ESP32 – WiFiManager – MQTT

Photo : Thomas Pesquet, ISS

Objectifs

- WiFiManager
- MQTT

Intérêts de WiFiManager

- Fournit un portail de configuration du réseau WiFi si la connexion ne fonctionne pas
- Le portail peut aussi servir à saisir d'autres configurations :
  - Adresse d'un service : API, MQTT, etc.
  - Nom d'utilisateur / mot de passe pour l'authentification d'un service

+ Simple à mettre en place

- Couteux en terme de poids dans la mémoire flash

Mise en place

- Importez la bibliothèque « https://github.com/tzapu/WiFiManager » dans Pio
- Créez un objet de type WiFiManager
- Utilisez la méthode « `autoConnect(<SSIDPortail>, <PasswordPortail>)` » : essaie de se connecter au WiFi si échec, démarre le portail de connexion avec le SSISPortail donné en paramètres
- Possibilité d'ajouter des paramètres avec la méthode « `addParameter(<Parametre>)` »

Exemple complet

```
#include <Arduino.h>
#include <WiFiManager.h>
#include <uri/UriRegex.h>

WiFiManager wm;
char const* SSIDPortail = "ConfigurationESP32";
char const* motPasseAPPortail = "Bonjour01.+";
IPAddress adresseIPPortail(192, 168, 23, 1);
IPAddress passerellePortail(192, 168, 23, 1);
IPAddress masqueReseauPortail(255, 255, 255, 0);
WebServer serveurWeb;
WiFiManagerParameter paramerePersonnalise("identifiant_unique_champ",
                                          "Nom du champ",
                                          "ValeurQuiEtaitSauvegardee", 40);
void setup() {
  Serial.begin(115200);

  wm.setDebugOutput(false);  // Mettre à true si vous avez des problèmes
  wm.setAPCallback([](WiFiManager* p_wiFiManager) {
    Serial.println("Connexion au réseau WiFi échouée, on lance le portail !");
  });

  wm.setConfigPortalTimeout(180);
```

Exemple complet

```
wm.setSaveParamsCallback([&paramerePersonnalise]() {
  Serial.println(
      "Sauvegarde de la configuration effectuée par l'utilisateur dans le "
      "portail...");
  Serial.println(String("Nouvelle valeur du paramètre : ") +
                 paramerePersonnalise.getValue());

  // Exemple d'actions...
  // Sauvegarde des données en JSON
  // Redémarrage : ESP.restart();
  // etc.
});

wm.addParameter(&paramerePersonnalise);

wm.setAPStaticIPConfig(adresseIPPortail, passerellePortail,
                       masqueReseauPortail);

wm.setParamsPage(true);

// Pour le débug, on peut forcer l'effacement de la configuration du WiFi
//wm.erase();
```

Exemple complet

```
  // Essaie de se connecter au réseau WiFi. Si échec, il lance le portail de
  // configuration. L'appel est bloquant -> rend la main après le timeout
  wm.autoConnect(SSIDPortail, motPasseAPPortail);

  // Pour lancer le portail manuellement
  // wm.startConfigPortal();

  serveurWeb.on(UriRegex("/.*"), []() {
    serveurWeb.send(200, "text/plain", "Bienvenue sur mon site web !");
  });

  if (WiFi.isConnected()) {
    serveurWeb.begin();
    Serial.println("Connecté au réseau : " + WiFi.SSID() +
                   " avec l'adresse : " + WiFi.localIP().toString());
  }
}

void loop() {
  if (WiFi.isConnected()) {
    serveurWeb.handleClient();
  }
}
```

MQTT

- Protocole standard normalisé par l'OASIS (Organization for the Advancement of Structured Information Standards)
- Protocole d'échanges de messages spécialisé pour l'internet des objets (IoT)
- Les messages sont publiés dans un sujet (topic)
- Les clients peuvent aussi s'abonner à un ou des sujets. Ils recevront alors les messages correspondants
- Protocole léger et donc code léger qui utilise peu de code et de données réseau
- Utilisé dans de grands domaines industriels tels que : automobile, la fabrication, les télécommunications, exploitation pétrolière, etc.

MQTT – Serveur

- Une implantation du serveur très utilisée est « Eclipse mosquitto » (https://mosquitto.org)
- Il est disponible en Docker.
- Pour ce module, nous allons nous baser sur l'image Docker en ajoutant l'authentification et en ajoutant deux utilisateurs.
- Pour utiliser l'image pré-configurée, utilisez le fichier docker-compose.yml donné dans le répertoire Git
- Vous pourrez visualiser ce qui se passe sur le serveur grâce à MQTT-Explorer accessible à partir de votre navigateur à l'adresse http://localhost:4000

MQTT – Connexion des clients

- Pour ce connecter à un service MQTT, il faut :
  - Adresse du serveur
  - Port : généralement 1883
  - Nom d'utilisateur / mot de passe
  - Sujet et message pour le testament (optionnel mais conseillé !) : si le client se déconnecte, le serveur envoie le message dans le sujet.
- Testament :
  - Généralement, on se sert de ce sujet pour donner la disponibilité d'un producteur. On a donc un sujet par producteur (i.e. par ESP32)
  - Algorithme général :
    - Déclaration d'un testament au moment de la connexion. Ex. sujet « monSuperPeripherique1234/availability » avec le message « offline »
    - Si la connexion est un succès, on envoie dans ce même sujet, le message « online »
    - On publie des messages et on s'abonne à des messages que l'on traite
    - Quand l'ESP32 passe hors ligne, le serveur MQTT exécute le testament, c'est-à-dire qu'il envoie dans le sujet « monSuperPeripherique1234/availability » le message « offline »

MQTT – Client

- Ajoutez la librairie « PubSubClient » dans le fichier « platformio.ini »

```
lib_deps =
        PubSubClient
```

- Création d'un objet de type « PubSubClient »

```
WiFiClient* espClient = nullptr;
PubSubClient* pubSubClient = nullptr;
String nomUnique;
unsigned long lastSentMessageDate = 0;

// ...
espClient = new WiFiClient();
pubSubClient = new PubSubClient(*espClient);
pubSubClient->setBufferSize(1024);
pubSubClient->setServer(MQTT_SERVER, MQTT_PORT);
nomUnique = String("ESP32Client") + String(ESP.getEfuseMac(), HEX);
```

MQTT – Client – Connexion

```
bool reconnectMQTTSiNecessaire() {
    if (!pubSubClient->connected()) {
        if (pubSubClient->connect(nomUnique.c_str(), MQTT_USER, MQTT_PASSWORD,
                                  (nomUnique + "/status").c_str(), 0, 0, "offline")) {
        Serial.println("Connecté au broker MQTT");
        pubSubClient->publish((nomUnique + "/status").c_str(), "online");

        // Si besoin – Ex. abonnement à « broadcast/# » (# jeton qui prend tout)
        pubSubClient->subscribe("broadcast/#");
        // Définition d'une fonction de traitement des messages
        pubSubClient->setCallback(
            [](char* topic, byte* payload, unsigned int length) {
                Serial.print("Message reçu...");
            });
        } else {
            Serial.print("Echec de connexion au broker MQTT : ");
        }
    }

    return pubSubClient->connected();
}
```

MQTT – Client – Publication messages

```
// Envoi de messages en continu
static unsigned long messageId = 0;

if (reconnectMQTTSiNecessaire()) {
    // Pour traiter les messages MQTT
    pubSubClient->loop();
    if (lastSentMessageDate + 5000 < millis()) {
        lastSentMessageDate = millis();
        String topic = nomUnique + "/data";
        String payload = String("Hello World! - ") + String(messageId++);
        pubSubClient->publish(topic.c_str(), payload.c_str());
    }
}
```

Références

- https://github.com/tzapu/WiFiManager : documentation de WiFiManager
- https://mqtt.org : documentation officielle