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

Febbraio 2024

Progetto e implementazione di un sistema embedded per il monitoraggio real-time della qualità dell'aria di un ambiente di lavoro

Design and implementation of an embedded system for real-time air quality monitoring of a working environment

Relatore:

Prof. Riccardo Berta

Correlatore:

Dott. Matteo Fresta

Candidato:

Dario Giardini

Indice

Sommario

Il presente elaborato è stato redatto con l'obiettivo di illustrare la realizzazione di un dispositivo per la rilevazione di agenti inquinanti presenti in un ambiente di lavoro. Il sistema acquisisce ed elabora i dati relativi alle sostanze. Successivamente le informazioni relative alle concentrazioni dei gas nell'aria vengono inviate ad una API (Application Programming Interface) framework chiamato Measurify.

Infine, per consentire una visualizzazione dei dati da parte dell'utente, è stata creata un'applicazione multipiattaforma che permette di visionare le condizioni attuali, le serie temporali delle ultime rilevazioni e la ricezione di notifiche push nel caso un gas superi la soglia stabilita.

Il progetto e la realizzazione di un sistema embedded per il monitoraggio in tempo reale della qualità dell'aria in un ambiente lavorativo sono finalizzati a soddisfare l'esigenza di garantire condizioni ottimali per la salute nei luoghi di lavoro.

1. Introduzione

In un'ottica di sempre maggiore attenzione al benessere e alla sicurezza delle persone, il monitoraggio delle sostanze presenti nell'aria si rivela cruciale, in special modo nei luoghi di lavoro e negli ambienti chiusi. La qualità dell'aria che respiriamo influisce direttamente sul nostro organismo e il monitoraggio degli agenti inquinanti è necessario per tutelare la salute dei lavoratori, rispettare le normative vigenti e garantire un ambiente di lavoro sano.

La presente tesi ha come oggetto la progettazione e la realizzazione di un sistema embedded per il monitoraggio della qualità dell'aria, utilizzando strumenti di facile sviluppo ed economicamente accessibili quali Arduino UNO Wi-Fi Rev2 come microcontrollore e un sensore di gas (Grove – Multichannel Gas Sensor).

Nel caso in questione si è deciso di effettuare una misurazione delle concentrazioni di Monossido di carbonio (CO), Biossido di Azoto (NO_2) e Metano (CH_4).

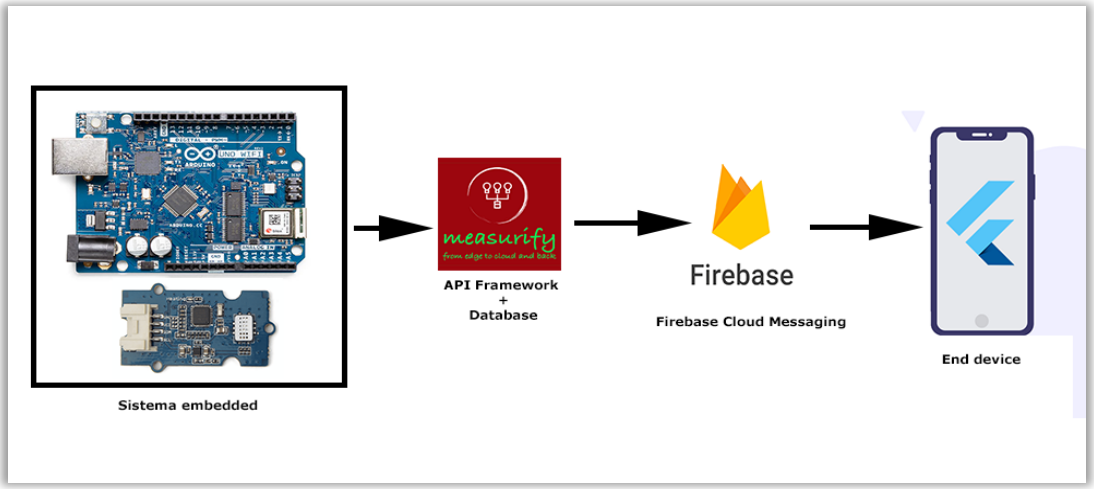

Figura 1.1 - Architettura del sistema

In figura è possibile prendere visione dell'architettura del sistema utilizzata in questa tesi.

Nel dettaglio, il sistema embedded una volta collocato nell'area di interesse, collegato all'alimentazione e connesso alla rete internet, inizierà ad effettuare rilevazioni delle sostanze inquinanti presenti nell'ambiente. Dopo aver acquisito un considerevole numero di misurazioni, esso invierà al database, tramite l'API framework Measurify, la media aritmetica dei valori relativi ai tre gas sopra citati. Nel caso l'API framework registri dei valori sopra la soglia impostata, verrà inviata una notifica push al cellulare dell'utente tramite la piattaforma Firebase Cloud Messaging, una soluzione di messaggistica che consente di inviare notifiche su più piattaforme.

In qualsiasi momento, l'utente finale potrà visualizzare tramite applicazione dedicata, sviluppata con Flutter, le concentrazioni relative ai gas, visualizzare una serie temporale delle ultime rilevazioni o verificare la concentrazione dei gas in caso di avvertimento di superamento soglia.

Le soglie sono state fissate prendendo come riferimento gli standard forniti dall'Agenzia Europea dell'Ambiente (EEA) [1] e dall'Organizzazione Mondiale della Sanità (WHO) [2]. Precisamente:

- diossido di azoto (NO_2): 106 ppm;
- monossido di carbonio (CO): 30 ppm;
- metano (CH_4): 1000 ppm.

La tesi è articolata in tre capitoli:

- nel primo verranno approfondite la progettazione e la realizzazione del sistema embedded, fornendo dettagli sulle configurazioni hardware e sulle scelte software;
- nel secondo sarà effettuata una verifica completa del corretto funzionamento del sistema;
- nel terzo e ultimo saranno esposte alcune considerazioni finali e commenti.

2. Metodi e strumenti utilizzati

2.1. Scheda Arduino e sensore

La scelta della scheda Arduino Uno WiFi Rev2 [3] per il progetto di tesi offre diversi vantaggi distintivi. Uno dei principali punti di forza è la presenza integrata del modulo Wi-Fi, che semplifica notevolmente l'implementazione della connettività wireless. Considerando il contesto del nostro sistema, che richiede una locazione fissa e non ha vincoli significativi in termini di dimensioni, questa tipologia di scheda rappresenta una soluzione ideale per soddisfare le esigenze specifiche del progetto.
Per quanto riguarda il sensore, è stato scelto il Grove – Multichannel Gas Sensor. È un dispositivo di rilevamento ambientale che incorpora il sensore MiCS-6814, il quale può individuare diversi gas nocivi, consentendo la misurazione simultanea di tre gas grazie ai suoi canali multipli [4].

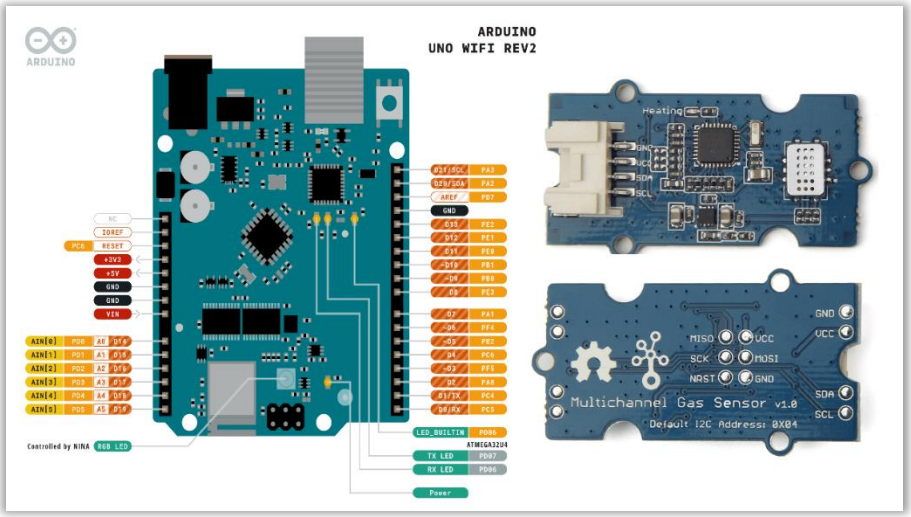

Figura 2.1 - Scheda Arduino e sensore di gas nel dettaglio

La comunicazione tra i due dispostivi avviene mediante il protocollo I^2C. Questo protocollo è caratterizzato da una configurazione master-slave, in cui un dispositivo master controlla la comunicazione con uno o più dispositivi slave [5]. La comunicazione avviene attraverso due linee di segnale: SDA (Serial Data Line) per la trasmissione dei dati e SCL (Serial Clock Line) per il timing di sincronizzazione. Questo consente di collegare più dispositivi sulla stessa linea, semplificando la connessione e riducendo il numero di pin necessari.

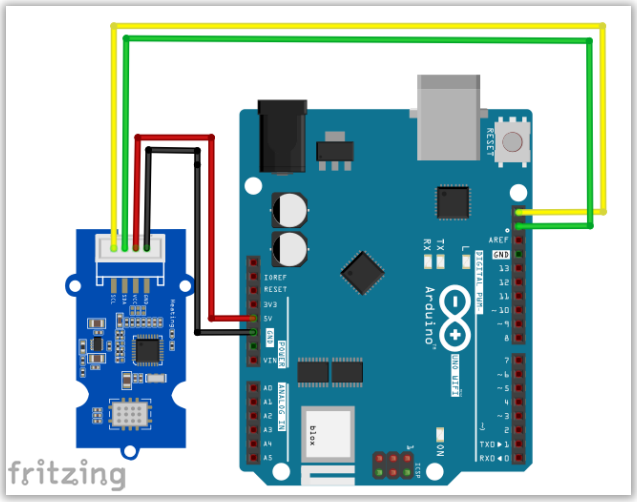

Figura 2.2 – Schema di collegamento

Dopo aver illustrato la configurazione hardware, passiamo a quella software: per sviluppare il seguente codice è stato necessario utilizzare l'ambiente di sviluppo Arduino IDE e il linguaggio di programmazione C++ insieme a diverse librerie specifiche per Arduino. Le librerie forniscono funzioni predefinite e strumenti di sviluppo che semplificano notevolmente l'implementazione di specifiche funzionalità. Di seguito sono elencate le librerie utilizzate nel codice:

- **Wire**: è utilizzata per la comunicazione I^2C, che consente di collegare sensori e dispositivi a microcontrollori. Nel codice fornito, la libreria è impiegata per comunicare con il sensore multicanale di gas.
- **MultichannelGasSensor:** offre un'interfaccia per il controllo del sensore multicanale di gas. Semplifica l'acquisizione dei dati dal sensore di gas e fornisce funzioni per misurarne la concentrazione in ppm (parti per milione/parts per million).
- **WiFiNINA:** è specifica per i moduli Wi-Fi basati sul chip NINA, come quelli presenti nelle schede Arduino compatibili con Wi-Fi. Semplifica la gestione delle connessioni e fornisce funzioni per la trasmissione di dati su reti wireless.

Continuando l'analisi del codice, possiamo esaminare le configurazioni iniziali definite al di fuori dei due cicli principali (*setup* e *loop*) di un programma Arduino. Queste includono le impostazioni Wi-Fi, i dettagli del server, la gestione del client Wi-Fi e alcune variabili cruciali per il funzionamento del programma.

Le variabili e gli array (Figura 2.3) vengono utilizzati nel processo di campionamento dei dati del sensore. Le nuove rilevazioni del sensore verranno memorizzate negli array *coData*, *no2Data* e *ch4Data* utilizzando gli indici appropriati. In seguito, le medie saranno calcolate e memorizzate nell'array *gasData*, per essere utilizzate nella valutazione delle soglie e nell'invio al server. Gli indici verranno aggiornati ciclicamente per consentire la memorizzazione continua dei dati del sensore.

```
float coData[SAMPLES_PER_HOUR];   //Carbon monoxid data
float no2Data[SAMPLES_PER_HOUR];  //Nitrogen dioxide data
float ch4Data[SAMPLES_PER_HOUR];  //Methane data
int coIndex = 0;                  //Start index
int no2Index = 0;                 //Start index
int ch4Index = 0;                 //Start index
float gasData[] = { 0, 0, 0 };    //Mean data {CO, NO2, CH4}
```

Figura 2.3 - Dichiarazione e inizializzazione variabili per la gestione dei valori

La parte di codice mostrata in figura 2.4 gestisce il controllo del tempo di campionamento dei dati del sensore e l'invio di tali dati al server Measurify. Inoltre, tiene traccia di eventuali mancati campionamenti o invii.
Viene dichiarata la variabile *previousMeasureMillis*, che memorizza il timestamp dell'ultima esecuzione della procedura di campionamento dei dati. La costante *measureInterval* è calcolata come il minimo tra 3.600.000 diviso per il numero massimo di campioni per ora (*SAMPLES_PER_HOUR*) e 60.000 millisecondi (un minuto), garantendo un intervallo di campionamento regolare.
Similmente, la sezione sottostante gestisce il tempo per l'invio dei dati al server. La variabile *previousPostMillis* tiene traccia del timestamp dell'ultima esecuzione della procedura di invio al server. La costante *postInterval* è calcolata come il massimo tra l'intervallo di campionamento e 600.000 millisecondi (10 minuti), garantendo che l'invio al server non avvenga più spesso di quanto si campioni il sensore.
Infine, le due variabili booleane fungono da indicatori di stato: *previousPostMissed* viene utilizzata per segnalare se il tentativo precedente di invio dati al server è fallito, mentre *previousDataMissed* indica se il campionamento dei dati del sensore è stato perso. Entrambe sono inizializzate a *false* per

evitare che il programma posti subito o campioni subito al suo avvio. Questi flag vengono successivamente utilizzati nel ciclo principale del programma per determinare se eseguire o meno le operazioni di campionamento e invio.

```
//Avoid overflow with currentMillis - previousMillis >= interval (unsigned)
unsigned long previousMeasureMillis = 0;
//Sampling interval in milliseconds
const unsigned long measureInterval = floor(min(3600000 / SAMPLES_PER_HOUR, 60000));

//Avoid overflow with currentMillis - previousMillis >= interval (unsigned)
unsigned long previousPostMillis = 0;
//Post interval in milliseconds
const unsigned long postInterval = max(measureInterval, 600000);

//Flag if connection to server failed
bool previousPostMissed = false;
//Flag  if sample failed
bool previousDataMissed = false;
```

Figura 2.4 - Dichiarazione e inizializzazione variabili e flag per la gestione del campionamento e invio dati

La funzione *setup()* (Figura 2.5) è responsabile della configurazione iniziale del programma Arduino.

```
void setup() {
  Serial.begin(9600);

  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to Network named: ");
    Serial.println(ssid);
    status = WiFi.begin(ssid, pass);
  }

  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  Serial.println("WiFi initialization completed");

  init(coData, SAMPLES_PER_HOUR);
  init(no2Data, SAMPLES_PER_HOUR);
  init(ch4Data, SAMPLES_PER_HOUR);

  Serial.println("Data initialization completed");

  Serial.println("Sensor initialization and pre-heating (5 minutes)");
  gas.begin(0x04);  //the default I2C address of the slave is 0x04
  gas.powerOn();
  delay(PRE_HEAT_TIME);
  Serial.println("Gas sensor initialization completed");
}
```

Figura 2.5 - Funzione setup()

In questa prima fase, la comunicazione seriale viene inizializzata per consentire l'invio di messaggi alla console. Successivamente, il programma tenterà di connettersi alla rete Wi-Fi specificata precedentemente finché non riuscirà a stabilire una connessione. Una volta stabilita, verranno inizializzati i dati dei sensori, impostando tutti gli elementi degli array *coData*, *no2Data*, e *ch4Data* a -1. Questa inizializzazione è utile per indicare che i dati non sono ancora stati acquisiti, dato che i valori reali risulteranno essere maggiori o uguali a zero.
Successivamente, il sensore di gas verrà attivato e sottoposto a un periodo di preriscaldamento di cinque minuti.

Una volta completata la fase di setup, il programma passa al ciclo principale per eseguire le operazioni di campionamento e invio dei dati.

Per quanto riguarda la funzione *loop()*, che verrà eseguita continuamente dopo aver terminato il setup, essa si articola in due parti principali:

- acquisizione dei dati, verifica della correttezza, calcolo della media;
- invio dei dati al server.

La prima parte (Figura 2.6) si verifica in due circostanze: se è trascorso un intervallo di tempo pari a *measureInterval* dal precedente campionamento dei dati oppure se in precedenza uno dei campionamenti è stato mancato.
Successivamente, il sistema procede all'acquisizione della concentrazione per ciascun tipo di gas. Per quanto concerne il monossido di carbonio (CO), ad esempio, il programma dopo aver ricevuto il valore dal sensore ne controlla la validità *(!isnan(measure) && measure >= 0*), lo memorizza nell'array *coData* all'indice corrente (*coIndex*), viene calcolata la media e salvata nell'array *gasData*. Questo processo si ripete in modo analogo per gli altri gas.
L'indice corrente di ciascun array, relativo al singolo gas, viene gestito in modo circolare. Quando si raggiunge la fine dell'array, l'indice torna a zero, garantendo una sovrascrittura dei dati più vecchi.
In seguito, viene aggiornato il timestamp contenuto nella variabile *previousMeasureMillis*, per pianificare così la successiva misurazione.

```
if (!isnan(measure = gas.measure_CO()) && measure >= 0) {
  coData[coIndex] = measure;
  gasData[0] = getMean(coData, SAMPLES_PER_HOUR);
  type = CARBON_MONOXIDE;

  Serial.print("CO value: ");
  Serial.print(coData[coIndex]);
  Serial.print(", CO average: ");
  Serial.print(gasData[0]);
  Serial.print(", Checking threshold: ");
  Serial.println(checkThreshold(gasData[0], type) ? "OVER" : "OK");

  coIndex = coIndex < (SAMPLES_PER_HOUR - 1) ? coIndex + 1 : 0;
}
```

Figura 2.6 – Esempio di campionamento, controllo validità e sforamento soglia

La funzione *checkThreshold* (Figura 2.8) accetta due parametri: *value* e *type*. Il parametro *value* rappresenta un valore numerico, mentre *type* è un'enumerazione che indica il tipo di gas (Figura 2.7). La funzione restituisce un valore booleano che indica se il valore del gas supera una specifica soglia associata al tipo fornito. All'interno della funzione, viene utilizzato uno switch statement per gestire i diversi tipi di gas (*CARBON_MONOXIDE*, *NITROGEN_DIOXIDE*, *METHANE*). Per ciascun tipo viene verificato se il valore fornito supera la soglia associata ad esso. Se il valore supera la soglia, la variabile *isOver* viene impostata a *true*, altrimenti rimane *false*.

```
//Gases name with thresholds
enum Gas {
  CARBON_MONOXIDE = 30,
  NITROGEN_DIOXIDE = 106,
  METHANE = 1000
};
```

Figura 2.7 – Gas e soglie

```
bool checkThreshold(float value, Gas type) {
  bool isOver = false;
  switch (type) {
    case CARBON_MONOXIDE:
      isOver = value > CARBON_MONOXIDE;
      break;
    case NITROGEN_DIOXIDE:
      isOver = value > NITROGEN_DIOXIDE;
      break;
    case METHANE:
      isOver = value > METHANE;
      break;
    default:
      break;
  }
  return isOver;
}
```

Figura 2.8 – Funzione controllo soglia

L'ultima parte (Figura 2.9) si occupa della verifica della connessione al server e dell'invio dei dati ad esso.

Nel dettaglio, il dispositivo cerca di effettuare una POST se è trascorso un intervallo di tempo pari a *postInterval* dal precedente invio dei dati oppure se uno degli invii è stato mancato in precedenza. Al verificarsi di almeno una delle due condizioni, si passa alla verifica dello stato della connessione. Se la connessione al server remoto dà un riscontro positivo, il programma procede all'invio effettivo dei dati attraverso la funzione *post(gasData)*. In caso contrario eseguirà un controllo della connessione Wi-Fi e qualora anche questo dia riscontro positivo, si tratterà quindi di un errore imputabile a fattori esterni.

In entrambi gli ultimi due scenari il flag *previousPostMissed* viene impostato a *true*.

Nel primo caso, il programma tenta di riconnettersi alla rete Wi-Fi fino a quando la connessione viene ripristinata o il timer di campionamento scade. Se il timer scade, viene impostato il flag *previousDataMissed* a *true* per evitare il blocco dell'acquisizione dei dati.

Il secondo caso rappresenta un errore esterno, poiché avendo escluso un errore dovuto alla connessione Wi-Fi, non rimane che un problema legato al server. In questo caso non è possibile risolvere il mancato invio in alcun modo.

```
if (previousPostMissed || checkTimer(currentMillis, previousPostMillis, postInterval)) {
  if (client.connect(server, port)) {
    previousPostMissed = false;
    Serial.println("Making POST...");
    post(gasData);
    previousPostMillis = currentMillis;
  } else if (WiFi.status() != WL_CONNECTED) {  //Check WiFi connection
    previousPostMissed = true;
    Serial.println("Cannot make POST. Lost WiFi connection, reconnecting...");
    //Attempt to reconnect until sample timer is over and flag missed post
    while (WiFi.status() != WL_CONNECTED) {
      unsigned long extraMillis = millis();
      WiFi.begin(ssid, pass);
      //This avoids sampling blockage
      if (checkTimer(extraMillis, previousMeasureMillis, measureInterval)) {
        previousDataMissed = true;
        break;
      }
    }
  } else {
    //Cannot connect to server and flag missed post
    previousPostMissed = true;
    Serial.println("Cannot make POST. Cannot connect to the server");
  }
```

Figura 2.9 - Blocco di codice per il controllo della connessione

2.2. Measurify

Per poter rendere i dati accessibili esternamente e quindi essere ricevuti dall'utente finale è stato necessario appoggiarsi ad un server con un database. La piattaforma usata utilizza Measurify per le chiamate di collegamento al database.
Measurify è un API framework di tipo RESTful utilizzato per la raccolta di dati provenienti da sensori in ambito IoT (Internet of Things), sviluppato dall'Elios Lab dell'Università di Genova [6].
Esso si basa sui seguenti concetti:

- *Thing*: il soggetto della misurazione
- *Device*: il dispositivo che effettua la misurazione
- *Feature*: la descrizione della misurazione
- *Measurement*: l'effettiva misurazione

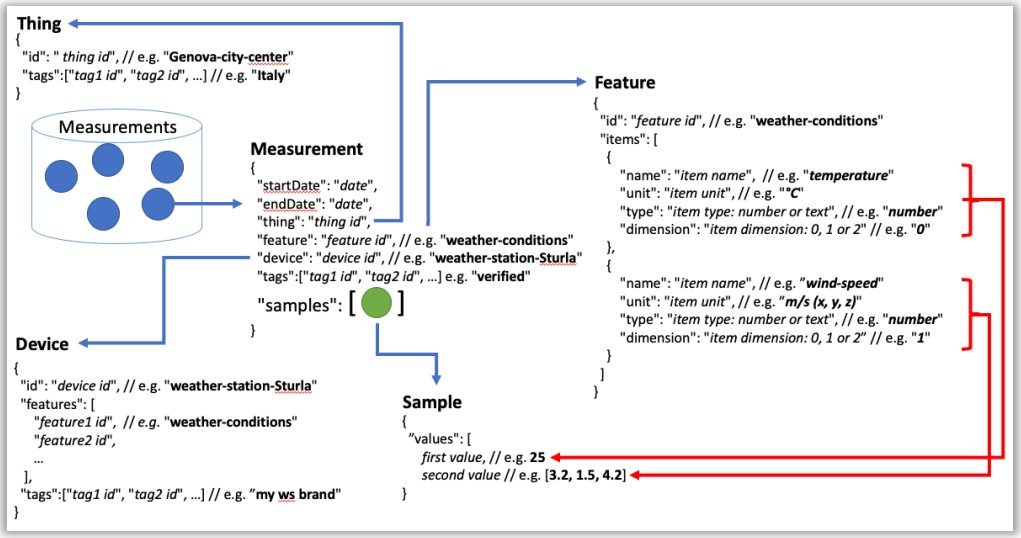

Figura 2.10 - Schema concettuale di Measurify

Per quanto riguardo il progetto, *thing* è l'ambiente dove sarà localizzato il sensore, *device* rappresenta il sistema embedded (Arduino + Gas sensor), *feature* fornisce una descrizione accurata dei dati presenti nella misurazione e infine *measurement* contiene i dati dei rilevamenti dei gas.

Il server, essendo una API di tipo RESTful, supporta i principali metodi HTTP (GET, POST, PUT e DELETE) necessari per gestire le misurazioni nel cloud.

Per comunicare con il server, è necessario innanzitutto effettuare l'accesso tramite una richiesta POST per ottenere un token, il quale ha una validità di 30 minuti. Nel mio caso, si è deciso di generare un token specificamente abilitato solo per l'invio di dati, evitando così il problema della scadenza.

L'invio delle misurazioni avviene attraverso una richiesta POST, il cui corpo (body) contiene i dati dei gas, strutturati in formato JSON (JavaScript Object Notation), in coerenza con quanto definito nella *feature*. Il formato JSON è un formato standard per lo scambio di dati tra sistemi ed è ampiamente utilizzato nell'ambito dello sviluppo software e delle comunicazioni tra client e server [7].

2.3. Postman

Postman è un ambiente di sviluppo API completo. Consente agli sviluppatori di effettuare richieste API, ispezionare i dati di richiesta e risposta e eseguire il debug degli endpoint API [8].

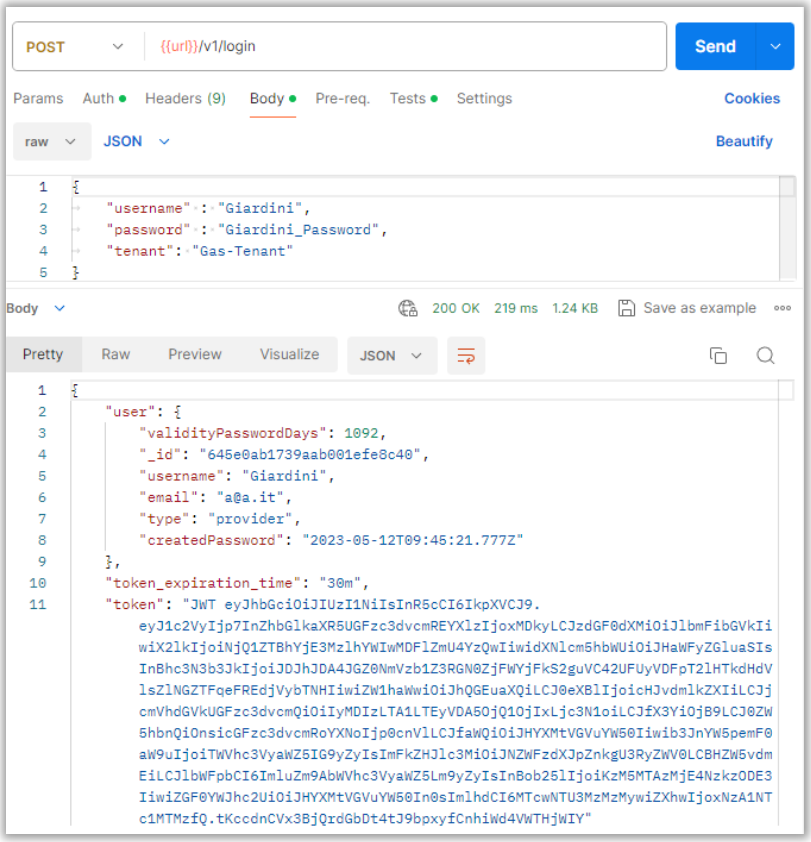

Figura 2.11 - Risposta del server ad un login

Postman ha svolto un ruolo fondamentale nel processo di sviluppo e nell'acquisizione della conoscenza sulla struttura delle richieste e delle risposte da parte di Measurify, semplificando l'interazione e la comprensione delle API.

2.4. Firebase Cloud Messaging

Per implementare la ricezione delle notifiche in-app in presenza di nuovi dati dal sensore, si è scelto di utilizzare Firebase Cloud Messaging (FCM), un servizio di messaggistica push sviluppato da Firebase, di proprietà di Google [9].
Per poter abilitare la ricezione delle notifiche è stato dapprima necessario aggiungere un file JSON di configurazione 'google-services.json' al progetto Flutter. Questo file contiene le informazioni di configurazione di Firebase, inclusi i dettagli del progetto e le chiavi necessarie per la comunicazione con FCM.
Successivamente è necessario effettuare una *subscription* tramite POST, al server Measurify, il cui corpo conterrà la *thing* in questione e un token specifico di ogni singolo dispositivo.
Va sottolineato che l'algoritmo di gestione della comunicazione tra il server Measurify e Firebase era già stato implementato, pertanto il mio compito ha riguardato esclusivamente la gestione all'interno dell'applicazione.

2.5. Flutter

Per quanto riguarda lo sviluppo di un'interfaccia finale di facile uso da parte dell'utente, si è optato per Flutter. Flutter è un toolkit open source multi-piattaforma per lo sviluppo software, creato da Google, che utilizza il linguaggio di programmazione open-source Dart, anch'esso sviluppato da Google. Consente ai programmatori di realizzare applicazioni nativamente compilate per dispositivi mobili, web e desktop, partendo da un unico codice sorgente [10].
Passando ora all'analisi della struttura dell'applicazione, sono stati creati tre file:

- *main.dart*: contiene l'interfaccia grafica e gestisce la ricezione dei dati e l'aggiornamento dei valori nell'interfaccia (tabella e grafico);
- *notifications.dar*t: si occupa dell'invio delle notifiche di avviso superamento soglia
- *constants.dart*: contiene le credenziali di accesso al server.

Concentrandoci sull'unità principale, il *main*, esso svolge quattro compiti: generazione del token, inizializzazione dell'interfaccia alla prima apertura dell'applicazione, gestione dell'arrivo dei dati quando l'app è in primo piano e gestione mentre risulta in background o terminata.

Queste quattro situazioni vengono gestiste tramite delle funzioni fornite direttamente dalla libreria di Firebase e sono rispettivamente:

- *getToken()*: genera un token di registrazione del dispositivo, che è un identificatore unico assegnato a ogni istanza dell'applicazione su un dispositivo specifico;
- *getInitialMessage()*: l'applicazione viene aperta da uno stato terminato;
- *onMessage*: l'applicazione si trova in prima piano;
- *onBackgroundMessage*/*onMessageOpenedApp*: l'applicazione si trova in background o terminata.

All'interno della funzione *getInitialMessage(),* innanzitutto, il codice esegue una richiesta POST per ottenere il token necessario per comunicare con il server Measurify. Se va a buon fine, viene eseguita una richiesta GET per ottenere una serie di misurazioni precedenti, garantendo così una continuità con i nuovi arrivi di dati.
Alla richiesta GET vengono inseriti due filtri: i dati sono stati registrati dalla mezzanotte del giorno corrente e in un numero non superiore a 60. Ciò al fine di non creare un'elevata dispersione nel grafico e di non fornire una lettura di dati troppo arretrati.

Per quanto riguarda la funzione *onMessage*, essa si occupa di codificare il contenuto del messaggio, effettuare un controllo delle soglie dei valori e successivo aggiornamento dell'interfaccia grafica.

La funzione *onBackgroundMessage* svolge un ruolo cruciale nella gestione delle notifiche quando l'applicazione non è in primo piano. Essa è responsabile di inviare notifiche quando uno dei tre gas monitorati supera una soglia critica. D'altra parte, *onMessageOpenedApp*, similmente alla funzione *onMessage*, si occupa della decodifica dei messaggi e dell'aggiornamento dell'interfaccia grafica quando l'applicazione viene riportata in primo piano dopo la ricezione di una notifica.

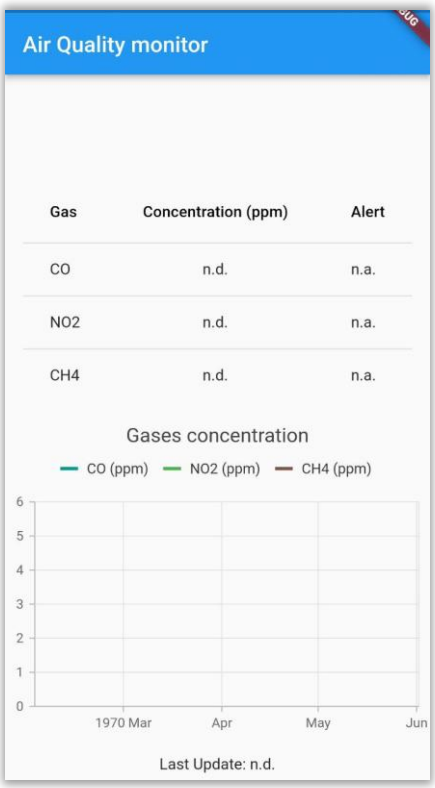

Figura 2.12 - Interfaccia grafica dell'applicazione

Come osservabile in figura 2.12, l'interfaccia è composta da due parti: una tabella e un grafico.
La tabella mostrerà la concentrazione in ppm e un messaggio di allerta ("HIGH" o "NORMAL") per ciascun gas.
Nel grafico verranno visualizzate serie temporali per ciascun inquinante. Attraverso semplici interazioni, è possibile modificare la visualizzazione mostrando una o più serie alla volta. Questa flessibilità è fondamentale per garantire un'ottimale visualizzazione dei dati, considerando i diversi range di valori associati ad ogni singolo gas.

Infine, nella parte inferiore saranno visibili data e ora relativi all'ultimo dato ricevuto.

3. Sperimentazione e Risultati

3.1. Acquisizione dati

Al fine di verificare il corretto funzionamento del sistema, sono stati effettuati i collegamenti necessari tra scheda e sensore, è stata connessa la board al pc tramite cavo usb ed è stato caricato lo sketch tramite l'IDE di Arduino, in attesa di visualizzare i messaggi di debug su monitor seriale.

```
Attempting to connect to Network named: iPhone
SSID: iPhone
WiFi initialization completed
Data initialization completed
Sensor initialization and pre-heating (5 minutes)
version = 2
```

Figura 3.1 - Connessione Wi-Fi e inizializzazione dei dati

Per prima cosa il sistema tenterà di instaurare una connessione Wi-Fi con l'endpoint specificato e in caso affermativo provvederà a notificare l'avvenuta inizializzazione delle variabili necessarie all'algoritmo.
Scaduti i cinque minuti di preriscaldamento, il sistema inizierà ad acquisire ciclicamente le concentrazioni misurate dal sensore.

```
Sampling gas data...
CO value: 4.46, CO average: 4.46, Checking threshold: OK
NO value: 0.22, NO average: 0.22, Checking threshold: OK
CH4 value: 669.49, CH4 average: 669.49, Checking threshold: OK
---------------------------------------------------------------
Sampling gas data...
CO value: 4.58, CO average: 4.52, Checking threshold: OK
NO value: 0.22, NO average: 0.22, Checking threshold: OK
CH4 value: 738.52, CH4 average: 704.00, Checking threshold: OK
---------------------------------------------------------------
Sampling gas data...
CO value: 4.55, CO average: 4.53, Checking threshold: OK
NO value: 0.23, NO average: 0.22, Checking threshold: OK
CH4 value: 724.21, CH4 average: 710.74, Checking threshold: OK
---------------------------------------------------------------
```

Figura 3.2 - Acquisizione dati e visualizzazione dei valori

Come è possibile osservare (Figura 3.2) vengono mostrate le concentrazioni attuali, le rispettive medie e un messaggio di controllo relativo alla soglia.

Di seguito (Figura 3.3) si è verificato l'effettivo tentativo di invio al server in caso di perdita della connessione Wi-Fi con successiva riconnessione e invio.

```
Cannot make POST. Lost WiFi connection, reconnecting...
Sampling gas data...
CO value: 4.09, CO average: 4.32, Checking threshold: OK
NO value: 0.26, NO average: 0.24, Checking threshold: OK
CH4 value: 486.29, CH4 average: 603.52, Checking threshold: OK
-------------------------------------------------------------
Cannot make POST. Lost WiFi connection, reconnecting...
Sampling gas data...
CO value: 4.09, CO average: 4.29, Checking threshold: OK
NO value: 0.27, NO average: 0.24, Checking threshold: OK
CH4 value: 486.29, CH4 average: 586.77, Checking threshold: OK
-------------------------------------------------------------
Cannot make POST. Lost WiFi connection, reconnecting...
Making POST...
POST request sent
```

Figura 3.3 - Acquisizione e invio dati in caso di perdita connessione Wi-Fi

È possibile notare come effettivamente il tentativo di riconnessione non sia bloccante: il sistema continuerà ad effettuare le misurazioni e nel mentre cercherà di ristabilire la connessione Wi-Fi.

Dopo aver visionato il corretto funzionamento tramite monitor seriale, si è proceduto a rendere il sistema totalmente indipendente alimentandolo tramite una sorgente 5V – 1A.

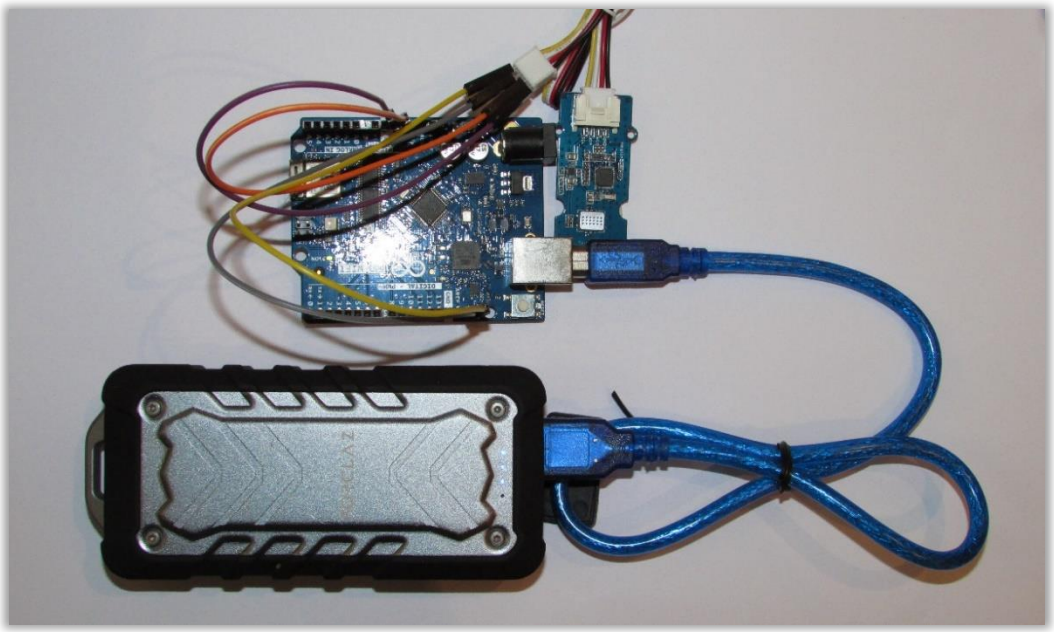

Figura 3.4 - Schema di collegamento per funzionamento stand-alone

3.2. Verifica invio dati al server

Prima di passare ad una verifica del funzionamento dell'applicazione è stato necessario esaminare la corretta ricezione delle misurazioni da parte del server.
Tramite l'utilizzo di Postman, effettuando una richiesta di tipo GET, si è ottenuta la seguente risposta:

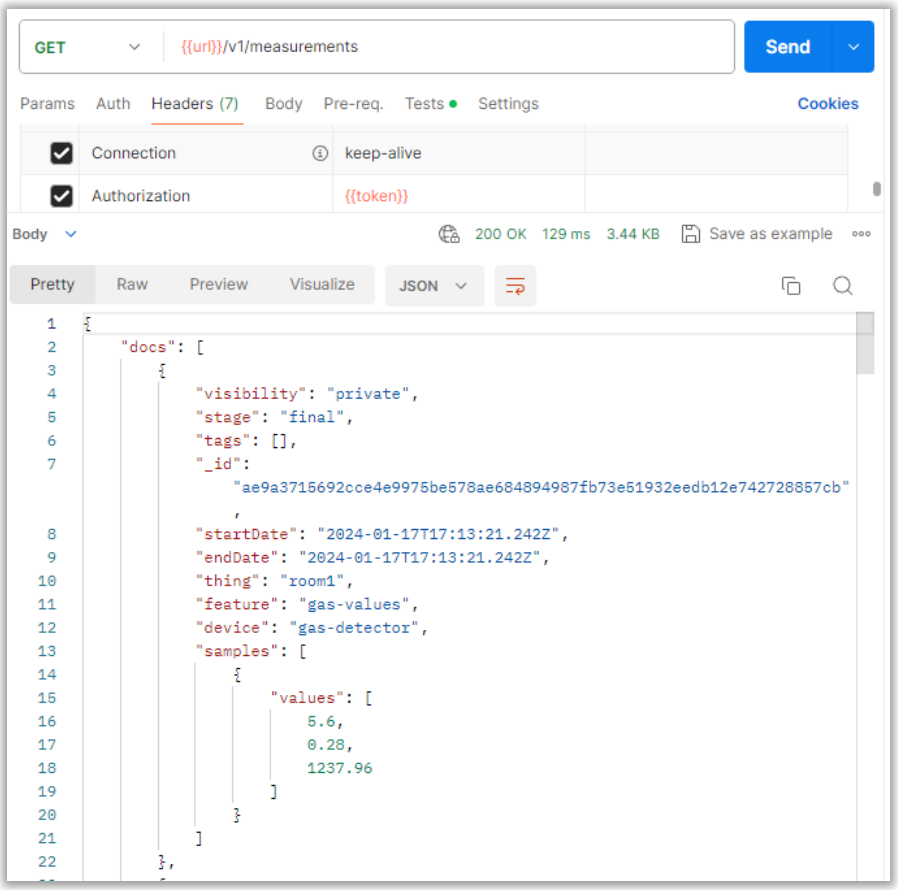

Figura 3.5 - Risposta con le misurazioni associate al tenant specificato (Gas-Tenant)

La risposta, strutturata secondo il formato JSON, contiene all'interno dell'array, sotto il campo *docs,* tutti i dati delle singole misurazioni.
I campi più importanti, in seguito necessari al codice dell'applicazione, sono:

- *startDate* e *endDate*: stringhe che contengono informazioni relative all'aggiornamento dei dati, rappresentate secondo lo standard ISO-8601;
- *values*: l'array contenente le tre concentrazioni dei gas (CO, NO_2 e CH_4).

3.3. Ricezione dati nell'applicazione

Dopo aver verificato la presenza dei dati sul server, non rimane che accertare la corretta ricezione e visualizzazione da parte dell'utente finale, ovvero nell'applicazione.

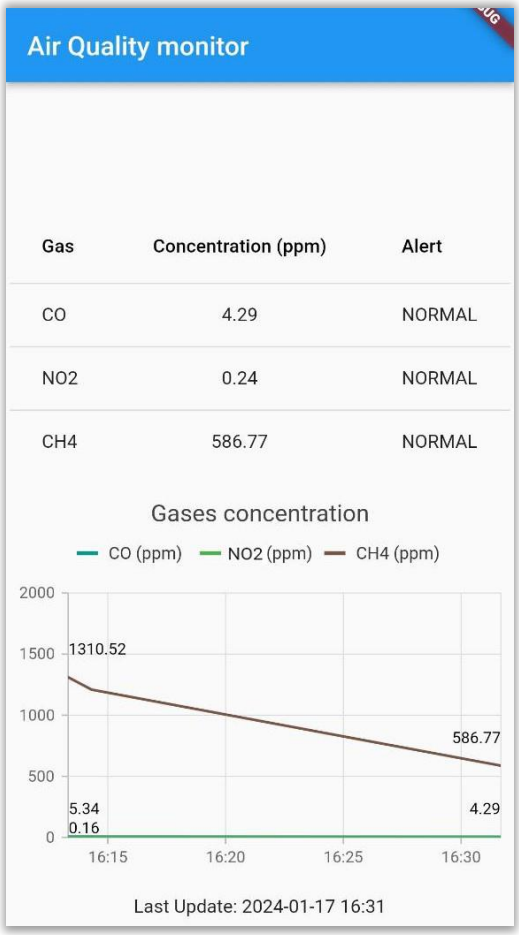

Figura 3.6 - Caricamento dati giornalieri

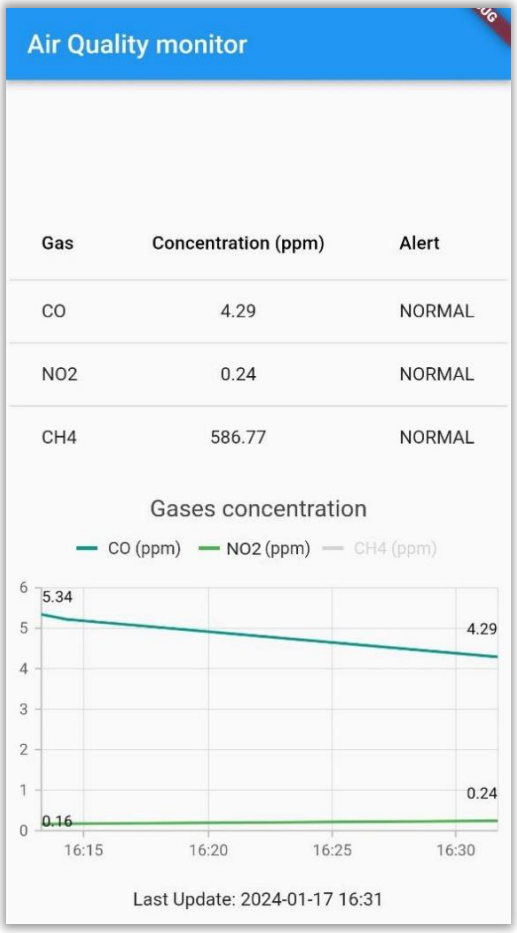

Figura 3.7 - Grafico con oscuramento serie CH_4

Le figure 3.6 e 3.7 mostrano come si presenta l'interfaccia dopo la prima apertura. Data la presenza di dati giornalieri precedenti, vengono popolati la tabella e il grafico.

Occorre ora accertare la corretta comunicazione con FCM e l'invio di notifiche nel caso in cui si presenti uno sforamento di soglia.

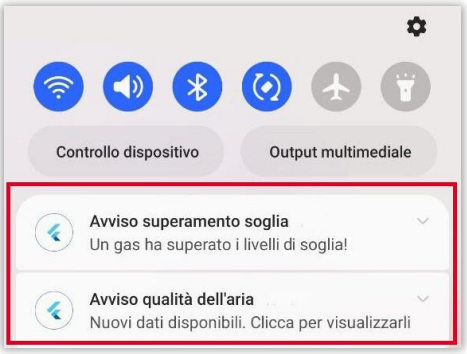

Figura 3.8 - Notifica presenza nuovo dato e superamento soglia

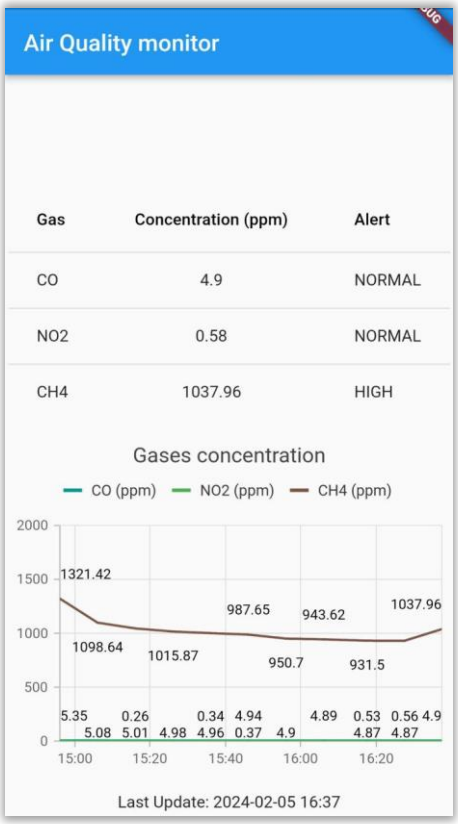

Figura 3.9 - Apertura app dopo ricezione notifica

Nel momento in cui un nuovo dato è disponibile, l'utente riceve immediatamente una notifica sullo smartphone (Figura 3.8). Nel caso specifico, se una delle concentrazioni di gas supera la soglia di guardia predefinita, l'utente riceve una notifica istantanea per essere prontamente informato della situazione.

Se l'utente interagisce con la notifica cliccando su di essa, si verifica un aggiornamento in tempo reale delle concentrazioni della tabella nell'applicazione. Verrà inserito un avviso testuale specifico relativo al gas che ha superato la soglia di guardia.

Allo tempo stesso, avviene l'aggiornamento dei dati all'interno del grafico, includendo le informazioni relative all'ultima ricezione.

4. Contributo personale e considerazione conclusive

Il mio contributo si è focalizzato nell'acquisizione dei dati, nella verifica dei limiti delle concentrazioni di gas e nella gestione delle notifiche e dell'interfaccia sul dispositivo finale dell'utente.

Ho adottato l'approccio delle medie orarie per fornire una rappresentazione più stabile e accurata delle concentrazioni di gas nell'ambiente circostante. Questa scelta è stata motivata dall'obiettivo di mitigare le variazioni di breve durata che potrebbero verificarsi e potenzialmente falsare i risultati.

Ho avuto un ulteriore riguardo circa la continuità delle misurazioni, implementando una logica che evitasse la perdita di dati in assenza di connessione, in attesa del suo ripristino.

Altro significativo contributo personale è stato apportato alla fase di implementazione del codice Dart per la gestione delle notifiche Firebase. Inizialmente, mi sono confrontato con un esempio di codice fornito, il quale, tuttavia, era stato aggiornato diverso tempo fa e non era più compatibile con le attuali versioni delle librerie. Questo ha rappresentato una sfida iniziale, che ho affrontato con determinazione riscrivendo e aggiornando autonomamente il codice.

Per ragioni di tempo e complessità, ci si è focalizzati su una soluzione di monitoraggio in postazione fissa, ovvero dove è disponibile una copertura Wi-Fi e alimentazione di rete. Tuttavia, una futura prospettiva di sviluppo potrebbe portare l'integrazione di una batteria e un modulo SIM. Questa aggiunta consentirebbe di rendere il sistema autonomo e indipendente dalle infrastrutture locali di alimentazione e connettività, estendendo la sua operatività non solo in ambito lavorativo, ma anche in diversi contesti ambientali.

5. Riferimenti bibliografici

[1] Agenzia Europea dell'Ambiente: https://commission.europa.eu/index_en

[2] Organizzazione Mondiale della Sanità: https://www.who.int/

[3] Arduino Uno WiFi Rev2: https://docs.arduino.cc/hardware/uno-wifi-rev2/

[4] Grove – Multichannel Gas Sensor: https://wiki.seeedstudio.com/Grove-Multichannel_Gas_Sensor/

[5] Protocollo I^2C: https://it.wikipedia.org/wiki/I%C2%B2C

[6] Measurify: https://measurify.org/

[7] JSON (JavaScript Object Notation): https://www.json.org/json-it.html

[8] Postman: https://www.postman.com/

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

[10] Flutter: https://flutter.dev/

6. Ringraziamenti

Desidero ringraziare il Prof. Riccardo Berta che mi ha permesso di realizzare questo progetto, accettando la mia proposta di tesi. Un ringraziamento speciale va al Dott. Matteo Fresta, per i suoi consigli e il suo aiuto durante lo svolgimento. Grazie alla mia famiglia e ai miei amici per avermi sostenuto durante il mio percorso universitario.