

Rapport : Réseau d'intégration Web IoT

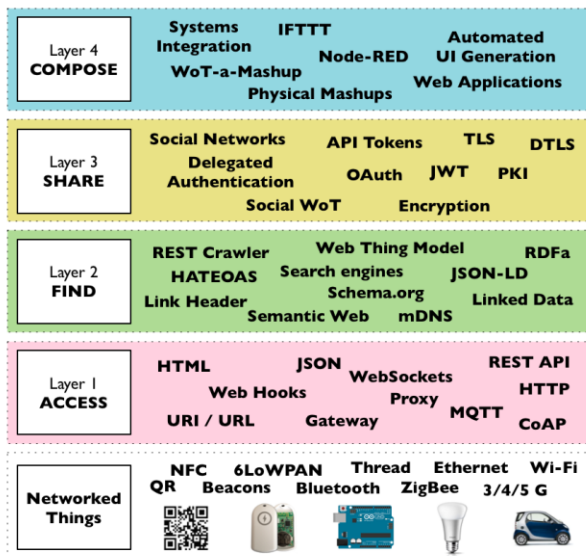
Préparer par : Silvia Garstea

OBJECTIF

On se propose de mettre en place un réseau web IoT composé d'objets à base de Raspberry Pi et ESP8266 (ESP32).

Le Web of Things est un raffinement de l'Internet des Objets en intégrant des objets intelligents non seulement dans Internet (réseau), mais dans l'architecture Web (application). (<https://webofthings.org/2017/04/08/what-is-the-web-of-things/>)

Tout comme l'architecture en couches OSI organise les nombreux protocoles et standards d'Internet, l'architecture WoT est une tentative de structurer la galaxie de protocoles et d'outils Web en un cadre utile pour connecter n'importe quel périphérique ou objet au Web. La pile de l'architecture WoT n'est pas composée de couches au sens strict, mais plutôt de niveaux qui ajoutent des fonctionnalités supplémentaires, comme le montre la figure ci-dessous. Chaque couche permet d'intégrer les objets au Web encore plus intimement et donc de rendre ces dispositifs plus accessibles pour les applications et les humains!

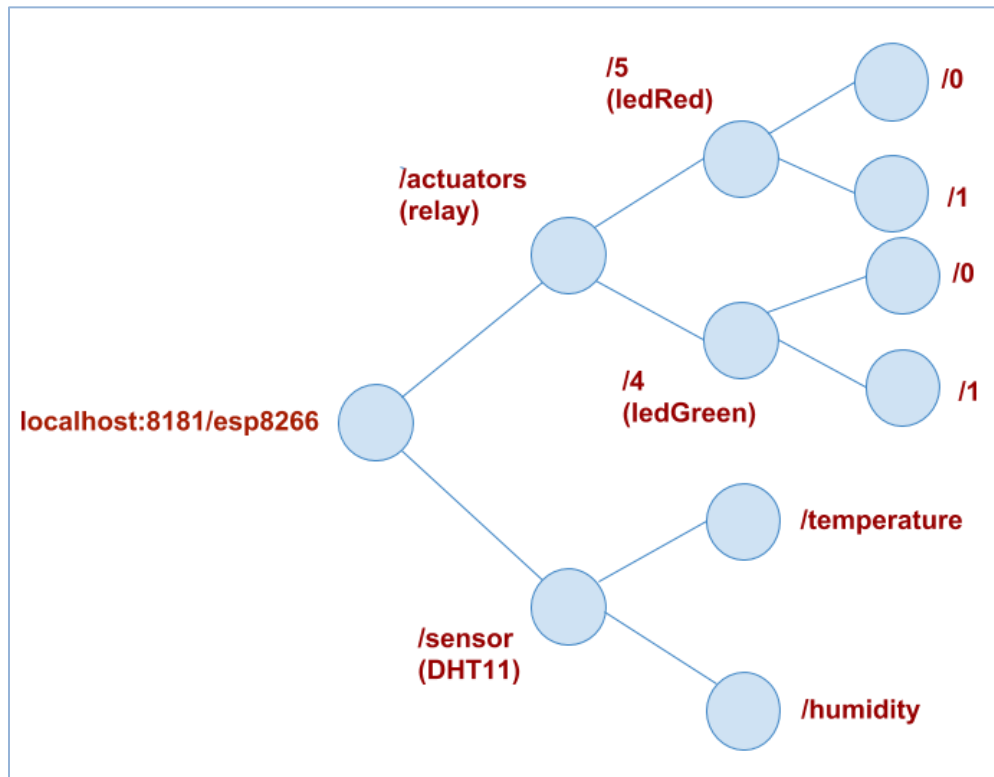


Source: Building the Web of Things; book.webofthings.io
Creative Commons Attribution 4.0

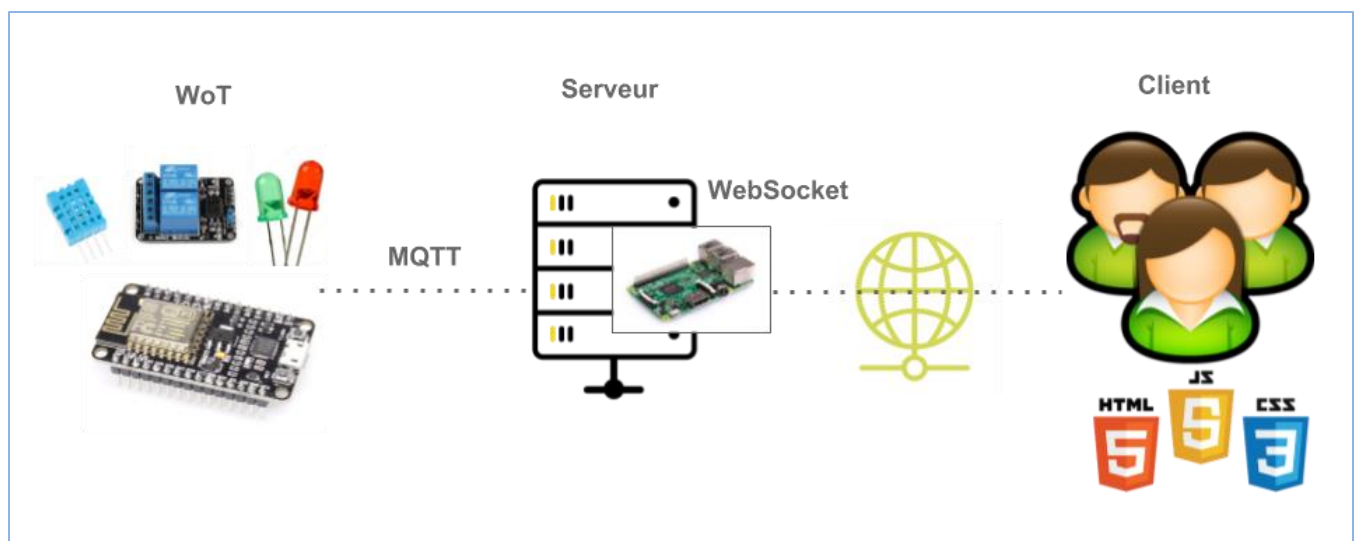
On va se concentrer plus sur le couche ACCESS. Cette couche est responsable de transformer n'importe quel objet en un objet Web qui peut être interagi en utilisant des requêtes HTTP comme n'importe quelle autre ressource sur le Web. En d'autres termes, un Web Thing est une API REST qui permet d'interagir avec quelque objet dans le monde réel, comme ouvrir une porte ou lire un capteur de température situé à travers la planète. Pour que ça soit en temps réel et / ou axés sur les événements, on va utiliser WebSocket comme protocol.

DÉMARCHES

1. La structure URI :



2. Le schéma du réseau :



3. Les dédales

Serveur en Python :

Prérequis :

- ✓ **Mosquitto**
- ✓ **Pour le serveur Web on va utiliser un microframework Python appelé *Flask***

```
pi@raspberrypi ~ $ sudo apt-get install python-pip python-flask
git-core

pi@raspberrypi ~ $ sudo pip install flask

pi@raspberrypi ~ $ sudo pip install paho-mqtt

pi@raspberrypi ~ $ sudo pip install flask-socketio
```

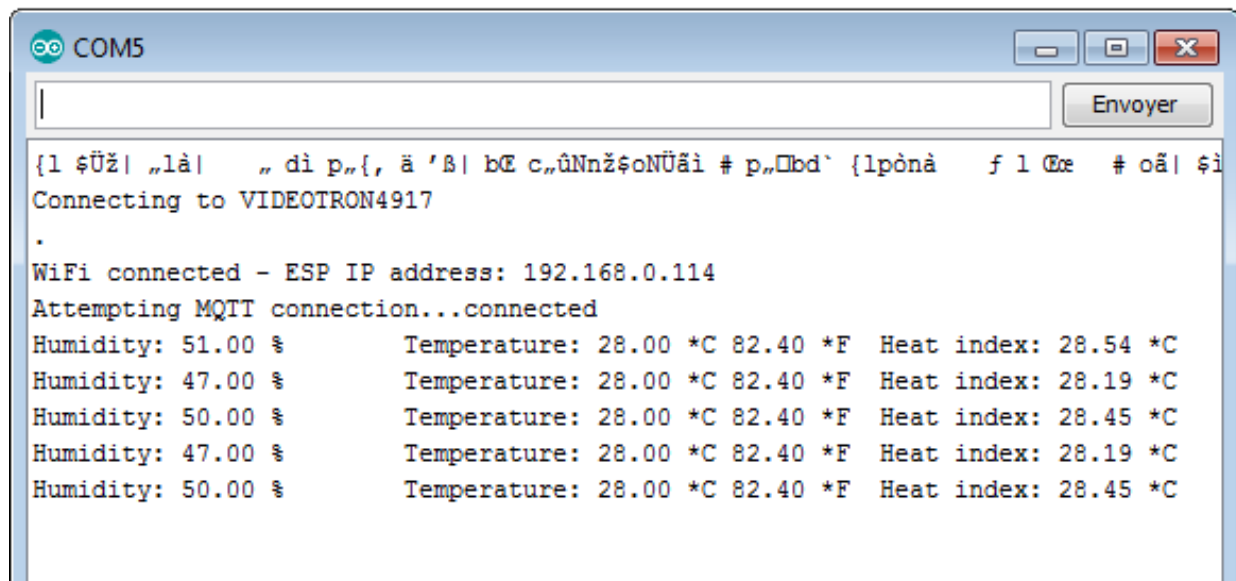
Code du serveur voir ANNEXE1.

Creation de la page HTML:

Flask utilise un moteur de template appelé *Jinja2* qu'est utilisé pour envoyer des données dynamiques de script Python à HTML.

Code de la page HTML voir ANNEXE 1.

Programmation de l'ESP8266(code voir ANNEXE1)



```
COM5
Connecting to VIDEOTRON4917
.
WiFi connected - ESP IP address: 192.168.0.114
Attempting MQTT connection...connected
Humidity: 51.00 %      Temperature: 28.00 *C 82.40 *F  Heat index: 28.54 *C
Humidity: 47.00 %      Temperature: 28.00 *C 82.40 *F  Heat index: 28.19 *C
Humidity: 50.00 %      Temperature: 28.00 *C 82.40 *F  Heat index: 28.45 *C
Humidity: 47.00 %      Temperature: 28.00 *C 82.40 *F  Heat index: 28.19 *C
Humidity: 50.00 %      Temperature: 28.00 *C 82.40 *F  Heat index: 28.45 *C
```

Lancement du serveur Web:

```
pi @ raspberrypi: ~ / serveur-web $ sudo python app.py
```

```
pi@raspberrypi: ~/web-server
pi@raspberrypi:~/web-server $ sudo python app.py
WebSocket transport not available. Install eventlet or gevent and gevent-websocket for improved performance.
Connected with result code 0
* Running on http://0.0.0.0:8181/ (Press CTRL+C to quit)
* Restarting with stat
WebSocket transport not available. Install eventlet or gevent and gevent-websocket for improved performance.
Connected with result code 0
* Debugger is active!
* Debugger pin code: 146-326-049
192.168.0.100 - - [19/Mar/2018 00:33:20] "GET /socket.io/?EIO=3&transport=polling&t=1521419600868-27 HTTP/1.1" 200 -
received json data here: {'data': u'I'm connected!'}
192.168.0.100 - - [19/Mar/2018 00:33:20] "POST /socket.io/?EIO=3&transport=polling&t=1521419601117-28&sid=3bae562803c943cb91a42cc9c7841600 HTTP/1.1" 200 -
Received message ' 28.99' on topic '/esp8266/DHT11/temperature' with QoS 0
temperature update
Received message ' 28.99' on topic '/esp8266/DHT11/temperature' with QoS 0
temperature update
192.168.0.100 - - [19/Mar/2018 00:33:25] "GET /socket.io/?EIO=3&transport=polling&t=1521419601117-29&sid=3bae562803c943cb91a42cc9c7841600 HTTP/1.1" 200 -
Received message ' 44.00' on topic '/esp8266/DHT11/humidity' with QoS 0
Received message ' 44.00' on topic '/esp8266/DHT11/humidity' with QoS 0
humidity update
humidity update
```

```
pi@raspberrypi: ~/web-server
humidity update
humidity update
192.168.0.100 - - [19/Mar/2018 00:36:15] "GET /socket.io/?EIO=3&transport=polling&t=1521419766316-70&sid=3bae562803c943cb91a42cc9c7841600 HTTP/1.1" 200 -
192.168.0.100 - - [19/Mar/2018 00:36:15] "POST /socket.io/?EIO=3&transport=polling&t=1521419776378-72&sid=3bae562803c943cb91a42cc9c7841600 HTTP/1.1" 200 -
192.168.0.100 - - [19/Mar/2018 00:36:15] "GET /socket.io/?EIO=3&transport=polling&t=1521419776270-71&sid=3bae562803c943cb91a42cc9c7841600 HTTP/1.1" 200 -
Received message ' 28.03' on topic '/esp8266/DHT11/temperature' with QoS 0
Received message ' 28.03' on topic '/esp8266/DHT11/temperature' with QoS 0
temperature update
temperature update
192.168.0.100 - - [19/Mar/2018 00:36:25] "GET /socket.io/?EIO=3&transport=polling&t=1521419776404-73&sid=3bae562803c943cb91a42cc9c7841600 HTTP/1.1" 200 -
Received message ' 45.00' on topic '/esp8266/DHT11/humidity' with QoS 0
Received message ' 45.00' on topic '/esp8266/DHT11/humidity' with QoS 0
humidity update
humidity update
192.168.0.100 - - [19/Mar/2018 00:36:25] "GET /socket.io/?EIO=3&transport=polling&t=1521419786262-74&sid=3bae562803c943cb91a42cc9c7841600 HTTP/1.1" 200 -
Received message ' 28.99' on topic '/esp8266/DHT11/temperature' with QoS 0
Received message ' 28.99' on topic '/esp8266/DHT11/temperature' with QoS 0
temperature update
temperature update
```

4. Contrôler les objets à partir d'un fureteur



The screenshot shows a web browser window with the address bar displaying `192.168.0.113:8181/esp8266/4/1`. The page content includes the title "ESP8266", a status message "GPIO 4 actuellement est **on**", a "Turn off" button, another status message "GPIO 5 actuellement est **off**", a "Turn on" button, and sensor data: "Lecture du DHT11 (updated 20h:9m:46s)", "Temperature: 28.99°C", and "Humidity: 44.00%".

ESP8266

GPIO 4 actuellement est **on**

Turn off

GPIO 5 actuellement est **off**

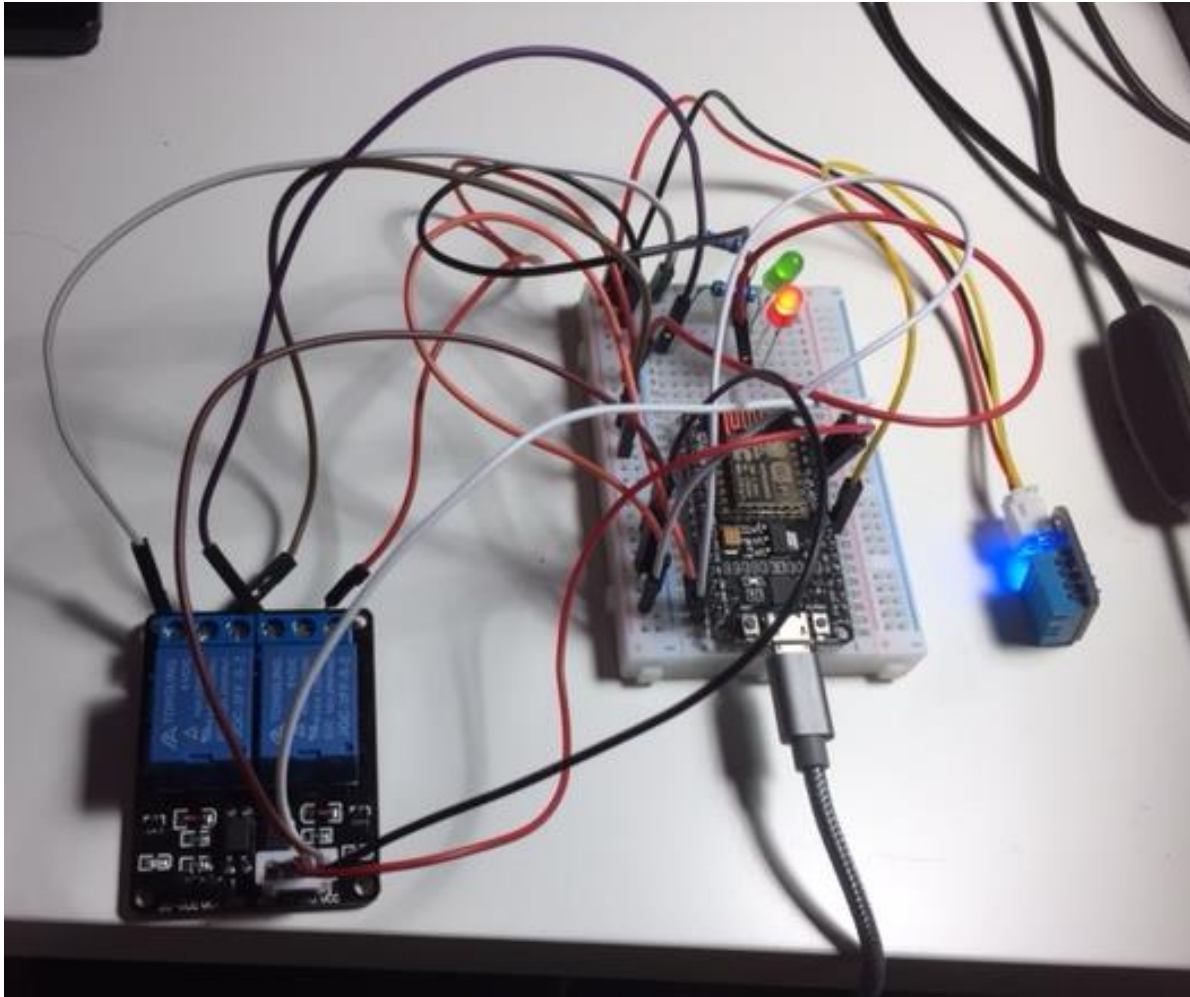
Turn on

Lecture du DHT11 (updated 20h:9m:46s)

Temperature: 28.99°C

Humidity: 44.00%

```
Humidity: 44.00 %      Temperature: 29.00 *C 84.20 *F  Heat index: 28.99 *C
Humidity: 46.00 %      Temperature: 28.00 *C 82.40 *F  Heat index: 28.11 *C
Humidity: 45.00 %      Temperature: 28.00 *C 82.40 *F  Heat index: 28.03 *C
Message arrived on topic: esp8266/4. Message: 0
Changing GPIO 4 to Off
Message arrived on topic: esp8266/5. Message: 0
Changing GPIO 5 to Off
Message arrived on topic: esp8266/4. Message: 1
Changing GPIO 4 to On
Message arrived on topic: esp8266/5. Message: 1
Changing GPIO 5 to On
Humidity: 45.00 %      Temperature: 28.00 *C 82.40 *F  Heat index: 28.03 *C
Humidity: 44.00 %      Temperature: 29.00 *C 84.20 *F  Heat index: 28.99 *C
Message arrived on topic: esp8266/4. Message: 0
Changing GPIO 4 to Off
Humidity: 44.00 %      Temperature: 29.00 *C 84.20 *F  Heat index: 28.99 *C
Message arrived on topic: esp8266/5. Message: 0
Changing GPIO 5 to Off
Message arrived on topic: esp8266/4. Message: 1
Changing GPIO 4 to On
Message arrived on topic: esp8266/5. Message: 1
Changing GPIO 5 to On
Humidity: 46.00 %      Temperature: 29.00 *C 84.20 *F  Heat index: 29.20 *C
```



5. Évaluer la possibilité d'utiliser uPython

On se propose dans cette partie d'évaluer la possibilité d'utiliser MicroPython, embarqué sur le ESP8266, pour contrôler à partir d'un fureteur un relais.

Les prérequis pour réaliser ce projet sont suivants :

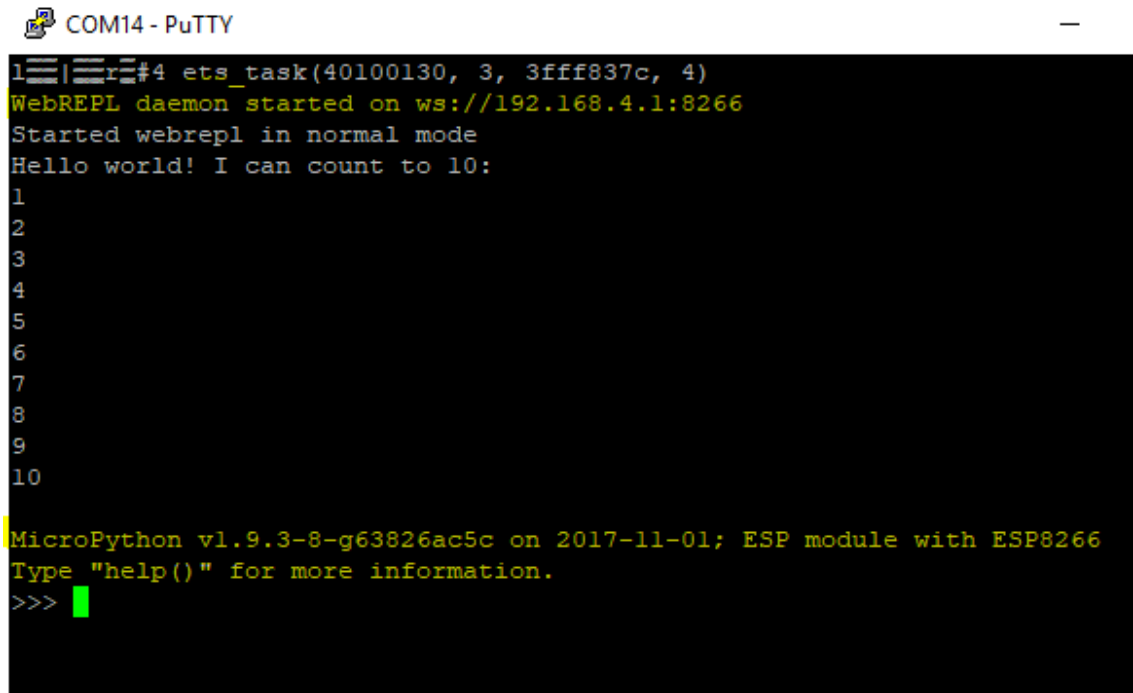
5.1 Préparer l'environnement - le firmware de MicroPython (Python préinstallé):

```
pip install esptool
esptool.py --port /dev/ttyUSB0 erase_flash
IMPORTANT pour Windows au lieu de /dev/ttyUSB0 mettez
COM7 (votre port)
esptool.py --port /dev/ttyUSB0 --baud 460800 write_flash --
flash_size=detect -fm dio 0 esp8266-20170108-v1.8.7.bin (la
version courant)
```

Pour plus de détail sur l'environnement voir le lien :

<http://docs.micropython.org/en/latest/esp8266/esp8266/tutorial/intro.html>

Pour tester le code sur ESP8266(32) on peut le faire sur REPL prompt [import webrepl_setup] (putty sur Windows ou picocom sur Linux), mais ce n'est pas un moyen plus efficace à faire.



```
COM14 - PuTTY

1#4 ets_task(40100130, 3, 3fff837c, 4)
WebREPL daemon started on ws://192.168.4.1:8266
Started webrepl in normal mode
Hello world! I can count to 10:
1
2
3
4
5
6
7
8
9
10

MicroPython v1.9.3-8-g63826ac5c on 2017-11-01; ESP module with ESP8266
Type "help()" for more information.
>>>
```

Pour flascher les codes écrits sur n'importe quel éditeur on va utiliser **ampy** :

```
pip install adafruit-ampy
```

5.2 Code en microPython:

```
# Begin configuration
TITLE      = "ESP8266 MicroPython WebSocket"
STA_SSID   = "xxxxxxxxxx"
STA_PSK    = "xxxxxxxxxx"
# End configuration

import network
import machine
import usocket

ap_if = network.WLAN(network.AP_IF)
if ap_if.active(): ap_if.active(False)
sta_if = network.WLAN(network.STA_IF)
if not sta_if.active(): sta_if.active(True)
if not sta_if.isconnected(): sta_if.connect(STA_SSID, STA_PSK)

pin = machine.Pin(5, machine.Pin.OUT)
```

```

pin.off()

def ok(socket, query):

    socket.write("HTTP/1.1 OK\r\n\r\n")
    socket.write("<!DOCTYPE html><title>"+TITLE+"</title><body><h1>")

    socket.write("LED status: ")
    if pin.value():
        socket.write("<span style='color:green'>ON</span>")
    else:
        socket.write("<span style='color:red'>OFF</span>")
    socket.write("<br>")
    if pin.value():
        socket.write("<form method='POST'
action='/off?"+query.decode()+"'>"+
        "<input type='submit' value='turn OFF'>"+
        "</form>")
    else:
        socket.write("<form method='POST'
action='/on?"+query.decode()+"'>"+
        "<input type='submit' value='turn ON'>"+
        "</form>")

    socket.write ("</h1></body>")

def err(socket, code, message):
    socket.write("HTTP/1.1 "+code+" "+message+"\r\n\r\n")
    socket.write("<h1>"+message+"</h1>")

def handle(socket):

    (method, url, version) = socket.readline().split(b" ")
    if b"?" in url:
        print("url is : ", url)
        (path, query) = url.split(b"?", 2)
    else:
        (path, query) = (url, b"")
    while True:
        header = socket.readline()
        if header == b"":
            return
        if header == b"\r\n":
            break

    if version != b"HTTP/1.0\r\n" and version != b"HTTP/1.1\r\n":
        err(socket, "505", "Version Not Supported")
    elif method == b"GET":
        if path == b"/":
            ok(socket, query)
        else:
            err(socket, "404", "Not Found")
    elif method == b"POST":
        if path == b"/on":
            pin.on()
            ok(socket, query)
        elif path == b"/off":

```

```

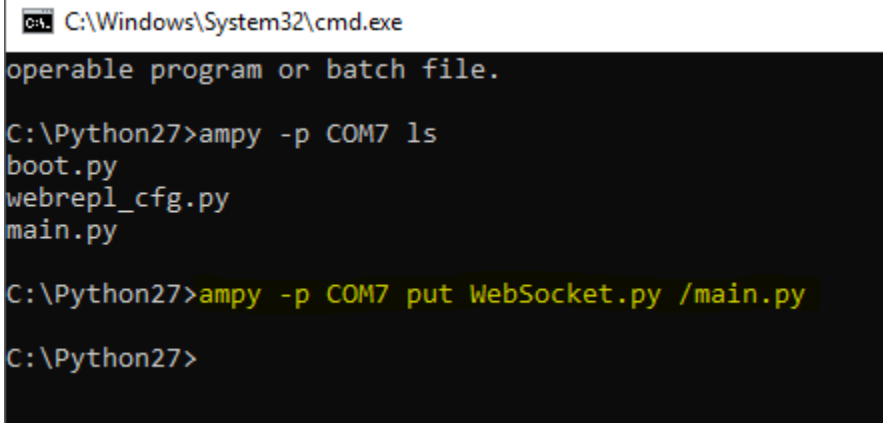
        pin.off()
        ok(socket, query)
    else:
        err(socket, "404", "Not Found")
    else:
        err(socket, "501", "Not Implemented")

server = usocket.socket()
server.bind(('192.168.0.116', 80))
server.listen(1)
while True:
    try:

        (socket, sockaddr) = server.accept()
        handle(socket)
    except:
        socket.write("HTTP/1.1 500 Internal Server Error\r\n\r\n")
        socket.write("<h1>Internal Server Error</h1>")
        socket.close()

```

5.3 Flasher le script:



C:\Windows\System32\cmd.exe

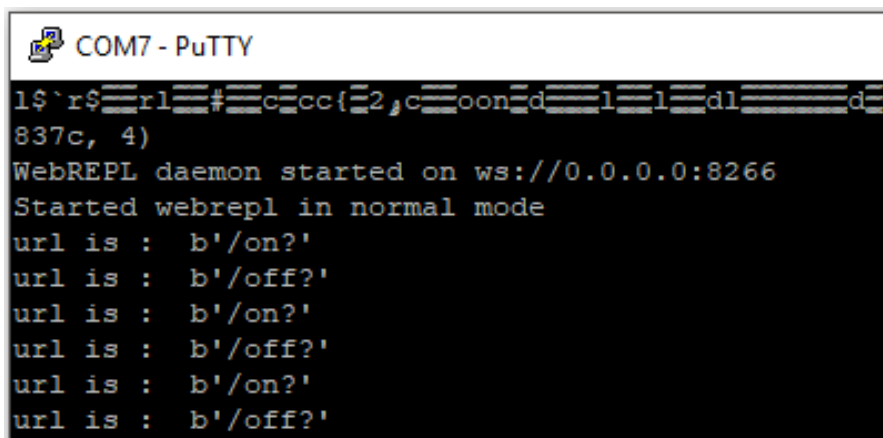
operable program or batch file.

C:\Python27>ampy -p COM7 ls
boot.py
webrepl_cfg.py
main.py

C:\Python27>ampy -p COM7 put WebSocket.py /main.py

C:\Python27>

Dans WebREPL :

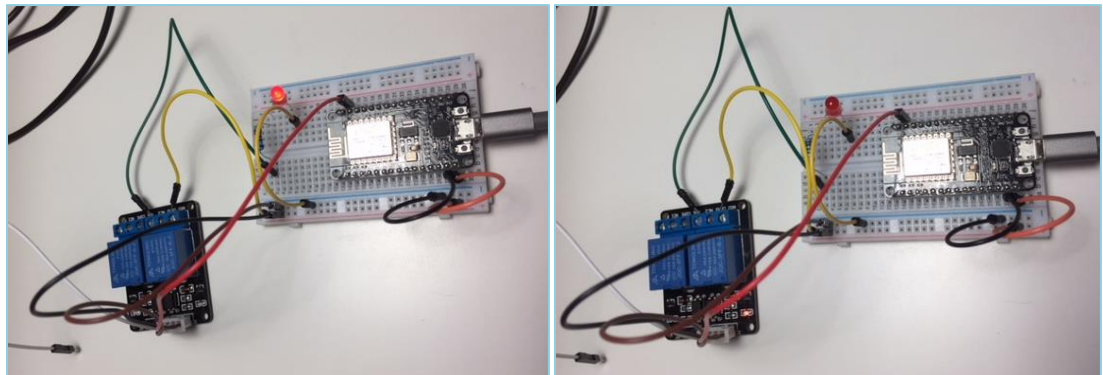
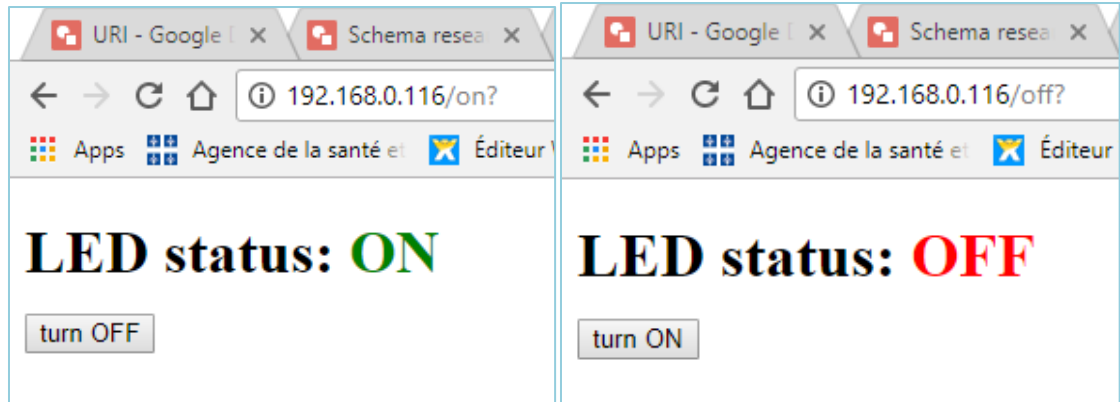


COM7 - PuTTY

```

l$`r$==rl==#==c==cc{=2,c==oon=d==l==l==dl====d=0
837c, 4)
WebREPL daemon started on ws://0.0.0.0:8266
Started webrepl in normal mode
url is : b'/on?'
url is : b'/off?'
url is : b'/on?'
url is : b'/off?'
url is : b'/on?'
url is : b'/off?'

```



ANNEXE1

Code serveur:

```
import paho.mqtt.client as mqtt
from flask import Flask, render_template, request
from flask_socketio import SocketIO, emit

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app)

# The callback for when the client receives a CONNACK response from the
server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("/esp8266/DHT11/temperature")
    client.subscribe("/esp8266/DHT11/humidity")
```

```

# The callback for when a PUBLISH message is received from the ESP8266.
def on_message(client, userdata, message):
    #socketio.emit('my variable')
    print("Received message '" + str(message.payload) + "' on topic '"
          + message.topic + "' with QoS " + str(message.qos))
    if message.topic == "/esp8266/DHT11/temperature":
        print("temperature update")
        socketio.emit('dht_temperature', {'data': message.payload})
    if message.topic == "/esp8266/DHT11/humidity":
        print("humidity update")
        socketio.emit('dht_humidity', {'data': message.payload})

mqttc=mqtt.Client()
mqttc.on_connect = on_connect
mqttc.on_message = on_message
mqttc.connect("localhost",1883,60)
mqttc.loop_start()

# Create a dictionary called pins to store the pin number, name, and pin
state:
pins = {
    4 : {'name' : 'GPIO 4', 'board' : 'esp8266', 'topic' : 'esp8266/4',
        'state' : 'False'},
    5 : {'name' : 'GPIO 5', 'board' : 'esp8266', 'topic' : 'esp8266/5',
        'state' : 'False'}
}

# Put the pin dictionary into the template data dictionary:
templateData = {
    'pins' : pins
}

@app.route("/")
def main():
    # Pass the template data into the template main.html and return it to the
    user
    return render_template('main.html', async_mode=socketio.async_mode,
    **templateData)

# The function below is executed when someone requests a URL with the pin
number and action in it:
@app.route("/<board>/<changePin>/<action>")
def action(board, changePin, action):
    # Convert the pin from the URL into an integer:
    changePin = int(changePin)
    # Get the device name for the pin being changed:
    devicePin = pins[changePin]['name']
    # If the action part of the URL is "1" execute the code indented below:
    if action == "1" and board == 'esp8266':
        mqttc.publish(pins[changePin]['topic'], "1")
        pins[changePin]['state'] = 'True'
    if action == "0" and board == 'esp8266':
        mqttc.publish(pins[changePin]['topic'], "0")
        pins[changePin]['state'] = 'False'
    # Along with the pin dictionary, put the message into the template data
    dictionary:

```

```

    templateData = {
        'pins' : pins
    }
    return render_template('main.html', **templateData)

@socketio.on('my event')
def handle_my_custom_event(json):
    print('received json data here: ' + str(json))

if __name__ == "__main__":
    socketio.run(app, host='0.0.0.0', port=8181, debug=True)

```

Code HTML :

```

<!DOCTYPE html>

<head>

    <title>RPi Web Server</title>

    <!-- Latest compiled and minified CSS -->

    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
integrity="sha384-
1q8mTJOASx8j1Au+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"
crossorigin="anonymous">

    <!-- Optional theme -->

    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-
theme.min.css" integrity="sha384-
fLW2N01lMqjakBkx3l/M9EahuwPsfeNvV63J5ezn3uZzapT0u7EYsXMjQV+0En5r"
crossorigin="anonymous">

    <!-- Latest compiled and minified JavaScript -->

    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"
integrity="sha384-
0mSbJDEHialfmuBBQP6A4Qrprq5OVfW37PRR3j5ELqxss1yVqOtnepnHVP9aJ7xS"

```

```

crossorigin="anonymous"></script>

<script src="https://code.jquery.com/jquery-3.1.1.min.js"
integrity="sha256-hVVnYaiADRT02PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8="
crossorigin="anonymous"></script>

<meta name="viewport" content="width=device-width, initial-scale=1">

<script type="text/javascript"
src="//cdnjs.cloudflare.com/ajax/libs/socket.io/1.3.6/socket.io.min.js"></sc
ript>

<script type="text/javascript" charset="utf-8">

    $(document).ready(function() {

        var socket = io.connect('http://' + document.domain + ':' +
location.port);

        socket.on('connect', function() {

            socket.emit('my event', {data: 'I\'m connected!'});

        });

        socket.on('dht_temperature', function(msg) {

            var nDate = new Date();

            $('#readingsUpdated').text(nDate.getHours() + 'h:' +
nDate.getMinutes() +

                'm:' + nDate.getSeconds() + 's').html();

            $('#temperature').text(msg.data).html();

        });

        socket.on('dht_humidity', function(msg) {

            $('#humidity').text(msg.data).html();

        });

    });

</script>

</head>

```

```

<body>

  <h1>Web Server - ESP8266</h1>

  {% for pin in pins %}

  <h2>{{ pins[pin].name }}

  {% if pins[pin].state == 'True' %}

    is currently <strong>on</strong></h2><div class="row"><div class="col-
md-2">

      <a href="/esp8266/{{pin}}/0" class="btn btn-block btn-lg btn-default"
role="button">Turn off</a></div></div>

    {% else %}

      is currently <strong>off</strong></h2><div class="row"><div
class="col-md-2">

      <a href="/esp8266/{{pin}}/1" class="btn btn-block btn-lg btn-primary"
role="button">Turn on</a></div></div>

    {% endif %}

  {% endfor %}

  <h3>Lecture du DHT11 (updated <span id="readingsUpdated"></span></h3>

  <h3>Temperature: <span id="temperature"></span>°C</h3>

  <h3>Humidity: <span id="humidity"></span>%</h3>

</body>

</html>

```

Code ESP8266:

```

// Loading the ESP8266WiFi library and the PubSubClient library

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

#include "DHT.h"

```



```

#define DHTTYPE DHT11

// Change the credentials below, so your ESP8266 connects to your router
const char* ssid = "xxxxxxxxxx";
const char* password = "xxxxxxxxxx";

// Change the variable to your Raspberry Pi IP address, so it connects to
//your MQTT broker
const char* mqtt_server = "192.168.0.113";

// Initializes the espClient
WiFiClient espClient;
PubSubClient client(espClient);

// Connect an LED to each GPIO of your ESP8266
const int ledGPIO5 = 5;
const int ledGPIO4 = 4;

// DHT Sensor
const int DHTPin = 14;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;

// Don't change the function below. This functions connects your ESP8266 to
your router
void setup_wifi() {
    delay(10);

    // We start by connecting to a WiFi network

```

```

Serial.println();

Serial.print("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {

    delay(500);

    Serial.print(".");

}

Serial.println("");

Serial.print("WiFi connected - ESP IP address: ");

Serial.println(WiFi.localIP());

}

// This functions is executed when some device publishes a message to a
// topic that your ESP8266 is subscribed to

// Change the function below to add logic to your program, so when a device
// publishes a message to a topic that
// your ESP8266 is subscribed you can actually do something

void callback(String topic, byte* message, unsigned int length) {

    Serial.print("Message arrived on topic: ");

    Serial.print(topic);

    Serial.print(". Message: ");

    String messageTemp;

    for (int i = 0; i < length; i++) {

        Serial.print((char)message[i]);

        messageTemp += (char)message[i];

    }

```

```

    Serial.println();

// If a message is received on the topic home/office/esp1/gpio2, you check
//if the message is either 1 or 0. Turns the ESP GPIO according to the
//message

    if(topic=="esp8266/4"){

        Serial.print("Changing GPIO 4 to ");

        if(messageTemp == "1"){

            digitalWrite(ledGPIO4, HIGH);

            Serial.print("On");

        }

        else if(messageTemp == "0"){

            digitalWrite(ledGPIO4, LOW);

            Serial.print("Off");

        }

    }

    if(topic=="esp8266/5"){

        Serial.print("Changing GPIO 5 to ");

        if(messageTemp == "1"){

            digitalWrite(ledGPIO5, HIGH);

            Serial.print("On");

        }

        else if(messageTemp == "0"){

            digitalWrite(ledGPIO5, LOW);

            Serial.print("Off");

        }

    }

    Serial.println();

```

```

}

// This functions reconnects your ESP8266 to your MQTT broker
// Change the function below if you want to subscribe to more topics with
your ESP8266
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266Client")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this
//example)
            client.subscribe("esp8266/4");
            client.subscribe("esp8266/5");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

// The setup function sets your ESP GPIOs to Outputs, starts the serial

```

```

communication at a baud rate of 115200

// Sets your mqtt broker and sets the callback function

// The callback function is what receives messages and actually controls the
LEDs

void setup() {
    dht.begin();

    pinMode(ledGPIO4, OUTPUT);
    pinMode(ledGPIO5, OUTPUT);

    Serial.begin(115200);

    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
}

// For this project, you don't need to change anything in the loop function.
// Basically it ensures that you ESP is connected to your broker

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    if(!client.loop())
        client.connect("ESP8266Client");
    now = millis();

    // Publishes new temperature and humidity every 10 seconds
    if (now - lastMeasure > 10000) {
        lastMeasure = now;

        // Sensor readings may also be up to 2 seconds 'old' (its a very slow

```

```

sensor)

    float h = dht.readHumidity();

    // Read temperature as Celsius (the default)

    float t = dht.readTemperature();

    // Read temperature as Fahrenheit (isFahrenheit = true)

    float f = dht.readTemperature(true);


    // Check if any reads failed and exit early (to try again).

    if (isnan(h) || isnan(t) || isnan(f)) {

        Serial.println("Failed to read from DHT sensor!");

        return;
    }

    // Computes temperature values in Celsius

    float hic = dht.computeHeatIndex(t, h, false);

    static char temperatureTemp[7];

    dtostrf(hic, 6, 2, temperatureTemp);


    // Uncomment to compute temperature values in Fahrenheit

    // float hif = dht.computeHeatIndex(f, h);

    // static char temperatureTemp[7];

    // dtostrf(hic, 6, 2, temperatureTemp);


    static char humidityTemp[7];

    dtostrf(h, 6, 2, humidityTemp);


    // Publishes Temperature and Humidity values

    client.publish("/esp8266/DHT11/temperature", temperatureTemp);

    client.publish("/esp8266/DHT11/humidity", humidityTemp);

```

```
Serial.print("Humidity: ");  
Serial.print(h);  
Serial.print(" %\t Temperature: ");  
Serial.print(t);  
Serial.print(" *C ");  
Serial.print(f);  
Serial.print(" *F\t Heat index: ");  
Serial.print(hic);  
Serial.println(" *C ");  
// Serial.print(hif);  
// Serial.println(" *F");  
}  
}
```