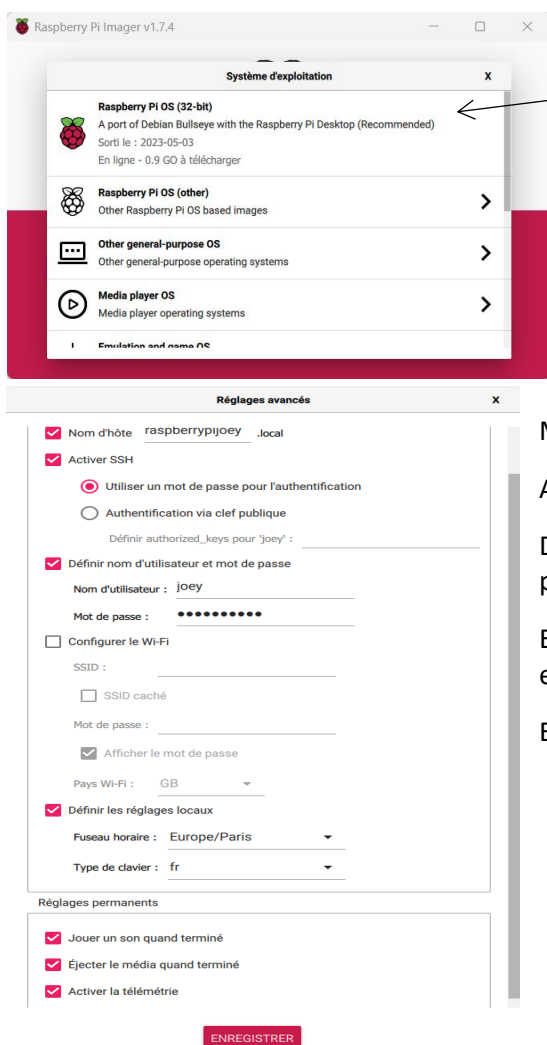


- 1. Le réseau local Wifi sera mis en place sur un RPI avec hostapd et permettra à l'ESP32 d'accéder à internet (routage+firewall). (Re)Installer entièrement une nouvelle image système sur le RPI à partir de zéro et bien expliquer la procédure d'installation et de paramétrage du RPI.**

Installation de l'OS sur le RPI :

Installer Raspberry Pi Imager afin de copier une image système sur une carte microSD pour un Raspberry Pi.

Configuration de l'OS avant l'écriture sur la carte :



On choisit le premier OS

Modifier le nom d'hôte (rajouter 'joey')

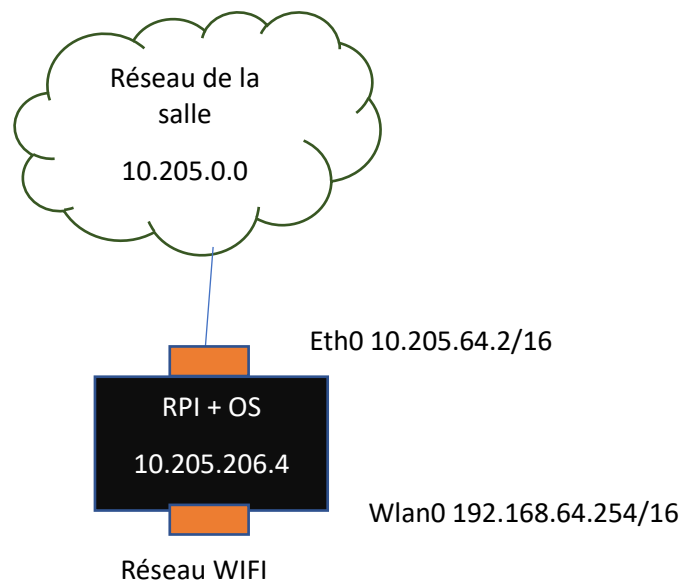
Activer SSH

Définir un nom d'utilisateur avec un mot de passe pour s'identifier

Ensuite choisir le fuseau horaire Paris et mettre le clavier en «fr»

Ensuite enregistrer et écrire sur la carte

Schéma du réseau :



- **Installation des paquets :**

```
sudo apt update
```

```
sudo apt install hostapd dnsmasq
```

- **Configuration du fichier interfaces networking :**

```
auto eth0
iface eth0 inet static
    address 10.205.64.2/16
    gateway 10.205.255.254

auto wlan0
iface wlan0 inet static
    address 192.168.64.254/16
    bridge-ports none
    bridge-stp off
    bridge-fd 0
```

- **Configuration du DHCP /etc/dhcpd.conf :**

```
interface eth0
```

```
static ip_address=10.205.64.2/16
```

```
static routers=10.205.255.254/16
```

```
static domain_name_servers=8.8.8.8 8.8.1.1
```

```
interface wlan0
static ip_address=192.68.64.254/24
static domain_name_servers=8.8.8.8 8.8.1.1
```

- **Configuration de hostapd :**

```
interface=wlan0
driver=nl80211
ssid=wifijoey
hw_mode=g
channel=11
auth_algs=1
beacon_int=100
dtim_period=2
max_num_sta=255
rts_threshold=2347
fragm_threshold=2346
wpa=2
wpa_passphrase=joeyjoey
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP
rsn_pairwise=CCMP
```

- **Configuration de dnsmasq** (*sudo nano /etc/dnsmasq.conf*) :

```
interface=wlan0
dhcp-range=192.168.64.1,192.168.64.250,255.255.255.0,24h
```

- **Activer le routage IP** (*sudo nano /etc/sysctl.conf*) :

```
net.ipv4.ip_forward=1
```

- **Configuration des règles de pare-feu :**

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

```
sudo nano /etc/rc.local
```

```
→ iptables-restore < /etc/iptables.ipv4.nat
```

- **Démarrer les services :**

```
sudo systemctl unmask hostapd
```

```
sudo systemctl enable hostapd
```

```
sudo systemctl start hostapd
```

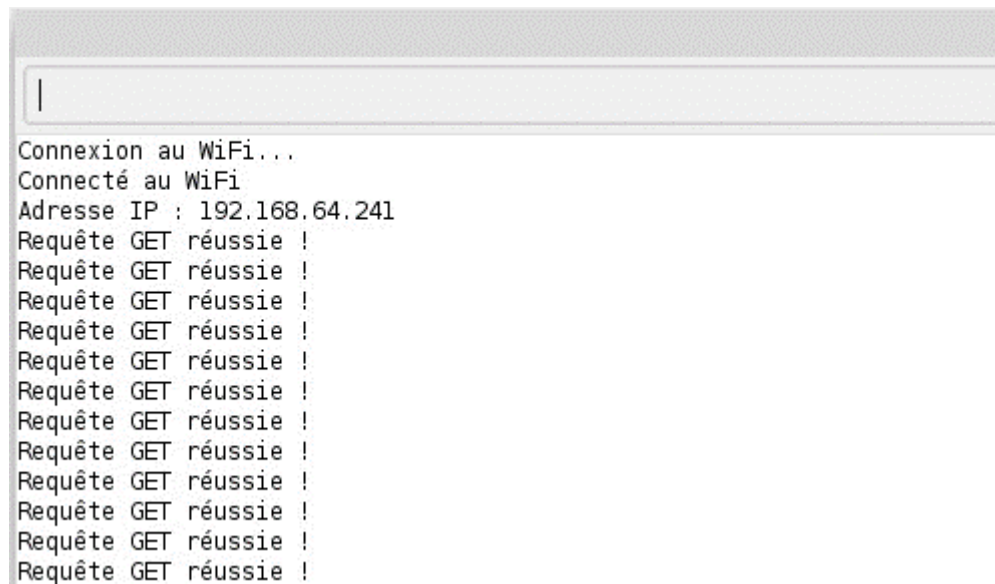
```
sudo systemctl enable dnsmasq
```

```
sudo systemctl start dnsmasq
```

- **Redémarrer le Raspberry Pi !**

Pour l'ESP32 :

J'ai effectué un petit programme C++ pour vérifier la connexion internet fournie par le Raspberry pi à mon ESP32 (* voir fichier connexion-wifi-esp32.ino). Ce programme connecte l'ESP32 à mon réseau wifi « wifijoey » et effectue une requête GET pour vérifier la connexion internet.



```
Connexion au WiFi...  
Connecté au WiFi  
Adresse IP : 192.168.64.241  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !  
Requête GET réussie !
```

2. Utiliser le système de fichier SPIFFS sur l'ESP32

Tout d'abord, il faut installer le « ESP32 Filesystem Uploader » dans l'IDE Arduino. Pour cela, il faut télécharger le fichier ESP32FS-1.0.zip sur le github suivant <https://github.com/me-no-dev/arduino-esp32fs-plugin/releases/>.

Puis dans le dossier « Arduino » il faut créer un dossier « tools » puis copier le dossier **ESP32FS** décompressé dans ce répertoire.

Structure : ~/joeygalligani/Arduino/tools/ESP32FS/tool/esp32fs.jar

Ensuite, nous allons déposer nos fichiers HTML et CSS dans un dossier /data dans le dossier /Arduino.

Maintenant, il faut aller dans l'IDE Arduino puis appuyer sur l'onglet « Outils » puis cliquer sur « ESP32 Sketch Data Upload ».

Ces étapes permettent de téléverser les fichiers contenus dans /Arduino/data dans l'ESP32. Si nous modifions les fichiers HTML et CSS il suffira de recliquer sur « ESP32 Sketch Data Upload » pour téléverser les fichiers.

Ensuite, pour utiliser les fichiers dans une infrastructure comme pour la SAE il faut faire un programme C++ en utilisant SPIFFS.

Initialisation de SPIFFS :

```
if (!SPIFFS.begin(true)) {  
    Serial.println("An Error has occurred while mounting SPIFFS");  
    return;  
}
```

Création du server Web sur le port 80 :

```
AsyncWebServer server(80);
```

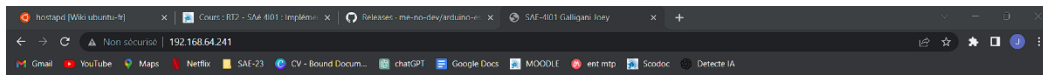
Ici nous indiquons le chemin pour charger le fichier style.css :

```
server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest* request) {  
    request->send(SPIFFS, "/style.css", "text/css");  
});
```

Ici nous indiquons le chemin pour charger le fichier de base du site index.html :

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest* request) {  
    request->send(SPIFFS, "/index.html", String(), false, processor);  
});
```

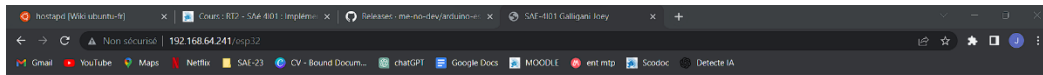
Ensuite, de la même manière j'ai indiqué le chemin de deux autres pages, une destinée aux capteurs Sigfox et l'autre aux capteurs DHT11. J'ai donc aussi rajouté deux fichiers (dht11.html et sigfox.html) dans le dossier data puis dans le C++ ajouté les deux routes pour charger ces fichiers.



SAE-4I01 Galligani Joey

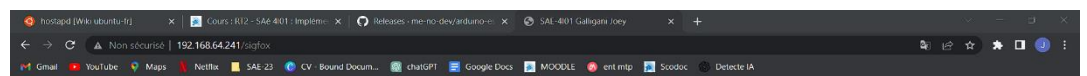
ESP32 - Capteur DHT11

SigFox - Capteur Sens'it



SAE-4I01 Galligani Joey

DHT11



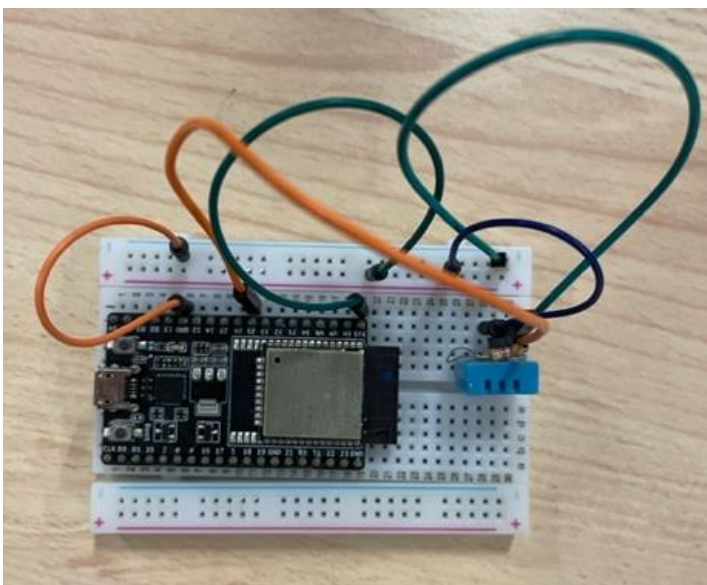
SAE-4I01 Galligani Joey

SigFox

3. Mettre en œuvre les technologies WEB : HTML/CSS/JavaScript/JSON

Récupération des données du DHT11

Sur l'ESP32 il faut mettre en place un circuit avec le DHT11 :



Sur l'ESP32 il faut téléverser un code Arduino qui charge les données du DHT11 (Température et Humidité) avec SPIFFS dans un fichier Json nommé data.json :

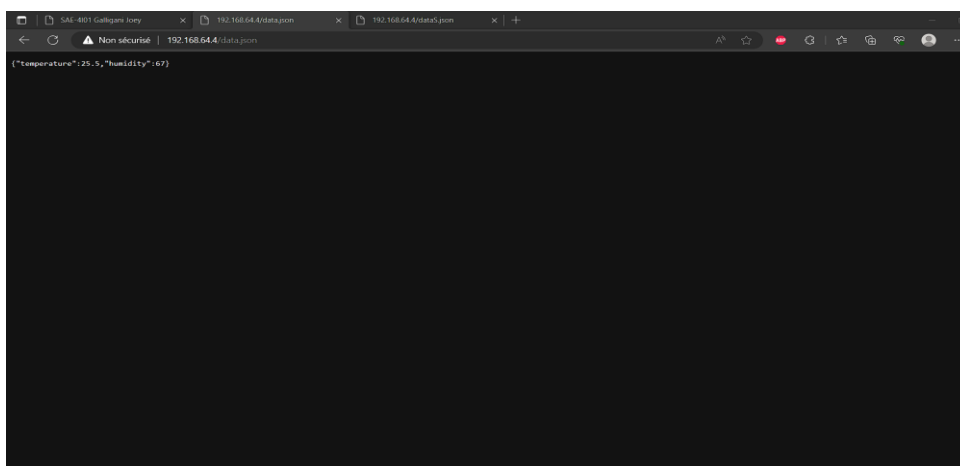
```
server.on("/data.json", HTTP_GET,
[] (AsyncWebServerRequest *request) {
    request->send(SPIFFS, "/data.json",
String(), false);
});
```

Cela permet d'indiquer le chemin pour charger data.json avec SPIFFS.

```
Void loop() {  
    // Lecture des valeurs du capteur  
    float temperature = dht.readTemperature() ;  
    float humidity = dht.readHumidity() ;  
  
    // Création de l'objet JSON  
    StaticJsonDocument<200> doc ;  
    doc[« temperature »] = temperature ;  
    doc[« humidity »] = humidity ;  
  
    // Convertir l'objet JSON en chaîne  
    String jsonStr ;  
    serializeJson(doc, jsonStr) ;  
  
    // Afficher la chaîne JSON dans la console série  
    Serial.println(jsonStr) ;  
  
    // Ouverture du fichier JSON en écriture  
    File file = SPIFFS.open(« /data.json », « w ») ;  
    if ( !file) {  
        Serial.println(« Erreur lors de l'ouverture du fichier JSON ») ;  
        return ;  
    }  
}
```

Cela permet de récupérer les données et de les insérer dans le data.json.

Avec l'URL <http://192.168.64.4/data.json> nous avons :



Nous voyons bien les données de température et d'humidité au format Json.

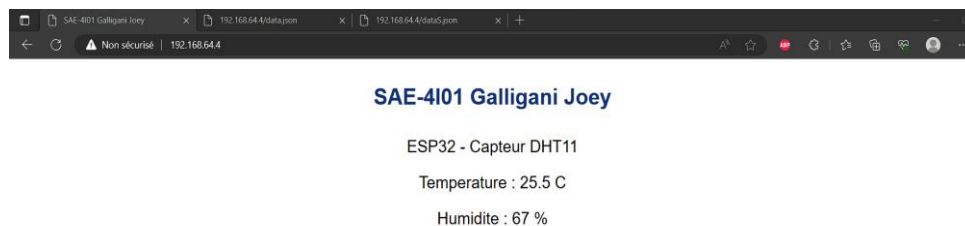
Ce qui va permettre d'afficher c'est donner de manière interprétée sur la page Web est un script JavaScript qui va permettre de parser et de récupérer uniquement les valeurs :

```
<script>
  // Fonction pour mettre à jour les données du capteur
  function updateSensorData() {
    $.get(« /data.json », function(data) {
      // Mettre à jour les valeurs de température et d'humidité sur la page
      $(« #temperature »).text(« Temperature : « + data.temperature +
« C ») ;
      $(« #humidity »).text(« Humidite : « + data.humidity + « % ») ;
    });
  }
</script>
```

Il suffit ensuite d'afficher cela avec l'HTML dans le body :

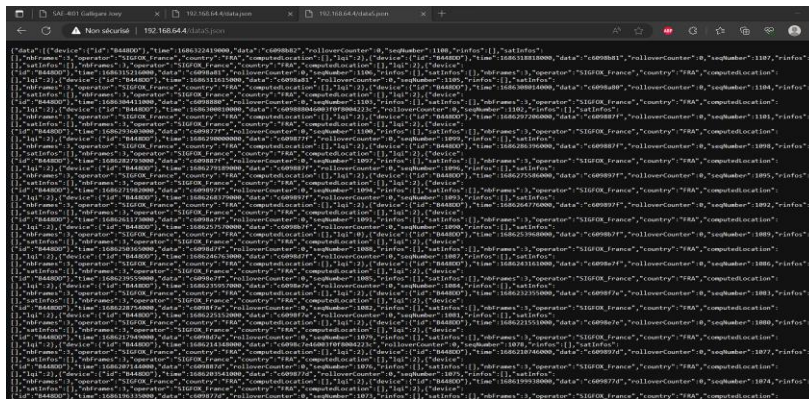
```
<p>ESP32 – Capteur DHT11</p>
<p id= »temperature »>Temperature : </p>
<p id= »humidity »>Humidite : </p><br><br><br>
```

Finalement sur ma page Web nous allons avoir la température et l'humidité que le DHT11 capte en temps réel :



Maintenant je souhaite intégrer à ma page web les données de température d'humidité et de batterie des capteurs Sens'it Sigfox de l'IUT.

Pour cela nous allons récupérer les données des capteurs à l'aide de la commande « curl -u 6334446bbdcdbd20e4daf92b4 :3806°9d6cead4f45803df27eee00994a <https://api.sigfox.com/v2/devices/B448DD/messages> » qui doit être exécuter par l'esp32 afin de stocker le résultat dans <http://192.168.64.4/dataS.json>. Nous obtenons cela :



Dans le code Arduino j'ai tout d'abord déclaré le chemin pour charger dataS.json :

```
server.on(« /dataS.json », HTTP_GET, [] (AsyncWebServerRequest *request) {
  request->send(SPIFFS, « /dataS.json », String(), false) ;
});
```

Ensuite, j'ai mis en place une architecture de manière que mon ESP32 effectue une requête curl sur l'API Sigfox et pousse le résultat dans un fichier dataS.json. J'ai alors mis en place un script JS qui récupère le résultat du curl dans dataS.json puis le parse et isole les données qui correspondent à un device choisi (j'ai choisi le). Le script va ensuite convertir cette data qui est en Hexadécimal en binaire et par la suite découper cette chaine de binaire en 3 parties. La première partie correspond à la tension dans la batterie, la deuxième à la température et la dernière à l'humidité (données captées à l'IUT). Ensuite il faut interpréter en appliquant des fonctions mathématiques pour rendre chaque donnée interprétable.

Code C++ :

```
// Effectuer la requête curl et stocker le résultat dans le fichier data.json

HTTPClient http ;
http.begin(« https://api.sigfox.com/v2/devices/B448DD/messages ») ;
http.setAuthorization(« 6334446bbdcbd20e4daf92b4 »,
« 3806°9d6cead4f45803df27eee00994a ») ;
int httpResponseCode = http.GET() ;

if (httpResponseCode == HTTP_CODE_OK) {
  String payload = http.getString() ;

  // Ouverture du fichier JSON en écriture
  File file = SPIFFS.open(« /dataS.json », « w ») ;
  if ( !file) {
    Serial.println(« Erreur lors de l'ouverture du fichier JSON ») ;
    return ;
  }
}
```

```

// Écrire la réponse dans le fichier
file.println(payload) ;

// Fermer le fichier
file.close() ;

Serial.println(« Résultat de la requête curl stocké dans le fichier data.json ») ;
} else {
  Serial.print(« Erreur lors de la requête curl. Code d'erreur : » ) ;
  Serial.println(httpResponseCode) ;
}

http.end() ;

delay(2000) ; // Attendre 2 secondes entre chaque lecture

```

Exemple de trame de data de capteurs Sens'it Sigfox :

Payload	ee098b6c																															
Hexidecimal values	0xee								0x09								0x8b								0x6c							
Binary values	0b11101110								0b00001001								0b10001011								0b01101100							
Individual bits	1	1	1	0	1	1	1	0	0	0	0	1	0	0	1	1	0	0	0	1	0	1	1	0	1	1	0	0	0	1	0	0
Grouping for Sens'it 3's Temperature and Humidity mode	Battery level				Reserved				Active mode				Flag		Mode-dependent data								Mode-dependent data									

Javascript :

```

$(document).ready(function() {
  // Fonction pour décoder le payload Sens'it
  function decodeSensitPayload(payload) {
    // Conversion du payload hexadécimal en binaire
    const binaryPayload = hexToBinary(payload) ;
    //console.log(hexToBinary(payload)) ;
    // Extraction des valeurs du niveau de batterie, de la température et de l'humidité
    const batteryLevel = (parseInt(binaryPayload.substr(0, 5), 2) * 0.05) + 2.7 ;
    //console.log(binaryPayload.substr(0, 5)) ;
    const humidity = parseInt(binaryPayload.substr(24, 8), 2) / 2 ;
    //console.log(binaryPayload.substr(24, 8)) ;
    const temperature = (parseInt(binaryPayload.substr(14, 10), 2) - 200) / 8 ;
    //console.log(binaryPayload.substr(14, 10))

    // Mettre à jour les valeurs décodées sur la page
    $('#temperature-data').text(« Temperature : « + Math.round(temperature) + « C » ») ;
  }

```

```
$(« #humidity-data »).text(« Humidite : « + humidity + « % ») ;  
$(« #battery-level »).text(« Niveau de batterie : « + (Math.round(batteryLevel * 10) / 10) +  
« Volts ») ;  
}
```

```
// Fonction pour convertir un nombre hexadécimal en binaire
```

```
function hexToBinary(payload) {  
  return parseInt(payload, 16).toString(2) ;  
}
```

```
// Fonction pour récupérer les données depuis le fichier data.json
```

```
function fetchData() {  
  $.getJSON(« dataS.json », function(data) {  
    // Sélectionnez ici l'appareil souhaité  
    var deviceId = « B448DD » ;
```

```
// Recherche de l'appareil dans les données
```

```
var deviceData = data.data.find(function(device) {  
  return device.device.id === deviceId ;  
});
```

```
// Affichage de la valeur de data
```

```
if (deviceData) {  
  const payload = deviceData.data ;  
  decodeSensitPayload(payload) ;  
} else {  
  $(« #data-container »).text(« Device not found ») ;  
}  
});  
}
```

```
// Appel initial de la fonction fetchData  
fetchData() ;
```

```
// Actualisation des données toutes les 2 secondes  
setInterval(fetchData, 2000) ;
```

```
// Mettre à jour les données du capteur toutes les 2 secondes  
setInterval(updateSensorData, 2000) ;
```

Nous obtenons sur 192.168.64.4 :

18:57

30

ESP32 - Capteur DHT11

Temperature : 25 C

Humidite : 63 %

Capteur Sigfox IUT Beziers

Device : B448DD

Temperature: 25 C

Humidite: 65%

Niveau de batterie: 3.9 Volts

- 4. On installera sur le RPI un serveur DNS permettant d'accéder au site WEB de l'ESP32 avec le nom *rt.sae401.bz* aussi bien de l'extérieur (*eth0*) que de l'intérieur (*wlan0*) du réseau du RPI (voir les vues DNS).**

Pour le coté wlan0 :

Il suffit de configurer le `dnsmasq.conf` sur le Raspberry Pi afin d'associer pour moi `joey.sae401.bz` à l'ip de l'esp32 soit 192.168.64.4 ce qui donne à rajouter à la fin du fichier de conf :

```
address=/joey.sae401.bz/192.168.64.4
```

En nous connectant donc au réseaux wifi « saeiomjoey » et en entrant dans le navigateur « `joey.sae401.bz` » nous arrivons sur la page web :

18:57



← wifi : saeiomjoey

ESP32 - Capteur DHT11

Temperature : 25 C

Humidite : 63 %

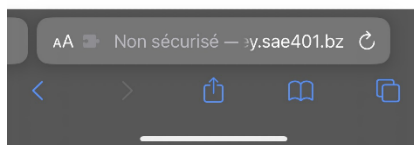
Capteur Sigfox IUT Beziers

Device : B448DD

Temperature: 25 C

Humidite: 65%

Niveau de batterie: 3.9 Volts



← joey.sae401.bz

Pour la partie DNS avec eth0 :

Nous allons rediriger les paquets qui arrive sur eth0 au port 80 sur le l'ip du site web et les paquets sortant du site web vers wlan0 sur port 80 sont redirigé vers eth0.

```
sudo iptables -t nat -A PREROUTING -i eth -p tcp --dport 80 -j DNAT --to-destination 192.168.64.4
sudo iptables -t nat -A POSTROUTING -o wlan -p tcp --dport 80 -j SNAT --to-source 10.205.64.2
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```



ESP32 - Capteur DHT11

Temperature : 26 C

Humidite : 61 %

Depuis un PC de la salle
B205 je peux aller sur la
page web en entrant l'IP
10.205.64.2 :

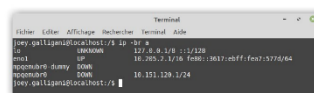
Capteur Sigfox IUT Beziers

Device : B448DD

Temperature: 24 C

Humidite: 64%

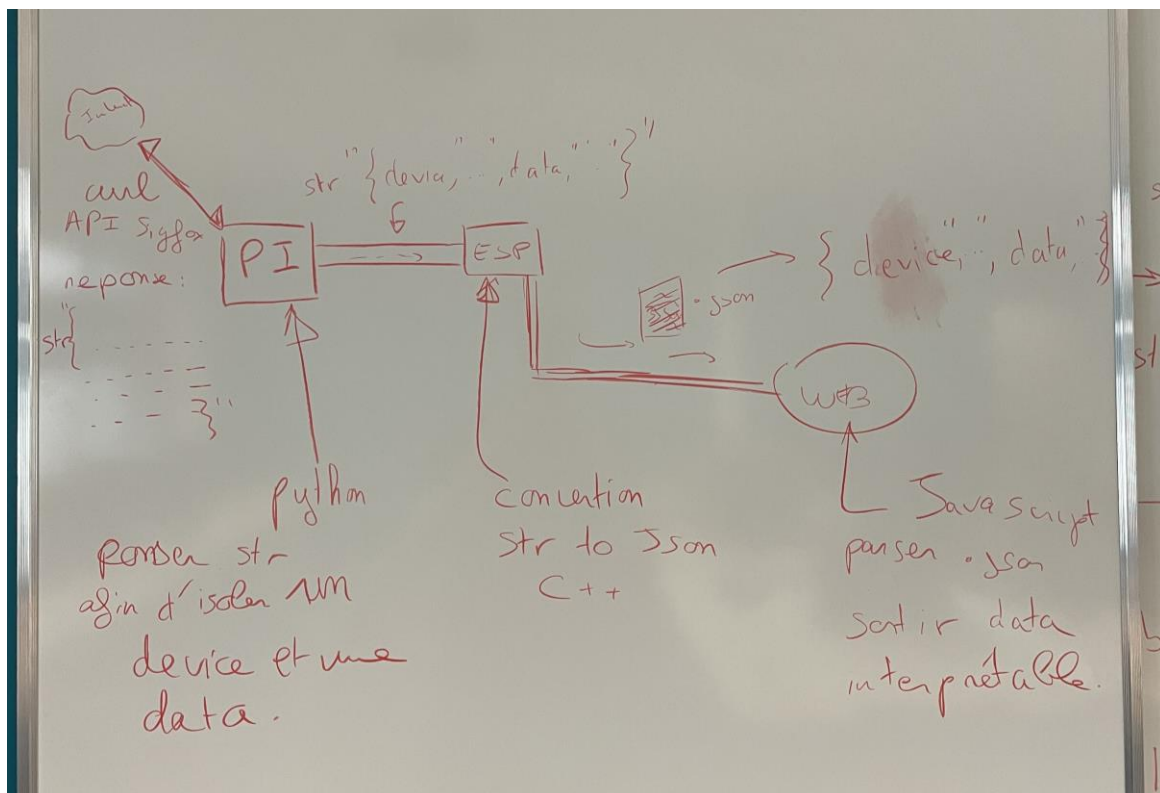
Niveau de batterie: 3.9 Volts



Il suffit maintenant de définir sur la machine client que le DNS à prendre est celle du RPI

5. Ajouter sur le RPI un serveur BLE GATT (bluetooth) permettant de fournir plusieurs valeurs de comptage en temps réel (ouverture de porte, présence d'équipement BT à proximité, nombre de personnes dans une pièce, etc...) et sur l'ESP32 la partie client GATT bluetooth permettant d'afficher la(es) valeur(s) sur la page WEB.

Schéma de l'infrastructure BLE :



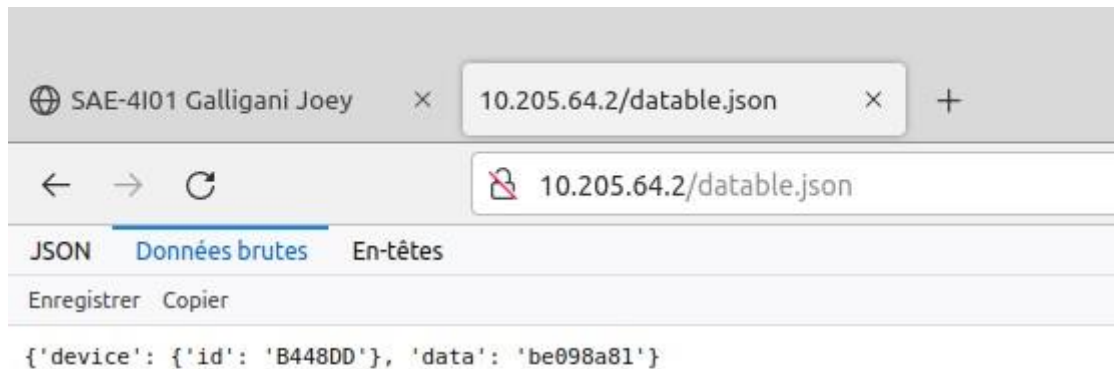
Sur le RPI j'ai tout d'abord effectué un : `git clone https://github.com/douglas6/cputemp.git`

Ensuite dans le fichier `/cputemp/cputemp.py` j'ai modifié le programme python en remplaçant les UUID du service et de la caractéristique avec deux UUID générer sur internet. Ensuite dans ce programme la fonction `get_temperature(self)` récupère la température du cpu et la retourne. C'est donc cette fonction qu'il faut modifier afin d'aller chercher les données que je veux. J'ai donc codé cette fonction de manière qu'elle effectue un Curl sur l'API Sigfox et qu'elle retourne une chaîne de caractère qui a le format json en étant codée en bit en affichant l'id et la data d'un device choisi. Il reste plus qu'à lancer le programme avec `python3`.

```
joey@raspberrypi:~/cputemp $ nano cputemp.py
joey@raspberrypi:~/cputemp $ python3 cputemp.py
GATT application registered
GATT advertisement registered
```

Ensuite pour le client BLE il suffit de modifier l'exemple disponible sur l'IDE Arduino à indiquer le chemin pour /databale.json avec spiffs puis créer une fonction pour écrire dans le fichier. Pour le programme C++ en détail voir WEBserver-joey.ino.

Nous obtenons bien ce que nous voulons :



Ensuite il suffit de mettre en place un script JavaScript qui va parser et afficher dans la page web les données interprétées. Pour avoir au final :

