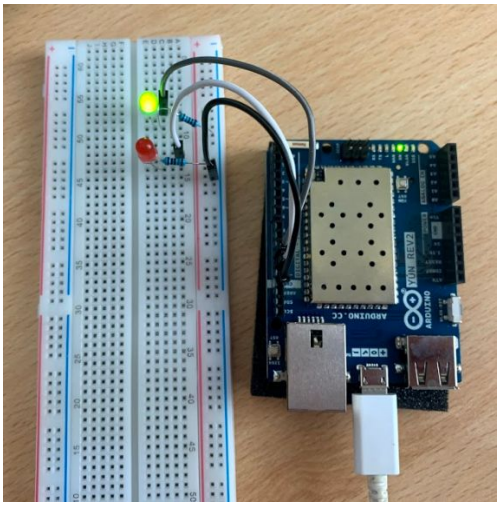
Tecnologie per IoT A.A. 2019/2020 - Report di laboratorio - Gruppo 16
250764 - Manco Davide Ioan
250011 - Manco Marco
248711 - Corvaglia Alessio

LABORATORIO HARDWARE 1

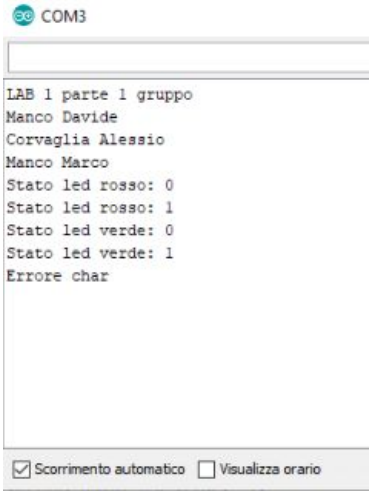
ESERCIZIO 1

Abbiamo collegato i due led ai pin 11 e 12 dell'Arduino Yùn attraverso una resistenza per limitare la corrente erogata e non danneggiarli. Dopo aver definito le varie costanti, relative al numero del pin al quale abbiamo collegato i due led, tramite la funzione setup abbiamo inizializzato i pin in modalità output. Avendo frequenze di lampeggiamento diverse e non multiple tra loro, il led verde è stato gestito tramite interrupt, mentre il led rosso tramite loop() e delay().



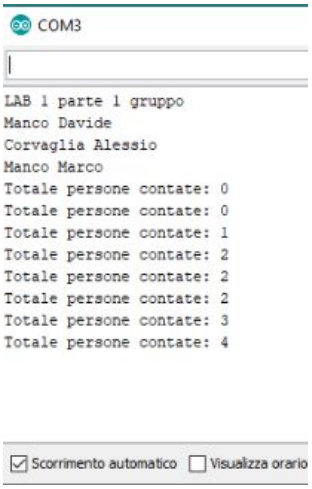
ESERCIZIO 2

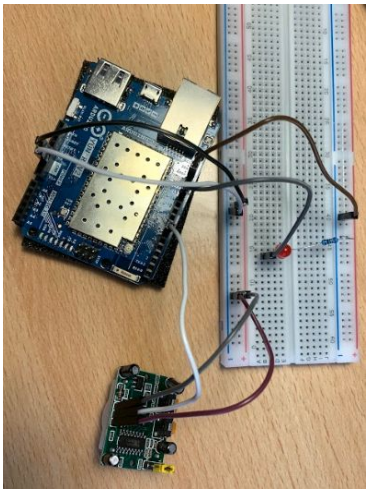
Si tratta del precedente sketch con aggiunta la possibilità di monitor seriale, dello stato dei led. Quando digitiamo "R" o "G" visualizziamo lo stato acceso/spento rispettivamente dei led rosso e verde. Dopo aver inizializzato la connessione seriale con il rate di trasmissione, nella funzione loop leggiamo ciclicamente il carattere immesso dall'utente e visualizziamo il relativo stato attuale del led.



ESERCIZIO 3

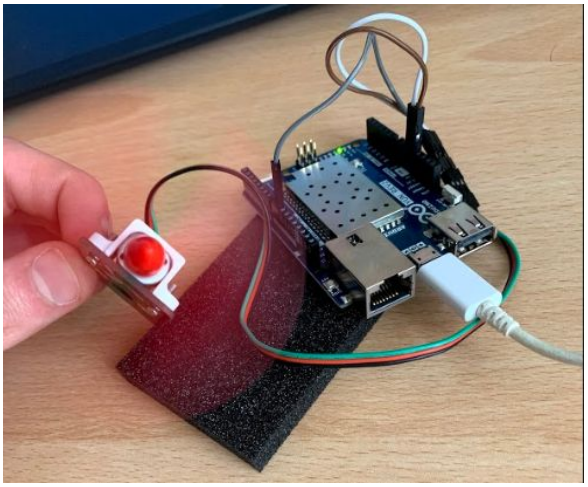
Dopo aver collegato il sensore PIR ed il LED come in foto, abbiamo inserito una procedura di interrupt legata al pin 7, per gestire gli interrupt ricevuti dal sensore PIR. Ad ogni interrupt ricevuto la funzione func_pir() fa variare lo stato del led e aggiorna il numero di persone contate. La funzione loop() ogni 30 secondi stampa sul monitor seriale il contatore delle persone. Il GPIO 7 del PIR è configurato come input mentre il GPIO 12 del led come output.

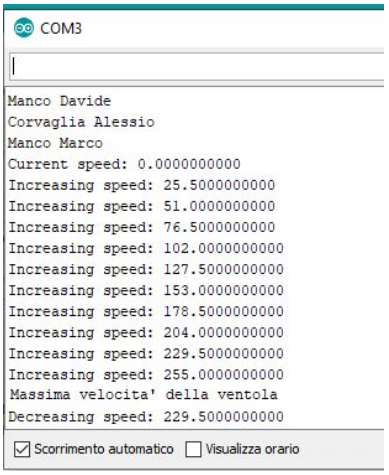




ESERCIZIO 4

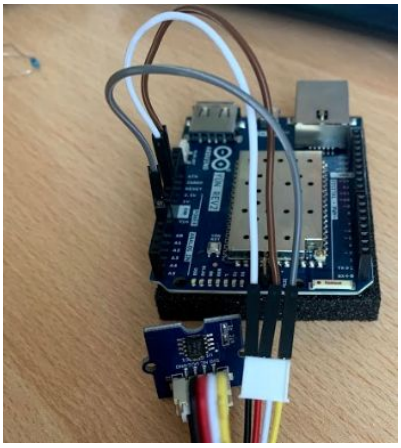
Dopo aver collegato la ventola al GPIO 9 impostato come output, inizializzato la connessione seriale e suddiviso il range di valori da 0 a 255 in 10 step ognuno da 25.5, abbiamo scritto la funzionalità PWM che regola la velocità in base al segno inserito dall'utente (+ e -) nella funzione ciclica loop(). Inoltre abbiamo evitato di uscire fuori range coi valori dando i messaggi di massima/minima velocità raggiunta.

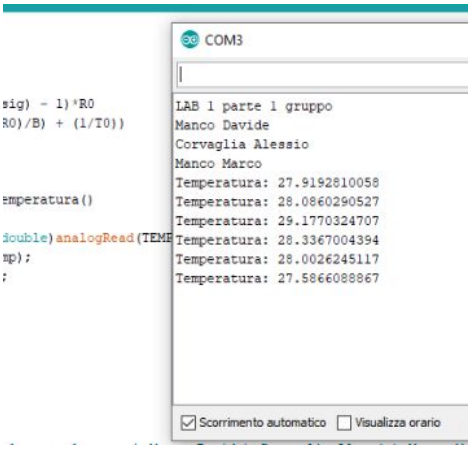




ESERCIZIO 5

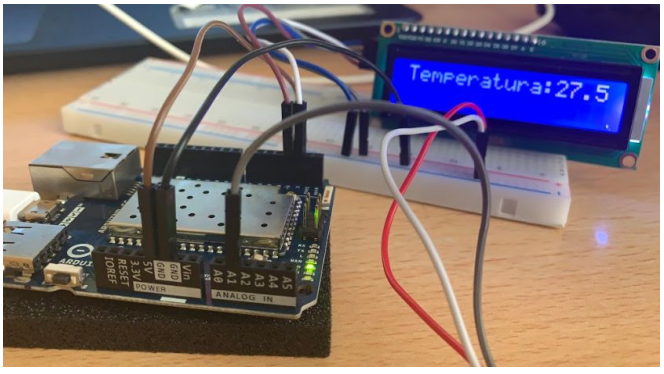
Dopo aver collegato il sensore, abbiamo implementato la sua funzione di taratura, passando da lettura di una tensione (convertita in digitale) alla relativa resistenza e da essa alla temperatura, ponendo attenzione alle conversioni tra °C e °K. Le varie conversioni sono visibili nel codice, in particolare nelle due macro R(vsig) e T(r). Il pin del sensore A1 è stato impostato come input, dopo aver letto il valore del sensore e fatto le conversioni, si stampa la temperatura finale nella funzione loop, ogni 10 secondi.





ESERCIZIO 6

Si tratta del precedente script, ma con output effettuato su un display LCD. Dopo aver incluso la libreria del display, creato un'istanza della classe (chiamandola lcd) ed averne inizializzato i parametri iniziali (dimensione in righe e colonne, retroilluminazione ecc.) abbiamo effettuato la print della frase iniziale "Temperatura:". Dopodichè, per evitare di riscrivere per intero il contenuto del display ad ogni refresh (quindi risparmiando l'invio di dati inutili sul bus I2C, protocollo col quale il display comunica con la board), tramite la funzione stampa_su_display() posizioniamo il cursore nel punto corretto e ogni 10 secondi andiamo ad aggiornare il valore di temperatura.



250764 - Manco Davide Ioan
250011 - Manco Marco
248711 - Corvaglia Alessio

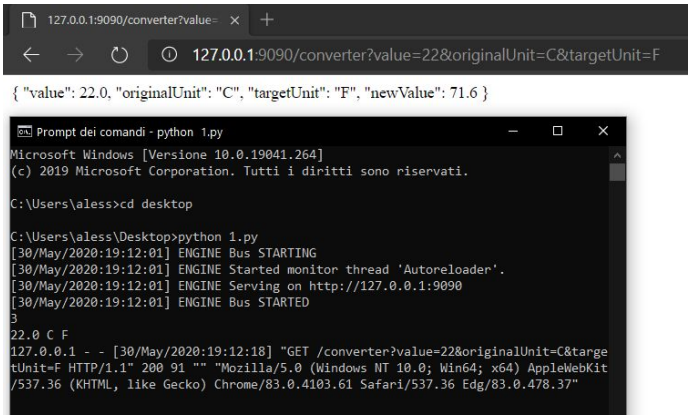
LABORATORIO SOFTWARE 1

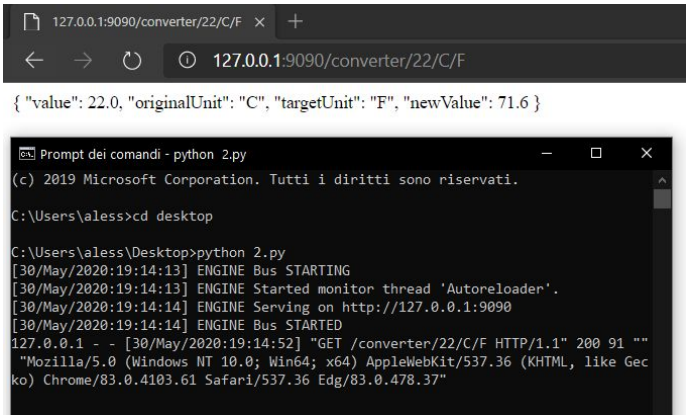
ESERCIZIO 1

Abbiamo sviluppato una classe Converter, ed all'interno il metodo GET, che inizialmente si occupa di controllare il numero di parametri passati ed in caso segnalare errore. Dopodichè vengono letti i parametri passati, effettuando il cast da stringa a float del valore di temperatura da convertire. Si stampano i dati della richiesta, si crea la struttura iniziale secondo lo standard JSON, si converte la temperatura tramite il metodo converting(), si aggiunge il valore convertito in una nuova voce del JSON creato e lo si ritorna.Il metodo converting(), riceve i parametri immessi dall'utente e in base alla targetUnit decide il metodo adeguato per la conversione. Ognuno dei tre metodi fromCelsiusToOther(), fromKelvinToOther(), fromFahrenheitToOther() riceve la originalUnit ed il valore ed effettua la conversione ritornando il risultato.
Il main di occupa di effettuare il montaggio del web server, sfruttando il framework CherryPy, in particolare viene esposta tutta la classe Converter, la quale in particolar modo al nostro scopo espone il metodo GET. Il web server è raggiungibile sulla porta 9090 dell'interfaccia di loopback locale 127.0.0.1.
I parametri sono passati con la classica sintassi del metodo GET, secondo un URL del tipo:
http://localhost:9090/converter?value=22&originalUnit=K&targetUnit=C

ESERCIZIO 2

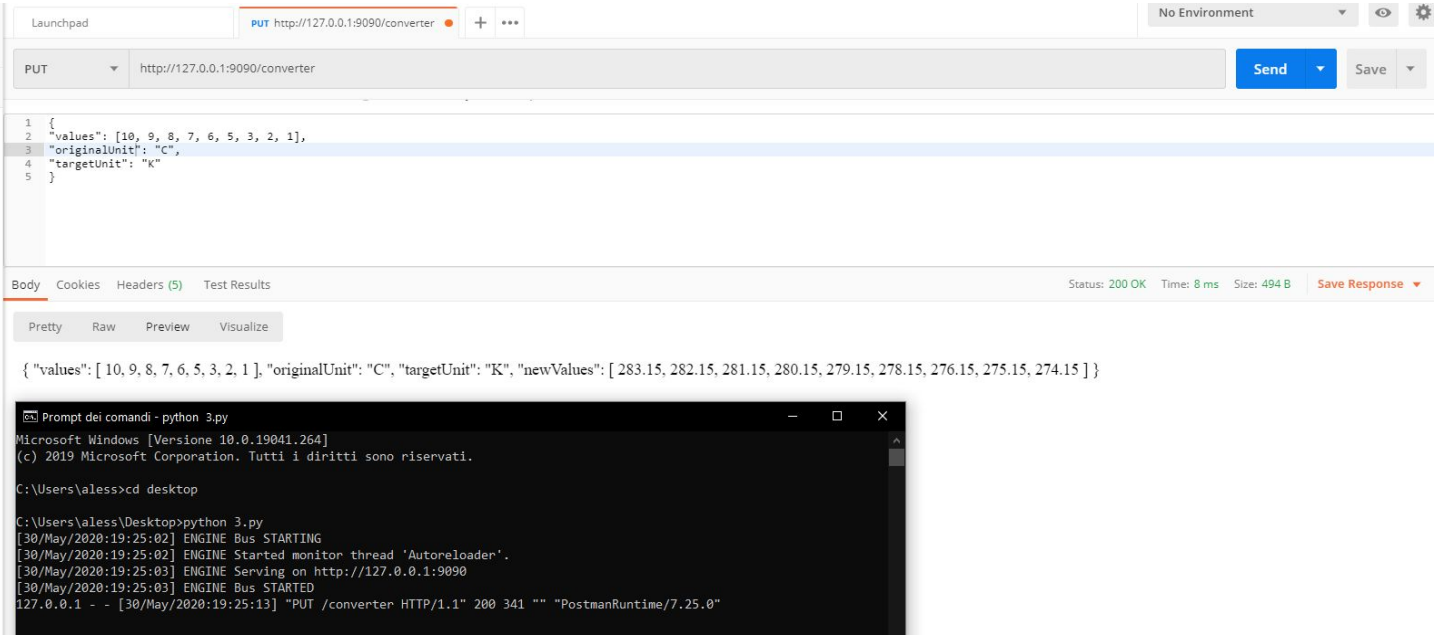
Il secondo esercizio mantiene la maggior parte del codice del precedente, infatti il codice inserito di seguito è solo quello relativo al metodo GET, unico ad essere cambiato. Viene usato l'URI (Universal Resource Identifier) in cui i tre parametri inseriti dall'utente vengono separati dal carattere "/".





ESERCIZIO 3

Anche per questo esercizio 3 è stata riutilizzata una buona parte del codice precedente. Il metodo GET viene sostituito dal metodo PUT che si occupa di ricevere un JSON nel body della richiesta (e quindi non più tramite parametri passati tramite indirizzo). Il JSON contiene una serie di temperature, con una originalUnit nella quale sono espresse ed una targetUnit verso la quale l'utente vuole effettuare la conversione. Per effettuare la richiesta PUT abbiamo utilizzato il software Postman (screen in alto) che consente di inviare le richieste HTTP verso un web server. Tramite il metodo loads() della classe JSON abbiamo tentato la lettura dei dati (dando errore se questa non va a buon fine) e per ogni temperatura effettuato la conversione ed inserita nella lista di temperature convertite che viene infine aggiunta al JSON iniziale e ritornata.



ESERCIZIO 4

In questo ultimo esercizio, abbiamo offerto due metodi. Il metodo GET si occupa di esporre la index.html e tutte le directory (rendendole quindi delle risorse accessibili) della una piattaforma open source freeboard. Il metodo POST invece si occupa di prelevare la configurazione corrente della dashboard freeboard e di salvarne in formato JSON la sua configurazione in un'apposita directory.

LABORATORIO HARDWARE 2

Nell'unico esercizio di questa esercitazione di laboratorio, abbiamo realizzato una sorta di smart-home controller in cui i diversi dispositivi interagiscono tra di loro.

La parte iniziale del codice si occupa di definire tutte le diverse costanti, variabili globali, prototipi, funzioni macro (per la ripartizione di intervalli di temperatura e le conversioni per la lettura della temperatura), import delle librerie e definizione delle costante relative al numero dei pin a cui i diversi dispositivi sono collegati.

La funzione setup() si occupa dell'inizializzazione della connessione seriale, utile per la lettura dei messaggi e l'input dei range di temperatura da parte dell'utente. Dopodichè vengono settati i vari pin in base al loro scopo (ad esempio i pin dei led e della ventola sono in output in quanto comandati dallo sketch mentre i pin dei sensori PIR, di rumore e di temperatura sono settati in input in quanto forniscono dati allo sketch). Viene inoltre inizializzata la visualizzazione del display LCD che ci permetterà di visualizzare tutta la configurazione attuale del nostro smart home controller. Inoltre si configura il pin del sensore PIR come pin dal quale ricevere degli interrupt, mentre il sensore di rumore viene gestito tramite la funzione loop(), almeno per quanto riguarda la funzionalità di presenza persone (nella funzionalità di accensione led con doppio battito di mani viene gestito tramite interrupt).

Nella funzione loop() vengono ciclicamente eseguite le diverse operazioni dello smart home controller. Viene inizialmente controllata la presenza di eventuali richieste da parte dell'utente di settare diversamente dai valori di default i set-point di temperatura. Questa funzionalità viene realizzata tramite la funzione cambia_val_temp() che opera secondo la sintassi <dispositivo>;<valore minimo>;<valore massimo>
Un esempio di comando valido è: L;21;25; che imposta i set point minimo e massimo del led riscaldatore rispettivamente a 21° e 25°. (idem per la ventola usando 'V' anzichè 'L').

Viene poi letta la temperatura dall'apposito sensore, con lo stesso identico procedimento del laboratorio HW 1 esercizio 5, con le dovute conversioni e tramite le funzioni regola_ventola() e regola_led() che ricevono in ingresso la temperatura corrente, si impostano le potenze del dispositivo riscaldante e rinfrescante. Entrambe le funzioni verificano inizialmente se impostare la potenza minima o massima o se impostare una potenza intermedia calcolata attraverso un'apposita macro di mapping MAP che si occupa di suddividere il range di temperature di attivazione dei dispositivi e di calcolare la giusta potenza di attivazione. Infine viene effettuato l'analogWrite sul pin del valore di potenza, i quanto sia la ventola sia il led sono gestiti con PWM (Pulse Width Modulation).

Sempre nella loop() viene gestita la stampa della configurazione sul display LCD. Due schermate differenti si alternano ogni 5 secondi. La prima riporta l'attuale valore di temperatura, se è presente o no qualcuno nella stanza, e la potenza di attivazione dei due dispositivi per il condizionamento dell'ambiente. La seconda schermata riporta i 4 set-point di temperatura attualmente in vigore. Dopo aver effettuato le dovute concatenazioni tra stringhe, queste vengono passate alla funzione stampa_on_lcd() che dopo aver ripulito il display stampa prima una riga e poi la successiva posizionando correttamente il cursore di scrittura.

Per i set-point di temperatura sono presenti due configurazioni, ognuna contenente 4 set-point (min e max per il led e min e max per la ventola) che si attivano quando viene rilevata o meno una presenza in casa. La presenza può essere rilevata congiuntamente o dal sensore PIR o dal sensore di rumore (si tratta di un semplice caso di SENSOR FUSION). La funzione regola_valori() richiamata periodicamente dalla loop si occupa di controllare le variabili di presenza di entrambi i sensori e se entrambi sono a 0 imposta i set-point di assenza, altrimenti imposta i set-point di presenza.
Quando però l'utente decide di settare i suoi range tramite l'interfaccia seriale, col metodo illustrato prima, questi non verranno più modificati in quanto il setting dell'utente ha una priorità maggiore. Pertanto anche solo dopo aver modificato uno dei due set-point di un dispositivo, una variabile valori_mod viene settata ad 1 ed un controllo preventivo impedirà, qualora venisse rilevata una presenza/assenza di modificare i set-point.

Il sensore PIR rileva le presenze tramite interrupt, pertanto all'arrivo di un interrupt si attiva la funzione people_in_room_pir() la quale salva il momento di rilevazione e segnala la presenza. La funzione loop() si occupa invece di richiamare periodicamente la funzione controlla_people_in_room_pir() la quale confronta l'ultima rilevazione del sensore di presenza, con il timestamp attuale, se è stato superato un determinato intervallo di tempo dall'ultima rilevazione, la variabile che memorizza la presenza o meno, torna a 0.

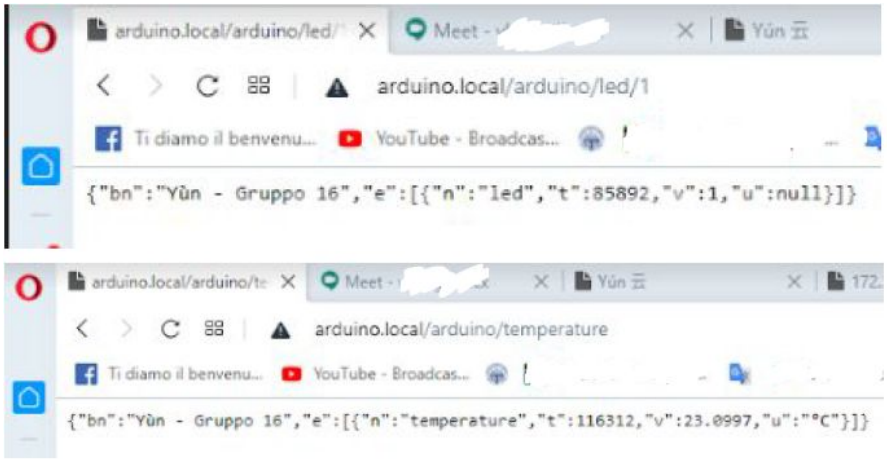
Allo stesso modo viene gestito il sensore di rumore, anche se per esso, per segnalare la presenza di una persona, è necessario effettuare un certo numero di eventi rumore in un dato lasso di tempo, questo per evitare che rumori sporadici portino ad un falso positivo. Pertanto viene gestita una "finestra mobile" all'interno della quale si memorizzano tutti i timestamp degli eventi rumore accaduti in un dato intervallo di tempo, rimuovendo man mano i vecchi timestamp non più utili. Se viene superata la soglia prefissata di eventi, viene segnalata la presenza.

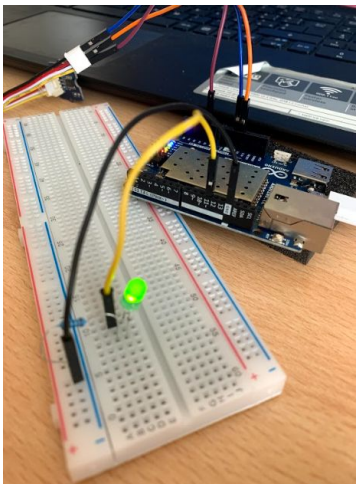
Infine è stata implementata una funzionalità extra, che ha però richiesto di rimuovere la funzionalità di rilevamento presenza tramite sensore di rumore. Tali parti di codice, sono comunque presenti nello sketch ma sono state commentate. La funzionalità consiste nell'accensione/spegnimento del led verde attraverso un doppio battito di mani. In questo caso è stato gestito il rilevamento dell'evento da parte del sensore con il meccanismo degli interrupt. I due battiti non devono avvenire ad una distanza maggiore di 3 secondi l'uno dall'altro, Anche qui si gestisce il tutto grazie ai timestamp degli eventi che permettono di rilevare due eventi sonori ravvicinati.

LABORATORIO HARDWARE 3

ESERCIZIO 1

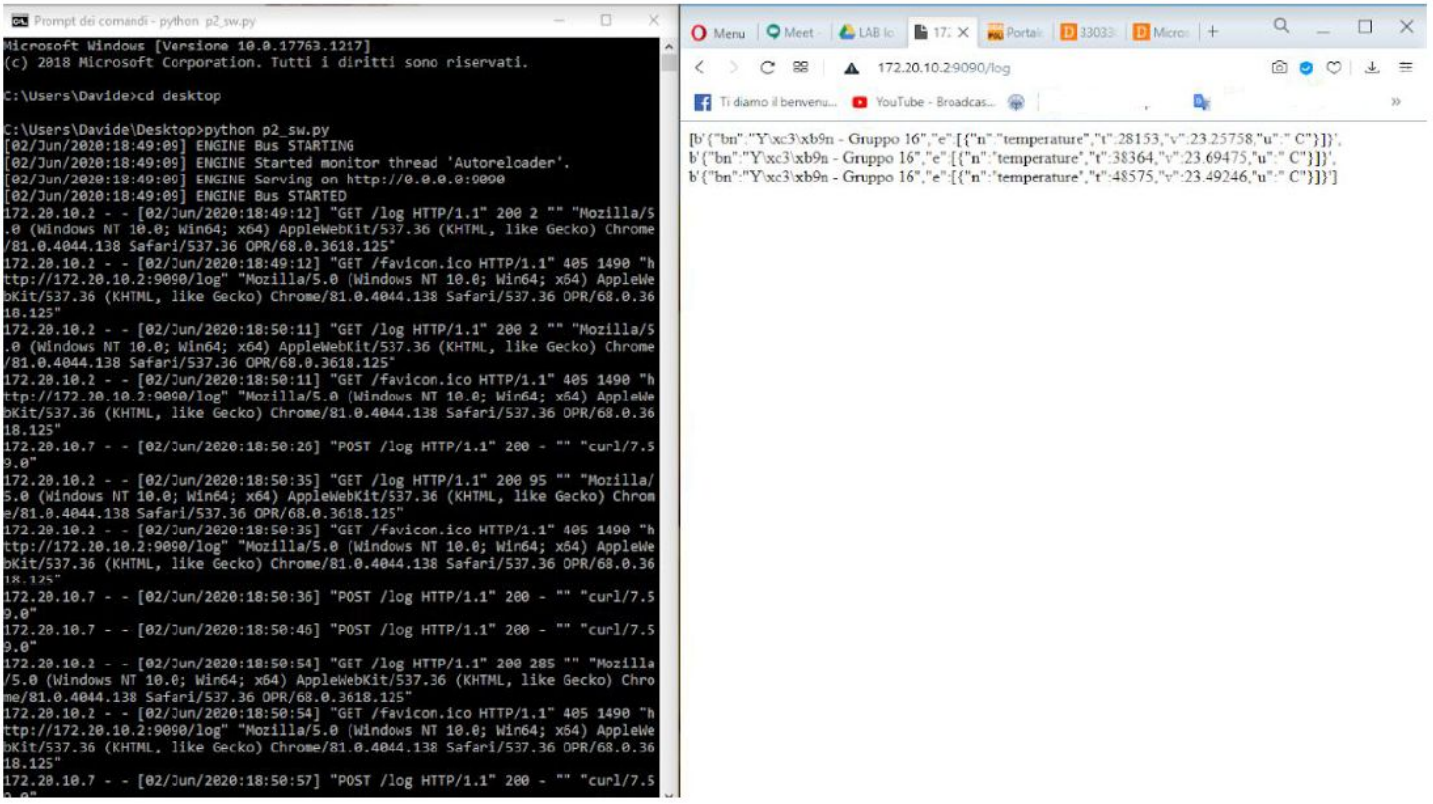
In questo esercizio implementiamo la funzionalità di arduino di poter rispondere a delle GET HTTP. In particolar modo tramite una GET del tipo: “http://arduino.local/arduino/led/1” possiamo accendere il LED ed analogamente con uno 0 spegnerlo. Mentre con una get “http://arduino.local/arduino/temperature” riceviamo una risposta contenente il JSON in formato SenML della misura del sensore di temperatura. Per far ciò dobbiamo interfacciarci tramite un bridge al processore linux presente su arduino. Nella funzione di setup inizializziamo il bridge con Bridge.begin(). Inoltre abbiamo un oggetto di tipo BridgeServer che si occupa di ascoltare le richieste in arrivo. Nella funzione loop() invece, alla ricezione di una richiesta sul server, la accettiamo e costruiamo l’oggetto di tipo BridgeClient. Una volta accettato il client si processa la richiesta e si chiude la connessione. La funzione process_req() riceve in input il client e ne legge l’URL, distingue parsificando la stringa tra comandi per il led (dove controlla che il comando sia valido ed effettua la digitalWrite sul pin del led) e per la temperatura (in cui lancia la funzione di lettura di quest’ultima). Se il comando per il led è sbagliato (quindi diverso da 0 o 1 viene ritornato un errore HTTP 400, ovvero “Bad Request”) oppure se si richiede una risorsa non esistente viene ritornato un errore HTTP 404, ovvero “Not Found”). La logica per la misurazione delle temperatura è identica a quella vista nei precedenti esercizi. In entrambe le richieste viene ritornato un JSON in formato SenML (per il led il nuovo stato impostato e per la temperatura la misura effettuata). Il formato SenML, richiede di includere il nome del sensore, il valore della lettura, la sua unità di misura, ed il timestamp del momento in cui la lettura è stata effettuata. La funzione encode_sen_ml(), riceve i vari dati e attraverso la libreria ArduinoJson permette di manipolare un oggetto come se fosse un array multidimensionale, quindi molto più semplice rispetto alla concatenazione “manuale” di stringhe. Il JSON serializzato (quindi trasformato in stringa) viene inviato alla funzione print_resp() che si occupa di effettuare delle println con il contenuto appropriato sull’oggetto BridgeClient.





ESERCIZIO 2

In questo esercizio, arduino effettua invece l'invio di richieste HTTP POST, inviando periodicamente verso un server la lettura in JSON, formattata in SenML, del sensore di temperatura. Oltre alle classiche inizializzazioni in setup(), la funzione loop() effettua la solita lettura di temperatura che viene codificata in JSON e SenML come nel precedente esercizio. Questa stringa viene inviata alla funzione postRequest() che si occupa, tramite la libreria Process di avviare un processo sul processore Linux di Arduino. Quindi tramite il comando curl, componiamo una richiesta HTTP POST contenente i dati di temperatura, verso un server. Il processo viene eseguito ed in caso di errori, questi vengono stampati sul monitor seriale. Il server che riceve le richieste POST non è altro che una evoluzione degli esercizi 1 e 2 del laboratorio software 1, realizzati in Python e con CherryPy. Infatti questi sono stati modificati per essere in grado di ascoltare le richieste POST ricevute e di salvare in una struttura dati tutte le letture. Quando al server viene inviata una richiesta GET sulla risorsa "/log" allora il server ci restituisce tutte quante le lettura immagazzinate. Ciò è stato realizzato modificando i metodi GET e POST delle precedenti versioni degli esercizi.



ESERCIZIO 3

In questo ultimo esercizio anziché utilizzare il paradigma Client/Server con HTTP, è stato utilizzato il paradigma publish/subscribe con MQTT. In particolar modo, tramite una libreria esterna MQTTClient inclusa nello sketch abbiamo effettuato due operazioni:

1) Subscribe sul topic tiot/16/led. Monitorando questo topic è possibile ricevere JSON in SenML per accendere e spegnere il led.

2) Publish periodico delle letture del sensore di temperatura (in JSON e SenML) sul topic tiot/16/temperature.

Dunque la funzione setup(), oltre alle usuali operazioni avvia il client MQTT sul message broker “mqtt.eclipse.org” sulla porta 1883 (un broker pubblico per effettuare piccoli test) e si sottoscrive al topic del led, specificando di attivare la funzione di callback changeLedValue() per mettere in atto l’operazione all’arrivo di un nuovo messaggio dal message broker.

La funzione loop() effettua il monitoring periodico del topic sottoscritto, legge la temperatura, la codifica esattamente come negli esercizi precedenti e tramite publish() invia il nuovo messaggio con la lettura sul message broker.

Un’aggiunta effettuata in questo script è la deserializzazione di un JSON ricevuto, ovvero quello per il led. Ciò viene fatto sempre con l’ausilio della libreria ArduinoJson, nella funzione changeLedValue() che si occupa di deserializzare il messaggio ricevuto e leggere nei campi appositi il valore da impostare per il led e farne la digitalWrite().

LABORATORIO SOFTWARE 2

ESERCIZIO 1

La classe catalog creata in questo esercizio ci permette di gestire un catalog di dispositivi, servizi e utenti di un sistema IoT distribuito. Tutte le comunicazioni avvengono in formato Restful con JSON. In particolar modo, tramite richieste HTTP PUT permettiamo l'inserimento/aggiornamento di nuove risorse all'interno del catalog. Particolare attenzione è stata fatta alla gestione del timestamp delle diverse operazioni, in quanto per evitare errori di sincronizzazione il tempo viene gestito solo dal catalog. Ogni device ha un id univoco, i suoi endpoint e le risorse che offre.Ogni utente ha un id univoco, nome, cognome, ed email. Ogni servizio ha un id univoco, una descrizione e gli endpoint. Tramite le richieste HTTP GET è possibile accedere alle informazioni del broker MQTT e dell'elenco di dispositivi, servizi e utenti. Inoltre è possibile accedere singolarmente ad ognuno di essi, specificandolo nell'URI. Il catalog inoltre permette il salvataggio e ripristino su file del "database" di tutte le risorse, tramite un'apposita funzione richiamata ad ogni operazione di modifica effettuata nel catalog. Trascorso un determinato intervallo di tempo il catalog si occupa di eliminare tutte le risorse che non sono state aggiornate entro il dato intervallo, considerate quindi "old". Per gestire più operazioni in parallelo, il catalog sfrutta i thread. Il catalog infine permette la gestione degli errori derivanti da richieste errate.

ESERCIZIO 2

Questo esercizio sfrutta/prova tutte le funzionalità di GET offerte dal catalog sviluppato nell'esercizio precedente. Quindi lo script python, tramite la libreria requests, invia le varie richieste HTTP GET per effettuare la visualizzazione/ricerca tramite ID del broker MQTT, dei dispositivi, dei servizi e degli utenti.

ESERCIZIO 3

Questo esercizio permette di simulare un dispositivo IoT, per effettuare il testing della funzione di PUT per i nuovi devices. Quindi periodicamente, viene inviata la richiesta HTTP PUT verso il catalog, con le info del device da inserire.

ESERCIZIO 4

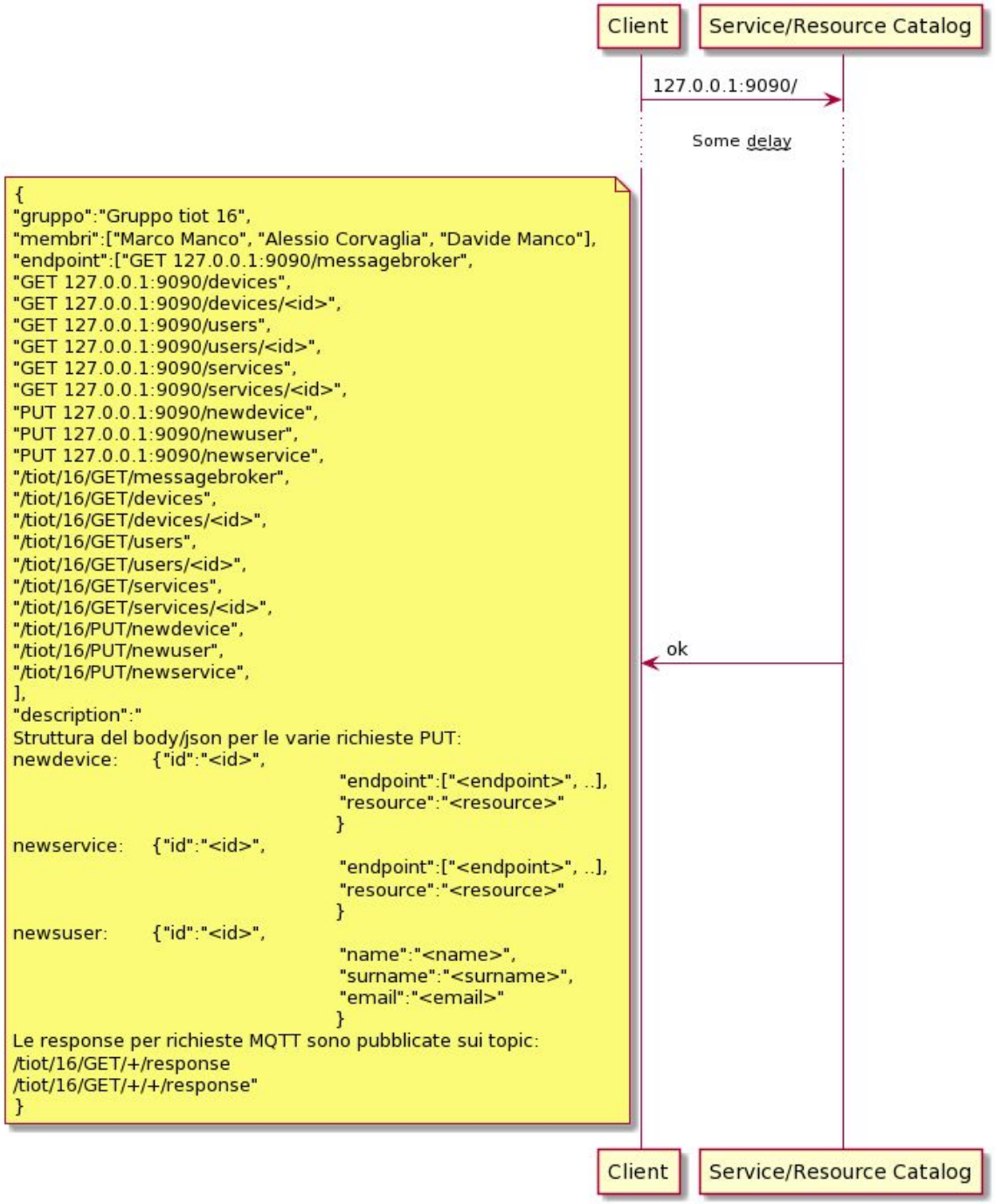
Questo esercizio estende nuovamente il server sviluppato nel lab sw 1, successivamente esteso nel lab hw 3. In particolar modo la board Arduino Yùn invia periodicamente le letture del sensore di temperatura al server sviluppato (che offre il metodo POST per riceverle e GET per offrire tutte le letture ricevute). L'estensione sviluppata consente di registrare e aggiornare periodicamente questo servizio di temperatura sul catalog.

ESERCIZIO 5

Questo esercizio modifica l'esercizio 1 del medesimo laboratorio estendendolo. In particolar modo le funzionalità del catalog, oltre ad essere offerte tramite metodi HTTP, anche tramite MQTT. Pertanto sono stati creati i diversi topic coi quali interfacciarsi, utilizzando le wildcards quanto necessario.

ESERCIZIO 6

In quest'ultimo esercizio, si ha una nuova implementazione delle funzionalità dell'esercizio 3 del medesimo laboratorio. Infatti, si emula un dispositivo IoT, che si sottoscrive al catalog periodicamente, comunicando il proprio id, endpoints e resources, questa volta tramite MQTT, permettendoci quindi di testare il corretto funzionamento dell'esercizio precedente.



LABORATORIO SOFTWARE 3

ESERCIZIO 1

In questo esercizio abbiamo semplicemente approcciato col client Mosquitto ed i suoi comandi per il publish ed il subscribe ai topics, utilizzando anche le wildcards. Questi comandi saranno utili per il debugging degli esercizi successivi.

ESERCIZIO 2

Lo scopo di questo esercizio è quello di estendere le funzionalità dell'esercizio 3 del laboratorio hardware 3, in cui, una volta rimossa la funzionalità dell'accensione/spegnimento del led, lo sketch registra la board sul catalog sviluppato negli esercizi precedenti ed invia periodicamente la lettura della temperatura sull'apposito topic MQTT. Dopodichè attraverso python è stato implementato un MQTT subscriber che si registra al catalog e da esso preleva le informazioni sul broker MQTT utilizzato, ovvero dove la board arduino sta inviando le sue letture di temperatura. Una volta recuperati gli endpoints, sempre dal catalog, il subscriber, si sottoscrive al topic della temperatura ricevendo le letture.

ESERCIZIO 3

Questo esercizio può essere considerato come la controparte del precedente, ovvero vengono invertiti i ruoli. Viene mantenuta sempre l'interazione di servizi e dispositivi attraverso il catalog che funge da punto di aggregazione. La board arduino dunque si registra al catalog come dispositivo e si sottoscrive sul topic del led. Il publisher MQTT, sviluppato in python, si occupa come prima di registrarsi come servizio al catalog, di recuperare le informazioni sul broker MQTT e sugli endpoints del servizio. Una volta ottenuti gli endpoints, effettua il publish per pilotare il led da remoto.

ESERCIZIO 4

Viene ora realizzata la versione connessa ad internet dello smart home controller, realizzato nel laboratorio hardware 2. Si tratta di una coppia di script, che dialogano con il supporto del catalog. La board arduino diventa quindi un dispositivo "stupido" che si occupa di inviare i dati delle letture dei propri sensori (temperatura, rumore e presenza) e di ricevere dati riguardanti l'attuazione dei propri dispositivi (display, ventola e led riscaldatore). L'invio dei dati avviene periodicamente (tramite loop) tramite MQTT su appositi topic. Questi dati sono formattati secondo lo standard JSON e SenML. La ricezione dei dati avviene nelle stesse modalità. Inoltre la board arduino si occupa di registrarsi periodicamente al catalog come device, comunicando le sue risorse ed i suoi endpoints. Allo script python viene assegnata ora tutta la parte di elaborazione, quindi gestione dei set point di temperatura in presenza/assenza di persone, regolazione di ventola e led riscaldatore, impostazione dei set point personalizzati e creazione dei messaggi da stampare sul display. Anche in questo caso si utilizza il catalog, poiché lo script si occupa di comunicare inizialmente con esso tramite REST, permettendo il recupero sulle informazioni del broker MQTT da utilizzare. A questo punto,

ancora attraverso il catalog accede alle risorse necessarie, ovvero le resources e gli endpoints comunicati dalla board arduino e a questo punto procedere con le operazioni di controllo casa descritte prima.

La questione della latenza delle comunicazioni via MQTT, ha fatto sì di lasciare in locale, sulla board arduino, il processamento delle letture per il sensore PIR e di rumore e dunque l'elaborazione finale sulla presenza o meno di qualcuno nella casa.

Il nostro smart home controller è stato ora realizzato su una piattaforma distribuita, dove vi sono più "attori" ed ognuno svolge il suo ruolo. Il sistema risulta così molto più semplice da estendere in quanto è possibile aggiungere semplicemente una nuova board con nuovi sensori e attuatori, che si registri al catalog esponendo le sue risorse. A questo punto basterebbe aggiungere la gestione delle nuove risorse sullo script python ottenendo quindi l'integrazione nel controller dei nuovi dispositivi aggiunti, nella già esistente archiettura di controllo smart.

Questa nuova architettura, permette quindi anche il controllo da remoto attraverso internet, mentre la precedente versione poteva lavorare solo in locale. Allo stesso tempo, essendo il sistema esposto su internet, va particolarmente curato l'aspetto legato alla sicurezza dell'intero sistema.

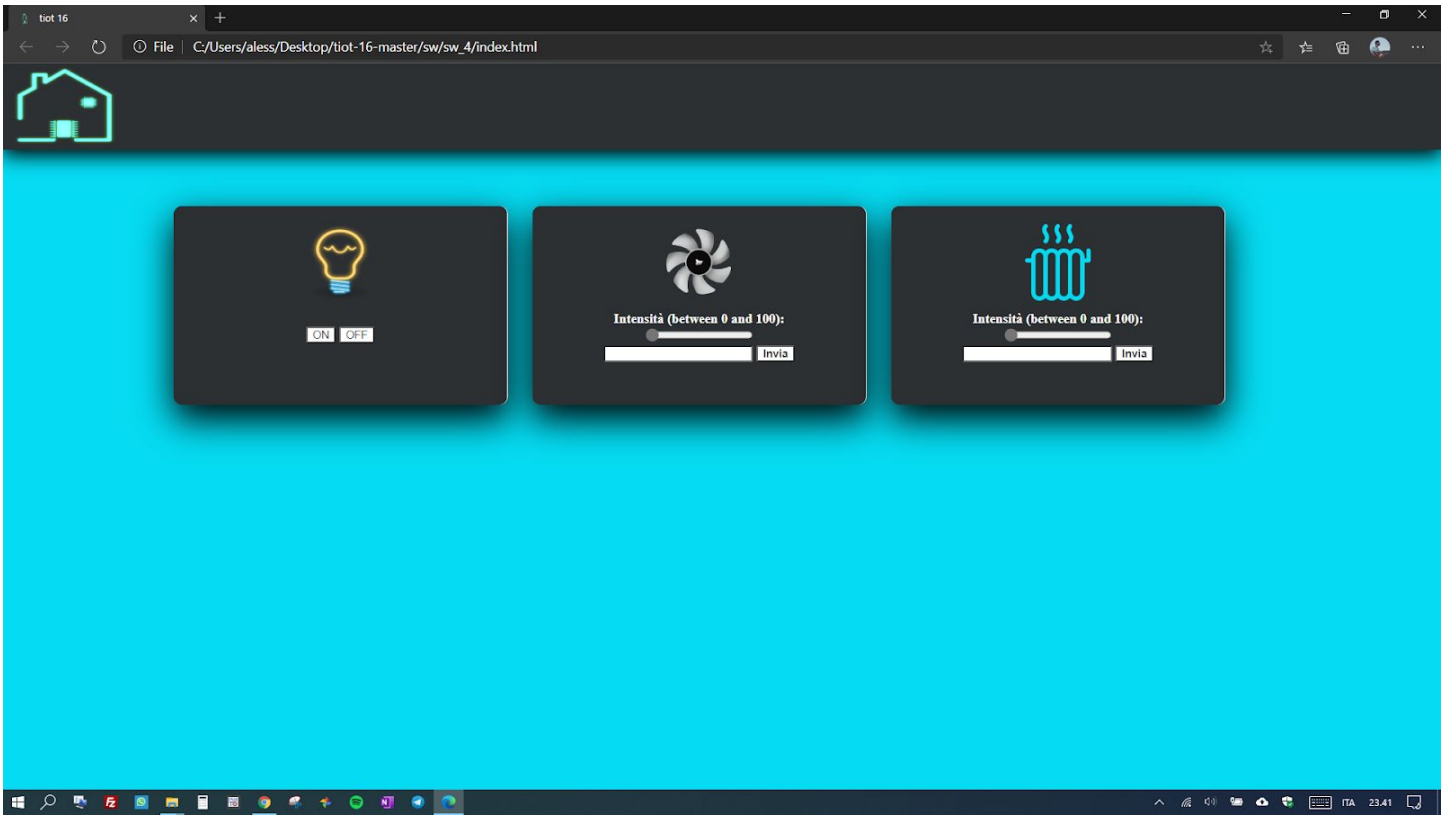
LABORATORIO SOFTWARE 4

Lo scopo di questo ultimo laboratorio è quello di realizzare alcune estensioni della piattaforma sviluppata nell'ultimo esercizio precedentemente descritto, ovvero lo smart home controller remoto. A questo proposito sono state aggiunte le seguenti nuove funzionalità e componenti:

1) Per simulare l'aggiunta di un nuovo dispositivo hardware all'interno dell'infrastruttura, non disponendo di un sensore di luminosità hardware vero e proprio, questo è stato emulato da un piccolo script python che si registra periodicamente al catalog come device offrendo la risorsa "brightness". Questo sensore, genera in maniera random dei valori di luminosità e li invia in formato JSON e SenML, tramite MQTT su un apposito topic.

2) Un nuovo dispositivo hardware è stato aggiunto alla board arduino, il led di colore verde. Questo si occupa di simulare un piccolo faretto. Questo, viene pilotato dal sensore di luminosità prima descritto, ovvero quando la luminosità rilevata dal sensore, scende sotto una certa soglia, allora viene acceso questo faretto per illuminare. Analogamente, quando la luminosità ritorna sopra la soglia fissata, il faretto viene spento. Pertanto lo script python che si occupa di gestire lo smart home controller, elaborerà tra le diverse cose, le misure di luminosità periodicamente ricevute dal sensore ed invierà su un topic MQTT lo stato acceso/spento per pilotare il faretto. (Nota: lo script python che pilota il faretto è separato dallo script dello smart home controller)

3) Realizzazione di una semplice pagina web (*screen dell'interfaccia nella pagina successiva*), che attraverso l'utilizzo di alcuni pulsanti, permette di pilotare manualmente il faretto prima descritto, il condizionatore d'aria (ventola) e il riscaldatore (led rosso). In particolar modo, quando l'utente invia un comando da questa interfaccia, tramite una variabile, l'intelligenza python che gestisce gli attuatori in base agli input dei sensori si ferma, lasciando il pieno controllo manuale all'utente del modo di funzionamento degli attuatori. Quindi, tramite un piccolo web server realizzato tramite CherryPy, esponiamo la pagina index.html, e i file di stile CSS ed immagini dell'interfaccia, rendendole dunque delle risorse, con un URL ben definito. Inoltre viene esposto il metodo GET, metodo attraverso il quale tramite le pressioni dei vari pulsanti, vengono inviate le richieste HTTP GET al webserver, passando i parametri secondo una sintassi del tipo (http://.../luce?subject=on). Il metodo GET si occuperà poi di prendere l'input e di tradurlo in un apposito messaggio MQTT (formattato in JSON e SenML) per effettuarne una publish sul topic corretto in base al dispositivo che l'utente vuole pilotare. Inoltre questo script si occupa anche di avviare il dialogo tramite REST con il catalog, di registrarsi con una publish ad esso periodicamente come servizio (tramite un thread in parallelo), e di recuperare da esso tramite delle subscribe le info sul broker usato e in seguito sugli endpoint dei vari dispositivi


tiot 16
File
C:/Users/aless/Desktop/tiot-16-master/sw/sw_4/index.html
ON
OFF
Intensità (between 0 and 100):
Invia
Intensità (between 0 and 100):
Invia