HELHa Charleroi - Informatique Industrielle - 3BINI - 2022/2023

Internet des objets (IoT) 01 – TP01 – TOFFOLO Nicolas

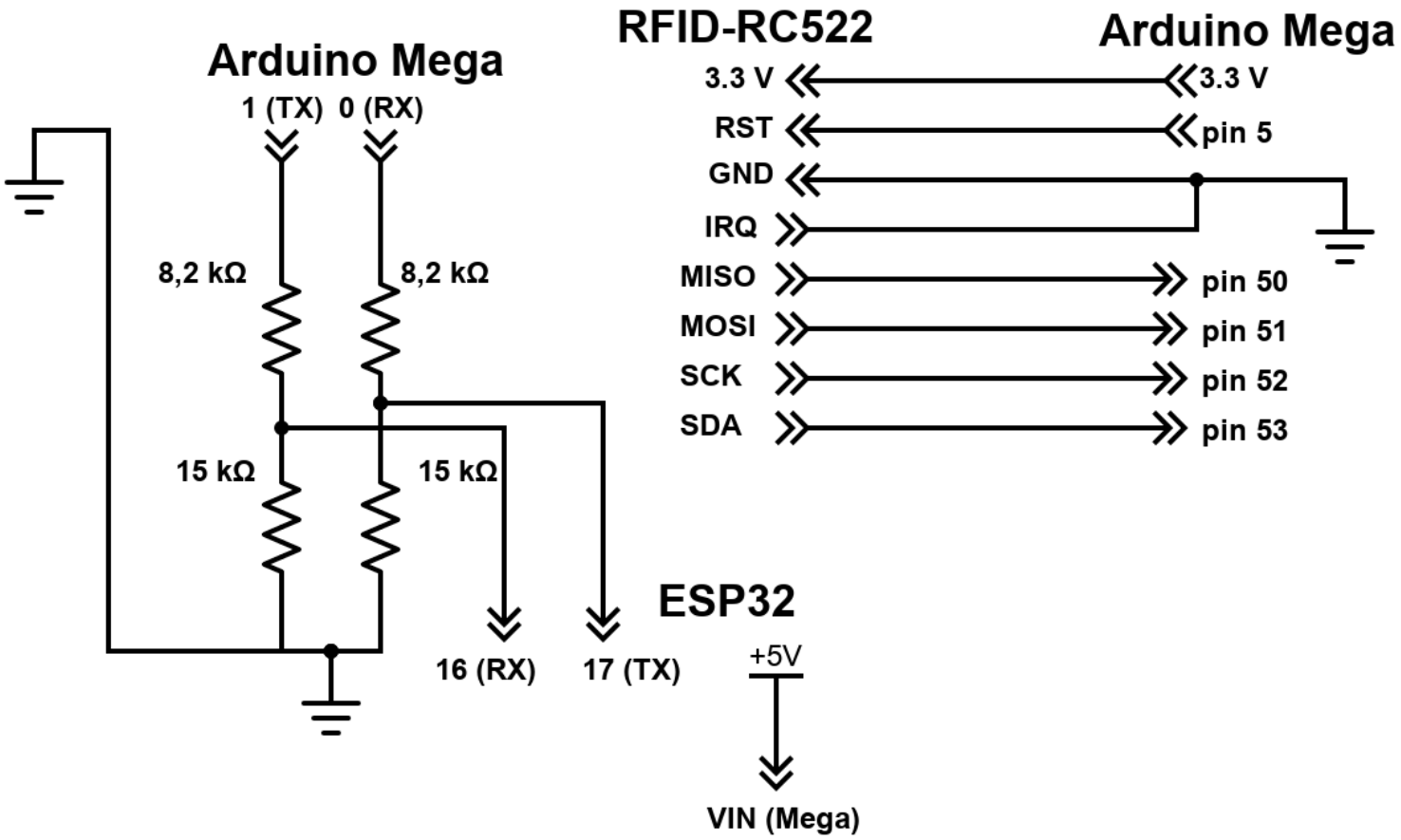
1. CAHIER DES CHARGES

Réalisation d'un montage et programmation du processus suivant :

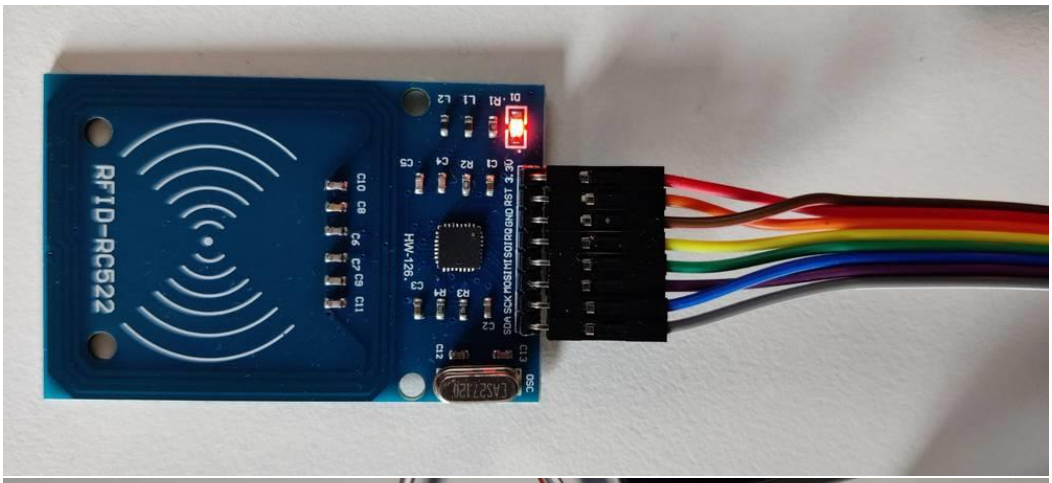
1. Connecter un module RFID (RC522) à l'Arduino
2. Associer des badges aux prénoms des membres du groupe
3. Lire un badge et afficher sur un écran LCD le prénom correspondant
4. Si le badge ne correspond à aucun ID connu => Afficher « INCONNU »

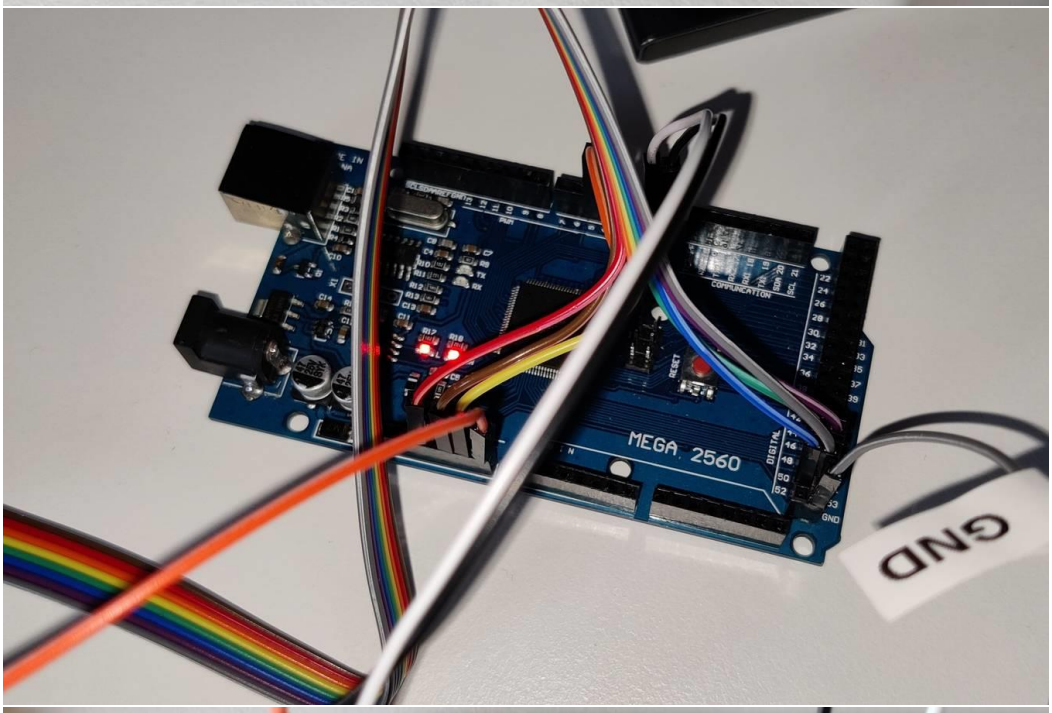
OPTION : Plutôt que d'afficher de résultat de la lecture sur un écran LCD, l'Arduino envoie le message à un ESP32, qui le transmet à un broker MQTT.

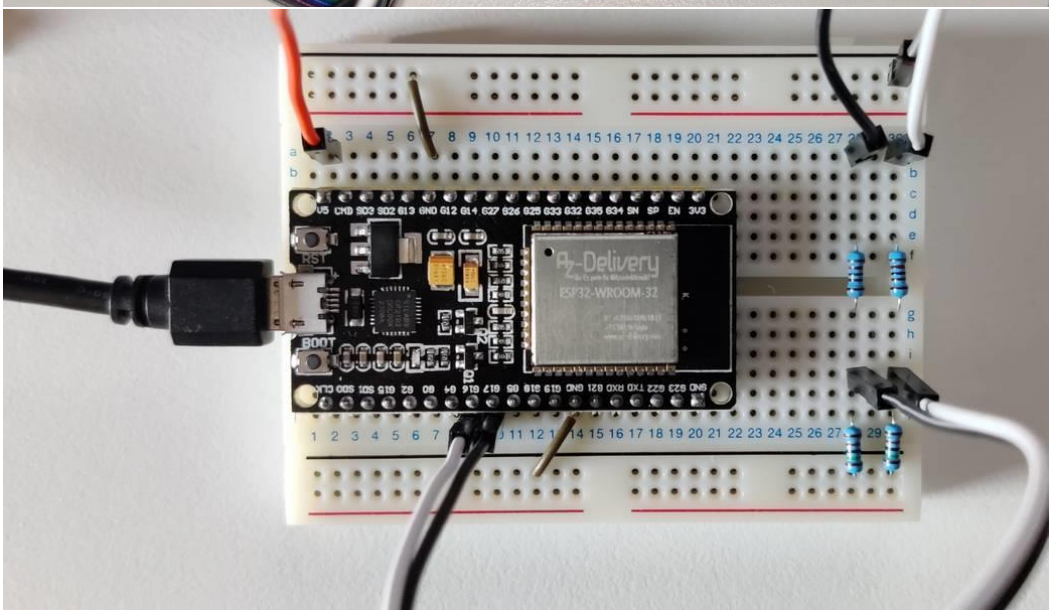
2. SCHÉMA ÉLECTRONIQUE



3. MONTAGE







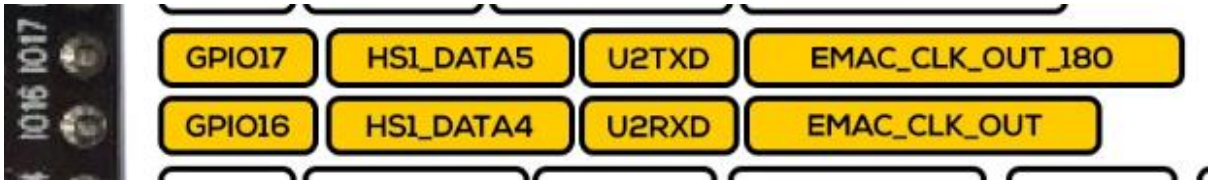
4. MATÉRIEL

4.1. UART sur ESP32

Les ESP32 possèdent trois paires de pins RX/TX utilisables pour une communication UART :

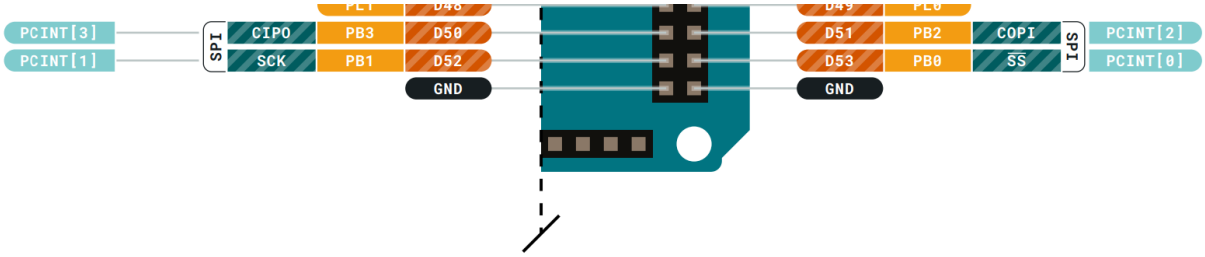
- Les pins 3 et 1 (RX0 et TX0)
- Les pins 9 et 10 (RX1 et TX1)
- Les pins 16 et 17 (RX2 et TX2)

J'ai choisi arbitrairement les pins 16 et 17 pour ce TP.



4.2. SPI sur Arduino Mega

L'Arduino Mega communique via le protocole SPI avec le module RFID-RC522. Les pins 50 à 53 permettent cette communication.



5. BIBLIOTHÈQUES UTILISÉES

5.1. Mega

`<MFRC522.h>` et `<SPI.h>` pour travailler et communiquer facilement avec le module RFID.

5.2. ESP32

`<WiFi.h>` pour avoir accès à internet avec l'ESP32.

`<PubSubClient.h>` pour communiquer facilement avec un broker MQTT.

6. DÉCLARATION DES VARIABLES

6.1. Mega

J'utilise le type de variable **Uid** présent dans la bibliothèque **MFRC522** pour récupérer proprement les lectures du module RFID. Ce type de variable est une structure :

```
typedef struct
{
    byte size; // Number of bytes in the UID. 4, 7 or 10.
    byte uidByte[10];
    byte sak; // The SAK (Select acknowledge) byte returned from the PICC after successful selection.
} Uid;
```

Par extension, un **dbItem** est une structure contenant un **Uid** lié à un nom :

```
typedef struct
{
    const char *name[10];
    MFRC522::Uid rfid;
} dbItem;
```

Global

```
const byte BYTE_NBR = 4, SAK_TYPE = 8, RST_PIN = 5, SS_PIN = 53

MFRC522 mfrc522

typedef struct {const char *name[10]; MFRC522::Uid rfid} dbItem
const dbItem DB[]
```

Local (function)

```
const unsigned long NBR_OF_ENTRIES
bool found
```

6.2. ESP32

Global

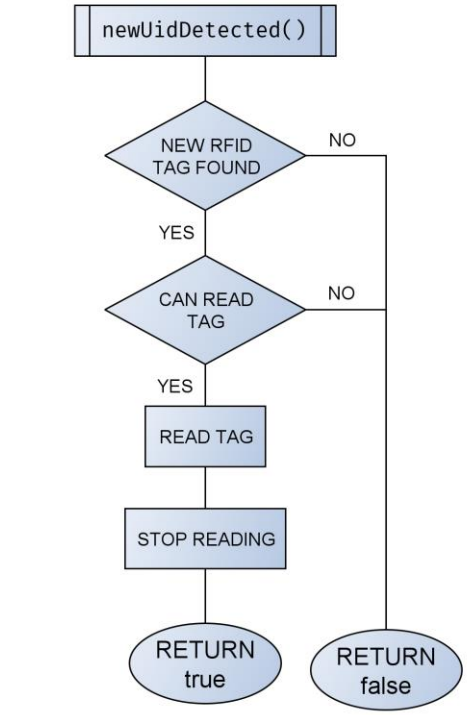
```
const byte RX = 16, TX = 17

const char *ssid, *password
WiFiClient wifiClient

PubSubClient mqttClient
```

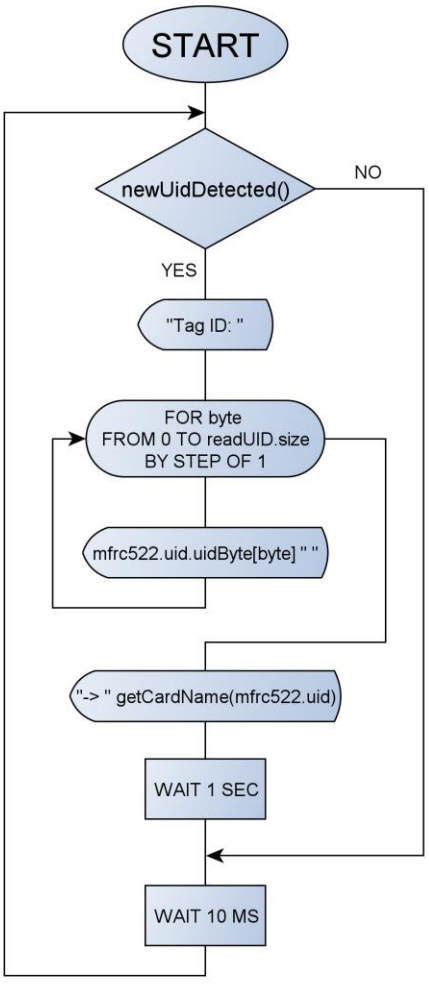
Local (functions)

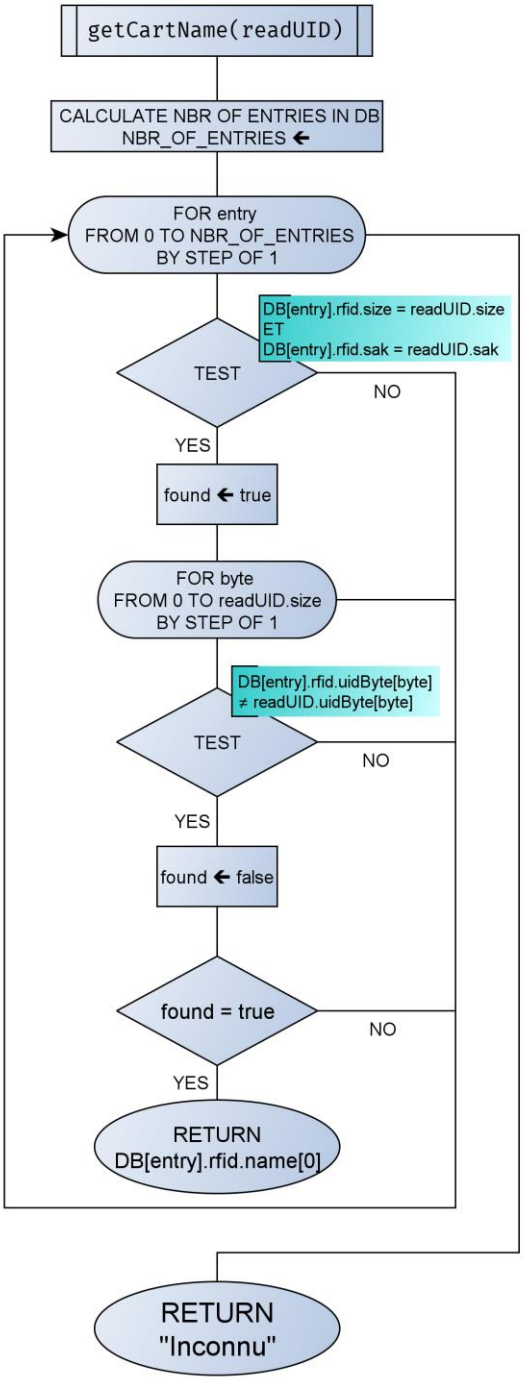
```
bool connected
bool sent

String message
```

7. ORDINOGRAMME

7.1. Mega








```
dbItem {
    const char* name[10]
    Uid         rfid
}

Uid {
    byte  size
    byte  uidByte[10]
    byte  sak
}
```

8. CODE

8.1. Mega

Fichier RFID_DB_TP2.h

```
#include <MFRC522.h>

#define BYTE_NBR 4
#define SAK_TYPE 8

typedef struct
{
    const char *name[10];
    MFRC522::Uid rfid;
} dbItem;

const dbItem DB[] = {
    {
        {"Nico"}, {BYTE_NBR, {0xE, 0xC5, 0x51, 0x3}, SAK_TYPE}
    },
    {
        {"Sajad"}, {BYTE_NBR, {0x4, 0x98, 0x51, 0x3}, SAK_TYPE}
    },
    {
        {"Ronaldo"}, {BYTE_NBR, {0x1A, 0x5A, 0xCD, 0x18}, SAK_TYPE}
    },
    {
        {"Timothee"}, {BYTE_NBR, {0x1A, 0xD6, 0xC2, 0x18}, SAK_TYPE}
    }
};

const char *getCardName(MFRC522::Uid readUID)
{
    const unsigned long NBR_OF_ENTRIES = sizeof(DB) / sizeof(dbItem);
    for (unsigned int entry = 0; entry < NBR_OF_ENTRIES; entry++)
    {
        if (DB[entry].rfid.size == readUID.size && DB[entry].rfid.sak ==
readUID.sak)
        {
            bool found = true;
            for (int byte = 0; byte < readUID.size; byte++)
            {
                if (DB[entry].rfid.uidByte[byte] !=
readUID.uidByte[byte])
                {
                    found = false;
                    break;
                }
            }
            if (found)
            {
                return DB[entry].name[0];
            }
        }
    }
    return "Inconnu";
}
```

Fichier `main.cpp`

```cpp
#include <Arduino.h>
#include <SPI.h>
#include <MFRC522.h>
#include "RFID_DB_TP2.h"

#define RST_PIN 5
#define SS_PIN 53

MFRC522 mfrc522(SS_PIN, RST_PIN);

boolean newUidDetected()
{
    // Checke si une nouvelle carte est présente
    if (!mfrc522.PICC_IsNewCardPresent()) return false;
    // Stocke les données du tag dans `mfrc522.uid` et renvoie `true` si tout est OK
    if (!mfrc522.PICC_ReadCardSerial()) return false;

    mfrc522.PICC_HaltA(); // Stop reading
    return true;
}

void setup()
{
    Serial.begin(115200);
    SPI.begin();
    mfrc522.PCD_Init();
    Serial.println("RFID ready on Mega");
}

void loop()
{
    if (newUidDetected())
    {
        Serial.print("Tag ID: ");
        for (int byte = 0; byte < mfrc522.uid.size; byte++)
        {
            Serial.print(mfrc522.uid.uidByte[byte], HEX);
            Serial.print(" ");
        }
        Serial.print("-> ");
        Serial.println(getCardName(mfrc522.uid));
        delay(1000);
    }
    delay(10);
}
```

8.2. ESP32

```
#include <Arduino.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include "ssid_password.h"
/* Contient :
   const char *ssid = "SSID_NAME";
   const char *password = "PASSWORD";
 */

#define RX 16
#define TX 17

WiFiClient wifiClient;
PubSubClient mqttClient("test.mosquitto.org", 1883, wifiClient);

IPAddress initWiFi()
{
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED)
    {
        Serial.print('.');
        delay(100);
    }

    return WiFi.localIP();
}

bool mqttConnect()
{
    bool connected = mqttClient.connected();
    if (!connected)
    {
        Serial.print("Connecting to MQTT...");
        mqttClient.connect("esp32_NicoToff");
        connected = mqttClient.connected();
        connected ?
           Serial.println(" Connected!") : Serial.println(" FAILED!!!");
    }
    return connected;
}

bool sendToMqtt(String topic, String message)
{
    bool sent = false;
    if (mqttConnect())
    {
        sent = mqttClient.publish(topic.c_str(), message.c_str());
    }
    if (!sent)
    {
        Serial.println("Couldn't send to MQTT!");
    }
    return sent;
}
```

```
void setup()
{
    Serial.begin(115200);
    Serial2.begin(115200, SERIAL_8N1, RX, TX);
    Serial.println("Hello by ESP32");

    Serial.print("Connecting to WiFi..");
    IPAddress ip = initWiFi();
    Serial.print("IP address: ");
    Serial.println(ip);
    Serial.print("On SSID: ");
    Serial.println(WiFi.SSID());

    mqttConnect();
}

void loop()
{
    if (Serial2.available())
    {
        String message = Serial2.readString();
        Serial.println(message);
        sendToMqtt("helha/iot/tp2", message);
        delay(2000);
    }
    delay(10);
}
```

8.3. Exemple de code (NodeJS) pour se connecter à MQTT

```
const mqtt = require("mqtt")
                .connect(`mqtt://test.mosquitto.org:1883`);

const TOPIC = "helha/iot/tp2";

mqtt.on("connect", () => {
    console.log("Connected to MQTT broker");
    mqtt.subscribe(TOPIC);
});

mqtt.on("message", (topic, message) => {
    console.log("<" + new Date() + "> " + message.toString());
});

// Reconnexion automatique toutes les 10 secondes si déconnecté
setInterval(() => {
    while (!mqtt.connected) {
        mqtt.reconnect();
    }
}, 10000);
```