UNIVERSITÀ DEGLI STUDI
DI GENOVA

DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA,
ELETTRONICA E DELLE TELECOMUNICAZIONI

CORSO DI STUDIO IN INGEGNERIA ELETTRONICA E
TECNOLOGIE DELL'INFORMAZIONE

Tesi di laurea triennale

Luglio 2021

Progetto e implementazione di un sistema embedded per il monitoraggio del distanziamento sociale

Candidati: Marco Macchia e Samuele Pedrazzi

Relatore: Prof. Riccardo Berta

Correlatore: Dott. Luca Lazzaroni

Sommario

La presente tesi prevede la progettazione e la realizzazione di un sistema embedded destinato all'acquisizione di dati relativi a misurazioni di distanza e temperatura per monitorare il distanziamento sociale. Il dispositivo comprende due sensori principali che misurano le grandezze appena citate, oltre a degli attuatori come un modulo di vibrazione e un led.
Per l'elaborazione delle informazioni provenienti dai sensori viene utilizzata una scheda MKR WiFi 1010 che permette il collegamento ad internet e l'invio dei dati acquisiti al cloud attraverso specifiche API messe a disposizione dal framework Measurify, il quale si interpone tra il sistema di acquisizione dati e l'applicazione smartphone che ricopre il ruolo di interfaccia grafica per l'utente. Attraverso il toolkit Flutter di Google è stata sviluppata l'app che garantisce un riscontro veloce e semplice delle misurazioni effettuate suddivise in fasce orarie per una maggiore leggibilità per l'utilizzatore.

Indice

1 Introduzione

Nell'ultimo decennio l'Internet of Things (IoT), o Internet delle Cose, ovvero quel percorso nello sviluppo tecnologico in base al quale teoricamente ogni oggetto dell'esperienza quotidiana acquista una sua identità nel mondo digitale, è riuscito a connettere miliardi di dispositivi in tutto il mondo al fine di migliorare e semplificare la vita di tutti i giorni.
L'IoT si può trovare in moltissimi settori quali la sanità, i trasporti, il settore automobilistico, la domotica ma potenzialmente l'idea è quella di riuscire a far diventare Smart (digitale) ogni oggetto nel mondo fisico, ossia collegare tutte le *cose* di un determinato contesto tra di loro per creare un ambiente connesso intelligente.

Al momento sono connessi circa 35 miliardi di dispositivi IoT [1] ma il numero è destinato ad aumentare vertiginosamente nei prossimi anni.
Per rendere Smart un oggetto fisico entra in gioco l'elettronica e specificatamente l'utilizzo di microcontrollori e sensori per il monitoraggio di un determinato ambiente oppure moduli per la connessione internet per collegare gli oggetti tra di loro e salvare su cloud i rilevamenti.
Basti pensare all'implementazione di un sistema IoT all'interno di un'automobile, la connessione tra veicoli, o tra questi e l'infrastruttura circostante fornirebbe grandi aiuti per la prevenzione e la rivelazione degli incidenti e garantirebbe un miglioramento della sicurezza stradale, oppure l'introduzione di sensori in grado di percepire un urto che grazie ad un sistema automatico permettano di avviare una chiamata di soccorso immediata al 112 ed inviino i dati relativi all'incidente come orario, posizione del veicolo incidentato e direzione di marcia.

Figura 1: Internet of Things

Di fronte all'emergenza sanitaria per il SARS-CoV-2 è naturale domandarsi se si possa utilizzare l'IoT per favorire le misure di prevenzione dei contagi ed agevolare le regole di contenimento negli spazi chiusi.
Per questo motivo l'idea è quella di progettare un sistema embedded in campo IoT che possa monitorare le distanze tra le persone al fine di assicurare la distanza di sicurezza tra i singoli individui.

Pensando ad un contesto più generale il dispositivo può essere adoperato ad esempio per un sistema centralizzato di monitoraggio che controlla gli utenti all'interno di una sala di un museo o in un qualsiasi spazio chiuso per gestire nel miglior modo possibile i flussi di persone o far rispettare la distanza minima tra i visitatori.

Andando a coinvolgere quindi le tematiche legate all'emergenza Covid-19 sfruttando la tecnologia dell'Internet of Things si è voluto implementare un dispositivo che permettesse di misurare la distanza tra le persone e anche di rilevare la temperatura corporea, per poi informare il fruitore tramite dispositivo e tramite applicazione smartphone se risulta essere troppo vicino ad una persona o lo è stato e quando, e quali misure di temperatura sono state rilevate.

Per poter disporre dei dati raccolti dal dispositivo in un secondo momento o quando non si abbia esso a disposizione, le misurazioni vengono inviate e salvate sul cloud tramite una board Arduino MKR WiFi 1010 che permette la connessione internet grazie al modulo WiFi-Nina.

L'applicazione consente di avere un'interfaccia utente sempre disponibile ed aggiornata a portata di mano. Per richiamare l'attenzione dell'utilizzatore è stato inoltre implementato un sistema di notifiche che informa, anche quando si ha l'applicazione chiusa, la ricezione di un alert, ovvero una distanza inferiore alla soglia richiesta.

2 Metodi e strumenti utilizzati

2.1 Measurify

La gestione dei dati rilevati dall'hardware di Arduino è un aspetto essenziale del progetto.
Per permettere all'utente di controllare le misurazioni ottenute e revisionarle durante la giornata in qualsiasi momento, a prescindere dallo stato del dispositivo fisico, viene impiegato un framework per la comunicazione sul cloud.

Precedentemente chiamato Atmosphere IoT Framework, Measurify [2] è una API di tipo RESTful che si concentra sull'elaborazione di dati e sulla gestione di oggetti intelligenti in ecosistemi IoT, sviluppata nell'Università di Genova da Elios Lab.

Measurify, come osservabile nella figura 2 si basa su alcuni concetti fondamentali:

- **Thing**: si tratta di un oggetto su cui possono essere effettuate delle misurazioni (measurements). Nel nostro caso si tratta dell'ambiente circostante, quindi ad esempio una stanza con all'interno delle persone.
- **Feature**: rappresenta la grandezza fisica a cui ci si riferisce quando si effettua una misurazione da un device.
- **Device**: si tratta di un dispositivo o sensore tramite il quale si può misurare una dimensione fisica (Feature), rispetto a una determinata cosa (Thing).
- **Measurement**: il termine fa riferimento all'effettivo valore che rappresenta la misurazione di una *feature*.

Nel nostro sistema la *thing* rappresenta un'eventuale spazio chiuso con all'interno degli individui, le *feature* sono la distanza e la temperatura, i *device* sono i sensori impiegati per il rilevamento delle *measurement* che sono le misure delle grandezze fisiche analizzate.

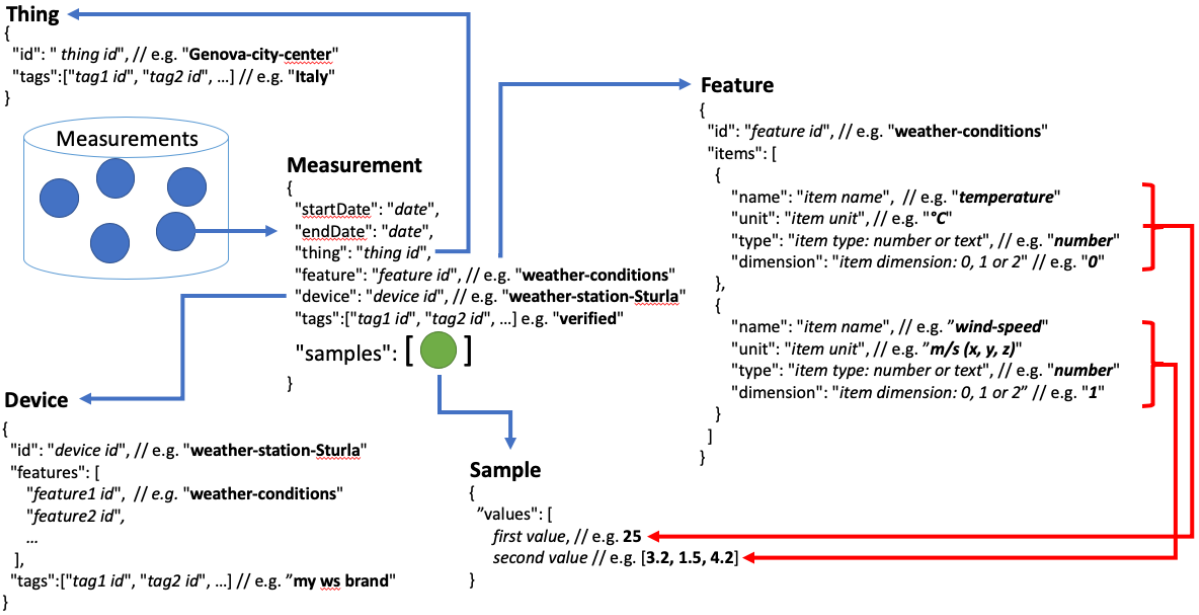


Figura 2: Schema dei concetti fondamentali di Measurify

I concetti sopra citati costituiscono delle risorse che a loro volta diventano oggetti nell'ambiente di Measurify. È possibile accedere ad ogni oggetto (memorizzato in formato JSON) tramite le API con una connessione internet.

Per poter accedere a tutte le risorse memorizzate all'interno dell'ambiente Measurify è necessario un token di autorizzazione, che il server invia al client durante la procedura di Login e che deve essere sempre presente nell'header di ogni chiamata HTTP.
La procedura di Login, eseguibile tramite POST all'indirizzo

https://students.atmosphere.tools/v1/login

prevede l'invio tramite body di tre credenziali, username, password e tenant come mostrato nel codice che segue:

```
{
    "username": "distancing-user-username",
    "password": "distancing-user-password",
    "tenant": "social-distancing-reminder"
}
```

In risposta, il server Measurify invia il token di autorizzazione, che viene memorizzato sul dispositivo per poter effettuare tutte le restanti chiamate necessarie.
Il token ha una validità di 30 minuti, terminati i quali è necessario effettuare nuovamente la procedura di Login e il salvataggio del nuovo token.

2.2 Postman

Ogni accesso al cloud per la ricezione o l'invio di dati da parte del dispositivo viene gestito attraverso le API RESTful tramite delle chiamate HTTP che differiscono in base all'azione che si vuole compiere (GET, POST, DELETE ecc.).

Postman è un software per sviluppo di API [3] che permette di predisporre un ambiente dove si possono fare chiamate per creare (POST), accedere (GET) o modificare (PUT) le risorse specifiche al progetto, nel nostro caso delle misurazioni di distanza, di temperatura o degli alert.

Caratteristica molto interessante di Postman è la possibilità di memorizzare le principali richieste HTTP usate per l'interazione con l'API e tutte le variabili d'ambiente che possono essere utilizzate nelle successive chiamate, come ad esempio il token di validazione.

Per controllare che la misura sia andata a buon fine è possibile interrogare le API per avere le misure precedentemente memorizzate. A volte è opportuno cambiare il comportamento del dispositivo al fine di ricevere delle misure solo in determinate condizioni. È quindi possibile effettuare una chiamata alla API per ottenere uno *script* con le informazioni utili alla definizione di come il micro-controllore si debba comportare nel memorizzare le richieste. Gli "script" nel nostro progetto definiscono come generare un *alert* che viene notificato tramite applicazione in caso di distanza ravvicinata sotto i 100 cm (soglia che può essere modificata a discrezione dell'utente tramite l'app stessa). Sotto è possibile osservare lo script relativo agli alert:

```
{
    "code": "distance().filter(<100).send(alert)"
}
```

2.3 Edge Engine

Edge Engine è una libreria, sviluppata anch'essa da Elios Lab, che permette di elaborare dati ricevuti da sensori attraverso degli script e di comandare attuatori sulla base delle misurazioni ottenute. [4]

Gli script sono un insieme di operazioni che possono essere eseguite sui flussi di dati in arrivo dai sensori per permettere un'elaborazione specifica delle misurazioni. Ogni script è descritto all'interno di un server cloud, ed Edge Engine può essere configurato per scaricarlo ed eseguirlo sul sistema locale. Periodicamente il motore controlla l'aggiunta di nuovi script o eventuali cambiamenti nelle descrizioni, e in caso affermativo effettua un aggiornamento locale.

L'idea finale è avere a disposizione una libreria utile per interagire con le API di Measurify, permettendo allo sviluppatore di eseguire della computazione direttamente sulla scheda Arduino, descrivendone però il comportamento sul cloud.

Durante la fase di sviluppo è stato necessario occuparsi del porting su Edge Engine per schede Arduino, come analizzato in seguito.

2.4 Arduino

2.4.1 Arduino MKR WiFi 1010

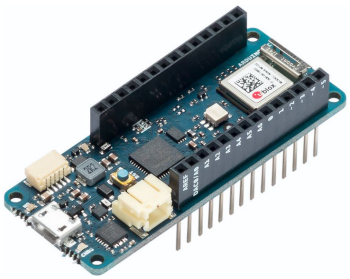


Figura 3: Scheda Arduino MKR WiFi 1010

La scheda Arduino MKR WiFi 1010 [5] è uno dei principali dispositivi elettronici entry-level dedicati all'ambito IoT. Essa nasce come versione aggiornata della scheda MKR 1000 ed implementa un nuovo modulo radio (visibile in figura 3 nella parte in alto) u-blox NINA-W102, il quale migliora la connettività WiFi e aggiunge la connettività BLE (Bluetooth Low Energy).

MKR 1010 è dotata di 8 pin I/O digitali e di 13 pin PWM, e supporta la possibilità di utilizzare fino a 10 pin di ricezione di segnali di interrupt.
Caratteristica rilevante della scheda è il basso consumo energetico, utile nel campo IoT, infatti, differentemente dalle altre schede Arduino, la MKR 1010 funziona a 3.3V.

2.4.2 Sensore ad ultrasuoni HC-SR04

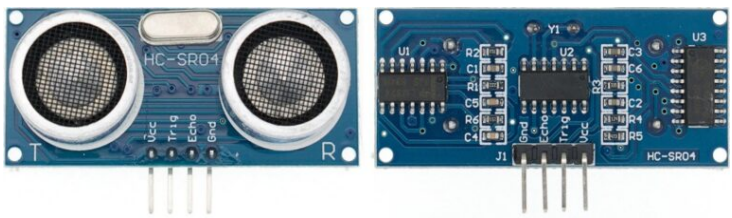


Figura 4: Sensore ad ultrasuoni HC-SR04

Il sensore di distanza ad ultrasuoni HC-SR04 [6], rappresentato in figura 4, è un sensore digitale utilizzato per rilevare la distanza di un oggetto tramite sonar. Il suo funzionamento consiste nel generare un treno di impulsi ad ultrasuoni: ne saranno emessi 8 a 40KHz quando il pin Trigger verrà portato per 10 microsecondi allo stato alto, successivamente le onde emesse viaggeranno attraverso l'aria e se presente un ostacolo rimbalzeranno tornando al pin Echo.

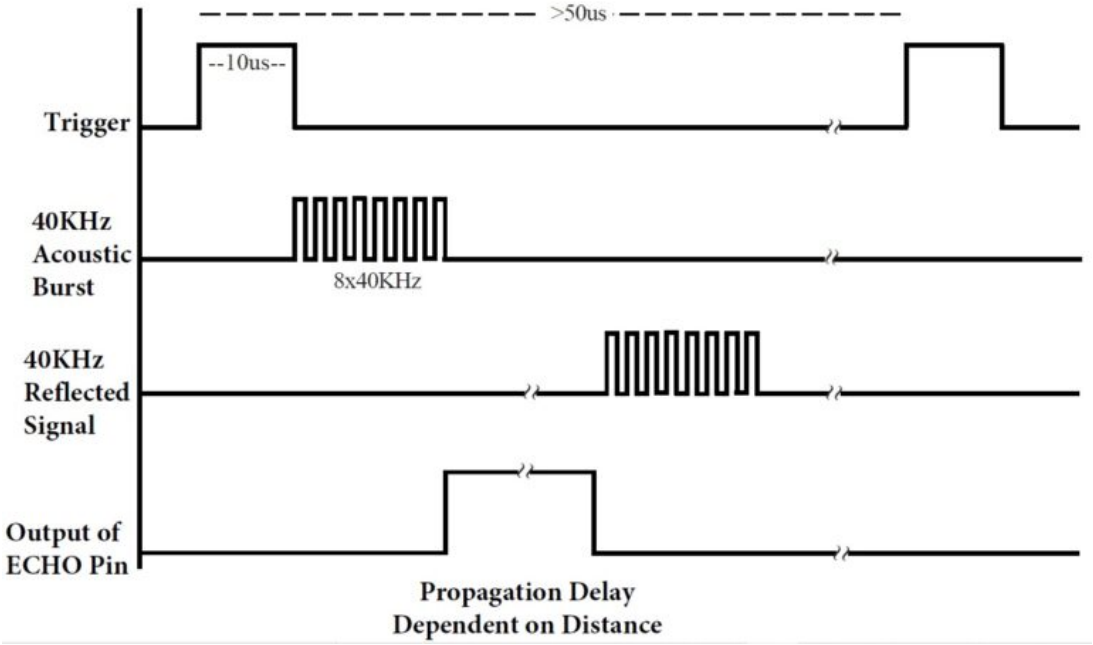


Figura 5: Diagramma temporale del funzionamento del sensore HC-SR04

In figura 5 viene riportato quanto descritto in precedenza. Andando a misurare la durata del segnale Echo si può risalire all'effettiva distanza dall'oggetto rilevato che ha causato la riflessione attraverso la legge oraria della velocità.

$$Distanza = \frac{Velocità\ del\ suono\ *\ Durata\ dell'impulso\ Echo}{2}$$

Poiché la distanza risulta essere doppia per via della riflessione contro l'oggetto si dovrà dividere per un fattore due la distanza. La velocità di propagazione dell'onda sonora dipenderà dalla densità del mezzo attraversato, che in questo caso è l'aria a cui va aggiunto un fattore legato alla temperatura dell'ambiente circostante (0.62*T, con T la temperatura attuale).
In arduino, la funzione per ottenere la distanza è la seguente:

```
int getDistance() {

  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);

  // Sets the triggerPin on HIGH state for 10 micro seconds
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(500000);
  digitalWrite(triggerPin, LOW);

  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);

  // Calculating the distance
  distance = duration * 0.034 / 2;

  return distance;

}
```

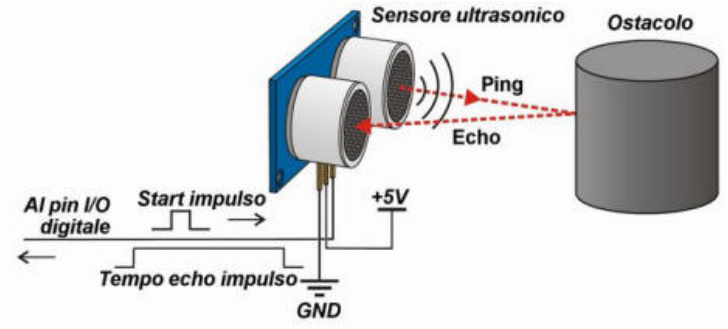


Figura 6: Schema di funzionamento del sensore HC-SR04

Il dispositivo opera ad una tensione di 5V e riesce ad effettuare una misura da 2 a 450 cm con un errore di precisione di circa 3mm e un range di apertura all'interno di 30°, condizione soddisfacente per i nostri obiettivi.

2.4.3 Sensore di temperatura ad infrarossi GY-906

Figura 7: Sensore di temperatura GY-906

In figura 7 è rappresentato il sensore di temperatura GY-906 [7], utilizzato all'interno del sistema embedded per la rilevazione della temperatura corporea dell'utilizzatore. Al suo interno è saldato il vero e proprio rilevatore di temperatura ad infrarossi, il sensore MLX90614 [8]. Esso opera ad una tensione di 3.3V, ma grazie ad un regolatore di tensione interno è possibile permettere il funzionamento del sensore sia con tensione di ingresso a 5V che a 3.3V.

Tramite la libreria dedicata sviluppata per Arduino [9] risulta molto semplice interagire con il sensore ed ottenere la temperatura corporea (espressa in gradi centigradi o in fahrenheit): in particolare, è sufficiente istanziare un oggetto *Adafruit_MLX90614* e chiamare il metodo *readObjectTempC()* per ottenere la misurazione richiesta, come mostrato di seguito:

```
Adafruit_MLX90614 mlx = Adafruit_MLX90614();
float temperature = mlx.readObjectTempC();
```

Tuttavia, per una maggiore attendibilità della lettura della temperatura, il dispositivo embedded calcola una media di 10 misurazioni lette in un secondo, ognuna presa ad una distanza di 100ms dall'altra, come mostrato nella funzione che segue:

```
void getTemperature(){
  temperature = 0;
  for (int i = 0; i < 10; i++) {
    temperature += mlx.readObjectTempC();
    delayMicroseconds(100000);
  }
  temperature /= 10;

}
```

Il sensore si avvale del protocollo di comunicazione I^2C per inviare le proprie misurazioni: i due pin caratteristici, SCL e SDA sono collegati ai medesimi pin della scheda MKR 1010, e al sensore di temperatura viene assegnato un indirizzo univoco necessario per instaurare correttamente la comunicazione.

2.4.4 Modulo di vibrazione

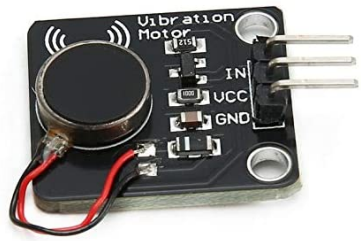


Figura 8: Modulo di vibrazione

Il modulo di vibrazione, qui raffigurato [10], è l'attuatore principale presente nel dispositivo: il suo compito è informare l'utente, tramite una vibrazione, che non si sta mantenendo il corretto distanziamento sociale. La vibrazione viene innescata modificando la tensione in entrata al pin IN, e raggiunge la sua intensità massima fornendo una tensione pari a 5V.

La funzione che gestisce il modulo di vibrazione è la seguente:

```
void vibrate(int duration, int interval, int times){
    for(int i=0;i<times;i++){
      digitalWrite(vibrationPin, HIGH);
      delayMicroseconds(duration*1000);
      digitalWrite(vibrationPin, LOW);
      delayMicroseconds(interval*1000);
    }
}
```

2.4.5 Altri componenti

Per una completa progettazione del sistema embedded sono stati introdotti altri componenti di minore rilevanza, quali un Led RGB e un pulsante, per permettere all'utente di avviare la misurazione della temperatura corporea e delle resistenze necessarie al corretto funzionamento dei componenti.

Il dispositivo finale è stato assemblato utilizzando come supporto una breadboard e come collegamenti dei jumper.

2.5 Librerie Arduino

Per garantire una buona affidabilità e rendere più semplice e leggibile il codice si sono utilizzate delle librerie fornite da Arduino e da programmatori di terze parti per gestire le funzionalità della scheda MKR 1010 e le interfacce I/O. Di seguito vengono brevemente esposte quelle principali:

- `<SR04.h>`,
  per configurare ed utilizzare i pin del sensore a ultrasuoni per misurare le distanze;
- `<Wire.h>`,
  che consente alla scheda MKR 1010 di comunicare tramite protocollo I2C con i sensori utilizzati;
- `<time.h>`,
  la quale viene usata per aggiungere funzionalità di cronometraggio ad Arduino con o senza hardware di cronometraggio esterno. Fornisce quindi la possibilità di sincronizzare manualmente i vari sensori ed eventualmente inserire dei ritardi per la corretta visualizzazione all'utente del funzionamento del dispositivo;

- `<EdgeEngine_library.h>`,

  che permette di usufruire delle funzionalità di Edge Engine come descritto precedentemente;

- `<SPI.h>`,

  Serial Peripheral Interface (SPI) è un protocollo di dati seriali sincrono utilizzato dai microcontrollori per comunicare rapidamente con uno o più dispositivi periferici su brevi distanze.

  Nel nostro progetto serve per la comunicazione master/slave tra la scheda MKR 1010 e i vari sensori;

- `<WiFiNINA.h>`,

  utilizzata per la connessione internet alla rete WiFi tramite il modulo WiFi NINA-W102 sulla scheda MKR 1010.

2.6 Flutter

Per visualizzare in modo facile ed intuitivo le misurazioni salvate sul cloud è necessario lo sviluppo di un'interfaccia grafica, la quale deve essere il più possibile indipendente dalla piattaforma sulla quale viene implementata.

Per fare ciò è possibile utilizzare Flutter, un toolkit per lo sviluppo di applicazioni e UI multi-piattaforma creato da Google [11]. Quello che contraddistingue Flutter dagli altri toolkit è la semplicità di scrittura del codice, basato sul linguaggio Dart, e la scelta di mettere a disposizione unicamente Widget per lo sviluppo di tutta l'interfaccia.

Dart è un linguaggio di programmazione sviluppato nel 2011 dall'azienda di Mountain View Google. Il suo obiettivo principale è fornire un linguaggio versatile e funzionale per lo sviluppo di web app e applicazioni native, sostituendo l'affermato linguaggio JavaScript.

Il sistema combinato Flutter engine+Dart consente un rapido sviluppo dell'interfaccia, grazie alla funzionalità cosiddetta "Hot Reload", che permette di applicare le modifiche appena implementate quasi istantaneamente senza bisogno di riavviare o reinstallare l'app.

La modularità di Flutter può venire arricchita includendo nel progetto librerie aggiuntive sviluppate sia dal team Google che da sviluppatori indipendenti: per aggiungerle al proprio progetto è necessario solamente indicare il nome e la versione del *pacchetto* nell'apposito file `pubspec.yaml`, e in automatico esso viene scaricato da internet e importato tra le librerie disponibili.

3 Sperimentazione e risultati

Il sistema embedded realizzato su breadboard, mostrato in figura 9 è composto, come precedentemente esposto, dalla scheda MKR WiFi 1010 che gestisce e monitora il sensore a ultrasuoni HC-SR04, il sensore di temperatura GY-906, il modulo di vibrazione, il led ed il pulsante. Il progetto Arduino è atto a svolgere l'ottenimento dei dati dai sensori, la loro elaborazione e il successivo invio al cloud Measurify.

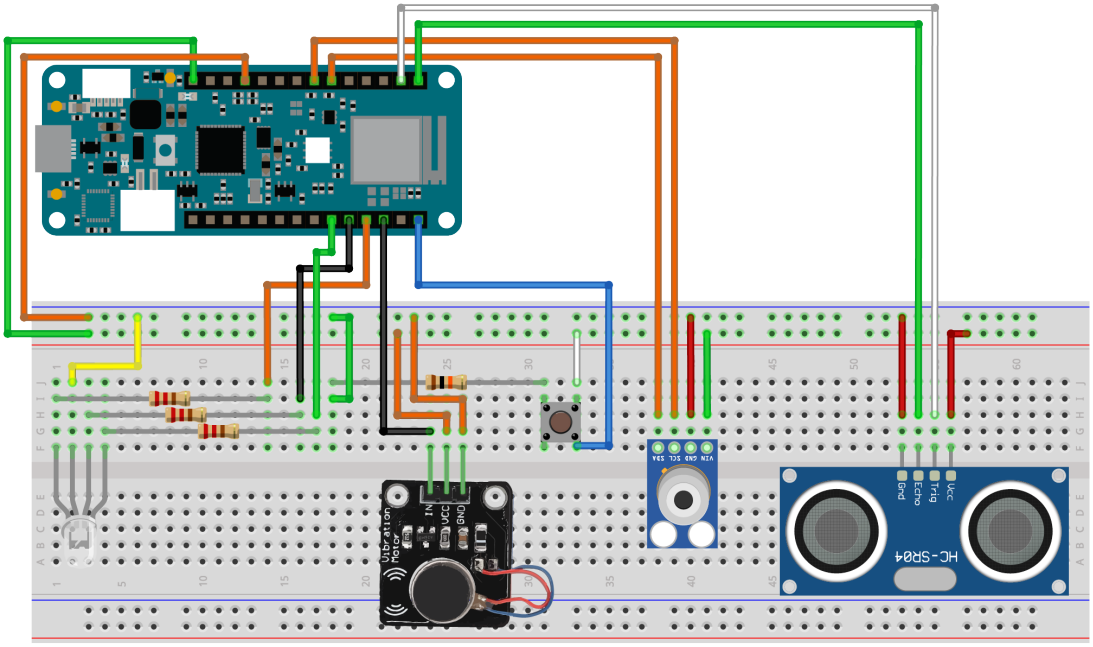


Figura 9: Schema del sistema embedded

3.1 Programmazione scheda Arduino

Dopo la descrizione delle funzioni necessarie per i singoli componenti del progetto e la realizzazione fisica su breadboard si è giunti allo sviluppo del codice per implementare le funzioni desiderate del nostro sistema. La programmazione è avvenuta nell'ambiente di sviluppo (IDE) di Arduino, utilizzando come linguaggio C++.
L'acquisizione dei dati da parte di Arduino viene gestita da uno sketch che implementa due diverse fasi di funzionamento: quella di setup e quella di loop.

3.1.1 Setup

In un primo momento vengono inizializzate tutte le variabili indispensabili per la connessione internet, la gestione dei sensori ed attuatori tramite la configurazione dei pin e delle chiamate HTTP.

```
/********Define features********/
sample* distanceSample = NULL;
sample* temperatureSample = NULL;
sample* alertSample = NULL;

/********Define ssid and password for WiFi connection********/
const char* ssidWifi = SECRET_SSID;
const char* passWifi = SECRET_PASS;

/********Define pin name for sensors and actuators********/
#define triggerPin 7
#define echoPin 6
#define buttonPin 5
#define ledPin 4
#define vibrationPin 3
#define redLedPin 2
#define greenLedPin 1
```

```
#define blueLedPin 0

/********Define variables for distance sensor********/
long duration;
int distance;
int delayCounter = 0;

/********Define variables for Edge Engine********/
edgine* Edge;
connection* Connection;
vector<sample*> samples;
int connection_state;
```

Immediatamente dopo il dispositivo inizializza una nuova connessione WiFi utilizzando come SSID e come Password le credenziali di accesso alla rete locale. Per la procedura di connessione viene utilizzato un apposito modulo inserito all'interno di Edge Edgine.

Successivamente vengono definiti gli stati delle porte I/O della scheda necessarie per l'interfacciamento con sensori ed attuatori, viene inizializzata la connessione seriale per poter comunicare con la console e viene iniziata la comunicazione tramite protocollo I^2C per il sensore di temperatura.

La fase di setup procede con la definizione delle *opzioni* necessarie per stabilire una connessione con Measurify, come ad esempio l'URL del server, le credenziali per il login e la *thing* e il *device* in uso. Una volta impostato tutto, si può finalmente procedere con il collegamento al cloud.

Viene riportato di seguito il codice per l'inizializzazione della parte per la comunicazione con Edge Engine e la verifica del corretto funzionamento dei sensori:

```
//start Edge engine
 Edge=edgine::getInstance();
 Edge->init(opts);

 /********Check sensors ********/
 if(getDistance() != 0) Serial.print("Sensore di distanza correttamente funzionante");
 else Serial.print("Errore nel sensore di distanza!");

 if (mlx.readObjectTempC() < 45)
   Serial.print("Sensore di temperatura correttamente funzionante");
 else Serial.print("Errore nel sensore di temperatura!");
```

3.1.2 Acquisizione dei dati

All'interno del ciclo di loop vengono periodicamente eseguite le istruzioni fondamentali che determinano le funzionalità del nostro dispositivo, ripetute durante tutto il periodo di funzionamento del device.
Alla pressione dell'apposito pulsante del dispositivo è generato un interrupt che chiama la funzione getTemperature(). Qui viene definito un nuovo campione (sample) di temperatura e tramite Edge Engine viene inoltre memorizzata la data attuale di rilevamento. Se la temperatura rimane al di sotto della soglia (impostata di default a 37°), il led RGB viene impostato sul colore verde, altrimenti viene mostrato il colore rosso insieme all'attivazione del modulo di vibrazione che richiama l'attenzione dell'utente. Si riporta il codice di quanto descritto.

```
void getTemperature() {
  Serial.println("Measuring temperature...");
  temperature = 0;
  for (int i = 0; i < 10; i++) {
    temperature += mlx.readObjectTempC();
    delayMicroseconds(100000);
  }
  temperature /= 10;
  Serial.print("Object temperature = ");
  Serial.print(temperature);
  Serial.println("*C");
  Serial.println();

  temperatureSample = new sample("temperature");
  temperatureSample->startDate = Edge->Api->getActualDate();
  temperatureSample->endDate = temperatureSample->startDate;
  temperatureSample->value = (temperature);
  samples.push_back(temperatureSample);
  if (temperature >= 37) {
    setLed(255, 0, 0);
```

```
    vibrate(500, 500, 2);
    delayMicroseconds(1500000);
  }
  else setLed(0, 255, 0);
}
```

Lo stesso procedimento viene svolto per il sensore di distanza che effettua una misura ogni 10 secondi e, come per la temperatura, interagisce con l'utente tramite led e vibrazione, impostando il led con colore rosso se si tratta di una misura sotto la soglia, o impostandolo con colore blu e generando una vibrazione se il valore supera la soglia desiderata:

```
  delay(10000);
  distanceSample = new sample("distance");
  distanceSample->startDate = Edge->Api->getActualDate();
  distanceSample->endDate = distanceSample->startDate;
  distanceSample->value = (getDistance());
  samples.push_back(distanceSample);

  alertSample = new sample("alert");
  alertSample->startDate = Edge->Api->getActualDate();
  alertSample->endDate = alertSample->startDate;
  alertSample->value = distanceSample->value;
  samples.push_back(alertSample);

  distance = distanceSample->value;
  if (distance < 100) {
    setLed(255, 0, 0);
    vibrate(1000, 1000, 2);
  }
  else setLed(0, 0, 255);
```

Il comportamento del ciclo di loop del dispositivo si può illustrare come segue:

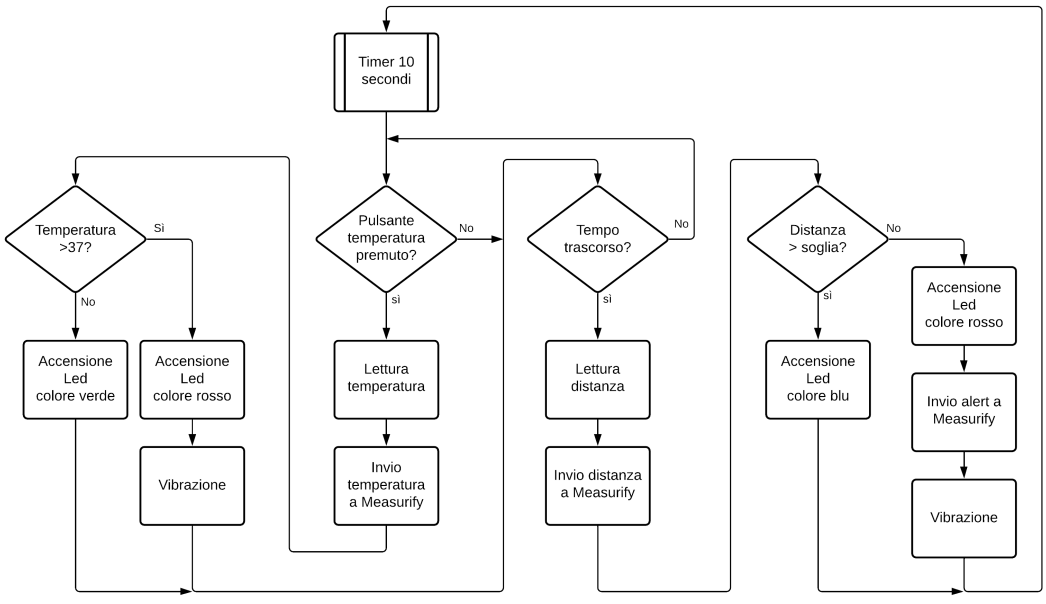


Figura 10: Diagramma di flusso del ciclo loop

3.1.3 Invio dei dati al cloud

Dopo aver effettuato le misurazioni, i dati vengono inviati al cloud tramite connessione internet.
Come già menzionato, per effettuare le varie POST la scheda MKR 1010 deve disporre di un token di autorizzazione ottenibili durante la procedura di login.

Per pubblicare i dati di una feature su Measurify è necessario prima di tutto inizializzare un nuovo *sample*, ossia un campione. Ad esso devono essere associati la feature corrispondente, la data di inizio e di fine misurazione e il valore del dato ottenuto. Ogni sample viene poi inserito all'interno di un vettore

di campioni, chiamato ”*samples*” che raccoglie tutti i *sample* prodotti durante un unico ciclo di iterazione del loop del dispositivo.

Terminata la creazione di ogni campione di misurazione, l’apposito comando di Edge Engine

```
Edge->evaluate(samples);
```

permette di pubblicare l’intero array di samples sul cloud, passando prima attraverso i vari scripts descritti sia localmente che su Measurify.
Una volta inviate le misurazioni è necessario cancellare e successivamente reinizializzare tutti i *sample* creati così da permettere un corretto svolgimento di una nuova iterazione del loop principale.

3.2 Applicazione Smartphone

Il modo più semplice e veloce per monitorare e visualizzare i dati acquisiti dal sistema è sicuramente quello di realizzare un’applicazione dedicata, a cui si può accedere tramite smartphone con una connessione internet.

3.2.1 Interfaccia Utente

L’app è composta da quattro schermate principali, navigabili attraverso una *BottomNavBar*, oltre alla schermata di benvenuto e a quella di login.

La home dell’applicazione, ossia la prima schermata principale, mostrata in figura 11 presenta al suo interno tre blocchi:

- il blocco in alto con la rappresentazione della media delle distanze mantenute durante la giornata divisa in fasce orarie (dalle 8.00 alla mezzanotte);
- il blocco in basso a sinistra con la visualizzazione delle ultime 7 distanze rilevate;
- il blocco in basso a destra, analogo al precedente, con la visualizzazione delle ultime 7 temperature rilevate.

In alto a destra è inoltre presente un pulsante per effettuare un nuovo fetch di tutte le misurazioni presenti sul server cloud, incluse quelle nuove.

Figura 11: Homepage dell’applicazione

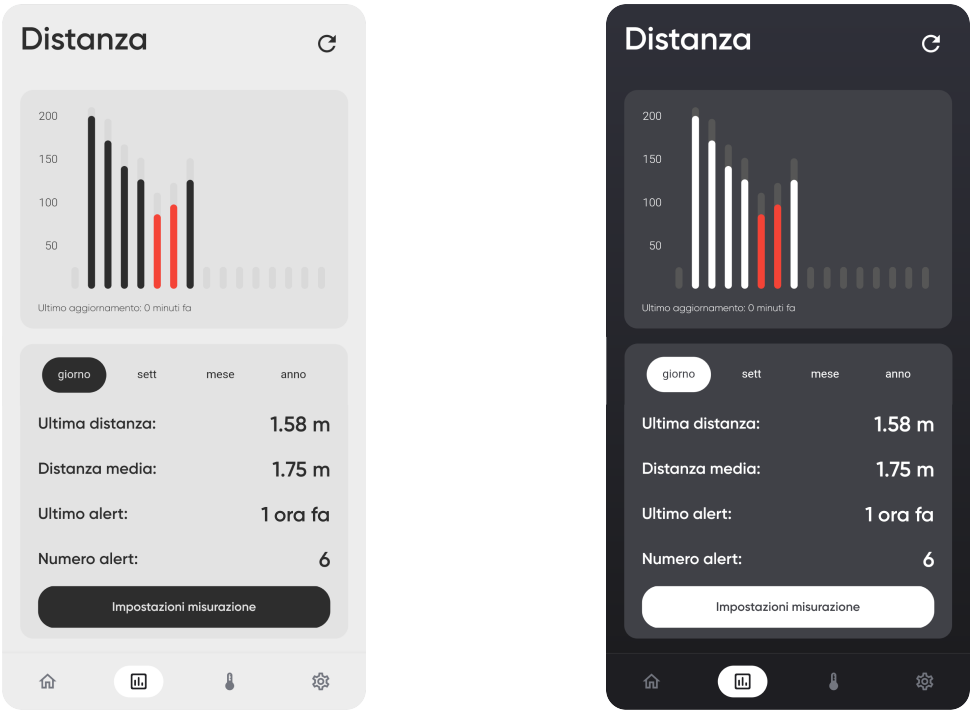
Il caricamento dell'interfaccia della Homepage viene affidato ad un FutureBuilder, ossia un costruttore grafico che si comporta in modo differente mentre si è in attesa di una variabile di tipo Future e una volta ricevuta. Quando viene generata l'UI della home il builder mostra una grafica comprensiva dei soli *container* della schermata, senza alcun dato all'interno, mentre appena vengono ricevute e memorizzate tutte le misurazioni presenti su Measurify il costruttore modifica l'interfaccia e aggiunge tutte le informazioni necessarie. Questo particolare Widget risulta essere molto utile per realizzare una User Interface intuitiva che mostri all'utilizzatore che l'applicazione sta eseguendo delle task in background.

Cliccando sui vari blocchi presenti in questa schermata, o facendo tap nella barra di navigazione è possibile visualizzare le pagine relative al dettaglio delle misurazioni di distanza e temperatura.
Per quanto concerne la distanza, il display mostra in alto lo stesso grafico a barre presente nella homepage, e in basso visualizza statistiche utili relative allo specifico periodo selezionato dallo slider, scegliendo tra giorno, settimana, mese e anno.
Le distanze e le medie inferiori alla soglia impostata dall'utente vengono visualizzate di colore rosso, per distinguerle facilmente da tutte le altre misurazioni superiori alla distanza minima.

La pagina della temperatura, analoga alla precedente, visualizza la temperatura attuale (ossia l'ultima inviata al cloud), una lista della ultime nove temperature rilevate e alcune statistiche utili quali il numero di misurazioni effettuate in un determinato lasso di tempo e la temperatura media, oltre al tempo trascorso dall'ultimo invio.

Tramite la barra di navigazione è possibile accedere all'ultima pagina, quella relativa alle impostazioni dell'app. Qui è possibile gestire:

- Il **profilo utente**, per modificare il proprio nome e le credenziali di accesso all'app;
- L' **aggiornamento automatico**, scegliendo se eseguire un fetch completo dei dati ogni volta che si entra nella homepage o se eseguirlo solo all'avvio dell'applicazione e alla pressione dell'apposito pulsante;
- La **distanza soglia**, indicando da quale distanza una misurazione può essere considerata come un alert ed essere visualizzato come tale, e aggiornando il valore di soglia nello script su Measurify;
- Il **tema dell'applicazione**, scegliendo se utilizzare il tema chiaro di default dell'app oppure il tema scuro, come mostrato in figura 12 rispettivamente nella sotto-figura 12a e nella sotto-figura 12b ;



(a) Tema chiaro

(b) Tema scuro

Figura 12: Differenza fra tema chiaro e tema scuro dell'applicazione

- Le **notifiche** inviate tramite Firebase Cloud Messaging al dispositivo.

Nella stessa schermata è inoltre possibile visualizzare alcune informazioni relative all'applicazione stessa ed effettuare il logout, ritornato alla schermata di accesso.

3.2.2 Funzionamento dell'applicazione

L'applicazione, una volta inserite le credenziali corrette nella schermata di login (differenti da username, password e tenant) effettua l'accesso su Measurify e salva il token di autenticazione. Immediatamente dopo inizia a fare varie richieste GET sulle tre features disponibili, e salva sotto forma di liste tutte le misurazioni ottenute. A questo punto il sistema ha a disposizione tutti i dati necessari, pertanto la grafica viene aggiornata e tutte le informazioni vengono mostrate all'utente.

Nelle tre schermate di visualizzazione delle misurazioni, home, pagina distanza e pagina temperatura, è necessario elaborare le varie *measurements* per potere realizzare statistiche e mostrare i grafici con le medie.

Per le statistiche l'applicazione si limita a effettuare semplici calcoli per recuperare le informazioni richieste, come il numero di alert ricevuti in un determinato lasso di tempo o il tempo trascorso dall'ultima misura di temperatura.
Per quanto riguarda invece i grafici, l'app filtra tutte le misure in base al periodo selezionato (di default è impostata la visualizzazione limitata al giorno corrente) e in base all'orario di misurazione (impostato dalle 8.00 alla mezzanotte). Dopo di che i dati vengono suddivisi in base alla fascia oraria in cui sono stati misurati, finendo in una delle 16 fasce possibili. Per ogni ora viene poi calcolata una media.

Le impostazioni vengono gestite attraverso le *SharedPreferences*, una feature di Android che consente di memorizzare e gestire una quantità limitata di coppie chiave-valore indipendentemente dalla propria app. Il file contenente le impostazioni è interamente gestito dal framework Android, e vengono messi a disposizione semplici metodi per accedere in lettura o in scrittura a tale file.

All'avvio dell'applicazione vengono effettuate diverse richieste all'API SharedPreferences per recuperare le variabili impostate nella schermata delle impostazioni, come ad esempio il tema dell'applicazione, e genera una UI differente a seconda del valore di tali variabili. Ogni volta che invece vengono alterate delle impostazioni, i nuovi valori vengono passati a SharedPreferences che in automatico gestisce il cambiamento.

L'applicazione implementa inoltre alcuni timer utili a tenere traccia del tempo trascorso dall'ultimo ottenimento dei dati presenti su Measurify e della distanza temporale dall'ultimo invio di un *alert* o di una temperatura. La grafica dell'interfaccia viene aggiornata in automatico per mostrare la miglior risoluzione temporale possibile, scegliendo tra minuti, ore o giorni.
Ogni volta che viene effettuato un nuovo fetch delle misurazioni disponibili sul cloud tutti i timer vengono resettati e fatti ripartire.

3.2.3 Gestione delle notifiche

L'applicazione è dotata di un sistema di notifiche push basato su Firebase Cloud Messaging (FCM) [12], che notificano l'utente ogni volta che viene effettuata la POST di un *alert* su Measurify.
Per poter ricevere le notifiche è necessario effetuare una *subscription* al server cloud, specificando thing, device e token Firebase. Il token Firebase è una stringa speciale che viene generata da FCM all'interno di ogni singola applicazione, e definisce in modo univoco l'app che riceverà le notifiche.

L'applicazione, ogni volta che viene effettuata una login, controlla tutte le sottoscrizioni presenti su Measurify e verifica che ne sia presente una con il token ad essa associato: se non è presente alcuna subscription, in automatico ne viene creata una, in modo da rendere completamente automatizzato e *invisibile* all'utente il processo di inizializzazione della ricezione delle notifiche.

4 Contributo personale e considerazioni conclusive

L'obiettivo della tesi è stato sviluppare un dispositivo fisico connesso al cloud tramite internet che permettesse all'utente di monitorare la distanza dalle persone intorno a lui, e la propria temperatura corporea.

Durante lo sviluppo del progetto, per utilizzare la libreria Edge Engine con la scheda MKR 1010 è stato necessario effettuare un porting per consentirne l'uso su una piattaforma diversa da quella inizialmente concepita, anche in vista di futuri utilizzatori.
Si è quindi scelto di riadattare il codice per l'utilizzo delle funzionalità implementate dalla libreria su Arduino anziché riscrivere il codice sorgente da principio, operazione che avrebbe richiesto assai più tempo e risorse.

Per fare ciò è stato necessario prima di tutto comprendere quali blocchi di codice fossero già compatibili con il nuovo sistema e quali invece risultassero inconciliabili. In particolare, la scheda Arduino a disposizione integra un modulo radio u-blox NINA-W102, mentre i sistemi precedentemente testati utilizzavano altri moduli ESP32, perciò tutti i tentativi di comunicazione tra la scheda e il server tramite protocollo HTTP fallivano istantaneamente.

Si è dovuto quindi implementare un nuovo sistema di gestione delle richieste HTTP e un nuovo sistema per poter leggere la risposta ricevuta dopo tali richieste, facendo attenzione a tenere conto di tutti i vincoli richiesti dall'architettura in uso. Inoltre molte delle funzioni sviluppate per sistemi basati su ESP32 risultavano incompatibili con Arduino, come ad esempio il calcolo dell'ora attuale, poiché il conteggio degli istanti temporali all'interno dei due sistemi risultava completamente differente, o l'ottenimento del token di autorizzazione, problema dovuto all'utilizzo di nuove librerie per comunicare tramite HTTP che interpretavano il corpo della risposta in modo differente.

Un altro aspetto che abbiamo dovuto affrontare è stato il passaggio dal protocollo HTTP ad HTTPS. Nel dettaglio la precedente libreria di terze parti utilizzate per la comunicazione non supportava HTTPS, pertanto è stato necessario sostituirla con un'altra libreria e modificare alcune delle funzioni già precedentemente rivisitate durante la prima fase del porting.

Il sistema sviluppato presenta alcune limitazioni progettuali attinenti a degli aspetti su cui non ci si è soffermati. Sebbene il dispositivo sia stato ideato per essere integrato all'interno di un oggetto relativamente piccolo e pratico (già esistente o non, come per esempio un'audioguida), non si è approfondita né la questione dello spazio utilizzato dall'insieme di tutti i componenti né quella della progettazione di un rivestimento di supporto, possibili sviluppi futuri che andrebbero a migliorare la qualità del dispositivo finale e del servizio offerto.

Un altro aspetto su cui non ci si è soffermati è l'alimentazione del dispositivo, che attualmente viene alimentato tramite PC. Immaginando un utilizzo centralizzato del servizio, con un insieme di molti dispositivi in uso quotidianamente, una possibile implementazione futura potrebbe essere la creazione di una specifica base fissa atta alla ricarica (tramite appositi terminali o tramite induzione) di una batteria posta all'interno del dispositivo.

5 Riferimenti bibliografici

[1] Statistica IoT. URL https://www.statista.com/statistics/471264/iot-number-of-connected devices-worldwide/.

[2] Measurify. URL https://measurify.org.

[3] Postman. URL https://www.postman.com.

[4] Edge Engine. URL https://github.com/measurify/edge.

[5] Arduino MKR WiFi 1010. URL https://store.arduino.cc/arduino-mkr-wifi-1010.

[6] HC-SR04. URL https://www.sparkfun.com/products/15569.

[7] GY-906. URL https://protosupplies.com/product/gy-906-mlx90614-non-contact-precision -thermometer-module.

[8] MLX90614, . URL https://www.sparkfun.com/products/9570.

[9] Libreria Adafruit_MLX90614, . URL https://github.com/adafruit/Adafruit-MLX90614-Library.

[10] Modulo di vibrazione. URL https://grobotronics.com/arduino-vibration-motor-module.html.

[11] Flutter. URL https://flutter.dev.

[12] Firebase Cloud Messaging. URL https://firebase.google.com/docs/cloud-messaging.