

# Title : IOC\_T07 - Mini Projet IOC

---

M1 SESI

Binôme : Kavish RAGHUBAR et Weiyi GONG

Lors de ce projet, nous allons utiliser :

- 1 Raspberry Pi 3 (qui va gérer la partie web : server.js)
- 1 ESP32 (buzzer, capteur de luminosité, bouton poussoir)

MQTT: MOSQUITTO dans ESP32

MQTT (Message Queuing Telemetry Transport) est un protocole de messagerie légère et rapide conçu pour les réseaux de capteurs et les appareils IoT. Mosquitto est un broker MQTT open-source qui implémente le protocole MQTT.

Lorsqu'un client MQTT souhaite publier un message sur un sujet particulier, il se connecte au broker Mosquitto et publie le message avec un sujet associé. Le broker Mosquitto reçoit ensuite le message et le distribue à tous les clients qui se sont abonnés à ce sujet.

De même, lorsqu'un client MQTT souhaite recevoir des messages sur un sujet particulier, il se connecte également au broker Mosquitto et s'abonne à ce sujet. Le broker Mosquitto stocke ensuite les messages correspondants et les transmet au client lorsqu'ils sont disponibles.

Mosquitto est facile à installer et à configurer et peut être utilisé pour créer des systèmes de messagerie IoT efficaces et évolutifs. Il est également compatible avec de nombreux langages de programmation et de nombreux clients MQTT open-source sont disponibles pour une utilisation avec Mosquitto.

Afin de connecter l'ESP32 au raspberry pi, nous devons télécharger et installer MOSQUITTO sur le raspberry pour compléter la communication entre l'ESP32 et le raspberry.

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

```
mosquitto -v
```

```
sudo systemctl enable mosquitto.service
```

Après avoir utilisé les commandes entrées dans le terminal ci-dessus, nous pouvons terminer le téléchargement et la vérification de mosquitto sur le raspberry pi, cependant, mosquitto sur cette base ne fonctionne toujours pas et nous devons faire quelques changements à la configuration demandée.

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

Et ajoutez le code suivant à la fin.

```
listener 1883
allow_anonymous true
```

Ensuite, nous relançons MOSQUITTO.

```
sudo systemctl restart mosquitto
```

Ensuite, nous pouvons définir quelques paramètres simples pour mosquitto, tels que le nom d'utilisateur et le mot de passe.

```
mosquitto_passwd <password file> <username>
```

Ceci complète l'installation de notre mosquitto, et nous pouvons maintenant envoyer quelques messages simples à notre esp32.

Afin de connecter l'esp32 au mosquitto, nous devons connaître l'adresse IP du mosquitto, nous devons donc entrer la commande suivante :

```
hostname -i
```

Si nous voulons tester mosquitto, nous devons le topicer et exécuter quelques tests

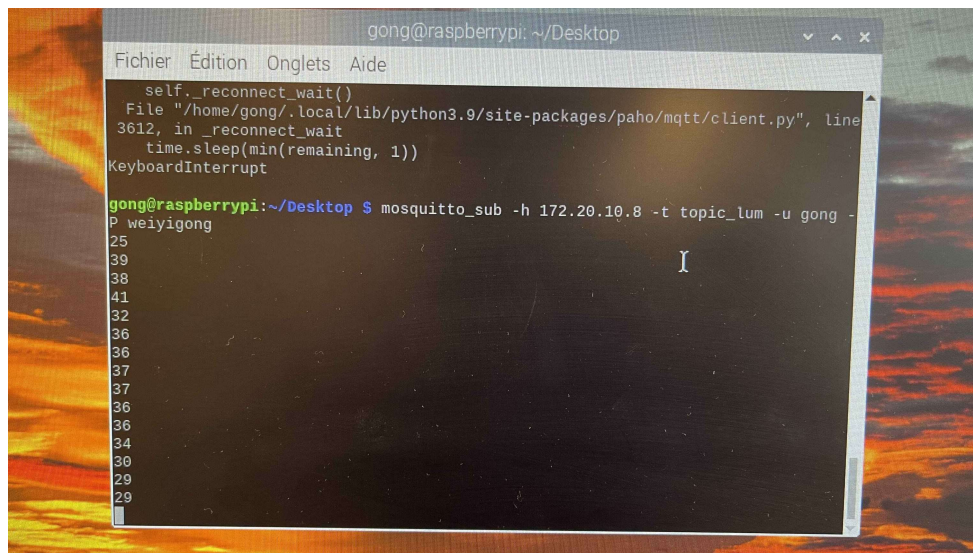
```
sudo apt install -y mosquitto mosquitto-clients
```

```
mosquitto -d
```

```
mosquitto_sub -h 172.20.10.8 -t topic_lum -u gong -P weiyigong
```

ESP32 Arduino luminosité transmission:

Capture d'écran:



```
gong@raspberrypi: ~/Desktop
Fichier  Edition  Onglets  Aide
self._reconnect_wait()
File "/home/gong/.local/lib/python3.9/site-packages/paho/mqtt/client.py", line
3612, in _reconnect_wait
    time.sleep(min(remaining, 1))
KeyboardInterrupt

gong@raspberrypi:~/Desktop $ mosquitto_sub -h 172.20.10.8 -t topic_lum -u gong -
P weiyigong
25
39
38
41
32
36
36
37
37
36
36
34
30
29
29
```

Dans l'esp32, sur la base de notre apprentissage précédent, nous devons activer le capteur photosensible de l'esp32 pour recevoir l'intensité de la lumière et l'envoyer au client mqtt.

A cela nous devons ajouter deux parties importantes, la première est de connecter l'esp32 au wifi et la seconde est de connecter l'esp32 au mqtt.

#### Connection de WIFI:

```
WiFi.begin(ssid,password)

WiFi.localIP()
```

Nous devons connaître le nom et le mot de passe du wifi pour pouvoir nous y connecter et utiliser le second code pour obtenir l'adresse IP de l'esp32 une fois qu'il est connecté au wifi.

#### Connection de MQTT:

```
client.setServer(mqtt_server,1883)
client.connected()
client.state()
```

Lors de la connexion à MQTT, nous devons d'abord définir les informations relatives au client, telles que le numéro de port et le nom, puis nous pouvons nous connecter directement, après quoi nous pouvons utiliser la troisième fonction pour demander l'état de la connexion.

Après cela, nous pouvons exécuter esp32 pour interagir avec mosquitto et nous pouvons voir que les données de luminosité sont constamment rafraîchies sur le port mqtt.

Pour la partie MQTT, nous avons consulté le tutoriel sur le site : <https://www.engineersgarage.com/raspberry-pi-esp32-esp8266-mqtt-iot/> (<https://www.engineersgarage.com/raspberry-pi-esp32-esp8266-mqtt-iot/>) qui détaille très clairement les étapes à suivre.  
test\_mqtt.ino

Dans ce code client, nous allons nous connecter au serveur afin d'envoyer la donnée du capteur de luminosité.

Librairies : DHT, PubSubClient pour MQTT, WiFi

Constantes : SSID, wifiPassword, mqtt\_server ip adresse,...

C'est avec la méthode publish() de la lib PubSubClient qu'on pourra envoyer les données au serveur MQTT. En cas d'échec de l'envoi, le programme (dans loop) va essayer de se reconnecter au serveur et re-envoyer la donnée à nouveau, après avoir attendu 1 min.

Transmission de MQTT MOSQUITTO vers Web:

Nous utilisons python pour écrire dans le base données.

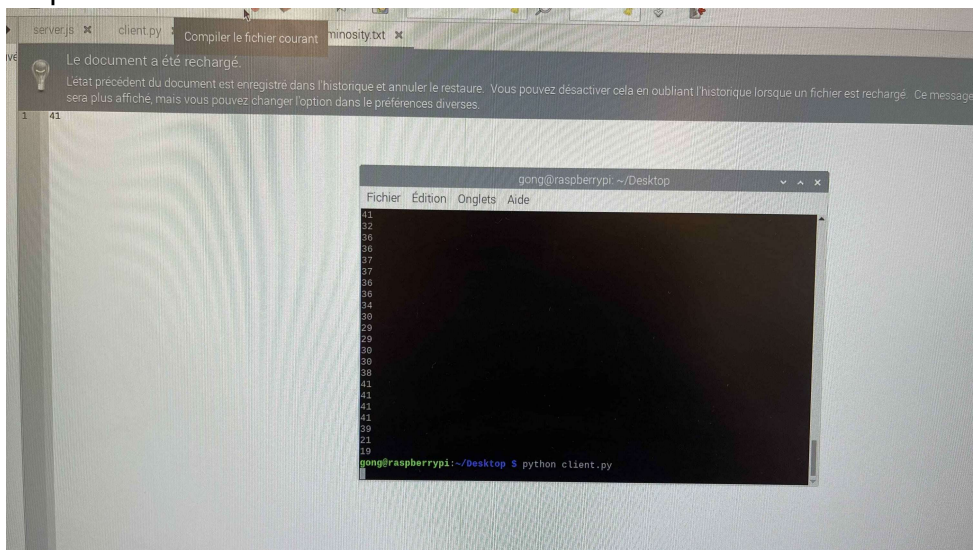
On a crée un fichier de python pour connecter le moquitto et le lire pour écrire dans le fichier text.

Pour lancer le python il faut taper dans le terminal:

```
python client.py
```

Avant lancer le python, il faut créer d'abord le fichier luminosity.txt dans le même répertoire.

Capture d'écran de fichier de txt:

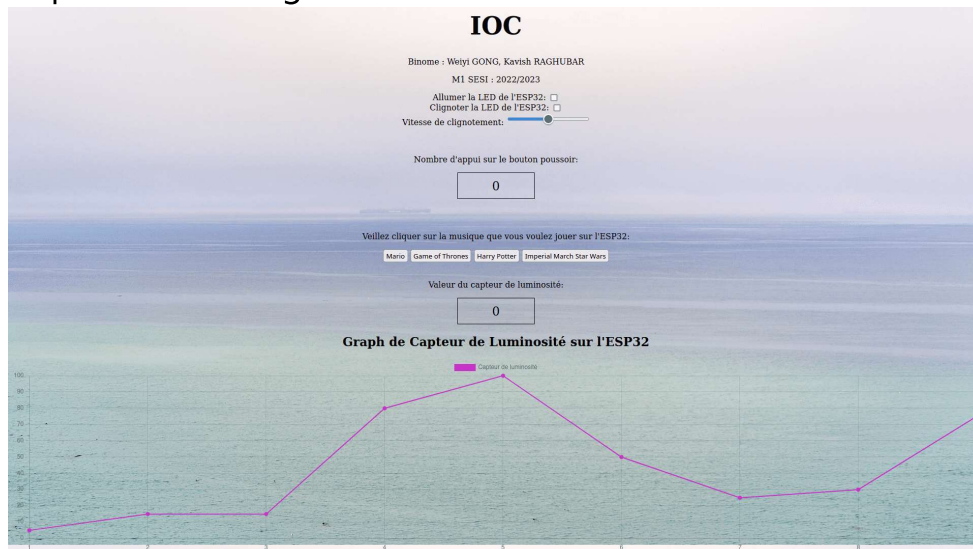


Nous traitons rasp comme un client de mqtt pour recevoir l'information sur l'intensité lumineuse de esp32, nous abonnons rasp au sujet que nous voulons et écrivons

l'information dans un fichier txt, de sorte que du côté serveur nous pouvons lire l'information dans le fichier plus facilement et l'afficher.

Pour lancer le server Javascript : `node server.js`

Capture d'écran globale du site web :



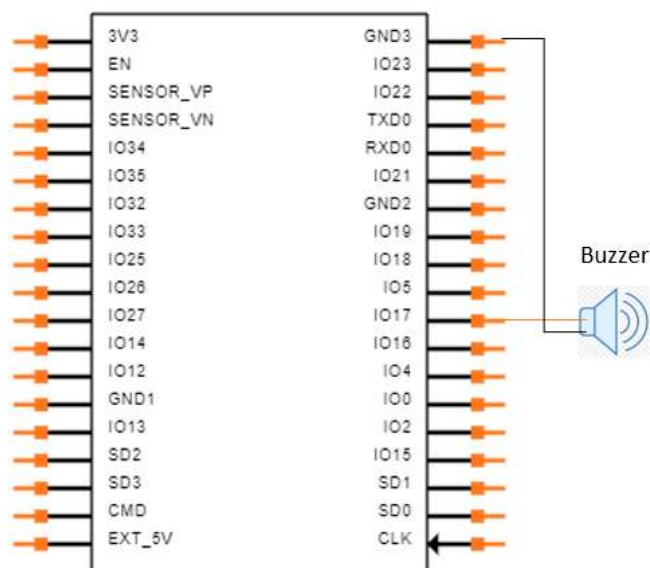
## Musique sur le Buzzer

Nous avons trouvé un model d'une Musique sur un tutoriel (pour plus d'information sur la source, veuillez voir le code), et par la suite, nous l'avons modifié afin de l'utiliser dans notre projet.

Nous avons une musique de thème : The legend of Zelda  
Veuillez voir la vidéo YouTube pour une démonstration.

<https://youtu.be/pegTSoTb5lw> (<https://youtu.be/pegTSoTb5lw>)

Le buzzer piézo-electrique est connecté sur la broche 17.



Dans le code arduino, le code commence par définir les notes à jouer, ainsi que leurs fréquences respectives.  
Les noms des notes sont suivis d'un nombre qui

représente la durée de la note. Par exemple, NOTE\_C4 est défini avec une fréquence de 262 Hz, ce qui correspond à une note de "do", et une durée d'un quart de note (4).

REST est défini avec une fréquence de 0 Hz et est utilisé pour créer une pause entre les notes.

La fonction `play_zelda()` contient la mélodie de la chanson et son timing. Elle commence par définir la variable `tempo` pour régler le tempo général de la chanson. La variable `buzzer` est définie sur le numéro de broche 17 utilisé pour connecter le buzzer piézoélectrique.

Le code fait également clignoter la LED(bleu) intégrée sur la carte ESP32 en synchronisation avec la musique. De base, `tempo` valait 170 mais comme nous avons utilisé le `blink` en synchro avec la musique, et que les fonctions `tone()` et `noTone()` sont bloquantes, nous avons multiplié ce `tempo` par 2 afin d'adapter la vitesse du son.

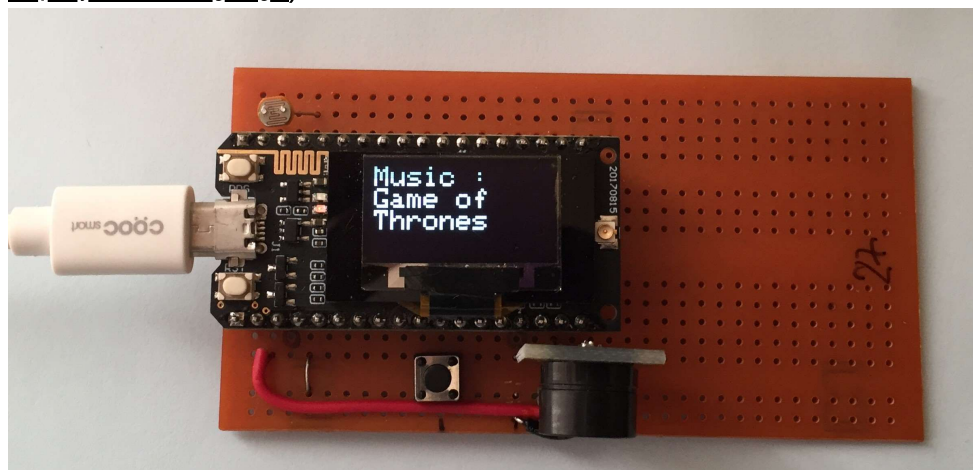
La variable `melody` est un tableau qui contient les notes et leurs durées. Les nombres négatifs sont utilisés pour représenter les notes pointées, qui sont des notes qui ont une durée de 1,5 fois celle de leur équivalent non pointé. Par exemple, une note pointée d'un quart de note est représentée par -4, ce qui signifie qu'elle dure trois huitièmes de note.

Nous avons ensuite fait appel à `play_zelda()` dans le `setup()`, pour ne la jouer qu'une seule fois.

On répète ces mêmes étapes pour les 4 autres musiques notamment:

-GameofThrones (<https://youtu.be/esAGPgWQsgo>)

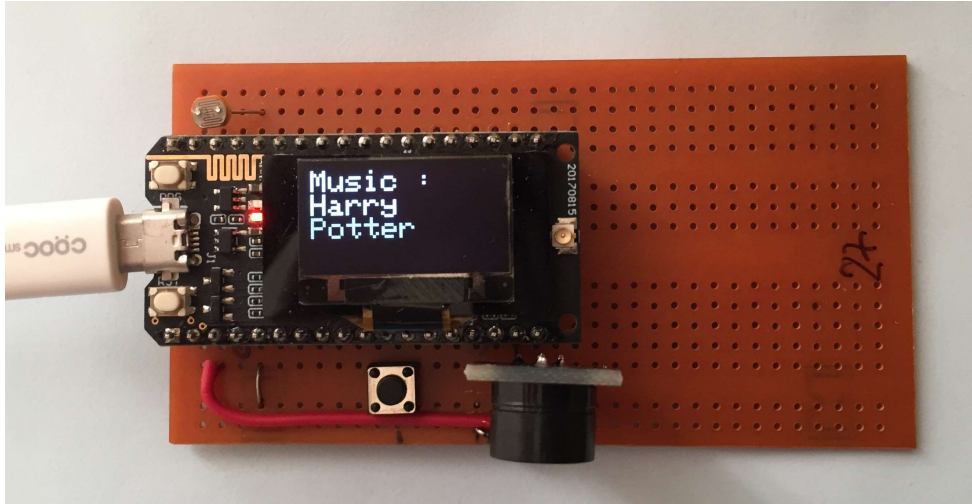
(<https://youtu.be/esAGPgWQsgo>)





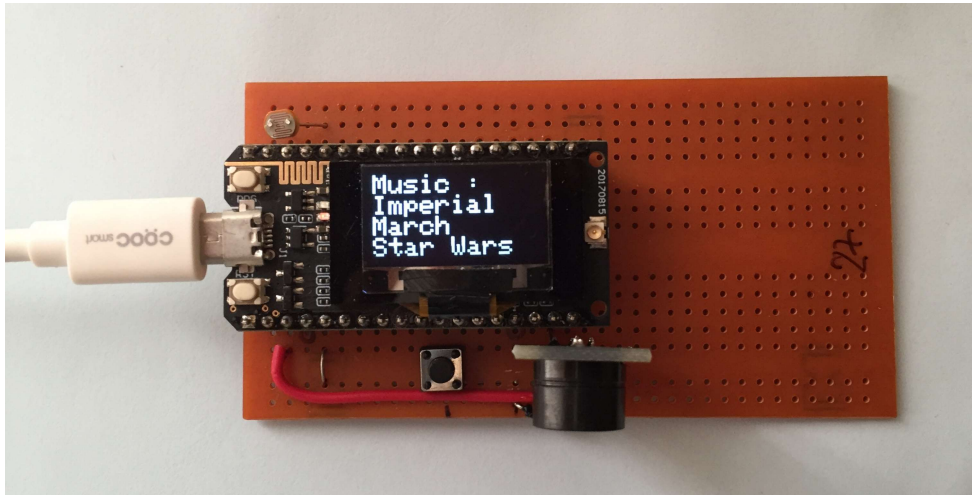
-HarryPotter (<https://youtu.be/1NoqqIQXHf0>)

(<https://youtu.be/1NoqqIQXHf0>)



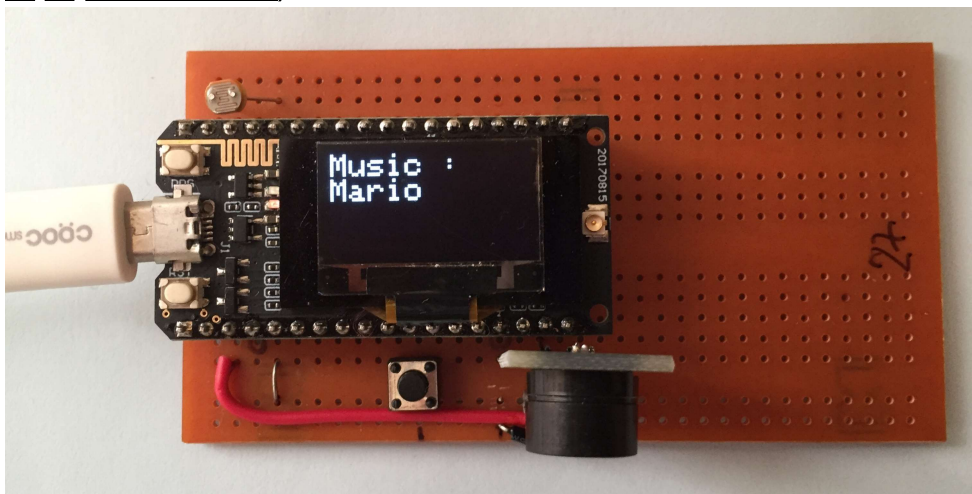
-ImperialMarchStarWars (<https://youtu.be/eff7urKNfC0>)

(<https://youtu.be/eff7urKNfC0>)



-SuperMarioBros (<https://youtu.be/LIE0dRQZ0d8>)

(<https://youtu.be/LIE0dRQZ0d8>)



Comme on peut le voir sur la capture d'écran ci-dessous, nous avons les boutons de musique sur le site web. On utilise le code html ci-dessous afin d'implémenter ces options.

<p><br>Veillez cliquer sur la musique que vous voulez

```
jouer sur l'ESP32:</p> <button  
onclick="playMusic('mario')">Mario</button> <button  
onclick="playMusic('got')">Game of Thrones</button>  
<button onclick="playMusic('harrypotter')">Harry  
Potter</button> <button  
onclick="playMusic('imperialmarch')">Imperial March  
Star Wars</button>
```



Du côté de l'ESP32, nous avons le nom de la Musique qui s'affiche. La fonction `affiche_oled()` est appelée lorsque la fonction `play_mario()` par exemple est appelée elle même. Pour ces musiques nous n'avons pas implémenté de clignotement synchro contrairement à "The legend of Zelda".

Sur le site web, avec un clique sur le bouton de la musique correspondante, on peut envoyer le choix (un entier par exemple), et en utilisant un `switch...case`, nous pouvons choisir la bonne musique à jouer sur l'esp32.

### **Serveur JavaScript**

Pour cette partie, nous avons implémenté le code pour gérer les fonctionnalités qui seront affichées sur le web. Pour ce faire nous avons écrit le code JS.

Afin d'avoir le background du site web, on utilise `style.css` :

```
body { background-image: url('/background.jpg'); }
```

Nous avons l'application Express qui va envoyer la page HTML avec le formulaire contenant :

- Les cases "allumer la led", "blink" à cocher
- curseur pour varier la vitesse de clignotement
- boutons pour selection de musique
- affichage (read only) : nombre d'appuis sur le bouton
- affichage (read only) : pourcentage de luminosité
- graphe : qui se génère en temps réel à partir d'un tableau (database)



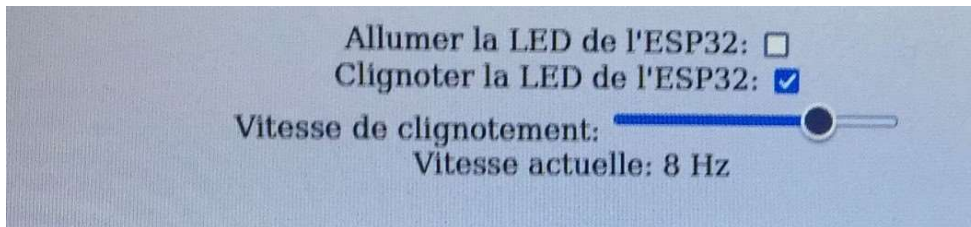
Nous n'avons pas pu tout implémenté, ce pendant cela nous laisse des idées pour faire plusieurs types d'interactions entre le Rpi (server) et l'ESP32 (client).

Variation de vitesse pour blink :

Nous avons écrit une fonction "updateBlinkSpeed()" qui va nous mettre à jour la vitesse de clignotement lorsque l'utilisateur déplace le curseur.

Voici un barre de défilement afin de faire varier la vitesse de clignotement de 2 Hz à 10 Hz par pas de 2.

```
<div> <label for="blink-speed">Vitesse de  
clignotement:</label> <input type="range" id="blink-  
speed" name="blink-speed" min="2" max="10" step="2"  
oninput="updateBlinkSpeed()"><br> <div id="blink-speed-  
value"></div> </div>
```



Dans la fonction playMusic(), nous avons eu l'idée de capturer l'appui sur le bouton sur le site, et ensuite c'est cette donnée qui sera envoyée au client (ESP32) afin de jouer la musique correspondante. L'idée est d'avoir le client Web publish la commande associée au topic 'topic\_music' alors que le client ESP32 sera subscribed à ce même topic. On pourra donc stocker ce message dans une variable 'choix\_music' et dès que l'utilisateur click sur un bouton sur le Web, on fera appel à la fonction playMusic() pour jouer la musique dans l'ESP32.

Serveur vers esp32: (maybe delete this part)

Nous avons également essayé le code qui envoie les informations sur le serveur à esp32, sur esp32 il suffit de définir esp32 comme un abonné et de s'abonner aux informations que nous voulons, par exemple, nous définissons esp32 comme un abonné à topic\_led, puis dans la fonction de rappel nous pouvons recevoir les valeurs correspondantes et les valeurs correspondantes que nous lisons ensuite peuvent être comparées, faire quelques activités et ainsi de suite.

Conclusion :

A la fin de ce mini-projet, nous avons pu acquérir plusieurs connaissances dans le monde de l'IoT(Internet of Things), notamment l'usage :

- de MQTT - Broker(Raspberry Pi) et Client (ESP32 et Web)
- du buzzer (en jouant de la musique)
- de l'écran OLED (affichage du nom de la musique)
- du capteur de luminosité
- de l'application web écrit Javascript (et l'affichage sur un graph en temps réel, l'évolution de la valeur de luminosité)
- d'une base de donnée (luminosity.txt) pour récupérer la valeur avec '**client.py**' (<http://client.py>) et l'écrire dans cette base de donnée et ensuite faire une lecture de ce fichier avec le code en JS.

Le développement de tous ces modules nous ont permit de mettre en pratique ces connaissances.