

# 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

Introduction .....	3
1. Test du matériel - Test unitaires .....	4
1.1. Test du capteur DHT11 .....	4
Figure 1.1.1 : Extrait de code montrant le test d'une variable et sa restitution.....	4
Figure 1.1.2 : Vu du Serial Monitor du test capteur pour la variable humidité.....	4
1.2. Test serveur Node-RED .....	5
Figure 1.2.1 : Schéma des flux du serveur Node-RED pour tester un dashboard..	5
Figure 1.2.2 : Extrait de code dans le bloc fonction pour tester l'interface visuel.	5
Figure 1.2.3 : Interface Utilisateur visualisant la réaction à la stimulation du serveur.....	5
1.3. Test de la LED ESP8266 .....	6
Figure 1.3.1 : Code succinct pour le module Arduino faisant clignoter la LED Bleu périodiquement. ....	6
Figure 1.3.2 : Photo du microcontrôleur LED allumé puis éteinte. ....	6
2. Montée des données - HTTP REST .....	7
Figure 2.1 : Extrait de code de l'envoi de la requête par la méthode POST au format JSON. ....	7
Figure 2.2 : Schéma des flux du serveur Node-RED pour la montée des données en HTTP REST.....	8
Figure 2.3 : Extrait de code dans le bloc fonction pour scinder la donnée de température et celle d'humidité. ....	8
3. Descente des données - HTTP REST .....	10

Figure 3.1 : Extrait de code de l'envoi de la requête pour récupérer dans payload de la consigne depuis l'interrupteur du dashboard. ....	10
Figure 3.2 : Schéma des flux du serveur Node-RED pour la descente des données en HTTP REST. ....	11
<b>4. Montée-Descente des données - MQTT .....</b>	<b>12</b>
Figure 4.1 : Extrait de code de la publication des relevés dans les sujets-journaux Température et Humidité pour le protocole MQTT. ....	12
Figure 4.2 : Schéma des flux du serveur Node-RED pour la montée des données en MQTT. ....	13
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. ....	14
Figure 4.5 : Schéma des flux du serveur Node-RED pour la descente des données en MQTT. ....	14
Figure 4.6 : Interface Utilisateur visualisant descente des données en MQTT. ...	15
<b>Annexe .....</b>	<b>16</b>
<b>A. Mise en place de l'environnement de travail .....</b>	<b>16</b>
A.1 Installation de Node-RED par étapes.....	16
A.2 Installation de Mosquitto-BROKER ( serveur ) par étapes .....	16
A.3 Configuration de Arduino IDE pour échanger avec l'ESP .....	17

## 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.

Pour notre initiation nous utiliserons le serveur généré avec Node-RED pour l'envoi des requêtes par le protocole HTTP REST, visuellement attrayant et Mosquitto en tant qu'intermédiaire appelé Broker pour le protocole MQTT. Initialisation disponible en en annexe [A. Mise en place de l'environnement de travail](#).



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

## 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 le capteur, puis le client wifi et enfin nous allons essayer de connecter le client wifi à nos adresses.

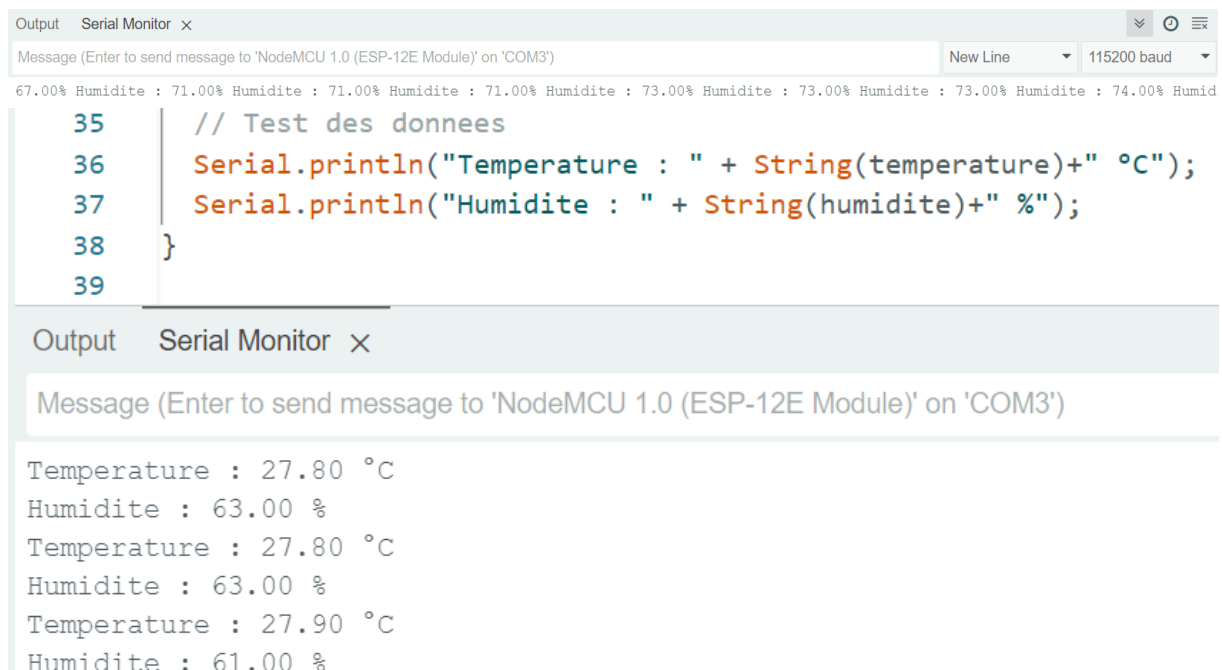
### 1.1. Test du capteur DHT11

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.

Figure 1.1.1 : Extrait de code montrant le test d'une variable et sa restitution.

```
24   float humidite = dht.readHumidity();
35   // Test des donnees
36   Serial.print("Humidite : ");
37   Serial.print(humidite);
38   Serial.print("% ");
```

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



The screenshot shows the Serial Monitor interface with the following code and output:

```
35   // Test des donnees
36   Serial.println("Temperature : " + String(temperature)+" °C");
37   Serial.println("Humidite : " + String(humidite)+" %");
38   }
39
```

The output shows the following data being printed:

```
67.00% Humidite : 71.00% Humidite : 71.00% Humidite : 71.00% Humidite : 73.00% Humidite : 73.00% Humidite : 73.00% Humidite : 74.00% Humid
```

```
Temperature : 27.80 °C
Humidite : 63.00 %
Temperature : 27.80 °C
Humidite : 63.00 %
Temperature : 27.90 °C
Humidite : 61.00 %
```

## 1.2. Test serveur Node-RED

D'autre part, pour tester le serveur, nous utilisons un simple GET avec dans le terminal « PS C:\Users\vladi> curl http://127.0.0.1:1880/test » relié au serveur tel que sur le schéma ci-dessous.

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

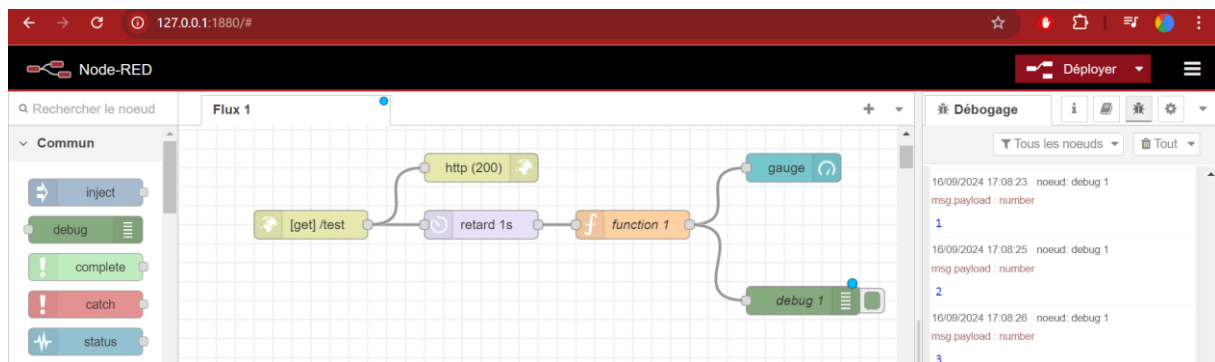


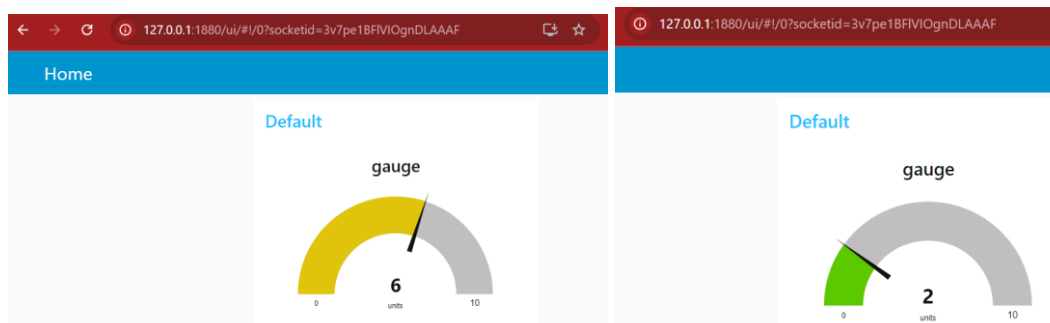
Figure 1.2.2 : Extrait de code dans le bloc fonction pour tester l'interface visual.

```

1  let test = global.get("test") || 0;
2
3  if (test >= 10 )
4  {test = 0 ;}
5  test += 1;
6  global.set("test", test); //Maj test
7
8  msg.payload = test; //Noeud
9
10 return msg;
11

```

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



Ainsi nous venons de conclure du bon fonctionnement du serveur.

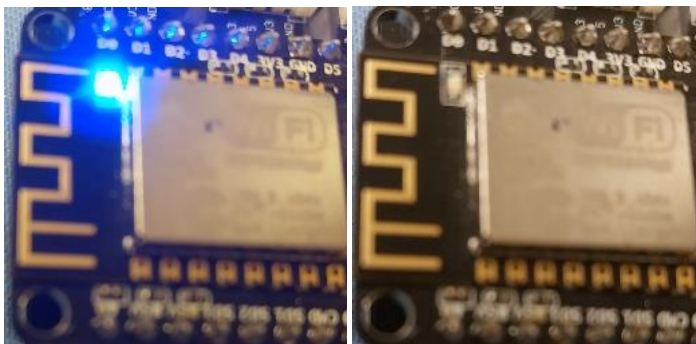
### 1.3. Test de la LED ESP8266

Enfin pour vérifier la descente des données, nous serons amené à utiliser la LED du module avec Arduino.

*Figure 1.3.1 : Code succinct pour le module Arduino faisant clignoter la LED Bleu périodiquement.*

```
Test_LED_clignotement.ino
1  void setup()
2  {
3      pinMode (LED_BUILTIN, OUTPUT); //Initiation de la led en tant que sortie
4  }
5
6  // Boucle pour la lecture des donnees
7  void loop()
8  {
9      // put your main code here, to run repeatedly:
10
11     digitalWrite(LED_BUILTIN, HIGH);
12     delay(500); //En ms = 0.5s
13     digitalWrite(LED_BUILTIN, LOW);
14     delay(500); //En ms = 0.5s
15
16 }
17
```

*Figure 1.3.2 : 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 : Extrait de code de l'envoi de la requête par la méthode POST au format JSON.*

```
69 http.begin(client, "http://" + SERVER_IP + "/donnees/"); // HTTP
70 http.addHeader("Content-Type", "application/json");
71
72 String postData = "{\"temperature\": " + String(temperature) + ", \"humidite\": " + String(humidite) + "}";
73
74
75 Serial.print("[HTTP] POST...\n"); //Debut d envoie POST
76 // start connection and send HTTP header and body
77
78 // Envoie de la requete POST avec les donnees JSON et recupere le CODE 200 ou autre si erreur
79 int httpCode = http.POST(postData);
```

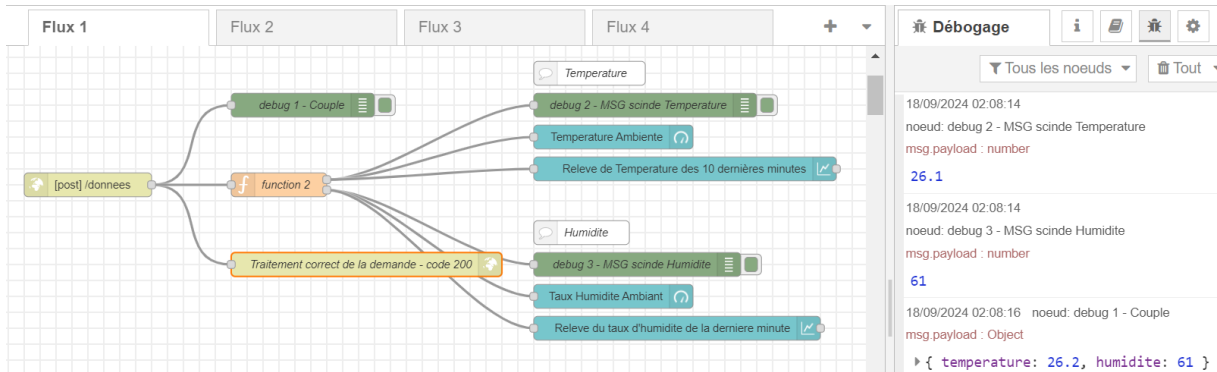
Nous aurons besoin d'une fonction pour diviser les sorties en 2, l'une pour les relevés d'humidité et l'autre pour les relevés de température. Les données seront ensuite visualisées depuis des dashboards au lien suivant :

<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.

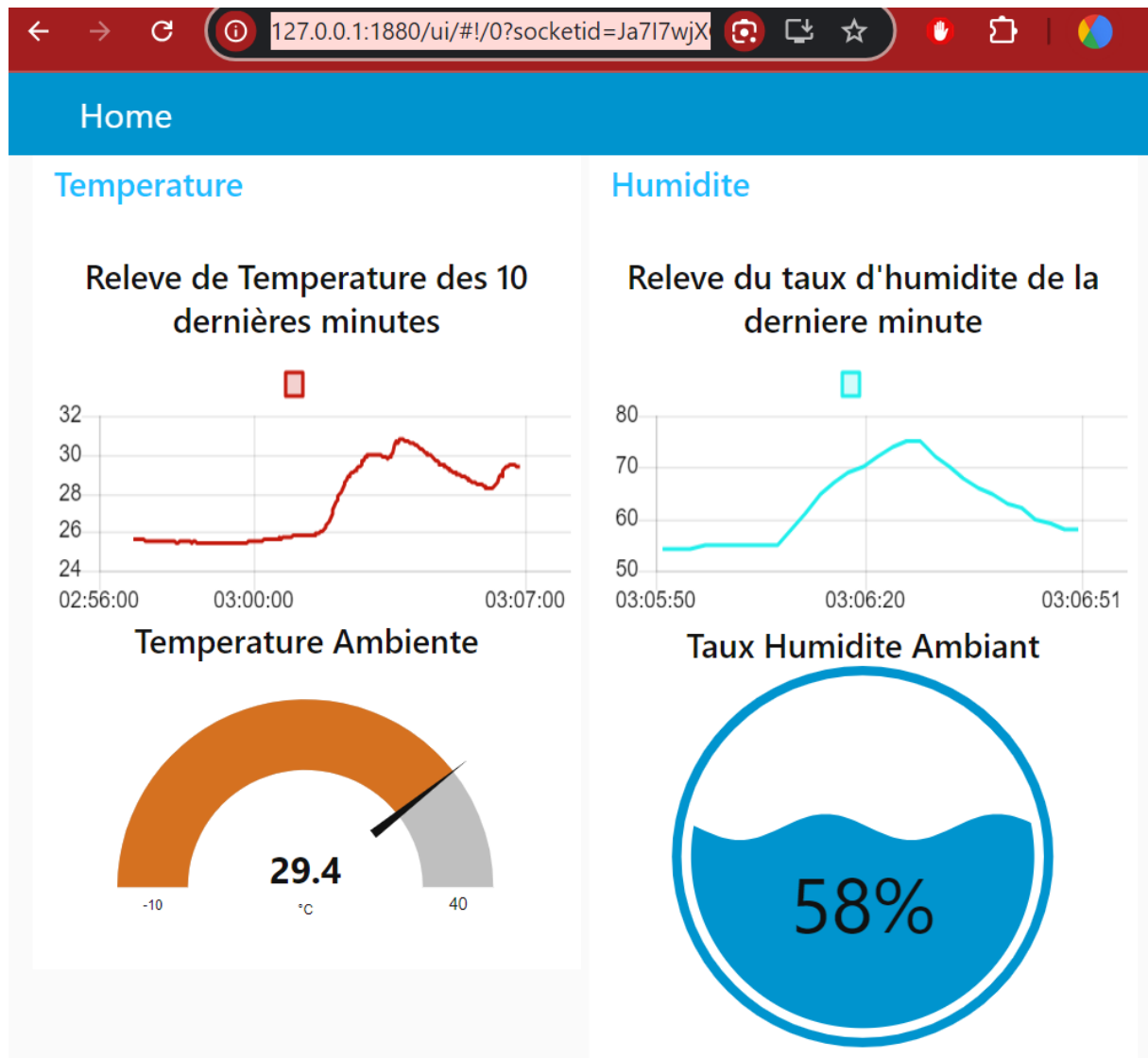
```

4 // Creation de variable global pour meilleur lisibilite
5 var temperature_msg_up = { payload: msg.payload.temperature };
6 var humidite_msg_down = { payload: msg.payload.humidity };
7
8 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 montée 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 : Extrait de code de l'envoi de la requête pour récupérer dans payload de la consigne depuis l'interrupteur du dashboard.

```

34     Serial.print("[HTTP] debut de la requete GET...\n");
35     //Configuration de l URL du serveur
36     http.begin(client, "http://" SERVER_IP "/"Led"); //URL pour la requete GET
37
38     //Envoi de la requete GET
39     int httpCode = http.GET();
40
41     if (httpCode > 0) {
42         Serial.printf("[HTTP] GET... code: %d\n", httpCode);
43
44         //Si la reponse du serveur est OK (HTTP 200)
45         if (httpCode == HTTP_CODE_OK) {
46             payload = http.getString(); //Recuperation reponse du serveur
47             Serial.println("Reponse du serveur : ");
48             Serial.println(payload);
49         }

```

Output Serial Monitor ×

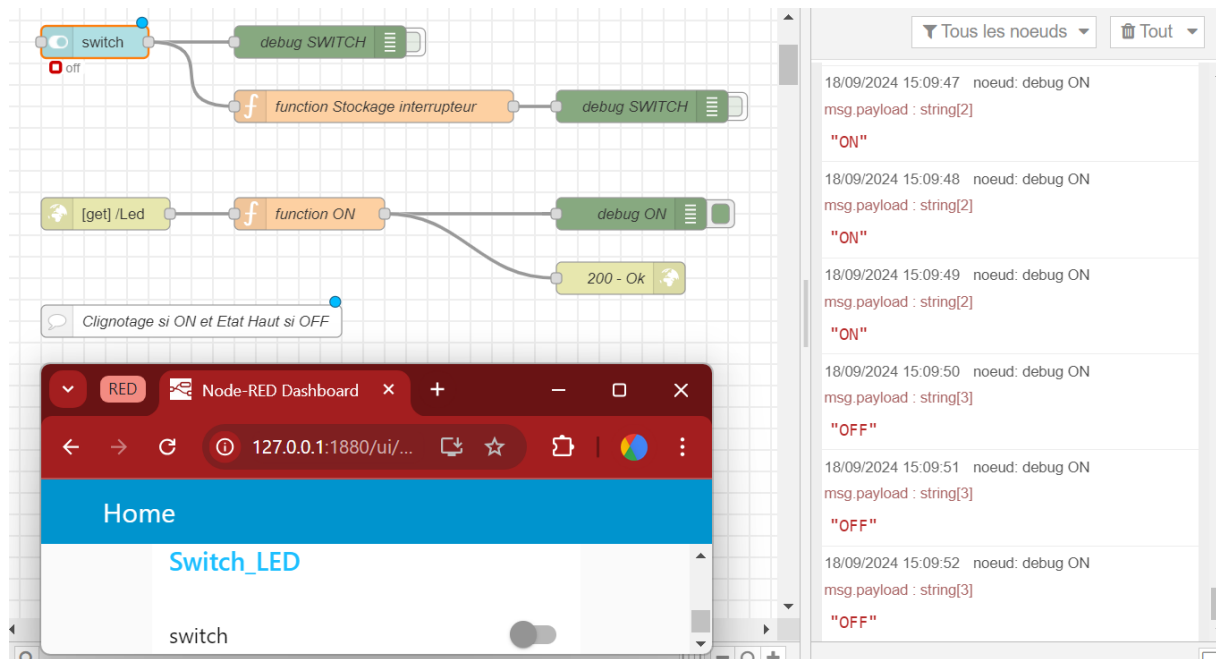
Message (Enter to send message to 'NodeMCU 1.0 (ESP-12E Module)' on 'COM3')

```

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...

```

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



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. Cela valide la descente des données et conclue le protocole http REST.

## 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 messenger recevant les informations-sujet des capteurs et les redistribuant à tous les abonnés 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.

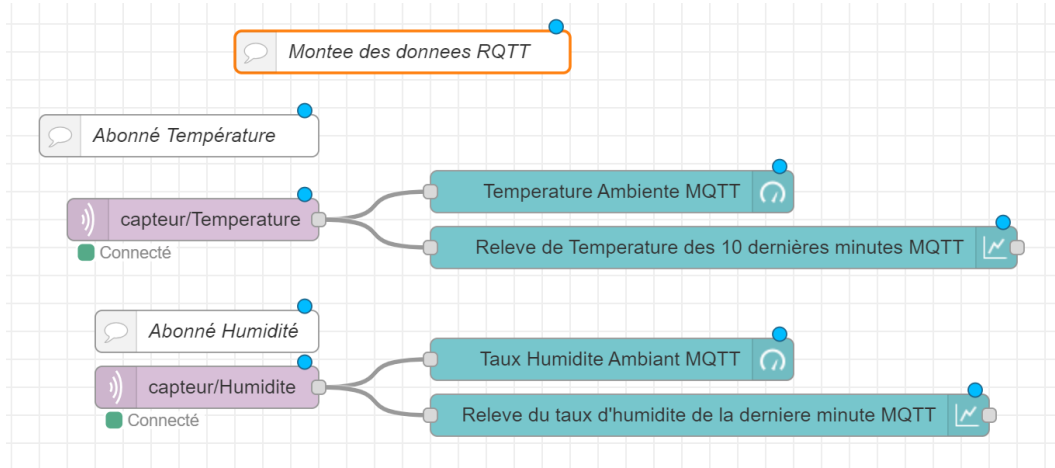
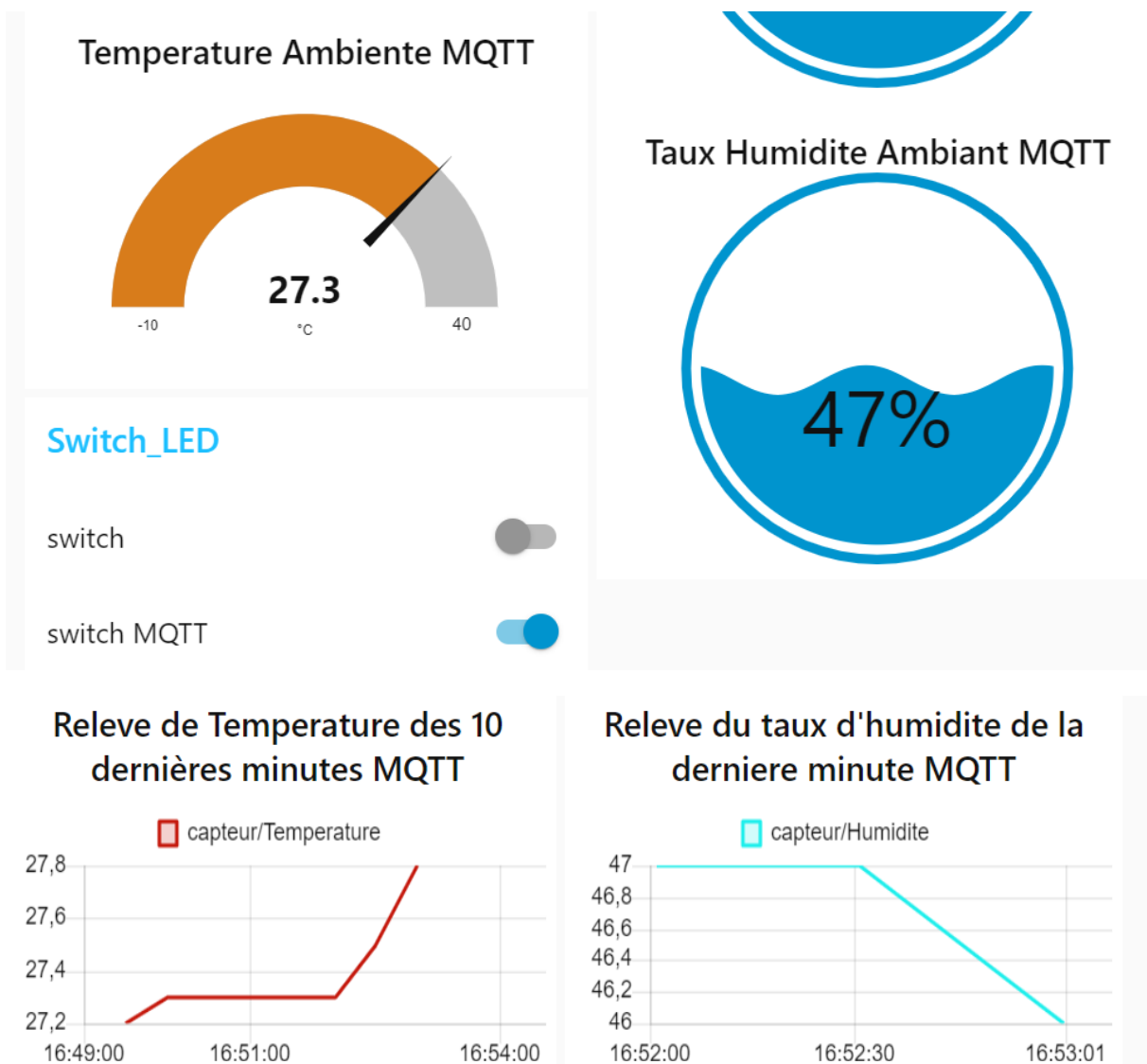


Figure 4.3 : Interface Utilisateur visualisant montée des données en MQTT.



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.

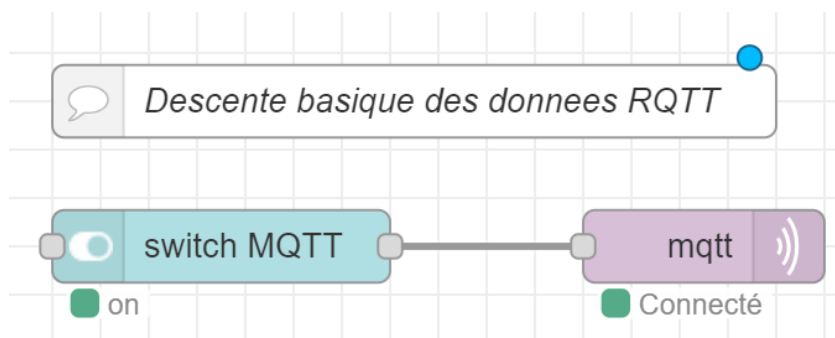


Figure 4.6 : Interface Utilisateur visualisant descente des données en MQTT.

## Switch\_LED

switch



switch MQTT



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.



## 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 : « `listen 1883` » et « `allow_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-ide.exe>

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/esp8266/Arduino/master/packages/esp8266/package\\_esp8266\\_index.json](https://raw.githubusercontent.com/esp8266/Arduino/master/packages/esp8266/package_esp8266_index.json) / [https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json), [http://arduino.esp8266.com/stable/package\\_esp8266com\\_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/](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](https://www.silabs.com/documents/public/software/CP210x_Windows_Drivers.zip)

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