



# *Rapport*

*Acquisition de Données avec Objets  
Connectés et Solutions Cloud.*

**Membre du groupe :**

...

# Rapport Projet IOT

---

## Introduction

De nos jours, la majorité des systèmes électroniques complexes que nous utilisons sont des systèmes embarqués. Ces systèmes sont présents dans une multitude d'appareils tels que les téléphones mobiles, les horloges, les baladeurs, les récepteurs GPS, les appareils électroménagers, les véhicules automobiles, ainsi que dans les secteurs du transport aérien, maritime et fluvial. Au cœur de ces systèmes embarqués se trouve un microcontrôleur, un circuit intégré qui réunit les éléments essentiels d'un ordinateur, à savoir un processeur, des mémoires, des périphériques et des interfaces d'entrées-sorties.

La carte Arduino est un outil puissant qui permet de mettre en œuvre le microcontrôleur ATmega328 de la société ATmel. Ce TP a pour objectif de vous familiariser avec la carte Arduino et son logiciel. À l'issue de ce travail, vous serez capable de mettre en service la carte et de télécharger un programme permettant de faire clignoter une LED. Vous découvrirez ainsi les sorties digitales de la carte Arduino et apprendrez à utiliser deux composants essentiels : la résistance et la LED.

Ensemble, nous allons explorer les bases de la programmation et de la mise en œuvre des microcontrôleurs à travers des exercices pratiques simples mais fondamentaux, vous préparant ainsi à créer vos propres projets électroniques embarqués.

Nous avons commencé par le plus simple :

### a. Allumer une LED. Voici comment procéder :

1. **Connexion des composants** : Connectez une LED et une résistance à la carte Arduino. La résistance doit être connectée en série avec la LED pour limiter le courant qui la traverse.
2. **Programmation** : Utilisez le logiciel Arduino pour écrire un programme simple qui envoie un signal à la sortie digitale de la carte, allumant ainsi la LED.
3. **Téléchargement du programme** : Téléchargez le programme sur la carte Arduino via le câble USB.
4. **Observation** : Une fois le programme téléchargé, observez la LED clignoter, indiquant que votre programme fonctionne correctement.

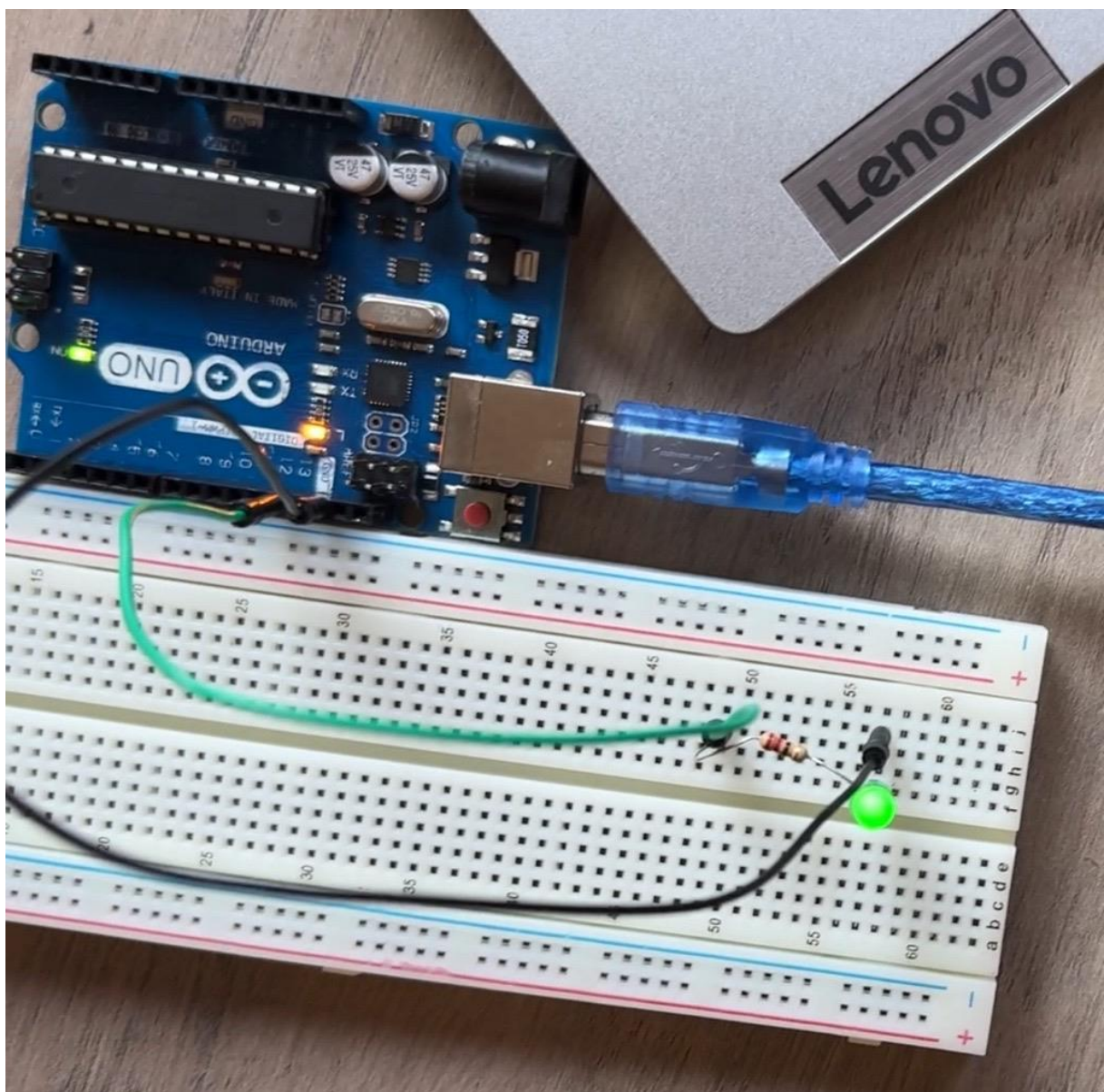
**Voici l'exemple du code utilisé :**

```

24 // Déclaration d'une variable 'led' pour indiquer le numéro de la broche utilisée pour le LED.
25 int led = 13;
26
27 // La fonction 'setup' s'exécute une seule fois lors du démarrage du microcontrôleur.
28 void setup() {
29     // Définir la broche 13 comme sortie pour contrôler la LED.
30     pinMode(led, OUTPUT);
31 }
32
33 // La fonction 'loop' s'exécute continuellement en boucle après l'exécution de 'setup'.
34 void loop() {
35     // Allumer la LED en envoyant un signal haut (HIGH) à la broche 13.
36     digitalWrite(led, HIGH);
37 }
38

```

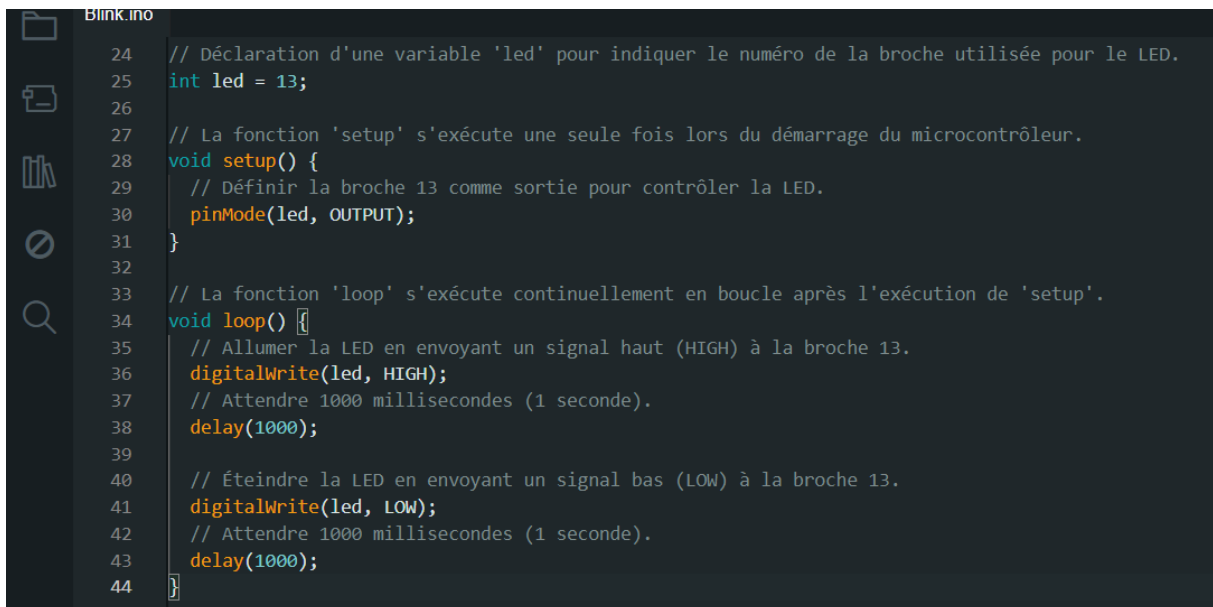
Et voici l'image à la pratique :



### b. Faire clignoter une LED avec delay :

1. **Modification du programme** : Ajoutez des instructions pour allumer et éteindre la LED avec des intervalles de temps définis. Utilisez la fonction delay pour introduire des pauses entre les changements d'état de la LED.

**Exemple de code :**



```
24 // Déclaration d'une variable 'led' pour indiquer le numéro de la broche utilisée pour le LED.
25 int led = 13;
26
27 // La fonction 'setup' s'exécute une seule fois lors du démarrage du microcontrôleur.
28 void setup() {
29     // Définir la broche 13 comme sortie pour contrôler la LED.
30     pinMode(led, OUTPUT);
31 }
32
33 // La fonction 'loop' s'exécute continuellement en boucle après l'exécution de 'setup'.
34 void loop() {
35     // Allumer la LED en envoyant un signal haut (HIGH) à la broche 13.
36     digitalWrite(led, HIGH);
37     // Attendre 1000 millisecondes (1 seconde).
38     delay(1000);
39
40     // Éteindre la LED en envoyant un signal bas (LOW) à la broche 13.
41     digitalWrite(led, LOW);
42     // Attendre 1000 millisecondes (1 seconde).
43     delay(1000);
44 }
```

2. **Téléchargement du programme** : Téléchargez le programme modifié sur la carte Arduino.
3. **Observation** : Une fois le programme téléchargé, observez la LED clignoter, alternant entre allumé et éteint avec une pause d'une seconde.

### c. Allumer Trois LED les unes après les autres avec Arduino

1. **Connexion des composants** : Connectez trois LED et trois résistances à la carte Arduino. Chaque résistance doit être connectée en série avec chaque LED pour limiter le courant.
2. **Programmation** : Écrivez un programme pour allumer les LED les unes après les autres avec un délai entre chaque allumage.

**Voici un exemple de code :**

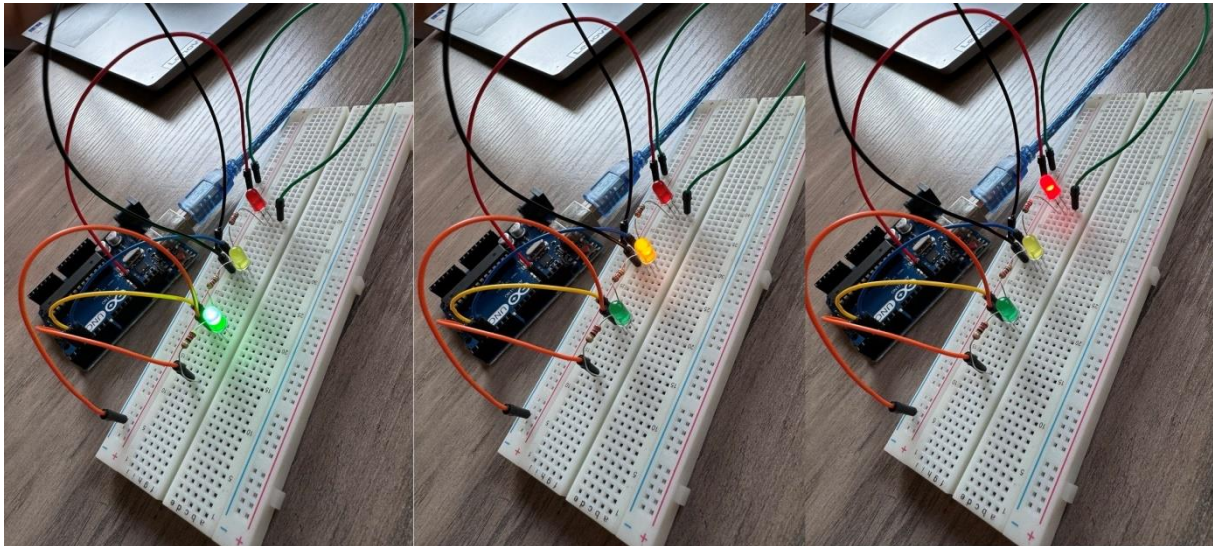


```

28 void setup() {
29     // Configurer les broches des LEDs comme sorties
30     pinMode(led1, OUTPUT);
31     pinMode(led2, OUTPUT);
32     pinMode(led3, OUTPUT);
33 }
34
35 void loop() {
36     // Allumer la LED 1
37     digitalWrite(led1, HIGH);
38     // Attendre 1 seconde
39     delay(1000);
40     // Éteindre la LED 1
41     digitalWrite(led1, LOW);
42
43     // Allumer la LED 2
44     digitalWrite(led2, HIGH);
45     // Attendre 1 seconde
46     delay(1000);
47     // Éteindre la LED 2
48     digitalWrite(led2, LOW);
49 }

```

**Voici la version pratique :**



#### d. Allumer une LED avec un bouton interrupteur sur Arduino :

##### 1. Connexion des composants :

- Connectez une LED et une résistance en série. Connectez la LED à une broche numérique de la carte Arduino (par exemple, la broche 2).
- Connectez un bouton interrupteur. Une borne du bouton doit être connectée à une broche numérique de la carte Arduino (par exemple, la broche 3), et l'autre borne doit être connectée à la masse (GND). Utilisez une résistance de pull-up interne pour maintenir un état stable lorsque le bouton n'est pas pressé.

##### 2. Programmation : Écrivez un programme pour allumer la LED lorsque le bouton est pressé Exemple de code :

```
23  */
24  const int ledPin = 13;    // Broche où est connectée la LED
25  const int buttonPin = 2;  // Broche où est connecté le bouton
26
27  int ledState = LOW;        // Variable pour stocker l'état de la LED
28  int buttonState;           // Variable pour lire l'état du bouton
29  int lastButtonState = LOW; // Variable pour stocker le dernier état du bouton
30  unsigned long lastDebounceTime = 0; // Dernière fois que l'état de la broche a été changé
31  unsigned long debounceDelay = 50;   // Délai de désencombrement
32
33  void setup() {
34      pinMode(ledPin, OUTPUT);    // Initialise la broche de la LED comme une sortie
35      pinMode(buttonPin, INPUT);  // Initialise la broche du bouton comme une entrée
36      digitalWrite(ledPin, ledState); // Initialise la LED à l'état initial
37  }
38
39  void loop() {
40      int reading = digitalRead(buttonPin);
41
42      // Vérifie s'il y a un changement dans la lecture du bouton
43      if (reading != lastButtonState) {
```

```

if (reading != lastButtonState) {
    lastDebounceTime = millis();
}

// Si le temps depuis le dernier changement d'état est supérieur au d
if ((millis() - lastDebounceTime) > debounceDelay) {
    // Si l'état du bouton a changé
    if (reading != buttonState) {
        buttonState = reading;
        // Si le nouvel état du bouton est HIGH, bascule l'état de la LED
        if (buttonState == HIGH) {
            ledState = !ledState;
        }
    }
}

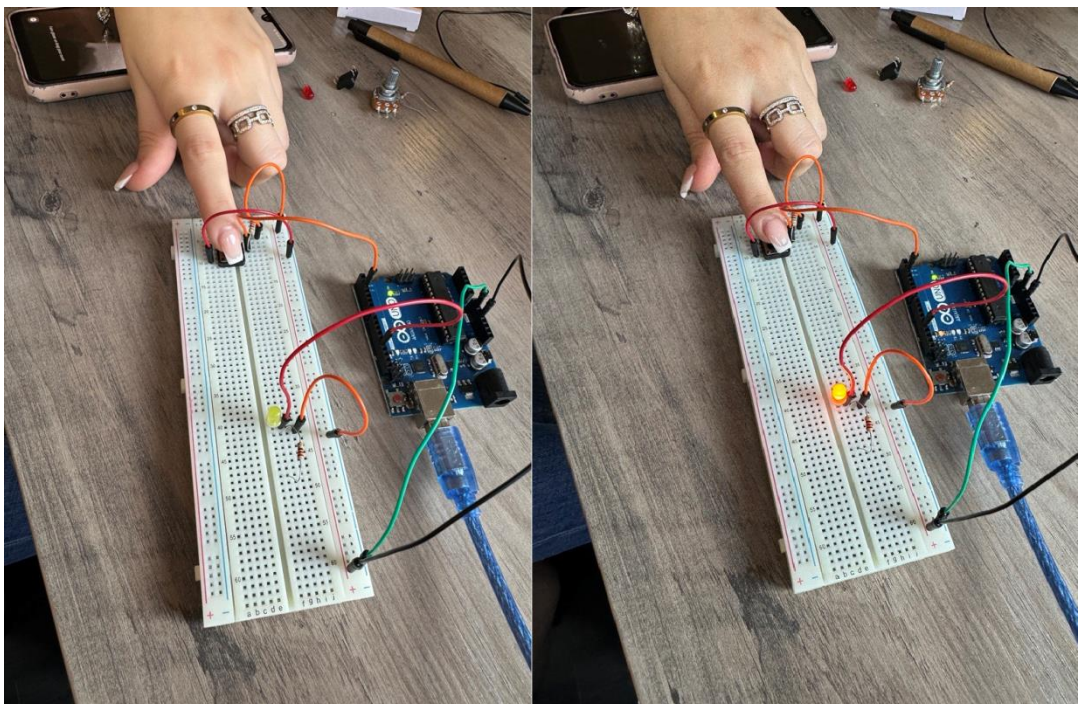
// Change l'état de la LED
digitalWrite(ledPin, ledState);

// Stocke l'état actuel du bouton pour le comparer au prochain cycle
lastButtonState = reading;

```

3. **Téléchargement du programme** : Téléchargez le programme sur la carte Arduino via le câble USB.
4. **Observation** : Une fois le programme téléchargé, appuyez sur le bouton pour allumer la LED. Relâchez le bouton pour éteindre la LED.

Voici un aperçu pratique :



## e. Étapes pour Allumer une LED avec un Capteur Ultrasonique sur Arduino

### 1. Connexion des composants :

- Connectez une LED et une résistance en série à une broche numérique de la carte Arduino (par exemple, la broche 2).
- Connectez un capteur ultrasonique (par exemple, le HC-SR04) à la carte Arduino :
  - VCC à 5V
  - GND à GND
  - Trig à une broche numérique (par exemple, la broche 9)
  - Echo à une autre broche numérique (par exemple, la broche 10)

### 2. Programmation :

- Écrivez un programme pour mesurer la distance avec le capteur ultrasonique et allumer la LED si un objet est détecté à une certaine distance.

```
24 const int trigPin = 9;    // Broche TRIG du capteur à ultrasons
25 const int echoPin = 10;   // Broche ECHO du capteur à ultrasons
26 const int ledPin = 13;    // Broche de la LED
27
28 long duration;
29 int distance;
30 const int distanceThreshold = 10; // Seuil de distance en cm pour allumer la LED
31
32 void setup() {
33     pinMode(trigPin, OUTPUT); // Initialise la broche TRIG comme une sortie
34     pinMode(echoPin, INPUT);  // Initialise la broche ECHO comme une entrée
35     pinMode(ledPin, OUTPUT);  // Initialise la broche de la LED comme une sortie
36
37     Serial.begin(9600); // Démarre la communication série à 9600 bauds
38 }
39
40 void loop() {
41     // Efface le trigPin en le passant à LOW
42     digitalWrite(trigPin, LOW);
43     delayMicroseconds(2);
44
45     // Configure le trigPin à HIGH pendant 10 microsecondes pour envoyer l'impulsion sonore
```

```
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Lit l'impulsion renvoyée sur le echoPin
    duration = pulseIn(echoPin, HIGH);

    // Calcule la distance (en cm)
    distance = duration * 0.034 / 2;

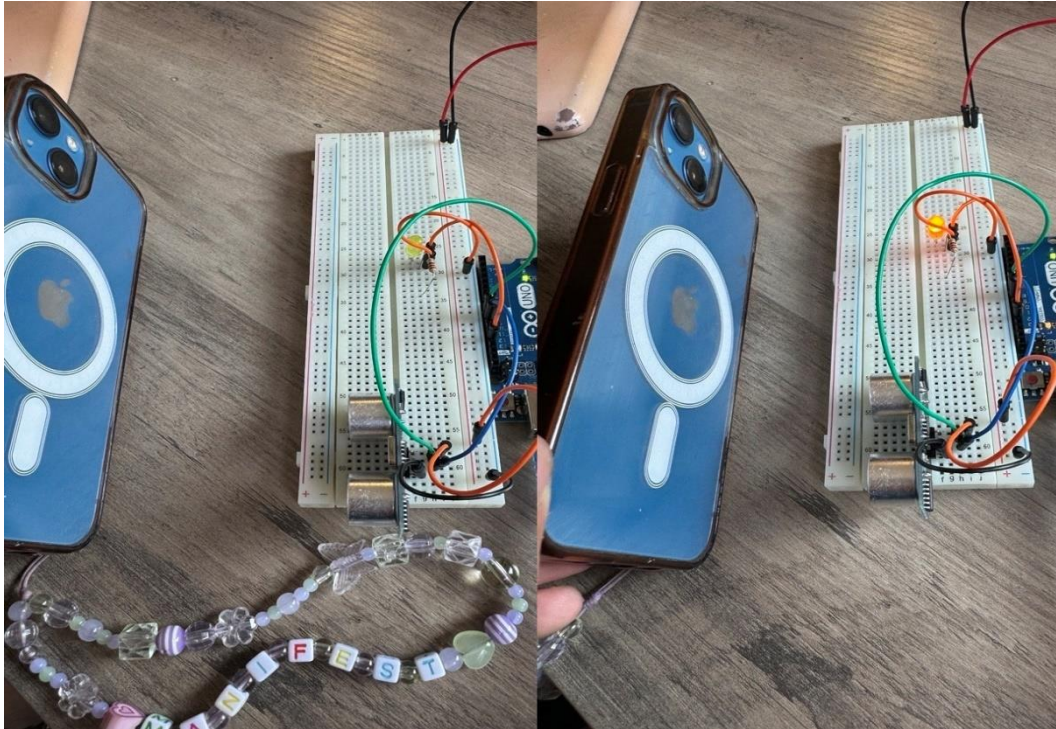
    // Affiche la distance sur le moniteur série
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");

    // Allume ou éteint la LED en fonction de la distance
    if (distance < distanceThreshold) {
        digitalWrite(ledPin, HIGH); // Allume la LED si l'objet est à moins de 10 cm
    } else {
        digitalWrite(ledPin, LOW);  // Éteint la LED sinon
    }
}
```



- **Téléchargement du programme :**
  - Téléchargez le programme sur la carte Arduino via le câble USB.
- **Observation :**
  - Une fois le programme téléchargé, approchez un objet du capteur ultrasonique. La LED s'allume lorsqu'un objet est détecté à une distance inférieure à 10 cm et s'éteint lorsque l'objet s'éloigne.

Voici un aperçu pratique :



#### f. Étapes pour Utiliser un Capteur d'Eau avec Arduino

1. **Connexion des composants :**
  - Connectez une LED et une résistance en série à une broche numérique de la carte Arduino (par exemple, la broche 2).
  - Connectez un capteur d'eau à la carte Arduino :
    - VCC à 5V
    - GND à GND
    - Signal à une broche analogique (par exemple, la broche A0)
2. **Programmation :**
  - Écrivez un programme pour lire la valeur du capteur d'eau et allumer la LED si le niveau d'eau est bas (valeur inférieure à 500).

Voici le bout de code :

```

#include <Arduino.h>

// Broche du capteur d'humidité de sol
const int waterSensorPin = A0;

// Seuil de niveau d'eau bas
const int seuilBas = 500; // Valeur à ajuster en fonction du calibrage du capteur

void setup() {
  // Initialisation de la communication série pour le débogage
  Serial.begin(9600);
}

void loop() {
  // Lecture de la valeur analogique du capteur d'humidité de sol
  int valeurCapteur = analogRead(waterSensorPin);

  // Affichage de la valeur lue pour débogage
  Serial.print("Valeur du capteur : ");

```

```

// Lecture de la valeur analogique du capteur d'humidité de sol
int valeurCapteur = analogRead(waterSensorPin);

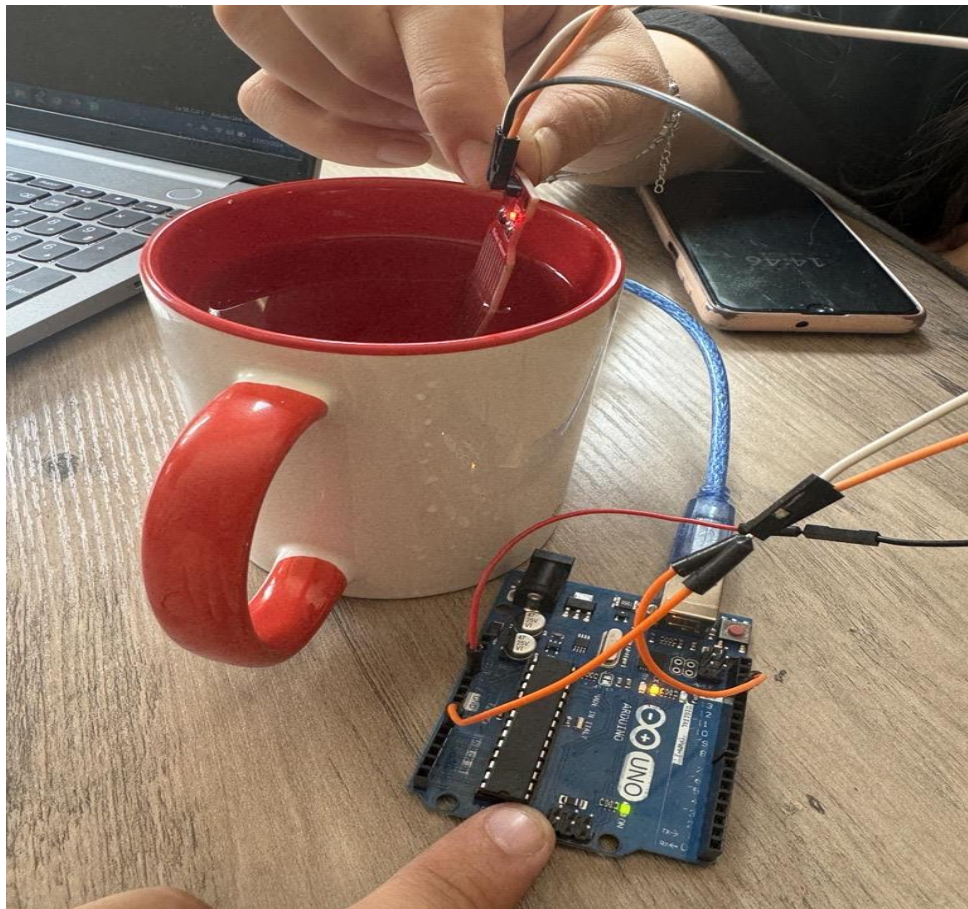
// Affichage de la valeur lue pour débogage
Serial.print("Valeur du capteur : ");
Serial.println(valeurCapteur);

// Vérification du niveau d'eau
if (valeurCapteur < seuilBas) {
  Serial.println("Niveau d'eau bas !");
  // Ici vous pouvez ajouter des actions en cas de niveau d'e
  // Par exemple, allumer une LED ou déclencher une alarme
}

delay(1000); // Attente d'une seconde entre chaque mesure
}

```

Voici la mise en place :



### 3. Téléchargement du programme :

- Téléchargez le programme sur la carte Arduino via le câble USB.

### 4. Observation :

- Une fois le programme téléchargé, observez la valeur lue par le capteur d'eau. La LED s'allume lorsque la valeur est inférieure à 500, indiquant un niveau d'eau bas, et s'éteint lorsque la valeur est supérieure à 500.

## Cohérence avec notre projet ArrolOT –Utilisation d’eau pour un projet d’Arrosage Automatique- :

L'utilisation d'un capteur d'eau peut être extrêmement bénéfique pour un projet d'arrosage automatique. Voici comment cela peut vous aider :

1. **Détection du Niveau d'Eau** : Le capteur d'eau permet de détecter le niveau d'humidité du sol. Si le niveau d'eau est bas (valeur inférieure à 500), cela indique que le sol est sec et a besoin d'être arrosé.
2. **Automatisation de l'Arrosage** : En intégrant le capteur d'eau avec un système de pompe à eau contrôlé par Arduino, vous pouvez automatiser le processus d'arrosage. Lorsque le capteur détecte un niveau d'eau bas, l'Arduino peut activer la pompe pour arroser les plantes.

3. **Conservation de l'Eau** : En arrosant uniquement lorsque c'est nécessaire, vous évitez le gaspillage d'eau. Le système assure que les plantes reçoivent la bonne quantité d'eau, améliorant ainsi l'efficacité de l'arrosage.
4. **Maintenance Minimale** : Un système d'arrosage automatique réduit la nécessité de surveiller constamment l'humidité du sol. Cela est particulièrement utile pour les jardins ou les cultures où un entretien manuel régulier peut être difficile.
5. **Personnalisation** : Le seuil de détection (500 dans ce cas) peut être ajusté en fonction des besoins spécifiques de vos plantes et des conditions de votre sol. Vous pouvez affiner le système pour répondre précisément aux exigences de votre projet.

### Explication du Processus de Lecture d'une Carte\Badge RFID avec Arduino

1. **Inclusion des Bibliothèques et Configuration des Broches :**
  - Le programme commence par inclure les bibliothèques nécessaires pour la communication SPI et pour le lecteur RFID RC522.
  - Les broches de la carte Arduino sont configurées pour se connecter au lecteur RFID. Les broches spécifiques pour la communication SPI (Slave Select et Reset) sont définies.
2. **Initialisation dans la Fonction setup():**
  - La communication série est initialisée, ce qui permet d'envoyer des messages au moniteur série pour le débogage et l'affichage des données lues.
  - La communication SPI est également initialisée pour permettre la communication entre l'Arduino et le lecteur RFID.
  - Le lecteur RFID est initialisé pour être prêt à lire les cartes RFID.
3. **Boucle Principale loop():**
  - La boucle principale vérifie continuellement si une nouvelle carte RFID est présente à proximité du lecteur.
  - Si une nouvelle carte est détectée, le lecteur tente de lire l'UID (identifiant unique) de la carte.
  - Si l'UID de la carte est lu avec succès, le programme l'affiche dans le moniteur série sous forme de chiffres hexadécimaux.
4. **Affichage et Terminaison de la Communication :**
  - L'UID de la carte RFID est imprimé sur le moniteur série, permettant à l'utilisateur de voir l'identifiant unique de la carte approchée.
  - La communication avec la carte RFID est terminée après avoir lu l'UID, et le programme est prêt à détecter une nouvelle carte.

Voici le code :

```

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

// Broches du module RFID MFRC522
#define RST_PIN 9    // Broche RST connectée à la broche 9
#define SS_PIN 10    // Broche SDA (ou SS) connectée à la broche 10

// UID de la carte RFID attendue (C7 06 06 B3)
byte carteAttendue[] = {0x23, 0x7A, 0x51, 0x03};

MFRC522 mfrc522(SS_PIN, RST_PIN); // Création de l'objet MFRC522

// Broche pour la LED
const int ledPin = 7; // Broche 7 de l'Arduino

void setup() {
  Serial.begin(9600); // Initialisation de la communication série
  pinMode(ledPin, OUTPUT); // Définition de la broche de la LED comme sortie

  SPI.begin(); // Initialisation de SPI bus
  mfrc522.PCD_Init(); // Initialisation du module MFRC522
}

```

```

46
47   SPI.begin(); // Initialisation de SPI bus
48   mfrc522.PCD_Init(); // Initialisation du module MFRC522
49   Serial.println("Lecteur RFID MFRC522 prêt !");
50 }
51
52 void loop() {
53   // Vérifie si une nouvelle carte est détectée
54   if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
55     // Vérifie si l'UID de la carte correspond à celui attendu
56     if (verifierUID()) {
57       Serial.println("Carte valide détectée !");
58       digitalWrite(ledPin, HIGH); // Allume la LED
59       delay(1000); // Garde la LED allumée pendant 1 seconde
60       digitalWrite(ledPin, LOW); // Éteint la LED
61     }
62     else {
63       Serial.println("Carte invalide !");
64     }
65   }

```



```

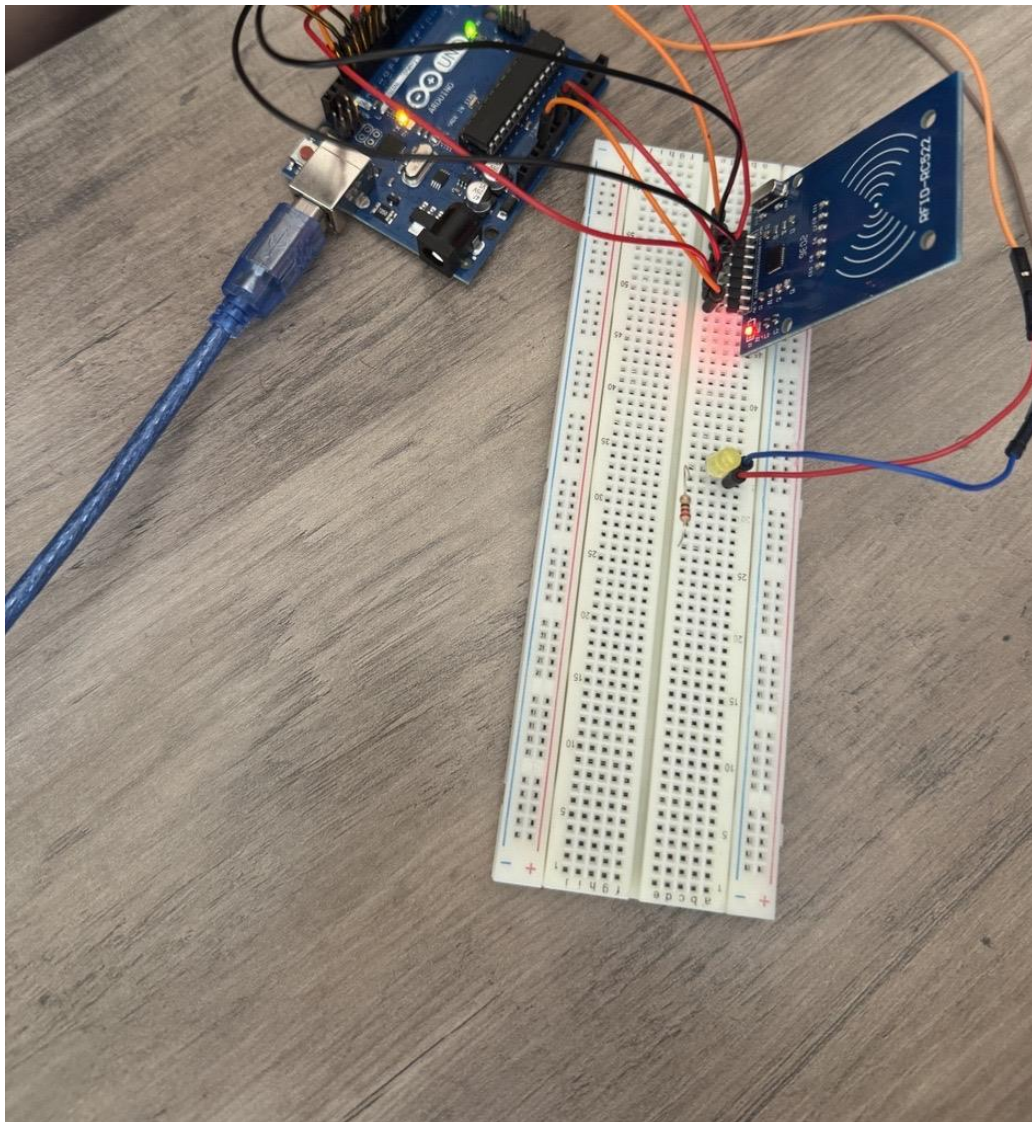
    else {
        Serial.println("Carte invalide !");
    }

    // Attente avant de lire une autre carte
    delay(1000);
}
}

// Fonction pour vérifier si l'UID de la carte correspond à celui attendu
bool verifierUID() {
    if (mfrc522.uid.size == sizeof(carteAttendue)) {
        for (byte i = 0; i < mfrc522.uid.size; i++) {
            if (mfrc522.uid.uidByte[i] != carteAttendue[i]) {
                return false;
            }
        }
        return true;
    }
    return false;
}
}

```

Voici la mise en pratique :



### Utilisation du Lecteur de Carte RFID dans un Projet d'Arrosage Automatique

#### 1. Accès Sécurisé :

- Vous pouvez utiliser des cartes RFID pour contrôler l'accès au système d'arrosage automatique, permettant seulement aux utilisateurs autorisés d'activer ou de désactiver le système.

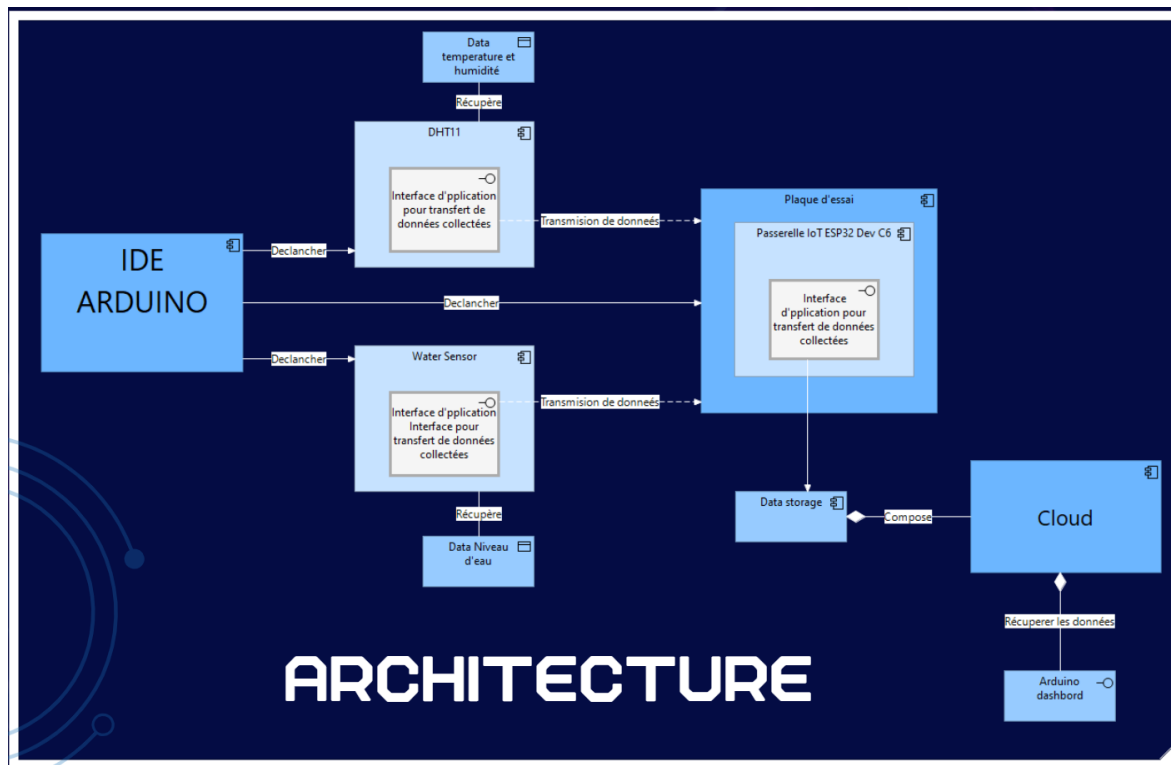
#### 2. Identification et Personnalisation :

- Chaque utilisateur peut être identifié par son UID unique, permettant des paramètres d'arrosage personnalisés pour différents utilisateurs ou zones spécifiques.

#### 3. Automatisation et Suivi :

- En intégrant le lecteur RFID avec d'autres capteurs (comme des capteurs d'humidité), le système peut activer l'arrosage en fonction de la détection d'une carte RFID et des besoins actuels du sol.
- Le système peut enregistrer les interactions avec les cartes RFID, permettant de suivre l'utilisation et d'optimiser les cycles d'arrosage.

## Architecture de la solution proposée



Une architecture IoT que nous avons mise en place pour ce système d'arrosage et de gestion de l'environnement utilisant des capteurs et des technologies cloud.

### Composants Clés de l'Architecture

#### 1. Capteurs et Microcontrôleurs

- Water Sensor: Ce capteur collecte des données sur la présence et le niveau d'eau.
- DHT11: Un capteur qui mesure la température et l'humidité.
- ESP32 Dev C6: Ce microcontrôleur sert de passerelle IoT, centralisant les données collectées par les différents capteurs.

#### 2. Plaque d'essai (Breadboard)

- Breadboard: Elle regroupe et connecte le capteur DHT11 et l'ESP32, permettant
- une intégration facile et modulable des capteurs.

#### 3. Passerelle IoT

- ESP32 Dev C6: En tant que passerelle, il reçoit les données des capteurs via la
- plaque d'essai et les transmet à l'interface de réception de données.

#### 4. Service Série et Configuration

- Arduino IDE: L'IDE Arduino configure et envoie des ordres à l'ESP32. Il sert également

## **5. Cloud Storage et Traitement**

- Cloud Storage & Processing: Les données collectées sont envoyées vers le cloud où
  - elles sont stockées et analysées pour fournir des insights pertinents.

## **6. Dashboard**

- Arduino Dashboard: Une interface de visualisation qui affiche les données sous
  - forme de graphes, permettant une interprétation rapide et intuitive des informations collectées.

## **Relations et Flux de Données**

- Flow: Le flux de données commence aux capteurs (Water Sensor et DHT11) qui envoient les
  - données collectées à la plaque d'essai. Ces données sont ensuite transmises au microcontrôleur
  - ESP32 qui les relaie à l'Arduino IDE via le service série. L'Arduino IDE transmet ensuite ces données
    - au cloud pour stockage et traitement. Finalement, les données traitées sont envoyées au dashboard
      - pour visualisation.
  - Triggering: L'interaction entre les composants se déclenche comme suit: le DHT11 envoie les
    - données à la breadboard, qui les transmet ensuite à l'ESP32 Dev C6. L'Arduino IDE configure et
      - déclenche les actions de l'ESP32.
    - Access: Les données stockées et analysées dans le cloud sont accessibles par le dashboard pour
      - visualisation en temps réel.





## Découvrez l'ESP32 : La Puissance au Cœur de vos Projets IoT

L'ESP32, développé par Espressif Systems, est un microcontrôleur puissant et polyvalent, parfait pour les projets IoT et les applications embarquées.

### Caractéristiques principales :

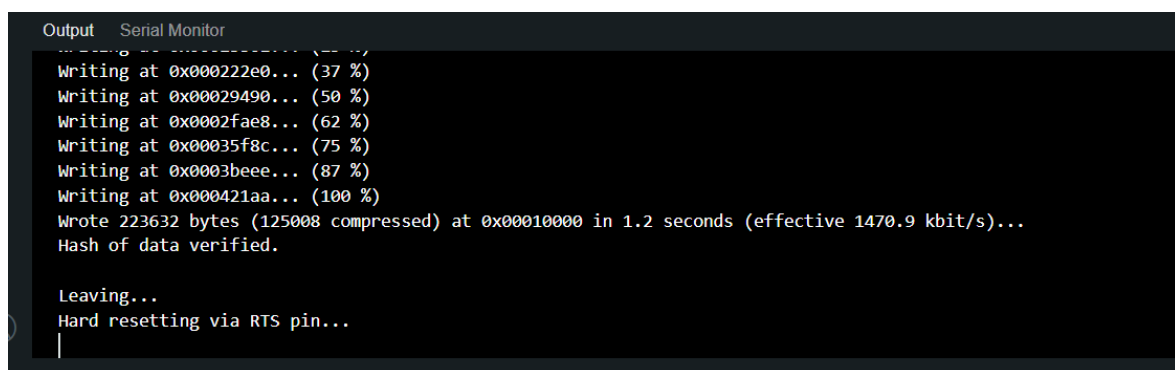
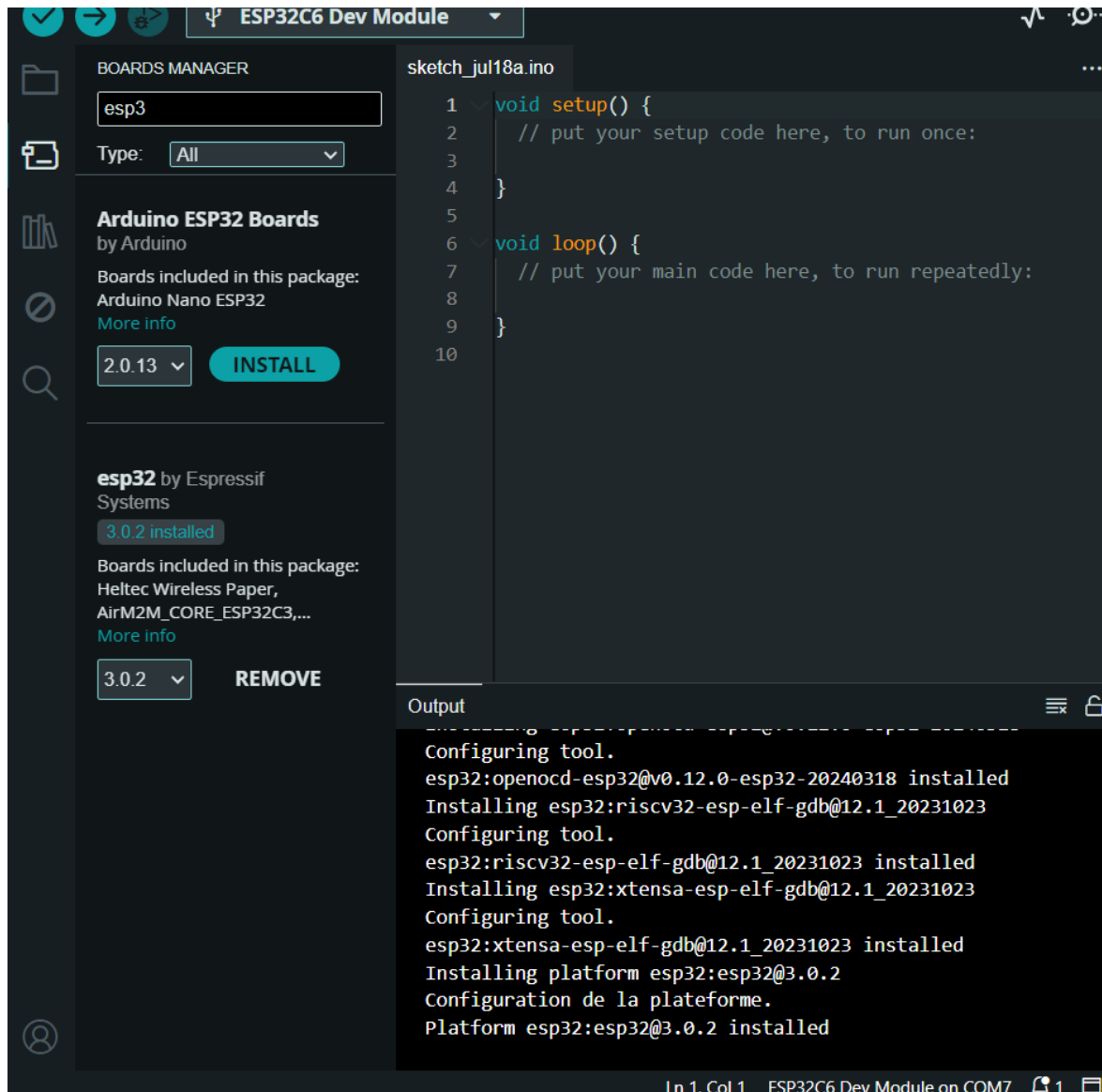
- **Connectivité** : Wi-Fi et Bluetooth intégrés.
- **Processeur** : Dual-core jusqu'à 240 MHz.
- **Interfaces** : Nombreux ports GPIO, ADC/DAC, UART, SPI, I2C, I2S, CAN.
- **Sécurité** : Cryptage matériel, démarrage sécurisé.
- **Développement** : Compatible avec l'IDE Arduino, PlatformIO, et ESP-IDF.

L'ESP32 est idéal pour des projets allant des maisons intelligentes aux systèmes industriels.

### Paramétrage d'une Carte ESP32 dans l'IDE Arduino

1. **Installer l'IDE Arduino** : On télécharge et on installe la dernière version de l'IDE Arduino.
2. **Ajouter le Gestionnaire de Cartes ESP32** : On ouvre l'IDE, on va dans les préférences et on ajoute l'URL suivante dans le champ des URLs supplémentaires des gestionnaires de cartes : [`https://dl.espressif.com/dl/package\\_esp32\\_index.json`](https://dl.espressif.com/dl/package_esp32_index.json).
3. **Installer les Cartes ESP32** : On accède au gestionnaire de cartes dans le menu Outils, on recherche "esp32" et on installe le paquet `esp32 by Espressif Systems`.
4. **Sélectionner la Carte ESP32** : Dans le menu Outils, on sélectionne son modèle d'ESP32.
5. **Sélectionner le Port** : On connecte l'ESP32 via USB et on sélectionne le port COM approprié dans le menu Outils.
6. **Configurer les Paramètres de la Carte** : On configure les options spécifiques à sa carte ESP32 dans le menu Outils.

Ces étapes permettent de configurer l'IDE Arduino pour programmer une carte ESP32.



### Connexion au Réseau Wi-Fi :

- **Paramétrer le Wi-Fi** : On écrit un programme pour connecter l'ESP32 à un réseau Wi-Fi en spécifiant le SSID (nom du réseau) et le mot de passe.
- **Téléverser le Programme** : On téléverse le programme sur la carte ESP32 via l'IDE Arduino.

- **Vérifier la Connexion** : On vérifie que l'ESP32 se connecte correctement au réseau Wi-Fi et affiche son adresse IP

```
34
35 void loop() {
36
37     delay(1000);
38
39
40
41     if (WiFi.status() == WL_CONNECTED) {
42
43         Serial.print("Est connecté au réseau avec ");
44
45         Serial.println(WiFi.localIP());
46
47     }
48
49     else {
50
51         Serial.println("N'est pas connecté au réseau");
52
53     }
54
55 }
56
```

sketch\_jul18a | Arduino IDE 2.3.3-nightly-20240630

File Edit Sketch Tools Help

ESP32C6 Dev Module

sketch\_jul18a.ino

```
1  #include "WiFi.h"
2
3  const char* ssid = "Livebox-D7D0";
4
5  const char* pwd  = "MRo4MEqDrMt6aaDRm5";
6
7
8
9  void setup() {
10
11     Serial.begin(9600);
12
13     Serial.println("Configuration du WiFi");
14
15     WiFi.begin(ssid, pwd);
16
17
18
19     while (WiFi.status() != WL_CONNECTED) {
20
21         delay(500);
22
23         Serial.println("Connexion au WiFi..");
24
25     }
26
27
28
29     Serial.println("Connecté au réseau WiFi");
30
31 }
32
33
```

```
sketch_jul18a.ino
1  #include "WiFi.h"
2
3  const char* ssid = "Livebox-D7D0";
4
5  const char* pwd = "MRo4MEqDrMt6aaDRm5";
6
7
8
9  void setup() {
10
11     Serial.begin(9600);
12
13     Serial.println("Configuration du WiFi");
14
15     WiFi.begin(ssid, pwd);
16
17
18
19     while (WiFi.status() != WL_CONNECTED) {
20         delay(500);
21     }
22 }
```

Output Serial Monitor

```
Writing at 0x000c9ec0... (85 %)
Writing at 0x000d054f... (88 %)
Writing at 0x000d63be... (91 %)
Writing at 0x000dc04b... (94 %)
Writing at 0x000e1afa... (97 %)
Writing at 0x000e91ce... (100 %)
Wrote 892240 bytes (558447 compressed) at 0x00010000 in 3.8 seconds (effective 1867.5 kbit/s)...
Hash of data verified.

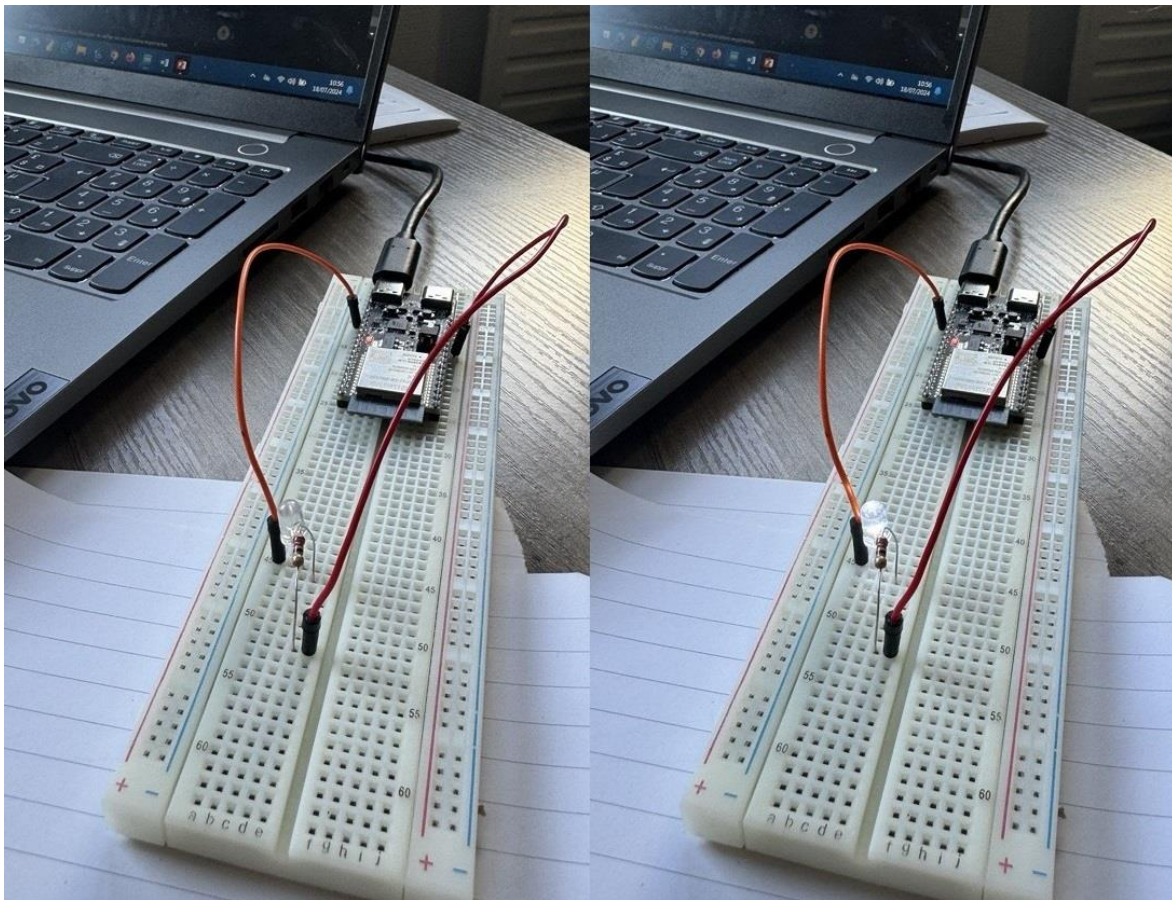
Leaving...
Hard resetting via RTS pin...
```

## Mise en pratique : Test de connectivité avec une LED

```
26 // Constantes
27 // Définir la broche de la LED
28 const int ledPin = 2; // Utilisez la broche GPIO 2 pour la LED
29
30 void setup() {
31     // Initialiser la broche de la LED comme une sortie
32     pinMode(ledPin, OUTPUT);
33 }
34
35 void loop() {
36     // Allumer la LED
37     digitalWrite(ledPin, HIGH);
38     delay(1000); // Attendre 1 seconde
39
40     // Éteindre la LED
41     digitalWrite(ledPin, LOW);
42     delay(1000); // Attendre 1 seconde
43 }
44
45
```



Voici le cas pratique :



## Vérifier la Température et Allumer une LED :

### 1. Écrire le Programme :

- Connecter un capteur de température (par exemple, DHT22 ou DS18B20) à l'ESP32.
- Le programme doit :
  - Se connecter au réseau Wi-Fi.
  - Lire la température du capteur.
  - Allumer une LED si la température dépasse 25°C.

Voici le programme :

```
#include <DHT.h>

// Définir les pins
#define DHTPIN 4      // Pin du capteur DHT11
#define DHTTYPE DHT11 // Type de capteur DHT
#define LEDPIN 5      // Pin de la LED

DHT dht(DHTPIN, DHTTYPE);

void setup() {
    // Initialiser la communication série
    Serial.begin(115200);

    // Initialiser le DHT11
    dht.begin();

    // Initialiser la pin LED comme sortie
    pinMode(LEDPIN, OUTPUT);

    // Éteindre la LED au démarrage
    digitalWrite(LEDPIN, LOW);
}

void loop() {
    // Lecture de la température et de l'humidité
    float h = dht.readHumidity();
```

```

// Afficher les valeurs lues sur le moniteur série
Serial.print("Humidité: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Température: ");
Serial.print(t);
Serial.println(" *C");

// Allumer la LED si la température dépasse 25 degrés Celsius
if (t > 45.0) {
|   digitalWrite(LEDPIN, HIGH);
} else {
|   digitalWrite(LEDPIN, LOW);
}

// Attendre 2 secondes avant la prochaine lecture
delay(2000);
}

```

## 2. Téléverser le Programme :

- On téléverse le programme sur l'ESP32 via l'IDE Arduino.

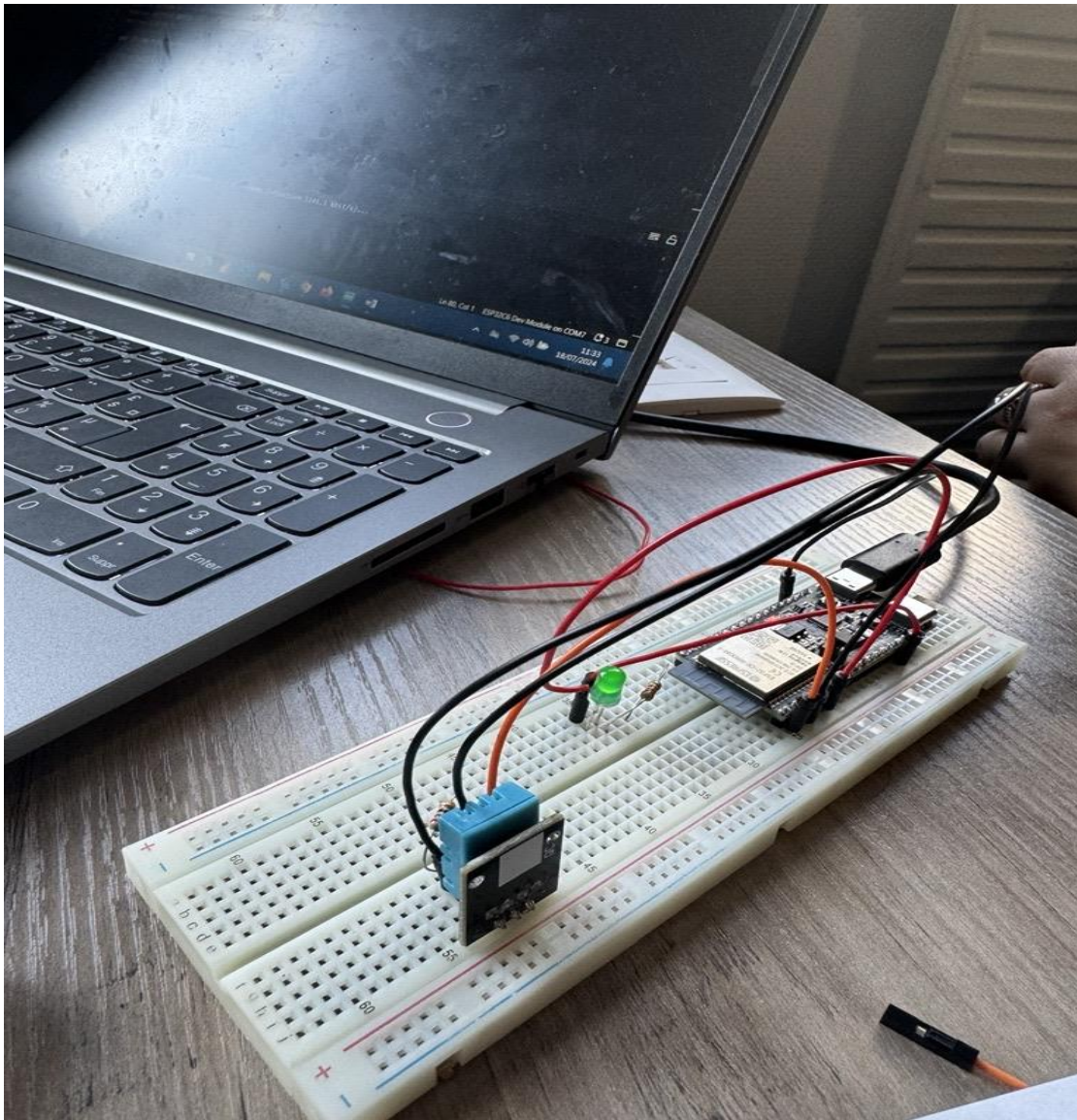
## 3. Observation :

- Si la température dépasse 25°C, la LED s'allume.

Ces étapes permettent de programmer l'ESP32 pour se connecter au Wi-Fi, lire la température, et contrôler une LED en fonction de la température mesurée.

V1

Voici une mise en pratique :



- **Créer un Compte sur Arduino Cloud IoT :**
  - On commence par créer un compte sur Arduino Cloud IoT si ce n'est pas déjà fait.
- **Ajouter un Nouvel Appareil :**
  - On ajoute un nouvel appareil (ESP32) dans le Arduino Cloud.
  - On sélectionne "Set up a third-party device" et choisit "ESP32".
  - On suit les instructions pour générer un **Device ID** et une **Device Secret Key**. Ces informations seront nécessaires pour connecter l'ESP32 au cloud.



## Device ID

7d22b87a-49ab-4432-bb0d-9a4fc8e4c3c4

## Secret Key

T!3w61mBhTpKm?QTCw#2Tz2oQ

- **Installer les Bibliothèques Nécessaires :**
  - On ouvre l'IDE Arduino et installe les bibliothèques nécessaires pour l'Arduino Cloud IoT et pour le capteur de température/humidité (par exemple, DHT).
  - Bibliothèques recommandées : ArduinoloTCloud, WiFi, DHT.
- **Écrire le Programme :**
  - On écrit un programme pour :
    - Se connecter au réseau Wi-Fi avec le SSID et le mot de passe.
    - Utiliser les **Device ID** et **Device Secret Key** pour se connecter à l'Arduino Cloud IoT sur le même réseau.

Configure network



Enter your network credentials to allow your device to connect to the Cloud.

Wi-Fi Name \*

Livebox-D7D0

Password

MRo4MEqDrMt6aaDRm5



Secret Key \*

.....



**IMPORTANT:** Remember to go to the "**Sketch**" tab and upload the sketch to load the credentials on the board.

SAVE



- Lire les valeurs de température et d'humidité à partir du capteur.
- Envoyer ces valeurs au tableau de bord Arduino Cloud IoT.
- Contrôler un relais pour l'arrosage automatique.
- **Ajouter des Variables au Tableau de Bord :**
  - Dans Arduino Cloud, on ajoute des variables pour la température, l'humidité et le relais.
  - On configure ces variables pour qu'elles soient synchronisées avec les données envoyées par l'ESP32.

Add variable
✕

Name

Temperature

↺

**Sync with other Things**
i

Temperature Sensor (°C) eg. 1 °C

↺
↻

▼

Declaration

CloudTemperatureSensor temperature ;

i

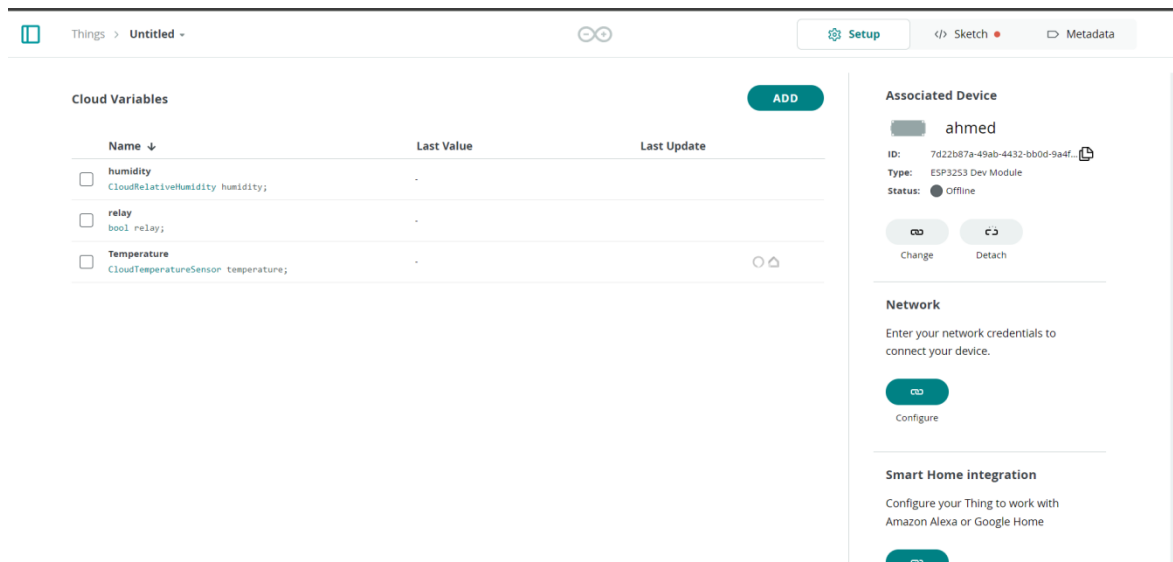
Variable Permission
i

☒
Read & Write

☐
Read Only

Variable Update Policy
i

CANCEL
ADD VARIABLE



- **Configurer le Tableau de Bord :**
  - On crée un tableau de bord (Dashboard) sur Arduino Cloud.
  - On ajoute des widgets pour afficher la température, l'humidité et contrôler le relais.
  - On configure les widgets pour qu'ils utilisent les variables correspondantes.

## Upload du Code sur Arduino Cloud IoT

Ensuite, nous avons téléversé le code sur Arduino Cloud IoT pour récupérer et surveiller les données à distance. Voici un bref résumé du processus :

1. **Préparation du Code :** Nous avons préparé le code Arduino pour se connecter à Arduino Cloud IoT, lire les données du capteur DHT11 (température et humidité), et contrôler une LED en fonction de la température.
2. **Configuration de l'Appareil :** Dans Arduino Cloud IoT, nous avons configuré notre appareil en utilisant le Device ID et la Device Secret Key, permettant ainsi une communication sécurisée avec le cloud.
3. **Téléversement du Code :** Nous avons téléversé le code sur l'ESP32 en utilisant l'IDE Arduino. Le programme lit les valeurs de température et d'humidité, et envoie ces données à Arduino Cloud IoT.

4. **Surveillance des Données** : Sur le tableau de bord Arduino Cloud, nous avons ajouté des widgets pour afficher les valeurs de température et d'humidité en temps réel, et pour contrôler la LED en fonction des seuils définis.

Ce processus nous permet de surveiller les conditions environnementales et de contrôler l'arrosage automatique à distance, optimisant ainsi la gestion de l'eau et améliorant l'efficacité du système.

En voici l'extrait du code :

```
#include <DHT.h>

// Définir les pins
#define DHTPIN 4      // Pin du capteur DHT11
#define DHTTYPE DHT11 // Type de capteur DHT
#define LEDPIN 5      // Pin de la LED

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  // Initialiser la communication série
  Serial.begin(115200);

  // Initialiser le DHT11
  dht.begin();

  // Initialiser la pin LED comme sortie
  pinMode(LEDPIN, OUTPUT);

  // Éteindre la LED au démarrage
  digitalWrite(LEDPIN, LOW);
}

void loop() {
  // Lecture de la température et de l'humidité
  float h = dht.readHumidity();
  float t = dht.readTemperature();
```

```

float t = dht.readTemperature();

// Vérifier si les lectures sont valides
if (isnan(h) || isnan(t)) {
    Serial.println("Échec de lecture du capteur DHT11 !");
    return;
}

// Afficher les valeurs lues sur le moniteur série
Serial.print("Humidité: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Température: ");
Serial.print(t);
Serial.println(" °C");

// Clignotement de la LED si la température dépasse 25 degrés Celsius
if (t > 25.0) {
    digitalWrite(LEDPIN, HIGH); // Allumer la LED

    // Attendre 500 ms (0.5 seconde)
    delay(500);

    digitalWrite(LEDPIN, LOW); // Éteindre la LED

    // Attendre à nouveau 500 ms (0.5 seconde)
    delay(500);
}

```

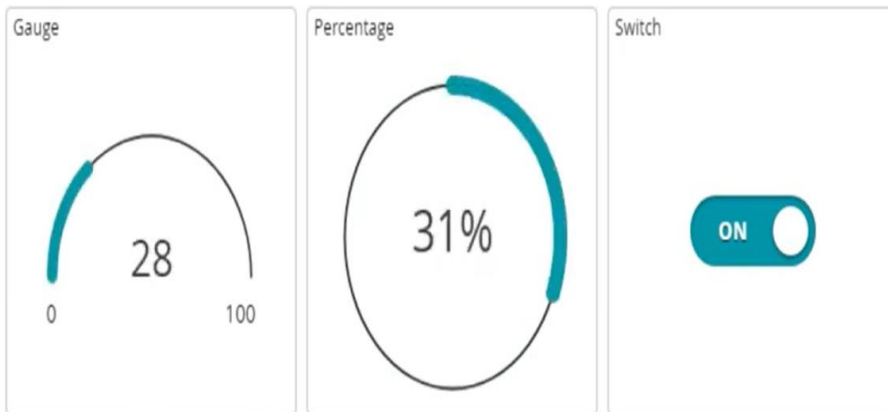
## Création d'un Dashboard avec des Données Fictives

Pour visualiser notre système, nous avons créé un dashboard sur Arduino Cloud IoT avec des données fictives. Voici les étapes que nous avons suivies :

### Configuration des Widgets :

- **Gauge (Jauge)** : Affiche la température actuelle. Dans notre exemple, elle est configurée pour montrer une valeur fictive de 28°C.
- **Percentage (Pourcentage)** : Indique le niveau d'humidité. Nous avons configuré ce widget pour afficher une valeur fictive de 31%.
- **Switch (Interrupteur)** : Permet de contrôler l'état du relais pour l'arrosage automatique, avec une position "ON" ou "OFF"

## Untitled



## Introduction à la Gestion de Projet

La gestion de projet est cruciale pour assurer la réussite du projet de capteurs de température et d'humidité. En optant pour la méthodologie Agile et l'utilisation de Trello comme outil de gestion, nous avons structuré notre travail de manière à favoriser la collaboration, la flexibilité et l'efficacité.

- **Objectif** : Livrer un système fonctionnel de mesure de la température et de l'humidité, respectant les délais et les exigences de qualité.
- **Méthodologie** : Agile, avec des itérations courtes et des révisions fréquentes pour s'adapter aux changements et aux retours.

### Organisation de l'Équipe

L'équipe est composée de six membres, chacun ayant des rôles et des responsabilités spécifiques.

Rôle	Nom	Responsabilités
Développeur Principal	Nadine	Coordination globale, gestion des parties prenantes et suivi des progrès.
Responsable Matériel	Ahmed	Développement du firmware pour l'ESP32 et intégration des capteurs.
Responsable matériel	Salah	Sélection et configuration des composants matériels (DHT11, ESP32, écran LCD).
Ingénieur Logiciel	Thinhinane	Dev des logiciels naissaicares y compris les bibliotheques et les scripts
Analyste de Données	Querdia	Collecte, analyse et visualisation des données.
Documentaliste	Sadok	Documentation du projet, y compris les rapports et les manuels d'utilisation.

### Planification du Projet avec la Méthodologie Agile

En suivant la méthodologie Agile, nous avons organisé le projet en plusieurs itérations (sprints) de deux semaines. Chaque sprint comprend la planification, l'exécution, la revue et la rétrospective.

Sprint	Dates	Tâches
Sprint 1	01/07/2024 - 07/07/2024	Planification initiale, configuration des outils, définition des rôles
Sprint 2	08/07/2024 - 14/07/2024	Développement initial, configuration matérielle, tests des capteurs et de l'écran LCD
Sprint 3	15/07/2024 - 21/07/2024	Intégration du code et des composants, tests de fonctionnalité
Sprint 4	22/07/2024 - 28/07/2024	Optimisation du code, rédaction de la documentation utilisateur
Sprint 5	29/07/2024 - 04/08/2024	Finalisation des tests, préparation de la présentation et des livrables

## Gestion de Projet et Communication

### Gestion de Projet et Communication

#### Utilisation de Trello

Trello a été utilisé pour suivre les tâches, gérer les priorités et faciliter la communication.

#### Tableau Trello

Organisé en colonnes : 'Backlog', 'À faire', 'En cours', 'En revue', 'Terminé'

#### Cartes

Tâches représentées par des cartes avec descriptions, checklists, dates et commentaires

#### Étiquettes

Catégories de tâches identifiées par des étiquettes (Développement, Matériel, Documentation, Test)

#### Assignment des Tâches

Chaque membre est assigné à des cartes spécifiques

#### Suivi et Communication

##### Réunions Hebdomadaires

Planification en début de sprint et revue en fin de sprint

##### Communication Continue

Utilisation de Slack pour la communication en temps réel et Google Drive pour le partage de documents

##### Rapports de Statut

Rapports hebdomadaires générés à partir de Trello

Voici un aperçu de la gestion des sprints et des statuts des tâches :

Sprint	Backlog	À faire	En cours	En revue	Terminé
Sprint 1	Tâches à prioriser	Configuration Trello	Assignment des rôles	Revue initiale	Configuration Trello
Sprint 2	Tests matériels	Implémentation initiale	Tests capteur DHT11	Revue code initial	Implémentation initiale
Sprint 3	Optimisation code	Intégration des composants	Tests intégration	Revue intégration	Intégration des composants
Sprint 4	Rédaction documentation	Documentation utilisateur	Tests documentations	Revue documentation	Documentation utilisateur
Sprint 5	Préparation présentation	Finalisation des tests	Révisions finales	Revue finale	Préparation présentation

## Gestion des Risques

Identification des risques et mise en place de stratégies d'atténuation pour assurer la réussite du projet.

- **Risques Techniques** : Problèmes matériels ou logiciels inattendus.
  - **Stratégies** : Tests réguliers, prototypage et validation itérative.
- **Risques de Planification** : Délais non respectés.
  - **Stratégies** : Ajustements flexibles du calendrier, réévaluation des priorités.
- **Risques de Communication** : Mauvaise communication entre les membres de l'équipe.



- **Stratégies** : Utilisation de Trello et Slack pour une communication claire et continue, réunions régulières.

Risques	Description	Stratégies
Risques Techniques	Problèmes matériels ou logiciels inattendus.	Tests réguliers, prototypage et validation itérative.
Risques de Planification	Délais non respectés.	Ajustements flexibles du calendrier, réévaluation des priorités.
Risques de Communication	Mauvaise communication entre les membres de l'équipe	Utilisation de Trello et Slack pour une communication claire et continue, réunions régulières.

## Aspect Sécurité du Projet de Capteurs de Température et d'Humidité

La sécurité est une composante essentielle de notre projet de capteurs de température et d'humidité. Elle couvre à la fois la protection des données collectées et la prévention des défaillances matérielles et logicielles.

Aspect	Objectifs	Mesures Mises en Place
<b>Sécurité des Données</b>	Confidentialité : Assurer que seules les personnes autorisées peuvent accéder aux données. Intégrité : Protéger les données contre toute modification ou altération non autorisée. Disponibilité : Assurer que les données sont accessibles aux utilisateurs autorisés quand ils en ont besoin.	Cryptage des Données : Utilisation de techniques de cryptage pour protéger les données pendant la transmission entre les capteurs et le système central. Authentification et Autorisation : Mise en place de mécanismes d'authentification pour vérifier l'identité des utilisateurs et contrôler l'accès aux données. Sauvegarde : Implémentation de procédures de sauvegarde régulières pour prévenir la perte de données en cas de défaillance du système.
<b>Sécurité Matérielle</b>	Protection Physique : Protéger les composants matériels contre les dommages physiques et les manipulations non autorisées. Surveillance et Maintenance : Mettre en place des procédures de surveillance et de maintenance régulières pour assurer le bon fonctionnement des composants matériels.	Boîtier de Protection : Utilisation de boîtiers pour protéger les capteurs et l'ESP32 contre les environnements hostiles (humidité, poussière, chocs). Câblage Sécurisé : S'assurer que les câbles sont correctement isolés et protégés contre les courts-circuits et les interférences électromagnétiques. Plan de Maintenance : Établir un plan de maintenance régulier pour vérifier l'état des composants et remplacer ceux qui sont défectueux ou usés.
<b>Sécurité Logicielle</b>	Robustesse du Code : Écrire du code robuste et résistant aux erreurs pour éviter les bugs et les vulnérabilités. Mises à Jour de Sécurité : Garder les bibliothèques et les logiciels utilisés à jour avec les derniers correctifs de sécurité. Tests et Vérifications : Effectuer des tests de sécurité réguliers pour identifier et corriger les vulnérabilités.	Revue de Code : Effectuer des revues de code régulières pour identifier les vulnérabilités potentielles et les corriger. Tests de Sécurité : Utiliser des outils de test de sécurité pour vérifier la résistance du logiciel aux attaques courantes (injections, débordements de mémoire, etc.). Environnement de Développement Sécurisé : S'assurer que l'environnement de développement est sécurisé et que seuls les membres autorisés ont accès au code source.

## Gestion des Incidents

Préparer et réagir aux incidents de sécurité pour minimiser leur impact.

Aspect	Objectifs	Mesures Mises en Place
Détection des Incidents	Mettre en place des mécanismes pour détecter rapidement les incidents de sécurité.	Système d'Alerte : Utiliser des systèmes d'alerte pour détecter et signaler les incidents de sécurité en temps réel.
Réponse aux Incidents	Établir des procédures de réponse aux incidents pour limiter les dommages et restaurer le système à son état normal.	Plan de Réponse aux Incidents : Développer un plan de réponse aux incidents détaillant les actions à entreprendre en cas d'incident de sécurité.
Enregistrement des Incidents	Documenter les incidents de sécurité pour analyser leur cause et améliorer les mesures de sécurité.	Analyse Post-Incident : Effectuer des analyses post-incident pour comprendre les causes des incidents et éviter qu'ils ne se reproduisent.

## Difficulté Rencontrée

### Problème de Compatibilité avec Arduino Cloud

Nous avons rencontré une difficulté majeure avec l'utilisation de la version gratuite d'Arduino Cloud. Cette version n'est pas compatible avec l'ESP32 de troisième génération que nous utilisons dans notre projet. En raison de cette incompatibilité, il n'a pas été possible de connecter notre ESP32 à Arduino Cloud et de récupérer des données réelles depuis les capteurs.

## Conséquences

- **Absence de Données Réelles** : L'impossibilité de connecter l'ESP32 à Arduino Cloud a empêché la collecte et l'analyse de données en temps réel provenant des capteurs de température et d'humidité.
- **Tests Limités** : Nous avons dû nous contenter de données fictives pour les tests et la validation de notre système, ce qui a limité l'évaluation de ses performances réelles.

## Solutions Envisagées

- **Passage à une Version Supérieure** : Considérer l'option de passer à une version payante d'Arduino Cloud compatible avec l'ESP32 de troisième génération.
- **Utilisation d'Autres Plateformes** : Explorer d'autres plateformes IoT compatibles avec l'ESP32 pour la collecte et l'analyse de données.
- **Prototypage Local** : Continuer les tests en local avec des solutions de prototypage alternatives pour valider le fonctionnement du système avant de passer à une solution cloud compatible.

Cette difficulté nous a permis de mieux comprendre les limitations des outils gratuits et de planifier des solutions alternatives pour la gestion de nos données IoT.

## Conclusion et Perspectives

En conclusion, notre projet d'arrosage automatique avec l'ESP32 et Arduino Cloud IoT a permis de développer une solution innovante pour la gestion efficace de l'eau. Malgré les défis rencontrés, notamment l'incompatibilité de la version gratuite d'Arduino Cloud avec l'ESP32 de troisième génération, nous avons réussi à avancer significativement en utilisant des données fictives pour tester et valider notre système.

## Perspectives

- 1. Compatibilité Cloud :** Passer à une version payante d'Arduino Cloud ou explorer d'autres plateformes IoT compatibles avec l'ESP32 pour permettre la collecte et l'analyse de données en temps réel.
- 2. Amélioration des Capteurs :** Intégrer des capteurs supplémentaires pour obtenir des données environnementales plus complètes et améliorer la précision de l'arrosage automatique.
- 3. Optimisation Logicielle :** Continuer à optimiser le code et les algorithmes de traitement des données pour augmenter l'efficacité et la fiabilité du système.
- 4. Extension des Fonctions :** Développer des fonctionnalités supplémentaires, telles que des notifications en temps réel et des contrôles à distance via une application mobile.
- 5. Déploiement à Grande Échelle :** Tester et déployer le système dans des environnements réels, tels que des jardins publics et des exploitations agricoles, pour évaluer sa performance à grande échelle et recueillir des retours d'expérience.

Nous sommes convaincus que ces perspectives permettront d'améliorer encore notre système et de répondre aux besoins de gestion de l'eau de manière plus efficace et durable.