ISTITUTO TECNICO INDUSTRIALE OMAR

Corso di Elettronica e Robotica

JASKARAN SINGH

AIR ONE

A.S. 2023/2024

Indice

Obiettivi

Il progetto presenta diversi obiettivi legati alla misurazione della qualità dell'aria e di altri parametri fondamentali per una corretta valutazione dell'ambiente circostante. Tra i principali obiettivi, vi è quello di sviluppare un dispositivo versatile e preciso in grado di rilevare una vasta gamma di parametri, tra cui temperatura, umidità, pressione atmosferica, luminosità e altri.

Inoltre, si mira a garantire che il dispositivo sia facilmente utilizzabile da parte degli utenti, con un'interfaccia intuitiva e accessibile. Un altro obiettivo importante è quello di integrare il dispositivo con tecnologie moderne per la raccolta e l'analisi dei dati, consentendo agli utenti di prendere decisioni informate basate sui risultati ottenuti.

Parallelamente, si punta a minimizzare l'impatto ambientale del dispositivo, ad esempio ottimizzando il consumo energetico e utilizzando materiali riciclabili e sostenibili. In questo contesto, la realizzazione del case del dispositivo è stata effettuata tramite stampa 3D, che rappresenta un'opportunità chiave per integrare la sostenibilità nel progetto. Difatti, utilizzando la stampa 3D, si possono adottare diverse strategie per massimizzare l'efficienza e ridurre l'impatto ambientale, come la produzione su misura del contenitore, l'utilizzo di materiali riciclati o biodegradabili e la progettazione ottimizzata per migliorare le prestazioni complessive del dispositivo.

Infine, il progetto mira a sensibilizzare e educare il pubblico sull'importanza della conservazione dell'ambiente, utilizzando i dati raccolti per promuovere comportamenti sostenibili e responsabili.

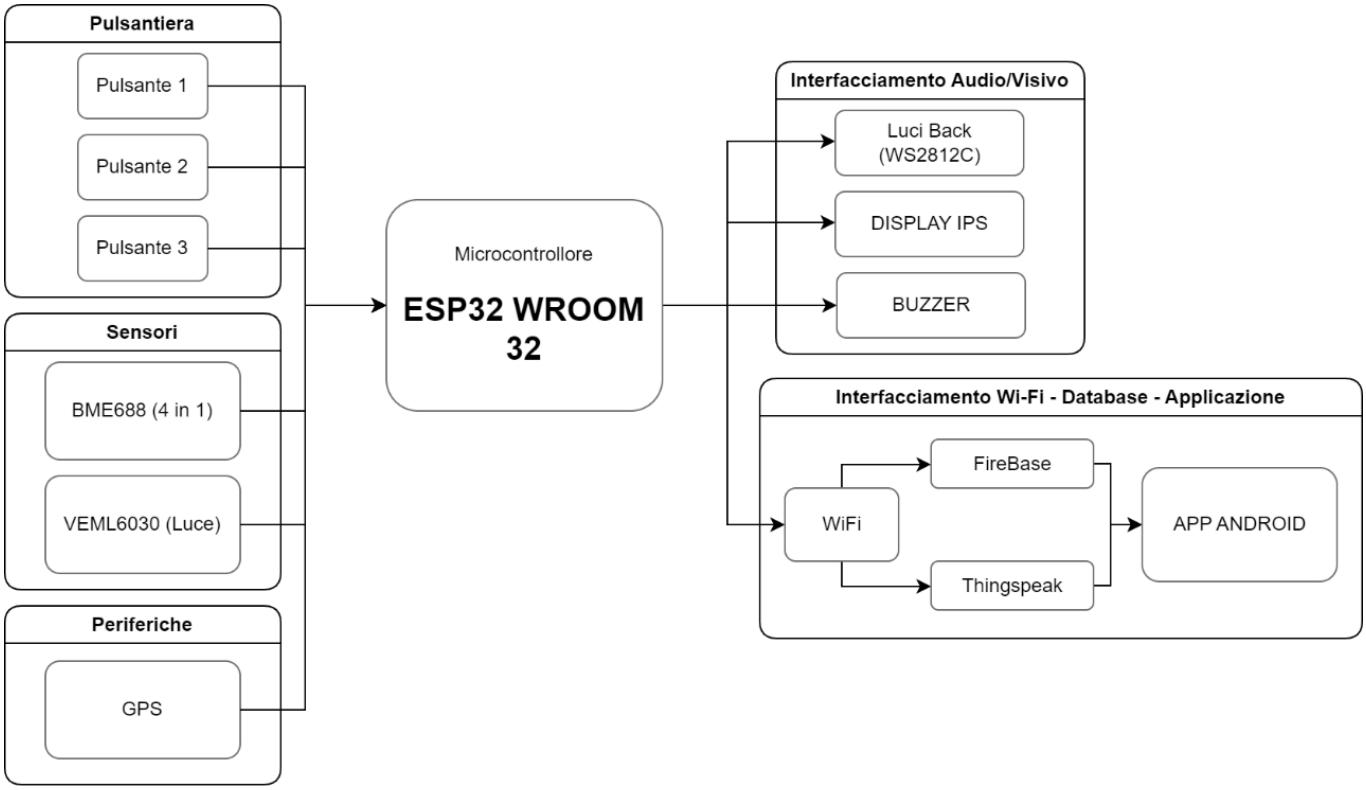
Descrizione

Il progetto proposto si concentra sulla realizzazione di un dispositivo multifunzionale per il monitoraggio ambientale, che comprende la misurazione di parametri come temperatura, umidità, qualità dell'aria, pressione atmosferica e luce. Utilizzando un microcontrollore ESP32, i dati acquisiti vengono trasferiti in tempo reale attraverso una connessione WiFi al Real-time Database di Firebase. Questo approccio garantisce un monitoraggio immediato e continuo delle condizioni ambientali. Inoltre, l'inclusione di un modulo GPS consente al dispositivo di fornire le esatte coordinate geografiche, rendendolo portatile e facilmente localizzabile.

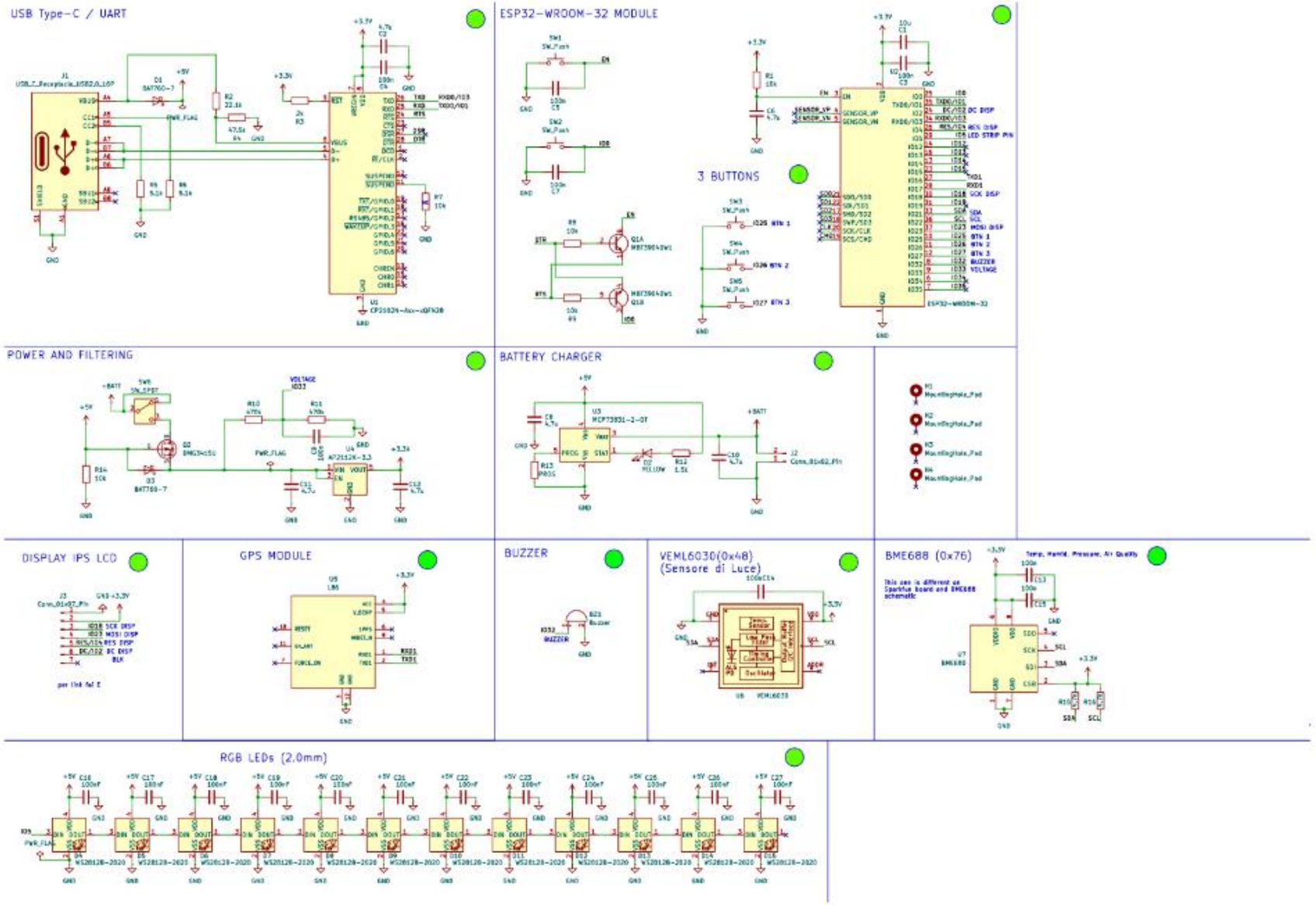
Un elemento chiave del progetto è l'applicazione mobile sviluppata interamente in React Native, una categoria del linguaggio di programmazione “JavaScript”, che funge da interfaccia utente per visualizzare e interpretare i dati raccolti. L'app offre un'esperienza intuitiva e accessibile, consentendo agli utenti di accedere facilmente ai dettagli delle misurazioni e di ricevere grafici con andamento in tempo reale. In aggiunta è presente un interfacciamento audio/visivo utilizzando un Display IPS, un Buzzer e dei LED SMD. Questi sono utilizzati per notificare eventuali problematiche legate alla connettività oppure al superamento di certe soglie numeriche, per quanto riguarda la misurazione dei parametri prima citati.

Infine, l'implementazione di tre pulsanti permettono di navigare facilmente sull'interfaccia presente sul Display, comodo nel caso di una connessione ad Internet non ottimale e per una rapida lettura dei dati.

Schema a Blocchi



Schema Elettrico:



Componenti

BME688 (Temperatura, Umidità, Pressione Atmosferica, Qualità dell'Aria)

Il sensore Bosch BME688 è un componente essenziale nel contesto del progetto di monitoraggio ambientale, poiché offre una soluzione completa per la misurazione della qualità dell'aria e di altri parametri ambientali. Dotato di capacità avanzate, il BME688 integra sensori per la misurazione della temperatura, umidità, pressione atmosferica e gas volatili, consentendo una valutazione accurata e completa delle condizioni ambientali.

Ciò che distingue il sensore BME688 è la sua capacità di rilevare una vasta gamma di gas volatili, inclusi composti organici volatili (VOC) potenzialmente dannosi presenti nell'aria, come solventi, idrocarburi e altri inquinanti. Questa funzionalità lo rende particolarmente adatto per applicazioni di monitoraggio dell'aria interna e esterna, difatti è in grado di misurare i seguenti parametri:

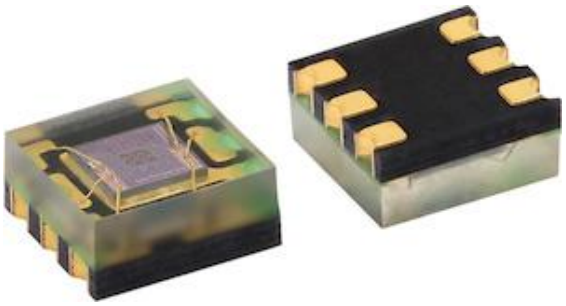
Output	Description
Raw pressure	Raw data from sensor API bypassed to BSEC output
Raw temperature	Raw data from sensor API bypassed to BSEC output
Raw relative humidity	Raw data from sensor API bypassed to BSEC output
Raw gas resistance	Raw data from sensor API bypassed to BSEC output
Sensor-compensated temperature (°C)	Temperature which is compensated for internal cross-influences caused by the BME sensor
Sensor-compensated relative humidity (%)	Relative humidity which is compensated for internal cross-influences caused by the BME sensor
Sensor-compensated gas resistance (Ohm)	Raw gas resistance compensated by temperature and humidity influences.
Ambient temperature (°C)	Ambient temperature after compensating the influence of device (where BME688 is integrated in) heatsources
Ambient relative humidity (%)	Ambient relative humidity after compensating influence of device (where BME688 is integrated in) heatsources
IAQ (0-500)	Index for Air Quality, especially recommended for mobile devices, since the auto-trim algorithm automatically adopts to different environments.
[illegible]	stationary devices (w/ o auto-trimming algorithm)
CO_2 equivalents (ppm)	Estimation of the CO_2 level in ppm. The sensor does not directly measure CO_2, but derives this from the average correlation between VOCs and CO_2 in human's exhaled breath.
b-VOC equivalents (ppm)	Conversion into breath-VOC equivalents in ppm concentration. The scaling is derived from lab tests with the b-VOC gas mixture described in Table 7.
Accuracy status (0-3)	Accuracy status of IAQ
Stabilization time status	Indicates if the sensor is undergoing initial stabilization during its first use after production
Run in status	Indicates when the sensor is ready after after switch-on
Gas (%)	Alternative indicator for air pollution which rates the current raw gas resistance value based on the individual sensor history: 0% = "lowest air pollution ever measured" 100% = "highest air pollution level ever measured"
Gas scan result (%)	The gas scan result is given in % for each of the used classes. In standard scan mode, the probability of H_2S and non H_2S class is provided by the variables GAS_ESTIMATE_1 & GAS_ESTIMATE_2 respectively. A maximum of 4 classes can

Inoltre, il sensore BME688 è caratterizzato da un'alta precisione e stabilità nelle misurazioni, garantendo dati affidabili e coerenti nel tempo. La sua capacità di compensare automaticamente gli effetti di temperatura e umidità ambientale assicura una misurazione accurata anche in condizioni variabili. Infatti, è il primo sensore con funzionalità collaborative con l'IA (in inglese AI, Artificial Intelligence), ossia Intelligenza Artificiale. Queste ultime possono essere attivate utilizzando il programma realizzato dall'azienda stessa, in grado di studiare diverse casistiche di ambientazione per poi metterle in atto durante i periodi di misurazione.

Un altro vantaggio del sensore BME688 è la sua compattezza, infatti l'ingombro è di soli 3.0 x 3.0 x 0.9 mm3, e il basso consumo energetico, lo rendono adatto per l'integrazione in dispositivi portatili e a batteria, come il dispositivo di monitoraggio ambientale proposto. Questo consente di implementare soluzioni di monitoraggio ambientale personalizzate e facilmente accessibili, senza compromettere le prestazioni o l'affidabilità.

VEML6030 (Sensore di Luce)

Il sensore di luce VEML6030 si rivela essenziale all'interno del progetto di monitoraggio ambientale per fornire una valutazione accurata dell'illuminazione circostante. Questo componente si distingue per la sua capacità di misurare la luminosità ambientale con grande precisione e sensibilità, consentendo di rilevare variazioni anche minime nella quantità di luce presente nell'ambiente.



La caratteristica principale del sensore VEML6030 è la sua capacità di operare in una vasta gamma di condizioni di illuminazione, dalle situazioni di scarsa luminosità notturna alle condizioni di forte luce solare. Ciò consentendo una valutazione accurata dell'illuminazione in diversi contesti e ambienti. Infatti, il sensore presenta all'interno un fotodiodo collegato ad un convertitore A/D (Analogico/Digitale), permettendo anche la comunicazione attraverso il bus I2C. Il convertitore da 16 bit installato all'interno permette al sensore di rilevare un range di valori da 0 Lux fino a 140 kLux, con una risoluzione pari a 0.0042 lux/ct.

Inoltre, il sensore VEML6030 offre una risposta rapida e stabile alle variazioni di luminosità, consentendo una raccolta dati affidabile e in tempo reale. La sua alta precisione e sensibilità consentono di catturare anche le variazioni più sottili nella luminosità ambientale, fornendo informazioni dettagliate alla scheda ESP32. Un altro vantaggio del sensore sono le sue dimensioni ridotte: 2 x 2 x 0.87, rendendolo ottimale per circuiti stampati.

Pertanto, grazie alla sua compattezza e al basso consumo energetico, il sensore VEML6030 è facilmente integrabile in dispositivi portatili e a basso consumo, come il dispositivo proposto. Questo consente di implementare soluzioni di monitoraggio ambientale personalizzate e accessibili, che offrono informazioni chiare e dettagliate sull'illuminazione circostante per migliorare il comfort e la sicurezza degli utenti.

GPS Quectel L86

Il GPS Quectel L86 è un componente fondamentale nel progetto di monitoraggio ambientale, poiché fornisce un'accurata geolocalizzazione del dispositivo in tempo reale. Dotato di tecnologia GPS/GLONASS integrata, il Quectel L86 offre una precisa determinazione delle coordinate geografiche, consentendo di tracciare con precisione la posizione del dispositivo in qualsiasi momento.

Una caratteristica distintiva del Quectel L86 è la sua elevata sensibilità e precisione nel ricevere segnali satellitari, che consente una rapida acquisizione del segnale e una stabile connessione GPS anche in condizioni ambientali avverse o in aree urbane dense.

Inoltre, il Quectel L86 offre una bassa potenza di funzionamento, consentendo un utilizzo prolungato senza esaurire rapidamente la batteria del dispositivo. Questo lo rende particolarmente adatto per dispositivi portatili, ottimale nel mio caso. In aggiunta è in grado di fornire l'ora e la data esatta, in modo tale da poter interfacciarlo alle periferiche esterne, ad esempio il Display LCD.

Microcontrollore ESP32 WROOM 32

Il microcontrollore ESP32 WROOM 32 è un componente fondamentale all'interno del progetto di monitoraggio ambientale proposto. Caratterizzato da elevate prestazioni e versatilità, l'ESP32 WROOM 32 è dotato di un potente processore dual-core Xtensa a 32 bit, che offre prestazioni elevate e una grande flessibilità per gestire una vasta gamma di operazioni. La presenza di connettività Wi-Fi e Bluetooth integrata consente una facile comunicazione e trasmissione dei dati, consentendo al dispositivo di essere connesso in modo rapido e affidabile a reti wireless e altri dispositivi compatibili.

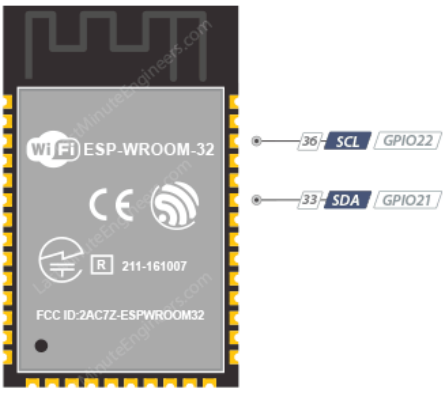
Inoltre, l'ESP32 WROOM 32 offre una serie di funzionalità avanzate, come una vasta gamma di porte di input/output (I/O), che consentono il collegamento di sensori e dispositivi esterni per l'acquisizione dei dati ambientali. Questo permette al dispositivo di essere altamente personalizzabile e adattabile alle esigenze specifiche del progetto. Inoltre, il supporto per diverse interfacce di comunicazione, come SPI, I2C e UART, facilita l'integrazione con altri componenti e dispositivi, ampliando le capacità del sistema.

Al microcontrollore sono collegati un Display LCD, il sensore BME688 e VEML6030, entrambi presentano il collegamento attraverso l'interfaccia I2C, poi sono presenti i tre pulsanti di input e la comunicazione da parte del GPS L86. Infine, ci sono i collegamenti

Interfaccia I2C

La connettività I2C, o Inter-Integrated Circuit, svolge un ruolo cruciale nel progetto, in quanto consente la comunicazione tra il microcontrollore ESP32 WROOM 32 e i sensori utilizzati per acquisire dati come temperatura, umidità, pressione atmosferica e luminosità. Questa interfaccia seriale bidirezionale offre diversi vantaggi, tra cui la semplicità di implementazione e l'efficienza nell'utilizzo dei pin del microcontrollore. Difatti, la comunicazione avviene attraverso quattro linee: SDA, SCL, Vcc, GND.

Uno dei principali vantaggi della connessione I2C è la sua capacità di gestire più dispositivi con un numero limitato di pin, grazie all'utilizzo di un bus dati condiviso, ossia il pin SDA. Ciò consente di collegare più sensori allo stesso bus I2C, riducendo il cablaggio. Mentre, per la comunicazione sincronizzata utilizziamo il pin di clock: SCL. Pertanto, per permettere il riconoscimento di ogni dispositivo, questi presentano dei indirizzi identificativi, ad esempio:



BME688 → 0x76

VEML6030 → 0x10

Interfaccia Utente

Display IPS LCD

Il display LCD IPS ST7789 con risoluzione 240x240 pixel rappresenta un componente fondamentale nel progetto, dato che offre un'interfaccia visiva intuitiva e chiara per la visualizzazione dei dati acquisiti e delle informazioni di stato del dispositivo. Dotato di tecnologia IPS (In-Plane Switching), il display garantisce angoli di visualizzazione ampi e colori vividi e realistici, assicurando una chiara leggibilità anche sotto diverse condizioni di illuminazione.

La risoluzione di 240x240 pixel offre una visualizzazione dettagliata e nitida dei dati ambientali, consentendo agli utenti di visualizzare informazioni precise e facilmente interpretabili. Questo è particolarmente importante in applicazioni di monitoraggio ambientale, dove la precisione e la chiarezza delle informazioni sono cruciali per prendere decisioni informate.

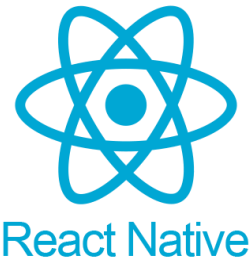
Inoltre, il display ST7789 offre una facile integrazione con il microcontrollore ESP32 WROOM 32, consentendo una rapida

implementazione e una configurazione semplice. Infatti, presenta la comunicazione tramite SPI.

La comunicazione SPI (Serial Peripheral Interface) è un protocollo seriale utilizzato per la trasmissione di dati tra dispositivi digitali. SPI coinvolge generalmente due tipi di dispositivi: il master e gli slave. Il dispositivo master coordina la comunicazione e può comunicare con uno o più dispositivi slave. Nella comunicazione SPI, i dati vengono trasmessi in modo sincrono su un bus seriale composto da quattro linee principali:

Il protocollo SPI supporta anche diverse modalità di trasmissione dei dati, che possono variare a seconda dei requisiti di timing e sincronizzazione del sistema. Le configurazioni più comuni includono modalità a 4 bit, a 8 bit e a 16 bit.

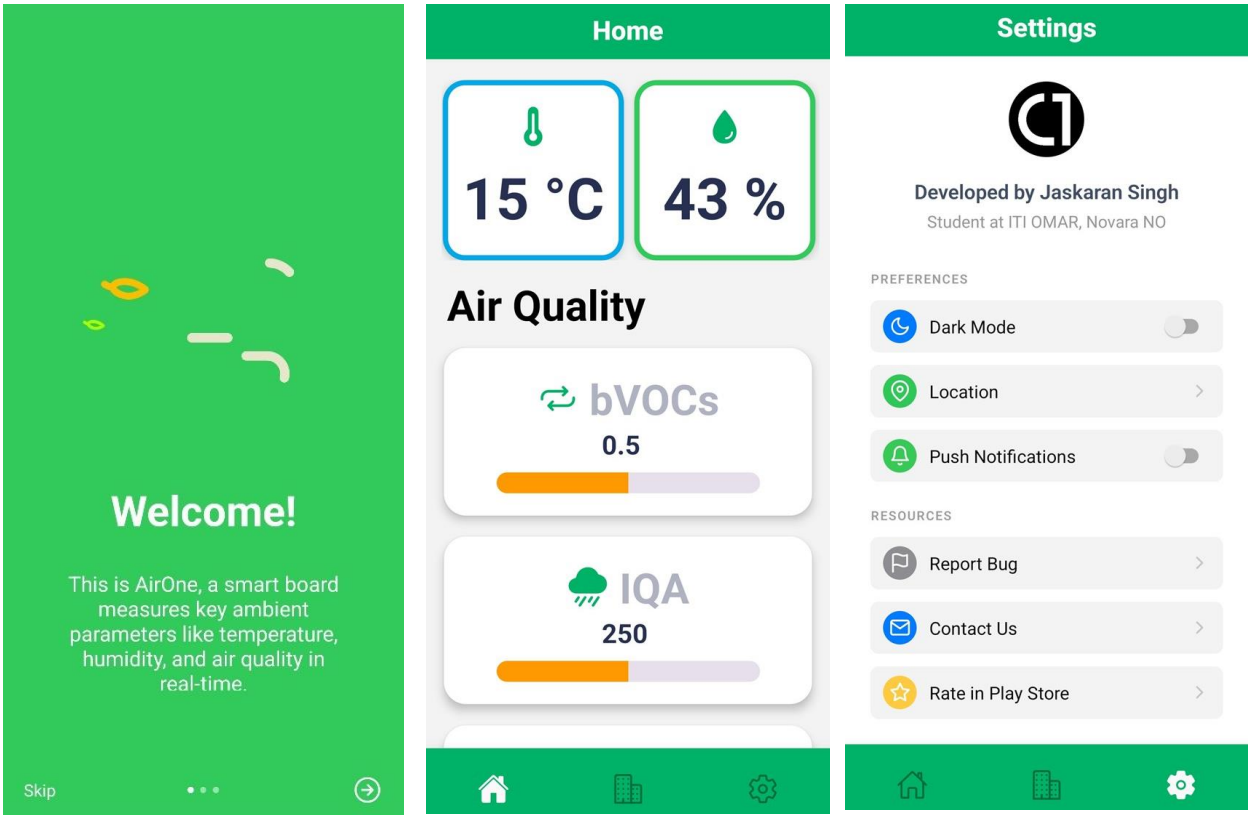
Applicazione Android



L'applicazione Andorid è stata realizzata con React Native, un linguaggio di programmazione basato su Javascript. Questo tipo è molto utilizzato nella realizzazione di applicazione e siti web, dato che presenta il "Cross-Platform", ossia la possibilità di utilizzare lo stesso codice sorgente per le piattaforme prima citate. Inoltre, un'altra caratteristica è la velocità nella creazione dell'App, e delle tante soluzioni efficienti, che consentono di ridurre i tempi di sviluppo.

Nell'applicazione sono state sviluppate tre pagine: home, outside e le impostazioni. Nella prima schermata, ossia quella di Home sono presenti i valori di temperatura, umidità e pressione sopra in alto, il bordo colorato varia in base ai valori di temperatura letti dal sensore BME688. Questo vale anche per le altre misure. Sotto notiamo la presenza della sezione "Qualità dell'Aria", ossia dove sono presenti tutti i parametri importanti per stabilire la qualità dell'aria della propria stanza, in cui è stato installato il dispositivo. La seguente pagina è quella "Outside", ossia il dispositivo è installato all'esterno, questa è uguale alla precedente. L'ultima è in via di sviluppo. Inoltre, non appena viene scaricata l'applicazione, saranno presenti tre diverse pagine di benvenuto, sulla quale sono presenti alcune informazioni utili per il corretto funzionamento dell'applicazione con la scheda Air One.

Per quanto riguarda la pagina dell'impostazione, rinominata come "Settings", sono presenti alcune funzioni utili come: la possibilità di selezionare il tema dell'applicazione, tema scuro oppure chiaro, la possibilità di attivare le notifiche istantanee, ciò mi permette di notificare l'utente nel momento in cui si presentano variazioni sulla scheda, e altre opzioni sulla creazione dell'app. Di seguito alcune immagini dell'applicazione:



Infine, questa applicazione è possibile scaricarla presso il Play Store di Google, oppure tramite il QR-code sul display LCD presente sulla scheda Air One.

Comunicazione Dati

Firebase Real-time Database con ESP32

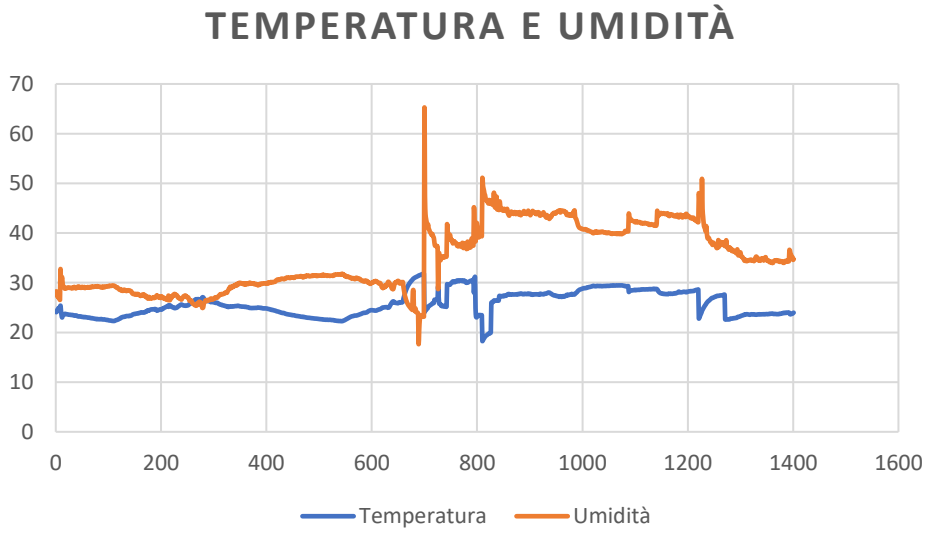
Per Air One, la comunicazione con il Realtime Database di Firebase permetterà di trasmettere dati ambientali in tempo reale. Questo si traduce in un'interfaccia diretta tra il dispositivo e il database, rendendo il processo rapido e affidabile. I dati verranno gestiti in formato JSON per una maggiore facilità d'uso e sicurezza. Grazie alla comunicazione istantanea, gli utenti riceveranno aggiornamenti immediati sulla qualità dell'aria e altri parametri ambientali.

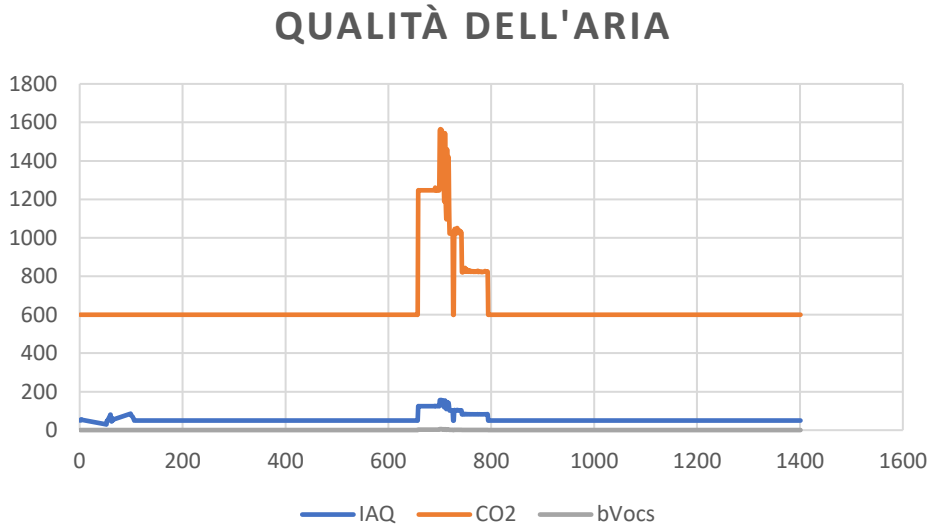
In breve, l'integrazione con il Realtime Database di Firebase migliorerà l'esperienza degli utenti, offrendo dati aggiornati e sicurezza dei dati. Questi vengono inviati direttamente sul database, che sono poi letti dall'applicazione Android. Di seguito l'interfaccia di Firebase sulla quale sono presenti tutti i dati ambientali, sono presenti sia i parametri interni (per la scheda installa nell'abitazione) e quelli esterni (per la scheda installata fuori dall'abitazione):

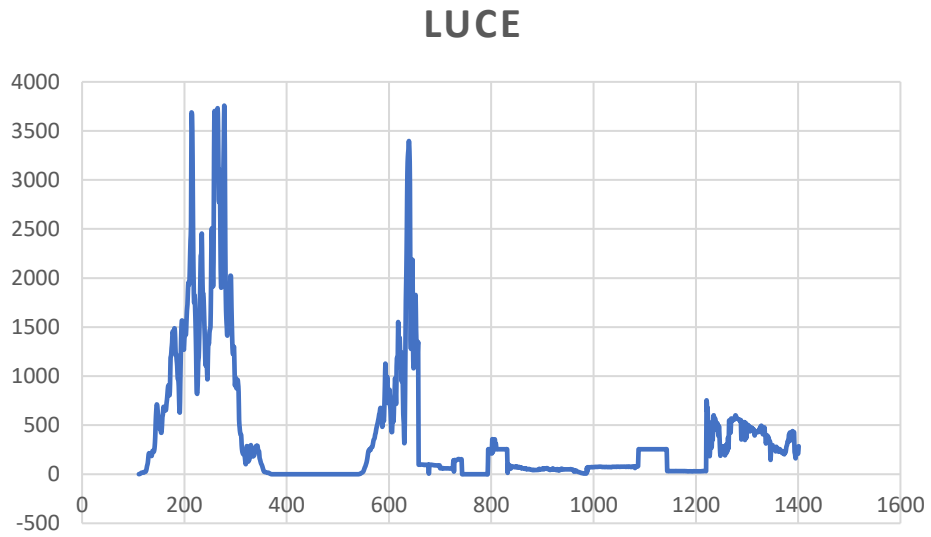
Thingspeak (Grafici Realtime)

Thingspeak è una piattaforma di analisi e visualizzazione di dati per l'IoT. Permette agli utenti di inviare dati dai loro dispositivi IoT, archiviare i dati nel cloud, e creare grafici in tempo reale per monitorare diverse metriche. La piattaforma è gestita da MathWorks, la stessa azienda dietro MATLAB, il che significa che offre potenti strumenti di analisi dei dati integrati.

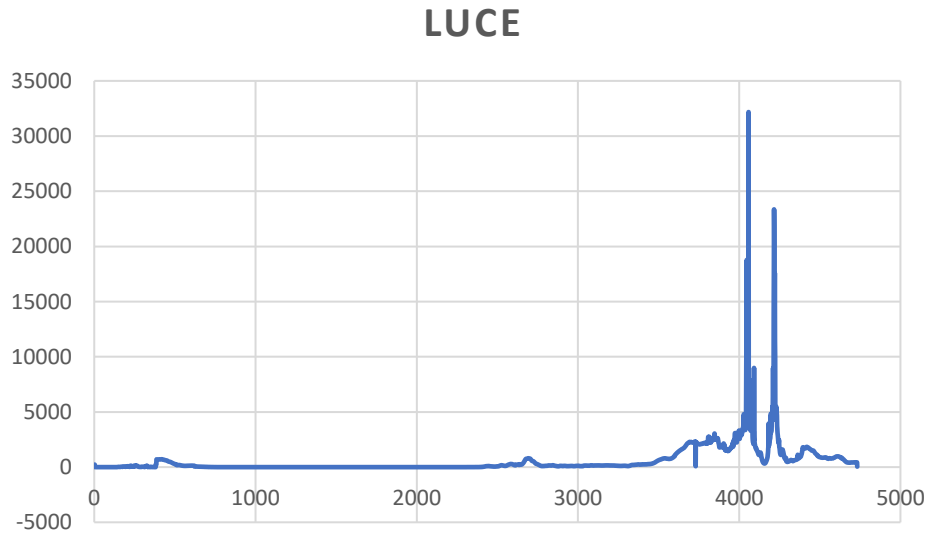
Per il progetto AirOne, che misura dati ambientali come la qualità dell'aria e molto altro, la piattaforma è utile per visualizzare i dati in tempo reale attraverso l'uso di grafici dinamici. Questo è stato possibile dopo aver creato il canale apposito per la scheda, interna ed esterna, per poi configurare tutti i grafici con i parametri misurati. Infine, l'invio di questi dati è stato possibile utilizzando la chiave API, ossia una stringa di caratteri univoca utilizzata per identificare l'accesso e l'autenticazione da parte della scheda.

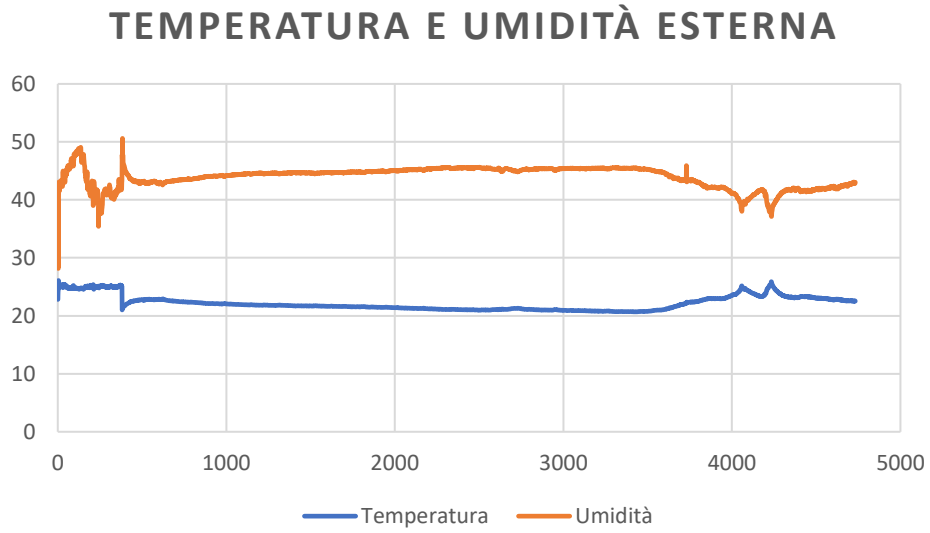


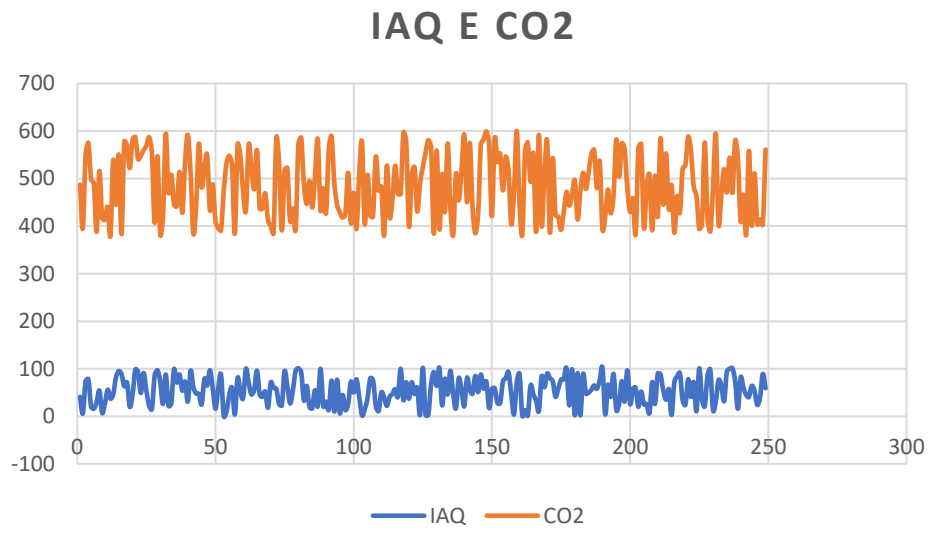


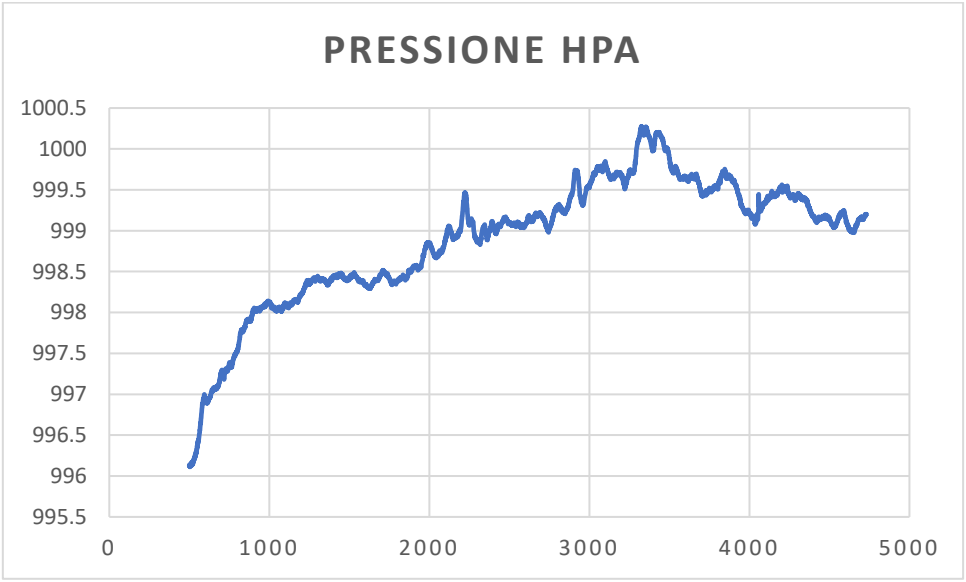


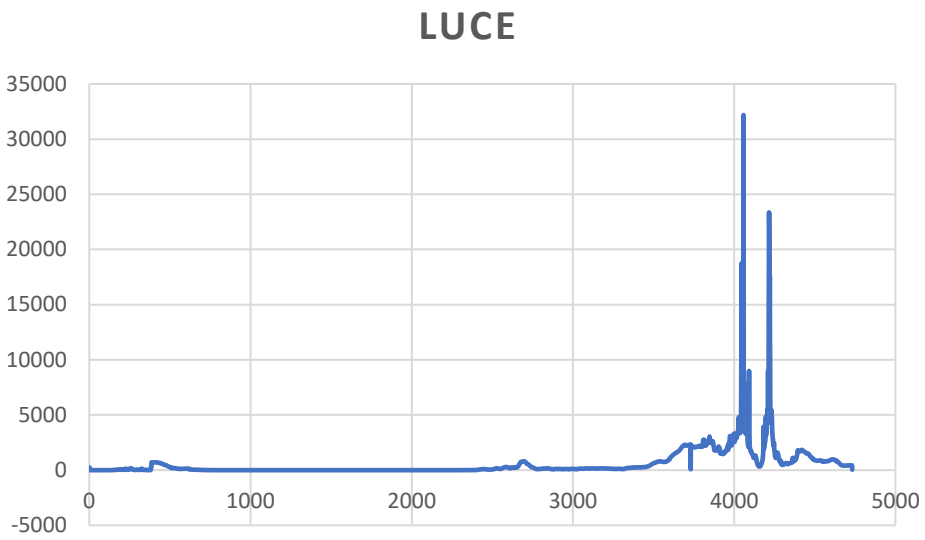
Questi sono i grafici che sono stati registrati dal dispositivo interno durante il suo funzionamento, per quanto riguarda il dispositivo esterno il funzionamento è stato praticamente lo stesso, solo con variazioni nella misura dei parametri ambientali:




TEMPERATURA E UMIDITÀ ESTERNA
60
50
40
30
20
10
0
0
1000
2000
3000
4000
5000
Temperatura
Umidità


IAQ E CO2
700
600
500
400
300
200
100
0
-100
0
50
100
150
200
250
300
IAQ
CO2


PRESSIONE HPA
1000.5
1000
999.5
999
998.5
998
997.5
997
996.5
996
995.5
0
1000
2000
3000
4000
5000



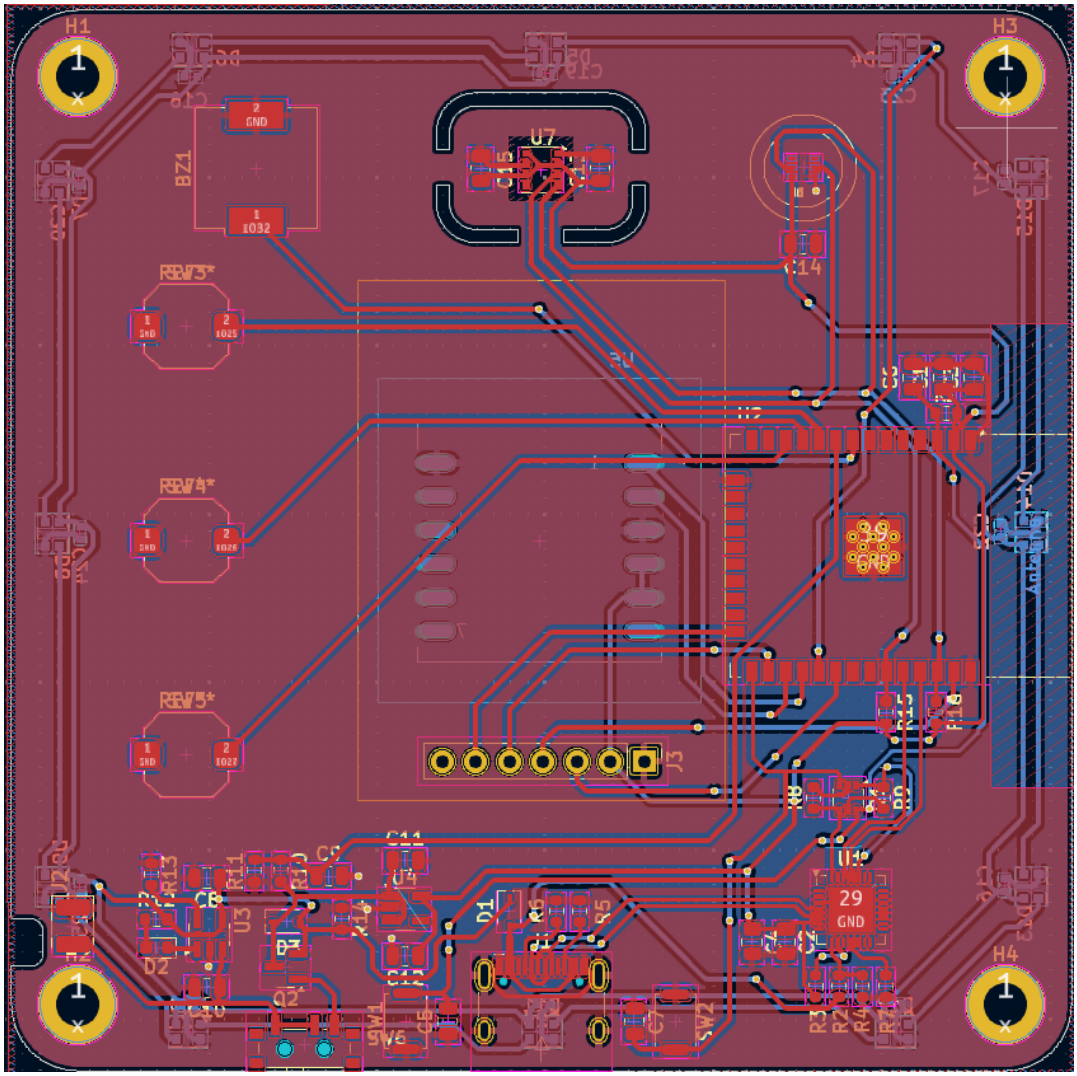
Da questi grafici notiamo che il sensore è molto preciso, e riesce a misurare quasi tutti i parametri in modo rapido e versatile. Inoltre, il sensore stesso richiede solo un tempo di accensione pari a circa 30 minuti, in modo tale da ottimizzare i valori in uscita sull'applicazione, così da non avere sbalzi di temperatura dovuti a errori.

Realizzazione Pratica

In seguito alla prototipazione su breadboard, sulla quale sono stati effettuati alcuni test riguardanti il collegamento al Wi-Fi e al Database su Firebase, è stato realizzato il circuito stampato. Quest'ultimo presenterà tutti i componenti necessari al corretto funzionamento della scheda finale; infatti, la maggior parte dei componenti utilizzati sono stati in SMD; dato che presentano alcuni vantaggi: l'ingombro; infatti, occupano meno spazio rispetto ai componenti THT, e la facilità d'utilizzo. Inoltre, la progettazione di un circuito stampato permette di risparmiare tempo ed avere delle connessioni migliori tra i vari componenti.

PCB su KiCad

Il PCB (Printed Circuit Board), ossia il circuito stampato, è stato realizzato sul programma Open-Source KiCad 8.0. Innanzitutto, è stato creato lo schema elettrico, presente nelle prime pagine. Dopo è stata progettato la board, come citato prima, sono stati utilizzati componenti SMD, come resistori e condensatori in formato 0603, pulsati, sensori BME688 e VEML6030.



Come si nota nell'immagine, il sensore BME688 è stato isolato realizzando una cornice intorno ad esso, in modo tale da non avere sbalzi di temperatura dovuti al riscaldamento della scheda stessa.

Montaggio Scheda

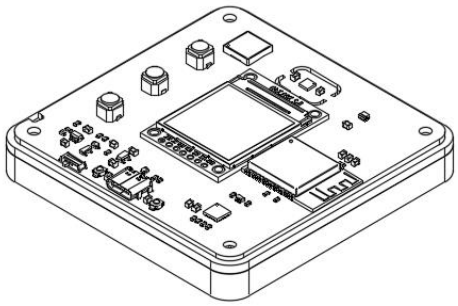
Dopo aver realizzato il circuito stampato sul programma, e aver fatto i controlli riguardanti le misure dei componenti, sono stati inviati i file GERBER all'azienda di produzione. Quest'ultima è stata JCLPCB, un'azienda situata in Hong Kong, alla quale è stato affidato il compito di realizzare il PCB. In seguito, lo stampato è stato montato saldando i componenti con della pasta saldante e una piastra riscaldante, utilizzata per sciogliere lo stagno alla temperatura richiesta. In aggiunta sono stati fatti dei ritocchi con il saldatore, in quanto lo stagno non aveva ben aderito su alcune piazzole.

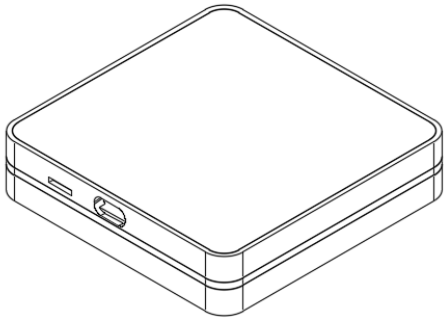
Struttura

Contenitore Stampato in 3D

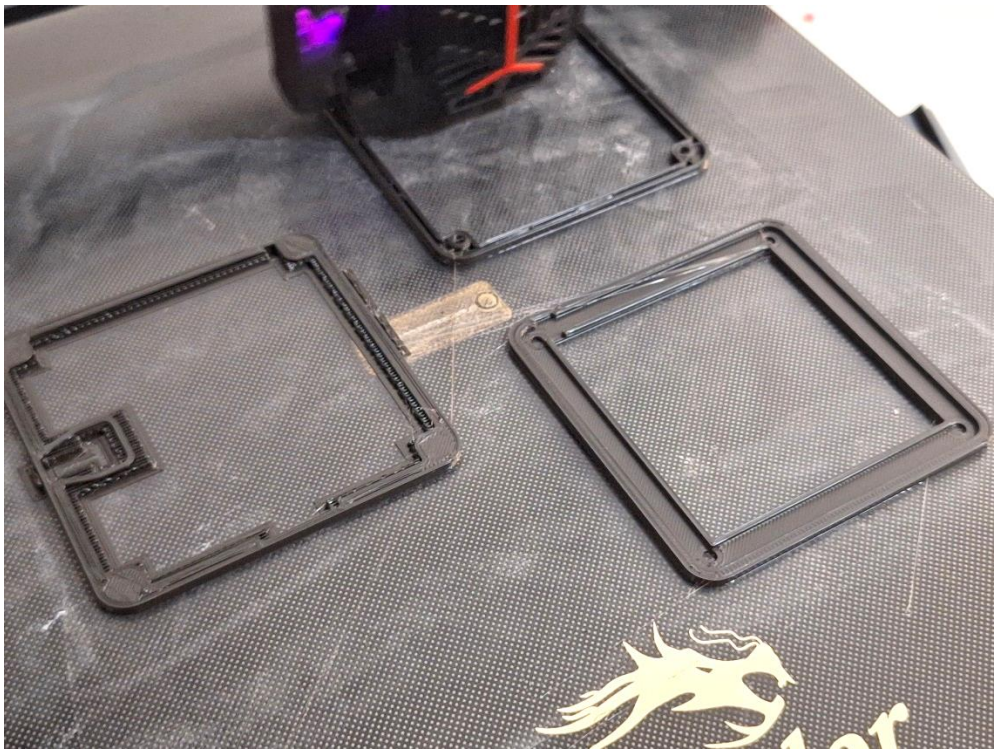
Il progetto prevede anche la creazione di un contenitore che permetta di isolare la parte elettronica dal mondo esterno, in modo tale da non instaurare problematiche. Inoltre, è stato isolato a parte il sensore multifunzione BME688, in quanto molto sensibile a tocchi involanti, dato che potrebbero rendere instabile le misurazioni

dettate dal produttore. Il contenitore, noto anche come Case, è stato progettato in Fusion 360 e poi stampato in 3D con filamento PLA. Di seguito sono presenti alcune immagini che raffigurano il case durante la stampa:





Di seguito qualche immagine del case in stampa, in questo è stato utilizzato del PLA di colore Nero, sul piatto sono presenti la parte superiore e inferiore con il diffusore dei LED SMD:

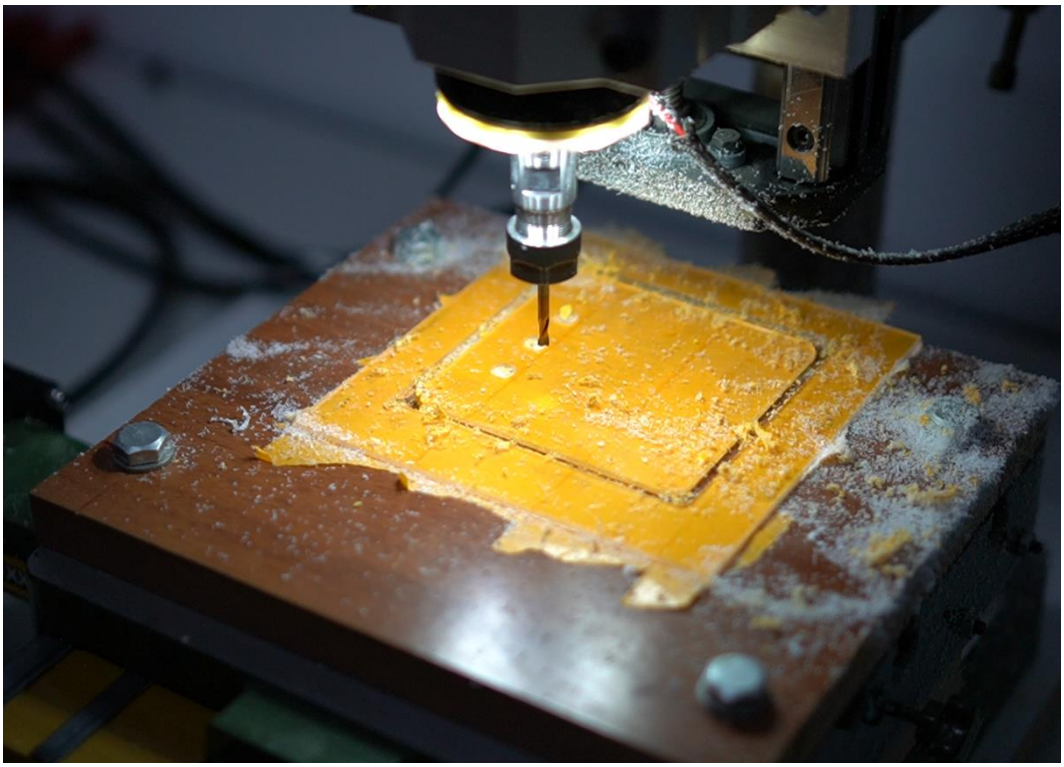


Dopo questo case stampato è stato montato con delle viti M3 e inserendo degli inserti all'interno dei fori appositi. In questo modo è stata realizzata una protezione intorno alla scheda e anche al sensore BME688.

Coperchio in Plexiglass

In seguito alla preparazione delle stampe 3D realizzate nel punto precedente, per andare a coprire la parte superiore e inferiore sono stati fresati due pezzi in plexiglass. Ciò è stato possibile utilizzando una CNC. In questo modo è possibile vedere il circuito stampato all'interno, e tutti i suoi componenti. In aggiunta è ottimale per una buona

protezione del circuito stesso, da eventuali problematiche. Di seguito un'immagine della fresatura del plexiglass:



In seguito, il plexiglass tagliato è stato montato sulla facciata anteriore e posteriore della scheda, in modo tale da finire il contenitore.

Risultati Ottenuti

I risultati sono stati ottimali sia per la scheda installata in casa, che per quella installata all'esterno. Difatti i dati da entrambe le schede venivano inviati ogni 20 secondi, in modo tale da permettere una buona ricezione sia dal database che dalla piattaforma ThingSpeak. Di seguito una panoramica dei grafici ottenuti durante il funzionamento delle schede, collegate entrambe al Wi-Fi e al Database di Firebase. I grafici sono stati riportati nella precedente sezione "Comunicazione dei Dati".

Inoltre, il sensore BME688 necessità di un tempo di configurazione iniziale, infatti bisogna lasciare collegato il sensore per circa 30 minuti, in modo tale da leggere dati veritieri.

Dai grafici prima illustrati, si nota che il dispositivo funziona correttamente, tutti i sensori presentano le loro uscite in modo ottimale e senza grandi errori. Tuttavia ci sono stati dei problemi legati alla calibrazione del sensore, in quanto richiedeva un tempo minimo di 30 giorni per una buona lettura dei dati, e dato il poco tempo disponibile ho deciso di accorciare questa calibrazione a circa 4/5 giorni. In questo modo sono riuscito ad ottenere valori accurati, inviandoli poi al database e all'applicazione Android.

Le schede, sia quella interna che quella esterna, sono state verificate prima con ambienti con una buona qualità dell'aria, con tutte le precauzioni, poi sono state monitorate all'esterno. In questo modo sono riuscito a valutare i valori misurati da

entrambe le schede per poter redigere il codice completo, con l'invio dei dati sul database di Firebase, e realizzare dei grafici dinamici in tempo reale su Thingspeak.

Codice Completo

Dispositivo Interno/Esterno

Di seguito è riportato il codice completo per il dispositivo installato all'interno. Mentre per quello esterno, i cambiamenti sono solo alcuni riguardanti gli indirizzi di collegamento al bus I2C, per quanto riguarda i sensori BME688 e il VEML6030, e la comunicazione dei dati su ThingSpeak per la creazione dei grafici. Pertanto sarà presente solo il codice del dispositivo installato all'esterno ossia quello "Outside", dato che i cambiamenti per quello interno sono minimi:

```
#include <Arduino.h>
#include "logoAirOne.h"

// Display
#include <TFT_eSPI.h>
TFT_eSPI tft = TFT_eSPI();  // Invoke library

// WIFI LIBs
#include <WiFi.h>
#include <FirebaseESP32.h>
#include <Wire.h>
#include "ThingSpeak.h"

// Provide the token generation process info.
#include <addons/TokenHelper.h>

// Provide the RTDB payload printing info and other helper functions.
#include <addons/RTDBHelper.h>

// Sensors Libs
#pragma region BME688
#include "bsec.h"

// Helper functions declarations
void checkIaqSensorStatus(void);
void errLeds(void);

// Create an object of the class Bsec
Bsec iaqSensor;

String output;
#pragma endregion

#pragma region VEML6030(LIGHT)

#include "SparkFun_VEML6030_Ambient_Light_Sensor.h"
```

```
#define AL_ADDR 0x10

SparkFun_Ambient_Light light(AL_ADDR);

float gain = .125;
int time_lenght = 100;
long luxVal = 0;

#pragma endregion

#pragma region FIREBASE& THINGSPEAK

#define WIFI_SSID "wifiname"
#define WIFI_PASSWORD "wifipassoword"

#define SECRET_CH_ID 0000000 // replace 0000000 with your channel number
#define SECRET_WRITE_APIKEY "XYZ"  // replace XYZ with your channel write
API Key

unsigned long myChannelNumber = SECRET_CH_ID;
const char* myWriteAPIKey = SECRET_WRITE_APIKEY;

/* 2. Define the API Key */
#define API_KEY "apikeyfirebase"

/* 3. Define the RTDB URL */
#define DATABASE_URL "databaseURL" //<databaseName>.firebaseio.com or
<databaseName>.<region>.firebasedatabase.app

/* 4. Define the user Email and password that alreadey registerd or added in
your project */
#define USER_EMAIL "email@example.com"
#define USER_PASSWORD "password"

// Define Firebase Data object
FirebaseData fbdo;

FirebaseAuth auth;
FirebaseConfig config;

WiFiClient client;

#pragma endregion

// BUZZER
#pragma region BUZZER
#include "pitches.h"
#define BUZZER_PIN 32  // ESP32 pin GPIO18 connected to piezo buzzer
```

```
int melody[] = {
  NOTE_G4, NOTE_C5, NOTE_E5
};

// Define the note durations: 4 = quarter note, 8 = eighth note, etc.
int noteDurations[] = {
  4, 4, 4
};

// Define the melody for the error sound
int errorMelody[] = {
  NOTE_A4, NOTE_E4, NOTE_C4
};

// Define the note durations: 4 = quarter note, 8 = eighth note, etc.
int errorNoteDurations[] = {
  8, 8, 4
};

int lowToneNote = NOTE_C3;
int lowToneDuration = 250;  // Adjust the duration as needed

#pragma endregion

#pragma region NEOPIXEL LEDs

#include <Adafruit_NeoPixel.h>

#define PIN 5
#define NUMPIXELS 12

Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
#define DELAYVAL 500  // Time (in milliseconds) to pause between pixels
#pragma endregion

unsigned long sendDataPrevMillis = 0;

bool flagConn = false;

void setup() {

  Serial.begin(115200);

  printLogoAirOne();

  pinMode(BUZZER_PIN, OUTPUT);

  pixels.begin();
  pixels.clear();  // Set all pixel colors to 'off'
```

```
  startVEML6030Light();
  startBME688Sensor();
  startWifiFirebaseThingspeak();

  turnOnLEDs();
  delay(1000);  // Wait for 1 second
  pixels.clear();
  pixels.show();  // Initialize all pixels to 'off'

  startSound();
}

void loop() {

  if (Firebase.ready() && (millis() - sendDataPrevMillis > 20000 ||
sendDataPrevMillis == 0)) {

    if (flagConn == false) {
      if (Firebase.ready()) {
        Serial.printf("Set int:  %s\n", Firebase.setString(fbdo,
F("/connOut"), "Y") ? "ok connOut" : fbdo.errorReason().c_str());
        flagConn = true;
      }
    }

    sendDataPrevMillis = millis();  // Delay Function

    if (iaqSensor.gasResistance > 0.8 * 30000) {
      Serial.println("Good IAQ");
    } else if (iaqSensor.gasResistance > 0.5 * 30000) {
      Serial.println("Moderate IAQ");
    } else {
      Serial.println("Bad IAQ");
    }

    readDatafromSensor();
    sendDataToFirebase();
    sendDataToThingspeak();
    Serial.println();
  }
}

void printLogoAirOne() {
  tft.begin();             // initialize a ST7789 chip
  tft.setSwapBytes(true);  // Swap the byte order for pushImage() - corrects
endianness
  tft.setRotation(2);

  tft.fillScreen(TFT_BLACK);
  tft.pushImage(0, 0, 240, 240, logo);
```

```
}

void startVEML6030Light() {
  Wire.begin();
  if (light.begin())
    Serial.println("Ready to sense some light!");
  else
    Serial.println("Could not communicate with the sensor!");

  light.setGain(gain);
  light.setIntegTime(time_lenght);

  Serial.println("Reading settings...");
  Serial.print("Gain: ");
  float gainVal = light.readGain();
  Serial.print(gainVal, 3);
  Serial.print(" Integration Time: ");
  int timeVal = light.readIntegTime();
  Serial.println(timeVal);
}

void startBME688Sensor() {
  delay(1000);
  iaqSensor.begin(BME68X_I2C_ADDR_HIGH, Wire);
  output = "\nBSEC library version " + String(iaqSensor.version.major) + "."
+ String(iaqSensor.version.minor) + "." +
String(iaqSensor.version.major_bugfix) + "." +
String(iaqSensor.version.minor_bugfix);
  Serial.println(output);
  checkIaqSensorStatus();

  bsec_virtual_sensor_t sensorList[13] = {
    BSEC_OUTPUT_IAQ,
    BSEC_OUTPUT_STATIC_IAQ,
    BSEC_OUTPUT_CO2_EQUIVALENT,
    BSEC_OUTPUT_BREATH_VOC_EQUIVALENT,
    BSEC_OUTPUT_RAW_TEMPERATURE,
    BSEC_OUTPUT_RAW_PRESSURE,
    BSEC_OUTPUT_RAW_HUMIDITY,
    BSEC_OUTPUT_RAW_GAS,
    BSEC_OUTPUT_STABILIZATION_STATUS,
    BSEC_OUTPUT_RUN_IN_STATUS,
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_TEMPERATURE,
    BSEC_OUTPUT_SENSOR_HEAT_COMPENSATED_HUMIDITY,
    BSEC_OUTPUT_GAS_PERCENTAGE
  };

  iaqSensor.updateSubscription(sensorList, 13, BSEC_SAMPLE_RATE_LP);
  checkIaqSensorStatus();
}
```

```
void startWifiFirebaseThingspeak() {
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();

  Serial.printf("Firebase Client v%s\n\n", FIREBASE_CLIENT_VERSION);

  /* Assign the api key (required) */
  config.api_key = API_KEY;

  /* Assign the user sign in credentials */
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;

  /* Assign the RTDB URL (required) */
  config.database_url = DATABASE_URL;

  config.token_status_callback = tokenStatusCallback;  // see
addons/TokenHelper.h
  Firebase.reconnectNetwork(true);

  fbdo.setBSSLBufferSize(4096, 1024);

  Firebase.begin(&config, &auth);

  Firebase.setDoubleDigits(5);

  WiFi.mode(WIFI_STA);
  ThingSpeak.begin(client);  // Initialize ThingSpeak
}

void sendDataToFirebase() {
  // Send data to firebase
  Serial.printf("Set int:  %s\n", Firebase.setInt(fbdo, F("/tempOut"),
iaqSensor.temperature - 5) ? "ok" : fbdo.errorReason().c_str());
  Serial.printf("Set int:  %s\n", Firebase.setInt(fbdo, F("/humidOut"),
iaqSensor.humidity) ? "ok" : fbdo.errorReason().c_str());
  Serial.printf("Set int:  %s\n", Firebase.setInt(fbdo, F("/pressureOut"),
iaqSensor.pressure / 100) ? "ok" : fbdo.errorReason().c_str());
  Serial.printf("Set int:  %s\n", Firebase.setInt(fbdo, F("/airOut"),
iaqSensor.iaq) ? "ok" : fbdo.errorReason().c_str());
```

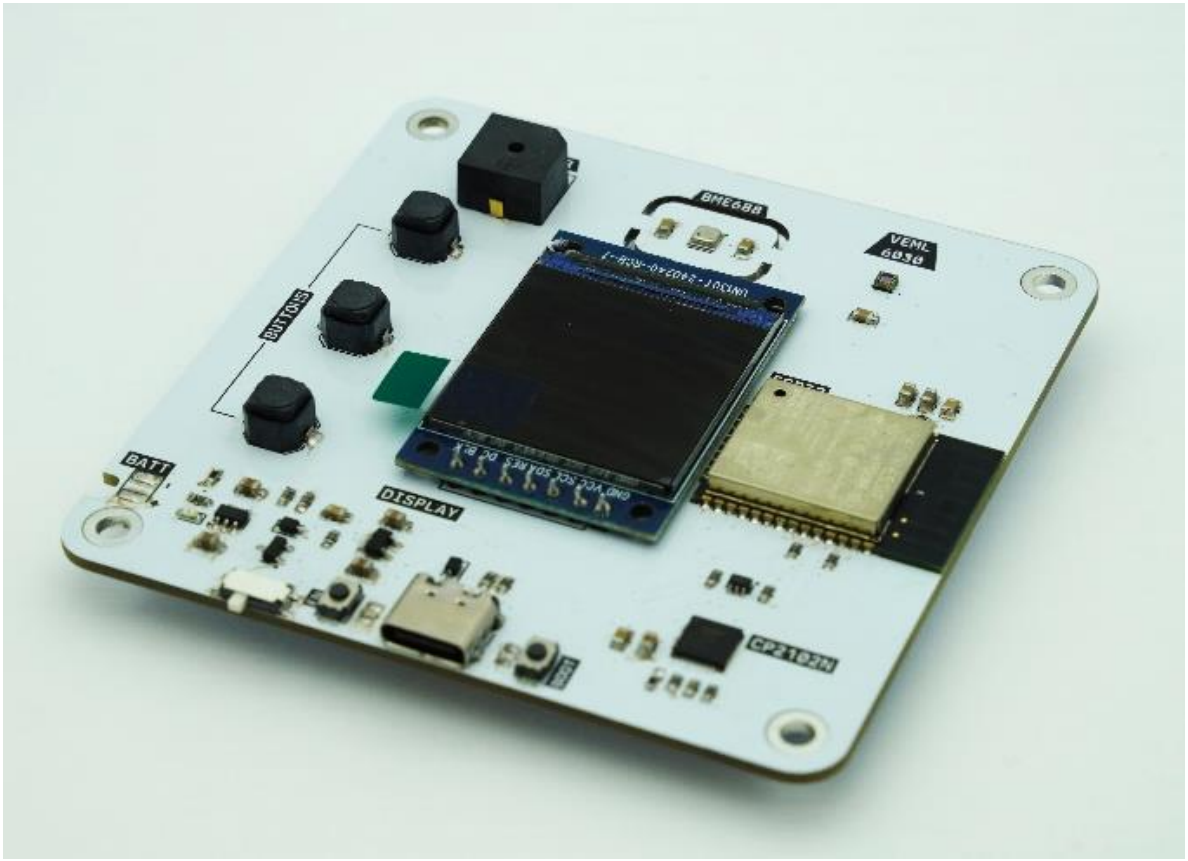
```
  Serial.printf("Set int:  %s\n", Firebase.setInt(fbdo, F("/luxOut"),
light.readLight()) ? "ok" : fbdo.errorReason().c_str());
  Serial.printf("Set int:  %s\n", Firebase.setInt(fbdo, F("/co2Out"),
iaqSensor.co2Equivalent) ? "ok" : fbdo.errorReason().c_str());
  Serial.printf("Set int:  %s\n", Firebase.setInt(fbdo, F("/bVocOut"),
iaqSensor.breathVocEquivalent) ? "ok" : fbdo.errorReason().c_str());
  Serial.printf("Set int:  %s\n", Firebase.setInt(fbdo, F("/gasOut"),
iaqSensor.gasResistance) ? "ok" : fbdo.errorReason().c_str());
}

void sendDataToThingspeak() {
  float lightValue = light.readLight();
  ThingSpeak.setField(1, iaqSensor.temperature - 5);
  ThingSpeak.setField(2, iaqSensor.humidity);
  ThingSpeak.setField(3, iaqSensor.iaq);
  ThingSpeak.setField(4, iaqSensor.co2Equivalent);
  ThingSpeak.setField(5, iaqSensor.breathVocEquivalent);
  ThingSpeak.setField(6, iaqSensor.pressure / 100);
  ThingSpeak.setField(7, lightValue);

  int x = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  if (x == 200) {
    Serial.println("Channel update successful.");
  } else {
    Serial.println("Problem updating channel. HTTP error code " +
String(x));
  }
}

void readDatafromSensor() {
  unsigned long time_trigger = millis();
  if (iaqSensor.run()) {  // If new data is available
    output = String(time_trigger);
    output += ", IAQ: " + String(iaqSensor.iaq);
    output += ", IAQ_AQ: " + String(iaqSensor.iaqAccuracy);
    output += ", STATIC_IAQ: " + String(iaqSensor.staticIaq);
    output += ", CO2: " + String(iaqSensor.co2Equivalent);
    output += ", BREATH: " + String(iaqSensor.breathVocEquivalent);
    output += ", PRESS:" + String(iaqSensor.pressure);
    output += ", GAS_R: " + String(iaqSensor.gasResistance);
    output += ", " + String(iaqSensor.stabStatus);
    output += ", " + String(iaqSensor.runInStatus);
    output += ", TEMP: " + String(iaqSensor.temperature - 5);
    output += ", HUMID: " + String(iaqSensor.humidity);
    output += ", GAS%: " + String(iaqSensor.gasPercentage);
    Serial.println(output);
  } else {
    checkIaqSensorStatus();
  }
}
```

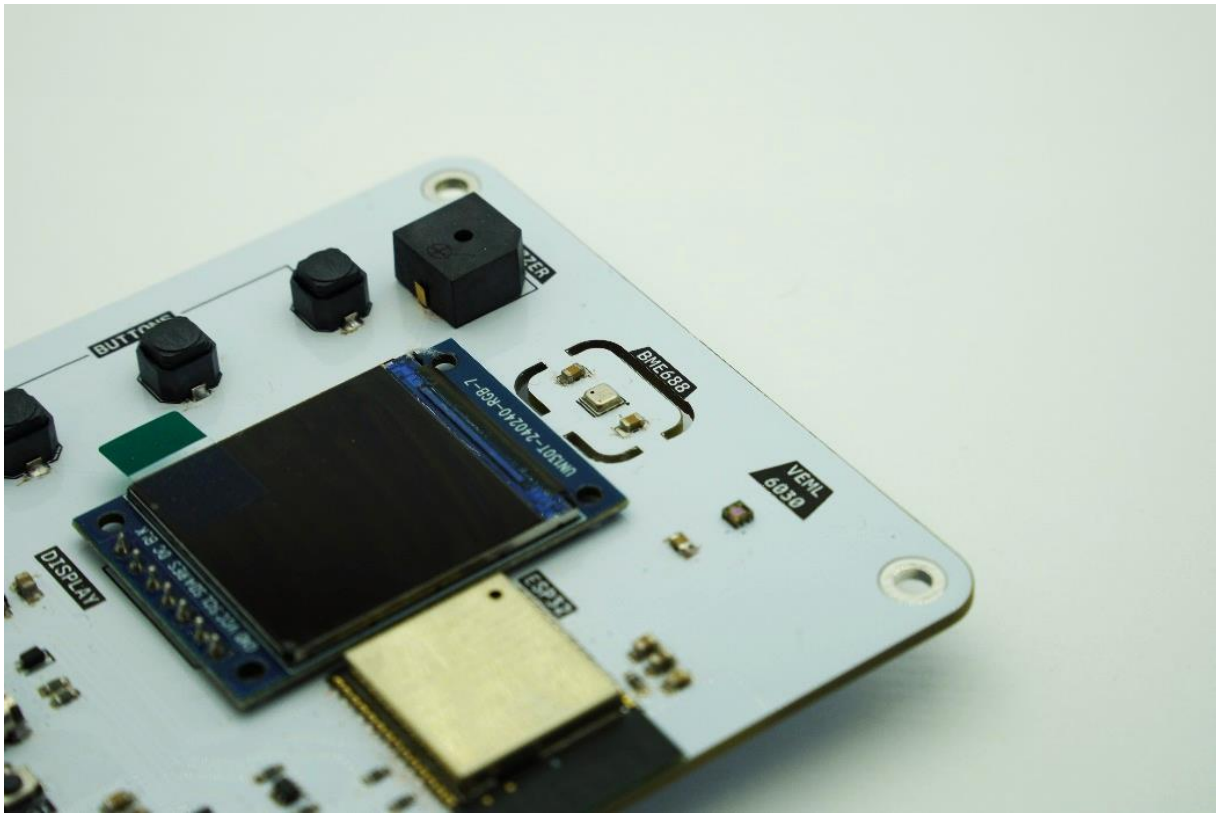
```
// Helper function definitions
void checkIaqSensorStatus(void) {
  if (iaqSensor.bsecStatus != BSEC_OK) {
    if (iaqSensor.bsecStatus < BSEC_OK) {
      output = "BSEC error code : " + String(iaqSensor.bsecStatus);
      Serial.println(output);
      for (;;)
        errLeds(); /* Halt in case of failure */
    } else {
      output = "BSEC warning code : " + String(iaqSensor.bsecStatus);
      Serial.println(output);
    }
  }

  if (iaqSensor.bme68xStatus != BME68X_OK) {
    if (iaqSensor.bme68xStatus < BME68X_OK) {
      output = "BME68X error code : " + String(iaqSensor.bme68xStatus);
      Serial.println(output);
      for (;;)
        errLeds(); /* Halt in case of failure */
    } else {
      output = "BME68X warning code : " + String(iaqSensor.bme68xStatus);
      Serial.println(output);
    }
  }
}

void startSound(void) {
  for (int i = 0; i < sizeof(melody) / sizeof(melody[0]); i++) {
    // Play the tone at a lower volume
    tone(BUZZER_PIN, melody[i], 1000 / noteDurations[i]);
    delay(1000 / noteDurations[i]);
    noTone(BUZZER_PIN);  // Stop the tone after the specified duration

    // Pause between notes
    delay(50);
  }
}

void errorSound(void) {
  for (int i = 0; i < sizeof(errorMelody) / sizeof(errorMelody[0]); i++) {
    // Play the tone
    tone(BUZZER_PIN, errorMelody[i], 1000 / errorNoteDurations[i]);
    delay(1000 / errorNoteDurations[i] + 50);
  }
}

void dataSound(void) {
  tone(BUZZER_PIN, lowToneNote);
```

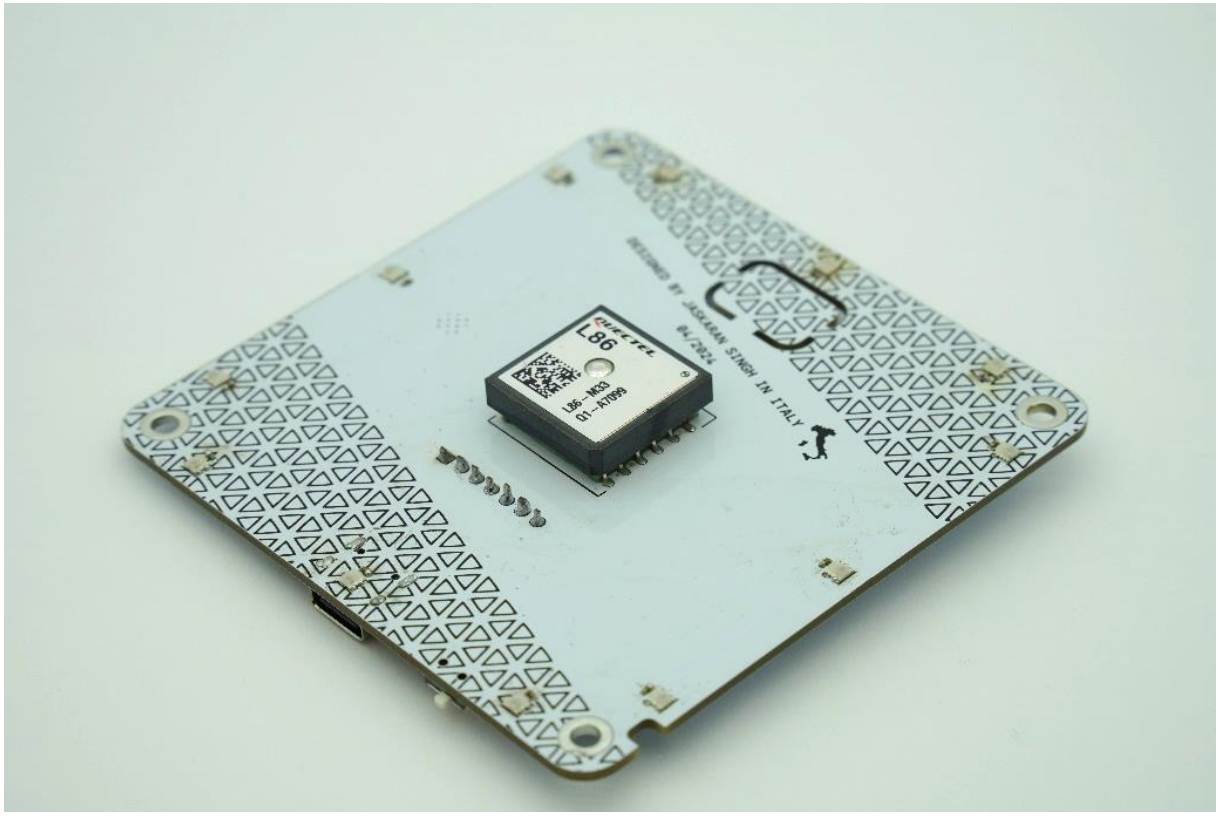
```
  delay(lowToneDuration);
  noTone(BUZZER_PIN);  // Stop the tone
}

void turnOnLEDs(void) {
  for (int i = 0; i < NUMPIXELS; i++) {
    pixels.setPixelColor(i, pixels.Color(9, 121, 105));  // Green color
    pixels.show();
  }
  delay(1400);  // Wait for 1 second

  // Turn off the NeoPixel strip
  pixels.clear();  // Set all pixel colors to 'off'
}
```

Immagini del Dispositivo

Dopo sono state scattate delle fotografie del dispositivo. Di seguito sono presenti le immagini del dispositivo con tutti i componenti SMD e THT montati sulla scheda, questa è uguale sia per la parte interna che esterna:




BUTTONS
BME688
VEML
6030
DISPLAY
ESP32


L86
DESIGNED BY JASKARAN SINGH IN ITALY

Video del Progetto

Ho deciso di registrare i principali passaggi della realizzazione del progetto e racchiuderli in un unico video. Quest'ultimo è stato poi caricato su una piattaforma online YouTube, di seguito è presente il link che è stato salvato nel seguente QR-code:

Conclusioni

Il progetto AirOne mi ha permesso di raggiungere gli obiettivi prefissati, ossia: la preparazione di una scheda facile da utilizzare e in grado di misurare i principali parametri ambientali, utili in ogni occasione. Inoltre, attraverso la preparazione del dispositivo sono insorti alcuni problemi legati alla realizzazione del progetto, ad esempio: era stata ideata un'alimentazione a batteri di litio, con un circuito saldato direttamente sulla scheda. Tuttavia, ciò non è stato possibile dato il non funzionamento del circuito stesso, infatti collegando la batteria venivano alimentate parti diverse del circuito rispetto a quella voluta.

In aggiunta, un miglioramento futuro potrebbe essere l'implementazione dell'applicazione Android anche su una pagina Web. Difatti il framework React Native, basato su Javascript, utilizzato per realizzare l'app, permette di creare anche siti Web basati sullo stesso codice. In questo modo, l'utente avrebbe accesso al monitoraggio del proprio ambiente anche attraverso l'apertura di una semplice pagina Web, e non necessariamente scaricare l'applicazione sul proprio dispositivo.

Infine, l'obiettivo è stato raggiunto con successo e in futuro verranno applicate alcune modifiche legate sia alla scheda che all'applicazione.