Laboratorio TECNOLOGIE PER IOT – HW PARTE 3

Gatti Gabriele s245636

Iobbi Amedeo s228856

Esercizio 1

L'esercizio richiede di implementare, mediante la connettività della scheda Arduino Yun, un server in grado di fornire all'utente informazioni ricevute dal sensore di temperatura e il controllo di un led mediante metodo GET.

```
#include <math.h>
#include <Bridge.h>
#include <BridgeServer.h>
#include <BridgeClient.h>
#include <ArduinoJson.h>
#define TEMP_PIN A1
BridgeServer srv;
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(TEMP_PIN, INPUT);
  digitalWrite(LED_BUILTIN, HIGH);
  Bridge.begin();
  digitalWrite(LED_BUILTIN, LOW);
  srv.listenOnLocalhost();
  srv.begin();
}
```

Il setup dello sketch si occupa di inizializzare le periferiche, ovvero il sensore di temperatura e il led, per quest'ultimo viene scelto di utilizzare il led integrato nella scheda, identificato dalla costante `LED_BUILTIN`, lo stesso led viene utilizzato per segnalare l'inizializzazione della connessione con il processore Linux della scheda Yun.

Viene inoltre istanziato un oggetto della classe `BridgeServer srv` allo scopo di gestire le connessioni in entrata verso la scheda.

```
void loop() {
  BridgeClient clt = srv.accept();

  if(clt){
    processRequest(clt);
    clt.stop();
  }
  delay(50);
}
```

Il loop si occupa di processare le richieste in ingresso su localhost mediante la funzione *processRequest*. Successivamente la connessione viene chiusa e viene introdotto un *delay* di 50 ms atto ad evitare il sovraccarico della scheda.

```
void processRequest(BridgeClient clt){
  String cmd = clt.readStringUntil('/');
  cmd.trim();
```

```
  if(cmd == F("led")){
    handleLed(clt);
  }else if(cmd == F("temperature")){
    handleTemp(clt);
  }else{
    handleError(clt, 404);
  }
}
```

Per processare la connessione innanzitutto viene eseguito un parsing dell'uri richiesto. A seconda che il path sia `led` oppure `temperature`, viene richiamata la funzione handler richiesta. Nel caso in cui si acceda ad un path non disponibile, viene richiamato *handleError* con codice d'errore `404`, corrispondente allo status `Not Found`.

```
void handleLed(BridgeClient clt){
  unsigned int val = clt.parseInt();
  if(val == 0 || val == 1){
    digitalWrite(LED_BUILTIN, val);
    printResponse(clt, 200, senMlJson( F("led"), val, F("")));
  }else{
    handleError(clt, 400);
  }
}
```

handleLed si occupa della gestione del led. Viene effettuato il parsing della seconda posizione dell'uri, essa può assumere solamente valore `0` o `1`. Il valore logico viene pertanto scritto sul pin del led sotto forma di tensione, determinando lo spegnimento o l'accensione del suddetto. Viene infine richiamata la funzione di gestione delle risposte al client *printResponse*. Essa invierà al richiedente le informazioni sullo stato del led, in formato `Json` grazie alla funzione *senMlJson* che verrà analizzata in seguito.

Nel caso in cui il valore non sia accettabile viene richiamato *handleError* con codice d'errore `400`, corrispondente allo status `Bad Request`.

Le stringhe statiche vengono il più possibile inserite nella macro *F* che provvede a memorizzarle nella memoria FLASH della scheda anziché in memoria centrale, così da risparmiare spazio per l'esecuzione del codice.

```
void handleTemp(BridgeClient clt){
  if(clt.peek()!= -1){
    handleError(clt, 404);
  }else{
    printResponse(clt, 200, senMlJson( F("temperature"), readTemp(), F("Cel")));
  }
}
```

handleTemp si occupa della trasmissione dei dati ottenuti mediante il sensore di temperatura. Mediante il metodo *peek* richiamato sull'oggetto client ci si assicura che non ci siano più caratteri nella uri richiesta, in caso contrario viene nuovamente richiamato l'handler dello status `Not Found`, altrimenti si procede fornendo il dato richiesto mediante la funzione *readTemp*, la medesima implementata nei laboratori HW precedenti; anch'esso formattato in `Json`, con lo standard SenML.

```
void handleError(BridgeClient clt, unsigned int err_code){
  if(err_code == 400){
    printResponse(clt, err_code, F("400 Bad Request"));
```

```
  }else if(err_code == 404){
    printResponse(clt, err_code, F("404 Not Found"));
  }
}
```

L'handler degli errori si occupa di identificare lo status raggiunto verificando i codice d'errore e successivamente chiama la funzione di risposta fornendo il body adeguato al tipo d'errore.

```
void printResponse(BridgeClient clt, unsigned int code, String body){
  clt.println("Status: " + String(code));
  if(code == 200){
    clt.println(F("Content-type: application/json; charset=utf-8"));
  }else{
    clt.println(F("Content-type: text/plain; charset=utf-8"));
  }
  clt.println();
  clt.println(body);
}
```

printResponse elabora gli header del pacchetto stampando prima lo status e poi in base allo stato della risposta imposta il tipo di contenuto in modo adeguato, `application/json` per i dati trasmessi dalla scheda, `text/plain` per i messaggi di errore. Infine viene inserito nel pacchetto il contenuto vero e proprio della risposta, memorizzato nel parametro *body*.

```
const int jsonCap = JSON_OBJECT_SIZE(6) + JSON_ARRAY_SIZE(1) + 40;
DynamicJsonDocument jsonOut(jsonCap);

String senMlJson(String obj, float value, String unit){
  jsonOut.clear();
  jsonOut["bn"] = "Yun";
  jsonOut["e"][0]["n"] = obj;
  jsonOut["e"][0]["t"] = millis();
  jsonOut["e"][0]["v"] = value;
  if(unit != ""){
    jsonOut["e"][0]["u"] = unit;
  }else{
    jsonOut["e"][0]["u"] = (char*)NULL;
  }

  String out;
  serializeJson(jsonOut, out);
  return out;
}
```

Infine, la funzione *senMlJson* gestisce la formattazione dei dati come JSON secondo lo standard SenML. Per prima cosa viene inizializzata la variabile `jsonCap` alla dimensione massima che può raggiungere l'oggetto *jsonOut*, che conterrà i dati formattati. Esso viene quindi preventivamente svuotato mediante il metodo *clear* in modo da evitare memory leakage come suggerito dal professore; si procede poi ad inserirne gli attributi e valori secondo le specifiche fornite dallo standard e dall'esercizio.

L'oggetto viene poi serializzato mediante *serializeJson* e memorizzato in una stringa che verrà ritornata al chiamante.

Esercizio 2

Lato Arduino

In questo esercizio è richiesto di scrivere uno sketch che permetta alla scheda di comunicare dati ricevuti dal sensore di temperatura, sfruttando il sottosistema Linux, ad un server realizzato mediante la libreria cherrypy utilizzata nei laboratori precedenti.

```
void setup() {
  //led utilizzato per verificare l'avvio della connessione bridge
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(TEMP_PIN, INPUT);
  digitalWrite(LED_BUILTIN, HIGH);
  Bridge.begin();
  digitalWrite(LED_BUILTIN, LOW);
  Serial.begin(9600);
}
```

Il setup è il medesimo dell'esercizio precedente, con l'aggiunta dell'inizializzazione della connessione seriale per funzionalità di logging di eventuali errori da parte del processo che verrà richiamato dal processore Linux.

```
void loop() {
  String jsonValue = senMLJson(readTemp());
  unsigned int retval = postRequest(jsonValue);
  if(retval != 0){
    Serial.print("Something went wrong, curl return value was ");
    Serial.println(retval);
  }
  delay(2000);
}
```

La funzione di loop elabora e formatta in JSON i dati ricevuti dal sensore di temperatura mediante le funzioni descritte nei paragrafi precedenti, per poi eseguire una richiesta POST mediante la funzione `postRequest` che ha come unico argomento la stringa precedente. Questa funzione in caso di successo restituisce `0`, nel caso in cui il valore sia diverso viene mostrato tramite l'interfaccia seriale.

Questa sequenza di operazioni viene eseguita ogni 2 secondi allo scopo di non sovraccaricare il server di richieste e soprattutto in quanto un intervallo temporale minore comporterebbe l'accumulo di rilevazioni di valori di temperatura identici tra di loro.

```
const String target = "192.168.1.12:8080/log";
int postRequest(String data){
  Process p;
  p.begin("curl");
  p.addParameter("-H");
  p.addParameter("Content-Type: application/json");
  p.addParameter("-X");
  p.addParameter("POST");
  p.addParameter("-d");
  p.addParameter(data);
  p.addParameter(target);
  p.run();
  return p.exitValue();
}
```

Questa funzione si occupa di istanziare un processo, corrispondente ad un eseguibile installato nel sottosistema Linux, `curl`, un'utility la cui funzionalità è eseguire richieste del protocollo HTTP personalizzate. In questo caso i parametri specificano anche altre informazioni della richiesta che si sta eseguendo:

- `-H Content-Type: application/json`: specifica di inserire l'header `Content-Type` con il valore specificato affinchè il server sia in grado di riconoscere che i dati inviati sono formattati secondo lo standard JSON.
- `-X POST`: indica il metodo da utilizzare per la richiesta.
- `-d data`: specifica che il contenuto della richiesta è il contenuto della variabile `data`, ossia la stringa JSON formattata secondo lo standard SenML.
- `target`: infine è l'indirizzo del server a cui inviare la richiesta, in questo caso si tratta dell'indirizzo del laptop su cui è avviato il server all'interno della rete casalinga, nello specifico deve essere eseguita sul path `/log`.

Lato Server

Passando ora al lato server viene richiesto di estendere gli esercizi 1 e 2 del laboratorio Software, ma si è deciso di reimplementare completamente il sistema in quanto le funzionalità dei due esercizi precedenti non sono prettamente necessarie e il codice risulta più leggero e di facile comprensione.

```
if __name__ == "__main__":
    conf={
        '/':{
            'request.dispatch':cherrypy.dispatch.MethodDispatcher(),
            'request.show_tracebacks': False,
        }
    }
    cherrypy.tree.mount(TemperatureServer(),'/',conf)
    cherrypy.config.update({'server.socket_host': '0.0.0.0'})
    cherrypy.engine.start()
    cherrypy.engine.block()
```

Viene mantenuta la configurazione standard di cherrypy con la disabilitazione delle informazioni di Traceback per questioni di sicurezza. Viene inoltre aggiunta l'istruzione `cherrypy.config.update({'server.socket_host': '0.0.0.0'})` affinchè il server sia raggiungibile attraverso tutti gli indirizzi IP della macchina e non più solo mediante `127.0.0.1` o `localhost`.

```
import cherrypy
import json

class TemperatureServer():
    exposed = True
    def __init__(self):
        self.temps = {}
        self.temps["list"] = []
```

La classe TemperatureServer viene esposta sul path / del server; alla sua inizializzazione l'attributo self.temps viene definito come un dizionario il cui unico elemento, corrispondente alla chiave "list" è una lista vuota.

```
    @cherrypy.tools.json_out()
    def GET(self, *uri, **params):
        if uri == ('log', ):
            return self.temps
        else:
            raise cherrypy.HTTPError(404)
```

Il metodo GET si occupa semplicemente di ritornare il dizionario appena inizializzato sotto forma di JSON grazie al decoratore @cherrypy.tools.json_out; il metodo si occupa inoltre di restituire la pagina di errore 404 Not Found a qualsiasi richiesta che non viene effettuata sul path /log, effettuando controlli sull'URI della richiesta.

```
    @cherrypy.tools.json_in()
    def POST(self, *uri, **params):
        if uri == ('log', ):
            json_temp = cherrypy.request.json
            try:
                keys = json_temp.keys()
                assert "bn" in keys and "e" in keys
                assert json_temp["bn"] == "Yun"

                keys = json_temp["e"][0].keys()
                assert "n" in keys and "t" in keys and "v" in keys and "u" in keys
                assert json_temp["e"][0]["n"] == "temperature" and json_temp["e"][0]["u"]
== "Cel"
                assert json_temp["e"][0]["t"] != None and json_temp["e"][0]["v"] != None
            except:
                raise cherrypy.HTTPError(400)

            self.temps["list"].append(json_temp)
        else:
            raise cherrypy.HTTPError(404)
```

Il metodo POST effettua anch'esso controlli sull'URI affinchè il path /log sia l'unico interrogabile, successivamente estrae il contenuto JSON della richiesta già sottoforma di dizionario grazie al decoratore @cherrypy.tools.json_in ed effettua una serie di controlli mediante assert allo scopo di verificare l'integrità dei dati ricevuti. In caso di fallimento di uno dei controlli viene visualizzata la pagina di errore 400 Bad Request, altrimenti il valore ricevuto viene aggiunto alla lista presente nel dizionario self.temps in modo tale che possa essere visualizzata dagli utenti che effettuano richieste GET al medesimo indirizzo.

Esercizio 3

```
#include <ArduinoJson.h>
#include <MQTTclient.h>
#include <Process.h>

#define MQTT_HOST F("test.mosquitto.org")

#define LED_PIN 10
#define TEMP_PIN A1

String my_base_topic = "/tiot/22";
float temp;

/*====== Costanti sensore temperatura ======*/
const long int R0 = 100000; //Ohm
const int B = 4275; //K
const float Vcc = 1023.0;
float R = 0.0;
float T = 0.0; //Kelvin
/*==========================================*/

const int capacity = JSON_OBJECT_SIZE(6) + JSON_ARRAY_SIZE(1) + 40;
DynamicJsonDocument doc_rec(capacity);
DynamicJsonDocument doc_snd(capacity);

//Funzione per leggere i valori di temperatura dal sensore
float readTemp() {

  int sensorValue = analogRead(TEMP_PIN); // = Vsig
  R = ((Vcc / sensorValue) - 1.0) * R0;
  T = (1 / ((log(R / R0) / B) + (1 / 298.15))) - 273.15; // Temperatura in gradi
celsius

  return T;
}
void setLED(const String& topic, const String& subtopic, const String& message) {
  DeserializationError err = deserializeJson(doc_rec, message);
  if (err) {
    Serial.print(F("deserializeJson() failed with code: "));
    Serial.println(err.c_str());
  }

  if (doc_rec[F("e")][0][F("n")] == F("led")) {

    int value = int(doc_rec[F("e")][0][F("v")]);
    switch (value) {
      case 0:
        digitalWrite(LED_PIN, LOW);
        break;
      case 1:
        digitalWrite(LED_PIN, HIGH);
        break;
      default:
        Serial.println(F("Valore per attivare/disabilitare il led non corretto"));
        break;
    }
  }
}

String senMlEncode (String title, float value, String unit) {

  doc_snd.clear();
  doc_snd[F("bn")] = "Yun";
```

```
  doc_snd[F("e")][0][F("n")] = title;
  doc_snd[F("e")][0][F("t")] = millis();
  doc_snd[F("e")][0][F("v")] = value;

  if (unit != "") {
    doc_snd[F("e")][0][F("u")] = unit;
  } else {
    doc_snd[F("e")][0][F("u")] = (char*)NULL;
  }

  String output;
  serializeJson(doc_snd, output);

  return output;
}

void setup() {

  Serial.begin(9600);

  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);

  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);
  Bridge.begin();
  digitalWrite(LED_BUILTIN, LOW);

  mqtt.begin(MQTT_HOST, 1883);
  mqtt.subscribe(my_base_topic + F("/led"), setLED);

}

void loop() {

  mqtt.monitor();

  //Leggo la temperatura
  temp = readTemp();

  //Continuo ad inviare i valori di temperatura misurati
  String message = senMlEncode(F("temperaure"), temp, F("Cel"));
  mqtt.publish(my_base_topic + F("/temperature"), message);

  delay(5000);

}
```

Nell' esercizio 3 dobbiamo far comunicare la Yun con via MQTT, agendo sia da publisher che da **subscriber**, per farlo utilizziamo come **hostname** quello messo a disposizione da **mosquitto**.

Per comunicare via MQTT utilizziamo la libreria ***MQTTclient***.

Nella funzione di setup dell' arduino inizializziamo mqtt sul host e sulla porta indicati e specifichiamo anche il topic del mio **subscriber** indicando anche la funzione ***setLED*** per gestire il led, che verrà richiamata ogni volta che faremo una **subscribe**.

La stringa passata all' interno della funzione sarà in formato **JSON**, attraverso la libreria **ArduinoJson** e dalle funzioni che mette a disposizione controlliamo che la stringa passata sia scritta in formato corretto, in caso affermativo andiamo a prendere il valore presente nel campo led e in base a quest' ultimo decidiamo se accendere o spegnere il led.

Nella funzione loop invece continuiamo a monitorare attraverso ***mqtt.monitor()*** se vengono fatte delle subscribe, leggiamo la temperatura dal sensore e pubblichiamo continuamente sul topic stabilito la temperatura rilevata, formattata in formato json. L'operazione di formattazione viene effettuata attraverso la funzione ***senMlEncode***.