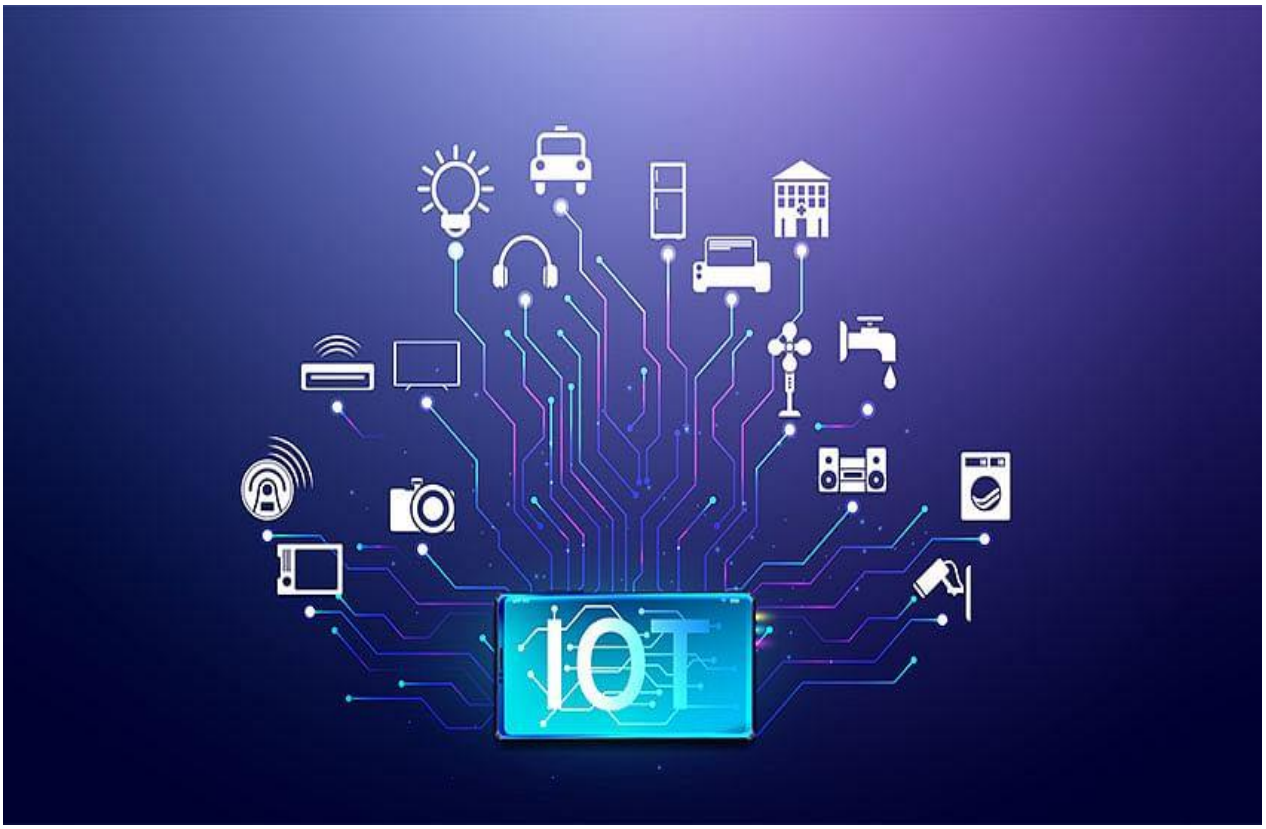
Taquet Timon

Nwogburu Michael

Année académique 2022-2023



PORTFOLIO

SYSTÈMES EMBARQUÉS ET INDUSTRIE 4.0

Professeur : E.Costa

Haute-école : EPHEC-ISAT

Table des matières

Introduction

Dans le cadre des cours de systèmes embarqués 2 et industrie 4.0, nous avons pu découvrir la puissance des microcontrôleurs 32 bits comme l'ESP32 pour travailler dans l'IOT (Internet Of Things). Nous allons vous présentez dans ce portfolio les 8 travaux pratiques qui nous on permit d'en apprendre plus sur le fonctionnement générale de tout cela, notamment à l'aide d'outils comme les Dashboards , les databases ou encore les serveurs. Ces différents outils sont très puissants pour faire interagir des composants Hardware avec des logiciels Software.

1) TP1_BUSCAN

Objectif du TP

L'objectif de ce Tp est d'introduire le protocole de communication **BUS CAN**.

Pour ce Tp nous devrons implémenter.

- Un compteur avec deux boutons, pour l'incrémentation et la décrémentation de celui-ci.
- Un potentiomètre
- Un écran LCD

La valeur du compteur et du potentiomètre devra être envoyée via le protocole de communication.

Pour finir, les données reçues devront être affichées sur l'écran LCD avec la valeur du compteur et du potentiomètre.

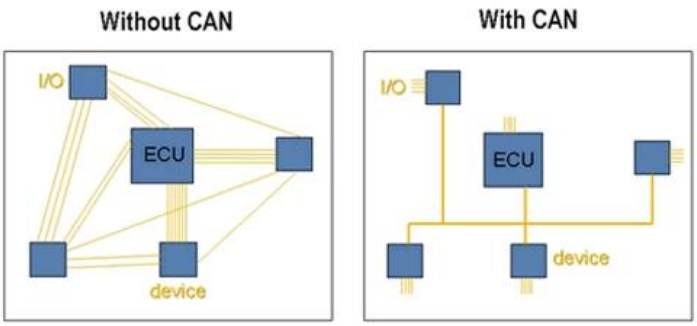
Introduction

CAN

Un bus CAN (Controller Area Network) est un système de bus série qui a pour objectif de mettre en réseau plusieurs dispositifs intelligents. Ces bus sont souvent utilisés dans les industries. À l'aide d'un périphérique d'interface CAN, il est possible d'envoyer plusieurs données jusqu'à un certain nombre de participants et tout ceci de concert.

Le protocole de communication CAN BUS permet aussi de réduire le coût de l'installation car les participants seront connectés à un bus directement de manière propre et simple.

Voit image ci-dessous

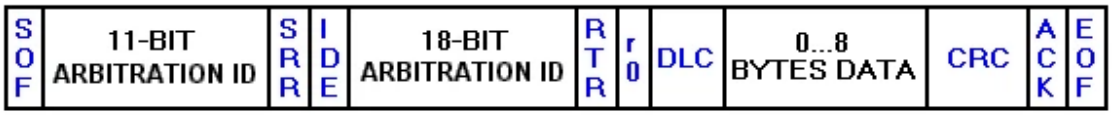


Can Protocol

Le protocole de communication CAN est un protocole d'accès multiple à détection de porteuse avec détection de collision et arbitrage sur la priorité des messages (CSMA/CD+AMP).

- CSMA- signifie que chaque nœud sur un bus doit attendre une période d'inactivité prescrite avant de tenter d'envoyer un message.
- CD- Collision détection.
- AMP- Arbitrage sur la priorité des messages.

CAN FRAME



- **SOF (start-of-frame) bit**- indique le commencement du message avec un logic 0 bit.

- **ARBITRATION ID**- identifie le message et indique la priorité du message. Les trames sont disponibles en deux formats : standard, qui utilise un ID d'arbitrage de 11 bits, et étendu, qui utilise un ID d'arbitrage de 29 bits.

- **IDE**- permet de différencier les trames standards et étendues.

- **RTR**- sert à différencier une trame distante d'une trame de données. Un bit RTR dominant (logique 0) indique une trame de données. Un bit RTR récessif (logique 1) indique une trame distante.

- **DLC**- indique le nombre d'octets que contient le champ de données.

- **Data Field**- contient de 0 à 8 octets de données.

- **CRC**- contient un code de contrôle de redondance cyclique de 15 bits et un bit de délimitation récessif. Le champ CRC est utilisé pour la détection d'erreur.

- **ACK (ACKnowledgement) slot** - tout contrôleur CAN qui reçoit correctement le message envoie un bit ACK à la fin du message. Le nœud émetteur vérifie la présence du bit ACK sur le bus et retente la transmission si aucun accusé de réception n'est détecté.

Code source avec commentaires

```
1   unsigned char Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags;  // can
 2 flags
 3 unsigned char Rx_Data_Len;                                     //
 4 received data length in bytes
 5 char RxTx_Data[8];                                             // can
 6 rx/tx data buffer
 7 char Msg_Rcvd;                                                 //
 8 reception flag
 9 const long ID_1st = 12111, ID_2nd = 3;                         // node
10 IDs
11 long Rx_ID;
12 int BP1 = 0;
13 int C1 = 0;  //bouton 1
14 int C_rcpt = 0;
15 int BP2 = 0;
16 char buff_envoi[8];
17 char buffR[2];  //bouton2
18 char buffLcd[8];
19 int yourpot = 0;
20
21 // CANSPI module connections
22 sbit CanSpi_CS at RC0_bit;
23 sbit CanSpi_CS_Direction at TRISC0_bit;
24 sbit CanSpi_Rst at RC2_bit;
25 sbit CanSpi_Rst_Direction at TRISC2_bit;
26 // End CANSPI module connections
27
28 sbit LCD_RS at RB4_bit;  //connexion microcontroleur/lcd
29 sbit LCD_EN at RB5_bit;
30 sbit LCD_D4 at RB0_bit;
31 sbit LCD_D5 at RB1_bit;
32 sbit LCD_D6 at RB2_bit;
33 sbit LCD_D7 at RB3_bit;
34 sbit LCD_RS_Direction at TRISB4_bit;
35 sbit LCD_EN_Direction at TRISB5_bit;
36 sbit LCD_D4_Direction at TRISB0_bit;
37 sbit LCD_D5_Direction at TRISB1_bit;
38 sbit LCD_D6_Direction at TRISB2_bit;
39 sbit LCD_D7_Direction at TRISB3_bit;
40
```

```
41 unsigned int valeur_pot;
42 int Vitesse_pot;
43 int memo_ADC;
44
45 void LCD(){
46      LCD_Cmd(_LCD_CLEAR);
47      sprintf(buffLcd, "Tr %d,%04d", C1, valeur_pot);
48      lcd_out(1, 1,buffLcd);
49      sprintf(buffLcd, "Re %d,%04d",C_rcpt, yourpot);;
50      lcd_out(2, 1, buffLcd);
51 }
52
53 void Transmission(){
54     buff_envoi[0] = valeur_pot >> 8;
55     buff_envoi[1] = valeur_pot & 0xFF;
56     buff_envoi[2] = C1;
57     CANSPIWrite(ID_1st, buff_envoi,3,Can_Send_Flags);
58     LCD();
59     /*sprintf(buffLcd, "Tr %d,%04d", C1, valeur_pot);
60     lcd_cmd(_lcd_clear);
61     lcd_out(1, 1,buffLcd);*/
62 }
63 void Reception(){
64     yourpot = RxTx_Data[0] << 8;
65     yourpot = yourpot|RxTx_Data[1];
66     C_rcpt =  RxTx_Data[2];
67     LCD();
68     /*sprintf(buffLcd, "Re %d,%04d",C_rcpt, yourpot);
69     lcd_cmd(_lcd_clear);
70     lcd_out(2, 1, buffLcd);*/
71
72
73 }
74
75
76 /*void ADC() {
77   valeur_pot = ADC_Read(0);  //prise de la valeur du potentiometre
78   //Bcd2Dec(valeur_pot); //prise de valeur sous forme de vitesse
79   Vitesse_pot = valeur_pot / 4;
80   if (abs(memo_ADC - Vitesse_pot) > 30) {
81     //sprintf(buff, "%04d, %04d", C1, Vitesse_pot);
82     //lcd_out(1, 1, buff);
83     //CANSPIWrite(ID_1st, buff, 3, Can_Send_Flags);
84     //memo_ADC = Vitesse_pot;
85   }
86 }  */
87
88 void Plus() {
89   if (Button(&PORTD, 0, 1, 1)) {
90     BP1 = 1;
91   }
92   if (BP1 & Button(&PORTD, 0, 1, 0)) {
93     BP1 = 0;
94     C1++;
95     Transmission();
96   }
97 }
```

```
 98 void Moins() {
 99   if (Button(&PORTD, 1, 1, 1)) {
100     BP2 = 1;
101   }
102   if (BP2 & Button(&PORTD, 1, 1, 0)) {
103     BP2 = 0;
104     C1--;
105     Transmission();
106   }
107 }
108
109 void main() {
110
111   ANSELC = 0;
112   ANSELA = 2;
113   ANSELB = 0;
114   TRISA.f1 = 1;
115   ANSELD = 0;
116   TRISD.f0 = 1;
117   TRISD.f1 = 1;  // Configure AN pins as digital I/O
118   //ANSELH = 0;
119
120   // clear PORTB
121   //TRISB = 0;                                                  // set
122 PORTB as output
123
124   Can_Init_Flags = 0;  //
125   Can_Send_Flags = 0;  // clear flags
126   Can_Rcv_Flags = 0;   //
127
128    Can_Send_Flags = _CANSPI_TX_PRIORITY_0 &                     // form
129 value to be used
130                     _CANSPI_TX_XTD_FRAME &                     //
131 with CANSPIWrite
132                     _CANSPI_TX_NO_RTR_FRAME;
133
134   Can_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE &              // Form
135 value to be used
136                     _CANSPI_CONFIG_PHSEG2_PRG_ON &             //  with
137 CANSPIInit
138                     _CANSPI_CONFIG_XTD_MSG &
139                     _CANSPI_CONFIG_DBL_BUFFER_ON &
140                     _CANSPI_CONFIG_VALID_XTD_MSG;
141
142
143   Lcd_init();
144   Lcd_Cmd(_LCD_CURSOR_OFF);          // Cursor off
145   lcd_out(1,1,"TEST");
146   delay_ms(1000);
147   SPI1_Init();  // initialize SPI1 module
148
149   CANSPIInitialize(3, 8, 3, 3, 1, Can_Init_Flags);
150 // Initialize external CANSPI module
151   CANSPISetOperationMode(_CANSPI_MODE_CONFIG, 0xFF);
152 // set CONFIGURATION mode
153   CANSPISetMask(_CANSPI_MASK_B1, -1, _CANSPI_CONFIG_XTD_MSG);
154 // set all mask1 bits to ones
```

```
155   CANSPISetMask(_CANSPI_MASK_B2, -1, _CANSPI_CONFIG_XTD_MSG);
156 // set all mask2 bits to ones
157   CANSPISetFilter(_CANSPI_FILTER_B2_F4, ID_2nd,
158 _CANSPI_CONFIG_XTD_MSG);  // set id of filter B2_F4 to 2nd node ID
159
160   CANSPISetOperationMode(_CANSPI_MODE_NORMAL, 0xFF);  // set NORMAL
161 mode
162   RxTx_Data[0] = 9;  // set initial data to be sent
163   CANSPIWrite(ID_1st, RxTx_Data, 1, Can_Send_Flags);  // send initial
164 message
165
166
167
168   while (1) {  // endless loop
        Msg_Rcvd = CANSPIRead(&Rx_ID, RxTx_Data, &Rx_Data_Len,
    &Can_Rcv_Flags);
        valeur_pot = ADC_Read(1);  //prise de la valeur du potentiometre
        Plus();
        Moins();

        if ( Msg_Rcvd) {   //(Rx_ID == ID_2nd) &&
           Reception();
          // send incremented data back
        }

        if (abs(memo_ADC - valeur_pot) > 50) {
          memo_ADC = valeur_pot;
          Transmission();
        }

      }
```

Analyse du code source

Déclaration des variables :

Nous commençons par déclarer les variables et les flags utilisés pour la réception des données avec la communication **CAN**.

Ces variables contiennent L'ID du participant avec lequel la communication devra s'établir, le compteur, le potentiomètre et pour finir la déclaration des pins du LCD.

Fonction main :

Cette fonction nous permet d'initialiser les variables, le sens des broches, les objets de classes et pour finir le paramétrage de la **trame CAN**.

Loop :

La fonction **loop** permet de recevoir des données avec la fonction **Reception()** et d'envoyer la valeur du potentiomètre lorsque la différence entre l'ancienne valeur et la valeur actuel est plus grande que 50 avec la fonction **Transmission()**.

Dans cette boucle, nous appellerons aussi la fonction **Plus()** et **Moins().**

Fonction Réceptions :

Cette fonction permet de récupérer la valeur du potentiomètre et du compteur pour les stocker dans la variable «yourtpot » et «C_rcpt». Ensuite, il sera affiché sur l'écran LCD grâce à la fonction **LCD().**

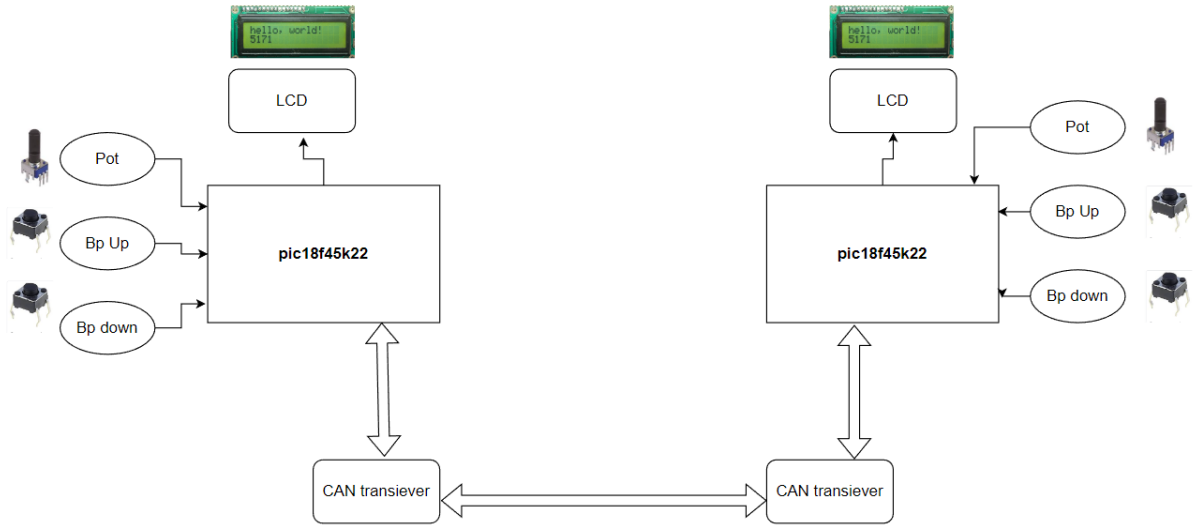
Fonction Transmission :

Cette fonction permet d'envoyer la valeur du potentiomètre et du compteur avec la fonction « **CANSPIWrite(ID,BUFF,Taille,Flag)** ».

Fonction LCD :

Cette fonction permet d'afficher de manière compact les informations sur le LCD grâce à la fonction « **sprintf(buff,)** » pour la mise en forme et « **lcd_out(ligne,colone)**» pour l'affichage.

Lien software hardware



Conclusion

Le Tp CAN permet de prendre connaissance d'un autre type de protocole filaire qui pourrait nous être utile dans divers domaines. Dans ce Tp, nous avons su établir une communication avec 2 participants et envoyer des données d'un sens comme de l'autre.

2) TP2_Webserver

Objectif du TP

Le but du TP est de créer un serveur local grâce à notre esp32.

Notre serveur local devra avoir :

- Une communication bidirectionnel en temps réel à l'aide d'un protocole websocket
- Une gestion des fichiers avec LITLLEFS
- Une page web

La valeur de la température, l'humidité et du bouton poussoir devront être affiché sur notre page web.

Une led devra s'allumer lorsqu'un client est connecté au serveur et s'éteindre si aucun client n'est connecté.

Nous devons aussi pouvoir allumer une led depuis un bouton se situant sur la page web.

Pour finir, l'état de la led, du bouton poussoir, l'adresse IP et les valeurs lues par le DHT22 devront être affichées sur l'écran OLED.

Pour se faire, nous allons utiliser les composants suivant

- DHT22
- OLED
- Bouton poussoir
- 2 led : client, statut

Introduction

Postman

Vue que nous travaillerons avec des serveurs, Postman est un bon outil pour pouvoir faire des « **Get requête** » et des **« Post requête »** de façon simple et efficace.

Les requêtes nous permettent de faire des transferts de données du client au serveur ou du serveur au client avec un **Protocol http.**

Cela peut nous aider à construire toute sorte de requêtes et d'en faire le débogage.

Le **get requête** porte le paramètre de demande ajouté dans la chaîne d'URL.

Le **post requête** porte le paramètre de demande dans le corps du message.

Les données à transmettre ne présentent aucun danger donc nous pouvons utiliser le **get requête.**

Code Postman :

```
// Variables Get
String inputMessage = "";
const char* PARAM_INPUT_1 = "temperature";
const char* PARAM_INPUT_2 = "humi";
const char* PARAM_INPUT_3 = "valeur_bp";
```

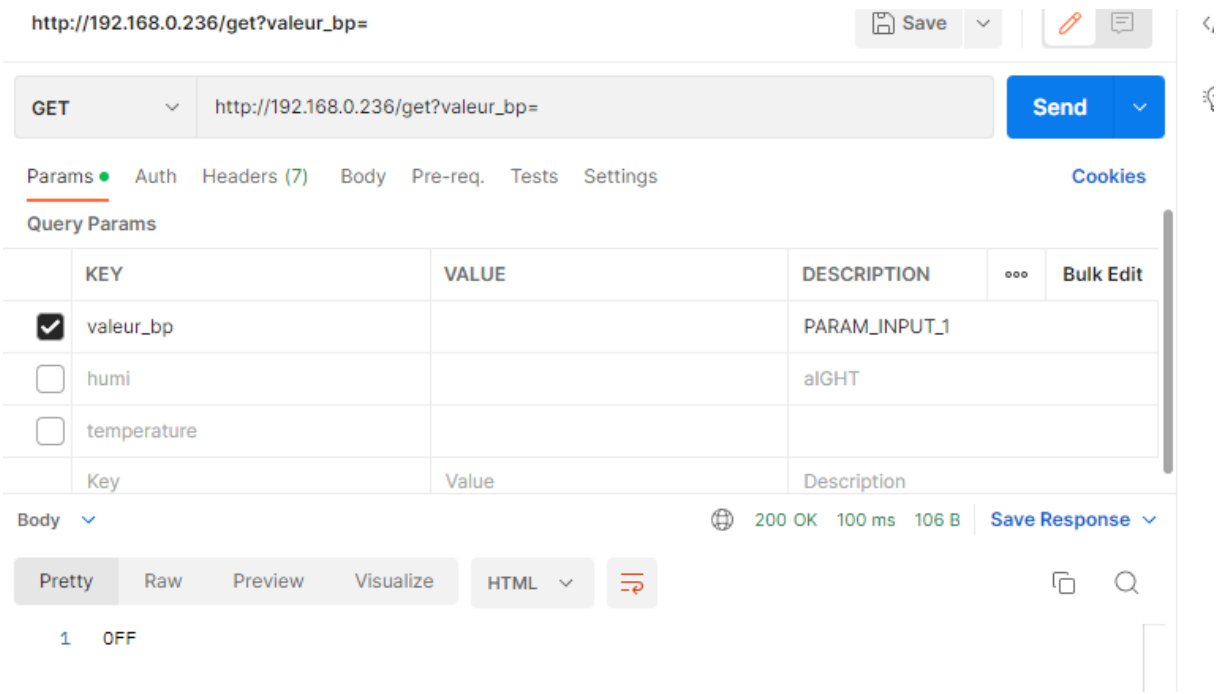
```
// Get request
server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request) {
  if (request->hasParam(PARAM_INPUT_1)) {
    inputMessage = request->getParam(PARAM_INPUT_1)->value();
    Serial.println("Get request:"+inputMessage);
    request->send(200, "text/html", String(temperature));
  }
  if (request->hasParam(PARAM_INPUT_2)) {
    inputMessage = request->getParam(PARAM_INPUT_2)->value();
    Serial.println("Get request:"+inputMessage);
    request->send(200, "text/html", String(humi));
  }
  if (request->hasParam(PARAM_INPUT_3)) {
    inputMessage = request->getParam(PARAM_INPUT_3)->value();
    Serial.println("Get request:"+inputMessage);
    request->send(200, "text/html", String(valeur_bp));
  }
});
```

Nous avons 3 conditions qui ont pour but de vérifier à quel requête le client fait appelle.

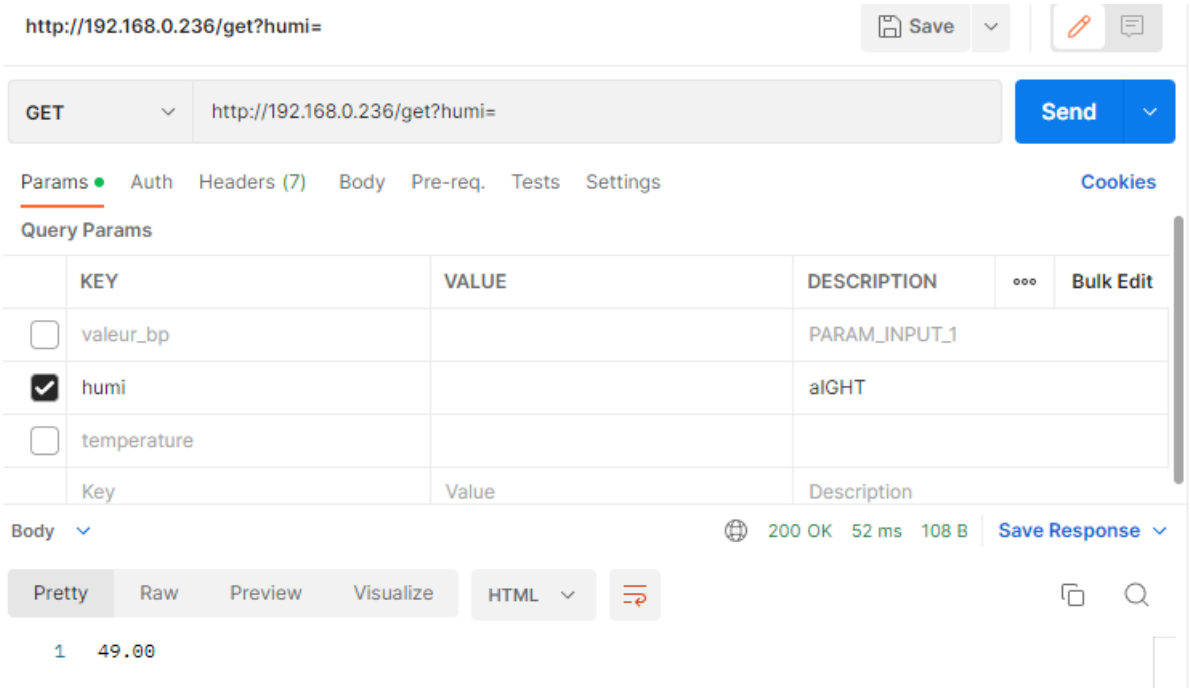
Les requêtes disponibles contiennent la valeur actuelle de la température, de l'humidité et du bouton poussoir.

Si l'une des conditions est vérifiée alors la valeur demandée sera affichée sur une page html sous forme de texte. Voir les exemples ci-dessous dans notre cas de figure.

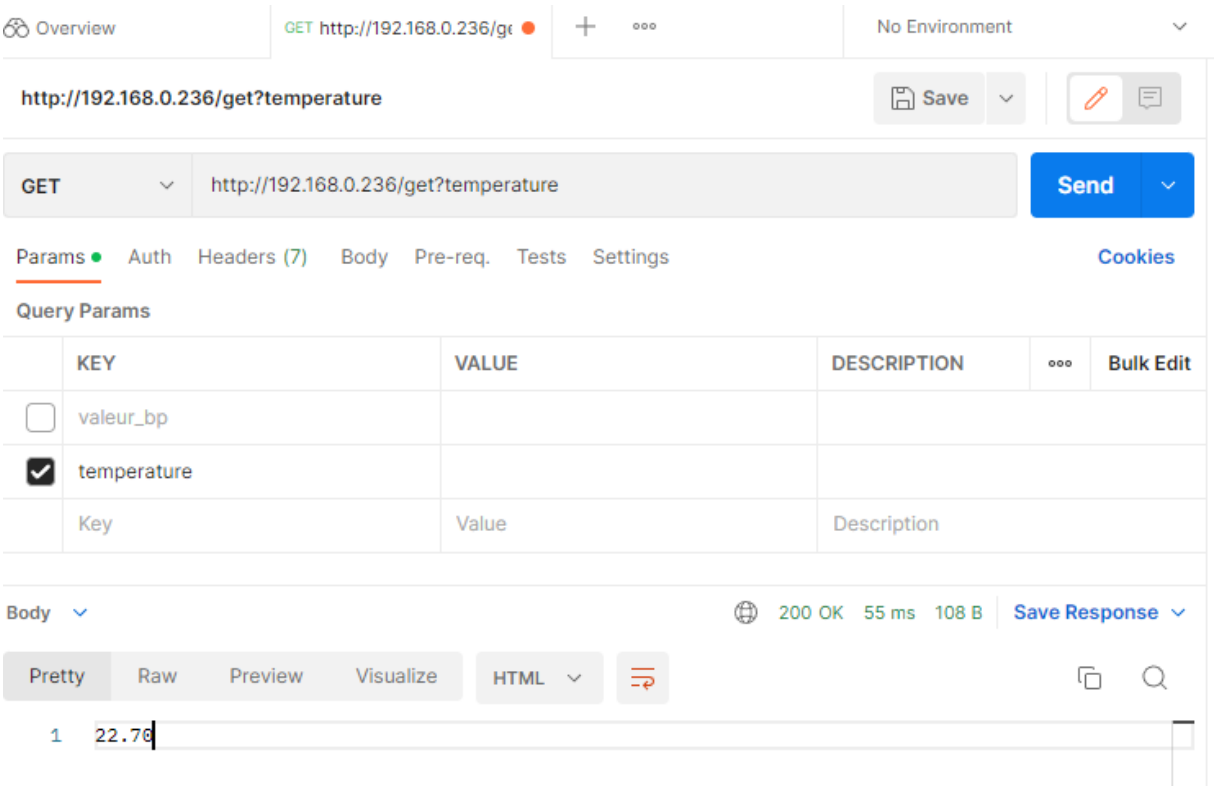
Exemple avec « valeur_bp »



Exemple avec « humi »



Exemple avec «temperature »



Web socket

Qu'est-ce que le WebSocket?

Le websocket est un protocol full duplex sur un socket TCP pour les navigateurs et les serveurs WEB.

L'avantage de ce protocol est le chargement de données qui est beaucoup plus rapide et à temps réels.

Avantage du WebSocket

Dans le cas d'une utilisation classique d'une connexion HTTP, le problème vient du fait que le client doit systématiquement charger la page HTML complète. La technologie AJAX a été créée pour répondre à ce problème. Celle-ci présente cependant un désavantage majeur, en tant que connexion unidirectionnelle : elle ne permet qu'une communication à sens unique, ce qui entraîne des délais considérables au regard du débit actuel et notamment dans les applications de discussions instantanées. En tant que connexion bidirectionnelle permettant un échange dans les deux sens, le WebSocket permet un contact direct avec le navigateur, ce qui permet un **temps de chargement plus**

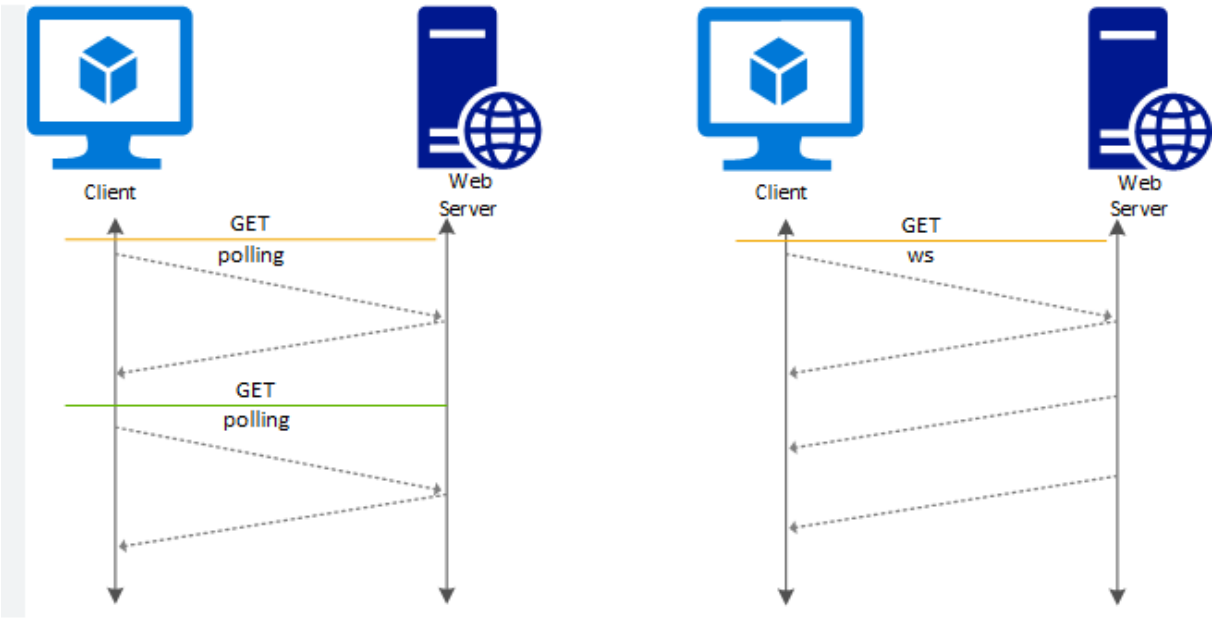
court. Dès qu'un message est disponible, par exemple dans une discussion instantanée avec un service d'assistance, il est affiché sans délai.

Exemples d'utilisations de WebSocket

Le WebSocket convient à toutes les utilisations nécessitant une connexion rapide et à temps réels.

Par exemple :

1. Les chats en ligne
2. L'actualisation d'une position GPS
3. Les jeux en ligne



LITTLEFS

LITTLEFS est une librairie de gestion de fichiers spécialement conçu pour les microcontrôleurs.

Dans le cadre de ce TP nous ferons appel à LTTLEFS pour charger les différents fichiers dans notre serveur web de manière propre et sécurisé.

Implémentation LITTLEFS

Pour implémenter LITTLEFS, il suffit de télécharger la librairie et de l'associer au projet voulu pour ensuite préciser dans platform.ini le filesystem utilisé (LITTLEFS dans notre cas).

```
[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32doit-devkit-v1
framework = arduino
monitor_speed = 115200
lib_deps =
    https://github.com/me-no-dev/ESPAsyncWebServer.git
    lorol/LittleFS_esp32@^1.0.6
    links2004/WebSockets@^2.3.7
    adafruit/DHT sensor library@^1.4.4
    adafruit/Adafruit SSD1306@^2.5.7
    adafruit/Adafruit GFX Library@^1.11.3
    bblanchon/ArduinoJson@^6.19.4
board_build.filesystem = littlefs
```

Chargement de fichiers au serveur

3 fichiers devront être chargés sur notre serveur local

- Index.html
- Index.css
- Index.js

```
// Implémentation de litlle FS
if(!LITTLEFS.begin(true)){
Serial.println("An Error has occurred while mounting LITTLEFS");
}
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
request->send(LITTLEFS, "/index.html","text/html",false);
});

server.on("/index.css", HTTP_GET, [](AsyncWebServerRequest *request){
request->send(LITTLEFS,"/index.css","text/css",false);
});

server.on("/index.js", HTTP_GET, [](AsyncWebServerRequest *request){
request->send(LITTLEFS,"/index.js","text/js",false);
});
```

Le but de la condition est de pousser nos fichiers dans notre serveur à l'aide de **LITTLEFS** qui sera donné en premier argument, le deuxième argument est le nom du fichier et le troisième est le type de donnée utilisée dans le fichier.

Langages de programmation utilisés

HTML

Le langage HTML est utilisé pour construire le contenu d'une page web.

Voir le github

JAVASCRIPT

Le JAVASCRIPT permet de créer des interactions avec les éléments de notre page WEB c'est aussi via javascript qu'on envoie et reçoit les données du websocket.

Voir le github

CSS

CSS est un langage de mise en forme des documents.

Voir le github

C++

Le C++ est un langage de programmation : il sert donc **à écrire des applications informatiques**. Il s'agit d'ailleurs d'un des langages de programmation les plus utilisé aujourd'hui.

Code source avec commentaires

```
1: #include <Arduino.h>
2: #include <SPI.h>
3: #include <Wire.h>
4: #include "FS.h"
5: #include <LittleFS.h>
6: #include <Adafruit_Sensor.h>
7: #include <Adafruit_GFX.h>
8: #include <Adafruit_SSD1306.h>
9: #include <DHT.h>
10: #include <DHT_U.h>
11: #include <WiFi.h>
12: #include <AsyncTCP.h>
13: #include <ESPAsyncWebServer.h>
14: #include <WebSocketsServer.h>
15: #include <ArduinoJson.h>
16:
17: // DHT
18: #define DHT22PIN 26
19: DHT dht(DHT22PIN, DHT22);
```

```
20:
21: //Oled
22: #define SCREEN_WIDTH 128
23: #define SCREEN_HEIGHT 64
24: #define OLED_RESET     -1
25: Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);
26:
27: // Variables pour la connexion wifi
28: const char* ssid = "NONE";
29: const char* password = "NONE";
30: const char* PARAM_MESSAGE = "message";
31:
32: // Variables Get
33: String inputMessage = "";
34: const char* PARAM_INPUT_1 = "temperature";
35: const char* PARAM_INPUT_2 = "humi";
36: const char* PARAM_INPUT_3 = "valeur_bp";
37:
38: // Variables pour le client
39: #define BP1 12
40: #define LED_Statut 32
41: #define LED_Client 2
42: float humi;
43: float temperature;
44:
45: String valeur_bp = "OFF";
46: String JSONtxt ="";
47: String valeur_LED_S ="OFF";
48:
49: AsyncWebServer server(80);
50: WebSocketsServer webSocket = WebSocketsServer(81);
51:
52: void affichage_oled(){
53: if (digitalRead(BP1) == 1)
54:     {
55:       valeur_bp = "ON";
56:     }
57:   else {
58:      valeur_bp ="OFF";
59:   }
60: display.clearDisplay();
61: display.setTextSize(1);
62: display.setTextColor(WHITE);
63: display.setCursor(5, 25);
64: display.println(WiFi.localIP());
65: display.setCursor(5,35);
66: display.println("BP:"+valeur_bp);
67: display.setCursor(5,45);
68: display.println("LED:"+valeur_LED_S);
69: display.setCursor(5,55);
70: display.println("T:"+String(temperature)+"C"+" H:"+String(humi)+"%");
71: display.display();
72: }
73:
74: void notFound(AsyncWebServerRequest *request) {
75:     request->send(404, "text/plain", "Not found");
```

```
76: }
77:
78: // Evenement du Websocket server
79: void webSocketEvent(uint8_t num, WStype_t type, uint8_t *payload,
size_t welength)
80: {
81:   switch(type) {
82:         case WStype_DISCONNECTED:
83:             Serial.println("[WSc] Disconnected!\n");
84:             if (webSocket.connectedClients()==0)
85:               digitalWrite(LED_Client,LOW);
86:               Serial.print("pas de clients connectés");
87:             break;
88:         case WStype_CONNECTED:
89:             Serial.printf("[WSc] Connected to url: %s\n", payload);
90:             digitalWrite(LED_Client,HIGH);
91:             // send message to server when Connected
92:             webSocket.broadcastTXT("Connected");
93:             break;
94:   }
95:   String Donnees_recue = (const char *)payload;
96:   Serial.print("Données recue = ");
97:   Serial.println(Donnees_recue);
98:
99: // Reception des données du client
100:   if(type == WStype_TEXT) {
101:     byte separator=Donnees_recue.indexOf('=');
102:     String var = Donnees_recue.substring(0,separator);
103:     Serial.print("var= ");
104:     Serial.println(var);
105:     String val = Donnees_recue.substring(separator+1);
106:     Serial.print("val= ");
107:     Serial.println(val);
108:     Serial.println(" ");
109:
110:     if(var == "LEDonoff"){
111:       if(val == "ON"){
112:         digitalWrite(LED_Statut,HIGH);
113:          valeur_LED_S ="ON";}
114:       else{
115:           digitalWrite(LED_Statut,LOW);
116:           valeur_LED_S="OFF";}
117:     }
118: }
119: }
120:
121: void setup() {
122:   Serial.begin(115200);
123:
124:   pinMode(BP1, INPUT_PULLDOWN);
125:   pinMode(LED_Statut, OUTPUT);
126:   pinMode(LED_Client, OUTPUT);
127:   digitalWrite(LED_Statut, LOW);
128:   digitalWrite(LED_Client, LOW);
129:
130:   // Connexion Wifi
131:   WiFi.mode(WIFI_STA);
```

```
132:   WiFi.begin(ssid, password);
133:   if (WiFi.waitForConnectResult() != WL_CONNECTED) {
134:     Serial.printf("WiFi Failed!\n");
135:     return;
136:   }
137:
138:   Serial.print("IP Address: ");
139:   Serial.println(WiFi.localIP());
140:   dht.begin();
141:
142:   // Implémentation de litlle FS
143:   if(!LITTLEFS.begin(true)){
144:   Serial.println("An Error has occurred while mounting LITTLEFS");
145:   }
146:   server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
147:   request->send(LITTLEFS, "/index.html","text/html",false);
148:   });
149:
150:   server.on("/index.css", HTTP_GET, [](AsyncWebServerRequest
*request){
151:   request->send(LITTLEFS,"/index.css","text/css",false);
152:   });
153:
154:   server.on("/index.js", HTTP_GET, [](AsyncWebServerRequest *request){
155:   request->send(LITTLEFS,"/index.js","text/js",false);
156:   });
157:
158:   // Get request
159:   server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request) {
160:     if (request->hasParam(PARAM_INPUT_1)) {
161:       inputMessage = request->getParam(PARAM_INPUT_1)->value();
162:       Serial.println("Get request:"+inputMessage);
163:       request->send(200, "text/html", String(temperature));
164:     }
165:     if (request->hasParam(PARAM_INPUT_2)) {
166:       inputMessage = request->getParam(PARAM_INPUT_2)->value();
167:       Serial.println("Get request:"+inputMessage);
168:       request->send(200, "text/html", String(humi));
169:     }
170:     if (request->hasParam(PARAM_INPUT_3)) {
171:       inputMessage = request->getParam(PARAM_INPUT_3)->value();
172:       Serial.println("Get request:"+inputMessage);
173:       request->send(200, "text/html", String(valeur_bp));
174:     }
175:   });
176:
177:  // Post request
178:   server.on("/post", HTTP_POST, [](AsyncWebServerRequest *request){
179:     String message;
180:     if (request->hasParam(PARAM_MESSAGE)) {
181:        message = request->getParam(PARAM_MESSAGE)->value();
182:       }
183:     else {message = "No message sent";}
184:     request->send(200, "text/plain", "Hello, POST: " + message);
185:     });
186:
187:     server.onNotFound(notFound);
```

```
188:      server.begin();
189:      webSocket.begin();
190:      webSocket.onEvent(webSocketEvent);
191:      if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D
Pour 128x64
192:        Serial.println(F("SSD1306 allocation failed"));
193:      }
194:
195: }
196:
197: void loop() {
198:    webSocket.loop();
199:    humi = dht.readHumidity();
200:    temperature = dht.readTemperature();
201:    affichage_oled();
202:    JSONtxt =
"{\"humi\":\""+String(humi)+"\",\"temperature\":\""+String(temperature)+"\"
,\"valeur_bp\":\""+valeur_bp+"\"}";
203:    webSocket.broadcastTXT(JSONtxt);
204:    delay(500);
205: }
```

Analyse du code source

Déclaration des variables globales et des objets de classes.

Dans cette partie nous déclarons nos variables globales pour la connexion wifi et les requêtes post et get, voir **ligne 27 à 47**.

Ensuite nous déclarons les objets de classes pour le dht22, l'écran Oled, le websocket et le serveur.

Fonction Setup

Cette fonction nous permet d'initialiser les variables, le sens des broches et les librairies utilisées.

La connexion au wifi se fait de la ligne **131 à 139.**

L'implémentation des fichiers à pousser dans notre serveur via **LITTLEFS** se fait de la **ligne 143 à 156.**

Le paramétrage de la fonction Get et post requête se fait de la **ligne 158 à 185.**

Pour le reste nous activons les objets de classe suivant : DHT22, Server, Websocket, Oled et Serial.

Fonctions

Affichage_oled()

Cette fonction a pour but d'afficher la température, l'humidité, l'état du bouton, l'état de la led et l'adresse IP du serveur voire **ligne 52 à 72.**

notFound()

Cette fonction est appelée quand le client fait appel à une requête non existante dans le serveur. La fonction renvoi une page avec comme texte « Not found ».
Voire **ligne 74 à 76**

webSocketEvent()

La fonction websocketevent est appelée lorsqu'un évènement se produit.

Nous pouvons alors vérifier si un client est toujours connecté au serveur et allumer une led en fonction de celui-ci voire **ligne 82 à 93.**

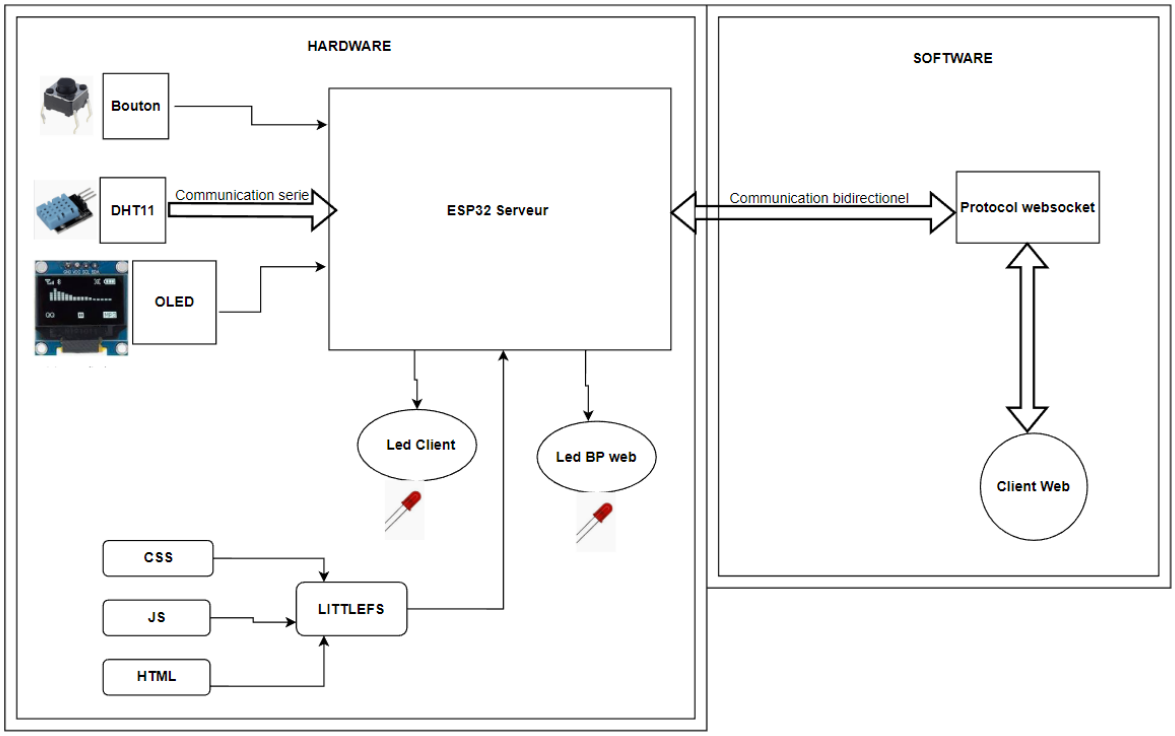
La réception des données se fait de la **ligne 100 à 108.**

Une condition permet de regarder si le switch du serveur a été activé et contrôler la led en fonction de celui-ci. Voir **ligne 110 à 116.**

Fonction Loop

Dans la fonction loop, nous envoyons les données du DHT22 et la valeur de bouton avec la méthode broadcastTXT().

Lien software hardware



Conclusion

Pour conclure, un serveur à bien été créer et modeler grâce au langage de programmation CSS, HTML et JavaScript. Ce serveur web communique avec notre ESP 32 de façon à envoyer ou recevoir des données.

E S P 3 2
Thermostat DHT11

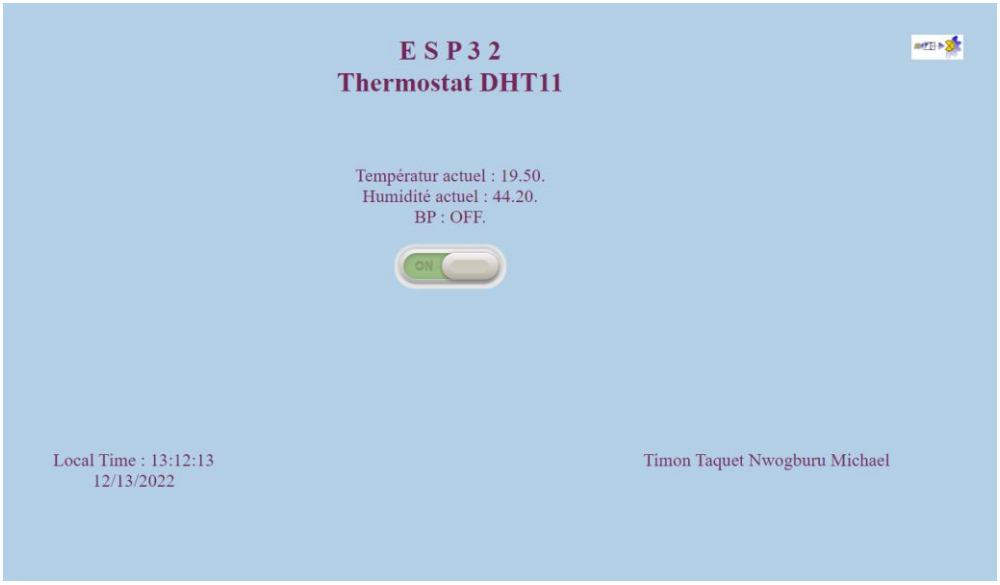
Température actuel : 19.50.
Humidité actuel : 43.90.
BP : OFF.

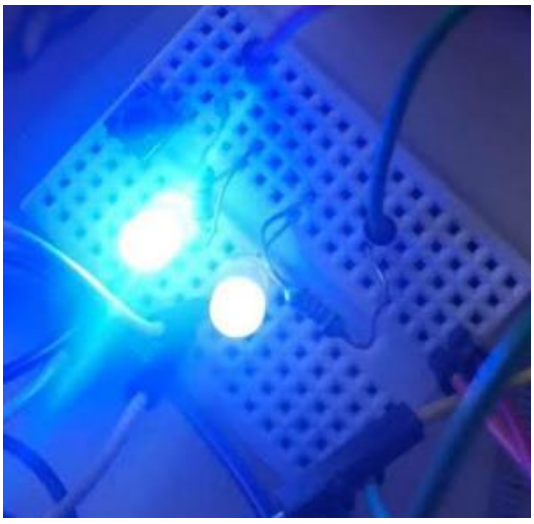
OFF

Local Time : 13:09:13
12/13/2022

Timon Taquet Nwogburu Michael

Bouton web activé





3) TP3_Google_Sheet

Objectif du TP

L'objectif de ce Tp est d'envoyer la température , l'humidité et la luminosité captées grâce à un DHT11 et un LDR vers une feuille de calcul « Google Sheets » lors de l'appuie sur un bouton poussoir. Une LED devra s'illuminer pendant un temps de 100ms lorsqu'il y a eu une transmission de données.

Les 3 valeurs données par le LDR et le DHT11 devront également être affichées sous forme de graphe dans une deuxième page Google et sur l'application PowerBi.

Introduction

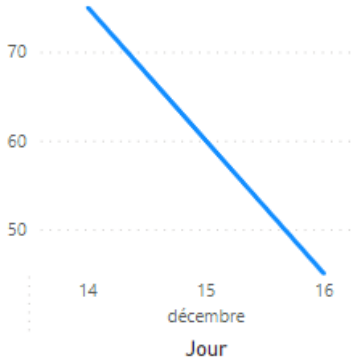
Google Sheets

Google sheets est une base de données et une extension de google dans laquelle nous pouvons introduire des données, faires des opérations sur celles-ci ou les transformer. Il est possible de collaborer avec d'autres utilisateurs pour que plusieurs personnes agissent sur cette feuille de calcul.

PowerBI Desktop

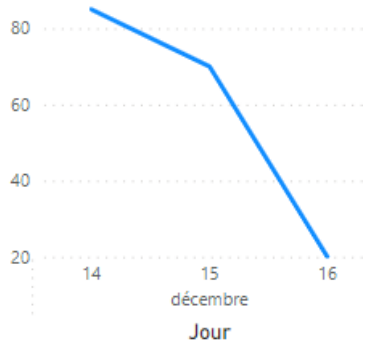
Microsoft Power Bi est une application qui permet de créer des « DashBoards » dans lesquelles nous pourront se connecter à une base de données pour les données et les visualiser ou encore les transformer. Nous avons importé notre base de données (Google Sheets) pour ensuite créer des graphique via nos données.

Premier graphique pour l'humidité en fonction du jours et du mois :



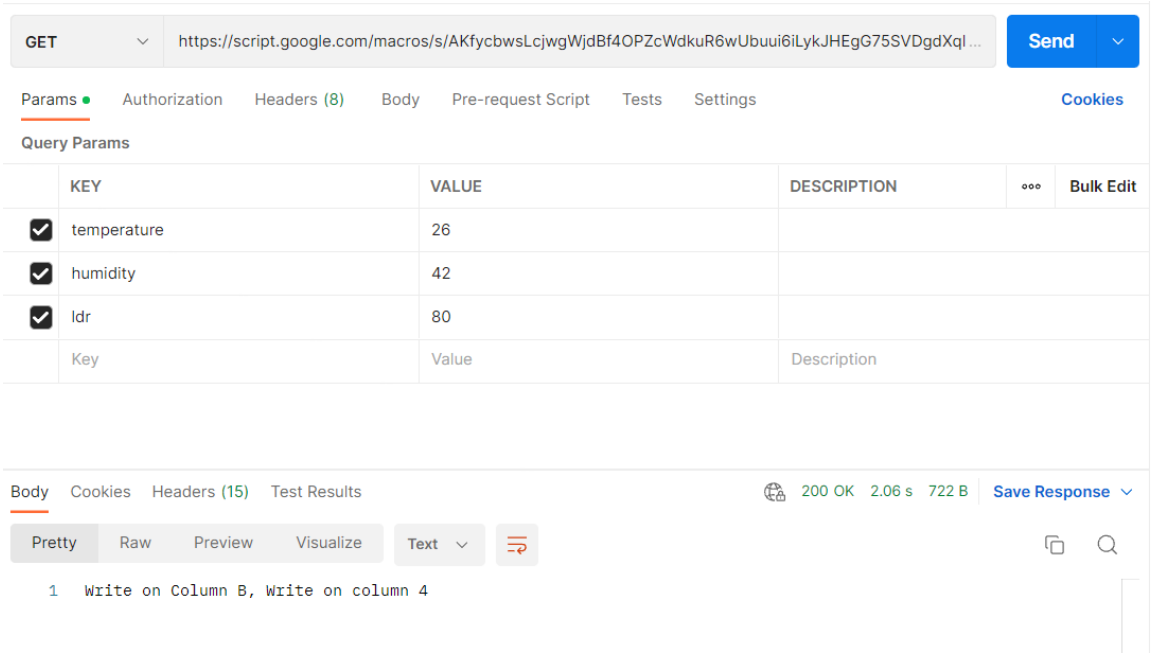
Le Deuxième graphique est pour la température en fonction du jours et du mois :

Et le dernier graphique est pour la luminosité en fonction du jours et du mois :



Postman

Encore une fois, nous pouvons utiliser Postman. Dans notre cas , nous l'utiliserons pour envoyer des données directement sur la feuille de calcul via un GET requête dans laquelle nous écrirons nos données dans l'URL du google script. Dans l'exemple , nous envoyons une température (26), un taux d'humidité (42) et la luminosité (80).



Ce qui nous donne sur notre feuille Google Sheets :

A	B	C	D
14/12/2022	26	42	80

Code source avec commentaires

```
1: #include <WiFi.h>
2: #include <HTTPClient.h>
3: #include <FS.h>
4: #include <DHT.h>
5: #include <DHT_U.h>
6: #define DHT11PIN 26
7: #define LED 13
8: DHT dht(DHT11PIN, DHT11);
9: // Variable LDR, BP
10:
11: #define LDR 34
12: #define BP_transmission 12
13: // prototype
14:
15: void sendData(String params);
```

```
16:
17: //parametre de connexion
18: const char *ssid = "LAPTOP_T";
19: const char *password = "TIMON123";
20: String GOOGLE_SCRIPT_ID =
"AKfycbxFqOgb6P3U4orcbAAxvYVCAY2hYx_XN869PfheWkXPuF3hAu6JwWQ6zagwkDx3HQh-";
21:
22: const char * root_ca=\
23: "-----BEGIN CERTIFICATE-----\n" \
24: "MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAwVzELMAkG\n" \
25: "A1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNVBAsTB1Jv\n" \
26: "b3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5MDExMjAw\n" \
27: "MDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9i\n" \
28: "YWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxT\n" \
29: "aWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDaDuaZ\n" \
30: "jc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6arymAZavp\n" \
31: "xy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCOXkNz8kHp\n" \
32: "1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6TRGHRjcdG\n" \
33: "snUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux2x8gkasJ\n" \
34: "U26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlDSgeF59N8\n" \
35: "9iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMBAf8E\n" \
36: "BTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkqhkiG9w0B\n" \
37: "AQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoRSLblCKOz\n" \
38: "yj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQLcFGUl5gE\n" \
39: "38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrglfCm7ymP\n" \
40: "AbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr+WymXUad\n" \
41: "DKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XSQRjbgbME\n" \
42: "HMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4A==\n" \
43: "-----END CERTIFICATE-----\n";
44:
45:
46: WiFiClientSecure client;
47:
48: void setup() {
49:   Serial.begin(115200);
50:   pinMode(LED, OUTPUT);
51:   pinMode(BP_transmission, INPUT_PULLDOWN);
52:   digitalWrite(LED,LOW);
53:   delay(10);
54:   dht.begin();
55:   WiFi.mode(WIFI_STA);
56:   WiFi.begin(ssid, password);
57:   Serial.println("Started");
58:   Serial.print("Connecting");
59:   while (WiFi.status() != WL_CONNECTED) {
60:     delay(500);
61:     Serial.print(".");
62:   }
63:   randomSeed(analogRead(0));
64:   Serial.println("TP3 GoogleSheets ready...");
65: }
66:
67: unsigned long lastTime = 0;
68: unsigned long timerDelay = 10000;
69: String strTemp, strHum, strldr, strParameter;
70: //fonction pour envoyer les donnes sur google sheets
71: void sendData(String params) {
```

```
72:
73:   HTTPClient http;
74:   String
url="https://script.google.com/macros/s/"+GOOGLE_SCRIPT_ID+"/exec?"+params;
75:   Serial.println(url);
76:   Serial.println("Making a request");
77:   // Your Domain name with URL path or IP address with path
78:   http.begin(url, root_ca); //Specify the URL and certificate
79:   // Send HTTP GET request
80:   int httpCode = http.GET();
81:   if (httpCode > 0) {
82:     Serial.print("HTTP Response code: ");
83:     Serial.println(httpCode);
84:     String payload = http.getString();
85:     Serial.println(payload);
86:   }
87:   else {
88:     Serial.print("Error code: ");
89:     Serial.println(httpCode);
90:   }
91:   // Free resources
92:   http.end();
93: }
94: void loop() {
95:   //Send an HTTP POST request every delay
96:
97:     //Check WiFi connection status
98:     //appuie sur le bouton pour envoyer les donnees
99:     if(WiFi.status()== WL_CONNECTED && digitalRead(BP_transmission) ==
HIGH){
100:       strTemp = String(dht.readTemperature());
101:       strTemp.replace(".",",");
102:       strHum = String(dht.readHumidity());
103:       strHum.replace(".",",");
104:       strldr= String(map(analogRead(LDR),0,4095,0,100));
105:       Serial.println(strTemp);
106:       Serial.println(strHum);
107:       strParameter = "temperature=" + strTemp + "&humidity=" + strHum
+ "&ldr=" + strldr ;
108:       Serial.println(analogRead(LDR));
109:       sendData(strParameter);
110:       digitalWrite(LED,HIGH);
111:       delay(100);
112:       digitalWrite(LED,LOW);
113:     }
114:
115:     lastTime = millis();
116:   }
117:
118:
119:
```

Analyse du code source

Partie connexion :

La première étape est d'établir une connexion entre notre ESP32 et la feuille de calcul. Pour ce faire nous devrons prendre l' « ID » du Google script (**Ligne 20**) pour l'utiliser lors de la connexion . Nous aurons également besoin de notre certificat « root_ca » qui est un certificat de sécurité nécessaire si nous voulons établir une bonne connexion avec google sheets.

Déclaration des variables :

Tout d'abord , nous déclarons les pins du matériel (Hardware) utilisé pour la réussite de ce TP (**Lignes 6 à 12**) , ensuite, nous créons la variable « root_ca » dans lequel se trouve le root_ca trouvé sur notre page google sheets. Enfin ,les variables qui contiendrons les données à envoyer à notre feuille de calcul sont déclarées à la ligne 69.

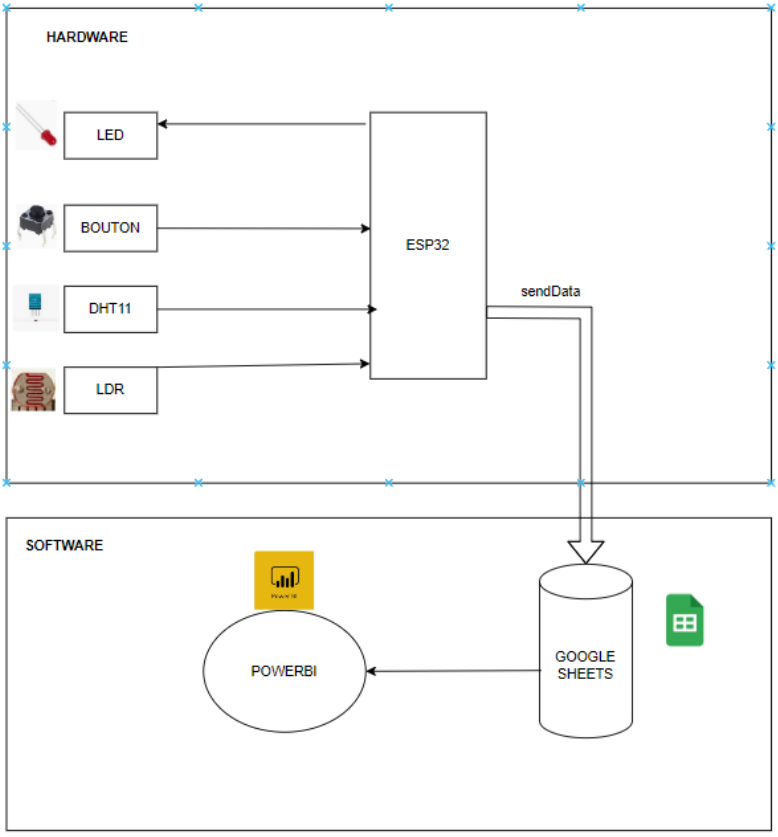
Fonction importante :

La fonction principale du code est la fonction « sendData » (**Ligne 71**) qui a pour rôle de se connecter à notre feuille de calcul via l'URL et le certificat « root_ca » (**Ligne 78**) et d'envoyer nos données sous format de String vers la feuille de calcul (**Ligne 74**). Ces données doivent être misent en argument lors de l'appel de la fonction.

Le fonctionnement principale :

Dans notre boucle de fonctionnement , nous allons faire une série d'actions si une connexion à bien été établie et que nous avons appuyé sur un bouton poussoir (**Ligne 99**). Dans cette série d'actions , nous allons prendre la température , l'humidité et la luminosité pour les stocker dans 3 variables différentes et les afficher dans la console (**Lignes 100 à 106**). Ensuite nous allons stocker ces variables dans une seule variable « strParameter » (**Ligne 107**) pour appeler la fonction « sendData » qui aura comme argument notre nouvelle variable (**Ligne 109**) . Grâce à cela, nos données seront envoyées sur la feuille de calcul. Enfin, nous allumons une LED pour une durée de 100 ms qui nous permettra de savoir qu'une transmission de données à bien été faite.

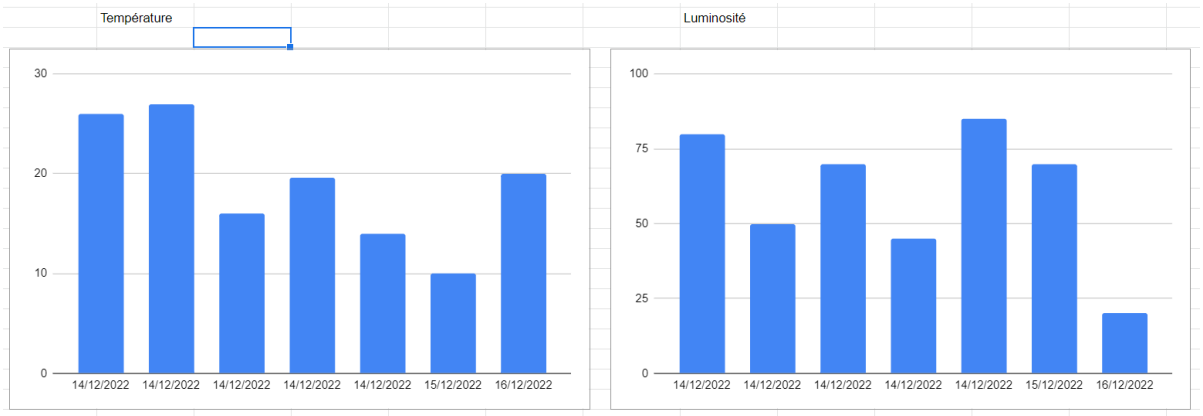
Lien software hardware

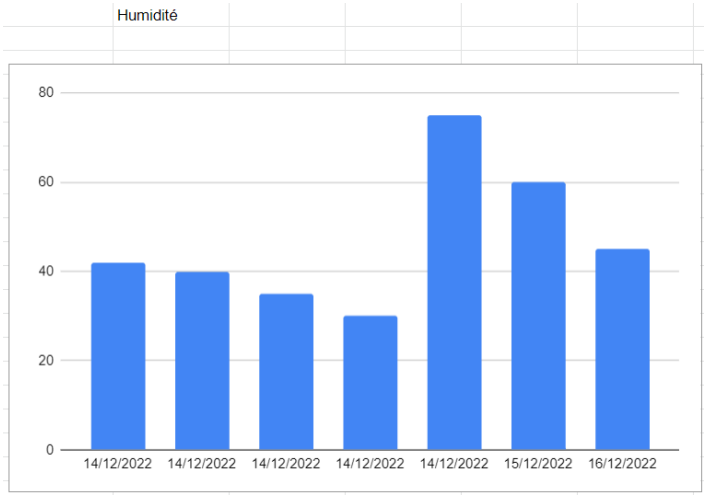


Conclusion

Pour conclure, les données du DHT11 et du LDR sont envoyées et directement notées dans la feuille de calcul Google Sheets lors de l'appuie sur le bouton poussoir. Ces données sont visualisées sous forme de graphe directement dans la feuille ou sur l'application POWERBI.

Graphes sur Google Sheets :





Feuille de calcul

14/12/2022	26	42	80
14/12/2022	27	40	50
14/12/2022	16	35,00	70
14/12/2022	19,6	30,00	45
14/12/2022	14	75,00	85
15/12/2022	10	60	70
16/12/2022	20	45	20

4) TP4_Adafruit_Io

Objectif du TP

L'objectif du TP4 est de nous familiariser avec un Dashboard de l'environnement ADAFRUIT_IO.
Avec le protocole MQTT comme nouvelle méthode de communication.

Nous devrons commencer par afficher 3 jauges et 3 graphes pour visualiser les variations des grandeurs suivantes :

- Température
- Humidité
- Luminosité

Un bouton on/off pour commander une LED suivie d'une zone de texte qui affichera bien l'état actuel de la LED.

Et pour finir, 3 glissières pour faire varier les couleurs d'une LED RGB Neopixel.

Pour ce faire nous allons utiliser les composants suivants :

- DHT22
- Bouton
- LDR
- Neopixel

Introduction

PROTOCOLE MQTT

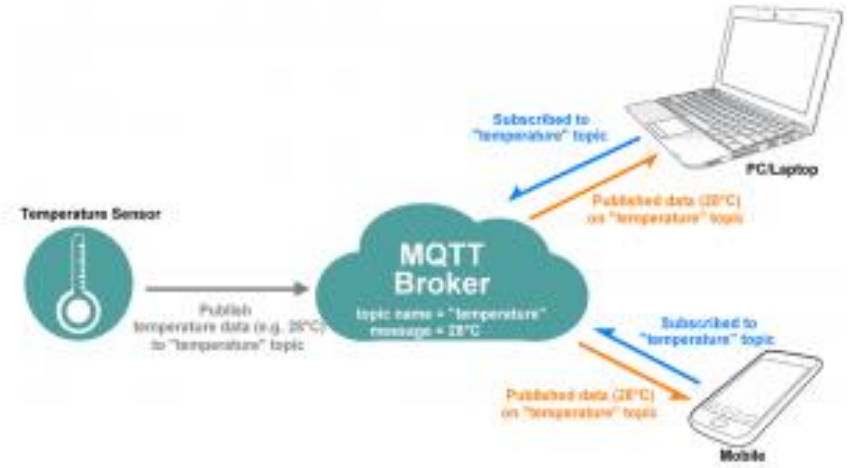
Le protocole MQTT qui signifie Message Queuing Telemetry Transport est un protocole qui a été créé pour l'Internet des objets.
La télémétrie est extrêmement légère et a été conçue pour les appareils contraints et avec une faible bande passante.

Pour pouvoir communiquer avec le protocole de type MQTT il faut d'abord avoir les éléments suivants :

- **Broker :** c'est un serveur qui distribue et qui reçois les informations au clients intéressés.
- **Client :** Les clients sont les appareils qui se connectent au broker par exemple pc, esp32, smartphone.

- **Topic :** Le topic est le sujet sur lequel les clients pourront souscrire des informations ou bien en publier.

Après avoir implémenté le broker, client et le topic, le client peut alors envoyer des informations à un certain **Topic** avec la fonction **Publish** et peut aussi en souscrire avec la fonction **Subscribe.**



ADAFRUIT_IO

Adafruit est une entreprise fondée en 2005 par Limor Fried, une ingénieure américaine, qui se spécialise dans la vente, la production de composants électroniques et mets à disposition des codes sources pour diverses utilisations.

Adafruit_IO est un environnement WEB permettant de collecter les données des objets connectés dans des feeds et d'organiser leur représentation dans des Dashboard.

Feeds

Les Feeds, qui signifie files de données, sont en quelques sortes des variables contenant des informations modifiables, ces informations pourront être retransmises ou modifiées selon leurs utilisations.

Après avoir créé un Feed, il ne reste plus qu'à le lier à un widget de notre Dashboard.

Code source avec commentaires

```
1 #include <AdafruitIO_WiFi.h>
2 #include <Adafruit_Sensor.h>
3 #include <Adafruit_NeoPixel.h>
4 #include <DHT.h>
5 #include <DHT_U.h>
6 #include <SPI.h>
7
8 #define IO_USERNAME "Michael_Nwg"
```

```
#define IO_KEY      "aio_MiTf37vC7EtaAPboALYoJyncahkg"
#define WIFI_SSID   "none"
#define WIFI_PASS   "none"


#define RED_PIN   4
#define GREEN_PIN 5
#define BLUE_PIN  2
#define LDR 39
#define RGB_PIN 0
#define LED_PIN 2
#define DATA_PIN  26
#define Push_Button1 12

// RGB Var
int couleurRed = 0;
int couleurBlue = 0;
int couleurGreen = 0;

// LDR
int LDR_Value= 0;


AdafruitIO_WiFi io (IO_USERNAME,IO_KEY,WIFI_SSID, WIFI_PASS);
DHT_Unified dht(DATA_PIN, DHT11);
Adafruit_NeoPixel strip = Adafruit_NeoPixel(30, RGB_PIN, NEO_GRB +
NEO_KHZ800);

// set up the  feed
AdafruitIO_Feed *RGB_BLUE = io.feed("RGB_BLUE");
AdafruitIO_Feed *RGB_RED = io.feed("RGB_RED");
AdafruitIO_Feed *RGB_GREEN = io.feed("RGB_GREEN");
AdafruitIO_Feed *LED = io.feed("LED");
AdafruitIO_Feed *Temperature = io.feed("temperature");
AdafruitIO_Feed *Humidite = io.feed("humidite");
AdafruitIO_Feed *luminosite = io.feed("luminosité");
AdafruitIO_Feed *TXT_BOX = io.feed("TXT_BOX");

// Fonction de reception des données //
void handle_LED(AdafruitIO_Data *data) {

  Serial.print("received <- ");

  if(data->toPinLevel() == HIGH){
    Serial.println("HIGH");
    TXT_BOX->save("Led: ON");
    }

  else{
    Serial.println("LOW");
    TXT_BOX->save("LED: OFF");
  }

  // write the current state to the led
  digitalWrite(LED_PIN, data->toPinLevel());
}
```

```
66
67 void handleRGB_BLUE(AdafruitIO_Data *data){
68   couleurBlue = data->toInt();
69   // print RGB value
70   Serial.print("B: <-" );
71   Serial.println(couleurBlue);
72   // RGB Neopixel
73   for (int i = 0; i < strip.numPixels(); i++)
74     {
75       strip.setPixelColor(i, strip.Color(couleurRed, couleurGreen,
76 couleurBlue));
77     }
78     strip.show();
79 }
80
81 void handleRGB_GREEN(AdafruitIO_Data *data){
82   couleurGreen = data->toInt();
83   // print RGB value
84   Serial.print("G: <-");
85   Serial.println(couleurGreen);
86   // RGB Neopixel
87   for (int i = 0; i < strip.numPixels(); i++)
88     {
89       strip.setPixelColor(i, strip.Color(couleurRed, couleurGreen,
90 couleurBlue));
91     }
92     strip.show();
93 }
94
95 void handleRGB_RED(AdafruitIO_Data *data){
96   couleurRed = data->toInt();
97   // print RGB value
98   Serial.print("R: <-");
99   Serial.println(couleurRed);
100   // RGB Neopixel
101   for (int i = 0; i < strip.numPixels(); i++)
102     {
103       strip.setPixelColor(i, strip.Color(couleurRed, couleurGreen,
104 couleurBlue));
105     }
106     strip.show();
107 }
108 // Fonction d'actualisation des données
109 void Actualisation(){
110
111   sensors_event_t event;
112   dht.temperature().getEvent(&event);
113   float celsius = event.temperature; // ver
114   Serial.print("celsius: ");
115   Serial.print(celsius);
116   Serial.println("C");
117   Temperature->save(celsius);
118
119   dht.humidity().getEvent(&event);
120   Serial.print("humidité: ");
121   Serial.print(event.relative_humidity);
122   Serial.println("%");
```

```
123   Humidite->save(event.relative_humidity);
124
125   LDR_Value = analogRead(LDR);
126   Serial.print("Luminosité -> ");
127   Serial.println(LDR_Value);
128   luminosite->save(LDR_Value);
129   }
130
131 void setup() {
132
133   pinMode(LED_PIN, OUTPUT);
134   pinMode(Push_Button1, INPUT_PULLDOWN);
135   Serial.begin(115200);
136   while(!Serial);
137   dht.begin();
138   Serial.print("Connecting to Adafruit IO");
139   io.connect();
140
141
142   // Création de message CALLBACK
143   LED->onMessage(handle_LED);
144   RGB_BLUE->onMessage(handleRGB_BLUE);
145   RGB_RED->onMessage(handleRGB_RED);
146   RGB_GREEN->onMessage(handleRGB_GREEN);
147
148
149   while(io.status() < AIO_CONNECTED) {
150     Serial.print(".");
151     delay(500);
152   }
153   Serial.println();
154   Serial.println(io.statusText());
155
156
157   LED->get();
158   RGB_BLUE->get();
159   RGB_RED->get();
160   RGB_GREEN->get();
161 }
162
163 void loop() {
164   io.run();
      if (digitalRead(Push_Button1) == HIGH){
        Actualisation();
      }
    }
```

Analyse du code source

Déclaration des données de connexion au wifi et à la plateforme adafruit IO

La première partie du code consiste d'abord d'inclure toutes les librairies nécessaires pour le bon fonctionnement du TP. Ceci se fait de **la ligne 2 à 7.**

Nous avons ensuite les variables qui seront utilisées pour la connexion au wifi et la connexion à la plateforme adafruit IO.
La connexion à la plateforme adafruit IO se fait grâce à un nom d'utilisateur « IO_USERNAME » et un code « IO_KEY ». Ceci se fait de **la ligne 9 à 12.**

Déclaration des variables globales et des objets de classes

La deuxième partie du code consiste à déclarer les variables globales et à créer des instances de classes pour le DHT, ADAFRUIT_IO, Neopixel.
Ceci se fait de **la ligne 16 à 36.**

Initialisation des feeds

L'initialisation des feeds permet de faire un lien entre les feeds de la plateforme ADAFRUIT_IO et l'esp32. Après cela, nous pourrons demander ou envoyer des informations aux feeds qui seront par la suite affichés sur le Dashboard sous forme de widget. Ceci se fait de **la ligne 39 à 46.**

Fonctions des réceptions et actualisation des données

La dernière partie du code contient les différentes fonctions qui seront appelées pour la réception de données et l'actualisation de données du Dashboard.

Handle_LED()

La fonction « Handle_LED()» sera appelé lorsqu'une donnée provenant du feed LED sera disponible.
La liaison entre le feed et la fonction « Handle_LED()» se fait à **la ligne 140**.

Le feed «LED» contiendra l'état du bouton on/off du Dashboard une led sera ensuite allumé ou éteinte en fonction de l'état du feed. Pour finir, l'état de la led sera affiché sur le Dashboard dans un TEXT BOX.

HandleRGB_BLUE() , HandleRGB_GREEN() , HandleRGB_RED()

Les fonctions HandleRGB… auront pour but de mettre à jour l'état de la led RGB neopixel. elles seront appelées lorsqu'il y'aura une variation des 3 sliders sur le Dashboard.
Chaque slider correspond à une couleur bleue, vert et rouge.
Le lien entre les feeds des sliders et des fonctions se fait de **la ligne 141 à 143.**

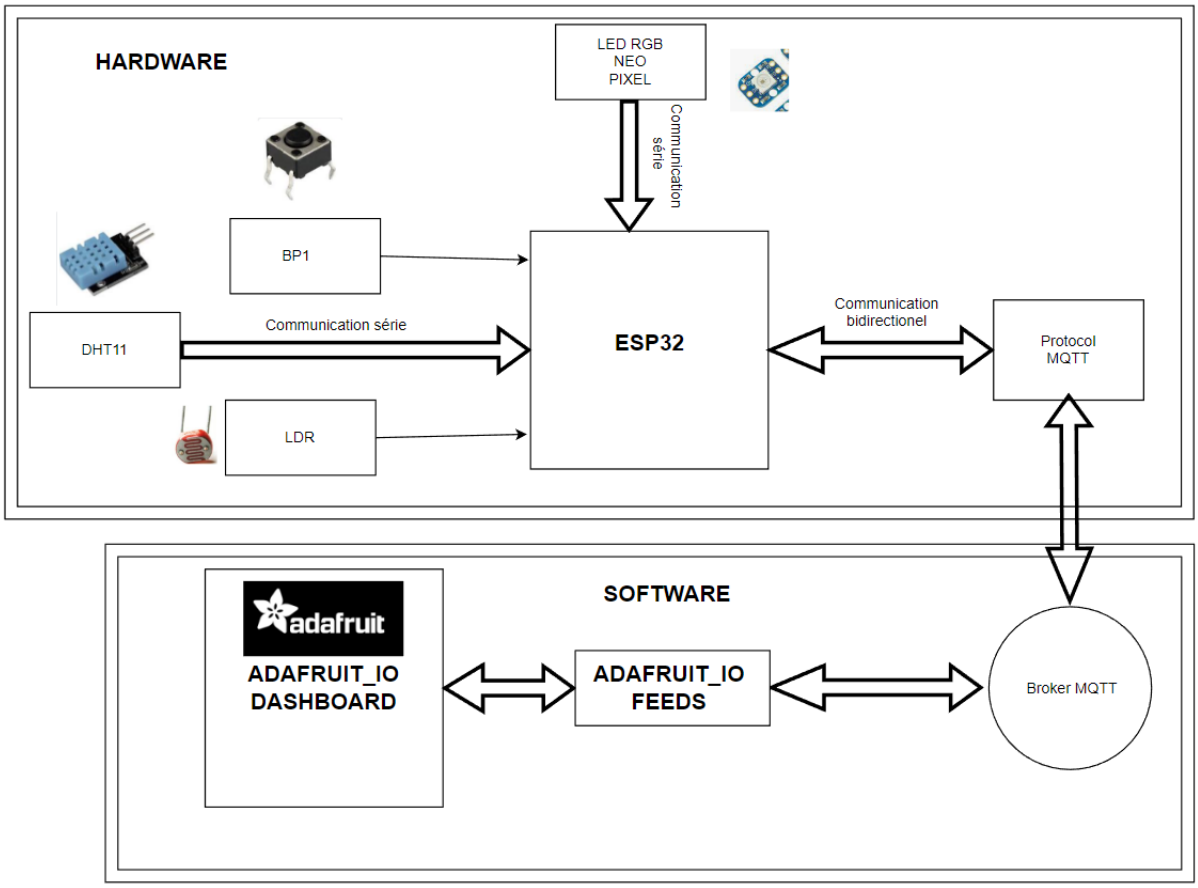
L'intensité de chaque couleur est modulable d'une valeur de 0 à 255.
La couleur de la led neopixel sera mis à jour en fonction de la donnée reçue avec la méthode «SetPixelColor » .

Actualisation ()

La fonction «Actualisation» permet de lire les valeurs reçues du DHT 11 et du LDR. Celles-ci seront par la suite envoyées aux feeds concernées, c'est-à-dire la température, l'humidité et la luminosité.

L'envoi des informations vers les feeds se fait grâce à la méthode «save » voir **la ligne 106 à 126.**

Lien software hardware



Conclusion

Pour conclure, comme montré ci-dessous, le Dashboard Adafruit permet de visualiser directement des données captées par nos capteur et à l'inverse d'envoyer des « ordres » en appuyant sur un bouton virtuel vers l'ESP32. Une communication bidirectionnelle a donc bien été établie entre des entrées et sortie « virtuels » et de composants bien réels.

Réalisation

Feeds :

New Feed | New Group

Default

Feed Name	Key	Last value	Recorded
LED	led	off	3 minutes ago
RGB_BLUE	rgb-blue	115	4 minutes ago
RGB_GREEN	rgb-green	155	4 minutes ago
RGB_RED	rgb-red	90	4 minutes ago
TXT_BOX	txt-box	LED: OFF	3 minutes ago
humidite	humidite	74	2 minutes ago
luminosité	luminosite	617	2 minutes ago
temperature	temperature	39	2 minutes ago

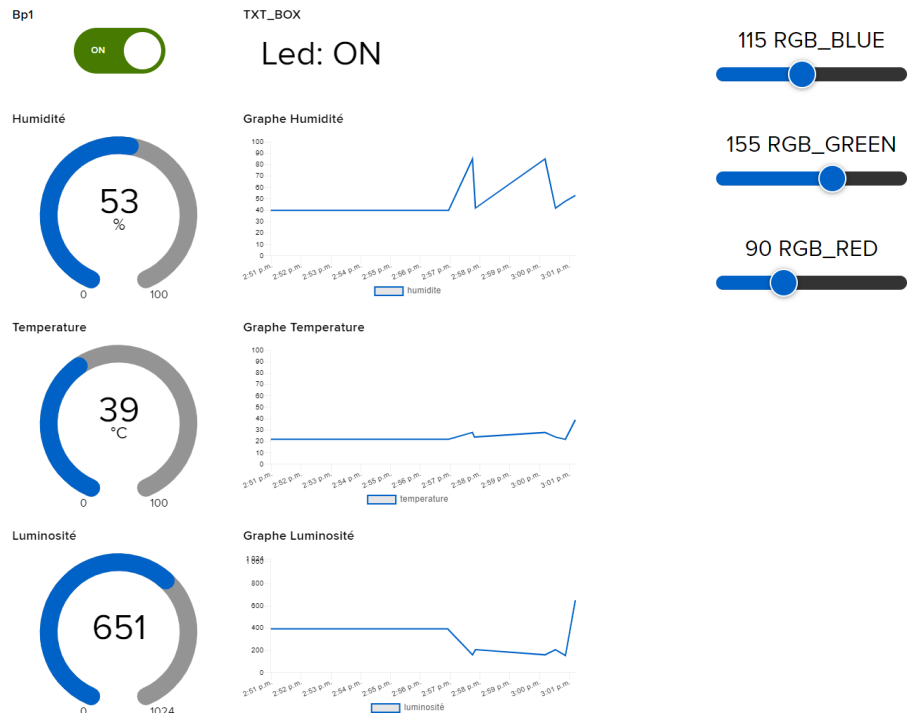
New Feed | New Group

Default

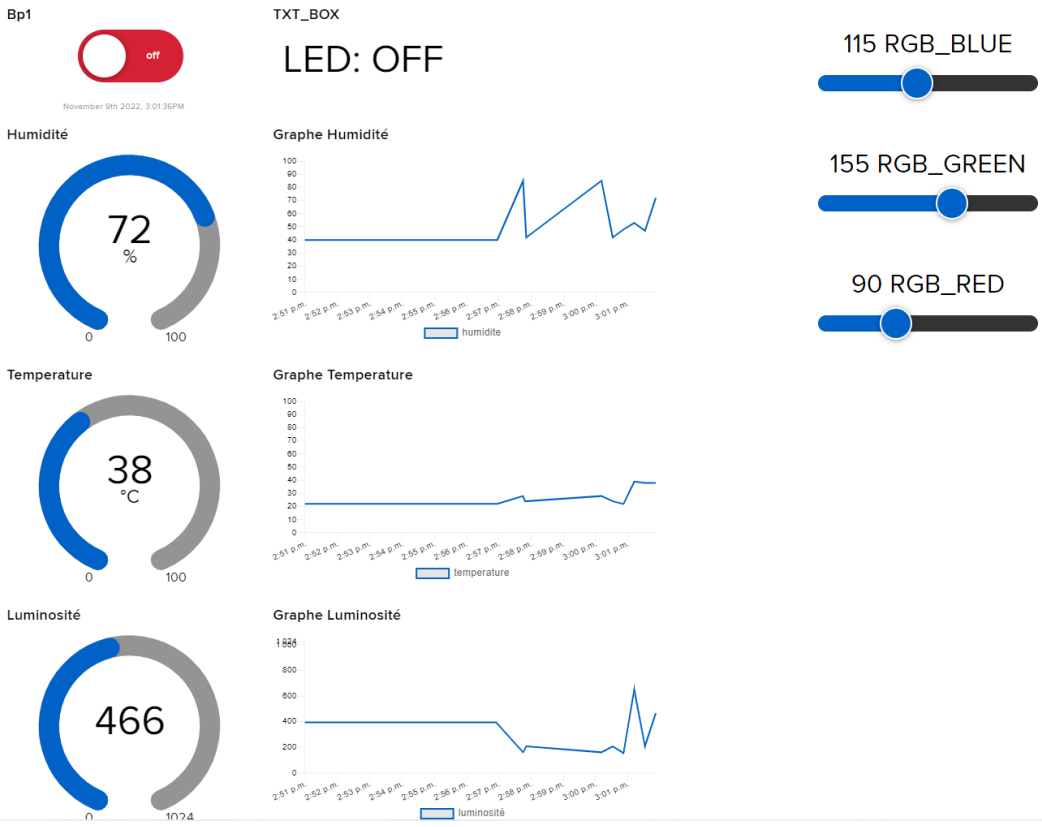
Feed Name	Key	Last value	Recorded
LED	led	1	1 minute ago
RGB_BLUE	rgb-blue	115	5 minutes ago
RGB_GREEN	rgb-green	159	less than a minute ago
RGB_RED	rgb-red	90	5 minutes ago
TXT_BOX	txt-box	Led: ON	less than a minute ago
humidite	humidite	85	less than a minute ago
luminosité	luminosite	162	less than a minute ago
temperature	temperature	28	less than a minute ago

Dashboard

Exemple avec LED ON :



Exemple avec LED OFF :



5) TP5_Blynk

Objectif du TP

L'objectif du TP4 est de nous familiariser avec un Dashboard de l'environnement Blynk à l'aide des éléments suivants :

Controllers :

- Button
- Vertical Slider

Displays :

- Labeled Value
- Led
- Gauge
- Chart

Les **Controllers** cités ci-dessus permettront d'avoir une interaction entre nos composants hardware comme la led **Neopixel RGB** avec un **Vertical Slider** et une led basic avec le **widget Button**.

Un timer est affiché sur le Dashboard avec le widget **Labeled Value**.
Le widget **Chart** devra nous permettre d'afficher la température et l'humidité sous forme de graphique qui sera actualisé tous les X temps par L'ESP.

La valeur lue par le LDR sera affichée sur le widget **Gauge** et pour finir, l'état de la led sera affichée avec le widget **Led.**

Pour ce faire nous utilisons les composants suivant :

- DHT22
- Led
- LDR
- Neopixel

Introduction

Blynk

Blynk a été conçu pour l'Internet des objets. Il peut contrôler le matériel à distance, il peut afficher les données des capteurs, stocker des données dans des datastreams, les visualiser et faire bien d'autres choses intéressantes.

Il y a trois composants principaux dans la plate-forme :

Blynk App permet de créer des interfaces pour des projets en utilisant les widgets disponibles.

Blynk Server responsable de toutes les communications entre le smartphone et le matériel.

Les bibliothèques Blynk permettent la communication avec le serveur et traitent toutes les commandes entrantes et sortantes.

Datastreams

Les datastreams sont des variables contenant des informations modifiables, ces informations pourront être retransmises ou modifiées selon leurs utilisations.

Après avoir créé un datastream, il ne reste plus qu'à le lier à un widget de notre Dashboard.

10 Datastreams

Id	Name	Alias	Color	Pin	Data Type	Units	Is Raw	Min	Max
4	Button Image	Button Image		V3	String		false		
7	Red Rgb	Red Rgb		V4	Integer		false	0	255
10	Green Rgb	Green Rgb		V6	Integer		false	0	255
9	Blue Rgb	Blue Rgb		V5	Integer		false	0	255
12	Humi	Humi		V8	Enumerable		false		
3	LDR	LDR		V2	Integer		false	0	1024
1	Switch Control	Switch Control		V0	Integer		false	0	1
2	Timer	Timer		V1	Integer		false	0	100000000
11	Temperature	Temperature		V7	Enumerable		false		
13	LED	LED		V9	Integer		false	0	1

Code source avec commentaires

```
1: // Données de connexion au serveur blynk
2: #define BLYNK_TEMPLATE_ID "TMPLdgNvmLUn"
3: #define BLYNK_DEVICE_NAME "Quickstart Template"
4: #define BLYNK_AUTH_TOKEN "BKj7d7LNUjBounCH0wzMH4x6aCAIWfgs"
5: #include <Arduino.h>
6: #include <Adafruit_NeoPixel.h>
7: #include <Adafruit_Sensor.h>
8: #include <DHT.h>
9: #include "FS.h"
10: #include <WiFi.h>
11: #include <WiFiClient.h>
12: #include <BlynkSimpleEsp32.h>
13:
14: #define BLYNK_PRINT Serial
15: char auth[] = BLYNK_AUTH_TOKEN;
16:
17: #define DHTpin 26
18: #define LDR 33
19: #define RGB_PIN 27
20: #define LED_Blynk 32
21:
22: // var couleur RGB
23: int Red = 0;
24: int Green = 0;
25: int Blue = 0;
26:
27: // Your WiFi credentials.
28: // Set password to "" for open networks.
29: char ssid[] = "none";
30: char pass[] = "none";
31:
32: BlynkTimer timer;
33: DHT dht(DHTpin, DHT22);
34: Adafruit_NeoPixel strip = Adafruit_NeoPixel(30, RGB_PIN, NEO_GRB +
NEO_KHZ800);
35:
36: int compteur = 0;
37:
38: BLYNK_WRITE(V4)
39: {
40:   Red = param.asInt();
41:   for (int i = 0; i < strip.numPixels(); i++)
42:   {
43:     strip.setPixelColor(i, strip.Color(Red, Green, Blue));
44:   }
45:   strip.show();
46: }
47:
48: BLYNK_WRITE(V6)
49: {
50:   Green = param.asInt();
51:   for (int i = 0; i < strip.numPixels(); i++)
```

```
52:   {
53:     strip.setPixelColor(i, strip.Color(Red, Green, Blue));
54:   }
55:   strip.show();
56: }
57:
58: BLYNK_WRITE(V5)
59: {
60:   Blue = param.asInt();
61:   for (int i = 0; i < strip.numPixels(); i++)
62:   {
63:     strip.setPixelColor(i, strip.Color(Red, Green, Blue));
64:   }
65:   strip.show();
66: }
67:
68: // This function is called every time the Virtual Pin 0 state changes
69: BLYNK_WRITE(V0)
70:   {
71:   // Set incoming value from pin V0 to a variable
72:   int value = param.asInt();
73:   digitalWrite(LED_Blynk,value);
74:   Blynk.virtualWrite(V9, value);
75:   }
76:
77: void Actualisation(){
78:   Blynk.virtualWrite(V2,analogRead(LDR));
79:   float celsius = dht.readTemperature();
80:   float humi = dht.readHumidity();
81:   Blynk.virtualWrite(V7, celsius);
82:   Blynk.virtualWrite(V8, humi);
83: }
84:
85:
86: // This function is called every time the device is connected to the
Blynk.Cloud
87: BLYNK_CONNECTED()
88: {
89:   // Change Web Link Button message to "Congratulations!"
90:   Blynk.setProperty(V3, "offImageUrl", "https://static-
image.nyc3.cdn.digitaloceanspaces.com/general/fte/congratulations.png");
91:   Blynk.setProperty(V3, "onImageUrl",  "https://static-
image.nyc3.cdn.digitaloceanspaces.com/general/fte/congratulations_pressed.p
ng");
92:   Blynk.setProperty(V3, "url", "https://docs.blynk.io/en/getting-
started/what-do-i-need-to-blynk/how-quickstart-device-was-made");
93: }
94:
95: // This function sends Arduino's uptime every second to Virtual Pin 2.
96: void myTimerEvent()
97: {
98:   // You can send any value at any time.
99:   // Please don't send more that 10 values per second.
100:    Blynk.virtualWrite(V1, millis() / 1000);
101:    Actualisation();
102: }
103:
```

```
104: void setup()
105: {
106:   pinMode(LED_Blynk, OUTPUT);
107:   pinMode(LDR,INPUT);
108:   digitalWrite(LED_Blynk, LOW);
109:   // Debug console
110:   Serial.begin(115200);
111:   dht.begin();
112:   Blynk.begin(auth, ssid, pass);
113:   // You can also specify server:
114:   //Blynk.begin(auth, ssid, pass, "blynk.cloud", 80);
115:   //Blynk.begin(auth, ssid, pass, IPAddress(192,168,1,100), 8080);
116:
117:   // Setup a function to be called every second
118:   timer.setInterval(1000L, myTimerEvent);
119:   strip.begin();
120:   strip.show();
121: }
122:
123: void loop()
124: {
125:   Blynk.run();
126:   timer.run();
127: }
128:
129:
130:
131:
```

Analyse du code source

Connexion à la plateforme BLYNK

Pour se connecter à un Dashboard blynk, il faut avoir le **TOKEN**, **NOM du device** et l'**ID** du Dashboard.

Ces informations sont retransmises à notre ESP32 de **la ligne 2 à 4.**

Pour terminer, une connexion wifi sera nécessaire pour se connecter au serveur blynk.

Déclaration des variables globales et des objets de classes.

Dans cette partie nous déclarons nos variables globales pour les couleurs du RGB voir la **ligne 23 à 25.**

Ensuite nous déclarons les objets de classes pour le dht22, la led neopixel et le timer blynk.

Voir la **ligne 32 à 34.**

Fonctions des réceptions et actualisation des données

Fonctions réceptions

Les fonctions BLYNK_WRITE (V…) sont appelées après chaque modification des datastream suivant : V0, V4, V5, V6.

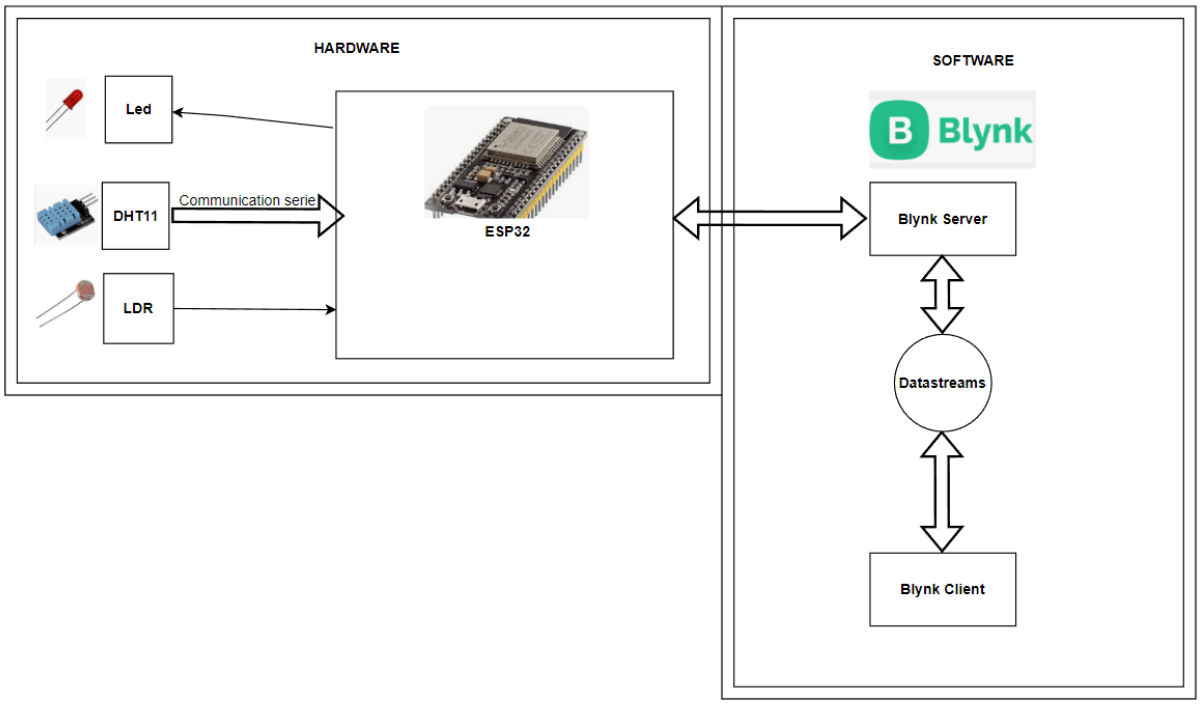
Ces fonctions permettent d'actualiser l'état de la neopixel et de la led avec la donnée reçue en paramètre. Par exemple **: Blue = param.asInt()**
Voir **la ligne 60.**

Fonction d'actualisation

La fonction « Actualisation » permet de lire les valeurs reçues du DHT22 et du LDR pour les envoyer vers leur datastreams concerné.

L'envoi des informations aux datastreams se fait grâce à la méthode
« virtualWrite(datasteam, variables)» voir **la ligne 77 à 82.**

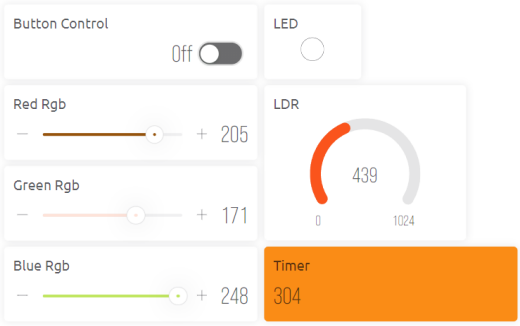
Lien software hardware

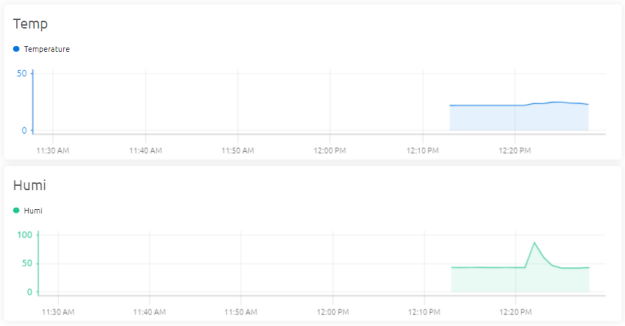


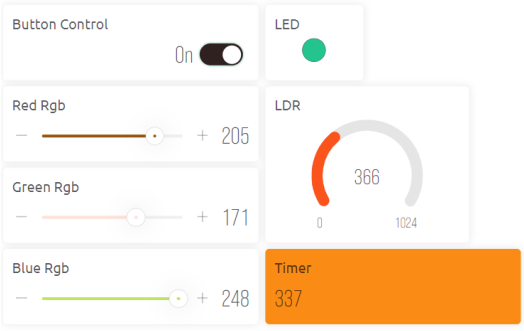
Conclusion

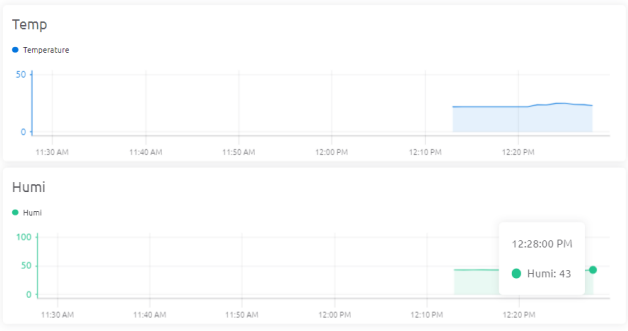
Pour conclure, une véritable interaction entre le Dashboard « Blynk » et l'ESP32 est présente. Dans un sens, Blynk envoie des données vers le côté Hardware pour allumer une LED lors de l'appuie sur un bouton « Virtuel » et à l'inverse notre ESP envoie des données qui sont directement affichées dans le Dashboard.

Réalisation









6) TP6_Nodered

Objectif du TP

L'objectif de ce TP est de créer un Dashboard sur l'application « Node-Red » où des données seront reçues et d'autre envoyées vers notre ESP32. Pour établir une communication , nous utiliserons le protocole MQTT dont le serveur sera stocké sur un docker.

D'un côté, notre ESP enverra la température, l'humidité et la luminosité vers Node-Red , ensuite ces données seront stockées sur une base de données appelée « MariaDB » sous forme d'historique. Et toujours ces mêmes données seront affichées sous forme de graphe sur notre Dashboard.

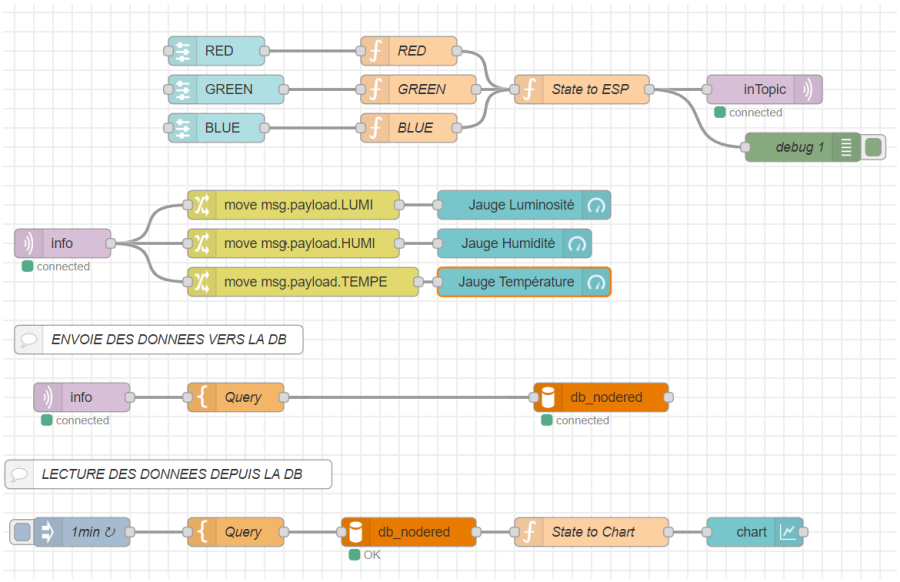
D'un autre côté, trois sliders allant de 0 à 255 sont visibles sur le Dashboard , lorsque ceux-ci sont modifiés, alors les valeurs de ces chiffres sont envoyées vers l'ESP. Ces 3 chiffres sont les paramètres de notre RGB. Nous pourrons donc changer la couleur du RGB Neopixel en jouant avec les sliders.

Introduction

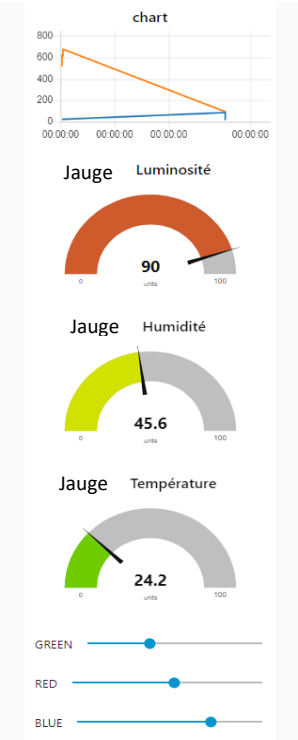
Node-Red

Node-Red est une application Web pour traiter ou visualiser des données. Il est possible , à l'aide de diverses palettes, d'utiliser cette application pour créer une interaction entre plusieurs app, base de données .. (Comme Firebase ou Blynk). Nous l'avons installer sur un Docker.

- Face « Programmation » :



- Face « DashBoard » :



Docker

Docker est une plateforme très utile permettant de créer facilement des conteneurs qui permettent d'exécuter plusieurs processus et applications de manière indépendante.

MariaDB / MyPhPAdmin

MariaDB est une base de données qui sera gérée via l'application Web MyPhPAdmin dans laquelle nous pouvons importer ou exporter des données enregistrées.

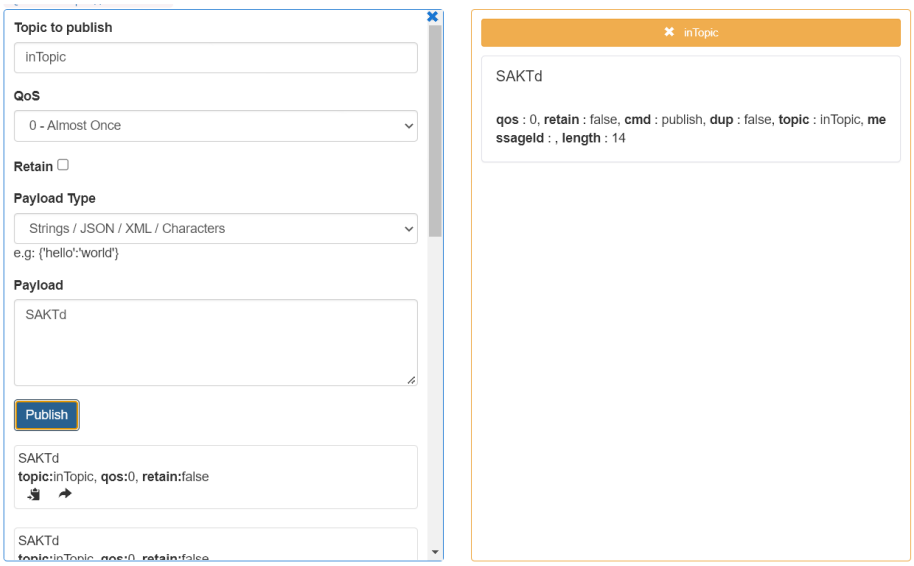
Voici à quoi ressemble la structure dans MyPhpAdmin :

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
☐	1	**ID**	int(11)			Non	*Aucun(e)*		AUTO_INCREMENT	Modifier Supprimer Plus
☐	2	**Temperature**	float			Non	*Aucun(e)*			Modifier Supprimer Plus
☐	3	**Humidite**	float			Oui	*NULL*			Modifier Supprimer Plus
☐	4	**LDR**	int(11)			Non	*Aucun(e)*			Modifier Supprimer Plus
☐	5	**Date**	datetime			Non	current_timestamp()			Modifier Supprimer Plus

Tout cocher *Avec la sélection :* Parcourir Modifier Supprimer Primaire Unique Index Spatial Texte entier

Mosquitto

Mosquitto est un broker qui permet d'établir une communication entre machines via le protocole MQTT , qui fonctionne avec un système de publication et de souscription.



MQTTBOX

MQTTBox est une application pouvant être utilisée comme extension Chrome qui aura pour rôle de créer de nouveaux clients pour « surveiller » notre serveur MQTT. Il est également possible de s'abonner à des « topics » ou bien de publier.

Code source avec commentaires

```
1:
2: #include <WifiClient.h>
3: #include <PubSubClient.h>
4: #include <Wifi.h>
5: #include <Adafruit_NeoPixel.h>
6: #include <Adafruit_Sensor.h>
7: #include <DHT.h>
8: #include <ArduinoJson.h>
9: //pin Variables
10:
11: #define PINLED 2
12: #define DHTpin 26
13: #define LDR 39
14: #define RGB_PIN 27
15: DHT dht(DHTpin, DHT11);
16:
17: //Var RGB
18: int Red = 0;
19: int Green = 0;
20: int Blue = 0 ;
21: // creation rgb
```

```
22: Adafruit_NeoPixel strip = Adafruit_NeoPixel(30, RGB_PIN, NEO_GRB +
NEO_KHZ800);
23: // JSON envoy
24: String output;
25: DynamicJsonDocument doc(96);
26:
27: //Json reception
28:
29: DynamicJsonDocument doc1(96);
30:
31:
32: const char* ssid = "LAPTOP_T";
33: const char* password = "TIMON123";
34: const char* mqtt_server = "192.168.1.51";
35:
36: WiFiClient espClient;
37: PubSubClient client(espClient);
38: unsigned long lastMsg = 0;
39: #define MSG_BUFFER_SIZE    (50)
40: char msg[MSG_BUFFER_SIZE];
41: int value = 0;
42:
43: void setup_wifi() {
44:
45:   delay(10);
46:   // We start by connecting to a WiFi network
47:   Serial.println();
48:   Serial.print("Connecting to ");
49:   Serial.println(ssid);
50:
51:   WiFi.mode(WIFI_STA);
52:   WiFi.begin(ssid, password);
53:
54:   while (WiFi.status() != WL_CONNECTED) {
55:     delay(500);
56:     Serial.print(".");
57:   }
58:
59:   randomSeed(micros());
60:
61:   Serial.println("");
62:   Serial.println("WiFi connected");
63:   Serial.println("IP address: ");
64:   Serial.println(WiFi.localIP());
65: }
66:
67: void callback(char* topic, byte* payload, unsigned int length) {
68:   Serial.print("Message arrived [");
69:   Serial.print(topic);
70:   Serial.print("] ");
71:   String messageTemp;
72:   for (int i = 0; i < length; i++) {
73:     Serial.print((char)payload[i]);
74:     messageTemp += (char)payload[i];
75:
76:   }
77:   Serial.println();
```

```
78:   Serial.println(messageTemp);
79:   if (String(topic) == "inTopic") {
80:
81:     DeserializationError error = deserializeJson(doc1, messageTemp);
82:
83: if (error) {
84:   Serial.print("deserializeJson() failed: ");
85:   Serial.println(error.c_str());
86:   return;
87: }
88:     //stockage des variables du json
89:     Red = doc1["red"];
90:     Green = doc1["green"];
91:     Blue = doc1["blue"];
92:
93:     for (int i = 0; i < strip.numPixels(); i++)
94:         {
95:         strip.setPixelColor(i, strip.Color(Red, Green, Blue));
96:         }
97:         strip.show();
98:
99:       Serial.print(Red + Green + Blue );
100:   }
101: }
102:
103: void reconnect() {
104:   // Loop until we're reconnected
105:   while (!client.connected()) {
106:     Serial.print("Attempting MQTT connection...");
107:     // Create a random client ID
108:     String clientId = "ESP8266Client-";
109:     clientId += String(random(0xffff), HEX);
110:     // Attempt to connect
111:     if (client.connect(clientId.c_str())) {
112:       Serial.println("connected");
113:       // Once connected, publish an announcement...
114:       client.publish("outTopic", "hello world");
115:       // ... and resubscribe
116:       client.subscribe("inTopic");
117:     } else {
118:       Serial.print("failed, rc=");
119:       Serial.print(client.state());
120:       Serial.println(" try again in 5 seconds");
121:       // Wait 5 seconds before retrying
122:       delay(5000);
123:     }
124:   }
125: }
126:
127: void setup() {
128:   pinMode(PINLED, OUTPUT);     // Initialize the BUILTIN_LED pin as an
output
129:   Serial.begin(115200);
130:   setup_wifi();
131:   dht.begin();
132:   client.setServer(mqtt_server, 1883);
133:   client.setCallback(callback);
```

```
134: }
135:
136: void loop() {
137:
138:   if (!client.connected()) {
139:     reconnect();
140:   }
141:   client.loop();
142:   //condition pour envoyer les données
143:   unsigned long now = millis();
144:   if (now - lastMsg > 2000) {
145:     lastMsg = now;
146:     float f = dht.readTemperature();
147:     float h = dht.readHumidity();
148:     int Vldr = analogRead(LDR);   //lecture de la luminosité
149:
150:
151:       //String(f)
152:       doc["TEMPE"] = String(f);
153:       doc["HUMI"] = String(h);
154:       doc["LUMI"] = String(Vldr);   //envoi de la valeur de la
luminosité
155:       String payload;
156:       serializeJson(doc, payload);
157:
158:       client.publish("info", payload.c_str()); //envoi de données sous
format string
159:       Serial.println();
160:       Serial.print("Données envoyées");
161:
162:       Serial.println(payload);
163:
164:
165:
166:
167:   }
168: }
```

Analyse du code source

Déclaration des variables :

Dans notre code, diverses variables importantes sont déclarées pour un bon fonctionnement du code, d'abord les variables des composants de type « Hardware » (**Lignes 11 à 14**), ensuite les variables qui nous permettent d'établir une connexion entre notre serveur MQTT et une connexion Wifi entre l'ESP et notre ordinateur (**Lignes 32 à 34**).

Fonction de connexion wifi et MQTT

Nous avons 2 fonctions qui permettent de se connecter ou de se reconnecter au serveur MQTT et la connexion au wifi.
La connexion au wifi se fait dans la fonction « **setup_wifi ()** » (**Lignes 43 à 65**).

La connexion au serveur MQTT se fait dans la fonction « **reconnect** » (**Lignes 103 à 125)** à chaque déconnexion la fonction sera appelée grâce à une condition qui vérifie l'état de celui-ci dans le loop (**Lignes 138 à 140).**

Fonction Callback

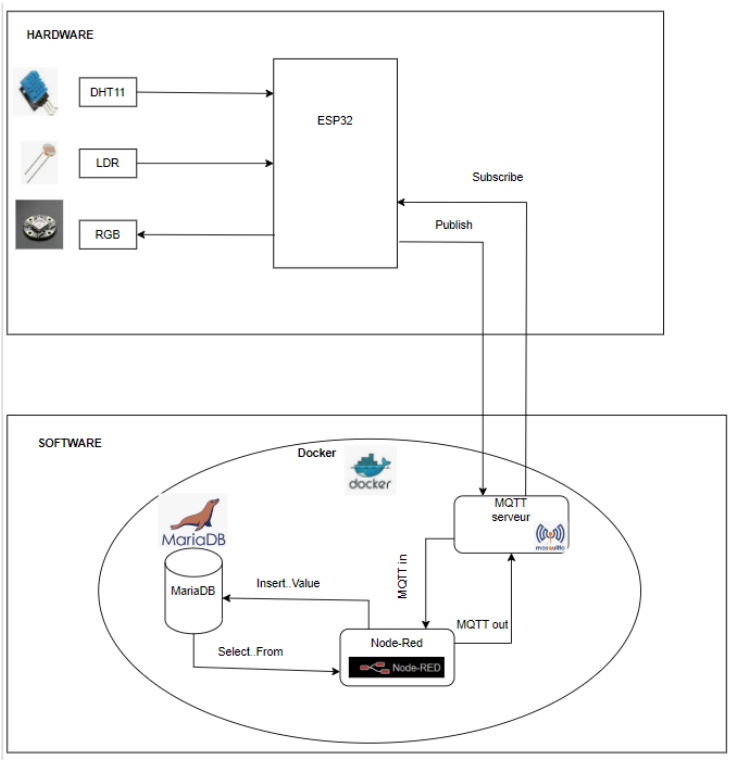
La fonction «callback » sera appelée lorsque le client aura envoyé une donnée dans un des topic mentionné dans la fonction «**reconnect**» à la **ligne 116.**

Si le topic correspond à «i**nTopic** » alors les données reçues seront reconverties sous format JSON pour, par la suite, prendre les valeurs de chaque slider et actualisé l'état de la led Neopixel.(**Lignes 79 à 101**).

Fonction loop

La fonction loop permet d'envoyer les données du DHT11 et du LDR.
Ceci se fera continuellement chaque 2 secondes.
L'envoi se fait avec la méthode **client.publish(topic,msg)** avec comme première argument le topic et deuxième argument le payload à envoyer qui sera dans notre cas en format JSON. (**Lignes 144 à 162**)

Lien Hardware et Software



Conclusion

Pour conclure, notre serveur MQTT nous permet en effet d'établir une communication entre l'ESP32 et Node-Red dans lequel la température, l'humidité et la luminosité sont visibles sous forme de graphe et de jauges. Ces trois données sont également stockées dans notre base de données MariaDB. Nos trois sliders du Dashboard changent bien la couleur de notre rgb Neopixel. Docker est une application fort utile et nous en avons la preuve car un tas d'actions sont réalisés sur diverses app et serveur de manière indépendantes les unes des autres.

7) TP7_DualCore

Objectif du TP

L'objectif de ce Tp est d'introduire la notion de DualCore et FREE RTOS.

Il faudra pour ce Tp créer les 3 tâches suivantes :

1. Tâche LedA
2. Tâche LedB
3. Tâche BP

La tâche LedA fera clignoter une led à T = 500 ms et la tâche LedB fera de même avec une autre led avec une période de T = 250 ms.

La tâche bp doit savoir alterner entre les tâches ledA et LedB à chaque appui sur le bouton poussoir.

Pour ce faire, nous allons utiliser les composants suivants

- Bouton poussoir
- 2 leds

Introduction

Dual Core

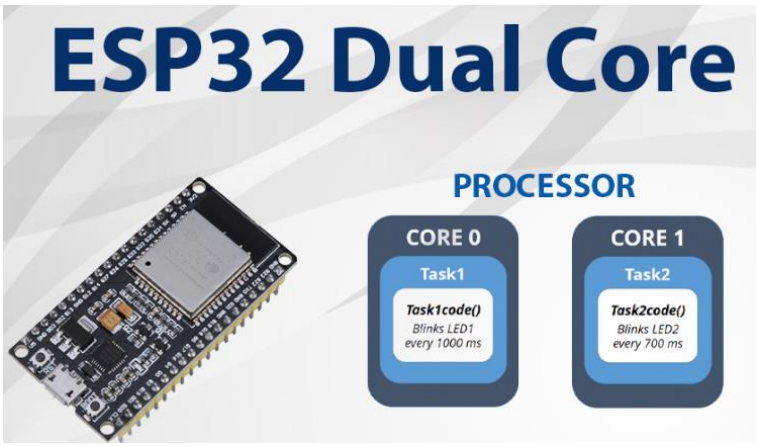
L'esp 32 bénéficie de deux cœurs, cela nous permet de lancer 2 programmes qui peuvent fonctionner Indépendamment et en parallèle.

Ceci peut s'avérer utile lorsque l'on veut, par exemple, mieux organiser notre code pour le rendre plus compacte, plus puissant et plus rapide.

Si on prend l'exemple d'une carte Arduino Uno, on peut vite se retrouver limité dans l'organisation dû au fait que le code fonctionne dans un seul loop.

Toutes les actions doivent se faire les unes à la suite des autres, cela ralenti considérablement la vitesse de calcul de celui-ci.

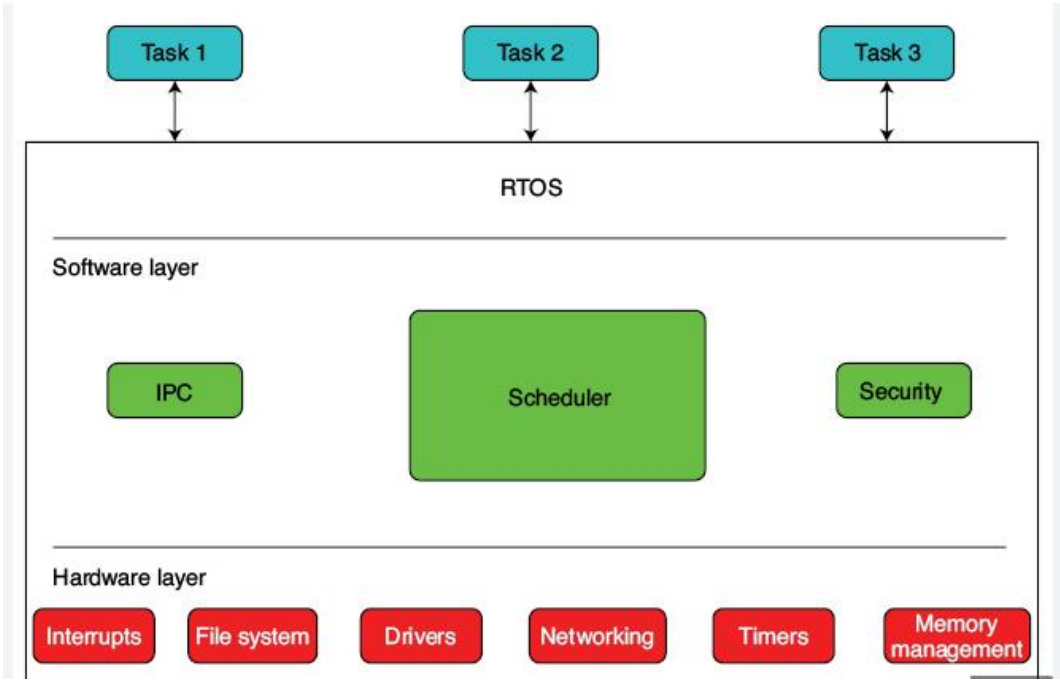
Mais grâce à un **dual core,** il est possible de séparer des sections de code pour attribuer une section de code à un premier cœur et une autre section au deuxième, ce qui est possible grâce à l'ESP 32.



Free Rtos

RTOS (Real Time Operating System) , ou encore « Systèmes d'exploitation temps réel » en français est une couche applicative bas niveau qui permet d'ordonnancer différentes tâches que doit exécuter notre microcontrôleur/microprocesseur.

Dans notre cas, **Free Rtos** nous aidera à créer la tâche **ledA**, **ledB** et **BP** et à la pousser dans le cœur souhaité.



Code source avec commentaires

```
1: #include <Arduino.h>
2: // semaphores.ino
3: // Practical ESP32 Multitasking
4: // Binary Semaphores
5: #define LED1_GPIO 2
6: #define LED2_GPIO 32
7: #define BP 26
8:
9: bool condition = false;
10: bool taskGlobal = false;
11: bool state = true;
12: SemaphoreHandle_t hsem;
13: TaskHandle_t taskA;
14: TaskHandle_t taskB;
15:
16: /*----------------TACHE Bouton-----------------*/
17: void bpTask(void *argp) {
18:
19:   for (;;) {
20:
21:     if(digitalRead(BP) == HIGH && state == true){
22:       condition = !condition;
23:       state = false;}
24:
25:     if(digitalRead(BP) == LOW && state == false ){state = true;}
26:
27:     if(condition == true && taskGlobal != true){
28:       taskGlobal = true;
29:       digitalWrite(LED1_GPIO , LOW);
30:       vTaskSuspend(taskA);
31:       vTaskResume(taskB);
32:       }
33:
34:     else if(condition == false && taskGlobal != false){
35:       taskGlobal = false;
36:       digitalWrite(LED2_GPIO , LOW);
37:       vTaskSuspend(taskB);
38:       vTaskResume(taskA);}
39:     }
40:  }
41:
42: /*----------------TACHE ledA-----------------*/
43: void ledA(void *argp) {
44:   for (;;) {
45:     digitalWrite(LED1_GPIO,digitalRead(LED1_GPIO)^1);
46:     delay(500);
47:     }
48:  }
49:
50: /*----------------TACHE ledB-----------------*/
51: void ledB(void *argp) {
```

```
52:   for (;;) {
53:     digitalWrite(LED2_GPIO,digitalRead(LED2_GPIO)^1);
54:     delay(250);
55:   }
56:  }
57:
58: void setup() {
59:   Serial.begin(115200);
60:   /*Setup bouton et leds*/
61:   pinMode(LED1_GPIO , OUTPUT);
62:   pinMode(LED2_GPIO , OUTPUT);
63:   pinMode(BP,INPUT_PULLDOWN);
64:
65:   digitalWrite(LED1_GPIO , HIGH);
66:   digitalWrite(LED2_GPIO , HIGH);
67:   delay(2000);
68:   digitalWrite(LED1_GPIO , LOW);
69:   digitalWrite(LED2_GPIO , LOW);
70:
71:   int app_cpu = xPortGetCoreID(); // core actuel
72:
73:   xTaskCreatePinnedToCore(
74:     ledA, // Function
75:     "taskA", // Task name
76:     3000, // Stack size (void*)LED1_GPIO, // arg
77:     NULL, // arg
78:     0, // Priority
79:     &taskA, // handle tache
80:     app_cpu); // CPU
81:
82:   xTaskCreatePinnedToCore(
83:     ledB, // Function
84:     "taskB", // Task name
85:     3000, // Stack size (void*)LED2_GPIO, // argument
86:     NULL, // argument
87:     0, // Priority
88:     &taskB, // handle tache
89:     app_cpu); // CPU
90:
91:   xTaskCreatePinnedToCore(
92:     bpTask, // Function
93:     "taskBp", // Task name
94:     3000, // Stack size (void*)LED1_GPIO, // arg
95:     NULL, // arg
96:     0, // Priority
97:     NULL, // No handle returned
98:     0); // CPU
99:
100:     vTaskSuspend(taskB); // suspendre la taches
101:     vTaskSuspend(taskA);
102: }
103:
104: // Not used
105: void loop() {
106:  vTaskDelete(nullptr);
107:  }
```

Analyse du code source

Création des tâches

La création des tâches se fait dans le setup de la **ligne 73 à 98** avec la fonction « **xTaskCreatePinnedTocCore** » qui prend 7 arguments qui seront expliqués ci-dessous.

1. Prend la fonction à exécuter quand la tâche sera appelée
2. Prend le nom que nous avons attribué à la tâche.
3. Prend la taille en octet que la tâches utilisera
4. Est l'argument que l'on donne à la fonction qui sera elle appeler par la tâche
5. Est la variable qui sera utilisé au cas où la tâches devra être suspendu ou autre
6. Prend l'ID du core où la tâche devra être exécuté

Dans notre cas les tâches ledA et ledB seront exécutées dans le core principal (1) et la tâche BP sera exécutée dans le core 2.

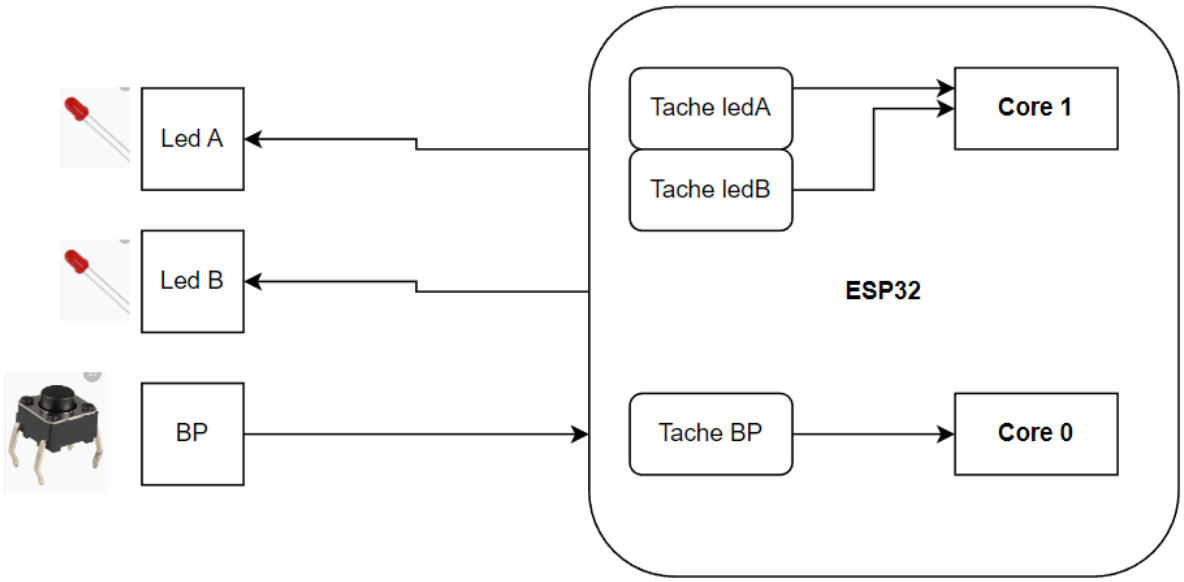
Fonction LedA LedB bpTask

La création des fonctions pour les tâches se fait de **la ligne 16 à 56.**
La led A clignotera de 500 ms dans une boucle for infini dans la fonction **ledA** et la led B clignotera à 250 ms dans une boucle infini dans la fonction **ledB.**

Les tâches ledA et ledB seront appelées avec la fonction « **vTaskResume (nom handle de la tache)**»
Et se feront suspendre avec la fonction « **vTaskSuspend(nom handle de la tache)**»

Ces fonctions sont utilisées dans la fonction «**bpTask**» qui suspendra ou résumera la tâche en fonction d'un bouton toggle.

Lien Software Hardware



Conclusion

Pour conclure, l'utilisation du free RTOS et du dual core a bien été implémenté. Le premier core contient les deux tâches qui feront clignoter les 2 ledss différente. Le deuxième core contient la tâche qui aura pour but de mettre en pause ou en route les tâches du premier core avec l'aide d'un bouton poussoir.

8) TP8_Firebase

Objectif du TP

L'objectif de ce Tp est divisé en deux parties principales. La première est de de stocker des données sur une base de données RT (Real Time) « FireBase » et d'interagir avec celle-ci pour agir sur l'état de deux LEDS.

Dans la deuxième partie, trois données différentes (la température, l'humidité et la luminosité) seront envoyées sur notre même base de données mais sous forme d'historique.

Introduction

Firebase

Firebase est une plate-forme qui nous aide à créer ou améliorer des applications. Nous disposerons de bases de données RT dans lesquelles il est possible d'interagir, d'y écrire des données ou d'en lire.

Code source avec commentaires

```
1:
2: #include <Arduino.h>
3: #if defined(ESP32)
4: #elif defined(ESP8266)
5: #include <ESP8266WiFi.h>
6: #endif
7: #include <Firebase_ESP_Client.h>
8: #include <WiFi.h>
9: #include <Adafruit_Sensor.h>
10: #include <DHT.h>
11: #include <Adafruit_GFX.h>
12: #include <Adafruit_SSD1306.h>
13:
14: //Provide the token generation process info.
15: #include "addons/TokenHelper.h"
16: //Provide the RTDB payload printing info and other helper functions.
17: #include "addons/RTDBHelper.h"
18:
19: //Oled
20: #define SCREEN_WIDTH 128
21: #define SCREEN_HEIGHT 64
22: #define OLED_RESET     -1
23: Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);
24:
25: // Insert your network credentials
26: #define WIFI_SSID "LAPTOP_T"
27: #define WIFI_PASSWORD "TIMON123"
```

```
28:
29: // Insert Firebase project API Key
30: #define API_KEY "AIzaSyBLuTCDpuHVK3jvKWU9Ia4DXOJ9oDd0KQE"
31:
32: // Insert RTDB URLefine the RTDB URL */
33: #define DATABASE_URL "https://tp8-firebase-5d8b5-default-rtdb.europe-
west1.firebasedatabase.app/"
34:
35: #define led1 14
36: #define led2 13
37: #define bp 12
38: #define DHTpin 26
39: #define LDR 34
40: DHT dht(DHTpin, DHT11);
41: //variable ldr_dht
42: float celsius;
43: long Lumi;
44: float Humi;
45:
46:
47: //Variable propre à la classe FireBase
48: FirebaseData stream;
49: FirebaseData fbdo;
50:
51: FirebaseAuth auth;
52: FirebaseConfig config;
53:
54: String valeurLed ="";
55: String valeurBp = "";
56:
57:
58: unsigned long sendDataPrevMillis = 0;
59: int count = 0;
60: bool signupOK = false;
61: volatile bool dataChanged = false;
62:
63: //fonction états LEDS via Firebase
64: void actualisation(String chemin,String valeur){
65:
66:  //prise de la valeur depuis la database
67:   if(String(chemin) == "/json/led1" && String(valeur) == "on"){
68:     Serial.printf("LED1 allumé %s",valeur);
69:     digitalWrite(led1,HIGH);
70:   }
71:   else if(String(chemin) == "/json/led1" && String(valeur) == "off"){
72:      Serial.printf("LED1 eteint%s",valeur);
73:     digitalWrite(led1,LOW);
74:   }
75:
76:   if(String(chemin) == "/json/led2" && String(valeur) == "on"){
77:     Serial.printf("LED2 allumé %s",valeur);
78:     digitalWrite(led2,HIGH);
79:   }
80:   else if(String(chemin) == "/json/led2" && String(valeur) == "off"){
81:      Serial.printf("LED2 eteint%s",valeur);
82:     digitalWrite(led2,LOW);
83:   }
```

```
84: }
85: // fonction Callback
86: void streamCallback(FirebaseStream data){
87:   Serial.printf("sream path, %s\nevent path, %s\ndata type, %s\nevent
type, %s\n\n",
88:                 data.streamPath().c_str(),
89:                 data.dataPath().c_str(),
90:                 data.dataType().c_str(),
91:                 data.eventType().c_str());
92:   printResult(data);
93:   Serial.println();
94:
95:   //appel de la fonction d'actualisation des etat des LEDS en apssant
sur Firebase
96:   actualisation(data.dataPath(),data.to<String>());
97:
98:
99:   // This is the size of stream payload received (current and max
value)
100:   // Max payload size is the payload size under the stream path since
the stream connected
101:   // and read once and will not update until stream reconnection takes
place.
102:   // This max value will be zero as no payload received in case of
ESP8266 which
103:   // BearSSL reserved Rx buffer size is less than the actual stream
payload.
104:   Serial.printf("Received stream payload size: %d (Max. %d)\n\n",
data.payloadLength(), data.maxPayloadLength());
105:
106:   // Due to limited of stack memory, do not perform any task that used
large memory here especially starting connect to server.
107:   // Just set this flag and check it status later.
108:   dataChanged = true;
109: }
110:
111: void streamTimeoutCallback(bool timeout)
112: {
113:   if (timeout)
114:     Serial.println("stream timed out, resuming...\n");
115:
116:   if (!stream.httpConnected())
117:     Serial.printf("error code: %d, reason: %s\n\n", stream.httpCode(),
stream.errorReason().c_str());
118: }
119: //prise de temp/hum/lumi
120: void Temp_HUM_LDR(){
121:
122:   celsius = dht.readTemperature();
123:   Humi = dht.readHumidity();
124:   //esp delivre 12 Bytes par defaut
125:   Lumi= map(analogRead(LDR), 0, 4095, 0, 100);
126: }
127: //fonction affichage oled
128: void affichage_oled(){
129:
130: display.clearDisplay();
```

```
131: display.setTextSize(1);
132: display.setTextColor(WHITE);
133: display.setCursor(1, 5);
134: display.println("Led1 : " + String(digitalRead(led1)));
135: display.setCursor(1,15);
136: display.println("Led2 : "+ String(digitalRead(led2)));
137: display.setCursor(1,25);
138: display.println("SW : "+ String(digitalRead(bp)));
139: display.setCursor(1,35);
140: display.println("T: "+String(celsius)+" C "+" H: "+String(Humi)+" %");
141: display.setCursor(1,45);
142: display.println("Luminosite : " + String(Lumi)+" %");
143: display.display();
144: }
145:
146: void setup(){
147:   pinMode(led1,OUTPUT);
148:   pinMode(led2,OUTPUT);
149:   pinMode(bp,INPUT_PULLDOWN);
150:   pinMode(LDR,INPUT);
151:   dht.begin();
152:
153:   Serial.begin(115200);
154:   WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
155:   Serial.print("Connecting to Wi-Fi");
156:   while (WiFi.status() != WL_CONNECTED){
157:     Serial.print(".");
158:     delay(300);
159:   }
160:   Serial.println();
161:   Serial.print("Connected with IP: ");
162:   Serial.println(WiFi.localIP());
163:   Serial.println();
164:   //partie oled
165:   if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D
Pour 128x64
166:       Serial.println(F("SSD1306 allocation failed"));
167:     }
168:
169:   //Assigner la cle API
170:   config.api_key = API_KEY;
171:
172:   //Assigner l url
173:   config.database_url = DATABASE_URL;
174:
175:   //Sign up
176:   if (Firebase.signUp(&config, &auth, "", "")){
177:     Serial.println("ok");
178:     signupOK = true;
179:   }
180:   else{
181:     Serial.printf("%s\n", config.signer.signupError.message.c_str());
182:   }
183:
184:   //Assign the callback function for the long running token generation
task
185:   config.token_status_callback = tokenStatusCallback;
```

```
186:
187:    if (!Firebase.RTDB.beginStream(&stream,
"/Signalisation/stream/data"))
188:      Serial.printf("sream begin error, %s\n\n",
stream.errorReason().c_str());
189:
190:    Firebase.RTDB.setStreamCallback(&stream, streamCallback,
streamTimeoutCallback);
191:
192:    Firebase.begin(&config, &auth);
193:    Firebase.reconnectWiFi(true);
194:
195:
196:    //Envoye de l'etat des LEDS à Firebase
197:    FirebaseJson json;
198:    json.add("led1", "Off");
199:    json.add("led2", "Off");
200:    Serial.printf("Set json... %s\n\n", Firebase.RTDB.setJSON(&fbdo,
"/Signalisation/stream/data/json", &json) ? "ok" :
fbdo.errorReason().c_str());
201:
202: }
203:
204: void loop(){
205:    Temp_HUM_LDR();
206:
207:    if (digitalRead(bp) == true){
208:      valeurBp = "On";
209:    }
210:
211:    else if (digitalRead(bp) == false){
212:      valeurBp = "Off";
213:    }
214:    affichage_oled();
215:    //Condition pour envoyer nos donnees sur Firebase
216:    if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis >
1500 || sendDataPrevMillis == 0)){
217:      sendDataPrevMillis = millis();
218:      }
219:      //envoie de temp_humi_lumi a firebase
220:      FirebaseJson json;
221:      json.add("Temperature", celsius);
222:      json.add("Humidité", Humi);
223:      json.add("Luminosité", Lumi);
224:      Serial.printf("Set json... %s\n\n", Firebase.RTDB.pushJSON(&fbdo,
"/test/stream/data/json", &json) ? "ok" : fbdo.errorReason().c_str());
225:
226:      //verification que l'etat du bouton s'est envoye
227:      if (Firebase.RTDB.setString(&fbdo, "Bouton/String", valeurBp)){
228:
229:      }
230:      else {
231:        Serial.println("FAILED");
232:        Serial.println("REASON: " +fbdo.errorReason());
233:      }
234:      //prise de la valeur du bouton
235:      if (Firebase.RTDB.getString(&fbdo, "/Bouton/String")) {
```

```
236:       if (fbdo.dataType() == "String") {
237:         String Valed = fbdo.stringData();
238:         Serial.println(Valed);
239:       }
240:     }
241:     else {
242:       Serial.println(fbdo.errorReason());
243:     }
244:
245:
246:
247:   if (dataChanged)
248:   {
249:     dataChanged = false;
250:
251:   }
252:
253:
254: }
255:
```

Analyse du code source

Partie connexion :

Pour établir une connexion entre notre ESP32 et Firebase, il faut entrée l'URL de notre Database et sa clé « API » ce qui est fait dans les lignes **170 et 173** dans le setup.

Déclaration des variables :

Dans cette partie du code, nous déclarons les variables globales. Notre Clé API et l'url de notre DataBase (**Lignes 30 et 33**) , les pins du matériel utilisé (**Lignes 35 à 39**) et également des variables propre à la classe Firebase (**Lignes 49 à 52**).

Fonctions importantes :

Plusieurs fonctions sont primordiales pour un bon fonctionnement de notre TP, premièrement, la fonction « actualisation » qui permet de lire les données de la Database et de faire une action en fonction de ce qui a été lu. Dans notre cas, cette fonction sert à allumer ou éteindre nos LEDS (**Lignes 64 à 84**).

Ensuite, La fonction « CallBack » est une fonction qui est appelée lorsqu'un changement dans la base de données de Firebase sera fait. Nous pourrons donc décider quelles actions faire dans cette fonction. Dans notre cas nous appelons la fonction « Actualisation » expliquée ci-dessus (**Lignes 86 à 109**).

Méthodes pour envoyer et recevoir les données sur notre DataBase :

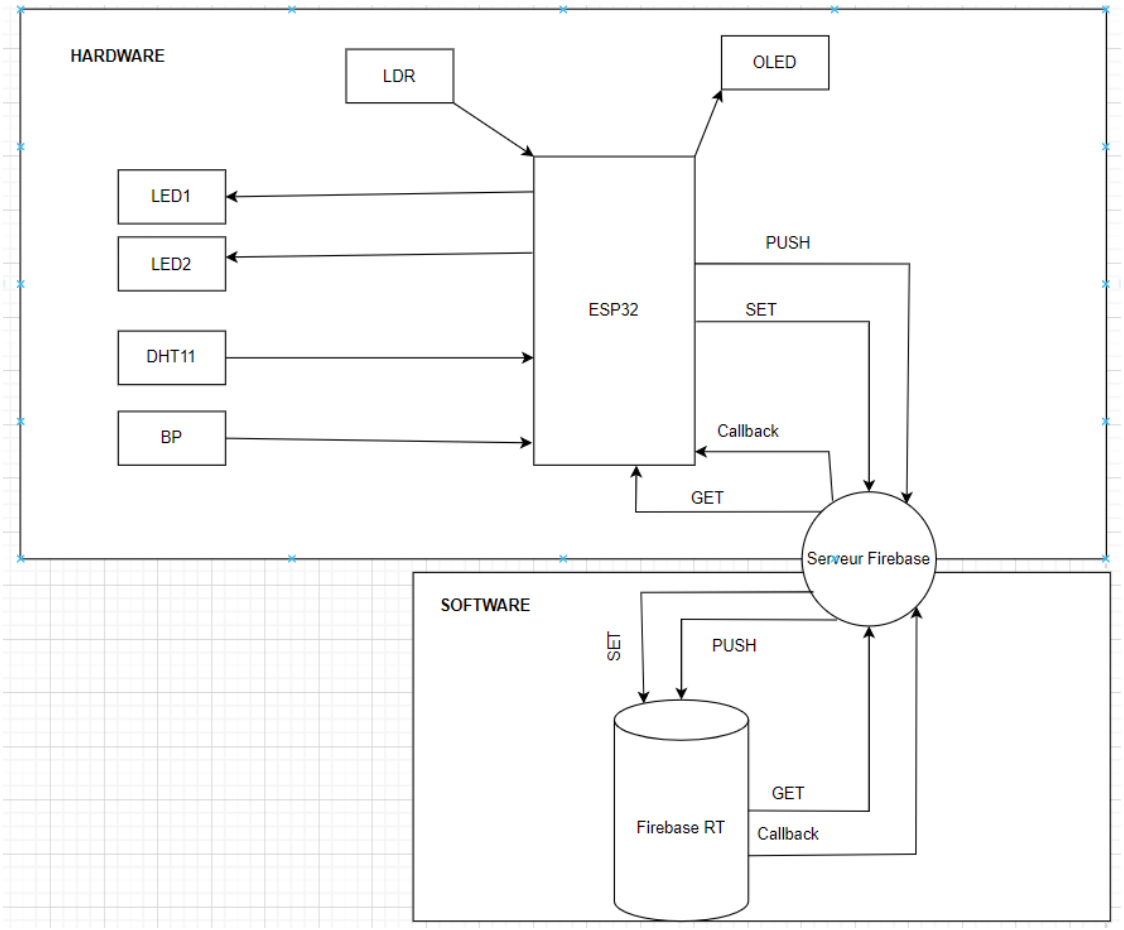
Premièrement , la méthode « signup » nous permet de savoir si la connexion à bien été établie , il renverra un 1 logique si c'est la cas. Cette valeur logique est stockée dans une variable « SignupOK » (**Lignes 176 à 179**).

Deuxièmement, nous utilisons deux méthodes différentes pour envoyer les données sous un format choisi (en JSON dans notre cas) à notre Database, d'un côté , la méthode « Firebase.RTDB.SETJSON » qui écrasera les données dans la database sur un chemin spécifié par une nouvelle donnée sous un type de donnée choisi (**Ligne 200 et 227**).

De l'autre côté , la méthode « Firebase.RTDB.PUSHJSON » fait les même actions que le « SET » à la différence qu'à la place de réécrire la valeur, les nouvelles valeurs seront écrites les unes à la suite des autres de façon à garder toutes nos données pour avoir un historique. (**Ligne 224**).

Troisièmement , la méthode « Firebase.RTDB.GETString » va prendre une valeur à un chemin spécifié dans notre database pour la stocker dans une variable mise en argument (**Ligne 235**).

Lien Software Hardware



Conclusion

Pour conclure, une communication bidirectionnelle entre Firebase et l'ESP32 est bien présente. Nos deux Leds ne s'allument que lorsqu'un changement a été fait directement dans la base de données Firebase. La température, humidité et luminosité sont envoyées vers la même base de données mais avec une méthode différente qui permet d'écrire les valeurs les unes à la suite des autres. Un tas de données sont donc gérées entre Firebase et notre ESP32.

Lien Github :

https://github.com/nwogburumichael/syst-mes_embarqu-s

Conclusion générale

Pour conclure, après avoir réalisé de nombreux travaux différents. Nous avons découvert le vaste monde qu'est l'IOT. Grâce aux diverses types de communications vu dans le cours des TP réalisés , nous avons pu faire interagir plusieurs composants hardware , comme des capteurs, des boutons ou encore des LEDS, avec des composants virtuels se trouvant dans des Dashboards comme Blynk, Adafruit IO , Node-Red. Nous avons pu enrichir nos connaissances , que ce soit du côté logique, ou du côté pratique ce qui nous a poussé à développer notre capacité d'adaptation et de recherche.

Tout ceci sont des compétences importantes pour devenir des personnes plus autonome et efficace dans le cadre de la vie professionnel ou privée.