

# TP2

Gilles Menez - UNS - UFR Sciences - Dépt. Informatique

24 janvier 2022

## Objectifs pédagogiques

- ✓ On se branche sur le réseau !
- ✓ On décortique l'utilisation du protocole HTTP

Le travail demandé consiste à "analyser" et "faire tourner".

Les sections intitulées "TODO : Votre travail !" nécessitent une réalisation **personnelle**.

## Table des matières

<b>1</b>	<b>JSON</b>	<b>3</b>
<b>2</b>	<b>Lien série (USB) avec l'hôte</b>	<b>4</b>
2.1	L'ESP écrit, l'hôte lit . . . . .	4
2.1.1	Plotting with Python . . . . .	6
2.2	L'hôte écrit, l'ESP lit . . . . .	6
2.3	TODO : Votre travail! . . . . .	7
2.3.1	JSON at work . . . . .	7
2.3.2	Au niveau du PC . . . . .	7
2.4	Contributions intéressantes . . . . .	7
2.5	by bricoleau . . . . .	7
2.6	by jandrassy . . . . .	7
<b>3</b>	<b>Connectivité WiFi</b>	<b>8</b>
3.1	Connexion basique . . . . .	8
3.2	Few words on WiFi and ESP . . . . .	11
3.3	Recherche des SSIDs environnants . . . . .	12
3.4	Connexion parmi une liste . . . . .	14
<b>4</b>	<b>Sockets</b>	<b>16</b>
<b>5</b>	<b>HTTP</b>	<b>17</b>
5.1	Bibliothèque <WiFi> . . . . .	17
5.2	L'ESP est un client (GET) HTTP . . . . .	17
5.2.1	Le serveur "httpbin.org" . . . . .	18
5.2.2	Même requête ... en utilisant "WiFiClient" . . . . .	19
5.2.3	Quelques URL! . . . . .	21

5.2.4	Requête ... en utilisant <HTTPClient> . . . . .	23
5.3	L'ESP est un serveur (GET) HTTP . . . . .	25
5.3.1	Version simpliste avec <WiFi.h> . . . . .	25
5.4	ESPAsyncWebServer . . . . .	27
5.4.1	Installation . . . . .	27
5.4.2	Pour faire quoi ? . . . . .	27
5.5	La partie Html . . . . .	27
5.5.1	Le Script . . . . .	30
5.5.2	La partie "Periodic report to" . . . . .	31
5.6	La partie ESP . . . . .	31
5.6.1	Requête/Route/Handler . . . . .	31
<b>6</b>	<b>Node-RED</b> . . . . .	<b>33</b>
6.1	Installation . . . . .	34
6.2	Execution . . . . .	34
6.3	Un exemple ... . . . .	35
6.4	Notion de Dashboard . . . . .	41
<b>7</b>	<b>cURL</b> . . . . .	<b>44</b>
7.1	Référence . . . . .	44
7.2	Structure générale d'une commande cURL . . . . .	44
7.3	HTTP : Voir le contenu de l'URL . . . . .	44
7.4	HTTP : request / response pour obtenir cette URL . . . . .	44
7.5	HTTP : requête GET avec paramètres . . . . .	45
7.6	HTTP : requête POST sans paramètre . . . . .	46
7.7	HTTP : requête POST avec paramètres et body en JSON . . . . .	46
7.8	HTTP : requête POST avec paramètres et body en "URL encoded" . . . . .	47
<b>8</b>	<b>TODO : Votre travail !</b> . . . . .	<b>48</b>
8.1	Montée en compétences . . . . .	48
8.2	Cahier des charges . . . . .	48
8.2.1	L'ESP . . . . .	48
8.2.2	Dashboard . . . . .	50
8.3	ANNEXE 1 : POST . . . . .	51
8.3.1	Quelques liens pour aider ... . . . .	51
8.3.2	Exemple de POST en HTML . . . . .	51

Dans les exercices qui vont suivre, l'idée est "d'extraire" les valeurs récupérées par l'Objet (ESP32).

- Soit, c'est "nous" qui allons les chercher.
- Soit, c'est lui qui "push" ses données.

## 1 JSON

Naturellement, à partir du moment où on échange des données, se pose le problème du format et de la structuration de ces données.

- JSON (JavaScript Object Notation) est une solution possible.

Dans le cours, on parle de JSON notamment dans le contexte d'un dialogue entre objets de l'IoT.

- C'est le moment d'utiliser JSON pour ce qui va suivre !

## 2 Lien série (USB) avec l'hôte

Nous commençons par utiliser la liaison série (USB) qui relie la carte ESP à son hôte (le PC).

Côté ESP, les primitives utilisées et d'autres sont décrites :

<https://www.arduino.cc/reference/en/language/functions/communication/serial/>

### 2.1 L'ESP écrit, l'hôte lit

Sur l'ESP vous lancez le sketch du "photoresistor" : l'ESP écrit sa valeur sur la ligne série (via un "Serial.println").

Sur le PC, vous lancez ce script Python qui lit sur le port série et affiche (dans la console où il est lancé) les données envoyées (par la fonction `Serial.println` du sketch ESP).

#### RMQs :

- ✓ Vous aurez sans doute besoin d'installer le package "serial" sous Python/Linux :

```
$ pip3 install pyserial
```

- ✓ La documentation :

[https://pyserial.readthedocs.io/en/latest/pyserial\\_api.html](https://pyserial.readthedocs.io/en/latest/pyserial_api.html)

- ✓ Il n'est pas possible d'avoir la console de l'IDE ouverte pendant que vous lisez la ligne série avec Python ... problème de partage de ressource (le port USB).

```

1  #Fichier readserial.py
2  import time
3  import serial
4
5  ser = serial.Serial(
6      port='/dev/ttyUSB0',
7      baudrate = 9600,
8      parity=serial.PARITY_NONE,
9      stopbits=serial.STOPBITS_ONE,
10     bytesize=serial.EIGHTBITS,
11     timeout=1 #https://pythonhosted.org/pyserial/shortintro.html#readline
12 )
13
14 while True:
15     try:
16         x = ser.readline() # read a '\n' terminated line
17         x = x.rstrip()
18         x = x.decode("utf-8")
19
20     print ("Valeur_{}_{}".format(x))
21

```

```

22 except KeyboardInterrupt:
23     print('exiting')
24     break
25
26 #On ecrit
27 #ser.write('1') ;ser.flush()
28
29 # close serial
30 ser.flush()
31 ser.close()

```

Cela donne quelque chose comme ça :

```

photoresistor
/* photoresistor/photoresistor.ino */

void setup(){
  Serial.begin(9600); // starts the serial port at 9600
}

void loop(){
  int sensorValue;
  sensorValue = analogRead(A5);    // Read analog input on ADC1_CHANNEL_5 (GPIO 33)
                                   // Pin "D33"
  Serial.println(sensorValue, DEC); // Prints the value to the serial port
                                   // as human-readable ASCII text
  delay(1000); // wait 1s for next reading
}

Writing at 0x00018000... (42 %)
Writing at 0x0001c000... (57 %)
Writing at 0x00020000... (71 %)
Writing at 0x00024000... (85 %)
Writing at 0x00028000... (100 %)
Wrote 214736 bytes (109604 compressed) at 0x00010000 in 1.5 seconds (effective 1177.1 kbps)
Hash of data verified.
Compressed 3072 bytes to 128...

1 ESP32 Dev Module sur /dev/ttyUSB0

menez@mowgli ~/EnseignementsCurrent/Cours_IoT/TPs/TP2/Latex
$ python3 ../Src_Python/readserial.py
Valeur : 1204
Valeur : 1220
Valeur : 1225
Valeur : 1222
Valeur : 1223
Valeur : 1218
Valeur : 1222
Valeur : 1220
Valeur : 1221
Valeur : 1224
Valeur : 1225
Valeur : 1222
Valeur : 1218
Valeur : 1220
Valeur : 1218
Valeur : 1218

```

### 2.1.1 Plotting with Python

Essayez le programme Python "fromserial.py" pour une expérience plus "graphique".

On va s'en servir un peu plus loin ...

## 2.2 L'hôte écrit, l'ESP lit

L'ESP peut aussi utiliser la liaison série pour récupérer des caractères :

<https://www.arduino.cc/en/serial/read>

Ce coup-ci, ce code doit être essayé avec la console de l'IDE Arduino :

```

1 #define TRUE 1
2 #define FALSE 0
3 char receivedChar;
4 String receivedStr; // souvent evite .. perf !?
5 int sensorValue;
6 int jour = FALSE;
7 float val = 0.0;
8
9 void setup() {
10   Serial.begin(9600);
11   Serial.println("<Serial□is□ready>");
12 }
13
14 void loop() {
15   // L'ESP ecrit sur le lien serial
16   sensorValue = analogRead(A5); // read analog input : light
17   Serial.print(sensorValue, DEC); // Prints the value to the serial port
18   Serial.print("\n"); // as human-readable ASCII text
19
20   // L'ESP lit
21   while(Serial.available() > 0) {
22     /* Read from serial —————*/
23     //receivedChar = Serial.read(); // Just one byte
24     receivedStr = Serial.readStringUntil('\n'); // A string
25     Serial.print("\nI□received□:□"); // say what you got:
26     //Serial.println(receivedChar);
27     Serial.println(receivedStr);
28
29     /* Action ? —————*/
30     //if (receivedChar == '1') jour = TRUE ; else jour = FALSE;
31
32     val = receivedStr.toFloat();
33     Serial.println(val);
34   }
35   delay(1000);
36 }

```

## 2.3 TODO : Votre travail !

Lorsque tous les capteurs/actionneurs/leds sont branchés,

### 2.3.1 JSON at work

Vous réfléchissez à une structuration JSON du message envoyé par l'ESP et qui donne son statut/état.

Je verrais bien un format comme cela :

```

1  {
2    "temperature": "21.6",
3    "light": "850",
4    "led1": "On",
5    "led2": "Off"
6  }
```

Ceci étant il manque les unités, les couleurs des Leds, ... à vous de voir !

Vous implémentez votre structuration et cela doit se voir dans la console de l'IDE Arduino.

### 2.3.2 Au niveau du PC

- ① Vous réalisez l'analyse du message JSON pour alimenter le graphisme construit par le programme python "fromserial.py".
  - Il y a un travail d'adaptation du script python car on veut voir les courbes de luminosité et de température.
- ② Vous implémentez une fonction d'arrêt de la régulation :

start / stop

sont les mots clés qui si ils sont tapés dans la console Python arrêtent ou démarrent la régulation.

- Du coup, les LEDs ne changent plus et l'évolution du graphisme Python est gelé .

Vous pouvez faire évoluer la structure JSON si besoin. J'espère trouver du JSON dans les deux sens de communication !

## 2.4 Contributions intéressantes

### 2.5 by bricoleau

Sympa cette idée d'avoir une console ESP dans une page Web !

<https://forum.arduino.cc/t/remote-terminal-monitor-for-esp8266-esp32/643879>

### 2.6 by jandrassy

Plus classique mais tout aussi utile ... telnet !

<https://github.com/jandrassy/TelnetStream>

### 3 Connectivité WiFi

Pour l'exploiter l'interface WiFi de l'ESP, nous utilisons l'API développée par Espressif et **include** dans l'IDE Arduino.

- <https://www.arduino.cc/en/Reference/WiFi>
- <https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi/examples>

#### 3.1 Connexion basique

Ce premier sketch illustre comment l'ESP peut se connecter à un Access Point (AP) d'un SSID qu'il **connaît statiquement** :

- l'ESP est "client" du réseau qu'il va utiliser pour communiquer.

On utilise la bibliothèque WiFi de l'arduino :

<https://www.arduino.cc/en/Reference/WiFi>

dont on se doute qu'elle "wrap" les primitives ESP-IDF :

<https://github.com/espressif/arduino-esp32/tree/master/libraries>

```

1  /** Basic Wifi connection: wificonnect.ino */
2  #include <SPI.h>
3  #include <WiFi.h> // https://www.arduino.cc/en/Reference/WiFi
4
5  #define SaveDisconnectTime 1000 // Time in ms for save disconnection, needed
   to avoid that WiFi works only every second boot: https://github.com/
   espressif/arduino-esp32/issues/2501
6
7  /* Credentials */
8  const char ssid [] = "HUAWEI-553A";
9  const char password [] = "QTM06RTT";
10
11 /*-----*/
12 String translateEncryptionType(wifi_auth_mode_t encryptionType) {
13     // cf https://www.arduino.cc/en/Reference/WiFiEncryptionType
14     switch (encryptionType) {
15         case (WIFI_AUTH_OPEN):
16             return "Open";
17         case (WIFI_AUTH_WEP):
18             return "WEP";
19         case (WIFI_AUTH_WPA_PSK):
20             return "WPA_PSK";
21         case (WIFI_AUTH_WPA2_PSK):
22             return "WPA2_PSK";
23         case (WIFI_AUTH_WPA_WPA2_PSK):
24             return "WPA_WPA2_PSK";
25         case (WIFI_AUTH_WPA2_ENTERPRISE):
26             return "WPA2_ENTERPRISE";
27     }
28 }

```



```

29 /*-----*/
30 void print_network_status_light() { // array of chars
31     char s[256];
32     sprintf(s, "\tIP_address:_%s\n", WiFi.localIP().toString().c_str()); Serial.
        print(s);
33     sprintf(s, "\tMAC_address:_%s\n", WiFi.macAddress().c_str()); Serial.print(s
        );
34     sprintf(s, "\tWifi_SSID:_%s\n", WiFi.SSID()); Serial.print(s);
35     sprintf(s, "\tWifi_Signal_Strength:_%ld_dBm\n", WiFi.RSSI()); Serial.print(s)
        ;
36     sprintf(s, "\tWifi_Signal_Strength:_%ld_%\n", constrain(2 * (WiFi.RSSI() +
        100), 0, 100)); Serial.print(s);
37     sprintf(s, "\tWifi_BSSID:_%s\n", WiFi.BSSIDstr().c_str()); Serial.print(s);
38     sprintf(s, "\tWifi_Encryption_type:_%s\n", translateEncryptionType(WiFi.
        encryptionType(0))); Serial.print(s);
39 }
40 void print_network_status() { // Utilisation de String !
41     String s = "";
42     s += "\tIP_address:_ " + WiFi.localIP().toString() + "\n";
43     s += "\tMAC_address:_ " + String(WiFi.macAddress()) + "\n";
44     s += "\tWifi_SSID:_ " + String(WiFi.SSID()) + "\n";
45     s += "\tWifi_Signal_Strength:_ " + String(WiFi.RSSI()) + "_dBm\n";
46     s += "\tWifi_Signal_Strength:_ " + String(constrain(2 * (WiFi.RSSI() + 100),
        0, 100)) + "_%\n";
47     s += "\tWifi_BSSID:_ " + String(WiFi.BSSIDstr()) + "\n";
48     s += "\tWifi_Encryption_type:_ " + translateEncryptionType(WiFi.
        encryptionType(0)) + "\n";
49     Serial.print(s);
50 }
51 /*-----*/
52 void connect_wifi() {
53     #define WiFiMaxTry 10
54     int i;
55     String hostname = "Mon_petit_objet_ESP32";
56
57     WiFi.mode(WIFI_OFF);
58     // Set WiFi to station mode
59     WiFi.mode(WIFI_STA);
60     // WiFi.config(INADDR_NONE, INADDR_NONE, INADDR_NONE, INADDR_NONE);
61     // Disconnect from an AP if it was previously connected
62     WiFi.disconnect(true); // delete old config
63     // WiFi.persistent(false); // Avoid to store Wifi configuration in Flash
64     delay(100); // ms
65
66     // Define hostname before begin
67     WiFi.setHostname(hostname.c_str());
68
69     Serial.println(String("\nAttempting to connect AP of SSID:_ ") + ssid);
70     WiFi.begin(ssid, password);

```

```

71  i = 0;
72  while(WiFi.status() != WL_CONNECTED && (i < WiFiMaxTry)){
73      delay(SaveDisconnectTime); // 500ms seems to work in most cases , may depend
        on AP
74      Serial.print(".");
75      i++;
76  }
77 }
78
79 /*—— Arduino IDE paradigm : setup+loop ——*/
80 void setup(){
81     Serial.begin(9600);
82     while(!Serial); //wait for a serial connection
83
84     connect_wifi();          // Connect wifi
85
86     if (WiFi.status() == WL_CONNECTED){
87         Serial.print("\nWiFi_connected:_:_\n");
88         print_network_status(); // Print status
89     }
90     // else
91     //   ESP.restart();
92 }
93
94 void loop(){
95     // no code
96     // WiFi.disconnect(); // at the end !
97 }

```

---

La partie importante dans ce code est la fonction "connect\_wifi" :

- ① On place l'ESP en mode station : STA mode or WiFi client mode.

Explicitly set the ESP to be a WiFi-client, otherwise, it by default, would try to act as both a client and an access-point and could cause network-issues with your other WiFi-devices on your WiFi-network.

- ② On donne les "credentials" pour accéder au réseau (SSID + MDP) et on se connecte au réseau :

```
WiFi.begin(...)
```

- ③ On attend que la connection soit effective.

Pour terminer, on affiche les caractéristiques de cette "connexion" :

- ✓ IP Address,
- ✓ Mac Address,
- ✓ ...

Et là, on aimerait bien que la classe "Print" de Arduino IDE (qui implémente la sortie de caractères sur stdout) supporte un pauvre printf avec format de la lib C ... mais non ! faut passer par sprintf :-)

## 3.2 Few words on WiFi and ESP

[https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp\\_wifi.html](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html)

The WiFi libraries (ESP-IDF) provide support for configuring and monitoring the ESP32 WiFi networking functionality.

This includes configuration for :

- ✓ Station mode (aka STA mode or WiFi client mode) :

**ESP32 connects to an access point.**

- ✓ AP mode (aka Soft-AP mode or Access Point mode) :

**Other stations connect to the ESP32.**

- ✓ Combined AP-STA mode :

**ESP32 is concurrently an access point and a station connected to another access point.**

- ✓ Various security modes for the above (WPA, WPA2, WEP, etc.)

- ✓ Scanning for access points (active & passive scanning).

The reason for client scanning is to determine a suitable AP to which the client may need to roam now or in the future.

A client can use two scanning methods : active and passive.

- During an **active scan**, the client radio transmits a probe request and listens for a probe response from an AP.
- With a **passive scan**, the client radio listens on each channel for beacons sent periodically by an AP.

A passive scan generally takes more time, since the client must listen and wait for a beacon versus actively probing to find an AP.

Another limitation with a passive scan is that if the client does not wait long enough on a channel, then the client may miss an AP beacon.

- ✓ Promiscuous mode for monitoring of IEEE802.11 WiFi packets.

In computer networking, "promiscuous mode" is a mode for a wired network interface controller (NIC) or wireless network interface controller (WNIC) **that causes the controller to pass all traffic it receives to the central processing unit (CPU)** rather than passing only the frames that the controller is specifically programmed to receive.

- This mode is used for packet sniffing.

### 3.3 Recherche des SSIDs environnants

Ceci étant, on peut avoir à déployer des objets dans des environnements/SSID moins "statiquement définis" ... que l'on **ne connaissait PAS avant de déployer l'ESP sur site**.

➤ Dans ce cas, il faut pouvoir chercher des SSIDs.

Ce sketch recherche les SSIDs environnants et affiche leurs caractéristiques.

Si on peut identifier certains critères (SSID, open/encrypted, puissance, ...) parmi les réseaux obtenus alors on peut essayer de se connecter.

➤ C'est ce que l'on fait sur la fin de ce code.

```

1  /** Scan Wifi Network : wifiscan.ino ***/
2  #include <WiFi.h>
3
4  /*-----*/
5  String translateEncryptionType(wifi_auth_mode_t encryptionType) {
6      switch (encryptionType) {
7          case (WIFI_AUTH_OPEN):
8              return "Open";
9          case (WIFI_AUTH_WEP):
10             return "WEP";
11          case (WIFI_AUTH_WPA_PSK):
12             return "WPA_PSK";
13          case (WIFI_AUTH_WPA2_PSK):
14             return "WPA2_PSK";
15          case (WIFI_AUTH_WPA_WPA2_PSK):
16             return "WPA_WPA2_PSK";
17          case (WIFI_AUTH_WPA2_ENTERPRISE):
18             return "WPA2_ENTERPRISE";
19      }
20  }
21
22  /*-----*/
23  void print_network_status(int i){ // i : SSID index !!!
24      String s = "";
25      s += "\tIP_address:_:" + WiFi.localIP().toString() + "\n"; // bizarre
        IPAddress
26      s += "\tMAC_address:_:" + String(WiFi.macAddress()) + "\n";
27      s += "\tWifi_SSID:_:" + String(WiFi.SSID(i)) + "\n";
28      s += "\tWifi_Signal_Strength:_:" + String(WiFi.RSSI(i)) + "_dBm\n";
29      s += "\tWifi_Signal_Strength:_:" + String(constrain(2 * (WiFi.RSSI(i) + 100)
        , 0, 100)) + "_%\n";
30      s += "\tWifi_BSSID:_:" + String(WiFi.BSSIDstr(i)) + "\n";
31      s += "\tWifi_Encryption_type:_:" + translateEncryptionType(WiFi.
        encryptionType(i)) + "\n";
32      Serial.print(s);
33  }
34
35  /*-----*/

```

```

36 void setup() {
37     Serial.begin(9600);
38
39     int N = WiFi.scanNetworks(); // Scan Networks
40
41     if (N>0){ // Print descriptions if some ?
42         Serial.print("\n-----\n");
43         Serial.print("Networks found: #");
44         Serial.print(N);
45         Serial.print("\n");
46         for (int i=0 ; i<N ; i++){
47             char s[100];
48             sprintf(s, "SSID%d: \n", i); Serial.print(s);
49             print_network_status(i);
50             delay(1000);
51         }
52
53         // On peut aussi essayer de trouver le bon SSID !
54         // selon des criteres a definir ... un que je connais ?
55         #define GoodWiFiRSSI -90
56         int thegoodone = -1;
57         for (int i=0 ; i<N ; i++){
58             if ((String(WiFi.SSID(i)) == "HUAWEI-553A") && (WiFi.RSSI(i) >
59                 GoodWiFiRSSI) ){
60                 thegoodone = i;
61                 break;
62             }
63
64         if (thegoodone != -1){
65             Serial.print("The good one could be");
66             Serial.print(thegoodone);
67             WiFi.mode(WIFI_STA);
68             WiFi.disconnect(true); // delete old config
69             String ssid = WiFi.SSID(thegoodone);
70             String password = String("QTM06RTT");
71             WiFi.begin(ssid.c_str(), password.c_str(), 0, WiFi.BSSID(thegoodone));
72             //WiFi.begin(ssid.c_str(), password.c_str());
73             while (WiFi.status() != WL_CONNECTED){
74                 delay(500);
75                 Serial.print(".");
76             }
77         }
78
79         if (WiFi.status() == WL_CONNECTED){
80             Serial.print("\nWiFi connected: \n");
81             print_network_status(thegoodone); // Print status
82         }
83     }

```

```

84 }
85
86 void loop() {
87     // no code
88 }

```

### 3.4 Connexion parmi une liste

La bibliothèque "WiFIMulti" permet de se connecter au "meilleur" réseau Wifi parmi une liste.

➤ On récupère une adresse IP.

C'est la fonction `connect_wifi()` qui évolue :

```

1  /*** Try to Connect to the best AP based on a given list ***/
2
3  #include <WiFi.h>
4  #include <WiFIMulti.h>
5
6  WiFIMulti wifiMulti; // Creates an instance of the WiFIMulti class
7
8  /*-----*/
9  String translateEncryptionType(wifi_auth_mode_t encryptionType) {
10     // cf https://www.arduino.cc/en/Reference/WiFiEncryptionType
11     switch (encryptionType) {
12         case (WIFI_AUTH_OPEN):
13             return "Open";
14         case (WIFI_AUTH_WEP):
15             return "WEP";
16         case (WIFI_AUTH_WPA_PSK):
17             return "WPA_PSK";
18         case (WIFI_AUTH_WPA2_PSK):
19             return "WPA2_PSK";
20         case (WIFI_AUTH_WPA_WPA2_PSK):
21             return "WPA_WPA2_PSK";
22         case (WIFI_AUTH_WPA2_ENTERPRISE):
23             return "WPA2_ENTERPRISE";
24     }
25 }
26 /*-----*/
27 void print_network_status() { // Utilisation de String !
28     String s = "";
29     s += "\tIP_address: " + WiFi.localIP().toString() + "\n"; // bizarre
        IPAddress
30     s += "\tMAC_address: " + String(WiFi.macAddress()) + "\n";
31     s += "\tWifi_SSID: " + String(WiFi.SSID()) + "\n";
32     s += "\tWifi_Signal_Strength: " + String(WiFi.RSSI()) + "\n";
33     s += "\tWifi_BSSID: " + String(WiFi.BSSIDstr()) + "\n";
34     s += "\tWifi_Encryption_type: " + translateEncryptionType(WiFi.
        encryptionType(0)) + "\n";

```

```

35 // a mon avis bug ! => manque WiFi.encryptedType() !
36 Serial.print(s);
37 }
38 /*—————*/
39 void connect_wifi(){
40 // Set WiFi to station mode
41 WiFi.mode(WIFI_STA);
42 // and disconnect from an AP if
43 // it was previously connected
44 WiFi.disconnect();
45 delay(100); // ms
46
47 Serial.println(String("\nAttempting to connect to SSIDs: "));
48 wifiMulti.addAP("HUAWEI-6EC2", "FGY9MLBL");
49 wifiMulti.addAP("HUAWEI-553A", "QTM06RTT");
50 wifiMulti.addAP("GMAP", "vijx4705");
51 while(wifiMulti.run() != WL_CONNECTED) {
52     delay(1000);
53     Serial.print(".");
54 }
55
56 if(wifiMulti.run() == WL_CONNECTED) {
57     Serial.print("\nWiFi connected: \n");
58 }
59 }
60
61 /*———— Arduino IDE paradigm : setup+loop ———*/
62 void setup(){
63     Serial.begin(9600);
64     while (!Serial); // wait for a serial connection
65     connect_wifi(); //connect wifi
66     print_network_status();
67 }
68
69 void loop(){
70     // no code
71     // WiFi.disconnect();
72 }

```

## 4 Sockets

Maintenant que l'on est capable d'établir un lien de communication, on peut échanger.

On pourrait par exemple utiliser des sockets ...et mais il faudrait tout re-écrire :-(

- <https://leanpub.com/kolban-ESP32>
- <https://github.com/nkolban/esp32-snippets/blob/master/sockets/>

Comme on n'a pas trop le temps de développer "nos" protocoles :-( et qu'en plus il vaut mieux rester sur la "voie standard".

On va donc utiliser des protocoles standards d'applications : HTTP et MQTT.



## 5 HTTP

Maintenant que la liaison au réseau est réalisée, on va pouvoir dialoguer avec l'ESP.

---

Cette notion de "dialogue" est assez large et peut utiliser des architectures bien différentes :

- récupérer les informations qu'il collecte dans le monde réel  
...et l'ESP ne ferait alors que répondre à des requêtes.
- laisser l'ESP choisir les instants de communications  
...c'est lui qui démarre le dialogue et le serveur (au centre du réseau) est en attente.
- ...

---

Dans cette partie, on souhaite baser ces dialogues sur le protocole HTTP.

On va utiliser des bibliothèques qui permettent de disposer de la faculté d'émettre et de recevoir des requêtes/réponses HTTP.

- Le "problème" c'est qu'il y en a pas mal qui font "presque" la même chose ...avec des interfaces "presques" identiques ... mais pas tout a fait :-(
- C'est un peu pénible!

J'en ai choisi quelques unes.

### 5.1 Bibliothèque <WiFi>

On vient de voir que cette bibliothèque est majeure dans l'établissement de la connexion au réseau.

Mais cette bibliothèque **offre plus** que l'établissement d'un lien WiFi.

Elle "déborde" largement et propose des fonctionnalités de type sockets :

- la spécification d'un port (couche transport),
- l'établissement d'une connexion,
- ...

### 5.2 L'ESP est un client (GET) HTTP

Dans ce premier exemple, on veut réaliser depuis l'ESP une requête `GET` sur un serveur HTTP :

- **L'ESP est donc le client.**

Pour commencer, le serveur Web ciblé est la machine : `httpbin.org`

Pour l'instant la requête est "toute simple" puisqu'elle ne contient pas de paramètres, juste l'URL :

```
http://httpbin.org/ip
```

La page HTML, produite en réponse par ce serveur, contient l'adresse IP de la machine qui a formulé la requête ... (donc de l'ESP).

Pour savoir ce que l'ESP va recevoir en réponse, vous pouvez utiliser `curl` pour formuler votre requête en ligne de commande :

```
~> curl http://httpbin.org/ip
{
  "origin": "134.59.131.45"
}
```

et éventuellement si vous voulez en savoir plus sur la requête effectuée :

```
~> curl -v http://httpbin.org/ip
* Trying 3.209.99.235:80...
* TCP_NODELAY set
* Connected to httpbin.org (3.209.99.235) port 80 (#0)
> GET /ip HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Mon, 24 Jan 2022 15:11:14 GMT
< Content-Type: application/json
< Content-Length: 32
< Connection: keep-alive
< Server: gunicorn/19.9.0
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
<
{
  "origin": "109.210.12.69"
}
* Connection #0 to host httpbin.org left intact
```

### 5.2.1 Le serveur "httpbin.org"

Le site <http://httpbin.org/> "**fait écho**" aux données utilisées dans votre requête :

<b><a href="http://httpbin.org/anything">http://httpbin.org/anything</a></b>	: Renvoie la plupart des éléments ci-dessous.
<b><a href="http://httpbin.org/ip">http://httpbin.org/ip</a></b>	: Renvoie l'adresse IP d'origine.
<b><a href="http://httpbin.org/user-agent">http://httpbin.org/user-agent</a></b>	: Renvoie l'agent utilisateur.
<b><a href="http://httpbin.org/headers">http://httpbin.org/headers</a></b>	: Renvoie le dictionnaire représentant l'en-tête.
<b><a href="http://httpbin.org/get">http://httpbin.org/get</a></b>	: Renvoie les données de la requête GET.
<b><a href="http://httpbin.org/post">http://httpbin.org/post</a></b>	: Renvoie les données de la requête POST.
...	==> On y reviendra!
...	...

Il y a un site Web associé sur lequel vous pouvez utiliser l'UI pour produire une requête et analyser sa réponse :

[http://httpbin.org/#/Request\\_inspection/get\\_ip](http://httpbin.org/#/Request_inspection/get_ip)

GET /ip Returns the requester's IP Address.

Parameters

No parameters

Execute Clear

Responses Response content type: application/json

Curl

```
curl -X GET "http://httpbin.org/ip" -H "accept: application/json"
```

Request URL

```
http://httpbin.org/ip
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "origin": "134.59.131.45" }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-origin: * connection: keep-alive content-length: 32 content-type: application/json date: Thu, 20 Aug 2020 13:39:38 GMT server: gunicorn/19.9.0</pre>

Responses

Code	Description
200	The Requester's IP Address.

Si ce site ne vous plaît pas ... vous pouvez en trouver d'autres sur :

<https://mixedanalytics.com/blog/list-actually-free-open-no-auth-needed-apis/>

### 5.2.2 Même requête ... en utilisant "WiFiClient"

Pour réaliser cette requête GET, on utilise dans une premier temps l'objet `WiFiClient` de `<WiFi.h>`

Cet objet `WiFiClient` s'apparente donc à une socket :

- Il permet de se connecter et d'échanger.
- Par contre, **il n'aide en rien sur la construction syntaxique du message** et donc la requête doit être composée explicitement.

```

1 #include <WiFi.h>
2 #include "classic_setup.h"
3
4 /*
5  * This sketch sends data via HTTP GET requests to host/url
6  * and returns the website in html format which is printed on the
7  * console
8  */
9
10 /*—— Arduino IDE paradigm : setup+loop ——*/
11 void setup(){

```

```

12 | Serial.begin(9600);
13 | while (!Serial); // wait for a serial connection
14 | connect_wifi(); // connect wifi
15 | print_network_status();
16 | }
17 |
18 | void loop() {
19 |     WiFiClient client;
20 |     // IPAddress server(216,58,205,195); // Google.com IP
21 |
22 |     //char host[100] = "http://worldtimeapi.org";
23 |     char host[100] = "httpbin.org";
24 |     const int httpPort = 80;
25 |
26 |     Serial.print("connecting to ");
27 |     Serial.println(host); // Use WiFiClient class to create TCP connections
28 |     if (client.connect(host, httpPort) != 1) {
29 |         Serial.println("connection failed");
30 |         return;
31 |     }
32 |     else{
33 |         Serial.println("connection succeeded");
34 |     }
35 |     // BOF BOF ... meme avec un reseau out ...
36 |     // la demande de connexion avec l'host rend 1 => Bizarre !!
37 |
38 |     // Now create a URI for the GET/HTTP request :
39 |     // this url contains the information we want to send
40 |     // to the server => GET style !!
41 |     //String url = "/api/timezone/Europe/Paris";
42 |     String url = "/ip";
43 |
44 |     // Now create HTTP request header
45 |     String req = String("GET ");
46 |     req += url + " HTTP/1.1\r\n";
47 |     req += "Host: " + String(host) + "\r\n";
48 |     req += "User-Agent: esp-idf/1.0 esp32\r\n";
49 |     req += "Accept: */*\r\n";
50 |     req += "Connection: close\r\n";
51 |     req += "\r\n"; // empty line : separator header/body
52 |
53 |     Serial.println("Request Header: ");
54 |     Serial.println(req);
55 |
56 |     // Send request to host through client socket
57 |     Serial.println("Send request to URL: ");
58 |     Serial.println(url);
59 |     client.print(req); // Send !
60 |
61 |     // https://www.arduino.cc/en/Reference/WiFiClientAvailable
62 |     unsigned long timeout = millis();
63 |     while (client.available() == 0) { // no answer => timeout mechanism !
64 |         if (millis() - timeout > 10000) {
65 |             Serial.println(">>>> Client Timeout!");
66 |             client.stop();
67 |             return;
68 |         }
69 |     }
70 |
71 |     //Wait for server response
72 |     //while (client.available() == 0);
73 |

```

```

74 // Read response
75 Serial.println("Response:");
76 // all the lines of the reply from server and print them to Serial
77 while (client.available()) { // Returns the number of bytes available for reading
78   String line = client.readStringUntil('\r');
79   Serial.print(line); // echo to console
80
81   // en version car/car
82   //char c = client.read();
83   //Serial.print(c);
84 }
85
86 Serial.println();
87 Serial.println("closing connection");
88
89 delay(10000); //ms
90 }

```

La construction du message (vers ligne 45) contenant la requête est conforme au protocole HTTP :

- A message has a header part and a message body separated by a blank/empty line.
  - ✓ The blank line is ALWAYS needed **even if there is no** message body.
  - ✓ The header starts with a command (here "GET") and has additional lines of key value pairs separated by a colon and a space.
  - ✓ If there is a message body, it can be anything you want it to be.
- Lines in the header and the blank line at the end of the header **must end with a carriage return and linefeed pair**

CRLF = `\r\n`.

<https://stackoverflow.com/questions/5757290/http-header-line-break-style>

- The "Connection : Close" header tells the server to close the connection after the request has been processed.  
You could have alternatively send Connection : Keep-Alive if you wanted the connection to be kept alive after the request was processed.
- Accept : \*/\*  
Client will accept any MIME type.

Ligne 59 on envoie cette requête et si il y a une réponse on a lit à la ligne 77.

### 5.2.3 Quelques URL !

Une URL peut avoir différentes formulations.

On en montre quelques unes :

- La plus simple : `http://host:port/path`

Exemple :

`curl -v "http://httpbin.org/get"`

- Une URL avec un paramètre : `http://host:port/path?query_string`

La "query string" est un ensemble de paires : nom/valeur

Exemple qui ne marche plus ... Google's Terms of Service :-(

```
curl -v "https://www.google.com/search?q=esp32"
```

Google propose désormais "son API" pour faire des recherches à partir de scripts ! Payant ?

- Une URL avec plusieurs paramètres : `http://host:port/path?key1=val1&key2=val2`

```
curl -v "http://httpbin.org/get?led1=ON&led2=OFF"
```

Exemple qui ne marche pas ... mais Google vous le dit => Google's Terms of Service :-(

```
curl -v "https://www.google.com/search?channel=fs&client=ubuntu&q=esp32"
```

Vous pouvez essayer cette requête (sans le curl) dans un navigateur ...

Pour l'instant la requête est simple et la réponse est affichée sur la console :

```
10:48:08.848 -> connecting to httpbin.org
10:48:09.080 -> Requesting URL: /ip
10:48:09.512 -> HTTP/1.1 200 OK
10:48:09.512 -> Date: Wed, 19 Aug 2020 08:48:09 GMT
10:48:09.545 -> Content-Type: application/json
10:48:09.578 -> Content-Length: 32
10:48:09.611 -> Connection: close
10:48:09.644 -> Server: gunicorn/19.9.0
10:48:09.644 -> Access-Control-Allow-Origin: *
10:48:09.677 -> Access-Control-Allow-Credentials: true
10:48:10.606 ->
10:48:10.606 -> {
10:48:10.606 ->   "origin": "80.214.145.35"
10:48:10.639 -> }
10:48:10.639 ->
10:48:10.639 -> closing connection
```

### 5.2.4 Requête ... en utilisant <HTTPClient>

Pour exprimer que l'on souhaite faire une requête "GET", sans pour autant devoir l'écrire complètement, on va utiliser la bibliothèque `HTTPClient`

➤ le HTTP en majuscules!!! sinon ce n'est pas la même :-)

<https://github.com/espressif/arduino-esp32/tree/master/libraries/HTTPClient/examples>

Cette bibliothèque développe des objets "plus" spécialisés pour HTTP et facilite l'utilisation des requêtes.

Dans l'exemple qui suit, on utilise cette bibliothèque pour envoyer périodiquement des requêtes GET à un serveur.

```
curl -v http://httpbin.org/get?led1="OFF"&led2="ON"
```

```
1  /*
2  * Fichier : esp_httpclient.ino
3  * Auteur : G.Menez
4  => based on Rui Santos excellent work
5  https://randomnerdtutorials.com/esp32-http-get-post-arduino/
6  */
7  #include <WiFi.h>
8  #include "classic_setup.h"
9  #include <HTTPClient.h>
10
11 // the following variables are unsigned longs because the time, measured in
12 // milliseconds, will quickly become a bigger number than can be stored in an int.
13 unsigned long lastTime = 0;
14 // Set timer
15 unsigned long loop_period = 10L * 1000; /* => 10000ms : 10 s */
16
17 char host[100] = "http://httpbin.org"; // = "http://worldtimeapi.org/";
18 const int httpPort = 80;
19 String path = "/get";
20 String params = "?led1=" "OFF" "&led2=" "ON" "";
21
22 /*-----*/
23 String httpGETRequest(const char* UrlServer) {
24     // return the response of the GET request to UrlServer
25     HTTPClient http; // http protocol entity => client
26
27     Serial.print("Requesting URL: ");
28     Serial.println(UrlServer);
29
30     // Your IP address with path or Domain name with URL path
31     http.begin(UrlServer); // Parse URL of the server
32
33     // Send HTTP request
34     int httpResponseCode = http.GET();
35
36     // Get the response => will fill payload String
37     String payload = "{}";
38     if (httpResponseCode > 0) {
39         Serial.print("HTTP Response code: ");
40         Serial.println(httpResponseCode);
41         payload = http.getString();
42     }
43     else {
44         Serial.print("Error code on HTTP GET Request: ");
```

```

45     Serial.println(httpResponseCode);
46 }
47 // End connection and Free resources
48 http.end();
49
50 return payload;
51 }
52
53 /*----- Arduino IDE paradigm : setup+loop -----*/
54 void setup(){
55     Serial.begin(9600);
56     while (!Serial); // wait for a serial connection
57     connect_wifi();//connect wifi
58     print_network_status();
59 }
60 /*-----*/
61 void loop() {
62     String url = String(host)+path+params;
63
64     //Send an HTTP request every loop_period in ms
65     if ((millis() - lastTime) > loop_period) {
66         //Check WiFi connection status
67         if(WiFi.status()== WL_CONNECTED){
68
69             String ret = httpGETRequest(url.c_str());
70             Serial.println(ret);
71         }
72         else {
73             Serial.println("WiFi_Disconnected");
74         }
75         lastTime = millis();
76     }
77 }

```

Cet exemple **est important** car il est représentatif d'un comportement possible de l'objet dans le réseau.

Ici, **le dialogue est à l'initiative de l'ESP** :

- Envoi périodique d'information/ des valeurs des capteurs à un serveur (situé plus au centre du réseau).
- Utilisation d'une requête GET du protocole HTTP.
- Dans ce code, bien que l'on ait accès à la charge/payload de la réponse à cette requête, nous ne l'utilisons pas.  
On pourrait sans doute en faire quelquechose!?

On comprend aussi que la transmission de données par ce type de requête peut vite devenir lourde et ne convient pas si on veut formater le message en JSON et le mettre dans la "payload" de la requête.

- Il va falloir faire des requêtes POST!



### 5.3 L'ESP est un serveur (GET) HTTP

On vient d'apprendre à faire des ESP des clients HTTP qui émettaient des requêtes.

➤ **On va maintenant faire des ESP des serveurs HTTP ... capable de répondre** à des requêtes!

#### 5.3.1 Version simpliste avec <WiFi.h>

On commence par réaliser un serveur HTTP simple.

➤ Il est simple car il ne gère pas les "paths" des URLs.

En fait, dès qu'il reçoit une demande de connexion sur le port, il émet un message qui est un morceau d'HTML qui contiendra le temps en ms depuis le début de l'exécution du programme par l'ESP.

Donc, si depuis un navigateur (qui serait le client), on se connecte sur ce serveur (invoque l'URL qui correspond à son IP), on devrait voir afficher du HTML!

➤ Attention, tout le monde (client et serveur) doit être joignable : Soyez attentif aux réseaux que vous utilisez!

Ce serveur est construit sur la bibliothèque <WiFi.h> => objets WiFiClient et WiFiServer.

```

1  /*
2   * Fichier : esp_wifiserver.ino
3   * Auteur : G.Menez
4   */
5  #include <WiFi.h>
6  #include "classic_setup.h"
7
8  // Set timer
9  unsigned long loop_period = 10L * 1000; /* => 10000ms : 10 s */
10
11 // Instanciation of a "Web server" on port 80
12 WiFiServer server(80);
13
14 /*----- Arduino IDE paradigm : setup+loop -----*/
15 void setup(){
16   Serial.begin(9600);
17   while (!Serial); // wait for a serial connection. Needed for native USB port only
18   connect_wifi(); // Connexion Wifi
19   print_network_status();
20
21   server.begin(); // Lancement du serveur
22 }
23
24 void loop() {
25
26   // listen for incoming clients
27   WiFiClient client = server.available();
28
29   if (client) { // incoming client
30     Serial.println("New client is connecting!");
31     // an http request ends with a blank line CRLF
32
33     boolean currentLineIsBlank = true;
34
35     while (client.connected()) {
36       if (client.available()) {
37
38         char c = client.read(); // Echo on the console
39         Serial.write(c);

```

```

40
41 // if you've gotten to a CRLF the http GET request has ended,
42 // so you can send a reply
43 if (c == '\n' && currentLineIsBlank) {
44     httpReply(client);
45     break;
46 }
47 if (c == '\r') { // you're starting a new line
48     currentLineIsBlank = true;
49 } else if (c != '\r') { // you've gotten a character on the current line
50     currentLineIsBlank = false;
51 }
52 }
53 }
54
55 // give the web browser time to receive the data
56 delay(loop_period); // ms
57
58 // close the connection :
59 client.stop();
60 Serial.println("client disconnected");
61 }
62 }
63 /*-----*/
64 void httpReply(WiFiClient client) {
65     // this method makes a simple HTTP GET reply
66     // the body syntax is HTML
67     // => supposed to be displayed by a navigator
68     client.println("HTTP/1.1 200 OK");
69     client.println("Content-Type: text/html");
70     client.println("Connection: close"); // the connection will be closed after
71                                         // completion of the response
72     client.println("Refresh: 5"); // refresh the page automatically every 5 sec
73
74     client.println(); // Empty line between header and body
75
76     client.println("<!DOCTYPE HTML>");
77     client.println("<html>");
78     client.print("Hello, je tourne depuis: "); // Returns the ms passed since the ESP
79                                               // began running the current program.
80     client.print(millis()/1000); // On pourrait sans doute donner une info
81     client.println("s<br/>"); // plus pertinente ? temperature ?
82     client.println("</html>");
83 }

```

## 5.4 ESPAsyncWebServer

Pour améliorer de nombreux points de cette première version, nous allons utiliser la bibliothèque "ESPAsyncWebServer" qui présente de nombreuses fonctionnalités intéressantes et notamment la gestion de "handlers" (induits par le paradigme asynchrone) :

<https://github.com/me-no-dev/ESPAsyncWebServer>

Construire un serveur web asynchrone présente plusieurs avantages, comme mentionné dans la page GitHub de la bibliothèque, tels que :

- "Gérer plus d'une connexion en même temps" ;
- "Lorsque vous envoyez la réponse, vous êtes immédiatement prêt à gérer d'autres connexions pendant que le serveur se charge d'envoyer la réponse en arrière-plan" ;
- "Un moteur de traitement des modèles simple pour gérer les modèles/handlers" ;
- Et bien plus encore ...

Pour ceux, qui connaissent la package "express" en Javascript, ils trouveront une similitude fonctionnelle. Pour ceux qui ne connaissent pas ... ils vont bientôt connaître !

### 5.4.1 Installation

Pour installer les deux bibliothèques nécessaires (AsyncTCP et ESPAsyncWebServer), j'ai juste extrait les archives obtenues par Git dans le répertoire "HomeRepertoire/Arduino/libraries".

- Il faut ensuite aussi re-ouvrir l'IDE Arduino.

### 5.4.2 Pour faire quoi ?

On reste dans le contexte de la régulation de température.

L'ESP conserve son rôle de serveur et répond selon l'URL/route qui lui est soumise renvoie :

- soit une page HTML qui donne son statut,
- soit la valeur (String) d'un de ses capteurs,
- soit ...

## 5.5 La partie Html

Dans l'exercice précédent, l'ESP renvoyait une page HTML.

- On pourrait continuer à placer du HTML dans le fichier ".ino" en utilisant une chaîne de caractère (PROGMEM, ...)

<https://github.com/me-no-dev/ESPAsyncWebServer#send-large-webpage-from-proGMEM-containing-templates>

mais ce n'est pas l'idéal ni à écrire (une grande chaîne de caractères) ni pour structurer le programme.

On peut faire mieux !

On crée un fichier HTML "statut.html" que l'on va placer dans la flash memory : cf SPIFFS

```

1  <!DOCTYPE HTML><html>
2  <head>
3    <meta name="viewport" content="width=device-width, initial-scale=1">
4    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
5    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCq
6    <style>
7      html {
8        font-family: Arial;
9        display: inline-block;
10       margin: 0px auto;
11       text-align: left;
12     }
13     h2 { font-size: 3.0rem; }
14     p { font-size: 3.0rem; }
15     .units { font-size: 1.2rem; }
16     .sensors-labels{
17       font-size: 1.5rem;
18       vertical-align:middle;
19       padding-bottom: 15px;
20     }
21     div {
22       max-width : 500px;
23       word-wrap: break-word;
24     }
25     .grid-container {
26       display: grid;
27       grid-template-columns: auto auto;
28       background-color: #2196F3;
29       padding: 10px;
30     }
31     .grid-item {
32       background-color: rgba(255, 255, 255, 0.8);
33       border: 1px solid rgba(0, 0, 0, 0.8);
34       padding: 20px;
35       font-size: 30px;
36       text-align: center;
37     }
38   </style>
39   <title>ESP32</title>
40 </head>
41
42 <body>
43   <h1>ESP32</h1>
44   <h3>Object status :</h3>
45   Uptime      : %UPTIME% s<br/>
46   Where       : %WHERE% s<br/>
47
48   <h3>Network link status :</h3>
49   WiFi SSID   : %SSID%<br/>
50   MAC address : %MAC%<br/>
51   IP address  : %IP%<br/>
52
53   <h3>Sensors status :</h3>
54   Temperature : %TEMPERATURE% C<br/>
55   Light       : %LIGHT% Lumen<br/>
56   Cooler      : %COOLER%<br/>
57   Heater      : %HEATER%<br/>
58
59   <h3>Sensors status WITH GLYPHS :</h3>
60   <div class="grid-container">
61     <div class="grid-item">
62       <i class="fas fa-thermometer-half" style="color:#059e8a;"></i>

```

```

63     </div>
64     <div class="grid-item">
65         <span class="sensors-labels">Temperature</span>
66         <span id="temperature">%TEMPERATURE%</span>
67         <sup class="units">&deg;C</sup>
68     </div>
69     <div class="grid-item">
70         <i class="far fa-lightbulb" style="color:#00add6;"></i>
71     </div>
72     <div class="grid-item">
73         <span class="sensors-labels">Light</span>
74         <span id="light">%LIGHT%</span>
75         <sup class="units">Lumen</sup>
76     </div>
77 </div>
78
79 <script>
80     setInterval(function ( ) {
81         var xhr = new XMLHttpRequest(); // Constructor
82         // XMLHttpRequest changes between states as it progresses
83         xhr.onreadystatechange = function() { // Handler to track XMLHttpRequest object state
84             // DONE = 4; when request complete and return OK (200)
85             if (this.readyState == 4 && this.status == 200) {
86                 var r = this.response //renvoie le texte reçu d'un serveur suite à l'envoi d'une requête.
87                 //console.log(r);
88                 // .innerHTML method is used to change the html contents of a DOM object
89                 document.getElementById("temperature").innerHTML = r; // temperature
90             }
91         };
92         xhr.open("GET", "/temperature", true); // true => asynchrone open call,
93         //Contrary to its name, does not open the connection. It
94         //only configures the request, but the network activity only
95         //starts with the call of send.
96         xhr.send(); //This method opens the connection and sends the request to server.
97     }, 1000 );
98
99     setInterval(function ( ) {
100         var xhr = new XMLHttpRequest();
101         xhr.onreadystatechange = function() {
102             if (this.readyState == 4 && this.status == 200) {
103                 document.getElementById("light").innerHTML = this.responseText;
104             }
105         };
106         xhr.open("GET", "/light", true);
107         xhr.send();
108     }, 1000 );
109 </script>
110
111 <h3>Thresholds :</h3>
112 Day/Night Light : %LT% Lumen<br/>
113 Day - Low Temp : %SBJ% C<br/>
114 Day - High Temp : %SHJ% C<br/>
115 Night - Low Temp : %SBN% C<br/>
116 Night - Low Temp : %SHN% C<br/>
117
118 <h3> Periodic sensor data sent to :</h3>
119 <form action="/target" method="post">
120
121     <label for="ip">IP Address :</label>
122     <input type="text" name="ip" placeholder="%PRT_IP%"><br/>
123
124     <label for="port">Port :</label>

```

```

125     <input type="text" name="port" placeholder="%PRT_PORT%"/><br/>
126
127     <label for="sp"> Sampling Period (in seconds) :</label>
128     <input type="text" name="sp" placeholder="%PRT_T%"/><br/>
129
130     <input type="submit" value="Change reporting host !"/>
131 </form>
132 </body>
133 </html>

```

La partie HTML (c'est à dire la page que va renvoyer l'ESP) fait classiquement apparaître deux parties :

- ① Le "header" avec les "incantations" et paramétrages :
  - "meta" (ici optimisé pour les mobiles),
  - "link" (pour charger les icons qui sont des ressources externes),
  - "style" (pour définir quelques styles CSS).
- ② Le "body" avec l'information qui représentera le statut de l'ESP.

Dans ce body, j'attire votre attention sur certains points :

- Des "placeholders" (TEMPERATURE, LIGHT, ...) destinés à être remplacés par leurs valeurs respectives grâce à la fonction "processor" transmise à la bibliothèque (cf `server.on(...)`).  
On pourra ainsi faire le lien entre "identificateur de placeholder" et "valeur du capteur correspondant".
- Une grid avec des glyphs ... pour jouer avec HTML ...  
Vous pouvez trouver d'autres icones sur le site fontawesome :  
<https://fontawesome.com/icons?d=gallery>  
Le tag `<i>` fait le reste.
- Le "script" (JS) permet d'installer des requêtes HTTP **périodiques** (coté client/navigateur => XMLHttpRequest).

## RMQ :

On n'oublie pas la console du navigateur pour mettre au point et chercher les bugs !

### 5.5.1 Le Script

Le "script" (JS) contenu dans la page permet d'installer des requêtes HTTP **périodiques coté client/navigateur**.

- La page affichée sur le navigateur va ainsi solliciter périodiquement le serveur (l'ESP) pour mettre à jour la température et la lumière.

<https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest>

Les objets XMLHttpRequest (XHR) du navigateur (accessible en JavaScript) permettent d'interagir avec des serveurs.

- On peut récupérer des données à partir d'une URL sans avoir à rafraîchir complètement la page.  
Cela permet à une page web d'être mise à jour sans perturber les actions de l'utilisateur.

La propriété "XMLHttpRequest.onreadystatechange" contient le gestionnaire d'évènement invoqué lorsque la propriété "readyState" change.

- J'ai essayé de commenter le code pour expliquer ... hope this help !

### 5.5.2 La partie "Periodic report to"

L'idée de cette partie est de pouvoir configurer (au niveau de l'ESP) l'adresse d'un serveur de données auquel l'ESP enverrait périodiquement les valeurs de ses capteurs par des requêtes POST.

Visiblement (cf les fichiers .html et .ino) l'ESP va recevoir une requête `"/target"` POST (issue du formulaire) dans laquelle les coordonnées de ce serveur seront indiquées.

- C'est à l'utilisateur de remplir et cliquer !
- C'est à l'ESP de gérer la requête.

## 5.6 La partie ESP

**ESPAsyncWebServer** est donc une bibliothèque permettant d'utiliser l'ESP comme un serveur Web Asynchrone.

La notion d'asynchronisme induit la possibilité d'associer des handlers qui sont utilisés pour exécuter (de façon asynchrone) des actions spécifiques selon la requête :

<https://github.com/me-no-dev/ESPAsyncWebServer#handlers-and-how-do-they-work>

### 5.6.1 Requête/Route/Handler

La requête est caractérisée par un type, une route et une lambda C++ qui prend en paramètre la requête et qui invoque une méthode (par exemple `"send"` ou `"send_P"` ou ...)

<https://github.com/me-no-dev/ESPAsyncWebServer/blob/master/README.md#request-life-cycle>

Ainsi, par exemple, du fait de la présence de l'appel :

```
// Declaring root handler, and action to be taken when root is requested
auto root_handler = server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    /* This handler will download statut.html (SPIFFS file) and will send it back */
    request->send(SPIFFS, "/statut.html", String(), false, processor);
    // cf "Respond with content coming from a File containing templates" section
    //in manual !
    // https://github.com/me-no-dev/ESPAsyncWebServer
});
```

si l'ESP reçoit une requête GET avec le path `"/"` alors en tant que serveur il répond en envoyant le fichier `"/statut.html"` (avec placeholders) contenu en SPIFFS.

🔍 192.168.219.4/

## ESP32

**Object status :**

Uptime : s



**Network link status :**

MAC address :  
IP address :

**Sensors status :**

Temperature : 26.19 C  
Light : 131 Lumen  
Cooler :  
Heater :

**Sensors status WITH GLYPHS :**

	Temperature 25.19 °C
	Light 129 Lumen

**Thresholds :**

Day/Night Light : Lumen  
Day - Low Temp : C  
Day - High Temp : C  
Night - Low Temp : C  
Night - Low Temp : C

**Periodic report to:**

IP Address :   
Port :   
Sampling Period :

La bibliothèque permet aussi de définir des requêtes renvoyant tout simplement du texte.  
La preuve :

```
~> curl http://192.168.219.4/temperature
26.12
```

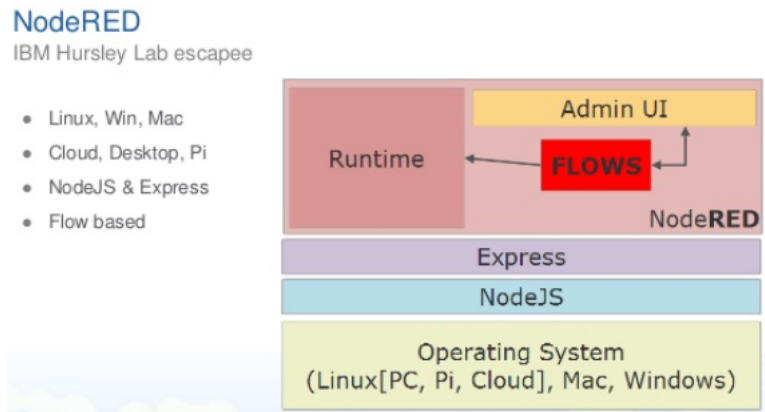


## 6 Node-RED

Node-Red (<https://nodered.org/>) est une proposition d'IBM permettant

- ① de lier aisément des sources de données à des composants de traitement, locaux ou distants,
- ② et de créer des chaînes de valeurs en quelques clics dans un environnement Web.

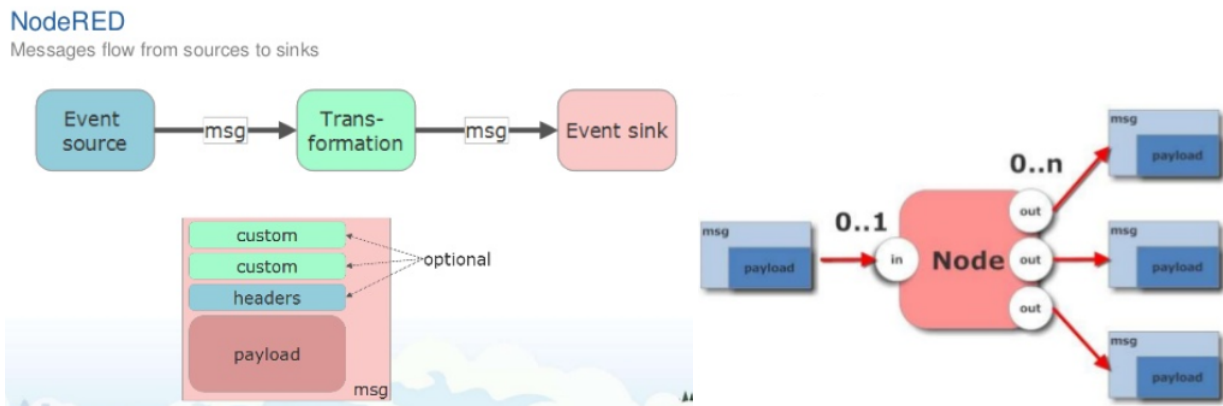
Techniquement, Node-Red est écrite en Javascript et repose au runtime sur Node.js.



<https://www.slideshare.net/AmitChaudhary112/salesforce-apex-hours-node-red-for-salesforce>

Node-Red utilise une approche de programmation visuelle qui permet aux développeurs de connecter les blocs de code ensembles.

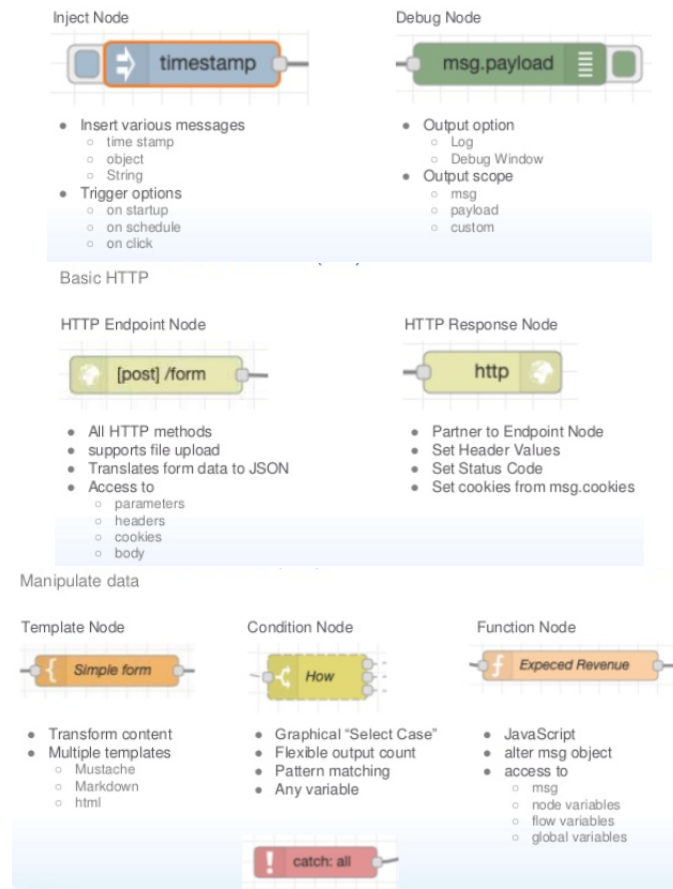
- Les noeuds connectés, généralement une combinaison de noeuds d'entrée, de noeuds de traitement et de noeuds de sortie, lorsqu'ils sont câblés ensemble, constituent un **"flow"**.



Les flows créés sont stockés/sérialisés au format JSON et il existe d'ailleurs une bibliothèque de "flows" en ligne :

<https://flows.nodered.org>.

Quelques noeuds :



## 6.1 Installation

Les prérequis sont `node` et `npm` ("son" gestionnaire de paquets) .

Ensuite Node-Red s'installe comme un package :

```
https://nodered.org/docs/getting-started/local
```

Depuis cette année, j'utilise "docker" pour exécuter Node-Red :

```
https://hub.docker.com/r/nodered/node-red
```

## 6.2 Execution

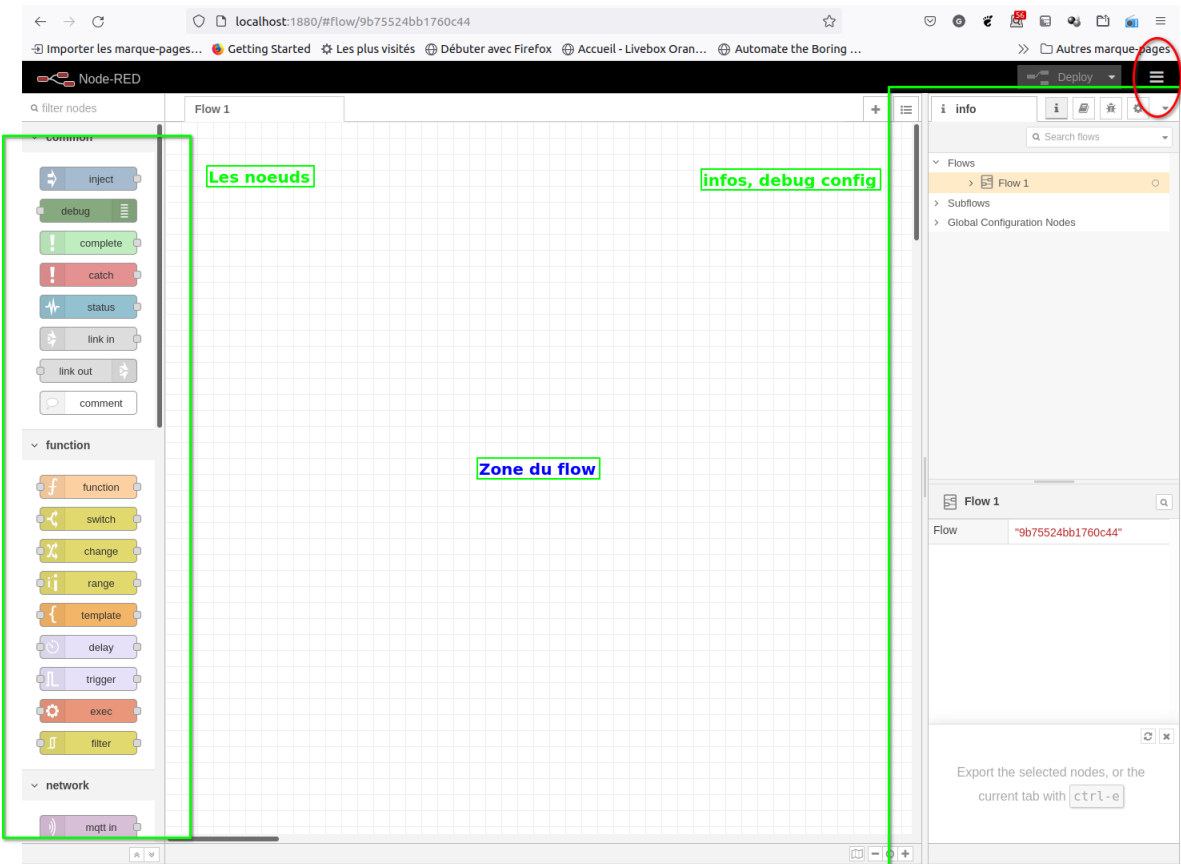
Dans un terminal, vous exécutez "node-red" et dans un navigateur vous pouvez alors accéder à l'interface grâce à l'URL :

```
http://localhost:1880
```

Avec docker :

```
docker run -it -p 1880:1880 -v /home/username/NodeRed/Data:/data --name mynodered nodered/n
```

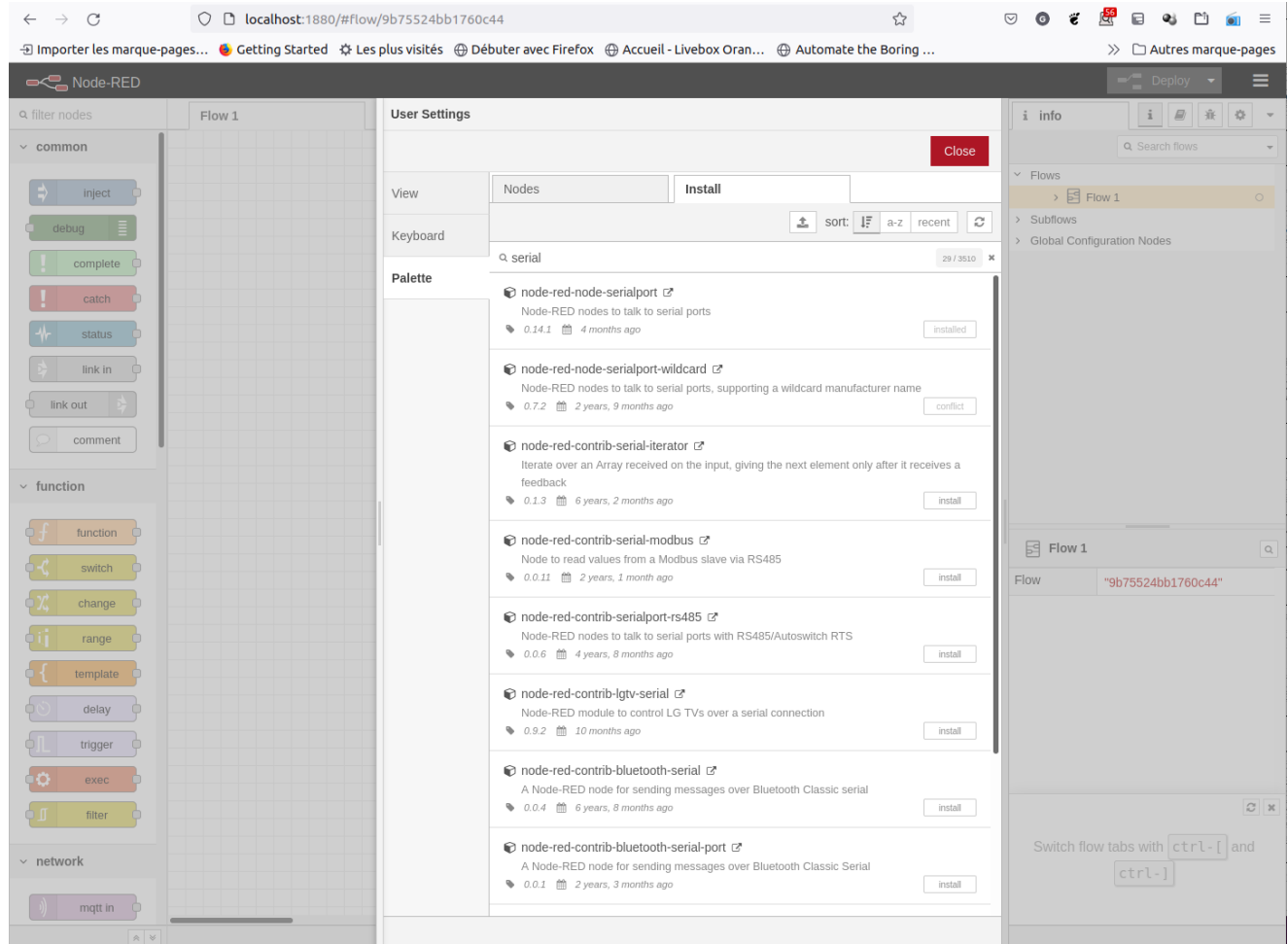
- /home/mark/NodeRed/Data est le répertoire où sont stockés les flows.
- mynodered est le nom du container.
- nodered/node-red est le nom de l'image.



A gauche les noeuds, au centre le flow (vide pour l'instant) et à droite infos, debug, config ...

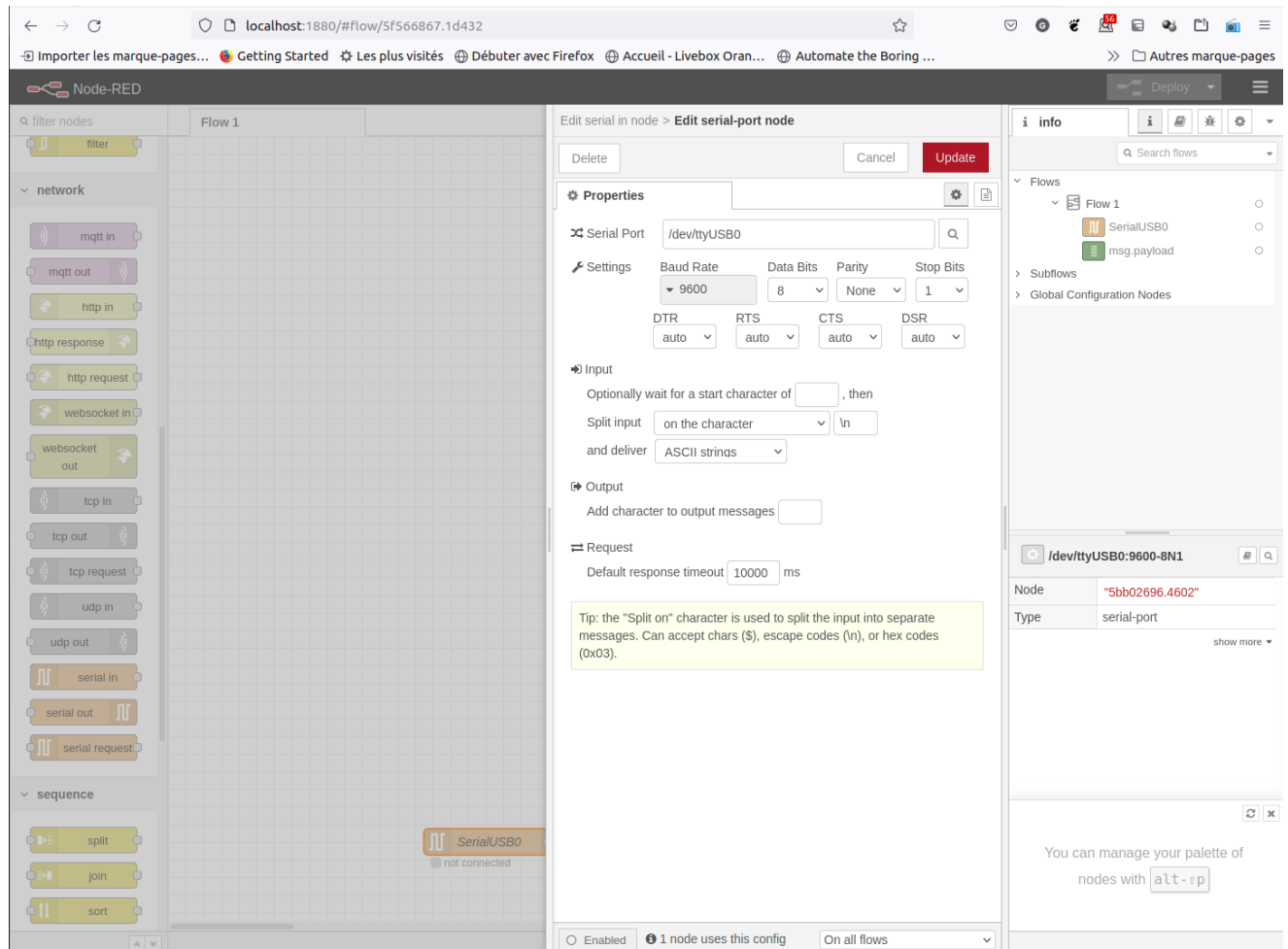
### 6.3 Un exemple ...

Pour commencer, rendez-vous sur votre interface Node-Red et accédez (menu coin haut droit) au gestionnaire de Palette ("Manage Palette") pour installer le package officiel : `node-red-node-serialport`



Ensuite, vous placez dans le flow différents composants :

- Un noeud de type **"input"** (carré gris à droite) qui va permettre d'injecter des données dans le flow. Vous choisissez le "serial in" dans la catégorie/palette "network" qui, comme son nom l'indique, permet d'utiliser le port série, plus précisément ce qui va sortir du port série et que l'on va injecter dans le flow. Il faut paramétrer ce noeud pour préciser le port série physique de la machine qu'il représente :



- Un noeud de type **"output"** (carré gris à gauche) qui va permettre d'envoyer des données vers le monde réel ou dans notre cas vers une console de debug.

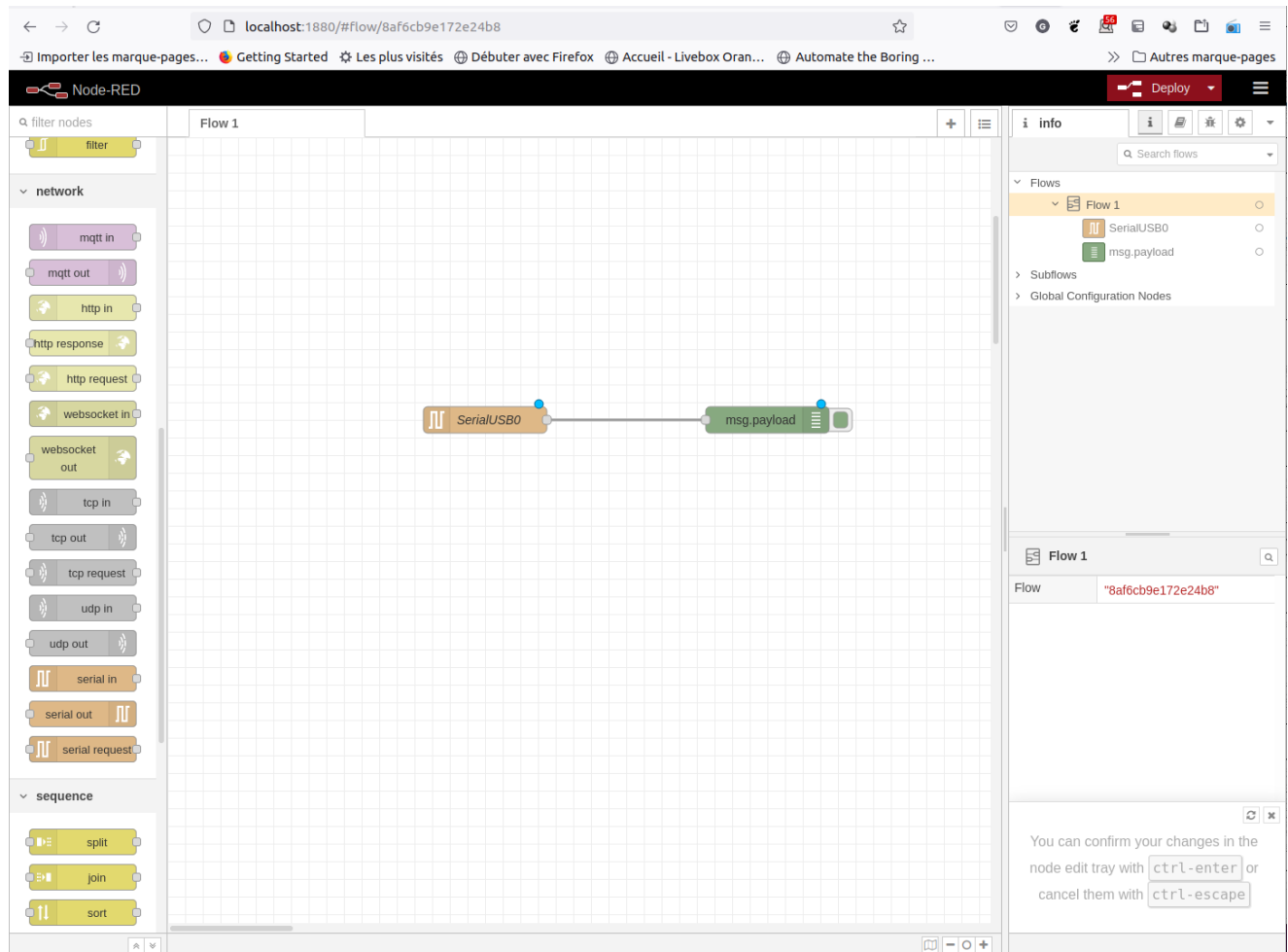
Vous choisissez dans la catégorie/palette "Common" le noeud "debug".

- Et un lien entre ces deux noeuds.

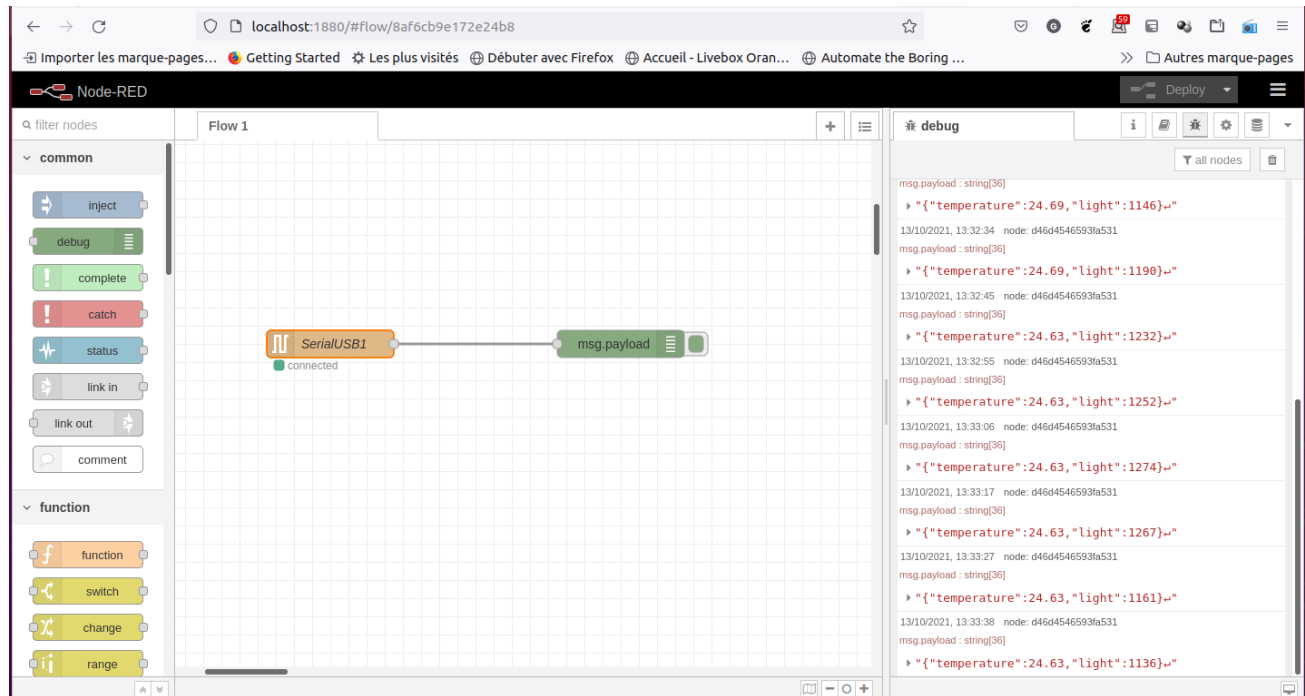
Lorsque le flow est créé, on le déploie (cf bouton rouge en haut à droite).

<http://noderedguide.com/node-red-lecture-2-building-your-first-flows-15/> :

- "Click the deploy button in the Node-RED window (top right). You'll see a pop-up saying the flow has been successfully deployed."
- "You will also notice that the blue dots on the nodes disappear, indicating there are no un-deployed changes."



Une fois les noeuds du flow correctement déployés, vous pouvez visualiser dans le debug (petit icône en haut à droite) les chaînes de caractères qui arrivent sur le port série :



Enfin presque, parce que si vous utilisez un NodeRed dans un container il faut avant permettre à votre container d'accéder à une ressource physique de la machine.

- ✓ <https://phoenixnap.com/kb/docker-privileged>
- ✓ <https://www.losant.com/blog/how-to-access-serial-devices-in-docker>

Après reboot, le lancement du container en background (-d) devient :

```
sudo docker run -d -it --privileged -p 1880:1880 -v /dev:/dev -v /home/menez/NodeRed/Data:/data --name mynodered nodered/node-red
f4b664eedaa3f989592d7a5190e19e099bb74e27d71f174d61c5ef47ea5b4ad8
```

**RMQ** : Pour faire tourner la manip, si vous utilisez un container, il faudra sans doute le stopper le temps d'uploader le firmware. Arduino IDE est plutôt rigide et exclusif sur la propriété du port ...

Le sketch ESP associé à ce flow est le suivant :

```
1 #include "sensors.h"
2 #include "OneWire.h"
3 #include "DallasTemperature.h"
4
5 /* —— Set timer —— */
6 unsigned long loop_period = 10L * 1000; /* => 10000ms : 10 s */
7
8 /* —— LED —— */
9 const int LEDpin = 19; // LED will use GPIO pin 19
10 // Ces variables permettent d'avoir une representation
11 // interne au programme du statut "electrique" de l'objet.
12 // Car on ne peut pas interroger une GPIO pour lui demander !
13 String LEDState = "off";
```

```

14
15 /* —— Light —— */
16 const int LightPin = A5; // Read analog input on ADC1_CHANNEL_5 (GPIO 33)
17
18 /* —— TEMP —— */
19 OneWire oneWire(23); // Pour utiliser une entite oneWire sur le port 23
20 DallasTemperature tempSensor(&oneWire); // Cette entite est utilisee
21     // par le capteur de
22     // temperature
23
24 /*—— Arduino IDE paradigm : setup+loop ——*/
25 void setup(){
26     Serial.begin(9600);
27     while (!Serial); // wait for a serial connection. Needed for native USB port only
28
29     // Initialize the LED
30     setup_led(LEDpin, OUTPUT, LOW);
31
32     // Init temperature sensor
33     tempSensor.begin();
34 }
35
36 void loop(){
37     String t = get_temperature(tempSensor);
38     String l = get_light(LightPin);
39
40     //affichage des données sous format JSON ... a l'arrache !
41     Serial.print("{");
42     Serial.print("\"temperature\":");
43     Serial.print(t);
44     Serial.print(",\"light\":");
45     Serial.print(l);
46     Serial.println("}");
47
48     //print_topic("temperature",t);
49     //print_topic("light",l);
50     delay(loop_period);
51 }

```

Avant de fermer/quitter la page, vous n'oubliez pas de sérialiser/export votre flow.

J'ai exporté mon flow dans "ma" library sous le nom `flow0.json` :

```

menez@mowgli ~
$ find . -name flow0.json
./node-red/lib/flows/flow0.json

menez@mowgli ~
$ cat ./node-red/lib/flows/flow0.json
[{"id":"5f566867.1d432","type":"tab","label":"Flow 1","disabled":false,"info":"","id":"ea79e5f3.a43b58","type":"serial in","z":"5f566867.1d432","name":"SerialUS80","serial":"5bb02696.4602","x":360,"y":800,"wires":[["285e105d.cb1c8"]]}, {"id":"285e105d.cb1c8","type":"debug","z":"5f566867.1d432","name":"","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","targetType":"msg","x":680,"y":800,"wires":[[]]}, {"id":"5bb02696.4602","type":"serial-port","z":"","serialport":"/dev/ttyUSB0","serialbaud":"9600","databits":"8","parity":"none","stopbits":"1","waitfor":"","dtr":"none","rts":"none","cts":"none","dsr":"none","newline":"\\n","bin":"false","out":"char","addchar":"","resettimeout":"10000"}]
menez@mowgli ~
$ █

```



## 6.4 Notion de Dashboard

Le dashboard ou "tableau de bord" en français est un "résumé graphique" d'informations "importantes"/"décisionnelles".

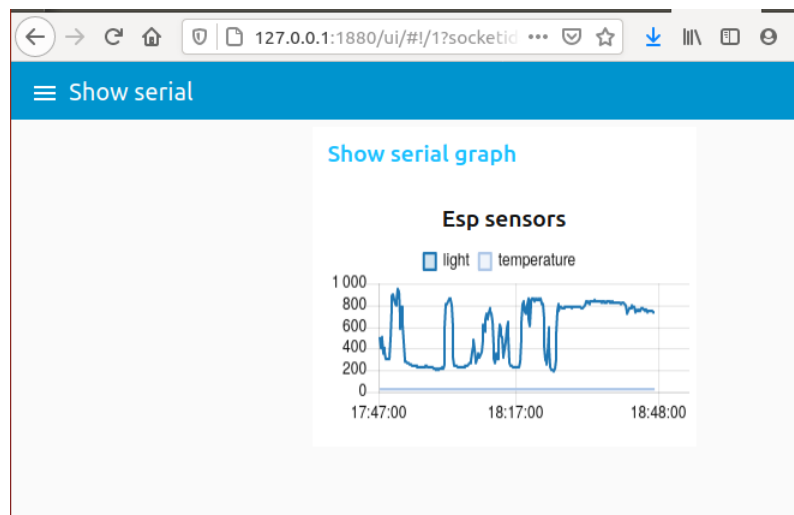


Dans notre cas, ces informations seront les données remontées par les capteurs.

Il existe des milliers d'outils (en local ou sur le Web) pour composer un dashboard mais puisqu'on est en train d'utiliser Node-Red ... on peut souligner qu'il existe une palette de noeuds dédiés à cela :

Vous installez la palette `node-red-dashboard`

21 noeuds sont alors disponibles et pour l'exemple qui suit nous utilisons le "graph".

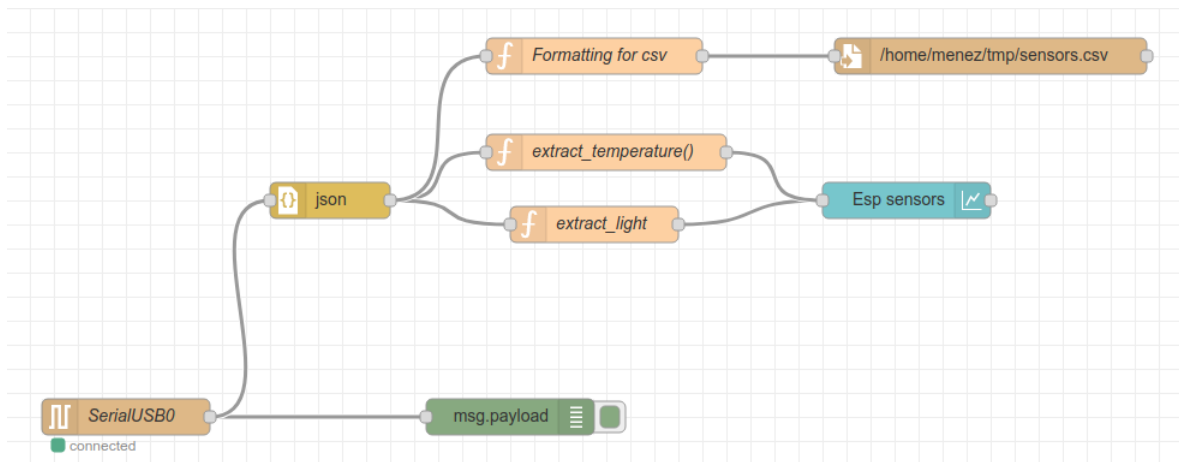


L'objectif de ce travail est de faire afficher les données des capteurs de l'ESP sur un graphique.

➤ Cet objet node-red va constituer implicitement une série à partir des valeurs successives qu'il reçoit.

➤ Vous remarquez l'url : "127.0.0.1 :1880/ui" qui correspond au dashboard du flow

Je vous donne sur le site (fichier "flow\_for\_graph.json") le flow suivant :



Mais cela serait bien si vous le reconstituez vous même : on le fera ensemble !

Le principe de ce flow est le suivant :

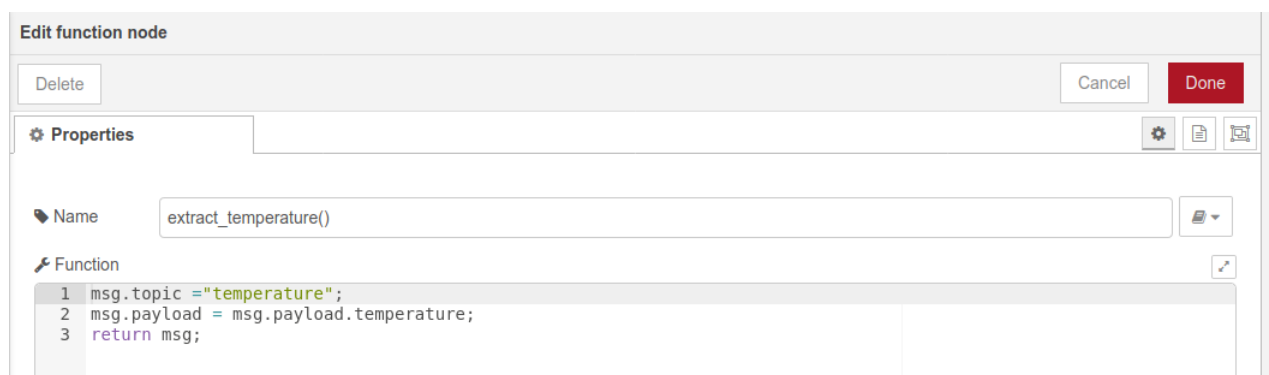
- Le noeud "SerialUSB0" produit une chaîne de caractères que l'on a syntaxiquement formulé en JSON.
- Le noeud "json" convertit cette chaîne en un objet json.  
De ce fait, la payload du message sortant est un objet JSON.
- Les fonctions (extract\_...()) qui suivent, accèdent aux champs du dictionnaire Json de la payload des messages transitant entre les noeuds.

Elles reformulent la payload pour qu'elle soit interprétable par le noeud graph.

- L'éditeur Javascript apparaît dès que vous double-cliquez sur le noeud !

Ainsi le message sortant envoyé au graphe doit contenir

- une payload qui est le champ temperature/light de la payload du message entrant
- et une référence à la courbe/série si il y en a plusieurs : Ici "temperature" ou "light".



- Le noeud de formatage est un exemple de transformation/exploitation du message et plus précisément de sa payload contenant un objet JSON.

- Le noeud graphe doit être introduit dans un group lui même appartenant à un tab.
  - Ceci se fait par l'interface de composition du flow.  
Cela participe à la définition du layout de la fenêtre du dashboard que vous obtenez par l'url :  
"127.0.0.1 :1880/ui".

## 7 cURL

From <https://fr.wikipedia.org/wiki/CURL> :

cURL (abréviation de client URL request library : " bibliothèque de requêtes aux URL pour les clients " ou see URL : " voir URL ") est une interface en ligne de commande, destinée à récupérer le contenu d'une ressource accessible par un réseau informatique.

### 7.1 Référence

<https://everything.curl.dev/http>  
<https://linuxize.com/post/curl-rest-api/>  
<https://www.it-connect.fr/curl-loutil-testeur-des-protocoles-divers/>

### 7.2 Structure générale d'une commande cURL

La ressource est désignée à l'aide d'une URL et doit être d'un type supporté.

```
curl [options] <url>
```

cURL supporte notamment les protocoles DICT, file, FTP, FTPS, Gopher, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTSP, SCP, SFTP, SMB, SMBs, SMTP, SMTPS, Telnet et TFTP.

Le logiciel permet de créer ou modifier une ressource (contrairement à wget), **il peut ainsi être utilisé en tant que client REST**.

### 7.3 HTTP : Voir le contenu de l'URL

Pour voir le contenu d'une URL :

```
curl http://httpbin.org/ip
{
  "origin": "134.59.131.45"
}
```

### 7.4 HTTP : request / response pour obtenir cette URL

On peut ainsi voir les headers des messages échangés :

```
curl -v http://httpbin.org/ip
* Trying 3.209.149.47:80...
* TCP_NODELAY set
* Connected to httpbin.org (3.209.149.47) port 80 (#0)
> GET /ip HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Wed, 06 Oct 2021 12:55:46 GMT
< Content-Type: application/json
```

```
< Content-Length: 32
< Connection: keep-alive
< Server: gunicorn/19.9.0
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
<
{
  "origin": "134.59.131.45"
}
* Connection #0 to host httpbin.org left intact
```

## 7.5 HTTP : requête GET avec paramètres

```
curl -v http://httpbin.org/get?led1="OFF"&led2="ON"
curl -v -X GET http://httpbin.org/get?led1="OFF"&led2="ON" -H "accept: application/json"
```

Dans la première, on se base sur le fonctionnement "par défaut" de curl.

Dans la deuxième requête, on spécifie l'utilisation d'une requête GET et on indique dans le header de cette requête que l'on souhaite une réponse en JSON. On obtient le même résultat ...

```
$ curl -v -X GET http://httpbin.org/get?led1="OFF"&led2="ON" -H "accept: application/json"
Note: Unnecessary use of -X or --request, GET is already inferred.
* Trying 54.159.86.231:80...
* TCP_NODELAY set
* Connected to httpbin.org (54.159.86.231) port 80 (#0)
> GET /get?led1=OFF&led2=ON HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.68.0
> accept: application/json
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Tue, 26 Oct 2021 13:42:51 GMT
< Content-Type: application/json
< Content-Length: 324
< Connection: keep-alive
< Server: gunicorn/19.9.0
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
<
{
  "args": {
    "led1": "OFF",
    "led2": "ON"
  },
  "headers": {
    "Accept": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.68.0",
```

```

    "X-Amzn-Trace-Id": "Root=1-617805db-7a514add77eaeb01549e5758"
  },
  "origin": "134.59.131.45",
  "url": "http://httpbin.org/get?led1=OFF&led2=ON"
}
* Connection #0 to host httpbin.org left intact

```

## 7.6 HTTP : requête POST sans paramètre

```

curl -v -X POST http://httpbin.org/post
* Trying 3.209.149.47:80...
* TCP_NODELAY set
* Connected to httpbin.org (3.209.149.47) port 80 (#0)
> POST /post HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Wed, 06 Oct 2021 13:03:10 GMT
< Content-Type: application/json
< Content-Length: 318
< Connection: keep-alive
< Server: gunicorn/19.9.0
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
<
{
  "args": {},
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.68.0",
    "X-Amzn-Trace-Id": "Root=1-615d9e8e-4a26394d221bb040307ec83e"
  },
  "json": null,
  "origin": "134.59.131.45",
  "url": "http://httpbin.org/post"
}
* Connection #0 to host httpbin.org left intact

```

## 7.7 HTTP : requête POST avec paramètres et body en JSON

```

curl -i -X POST -H "Content-Type: application/json" -d '{"key": "val"}' http://httpbin.org/post?led1="OFF"&led2="ON"

HTTP/1.1 200 OK

```

```
Date: Tue, 26 Oct 2021 13:52:29 GMT
Content-Type: application/json
Content-Length: 480
Connection: keep-alive
Server: gunicorn/19.9.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
```

```
{
  "args": {
    "led1": "OFF",
    "led2": "ON"
  },
  "data": "{\"key\":\"val\"}",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Content-Length": "13",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.68.0",
    "X-Amzn-Trace-Id": "Root=1-6178081d-684cc42c7795a7433fd8afbc"
  },
  "json": {
    "key": "val"
  },
  "origin": "134.59.131.45",
  "url": "http://httpbin.org/post?led1=OFF&led2=ON"
}
```

## 7.8 HTTP : requête POST avec paramètres et body en "URL encoded"

```
curl -d "param1=v1&param2=v2" -H "Content-Type: application/x-www-form-urlencoded" -X POST http://192.168.1.3:80/ip
```

## 8 TODO : Votre travail !

Le contexte ce travail est celui du dialogue en "HTTP" entre un objet (ESP) et une station que l'on pourrait qualifier de "supervision".

### 8.1 Montée en compétences

L'idée globale est de maîtriser les échanges d'informations (format texte/JSON) entre les composants de l'écosystème que vous êtes en train de créer :

- L'ESP,
- Le serveur,
- La machine de supervision/le navigateur.

Il y a pas de mal de codes à lire et à comprendre. Il y a aussi les outils à essayer. Je les énumère pour ne pas oublier :

- Http
- Json
- SPIFFS
- Node Red
- OTA ?

J'ai donné plus d'exemples de codes utilisant des requêtes GET/HTTP **MAIS** il faut remarquer que beaucoup d'API (OpenWeatherMap.org, IFTTT :

[https://www.frandroid.com/comment-faire/214864\\_ifttt-recettes-preferees,...](https://www.frandroid.com/comment-faire/214864_ifttt-recettes-preferees,...)) utilisent des requêtes POST/HTTP pour transmettre des informations.

**Il "faut" donc apprendre à gérer les requêtes POST :**

- Requêtes qu'il pourrait recevoir et qui contiendraient une payload (sans doute en Json),
- Requête qu'il pourrait émettre avec une payload (Html ou Json).

### 8.2 Cahier des charges

Votre travail fait intervenir 3 éléments dans le contexte de la gestion de régulation de température (idem TP1) :

- ① L'ESP
- ② Une page standard dans un navigateur qui jouera le rôle d'une machine de supervision.
- ③ Une page Node-Red dans un navigateur qui jouera le rôle du serveur de données au centre du réseau.  
Pour l'instant la persistance n'utilise pas de bases de données.

#### 8.2.1 L'ESP

L'ESP doit savoir répondre à :

- ① Une requête GET avec le path "/".

Sa réponse est une page Html qui donne un statut de l'ESP : identité de l'ESP, valeurs de ses capteurs, valeurs de ses seuils, ...



- Sa sémantique est au moins celle fournie dans le sujet (fichier "statut.html").

Si vous êtes à l'aise avec Html et CSS alors vous pouvez rendre les choses un peu plus "jolies" :

- Faites propre et simple (car ce n'est pas un cours sur Html!)
- Pour l'instant, pas de Javascript : **on garde les solutions plus dynamiques (Javascript) ou plus complexes (database) pour plus tard.**

- ② Une requête POST avec le path "/target" plus des paramètres qui désigneront la station "serveur de données" avec

- son "ip"
- son "port"
- et "sp" : la période d'échantillonnage souhaitée.  
Une valeur nulle **bloque** l'émission périodique.

A partir du moment où cette information est connue de l'ESP (et avec une valeur positive non nulle) ce dernier va émettre périodiquement ses données vers ce port **avec une requête POST**

- avec comme path la chaîne "/esp"
- comme valeur son adresse MAC
- et en body les valeurs données si dessous.

A la fin du TP, ce serveur de données devrait correspondre à un flow Node-Red qui saura faire l'affichage des données récupérées.

- ③ Une requête GET avec le path "/value" plus des paramètres qui spécifient les noms des valeurs attendues :

- ✓ temperature
- ✓ light
- ✓ cooler (led ou fan)
- ✓ heater (led ou radiateur)
- ✓ ip
- ✓ port
- ✓ sp
- ✓ light\_threshold (seuil jour/nuit)
- ✓ sbn (seuil bas nuit)
- ✓ shn (seuil haut nuit)
- ✓ sbj (seuil bas jour)
- ✓ shj (seuil haut jour)
- ✓ uptime
- ✓ ssid
- ✓ mac
- ✓ ip\_esp
- ✓ uptime
- ✓ where (qui indique où se trouve physiquement l'objet)

Une telle requête peut donc retourner plusieurs informations ...l'exprimer en JSON!? en donnant les valeurs sous forme textuelle.

Si un paramètre n'existe pas, la réponse pourrait être une réponse "404".

- ④ Une requête GET avec le path `"/set"` et en paramètres les noms/valeurs permettant de positionner les attributs de l'objet.

Si un paramètre n'existe pas, la réponse pourrait être une réponse "404".

### 8.2.2 Dashboard

Dans cette partie, vous devez créer un dashboard qui viendrait s'interfacer avec la requête POST de statut périodique mise en place à l'exercice précédent.

Vous pouvez utiliser l'outil de votre choix **à condition qu'il soit gratuit, accessible en illimité et sans inscription vous engageant pour la vie et la mort** :

- Node Red

- grafana

<https://grafana.com/oss/grafana/?pg=get&plcmt=selfmanaged-box1-cta2qui>

- ...

Plusieurs étapes/évolutions :

- ① Le minimum :

Vous recevez le statut de l'objet via une requête HTTP/POST périodique et vous devez le représenter graphiquement "au mieux".

J'ai mis des exemples de solutions potentielles (M1 2021) dans le fichier `"flows_all2021.json"`.

Elles donnent des exemples qui vous permettront de mieux comprendre et de démarrer.

Vous pouvez aussi regarder la documentation et les exemples/tutoriaux en ligne :

<https://nodered.org/docs/tutorials/second-flow>

Et aussi la bibliothèque :

<https://flows.nodered.org/>

- ② Le medium :

Les objets sont rarement "uniques" et la plupart du temps on doit gérer une "flotte d'objets" :

- Est ce que l'on peut avoir un dashboard qui gèrerait un ensemble d'objets ...comme autant de pièces d'un bâtiment dont on régulerait la température?

- ③ Le maximum :

C'est une partie un peu exploratoire sur laquelle je n'ai pas de solution déjà faite ... faut réfléchir et proposer :-) une solution, une ébauche , un chemin?

Les solutions minimales sont un "début" MAIS elles sont souvent très statiques et on peut très vite se demander :

- Est ce qu'un objet peut "s'inscrire" dynamiquement dans le dashboard?

Bref ...comment rendre ces dashboards dynamiques au niveau des objets/sources de données qu'ils interfacent?

- Et si tous les objets n'ont pas le même schéma JSON? ...là je délire!

## 8.3 ANNEXE 1 : POST

Je vous donne quelques liens qui peuvent aider dans la compréhension et la réalisation de votre travail. C'est un peu en vrac mais au niveau Master vous devez être capable de remettre de l'ordre ;-)

### 8.3.1 Quelques liens pour aider ...

- ✓ <https://stackoverflow.com/questions/20906066/making-href-anchor-tag-request-post-instead-of-get>
- ✓ <https://stackoverflow.com/questions/3915917/make-a-link-use-post-instead-of-get>
- ✓ <https://stackoverflow.com/questions/426310/how-do-you-post-data-with-a-link>
- ✓ <https://github.com/me-no-dev/ESPAsyncWebServer#get-post-and-file-parameters>
- ✓ <https://github.com/esp8266/Arduino/issues/1390>
- ✓ <https://randomnerdtutorials.com/esp32-http-get-post-arduino/>
- ✓ [https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/ESP32/HTTP/ESP32\\_HTTP\\_POST.ino](https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/blob/master/Projects/ESP32/HTTP/ESP32_HTTP_POST.ino)
- ✓ <https://techtutorialsx.com/2017/05/20/esp32-http-post-requests/>
- ✓ <https://randomnerdtutorials.com/esp8266-nodemcu-http-post-ifttt-thingspeak-arduino/>

### 8.3.2 Exemple de POST en HTML

Le code source de <http://httpbin.org/forms/post> fournit un exemple HTML d'utilisation d'une requête POST à partir de la balise "form" :

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  </head>
5  <body>
6  <!-- Example form from HTML5 spec http://www.w3.org/TR/html5/forms.html#writing-a-form's-user-interface -->
7  <form method="post" action="/post">
8  <p><label>Customer name: <input name="custname"></label></p>
9  <p><label>Telephone: <input type="tel" name="custtel"></label></p>
10 <p><label>E-mail address: <input type="email" name="custemail"></label></p>
11 <fieldset>
12 <legend> Pizza Size </legend>
13 <p><label> <input type="radio" name="size" value="small"> Small </label></p>
14 <p><label> <input type="radio" name="size" value="medium"> Medium </label></p>
15 <p><label> <input type="radio" name="size" value="large"> Large </label></p>
16 </fieldset>
17 <fieldset>
18 <legend> Pizza Toppings </legend>
19 <p><label> <input type="checkbox" name="topping" value="bacon"> Bacon </label></p>
20 <p><label> <input type="checkbox" name="topping" value="cheese"> Extra Cheese </label></p>
21 <p><label> <input type="checkbox" name="topping" value="onion"> Onion </label></p>
22 <p><label> <input type="checkbox" name="topping" value="mushroom"> Mushroom </label></p>
23 </fieldset>
24 <p><label>Preferred delivery time: <input type="time" min="11:00" max="21:00" step="900" name="delivery"></label></p>
25 <p><label>Delivery instructions: <textarea name="comments"></textarea></label></p>
26 <p><button>Submit order</button></p>
27 </form>
28 </body>
29 </html>

```