Laboratoires IoT de base

Mise en oeuvre des architectures IoT à la base de IoT-DevKit de SmartComputerLab

Contenu

Laboratoires IoT de base

Mise en oeuvre des architectures IoT à la base de IoT-DevKit de SmartComputerLab

0. Introduction

Dans les laboratoires IoT nous allons mettre en œuvre plusieurs architectures IoT intégrant les terminaux (T), les passerelles (*gateways* - G), et les serveurs (S) IoT type **ThingSpeak** ou brokers **MQTT**. Le développement sera réalisé sur les cartes **IoTDevKit** de **SmartComputerLab**.

Le kit de développement contient une carte de base "basecard" pour y accueillir une unité centrale et un ensemble de cartes d'extension pour les capteurs, les actuateurs et les modems supplémentaires.
L'unité centrale est une carte équipée d'un **SoC ESP32** et d'un modem **LoRa** (*Long Range*).

Le premier laboratoire permet de préparer l'environnement de travail et de tester l'utilisation des capteurs connectés sur le bus I2C (température/humidité/luminosité/pression) et d'un afficheur. Pour nos développements et nos expérimentations nous utilisons le **IDE Arduino** comme outils de programmation et de chargement des programmes dans la mémoire flash de l'unité centrale.

Le deuxième laboratoire est consacré à la pris en main du modem WiFi intégré dans la carte principale (**Heltec ESP32-WiFi-LoRa**). La communication WiFi en mode station et un client permet d'envoyer les données des capteurs vers un serveur **WEB**. Dans notre cas nous utilisons le serveur de type **ThingSpeak**; (**`ThingSpeak.fr/com`**). Ces serveurs sont accessibles gratuitement, mais leur utilisation peut être limitée un temps (le cas de **`ThingSpeak.com`**).

Le troisième laboratoire permet de découvrir le protocole **MQTT** et d'expérimenter avec plusieurs exemples de **brokers** et **clients** basés sur différentes bibliothèques. Dans le laboratoire vous pouvez communiquer et échanger vos donnés IoT à distance via les broker externes.

Le quatrième laboratoire est orienté sur la technologie de communication **Bluetooth** – **Bluetooth Classic** (**BT**) et **Bluetooth Low Energy** (**BLE**). Vous allez expérimenter avec cette technologie à l'aide de vos smartphones. Vous allez développer une passerelle **Bluetooth-WiFI** sur votre **DevKit**.

Le cinquième laboratoire cible développement de simples **serveurs WEB** s'exécutant sur notre **DevKit**. Ces serveurs peuvent fonctionner en mode de station (**`STA`**) ou en mode Point d'Accès (**`AP`**). Il peuvent utiliser exclusivement la mémoire interne ou un contenu externe enregistré sur un **carte SD**.

Le sixième laboratoire introduit la programmation **OTA** (**Over The Air**). Ce mode de programmation permet de charger à distance les nouvelles applications via une connexion WiFi.

Après ces laboratoires préparatifs nous vous proposons plusieurs **laboratoires thématiques** qui permettrons de concevoir et développer différentes sortes d'applications IoT sous la forme de **mini-projets**. On vous présentera quelques exemples des architectures IoT dans les domaines d'application différents tels que la messagerie, le sécurité, l'environnement, le commerce,etc.

Vous pouvez vous inspirer de ces laboratoires ou de proposer d'autres de complexité comparable.

Nous vous fournirons (selon les disponibilités) des cartes d'extension et de capteurs/actionneurs nécessaires pour la réalisation de ces projets.

0.1 ESP32 Soc – une unité avancée pour les architectures IoT

ESP32 est une unité de microcontrôleur avancée conçue pour le développement d'architectures IoT. Un ESP32 intègre deux processeurs RISC 32-bit fonctionnant à 240 MHz et plusieurs unités de traitement et de communication supplémentaires, notamment un processeur ULP (**Ultra Low Power**), des modems WiFi / Bluetooth /BLE et un ensemble de contrôleurs E/S pour bus série (UART, I2C, SPI). , ..). Ces blocs fonctionnels sont décrits ci-dessous dans la figure suivante.

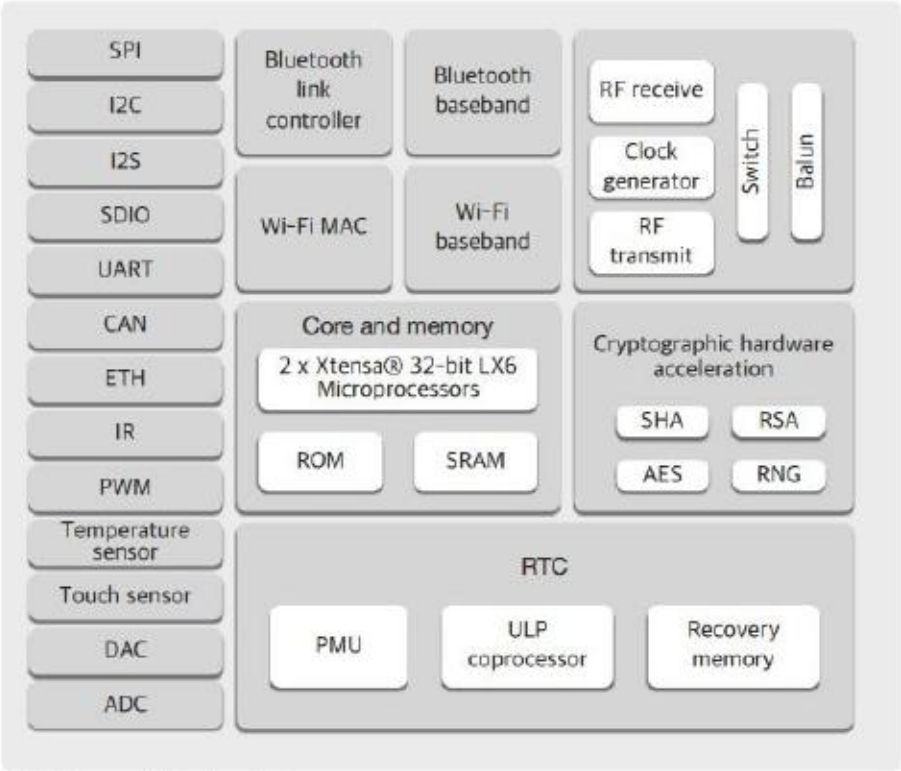


Figure 0.1 ESP32 SoC – architecture interne

0.2 Carte Heltec WiFi LoRa

De nos jours, les SoC ESP32 sont intégrés dans un certain nombre de cartes de développement qui incluent des circuits supplémentaires et des modems de communication. Notre choix est la carte **ESP32-Heltec WiFi-LoRa** qui intègre le modem LoRa **Semtech S1276/78** et (éventuellement) un écran . La figure suivante montre la carte **ESP32-Heltec WiFi-LoRa**

Heltec Wifi LoRa 32 *V2* 868MHz *top*

Heltec Wifi LoRa 32 *V2* 868MHz *bottom*

Figure 0.2 Carte MCU **ESP32-Heltec WiFi-LoRa**

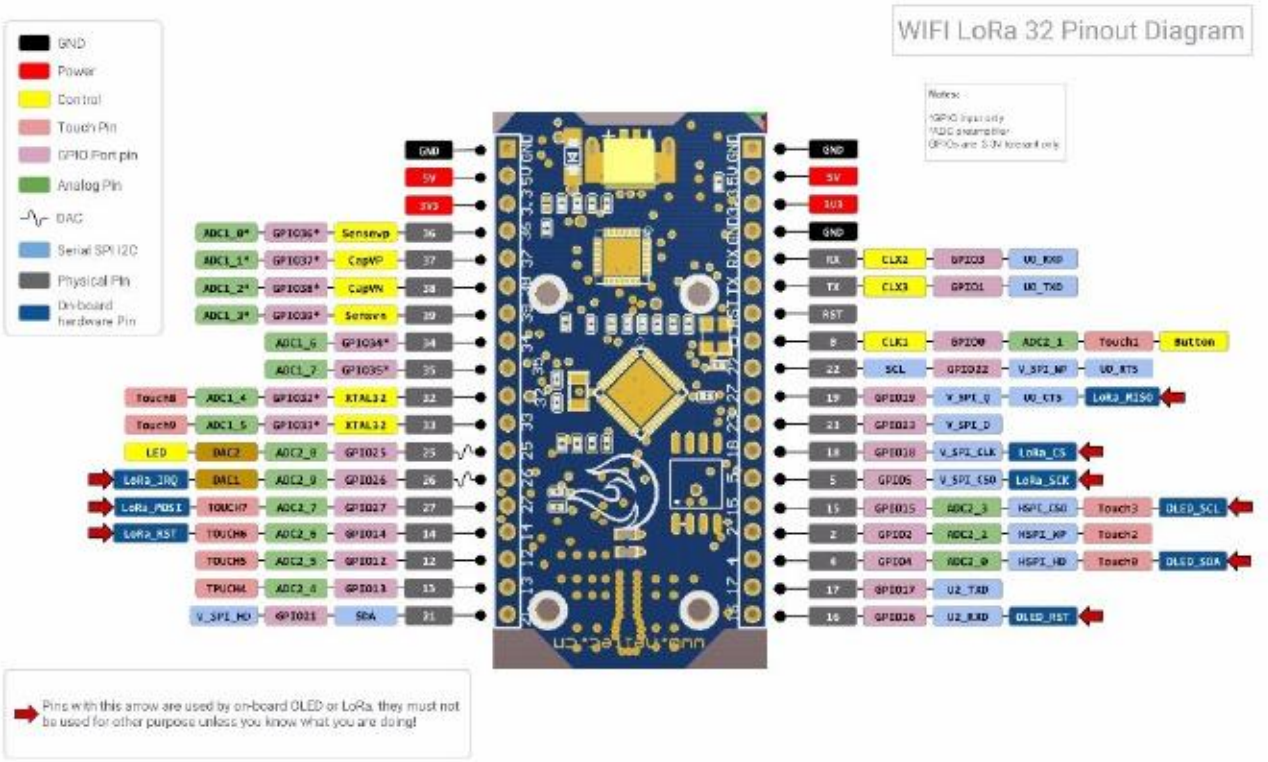


Figure 0.3. Carte **ESP32-Heltec WiFi-LoRa** et son *pin-out*

Comme nous pouvons le voir sur la figure ci-dessus, la carte Heltec expose des broches 2x18. Certaines de ces broches sont utilisées pour connecter le modem LoRa SX1276/8 (SPI) et l'afficheur OLED (I2C), d'autres broches peuvent être utilisées pour connecter des composants externes tels que des capteurs ou des actionneurs.

0.3 IoT DevKit une plate-forme de développement IoT

Une intégration efficace de la carte Heltec sélectionnée dans les architectures IoT nécessite l'utilisation d'une plate-forme de développement telle que IoT DevKit proposée par **SmartComputerLab**.
L'**IoTDevKit** est composé d'une carte de base et d'un grand nombre de cartes d'extension conçues pour l'utilisation efficace des bus de connexion et de tous les types de capteurs et d'actionneurs.

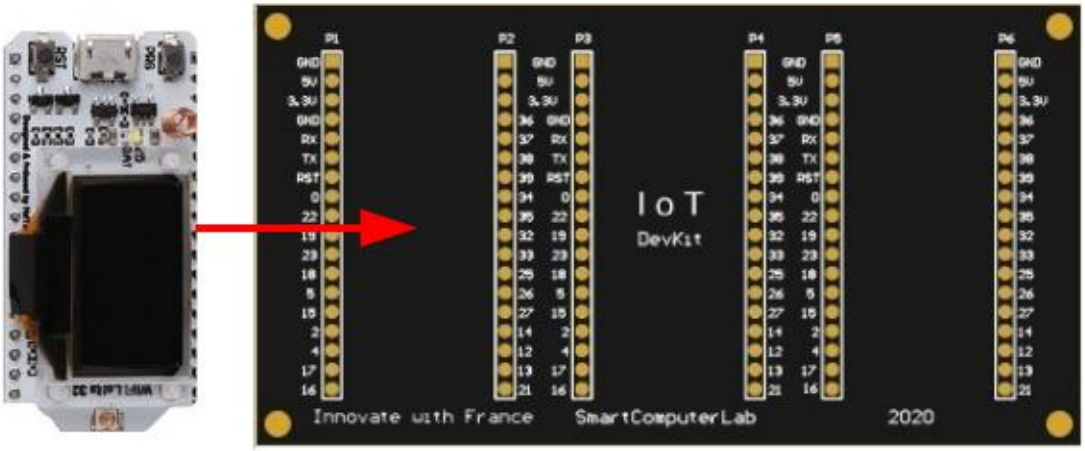
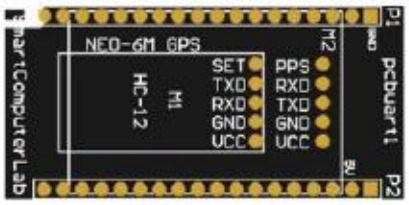


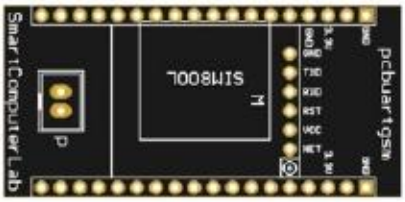
Figure 0.4. **IoT DevKit**: carte de base (type bridge) avec l'unité principale et les emplacements pour les cartes d'extension

0.3.1 Cartes d'extension simples – quelques exemples

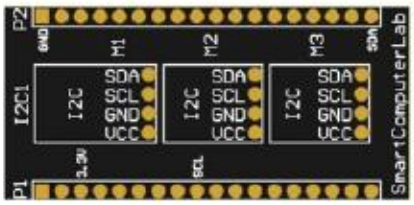
Figure 0.5. IoT DevKit: cartes d'extension, exemple d'implantation sur les d'extension **I2C** avec capteurs **BH1750** et **HTU21D** et une carte d'extension **UART** avec module GPS NEO-M6

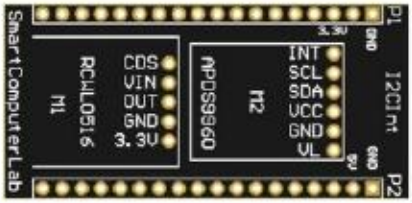
Cartes d'extension pour le bus **UART** (deux exemples):



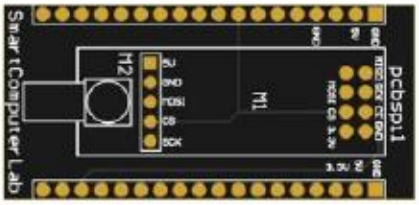


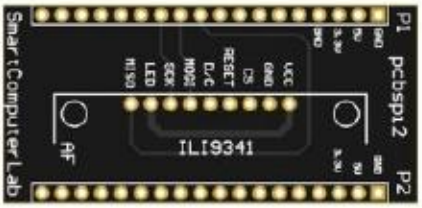
Cartes d'extension pour le bus **I2C** (deux exemples):



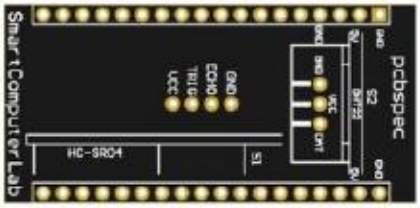


Cartes d'extension pour le bus **SPI** (deux exemples):





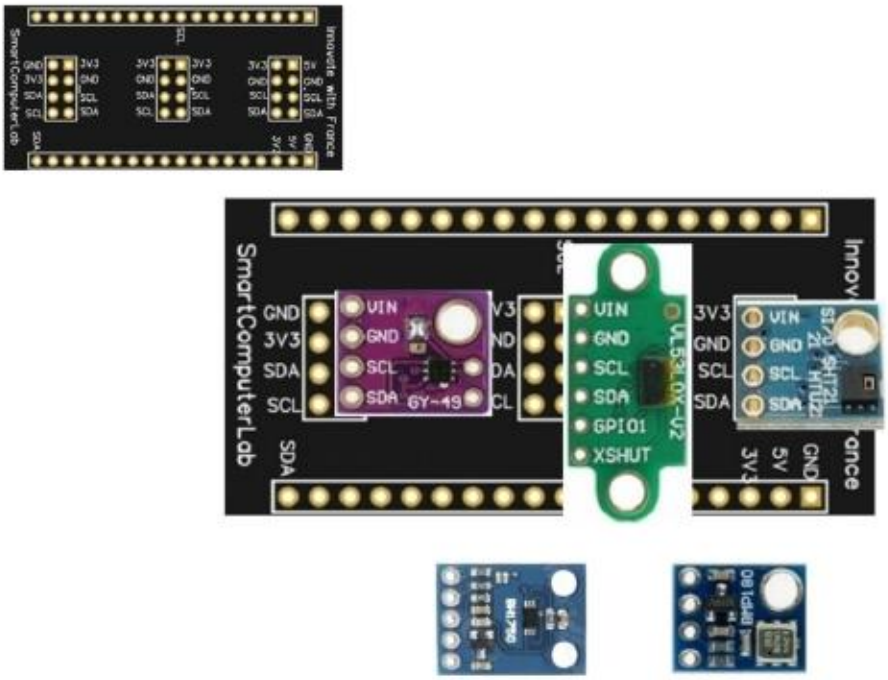
Cartes d'extension spécifiques (un exemple):



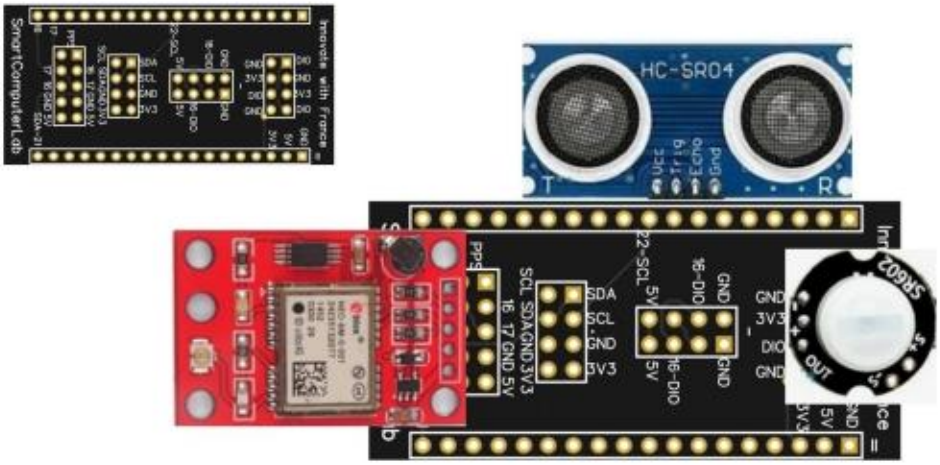
Cartes d'extension **I2C** avec capteurs **BH1750** et **HTU21D** et une carte d'extension **UART** avec module GPS NEO-M6

0.3.2 Cartes d'extension multi-capteurs – quelques exemples

Carte **multi-capteurs I2C**



Carte **multi-capteurs** : PIR, Echo, GPS (**UART**), **I2C**



0.4 L'installation de l'Arduino IDE sur un OS Ubuntu

Pour nos développements et nos expérimentations nous utilisons l'**IDE Arduino** comme outils de programmation et de chargement des programmes dans la mémoire flash de l'unité centrale.
Afin de pouvoir développer le code pour les application nous avons besoin d'un environnement de travail comprenant; un PC, un OS type Ubuntu 16.04LTS, l'environnement Arduino IDE, les outils de compilation/chargement pour les cartes ESP32 et les bibliothèque de contrôle pour les capteurs, actuateurs (par exemple **relais**), et les modems de communication.

Pour commencer mettez le système à jour par :

```
sudo apt-upgrade  et  sudo apt update
sudo apt-get install python-serial python3-serial
```

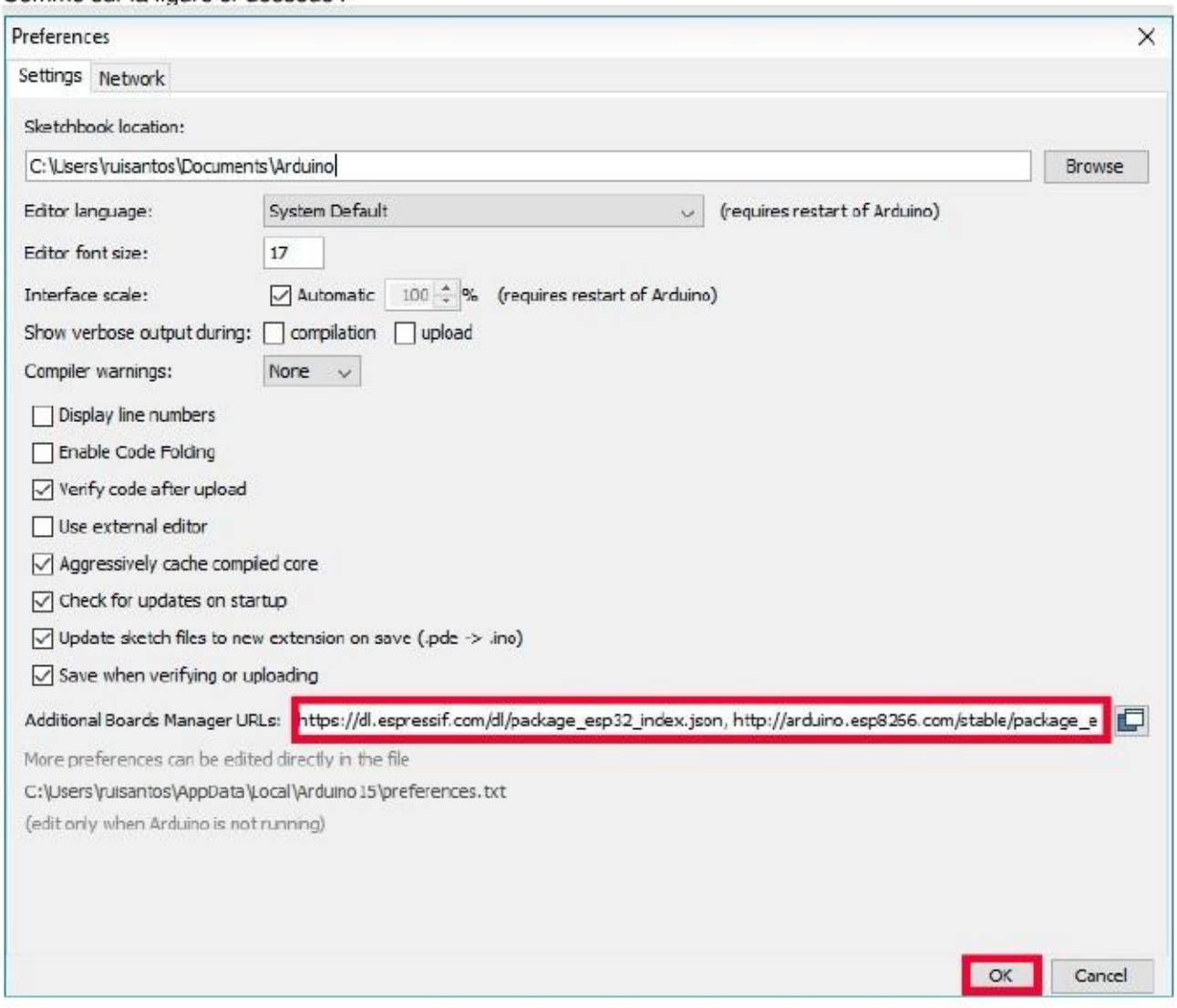
Ensuite il faut installer le dernier Arduino IDE à partir de **`https://www.arduino.cc`**

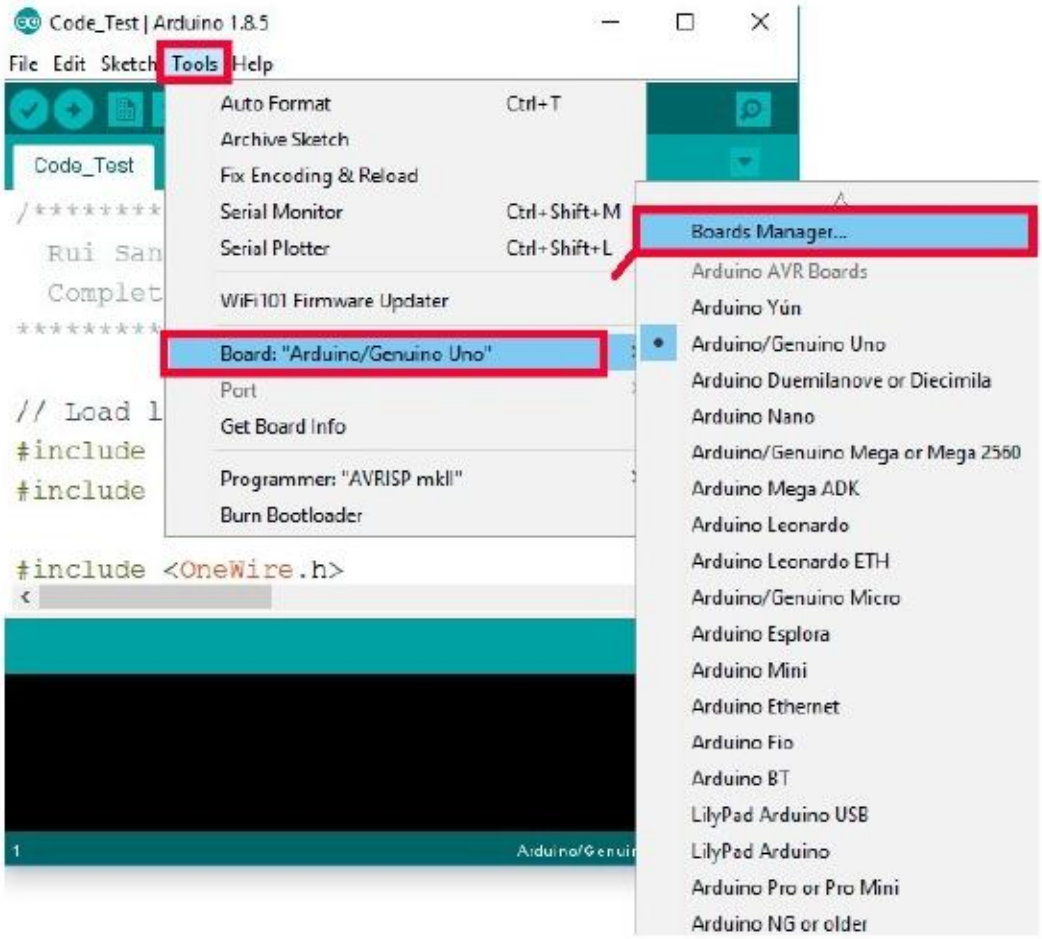
0.4.1 Installation des nouvelles cartes ESP32 et ESP8266

Après son installation allez dans les **Preferences** et ajoutez deux **Boards Manager URLs** (séparées par une virgule) :

`https://dl.espressif.com/dl/package_esp32_index.json`,
`http://arduino.esp8266.com/stable/package_esp8266com_index.json`

Comme sur la figure ci-dessous :





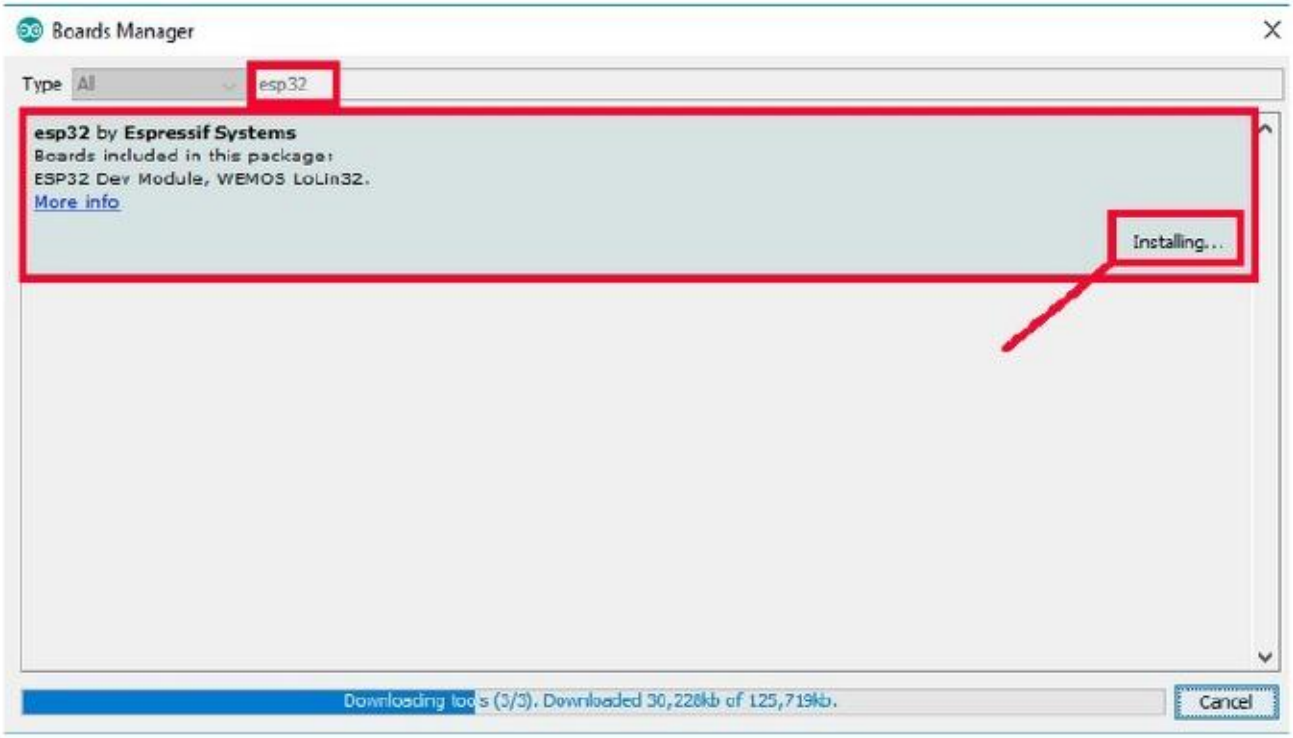


Figure 0.6. L'intégration des cartes **ESP32** et **ESP8266** dans l'environnement IDE Arduino

0.4.2 Préparation d'un code Arduino pour la compilation et chargement

La compilation doit être effectuée sur la carte **Heltec_WIFI_LoRa_32.** Elle doit être sélectionnée dans le menu **`Tools→Board`**

Avant la compilation il faut installer les bibliothèques nécessaires pour piloter différents dispositifs.
Par exemple l'image ci-dessous montre comment installer la bibliothèque **`U8g2`** qui pilote les écrans type **OLED**.

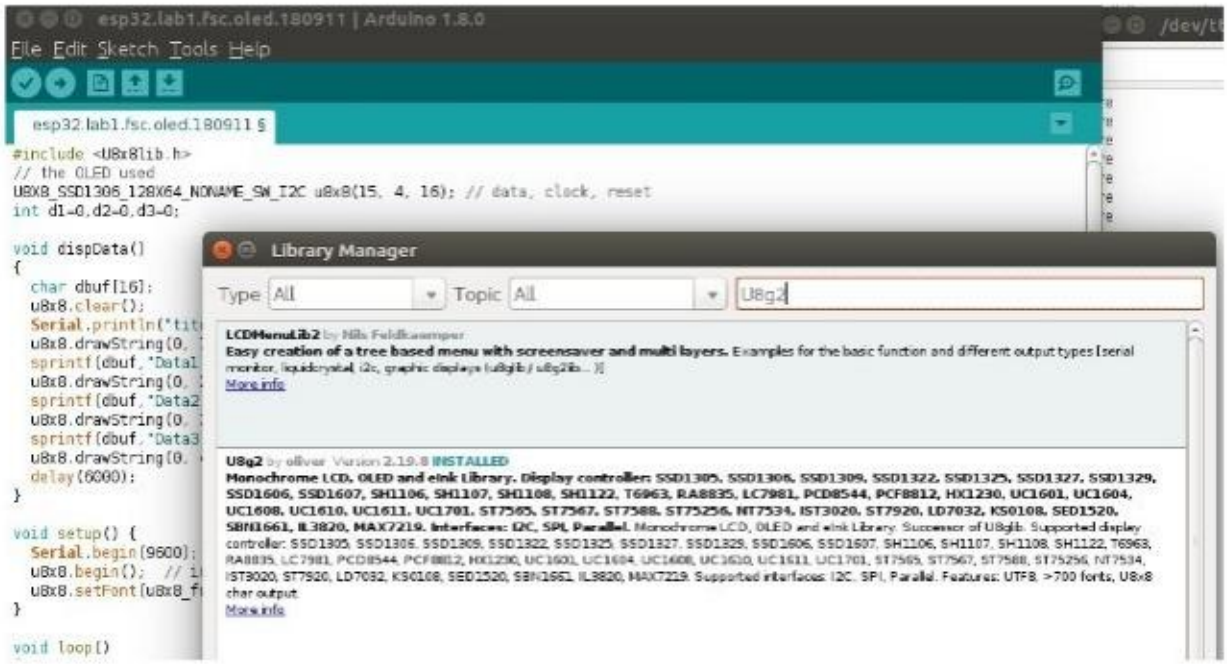


Figure 0.7. Intégration d'une bibliothèque Arduino (**U8g2**)

Laboratoire 1

1.1 Premier exemple – l'affichage des données sur l'écran OLED

Dans cet exercice nous allons simplement afficher un titre et 2 valeurs numériques sur l'écran **OLED** intégré dans la carte ESP32.

Figure 1.1 Ecran **OLED** de la carte ESP32 (Heltec WiFi LoRa 32)

Vous devez installer la bibliothèque **U8g2** (**https://github.com/olikraus/u8g2**). Cela peut être trouvé dans le gestionnaire de bibliothèque IDE Arduino. Ouvrez **Sketch> Include Library> Manage Libraries** et recherchez, puis installez **U8g2**. Notez que l'écran **OLED** est connecté sur un bus I2C avec trois broches : **SCL** – **clock**, **SDA** – **data/address** et **RESET**.

```
#include <U8x8lib.h>  // bibliothèque à charger a partir de
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15,4,16); // clock, data, reset
int d1=0,d2=0,d3=0;

void dispData()
{
  char dbuf[16];
  u8x8.clear();
  Serial.println("titre");
  u8x8.drawString(0,1,"titre"); // 0 – colonne (max 15), 1 – ligne (max 7)
  sprintf(dbuf,"Data1:%d",d1); u8x8.drawString(0,2,dbuf);
  sprintf(dbuf,"Data2:%d",d2); u8x8.drawString(0,3,dbuf);
  sprintf(dbuf,"Data3:%d",d3);u8x8.drawString(0,4,dbuf);
  delay(6000);
}

void setup() {
  Serial.begin(9600);
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
}

void loop()
{
d1++;d2+=2;d3+=4 ;
// ici appeler la fonction d'affichage
}
```

A faire

Compléter, compiler, et charger ce programme.

1.2 Deuxième exemple – capture et affichage des valeurs

1.2.1 Capture de la température/humidité par SHT21

Dans cet exercice nous allons lire les valeurs fournies par le capteur Température/Humidité **SHT21** et afficher les 2 valeurs sur l'écran OLED intégré dans la carte ESP32.
Le capteur **SHT21** doit être connecté sur le bus **I2C**, donc il nous faut une carte d'extension I2C avec une configuration des broches identique à celle du capteur (VIN correspond à 3V3).

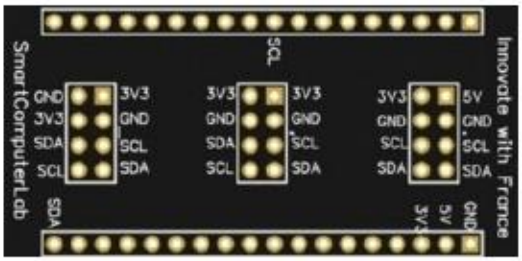


Figure 1.2. IoT DevKit : carte d'extension multi **I2C** et un capteur de température/humidité **SHT21**

Code Arduino

Code Arduino

```
// https://github.com/chaveiro/SHT2x-Arduino-Library

#include <Wire.h>
#include "SHT2x.h"

SHT2x SHT2x;

void setup()
{
  Wire.begin(21,22);
  SHT2x.begin();
  Serial.begin(9600);
}

void loop()
{
  uint32_t start = micros();
  Serial.print("Humidity(%RH): ");
  Serial.print(SHT2x.GetHumidity(),1);
  Serial.print("\tTemperature(C): ");
  Serial.print(SHT2x.GetTemperature(),1);

  uint32_t stop = micros();
  Serial.print("\tRead Time: ");
  Serial.println(stop - start);
  delay(1000);
}
```

1.2.2 Capture de la température/humidité par HTU21D

Dans cet exercice nous allons lire les valeurs fournies par le capteur Température/Humidité **HTU21D** et afficher les 2 valeurs sur l'écran OLED intégré dans la carte ESP32.
Le capteur **HTU21D** doit être connecté sur le bus **I2C**, donc il nous faut une carte d'extension **I2C** et l'emplacement avec la configuration des broches identique à celle du capteur.

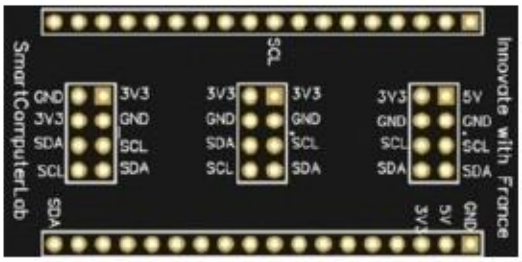


Figure 1.3 IoT DevKit : carte d'extension multi **I2C** et un capteur de température/humidité **HTU21D**

Code Arduino

```
#include <Wire.h>
#include "SparkFunHTU21D.h"
//Create an instance of the object
HTU21D sensor;

void setup()
{
  Serial.begin(9600);
  Serial.println("HTU21D Example!");
  myHumidity.begin();
}

void loop()
{
  float humd = sensor.readHumidity();
  float temp = sensor.readTemperature();
  Serial.print("Time:");
  Serial.print(millis());
  Serial.print(" Temperature:");
  Serial.print(temp, 1);
  Serial.print("C");
  Serial.print(" Humidity:");
  Serial.print(humd, 1);
  Serial.print("%");
  Serial.println();
  delay(1000);
}
```

Attention :

Pour le bon fonctionnement il faut **enlever le capteur SHT21**.

1.2.3 Capture de la luminosité par BH1750

Dans cet exercice utilisez la carte **I2C1** pour le capteur **de la luminosité BH1750**

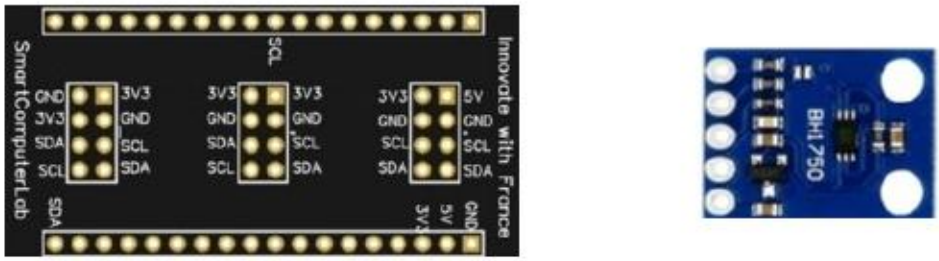


Figure 1.4. **IoT DevKit :** cartes d'extension multi - **I2C** pour le capteur Luminosité **BH1750**

Code Arduino

Attention : Avant la compilation il faut installer la bibliothèque **`BH1750.h`**

https://github.com/claws/BH1750

```
#include <Wire.h>
#include <BH1750.h>
BH1750 lightMeter;

void setup(){
  Wire.begin(21,22);  // carte d'extension I2C1 ou petite carte I2C
  Serial.begin(9600);
  lightMeter.begin();
  Serial.println("Running...");
  delay(1000);
}

void loop() {
  uint16_t lux = lightMeter.readLightLevel();
  delay(1000);
  Serial.print("Light: ");
  Serial.print(lux);
  Serial.println(" lx");
  delay(1000);
}
```

1.2.4 Capture de la luminosité par MAX44009

Dans cet exemple nous utilisons un capteur de luminosité type **MAX44009** (GY-49). Ce capteur est connectée comme le capteur BH1750 sur le bus I2C.
Nous communiquons avec ce capteurs **directement** par les trames **I2C** ce qui permet de mieux comprendre le **fonctionnement de ce bus**.

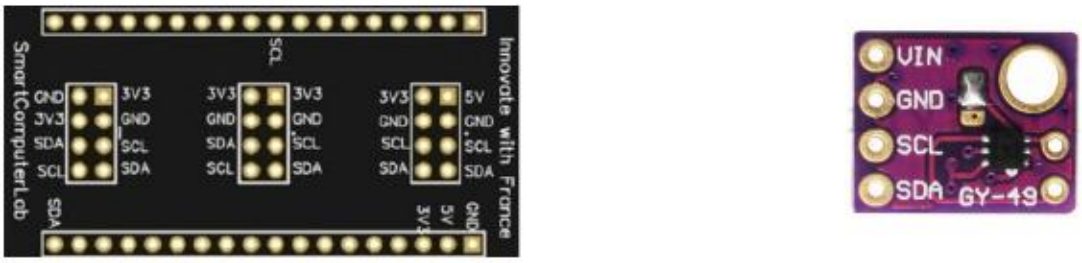


Figure 1.5 IoT DevKit : cartes d'extension **I2C** et un capteur de luminosité **MAX44009 (GY-49)**

Code Arduino :

```
#include<Wire.h>
#define Addr 0x4A

void setup()
{
Wire.begin(21,22);
Serial.begin(9600);
Wire.beginTransmission(Addr);
Wire.write(0x02); Wire.write(0x40);
Wire.endTransmission();
delay(300);
}

void loop()
{
unsigned int data[2];
Wire.beginTransmission(Addr);
Wire.write(0x03);
Wire.endTransmission();
Wire.requestFrom(Addr, 2); // Request 2 bytes of data
// Read 2 bytes of data luminance msb, luminance lsb
if (Wire.available() == 2)
{
data[0] = Wire.read(); data[1] = Wire.read();
}
// Convert the data to lux
int exponent = (data[0] & 0xF0) >> 4;
int mantissa = ((data[0] & 0x0F) << 4) | (data[1] & 0x0F);
float luminance = pow(2, exponent) * mantissa * 0.045;
Serial.print("Ambient Light luminance :"); Serial.print(luminance);
Serial.println(" lux");
delay(500);
}
```

1.2.5 Capture de la pression/température avec capteur BMP180

Le capteur BMP180 permet de capter la pression atmosphérique et la température. Sa précision est seulement de **+/- 100 Pa** et **+/-1.0C**.

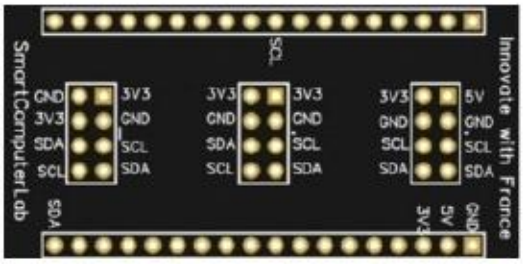


Figure 1.6. IoT DevKit : carte d'extension multi **I2C** et un capteur de pression/température **BMP180**

La valeur standard de la pression atmosphérique est :

101 325 Pa = 1,013 25 bar = 1 atm

Code Arduino - BMP180

```
#include <Arduino.h>
#include <Wire.h>
#include <BMP180I2C.h>
#define I2C_ADDRESS 0x77
BMP180I2C bmp180(I2C_ADDRESS);

void setup() {
      Serial.begin(9600);
      Wire.begin();
      if (!bmp180.begin())
      {
            Serial.println("check your BMP180 Interface and I2C Address.");
            while (1);
      }
      bmp180.resetToDefaults();
      //enable ultra high resolution mode for pressure measurements
      bmp180.setSamplingMode(BMP180MI::MODE_UHR);
}

void loop() {
      delay(1000);
      if (!bmp180.measureTemperature())
      {
      Serial.println("could not start temperature measurement");
            return;
      }
      //wait for hasValue() returned true.
      do
      {
            delay(100);
      } while (!bmp180.hasValue());
      Serial.print("Temperature: ");
      Serial.print(bmp180.getTemperature());
      Serial.println(" degC");
```

```
      if (!bmp180.measurePressure())
      {
      Serial.println("could not start perssure measurement");
            return;
      }
      do
      {
            delay(100);
      } while (!bmp180.hasValue());
      Serial.print("Pressure: ");
      Serial.print(bmp180.getPressure());
      Serial.println(" Pa");
}
```

A faire

1. Complétez les programmes ci-dessus afin d'afficher les données de la Luminosité sur l'écran OLED

2. Développer une application avec la capture et l'affichage de l'ensemble de trois données : Température/Humidité et Luminosité.

3. Développer une application avec la capture et l'affichage de l'ensemble de trois données : Température/Humidité et Pression (BMP180).

1.2.6 Détection de mouvement avec capteur PIR (SR602)

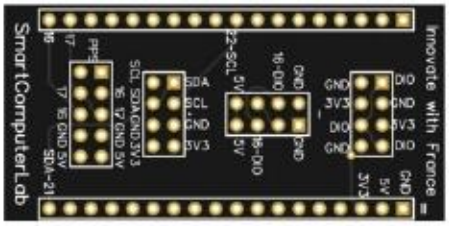


Figure 1.7 Carte **multi-capteurs (I2C, UART, one-wire)** et capteur **PIR (SR602)**

Code Arduino

```
#include <U8x8lib.h>  // bibliothèque à charger a partir de
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15,4,16); // clock, data, reset
int d1=0,d2=0,d3=0;
#define PIR 16  // problème ?
bool MOTION_DETECTED = false;
int counter=0;

void dispData()
{
  char dbuf[16];
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);u8x8.clear();
  u8x8.drawString(0,1,"Motion detected");
  sprintf(dbuf,"Count:%d",counter); u8x8.drawString(0,3,dbuf);
  delay(100);
}

void pinChanged() { MOTION_DETECTED = true; }

void setup() {
Serial.begin(9600);pinMode(PIR,INPUT);
attachInterrupt(PIR, pinChanged, RISING);
}

void loop()
{
  int i=0;
  if(MOTION_DETECTED){ Serial.println("Motion detected.");
      delay(1000);counter++;
      MOTION_DETECTED = false;
      Serial.println(counter);
      dispData();
      pinMode(PIR,INPUT) attachInterrupt(PIR, pinChanged, RISING);
      }
}
```

A faire :

1. Analyser le programme ci-dessus

Pourquoi **doit on réinitialiser l'écran et l'interruption à chaque pas d'utilisation.**

1.2.7 Mesure de distance avec capteur VL53L0X

Le **VL53L0X** est un module de **télémétrie laser** à **temps de vol** (**ToF**) offrant une mesure de distance précise quelles que soit la cible contrairement aux technologies conventionnelles. Il peut mesurer des distances absolues jusqu'à 2m. **(le capteur fonctionne mieux avec Vcc=5V)**

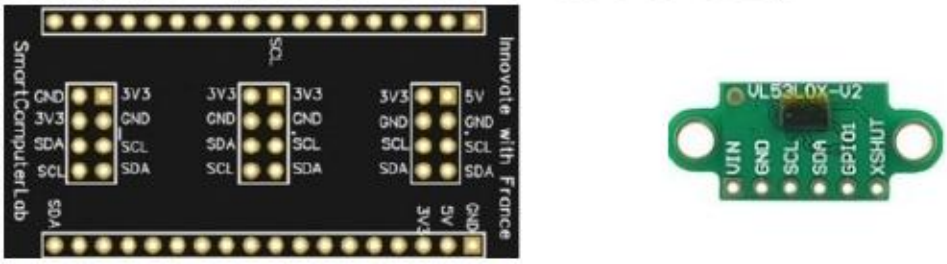


Figure 1.8 Carte multi-capteurs (I2C) et capteur **VL53L0X (laser-ToF)**

Fonctions :

```
setMode(ModeState mode, PrecisionState precision)
```

- `Continuous--->Continuous measurement model`
- `Single------->Single measurement mode`
- `High--------->Accuracy of 0.25 mm`
- `Low---------->Accuracy of 1 mm`

```
void start() - This function is used to enabled VL53L0X
float getDistance()  - This function is used to get the distance
uint16_t getAmbientCount() - This function is used to get the ambient count
uint16_t getSignalCount() - This function is used to get the signal count
uint8_t getStatus(); - This function is used to get the status
void stop() - This function is used to stop measuring
```

Le code

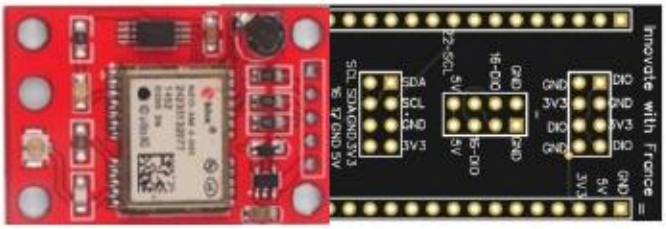
```
#include "Arduino.h"
#include "Wire.h"
#include "DFRobot_VL53L0X.h"

DFRobotVL53L0X sensor;

void setup() {
  Serial.begin(9600);
  Wire.begin(21,22);    // SDA, SCL
  sensor.begin(0x50);//Set I2C sub-device address
  //Set to Back-to-back mode and high precision mode
  sensor.setMode(Continuous,High);
  //Laser rangefinder begins to work
  sensor.start();
}

void loop()
{
  Serial.print("Distance: ");
  Serial.print(sensor.getDistance());Serial.println("mm");

  delay(500); // delay does not affect the measurement accuracy
}
```

A faire :

1. Testez le programme ci-dessus et ajoutez l'affiche sur l'écran OLED

1.2.8 Temps réel et positionnement GPS avec NEO-6M

NEO-6M est un module de **global positioning system** (GPS). Il est très populaire, économique et de hautes performances avec une antenne patch en céramique, une puce de mémoire intégrée et une batterie de secours est intégrée sur le module.

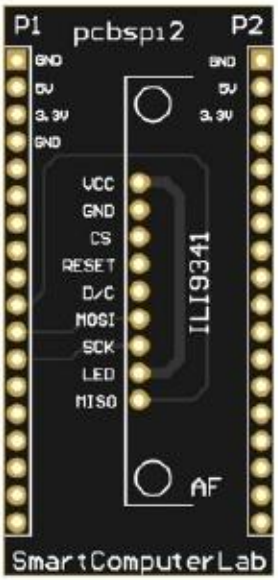


Le code

```
#include <TinyGPS++.h>
HardwareSerial uart(2);
TinyGPSPlus gps;
void setup(){
  Serial.begin(9600);
  uart.begin(9600, SERIAL_8N1, 17, 16);}

  int i=0,ihour;
  char gpst[2048],ttime[16],ftime[16],hour[8],nhour[2];
  char *ptr=NULL;

  void loop(){
    while (uart.available() > 0)
      {
        gps.encode(uart.read());
        if (gps.location.isUpdated())
          {
            Serial.print("Latitude= ");
            Serial.print(gps.location.lat(), 6);
            Serial.print(" Longitude= ");
            Serial.println(gps.location.lng(), 6);
            Serial.print("Date:");
            Serial.print(gps.date.month());
            Serial.print("/");
            Serial.print(gps.date.day());
            Serial.print("/");
            Serial.print(gps.date.year());
            Serial.print("   Time:");
            if (gps.time.hour() < 10)
              Serial.print("0");
            Serial.print(gps.time.hour());
            Serial.print(":");
            if (gps.time.minute() < 10)
              Serial.print("0");
            Serial.print(gps.time.minute());
            Serial.print(":");
            Serial.println(gps.time.second());
          }
        }
      }
```

1.2.9 Affichage sur l »écran TFT - IL9341

L'exemple suivant montre l'utilisation d'une carte pour les écrans **SPI-TFT** type **IL9341**. Nous utilisons ici la bibliothèque **`TFT_eSPI.h`** avec une interface **SPI configurable** dans le fichier **`User_Setup.h`**



Un exemple du code :

Attention : Il faut préparer les connections sur le bus SPI dans le fichier User_Setup.h présent dans votre répertoire **`TFT_eSPI-master`** dans le **`libraries`** de **`Arduino`**.

```
#include <TFT_eSPI.h> // Graphics and font library for ILI9341 driver chip
#include <SPI.h>
// to be defined in User_Setup.h
//#define TFT_MISO 19 // Not connected
//#define TFT_MOSI 23
//#define TFT_SCLK 18
//#define TFT_CS 5 // Chip select control pin
//#define TFT_DC 2 // Data Command control pin
//#define TFT_RST 4 // Reset pin (could connect to RST pin)
// GND tp GND , VCC to 3.3V and LED to 3.3V
#define TFT_GREY 0x5AEB // New colour
TFT_eSPI tft = TFT_eSPI(); // Invoke library
void setup(void) {
Serial.begin(9600);
tft.init();
tft.setRotation(3);
}

int count=0;
char disp[12];

void loop() {
Serial.println("in the loop");
// Fill screen with grey
// tft.fillScreen(TFT_GREY);
tft.fillScreen(TFT_BLACK);
// Set "cursor" at top left corner of display (0,0) and select font 2
// or stay on the line is there is room for the text with tft.print)
tft.setCursor(0, 0, 4);
```

```
// Set the font colour to be white with a black background, set text size
multiplier to 1
if(count>12)tft.setTextColor(TFT_RED,TFT_BLACK);
else tft.setTextColor(TFT_GREEN,TFT_BLACK);
tft.setTextSize(8);
// We can now plot text on screen using the "print" class
sprintf(disp,"%3.3d",count); count++;
tft.println(disp);
tft.setCursor(20, 150, 2);
tft.setTextSize(3);
if(count>12) tft.println("Porte fermee");
else tft.println("Porte ouverte");

tft.setCursor(200, 220, 2);
// Set the font colour to be yellow with no background, set to font 7
tft.setTextColor(TFT_YELLOW); tft.setTextSize(1);
tft.println("SmartComputerLab");
// Set the font colour to be red with black background, set to font 4
//tft.setTextColor(TFT_RED,TFT_BLACK);
//tft.setTextFont(4);
//tft.println(3735928559L, HEX); // Should print DEADBEEF
// Set the font colour to be green with black background, set to font 4
//tft.setTextColor(TFT_GREEN,TFT_BLACK);
//tft.setTextFont(4);
//tft.println("Groop");
//tft.println("I implore thee,");
// Change to font 2
//tft.setTextFont(2);
//tft.println("my foonting turlingdromes.");
//tft.println("And hooptiously drangle me");
//tft.println("with crinkly bindlewurdles,");
// This next line is deliberately made too long for the display width to test
// automatic text wrapping onto the next line
//tft.println("Or I will rend thee in the gobberwarts with my
blurglecruncheon,see if I don't!");
// Test some print formatting functions
//float fnumber = 123.45;
// Set the font colour to be blue with no background, set to font 4
//tft.setTextColor(TFT_BLUE);
//tft.setTextFont(4);
//tft.print("Float = "); tft.println(fnumber);
// Print floating point number
//tft.print("Binary = "); tft.println((int)fnumber, BIN); // Print as integer
value in binary
//tft.print("Hexadecimal = "); tft.println((int)fnumber, HEX); // Print as
integer number in Hexadecimal
delay(3000);
}
```

A faire :

1. Modifier le contenu du programme ci-dessus pour afficher , par exemple:
`IoT - Lab1` sur votre écran TFT
2. Ajouter un **capteur** , par exemple température-humidité sur la carte multi-capteurs **I2C** et afficher les valeurs captées
3. Ajouter le **modem GPS** , la carte multi-capteurs **PIR, UART** et afficher les valeurs captées

Laboratoire 2 – communication en WiFi et serveur `ThingSpeak.fr`

2.1 Introduction

Dans ce laboratoire nous allons nous intéresser aux moyens de la **communication** et de la **présentation** (**stockage**) des résultats. Etant donné que le SoC ESP32 intègre les modems de WiFi et de Bluetooth nous allons étudier la communication par **WiFi**.

Principalement nous avons deux modes de fonctionnement **WiFi** – mode station **STA**, et mode point d'accès **`softAP`**.
Dans le mode **STA** nous pouvons également tester l'état de fonctionnement : connecté ou de-connecté. Dans le mode **AP** nous pouvons déterminer l'adresse IP, le masque du réseau, et le canal de communication WiFi (1-13).
Un troisième mode appelé **ESP-NOW** permet de communiquer entre les cartes directement (sans un point d'accès), donc plus rapidement et à plus grande distance, par le biais de trames physiques **MAC**.

Une fois la communication WiFi est opérationnelle nous pouvons choisir un des multiples protocoles pour transmettre nos données : **UDP**, **TCP**, **HTTP/TCP**, ..Par exemple la carte peut fonctionner comme client ou serveur WEB. Dans une application fonctionnant comme un client WEB nous allons contacter un serveur externe type **`ThingSpeak`** pour y envoyer et stocker nos données.
Bien sur nous aurons besoin de bibliothèques relatives à ces protocoles.

2.1.1 Un programme de test – scrutation du réseau WiFi

Pour commencer notre étude de la communication WiFi essayons un exemple permettant de scruter notre environnement dans la recherche de point d'accès visibles par notre modem.

```
#include "WiFi.h"
#include <U8x8lib.h>
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16);

void setup()
{
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
  u8x8.begin(); u8x8.setFont(u8x8_font_chroma48medium8_r);
}

void loop()
{
  int n = WiFi.scanNetworks();
  if(n==0) u8x8.drawString(0,0, "Searching !");
  else {
   u8x8.drawString(0, 0, "Networks found: "); u8x8.clear();
   for(int i=0;i<n;++i) {
      char currentSSID[16];
      memset(currentSSID,0x00,16);
      WiFi.SSID(i).toCharArray(currentSSID,16);
      u8x8.drawString(0,i+1,currentSSID);
     }
  }
}
```

2.2 Mode WiFi – STA, client WEB et serveur `ThingSpeak`

Une connexion WiFi permet de créer une application avec un client WEB. Ce client WEB peut envoyer les requêtes HTTP vers un serveur WEB.
Dans nos laboratoires nous allons utiliser un serveur IoT externe type **ThingSpeak**. **ThingSpeak** est un serveur «*open source*» qui peut être installé sur un PC
ThingSpeak est une plate-forme open source «Internet des objets» pour stocker et récupérer des données d'objets en utilisant HTTP sur Internet. Avec **ThingSpeak**, vous pouvez créer des applications de journalisation de capteurs, des applications de suivi de localisation et un réseau social d'objets avec des mises à jour de statut.

2.2.1 Envoi des données sur ThingSpeak

Nous pouvons donc envoyer les requêtes HTTP (par exemple en format **`GET`** et les données attachées à **URL**) sur le serveur **`ThingSpeak.com`** de **Matlab** ou **`ThingSpeak.fr`** de **SmartComputerLab**.
La préparation de ces requêtes peut être laborieuse, heureusement il existe une bibliothèque **`ThingSpeak.h`** qui permet de simplifier cette tâche.
Il faut donc installer la bibliothèque **`ThingSpeak.h`**. On peut la trouver à l'adresse suivante :

`https://github.com/mathworks/thingspeak-arduino`

Attention
Si vous voulez travailler avec un serveur **ThingSpeak** autre que **`ThingSpeak.com`** le fichier **`ThingSpeak.h`** doit être modifié pour y mettre l'**adresse IP** et le **numéro de port** correspondant.

Exemple de modification du fichier **`ThingSpeak.h`** pour **`ThingSpeak.fr`** :

```
//#define THINGSPEAK_URL "api.thingspeak.com"
//#define THINGSPEAK_PORT_NUMBER 80
#define THINGSPEAK_URL "90.49.254.215"
#define THINGSPEAK_PORT_NUMBER 443
```

Après l'inscription sur le serveur **ThingSpeak** vous créez un **channel** composé de max 8 **fields**. Ensuite vous pouvez envoyer et lire vos données dans ce **fields** à condition de fournir la clé d'écriture associée au **channel**.
Une écriture/envoi de plusieurs valeurs en virgule flottante est effectué comme suit :

```
ThingSpeak.setField(1, sensor[0]);  // préparation du field1
ThingSpeak.setField(2, sensor[1]);  // préparation du field2
```

puis

```
ThingSpeak.writeFields(myChannelNumber[1], myWriteAPIKey[1]);  // envoi
```

Voici le début d'un tel programme (déclarations et **`setup()`**):

```
#include <WiFi.h>
#include "ThingSpeak.h"
WiFiClient  client;
char ssid[] = "PhoneAP";             //  your network SSID (name)
char pass[] = "smartcomputerlab";                  // your network passw
unsigned long myChannelNumber = 1;  // your channel number
const char * myWriteAPIKey = "HEU64K3PGNWAAAA4";  // your API write key
IPAddress ip;

void setup() {
 Serial.begin(9600);
WiFi.disconnect(true);  // effacer de l'EEPROM WiFi credentials
delay(1000);
WiFi.begin(ssid, pass);
delay(1000);
```

```
    while (WiFi.status() != WL_CONNECTED) {
    delay(500); Serial.print(".");
    }
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");Serial.println(ip);
  Serial.println("WiFi setup ok");
  delay(1000);
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
}
```

Pour simplifier l'exemple dans la fonction de la boucle principale **loop()** nous allons envoyer les données d'un compteur.

```
int tout=20000;  // en millisecondes
float luminosity=100.0, temperature=10.0;

void loop()
{
ThingSpeak.setField(1, luminosity);  // préparation du field1
ThingSpeak.setField(2, temperature);  // préparation du field1
    while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
    }
ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
luminosity++;
temperature++;
delay(tout);
}
```

Attention : pour le serveur **ThingSpeak.com** la valeur minimale de **tout** est 20 secondes.

2.2.2 Réception des données à partir de ThingSpeak

Les données envoyées vers **ThingSpeak** sont stockées dans sa base **Mysql**. Chaque donnée est marquée par la date de réception et sa valeur en format **JSON**.
Le programme suivant montre comment récupérer les données à partir de ThinSpeak. Dans cet exemple nous lisons simplement les dernières valeurs enregistrées dans 2 champs du canal.
Pour pouvoir récevoir les données privées, il faut associer la clé de lecture (**myReadAPIKey**) à chaque lecture des données.

```
#include <WiFi.h>
#include "ThingSpeak.h"
WiFiClient  client;
const char* ssid = "Livebox-08B0";
const char* pass = "G79ji6dtEptVTPWmZP";
unsigned long myChannelNumber =  1088701;  // IoTDevKit1
const char * myWriteAPIKey = "VKRHS9PGD9K1HJ6R";
const char * myReadAPIKey ="OP2YX22NPFCO763Z";
IPAddress ip;

void setup() {
 Serial.begin(9600);
WiFi.disconnect(true);  // effacer de l'EEPROM WiFi credentials
delay(1000);
WiFi.begin(ssid, pass);
delay(1000);
    while (WiFi.status() != WL_CONNECTED) {
```

```
          delay(500); Serial.print(".");
    }
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  Serial.println("WiFi setup ok");
  delay(1000);
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
}

int tout=20000;  // en millisecondes
float luminosity=0, temperature=0;

void loop()
{
while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
    }
luminosity = ThingSpeak.readIntField(1088701,1,myReadAPIKey );  // key NULLL for
public channel view
delay(tout);
temperature = ThingSpeak.readIntField(1088701,2,myReadAPIKey );
delay(tout);
Serial.println(luminosity);Serial.println(temperature);
}
```

2.2.3 Accès WiFi – votre Phone (PhoneAP) ou un routeur WiFi-4G

La connexion WiFi nécessite la disponibilité d'un point d'accès. Un tel point d'accès peut être créé par votre smarthphone avec une application Hot-Spot mobile ou un routeur dédié type B315 de Huawei

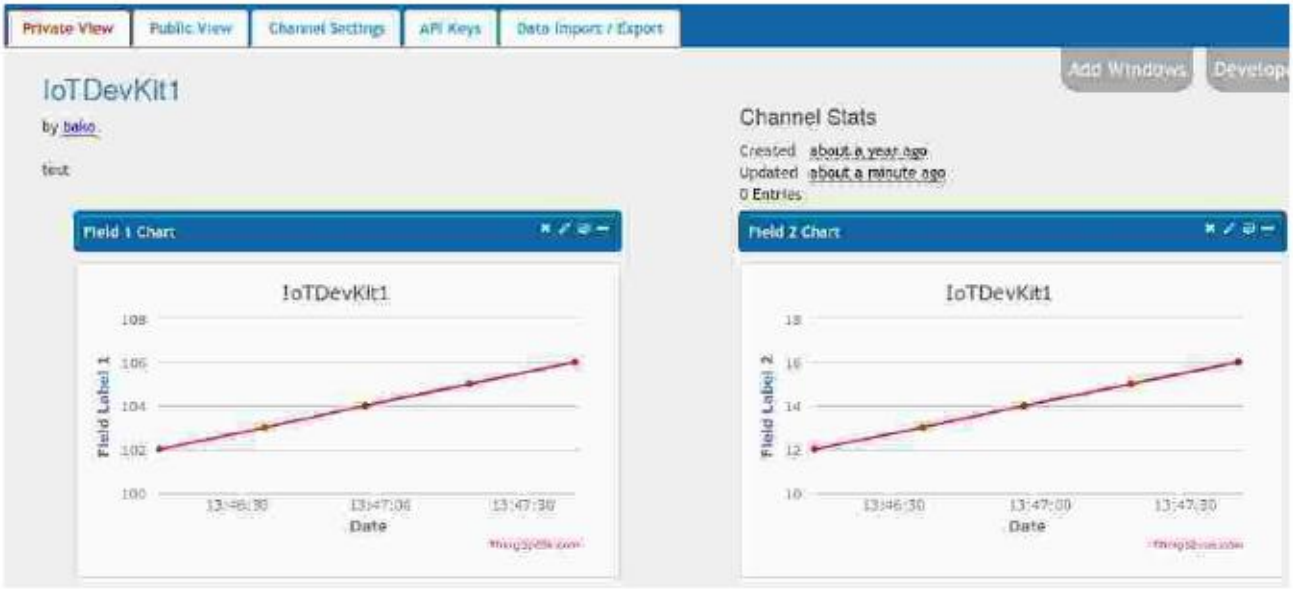


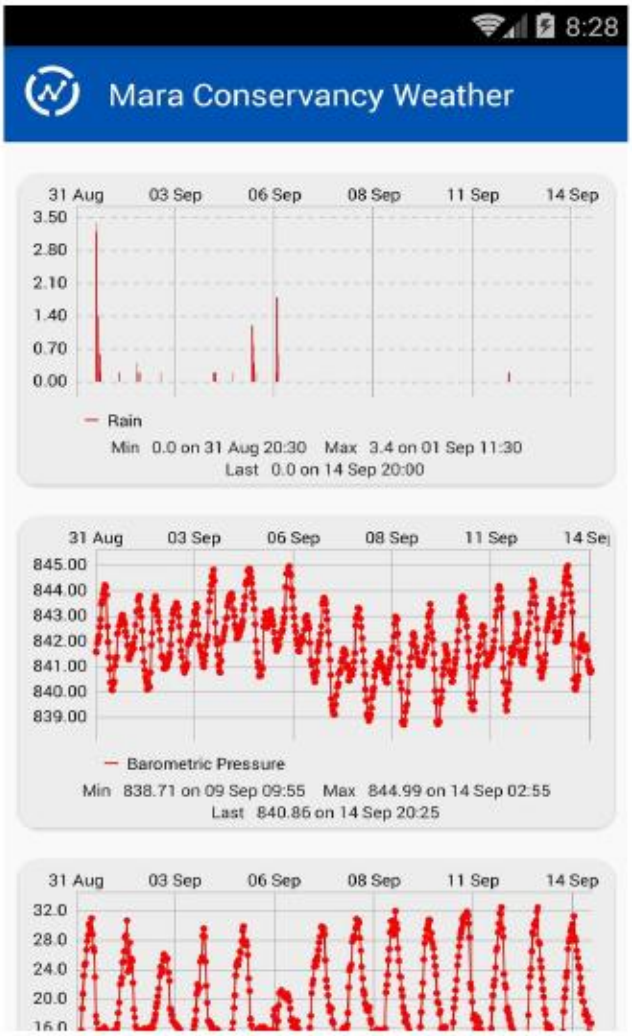
Figure 2.1. **ThingSpeak** : l'affichage des données envoyées sur le serveur type **ThingSpeak**

2.2.4 `ThingView` - `ThingSpeak` viewer (Android)

ThingView vous permet de visualiser vos chaînes ThingSpeak de manière simple, entrez simplement l'ID de chaîne et vous êtes prêt à partir.

Pour les **canaux publiques**, l'application respectera vos paramètres Windows: couleur, échelle de temps, type de graphique et nombre de résultats. La version actuelle prend en charge les graphiques en courbes et en colonnes, les graphiques splines sont affichés sous forme de graphiques en courbes.

Pour les **canaux privés**, les données seront affichées en utilisant les paramètres par défaut, car il n'y a aucun moyen de lire les paramètres de fenêtres privées avec la clé api uniquement.



A faire

1. Créer un canal **`ThingSpeak`** puis récupérer les paramètres du canal : *number* et *write key*.
2. Intégrer l'utilisation des capteurs de Température/Humidité et de la Luminosité.
3. Envoyer les données cycliquement (par exemple toutes les 30 seconde) vers le serveur **`ThingSpeak`**.
4. Recevoir les données cycliquement (par exemple toutes les 30 seconde) à partir du serveur **`ThingSpeak`**. et les afficher sur l'écran **OLED**.

Important : Ce travail peut être effectué à **distance** entre **2 membres du binôme** qui connaissent le même canal ThingSpeak avec ses clés d'écriture et de lecture.

2.3 Mode WiFi – STA, avec une connexion à `eduroam`

La carte ESP32 permet également d'établir un lien de communication WiFi avec un point d'accès type **`eduroam`** (**enterprise AP**).
Ci-dessous le code permettant d'effectuer une telle connexion. Comme dans l'exemple précédent le programme communique avec un serveur **ThingSpeak**.

2.3.1 Envoi des données sur ThingSpeak avec point d'accès eduroam

```
#include "ThingSpeak.h"
#include "esp_wpa2.h"
#include <WiFi.h>
#define EAP_IDENTITY "bakowski-p@univ-nantes.fr" //eduroam login -->
identity@youruniversity.domain
#define EAP_PASSWORD "..." //your password
String line; //variable for response
const char* ssid = "eduroam"; // Eduroam SSID

WiFiClient  client;
unsigned long myChannelNumber =  1234;  // no de votre canal ThingSpeak
const char * myWriteAPIKey = "9KLC1D8XJUGKHRO6"; // code d'ecriture

float temperature=0.0,humidity=0.0,tem,hum;

void setup() {
  Serial.begin(9600);
  delay(10);Serial.println();
  Serial.print("Connecting to network: ");
  Serial.println(ssid);
  WiFi.disconnect(true);  //disconnect form wifi to set new wifi connection
  WiFi.mode(WIFI_STA);
  esp_wifi_sta_wpa2_ent_set_identity((uint8_t *)EAP_IDENTITY,
strlen(EAP_IDENTITY)); //provide identity
  esp_wifi_sta_wpa2_ent_set_username((uint8_t *)EAP_IDENTITY,
strlen(EAP_IDENTITY)); //provide username
  esp_wifi_sta_wpa2_ent_set_password((uint8_t *)EAP_PASSWORD,
strlen(EAP_PASSWORD)); //provide password
  esp_wpa2_config_t config = WPA2_CONFIG_INIT_DEFAULT();
  esp_wifi_sta_wpa2_ent_enable(&config);

  WiFi.begin(ssid); //connect to Eduroam function
  WiFi.setHostname("ESP32Name"); //set Hostname for your device - not neccesary
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address set: ");
  Serial.println(WiFi.localIP()); //print LAN IP
  ThingSpeak.begin(client);
}

void loop() {
  delay(5000);
  if (WiFi.status() != WL_CONNECTED) { //if we lost connection, retry
    WiFi.begin(ssid);
    delay(500);
}
  Serial.println("Connecting to ThingSpeak.fr or ThingSpeak.com");
  // modify the ThingSpeak.h file to provide correct URL/IP/port
  WiFiClient client;
```

```
  delay(1000);
  ThingSpeak.begin(client);
  delay(1000);
  Serial.println("Fields update");
  ThingSpeak.setField(1, temperature);
  ThingSpeak.setField(2, humidity);
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(6000);
  temperature+=0.1;
  humidity+=0.2;
  tem =ThingSpeak.readFloatField(myChannelNumber,1);
  Serial.print("Last temperature:");
  Serial.println(tem);
  delay(6000);
  hum =ThingSpeak.readFloatField(myChannelNumber,2);
  Serial.print("Last humidity:");
  Serial.println(hum);
  delay(6000);
}
```

2.3.2 Réception des données partir du ThingSpeak avec point d'accès eduroam

Le programme ci-dessous permet de récupérer les données envoyées par notre programme précédent (émetteur) Les données sont lues sur le serveur **ThingSpeak.fr**, dans le canal pré-configuré puis elle sont affichées sur l'écran **OLED**. Le temps réel est lu par le biais du protocole **NTP**, dont les données sont portées par le protocole **UDP**.
Malheureusement le port numéro **123** de **UDP** n'est pas (normalement) ouvert sur un lien **eduroam**. L'affichage donnée simplement la durée de fonctionnement du récepteur.

```
#include <WiFi.h>
#include "esp_wpa2.h"
#include "ThingSpeak.h"
#include <Wire.h>
#include "OLED.h"
#include <NTPClient.h>
#include <WiFiUdp.h>
OLED display(21, 22);  // SDA-D2,SCL-D1 ESP32 - 22-SCL, 21-SDA
const char* ssid = "eduroam"; // your ssid
#define EAP_ID "bakowski-p"
#define EAP_USERNAME "bakowski-p"
#define EAP_PASSWORD ".." //removed for obvious reasons
int status = WL_IDLE_STATUS;
WiFiClient  client;
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP);
unsigned long myChannelNumber =  2;  // numero du canal
const char * myWriteAPIKey = "WT4X574BN1FGDZFW "; // clé de lecture
float temp=0.0,humi=0.0,inter=0.0;

void mydispfloat(int clr,int line,float field)
{
  char disp[32];
  snprintf(disp,20,"%f",field);
  if(clr) display.clear();
  display.print(disp,line);
}
```

```
void mydispstring(int clr,int line,char *field)
{
  char disp[32];
  snprintf(disp,20,"%s",field);
  if(clr) display.clear();  display.print(disp,line); }
void setup() {
  Serial.begin(9600);delay(1000);
WiFi.disconnect(true);
esp_wifi_sta_wpa2_ent_set_identity((uint8_t *)EAP_ID, strlen(EAP_ID));
esp_wifi_sta_wpa2_ent_set_username((uint8_t *)EAP_USERNAME,
strlen(EAP_USERNAME));
esp_wifi_sta_wpa2_ent_set_password((uint8_t *)EAP_PASSWORD,
strlen(EAP_PASSWORD));
esp_wpa2_config_t config = WPA2_CONFIG_INIT_DEFAULT();
esp_wifi_sta_wpa2_ent_enable(&config);
WiFi.begin(ssid);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");Serial.println(ip);
  Serial.println("setup ok");
  ThingSpeak.begin(client);
  display.begin();
  timeClient.update();
}

String date;
char disp[24];
void loop() {
    delay(1000);
    Serial.println("in the loop");timeClient.update();
    date=timeClient.getFormattedTime();date.toCharArray(disp,24);
    mydispstring(1,0,"Channel 2");
    temp =ThingSpeak.readFloatField(myChannelNumber,1);
    Serial.print("Last temperature:"); Serial.println(temp);
    mydispstring(0,1,"Last temperature");mydispfloat(0,2,temp);
    delay(5000);
    humi =ThingSpeak.readFloatField(myChannelNumber,2);
    Serial.print("Last humidity:");
    Serial.println(humi); mydispstring(0,3,"Last humidity");
    mydispfloat(0,4,humi); delay(5000);
    inter =ThingSpeak.readFloatField(myChannelNumber,3);
    Serial.print("Last interrupt:");
    Serial.println(humi); mydispstring(0,5,"Last interrupt");
    mydispfloat(0,6,inter);mydispstring(0,7,disp);
    delay(15000);
}
```

A faire (si vous êtes à l'école ou à l'université – accès eduroam)

1. Comme dans le sujets précédant fonctionnant avec un point d'accès maison ou smartphone créer un canal **ThingSpeak** puis récupérer les paramètres du canal : **number** et **write key**.

2. Intégrer l'utilisation des capteurs de Température/Humidité et de la Luminosité.
Envoyer ces données cycliquement (par exemple toutes les 30 seconde) vers le serveur **ThingSpeak**.

3. Récupérer les données cycliquement (par exemple toutes les 30 seconde) à partr du serveur **ThingSpeak**.
Afficher les résultats sur l'écran OLED.

Laboratoire 3 – MQTT broker et clients

3.1 Protocole MQTT

MQTT (**Message Queuing Telemetry Transport**) est un protocole de messagerie type **publication** - **abonnement** qui fonctionne sur la pile de protocoles TCP/IP. La première version du protocole a été développée par Andy Stanford-Clark d'IBM et Arlen Nipper de Cirrus Link en 1999.

Les messages **MQTT** peuvent être aussi petits que **2 octets**, alors que HTTP (cas de messages envoyés vers serveur ThingSpeak dans le Lab précédant) nécessite des en-têtes qui contiennent de nombreuses informations peu importants pour les équipements IoT. Si vous avez plusieurs appareils en attente d'une demande avec HTTP, vous devrez envoyer une action **`POST`** à chaque client.

Avec MQTT, lorsqu'un serveur (**broker**) reçoit des informations d'un client, il les distribuera automatiquement à chacun des clients intéressés (abonnés).

3.1.1 Les bases

Les termes (opérations) utilisés dans un réseau **MQTT**:

- **Broker** (**courtier**) - Le c est le serveur qui distribue les informations aux clients intéressés connectés au serveur.
- **Client** - Le périphérique qui se connecte au broker pour envoyer ou recevoir des informations.
- **Topic (sujet)**- Le nom sur lequel porte le message. Les clients publient, s'abonnent ou font les deux à un topic.
- **Publish (publier)** - Clients qui envoient des informations au broker pour les distribuer aux clients intéressés en fonction du nom du topic.
- **Subscribe (s'abonner**) - Les clients indiquent au broker le ou les topics qui les intéressent. Lorsqu'un client s'abonne à un topic, tout message publié au broker est distribué aux abonnés de ce topic. Les clients peuvent également se désabonner pour ne plus recevoir de messages du broker sur ce topic.

3.1.2 Comment fonctionne MQTT

Comme mentionné précédemment, MQTT est un protocole de messagerie de publication-abonnement.
Les clients se connecteront au réseau. Ils peuvent s'abonner ou publier sur un topic.
Lorsqu'un client publie sur un topic, les données sont envoyées au broker, qui les ensuite distribue à tous les clients abonnés à ce topic.

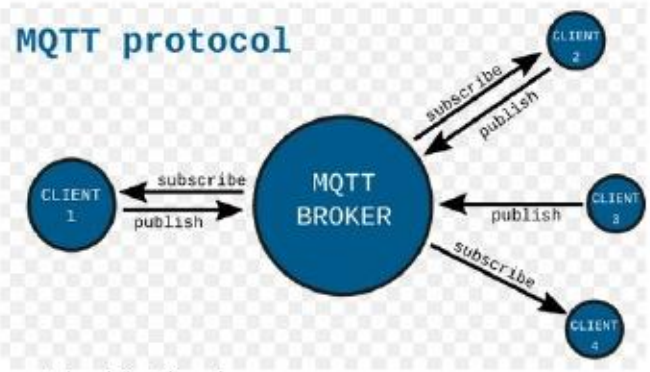


Figure 3.1 Protocole MQTT avec abonnement / publication de messages

Les topics sont organisés dans une structure de type répertoire. Un topic peut être «**LivingRoom**» ou «**LivingRoom/Light**» si vous avez plusieurs clients dans ce **topic parent**. Le client abonné écoutera les messages entrants du topic souscrit et réagira à ce qui a été publié sur ce sujet, par exemple «activé» ou «désactivé».
Les clients peuvent s'abonner à un topic et publier sur un autre également. Si le client s'abonne à «**LivingRoom/Light**», il peut également souhaiter publier sur un autre topic tel que «**LivingRoom/Light/Humidity**» afin que d'autres clients puissent surveiller l'état de cette lumière.

Maintenant que nous comprenons la théorie du fonctionnement de MQTT, construisons un exemple rapide et facile avec notre **IoT DevKit** basé sur ESP32.

3.4.3 Configuration du Broker

Il existe une grande collection de brokers MQTT disponibles qui peuvent fonctionner à partir d'un **serveur distant**, ou **localement**, à la fois **sur votre ordinateur** de bureau ainsi que **sur un SBC** comme Nano-Pi (DevKit IoT sur NanoPi - Linux).
Dans l'exemple utilisé dans ce Lab, nous allons utiliser un Nano Pi connecté à notre réseau local exécutant un broker gratuit et open-source appelé **`mosquitto`**.

> Pour les utilisateurs de Windows voici le lien d'installation de **Mosquitto v 1.5.8**
> **`http://www.steves-internet-guide.com/install-mosquitto-broker/`**

Pour installer **`mosquitto`** (Ubuntu):

```
sudo apt-get install mosquitto -y
```

Une fois installé, nous voudrons nous assurer que notre broker fonctionne correctement en créant un client de test pour écouter un topic. Nous le ferons en installant les clients **`mosquitto`**:

```
sudo apt-get install mosquitto mosquitto-clients -y
```

Une fois les clients installés, nous nous abonnerons au topic **`test_topic`** en entrant:

```
mosquitto_sub -t "test_topic"
```

Nous disons à **`mosquitto`** que nous aimerions nous abonner à un topic en entrant **`mosquitto_sub`**, et que nous aimerions nous abonner à un topic désigné par **`-t`** avec le nom **`test_topic`**.
Désormais, chaque fois que nous publions sur **`test_topic`**, le message envoyé apparaîtra dans cette fenêtre.

Parce que notre terminal écoute les messages de notre broker, nous devrons ouvrir une deuxième fenêtre de terminal pour publier les messages. Une fois ouvert, nous publierons sur **`test_topic`** avec la commande suivante:

```
mosquitto_pub -t "test_topic" -m "HELLO WORLD!"
```

Tout comme avant, nous utilisons **`-t`** pour désigner le topic, mais cette fois, nous ajoutons un message à publier sur le topic en utilisant **`mosquitto_pub`** et en utilisant **`-m`** pour désigner le message que nous aimerions publier.

Une fois que nous avons appuyé sur Entrée, nous devrions voir notre message apparaître sur la fenêtre du terminal de l'abonné, comme indiqué ci-dessous. Vous pouvez remplacer ce texte par n'importe quelle chaîne que vous souhaitez après **`-m`** pour envoyer votre message à tous les clients abonnés à **`test_topic`**.

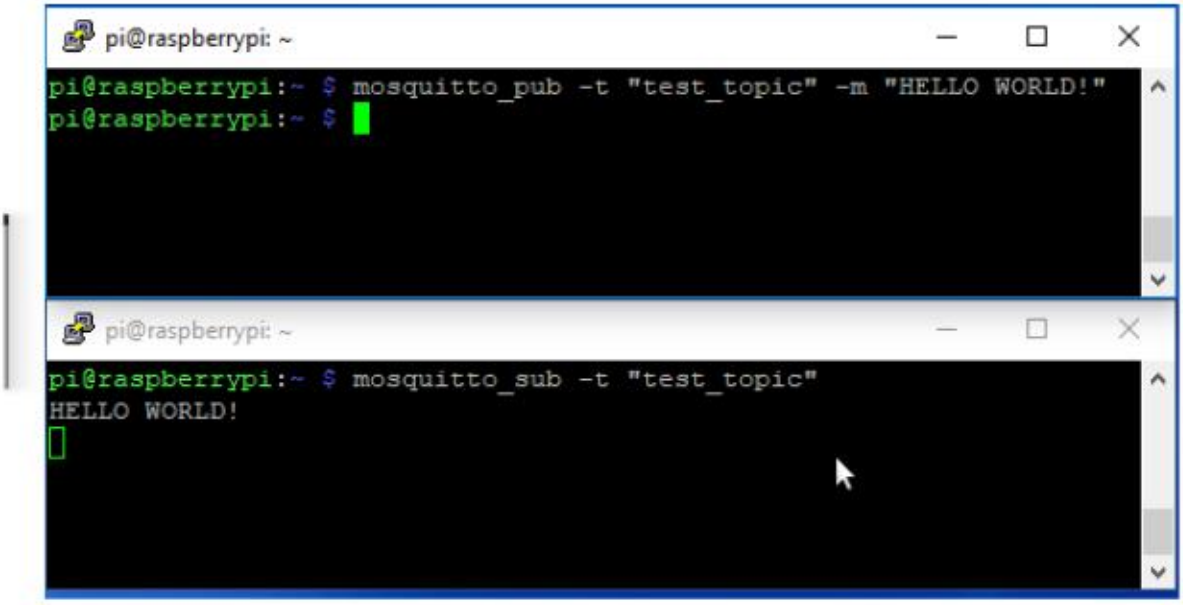


Figure 3.2 Commandes **`mosquitto`** avec abonnement / publication de messages

3.2 Serveur externe de MQTT - `test.mosquitto.org`

Instead of your local **`mosquitto`** server you can as well use the external **`mosquitto`** server available at **`test.mosquitto.org`** .
Au lieu de votre serveur **mosquitto** local, vous pouvez également utiliser le serveur mosquitto externe disponible sur **`test.mosquitto.org`**.
Ce serveur Mosquitto écoute sur les ports suivants:

- **1883**: MQTT, non crypté
- **8883**: MQTT, crypté
- **8884**: MQTT, crypté, certificat client requis
- **8080**: MQTT sur **WebSockets**, non crypté
- **8081**: MQTT sur **WebSockets**, crypté

Les ports cryptés prennent en charge **`TLS v1.3, v1.2`** ou **`v1.1`** avec des certificats **`x509`** et nécessitent le support client pour se connecter. Dans tous les cas, vous devez utiliser le fichier d'autorité de certification (**`mosquitto.org.crt`** (format **PEM**) ou **`mosquitto.org.der`** (format **DER**)) pour vérifier la connexion au serveur.
Le port **8884** oblige les clients à fournir un certificat pour authentifier leur connexion. Il est désormais possible de générer votre propre certificat.
Vous êtes libre d'utiliser **`test.mosquitto.org`** pour n'importe quelle application, mais veuillez ne pas en abuser ni vous y fier pour quoi que ce soit d'important. Vous devez également créer votre client pour faire face au redémarrage du broker.
Si vous avez installé les clients **`mosquitto`**, essayez:

```
mosquitto_sub -h test.mosquitto.org -t "smtr-lab3" -v
```

3.2.1 Configuration des clients

Maintenant que nous savons que notre broker est opérationnel, il est temps d'ajouter nos clients. Nous allons créer un client qui s'abonnera à **`room/light`** et répondra au message en allumant ou éteignant la LED (broche 25 sur notre carte ESP32). Dans l'exemple suivant, nous utilisons la bibliothèque **`PubSubClient.h`** , elle doit être installée précédemment dans votre IDE Arduino.

3.2.1.1 Le client abonné en attente de messages MQTT

L'exemple suivant permet d'envoyer (publier) les messages et les recevoir sur la fonction **`callback()`**.
Nous utilisons ici notre broker local installé sur notre PC ou SBC.
Attention
Si vous copiez cet exemple directement il faut que vous **personnalisiez le ID – le nom de votre `device`**.

```
#include <WiFi.h>
#include <PubSubClient.h>
#define ssid "PhoneAP"
#define pass "smartcomputerlab"
const byte LIGHT_PIN = 25;            // Pin to control the light with
const char *ID = "My_Device...";     // Name of our device, must be unique
const char *TOPIC = "smtr-lab3";     // Topic to subcribe to
const char *STATE_TOPIC = "smtr-lab3/state";
                          // Topic to publish the light state to
IPAddress broker(192,168,43,159); // IP address of your MQTT broker
WiFiClient wclient;
PubSubClient client(wclient); // Setup MQTT client
// Handle incomming messages from the broker
void callback(char* topic, byte* payload, unsigned int length) {
  String response;
  for (int i = 0; i < length; i++) {
    response += (char)payload[i];
  }
  Serial.print("Message arrived [");
  Serial.print(topic); Serial.print("] ");
  Serial.println(response);
```

```
  if(response == "on")  // Turn the light on
  {
    digitalWrite(LIGHT_PIN, HIGH);
  }
  else if(response == "off")  // Turn the light off
  {
    digitalWrite(LIGHT_PIN, LOW);
  }
}
// Connect to WiFi network
void setup_wifi() {
  Serial.print("\nConnecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass); // Connect to network
  while (WiFi.status() != WL_CONNECTED) { // Wait for connection
    delay(500); Serial.print(".");
  }
  Serial.println();
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}
// Reconnect to client
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if(client.connect(ID)) {
      client.subscribe(TOPIC);
      Serial.println("connected");
      Serial.print("Subcribed to: ");
      Serial.println(TOPIC);
      Serial.println('\n');
    } else {
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(9600); // Start serial communication at 115200 baud
  pinMode(LIGHT_PIN, OUTPUT); // Configure LIGHT_PIN as an output
  delay(100);
  setup_wifi(); // Connect to network
  client.setServer(broker, 1883);
  client.setCallback(callback);// Initialize the callback routine
}

void loop() {
  if (!client.connected())  // Reconnect if connection is lost
  {
    reconnect();
  }
  client.loop();
}
```

Une fois que le deuxième ESP32 se connecte au réseau, il s'abonnera automatiquement au topic **room/light**, la LED intégrée connectée à la broche **GPIO 25** devrait répondre et s'allumer et s'éteindre.

L'affichage de publication sur **mosquitto**:

```
pi@nano-pi:~$ mosquitto_pub m "on" smtr-lab3
pi@nano-pi:~$ mosquitto_pub -m "off" -t smtr-lab3
```

L'affichage sur le terminal Arduino IDE :

```
Connecting to PhoneAP
..
WiFi connected
IP address: 192.168.43.185
Attempting MQTT connection...connected
Subcribed to: room/light

Message arrived [room/light] on
Message arrived [room/light] off
```

3.2.1.2 Le client publiant des messages MQTT

```
#include <WiFi.h>
#include <PubSubClient.h>
#define ssid "PhoneAP"
#define pass "smartcomputerlab"
const char* mqttServer = "5.196.95.208";  // test.mosquitto.org

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(9600);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to the WiFi network");
  client.setServer(mqttServer,1883);
  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("connected");
    } else {
      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);
    }
  }
  client.publish("esp/test", "Hello from ESP32");
  Serial.println("published");
}

void loop() {
  client.publish("esp/test", "Hello from ESP32");
  Serial.println("published");
  delay(5000);
}
```

L'affichage sur le terminal Arduino IDE :

```
.........................Connected to the WiFi network
Connecting to MQTT...
connected
published
published
published
```

L'affichage sur **mosquitto** subscriber:

```
bako@bako:~$ mosquitto_sub -h test.mosquitto.org -t "esp/test"
Hello from ESP32
Hello from ESP32
Hello from ESP32
Hello from ESP32
```

3.2.2 Utilisation de la bibliothèque MQTT

Les exemples suivants montrent comment exploiter la bibliothèque Arduino **MQTT.h** qui semble plus simple à utiliser que la bibliothèque **PubSubClient.h**.

3.2.2.1 Publication et réception de messages de test

L'exemple suivant publie et reçoit simples messages (« **coucou** »)MQTT sur le topic **/hello** . Nous utilisons le broker gratuit :

```
#include <WiFi.h>
#include <MQTT.h>
const char ssid[] = "PhoneAP";
const char pass[] = "smartcomputerlab";
WiFiClient net;
MQTTClient client;
unsigned long lastMillis = 0;

void connect() {
  Serial.print("checking wifi...");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print("."); delay(1000);
  }
  Serial.print("\nconnecting...");
  while (!client.connect("IoT.DevKit")) {
    Serial.print("."); delay(1000);
  }
  Serial.println("\nconnected!");
  client.subscribe("/hello");
}

void messageReceived(String &topic, String &payload) {
  Serial.println("incoming: " + topic + " - " + payload);
}

void setup() {
  Serial.begin(9600);
  WiFi.begin(ssid,pass);
  client.begin("5.196.95.208", net);
  client.onMessage(messageReceived);
  connect();
}
```

```
void loop() {
  client.loop();
  delay(10);  // <- fixes some issues with WiFi stability
  if (!client.connected()) { connect(); }
  if (millis() - lastMillis > 1000) {   // publish a message every second.
    lastMillis = millis();
    client.publish("/hello", "coucou");
  }
}
```

3.2.3 Publication et réception des valeurs des capteurs

Le deuxième exemple avec la bibliothèque **`MQTT.h`** inclut l'utilisation d'un simple capteur **DHT22**. La carte envoie (et reçoit) les messages MQTT et affiche le contenu sur l'écran **OLED**.
Nous utilisons, comme dans l'exemple précédant le broker :**`test.mosquitto.org - "5.196.95.208"`**

```
#include <WiFi.h>
#include <MQTT.h>
#include "DHT.h"
#define DHTTYPE DHT22
DHT dht(17, DHTTYPE);
#include <U8x8lib.h>
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15,4,16); // data, clock,
int dhtPin = 17;
const char ssid[] = "PhoneAP";
const char pass[] = "smartcomputerlab";
WiFiClient net;
MQTTClient client;
unsigned long lastMillis = 0;

void connect() {
  Serial.print("checking wifi...");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print("."); delay(1000);
  }
  Serial.print("\nconnecting...");
  u8x8.drawString(0,5,"connecting to");
  u8x8.drawString(0,6,"5.196.95.208");
  while (!client.connect("IoT.DevKit")) {
    Serial.print("."); delay(1000);
  }
  Serial.println("\nconnected !");
  u8x8.drawString(0,7,"connected !");
  client.subscribe("/dht22");
}

void messageReceived(String &topic, String &payload) {
  char dbuff[32], btemp[16]; int len=32;
  Serial.println("incoming: " + topic + " - " + payload);
  u8x8.clear();u8x8.drawString(0,0,"5.196.95.208");
  u8x8.drawString(0,1,"IoT.DevKit");u8x8.drawString(0,2,"MQTT topic");
  topic.toCharArray(dbuff,len);u8x8.drawString(0,3,dbuff);
  u8x8.drawString(0,4,"Payload");payload.toCharArray(dbuff,len);
  memset(btemp,0x00,16);
  memcpy(btemp,dbuff,10);
  u8x8.drawString(0,5,btemp);u8x8.drawString(0,6,dbuff+12);
}
byte mac[6];
char dbuff[32];
```

```
void setup() {
  Serial.begin(9600);
  pinMode(dhtPin,INPUT);
  dht.begin();delay(100);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
  };
  IPAddress ip = WiFi.localIP();
  Serial.println(ip);
  WiFi.macAddress(mac);
  Serial.print("MAC: ");Serial.print(mac[5],HEX);Serial.print(":");
  Serial.print(mac[4],HEX);Serial.print(":");
  Serial.print(mac[3],HEX);Serial.print(":");
  Serial.print(mac[2],HEX);Serial.print(":");Serial.print(mac[1],HEX);
  Serial.print(":"); Serial.println(mac[0],HEX);
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.clear();
  u8x8.drawString(0,0,"IP address");
  sprintf(dbuff,"%d.%d.%d.%d",ip[0],ip[1],ip[2],ip[3]);
  u8x8.drawString(0,1,dbuff);
  u8x8.drawString(0,2,"MAC address");
  sprintf(dbuff,"%2.2x%2.2x%2.2x%2.2x%2.2x%2.2x",
       mac[0],mac[1],mac[2],mac[3],mac[4],mac[5]);
  u8x8.drawString(0,3,dbuff);
  client.begin("5.196.95.208", net);
  client.onMessage(messageReceived);
  connect();
}

float temperature,humidity;

void loop() {
  char cbuf[32];
  client.loop();
  delay(10);  // <- fixes some issues with WiFi stability
  humidity = dht.readHumidity();          // Read humidity (percent)
  temperature = dht.readTemperature();     // Read temperature as Celsius
  if (!client.connected()) {
    connect(); }
  sprintf(cbuf,"temp:%2.2f, humi:%2.2f",temperature,humidity);
  if (millis() - lastMillis > 20000) {
    lastMillis = millis();
    client.publish("/dht22", cbuf);
  }
}
```

Ce qui suit est le résultat affiché sur le terminal Arduino IDE:

```
192.168.43.185
MAC: D8:3F:A9:BF:71:3C
checking wifi...
connecting...
connected !
incoming: /dht22 - temp:26.10, humi:55.40
incoming: /dht22 - temp:26.10, humi:54.30
incoming: /dht22 - temp:26.10, humi:54.10
incoming: /dht22 - temp:26.10, humi:54.00
```

3.3 Utilisation d'une connexion sécurisée avec `SSL` et `WiFiClientSecure`

Les messages MQTT envoyés au numéro de port **1883** ne sont pas cryptés. Afin d'utiliser une connexion sécurisée, nous avons besoin du protocole **SLL** sur le port numéro **8883**.
Ce qui suit est un exemple simple d'envoi de messages texte sécurisés au broker MQTT-`broker.shift.io`.

```
#include <WiFiClientSecure.h>
#include <MQTT.h>
const char ssid[] = "PhoneAP";
const char pass[] = "smartcomputerlab";
WiFiClientSecure net;
MQTTClient client;
unsigned long lastMillis1 = 0, lastMillis2=0;

void connect() {
  Serial.print("checking wifi...");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");  delay(1000);
  }
  Serial.print("\nWiFi connected...");
  Serial.print("\nconnecting to MQTT server...");
  while (!client.connect("esp32sec", "try", "try")) {
    Serial.print("."); delay(1000);
  }
  Serial.println("\nconnected!");
  client.subscribe("/esp32hello");
}

void messageReceived(String &topic, String &payload)
{
  Serial.println(topic + " - " + payload);
}

void setup() {
  Serial.begin(9600);
  WiFi.begin(ssid, pass);
  // MQTT brokers usually use port 8883 for secure connections.
  client.begin("broker.shiftr.io", 8883, net);
  client.onMessage(messageReceived);
  // Name:  broker.shiftr.io - Address: 34.76.6.166
  connect();
}

void loop() {
  client.loop();
  delay(10);  // <- fixes some issues with WiFi stability
  if (!client.connected()) { connect();}
  // publish a message roughly every second.
  if (millis() - lastMillis1 > 6000) {
    lastMillis1 = millis();
    client.publish("/esp32hello", "coucou");
  }
    if (millis() - lastMillis2 > 7000) {
    lastMillis2 = millis();
    client.publish("/esp32hello", "bonjour");
  }
}
```

L'affichage sur le terminal Arduino IDE:

```
checking wifi.....................................checking wifi....
WiFi connected...
connecting to MQTT server....
connected!
/esp32hello - coucou
/esp32hello - bonjour
/esp32hello - coucou
/esp32hello - bonjour
/esp32hello - coucou
```

3.4 Utilisation de `WiFiMQTTManager`

`WiFiMQTTManager` est une bibliothèque ESP qui étend les bibliothèques client **`WiFiManager`** et **`PubSub`** et ajoute la possibilité d'enregistrer les paramètres de connexion **SSID**, mot de **passe WiFi** et **MQTT** afin qu'ils n'aient pas à être codés en dur dans vos sketch.
Le code de base lance le **`WiFiManager`** qui bascule éventuellement en mode point d'accès pour afficher une page web (**`192.168.4.1`**) qui permet de récupérer les identificateurs WiFi et ceux du serveur **MQTT**.
Dans le fichier associé : secrets.h il faut mettre votre mot de passe (default est **`CHANGEME`**).

3.4.1 Le code

```
#include "secrets.h"
#include <WiFiMQTTManager.h>
// Button that will put device into Access Point mode to allow for re-entering
WiFi and MQTT settings
#define RESET_BUTTON 0
WiFiMQTTManager wmm(RESET_BUTTON, AP_PASSWORD); // defined in secrets.h file

void setup() {
  Serial.begin(9600);
  Serial.println("WiFiMQTTManager Basic Example");
  // set debug to true to get verbose logging
  // wm.wm.setDebugOutput(true);
  // most likely need to format FS but only on first use
  // wmm.formatFS = true;
  // optional - define the function that will subscribe to topics if needed
  wmm.subscribeTo = subscribeTo;
  // required - allow WiFiMQTTManager to do it's setup
  wmm.setup(__SKETCH_NAME__);
  // optional - define a callback to handle incoming messages from MQTT
  wmm.client->setCallback(subscriptionCallback);
}
void loop() {
  // required - allow WiFiMQTTManager to check for new MQTT messages,
  // check for reset button push, and reconnect to MQTT if necessary
  wmm.loop();
  // publishing to MQTT a sensor reading once every 10 seconds
  long now = millis();
  if (now - wmm.lastMsg > 10000) {
    wmm.lastMsg = now;
    float temperature = 23.54; // read sensor here
    Serial.print("Temperature: ");
    Serial.println(temperature);
    char topic[100];
    //snprintf(topic, sizeof(topic), "%s%s%s", "sensor/", wmm.deviceId,
"/temperature");
    snprintf(topic, sizeof(topic), "%s", "smtr-lab3");
```

```
    wmm.client->publish(topic, String(temperature).c_str(), true);
  }
}

void subscribeTo() {
  Serial.println("subscribing to some topics...");
  char topic[100];
  wmm.client->subscribe("smtr-lab3"); // subscribe to some topic(s)
}

void subscriptionCallback(char* topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);Serial.print(". Message: ");
  String messageTemp;
  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }
  // do something
}
```

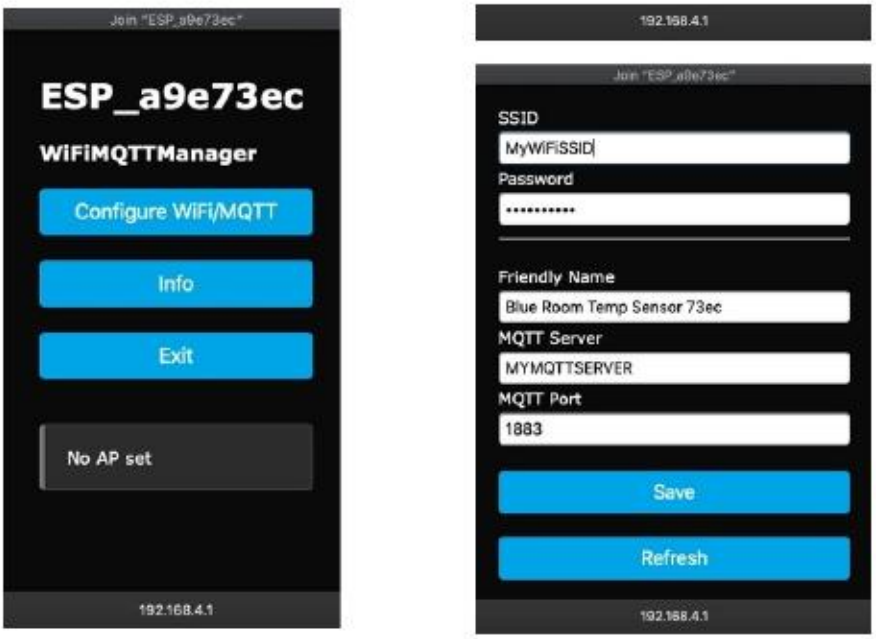


Figure 3.3 Pages WEB de **`WiFiMQTTManager`** pour la configuration du mode de passe et de l'adresse IP du serveur **MQTT**.

3.5 Envoi de messages MQTT au serveur `ThingSpeak`

Le serveur IoT **`ThingSpeak`** est également capable de recevoir les messages MQTT et d'afficher leur contenus sur les chronogrammes. L'exemple suivant montre l'utilisation de la bibliothèque **`PubSubClient.h`**

3.5.1 Le code

Dans la partie initiale du code, vous devez définir et initialiser les variables.
Assurez-vous de modifier les « **credentials** » WiFi , l'ID de canal **`ThingSpeak`** et les clés **API**.
Trouvez votre identifiant de chaîne en haut de la page principale de votre chaîne.

Chaque **champ** (*field*) du canal est considéré comme **topic**.
Pour écrire (publier) dans un topic, vous devez fournir une ligne comme:

```
String temperature_topic =
"channels/1226765/publish/fields/field1/U4TY3T09ZSMVV5SR"
```

où:

`1226765` est le numéro de canal et
`U4TY3T09ZSMVV5SR` est la clé API d'écriture

```
#include <WiFi.h>
#include <PubSubClient.h> // MQTT

const char* ssid = "PhoneAP";
const char* password = "smartcomputerlab";
const char* mqtt_server = "mqtt.thingspeak.com";
String temperature_topic =
"channels/1226765/publish/fields/field1/U4TY3T09ZSMVV5SR";
String humidity_topic =
"channels/1226765/publish/fields/field2/U4TY3T09ZSMVV5SR";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
byte temperature=23;
byte humidity=66;

void setup() {
  Serial.begin(9600);
  WiFi.begin(ssid, password);
  int wifiTryConnectCounter = 0;
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);Serial.print(".");
  }
  Serial.println("WiFi connected");
  //MQTT
  client.setServer(mqtt_server, 1883);
  // Create a random client ID
  String clientId = "ESPClient-";
  clientId += String(random(0xffff), HEX);
  // Attempt to connect
  if (client.connect(clientId.c_str())) {
   Serial.println("MQTT server connected");
  } else {
    ESP.deepSleep(sleepTimeS * 1000000);
  }
```

```
  //Sensor
  temperature=23; humidity=66;
  //Send
  sprintf(msg,"%ld", (int)temperature);
  //client.publish(temperature_topic.c_str(), msg);
  sprintf(msg,"%ld", (int)humidity);
  client.publish(humidity_topic.c_str(), msg);
  //Sleep and repeat
  client.loop(); // MQTT work
  ESP.deepSleep(20 * 1000000);  // sleeping 20 seconds
}

void loop() {
  // Will not run
}
```

3.5.2 Mode `deepSleep()` et les données dans SRAM et EPROM

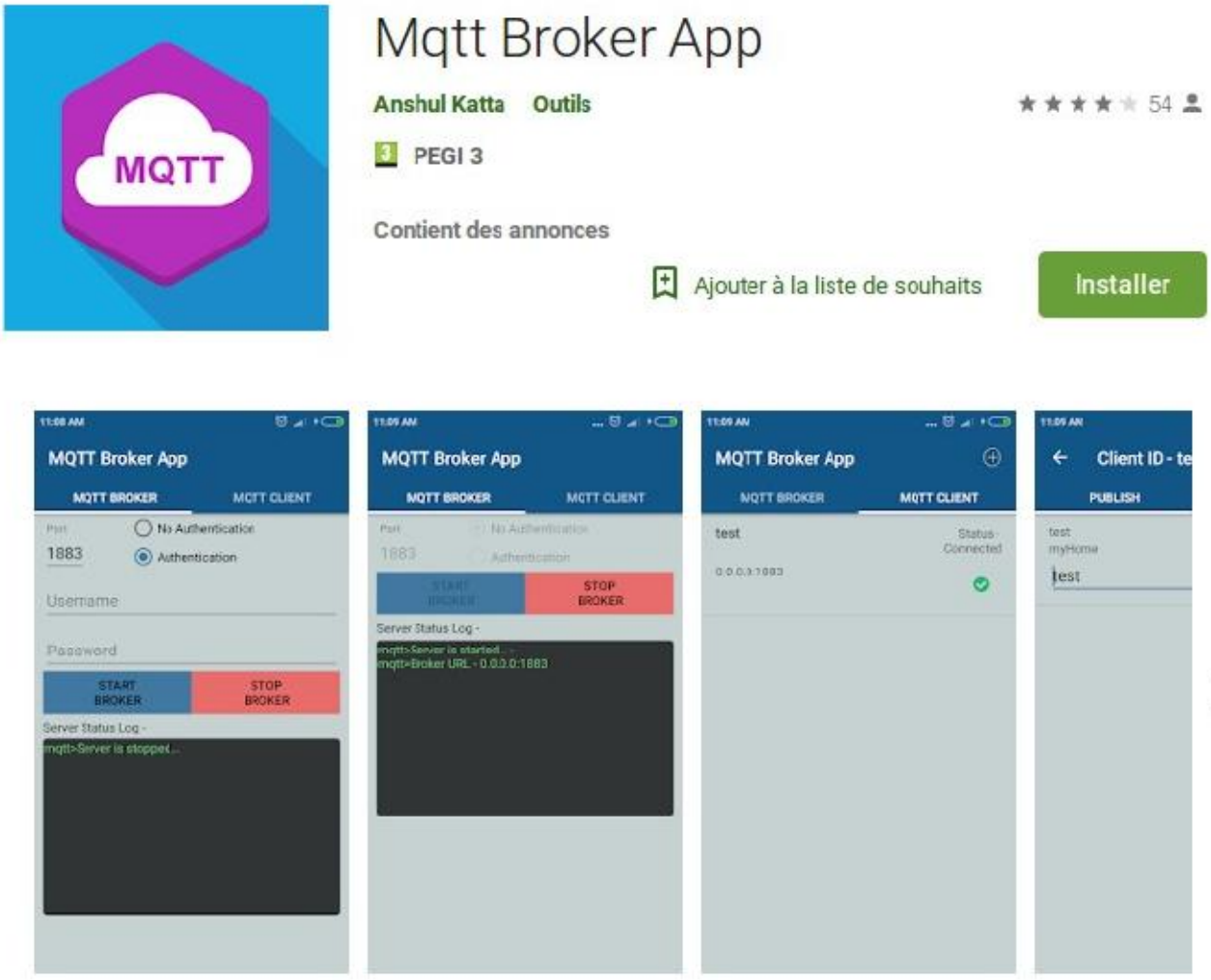
Les programme ci-dessus utilise la fonction **`ESP.deepSleep(20*1000000)`**.
Cette fonction arrête l'exécution de l'application en activant un signal de **time-out** après 20 secondes.
Pendant le **`deepSleep()`** le programme perd toutes les valeurs de données dans la mémoire **SRAM**.
La boucle **`loop()`** (tâche de fond) n'est jamais exécutée !
Pour garder les données du programme il faut les enregistrer dans la mémoire **EEPROM** juste avant le lancement de **`deepSleep()`** et les restaurer au début de **`setup()`**.

Ci dessous vous trouvez les éléments du code nécessaires pour réaliser la **sauvegarde** et la **restauration** des donnés par le biais de la **EEPROM**.

```
#include "EEPROM.h"
#define EEPROM_SIZE 128
uint8_t  mbuff[128];
// in setup()
  if(!EEPROM.begin(EEPROM_SIZE))
  {
    Serial.println("failed to initialise EEPROM");
  }
// to read
  for (i = 0; i < EEPROM_SIZE; i++)
  {
    mbuff[i]=EEPROM.read(i);
  }
// write and commit
for (i = 0; i < EEPROM_SIZE; i++)
      {
      EEPROM.write(i,rbuff[i]);
      }
    EEPROM.commit();
```

3.6 Android MQTT et Application Broker

L'application Android **Mqtt Broker** nous permet de fonctionner en mode broker et client.



Vous pouvez le tester avec les exemples présentés ci-dessus.

A faire:

1. Installez le broker et le client **`mosquitto`** sur votre ordinateur et testez le.

2. Utilisez votre DevKit pour implémenter l'exemple **3.2.1.1** fonctionnant avec votre broker (il faut que votre ordinateur et votre carte soient connectés au même réseau local).

3. Etudiez et testez sur votre DevKit l'exemple **3.2.1.2** avec un broker externe - **`test.mosquitto.org`**

4. Testez l'exemple **3.2.2.1** (bibliothèque **`MQTT.h`**)

5. Sujet pour deux DevKits d'un binôme communiquant par le biais du serveur **`test.mosquitto.org`**
   a. Testez 'exemple **3.2.3** , puis modifiez le capteurs en utilisant les capteurs et SHT21/BH1750 et trois topics TEMP/HUMI/LUMI
   b. Décomposez le programme de **3.2.3** en deux parties (**une par le membre du binôme**) **une publiant** seulement (exemple) :

```
client.publish("/TEMP", Tcbuf);
client.publish("/HUMI", Hcbuf);
client.publish("/LUMI", Lcbuf);
..
```

   et l'autre avec la fonction :

```
void messageReceived(String &topic, String &payload){ … }
```

   qui affiche le message reçu sur son écran **OLED**.

6. Mettez en ouvre le même exemple (**3.2.3**) avec un mode **MQTT sécurisé** avec **`WiFiClientSecure.h`** sur le serveur **`broker.shift.io`**

Sujets supplémentaires :

1. Testez le sujet 3.4 (le code 3.4.1)
2. Mettez en œuvre le mode deepSleep() avec la mémoire EEPROM pour publier dans les exécutions alternativement :

```
client.publish(temperature_topic.c_str(), msg);
```

et

```
client.publish(humidity_topic.c_str(), msg);
```

3. Essayez l'application Android **Mqtt Broker** en mode client et broker pour communiquer avec votre IoT DevKit.

Laboratoire 4 - Bluetooth Classic (BT) et Bluetooth Low Energy (BLE)

L'ESP32 est livré avec Wi-Fi, **Bluetooth Low Energy** et **Bluetooth Classic**. Dans ce laboratoire, vous apprendrez comment utiliser ESP32 **Bluetooth Classic** et **Bluetooth Low Energy** (BLE) avec Arduino IDE pour échanger des données entre un ESP32 et un **smartphone Android**.

Pour commencer nous contrôlerons un ESP32 et enverrons des relevés de capteur à un smartphone Android à l'aide de **Bluetooth Classic**.

4.1 Bluetooth Classique

Bluetooth Classique est une norme et protocole de communication permettant l'échange bidirectionnel de données à courte distance en utilisant des ondes radio sur la bande (ISM) de fréquence de 2.4 GHz. Son but est de mettre en œuvre les connexions entre les appareils électroniques à proximité en supprimant des liaisons filaires.
Le nom « *Bluetooth* » est directement inspiré du surnom anglicisé du roi viking danois Harald *à la dent bleue* (en danois *Harald Blåtand*, en anglais *Harald Bluetooth*), connu pour avoir réussi à unifier les tribus danoises au sein d'un même royaume, introduisant du même coup le christianisme. Ce nom a été proposé en 1996 par Jim Kardach d'Intel, un ingénieur travaillant alors sur le développement d'un système qui allait permettre aux téléphones cellulaires de communiquer avec des ordinateurs. Au temps où Kardach a fait cette proposition, un homologue d'Ericsson lui avait parlé de ce souverain après avoir lu le roman historique *Orm le Rouge* de Frans Gunnar Bengtsson, qui se déroule sous son règne1. L'implication est que de la même façon que le roi Harald a unifié son pays et rassemblé le Danemark et la Norvège, Bluetooth relie les télécommunications et les ordinateurs et **unifie** les appareils entre eux.

Couche physique

Le système Bluetooth opère dans les bandes de fréquences ISM (*Industrial, Scientific and Medical*) 2.4GHz. Un *transceiver* à sauts de fréquences est utilisé pour limiter les interférences et l'atténuation.
Pour le **Bluetooth classique** (hors version BLE), deux modulations sont définies : une obligatoire utilisant une modulation de fréquence binaire pour minimiser la complexité de l'émetteur ; une modulation optionnelle (mode EDR) utilise une modulation de phase (PSK à 4 et 8 symboles). La rapidité de modulation est de 1 Mbaud pour toutes les modulations. La transmission duplex utilise une division temporelle.

Les **79 canaux RF du Bluetooth classique** (**40 en mode BLE**) sont numérotés de 0 à 78 et séparés de 1 en commençant par **2402 MHz** . Le codage de l'information se fait par sauts de fréquences et la période est de 625 microsecondes, ce qui permet 1 600 sauts par seconde.
La **bande de base** (**baseband**) est gérée au niveau matériel. C'est au niveau de la bande de base que sont définies les adresses matérielles des périphériques (équivalentes à l'adresse MAC). Cette adresse est nommée `BD_ADDR` (**Bluetooth Device Address**) et est codée sur **48 bits**.

Couche liaison

Le **contrôleur de liaisons encode** et **décode** les paquets Bluetooth selon la charge utile et les paramètres liés au canal physique, transport logique et liaisons logiques.
Le gestionnaire de liaisons crée, gère et détruit les canaux **L2CAP** pour le transport des protocoles de services et les flux de données applicatives. Il utilise le protocole L2CAP pour interagir avec son homologue sur les équipements distants.

Cette couche gère les liens entre les périphériques « maîtres » et « esclaves » ainsi que les types de liaisons (synchrones ou asynchrones).

C'est le gestionnaire de liaisons qui implémente les mécanismes de sécurité comme :

- l'authentification ;
- l'appairage (l'association) ;
- la création et la modification des clés ;
- et le chiffrement.

À l'heure actuelle, l'utilisation de Bluetooth Classic est beaucoup plus simple que Bluetooth Low Energy. Si vous avez déjà programmé un Arduino avec un module Bluetooth comme le **HC-06**, c'est très similaire. Il utilise le protocole et les fonctions **série standard**.

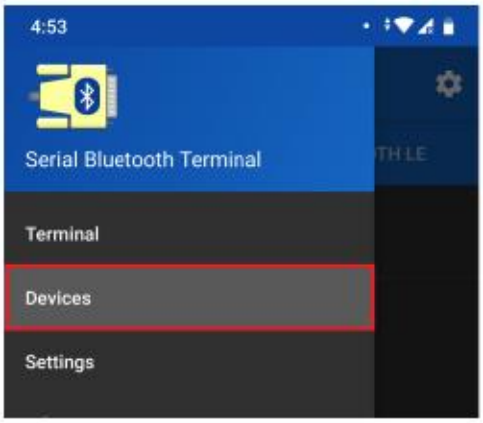
4.2 Bluetooth Classique avec ESP32

Dans cette partie du laboratoire, nous allons commencer par utiliser un exemple fourni avec l'IDE Arduino. Ensuite, nous allons construire un projet simple pour échanger des données entre l'ESP32 et votre smartphone Android.
Cherchez le premier exemple avec : **`File > Examples > BluetoothSerial > SerialtoSerialBT.`**

4.2.1. Bluetooth simple lecture écriture en série

Le code suivant démarre la communication Bluetooth et échange les données via les fonctions **`SerialBT.read ()`** et **`SerialBT.write ()`**.

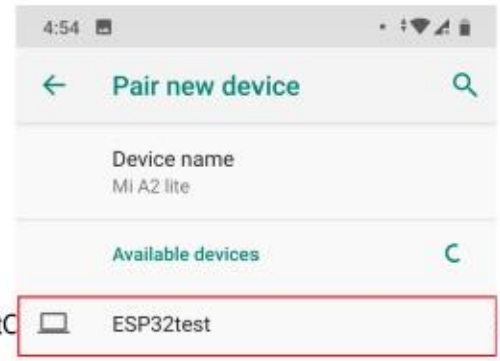
```
BluetoothSerial SerialBT;

void setup() {
  Serial.begin(9600);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.println("The device started, now you can pair it with bluetooth!");
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```



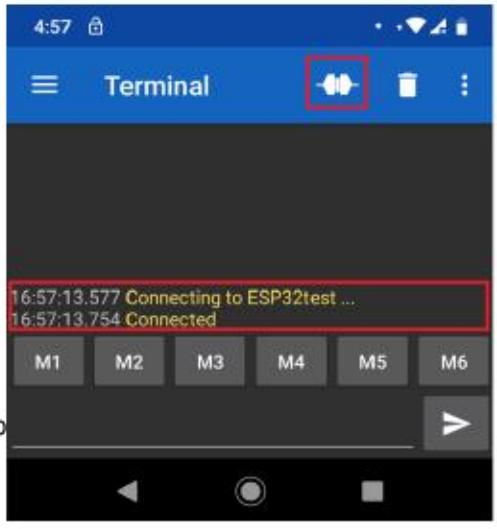
Accédez à votre smartphone et ouvrez l'application **Serial Bluetooth Terminal**. Assurez-vous d'avoir activé le Bluetooth de votre smartphone. Pour vous connecter à l'ESP32 pour la première fois, vous devez coupler un nouvel appareil.

Accédez à **Devices**.

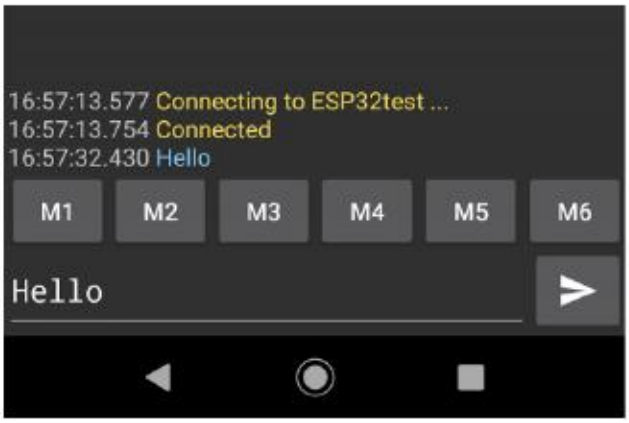
Cliquez sur l'icône des paramètres et sélectionnez **Pair new device**. Vous devriez obtenir une liste des appareils Bluetooth disponibles, y compris **`l'ESP32test`**.
Associez-le à l'**`ESP32test`**.



SmartC DevKit Laborato



Revenez ensuite au terminal Bluetooth série. Cliquez sur l'icône en haut pour vous connecter à l'ESP32. Vous devriez recevoir un message **Connected**.
Après cela, tapez quelque chose dans l'application **Serial Bluetooth Terminal**. Par exemple, «**Hello**».



4.2.2 Échange de données à l'aide de Bluetooth Serial et de votre smartphone

Maintenant que vous savez comment échanger des données à l'aide de **Bluetooth Serial**, vous pouvez modifier l'esquisse précédente pour en faire quelque chose d'utile. Par exemple, contrôlez les sorties ESP32 lorsque vous recevez un certain message ou envoyez des données à votre smartphone comme des lectures de capteur. Le projet que nous allons créer envoie des relevés de température toutes les 10 secondes à votre smartphone. Nous utiliserons le capteur de température/humidité **SHT21**.
Grâce à l'application Android, nous enverrons des messages pour contrôler une sortie ESP32. Lorsque l'ESP32 reçoit le message **`led_on`**, nous allumons le GPIO, lorsqu'il reçoit le message **`led_off`**, nous éteignons le LED GPIO (pin **25**).

4.2.2.1 Le code

```
#include "BluetoothSerial.h"
#include <Wire.h>
#include <SHT21.h>  // include SHT21 library
SHT21 sht;
#include <U8x8lib.h>
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16);
BluetoothSerial SerialBT;  // Bluetooth Serial object
const int ledPin =  25;   // GPIO where LED is connected to

String message = "";
char incomingChar;
String temperatureString = "";
// Timer: auxiliar variables
unsigned long previousMillis = 0;// Stores last time temperature was published
const long interval = 10000;   // interval at which to publish sensor readings

void setup() {
  pinMode(ledPin, OUTPUT);
  Wire.begin(21,22);
  Serial.begin(9600);
  SerialBT.begin("ESP32");
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.clear();
  u8x8.drawString(0,0,"Start BT:ESP32");
  Serial.println("The device started, now you can pair it with bluetooth!");
}
```

```
void loop() {
  char cbuff[40]; int i=0;
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval){
    previousMillis = currentMillis;
    temperatureString = String(sht.getTemperature()) + "C";
    SerialBT.println(temperatureString);
  }
  // Read received messages (LED control command)
  i=0;
  if (SerialBT.available()){
    char incomingChar = SerialBT.read();
    if (incomingChar != '\n'){
      message += String(incomingChar);
    }
    else{
      message = "";
    }
    Serial.write(incomingChar); message.toCharArray(cbuff,40);
  }
  // Check received message and control output accordingly
  u8x8.drawString(0, 3,"Received");
  u8x8.drawString(0, 5, cbuff);
  if (message =="led_on"){
    digitalWrite(ledPin, HIGH); Serial.println("LED-ON");
  }
  else if (message =="led_off"){
    digitalWrite(ledPin, LOW);Serial.println("LED-OFF");
  }
  delay(20);
}
```

4.2.3 A faire:

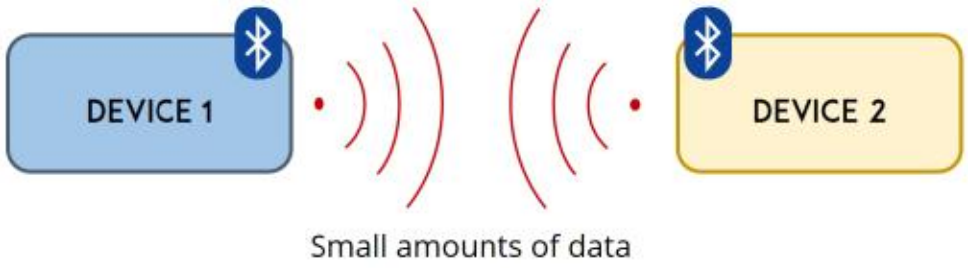
Installez l'application et testez le programme ci-dessus.

4.3 Bluetooth Low Energy (BLE) avec ESP32

Cette partie du laboratoire commence par une introduction rapide au **BLE** avec l'ESP32. Tout d'abord, nous explorerons ce qu'est le **BLE** et à quoi il peut être utilisé, puis nous examinerons quelques exemples avec l'ESP32 utilisant Arduino IDE. Pour une introduction simple, nous allons créer un serveur ESP32 BLE et un scanner ESP32 BLE pour trouver ce serveur.

4.3.1 Qu'est-ce que Bluetooth Low Energy?

Bluetooth Low Energy, **BLE** pour faire court, est une variante d'économie d'énergie de **Bluetooth**. L'application principale de **BLE** est la transmission à courte distance de petites quantités de données (faible bande passante). Contrairement à **Bluetooth** qui est **toujours activé**, **BLE reste constamment en mode veille**, sauf lorsqu'une connexion est établie. Cela lui fait consommer très peu d'énergie. Le BLE consomme environ **100 fois moins d'énergie** que le **Bluetooth**. C'est pourquoi c'est le choix parfait pour la communication à courte distance avec et entre les appareils **IoT**.



De plus, **BLE** prend en charge non seulement la communication point à point, mais également le mode de diffusion et le réseau maillé (**mesh**).
Le tableau ci-dessous compare **BLE** et **Bluetooth Classic** (**BT**) plus en détail.

	Bluetooth Low Energy (LE)	Bluetooth Basic Rate/ Enhanced Data Rate (BR/EDR)
Optimized For...	Short burst data transmission	Continuous data streaming
Frequency Band	2.4GHz ISM Band (2.402 – 2.480 GHz Utilized)	2.4GHz ISM Band (2.402 – 2.480 GHz Utilized)
Channels	40 channels with 2 MHz spacing (3 advertising channels/37 data channels)	79 channels with 1 MHz spacing
Channel Usage	Frequency-Hopping Spread Spectrum (FHSS)	Frequency-Hopping Spread Spectrum (FHSS)
Modulation	GFSK	GFSK, π/4 DQPSK, 8DPSK
Power Consumption	~0.01x to 0.5x of reference (depending on use case)	1 (reference value)
Data Rate	LE 2M PHY: 2 Mb/s LE 1M PHY: 1 Mb/s LE Coded PHY (S=2): 500 Kb/s LE Coded PHY (S=8): 125 Kb/s	EDR PHY (8DPSK): 3 Mb/s EDR PHY (π/4 DQPSK): 2 Mb/s BR PHY (GFSK): 1 Mb/s
Max Tx Power*	Class 1: 100 mW (+20 dBm) Class 1.5: 10 mW (+10 dbm) Class 2: 2.5 mW (+4 dBm) Class 3: 1 mW (0 dBm)	Class 1: 100 mW (+20 dBm) Class 2: 2.5 mW (+4 dBm) Class 3: 1 mW (0 dBm)
Network Topologies	Point-to-Point (including piconet) Broadcast Mesh	Point-to-Point (including piconet)

En raison de ses propriétés, **BLE** convient aux applications qui ont besoin d'échanger de petites quantités de données s'exécutant périodiquement sur une pile bouton. Par exemple, **BLE** est d'une grande utilité dans les secteurs de la santé, du fitness, du ***tracking***, des balises (***beacons***), de la sécurité et de la domotique.

4.3.2 Server BLE (notifier) et Client BLE

Avec Bluetooth Low Energy, il existe deux types d'appareils: le **serveur** et le **client**. L'ESP32 peut faire office de **client** ou de **serveur**.

Le serveur annonce son existence, il peut donc être trouvé par d'autres appareils et il contient les données que le client peut lire. Le client analyse les périphériques à proximité et lorsqu'il trouve le serveur qu'il recherche, il établit une connexion et écoute les données entrantes. C'est ce qu'on appelle la **communication point à point**.

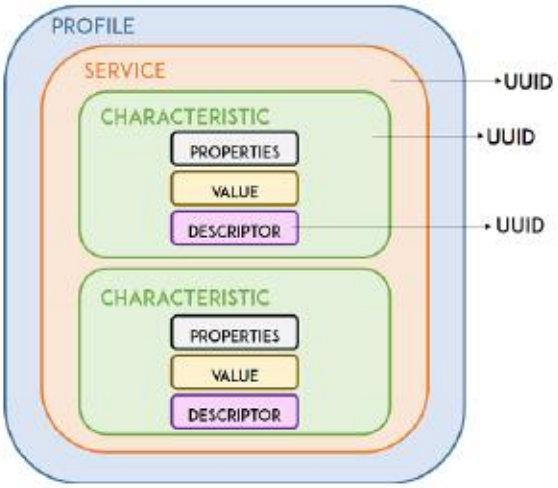
Comme mentionné précédemment, **BLE** prend également en charge le mode de **diffusion** et le **réseau maillé**:

- **Mode diffusion**: le serveur transmet des données à de nombreux clients connectés;
- **Réseau maillé**: tous les appareils sont connectés, il s'agit d'une connexion **plusieurs à plusieurs**.

Même si les configurations de réseau de diffusion et de maillage sont possibles à implémenter, elles ont été développées très récemment, donc il n'y a pas beaucoup d'exemples implémentés pour l'ESP32 en ce moment.

4.3.3 GATT

GATT signifie **Generic Attributes** et définit une structure de données hiérarchique qui est exposée aux périphériques BLE connectés. Cela signifie que GATT définit la façon dont deux appareils BLE envoient et reçoivent des **messages standard**. La compréhension de cette hiérarchie est importante, car elle facilitera la compréhension de l'utilisation du BLE et de l'écriture de vos applications.



4.3.4 Services BLE

Le niveau supérieur de la hiérarchie est un profil, qui est composé d'un ou plusieurs services. Habituellement, un appareil BLE contient plusieurs services.

Chaque service contient au moins une caractéristique et peut également référencer d'autres services.
Un **service** est simplement une collection d'informations, comme les **lectures de capteurs**, par exemple.

Il existe des **services prédéfinis** pour plusieurs types de données définies par le **SIG** (Bluetooth **Special Interest Group**) comme: le niveau de la batterie, la pression artérielle, la fréquence cardiaque, l'échelle de poids, etc. Vous pouvez consulter ici d'autres services définis.

Name	Uniform Type Identifier	Assigned Number	Specification
Generic Access	org.bluetooth.service.generic_access	0x1800	GSS
Alert Notification Service	org.bluetooth.service.alert_notification	0x1811	GSS
Automation IO	org.bluetooth.service.automation_io	0x1815	GSS
Battery Service	org.bluetooth.service.battery_service	0x180F	GSS
Blood Pressure	org.bluetooth.service.blood_pressure	0x1810	GSS
Body Composition	org.bluetooth.service.body_composition	0x181B	GSS
Bond Management Service	org.bluetooth.service.bond_management	0x181E	GSS
Continuous Glucose Monitoring	org.bluetooth.service.continuous_glucose_monitoring	0x181F	GSS
Current Time Service	org.bluetooth.service.current_time	0x1805	GSS
Cycling Power	org.bluetooth.service.cycling_power	0x1818	GSS
Cycling Speed and Cadence	org.bluetooth.service.cycling_speed_and_cadence	0x1816	GSS
Device Information	org.bluetooth.service.device_information	0x180A	GSS
Environmental Sensing	org.bluetooth.service.environmental_sensing	0x181A	GSS
Fitness Machine	org.bluetooth.service.fitness_machine	0x1826	GSS
Generic Attribute	org.bluetooth.service.generic_attribute	0x1801	GSS

4.3.5 Caractéristiques BLE

La caractéristique appartient toujours à un service et c'est là que les données réelles sont contenues dans la hiérarchie (valeur).

La caractéristique a toujours **deux attributs**:

1. la déclaration de caractéristique (qui fournit des métadonnées sur les données) et
2. la valeur de caractéristique.

De plus, la **valeur** de caractéristique peut être suivie de descripteurs, qui développent davantage les métadonnées contenues dans la déclaration de caractéristique.
Les propriétés décrivent comment peut interagir avec la valeur de caractéristique. Fondamentalement, ils contiennent les opérations et procédures pouvant être utilisées avec la caractéristique:

- Diffusion
- Lecture
- Écriture sans réponse
- Écriture
- Notification
- Écritures authentifiées et signées
- Propriétés étendues

4.3.5.1 UUID

Chaque **service**, caractéristique et descripteur possède un **UUID** (**Universally Unique Identifier**).
Un **UUID** est un numéro unique de **128 bits** (16 octets).
Par exemple:

`55072829-bc9e-4c53-938a-74a6d4c78776`

Il existe des UUID raccourcis pour tous les types, services et profils spécifiés dans le SIG (Bluetooth **Special Interest Group**). Mais si votre application a besoin de son **propre UUID**, vous pouvez le générer à l'aide de ce site Web de générateur d'UUID. `https://www.uuidgenerator.net/`

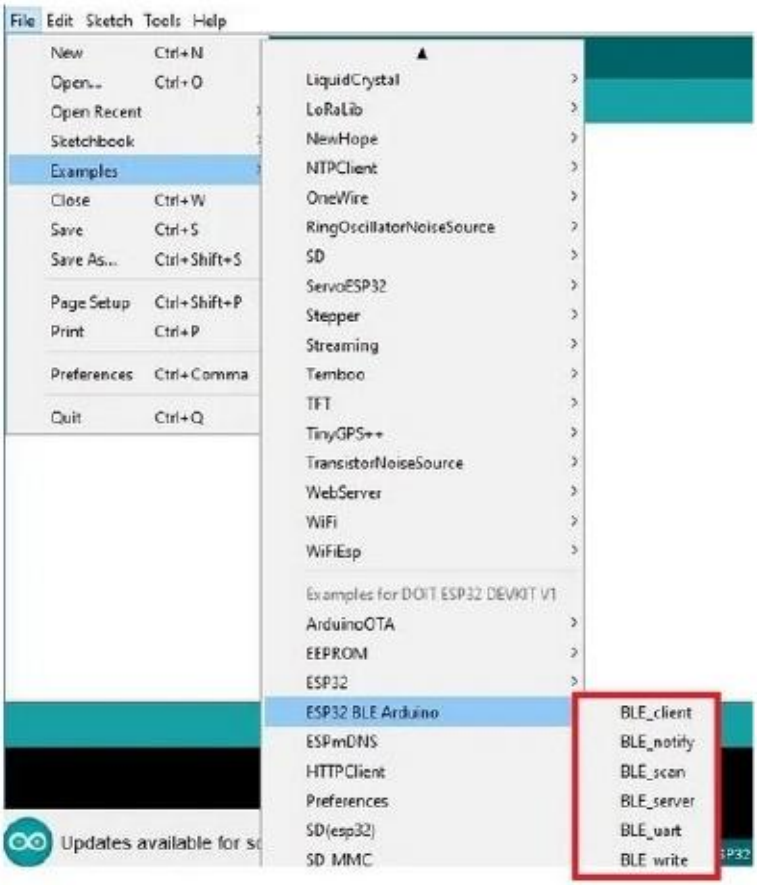
En résumé, **l'UUID** est utilisé pour identifier de manière unique les informations.
Par exemple, il peut identifier un **service particulier** fourni par un appareil Bluetooth.

4.4 BLE avec ESP32

L'ESP32 peut agir comme un **serveur BLE** ou comme un **client BLE**. Il existe plusieurs exemples BLE pour l'ESP32 dans la bibliothèque ESP32 BLE pour Arduino IDE. Cette bibliothèque est installée par défaut lorsque vous installez l'ESP32 sur l'IDE Arduino.
Dans votre Arduino IDE, vous pouvez aller dans **`File->Examples->ESP32 BLE Arduino`** et explorer les exemples fournis avec la bibliothèque **BLE**.

Remarque: pour voir les exemples ESP32, vous devez avoir la carte ESP32 sélectionnée dans **`Tools->Board`** , par exemple : **(Heltec LoRa WiFi)**.

Pour une brève introduction à l'ESP32 avec BLE sur l'IDE Arduino, nous allons créer un **serveur** ESP32 BLE, puis un **scanner** ESP32 BLE pour trouver ce serveur.
Nous utiliserons et expliquerons les exemples fournis avec la **bibliothèque BLE**.



4.4.1 ESP32 : Serveur BLE - notifier

Pour créer un serveur ESP32 BLE, ouvrez votre Arduino IDE et allez dans **`File->Examples ESP32 BLE`** Arduino et sélectionnez l'exemple **`BLE_server`**. Le code suivant devrait se charger et vous le modifiez selon le besoin.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"

char value[8]="ESP32";

void setup() {
  Serial.begin(9600);
```

```
  Serial.println("Starting BLE work!");
  BLEDevice::init("IoTDevkit-ESP32");  // put here device name
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);
    BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                           CHARACTERISTIC_UUID,
                                           BLECharacteristic::PROPERTY_READ |
                                           BLECharacteristic::PROPERTY_WRITE
                                         );

  pService->start();
  // BLEAdvertising *pAdvertising = pServer->getAdvertising();  // this still is
working for backward compatibility
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x06); // to help with iPhone connections issue
  pAdvertising->setMinPreferred(0x12);
  BLEDevice::startAdvertising();
  Serial.println("Characteristic defined! Now you can read it in your phone!");
  pCharacteristic->setValue("Hello World says Smartcomputerlab");
  pCharacteristic->notify();
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(2000);
}
```

Comment fonctionne le code

Voyons rapidement comment fonctionne l'exemple de code du serveur BLE. Il commence par importer les bibliothèques nécessaires pour les capacités BLE.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
```

Ensuite, vous devez définir un `UUID` pour le service et la caractéristique (**Service** , **Characteristic**).

```
#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
```

Vous pouvez laisser les **UUID** par défaut ou vous rendre sur `uuidgenerator.net` pour créer des **UUID** aléatoires pour vos services et caractéristiques.
Dans la fonction `setup()`, il démarre la communication série à un débit de 9600 bauds.

```
Serial.begin(9600);
```

Ensuite, vous créez un périphérique **BLE** appelé "`IoTDevkit-ESP32`". Vous pouvez changer ce nom en ce que vous voulez.

```
// Create the BLE Device
BLEDevice::init("IoTDevkit-ESP32");
```

Dans la ligne suivante, vous définissez le périphérique **BLE** en tant que **serveur**.

```
BLEServer *pServer = BLEDevice::createServer();
```

Après cela, vous créez un service pour le serveur BLE avec l'UUID défini précédemment.

```
BLEService *pService = pServer->createService(SERVICE_UUID);
```

Ensuite, vous définissez la **caractéristique de ce service**. Comme vous pouvez le voir, vous utilisez également l'UUID défini précédemment et vous devez passer comme **arguments** les propriétés de la caractéristique. Dans ce cas, c'est: **`READ`** et **`WRITE`**.

```
BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                     CHARACTERISTIC_UUID,
                                     BLECharacteristic::PROPERTY_READ |
                                     BLECharacteristic::PROPERTY_WRITE
                                     );
```

Après avoir créé la caractéristique, vous pouvez définir sa **`valeur`** avec la méthode **`setValue ()`**.

```
pCharacteristic->setValue("Hello World from Smartcomputerlab");
```

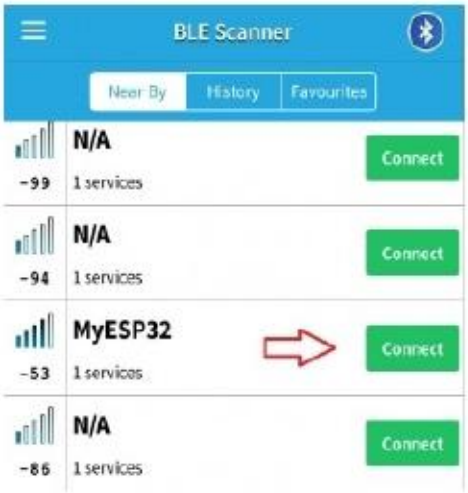
Dans ce cas, nous définissons la valeur sur le texte **`Hello World from Smartcomputerlab`**. Vous pouvez changer ce texte comme bon vous semble. Dans les projets futurs, ce texte peut être une lecture de capteur, ou l'état d'une lampe, par exemple.
Enfin, vous pouvez démarrer le service et la publicité (**advertising**), afin que d'autres appareils BLE puissent numériser et trouver cet appareil BLE.

```
BLEAdvertising *pAdvertising = pServer->getAdvertising();
pAdvertising->start();

pCharacteristic->notify();
```

Ceci est juste un exemple simple sur la façon de créer un serveur BLE. Dans ce code, rien n'est fait dans la fonction (tache) **`loop()`**, mais vous pouvez ajouter une action lorsqu'un nouveau client se connecte.

4.4.2 Application BLE Scanner

La plupart des smartphones modernes devraient avoir des capacités BLE. Vous pouvez scanner votre serveur ESP32 BLE avec votre smartphone et voir ses services et caractéristiques. Pour cela, nous utiliserons une application gratuite appelée **`BLE scanner`** , elle fonctionne sur **Android** (Google Play Store) et **iOS** (App Store).
N'oubliez pas d'aller dans les paramètres Bluetooth et d'**activer l'adaptateur Bluetooth** sur votre smartphone. Vous pouvez également vouloir le rendre visible aux autres appareils pour tester d'autres croquis plus tard.



Une fois que tout est prêt dans votre smartphone et que l'ESP32 exécute le code du serveur BLE, dans l'application, appuyez sur le bouton de **SCANNER** pour rechercher les appareils à proximité.
Vous devriez trouver un ESP32 avec le nom "**`MyESP32`**".

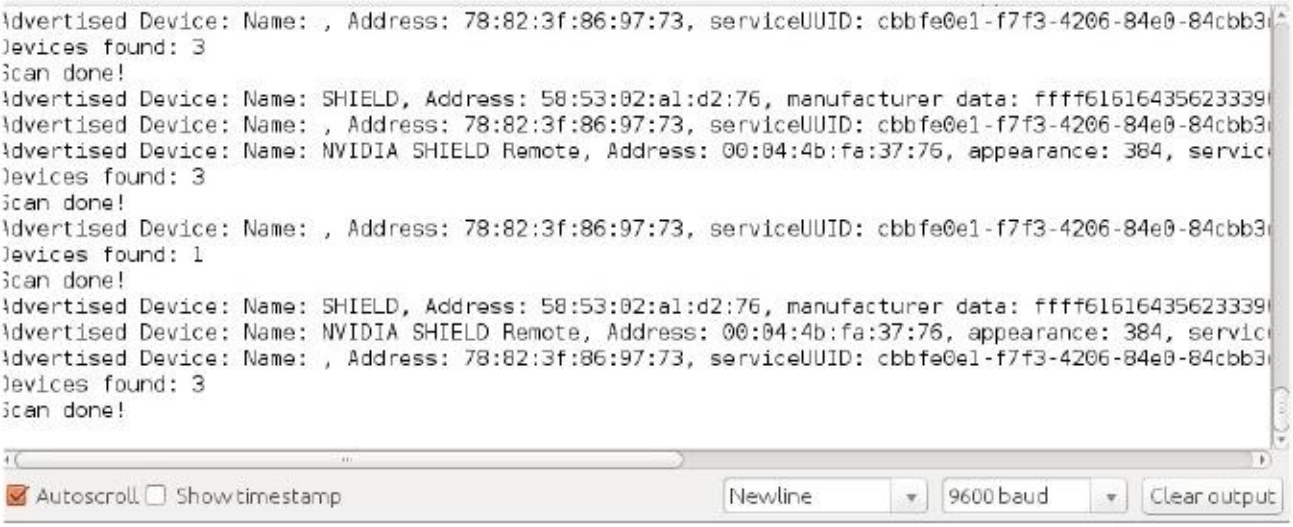
Cliquez sur le bouton **Connect**.

4.4.3 ESP32 : Scanner BLE

La création d'un scanner ESP32 BLE est simple. (Prenez un autre ESP32 (pendant que l'autre exécute l'esquisse du serveur BLE). Dans votre Arduino IDE, allez dans **File>Examples>ESP32 BLE Arduino** et sélectionnez l'exemple **BLE_scan**. Le code suivant devrait se charger.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h>
int scanTime = 5; //In seconds
BLEScan* pBLEScan;

class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
    void onResult(BLEAdvertisedDevice advertisedDevice) {
      Serial.printf("Advertised Device: %s \n",
advertisedDevice.toString().c_str());
    }
};

void setup() {
  Serial.begin(9600);
  Serial.println("Scanning...");
  BLEDevice::init("");
  pBLEScan = BLEDevice::getScan(); //create new scan
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true); //active scan uses more power, but get results
faster
  pBLEScan->setInterval(100);
  pBLEScan->setWindow(99);  // less or equal setInterval value
}

void loop() {
  BLEScanResults foundDevices = pBLEScan->start(scanTime, false);
  Serial.print("Devices found: ");
  Serial.println(foundDevices.getCount());
  Serial.println("Scan done!");
  pBLEScan->clearResults(); // delete results fromBLEScan buffer memory
  delay(2000);
}
```

Voici le résultat d'un scan :



Ce code initialise l'ESP32 en tant que **périphérique BLE** et recherche les périphériques à proximité.
Téléchargez ce code sur votre ESP32.

Vous pouvez déconnecter l'autre ESP32 de votre ordinateur et l'alimenter par la batterie de DevKit, vous êtes donc sûr de télécharger le code sur la bonne carte ESP32.

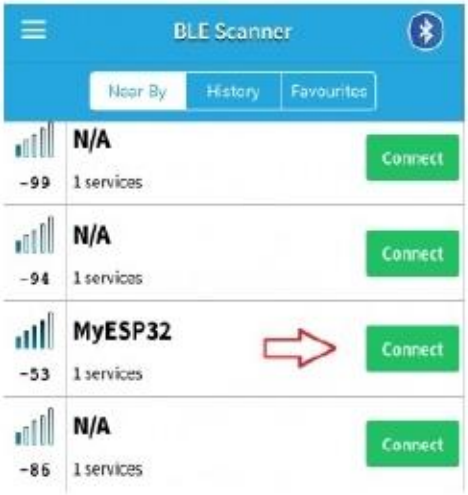
Une fois le code téléchargé et vous devriez avoir les deux cartes ESP32 sous tension:

- Un ESP32 avec l'esquisse **`BLE_server`**;
- Autre avec esquisse ESP32 **`BLE_scan`**.

Accédez au moniteur série avec l'ESP32 exécutant l'exemple «**`BLE_scan`**», appuyez sur le bouton **ACTIVER** ESP32 (avec l'esquisse «**`BLE_scan`**») pour redémarrer et attendez quelques secondes pendant la numérisation.

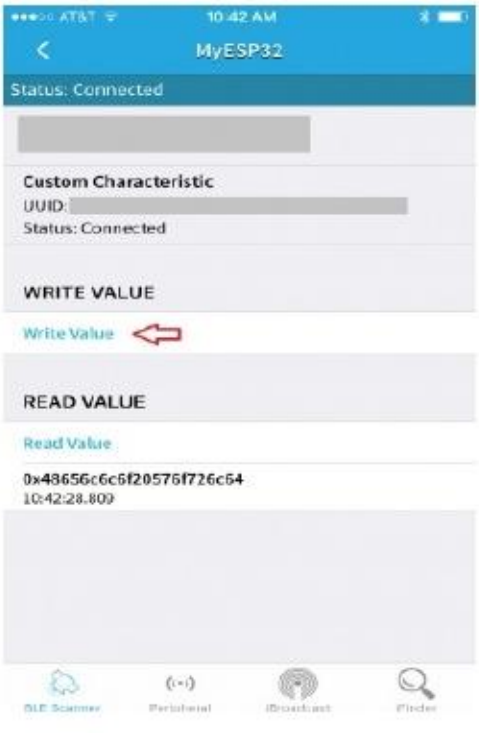
4.4.4 Test d'un client ESP32 BLE avec votre smartphone

La plupart des smartphones modernes devraient avoir des capacités BLE. Vous pouvez scanner votre serveur ESP32 BLE avec votre smartphone et voir ses services et caractéristiques. Pour cela, nous utiliserons une application gratuite appelée **BLE scanner** , elle fonctionne sur **Android** (Google Play Store) et **iOS** (App Store).
N'oubliez pas d'aller dans les paramètres Bluetooth et d'**activer l'adaptateur Bluetooth** sur votre smartphone.
Vous pouvez également vouloir le rendre visible aux autres appareils pour tester d'autres croquis plus tard.



Une fois que tout est prêt dans votre smartphone et que l'ESP32 exécute le code du serveur BLE, dans l'application, appuyez sur le bouton de **SCANNER** pour rechercher les appareils à proximité.
Vous devriez trouver un ESP32 avec le nom "**`MyESP32`**".

Cliquez sur le bouton **Connect**.



Comme vous pouvez le voir dans les figures ci-dessus, l'ESP32 dispose d'un **service** avec **l'UUID** que vous avez défini précédemment. Si vous appuyez sur le service, il élargit le menu et affiche la **caractéristique** avec **l'UUID** que vous avez également défini.

La caractéristique a les propriétés **READ** et **WRITE**, et la valeur est celle que vous avez précédemment définie dans l'esquisse du serveur BLE. Donc, tout fonctionne bien.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
#define SERVICE_UUID        "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
#include <U8x8lib.h>  // bibliothèque à charger a partir de
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15, 4, 16);
char texte[40];
bool received = false;
char car;
int var=0;

class MyCallbacks:
    public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string value = pCharacteristic->getValue();
      int i;
      if (value.length() > 0) {
        Serial.println("*********");
        Serial.print("New value: ");
        for (int i = 0; i < value.length(); i++)
          {
            Serial.print(value[i]);
            }
        delay(1000);
        Serial.println("test");
        //received = true;
        for(i=0;i< value.length();i++) texte[i]=(char)value[i];
      }
      else
        {
        Serial.println();
        Serial.println("*********");
      }
    }
};

void setup() {
  Serial.begin(9600);
  Serial.println("1- Download and install an BLE scanner app in your phone");
  Serial.println("2- Scan for BLE devices in the app");
  Serial.println("3- Connect to MyESP32");
  Serial.println("4- Go to CUSTOM CHARACTERISTIC in CUSTOM SERVICE and write
something");
  Serial.println("5- See the magic =)");
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.clear();
  u8x8.drawString(0,0,"Start BLE");
  BLEDevice::init("MyESP32");
  BLEServer *pServer = BLEDevice::createServer();
  BLEService *pService = pServer->createService(SERVICE_UUID);

BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                        CHARACTERISTIC_UUID,
```

```
                                        BLECharacteristic::PROPERTY_READ |
                                        BLECharacteristic::PROPERTY_WRITE
                                      );
  pCharacteristic->setCallbacks(new MyCallbacks());
  pCharacteristic->setValue("Hello World");
  pService->start();
  BLEAdvertising *pAdvertising = pServer->getAdvertising();
  pAdvertising->start();
}

void loop() {
  //xQueueReceive( xQueue, texte, 1000 );
  if(texte[0])  { Serial.println("received"); Serial.println(texte);}
  u8x8.clear();
  u8x8.drawString(0, 2,"Received");
  u8x8.drawString(0, 4,texte);
  delay(5000);
}
```

4.4.5 Serveur BLE avec un capteur– et fonction `notify`

L'exemple suivant active les fonctions du serveur puis il attend la notification du client. Il se connecte ensuite au client et envoie les données capturées à partir d'un capteur type **SHT21**.

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

#include <Wire.h>  // include SHT21 library
#include <SHT21.h>  // include SHT21 library
SHT21 sht;

BLEServer *pServer = NULL;
BLECharacteristic * pTxCharacteristic;
bool deviceConnected = false;
bool oldDeviceConnected = false;
uint8_t txValue = 0;

#define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
// UART service UUID
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };

    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};
```

```
class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string rxValue = pCharacteristic->getValue();
     if (rxValue.length() > 0) {
        Serial.println("*********");
        Serial.print("Received Value: ");
        for (int i = 0; i < rxValue.length(); i++)
          Serial.print(rxValue[i]);
        Serial.println();
        Serial.println("*********");
      }
    }
};

void setup() {
  Serial.begin(9600);
  Wire.begin(21,22);
  // Create the BLE Device
  BLEDevice::init("ESP32 BLE Service");
  // Create the BLE Server
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  // Create the BLE Service
  BLEService *pService = pServer->createService(SERVICE_UUID);
  // Create a BLE Characteristic
  pTxCharacteristic = pService->createCharacteristic(
            CHARACTERISTIC_UUID_TX,
            BLECharacteristic::PROPERTY_NOTIFY
            );

  pTxCharacteristic->addDescriptor(new BLE2902());
  BLECharacteristic * pRxCharacteristic = pService->createCharacteristic(
            CHARACTERISTIC_UUID_RX,
            BLECharacteristic::PROPERTY_WRITE
            );
  pRxCharacteristic->setCallbacks(new MyCallbacks());
  // Start the service
  pService->start();
  // Start advertising
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
}

union {
  uint8_t frame[16];
  float sensor[4];
  char text[16];
} sfr;

union {
  uint8_t frame[64];
  char text[64];
} buff;

void loop()
{
    if (deviceConnected) {
        sfr.sensor[0] = sht.getTemperature();  // get temp from SHT
```

```
        sfr.sensor[1] = sht.getHumidity(); // get temp from SHT
        sprintf(buff.text,"Temp:%2.2f, Humi:%2.2f",sfr.sensor[0],sfr.sensor[1]);
       pTxCharacteristic->setValue(buff.frame,strlen(buff.text));
        pTxCharacteristic->notify();
            delay(3000); // bluetooth stack may go into congestion
      }
    // disconnecting
    if (!deviceConnected && oldDeviceConnected) {
        delay(500); // give the bluetooth stack the chance to get things ready
        pServer->startAdvertising(); // restart advertising
        Serial.println("start advertising");
        oldDeviceConnected = deviceConnected;
    }
    // connecting
    if (deviceConnected && !oldDeviceConnected) {
            // do stuff here on connecting
        oldDeviceConnected = deviceConnected;
    }
}
```

4.5 Développement d'une passerelle BLE-WiFi vers serveur IoT

Le client BLE présenté dans l'exemple précédant reçoit les messages avec les données du capteur et les affiche sur l'écran OLED. Dans l'exemple suivant, nous montrons comment créer une simple passerelle BLE-WiFi qui envoie les **données reçues de votre smartphone** au serveur **ThingSpeak**.

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <WiFi.h>
#include "ThingSpeak.h"
WiFiClient  client;
const char* ssid = "Livebox-08B0";
const char* pass = "G79ji6dtEptVTPWmZP";
unsigned long myChannelNumber =  1088701;  // IoTDevKit1
const char * myWriteAPIKey = "VKRHS9PGD9K1HJ6R";
const char * myReadAPIKey ="OP2YX22NPFCO763Z";

BLEServer *pServer = NULL;
BLECharacteristic * pTxCharacteristic;
bool deviceConnected = false;
bool oldDeviceConnected = false;
uint8_t txValue = 0;

#define SERVICE_UUID  "6E400001-B5A3-F393-E0A9-E50E24DCCA9E" // UART UUID
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };
    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string rxValue = pCharacteristic->getValue();
      float temp=21.65,humi=56.8; char val[64];
      if (rxValue.length() > 0) {
        Serial.println("*********");
        Serial.print("Received Value: ");
        for (int i = 0; i < rxValue.length(); i++)
         { Serial.print(rxValue[i]); val[i]=rxValue[i]; }
        Serial.println();
        Serial.println("*********");
    temp=(float)atof(val);
    ThingSpeak.setField(1, temp);  // préparation du field1
    ThingSpeak.setField(2, humi);  // préparation du field2
    while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
    }
    ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
      }
    }
};
```

```
void setup() {
  Serial.begin(9600);
  WiFi.disconnect(true);  // effacer de l'EEPROM WiFi credentials
  delay(1000);
  WiFi.begin(ssid, pass);
  delay(1000);
    while (WiFi.status() != WL_CONNECTED) {
          delay(500); Serial.print(".");
    }
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  Serial.println("WiFi setup ok");
  delay(1000);
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Wire.begin(21,22);
  // Create the BLE Device
  BLEDevice::init("ESP32 BLE Service");
  // Create the BLE Server
  pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
  // Create the BLE Service
  BLEService *pService = pServer->createService(SERVICE_UUID);
  // Create a BLE Characteristic
  pTxCharacteristic = pService->createCharacteristic(
      CHARACTERISTIC_UUID_TX,
      BLECharacteristic::PROPERTY_NOTIFY
      );
  pTxCharacteristic->addDescriptor(new BLE2902());
  BLECharacteristic * pRxCharacteristic = pService->createCharacteristic(
      CHARACTERISTIC_UUID_RX,
      BLECharacteristic::PROPERTY_WRITE
      );
  pRxCharacteristic->setCallbacks(new MyCallbacks());
  // Start the service
  pService->start();
  // Start advertising
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
}

union {
  uint8_t frame[16];
  float sensor[4];
  char text[16];
} sfr;

union {
  uint8_t frame[64];
  char text[64];
} buff;
```

```
void loop()
{
    if (deviceConnected) {
        sfr.sensor[0] = sht.getTemperature();  // get temp from SHT
        sfr.sensor[1] = sht.getHumidity(); // get temp from SHT
        sprintf(buff.text,"Temp:%2.2f, Humi:%2.2f",sfr.sensor[0],sfr.sensor[1]);
        pTxCharacteristic->setValue(buff.frame,strlen(buff.text));
        pTxCharacteristic->notify();
      delay(3000); // bluetooth stack
      }
    // disconnecting
    if (!deviceConnected && oldDeviceConnected) {
        delay(500); // give the bluetooth stack the chance to get things ready
        pServer->startAdvertising(); // restart advertising
        Serial.println("start advertising");
        oldDeviceConnected = deviceConnected;
    }
    // connecting
    if (deviceConnected && !oldDeviceConnected) {
            // do stuff here on connecting
        oldDeviceConnected = deviceConnected;
    }
}
```

Channel Stats

Created: 5 months ago
Entries: 5

4.6 Résumé

Dans ce laboratoire, nous vous avons montré les principes de base du **Bluetooth Classic** et du **Bluetooth Low Energy** et vous avons montré quelques exemples avec l'ESP32. Nous avons exploré le code du serveur BLE et l'esquisse du client BLE.
Le dernier exemple donné est une **passerelle BLE-WiFi** qui montre comment nous pouvons exploiter ces deux modules radio pour construire des architectures IoT.

Travail à faire (sur une carte DevKit)

1. **Développer une application** client-**serveur** BLE où le DevKit envoie (serve) les donnes captées par un **SHT21** vers votre smartphone (à la place du message

```
if (deviceConnected) {
    pCharacteristic->setValue((uint8_t*)&value, 16);
    pCharacteristic→notify();
  ..
```

2. **Développer une application client**-serveur **BLE** où le **DevKit** reçoit un message envoyé par votre smartphone .

```
class MyCallbacks:
    public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string value = pCharacteristic→getValue();
..
```

3. **Développer une application** passerelle **BLE-WiFi** où le **DevKit** reçoit un message type **valeur numérique** envoyé par votre smartphone (par BLE) et l'achemine vers le serveur IoT – ThingSpeak par une connexion WiFi.

Laboratoire 5 - Développement de serveurs locaux WEB-IoT

Dans ce laboratoire, nous nous concentrerons sur la mise en œuvre de simples serveurs WEB locaux pour contrôler depuis un navigateur Web les terminaux et les passerelles IoT.

5.0 ESP32 – organisation de la mémoire

Avant d'aborder notre sujet principal, qui est le développement des micro-serveurs WEB (IoT) nous allons nous intéresser à l'organisation de la mémoire dans un ESP32. Les mémoires internes **EEPROM** et **SRAM** permettent de loger la partie essentielle des données en traitement et de préserver certains paramètres du système.

La section **interne** d'**EEPROM** (max 512 bytes) est utilisée, entre autres, pour garder les derniers **identificateurs WiFi (credentials : `ssid`, `passwd`)**. Nous pouvons y accéder grâce à la bibliothèque **`EEPROM.h`**.

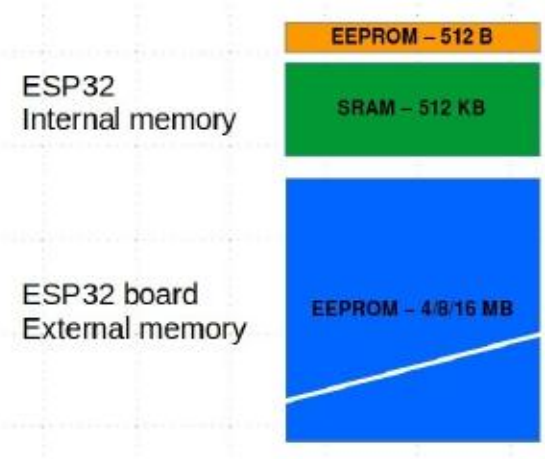


Figure 5.1 Organisation (simplifiée) de la mémoire d'un ESP32

Voici un exemple :

```
#include <EEPROM.h>
#define EEPROM_SIZE 1        // number of bytes you want to access
const int buttonPin = 0;     // the number of the pushbutton pin
const int ledPin = 25;       // the number of the LED pin
int ledState = HIGH;          // the current state of the output pin
int buttonState;              // the current reading from the input pin
int lastButtonState = LOW;   // the previous reading from the input pin
// variables are unsigned longs because the time is measured in milliseconds,
unsigned long lastDebounceTime = 0;  // the last time the output pin was toggled
unsigned long debounceDelay = 50;    // the debounce time

void setup() {
  Serial.begin(9600);
  // initialize EEPROM with predefined size
  EEPROM.begin(EEPROM_SIZE);  // max 512
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
  // read the last LED state from flash memory
  ledState = EEPROM.read(0);  // set the LED to the last stored state
  delay(100);
  Serial.println();
  Serial.print("Saved state:");Serial.println(ledState);
  digitalWrite(ledPin, ledState);
}

void loop() {
  int reading = digitalRead(buttonPin);  // pressed to  GND
  if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
```

```
  }
  if((millis()-lastDebounceTime)>debounceDelay) {
      // must be pressed at least debounce time
    if (reading != buttonState) {
      buttonState = reading;
      // only toggle the LED if the new button state is LOW
      if (buttonState == LOW) {
        ledState = !ledState;
      }
    }
  }

  lastButtonState = reading;    // save the last reading
  // if the ledState variable is different from the current LED state
  if (digitalRead(ledPin)!= ledState) {
    Serial.print("State changed to:");Serial.println(ledState);
    digitalWrite(ledPin, ledState);     // change the LED state
    // save the LED state in flash memory
    EEPROM.write(0, ledState);
    EEPROM.commit();  // must commit to EEPROM from write buffer
    Serial.println("State saved in flash memory");
  }
}
```

5.1 `WiFiManager` – Initialisation des identifiants (`credentials`) WiFi

Lorsque votre ESP32 démarre, il se met en mode **Station** (`STA`) et tente de se connecter à un point d'accès précédemment enregistré dans sa **EEPROM** si cela échoue (ou aucun réseau précédent n'est enregistré), il place l'**ESP** en mode **Point d'Accès** (`AP`) et fait tourner un `DNS` et un **serveur Web** (avec une **adresse IP** `192.168.4.1`, et le numéro de **port 80** par défaut)

En utilisant n'importe quel appareil compatible WiFi avec un navigateur (ordinateur, téléphone, tablette) connectez-vous au point d'accès nouvellement créé (par exemple : `ESP32`).

En raison du portail captif et du serveur DNS, vous obtiendrez une fenêtre contextuelle de type `` Rejoindre au réseau '' ou tout domaine auquel vous essayez d'accéder redirigé vers le portail de configuration.

Choisissez l'un des points d'accès présentés (par exemple le WiFi de votre Box Internet), entrez le mot de passe, cliquez sur **Enregistrer.** ESP essaiera de se connecter. En cas de succès, il abandonne le contrôle à votre application. Sinon, reconnectez-vous à AP et reconfigurez.

Figure 5.2 Page WEB généré à l'adresse `192.168.4.1` par le `WiFiManager`

Le code suivant permet de tester le fonctionnement de **WiFiManager**. Il peut être réutilisé dans les application qui nécessite une connexion WiFi avec un Point d'Accès initialement inconnu.

```
#include "Arduino.h"
#include <WiFiManager.h>
WiFiManager wm;
const char* ssid = "ESP32";
const char* password = "smartcomputerlab";
#define buttonPin  0 // PRG button on ESP32 board
#include <U8x8lib.h> // bibliothèque à charger a partir de
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15,4,16); //SDA, SCL, RST
int count=10;
IPAddress ip;
char buff[32];

void setup()
{
  Serial.begin(9600);
  u8x8.begin(); // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);u8x8.clear();
  wm.resetSettings();  // resests the EEPROM credentials, to be commented !!!
  u8x8.drawString(0,0,"Starting WiFi");
  pinMode(buttonPin,INPUT_PULLUP);
  WiFi.begin(); //Wait for WiFi to connect to AP
  Serial.println("Waiting for WiFi");
  while (WiFi.status()!= WL_CONNECTED && count>0){
    delay(500); count--; Serial.print(count);}
  if(!count)
  {
  u8x8.drawString(0,1,"SmartConfig");
  u8x8.drawString(0,2,"Launch App");
  u8x8.drawString(0,3,"Set SSID");
  u8x8.drawString(0,4,"Enter PASS");
  delay(2000);
  if(!wm.autoConnect(ssid, password))  // start WiFi Manager
    Serial.println("Erreur de connexion.");
  else
    Serial.println("Connexion etablie!");
  delay(1000);
  Serial.println("");
  Serial.println("Waiting for WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500); Serial.print(".");
    }
  }
  Serial.println("WiFi Connected.");
  u8x8.drawString(0,5,"WiFi Connected");
  Serial.print("IP Address: ");
  ip=WiFi.localIP();
  Serial.println(ip); sprintf(buff,"%d.%d.%d.%d",ip[0],ip[1],ip[2],ip[3]);
  u8x8.drawString(0,6,buff);
  delay(3000);
}

void loop() {
// put your main code here, to run repeatedly:
Serial.println("in the loop");
delay(2000);
}
```

La ligne :

```
//    wm.resetSettings();
```

permet d'**effacer les identifiants** WiFi (**ssid**, **password**) stockés sur **l'EEPROM** interne.

5.2 Serveurs WEB simples en mode station (STA) et en mode point d'accès (AP)

Pour commencer, nous présentons un code Arduino simple avec le serveur Web intégré directement dans le programme. Ce programme propose une simple fonctionnalité WEB pour allumer et éteindre une LED (**broche 25**) et un relais sur une carte d'extension de type **mini D1** (si il est là). Les pages WEB sont générées par les fonctions **server.send()** dans lesquelles est incorporé le code texte. Ces fonctions sont appelées lorsqu'une requête HTML est envoyée.
Le programme suivant peut servir de base pour plus de développements. Par exemple, en HTML, nous pouvons intégrer des formulaires pour préparer les données/paramètres à envoyer à l'application.

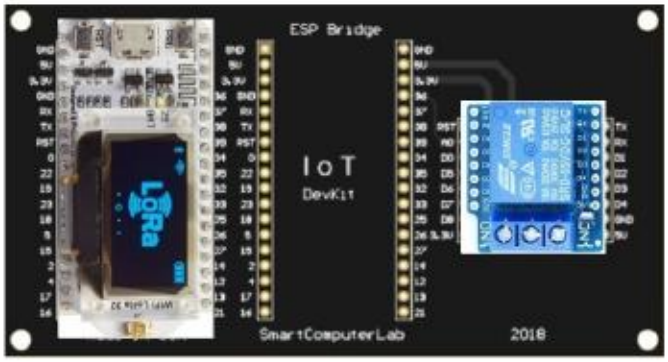


Figure 5.3 Un carte d'extension de type relais

5.2.1 Serveur WEB en mode station - STA

Le serveur ci-dessus est accessible à toutes les stations clientes connectées à Internet par le même point accès.
Le point d'accès utilisé peut être fixé d'avance sur:

ssid - **"PhoneAP"**, **password** - **"smartcomputerlab"**, pour un smartphone

ou sur :

ssid - "Livebox-08B0", password - "G79ji6dtEptVTPWmZP", pour une box

Il peut être également identifié par le **WiFiManager** (le cas du programme proposé ci dessous)
Le numéro de **port** est fixé à **5000** , mais il peut être également modifié.

```
#include "Arduino.h"
#include <WiFiManager.h>
WiFiManager wm;
const char* ssid = "ESP32";
const char* password = "smartcomputerlab";
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>

int port=5000;
WebServer server(port);
const int led = 25;

void handleRoot()
{  // a simple WEB page with action
  digitalWrite(led, 1);
  server.send(200, "text/plain", "hello from esp32!");
  digitalWrite(led, 0);
}
```

```
void handleON()
{
  digitalWrite(led, 1);
  server.send(200, "text/plain", "LED set ON");
}

void handleOFF()
{
  digitalWrite(led, 0);
  server.send(200, "text/plain", "LED set OFF");
}

void handleNotFound()
{
  digitalWrite(led, 1);
  String message = "File Not Found\n\n";
  message += "URI: "; message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET)?"GET":"POST";
  message += "\nArguments: "; message += server.args();
  message += "\n";
  for (uint8_t i=0; i<server.args(); i++){
    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";}
  server.send(404, "text/plain", message); delay(1000) ;
  digitalWrite(led, 0);
}

int count=20;
IPAddress ip;

void setup(void)
{
  pinMode(led, OUTPUT);digitalWrite(led, 0);
  Serial.begin(9600);
  WiFi.begin();
  Serial.println("Waiting for WiFi");
  while (WiFi.status()!= WL_CONNECTED && count>0) { //stored: ssid, passwd
    delay(500); count--;Serial.print(count);
  }
  if(!count)
  {
    delay(1000); // using WiFi Manager
    if(!wm.autoConnect(ssid, password)) Serial.println("Erreur de connexion.");
    else
     Serial.println("Connexion etablie!");
    delay(1000);
    Serial.println("");
    //Wait for WiFi to connect to AP
    Serial.println("Waiting for WiFi");
    while (WiFi.status() != WL_CONNECTED) {
      delay(500); Serial.print("."); }
  }
Serial.println("WiFi Connected.");
ip=WiFi.localIP();
Serial.println(ip);
if (MDNS.begin("esp32")) {
  Serial.println("MDNS responder started");
  }
```

```
// entry points to WEB pages
server.on("/", handleRoot);
server.on("/on", handleON);server.on("/off", handleOFF);
server.on("/inline", [](){
server.send(200, "text/plain", "this works as well");});
server.onNotFound(handleNotFound);
server.begin();
Serial.print("HTTP server started on port:");Serial.println(port);
}

void loop() {
server.handleClient();  // service  activation
}
```

Affichage sur le terminal :

```
Waiting for WiFi
19
WiFi Connected.
192.168.1.84
MDNS responder started
HTTP server started on port:5000
```

A faire :

1. Tester l'exemple ci dessus avec votre point d'accès fixe (une box) et/ou mobile (un smartphone).
2. Afficher sur l'écran OLED : l'adresse IP locale et l'état de LED («**on**», «**off**»)

5.2.2 Un simple serveur WEB en mode `softAP`

Le programme suivant a les mêmes fonctions que l'exemple précédent. La seule différence est le **mode d'accès**. Ici, nous avons un serveur WEB associé au point d'accès (AP). Le point d'accès a son nom (**SSID**) - `ESP32`, le mot de passe n'est pas utilisé (`open`).L'**adresse IP** par défaut est `192.168.4.1 (port=80)`

5.2.2.1 Code complet

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
int port=80;
WebServer server(port);
const int led = 25;
long int initmillis=0;
int sec=0, mnt=0, hr=0;
int ledR,ledG,ledB;  // pins : 33, 34, 35

void handleds() {  // WEB page with forms and arguments
digitalWrite(led,1);
char temp[1280];
sec = (millis()-initmillis)/1000; mnt = sec/60; hr = mnt/60;
for(int i=0; i<server.args() ;i++)
{
if(server.argName(i)=="ledR" && server.arg(i)=="on")
{ ledR=1;digitalWrite(33,LOW); Serial.println("ledR-on");}
if(server.argName(i)=="ledR" && server.arg(i)=="off")
{ ledR=0;digitalWrite(33,HIGH); Serial.println("ledR-off");}
if(server.argName(i)=="ledG" && server.arg(i)=="on")
{ ledG=1;digitalWrite(34,LOW); Serial.println("ledG-on");}
if(server.argName(i)=="ledG" && server.arg(i)=="off")
{ ledG=0;digitalWrite(34,HIGH); Serial.println("ledG-off");}
if(server.argName(i)=="ledB" && server.arg(i)=="on")
{ ledB=1;digitalWrite(35,LOW); Serial.println("ledB-on");}
if(server.argName(i)=="ledB" && server.arg(i)=="off")
{ ledB=0;digitalWrite(35,HIGH); Serial.println("ledB-off");}
}
snprintf ( temp, 1280,
"<html>\
<head>\
<meta http-equiv='refresh' content='60'/>\
<title>DevKit</title>\
<style>\
body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-Serif;
Color: #000088; }\
</style>\
</head>\
<body>\<br>\
<h2><form action='/box' method='get' target='_self'>\
<label for='ledR'>LedR</label>\
<input type='text' name='ledR' id='ledR' value='off'><br>\
<label for='ledR'>LedG</label>\
<input type='text' name='ledG' id='ledG' value='off'><br>\
<label for='ledB'>LedB</label>\
<input type='text' name='ledB' id='ledB' value='off'><br>\
<input type='submit' name='submit' value='submit'/>\
</form><br><br>\
<hr>\
Lab5 - you need the RGB module<br>\
```

```
otherwise use OLED to display the R-G-B message<br>\
Uptime safter start: %02d:%02d:%02d</h2>\
</body>\
</html>",
hr, mnt % 60, sec % 60
);
server.send ( 200, "text/html", temp ); digitalWrite (led, 0 );}
void handleRoot() {
digitalWrite(led, 1);
server.send(200, "text/plain", "hello from esp32!");
digitalWrite(led, 0);
}
void handleON() {
digitalWrite(led, 1);server.send(200, "text/plain", "LED set ON");
}

void handleOFF() {
digitalWrite(led, 0); server.send(200, "text/plain", "LED set OFF");
}
void handleNotFound(){
digitalWrite(led, 1);
String message = "File Not Found\n\n";
message += "URI: ";message += server.uri();
message += "\nMethod: ";
message += (server.method() == HTTP_GET)?"GET":"POST";
message += "\nArguments: ";
message += server.args();
message += "\n";
for (uint8_t i=0; i<server.args(); i++){
message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
}
server.send(404, "text/plain", message); digitalWrite(led, 0);
}

void setup(void){
pinMode(led,OUTPUT);pinMode(33,OUTPUT);pinMode(34,OUTPUT);pinMode(35,OUTPUT);
digitalWrite(led,0);pinMode(33,0);pinMode(34,0);pinMode(35,0);
Serial.begin(9600);
WiFi.mode(WIFI_AP);  // mode with access point
WiFi.softAP("esp32");
Serial.print("IP address: ");Serial.println(WiFi.softAPIP());
server.on("/", handleRoot);
server.on("/on", handleON); server.on("/off", handleOFF);
server.on ("/box", handleds );
server.on("/inline", [](){
server.send(200, "text/plain", "this works as well");
});
server.onNotFound(handleNotFound);
server.begin();Serial.print("HTTP server started on
port:");Serial.println(port);
}

void loop(void){ server.handleClient();}
```

A faire :

1 Modifier le contenu de la page Web pour la formater plus correctement.
2. Modifier le code pour pouvoir afficher les paramètres envoyés par la page web sur écran OLED.

5.3 Un serveur WEB avec système SPIFFS

5.3.1 Système de fichiers SPIFFS

SPIFFS - (S)erial (P)eripheral (I)nterface (F)lash (F)ile (S)ystem signifie que notre ESP peut contenir un **système de fichiers** simple dans la **mémoire du programme SPI** (flash) , qui contient également notre code de programme.
Les fichiers peuvent être créés, modifiés ou supprimés dans ce système de fichiers. Ces fichiers peuvent être utilisés ou modifiés par notre code de programme pendant le temps d'exécution, ainsi que ont été créés par nous avant.
Cette zone de mémoire, une fois générée, est préservée avec des mises à jour de code! Cela signifie qu'un programme mis à jour peut continuer à travailler avec les données stockées sous forme de fichier sur elle directement en référence au nom du fichier.
Voici un exemple du code avec un ensemble de fonctions permettant une gestion complète des fichiers enregistres dans ce système.

```
#include "FS.h"
#include "SPIFFS.h"
#define FORMAT_SPIFFS_IF_FAILED true

void listDir(fs::FS &fs, const char * dirname, uint8_t levels){
    Serial.printf("Listing directory: %s\r\n", dirname);
    File root = fs.open(dirname);
    if(!root){ Serial.println("- failed to open directory");  return; }
    if(!root.isDirectory()){ Serial.println(" - not a directory"); return; }
    File file = root.openNextFile();
    while(file){
        if(file.isDirectory()){
            Serial.print("  DIR : "); Serial.println(file.name());
            if(levels){  listDir(fs, file.name(), levels -1); }
        } else {
            Serial.print("  FILE: "); Serial.print(file.name());
            Serial.print("\tSIZE: "); Serial.println(file.size()); }
        file = root.openNextFile();
    }
}
void readFile(fs::FS &fs, const char * path){
    Serial.printf("Reading file: %s\r\n", path);
    File file = fs.open(path);
    if(!file || file.isDirectory()){
        Serial.println("- failed to open file for reading"); return;}
    Serial.println("- read from file:");
    while(file.available()){   Serial.write(file.read()); }
}

void writeFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Writing file: %s\r\n", path);
    File file = fs.open(path, FILE_WRITE);
    if(!file){Serial.println("- failed to open file for writing"); return;}
    if(file.print(message)){  Serial.println("- file written"); }
    else { Serial.println("- frite failed"); }
}

void appendFile(fs::FS &fs, const char * path, const char * message){
    Serial.printf("Appending to file: %s\r\n", path);
    File file = fs.open(path, FILE_APPEND);
    if(!file){ Serial.println("- failed to open file for appending"); return;}
    if(file.print(message)){  Serial.println("- message appended"); }
    else { Serial.println("- append failed"); }
}
```

```
void renameFile(fs::FS &fs, const char * path1, const char * path2){
    Serial.printf("Renaming file %s to %s\r\n", path1, path2);
    if (fs.rename(path1, path2)) {  Serial.println("- file renamed");  }
    else { Serial.println("- rename failed");  }
}

void deleteFile(fs::FS &fs, const char * path){
    Serial.printf("Deleting file: %s\r\n", path);
    if(fs.remove(path)){  Serial.println("- file deleted");  }
    else {   Serial.println("- delete failed"); }
}

void setup(){
    Serial.begin(9600);
    if(!SPIFFS.begin(FORMAT_SPIFFS_IF_FAILED)){
        Serial.println("SPIFFS Mount Failed");  return;  }
    Serial.println(SPIFFS.totalBytes());
    Serial.println(SPIFFS.usedBytes());
    listDir(SPIFFS, "/", 0);
    writeFile(SPIFFS, "/hello.txt", "Hello ");
    appendFile(SPIFFS, "/hello.txt", "World!\r\n");
    readFile(SPIFFS, "/hello.txt");
    renameFile(SPIFFS, "/hello.txt", "/foo.txt");
    readFile(SPIFFS, "/foo.txt"); deleteFile(SPIFFS, "/foo.txt");
    deleteFile(SPIFFS, "/test.txt");
    Serial.println( "Test complete" );
}

void loop(){}
```

5.3.2 Un serveur WEB avec le système de fichiers SPIFFS

Pour construire un **serveur WEB** avec les **pseudo-fichiers** enregistrés dans la même mémoire **Flash** que le code du programme, nous avons besoin de trois fichiers différents:

- le script Arduino,
- le fichier **HTML** et
- le fichier **CSS**.

Les fichiers **HTML** et **CSS** doivent être enregistrés dans un dossier appelé **data** dans le dossier du script Arduino, comme indiqué ci-dessous:

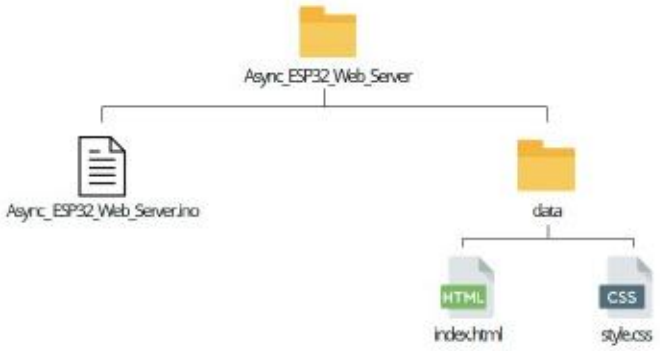


Figure 5.4 Organisation des fichiers **SPIFFS** pour un serveur WEB

Attention:

Les fichiers SPIFFS doivent être téléchargés avant de charger le script Arduino. Pour effectuer cette opération, nous devons **installer le plugin dans notre IDE**: me-no-dev/arduino-esp32fs-plugin

Ce plugin doit être décompressé dans le répertoire **~/Arduino/tools**. Après son extraction, vous devriez voir un chemin comme celui-ci:
<Rép_accueil> /Arduino/tools/ESP32FS/tool/esp32fs.jar

Usage

1. Ouvrez un script (ou créez-en une nouvelle et enregistrez-la).
2. Accédez au répertoire du script (**Sketch->Show Sketch Folder**)..
3. Créez un répertoire nommé **data** et tous les fichiers de votre choix dans le système de fichiers :
   index.html, ...
   style.css.
4. Assurez-vous que vous avez sélectionné une carte, un port et **fermé Serial Monitor**.
5. Sélectionnez l'option de menu **Tools->ESP32 Sketch Data Upload** de données ESP32 Sketch. Cela devrait commencer à télécharger les fichiers dans le système de fichiers flash ESP32.
6. Une fois terminé, la barre d'état IDE affichera le message **SPIFFS Image Uploaded**. Cela peut prendre quelques minutes pour les systèmes de fichiers de grande taille.
7. Après avoir enregistré les fichiers **SPIFFS**, vous pouvez télécharger le script compilé.

5.3.2.1 Le fichier HTML (index.html) dans le répertoire data

Ce fichier est un code HTML très simple. Il intègre une ligne de texte et 2 boutons: **ON** et **OFF**

```
<!DOCTYPE html>
<html><head>
 <title>ESP32 Web Server</title>
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <link rel="icon" href="data:,">
 <link rel="stylesheet" type="text/css" href="style.css">
</head><body>
 <h1>ESP32 Web Server</h1>
 <p>GPIO state: <strong> %STATE%</strong></p>
 <p><a href="/on"><button class="button">ON</button></a></p>
 <p><a href="/off"><button class="button button2">OFF</button></a></p>
</body></html>
```

5.3.2.2 Le fichier CSS (style.css) dans le répertoire data

Le fichier **CSS** est également assez simple; il est présenté ci-dessous:

```
html {
 font-family: Helvetica;
 display: inline-block;
 margin: 0px auto;
 text-align: center;
}
h1{
 color: #0F3376;
 padding: 2vh;
}
p{
 font-size: 1.5rem;
}
.button {
 display: inline-block;
 background-color: #008CBA;
 border: none;
 border-radius: 4px;
 color: white;
 padding: 16px 40px;
 text-decoration: none;
 font-size: 30px;
```

```
 margin: 2px;
 cursor: pointer;
}
.button2 {
 background-color: #f44336;
}
```

5.3.3.3 Le fichier principal - script Arduino

Le code Arduino (fichier `.ino`) contient des bibliothèques **`WiFi.h`** , **`AsyncTCP.h`**, **`ESPAsyncWebServer.h`** et **`SPIFFS.h`**. Ces bibliothèques vous permettent de créer un point d'accès avec un service WEB qui contient certains fichiers WEB dans le système **SPIFFS**.

```
#include <Arduino.h>
#include <WiFiManager.h>
WiFiManager wm;
const char* ssid = "ESP32";
const char* password = "smartcomputerlab";
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include "SPIFFS.h"
int count=10;
IPAddress ip;
const int ledPin = 25;
String ledState; // stores LED state
int port=80;
// Create AsyncWebServer object on port 80
AsyncWebServer server(port);
String processor(const String& var)
{
Serial.println(var);
if(var == "STATE"){
  if(digitalRead(ledPin)){ ledState = "ON";}
  else{ ledState = "OFF";}
    Serial.print(ledState);
    return ledState;
    }
return String();
}

void setup(){
Serial.begin(9600);
pinMode(ledPin, OUTPUT);
if(!SPIFFS.begin(true)){
  Serial.println("An Error has occurred while mounting SPIFFS");
  return;
  }
WiFi.begin();
//Wait for WiFi to connect to AP
Serial.println("Waiting for WiFi");
while (WiFi.status()!= WL_CONNECTED && count>0) {
  delay(500); count-;Serial.print(count); }
if(!count)
{
if(!wm.autoConnect(ssid, password))
  Serial.println("Erreur de connexion.");
else
  Serial.println("Connexion etablie!");
delay(1000);
```

```
Serial.println("");
//Wait for WiFi to connect to AP
Serial.println("Waiting for WiFi");
while (WiFi.status() != WL_CONNECTED) {
  delay(500) ;Serial.print(".");
  }
}
Serial.print("IP Address: ");
ip=WiFi.localIP();
Serial.println(ip);
// Route for root / web page
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
request->send(SPIFFS,"/index.html",String(),false,processor);
});
// Route to load style.css file
server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request){
request->send(SPIFFS, "/style.css", "text/css");
});
// Route to set GPIO to HIGH
server.on("/on", HTTP_GET, [](AsyncWebServerRequest *request){
digitalWrite(ledPin, HIGH);
request->send(SPIFFS, "/index.html", String(), false, processor);
});
// Route to set GPIO to LOW
server.on("/off", HTTP_GET, [](AsyncWebServerRequest *request){
digitalWrite(ledPin, LOW);
request->send(SPIFFS, "/index.html", String(), false, processor);
});
server.begin();
// Start server
}
void loop()
{
Serial.println("in the loop"); delay(2000);
}


}
```

Figure 5.5 Page WEB générée par le code ci-dessus (la variable d'état est associée à la LED - broche 25)

5.4 Mini serveur WEB avec une carte SD

Le développement d'un serveur plus volumineux comprenant plusieurs pages HTML et plusieurs formats de composants nécessite beaucoup plus de mémoire que **SPIFFS**. Dans ce cas, nous avons besoin d'une carte mémoire SD.

5.4.1 Un programme de test de la carte SD

Vous trouverez ci-dessous un code simple pour tester la carte SD. La carte SD doit être au format **FAT16**.

```
#include <mySD.h>
// pins for long ext card
#define MICROSD_PIN_CHIP_SELECT   2 //23
#define MICROSD_PIN_MOSI          27  //16
#define MICROSD_PIN_MISO          25  //17
#define MICROSD_PIN_SCK           5  //5
File root;

void setup()
{
  Serial.begin(9600);
  Serial.print("Initializing SD card...");
  /* initialize SD library with SPI pins */
  if (!SD.begin(MICROSD_PIN_CHIP_SELECT, MICROSD_PIN_MOSI, MICROSD_PIN_MISO,
MICROSD_PIN_SCK)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");
  root = SD.open("/");
  if (root) {
    printDirectory(root, 0);
    root.close();
  } else {
    Serial.println("error opening test.txt");
  }
  /* open "test.txt" for writing */
  root = SD.open("test.txt", FILE_WRITE);  // write test.tx file
  if (root) {
    root.println("Hello world!");
    root.flush();
    root.close();
  } else {
    Serial.println("error opening test.txt");
  }
  delay(1000);
  root = SD.open("test.txt");  // reopen and read
  if (root) {
    while (root.available()) {  // read bytes if available
      Serial.write(root.read());
    }
    root.close();
  } else {
    Serial.println("error opening test.txt");
  }
  Serial.println("done!");
}
void loop()
{ }
```

```
void printDirectory(File dir, int numTabs) {
  while(true) {
     File entry =  dir.openNextFile();
     if (! entry) {
       break;
     }
     for (uint8_t i=0; i<numTabs; i++) {
       Serial.print('\t');   // we'll have a nice indentation
     }
     // Print the name
     Serial.print(entry.name());
     /* Recurse for directories, otherwise print the file size */
     if (entry.isDirectory()) {
       Serial.println("/");
       printDirectory(entry, numTabs+1);
     } else {
       /* files have sizes, directories do not */
       Serial.print("\t\t");
       Serial.println(entry.size());
     }
     entry.close();
   }
}
```

5.3.2 Le programme mini-serveur WEB avec une carte SD

La carte SD doit être au format **FAT16**. La page Web racine est le dossier racine de la carte SD. Les extensions de fichier de plus de 3 caractères ne sont pas prises en charge par la bibliothèque SD (FAT16) . Les noms de fichiers de plus de **8 caractères** seront tronqués par la bibliothèque SD, donc gardez les noms de fichiers plus courts.
`index.htm` est l'index par défaut (fonctionne également sur les sous-dossiers)

```
#include <WiFiClient.h>
#include <ESP32WebServer.h>
#include <WiFi.h>
#include <ESPmDNS.h>
#include <SPI.h>
#include <mySD.h>

#define SD_CS             2 //23
#define SD_MOSI          17  //27  //16
#define SD_MISO          25  //17
#define SD_SCK           5  //13  //5

const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";


String serverIndex = "<script src='https://ajax.googleapis.com/ajax/libs/jquery/
3.2.1/jquery.min.js'></script>"
"<form method='POST' action='#' enctype='multipart/form-data' id='upload_form'>"
    "<input type='file' name='update'>"
    "<input type='submit' value='Upload'>"
"</form>"
"<div id='prg'>progress: 0%</div>"
"<script>"
"$('form').submit(function(e){"
    "e.preventDefault();"
      "var form = $('#upload_form')[0];"
```

```
      "var data = new FormData(form);"
      " $.ajax({"
            "url: '/update',"
            "type: 'POST',"
            "data: data,"
            "contentType: false,"
            "processData:false,"
            "xhr: function() {"
                "var xhr = new window.XMLHttpRequest();"
                "xhr.upload.addEventListener('progress', function(evt) {"
                    "if (evt.lengthComputable) {"
                        "var per = evt.loaded / evt.total;"
                        "$('#prg').html('progress: ' + Math.round(per*100) +
'%');"
                    "}"
               "}, false);"
               "return xhr;"
            "},"
            "success:function(d, s) {"
                "console.log('success!')"
           "},"
            "error: function (a, b, c) {"
            "}"
          "});"
"});"
"</script>";

ESP32WebServer server(80);
File root;
bool opened = false;

String printDirectory(File dir, int numTabs) {
  String response = "";
  dir.rewindDirectory();

  while(true) {
     File entry =  dir.openNextFile();
     if (! entry) {
       // no more files
       //Serial.println("**nomorefiles**");
       break;
     }
     for (uint8_t i=0; i<numTabs; i++) {
       Serial.print('\t');   // we'll have a nice indentation
     }
     // Recurse for directories, otherwise print the file size
     if (entry.isDirectory()) {
       printDirectory(entry, numTabs+1);
     } else {
       response += String("<a href='") + String(entry.name()) + String("'>") +
String(entry.name()) + String("</a>") + String("</br>");
     }
     entry.close();
   }
   return String("List files:</br>") + response + String("</br></br> Upload
file:") + serverIndex;
}
```

```
void handleRoot() {
  root = SD.open("/");
  String res = printDirectory(root, 0);
  server.send(200, "text/html", res);
}

bool loadFromSDCARD(String path){
  path.toLowerCase();
  String dataType = "text/html";  //"text/plain";
  if(path.endsWith("/")) path += "index.htm";

  if(path.endsWith(".src")) path = path.substring(0, path.lastIndexOf("."));
  else if(path.endsWith(".jpg")) dataType = "image/jpeg";
  else if(path.endsWith(".txt")) dataType = "text/plain";
  else if(path.endsWith(".zip")) dataType = "application/zip";
  else if(path.endsWith(".pdf")) dataType = "application/pdf";
  else if(path.endsWith(".png")) dataType = "application/png";
  Serial.println(dataType);
  File dataFile = SD.open(path.c_str());
  if (!dataFile)
    return false;
  if (server.streamFile(dataFile, dataType) != dataFile.size()) {
    Serial.println("Sent less data than expected!");
  }
  dataFile.close();
  return true;
}

void handleNotFound(){
  if(loadFromSDCARD(server.uri())) return;
  String message = "SDCARD Not Detected\n\n";
  message += "URI: ";
  message += server.uri();
  message += "\nMethod: ";
  message += (server.method() == HTTP_GET)?"GET":"POST";
  message += "\nArguments: ";
  message += server.args();
  message += "\n";
  for (uint8_t i=0; i<server.args(); i++){
    message += " NAME:"+server.argName(i) + "\n VALUE:" + server.arg(i) + "\n";
  }
  server.send(404, "text/plain", message);
  Serial.println(message);
}

void setup(void){
  Serial.begin(9600);
  pinMode(SD_MOSI,OUTPUT);pinMode(SD_CS,OUTPUT);
  pinMode(SD_SCK,OUTPUT);pinMode(SD_MISO,INPUT);
   Serial.print("Initializing SD card..."); delay(500);
  /* initialize SD library with SPI pins */
  if (!SD.begin(SD_CS, SD_MOSI, SD_MISO, SD_SCK)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  WiFi.begin(ssid, password);
```

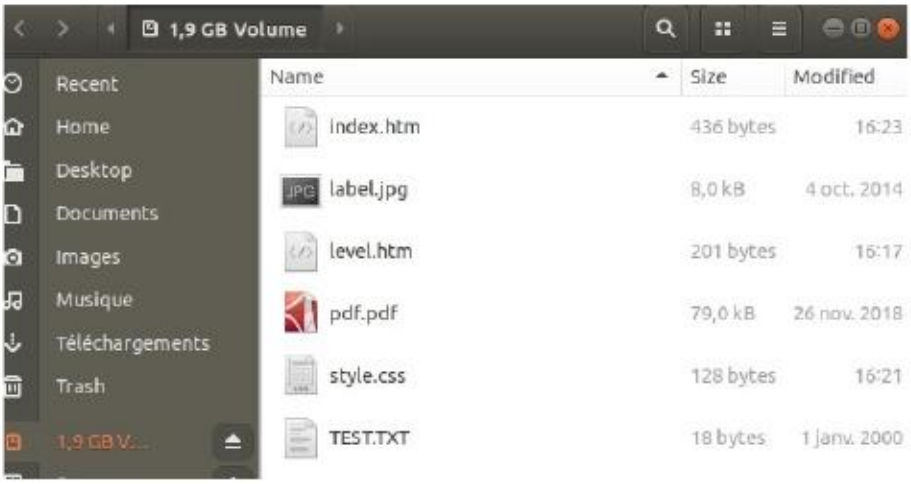
```
  Serial.println("");
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  //use IP or iotsharing.local to access webserver
  if (MDNS.begin("iotsharing")) {
    Serial.println("MDNS responder started");
  }
  //handle uri
  server.on("/", handleRoot);
  server.onNotFound(handleNotFound);
  /*handling uploading file */
  server.on("/update", HTTP_POST, [](){
    server.sendHeader("Connection", "close");
  },[](){
    HTTPUpload& upload = server.upload();
    if(opened == false){
      opened = true;
      root = SD.open((String("/") + upload.filename).c_str(), FILE_WRITE);
      if(!root){
        Serial.println("- failed to open file for writing");
        return;
      }
    }
    if(upload.status == UPLOAD_FILE_WRITE){
      if(root.write(upload.buf, upload.currentSize) != upload.currentSize){
        Serial.println("- failed to write");
        return;
      }
    } else if(upload.status == UPLOAD_FILE_END){
      root.close();
      Serial.println("UPLOAD_FILE_END");
      opened = false;
    }
  });
  server.begin();
  Serial.println("HTTP server started");
}

void loop(void){
  server.handleClient();
}
```

Notez que le serveur WEB ci-dessus ne sert **qu'un seul client à la fois**.

A faire :

1. SPIFFS: Construisez un serveur Web avec **SPIFFS** et **SoftAP**, utilisez les pages présentées dans l'exemple **5.2.3**
2. SD: Proposez vos propres pages WEB avec `.htm`, `.css` et ajoutez des images `.jpg` et `.pdf`.
3. SD: Créez un serveur Web sur **softAP**

Fichiers sur la carte SD (< **2GB** – FAT16) :



Fichier **index.htm**

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>SmartComputerLab</title>
<link rel="stylesheet" href="style.css">
</head>
<body>

<div style="text-align: center;"><b>This is test index.</b><br></div>
<hr>
<div style="text-align: center;"><a href="level.htm">How are you ?</a> <br>
<h1>image</h1>
<img src="label.jpg" width="200" height="62" alt=""><br>
<hr>
<a href="pdf.pdf">pdf</a></div><br>
<h2>end</h2>
</body>
</html>
```

Fichier **style.css**

```
body {
  background-color: lightblue;
}

h1 {
  color: navy;
  margin-left: 20px;
}

h2 {
  color: red;
  margin-left: 32px;
}
```

Laboratoire 6 - Programmation OTA (Over The Air)

6.1 Introduction

La **programmation OTA** permet la mise à jour - téléchargement d'un nouveau programme sur ESP32 **via Wi-Fi** au lieu de forcer l'utilisateur à connecter l'ESP32 à un ordinateur via USB pour effectuer la mise à jour. La fonctionnalité **OTA** est extrêmement utile s'il n'y a pas d'accès physique au module ESP. Cela permet de réduire le temps passé à mettre à jour chaque module ESP pendant la maintenance.
Une caractéristique importante de l'**OTA** est qu'un seul emplacement central peut envoyer une mise à jour à **plusieurs ESP** partageant le même réseau.
Le seul inconvénient est que vous devez ajouter du **code supplémentaire** pour **OTA** à chaque programme que vous téléchargez, afin de pouvoir utiliser OTA dans la prochaine mise à jour.

6.1.1 Mémoire flash de ESP32

Avant de passer à la programmation nous allons étudier les partitions de la mémoire flash d'un ESP32.
La mémoire flash est divisée en plusieurs partitions logiques pour stocker divers composants. La manière typique de procéder est illustrée dans la figure.

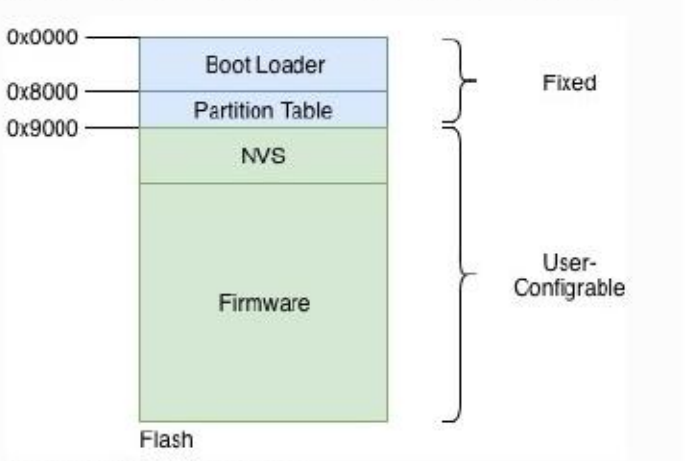


Figure 6.1 Partitions de la mémoire flash

Comme on peut le voir sur la figure dessus, la structure est statique jusqu'à l'adresse flash `0x9000`.
La première partie du flash contient le code du **boot**, qui est immédiatement suivi de la table de partition. La table de partition stocke ensuite la manière dont le reste du flash doit être interprété.
En règle générale, une installation aura au moins 1 partition **NVS** (**Non Volatile Storage – WiFi credentials** , ..) et 1 partition pour le code utilisateur

6.1.2 Mécanisme OTA

Pour les mises à niveau du code, un schéma de partition **actif-passif** est utilisé.

Deux partitions flash sont réservées au composant 'firmware', comme indiqué sur la figure suivante.
La partition **OTA-Data** se souvient laquelle de ces dernières est la partition active.

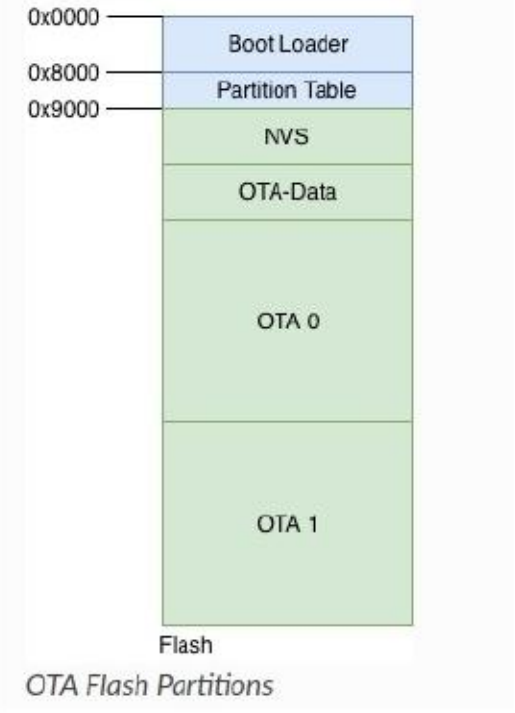


Figure 6.2 Partition **OTA-Data** et partitions **active-passive** pour le code

Les changements d'état typiques qui se produisent dans le flux de travail de mise à niveau du **code OTA** sont indiqués dans la **Figure 6.3.**

1. **Étape 0**: **OTA 0** est le **code actif**. La partition de données **OTA** stocke ces informations comme on peut le voir.
2. **Étape 1**: Le processus de **mise à niveau du code** commence. La partition passive est identifiée, effacée et un nouveau code est en cours d'écriture sur la partition **OTA 1**.
3. **Étape 2**: La mise à niveau du code est entièrement écrite et la vérification est en cours.
4. **Étape 3**: la mise à niveau du code est réussie, la partition de données OTA est mise à jour pour indiquer que **OTA 1** est désormais la **partition active**. Au prochain démarrage, le code de cette partition démarrera.

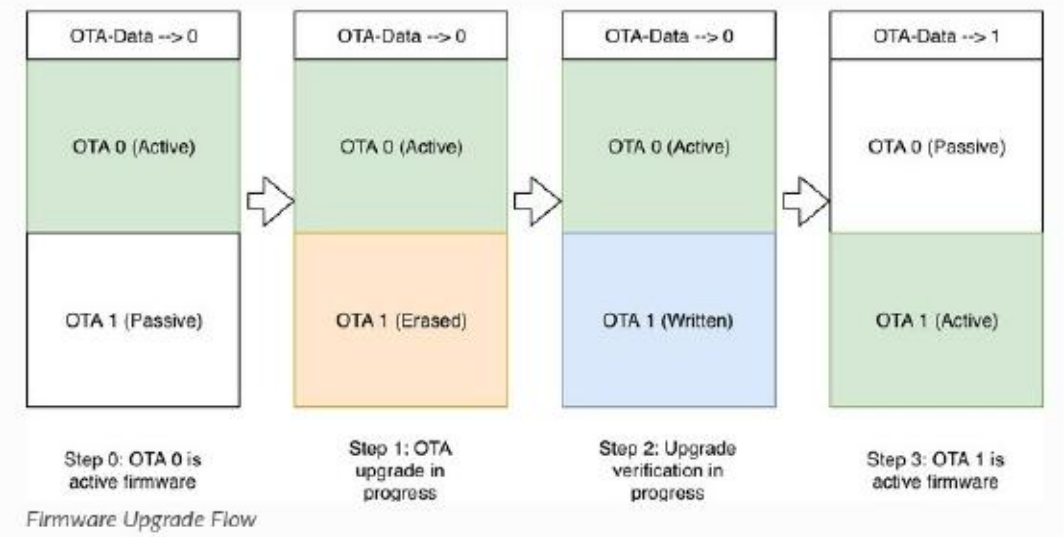


Figure 6.3 Flux de la mise à niveau du code dans la mémoire flash

6.2 Implémentation d'OTA sur carte ESP32 par OTA de base

Il existe **trois façons** d'implémenter la fonctionnalité OTA dans ESP32.

1. **OTA de base** - Les mises à jour Over-The-Air sont envoyées via **Arduino IDE** ou **PlatformIO**.
2. **Web Updater OTA** - Les mises à jour en direct sont envoyées via un **navigateur Web**.
3. La bibliothèque **WebOTA** permet également d'envoyer le croquis Arduino compilé (`.bin`) directement via l'**interface WEB**.

Chacun moyen a ses propres avantages. Vous pouvez les implémenter selon les besoins de votre projet.

6.2.1 La mise en œuvre de l'OTA de base

Pour commencer, téléchargez le micrologiciel OTA de base via un port série. Il s'agit d'une étape obligatoire pour pouvoir effectuer les prochaines mises à jour - téléchargements via WiFi.

Dans la phase suivante, vous pouvez télécharger de nouveaux programmes sur ESP32 à partir d'Arduino IDE via les airs - **WiFi avec une adresse IP**.

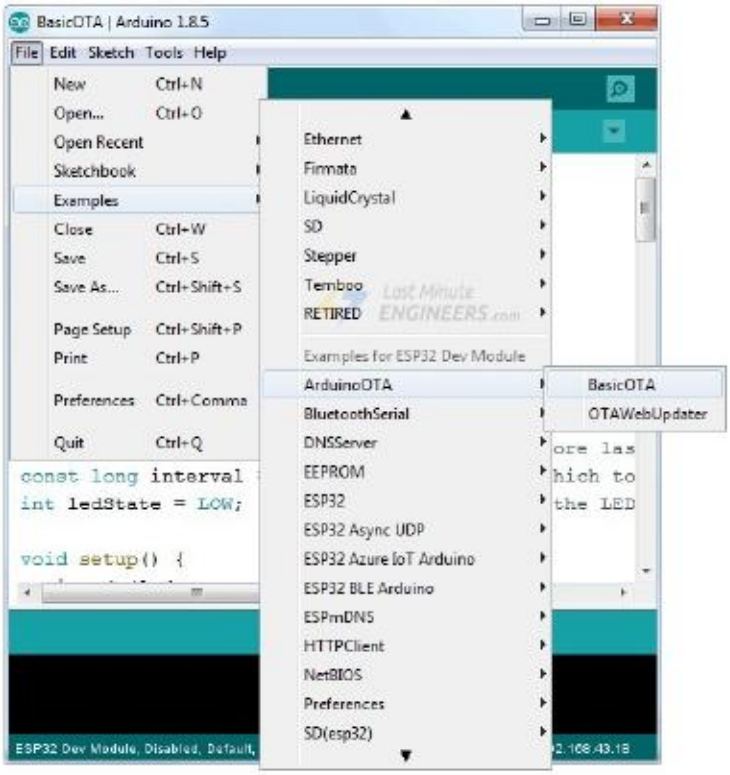


Figure 6.4 La sélection du code **OTA** de base

Avant de télécharger le code, vous devez apporter des modifications pour le rendre opérationnel. Vous devez modifier les deux variables suivantes avec vos informations d'identification réseau, afin qu'ESP32 puisse établir une connexion avec le réseau existant.

```
const char* ssid = ".........";
const char* password = ".........";
```

Une fois que vous avez terminé, téléchargez le code par **câble USB**.

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
const char* ssid = "PhoneAP";
const char* password = "smartcomputerlab";

void setup() {
  Serial.begin(9600);
  Serial.println("Booting");
  pinMode(led,OUTPUT);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
  }
  ArduinoOTA.onStart([]() {
      String type;
      if (ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
```

```
      else // U_SPIFFS
        type = "filesystem";
      Serial.println("Start updating " + type);
    })
    .onEnd([]() {
      Serial.println("\nEnd");
    })
    .onProgress([](unsigned int progress, unsigned int total) {
      Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    })
    .onError([](ota_error_t error) {
      Serial.printf("Error[%u]: ", error);
      if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
      else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
      else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
      else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
      else if (error == OTA_END_ERROR) Serial.println("End Failed");
    });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  ArduinoOTA.handle();
}
```

Maintenant, ouvrez le moniteur série à un débit de 9600 bauds. Si tout va bien, l'**adresse IP dynamique** obtenue de votre routeur sera affichée. **Écrivez le**.

6.2.2 Télécharger un nouveau code via WiFi

Maintenant, téléchargeons un nouveau programme.
Il est nécessaire d'**ajouter le code** pour **OTA** dans chaque croquis que vous téléchargez. Sinon, vous perdrez la capacité OTA et ne pourrez plus effectuer de futurs téléchargements en direct. Il est donc recommandé de modifier le code ci-dessus pour inclure votre nouveau code.

À titre d'exemple, nous allons inclure un simple sketch Blink dans le code OTA de base. N'oubliez pas de modifier les variables **`ssid`** et **`password`** avec vos informations d'identification réseau.

```
#include <WiFi.h>
#include <ESPmDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
const char* ssid = "PhoneAP";
const char* password = "smartcomputerlab";
//variables for blinking an LED with Millis
const int led = 25; // ESP32 Pin to which onboard LED is connected
unsigned long previousMillis = 0;  // will store last time LED was updated
const long interval = 1000;  // interval at which to blink (milliseconds)
int ledState = LOW;  // ledState used to set the LED

void setup() {
  Serial.begin(9600);
  Serial.println("Booting");
  pinMode(led,OUTPUT);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
```

```
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000); ESP.restart();
  }

  ArduinoOTA.onStart([]() {
      String type;
      if (ArduinoOTA.getCommand() == U_FLASH)
        type = "sketch";
      else // U_SPIFFS
        type = "filesystem";
      // NOTE: if updating SPIFFS this would be the place to unmount SPIFFS
using SPIFFS.end()
      Serial.println("Start updating " + type);
    })
    .onEnd([]() {
      Serial.println("\nEnd");
    })
    .onProgress([](unsigned int progress, unsigned int total) {
      Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
    })
    .onError([](ota_error_t error) {
      Serial.printf("Error[%u]: ", error);
      if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
      else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
      else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
      else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
      else if (error == OTA_END_ERROR) Serial.println("End Failed");
    });
  ArduinoOTA.begin();
  Serial.println("Ready");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}
void loop() {
  ArduinoOTA.handle();
  //loop to blink without delay
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
  // save the last time you blinked the LED
  previousMillis = currentMillis;
  // if the LED is off turn it on and vice-versa:
  ledState = not(ledState);
  // set the LED with the ledState of the variable:
  digitalWrite(led,  ledState);
  }
}
```

Remarque

Dans le programme ci-dessus, nous n'avons pas utilisé **`delay()`** pour faire clignoter la LED, car ESP32 met votre programme en pause pendant **`delay()`**. Si la prochaine demande OTA est générée alors qu'Arduino attend le délai d'expiration(), **votre programme manquera cette demande**.

Pour réaliser le delai nous avons utilisé le **timer** : **`currentMillis = millis()`**

Une fois que vous avez copié le croquis ci-dessus sur votre IDE Arduino, accédez à l'option Tools-> Port et vous devriez obtenir la sortie suivante: **`esp32-xxxxxx at esp_ip_address`**

Si vous ne le trouvez pas, vous devrez peut-être redémarrer votre IDE.

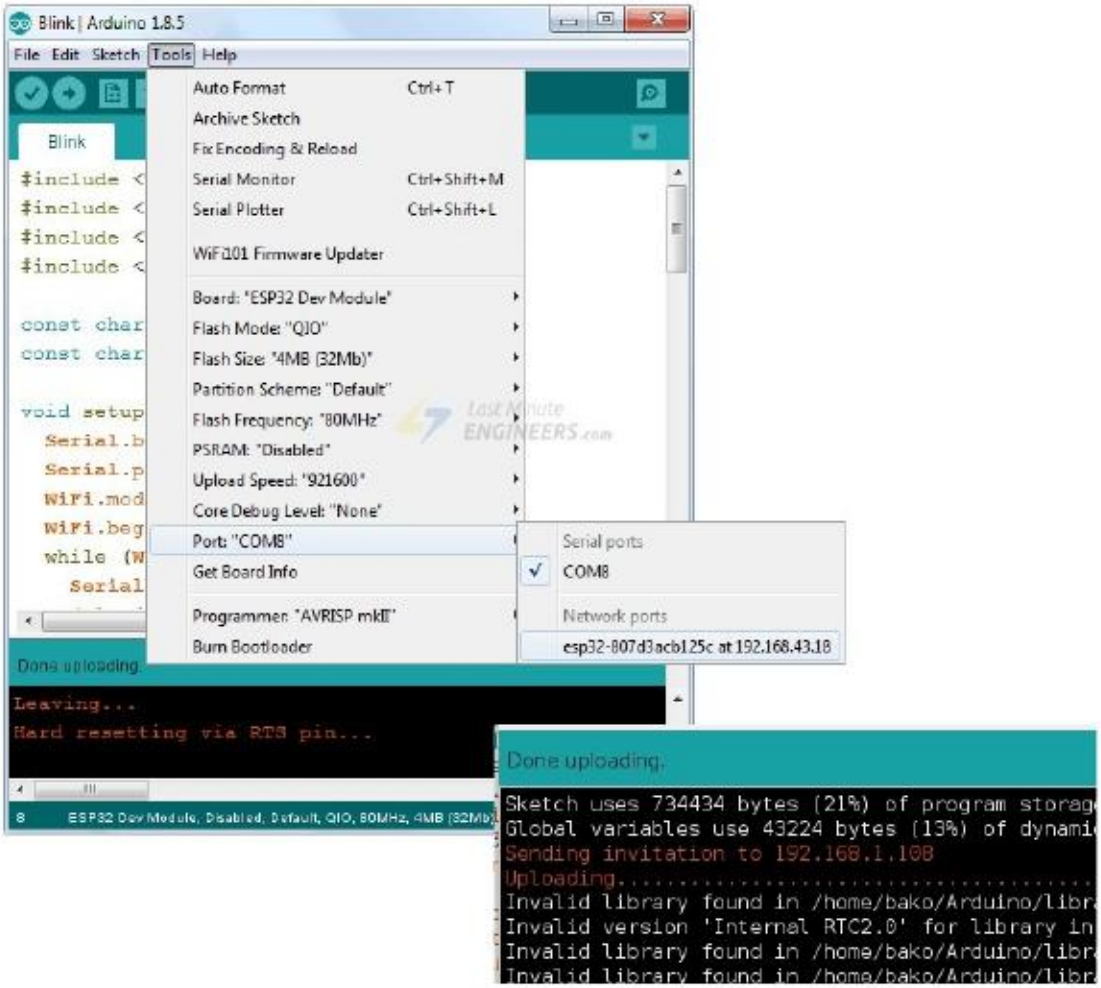


Figure 6.5 Sélection du port de transfert WiFi et de l'adresse IP

Sélectionnez le port et cliquez sur le bouton **`Download`**. Dans quelques secondes, le nouveau programme sera téléchargé. Et vous devriez voir la LED intégrée clignoter.

Pour vérifier le processus, vous pouvez modifier le programme en modifiant (par exemple) la valeur de:

```
int long const = 5000;
```

et téléchargez-le à nouveau via WiFi pour voir le résultat.

A faire :

1. Testez les programmes ci-dessus.
2. Modifiez le code en ajoutant un affichage sur l'écran OLED
3. Modifiez le code en ajoutant une lecture du capteur et un affichage sur l'écran OLED.

6.3 OTA sur carte ESP32 avec serveur WEB

Comme dans la solution précédente la première étape consiste à télécharger par USB le code contenant la routine OTA. Il s'agit d'une étape obligatoire pour pouvoir effectuer de futures mises à jour/téléchargements via WiFi.

Le nouveau programme OTA crée un **serveur Web** en mode **STA**, accessible via un navigateur Web. Une fois que vous êtes connecté au serveur Web, vous pouvez télécharger de nouveaux programmes avec la routine OTA.
Vous pouvez maintenant télécharger de nouveaux programmes sur l'ESP32 en générant et en téléchargeant le fichier **`.bin`** compilé dans l'environnement Arduino, via un **serveur Web**.

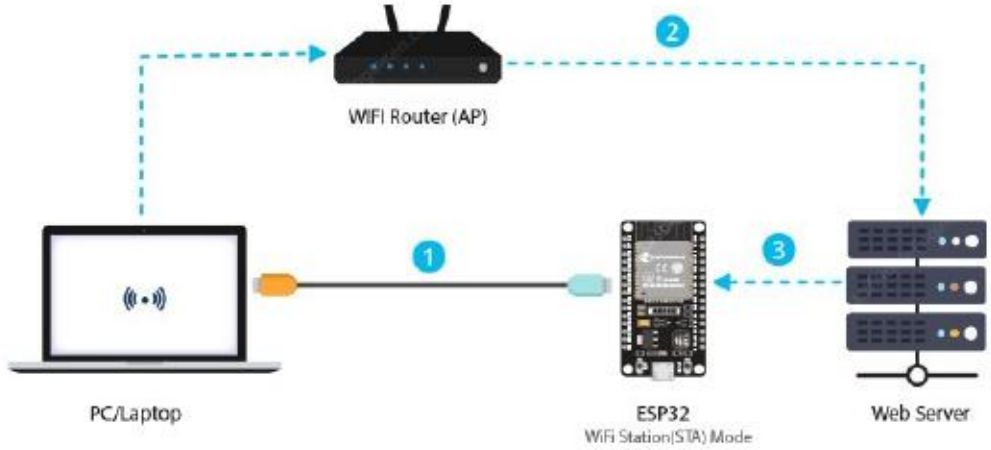


Figure 6.6 Transfert du code (fichier) binaire par le serveur WEB sur la carte ESP32

The ESP32 add-on for the Arduino IDE comes with an OTA library and an **`OTAWebUpdater`** example. You can access it via **`File> Examples> ArduinoOTA> OTAWebUpdater`** or via **`github.`**
Le **toolkit** d'ESP32 pour l'IDE Arduino est livré avec une bibliothèque OTA et un exemple **`OTAWebUpdater`**. Vous pouvez y accéder via **`File> Examples> ArduinoOTA> OTAWebUpdater`** ou via **`github`**.

Pour commencer, connectez votre ESP32 sur votre ordinateur et téléchargez le code ci-dessous.
Comme d'habitude vous devrait proposer les identificateurs WiFi de votre point d'accès. afin qu'ESP32 puisse établir une connexion avec le réseau existant.

6.3.1 Le programme de départ avec Webserver

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Update.h>

const char* host = "esp32";
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";

WebServer server(80);

// Login page

const char* loginIndex =
 "<form name='loginForm'>"
    "<table width='20%' bgcolor='A09F9F' align='center'>"
        "<tr>"
            "<td colspan=2>"
                "<center><font size=4><b>ESP32 Login Page</b></font></center>"
                "<br>"
            "</td>"
```

```
            "<br>"
            "<br>"
        "</tr>"
        "<td>Username:</td>"
        "<td><input type='text' size=25 name='userid'><br></td>"
        "</tr>"
        "<br>"
        "<br>"
        "<tr>"
            "<td>Password:</td>"
            "<td><input type='Password' size=25 name='pwd'><br></td>"
            "<br>"
            "<br>"
        "</tr>"
        "<tr>"
            "<td><input type='submit' onclick='check(this.form)'
value='Login'></td>"
        "</tr>"
    "</table>"
"</form>"
"<script>"
    "function check(form)"
    "{"
    "if(form.userid.value=='admin' && form.pwd.value=='admin')"
    "{"
    "window.open('/serverIndex')"
    "}"
    "else"
    "{"
    " alert('Error Password or Username')/*displays error message*/"
    "}"
    "}"
"</script>";

//Server Index Page
const char* serverIndex =
"<script
src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></
script>"
"<form method='POST' action='#' enctype='multipart/form-data' id='upload_form'>"
   "<input type='file' name='update'>"
        "<input type='submit' value='Update'>"
    "</form>"
 "<div id='prg'>progress: 0%</div>"
 "<script>"
  "$('form').submit(function(e){"
  "e.preventDefault();"
  "var form = $('#upload_form')[0];"
  "var data = new FormData(form);"
  " $.ajax({"
  "url: '/update',"
  "type: 'POST',"
  "data: data,"
  "contentType: false,"
  "processData:false,"
  "xhr: function() {"
  "var xhr = new window.XMLHttpRequest();"
  "xhr.upload.addEventListener('progress', function(evt) {"
```

```
  "if (evt.lengthComputable) {"
  "var per = evt.loaded / evt.total;"
  "$('#prg').html('progress: ' + Math.round(per*100) + '%');"
  "}"
  "}, false);"
  "return xhr;"
  "},"
  "success:function(d, s) {"
  "console.log('success!')"
 "},"
 "error: function (a, b, c) {"
 "}"
 "});"
 "});"
 "</script>";

void setup(void) {
  Serial.begin(9600);
  // Connect to WiFi network
  WiFi.begin(ssid, password);
  Serial.println("");
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  /*use mdns for host name resolution*/
  if (!MDNS.begin(host)) { //http://esp32.local
    Serial.println("Error setting up MDNS responder!");
    while (1) {
      delay(1000);
    }
  }
  Serial.println("mDNS responder started");
  /*return index page which is stored in serverIndex */
  server.on("/", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", loginIndex);
  });
  server.on("/serverIndex", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", serverIndex);
  });
  /*handling uploading firmware file */
  server.on("/update", HTTP_POST, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
    ESP.restart();
  }, []() {
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
      Serial.printf("Update: %s\n", upload.filename.c_str());
      if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
```

```
      Update.printError(Serial);
    }
  } else if (upload.status == UPLOAD_FILE_WRITE) {
    /* flashing firmware to ESP*/
    if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
      Update.printError(Serial);
    }
  } else if (upload.status == UPLOAD_FILE_END) {
    if (Update.end(true)) { //true to set the size to the current progress
      Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
    } else {
      Update.printError(Serial);
    }
  }
});
server.begin();
}

void loop(void) {
  server.handleClient(); delay(1);
}
```

6.3.2 Accéder au serveur Web

Le programme **`OTAWebUpdater`** crée un serveur Web en mode **STA** accessible via un navigateur Web et qui permet de télécharger de nouveaux programmes sur votre ESP32 via WiFi.
Pour accéder au serveur Web, ouvrez le moniteur série à une vitesse de transmission de 9600 bauds. Si tout va bien, l'**adresse IP dynamique** obtenue de votre routeur (carte) sera affichée.

Ensuite, chargez un navigateur et pointez-le vers l'adresse IP indiquée sur le moniteur série. L'ESP32 doit servir de page Web demandant les informations de connexion (par défaut : **`admin`** et **`admin`**).

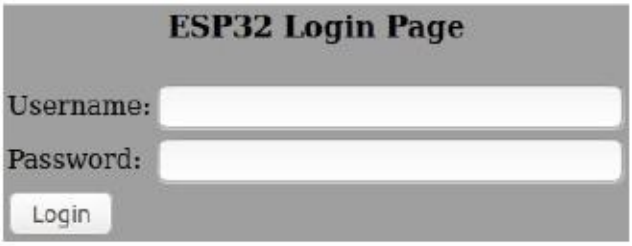


Figure 6.7 Page initiale du serveur WEB sur la carte ESP32

Si vous souhaitez modifier l'ID utilisateur et le mot de passe, modifiez le code ci-dessous dans votre programme.

```
"if (form.userid.value == 'admin' && form.pwd.value == 'admin')"
```

Une fois connecté au serveur, vous serez redirigé vers la page **`/serverIndex`**.

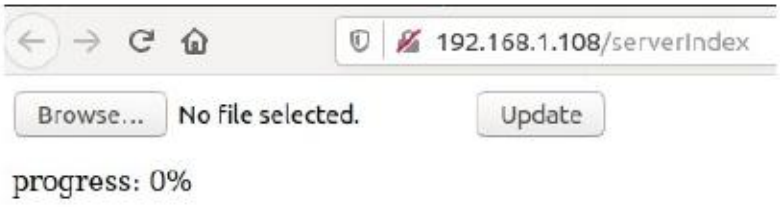


Figure 6.8 Page de recherche et de chargement d'un fichier **`.bin`**

Cette page vous permet de télécharger de nouveaux programmes sur votre ESP32 via WiFi. Ce nouveau programme en cours de téléchargement doit être au format **binaire** **`.bin`** .

6.3.3 Téléchargez un nouveau programme via WiFi (.bin)

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Update.h>
const char* host = "esp32";
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";
//variabls for blinking an LED with Millis
const int led = 25; // ESP32 Pin to which onboard LED is connected
unsigned long previousMillis = 0;  // will store last time LED was updated
const long interval = 1000;  // interval at which to blink (milliseconds)
int ledState = LOW;  // ledState used to set the LED
WebServer server(80);
/* Style */
String style =
"<style>#file-input,input{width:100%;height:44px;border-radius:4px;margin:10px
auto;font-size:15px}"
"input{background:#f1f1f1;border:0;padding:0 15px}body{background:#3498db;font-
family:sans-serif;font-size:14px;color:#777}"
"#file-input{padding:0;border:1px solid #ddd;line-height:44px;text-
align:left;display:block;cursor:pointer}"
"#bar,#prgbar{background-color:#f1f1f1;border-radius:10px}#bar{background-
color:#3498db;width:0%;height:10px}"
"form{background:#fff;max-width:258px;margin:75px auto;padding:30px;border-
radius:5px;text-align:center}"
".btn{background:#3498db;color:#fff;cursor:pointer}</style>";
/* Login page */
String loginIndex =
"<form name=loginForm>"
"<h1>ESP32 Login</h1>"
"<input name=userid placeholder='User ID'> "
"<input name=pwd placeholder=Password type=Password> "
"<input type=submit onclick=check(this.form) class=btn value=Login></form>"
"<script>"
"function check(form) {"
"if(form.userid.value=='admin' && form.pwd.value=='admin')"
"{window.open('/serverIndex')}"
"else"
"{alert('Error Password or Username')}"
"}"
"</script>" + style;

String serverIndex =
"<script
src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'></
script>"
"<form method='POST' action='#' enctype='multipart/form-data' id='upload_form'>"
"<input type='file' name='update' id='file' onchange='sub(this)'
style=display:none>"
"<label id='file-input' for='file'>   Choose file...</label>"
"<input type='submit' class=btn value='Update'>"
"<br><br>"
"<div id='prg'></div>"
"<br><div id='prgbar'><div id='bar'></div></div><br></form>"
"<script>"
"function sub(obj){"
```

```
"var fileName = obj.value.split('\\\\');"
"document.getElementById('file-input').innerHTML = '   '+
fileName[fileName.length-1];"
"};"
"$('form').submit(function(e){"
"e.preventDefault();"
"var form = $('#upload_form')[0];"
"var data = new FormData(form);"
"$.ajax({"
"url: '/update',"
"type: 'POST',"
"data: data,"
"contentType: false,"
"processData:false,"
"xhr: function() {"
"var xhr = new window.XMLHttpRequest();"
"xhr.upload.addEventListener('progress', function(evt) {"
"if (evt.lengthComputable) {"
"var per = evt.loaded / evt.total;"
"$('#prg').html('progress: ' + Math.round(per*100) + '%');"
"$('#bar').css('width',Math.round(per*100) + '%');"
"}"
"}, false);"
"return xhr;"
"},"
"success:function(d, s) {"
"console.log('success!') "
"},"
"error: function (a, b, c) {"
"}"
"});"
"});"
"</script>" + style;

void setup(void) {
  Serial.begin(9600);pinMode(led,OUTPUT);
  WiFi.begin(ssid, password);Serial.println("");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
  }
  Serial.println("");Serial.print("Connected to ");
  Serial.println(ssid);Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  /*use mdns for host name resolution*/
  if (!MDNS.begin(host)) { //http://esp32.local
    Serial.println("Error setting up MDNS responder!");
    while (1) {  delay(1000);}
  }
  Serial.println("mDNS responder started");
  /*return index page which is stored in serverIndex */
  server.on("/", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", loginIndex);
  });
  server.on("/serverIndex", HTTP_GET, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", serverIndex);
  });
```

```
  /*handling uploading firmware file */
  server.on("/update", HTTP_POST, []() {
    server.sendHeader("Connection", "close");
    server.send(200, "text/plain", (Update.hasError()) ? "FAIL" : "OK");
    ESP.restart();
  }, []() {
    HTTPUpload& upload = server.upload();
    if (upload.status == UPLOAD_FILE_START) {
      Serial.printf("Update: %s\n", upload.filename.c_str());
      if (!Update.begin(UPDATE_SIZE_UNKNOWN)) { //start with max available size
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_WRITE) {
      /* flashing firmware to ESP*/
      if (Update.write(upload.buf, upload.currentSize) != upload.currentSize) {
        Update.printError(Serial);
      }
    } else if (upload.status == UPLOAD_FILE_END) {
      if (Update.end(true)) { //true to set the size to the current progress
        Serial.printf("Update Success: %u\nRebooting...\n", upload.totalSize);
      } else {
        Update.printError(Serial);
      }
    }
  });
  server.begin();
}

void loop(void) {
  server.handleClient(); delay(1);
 unsigned long currentMillis = millis();
 if (currentMillis - previousMillis >= interval) {
 previousMillis = currentMillis;
 ledState = not(ledState);
 digitalWrite(led,  ledState);
 }
}
```

6.3.4 Générez un fichier `.bin` dans l'IDE Arduino

Afin de télécharger un nouveau croquis sur l'ESP32, nous devons d'abord générer le fichier binaire `.bin` compilé de votre programme. Pour ce faire, sélectionnez **`Sketch> Export compiled Binary`**

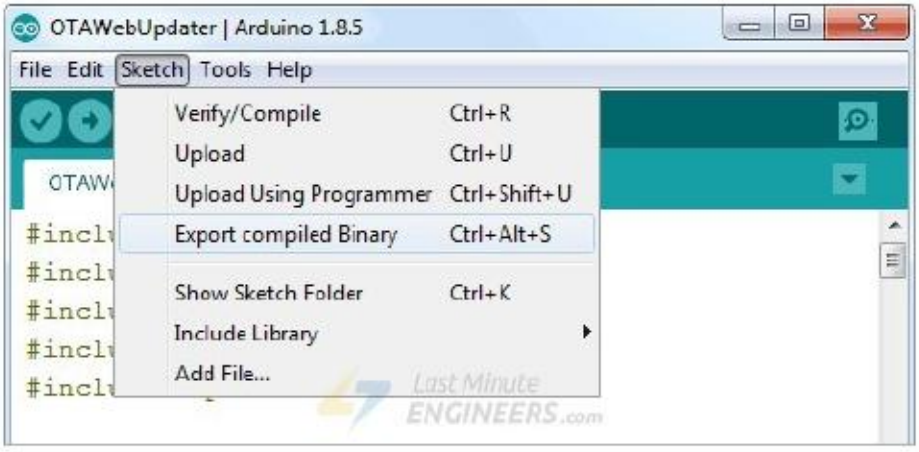


Figure 6.9 Génération du code binaire pour transfert sur la carte ESP32

6.3.5 Download a new sketch live on the ESP32

Une fois le fichier `.bin` généré, vous êtes maintenant prêt à télécharger le nouveau code sur l'ESP32 via WiFi. Ouvrez la page `/serverIndex` dans votre navigateur. Cliquez sur Choisir un fichier... Sélectionnez le fichier `.bin` généré (dans le même répertoire que votre sketch `.ino`) , puis cliquez sur **Update**.

Figure 6.10 Transfert du code binaire sur la carte ESP32

Dans quelques secondes, le nouveau programme sera téléchargé. Et vous devriez voir la LED intégrée clignoter.

A faire

1. **Testez** les programmes ci-dessus.
2. **Modifiez** le code en ajoutant un affichage sur l'écran OLED
3. **Modifiez** le code en ajoutant une lecture du capteur et un affichage sur l'écran OLED.

6.4 Implémentation d'OTA avec la bibliothèque `WebOTA`

La solution précédente nécessite l'insertion du **code HTML** de la page WEB créée pour le transfert du code `.bin` vers la carte ESP32. La bibliothèque **WebOTA** promet de simplifier cette préparation.
Vous aurez besoin d'une inclusion, bien sûr. Si cela ne vous dérange pas d'utiliser le port `8080` et le chemin par défaut `/webota`, appelez simplement `handle_webota`) depuis votre boucle principale (`loop()`).
Si vous souhaitez modifier les valeurs par défaut, vous devez ajouter un appel supplémentaire dans votre configuration. Vous devez également configurer quelques variables globales pour spécifier vos paramètres réseau.
Le seul inconvénient est que les déclarations avec un long délai (`delay()`) dans votre boucle peuvent empêcher certaines choses de fonctionner correctement et ne sont pas une bonne idée.
Si vous en avez, vous pouvez remplacer tous vos appels `delay()` par `webota_delay()`, ce qui empêchera le système d'ignorer les demandes de mise à jour.
Le code initial doit être chargé par l'environnement Arduino. Pour effectuer une mise à jour, accédez simplement à la carte ESP32 avec un navigateur Web et utilisez le numéro de port et le chemin corrects. De là, vous pouvez télécharger une nouvelle image binaire créée dans l'IDE Arduino.

Remarque :
Le code à charger n'est pas authentifié. Cela signifie que n'importe qui peut télécharger du code sur votre ESP. Cela peut être acceptable sur un réseau privé, mais sur Internet, c'est certainement problématique.

6.4.1 Code initial

```
#include <WebOTA.h>
#define LED_PIN 25
const char* host     = "ESP-OTA"; // Used for MDNS resolution
const char* ssid = "Livebox-08B0";
const char* password = "G79ji6dtEptVTPWmZP";

void setup() {
  Serial.begin(9600);
  pinMode(LED_PIN, OUTPUT);
  init_wifi(ssid, password, host);
  // Defaults to 8080 and "/webota" , webota.init(80, "/update");
}

void loop() {
  int md = 5000;
  digitalWrite(LED_PIN, HIGH);
  webota.delay(md);
  digitalWrite(LED_PIN, LOW);
  webota.delay(md);
  webota.handle();
}
```

6.4.2 Chargement initial et final

Dans la phase de chargement initial, nous obtenons l'adresse **IP** du serveur WEB sur la carte.

```
WebOTA url   : http://ESP-OTA.local:8080/webota

Firmware update initiated: esp32.HT.WebOTA.init.201119.ino.heltec_wifi_lora_32_V2.bin
50k 100k 150k 200k 250k 300k 350k 400k 450k 500k 550k 600k 650k 700k
Firmware update successful: 759856 bytes
Rebooting...
        ¶¶¶¶1¶X j$I¶\∩  ¶
Connecting to Wifi.
Connected to 'Livebox-08B0'

IP address   : 192.168.1.108
MAC address  : 24:6F:28:96:EF:48
mDNS started : ESP-OTA.local
WebOTA url   : http://ESP-OTA.local:8080/webota
```

Figure 6.11 Affichage des paramètres: IP et URL du serveur WEB

L'affichage se fait jusqu'à la ligne avec l'url **WebOTA**. Le serveur est accessible avec ;
`192.168.1.108:8080/webota`

Il vous demande de fournir l'emplacement de votre code Arduino compilé en **`.bin`**.

`esp32.HT.WebOTA.init.201119.ino.heltec_wifi_lora32_V2.bin`

Figure 6.12 La page initiale de chargement du code binaire depuis le serveur WEB

A faire :

1. Testez l'exemple ci-dessus.
2. Modifiez le code en ajoutant un affichage sur l'écran OLED
3. Modifiez le code en ajoutant une lecture du capteur et un affichage sur l'écran OLED.

Table of Contents