# TUTORIEL : BAC JARDIN CONECTE ORE

1) Matériel nécessaire :

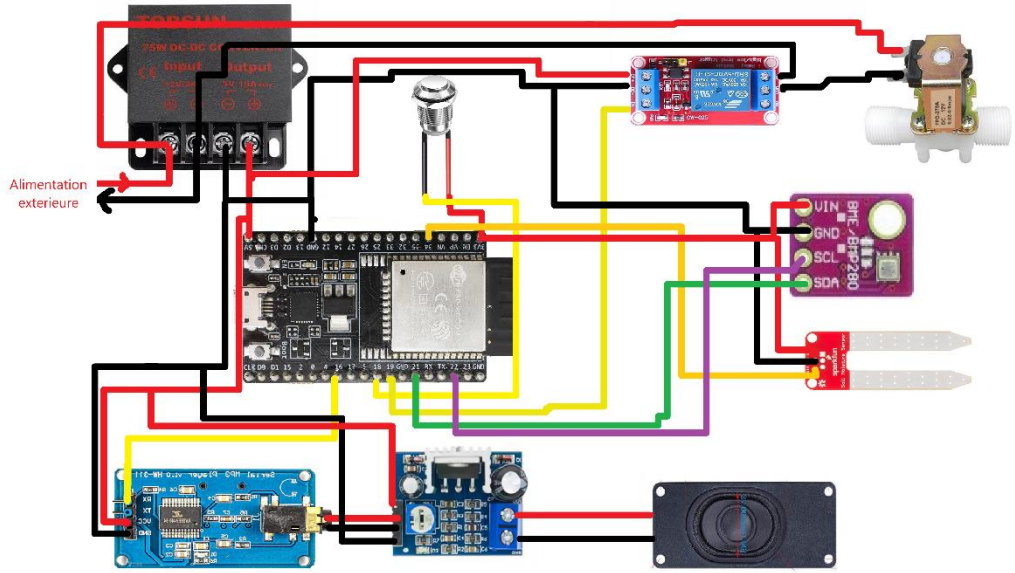
Ce projet consiste à faire des bacs permettant de contrôler l'arrosage, mesurer les données météorologiques et aussi lancer par un bouton sur les bacs une explication de ce qu'il y a dedans.

Le matériel nécessaire sera :

- Des fils de connexion.
- Une carte esp32
- Un bme280
- Un capteur d'humidité du sol (résistif)
- Un lecteur MP3 de carte SD (avec la carte SD contenant les audios voulus dedans)
- Un amplificateur.
- Un haut-parleur.
- Un bouton
- Un relais
- Une électrovanne.
- Un convertisseur de courant.

2) Montage à faire :

Le montage complet ressemble à cela

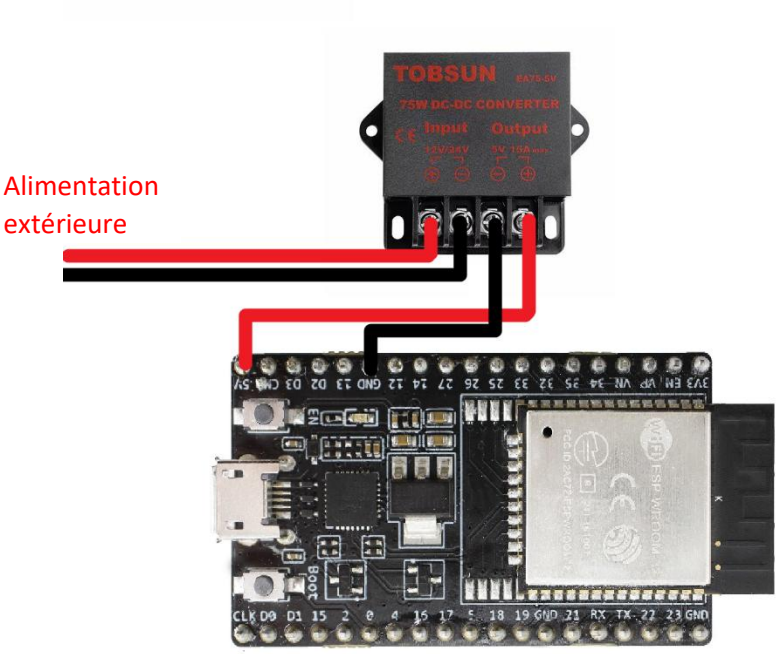



Tout comme celui de la serre du marathon créatif il est assez difficile de le rendre lisible on va donc voir chaque grande partie individuellement.

- La partie alimentation de la carte :

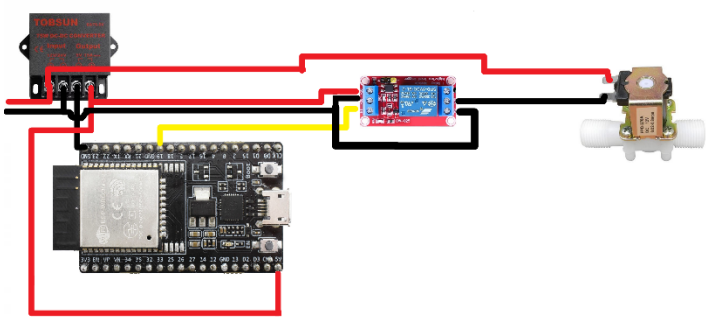
Le courant disponible dans le jardin est du 12V ,cependant une carte Arduino a besoin de 5V pour fonctionner on va donc convertir le 12V extérieur en 5V grâce à une convertisseur.

Afin d'alimenter une esp 32 avec une source extérieure il faut la connecter au prise 5V et un GND.




- La partie électrovanne :

Notre électrovanne fonctionne sur 12V, la carte ne pouvant pas gérer une telle tension on doit utiliser un relais pour cela, et on va donc relier le relais à notre carte (ou bien le 5V sortant du convertisseur ) en entrée et la sortie sera sur le 12V de l'extérieur.

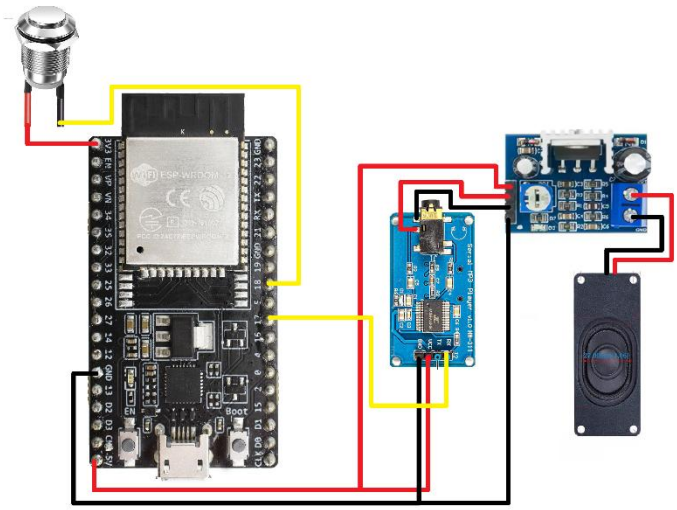


Comme on veut contrôler quand lancer l'arrosage on doit placer le – du 12V sur NO et pas NC et les entrée 5V sont sur DC+ et DC-, la connexion IN doit aller sur n'importe quelle prise numérotée de la carte ici 19.

- La partie audio :

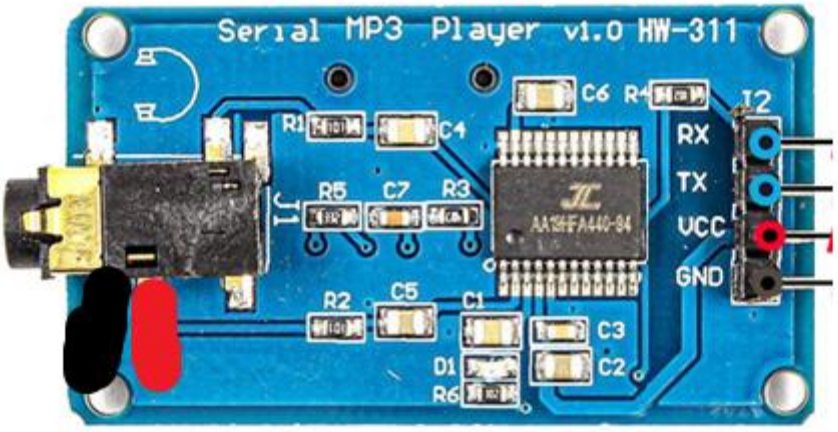
Cette partie est composé d'un bouton pour contrôler le lancement de la musique. A l'aide d'un serial MP3 player qui est un composant Arduino permettant de lire des musique(ou tout

enregistrement audio au format MP3 qu'on lui donne via une carte SD), d'un amplificateur car le son en sortie du lecteur était trop faible pour qu'on puisse entendre le son à travers la boite installer dans le jardin et d'une enceinte afin d'émettre les sons que l'on souhaite.



Le bouton se connecte au 3v3 et à un pin numéroté de notre choix (ici 18)

Le lecteur MP3 se relie à la carte par 4 fils : VCC sur 5V (ou sur le 5V en sortie de convertisseur) , GND sur la masse de la carte, RX et TX sur 2 prise définie dans le code. La carte n'ayant pas besoin d'un retour du lecteur dans notre cas seul TX de la carte (et donc RX du lecteur) sont à connecter (ici au pin 17) .
Ce qui sort du lecteur doit alors arriver dans l'ampli sur les 2 pins du centre (IN et GND) , et les 2 autres doivent être relier au 5V (VCC) et au GND de la carte (GND), pour cela on peut avoir une prise jack qui sort en fil, soit souder nous même à l'endroit suivant ( rouge sur in noir sur GND)

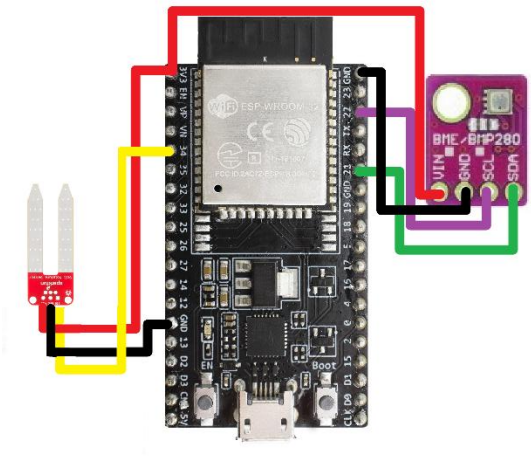



Et enfin les sorties de l'ampli partent dans l'enceinte.

- La partie capteur.

On a ici 2 capteurs utile qui sont connecté à la carte le 1er est un capteur d'humidité résistif qui est à alimenter en 3V et qui doit donner son signal sur une prise analogique (ADC sur le pinout de la

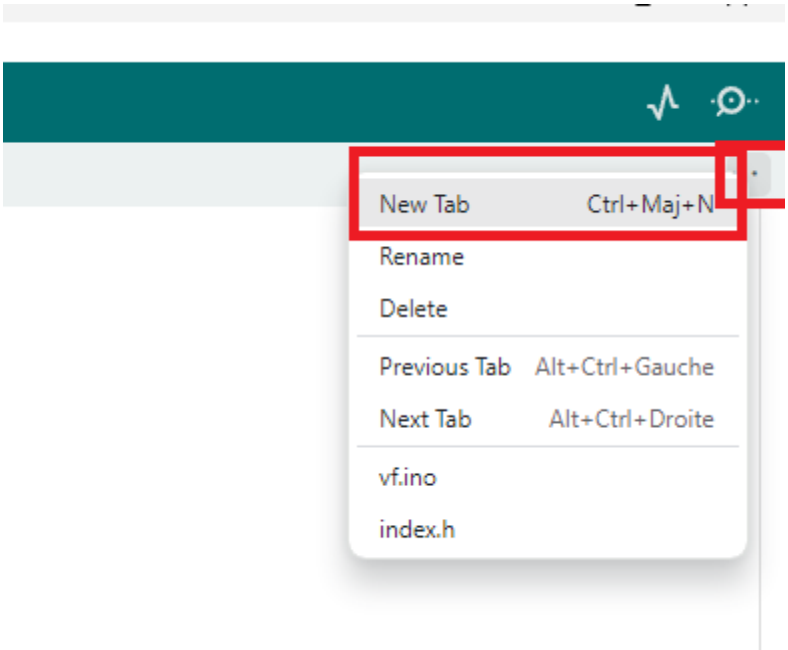
carte, ici le 34) et l'autre est un BME 280 qui demande la même alimentation et se connecte en I2C . Pour connecter un objet en I2C il faut relier ses prises SCL et SDA à celle de la carte. Le pinout de votre carte vous donnera les prise SCL et SDA par défaut (22 et 21 respectivement pour une ESP 32)




3) <u>Explication du code :</u>

Ce code est assez complexe et découper en 4 parties distincte : settings, attributes , fonction et le corps du code qui est le fichier .ino standard. Ces parties ne sont pas forcément nécessaire et on pourrait les mettre à la suite les unes des autres dans un seul fichier. Cependant elles sont mises en place par soucis de lisibilité.

On mettra les 3 autres à part le .ino dans des fichiers .h que l'on peut créer en appuyant sur le bouton suivant :



1.1)settings.h

Dans cette partie nous allons mettre en place tous les éléments pour la suite du code à commencer par les librairies et variables liées à chaque élément de notre code.

A commencer par le BME 280 qui a sa bibliothèque dédiée :

```
//Listes des librairies utile au fonctionnement des object connecté à la
carte.
#include <Adafruit_BME280.h> // Utile pour extraire les infos du BME-BMP 280.
```

et qu'on déclare comme existant avec le nom qu'il aura dans la suite du code :

```
//déclaration du BMP connecter à la carte.
Adafruit_BME280 bmp;
```

On continu avec le serial MP3 player , qui a sa bibliothèque dédiée

```
#include <SerialMP3Player.h>
```

On va ensuite définir les pins RX et TX:

```
#define RX 16// RX et TX sont les pins nécessaires pour faire fonctionner le
lecteur.
#define TX 17//
```

Ainsi que le pin du bouton permettant de déclencher le lancement du lecteur.

```
#define BtnPIN 18
```

On passe ensuite aux choses à mettre en place pour le contrôle de l'électrovanne via le capteur d'humidité à savoir :

Le pin du capteur d'humidité :

```
#define moisturePin 34
```

Le taux d'humidité critique en dessous duquel on ne doit pas passer.

```
int humidityRate=30;
```

Le pin du relais qui contrôle l'électrovanne.

```
#define relayPin 19
```

On va ensuite gérer les bibliothèques importantes pour l'API météo.

- HTTPclient nous permettra de nous connecter à l'API

```
#include <HTTPClient.h>
```

- ArduinoJSON.h nous permettra de stocker les informations tirer dans l'API afin de les traiter plus tard dans le code.

```
#include <ArduinoJson.h>
```

On va ensuite définir plusieurs strings permettant d'accéder à l'API météo :

Tout d'abord la clef qui permet de confirmer qu'on a bien accès à cet API :

```
String api_key = "48af8fee764ae56aa1395fe51afba45c"; // clefs
```

Ensuite le nom de la ville

```
String api_place="Quetigny"; // Lieux des infos voulues
```

Et le code à 2 lettres du pays :

```
String api_country="FR"; //Code à 2 lettres du pays de l'endroit voulu.
```

Avant de mettre bout à bout les morceaux dans l'ordre suivant : site de l'API, ville, pays , clef.

```
String url =
"http://api.openweathermap.org/data/2.5/forecast?q="+api_place+","+api_country
+"&APPID=" + api_key;
```

On a ensuite 10 variables définies, toute float (nombre à virgules) et alternant entre celles initialiser à 1.0 qui serviront à stockés les probabilités de pluie finale et celles initialisées à 0 qui stockeront sur un jour complet les probabilités de « non pluie » (en calculant et agrégeant les 1- probabilités de pluies)

```
float p_J1=1.0; //p sont les proba intermédiaires de "non pluies" qui seront
cumulées
float pop_J1=0.0;// pop sont les proba de pluies  calculé comme  1-proba non
pluies à la fin
float p_J2=1.0 ;
float pop_J2=0.0;
float p_J3=1.0 ;
float pop_J3=0.0;
float p_J4= 1.0;
float pop_J4=0.0;
float p_J5= 1.0;
float pop_J5=0.0;
```

On définit ensuite une variable qui stockera le temps depuis la dernière requête à l'API

```
unsigned long lastTime = 0;
```

Et le nombre de milliseconde que l'on souhaite entre 2 appels à l'API. Ce dernier ne donnant que des valeurs toute les 3h on veut le mettre à jour toute les 3 heures :

$$3 \times ((60 \times 60) \times 1000) = 10\,800\,000$$

```
unsigned long timerDelay = 10800000;
```

Enfin on initialise la chaine de caractère dans laquelle on stockera la réponse de l'API qui servira donc de buffer (mémoire tampon) :

```
String jsonBuffer;
```

On rentre alors dans la partie dédiée à la connexion IoT des bacs. Ici on utilisera le site ThingsBoard.

Les bibliothèques utiles sont :

- WiFi.h : qui permet d'initialiser le client Wi-Fi.

```
#include <WiFi.h>
```

- WiFiManager : qui permet de reconnecter à une connexion Wi-Fi à la main si les mot de passe du Wi-Fi change ou bien en cas de problème de Wi-Fi.

```
#include <WiFiManager.h>
```

- Arduino_MQTT_Client : permet à la carte se connecter à ThingsBoard via un protocole MQTT , ce qui permet au site et à la carte de parler un langage commun et donc de se comprendre lors des échanges d'informations.

```
#include <Arduino_MQTT_Client.h>
```

- Thingsboard : qui permet d'utiliser de nombre de fonction utile liée à ThingsBoard sans avoir à tout recoder soit même pour ce qui est de la connexion, envoie des attributs et télémétrie etc..

```
#include <ThingsBoard.h>
```

On a ensuite le nom du serveur ThingsBoard, ici on utilise une IP d'un serveur interne à ORE. Cependant je recommande thingsboard.cloud qui est le plus pratique car il est gratuit, accessible sur portable et on peut renouveler son compte tous les mois avec la même adresse en supprimant simplement son ancien compte.

```
constexpr char THINGSBOARD_SERVER[] = "192.168.10.29";
```

On va ensuite donner un token :

```
constexpr char TOKEN[]="B1_TEST_TOKEN";
```

Ce dernier est une suite de lettre et de chiffre qui se trouvent sur le serveur ThingsBoard dans la catégorie Entity (Entité) -> Device (Appeler dispositif si directement dans le site ou appareil ) et qui permet au site d'identifié la carte Arduino de son côté et de stocker ces données à un endroit précis.


Cela permet d'éviter toute confusion si 2 cartes ont une variable ayant le même nom, grâce au token chacune de ces variables sera stockée individuellement dans la mémoire du serveur.

Une fois cela fait il faut donner au serveur le port sur lequel on se connecte (par défaut le port 1883)

```
constexpr uint16_t THINGSBOARD_PORT = 1883U;
```

La taille maximale du message échangé entre la carte et ThingsBoard.

```
constexpr uint32_t MAX_MESSAGE_SIZE = 1024U;
```

Et le nombre maximal d'attributs qui peuvent s'échanger entre la carte et le serveur (afin que le serveur donne suffisamment de place pour les traiter dans sa mémoire).

```
constexpr size_t MAX_ATTRIBUTES = 3U;
```

Un attribut se différencie d'une télémétrie (qu'on verra plus tard) dans le fait qu'une télémétrie est un envoi à sens unique de la carte vers le serveur et ne peux pas être modifié, là ou un attribut peut être modifié d'un côté comme de l'autre en fonction du besoin.

Enfin on définit le client Wi-Fi :

```
WiFiClient wifiClient;
```

Puis le client Wi-Fi Manager qui permet de de choisir la connexion Wi-Fi manuellement depuis la carte :

```
WiFiManager WM;
```

Et pour cela on va donc définir le nom et code d'une connexion Wi-Fi que la carte générera pour nous permettre de la reconnecté à un Wi-Fi fonctionnel.

```
const char* ssid = "Bac_3";
const char* password = "CodeBac3";
```

Avec ce client Wi-Fi on définit le client MQTT

```
Arduino_MQTT_Client mqttClient(wifiClient);
```

Qui sert enfin à définir le client de Thingsboard.

```
ThingsBoardSized<Default_Fields_Amount, Default_Subscriptions_Amount,
MAX_ATTRIBUTES> tb(mqttClient, MAX_MESSAGE_SIZE);
```

Auquel on doit donner les informations définies précédemment : le nombre maximum d'attributs, et la taille maximale des messages, les autres valeurs sont laissées par défaut.

On rajoute enfin une constante par forcément nécessaire à savoir le baud rate de la carte qui nous permettra plus tard de lancer le moniteur série et donc écrire des choses dessus dans l'IDE pour d'éventuel débug qui serais nécessaire.

```
 constexpr uint32_t SERIAL_DEBUG_BAUD = 115200U;
```

### 1.2)attributes.h

On va maintenant définir le attributes.h qui sera dédier à la gestion de tous les objets utiles pour les attributs et les télémétries.

On commence par donner les noms de nos 3 attributs dans des variables constantes :

```
constexpr char T_ARROSAGE_ATTR[] = "t_arro"; //temps d'arrosage
constexpr char BTN_MANUEL_ATTR[]="btn_manuel"; //variable du bouton arrosage
manuel
constexpr char BTN_AUTO_ATTR[]= "btn_auto";//variable du bouton type
d'arrosage
```

On donne ensuite les valeurs par défaut de ces attributs

```
volatile bool btn_manuel =false;//Le mode manuel l'arrosage ne se lance pas
avant l'input utilisateur.
volatile bool btn_auto= false;// Ici on souhaite que le mode auto soit mis par
défaut
volatile uint16_t t_arro=1000U; //temps par défaut d'arrosage de 1ssec,
```

On définit ensuite 2 variables qui serviront de bornes à l'attributs temps d'arrosage.

```
constexpr uint16_t T_ARROSAGE_MS_MIN = 10U;
constexpr uint16_t T_ARROSAGE_MS_MAX = 60000U;
```

Puis 2 variables qui stockeront l'information du changement des attributs, tout d'abord un booléen qui servira à savoir s'il faut mettre à jour les booléens sur cette boucle.

```
volatile bool attributesChanged = true;
```

Et le temps depuis le dernier changement.

```
uint32_t previousStateChange;
```

Et 2 variables servant pour la télémétrie et définissant respectivement l'intervalle souhaitez entre les envois et le temps depuis le dernier envoi.

```
constexpr int16_t telemetrySendInterval = 2000U; //Temps entre les envoie de
télémétrie
uint32_t previousDataSend;
```

Ont défini ensuite une liste de chaine de caractère qui sera la liste des noms de nos attributs. Il est important que le nombre devant le U dans la ligne soit égal au nombre d'attributs qu'on ajoute et inférieur ou égal au nombre défini dans les settings.

```
constexpr std::array<const char *, 3U> SHARED_ATTRIBUTES_LIST = {
  T_ARROSAGE_ATTR,
  BTN_MANUEL_ATTR,
  BTN_AUTO_ATTR
};
```

Ensuite ont défini la fonction processSharedAttributes qui permet de traiter les valeurs des attributs après changement sur ThingsBoard.

```
void processSharedAttributes(const JsonObjectConst &data) {
```

Cette ligne permet de passer par toutes les valeurs qui revienne du serveur ThingsBoard.

```
for (auto it = data.begin(); it != data.end(); ++it) {
```

La 1ère condition permet grâce à strcmp de comparer les chaines de caractère qui sort de ThingsBoard au nom des attributs.

```
 if (strcmp(it->key().c_str(), BTN_MANUEL_ATTR) == 0) {
```

Si les 2 sont identique (donc la fonction retourne 0) alors on sait qu'il faut entrer dans la boucle et changer la valeur de notre variable liée à l'attributs pour la carte.

```
btn_manuel = it->value().as<bool>();
    }
```

La seconde condition fait exactement la même chose mais pour l'autre attribut contrôlants les modes d'arrosage :

```
 else if (strcmp(it->key().c_str(), BTN_AUTO_ATTR) == 0) {
      btn_auto = it->value().as<bool>();
    }
  }
```

On continue avec notre 3ème attributes qui est avec le même début à 2 différences près : car on le me dans une nouvelle variable qu'on défini à l'instant et pas directement dans la variable utile à la carte, et on le défini comme un entier (int) , non signé (u) et encodé sur 16 bits (16) et pas un booléen.

```
else if (strcmp(it->key().c_str(), T_ARROSAGE_ATTR) == 0) {

      const uint16_t new_interval = it->value().as<uint16_t>();
```

Ces changements sont nécessaires car on rajoute une autre condition, if par la suite. Afin de vérifier qu'il est bien dans l'intervalle que l'on souhaite avant de stocker ce nouvel intervalle dans la variable de notre carte.

```
if (new_interval >= T_ARROSAGE_MS_MIN && new_interval <= T_ARROSAGE_MS_MAX) {
        t_arro = new_interval;
}
```

Enfin on passe le booléen qui stocke l'information du changement des attribut vers « vrai » :

```
  attributesChanged = true;
}
```

On arrive aux callbacks qui sont 2 lignes longues mais bien distincte , la 1ère est la « Shared_Attributes Callbacks », elle permet de traiter les attributs partagés qui aurais été reçu depuis ThingsBoard sans demande préalable et on lui donne donc la fonction à utiliser et 2 itérateur qui donne le début et la fin de la liste des attributs.

```
const Shared_Attribute_Callback<MAX_ATTRIBUTES>
attributes_callback(&processSharedAttributes, SHARED_ATTRIBUTES_LIST.cbegin(),
SHARED_ATTRIBUTES_LIST.cend());
```

La 2nde est la « Attributes Request Callbacks », elle permet de traiter les attributs partager qui aurais été reçu depuis ThingsBoard avec une demande de la carte et on lui donne donc la fonction à utiliser et 2 itérateur qui donne le début et la fin de la liste des attributs.

```
const Attribute_Request_Callback<MAX_ATTRIBUTES>
attribute_shared_request_callback(&processSharedAttributes,
SHARED_ATTRIBUTES_LIST.cbegin(), SHARED_ATTRIBUTES_LIST.cend());
```

On va ensuite définir des fonctions tout d'abord :

```
void processSetBtnManuel(const JsonVariantConst &data, JsonDocument &response)
{
```

Qui permet de traiter le changement d'un attribut par Thingsboard. Ce qui correspond à tous les JSON en haut, l'un (data) qui correspond à la donnée qui sort de Thingsboard et l'autre (response) à ce qui ressort de la carte vers ThingsBoard avec la valeur nouvelle.

On commence par mettre la valeur sortant de ThingsBoard dans une variable.

```
  int new_mode = data;
```

Et créer le document JSON qui servira à stocker notre réponse :

```
  StaticJsonDocument<1> response_doc;
```

On vérifie ensuite avec une condition que l'on a bien un booléen qui ne peut prendre que la valeur 0 (false) ou 1 (true):

```
if (new_mode != 0 && new_mode != 1) {
```

Si ce n'est pas le cas on envoie un message d'erreur qu'on stocke dans notre réponse à :

```
 response_doc["error"] = "Mode Inconnu!";
 response.set(response_doc);
```

Et on ressort directement de la fonction.

```
    return; // si erreur il y a on sors imédiatement de la fonction
  }
```

Sinon on stocke le nouveau mode dans la variable de l'attribut du côté de la carte :

```
  btn_manuel = new_mode;
```

On dit que l'attributs a été changer en changeant le booléen.

```
  attributesChanged = true;
```

Et on remet la nouvelle variable dans notre document de réponse :

```
  response_doc["newMode"] = (int)btn_manuel;
  response.set(response_doc);
}
```

Et ensuite on construit la fonction :

```
void processSetBtnAuto(const JsonVariantConst &data, JsonDocument &response) {
```

Qui est construit exactement de la même manière au nom des variables prêt vu qu'on a simplement changer le nom de la variable.

```
  Serial.println("Reception etat du bouton mode auto via methode RPC");
  int new_mode = data;
  StaticJsonDocument<1> response_doc;
```

```
  if (new_mode != 0 && new_mode != 1) { /// Si le mode n'est pas un entier
soit 0 ou 1 (booléen true/false) alors on a une erreur
    response_doc["error"] = "Mode Inconnu!";
    response.set(response_doc);
    return; // si erreur il y a on sort imédiatement de la fonction
  }
  //Si pas d'erreur on met à jour l'état du bouton de l'état manuel.
  btn_auto = new_mode
  //On stocke que l'on a bien changer un attribut.
  attributesChanged = true;
  // On renvoi l'état actuel du bouton.
  response_doc["newMode"] = (int)btn_auto;
  response.set(response_doc);
}
```

Ces 2 fonctions servent à savoir comment traiter les nouvelles valeurs de nos boutons qui arrivent et doivent être envoyer du côté de ThingsBoard aussi , on les appelle alors les RPC (Remote Procedure Calls). Et on doit fournir à ThingsBoard la liste de la même manière que pour les attributs avec une liste :

```
const std::array<RPC_Callback, 2U> callbacks= {
  RPC_Callback{"setBtnAuto",processSetBtnAuto},
  RPC_Callback{"setBtnManuel",processSetBtnManuel}
  };
```

Les 2 différences notables sont que le nombre devant U n'est plus limité par une constante qu'on définit avant et qu'on doit donner le nom de la fonction coté TB , entre guillemets, et la fonction en elle-même après.

1.3)fonction.h

Cette page sera exclusivement dédiée à la construction des fonctions qui serviront dans la suite du code et qui rendrais le code peut lisible si mises dans le .ino, et ont donc été regroupé sur leur propre partie du code.

1.3)1. InitWiFi() :

Cette fonction sert à lancer le Wi-Fi grâce à WiFiManager.

```
void InitWiFi() {
```

On comment par passer le mode de l'antenne Wi-Fi en émission

```
 WiFi.mode(WIFI_STA);
```

Ensuite on met une condition « tant que » sur l'état de connexion via les protocoles de WiFiManager ( d'abord une connexion directe via le nom et mot de passe en mémoire sur la carte, sinon passage en mode émission et demande d'un code valable à l'utilisateur)

```
 while(!WM.autoConnect(ssid, password))
{
```

On ne sort de la boucle que lorsque la carte est effectivement connectée au Wi-Fi.

```
  Serial.print(".");
 }
  Serial.print("Connecte");
}
```

### 1.3)2. reconnect() :

Cette fonction permet tout simplement de reconnecter la carte au wifi si elle n'est pas ou plus connectée au Wi-Fi

```
const bool reconnect() {
```

On commence par mettre dans une variable le statut du wifi

```
const wl_status_t status = WiFi.status();
```

On met ensuite une condition sur le statut. S'il est connecté alors on sort instantanément de la fonction.

```
if (status == WL_CONNECTED) {// si le status est connecté
    return true; //return faire sortir de la fonction sans traiter les choses
après le IF
  }
```

Si ce n'est pas le cas alors on lance la fonction de connexion au Wi-Fi.

```
InitWiFi();
  return true;
}
```

### 1.3)3. TBLancement() :

Cette fonction permet de lancer toutes les choses utiles pour TB et aussi de mettre les messages d'erreur associé à chaque fois.

```
void TBLancement (){
```

Tout d'abord le lancement de la fonction est conditionné au fait qu'on ne soit pas déjà connecté.

```
  if (!tb.connected()) {
```

Si ce n'est pas le cas ( d'où le « ! » qui indique que la condition doit être fausse) alors on teste le fait de se connecter avec au serveur avec le token et part le port donné. Si cela échoue alors on écrit un message d'erreur sur le moniteur série.

```
if (!tb.connect(THINGSBOARD_SERVER, TOKEN, THINGSBOARD_PORT)) {
  Serial.println("Echec de la connexion");
      return;
    }
```

Puis on  tente de s'enregistrer pour les RPC, et de la même manière si cela échoue on envoie un message d'erreur.

```
 if (!tb.RPC_Subscribe(callbacks.cbegin(), callbacks.cend())) {// Si il y a un
echec pour les fonction RPCs.
      Serial.println("Failed to subscribe for RPC");
      return;
    }
```

On fait la même chose pour les attributs partagés.

```
  if (!tb.Shared_Attributes_Subscribe(attributes_callback)) { // Si il y a un
échec pour les attributs partagés.
      Serial.println("Failed to subscribe for shared attribute updates");
      return;
    }
```

Et pour les requêtes au serveur.

```
   if (!tb.Shared_Attributes_Request(attribute_shared_request_callback)) {
      Serial.println("Failed to request for shared attributes");
      return;
    }
```

### 1.3)4. httpGETRequest() :

Cette fonction sert à extraire les données d'un API.  Ici les données météo sur les 5 prochains jours. Il ressort une chaine de caractère.  D'où le fait qu'il soit défini comme une « string » :

```
String httpGETRequest(const char* serverName) {
```

On lance d'abord les client Wi-Fi pour se connecter à internet et HTTP pour aller sur les sites internet.

```
 WiFiClient client;
  HTTPClient http;
```

On lance d'abords la connexion au site en donnant notre client Wi-Fi et l'adresse du site que l'on souhaite.

```
  http.begin(client, serverName);
```

On stocke dans un entier la réponse du site à la fonction GET (qui demande des informations à ce site)

```
  int httpResponseCode = http.GET();
```

Et on initialise la chaine de caractère qui stockera notre réponse pour la suite.

```
  String payload = "{}";
```

Si notre réponse est non nulle :

```
  if (httpResponseCode>0) {
```

Alors on stocke le retour du site sous forme de chaine de caractère.

```
    Serial.print("HTTP Response code: ");// on print la réponse dans le serial
    Serial.println(httpResponseCode);
    payload = http.getString();// On la stocke dans notre payload
  }
```

Sinon on envoi un message d'erreur.

```
  else {// sinon on envoie un message d'erreur.
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
```

Et on termine la connexion http.

```
  http.end();
```

On donne la chaine de caractère de l'API en sortie de la fonction.

```
  return payload;
}
```

### 1.3)5. getInfoAPI() :

Cette fonction permet de traiter les informations que l'on souhaite avoir parmi toute celle envoyées par l'API du site internet.

```
 void getInfoAPI(){
```

Tout d'abord on vérifie si le délai choisi en chaque appel à l'API a été dépassé.

```
if ((millis() - lastTime) > timerDelay) {
```

Ensuite on vérifie que le Wi-Fi est bien connecté et si c'est le cas :

```
if(WiFi.status()== WL_CONNECTED){
```

Alors on prend ce qui sort de l'API avec la fonction précédente et on le stocke dans la variable buffer qu'on a défini plus haut.

```
jsonBuffer = httpGETRequest(url.c_str());
```

Ensuite on définit une variable de type document json, et on utilise une fonction de la bibliothèques JSON pour remettre notre chaine de caractère dans un bon format pour être utilisé par la suite.

```
JsonDocument myObject;// On  défini un objet json qui sera nos infos.
deserializeJson(myObject,jsonBuffer);
```

On enchaine par la suite  5 boucles ayant toute la même forme , je ne vais donc en détaillée qu'une seule.  Tout d'abord on définit la probabilité de non pluie à 1 par défaut.

```
p_J1=1.0;
```

Ensuite on boucle sur les valeurs qui servent au 24H prochaines heures, comme l'API donné une valeur toute les 3h, on prend les valeurs de 0 à7  ( jour 2 serais de 8 à 15,  jour 3 de 16 à 23 et ainsi de suite)

```
for (int i=0;i<8;i++){
```

On définit une variable temporaire appeler buff qui servira à stocker temporairement la valeur de probabilité de pluie pour la valeur donnée.  En allant dans le document JSON avec les bonnes catégories.

```
float buff=float(myObject["list"][i]["pop"]);
```

On redéfinit ensuite la probabilité de non-pluie comme sa valeur précédente multiplié par 1- la probabilité de pluie en cours. On obtient donc en sortie de boucle la probabilité de non-pluie sur les 24H concernée par la boucle.

```
p_J1=p_J1*(1-buff);
```

Enfin on définit la probabilité de pluie sur la journée comme 1 moins la probabilité de  non-pluie multipliée par 100.

```
pop_J1= (1-p_J1)*100;
```

Si le Wi-Fi n'est pas connecté, alors on renvoi un message d'erreur via le moniteur série.

```
else {
      Serial.println("WiFi Disconnected"); // si le wif est pas connecté on
envoie un msg d'erreur.
```

```
    }
```

Enfin ont redéfini la constante lastTime (temps de la dernière demande à l'API)  à l'instant actuel grâce à la

```
lastTime = millis();
```

### 1.3)6. ResetWiFi() :

Cette fonction est surtout utile pour le débug mais a été garder dans le code final car elle peut toujours s'il y a besoin de  changer de connexion Wi-Fi mais que celle précédemment en mémoire n'a pas disparue.

```
void ResetWiFi(){
```

On vérifie simplement que la valeur analogique sur une prise de notre choix, et non utilisé dans le code (ici 35) ne dépasse pas une certaine valeur.

```
 if (analogRead(35)>1000){
```

Si c'est le cas alors on remet à 0 les infos de Wi-Fi Manager

```
WM.resetSettings();
```

Et on relance la carte :

```
 ESP.restart();
  }
}
```

### 1.4)Bac_jardin.ino

On passe maintenant au cœur du code qui doit commencer par inclure les parties précédentes afin que ces dernières ne soit pas faites pour rien :

```
#include "settings.h"
#include "attributes.h"
#include "fonctions.h"
```

On met ensuite en place le  void setup :

```
void setup() {
```

Tout d'abord en lançant le moniteur série qui va permettre de débug un grand nombre de chose une fois lancé.

```
  Serial.begin(SERIAL_DEBUG_BAUD);
```

On va aussi lancer notre capteur BME280 :

```
  bmp.begin(0x76);
```

Puis définir les modes de connexion de nos objets ( entrée pour le capteur d'humidité du sol et le bouton, sortie pour le relais)

```
  pinMode(moisturePin,INPUT);
  pinMode(relayPin,OUTPUT);
  pinMode(BtnPIN,INPUT);
```

On lance ensuite le lecteur MP3

```
  mp3.begin(SERIAL_DEBUG_BAUD);
```

Et on lui demande de prendre une compte notre carte SD.

```
mp3.sendCommand(CMD_SEL_DEV, 0, 2);
```

On termine enfin en utilisant notre fonction lançant la connexion Wi-Fi (voir partie fonction) :

```
InitWiFi();
```

On fait ensuite le void loop qui sera commencer par le lancement de la fonction pour lancer ThingsBoard (voir partie fonction) :

```
void loop() {
TBLancement ();
```

Puise celle pour voir s'il faut reset les paramètres Wi-Fi :

```
ResetWiFi();
```

Puis celle qui permet d'extraire les informations utiles de l'API

```
getInfoAPI();
```

Et enfin on va faire une condition sur le fait que les attributs aient changés.

```
 if (attributesChanged) {
```

Si c'est le cas on remet la variable en false afin de détecter le prochain changement,

```
attributesChanged = false;
```

Et on envoi leurs nouvelles valeurs à ThingsBoard

```
tb.sendAttributeData(T_ARROSAGE_ATTR, t_arro);
tb.sendAttributeData(BTN_MANUEL_ATTR,btn_manuel);
tb.sendAttributeData(BTN_AUTO_ATTR,btn_auto);
```

Et leurs valeurs en télémétries.

```
tb.sendTelemetryData(T_ARROSAGE_ATTR, t_arro);
tb.sendTelemetryData(BTN_MANUEL_ATTR,btn_manuel);
tb.sendTelemetryData(BTN_AUTO_ATTR,btn_auto);
```

On définit ensuite la valeur de l'humidité du sol à l'instant T qui permet de mettre en place les différents modes d'arrosage.

```
int moisture=map(analogRead(moisturePin),0,2830,0,100);
```

Si la variable du mode automatique est vraie

```
 if (btn_auto==true){
```

Et que notre humidité est en dessous du seuil qu'on veut fixer :

```
 if (moisture>=humidityRate){
```

Alors on allume l'électrovanne pendant un temps donné par l'attributs t_arro.

```
digitalWrite(relayPin,HIGH);
delay(t_arro);
digitalWrite(relayPin,LOW);
```

Et on remet l'attributs du bouton manuel à false, en précisant qu'on a changé la valeur d'un attribut, car cela permet de ne pas avoir de « mémoire » des appuis sur le bouton arrosage manuel lors du mode automatique et que ce dernier ne se lance trop de fois lors du changement de mode fait.

```
btn_manuel=false;
attributesChanged = true;
```

Si l'humidité mesurée n'est pas supérieure :

```
 else {
```

Alors on éteint notre électrovanne :

```
digitalWrite(relayPin,LOW);
```

Et on refait la manœuvre pour le bouton manuel :

```
btn_manuel=false;
attributesChanged = true;
}
```

Et si le bouton du mode automatique est sur false :

```
  else { // si le mode auto est sur false.
```

Alors on doit tester l'état du bouton manuel :

```
  if (btn_manuel==true){ // et que le btn d'arosage manuel est appuyé
```

Et s'il est actif alors on arrosage pendant un temps donné t_arro

```
digitalWrite(relayPin,HIGH); // on lance l'arrosage pour une durrée t_arro
delay(t_arro);
digitalWrite(relayPin,LOW);
```

Et on conclut avec la manœuvre de retour du bouton manuel à false.

```
btn_manuel=false;
attributesChanged = true;
  }
```

Et si le bouton manuel est sur false alors on éteint la pompe.

```
else {// si le btn manuel  n'est pas  activé alors on éteint la pompe.

    digitalWrite(relayPin,LOW);
  }
 }
```

Ensuite on teste l'état du bouton et si ce dernier est appuyé  (donc le courant qui passe est non nul)

```
  if (digitalRead(BtnPIN)>0){
```

Alors on lance la musique qui est sur la carte SD.

```
    mp3.play(1);
  }
```

Enfin on prend toutes les mesures utiles venant du BME280

```
  float temp=bmp.readTemperature();
  float alt=bmp.readAltitude(1000);
  float press=bmp.readPressure()/100;
  float humi_air=bmp.readHumidity();
```

A noté : pour l'altitude il faut donner une pression de référence afin de calculer l'altitude avec la différence, et on divise la pression par 100 car celle donné est en Pascal alors que de façon standard on va avoir du hPa

On a ensuite une condition sur le temps depuis le dernier envoi de télémétrie

```
  if (millis() - previousDataSend > telemetrySendInterval) {
```

Si on est bien au-dessus de l'écart souhaiter on va redéfinir le moment de la dernière télémétrie :

```
previousDataSend = millis();
```

Puis on envoi de toutes les valeurs de télémétrie de nos capteurs.

```
tb.sendTelemetryData("temperature",temp);
tb.sendTelemetryData("pression",press);
tb.sendTelemetryData("altitude",alt);
tb.sendTelemetryData("humidite_air",humi_air);
tb.sendTelemetryData("humidite",moisture);
```

Ainsi que les valeurs sortantes de l'API

```
tb.sendTelemetryData("pluie_jour_1",pop_J1);
tb.sendTelemetryData("pluie_jour_2",pop_J2);
tb.sendTelemetryData("pluie_jour_3",pop_J3);
tb.sendTelemetryData("pluie_jour_4",pop_J4);
tb.sendTelemetryData("pluie_jour_5",pop_J5);
```

Un délai afin que les boucle ne soit pas trop rapprocher que TB ne bloque pas car trop de requête :

```
delay(1000);
```

Et on utilise la fonction loop (boucle) intégrer à la librairies ThingsBoard pour que tout se passe plutôt bien coté TB.

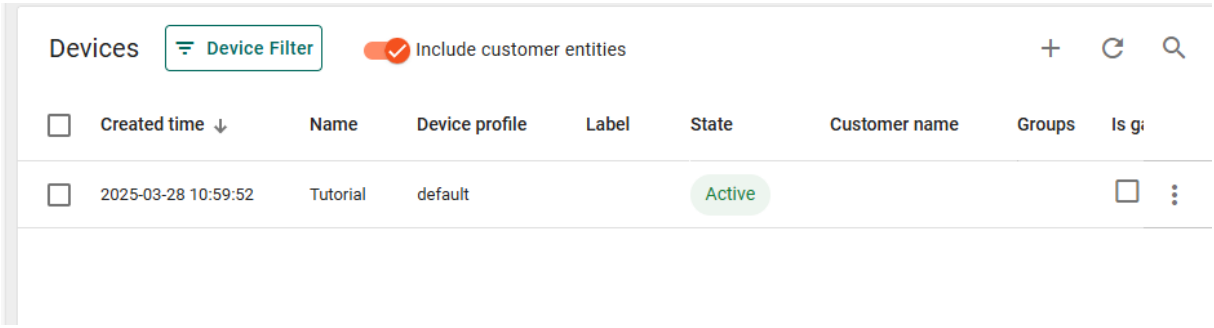
```
tb.loop();
```

```
}
```

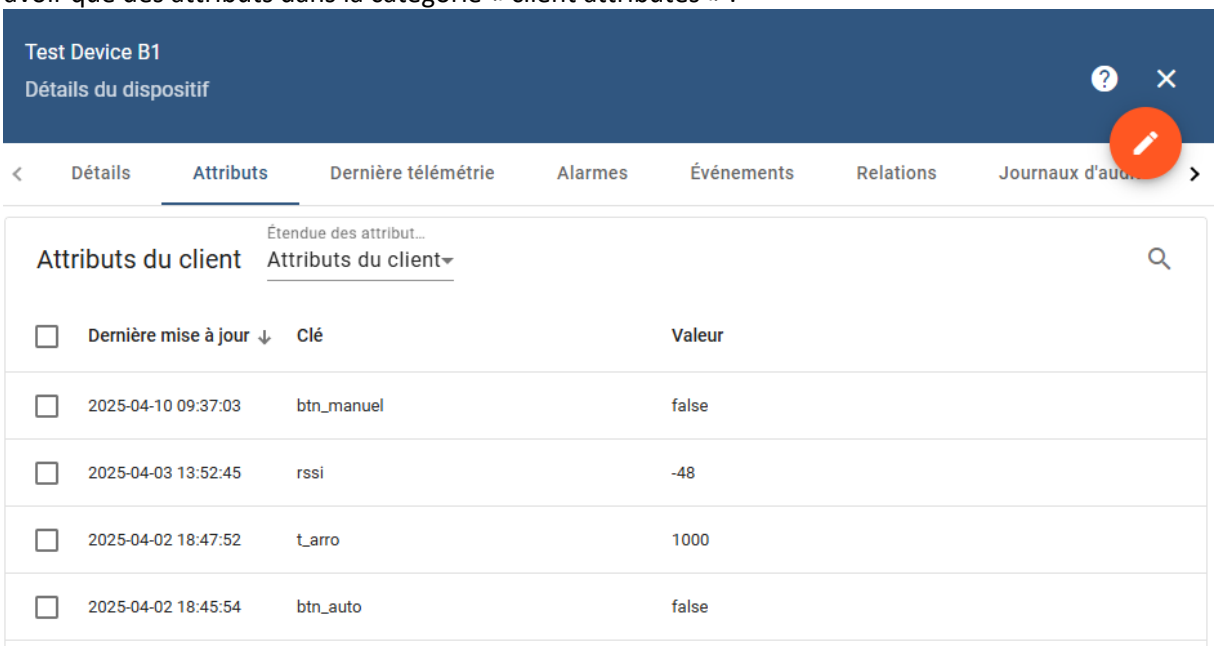
4) Mises en place du ThingsBoard.

Une fois le code fait et envoyer (voir code complet plus bas pour le faire). On va devoir mettre en place le fonctionnement coté ThingsBoard.

Une fois votre device/dispositif (je vais les appeler device dans la suite pas soucis de clarté) créer sur ThingsBoard et le code envoyer sur votre carte avec votre token vous devriez avoir votre device qui est affiché active (inactive par défaut) .

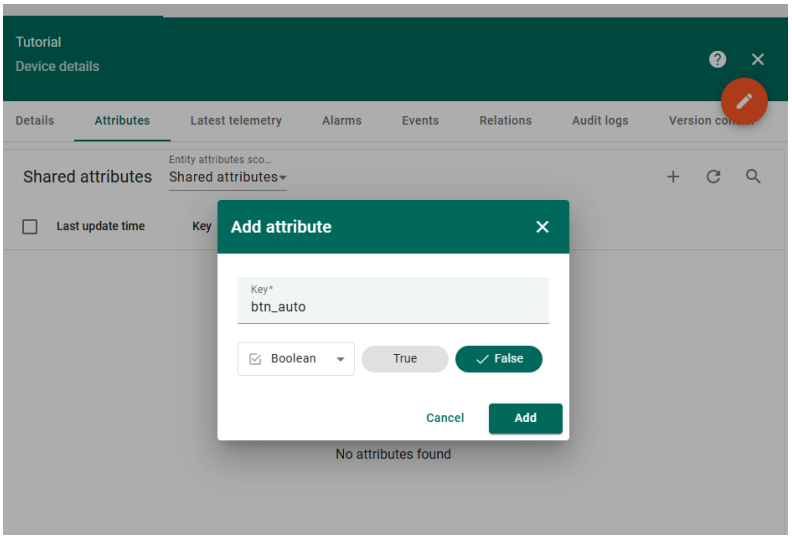



En cliquant dessus une fenêtre devrait s'ouvrir et aller dans la catégorie attributes, il ne devrait y avoir que des attributs dans la catégorie « client attributes » :

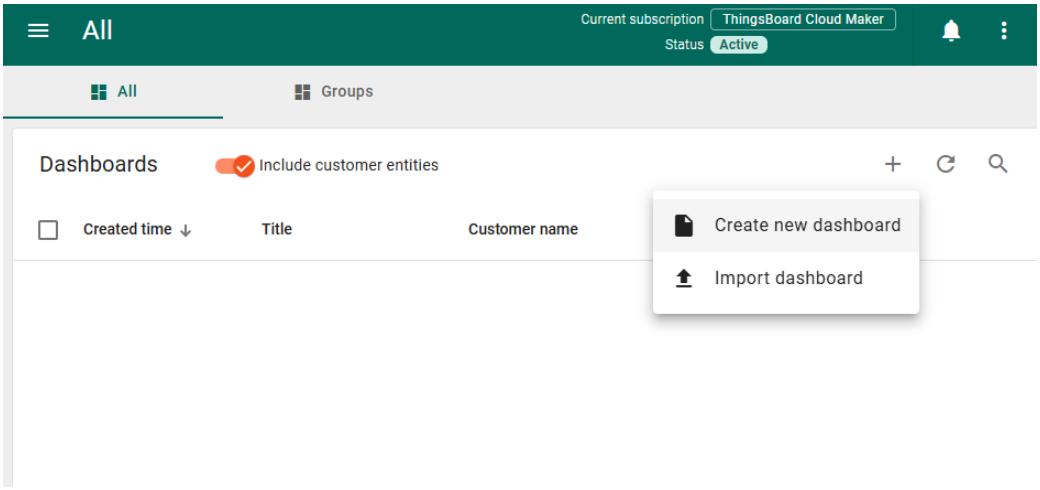



On va donc devoir ajouter les attributs partagés à la main en allant dans la catégorie shared attributes (menu défilant sur la case client attributes)

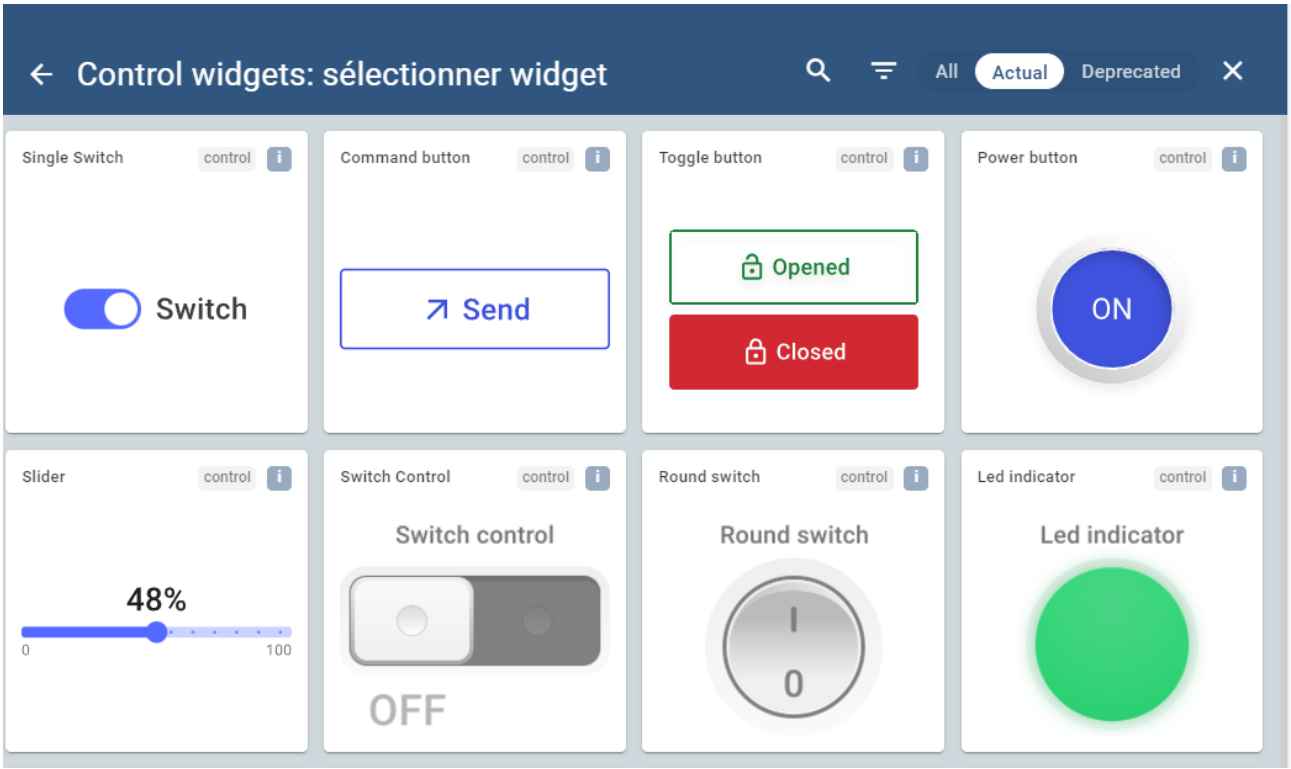
Puis un cliquant sur le « + » , mettant le type de variable sur booléen et en rentrant le nom des attributs que l'on souhaite avoir en partagés (ou integer pour entier dans le cas de t_arro et donné sa valeur par défaut)



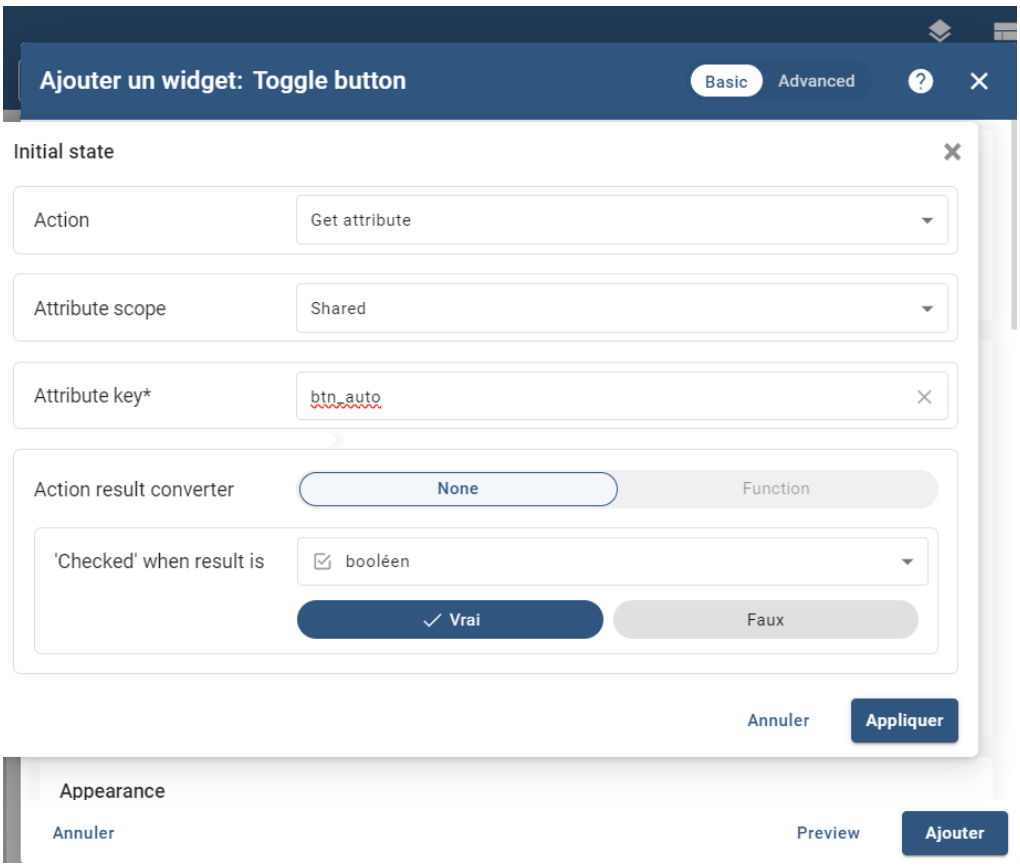
Un fois cela fait on peut passer dans la partie Dashboard. Pour ça il faut aller dans la partie dashboard/Tableaux de bord (je vais juste utiliser dashboard pour être clair)  et on va en créer un



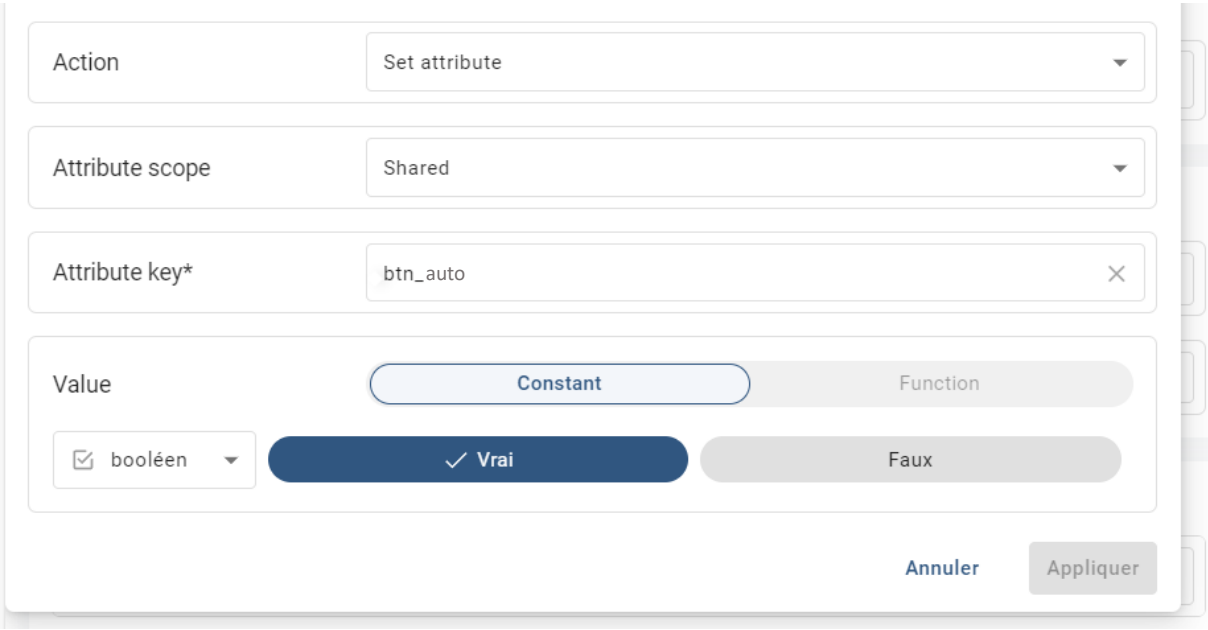

Donné lui le nom que vous souhaitez, ne vous souciez pas de la fenêtre qui apparait une fois créer et vous pouvez simplement la fermée. Vous devriez arriver sur un écran qui a  « add new widget » Cliquer dessus et aller dans la catégorie control widgets.



Choisissez « toggle button », choisissez le bon device dans la case en haut et mettez les paramètres suivants, qui permette de mettre le bouton par défaut à l'état de votre carte :

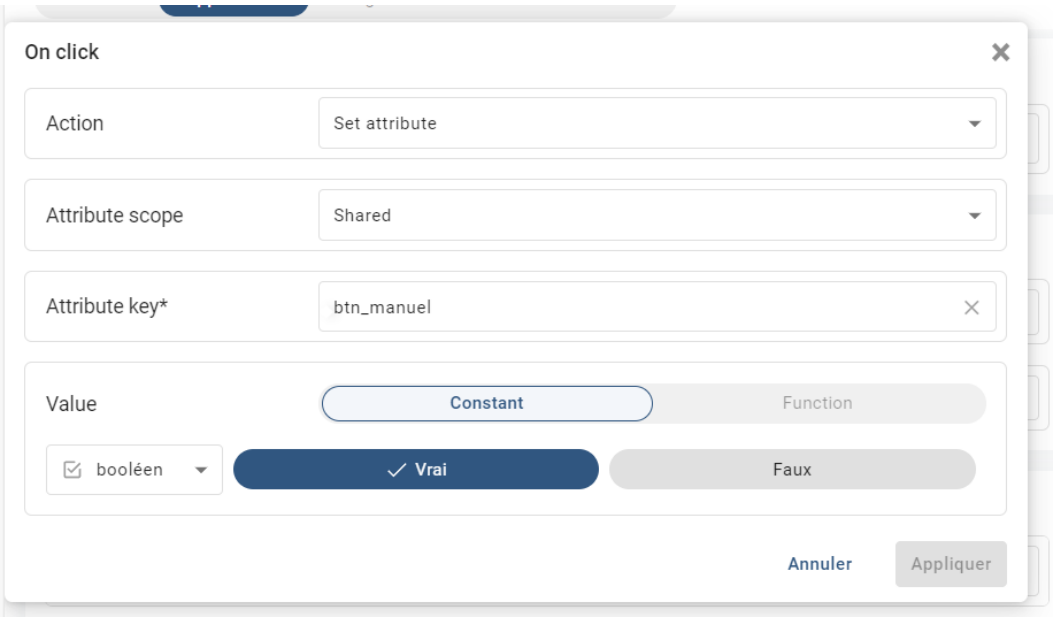



Et pour les options check / uncheck mettez les paramètres suivants en prenant soins de changer la « value » en bas pour différencier les 2 états.

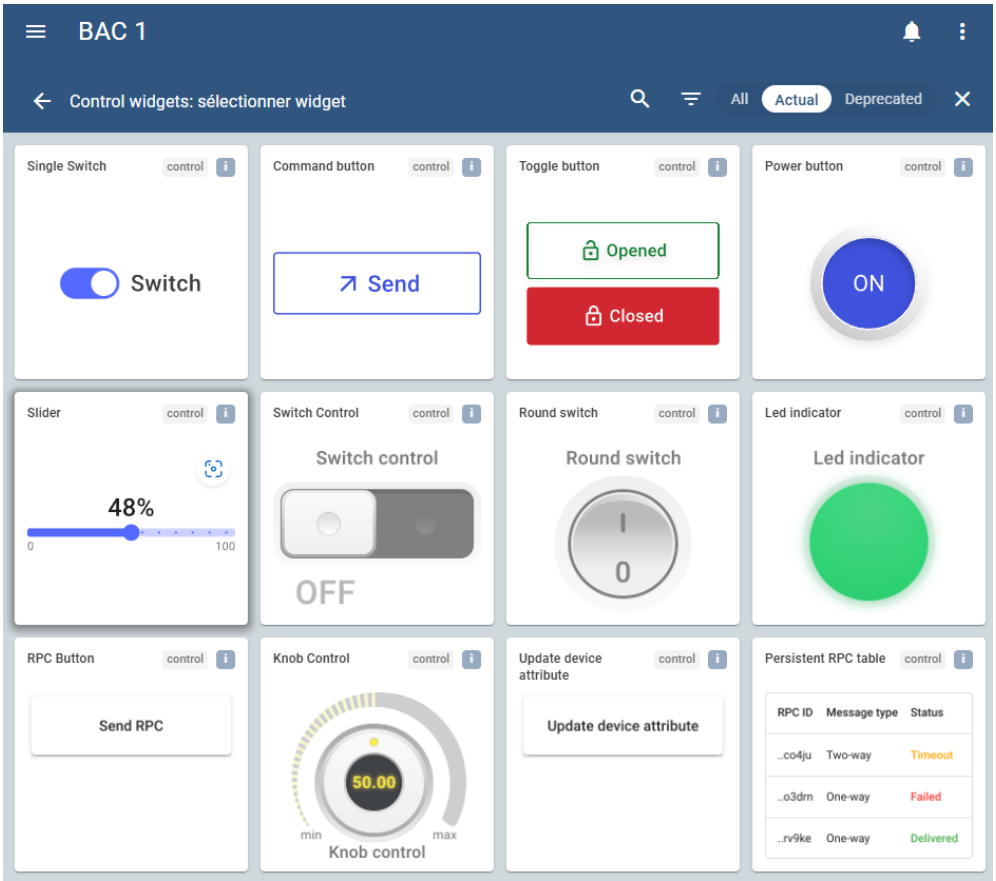



Pour ce qui concerne le bouton d'arrosage manuel on va rajouter un autre widget (bouton «+ » en haut à gauche) et choisir cette fois ci le « command button », les informations à rentrer son

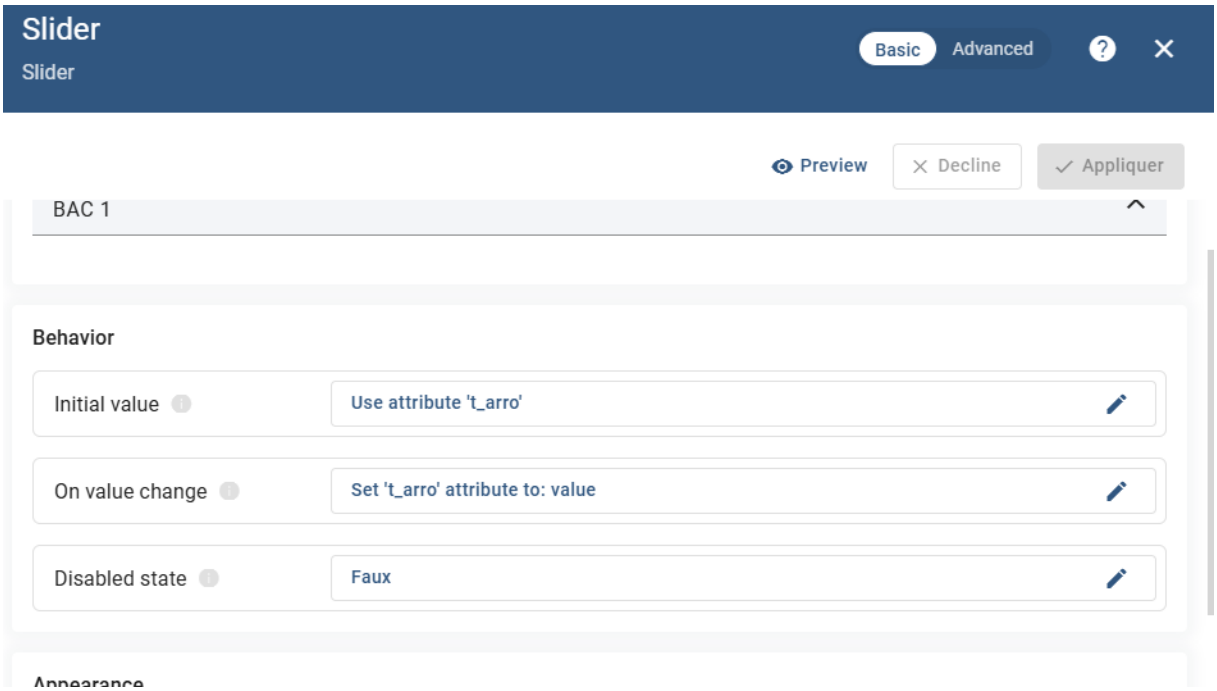
globalement les même que pour le bouton précédent sauf qu'il n'y a qu'un seul état (celui appuyé ) à remplir.



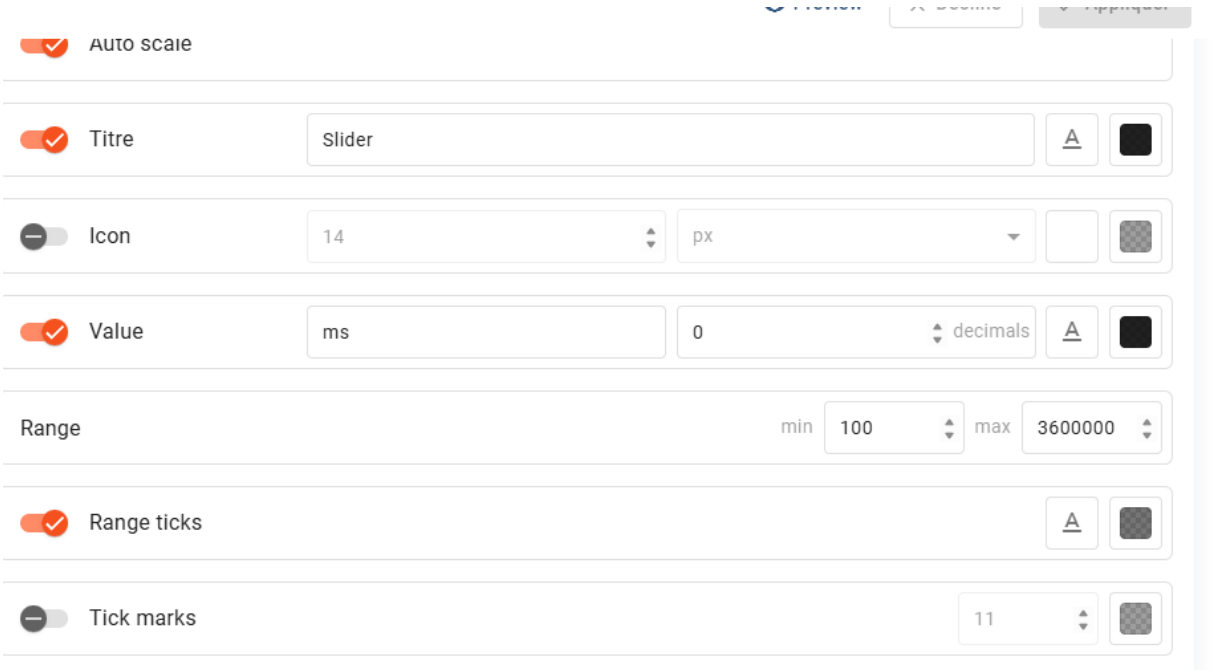

Afin de contrôler le temps d'arrosage t_arro en milliseconde on va devoir rajouter le un widget venant de « control widgets » puis « sliders ».



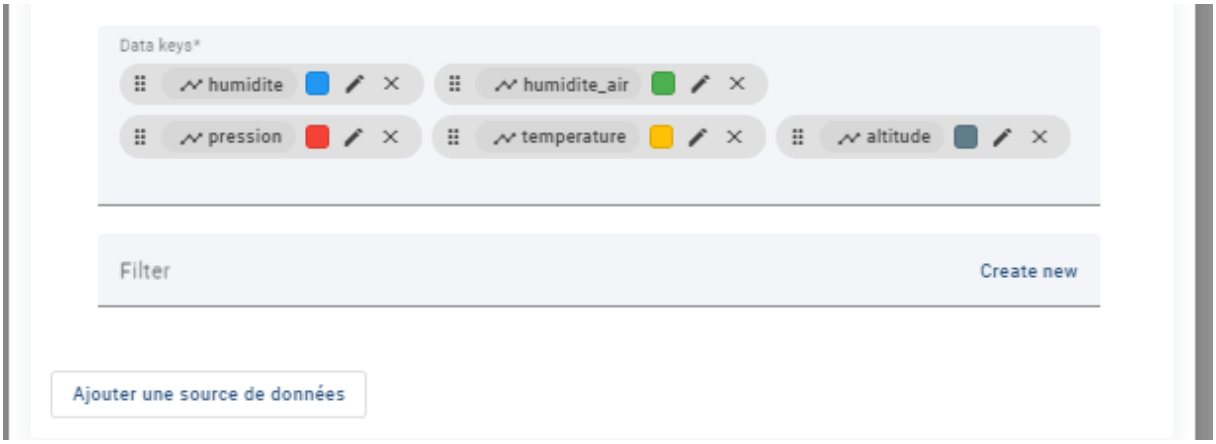
On met dans ce dernier le bon device et en « initial value » get attribute -> shared-> t_arro et en « on value change » set attribute -> Shared-t_arro.




Il est conseillé de décocher l'option « tick marks » afin de ne pas être limité à certaines valeurs dans votre ranger et pouvoir choisir

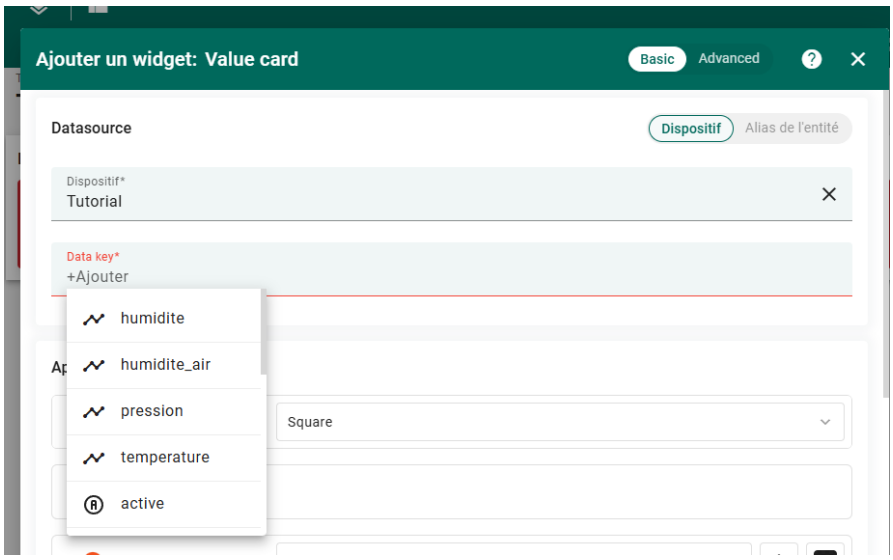


On peut maintenant afficher les valeurs de nos capteurs, soit en les mettant dans un tableau grâce aux widgets de la catégorie « card », ou il faut alors donner le nom du device et choisir toutes les variables qu'on veut dans notre tableau :




soit en prenant un affichage individuel pour chaque valeur et dans ce cas il faut aller chercher « value card » dans la catégorie card , ou une de ses variables avec ou sans graphique etc..

Dans tous les cas il suffit alors de renseigner le device et la valeur à afficher.




En ce qui concerne la mise ne page vous pouvez modifier la taille directement en tirant sur les objets, et leur couleur dans les paramètres de chaque widget.

Il est conseiller d'enregistrer les changements régulièrement et surtout de ne pas quitter sans le faire sinon tout changement non enregistrer sera perdu.

5) Code brut à recopier :

- settings.h

```
 //-------------------------------------------------------------------------
------
//Listes des librairies utile au fonctionnement des object connecté à la
carte.
#include <Adafruit_BME280.h> // Utile pour extraire les infos du BME-BMP
280.
//déclaration du BMP connecter à la carte.
Adafruit_BME280 bmp;
//-------------------------------------------------------------------------
-----
#include <SerialMP3Player.h>
// Constante du lecteur carte SD vers prise jack ou enceinte ici
#define RX 16// RX et TX sont les pins nécessaires pour faire fonctionner
le lecteur.
#define TX 17
#define BtnPin 18 // pin du bouton qui permettra à la chanson de se lancée
// Penser à  bien mettre  TX devant RX  lors de la déclaration,  sinon
connecter le fil du TX du lecteur au RX declaré et inversément.
SerialMP3Player mp3(TX,RX);
//-------------------------------------------------------------------------
-----
//Definition du pins du capteur d'humidité (si capacitif faire attention à
ce que ce soit un pin analogique)
 #define moisturePin 34
  //Definition du  taux d'humidité critique necessitant l'arrosage et sa
valeur par défaut.
 int humidityRate=30;
 //Definition du pin du relais permettant d'activation de la pompe.
 #define relayPin 19
 //-------------------------------------------------------------------------
------
#include <HTTPClient.h> // Permet de se connecter à un site internet (ici
API météo)
#include <ArduinoJson.h> // Permet d'extraire et de traiter les
informations de l'API météo
//// Constantes pertinentes pour l'API
String api_key = "48af8fee764ae56aa1395fe51afba45c"; // clefs
String api_place="Quetigny"; // Lieux des infos voulues
String api_country="FR"; //Code à 2 lettres du pays de l'endroit voulu.
String url =
"http://api.openweathermap.org/data/2.5/forecast?q="+api_place+","+api_coun
try+"&APPID=" + api_key;
// lien complet de l'API pour checker  la forme envoyée, au cas ou elle
évolurais ou de nouvelle infos serais interessante à aller
cherchée(control+click pour ouvrir)
```

```
//http://api.openweathermap.org/data/2.5/forecast?q=Quetigny,FR&APPID=48af8fee764ae56aa1395fe51afba45c

//Definition des proba pour chaque jour parmis les  5 jours dispo sur l'API
float p_J1=1.0; //p sont les proba intermédiaires de "non pluies" qui seront cumulées
float p_J2=1.0 ;
float p_J3=1.0 ;
float p_J4= 1.0;
float p_J5= 1.0;
float pop_J1=0.0;// pop sont les proba de pluies  calculé comme  1-proba non pluies à la fin
float pop_J2=0.0;
float pop_J3=0.0;
float pop_J4=0.0;
float pop_J5=0.0;


// temps depuis le dernier call de l'API
unsigned long lastTime = 0;
// temps de  delay entre les requete API.
unsigned long timerDelay = 10800000; // ICI  10sec pour les phases de test mais vu que l'api donne les donnée par tranches de 3h on peut réduire à une requête par 3h par la suite.

// On défini le futur buffer  utile dans notre code.
 String jsonBuffer;
//-------------------------------------------------------------------------------
// Liste des librairies utile au fonctionnement de la connexion à internet et aux choses liées.
#include <WiFi.h>// Permet d'initier le WiFiClient (ou la connexion au WiFI  si hard coder, et pas de surcouche WiFi Manager)
#include <WiFiManager.h>//Permet de reconnecter au WiFi manuelle  encas de changement d'endroit ou redemarrage sans avoir à le hard coder.
#include <Arduino_MQTT_Client.h> // Permet d'utilisé le protocole MQTT pour comuniquer via les ports MQTT de ThingsBoard (TB)
#include <ThingsBoard.h> // Permet de communiquer avec le serveur TB une fois ce  dernier
//-------------------------------------------------------------------------------
//Nouveau Serveur 1
constexpr char THINGSBOARD_SERVER[] = "192.168.10.29";
//token  serveur TB de ore 1
constexpr char TOKEN[]="B1_TEST_TOKEN"; //device: test pompe TB simon  , dashboard: pareil qu'au dessus , msg d'erreur en français interne au code de l'IDE "echec de la connexion"
```

```
// Port MQTT utilisé pour  connecter à TB, le  1883 est celui non crypté
par défaut.
constexpr uint16_t THINGSBOARD_PORT = 1883U;
// Taille maximale des packets échanger entre serveur et carte
//bien vérifier qu'on ne dépasse la dépasse pas par la suite afin d'avoir
toute les données envoyée/reçue.
constexpr uint32_t MAX_MESSAGE_SIZE = 1024U;
// Nbr d'attribute d'un type pour lesquels on souhaite s'inscrire à TB
constexpr size_t MAX_ATTRIBUTES = 3U;
//------------------------------------------------------------------------
-----
//Déclaration du client TB en 3 étapes
// Initialisation du client Wi-Fi via WiFi.h/*
WiFiClient wifiClient;
//Initialisation du WiFi manager
WiFiManager WM;
const char* ssid = "Bac_3";
const char* password = "CodeBac3";
// Initalisation du client MQTT
Arduino_MQTT_Client mqttClient(wifiClient);
// Intitialisation de TB avec la taille de buffer maximale.
ThingsBoardSized<Default_Fields_Amount, Default_Subscriptions_Amount,
MAX_ATTRIBUTES> tb(mqttClient, MAX_MESSAGE_SIZE);
//------------------------------------------------------------------------
-----
// Baude rate  utile pour avoir le moniteur série lisible afin de debug si
besoin.
constexpr uint32_t SERIAL_DEBUG_BAUD = 115200U;
```

- attributes.h

```
//Nommé tout les attributs pertinent pour la suite
constexpr char T_ARROSAGE_ATTR[] = "t_arro"; //temps d'arrosage
constexpr char BTN_MANUEL_ATTR[]="btn_manuel"; //variable du bouton arrosage
manuel
constexpr char BTN_AUTO_ATTR[]= "btn_auto";//variable du bouton type
d'arrosage


//Valeurs par défauts de nos attributs.
volatile bool btn_manuel =false;//Le mode manuel l'arrosage ne se lance pas
avant l'input utilisateur.
volatile bool btn_auto= false;// Ici on souhaite que le mode auto soit mis par
défaut
volatile uint16_t t_arro=1000U; //temps par défaut d'arrosage de
1ssec,  surtout un place holder qui sera à modif par le jardinier.
```

```
// On défini les limite des temps d'arrosage possible afin d'éviter des temps
absurde à cause de faute de frappe (encore une fois place  holder à modifié
après consulation des jardiniers)
constexpr uint16_t T_ARROSAGE_MS_MIN = 10U;
constexpr uint16_t T_ARROSAGE_MS_MAX = 60000U;

// Bool utile pour stocker si on a changer un attributs ou pas.
volatile bool attributesChanged = true;
// et constante qui servira à compter le temps entre les changement d'état.
uint32_t previousStateChange;

// Constante pour la  télémétrie
constexpr int16_t telemetrySendInterval = 2000U; //Temps entre les envoie de
télémétrie
uint32_t previousDataSend; // Temps depuis la dernière télémétrie.


// List of shared attributes for subscribing to their updates
constexpr std::array<const char *, 3U> SHARED_ATTRIBUTES_LIST = {
  T_ARROSAGE_ATTR,
  BTN_MANUEL_ATTR,
  BTN_AUTO_ATTR
};
//-------------------------------------------------------------------------
--
void processSharedAttributes(const JsonObjectConst &data) {
  for (auto it = data.begin(); it != data.end(); ++it) {
    // cette condition permet de verifier si 2 chaines de caractère entre
l'input
    //et la valeur stocké dans la carte sont identique.

    // même principe de comparasion
     if (strcmp(it->key().c_str(), BTN_MANUEL_ATTR) == 0) {
      //si y'a modification alors on  stocke le nouvel état dans la variable
du btn manuel.
      btn_manuel = it->value().as<bool>();
      Serial.print("Changement état bouton manuel: ");
      Serial.println(btn_manuel );

    }
    //Exactement le même else if pour l'autre bouton.
    else if (strcmp(it->key().c_str(), BTN_AUTO_ATTR) == 0) {
      btn_auto = it->value().as<bool>();
      Serial.print("Changement état bouton auto: ");
      Serial.println(btn_auto );
    }
    else if (strcmp(it->key().c_str(), T_ARROSAGE_ATTR) == 0) {
      //Si ce n'est pas le cas avec on prendre la chaine de caractère input
      //et on le met sont la forme d'un entier qu'on pourra traiter
```

```
      const uint16_t new_interval = it->value().as<uint16_t>();
      //Si le nouvel intervalle est entre les valeurs min et max autorisé.
      if (new_interval >= T_ARROSAGE_MS_MIN && new_interval <=
T_ARROSAGE_MS_MAX) {
        // alors on def le temps d'arrosage comme le nouveau intervalle et on
le comunique sur le moniteur série.
        t_arro = new_interval;
        Serial.print("Temps arrosage modifier : ");
        Serial.print(new_interval);
        Serial.println(" ms");
      }
    }
  attributesChanged = true; // On  stocke que l'on a changer les  attributs
  }
}
//-------------------------------------------------------------------------
--
//Definition des callback pour  les atttributs partagé (on peut aussi pour les
autres attributs: client mais nous n'en avons pas ici )
const Shared_Attribute_Callback<MAX_ATTRIBUTES>
attributes_callback(&processSharedAttributes, SHARED_ATTRIBUTES_LIST.cbegin(),
SHARED_ATTRIBUTES_LIST.cend());
const Attribute_Request_Callback<MAX_ATTRIBUTES>
attribute_shared_request_callback(&processSharedAttributes,
SHARED_ATTRIBUTES_LIST.cbegin(), SHARED_ATTRIBUTES_LIST.cend());
//-------------------------------------------------------------------------
--
void processSetBtnManuel(const JsonVariantConst &data, JsonDocument &response)
{
  Serial.println("Reception etat du bouton manuel via methode RPC");
  // Stocke la donnée reçue comme "nouveau mode"
  int new_mode = data;

  Serial.print("Mode du bouton changer : ");
  Serial.println(new_mode);
  StaticJsonDocument<1> response_doc;

  if (new_mode != 0 && new_mode != 1) { /// Si le mode n'est pas un entier
soit 0 ou 1 (booléen true/false) alors on a une erreur
    response_doc["error"] = "Mode Inconnu!";
    response.set(response_doc);
    return; // si erreur il y a on sors imédiatement de la fonction
  }
  //Si pas d'erreur on met à jour l'état du bouton de l'état manuel.
  btn_manuel = new_mode;

  //On stocke que l'on a bien changer un attribut.
  attributesChanged = true;
```

```
  // On renvoi l'état actuel du bouton.
  response_doc["newMode"] = (int)btn_manuel;
  response.set(response_doc);
}
//-------------------------------------------------------------------------
--
void processSetBtnAuto(const JsonVariantConst &data, JsonDocument &response) {

  Serial.println("Reception etat du bouton mode auto  via methode RPC");
  // Stocke la donnée reçue comme "nouveau mode"
  int new_mode = data;

  Serial.print("Mode modifier vers : ");
  Serial.println(new_mode);
  StaticJsonDocument<1> response_doc;

  if (new_mode != 0 && new_mode != 1) { /// Si le mode n'est pas un entier
soit 0 ou 1 (booléen true/false) alors on a une erreur
    response_doc["error"] = "Mode Inconnu!";
    response.set(response_doc);
    return; // si erreur il y a on sors imédiatement de la fonction
  }
  //Si pas d'erreur on met à jour l'état du bouton de l'état manuel.
  btn_auto = new_mode;

  //On stocke que l'on a bien changer un attribut.
  attributesChanged = true;

  // On renvoi l'état actuel du bouton.
  response_doc["newMode"] = (int)btn_auto;
  response.set(response_doc);
}
//-------------------------------------------------------------------------
--
// Définition des fonction RPC qui seront utile dans TB par la suite
const std::array<RPC_Callback, 2U> callbacks= {
  RPC_Callback{"setBtnAuto",processSetBtnAuto},
  RPC_Callback{"setBtnManuel",processSetBtnManuel}
  };
  // la partie entre "" correspond au nom qu'ils auront pour TB
  // la partie après correspond au nom de la fonction dans le code arduino
```

- fonctions.h

```
// Fonction qui initie la connection Wifi et permet d'avoir un retour visuel
niveau moniteur série pour son avancement.

void InitWiFi() {
```

```
 WiFi.mode(WIFI_STA);
 while(!WM.autoConnect(ssid, password))
 {
  Serial.print(".");
 }
  Serial.print("Connecte");
}
//------------------------------------------------------------------------
--

// Fonction de reconnect si la carte detecte qu'elle a été déconnctée.
const bool reconnect() {
  // Teste que l'on a pas  été connecté dans une variable status.
  const wl_status_t status = WiFi.status();
  if (status == WL_CONNECTED) {// si le status est connecté
    return true; //return faire sortir de la fonction sans traiter les chose
après le IF
  }
  //Si if pas vérifié alors on relance InitWiFi
 InitWiFi();
 Serial.println("reconnect qui fait chier");
  return true;
}
// ------------------------------------------------------------------------
---
void TBLancement (){
  if (!tb.connected()) {
    // Connexion à  TB
    Serial.print("Connexion a : ");
    Serial.print(THINGSBOARD_SERVER);
    Serial.print(" avec le device  ");
    Serial.println(TOKEN);
    if (!tb.connect(THINGSBOARD_SERVER, TOKEN, THINGSBOARD_PORT)) { // Si tb
connect n'est tjr pas true.
      Serial.println("Echec de la connexion");
      return;
    }
    // Si on a pu se connecter alors on  envoie notre adress max
en  attribute.
    tb.sendAttributeData("macAddress", WiFi.macAddress().c_str());

    Serial.println("Subscribing for RPC...");
    //On va ici faire bcp de conditions afin de sortir des message d'erreur
précis pour le debug.
    if (!tb.RPC_Subscribe(callbacks.cbegin(), callbacks.cend())) {// Si il y a
un echec pour les fonction RPCs.
      Serial.println("Failed to subscribe for RPC");
      return;
    }
```

```
  if (!tb.Shared_Attributes_Subscribe(attributes_callback)) { // Si il y a un
échec pour les attributs partagés.
      Serial.println("Failed to subscribe for shared attribute updates");
      return;
    }

    Serial.println("Inscription réussie");

    // Si on ne peut pas demander l'état actuel des attributs partagés.
    if (!tb.Shared_Attributes_Request(attribute_shared_request_callback)) {
      Serial.println("Failed to request for shared attributes");
      return;
    }
  }
}

// ----------------------------------------------------------------------------
---
String httpGETRequest(const char* serverName) {
  WiFiClient client;
  HTTPClient http;

  // On lance la  connexion au site avec le  url donné à la fonction et le
wifi client défini plus haut.
  http.begin(client, serverName);

  // On demande une réponse  au serveur HTTP
  int httpResponseCode = http.GET();
  // On défini le payload qui sera notre  sorte de fonction comme une string
vide.
  String payload = "{}";

  if (httpResponseCode>0) {// Si la réponse du  serveur est pas nulle
    Serial.print("HTTP Response code: ");// on print la réponse dans le serial
    Serial.println(httpResponseCode);
    payload = http.getString();// On la stocke dans notre payload
  }
  else {// sinon on envoie un message d'erreur.
    Serial.print("Error code: ");
    Serial.println(httpResponseCode);
  }
  // On fini les demande au serveur HTTP
  http.end();

  // la fonction ressort ce que le serveur HTTP à donné.
  return payload;
}
// ----------------------------------------------------------------------------
---
```

```
void getInfoAPI(){
 if ((millis() - lastTime) > timerDelay) {// si le  delay entre demande api
est dépasser
    // On teste le status du WiFi.
    if(WiFi.status()== WL_CONNECTED){
      jsonBuffer = httpGETRequest(url.c_str()); // on met dans le buffer la
réponse si on connecte à l'API
      Serial.println(jsonBuffer);// on le print
      JsonDocument myObject;// On  défini un objet json qui sera nos infos.
      deserializeJson(myObject,jsonBuffer);// On stocke le  buffer désérialisé
dans l'objet.
      // On va avoir  5 boucle similaire donc je vais en détailler qu'une .
      p_J1=1.0; // Notre proba de  non pluie est à priori 1.
      Serial.println("J1");
      for (int i=0;i<8;i++){
        float buff=float(myObject["list"][i]["pop"]);// On prend la proba de
pluie venu de l'API à l'étape en cours.
        p_J1=p_J1*(1-buff);// La proba de non pluie sur ce jour là correspond
à la proba de non pluie jusque là  multiplié par 1- la proba de pluie à cette
étape.
        Serial.println(p_J1);
      }
      Serial.println("J2");
      p_J2=1.0;
      for (int i=8;i<16;i++){
        float buff=float(myObject["list"][i]["pop"]);
        p_J2=p_J2*(1-buff);
        Serial.println(p_J2);
      }
      Serial.println("J3");
      p_J3=1.0;
      for (int i=16;i<24;i++){
        float buff=float(myObject["list"][i]["pop"]);
        p_J3=p_J3*(1-buff);
        Serial.println(p_J3);
      }
      Serial.println("J4");
      p_J4=1.0;
      for (int i=24;i<32;i++){
        float buff=float(myObject["list"][i]["pop"]);
        p_J4=p_J4*(1-buff);
        Serial.println(p_J4);
      }
      Serial.println("J5");
      p_J5=1.0;
      for (int i=32;i<40;i++){
        float buff=float(myObject["list"][i]["pop"]);
        p_J5=p_J5*(1-buff);
        Serial.println(p_J5);
```

```
      }
      // Une fois qu'on a toute nos proba de non plus chaque jour on fais  1-
proba pour avoir les proba de pluie sur les journées, et on les  transforme en
% par soucis de lisibilité sur TB.
      pop_J1= (1-p_J1)*100;
      pop_J2= (1-p_J2)*100;
      pop_J3= (1-p_J3)*100;
      pop_J4= (1-p_J4)*100;
      pop_J5= (1-p_J5)*100;
      Serial.print("POP sur  24h: ");
      Serial.println(pop_J1);
    }
    else {
      Serial.println("WiFi Disconnected"); // si le wif est pas  connecté
on  envoie un msg d'erreur.
    }
    lastTime = millis(); // on reset le lasttime auquel on a pris les donnée
de l'API.
  }
}
// ----------------------------------------------------------------------------
---
void ResetWiFi(){
  if (analogRead(35)>1000){
    WM.resetSettings();
    ESP.restart();
  }
}
```

- Bac_jardin.ino

```
//Include les  2 autres fichier .h lié à ce dossier qui seront essentiel par
la suite
#include "settings.h"
#include "attributes.h"
#include "fonctions.h"
void setup() {
  //On lance le moniteur série.
  Serial.begin(SERIAL_DEBUG_BAUD);
  //On lance le BMP
  bmp.begin(0x76);
  //On défini que l'on veut mesurer le courant dans le pin du capteur
d'humidité
  pinMode(moisturePin,INPUT);
  // On défini que l'on a un message à envoyer au relais
  pinMode(relayPin,OUTPUT);
    // passe le pin du bouton pour lancer la musique en input
  pinMode(BtnPIN,INPUT);
```

```
  // Init le lecteur MP3
  mp3.begin(SERIAL_DEBUG_BAUD);
  // Demande au MP3 de prendre la carte SD en compte.
  mp3.sendCommand(CMD_SEL_DEV, 0, 2);

  // On lance le Wi-Fi
  InitWiFi();
}

void loop() {
TBLancement ();
ResetWiFi();
getInfoAPI();//On prend les infos API , voir listes fonctions.
  if (attributesChanged) { // si les attributs ont changés.
   attributesChanged = false; // on repasse la variable en false.
   // On envoie les valeurs attributs de nos attributs.
    tb.sendAttributeData(T_ARROSAGE_ATTR, t_arro);
    tb.sendAttributeData(BTN_MANUEL_ATTR,btn_manuel);
    tb.sendAttributeData(BTN_AUTO_ATTR,btn_auto);
    //On envoie les valeurs télémétrie de nos attributs.
    tb.sendTelemetryData(T_ARROSAGE_ATTR, t_arro);
    tb.sendTelemetryData(BTN_MANUEL_ATTR,btn_manuel);
    tb.sendTelemetryData(BTN_AUTO_ATTR,btn_auto);
  }
//On mesure la valeur que renvoie le capteur que l'on remap sur une échelle
de  0 à 100
int moisture=map(analogRead(moisturePin),0,2830,0,100);
//on envoie un retour sur le moniteur série.
  if (btn_auto==true){// si mode automatique actif
    Serial.println("mode auto actif");
    if (moisture>=humidityRate){ // et que l'humidité est en dessous de la
limite fixé dans settings.
      Serial.println("arrosage automatique en cours ");
      digitalWrite(relayPin,HIGH); /// on allume l'arrosage pour une
durée  t_arro
      delay(t_arro);
      digitalWrite(relayPin,LOW);
      btn_manuel=false;
      attributesChanged = true;
    }
    else { // si l'humidité est bonne alors on laisse la pompe éteinte.
    digitalWrite(relayPin,LOW);
    btn_manuel=false;
    attributesChanged = true;
    }
  }
  else { // si le mode auto est sur false.
  Serial.println("mode manuel actif");
  if (btn_manuel==true){ // et que le btn d'arosage manuel est appuyé
```

```
    Serial.println("arrosage manuel en cours ");
    digitalWrite(relayPin,HIGH); // on lance l'arrosage pour une durrée t_arro
    delay(t_arro);
    digitalWrite(relayPin,LOW);
    btn_manuel=false;
    attributesChanged = true;
  }
  else {// si le btn manuel  n'est pas  activé alors on éteint la pompe.
    digitalWrite(relayPin,LOW);
  }
  }
  if (digitalRead(BtnPIN)>0){
    mp3.play(1);
  }
  // On prend toute les mesures utile  du BMP.
  float temp=bmp.readTemperature();
  float alt=bmp.readAltitude(1000);
  float press=bmp.readPressure()/100;
  float humi_air=bmp.readHumidity();

  if (millis() - previousDataSend > telemetrySendInterval) {// si le temps
depuis le dernier envoi de télémétrie est plus grand que la  valeur qu'on a
fixé.
   previousDataSend = millis();
  //On envoie nos valeurs  sur TB comme telemetry.
    tb.sendTelemetryData("temperature",temp);
    tb.sendTelemetryData("pression",press);
    tb.sendTelemetryData("altitude",alt);
    tb.sendTelemetryData("humidite_air",humi_air);
    tb.sendTelemetryData("humidite",moisture);
    // On envoie alors les valeurs de télémétrie des proba de pluies pour
les  5 prochains jours.
    tb.sendTelemetryData("pluie_jour_1",pop_J1);
    tb.sendTelemetryData("pluie_jour_2",pop_J2);
    tb.sendTelemetryData("pluie_jour_3",pop_J3);
    tb.sendTelemetryData("pluie_jour_4",pop_J4);
    tb.sendTelemetryData("pluie_jour_5",pop_J5);
    // et comme attribut pour celle lié au

  }
  delay(1000);
  tb.loop();
}
```