Relazione finale Tecnologie per IoT

Gruppo 23
Alunno: Poma Giorgio Gaspare
Numero di Matricola: S247528

Alunno: Rinaldi Pasquale
Numero di Matricola: S248291

Alunno: Turriziani Damiano
Numero di Matricola: S248121

Abstract: *L'evoluzione delle tecnologie è ormai un elemento cruciale nello sviluppo della società umana cosi come la conosciamo. L'interconnessione dei dispositivi che comunicano da ogni parte del mondo è alla base della nostra vita, per cui capire ed ottimizzare i processi, lo scambio di dati e l'utilizzo di componenti hardware deve essere l'obbiettivo primario di un futuro ingegnere. Nelle pagine successive verrà analizzato lo sviluppo dei vari laboratori che hanno caratterizzato lo svolgimento del corso. Ci soffermeremo soprattutto sulle possibili ottimizzazioni implementabili e su quelle implementate. Il codice utilizzato verrà fornito mediante l'utilizzo di link ipertestuali, collegati direttamente al repository generale presente su Github. (* https://github.com/pippo314/IoT_LABS *)*

Software: Laboratorio 1

ESERCIZIO_1.1

In questo primo esercizio per ricevere il valore convertito della temperatura utilizziamo la funzione GET esposta da un webserver contenuto nel file sw1_1.py. Dal lato cherrypy, avviato dal file sw1_1_main.py, la richiesta viene gestita attraverso:

- uri: contenente il path dell'URL, in questo caso la stringa "converter" per accedere al servizio adeguato.
- params: contenente la query-string dell'URL sottoforma di coppie chiave-valore, quindi le unita' di misura ed il valore della temperatura da convertire.

Dopo una serie di controlli sulle chiavi e sui valori, il valore della temperatura viene convertito e viene ritornato il JSON finale, contenente il risultato.

ESERCIZIO_1.2

Il programma, simile all'esercizio precedente, converte un dato valore di temperatura ricevendo sia l'unità di misura originaria, sia la nuova unita' di misura mediante il path dell'URL. La risposta della funzione GET viene ritornata mediante un file JSON.

ESERCIZIO_1.3

A differenza dei primi due esercizi abbiamo implementato una funzione PUT, la quale riceve un file JSON contenente i valori da convertire, l'unità originaria e l'unità di conversione. La funzione, ricevuta la richiesta, legge i dati contenuti nel body, li decodifica e li salva in variabili separate. Successivamente viene effettuato un controllo per verificare che le variabili non siano vuote e infine il programma procede alla conversione dei valori, ritornando i dati come JSON.

ESERCIZIO_1.4

In quest'ultimo esercizio del primo laboratorio utilizziamo i metodi GET e POST il primo per ritornare un file HTML, il secondo per salvare una nuova configurazione della dashboard.

Software: Laboratorio 2

ESERCIZIO_2.1

Ci siamo approcciati a questo esercizio dividendo il codice in 4 file diversi, contenuti nella cartella catalog, per renderlo più comprensibile e di semplice interpretazione. I file utilizzati hanno il determinato scopo di gestire al meglio una o più funzionalità e sono:

1. sw2_1_Classi.py: contiene le classi python che rappresentano gli oggetti contenuti nel catalog ovvero i dispositivi, gli utenti e i servizi. Ogni oggetto, rispettando il paradigma della programmazione a oggetti, contiene i metodi e gli attributi che soddisfano le specifiche del problema.
2. sw2_1_Collector.py: questo file contiene la dichiarazione della classe `Collector`, una collezione generica di oggetti nella forma di dizionario dove la chiave è l'ID univoco dell'oggetto e il valore è l'oggetto stesso. Questa collezione offre delle funzioni per l'inserzione, la ricerca, l'aggiornamento, la verifica di presenza e l'elencazione degli elementi contenuti in essa.
3. sw2_1_Catalog.py: contiene le classi web che espongono i servizi richiesti dal problema. Ad esempio, la classe `Devices` gestirà le richieste riguardanti i dispositivi da inserire nel catalog e così via. La classe Broker inoltre fornisce le informazioni relative al message broker e la porta utilizzati nel protocollo MQTT. Ogni webclass contiene un attributo di tipo `Collector` per memorizzare i dati contenuti nelle richieste inviate al webserver.
4. sw2_1_main.py: contiene il main che avvia il webserver tramite cherrypy, montando ogni classe web sul path corrispondente per gestire più chiaramente le richieste.

ESERCIZIO_2.2

L'implementazione di questo client è stata di rapida concezione e sviluppo. Come da richiesta, il client si occupa di interagire con l'utilizzatore per ottenere i dati contenuti nel Catalog, effettuando le varie richieste di GET.

ESERCIZIO_2.3

Esattamente come da testo fornitoci, il client crea un nuovo oggetto device all'interno del catalog inserendo il nuovo elemento tramite una funzione PUT con parametri l'indirizzo a cui effettuare la richiesta ed un JSON contenente i dati del dispositivo. La PUT viene effettuata ogni 60 secondi con lo stesso elemento così da permettere l'aggiornamento del timestamp dell'elemento.

ESERCIZIO_2.4

Questo esercizio e' stato completato tramite l'implementazione di un thread per effettuare le richieste ogni 60 secondi. In particolare, e' stata importata la libreria `thread`, creata la classe web `WebServer` che si occupa delle richieste effettuate al server, e la

classe `IlMioThread` che effettua una PUT sulla classe web. Nel main del programma vengono inizializzate ed avviate le due classi.

ESERCIZIO_2.5.

Qui il Catalog viene modificato così da poter accettare anche le richieste MQTT in arrivo per ampliare il numero dei dispositivi. Cio' viene eseguito mediante un subscriber MQTT inserito nella classe `Devices` che gestisce le richieste inoltrate tramite MQTT e richiama le funzioni base per l'aggiunta di un dispositivo o, nel caso in cui sia gia' presente un dispositivo con stesso ID, per l'aggiornamento del timestamp.

ESERCIZIO_2.6

Prendendo spunto dall'esercizio di riferimento, abbiamo sviluppato un publisher MQTT per la comunicazione diretta con il subscriber implementato nell'esercizio 2.5. Dopo aver creato la nuova istanza del publisher che si collega al broker, alla porta e si sottoscrive al topic, vengono effettuate delle richieste al catalog tramite l'utilizzo della publish. Se il dispositivo è già presente nella lista di dispositivi, viene semplicemente aggiornato il timestamp tramite l'opportuna funzione nel catalog.

Software: Laboratorio 3

ESERCIZIO_3.2

L'esercizio e' stato svolto mediante la creazione di un subscriber MQTT il quale, dopo l'inizializzazione con i dati necessari, richiede al catalog tramite la funzione `getDeviceTopic()`, i topic della scheda Arduino e li salva. Successivamente tramite la funzione `registerOnCatalog()`, inoltra una richiesta di PUT al catalog inviando un JSON con all'interno i propri dati. Dopo la registrazione vi è l'acquisizione dei dati riguardanti il broker e la porta, utilizzati dal dispositivo. Infine, inizia l'ascolto sottoscrivendosi al topic dato.

ESERCIZIO_3.3

Esattamente come per l'esercizio precedente, il publisher viene inizializzato e viene effettuata la registrazione al Catalog tramite richiesta PUT nel cui body è presente un file JSON contenente le varie informazioni del servizio. Dopo la registrazione si ottengono le informazioni necessarie a stabilire la connessione MQTT con il message broker ed in seguito viene inizializzato il file JSON da inoltrare al dispositivo. Tramite un ciclo while viene acquisito un valore da tastiera, nel caso di valore pari ad 1 viene modificato il file JSON da inoltrare e inviata una richiesta di accensione del led, al contrario in caso di 0 una richiesta di spegnimento.

ESERCIZIO_3.4

Essendo questo un esercizio piu' complesso da implementare rispetto i precedenti e visto la quantita' di strumenti da utilizzare, quest'ultimo progetto e' stato diviso in quattro file così come segue:

1. **sw3_4_ServerMain.py:** file che si occupa della configurazione iniziale e dell'avvio del webserver tramite cherrypy. Il webserver espone la RESTful API RemoteController.
2. **sw3_4_MyMQTT.py:** definizione publisher e subscriber per le comunicazioni con il dispositivo Arduino.
3. **sw3_4_RemoteController.py:** file che contiene la definizione della web-class `RemoteController`, nucleo della comunicazione tra l'utente e la board.
4. **sw3_4_menu.py:** menù a riga di comando che permette all'utente di inviare i comandi di attuazione e ricevere le informazioni da Arduino.

Spiegazione Generale

Per chiarire la struttura del programma potrebbe essere utile illustrare il flusso di comunicazione tra gli elementi del sistema nel caso di invio di comandi di attuazione e richiesta di informazioni dei sensori. Prima che abbia inizio la comunicazione sarà necessario che il dispositivo si registri al catalog (tramite mqtt o no) e che il client mqtt,

definito come attributo nella webclass RemoteController, ottenga, tramite una richiesta al catalog, il message broker e la porta che il dispositivo intende utilizzare.

Quest'ultimo passaggio viene effettuato dal client MQTT del RemoteController quando viene inizializzato.

Dopo aver avviato il menu il sistema è pronto affinché avvenga la comunicazione:

- comandi di attuazione: con un apposito comando dal menù si possono scegliere una serie di azioni che la board in questione può eseguire, come lo spegnimento e l'accensione di led e motore DC o la stampa sullo schermo LCD di messaggi personalizzati. Tramite una richiesta PUT viene inviato al RemoteController un JSON nel seguente formato: { "Command": "code" } dove "code" è il codice in stringa corrispondente al comando da eseguire. Il server quindi filtra le richieste valide e invia tramite MQTT un JSON nel formato SenMl che, ricevuto dalla scheda, permetterà l'attuazione del comando digitato nel menu.
- richiesta informazioni sensori: sempre attraverso un comando opportuno è possibile ricevere dalla scheda Arduino le informazioni dei sensori, ad esempio la temperatura, la presenza o meno di un individuo o il rumore rilevato. Questa volta la richiesta al server remoto viene fatta tramite una GET che conterrà il codice del comando nella query-string dell'URL (http://serverIp:serverPort/?Command=code). Il server, in particolare l'attributo della classe che fa da publisher e subscriber, contiene una struttura dati per salvare i messaggi MQTT che Arduino invia periodicamente con le informazioni sui valori letti dai sensori. Alla ricezione della richiesta il server quindi risponderà inoltrando semplicemente il messaggio ricevuto dalla board riguardo il sensore o la grandezza richiesti.

Annotazioni

Nella nostra implementazione del controllore remoto le le funzionalità aggiuntive rispetto al laboratorio 2 di Hardware sono molteplici. Un lato positivo è la possibilità di controllare la board da remoto, inviando comandi di accensione e spegnimento degli impianti o richiedendo informazioni sui sensori disponibili non dovendo essere per forza nella stessa stanza del controllore. Anche la presenza del Catalog che registra dispositivi, servizi e utenti può essere un grande pro offrendo la possibilità di monitorare i dispositivi connessi o con qualche aggiunta gli utenti che hanno eseguito accesso al sistema. Il controller remoto offre inoltre delle potenzialità interessanti, dall'interfaccia utente tramite sito web, app o bot telegram al salvataggio dei dati di sensori su un database. La difficoltà maggiore però si è presentata cercando far comunicare correttamente tutti gli elementi del sistema garantendo prestazioni decenti e carichi di lavoro accettabili.

In conclusione, le enormi possibilità offerte dal controllore remoto sono bilanciate o meglio limitate dalla capacità ristretta della scheda ArduinoYùn in termini di memoria che richiede attenzioni e ottimizzazioni aumentando la complessità del progetto.

Software: Laboratorio 4

Premessa: In quest'ultimo laboratorio abbiamo utilizzato dei moduli aggiuntivi rispetto a quelli forniti dai docenti, eccone l'elenco:

- Interruttori MQTT per boiler e serrande.
- Un buzzer passivo modello CE482067.
- Un modulo RTC modello DS1302.
- Un tastierino numerico modello HMTTAS16

Introduzione al progetto

Rispetto ai laboratori precedenti sono state applicate varie migliorie ed aggiunte nuove funzioni tramite l'utilizzo di interruttori compatibili MQTT, un ulteriore sensore di movimento, un buzzer per il suono ed un tastierino numerico. Gli interruttori MQTT sono considerati già connessi al catalog. Il valore dei messaggi inviati verso questi ultimi tramite JSON sarà 1 per l'accensione e 0 per lo spegnimento L'implementazione avviene mediante l'introduzione di un secondo dispositivo Arduino Yùn che verrà inserito all'interno del catalog esattamente come il primo. I due dispositivi saranno in grado di gestire una smart-home mediante una comunicazione MQTT costante con il server. Il primo dispositivo, chiamato d'ora in avanti Yun_camera avrà il compito di gestire il sensore di temperatura, il modulo RTC per essere sempre informato in tempo reale della data e dell'ora, un buzzer passivo che fungerà da sveglia, lo schermo LCD, un led che si ipotizza essere il riscaldatore resistivo ed infine la ventola. Il secondo dispositivo, chiamato Yun_esterno sarà dotato di due sensori di movimento, uno per l'esterno ed uno per l'interno della casa , un sensore di rumore interno alla casa, un tastierino numerico e dei led.

Yùn Camera

Questa componente del progetto prende spunto dal laboratorio precedente e lo migliora inserendo nuove features grazie alla suddivisione in più dispositivi e all'utilizzo dei nuovi moduli, tra cui il buzzer e l'RTC. Quest'ultimo assume grande rilievo nella gestione della tempistica delle varie funzioni ed in particolare è responsabile diretto o indiretto delle seguenti implementazioni:

- Riscaldamento nei mesi invernali e raffreddamento nei mesi estivi.
- Attivazione/disattivazione degli interruttori MQTT con cadenza giornaliera.
- Accensione/spegnimento del sensore di movimento esterno per la gestione della lampada.
- Attivazione del Buzzer passivo che funge da sveglia.
- Gestione dell'orologio digitale riprodotto nello schermo LCD.

Questo nuovo modulo e' stato implementato mediante l'utilizzo della libreria `<swRTC.h>`, è stata istanziata una variabile globale e inizializzata nel `setup`. La gestione dei dispositivi non collegati direttamente alla board avviene mediante l'utilizzo del protocollo MQTT e di file con formato JSON per lo scambio di informazioni. Tutte le richieste ricevute ed inviate dall'ecosistema creato passano per un server, il quale, dopo

aver processato le informazioni ricevute, svolge le operazioni adeguate. Questo device si occupa di inviare al server le informazioni riguardanti la temperatura rilevata, le richieste di attivazione e disattivazione degli interruttori MQTT e il segnale di accensione per il sensore di movimento esterno passata una certa ora, impostata di default alle 22:00. Dovrà invece ricevere dal server e gestire le richieste relative alla presenza di persone nell'alloggio, al cambiamento dei set-points ed infine all'utilizzo dello schermo LCD. I messaggi saranno inoltrati mediante la funzione `encodeAndSend` e ricevuti mediante la `decodeCommand`. Come in tutti gli altri laboratori i vari devices si registreranno sul catalog rendendo disponibili le informazioni relative all'ID, alle risorse e ai topic.

Yùn Esterno

Lo Yun_esterno si occupera' invece della sicurezza e del rilevamento delle persone all'interno e all'esterno della casa. I componenti utilizzati dalla scheda Arduino si suddividono anch'essi in interni o esterni. I componenti esterni saranno un sensore di movimento ed un led. Il sensore di movimento sarà attivo solamente la notte e accenderà la lampada nel caso in cui venga rilevato qualcuno. Tutti gli altri moduli sono contenuti all'interno dell'abitazione. In particolare, il sensore di movimento unito al sensore di rumore ed al tastierino numerico, saranno utilizzati per implementare un sistema di sicurezza. L'ultimo componente è il led interno, che sarà possibile accendere con il battito delle mani.

I devices scambiano i dati con un server principale sempre attivo a cui sono relegate le funzioni di comunicazione. Questo programma si occupera' di contattare i dispositivi MQTT quali gli interruttori e permettera' lo scambio di dati tra le due Yun.

Server

La funzione del server cherrypy in questo esercizio, similmente all'esercizio precedente, è quella di ospitare il servizio web del controller remoto e gestire i dati dai sensori. Inoltre il client MQTT presente sul server è diventato il nodo centrale per la comunicazione tra le board, ovvero ogni volta che due schede devono inviare e ricevere messaggi tra loro lo fanno tramite il sever.

Scegliendo questa modalità di comunicazione centralizzata si è aumentato il carico di lavoro e la complessità del server diminuendone la scalabilità su grandi quantità di dispositivi e sensori lasciando però meno operazioni da compiere alle singole schede (deserializzazione meno complessa) che risultavano molto più suscettibili a workloads elevati. Nel nostro contesto questa scelta ci ha permesso quindi di assicurare un consumo di memoria minimo per le schede di Arduino.

Hardware: Laboratorio 1

ESERCIZIO_1.1

Come da richieste, abbiamo definito i pin dei due led ed i semi periodi come costanti e li abbiamo pilotati mediante interrupt e funzione `delay`.

ESERCIZIO_1.2

Lo sketch è sviluppato modificando quello precedente, inserendo in aggiunta il messaggio di benvenuto dopo la connessione, controllando la ricezione di un carattere da seriale e agendo di conseguenza.

ESERCIZIO_1.3

Il sensore di movimento collegato al pin 7 controlla i movimenti. Ogni volta che viene percepito un movimento dal sensore, la variabile `tot_count` viene incrementata. Il valore della variabile viene stampato ogni 30 secondi.

ESERCIZIO 1.4

La ventola, collegata al pin 10, ha velocità di rotazione inizializzata a 0 nel `setup`. Successivamente, quando viene letto da seriale un carattere '+' oppure '-', la variabile che gestisce la velocità della ventola viene aumentata di 25 o diminuita dello stesso valore fino ad un massimo di 255 ed un minimo di 0.

ESERCIZIO_1.5

Il sensore di temperatura è collegato al pin analogico 5. Il valore viene letto ogni 10 secondi e mostrato tramite seriale dopo aver effettuato le opportune operazioni dettate dal datasheet del sensore.

ESERCIZIO_1.6

Per implementare questo sketch abbiamo importato la libreria `LiquidCrystal_PCF8574.h`. Il valore letto dal sensore di temperatura viene mostrato sullo schermo LCD tramite la funzione `print` della libreria sopracitata.

Hardware: Laboratorio 2

ESERCIZIO_2.1

Condizionatore e Riscaldatore

La ventola è collegata al pin 5 che la controlla in base ai valori ricevuti dal sensore di temperatura e dai set-points presenti nel sistema. Successivamente si utilizza un LED collegato al pin 9 per simulare il comportamento di un riscaldatore, anche questo viene controllato in base a valori ricevuti dal sensore di temperatura e ai set-points. I set-points all'avvio dell'Arduino vengono inizializzati a valori costanti: 20/30 per la ventola e 10/20 per il led.

Sensore di movimento

Per monitorare la presenza di persone si utilizzano in modo complementare un sensore di movimento ed uno di rumore. Per quanto riguarda il sensore di movimento utilizzato questo è un PIR collegato al pin 7. Si è scelto questo pin poiché può essere utilizzato, come visto nelle specifiche dell'Arduino Yùn, per gestire segnali di Interrupt. Il sensore PIR opportunamente "collegato" alla funzione di ISR ogni volta che il suo stato commuta da HIGH a LOW (metodo FALLING), richiama la funzione `PIRisr()`, quest'ultima modifica il valore di `pir_is_present` (flag utilizzato per verificare la presenza di persone) e aggiorna la variabile `last` con il valore di `mills()` corrente. All'interno della funzione `loop` si controlla richiamando la funzione `PIRtimeout` se è trascorso un tempo pari a 30 minuti.

Sensore di rumore

Il sensore di rumore è utilizzato in modo complementare a quello di movimento per rilevare la presenza di individui che non si trovano a portata del sensore PIR. Il controllo sul sensore di rumore avviene tramite la funzione `readSound()` che si trova all'interno della funzione `loop()`. Si è scelto di andare a controllare lo stato del sensore ogni 500ms, ciò permette di avere un intervallo abbastanza piccolo da poter rilevare rumori "vicini" ma non troppo da considerare rumori dovuti all'eco di rumori già considerati, inoltre si è scelto questo valore dopo aver opportunamente regolato la sensibilità del sensore di rumore attraverso il potenziometro. La funzione *`readSound()`* implementa 3 funzionalità:

1. controllo dopo un tempo di 10 minuti se `n_sound_events` (variabile che mantiene il conto del numero di eventi avvenuti) ha assunto un valore maggiore a 50, in questo caso si modifica la variabile `sound_is_present` in modo da segnalare la presenza di una persona.
2. oltre all'intervallo di 10 minuti abbiamo pensato di utilizzare due intervalli da 5 minuti in modo da considerare esattamente in quale dei due intervalli è avvenuto un dato evento e in questo modo si riesce a scongiurare un possibile problema che potrebbe avvenire nel caso in cui avvengano un numero di eventi maggiori della soglia nell'ultimi minuti dell'intervallo da 10 minuti e nei primi minuti dell'intervallo da 10 minuti successivo.
3. controllo dopo un tempo di 60 minuti della variabile `sound_is_present` in modo da verificare se è presente oppure non presente un individuo.

Combinazione Sensori Rilevamento e modifica set-points

Andando a combinare le variabili `pir_is_present` e `sound_is_present` si riesce a determinare se è presente un individuo oppure no, infatti basta che uno dei due sensori abbia rilevato una presenza che la variabile `is_present` viene modificata in modo opportuno per segnalare la presenza di un individuo. Il controllo avviene nella funzione `PIRtimeout()`, poiché l'intervallo di rilevazione del PIR è minore rispetto all'intervallo di rilevazione del sensore di rumore.

La modifica dei set-points avviene passando i valori da modificare per condizionatore e riscaldatore tramite seriale, con la funzione `insertSetPoints()`, rispettando il seguente formato: per il condizionatore { min_presente, max_presente, min_non_presente, max_non_presente } lo stesso formato vale anche per il riscaldatore. I valori vengono salvati in due vettori e attraverso la funzione `changeSetPoints()` vengono assegnati.

Visualizzazione su display LCD

Attraverso l'utilizzo di un display LCD visualizziamo il valore della temperatura misurata, la percentuale di attivazione della ventola (condizionatore) e del led (riscaldatore). Inoltre si rappresenta attraverso la variabile `is_present` se è presente un individuo e si rappresentano i valori correnti dei 4 set-points, 2 per il condizionatore e 2 per il riscaldatore.

Lampadina smart

Per la lampadina smart abbiamo pensato, come nel caso della rilevazione dei rumori, di utilizzare un intervallo di controllo pari a 500 ms che permette di avere una buona risposta nel caso in cui venga percepito un rumore andando a 'saltare' l'eventuale eco del segnale ed inoltre è un buon compromesso se si considera il tempo di attivazione. Il controllo sulla rilevazione dei rumori prevede che dopo il primo rumore registrato si vada ad incrementare una variabile, in questo modo raggiunte le due rilevazioni si considerano i segnali ricevuti come due battiti di mani e perciò si accende il led. Tra le due rilevazioni deve passare almeno 400 ms altrimenti i rumori misurati nei ms precedenti sono considerati come eco del primo. Il procedimento di spegnimento del led è simmetrico ed inoltre nel caso in cui non vengano registrati rumori il led viene spento andando a controllare il valore di `is_present` .

Hardware: Laboratorio 3

ESERCIZIO_3.1

Come da richieste, per sviluppare l'esercizio è stato implemento un server capace di gestire richieste GET. Dopo l'inizializzazione del server nel `setup`, vengono accettati i client nel `loop` e successivamente processati per elaborarli e rispondere correttamente alle richieste. Nella funzione `process` viene analizzato il contenuto dell'uri passato al server. Se i parametri sono corretti tramite la funzione `printResponse` viene restituito al client il valore richiesto tramite un JSON contenente i dati, al contrario se i parametri non sono corretti viene restituito il codice dell'errore tramite le funzioni di `printErrResponse`, `printTempResponse` e `printBadCommand`.

ESERCIZIO_3.2

In questo esercizio la Yùn deve effettuare richieste POST verso un server, in modo da fornire al server dati riguardanti la temperatura in modio periodico. Per gestire le richieste si è utilizzata la funzione `postRequest` la quale utilizza la libreria `Process.h` per gestire il programma "curl" che si trova installato sul processore Linux della Yùn. Il programma "curl" riceve in ingresso, dopo un'opportuna serie di comandi, il body della POST, il quale contiene il dato di temperatura formattato in SenML. Il comando `p.exitValue` ritorna il codice dello stato dell'esecuzione del processo, se il codice di ritorno è diverso da 0 allora si è verificato un problema nell'esecuzione e vado a stampare sul seriale il codice. La seconda parte dell'esercizio prevedeva la modifica degli Esercizi 1,2 del laboratorio di Software, infatti come riportato nel file "server.py" si implementa la funzione POST, in modo tale tutte le richieste dirette alla risorsa "/log" vengono gestite andando a memorizzare il contenuto JSON all'interno di una lista. Mentre la funzione GET, se riceve richieste sulla stessa risorsa deve ritornare la lista di tutti i JSON memorizzati nella lista.

ESERCIZIO_3.3

La realizzazione di uno sketch per poter far comunicare la Yùn via MQTT è stata possibile andando ad utilizzare la libreria `MQTTclient.h`, basata su mosquitto che deve essere installato sul processore Linux. La Yùn in questo esercizio deve comportarsi sia da subscriber che da publisher, per questo motivo abbiamo utilizzato due topic diversi: topic_publish="/tiot/23/temperature" e topic_subscribe="/tiot/23/led".
L'operazione di publish prevede l'invio di dati formattati in SenML ad un server ogni 10 secondi. Mentre l'operazione di subscribe prevede la sottoscrizione a `topic_subscribe`, in questo caso la Yùn si aspetta di ricevere un file JSON attraverso il quale ricavare i valori per andare ad operare sul LED. Per controllare il formato dei dati ricevuti si utilizza il comando `deserializeJson` che fornisce oltre all'operazione di deserializzazione anche un valore in uscita per il controllo del formato.