Report Laboratorio HW parte 3 del corso Tecnologie per IoT

La parte 3 del laboratorio Hardware del corso è basato sulla comunicazione della scheda Arduino Yun con un network tramite interfacce REST e MQTT.

Esercizio 1

Nel primo esercizio è richiesta l’implementazione di uno sketch per la Yùn che esegua un server HTTP in grado di rispondere a richieste GET provenienti dalla rete locale. Sulla scheda sono connessi un led ed un sensore di temperatura, che verranno utilizzati alla ricezione di un’opportuna richiesta GET dal server, che dovrà restituire in risposta nel body in formato JSON compatibile con la convenzione senML utilizzata per i sensori, più precisamente dovrà essere possibile settare lo status del led attraverso la medesima richiesta.

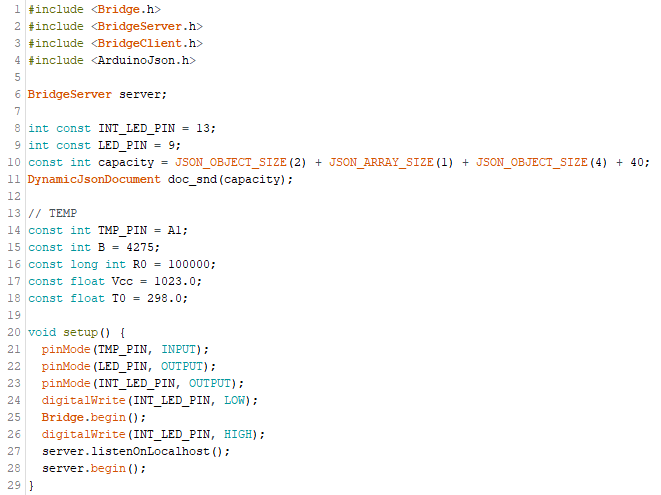
*Esempio di richiesta per il led:*

http://<hostname>:<porta>/arduino/led/1

*Esempio di richiesta per il sensore di temperatura:*

http://<hostname>:<porta>/arduino/temperature

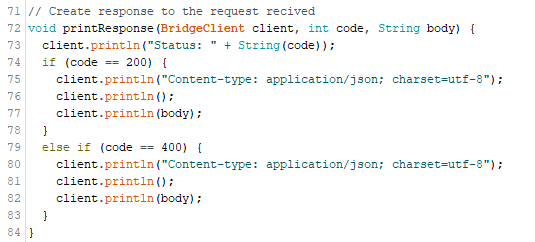
Per interfacciare la scheda abbiamo fatto uso delle librerie *Bridge.h*, *BridgeServer.h* e *BridgeClient.h* per poter esporre le risorse e poter gestire le richieste GET ricevute. Inoltre come suggerito abbiamo fatto uso della libreria *ArduinoJson.h* e dei documenti dinamici JSON predefiniti per lavorare agilmente con questo formato di risposta:



Nel *loop()* abbiamo predisposto il server alla ricezione di richieste HTTP GET, che mettono in esecuzione la funzione *process()* in cui avviene la gestione dei principali errori sul formato della richiesta con il lancio di errori con code: 400. Viene inoltre effettuato il riconoscimento della richiesta, led o temperatura e settato nel primo caso il valore ricevuto come nuovo stato, e fatta partire la creazione di una risposta attraverso la *printResponse()*:



Per completezza alleghiamo anche una immagine relativa alla creazione del messaggio di risposta:



Non ci soffermiamo sul funzionamento della funzione actualTemp() per il valore della temperatura, che è la stessa utilizzata già nei precedenti laboratori per effettuare il calcolo del valore corrente rispetto all’input ricevuto dal sensore, mentre la funzione senMlEncode() utilizzata per entrambe le risposte si occupa della costruzione del documento JSON secondo lo standard senML, inserendo negli opportuni campi il nome della misura “temperature” o la scritta “led”, il valore attuale, il timestamp e relativa unità di misura nel caso della temperatura. Infine convertiamo il contenuto del file JSON in stringa per poterlo mandare in risposta.

Esempio formato senML:

{

“bn”: “Yùn”

“e”: [

{

“n”: <”temperature”>/<”led”>,

“t”: <timestamp using millis()>,

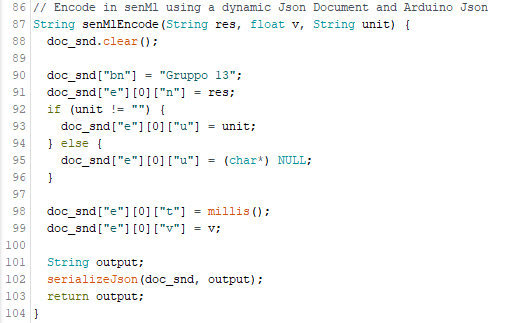
“v”: value,

“u”: “Cel”/null

}

]

}



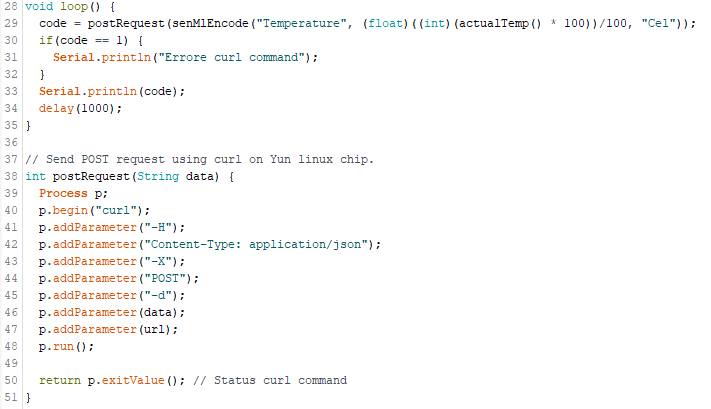
Esercizio 2

Il secondo esercizio consiste nella modifica del codice del server basato su *cherrypy* scritto in uno dei precedenti laboratori, che si occupava della conversione dei valori di temperatura ricevuti a seconda delle unità di misura. La modifica sarà utile a gestire delle richieste HTTP POST periodiche contenenti un log periodico della temperatura sempre secondo il formato JSON compatibile senML inviate dalla Yùn sullo stesso network.

La scheda invierà i dati facendo uso di una shell gestita dal secondo processore della Yùn con distribuzione linux, attraverso il comando *curl*. Inoltre il server avrà esposto anche un metodo http GET che restituirà una lista in formato JSON con tutti logs ricevuti fino a quel momento.

*Esempio richiesta POST:*

URI: http://<hostname>:<port>/log



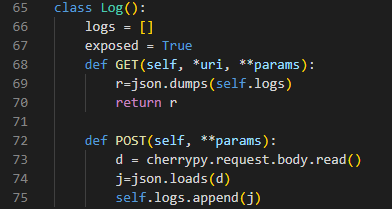
L’unica differenza sostanziale sullo sketch consiste nella gestione della richiesta tramite *curl* e la shell, che viene effettuata tramite la libreria *Process.h*, che permette di scrivere direttamente un comando tramite i metodi *begin()* e *addParameter()* per l’aggiunta dei parametri della richiesta.

La stringa data passata alla funzione postRequest() è output della stessa funzione senMlEncode utilizzata nell’esercizio precedente per la costruzione del JSON di risposta, mentre il valore è stato arrotondato con un artifizio algebrico a due cifre dopo la virgola.

E inoltre aggiunto un controllo sull’exit code di *curl* per poter inviare un errore sul seriale.

Lato server invece la differenza sta nell’esposizione nel *main()* di una nuova classe *Log()* sul percorso log/, con la gestione della POST e della GET e il setting di un IP che rendesse raggiungibile dal network locale:





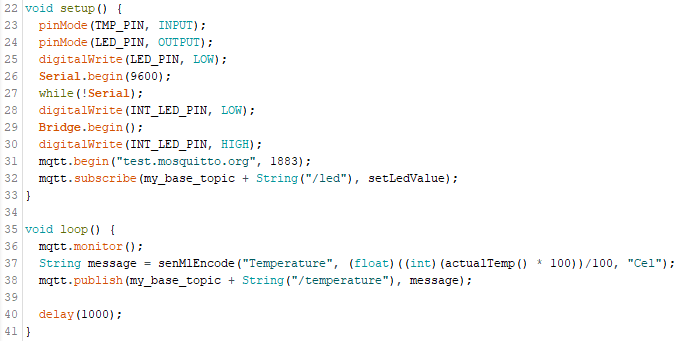
Al ricevimento del JSON a seguito della POST si fa una lettura di questo come dizionario, lo convertiamo in JSON e lo inseriamo in una lista *logs* in comune per tutte le classi *Log().*

La GET farà il processo inverso, convertendo in stringa la lista di logs e ritornandola.

Esercizio 3

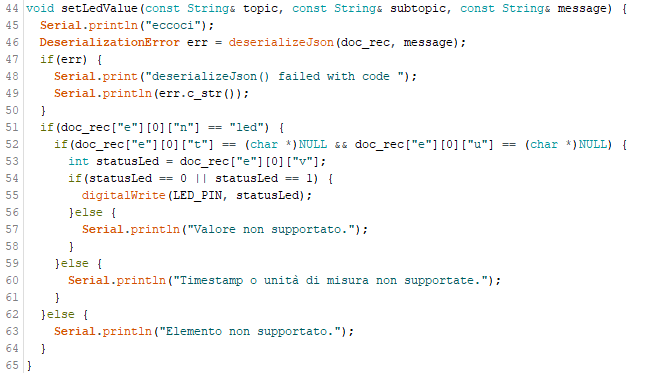
Il terzo esercizio similmente al primo richiede di poter ricevere periodicamente il log in formato JSON compatibile senML la temperatura rilevata dal sensore e di poter settare a distanza lo stato del led, tutto utilizzando un paradigma Publish/Subscribe con MQTT e il broker mosquitto.

Utilizzeremo la libreria MQTTclient.h fornitaci per porter implementare le funzioni di *publish* e di *subscribe* ai topic corretti e abbiamo inoltre installato le utility per poterli effettuare sempre tramite una shell linux.



Nel *setup()* abbiamo inizializzato la connessione col broker *test.mosquitto.org* tramite la libreria MQTT e abbiamo effettuato una *subscribe* al topic “*/tiot/13/led*” per poter ricevere input di cambi di stato al led, mentre nel loop oltre a monitorare periodicamente nuovi messaggi al topic a cui siamo sottoscritti, inviamo anche tramite una *publish* un messaggio in formato JSON compatibile senML con la funzione degli esercizi precedenti al topic *“/tiot/13/temperature*”, che agisce come richiesto.

Molto interessante è analizzare la *monitor()*, che richiama la funzione di callback che decodifica i dati nel solito formato utilizzato, per poi poter settare lo stato del led come richiesto:



Utilizziamo in primis la funzione *deserializeJson()* sul messaggio ricevuto inserendolo nel nostro documento JSON e gestendo eventuali errori segnalandoli sul seriale. Nel caso sia tutto corretto controllando i vari campi del documento, andiamo ad estrapolare lo status del led che ci viene inviato, per poi farne la scrittura digitale sul pin corretto relativo al led.

Molto interessante è stato inoltre verificarne il funzionamento durante lo svolgimento del laboratorio in gruppo a distanza, potendo ricevere i valori di temperatura su qualsiasi pc e potendo settare lo stato del led anche non in presenza.