

# **Tecnologie per IoT**

## **LABORATORI**

---

**Docenti:**

**Patti Edoardo**

**Jahier Pagliari Daniele**

**Poncino Massimo**

---

**25/07/2020**

**Gruppo 7**

---

**Vitanza Francesco - S249288**

**Sinigaglia Giovanni - S248223**

**Vitrani Stefano - S247165**



**POLITECNICO  
DI TORINO**

# LABORATORIO 1 HW

Brevi sketch Arduino

## MATERIALE UTILIZZATO

- Arduino YUN rev 2
- Breadboard
- Cavi per ponticelli tra breadboard, sensori e l'MCU
- LED rosso e verde
- Resistenze da 220 Ohm
- Display LCD Formato 16x2 con tecnologia I2C DFR0063
- Motore DC + Fan DFR0063
- Sensore di temperatura GRV Temp Grove NCP18WF104 con tolleranza di 1.5°C
- Sensore di movimento (PIR) SEN0018

## REALIZZAZIONE.

1. **Scopo:** alternare il led verde e il led rosso tra stato on e off con due semiperiodi differenti (rispettivamente 7 e 3).

La gestione è avvenuta con l'utilizzo della libreria *TimerOne.h*: è stato utilizzato *Timer1* per il led verde. Tramite la funzione *Timer1.attachInterrupt(greenLedOn)* una volta scaduto il tempo impostato durante *Timer1.Initialize(greenLedPeriod \* 1e06)* viene scatenato l'interrupt che richiama la funzione *greenLedOn* che cambia lo stato (on/off) del led verde. Il led rosso gestito nel loop tramite *delay*.

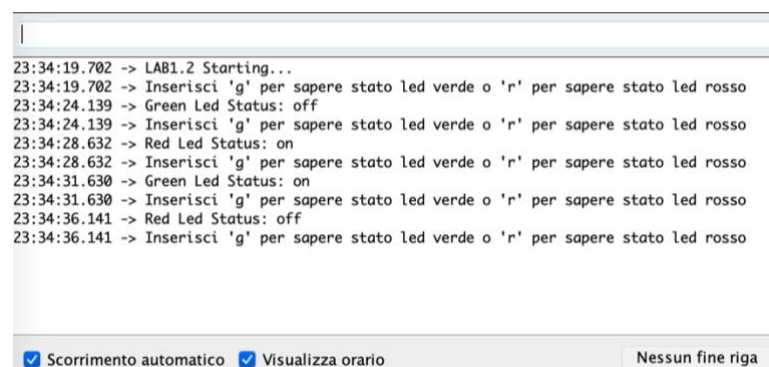
NB: gli interrupt scavalcano i delay quando scatenati.

2. **Scopo:** Ottenere da sketch precedente lo stato dei led tramite seriale.

Durante il setup viene richiesto di inserire "r" o "g", tramite seriale, per conoscere lo stato rispettivamente del led rosso e verde. Se l'utente ha scritto un carattere viene richiamata la funzione di stampa dello status *printStatus()*. Se il carattere non è "r" o "g" viene stampato "carattere non valido".

Vengono usate variabili di tipo *volatile* perché, quando si hanno accessi

concorrenti da ISR, la risorsa condivisa va sempre letta da RAM e non dalla cache della CPU.

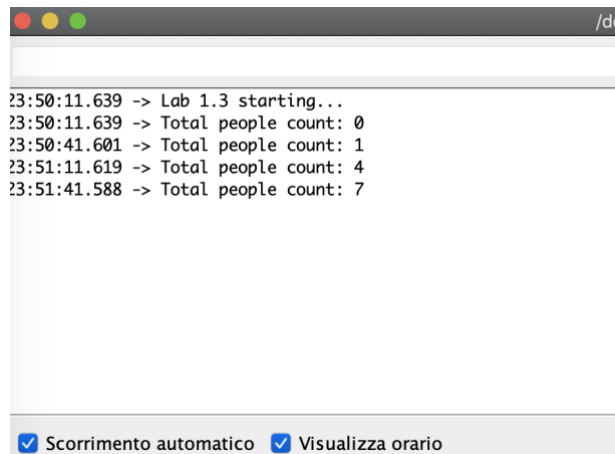


```
23:34:19.702 -> LAB1.2 Starting...
23:34:19.702 -> Inserisci 'g' per sapere stato led verde o 'r' per sapere stato led rosso
23:34:24.139 -> Green Led Status: off
23:34:24.139 -> Inserisci 'g' per sapere stato led verde o 'r' per sapere stato led rosso
23:34:28.632 -> Red Led Status: on
23:34:28.632 -> Inserisci 'g' per sapere stato led verde o 'r' per sapere stato led rosso
23:34:31.630 -> Green Led Status: on
23:34:31.630 -> Inserisci 'g' per sapere stato led verde o 'r' per sapere stato led rosso
23:34:36.141 -> Red Led Status: off
23:34:36.141 -> Inserisci 'g' per sapere stato led verde o 'r' per sapere stato led rosso
```

### 3. Scopo: Identificazione presenza persone tramite sensore di movimento PIR.

Dalla lettura del datasheet del sensore PIR si può constatare la presenza di due “viti” che permettono la modifica di alcune specifiche del sensore tra cui: ogni quanto si può rilevare la presenza e la sensibilità del sensore. Utile settare tali parametri in modo che: si rilevi il più velocemente possibile la presenza (5 secondi il minimo per il sensore fornito) e che la sensibilità sia la più alta possibile.

Il sensore viene gestito tramite interrupt. Nel codice presente è stato scelto come pin di gestione interrupt il pin 7. L’ISR è *count* che incrementa il contatore di conteggio presenza persone e accende il led. Nella funzione loop viene effettuata la stampa del count aggiornato con un delay di 30 secondi.



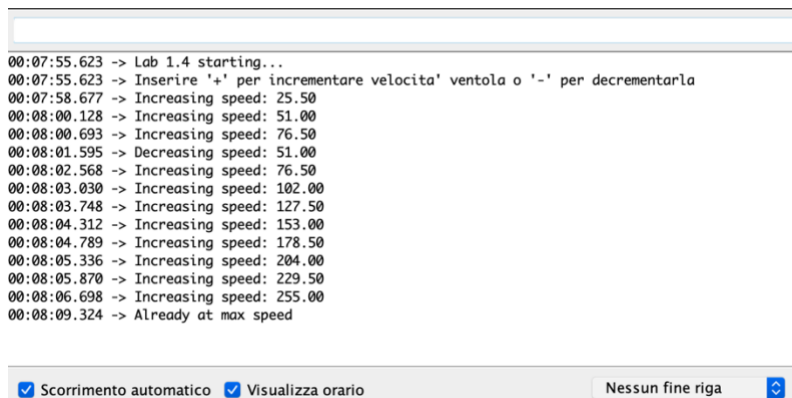
```
23:50:11.639 -> Lab 1.3 starting...
23:50:11.639 -> Total people count: 0
23:50:41.601 -> Total people count: 1
23:51:11.619 -> Total people count: 4
23:51:41.588 -> Total people count: 7
```

☒ Scorrimento automatico ☒ Visualizza orario

### 4. Scopo: Controllo ventola (motore) PWM

I pin che dispongono di segnali PWM sono segnati sull’ Arduino da una tilde (~ ad esempio pin 6).

Nel loop viene richiesto il carattere all’utente tramite seriale. Se non è ne “+” ne “-” viene restituito comando non valido. Se viene selezionato uno dei due caratteri vengono richiamate rispettivamente le due funzioni *increaseSpeed()* e *decreaseSpeed()*.



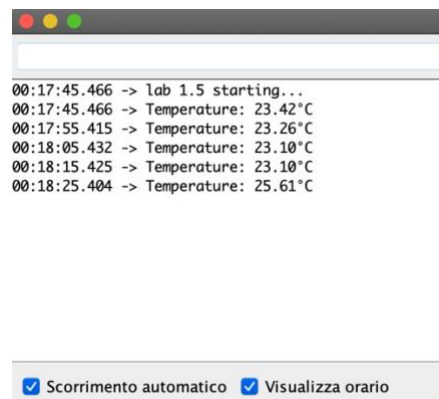
```
00:07:55.623 -> Lab 1.4 starting...
00:07:55.623 -> Inserire '+' per incrementare velocita' ventola o '-' per decrementarla
00:07:58.677 -> Increasing speed: 25.50
00:08:00.128 -> Increasing speed: 51.00
00:08:00.693 -> Increasing speed: 76.50
00:08:01.595 -> Decreasing speed: 51.00
00:08:02.568 -> Increasing speed: 76.50
00:08:03.030 -> Increasing speed: 102.00
00:08:03.748 -> Increasing speed: 127.50
00:08:04.312 -> Increasing speed: 153.00
00:08:04.789 -> Increasing speed: 178.50
00:08:05.336 -> Increasing speed: 204.00
00:08:05.870 -> Increasing speed: 229.50
00:08:06.698 -> Increasing speed: 255.00
00:08:09.324 -> Already at max speed
```

☒ Scorrimento automatico ☒ Visualizza orario Nessun fine riga

Tali funzioni incrementano/decrementano una costante che identifica la velocità (limitata tramite la funzione constrain tra 0 e 255) e scrive tale valore sul pin di controllo del motore tramite funzione *AnalogWrite()*. Dopo di che viene Stampato su seriale quale è stata la funzione effettuata

### 5. Scopo: Lettura di valori di temperatura (analogici)

Questa volta, avendo a che fare con un sensore analogico è necessario associarlo ad un pin analogico (ad esempio A1). La lettura viene effettuata ogni 10 secondi tramite la funzione *analogRead()* e il valore viene visualizzato tramite seriale (entrambe le operazioni effettuate nella loop function). Il valore analogico letto viene convertito tramite la funzione *convert()*.



```
00:17:45.466 -> lab 1.5 starting...
00:17:45.466 -> Temperature: 23.42°C
00:17:55.415 -> Temperature: 23.26°C
00:18:05.432 -> Temperature: 23.10°C
00:18:15.425 -> Temperature: 23.10°C
00:18:25.404 -> Temperature: 25.61°C
```

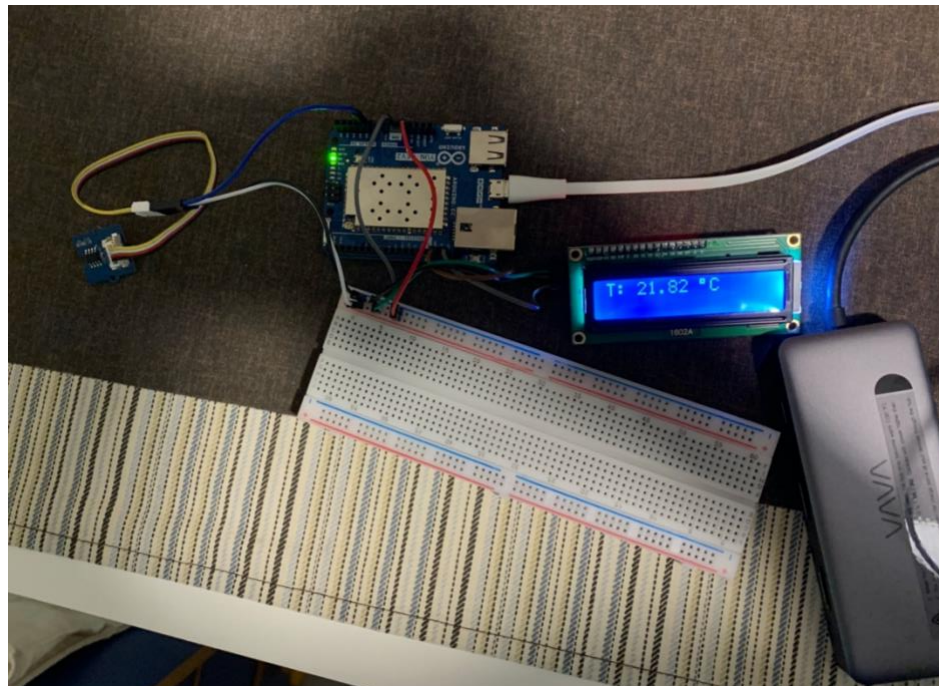
☒ Scorrimento automatico ☒ Visualizza orario

## 6. Scopo: Utilizzo del display LCD tramite protocollo LCD

Per questo esercizio è stato utilizzato lo sketch precedente. La stampa viene effettuata direttamente nella loop function e vengono utilizzate varie funzioni di inizializzazione dello schermo in setup per settare aspetti come: retroilluminazione, pulizia da scritte, primi caratteri stampati. La stampa viene fatta spostando il cursore tramite la funzione `lcd.setCursor(y,x)` dove “x” rappresenta la posizione del carattere di riga mentre “y” la posizione di colonna. Tramite la funzione `lcd.Print()` viene stampata la stringa desiderata (passata come parametro) dove è stato impostato il cursore o dove è rimasto l’ultima volta che è stato stampato qualcosa.

```
void setup()
{
  //Codice di inizializzazione var
  //inizializzazione schermo
  lcd.begin(16, 2); //caratteri visibili sullo schermo
  lcd.setBacklight(255); //accensione retroilluminazione
  lcd.home();
  lcd.clear();
  lcd.print("T: "); //scrittura in posizione (0,0) -> lcd.setCursor(0,0);
}

void loop() {
  //stampa tramite schermo lcd
  lcd.setCursor(3, 0); //sposto cursore direttamente su posizione riga 0 e colonna 3
  float temp = analogRead(TEMP_PIN); //lettura temperatura
  float celsius = convert(temp); //conversione in celsius temperatura
  lcd.print(celsius); lcd.print(" "); lcd.print((char)0xDF); lcd.print("C"); //0xDF = °
  delay(10*1e3);
}
```





## LABORATORIO 2 HW

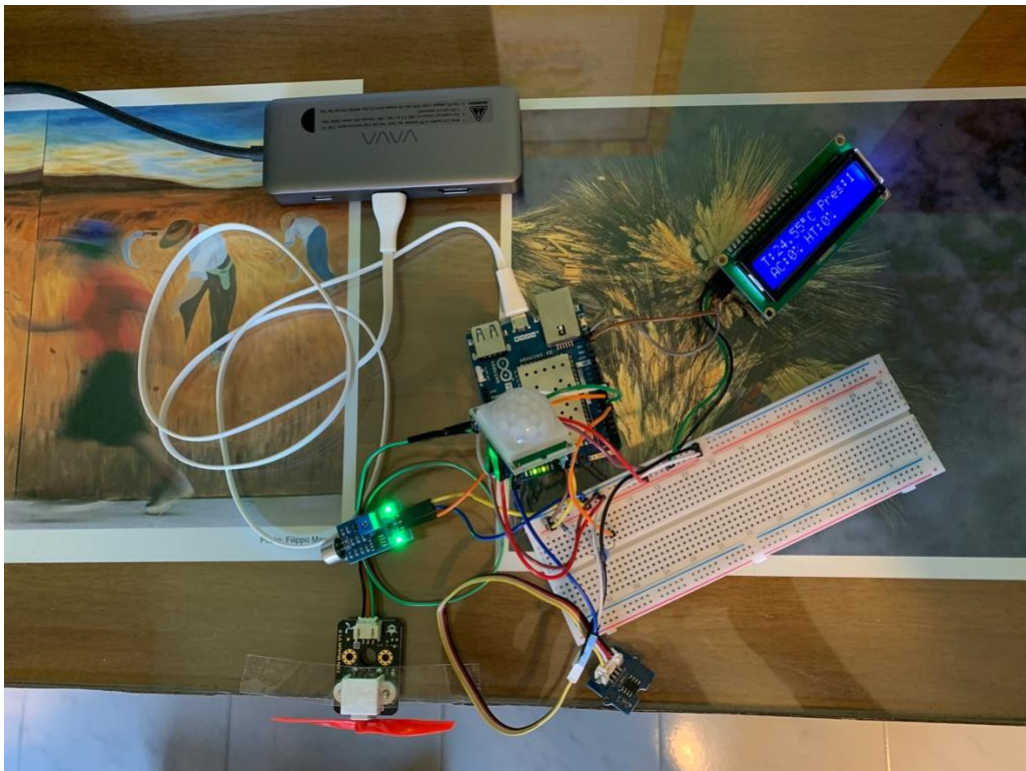
SMART HOME CONTROLLER (Versione locale)

### MATERIALE UTILIZZATO.

- Arduino YUN rev 2
- Breadboard
- Cavi per ponticelli tra breadboard, sensori e l'MCU
- LED rosso e verde
- Resistenze da 220 Ohm
- Display LCD Formato 16x2 con tecnologia I2C DFR0063
- Motore DC + Fan DFR0063
- Sensore di temperatura GRV Temp Grove NCP18WF104 con tolleranza di 1.5°C
- Sensore di movimento (PIR) SEN0018
- Sensore di rumore Grove – Sound Sensor LM386

### SCOPO.

Realizzazione di un controllore, per casa smart, per monitoraggio, con vari sensori, e modifica, tramite condizionamento e riscaldamento dell'aria, delle caratteristiche di condizioni in una casa, tramite attuatori.



### PROBLEMI RISCONTRATI.

- Limite di pin abilitati a utilizzo interrupt;
- Pin SCA, SDL (I2C), necessari per funzionamento monitor, legati a PIN 2,3 che quindi non possono essere utilizzati quando è presente uno schermo.

## SVILUPPO E DESCRIZIONE DEL PROGETTO.

### PUNTO 1 e 2: SISTEMA DI RISCALDAMENTO E CONDIZIONAMENTO

Il primo punto cruciale nella gestione di una casa “smart” è il condizionamento dell’aria e riscaldamento. Per questo scopo sono stati emulati questi attuatori tramite un led Rosso (*Riscaldamento*) in modalità PWM che, tramite la variazione dell’intensità luminosa indica quanto il sistema sta riscaldando.

Il condizionatore è stato emulato usato un motore DC + Fan che sempre tramite segnale PWM regola la velocità del motore. Esistono quattro set point: temperatura

minima affinché si avvii il condizionamento e temperatura massima alla quale il condizionamento è massimo. Similmente e simmetricamente è stato fatto per il riscaldamento: la temperatura più alta corrisponde al punto di accensione del riscaldamento, mentre la temperatura più bassa corrisponde al riscaldamento massimo.

Entrambe le operazioni sono state gestite tramite funzioni (heat e fresh). All’ interno sono stati mappati valori tra 0 e 255 utilizzando la funzione *map* per impostare il segnale PWM.

```
//FUNZIONE PER GESTIONE DI CONDIZIONAMENTO DELL'ARIA
//-----

int fresh(float temp) {
    float vel = map(temp, min_temp_ac, max_temp_ac, 0, 255); //rimappa
    vel = constrain(vel, 0, 255);
    // Serial.print("Vel Fresh:"); Serial.print(vel); //debug
    analogWrite(MOTOR_PIN, int(vel));
    return vel;
}

//FUNZIONE PER GESTIONE DI CONDIZIONAMENTO DEL RISCALDAMENTO
//-----

int heat(float temp) {
    float th = map(temp, min_temp_th, max_temp_th, 255, 0);
    th = constrain(th, 0, 255);
    // Serial.print("th Fresh:"); Serial.print(th); //debug

    analogWrite(LED_PIN, int(th));
    return th;
}
```

### PUNTO 3: RILEVAMENTO DI MOVIMENTI TRAMITE SENSORE PIR

NOTA: A causa del numero limitato di pin che riescono a gestire gli interrupt abbiamo deciso di assegnare il segnale di controllo del sensore al pin 8 che però non riesce a gestire l’interrupt.

Si è utilizzato un metodo polling nel loop, considerando che il segnale HIGH del pir rimane attivo per almeno 3 secondi.

```
//GESTIONE SENSORE PIR SENZA INTERRUPT
int pir=digitalRead(PIR_PIN);
if(pir==1)
    movement_detected();
```

NOTA2: ts1 è variabile di temporizzazione comune sia al sensore pir che sound che indica l’ultimo rilevamento di suono o di movimento. Viene aggiornato con *millis()* e all’aggiornamento viene stampato sulla seriale “C’è qualcuno.” come debug. Questa variabile è utilizzata per scegliere i valori di setpoint in presenza o assenza in base al timeout scelto.

### PUNTO 4: RILEVAMENTO PRESENZA PERSONA TRAMITE SENSORE DI RUMORE

In questo caso abbiamo optato per l’utilizzo di interrupt. L’ISR legata all’interrupt si chiama *sound\_detected()*. Essa sfrutta ts1 e un’altra variabile di temporizzazione ts2 che verifica quanto tempo è passato dall’ultima presenza verificata (o con il PIR o con il sensore di rumore).

Se ts2 - ts1 supera il *sound\_interval*, come da specifiche impostato a 10 minuti allora vuol dire che non è confermata la presenza.

Se vengono rilevati almeno 50 suoni in 10 minuti, la presenza è confermata. Si è scelto di contare almeno 50 suoni per evitare di percepire il rumore della ventola o altri rumori di fondo. Oltre a questa accortezza, ovviamente il sensore va calibrato secondo le necessità e le condizioni ambientali.

### PUNTO 5 e 6: FUNZIONI *isSomeone()* e *isntSomeone()*

Funzione *isSomeone()* chiamata se viene confermata la presenza di qualcuno, secondo le specifiche, tramite sensore PIR o sensore di suono.

Funzione *isntSomeone()* chiamata se non è più presente nessuno. Inizialmente viene considerata questa condizione.

Nella loop function viene gestito il caso di nessuna presenza nell'abitazione richiamando la funzione *isntSomeone()* se, confrontando l'istante in cui viene fatta la verifica (variabile *ts3*) con la variabile *ts1*, aggiornata se rilevato suono o movimento, supera il *timeout\_sound* e il *timeout\_pir* che da specifiche sono rispettivamente di 60 minuti e 30 minuti.

*isSomeone()* setta una variabile booleana di indicazione della presenza (*someone = true*), stampa su seriale che è presente qualcuno e se non sono presenti valori personalizzati di set-point allora si usano i set-point di default chiamati *\_somebody*.

*isntSomeone()* dualmente setta la variabile booleana *someone=false*, stampa su seriale che non è presente nessuno e se non sono presenti valori personalizzati di set-point allora si usano i set-point di default chiamati *\_nobody*.

### PUNTO 7: STAMPA DI INFORMAZIONI RILEVANTI SU LCD

Lo schermo LCD serve a mostrare informazioni rilevanti su due schermate che si alternano ogni 5 secondi. Per fare questo, nella loop function, tramite l'utilizzo di variabili temporali *ts5* e *ts6* (*millis()*), viene alternata la stampa su lcd, effettuata con due funzioni differenti.

*StampaLCD()* stampa la schermata con le informazioni di temperatura, se è presente o meno qualcuno (variabile *someone=true*) e quali sono le percentuali di funzionamento dei sistemi di condizionamento e riscaldamento (ottenuti tramite funzioni *fresh()* e *heat()*).

*StmapaLCDmM()* invece stampa i set-point di temperatura impostati in quell'istante.

Video funzionamento [qui](#)

### PUNTO 8: AGGIORNAMENTO SET- POINT TRAMITE PORTA SERIALE

L'aggiornamento dei valori dei set point viene effettuata nella loop function per mezzo di semplici comandi di interfacciamento con porta seriale. Una volta inserito il valore modificato del set-point viene settata a true la variabile *personal* che identifica la presenza di valori personalizzati (ciò farà operare diversamente le funzioni *isSomeone()* e *isntSomeone()*). Tramite seriale/Schermo LCD l'utente può verificare l'effettivo cambiamento dei valori di set-point. Per tornare ai valori di default basta semplicemente seguire le istruzioni (digitare 'r').



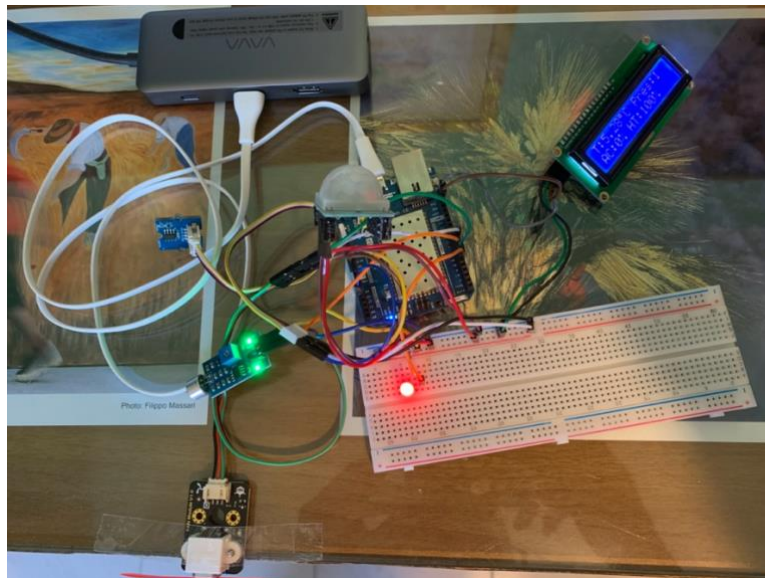
Video funzionamento [qui](#)

**PUNTO 9: RENDERE LED VERDE UNA LUCE SMART CHE SI ATTIVI AL DOPPIO BATTITO DI MANI**

Parallelamente al file relativo al lab2 esiste un file lab2bonus con all'interno il codice che implementa soltanto tale funzionalità.

Si è notato che statisticamente un battito di mani corrisponde a circa 25-30 eventi di suoni. Perciò il led cambia stato se e solo se vengono rilevati più di 25 eventi sonori.

Per testare il funzionamento del riscaldatore si è temporaneamente abbassata lato SW la temperatura di 20°C. Qui si può vedere il riscaldamento al massimo.





## LABORATORIO 3 HW

### COMUNICAZIONE TRAMITE INTERFACCE REST & MQTT

#### MATERIALE UTILIZZATO.

- Arduino YUN rev 2
- Breadboard
- Cavi per ponticelli tra breadboard, sensori e l'MCU
- LED rosso e/o verde + Resistenze da 220 Ohm
- Sensore di temperatura GRV Temp Grove NCP18WF104 con tolleranza di 1.5°C

#### REALIZZAZIONE.

Per questo laboratorio è stato necessario l'utilizzo della libreria *BRIDGE* che consente all'Arduino di poter far comunicare i due processori (ATmega32U4 e AR9331 che utilizza una versione linux). Allo stesso modo vengono utilizzate altre due librerie:

- *BridgeClient* che è un client HTTP basato su Arduino
- *BridgeServer* che è un server HTTP basato su Arduino

#### EX1 -> Arduino Yùn come server http

```
void loop() {  
  BridgeClient client = server.accept();  
  if (client){  
    process(client);  
    client.stop(); //chiusura connessione da parte del client  
  }  
  delay(50);  
}
```

(il server trasmetterà al client, tramite una POST, un file codificato in SenML con i vari dati) e azionare o spegnere un led.

Nella loop function il server, inizializzato in setup, vede se qualche client sta facendo una richiesta, l'accetta e la serve tramite la funzione *process(client)*.

La funzione process legge il primo elemento dell'uri. Se esso corrisponde a *led* legge il campo successivo che corrisponde a "0" o "1". Se 0 spegne il led mentre se 1 lo accende. Video [qui](#).

In ogni caso manderà una risposta al client, tramite la funzione *printResponse()* con un codice identificativo di successo(200) o di errore(400 – Bad request) nel caso l'utente non avesse richiesto correttamente il servizio.

Se il primo campo non è led ma *temperature* legge il valore del sensore, lo converte in celsius con la funzione *convert()* e successivamente manda i dati sempre con la funzione *printResponse()*, con il relativo codice di successo/errore, ma tramite il corpo della risposta POST includerà la temperatura rilevata dal sensore in formato SenML.

La funzione con cui viene creato il corpo del body in formato SenML è *senMLEncode()*.

Lo scopo di questo esercizio è creare un servizio REST che accetti richieste GET per controllare vari sensori. In particolar modo richiedere dati sul sensore di temperatura

```
Ex1 §  
void process(BridgeClient client){  
  String command = client.readStringUntil('/');  
  command.trim();  
  
  if(command == "led") {  
    int val = client.parseInt();  
    if(val == 0 || val == 1) {  
      digitalWrite(LED_PIN, val);  
      printResponse(client, 200, senMLEncode(F("led"), val, F("")));  
    }  
    else {  
      printResponse(client, 400, "Bad Request");  
    }  
  }  
  else if(command == "temperature"){  
    float temp = convert(analogRead(TEMP_PIN));  
    printResponse(client, 200, senMLEncode(F("temperature"), temp, F("°C")));  
  }  
  else {  
    printResponse(client, 404, "Request Not Found");  
  }  
}
```

## EX2 -> Arduino Yùn come client HTTP

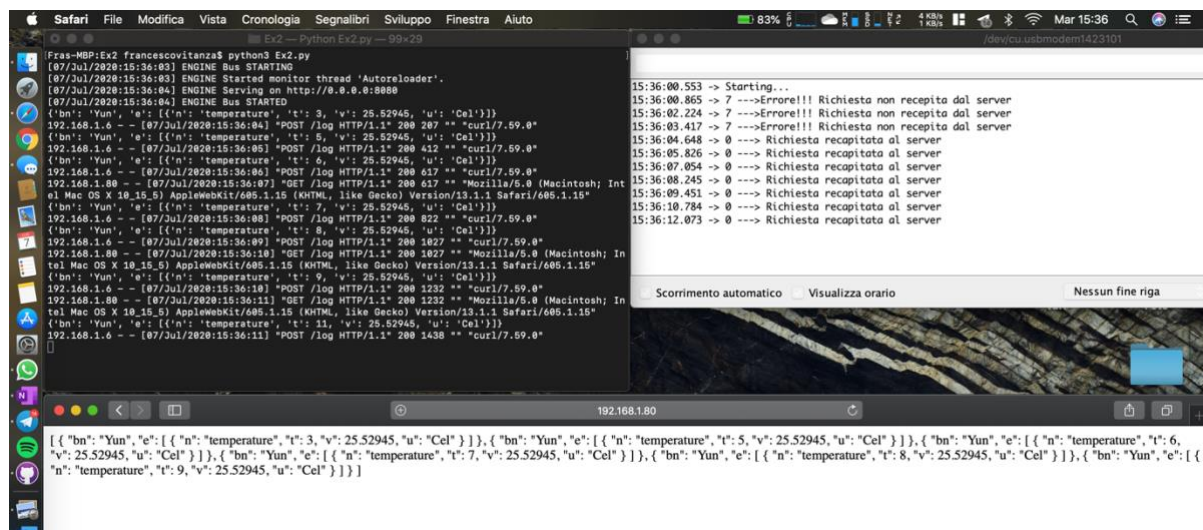
Per la realizzazione è stato creato anche un server python tramite l'utilizzo della libreria cherrypy. Tale server riceve, tramite richiesta POST, i dati relativi alla temperatura ogni secondo e gli stessi dati possono essere visionati se viene fatta una richiesta get sempre su [indirizzo server:8080/loq](http://192.168.1.80/loq).

Per quanto riguarda l'Arduino, lo sketch è stato realizzato con l'utilizzo delle librerie *Bridge* e *Process* dove *Process* permette l'utilizzo del comando curl di linux e *Bridge* permette l'utilizzo del processore Linux-like presente su Arduino Yun.

Nel setup si inizializza l'utilizzo del Bridge e si memorizza il tempo in cui si è avviato lo sketch.

Nella loop function:

- si legge il valore proveniente dal sensore di temperatura;
- tramite la funzione *senMLEncode()* creata al punto precedente si crea, secondo il dataformat SenML, l'oggetto serializzato che poi verrà inviato come stringa al server.
- La funzione *postRequest()* a cui viene passato l'oggetto serializzato esegue il comando Curl mediante i metodi della libreria process e restituisce il valore di uscita (se il server risponde correttamente restituirà 0).
- Tutta questa serie di operazioni viene ripetuta con un delay di 1 secondo.



In alto a destra troviamo il server che è in ascolto, in alto a destra possiamo vedere le richieste periodiche dell'arduino e i vari esiti che ricevono (quando riceveva 7 il server non era ancora stato avviato). In basso invece è presente la richiesta GET fatta al server con i dati recepiti fino a quel momento.

## EX3 -> Arduino Yùn come publisher MQTT

Per la realizzazione dello sketch è stato necessario l'utilizzo delle librerie *MQTTclient.h*.

Per verificare l'effettivo funzionamento sono stati utilizzati due comandi da terminale: *mosquitto\_pub* e *mosquitto\_sub*.

Nello sketch, dopo il setup iniziali e la sottoscrizione al topic, nella funzione loop viene monitorato l'arrivo o meno di un messaggio nel topic */tiot/7/led* e quando presente, tramite la funzione *setLedValue* viene deserializzato il SenML in ingresso e se viene ricevuto "1" in value, il led viene acceso altrimenti se viene ricevuto uno 0 il led viene spento.

Per quanto riguarda la funzione di temperatura, l'arduino deve operare come publisher e quindi ogni 10 secondi viene letto il valore di temperatura dal sensore e pubblicato nel topic `/tiot/7/temperature` in formato SenML tramite la funzione `SenMLEncode()`, che serializza i dati nel dataformat SenML.

The image shows a screenshot of an Arduino IDE and a terminal window. The terminal window displays the output of the `mosquitto_sub` command, showing the subscription to the topic `/tiot/7/temperature` and the received MQTT messages in JSON format. The Arduino IDE shows the code for the `loop` function, which reads the temperature from the sensor and publishes it to the topic `/tiot/7/temperature` using the `SenMLEncode` function.

```
francescovitanza ~ mosquitto_sub -h test.mosquitto.org -t /tiot/7/temperature -- 117x33
Last login: Wed Jul 8 11:21:17 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208856.
francescovitanza@francescovitanza:~$ mosquitto_sub -h test.mosquitto.org -t /tiot/7/led -m '{"bn": "Yun", "e": [{"n": "led", "v": 1, "u": null}]}'
{"bn": "Yun", "e": [{"n": "led", "v": 1, "u": null}]}
francescovitanza@francescovitanza:~$ mosquitto_sub -h test.mosquitto.org -t /tiot/7/led -m '{"bn": "Yun", "e": [{"n": "led", "v": 0, "u": null}]}'
{"bn": "Yun", "e": [{"n": "led", "v": 0, "u": null}]}
francescovitanza@francescovitanza:~$ mosquitto_sub -h test.mosquitto.org -t /tiot/7/temperature
{"bn": "Yun", "e": [{"n": "temperature", "t": 21, "v": 25.28485, "u": "Cel"}]}
{"bn": "Yun", "e": [{"n": "temperature", "t": 31, "v": 25.2834, "u": "Cel"}]}
```

```
void loop() {
  mqttn.monitor(); //funzione di monitoraggio per sezione led
  if(cnt==20)
  {
    float temp = convert(analogRead(TEMP_PIN)); //lettura e conversione
    String message = senMLEncode("temperature", temp, "Cel"); //serializzazione
    mqttn.publish(my_base_topic + String("/temperature"), message); //pubblicazione
    //debug dati su porta seriale
    Serial.print("Temp value: "); Serial.println(temp);
    Serial.print("Topic: "); Serial.println(my_base_topic + String("/temperature"));
    Serial.print("Message: "); Serial.println(message);
    cnt=0;
  }
  else
  {
    cnt=cnt+1;
  }

  delay(500);
}
```

```
11:35:41.038 -> Starting...
11:35:43.573 -> Led value: 1
11:35:49.782 -> Led value: 0
11:35:51.505 -> Temp value: 25.28
11:35:51.505 -> Topic: /tiot/7/temperature
11:35:51.505 -> Message: {"bn": "Yun", "e": [{"n": "temperature", "t": 10, "v": 25.28485, "u": "Cel"}]}
11:36:02.302 -> Temp value: 25.28
11:36:02.302 -> Topic: /tiot/7/temperature
11:36:02.302 -> Message: {"bn": "Yun", "e": [{"n": "temperature", "t": 21, "v": 25.28485, "u": "Cel"}]}
11:36:13.079 -> Temp value: 25.20
11:36:13.079 -> Topic: /tiot/7/temperature
11:36:13.079 -> Message: {"bn": "Yun", "e": [{"n": "temperature", "t": 31, "v": 25.2834, "u": "Cel"}]}
```

In figura la funzione `loop` per monitoraggio periodico topic led e pubblicazione dati relativi alla temperatura nel topic `temperature`.

Video [qui](#)

## LABORATORIO 1 SW

Servizi Web con cherryppy

### STRUMENTI UTILIZZATO.

- PC con IDE, terminale e browser
- Libreria cherryppy

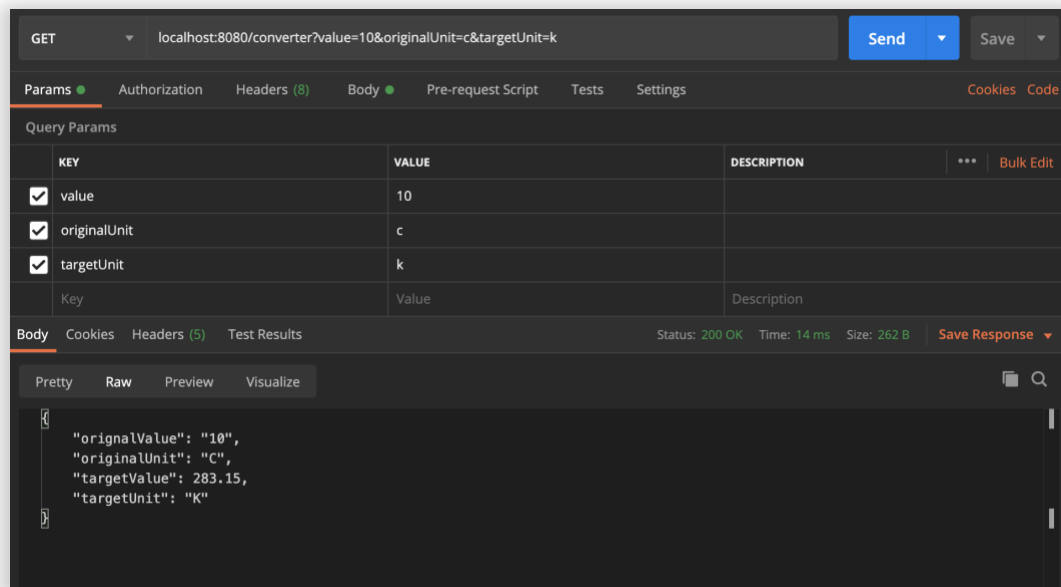
### REALIZZAZIONE.

1. **Scopo:** sviluppare un servizio web che permetta la conversione di un valore di temperatura nelle unità di misura K, °C e °F. I dati sono ricevuti come parametri di una GET.

Si è sviluppato il servizio web utilizzando la libreria cherryppy.

Il metodo GET effettua un controllo sui parametri e scatena un errore 400 (Bad Request) nel caso in cui questi non siano forniti correttamente. In caso positivo richiama la funzione *convert* che riceve come parametro l'udm iniziale, l'udm finale e una lista di valori (per compatibilità con la richiesta 3; si noti che in questo punto e nel punto 2 la lista conterrà un solo elemento) e restituisce il valore convertito nell'udm richiesta. Il valore convertito è poi inserito all'interno di un dizionario che attraverso la funzione *json.dumps()* viene restituito in formato JSON.

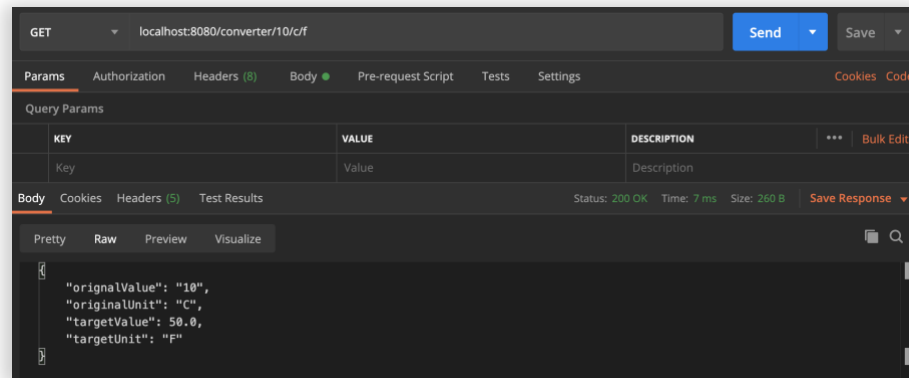
*Esempio:*



2. **Scopo:** sviluppare un servizio web che permetta la conversione di un valore di temperatura nelle unità di misura K, °C e °F. I dati sono separati da '/' nell'URL.

L'esercizio è esattamente identico al precedente, con la sola differenza che i dati sono ricevuti nell'url. Ci si aspetta il valore in prima posizione, seguito dall'udm iniziale e l'udm finale. Se il formato non è corretto viene scatenato un errore 400 (Bad Request).

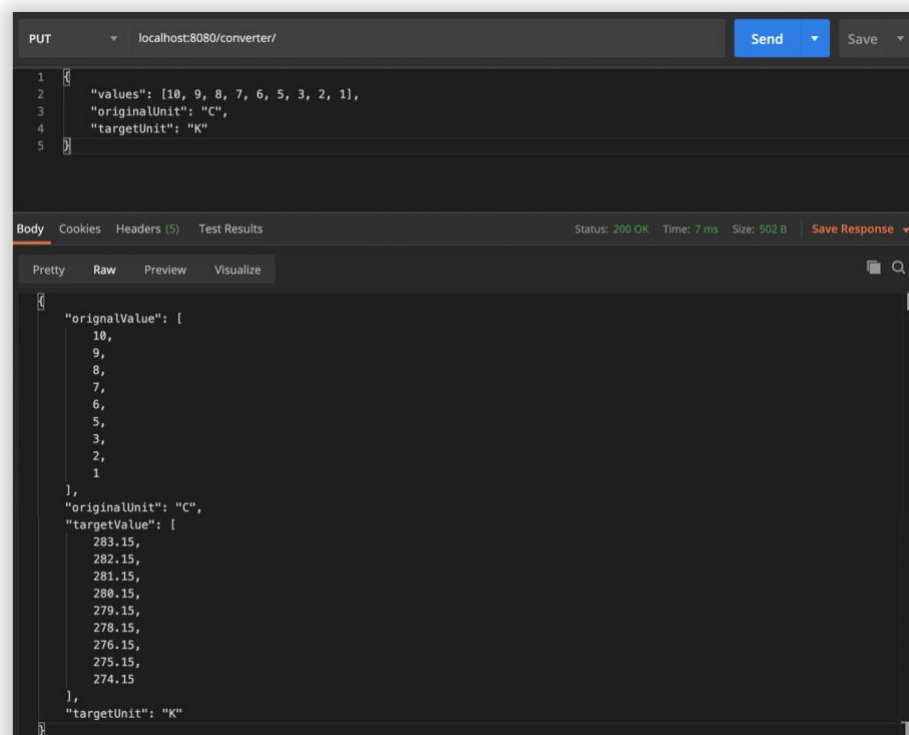
Esempio:



3. **Scopo:** sviluppare un servizio web che permetta la conversione di una lista di valori di temperatura nelle unità di misura K, °C e °F. I dati sono ricevuti in formato JSON nel body di una richiesta PUT.

Questo esercizio è un'estensione dei precedenti. Si noti che la funzione *convert* è già implementata per funzionare con una lista di valori. I valori che vengono passati come parametro alla funzione *convert* sono letti dal body tramite la funzione *json.load()*.

Esempio:

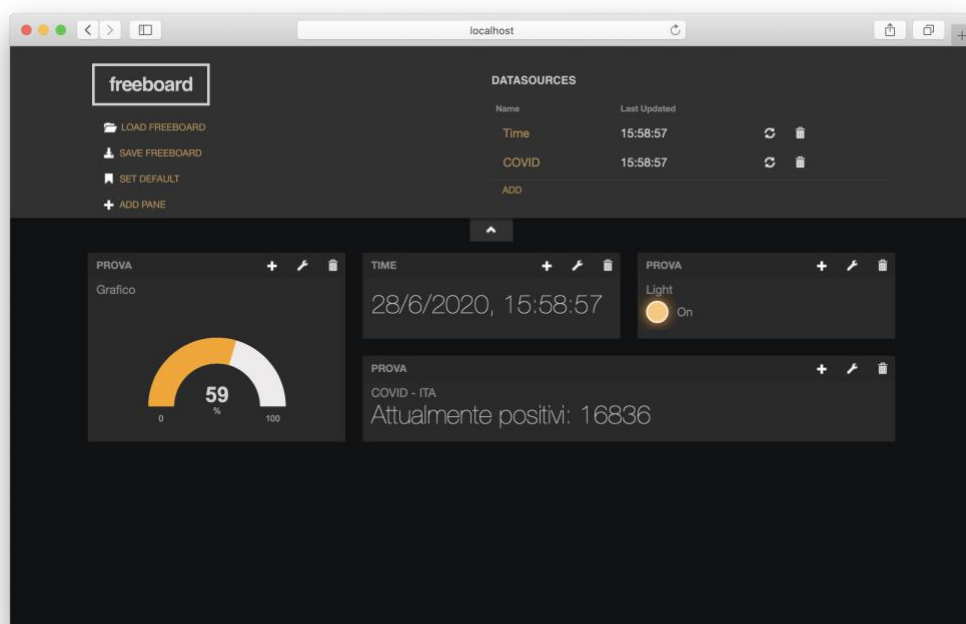
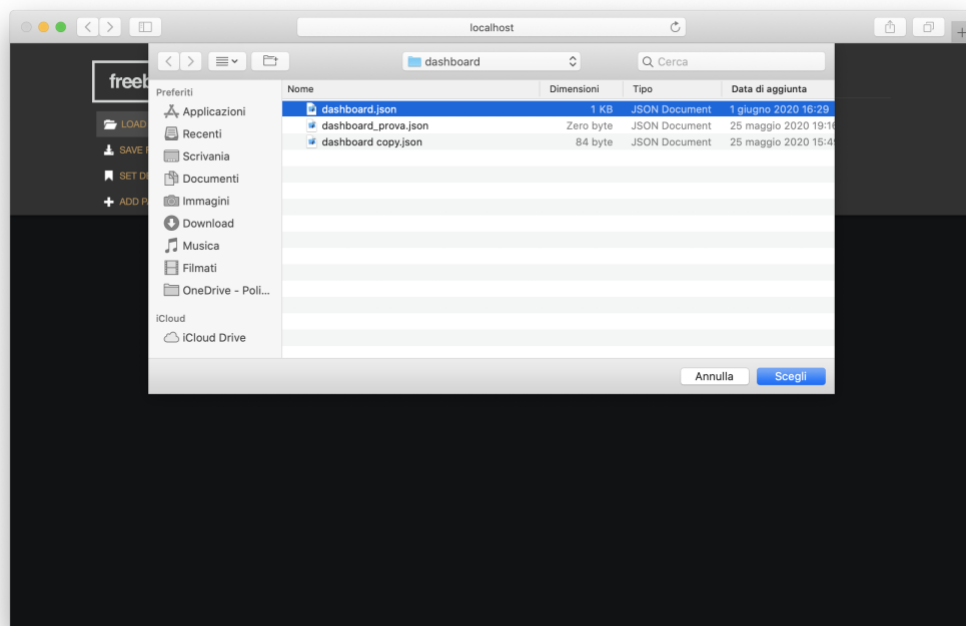




#### 4. Scopo: sviluppare un servizio web che permetta l'utilizzo di freeboard.

L'utilizzo di freeboard è stato problematico: la mancanza di documentazione ci ha richiesto un quantitativo di tempo superiore alle aspettative e nonostante ciò non ci riteniamo soddisfatti del risultato ottenuto. In particolare, anche se non era richiesto dal testo dell'esercizio, era nostra intenzione caricare in automatico l'ultima dashboard salvata all'avvio del servizio web, ma nessun metodo provato ha funzionato.

Il servizio web restituisce l'ambiente freeboard all'avvio, con la configurazione di default. Per caricare l'ultima configurazione salvata è sufficiente cliccare sul bottone "Load Freeboard" e scegliere il file *dashboard.json* disponibile nella cartella. In ogni momento si può modificare la configurazione dei widget presenti e per salvarla è sufficiente cliccare sul bottone "Save Freeboard" dal quale viene richiamata la funzione che aggiorna il file *dashboard.json*.



## LABORATORIO 2 SW

Sviluppo Catalog e client con paradigmi di comunicazione REST e MQTT.

### STRUMENTI UTILIZZATO.

- PC con IDE, terminale e browser
- Libreria cherrypy
- Libreria MQTT
- Arduino con sensori e LED (rif. Lab HW 3.2)

### REALIZZAZIONE.

1. **Scopo:** sviluppare un Catalog in stile REST che gestisca dispositivi, utenti e servizi di una piattaforma.

Tutto il sistema è gestito da cherrypy che si richiama diverse classi che gestiscono le richieste. I dati vengono salvati in un file JSON denominato *data.json* all'interno della cartella *res*.

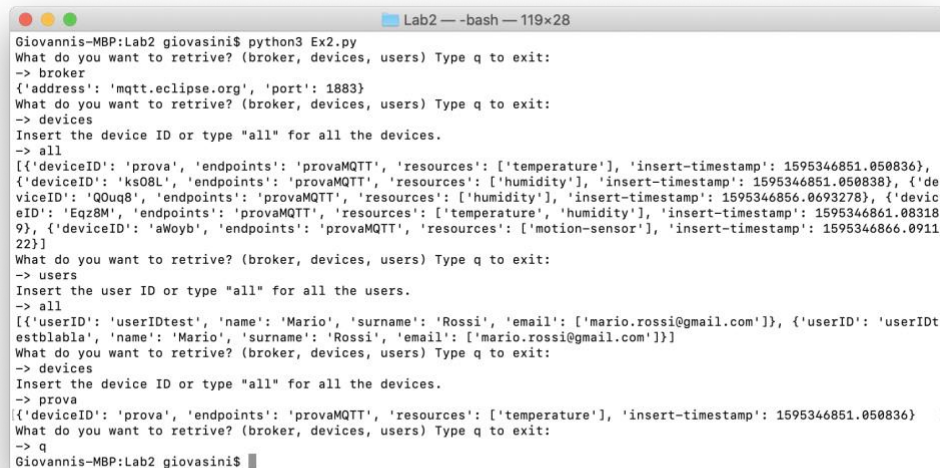
Le funzionalità implementate sono le seguenti:

- a. Il broker, essendo definito una volta per tutte, viene semplicemente letto dal file *data.json* e restituito quando viene effettuata una richiesta GET /broker.
  - b. Per aggiungere un nuovo dispositivo bisogna fare una richiesta POST /devices e fornire come body della richiesta i dati del nuovo device in formato JSON. Il nuovo dispositivo viene aggiunto al file *data.json* se non è già presente un dispositivo con lo stesso ID. Se il DeviceID esiste già viene scatenato un errore HTTP 409.
  - c. Facendo una richiesta GET /devices il sistema legge la lista di dispositivi nel file *data.json* e la restituisce in formato JSON.
  - d. Facendo una richiesta GET /devices/deviceID il sistema legge il file, cerca il device ricercato e se lo trova restituisce un JSON contenente i suoi dati. Se non trova il dispositivo viene scatenato un errore HTTP 404.
  - e. L'inserimento di un nuovo utente è molto simile all'inserimento di un nuovo device. Viene fatta una richiesta POST /users e nel body vengono forniti i dati del nuovo utente. Se l'utente non esiste già viene aggiunto alla lista di utenti.
  - f. La lista di utenti in formato JSON è restituita quando viene fatta una richiesta GET /users.
  - g. I dati di un utente sono restituiti facendo una richiesta GET /users/userID, ovviamente se l'utente esiste. In caso contrario si scatena un errore HTTP 404.
- 8) 9) 10) La gestione dei servizi è del tutto analoga. Le richieste fanno fatte su /services.
- 11) Ai dispositivi e ai servizi che vengono registrati viene aggiunta la data e l'ora di inserimento. Insieme a cherrypy, lavora in parallelo un thread che verifica periodicamente se un dispositivo o un servizio è stato registrato da più di 2 minuti e in quel caso lo rimuove.

Nota: tutti gli accessi in scrittura al file, essendo potenzialmente concorrenti con quest'ultimo thread descritto, vengono effettuati utilizzando il meccanismo di lock.

## 2. Scopo: sviluppare un client in python che utilizzi i servizi forniti dall'esercizio 1.

Si è sviluppato un programma che interagisce con l'esercizio precedente mediante terminale. Il menu di interfaccia è interattivo e l'utente può richiedere le informazioni che desidera.



```
Giovannis-MBP:Lab2 giovasini$ python3 Ex2.py
What do you want to retrieve? (broker, devices, users) Type q to exit:
-> broker
{'address': 'mqtt.eclipse.org', 'port': 1883}
What do you want to retrieve? (broker, devices, users) Type q to exit:
-> devices
Insert the device ID or type "all" for all the devices.
-> all
[{'deviceID': 'prova', 'endpoints': 'provaMQTT', 'resources': ['temperature'], 'insert-timestamp': 1595346851.050836},
{'deviceID': 'ks08L', 'endpoints': 'provaMQTT', 'resources': ['humidity'], 'insert-timestamp': 1595346851.050838}, {'deviceID': 'Q0uq8', 'endpoints': 'provaMQTT', 'resources': ['humidity'], 'insert-timestamp': 1595346856.0693278}, {'deviceID': 'Eqz8M', 'endpoints': 'provaMQTT', 'resources': ['temperature', 'humidity'], 'insert-timestamp': 1595346861.083189}, {'deviceID': 'aWoyb', 'endpoints': 'provaMQTT', 'resources': ['motion-sensor'], 'insert-timestamp': 1595346866.091122}]
What do you want to retrieve? (broker, devices, users) Type q to exit:
-> users
Insert the user ID or type "all" for all the users.
-> all
[{'userID': 'userIDtest', 'name': 'Mario', 'surname': 'Rossi', 'email': ['mario.rossi@gmail.com']}, {'userID': 'userIDtestblabla', 'name': 'Mario', 'surname': 'Rossi', 'email': ['mario.rossi@gmail.com']}]
What do you want to retrieve? (broker, devices, users) Type q to exit:
-> devices
Insert the device ID or type "all" for all the devices.
-> prova
[{'deviceID': 'prova', 'endpoints': 'provaMQTT', 'resources': ['temperature'], 'insert-timestamp': 1595346851.050836}]
What do you want to retrieve? (broker, devices, users) Type q to exit:
-> q
Giovannis-MBP:Lab2 giovasini$
```

Utilizzo come variabile globale il `server_ip` del Catalog su cui fare le richieste e in base alle scelte dell'utente vengono inviate le richieste GET, secondo le specifiche descritte sopra, e stampati su console i dati restituiti dal Catalog.

## 3. Scopo: sviluppare un client che utilizzi i servizi forniti dal Catalog registrando nuovi dispositivi o aggiornando le registrazioni di dispositivi esistenti.

Questo programma genera un numero random ogni 60 secondi e in base a criteri scelti arbitrariamente aggiorna un dispositivo o ne crea uno nuovo.

Se il numero è multiplo di 10 li aggiorna tutti, se è pari sceglie un dispositivo da aggiornare, se è dispari ne aggiunge uno nuovo.

Per aggiornare i dispositivi fa una richiesta GET /refresh sul Catalog che dopo averla elaborata restituisce un codice di stato (HTTP).

Quando si deve aggiungere un dispositivo nuovo, questo viene creato in un dizionario con dati casuali, a partire dal codice, fino alle risorse, per poi essere inviato nel body di una richiesta POST su `server_ip/devices` in formato JSON.

Nota: la funzionalità di aggiornamento (apparentemente non richiesta) nell'esercizio 1, probabilmente non è stata implementata nel modo più efficiente possibile: questo è stato fatto successivamente.

## 4. Scopo: estendere l'esercizio 2 del Lab HW 3 permettendo ad Arduino di inviare la sua registrazione al Catalog.

Si è preso l'esercizio 2 del Lab HW 3 e si sono aggiunte le funzioni per l'invio della registrazione. Per problemi di spazio su Arduino, alcune funzionalità precedenti sono state eliminate. Si è utilizzato questo esercizio per testare semplicemente l'invio della registrazione da Arduino. L'ottimizzazione dello spazio su Arduino si è studiata nel laboratorio successivo.

**5. Scopo: estendere le funzionalità del Catalog (esercizio 1) per funzionare anche in MQTT.**

Si è fatta una copia del Catalog del punto 1 e modificato per funzionare anche in MQTT. Utilizzando la libreria *paho.mqtt.client* si è implementato un Subscriber che si iscrive al topic /catalog IoT Tech/refresh/or/add/device. Quando riceve un messaggio esegue le stesse operazioni di quando riceve una richiesta HTTP di registrazione o aggiornamento.

**6. Scopo: sviluppare un client che utilizzi i nuovi servizi forniti dal Catalog registrando nuovi dispositivi o aggiornando le registrazioni di dispositivi esistenti in MQTT**

Si è sviluppato un client del tutto simile all'esercizio 3 che però non lavora con i servizi REST ma con MQTT. Si è implementato un publisher MQTT utilizzando la libreria *paho.mqtt.client*. Lo stesso body inviato precedentemente all'interno del POST è inviato ora come messaggio sul topic definito al punto precedente.

## LABORATORIO 3 SW

Sviluppo Catalog e client con paradigmi di comunicazione REST e MQTT.

### STRUMENTI UTILIZZATO.

- PC con IDE, terminale e browser
- Libreria cherrippy
- Libreria MQTT
- Arduino con sensori e LED (rif. Lab HW 3.2)

### REALIZZAZIONE.

Si è riutilizzato il Catalog sviluppato nel lab precedente per tutti gli esercizi di questo laboratorio.

#### **1. Scopo: prendere dimestichezza con MQTT.**

Si è utilizzato il terminale con i comandi *mosquitto\_pub* e *mosquitto\_sub* provando topic diversi, anche su broker differenti. Inoltre abbiamo provato la sottoscrizione a un topic utilizzando le wildcards.

Dopodiché abbiamo anche sviluppato più client python che utilizzando la libreria *paho.mqtt.client*, testando anche la possibilità di avere più subscriber iscritti allo stesso topic.

#### **2. Scopo: sviluppare un client in python che utilizzi i servizi forniti dall'esercizio 1.**

Facendo riferimento al Lab 3.3 HW si è realizzato uno sketch Arduino che, ogni 100 secondi, invia tramite la funzione *postRequest()* una richiesta POST al Catalog contenente nel body il SenML di registrazione costruito mediante la funzione *senMLEncodeRegistration()*. Nel setup si inizializza il client MQTT che, ogni 10 secondi, invia un messaggio in formato SenML contenente i dati di temperatura.

Dall'altro lato, in Python, si è sviluppato un client che, utilizzando i servizi forniti dal Catalog, ottiene le informazioni per sottoscrivere al topic dove riceve i dati di temperatura inviati dall'Arduino. Si ottiene un log con tutti i dati di temperatura.



The screenshot displays three terminal windows. The top window, titled 'Terminale', shows the Arduino IDE serial monitor output, displaying MQTT messages from the Arduino to the broker. The middle window, titled 'Ex2\_Lab3SW\_Prova - Python Ex2.py - 80x21', shows the Python client logs, including connection status and data received from the broker. The bottom window, titled 'Ex2\_Lab3SW\_Prova - Python Catalog.py - 80x24', shows the MQTT broker logs, including device registration and message handling.

In alto il log su seriale di Arduino, a destra il log del Catalog e a sinistra il log del client MQTT.

### 3. Scopo: sviluppare un client in python che interagisca con l'Arduino tramite MQTT.

In questo esercizio si sono invertiti i ruoli tra client Python e Arduino. Infatti in questo caso l'Arduino si sottoscrive a un topic sul quale il client Python pubblica messaggi di comando per controllare un led. Le modalità di invio dei messaggi sono le stesse dell'esercizio precedente. Il client Python invia il messaggio di comando ogni 10 secondi, invertendo lo stato del led. L'Arduino, quando riceve il messaggio, imposta il led sullo stato ricevuto.

The screenshot displays three terminal windows. The top window, titled 'Terminale', shows the Arduino IDE serial monitor output, displaying MQTT messages from the Arduino to the broker. The middle window, titled 'Ex3\_Lab3SW\_Prova - Python Ex3.py - 80x24', shows the Python client logs, including connection status and data received from the broker. The bottom window, titled 'Ex3\_Lab3SW\_Prova - Python Catalog.py - 80x24', shows the MQTT broker logs, including device registration and message handling.

In alto il log su seriale di Arduino, a sinistra il log del Catalog e a destra il log del client MQTT.

Video [qui](#)

#### 4. Scopo: sviluppare la versione remota del laboratorio 2 HW.

Si è modificato il laboratorio 2 HW aggiungendo le funzionalità necessarie.

Dal lato Arduino per problemi di memoria non si è più utilizzata la libreria `<ArduinoJSON.h>`, ma i messaggi inviati sono stati composti manualmente concatenando le stringhe.

Inoltre, sempre per risparmiare memoria, si è utilizzata la funzione `F("string")` per salvare le stringhe utilizzate nella memoria flash invece che in RAM.

L'Arduino si registra al Catalog facendo una richiesta POST e inserendo nei propri endpoints i topic a cui il client python può sottoscrivere ("temp" e "move"; "sound" non è più implementato: vedi righe successive).

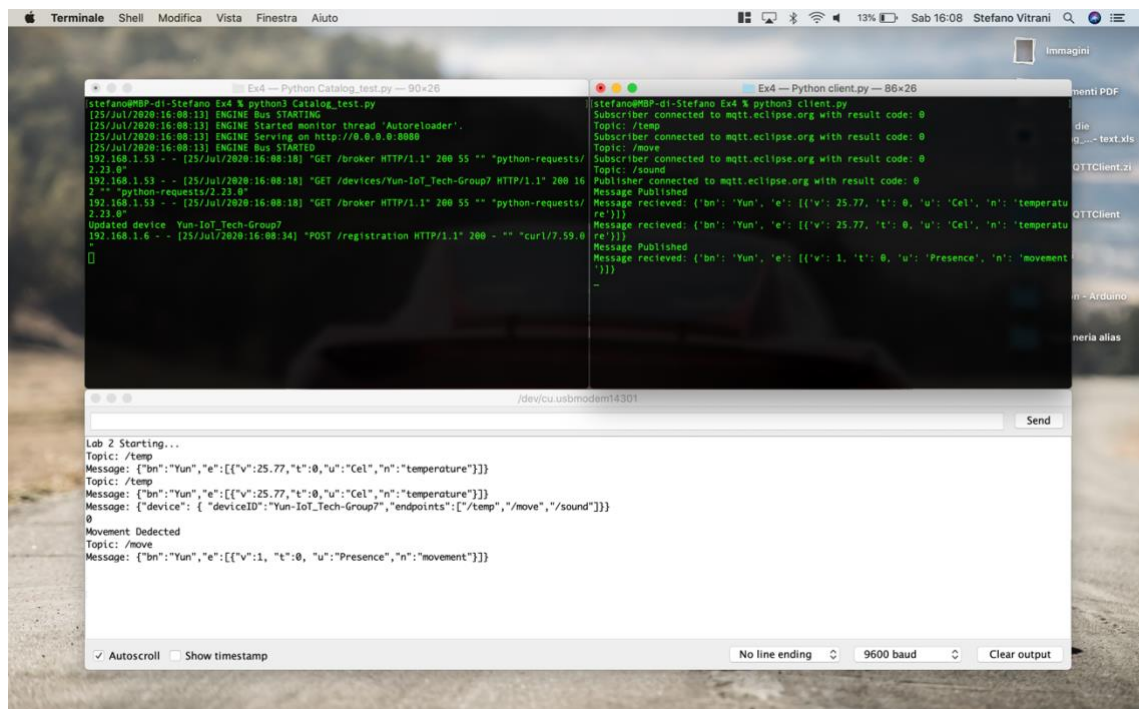
Inoltre l'Arduino dovrà sottoscrivere al topic "messageLCD" sul quale vengono inviati i messaggi da stampare sull'LCD.

Le funzioni `fresh()` e `heat()` rimangono invariate e ogni 5 secondi viene inviato un messaggio in senML contenente il dato di temperatura sul relativo topic.

Ogni 3 secondi, in polling, viene controllato lo stato del PIR che se è attivo attiva la funzione `movement_detected()` e manda il messaggio in senML sul topic "move" notificando la presenza. Se per 60 secondi nessuna presenza è rilevata viene inviato un messaggio simile che notifica l'assenza di movimenti.

A differenza del lab 2 HW, dove il processore Linux non era attivo, in questo esercizio non è stato possibile utilizzare il sensore di suono collegato al pin 7 perché, come abbiamo scoperto dalla documentazione ufficiale Arduino, questo, con l'attivazione del Bridge, è utilizzato per la comunicazione tra microcontrollore e processore Linux.

Dall'altro lato, in Python, è implementato il Catalog allo stesso modo degli esercizi precedenti. Si è aggiunta un client MQTT, separato dal Catalog, che richiede al Catalog gli endpoints dell'Arduino e a cui si sottoscrive. Inoltre questo client pubblica sul topic il messaggio da stampare sull'LCD.



```
stefano@Pi:~$ python3 Catalog_test.py
[25/Jul/2020:16:00:13] ENGINE Bus STARTING
[25/Jul/2020:16:00:13] ENGINE Started monitor thread 'Autoreloader'.
[25/Jul/2020:16:00:13] ENGINE Serving on http://0.0.0.0:8080
[25/Jul/2020:16:00:13] ENGINE Bus STARTED
192.168.1.53 - - [25/Jul/2020:16:00:18] "GET /broker HTTP/1.1" 200 55 "" "python-requests/2.23.0"
192.168.1.53 - - [25/Jul/2020:16:00:18] "GET /devices/Yun-IoT_Tech-Group? HTTP/1.1" 200 162 "" "python-requests/2.23.0"
192.168.1.53 - - [25/Jul/2020:16:00:18] "GET /broker HTTP/1.1" 200 55 "" "python-requests/2.23.0"
Updated device Yun-IoT_Tech-Group
192.168.1.6 - - [25/Jul/2020:16:00:34] "POST /registration HTTP/1.1" 200 - "" "curl/7.59.0"

stefano@Pi:~$ python3 client.py
Subscriber connected to mqtt.eclipse.org with result code: 0
Topic: /temp
Subscriber connected to mqtt.eclipse.org with result code: 0
Topic: /move
Subscriber connected to mqtt.eclipse.org with result code: 0
Topic: /sound
Publisher connected to mqtt.eclipse.org with result code: 0
Message Published
Message received: {'bn': 'Yun', 'e': [{'v': 25.77, 't': 0, 'u': 'Cel', 'n': 'temperature'}]}
Message received: {'bn': 'Yun', 'e': [{'v': 25.77, 't': 0, 'u': 'Cel', 'n': 'temperature'}]}
Message Published
Message received: {'bn': 'Yun', 'e': [{'v': 1, 't': 0, 'u': 'Presence', 'n': 'movement'}]}

Lab 2 Starting...
Topic: /temp
Message: {'bn': 'Yun', 'e': [{'v': 25.77, 't': 0, 'u': 'Cel', 'n': 'temperature'}]}
Topic: /temp
Message: {'bn': 'Yun', 'e': [{'v': 25.77, 't': 0, 'u': 'Cel', 'n': 'temperature'}]}
Message: {'device': {'deviceID': 'Yun-IoT_Tech-Group7', 'endpoints': ['/temp', '/move', '/sound']}}
0
Movement Detected
Topic: /move
Message: {'bn': 'Yun', 'e': [{'v': 1, 't': 0, 'u': 'Presence', 'n': 'movement'}]}
```

*In alto a sinistra il Catalog, a destra il client MQTT e in basso il log sulla seriale di Arduino.*

## LABORATORIO 4 SW

Sviluppo sistema di controllo Basic Smart Home.

### STRUMENTI UTILIZZATO.

- PC con IDE, terminale e browser
- Client e API Telegram
- Libreria cherrypy
- Libreria MQTT
- Arduino con sensore di temperatura e LED

### OBIETTIVI.

Sviluppo di una Basic Smart Home controllata da Arduino tramite interfaccia web e bot Telegram utilizzando paradigmi di comunicazione REST e MQTT.

In questa versione ristretta si è utilizzato solamente un sensore di temperatura e alcuni led (a simulare le luci smart), ma si potrebbe ampliare semplicemente con una scheda di controllo più performante, visti i problemi riscontrati negli esercizi precedenti.

### REALIZZAZIONE.

Il Catalog è stato ereditato dagli esercizi precedenti e modificato secondo le nuove necessità.

In particolare abbiamo migliorato la gestione della registrazione dei dispositivi. Oltre ai dati sul device e i servizi offerti, vengono memorizzate le informazioni sullo stato delle periferiche.

Il Catalog offre anche il Web Server che permette all'utente di controllare la Smart Home in modo semplice e intuitivo tramite una pagina Web.

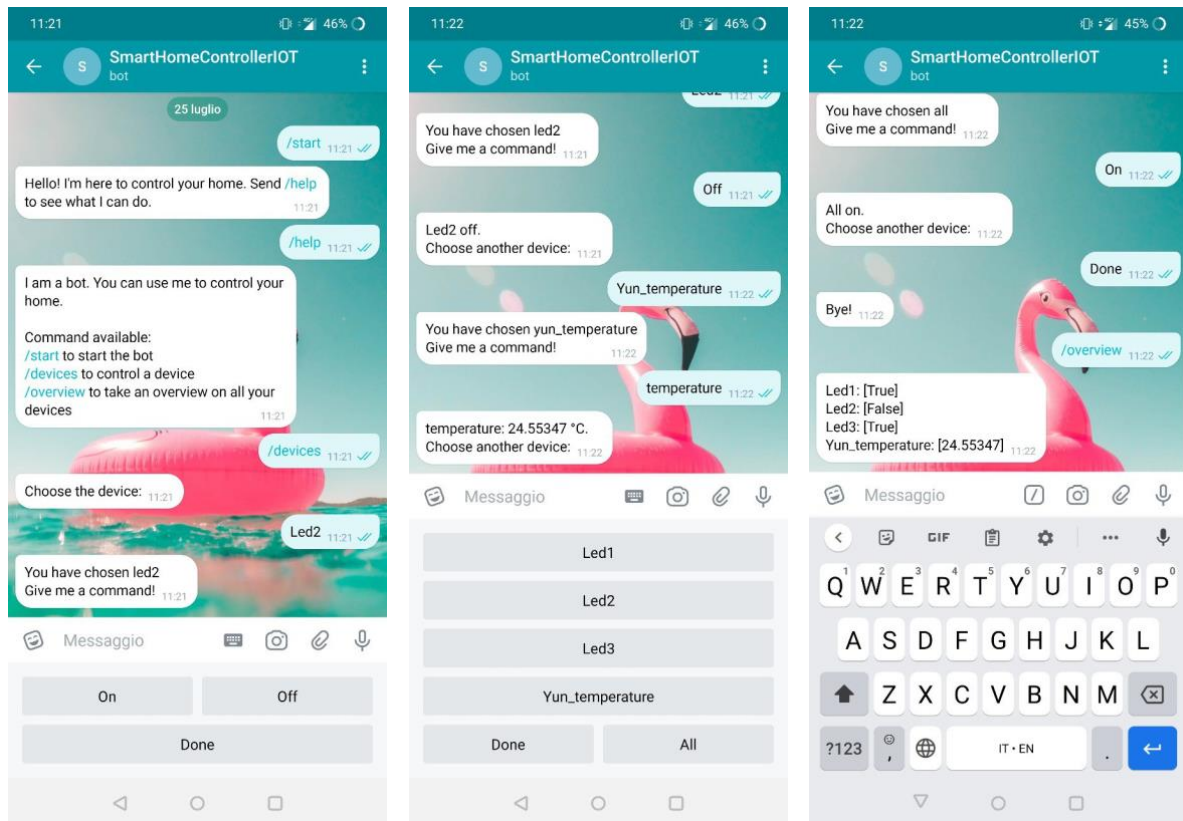
Si è implementato un bot Telegram utilizzando la libreria di Python *python-telegram-bot* per gestire la Smart Home con una chat su un client Telegram.

Il bot interagisce con il Catalog tramite il paradigma REST e non è necessario che le due applicazioni vengano eseguite sulla stessa macchina.

Per avviare il bot in chat è sufficiente aprire la conversazione con SmartHomeControllerIOT e inviare il comando */start*. Il bot risponde con un messaggio di benvenuto e invita l'utente a inviare il comando */help* per conoscere i comandi disponibili.

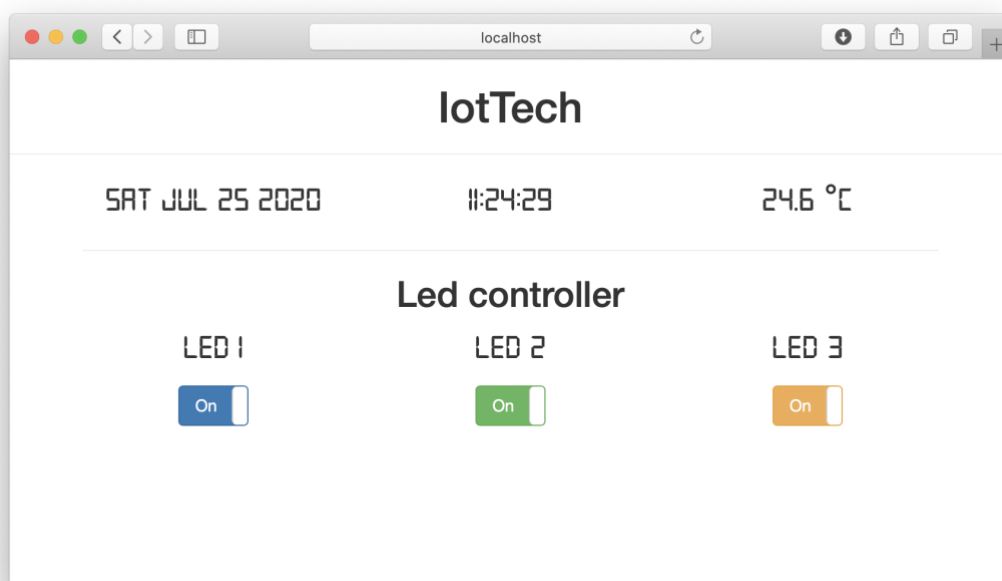
Inviando il comando */devices* al bot in chat si apre una tastiera personalizzata che mostra tutti i dispositivi disponibili. Cliccando su uno dei dispositivi appare un'altra tastiera che riporta le funzionalità del dispositivo, mentre attraverso il comando */overview* si possono ottenere informazioni generali sullo stato dei dispositivi.

Per l'implementazione del back-end si è studiata l'ampia documentazione fornita dagli sviluppatori della libreria sopra citata.



La pagina HTML restituita dal web server è costruita utilizzando la libreria Bootstrap. Questa libreria permette di ottenere un design più accattivante e una gestione più semplice della pagina dinamica. La pagina, suddivisa in contenitori, si adatta automaticamente al dispositivo utilizzato rendendola fruibile anche da dispositivi mobile.

Nella pagina, sotto al titolo, è presente una prima riga che riporta data, ora e temperatura attuale della casa. Al di sotto sono disponibili i bottoni di controllo dei led della casa.



Video [qui](#). Nota: su iPad a sinistra è aperto il bot Telegram, a destra il browser.