IOT S4 2022
TP5 :Introduction webserver ESP8266

1. Un premier serveur web

- Modifier l'exemple `ESP8266WebServer->HelloServer` pour mettre les paramètres du Wifi de la salle ou de votre téléphone en mode partage de connexion. Charger le et vérifier son fonctionnement.

- Rajouter une page "`ok.html`" qui affiche le texte "`j'ai compris comment ça marche !`" en gras.

- Rajouter une page "`uptime.html`" qui affiche le nombre de secondes écoulées depuis le reset[1].

- Intégrer un serveur web avec votre programme général :

  - ajouter les fichiers `webserver.h` et `webserver.cpp`
  - créer les fonctions `setup_webserver()` et `loop_webserver()`
  - créer une page racine simple affichant un message de bienvenue.
  - créer une réponse pour les URL non disponibles comme sur l'exemple.

- Vérifier que les autres fonctionnalités ne sont pas affectées (mqtt,capteurs,led,…)

2. Une page HTML qui affiche la valeur de votre capteur

Dans les exemples fournis avec l'IDE arduino, le code html est écrit dans une chaîne de caractères (entre " "), ce qui oblige à "échapper" (mettre un \ devant) les guillemets et les fins de ligne.
On aura intérêt à utiliser les "raw string" c++ qui permettent d'insérer le html sans modification :

```
const char INDEX_HTML[] PROGMEM = R"=====(
          <html>   "ici on peut mettre le code html tel quel"
          </html>  )=====";
```

En vous inspirant de l'exemple fourni `webExample` ajouter une page qui affiche la valeur courante de votre capteur.

3. Commande d'un actionneur par une requête GET

Une requête de type GET permet de demander une ressource (exemple page html) et d'envoyer simultanément des informations (paramètres) supplémentaires que le serveur peut traiter.
La liste des informations introduite par un « ? » (point d'interrogation).
Chaque paramètre est nommé, il se compose donc d'un nom et d'une valeur : nom=valeur.
Les paramètres sont séparés par le caractère &.
La requête a la forme : `ip.votreEsp/ressource?nom1=valeur&nom2=valeur2`

1 Pour recharger automatiquement la page utiliser le `meta refresh`.

Par exemple :
- si on souhaite commander une led (on/off) on pourra envoyer une requête de la forme :
`/led ?state=on ou /led ?state=off`
- si on souhaite commander une led RGB on pourra envoyer une requête de la forme :
`/commandeRGB?red=230&green=120&blue=15`

led et commandeRGB son des noms de page (URL).
Les paramètres sont `state`,`red`,`green`,`blue`.

Pour comprendre comment on peut récupérer les arguments, analysez le comportement de la fonction **`handleNotFound()`** on demandant une page inexistante.

- Commander la led ANT (D4) par la requête `/led`
La page affiche simplement "`led is on`" ou "`led is off`"

- Commander la led ANT (D4) par la requête `/ledbtn`
La page affiche deux boutons "ON" et "OFF" qui permettent de piloter la led
ainsi que le message "`led is on`" ou "`led is off`"

- Commander une led de la bande par la requête `/commandeRGB`

3. Une page (statique) en Flash

La page précédente était stockée dans la RAM de l'ESP. Il est ainsi facile d'en modifier le contenu mais on va vite être à court de RAM avec ce principe. D'autre part la mise au point est difficile car le code html doit être intégré directement dans le code c++.
L'ESP partage sa mémoire Flash en deux : une partie pour le code et une autre partie pour un système de fichier appelé SPIFFS (SPI Flash File System). Avec 4Mo de flash, on a le choix entre plusieurs répartitions code/SPIFFS : 4Mo/0Mo, 3Mo/1Mo, 2Mo/2Mo ou 1Mo/3Mo.
1Mo de code est souvent suffisant pour la plupart des applications.

Procédure de chargement en SPIFFS :
Même si vous utilisez un autre IDE pour développer votre projet, le plus simple est d'utiliser l'IDE arduino pour charger des fichiers. Il faut d'abord installer une commande additionnelle à l'IDE arduino à télécharger à https://github.com/esp8266/arduino-esp8266fs-plugin/releases
La procédure d'installation est décrite à https://github.com/esp8266/arduino-esp8266fs-plugin
Créer un sketch arduino quelconque et créer un répertoire nommé **data** dans le répertoire de ce sketch. Les fichiers présent dans ce répertoire seront chargés dans la mémoire Flash de l'ESP par la commande "**Sketch data upload**" de l'IDE arduino (attention, la fenêtre "moniteur série" doit être fermée pendant le téléchargement).

- Étudier attentivement l'exemple "A-SPIFFS_File_server" du "ESP8266-beginer-guide" (https://github.com/tttapa/ESP8266) et le document "ESP8266 arduino Server Web SPIFFS et WebSocket" partie 1.

- Ajouter à votre programme la possibilité d'accéder à un fichier html statique stocké en SPIFFS en reprenant les fonctions de l'exemple précédent.

- Ajouter la possibilité de servir correctement des images au format jpeg en ajoutant la prise en compte du type MIME correspondant pour les extensions jpg , JPG , jpeg et JPEG.

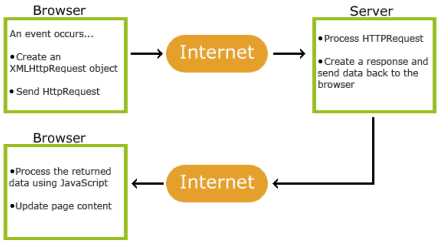
- Vérifier que vous pouvez accéder aux pages et images stockées dans `data`.

4. Des pages dynamiques !

Jusqu'à présent les pages étaient soit "statiques" stockées en Flash (code ou SPIFFS), soit générées en RAM sur demande par le serveur. La seule manière de rafraîchir la page est de la recharger en totalité. C'est acceptable pour une petite page mais ce n'est pas très fluide ni très réactif. Nous allons voir une méthode qui permettent un rafraîchissement très fluide en ne transmettant uniquement les informations à modifier.

4.1 AJAX : Asynchronous JavaScript and XML

AJAX est une technique qui permet de rafraîchir rapidement un page web en échangeant de petites quantités de données au format XML entre le client et le serveur par l'intermédiaire de requête HTTP. Voir : https://www.w3schools.com/asp/asp_ajax_intro.asp



Le navigateur charge la page HTML contenant un code javascript qui envoie une requête XMLHttp. Le serveur traite cette requête (lit un capteur, change un sortie,…) et crée une réponse qu'il retourne au navigateur. Dans le navigateur, le script JS met à jour la page en fonction de la réponse.

- Étudier l'exemple : `ajaxExemple.html`

Ce fichier contient une page html simple et un code JS qui exécute périodiquement la fonction `getUptime()` ci- dessous :

```
function getUptime() {
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
                if (this.readyState == 4 && this.status == 200) {
                // l'élement uptime de la page prend la valeur de la réponse
                document.getElementById("uptime").innerHTML = this.responseText;
                }
        };
        xhttp.open("GET", "getUptime", true);
        xhttp.send();
}
```

La requête html de type GET, `getUptime` est envoyé au serveur. Lorsque la réponse revient la fonction enregistrée dans le champ `onreadystatechange` de la variable `xhttp` s'exécute. Celle-ci récupère l'élément `uptime` de la page et y place le texte reçu `responseText.`

Dans le serveur, il faut enregistrer une fonction `handleUptime` qui répond à la requête `getUptime` :

```
server.on("/getUptime", handleUptime);
```

La fonction lit le uptime (millis) , le transforme en texte et l'envoie en mode texte au serveur :

```
void handleUptime() {
        String uptime = String(millis()/1000) ;
        server.send(200, "text/plain", uptime);
}
```

- Charger l'exemple en SPIFFS, ajouter le code qui répond à la requête `uptime` et vérifier le fonctionnement.

On veut maintenant ajouter le code qui répond à la requête de commande de la LED.
Pour cela il faut créer une "page" servie par une fonction enregistrée dans server.on() (ici setLED) qui exploite un argument (ici ledState).
En fonction de la valeur de l'argument la fonction exécute la commande (ici change l'état de la Led) et retourne le texte qui sera affiché dans la page.

-Ajouter le code permettant de commander la led et d'afficher l'état. Inspirez vous de la fonction `HandleNotFound` de l'exemple.

4.2 Une page dynamique qui affiche l'état de vos capteurs

- Intégrer l'accès SPIFFS à votre programme complet.
- En utilisant la méthode AJAX ajoutez une page qui affiche les données de vos capteurs et améliorez les commandes des leds.