

SORBONNE UNIVERSITÉ - SCIENCES ET INGÉNIERIE



MASTER SYSTÈMES ÉLECTRONIQUES
SYSTÈMES INFORMATIQUES (SESI)

UE : IOC

Rapport du projet

Plateforme IOC

Camelia **BOURAS** 21216061

Année universitaire 2023/2024

Présentation générale du projet

Le but de ce projet est de mettre en oeuvre les connaissances qu'on a acquis durant les TP's du module.

On veut réaliser une plateforme qui va nous permettre de communiquer avec les capteurs sur une esp32. Il s'agit de créer un site web dynamique, accessible depuis un client sur le réseau Internet, qui présente des données collectées à partir de capteurs distants. Le serveur web fonctionne sur une station de base composée d'une Raspberry Pi. Cette station agit comme une passerelle entre le réseau Internet et le réseau de capteurs connectés via des protocoles radio sans fil. Le serveur web intègre un serveur HTTP, une application gateway pour accéder aux capteurs, et une base de données. Les capteurs sont installés sur des modules basés sur des microcontrôleurs.

Pour faire la connexion entre le serveur web et l'ESP32 on va utiliser un broker MQTT.

Le diagramme ci-dessous présente l'architecture matérielle de la plateforme et qui sera détaillé par la suite le long de ce rapport.

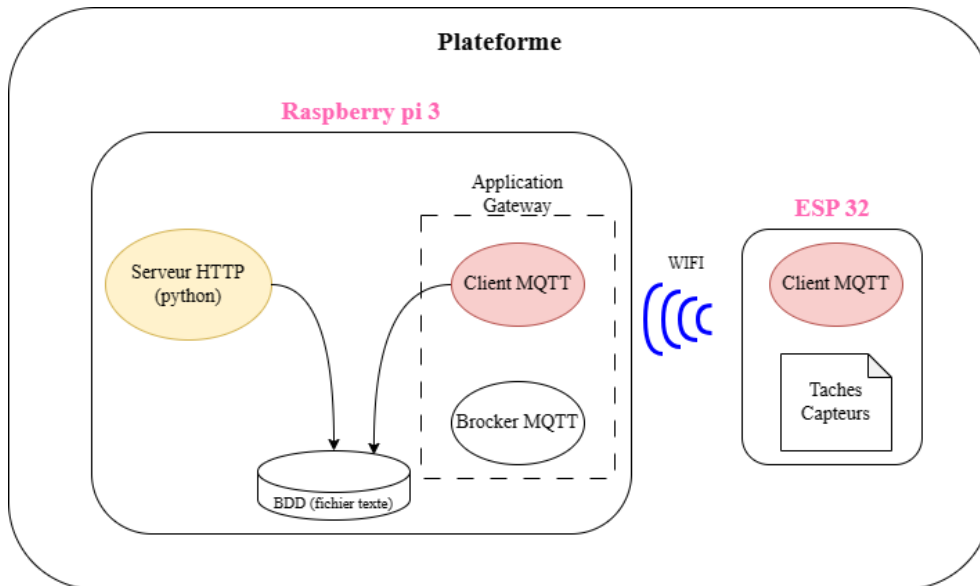


FIGURE 1 – Architecture matérielle de la plateforme

I/ ESP32 et communication avec la RaspberryPI

Dans le fichier de communication :

il faut commencer par inclure ce qui est important :

PubSubClient.h pour la communication avec un serveur MQTT

wifi.h bibliothèque de gestion de communication wifi.

```
#include <PubSubClient.h>
#include <SPI.h>
#include <WiFi.h>
```

Ensuite il faut faire rentrer les paramètres du réseau wifi .

Ce code assure la bonne connexion au wifi et au bon broker.

L'ESP32 agit comme un client MQTT producteur qui envoie des messages au broker MQTT via la connexion WiFi.

On étudie les entrées GPIO de l'esp32 :

La photoresistance : on mesure l'éclairage en pourcentage :

```
int luminosity = map(analogRead(photoresistorPin), 0, 4095, 0,
100);
```

Et la même chose pour l'état du bouton.

Ca se fait dans la loop où notre client MQTT producteur publie ces deux valeurs accompagnées de leur topic.

Le broker à la réception a pour but de traiter ces valeurs et les distribuer selon leur topic.

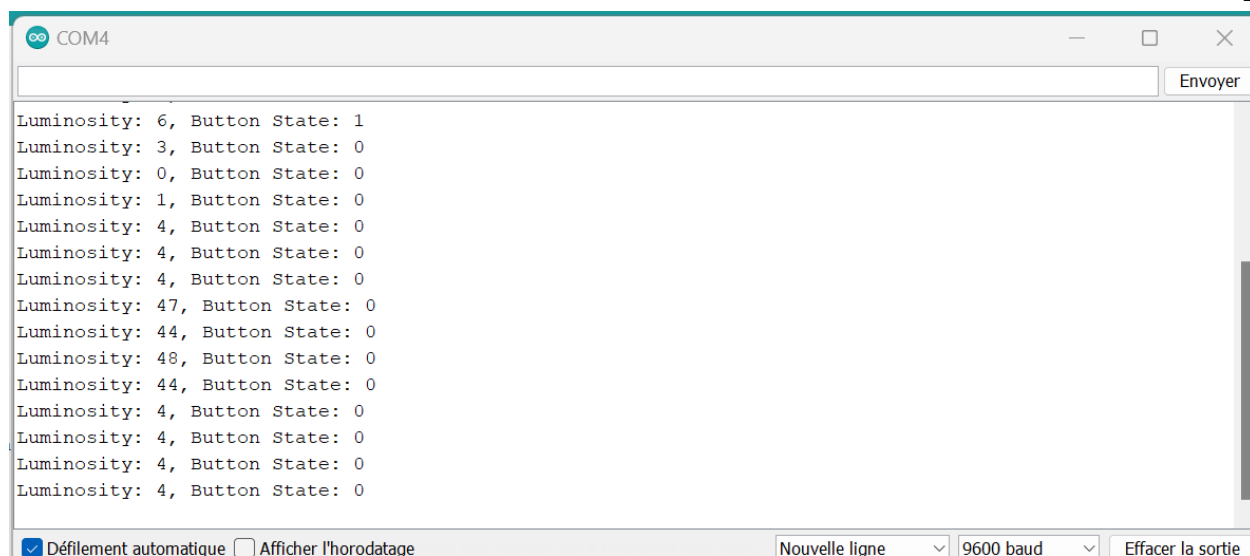


FIGURE 2 – Serial test du ESP32

Dans cette partie je me suis trompé de broche car j'ai inversé le schema de la carte esp32 dans ma tete :)

II/ Raspberry Pi3

La connexion à la Raspberry : Pour activer le protocole SSH sur la RaspberryPi3 j'ai branché la carte SD au PC afin de rajouter un fichier **ssh** sans extension vide que j'ai crée.

J'ai branché la carte RaspberryPi au PC avec un câble Ethernet pour pouvoir communiquer avec elle via réseau local car elle n'était pas encore connectée au wifi.

On utilise PUTTY pour se connecter à la carte via SSH avec comme hostname : "raspberrypi.local"

Pour le nom d'utilisateur et le mot de passe j'ai utilisé respectivement : "pi" et "raspberrypi".

Pour connecter la carte au wifi :

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Et on rajoute les coordonnées du réseau wifi dans un paragraphe Network avec les deux champs **ssid** et **psk**.

Pour vérifier la connexion :

```
ifconfig
```

Et on retrouve bien le wlan0 avec l'adresse IP : 192.168.1.19 qu'on va utiliser from now on pour se connecter via ssh(PUTTY).

PuTTY est un logiciel client de connexion à distance gratuit principalement utilisé pour établir des connexions sécurisées entre un poste local et un serveur distant via les protocoles SSH (Secure Shell),

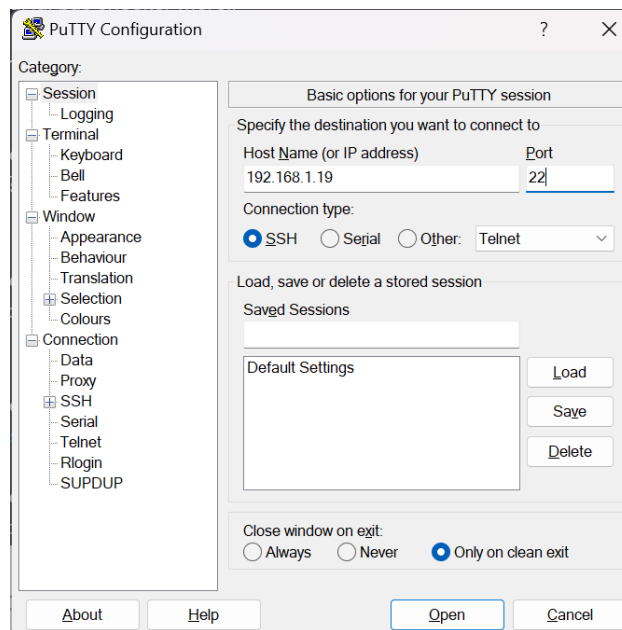


FIGURE 3 – Interface PUTTY

g

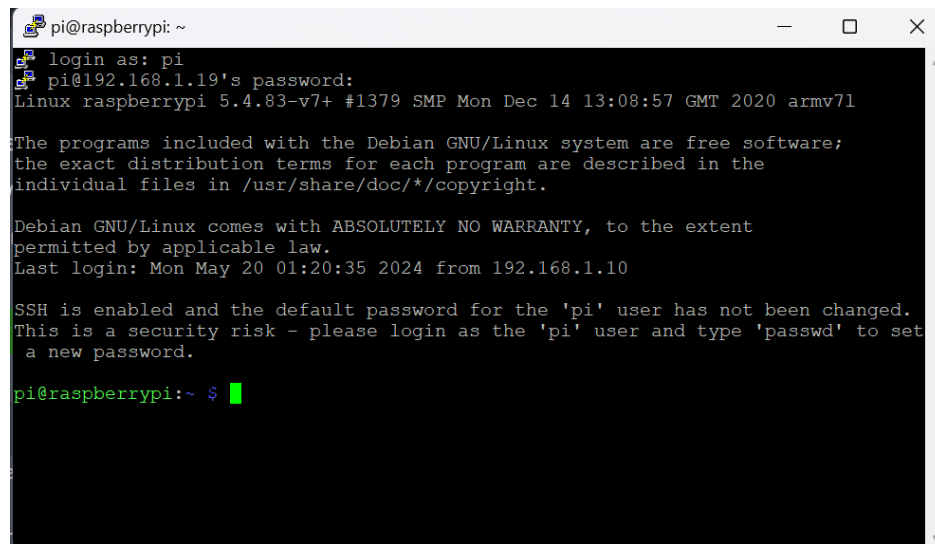


FIGURE 4 – Connexion PUTTY

a) Protocole MQTT et broker MQTT

Le protocole MQTT est un de plusieurs protocoles de communication basées sur le concept de deux types de clients : Publisher (Producer) , Subscriber(Consumer).

Dans notre cas le publisher est l'ESP32 et le subscriber est le serveur.

Afin de relier entre les publishers et subscribers on retrouve le MQTT Broker qui a pour rôle de traiter et filtrer les messages qu'il reçoit de la part des publishers et les directionne vers le subscribers correspondant selon le TOPIC à lequel ce subscriber a demandé.

Dans la figure suivante on retrouve le schéma général d'une connexion MQTT.

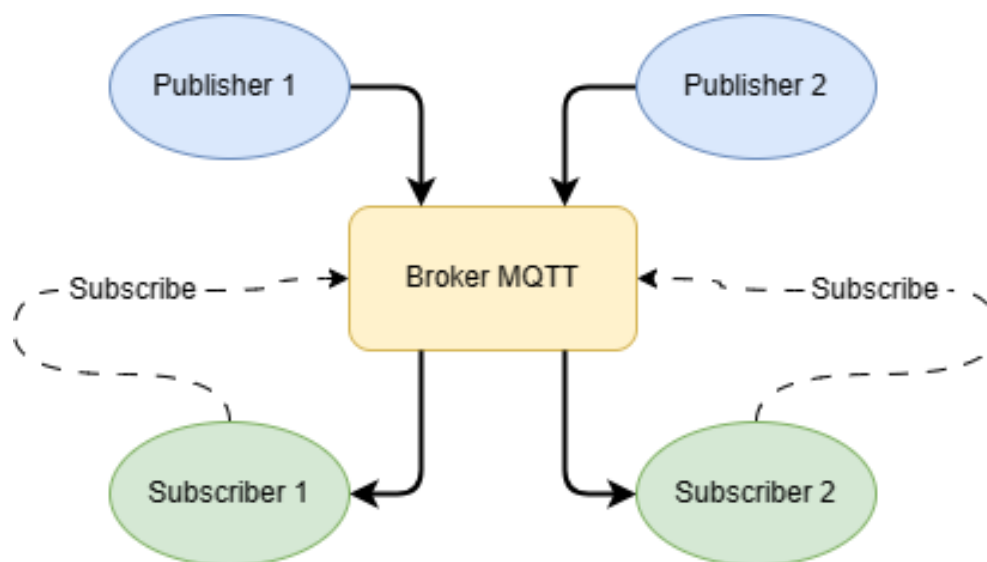


FIGURE 5 – Concepte du protocole MQTT

Broker MQTT

Pour créer un broker il faut installer Mosquitto

```
sudo apt install mosquitto  
sudo apt install mosquitto-clients
```

En essayant d'accéder /mosquitto.conf pour la configuration

```
sudo nano /etc/mosquitto/mosquitto.conf
```

mosquitto.conf : C'est le fichier de configuration principal du broker MQTT Mosquitto. C'est dans ce fichier qu'on peut spécifier la configuration principale du broker, telle que le port d'écoute, les autorisations d'accès.

on modifie avec un editeur de texte :

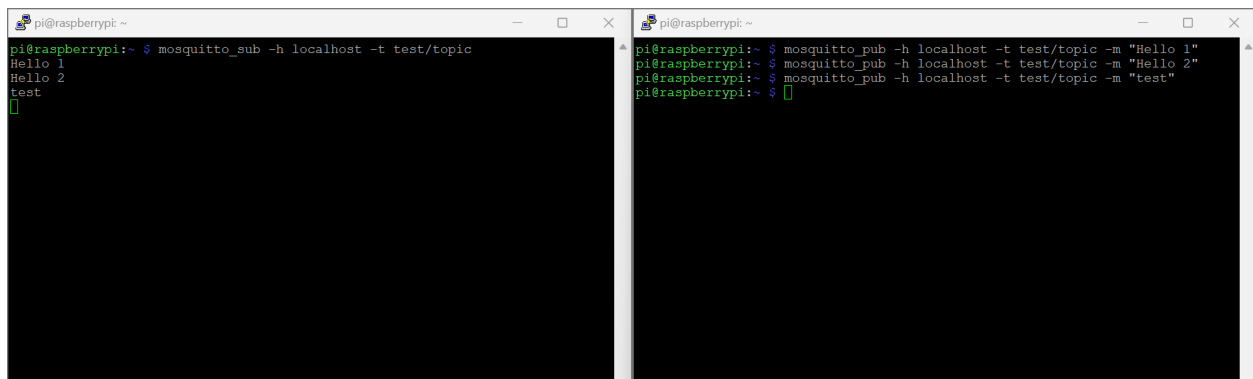
Port 1883 et Autorisation d'entrée

```
allow_anonymous true
listener 1883
```

Et on lance Mosquitto :

```
sudo systemctl start mosquitto
```

Pour tester le bon fonctionnement du broker j'ai lancé quelques tests :



```
pi@raspberrypi:~$ mosquitto_sub -h localhost -t test/topic
Hello 1
Hello 2
test
[]

pi@raspberrypi:~$ mosquitto_pub -h localhost -t test/topic -m "Hello 1"
pi@raspberrypi:~$ mosquitto_pub -h localhost -t test/topic -m "Hello 2"
pi@raspberrypi:~$ mosquitto_pub -h localhost -t test/topic -m "test"
```

FIGURE 6 – Test broker

b) Subscriber

Dans un fichier .py on cree notre client MQTT qui est un client subscriber connecté au broker MQTT selon les deux topics **lumiere** et **bouton**.

Ce Subscriber si il reçoit le message => selon le topic qu'in suit il enregistre les valeur de l'eclairage et l'etat du bouton dans la bases de données : dans mon cas il s'agit d'un fichier texte **DATA_BASE = "../BDDtoWEB.txt"**.

```
#création de client: subscriber
client = mqtt.Client()

#assigner les fonctions à notre subscriber(raspberry pi3)
client.on_connect = on_connect
```



```

client.on_message = on_message

#connecter le subscriber au broker
client.connect(MQTT_BROKER, MQTT_PORT)

#boucle infinie
client.loop_forever() # Keep the client running

```

Durant cette étape : j'ai créé un répertoire sur raspberry et installer la bibliothèque mqtt :

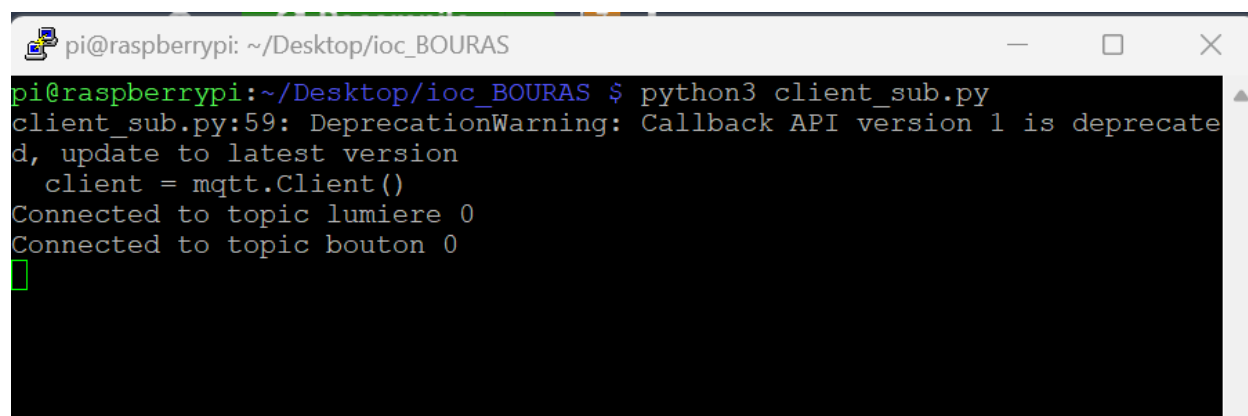
```

pip3 install paho-mqtt
pip3 install typing_extensions

```

Et on exécute le script python qu'on a copié dans le répertoire ioc_bouras :

```
python3 client_sub.py
```



```

pi@raspberrypi: ~/Desktop/ioc_BOURAS
pi@raspberrypi:~/Desktop/ioc_BOURAS $ python3 client_sub.py
client_sub.py:59: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client()
Connected to topic lumiere 0
Connected to topic bouton 0
█

```

FIGURE 7 – Premier test sans brancher l'esp32 on remarque la connexion

En guise de test : Après avoir ouvert un nouveau terminal PUTTY :

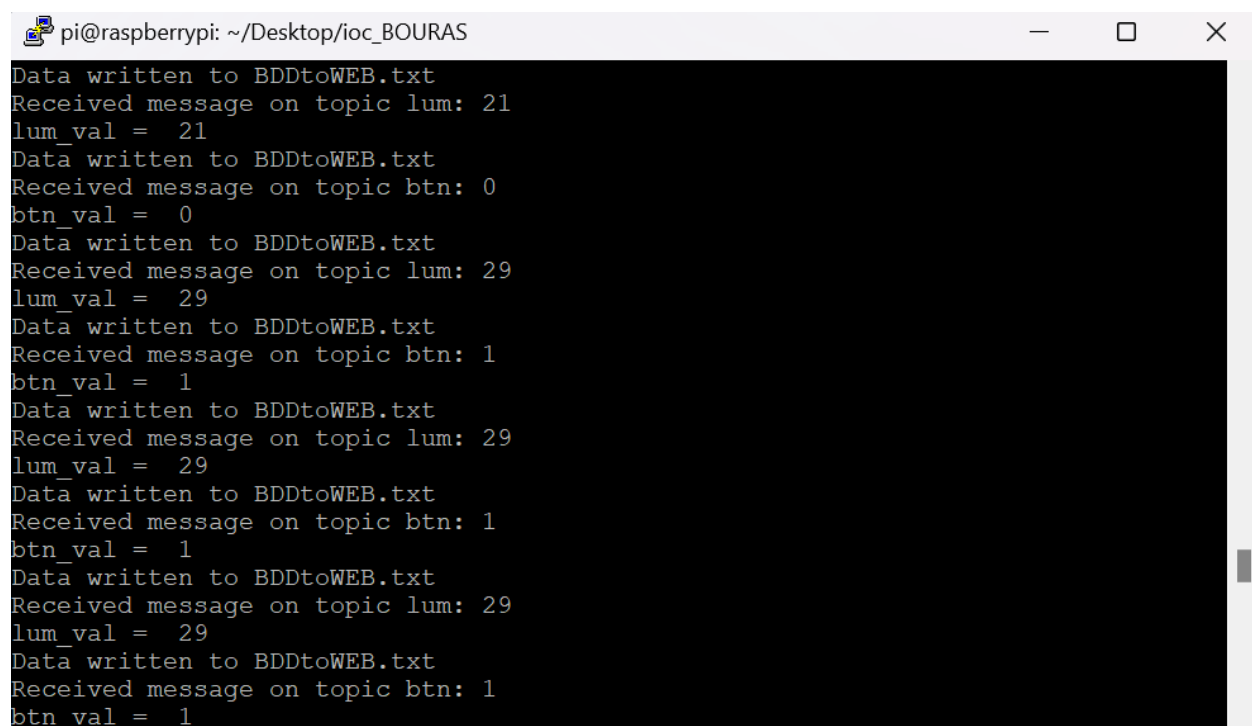
```
mosquitto_sub -h localhost -t "lum" -t "btn"
```

On trouve que ça donne bien les bonnes valeurs.

Après avoir branché l'esp32 et lu les valeurs sur la serial du ArduinoIDE j'ai pu visualiser ce que un client subscriber mosquitto pourrait visualiser et ça correspondait bien aux valeurs de l'esp32.

On relance notre serveur mqtt sur raspberryPi et on obtient les mêmes valeurs qu'on retrouve sur l'esp32.

```
python3 client_sub.py
```



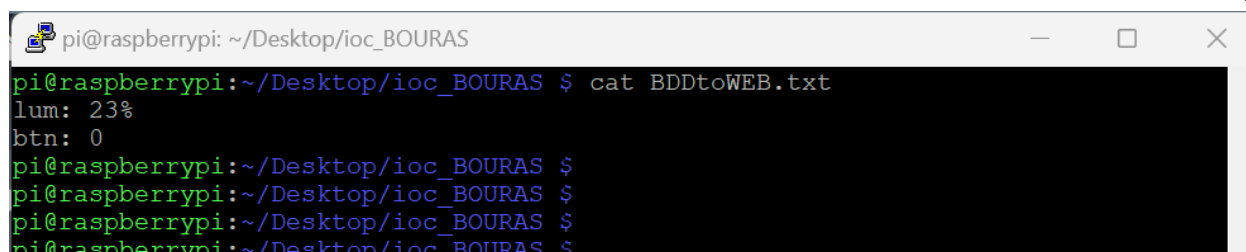
The screenshot shows a terminal window titled "pi@raspberrypi: ~/Desktop/ioc_BOURAS". The terminal output displays a sequence of MQTT messages being received on two topics, "lum" and "btn", and the corresponding values being written to a file named "BDDtoWEB.txt". The messages are as follows:

```
Data written to BDDtoWEB.txt
Received message on topic lum: 21
lum_val = 21
Data written to BDDtoWEB.txt
Received message on topic btn: 0
btn_val = 0
Data written to BDDtoWEB.txt
Received message on topic lum: 29
lum_val = 29
Data written to BDDtoWEB.txt
Received message on topic btn: 1
btn_val = 1
Data written to BDDtoWEB.txt
Received message on topic lum: 29
lum_val = 29
Data written to BDDtoWEB.txt
Received message on topic btn: 1
btn_val = 1
Data written to BDDtoWEB.txt
Received message on topic lum: 29
lum_val = 29
Data written to BDDtoWEB.txt
Received message on topic btn: 1
btn_val = 1
```

FIGURE 8 – Les données récupérées

Dans cette partie j'ai fait une faute de nommer les topics différemment pour le fichier python et celui du esp32, donc je l'ai corrigé apres un loooong debug :)

Pour le fichier text : "BDDtoWEB.txt" on retrouve bien les bonnes valeurs qui sont mises à jour à chaque nouvelle mesure sur l'ESP32 :



```

pi@raspberrypi: ~/Desktop/ioc_BOURAS
pi@raspberrypi:~/Desktop/ioc_BOURAS $ cat BDDtoWEB.txt
lum: 23%
btn: 0
pi@raspberrypi:~/Desktop/ioc_BOURAS $
pi@raspberrypi:~/Desktop/ioc_BOURAS $
pi@raspberrypi:~/Desktop/ioc_BOURAS $
pi@raspberrypi:~/Desktop/ioc_BOURAS $

```

FIGURE 9 – Contenu du fichier text

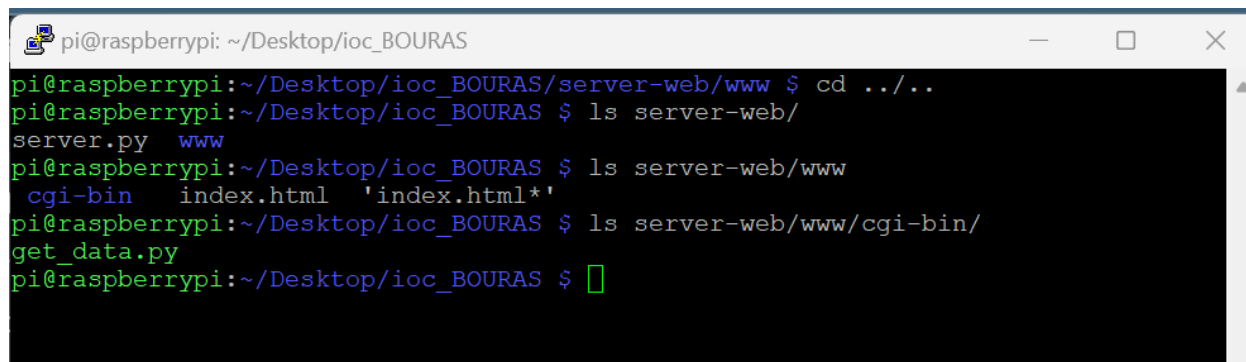
c) Serveur HTTP

Pour le serveur HTTP je me suis basé sur la page par défaut d'Apache2 sur Ubuntu accessible apres l'installation de Apache2. Mais apres avoir amélioré mon projet j'ai décidé d'opter pour le serveur HTTP en python déjà étudié dans le tme5 : Serveur WEB minimaliste.

Après avoir bien compris le concept :

0.0.1 Création serveur du serveur web

En s'inspirant de la structure de fichier sur le tme. J'ai crée l'hierarchie de dossier suivantes :



```

pi@raspberrypi: ~/Desktop/ioc_BOURAS
pi@raspberrypi:~/Desktop/ioc_BOURAS/server-web/www $ cd ../../
pi@raspberrypi:~/Desktop/ioc_BOURAS $ ls server-web/
server.py  www
pi@raspberrypi:~/Desktop/ioc_BOURAS $ ls server-web/www
cgi-bin  index.html  'index.html*'
pi@raspberrypi:~/Desktop/ioc_BOURAS $ ls server-web/www/cgi-bin/
get_data.py
pi@raspberrypi:~/Desktop/ioc_BOURAS $ 

```

FIGURE 10 – Répertoire du serveur HTTP

- **server.py** : pour configurer le serveur HTTP.
- repertoire www qui contient les fichiers de la page WEB.
- **index.html** : code html de la page web qui affiche les valeur lum_val et btn_val.
- repertoire cgi.bin : qui contient les fichiers script dynamiques.
- **get_data.py** : script cgi qui lit le contenu du fichier .txt et les envoie au fichier html.

Dans un navigateur web on tappe dans la bare de recherche :

```
http://192.168.1.19:8000/www/
```

Dans terminal 1 Pour lancer le sub qui reçoit les valeurs et écrit dans le fichier texte :

```
pi@raspberrypi:~/Desktop/ioc_BOURAS $ python3 client_subscriber.py
```

Dans terminal 2 qui lance le serveur http :

```
pi@raspberrypi:~/Desktop/ioc_BOURAS/server-web $ python3 server.py
```

```
pi@raspberrypi:~/Desktop/ioc_BOURAS
pi@raspberrypi:~/Desktop/ioc_BOURAS $ ls
BDDtoWEB.txt client subscriber.py server-web
pi@raspberrypi:~/Desktop/ioc_BOURAS $ python3 client_subscriber.py
client_subscriber.py:61: DeprecationWarning: Callback API version 1 is deprecated, update to latest version
  client = mqtt.Client()
Connected with result code 0
Subscribed to topic lum
Subscribed to topic btn
Received message on topic lum: 35
lum_val = 35
Data written to BDDtoWEB.txt
Received message on topic btn: 0
btn_val = 0
Data written to BDDtoWEB.txt
Received message on topic lum: 35
lum_val = 35
Data written to BDDtoWEB.txt
Received message on topic btn: 0
btn_val = 0
Data written to BDDtoWEB.txt
Received message on topic lum: 35
lum_val = 35
Data written to BDDtoWEB.txt

pi@raspberrypi:~/Desktop/ioc_BOURAS/server-web
pi@raspberrypi:~/Desktop/ioc_BOURAS/server-web $ python3 server.py
Serving HTTP on port 8000
192.168.1.12 - - [20/May/2024 13:15:54] "GET /events HTTP/1.1" 200 -
Exception happened during processing of request from ('192.168.1.12', 56777)
Traceback (most recent call last):
  File "/usr/lib/python3.7/socketserver.py", line 316, in _handle_request_noblock
    self.process_request(request, client_address)
  File "/usr/lib/python3.7/socketserver.py", line 347, in process_request
    self.finish_request(request, client_address)
  File "/usr/lib/python3.7/socketserver.py", line 360, in finish_request
    self.RequestHandlerClass(request, client_address, self)
  File "/usr/lib/python3.7/http/server.py", line 646, in __init__
    super().__init__(*args, **kwargs)
  File "/usr/lib/python3.7/socketserver.py", line 720, in __init__
    self.handle()
  File "/usr/lib/python3.7/http/server.py", line 426, in handle
    self.handle_one_request()
  File "/usr/lib/python3.7/http/server.py", line 414, in handle_one_request
    method()
  File "server.py", line 17, in do_GET
    self.wfile.write(f"data: {json.dumps(data)}\n\n".encode('utf-8'))
  File "/usr/lib/python3.7/socketserver.py", line 799, in write
```

FIGURE 11 – Terminaux

Sur la page web on remarque bien la lecture des données :

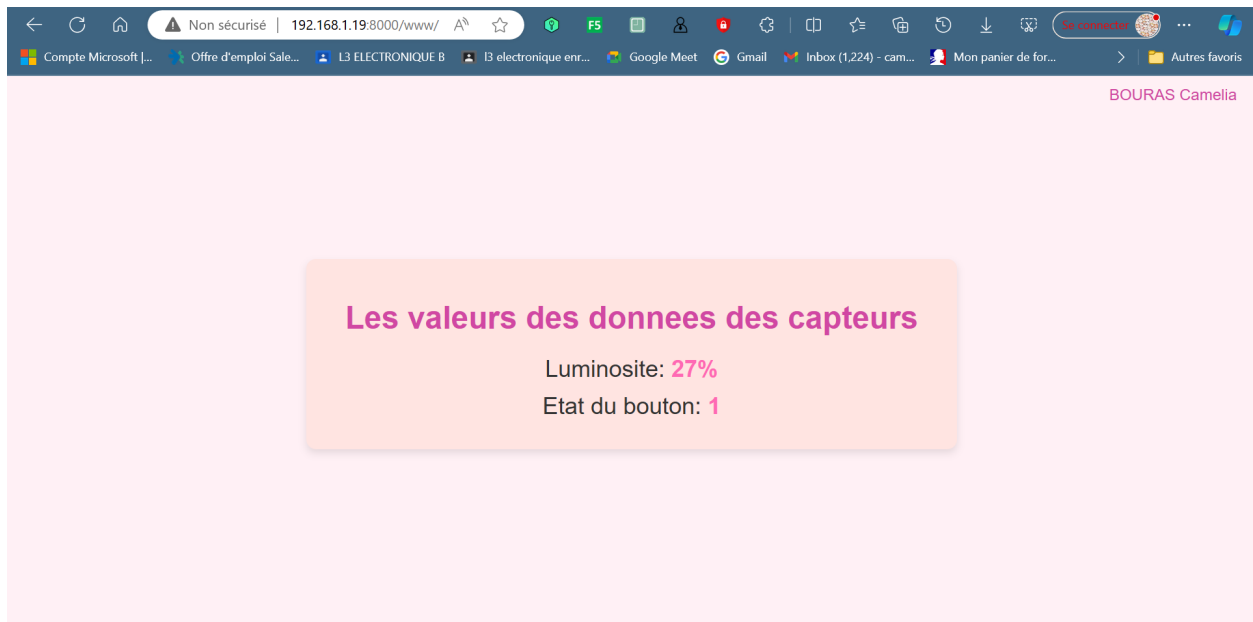


FIGURE 12 – Site WEB diffusant les données des capteurs

Il faut noter qu'on peut pas refresh la page WEB vu que cette derniere le fait automatiquement pour recevoir les données chaque 1 seconde (selon la mesure de l'esp32). :)

1 Références

- Download PUTTY : <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>
- Video youtube pour connecter la raspberrypi au wifi : <https://www.youtube.com/watch?v=VxeUkd0isLI>