Laboratorio HW3 **Tecnologie per IoT** **Gruppo 9**

Eugenio Principi - s236654
Arya Houshmand - s247275
Leonardo Perugini - s249807

Esercizio 3.1

In questo vengono resi disponibili i dati rilevati dal sensore di temperatura e led tramite richiesta HTTP. Il server HTTP è stato installato su Arduino Yun seguendo quanto scritto nelle slide e utilizzando la libreria *BridgeServer.*
L'utente tramite una richiesta GET al server Arduino può richiedere i dati relativi al led oppure alla temperatura: in particolare per quanto riguarda il led, l'utente decide se spegnere o accendere il led a seconda dell'URL inserito; invece nel caso della temperatura viene semplicemente fornito l'ultimo valore misurato dal sensore. In ogni caso le risposte fornite dalla scheda Yun seguiranno il formato SenML come indicato.

Nel *setup* e nel *loop* viene prima avviato il server su Arduino, e dopodiché nel loop è chiamata la funzione incaricata di gestire le richieste GET dell'utente.

```
void setup() {
  // put your setup code here, to run once:
  pinMode(LED_PIN, OUTPUT);
  pinMode(TEMP_PIN, INPUT);
  digitalWrite(LED_PIN, LOW);
  Bridge.begin();
  digitalWrite(LED_PIN, HIGH);
  server.listenOnLocalhost();
  server.begin();
}

void loop() {
  // put your main code here, to run repeatedly:
  BridgeClient client = server.accept();
  if(client) {
    process(client);
    client.stop();
  }
  delay(1000);
}
```

Nella funzione *process* vengono parsificate le URL e a seconda del comando inviato dall'utente viene elaborata la risposta, controllando prima che la richiesta sia stata scritta correttamente.

```
void process(BridgeClient client) {
  String command = client.readStringUntil('/');
  command.trim();

  if(command == "led") {
    int val = client.parseInt();
    if(val == 0 || val == 1) {
      digitalWrite(LED_PIN, val);
      printResponse(client, 200, senMlEncode(F("led"), val, F("")));
    }
    else{
      printResponse(client, 400, "");
    }

  }
  else if (command == "temp"){
      volatile float tmp = returnTemp();
      printResponse(client, 200, senMlEncode(F("temp"), tmp, F("Cel")));

    }
    else{
      printResponse(client, 404, "");
    }
}
```

La funzione *senMlEncode* costruisce semplicemente l'output in formato SenML.

Esercizio 3.2

In questo esercizio la scheda Yun invia delle richieste HTTP POST verso un server locale, in cui vengono fornite informazioni riguardo a misurazioni effettuate dal sensore di temperatura.
Attraverso la URI localhost:8080/log si può accedere a quanto esposto dal sensore; il server in locale è realizzato tramite il framework CherryPy su Python (estendendo gli esercizi del laboratorio SW1).
Infine tramite la richiesta di tipo GET viene restituita l'intera lista di misurazioni formattata secondo il codice SenML in un unico JSON.

```
int curl_function(String text){
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(text);
 p.addParameter("http://192.168.43.184:8080/log");
 p.run();
 return(p.exitValue());
}
```

La funzione *curl_function* si occupa di inviare le richieste POST contenenti le rilevazioni di temperatura verso il server locale. Qui di seguito è riportato il codice attraverso il quale vengono gestite le richieste HTTP.

```
class ArduinoTemperature(object):
    exposed = True

    def __init__(self):
        self.list = list()
        self.obj = {"all":[]}


    def POST(self, *uri, **params):
        data = cherrypy.request.body.read()
        json_data = json.loads(data.decode('utf-8'))
        self.list.append(json_data)

        self.obj['all'].append(json_data)

    def GET(self, *uri, **params):
        res = json.dumps(self.obj)
        return res

if __name__=="__main__":
    #Standard configuration to serve the url "localhost:8080"
        conf={
            '/':{
                    'request.dispatch':cherrypy.dispatch.MethodDispatcher(),
                    'tool.session.on':True
                }
        }

        cherrypy.tree.mount(ArduinoTemperature(), '/log', conf)
        cherrypy.config.update({'server.socket_host': '0.0.0.0'})
        cherrypy.config.update({'server.socket_port': 8080})
        cherrypy.engine.start()
        cherrypy.engine.block()
```

Esercizio 3.3

Nell'ultimo esercizio viene utilizzato il paradigma di comunicazione Publish/ Subscribe sulla scheda Arduino Yun, attraverso il broker *test.mosquitto.org*. In particolare quando la Yun funziona da publisher invia periodicamente le rilevazioni di temperatura formattate con SenML.

```
void setup() {
  // put your setup code here, to run once:

  pinMode(LED_PIN, OUTPUT);
  pinMode(TEMP_PIN, INPUT);
  digitalWrite(LED_PIN, HIGH);
  Bridge.begin();
  mqtt.begin("test.mosquitto.org", 1883);
  mqtt.subscribe(my_base_topic + String("/led"), setLedValue);
}

void loop() {
  volatile float temp = returnTemp();
  Serial.println(temp);

  mqtt.monitor();

  String message = senMlEncode("temperature", temp, "Cel");
  mqtt.publish(my_base_topic + String("/temperature"), message);

  delay(4000);
}
```

Nella fase di setup la board effettua l'iscrizione al topic …/led, mentre nel loop pubblica periodicamente (in questo caso 4 secondi) i valori di temperatura.

L'utente tramite linea di comando può iscriversi al topic temperature e ricevere le misurazioni desiderate.

```
MacBook-Pro-di-Eugenio:~ eugenioprincipi$ mosquitto_sub -h test.mosquitto.org -t
 '/tiot/9/temperature'
{"bn":"Yun","e":[{"n":"temperature","t":70066,"v":26.67703,"u":"Cel"}]}
{"bn":"Yun","e":[{"n":"temperature","t":74507,"v":26.67703,"u":"Cel"}]}
{"bn":"Yun","e":[{"n":"temperature","t":78994,"v":26.67703,"u":"Cel"}]}
{"bn":"Yun","e":[{"n":"temperature","t":83423,"v":26.67703,"u":"Cel"}]}
```

Nel caso in cui si voglia invece cambiare lo stato del led, l'utente può effettuare una publish tramite linea di comando al topic tiot/9/led e inserire un messaggio in formato SenML dove specifica nel campo val il comportamento desiderato.
Qui la scheda Arduino nel setup ha effettuato l'iscrizione al topic led ed è in grado di gestire le richieste corrispondenti.

```
void setLedValue(const String& topic, const String& subtopic, const String& message){
  DeserializationError err = deserializeJson(doc_rec, message);
  volatile int val;

  if(err){
    Serial.print(F("deserializeJson() failed with code "));
    Serial.println(err.c_str());
  }
  if(doc_rec["e"][0]["n"] == "led"){
    val = doc_rec["e"][0]["v"];

    if(val == 1 || val == 0){
      digitalWrite(LED_PIN, val);
    }
    else{
      Serial.println("Errore di comando, valore non accettato");
    }
  }
}
```