Relazione di Progetto di "Smart City e Tecnologie Mobili"

SMART PM DETECTOR

Numero del gruppo: 83

Componenti: Alessia Ventani, Simone Venturi

Indice

	pagina
1. Introduzione	3
2. Stato dell’arte	4
3. Analisi dei requisiti	5
4. Progettazione	8
5. Implementazione	17
6. Testing e performance	25
7. Analisi e deployment su larga scala	30
8. Piano di lavoro	33
9. Conclusioni	36
10. Riferimenti bibliografici	37
11. Indice immagini	38
12. Indice tabelle	38

1 Introduzione

L'aria che respiriamo ogni giorno è un parametro che viene utilizzato per valutare la qualità della vita in un determinato Paese. Non sempre questa è perfetta, anzi, l'azione dell'uomo e alcune sorgenti naturali producono sostanze piccolissime che aleggiano nell'aria e vengono trasportate. Queste piccole particelle sono chiamate particolato e costituiscono un fattore importante nella valutazione della qualità dell'aria, in quanto sono potenzialmente molto dannose per l'uomo.
Il particolato viene suddiviso in sottocategorie in base al diametro delle particelle prese in esame, in particolare:

- PM_{10}: particelle con dimensioni minori di 10 μm, pericolose poiché sono in grado di raggiungere la trachea e i bronchi;
- $PM_{2.5}$: particelle con dimensioni minori di 2.5 μm, ancora più pericolose in quanto sono in grado di raggiungere gli alveoli e attraverso questi trasmettersi nel sangue.

Il progetto ha come obiettivo quello di creare un sistema modulare in grado di rilevare e monitorare la qualità dell'aria della zona nella quale sono posizionate delle piccole stazioni di rilevamento utilizzando un budget ridotto. In particolare, il progetto consiste nel progettare ed assemblare delle postazioni di rilevamento contenenti sensori a basso costo al fine di rilevare la concentrazione delle polveri sottili (pm 2.5), la temperatura, l'umidità e la posizione geo-spaziale. Queste postazioni devono essere in grado di generare dati nel tempo e poterli raccogliere in una base di dati. Dopo aver dato una persistenza ai dati, questi potranno essere consultati o usati per elaborazioni da servizi esterni.

Il gruppo si pone come obiettivo quello di costruire del software efficiente e modulare che riesca ad acquisire correttamente i valori dai sensori attraverso l'uso di un microcontrollore, a tal proposito verrà utilizzato il seguente hardware:

- Arduino Uno;
- Raspberry pi 3;
- DHT22: sensore di umidità e temperatura;
- DSM501A: sensore ottico di rilevamento polveri;
- GPS NEO-6M: sensore GPS.

Inoltre, il gruppo deve trovare una soluzione intelligente riguardo il problema della gestione dei dati (prodotti da, potenzialmente, un numero elevato di postazioni di rilevamento), che devono essere accessibili da servizi esterni, e di generare uno scenario di utilizzo di questi. Il caso d'uso che si è scelto di ricreare è la consultazione tramite web app dei dati raccolti selezionando la stazione di rilevamento desiderata. I dati possono essere visualizzati parzialmente in forma grafica attraverso diagrammi a linee, e filtrabili al fine di ricercare particolari situazioni verificate nel tempo.

2 Stato dell'arte

Lo scopo principale del sistema, come specificato nel paragrafo precedente, è la misurazione della concentrazione del particolato nell'aria.
Attualmente esistono diverse tecnologie per la misurazione di questo fenomeno:

- sensori elettrochimici: analizzano il risultato della reazione chimica fra i gas presenti nell'aria e l'elettrodo presente nel sensore. Il loro prezzo può variare in base alla categoria specifica, da 10 a 150 euro, ma il risultato della misurazione dipende dalla temperatura e dall'umidità;
- rivelatore di fotoionizzazione: ha un costo relativamente elevato, dai 400 ai 5000 euro, ma non è sensibile ai cambiamenti climatici; (1)
- contatori ottici di particelle/ sensori ottici: basati anche sulla tecnologia del light scattering.

Per il progetto si è deciso di utilizzare quest'ultima categoria poiché ha un ottimo rapporto qualità/prezzo e permette di realizzare delle stazioni a basso costo. Inoltre, questi sensori sono facilmente reperibili e integrabili con processori a basso costo come Arduino e Raspberry. Infine, come riportato in uno studio del 2015 (2), raggiungono un ottimo grado di precisione nella misurazione.

Sul mercato esistono diverse soluzioni proprietarie basati su questa tipologia di sensori (3). Un esempio è dato dalla soluzione proposta da Aeroqual con il prodotto Micro Air Quality Monitoring System. Esso permette di misurare la concentrazione di PM, di Ozono, di Diossido di Azoto, temperatura e umidità. I sensori utilizzati da questa soluzione sono prodotti dall'azienda stessa ma, come può essere letto nel foglio di specifiche del prodotto, la tecnologia utilizzata per la rilevazione dei Pm è il Laser scattering. (4) Una volta comprato il prodotto, è poi possibile accedere ai dati da una interfaccia web, sotto licenza software, o con una API open.

Volendo confrontare questa soluzione con il progetto realizzato, si sono utilizzati dei sensori basati sulle stesse tecnologie e si è progettata una architettura per poter accedere ai dati da remoto, sia con una api e da una interfaccia remota. Come processore, non potendo realizzare una scheda apposita come nella soluzione di Aeroqual, si utilizzano un Arduino e un Raspberry .

Nella documentazione di Arduino esistono diversi progetti di esempio che costituiscono un buon punto di partenza (5)(6). Queste sono molto simili al modello realizzato e utilizzano gli stessi sensori ma realizzano delle stazioni in cui i dati sono visualizzabili sono da uno schermo e in locale. La soluzione proposta invece ha una architettura che permette la visualizzazione dei dati da remoto e la possibilità di creare una rete di stazioni controllabili da un unico punto.

In generale in letteratura e nella documentazione delle tecnologie utilizzate, sono presenti diversi esempi che possono essere utilizzati come modelli ma che non rispecchiano a pieno il sistema realizzato che ha l'obiettivo di produrre una stazione a basso costo, riutilizzabile e che può essere replicata sul territorio con un budget limitato.

3 Analisi dei requisiti

Di seguito vengono riportati i principali requisiti che il progetto deve soddisfare.

I requisiti di business

Al fine di individuare quali potrebbero essere i requisiti principali sulla realizzazione di questo progetto si è simulata una intervista con un tecnico specializzato nella raccolta dati ed esperto nel settore.

Quindi sono stati evidenziati i seguenti aspetti fondamentali:

- è necessario raccogliere dati utilizzando dei sensori che possano avere una precisione accettabile al fine di avere dei dati attendibili;
- oltre al dato relativo alla percentuale del particolato è necessario associare ad esso anche i valori dell'umidità e della temperatura poiché questi influiscono sulla rilevazione ottenuta;
- la qualità dell'aria è strettamente legata alla zona geografica e all'ambiente in cui si rilevano i dati ed è quindi necessario salvare la posizione esatta in cui la stazione è stata posizionata;
- occorre avere una persistenza dei dati in maniera da poterli analizzare ed effettuare dei confronti;
- è fondamentale poter filtrare i dati temporalmente e riuscire a visualizzarli da remoto con l'ausilio di grafici.

In seguito si è simulata anche una intervista con un utente normale, che non ha una conoscenza specifica ma vuole approcciarsi al problema e misurare la qualità dell'aria. Analizzando un utente con un background diverso dal precedente sono emersi ulteriori aspetti:

- il prezzo ha un ruolo fondamentale: se viene mantenuto basso allora un utente è più invogliato a realizzare questo progetto e ad incrementare la quantità di dati raccolti;
- la stazione per la raccolta dati deve essere utilizza facilmente e di dimensioni piccole così da poter essere maneggiata facilmente;
- i dati devono essere rappresentati in maniera chiara e con una legenda riguardante i valori ottimali dei dati raccolti.

I requisiti funzionali

Di seguito i requisiti scaturiti dall'analisi precedente:

1. la stazione deve essere in grado di raccogliere diversi dati e, quindi, devono essere presenti i sensori per il rilevamento di: temperatura, umidità e livello di particolato ($PM_{2.5}$ e PM_{10})
2. la stazione deve avere due componenti principali:
    a. una componente per la raccolta dei dati dai sensori;
    b. una seconda componente per l'invio dei dati ad un servizio esterno per il salvataggio e per garantire la loro persistenza;
3. la stazione deve essere in grado di identificare la sua posizione e quindi deve essere presente un sensore GPS in grado di comunicare le coordinate geografiche;

4. i dati devono poter essere raccolti in un unico punto da remoto, è dunque necessario che le stazioni siano in grado di comunicare in rete, in particolare avendo una interfaccia di rete WiFi capace di trasmettere i dati;
5. la stazione deve registrarsi al servizio per il salvataggio dei dati ed essere identificabile rispetto alle altre;
6. il database in cui vengono raccolti i dati deve essere accessibile e deve essere in grado di raccogliere una grande mole di dati offrendo una API per la scrittura e la lettura dei dati;
7. i dati devono essere consultabili da remoto con una interfaccia grafica;
8. deve essere possibile visualizzare l'elenco di stazioni registrate nel database e distinguere quella di interesse in due modalità:
    a. cercando la stazione desiderata attraverso l'identificativo che la contraddistingue;
    b. cercando la stazione in una mappa e individuandola attraverso la sua posizione geografica;
9. l'interfaccia deve mostrare gli ultimi dati raccolti in maniera chiara attraverso l'ausilio di grafici;
10. l'interfaccia deve permettere di filtrare i dati per tipologia di dato (umidità, temperatura…) e per arco temporale desiderato;
11. l'interfaccia grafica deve essere consultabile in maniera comoda anche da tablet e supporti mobili;
12. se i dati cambiano all'interno del database, o ne vengono aggiunti di nuovi, i dati visualizzati nell'interfaccia devono aggiornarsi di conseguenza in automatico e in sincrono.

I requisiti non funzionali

Il sistema progettato deve tener conto dei seguenti aspetti:

- scalabilità: l'architettura complessiva deve essere pensata per poter essere scalabile facilmente. Più sensori sono presenti nel territorio maggiore è la copertura e maggiore è la quantità di dati che possono essere analizzati. Per questo è necessario avere un database in grado di gestire volumi di dati elevati e che sia semplice aggiungere una nuova stazione e di conseguenza i suoi dati;
- disaccoppiamento raccolta dati/persistenza dei dati: i dati devono, una volta raccolti, devono essere salvati e consultabili indipendentemente dal funzionamento delle stazioni. Inoltre deve essere possibile accedere i dati da remoto e, volendo, poter costruire diversi metodi di consultazione sui dati raccolti;
- robustezza: alle stazioni, quando sono funzionanti, deve essere possibile comunicare i dati da salvare ad un servizio sempre disponibile e resistente ad eventuali guasti. Esso inoltre deve essere disponibile ad eventuali interfacce grafiche che visualizzano ed elaborano i dati. Se le stazioni perdono la connessione ad internet i dati non devono essere persi ma comunicati una volta che viene ripresa la connessione. Non è necessario che i dati raccolti siano inseriti in ordina temporale ma devono essere caratterizzati da un campo che indichi la data e l'orario esatto del campionamento;
- mobilità delle stazioni: le stazioni possono essere mosse da una zona all'altra al fine di poter utilizzare lo stesso hardware e raccogliere dati in luoghi differenti. Il sistema, quindi, deve gestire la possibilità che una stazione possa essere utilizzata più volte ma le serie registrate devono essere distinte e distinguibili. Inoltre, il sistema deve essere pensato affinché possa essere facilmente modificato per raccogliere la posizione nel tempo della stazione nel caso la stazione venga posta su un mezzo in movimento.

I requisiti di implementazione

Dal punto di vista implementativo occorre soddisfare i seguenti requisiti:

- il costo della stazione nel suo totale deve essere relativamente basso: esso deve aggirarsi intorno ai 50/70 euro affinché la stazione possa essere riprodotta senza investimenti enormi in budget e quindi avere un incentivo in più per la sua realizzazione;
- l'hardware utilizzato deve essere "off the shelf". Le soluzioni commerciali sono caratterizzate da sensori e processori appositamente realizzati e studiati dall'azienda produttrice che non permettono il loro utilizzo al di fuori del prodotto finito e comprato. Per poter essere realizzata da chiunque quindi, la stazione deve essere composta da processori e elementi hardware che siano reperibili sul mercato facilmente;
- ogni stazione deve comunicare i dati raccolti ogni 30 secondi al database di riferimento, connessione permettendo.

4 Progettazione

Di seguito vengono riportate le principali scelte progettuali effettuate al fine di garantire un corretto sviluppo dell'elaborato coerentemente con i requisiti evidenziati in fase di analisi.

Design Architetturale

La prima operazione che si è resa necessaria per la definizione dell'architettura del progetto, è stata la definizione della separazione delle funzioni fra i vari componenti che costituiscono il progetto. Analizzando i requisiti riportati nella pagina precedente, si può osservare come è possibile suddividere l'intero progetto in componenti con le seguenti funzionalità:

1. rilevamento dei dati effettuato da una postazione con il compito principale di leggere correttamente i valori dei sensori, produrre dati pronti per essere trasmessi ed infine comunicarli al server;
2. salvataggio dei dati raccolti svolto un database remoto che permette di raccogliere i dati di tutte le stazioni di rilevamento in un unico punto e li rende disponibili per eventuali servizi esterni che devono consultarli;
3. visualizzare i dati attraverso una applicazione web che sfrutta il database remoto e l'api che espone per rendere i dati accessibili all'utente finale.

Questa divisione principale permette di suddividere in componenti indipendenti e isolate l'intero sistema. In questo modo è possibile ottenere una qualità del codice prodotto più elevata ma anche un sistema modulare e scalabile. Questo ultimo dettaglio rende la raccolta dei dati e la loro successiva visualizzazione non vincolata dal numero di stazioni attive e i dati raccolti possono essere analizzati e consultati anche se esse non sono più attive.

Una possibile alternativa a questa architettura è mantenere un database locale in ogni singola stazione. In questo caso però si possono riscontrare diverse problematiche:

- se una stazione non è in rete non è possibile visualizzarne i dati;
- se la stazione cambia posizione o deve essere nuovamente utilizzata, occorre effettuare una copia dei dati raccolti e svuotare il database;
- ogni servizio deve accedere ad ogni stazione aumentano l'overhead;
- ogni stazione deve avere un processore in grado di gestire un database e tutte le operazioni associate ad esso.

Dati questi presupposti, si può affermare che questa strada non è percorribile e che un unico database remoto per ogni stazione risulta essere la soluzione migliore.

Inoltre, avendo un singolo servizio dove i dati vengono salvati e memorizzati, è possibile realizzare un numero elevato di applicativi che sfruttano i dati raccolti e che sono indipendenti fra loro e dalle stazioni. In questo progetto si realizza una sola interfaccia grafica di esempio ma, con questa architettura, è possibile costruirne altre in maniera semplice. Dunque l'intero sistema è facilmente scalabile: sia nel numero di stazioni sia nel numero di servizi di visualizzazione e analisi dei dati.

L'architettura complessiva quindi è riassumibile in Figura 1 e rispetta tutti i requisiti definiti precedentemente.

Figura 1 - Architettura sistema.

Per esemplificare un tipo di servizi che possono essere implementati si è sviluppata una applicazione web che rispecchi i requisiti individuati.

Design di dettaglio

Dall'architettura precedente è stato definito lo schema principale del progetto con le tecnologie sfruttate nelle singole componenti, visibile in Figura 2.

Figura 2 - Schema tecnologie utilizzate.

Di seguito si riporta una descrizione del design di dettaglio di ogni elemento dell'architettura.

Design di dettaglio della stazione

La stazione svolge i compiti di lettura dei valori dai sensori e la produzione da questi di dati strutturati pronti per la trasmissione verso il server che avrà poi il compito di garantire la persistenza e l'uso futuro. Risulta perciò necessaria un'adeguata progettazione dei singoli compiti che la stazione dovrà eseguire, una strategia per la loro integrazione e un modo per mantenere il consumo energetico basso durante la messa in funzione della postazione.

Dopo un’attenta analisi, sono stati delineati tre task:

1. PMDetectorTask: il task che ha il compito di rilevare la concentrazione del particolato nell’aria;
2. GPSTask: il task che ha il compito di collezionare la posizione geospaziale della stazione e il tempo dell’analisi;
3. ComunicationTask: il task che ha il compito della comunicazione dei dati rilevati al fine di garantirgli una persistenza.

Architettura fisica della stazione

La stazione di rilevamento contiene al suo interno il seguente hardware:

- Arduino Uno;
- Raspberry pi 3;
- DHT22: sensore di umidità e temperatura;
- DSM501A: sensore ottico di rilevamento polveri;
- GPS NEO-6M: sensore GPS.

Come mostrato in Figura 3, il microcontrollore Arduino Uno ha la funzione di ospitare i tre sensori, leggere i loro valori e trasmetterli tramite la propria seriale al SoC, System-on-a-chip, al quale è collegato, in questo caso il Raspberry. L’uso di un SoC risulta importante in quanto offre la possibilità di avere connettività ed inoltre lascia aperta la possibilità di ospitare altri microcontrollori che svolgono altre funzioni ed ingrandire la stazione, aggiungendo altre funzionalità.

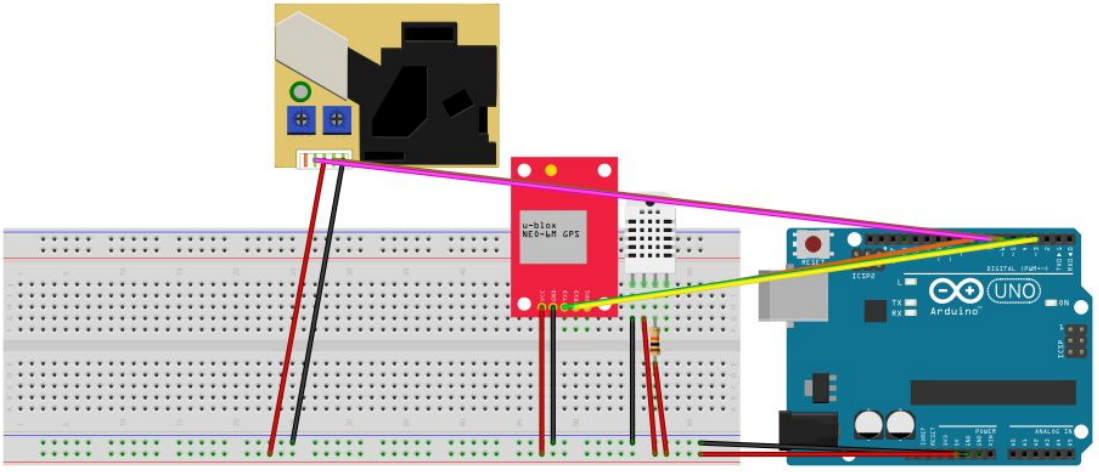

Figura 3 - Schema Arduino e sensori.

Design dei task della stazione

L’individuazione di task in fase di progettazione rende più semplice lo sviluppo del codice in fase di implementazione, inoltre garantirà al software più modularità in quanto il codice sarà separato per compiti. L’esecuzione dei task poi sarà a carico di uno scheduler che conserva in memoria i task e,

ad ogni lasso di tempo predeterminato che trascorrerà, avrà il compito di eseguire se necessario i task. Questo meccanismo è ottimizzato al fine di usare efficientemente l'energia. In Figura 4 viene mostrato il diagramma delle classi del codice progettato.

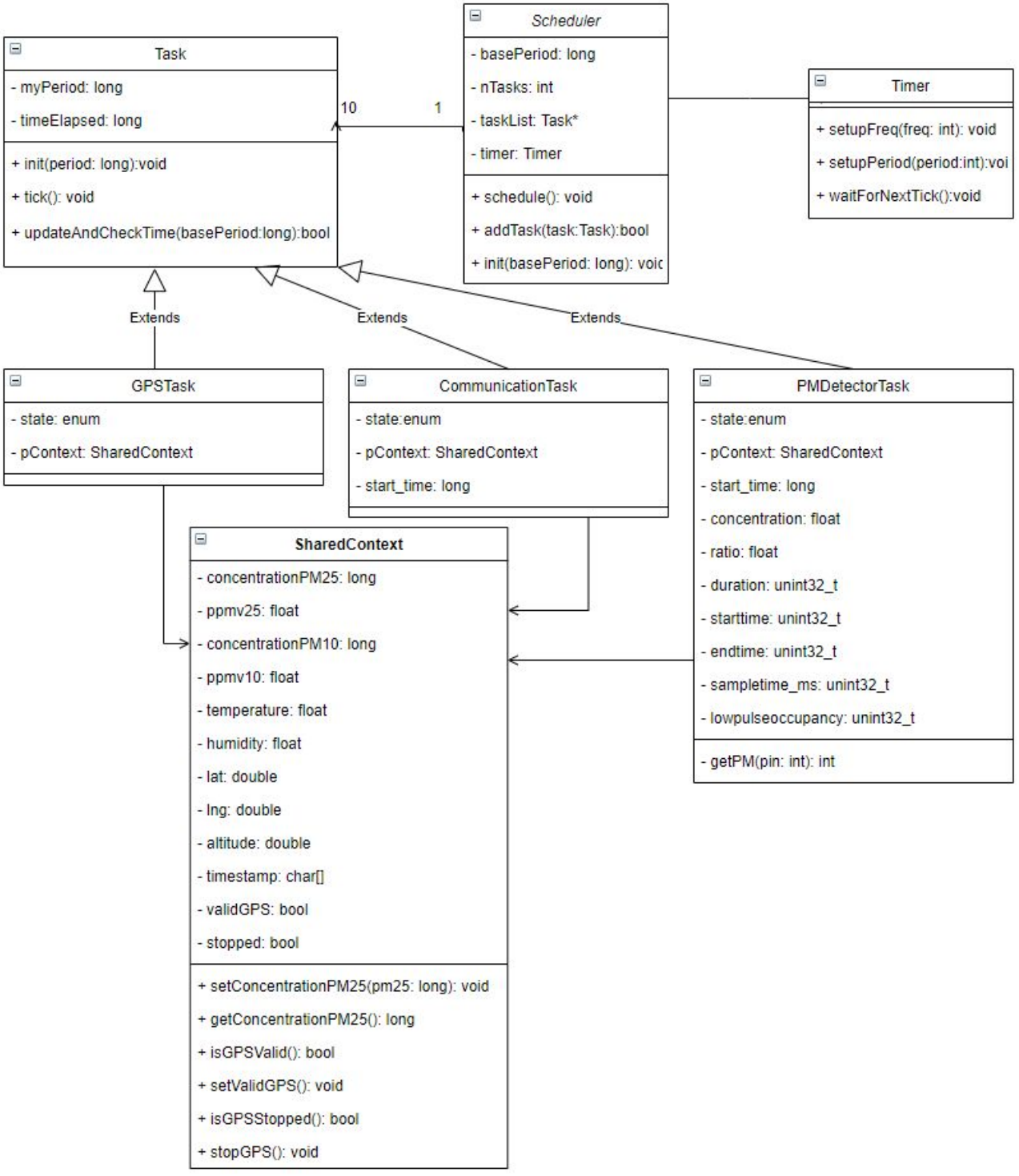

Figura 4 - Diagramma di classe codice Arduino.

PMDetectorTask ha uno stato iniziale IDLE, il sensore infatti come riportato nel suo datasheet ha un tempo di calibrazione iniziale di 60 secondi, trascorso questo tempo il sensore di rilevamento

ottico delle polveri sarà in grado di lavorare correttamente ed il task alternerà i due stati di scanning, uno per il $PM_{2,5}$ l'altro per il PM_{10}, come visibile in Figura 5.

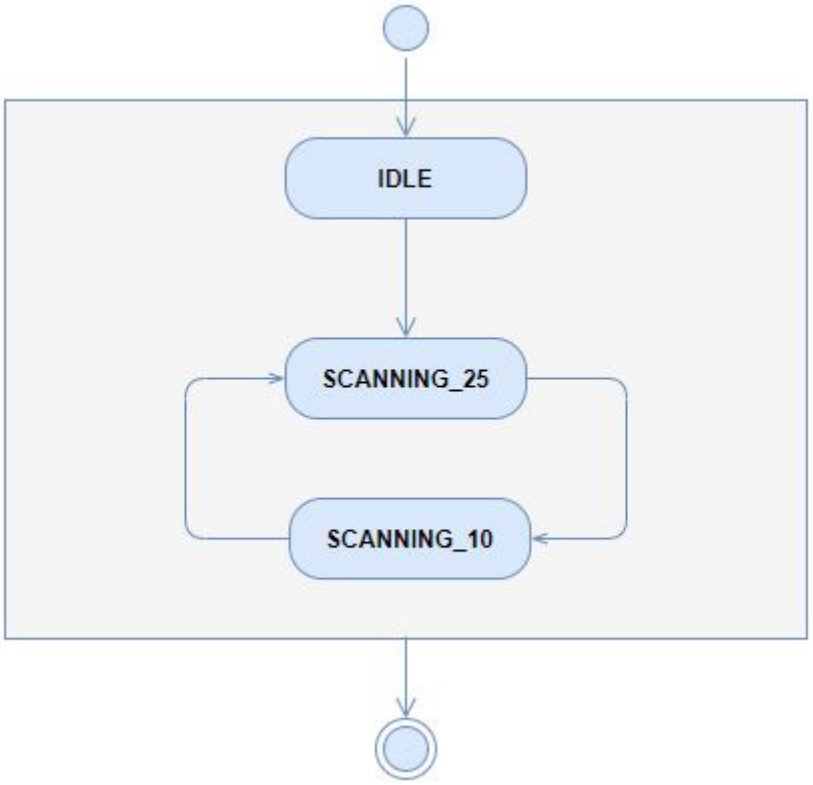

Figura 5 - Diagramma a stato PMDetectorTask

GPSTask ha un design molto semplice, prevede due soli stati nei quali legge i valori dal sensore di localizzazione geospaziale. La scelta dei due stati distinti è legata all'ottimizzazione delle risorse nel lungo periodo di attività della stazione poiché durante, la sua messa in funzione, dovrà comunicare solo una volta la sua posizione, mentre dovrà comunicare costantemente l'istante temporale della rilevazione dei dati.

Dall'analisi dei task eseguita in fase embrionale del progetto, è avvenuto un successivo raffinamento che ha portato alla suddivisione in due parti del task ComunicationTask, in quanto la comunicazione dei dati ha due percorsi distinti: quella necessaria per il trasferimento dei dati tramite seriale da Arduino a Raspberry e quella tra la scheda Raspberry al database.

CommunicationTask è stato progettato seguendo strettamente le funzionalità identificate in fase di analisi. La comunicazione tra la scheda Arduino e il SoC Raspberry è prevalentemente unidirezionale, ovvero il microcontrollore deve comunicare con frequenza pari a 30 secondi le registrazioni effettuate in quel lasso di tempo. La bidirezionalità, invece, è utilizzata solamente in uno stato del task, quello di READING, presente unicamente al fine di garantire robustezza alla stazione. Infatti lo stato in questione ascolta la risposta del Raspberry dopo il primo invio da parte dell'Arduino dei dati relativi alla posizione della stazione. Una volta registrato il record, la stazione dovrà registrare solamente i dati relativi alla qualità dell'aria. Questo è gestito dallo stato SEND_DATA che non ha bisogno di alcun feedback su

seriale. Il design del task è avvenuto in fase postuma rispetto al design degli altri task più chiari dopo l'analisi dei requisiti ed è mostrato in Figura 6.

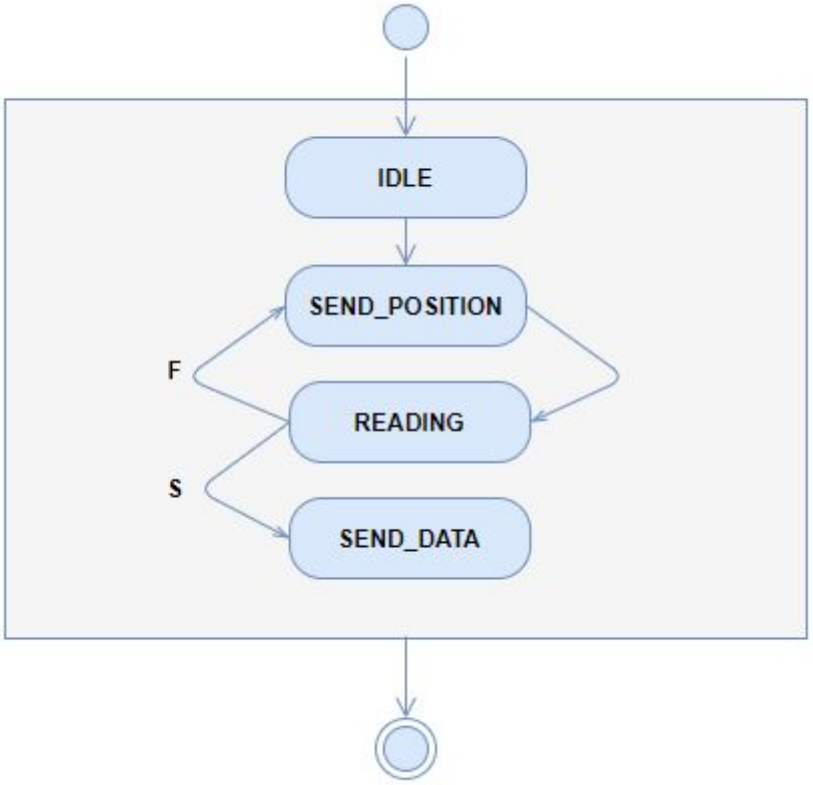

Figura 6 - Diagramma a stato CommunicationTask .

L'ultima classe individuata in fase di progettazione è SharedContext che ha la funzione di mantenere uno stato condiviso e mutevole del sistema, il suo scopo principale è quello di raccogliere al suo interno ogni variabile condivisa da più task o tenere in memoria variabili ausiliarie al fine di un corretto coordinamento dei task.

Design di dettaglio del database

Dall'architettura che si è progettata risulta chiaro che il database per la raccolta dei dati rappresenta l'elemento centrale ed è fondamentale che sia sempre accessibile sia dalle stazioni che dall'applicazione.

Essendo un sistema distribuito, inoltre, è necessario garantire l'integrità dei dati e che il database sia in grado di gestire un elevato numero di connessioni. Assicurare l'accessibilità di un unico database presente su una macchina fisica e uno spazio di storage potenzialmente elevato su un server implementato ad hoc è molto complesso.

Per ovviare a questa difficoltà e semplificare il lavoro, si è deciso di utilizzare la piattaforma per lo sviluppo di applicazioni Firebase di Google (7) e in particolare, dei servizi che questa piattaforma offre, si è deciso di utilizzare Firestore, (8) un database NoSQL flessibile e scalabile real-time. Esso permette il salvataggio dei campionamenti inviati dalle stazioni in documenti e di organizzare

quest'ultimi in collezioni, inoltre ha un supporto nativo per interrogazioni complesse e articolate. La sua caratteristica peculiare è la possibilità di sincronizzare tutte le applicazioni che si appoggiano ad esso, ai dati che vengono salvati e quindi l'interfaccia web per la visualizzazione di essi, si aggiornata in tempo reale e in automatico ogni volta che una stazione scrive un nuovo dato sul database. In questa maniera si permette all'utente di avere sempre sott'occhio l'ultima versione dei dati.

Essendo un servizio basato sull'infrastruttura di google, vengono garantite le seguenti caratteristiche: replicazione dei dati multi-regione, consistenza dei dati, atomicità delle operazioni base, protezioni dei dati attraverso autenticazione e supporto, appunto, per le transazioni real-time.

I dati all'interno del database sono suddivisi per stazione, infatti ad ognuna di esse corrisponde una collezione che contiene tutti i dati raccolti da essa. Inoltre vi è una collezione che racchiude tutte le informazioni riguardanti le stazioni.

L'interazione con questo tipo di database distribuito avviene attraverso l'api fornita dallo stesso Firestore che permette di scrivere e leggere ma anche di agganciare dei listener al database che permettono la sincronizzazione real-time fra applicazioni e database.

Design di dettaglio dell'interfaccia web

Partendo dall'architettura che è stata progettata, è possibile realizzare diverse tipologie di applicazioni che possono collegarsi al database di Firestore. Il progetto si pone l'obiettivo di realizzarne una di esempio che rispetti i requisiti individuati nella fase di analisi.

Dalle interviste con gli utenti, è stato sottolineato il fatto che questa applicazione debba essere consultabile sia da un personal computer che da un dispositivo mobile come un tablet. Il design dell'applicazione quindi deve essere pensato per adattarsi ad entrambe le situazioni.

Nel caso d'uso realizzato, sono stati esclusi gli smartphone dall'elenco dei dispositivi da considerare per la progettazione. Questa scelta è stata effettuata poiché l'analisi dei dati o la visualizzazione di grafici risulta molto più scomoda da un dispositivo con uno schermo di piccole dimensioni. Ciò non esclude il fatto che l'applicazione possa essere utilizzata anche da questi ultimi ma il suo design in questo caso non sarà ottimale proprio perché pensato per un altro genere di dispositivi.

Dai requisiti che si sono individuati è possibile identificare quella che è la struttura generale della pagina web, osservabile in Figura 7.

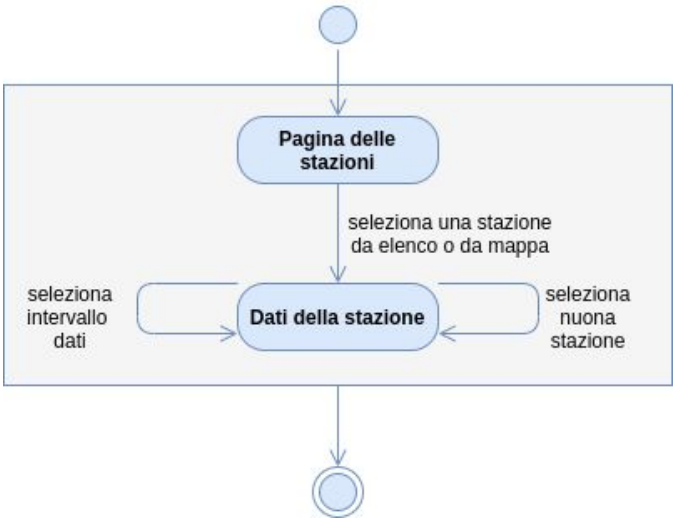

Figura 7 - Diagramma navigazione applicazione web.

Partendo da questi presupposti si è deciso di progettare una applicazione con Vue (9), utilizzato molto spesso per costruire interfacce utenti. Con l'ausilio di questo progressive framework si è in grado di costruire applicazioni single-page in maniera moderne e interagire con Firebase grazie alle librerie già presenti e installabili.

Con questo framework la struttura generale del codice è già definita ed è costituita da due componenti fondamentali:

- **views:** rappresentano le pagine del sito web e hanno degli url associati ad esse;
- **components**: elementi che vengono utilizzati più volte nel codice e che vengono richiamati dalle views quando viene caricata la pagina o quando si compie un'azione. Permettono di non avere ridondanze del codice e di non caricare una nuova pagine ad ogni singola modifica. Con le views è possibile realizzare pagine reattive e modificabili.

Quindi gli elementi fondamentali dell'app sono:

- due views principali:
    - una per la selezione della stazione da consultare attraverso una mappa o con l'elenco delle stazioni;
    - una seconda con tre sezioni principali: i grafici per visualizzare gli ultimi dati, un grafico per filtrarli temporalmente e l'elenco delle stazioni per cambiare i dati da visualizzare;
- due componenti per i grafici:
    - un grafico che in maniera sincrona al database visualizza gli ultimi dati inseriti nel database specificando la tipologia da visualizzare (dati sul particolato, umidità o temperatura) e il codice della stazione di riferimento. Esso deve avere la caratteristica di aggiornarsi in automatico se vengono aggiunti nuovi dati a firebase;
    - un grafico che mostra una specifica tipologia di dati di uno specifico periodo temporale. A differenza del componente precedente non si aggiorna e filtra i dati per una data-ora iniziale e una finale.
- in aggiunta si è sfruttato il componente per visualizzare elementi sulla mappa di google maps che scaricabile dal repository npm: GmapMap.

Oltre agli elementi descritti precedentemente è necessario definire alcuni file aggiuntivi:

- main.js per le impostazioni generali dell'applicazioni;
- router.js che definisce il router per passare da una views all'altra;
- firestore.js con le impostazioni per l'accesso a Firestore.

La struttura del codice quindi risulta è rappresentata in Figura 8.

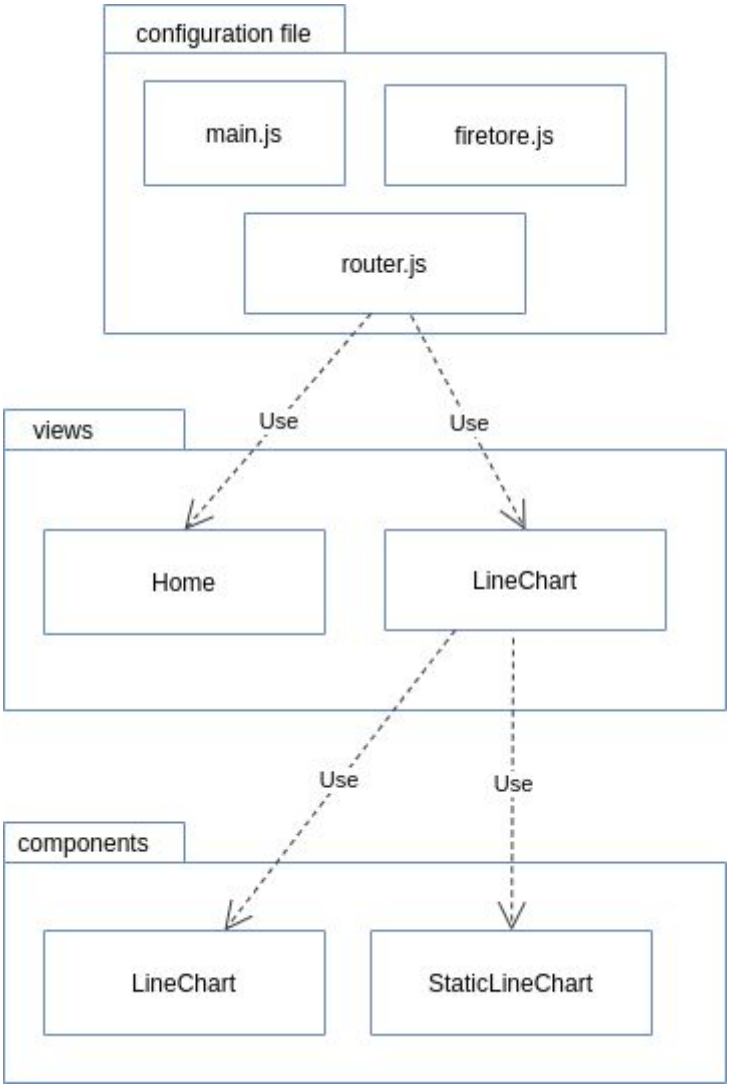

Figura 8 - Diagramma di package applicazione web.

5 Implementazione

Di seguito sono elencate le scelte implementative dei vari componenti del sistema.

Implementazione della stazione

Dalla progettazione eseguita precedentemente, sono stati implementati i task seguendo abbastanza fedelmente quanto riportato nella fase di design. I task, così come sono stati progettati, risultano di semplice comprensione e racchiudono al loro interno unicamente la logica della loro funzione:

- GPSTask esegue, ogni 20 secondi, una lettura dal sensore di geolocalizzazione e salva i valori di interesse nell'unico oggetto condiviso di tipo SharedContext;
- PMDetectorTask è un task che viene eseguito ogni 15 secondi e, come mostrato nella progettazione di dettaglio, alterna i due compiti di rilevamento della concentrazione delle polveri di diametro inferiore a 2.5µm ed inferiore a 10µm. Il campionamento del sensore ottico dura 3 secondi, al termine dei quali vengono presi i valori di temperatura ed umidità dell'aria e calcolata la concentrazione del particolato;
- CommunicationTask esegue come da requisito una trasmissione su seriale ogni 30 secondi.

Lo scheduler che è stato implementato al fine di eseguire i task è di tipo cooperativo, esegue la sua routine ogni 5 secondi. La frequenza pari a 5 secondi è stata scelta poiché questo valore deve essere necessariamente un divisore comune dei valori di esecuzione periodica assegnati ai task, inoltre questo numero non può essere minore al tempo di esecuzione dei task stessi. Considerando il Worst-Case-Execution-Time, ovvero il tempo necessario per l'esecuzione di tutti e tre i task durante lo stesso periodo di schedulazione, questo valore è mediamente pari a poco più di 3 secondi e sempre inferiore ai 4 secondi, il che rende la frequenza dello scheduler di 5 secondi un valore idoneo.

Il software prodotto che concerne il componente Raspberry, invece, è piuttosto semplice e prevede:

- la lettura su seriale periodica con periodo pari a 25 secondi;
- la correzione di errori di rilevamento da parte della stazione, ad esempio il sensore GPS ha un periodo di calibrazione abbastanza lungo e potrebbe non fornire dati temporali corretti per parte della sua esecuzione o non fornire dati attendibili in particolari situazioni;
- la scrittura su database remoto dei record;

Le principali difficoltà che sono state incontrate durante la fase implementativa della stazione di raccolta dati sono state legate all'obiettivo di dare una robustezza alla postazione stessa. Infatti, si è cercato in ogni modo di correggere quanti più errori legati alla sensoristica possibili, inoltre di gestire adeguatamente l'evenienza di assenza di connessione. Quest'ultima casistica è stata trattata anche in fase di design ed ha portato ad un risultato finale notevole, infatti la stazione non prosegue il suo funzionamento se non ha eseguito la registrazione della propria posizione esattamente una volta e, durante la raccolta dati, tutti i record che non sono stati salvati nella base di dati remota vengono conservati e provati a salvare in un momento successivo.

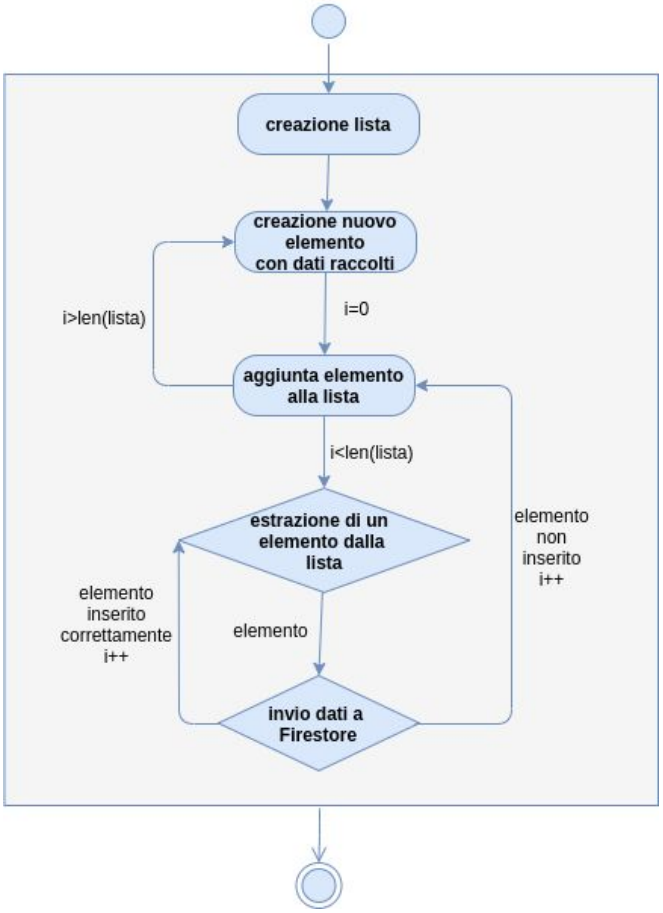

Figura 8 - Diagramma flusso controllo invio dati Rasperry-Firestore.

Questo comportamento del Raspberry è stato implementato come è mostrato in Figura 8. I dati sono salvati in una struttura interna e, se non vengono inviati al database, vengono mantenuti all'interno di essa fino a quando non sarà possibile inviarli e salvarli nel database.

Implementazione del database su Firestore

Come è stato già illustrato nel capitolo precedente, per il salvataggio dei dati si è deciso di utilizzare la piattaforma Firebase. Fra i due principali servizi offerti per lo storage, la scelta è ricaduta su Firestore: esso, rispetto al più semplice servizio di salvataggio di dati real-time, permette di organizzare i dati in documenti e dividerli all'interno di collezioni.

Inoltre è possibile, sfruttando l'api di Firestore, implementare un listener su una determinata collezione e così poter sincronizzare l'applicazione web con i cambiamenti effettuati.

Dal punto di vista implementativo si è dovuto decidere come organizzare i dati all'interno di Firestore. La struttura scelta per le informazioni può essere descritta come segue:

- è presente un'unica collezione per contenere tutti i documenti con le informazioni relative alle stazioni, denominata "stations";

- sono presenti diverse collezioni, pari al numero di stazioni registrate e aventi lo stesso nome, che contengono tutti i documenti relativi ad esse, ovvero le misurazioni effettuate nel tempo dalla stazione di afferenza.

Effettuata questa prima distinzione si è poi deciso lo schema principale dei documenti che sono presenti nelle collezioni.

Quindi, un documento appartenente alla collezione “stations” ha la struttura riportata in Figura 10.

Figura 10 - Documento di una stazione.

Dunque questo tipo di documento racchiude le informazioni spaziali e l’identificativo della stazione.

Per ogni stazione questo documento viene riempito all’avvio e nella fase di registrazione di quest’ultima.

Ogni stazione, inoltre, ha una collezione denominata come il suo identificativo e al suo interno ha dei documenti che hanno uno schema definito, visibile in Figura 11.

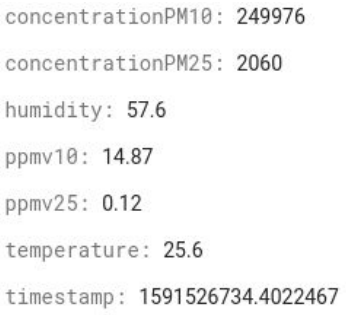

Figura 11 - Documento con i dati raccolti dai sensori.

All’interno di questo tipo di documento troviamo i dati relativi alle misurazioni effettuate dai sensori della stazione riguardo al particolato, PM10 e PM2.5, i rispettivi valori in parte per milione, ppmv10 e ppmv2.5, la temperature, l’umidità e infine il momento temporale in cui è stata fatta la misurazione, espresso in epoche di Unix.

Implementazione dell’applicazione web

Come descritto nel capitolo precedente, per lo sviluppo di un’applicazione web di esempio si è deciso di utilizzare il framework Vue.js. Questa scelta è stata presa poichè con l’ausilio di questo strumento è molto semplice realizzare applicazioni web mobile-first, in questo caso devo essere visualizzate da tablet anche, moderne e dalla grafica semplice e efficace. Infatti dal repository online npm, è possibile scaricare una grande quantità di componenti che permettono di

applicazione single-page in pochi passi e usando i principali standard per il web. Inoltre si ha la possibilità di integrare Vue con un back end e quindi un database interno, ma anche di sfruttare api esterne e quindi poter collegare l'app, come nel caso illustrato, a Firestore.

Il codice dell'applicazione che è stata sviluppata, essendo un esempio, è molto semplice e facilmente riproducibile e anche migliorabile. Infatti il sito web è composto da pochi elementi ma di certo la parte più complessa è stata la realizzazione dei componenti che fossero aggiornati real-time con il database, senza aver bisogno di ricaricare la pagina più volte.

Dunque l'interazione con il database, che riprende la struttura illustrata nei capitoli precedenti, rappresenta la componente principale dell'applicazione e per implementarla si è deciso di utilizzare il plug-in di Vue per Firestore (10).

La comunicazione fra applicazione e firestore avviene in determinati momenti e da parte di specifici elementi. Quando l'utente accede alla pagina principale si stabilisce un primo collegamento con il database, sfruttando il plug-in installato.

```
db.collection("stations").onSnapshot(ref => {
ref.docChanges().forEach(change => {
  const { newIndex, oldIndex, doc, type } = change
  if (type === 'added') {
    this.stations.splice(newIndex, 0, doc.data())
  } else if (type === 'modified') {
    this.stations.splice(oldIndex, 1)
    this.stations.splice(newIndex, 0, doc.data())
  } else if (type === 'removed') {
    this.stations.splice(oldIndex, 1)
  }
  this.takeposition()
 })
})
```

Figura 13 - Codice elenco stazioni.

Con il codice mostrato in Figura 13, si sta effettuando uno snapshot del database e ogni volta che un documento viene aggiunto, eliminato o modificato nella collezione "stations", si crea un evento che viene gestito dal codice. In questa maniera si riesce ad ottenere una lista real-time delle stazioni presente nel database senza che l'utente debba ricaricare la pagina.

Un meccanismo analogo viene utilizzato nel componente LineChart per ottenere una lista aggiornata delle misurazioni effettuate dalle stazioni e inserite nel database.

L'altro momento in cui occorre una interazione con firestore è nel componente StaticLineChart. Come si può intuire dal nome assegnato, in questo caso di effettua una semplice query sul database con filtraggio per avere solo i dati di nostro interesse e il risultato non si aggiorna ma viene solo rappresentato nel grafico, codice in Figura 14.

```
    this.renderChart(getData2Visualize(this.samples, this.data, colors[this.data]), this.options)
 })}
```

```
db.collection(this.collection).where('timestamp', '>=', Math.round(this.starttimestamp.getTime()/1000))
.then(snapshot => {
  snapshot.forEach(doc => {
            let item = doc.data()
            item.id = doc.id
            this.samples.push(item)})
```

Figura 14 - Codice componente StaticLineChart.

Terminata l'integrazione con il database e la realizzazione dei componenti, si sono implementate le pagine principali con la struttura descritta nel capitolo precedente.

Il sito si compone di due pagine, quindi rispettive view in Vue, nella prima, denominata Home, è possibile selezionare la stazione di cui si vogliono analizzare i dati con due elementi: una lista degli identificativi delle stazioni e una mappa nella quale le stazioni sono posizionate in base alle loro coordinate geografiche.

Il risultato finale è riportato in Figura 15.

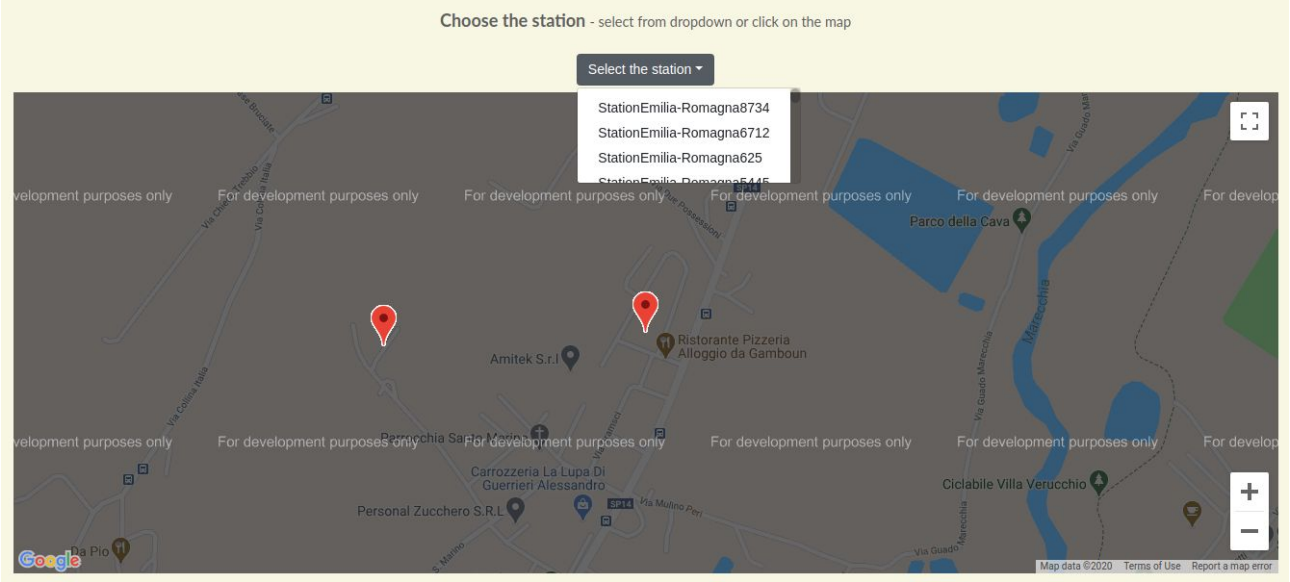

Figura 15 - La mappa delle stazioni.

La mappa di Google è stata realizzata con l'ausilio del componente già pronto vue2-google-maps (11). La mappa è esattamente quella utilizzata dal servizio Google, in essa però compare la scritta "for development only" perchè, per utilizzarla in un ambiente di produzione, occorre associare al proprio account con cui si inizializza la key per i servizi google, un sistema di pagamento. Essendo una web-app di prova questa operazione non è stata fatta.

Una volta che una determinata stazione è stata scelta da parte dell'utente finale, viene caricata una seconda view, denominata LineChart, che si compone di 4 sezioni che sono costituite dal 4 elementi di tipo Card.

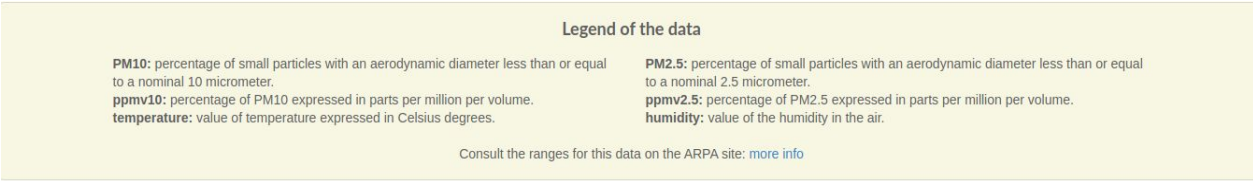

Figura 16 - La legenda.

La prima Card, Figura 16, contiene alcune spiegazioni basilari sui dati raccolti e un link dove poter approfondire e confrontare i valori rilevati con gli standard italiani.

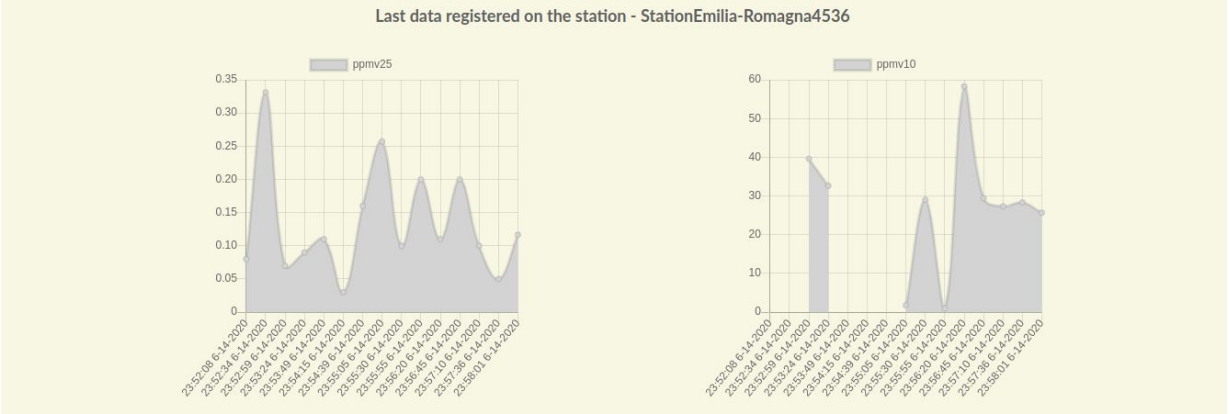

Figura 17 - Grafici ppmv10 e ppmv2.5.

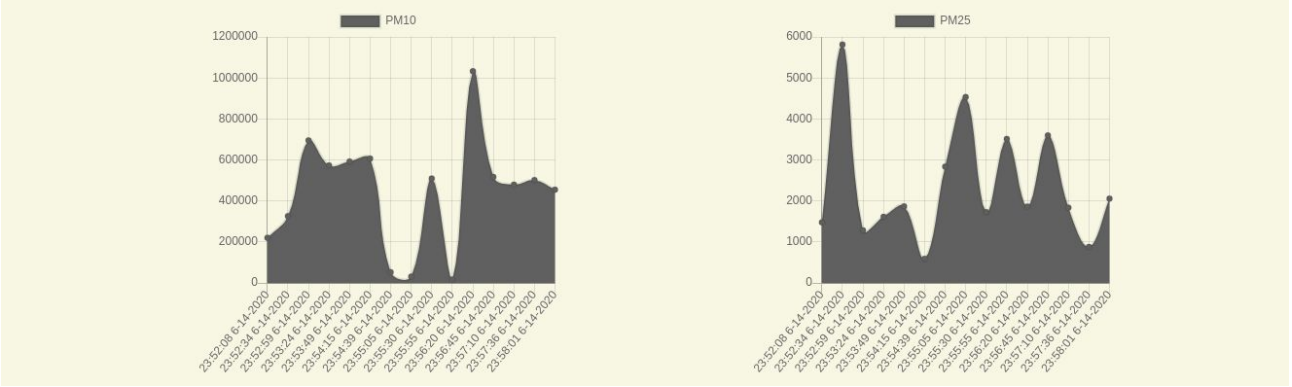

Figura 18 - Grafici PM10 e PM2.5.

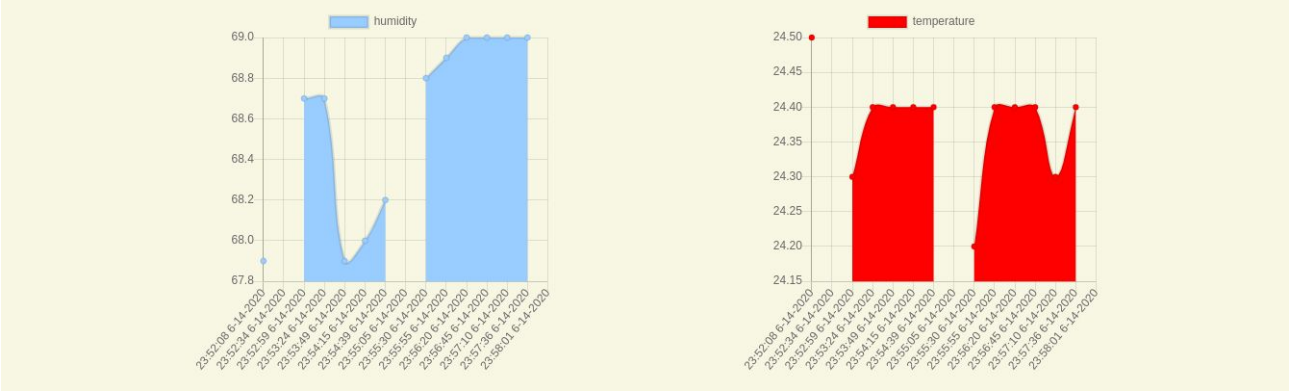

Figura 19 Grafici umidità e temperatura.

La seconda sezione, Figure 17-18-19, comprende i grafici riportanti gli ultimi 15 dati raccolti dalla stazione che vengono aggiornati in automatico quando si rilevano dei nuovi campioni sfruttando il meccanismo precedentemente illustrato.

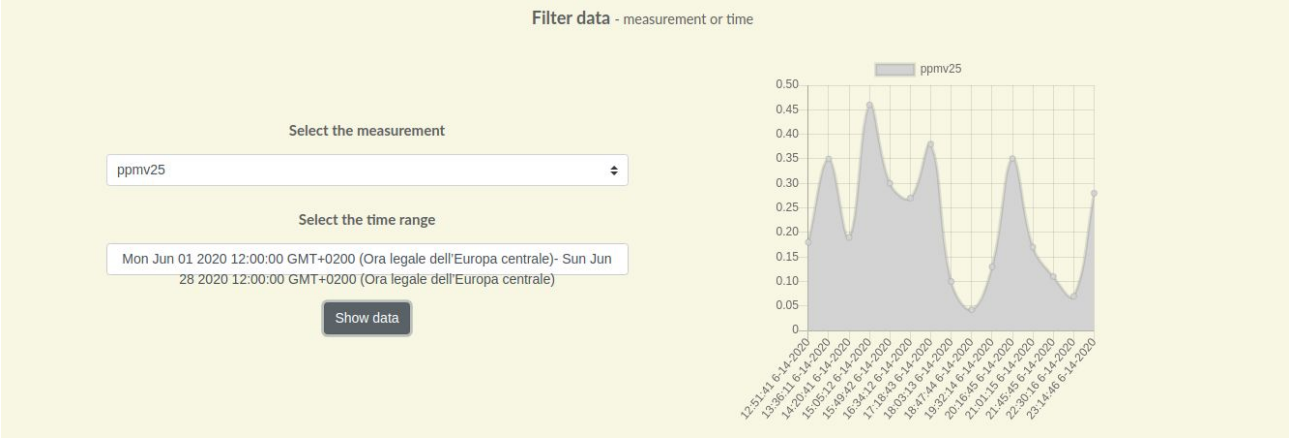

Figura 20 - Grafico con filtro dei dati.

Il terzo elemento presente, Figura 20, permette di filtrare i dati per tipologia e per data: una volta selezionati i parametri voluti e premuto il bottone “show data” il grafico si aggiorna con i campioni desiderati. Se il numero di dati risulta essere troppo elevato e quindi non rappresentabile in maniera efficace nel grafico, si effettua un sampling e si visualizzano solo 15 elementi.

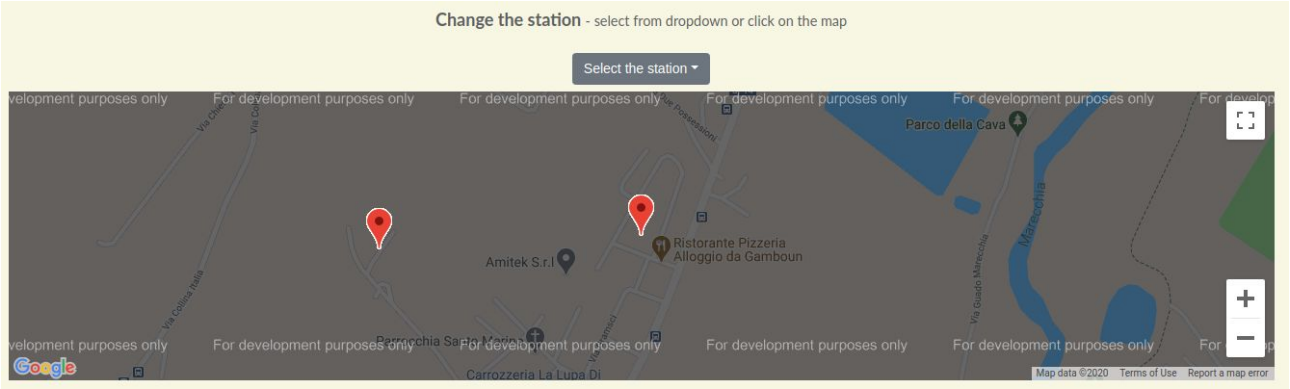

Figura 21 - Mappa terminale.

L’ultima sezione, Figura 21, riprende lo schema della pagina iniziale e permette all’utente di modificare la stazione scelta con le due modalità descritte. In questo caso la pagina non viene ricaricata ma vengono modificati solo i dati per farli coincidere con quelli voluti.

Dunque il funzionamento dell’applicazione può essere riassunto nello schema riportato in Figura 22.

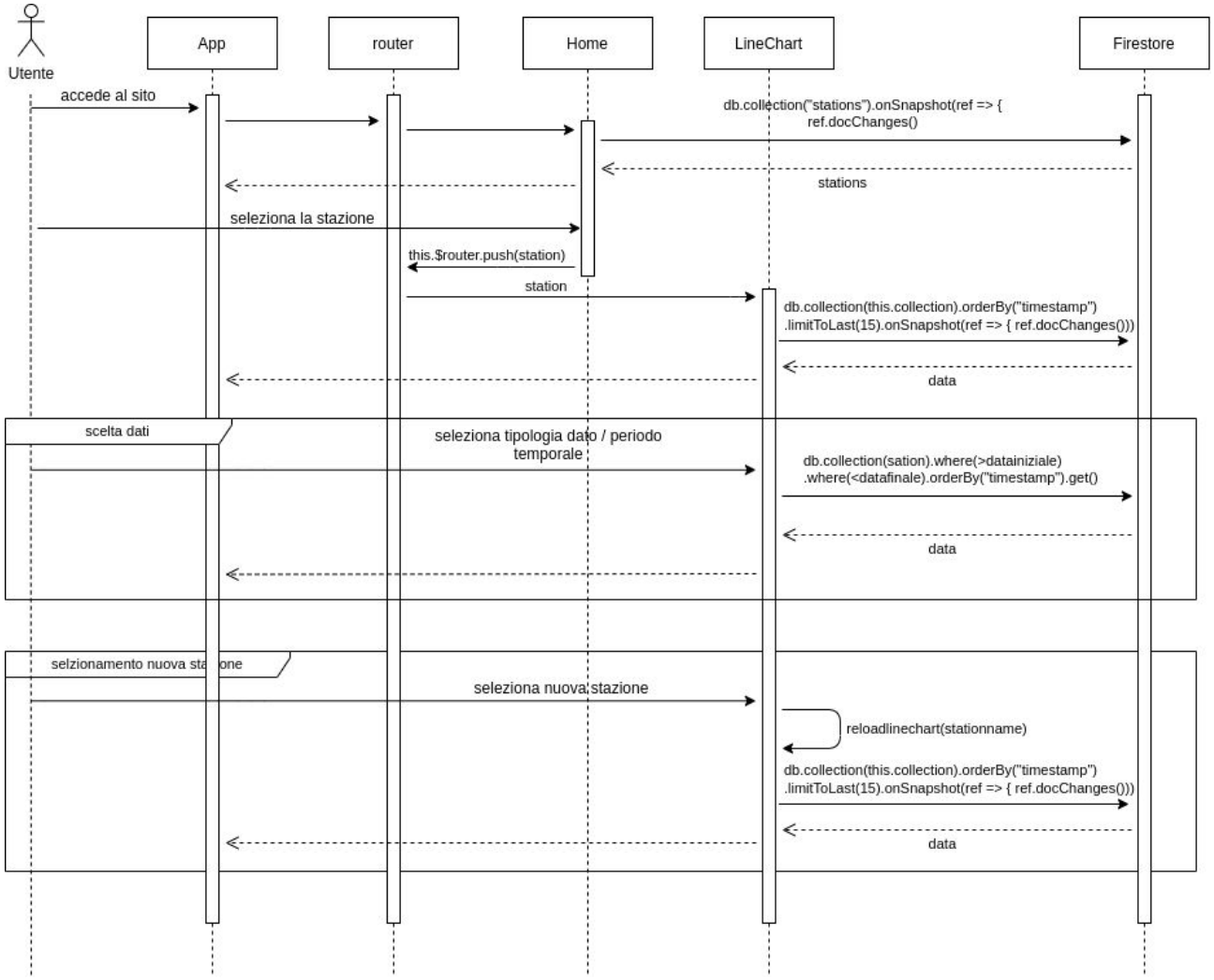

Figura 22 - Diagramma di sequenza applicazione web - Firestore .

Il codice implementato presente nell'archivio consegnato è suddiviso nelle seguenti cartelle:

- arduino: contenente tutto il codice relativo a Arduino Uno;
- raspberry: contenente tutto il codice relativo al Raspberry;
- console-web: rappresenta il codice del progetto Vue dell'applicazione web.

6 Testing e performance

I test sono stati effettuati grazie all'ausilio di un computer portatile. Di seguito si riportano le loro caratteristiche tecniche.

Computer portatile - ASUS f555l - 2015

Architecture	x86_64	Stepping	4
CPU op-mode(s)	32-bit, 64-bit	CPU MHz	1211.009
Byte Order	Little Endian	CPU max MHz	3000,0000
CPU min MHz	500,000	Address sizes	39 bits physical, 48 bits virtual
CPU(s)	4	BogoMIPS	4788.57
On-line CPU(s) list	0-3	Virtualization	VT-x
Thread(s) per core	2	L1d cache	64 KiB
Core(s) per socket	2	L1i cache	64 KiB
Socket(s)	1	L2 cache	512 KiB
NUMA node(s)	1	L3 cache	4 MiB
Vendor ID	GenuineIntel	NUMA node0 CPU(s)	0-3
CPU family	6	O.S.	Linux
Model	61	Model name	Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz

Tabella 1 - Specifiche ASUS f555l.

Inoltre sono stati utilizzati un Arduino e un Raspberry per comporre la stazione.

Raspberry Pi 3 - Model B

Architecture	armv71	CPU MHz	1200,0000
CPU op-mode(s)	32-bit, 64-bit	CPU max MHz	1200,0000
Byte Order	Little Endian	BogoMIPS	38.40
CPU min MHz	600,000	L1 cache	32 KiB
CPU(s)	4	L2 cache	512 KiB
On-line CPU(s) list	0-3	Core(s) per socket	4
Thread(s) per core	1	Socket(s)	1
Model	4	Model name	Armv7 Processor rev (v71)
O.S.	Raspbian		

Tabella 2 - Specifiche Raspberry.

Arduino Uno

Microcontroller	Microchip ATmega328 P	DC Current for 3.3V Pin	50 mA
Operating Voltage	5 Volts	Flash Memory	32 KB of which 0.5 KB used by bootloader
Input Voltage	7 to 20 Volts	SRAM	2 KB
Digital I/O Pins	14	EEPROM	1 KB
UART	1	Clock Speed	16 MHz
I2C	1	Lenght	68.6mm
SPPI	1	Widht	53.4mm
Analog Input pins	6	Weight	25g
DC Current per I/O Pin	20 mA		

Tabella 3 - Arduino.

Al fine di verificare il funzionamento del sistema al completo e partendo dai requisiti stilati in fase di progettazione, sono state effettuate le seguenti operazioni:

- controllare che il comportamento della stazione rispecchi i requisiti;
- testare la robustezza del sistema;
- valutare la reattività dell'interfaccia grafica.

Per quanto riguarda la stazione sono stati testati i seguenti aspetti:

- la registrazione della stazione nel database deve avvenire una sola volta;
- nella registrazione, eventuali errori di registrazione devono essere gestiti;
- i dati, in condizioni normali, devono essere trasmessi e salvati a Firestore;
- se ci sono problemi di connessione, i dati devono essere immagazzinati per poi essere rinviati quando la connessione viene ristabilita;
- la misurazione dei dati da parte dei sensori e la trasmissione dei dati deve avvenire ogni 30 secondi.

Il database, invece, deve passare i seguenti test:

- tempo di inserimento dei dati molto breve;
- tempo di esecuzione di una query ristretto.

Data la scelta di utilizzare Firestore come servizio, questi aspetti sono garantiti dall'architettura del servizio offerto da Google e quindi, assicurati ad eccezione di disservizi dell'azienda.

Dell'interfaccia grafica, essendo solo un esempio, si sono testate le seguenti funzionalità:

- deve essere possibile selezionare una stazione sia da mappa che da elenco;
- i grafici devono aggiornarsi in automatico quando vengono salvati nuovi dati su Firestore;
- la selezione dei dati da visualizzare deve essere effettuata in un tempo breve affinché la stazione sia reattiva.

Di seguito sono riportati i report dei test effettuati, Figure 23-24, sugli aspetti precedentemente illustrati.

id	Modulo	Scenario	Passi effettuati	Risultato aspettato	Risultato ottenuto	Stato
1.1	Stazione	Registrazione della stazione su Firebase	1. accensione	stazione aggiunta su Firestore	stazione aggiunta su Firestore	ok
1.2	Stazione	Registrazione in assenza di rete WiFi	1. accensione 2. spegnimento WiFi	stazione non si registra su Firestore	stazione non si registra su Firestore	ok
1.3	Stazione	Registrazione con problemi di connessione	1. accensione 2. spegnimento WiFi 3. riaccensione WiFi	stazione aggiunta su Firestore	stazione aggiunta su Firestore	ok
1.4	Stazione	Registrazione con problemi di Firestore	1. accensione 2. connessione Firestore impedita 3. ripristino connessione	stazione aggiunta su Firestore	stazione aggiunta su Firestore	ok
2.1	Stazione	Campionamento dei dati	1. modifica codice Rasberry per verificare l'arrivo dei dati 2. accensione della stazione	i dati dall'Arduino a Raspberry arrivano ogni 30 secondi	stampa dei dati ricevuti dall'Arduino ogni 30 secondi	ok
3.1	Stazione	Invio dati a Firestore	1. modifica codice per stampare la lista dei dati inviati a Firestore 2. accensione stazione	i dati sono salvati su Firestore	i dati sono salvati su Firestore	ok
3.2	Stazione	Invio dati a Firestore senza connessione	1. accensione macchina 2. attesa registrazione stazione 3. staccare connessione WiFi	i dati sono mantenuti in una lista nel Raspberry	la lista contiene i dati ricevuti	ok
3.3	Stazione	Invio dati a Firestore senza connessione	1. accensione macchina 2. attesa registrazione stazione 3. staccare connessione WiFi	Il numero di tentativi consecutivi è pari al numero di dati raccolti e non inviati	Il numero di tentativi consecutivi è pari al numero di dati raccolti e non inviati	ok

Figura 23 - Tabella test 1.

id	Modulo	Scenario	Passi effettuati	Risultato aspettato	Risultato ottenuto	Stato
3.3	Stazione	Invio dati a Firestore senza connessione e successivo ripristino WiFi	1. accensione macchina 2. attesa registrazione stazione 3. staccare connessione WiFi 4. ripristinare connessione WiFi	I dati sono salvati su Firestore una volta ripristinata la connessione	I dati sono salvati su Firestore una volta ripristinata la connessione	ok
4.1	Interfaccia	Scelta stazione da elenco	1. avvio server 2. apertura da browser della pagina iniziale 3. scelta di una stazione dall'elenco	Visualizzazione delle pagina con i dati relativi alla stazione scelta	Visualizzazione delle pagina con i dati relativi alla stazione scelta	ok
4.2	Interfaccia	Scelta stazione da mappa	1. avvio server 2. apertura da browser della pagina iniziale 3. scelta stazione da elenco	Visualizzazione delle pagina con i dati relativi alla stazione scelta	Visualizzazione delle pagina con i dati relativi alla stazione scelta	ok
4.3	Interfaccia	Filtro dei dati	1. avvio server 2. apertura da browser della pagina iniziale 3. scelta stazione da elenco/mappa 4. selezione del periodo e del tipo di dato da visualizzare	Caricamento di un grafico con solo i dati voluti	Caricamento di un grafico con solo i dati voluti	ok
4.5	Interfaccia	Filtro dei dati	1. avvio server 2. apertura da browser della pagina iniziale 3. scelta stazione da elenco/mappa 4. selezione del periodo / tipo dato (range 4 mesi)	Caricamento di un grafico con solo i dati in un tempo inferiore ai 5 secondi	Orario inizio operazione: 21:48:29 Orario fine operazione 21:48:31	ok

Figura 24 - Tabella test 2.

Di seguito si riportano gli screen realizzati duranti la prova dei test principali, Figure 25-26-27-28.

Figura 25 - Test 3.2.

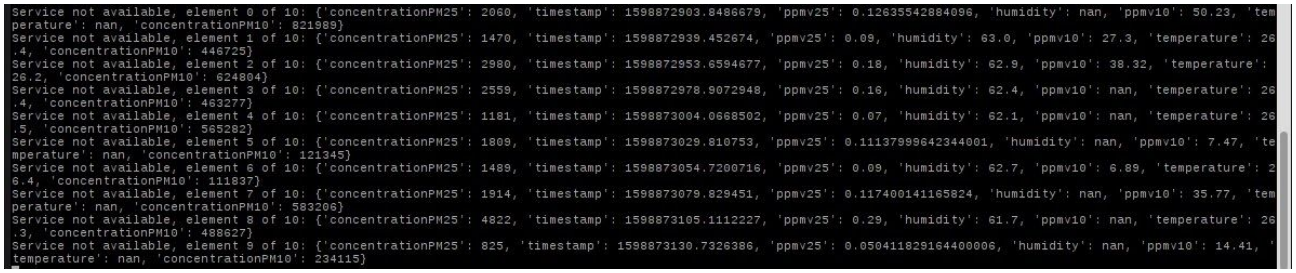

Figura 26 - Test 3.3.

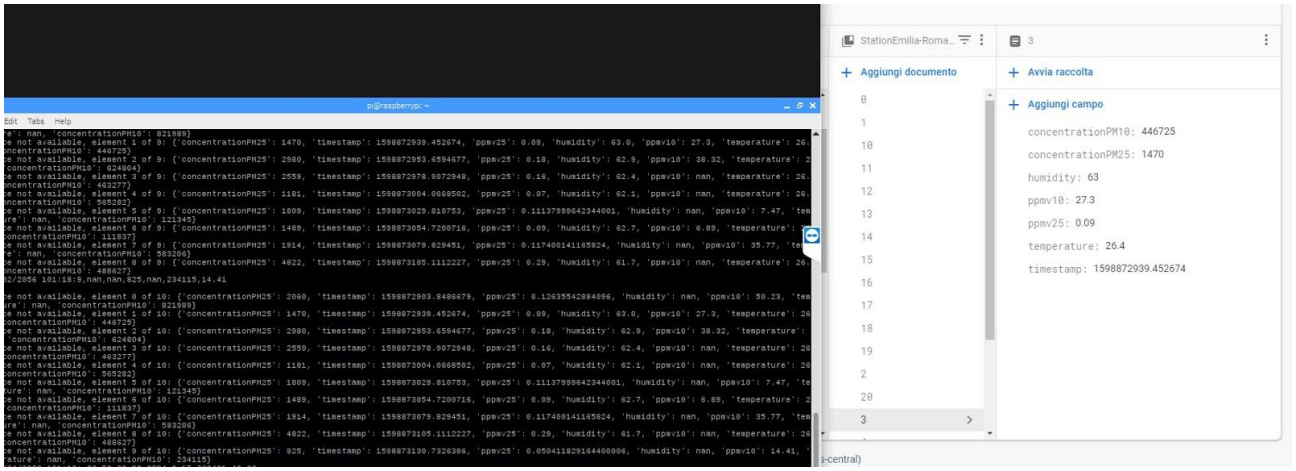

Figura 27 - Test 3.4.

```
Sun Aug 30 2020 21:48:29 GMT+0200
(Ora legale dell'Europa centrale)
Sun Aug 30 2020 21:48:31 GMT+0200
(Ora legale dell'Europa centrale)
```

Figure 28 - Test 4.5.

Attraverso i test effettuati si è potuto verificare che il sistema realizzato rispetti i requisiti che sono stati individuati durante la fase di progettazione. Il progetto dunque ha portato alla realizzazione di una stazione che è in grado di prelevare informazioni dall'esterno ed inviarle. In caso di errori o di mancanza di connessione, i dati, come voluto, sono mantenuti localmente fino a quando la stazione non riprende la connessione. Questa proprietà garantisce la robustezza desiderata dal sistema. Inoltre l'interfaccia, basata su Firebase, permette la visualizzazione dei dati e l'interrogazione del database in maniera efficiente, rimanendo reattiva per l'utente.

7 Analisi di deployment su larga scala

Il progetto realizzato può essere riprodotto in larga scala con alcuni accorgimenti. In particolare l'architettura progettata, con un disaccoppiamento completo fra stazione-database-interfaccia web, permette di poter aumentare facilmente il numero di stazioni di controllo, Figura 29.

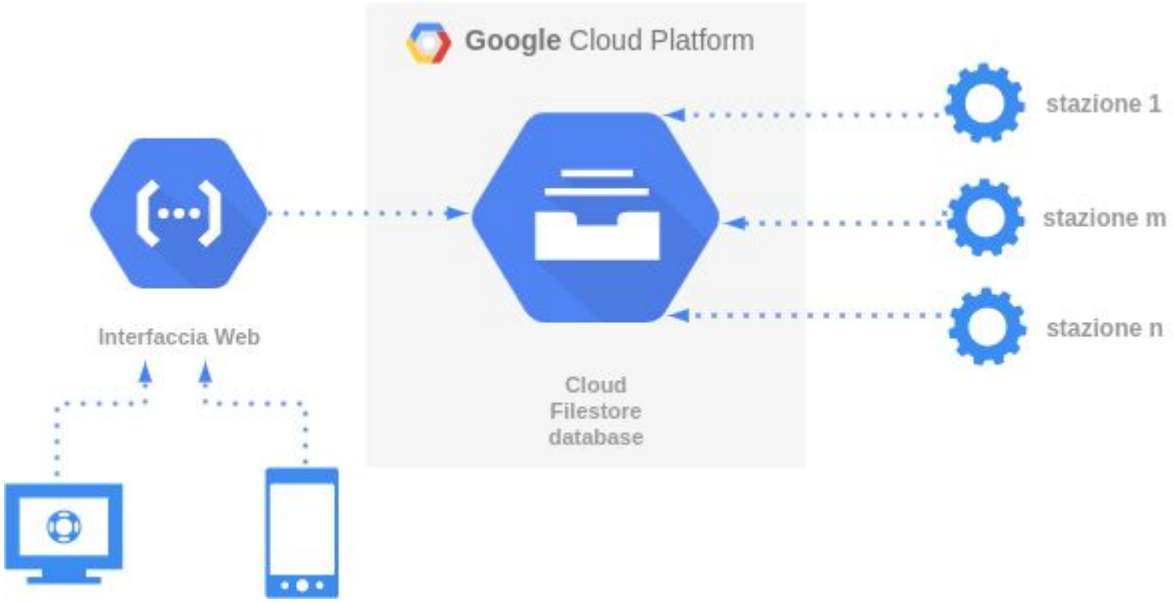

Figura 29 - Architettura su larga scala.

In questo caso, ogni stazione di controllo, con i suoi sensori, rappresenta un punto di raccolta di informazioni indipendente e autonomo dagli altri. L'unico vincolo che si pone è quello di avere accesso ad una rete WiFi per comunicare i dati raccolti. In una città, oggi giorno, avere a disposizione una connessione non rappresenta un particolare vincolo o uno ostacolo alla realizzazione del progetto. Se, d'altro canto, si vuole predisporre le stazioni in aree boschive o lontane da connessioni, si rende necessaria la modifica della stazione e la sostituzione dell'interfaccia WiFi con un supporto per una scheda di rete Sim.

Dal punto di vista dell'immagazzinamento dei dati non ci sono particolari vincoli di performance dato che si è deciso di utilizzare il database Firebase che, per sua natura, è progettato per avere carichi di lavori pesanti e gestire moli di dati elevate. Un database possiede alcune limitazioni di performance: per esempio il numero massimo di connessioni contemporanee al database è pari a 200000. Esso rappresenta il limite massimo di elementi che si possono connettere in contemporanea al database e, nel caso di questo progetto, rappresentato la somma delle connessioni per la visualizzazione dei dati nell'interfaccia grafica e per le stazioni di raccolta dati. In numero ipotetico di stazioni supportate è quindi molto elevato. Inoltre il database è in grado di dare 100000 risposte al secondo (12). In caso di una rete di sensori molto numerosa, è necessario effettuare una fase di studio delle performance anche con gli strumenti messi a disposizione da Google e, nel caso si verifichi che il carico di lavoro è eccessivo, scalare su più database. Dunque, aver scelto un database cloud come Firebase, permette di avere una struttura solida, resiliente e che è in grado di reggere carichi di lavoro intensi senza troppo lavoro da parte del gestore. Un

intervento si rende necessario se si creasse una rete di sensori sparsi sul territorio nazionale o su numero città.

Se dal punto di vista delle performance non ci sono grandi vincoli, nel caso di una implementazione in larga scala, si può rendere necessaria una ristrutturazione dei dati all'interno del database e la suddivisione delle stazioni per aree geografiche. In questa maniera ogni area geografica rappresenta una collezione diversa e la ricerca può diventare più semplice e esaustiva.

Modificando la struttura dei dati è anche possibile modificare l'interfaccia grafica, ora pensata per un numero di stazioni non numeroso, predisponendo una selezione iniziale dell'area geografica/città a cui si fa riferimento e che permette, con un grande numero di stazioni, di poter visualizzare meglio i dati.

Il server per l'interfaccia grafica, nel caso in cui ci sia un numero elevato di utenti che accedono ad essa, può essere implementato utilizzando il servizio di hosting messo a disposizione da Firebase (13) che permette di avere un'applicazione dalle alte performance con poco sforzo.

Inoltre, con un numero elevato di stazioni da controllare e disposte su un territorio geografico vasto, si rende necessaria verificare il loro stato. Una componente da sviluppare, quindi, sarebbe un sistema di allarmi che permette di controllare da remoto, lo stato della batteria delle stazioni, automatizzando i controlli che, con poche stazioni, possono essere svolte manualmente. Dall'interfaccia grafica quindi deve essere possibile vedere lo stato delle stazioni, se sono attive o meno, l'ultima volta che hanno mandato dei dati e, dopo un lasso di tempo, notificare che la stazione non invia più dati e che è da controllare. Questa soluzione permette di avere un completo controllo della situazione e rende il sistema resiliente ai guasti o alle perdite di connessione delle singole stazioni.

Per l'implementazione di questo monitoraggio si presuppone che ci sia un server adibito al controllo della frequenza dei dati ricevuti dalle applicazioni e segnali quelle che hanno dei problemi e determina l'introduzione di un server specializzato che analizzi i dati inviati e che comunichi nel caso, eventuali problemi, Figura 30.

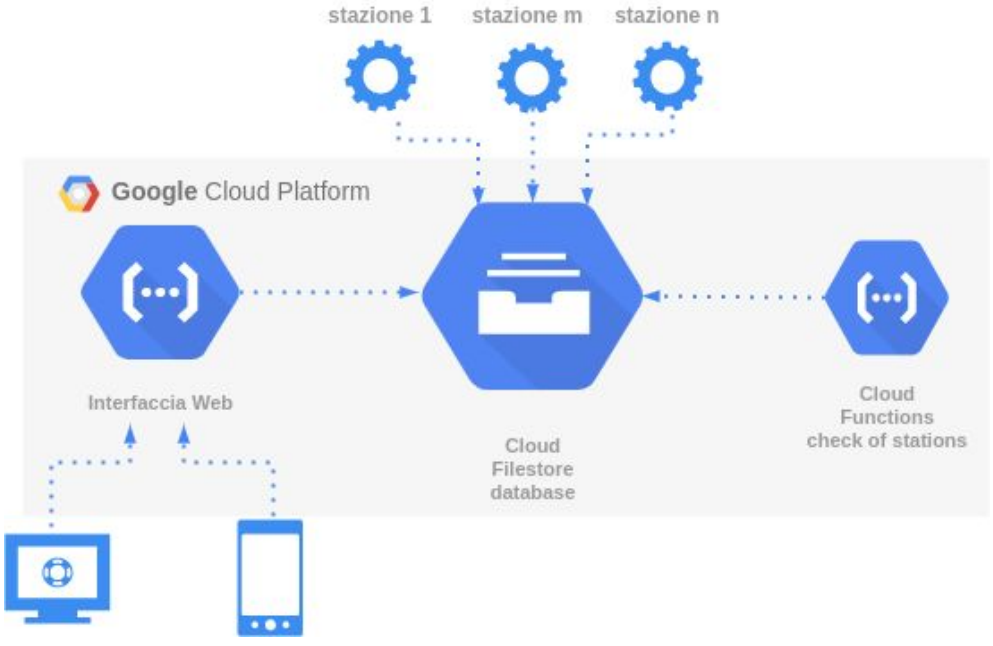

Figura 30 - Architettura su larga scala estesa.

Questo server deve essere in grado di controllare una vasta mole di dati e non deve essere soggetto a guasti o ad interruzioni di rete. Per ovviare a questi problemi tipici di un sistema distribuito, è possibile utilizzare un altro servizio di Firebase, in particolare le functions (14). Esse possono eseguire porzioni di codice sui dati di Firebase e notificare l'applicazione solo in caso di un effettivo problema. Essendo basate sul cloud computing, questo permette di eliminare tutti i problemi legati alla resilienza e ai guasti di rete che ci sarebbero stati con la costruzione di un server su una macchina fisica.

Per concludere questa possibile implementazione su larga scala, il costo principale di essa deriva dal numero di stazioni che si decide di produrre ma esse possono essere anche spostate e riutilizzate nel corso del tempo in aree differenti, abbattendo ulteriormente il costo. Quindi con questi strumenti è possibile coprire una vasta area con un budget ridotto e con ottimi risultati.

8 Piano di lavoro

La realizzazione del progetto, secondo un piano iniziale, doveva essere svolta nel primo trimestre del 2020. A causa della pandemia non è stato possibile ordinare alcun componente per molto tempo, quindi si è deciso di suddividere il lavoro nei mesi di Giugno e Agosto 2020, dovendo conciliare il tempo con gli impegni lavorativi dei componenti del gruppo.

Il lavoro si è diviso in tre sezioni principali:

- studio del problema: fase che ha compreso l'analisi teorica dell'inquinamento atmosferico, la ricerca delle tecnologie e dei sensori da utilizzare;
- la realizzazione della stazione fisica;
- l'implementazione del database su Firestore e dell'applicazione web di esempio.

Nella prima fase, che ha dato il via al progetto, è stato effettuato uno studio del problema dell'inquinamento e di quali fossero i principali parametri utilizzati per misurare la qualità dell'aria. Successivamente si è deciso quali fossero le tecnologie principali da poter sfruttare, tra cui i sensori da ordinare, tenendo conto di quelli che erano i dispositivi di cui già si era in possesso. Questa analisi è stata effettuata in sintonia da i componenti del gruppo.

Successivamente il lavoro, per motivi di praticità, è stato suddiviso fra i partecipanti:

- Simone Venturi si è occupato prevalentemente della realizzazione della stazione fisica, avendo a disposizione nella sua abitazioni i sensori, l'Arduino e il Raspberry. I suoi compiti principali sono stati:
  - assemblamento della stazione;
  - implementazione del task per PM, temperatura e umidità;
  - comunicazione tra Arduino e Raspberry;
  - comunicazione tra Raspberry e Firestore;
  - monitoraggio dei dati e correzione di eventuali errori.
- Alessia Ventani, di conseguenza si è concentrata sulla progettazione del sistema di salvataggio dei dati, della loro struttura su Firestore e della realizzazione dell'applicazione web. I suoi contributi principali sono:
  - realizzazione del task di Arduino per il GPS;
  - progettazione e implementazione del database Firestore;
  - implementazione dell'interfaccia grafica e il suo collegamento con il database.

Temporalmente i task sono stati suddivisi nei due mesi di Giugno e Agosto:

- nel primo mese si è fatta la progettazione e realizzata la stazione;
- ad Agosto si è implementata l'interfaccia web e testato il sistema nel complesso.

Il mese di Luglio è stato sfruttato per monitorare i dati raccolti dalla stazione e analizzare i cambiamenti che fossero necessari.

Di seguito viene riportato il grafico Gantt, Figura 31, con la suddivisione dei task nel tempo e una tabella riportante il numero di giorni/uomo per ognuno di essi, Tabella 4.

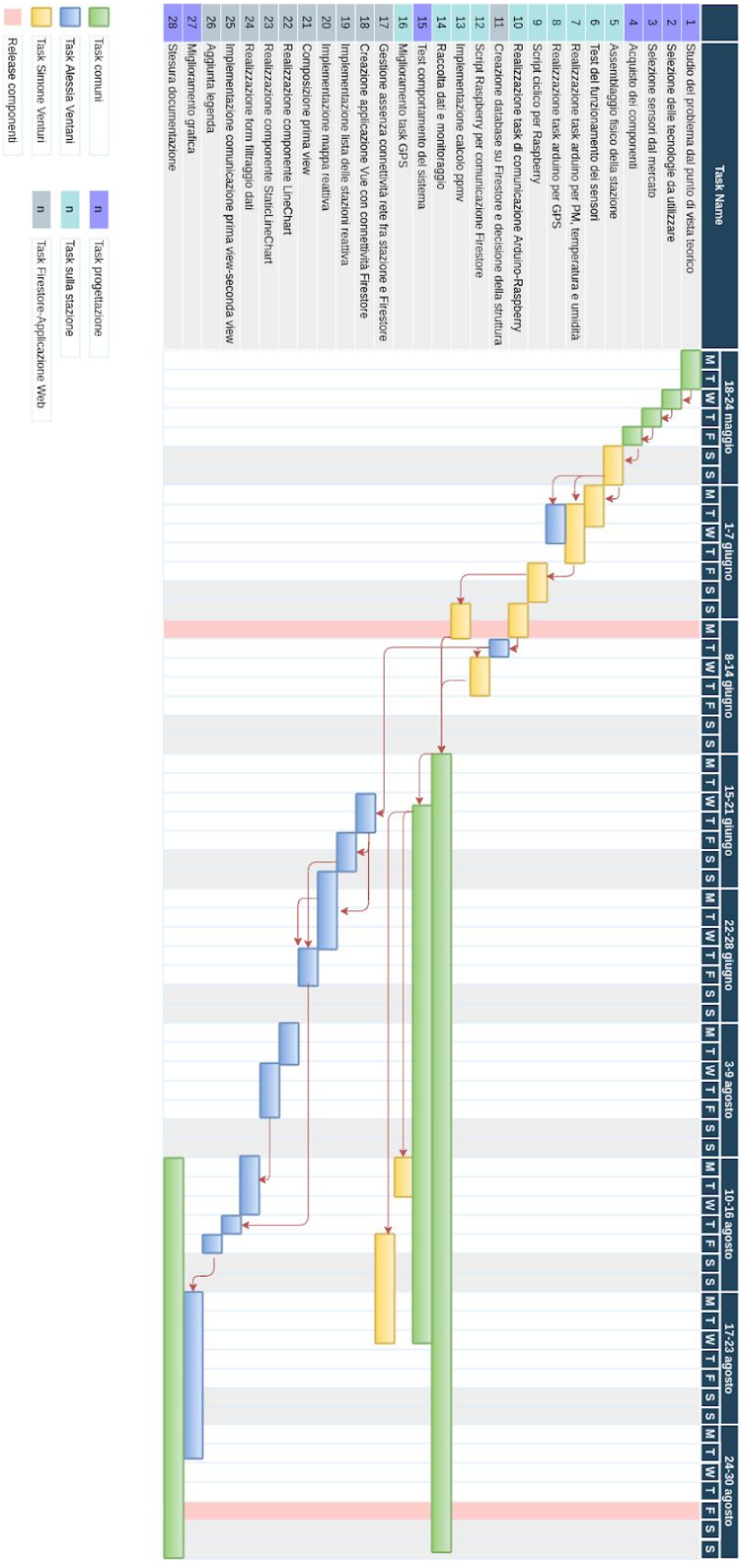

Figura 31 - Gantt pianificazione lavoro.

id	task	durata in giorni	ore/uomo
1	Studio del problema dal punto di vista teorico	2	6
2	Selezione delle tecnologie da utilizzare	1	4
3	Selezione sensori dal mercato	1	2
4	Acquisto dei componenti	1	1
5	Assemblaggio fisico della stazione	2	8
6	Test del funzionamento dei sensori	2	6
7	Realizzazione task arduino per PM, temperatura e umidità	3	10
8	Realizzazione task arduino per GPS	2	8
9	Script ciclico per Raspberry	2	4
10	Realizzazione task di comunicazione Arduino-Raspberry	2	8
11	Creazione database su Firestore e decisione della struttura	1	3
12	Script Raspberry per comunicazione Firestore	2	9
13	Implementazione calcolo ppmv	2	6
14	Raccolta dati e monitoraggio	agosto	-
15	Test comportamento del sistema	27	12
16	Miglioramento task GPS	2	6
17	Gestione assenza connettività rete fra stazione e Firestore	6	8
18	Creazione applicazione Vue con connettività Firestore	2	4
19	Implementazione lista delle stazioni reattiva	2	8
20	Implementazione mappa reattiva	4	10
21	Composizione prima view	2	3
22	Realizzazione componente LineChart	2	12
23	Realizzazione componente StaticLineChart	3	15
24	Realizzazione form filtraggio dati	3	9
25	Implementazione comunicazione prima view-seconda view	1	2
26	Aggiunta legenda	1	1
27	Miglioramento grafica	9	6
28	Stesura documentazione	continuo	continuo

Tabella 4 - Costo task ore/uomo.

9 Conclusioni

Alla fine del lavoro di implementazione è possibile affermare che l'obiettivo del progetto è stato pienamente raggiunto. Infatti, si è riusciti a sviluppare un intero sistema compreso di stazione per il rilevamento dei dati, server per il salvataggio di essi e in più una possibile implementazione di applicazione web che li mostra all'utente.
L'altro requisito fondamentale era la costruzione della stazione con un budget inferiore ai 70-80 euro: con i sensori utilizzati e i servizi gratuiti sfruttati il costo finale si è aggirato intorno ai 50-60 euro, molto esiguo rispetto ai corrispettivi commerciali.

Con la stazione implementata è stato possibile raccogliere diversi dati riguardanti la quantità di PM nel comune di Poggio Torriana. Nella cartella data dell'archivio, sono stati riportati due file con esempi dei dati raccolti. Il primo, stations.json, racchiude l'elenco delle informazioni delle stazioni che sono state utilizzate come prova: esse corrispondono ad una sola fisica, ma come da requisiti, ogni volta che viene riaccesa, viene registrata come una nuova all'interno del database. Il secondo file, example-data-station.json, contiene alcuni campionamenti effettuati da una singola stazione.
Dalle prove e dai dati raccolti si può concludere come la stazione implementata sia funzionante e che i valori raccolti nel comune di interesse, non superano, in media, le soglie consentite dalle norme vigenti indicando una buona qualità dell'aria (PM2.5<25 microgrammo/metrocubo e PM10<50 microgrammo/metrocubo).

Anche se l'architettura implementata è completa, il progetto può avere dei risvolti futuri. Per prima cosa si può rendere la stazione indipendente da una rete WiFi, inserendo un componente che possa leggere una sim-card, inoltre può essere sviluppato un sistema di allarmi per la gestione dello stato delle stazioni.

Con qualche ulteriore modifica poi, come già descritto, può essere allargato il progetto ed essere sviluppato su larga scala. Dato il costo esegui di una stazione e il possibile riuso dei suoi componenti, sarebbe relativamente semplice coinvolgere una parte della popolazione nella raccolta dati e così avere un database più completo e effettuare analisi più complesse.

Sicuramente una soluzione a basso costo come questa, può essere un aiuto fondamentale per il monitoraggio della qualità dell'aria che si respira.

10 Riferimenti bibliografici

1. “Measuring air pollution with low-cost sensors”, 10/06/2018 https://ec.europa.eu/environment/air/pdf/Brochure%20lower-cost%20sensors.pdf
2. “Laboratory Evaluation and Calibration of Three Low-Cost Particle Sensors for Particulate Matter Measurement”, Yang Wang, Jiayu Li, He Jing, Qiang Zhang, Jingkun Jiang & Pratim Biswas, 19/10/2015 , https://doi.org/10.1080/02786826.2015.1100710,
3. “Air Quality Monitoring Stations - Product Overview”, 21/10/2016, https://aqicn.org/products/monitoring-stations/
4. “Aeroqual AQY 1”, https://www.aeroqual.com/wp-content/uploads/AQY1-Spec-Sheet.pdf
5. “Arduino Air Quality Monitor with DSM501A Sensor © GPL3+”, https://create.arduino.cc/projecthub/mircemk/arduino-air-quality-monitor-with-dsm501a-sensor-b4f8fc
6. “Air Quality Monitoring With DSM501A With Nokia LCD”, https://www.instructables.com/id/Air-Quality-Monitoring-With-DSM501A-With-Nokia-LCD/
7. “Firebase di Google”, 21/06/2016, https://firebase.google.com/docs
8. “Firestore”, 8/10/2017, https://firebase.google.com/docs/firestore
9. “The Progressive JavaScript Framework” ,7/02/2014, https://vuejs.org/
10. “Vuefire”,2/04/2019 https://vuefire.vuejs.org/vuefire/getting-started.html#installation
11. “vue2-google-maps”, 4/10/2017, https://www.npmjs.com/package/vue2-google-maps
12. “Realtime Database Limits”, 27/05/2018, https://firebase.google.com/docs/database/usage/limits
13. “Fast and secure web hosting”,14/05/2017, https://firebase.google.com/products/hosting/?authuser=0
14. “Cloud Functions for Firebase” , 9/03/2017, https://firebase.google.com/docs/functions/?gclid=CjwKCAjwps75BRAcEiwAEiACMcZSi_-yjQNEHlYtOfFIq5u_xd1QHp6aXftWHshFpYOnFLfxEPeh6BoCzM0QAvD_BwE

11 Indice Immagini

- Figura 1 - Architettura sistema.
- Figura 2 - Schema tecnologie utilizzate.
- Figura 3 - Schema Arduino e sensori.
- Figura 4 - Diagramma di classe codice Arduino.
- Figura 5 - Diagramma a stato PMDetectorTask
- Figura 6 - Diagramma a stato CommunicationTask .
- Figura 7 - Diagramma navigazione applicazione web.
- Figura 8 - Diagramma di package applicazione web.
- Figura 8 - Diagramma flusso controllo invio dati Rasperry-Firestore.
- Figura 10 - Documento di una stazione.
- Figura 11 - Documento con i dati raccolti dai sensori.
- Figura 13 - Codice componente StaticLineChart.
- Figura 13 - Codice componente StaticLineChart.
- Figura 14 - Codice per la lista delle stazioni.
- Figura 15 - La mappa delle stazioni.
- Figura 16 - La legenda.
- Figura 17 - Grafici ppmv10 e ppmv2.5.
- Figura 18 - Grafici PM10 e PM2.5.
- Figura 19 Grafici umidità e temperatura.
- Figura 20 - Grafico con filtro dei dati.
- Figura 21 - Mappa terminale.
- Figura 22 - Diagramma di sequenza applicazione web - Firestore .
- Figura 23 - Tabella test 1.
- Figura 24 - Tabella test 2.
- Figura 25 - Test 3.2.
- Figura 26 - Test 3.3.
- Figura 27 - Test 3.4.
- Figure 28 - Test 4.5.
- Figura 29 - Architettura su larga scala.
- Figura 30 - Architettura su larga scala estesa.
- Figura 31 - Gantt pianificazione lavoro.

12. Indice tabelle

- Tabella 1 - Specifiche ASUS f555l.
- Tabella 2 - Specifiche Raspberry
- Tabella 3 - Arduino
- Tabella 4 - Costo task ore/uomo.