



# 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

14 Giugno 2022

Anno accademico 2020/2021

## **Progetto e realizzazione di uno strumento per il monitoraggio di variabili fisiologiche**

Candidato: Andrea Caliendo

Relatore: Prof. Riccardo Berta

Correlatore: Dott. Luca Lazzaroni

## Sommario

La presente tesi ha lo scopo di progettare e realizzare un dispositivo in grado di misurare pressione sanguinea, frequenza cardiaca e saturazione dell'ossigeno per il monitoraggio dei parametri fisiologici di un utente. I campi di applicazione sono molteplici, dal monitoraggio remoto di pazienti all'attività sportiva. La progettazione è basata sulla scheda Arduino Nano 33 IoT e sul sensore Max32664-D che misura la pressione sanguinea, la frequenza cardiaca e la saturazione dell'ossigeno.

La comunicazione è basata su connessione WI-FI e bluetooth. I dati raccolti dalla scheda vengono inviati al cloud, tramite le API del framework Measurify, dove sono controllati, memorizzati ed eventualmente elaborati. Infine, una applicazione grafica per smartphone realizzata con il framework Flutter permette all'utente finale di accedere ai dati memorizzati e di visualizzarli.

## Indice

1.	INTRODUZIONE .....	4
2.	METODI E STRUMENTI UTILIZZATI .....	6
2.1.	ARDUINO NANO 33 IOT.....	6
2.2.	SENSORE MAX32664-D.....	7
2.3.	MEASURIFY .....	8
2.4.	POSTMAN .....	9
2.5.	EDGE ENGINE .....	10
2.6.	FLUTTER.....	10
3.	SPERIMENTAZIONE E RISULTATI .....	12
3.1.	ACQUISIZIONE DATI SULLA SCHEDA ARDUINO .....	12
3.2.	INVIO DEI DATI AL CLOUD .....	15
3.3.	FLUTTER.....	16
4.	CONTRIBUTO PERSONALE E CONSIDERAZIONI CONCLUSIVE .....	18
5.	RIFERIMENTI BIBLIOGRAFICI .....	19

## 1. Introduzione

Negli ultimi anni, internet ha subito una profonda evoluzione arrivando al cosiddetto **Internet of Things** (IoT) o Internet delle cose. Si stima, secondo la *Cisco System*, la nascita dell'IoT intorno agli anni 2008/2009, quando il numero di dispositivi connessi ad internet superò il numero degli esseri umani, anni infatti in cui è esploso il mercato degli smartphone.

Anche se, in realtà si è incominciato a parlare di IoT alla fine degli anni 90. [1]

Il concetto di Internet of Things si è diffuso maggiormente a livello pubblico grazie allo sviluppo di questa tecnologia specialmente nei settori della domotica, dell'automotive, fino in questi ultimi tempi al settore del e-health.

Con "*Internet of things*" si indica generalmente un insieme di tecnologie che permettono di collegare a Internet qualsiasi tipo di apparato. Lo scopo di queste diverse progettazioni è sostanzialmente quello di monitorare e controllare dei dati provenienti da diverse fonti, per poi trasferire le relative informazioni, ed infine svolgere azioni conseguenti o diversi utilizzi.

"L'*Internet of things*" ha l'obiettivo di rendere Smart (digitale) qualsiasi oggetto, a condizione che tale oggetto abbia a disposizione la possibilità di connettersi ad internet; quindi, che possa utilizzare un indirizzo IP.

Per ottenere tale risultato, vengono utilizzati dei microcontrollori e dei sensori per poter collegare i vari oggetti e salvare poi sul cloud le rilevazioni effettuate.

Ormai viviamo a stretto contatto con l'IoT, vista la loro ampia utilizzazione e utilità:

Si pensi ad esempio alle cosiddette "Smart Home", in cui troviamo lampadine, frigoriferi, termosifoni, sistemi di videosorveglianza ed altri apparecchi a servizio della casa, che sono collegati tutti tra loro e muniti di connessione ad internet, consentendo al proprietario la gestione da remoto tramite un'unica interfaccia grafica.

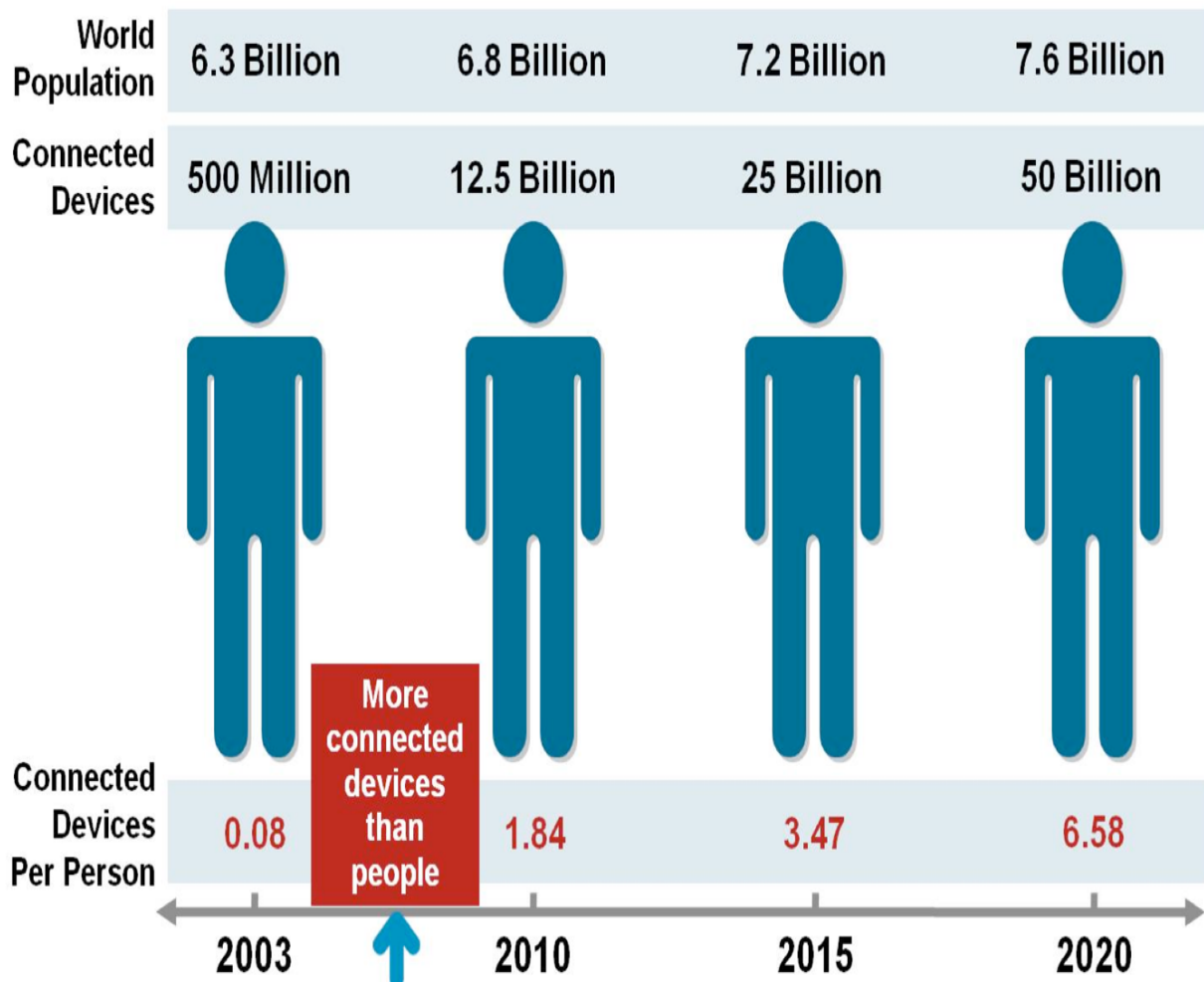


Figura 1 Numero di dispositivi connessi in relazione agli esseri umani

Uno dei tanti possibili utilizzi di quel sistema si può individuare nell'ambito del cosiddetto e-health. In particolare, nel settore della prevenzione di alcune patologie tra le quali quelle legate all'apparato cardio circolatorio, con particolare attenzione al cuore e al battito cardiaco. Si può pensare anche al monitoraggio di determinati individui (ad esempio atleti) in specifiche situazioni di sforzo fisico.

In particolare, nella diagnostica delle malattie cardiache, si deve tener conto che per molti pazienti, soprattutto anziani, l'utilizzo di strumentazioni e-health può rappresentare uno strumento fondamentale e assai utile sia per il paziente che per il medico curante. Difatti, attraverso questi apparecchi, si offre la possibilità di trasmettere i propri dati fisiologici direttamente al medico senza la necessità di spostamento (causa di possibili complicazioni per il paziente e/o ritardi nella diagnosi, e quindi nella cura) e con la possibilità di memorizzare più facilmente le informazioni del paziente.

Questa tesi intende, perciò, cercare di fornire una possibile risposta a queste problematiche.

Lo scopo della tesi, infatti, è quello di misurare tramite il sensore MAX32664-D, la frequenza cardiaca, la saturazione dell'ossigeno nel sangue e di stimare la pressione sistolica (pressione minima) e la pressione diastolica (pressione massima) di un soggetto (paziente), affinché questo possa essere, tranquillamente, monitorato continuamente anche se si trova (ad esempio) a casa sua.

Ciò può avvenire perché il sensore viene collegato ad una scheda Arduino Nano 33 IoT, che ha la possibilità di inviare i dati al cloud Measurify attraverso il collegamento Internet, collegamento reso possibile grazie al modulo WIFI-Nina. (inserito nella scheda stessa)

L'utente, che in questo caso può essere sia il paziente che il medico curante, potrà così analizzare i dati raccolti dall'app sviluppata, che inoltre manterrà in memoria tali misurazioni e lo storico di quel determinato paziente.

Questo progetto avrebbe il significativo vantaggio di evitare, per soggetti già fragili, come cardiopatici e altro, lo stress di recarsi a ripetute visite di controllo. Comporterebbe altresì i minori accessi alle strutture ospedaliere e/o studi medici, con conseguenti risparmi in termini di tempo e di costi a carico del servizio sanitario pubblico, garantendo in fine un minore stress psicologico nei pazienti che verrebbero così monitorati durante la loro normale vita.

Oppure, uscendo dal campo diagnostico terapeutico vero e proprio, si potrebbero monitorare gli atleti durante gli allenamenti per ottenere le risposte del corpo alle diverse sollecitazioni (sforzi intensi, prolungati o altro).

## 2. Metodi e strumenti utilizzati

### 2.1. Arduino Nano 33 IoT

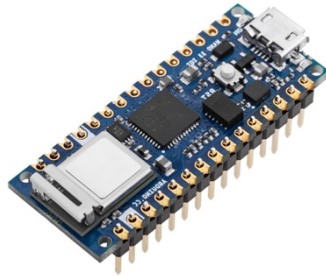


Figura 2 Arduino Nano 33 IoT

Per sviluppare il progetto di tesi è stata utilizzata la scheda Arduino Nano 33 IoT.

Questa scheda ha dimensioni miniaturizzate, contiene un processore Cortex M0+ SAMD21, un modulo WiFi e bluetooth basato su ESP32 e un chip crittografico in grado di memorizzare in modo sicuro certificati e chiavi pre-condivise.

La scheda non disponendo di un caricabatterie deve essere alimentata tramite USB, come la maggior parte delle schede della gamma Nano.

I pin totali sono 15, come si può vedere nella figura 3. Il funzionamento della scheda è garantito dal pin di voltaggio a 3.3V, mentre quello da 5V non fornendo tensione viene collegato (tramite ponte) direttamente all'ingresso USB. [2]

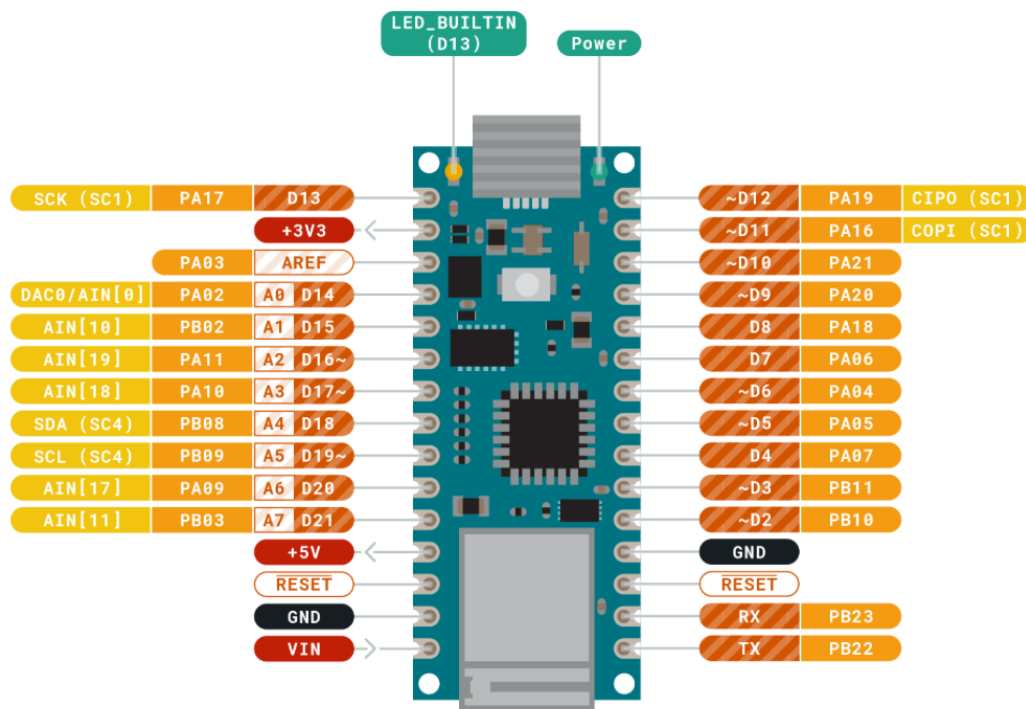


Figura 3 Pin out Arduino Nano 33 IoT

## 2.2. Sensore MAX32664-D



Figura 4 Sensore MAX32664-D

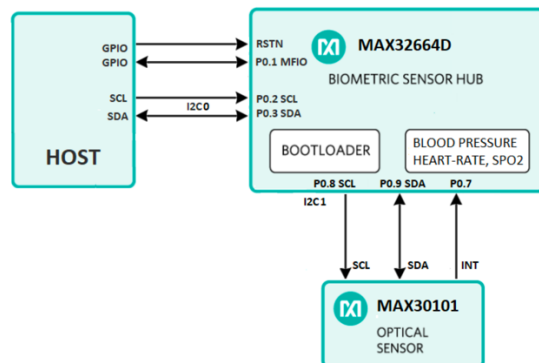


Figura 5 Schema dell'architettura del sensore

Il sensore MAX32664D, rappresentato in figura 4, è una variante della famiglia di sensori-hub MAX32664 che consente di acquisire sia dati grezzi, nonché quelli calcolati, sulla pressione sanguigna sistolica e diastolica, sulla SpO2 e sulla frequenza cardiaca attraverso il contatto con le dita.

Il calcolo della saturazione sanguigna di ogni individuo viene effettuato attraverso la misurazione dell'emoglobina (Hb) che è una proteina che trasporta l'ossigeno nei globuli rossi.

Le due forme principali di Hb presenti nel sangue sono l'emoglobina ossigenata (ossi-emoglobina, HbO2) e l'emoglobina deossigenata (deossi-emoglobina, RHb).

La SpO2 è la misura della saturazione dell'ossigeno capillare periferico. Più precisamente, la SpO2 è una stima della quantità di ossigeno nel sangue capillare, ottenuta come una percentuale della quantità di emoglobina ossigenata rispetto all'emoglobina totale, ed espressa come segue:

$$SpO_2 = 100 \times \frac{C[HbO_2]}{C[HbO_2] + C[RHb]}$$

dove  $C[HbO_2]$  e  $C[RHb]$  sono le concentrazioni di HbO2 e RHb

La pulsossimetria è uno strumento utilizzato per la misurazione non invasiva dell'ossigenazione del sangue (SpO2). La pulsossimetria si basa su due principi: la modulazione della luce trasmessa dall'assorbimento del sangue arterioso pulsatile e le diverse caratteristiche di assorbimento di HbO2 e RHb per le diverse lunghezze d'onda.

La pulsossimetria può essere classificata come trasmissiva o riflessiva:

- La pulsossimetria trasmissiva è quella in cui il fotodiode e il LED sono posizionati su lati opposti del corpo umano (ad esempio, il dito). Il tessuto corporeo assorbe parte della luce e il fotodiode raccoglie la luce residua che attraversa il corpo.
- La pulsossimetria a riflessione si ha quando il fotodiode e il LED si trovano sullo stesso lato. Il fotodiode raccoglie la luce riflessa da varie profondità sotto la pelle.

Il sensore in esame utilizza quest'ultima soluzione.

Il sangue arterioso pulsante assorbe e modula la luce incidente che attraversa il tessuto e forma il segnale fotopletiografico (PPG). La componente AC dei segnali PPG rappresenta la luce assorbita dal sangue arterioso pulsante.

Questa componente AC è sovrapposta a un segnale DC che cattura gli effetti della luce assorbita da altri componenti del sangue e del tessuto (ad esempio, sangue venoso e capillare, osso, acqua, ecc.) Il rapporto tra il segnale AC e il livello DC è chiamato indice di perfusione (PI).

Il sensore prima di poter essere utilizzato, deve essere calibrato. Per farlo bisogna tenere conto del fatto che la misurazione del SpO2 si ottiene come segue:

$$SpO_2 = aR^2 + bR + c$$

dove R è determinato dalla seguente equazione:

$$R = \frac{AC_{red}/DC_{red}}{AC_{ired}/DC_{ired}}$$

In cui a, b e c sono i coefficienti di calibrazione. [3]

La prima operazione che esegue l'algoritmo del sensore è quella di caricare i coefficienti di calibrazione e iniziare il processo, prima di riuscire a misurare le diverse variabili.

I coefficienti utilizzati sono quelli di default, ossia:

	a	b	c
MAX30101	1.5958422	-34.6596622	112.6898759

## 2.3. Measurify

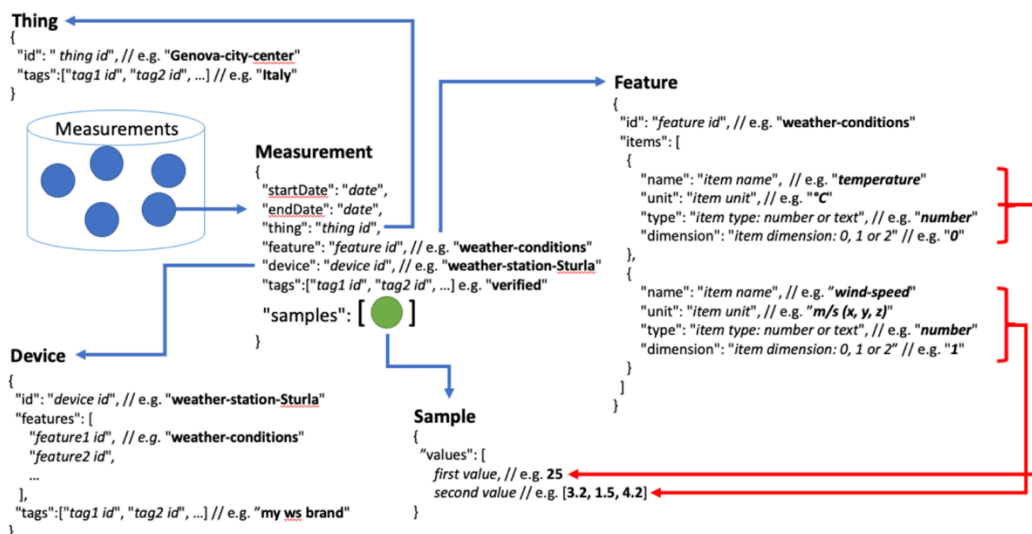
Dopo aver parlato dei componenti che si occupano della raccolta dei dati, un altro importante aspetto da trattare è quello del server Measurify che si utilizza per memorizzare i dati e successivamente elaborarli.

Measurify è un API di tipo Restful, sviluppata dal laboratorio Elios Lab, del Università di Genova.

È una API basata su cloud, che serve per la gestione e la elaborazione di dati provenienti da vari ambienti come il campo IoT. [4]

Come si può osservare dalla figura 6, questa si concentra principalmente su:

- **Thing:** si tratta di un oggetto per il quale si sta operando una misurazione. Nel nostro caso le things sono due e simulano due persone di cui dobbiamo misurare le variabili fisiologiche.
- **Feature:** si tratta di quale grandezza fisica si sta misurando (la temperatura, l'umidità, il peso, ecc.) tramite un dispositivo
- **Device:** si tratta di un dispositivo o un sensore che può misurare una certa grandezza fisica (feature) su un determinato oggetto (thing).
- **Measurement:** si tratta di una misura effettuata da un dispositivo (device) per una certa grandezza fisica (feature) su un determinato oggetto (thing).





Nel presente progetto, abbiamo detto che la *things* corrisponde alla simulazione di due persone ("mario.rossi" e "giuseppe.bianchi") che devono tenere sotto controllo le proprie misurazioni, la *feature*, come raffigurato nella figura 7, che si chiama "*physiological-state*" ed è l'unica feature del progetto e contiene tutte quattro le variabili fisiche che il sensore misura: la pressione sistolica, la pressione diastolica, la frequenza cardiaca e la saturazione.

```
"_id": "physiological-state",
"items": [
  {
    "dimension": 0,
    "type": "number",
    "name": "sys",
    "unit": "mmHg"
  },
  {
    "dimension": 0,
    "type": "number",
    "name": "dia",
    "unit": "mmHg"
  },
  {
    "dimension": 0,
    "type": "number",
    "name": "hr",
    "unit": "bpm"
  },
  {
    "dimension": 0,
    "type": "number",
    "name": "spo2",
    "unit": "%"
  }
]
```

Figura 7 Feature

Il *device* si chiama "*body-monitor*" e rappresenta il sensore MAX32664-D che raccoglie le misurazioni. Per quanto riguarda il measurement contiene un campo *samples* che, tramite un *array*, salva tutte e quattro le misurazioni ogni volta che il sensore è in azione.

Le risorse appena citate, che rappresentano degli oggetti JSON sul Measurify. Per poter dialogare con queste risorse, visualizzando e salvando dei dati, bisogna utilizzare le richieste HTTP, che avvengono tra client (in questo caso il nostro microcontrollore) e il server (Measurify).

Per poter iniziare a fare queste richieste bisogna loggarsi con il server, facendo una POST di login. La richiesta è così gestita:

POST <https://students.measurify.org/v1/login>

E nel body della richiesta deve essere specificato username, password e tenant come nella figura

```
1 {
2   ... "username": "body-monitor-username",
3   ... "password": "body-monitor-password",
4   ... "tenant": "body-monitor"
5 }
```

Figura 8 Body della richiesta "POST login"

La risposta a questa richiesta, se è andata a buon fine, dovrebbe contenere un token che va memorizzato e utilizzato sempre anche in futuro per poter fare le altre richieste al server tenendo però presente che questo ha una validità limitata a trenta minuti.

Quando questo limite di tempo sarà scaduto, sarà necessario ri-logarsi per poter aver un altro token.

Le altre richieste al server che utilizziamo maggiormente sono la POST

<https://students.measurify.org/v1/measurements> che si utilizza per poter memorizzare i dati raccolti sul server, e la GET <https://students.measurify.org/v1/measurements> con la quale invece si possono visualizzare i dati salvati sul server.

## 2.4. Postman

Postman è l'API che serve per la creazione e l'utilizzo di altre API: è molto utile per effettuare le nostre richieste http al server di Measurify, e per poter riuscire a tener a disposizione dell'utente la memorizzazione delle measurement. [5]

Postman svolge un'importante funzione in questo ambito, perché possiamo crearci una nostra collezione dove salvare le richieste e le variabili locali (dalla semplice url, senza doverlo specificare tutte le volte fino a variabili più complesse). Così facendo le nostre richieste verranno salvate senza la necessità di inserire ogni volta i medesimi campi.

Ad esempio, oltre salvare nel body della richiesta della POST Login le variabili di accesso, si può utilizzare la funzione il tab "Test" di Postman (figura 9)

```
1 pm.environment.set("token", pm.response.json()["token"]);
```

Figura 9 - Script per salvare il token dal server

Con questo codice appena illustrato, successivamente alla richiesta al server, potremo salvare sulla variabile locale “token”, il token restituito dal server, in modo che questo possa essere automaticamente inserito come variabile in tutte le richieste nella sezione *Headers*, ottenendo con ciò che tutte le successive richieste abbiano già memorizzato il token necessario.

Spesso ci può essere la necessità di modificare il comportamento del dispositivo, in modo che mandi le misure solo a determinate condizioni o dopo aver eseguito alcuni calcoli. Per questo è possibile effettuare una richiesta alle API, in momenti specifici, per ottenere uno “script” con le informazioni utili alla definizione di come il microcontrollore si debba comportare nel memorizzare le richieste. In questo progetto lo script è

```
"code": "physiological-state().send()"
```

Con questo vengono così inviati semplicemente i dati raccolti dal sensore, tramite la funzione *send()*.

## 2.5 Edge Engine

Edge Engine, creata dal Elios Lab, è una libreria scritta in codice C standard e che può essere compilata per board con microcontrollore (Arduino) e per piattaforme desktop (Linux, MacOS, Windows).

L'idea è quella di avere una libreria che ci aiuti ad interagire con le API di Measurify per supportare il programmatore ad eseguire la computazione direttamente su Arduino, descrivendo però la computazione sul cloud. [6]

Questa libreria elabora i flussi dei dati provenienti dai sensori tramite script o esegue comandi utilizzando gli attuatori ad esso collegati. Gli script sono costituiti da un insieme di operazioni note che possono essere utilizzate insieme per eseguire calcoli, anche complessi, su ciascun flusso di misura. L'Edge Engine può essere configurato per recuperare dal cloud, per esempio Measurify, gli script da eseguire localmente e controllare periodicamente se questi sono cambiati o se ci sono nuovi script da eseguire.

In fase di realizzazione della tesi ho dovuto cambiare il codice di Edge Engine affinché potesse supportare la scrittura di dati multidimensionali perché prima erano previsti solo quelli monodimensionali, ma di tutto ciò tratterò meglio nella sezione 3.

## 2.6. Flutter

Per quanto riguarda la parte di visualizzazione dei dati raccolti si è sviluppata un'applicazione utilizzando Flutter.

Quest'ultimo è un Framework UI mobile gratuito e open-source creato da Google e rilasciato nel maggio 2017.

Semplicemente può dirsi che questo consente di creare un'applicazione mobile nativa con un'unica base di codice. Ciò significa che è possibile utilizzare un unico linguaggio di programmazione e un'unica base di codice per creare applicazioni diverse (per iOS e Android, per web).

Flutter è composto da due parti importanti:

- Un SDK (Software Development Kit), ossia una raccolta di strumenti che aiutano a sviluppare le diverse applicazioni. Questo include anche strumenti per compilare il codice in codice macchina nativo (codice per iOS e Android).
- Un framework (libreria UI basata su widget), ossia una raccolta di elementi riutilizzabili dell'interfaccia utente (pulsanti, input di testo, cursori e così via) che si possono personalizzare per ogni esigenza.

Il linguaggio di programmazione per scrivere applicazioni Flutter, è il Dart, che è stato creato nel 2011, sempre da Google. Dart si concentra sullo sviluppo front-end e può essere utilizzato per creare applicazioni mobili e web. Dart è un linguaggio di programmazione orientato agli oggetti, e la sua sintassi è paragonabile a quella di JavaScript.

Durante lo sviluppo, le app Flutter vengono eseguite in una macchina virtuale che offre il caricamento a caldo delle modifiche senza la necessità di una ricompilazione completa. Per il rilascio, le app Flutter vengono compilate direttamente in codice macchina, sia con istruzioni Intel x64 che ARM, o in JavaScript se destinate al web. [7]

### 3. Sperimentazione e risultati

#### 3.1. Acquisizione dati sulla scheda Arduino

La prima fase del progetto è stata quella di acquisizione dei dati sulla scheda Arduino e successivamente l'invio dei dati stessi al server cloud di Measurify.

Per la prima parte, occorre programmare la scheda Arduino con uno sketch, scritto in C++, suddiviso in due parti, la parte di Setup e quella di Loop.

Le librerie che sono state utilizzate per implementare il codice sono:

- Wire.h, questa libreria ci consente di comunicare con i dispositivi I<sup>2</sup>C
- Max32664.h, questa libreria ci consente di riuscire a configurare e a memorizzare i dati dal sensore
- EdgeEngine\_library.h, questa libreria è quella che ho citato in precedenza e che serve essenzialmente per collegare il microcontrollore al server Measurify e fare tutte le richieste necessarie.

Prima di andare ad esaminare la fase dell'acquisizione dei dati, mi vorrei soffermare su un importante passaggio che ho dovuto affrontare affinché EdgeEngine acquisisse i miei quattro dati salvandoli tutti e quattro nello stesso array.

Il primo cambiamento è stato fatto nella classe *Sample*

Infatti, prima i valori venivano salvati in un campo di tipo stringa mono dimensione, mentre ora questo è stato invece cambiato in vector di float, ed è stato aggiunto un metodo di conversione da vector a stringa. Infine, in questa classe è stato aggiunto anche un altro campo di *int* che serve per indicare la grandezza che dovrà avere il vector di campioni; tale valore verrà preso direttamente dall'oggetto JSON: feature.

```
class sample{
private:
//variables

//methods

public:
//variables
float value;
string startDate;
string endDate;
string url;
string thing;
string feature;
string device;
string scriptId;

//methods

//constructors
sample(string);
sample(const sample&);
};

class sample{
private:
//variables

//methods

public:
//variables
vector<float> myArray;
string startDate;
string endDate;
string url;
string thing;
string feature;
string device;
string scriptId;
int sizeOfSamples;

//methods
string ArrayToString(vector<float> myArray);

//constructors
sample(string);
sample(const sample&);
};

string sample::ArrayToString(vector<float> myArray){
stringstream sstream;
if(myArray.size() == 1){
    sstream << myArray[0];
}else{
    sstream << "[";
    for(int i=0; i < myArray.size(); i++){
        sstream << myArray[i];
        if(i < (myArray.size()-1)){
            sstream << ",";
        }
    }
    sstream << " ]";
}
return sstream.str();
}
```

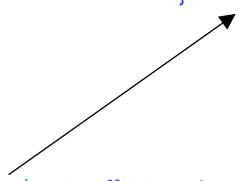


Figura 10 Codice prima e dopo della classe "Sample"

L'altra modifica che ho ritenuto di apportare alla libreria è stata l'aggiunta, dopo la GET SCRIPT, della GET FEATURE, che viene eseguita quando inizia tutta la gestione del *login* da parte del microcontrollore al server e le successive richieste per memorizzare i dati.

Il compito di questa GET è quello di accedere all'oggetto feature.

Siccome l'oggetto JSON feature è composto come visto in precedenza nella figura 7, una volta ricevuta la risposta viene a cercarsi il campo *items* che al proprio interno contiene la descrizione di quanti valori si aspetterà come campioni.

Così facendo verrà sfruttata la sintassi del JSON per eseguire questo calcolo, perché si andrà ad incrementare una variabile (*counter*), precedentemente aggiunta tra le variabili globali di Eagine, ogni volta che si incontrerà una parentesi graffa aperta. Tale processo continuerà finché il campo *items* non terminerà (e ciò avverrà sempre grazie alla sintassi, perché tale istruzione verrà impartita tramite la chiusura di tre parentesi "}}]").

In questo modo, successivamente, il valore della variabile *counter* verrà assegnato alla variabile della classe *sample* che era stata creata per contenere il numero massimo di variabili (*sizeOfSamples*).

```
// GET FEATURES
deleteSpaces(features);
int beginOfValue = features.find('\\"') +1;
int endOfValue = features.find('\\"', beginOfValue) -1 ;
string fieldValue = features.substr(beginOfValue, endOfValue);

fieldName = opts.url+"/"+opts.ver+"/features/"+fieldValue;

response = Api->GETFeatures(fieldName, token); // Get the features
if(isOkResponse(response)){
    deleteSpaces(response);
    if (response.find(fieldValue) == -1)
    {
        #ifdef ARDUINO
        Serial.println(fieldName.c_str());
        Serial.println(" field is not present!");
        #else
        cout << fieldName.c_str();
        cout << " field is not present!" << endl;
        #endif
    }
    else{
        beginOfValue = response.find("items");
        endOfValue = response.find("}}]")-1;
        fieldValue = response.substr(beginOfValue, endOfValue);

        for (char i : fieldValue){
            if (i == '{'){
                counter++;
            }
        }
    }
}
```

Figura 11 GET Feature, codice aggiunto alla libreria.

Come si accennava prima l'iniziale fase per l'acquisizione dei dati è la parte di Setup, che serve per preparare sia il sensore alla ricezione dei dati, che di stabilire la connessione ad internet.

Per prima cosa vengono inizializzate tutte le variabili necessarie: i pin per la gestione del sensore e le credenziali della rete locale per poter accedere al server.

```
#define RESET_PIN 04
#define MFIO_PIN 02
#define RAWDATA_BUFFLEN 250

MAX32664 MAX32664(RESET_PIN, MFIO_PIN, RAWDATA_BUFFLEN);

sample *physiologicalSample = NULL;
vector<sample*> samplesVect;

const char* ssidWifi = "";
const char* passWifi = "";

edgine* Edge;
connection* Connection; //Wrapper for the wifi connection
```

Dopo l'inizializzazione delle variabili necessarie, il programma procede con una nuova connessione WIFI tramite le due variabili, prima inizializzate, di SSID e Password della rete locale. Questa connessione è gestita da un modulo descritto nella libreria di EdgeEngine.



```
Wire.begin();

loadAlgomodeParameters();

int result = MAX32664.hubBegin();
if (result == CMD_SUCCESS){
    Serial.println("Sensorhub begin!");
}else{
    //stay here.
    while(1){
        Serial.println("Could not communicate with the sensor!");
        Serial.println("please make proper connections");
        delay(5000);
    }
}

bool ret = MAX32664.startBPTcalibration();
while(!ret){
    delay(10000);
    Serial.println("failed calib, please restart");
    ret = MAX32664.startBPTcalibration();
}
delay(1000);

Serial.println("start in estimation mode");
ret = MAX32664.configAlgoInEstimationMode();
while(!ret){
    Serial.println("failed est mode");
    ret = MAX32664.configAlgoInEstimationMode();
    delay(10000);
}

void loadAlgomodeParameters(){
    algomodeInitialiser algoParameters;

    algoParameters.calibValSys[0] = 120;
    algoParameters.calibValSys[1] = 122;
    algoParameters.calibValSys[2] = 125;

    algoParameters.calibValDia[0] = 80;
    algoParameters.calibValDia[1] = 81;
    algoParameters.calibValDia[2] = 82;

    algoParameters.spo2CalibCoefA = 1.5958422;
    algoParameters.spo2CalibCoefB = -34.659664;
    algoParameters.spo2CalibCoefC = 112.68987;

    MAX32664.loadAlgorithmParameters(&algoParameters);
}
```

Figura 12 Prima fase della Setup()

Successivamente viene stabilita la comunicazione tramite protocollo I<sup>2</sup>C per il sensore.

Vengono caricati i valori di calibrazione di cui abbiamo parlato precedentemente, in seguito inizia la calibrazione del sensore.

In questa fase il fruitore del progetto dovrà tenere un dito appoggiato al sensore finché il processo non avrà finito. In caso di calibrazione errata, verrà ripetuta finché questa non andrà a buon fine.

Finita la calibrazione, inizierà la configurazione vera e propria per poter stimare i due valori della pressione sistolica e diastolica.

Infine, prima della conclusione della fase di Setup si procederà definendo le *options*, (struttura definita da Edge) che serviranno a creare la connessione diretta al cloud, con tutte le rotte necessarie a fare le richieste HTTP.

Ad esempio, vengono definiti: l'URL del cloud, le credenziali per poter fare il login su Measurify, la *thing* e il *device* in questione.

Infine, viene inizializzato Edge Engine per poter avviare la comunicazione con esso e procedere all'invio dei dati al cloud

```
options opts;
opts.username = "body-monitor-username";
opts.password = "body-monitor-password";
opts.tenant = "body-monitor";
//route
opts.url = "https://students.measurify.org";
opts.ver = "v1";
opts.login = "login";
opts.devs = "devices";
opts.scps = "scripts";

opts.measurements = "measurements";
opts.info = "info";
opts.issues = "issues";
//Edge identifiers
opts.thing = "mario.rossi";
opts.device = "body-monitor";
opts.id = "body-monitor";

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

## 3.2. Invio dei dati al cloud

La seconda parte è invece quella della fase di Loop, nella quale vengono ripetute delle azioni per poter acquisire i dati e inviarli al cloud per memorizzarli, questo almeno finché il device è attivo.

Per prima cosa, viene generato un nuovo campione che conterrà le quattro misurazioni, poi viene memorizzata la data e l'ora per tenere traccia dei rilevamenti, viene inizializzato il campo del numero di misurazioni massime che deve contenere, come prima spiegato.

Successivamente, viene creato un array dalle dimensioni appena ottenute (ovviamente avrà la lunghezza giusta perché assumerà tale dato direttamente da come è composto l'oggetto JSON sul cloud) e infine inizializzato tutto a zero, affinché in caso di malfunzionamento non termini automaticamente l'algoritmo, ma al massimo che venga salvata (in ipotesi) una misurazione nulla.

Si allega il codice della parte appena spiegata:

```
physiologicalSample = new sample("physiological-state");
physiologicalSample->startDate = Edge->Api->getActualDate();
physiologicalSample->endDate = physiologicalSample->startDate;
physiologicalSample->sizeOfSamples = Edge->getItems();
physiologicalSample->myArray.assign(physiologicalSample->sizeOfSamples, 0);
```

Dopo aver stanziato il campione, inizia la lettura dei dati dal sensore e la memorizzazione delle misurazioni nell'array definito nella classe *Samples*.

```
uint8_t num_samples = MAX32664.readSamples();

if(num_samples){
    Serial.print("sys = ");
    Serial.print(MAX32664.max32664output.sys);
    physiologicalSample->myArray[0]= MAX32664.max32664output.sys;
    Serial.print(", dia = ");
    Serial.print(MAX32664.max32664output.dia);
    physiologicalSample->myArray[1]= MAX32664.max32664output.dia;
    Serial.print(", hr = ");
    Serial.print(MAX32664.max32664output.hr);
    physiologicalSample->myArray[2]= MAX32664.max32664output.hr;
    Serial.print(" spo2 = ");
    Serial.println(MAX32664.max32664output.spo2);
    physiologicalSample->myArray[3]= MAX32664.max32664output.spo2;
}
delay(100);
```

Successivamente tale rilevamento viene caricato sul vettore di campioni inizialmente inizializzato (`samplesVect.push_back (physiologicalSample)`)

In questo mio progetto, tutti i campioni sono misurati soltanto dall'unico sensore esistente, ma, in generale, nulla vieta che i dati possano anche essere generati da sensori diversi e quindi questi verrebbero stanziati ogni volta e poi caricati tutti nello stesso vettore di campioni.

Una volta caricati tutti i campioni delle misurazioni, si caricherà tutto il vettore sul cloud passando attraverso l'insieme degli script di Measurify.

Prima di riiniziare il loop, occorre cancellare tutti i campioni raccolti dal vettore dei *Samples* e le variabili inizializzate per poter far iterare il percorso senza problemi.

### 3.3. Flutter

La parte finale del progetto di tesi, è stato quello di creare un'interfaccia grafica per avere la visualizzazione di tutti i dati memorizzati nel cloud.

La prima schermata, quando si avvia l'applicazione, è quella che permette di fare il login al server di Measurify.

La schermata è composta da tre `TextFormField` (Username, password e tenat), con la caratteristica che quando si procederà a scrivere nei campi, la barra sottostante ad ogni specifico campo diventerà invisibile, e ciò per far capire che si sta scrivendo proprio in quel campo. Un'altra particolarità è quella dell'icona del campo password, che oltre a essere un'icona è anche un bottone che se premuto renderà la password, che normalmente è oscurata dai classici pallini, invece visibile.

Nella parte inferiore, dopo i `TextFormField`, si trova un bottone di *Login* che se premuto provvederà a provare a mandare una richiesta http al cloud, provando ad accedervi. Il sistema procederà a paragonare le tre variabili locali (dove si è salvato quello che l'utente ha scritto nei campi antecedenti) con quelle che si aspetta di ricevere.

Se le credenziali inserite sono sbagliate e quindi non vengono riconosciute dal server, verrà visualizzato in fondo alla pagina di login un messaggio di errore, con conseguente impossibilità ad accedere al cloud, e questo messaggio resterà visibile per un periodo limitato e poi sparirà.

Se la risposta al server è invece positiva da parte del cloud, questo nella sua risposta conterrà un token, che verrà salvato in una variabile globale, per poter fare le successive richieste a Measurify in autonomia.

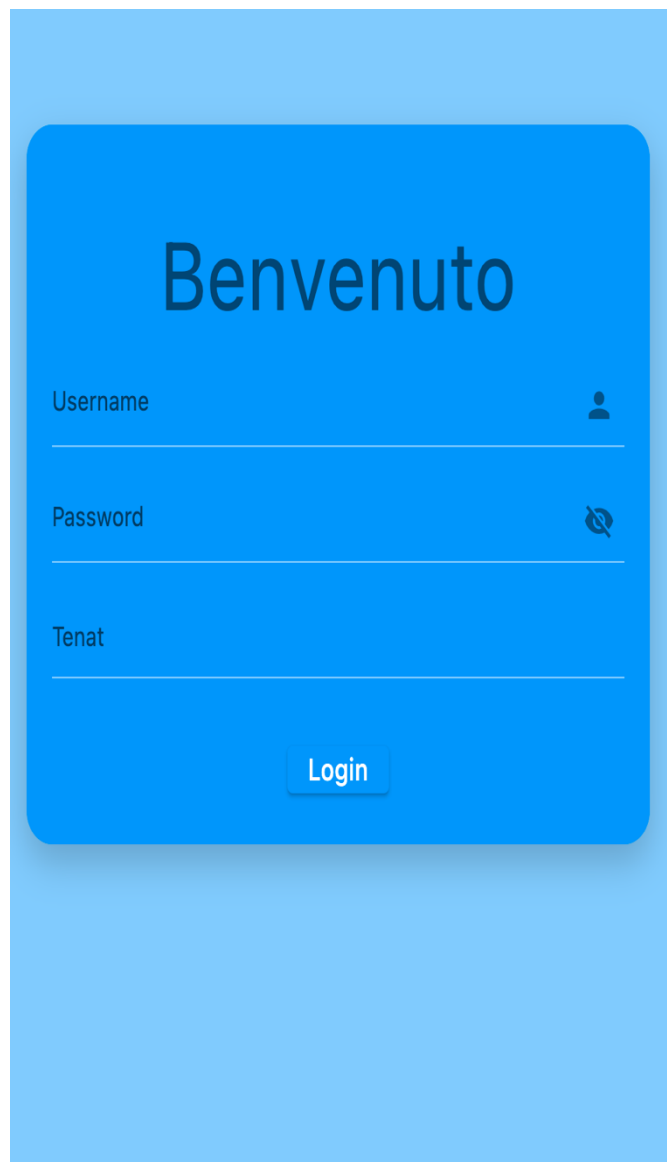


Figura 13 Schermata home di login



Appena la fase di login è terminata con successo, l'utente verrà automaticamente indirizzato nella seconda schermata, nella quale troverà un altro box (come da figura 14).

In questo box viene richiesto all'utente di scegliere quante misurazioni intende visualizzare. Dopo di che l'utente inserirà il numero di queste nel campo sottostante. Una volta cliccato il bottone "Salva", il numero inserito verrà salvato automaticamente in una variabile globale che verrà poi usata per poter filtrare la successiva richiesta.

In questo momento, l'utente vedrà una pagina di caricamento che lo avvertirà di attendere la fine del caricamento stesso per poter poi vedere le sue misurazioni.

Con il caricamento viene anche preparata la pagina di visualizzazione, che si ottiene facendo una GET al cloud alla rotta che contiene tutte le misurazioni fatte dal dispositivo e già memorizzate, aggiungendo però il limite di visualizzazione salvato precedentemente nella variabile globale.

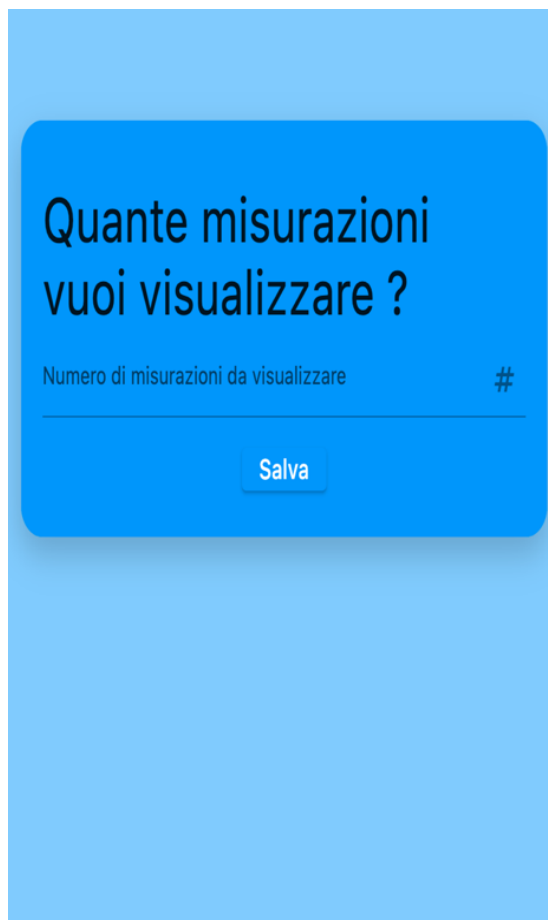
Appena fatta la richiesta si riceverà la risposta.

Siccome i dati a noi necessari vengono memorizzati nello stesso array e sempre nella stessa posizione (ad esempio se avrò bisogno della pressione sistolica, troverò questo dato sempre nella prima posizione del array) si potrà allora a ricavare dalle risposte ottenute i valori singoli che verranno salvati in un nuovo array in cui è contenuta solo quella specifica variabile. Quindi, alla fine, si avranno quattro distinte liste, a seconda del diverso dato salvato.

Successivamente, appena il sistema avrà finito di caricare i dati, comparirà l'ultima schermata (come da figura 15) che è composta da quattro Card. In ciascuna di queste verrà visualizzata la misurazione specifica, individuata con la data in cui è stata effettuata.

Come spiegavo in precedenza, questi dati sono contenuti in liste che hanno una dimensione proporzionale al numero scelto ed inserito dall'utente.

Sotto le quattro Card si trova poi un bottone di refresh, che servirà all'utente per avere una schermata sempre aggiornata. Ciò anche nel caso in cui proprio in quel momento l'utente intenda effettuare una nuova misurazione che porterà quindi all'immediato aggiornamento dei dati.



Quante misurazioni vuoi visualizzare ?

Numero di misurazioni da visualizzare

Salva

Figura 14 Schermata di impostazioni



	Pressione Sistolica
114.0 mmHg 07/06/2022	
	Pressione Diastolica
75.00 mmHg 07/06/2022	
	Frequenza Cardiaca
67 bpm 07/06/2022	
	Saturazione
97% 07/06/2022	




Figura 15 Schermata di visualizzazione dei dati

## 4. Contributo personale e considerazioni conclusive

L'obbiettivo della tesi è stato, come già illustrato, quello di sviluppare un dispositivo fisico in grado di poter stimare la pressione sanguinea, misurare la frequenza cardiaca e la saturazione nel sangue, salvando tutti questi dati sul cloud, tramite internet.

Durante il progetto è stato necessario apportare modifiche alla libreria EdgeEngine affinché potesse supportare valori multidimensionali, dato che i valori misurati dal mio sensore venivano recuperati tutte e quattro in un'unica volta soltanto.

Per fare ciò, ho dovuto studiare più nel dettaglio le funzionalità della libreria per capirne l'essenza e poter aggiungere le nuove funzionalità affinché potesse funzionare sia nel caso in cui dovesse ricevere un valore monodimensionale oppure un valore multidimensionale (come nel mio caso).

Il codice è stato scritto e testato non solamente per essere compilato con le schede Arduino, ma anche per poter essere compilato su Windows e/o MacOS.

Sono stati eseguiti, prima, dei test con dati puramente fittizi per verificare la funzionalità del codice aggiunto; in seguito, si è cominciato con l'implementazione del codice del sensore, prima testandolo singolarmente e poi cercando di farlo interagire con la libreria di Edge.

Allo stato attuale, il codice sembra funzionare, ma presenta il limite di una sola misurazione alla volta, problematica questa da approfondire ancora e quindi risolvere.

Per quanto concerne l'interfaccia grafica con l'utente, si è usato il toolkit rilasciato da Google denominato "Flutter" e per poter implementare tale applicazione si è dovuto ricercare e studiarne la documentazione. L'applicazione è stata fatta con lo scopo che sia molto semplice ed intuitiva, seguendo l'utente passo dopo passo. La principale necessità dell'applicazione è infatti quella di mostrare e di immagazzinare i valori misurati all'utente, per poi renderli disponibili anche in un secondo momento.

Infine, va ricordato che le future implementazioni di questo progetto potrebbero essere finalizzate alla creazione di un sistema più comodo e indossabile per essere usato spesso in modo da poter misurare i valori più volte nell'arco della stessa giornata.

Un'altra implementazione potrebbe riguardare invece l'interfaccia grafica, aggiungendo impostazioni sia di visualizzazione che sia una parte dedicata al medico curante nella quale lo stesso possa avere a disposizione tutte le situazioni cliniche dei suoi pazienti che usano questo strumento seguendo l'evoluzione delle stesse, adeguando - se del caso - le terapie necessarie.

Un ulteriore problema che non si è oggi affrontato nello studio di questo apparecchio, oggetto studio di questa tesi è quello della possibilità di implementare l'alimentazione, perché per ora, questa è possibile solamente tramite un collegamento USB al pc, mentre sarebbe più semplice, comodo e maneggevole munirla di una batteria ricaricabile.

## 5. Riferimenti bibliografici

- [1] Cisco nascita dell'IoT - [https://www.cisco.com/c/dam/en\\_us/about/ac79/docs/innov/IoT\\_IBSG\\_0411FINAL.pdf](https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf)
- [2] Arduino Nano 33 IoT- <https://docs.arduino.cc/hardware/nano-33-iot>
- [3] Sensore Max32664-D- <https://www.maximintegrated.com/en/design/technical-documents/app-notes/6/6845.html>
- [4] Measurify- <https://measurify.org>
- [5] Postman - <https://www.postman.com/product/what-is-postman/>
- [6] EdgeEngine- <https://github.com/measurify/edge>
- [7] Flutter- <https://flutter.dev>