

01UEIPL Tecnologie per IoT – a.a. 2020/21

Esercitazione di Laboratorio 1

PARTE 3

Comunicazione tramite interfacce REST e MQTT

3.1) Arduino Yùn come server HTTP

Si realizzi uno sketch che esegua un server HTTP sulla Yùn, in grado di rispondere a richieste GET provenienti dalla rete locale. Lo sketch deve rispettare le seguenti specifiche:

- Il server HTTP deve essere realizzato tramite la libreria `BridgeServer`, già disponibile nella IDE di Arduino.
- Il server deve esporre come “risorse” uno dei LED ed il sensore di temperatura. In particolare:
 - Le richieste GET alle risorse `http://<hostname>:<porta>/arduino/led/1` oppure `http://<hostname>:<porta>/arduino/led/0` devono venire gestite accendendo o spegnendo il LED rispettivamente. Nella risposta deve venire fornito lo stato attuale della risorsa dopo la modifica.
 - Le richieste GET alla risorsa `http://<hostname>:<porta>/arduino/temperature` devono venire gestite fornendo nella risposta il valore di temperatura letto dal sensore.
- Tutte le altre richieste devono essere gestite fornendo in risposta un appropriato codice di errore HTTP.
- Il body di tutte le risposte prodotte dalla Yùn deve essere in formato JSON e seguire la convenzione senML come nel seguente esempio:

```
{
  "bn": "Yùn"
  "e": [
    {
      "n": <"temperature"/><"led">,
      "t": <timestamp using millis(>,
      "v": value,
      "u": "Cel"/null
    }
  ]
}
```

La serializzazione dei dati in JSON può essere effettuata “manualmente” concatenando opportunamente delle stringhe, oppure tramite la libreria [ArduinoJson](#) disponibile nel Library Manager della IDE di Arduino.

Si verifichi il funzionamento dello sketch utilizzando un browser o una utility da linea di comando per effettuare richieste HTTP come ad esempio `curl`.

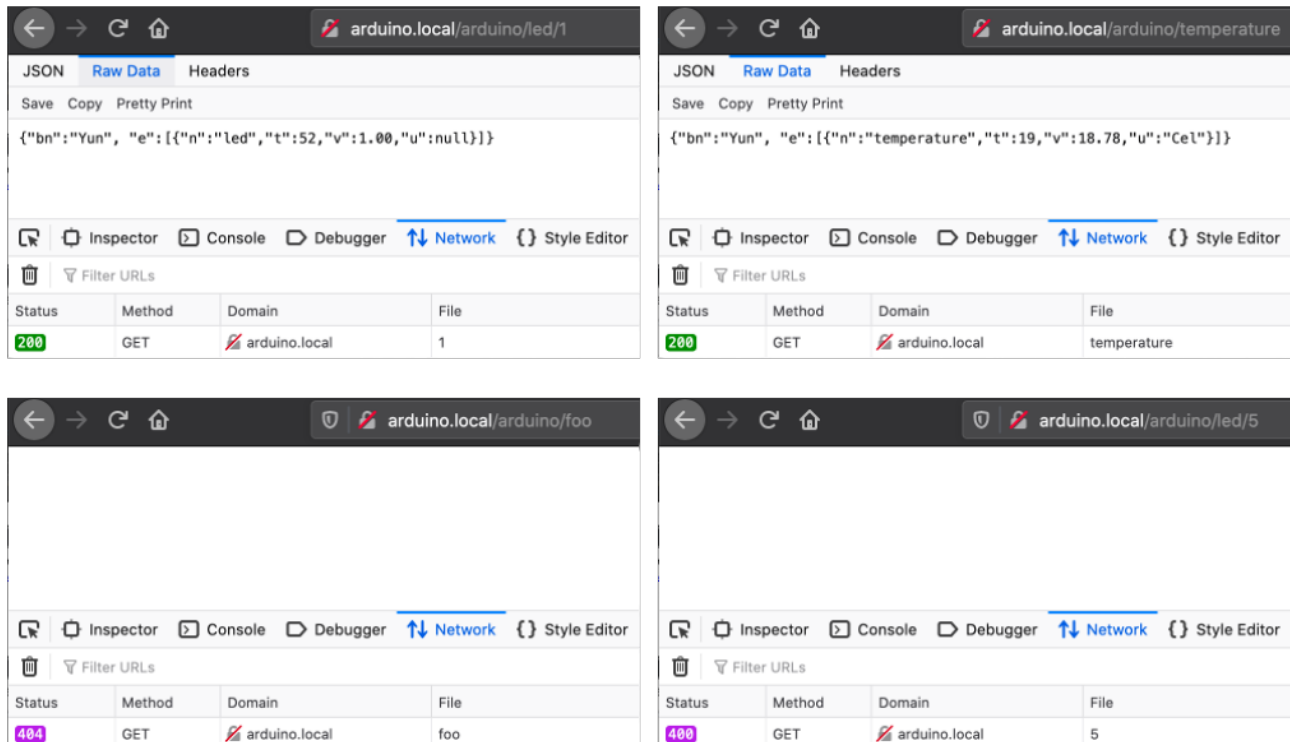


Figura 1: esempi di output prodotti dalla Yùn in risposta a diverse richieste GET per l'Esercizio 3.1.

3.2) Arduino Yùn come client HTTP

Si realizzi uno sketch che effettui richieste HTTP POST dalla Yùn verso un server, per realizzare un logging periodico dei dati del sensore di temperatura. Lo sketch deve rispettare le seguenti specifiche:

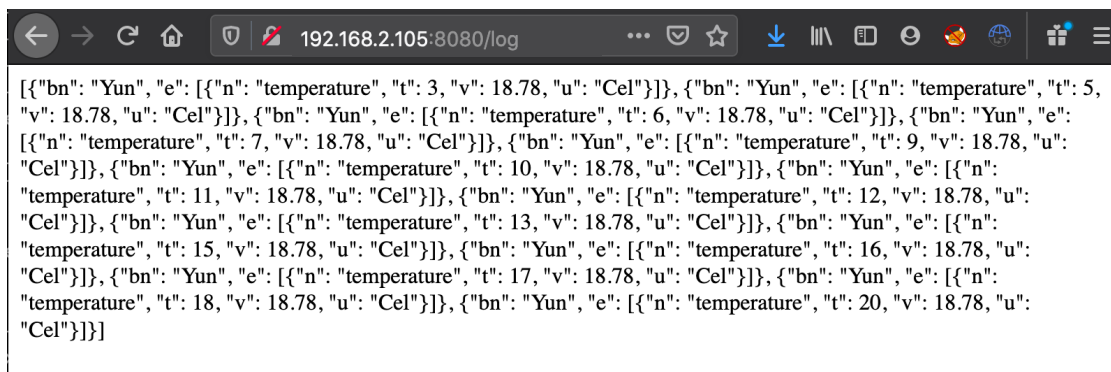
- Il client HTTP deve essere realizzato tramite il programma `curl`, installato sul processore Atheros della Yùn, invocando quest'ultimo dallo sketch tramite la libreria `Process`, disponibile nella IDE di Arduino.
- Lo sketch deve leggere periodicamente la temperatura dal sensore e inviare una richiesta POST ad un server, utilizzando il seguente URI: `http://<hostname>:<port>/log`

- Il body della POST deve contenere il dato di temperatura, formattato in SenML in modo analogo all'Esercizio 3.1
- Lo sketch deve stampare eventuali valori di ritorno corrispondenti ad errori prodotti da `curl` sulla porta seriale.

Si estendano quindi gli Esercizi 1 e 2 del laboratorio Software (Parte 1), in modo da gestire le richieste provenienti dall'Arduino. Il server deve rispettare le seguenti specifiche:

- Le richieste POST provenienti dalla Yùn verso la risorsa `"/log"` devono essere gestite memorizzando il JSON contenuto nel body in una lista
- Le richieste GET verso la stessa risorsa devono essere gestite restituendo l'intera lista formattata come un unico JSON.

Si verifichi il funzionamento dello sketch utilizzando un browser.



```
[{"bn": "Yun", "e": [{"n": "temperature", "t": 3, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 5, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 6, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 7, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 9, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 10, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 11, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 12, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 13, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 15, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 16, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 17, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 18, "v": 18.78, "u": "Cel"}]}, {"bn": "Yun", "e": [{"n": "temperature", "t": 20, "v": 18.78, "u": "Cel"}]}]
```

Figura 1: Esempio di output prodotto dal server cherrypy dell'Esercizio 3.2

3.3) Arduino Yùn come publisher e subscriber MQTT

Si realizzi uno sketch che consenta alla Yùn di comunicare via MQTT, agendo sia come publisher che come subscriber. Lo sketch deve rispettare le seguenti specifiche:

- Le funzioni di `publish` e `subscribe` devono essere implementate con la libreria `MQTTclient` resa disponibile sul portale e basata sulla suite `mosquitto`, che andrà installata sul processore Atheros della Yùn.
- L'hostname del broker MQTT da utilizzare per questo esercizio è `test.mosquitto.org`, ovvero un broker pubblico e liberamente utilizzabile per test che coinvolgono piccole quantità di dati.

- Lo sketch dovrà pubblicare periodicamente (ad es. ogni 10s) un valore di temperatura sul topic `/tiot/<group_number>/temperature` formattato in SenML in modo analogo all'Esercizio 3.1
- Lo sketch dovrà inoltre "iscriversi" al topic `/tiot/<group_number>/led` sul quale, tramite PC, verranno inviati dati in formato SenML per controllare uno dei LED della board. Il codice dovrà reagire a tali messaggi controllandone il formato e, nel caso in cui esso sia corretto, accendendo o spegnendo il LED.

Si consiglia di utilizzare la libreria `ArduinoJson` per la deserializzazione dei messaggi JSON di controllo ricevuti, dal momento che questa funzionalità è decisamente più complessa della serializzazione necessaria per la trasmissione.

Si verifichi il funzionamento dello sketch utilizzando (da PC) le utility da linea di comando della suite `mosquitto` (`mosquitto_sub` e `mosquitto_pub`).

```
→ ~ mosquitto_sub -h test.mosquitto.org -t '/tiot/0/temperature'
{"bn":"Yun","e":[{"t":45,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":46,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":47,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":49,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":50,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":52,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":53,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":54,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":55,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":57,"n":"temperature","v":18.62283,"u":"Cel"}]}
{"bn":"Yun","e":[{"t":58,"n":"temperature","v":18.62283,"u":"Cel"}]}
```

Figura 2: esempio di dati ricevuti da un PC che faccia "subscribe" sul topic relativo alle misurazioni di temperatura.

```
→ ~ mosquitto_pub -h test.mosquitto.org -t '/tiot/0/led' -m '{"bn": "Yun", "e": [{"n": "led", "t": null, "v": 1, "u": null}]}'
→ ~ mosquitto_pub -h test.mosquitto.org -t '/tiot/0/led' -m '{"bn": "Yun", "e": [{"n": "led", "t": null, "v": 0, "u": null}]}'
```

Figura 3: esempi di comandi inviabili da un PC alla Yun per accendere e spegnere il LED