Article original

Article datant de 2015

- Introduction
- Contexte et problématique
- ESP8266 et NodeMCU
- Fonctionnement d’internet et du projet
- Montage
- Programmation
- Boitier
- App Android
- Utilisation
- Limites et conclusion

Introduction

> C'est un projet de mon blog écrit il y a plusieurs années que je capitalise ici. La plupart des informations restent tout à fait valables mais il se peux que depuis quelques bouts de code ait évolués ... N'hésitez pas à commenter ou à venir en parler sur le chat !

Bonjour à tous,

Aujourd’hui je vous propose un projet que j’ai réalisé en à peine quelques heures sur une plateforme qui m’était inconnue ! Bon pas totalement vous allez le voir car intégrée à l’IDE Arduino □□

Contexte et problématique

Le point de départ de tout ça c'est mon meilleur ami, enfin son portail d'entrée de sa copropriété pour être précis. En effet, sa copropriété ne lui a refilé qu'une seule télécommande, ce qui n'était pas forcément très dérangeant au départ mais maintenant il a un colocataire … Ça devenait compliqué …





Donc au départ quand il me présente le problème, je me dit basiquement « oh y'a 99% de chance pour que ça soit du 315 MHz, du 433 MHz ou du 968 MHz et j'aurais donc le loisir de me servir d'une copie chinoise » sauf que … j'amène mon matos d'analyse radio (si le sujet vous intéresse, chercher « RTL SDR » sur google) eeeeeetttt … Niet, j'arrive à rien analyser … Ok tu le prends

comme ça ? J’ouvre l’engin, regarde la puce, en galérant un peu je tombe finalement sur la fiche de donnée et outch : c’est du 3 kHz ! Première fois que je croise ça perso … Bref j’ai écumé le net pas moyen de trouver une télécommande dans cette fréquence. Bref on l’avais plutôt dans l’os pour le coup.

Et puis quelques jours plus tard je me dis que je pourrais tout simplement attaquer le problème par un autre bord : et si je connectais la télécommande par le wifi et un relais ? Et oui, je peux simuler l’appui sur le bouton avec un relais piloté par un serveur connecté à internet. Alors c’est possible car je ne l’avais pas précisé, sa télécommande marche depuis son appartement. Et aujourd’hui, avec l’ESP8266 les prix pour se faire ce genre de montage ont drastiquement chuté !

ESP8266 et NodeMCU

Let’s talk about the ESP8266

Alors pour ceux qui n’ont pas ou peu entendu parlé, l’ESP8266 est un microcontroleur ARM avec tout ce qu’il faut pour faire du wifi pour un prix dérisoire (on trouve des modules à 3$). Au départ c’était pas mal de bricolage, mais à 3$ ça a motivé beaucoup de bidouilleurs, et c’est aujourd’hui beaucoup plus facile et fiable, notamment grace à une petite carte de développement et un firmware portant tout deux le même nom de NodeMCU. Vous pouvez la trouver ici pour 7,45$.

A quoi elle ressemble :



Pin mapping :



Une petite note sur le sujet, malgré la sérigraphie D0 D1 etc … ressemblant furieusement à celle des Arduino, la numérotation est celle marquée sous la forme « GPIOx » et il faut utiliser le x dans l’IDE Arduino comme numéro de pin (donc par exemple, digitalWrite(5,HIGH) pour le GPIO5 sérigraphié « D1 »).

L’avantage c’est que c’est une « vraie » board de développement : il y a tout ce qu’il faut pour la programmer, et brancher des choses dessus (chose plus complexe avec les modules seuls).

Pour l’installer sur votre IDE Arduino (version 1.6.4 minimum) je vous invite à suivre cette procédure très simple.

Après c’est comme une Arduino : même langage et il y a de nombreux exemples pour vous montrer ce que la bête apporte.

Fonctionnement d’internet et du projet

Il me parait important avant d’expliquer certains principes fondamentaux et indispensables ici sur le fonctionnement d’internet. Tout d’abord la notion d’**adresse IP** et de **DNS**. Alors l’adresse IP vous en avez sans doute entendu parler c’est ce chiffre étrange qui ressemble à 186.14.26.54 par exemple. En fait, 4 valeurs allant de 0 à 255 (un octet quoi) séparé par des points. Ce numéro correspond à une adresse dans un réseau donné. Chez vous si vous n’avez pas de réseau bidouillé (auquel cas de toute façon vous n’avez même pas pris la peine d’attaquer la lecture de ce paragraphe) vous avez donc : une adresse IP sur internet (votre box) et une adresse par périphérique réseau (votre PC, votre tablette en wifi, votre smartphone en wifi …). Toutefois les adresses réseaux « internet » et « locale » n’ont rien à voir. C’est un petit peu comme « avenue du Général de Gaulle » : y’en a une à Paris, une à Marseille etc … Pourtant ce n’est pas la même avenue □□ Et sinon comme toute adresse elle permet de transmettre des messages à un destinataire précis.

Concrètement prenons un exemple : vous allez sur un site internet. En tapant l’adresse IP du site dans votre navigateur internet, vous envoyez une requête de type GET vers cette adresse pour récupérer le site internet et l’afficher (qui est en fait un simple fichier texte codé). Cette requête fait parti du protocole HTTP. En fait internet c’est des messages qui transitent. Mais il faut un peu mettre tout le monde d’accord organiser sécuriser tout ça. La intervient le protocole TCP/IP qui permet d’ouvrir un « canal » de discussion. Après il faut savoir ce qu’on demande : arrive le protocole HTTP. C’est exactement comme deux personnes qui discutent : on a des cordes vocales pour produire différents sons, ensuite on forme des mots, puis des phrases et on rajoute la langue et la politesse par dessus « coucou moi c’est Bob » « coucou moi c’est Peter ». Internet c’est un peu pareil, et même en élargissant c’est le cas de toute les communications numériques.

Bref revenons en à notre site internet : vous aller me dire « mais moi je tape pas une adresse IP dans ma barre de navigation ??! » Et oui vous tapez http://www.google.fr ou http://www.arduino.cc. C’est la que le serveur DNS fait son apparition : quand vous taper une adresse, un appel vers un serveur DNS (vous pouvez configurer ça dans votre panneau de configuration) est réalisé pour lui demander quel est l’IP correspondant à ce site. Et hop magique vous récupérez l’adresse IP du site et entamer la discussion avec □□

« Ok et mon réseau local à moi ? » Donc maintenant vous avez compris que vous pouvez taillé le bout de gras avec tout internet juste avec votre navigateur. Donc maintenant imaginons vous êtes au boulot, vous connaissez l’adresse IP de votre maison (au passage, aujourd’hui tous le monde est en IP fixe, c’est à dire qu’elle ne change normalement jamais, il y a quelques années ce n’était pas le cas) et que vous la tapez dans votre navigateur. Sauf que qui va répondre à l’autre bout ? Personne car vous n’avez pas de serveur. Votre message va arriver à votre box qui fait office de

routeur, c'est à dire que c'est la standardiste : « bonjour vous voulez vous adressez à qui ? ». Car si vous faites le test chez vous à partir de différents PC ou autre objet connecté à votre réseau, vous verrez que vous avez la même IP sur internet. C'est normal, c'est le routeur votre point d'entrée au réseau internet. Après il y a votre réseau local ou vous avez une autre adresse IP, cette fois différente pour chaque objet connecté à votre réseau. Et la vous allez me dire « mais comment je fais alors pour m'adresser à un périphérique de mon réseau local depuis l'extérieur ? » c'est la que la notion de « port » intervient. En effet, votre routeur (« la standardiste ») reçoit une communication sur son adresse IP internet, mais reçoit en plus un numéro de port. C'est ce numéro qui va dire ensuite sur le réseau local avec qui vous voulez communiquer. Typiquement en http c'est le port 80 qui est utilisé. C'est la aussi transparent pour l'utilisateur, quand vous visitez un site internet, en fait l'adresse IP complète envoyée est de la forme : xxx.xxx.xxx.xxx:80

Mais les navigateurs prennent en charge des numéros de port différents, il suffit de le spécifier en tapant http://xxx.xxx.xxx.xxx:yy yy étant votre numéro de port. Mais la vous allez me dire « mais comment il sait que ce port est pour ce périphérique précis de mon réseau ? » Et bien c'est la où il faut aller configurer sa box pour lui dire que les messages arrivant sur ce port la doivent être adressés à cette IP la du réseau local.

Sur la Freebox révolution voici la démarche .

IP de destination = IP de l'ESP8266

Protocole = TCP

IP Source = toute
Port de début = Port de fin = port avec lequel vous lancerez l'appel

Port de destination = port que l'on va paramétrer pour le serveur ESP8266.

En lisant ces deux dernières lignes vous avez peut-être compris que le numéro de port que vous allez taper dans votre navigateur n'est pas obligatoirement celui que l'on va spécifier sur notre serveur. On pourra donc se connecter en tapant par exemple xxx.xxx.xxx.xxx:260 alors que le serveur sera sur le port 504 pourvu que le routeur soit configuré avec un port de début 260 et un port de destination 504 pour l'IP local de l'ESP8266.

Sur les autres box ce sera guère différent dans l'esprit, attention certaines font la différence entre le firewall et le routeur, ce qui fait qu'il faut prévoir la configuration dans les deux et ne pas se retrouvé avec une redirection bloquée par le pare-feu □□

Bon j'espère vous avoir appris plus de trucs que de vous avoir embrouillé mais il me semblait indispensable de préciser ces points afin de comprendre le code (très court en plus) qui va suivre.

Montage

Le montage est très simple : la carte, un relais, et on tire les deux fils du bouton. J’ai ajouté également un bouton manuel pour pouvoir piloter la télécommande même si tout est planté et sans avoir forcément à passer par un PC ou un smartphone pour ouvrir le portail. Pour la petite histoire, je monte le projet et pouf marche pas … Hum je regarde mon relais : arf 12V !! Donc il ne se fermait pas à 5V. J’avais pas de matos sous la main donc j’ai viré le relais et j’ai utilisé le mosfet qui pilotait le relais pour faire le contact. Voici les deux schémas en un, au crayon gris celui que j’avais imaginé, en rouge les raccordements une fois le relais dessoudé :

SCAN2

La node MCU est simplement alimenté par un chargeur USB de téléphone.

Programmation

On va donc construire un mini serveur, qui quand il recevra une requête exécutera différentes actions. On va faire en sorte de lui donner les fonctions suivantes :

- répondre par une page un bouton « ouvrir »
- procéder à l'ouverture en elle même
- quand l'ordre d'ouverture est exécuté envoyer une page de confirmation

On va donc lancer son IDE Arduino et commencer par inclure les librairies qu'il faut :

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
```

Ensuite on va créer notre serveur sur un port normalement libre (grosso modo éviter les ports « classiques » des différents protocoles internet, une recherche google vous les donnera).

```
long port = 504;
ESP8266WebServer server(port); // serveur HTTP
```

On va ajouter une petite fonctionnalité gadget qui est le mDNS : vous vous souvenez du serveur DNS ? Ca marche sur internet, mais chez vous ? Et oui y'en a pas ... Donc si vous voulez accéder à votre module sans connaitre son IP par coeur, il faut utiliser du mDNS. L'idée c'est d'envoyer un message à tout le monde le réseau et répond qui se sent concerné. On lance donc un « serveur » mDNS qui permettra de répondre à un domaine précis. Ici on va rentrer ça :

```
MDNSResponder mdns; // serveur mDNS

...

if (mdns.begin("esp8266", WiFi.localIP())) {
Serial.println("MDNS responder started");
}
```

Et on pourra appeler notre module dans le navigateur, connecté sur son réseau local (ça ne marchera pas depuis l'extérieur) en tapant « esp8266.local ».

Ensuite on va définir les réponses de notre serveur :

```
server.on("/", handle_root);
server.on("/open", handle_open);
```

Avec ces commandes, si je tape monip:monport/open j'exécute directement la routine d'ouverture, et il me répondra avec « opening ». Si je tape monip:monport il me répondra avec une page internet que vous visualisez ici en HTML. Lorsque j'appuierais sur le bouton « Ouvrir » de la page, ça appellera la page /open et donc l'ouverture.

Evidemment il faut se raccorder au wifi, on spécifie donc ces paramètres de connexion au départ :

```
char* ssid = "xxxxx"; // votre SSID
char* password = "yyyyyyy"; // votre mot de passe wifi
```

Le reste du code est au final assez logique : on pilote le relais, on lance la connexion, on lance les serveurs et on debug sur le port série. Ce qui donne au final :

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

char* ssid = "xxxxx"; // votre SSID
char* password = "yyyyyyy"; // votre mot de passe wifi
MDNSResponder mdns; // serveur mDNS
long port = 504;
ESP8266WebServer server(port); // serveur HTTP

const int led = 16; // led integree au NodeMCU, attention logique inverse
const int relay = 5; // relais connecte au GPIO5

void setup(void) {

/* Configuration des entree/sortie */
pinMode(relay, OUTPUT);
pinMode(led, OUTPUT);

digitalWrite(led, 1);
digitalWrite(relay, 0);

Serial.begin(115200); // initialisation du port serie
```

```
connect(ssid, password); // connexion au reseau Wifi

Serial.println("");

/* demarrage du serveur mDNS sur esp8266.local */
if (mdns.begin("esp8266", WiFi.localIP())) {
Serial.println("MDNS responder started");
}

/* ajout des actions du serveur */
server.on("/", handle_root);
server.on("/open", handle_open);

server.begin(); // demarrage du serveur

Serial.println("HTTP server started");
}

void loop(void) {

server.handleClient(); // gestion du serveur

/* Si connecté au wifi alors la LED s'allume */
if (WiFi.status() == WL_CONNECTED) digitalWrite(led, 0);
else digitalWrite(led, 1);

}

/* Routine pour se connecter à un reseau wifi */
void connect(char *_SSID, char* _PWD) {

Serial.println("");
Serial.print("Connecting ");
Serial.print(_SSID);
Serial.print(" password = ");
Serial.print( _PWD);

WiFi.begin(_SSID, _PWD);
Serial.println("");
```

```
int h = 0;

// Wait for connection
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");

if (h++ > 40) { // si trop long on abandonne
Serial.println();
Serial.println("Failed to connect");
return;
}

}

Serial.println("");
Serial.print("Connected to ");
Serial.println(ssid);
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

}

/* Action serveur sur /open */
void handle_open() {
server.send(200, "text/plain", "opening"); // repond "opening"
open(); // actionne le relais
}

void handle_root() {

/* page[] contient notre page web et renvois vers le domaine /open si on appuie sur le bouton
*/
char page[] = "<h1>Ouverture portail</h1><p><a href=\"open\"><button>Ouvrir</button></a></p>";

server.send(200, "text/html", page); // repond avec la page web codee en HTML

}

void open() {
```

```
digitalWrite(relay, 1);
Serial.println("Opening");
delay(1000);
digitalWrite(relay, 0);

}
```

Sur Github

Boitier

J’ai fait une jolie boîte en medium découpée à la découpeuse laser □□ pour sa fabrication au total ça m’a pris …. 10 min. Comment ? Makercase.com est un site qui vous permet de générer des plans de boitier simple. J’ai rentré mes dimensions, l’épaisseur de mon matériau et le type d’assemblage et hop un fichier svg. Alors la je trouve ça assez con vu que c’est quand même le DXF qui est le fichier de base dans le domaine. On doit donc l’ouvrir sous Inkscape, tout sélectionner, faire « convertir objet en chemin » et enregistrer sous au format DXF pour avoir un fichier exploitable par la découpeuse. Mais avant un petit tour sous DraftSight pour ajouter les trous pour la fixation de la carte, le relais et le bouton, et un passage pour le câble d’alimentation. On découpe, et voila le résultat (désolé photo avec téléphone de secours) :

photo (1)

photo (2)

photo (5)

photo (4)

photo (7)

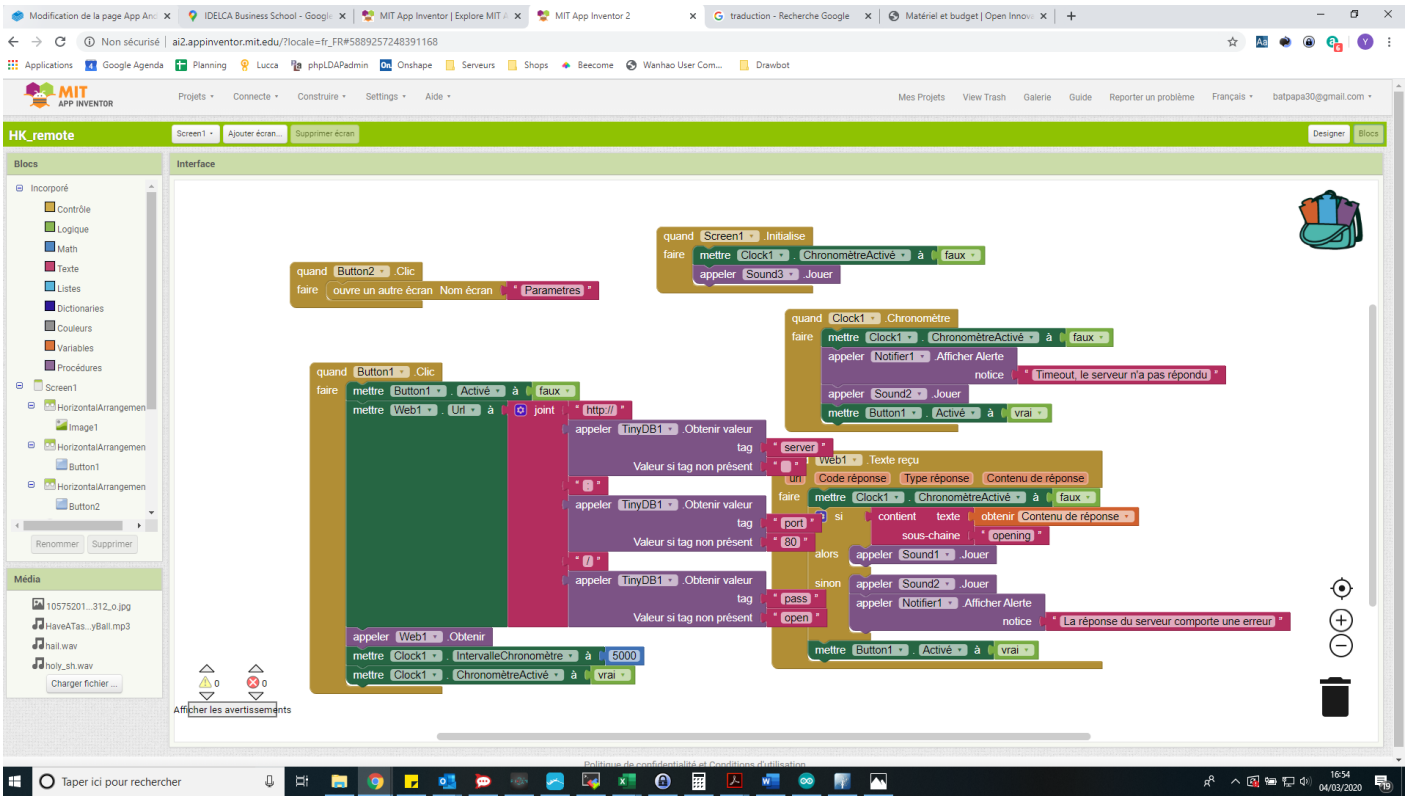
Classe hein ?

Au passage vous voyez :

- ma « correction » pour virer le relais et utiliser le mosfet.
- les fils sortant de la télécommande (désolé j’ai oublié de prendre une photo des deux fil soudés au bouton) elle même collé au fond du boitier avec du double-face.
- la nodeMCU est à l’envers, pour que je puisse accéder au pins par au-dessus.
- le bouton poussoir en façade.

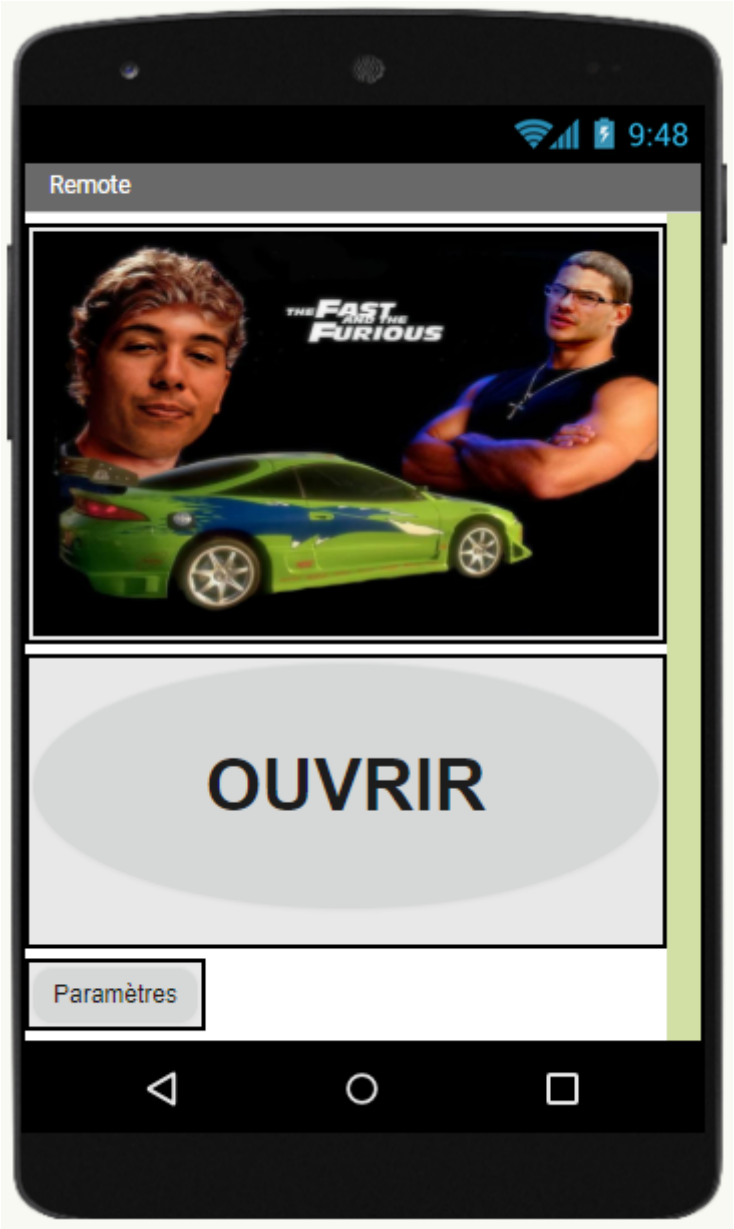
App Android

Alors y’a un super outil pour les gros noob d’Android comme moi, c’est appinventor. il vous suffit de créer un compte, et vous pouvez développer des applis toute simples, avec un codage graphique à la codeblock. C’est très intuitif.



En plus le debuging est simple : soit par connexion USB avec votre tel, soit par wifi en installant une application dédiée. A la fin quand votre projet est fini, vous pouvez : soit générer un QR code qui vous renverra un lien de téléchargement temporaire, soit télécharger le fichier .apk à installer en manuel (la aussi je vous renvois vers google c’est tout simple).

Voici l’appli : https://ai2.appinventor.mit.edu/?galleryId=5667337949151232



Bon l'écran de base c'est un gros troll pour mon pote avec un magnifique montage que vous pouvez faire péter évidemment

Il suffit d'aller dans « Paramètres », de rentrer votre IP et votre port et voila ça devrait fonctionner. Quand on appuie sur le bouton « ouvrir » ça envoi une bête requête, tant que l'app ne reçoit pas « opening » le bouton reste verrouillé ou bien il se déverrouille au bout de 5 secondes.

Une petite base donnée (« tinyDB ») permet à la fois de garder en mémoire les données pour ne pas avoir à les retaper à chaque lancement, et de passer les infos d'un screen à l'autre. En effet sur AppInventor on ne peut pas créer de variables globales, donc le seul moyen de se faire passer des infos d'un screen à l'autre c'est de passer par une base de donnée …

Au passage dans paramètre certains champs sont prévus mais inactifs : en fait à l'origine je pensais passer par du MQTT plutôt que du HTTP mais ça nécessitait trop de moyens …

> Je reviens sur ce dernier point : j'ai pas testé cette solution (qui n'existait pas à l'époque dela rédaction)http://bienonline.magix.net/public/android-AI2-MQTT-en.html Donc potentiellement on pourrait basculer en MQTT, et la il n'y aurait plus vraiment besoin de coder une application ...

Utilisation

Il me suffit maintenant soit de taper « esp8266.local » en local, ou « votreip:port » depuis l'extérieur pour accéder à ça :

(bien évidemment sur les captures j'ai pas laissé l'IP et le port de mon ami □□)

cc

On clique et normalement si tout va bien votre relais se ferme et il s'affiche :

cc2

Mais poussons le vice un poil plus loin …

Limites et conclusion

Un projet bouclé en à peine quelques heures, grâce à une série d'outils puissants de simplicité (bon j'imagine que si vous débutez complètement ça vous prendra plutôt quelques jours, surtout si tout ne marche pas du premier coup …). Ça démontre bien toutes les mutations en cours dans le domaine et qu'on est loin du temps ou il aurait fallut 4 ingénieurs pendant un mois pour faire ça …

Alors la principale limite ici c'est que ce n'est pas du tout sécurisé, n'importe qui qui a votre adresse IP et le bon port peut piloter votre système. Ici c'est un portail collectif d'une grande résidence donc bon … Et faut arriver à dégoter la bonne IP et le bon port ! Mais bon dans l'absolu c'est assez facilement crackable pour quelqu'un qui veut vous nuire, donc n'utilisez pas ce système pour votre porte d'entrée ou de garage.

Une autre limite est que si internet est capricieux chez vous, bin vous ouvrirez plus rien …

En terme de coût on est tout simplement sous la barre des 10€ sans pousser le vice outre mesure. Avec un circuit dédié on pourrait même tomber sous les 5€ sans problème □□

A bientôt et n'hésitez pas à poster vos questions en commentaire de cet article ou sur le chat □□