Relazione di progetto di "Smart City e Tecnologie Mobili"

Piattaforma di monitoraggio dell'inquinamento acustico

Numero del gruppo: 117

Componenti del gruppo: Fabio Muratori (fabio.muratori2@studio.unibo.it)

Indice

1 Introduzione

L'inquinamento acustico nelle città è un problema crescente che colpisce la qualità della vita delle persone e ha effetti negativi sulla salute umana. Può essere causato da diversi fattori, tra cui il traffico stradale, la costruzione di edifici, le attività industriali e lo svago notturno. L'esposizione prolungata al rumore può causare stress, disturbi del sonno, riduzione dell'udito e altri problemi di salute.
L'OMS (Organizzazione Mondiale della Sanità) ha affermato che il livello di rumore massimo deve essere di 65 dB per il giorno e di 55 dB per la notte e che il livello ottimale non dovrebbe oltrepassare i 45 dB.
In Italia, il problema dell'inquinamento acustico è stato affrontato attraverso la legislazione nazionale e regionale. Nel 1990 è stata emanata la Legge quadro sulla tutela del rumore[1], che stabilisce i limiti per l'esposizione al rumore e fornisce le linee guida per la prevenzione e il controllo dell'inquinamento acustico. Questa legge è stata successivamente integrata da altre leggi regionali e comunali, che ne hanno specificato la applicazione in diversi contesti territoriali. Per affrontare questo problema, molte città hanno introdotto regolamenti per limitare il rumore e promuovere la sostenibilità acustica. Questi possono includere limiti di volume per i concerti e le feste notturne, la costruzione di barriere acustiche lungo le strade e la promozione di modalità di trasporto più silenziose. In sintesi, l'Italia ha preso misure significative per affrontare l'inquinamento acustico nelle città, ma c'è ancora molto lavoro da fare per garantire un ambiente acusticamente salubre a tutti i cittadini.
L'obiettivo di questo progetto è quello di realizzare una piattaforma dedita al monitoraggio del livello acustico nelle città attraverso l'utilizzo di soluzioni *Internet of Things*. Il sistema dovrà essere dotato di apparecchi in grado di registrare i livelli di rumore nell'arco delle giornate e le misurazioni dovranno inoltre essere rese accessibili da remoto per consentire analisi e monitoraggio. Per consentire una migliore scalabilità del sistema verranno utilizzati:

- una piattaforma di messaggistica MQTT;
- un database per la raccolta dei dati;

[1] Legge 447/1995 "Legge quadro sull'inquinamento acustico" (recentemente modificata dal D. Lgs 42/2017)

- un computer SoC[2] a basso costo;
- una una piattaforma open-source dotata di un microcontrollore programmabile.

Il contributo di questo progetto si può riassumere in:

- ricerca ed utilizzo di approcci e piattaforme che offrano soluzioni cloud-based e scalabili;
- realizzazione di software e hardware prototipale per la misurazione e monitoraggio del livello acustico;
- integrazione dei componenti sopra citati per la realizzazione della piattaforma;
- realizzazione di un interfaccia web per la visualizzazione delle misurazioni e dei dispositivi installati, anche in tempo reale.

[2]Un System-on-Chip (SoC) è un circuito integrato che combina più componenti di un computer in un singolo chip con l'obiettivo di aumentare l'efficienza e ridurre le dimensioni e il costo dei dispositivi.

2 Stato dell'arte

I sistemi di monitoraggio acustico sono tecnologie utilizzate per rilevare, registrare e analizzare i suoni in vari ambienti. Questi sistemi possono essere utilizzati per una varietà di applicazioni come il monitoraggio della fauna selvatica, la gestione del rumore del traffico, il monitoraggio del rumore industriale, la sicurezza e la sorveglianza. In genere sono costituiti da microfoni, data-logger[3] e software per l'analisi e la visualizzazione dei dati. L'obiettivo dei sistemi di monitoraggio acustico è raccogliere e interpretare informazioni su livelli sonori, frequenze e modelli per comprendere e gestire meglio l'ambiente sonoro.
Questi data-logger non devono essere necessariamente infrastrutture o dispositivi dedicati: infatti anche semplici smartphone sono in grado di partecipare ad architetture per il monitoraggio. Ci si riferisce a questo tipo di approccio con il nome di *piattaforma collaborativa* o *mappa collaborativa*. La peculiarità di questi sistemi è la volontaria partecipazione della popolazione e la qualità dei dati è strettamente legata alla quantità di utenti che offrono i propri dispositivi ad un monitoraggio prolungato per una data zona geografica. Un esempio di questo tipo di infrastrutture è NoiseTube [4].
Inoltre, esistono sistemi in grado di stimare il livello di rumore acustico in base alla zona geografica, alla presenza di edifici, strade o fabbriche. In questo caso non sono utilizzati dispositivi fisici ma metodologie di stima euristica o per mezzo di intelligenza artificiale. Data l'enorme disponibilità di informazioni geografiche, sistemi di questo tipo tendenzialmente hanno un costo di implementazione ridotto a discapito della qualità dei risultati proposti. Un esempio di questo tipo di metodologia è OMS Global Noise Pollution Map [2].
Un altro esempio di sistema di monitoraggio acustico è SongMeter [6] un sistema progettato per lo studio e la ricerca della fauna selvatica ed è comunemente utilizzato per studiare le popolazioni di uccelli e pipistrelli. È costituito da un microfono compatto e resistente agli agenti atmosferici che registra i dati audio e può essere collocato in una varietà di ambienti, come foreste, zone umide o aree agricole, per raccogliere dati sull'attività acustica della fauna selvatica. Le registrazioni pos-

[3]Un data logger è un dispositivo che registra dati da sensori o altre fonti. Può acquisire, archiviare e trasmettere i dati registrati e viene utilizzato in una vasta gamma di applicazioni

sono quindi essere analizzate in un secondo momento per valutare l'impatto delle attività umane sulle popolazioni selvatiche.
Per quanto riguarda sistemi di monitoraggio attivo e prolungato nel tempo, sono stati individuati molteplici pubblicazioni ma non sono state trovate soluzioni già attive. Ad esempio [5] e [3] [1] propongono soluzioni interessanti per il monitoraggio della qualità dell'aria e dell'inquinamento acustico per mezzo di soluzioni IoT.

3 Analisi dei requisiti

In questa sezione si vogliono esplorare quelli che sono i requisiti, vincoli ed altre caratteristiche attese dal sistema di monitoraggio dell'inquinamento acustico proposto.

3.1 Scopo del progetto

Lo scopo del progetto è quello di raccogliere, analizzare e visualizzare in tempo reale i dati sul rumore ambientale, fornendo informazioni precise e affidabili sulla qualità acustica delle aree monitorate. Il progetto dovrà utilizzare sensori di rumore connessi ad una piattaforma predisposta all'utilizzo scalabile e cloud. Accedendo ad un portale web, le informazioni raccolte saranno utilizzate per fornire una visualizzazione aggiornata dei dati sulla qualità acustica. In sintesi, lo scopo di questo progetto per la misurazione e il monitoraggio dell'inquinamento acustico è quello di fornire una soluzione scalabile, efficiente e accessibile per la gestione dell'inquinamento acustico e per la tutela della qualità acustica delle aree monitorate.

3.1.1 Requisiti funzionali

I requisiti funzionali in un progetto informatico descrivono le funzionalità che il sistema deve fornire per soddisfare le esigenze degli utenti. Di seguito sono riassunti brevemente i requisiti funzionali della piattaforma che si vuole realizzare per il monitoraggio dell'inquinamento acustico:

- la piattaforma deve essere in grado di raccogliere i dati sul rumore ambientale da sensori di rumore installati potenzialmente in diverse aree;
- la piattaforma deve essere in grado di registrare e storicizzare i dati raccolti dai sensori;
- la piattaforma deve fornire una visualizzazione interattiva dei dati sul rumore ambientale ed accessibile attraverso un'interfaccia web o mobile.

3.1.2 Requisiti non funzionali

I requisiti non funzionali descrivono le proprietà del sistema che non riguardano le sue funzionalità specifiche, ma che sono comunque importanti per il suo corretto funzionamento. Di seguito sono riassunti brevemente i requisiti non funzionali della piattaforma che si vuole realizzare per il monitoraggio dell'inquinamento acustico:

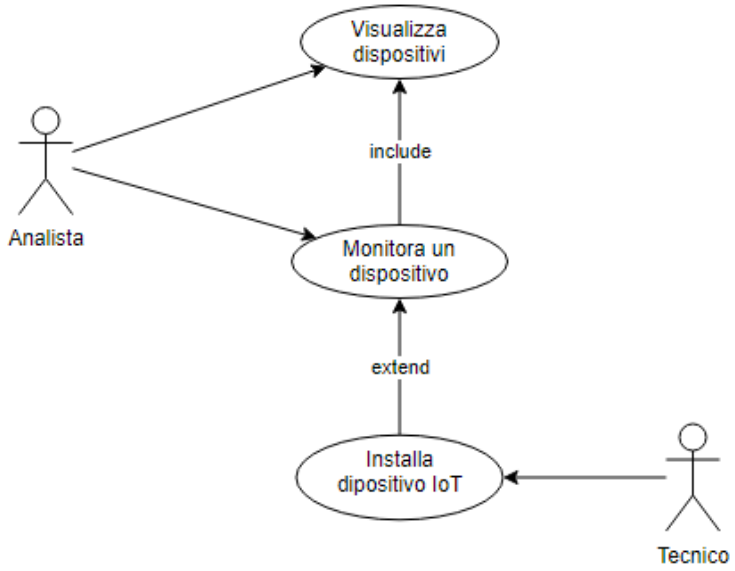
- scalabilità - la piattaforma deve essere in grado di gestire un aumento del carico di lavoro e dei dati senza compromettere le prestazioni;
- prestazioni - la piattaforma deve garantire una rapida raccolta, analisi e trasmissione dei dati sul rumore ambientale;
- usabilità - la piattaforma deve essere facile da usare per gli utenti, con un'interfaccia intuitiva e facile da comprendere;
- precisione e affidabilità dei dati - la piattaforma deve garantire la corretta trasmissione, l'integrità e l'affidabilità dei dati raccolti sul rumore ambientale;
- compatibilità con i dispositivi IoT - la piattaforma dovrebbe essere compatibile con una vasta gamma di dispositivi IoT e fornire un modo semplice per raccogliere dati da questi dispositivi;
- estensibilità e modularità - la piattaforma deve essere progettata per supportare modifiche e miglioramenti futuri.

3.1.3 Utenti e stakeholder

I potenziali utenti e gli stakeholder della piattaforma che si vuole realizzare per il monitoraggio dell'inquinamento acustico sono:

- enti di controllo, i quali potrebbero utilizzare i dati per monitorare e regolamentare l'inquinamento acustico e prendere decisioni informate;
- le aziende, le quali potrebbero utilizzare i dati per valutare l'impatto dei loro processi produttivi sull'inquinamento acustico e adottare soluzioni per ridurlo;

Figura 1: Diagramma UML dei casi d'uso con i principali attori della piattaforma di Iot per il monitoraggio dell'inquinamento acustico.



- i ricercatori, i quali potrebbero utilizzare i dati per lo studio dell'inquinamento acustico e per sviluppare soluzioni efficaci;
- i consulenti ambientali, i quali potrebbero utilizzare i dati per fornire consulenza alle comunità e alle aziende su come gestire l'inquinamento acustico.

In Figura 1 è riportato il diagramma UML dei casi d'uso che mostra i principali attori della piattaforma. Per gli scopi di questo progetto, la figura del Tecnico è stata messa in secondo piano ma ci si è concentrati sulle necessità degli utenti che svolgono il compito di monitoraggio dei livelli di inquinamento acustico.

3.1.4 Assunzioni, vincoli e dipendenze

Data la natura fortemente distribuita del sistema che si vuole realizzare e la necessità di avere dispositivi di misurazione dislocati geograficamente, si assume che tali apparecchi siano dotati di una connessione stabile ad internet.
Per quanto riguarda la componente geografica e di geo-localizzazione, non sono espressi vincoli sulla necessità di ottenere in tempo reale la posizione dei dispositivi. Questo dato dovrà essere codificato al momento dell'installazione in loco.

Inoltre, si vuole precisare che il progetto non è focalizzato sulla realizzazione di dispositivi in grado di misurare i livelli di rumore con elevata accuratezza ma alla realizzazione di un *proof of conpect*. Pertanto si esclude l'utilizzo di apparecchiatura professionale.
Data la complessità del sistema si è deciso di non perseguire approfonditamente tematiche di sicurezza e manutenibilità dei sistemi fisici. Anche l'aspetto grafico dei componenti che si desiderano realizzare è considerato come un obiettivo secondario.
Un altro punto di riflessione riguarda la granularità dei dati raccolti dai sistemi di misurazione. Seppur si ritiene di potenziale interesse la memorizzazione di dati campionati ad elevate frequenze, questa operazione potrebbe richiedere la gestione di una mole di dati elevata, un elevato traffico di rete e costi e consumi maggiori nelle attività di monitoraggio da parte dei dispositivi IoT. Pertanto, si ipotizza l'utilizzo di una frequenza di campionamento delle telemetrie non elevata e potenzialmente la messa in atto di semplici metodi per la storicizzazione dei dati o metodi di *edge computing*[4].

[4]L'edge computing è un paradigma di calcolo che si concentra sulla decentralizzazione dei processi di elaborazione dei dati, portandoli vicino ai punti di origine dei dati stessi. Ciò significa che i dati vengono elaborati e analizzati direttamente sui dispositivi che li generano senza la necessità di inviarli a un centro di elaborazione dati remoto.

4 Progettazione

Questo capitolo fornirà spunti sulla progettazione della piattaforma IoT cloud-based per il monitoraggio dell'inquinamento acustico, ed andrà ad evidenziare come le scelte fatte vadano a rispondere ai requisiti esposti nel Capitolo 3.
Data l'analisi dei requisiti, si evidenziano diversi fronti sullo sviluppo e progettazione del sistema desiderato. Di seguito sono introdotti tutti i macro-componenti del progetto, successivamente esplorati più in dettaglio:

- componenti per la misurazione e ricezione dei dati;
- base di dati per la raccolta e storicizzazione dei dati misurati;
- componenti per il monitoraggio e fornitura di interfacce web agli utenti.

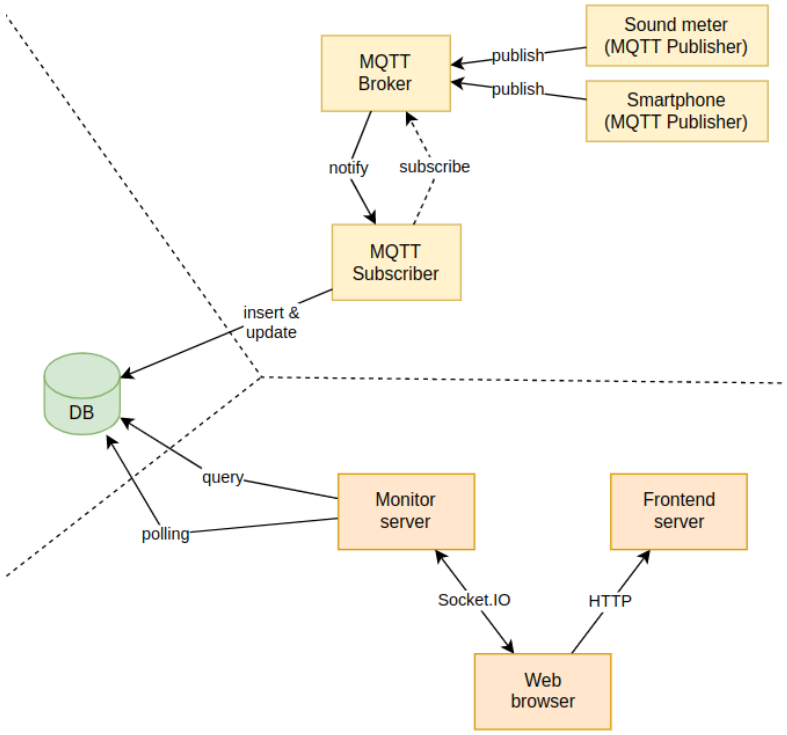
In questo modo e possibile separare nettamente le tematiche relative all'ambito IoT da ciò che riguarda la fruizione dei dati, favorendo l'adattabilità e la modularità della piattaforma. Inoltre il database diventa il punto centrale dell'architettura, considerando il fatto che ricopre il ruolo di anello di congiunzione tra le altre due parti del sistema.
La Figura 2 fornisce una rappresentazione grafica dei componenti e dei legami che intercorrono tra di essi sotto forma di canali di comunicazione.

4.1 Architettura MQTT

La comunicazione tra dispositivi in ambito IoT è fondamentale per l'invio e la gestione di dati raccolti da sensori e altri dispositivi connessi. Questi dati, noti come telemetrie, vengono trasmessi attraverso protocolli di comunicazione standard. Un'architettura cloud-based permette una gestione centralizzata e scalabile dei dati raccolti, offrendo maggiori opportunità per la visualizzazione, l'analisi e la gestione dei dati. I protocolli utilizzati generalmente in ambito IoT per la comunicazione e la trasmissione dei dati sono:

- MQTT (Message Queuing Telemetry Transport), un protocollo leggero di messaggistica pubblica/sottoscrizione per la connettività IoT;
- CoAP (Constrained Application Protocol), un protocollo web per la connettività IoT che supporta la comunicazione machine-to-machine (M2M);

Figura 2: Struttura di massima della piattaforma di monitoraggio dell'inquinamento acustico.



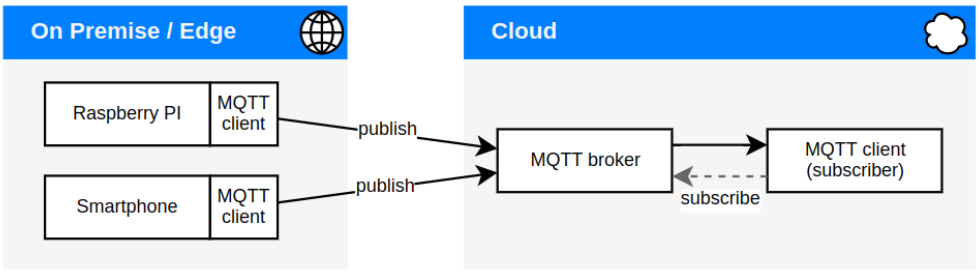
- HTTP (Hypertext Transfer Protocol), un protocollo di comunicazione standard utilizzato per la trasmissione di dati sul web;
- XMPP (Extensible Messaging and Presence Protocol), un protocollo di messaggistica che supporta la connettività M2M;
- AMQP (Advanced Message Queuing Protocol), un protocollo di messaggistica standard per la comunicazione tra applicazioni e dispositivi IoT;
- DDS (Data Distribution Service), un protocollo di comunicazione progettato per sistemi distribuiti e affidabili, utilizzato in ambito IoT.

Tra le varie alternative, si è scelto di utilizzare il protocollo MQTT uno dei sistemi più utilizzati in ambito IoT che offre alcuni vantaggi rispetto ad altri protocolli come ad esempio:

- ridotto consumo di banda - MQTT è un protocollo di messaggistica leggero, il che significa che utilizza poca larghezza di banda per la trasmissione dei dati rendendolo adatto in contesti con connessioni a banda limitata o costose;
- affidabilità - MQTT supporta la consegna affidabile dei messaggi, il che significa che i messaggi vengono inviati fino a quando non vengono consegnati o il sistema non viene interrotto aspetto che rende tale protocollo ideale per le applicazioni critiche che richiedono affidabilità elevata;
- semplicità - MQTT è un protocollo semplice e facile da implementare, il che lo rende ideale per le applicazioni IoT che richiedono una soluzione di comunicazione rapida e semplice;
- supporto multi-piattaforma - MQTT è supportato da molte piattaforme hardware e software diverse, il che lo rende facile da integrare in molte diverse soluzioni IoT.

In Figura 3 è proposta una classica architettura MQTT e sono evidenziati gli "attori" in gioco. Il protocollo prevede tre figure principali che partecipano alla comunicazione: il Subscriber, il Broker e il Publisher. Il Subscriber, talvolta chiamato semplicemente client, è il dispositivo che desidera sottoscrivere informazioni su un determinato argomento. Il Broker è il server che gestisce le connessioni e

Figura 3: Architettura di una generica piattaforma MQTT.



le pubblicazioni tra i client. Il Publisher è il client che invia i messaggi su un determinato argomento. Questi messaggi vengono quindi inviati a tutti i client che hanno sottoscritto quell'argomento, attraverso il Broker.
MQTT offre inoltre un importante strumento per definire la qualità delle comunicazioni di una sessione. Il meccanismo QoS (Quality of Service) in MQTT è utilizzato per garantire la qualità del servizio nella trasmissione dei messaggi. Esistono tre livelli di QoS in MQTT:

- QoS 0 (Livello di servizio 0) - il messaggio viene inviato al più una volta, senza conferma di consegna e se il messaggio non viene consegnato, non verrà ritentata la consegna;
- QoS 1 (Livello di servizio 1) - il messaggio viene inviato almeno una volta, con conferma di consegna ed in caso di errore il messaggio viene ritentato fino a quando non viene consegnato eventualmente anche più volte;
- QoS 2 (Livello di servizio 2) - il messaggio viene inviato ed è garantita la consegna senza duplicati.

Definire il livello di QoS in base alle esigenze dell'applicazione è importante perché ogni livello offre una diversa garanzia di qualità del servizio nella trasmissione dei messaggi. Ad esempio, se l'applicazione richiede una consegna affidabile dei messaggi, si può scegliere il livello di QoS 2 che garantisce che il messaggio sia consegnato correttamente e che non venga perso. D'altro canto, se l'applicazione non richiede una consegna affidabile dei messaggi ma richiede una maggiore velocità di trasmissione, si può scegliere il livello di QoS 0. Inoltre, scegliere il giusto livello di QoS influisce sulle risorse utilizzate dalle varie piattaforme, come la larghezza di banda e la memoria. Nella pratica, in base ai requisiti:

- QoS 0 è utilizzato se non è fondamentale ricevere tutti i messaggi ma è ammissibile perderne qualcuno;
- QoS 1 è utilizzato se è essenziale ricevere tutti i messaggi e l'applicazione è in grado di gestire messaggi duplicati;
- QoS 2 è utilizzato quando è essenziale ricevere tutti i messaggi una ed una sola volta.

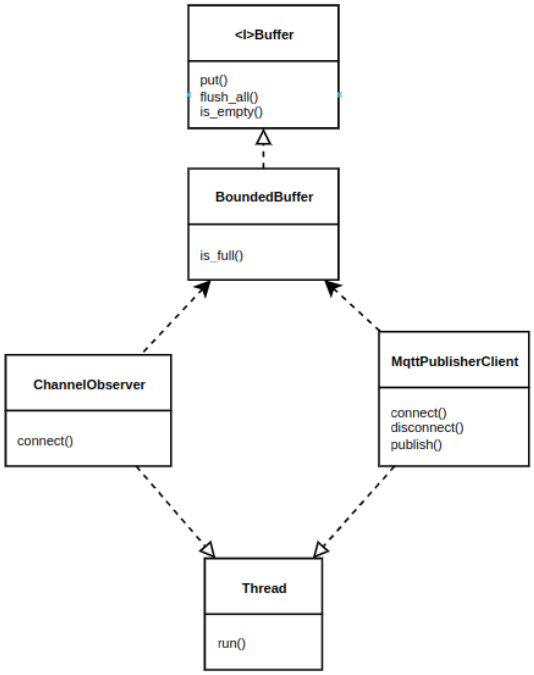
In relazione alle esigenze della piattaforma di monitoraggio che si intende realizzare, si ritiene che il livello di qualità QoS 2 sia il più senza escludere tuttavia il livello QoS 1 qualora le prestazioni raggiungano un livello critico.
Oltre alla metodologia per la trasmissione dei dati è importante soffermarsi anche sulle piattaforme "*on the edge*" utilizzate per la realizzazione del client MQTT dedito alla pubblicazione dei dati.

4.1.1 MQTT Publisher

In un'architettura MQTT, il Publisher è il componente che invia messaggi al Broker MQTT. Il compito principale del Publisher è quello di pubblicare messaggi sul Broker MQTT in modo che possano essere ricevuti dai componenti Subscriber. Il Publisher invia i messaggi su un argomento specifico, una stringa utilizzata per identificare la categoria di messaggi che il Publisher desidera pubblicare. Questo argomento è utilizzato dal Broker per indirizzare i messaggi ai Subscriber che hanno espresso interesse per quell'argomento effettuando una sottoscrizione esplicita. Il Publisher può anche specificare il livello di qualità del servizio (QoS) per ogni messaggio che invia.
Per quanto riguarda la logica del Publisher dell'architettura MQTT descritta in Figura 3, è possibile vedere in Figura 4 il diagramma UML delle classi. Questa soluzione proposta fa utilizzo di strutture e meccanismi di programmazione concorrente come ad esempio i *monitor*[5] visto che il software in esecuzione sulla

[5]Un monitor è un meccanismo di sincronizzazione utilizzato per garantire la sicurezza dei dati condivisi tra diversi thread di un'applicazione.

Figura 4: Diagramma delle classi UML del Publisher nell'architettura MQTT.



piattaforme hardware dovrà gestire in maniera distinta due tipi di meccanismi: la ricezione di telemetrie e l'invio delle stesse tramite un canale TCP[6].
Questa architettura è stata volutamente definita il più possibile generica in modo da facilitarne l'adozione da parte di piattaforme eterogenee.

4.1.2 MQTT Subscriber

Il compito di questo elemento è quello di ricevere le telemetrie tramite l'architettura Publish-Subscribe implementata dalla piattaforma MQTT. L'obiettivo è quello di convertire ogni messaggio ricevuto in operazioni di inserimento dati all'interno di un database in modo tale da garantire la persistenza delle informazioni che verranno in maniera asincrona accedute da altri elementi della piattaforma.
Questo è anche il punto ottimale in cui applicare operazioni di monitoraggio dei dati ricevuti dai vari sensori. Una potenziale implementazione delle attività di

[6]TCP (Transmission Control Protocol) è un protocollo di livello di trasporto di rete utilizzato per trasmettere dati in maniera affidabile tra due host su una rete ed è alla base di molte implementazioni di piattaforme MQTT.

monitoraggio (non realizzate in questo elaborato di progetto) potrebbe essere la ricerca di livelli allarmanti di rumore tra i flussi di dati ricevuti ed il salvataggio all'interno di report automatici o particolari collezioni del database.

4.2 Gestione dei dati

I sistemi di gestione dei dati (DBMS) sono utilizzati per organizzare, archiviare e recuperare informazioni. Esistono due tipi principali di soluzioni per la gestione dei dati: basate su SQL (Structured Query Language) e basate su No-SQL.
Soluzioni SQL sono basate su un modello relazionale di dati dove essi sono organizzati in tabelle con una struttura rigida e predefinita, supportano una gestione completa di vincoli e transazioni. I vantaggi di queste soluzioni includono una maggiore velocità e affidabilità nell'esecuzione delle query, una migliore gestione delle integrità dei dati e una maggiore facilità di utilizzo per gli sviluppatori che già conoscono SQL. Esempi di sistemi basati su SQL includono MySQL, PostgreSQL e Microsoft SQL Server.
Soluzioni NoSQL (Not Only SQL) invece, sono basate su un modello di dati non relazionale dove i dati sono organizzati in documenti composti di chiavi e valori. Queste soluzioni sono più flessibili e scalabili rispetto alle soluzioni SQL, poiché non richiedono una struttura rigida dei dati. Inoltre, sono progettate per gestire grandi quantità di dati e fornire una scalabilità orizzontale, che consente di gestire una crescita del carico di lavoro senza interruzioni. Un esempio di sistemi basati su NoSQL è MongoDB.
Considerando la natura del sistema che si vuole realizzare e considerando il fatto che molte delle funzionalità definite richiedono di ragionare sempre a livello di dispositivo, è stato ritenuto naturale adottare un approccio non relazionale nell'organizzazione dei dati. L'alternativa basata su un modello relazionale infatti potrebbe comportare l'esecuzione di operazioni di join costose tra un ipotetica tabella che racchiude le misurazioni (potenzialmente di grandi dimensioni) ed un'altra contenente informazioni sui dispositivi.
Di seguito è riportata la struttura di un documento relativo ad una particolare installazione di un dispositivo:

```
{
    _id: string,
```

```
    gps_lat: Float,
    gps_lng: Float,
    device_name: String,
    last_timestamp: Date,
    active_since: Date,
    data: [
        {
            datetime: Date,
            value: Integer,
        },
    ]
}
```

Ogni documento contiene informazioni sul dispositivo installato come:

- le coordinate gps, chiavi `gps_lat` e `gps_lng` ;
- nome univoco del dispositivo, chiave `device_name`;
- istante temporale di ultima ricezione di dati, chiave `last_timestamp`;
- istante temporale di inizio attività, chiave `active_since`;
- i dati relativi alle misurazioni dei livelli di rumore, chiave `data`;

Il dato memorizzato nel campo `value` è il livello in decibel di rumore registrato dai dispositivi fisici sotto forma di valore intero invece che decimale in relazione al fatto che le esigenze di progetto non richiedono un livello di dettaglio così elevato.

4.2.1 Processamento dei dati

In fase di progettazione è stato stabilito che il livello adeguato per la memorizzazione di dati è di una misurazione al minuto. Per limitare ulteriormente la mole di dati generati e da gestire è stato ideato un semplice approccio di compressione delle sequenze di livelli di rumore: se tra due istanti temporali successivi è misurata una variazione nulla, è possibile omettere il secondo valore dalla traccia trasmessa. Questo approccio necessita tuttavia dell'aggiunta di un campo `datetime` che identifichi il primo istante temporale di misurazione. In questo modo è possibile

mantenere solamente informazioni sulla variazione di rumore nel tempo ed omettere potenziali periodi prolungati di rumore costante, ed anche permettere la gestione di potenziali malfunzionamenti o periodi di assenza di misurazioni. Infine, sono stati espressi vincoli riguardo la frequenza massima con cui ricevere dati ma non è stato imposto alcun vincolo su come il livello di rumore debba essere calcolato all'interno dell'intervallo di tempo di 1 minuto. Ogni dispositivo fisico campiona dati con una frequenza potenzialmente maggiore, ad esempio ogni 100 millisecondi, permettendo quindi di percepire anche variazioni brevi ma intense del rumore. Il livello maggiore di variazione dall'ultimo valore inviato è di conseguenza quello candidato alla trasmissione allo scadere del secondo successivo.
L'esempio in Figura 5 mostra graficamente il ragionamento descritto se si considera una frequenza di campionamento di 100 millisecondi ed una frequenza di memorizzazione di 1 secondo:

1. data la traccia in figura sono individuati gli intervalli ogni secondo;
2. per ogni intervallo è identificato il valore massimo registrato (in figura evidenziati in rosso);
3. è effettuata la discretizzazione dei valori all'intero più vicino;
4. se tra intervalli successivi è identificato lo stesso valore, si tiene traccia solamente del primo.

Nell'esempio in figura sono individuati i seguenti livelli al punto 3:

- 42 db nel primo intervallo;
- 43 db nel secondo intervallo;
- 43 db nel terzo intervallo;
- 41 db nel quarto intervallo;

Notare come in Figura 6 il valore identificato nel terzo intervallo sia colorato di grigio scuro visto che non verrà trasmesso poiché identico al valore nell'intervallo precedente.

Figura 5: Esempio di elaborazione dei livelli di rumore registrati.

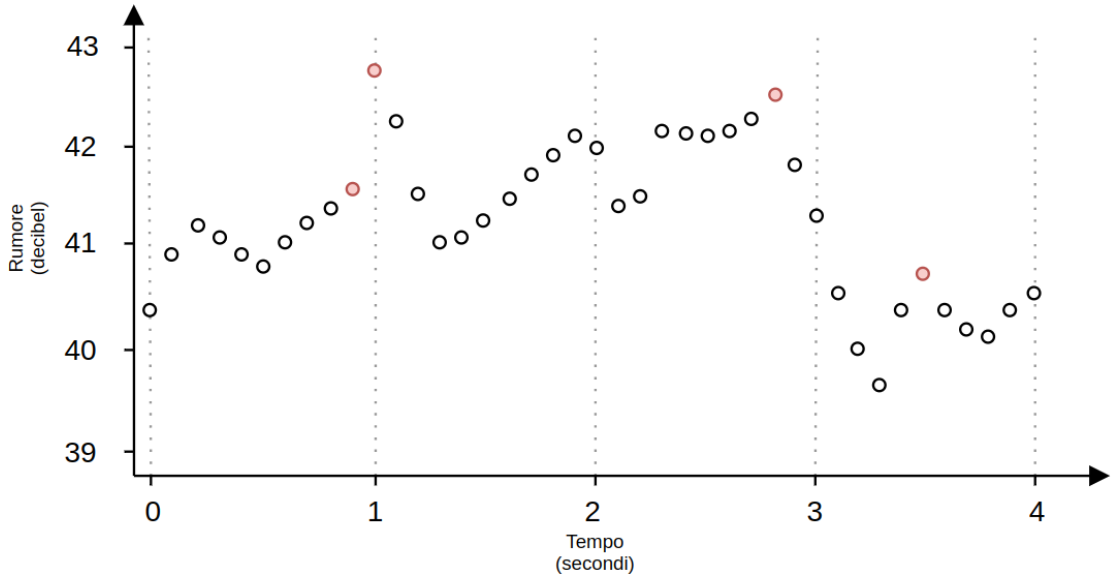
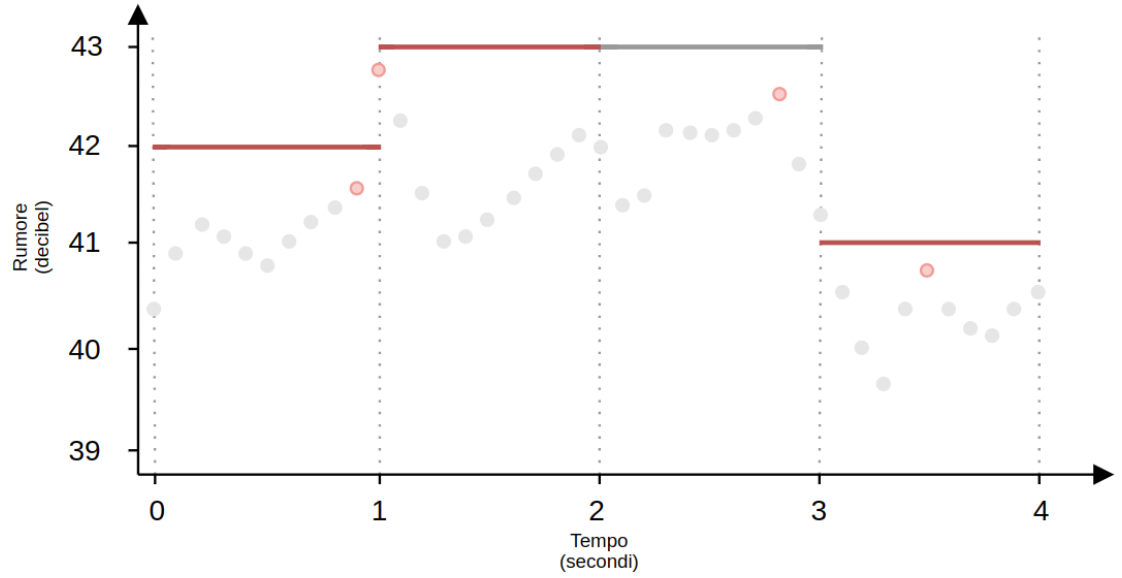


Figura 6: Risultato dell'esempio di elaborazione dei livelli di rumore registrati.



In questo elaborato non sono state trattate tematiche di qualità dei dati registrati ne tanto meno della rimozione di valori *outlier*[7].

4.3 Monitoraggio dei dati

Per quanto riguarda le funzionalità offerte agli utenti finali della piattaforma, è stato deciso di applicare una suddivisione delle componenti necessarie in due distinte strutture:

- un server che si occupa di fornire l'applicazione web accessibile da un qualsiasi browser;
- un server che implementa le logiche di monitoraggio dei dispositivi e dei dati in tempo reale.

Questa soluzione nasce dal fatto che le due funzionalità hanno un volume dati fondamentalmente differente e potenzialmente il server che effettua monitoraggio implementa delle logiche applicative *data-intensive*[8] e deve mantenere un livello di scalabilità superiore. Inoltre, questo approccio favorisce aspetti di qualità come la modularità, l'efficienza e la manutenibilità della piattaforma.
Per quanto riguarda il monitoraggio dei dati presenti sul database, sono state individuate alcune alternative: meccanismo dei trigger, polling periodico ed architettura ad eventi.
Un trigger è un meccanismo che esegue automaticamente azioni specifiche, come l'esecuzione di una procedura, in risposta a un'operazione di inserimento, aggiornamento o eliminazione di dati nel database. Questo approccio consente di implementare applicazioni completamente reattive ma l'implementazione reale può differire notevolmente in base al DBMS utilizzato.
Il polling è un meccanismo comunemente utilizzato per monitorare periodicamente una determinata risorsa. Questo approccio risulta essere di facile implementazione ma può comportare un traffico aggiuntivo. Un'architettura ad eventi consente

[7]Dei valori outlier sono valori anomali che si trovano al di fuori della distribuzione normale dei dati.

[8]Le applicazioni data-intensive sono quei sistemi software che si concentrano sulla gestione, l'elaborazione e l'analisi di grandi quantità di dati. Questi sistemi spesso richiedono la capacità di elaborare enormi volumi di dati in tempo reale.

l'esecuzione di azioni in seguito ad eventi. Questo approccio consente di eseguire operazioni solamente quando necessario ma spesso richiede la creazione di strutture apposite (come delle code di eventi) che possono complicare ulteriormente l'architettura della piattaforma. Tra le tre alternative si è scelto di utilizzare un approccio basato su polling periodico

- l'aggiunta di nuove misurazioni per un dispositivo monitorato avviene una volta al minuto, aspetto che rende l'attività di polling poco frequente da parte di un singolo utente;
- predisposizione alla scalabilità orizzontale con la creazione di più server dediti al monitoraggio se necessario;
- facilità di implementazione e ottimizzazione;
- il polling sarebbe gestito dal server e non dai client, aspetto che riduce i rischi dovuti alla sicurezza e al sovraccarico.

5 Implementazione

In questa sezione, saranno esposte più nel dettaglio le implementazione delle parti salienti della piattaforma IoT per il monitoraggio dei livelli di rumore, compresi i componenti hardware e software necessari, l'architettura del sistema e le tecnologie utilizzate per l'elaborazione e la trasmissione dei dati.
In Figura 7 è possibile osservare la struttura della piattaforma di monitoraggio e le tecnologie impiegate nelle varie aree di interesse. Di seguito sono esplorate tre differenti ambiti: architettura MQTT per la gestione delle telemetrie, gestione dei dati e implementazione delle funzionalità per gli utenti finali.

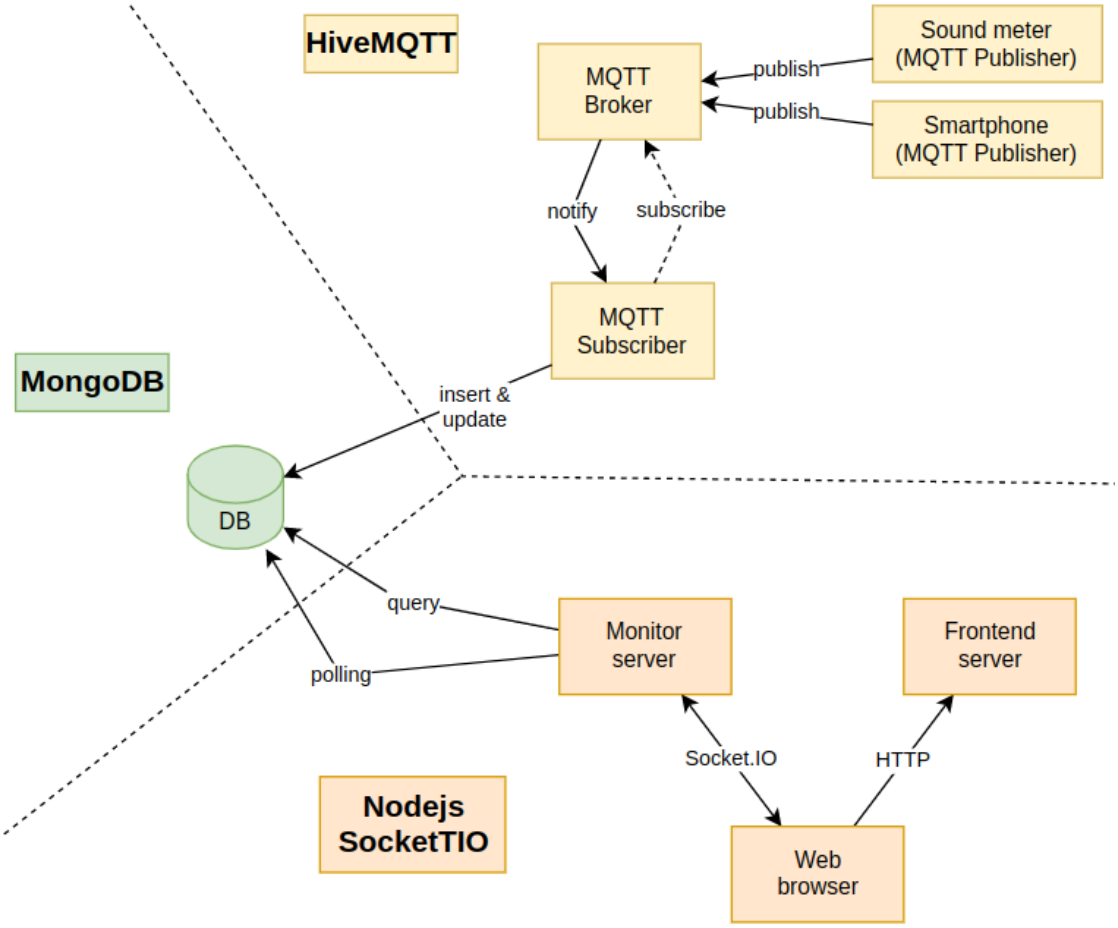
5.1 Piattaforma MQTT

MQTT (Message Queuing Telemetry Transport) è un protocollo di messaggistica leggero utilizzato per l'Internet of Things (IoT) e la telemetria. Lo scopo di una piattaforma MQTT è fornire un modo efficiente e affidabile per la comunicazione tra i dispositivi IoT, gli agenti di raccolta dati e i server di backend. Questa piattaforma consente di pubblicare e sottoscrivere messaggi su canali specifici, con opzioni di qualità del servizio (QoS) per garantire la consegna affidabile dei messaggi. La piattaforma MQTT è ottimizzata per la connettività a basso consumo energetico e adatta alle reti con larghezza di banda limitata, rendendola una soluzione ideale per la comunicazione tra dispositivi IoT.
I principali framework MQTT disponibili sono:

- Mosquitto - un server MQTT open-source sviluppato e fornito da Eclipse;
- HiveMQ - una piattaforma MQTT enterprise-level con funzionalità avanzate di gestione e monitoraggio;
- AWS IoT Core - un servizio di IoT di Amazon Web Services che supporta MQTT.

Tra le opzioni proposte, le ultime due sono quelle che sembrano fornire funzionalità più avanzate e dispongono di un supporto per il deployment e la scalabilità più ricchi.
Le principali differenze tra HiveMQ e AWS IoT includono:

Figura 7: Struttura della piattaforma del progetto, tecnologie e ulteriori piattaforme di supporto.



- infrastruttura - HiveMQ è una piattaforma di messaggistica autonoma che richiede la gestione dell'infrastruttura, mentre AWS IoT è una piattaforma di messaggistica gestita offerta da Amazon Web Services;
- scalabilità - AWS IoT è progettato per essere altamente scalabile, mentre HiveMQ può richiedere una configurazione manuale per la scalabilità;
- AWS IoT offre una vasta gamma di funzionalità integrate, tra cui l'elaborazione dei dati in tempo reale, l'analisi dei dati e l'integrazione con altri servizi AWS, mentre HiveMQ si concentra principalmente sulla messaggistica MQTT;
- costi - AWS IoT può essere più costoso rispetto a HiveMQ, poiché richiede l'utilizzo di servizi AWS come Amazon Kinesis e Amazon S3 per alcune funzionalità avanzate

Tra HiveMQ e AWS IoT Core, si è scelto di utilizzare la prima piattaforma per i seguenti motivi:

- l'architettura risulta essere più elastica, visto che HiveMQ è stato progettato per essere una piattaforma *standalone* ed eventualmente integrabile secondo le necessità;
- HiveMQ permette lo sviluppo in locale ma ha anche strumenti per un utilizzo scalabile e distribuito;
- la configurazione e primo utilizzo di HiveMQ sembrano essere più semplici rispetto ad AWS IoT Core.

Per quanto riguarda l'implementazione, le principali entità e linguaggi utilizzati per ciascuna di esse sono:

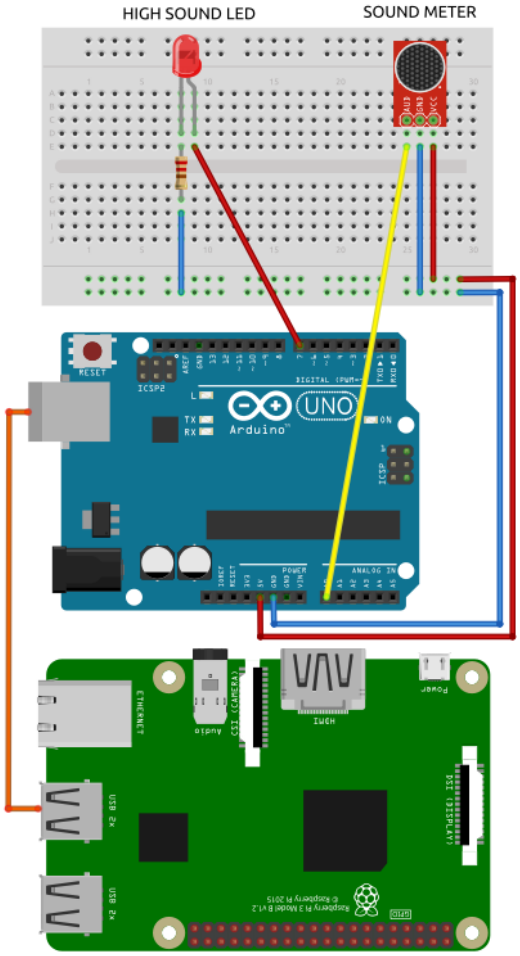
- il Broker è fornito da HiveMQ;
- i client Publisher sono implementati sulle piattaforme fisiche RaspberryPI e Android;
- il client Subscriber è implementato in Python.

A seguire sono esplorate le componenti client separatamente.

5.1.1 MQTT Publisher - Raspberry PI e Arduino

Raspberry Pi è un single-board computer a basso costo e basso consumo energetico utilizzato per una vasta gamma di progetti: grazie alla sua configurazione economica e alla sua facilità d'uso, questa piattaforma è diventato uno strumento popolare per realizzare rapidamente prototipi e progetti IoT.
Il componente fisico dedito alla misurazione dei livelli di rumore è `KY-038`, un microfono a condensatore analogico. Il dispositivo converte le onde sonore in un segnale analogico compreso tra 0 e 1024. La conversione di tale valore in decibel tuttavia non è immediata ma è necessario effettuare una calibrazione approssimativa osservando i valori registrati in assenza e presenza di rumore e comparandoli con un misuratore di rumore che restituisca valori in decibel.
Essendo Raspberry Pi principalmente progettato per la gestione di dati digitali, è necessario utilizzare un ulteriore componente con lo scopo di convertire i dati analogici provenienti dal sensore di rumore. Per questo scopo è stato utilizzato Arduino UNO, una scheda microcontrollore che offre una piattaforma aperta per la programmazione di progetti di automazione e controllo elettronico, permettendo di creare e personalizzare facilmente dispositivi IoT.
Arduino offre una vasta gamma di interfacce di input/output, come ad esempio ADC (Analog-to-Digital Converter) e DAC (Digital-to-Analog Converter), che consentono di acquisire e trasmettere dati analogici da e verso sensori e attuatori. Questo rende più facile la gestione dei segnali analogici provenienti dal mondo fisico, garantendo una maggiore flessibilità e precisione nella misurazione dei dati.
Per inviare dati analogici da Arduino a Raspberry Pi sono state individuate due opzioni accomunate dal fatto che viene instaurato un canale di comunicazione seriale. La comunicazione seriale tra dispositivi IoT consiste nello scambio di dati attraverso una linea seriale, ovvero un singolo bit alla volta, che viene trasmesso in modo sequenziale dal dispositivo mittente al dispositivo destinatario. La comunicazione seriale è utilizzata per connettere dispositivi a distanza relativamente ridotta, ad esempio, per la comunicazione tra componenti di un sistema IoT. Questo tipo di comunicazione è semplice, affidabile e a basso costo ma ha anche limitazioni, come la velocità di trasmissione limitata.
Le due alternative sopra citate sono rispettivamente una connessione seriale tramite porta USB tra le due schede oppure utilizzare una connessione seriale tramite

Figura 8: Schema contenente la struttura fisica del dispositivo di misurazione dei livelli di rumore.



GPIO. La prima soluzione è stata scelta in fase di progettazione del prototipo per questo progetto: si consideri che la scheda Arduino UNO deve già essere connessa ad una porta USB di Raspberry PI per l'alimentazione e questo canale può essere di fatto utilizzata in modalità seriale per la trasmissione di dati senza necessitare di componenti o costi aggiuntivi. In Figura 8 è visibile la disposizione dei dispositivi citati.

5.1.2 MQTT Publisher - Android

Questa soluzione è stata realizzata in conseguenza alla scarsa qualità riscontrata nell'utilizzo del dispositivo di misurazione di rumore `KY-038` proposto nella prima soluzione. Gli smartphone dispongono di tutte le strumentazioni e funzionalità necessarie al progetto che si vuole realizzare. Pertanto è stato ritenuto fondamentale identificare una soluzione alternativa a quella descritta nella Sezione **??**.
Le necessità richieste dal sistema che si vuole realizzare suggeriscono la progettazione di questa applicazione per mezzo del framework denominato Foreground Service offerto dall'architettura Android: questo servizio è pensato proprio per lo sviluppo di particolari tipi di applicazioni che necessitano di rimanere attive per un periodo prolungato anche senza l'intervento dell'utente. Il Foreground Service garantisce che l'attività eseguita non venga terminata dal sistema anche se il dispositivo è a corto di memoria o se l'utente naviga tra le applicazioni. Ciò è importante per le applicazioni che necessitano di una disponibilità costante e affidabile, come quelle che si occupano di monitorare i livelli di rumore.
La struttura del codice progettata segue ovviamente gli stessi principi espressi in Figura 4, aspetto che valida ulteriormente le scelte fatte in fase di progettazione.
Su piattaforma Android, per ottenere il livello di rumore è stato utilizzata la seguente implementazione opportunamente incapsulata all'interno di un Foreground Service:

```
{
    public class ForegroundService extends Service {
        @Override
        public int onStartCommand(...) {
        /* Configurazione metodo per estrazione livelli rumore */
            recorder = new MediaRecorder();

        /* Configurazione di un thread per il monitoraggio periodico */
            new Thread(() -> {
                while (isRunning) {
                    int value = recorder.getMaxAmplitude();
                }
                Thread.sleep(CHECK_TIME);
            }).start();
```

```
        }
    }
}
```

Un'interessante differenza tra il sistema Android e quello Arduino descritto nella Sezione 5.1.1 stà nella qualità e semantica del segnale misurato:

- su Raspberry, tramite l'ausilio del microcontrollore Arduino e misuratore `KY-038`, è estratto un livelli di rumore arbitrario compreso nel range [0, 1024];
- su Android è possibile estrarre il valore di massima `ampiezza` del segnale registrato dal microfono.

Al valore ottenuto su piattaforma Android, bisogna applicare la seguente formula per ottenere il livello di rumore in Decibel:

$$decibel = 20 \cdot \log_{10}(\frac{amp}{amp_{ref}}) \tag{1}$$

dove `amp` è il valore fornito dal dispositivo e `amp_ref` è un valore utilizzato per la calibrazione dei risultati. In questo caso, non avendo a disposizione strumentazione avanzata per la calibrazione, è stato mantenuto il valore ad 1. Va inoltre detto che ogni smartphone dispone di un dispositivo diverso, aspetto che rende la configurazione e l'utilizzo di questo approccio su larga scala, difficile e soggetto a imprecisione.

5.1.3 MQTT Subscriber

La realizzazione del componente Subscriber nell'architettura MQTT ha visto la realizzazione di un programma scritto in linguaggio Python. Di seguito è riportata l'implementazione semplificata del componente in cui sono omesse le query MongoDB esplorate nella Sezione 5.2:

```
client = pymongo.MongoClient(ENV_URL)
db = client.sound_pollution

def on_message(client, userdata, msg):
    # Processamento del messaggio ricevuto dal Broker MQTT
```

```
    msg_metadata, msg_data = str(msg.payload.decode('ASCII')).split("\t")

    # Verificare se il dispositivo esiste nella collezione MongoDB
    meter = db.meters.find_one(...)
    if meter is None:
        # Aggiungere un document nella collezione MongoDB
        db.meters.insert_one(...)
    else:
        # Aggiornare un documento esiste nella collezione MongoDB
        db.meters.update_one(...)

# Configurazione del client MQTT
client = mqtt.Client(CLIENT_NAME)
client.on_message = on_message
client.connect(ENV_IP, ENV_PORT)

# Sottoscrizione ad un topic
client.subscribe(MQTT_TOPIC)
# Processa un messaggio alla volta
client.loop_forever()
```

5.2 Database

Un sistema di gestione del database (DBMS) è fondamentale nell'implementazione di una piattaforma IoT per il monitoraggio dei livelli di rumore scalabile, poiché gestisce e organizza i dati raccolti dai sensori in modo efficiente. Il DBMS fornisce una soluzione affidabile e scalabile per la gestione dei dati, che può essere facilmente acceduta e analizzata da altri componenti della piattaforma, come analisti e sviluppatori di applicazioni.
Il database NoSQL utilizzato nell'implementazione e MongoDB. In fase di progettazione è stato ritenuto opportuno esplorare questa soluzione in confronto alle controparti SQL.
Di seguito sono esposte alcune delle *query* di rilievo realizzate. Un aspetto fondamentale che le accomuna è la necessità di gestire potenzialmente enormi moli di

dati. Se si pensa che dato un particolare dispositivo, esso può generare fino a 1440 misurazioni in una sola giornata.

QUERY 1: aggiornamento di un documento esistente con l'aggiunta di nuove misurazioni.

```
    db.meters.insert_one({
        "device_name": name,
        "gps_lat": float(gps_lat),
        "gps_lng": float(gps_lng),
        "data": [
            {
                "datetime": datetime.fromisoformat(item[0]),
                "value": int(item[1])
            } for item in msg_data
        ],
        "last_update": datetime.now(timezone.utc),
        "active_since": datetime.now(timezone.utc)
    })
```

QUERY 2: aggiornamento di un documento esistente con l'aggiunta di nuove misurazioni.

```
db.meters.update_one({ "device_name" : name }, {
        "$set": {
            "last_update": datetime.now(timezone.utc)
        },
        "$push" : {
            "data" : {
                "$each" : [
                    {
                        "value": int(item[1]),
                        "datetime" : datetime.fromisoformat(item[0]),
                    } for item in msg_data
                ]
            }
        }
    }, upsert=True
```

)

QUERY 3: selezione delle misurazioni ottenute nell'arco di 24 ore per un particolare dispositivo.

```
db.meters.findOne({ "device_name": device_name }, {
        projection: {
        ...
        data: {
          $filter: {
            input: "$data",
            as: "element",
            cond: {
              $gte: [ "$$element.datetime", new Date(new Date().getTime()
                  - (24 * 60 * 60 * 1000))]
            }
          }
        }
        }
    })
```

MongoDB offre diversi meccanismi per garantire la scalabilità dei dati, tra cui:

- replica distribuita - MongoDB supporta la replica dei dati su più nodi, il che consente di distribuire il carico di lavoro su più macchine e garantire la disponibilità dei dati anche in caso di guasto di un nodo;
- sharding[9] - MongoDB supporta lo sharding dei dati su più nodi, il che consente di distribuire i dati su più macchine e gestire il carico di lavoro crescente.

Queste soluzioni non sono state esplorate ulteriormente nell'implementazione del prototipo per questa piattaforma.

[9]Lo sharding è il processo di suddivisione dei dati in più parti (chiamate shard) che vengono poi distribuite su più macchine.

5.3 Frontend

Nella realizzazione delle componenti Frontend della piattaforma di monitoraggio dell'inquinamento acustico sono state utilizzate le seguenti tecnologie:

- NodeJS per la realizzazione del server dedito al monitoraggio del database;
- React per la realizzazione dell'interfaccia utente accessibile tramite browser web;
- Socket.IO come canale bidirezionale nelle comunicazioni tra pagina web e server di monitoraggio.

Nella realizzazione delle comunicazioni tra Frontend e Backend è stato necessario valutare anche l'utilizzo di API Rest, ritenuto non sufficiente per le necessità di progetto. Infatti, per migliorare la scalabilità e le prestazioni del sistema, è necessario instaurare un canale di comunicazione a due vie in cui il server di monitoraggio può comunicare ai vari client i dati aggiornati presenti sul database.
Di seguito è riportato il codice relativo alla gestione delle richieste di monitoraggio implementate in Javascript e con l'ausilio del framework Socket.IO:

```
 const active_watchers = {}

io.on('connection', (socket) => {
  socket.on('start_watch', (device_name) => {
    start_watching(socket, device_name, collection)
  });

  socket.on('stop_watch', () => {
    stop_watching(socket);
  });

  socket.on('disconnect', () => {
    stop_watching(socket);
  });
});
```

Nel dettaglio la funzione `start_watching` inizializza il processo di monitoraggio periodico dei dati sul database, implementato per mezzo della funzione `setInterval` con la quale è possibile eseguire codice ad intervalli periodi. Ovviamente questo meccanismo deve essere opportunamente gestito in fase di terminazione di una sessione di monitoraggio dalla funzione `stop_watching`. Di seguito è riportata una versione semplificata delle due funzioni:

```
function start_watching(socket, device_name, metersCollection) {
  var watcher = active_watchers[device_name];
  if (watcher === undefined) {
    // la collezione non e' monitorata da nessuno
    active_watchers["device_name"] = {
      "device_name": device_name,
      "sockets": [socket],
      "interval": setInterval(() => watcher_behaviour(socket,
          device_name, metersCollection), 1000 * 10)
    };
  } else {
    // la collezione era gia' monitorata da qualche altro utente
    active_watchers[device_name]["sockets"].push(socket);
  }
}

function stop_watching(socket) {
  // trova la collezione osservata dall'utente
  const watcher = watchers.filter(...) ;

  // rimuovi la socket dell'utente da quelle registrate
  delete watcher.sockets[socket];

  // la collezione non e' piu' monitorata da nessun utente?
  if (watcher.sockets.length == 0) {
    // terminazione dell'oggetto Interval
    clearInterval(watcher.interval);
    delete active_watchers[watcher];
```

```
  }
}
```

Infine, la funzione `watcher_behaviour` esegue una query per reperire gli ultimi dati della collezione.

6 Testing e performance

In base ai requisiti definiti nell'introduzione a questo elaborato, è possibile definire una serie di punti su cui verificare lo stato di avanzamento della soluzione realizzata. Questi punti riassumono tutti quelli che sono i requisiti richiesti e sono:

- osservare il funzionamento all'atto pratico dei dispositivi hardware impiegati ed identificarne pregi, difetti e performance;
- verificare il corretto funzionamento della piattaforma MQTT adottata, della connessione e disconnessione dei dispositivi e delle performance;
- verificare la corretta gestione dei dati generati dalle piattaforme hardware, della corretta trasmissione degli stessi;
- osservare il funzionamento dell'interfaccia utente realizzata, testando anche le capacità real-time ed i tempi di visualizzazione dei dati aggiornati.

Tutti i test che si esporranno in seguito, sono stati eseguiti all'interno di una rete privata e l'utilizzo configurazioni prestabilite. In un contesto reale tuttavia è opportuno approfondire tematiche di sicurezza e scalabilità che non sono state testate in questo elaborato.
Molti dei componenti dell'architettura realizzata sono stati testati all'interno di un singolo computer dotato di un processore a 4 core da 2.5GHz, 8GB di RAM ed SSD e sono:

- il Broker HiveMQ dell'architettua MQTT per la trasmissione dei dati;
- i due server NodeJS per il monitoraggio dei dati e fornitura di una pagina web sono stati;
- il database MongoDB per l'immagazzinamento dei dati.

Il restante elemento dell'architettura è il client MQTT dedito alla misurazione di livelli di rumore. Per natura un dispositivo di questo tipo è dislocato geograficamente ed in questo caso sono stati utilizzati due differenti hardware connessi al resto dell'architettura tramite rete locale. Le due piattaforme utilizzate sono:

- Raspberry PI v3 dotato di 1GB RAM e processore a 4 core da 1.2GHz;

- uno smartphone dotato di 2GB di RAM e processore ARM a 4 core da 2GHz.

In seguito a primi test, è stato osservato che la piattaforma Raspberry PI fornisce dei dati di scarsa qualità. Questo fatto è da attribuire al dipositivo di misurazione `KY-038`, il quale soffre di bassa sensibilità alle onde sonore ed anche *drifting*[10] delle misurazioni. Questo è il motivo che ha portato alla realizzazione dell'applicazione Android.
Tramite questa configurazione è stato possibile osservare il funzionamento dell'architettura in real-time con la visualizzazione di risultati tramite interfaccia web. I test condotti sono consistiti nella produzione artificiale di rumore in prossimità dei sensori. È stato possibile osservare che tra la produzione di rumore e conseguente visualizzazione su interfaccia web può trascorrere un tempo mai superiore a 2 minuti. Questo è dovuto al fatto che se un misuratore cattura la presenza di rumore, può passare fino ad 1 minuto prima della trasmissione effettiva al Broker MQTT, questo per il meccanismo di gestione dei dati descritto nella Sezione 4.2.1. La trasmissione del dato tra Broker e client Subscriber e la conseguente aggiunta nel database è pressochè immediata. Parallelamente, il server che sta monitorando i dati reperisce gli ultimi dati una volta al minuto, causando un ulteriore ritardo di 1 minuto.
Nelle Figure 9 e 10 è possibile osservare il grafico prodotto dai test effettuati. Si nota che in Figura10 il segnale risulta essere più stabile a causa della scarsa sensibilità del dispositivo nel distinguere variazioni di rumore.

[10]Il "drifting" di un sensore è variazione graduale e non intenzionale delle sue letture rispetto ad un valore iniziale. Il valore misurato dal sensore può diventare sempre più lontano dalla vera misura che si vuole ottenere, anche se non ci sono cambiamenti nella situazione che sta essendo misurata.

Figura 9: Esempio di esecuzione dell'applicazione web. I dati mostrati provengono dal sensore di uno smartphone.

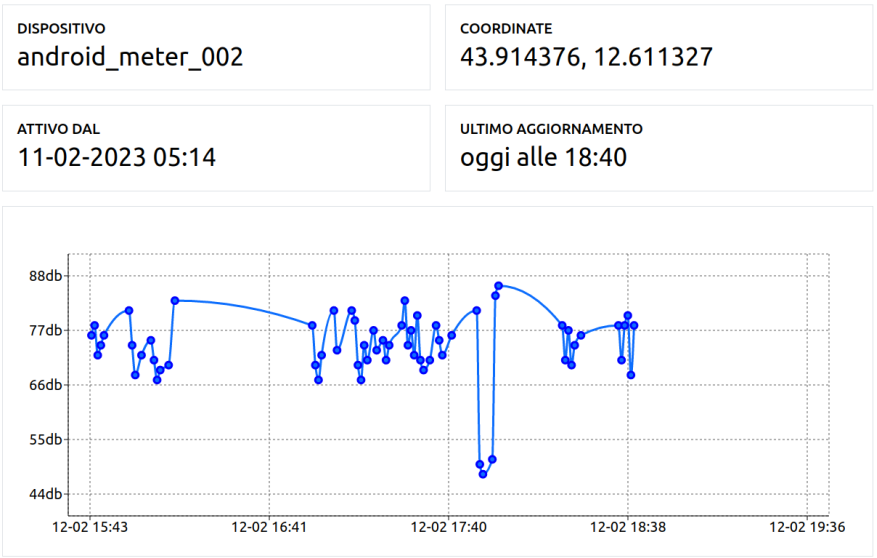
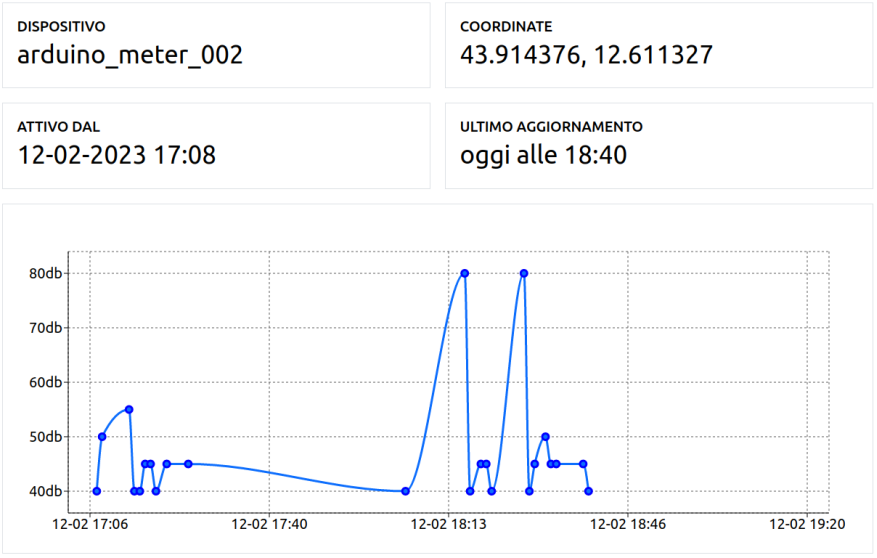


Figura 10: Esempio di esecuzione dell'applicazione web. I dati mostrati provengono dalla piattaforma Raspberry PI e Arduino.



7 Analisi di deployment su larga scala

In una piattaforma di IoT per il monitoraggio ambientale si possono incontrare molte criticità che possono influire sulla resilienza, ovvero la disponibilità e l'affidabilità del sistema.
I sensori e i dispositivi IoT possono a volte perdere la connessione a Internet a causa di problemi di rete o di copertura. Per ovviare a questo problema, si possono utilizzare semplici metodi come la ritrasmissione dei pacchetti o il mantenimento dei dati (aspetto parzialmente trattato nella progettazione e realizzazione di questo progetto) fino alla loro effettiva ricezione.
La piattaforma di monitoraggio può essere soggetta ad un aumento del volume dei dati da parte dei sensori e dei dispositivi IoT, il che può richiedere una maggiore capacità di elaborazione o di archiviazione. Per ovviare a questo problema, è possibile utilizzare tecnologie di virtualizzazione o di scaling automatico per garantire che la piattaforma sia in grado di gestire efficacemente un aumento del carico di lavoro. Nella Sezione 7.1 si tratta una possibile evoluzione dell'infrastruttura implementata in ottica di scalabilità identificando funzionalità avanzate offerte dai vari framework utilizzati.
Un altro tema e la sicurezza e attendibilità dei dati: i dati trasmessi da sensori e dispositivi IoT possono essere influenzati da disturbi o dalla presenza di interferenze. Per garantire l'integrità dei dati, è possibile utilizzare tecnologie di crittografia e di controllo degli errori per proteggerli durante la trasmissione e la memorizzazione.
Nella pratica, il tema del deployment su larga scala di una piattaforma per il monitoraggio dei livelli di rumore può essere scomposto in due aree separate: il tema dell'installazione in punti geografici di dispositivi IoT ed il supporto offerto da piattaforme cloud per la gestione dei dati e fornitura di servizi.
Per quanto riguarda l'installazione e la messa in funzione di una vasta rete di sensori e dispositivi IoT per la rilevazione e trasmissione di dati registrati sui livelli di rumore, si possono immaginare le seguenti fasi nell'ottica di realizzare dispositivi che richiedono poca manutenzione, bassi costi e facilità d'uso:

1. progettazione e realizzazione di una piattaforma più compatta, dotata di sensori migliori ed in grado di sopportare intemperie;

2. configurazione dei vari dispositivi all'interno dell'architettura MQTT utilizzata e creazione di procedure di misurazione del rumore in ottica plug--play [11];

3. installazione in punti chiave e di interesse al pubblico, ad esempio all'interno di edifici, su impianti industriali o su infrastrutture pubbliche;

4. monitoraggio dell'attività dei sensori, manutenzione periodica e verifica della qualità dei dati misurati.

7.1 Evoluzione dei framework utilizzati

Di seguito sono trattate alcune evoluzioni possibili per i sistemi implementati e piattaforme utilizzati nella realizzazione del progetto. I principali framework utilizzati sono:

- HiveMQ come piattaforma di gestione delle comunicazioni tra dispositivi IoT e sistemi sotfware;

- MongooDB come piattaforma per la gestione ed immagazzinamento dei dati generati dai dipositivi IoT;

- NodeJS come piattaforma per l'implementazione di servizi offerti agli utenti finali.

Per quanto riguarda le piattaforme MQTT è facile notare come il Broker dell'architettura possa diventare un collo di bottiglia tra chi genera e gestisce i dati. HiveMQ offre una soluzione cloud-based dotata di funzionalità in ottica scalabile e di resilienza. Tra le principali si individuano le seguenti:

- cluster di nodi - HiveMQ supporta la creazione di cluster di più client Broker, che consente di distribuire la larghezza di banda e la capacità di elaborazione tra più macchine fisiche: un sistema di questo tipo può gestire una quantità più elevata di messaggi e traffico, aumentando la disponibilità complessiva del sistema;

[11] Plug Play è un termine che indica un prodotto o sistema che è progettato per essere facile da installare e utilizzare, senza la necessità di configurazioni o impostazioni complesse

- load balancing - HiveMQ utilizza un meccanismo di bilanciamento del carico per distribuire equamente il traffico di messaggi tra i client del cluster aiutando a evitare la congestione di un singolo nodo e garantendo una disponibilità continua del sistema;

- replica dei messaggi - HiveMQ supporta la replicazione dei messaggi tra i nodi del cluster ossia se un nodo dovesse diventare non disponibile, un altro nodo del cluster può prendere il suo posto e continuare a fornire servizi ai client.

Anche MongoDB fornisce già delle soluzioni cloud-based; alcuni dei meccanismi adottabili per garantire la scalabilità e la disponibilità anche in presenza di grandi quantità di dati e traffico sono:

- replica-set - MongoDB utilizza repliche dei dati per mantenere più copie su più nodi del cluster per garantire una maggiore disponibilità del servizio;

- sharding - un meccanismo di divisione dei dati in parti più piccole che possono essere distribuite su più nodi del cluster permettendo al sistema di gestire grandi quantità di dati e di elaborare richieste in parallelo su più nodi, aumentando la velocità complessiva del sistema;

- scalabilità automatica - MongoDB è progettato per l'autoscalabilità, il che significa che il sistema può automaticamente adattarsi a eventuali cambiamenti nel carico di lavoro.

Node.js è una piattaforma l'esecuzione di applicazioni web. Un unico server potrebbe non essere in grado di gestire un elevato traffico, aspetto che potrebbe portare all'assenza di servizio per congestione (DOS). In questo caso è necessario appoggiarsi a soluzioni cloud-based per l'hosting di servizi web in grado di fornire scalabilità. Ci sono molte opzioni disponibili tra cui Amazon Web Services (AWS), Google Cloud Platform (GCP) e Microsoft Azure. Queste soluzioni sono in grado di gestire in maniera dinamica l'elaborazione e la memoria.

8 Piano di lavoro

Essendo l'elaborato stato svolto in autonomia, è stato possibile realizzare prototipi e testare i vari componenti in maniera estremamente rapida. Una volta definiti i requisiti, è stata successivamente realizzata la progettazione del sistema di massima e sono state identificate le tecnologie da utilizzare. Di fondamentale importanza è stata la suddivisione dell'architettura in tre componenti indipendenti e l'identificazione dei cananli di comunicazione tra di essi. Grazie a ciò, è stato possibile seguire un approccio ciclico ed evolutivo nello sviluppo ed ogni iterazione ha consentito di raffinare il sistema secondo i requisiti definiti. La modularità ha consentito anche una migliore sperimentazione dei vari componenti in fase di test.
In linea di massima, la realizzazione dell'implementazione ha seguito il seguente ordine:

1. realizzazione della logica dei sensori e misurazione livelli di rumore;
2. realizzazione infrastruttura MQTT ed integrazione con i sensori;
3. realizzazione del database ed integrazione con infrastruttura MQTT;
4. realizzazione delle funzionalità di monitoraggio ed integrazione con il database.

La realizzazione delle componenti di monitoraggio è avvenuta solamente quando le altre componenti per la generazione e gestione dei dati erano già operative.
L'approccio ciclico ha reso difficile stabilire con esattezza il tempo impiegato su ciascun componente del sistema. Il tempo impiegato è riportato di seguito:

Attività	**Giorni uomo**
Studio e primo utilizzo di HiveMQ	1.5
Realizzazione del client Publisher su Raspberry PI e Arduino	2
Realizzazione del client Publisher su Android	1
Realizzazione del client Subscriber	1.5
Realizzazione del database MongoDB e delle query utilizzate	2
Realizzazione del server di monitoraggio in NodeJS	1.5
Realizzazione del frontend in React	1.5

Il resto del tempo è stato dedicato alla stesura della relazione (avvenuta in maniera trasversale all'implementazione del progetto) il test e la lettura della documentazione utilizzata.

9 Conclusioni

La progettazione di una piattaforma IoT per la misurazione, raccolta e monitoraggio dei livelli di rumore rappresenta una sfida significativa e l'inquinamento in ambienti urbani rappresenta un problema sempre più crescente per la qualità della vita dei cittadini e per la loro salute. Per questo motivo, la necessità di una soluzione tecnologica che possa raccogliere, monitorare e analizzare i dati sui livelli di rumore sta diventando sempre più pressante.
Eventuali sviluppi futuri potrebbero essere riassunti di seguito:

- migliorare la qualità dei dispositivi fisici dediti alla misurazione dei livelli di rumore;
- permettere la generazione di report periodici automatici in modo tale da consentire un monitoraggio non necessariamente real-time;
- l'adozione di un approccio che consenta la generazione di notifiche in tempo reale;
- realizzare dei digital-twin per il monitoraggio dei dispositivi fisici stessi;
- trasporre questa infrastruttura su piattaforme cloud;
- migliorare l'applicazione Android e permettere agli utenti di monitorare in autonomia il proprio ambiente.

Appendice

.1 Legislatura in materia di inquinamento acustico

La normativa italiana in materia di inquinamento acustico è stata elaborata per proteggere la salute e il benessere della popolazione dai danni causati dall'esposizione prolungata a livelli elevati di rumore. La sua applicazione è importante per garantire un ambiente più salubre e confortevole per tutti.

.1.1 Legge 447/95

La Legge 447/95 "Norme per la protezione dell'ambiente dall'inquinamento acustico" è una normativa italiana che stabilisce i criteri e le modalità per la prevenzione e la riduzione dell'inquinamento acustico.
La legge definisce l'inquinamento acustico come *l'impatto negativo del rumore sull'ambiente e sulla salute umana*. Prevede la classificazione delle zone in base al livello di inquinamento acustico e stabilisce limiti e norme specifiche per ciascuna di esse.
La legge prevede inoltre la possibilità di adottare misure per la prevenzione e la riduzione dell'inquinamento acustico, tra cui la regolamentazione delle fonti di rumore, l'utilizzo di tecnologie a basso impatto acustico e la sensibilizzazione della popolazione sull'importanza di una gestione sostenibile del rumore.
La Legge 447/95 prevede anche l'obbligo per le amministrazioni pubbliche di elaborare piani di tutela acustica e di stabilire procedure per la valutazione e la gestione del rumore negli ambienti esterni.
Inoltre, la legge stabilisce sanzioni per le violazioni dei limiti acustici e della normativa in materia di inquinamento acustico, per garantire che i responsabili delle fonti di rumore ne tengano conto nella loro attività.

.1.2 Direttiva 2008/1/CE

La Direttiva 2008/1/CE, nota anche come Direttiva "Quadro sull'inquinamento acustico", è una direttiva europea che mira a garantire una tutela della salute umana e dell'ambiente dall'inquinamento acustico. La direttiva stabilisce un quadro normativo per la prevenzione e la riduzione dell'inquinamento acustico a livello

europeo e definisce i compiti e le responsabilità degli Stati membri nella tutela della salute umana e dell'ambiente da questa forma di inquinamento.
La direttiva stabilisce, in particolare, che gli Stati membri devono prendere le misure necessarie per prevenire e controllare l'inquinamento acustico nell'ambiente esterno e all'interno degli edifici, garantendo la qualità acustica dell'ambiente e la salute dei cittadini. Inoltre, la direttiva stabilisce che gli Stati membri devono adottare misure per la valutazione e la gestione dell'inquinamento acustico, tra cui la definizione di limiti acustici, la mappatura acustica del territorio e la realizzazione di piani di gestione acustica.

Riferimenti bibliografici

[1] Sadman Shahriar Alam, Akib Jayed Islam, Md. Mahmudul Hasan, Mohammad Nokib Monsur Rafid, Nishako Chakma, and Md. Nafiz Imtiaz. Design and development of a low-cost iot based environmental pollution monitoring system. In *2018 4th International Conference on Electrical Engineering and Information Communication Technology (iCEEiCT)*, pages 652–656, 2018.

[2] Organizazione Mondiale della Sanità. Global noise pollution map. https://lukasmartinelli.ch/maps/noise-pollution.html, 2023.

[3] Afidatul Nadia Mok Hat, W.M. Wan Syahidah, C.W.Y.C.W. Zuhelmi, F. Atan, and S. Khadijah. Iot base on air and sound pollution monitoring system. *Journal of Physics: Conference Series*, 2319(1):012013, aug 2022.

[4] Software Languages Lab. Noisetube. https://www.noisetube.net/, 2023.

[5] Arnab Kumar Saha, Sachet Sircar, Priyasha Chatterjee, Souvik Dutta, Anwesha Mitra, Aiswarya Chatterjee, Soummyo Priyo Chattopadhyay, and Himadri Nath Saha. A raspberry pi controlled cloud based air and sound pollution monitoring system with temperature and humidity sensing. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 607–611, 2018.

[6] Inc. Wildlife Acoustics. Song meter sm2bat. https://www.wildlifeacoustics.com/uploads/user-guides/Song-Meter-SM2Bat-Suppement.pdf, 2011.