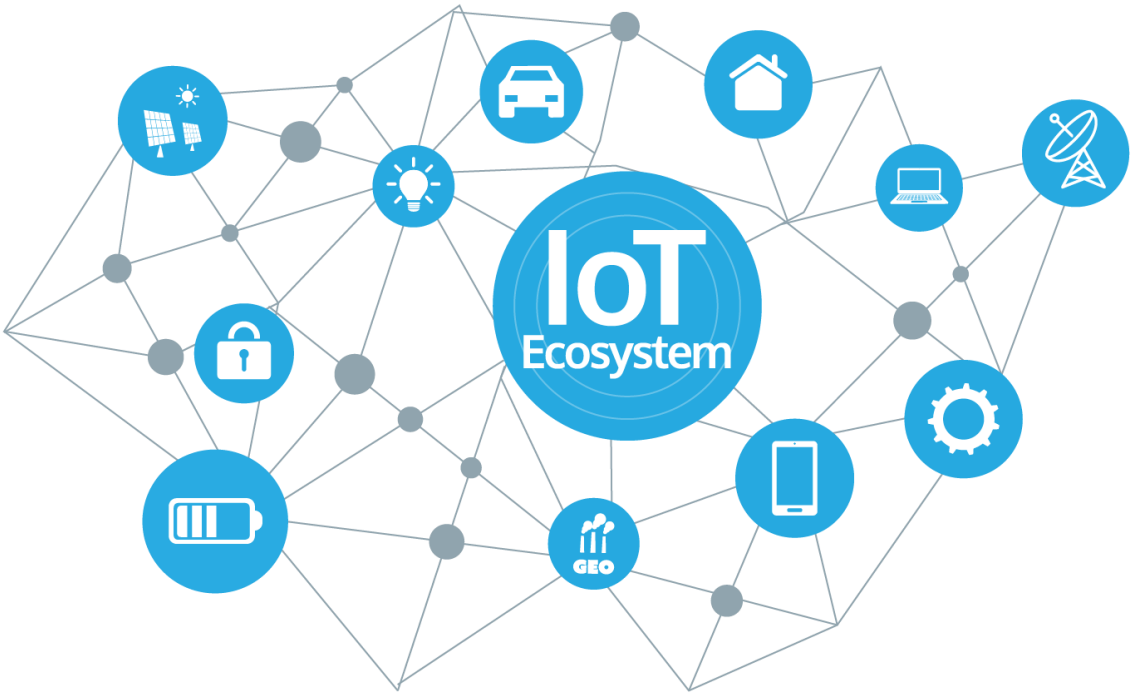
Iot RAPPORT

Travaux pratiques HTTP & MQTT



ABI KHALIL Charbel

RAMOUL Inès

18.03.2019

Master 1 MIAGE

INTRODUCTION

L'Internet des objets, ou IoT (Internet of Things), est un scénario dans lequel les objets, les animaux et les personnes se voient attribuer des identifiants uniques, ainsi que la capacité de transférer des données sur un réseau sans nécessiter aucune interaction humain-à-humain ou humain-à-machine.

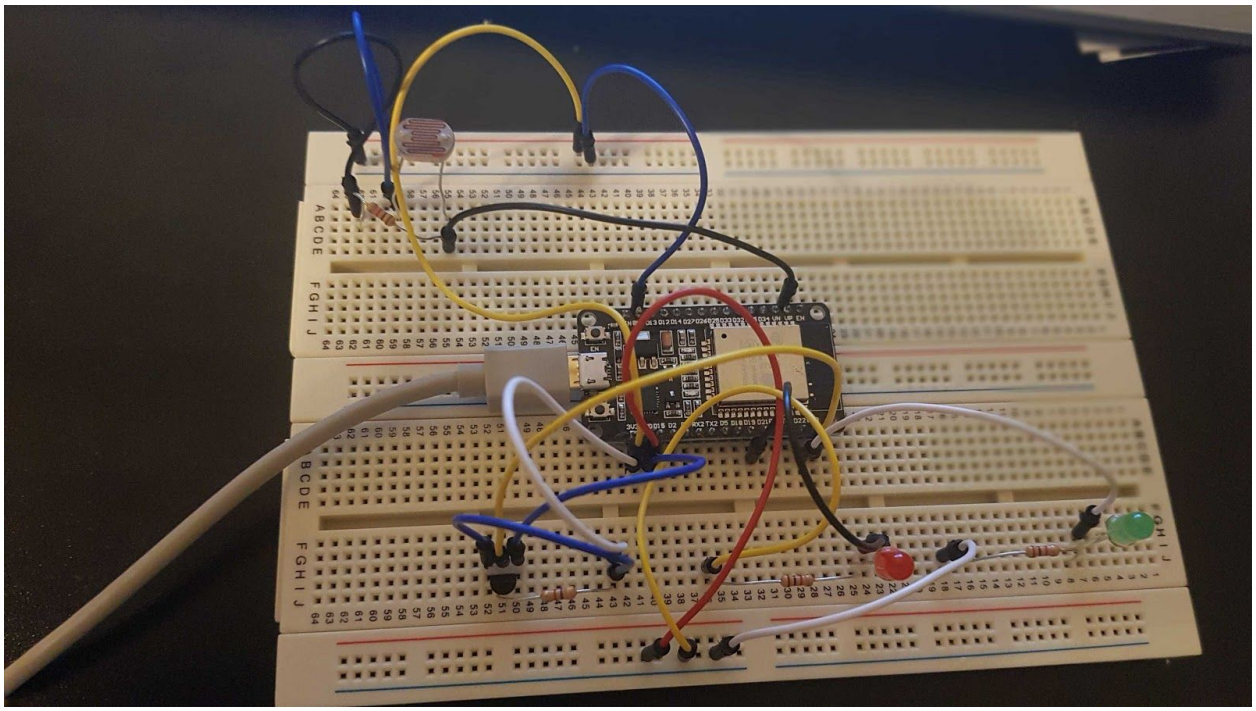
MATERIAUX

1. ESP32
2. Leds
3. Resistances (1K et 4.7K)
4. Photoresistor
5. Capteur de température
6. Cables

PROCEDURE

1. Une LED branchée au D19
2. Une LED branchée au D22 (elle joue le rôle du radiateur)
3. Capteur de température branché au D23
4. Librairies utilisées:
    a. WiFi.h
    b. PubSubClient.h
    c. OneWire.h
    d. DallasTemperature.h
    e. ESPAsyncWebServer.h
    f. ArduinoJson.h
    g. AsyncJson.h
5. **IHM(dashboard):** Il faut avoir un compte CloudMQTT. Vous pouvez vérifier dans ce lien comment le faire et comment créer une instance de broker.

    https://www.cloudmqtt.com/docs/index.html



RESULTATS Http

Pour Http, nous avons utilisé une librairie "ESPAsyncWebServer.h" pour des soucis d'efficacité. En effet ce type de serveur est totalement asynchrone et n'exécute pas sur le thread de la fonction loop(), en clair tout se fait dans le setup() et lorsqu'on a mis en forme nos données au format JSON pour pouvoir l'afficher facilement dans notre interface WEB sous la forme "response", le client se termine et la mémoire est directement libérée ce qui favorise la gestion de la mémoire.

L'asynchrone en JavaScript repose sur un circuit partant de la pile d'exécution, sortant du moteur JavaScript via les API du système hôte qui peut être le navigateur ou NodeJS, la file d'attente des callbacks, la boucle des événements, pour enfin revenir au moteur JavaScript sur la pile d'exécution.

Pour revenir sur le tp partie ESP32:

Vous trouverez les fichiers concernant le tp2 HTTP dans le **dossier tp2_http**.

Pour voir le résultat, il faut téléverser le code arduino esp_control.ino (dossier esp_control) sur votre esp32 après avoir branché les composantes requises sur les bons pins, ensuite ouvrir espcontrol.html dans votre navigateur (firefox ou chrome).

ATTENTION il faut que le fichier index.js soit dans le même répertoire que le fichier “espcontrol.html”.

- lorsque la led est éteinte, c’est-à-dire http://192.168.43.209/allume on renvoit { led : eteinte} (format json)
- lorsque la led est allumée, c’est-à-dire http://192.168.43.209/eteindre on renvoit {led : allumee}

Sur http://192.168.43.209/home on récupère l’état général de nos capteurs et de la led sur l’esp32, sous la forme d’un json (ex: {"led":"allumée","temperature":23.25,"sensorValue":146}), à savoir que “sensorValue” correspond au photoresistor sur branché sur A0.

L’avantage de notre interface web est qu’elle met à jour automatiquement chaque 3 secondes la réception de la requête “get” via l’url http://192.168.43.209/home, on a toujours un état frais des mesures de nos capteurs et de l’état de notre led.

RESULTATS MQTT

Pour MQTT, notre serveur utilisé est “m16.cloudmqtt.com” (c’est le serveur fourni après la création de l’instance CloudMqtt) . Nous avons définit deux topics, un pour la Led et un autre concernant la température: "miage1/menez/sensors/led" et "miage1/menez/sensors/temp"

Ensuite nous connectons l’ESP32 au Wifi et nous connectons notre serveur au client au port 13650 (fourni par CloudMQTT).

Sur le terminal nous avons des *Subscribers* qui s'abonnent au broker pour la capabilité de recevoir le message ou l'information partagée par le *Publisher*.

La commande pour s'abonner à un broker: mosquitto_sub -h m16.cloudmqtt.com -u xloreebd -P w5sTyyE-vwdA -t miage1/menez/sensors/led (pour la led) |miage1/menez/sensors/temp (pour la température). (ou **l'idéal**, vous consulter le dashboard CloudMQTT comme indiqué dans les deux figures ci-dessous)

La commande pour publier sur un broker: mosquitto_pub -h m16.cloudmqtt.com -u xloreebd -P w5sTyyE-vwdA -t miage1/menez/sensors/led -m "*le message*". (ou **l'idéal**, vous mettez le topic et le message dans le dashboard CloudMQTT comme indiqué dans les deux figures ci-dessous)

PS: Le username et le mot de passe sont fournis par CloudMQTT après la création du compte. Ils seront différent pour vous Monsieur après la création de votre compte.

Nous avons défini plusieurs critères:

1. Si le message est "on", la Led rouge branchée sur le D19 s'allume
2. Si le message est "off", la Led rouge branchée sur le D19 s'éteint
3. Si le message est "onchauffage", le chauffage branché sur le D22 s'allume
4. Si le message est "offchauffage", le chauffage branché sur le D22 s'éteint

Pour le topic de la température, nous avons répondu aux besoins de notre client : l'UCA !

L'UCA peut maintenant connaître la température de la salle qui sera affiché après l'abonnement au topic de la température et qui affiche aussi la valeur du contrat de la température constante (que vous pouvez changer dans le code) et vérifie si la température de la chambre est bien supérieur au seuil.

Si c'est pas le cas, le radiateur (représenté par la led verte) sera allumé, et ensuite éteint quand la température respectera le contrat de température de nouveau.

Pour le seuil, c'est le mode jour-nuit (critère horaire) qui l'indiquera, des que la salle est inoccupée. C'est le critère de luminosité qui indique cela en utilisant le capteur photoresistor.

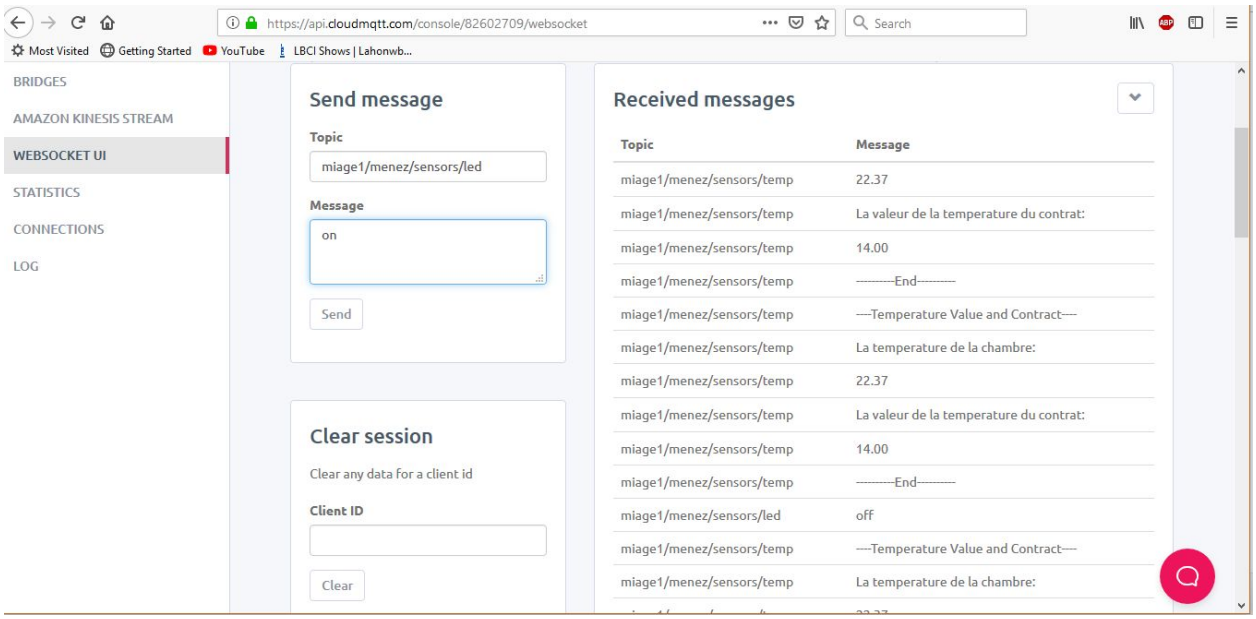


Figure 1: L'affichage de la température de la chambre par rapport à la valeur du seuil

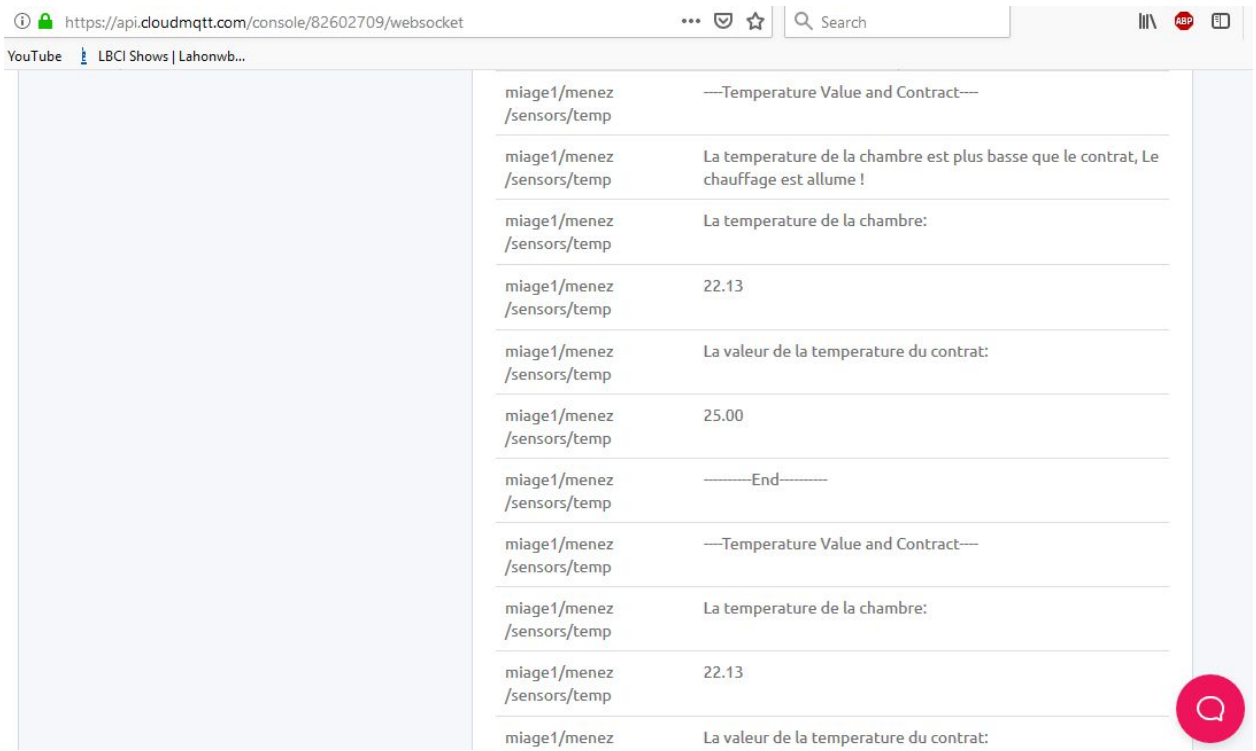


Figure 2: La température de la chambre est plus basse que le seuil, le radiateur (led verte) est allumé ! Et l'UCA est averti par un message

Avantages de MQTT

Légèreté

Beaucoup moins verbeux que HTTP, avec un côté asynchrone natif.

Flexibilité

MQTT est basé sur la couche réseau TCP/IP, utilisée par les protocoles internet dont HTTP.

On peut donc le trouver sur n'importe quelle plateforme matérielle, que ce soit un Arduino, un Raspberry Pi, un PC ou même un Cloud Microsoft Azure ou Amazon AWS.

De plus, on peut faire passer n'importe quel message sur les topics, par exemple du binaire ou du JSON, selon les besoins.

Sécurité

De nombreuses possibilités de sécurisation sont disponibles. Mot de passe, authentification par certificats client et serveur, chiffrement SSL/TLS, listes de contrôle d'accès…

C'est une qualité à ne pas négliger dans un monde où les objets connectés constituent un vecteur d'attaque important.

Intégrité des données

MQTT introduit la notion de *qualité de service* (QOS) qui permet à un client de s'assurer qu'un message a bien été transmis, avec différents niveaux de fiabilité.

Il y a aussi une fonctionnalité assez formidable : le *will message*, qu'on pourrait traduire par "testament". Il s'agit d'un topic qui est envoyé automatiquement lorsqu'un client se

déconnecte. Cela permet à n'importe quel client d'être notifié *post mortem* de la déconnexion d'un autre client et d'agir en conséquence.

CONCLUSION

MQTT est donc un protocole léger, facile à appréhender, très souple et sécurisable.

Il est de plus en plus au cœur des projets IoT et supporté par la plupart des services de Cloud.

MQTT est un protocole dynamique qui continue à évoluer. Par exemple, il intègre dorénavant les WebSockets qui lui ouvrent une porte sur le monde du Web.

Au sein des équipes de SII, notamment à Grenoble, nous mettons d'ores et déjà en œuvre ce protocole sur des systèmes éclectiques comme des équipements de gestion de réseau électrique, des box telecom et domotiques, des puces spécialisées pour l'intelligence artificielle, ou encore des poubelles connectées.

REFERENCES

1. https://blog.groupe-sii.com/le-protocole-mqtt-dans-liot/