Application IoT sur Arduino

I. Introduction

A. Prérequis :

L'IDE Arduino doit être installée en mode portable de préférence. La chaine de développement pour Intel Curie correspondant à la carte Genuino utilisée doit être opérationnelle.

B. Objectifs :

Cette séance est consacrée à la mise en place de l'application :

- Installation et mise en œuvre d'un driver pour le module WIFI ESP8266 ;
- Connexion au réseau WiFi ;
- Installation et mise en œuvre d'une bibliothèque MQTT ;
- Connexion au broker ;
- Publication de messages ;
- Souscriptions aux topics ;
- Finalisation de l'application et commandes des LEDS et boutons-poussoirs ;
- Mise en évidence des problèmes de sécurité.

La démarche pas à pas est détaillée dans la partie suivante.

C. Validation :

En fin de séance, l'application doit être fonctionnelle.

II. Programmation de l'application

A. Vérification de la bonne configuration de la carte

- Module ESP8266 présent et orienté vers le connecteur USB ;
- Bornier de sélection J7 positionné sur RXDH ;
- Bornier de sélection J5 positionné sur TXDH ;
- Bornier de sélection J6 non connecté.

Valider la chaine développement en utilisant cette application minimaliste :

```
void setup()
{
     pinMode (LED_BUILTIN,OUTPUT);
}
void loop()
{
     digitalWrite(LED_BUILTIN,LOW);
     delay(200);
     digitalWrite(LED_BUILTIN,HIGH);
     delay(200);
}
```

Pour suivre la compilation et la programmation de la carte activez dans Fichiers→Préférences :

- Afficher les résultats détaillés pendant la compilation et le téléversement.

Pour valider cette étape, la LED D5 doit clignoter.

B. Ajout du module Wifi Esp8266

Le module Esp8266 permet d'ajouter une connectivité WiFi à la carte Arduino.
Cependant afin de rendre les communications compatibles avec l'API réseau Arduino il faut ajouter un pilote de périphérique spécifique.
Ce pilote de périphérique, fourni, est basé sur celui-ci disponible sur GitHub.
Il faudra cependant utiliser la version légèrement modifiée fournie dans ce fichier de bibliothèque Arduino.
Télécharger le pilote de périphérique dans la section **Ressources** du cours.
Pour l'ajouter à l'Ide Arduino : croquis→inclure une bibliothèque→ajouter une bibliothèque .zip
Créer un nouveau croquis.

Inclure le fichier d'entête du pilote de périphérique.

```
#include <WiFiEsp.h>
```

Vérifiez qu'il n'y a pas d'erreurs. Si nécessaire, nommer ce nouveau programme app_mqtt.

Pour valider cette étape le programme doit compiler sans erreurs

C. Initialisation du module Wifi

Créez une fonction `WifiConnect` qui sera appelée à partir de la fonction setup.
Pour fonctionner, le module ESP doit être commandé par un port série configuré à 9600 bauds.
Initialiser le port série `Serial1` à 9600 bauds :

```
Serial1.begin(9600);
```

et attendre que le port soit configuré:

```
while(!Serial1);
```

Le port `Serial` différent du port `Serial1` est utilisé par le pilote de périphérique pour donner des informations sur la communication. Les informations sont visibles sur le moniteur série de l'Ide Arduino (voir outils).
Initialiser ce port à 9600 et attendre la fin de sa configuration.
Initialiser le module WIFI en spécifiant une référence sur le port de communication série à utiliser (toujours dans la fonction `WifiConnect`):

```
WiFi.init(&Serial1);
```

Exécuter le programme.

Pour valider cette étape, l'initialisation du module WiFi doit être correcte en regardant le moniteur série de l'IDE Arduino.

D. Connexion au réseau Wifi

Pour ce TP un réseau WIFI spécifique est mis en place, ce réseau a comme SSID (pour le TP1 : "lab_IoT" et pour les autres TP : "ZZ_HSH" - Home Sweet Home -)
Le mot de passe protégeant la connexion sera affiché au tableau.
Démarrer la connexion du module ESP8266 au réseau WIFI dans `WifiConnect` :

```
WiFi.begin((char*)"lab_IoT","XXX");
WiFi.begin((char*)"ZZ_HSH","XXX");
```

Pour valider cette étape, une adresse IP de la forme XXX.XXX.XXX.XXX est obtenue par le Module Esp8266 (voir moniteur série)

E. Compatibilité protocole MQTT et module ESP8266

La librairie utilisée est basée sur la librairie officielle de MQTT projet Paho.

Cependant, quelques modifications ont dû être apportées pour garantir le fonctionnement avec le module ESP8266 en mode AT.

Dans la section **Ressources**, télécharger cette bibliothèque et l'ajouter à IDE Arduino.

Ajouter les fichiers d'entête nécessaire au fonctionnement de la librairie MQTT :

```
#include <Countdown.h>
#include <IPStack.h>
#include <MQTTClient.h>
```

Pour valider cette étape, la vérification du croquis (compilation) doit se dérouler erreurs.

F. Pile TCP/IP

Le client MQTT pour fonctionner utilise l'accès à la pile TCP/IP.

La pile TCP/IP (`IPStack`) utilise une instance de pilote de périphérique réseau qui doit être le pilote du périphérique WIFI utilisé.

Il faut donc déclarer les variables globales suivantes :

- Une instance du pilote de périphérique WiFi ;
- La pile TCP/IP (`IPStack`) utilisant ce pilote de périphérique ;
- Le client utilisant cette connexion WIFI.

Ajouter ces lignes à votre programme :

```
WiFiEspClient c;
IPStack ipstack(c);
MQTT::Client<IPStack, Countdown> client =
MQTT::Client<IPStack, Countdown>(ipstack);
```

Pour valider cette étape, le programme doit compiler sans erreurs.

G. Connexion au serveur MQTT

Un broker est mis à disposition pour ce TP.

Pour le TP1 :

- Il est disponible à l'adresse test.mosquitto.org et sur le port 1883.

Pour les autres TP :

- Il est disponible à l'adresse suivante `192.168.1.136` à partir du réseau Wifi ;
- Il est en écoute sur le port `10855`.

Dans une fonction `BrokerConnect` que vous appellerez depuis la fonction `setup`, démarrez la connexion TCP avec le broker et affichez le code d'erreur le cas échéant :

```
ipstack.connect((char *)"Adresse broker", port );
```

Pour valider cette étape, la tentative de connexion est « OK »

H. Connexion au broker

A partir du TP2, la configuration actuelle du broker n'est pas sécurisée, il accepte les connexions anonymes. Pour se connecter au broker il suffit donc d'exécuter :

```
client.connect();
```

Le code de retour est égal à 0 si la connexion se fait sans erreur. Néanmoins durant le TP1 un nom d'utilisateur et un mot de passe sera nécessaire et cette exécution va générer une erreur.

Ajoutez la connexion au broker dans `BrokerConnect` et affichez un message (`Serial.println("xxx")`) pour indiquer le succès ou l'échec de la connexion.

Pour le TP1, il faudra réaliser une connexion au broker en utilisant des informations de configuration :

```
client.connect(configMQTT);
```

Où `configMQTT` est une structure de type `MQTTPacket_connectData` initialisée avec `MQTTPacket_connectData_initializer`.

Le champ `configMQTT.username.cstring` permet de définir le nom d'utilisateur "GX" (à transtyper en (char*)) et le champ `configMQTT.password.cstring` permet de définir le mot de passe "isimaGX" (à transtyper en (char*))

```
MQTTPacket_connectData configMQTT =
MQTTPacket_connectData_initializer;
```

```
configMQTT.username.cstring = (char*)"GX";
configMQTT.password.cstring = (char*)"isimaGX";
```

Remarque : Votre nom d'utilisateur sera "GX", où X est votre numéro de groupe, et votre mot de passe sera "isimaGX".

Pour valider la connexion, demandez à l'enseignant, l'information est disponible dans les journaux. Veuillez lui fournir l'adresse IP de votre carte

Pour valider l'étape, votre connexion au broker doit être validée.

I. Configuration des entrées/sorties de la carte

Le détail de la carte fille (shield) Arduino est donné dans la section ressources Document technique carte Shield Wifi Esp8266. La carte est configurée comme suit :

- Bouton-Poussoir 1 connecté à la broche 8
- Bouton-Poussoir 2 connecté à la broche 9
- LED 1 connectée à la broche 12
- LED 2 connectée à la broche 13

En utilisant la fonction `pinMode`, initialisez correctement (entrée (INPUT) / sortie (OUTPUT)) les broches dans une fonction `PortsSetup()` que vous appellerez à partir de `setup()`.

Dans la boucle principale, copiez l'état du bouton-poussoir 1 sur la LED 1 (idem pour le bouton-poussoir 2 et la LED 2).

Pour lire la valeur d'une entrée il faut utiliser

```
digitalRead(numéro de broche)
```

Pour spécifier la valeur d'une sortie utilisez

```
digitalWrite(numéro de broche,valeur)
```

La boucle principale du programme intègrera une temporisation de 200ms : `delay(200)`.

Pour valider l'étape, les boutons poussoirs doit fonctionner en logique inversée (la valeur lue est 1 lorsqu'ils sont relâches).

J. Modification par un changement d'état

L'état des boutons poussoirs ne sera pas critique dans le reste de l'application. Seul le changement d'état est important.

Modifiez votre programme afin que l'état des LEDS soit inversé lors de uniquement lors de l'activation (passage de 1 à 0) du bouton poussoir correspondant.

Réaliser les modifications dans la partie `loop()` du programme.

Pour valider l'étape, lors d'un appui, la LED s'allume, à l'appui suivant elle s'éteint.

K. Publication dans le broker

Nous souhaitons maintenant reporter l'information correspondant à l'appui d'un bouton poussoir. Pour cela, un message va être publié lors de chaque changement d'état.

Le message sera publié sur le **topic** suivant :

isima/GX/BPs/BPn où n=[1|2]

Le message publié sera "ON" dans le cas d'un appui sur le bouton et "OFF" lors de son relâchement.

Pour publier un message, vous utiliserez la fonction `publish` :

```
client.publish("topic",(void *)"valeur",longueur de la chaine valeur);
```

Pour valider l'étape, vos publications doit apparaitre dans le journal du broker

L. Mise en place des callbacks

Afin d'assurer la téléopérabilité de l'application, il faudrait que chaque modification de la valeur d'un bouton poussoir sur le broker soit reportée à l'application. Pour cela, il va falloir surveiller l'ensemble du topic BP pour la qualité de service 0.

Pour demander la surveillance du topic de BP1 il faut utiliser la fonction :

```
client.subscribe("/ISIMA/TP_03/GROUPE_NOM1_NOM2/BPs/BP1", MQTT::QOS0, CallBack);
```

La fonction `CallBack` a comme prototype : `void CallBack(MQTT::MessageData& md)` et se contentera, dans un premier temps d'afficher le message "Nouveau Message Arrivé" (`Serial.println()`).

Enfin la gestion des souscriptions exige que la connexion avec le broker soit surveillée régulièrement. Pour cela, remplacez le délais par : `client.yield(200);` dans la boucle principale du programme.

Proposer un programme permettant de surveiller les deux topics correspondant à BP1 et BP2 avec une seule souscription et une seule fonction de `CallBack`.

Pour valider l'étape, votre programme doit être fonctionnel

M. Actions sur les LEDs via un message MQTT

Lors de la réception d'un message le champ : `md.topicName.lenstring.data` contient le topic, attention la longueur de se champ est `md.topicName.lenstring.len`, ce n'est pas une chaine standard du langage C elle n'est pas terminée par '\0'.

Le champ : `md.message.payload` contient le contenu du message (au format (`void *`)) il faudra le transtyper en (`char *`).

Renommer la fonction `CallBack` en `CallBackBpMessageHandler`. Modifier la fonction `BrokerConnect` en conséquence.

Supprimez la gestion des LEDS de la boucle `loop()`.

Modifiez la fonction de `CallBack` afin que l'état de la LED ne soit inversé que lors de la réception d'un message contenant le message "ON" sur le topic correspondant à BP1 pour la LED 1 et BP2 pour la LED 2.

Pour valider l'étape, votre programme doit être fonctionnel

N. Commande des LEDs

Les LEDS devront pouvoir être commandées à distance. Pour cela, leur commande se fera après réception d'un message sur un topic spécifique.

Le schéma à implanter est le suivant :

Lorsque l'appui sur un bouton poussoir a été signalé, un message sur le topic:

isima/GX /LEDs/LEDn où n=[1|2]

est publié avec la valeur "ON" ou "OFF".

L'application a souscrit à ce même topic. Lors de la réception d'un message sur ce topic, l'état de la LED est modifié en conséquence.

Attention :Il n'est pas conseillé d'envoyer un message depuis un CallBack (la bibliothèque n'est pas threadSafe). Il est préférable de positionner un drapeau dans le CallBack et d'envoyer le message depuis la boucle principale en fonction de l'état du drapeau.

L'application finale contiendra deux `CallBack` un pour les boutons-poussoirs et le deuxième pour gérer les LEDs.

Pour valider l'étape, votre programme doit être fonctionnel

O. Configuration MQTT : Client ID

Le protocole MQTT exige que chaque client soit différencié. Pour cela lors de la connexion un nom de client peut être spécifié.

La connexion au broker se fait alors en utilisant des informations de configuration :

```
client.connect(configMQTT);
```

Où `configMQTT` est une structure de type `MQTTPacket_connectData` initialisée avec `MQTTPacket_connectData_initializer`. Le champ `configMQTT.clientID.cstring` permet de définir le nom de client "GROUPE_NOM1_NOM2" (à transtyper en (`char*`)).

Pour valider l'étape, vérifier que le nom de votre groupe apparait dans les journaux du broker comme client.

P. Accès à un autre IoT par la carte Arduino

Modifier votre programme afin qu'il modifie l'état de la LED2 d'un IoT d'un autre groupe.

Pour valider l'étape, vous devez être capable d'allumer la LED2 d'un autre groupe sans accéder à son IoT.

Q. Accès à l'IoT par un script Python

Écrire un script en Python qui modifie l'état des LEDS de votre IoT. Le script allumera alternativement chaque LED pendant 250ms.

Pour valider l'étape, vous devez être capable de commander la LED2 de votre IoT par un script Python

R. Utilisation des fonctionnalités MQTT

Faire les manipulations suivantes :

- Appuyer sur les boutons poussoirs afin que la LED 1 soit allumée et la LED 2 éteinte ;
- Téléversez de nouveau le programme sur la cible.

Vous constaterez que lors du nouveau démarrage les deux LEDS sont éteintes, l'état, bien que reporté correctement au broker n'est pas persistant.

Proposer une solution assurant que lors de la mise en route du système, la dernière configuration des LEDS soit restaurée.

Attention : votre solution doit utiliser le protocole MQTT (pas une EEPROM locale par exemple).

Pour valider l'étape, l'état des LEDS est bien restauré après réinitialisation du programme.

S. Utilisation d'un tableau de bord pour commander l'IoT

Installez un client MQTT (DASH MQTT, LINEAR MQTT Dashboard ou autre) sur votre Smartphone, configurez-le afin qu'il commande l'état des LEDS ou simule l'appui sur un bouton poussoir.

Pour valider l'étape, la commande des LEDs soit se faire à l'aide de votre smartphone.

III. Exemple d'une solution avec toutes les fonctionnalités demandées

```
#include <SPI.h>
#include <Countdown.h>
#include <IPStack.h>
#include <MQTTClient.h>

#include <WiFiEsp.h>

// Variables => pins des input et output
const int led_D5 = 13;
const int led_D6 = 12;
const int bouton_S2 = 8;
const int bouton_S3 = 9;

bool ChangeLed_D5=false, ChangeLed_D6=false;
bool Bp_S2 =true, Bp_S3 =true;

WiFiEspClient c;
IPStack ipstack(c);
MQTT::Client<IPStack, Countdown> client =
MQTT::Client<IPStack, Countdown>(ipstack);

// Noms des topics
String subLED  = "isima/GX/LEDs/#";
String subBP   = "isima/GX/BPs/#";
String pubBP1  = "isima/GX/BPs/BP1";
String pubBP2  = "isima/GX/BPs/BP2";
String pubLED1 = "isima/GX/LEDs/LED1";
String pubLED2 = "isima/GX/LEDs/LED2";

void CallBackBpMsg(MQTT::MessageData& md)
{
  MQTT::Message &message = md.message;
  MQTTString &topic = md.topicName;

  Serial.print("Message BP arrived: qos ");
  Serial.print(message.qos);
  Serial.print(", retained ");
  Serial.print(message.retained);
  Serial.print(", dup ");
  Serial.print(message.dup);
  Serial.print(", packetid ");
  Serial.println(message.id);

  char* topicName = new char[topic.lenstring.len+1]();
  memcpy(topicName,topic.lenstring.data,topic.lenstring.len);
  Serial.print(", topic ");
```

```
  Serial.println(topicName);

  char* msgPL = new char[message.payloadlen+1]();
  memcpy(msgPL,message.payload,message.payloadlen);
  Serial.print("Payload ");
  Serial.println(msgPL);

  if(!strncmp(&topic.lenstring.data[topic.lenstring.len-
3],"BP1",3))
    if(!strncmp(msgPL,"ON",2))
      ChangeLed_D6=true;

  if(!strncmp(&topic.lenstring.data[topic.lenstring.len-
3],"BP2",3))
    if(!strncmp(msgPL,"ON",2))
      ChangeLed_D5=true;

  delete msgPL;
  delete topicName;
}

void CallBackLedMsg(MQTT::MessageData& md)
{
  MQTT::Message &message  = md.message;
  MQTTString &topic = md.topicName;

  Serial.print("Message LED arrived: qos ");
  Serial.print(message.qos);
  Serial.print(", retained ");
  Serial.print(message.retained);
  Serial.print(", dup ");
  Serial.print(message.dup);
  Serial.print(", packetid ");
  Serial.println(message.id);

  char* topicName = new char[topic.lenstring.len+1]();
  memcpy(topicName,topic.lenstring.data,topic.lenstring.len);
  Serial.print(", topic ");
  Serial.println(topicName);

  char* msgPL = new char[message.payloadlen+1]();
  memcpy(msgPL,message.payload,message.payloadlen);
  Serial.print("Payload ");
  Serial.println(msgPL);

  if (!strncmp(&topic.lenstring.data[topic.lenstring.len-
4],"LED1",4))
    if (!strncmp(msgPL,"ON",2))
      digitalWrite(led_D6,HIGH);
    else
      digitalWrite(led_D6,LOW);
```

```
  if (!strncmp(&topic.lenstring.data[topic.lenstring.len-
4],"LED2",4))
    if (!strncmp(msgPL,"ON",2))
      digitalWrite(led_D5,HIGH);
    else
      digitalWrite(led_D5,LOW);

  delete msgPL;
  delete topicName;
}
void WifiConnect()
{
  Serial1.begin(9600);
  while(!Serial1);

  Serial.begin(9600);
  while(!Serial);

  WiFi.init(&Serial1);
  WiFi.begin((char*)"lab_IoT","…");
}

void BrokerConnect()
{
  MQTTPacket_connectData configMQTT =
MQTTPacket_connectData_initializer;
  configMQTT.clientID.cstring = (char*)"GROUPE_X";
  configMQTT.username.cstring = (char*)"GX";
  configMQTT.password.cstring = (char*)"…";
  configMQTT.willFlag = 0;

  ipstack.connect((char *)"test.mosquitto.org",1883);

  int rc = client.connect(configMQTT);

  if(rc == 0)
    Serial.println("Connected OK");
  else
    Serial.println("Not Connected ERROR");

  client.subscribe(subBP.c_str(), MQTT::QOS0, CallBackBpMsg);
  client.subscribe(subLED.c_str(),MQTT::QOS0, CallBackLedMsg);
 }

void PortsSetup()
{
  // initialisation des broches 12 et 13 comme étant des
sorties
  pinMode(led_D5, OUTPUT);
  pinMode(led_D6, OUTPUT);
```

```
  // initialisation des broches 8 et 9 comme étant des entrées
  pinMode(bouton_S2, INPUT);
  pinMode(bouton_S3, INPUT);
}

void setup() {
  WifiConnect();
  BrokerConnect();
  PortsSetup();
}
void loop() {
  client.yield(100);

  if (Bp_S2 != digitalRead(bouton_S2))
  {
    Bp_S2 = !Bp_S2;

    MQTT::Message message;
    message.qos = MQTT::QOS0;
    message.retained = false;
    message.payload = (void *)(Bp_S2?"OFF":"ON");
    message.payloadlen = strlen(Bp_S2?"OFF":"ON");

    client.publish(pubBP1.c_str(),message);
  }
  if (Bp_S3 != digitalRead(bouton_S3))
  {
    Bp_S3 = !Bp_S3;

    MQTT::Message message;
    message.qos = MQTT::QOS0;
    message.retained = false;
    message.payload = (void *)(Bp_S3?"OFF":"ON");
    message.payloadlen = strlen(Bp_S3?"OFF":"ON");

    client.publish(pubBP2.c_str(),message);
  }

  if(ChangeLed_D6)
  {
    MQTT::Message message;
    message.qos = MQTT::QOS0;
    message.retained = true;
    message.payload = (void
*)(!digitalRead(led_D6)?"ON":"OFF");
    message.payloadlen =
strlen(!digitalRead(led_D6)?"ON":"OFF");

    client.publish(pubLED1.c_str(),message);

    ChangeLed_D6=false;
```

```
  }
  if(ChangeLed_D5)
  {
    MQTT::Message message;
    message.qos = MQTT::QOS0;
    message.retained = true;
    message.payload = (void
*)(!digitalRead(led_D5)?"ON":"OFF");
    message.payloadlen =
strlen(!digitalRead(led_D5)?"ON":"OFF");

    client.publish(pubLED2.c_str(),message);

    ChangeLed_D5=false;
  }
}
```