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 2020

Progetto e implementazione di un sistema embedded di controllo remoto

Candidati: William Perri, Lorenzo Salvemini

Relatore: Prof. Riccardo Berta

Sommario

Mike è un'applicazione per dispositivi mobile che permette all'utente di interfacciarsi con un sistema fisico collegato al cancello del giardino o alla porta di casa. Questo sistema funziona da videocitofono, avrà un pulsante che, quando premuto, notificherà l'utente sul suo dispositivo mobile.
A questo punto l'utente potrà visionare sul dispositivo mobile lo stream video del citofono, interagire con la persona che ha citofonato grazie a microfono e speaker installati sul dispositivo ed infine potrà decidere se sbloccare la serratura del cancello/porta.
Il nostro sistema sarà quindi formato da:

- Dispositivo installato all'ingresso
- Applicazione installata sul dispositivo mobile

I due moduli comunicheranno fra loro grazie alle API di Measurify [1] che si occuperà di gestire il lato server del nostro dispositivo e della nostra applicazione, facendo in modo di far comunicare il dispositivo fisico con l'applicazione ogni qualvolta qualcuno suoni il campanello.

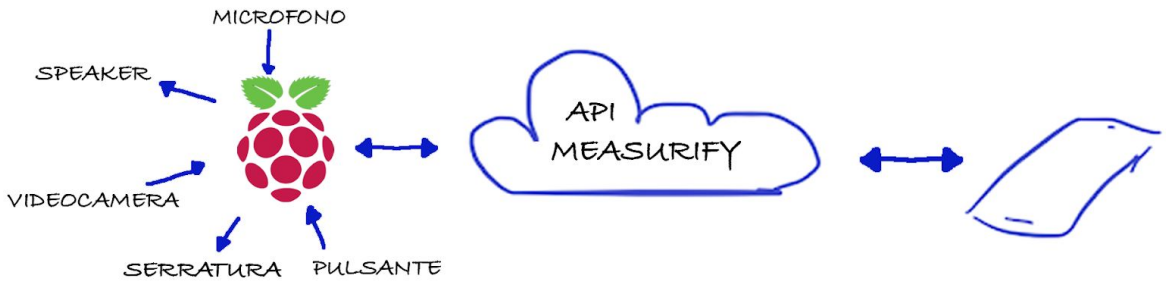
1. Introduzione

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 della 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 Smart Home, 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 con esso: 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 è basata su una nostra necessità. Aspettiamo un pacco, ma questo arriva quando non c'è nessuno in casa. Ci basta solo un sistema per poter aprire il cancello del giardino e permettere al corriere di lasciarlo lì.
Per il nostro sistema ci siamo posti degli obiettivi: deve avvertire l'utente con una notifica sullo smartphone nel caso in cui qualcuno suoni il campanello di casa, a questo punto l'utente può vedere lo stream video di una telecamera posta nel citofono, può comunicare con la persona che ha citofonato ed infine può decidere di aprire la serratura

2. Metodi e strumenti utilizzati

Il nostro progetto è formato un microcontrollore fisicamente localizzato al cancello. Utilizziamo un Raspberry PI [2] collegato a sensori e attuatori, rispettivamente la videocamera, il microfono, il pulsante, lo speaker e la serratura. Il microcontrollore esegue codice scritto in Node.js. Il sistema fisico comunica attraverso richieste HTTP con il server Measurify. Questo svolgerà da elemento intermedio tra il Raspberry e l'applicazione Mike installata sullo smartphone dell'utente.



A. Applicazione Mike

Mike è un'applicazione nativa sviluppata con il framework Flutter.
Flutter [3] è un framework open-source creato da Google per la creazioni di applicazioni native per iOS e Android. Con un successivo aggiornamento, viene introdotto il supporto anche per applicazioni web. Flutter usa Dart [4] , un linguaggio di programmazione creato da Google.

Abbiamo scelto questo toolkit perché:

- permette di sviluppare applicazioni native sia per iOS sia per Android utilizzando lo stesso codice;
- la sua funzionalità *hot reload* permette di, velocemente, vedere i cambiamenti fatti visivamente sull'applicazione con un reload veloce, senza perdere lo stato dell'applicazione;
- l'interfaccia grafica è arricchita dai widget in stile Material Design e Cupertino;
- dà accesso ad una libreria di pacchetti facilmente implementabili [5].

B. Measurify

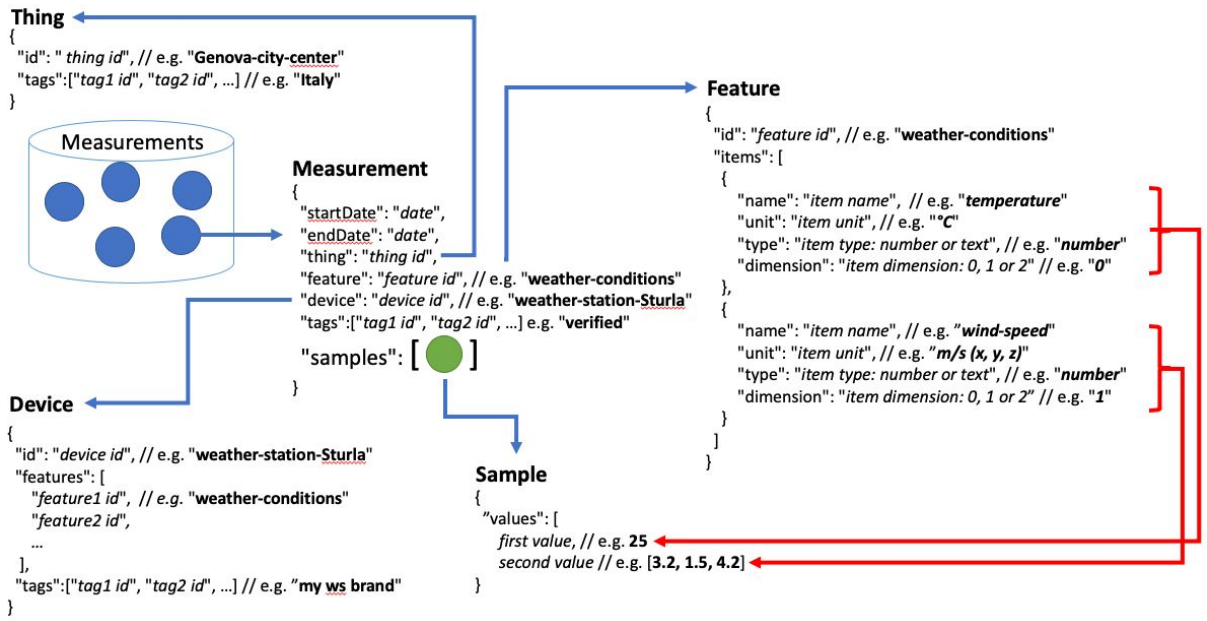
Measurify è un framework improntato al mondo dell'IoT, sviluppato da Elios Lab [6] dell'Università degli Studi di Genova, mette a disposizione delle API cloud-based e orientate alle misurazioni per gestire dispositivi intelligenti in ecosistemi IoT.

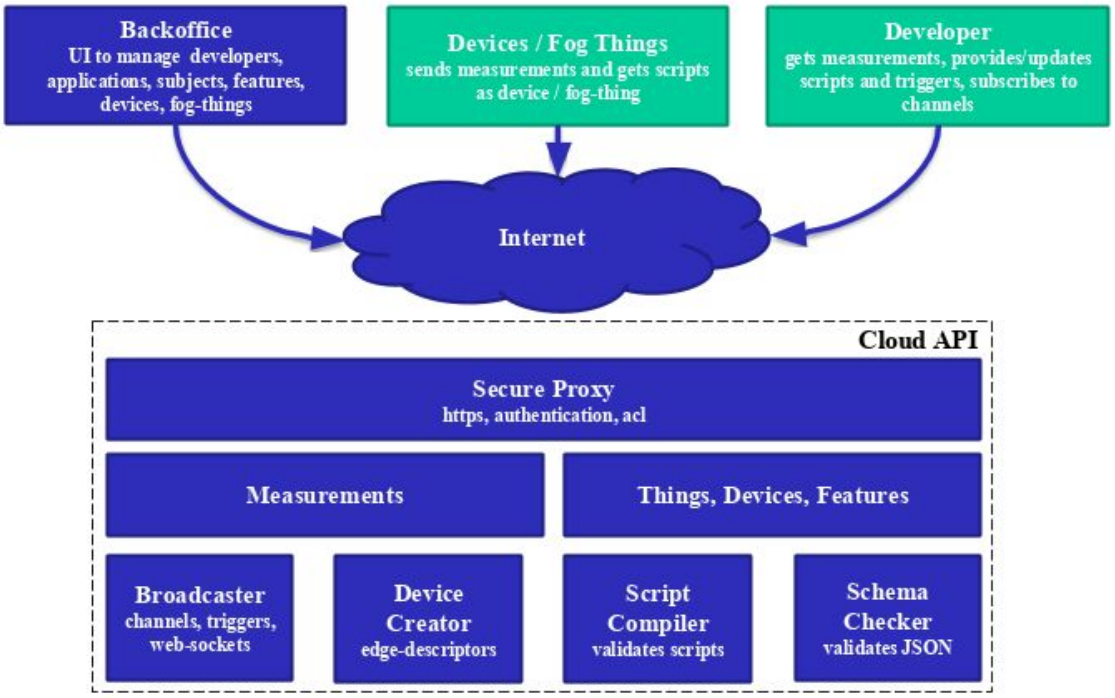
Measurify si concentra sul concetto di **measurement**, questo è un tipo di dato molto comune nel mondo IoT, Measurify è stato pensato per rappresentare il contesto dell'applicazione ed i suoi elementi come oggetti astratti collegati fra di loro.

Questi oggetti sono stati modellati su Measurify come **risorse**, con i propri modelli e funzionalità, accessibili attraverso un set di RESTful API.

Le API di Measurify si basano su alcuni concetti chiave:

THING	una “cosa” (una persona, un ambiente, un oggetto, ecc) per la quale si sta facendo una misurazione. Ad esempio la casa in cui si vuole installare il citofono.
FEATURE	che cosa si sta misurando (la temperatura, l’umidità, il peso, ecc). Nel nostro caso abbiamo creato la feature *entry*, che indicherebbe la citofonata.
DEVICE	un dispositivo (hardware/software) che può misurare una certa dimensione (feature) su una determinata cosa (thing). Ad esempio la videocamera o il microfono del citofono.
MEASUREMENT	una misura effettuata da un dispositivo (device) per una certa dimensione (feature) su una determinata cosa (thing). Ad esempio: il fatto che in casa della persona (thing), sia stata fatta una citofonata (feature) in un certo istante e che lo dica il dispositivo videocitofono (device).





Architettura cloud API

Dall'architettura possiamo vedere come le Cloud API permettano di gestire le measurement, le thing, i device e le feature, a seguito di autenticazione.

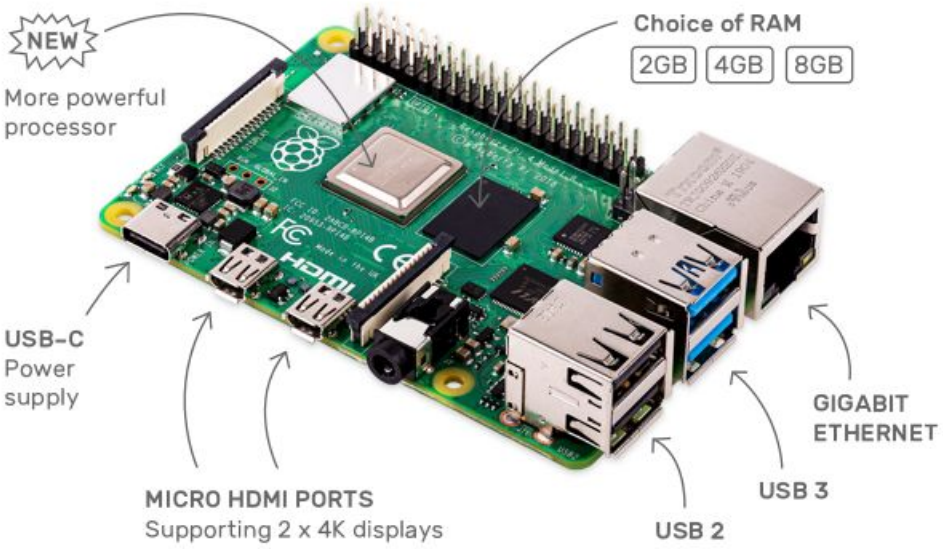
Abbiamo scelto di utilizzare Measurify perché disponeva di tutto ciò di cui avevamo bisogno, dietro autenticazione sulle Cloud API apriva un canale video sul device entryphone, dispone della predisposizione per ricevere notifiche push, parte essenziale del nostro progetto per essere notificati quando il campanello suona (nuova measurement) ed inoltre grazie alle Cloud API è possibile interfacciare anche un microcontrollore che gestisca tutti i dispositivi fisici.

C. Sistema videocitofono

Dalla porta si trova il sistema fisico del videocitofono, formato da un Raspberry PI collegato a vari componenti:

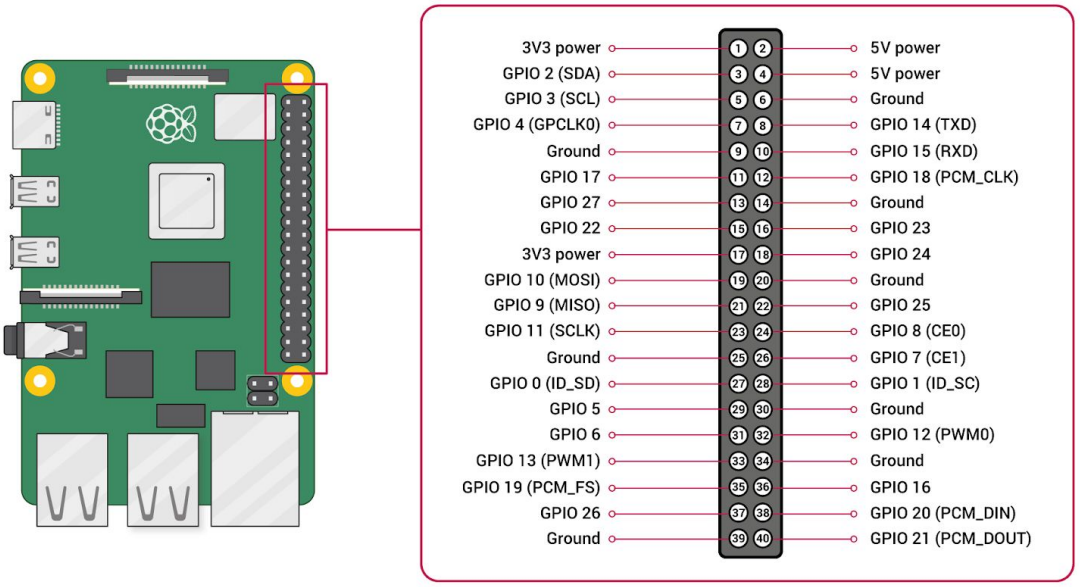
- una videocamera;
- un microfono ed uno speaker per permettere di comunicare;
- un pulsante per citofonare;
- una serratura elettronica che possa essere sbloccata quando riceve un segnale dal Raspberry.

Il Raspberry è un single-board computer molto compatto ed economico, l'ideale per il nostro progetto in quanto facilmente integrabile nel nostro dispositivo fisico ma è anche semplice da programmare in quanto La Raspberry Pi Foundation diffonde ufficialmente sistemi operativi basati su GNU/Linux.

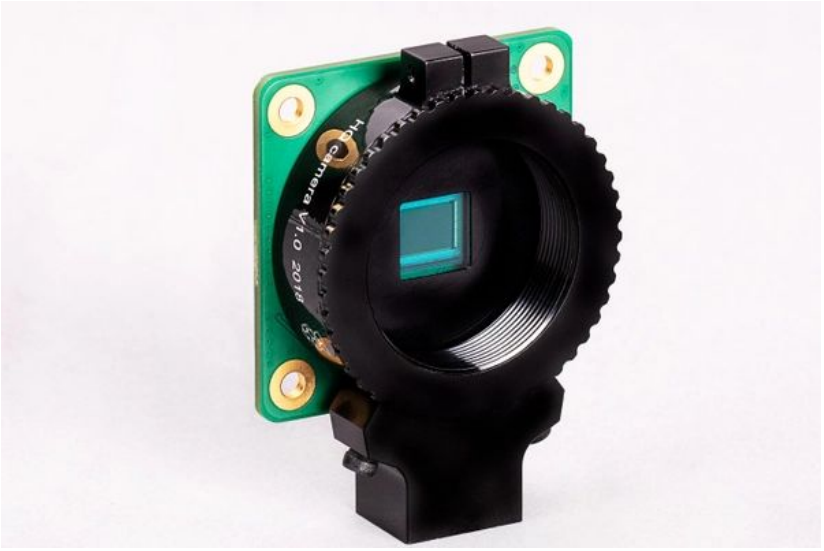


Com’è fatto un Raspberry Pi 4?[7]

Il Raspberry Pi 4 mette a disposizione 40 pin GPIO per poterlo interfacciare con l’esterno



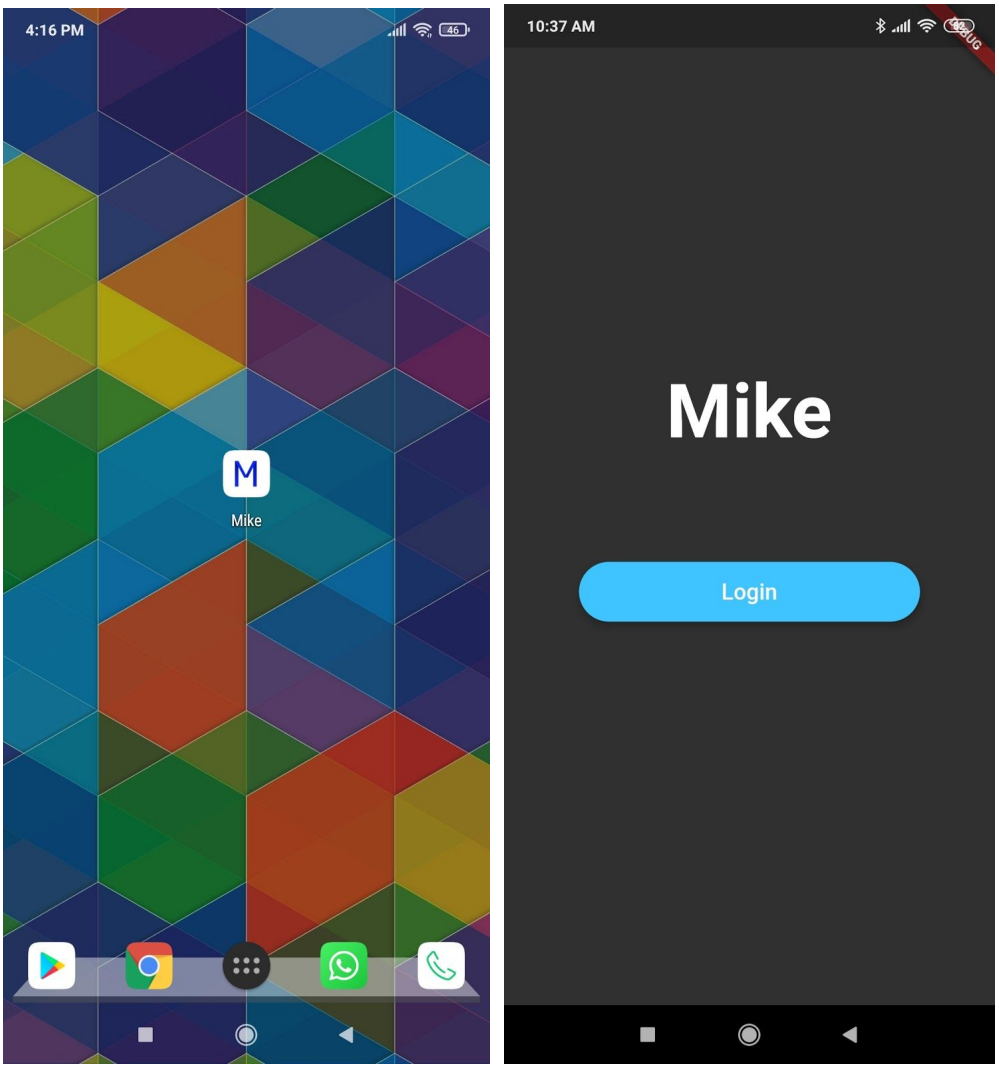
Inoltre la stessa casa produttrice ha recentemente messo sul mercato un modulo fotocamera totalmente compatibile con il Raspberry[8].



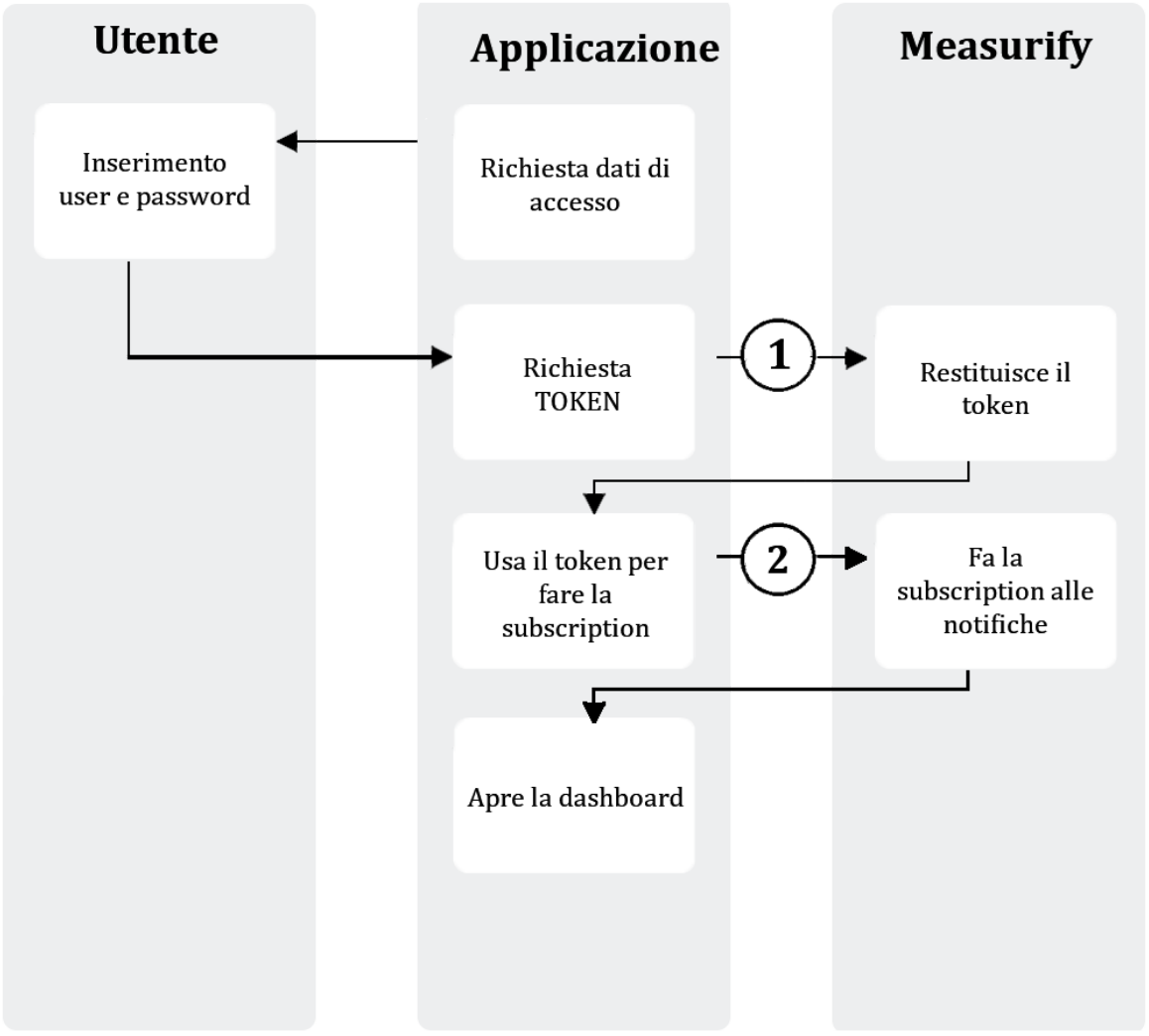
Alla pressione del pulsante, il Raspberry lo rileva e manda una richiesta di POST a Measurify, la richiesta corrisponde ad avere una persona che sta suonando il campanello.
A questo punto Measurify caratterizza questa Measurement con una data ed un'ora e invierà la notifica push agli utenti che avevano effettuato la subscription.
Quando, e se, l'utente vorrà vedere lo stream video del citofono, il Raspberry utilizzerà i moduli esterni installati per permettere di far comunicare l'utente con chi ha suonato il citofono, instaurerà una connessione webRTC con lo smartphone, sul quale l'utente vedrà lo stream video.
A questo punto se l'utente apre la porta con l'applicazione, il Raspberry invia un segnale alla serratura elettronica.

3. Sperimentazione e risultati

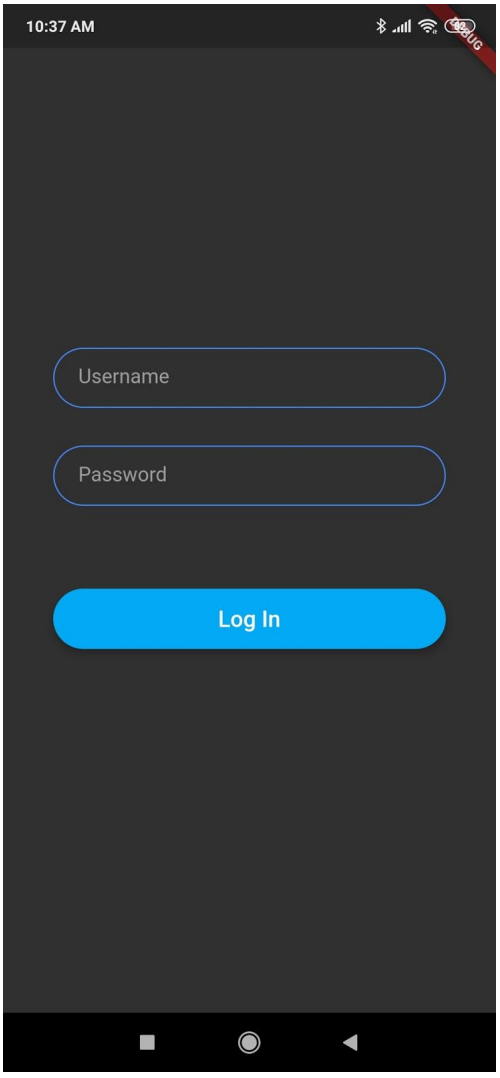
A. Quando si avvia Mike



La prima schermata di Mike mostra, con uno stile minimalista, il nome dell'applicazione e un bottone che trasporta l'utente alla schermata di autenticazione.


Utente
Applicazione
Measurify
Inserimento user e password
Richiesta dati di accesso
Richiesta TOKEN
1
Restituisce il token
Usa il token per fare la subscription
2
Fa la subscription alle notifiche
Apre la dashboard

1. *Richiesta dati di accesso*



Nella schermata di login ci sono due *textfield*, uno per l'username e uno per la password. Il secondo oscura il testo inserito.

Una volta inseriti e premuto il bottone "Log In", una richiesta `POST` viene fatta sulla route `/v1/login` di Measurify:

`POST http://students.atmosphere.tools/v1/login`

Assieme alla richiesta viene mandato come body un file JSON con lo username e password.

```
{
   "username" : "entryphone-username",
   "password" : "entryphone-password"
}
```

Una corretta autenticazione concluderà con in ritorno dal server un oggetto JSON con un campo "token". Questo token verrà usato come autenticazione per tutte le future richieste a Measurify.

```
{
   "token":
"JWTeyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjp7InN0YXR1cyI6ImVuYWJsZWQiLCJfaW
QiOiI1ZTRmOGI2YWY2ZmRjYjM0ZGQyY2I0NDMiLCJ1c2VybmFtZSI6ImVudHJ5cGhvbmUtdXNlcm5
```

```
hbWUiLCJwYXNzd29yZCI6ImVudHJ5cGhvbmUtcGFzc3dvcmQiLCJ0eXBlIjoicHJvdmlkZXIiLCJfX3Yi
OjB9LCJ0ZW5hbnQiOnsicGFzc3dvcmRoYXNoIjpmYWxzZSwiX2lkIjoiYXRtb3NwaGVyZS1wcm9kIn0s
ImlhdCI6MTU5NDIzNjEzNCwiZXhwIjoxNTk0MjM3OTM0fQ.2blBzvyIy6iwIKY-jX-Gu_njFjj_bSCKyQ2vy
son1og"
}
```


Nel caso in cui invece username e/o password non fossero corrette, la risposta alla richiesta avrà uno stato `401 Unauthorized` :


```
{
  "status": 401,
  "value": 4,
  "message": "Authentication error",
  "details": "Incorrect username or password"
}
```


In questo caso l'applicazione notificherà che la password o lo username è incorretta tramite un *alert*.

2. *Subscription*

Dopo una riuscita autenticazione da parte dell'utente, in background la app utilizzerà il servizio Firebase Messaging [9] per sottoscriversi alle *push notifications* e ottenere token di Firebase. Questo token viene usato per inviare le notifiche all'applicazione. Quindi l'applicazione per sottoscriversi ad un determinato *device* di Measurify deve inviare quest'ultimo il token ottenuto
.
La subscription viene effettuata mediante una richiesta `POST` sulla route `/v1/subscriptions` di Measurify:

`GET http://students.atmosphere.tools/v1/subscriptions`

Questa richiesta necessita di due parametri nel body:

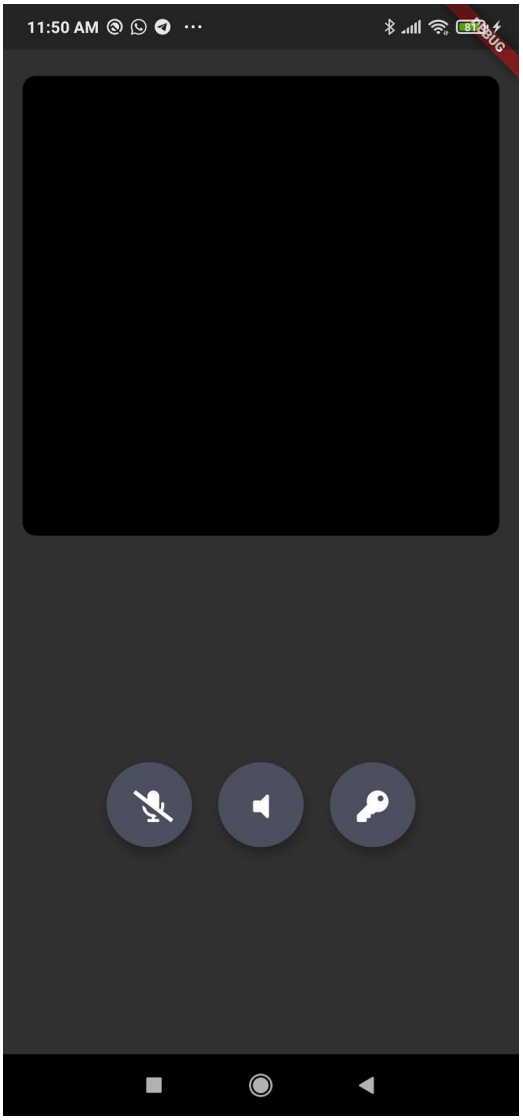
- Il token generato da Google Firebase
- Il *device* per cui il mio dispositivo deve ricevere le notifiche (nel nostro caso entryphone)


```
{
  "token":
"fl8WoZbk3zE:APA91bFA46tVSEYy4OxxvSm_Vo9W-mPmkOlzfmhYsdh4gBVAmYSrnCrsOv6tlgXJY
H7VrxbkNPhVcqefo36REcAZuhpVEPbVCb5dzwY_XTS_NPT8gg-gj4ZpwyZFgUxbMIktkqI6h1qw",
  "device": "entryphone"
}
```


HEADER KEY	VALUE
Authorization	*Required* Token consegnato all'applicazione dopo il login.
Content-Type	*Required* Deve contenere `"application/json"`

Grazie a questo servizio offerto da Google è possibile far arrivare le notifiche anche quando l'applicazione non è aperta sul dispositivo, le cosiddette *push notifications*.
A livello di sviluppo, Firebase Cloud Messaging [10] (FCM) ha bisogno solamente di avere un file (google-services.json) all'interno della nostra applicazione (android / app) per poter funzionare.

Il token di un dispositivo solitamente non cambia, ma può accadere, pertanto la nostra applicazione ad ogni avvio farà nuovamente la subscription con il token attuale, in maniera tale da diminuire al minimo ogni possibilità di perdere delle notifiche.
Alla conferma della subscription la finestra di login si chiude per lasciare spazio alla dashboard principale della nostra applicazione:

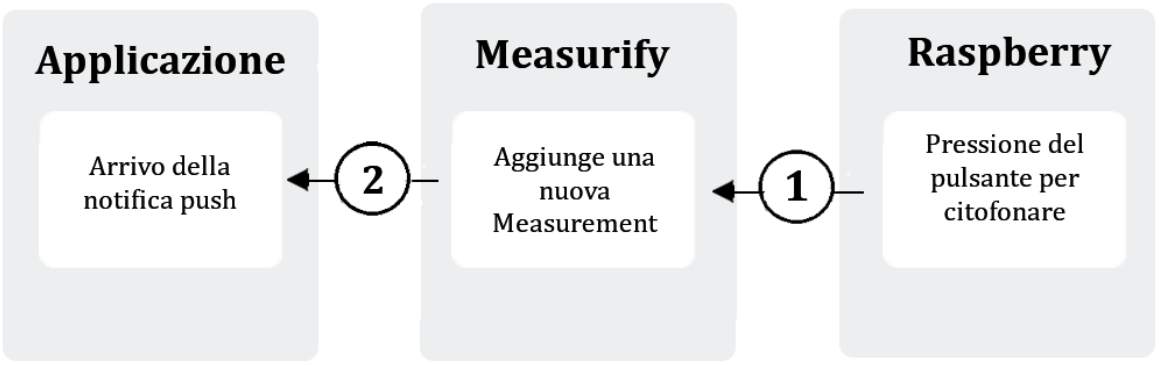


Abbiamo scelto un'interfaccia minimalista ma funzionale, per renderla il più *user-friendly* possibile, facile da comprendere ma allo stesso tempo che non gli manchi nulla per il nostro scopo.
Possiamo vedere quattro semplici elementi che rappresentano il cuore della nostra applicazione:

- il riquadro nero è quello dove andrà poi il video della videocamera utilizzata come videocitofono;
- il pulsante a sinistra viene utilizzato per disattivare o attivare il microfono per poter comunicare con la persona che sta citofonando;
- il pulsante al centro permette di attivare/disattivare l'audio che riceviamo dal citofono;
- l'ultimo pulsante permette di aprire il cancello/porta (viene spiegato in 3.III nei dettagli).

Quando si apre la schermata, l'applicazione cercherà di aprire una connessione audio/video con il videocitofono. Questo avviene utilizzando sempre Measurify come interfaccia. Per la trasmissione viene usato il framework WebRTC.

B. Quando viene suonato il campanello



1. *Viene creata una nuova misurazione e aggiunta su Measurify*

Quando la persona alla porta preme il pulsante che svolge il compito di campanello, il sistema invia una richiesta HTTP `POST` al fine di creare una *measurement* all'interno di Measurify:

`POST http://students.atmosphere.tools/v1/measurements`

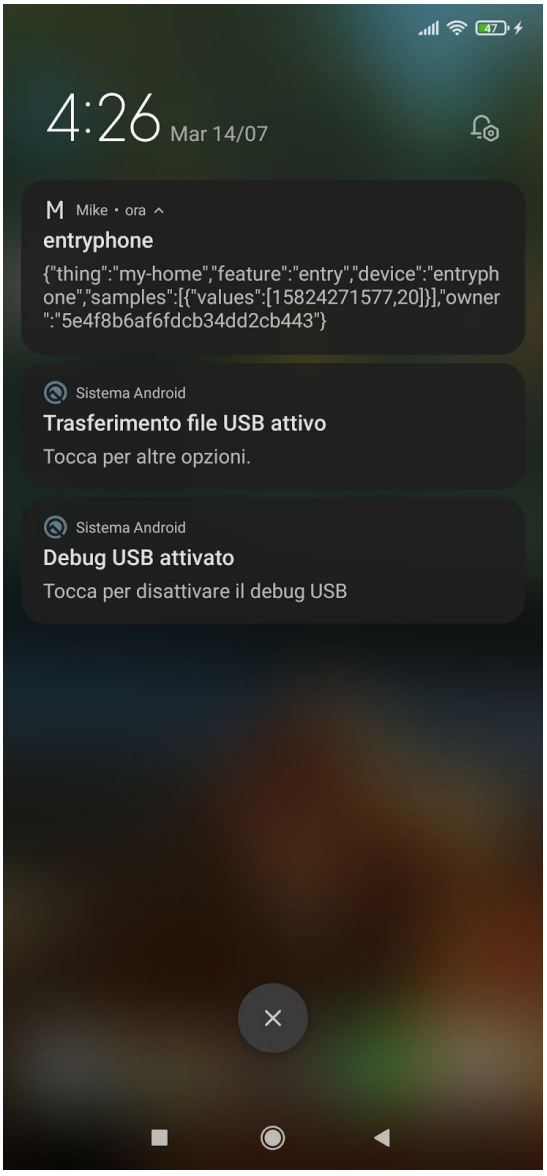
Nel *body* della richiesta viene specificato per quale *device* della determinata *thing* aggiungere una misurazione relativa alla *feature*. Nel nostro caso è il citofono a fare la misurazione, che viene identificata come *entry*. Nel campo *samples* vengono indicate i valori della determinata misurazione.

```
{
  "thing": "my-home",
  "feature": "entry",
  "device": "entryphone",
  "samples": [ { "values": [1582271577, 20] } ]
}
```

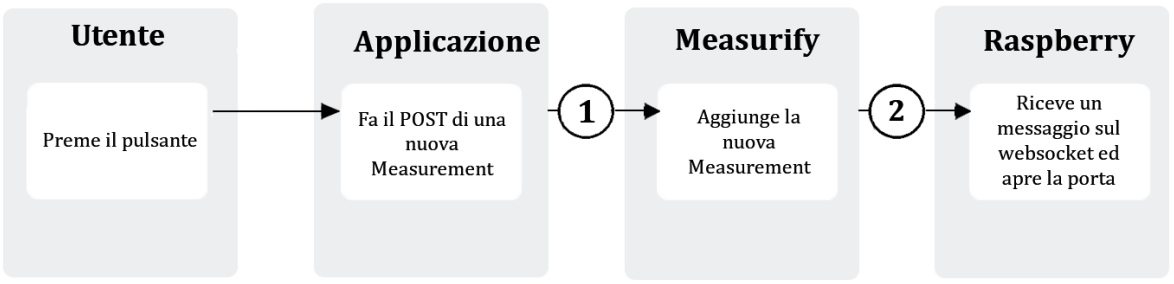
2. *L'utente viene notificato*

Quando Measurify riceve una nuova *measurement* del dispositivo *entryphone*, quindi quando qualcuno suona il citofono, Measurify utilizzerà Google Firebase per inviare la notifica a tutti i dispositivi precedentemente sottoscritti.

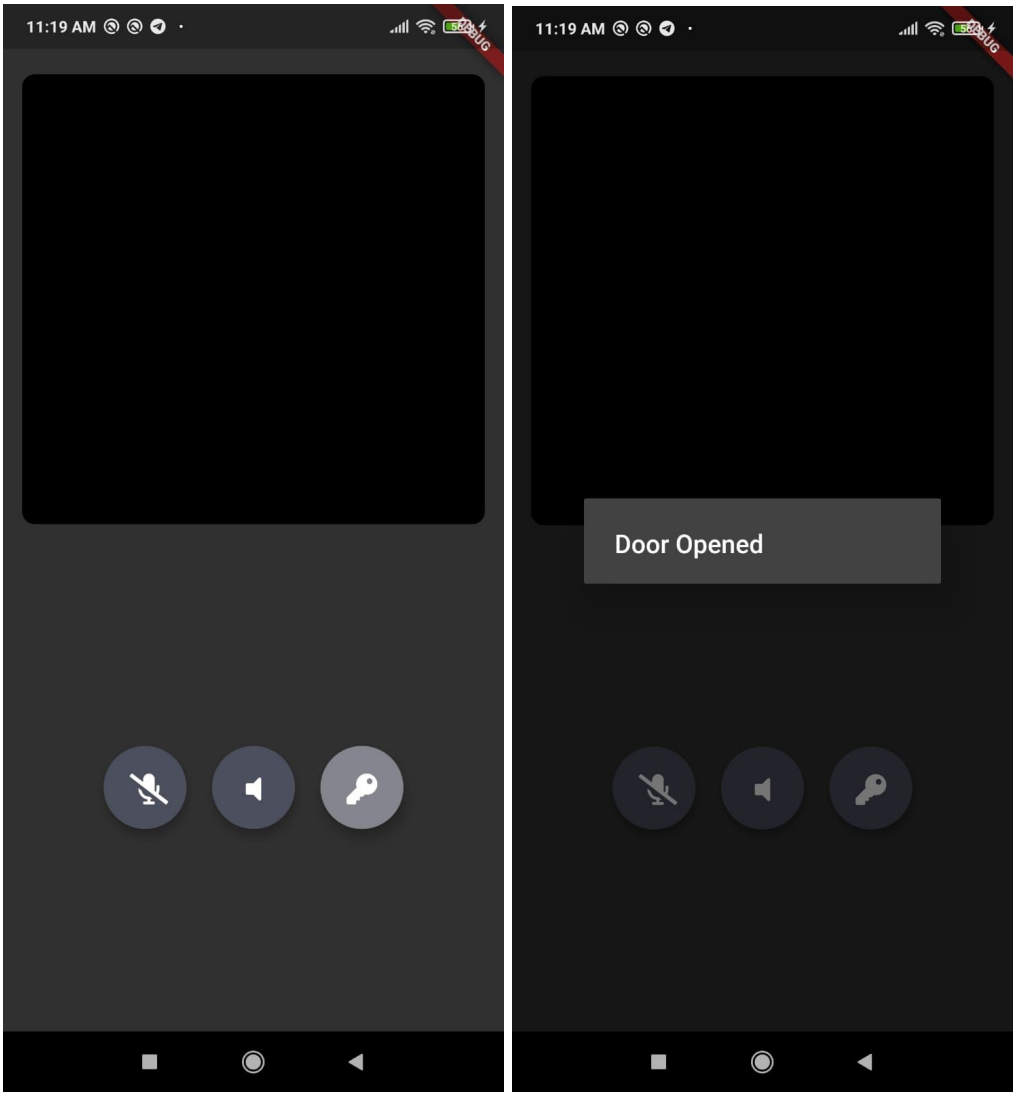
A questo punto l’utente riceve la notifica e cliccandola potrà passare alla nostra applicazione.



C. Quando si apre la porta da Mike



1. *L'utente preme il bottone con l'icona della chiave*



Quando l'utente preme il pulsante per aprire la porta, l'applicazione invia una richiesta `POST` a Measurify per aggiungere una nuova misurazione relativa al *device* `door-opener` e *thing* `lock`:

```
{
  "thing": "lock",
  "feature": "opener",
  "device": "door-opener",
  "samples": [ { "values": [] } ]
}
```

```
static Future<String> postALockMeasurement() async {
  String body = jsonEncode({
    "thing": "lock",
    "feature": "entry",
    "device": "door-opener",
    "samples": [
      {
        "values": [1, 1]
      }
    ]
  });

  Map<String, String> headers = {
    "Authorization": authToken,
    "Content-Type": "application/json"
  };

  String url = "http://students.atmosphere.tools/v1/measurements";
  http.Response response = await http.post(url, headers: headers, body: body);
```

2. *Il sistema fisico sblocca la serratura*

Il Raspberry PI all'accensione apre una connessione Websocket col server per sottoscriversi alla thing `lock` :

```
  let ws = new WebSocket(
            "ws://students.atmosphere.tools/v1/streams?thing=lock&token={{token}}"
          );
  ws.onopen = () => console.log("websocket connection open");
  ws.onmessage = (event) => {
     console.log(event.data);
  };
```

Il Raspberry è ora in ascolto. In questo modo, quando viene aggiunta una nuova misurazione sul *device* `door-opener` che fa parte della *thing* `lock` il Raspberry riceverà la Measurement e potrà inviare al modulo fisico della serratura il segnale per aprirla.

4. Contributo personale e considerazioni conclusive

Applicazione Mike

Vista la necessità di un'interfaccia per smartphone, abbiamo scelto di sviluppare un'applicazione nativa in Flutter. Questa scelta è stata dettata dalle comodità che questo framework mette a disposizione. Non conoscendolo però, abbiamo dovuto studiarlo e poi scrivere l'applicazione. Per la scrittura del codice abbiamo usato interamente l'IDE Visual Studio Code [11] , che ci ha permesso di avere un singolo ambiente di lavoro sia per il codice in Node.js sia per l'applicazione in Flutter. Per testare l'applicazione abbiamo utilizzato sia l'emulatore Android di Android Studio [12] sia un device reale collegato via USB al computer. Mediante il comando `flutter run` l'applicazione veniva installata sul device collegato e avviata in modalità debug.

Per scrivere l'applicazione abbiamo dovuto studiare e mettere in pratica come fare richieste HTTP ad un API. A tal fine abbiamo inizialmente usato Postman [13], per fare prove e tentativi, come ausilio allo sviluppo. Questo ci ha permesso di concentrarci sulle singole richieste per poi metterle all'interno del codice e, inoltre, testare che le richieste fatte sull'applicazione fossero andate a buon fine.

Una parte che era per noi fondamentale per il corretto funzionamento dell'applicazione, era l'implementazione delle *push notification*. Per fare ciò abbiamo dovuto fare utilizzo di Firebase Console Messaging e abbiamo dovuto studiare come introdurre la funzionalità.

Raspberry PI

A causa di una mancanza fisica del dispositivo, abbiamo dovuto farne a meno. Abbiamo scritto il codice in Node.js in modo tale che potesse girare sia su un Raspberry sia sul computer per poterlo testare. Per questo motivo abbiamo utilizzato una pagina mostrata sul browser come interfaccia utente. Visto che non avevamo mai fatto, o poco, lavoro in questo campo, abbiamo dovuto ricercare come scrivere in html (abbiamo usato il template engine handlebars, che ha una sintassi simile a html) e aggiungerlo all'interno del codice in Node.js.

Per quanto riguarda la trasmissione audio/video tra il Raspberry e lo smartphone, abbiamo sfruttato un framework apposito, WebRTC.

Considerazioni conclusive

Il risultato non coincide con quello che avevamo progettato a causa della mancanza fisica del dispositivo fisico: il Raspberry. Tuttavia, abbiamo fatto di necessità virtù e questa mancanza ci ha portato a studiare le interfacce web e come inserirle in un progetto in Node.js. Questo è risultato molto interessante.

Ci sono altre cose che si potrebbero aggiungere per avere una migliore esperienza per l'utente:

- Mike all'avvio potrebbe controllare che l'ultimo token che ha ricevuto sia ancora valido, in tal caso non sarebbe necessario compiere un'altra volta il login.

- Quando la porta riceve il comando di aprire la porta, e dopo aver fatto ciò, potrebbe aggiungere una nuova measurement che comunichi che l'apertura è avvenuta oppure che c'è stato un errore. Per adesso l'applicazione Mike controlla solo che la sua misurazione sia aggiunta con successo sul server.

- un log delle citofonate avvenute.

- un lettore RFID collegato al Raspberry in modo tale da sostituire le chiavi,

Siamo soddisfatti di quello che abbiamo fatto finora. Abbiamo imparato e scoperto cose molto interessanti. Una tra tutte è stato Flutter che si è rilevato molto comodo da usare, ma allo stesso tempo molto soddisfacente in quanto mette a disposizione delle grafiche che permettono con facilità di ottenere un'interfaccia di una certa qualità. Per il momento non ci siamo concentrati troppo su questo aspetto, però se in futuro vorremo migliorarlo, sappiamo che Flutter ci mette tutto quello che ci serve a disposizione.
Inoltre la funzionalità *hot reload* comporta un notevole risparmio di tempo nello sviluppo dell'interfaccia.

5. Riferimenti bibliografici

[1] https://measurify.org/
[2] https://www.raspberrypi.org
[3] https://flutter.dev
[4] https://dart.dev/
[5] https://pub.dev/
[6] https://elios.diten.unige.it/
[7] https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
[8] https://www.raspberrypi.org/products/raspberry-pi-high-quality-camera/
[9] https://firebase.google.com
[10] https://firebase.google.com/docs/cloud-messaging/
[11] https://code.visualstudio.com/
[12] https://developer.android.com/studio/
[13] https://www.postman.com/