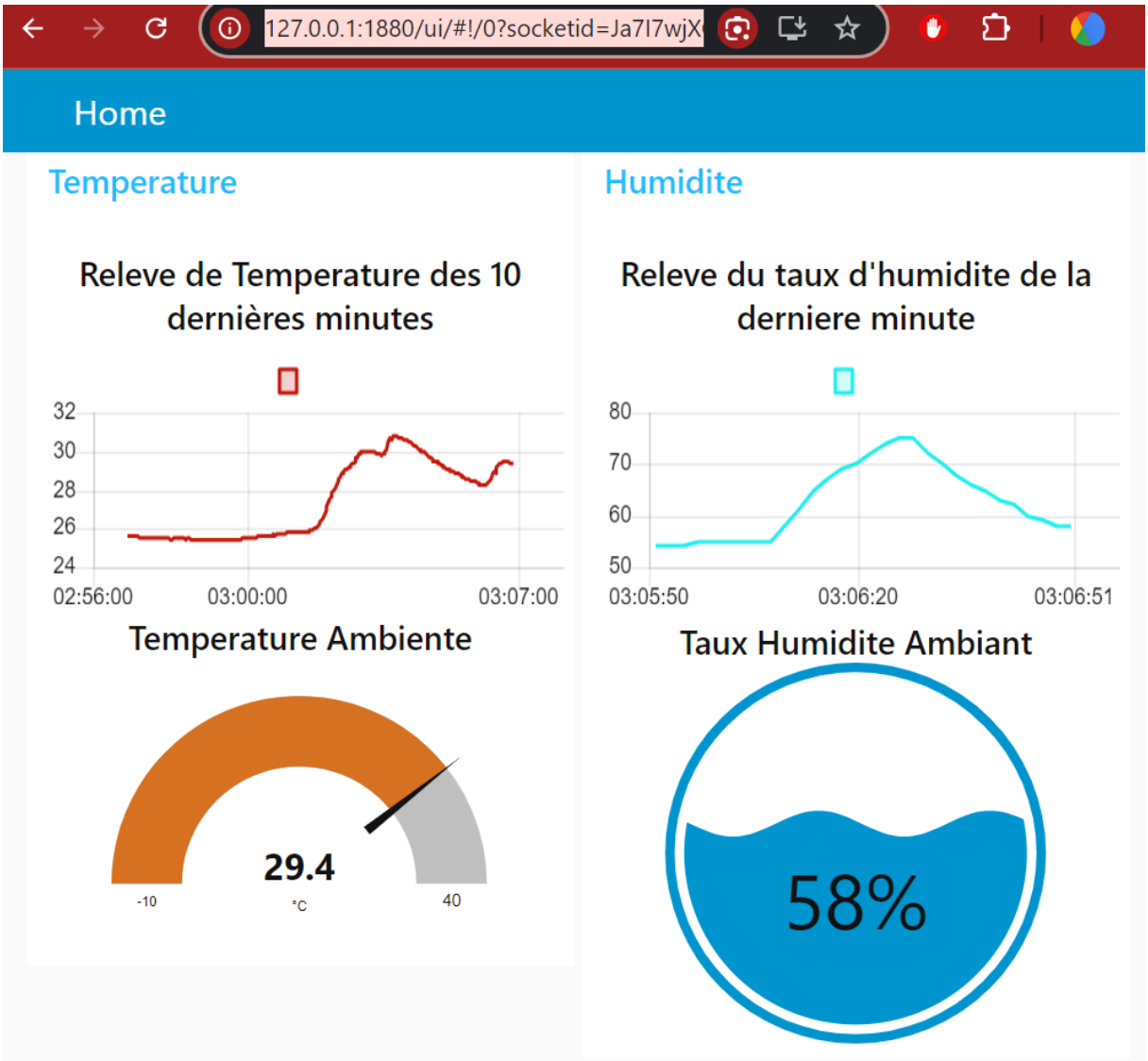
Rapport de Travaux Pratiques - 2024.09.16

Internet des objets - Protocoles HTTP REST et MQTT pour la communication bidirectionnelle avec des capteurs IoT

Table des matières

Figure 2.4 : Interface Utilisateur visualisant montée des données en HTTP REST.



Introduction

Dans le cadre de notre parcours universitaire en Électronique Informatique, nous sommes amenés à interagir avec des objets connectés, pouvant à la fois servir pour des relevés ponctuels dans des environnements disposant de connexions Internet stables, à haut débit, et sans contrainte énergétique. Cela permet l'utilisation de protocoles comme HTTP REST, qui est plus ergonomique, comme dans le cas d'un capteur dans un appartement de ville connecté via une prise, utilisé pour vérifier la bonne extinction des lumières quelques fois par jour.

À l'inverse, certains systèmes fonctionnent dans des environnements à faible bande passante, où une garantie de transmission asynchrone et à latence minimale est souvent nécessaire, notamment dans des milieux fortement contraints énergétiquement. Par exemple, tout à fait aléatoirement, une ruche connectée dans une région reculée, comme au fin fond de la Creuse, avec une connexion Internet très faible, doit envoyer des relevés récurrents de manière compacte, en utilisant le peu d'énergie disponible.

Nous allons donc explorer deux protocoles de communication largement utilisés dans les systèmes embarqués et les environnements IoT : HTTP REST et MQTT. Nous démontrerons ces protocoles par la transmission bidirectionnelle des données entre un

capteur, ici un DHT11, et un microcontrôleur ESP8266, tout en exploitant un serveur local Node-RED. L'initialisation du Broker Mosquitto, la génération du serveur avec Node-RED sont disponibles en annexe A. Mise en place de l'environnement de travail.

L'objectif sera de comparer ces deux approches à travers la mise en place d'une communication entre un capteur DHT11 et un serveur Node-RED. Nous allons configurer deux démonstrations pour vérifier la communication bidirectionnelle via les deux protocoles.

Nous conclurons par une analyse des résultats obtenus, en mettant en avant les avantages et les limitations de chaque méthode de communication en fonction des contraintes environnementales.

Aide rédactionnel, au débogage, au code et soutien : Yulin, Maxime, Ayman, Victor, Quentin, Nicolas, ChatGPT, HARIAN Elyoth, DOUZE Yann, Benjamin, Daniel.

1.Test du matériel - Test unitaires.

Après avoir configuré l'environnement de travail disponible dans l'annexe A. Mise en place de l'environnement de travail, nous allons vérifier les différents composants pour s'assurer de leur bon fonctionnement.

1.1. Test du capteur DHT11

Nous avons connecté le capteur, à la borne D7 du microcontrôleur, correspondant d'après la documentation technique à la sortie hardware GPIO 13 par A.3 Configuration de Arduino IDE pour échanger avec l'ESP pour tester le bon fonctionnement du capteur DHT11. Le capteur a été testé en utilisant le moniteur série, où les valeurs mesurées ont été vérifiées par la suite avec un thermomètre. Les résultats obtenus montrent que le capteur fonctionne correctement et transmet bien les données au microcontrôleur.

Figure 1.1.1 : Diagramme de séquence du relevé des données du capteur.

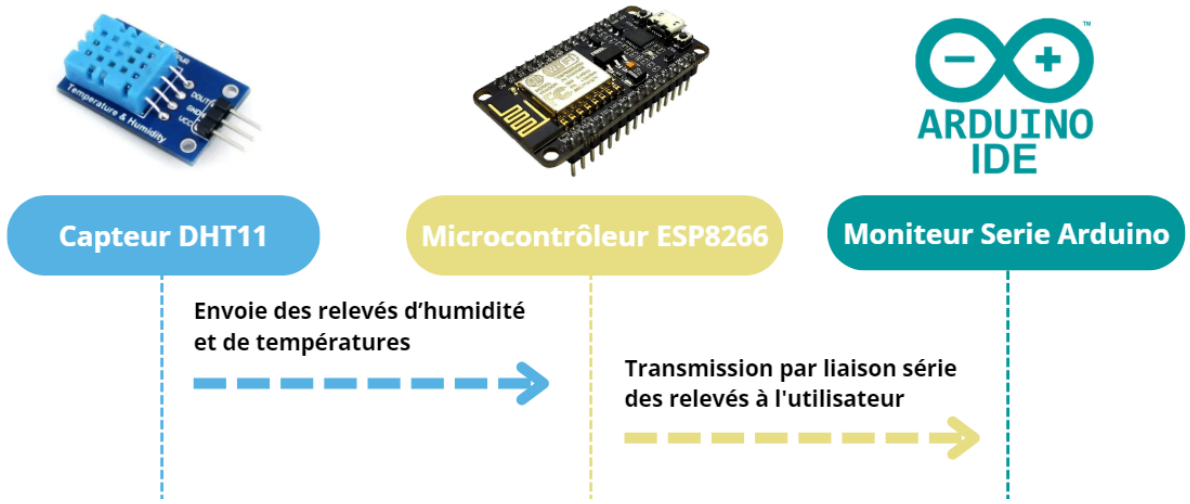
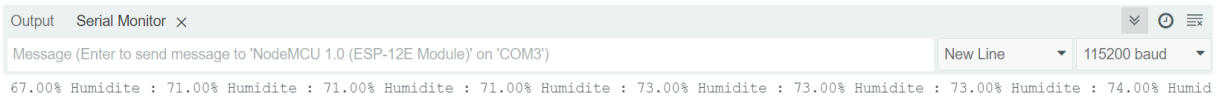


Figure 1.1.2 : Vu du Serial Monitor du test capteur pour la variable humidité.



1.2. Test serveur Node-RED

D'autre part, nous avons configuré le serveur Node-RED pour nous assurer qu'il est capable de recevoir des requêtes HTTP et de les traiter correctement. À l'aide de la commande curl depuis le terminal « PS C:\Users\vladi> curl http://127.0.0.1:1880/test », nous avons testé la capacité du serveur à répondre à une requête GET. L'interface utilisateur de Node-RED a également été testée, notamment le tableau de bord qui servira à visualiser les données envoyées par l'ESP8266.

Figure 1.2.1 : Schéma des flux du serveur Node-RED pour tester un dashboard.

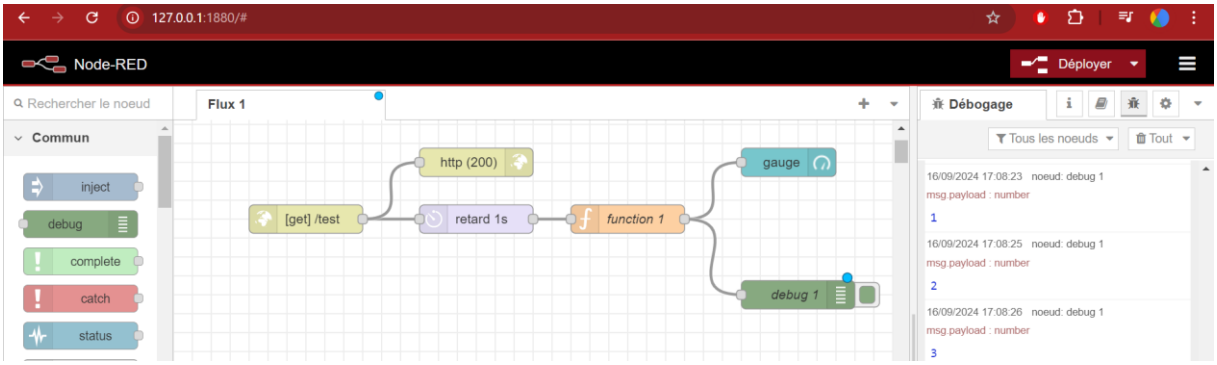


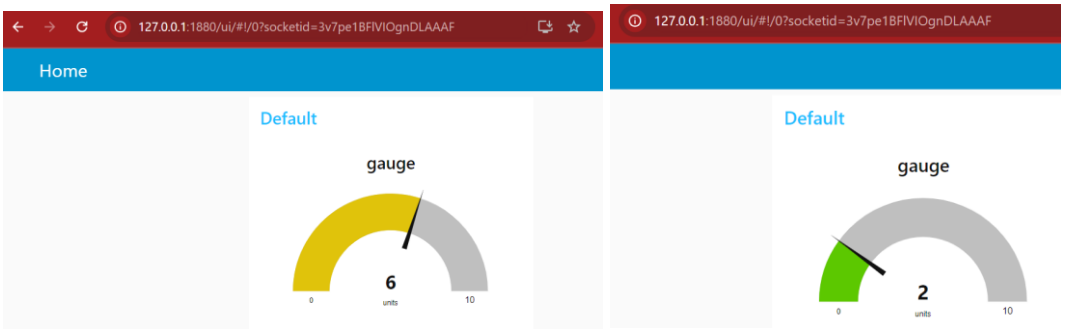
Figure 1.2.2 : Extrait de code dans le bloc fonction pour tester l'interface visuel.

```
let test = global.get('test') || 0

if (test >= 10) {
  test = 0
}
test += 1
global.set('test', test) //Maj test

return msg
```

Figure 1.2.3 : Interface Utilisateur visualisant la réaction à la stimulation du serveur



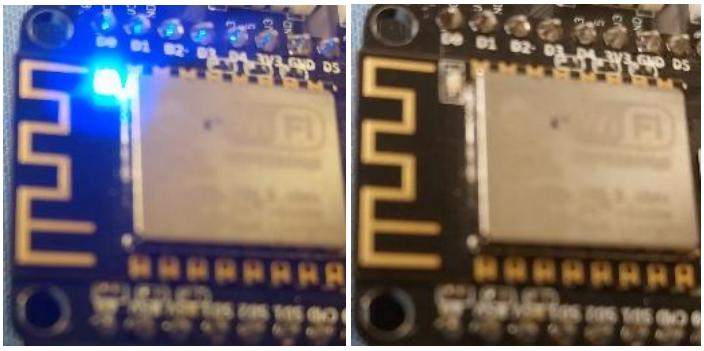
La jauge, s'incrémente bien à chaque requête reçue, ce qui nous permet de valider le bon fonctionnement du serveur.

1.3. Test de la LED ESP8266

Le dernier test concerne la LED intégrée de l'ESP8266, qui sera utilisée pour valider la descente des données. Après avoir chargé un programme simple pour faire clignoter la LED à une intervalle régulière, nous avons confirmé que le contrôle de la LED fonctionne correctement. Ce test est essentiel pour s'assurer du fonctionnement

de la LED, pour la validation de la descente des données avec les protocoles que nous allons mettre en place.

Figure 1.3.1 : Photo du microcontrôleur LED allumé puis éteinte.



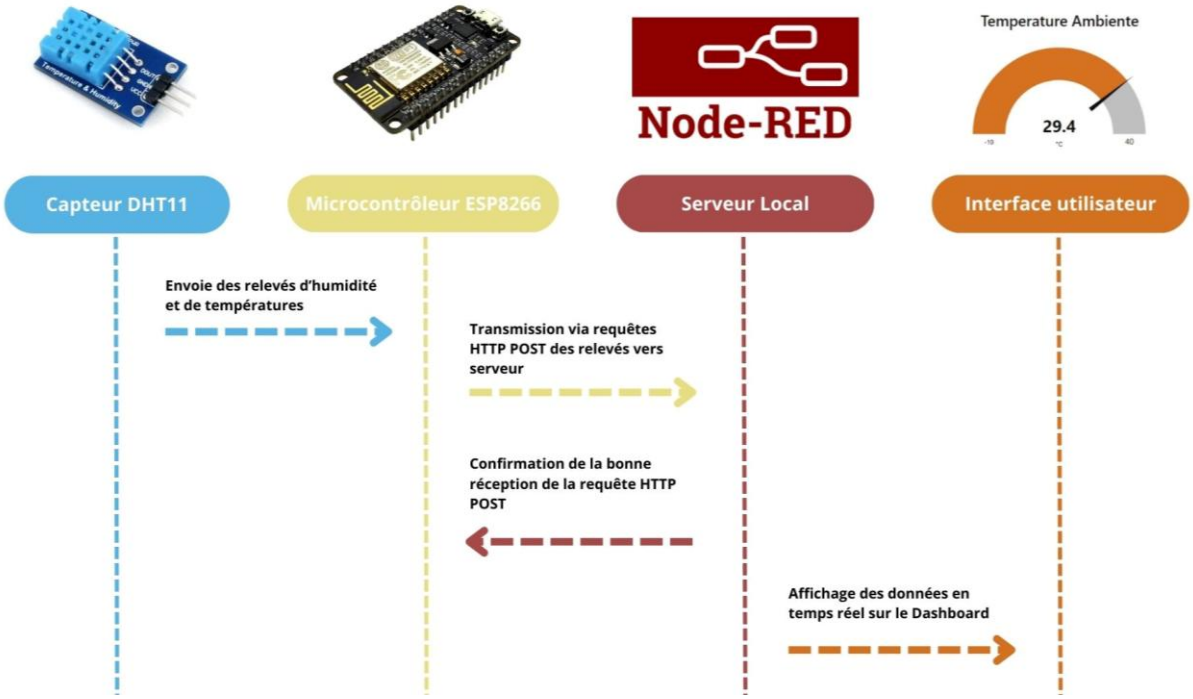
La LED clignote avec une période de 1s, ce qui est conforme aux attentes et valide ce dernier essai préliminaire.

2. Montée des données - HTTP REST

Maintenant que nous avons vérifié la fonctionnalité du serveur et du capteur, nous allons vérifier la montée des données en réceptionnant les données du capteur et en les visualisant grâce à l'interface de Node-RED. Dans ce premier temps nous allons effectuer la montée des données avec le protocole HTTP de REST.

Le REST est un protocole fonctionnant en milieux à réseaux stables et rapides tel qu'une connexion Wifi sans obstacles et sans se préoccuper de l'impact sur la bande passante ou de la consommation énergétique. En effet, HTTP REST fonctionne en envoyant régulièrement des requêtes HTTP entre ici un capteur appelé client et notre serveur Node.js sous NODE-RED. Surtout que chaque requête nécessite une nouvelle connexion, ce qui peut vider une petite batterie rapidement.

Figure 2.1 : Diagramme de séquence de la montée des données avec le protocole HTTP REST



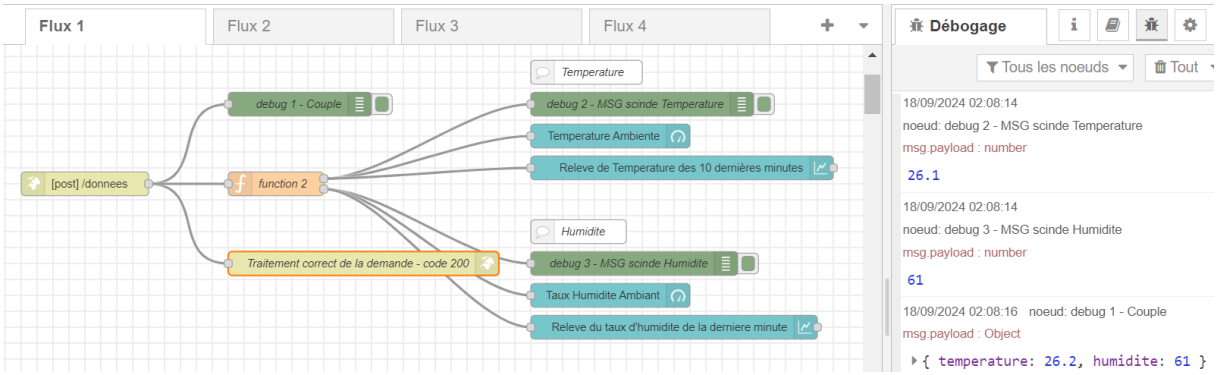
Pour la montée des données, le capteur DHT11 mesure la température et l'humidité, puis envoie ces données à l'ESP8266. L'ESP8266 les envoie ensuite au serveur Node-RED sous forme de requêtes HTTP (méthode POST) au format JSON. Le serveur les reçoit, les traite, et les affiche dans un tableau de bord en temps réel.

Nous aurons besoin d'une fonction pour sciender les sorties en 2, l'une pour les relevés d'humidité et l'autre pour les relevés de température. Les données seront ensuite visualisées depuis des dashboards au lien suivant :

http://127.0.0.1:1880/ui/#!/0?socketid=Ja7I7wjXGxKUvuxHAAAL

Pour transmettre une donnée de température, le capteur envoie une requête POST au serveur, contenant les données sous forme de JSON. Le serveur reçoit cette requête, extrait les données utiles, et pour confirmer la réception des données renvoie un code statut HTTP, comme dans notre cas 200 OK.

Figure 2.2 : Schéma des flux du serveur Node-RED pour la montée des données en HTTP REST.

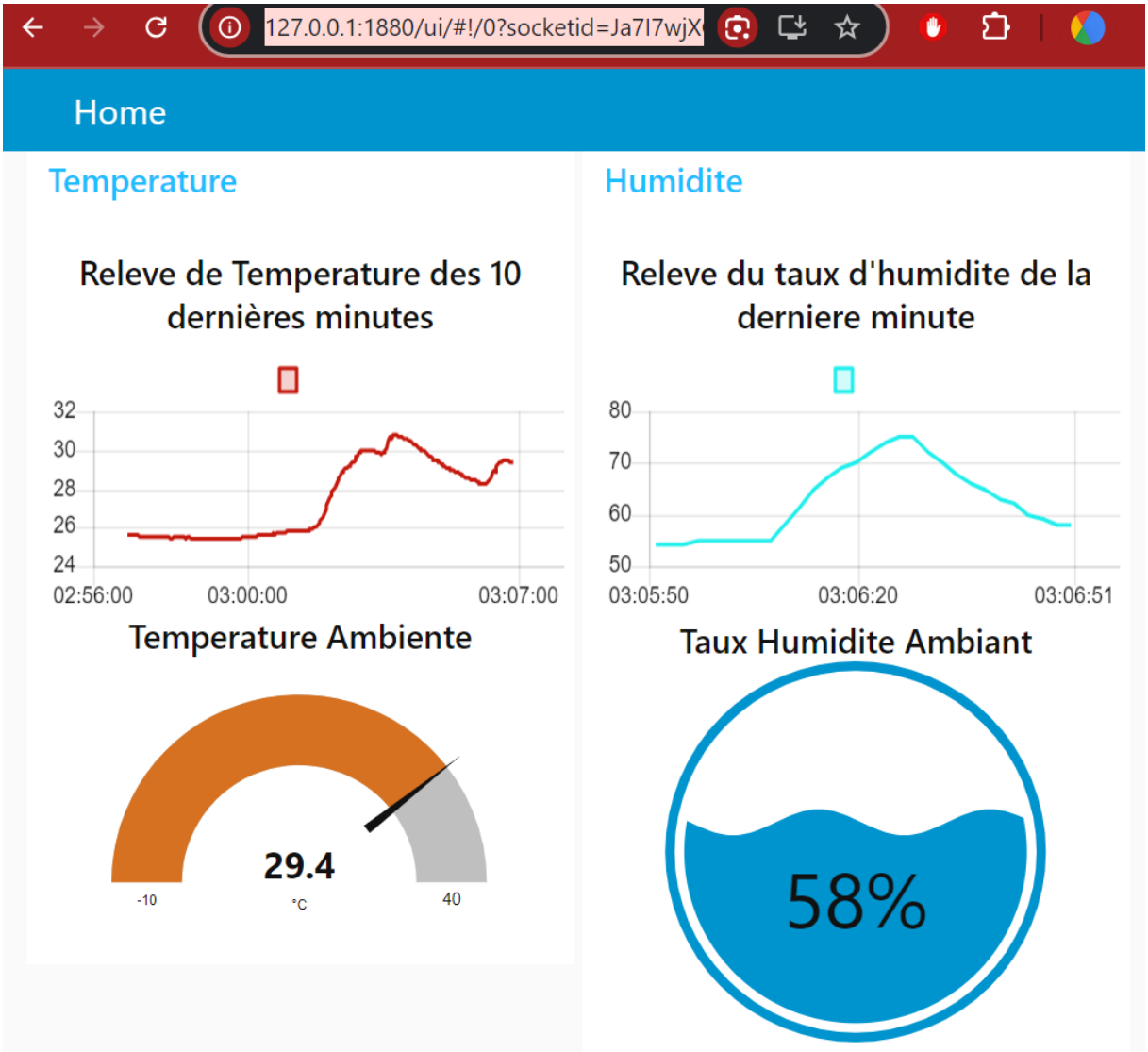


Sur notre flow, nous pouvons constater deux blocs distincts, le retour du code 200 pour accuser la bonne réception des données, et quelques blocs de débogage utiles pour obtenir un retour en temps réel des données avant la mise en place de dashboards fonctionnels.

Figure 2.3 : Extrait de code dans le bloc fonction pour scinder la donnée de température et celle d'humidité.

```
1 // Creation de variables globales et separees pour meilleur lisibilite
2 var temperature_msg_up = { payload: msg.payload.temperature };
3 var humidite_msg_down = { payload: msg.payload.humidite };
4
5 return [ temperature_msg_up, humidite_msg_down ];
```

Figure 2.4 : Interface Utilisateur visualisant montée des données en HTTP REST.

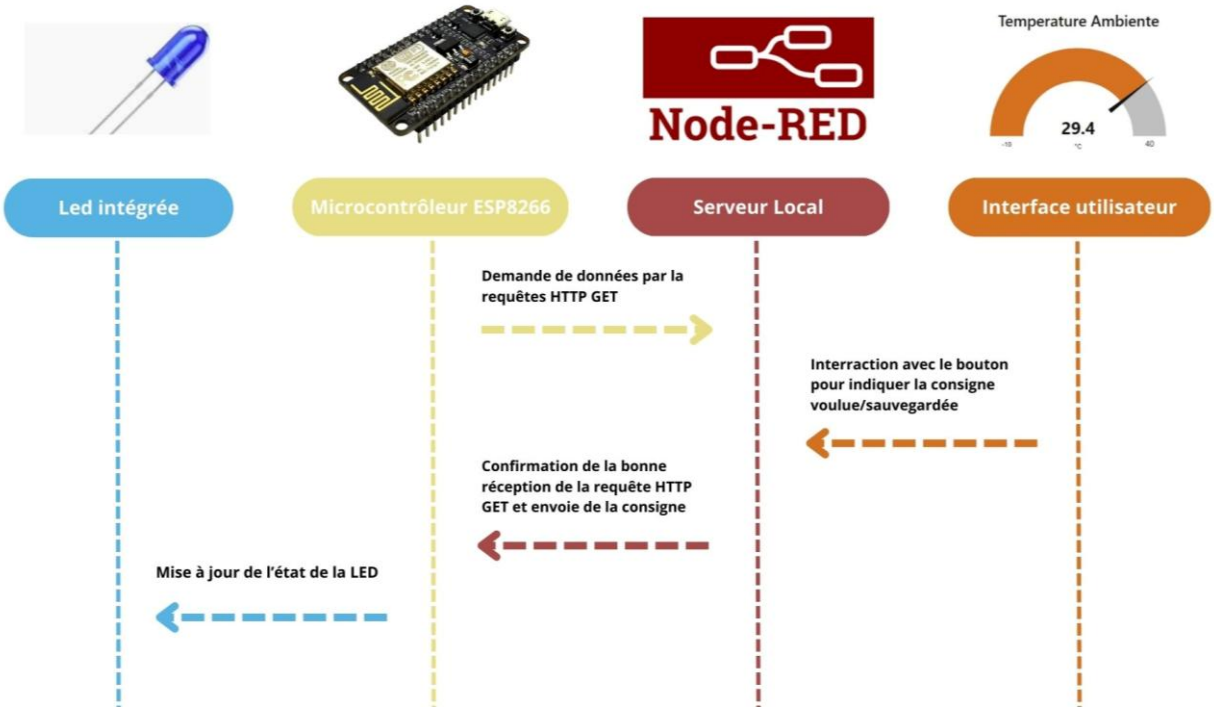


On remarque une variation des données, ce qui est bon signe et un bond fulgurant lors de la prise en main ce qui est cohérent avec la réalité. Les données postées par le capteur depuis le microcontrôleur valide ainsi cette partie.

3. Descente des données - HTTP REST

Pour vérifier la descente des données, nous nous proposons de faire clignoter la leds avec une période de 1 seconde en actionnement et la gardant à l'état haut, allumé sans clignotement en non-actionnement .

Figure 3.1 : Diagramme de séquence de la descente des données avec le protocole HTTP REST



L'interface utilisateur sur le tableau de bord Node-RED permet de contrôler la LED de l'ESP8266 par l'actionnement d'un bouton. Lorsque l'utilisateur appuie sur un interrupteur dans l'interface, une requête HTTP est envoyée à l'ESP8266, lui ordonnant de modifier l'état de la LED.

Figure 3.2 : Schéma des flux du serveur Node-RED pour la descente des données en HTTP REST.

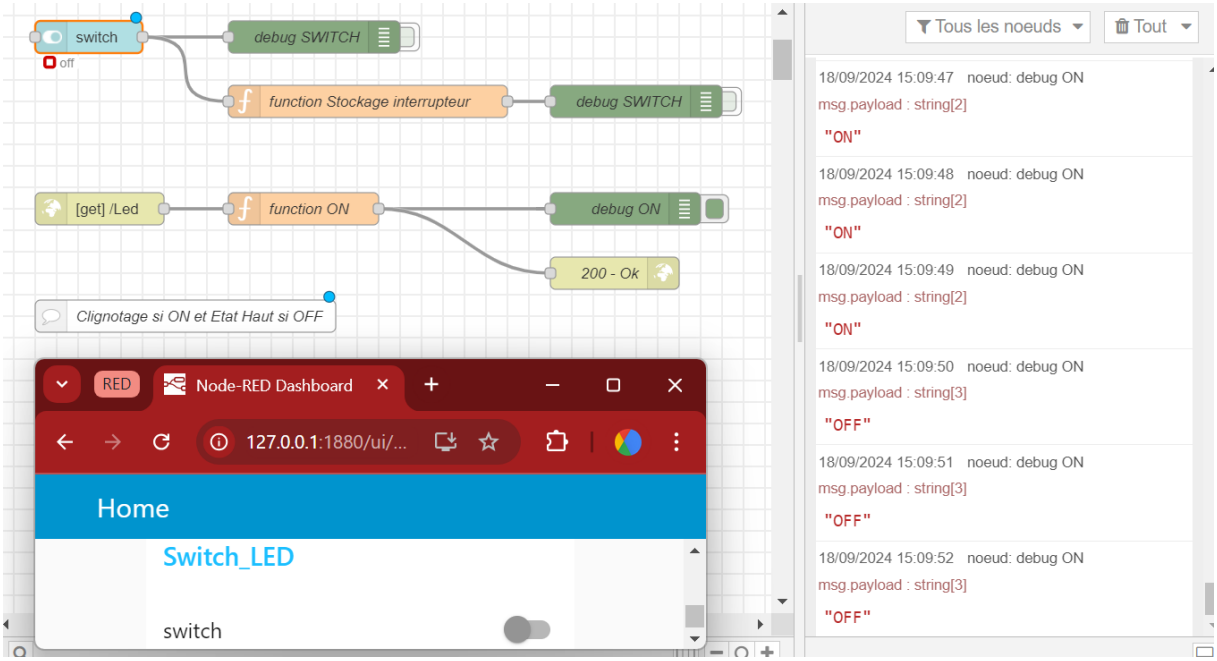


Figure 3.3 : Extrait de communication série avec le microcontrôleur de la récupération de la consigne depuis l'interrupteur du dashboard.

```
Output  Serial Monitor ×
Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM3')
OFF
La reponse est bonne
[HTTP] debut de la requete GET...
[HTTP] GET... code: 200
Reponse du serveur :
OFF
La reponse est bonne
[HTTP] debut de la requete GET...
```

Nous effectuons un appui sur l'interrupteur, tel qu'à l'état allumé dit ON, la LED intégrée du microprocesseur clignote en boucle ce qui est le cas, et reste bien à l'état haut allumé. Lorsque l'on éteint l'interrupteur comme ci-dessous le microcontrôleur reçoit bien une consigne OFF pour arrêter le clignotement à l'état allumé.

Cette partie valide le fonctionnement du protocole REST pour la montée et la descente des données. Cependant, REST présente des limites en termes de consommation énergétique, car chaque action nécessite l'ouverture d'une nouvelle connexion HTTP.

4. Montée-Descente des données - MQTT

Enfin pour tester le protocole MQTT, de manière similaire à précédemment, nous avons effectué à l'aide de la source suivante : https://randomnerdtutorials.com/esp8266-and-node-red-with-mqtt/ , repris, complété et adapté la montée et la descente des données, cette fois ci sans clignotement de la LED. Le point essentiel à retenir est l'intermédiaire appelé broker A.2 Installation de Mosquitto-BROKER (serveur) par étapes , étant un messager recevant les informations-sujet des capteurs et les redistribuant à tous les abonnées au sujet.

Figure 4.1 : Extrait de code de la publication des relevés dans les sujets-journaux Température et Humidité pour le protocole MQTT.

```
150   // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
151   float humidity = dht.readHumidity();
152   // Read temperature as Celsius (the default)
153   float temperatureC = dht.readTemperature();
154   // Read temperature as Fahrenheit (isFahrenheit = true)
155   float temperatureF = dht.readTemperature(true);
156
157   // Check if any reads failed and exit early (to try again).
158   if (isnan(humidity) || isnan(temperatureC) || isnan(temperatureF)) {
159     Serial.println("Failed to read from DHT sensor!");
160     return;
161   }
162
163   // Publishes Temperature and Humidity values
164   client.publish("capteur/Temperature", String(temperatureC).c_str());
165   client.publish("capteur/Humidite", String(humidity).c_str());
166   //Uncomment to publish temperature in F degrees
167   //client.publish("capteur/temperature", String(temperatureF).c_str());
```

Figure 4.2 : Schéma des flux du serveur Node-RED pour la montée des données en MQTT.

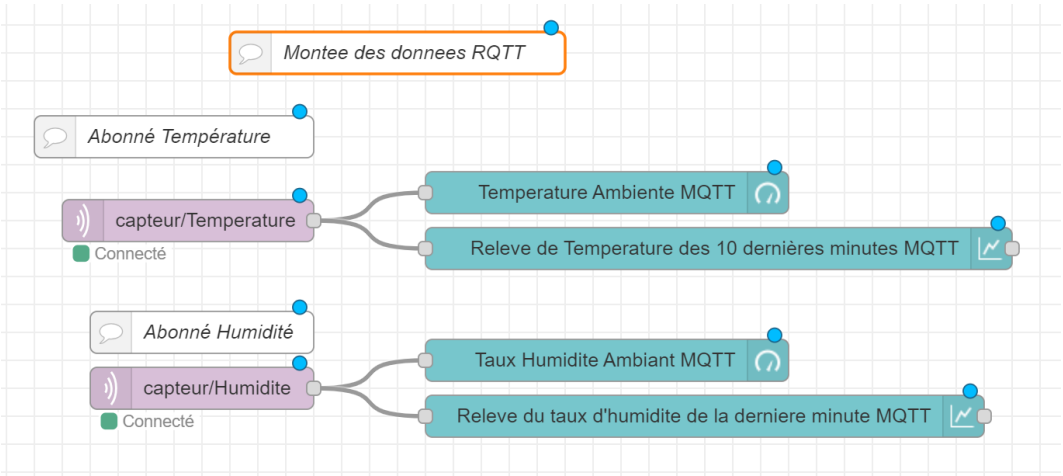
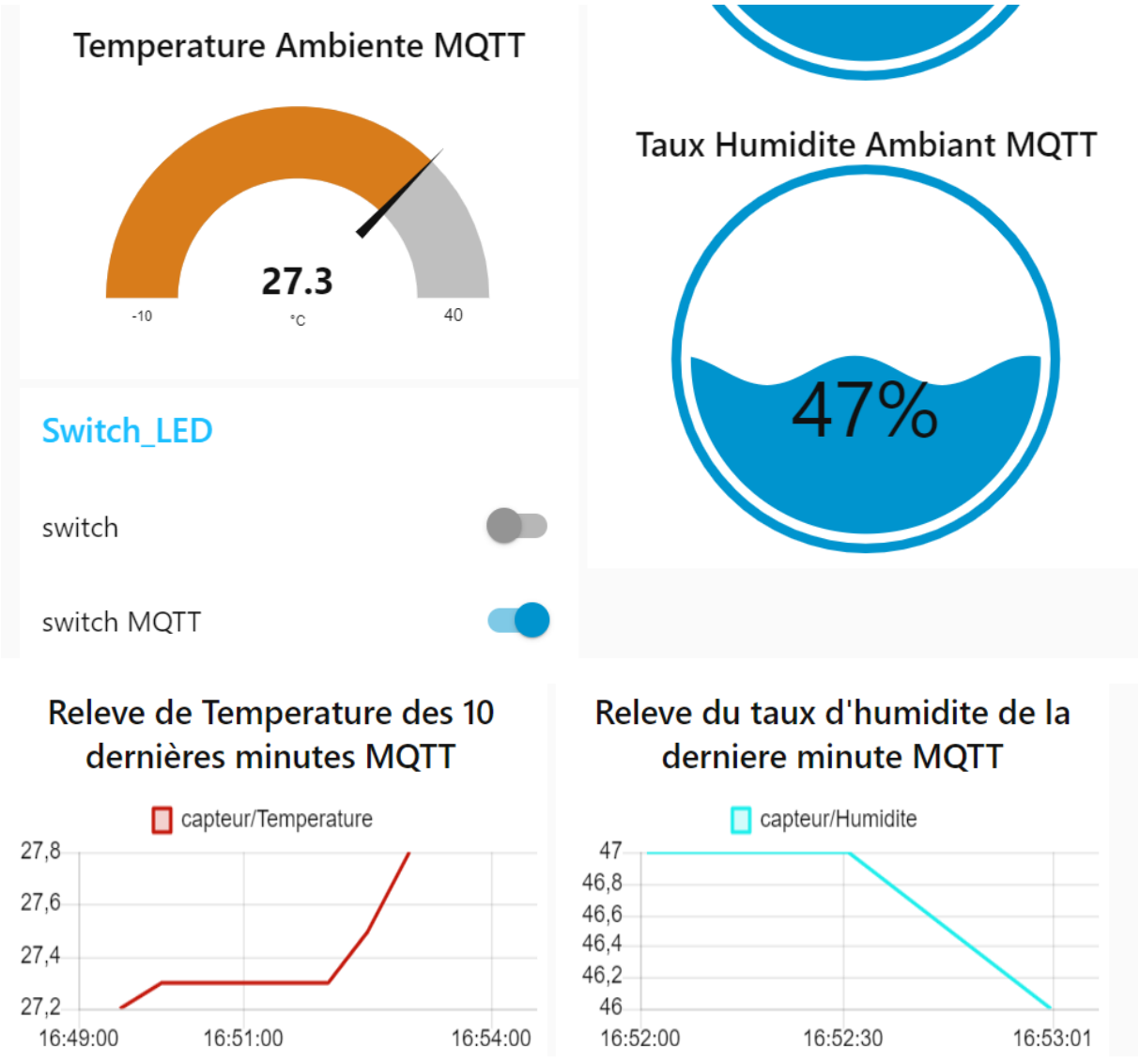


Figure 4.3 : Interface Utilisateur visualisant montée des données en MQTT.



Pour la montée des données en MQTT, le capteur est dynamique et transmet bien les évolutions ce qui est cohérent avec les attentes.

Enfin, pour la descente nous allons grâce au tuto cité précédemment, réadapté à la LED intégrée, réceptionner par abonnement de la LED au sujet de son actionnement par un bouton.

Figure 4.4 : Extrait de code de la réception du sujet-journal pour l'abonnement à la LED pour savoir en descente de donnée lorsque l'interrupteur est activé et donc que la LED doit changer d'état pour le protocole MQTT.

```
75    if(topic=="capteur/LED_BUILTIN"){
76      Serial.print("Changing Room lamp to ");
77      if(messageTemp == "on"){
78        digitalWrite(LED_BUILTIN, HIGH);
79        Serial.print("On");
80      }
81      else if(messageTemp == "off"){
82        digitalWrite(LED_BUILTIN, LOW);
83        Serial.print("Off");
84      }
85    }
86    Serial.println();
```

Figure 4.5 : Schéma des flux du serveur Node-RED pour la descente des données en MQTT.

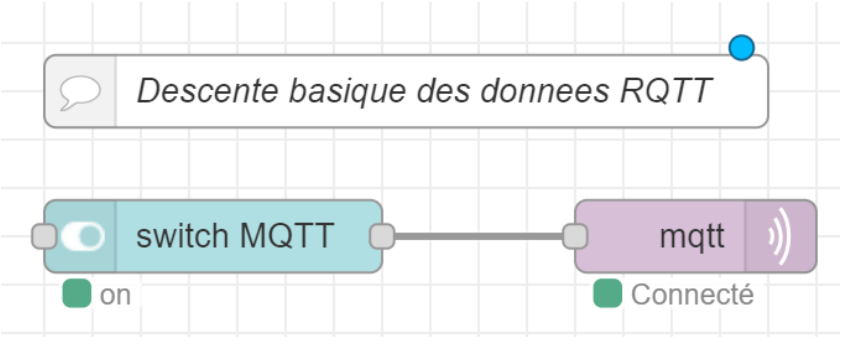


Figure 4.6 : Interface Utilisateur visualisant descente des données en MQTT.

Switch_LED

L'interrupteur, active-désactive la LED de manière asynchrone ce qui est attendu, et correct.

Pour conclure, nous avons vu que le MQTT est particulièrement efficace dans des environnements ruraux, ayant une faible qualité réseaux et souvent assez instable. Celui-ci est très économe en données, grâce à l'échange de messages très concis via un broker tel que :

```
Topic : capteur/Temperature
Payload : 30
```

Là où le protocole REST pour transmettre la même donnée, enverra au format JSON un contenu plus lourd à transmettre tel que :

```
POST /Temperature HTTP/1.1
Host : 127.0.0.1:1880
Content-Type: application/json
Content-Length: 18
{
 "temperature": 30
}
```

Nous aurons donc tendance à préférer pour nos utilisations souvent en milieux restreints de ce deuxième protocole très peu énergivore pour une efficacité très optimale.

Conclusion

Pour finir, nous venons de mettre en œuvre deux protocoles de communication bidirectionnelle avec des capteurs IoT : HTTP REST et MQTT. Chaque protocole a ses avantages et ses inconvénients selon l'environnement dans lequel il est utilisé. Le protocole HTTP REST se distingue par sa simplicité de mise en œuvre et sa compatibilité avec des environnements où la bande passante et la consommation d'énergie ne sont pas des contraintes majeures. En revanche, le protocole MQTT, plus léger, est bien plus adapté aux environnements contraints, comme les zones rurales incluant Saint Cyr où la bande passante est limitée et où l'énergie doit être économisée.

En conclusion, bien que le protocole REST soit plus ergonomique et intuitif dans des scénarios urbains tel que la domotique, le MQTT s'impose comme la solution idéale pour les environnements à faible connectivité. Nous avons donc vu non seulement les avantages de chaque protocole, mais aussi, plus globalement les défis liés à la gestion des données dans des systèmes embarqués.

Annexe

A. Mise en place de l'environnement de travail

A.1 Installation de Node-RED par étapes

Pour explorer la communication bidirectionnelle de manière plus visuellement agréable, avec le protocole HTTP REST (envoie requête serveur) MQTT (intermédiaire d'un serveur Broker gérant la réception et renvoie des sujets) et nous aurons besoin d'un serveur ici Node-RED installable tel que :

1) Installation de Node.js depuis : https://nodejs.org/dist/v20.17.0/node-v20.17.0-x64.msi En cochant l'installation automatique des outils.

2) Vérification de l'installation depuis un terminal de commande : node -v; npm -v
3) Installation de Node-RED depuis le terminal de commande : npm install -g --unsafe-perm node-red
4) Lancement depuis le terminal de commande avec : node-red
5) Ouverture du lien dans un navigateur sans fermer le terminal de commande : http://127.0.0.1:1880/

6) Installation du Dashbord : Gérer la palette-> installer ->DashBordNodeRed

A.2 Installation de Mosquitto-BROKER (serveur) par étapes

Pour utiliser le protocole de messagerie MQTT (ayant pour principe de clients abonnés auprès d'un serveur à des publications reçus par d'autres clients, appelé *Publish-subscribe*) nous explorerons Mosquitto en tant que Broker tel que :

1) Installation de mosquitto depuis : https://mosquitto.org/files/binary/win64/mosquitto-2.0.18a-install-windows-x64.exe

2) Ouverture depuis le menu démarrer de Windows de l'application « service » pour démarrer Mosquitto Broker.

3) Pour vérifier le fonctionnement de Mosquitto après avoir ajouté le chemin de celui-ci dans la variable PATH (a - depuis le menu démarrer de Windows de l'application « afficher les paramètres système avancées » b - en bas à droite « Variables d'environnement ... » c - dans « Variables système » copier le chemin de

MOSQUITTO_DIR : C:\Program Files\mosquitto d – créer un fichier bloc note vide de configuration « mosquitto.conf » avec deux lignes : « listena 1883 \nallow_anonymous » e - ajouter la copie dans le chemin de PATH) depuis un terminal de commande : mosquitto -v -c "C:\Users\vladi\Desktop\Elec\IoT\mosquitto.conf"

4) Pour vérifier l'interaction de manière plus pertinente avec notre broker :

a) Nous allons créer deux clients pour simuler les deux rôles clés du protocole MQTT (abonné/subscriber et publieur/publisher).

i) Pour créer le premier client abonné depuis un premier terminal de commande : mosquitto_sub -h localhost -t "capteur/ruche_1/humidite"

ii) Pour créer le deuxième client, cette fois ci publieur, dans un deuxième terminal de commande : C:\Users\vladi> mosquitto_pub -h localhost -t "capteur/ruche_1/humidite" -m "R1 - 98%"

b) Dans le cas où l'on voudrait s'abonner à plusieurs sujets, on pourrait utiliser « # » comme pour s'abonner aux relevés des capteurs de toutes les ruches par exemple avec le sujet : mosquitto_sub -h localhost -t "capteur/#"

5) Installation librairie PubSubClient:
https://github.com/knolleary/pubsubclient/archive/master.zip

A.3 Configuration de Arduino IDE pour échanger avec l'ESP

1) Installation de Arduino IDE depuis :

https://downloads.arduino.cc/arduino-ide/arduino-

2) Dans Arduino IDE, indiquer le module à utiliser depuis File->Preferences...-> et indiquer dans « Additional boards manager URLs : » l'url depuis recherche internet «esp8266 url for arduino ide » : https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json / https://dl.espressif.com/dl/package_esp32_index.json, http://arduino.esp8266.com/stable/package_esp8266com_index.json

sur le site https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/ ou celui de l'ESP8266

3) Installation de la carte ESP depuis Tools->Board->Boards Manager... : « Arduino Nano ESP32 » et dans notre cas ESP8266

4) Après avoir relié le capteur avec le microcontrôleur, Tools->Board : Node MCU 1.0 (ESP-12E Module) -> esp8266 -> Node MCU 1.0 (ESP-12E Module)

5) Installation librairie DTH :

a) https://www.gotronic.fr/art-capteur-d-humidite-et-de-t-grove-101020011-18963.htm

b) fiche technique : http://wiki.seeedstudio.com/Grove-TemperatureAndHumidity_Sensor/

c) « **Software**

- **Step 1.** Download the Seeed DHT library » -> code -> download zip

d) Inclusion de la librairie téléchargée : Sketch -> Include Library -> Add .ZIP Library... -> Grove_Temperature_And_Humidity_Sensor-master.zip

5) Installation du driveur (si le port n'est pas détecté) depuis
https://randomnerdtutorials.com/getting-started-with-esp8266-wifi-transceiver-review/ ->
https://randomnerdtutorials.com/install-esp32-esp8266-usb-drivers-cp210x-windows/ ->
https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads ->
https://www.silabs.com/documents/public/software/CP210x_Windows_Drivers.zip

6) Connecter le Port : Tools -> Port...-> COM3