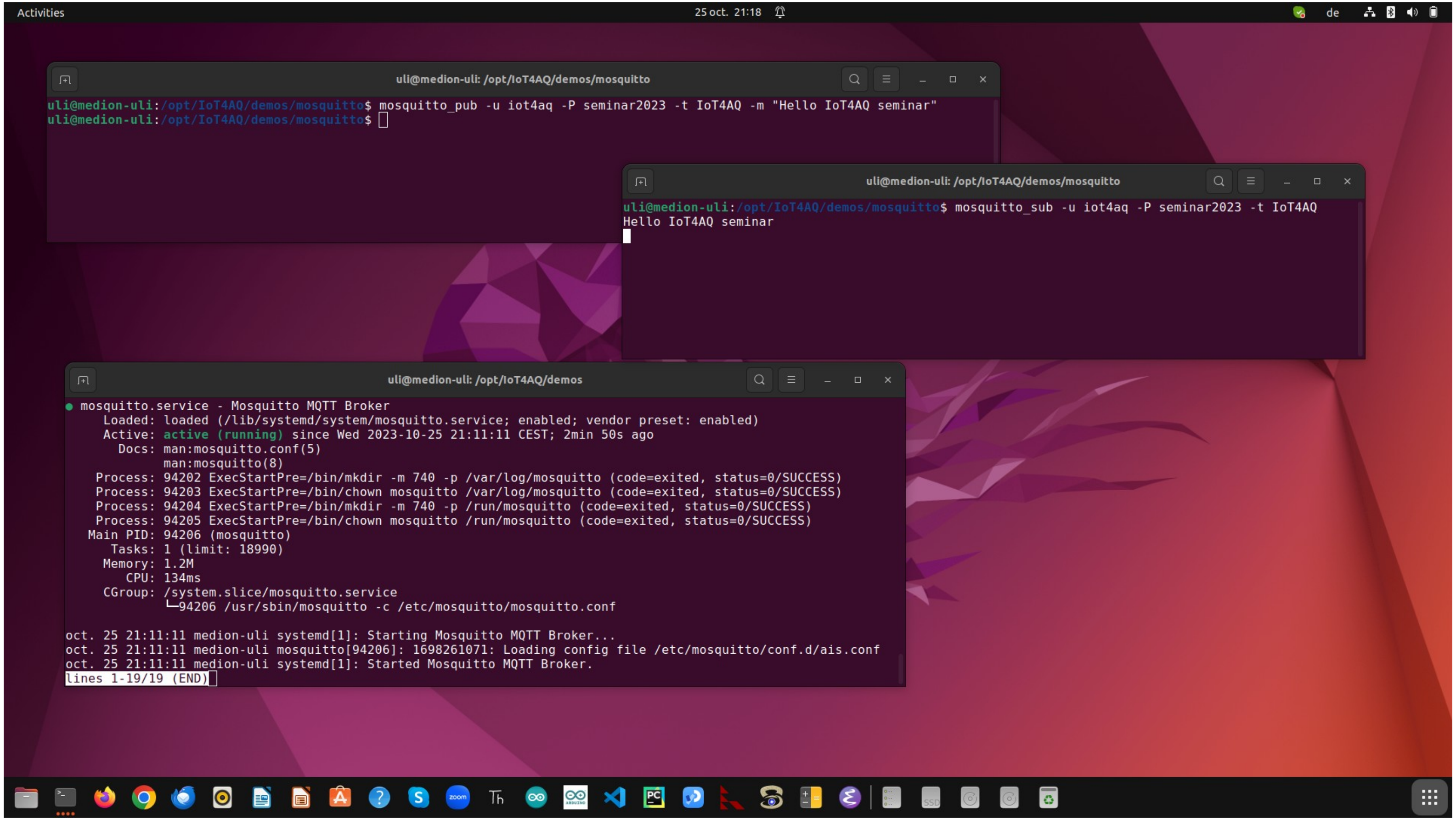


```
listener 1883
allow_anonymous false
password_file /etc/mosquitto/ais_passwd
```

```
-:%%-  ais.conf        All L1       (Conf[Space
Note: file is write protected
```

The mosquitto broker with password

This works also with a remote subscriber or publisher



Activities
25 oct. 21:18
de
uli@medion-uli: /opt/IoT4AQ/demos/mosquitto
uli@medion-uli: /opt/IoT4AQ/demos/mosquitto
uli@medion-uli: /opt/IoT4AQ/demos

```
uli@medion-uli:/opt/IoT4AQ/demos/mosquitto$ mosquitto_pub -u iot4aq -P seminar2023 -t IoT4AQ -m "Hello IoT4AQ seminar"
uli@medion-uli:/opt/IoT4AQ/demos/mosquitto$ 
```

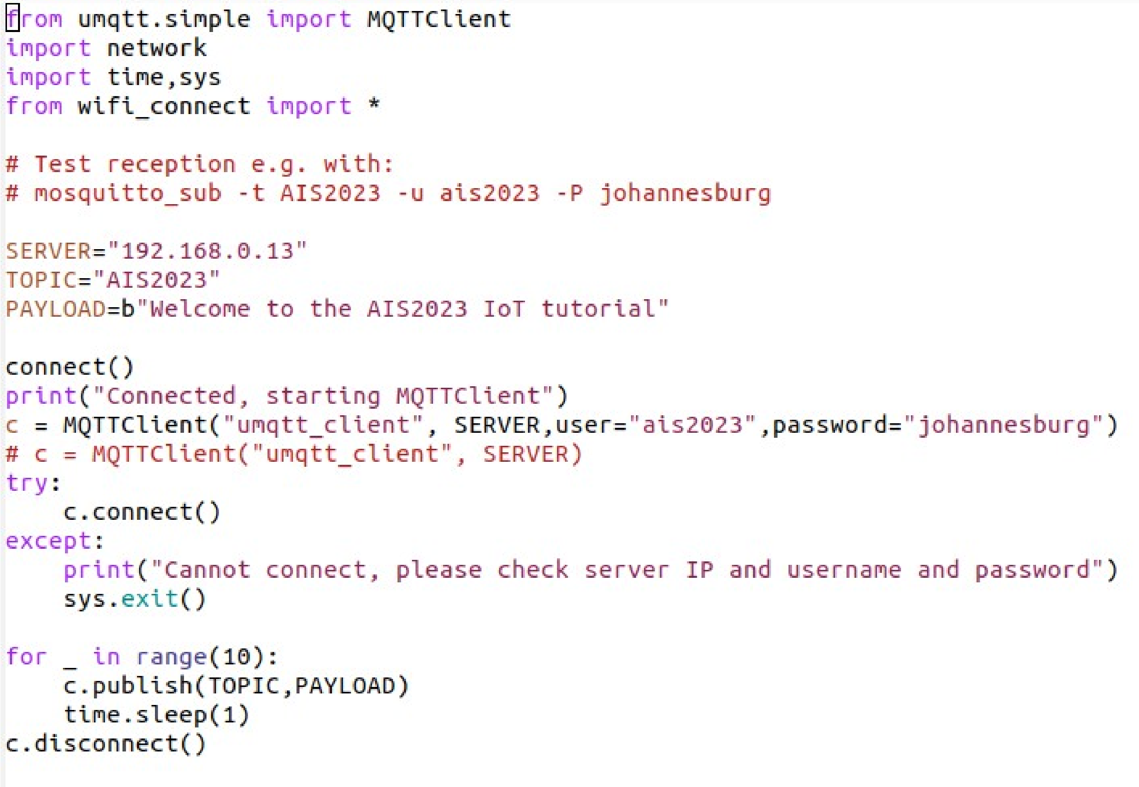
```
uli@medion-uli:/opt/IoT4AQ/demos/mosquitto$ mosquitto_sub -u iot4aq -P seminar2023 -t IoT4AQ
Hello IoT4AQ seminar
```

```
● mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2023-10-25 21:11:11 CEST; 2min 50s ago
       Docs: man:mosquitto.conf(5)
             man:mosquitto(8)
    Process: 94202 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
    Process: 94203 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
    Process: 94204 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
    Process: 94205 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
   Main PID: 94206 (mosquitto)
      Tasks: 1 (limit: 18990)
     Memory: 1.2M
        CPU: 134ms
     CGroup: /system.slice/mosquitto.service
             └─94206 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

oct. 25 21:11:11 medion-uli systemd[1]: Starting Mosquitto MQTT Broker...
oct. 25 21:11:11 medion-uli mosquitto[94206]: 1698261071: Loading config file /etc/mosquitto/conf.d/ais.conf
oct. 25 21:11:11 medion-uli systemd[1]: Started Mosquitto MQTT Broker.
lines 1-19/19 (END)
```

MQTT client on the ESP32

… and if I could have the MQTT client on the ESP32.

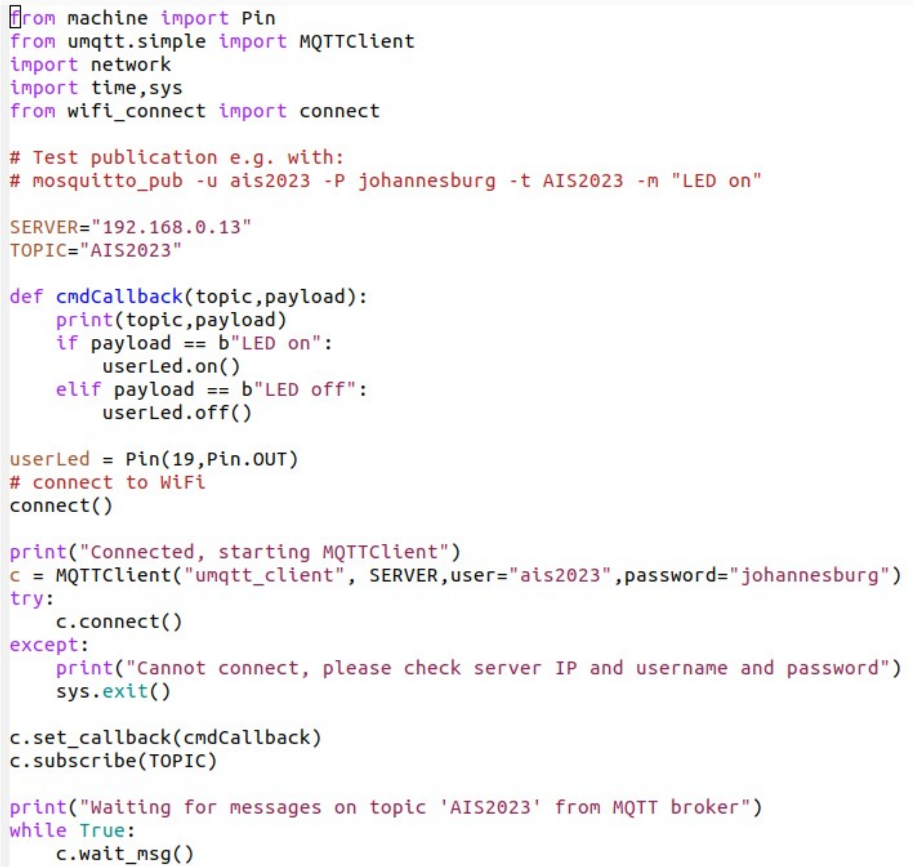
- If it was the publishing client it could send measurements to the broker and thus to any subscribed client
- If it was the subscribing client it could receive commands from the broker and thus from any publishing client
- micropython-lib supplies the umqtt library giving us access to MQTT



```
from umqtt.simple import MQTTClient
import network
import time,sys
from wifi_connect import *

# Test reception e.g. with:
# mosquitto_sub -t AIS2023 -u ais2023 -P johannesburg

SERVER="192.168.0.13"
TOPIC="AIS2023"
PAYLOAD=b"Welcome to the AIS2023 IoT tutorial"

connect()
print("Connected, starting MQTTClient")
c = MQTTClient("umqtt_client", SERVER,user="ais2023",password="johannesburg")
# c = MQTTClient("umqtt_client", SERVER)
try:
    c.connect()
except:
    print("Cannot connect, please check server IP and username and password")
    sys.exit()

for _ in range(10):
    c.publish(TOPIC,PAYLOAD)
    time.sleep(1)
c.disconnect()
```

ɑ

Subscribing client on the ESP32



```
from machine import Pin
from umqtt.simple import MQTTClient
import network
import time,sys
from wifi_connect import connect

# Test publication e.g. with:
# mosquitto_pub -u ais2023 -P johannesburg -t AIS2023 -m "LED on"

SERVER="192.168.0.13"
TOPIC="AIS2023"

def cmdCallback(topic,payload):
    print(topic,payload)
    if payload == b"LED on":
        userLed.on()
    elif payload == b"LED off":
        userLed.off()

userLed = Pin(19,Pin.OUT)
# connect to WiFi
connect()

print("Connected, starting MQTTClient")
c = MQTTClient("umqtt_client", SERVER,user="ais2023",password="johannesburg")
try:
    c.connect()
except:
    print("Cannot connect, please check server IP and username and password")
    sys.exit()

c.set_callback(cmdCallback)
c.subscribe(TOPIC)

print("Waiting for messages on topic 'AIS2023' from MQTT broker")
while True:
    c.wait_msg()
```

d

ThingsBoard

Thingsboard is an OpenSource IoT platform

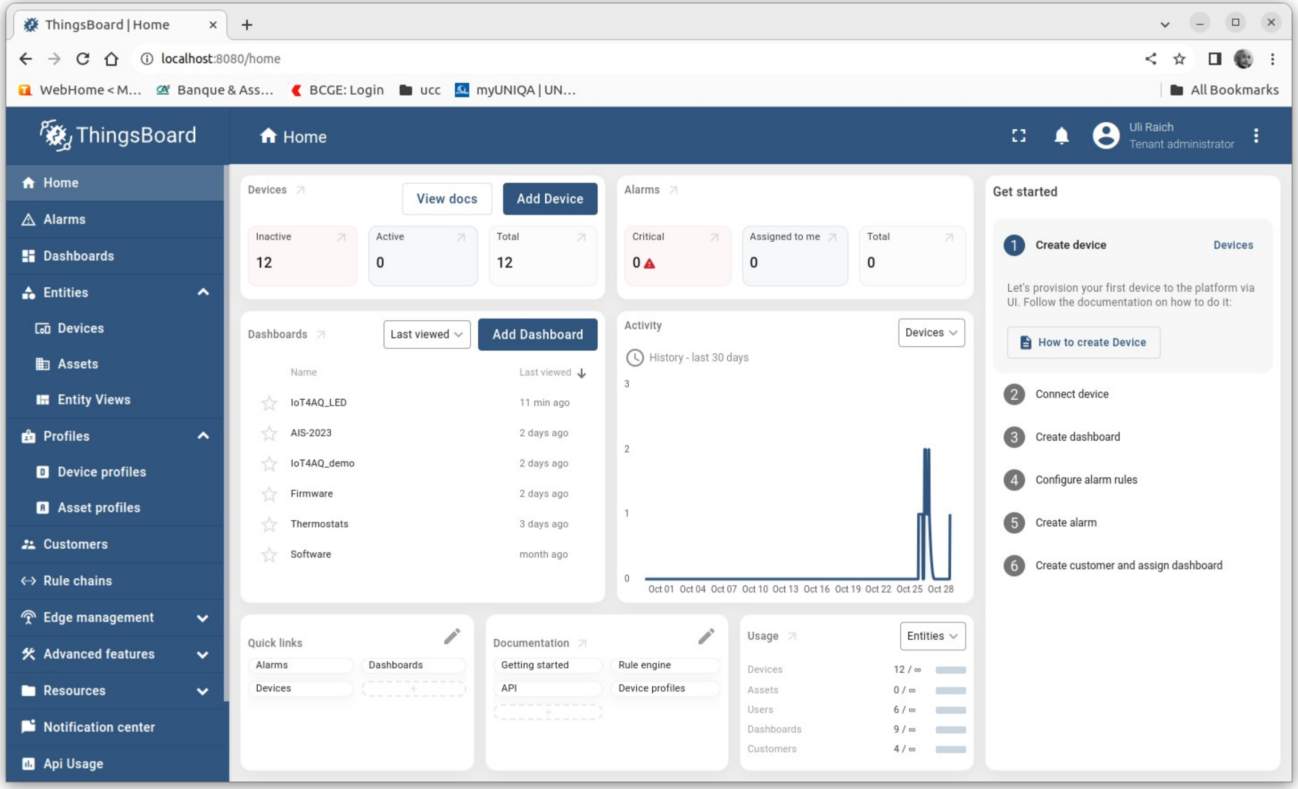
You can send and receive data from it through HTTP or MQTT

It provides the MQTT broker

You can set up a dash board with user interface elements its provides and connect these to your sensors and actuators

You can use a server in the cloud or install your own server on your local computer.

ThingsBoard home



Devices

Inactive	Active	Total
12	0	12

Alarms

Critical	Assigned to me	Total
0 ▲	0	0

Dashboards

Name	Last viewed
☆ IoT4AQ_LED	11 min ago
☆ AIS-2023	2 days ago
☆ IoT4AQ_demo	2 days ago
☆ Firmware	2 days ago
☆ Thermostats	3 days ago
☆ Software	month ago

Activity

Get started

1 Create device — Devices

Let's provision your first device to the platform via UI. Follow the documentation on how to do it:

2 Connect device

3 Create dashboard

4 Configure alarm rules

5 Create alarm

6 Create customer and assign dashboard

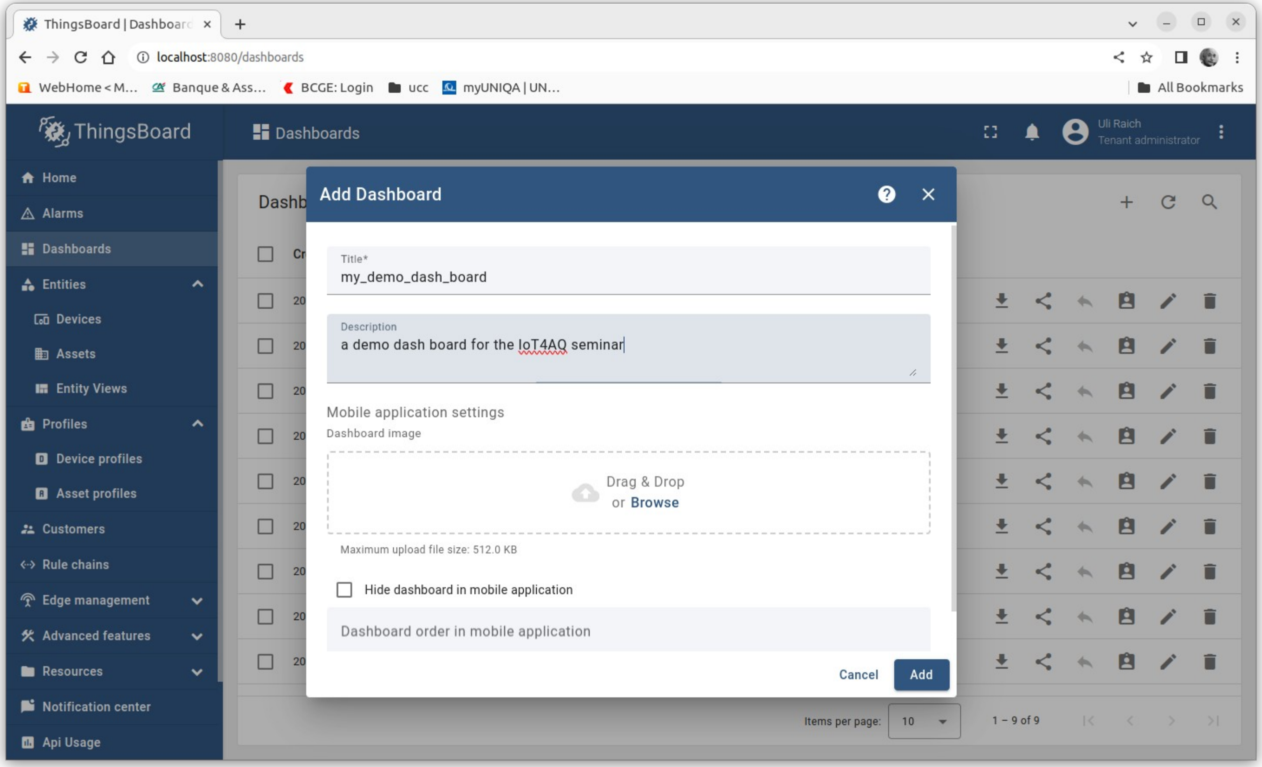
Quick links: Alarms, Dashboards, Devices

Documentation: Getting started, Rule engine, API, Device profiles

Usage

Devices	12 / ∞
Assets	0 / ∞
Users	6 / ∞
Dashboards	9 / ∞
Customers	4 / ∞

Setting up the dash board



Add Dashboard

Title*
my_demo_dash_board

Description
a demo dash board for the IoT4AQ seminar

Mobile application settings

Dashboard image

Drag & Drop or Browse

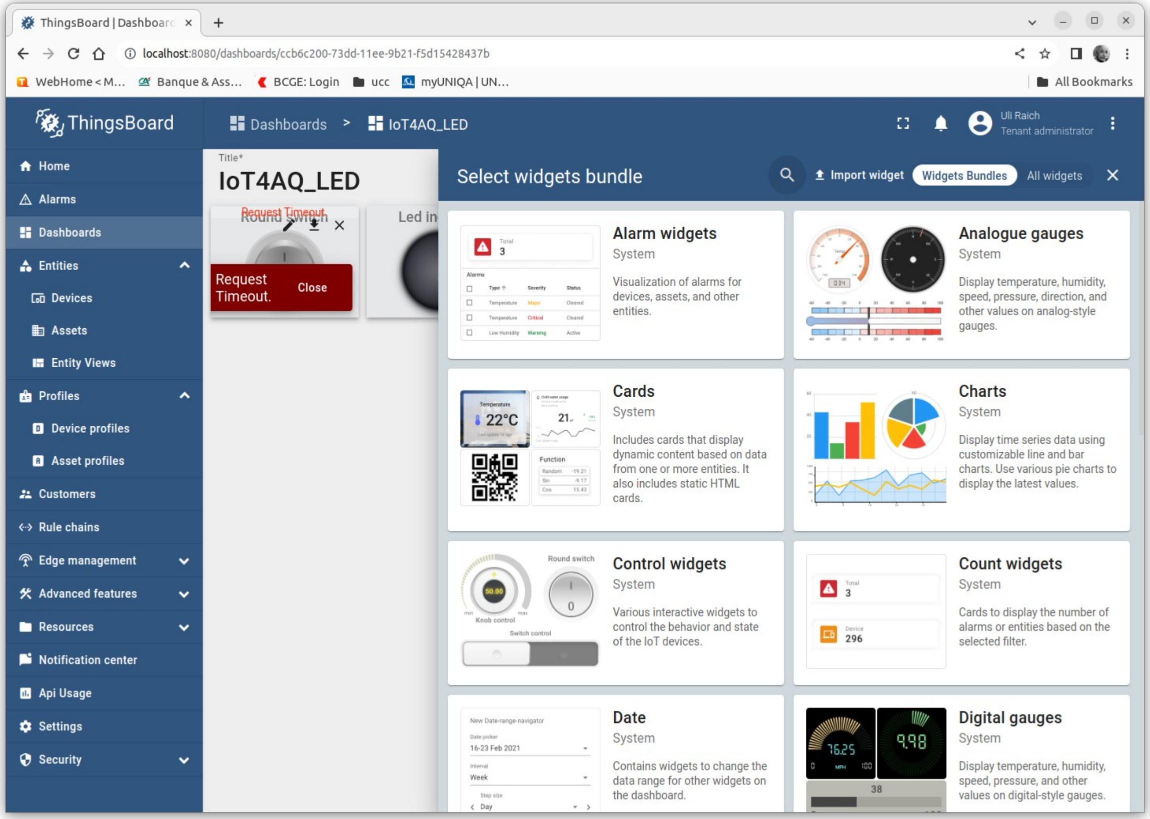
Maximum upload file size: 512.0 KB

□ Hide dashboard in mobile application

Dashboard order in mobile application

Cancel Add

Widgets



Alarm widgets
System
Visualization of alarms for devices, assets, and other entities.

Analogue gauges
System
Display temperature, humidity, speed, pressure, direction, and other values on analog-style gauges.

Cards
System
Includes cards that display dynamic content based on data from one or more entities. It also includes static HTML cards.

Charts
System
Display time series data using customizable line and bar charts. Use various pie charts to display the latest values.

Control widgets
System
Various interactive widgets to control the behavior and state of the IoT devices.

Count widgets
System
Cards to display the number of alarms or entities based on the selected filter.

Date
System
Contains widgets to change the data range for other widgets on the dashboard.

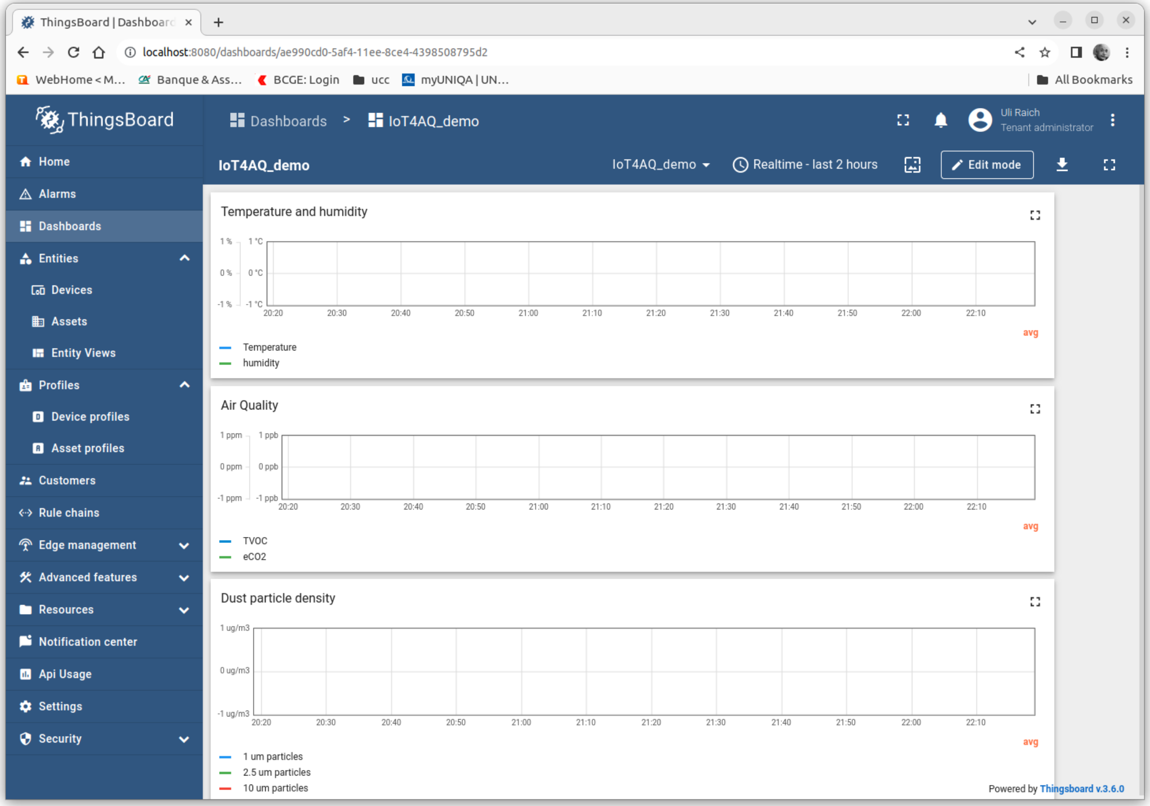
Digital gauges
System
Display temperature, humidity, speed, pressure, and other values on digital-style gauges.

IoT4AQ dashboard

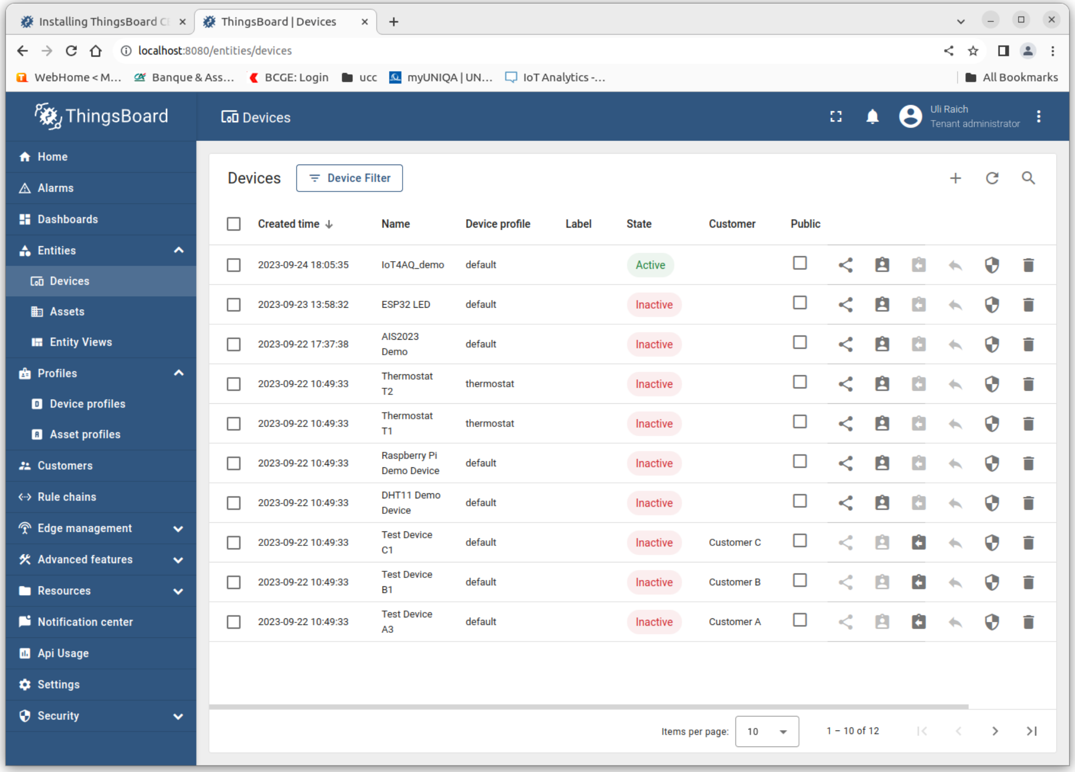
The demo dashboard for the Seminar contains only timeline graphs

- Temperature and humidity
- eCO_2 and TVOC
- Dust concentration

All of the devices are acquisition only



Creating a new device

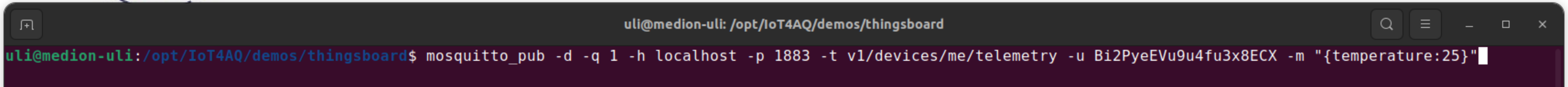


Devices

Created time	Name	Device profile	Label	State	Customer	Public
2023-09-24 18:05:35	IoT4AQ_demo	default		Active		□
2023-09-23 13:58:32	ESP32 LED	default		Inactive		□
2023-09-22 17:37:38	AIS2023 Demo	default		Inactive		□
2023-09-22 10:49:33	Thermostat T2	thermostat		Inactive		□
2023-09-22 10:49:33	Thermostat T1	thermostat		Inactive		□
2023-09-22 10:49:33	Raspberry Pi Demo Device	default		Inactive		□
2023-09-22 10:49:33	DHT11 Demo Device	default		Inactive		□
2023-09-22 10:49:33	Test Device C1	default		Inactive	Customer C	□
2023-09-22 10:49:33	Test Device B1	default		Inactive	Customer B	□
2023-09-22 10:49:33	Test Device A3	default		Inactive	Customer A	□

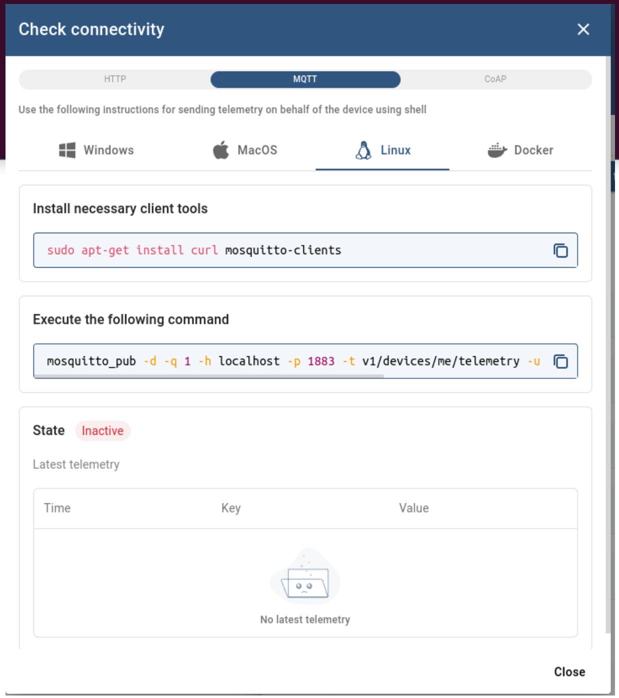
Items per page: 10 1 – 10 of 12

Accessing the device



uli@medion-uli: /opt/IoT4AQ/demos/thingsboard

```
uli@medion-uli:/opt/IoT4AQ/demos/thingsboard$ mosquitto_pub -d -q 1 -h localhost -p 1883 -t v1/devices/me/telemetry -u Bi2PyeEVu9u4fu3x8ECX -m "{temperature:25}"
```



Check connectivity

Use the following instructions for sending telemetry on behalf of the device using shell

Install necessary client tools

```
sudo apt-get install curl mosquitto-clients
```

Execute the following command

```
mosquitto_pub -d -q 1 -h localhost -p 1883 -t v1/devices/me/telemetry -u
```

State Inactive

Latest telemetry

Time	Key	Value

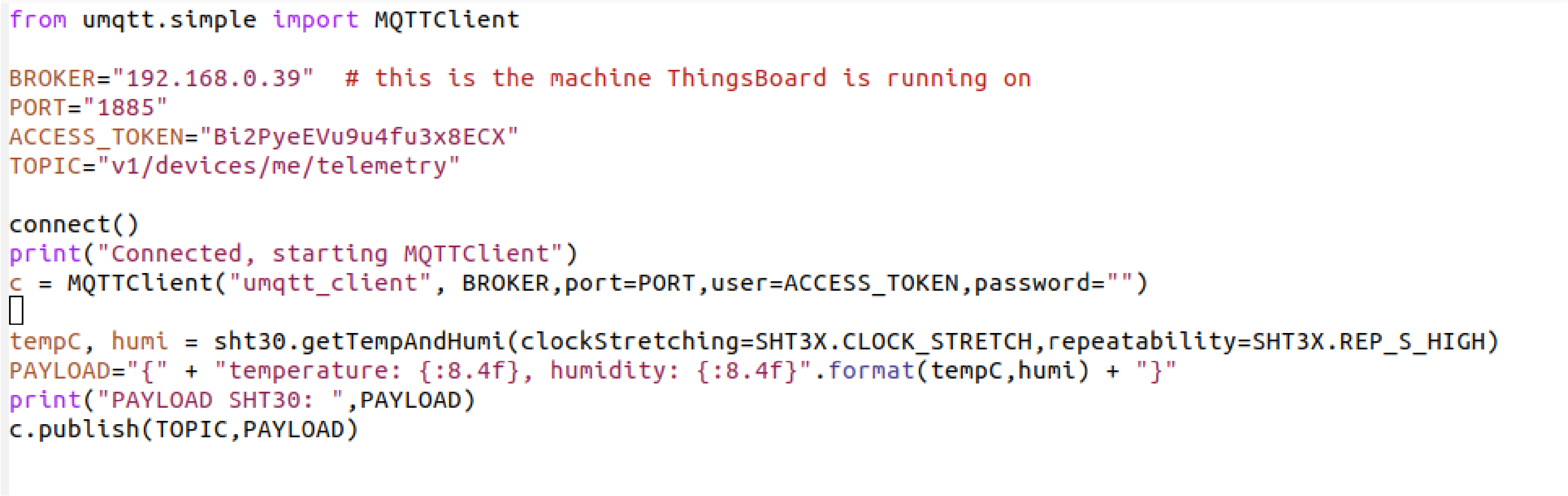
No latest telemetry

Now we know how to update widgets on the dashboard.

Since we already know how to create MQTT messages on the ESP32 we can also send measurement data to the dashboard

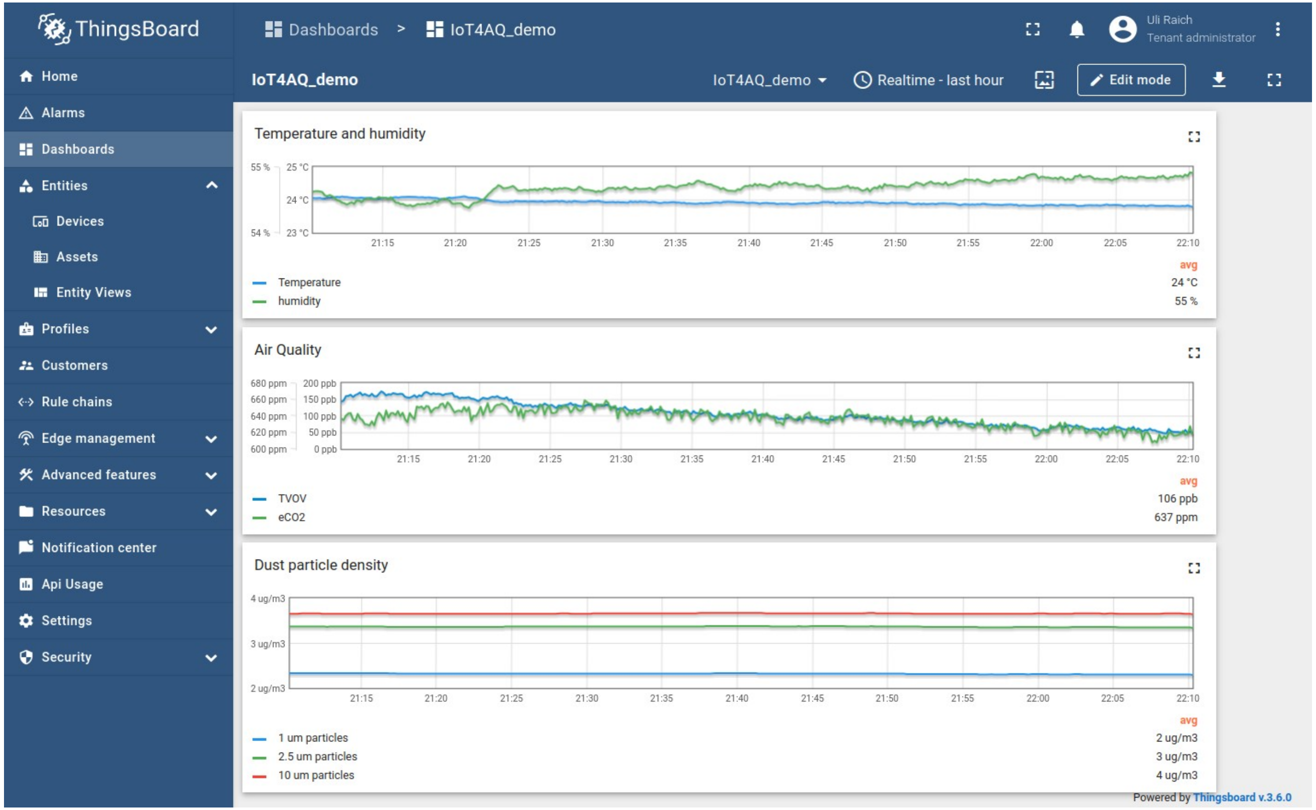
Code snippets

Code snippets from the ESP32 program sending data to the dashboard



```
from umqtt.simple import MQTTClient

BROKER="192.168.0.39"  # this is the machine ThingsBoard is running on
PORT="1885"
ACCESS_TOKEN="Bi2PyeEVu9u4fu3x8ECX"
TOPIC="v1/devices/me/telemetry"

connect()
print("Connected, starting MQTTClient")
c = MQTTClient("umqtt_client", BROKER,port=PORT,user=ACCESS_TOKEN,password="")
□
tempC, humi = sht30.getTempAndHumi(clockStretching=SHT3X.CLOCK_STRETCH,repeatability=SHT3X.REP_S_HIGH)
PAYLOAD="{" + "temperature: {:8.4f}, humidity: {:8.4f}".format(tempC,humi) + "}"
print("PAYLOAD SHT30: ",PAYLOAD)
c.publish(TOPIC,PAYLOAD)
```

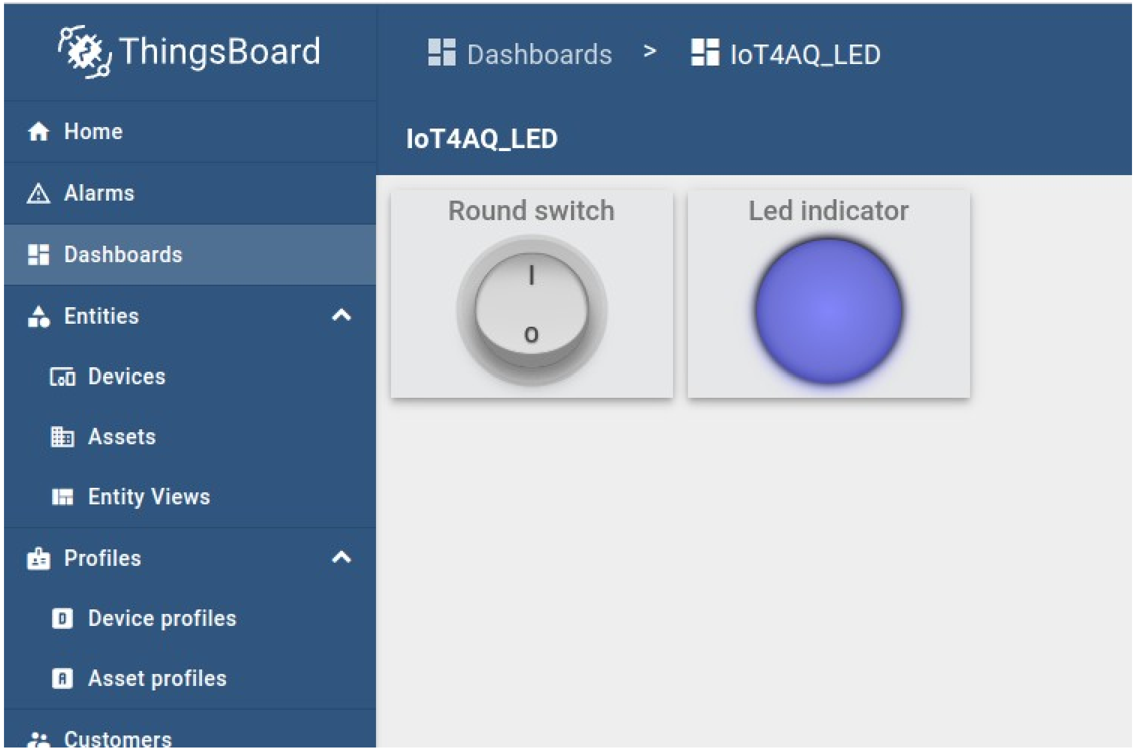
The IoT4AQ dashboard



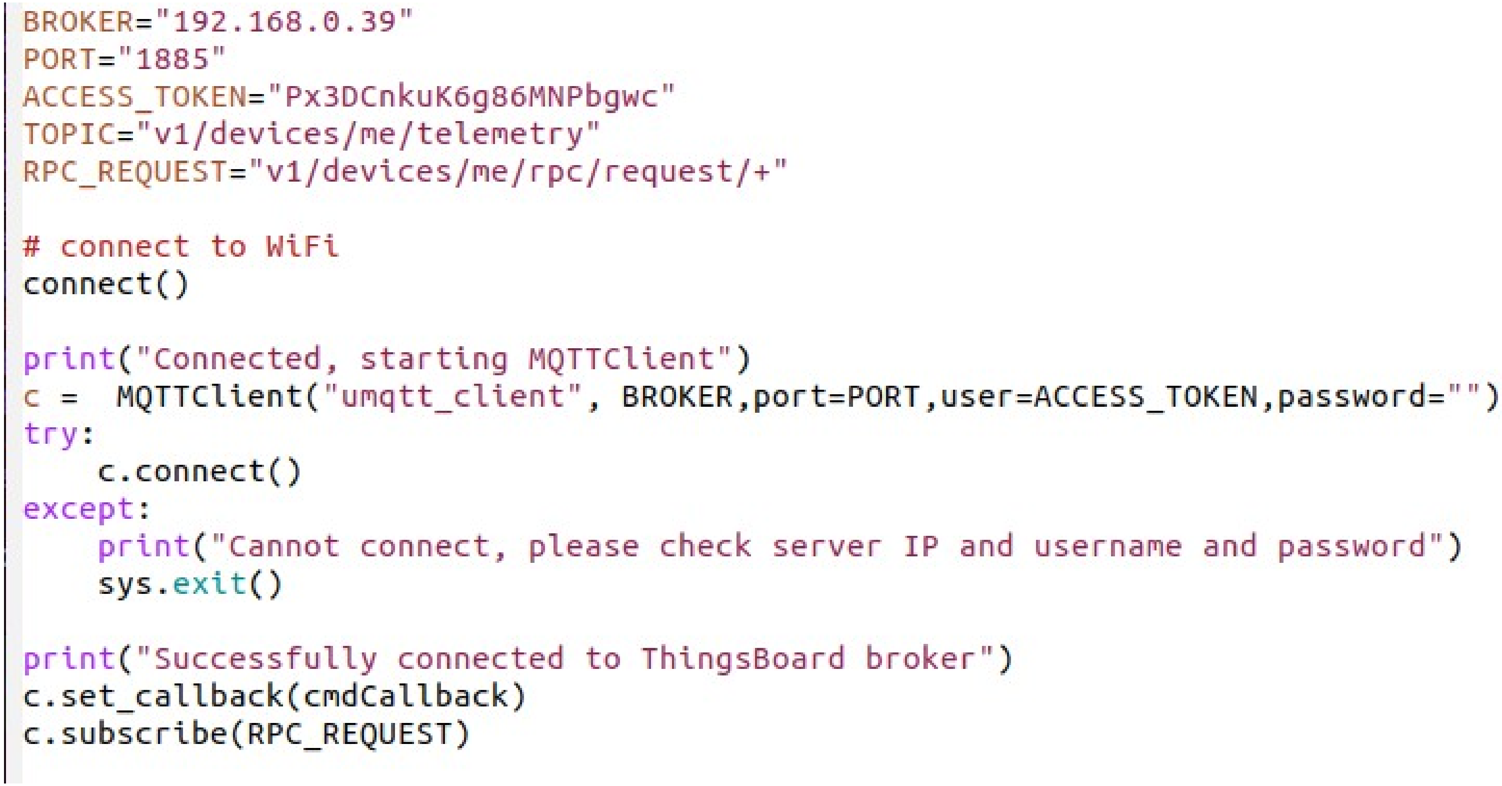
Controlling a device

The switch controls the
user led on the ESP32 CPU board

The LED indicator shows the
current state.

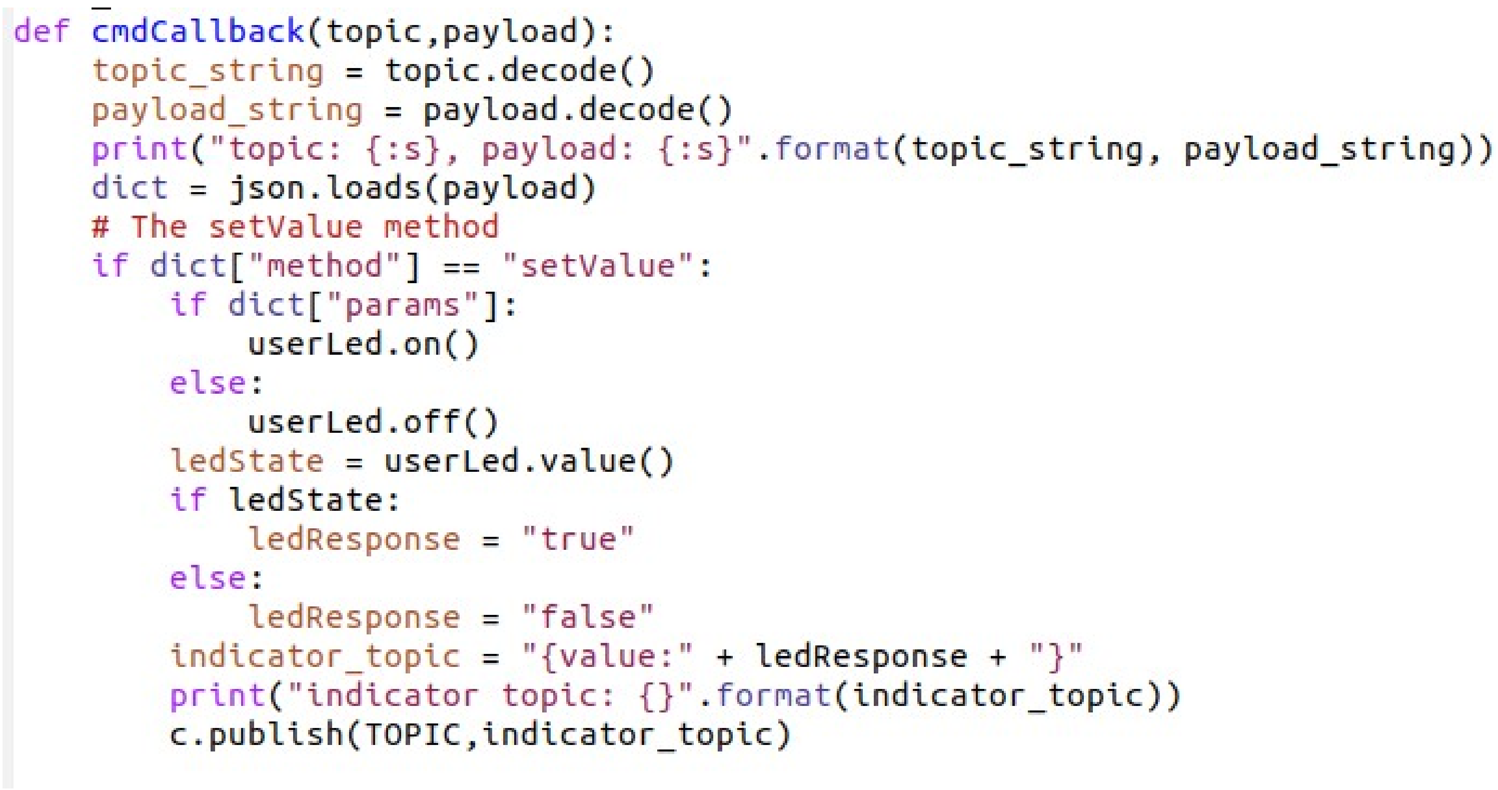


Create a MQTT subscriber



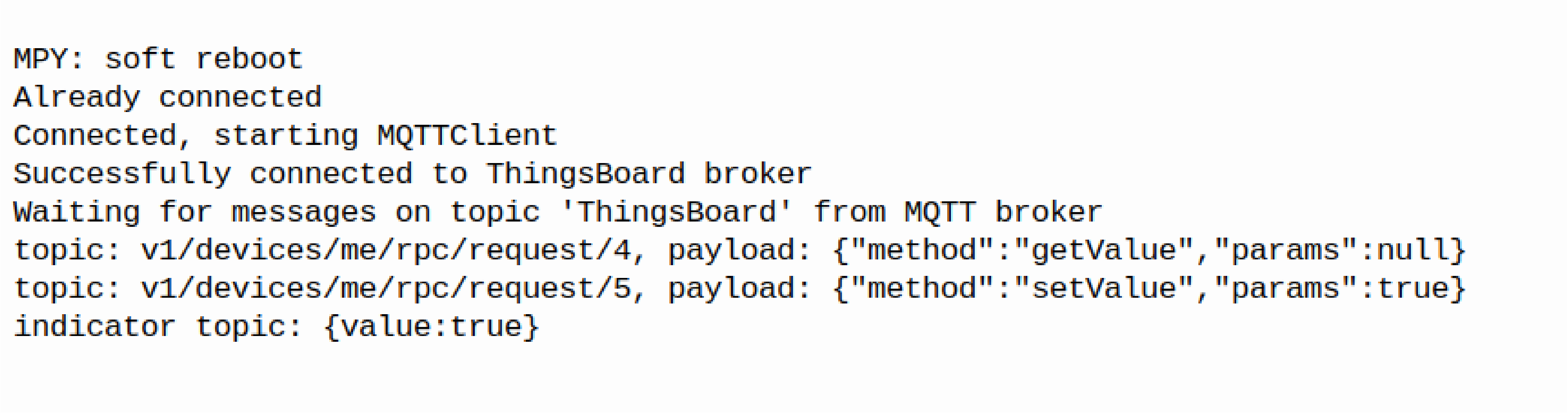
```
BROKER="192.168.0.39"
PORT="1885"
ACCESS_TOKEN="Px3DCnkuK6g86MNPbgwc"
TOPIC="v1/devices/me/telemetry"
RPC_REQUEST="v1/devices/me/rpc/request/+"

# connect to WiFi
connect()

print("Connected, starting MQTTClient")
c =  MQTTClient("umqtt_client", BROKER,port=PORT,user=ACCESS_TOKEN,password="")
try:
    c.connect()
except:
    print("Cannot connect, please check server IP and username and password")
    sys.exit()

print("Successfully connected to ThingsBoard broker")
c.set_callback(cmdCallback)
c.subscribe(RPC_REQUEST)
```

The subscriber callback



```
def cmdCallback(topic,payload):
    topic_string = topic.decode()
    payload_string = payload.decode()
    print("topic: {:s}, payload: {:s}".format(topic_string, payload_string))
    dict = json.loads(payload)
    # The setValue method
    if dict["method"] == "setValue":
        if dict["params"]:
            userLed.on()
        else:
            userLed.off()
        ledState = userLed.value()
        if ledState:
            ledResponse = "true"
        else:
            ledResponse = "false"
        indicator_topic = "{value:" + ledResponse + "}"
        print("indicator topic: {}".format(indicator_topic))
        c.publish(TOPIC,indicator_topic)
```

Log from ESP32 MQTT subscriber



```
MPY: soft reboot
Already connected
Connected, starting MQTTClient
Successfully connected to ThingsBoard broker
Waiting for messages on topic 'ThingsBoard' from MQTT broker
topic: v1/devices/me/rpc/request/4, payload: {"method":"getValue","params":null}
topic: v1/devices/me/rpc/request/5, payload: {"method":"setValue","params":true}
indicator topic: {value:true}
```

LOONEY TUNES
Thats all Folks