

Rapport de projet

- Internet Of Things -

Paris - 15 avril 2018



Par Massinissa BENAISSE, Johlian RAJAONA et Anis SARDAOUI

à l'attention de M. OSMANI



Master 1

Information System and
Cloud Engineering

Promotion 2019

Table des matières

Rapport de projet	1
Table des matières	2
Introduction.....	3
I - Projet donné par notre professeur	4
1) Complexité du code	4
2) Exemple descriptif	4
II- Application mobile.....	6
1) Thunkable	6
2) Difficultés	7
III- Serveur WEB via l'Arduino IDE	8
1) Ajout du plugin pour ESP32	8
2) Importation des headers	8
3) Programme gérant le back-end du système	9
4) Gestion du front-end	9
Annexes	10
Annexe A	11
Annexe B	12
Annexe C	13

Lien video de démonstration sur Youtube : <https://youtu.be/PdSz51LADg0>

Introduction

Dans cette partie nous présentons la genèse du projet. Comment celui-ci s'est mis en place, en adéquation avec les attentes de notre professeur. Nous expliquerons aussi, la logique qui a été suivi, pour rédiger ce rapport : afin que vous puissiez le comprendre totalement.

Dans le cadre de notre cours d'introduction aux IoT, nous avons eu plusieurs TP à réaliser sur ESP32. Si le premier TP n'était qu'un tutoriel de prise en main des nouveaux outils, le deuxième en revanche, était un peu plus complexe puisqu'il impliquait l'utilisation d'un capteur de température BME 280.

En effet, le deuxième TP permettait d'introduire l'utilisation du capteur, afin de pouvoir le réutiliser en projet.

Le projet donné initialement par notre professeur, consistait en la programmation d'une station météo (utilisant le capteur BME 280) qui diffusait les résultats sur un serveur WEB via la carte Wi-Fi de l'ESP32. Nous avions à notre disposition plusieurs morceaux de code, que nous avions récupérer sur le GitHub de notre intervenant TP. Il fallait donc reconstituer l'arborescence des codes et les compléter.

Ce projet là, nous a paru assez complexe à réaliser (on explique pourquoi dans la première partie de ce rapport). Nous nous sommes donc entretenu avec notre professeur afin d'étudier les autres possibilités. Après nous être entretenus avec notre professeur nous avons décidé de créer une application web ou mobile (qui afficherait les températures comme voulu initialement, mais qui offrirait la possibilité d'avoir un suivi de l'évolution : par heure, par minute ... à l'instar de l'application santé d'apple avec le suivi d'activité). Nous sommes donc parti sur ce projet pendant un moment, en essayant de créer une application mobile via le framework de thunkable. Puis confronté là aussi à des difficultés (notamment liée à la base de donnée locale), difficulté que nous détaillerons dans la deuxième partie, nous avons abandonné ce projet là.

Finalement, après une recherche approfondi, nous nous sommes rendus compte qu'il existe un IDE ("Arduino") mis en place par genuino, pour contrôler les Arduino, qui était compatible avec l'ESP32 en ajoutant un plug-in dédiée. La force de cet IDE réside dans le fait que, l'interface utilisateur est simplifiée et que le code nécessaire pour contrôler le capteur BME 280, est moins complexe. Nous sommes donc parti sur cette solution là, solution qui est détaillé dans la troisième partie de ce rapport.

Vous l'aurez donc compris, pour que ce rapport soit compréhensible par le plus grand nombre de personnes, nous avons décidé de décrire notre travail, tel qu'il a été fait : c'est-à-dire décrire le travail effectué chronologiquement.

I - Projet donné par notre professeur

Le projet d'origine consiste à utiliser le langage C pour permettre le fonctionnement de l'ESP32 et instaurer la communication avec le BME280. On nous a ainsi fourni le code d'origine pour démarrer ce projet.

1) Complexité du code

Il faut savoir que dans ce code, il y a plusieurs aspects complexes. Tout d'abord, cela est lié au langage utilisé. Le langage C, bien que faisant office de langage de base, n'est pas celui qui est le plus facile à utiliser pour des projets comme celui qui nous a été donné.

Ensuite, dans le projet d'origine, il a été demandé de remplir les espaces vides (indiqués par des étapes TODO). Néanmoins, il y a plusieurs obstacles à prendre en compte :

-Tout d'abord, pour une étape TODO, il y a une réflexion à avoir.

```
switch (net_proto) {
    case IPv4:
        rc = setup_ipv4();
        if (rc < 0) {
            printf("[IoT-Labs]");
            return rc;
        }
        break;

    case IPv6:
        // TODO consider using the following function call in order to use IPv6:
        //     `tcpip_adapter_create_ip6_linklocal()`
        rc = ENOSYS;
        break;

    case _6LowPAN:
        // FIXME BLE would work only with 6LowPAN over it. Any other combination
        //     has to throw an exception
        // TODO take a look at
        //     http://git.savannah.nongnu.org/cgit/lwin.git/tree/src/include/netif/lowpan6\_ble.h
        rc = ENOSYS;
        break;
}
```

2) Exemple descriptif

Prenons un exemple avec un bout de code ci-dessus. Pour savoir comment effectuer une étape TODO, il aurait fallu comprendre la logique des paramètres et connaître la façon dont on aurait configuré le protocole.

```

int8_t
setup_ipv4(void)
{
    int8_t rc;
    tcpip_adapter_ip_info_t ip_info;

    // wait for connection
    printf("[IoT-Labs] Waiting for connection to the wifi network... \n");
    xEventGroupWaitBits(wifi_event_group, IPv4_CONNECTED_BIT, false, true,
                        portMAX_DELAY);
    printf("[IoT-Labs] connected!\n");

    // print the local IP address
    rc = tcpip_adapter_get_ip_info(TCPIP_ADAPTER_IF_STA, &ip_info);
    if (rc < 0) {
        printf("[IoT-Labs] Error while getting ip information\n");
        return rc;
    }

    printf("[IoT-Labs] IP Address: %s\n", ip4addr_ntoa(&ip_info.ip));
    printf("[IoT-Labs] Subnet mask: %s\n", ip4addr_ntoa(&ip_info.netmask));
    printf("[IoT-Labs] Gateway:      %s\n", ip4addr_ntoa(&ip_info.gw));

    return rc;
}

```

En regardant déjà la fonction pour IPv4 qui est utilisée dans le cas où on choisit IPv4, on voit déjà une structure un peu compliquée à comprendre. Il aurait fallu qu'on comprenne la logique de cette fonction pour avoir une idée des fonctions pour IPv6 et 6LowPAN.

Le deuxième obstacle, qui est lié au premier, est le nombre de TODO à réaliser. Les autres étapes du même acabit voire encore plus compliqué que celle présentée précédemment. Il y a par exemple les fonctions concernant l'ESP32 pour la communication, ou encore l'appel des couches et des protocoles à associer.

Le dernier obstacle concerne l'implémentation du code dans l'ESP32. Il est assez compliqué de pouvoir implémenter du code en C dans un équipement tel que l'ESP32, notamment s'il est difficile de trouver un support logiciel adapté avec des drivers.

Cet ensemble d'obstacles rend ce projet assez compliqué à mettre en place. En effet, aussi bien dans la réalisation que dans l'implémentation, il y a une grande difficulté pour mener à bien ce projet.

II- Application mobile

Nous avons réfléchis à diffèrent moyen pour afficher nos résultats afin qu'ils soient mis en valeur et affiché de la façon la plus esthétique et pratique pour l'utilisateur. Pour ce faire nous avons penser a mettre au point une application mobile qui serait un moyen efficace de répondre à nos attentes. On s'est donc basé sur le concept ergonomique de l'application d'apple : **santé**, qui affiche plusieurs données par le biais de graphique par exemples et qui est très intuitive.

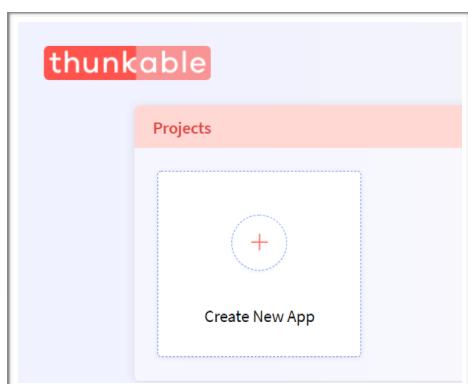
Ainsi nous avons commencé à chercher un moyen pour réaliser ce concept, quel logiciel, quelle base de données utiliser ? Comment les relier à notre système ?

1) Thunkable

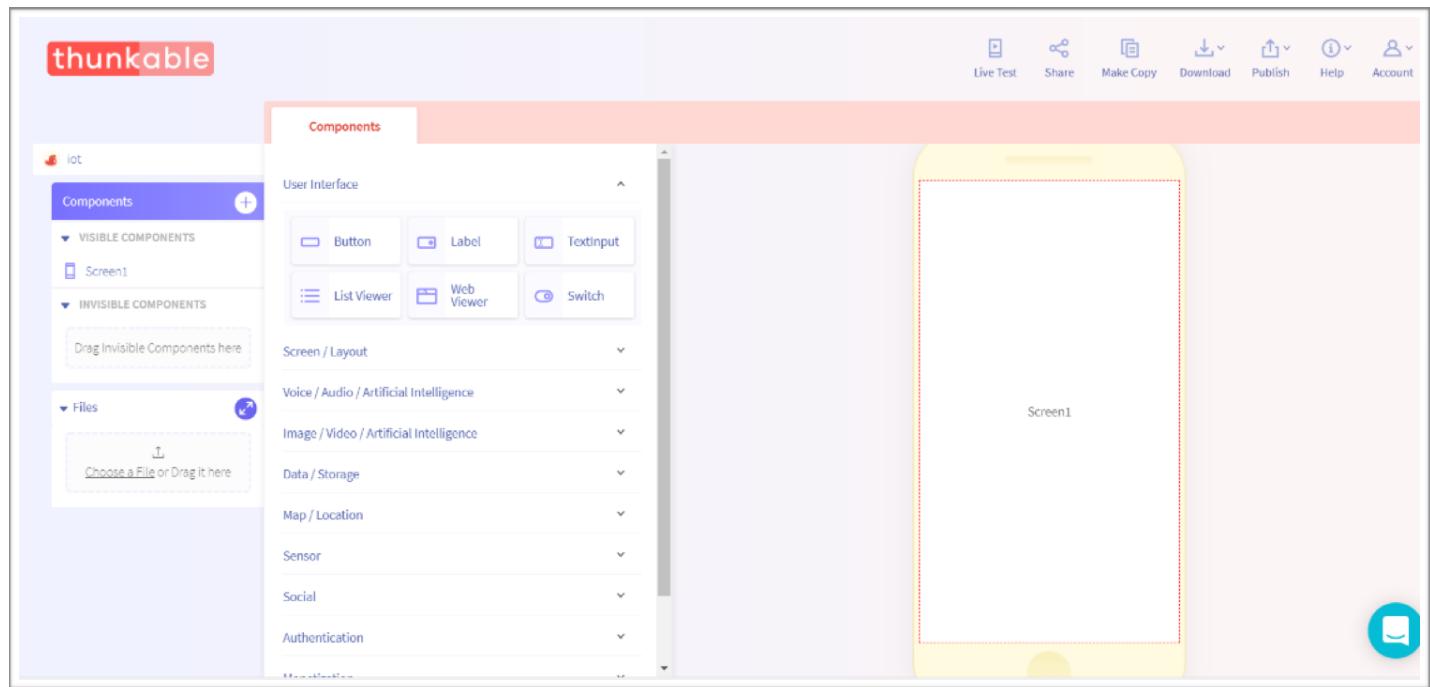
Il est apparu après quelques recherches que THUNKABLE pouvait s'avérer être un bon moyen de développer notre concept d'application



En effet THUNKABLE est une plateforme qui permet de créer sa propre application mobile. De plus le design des applications qu'il permet de construire et la facilité de la prise en main que THUNKABLE offre ont fait que nous avons naturellement voulu utiliser cette plateforme pour notre projet.



Nous avons donc tenté de créer une nouvelle application et nous avons bien eu accès à l'environnement de thunkable et les différentes fonctionnalités qu'il propose pour réaliser le design de celle-ci



2) Difficultés

Toutefois alors que nous cherchions une solution facilement développable et capable d'afficher les données du capteur que nous avons utilisé, plusieurs problèmes ont rendu impossible pour nous le développement de l'application comme nous le souhaitions à la base.

En effet il nous a été impossible de connecter l'ESP32 à l'application pour que celle-ci remonte ses données. De plus nous n'avons pas pu mettre en place une base de données à l'aide de cette plateforme, base de données qui été indispensable, afin de pouvoir afficher des graphiques, se basant sur des données précédemment enregistrées. Le temps dont nous disposions ne permettait pas de continuer à essayer de résoudre ces problèmes, qui rendait complètement impossible notre avancée dans la réalisation du projet.

Tous ces éléments nous ont fait opter pour une autre solution : la mise en place d'un site proposant le même contenu que celui que nous voulions réaliser au départ.

Cette expérience nous aura néanmoins permis, de nous familiariser avec l'outil Thunkable, qui aurait peut-être pu répondre, à nos besoins s'il nous était possible d'y consacrer plus de temps.

III- Serveur WEB via l'Arduino IDE

Nous avons donc décidé que la solution finale serait de mettre en place un serveur WEB diffusant les mesures de température en temps réel, (comme prévu initialement par notre professeur) sauf que nous allions utilisé l'IDE Arduino. La première chose que nous avions fait, avant de programmer le serveur WEB, était de rajouter le plugin ESP32

1) Ajout du plugin pour ESP32

Il faut savoir que nous avons travailler sur un système d'exploitation UNIX (Mac OS Sierra 10.12) donc toute la démarche qui sera décrite ci-dessous, devra être adapter en fonction du système d'exploitation utilisé.

La première chose à faire, est de créer un dossier « espressif » dans le sous-dossier hardware de Arduino (IDE) : on peut réaliser cette action, en executant dans le terminal la commande suivante :

```
mkdir -p ~/Documents/Arduino/hardware/espressif
```

Une fois le dossier créé, il faut s'y déplacer afin d'importer un projet via l'outil collaboratif GIT, en executant la commande suivante :

```
git clone https://github.com/espressif/arduino-esp32.git esp32
```

Une fois le dossier esp32 importé, il faut encore une fois se déplacer dans le sous-dossier « tools » afin d'y lancer la commande :

```
python get.py
```

Il convient ensuite de relancer l'IDE Arduino, et ce sera tout, le plugin ESP32 a été installer avec succès

2) Importation des headers

Avant de commencer à programmer le système, il faut importer quelques libraries pour que notre code fonctionne correctement. Les deux premières libraries à importer sont les libraries « WiFi.h » et « WiFiClient.h » directement disponible depuis l'IDE.

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

Ces deux libraries facilitent l'utilisation de la carte wifi de l'ESP32, en nous permettant notamment l'émission de données (diffusion des mesures de température via Serveur WEB).

Ensuite, il faut importer deux autres libraries : la première est « WebServer.h » que vous trouverez dans le code source du projet, et « Adafruit_BME280.h ». Nous avons réutiliser ces deux dernières bibliothèques qui existaient déjà, et qui étaient en libre accès sur GitHub.

WebServer.h permet de configurer l'environnement réseau, avec notamment l'implémentation des protocoles IP, et l'utilisation des différentes méthodes de HTTP. C'est aussi ici, qu'on définit les ports sur lequel va être lancer le serveur.

Adafruit_BME280.h quant à elle, est une library permettant de configurer le BME280 en I2C ou en SPI (respectivement half-duplex et full-duplex). Cette bibliothèque permet également de configurer chaque port du capteur, nous permettant par la même occasion de configurer les broches SDA et SCL.

Pour pouvoir ajouter les bibliothèques « WebServer.h » et « Adafruit_BME280.h » au programme Arduino, il faut aller dans le menu **Croquis** puis dans la rubrique **Inclure une bibliothèque** et il faut choisir l'option **Ajouter la bibliothèque .ZIP...**. Une nouvelle fenêtre s'ouvre alors, il faut donc choisir les dossiers WebServer puis refaire exactement la même manipulation pour le dossier Adafruit. Il faudra préalablement compresser ces deux dossiers indépendamment en **.ZIP**.

3) Programme gérant le back-end du système

Le programme gérant le back-end (fonctionnement de l'ESP32 et diffusion sur serveur) est assez simple à expliquer, celui-ci se trouve dans le dossier tar.gz du code source, mais aussi en **Annexe B** de ce rapport.

Tout d'abord il y a la déclaration du port sur lequel nous voulons diffuser (par défaut à 80 : port TCP). Ensuite on définit quelques variables globales : ssid et password du réseau WiFi, Altitude (important puisqu'il modifie les résultats de pression), les broches SDA et SCL, et la connexion au BME280. Enfin on initialise à 0, les variables qui vont contenir les valeurs de température, d'humidité et de pression.

Ensuite il y a 9 méthodes qui gère l'ESP32 : setup, loop, connectToWifi, beginServer, handleRoot, initSensor, getTemperature, getHumidity, getPressure.

- Le setup permet de lancer les méthodes initSensor, connectToWifi et beginServer
- loop permet de faire des mesures à intervalles régulier
- ConnectToWifi permet la connection au réseau wifi et permet d'afficher l'IP du serveur
- beginServer lance le serveur
- handleRoot permet de lancer le front-end du serveur
- initSensor initialise et synchronise le BME280
- getTemperature permet de faire une mesure de température
- getHumidity permet de faire une mesure d'humidité
- getPressure permet de faire une mesure de pression

4) Gestion du front-end

Pour le front-end, nous avons utilisé du HTML, du CSS et du javaScript. Nous avons d'abord désigné la page que l'on voulait avoir, et nous l'avons codé à part. Nous voulions avoir une page simple affichant en entête les enjeux d'une telle page (projet) nos noms, un logo de l'EFREI. Puis au centre de la page, bien en évidence les mesures du capteur. Nous voulions ensuite une petite rubrique comment s'habiller qui est statique et qui affiche les conseils à tout moment. Puis une rubrique en rouge, qui est dynamique et qui affiche les conseils en fonction de la température du capteur. Cette dernière rubrique a été codé en utilisant des **if** en JavaScript. Enfin nous voulions afficher un graphique qui affichait les mesures de température en temps réel. Pour ce faire nous avons utilisé un script JavaScript basé sur CanvasJS.

Une fois le code du site achevé, il a fallu l'intégrer à la méthode `getpage()` du code C++ dans Arduino IDE. Il a donc fallu transformer tout le code (HTML, CSS, JS) en string. Nous sommes très content du rendu (vous pouvez voir un screen de cette page en Annexe C de ce rapport).

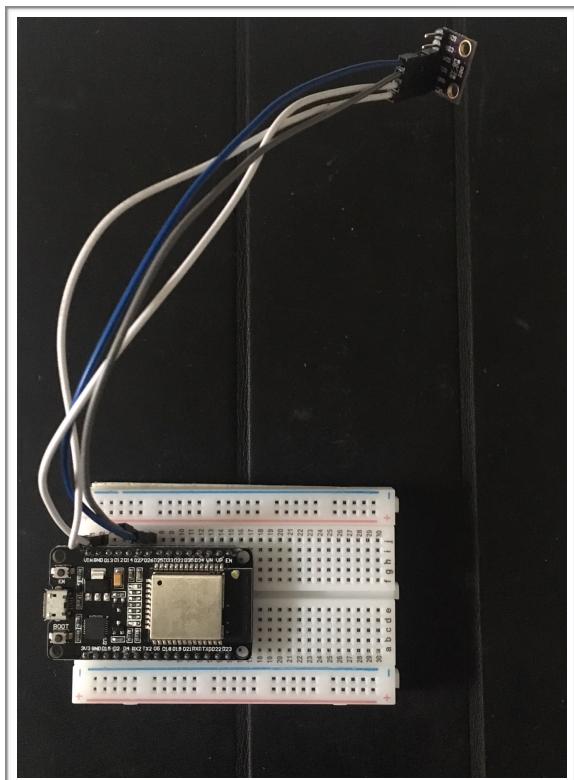
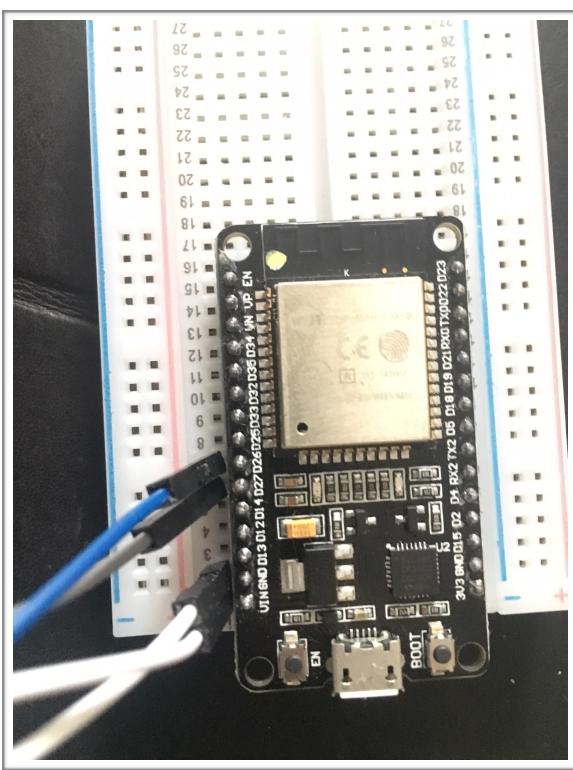
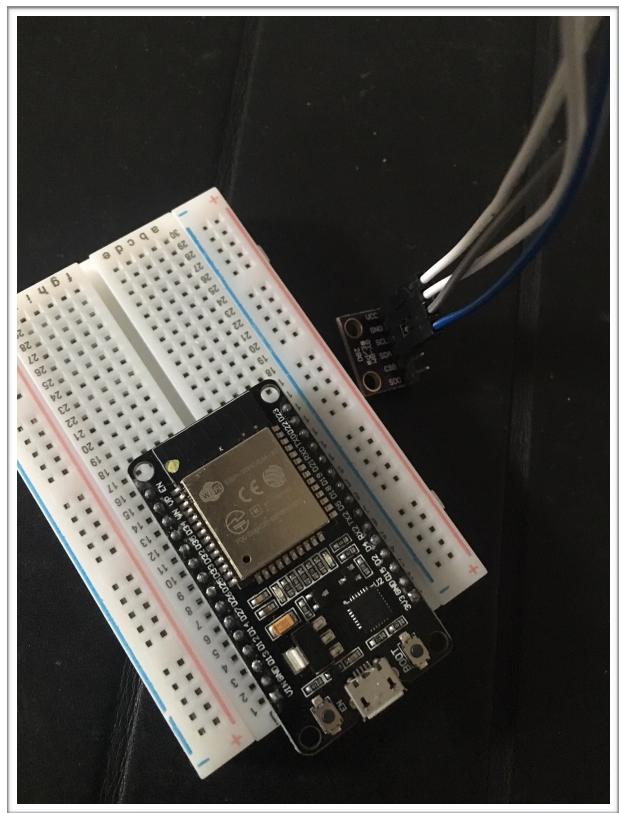
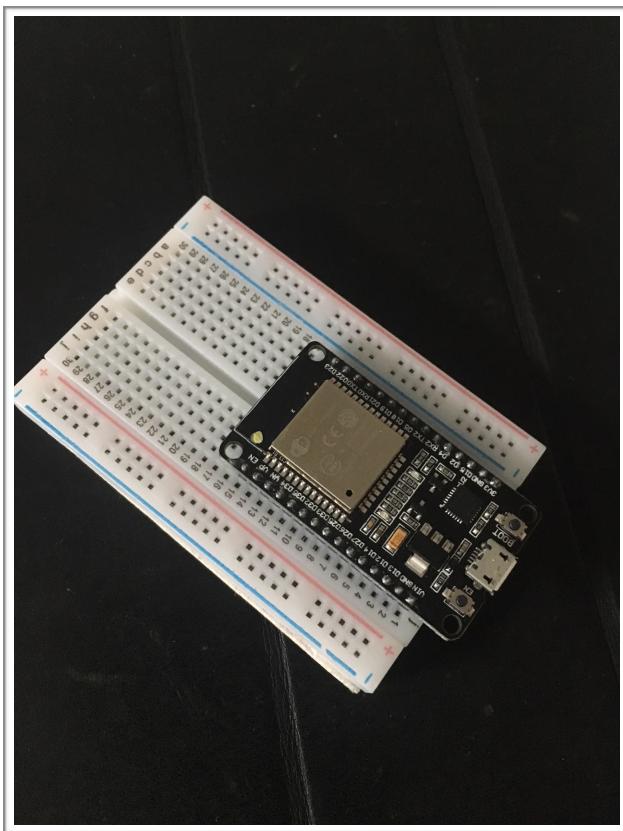
Annexes

[Annexe A](#) : Quelques photos des branchements entre l'ESP32 et le BME280

[Annexe B](#) : Code du back-end

[Annexe C](#) : Un screen du serveur, une fois lancé

Annexe A



Annexe B

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include "Adafruit_BME280.h"

WebServer server ( 80 );
const char* ssid      = "iPhone 6s de Anis";
const char* password = "anis19961";
#define ALTITUDE 55.0 // Altitude in PARIS
#define I2C_SDA 27
#define I2C_SCL 26 //half duplex
#define BME280_ADDRESS 0x76 // ou 0X77 si
                        //ne marche pas

float temperature = 0;
float humidity = 0;
float pressure = 0;

Adafruit_BME280 bme(I2C_SDA, I2C_SCL);

void setup()
{
    Serial.begin(9600);
    initSensor();
    connectToWifi();
    beginServer();
}

void loop()
{
    server.handleClient();
    getTemperature();
    getHumidity();
    getPressure();
    delay(1000);
}

void connectToWifi()
{
    WiFi.enableSTA(true);
    delay(2000);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

void beginServer()
{
    server.on ( "/", handleRoot );
    server.begin();
    Serial.println( "HTTP server started" );
}

void handleRoot(){
    server.send ( 200, "text/html", getPage() );
}
void initSensor()
{
    bool status = bme.begin(BME280_ADDRESS);
```

```
if (!status) {
    Serial.println("Could not find a valid
BME280 sensor, check wiring!");
    while (1);
}

float getTemperature()
{
    temperature =
bme.readTemperature();
}

float getHumidity()
{
    humidity = bme.readHumidity();
}

float getPressure()
{
    pressure = bme.readPressure();
    pressure =
bme.seaLevelForAltitude(ALTITUDE,
pressure);
    pressure = pressure / 100.0F;
}
```

Annexe C

EFREI Students:
- Anis SARDAOUI
- Massinissa BENAISSA
- Johlian RAJAONA

Master 1
Information System and Cloud Engineering
Promotion 2019



IoT Project

WEATHER STATION using ESP32 + BME280

Measurements:

Temperature: **13.26°C**
Humidity: **60.82 %**
Barometric Pressure: **1012.35 hPa**

Comment s'habiller ?

- > jusqu'à 10 ° CLIMAT FROID : doudoune, manteau ou équivalent Une écharpe et des gants sont nécessaires
- > de 10 ° à 20 ° : veste, pull-over, chemise à longues manches ou équivalent
- > des 20 ° : chemise à manches courtes ou équivalent

Votre CAS :

RAPPEL température : 13.26°C

Vous devez prendre : doudoune, manteau ou équivalent. Une écharpe et des gants sont nécessaires

Prise de température

