



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

Ottobre 2020

Progetto e implementazione di un sistema embedded per il controllo di presenza da remoto

Candidato: Laura Triglia, Sebastiano Gioia

Relatore: Prof. Riccardo Berta

Sommario

Lo scopo di questo lavoro di tesi è quello di illustrare il progetto di un sistema per monitorare una stanza e notificare eventuali nuove presenze nella stanza stessa.

Il progetto quindi sarà formato da:

- Un **modulo fisso**, un dispositivo che sia installato nella stanza da monitorare
- Un **modulo mobile**, che serva a notificare l'utente dell'eventuale presenza rilevata

I due moduli comunicheranno fra loro grazie alle API di Measurify [1] che si occuperà di gestire il lato server, facendo quindi in modo di far comunicare il modulo fisso con quello mobile ogni qualvolta qualcuno o qualcosa entri nella stanza.

Nel concreto, i rilevamenti di presenza collezionati dal dispositivo installato nella stanza vengono mandati all'API Measurify, tramite la quale uno smartphone, ad essa opportunamente associato, viene notificato della nuova presenza nella stanza tramite un'app appositamente progettata.

Indice

1	INTRODUZIONE	3
2	METODI E STRUMENTI UTILIZZATI	4
2.1	MODULO SERVER: MEASURIFY	5
2.1.1	COMUNICAZIONE PERSON-MEASURIFY VIA DESKTOP: POSTMAN	7
2.2	MODULO FISSO: SISTEMA SENSORE DI PRESENZA	8
2.2.1	LATO SOFTWARE: AMBIENTE ARDUINO	8
2.2.1.1	Linguaggio Wiring	9
2.2.1.2	Comunicazione machine-Measurify: libreria Edge Engine	9
2.2.2	LATO HARDWARE	10
2.2.2.1	Sensore PIR	10
2.2.2.2	Scheda ESP32	10
2.2.2.3	Componenti elettronici di contorno	11
2.3	MODULO PORTATILE: APPLICATIVO SMARTPHONE	12
2.3.1	LATO SOFTWARE: FRAMEWORK FLUTTER	12
2.3.1.1	Linguaggio Dart	13
2.3.2	COMUNICAZIONE MEASURIFY-PERSON VIA MOBILE: FIREBASE E NOTIFICHE PUSH	14
3	SPERIMENTAZIONI E RISULTATI	15
3.1	SVILUPPO MODULO FISSO	15
3.1.1.1	Punto di vista circuitale e scelte per la misura della presence	15
3.1.1.2	Punto di vista funzionale e comunicazione con Measurify	16
3.2	SVILUPPO MODULO PORTATILE	17
3.2.1.1	Punto di vista funzionale e strutturale	17
3.2.1.2	Funzionamento concreto dell'applicazione	19
4	CONTRIBUTO PERSONALE E CONSIDERAZIONI CONCLUSIVE	21
4.1	TESTING DELLA LIBRERIA EDGE ENGINE	21
4.2	MIGLIORAMENTI POSSIBILI	21
4.2.1	LATO MODULO FISSO EMBEDDARE IL SISTEMA	22
4.2.2	LATO MODULO MOBILE:	22
5	RIFERIMENTI BIBLIOGRAFICI	23

1 Introduzione

Nel XXI secolo quasi ogni oggetto è ormai connesso ad internet, negli ultimi anni si sente parlare sempre di più di Internet of Things (IoT) che si sta evolvendo per far parte della nostra vita quotidiana. Questo concetto consiste nel permettere ad ogni dispositivo di comunicare fra di loro, scambiandosi informazioni. L'obiettivo dell'IoT è quello di far sì che il mondo reale venga mappato in uno virtuale, vengano raccolte ed elaborate informazioni. Negli ultimi anni per IoT si intendono le cose che hanno un'identità ed una personalità virtuale che operano che si connettono e comunicano con il loro ambiente sociale e con l'utente in una maniera ritenuta "intelligente".

Proprio per questo una delle basi dell'IoT sono i sensori, ovvero dispositivi che raccolgono dati sul mondo reale circostante. Questi dati vengono poi elaborati, utilizzati per prendere decisioni, salvati in database e possono portare ad una risposta, sempre nel mondo reale, tramite degli attuatori. Esistono innumerevoli esempi di IoT, perfino le automobili hanno a disposizione diversi dispositivi intelligenti.

Quello che ormai tutti siamo abituati a vedere sono i dispositivi per la SmartHome, che ci rendono la vita in casa più semplice e più "connessa". Basti pensare ad ogni assistente vocale che sta entrando nelle nostre case e ad ogni dispositivo che si interfaccia connesso: la lampadina che si accende e spegne mediante un comando vocale, il termostato che ci permette di regolare la temperatura dallo smartphone ancora prima di arrivare a casa, gestione degli elettrodomestici e molto altro.

La nostra idea si ispira al contesto della home security, andando a guardare l'obiettivo che vi è alla base: creare un sistema che sia in grado di notificare all'utente in tempo reale la rilevazione di una presenza esterna all'interno del luogo in cui si è installato il sistema.

Per tanto l'esercizio sta proprio nella realizzazione pratica di quell'obiettivo, con un sistema basilare che svolga le funzioni necessarie.

2 Metodi e strumenti utilizzati

Sono due le parole chiave che hanno legato ogni aspetto di questo progetto:

- Semplicità
- Completezza

Si è scelto quindi di seguire una “progettazione modulare”, ossia capire quali fossero i problemi legati alla realizzazione dell'obiettivo primario, scomporlo sulla base di questi ultimi e risolvere ognuno di questi problemi. Di conseguenza, si sono progettati tanti moduli ognuno dei quali risolve un micro-problema; tramite la comunicazione tra questi moduli è possibile arrivare alla soluzione del problema progettando quindi tanti moduli i quali nel loro singolo risolvono il micro-problema e nel complesso, comunicando tra loro, consentono di raggiungere l'obiettivo primario.

Tale approccio modulare è **semplice** poiché consente di ridurre la difficoltà della realizzazione dell'obiettivo generale, che nella sua complessità è da considerare sotto molteplici aspetti. Procedendo in maniera modulare si riesce a mantenere la visione generale del sistema e nello stesso tempo tale pratica consente di focalizzarsi meglio su aspetti precisi. Inoltre è stato possibile ottimizzare i tempi sia nella risoluzione sia nella manutenzione del singolo modulo. Sintetizzando, il sistema risulta semplice non solo perché i moduli descrivono un problema di minore difficoltà rispetto a quello generale, ma anche perché la gestione di questi è semplificata.

Il progetto inoltre è **completo** perché non solo perché l'obiettivo è completo di molteplici aspetti e quindi il sistema deve saper svolgere molteplici funzionalità, ma una seconda motivazione sta anche nel fatto che essendo anche i moduli progettati con logica modulare essi sono estendibili, nel senso che si possono aggiungere altri sotto-moduli che vadano ad aggiungere funzioni accessorie, in modo da completare in questo senso l'obiettivo principale.

Si è deciso di suddividere l'obiettivo principale in tre macro-moduli, scomponendo quindi il problema generale in tre macro-aspetti:

- **Modulo fisso** per quanto riguarda l'aspetto della rilevazione della presenza
- **Modulo portatile** per quanto riguarda l'aspetto della notifica all'utente della avvenuta rilevazione
- **Modulo server** per quanto riguarda l'aspetto di interconnessione dei moduli precedenti (il cloud).

I tre moduli sono quindi collegati come rappresentato in Figura1.

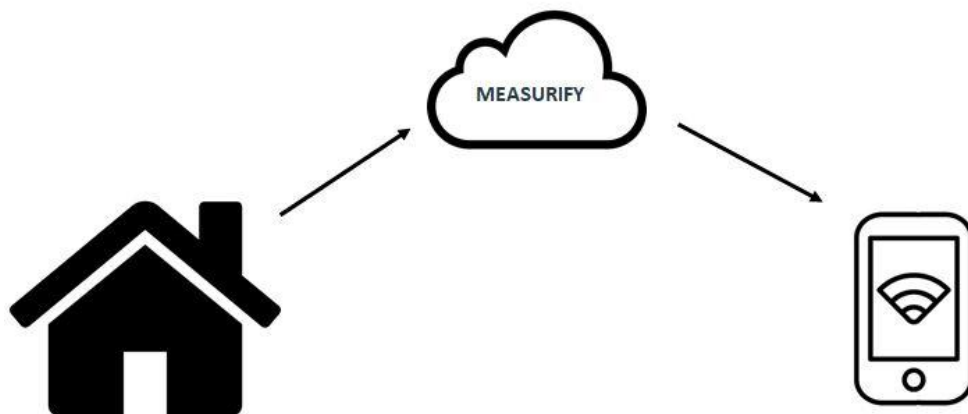


Figura 1 schema delle interconnessioni dei moduli

In breve ogni qual volta che il modulo fisso rileva la presenza, la manda al modulo server, il quale poi provvederà non solo a salvare la misura, ma anche a comunicare al modulo portatile dell'avvenuta rilevazione, in modo da avvisare l'utente dell'avvenuta presenza di un corpo esterno nella stanza.

Nei prossimi sotto-capitoli verranno meglio apprenditi i moduli nel loro singolo.

2.1 Modulo server: Measurify

Dovendo rilevare la presenza, quindi collezionare misure, si è optato di utilizzare le API Measurify, in quanto ben si presta a funzioni di questo genere.

Measurify[9] è un framework improntato al mondo dell'IoT, sviluppato da "Helios Lab" dell'Università degli Studi di Genova, il quale mette a disposizione delle API cloud-based e orientate alle misurazioni per gestire i dispositivi intelligenti in ecosistemi IoT. Pensato appositamente per il mondo IoT, Measurify ruota attorno al concetto di **measurement**, tipo di dato molto comune in questo ambiente.

Measurify è stato pensato per rappresentare il contesto dell'applicazione e dei suoi elementi come oggetti astratti collegati fra loro.

Le API di Measurify si basano su alcuni concetti chiave:

- **Thing**: si tratta di una "cosa" (una persona, un ambiente, un oggetto, ecc.) per la quale si sta facendo una misurazione.
Nel caso specifico del progetto, la thing corrisponde alla stanza dove si trova il sensore.
- **Feature**: riguarda ciò che viene misurato (la temperatura, l'umidità, il peso, ecc.).
Nel caso specifico del progetto, si tratta della rilevazione dell'ingresso di un corpo nella stanza.
- **Device**: si tratta di un dispositivo (hardware/software) che può misurare una certa dimensione (feature) su una determinata cosa (thing).
Nel caso specifico di questo progetto il device è il sensore PIR che rileva la presenza nella stanza.
- **Measurement**: si tratta di una misura effettuata da un dispositivo (device) per una certa dimensione (feature) su una determinata cosa (thing).
Nel caso specifico del progetto, essendo la misura di tipo booleano (0 = non è entrato nessuno, 1 = è entrato qualcosa), le measurement saranno verosimilmente tutti valori uguali a 1, poiché non sarebbe sensato mandare anche le misure negative.

Tali concetti diventano in Measurify oggetti, i quali sono poi modellati come **risorse**; queste contengono al loro interno gli oggetti veri e propri, con i propri modelli e funzionalità accessibili attraverso un set di RESTful API. Grazie alle API è possibile accedere tramite connessione ad Internet agli oggetti di ogni risorsa. Gli oggetti sono modellizzati in formato JSON e sono collegati insieme tra loro come mostrato nella Figura2:

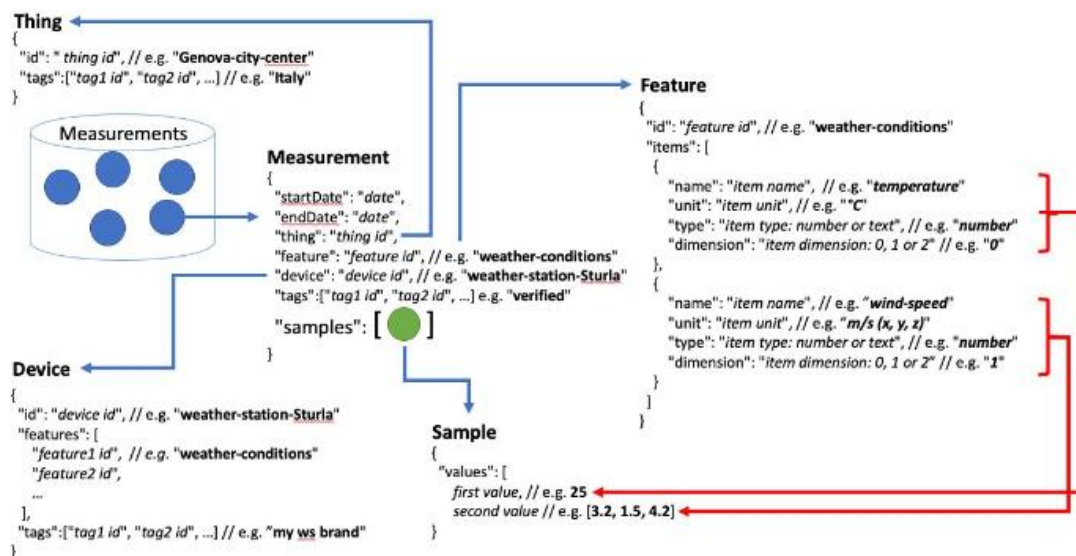


Figura 2: Modellizzazione e interconnessioni dei vari oggetti che costituiscono Measurify

Come si può vedere dalla Figura2, al centro di tutto vi è l'oggetto measurement al quale poi fanno riferimento anche gli altri oggetti che compongono tutta la struttura.

La figura3 invece, mostra l'architettura delle Cloud API, e quindi come queste permettano di gestire le varie risorse:

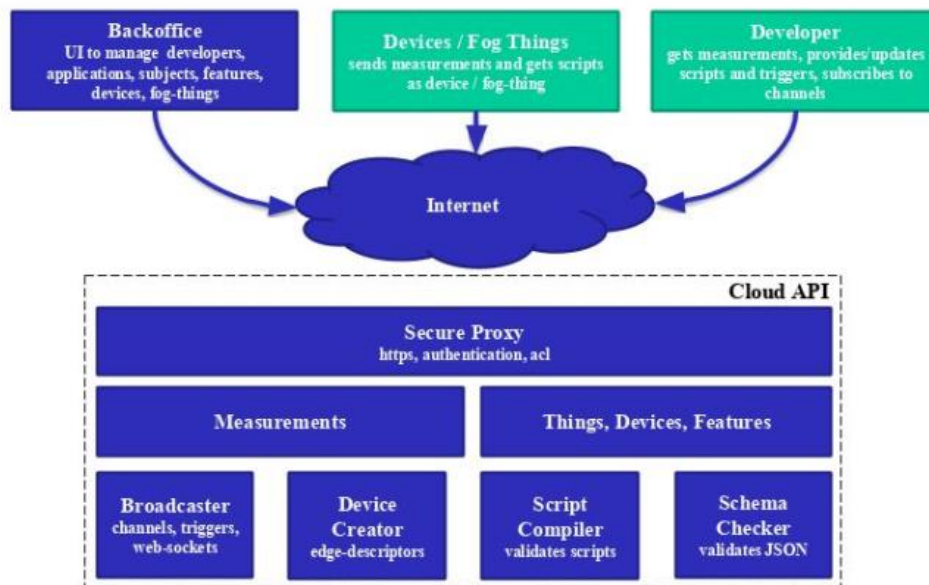


Figura 3: Architettura delle API Measurify

Come si può notare nell'architettura è presente prima un modulo Secure Proxy, il quale non permette di accedere alle risorse senza prima aver effettuato un'autenticazione. Inoltre, sempre in figura3, si può osservare come le risorse siano suddivise in due macro-generi (le measurement e tutti le altre risorse) i quali poi sono composti al loro interno di altri oggetti.

È stato scelto di utilizzare Measurify anche perché dispone della predisposizione per ricevere notifiche push (argomento che verrà esaminato nel paragrafo 2.2.1.2), e perché dispone anche della possibilità di aprire un canale video stream, implementazione interessante considerando l'idea di effettuare aggiornamenti funzionali futuri, come ad esempio la possibilità di accedere alla visione in tempo reale della stanza tramite video-camera.

2.1.1 Comunicazione person-Measurify via desktop: Postman

Per prendere confidenza con le API Measurify ed effettuare tutti le prove necessarie al fine della realizzazione del progetto, è stato usato il programma Postman[5], che consente di effettuare request http permettendo quindi l'interazione user-API.

Senza entrare troppo nel dettaglio di come funziona il programma, sono mostrate di seguito le operazioni da effettuare per accedere a Measurify:

- 1) Come mostrato in figura4, si seleziona il tipo di chiamata che si vuole effettuare (in questo una POST), e la route sulla quale fare la request:



Figura 4: Setting della route nonché del tipo di chiamata da effettuare

- 2) A questo punto, come mostrato in figura5, si passa alla creazione del body da inviare con la chiamata. In questo caso viene creato, nella sezione body del programma, un file in formato JSON contenente le informazioni necessarie per effettuare il login a Measurify (username e password):

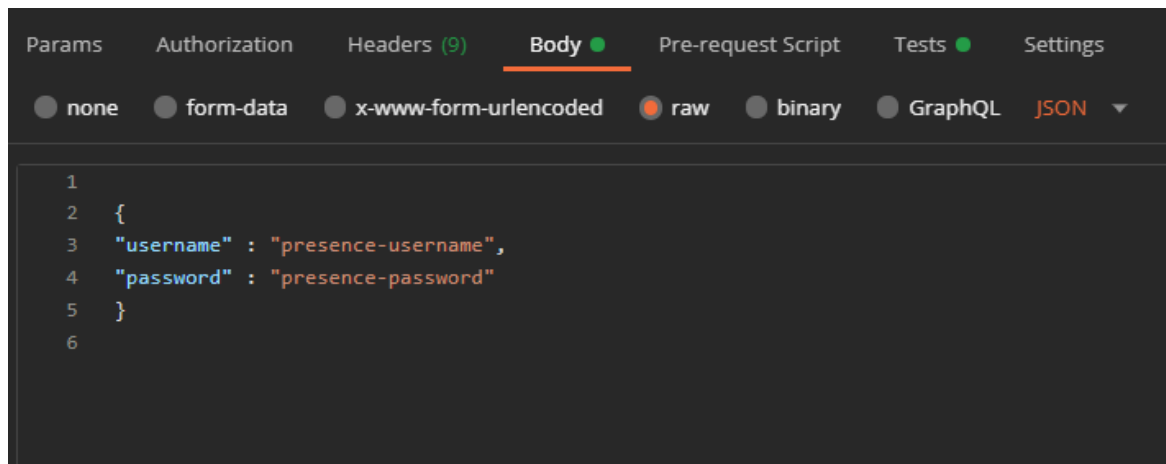


Figura 5: Creazione del body

- 3) Arrivati a questo punto, si clicca il tasto "Send" mostrato in figura4 e, se la procedura va a buon fine, Measurify risponde con un messaggio che ci notifica con lo statusCode = 200 l'esito positivo della chiamata e contenente all'interno del suo body, il file JSON contenente il token, come mostra figura6:

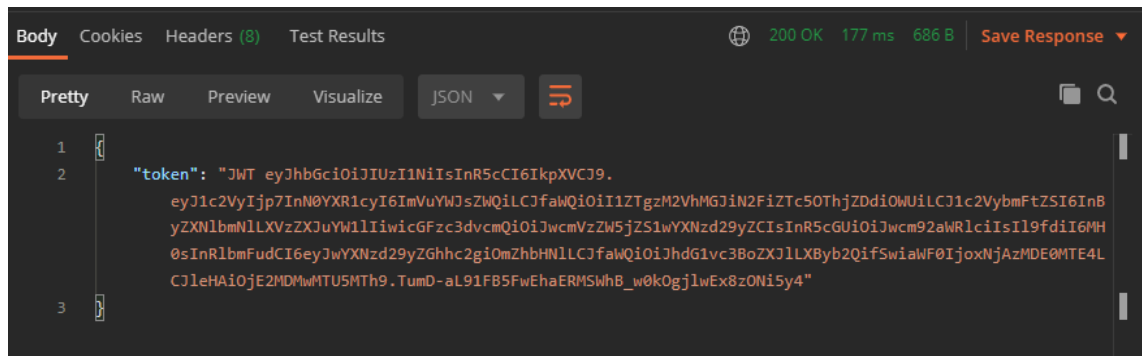


Figura 6: risposta da Measurify col token

Il token è un codice univoco, che servirà ad accedere alle risorse tramite l'effettuazione di chiamate future; nello specifico, questo token verrà messo nel campo Headers utilizzando la chiave "Authorization", durante la preparazione di ogni altra chiamata successiva a quella per fare il login.

2.2 Modulo Fisso: Sistema sensore di presenza

Come detto nel capitolo 2, il modulo fisso serve alla realizzazione dell'aspetto riguardante la rilevazione della presenza di corpi in una stanza. Per fare ciò si è scelto di creare un dispositivo embedded che fosse in grado di rilevare la presenza tramite un sensore, di elaborare tale informazione costruendo una measurement in formato JSON e infine di effettuare chiamate http per comunicare e mandare le misure a Measurify in automatico. Nel concreto quindi, si è scelto di utilizzare un **sensore PIR** e la scheda programmata **ESP32** compatibile con l'**ambiente Arduino**, il quale è uno dei migliori da utilizzare in casi del genere, per via della sua semplicità e del suo aspetto modulare (si possono connettere più componenti tra di loro al fine di creare dispositivi più complessi perché aventi più funzioni).

2.2.1 Lato software: Ambiente Arduino

Il progetto del modello fisso è basato sull'ambiente Arduino[6], il quale permette tramite le sue molteplici funzionalità di creare in modo relativamente semplice un sistema più complesso.

A livello hardware l'ambiente offre numerose schede programmabili, in genere molto intuitive da usare, nonché numerosi moduli accessori, i quali possono essere facilmente connessi alle schede in modo da ottenere sistemi embedded più funzionali.

A livello software l'ambiente Arduino offre gratuitamente Arduino IDE, il quale, consente la programmazione nonché il caricamento del programma sulla scheda da PC, in maniera relativamente facile e intuitiva. La figura7 mostra uno screen di Arduino IDE con lo scheletro di un programma di un file ".ino":

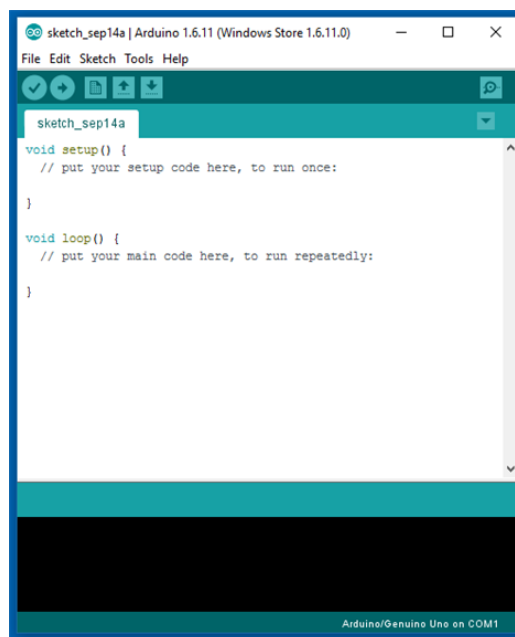


Figura 7: Arduino IDE con struttura di un file di codice in Wiring

Per questi motivi, la scelta di utilizzare questo ambiente risulta la più semplice da utilizzare, nonché la più completa.

2.2.1.1 Linguaggio Wiring

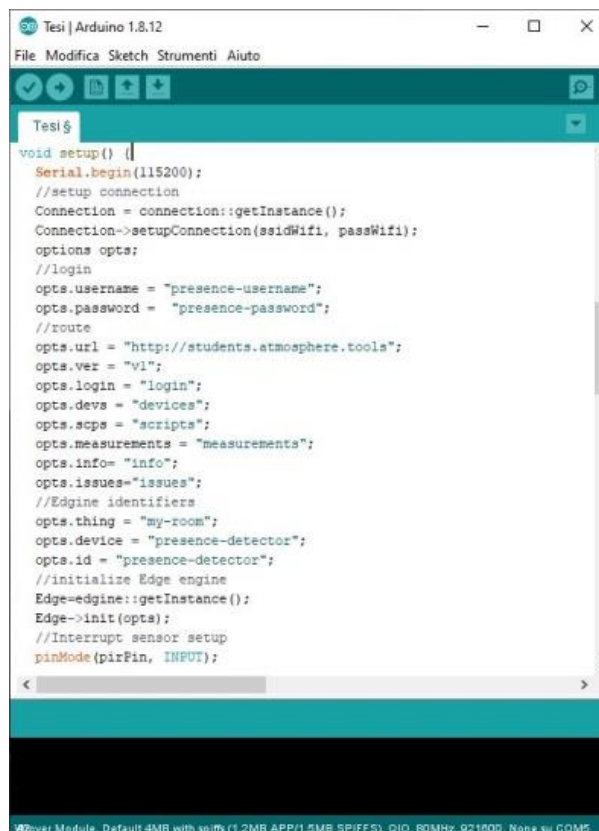
Il linguaggio di programmazione utilizzato in Arduino IDE è il Wiring il quale si basa sul ben più noto linguaggio C. Per tale motivo la programmazione risulta relativamente semplice.

Un programma “arduino.ino”, come mostrato in figura7, è composto da due metodi principali quello di `setup()` e quello di `loop()`:

- Nella funzione di **`setup()`** viene implementato tutto il codice inerente alla dichiarazione dei pin e il settaggio dei parametri richiesti dalle librerie per il loro funzionamento.
- Nella funzione di **`loop()`** viene implementato il codice inerente al compito che deve adempiere il microcontrollore

Nel contesto del nostro progetto, nel metodo di **`setup()`** sono state implementate tutte le istruzioni per settare i pin sui quali collegare i pin del sensore PIR, nonché tutte le istruzioni per configurare la scheda alla connessione a Internet e alla comunicazione con Measurify.

Nel metodo di **`loop()`** invece vengono implementate tutte le istruzioni riguardanti la lettura del sensore PIR, la registrazione di un’eventuale misura, la creazione della measurement ed infine la sua trasmissione a Measurify. La figura8 mostra i codici di due metodi:



```
void setup() {  
  Serial.begin(115200);  
  //setup connection  
  Connection = connection::getInstance();  
  Connection->setupConnection(ssidWifi, passWifi);  
  options opts;  
  //login  
  opts.username = "presence-username";  
  opts.password = "presence-password";  
  //route  
  opts.url = "http://students.atmosphere.tools";  
  opts.ver = "v1";  
  opts.login = "login";  
  opts.devs = "devices";  
  opts.scps = "scripts";  
  opts.measurements = "measurements";  
  opts.info = "info";  
  opts.issues = "issues";  
  //Edge engine identifiers  
  opts.thing = "my-room";  
  opts.device = "presence-detector";  
  opts.id = "presence-detector";  
  //initialize Edge engine  
  Edge=edgine::getInstance();  
  Edge->init(opts);  
  //Interrupt sensor setup  
  pinMode(pirPin, INPUT);  
}
```

Figura 8: codice del file “.ino” per il funzionamento della scheda

2.2.1.2 Comunicazione machine-Measurify: libreria Edge Engine

La connessione tra il cloud di Measurify e l’hardware è stato facilitato dall’implementazione nel codice Arduino della libreria EdgeEngine. Edge Engine è un’applicazione scritta in codice C standard e può essere compilata per board con microcontrollore, come Arduino, o per piattaforme desktop, come Linux, Windows e MacOS.

Nel capitolo3 verrà meglio approfondito il suo utilizzo e il grande aiuto che ha portato.

2.2.2 Lato hardware

Dal punto di vista Hardware il modulo fisso si compone di una scheda programmabile collegata ad un sensore PIR per effettuare il rilevamento della presenza. Di seguito sono meglio descritti i due componenti.

2.2.2.1 Sensore PIR

Un sensore PIR, o sensore ad infrarossi passivo (figura è un sensore elettronico che misura i raggi infrarossi irradiati dagli oggetti nel suo campo di vista. Il suo funzionamento è piuttosto semplice: tutti gli oggetti con temperatura superiore allo zero assoluto emettono energia sotto forma di radiazioni luminose; la maggior parte delle volte queste radiazioni sono invisibili all'occhio umano, poiché aventi una frequenza inferiore al nostro spettro di frequenze luminose visibile, ma possono essere rilevate tramite specifici dispositivi elettronici progettati a tal scopo. Il termine passivo si riferisce al fatto che i PIR non emettono energia in nessuna forma ma lavorano esclusivamente rilevando l'energia sprigionata dagli oggetti. È dunque sembrato il componente elettronico più adatto al sistema che stavamo progettando.



Figura 9: Sensore PIR

2.2.2.2 Scheda ESP32



Figura 10: Modulo ESP32

Inizialmente si era pensato di utilizzare la scheda Arduino Uno, che tuttavia essendo sprovvista di un modulo Wi-Fi non soddisfa i requisiti per il sistema da progettare. La connessione internet è fondamentale per la comunicazione tra hardware e Measurify. Per la lettura del sensore è necessario avere una scheda avente dei pin digitali, sui quali effettuare la lettura. La scelta è dunque ricaduta su una scheda Wi-Fi basata su **ESP32** che **integra** un modulo per comunicare con protocollo Wi-Fi 802.11 b/g/n, un modulo per comunicare via Bluetooth dual-mode (classico e BLE) e 34 pin digitali. **Supporta** una velocità di trasmissione di dati fino a 150 Mbps, ha una potenza di uscita sull'antenna pari a 20,5 dBm per garantire la massima portata.

Inoltre **dispone** di interfacce per sensori di temperatura, touch sensor, SD card, UART, SPI, SDIO, I2C, LED PWM, Motor PWM, I2S, IR, connettore micro USB, pulsante di Boot e pulsante di reset.

2.2.2.3 Componenti elettronici di contorno

I componenti appena mostrati sono i principali componenti elettronici necessari allo sviluppo del progetto, ma non solo i soli. È stato ovviamente necessario l'utilizzo di ulteriori componenti quali:

- Una breadboard, strumento utilizzato per creare prototipi di circuiti elettrici. A differenza della basetta millefori, che è base di rame con tanti fori (da qui il nome) su cui vengono saldati i componenti e i collegamenti che formano il prototipo, la breadboard non richiede saldature ed è completamente riutilizzabile poiché è strutturata in modo da avere delle linee di continuità che servono a collegare i componenti in modo “plug and play”.
- Un alimentatore per poter fornire l'energia necessaria al sistema
- Jumper maschio-femmina

Tutti gli strumenti hardware sopracitati sono mostrati in figura11:

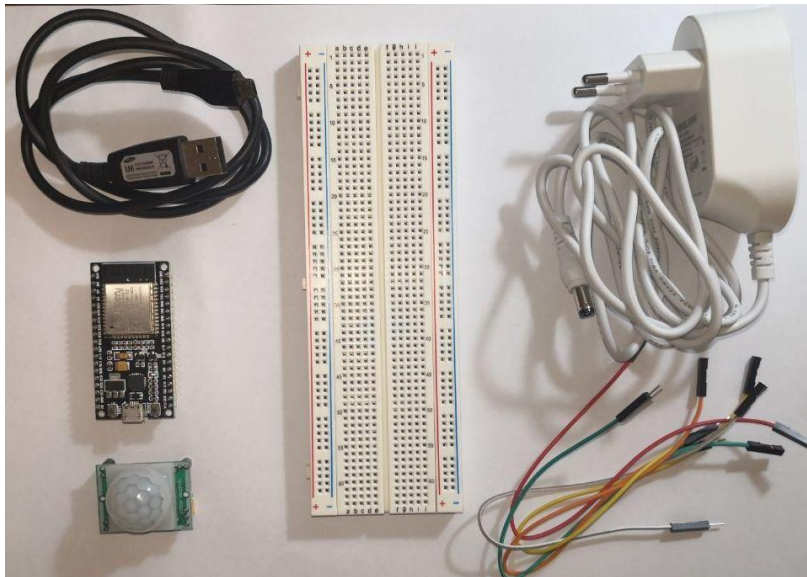


Figura 11: Componenti hardware del sistema

2.3 Modulo portatile: Applicativo smartphone

Come modulo portatile si è scelto banalmente lo smartphone, il quale funziona sia da lato hardware che software ed è un dispositivo portatile in più che l'utente ha già con sé. Pertanto, non avendo il problema della scelta della componentistica hardware ci si è potuti concentrare **solo** sul **lato software** e si è scelti di sviluppare un'applicazione utilizzando il **Framework Flutter** [1].

2.3.1 Lato software: Framework Flutter

Flutter è un framework open-source creato da Google per la creazione di interfacce native per iOS e Android. Flutter ha quattro componenti principali, che costituiscono la sua struttura:

- Dart Platform: è un linguaggio di programmazione sviluppato da Google. Il compilatore Dart permette di scrivere programmi sia per il web sia per desktop e server
- Flutter Engine: è scritto principalmente in C++, fornisce supporto per il rendering a basso livello utilizzando la libreria grafica di Google, Skia Graphics. Inoltre, si interfaccia con SDK della piattaforma specifica come quelli di Android o iOS. Una particolarità molto apprezzata del Flutter engine, grazie al codice scritto in Dart, è quella di poter effettuare un "hot-reload" dell'applicazione dove la modifica del codice viene iniettata immediatamente all'interno dell'applicazione così da visualizzare all'istante le modifiche effettuate senza un riavvio completo o un cambio di stato.
- Foundation Library: scritta in Dart, fornisce classi e funzioni di base utilizzate per costruire applicazioni che utilizzano Flutter, come le API per comunicare con l'engine.
- Widgets: la progettazione dell'interfaccia utente in Flutter prevede l'assemblaggio e/o la creazione di vari widget. Un widget in Flutter rappresenta una descrizione immutabile dell'interfaccia utente; grafici, testo, forme e animazioni vengono creati utilizzando i widget. Inoltre il framework di Flutter contiene due set di widget conformi a specifici linguaggi di programmazione. I widget in stile Material Design implementano il design di Google, mentre i widget in Cupertino imitano il design iOS di Apple.

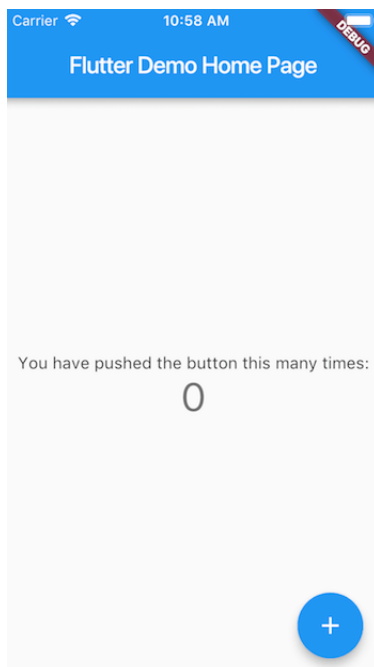


Figura 12: Esempio Flutter App

Per poter programmare in Flutter c'è bisogno di un editor di testo, come VsCode [2] o Android Studio [3]. È stato scelto di utilizzare questo Framework perché grazie ad esso si ha "preinstallata" la caratteristica della multiplatforma, quindi in questo modo si potuto programmare l'app una sola volta e in un linguaggio relativamente semplice poiché molto intuitivo.

2.3.1.1 Linguaggio Dart

Come detto in precedenza, il linguaggio Dart è un linguaggio di programmazione sviluppato da Google con sul quale si basa Flutter. Il linguaggio si ispira al Java e al JavaScript e si basa sul concetto di widget, il quale si divide in due categorie:

- Stateless Widget: sono widget immutabili, ovvero significa che le loro proprietà non possono cambiare nel tempo e che tutti i valori sono quelli finali
- Stateful Widget: mantengono uno stato che potrebbe cambiare durante il tempo di vita del widget. Per implementare tale widget sono necessarie:
 - una classe Stateful Widget che crea un'istanza di una classe State. Questa classe è immutabile e può essere scartata e successivamente rigenerata
 - una classe state che persiste oltre il tempo di vita del widget.

Tramite queste due categorie sono stati creati tutti i widget presenti nell'applicazione.

Inoltre, è importante sottolineare che in Flutter è possibile importare pacchetti, per permettere estensioni come Firebase o altre funzionalità. Per fare ciò è sufficiente scrivere nel programma un comando come

```
import 'package:flutter/material.dart';
```

e specificare nel file pubspec.yaml tale dipendenza dal pacchetto. L'istruzione riportata come esempio è presente in tutti i programmi Flutter, essendo il pacchetto che permette di utilizzare il design MaterialApp().

Tutti i pacchetti sono scaricabili dalla repository Pub.dev. Nel caso specifico del progetto, sono stati utilizzati i seguenti pacchetti:

- **cupertino_icons** per quanto riguarda la possibilità di inserire componenti grafici in stile ios;
- **http** per quanto riguarda la possibilità di effettuare richieste http (per comunicare con Measurify);
- **firebase_messaging** per consentire l'abilitazione delle notifiche push

La figura 13 mostra un file.dart dell'applicazione, scritta utilizzando come editor VsCode:

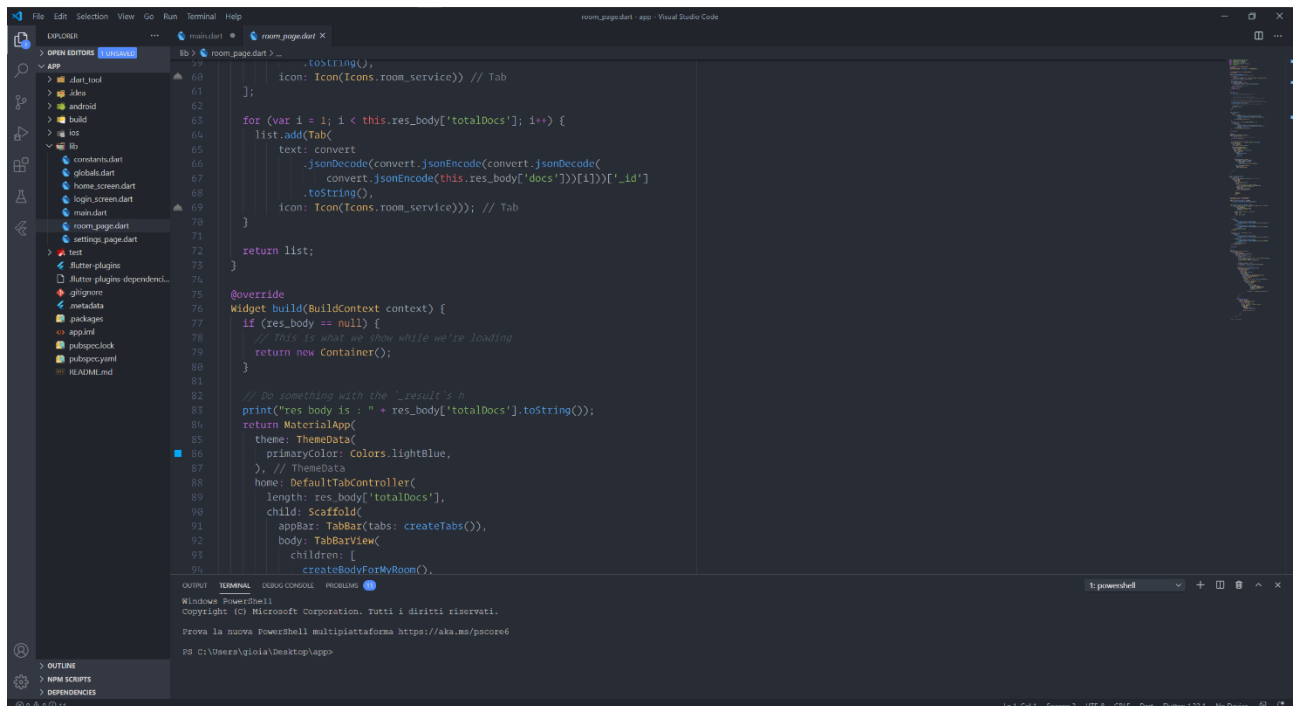


Figura 13: Esempio di un file.dart in VsCode

2.3.2 Comunicazione Measurify-person via mobile: Firebase e Notifiche Push

Il pacchetto **http** consente solo di poter effettuare chiamate verso Measurify ma questo non è sufficiente per implementare le notifiche push ed avere quindi un riscontro in tempo reale delle presenze collezionate. Per tale ragione si è scelto di ovviare al problema utilizzando il tool di Google Firebase Cloud Messaging.

Firebase Cloud Messaging[11] è una soluzione di messaggistica multiplatforma che consente di inviare messaggi in modo affidabile senza alcun costo. Utilizzando Firebase Cloud Messaging, è possibile notificare a un'app client che sono disponibili nuovi messaggi di posta elettronica o altri dati per la sincronizzazione. Nel caso specifico di questo progetto è stato necessario utilizzare questo servizio per avere in tempo reale, tramite messaggi inviati da Measurify, la notifica che era stata registrata una nuova presenza nella stanza.

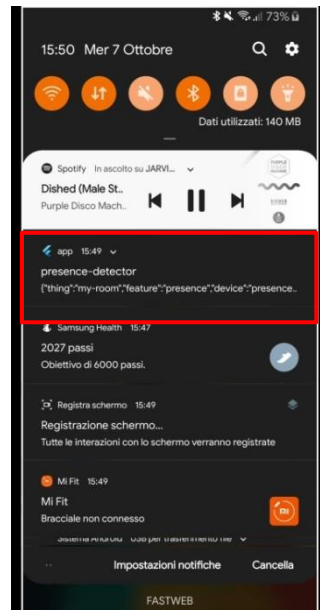


Figura 14: Ricezione notifiche Push da Measurify

Firebase è basato su un sistema a token, nel senso che Firebase genera un token per ogni dispositivo che ne fa richiesta. Questo token, nel caso specifico del progetto, serve per effettuare le subscription ad un "device" di Measurify per le push notification. In questo modo, ogni qual volta che ci sarà una rilevazione della presenza da parte di quel device, Measurify manderà una notifica a tutti i dispositivi che hanno fatto la sottoscrizione.

3 Sperimentazioni e risultati

Nella realizzazione del progetto, si è scelto di iniziare con lo sviluppo prima del modulo fisso e, una volta effettuate tutte le sperimentazioni, si è proceduti poi con lo sviluppo del modulo portatile e quindi dell'applicativo.

3.1 Sviluppo modulo fisso

Per sviluppo di questo modulo era necessario soddisfare 2 obiettivi principali:

- Effettuare la misurazione della presenza
- Impacchettare la misurazione in oggetto "measurement" in formato JSON e mandarlo a Measurify

3.1.1.1 Punto di vista circuitale e scelte per la misura della presence

Il primo step per la realizzazione del modulo è stato la realizzazione circuitale dello stesso. Come già accennato nel capitolo 2 sono stati scelti un sensore PIR per la parte sensoristica, e un modulo ESP32 come scheda programmata e per quanto riguarda la parte della comunicazione con le API via Internet. I due dispositivi sono quindi collegati come mostrato in figura 15:

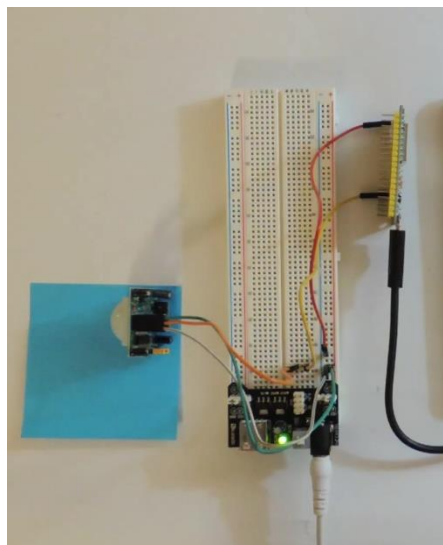


Figura 15: circuito modulo fisso

Per quanto riguarda invece le **scelte per la misura della presence**, è stato scelto di settare alcune variabili del sensore PIR. Difatti il sensore stesso può essere settato in diversi modi grazie ai due trimmer integrati. Tramite il primo è possibile modificare la sensibilità alla distanza che varia da 3 metri fino a 5 metri, mentre con il secondo è possibile modificare il ritardo del segnale, ovvero decidere per quanto il segnale rimanga alto anche dopo che il movimento è terminato: in questo caso il range va da pochi secondi fino a 5 minuti. Il ritardo del segnale può essere ulteriormente personalizzato, modificando il ponticello che è presente insieme ai tre connettori. Sono presenti due posizioni:

- Nella posizione "L" il segnale parte immediatamente dopo il primo rilevamento e non viene più aggiornato fino a quando il timer non scade, portando il segnale a 0V. Se successivamente vengono rivelate altre presenze, il timer riparte.
- Nella posizione "H" re-inizializza il timer ogni volta che viene rilevato un movimento, quindi se viene eseguito un movimento continuo davanti al sensore, il timer continuerà a ripartire ed unicamente alla fine quando non ci sono più movimenti manterrà il segnale alto per il tempo impostato

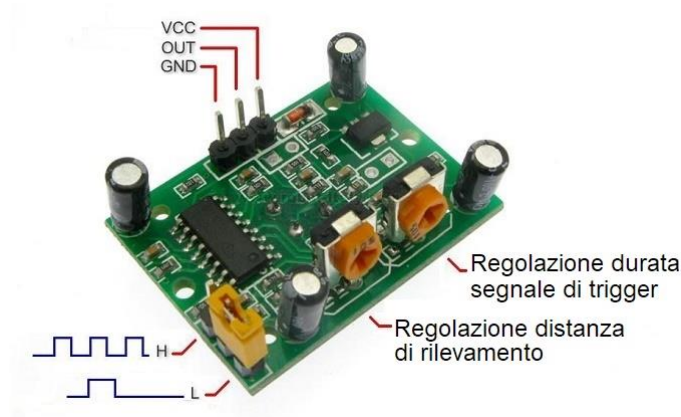


Figura 16: Settaggi sensore PIR

Per tale sperimentazione è stato scelto il livello high, considerando una situazione reale in cui una persona nella stanza fa un movimento nel raggio di azione del sensore più di una volta. Una volta utilizzato un sensore PIR funzionante e settato secondo i parametri richiesti, il sistema di allarme agiva come era richiesto.

3.1.1.2 Punto di vista funzionale e comunicazione con Measurify

Una volta creato il circuito, il passo successivo è stato quello di sperimentare la comunicazione tra il modulo e Measurify. Dal punto di vista funzionale, il processo di invio delle measurement avviene come mostrato in figura17:



Figura 17: Schema funzionale del modulo fisso

A questo punto l'utilizzo della libreria EdgeEngine è stato fondamentale, poiché ha permesso di concentrare il lavoro solo sulla scrittura del codice relativo al processo di collezione dei dati e alla verifica del suo funzionamento, piuttosto che sulla verifica della scadenza del token o sull'inserimento manuale delle presenze nelle "measurement". Di fatto, grazie all'utilizzo della libreria ci si è dovuti occupare solo del blocco 2 in figura17, risparmiando così tempo e lavoro.

Nel concreto con EdgeEngine è basta settare nel file ".ino" le credenziali per l'accesso a Measurify; sarà la libreria stessa a svolgere il compito dell'aggiornamento del token.

Dal punto di vista della programmazione del codice, come si può vedere nella figura8, le credenziali vengono settate nel metodo di setup(). Sempre nella stessa figura si può notare come nel metodo di loop() bastano poche istruzioni per eseguire ultimi due blocchi funzionali in figura17, tutto questo grazie alla libreria.

3.2 Sviluppo modulo portatile

Terminata la progettazione del modulo fisso si è passati alla progettazione del modulo portatile. Come già anticipato nel capitolo due, avendo scelto lo smartphone come supporto hardware, la progettazione del modulo portatile si riduce al solo sviluppo dell'applicazione.

Per la progettazione del modulo portatile si è preferito scegliere, poiché più conveniente, di iniziare strutturando l'app in base alle sue funzioni, per poi concentrarsi sull'effettiva programmazione effettuando infine tutte le verifiche di sorta.

3.2.1.1 Punto di vista funzionale e strutturale

Dal **punto di vista strutturale** l'applicazione si struttura così come mostrato in figura 18:

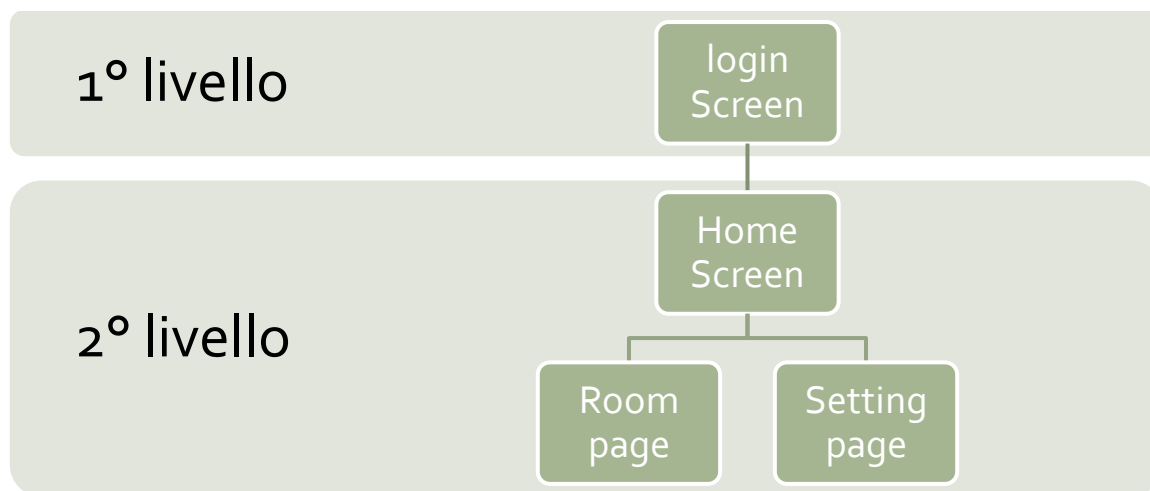


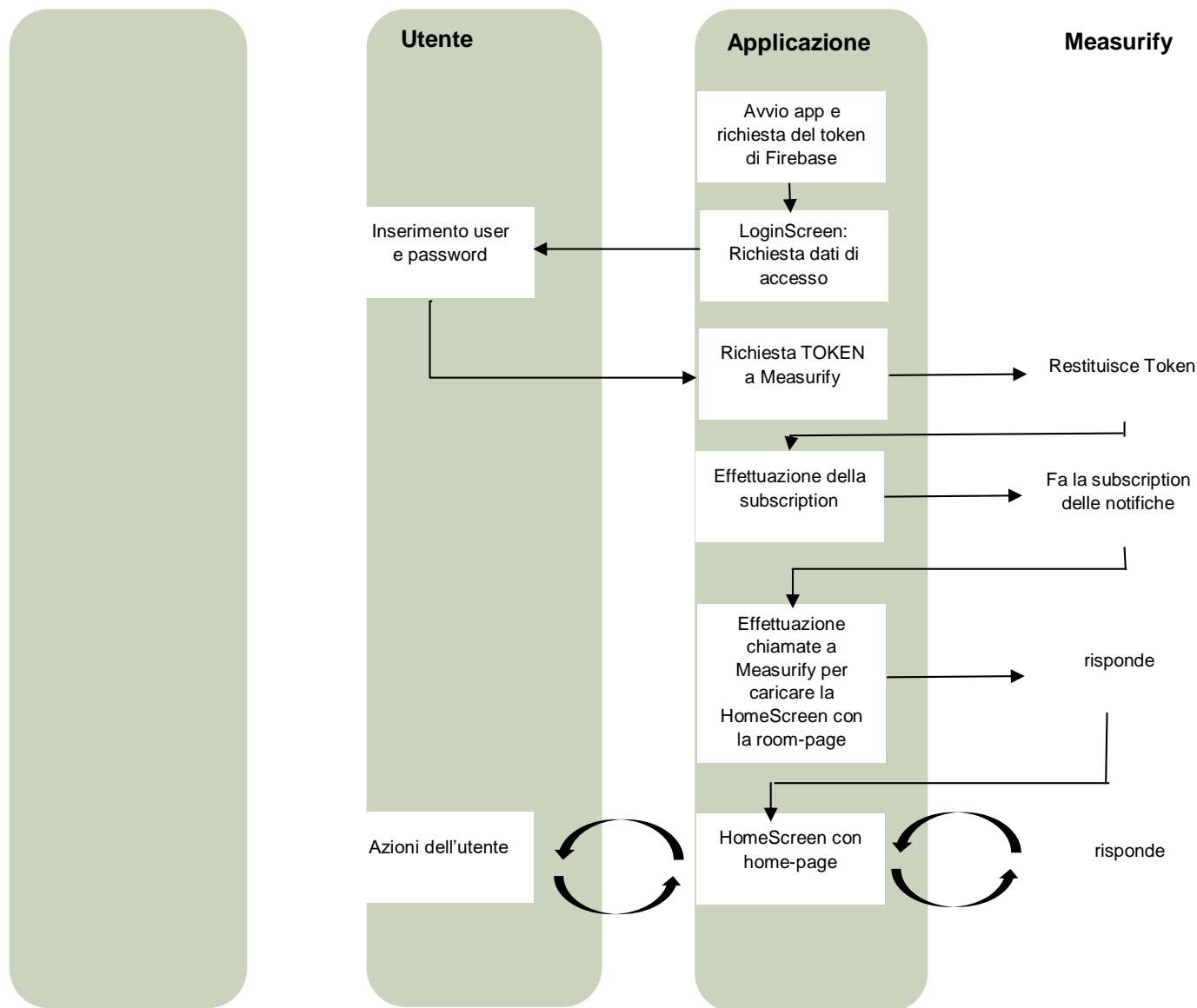
Figura 18: schema strutturale dell'applicativo

L'applicazione, quindi è strutturata in due livelli:

- Il **primo livello** è funzionale alla sola operazione di login. Per tanto ci sarà una sola schermata chiamata loginScreen.
- Il **secondo livello** invece è funzionale alla parte della comunicazione con Measurify una volta che l'operazione di login è stata effettuata con successo. In particolare, vi è una schermata intitolata HomeScreen nella quale, vengono visualizzate a scelta la page per la gestione delle things e la page per la gestione delle notifiche push, la quale però purtroppo non è stata solo sviluppata dal punto di vista grafico non funzionale. La scelta delle page è gestita tramite l'utilizzo del widget "bottomNavigationBar".

Per quanto riguarda lo stile grafico, si è scelto lo stile Android, per tanto l'app è composta da un insieme di classi Stateful Widget (per le schermate e le page) tutti in stile Material.

Dal **punto** di vista **funzionale**, invece il processo dell'applicativo si svolge in questo modo:



Gli ultimi blocchi funzionali rappresentano le “azioni dell'utente”. Da qui in poi quindi, l'utente ha l'app settata in modo tale che possa:

- gestire la parte relativa alla room-page e quindi visualizzare le things, i dispositivi per ogni thing e le notifiche delle misurazioni effettuate per ogni dispositivo
- gestire l'abilitazione delle notifiche push

Ogni volta che l'utente farà un'operazione del genere l'applicazione effettuerà le relative chiamate http a Measurify. Per tanto l'applicazione è progettata affinché possa avere un comportamento che si adatti a cambiamenti dell'intero sistema nel tempo. Basti pensare al caso di eventuali aggiunte di device all'interno della stanza: anche se il modulo fisso cambia non serve intervenire a livello software poiché di fatto il modulo fisso cambia in automatico in questo senso. Per estensione, si può utilizzare l'applicazione anche per progetti futuri, almeno per quanto riguarda la struttura, che vedano l'implementazione delle API di Measurify.

3.2.1.2 Funzionamento concreto dell'applicazione

In base a quanto detto sopra l'app si avvia mostrando la loginScreen come mostrato in figura figura19 in questa schermata cosa succede:

Durante l'avvio della schermata di Login, viene effettuata in background la procedura per ottenere il token di Firebase, il quale, una volta ottenuto viene salvato in memoria. Nella schermata ci sono due "textfield", uno per l'username e uno per la password. Il secondo oscura il testo inserito.

Una volta inseriti i dati e premuto il bottone "login", l'applicazione manda una richiesta POST sulla route /v1/login di Measurify:

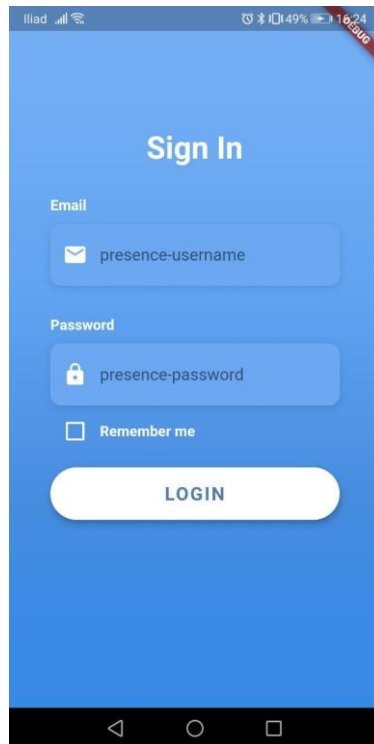


Figura 19: Schermata Login

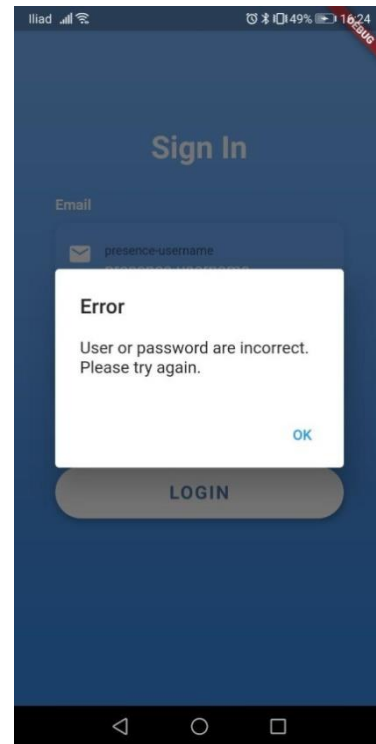


Figura 20: messaggio Errore Login

POST <http://students.atmosphere.tools/v1/login>

A questo punto, se l'operazione di login è andata a buon fine, l'app passa ad effettuare la procedura per effettuare la subscription, altrimenti, se i dati non sono corretti, Measurify ritornerà con una risposta con StatusCode = 401 Unauthorized, e l'app lancerà un messaggio di errore tramite un alert, come mostrato in figura20.

Procedura per la subscription

Dopo una riuscita autenticazione da parte dell'utente, e nel frattempo che la prossima schermata venga lanciata, l'app effettua la chiamata http per connettere il telefono al "device" opportuno in modo da abilitare così le notifiche push.

La chiamata sarà quindi una richiesta POST questa volta fatta sulla route /v1/subscriptions di Measurify:

POST http://students.atmosphere.tools/v1/subscriptions

Con all'interno del body un file JSON costituito da 3 elementi, ovvero la thing e il device cui si vuole collegare il cellulare e il token ottenuto da Firebase, come mostrato in figura21:

```
{
  "token" :
    "cV1HZszTS1C-PMHGjX05eE:APA91bGeuGayJk6yjcSEvUQ7t2oE2mDsnIAQ5VTj8_zGoDAIIYWRJOY7hEi3fN3dBrVG8-32rFI
    F3semaSDWIqfxub7jUMXEVCWq-uJN1i1Z5fLwkSYSB1H4CzRDsZSwrqf8W6cFUY",
  "thing" : "my-room",
  "device" : "presence-detector"
}
```

Figura 21: Body di una request per la sottoscrizione al servizio notifiche push

Da questa chiamata in poi l'applicazione dovrà inserire nel campo Headers la chiave Authorization con dentro il valore del token ottenuto nella schermata di login, altrimenti il dispositivo risulterà a Measurify come non loggato e quindi le API risponderanno con statusCode = 401.

Il token di solito non cambia, ma può accadere, pertanto è convenuto far fare questa operazione ogni volta che viene effettuato il login, in modo da diminuire al minimo ogni possibilità di perdere delle notifiche.

HomeScreen

Alla conferma del login l'applicazione lancia la schermata di Home. Prima di caricare realmente la schermata però l'app effettua tutte le chiamate http verso Measurify necessarie all'ottenimento di quelle informazioni necessarie per costruire la pagina room-page, come numero di thing, numero di device ecc.

Di seguito sono mostrate la figura22, la quale mostra la schermata di HomeScreen impostata su room-page, e la figura23 la quale mostra la cronologia delle misurazioni dopo aver premuto il pulsante "notification":

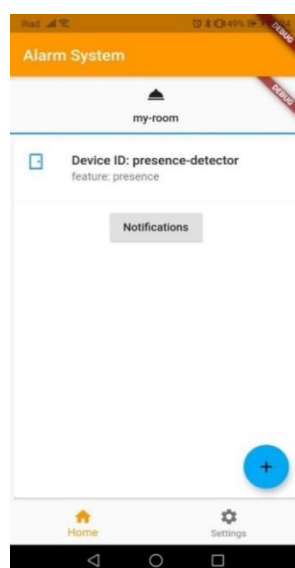


Figura 23: Schermata Home

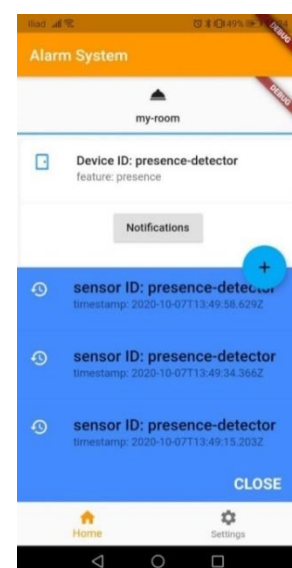


Figura 22: Cronologia delle misurazioni

4 Contributo personale e considerazioni conclusive

Il progetto ha avuto il focus di testare le potenzialità di Measurify, con una attenzione particolare alla comunicazione tra hardware e software. Il contributo personale dato al progetto è stato totale per tutto quello che concerne la comunicazione tra i vari dispositivi utilizzati e l'idea del progetto stesso.

È stato un lavoro stimolante, che ha permesso un approfondimento nel mondo dell'IOT. Grazie a questo progetto è possibile entrare nei meccanismi di progettazione moderna e avere una visione il più aperta possibile su come funzioni in questo momento la gestione di un sistema embedded. Si è rivelato essere non solo un lavoro stimolante, ma anche molto soddisfacente poiché tutti i risultati che erano stati prefissati sono stati raggiunti e tutti con esiti positivi. La parte più interessante è stata proprio sperimentare tutte le diverse funzionalità di Measurify, avvalersi di una libreria come Edge Engine che ci ha permesso un notevole risparmio di lavoro sul codice Arduino, rimanendo tuttavia un progetto basato sulla semplicità.

La parte che ha sicuramente richiesto più lavoro è stato lo sviluppo dell'applicazione. Tuttavia, poter applicare e approfondire nuove conoscenze in un campo che ancora era di nostra conoscenza è stato molto interessante. Flutter si è rivelato essere uno strumento più che adatto per la creazione di applicazioni semplici, ma di impatto.

4.1 Testing della libreria Edge Engine

Come già ampiamente approfondito, l'utilizzo della libreria EdgeEngine ha avuto un ruolo fondamentale nella prima parte della progettazione. È stato possibile concentrarsi su aspetti differenti del progetto e rendere il progetto stesso più complesso e completo. Tramite tale libreria la comunicazione tra sensore e Measurify è stata molto semplificata. Inoltre aver potuto scambiare informazioni con i due tesisti Luca Lazzaroni e Andrea Mazzara, autori di Edge Engine, ha permesso di capire più a fondo come funziona un lavoro in team e come approcciare un prodotto da utilizzatori.

Difatti per quanto riguarda Edge Engine il nostro contributo è stato quello di testare il lavoro fatto dai due tesisti, proprio come utilizzatori finali, e di conseguenza avere un feedback in entrambi le direzioni: per loro, questo progetto ha costituito il primo test con un cliente e un sistema fisico, per questo progetto ha permesso di avvalersi di tutte le funzionalità di Edge Engine usandola come utilizzatori finali.

4.2 Miglioramenti Possibili

Siamo consci del fatto che nella progettazione del sistema abbiamo volontariamente non tenuto conto, per semplicità di esercizio, di alcuni aspetti che nella realtà sono molto importanti e non possono essere trascurati. Un principale aspetto che non abbiamo considerato è quello dei consumi dei due moduli.

Pertanto, il sistema risulta ancora in uno stato iniziale e quindi sicuramente numerosi upgrade possono essere fatti per tutti e due i moduli.

4.2.1 Lato modulo fisso

Per quanto riguarda il modulo fisso, durante la progettazione non si è considerato il fatto che questo avesse un consumo energetico fisso e che quindi molto verosimilmente il modulo deve essere collegato ad una presa di corrente per funzionare in modo stabile.

Considerando quindi i miglioramenti possibili, partendo proprio dall'aspetto energetico si potrebbe equipaggiare il modulo di un'unità energetica d'emergenza, un powerbank dunque, che entri in gioco quando va via la luce (sempre considerando il fatto che in caso di blackout la connessione internet sia comunque disponibile).

Sono possibili anche implementazioni mirate al miglioramento e all'espansione del progetto stesso. Visto che Measurify è in grado di analizzare anche dati stream, un'idea potrebbe essere aggiungere una videocamera, sia utilizzando una scheda Raspberry Pie oppure un modulo Cam Arduino. L'implementazione di una videocamera permetterebbe di avere uno stream video della stanza; l'applicazione non solo notificherebbe la nuova presenza, ma permetterebbe anche di vedere cosa succede nella stanza, in questo modo il sistema sarebbe molto più completo e funzionale.

Ci sono molte altre implementazioni possibili, come l'utilizzo di un cicalino che si attivi nel momento in cui la presenza viene rilevata, dando così anche un aspetto sonoro al sistema stesso e avvertendo l'intruso della presenza di un sistema di allarme nell'abitazione.

4.2.2 Lato modulo portatile

Un aspetto non considerato è stato il consumo di risorse dell'applicazione sullo smartphone. Sicuramente nell'ottica di rendere fruibile al pubblico sarebbe un aspetto da considerare; così come l'aspetto del debugging.

La parte di codice sicuramente da migliorare è quella relativa alla schermata settingsPage, non essendo riusciti in una corretta implementazione della funzione di attivazione delle notifiche push da app. Più in generale il codice può essere strutturato in una maniera più funzionale affinché il corpo sia organizzato in maniera da consentire una maggiore leggibilità e una migliore manutenibilità.

Ancora, a livello funzionale applicativo i messaggi di errore potrebbero essere gestiti in una maniera più dettagliata.

Tuttavia, avendo settato come punto di vista della progettazione dell'applicazione una prospettiva il più completa possibile, l'applicazione possiede già uno scheletro adattabile a implementazioni future.

5 Riferimenti bibliografici

- [1] «Flutter,» [Online]. Available: <https://flutter.dev>.
- [2] «VsCode,» [Online]. Available: <https://code.visualstudio.com/>.
- [3] «AndroidStudio,» [Online]. Available: <https://developer.android.com/studio/>.
- [4] «Measurify,» [Online]. Available: <https://measurify.org/>.
- [5] «Postman,» [Online]. Available: <https://www.postman.com/>.
- [6] «Arduino,» [Online]. Available: <https://www.arduino.cc/>.
- [7] «Dart,» [Online]. Available: <https://dart.dev/>.
- [8] «Pub,» [Online]. Available: <https://pub.dev/>.
- [9] «Elios,» [Online]. Available: <https://elios.diten.unige.it/>.
- [10] «Firebase(gen),» [Online]. Available: <https://firebase.google.com>.
- [11] «Firebase(mex),» [Online]. Available: <https://firebase.google.com/docs/cloud-messaging/>.