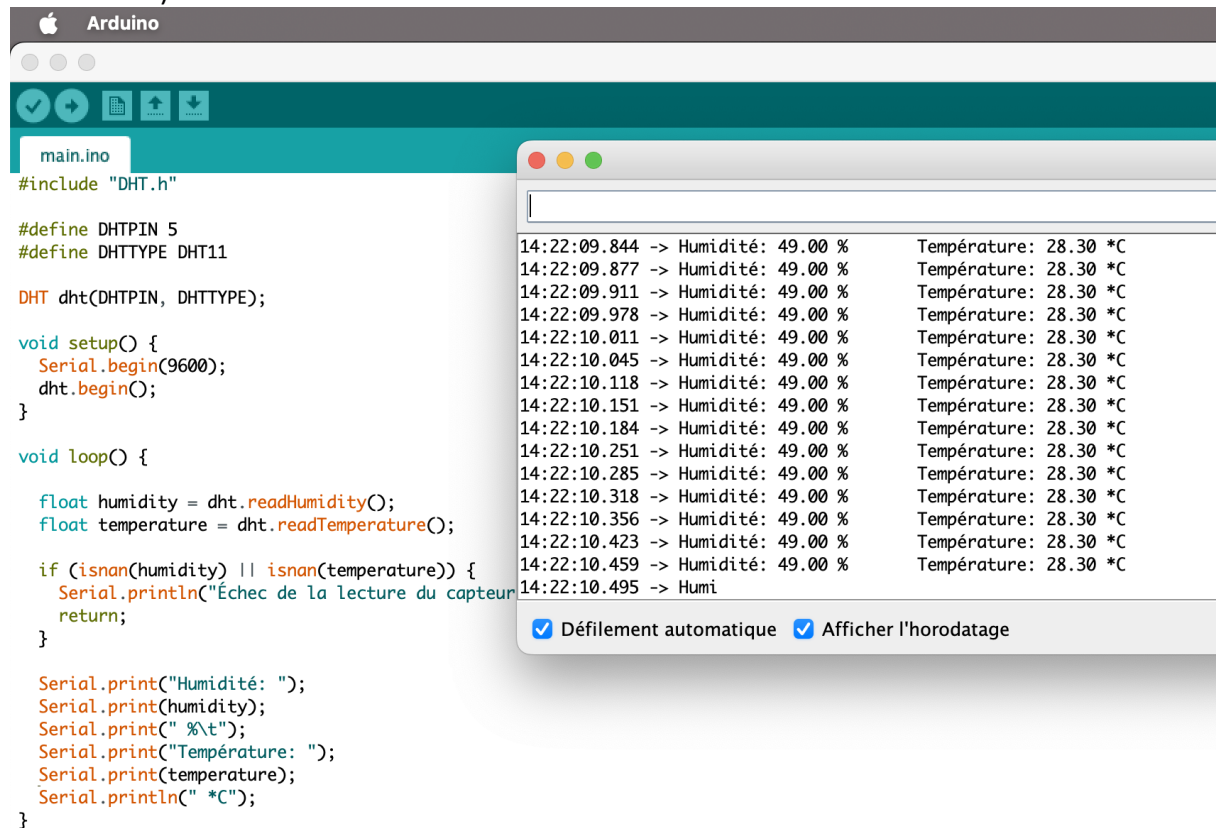


Compte rendu TP1 IOT : HTTP REST et MQTT avec ESP32, DHT11 et Node-RED

MAHON Grégoire
LELONG Armand
EI2I-4-II (Groupe B)

Partie 3 : Communication HTTP REST

Connexion du capteur DHT11 à la carte ESP32 (après avoir installé les librairies nécessaires) et lecture des données :



```
main.ino
#include "DHT.h"

#define DHTPIN 5
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();
}

void loop() {

  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Échec de la lecture du capteur");
    return;
  }

  Serial.print("Humidité: ");
  Serial.print(humidity);
  Serial.print(" %\t");
  Serial.print("Température: ");
  Serial.print(temperature);
  Serial.println(" *C");
}
```

Serial Monitor Output:

Time	Humidity	Temperature
14:22:09.844	49.00 %	28.30 *C
14:22:09.877	49.00 %	28.30 *C
14:22:09.911	49.00 %	28.30 *C
14:22:09.978	49.00 %	28.30 *C
14:22:10.011	49.00 %	28.30 *C
14:22:10.045	49.00 %	28.30 *C
14:22:10.118	49.00 %	28.30 *C
14:22:10.151	49.00 %	28.30 *C
14:22:10.184	49.00 %	28.30 *C
14:22:10.251	49.00 %	28.30 *C
14:22:10.285	49.00 %	28.30 *C
14:22:10.318	49.00 %	28.30 *C
14:22:10.356	49.00 %	28.30 *C
14:22:10.423	49.00 %	28.30 *C
14:22:10.459	49.00 %	28.30 *C
14:22:10.495	49.00 %	28.30 *C

Serial Monitor Settings: Défilement automatique (checked), Afficher l'horodatage (checked)

Figure 1 : Lecture de la température et de l'humidité sur le port série

Nous avons ensuite récupéré les données et nous les avons mises en forme dans des graphiques « UI Chart » :

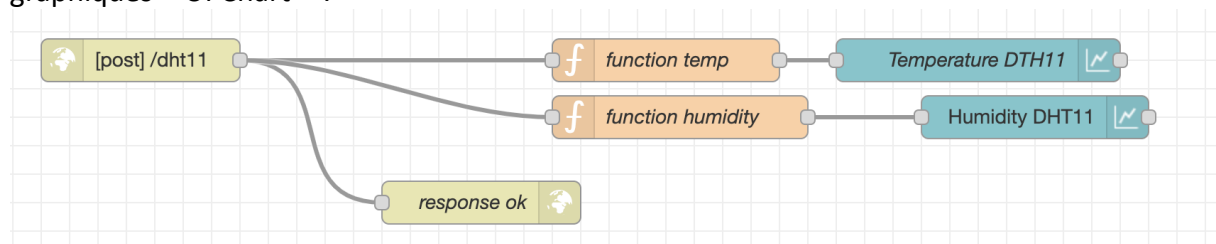
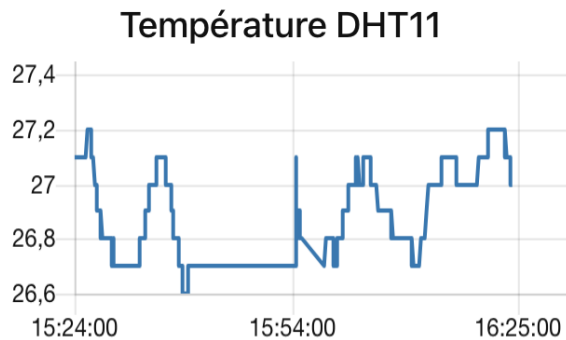


Figure 2 : Flux http humidité et température

TP1 Temperature DHT11



TP1 Humidity DHT11

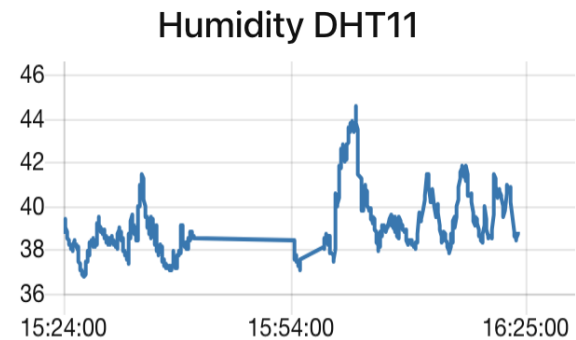


Figure 3 : Graphiques d'humidité et température http

Cela nous permet de suivre les données lues par le DHT11 en « temps réel ». Cependant il faut tout de même prendre en compte le temps de la requête et de la réponse.

Par la suite, l'objectif est d'allumer la led de la carte esp32 en fonction de l'état d'un switch « UI Switch » que nous avons ajouté sur notre flux.

Pour cela, notre programme envoie une requête GET /ledState qui requête à node-red l'état du switch. Ce switch renvoie un booléen (true ou false en fonction de son état).

On enregistre ensuite l'état de ce switch dans une variable globale « led » qui permet de connaître cet état dans d'autres palettes du flux. Il suffit ensuite de caster l'état booléen du switch en string (pour qu'il soit accepté en tant que corps de réponse http). Pour cela, on utilise la méthode toString().

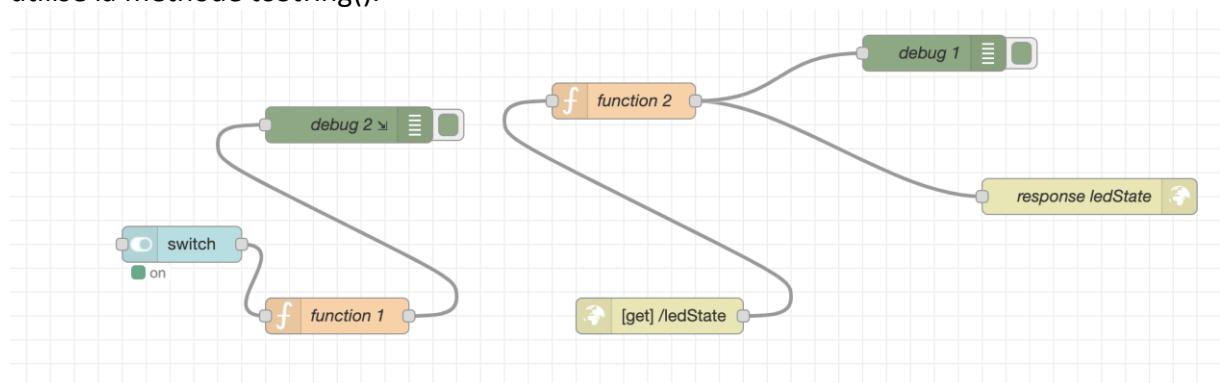


Figure 4 : Flux requête GET pour l'état de la LED en fonction du switch

PS : Les palettes « debug » permettent d'afficher les informations de sortie des palettes concernées. Cela est très utile pour le débogage.

Voici les étapes que nous avons suivi pour la réalisation de cette partie :

1. **Connexion au réseau WiFi** : Initialisation de la connexion de l'ESP32 au réseau sans fil en utilisant les identifiants SSID et mot de passe.
2. **Lecture des données du capteur DHT11** : Configuration et démarrage du capteur DHT11 pour la lecture des valeurs de température et d'humidité.
3. **Envoi des données via HTTP** : Formatage et envoi des données lues à Node-RED à l'aide d'une requête HTTP POST.
4. **Traitement des données dans Node-RED** : Configuration de Node-RED pour recevoir les données HTTP, les traiter et les préparer pour l'affichage.
5. **Affichage des données** : Mise en place des graphiques "UI Chart" dans Node-RED pour visualiser les données en temps réel.
6. **Contrôle de la LED** : Envoi de l'état d'un switch "UI Switch" à l'ESP32 par une requête http « /ledState » pour contrôler la LED de l'esp32.
7. **Débogage** : Utilisation des outils de débogage de Node-RED pour surveiller et valider le flux de données.

Partie 4: Communication MQTT

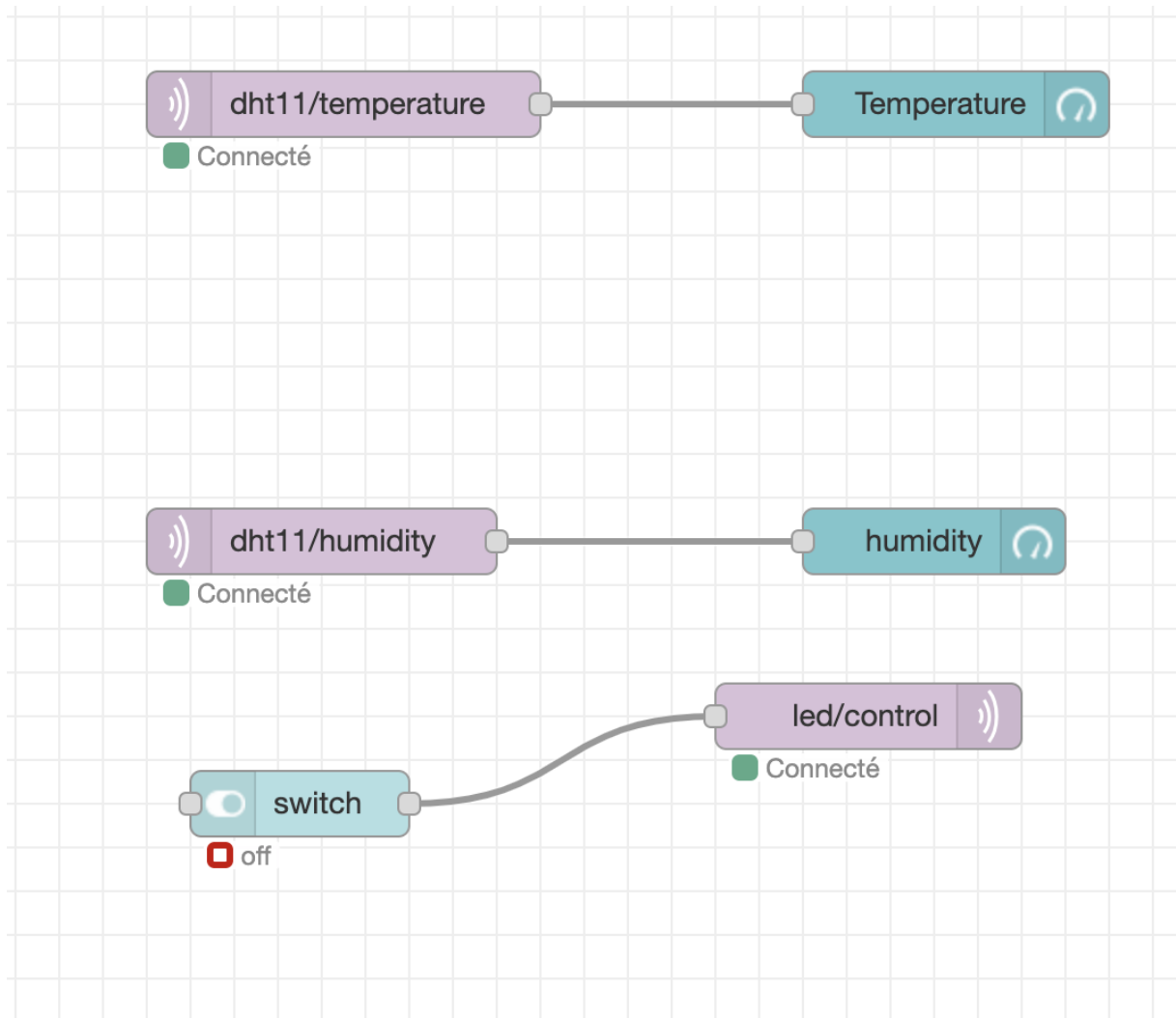


Figure 5 : Flux mqtt de la température

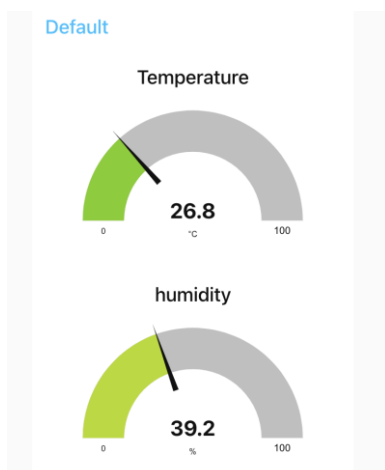


Figure 6 : Graphes de température et d'humidité

On a ensuite réalisé le flux pour le mqtt ainsi que le code arduino.

Cela nous permet d'afficher via des jauges les valeurs de température et d'humidité lues par le capteur DHT11.

Nous avons également ajouté, comme pour la première partie (http), un switch qui permet de contrôler l'état de la led de notre carte.

Cela est très satisfaisant par rapport au protocole http car les temps de réponses sont beaucoup plus rapides.

Voici chacune des étapes que nous avons suivi pour réaliser cette partie :

1. **Connexion au Broker MQTT** : L'ESP32 établit une connexion MQTT en envoyant un paquet CONNECT au broker Mosquitto. Cette étape inclut l'authentification et la négociation des paramètres de session. Si la connexion est perdue, l'ESP32 utilise un mécanisme de reconnexion pour restaurer la session (cf code).
2. **Publication des données** : Les mesures de température et d'humidité sont envoyées au broker en utilisant le protocole MQTT. Chaque donnée est convertie en chaîne de caractères et publiée sur son topic MQTT respectif à un intervalle défini, assurant une mise à jour régulière des valeurs.
3. **Contrôle de la LED via MQTT** : L'ESP32 reçoit des commandes MQTT pour le contrôle de la LED. Lorsqu'un message est reçu sur le topic dédié, la fonction de rappel (callback) interprète la commande et change l'état de la LED intégrée à la carte.
4. **Intégration avec Node-RED** : Dans Node-RED, les nœuds MQTT sont configurés pour interagir avec le broker. Ils permettent de visualiser les données en temps réel et de créer des interfaces pour envoyer des commandes de contrôle à l'ESP32.
5. **Débogage** : L'utilisation des nœuds de débogage dans Node-RED aide à identifier les problèmes en affichant les données MQTT entrantes et sortantes, facilitant ainsi la correction des erreurs dans le flux de données.
6. **Flux MQTT** : Le flux MQTT est visualisé dans Node-RED par des liaisons entre les nœuds de souscription aux topics concernés, de traitement des données, et de publication, illustrant le chemin parcouru par les données depuis l'ESP32 jusqu'à l'affichage utilisateur et vice versa pour les commandes de contrôle.

Nous avons cependant connu quelques problèmes avec la connexion de l'esp32 au broker, il a donc fallu modifier les fichiers de configuration de mosquitto en suivant les remarques de notre chargé de TP.

Comparatif protocoles MQTT et HTTP :

MQTT est un protocole de messagerie léger conçu pour les réseaux à faible bande passante et des appareils aux ressources limitées. Il est basé sur le modèle de publication-abonnement, ce qui le rend extrêmement réactif. Dès qu'un message est publié sur un topic auquel un client est abonné, le message est immédiatement transmis au client. Cette communication quasi instantanée permet une mise à jour en temps réel des états, ce qui est crucial pour les systèmes de contrôle et de surveillance en temps réel.

HTTP REST, bien qu'universel et facile à utiliser, est un protocole basé sur la demande-réponse qui nécessite un client pour initier des requêtes pour obtenir des mises à jour. Cela peut introduire une latence car le client doit périodiquement vérifier les mises à jour, ce qui est moins efficace que la transmission de messages en temps réel de MQTT.

De plus, MQTT conserve les sessions des clients, ce qui signifie qu'il peut gérer des interruptions de connexion de manière transparente avec des mécanismes comme les messages de dernier testament (Last Will and Testament) et la reprise de session. En comparaison, HTTP est sans état et chaque requête est indépendante, ce qui peut entraîner une surcharge supplémentaire pour maintenir l'état à travers les sessions.

En résumé, pour les applications IoT nécessitant une interaction en temps réel et des mises à jour fréquentes des données, MQTT est plus adapté que HTTP REST en raison de sa conception basée sur les abonnements, sa légèreté et sa capacité à gérer efficacement les sessions persistantes. Il est cependant plus difficile à mettre en place pour les débutants.