



Tecnologie per IoT LABORATORI

Gruppo: 17

Barisione(249190) & Pegoraro(245327)

4 luglio 2020

Indice

| | |
|---|-----------|
| 1 HW Parte 1: Introduzione all'ambiente Arduino e primi semplici sketch | 3 |
| 1.1 Pilotaggio di LED | 3 |
| 1.2 Comunicazione tramite Porta Seriale | 3 |
| 1.3 Identificazione della presenza tramite Sensore PIR | 3 |
| 1.4 Controllo di un motore (ventola) tramite PWM | 4 |
| 1.5 Lettura di valori analogici (temperatura) | 4 |
| 1.6 Comunicazione con uno smart actuator (display LCD) tramite protocollo I2C | 5 |
| 2 HW Parte 2: Smart home controller (versione "locale") | 5 |
| 2.1 Smart home controller (versione "locale") | 5 |
| 3 HW Parte 3: Comunicazione tramite interfacce REST e MQTT | 6 |
| 3.1 Arduino Yùn come server HTTP | 6 |
| 3.2 Arduino Yùn come client HTTP | 6 |
| 3.3 Arduino Yùn come publisher e subscriber MQTT | 7 |
| 4 SW Parte 1: Esercitazione di Laboratorio 1 | 8 |
| 4.1 Exercise 1 | 8 |
| 4.2 Exercise 2 | 8 |
| 4.3 Exercise 3 | 8 |
| 4.4 Exercise 4 | 11 |
| 5 SW Parte 2: Lab Software Part 2 | 12 |
| 5.1 Exercise 1 | 12 |
| 5.2 Exercise 2 | 12 |
| 5.3 Exercise 3 | 12 |
| 5.4 Exercise 4 | 12 |
| 5.5 Exercise 5 | 12 |
| 5.6 Exercise 6 | 13 |

| | | |
|----------|--|-----------|
| 6 | SW Parte 3: Lab Software Part 3 | 13 |
| 6.1 | Exercise 1 | 13 |
| 6.2 | Exercise 2 | 14 |
| 6.3 | Exercise 3 | 14 |
| 6.4 | Exercise 4 | 15 |
| 7 | SW Parte 4: Lab Software Part 4 | 16 |

1 HW Parte 1: Introduzione all'ambiente Arduino e primi semplici sketch

1.1 Pilotaggio di LED

Spiegazione

L'esercizio richiedeva di implementare un sketch in grado di pilotare due led per farli lampeggiare con periodi indipendenti fra di loro.

Il LED verde viene pilotato tramite un interrupt mentre il LED rosso è pilotato tramite la funzione inserita all'interno del Void loop, i due LED lampeggiano con due periodi diversi.

1.2 Comunicazione tramite Porta Seriale

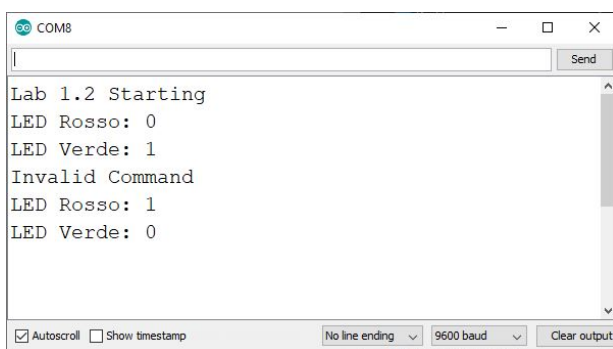
Spiegazione

In questo secondo esercizio veniva richiesto di modificare l'esercizio precedente trasmettendo lo stato del led tramite seriale.

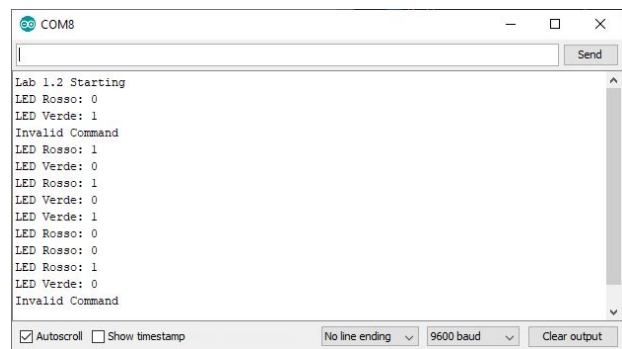
Una volta avviata la seriale viene messo a display un messaggio di benvenuto, dopodiché quest'ultima aspetta in ingresso i caratteri 'R' o 'L'. Ricevendo in input questi caratteri il programma restituisce lo stato dei rispettivi LED (acceso = 1, spento = 0).

Per implementare il "risveglio" della seriale tramite dei caratteri è stata utilizzata la funzione `serialEvent()`.

Risultati



(a) Esempio 1



(b) Esempio 2

Figura 1: Esempio dell'output

1.3 Identificazione della presenza tramite Sensore PIR

Spiegazione

In questo esercizio veniva richiesto di implementare uno sketch in grado di contare il numero di persone che passano di fronte al sensore PIR e stampare sulla seriale, ogni 30 secondi, il numero di persone che sono passate davanti al sensore.

Tutto ciò è stato fatto usando gli interrupt, più nello specifico usando la funzione `attachInterrupt()` e `digitalPinToInterrupt`, per questa funzione bisogna specificare il PIN al quale è collegato il sensore, funzione a cui fa riferimento, cioè la ISR, e a quali fronti è sensibile l'interrupt.

Risultati

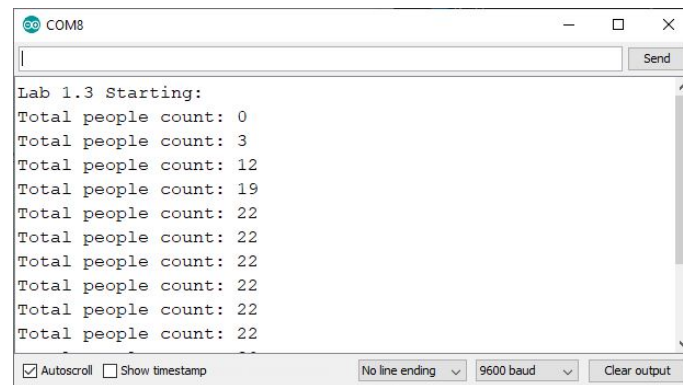


Figura 2: Esempio dell'output

1.4 Controllo di un motore (ventola) tramite PWM

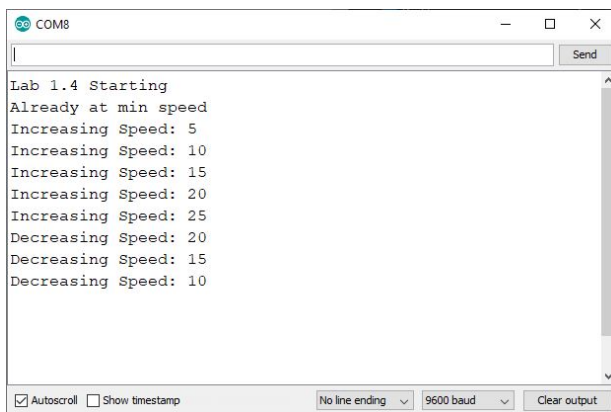
Spiegazione

In questo esercizio veniva richiesto di controllare la rotazione di un motore tramite la modalità PWM.

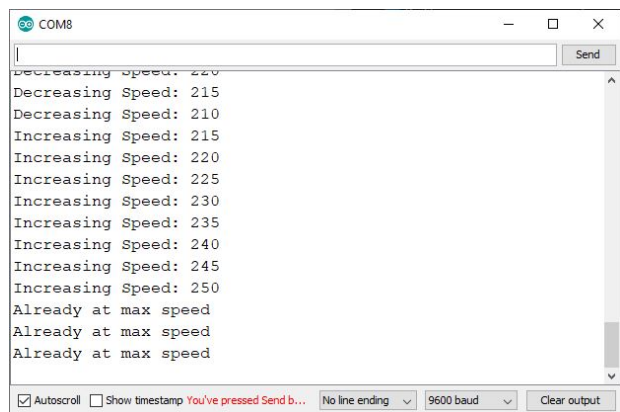
Inoltre ricevendo da seriale il carattere "+" la velocità del motore doveva essere incrementata al contrario ricevendo il carattere "-" la velocità del motore doveva essere decrementata.

Nel caso in cui la velocità massima del motore o la velocità minima fosse stata raggiunta viene stampato su seriale che è stato raggiunto rispettivamente il valore massimo è il valore minimo di velocità.

Risultati



(a) Esempio 1



(b) Esempio 2

Figura 3: Esempio dell'output

1.5 Lettura di valori analogici (temperatura)

Spiegazione

In questo esercizio veniva richiesto di sviluppare uno sketch in grado di monitorare periodicamente la temperatura dell'ambiente e inviare i risultati tramite porta seriale.

Il valore della tensione dei pin viene letto periodicamente ad esempio ogni 10 secondi e convertito in un valore di temperatura mediante le opportune funzioni riportate il datasheet del sensore.

Per convertire il valore della tensione si è usata la seguente funzione:

$$temperature = (1.0/((\log(R/R0)/B) + (1/298.15))) - 273.15;$$

1.6 Comunicazione con uno smart actuator (display LCD) tramite protocollo I2C

Spiegazione

In quest'ultimo esercizio veniva richiesto di modificare l'esercizio precedente andando a stampare la temperatura letta dal sensore sul display LCD. Il tutto è stato svolto usando la libreria `LiquidCrystal_PCF8574`.

2 HW Parte 2: Smart home controller (versione "locale")

2.1 Smart home controller (versione "locale")

Spiegazione

In questo esercizio veniva richiesto di utilizzare Arduino Yun come sistema di controllo per la Smart home. Al fine di svolgere le funzioni richieste sono state definite differenti funzioni:

- la funzione `ReadTemperature()` si occupa di leggere la temperatura del sensore.
- la funzione `CheckPresenceS()` di leggere la presenza di eventuali persone a determinati intervalli di tempo prestabiliti.
la funzione `CheckPresenceP()` si attiva quando viene rilevata una persona nella stanza, se dopo un tempo inferiore a 30 minuti viene rilevato un'ulteriore movimento dal sensore il contatore di persone presenti nella stanza viene incrementato, altrimenti viene riportato di nuovo a zero. La verifica che una seconda persona è stata rilevata in un intervallo di tempo inferiore ai 30 minuti viene fatta dalla funzione `CheckPresenceS()`.
- la funzione di `HeaterOn()` si occupa di accendere il led in base alla temperatura rilevata, e in base ai set-point impostati. Il led si accende se la temperatura letta dal sensore è compresa tra i set-point impostati, se compresa il led si accende con una differente intensità per le diverse temperature.
- la funzione `mooveFan()` si occupa di accendere la ventola se la temperatura rilevata del sensore si trova tra i set-point impostati, anche in questo caso la ventola gira con una differente velocità in base alla temperatura letta.
- usando la funzione `UpdateScreen()` vengono mostrati sul display i valori dei differenti sensori, nello specifico:
 - Il valore attuale della temperatura misurata
 - La percentuale di attivazione della ventola e del riscaldatore (tra 0 e 100%).
 - Quante persone sono state rilevate nella stanza.
 - Il valore attuale dei 4 set-point. Questi set-point sono impostati tramite seriale usando la funzione `serialEvent()`.

È stata svolta anche la parte bonus dell'esercizio la quale richiedeva di accendere il LED verde tramite il doppio battito di mani, lo spegnimento di quest'ultimo poteva venire con un ulteriore doppio battito di mani, oppure automaticamente quando il sistema non rivelava presenza di persone. Questa funzione è stata sviluppata con la funzione `LightUp()`, gestita anch'essa in interrupt.

3 HW Parte 3: Comunicazione tramite interfacce REST e MQTT

3.1 Arduino Yùn come server HTTP

Spiegazione

In questo esercizio veniva chiesto di realizzare uno sketch che eseguisse un server HTTP sulla Yùn, in grado di rispondere a richieste GET provenienti dalla rete locale. Sono state usate le seguenti librerie: `Bridge.h`, `BridgeClient.h`, `BridgeServer.h` e `ArduinoJson.h`. Il server deve esporre come “risorse” uno dei LED ed il sensore di temperatura.

Al fine di implementare le funzioni richieste sono state utilizzate le seguenti funzioni:

- la funzione `ReadTemperature()` si occupa di leggere la temperatura del sensore.
- la funzione `Process()` gestisce le richieste di GET provenienti dalla rete locale, quest’ultima funzione è richiamata all’interno del `Void loop()`.
- la funzione `senMlEncode()` viene usata per codificare la risposta dell’arduino in formato JSON.
- infine la funzione `printResponse()` si occupa di stampare il risultato della richiesta GET fornendo anche il rispettivo codice di errore.

3.2 Arduino Yùn come client HTTP

Spiegazione

In questo caso veniva richiesto di realizzare uno sketch che effettuasse delle richieste HTTP POST dalla Yùn verso un server, per realizzare un logging periodico dei dati del sensore di temperatura. Lo sketch deve leggere periodicamente la temperatura dal sensore e invia una richiesta POST ad un server, il body della POST contenente il dato della temperatura viene formattato in SenML.

Sono state usate le seguenti funzioni:

- la funzione `ReadTemperature()` si occupa di leggere la temperatura del sensore.
- la funzione `Update_Stat()` si occupa di aggiornare il file JSON che verrà inviato come body della POST contenente la temperatura misurata. La funzione `postRequest()` viene usata per inviare le richieste POST, verso un server. Le richieste vengono inviate al seguente URI: `"http://192.168.1.4:8080/log"`
- la funzione `printResponse()` stampa eventuali errori prodotti dalla `curl`

Inoltre veniva richiesto di estendere esercizi 1 e 2 del Lab SW parte 1 in modo da gestire le richieste provenienti dall’Arduino.

E’ stata quindi implementata, in python (tramite l’utilizzo della libreria `cherrypy`), una classe `TConverter()` contenente i seguenti metodi:

- `AppendFile()`, questa funzione viene utilizzata per memorizzare il JSON contenuto nel body della POST proveniente dalla Yun in una lista, questa funzione è quindi richiamata all’interno del metodo POST
- il metodo GET richiama una funzione `LoadFile()` la quale si occupa di restituire l’intera lista come un unico JSON. Perciò nel caso in cui vengano effettuate delle richieste di GET verso la risorsa quest’ultima restituisce l’intera lista formattata come un unico JSON. All’interno del metodo GET viene richiamato anche il metodo `converter` il quale si occupa di convertire i valori della temperatura nel formato specificato nell’URI.

3.3 Arduino Yùn come publisher e subscriber MQTT

Spiegazione

In questo esercizio veniva richiesto di realizzare uno sketch che consenta alla Yùn di comunicare via MQTT, agendo sia come publisher che come subscriber. Per svolgere questa funzione si è utilizzata la libreria *MQTTclient.h*.

Utilizzando la seguente funzione: `mqtt.publish("tiot/17/temperature",message);` viene pubblicato periodicamente in un formato SenML, al topic: *"tiot/17/temperature"* il messaggio contenente il valore della temperatura letta dal sensore. La funzione `senMLEncode()` si occupa di formattare correttamente la temperatura nel formato richiesto, quest'ultima viene letta utilizzando la funzione `ReadTemperature()`.

Infine tramite la seguente funzione: `mqtt.subscribe("tiot/17/led",setLedValue);` dichiarata nel setup, l'arduino si sottoscrive al topic: *"tiot/17/led"*, tramite il quale verranno inviate informazioni in formato SenML per controllare uno dei Led presenti sulla board, le informazioni ricevute sono in formato JSON quest'ultime vengono deserializzate usando la funzione `deserializeJson`. Tramite queste informazioni ricevute viene cambiato lo stato del led.

4 SW Parte 1: Esercitazione di Laboratorio 1

4.1 Exercise 1

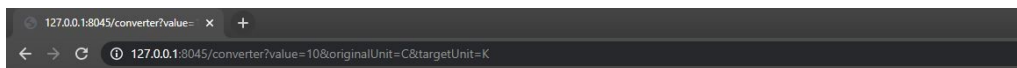
Spiegazione

In questo esercizio veniva richiesto di implementare un servizio web utilizzando il metodo GET (facendo uso della libreria `cherrypy`).

Questo programma si occupa di convertire il valore numerico della temperatura nell'unità di misura specifica nell'URI: `http://localhost:8080/converter?value=10&originalUnit=C&targetUnit=K`, la stringa contiene il valore numerico della temperatura con la corrispettiva unità di misura, e l'unità di misura in cui quest'ultima deve essere convertita.

Il programma dovrà poi restituire un file JSON contenente il valore della temperatura originale con la corrispettiva unità di misura, inoltre dovrà anche contenere il valore finale in cui è stata convertita con la corrispettiva unità di misura finale.

Risultati



[10, "C", 283, "K"]

Figura 4: Valore convertito

4.2 Exercise 2

Spiegazione

In questo esercizio veniva richiesto di implementare la stessa conversione di prima, solo che in questo caso i valori sono separati dal carattere `"/"`. È stato utilizzato il seguente URI: `http://localhost:8080/converter/10/C/K`

4.3 Exercise 3

Spiegazione

In questo esercizio veniva richiesto di sviluppare un codice Python che implementasse la conversione dei gradi in Celsius, Fahrenheit e Kelvin attraverso il metodo PUT.

I valori da convertire con le rispettive unità di misura vengono mandati al programma attraverso un file JSON. Per testare la funzionalità del programma è stato usato il software POSTMAN il quale invia il file JSON e mette a display la risposta del programma cioè la rispettiva conversione dei valori nell'unità di misura specificata.

È stata quindi definita una classe `TConverter()`, contenente la funzione `converter`, la quale si occupa, in base alle unità di misura ricevute nell'URI di convertire correttamente il valore della temperatura. Questa funzione è richiamata all'interno del metodo PUT, il quale si occupa di pubblicare i valori convertiti nel body.

Risultati

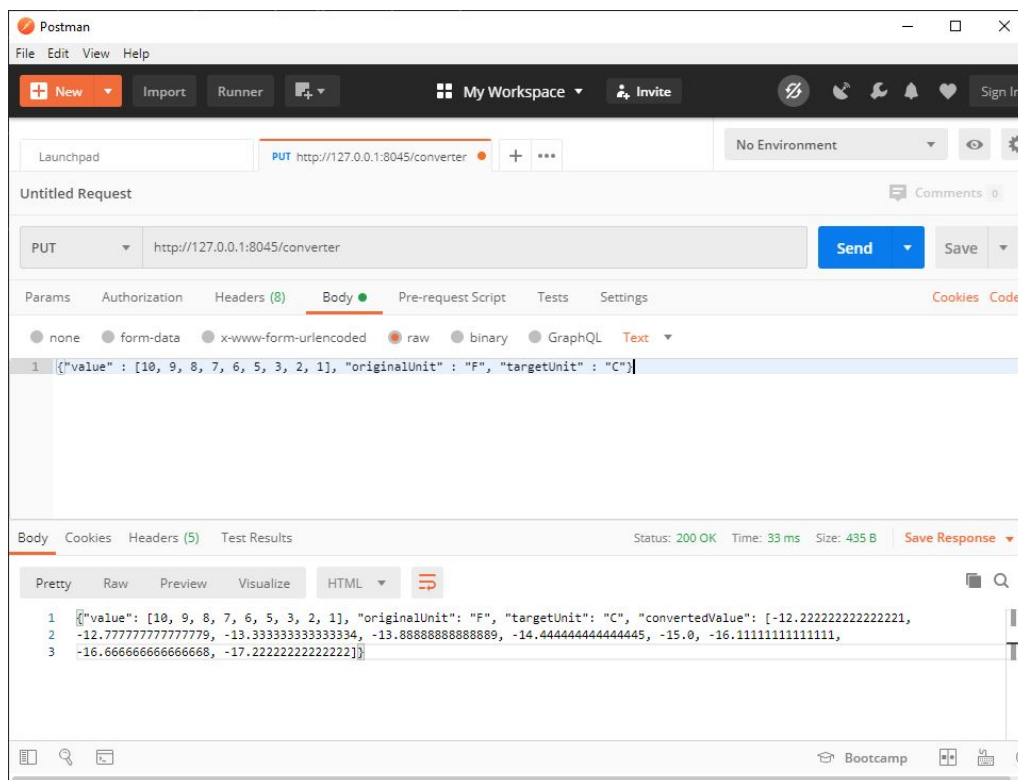


Figura 5: Esempio di conversione da Fahrenheit a Celsius

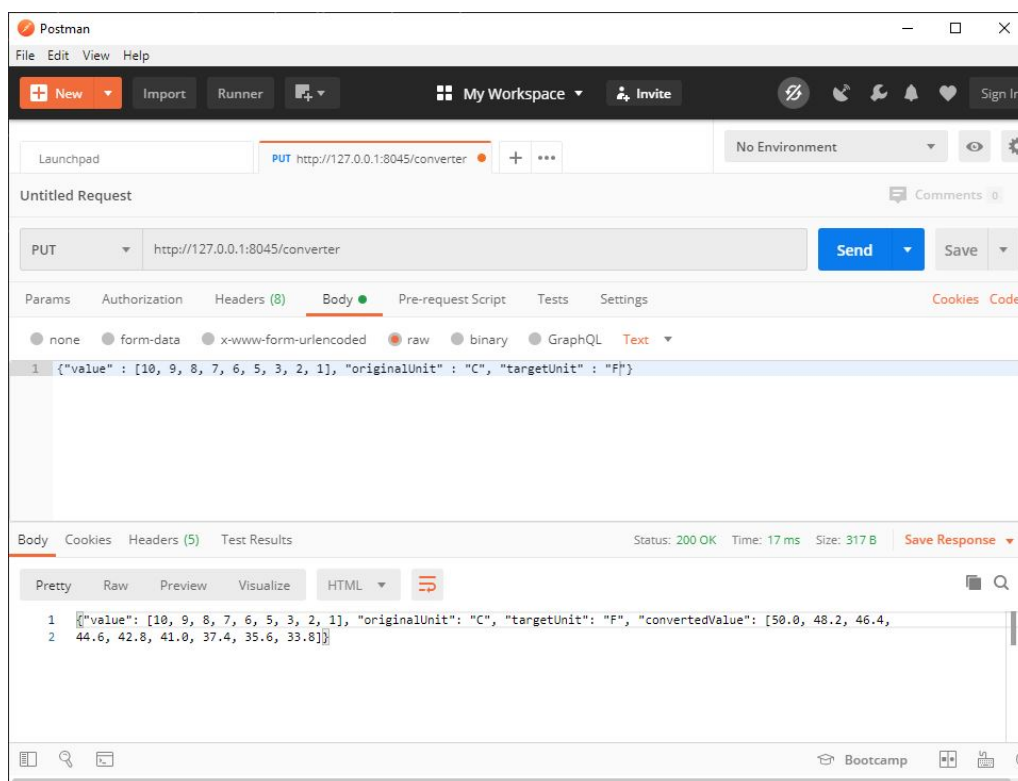


Figura 6: Esempio di conversione da Celsius a Fahrenheit

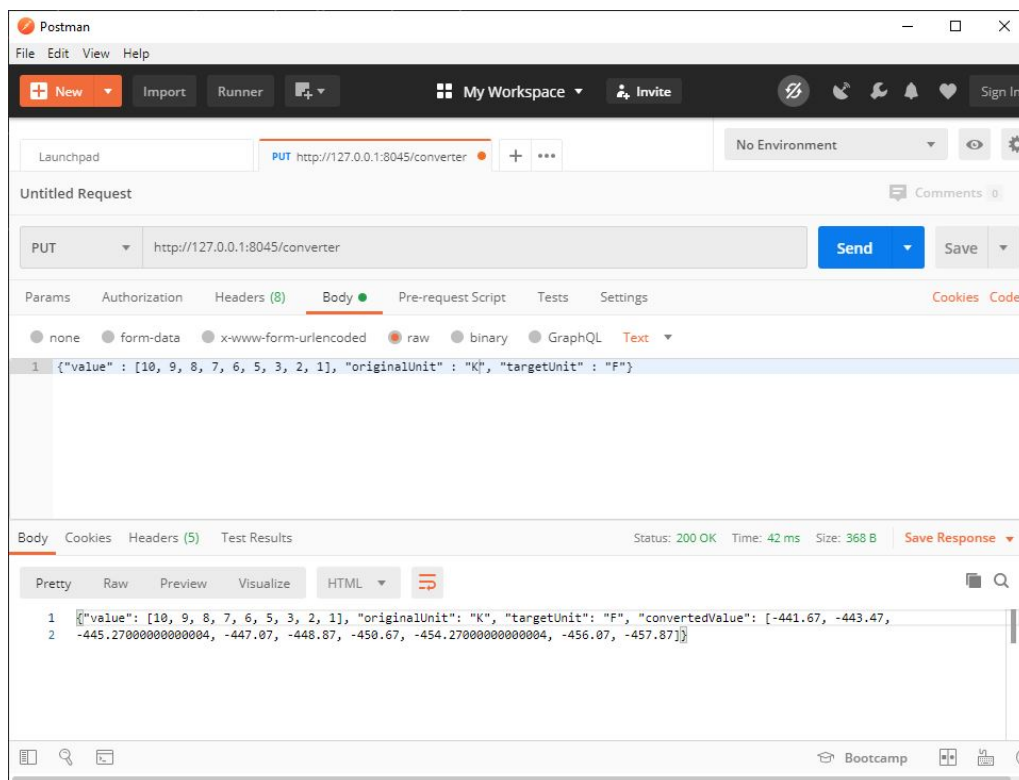


Figura 7: Esempio di conversione da Kelvin a Fahrenheit

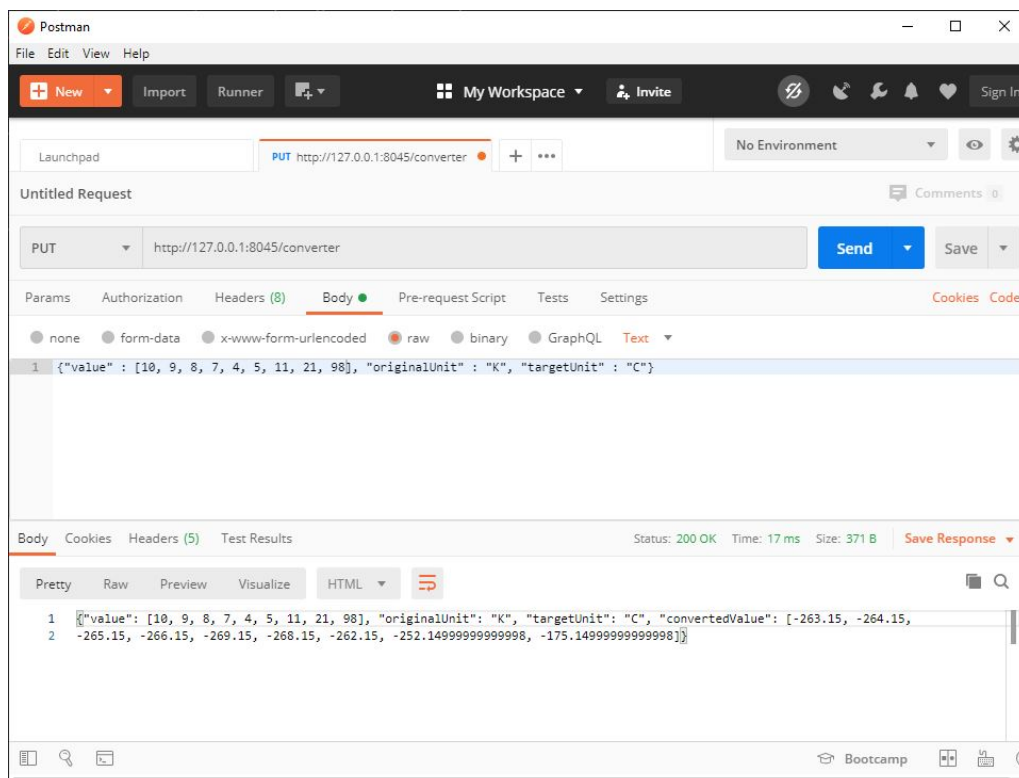


Figura 8: Esempio di conversione da Kelvin a Celsius

4.4 Exercise 4

Spiegazione

In questo esercizio veniva richiesto di sviluppare un servizio rest che implementasse una Freeboard con cherryypy utilizzando i metodi GET e PUT.

Risultati

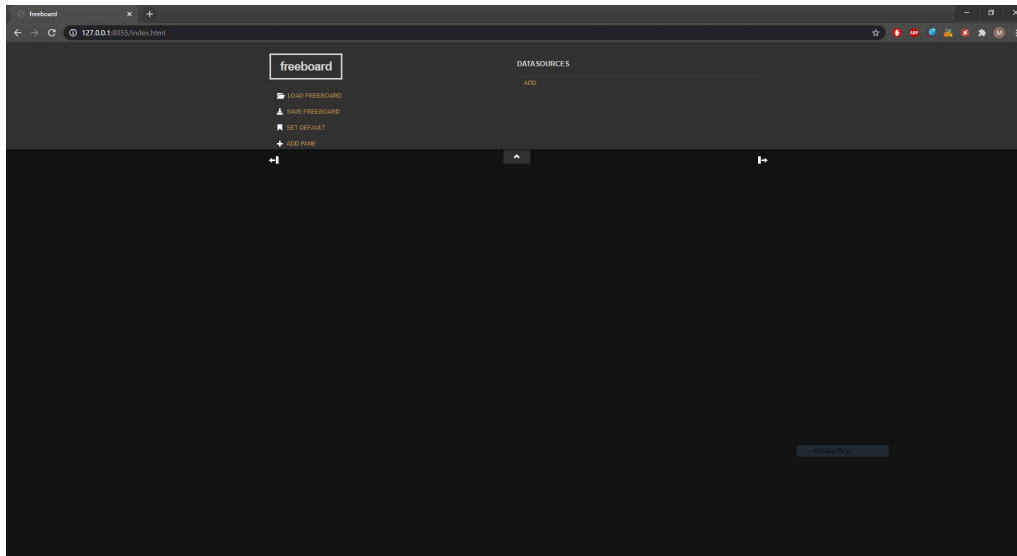


Figura 9: Free Board

5 SW Parte 2: Lab Software Part 2

5.1 Exercise 1

In questo esercizio veniva richiesto di sviluppare un Catalog in stile Rest.

Al fine di implementare le funzionalità richieste sono state sviluppate tre differenti classi (**Device()**, **Service()** e **User()**) contenenti tutte i metodi GET e POST.

Il metodo GET della classe **Device()** contiene la funzione `returnDevice()` la quale ha il compito di ritornare tutte le caratteristiche dei dispositivi registrati nel catalog oppure se specificato solo le singole caratteristiche di un dispositivo.

Il metodo PUT della classe **Device()** contiene invece la funzione `addDevice()` la quale ha il compito di registrare i nuovi dispositivi. Un singolo dispositivo viene registrato specificandone le seguenti informazioni: ID, endPoint e availableResources.

Analogamente il metodo PUT della classe **User()** contenente la funzione `addUser()` registra un nuovo utente tramite le seguenti informazioni: ID, name, surname. Anche in questo caso il metodo GET della classe ritorna tutti gli utenti registrati al catalog oppure se specificato tramite ID le informazioni del singolo utente.

Infine la classe **Service()** contenente le funzioni `addService()` e `returnService()` aggiunge e ritorna i servizi registrati nel catalog.

Tutte le funzioni richiamate dei metodi GET e PUT (contenuti nelle classi sopra descritte) sono state definite all'interno di una classe chiamata **WorkCatalog()**. Inoltre tutte le informazioni all'interno del catalog sono scambiate tramite file JSON.

5.2 Exercise 2

Spiegazione

In questo esercizio veniva richiesto di sviluppare un Client che richiamasse i diversi servizi forniti dal catalog, lo scopo era quello di testarne il funzionamento.

Sono state quindi sviluppate tre differenti funzioni `getBroker()`, `getDevice()`, `getUser()` le quali hanno il compito di raccogliere informazioni rispettivamente riguardanti il broker i device gli utenti registrati all'interno del catalog.

5.3 Exercise 3

Spiegazione

In questo caso veniva richiesto di creare un applicazione client che registrasse periodicamente ogni tot tempo, un nuovo device IoT nel catalog. Tramite l'utilizzo dei thread la funzione `sendData()` registra periodicamente un dispositivo nel catalog.

5.4 Exercise 4

Spiegazione

In questo esercizio veniva richiesto di estendere la funzionalità del temperature Web Service, la quale periodicamente registra o aggiorna le differenti informazioni. Tramite la funzione `postRequest()` vengono registrate le nuove informazioni all'interno del Catalog.

5.5 Exercise 5

Spiegazione

In questo esercizio veniva richiesto di estendere le funzionalità del catalog per lavorare come mqtt subscriber, andando quindi sottoscrivere ad un determinato topic per poi registrare un nuovo device oppure aggiornare una vecchia registrazione. L'informazione riguardante il nuovo device viene ricevuta in formato JSON dalla

funzione `myOnMessageReceived()`, contenuta nella classe `simpleSubscriber()`, la quale ha il compito di salvare le informazioni ricevute nelle corrette istanze.

5.6 Exercise 6

Spiegazione

Infine questo esercizio richiedeva di sviluppare un mqtt publisher che si occupasse di registrare periodicamente un nuovo device IoT.

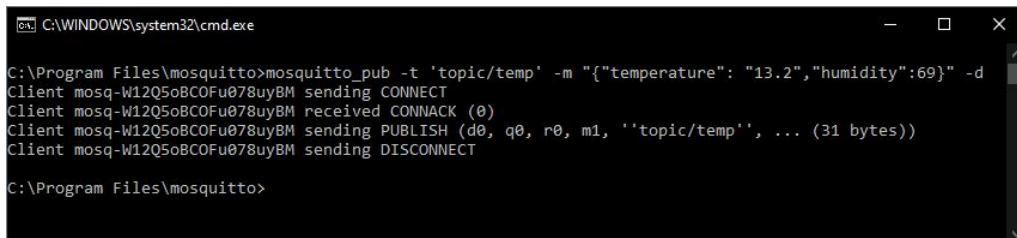
Il tutto è stato implementato usando il metodo `myPublish()` contenuto nella classe `simplePublisher()`. In questo metodo deve essere specificato il topic sul quale si vuole pubblicare e che tipo di messaggio si vuole pubblicare (ex. `test.myPublish('/deviceMQTT', message)`).

6 SW Parte 3: Lab Software Part 3

6.1 Exercise 1

Spiegazione

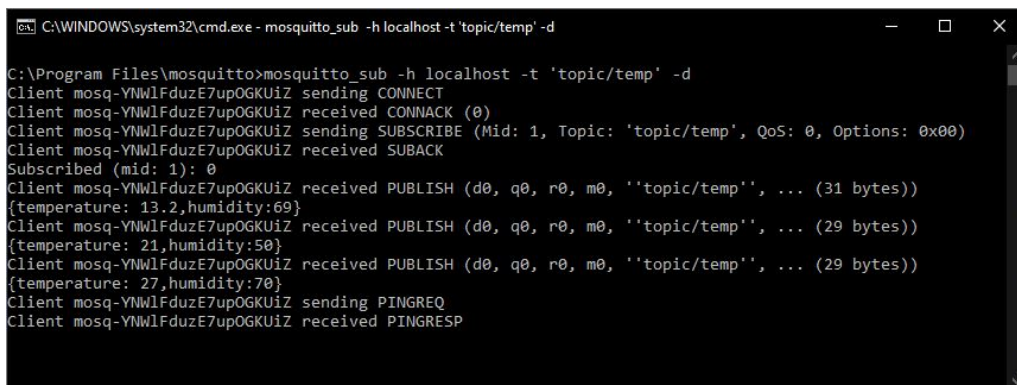
L'esercizio richiedeva di verificare il funzionamento delle funzioni `mosquitto_pub` e `mosquitto_sub`.



```
C:\WINDOWS\system32\cmd.exe
C:\Program Files\mosquitto>mosquitto_pub -t 'topic/temp' -m '{"temperature": "13.2","humidity":69}" -d
Client mosq-W12Q5oBCOFu078uyBM sending CONNECT
Client mosq-W12Q5oBCOFu078uyBM received CONNACK (0)
Client mosq-W12Q5oBCOFu078uyBM sending PUBLISH (d0, q0, r0, m1, 'topic/temp', ... (31 bytes))
Client mosq-W12Q5oBCOFu078uyBM sending DISCONNECT
C:\Program Files\mosquitto>
```

Figura 10: Publisher

Nell'esempio e' stato pubblicato un file json al topic: *topic/temp*.



```
C:\WINDOWS\system32\cmd.exe - mosquitto_sub -h localhost -t 'topic/temp' -d
C:\Program Files\mosquitto>mosquitto_sub -h localhost -t 'topic/temp' -d
Client mosq-YNW1FduzE7upOGKUiZ sending CONNECT
Client mosq-YNW1FduzE7upOGKUiZ received CONNACK (0)
Client mosq-YNW1FduzE7upOGKUiZ sending SUBSCRIBE (Mid: 1, Topic: 'topic/temp', QoS: 0, Options: 0x00)
Client mosq-YNW1FduzE7upOGKUiZ received SUBACK
Subscribed (mid: 1): 0
Client mosq-YNW1FduzE7upOGKUiZ received PUBLISH (d0, q0, r0, m0, 'topic/temp', ... (31 bytes))
{temperature: 13.2,humidity:69}
Client mosq-YNW1FduzE7upOGKUiZ received PUBLISH (d0, q0, r0, m0, 'topic/temp', ... (29 bytes))
{temperature: 21,humidity:50}
Client mosq-YNW1FduzE7upOGKUiZ received PUBLISH (d0, q0, r0, m0, 'topic/temp', ... (29 bytes))
{temperature: 27,humidity:70}
Client mosq-YNW1FduzE7upOGKUiZ sending PINGREQ
Client mosq-YNW1FduzE7upOGKUiZ received PINGRESP
```

Figura 11: Subscriber - il subscriber si è sottoscritto al topic: *topic/temp*

```

C:\WINDOWS\system32\cmd.exe - mosquitto_sub -h localhost -t 'topic/+ -d

C:\Program Files\mosquitto>mosquitto_sub -h localhost -t 'topic/+ -d
Client mosq-eQo3GUeyfBFadLEJeg sending CONNECT
Client mosq-eQo3GUeyfBFadLEJeg received CONNACK (0)
Client mosq-eQo3GUeyfBFadLEJeg sending SUBSCRIBE (Mid: 1, Topic: 'topic/+', QoS: 0, Options: 0x00)
Client mosq-eQo3GUeyfBFadLEJeg received SUBACK
Subscribed (mid: 1): 0
Client mosq-eQo3GUeyfBFadLEJeg received PUBLISH (d0, q0, r0, m0, ''topic/temp'', ... (29 bytes))
{temperature: 27,humidity:70}
Client mosq-eQo3GUeyfBFadLEJeg received PUBLISH (d0, q0, r0, m0, ''topic/temp'', ... (29 bytes))
{temperature: 40,humidity:36}

```

Figura 12: Subscriber - utilizzata la wildcards +

```

C:\WINDOWS\system32\cmd.exe - mosquitto_sub -h localhost -t 'topic/# -d

^C
C:\Program Files\mosquitto>mosquitto_sub -h localhost -t 'topic/# -d
Client mosq-HdHGJXCGaktJ26kySj sending CONNECT
Client mosq-HdHGJXCGaktJ26kySj received CONNACK (0)
Client mosq-HdHGJXCGaktJ26kySj sending SUBSCRIBE (Mid: 1, Topic: 'topic/#', QoS: 0, Options: 0x00)
Client mosq-HdHGJXCGaktJ26kySj received SUBACK
Subscribed (mid: 1): 0
Client mosq-HdHGJXCGaktJ26kySj received PUBLISH (d0, q0, r0, m0, ''topic/temp'', ... (29 bytes))
{temperature: 40,humidity:36}
Client mosq-HdHGJXCGaktJ26kySj received PUBLISH (d0, q0, r0, m0, ''topic/temp'', ... (32 bytes))
{temperature: 30 humidity:24} -d

```

Figura 13: Subscriber - utilizzata la wildcards #

6.2 Exercise 2

Spiegazione

In questo esercizio veniva richiesto di sviluppare un mqtt subscriber il quale riceveva la temperatura inviata da Arduino. Quest'ultima viene inviata dalla funzione `mqtt.publish()`, in formato JSON, codificata usando la funzione `SenMLEncode`.

All'avvio il servizio mqtt subscriber deve invocare i servizi web forniti dal catalog, deve quindi registrarsi come nuovo servizio, richiedere informazioni sul message broker e richiedere informazioni sull'end-point usato dall' Arduino Yun per comunicare.

L'mqtt subscriber è stato implementato usando la funzione `myOnMessageReceived()`, contenuta all'interno della classe **MySubscriber()**, la quale si occupa di ricevere i dati di temperatura inviati dall' Arduino informato Json e decodificarli.

Nel main del file Python tramite la funzione `broker=requests.get('http://localhost:8080/broker/')`, vengono ricavate le informazioni riguardanti il message broker. Inoltre sempre all'interno del main vengono ricavate le informazioni dell'ID della Yun.

Al fine di lavorare correttamente con python, sono stati modificati i file di configurazione dell'Arduino Yun, inoltre non è possibile fare lavorare insieme due file (uno python e uno sketch di arduino), se non sono stati sviluppati apposta per comunicare l'uno con l'altro.

6.3 Exercise 3

Spiegazione

In questo esercizio veniva di nuovo richiesto di far collaborare mqtt publisher sviluppato con python con l'arduino. Nello specifico veniva richiesto di sviluppare un mqtt publisher il quale inviava comandi di attuazione che indicavano l'accensione o lo spegnimento del LED controllato da Arduino yun.

Utilizzando questa funzione: `mqtt.subscribe("tiot/17/led",setLedValue)`; l'arduino si sottoscrive topic per ricevere informazioni sulle operazioni da fare sul LED, tramite la funzione `setLedValue()` decodifico l'informazione ricevuta dal publisher, in formato SenML e in base a quest'ultima cambio lo stato del LED.

Il publisher è stato sviluppato in Python quest'ultimo pubblica periodicamente un messaggio in formato SenML, utilizzando la classe `MyPublisher()`.

6.4 Exercise 4

Spiegazione

In questo esercizio veniva richiesto di fare una versione remota della smart home sviluppata nei laboratori precedenti.

Questo Remote Smart home controller è stato implementato utilizzando la libreria `mqtt` e lavorando sia come publisher che come subscriber.

Come prima operazione il servizio si registra al Catalog utilizzando la funzione `sendData()`, dopodiché richiede le informazioni riguardanti il broker e l'end-point usato della Yun per comunicare, queste operazioni sono svolte nel main del file `Smart House Controller.py`, usando la funzione `request.get()`.

Infine vengono pubblicati sui differenti topic le varie operazioni da svolgere sui differenti sensori.

Utilizzando la funzione `myPublish` dichiarata nella classe **Publisher()**, il main invia informazioni sul settaggio dei sensori.

- `"/tiot/17/house/control/light"` : pubblicando sul seguente topic vado a impostare lo stato del led presente sulla scheda.
- `"/tiot/17/house/control/screen"`: pubblicando su questo topic, mando il testo che deve essere scritto sul display LCD attaccato all'arduino.
- `"/tiot/17/house/control/fan"`: pubblicando su questo topic vado a modificare la velocità della ventola attaccata all'arduino.
- `"/tiot/17/house/control/heat"`: pubblicando su questo topic invece vado a modificare l'intensità luminosa del led collegato ad Arduino.

L'arduino si sottoscrive a tutti i topic elencati sopra tramite la funzione:

`mqtt.subscribe("tiot/17/house/control/#",setValue)`.

Oltre a ricevere informazioni, l'arduino lavora anche come publisher, mandando informazioni relative alla temperatura, alla presenza e al rumore. Partendo dalle funzioni sviluppate nel lab precedente i risultati ottenuti dai sensori vengono presi e pubblicati su differenti topic:

- **`mqtt.publish(F("tiot/17/house/sens/temperature"),message)`**; utilizzando la seguente funzione riportata nel void loop() di arduino, viene pubblicata la temperatura rilevata dal sensore dalla corrispondente funzione (`ReadTemperature()`) e pubblicata sul topic specificato. L'informazione pubblicata viene quindi ricevuta dal subscriber, scritto in python utilizzando la classe **Subscriber()**.
- Analogamente a come descritto prima, pubblico i dati ricavati con arduino sui rispettivi topic.
`mqtt.publish(F("tiot/17/house/sens/presence"),message)`; con questo comando pubblico i dati riguardanti le presenze, mentre su quest'altro topic:
`mqtt.publish(F("tiot/17/house/sens/noise"),message)`; pubblico i dati riguardanti il sensore di rumore. Il tutto viene ricevuto dal subscriber implementato in python.

Tutte le funzioni riguardanti le misure dei sensori(ex. temperatura, rumore, presenza ...) sono state riciclate dall'esercizio 2.1 del Lab HW parte 2.

Vogliamo solo far notare che il programma funziona molto lentamente a causa della scarsa capacità computazionale di arduino.

Sviluppando questo esercizio si è potuta apprezzare la differenza di implementare una smart home controller gestita solo da arduino oppure come servizio remoto. Lo sviluppo della smart home come servizio remoto porta molti vantaggi sia a livello computazionale che di manutenzione. Infatti in questo caso l'arduino non doveva preoccuparsi della computazione e dell'elaborazione dei dati, in quanto quest'ultima veniva svolta dal server online. Questa implementazione permette di alleggerire il carico di lavoro e la memoria occupata dell'arduino. Inoltre avendo un servizio online e quindi essendo svincolato dalla capacità di calcolo dell'arduino ho la possibilità di svolgere operazioni più complesse. Sviluppando invece lo smart home controller in versione locale sono costretto a più vincoli, in quanto l'arduino si deve occupare di gestire tutti i servizi e tutte le computazioni, andando così a rallentarlo e ad aumentare la memoria interna occupata. Inoltre questa tipologia di implementazione permette poca versatilità, perchè nel caso in cui si volesse aggiungere una nuova funzione di misurazione oppure aggiungere un qualche tipo di computazione, bisognerebbe andare a lavorare sul codice del singolo arduino tenendo conto anche della potenza di calcolo di quest'ultimo.

7 SW Parte 4: Lab Software Part 4

Spiegazione

In quest'ultimo laboratorio si è deciso di sviluppare un applicazione che controllare la smart home, sviluppata nel Lab SW parte 3 esercizio 4. L'applicazione è stata sviluppata usando la libreria di python Kivy, è stata scelta questa libreria in quanto quest'ultima permette una diffusione multiplatforma del programma implementato. Quest'applicazione scritta in python può essere trasformata in un file .apk, .exe, in modo che possa lavorare su diversi sistemi operativi.

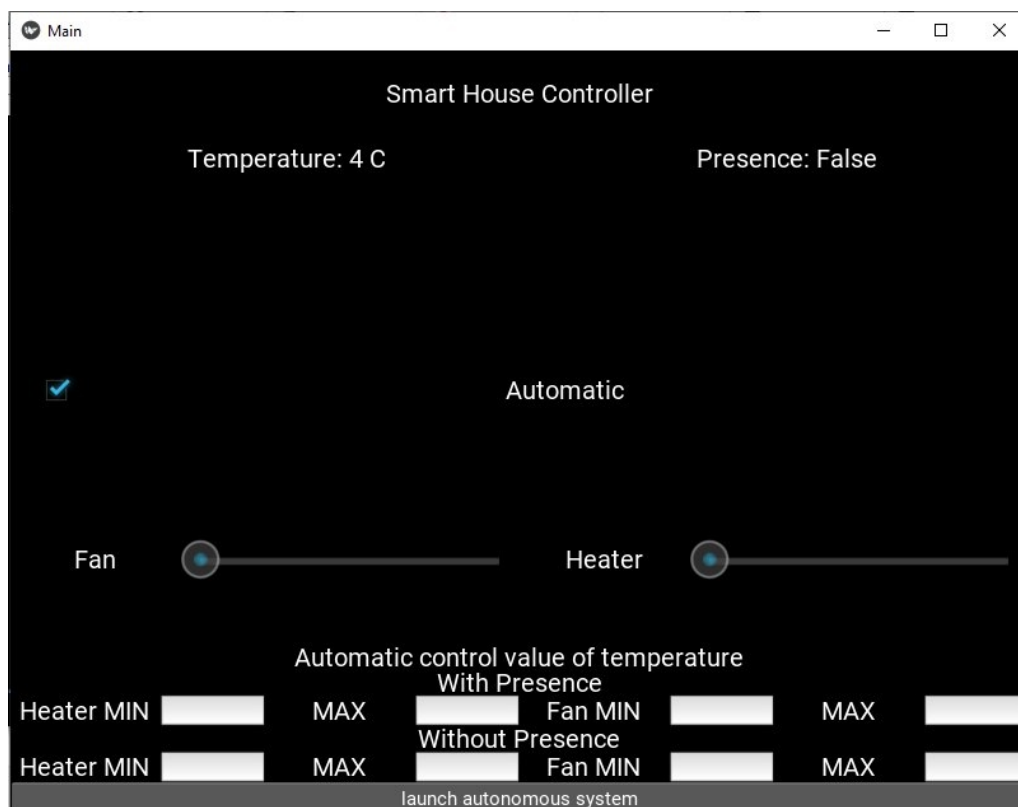


Figura 14: Applicazione

La figura rappresenta la schermata dell'applicazione. Come si può notare al centro della figura in alto vi è il titolo dell'applicazione, subito sotto possiamo notare che sono mostrati i valori della temperatura e della presenza (la variabile presence è stata definita come un booleano e assume il valore True quando viene rilevata la presenza di una persona).

Al fondo della schermata invece si possono settare i diversi set-point:

- Heater MIN → questo parametro indica il valore minimo di temperatura tale per cui viene acceso il led, analogamente il parametro Heater MAX corrisponde al valore massimo di temperatura oltre il quale il led (che emula un "caminetto") viene spento. Al fine di aumentare la versatilità del sistema si è deciso di dare la possibilità all'utente di impostare set-point diversi in base alla presenza o meno di persone nella stanza.
- Fan MIN → impostando questo parametro vado ad definire la temperatura minima alla quale si ha l'accensione della ventola, si è definito un valore massimo di temperatura oltre il quale la ventola viene spenta. Anche in questo caso ho la possibilità di impostare set-point diversi in base al valore della variabile Presence.

Infine al centro possiamo notare la presenza di due slider, attraverso i quali vado ad aumentare o a diminuire nel caso del led l'intensità, mentre nel caso della ventola la velocità di rotazione.

Questa applicazione permette un controllo molto più facile e smart del sistema smart home, permettendo di settare in qualunque momento i valori di set-point ideali.

Inoltre è stato sviluppato, un ulteriore servizio chiamato serra, il quale si occupa della gestione e del controllo delle piante questo servizio si registra sul catalog, dal quale prende le informazioni del dispositivo IoT che invia i dati. Questo servizio si occupa di leggere la temperatura e l'umidità del terreno e di conseguenza va a regolare il riscaldamento e la luminosità a cui è esposta la pianta e inoltre in caso necessario innaffiarla. La funzione che si occupa di questi controlli è la funzione `compute()`, la quale in base ai valori settati modifica l'ambiente in cui la pianta si trova. Tutti questi dati vengono scambiati tramite publisher e subscriber.