TapController :

Une application pour arroser votre jardin depuis l'autre bout du monde

Réalisée par : Zouhair KHATOURI

1) Description du projet :

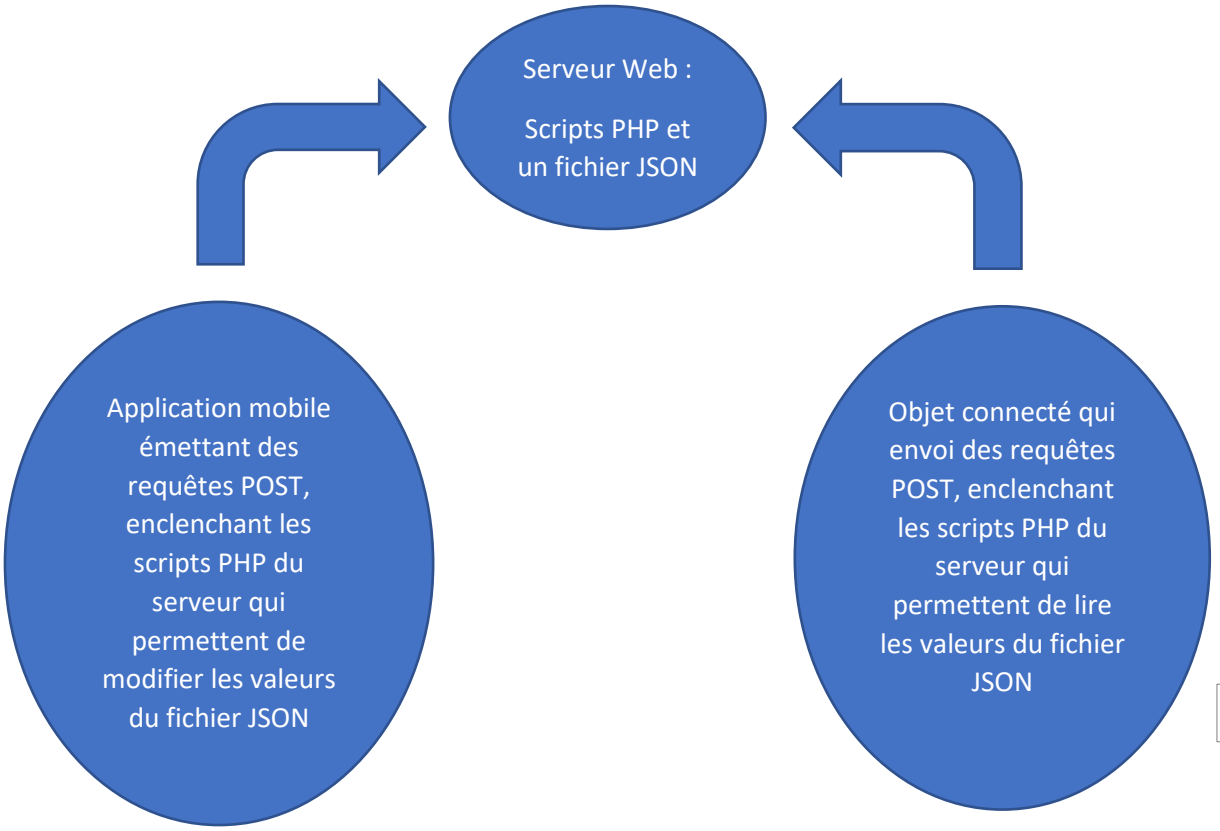
Le projet a pour but la conception d'un objet connecté qui permettra la manipulation d'un robinet depuis son téléphone. Il doit donc nécessairement comprendre trois structures qui vont interagir entre eux à savoir :

- Le hardware : à savoir l'objet connecté en soit, en plus du code qui lui sera incorporé.
- Le software : c'est-à-dire l'application mobile qui permettra à l'utilisateur de manipuler le robinet.
- L'interface entre hardware et software : que j'ai faite grâce à un serveur web hébergé (gratuitement) sur 000webhost.

Bien que la structure du hardware soit le cœur des sujets sur lesquels ont a travaillé durant les tutoriaux, les deux autres structures comprennent aussi des éléments qui seront intéressants d'évoquer afin de pouvoir expliquer le fonctionnement de l'ensemble.

2) Schéma global du fonctionnement de l'objet :

Le système fonctionne comme il suit :



Avant de rentrer dans les détails techniques du code, décrivons les composantes nécessaires au fonctionnement de l'objet.

3) Composantes de l'objet :

1. Une carte Arduino UNO faisant office de microcontrôleur.
2. Afin de faire tourner le robinet, on a besoin d'un moteur électrique. Dans mon projet j'ai utilisé un moteur à pas de type : 28byj-48. Ce moteur nécessite une interface de puissance afin de le commander, pour ce model il faut utiliser le drive ULN2003. La manipulation de ce moteur peut être faite avec des bibliothèques spécialisées, toutefois, j'ai préféré utiliser un code de j'ai repris depuis internet (voir les sources).
3. Un module Wifi permettant d'envoyer les requêtes POST au serveur. Pour cela, j'ai utilisé une ESP8266-01 (configuré en mode client), qui est en fait une carte de type SoC pouvant être programmée à l'image d'une carte Arduino UNO. Encore là, j'ai opté pour un code qui n'utilise pas des librairies, et utilisant des AT afin d'envoyer les requêtes au serveur.
4. Un module Bluetooth de type HC-05 (configuré en mode esclave). Ce module est nécessaire car l'utilisateur doit pouvoir au moins une fois synchroniser son application avec le hardware, afin de transmettre à ce dernier le SSID et le mot de passe du point d'accès au quel la carte ESP-01 va se connecter dans les utilisations futures.
5. Une LED faisant office de de voyant indiquant l'état de connectivité de la ESP8266-01.

Pour plus de détails sur les branchements veuillez vous référer au schéma du circuit.

4) Paramètres d'état du système connecté :

Vu qu'on manipule un robinet, le paramètre qu'il faut pouvoir commander est les pourcentages d'ouverture de ce dernier. Dans le code, je m'y suis référé par la variable ***double flow***. On peut imaginer un système où l'utilisateur commande le « flow » à distance à chaque instant, toutefois, cela est risqué car en cas de coupure de connexion par exemple, le robinet restera ouvert et l'eau ne cessera d'être gaspillée. Pour remédier à cela, j'ai restreint l'utilisateur à devoir imposer, en plus du flow du robinet, la durée de l'ouverture de ce dernier. Une fois cette durée écoulée, le robinet se fermera automatique sans devoir attendre un ordre de fermeture. On voit ainsi que la durée de la commande est aussi une caractéristique importante de la commande qui doit lue et écrite dans le serveur, je m'y suis référé par ***int duration*** dans le code. Justement, pour chaque client (identifié par une clé unique statiquement fournie au quelle je m'y suis référé par ***String clientID*** dans le code) le fichier JSON du serveur contient une ligne sous la forme :

Sha1(clientID) : »flow#duration »

Où l'on procède au préalable, via le script PHP du serveur, à l'encryptage Sha1 de la clé clientID. Le symbole # permet de séparer le flow de la durée lorsque ces deux valeurs sont lues dans le code de la carte Arduino.

5) Principales fonctions du code Arduino :

Sans rentrer dans trop de détails techniques, je vais énumérer ci-dessous les fonctions présentes dans le code Arduino et leurs utilités techniques :

- ***Void setup()*** : initialise l'état de la carte, i.e. définition des pins et des Serials pour la ESP-01 et le Bluetooth.
- ***Void loop()*** : Si l'objet connecté n'est pas dèja associé à une application (***paired = false***), le module Bluetooth est allumé, et écoute les messages qu'il capte pour éventuellement remplir les qui correspondent au SSID et le mot de passe du point d'accès au quelle la ESP8266-01 va se connecter. Si l'objet connecté est déjà associé à une application, loop() continue d'envoyer des requêtes POST afin de capter des changements de flow et duration (on fait du « Busy waiting » ce qui n'est pas très efficace comparé avec de l'MQTT).
- ***Void openTheTap(double dflow)*** : ouvre le robinet en faisant tourner le moteur. J'ai considéré qu'un dflow qui vaut 1 correspond à un tour complet du moteur (une ouverture complète du robinet initialement fermé).
- ***Void closeTheTap()*** : ferme le robinet totalement.
- ***Void stepper(double nbStep)*** : ceci est une méthode que j'ai copié sur internet. Elle permet de faire tourner le moteur de nbStep pas dans le sens trigonométrique.
- ***Boolean connectToAP()*** : fait connecter le module ESP8266-01 au point d'accès dont les paramètres sont obtenus via Bluetooth, de chez l'utilisateur, et sont stockés dans les variables ***String ssid*** et ***String pswd***.
- ***String httppost(String uri, String data)*** : cette méthode POST les données data (sous la forme de key1=value1&key2=value2…) à l'adresse uri. La requête est envoyée grâce aux commandes « AT » du module ESP8266-01.
- ***Double getData()*** : renvoie le « flow » écrit dans le fichier JSON du serveur, tout en mettant à jour la variable duration.
- ***Void executeInstruction()*** : ne fait que regrouper la suite d'instructions suivantes à la suite : lecture dans le serveur , actionnement du moteur, attente de l'écoulement de la durée spécifiée dans le serveur, fermeture du robinet, réinitialisation des valeurs écrites dans le serveurs.
- ***wakeUp()*** : procède à un hard reset du module ESP-01. Le code est tel que cette fonction doit être appelée après 10 échecs de tentatives d'envoi de requêtes POST (***int failure*** > 1). En effet, il est indispensable de procédé ainsi car la ESP8266-01 entre par défaut dans une période de sommeil (DeepSleep) après quelques cycles du code.

6) Fonction de l'application mobile :

Lorsque l'application mobile est lancée, elle demande éventuellement à l'utilisateur d'activer son Bluetooth et de s'associer à l'objet connecté qu'il aimerait manipuler. Quand cela est fait, une seconde activité est lancée où l'utilisateur doit recopier l'identifiant client de

l'objet (c'est un paramètre d'usine fournit avec l'objet), et où il doit entrer le SSID et le mot de passe du réseau au quel l'objet se connectera dans le futur. Quand cela est fait, et que le bouton « Pair » est cliqué, une chaine de caractère sous la forme « SSID#MotDePasse » est envoyée via Bluetooth à l'objet. Si tout se passe bien la dernière activité est lancée, sinon, on reste sur la même en indiquant via un Toast qu'il faudra réessayer en vérifiant les paramètres du hardware. A la troisième activité, l'utilisateur peut envoyer des commandes, qui, lorsqu'elles sont traduites sous forme d'instructions, envoie des requêtes POST au serveur, afin d'écrire la valeur de flow et de duration que l'utilisateur aura choisis. L'envoie des requêtes POST se fait grâce à l'API okhttp. Une telle opération doit être effectuée d'une façon asynchrone afin d'éviter le blocage du thread de l'interface graphique. Les interfaces graphiques des trois activités sont présentées ci-dessous.

1) Conclusion

Finalement, je suis satisfait du fait que le système soit fonctionnel. Toutefois, plusieurs optimisations auront pu être faite : l'utilisation nodeMCU, utilisation du protocole MQTT … Toutes ces optimisations non réalisées ont fait que les commandes arrivent parfois en retard, mais cela n'est pas catastrophique pour le type d'usage qu'on compte en faire.