Mike Wilfried TAMBA
Bastien CARRÉ – CAZÉ

SAE – IOT

1ère Partie : Badge électronique

Objectif : Étudier les processus de lecture / écriture d'étiquettes électronique (RFID) à partir d'un module RC522 couplé à un ESP32.

Question de cours :

Qu'est-ce qu'un objet connecté ?

Un objet connecté est un objet ordinaire, comme une montre, un bracelet, un réfrigérateur ou un jouet, qui a la capacité de communiquer des informations diverses à un autre objet ou à Internet. Il peut capter, transmettre et parfois traiter des données pour aider à la décision ou enclencher une action. L'objet peut avoir sa propre fonctionnalité mécanique ou électrique et est conçu pour communiquer, souvent une simple information, comme la détection de la lumière ou une mesure de température.

Qu'est-ce que l'iot ?

L'Internet des objets (IoT) est un réseau de terminaux physiques, appelés « objets », qui intègrent des capteurs, des logiciels et d'autres technologies pour se connecter à d'autres terminaux et systèmes sur Internet et échanger des données avec eux. Ces objets peuvent être des appareils domestiques simples ou des outils industriels complexes. L'IoT relie tout élément capable de transférer des données sur un réseau, sans nécessiter d'interactions entre humains ou entre un humain et un ordinateur. Cependant, l'interaction personne-machine est rendue possible, par exemple pour le paramétrage, la configuration ou l'accès aux informations.

Quels sont les principaux protocoles réseaux de l'iot ?

Les principaux protocoles réseaux de l'IoT incluent plusieurs options qui répondent à différents besoins en fonction de la distance, de la consommation d'énergie, et de la quantité de données à transférer. Un réseau IoT désigne un ensemble de dispositifs interconnectés qui communiquent avec d'autres dispositifs sans intervention humaine. Il peut utiliser un réseau cellulaire, local, étendus ou maillés.

Annexe

La technologie NFC fonctionne à une fréquence de 13,56 MHz. Cette fréquence est spécifique à la sous- catégorie des RFID à haute fréquence (HF), ce qui la distingue des autres sous-catégories RFID qui fonctionnent à des fréquences différentes. La fréquence de 13,56 MHz est choisie pour ses avantages techniques, notamment sa capacité à être librement utilisée dans le monde entier et son débit de communication intéressant.

Mike Wilfried TAMBA BUT3 GEII
Bastien CARRÉ – CAZÉ

Le principe de fonctionnement de la technologie RFID (Radio Frequency Identification) **repose sur l'échange d'informations entre un lecteur et des tags ou étiquettes RFID**. Voici les étapes clés de ce processus :

1. **Émission d'ondes électromagnétiques** : Le lecteur émet des ondes électromagnétiques qui induisent un courant dans l'antenne du tag RFID. Ce courant alimente la puce électronique du tag.
2. **Transmission des données** : La puce du tag, alimentée par ce courant, peut alors transmettre les informations stockées vers le lecteur. Cette communication peut se faire par rétromodulation, où le signal radio émis par le lecteur est partiellement réfléchi par le tag avec une modulation pour transmettre les données.
3. **Interprétation des données** : Le lecteur reçoit ces données et les interprète, puis les transmet à un système de gestion de données pour traitement et stockage.

Il existe différents types de tags RFID, notamment les tags passifs, actifs et semi-passifs :

- **Tags passifs** : Ces tags n'ont pas de batterie et fonctionnent grâce à l'énergie transférée par électromagnétisme. Ils sont utilisés dans des applications où la portée de lecture est limitée, généralement de quelques centimètres à plusieurs dizaines de centimètres.
- **Tags actifs** : Ces tags sont munis d'une batterie pour émettre sur de plus longues distances et enregistrer de nouvelles données. Ils sont utilisés pour des applications nécessitant une portée de lecture plus grande, pouvant atteindre plusieurs mètres.
- **Tags semi-passifs** : Ces tags n'envoient pas de signaux sauf si activation par un lecteur RFID, mais ils utilisent une batterie pour augmenter la portée du signal retourné du tag vers le lecteur.

La technologie RFID utilise différentes gammes de fréquences, notamment les basses fréquences (LF), hautes fréquences (HF) et ultra-hautes fréquences (UHF), chacune ayant ses propres avantages en termes de portée de lecture et de vitesse de transmission des données.

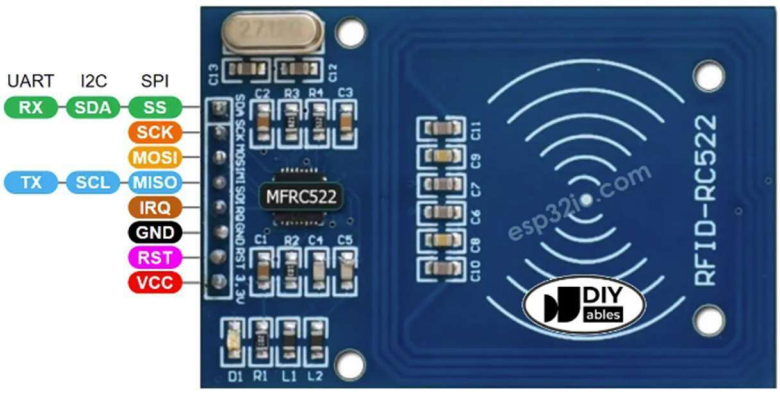
Lecture du badge RFID : Le lecteur RFID émet des ondes radio à une fréquence spécifique (par exemple, 13,56 MHz pour la technologie NFC). Le badge RFID, qui est une puce passive sans batterie, capte ces ondes pour générer une petite quantité d'énergie qui alimente la puce et active la communication. Le badge répond alors au lecteur avec son identifiant unique (UID) et d'autres données stockées.

Écriture sur le badge RFID : Pour écrire des données sur le badge, le lecteur RFID envoie des instructions spécifiques à la puce RFID. Ces instructions peuvent inclure la modification des données dans les différents secteurs du badge, comme le secteur 0 qui contient généralement l'UID. Cependant, certains secteurs peuvent être verrouillés en lecture seule pour des raisons de sécurité.

Mike Wilfried TAMBA BUT3 GEII
Bastien CARRÉ – CAZÉ

Après l'étude théorique de la technologie RFID, on va connecter un lecteur RFID a une carte Arduino pour contrôler les informations entrantes et sortantes du capteur.

Brochage

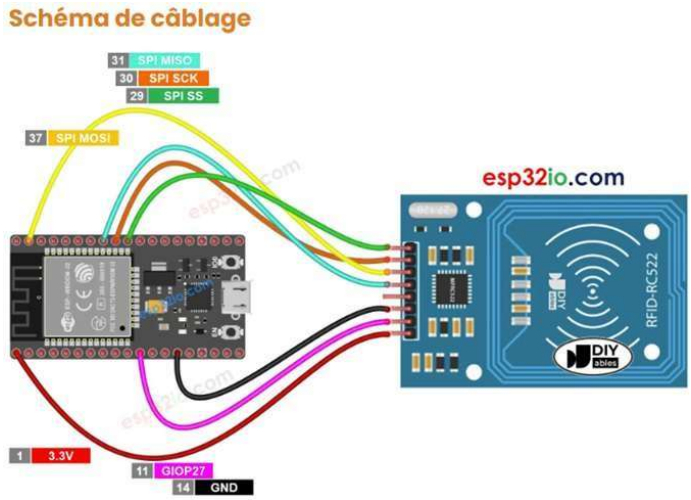


Le module RFID est composé de 8 broches, nous pouvons constater qu'il y a une broche qui est partagée en trois interfaces de communication : SPI, I2C, UART. Dans notre cas, nous utilisons l'interface SPI pour communiquer avec l'ESP32. Les broches sont :

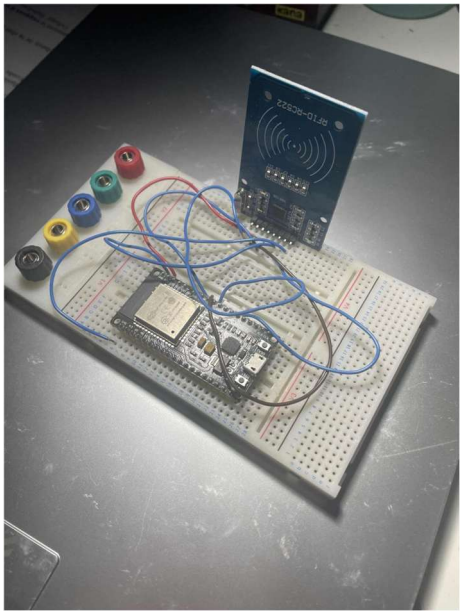
- GND : connecter cette broche au GND de l'ESP32 (0V)
- VCC : connecter cette broche au 3.3V de l'ESP32 (3.3V)
- RST : broche pour la réinitialisation et la mise hors tension. Lorsque cette broche est activée, elle met le module RFID dans un état de démarrage propre, annulant ainsi toutes les configurations précédentes et ramenant le module à ses paramètres par défaut.
- IRQ : cette broche peut être utilisée pour alerter l'ESP32 lorsque l'étiquette RFID entre dans sa plage de détection (Cependant dans notre cas, elle est inutilisé)
- MISO/SCL/TX :
    - Broche MISO : C'est la broche par laquelle le périphérique esclave envoie les données au périphérique maître.
    - Broche SCL si l'interface I2C est activée
    - Broche TX si l'interface UART est activée
- MOSI : C'est la broche par laquelle le périphérique maître envoie les données au périphérique esclave.
- SCK : C'est l'horloge générée par le périphérique maître pour synchroniser la transmission des données.
- SS/SDA/RX :
    - SS : Cette broche est utilisée par le périphérique maître pour sélectionner le périphérique esclave avec lequel il souhaite communiquer
    - SDA si l'interface I2C est activée
    - UART si l'interface

Lorsqu'une transmission de données SPI se produit, le périphérique maître active l'horloge (SCLK) et envoie des bits de données via la broche MOSI, tandis que le périphérique esclave envoie ses données en utilisant la broche MISO. Les deux périphériques doivent être configurés avec des paramètres compatibles en termes de fréquence d'horloge, de mode de transmission (phase et polarité), etc.

Mike Wilfried TAMBA BUT3 GEII
Bastien CARRÉ – CAZÉ



A l'aide d'un logiciel en ligne on établi le schéma de câblage ci dessus.
On câble donc le montage suivant



On commence par tester le lecteur avec un exemple qui va lire l'UID de la carte qui est un identifiant unique attribué à la carte par défaut. Pour cela on utilise le programme suivant :

```
#include <MFRC522.h>

#define SS_PIN  5  // ESP32 pin GPIO5
#define RST_PIN 27 // ESP32 pin GPIO27

MFRC522 rfid(SS_PIN, RST_PIN);

void setup() {
  Serial.begin(9600);
  SPI.begin(); // init SPI bus
  rfid.PCD_Init(); // init MFRC522

  Serial.println("Tap an RFID/NFC tag on the RFID-RC522 reader");
}

void loop() {
  if (rfid.PICC_IsNewCardPresent()) { // new tag is available
    if (rfid.PICC_ReadCardSerial()) { // NUID has been readed
      MFRC522::PICC_Type piccType = rfid.PICC_GetType(rfid.uid.sak);
      Serial.print("RFID/NFC Tag Type: ");
      Serial.println(rfid.PICC_GetTypeName(piccType));

      // print UID in Serial Monitor in the hex format
      Serial.print("UID:");
      for (int i = 0; i < rfid.uid.size; i++) {
        Serial.print(rfid.uid.uidByte[i] < 0x10 ? " 0" : " ");
        Serial.print(rfid.uid.uidByte[i], HEX);
      }
      Serial.println();

      rfid.PICC_HaltA(); // halt PICC
      rfid.PCD_StopCrypto1(); // stop encryption on PCD
    }
  }
}
```

On ecrit ensuite un programme pour écrire des données sur le badge.

```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN    5  // Slave Select Pin
#define RST_PIN   0  // Reset Pin

MFRC522 mfrc522(SS_PIN, RST_PIN); // Crée une instance de MFRC522

void setup() {
  Serial.begin(115200); // Initialise la communication série
  while (!Serial); // Attendre que le port série soit prêt (utile pour les cartes basées sur USB)
  SPI.begin(18, 19, 23, 5); // Initialise SPI (SCK, MISO, MOSI, SS)
  mfrc522.PCD_Init(); // Initialise le module RC522
  Serial.println("Programme de lecture/écriture RFID initialisé");
  Serial.println("Placez votre badge RFID sur le lecteur...");
}

void loop() {
  // Vérifie si un nouveau badge est présent
  if (!mfrc522.PICC_IsNewCardPresent() || !mfrc522.PICC_ReadCardSerial()) {
    delay(50);
    return;
  }

  // Affiche l'UID du badge
  Serial.print("UID du badge : ");
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? "0" : "");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
  Serial.println();

  // Demande les informations à écrire
  Serial.println("Entrez le nom (max 16 caractères) :");
  String nom = readSerialInput();
  Serial.println("Entrez le prénom (max 16 caractères) :");
  String prenom = readSerialInput();
  Serial.println("Entrez la profession (max 16 caractères) :");
  String profession = readSerialInput();

  // Écrit les informations sur le badge
  if (writeToCard(nom, prenom, profession)) {
    Serial.println("Écriture réussie. Données enregistrées sur le badge.");
  } else {
    Serial.println("Échec de l'écriture sur le badge.");
  }

  // Arrête la communication avec le badge
  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();
```

```
  Serial.println("Placez un nouveau badge ou réinsérez le même pour une nouvelle opération.");
  Serial.println();
}

String readSerialInput() {
  String input = "";
  while (Serial.available() == 0) {
    // Attendre l'entrée de l'utilisateur
  }
  while (Serial.available() > 0 && input.length() < 16) {
    char c = Serial.read();
    if (c >= 32 && c <= 126) { // Filtrer les caractères imprimables ASCII
      input += c;
    }
    delay(10); // Attendre un peu pour permettre à l'utilisateur de taper
  }
  // Vider le buffer série si l'entrée dépasse 16 caractères
  while (Serial.available() > 0) {
    Serial.read();
  }
  Serial.println("Vous avez entré : " + input);
  return input;
}

bool writeToCard(String nom, String prenom, String profession) {
  MFRC522::MIFARE_Key key;
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;

  byte buffer[16] = {0};
  byte block;
  MFRC522::StatusCode status;
  byte len;

  // Authentifie le secteur 1 (blocs 4 à 7)
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, 7, &key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Erreur d'authentification : ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return false;
  }
```

```
  // Écriture du nom dans le bloc 4
  block = 4;
  len = nom.length();
  memset(buffer, 0, sizeof(buffer));
  memcpy(buffer, nom.c_str(), len > sizeof(buffer) ? sizeof(buffer) : len);
  status = mfrc522.MIFARE_Write(block, buffer, 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Erreur d'écriture du nom : ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return false;
  }

  // Écriture du prénom dans le bloc 5
  block = 5;
  len = prenom.length();
  memset(buffer, 0, sizeof(buffer));
  memcpy(buffer, prenom.c_str(), len > sizeof(buffer) ? sizeof(buffer) : len);
  status = mfrc522.MIFARE_Write(block, buffer, 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Erreur d'écriture du prénom : ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return false;
  }

  // Écriture de la profession dans le bloc 6
  block = 6;
  len = profession.length();
  memset(buffer, 0, sizeof(buffer));
  memcpy(buffer, profession.c_str(), len > sizeof(buffer) ? sizeof(buffer) : len);
  status = mfrc522.MIFARE_Write(block, buffer, 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Erreur d'écriture de la profession : ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return false;
  }

  return true;
}
```

Il nous faut ensuite écrire un programme pour lire les données qu'on vient d'écrire.

```
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN    5  // Slave Select Pin
#define RST_PIN   0  // Reset Pin

MFRC522 mfrc522(SS_PIN, RST_PIN); // Crée une instance de MFRC522

void setup() {
  Serial.begin(115200); // Initialise la communication série
  while (!Serial);      // Attendre que le port série soit prêt
  SPI.begin(18, 19, 23, 5); // Initialise SPI (SCK, MISO, MOSI, SS)
  mfrc522.PCD_Init(); // Initialise le module RC522
  Serial.println("Programme de lecture RFID initialisé");
  Serial.println("Placez votre badge RFID sur le lecteur...");
}

void loop() {
  // Vérifie si un nouveau badge est présent et peut être lu
  if (!mfrc522.PICC_IsNewCardPresent() || !mfrc522.PICC_ReadCardSerial()) {
    delay(50);
    return;
  }

  // Affiche l'UID du badge
  Serial.print("UID du badge : ");
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? "0" : "");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
  Serial.println();

  // Lit les informations du badge
  readFromCard();

  // Arrête la communication avec le badge
  mfrc522.PICC_HaltA();
  mfrc522.PCD_StopCrypto1();

  Serial.println("Placez un nouveau badge pour une nouvelle lecture.");
  Serial.println();
}

void readFromCard() {
  MFRC522::MIFARE_Key key;
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
```

```
  byte buffer[18];
  byte size = sizeof(buffer);
  MFRC522::StatusCode status;

  // Authentifie le secteur 1 (blocs 4 à 7) avec la clé A
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, 7, &key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Erreur d'authentification : ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  }

  String nom = readBlock(4, "Nom");
  String prenom = readBlock(5, "Prénom");
  String profession = readBlock(6, "Profession");

  // Affiche les informations lues
  Serial.println("Informations du badge :");
  Serial.println("Nom : " + nom);
  Serial.println("Prénom : " + prenom);
  Serial.println("Profession : " + profession);
}

String readBlock(byte blockAddr, const char* fieldName) {
  byte buffer[18];
  byte size = sizeof(buffer);
  MFRC522::StatusCode status;

  status = mfrc522.MIFARE_Read(blockAddr, buffer, &size);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Erreur de lecture du bloc ");
    Serial.print(blockAddr);
    Serial.print(" (");
    Serial.print(fieldName);
    Serial.print(") : ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return "";
  }

  String result = "";
  for (byte i = 0; i < 16 && buffer[i] != 0; i++) {
    if (isPrintable(buffer[i])) {
      result += (char)buffer[i];
    }
  }
  return result.c_str(); // Convertit en c-string pour éliminer les espaces de fin
}

bool isPrintable(char c) {
  return (c >= 32 && c <= 126); // Caractères imprimables ASCII
}
```

Grace a ces deux programmes, chaque membre du binôme s'équipe d'un badge et nous ecrivons tous les deux nos noms sur les badges.

```
UID du badge : F31F9CAB
Informations lues depuis le badge :
Nom : TAMBA
Prénom : MIKE WILFRIED
Profession : ETUDIANT
Âge : 22
Compteur : 1
UID du badge : 03122A02
Informations lues depuis le badge :
Nom : CARRE - CAZE
Prénom : BASTIEN
Profession : ETUDIANT
Âge : 20
Compteur : 1
```

Défilement automatique Afficher l'horodatage

On retrouve des chaînes de caractères et des entiers dans les informations contenus dans les badges.

On code ensuite un site internet pour lire les données en ligne.

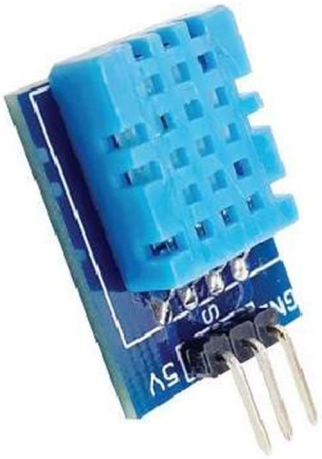
Mike Wilfried TAMBA BUT3 GEII
Bastien CARRÉ – CAZÉ

2ème Partie : Node Red

Nous avons ensuite attaqué la seconde partie du node Red.

Node Red est un outil de développement basé sur les flux pour la programmation visuelle pour connecter ensemble des périphériques matériels, des API et des services en ligne dans le cadre de l'Internet des objets (IoT).

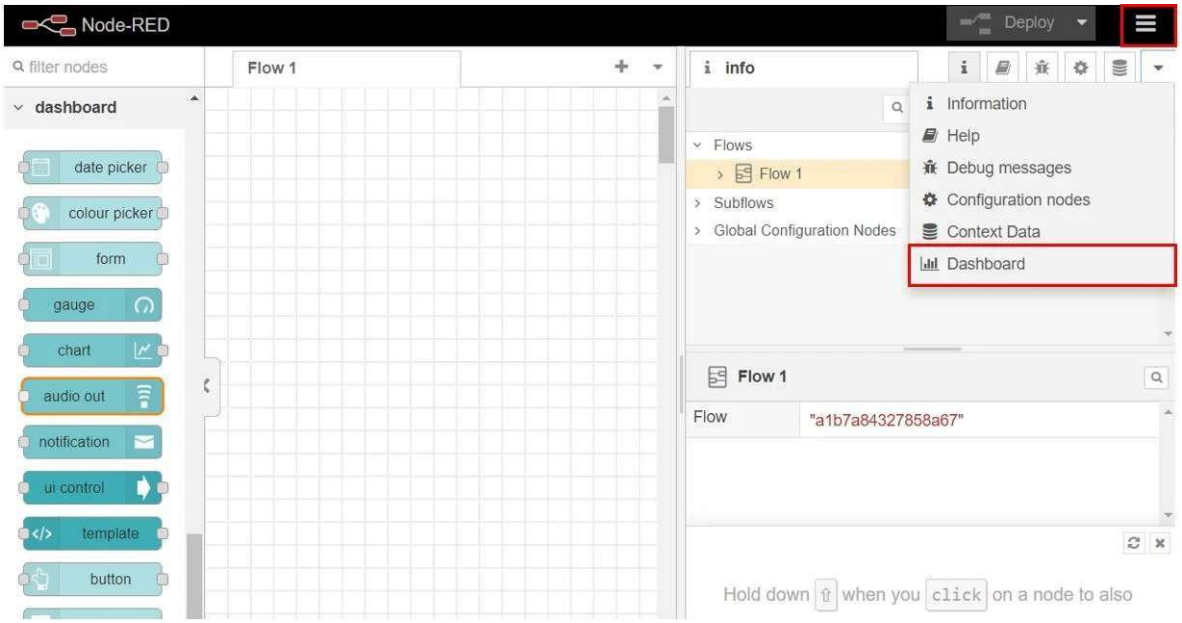
Pour cette partie, nous devions réaliser une interface et relier notre capteur sur cette dernière afin de transférer les données reçues de ce capteur directement sur un serveur MQTT dans le but de les visualiser en temps réel.



Le capteur en question est un DHT11 Sensor,
il est capable de mesurer la température ainsi que l'humidité

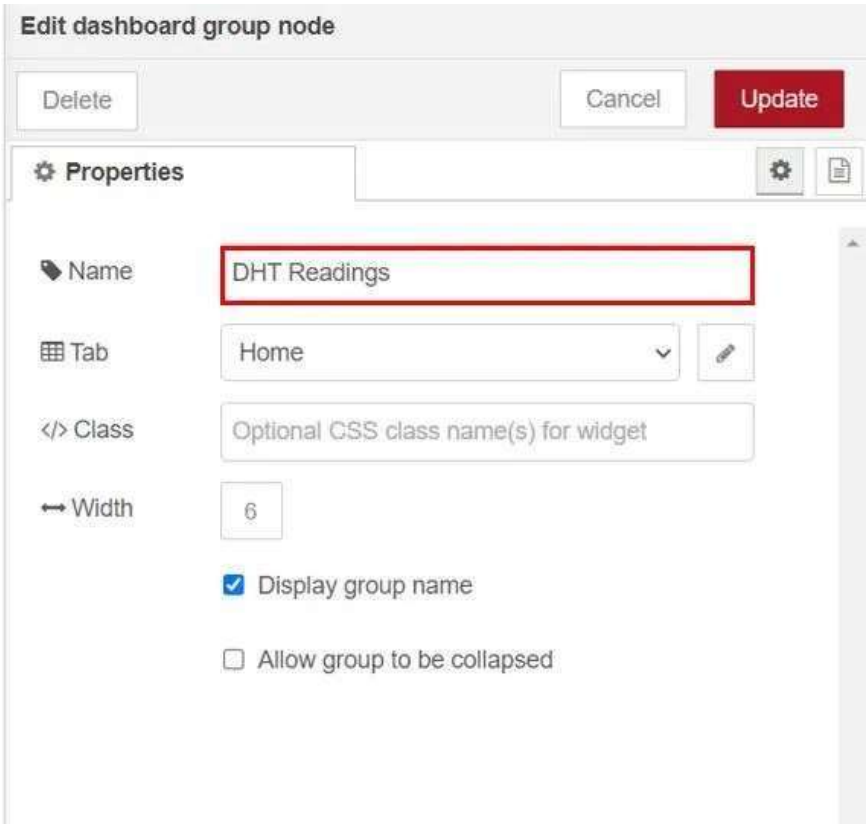
Nous avons commencé par ajouter des blocs sur l'interface principal, nous avons renommé un "Flow" à notre nom.

Nous avons ajouté un "DashBoard" afin de créer un publisher, qui publieras les données du capteur et un subscriber qui lira les données sur le publisher et les afficherai sur un interface dédié de Node Red



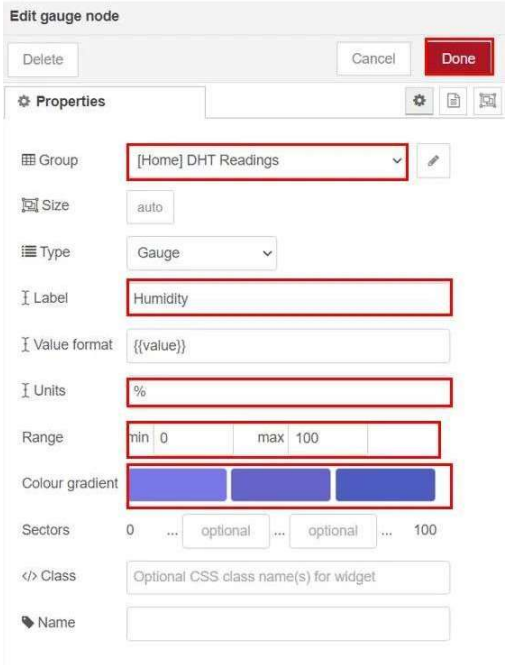
Pour ce faire il nous as fallu créer un groupe dans le DashBoard qui contiendras plus tard les publisher et le subscriber. Nous l'avons nommé "DHT Reading"

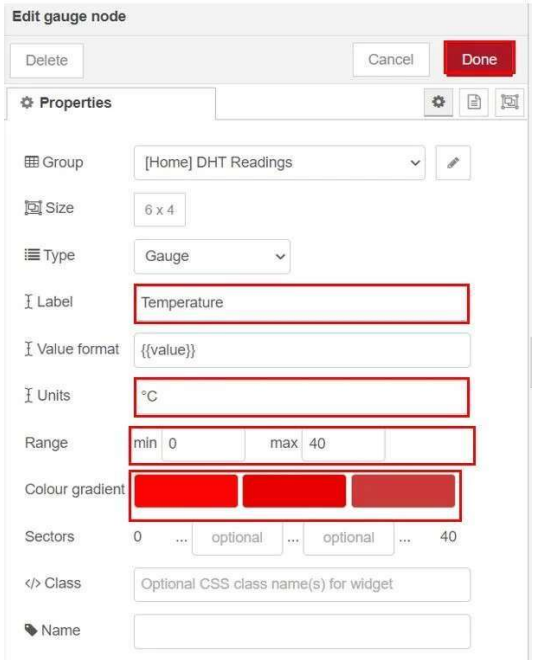
C'est dans cet interface que nous avons paramétrer nos publishers et subscriber

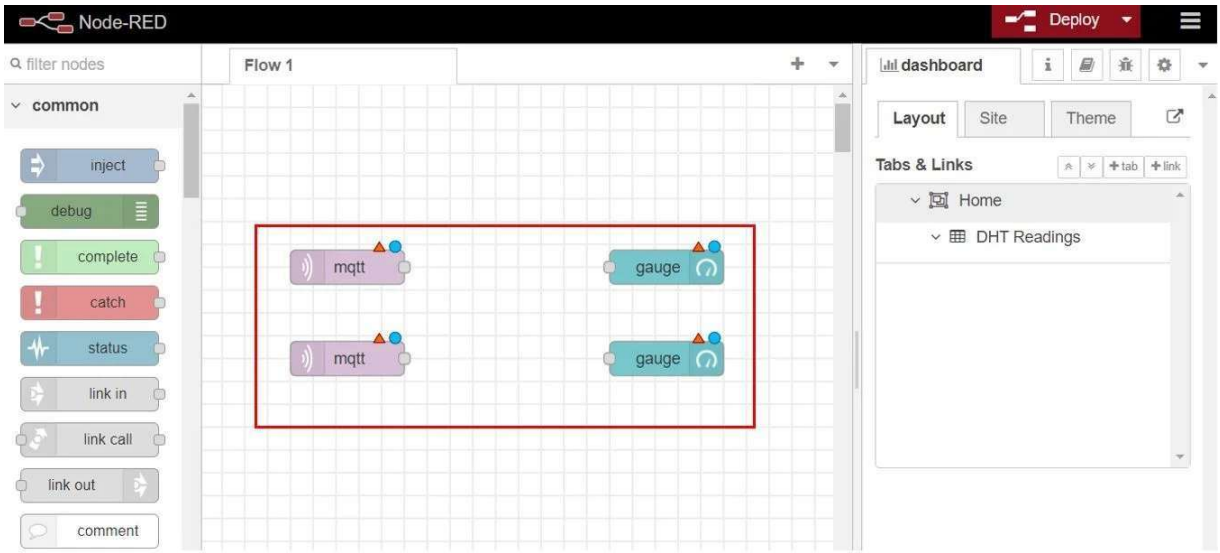


Il nous a fallu également rajouter des blocs "mqtt" et "gauge"

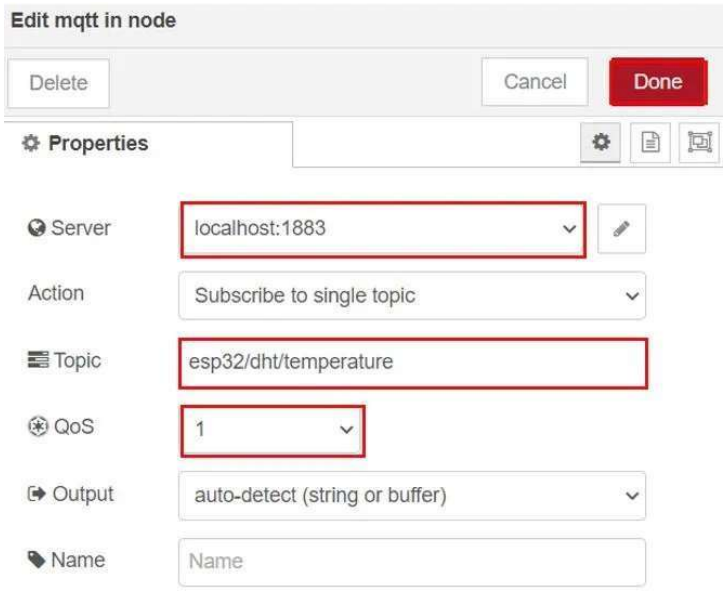
Le bloc "mqtt" contenait les informations du capteur et les transmettait à une gauge que nous avons paramétrés (en modifiant notamment la couleur, l'unité, le nom ainsi que le min/max)



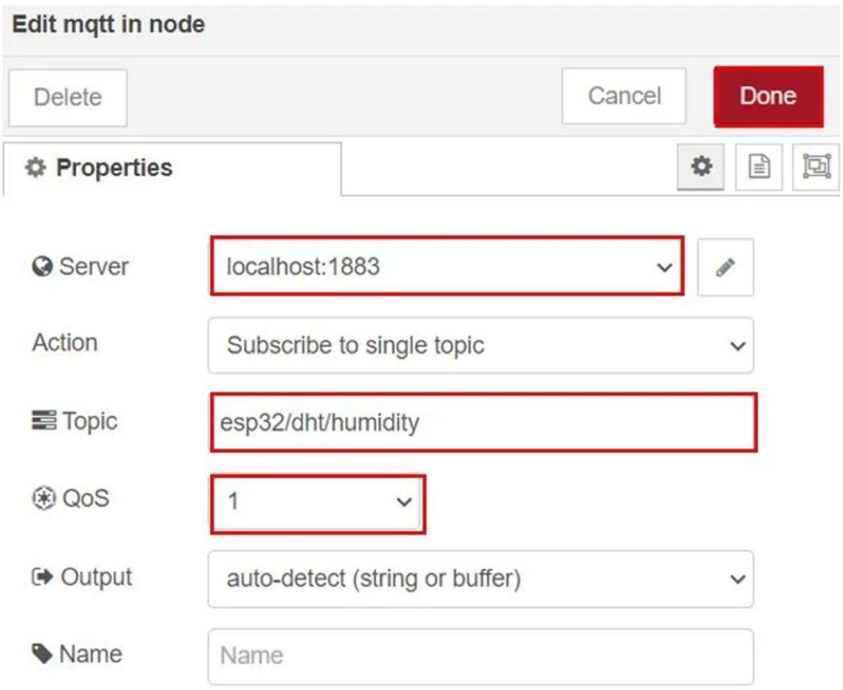




Pour paramétrer correctement le Broker MQTT nous avons dû nous synchroniser sur le bon port, ici 1883 car on utilise le Broker Mosquitto. Les relevés de température sont publiés dans le sujet “esp32/dht/temperature”.



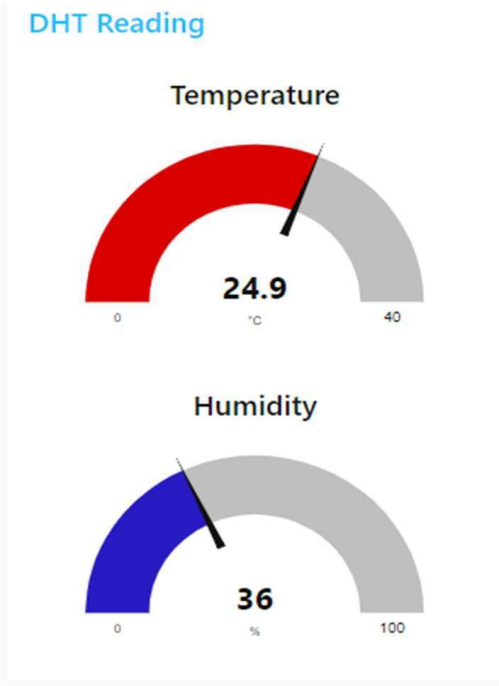
Puis nous avons fait la même chose pour l'humidité avec un sujet différent qui est "esp/dht/humidity".



Nous avons pu confirmer la validité de notre code en récupérant ses valeurs de la salle de classe en temps réel.

Mike Wilfried TAMBA
Bastien CARRÉ – CAZÉ

Nous ajoutons ensuite une interface

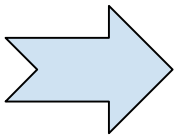


A l'aide du logiciel en ligne Wokwim on simule un esp 32, ce qui nous permet de ne pas avoir de contrainte physiques et de travailler depuis chez nous.

Mike Wilfried TAMBA BUT3 GEII
Bastien CARRÉ – CAZÉ

Le programme suivant est conçu pour lire les données de température et d'humidité à partir d'un capteur DHT11, puis les publier via le protocole MQTT à un broker MQTT sur un réseau Wi-Fi.

```
#include "DHT.h"
#include <WiFi.h>
extern "C" {
#include "freertos/FreeRTOS.h"
#include "freertos/timers.h"
}
#include <AsyncMqttClient.h>
#include <Wire.h>
#include <OneWire.h>
#include <DallasTemperature.h>
```



Ici, nous avons ajouté les différentes bibliothèques pour le bon fonctionnement du programme

```
//replace with your network credentials
#define WIFI_SSID "BlackJStar"
#define WIFI_PASSWORD "Jenny509"
```

La partie connection WiFi se fait grâce au point d'accès du téléphone de Jonathan

```
// Raspberry Pi Mosquitto MQTT Broker
#define MQTT_HOST IPAddress(192, 168, 10, 8)
#define MQTT_PORT 1883
```

Nous avons ajouté l'adresse IP ainsi que le port du

```
//MQTT Topics
#define MQTT_PUB_TEMP_DHT "esp32/dht/temperature"
#define MQTT_PUB_HUM_DHT  "esp32/dht/humidity"
```

Ce sont les topics MQTT sous lesquels les données de température et d'humidité ser publiées.

```
#define DHTPIN 15
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

const int SensorDataPin = 4;

OneWire oneWire(SensorDataPin);
DallasTemperature sensors(&oneWire);
```

Cette section configure le capteur DHT11 (pour la température et l'humidité) ainsi que le capteur OneWire pour la température.

```
float temperature_DHT, humidity_DHT; //variables for DHT

AsyncMqttClient mqttClient;
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;

unsigned long previousMillis = 0;
```

Ces variables sont utilisées pour contrôler l'intervalle de lecture et de publication des données du capteur.

```
const long interval = 10000;

void connectToWifi() {
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}
```

Connection au WiFi en point d'accès

```
void connectToMqtt() {
  Serial.println("Connecting to MQTT...");
  mqttClient.connect();
}
```

Connection au serveur MQTT grâce aux précédents paramètres (Adresse IP, port …)

```
void WiFiEvent(WiFiEvent_t event) {
  Serial.printf("[WiFi-event] event: %dn", event);
  switch (event) {
    case SYSTEM_EVENT_STA_GOT_IP:
      Serial.println("WiFi connected");
      Serial.println("IP address: ");
      Serial.println(WiFi.localIP());
      connectToMqtt();
      break;
    case SYSTEM_EVENT_STA_DISCONNECTED:
      Serial.println("WiFi lost connection");
      xTimerStop(mqttReconnectTimer, 0);
      xTimerStart(wifiReconnectTimer, 0);
      break;
  }
}
```

Cette partie là gère tout ce qui est lié à la connection, donc l'affichage de la connexion établi, l'adresse IP du point d'accès ainsi que la déconnection (ou signal perdu)

```
void onMqttConnect(bool sessionPresent) {
  Serial.println("Connected to MQTT.");
  Serial.print("Session present: ");
  Serial.println(sessionPresent);
}
```

Cette partie là gère tout ce qui est lié à la connection au serveur MQTT, donc l'affichage de la connexion établi ainsi que la déconnection.

```
void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
  Serial.println("Disconnected from MQTT.");
  if (WiFi.isConnected()) {
    xTimerStart(mqttReconnectTimer, 0);
  }
}

void onMqttPublish(uint16_t packetId) {
```

```
  Serial.print("Publish acknowledged.");
  Serial.print("  packetId: ");
  Serial.println(packetId);
}

void setup() {
  Serial.begin(115200);
  Serial.println();

  dht.begin();
  delay(1000);

  sensors.begin();
  delay(1000);

  mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFALSE, (void*)0,
reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt));
  wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFALSE, (void*)0,
reinterpret_cast<TimerCallbackFunction_t>(connectToWifi));

  WiFi.onEvent(WiFiEvent);

  mqttClient.onConnect(onMqttConnect);
  mqttClient.onDisconnect(onMqttDisconnect);
  mqttClient.onPublish(onMqttPublish);
  mqttClient.setServer(MQTT_HOST, MQTT_PORT);
  connectToWifi();
}

void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    //Read from DHT
    humidity_DHT = dht.readHumidity();
    temperature_DHT = dht.readTemperature();

    if (isnan(temperature_DHT) || isnan(humidity_DHT)) {
      Serial.println(F("Failed to read from DHT sensor!"));
      return;
    }
    //delay(3000);
```

Ici, on initialise la communication série, les capteurs, les minuteries et la configuration des connexions Wi-Fi et MQTT

Dans la boucle, elle lit la température et l'humidité à partir du capteur DHT11, publie les données sur les sujets MQTT définis, puis attend l'intervalle spécifié avant de recommencer.

Publication de la température
reçu du capteur par le publisher dans le topic

```
    // Publish an MQTT message on topic esp32/dht/temperature
    uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP_DHT, 1, true,
String(temperature_DHT).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", MQTT_PUB_TEMP_DHT,
packetIdPub1);
    Serial.printf("Message: %.2f \n", temperature_DHT);

    // Publish an MQTT message on topic esp32/dht/humidity
    uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM_DHT, 1, true,
String(humidity_DHT).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ", MQTT_PUB_HUM_DHT,
packetIdPub2);
    Serial.printf("Message: %.2f \n", humidity_DHT);
  }
}
```

Publication de l'humidité reçu du capteur par le publisher dans le topic

Conclusion

Ce projet nous as permis d'approfondir nos connaissances en Arduino ainsi que dans l'utilisation d'un ESP32, on a pu voir une nouvelle utilisation qui permettait de récupérer des données à distance. On a aussi appris a utiliser node red et wokwi qui nous ont permit respectivement de ne pas écrire de code et de se concentrer sur la logique programmatoire et de simuler un ESP depuis chez nous. La combinaison des 2 offre une grande accessibilité et facilite l'apprentissage.