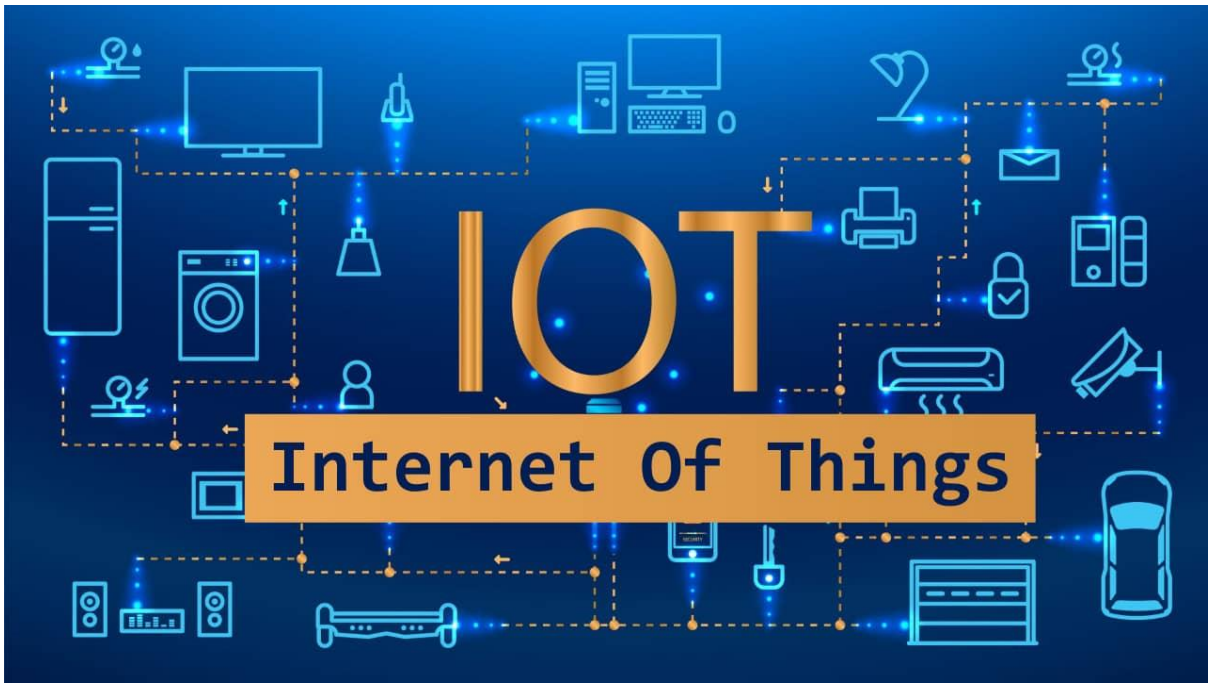
PROJET IOT



HECHMI Hazem
JAFAIS Noumane
KONE Issiaka

BUT 3 GEII ESE

S5.01A - SAE IOT

M. LAVAL

2023/2024

1ère partie : Étiquettes électronique (RFID)

La première partie consiste à étudier les processus de lecture / écriture d'étiquettes électronique (RFID) à partir d'un module RC522 couplé à un ESP32.

2ème partie : Plateforme IOT (NodeRED)

La seconde partie consiste de récupérer la valeur de de température du capteur DHT11 couplé à un ESP32 et de l'afficher via le logiciel Node Red.

1ère partie : Étiquettes électronique (RFID)

Le système RFID (Radio Frequency Identification) est une technologie qui utilise des étiquettes équipées de puces électroniques pour stocker et transmettre des données. Ces étiquettes sont attachées à des objets ou intégrées dans des cartes. Un lecteur RFID émet des signaux qui activent les tags, permettant la lecture des informations stockées, telles que des identifiants uniques. Les puces RFID servent à identifier et de localiser des objets ou des personnes. On les retrouve dans les cartes bancaires ou encore dans les badges.

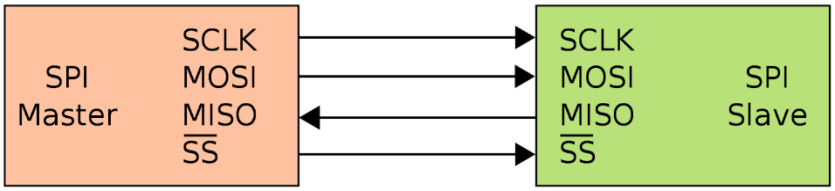
L'objectif de notre projet est d'analyser les processus de lecture et d'écriture des étiquettes électroniques (RFID) en utilisant un module RC522 connecté à un ESP32. Il faudra donc écrire sur l'étiquette RFID à l'aide du module RC522 des informations telles que le nom, le prénom, l'identifiant (UID) et un montant (valeur réelle). Ensuite, il faudra visualiser les données de l'étiquette en utilisant un serveur web sur l'ESP32.

Pour la programmation de l'ESP32, nous utiliserons le logiciel Arduino. Nous utiliserons le protocole SPI. Les modules RFID, RC522 fonctionnent sur 13,56 MHz, basés sur le contrôleur MFRC522 conçu par NXP Semiconductor. Le module RC522 prend en charge les protocoles de communication SPI et I2C. Ici, nous utilisons l'interface de communication SPI avec la carte ESP32. Ce module fonctionne sur une alimentation +3,3 V/13-26 mA.

L'interface d'un protocole SPI :

Le bus SPI utilise quatre signaux logiques :

- **SCLK** — Serial Clock, Horloge (généré par le maître)
- **MOSI** — Master Output, Slave Input (généré par le maître)
- **MISO** — Master Input, Slave Output (généré par l'esclave)
- **SS** — Slave Select, Actif à l'état bas (généré par le maître)



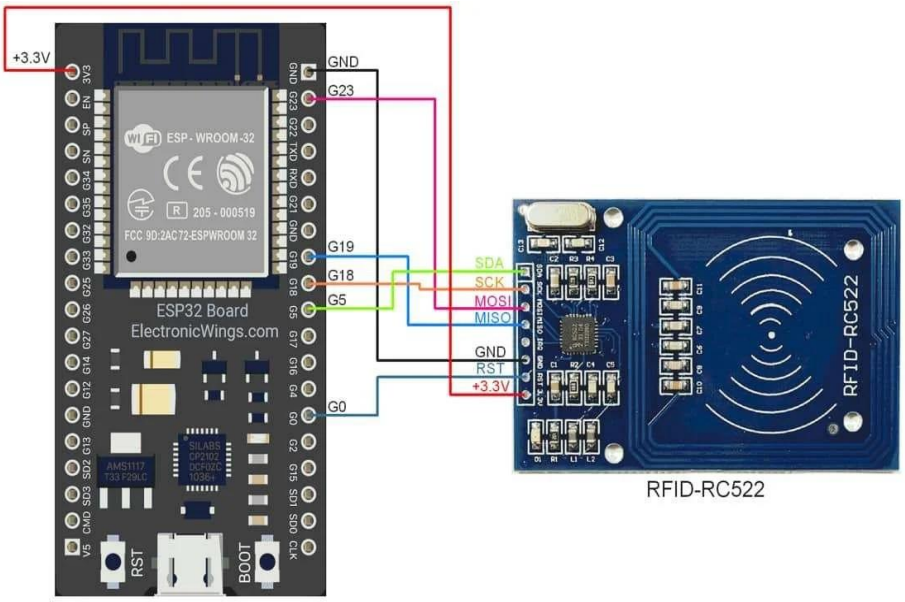
Le fonctionnement d'un protocole SPI :

Une transmission SPI typique est une communication simultanée entre un maître et un esclave :

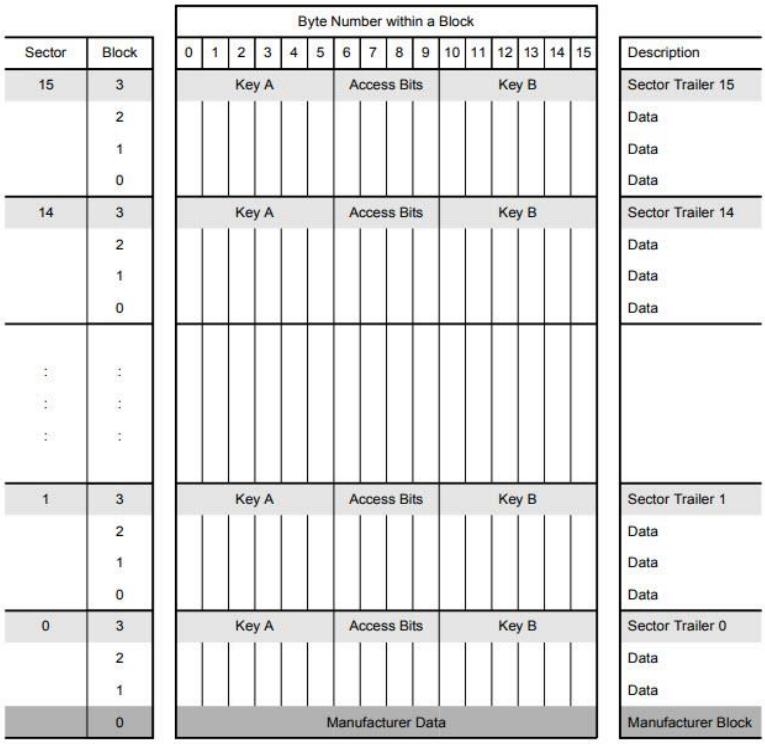
- Le maître génère l'horloge et sélectionne l'esclave avec qui il veut communiquer par l'utilisation du signal SS
- L'esclave répond aux requêtes du maître

À chaque coup d'horloge le maître et l'esclave s'échangent un bit. Après huit coups d'horloges le maître a transmis un octet à l'esclave et vice versa.

Nous commencerons par réaliser le câblage d'un ESP32 avec le module RC522. Le module RC522 est un lecteur RFID qui permet de lire et écrire des données.



Il existe plusieurs types d'étiquette RFID, nous utiliserons une carte de type MIFARE 1K. Cette carte possède une mémoire de 1024 octets organisée en 16 secteurs (secteur 0 à secteur 15). Chaque secteur se compose de 4 blocs. La taille maximale d'un bloc est de 16 octets. Les données sur la carte seront enregistrées en hexadécimal. Ils seront convertis en fonction du code ASCII.



Nous utilisons le bloc 4 pour stocker le nom, le bloc 5 pour le prénom et le bloc 6 pour le montant. Le nom et le prénom seront de type Chaîne de caractères (String) et le montant sera une valeur réelle (float). Nous utiliserons la fonction `mfrc522.MIFARE_Write()` pour l'écriture des données.

```
int blockNum1 = 4; // Pour le nom
int blockNum2 = 5; // Pour le prénom
int blockNum3 = 6; // Pour le montant

//////// … /////////

/* Affectation de la valeur de montant réelle en une chaîne de caractère */

byte *tempBytes = (byte *)&montant;

for (int I = 0; I < sizeof(montant); i++) {
 blockData3[i] = tempBytes[i];
}

//////// … /////////

/* Définition des données à écrire sur la carte */

byte blockData1[16] = { "JAFAIS HECHMI" };
byte blockData2[16] = { "KONE" };
byte blockData3[4];
float montant = 123.456;
```

Le programme d'écriture complet sur la carte RFID se trouve en ANNEXE 1.1.

Nous obtenons le résultat suivant dans le moniteur série.

```
**Card Detected**
Card UID: B3 0B A0 A5
PICC type: MIFARE 1KB

Writing to Data Blocks...
PCD_Authenticate() block 4 success
Writing block 4 success
PCD_Authenticate() block 5 success
Writing block 5 success
PCD_Authenticate() block 6 success
Writing block 6 success
Data Montant written successfully

Reading from Data Blocks...
Block 4 : JAFAIS HECHMI
Block 5 : KONE
Montant : 123.46
```

3	15	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF
	14	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
	13	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
	12	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
2	11	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF
	10	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
	9	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
	8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
1	7	00 00 00 00	00 00 FF 07	80 69 FF FF	FF FF FF FF
	6	79 E9 F6 42	FF FF FF FF	FF FF 00 24	04 40 3F 04
	5	4B 4F 4E 45	20 20 20 20	20 20 20 20	20 20 20 20
	4	4A 41 46 41	49 53 20 48	45 43 48 4D	49 20 20 20

Nous visualisons les données de la carte et nous obtenons 4A 41 46 41 49 53 20 48 45 43 48 4D 49 pour le nom, 4B 4F 4E 45 pour le prénom et 79 E9 F6 42 pour le montant.

"4A 41 46 41 49 53 20 48 45 43 48 4D 49"$_{16}$ = "JAFAIS HECHMI"

"4B 4F 4^{E} 45"$_{16}$ = "KONE"

"79 E9 F6 42"$_{16}$ = "123.46" (valeur arrondie au dixième près)

D'après le code ASCII, ces données correspondent bien aux données que nous avons définies.

Maintenant, nous allons réaliser le programme pour visualiser les données stockées dans la carte en utilisant un web server.

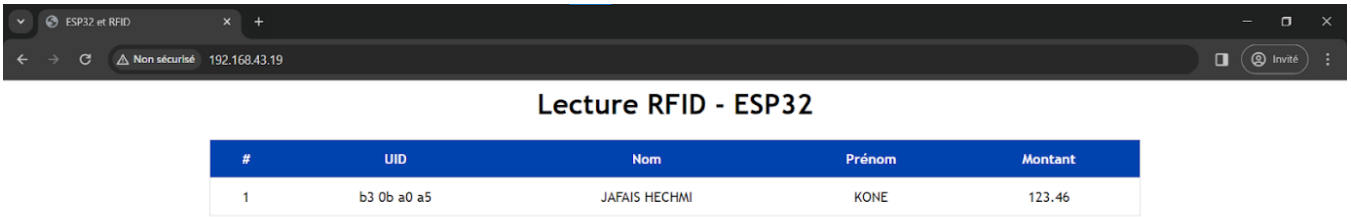
L'ESP32 va se connecter à un point d'accès, nous allons créer un server web sur ce réseau. Pour accéder au server web, il suffira d'entrer l'adresse IP fourni dans un navigateur. Nous utiliserons la fonction `mfrc522.MIFARE_Read()` pour la lecture des données. Le programme de lecture complet sur la carte RFID se trouve en ANNEXE 1.2.

Nous obtenons le résultat suivant dans le moniteur série.

```
Connexion au reseau WiFi
Nom du reseau WiFi : SMJ6
Adresse IP de la page web : 192.168.43.19
Le serveur web est en fonctionnement

PCD_Authenticate() block 4 success
Block 4 : JAFAIS HECHMI
PCD_Authenticate() block 5 success
Block 5 : KONE
PCD_Authenticate() block 6 success
Montant : 123.46
```

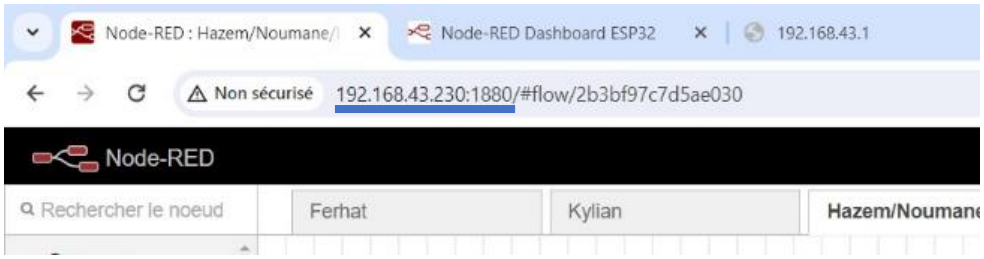
Nous mettons l'adresse IP de la page web dans un navigateur, nous passons la carte devant le lecteur et nous obtenons ce tableau. Nous pouvons constater que les informations correspondant bien aux données que nous avons fournies. À chaque lecture de carte, une nouvelle ligne apparaitra dans le tableau avec les données de la carte.



2ème partie : Plateforme IOT (NodeRED)

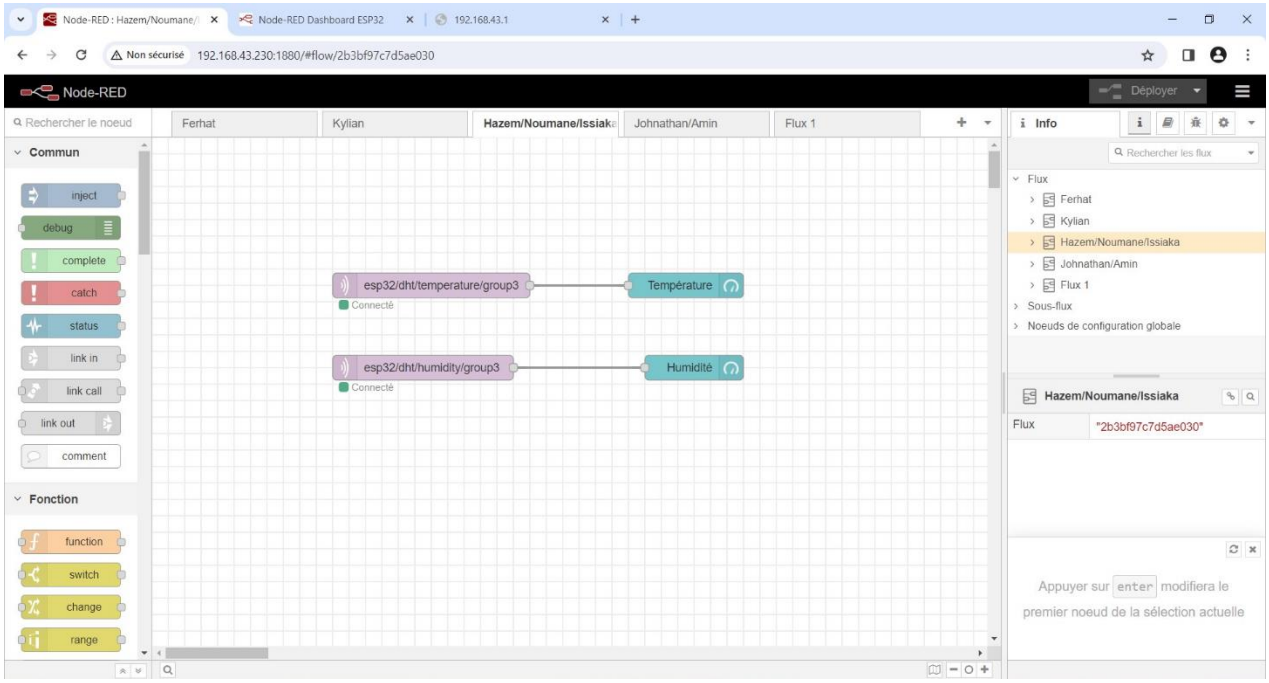
Node-RED est un environnement de programmation qui simplifie le développement en reliant des éléments. Avec une interface graphique intuitive, il permet la création d'applications en connectant diverses fonctions de manière efficace. Node-RED est souvent utilisé pour l'automatisation domotique et la gestion de données en temps réel. Node-RED peut être installé sur diverses plateformes (Windows, Linux …).

Dans le cadre du projet, Node-RED sera installé sur un Raspberry Pi. Le Raspberry sera connecté sur le même réseau que notre ordinateur. Pour accéder à Node-RED, il suffira d'entrer l'adresse IP du Raspberry Pi. (http://192.168.XX.XX :1880)

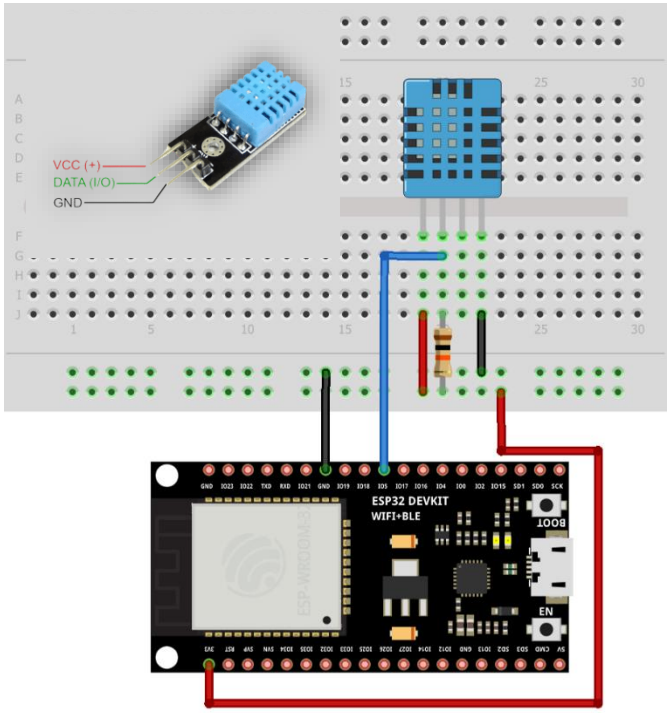


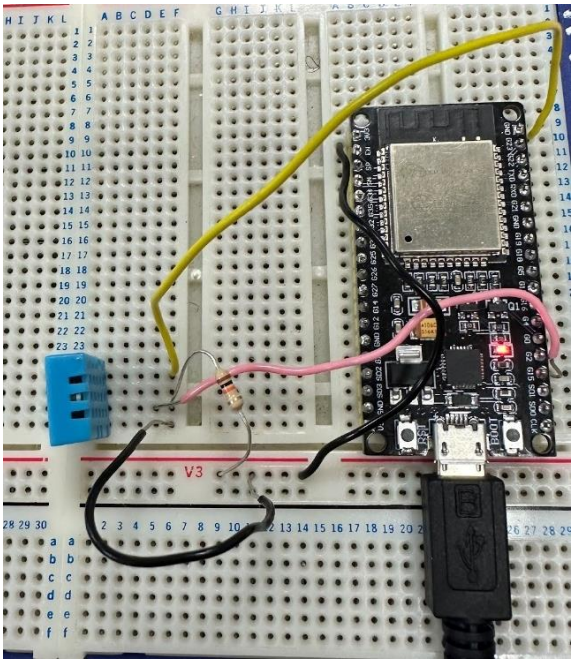
Pour avoir l'interface utilisateur sur Node-RED, nous devons installer une extension "Node-RED Dashboard". Les valeurs fournies par le capteur seront publiées dans topic de MQTT, pour récupérer ces valeurs des capteurs, nous devons être abonné au même topic, et nous aurons accès à cette valeur. Nous utiliserons donc des fonctions de MQTT. Nous allons également utiliser une gauge pour afficher les valeurs de température et de l'humidité.

Nous relions la fonction MQTT à la gauge correspondante.

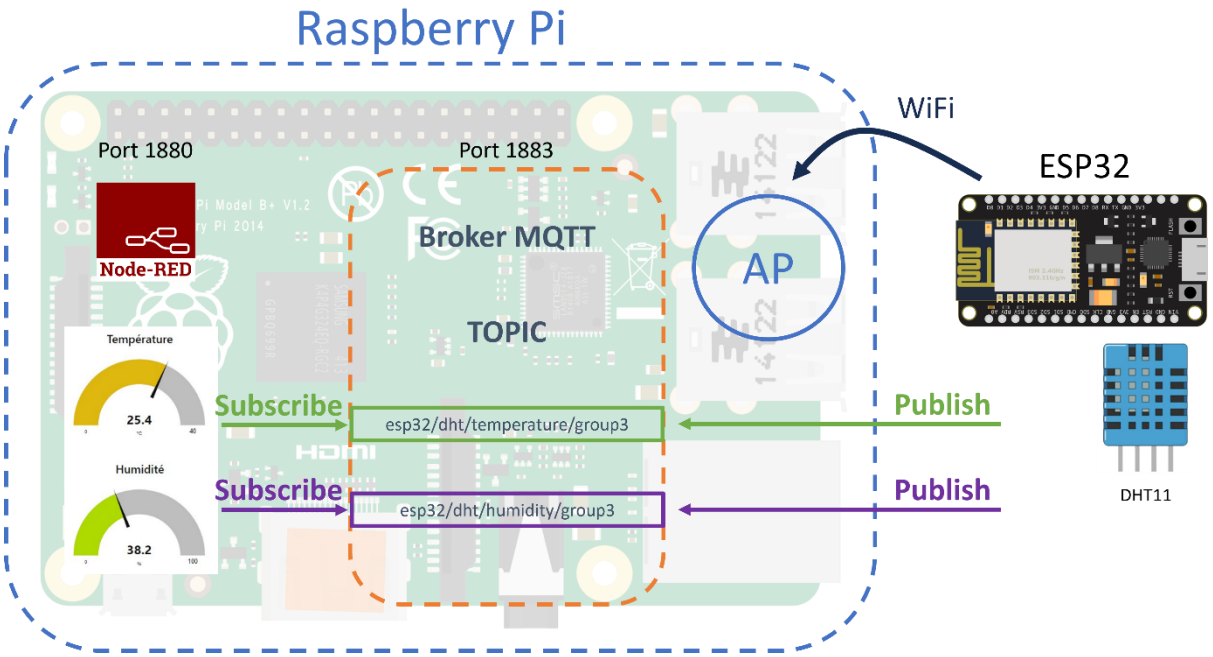


L’ESP32 sera câblé avec le capteur DHT11.





L’ESP32 va publier la valeur de température sur le topic “esp32/dht/temparature/group3” et la valeur d’humidité sur le topic “esp32/dht/humidity/group3”. Pour cela, nous utiliserons Arduino pour la programmation de la carte ESP32. L’ESP32 sera connectée au point d’accès du Raspberry Pi et au cloud MQTT du Raspberry Pi, il va pouvoir publier les valeurs sur les topics.



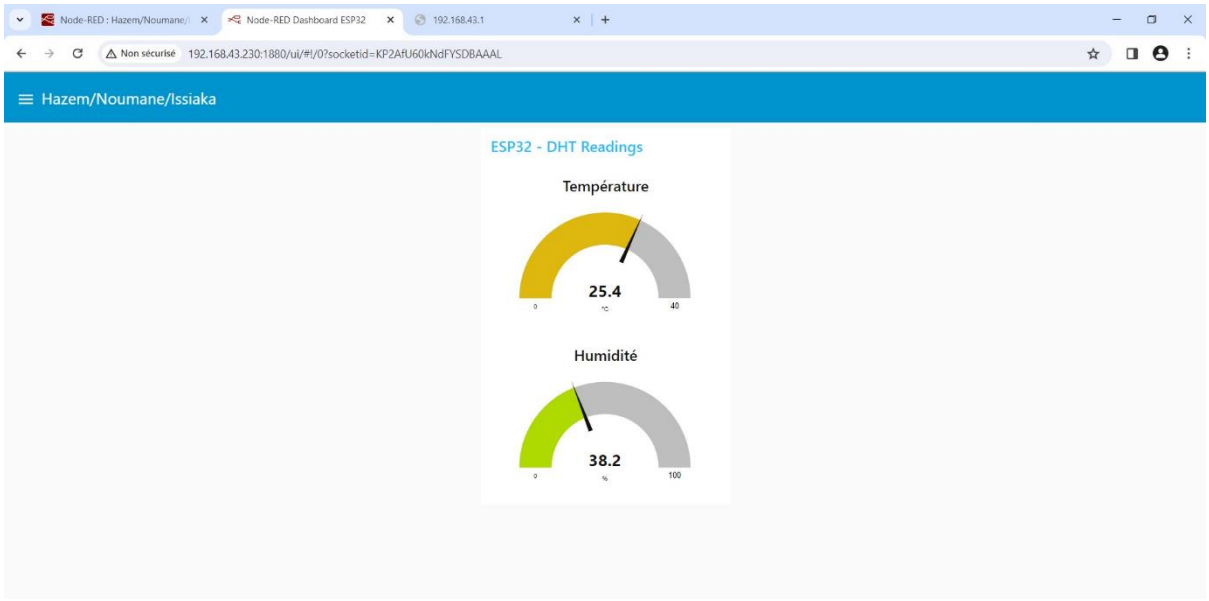
Le programme complet se trouve en ANNEXE 1.3.

Nous téléversons le programme et nous obtenons ce résultat.

```
Connecting to Wi-Fi...
[WiFi-event] event: 0
[WiFi-event] event: 2
[WiFi-event] event: 4
[WiFi-event] event: 7
WiFi connected
IP address :
192.168.43.50
Connecting to MQTT...
Connected to MQTT.
Session present: 0
Publishing on topic esp32/dht/temperature/group3 at QoS 1, packetId: 1 Message: 25.40
Publishing on topic esp32/dht/humidity/group3 at QoS 1, packetId: 2 Message: 38.20
```

Pour accéder à l'affichage du tableau de bord, il suffira d'entrer l'adresse IP du Raspberry Pi puis d'ajouter "/ui" (http ://192.168.XX.XX :1880/ui)

Nous visualisons les valeurs sur Node-RED.



Conclusion

Le projet s'est divisé sur deux axes. Ce projet IoT nous à permis de mettre en œuvre avec succès la lecture et l'écriture d'étiquettes électroniques RFID à l'aide d'un module RC522 et ESP32, ainsi que la visualisation des données via un serveur web. La seconde partie nous a permis d'établir une plateforme IoT avec Node-RED sur un Raspberry Pi, intégrant un capteur DHT11 pour la publication des valeurs de température et d'humidité via MQTT. Ce projet nous a permis une exploration approfondie des technologies de l'Internet des Objets. Ce projet nous a également permis de développer nos compétences personnelles en programmation Arduino et ainsi qu'en configuration de serveurs web et d'environnements Node-RED. Ce projet a été une opportunité d'approfondir nos connaissances et compétences dans le domaine émergent de l'IoT.

ANNEXE

1.1 Code Arduino RFID – Écriture

```
/**** Programme LECTURE RFID ******/

/**** Brochage des pins ******/

RC522    ---   ESP32

SS/SDA   -->   5
SCK      -->   18
MOSI     -->   23
MISO     -->   19
RST      -->   0
GND      -->   GND
3.3V     -->   3.3V

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

/* Configuration des pins */
#define SS_PIN 5
#define RST_PIN 0

MFRC522 mfrc522(SS_PIN, RST_PIN);
MFRC522::MIFARE_Key key;

/* Définition du bloc dans lequel nous voulons écrire des données */
int blockNum1 = 4;
int blockNum2 = 5;
int blockNum3 = 6;

/* Création d'un tableau pour lire les données des blocs*/
/* La longueur doit être supérieure de 2 octets à la taille du bloc (16 octets) */
byte bufferLen1 = 18;


/* Création des tableaux de 16 octets chacun et les remplir avec des données*/
byte blockData1[16] = { "JAFAIS HECHMI" };
byte blockData2[16] = { "KONE" };
byte blockData3[4];
float montant = 123.456;


MFRC522::StatusCode status;

void setup() {
  Serial.begin(9600);
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.println("Scan a Tag to write data...");
}
```

```
void loop() {

  for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
  }

  if (!mfrc522.PICC_IsNewCardPresent()) {
    return;
  }

  /* Select one of the cards */
  if (!mfrc522.PICC_ReadCardSerial()) {
    return;
  }

  Serial.print("\n");
  Serial.println("**Card Detected**");
  /* Ecrire UID de la carte */
  Serial.print(F("Card UID:"));
  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.print("\n");
  /* Ecrire le type de carte (par exemple, MIFARE 1K) */
  Serial.print(F("PICC type: "));
  MFRC522::PICC_Type piccType = mfrc522.PICC_GetType(mfrc522.uid.sak);
  Serial.println(mfrc522.PICC_GetTypeName(piccType));

  byte *tempBytes = (byte *)&montant;

  for (int i = 0; i < sizeof(montant); i++) {
    blockData3[i] = tempBytes[i];  // copy byte from float variable into buffer
  }

  Serial.print("\n");
  /* Ecriture des données */
  Serial.println("Writing to Data Blocks...");
  write_string(blockNum1, blockData1);
  write_string(blockNum2, blockData2);
  write_float(blockNum3, blockData3);


  /* Lecture des données */
  Serial.print("\n");
  Serial.println("Reading from Data Blocks...");
  read_string(blockNum1, blockData1, bufferLen1);
  read_string(blockNum2, blockData2, bufferLen1);
  read_float(blockNum3, blockData3, bufferLen1);

  delay(5000);
}
```

```
/* Boucle d’écriture d’une chaine de caractère */
void write_string(int blockNum, byte blockData[]) {
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNum,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print("PCD_Authenticate() block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    Serial.println("PCD_Authenticate() block " + String(blockNum) + " success");
  }

  for (byte i = strlen((char *)blockData); i < 16; i++) {
    blockData[i] = ' ';
  }

  status = mfrc522.MIFARE_Write(blockNum, blockData, 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("MIFARE_Write() block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    Serial.println("Writing block " + String(blockNum) + " success");
  }
}

/* Boucle d’écriture d’une valeur réelle */
void write_float(int blockNum, byte blockData[]) {
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNum,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print("PCD_Authenticate() block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    Serial.println("PCD_Authenticate() block " + String(blockNum) + " success");
  }

  status = mfrc522.MIFARE_Write(blockNum, blockData, 16);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("MIFARE_Write() block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    Serial.println("Writing block " + String(blockNum) + " success");
    Serial.println("Data Montant written successfully");
  }
}
```

```
/* Boucle de lecture d'une chaine de caractère */
void read_string(int blockNum, byte blockData[], byte bufferLen) {
  status = mfrc522.MIFARE_Read(blockNum, blockData, &bufferLen);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Reading block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    Serial.print("Block " + String(blockNum) + " : ");
    for (int i = 0; i < 16; i++) {
      Serial.write(blockData[i]);
    }
    Serial.println(" ");
  }
}

/* Boucle de lecture d'une valeur réelle */
void read_float(int blockNum, byte blockData[], byte bufferLen) {
  status = mfrc522.MIFARE_Read(blockNum, blockData, &bufferLen);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Reading block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    float readmontant;
    readmontant = *((float *)blockData);
    Serial.print(F("Montant : "));
    Serial.println(readmontant);
    Serial.println(" ");
  }
}
```

1.2 Code Arduino RFID – Lecture

```
/**** Programme LECTURE RFID ******/

/**** Brochage des pins ******/

RC522    ---   ESP32

SS/SDA   -->   5
SCK      -->   18
MOSI     -->   23
MISO     -->   19
RST      -->   0
GND      -->   GND
3.3V     -->   3.3V


#include <SPI.h>
#include <MFRC522.h>
#include "WiFi.h"
#include <WebServer.h>
#include <WiFiClient.h>

/* Point d'accès du reseau */
const char* ssid = "SMJ6";
const char* password = "12345679";

const int pinRST = 27;
const int pinSDA = 5;

/* Déclaration des variables pour les blocks */
int blockNum1 = 4;
int blockNum2 = 5;
int blockNum3 = 6;

byte bufferLen1 = 18;

/* Déclaration des variables pour le stockage des données */
byte blockData1[16];
byte blockData2[16];
byte blockData3[4];

int CardCount;

MFRC522 mfrc522(pinSDA, pinRST);
MFRC522::MIFARE_Key key;
MFRC522::StatusCode status;

WebServer serveur(80);

/* Construction de la page web */
void handle_root() {

  String contenu;  // les informations concernant les scans de tags RFID

  /* Boucle pour afficher les données sur tableau */
  for (int j = 1; j <= CardCount; j++) {
    if (CardCount != 0) {
      contenu.concat("<tr><td>" + String(j) + "</td>");
```

```
      contenu.concat("<td>");
      for (int i = 0; i < mfrc522.uid.size; i++) {
        contenu.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
        contenu.concat(String(mfrc522.uid.uidByte[i], HEX));
      }
      contenu.concat("</td>");

      contenu.concat("<td>");
      for (int i = 0; i < 16; i++) {
        contenu.concat(blockData1[i]);
      }
      contenu.concat("</td>");

      contenu.concat("<td>");
      for (int i = 0; i < 16; i++) {
        contenu.concat(blockData2[i]);
      }
      contenu.concat("</td>");

      contenu.concat("<td>");
      for (int i = 0; i < 4; i++) {
        contenu.concat(blockData3[i]);
      }
      contenu.concat("</td>");

      contenu.concat("</tr>");
    }
  }

  serveur.send(200, "html", "<head>  <title>ESP32 et RFID</title>
                <meta http-equiv=Refresh content=2><style>
                body {text-align: center; font-family: \"Trebuchet MS\", Arial;}
                table {border-collapse: collapse; width:70%; margin-left:auto;
margin-right:auto; }
                th {padding: 12px; background-color: #0043af; color: white; }
                tr {border: 1px solid #ddd; padding: 12px; }
                tr:hover { background-color: #bcbcbc; }
                td {border: none; text-align:center; padding: 12px; }
                </style>
                </head> <body>
                <H1> Lecture RFID - ESP32 </H1>
                <table>
                <tr><th>#</th>
                <th>UID</th>
                <th>Nom</th>
                <th>Pr&eacute;nom</th>
                <th>Montant</th>
                </tr> " + contenu + " </table></body></html>");

  delay(100);
}
```

```
// Initialisation du WiFi, connexion au WiFi et affichage de l'adresse IP

void setup() {
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.begin(9600);

  Serial.println();

  WiFi.begin(ssid, password);
  Serial.print("Connexion au reseau WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(1000);
  }

  Serial.println();

  Serial.print("Nom du reseau WiFi : ");
  Serial.println(ssid);
  Serial.print("Adresse IP de la page web : ");
  Serial.println(WiFi.localIP());

  serveur.on("/", handle_root);

  serveur.begin();
  Serial.println("Le serveur web est en fonctionnement");
}



void loop() {

  if (mfrc522.PICC_IsNewCardPresent())
  {
    if (mfrc522.PICC_ReadCardSerial())
    {
      CardCount++;
      read_string(blockNum1, blockData1, bufferLen1);
      read_string(blockNum2, blockData2, bufferLen1);
      read_float(blockNum3, blockData3, bufferLen1);

      delay(2000);
    }
  }

  serveur.handleClient();
}
```

```
/* Boucle lecture d'une chaines de caratères */
void read_string(int blockNum, byte blockData[], byte bufferLen) {

  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNum,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print("PCD_Authenticate() block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    Serial.println("PCD_Authenticate() block " + String(blockNum) + " success");
  }

  status = mfrc522.MIFARE_Read(blockNum, blockData, &bufferLen);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Reading block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    Serial.print("Block " + String(blockNum) + " : ");
    for (int i = 0; i < 16; i++) {
      Serial.write(blockData[i]);
    }
    Serial.println(" ");
  }
}

/* Boucle lecture d'une valeur réelle */
void read_float(int blockNum, byte blockData[], byte bufferLen) {
  status = mfrc522.PCD_Authenticate(MFRC522::PICC_CMD_MF_AUTH_KEY_A, blockNum,
&key, &(mfrc522.uid));
  if (status != MFRC522::STATUS_OK) {
    Serial.print("PCD_Authenticate() block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    Serial.println("PCD_Authenticate() block " + String(blockNum) + " success");
  }

  status = mfrc522.MIFARE_Read(blockNum, blockData, &bufferLen);
  if (status != MFRC522::STATUS_OK) {
    Serial.print("Reading block " + String(blockNum) + " failed: ");
    Serial.println(mfrc522.GetStatusCodeName(status));
    return;
  } else {
    float readmontant;
    readmontant = *((float*)blockData);
    Serial.print(F("Montant : "));
    Serial.println(readmontant);
    Serial.println(" ");
  }
}
```

1.3 Code Arduino NodeRed – DHT11

```
/**** Programme NodeRED ******/

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

// Point d'accès du Raspberry Pi
#define WIFI_SSID "AndroidAPD1CD"
#define WIFI_PASSWORD "GEII2024"

// Définition des paramètres du cloud MQTT du Raspberry Pi
#define MQTT_HOST IPAddress(192, 168, 43, 55)
#define MQTT_PORT 1883

// MQTT Topics
#define MQTT_PUB_TEMP "esp32/dht/temperature/group3"
#define MQTT_PUB_HUM "esp32/dht/humidity/group3"

// PIN connectée au capteur DHT
#define DHTPIN 15

// Définition du type de capteur
#define DHTTYPE DHT11

// Initialisation du capteur DHT
DHT dht(DHTPIN, DHTTYPE);

// Variables pour stocker les valeurs du capteur
float temp;
float hum;

AsyncMqttClient mqttClient;
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;


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

  dht.begin();

  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.setServer(MQTT_HOST, MQTT_PORT);
```

```
  connectToWifi();
}

void loop() {
  // Lecture des valeurs
  hum = dht.readHumidity();
  temp = dht.readTemperature();

  // Boucle de vérification si le capteur est bien connectée
  if (isnan(temp) || isnan(hum)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }

  // Publication de la temperature sur le topic esp32/dht/temperature/group3
  uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());
  Serial.printf("Publishing on topic %s at QoS 1, packetId: %i ", MQTT_PUB_TEMP,
packetIdPub1);
  Serial.printf("Message: %.2f \n", temp);

  // Publication de l'humidité sur le topic esp32/dht/humidity/group3
  uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(hum).c_str());
  Serial.printf("Publishing on topic %s at QoS 1, packetId: %i ", MQTT_PUB_HUM,
packetIdPub2);
  Serial.printf("Message: %.2f \n", hum);
}


void connectToWifi() {
  Serial.println("Connexion au Wi-Fi...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.println("WiFi connecté");
  Serial.println("IP address : ");
  Serial.println(WiFi.localIP());
  connectToMqtt();
}

void connectToMqtt() {
  Serial.println("Connexion au MQTT...");
  mqttClient.connect();
}


void onMqttConnect(bool sessionPresent) {
  Serial.println("Connecté au MQTT.");
}

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