Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea Triennale in Ingegneria e Scienze Informatiche

PROGETTAZIONE E SVILUPPO DI UN SISTEMA A MICROSERVIZI PER LA RACCOLTA DI DATI ATMOSFERICI

Relatore:
Prof.ssa Silvia Mirri

Presentata da:
Matteo Zeno Bagli

Correlatore:
Dr. Giovanni Delnevo
Dr. Kelvin Olaiya

Sessione II
Anno Accademico 2024-2025

A Miriana

Introduzione

Il monitoraggio della qualità dell'aria urbana rappresenta oggi una delle sfide più rilevanti per la salute pubblica e la sostenibilità ambientale. In un'epoca caratterizzata dalla crescente urbanizzazione e dall'intensificarsi delle attività antropiche, la necessità di disporre di sistemi affidabili per il rilevamento e l'interpretazione dei dati sull'inquinamento atmosferico diventa sempre più necessario. Le concentrazioni di particolato $PM_{2.5}$ e PM_{10}, biossido di azoto, ozono troposferico e anidride solforosa nelle aree metropolitane influenzano direttamente il benessere dei cittadini, richiedendo strumenti di monitoraggio che siano al contempo tecnicamente accurati e facilmente accessibili.

L'evoluzione tecnologica degli ultimi anni ha aperto nuove prospettive nell'ambito del monitoraggio ambientale, rendendo possibile lo sviluppo di sistemi distribuiti di sensori che operano in tempo reale. Parallelamente, la diffusione delle tecnologie web e dei dispositivi mobili ha creato l'opportunità di democratizzare l'accesso alle informazioni ambientali, trasformando dati tecnici complessi in visualizzazioni comprensibili e utilizzabili da un'ampia gamma di utenti. In questo contesto si inserisce il presente elaborato di tesi, che propone lo sviluppo di AirQualityInsight, un sistema integrato per la simulazione, raccolta e visualizzazione di dati sulla qualità dell'aria.

Questo applicativo fornisce all'utente una dashboard web che presenta una mappa interattiva di Bologna, sulla quale vengono disegnate in tempo reale le misurazioni effettuate da un insieme di sensori dislocati sul territorio. La dashboard web, progettata secondo i principi del design mobile-first, offre

un'esperienza utente fluida e intuitiva sia su dispositivi desktop che mobili. La mappa interattiva, arricchita da layer informativi sui confini amministrativi e le ZTL, permette agli utenti di esplorare i dati ambientali nel loro contesto geografico, facilitando la comprensione delle correlazioni spaziali tra inquinamento e caratteristiche territoriali.

Una particolare attenzione è stata dedicata alla visualizzazione dei dati attraverso heatmap dinamiche che traducono le concentrazioni di inquinanti in rappresentazioni cromatiche intuitive. Questa tecnica di visualizzazione, basata sulla scala di colori dell'European Air Quality Index (EAQI), permette di identificare immediatamente le aree critiche e di monitorare l'evoluzione temporale della qualità dell'aria. L'integrazione di funzionalità di filtraggio e aggregazione consente agli utenti di personalizzare la visualizzazione in base alle proprie esigenze, concentrandosi su specifici inquinanti o periodi temporali.

Il presente elaborato si propone di dimostrare come l'integrazione strategica di tecnologie moderne possa dare vita a sistemi di monitoraggio ambientale che coniughino efficacemente accuratezza tecnica e accessibilità d'uso. Attraverso l'implementazione di AirQualityInsight, si intende evidenziare le potenzialità delle architetture distribuite nel contesto delle smart cities, proponendo soluzioni che possano essere replicate e adattate a diverse realtà urbane. La metodologia adottata, che privilegia l'approccio incrementale e la modularità dei componenti, rende il sistema facilmente estensibile e manutenibile, caratteristiche fondamentali per la sostenibilità a lungo termine di infrastrutture tecnologiche destinate al servizio pubblico.

Di seguito viene illustrata la struttura e gli obiettivi di ciascun capitolo del presente elaborato di tesi:

- Il **primo capitolo** (1) introduce l'obiettivo del progetto di tesi e fornisce il quadro teorico necessario alla comprensione del lavoro sviluppato. Viene presentata un'analisi dello stato dell'arte dei sistemi esistenti affini alla soluzione proposta, unitamente ai concetti fondamentali del

dominio applicativo di riferimento. Il capitolo si conclude con una descrizione dettagliata delle funzionalità del sistema implementato.

- Il **secondo capitolo** (2) illustra le tecnologie adoperate nello sviluppo del progetto. Considerata la natura web-based dell'applicazione, le tecnologie vengono categorizzate distinguendo tra quelle utilizzate per il front-end e quelle impiegate per il back-end.

- Il **terzo capitolo** (3) presenta in modo approfondito la fase di progettazione e implementazione del sistema. Dopo aver delineato l'architettura generale, viene fornita una descrizione dettagliata di ciascun componente, dai servizi implementati all'interfaccia utente front-end.

Indice

Elenco delle figure

Elenco delle tabelle

Acronimi

ACID Atomicity, Consistency, Isolation and Durability. 47

AJAX Asynchronous JavaScript and XML. 28

API Application Programming Interface. vi, 17, 19, 25, 27, 28, 33, 43, 54, 56, 62, 63, 65, 67, 69, 71, 85, 90, 91

AQI Air Quality Index. 2, 4, 20, 21

BSON Binary JSON. 46

CAP Codice Avviamento Postale. 77, 78

CLI Command Line Interface. 25

CO Monossido di carbonio. v, 7

CORS Cross-Origin Resource Sharing. 62

CQRS Command Query Responsibility Segregation. 45

CRS Coordinate Reference Systems. 8

CSS Cascading Style Sheets. 24, 27

CSV Comma Separated Values. 17

DOM Document Object Model. 24, 25

EAQI European Air Quality Index. ii, ix, xi, 2–5, 73–75, 89

EEA European Environment Agency. 2, 80

EPSG European Petroleum Survey Group. v, 8–11, 14

FIFO First In First Out. 77

GIS Geographic Information System. 15, 19

GPS Global Positioning System. 12, 14

HTML HyperText Markup Language. 24, 27

HTTP HyperText Transfer Protocol. 31, 34, 63, 70

IOGP International Association of Oil & Gas Producers. 8

IOT Internet of Things. 34, 35, 43, 47, 91

IP Internet Protocol. 56

IQA Indice di Qualità dell'Aria. 2

ISO Internation Standard Organization. 63

JSON Javascript Object Notation. 17, 47, 60, 62, 63, 67, 70, 71

NO2 Diossido di azoto. v, 2, 6

npm Node Package Manager. 33

O3 Ozono. v, 2, 7

OSM OpenStreetMap. v, 8, 10, 11, 13–17, 19, 26, 28, 91

PM10 Particolato PM10. 2

PM2.5 Particolato PM2.5. 2, 80

PWA Progressive Web App. 25

Capitolo 1

Introduzione del sistema

Il primo capitolo ha due obiettivi: il primo obiettivo è l'introduzione del progetto di tesi, con le sue caratteristiche e funzionalità; il secondo obiettivo è l'analisi dello stato dell'arte degli attuali sistemi simili al progetto dell'elaborato di tesi, descrivendone le caratteristiche principali, ponendo i diversi sistemi a confronto e approfondendone le funzionalità.

1.1 Scopo del progetto

Lo scopo del progetto di tesi è quello di presentare un sistema di simulazione atto alla collezione ed interpretazione di dati sulla qualità dell'aria di una determinata area geografica.

Tali dati provengono da misurazioni fittizie prodotte da sensori distribuiti in un'area di studio. I sensori inviano delle rilevazioni ad un intermediario, il quale le raccoglie e rende disponibili. Viene quindi disposta una dashboard attraverso la quale gli utenti possono fruirne sotto forma di tabelle e di una mappa interattiva.

La dashboard consiste in una web app mobile-first che permette di visualizzare l'area geografica coinvolta e i dati sulla qualità dell'aria. Lo sviluppo della dashboard prende come riferimento le applicazioni della stessa tipologia attualmente presenti come stato dell'arte.

L'area geografica di interesse è il comune di Bologna, entro il cui perimetro vengono posizionati i sensori di simulazione [4].

1.2 Qualità dell'aria

I dati sulla qualità dell'aria vengono definiti dai paesi secondo indici e scale. L'Indice di Qualità dell'Aria (IQA) (in inglese Air Quality Index)rappresenta il modo in cui i governi scelgono di comunicare con la popolazione la qualità dell'aria. Esso converge il livello di diversi inquinanti in un unico indice comprensibile, consentendo di identificare più facilmente il livello di inquinamento e l'eventuale rischio associato.

Regioni e paesi diversi utilizzano scale differenti per indicare la qualità dell'aria in base all'inquinamento locale e a considerazioni sulla salute. Esistono decine di indici locali utilizzati in tutto il mondo; ad esempio, in alcune regioni dell'Australia si utilizzano sistemi basati su numeri, mentre altre ne usano uno basato su categorie. Canada, Giappone e Stati Uniti definiscono indici di qualità dell'aria distinti, così come l'Agenzia europea dell'ambiente [5].

Il servizio online "The European Air Quality Index" (EAQI) dell'European Environment Agency (EEA) e della Commissione Europea fornisce informazioni sulla qualità dell'aria basate su più di 2000 stazioni di rilevamento in tutta Europa [6]. L'indice consiste in una mappa interattiva che mostra la qualità dell'aria a livello locale, andando ad analizzare i 5 livelli di inquinanti più pericolosi per le persone e per l'ambiente:

- Particolato PM2.5 e PM10
- Ozono (O3)
- Diossido di azoto (NO2)
- Anidride solforosa (SO2)

Gli utenti possono fare zoom sulla mappa o ricercare città Europee per controllare la qualità globale dell'aria e verificare i livelli di inquinanti registrati dai sensori dalle stazioni locali. L'Indice mostra un rating globale per ogni stazione di rilevamento, andando a marcare la mappa con un punto colorato, per ognuno dei 5 inquinanti. Il colore indica un livello di qualità dell'aria:

- Molto buono (verde acqua)
- Buono (verde)
- Moderato (giallo)
- Basso (rosso)
- Molto basso (rosso ciliegia)
- Estremamente basso (viola)

Nella tabella 1.1 sono riportati i valori relativi al livello di qualità dell'aria.

Inquinante	**Buono**	**Discreto**	**Moderato**	**Scarso**	**Molto scarso**	**Estremamente scarso**
Particelle inferiori a 2.5 μm ($PM_{2.5}$)	0-5	6-15	16-50	51-90	91-140	>140
Particelle inferiori a 10 μm (PM_{10})	0-15	16-45	46-120	121-195	196-270	>270
Biossido di azoto (NO_2)	0-60	61-100	101-120	121-160	161-180	>180
Ozono (O_3)	0-10	11-25	26-60	61-100	101-150	>150
Biossido di zolfo (SO_2)	0-20	21-40	41-125	126-190	191-275	>275

Tabella 1.1: Indici di qualità dell'aria per diversi inquinanti secondo l'European Air Quality Index (EAQI), concentrazioni espresse in $\mu g/m^3$ [3].

Il portale riporta inoltre una serie di messaggi atti a fornire raccomandazioni, sia alla fascia di popolazione normale che a quella più sensibile, in funzione dell'Air Quality Index (AQI) misurato. La fascia di soggetti più sensibile comprende sia adulti che bambini con problemi respiratori che cardiaci.

Nella tabella 1.2 vengono indicati i messaggi suggeriti alle fasce di popolazione relative in relazione al livello di qualità dell'aria.

AQI	**Popolazione normale**	**Popolazione sensibile**
Buono	La qualità dell'aria è buona. Goditi le tue solite attività all'aperto.	La qualità dell'aria è buona. Goditi le tue solite attività all'aperto.
Discreto	Goditi le tue solite attività all'aperto.	Goditi le tue solite attività all'aperto.
Moderato	Goditi le tue solite attività all'aperto.	Considera di ridurre le attività intense all'aperto, se avverti sintomi.
Scadente	Considera di ridurre le attività intense all'aperto, se avverti sintomi come irritazione agli occhi, tosse o mal di gola.	Considera di ridurre le attività fisiche, soprattutto all'aperto, specialmente se avverti sintomi.
Molto scadente	Considera di ridurre le attività intense all'aperto, se avverti sintomi come irritazione agli occhi, tosse o mal di gola.	Riduci le attività fisiche, soprattutto all'aperto, specialmente se avverti sintomi.
Estremamente scadente	Riduci le attività fisiche all'aperto.	Evita le attività fisiche all'aperto.

Tabella 1.2: Messaggi relativi alla salute

Nelle seguenti figure 1.1 vengono mostrate le due possibili grafiche della mappa interattiva riportante l'European Air Quality Index (EAQI).

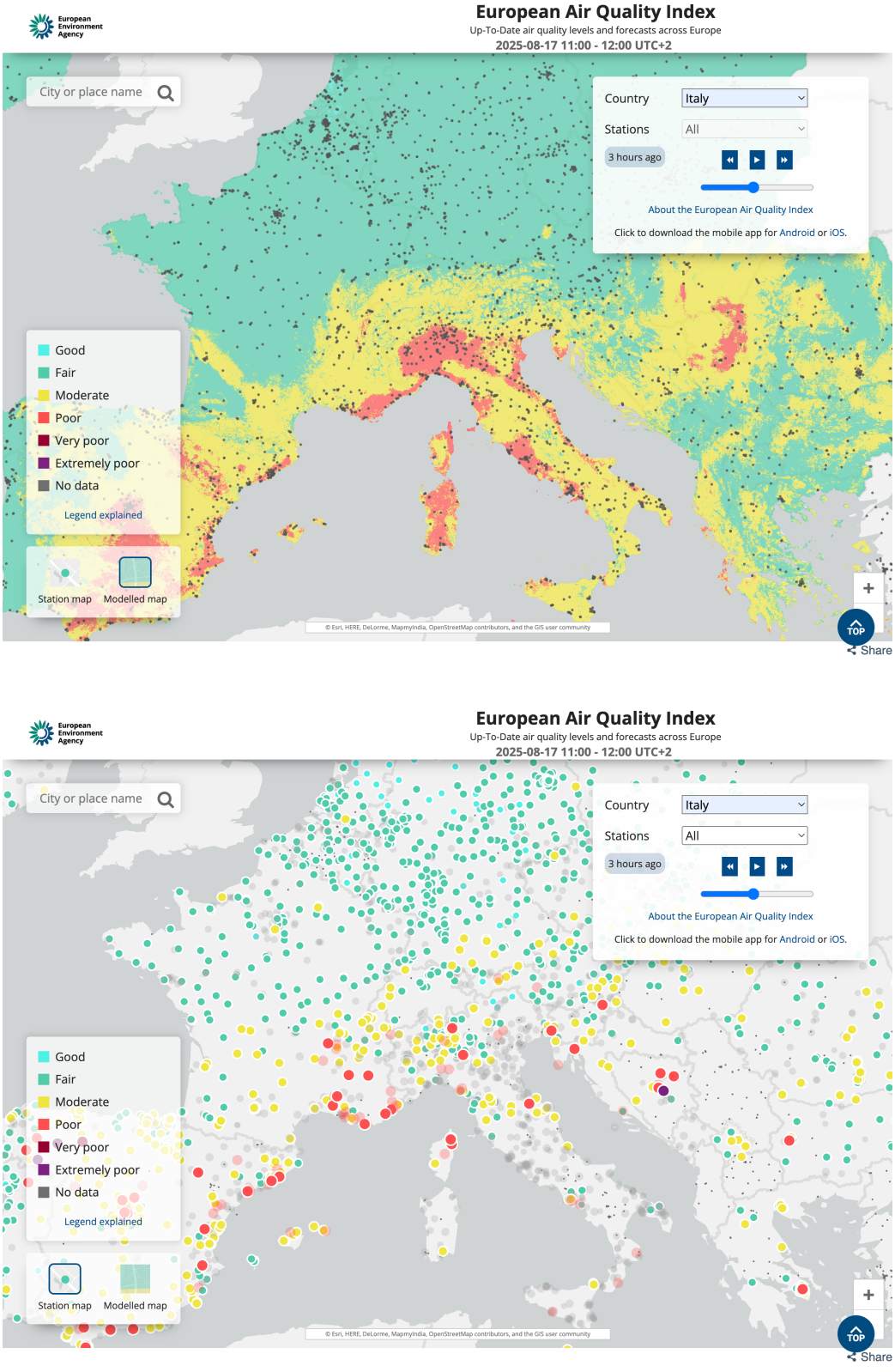


Figura 1.1: European Air Quality Index (EAQI) per Bologna: (sopra) Mappa "modellata" interattiva, (sotto) Mappa "a stazioni" interattiva

1.3 Principali inquinanti atmosferici e loro origini

La valutazione della qualità dell'aria si basa sul monitoraggio di specifici contaminanti presenti nell'atmosfera. Secondo Google [7], i parametri più frequentemente rilevati nelle aree urbane includono i seguenti elementi.

1.3.1 Materiale particolato (PM)

Insieme di particelle microscopiche, solide e liquide, sospese nell'atmosfera. Le frazioni PM_{10} e $PM_{2.5}$ identificano particelle con diametro rispettivamente inferiore a 10 e 2,5 micrometri. Le principali sorgenti comprendono:

- Traffico veicolare e combustione nei motori
- Riscaldamento domestico a biomassa
- Processi industriali
- Fenomeni naturali come incendi boschivi e tempeste di sabbia

1.3.2 Biossido di azoto (NO2)

Gas caratteristico dell'inquinamento urbano, generato prevalentemente da:

- Emissioni del trasporto su strada
- Attività industriali
- Impianti di produzione energetica
- Sistemi di riscaldamento civile

1.3.3 Ozono troposferico (O3)

Diversamente dall'ozono stratosferico che ci protegge dai raggi UV, quello presente negli strati bassi dell'atmosfera costituisce un inquinante secondario. Si forma attraverso reazioni fotochimiche tra:

- Composti organici volatili
- Ossidi di azoto
- Radiazione solare

Le fonti primarie dei precursori sono veicoli, centrali termoelettriche e processi industriali.

1.3.4 Anidride solforosa (SO2)

Gas dall'odore caratteristico e dalle proprietà irritanti, originato da:

- Combustione di carbone e derivati petroliferi negli impianti energetici
- Raffinazione del petrolio
- Produzione di cemento
- Attività vulcanica

1.3.5 Monossido di carbonio (CO)

Gas inodore e tossico prodotto dalla combustione incompleta di combustibili fossili in veicoli e macchinari industriali.

1.4 Sistema di coordinate

Nella sezione seguente verranno introdotti i sistemi di coordinate presenti nei sistemi di elaborazione trattati.

1.4.1 EPSG:4326

Per il progetto di tesi, sono state utilizzate come base per la collocazione dei sensori di rilevamento della qualità dell'aria le mappe di OpenStreetMap (OSM). Tale strumento utilizza il sistema di coordinate World Geodetic System 1984 (WGS84) per rappresentare latitudine e longitudine.

Nello specifico, il sistema di riferimento prevede un formato di coordinate geografiche decimali, in codice EPSG 4326. L'acronimo EPSG sta per European Petroleum Survey Group, fondato negli anni '80 dall'industria petrolifera europea con scopo originale la standardizzazione dei sistemi di coordinate per l'esplorazione petrolifera offshore nel Mare del Nord. Le compagnie petrolifere infatti usavano sistemi di coordinate diversi, creando confusione e errori costosi [8, 9].

Dal 1986 al 2005, tale sistema di coordinate viene gestito dall'European Petroleum Survey Group. Dal 2005 ad oggi, viene trasferito alla International Association of Oil & Gas Producers (IOGP), ma il nome EPSG è rimasto di comune utilizzo.

Il registro EPSG è diventato lo standard mondiale per i sistemi di coordinate (Coordinate Reference Systems - CRS), i datum geodetici, le proiezioni cartografiche, le trasformazioni tra sistemi e le unità di misura.

Andiamo a definire ciascuno degli elementi elencati:

- Sistemi di coordinate (CRS): Un framework completo che definisce come identificare univocamente ogni punto sulla Terra usando numeri, come latitudine/longitudine o coordinate piane.

- Datum geodetici: la forma matematica di riferimento della Terra (ellissoide) e il suo posizionamento nello spazio, che serve come "punto zero" per tutte le misurazioni.

- Proiezioni cartografiche: i metodi matematici per "appiattire" la superficie curva della Terra su una mappa bidimensionale.

- Trasformazioni tra sistemi: le formule matematiche per convertire coordinate da un sistema di riferimento a un altro.
- Unità di misura: le scale di riferimento per esprimere distanze, angoli e posizioni (metri, gradi, piedi, radianti, ecc.) all'interno di ciascun sistema di coordinate.

Ogni sistema di coordinate ha un codice EPSG univoco:

- EPSG:4326 = WGS84 (lat/lon in gradi) [10].
- EPSG:3857 = Web Mercator (usato da Google Maps) [11].
- EPSG:32633 = UTM zone 33N (comune in Italia centrale) [12].
- EPSG:3003 = sistema italiano storico. Italy zone 1 [13].

Il sistema utilizzato per il progetto è il datum WGS84, codice EPSG:4326, in formato di coordinate geografiche decimali che rappresenta la latitudine, la longitudine, con una precisione fino a 7-8 cifre decimali.

Il riferimento fondamentale per il calcolo della latitudine è l'Equatore (latitudine 0°), definito come il cerchio massimo che divide la Terra in emisfero Nord e Sud. Si misura come angolo dal piano equatoriale verso nord o sud con un range da -90° (Polo Sud) a +90° (Polo Nord).

I paralleli di riferimento sono:

- Tropico del Cancro: 23°27'N
- Tropico del Capricorno: 23°27'S
- Circolo Polare Artico: 66°33'N
- Circolo Polare Antartico: 66°33'S

Di seguito un'immagine rappresentativa 1.2.

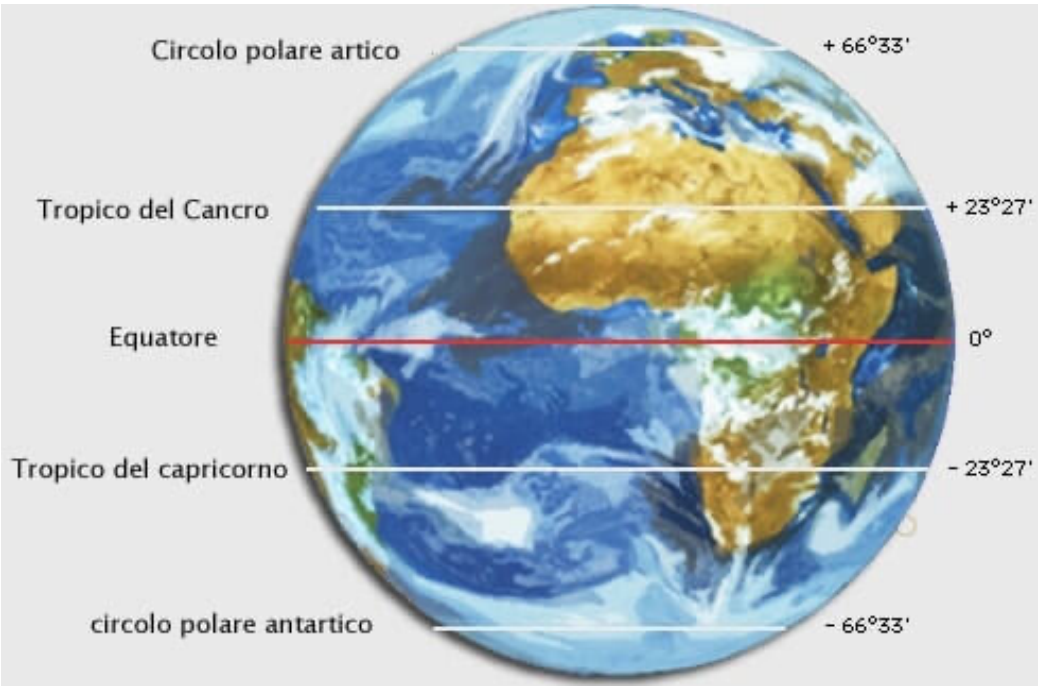


Figura 1.2: Paralleli della terra [1].

Per la longitudine, il riferimento principale è il Meridiano di Greenwich (longitudine 0°), definito come il meridiano che passa per l'Osservatorio Reale di Greenwich, London. È stato stabilito nel 1884 alla Conferenza Internazionale del Meridiano, e si misura come angolo dal meridiano di Greenwich verso est o ovest con un range da -180° (ovest) a +180° (est). I meridiani di riferimento sono:

- Linea del Cambio di Data: 180°
- Antimeridiano: Opposto a Greenwich

Il sistema di coordinate geografiche funziona come una griglia sferica, dove la latitudine sono le linee orizzontali (paralleli), la longitudine sono le linee verticali (meridiani) e le unità sono espresse in gradi decimali o gradi-minuti-secondi.

Anche se OpenStreetMap (OSM) memorizza i dati in WGS84, per la visualizzazione delle mappe utilizza spesso la proiezione Web Mercator (EP-

SG:3857), che è quella usata anche da Google Maps, Bing Maps e altri servizi di mappe online. Questa proiezione "schiaccia" le aree polari ma mantiene gli angoli, rendendola ideale per la navigazione. La proiezione Web Mercator (nota anche come Pseudo-Mercator o Google Mercator) è la proiezione cartografica più utilizzata per le mappe online moderne.

1.4.2 EPSG:3857

La proiezione WebMercator con codice EPSG:3857 (o EPSG:900913), rappresenta la terra come una sfera perfetta, invece che come un ellissoide rappresentato dallo standard WGS84. Utilizza come unità i metri.

La proiezione "avvolge" la Terra con un cilindro tangente all'equatore, poi proietta tutti i punti sulla superficie del cilindro. Tale sistema ha come vantaggio la conservazione degli angoli, la facilità di implementazione nei sistemi informatici, uno zoom semplice da utilizzare e un tile system efficiente che divide la mappa in quadrati per il caricamento veloce.

Gli svantaggi tuttavia sono la distorsione sia delle aree che delle distanze: infatti la Groenlandia appare grande quanto l'Africa, ma in realtà l'Africa è 14 volte più grande. Le distanze soprattutto alle latitudini elevate appaiono estremamente distorte. Inoltre non vengono mostrati i poli: la proiezione si interrompe infatti a circa 85°N e 85°S.

Rimane comunque il sistema più diffuso tra i servizi di mappe online, quali Google Maps, OpenStreetMap (OSM), Bing Maps, Apple Maps, Mapbox. Infatti risulta molto efficiente per la navigazione urbana e la consultazione online, ma di scarso valore per analisi geografiche che richiedono precisione nelle aree o distanze.

1.4.3 WGS84

Il sistema WGS84 fu creato dal Dipartimento della Difesa USA negli anni '80, ufficializzato nel 1984, rivisto nel 1996 e 2004, e periodicamente

raffinato. Lo scopo originale era creare un sistema unificato per il GPS militare americano [14].

WGS84 definisce la forma della Terra usando un ellissoide di riferimento con questi parametri che definiscono il modello matematico della Terra, la quale nella realtà è un geoide.

Semiasse maggiore (a) è il raggio equatoriale della Terra, definito come distanza dal centro della Terra all'equatore. Il valore è di 6.378.137 m.

Schiacciamento (f) misura quanto la Terra è "schiacciata" ai poli. Tale valore risulta in uno schiacciamento della Terra del 0,34%, in quanto la Terra non è una sfera perfetta. Il valore numerico è di $298{,}257223563^{-1}$.

Formula:

$$f = \frac{a - b}{a} \tag{1.1}$$

Semiasse minore (b) è il raggio polare della Terra, definito come distanza dal centro della Terra ai poli. Risulta circa 21,4 km più corto del semiasse maggiore, con un valore di 6.356.752,314245 metri.

Formula:

$$b = a \times (1 - f) \tag{1.2}$$

Eccentricità misura la deviazione dalla forma sferica. Per un valore vicino a 0, si ha una sfera perfetta. Il valore WGS84 indica un ellissoide molto vicino alla sfera.

Eccentricità = 0,0818191908

Formula:

$$e^2 = \frac{a^2 - b^2}{a^2} \tag{1.3}$$

Il sistema di coordinate 3D ha origine nel centro di massa terrestre, con un sistema di riferimento temporale basato sul Tempo Atomico Internazionale. Possiede una precisione teorica centimetrica, mentre l'accuratezza pratica dipende dal metodo di misurazione.

Di seguito il modello dell'elissoide WGS84 1.3.

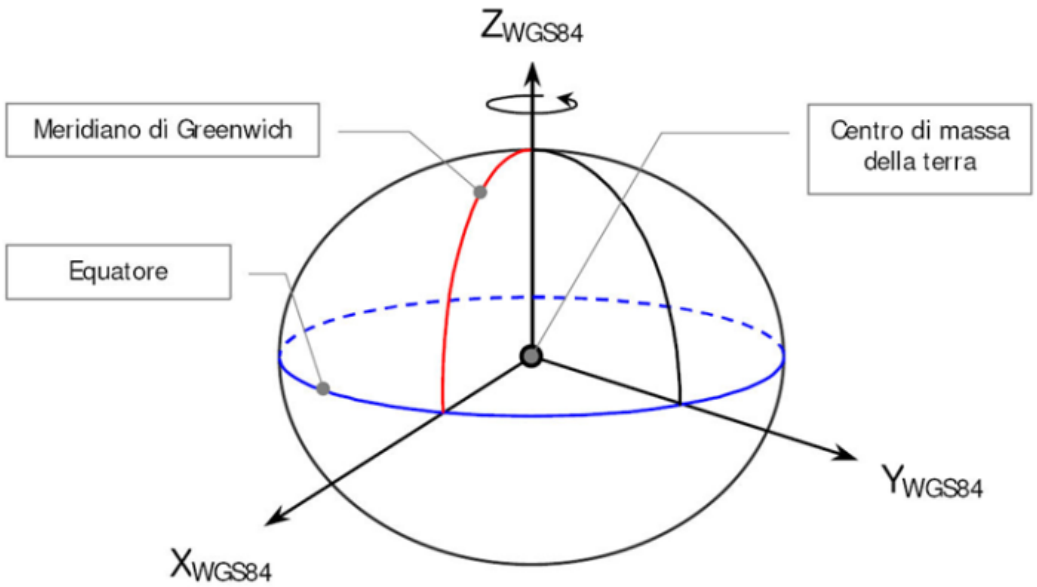


Figura 1.3: Ellissoide WGS84 [2].

1.5 OpenStreetMap

OpenStreetMap (OSM) è un progetto collaborativo di mappatura web che raccoglie dati geospaziali per creare e distribuire mappe online, liberamente disponibili a chiunque abbia una connessione Internet [15]. Una volta accessibile, OpenStreetMap (OSM) permette agli utenti Internet di contribuire e modificare i dati geospaziali, rendendolo di fatto l'equivalente cartografico di Wikipedia.

OpenStreetMap (OSM) è emerso come un progetto globale e una comunità che opera con l'obiettivo di creare e mantenere un database libero e modificabile e una mappa del mondo basata sui contributi di volontari cartografi [16]. Con il suo database che include quasi 7,5 miliardi di punti dati (nodi), contribuiti da circa 1,8 milioni di utenti a marzo 2022, è forse l'esempio più riuscito di un progetto di geoinformazione crowdsourced e del concetto di informazione geografica volontaria [16].

1.5.1 Storia e fondazione

Il progetto OpenStreetMap (OSM) è stato fondato nel 2004 nel Regno Unito da Steve Coast, allora studente presso l'University College London [17], il quale ne ha registrato il dominio openstreetmap.org [17]. La prima strada è stata inserita il 10 dicembre 2004 dopo che Steve aveva pedalato intorno a Regent's Park a Londra con un ricevitore GPS (Global Positioning System) [18].

Il 22 agosto 2006, la OpenStreetMap (OSM) Foundation è stata istituita per incoraggiare la crescita, lo sviluppo e la distribuzione di dati geospaziali liberi e fornire dati geospaziali per chiunque li usi e li condivida [19].

1.5.2 Caratteristiche tecniche

OpenStreetMap (OSM) è mantenuto da cartografi volontari di tutto il mondo che utilizzano dispositivi GPS, fotocamere portatili e laptop per la mappatura sul campo. I dati raccolti sono integrati con fotografie aeree open source digitalizzate e mappe gratuite da fonti governative e commerciali [15].

Il progetto utilizza il sistema di coordinate WGS84 (EPSG:4326) per rappresentare latitudine e longitudine, permettendo l'integrazione con sistemi GPS e altre piattaforme cartografiche digitali.

OpenStreetMap (OSM) è stato definito come "la wikificazione delle mappe" da alcuni ricercatori [20], evidenziando il parallelo con il modello collaborativo di Wikipedia. Il progetto ha attirato considerevole attenzione da molteplici attori, incluse industrie, governi e organizzazioni umanitarie [16].

1.5.3 Applicazioni e ricerca accademica

La comunità accademica ha mostrato un interesse duraturo per OpenStreetMap (OSM), documentato in diverse recenti review [20, 21]. Le applicazioni di ricerca includono studi sulla qualità dei dati, analisi delle dinamiche della comunità OSM, e utilizzo dei dati OSM per applicazioni di machine learning e remote sensing [21].

OpenStreetMap (OSM) rappresenta oggi uno dei più grandi progetti di dati aperti su Internet e un esempio notevole di sistemi di informazione geografica partecipativa (participatory GIS) [22].

1.5.4 Struttura dati di OpenStreetMap (OSM)

Per rappresentare informazioni geografiche, OpenStreetMap (OSM) utilizza un modello dati basato su tre elementi fondamentali: nodi (*nodes*), percorsi (*ways*) e relazioni (*relations*) [23].

Nodi (*Nodes*) Un nodo rappresenta un singolo punto geografico sulla mappa, caratterizzato da:

- Un identificatore numerico unico a 64 bit
- Coordinate geografiche (latitudine e longitudine) in WGS84
- Un set opzionale di tag (coppie chiave-valore)
- Numero di versione e timestamp per il controllo delle modifiche

I nodi possono rappresentare elementi puntuali come punti di interesse (negozi, semafori, fermate autobus ...) oppure essere utilizzati come vertici per definire la geometria di percorsi più complessi [24].

Percorsi (*Ways*) Un percorso è una sequenza ordinata di nodi che forma una polilinea o un poligono chiuso. I percorsi non memorizzano direttamente la loro posizione geografica, ma mantengono una lista ordinata di identificatori di nodi [24]. Possono rappresentare:

- Elementi lineari: strade, fiumi, confini
- Elementi areali: edifici, laghi, parchi (quando il percorso è chiuso)

Relazioni (*Relations*) Le relazioni sono collezioni strutturate di oggetti (nodi, percorsi o altre relazioni) utilizzate per definire relazioni logiche o geografiche tra elementi diversi [25]. Ogni membro di una relazione può avere un ruolo opzionale che descrive la sua funzione. Esempi includono:

- Multipoligoni (aree con buchi o confini complessi)
- Percorsi di trasporto pubblico
- Confini amministrativi
- Limitazioni di svolta

Sistema di tag Tutti gli elementi OSM possono essere descritti attraverso tag, costituiti da coppie chiave-valore in formato Unicode (massimo 255 caratteri) [26]. Il sistema di tagging libero permette di includere un numero illimitato di attributi per ogni elemento, con la comunità che si accorda su combinazioni standard per le caratteristiche più comuni.

1.5.5 Applicazioni e casi d'uso

OpenStreetMap (OSM) trova applicazione in numerosi contesti, dalla ricerca accademica alle applicazioni commerciali e umanitarie.

Ricerca accademica OpenStreetMap (OSM) rappresenta una risorsa preziosa per la ricerca in diversi ambiti:

- **Machine Learning e Remote Sensing**: utilizzato per training di modelli di classificazione del territorio e miglioramento della copertura dei dati [21].
- **Analisi urbana**: studio della qualità dell'aria, accessibilità, pianificazione dei trasporti.
- **Analisi della mobilità**: creazione di database nazionali per infrastrutture ciclabili [27].

Applicazioni commerciali Molte aziende utilizzano OpenStreetMap (OSM) come base per i loro servizi:

- Servizi di mappe (Facebook, Foursquare, Strava).
- Applicazioni di delivery e logistica.
- Servizi di car sharing e bike sharing.
- Piattaforme di e-commerce per localizzazione servizi.

1.5.6 Integrazione programmatica

Per integrare i dati OpenStreetMap (OSM) in applicazioni software, sono disponibili diverse librerie e strumenti.

API e strumenti di accesso ai dati OpenStreetMap (OSM) fornisce diversi metodi per accedere e interrogare i suoi dati, adatti a diverse esigenze applicative.

Overpass API L'Overpass API è un motore di database ottimizzato per interrogazioni di sola lettura sui dati OSM [28]. A differenza dell'API principale ottimizzata per l'editing, Overpass API è progettata per consumatori di dati che necessitano di pochi elementi selezionati rapidamente o fino a circa 10 milioni di elementi in alcuni minuti, utilizzando criteri di ricerca come posizione, tipo di oggetti, proprietà dei tag o combinazioni di questi [28].

L'API utilizza un linguaggio di query specializzato (Overpass QL) e offre diversi formati di output tra cui XML, JSON e CSV. Le query possono essere testate interattivamente attraverso Overpass Turbo [29], un'interfaccia web che permette di visualizzare i risultati su una mappa interattiva.

Ad esempio, per la scelta della collocazione dei sensori, è stata realizzata una query per ottenere tutti gli incroci fra le arterie principali e le strade secondarie del comune di Bologna. Questa scelta è stata fatta in quanto tali intersezioni, soprattutto quelle semaforizzate, costituiscono zone

di concentrazione delle emissioni veicolari, dove il traffico in attesa genera un incremento localizzato dell'inquinamento atmosferico, rendendo quindi la misurazione più significativa.

La seguente query 1.1 in Overpass QL estrae tali intersezioni:

Listing 1.1: Query in Overpass QL di esempio

```
[out:json][timeout:90];

// Area corrispondente al comune di Bologna
area[name="Bologna"][admin_level=8]->.bologna;

// Strade principali
(
  way(area.bologna)["highway"="motorway"];
  way(area.bologna)["highway"="trunk"];
  way(area.bologna)["highway"="primary"];
  way(area.bologna)["highway"="secondary"];
  way(area.bologna)["highway"="tertiary"];
)->.major;

// Strade secondarie
(
  way(area.bologna)[highway="unclassified"];
  way(area.bologna)[highway="residential"];
  way(area.bologna)[highway="living_street"];
  way(area.bologna)[highway="service"];
  way(area.bologna)[highway="pedestrian"];
  way(area.bologna)[highway="track"];
)->.minor;

// Intersezioni stradali
node(w.major)(w.minor);
out;
```

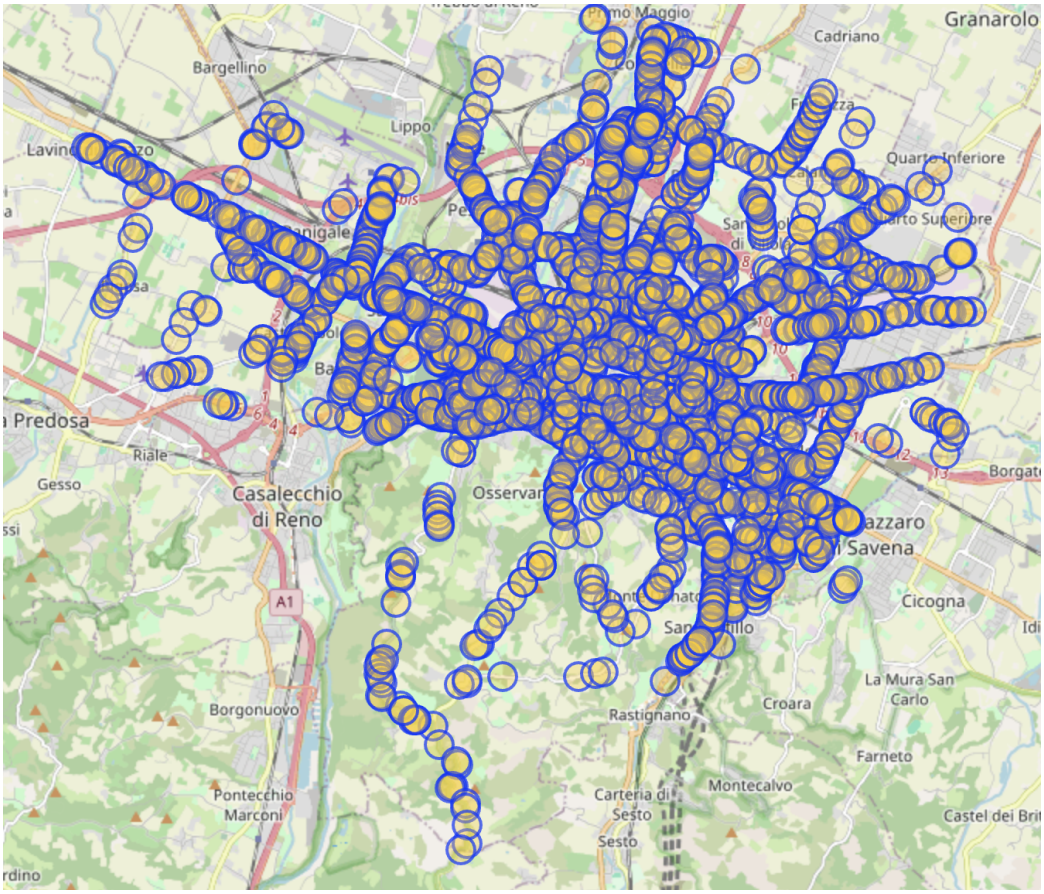


Figura 1.4: Intersezioni fra le strade principali e secondarie nel comune bolognese

Formati di dati I dati OSM sono disponibili in diversi formati [30]:

- **OSM XML**: formato nativo basato su XML
- **PBF**: formato binario compresso per maggiore efficienza
- **GeoJSON**: per integrazione con applicazioni web
- **Shapefile**: per compatibilità con software Geographic Information System (GIS) tradizionali

L'ecosistema di strumenti e API di OpenStreetMap (OSM) offre quindi una piattaforma completa e flessibile per lo sviluppo di applicazioni geo-

spaziali, dalla semplice visualizzazione di mappe fino a complessi sistemi di analisi territoriale e routing.

1.6 Funzionalità del sistema

La presente sezione illustra in modo approfondito i requisiti del sistema da implementare, derivanti dallo studio preliminare. Il progetto di tesi si propone di sviluppare un'applicazione web conforme ai requisiti funzionali e non funzionali che verranno esposti.

1.6.1 Requisiti funzionali

- Visualizzazione della mappa: l'utente dovrà poter visualizzare una mappa in cui
  - verrà centrata l'area interessata dall'applicativo
  - verranno mostrate le misurazioni in tempo reale ed i conseguenti Air Quality Index (AQI) relativi ai sensori
  - sarà possibile scegliere quali livelli osservare
  - sarà presentato un livello heatmap le Air Quality Index (AQI) ed ogni altro inquinante in esame
  - sarà possibile arrestare e riprendere la raccolta delle registrazioni live
  - verranno predisposte tabelle per elencare sensori, misurazioni, statistiche e log
  - dovrà essere possibile visualizzare le informazioni relative ad ogni sensore quali nome, posizione, data ultima registrazione, livello di qualità dell'aria corrente e livelli singoli inquinanti
- Fruizione dei dati: verranno predisposte delle rotte specifiche per fare interrogazioni preformate alle registrazioni salvate.

1.6.2 Requisiti non funzionali

- Design: l'interfaccia web dovrà essere mobile-first nativamente e responsive.
- Usabilità: l'interfaccia web dovrà risultare semplice ed intuitiva, rendendo l'esperienza più semplice, favorendo elementi grafici come colori e simboli rispetto al testo.
- Deployment: il progetto dev'essere strutturato in modo da essere eseguito in un ambiente virtualizzato, rendendo così il deploy possibile attraverso un solo comando. L'installazione di dipendenze deve limitarsi esclusivamente agli strumenti di containerizzazione e/o virtualizzazione appropriati.

1.7 Principali difficoltà del progetto

Le principali sfide affrontate durante lo sviluppo del progetto riguardano:

- La gestione delle misurazioni registrate e l'orchestrazione dei vari microservizi.
- La generazione di pseudo misurazioni aleatorie (mocking).
- La visualizzazione in tempo reale dei dati sulla mappa e la corretta presentazione della heatmap (mappa di calore) dei correnti Air Quality Index (AQI) relativi ai rispettivi sensori.

Capitolo 2

Tecnologie

Nel seguente capitolo verranno presentate le tecnologie adottate per la realizzazione del progetto AirQualityInsight. Data la natura di applicazione web del progetto, le tecnologie sono state classificate distinguendo tra quelle utilizzate per il front-end e quelle per il back-end.

2.1 Applicazioni front-end

In questa sezione verranno presentate le principali tecnologie front-end impiegate nello sviluppo del progetto AirQualityInsight. L'applicazione front-end verrà sviluppata in Javascript utilizzando i seguenti framework:

- Vue per lo sviluppo dell'architettura e della struttura generale dell'applicazione.
- Leaflet per la visualizzazione della mappa interattiva.

Di seguito, la descrizione dettagliata delle singole tecnologie.

2.1.1 Vue

Vue.js è un framework JavaScript progressivo per la costruzione di interfacce utente, creato da Evan You nel 2014 [31]. Nato dall'esperienza dell'au-

tore con AngularJS [32] durante il suo periodo in Google, Vue è stato progettato per essere facilmente adottabile, combinando le peculiarità di Angular e React [33] con una curva di apprendimento più veloce per gli sviluppatori che si interfacciano con esso.

La caratteristica distintiva di Vue risiede nella sua natura progressiva, che consente di adottarlo gradualmente in base alle necessità del progetto. Al livello più elementare, Vue può essere utilizzato come una semplice libreria JavaScript per arricchire pagine HTML esistenti con funzionalità interattive. A livello intermedio invece, il framework è in grado di gestire componenti complessi in relazione fra loro utilizzando sistemi di routing sofisticati. Infine, al livello più avanzato, Vue permette la costruzione di Single Page Applications (SPA) [34] complete.

Il sistema di reattività costituisce uno dei pilastri fondamentali dell'architettura di Vue. Questo meccanismo garantisce infatti la sincronizzazione automatica tra il modello dati e la vista, attraverso un sistema di data binding bidirezionale, ossia un meccanismo di sincronizzazione automatica che mantiene allineati i dati tra il modello dell'applicazione (model) e l'interfaccia utente (view) in entrambe le direzioni. Le computed properties permettono la definizione di proprietà calcolate che si aggiornano automaticamente quando cambiano le loro dipendenze, mentre i watchers offrono la possibilità di creare osservatori personalizzati per reagire a modifiche specifiche dei dati.

Vue adotta un'architettura basata su componenti, in cui ciascun componente costituisce un elemento modulare e riutilizzabile dell'interfaccia utente. I Single File Components (SFC), caratterizzati dall'estensione `.vue`, incapsulano template HTML, logica JavaScript e stili CSS in un unico file, facilitando la manutenzione e l'organizzazione del codice. La comunicazione tra componenti avviene attraverso un sistema ben definito di props per il passaggio di dati da genitore a figlio e di eventi per la comunicazione inversa.

A livello tecnico, Vue implementa un sistema di template dichiarativo che utilizza una sintassi intuitiva. Il framework utilizza un Virtual DOM per ottimizzare le prestazioni, effettuando confronti tra stati precedenti e nuovi

per minimizzare gli aggiornamenti del DOM reale.

L'ambiente Vue si caratterizza per la presenza di molteplici strumenti di tipologie differenti. Per la gestione e compilazione dei progetti Vue, vengono maggiormente utilizzati Vue CLI [35] e Vite [36]: Vue CLI permette la per la creazione e gestione di progetti da riga di comando, mentre Vite rappresenta un build tool con tempi di compilazione ridotti e maggiore semplicità d'utilizzo. Per il routing esiste Vue Router [37], il quale gestisce il routing nelle Single Page Applications (SPA). Per la gestione centralizzata dello stato si hanno Vuex [38] e Pinia [39]. Infine, per il debugging, Vue DevTools [40] fornisce strumenti avanzati attraverso estensioni per browser.

L'evoluzione di Vue ha visto il passaggio da Vue 2, che ha consolidato l'adozione del framework nell'ambiente enterprise, a Vue 3, rilasciato nel 2020. L'innovazione più significativa di questa major release è probabilmente la Composition API, che permette una migliore organizzazione della logica dei componenti e facilita la riusabilità del codice rispetto alla Options API. Inoltre, è stato migliorato il supporto nativo per il Typescript e, con l'introduzione del supporto per il tree-shaking, sono state ridotte le dimensioni generali dei bundle.

In conclusione, Vue.js trova applicazione ideale nello sviluppo di applicazioni web moderne, dalle Single Page Applications (SPA) alle Progressive Web App (PWA), dai dashboard amministrativi alle piattaforme e-commerce. La sua natura incrementale lo rende particolarmente adatto per la migrazione graduale di applicazioni legacy e per la prototipazione rapida di nuove funzionalità.

2.1.2 Leaflet

Leaflet rappresenta una delle librerie JavaScript open-source più popolari per la creazione di mappe interattive ottimizzate per dispositivi mobili e l'integrazione di funzionalità cartografiche nelle applicazioni web. Sviluppata inizialmente da Vladimir Agafonkin nel 2011 [41] è stata successivamente mantenuta da una comunità attiva di sviluppatori per la cartografia digitale.

I punti cardine di Leaflet sono la semplicità, l'efficienza e l'usabilità, qualità che lo rendono uno strumento di larga diffusione per sviluppatori che necessitano di implementare mappe interattive senza la complessità di librerie più pesanti, grazie anche ad un footprint di soli 39 KB di JavaScript compresso [42].

L'architettura modulare di Leaflet costituisce uno dei suoi principali punti di forza. Tale libreria è stata infatti progettata seguendo il principio della responsabilità singola (SRP), dove ogni componente gestisce solo alcuni aspetti specifici della funzionalità cartografica. Questa approccio consente di scegliere di utilizzare solo i moduli necessari per il proprio progetto, riducendo così l'impatto sulle prestazioni e facilitando la manutenzione del codice.

Le funzionalità core di Leaflet includono la gestione di layer cartografici multipli, il supporto per vari formati di tile server, la gestione di marker personalizzabili, popup informativi, controlli di navigazione e zoom interattivo. La libreria supporta nativamente i più comuni sistemi di proiezione cartografica, con particolare attenzione alla proiezione Web Mercator utilizzata dalla maggior parte dei servizi di tile moderni come OpenStreetMap (OSM) e Google Maps.

È possibile inoltre installare moduli accessori (plugin) sviluppati da terzi per integrare funzionalità aggiuntive. Tali moduli vengono realizzati, mantenuti e resi disponibili dalla comunità open source. Questi vanno ad estendere le funzionalità base della libreria, ad esempio, aggiungendo supporto per clustering di marker, drawing tools, integrazione con servizi di geocoding, visualizzazione di heatmap, gestione di dati GPX e altro ancora. Questa modularità permette di costruire applicazioni cartografiche complesse partendo da una base leggera e aggiungendo solo le funzionalità effettivamente necessarie.

Dal punto di vista delle prestazioni, Leaflet implementa diverse ottimizzazioni per garantire un'esperienza utente fluida. Il sistema di gestione dei tile implementa strategie di caching e lazy loading, caricando solo le porzioni di mappa effettivamente presenti nell'area di visualizzazione. Il rendering

dei marker è ottimizzato attraverso tecniche di virtualizzazione che gestiscono efficientemente svariati punti disegnati sulla mappa senza appesantire le prestazioni di scrolling e zoom.

L'API di Leaflet offre un'interfaccia intuitiva e ben documentata che segue convenzioni JavaScript moderne. La libreria supporta sia approcci programmatici tradizionali che pattern più moderni come la programmazione funzionale e l'utilizzo di Promise per operazioni asincrone. L'integrazione con framework JavaScript contemporanei come Vue.js, React e Angular è facilitata da wrapper specifici e da una architettura event-driven che si integra naturalmente con i sistemi di reattività di questi framework.

La compatibilità cross-platform di Leaflet consente di supportare i browser moderni desktop e mobile. La libreria gestisce automaticamente le differenze tra dispositivi touch e mouse, offrendo un'esperienza di navigazione ottimizzata per ogni tipo di interfaccia.

Dal punto di vista della personalizzazione, Leaflet offre un controllo granulare sull'aspetto e il comportamento delle mappe. Il sistema di styling basato su CSS permette di personalizzare completamente l'aspetto dei controlli, marker e popup, mentre l'API JavaScript consente di definire comportamenti interattivi complessi. La libreria supporta la creazione di marker personalizzati utilizzando HTML, CSS e SVG, permettendo la realizzazione di interfacce cartografiche su misura.

L'integrazione con servizi di tile esterni è una delle principali caratteristiche del framework, che consentono di supportare nativamente servizi come OpenStreetMap, Google Maps, HERE ed altri. Questa flessibilità permette agli sviluppatori di scegliere il provider di tile più adatto alle proprie esigenze in termini di qualità, copertura geografica e costi, mantenendo la stessa API di sviluppo.

La libreria dei dati geografici supporta il formato GeoJSON nativo, permettendo la visualizzazione di geometrie complesse come poligoni, linee e punti direttamente da dati strutturati, come le aree comunali ed i confini territoriali ed amministrativi di regioni, province ed altre suddivisioni geo-

grafiche. L'integrazione con servizi REST e API geografiche è semplificata dalla gestione intrinseca di richieste AJAX e dalla capacità di processare dati in tempo reale.

La community di Leaflet mantiene attivo il core della libreria e contribuisce con supporto tecnico, numerosi plugin, tutorial, esempi d'utilizzo ed una documentazione ufficiale completa ed aggiornata.

In termini di performance e scalabilità, Leaflet gestisce elasticamente applicazioni di varia natura e dimensione. Per le applicazioni più semplici, la libreria offre una soluzione plug-and-play che richiede configurazione minima per installare le sole funzionalità necessarie, in modo da migliorare l'esperienza d'utilizzo.

Il seguente esempio 2.1 mostra come sia possibile realizzare una mappa con tile OpenStreetMap (OSM) centrata su Piazza Maggiore (Bologna), ottenendo la mappa mostrata in figura 2.1:

Listing 2.1: Mappa Bologna con Leaflet

```
1  // Coordinates of Piazza Maggiore
2  const lat = 44.4939;
3  const lng = 11.3426;
4
5  const map = L.map('map').setView([lat, lng], 13);
6
7  // Tiles layer (OpenStreetMap)
8  L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x
       }/{y}.png', {
9    maxZoom: 19,
10   attribution: '<a href="http://www.openstreetmap.org/
       copyright">OpenStreetMap</a>'
11 }).addTo(map);
12
13 // Current position marker
14 const marker = L.marker([lat, lng]).addTo(map);
15
```

```
16  // Marker popup
17  marker.bindPopup(`
18    <div style="text-align: center;">
19      <h4>Piazza Maggiore, Bologna</h4>
20      <p>Lat: ${lat}</p>
21      <p>Lng: ${lng}</p>
22    </div>
23  `).openPopup();
24
25  // Circle on area
26  L.circle([lat, lng], {
27    color: 'red',
28    fillColor: '#f03',
29    fillOpacity: 0.2,
30    radius: 1000 // meters
31  }).addTo(map);
32
33  // Optional controls
34  L.control.scale({
35    imperial: false,
36    metric: true
37  }).addTo(map);
```

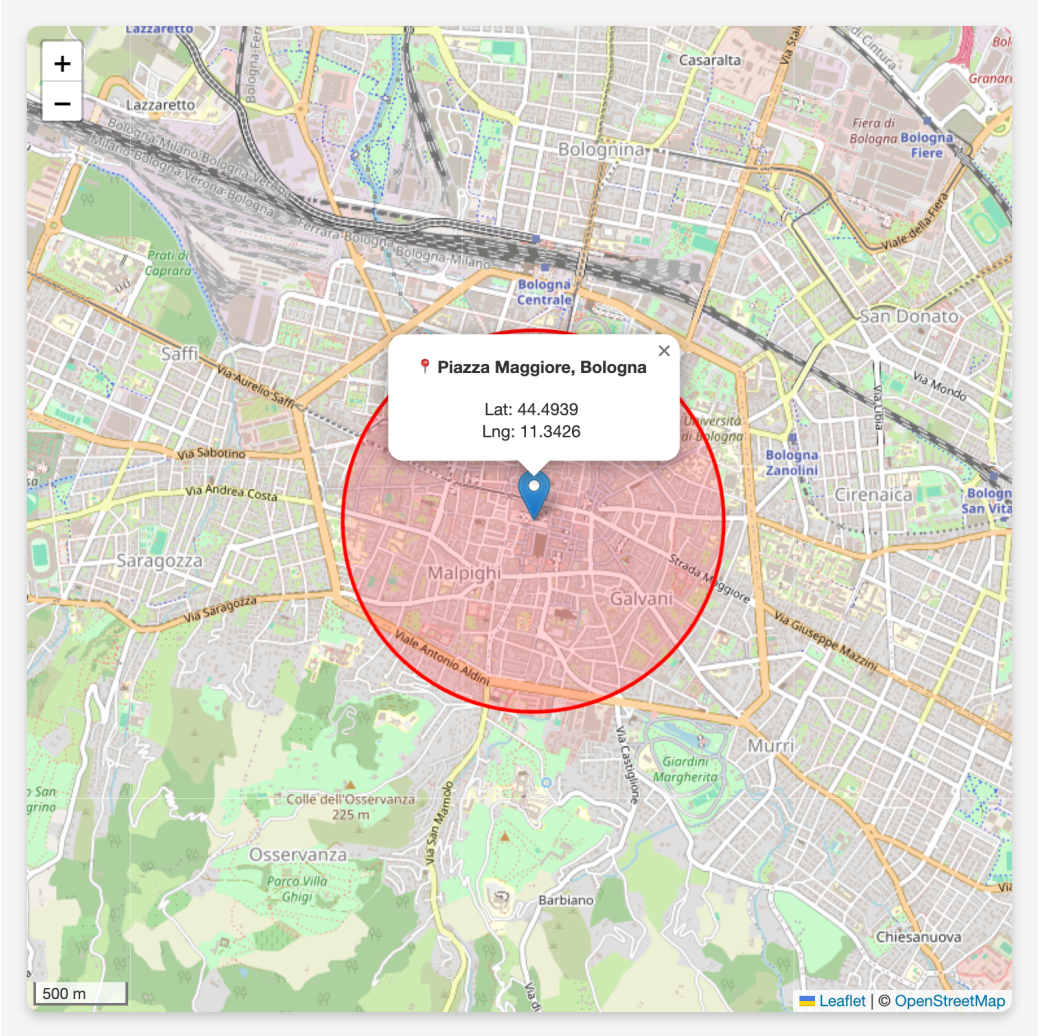


Figura 2.1: Piazza Maggiore, Bologna

In conclusione, Leaflet si posiziona come una soluzione affidabile per l'integrazione di funzionalità cartografiche in applicazioni web moderne. Combinando leggerezza, potenza, estensibilità e facilità d'uso diventa una scelta indicata per sviluppatori che necessitano di implementare mappe interattive performanti e personalizzabili, dal prototipo rapido all'applicazione complessa.

2.2 Applicazioni back-end

In questa sezione verranno presentate le principali tecnologie back-end impiegate nello sviluppo del progetto AirQualityInsight.

Verranno utilizzati Node.js per realizzare il server, Python per simulare i dispositivi di misurazione della qualità dell'aria, Kafka come broker dei messaggi e MongoDB per mantenere un database documentale delle registrazioni.

2.2.1 Node.js

Node.js è un runtime system open-source multipiattaforma costruito sul motore JavaScript V8 di Google Chrome [43]. A differenza dei tradizionali ambienti JavaScript che operano esclusivamente nel browser, Node.js permette l'esecuzione di codice JavaScript lato server, abilitando lo sviluppo di applicazioni web complete utilizzando un unico linguaggio di programmazione.

Le principali caratteristiche di Node.js sono l'adozione di un paradigma event-driven, un approccio non-blocking I/O ed un processo single-threaded. Il modello di programmazione orientato agli eventi consente di gestire efficientemente operazioni asincrone basate sulla manifestazione di determinati eventi. Le operazioni di input e di output non bloccanti migliorano le performance dell'applicativo. L'uso di un singolo thread principale con un event loop riduce la complessità della gestione della concorrenza.

Il seguente esempio 2.2 mostra come creare un server HTTP minimale utilizzando Node.js:

Listing 2.2: Server HTTP base in Node.js

```
1 const http = require('http');
2
3 const server = http.createServer((req, res) => {
4   res.writeHead(200, {'Content-Type': 'text/plain'});
5   res.end('Hello world!');
```

```
6 });
7
8 const PORT = 3000;
9 server.listen(PORT, () => {
10   console.log(`Server listening on port ${PORT}`);
11 });
```

Node.js offre ampie funzionalità per l'interazione con il file system. L'esempio seguente 2.3 dimostra la lettura asincrona di un file:

Listing 2.3: Lettura asincrona di file

```
1 const fs = require('fs');
2
3 // Asynchronous reading
4 fs.readFile('file.txt', 'utf8', (err, data) => {
5   if (err) {
6     console.error('Error in file reading:', err);
7     return;
8   }
9   console.log('File content:', data);
10 });
11
12 // Synchronous reading (not file for large files)
13 try {
14   const data = fs.readFileSync('example.txt', 'utf8');
15   console.log('File content:', data);
16 } catch (err) {
17   console.error('Error in file reading:', err);
18 }
```

L'architettura di Node.js si basa sul concetto di event loop, un meccanismo che permette di gestire multiple operazioni I/O senza bloccare l'esecuzione del programma principale [44]. Questo approccio lo rende particolarmente adatto per applicazioni che richiedono alta concorrenza con operazioni I/O

intensive, come API REST, applicazioni real-time su dispositivi distribuiti e microservizi.

Le performance di Node.js sono generalmente migliori rispetto ai server tradizionali multi-threaded per applicazioni I/O-bound, grazie alla riduzione dell'overhead dovuto al context switching tra thread e alla gestione efficiente della memoria. Questo poiché i sistemi tradizionali adottano tecniche di gestione delle richieste creando un nuovo thread per ogni nuova connessione, mentre Node.js, operando su un singolo thread ed utilizzando chiamate I/O non bloccati, riesce a supportare un maggior numero di richieste concorrenti nell'event loop.

Quello che ha reso popolare tale framework è la praticità e la versatilità che lo contraddistinguono. Risulta infatti una buona scelta nel realizzare velocemente applicazioni web scalabili, grazie alla sua capacità di gestire un numero elevato di connessioni simultanee.

Node.js dispone di un gestore di pacchetti chiamato "npm" (Node Package Manager), il quale fornisce un insieme di librerie e componenti riutilizzabili e disponibili al pubblico, facilmente installabili tramite un repository online, con gestione delle versioni e delle dipendenze. Tali librerie sono accessibili attraverso uno strumento command-line dedicato.

I moduli principali sono:

- Express, un framework di sviluppo web [45];
- Socket.io, un componente server-side di due WebSocket components comuni [46];
- Mongodb, che fornisce API per l'omonimo database [47];
- Redis, un sistema di caching [48].

È possibile per chiunque realizzare e pubblicare la propria libreria su Node Package Manager (npm).

In conclusione, Node.js rappresenta una soluzione moderna e efficace per lo sviluppo di applicazioni server-side, disponendo di una vasta scelta di librerie disponibili attraverso Node Package Manager (npm) ed una community

attiva. La sua architettura event-driven e le performance elevate lo rendono una scelta ideale per molte tipologie di applicazioni web moderne.

WoT

Il Web of Things (WoT) rappresenta un paradigma architetturale che estende i principi del WWW all'IOT, permettendo l'interoperabilità tra dispositivi eterogenei attraverso standard web consolidati [49]. L'implementazione analizzata trasforma sensori di qualità dell'aria in *Things* web-accessibili, seguendo le specifiche W3C e utilizzando l'ontologia Smart Applications REFerence ontology (SAREF) [50]. Ogni sensore fisico viene quindi trasformato in una risorsa web identificabile univocamente, accessibile attraverso protocolli HTTP standard e descritta mediante metadati strutturati secondo il vocabolario WoT. Questa approccio garantisce l'interoperabilità semantica e l'integrazione seamless con applicazioni web esistenti.

L'architettura utilizza le Thing Descriptions (TD) come elemento centrale, conformi alla specifica W3C WoT Thing Descriptions v1.1 [49]. La struttura della TD include diversi componenti fondamentali che definiscono l'identità, le capacità e le modalità di interazione del dispositivo. Ogni sensore viene identificato univocamente attraverso URI strutturati (urn:sensor:air-quality:${sensor_id}) e classificato semanticamente usando il tipo saref:Sensor.

Il contesto semantico integra il vocabolario WoT standard con l'ontologia SAREF, facilitando l'interpretazione automatica delle funzionalità del dispositivo. Le proprietà modellano parametri ambientali con schemi di dati precisi che includono tipo, unità di misura e range di validità.

La directory WoT (`/wot/things`) supporta il discovery automatico dei dispositivi, fornendo informazioni sommarie su identità, stato operativo e ultimo contatto. Le notifiche proattive via WebSocket mantengono sincronizzate le applicazioni client senza polling periodico.

L'implementazione rispetta le specifiche W3C WoT [51], utilizzando vocabolari semantici standard e implementando gli elementi necessari. L'uso

dell'ontologia SAREF [50] aggiunge un layer semantico che facilita l'interoperabilità con sistemi di building automation e smart city.

La separazione tra modello logico (Thing Descriptions) e implementazione fisica (sensori hardware) facilita l'evoluzione dell'architettura e l'integrazione di nuove tipologie di dispositivi, confermando la validità del paradigma WoT per applicazioni IOT di larga scala e garantiscono compatibilità con toolchain WoT esistenti e scalabilità orizzontale [52].

2.2.2 Python

Python è un linguaggio di programmazione di alto livello, interpretato e multi-paradigma. È stato creato da Guido Van Rossum e rilasciato per la prima volta nel 1991 [53]. Il nome deriva dalla serie televisiva britannica "Monty Python's Flying Circus", riflettendo l'approccio creativo che caratterizza l'intera filosofia del linguaggio.

La filosofia di Python è codificata nel famoso "Zen of Python" di Tim Peters, un insieme di principi guida che enfatizzano la leggibilità, la semplicità e la qualità del codice. Tra questi principi spicca il motto *"Beautiful is better than ugly"* e *"Simple is better than complex"*, che hanno influenzato profondamente la progettazione del linguaggio e la sua evoluzione nel corso degli anni [54].

Uno degli elementi che contraddistinguono Python rispetto ad altri linguaggi di programmazione è l'assenza di parentesi graffe per delimitare i blocchi di codice, ma bensì l'utilizzo dell'indentazione stessa. Tale caratteristica ha promosso l'apprendimento del linguaggio a programmatori alle prime armi, che hanno potuto così concentrarsi maggiormente sul contenuto del codice grazie alla sua forma più pulita e leggibile.

Il supporto di diversi modelli di programmazione da parte di Python lo ha reso un linguaggio versatile e popolare. Si possono infatti scegliere approcci procedurali per script semplici, orientati agli oggetti per applicazioni più complesse o funzionali per elaborazioni matematiche più avanzate. Que-

sta flessibilità permette di scegliere il paradigma migliore per il problema specifico da risolvere.

Una peculiarità del linguaggio Python è il concetto dove tutto è un oggetto, inclusi numeri, stringhe, funzioni e persino le classi stesse. Tale uniformità concettuale semplifica notevolmente il modello mentale necessario per comprendere il linguaggio, facilitando quindi l'apprendimento di concetti avanzati.

Il sistema di tipizzazione è forte (strong typing) ed eseguito a run-time (dynamic typing), evitando così errori comuni dipesi dalle operazioni implicite tra tipi incompatibili. È stato introdotto il supporto per type hints, permettendo di annotare esplicitamente i tipi delle variabili e dei parametri delle funzioni. Queste annotazioni, pur non influenzando l'esecuzione del programma, migliorano significativamente la leggibilità del codice e abilitano strumenti di analisi statica per la rilevazione precoce di errori di tipo.

Python gestisce in autonomia la memoria allocata grazie ad un sistema garbage collector. Questo solleva il programmatore dalla necessità di gestire manualmente l'allocazione e la deallocazione della memoria, riducendo drasticamente i bug legati alla gestione delle risorse. Il garbage collector rappresenta un utile strumento che, sfruttando il reference counting ed integrato con un rilevatore di cicli, permette di individuare e segnalare riferimenti circolari.

Anche se spesso viene inteso come linguaggio interpretato, Python in realtà non converte direttamente il codice sorgente in linguaggio macchina, ma passa prima una fase di pre-compilazione bytecode, evitando di reinterpretare integralmente il codice e migliorando le prestazioni.

L'ecosistema di Python è arricchito dal Python Package Index (PyPI), un repository centrale che ospita centinaia di migliaia di pacchetti di terze parti [55]. Viene così praticamente ricoperto ogni dominio applicativo, dalla sviluppo web all'intelligenza artificiale, dall'analisi dei dati alla computer vision.

Il sistema di gestione dei pacchetti, principalmente attraverso pip, rende

estremamente semplice l'installazione e la gestione delle dipendenze. L'introduzione di strumenti come virtualenv e, più recentemente, pipenv e poetry, ha ulteriormente migliorato la gestione degli ambienti di sviluppo isolati, permettendo di evitare conflitti tra diverse versioni delle librerie. Questo risulta molto comodo qualora si lavori a progetti che utilizzando versioni differenti delle dipendenze.

La libreria standard provvede un ricco numero di moduli base, come quelli per le operazioni su file system, networking, regex, database, threading, e molto altro. Questa completezza riduce la necessità di dipendenze esterne per molte operazioni comuni.

Uno dei campi in cui il linguaggio è maggiormente usato è quello scientifico, dove librerie come NumPy e SciPy hanno trasformato Python in uno strumento fondamentale per il calcolo numerico, fornendo strutture dati efficienti e algoritmi ottimizzati per operazioni matematiche complesse. Pandas ha rivoluzionato l'analisi dei dati, offrendo strutture dati potenti per la manipolazione e l'analisi di dataset strutturati.

Il seguente esempio 2.4 mostra come sia possibile generare misurazioni aleatorie in Python:

Listing 2.4: Generazione di misurazioni aleatorie in Python

```
1  import random
2  import numpy as np
3  from datetime import datetime, timedelta
4
5  class MeasurementGenerator:
6    def __init__(self):
7      self.sensors = {
8        'temperature': {'min': 15, 'max': 35, 'unit': 'C'},
9        'humidity': {'min': 30, 'max': 90, 'unit': '%'},
10       'pressure': {'min': 990, 'max': 1030, 'unit': 'hPa
           '}
11     }
12
```

```
13  def generate_measurement(self, sensor_type):
14      config = self.sensors[sensor_type]
15      value = random.uniform(config['min'], config['max
            '])
16      return {
17        'timestamp': datetime.now().isoformat(),
18        'sensor': sensor_type,
19        'value': round(value, 2),
20        'unit': config['unit']
21      }
22
23  def generate_time_series(self, sensor_type, hours=24,
       interval_minutes=60):
24    measurements = []
25    start_time = datetime.now() - timedelta(hours=hours)
26
27    for i in range(0, hours * 60, interval_minutes):
28      timestamp = start_time + timedelta(minutes=i)
29      config = self.sensors[sensor_type]
30
31      base_value = (config['min'] + config['max']) / 2
32      daily_variation = 5 * np.sin(2 * np.pi * i / (24 *
            60))
33      noise = random.gauss(0, 1)
34      value = base_value + daily_variation + noise
35
36      measurements.append({
37        'timestamp': timestamp.isoformat(),
38        'sensor': sensor_type,
39        'value': round(np.clip(value, config['min'],
              config['max']), 2),
40        'unit': config['unit']
41      })
```

```
42
43      return measurements
44
45  if __name__ == '__main__':
46    generator = MeasurementGenerator()
47
48    single_measurement = generator.generate_measurement('
        temperature')
49    print(f"Single: {single_measurement}")
50
51    time_series = generator.generate_time_series('
        temperature', hours=12)
52    print(f"Generated {len(time_series)} measurements")
53    print(f"First: {time_series[0]}")
54    print(f"Last: {time_series[-1]}")
```

Eseguendo il codice riportato nell'esempio 2.4 si può ottenere un output come riportato in lista 2.5:

Listing 2.5: Output di esempio ottenuto dalla generazione di misurazioni aleatorie in Python

```
1   Single: {'timestamp': '2025-08-28T18:55:41.327822', '
      sensor': 'temperature', 'value': 30.71, 'unit': 'C'}
2   Generated 12 measurements
3   First: {'timestamp': '2025-08-28T05:55:41.327850', '
      sensor': 'temperature', 'value': 25.79, 'unit': 'C'}
4   Last: {'timestamp': '2025-08-28T22:55:41.327850', '
      sensor': 'temperature', 'value': 26.01, 'unit': 'C'}
```

Anche nello sviluppo web, Python rimane una delle scelte preferire dagli sviluppatori, i quali possono usare framework come Django e Flask [56, 57] per realizzare le proprie applicazioni web. Fra i due, Flask risulta più minimale e flessibile per progetti che richiedono un controllo più granulare dell'ar-

chitettura, mentre Django include tutto il necessario per sviluppare sistemi complessi.

Il seguente esempio 2.6 mostra come sia possibile realizzare un semplice server usando Flask:

Listing 2.6: Server Flask base in Python

```
from flask import Flask, jsonify, request

app = Flask(__name__)

users = [
  {"id": 1, "name": "John", "email": "john@example.com"},
  {"id": 2, "name": "Jane", "email": "jane@example.com"}
]

@app.route('/')
def home():
  return {"message": "Flask server running", "users_count
     ": len(users)}

@app.route('/api/users', methods=['GET'])
def get_users():
  return jsonify(users)

@app.route('/api/users/<int:user_id>', methods=['GET'])
def get_user(user_id):
  user = next((u for u in users if u["id"] == user_id),
     None)
  if user:
    return jsonify(user)
  return jsonify({"error": "User not found"}), 404

@app.route('/api/users', methods=['POST'])
def create_user():
```

```
27    data = request.get_json()
28    if not data or 'name' not in data or 'email' not in
          data:
29      return jsonify({"error": "Name and email required"}),
            400
30
31    new_user = {
32      "id": max([u["id"] for u in users]) + 1,
33      "name": data["name"],
34      "email": data["email"]
35    }
36    users.append(new_user)
37    return jsonify(new_user), 201
38
39  if __name__ == '__main__':
40    app.run(debug=True, port=5000)
```

Eseguendo il codice riportato nell'esempio 2.6 si può ottenere un output come riportato in lista 2.7:

Listing 2.7: Output di esempio ottenuto dall'interazione con le rotte disposte dal server web in Flask

```
1   // Main url
2   curl 127.0.0.1:5000
3   {
4     "message": "Flask server running",
5     "users_count": 2
6   }
7
8   // Get all users
9   curl 127.0.0.1:5000/api/users
10  [
11    {
12      "email": "john@example.com",
```

```
13      "id": 1,
14      "name": "John"
15    },
16    {
17      "email": "jane@example.com",
18      "id": 2,
19      "name": "Jane"
20    }
21  ]
22
23  // Show informations about user #1
24  curl 127.0.0.1:5000/api/users/1
25  {
26    "email": "john@example.com",
27    "id": 1,
28    "name": "John"
29  }
30
31  // Show informations about user #2
32  curl 127.0.0.1:5000/api/users/2
33  {
34    "email": "jane@example.com",
35    "id": 2,
36    "name": "Jane"
37  }
38
39  // Create new user
40  curl -X POST http://127.0.0.1:5000/api/users \
41    -H "Content-Type: application/json" \
42    -d '{"name": "Mario Rossi", "email": "mario.
        rossi@example.com"}'
43  {
44    "email": "mario.rossi@example.com",
```

```
45      "id": 3,
46      "name": "Mario Rossi"
47    }
```

L'approccio "pythonic" enfatizza la leggibilità del codice e la rapidità di sviluppo, permettendo di creare prototipi funzionali in tempi molto brevi e di scalare gradualmente verso applicazioni enterprise.

Sempre nel campo scientifico, Python è diventato il linguaggio de-facto per data science e machine learning, grazie alle librerie specializzate. TensorFlow e PyTorch, ad esempio, hanno reso accessibili le tecniche di deep learning senza il bisogno di avere nozioni approfondite di matematica avanzata.

Un altro campo in cui il linguaggio è fortemente utilizzato è quello delle DevOps e dell'amministrazione di sistemi, grazie alla capacità di interagire facilmente con il sistema operativo, nel processare file di testo, interfacciarsi con database e API REST. Viene infatti scelto per automatizzare processi ripetitivi, creare pipeline di elaborazione dati ed integrazione di sistemi eterogenei, settori dove la rapidità di sviluppo e la manutenibilità del codice sono cruciali.

Python rimane uno dei linguaggi più popolari e trasversali, godendo di una forte comunità che ne segue gli sviluppi e lo aggiorna in modo continuativo, con rilasci annuali che introducono nuove funzionalità e miglioramenti delle performance. L'adozione crescente in settori come l'intelligenza artificiale, l'Internet of Things (IOT) e l'edge computing suggerisce che Python rimarrà rilevante e in continua evoluzione, adattandosi alle esigenze di un panorama tecnologico in rapido cambiamento.

2.2.3 Kafka

Apache Kafka è una piattaforma di streaming distribuito basato su Java e Scala, progettata per gestire flussi di dati in tempo reale su larga scala [58]. Sviluppato originariamente da LinkedIn e successivamente donato alla Apa-

che Software Foundation nel 2011, Kafka ha acquisito presto popolarità nel panorama della messaggistica e del processing di eventi in sistemi distribuiti [59].

La filosofia di Kafka si basa sul concetto di event streaming, dove i dati vengono trattati come una sequenza immutabile di eventi che possono essere pubblicati, memorizzati e processati in tempo reale [60]. Questo paradigma si discosta significativamente dai tradizionali message broker, offrendo persistenza duratura, elevata throughput e capacità di replay dei messaggi, caratteristiche fondamentali per architetture moderne basate su microservizi e event-driven architecture.

L'architettura distribuita di Kafka consente la gestione in tempo reale di grandi volumi di dati, rendendolo una valida scelta in campi quali analytics, monitoring, fraud detection e real-time recommendation systems [61]. Il sistema organizza i dati in topic, entità logiche che rappresentano categorie di messaggi correlati [58]. Ogni topic è suddiviso in partition, unità fisiche di parallelizzazione che permettono la distribuzione del carico e la scalabilità orizzontale del sistema.

Il modello di persistenza di Kafka utilizza un commit log distribuito, dove ogni messaggio viene assegnato a un offset sequenziale all'interno di una partizione [59]. Questa struttura garantisce l'ordinamento dei messaggi all'interno di ciascuna partizione e permette un accesso efficiente, sia sequenziale che random, ai dati storici. La persistenza è implementata attraverso segment files ottimizzati per operazioni append-only, minimizzando la latenza di scrittura e massimizzando la throughput.

I broker Kafka formano un cluster distribuito che gestisce la replica dei dati attraverso il meccanismo di leader-follower replication [60]. Ogni partizione ha un broker leader che gestisce tutte le operazioni di lettura e scrittura, mentre i follower mantengono copie sincronizzate dei dati. Il sistema di elezione del leader, basato su Apache ZooKeeper (e successivamente su KRaft metadata management), garantisce alta disponibilità e fault tolerance.

Il modello publish-subscribe di Kafka si basa sull'interazione tra produt-

tori, che pubblicano messaggi sui topic, e consumatori, che leggono e processano questi messaggi [61]. I produttori possono configurare diverse strategie di partitioning, utilizzando chiavi di partizionamento per garantire che messaggi correlati vengano sempre inviati alla stessa partizione, preservando l'ordinamento temporale.

Kafka supporta diverse semantiche di delivery [58]: la semantica at-least-once garantisce che ogni messaggio venga consegnato almeno una volta, ma può comportare duplicazioni; la semantica at-most-once assicura l'assenza di duplicati ma può causare perdite di messaggi in caso di failure; la semantica exactly-once, introdotta nelle versioni più recenti, combina transazioni e idempotenza per garantire elaborazione esatta dei messaggi.

I consumer groups rappresentano un meccanismo di parallelizzazione del consumo di messaggi [59]. Ogni consumer all'interno di un gruppo riceve messaggi da un sottoinsieme delle partizioni del topic, permettendo scalabilità orizzontale del processing. Il consumer group rebalancing automatico redistribuisce le partizioni tra i consumatori attivi, favorendo load balancing dinamico e fault tolerance.

Kafka Streams costituisce una libreria client per il processing di stream di dati che elimina la necessità di framework esterni per elaborazioni in tempo reale [60]. Questa libreria implementa il paradigma di stream processing attraverso topology di trasformazioni che possono includere operazioni di filtering, mapping, aggregation e joining tra stream differenti.

La scalabilità orizzontale di Kafka è limitata principalmente dal numero di partizioni per topic, che determina il massimo grado di parallelismo achievable [58]. L'aggiunta di broker al cluster permette di aumentare la capacità di storage e processing, ma richiede careful planning del numero di partizioni e della strategia di replica. Il processo di partition reassignment può essere utilizzato per bilanciare il carico tra broker esistenti e nuovi.

Kafka si rivela particolarmente efficace in architetture event-driven dove la decoupling tra componenti e la capacità di replay degli eventi sono requisiti fondamentali [60]. Pattern architetturali come Event Sourcing, Command

Query Responsibility Segregation (CQRS) e Saga pattern sono indicati per tale strumento.

I casi d'uso tipici di Kafka includono real-time analytics, log aggregation, metrics collection, stream processing per machine learning e data pipeline per data lake e data warehouse [61]. La capacità di Kafka di fungere sia da message broker che da storage system può essere sfruttata in architetture dove i dati devono essere processati da multiple applicazioni con diverse temporal requirements.

2.2.4 MongoDB

MongoDB è uno dei database NoSQL orientati ai documenti più noti ed utilizzati [62]. Sviluppato inizialmente da 10gen (ora MongoDB Inc.) nel 2009, questo database ha rivoluzionato il modo in cui gli sviluppatori approcciano la persistenza dei dati, offrendo un'alternativa flessibile e scalabile ai tradizionali database relazionali [63].

La filosofia alla base di MongoDB si discosta significativamente dal paradigma relazionale in favore di un modello basato su documenti Binary JSON (BSON) che permette una maggiore flessibilità nella strutturazione dei dati [64]. Questa caratteristica consente agli sviluppatori di memorizzare oggetti complessi e annidati senza la necessità di normalizzazione, tipica dei database SQL tradizionali.

L'architettura di MongoDB si avvale di un approccio document-oriented, dove ogni record è rappresentato come un documento flessibile che può contenere campi di diversi tipi di dati [65]. Questa struttura elimina la rigidità dello schema fisso, permettendo l'evoluzione dinamica delle strutture dati durante il ciclo di vita dell'applicazione.

Il sistema di indicizzazione di MongoDB supporta indici di varia natura quali composti, testuali, geospaziali ed altro, offrendo prestazioni ottimizzate per diversi tipi di query [66]. Gli indici vengono implementati utilizzando strutture dati B-tree per ottenere operazioni di ricerca efficienti anche su grandi volumi di dati. Inoltre, MongoDB supporta indici parziali e Time To

Live (TTL), consentendo una gestione automatica dei dati basata su criteri temporali.

La gestione della memoria in MongoDB utilizza il memory mapping per migliorare le prestazioni di lettura e scrittura [67]. Il WiredTiger storage engine ottiene ciò attraverso la compressione dei dati e il controllo della concorrenza a livello di documento.

MongoDB utilizza un linguaggio di query basato su JavaScript che si integra naturalmente con gli ambienti di sviluppo web moderni [65]. Le query vengono espresse attraverso documenti JSON che specificano i criteri di ricerca, proiezione e ordinamento. Questa sintassi risulta particolarmente intuitiva per gli sviluppatori familiari con JavaScript e altri linguaggi di programmazione moderni.

L'Aggregation Framework rappresenta uno strumento potente per l'elaborazione e l'analisi dei dati, offrendo funzionalità comparabili a quelle dei sistemi SQL attraverso pipeline di trasformazione [66]. Le operazioni di aggregazione includono filtering, grouping, sorting, reshaping e computational operations, permettendo analisi complesse direttamente a livello di database.

Le prestazioni di MongoDB dipendono significativamente dalla progettazione dello schema e dalla strategia di indicizzazione adottata [62]. A differenza dei database relazionali, dove la normalizzazione è spesso prioritaria, in MongoDB è frequentemente vantaggioso denormalizzare i dati per ottimizzare le performance di lettura.

MongoDB si rivela particolarmente efficace in scenari caratterizzati da rapido sviluppo, schema evolutivo e necessità di scalabilità orizzontale [63]. Applicazioni web, sistemi di gestione contenuti, piattaforme di social media e applicazioni IOT rappresentano casi d'uso ideali per questo database. Tuttavia, per applicazioni che richiedono transazioni ACID complesse o relazioni complesse tra entità, i database relazionali tradizionali possono risultare più appropriati.

Il seguente esempio 2.8 presenta quale query di esempio in un database Mongodb:

Listing 2.8: Query MongoDB

```
// Document example
{
  "_id": ObjectId("..."),
  "station": "Bologna Piazza Maggiore",
  "coordinates": {"lat": 44.4939, "lng": 11.3426},
  "pm25": 35.2,
  "pm10": 42.8,
  "no2": 48.5,
  "o3": 62.1,
  "temperature": 22.5,
  "humidity": 65,
  "aqi": 78,
  "level": "Moderate",
  "timestamp": ISODate("2025-08-23T14:30:00Z")
}

// Find stations with PM2.5 above 25 micrograms per
   cube meter
db.air_quality.find({"pm25": {$gt: 25}})

// Find stations with good air quality level in
   previous 24 hours
db.air_quality.find({
  "level": "Good",
  "timestamp": {$gte: new Date(Date.now() -
     24*60*60*1000)}
})

// Daily average of PM2.5 per station
db.air_quality.aggregate([
  {$match: {"timestamp": {$gte: ISODate("2025-08-23T00
     :00:00Z")}}},
```

```
29      {$group: {_id: "$station", avg_pm25: {$avg: "$pm25"},
            measurements: {$sum: 1}}},
30      {$sort: {avg_pm25: -1}}
31    ])
```

2.3 Deployment

In questa sezione verranno descritti gli strumenti adoperati nel deployment, quali Docker come ambiente di containerizzazione e Docker compose come strumento d'orchestrazione dei container realizzati dal primo.

2.3.1 Docker

Docker è un progetto open-source, il cui sviluppo è a cura della Docker Inc, divenuto uno dei principali attori nel campo dello sviluppo e deployment di applicazioni basate su container Linux [68]. Questa tecnologia di containerizzazione permette di incapsulare un'applicazione insieme a tutte le sue dipendenze in un contenitore leggero e portabile, garantendo che l'applicazione funzioni in modo consistente fra differenti sistemi operativi. Questo container rappresenta infatti un sistema isolato dove l'applicazione dispone di tutte le librerie necessarie alla sua esecuzione.

La filosofia alla base di Docker è fondata sul concetto di "write once, run anywhere", permettendo agli sviluppatori di creare applicazioni che possono essere eseguite senza modifiche sui sistemi operativi che supportano Docker [69]. I container Docker condividono il kernel del sistema operativo host, rendendoli più efficienti rispetto alle tradizionali macchine virtuali in termini di utilizzo delle risorse e tempi di avvio. Non è quindi necessario avere installati sul proprio sistema operativo strumenti usati dall'applicazione poiché saranno disponibili direttamente all'interno del container.

I container Docker vengono costruiti sulla base di file Dockerfile, i quali riportano tutte le istruzioni necessarie a realizzare l'immagine. Un'imma-

gine Docker è un pacchetto completo e immutabile che contiene l'essenziale per eseguire un'applicazione: codice, librerie, dipendenze, configurazioni. Funziona come uno "stampo digitale" che garantisce la riproducibilità dell'ambiente software, eliminando i problemi di compatibilità tra diversi sistemi.

Le immagini sono costruite a livelli sovrapposti, dove ogni strato aggiunge componenti specifici, rendendo efficiente sia l'archiviazione che la condivisione. Questo risulta ottimale quando immagini differenti necessitano di elementi in comune, condividendo fra loro livelli e riducendo lo spazio utilizzato, gli eventuali tempi di download e di compilazione degli stessi. Una volta creata, l'immagine rimane invariata e può essere utilizzata per generare molteplici container identici, che rappresentano le istanze in esecuzione dell'applicazione.

Docker permette anche la realizzazione di volumi, per disporre di dati persistenti, e di reti, per mettere in comunicazione più container fra loro. I volumi trovano impiego qualora sia necessario mantenere dati una volta arrestata l'esecuzione di un container, come per i database. Le reti sono usate invece per sistemi più complessi in cui due o più container devono poter comunicare. Un container può essere disposto su una o più reti virtuali per interagire con altri container. Avere più reti può risultare utile nel caso in cui determinati container abbiano bisogno di vedere solo altri container su certi livelli, distinguendo ad esempio servizi di back-end e di front-end.

Nel seguente esempio viene mostrato come sia possibile realizzare un container con l'ultima versione disponibile di Python per eseguire il proprio script `app.py` e le relative dipendenze definite nel file `requirements.txt` 2.9:

Listing 2.9: Dockerfile Python

```
1 FROM python:latest
2 WORKDIR /app
3 COPY requirements.txt .
4 RUN pip install --no-cache-dir -r requirements.txt
5 COPY . .
```

```
6   EXPOSE 8000
7   CMD ["python", "app.py"]
```

2.3.2 Docker compose

Docker Compose è uno strumento che semplifica la gestione di applicazioni composte da più container Docker. Attraverso un singolo file di configurazione YAML, permette di definire tutti i servizi che compongono l'applicazione, specificando come devono comunicare tra loro, quali porte esporre, quali volumi condividere e come configurare le reti.

Il vantaggio di Docker Compose è la capacità di trasformare la complessità di gestire assieme un insieme di più container Docker. Ad esempio, è possibile avviare tutti i container, con relativi volumi e reti, con un semplice 'docker-compose up' piuttosto che caricare manualmente ogni container, volume e rete. Con questo singolo comando è infatti possibile avviare simultaneamente i servizi necessari all'applicazione, creando automaticamente le connessioni e le dipendenze specificate nel file di configurazione.

Docker Compose trova impiego principalmente negli ambienti di sviluppo e testing, dove permette di replicare facilmente le architetture di applicazioni complesse. Basandosi su un file `docker-compose.yml`, è possibile ricreare l'intero stack applicativo con un comando, agevolando lo sviluppo. In aggiunta, gestisce automaticamente aspetti come reti isolate, volumi persistenti e variabili d'ambiente, rendendo trasparente la complessità dell'orchestrazione multi-container.

Nel seguente esempio riproduciamo quanto illustrato in 2.9 ma utilizzando invece direttamente Docker Compose 2.10:

Listing 2.10: Docker Compose Python

```
1   version: '3.8'
2
3   services:
4     python-app:
```

```
5       image: python:latest
6       ports:
7         - "8000:8000"
8       volumes:
9         - .:/app
10      working_dir: /app
11      environment:
12        - ENV=test
13      command: >
14        sh -c "pip install -r requirements.txt 2>/dev/
              null || true &&
15              python app.py"
16      restart: unless-stopped
```

Capitolo 3

Sviluppo dell'elaborato di progetto

In questo capitolo verrà descritto il progetto sviluppato, in particolare l'architettura generale del sistema, la descrizione dei singoli componenti e l'applicazione front-end.

3.1 Architettura generale del sistema

In questa sezione verrà descritta l'architettura generale del sistema, analizzando i componenti principali, le loro interazioni e le scelte progettuali adottate.

3.1.1 Architettura a microservizi

Il progetto AirQualityInsight è stato realizzato utilizzando un'insieme di microservizi.

I microservizi rappresentano un approccio architetturale per lo sviluppo di applicazioni software che prevede la scomposizione di un sistema monolitico, dove tutti i processi sono interdipendenti e funzionano come un singolo servizio, in un insieme di servizi indipendenti, ognuno dei quali implementa

una specifica funzionalità di business [70]. Ogni microservizio viene eseguito nel proprio processo e comunica attraverso API REST (Representational State Transfer), event streaming e broker di messaggistica asincrona [71].

Le caratteristiche distintive di questa architettura includono l'autonomia di deployment, la responsabilità su specifici domini di business, la gestione decentralizzata dei dati e la possibilità di utilizzare tecnologie eterogenee per diversi servizi. Utilizzare i microservizi piuttosto che un sistema monolitico presenta diversi vantaggi: semplifica l'aggiornamento del codice; permette di implementare nuove funzionalità senza modificare l'intera architettura applicativa; aumenta il grado di libertà nella scelta sulle tecnologie e linguaggi di programmazione da adottare, che possono così essere differenti per ciascun componente; favorisce la scalabilità orizzontale e consente di dimensionare indipendentemente ogni servizio in base alle specifiche esigenze di carico [72], eliminando gli sprechi e riducendo i costi derivanti dalla necessità di scalare l'intera applicazione quando solo una specifica funzione richiede risorse aggiuntive.

Tuttavia, l'adozione dei microservizi introduce anche sfide significative, tra cui la complessità nella gestione della comunicazione inter-servizio, la necessità di implementare pattern di resilienza e la gestione della consistenza dei dati in un ambiente distribuito [73]. La governance e il monitoraggio di sistemi distribuiti richiedono inoltre strumenti e pratiche specifiche per garantire osservabilità e debugging efficaci [74].

In conclusione, questo approccio offre vantaggi nella progettazione di sistemi complessi, rendendo più chiara la suddivisione dei vari aspetti del dominio applicativo, e nella realizzazione degli stessi, facilitando la resilienza del sistema.

3.1.2 Architettura del sistema

L'architettura del sistema AirQualityInsight si basa su 5 componenti principali, come illustrato in figura 3.1: i sensori atti a realizzare misurazioni

simulate, il broker di messaggistica, il server per la fruizione degli stessi, la dashboard front-end ed infine il database non relazionale.

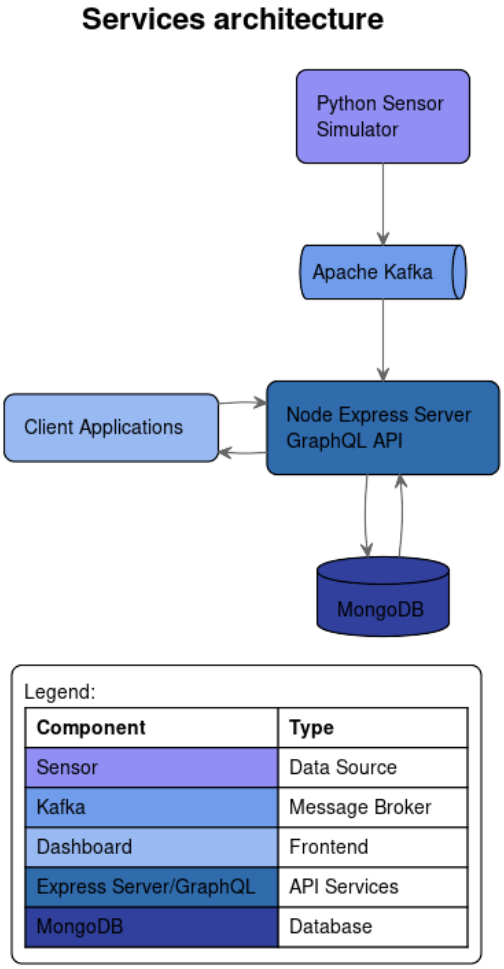


Figura 3.1: Architettura generale del sistema

Di seguito vengono esplicitate le varie responsabilità per ognuno dei servizi precedentemente elencati.

- Sensor service: questo servizio simula l'esercizio di un insieme di sensori, i quali registrano, con cadenza regolare, le misurazioni della qualità dell'aria. Ogni sensore è provvisto di un id univoco, un nome, una posizione geografica (le coordinate della sua collocazione) ed un indirizzo

IP. Tali misurazioni vengono inviate al broker dei messaggi in modo che vengano poi trasmesse ai servizi in ascolto.

- API service: questo servizio consuma dal broker le misurazioni, le salva sul database non relazione e le rende disponibili attraverso API.
- Dashboard service: questo servizio fornisce un'interfaccia grafica tramite cui è possibile consultare la mappa interattiva, aggiornata in tempo reale con i dati forniti dal server API, e le relative tabelle.

3.2 Sensor service

In questa sezione verrà descritto il servizio che simula il sensore e la relativa generazione di misurazioni fittizie sulla qualità dell'aria.

3.2.1 Modello sensore

Il sensore è dotato di un insieme di proprietà specifiche che ne determinano il funzionamento. Queste proprietà sono:

- Id (`sensor_id`): stringa, identificatore univoco del sensore.
- Nome (`name`): stringa, nome del sensore.
- Posizione (`location`): oggetto, posizione in cui è collocato il sensore. Si tratta di un oggetto composto dal tipo (in questo caso `Point`) e dalle coordinate, longitudine e latitudine, ambo valori numerici a virgola mobile (`double`).
- IP (`ip`): stringa, indirizzo ip del sensore.
- Attivo (`active`): booleano, indica se il sensore è attivo (`true`) oppure no (`false`).
- Ultima misurazione (`last_seen`): data, ultima volta che il sensore ha registrato una misurazione.

3.2.2 Generazione pseudo-misurazioni

La classe `AirQualitySensor` è dotata di un metodo `generate_reading` che produce misurazioni simulate. Ad ogni sensore viene fornita la configurazione presentata nel listato 3.1 per la generazione delle misurazioni.

Listing 3.1: Configurazione sensore

```
SENSOR_CONFIG = {
  'sampling_rate': 10,            # seconds (def. 60)

  # Sensor ranges
  'temperature_range': (-15, 35), # Celsius degrees
  'humidity_range': (30, 100),    # %
  'pressure_range': (980, 1020),  # hPa
  'voc_range': (0, 3),            # ppm
  'co2_range': (400, 2000),       # ppm
  'pm25_range': (0, 150),         # micrograms/m^3
  'pm10_range': (0, 300),         # micrograms/m^3
  'no2_range': (0, 200),          # micrograms/m^3
  'o3_range': (0, 200),           # micrograms/m^3
  'so2_range': (0, 300),          # micrograms/m^3
}
```

Per mantenere coerenza nei dati e evitare valori troppo discordanti, ogni nuova misurazione generata dal sensore si basa sulla lettura precedente (quando disponibile) e si discosta da essa di una percentuale compresa tra l'1% e il 5%.

Parte del codice Python utilizzato per la generazione di queste misurazioni simulate è riportato nel listato 3.2 che segue.

Listing 3.2: Metodo per la generazione di pseudo-misurazioni

```
def generate_reading(self):
  """Generate a realistic sensor reading with some
     correlation between values
```

```
3       and small random changes (1-5%) from previous
            readings if available"""
4
5       # Apply random change of 1-5% to previous values
6       def random_change(value):
7           percent_change = random.uniform(0.01, 0.05)  #
                1-5%
8           direction = random.choice([-1, 1])  # Increase or
                decrease
9           return value * (1 + direction * percent_change)
10
11      # Gas pollutants with random changes #
12
13      # PM2.5 and PM10 with correlation maintained
14      pm25 = random_change(self.last_reading['pm25'])
15      pm25 = np.clip(pm25, *self.config['pm25_range'])
16
17      pm10 = max(pm25 + random_change(self.last_reading['
            pm10'] - self.last_reading['pm25']), pm25)
18      pm10 = np.clip(pm10, pm25, self.config['pm10_range
            ']][1])
19
20      # Nitrogen dioxide levels
21      no2 = random_change(self.last_reading['no2'])
22      no2 = np.clip(no2, *self.config['no2_range'])
23
24      # Ozone levels
25      o3 = random_change(self.last_reading['o3'])
26      o3 = np.clip(o3, *self.config['o3_range'])
27
28      # Sulfur dioxide levels
29      so2 = random_change(self.last_reading['so2'])
30      so2 = np.clip(so2, *self.config['so2_range'])
```

3.2.3 Distribuzione sensori

La scelta dei sensori è stata fatta relativamente ai punti di maggiore traffico, quali le intersezioni stradali fra le arterie principali e le strade secondarie. Bologna presenta una moltitudine di semafori, luogo dove veicoli fermi in attesa tendono a creare punti di maggiore concentrazione di inquinanti. Come anticipato precedentemente con la query 1.1, sono stati estrapolati tutti i punti che rappresentano un'intersezione stradale di questo tipo. Data l'elevata densità di posizioni rilevate, è stata applicata una selezione sistematica per mantenere un singolo punto per ogni intervallo di spazio regolare. Questa selezione ha permesso anche di gestire le situazioni di affollamento, tipiche di rotonde, nelle quali ogni svincolo è rappresentato come un incrocio, e di centri abitati, nei quali la concentrazione di queste intersezioni risulta più elevato.

Nelle immagini 3.3a e 3.3b viene mostrato come, partendo da un dataset consistente di punti, questi siano stati poi filtrati secondo i criteri precedentemente esposti. È stata implementata una griglia di campionamento con spaziatura di 250 metri, parametro calibrato sperimentalmente per ottenere una densità ottimale di punti. Per fare ciò è stata realizzata una pagina dedicata allo scopo, dato che si tratta di un processo che può essere riprodotto cambiando la distanza fra i punti tramite input numerico. La pagina presenta una parte iniziale dove è possibile scegliere la città (nel nostro caso Bologna) e lo spazio minimo fra un sensore e l'altro. Viene anche indicata la query di estrazione, la stessa citata in precedenza nel listato 1.1. Questa parte d'intestazione è riportata nell'immagine 3.2, dove si possono notare gli elementi citati.

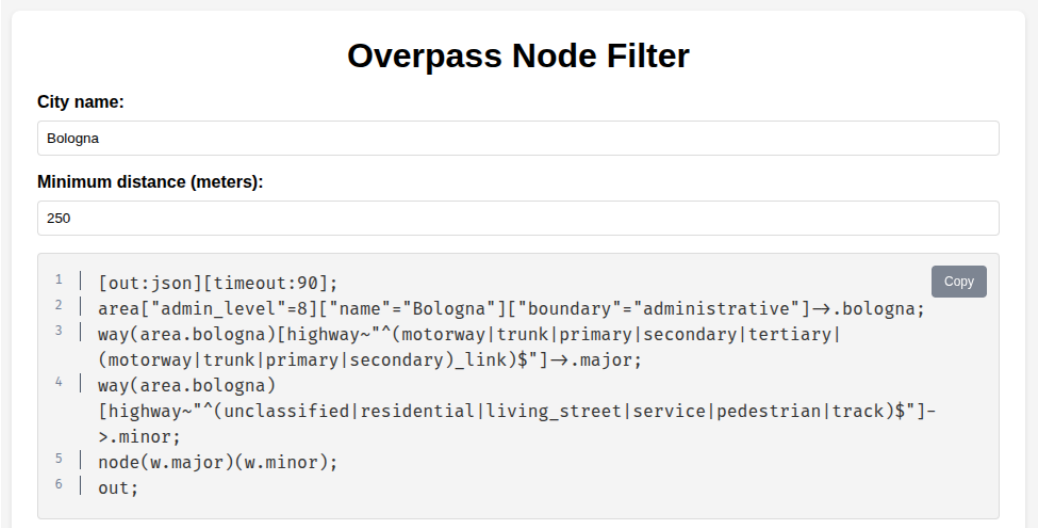


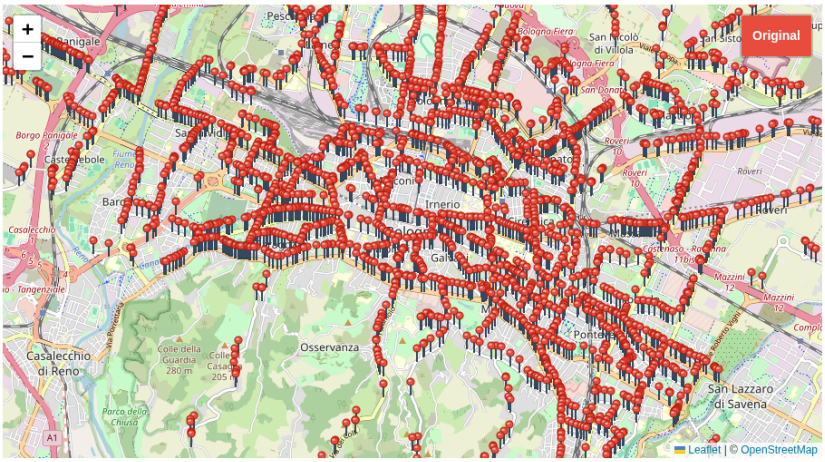
Figura 3.2: Intestazione pagina

Sotto di essa, è disponibile la mappa che mostra i nodi risultanti dalla query. Tali nodi vengono disposti su 3 livelli, che sono: nodi originali, nodi filtrati e nodi differenza. I primi, quelli originali (in rosso, immagine 3.3a), sono ottenuti direttamente dalla query 3.1. I secondi, quelli filtrati (in blu, immagine 3.3b), sono il risultato del trattamento effettuato su di essi, dato dall'elezione di un solo nodo ogni 250 metri 3.2. Gli ultimi infine, quelli di differenza (in verde, immagine 3.3c), sono semplicemente i nodi originali meno quelli filtrati 3.3. La richiesta viene effettuata nel momento in cui viene cliccato il pulsante di invio, che riporta la dicitura *"Submit request"*. Si tratta di una chiamata POST all'indirizzo `https://overpass-api.de/api/interpreter"`, il quale restituisce un JSON in caso di esito positivo.

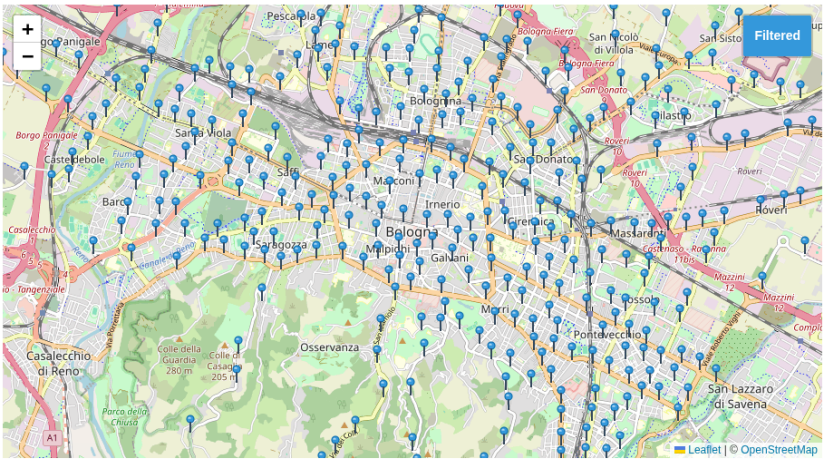
$$O = \text{insieme degli elementi originali} \tag{3.1}$$

$$F = \text{sottoinsieme filtrato, dove } F \subseteq O \tag{3.2}$$

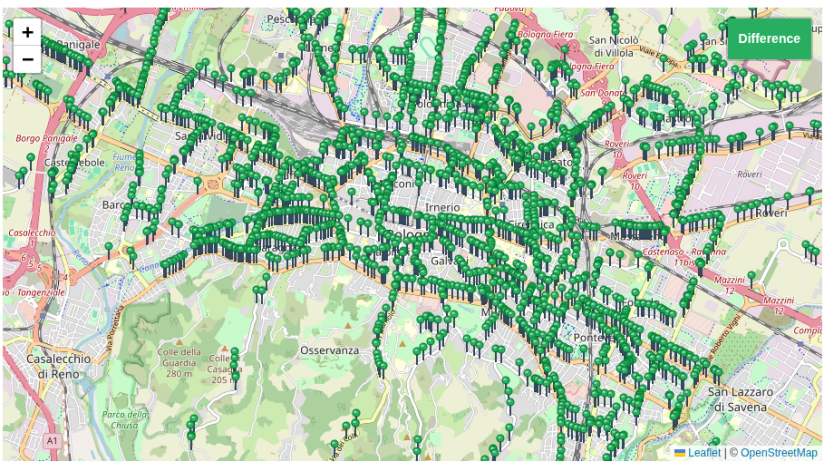
$$D = O \setminus F = \{x \in O : x \notin F\} \tag{3.3}$$



(a) Risultato originale della query



(b) Punti filtrati finali



(c) Punti di differenza

Figura 3.3: Acquisizione punti delle intersezioni stradali

Come ultimo elemento, la parte finale della pagina, esposta nell'immagine 3.4, riporta le statistiche quali il numero di sensori originale, il numero di sensori filtrati e la riduzione espressa come percentuale. Premendo il pulsante con scritto *"Download filtered JSON"* è possibile scaricare il JSON dei sensori filtrati ottenuti. Tale JSON presenta i sensori usati per popolare la mappa dell'applicazione principale.

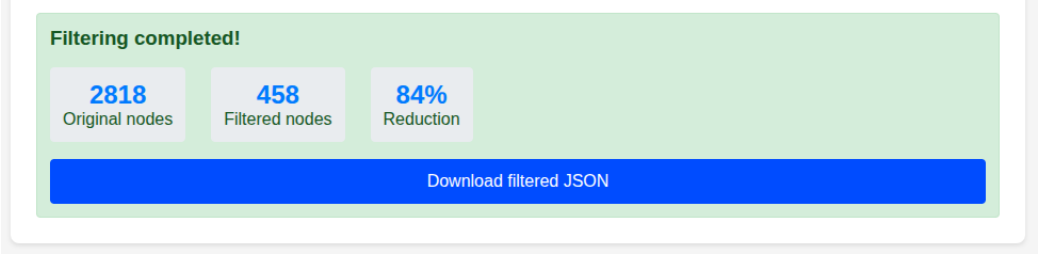


Figura 3.4: Piè di pagina

3.3 API service

In questa sezione verrà descritto il servizio API che riceve le misurazioni simulate della qualità dell'aria dal broker di messaggi, le trasforma ed eroga ai client che lo interrogano.

3.3.1 Implementazione del Server

Il server Node.js fornisce un servizio web per la gestione e l'accesso ai dati delle misurazioni ambientali. Utilizza Express.js per implementare un'interfaccia REST che espone diversi endpoint per le operazioni sui dati. Il server integra MongoDB per la memorizzazione persistente dei dati sensoriali, Apache Kafka per il consumo continuo dei flussi di dati in arrivo dai sensori, e Socket.IO per fornire notifiche push in tempo reale ai client web quando vengono registrate nuove misurazioni o superati i valori di soglia configurati. La configurazione CORS permette l'accesso sicuro da domini diversi, consentendo l'integrazione con applicazioni web distribuite.

3.3.2 Endpoint Disponibili

Di seguito verranno elencati gli endpoint disponibili esposti dalle API. Tutti gli endpoint di seguito elencati restituiscono un JSON ed un codice di stato HTTP che può essere `200 OK` in caso di successo o `500 Internal Server Error` in caso di errore.

Health Check - `GET /health` Endpoint per il monitoraggio dello stato del servizio. Verifica la disponibilità del server e restituisce lo stato operativo. Risponde con un semplice JSON con lo status, il tempo di attività, il numero di sensori attivi e le misurazioni registrate come da esempio nel listato 3.3.

Listing 3.3: Risposta di sucesso per endpoint `health`

```
{
  "status":"healthy",
  "timestamp":"2025-08-30T13:58:44.198Z",
  "uptime":39.198012656,
  "wot_things":459,
  "active_measurements":258
}
```

Recupero Misurazioni - `GET /api/measurements` Endpoint per il recupero delle misurazioni ambientali con supporto per filtri e paginazione. Restituisce le misurazioni archiviate nel database con possibilità di filtraggio per sensore e intervallo temporale.

È possibile utilizzare dei parametri opzionali di query, quali `sensorId` come identificativo del sensore specifico, data di inizio `startDate` e fine `endDate` del periodo di misurazione (in formato ISO 8601). Se specificato `sensorId`, filtra per sensore specifico, mentre se specificati `startDate` e `endDate`, filtra per intervallo temporale. Le misurazioni in risposta sono in ordine crescente per timestamp e con limite massimo di 1000 record per richiesta.

Un esempio di risposta di successo è riportato nel listato 3.4

Listing 3.4: Risposta di sucesso per endpoint `measurements`

```
[
  {
    "_id":"68a4def8c82270d8766481f6",
    "sensor_id":"SENSOR00436",
    "timestamp":"2025-08-19T22:30:48.494Z",
    "temperature":2.64,
    "humidity":75.87,
    "pressure":1019.63,
    "voc":0.647,
    "co2":400,"pm25":55.43,
    "pm10":177.75,
    "no2":132.4,
    "o3":65.45,
    "so2":35.95,
    "__v":0
  },
  ...
]
```

Lista Sensori - `GET /api/sensors` Endpoint per il recupero dell'elenco completo dei sensori registrati nel sistema. Restituisce tutti i sensori configurati con le relative informazioni di localizzazione e configurazione. È possibile utilizzare come parametro opzionale di query `sensorId` come identificativo del sensore specifico, estraendo il solo sensore indicato (se disponibile).

Un esempio di risposta di successo è riportato nel listato 3.5

Listing 3.5: Risposta di sucesso per endpoint `sensors`

```
[
  {
    "location": { "type": "Point", "coordinates":
       [11.3482468, 44.4856545] },
    "_id": "68a4b0b50bb78efcdfbaa8b9",
```

```
5       "sensor_id": "SENSOR00001",
6       "name": "d0d54bf349674398",
7       "ip": "192.168.0.1",
8       "active": true,
9       "last_seen": "2025-08-30T10:21:32.252Z"
10    },
11    ...
12  ]
```

Ultima Misurazione - `GET /api/latest` Endpoint per il recupero della misurazione più recente nel sistema. Restituisce l'ultima misurazione registrata, ordinata per timestamp decrescente.

Un esempio di risposta di successo è riportato nel listato 3.6

Listing 3.6: Risposta di sucesso per endpoint `latest`

```
1  {
2    "_id":"68b421c342b53a3e45108d73",
3    "sensor_id":"SENSOR00238",
4    "timestamp":"2025-08-30T12:19:47.467Z",
5    "temperature":15.42,
6    "humidity":32.71,
7    "pressure":980,
8    "voc":1.937,
9    "co2":417.1,
10   "pm25":16.76,
11   "pm10":131.12,
12   "no2":57.75,
13   "o3":130.33,
14   "so2":112.44,
15   "__v":0
16 }
```

Directory WoT - `GET /wot/things` Endpoint per il discovery automatico dei dispositivi registrati nel gateway. Restituisce un elenco di tutti i Thing disponibili con informazioni sommarie su identità, stato operativo e ultimo contatto. Supporta la notifica proattiva via WebSocket per mantenere sincronizzate le applicazioni client senza polling periodico.

Un esempio di risposta di successo è riportato nel listato 3.7

Listing 3.7: Risposta di successo per endpoint `/wot/things`

```
[
  {
    "id": "SENSOR00001",
    "title": "d0d54bf349674398",
    "description": "Air Quality Sensor monitoring
       environmental parameters at
       11.3482468,44.4856545",
    "base": "http://localhost:3000/wot/things/SENSOR00001
       ",
    "status": "active",
    "lastContact": "2025-08-30T10:21:32.252Z",
    "properties": [
      "temperature",
      "humidity",
      "pressure",
      "voc",
      "co2",
      "pm25",
      "pm10",
      "no2",
      "o3",
      "so2",
      "status",
      "location"
    ],
```

```
23      "actions": ["getLatestMeasurement"]
24    },
25    ...
26  ]
```

Thing Descriptions - `GET /wot/things/{sensorId}` Endpoint per il recupero della Thing Descriptions completa per un sensore specifico. Restituisce la TD conforme alla specifica W3C WoT TD v1.1, includendo il contesto semantico, le proprietà disponibili, le azioni supportate e tutti i metadati del dispositivo. La risposta contiene l'intera struttura JSON della Thing Descriptions con vocabolari Smart Applications REFerence ontology e definizioni complete delle forme di interazione.

Un esempio di risposta di successo è riportato nel listato 3.8

Listing 3.8: Risposta di successo per endpoint `/wot/things/sensorId`

```
1  {
2    "@context": [
3      "https://www.w3.org/2022/wot/td/v1.1",
4      { "saref": "https://saref.etsi.org/core/" }
5    ],
6    "@type": ["Thing", "saref:Sensor"],
7    "id": "urn:sensor:air-quality:SENSOR00001",
8    "title": "d0d54bf349674398",
9    "description": "Air Quality Sensor monitoring
       environmental parameters at 11.3482468,44.4856545",
10   "base": "http://localhost:3000/wot/things/SENSOR00001",
11   "securityDefinitions": { "nosec_sc": { "scheme": "nosec
       " } },
12   "security": ["nosec_sc"],
13   "properties": {
14     "temperature": {
15       "type": "number",
16       "title": "Temperature",
```

```
17        "description": "Ambient temperature in Celsius",
18        "unit": "Celsius degrees",
19        "minimum": -15,
20        "maximum": 35,
21        "readOnly": true,
22        "observable": false,
23        "forms": [
24          {
25            "href": "http://localhost:3000/wot/things/
                SENSOR00001/properties/temperature",
26            "contentType": "application/json",
27            "op": ["readproperty"]
28          }
29        ]
30      },
31      "humidity": {...},
32      "pressure": {...},
33      "voc": {...},
34      "co2": {...},
35      "pm25": {...},
36      "pm10": {...},
37      "no2": {...},
38      "o3": {...},
39      "so2": {...},
40      "status": {
41        "type": "string",
42        "title": "Status",
43        "description": "Current sensor status",
44        "enum": ["active", "inactive"],
45        "readOnly": true,
46        "observable": false,
47        "forms": [
48          {
```

```
49          "href": "http://localhost:3000/wot/things/
               SENSOR00001/properties/status",
50          "contentType": "application/json",
51          "op": ["readproperty"]
52        }
53      ]
54    },
55    "location": {
56      "type": "object",
57      "title": "Location",
58      "description": "Sensor geographical location",
59      "properties": {
60        "latitude": { "type": "number" },
61        "longitude": { "type": "number" }
62      },
63      "readOnly": true,
64      "observable": false,
65      "forms": [
66        {
67          "href": "http://localhost:3000/wot/things/
               SENSOR00001/properties/location",
68          "contentType": "application/json",
69          "op": ["readproperty"]
70        }
71      ]
72    }
73  },
74  "actions": {
75    "getLatestMeasurement": {
76      "title": "Get Latest Measurement",
77      "description": "Get the latest complete measurement
           from this sensor",
78      "forms": [
```

```
79          {
80            "href": "http://localhost:3000/wot/things/
                 SENSOR00001/actions/getLatestMeasurement",
81            "contentType": "application/json",
82            "op": ["invokeaction"]
83          }
84        ]
85      }
86    }
87  }
```

Proprietà Singola - `GET /wot/things/{sensorId}/properties/{propertyName}` Endpoint per l'accesso alle singole proprietà del Thing. Restituisce il valore corrente della proprietà richiesta incapsulato in un oggetto JSON standardizzato. Supporta tutte le proprietà ambientali (`temperature`, `humidity`, `pressure`, `voc`, `co2`, `pm25`, `pm10`, `no2`, `o3`, `so2`) oltre a proprietà di sistema (`status`, `location`).

Un esempio di risposta di successo per la proprietà `temperature` è riportato nel listato 3.9

Listing 3.9: Risposta di successo per proprietà singola

```
1  {
2    "value": 23.5
3  }
```

Tutte le Proprietà - `GET /wot/things/{sensorId}/properties` Endpoint per il recupero di tutte le proprietà del Thing in una singola chiamata. Ottimizza l'efficienza della comunicazione permettendo di recuperare lo stato completo del sensore senza multiple richieste HTTP. Restituisce un oggetto JSON contenente tutte le proprietà con i relativi valori correnti.

Un esempio di risposta di successo è riportato nel listato 3.10

Listing 3.10: Risposta di successo per tutte le proprietà

```
{
  "temperature": { "value": -7.15 },
  "humidity": { "value": 54.52 },
  "pressure": { "value": 1020 },
  "voc": { "value": 1.71 },
  "co2": { "value": 431.7 },
  "pm25": { "value": 36.91 },
  "pm10": { "value": 90.61 },
  "no2": { "value": 133.43 },
  "o3": { "value": 54.66 },
  "so2": { "value": 270.31 },
  "status": { "value": "active" },
  "location": { "value": { "latitude": 11.3482468, "
     longitude": 44.4856545 } }
}
```

Invocazione Azioni - `GET /wot/things/{sensorId}/actions/{actionName}` Endpoint per l'invocazione delle azioni definite nella Thing Descriptions. Attualmente supporta l'azione `getLatestMeasurement` che restituisce l'insieme completo delle misurazioni più recenti del sensore. L'azione viene eseguita immediatamente e restituisce il risultato in formato JSON strutturato con tutti i parametri ambientali misurati.

Un esempio di risposta di successo per l'azione `getLatestMeasurement` è riportato nel listato 3.11

Listing 3.11: Risposta di successo per azione `getLatestMeasurement`

```
{
  "result": {
    "sensor_id": "SENSOR00001",
    "timestamp": "2025-08-30T10:21:39.092Z",
    "temperature": -6.74,
    "humidity": 55.31,
```

```
7       "pressure": 1020,
8       "voc": 1.791,
9       "co2": 421.9,
10      "pm25": 35.67,
11      "pm10": 89.71,
12      "no2": 129.49,
13      "o3": 56.53,
14      "so2": 286.01
15    }
16  }
```

3.4 Dashboard service

Questa sezione sarà dedicata alla descrizione dell'applicazione frontend sviluppata per il progetto AirQualityInsight. L'esposizione verrà articolata in due parti distinte: nella prima sottosezione verrà presentata l'interfaccia utente e verranno illustrate le scelte progettuali che hanno guidato la definizione del design, mentre nella seconda sottosezione verrà fornita una descrizione dettagliata dell'implementazione tecnica dell'applicazione.

3.4.1 Design e architettura dell'interfaccia

In questa sottosezione verrà illustrata l'interfaccia dell'applicazione e verranno esaminate le scelte progettuali che hanno determinato la sua configurazione. La progettazione dell'interfaccia è stata condotta tenendo conto degli obiettivi e delle funzionalità stabilite durante la fase di analisi, nonché dello studio dello stato dell'arte delle applicazioni attualmente disponibili nel settore. Il processo di design dell'interfaccia grafica è stato orientato dai seguenti principi guida:

- i requisiti funzionali e non funzionali definiti rispettivamente nelle sottosezioni 1.6.1 e 1.6.2

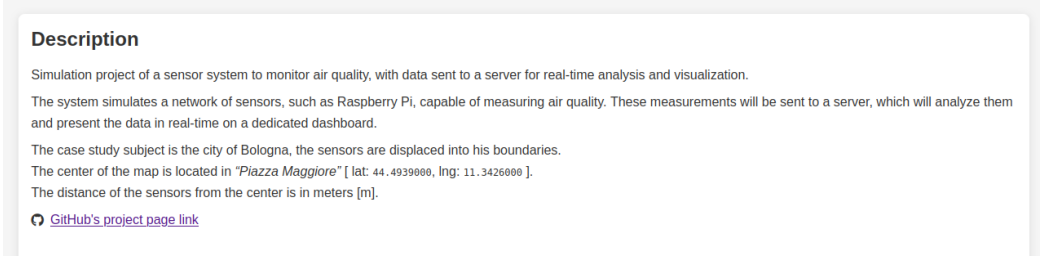
- l'approccio mobile-first
- l'adozione del principio KISS, orientando l'interfaccia verso uno stile minimalista che privilegia strumenti di comunicazione non testuali, quali icone, elementi cromatici e simboli grafici
- l'analisi dello stato dell'arte delle applicazioni per il monitoraggio della qualità dell'aria, riportata nel capitolo 1: si è scelto di mantenere continuità con le applicazioni esistenti per il monitoraggio della qualità dell'aria, al fine di offrire agli utenti un'esperienza familiare e sfruttare le soluzioni progettuali già consolidate

L'architettura dell'interfaccia è stata strutturata mediante la suddivisione dei componenti in tre categorie principali, classificate in base alla loro collocazione spaziale e alle rispettive funzionalità, sia individuali che sinergiche con gli altri elementi del sistema. Nei paragrafi seguenti verranno presentate le categorie identificate e i rispettivi componenti costitutivi.

3.4.2 Implementazione

In questa sottosezione verrà approfondita nel dettaglio l'implementazione dell'applicazione web di AirQualityInsight. Tale applicazione è stata realizzata basandosi principalmente su Vue e Leaflet. Nelle seguenti sottosezioni verranno esaminati i componenti dell'applicativo, ognuno dei quali fa riferimento ad una porzione dell'interfaccia utente.

Intestazione La prima parte dell'applicazione web fornisce un'introduzione riguardo lo scopo del progetto e le sue funzionalità. Nella seguenti immagini, vengono presentate all'utente, dall'alto verso il basso, una breve descrizione 3.5, la tabella relativa ai criteri di valutazione della qualità dell'aria 3.6, una guida sintetica sull'utilizzo della pagina 3.7 ed infine il metodo di calcolo utilizzato per ottenere l'European Air Quality Index (EAQI) 3.8.



Description

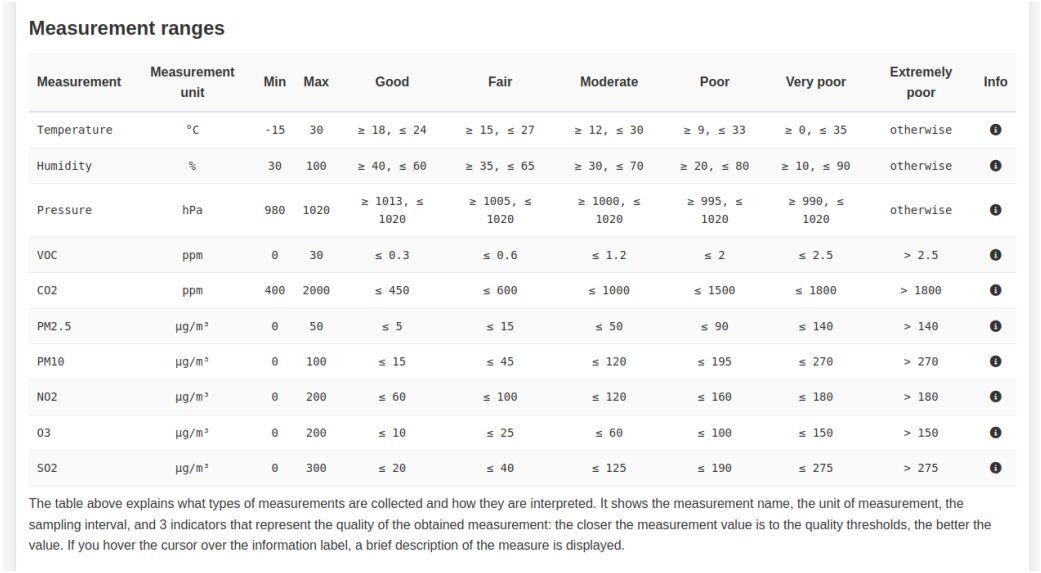
Simulation project of a sensor system to monitor air quality, with data sent to a server for real-time analysis and visualization.

The system simulates a network of sensors, such as Raspberry Pi, capable of measuring air quality. These measurements will be sent to a server, which will analyze them and present the data in real-time on a dedicated dashboard.

The case study subject is the city of Bologna, the sensors are displaced into his boundaries.
The center of the map is located in *"Piazza Maggiore"* [lat: 44.4939000, lng: 11.3426000].
The distance of the sensors from the center is in meters [m].

GitHub's project page link

Figura 3.5: Descrizione

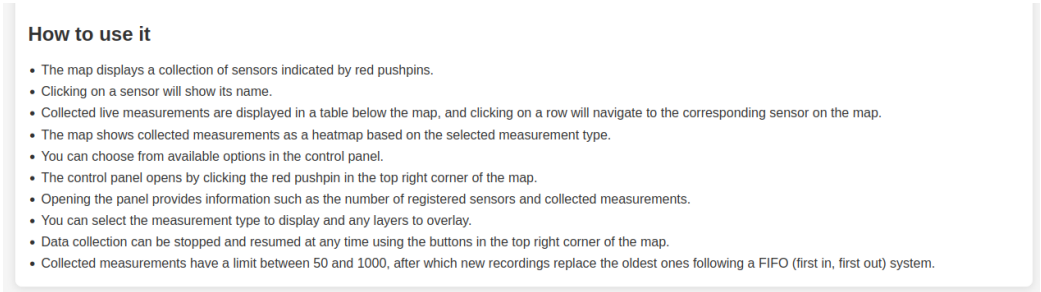


Measurement ranges

Measurement	Measurement unit	Min	Max	Good	Fair	Moderate	Poor	Very poor	Extremely poor	Info
Temperature	°C	-15	30	≥ 18, ≤ 24	≥ 15, ≤ 27	≥ 12, ≤ 30	≥ 9, ≤ 33	≥ 0, ≤ 35	otherwise	
Humidity	%	30	100	≥ 40, ≤ 60	≥ 35, ≤ 65	≥ 30, ≤ 70	≥ 20, ≤ 80	≥ 10, ≤ 90	otherwise	
Pressure	hPa	980	1020	≥ 1013, ≤ 1020	≥ 1005, ≤ 1020	≥ 1000, ≤ 1020	≥ 995, ≤ 1020	≥ 990, ≤ 1020	otherwise	
VOC	ppm	0	30	≤ 0.3	≤ 0.6	≤ 1.2	≤ 2	≤ 2.5	> 2.5	
CO2	ppm	400	2000	≤ 450	≤ 600	≤ 1000	≤ 1500	≤ 1800	> 1800	
PM2.5	µg/m³	0	50	≤ 5	≤ 15	≤ 50	≤ 90	≤ 140	> 140	
PM10	µg/m³	0	100	≤ 15	≤ 45	≤ 120	≤ 195	≤ 270	> 270	
NO2	µg/m³	0	200	≤ 60	≤ 100	≤ 120	≤ 160	≤ 180	> 180	
O3	µg/m³	0	200	≤ 10	≤ 25	≤ 60	≤ 100	≤ 150	> 150	
SO2	µg/m³	0	300	≤ 20	≤ 40	≤ 125	≤ 190	≤ 275	> 275	

The table above explains what types of measurements are collected and how they are interpreted. It shows the measurement name, the unit of measurement, the sampling interval, and 3 indicators that represent the quality of the obtained measurement: the closer the measurement value is to the quality thresholds, the better the value. If you hover the cursor over the information label, a brief description of the measure is displayed.

Figura 3.6: Tabella valori riferimento European Air Quality Index (EAQI)



How to use it

- The map displays a collection of sensors indicated by red pushpins.
- Clicking on a sensor will show its name.
- Collected live measurements are displayed in a table below the map, and clicking on a row will navigate to the corresponding sensor on the map.
- The map shows collected measurements as a heatmap based on the selected measurement type.
- You can choose from available options in the control panel.
- The control panel opens by clicking the red pushpin in the top right corner of the map.
- Opening the panel provides information such as the number of registered sensors and collected measurements.
- You can select the measurement type to display and any layers to overlay.
- Data collection can be stopped and resumed at any time using the buttons in the top right corner of the map.
- Collected measurements have a limit between 50 and 1000, after which new recordings replace the oldest ones following a FIFO (first in, first out) system.

Figura 3.7: Guida

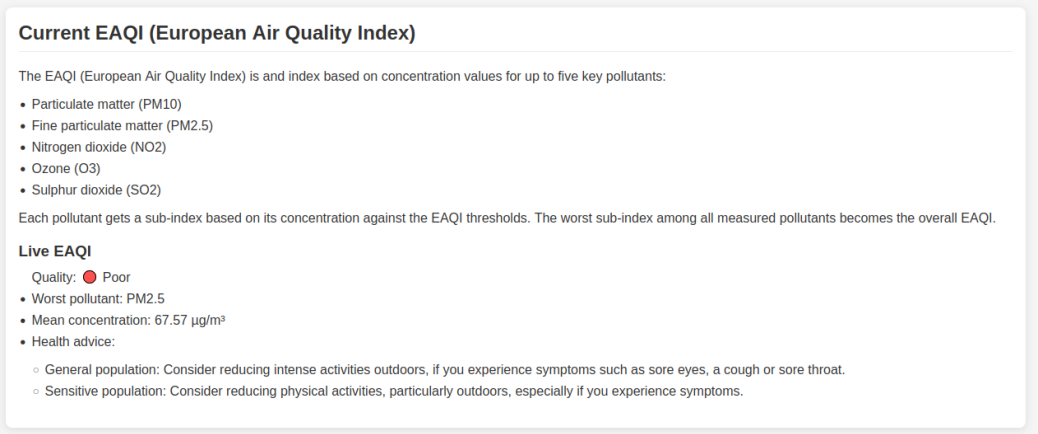


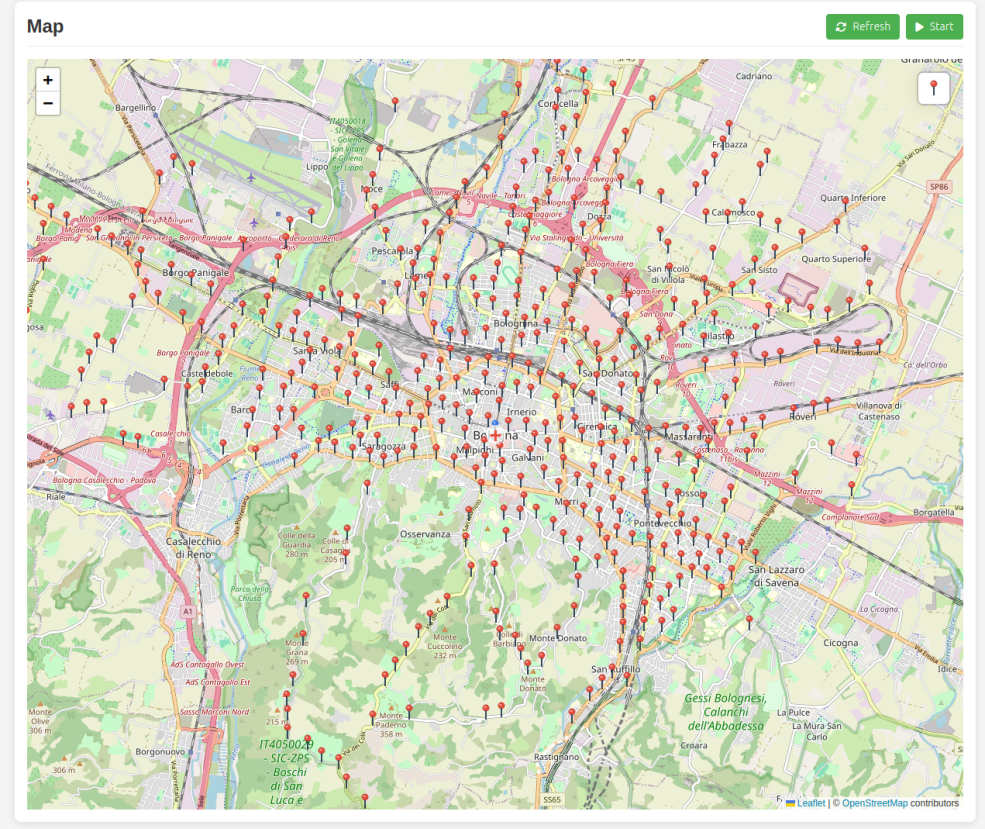
Figura 3.8: Calcolo European Air Quality Index (EAQI)

Mappa Accedendo alla pagina principale, l'utente si trova di fronte a una mappa dettagliata della città di Bologna, come documentato nell'immagine 3.9a. I sensori attivi sul territorio sono contrassegnati da spilli rossi, mentre il centro della mappa (Piazza Maggiore) è evidenziato da uno spillo blu come si può osservare nell'immagine 3.9b. Nel momento in cui viene ricevuta una nuova misurazione, il sensore responsabile viene lampeggia per un paio di secondi.

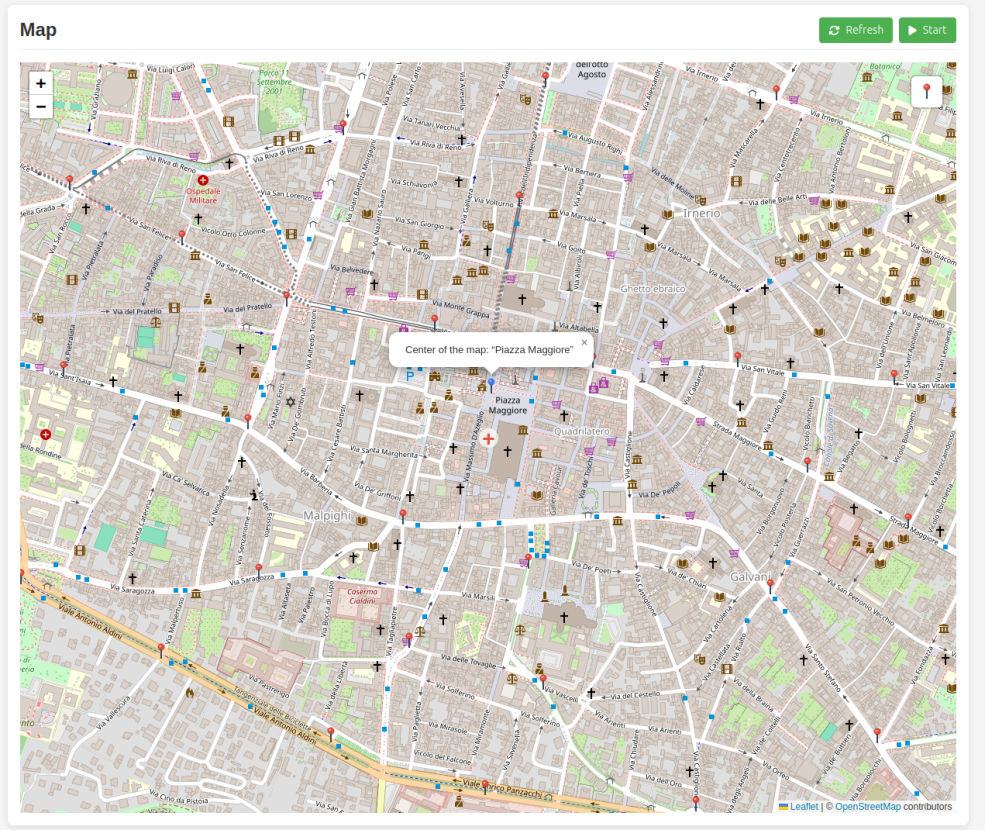
L'interfaccia cartografica è dotata di diversi controlli interattivi: sul lato sinistro sono posizionati i pulsanti per aumentare o diminuire lo zoom (+ e -) della mappa, mentre sul lato destro si trovano due pulsanti verdi ed uno bianco avente come icona uno spillo rosso.

Al centro della mappa è presente un mirino a croce (*crosshair*) rosso dentro una cornice circolare bianca, che indica il centro della mappa attualmente visualizzata. Il sistema registra automaticamente le coordinate geografiche ad esso, conservandole in memoria e rendendole disponibili per la consultazione. Quando l'utente naviga sulla mappa spostando la visualizzazione, le coordinate del punto centrale vengono aggiornate per mantenere la corrispondenza con la nuova area geografica inquadrata.

Il pulsante verde contrassegnato dalla dicitura *"Start"* rappresenta l'interruttore principale per attivare o disattivare la ricezione in tempo reale



(a) Mappa principale



(b) Centro della mappa

Figura 3.9: Mappa

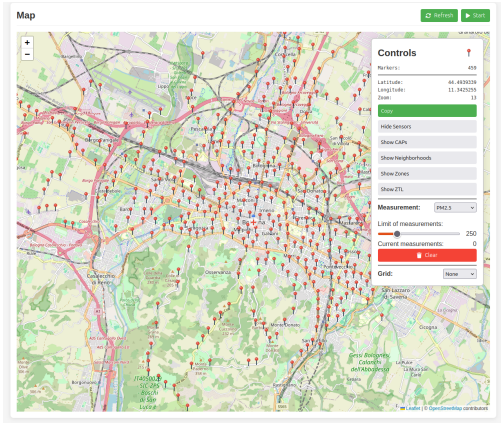
delle misurazioni dai sensori. All'avvio dell'applicazione, questa funzionalità risulta disabilitata per impostazione predefinita. Una volta attivata, il sistema inizierà a visualizzare graficamente sulla mappa i valori ricevuti dai sensori e aggiornerà automaticamente le tabelle informative sottostanti, come documentato nelle figure 3.14, 3.15, 3.16 e 3.17.

Il pulsante avente come icona lo spillo rosso attiva l'espansione di un pannello di controllo che fornisce informazioni dettagliate sulla mappa e strumenti aggiuntivi per l'interazione con l'interfaccia cartografica, come mostrato nelle figure 3.10a e 3.10b.

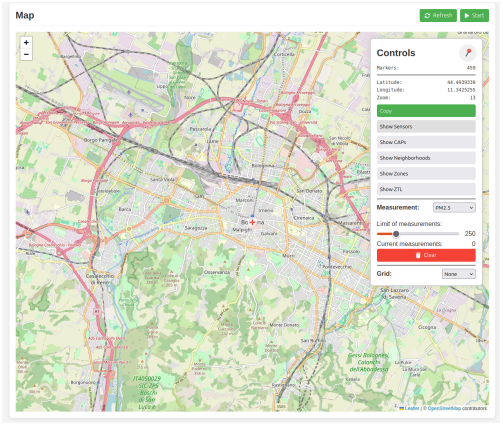
Procedendo dall'alto verso il basso, la prima informazione visualizzata concerne il conteggio complessivo di tutti i sensori presenti sulla mappa, indipendentemente dal loro stato. La sezione successiva presenta un pulsante verde dedicato alla copia delle coordinate geografiche del centro mappa (corrispondenti alla posizione del mirino). Seguono una serie di pulsanti grigi che consentono di attivare o disattivare la visualizzazione dei diversi layer cartografici: sensori, demarcazioni territoriali come le zone di avviamento postale (CAP) (figura 3.10c), quartieri (figura 3.10d), zone amministrative (figura 3.10e) e ZTL (figura 3.10f).

Nella parte inferiore del pannello si trovano un menu a tendina per la selezione del tipo di inquinante da visualizzare, uno slider orizzontale per impostare il numero di misurazioni da registrare (range da 50 a 1000), un indicatore del numero di misurazioni attualmente memorizzate e un pulsante rosso con icona cestino per l'eliminazione dei dati registrati. Il sistema implementa un meccanismo di registrazione delle misurazioni dei sensori basato su una coda (struttura dati FIFO), che garantisce l'eliminazione automatica delle registrazioni più datate quando viene raggiunta la capacità massima prestabilita, conservando i dati più recenti.

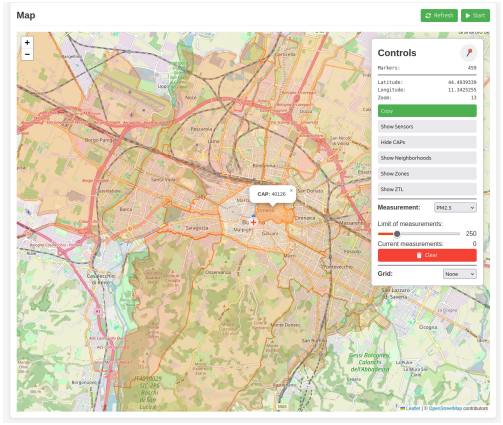
Conclude l'interfaccia un menu a tendina aggiuntivo per l'attivazione di griglie di riferimento sulla mappa quali una grigia (figura 3.11a), una rossa (figura 3.11b) ed una grigia crosshair che divide la mappa in quattro quarti (figura 3.11c).



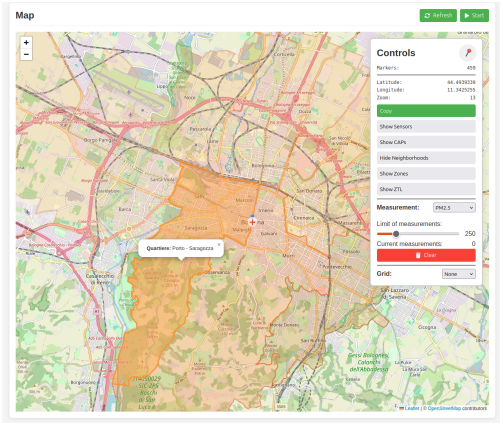
(a) Pannello controllo espanso con sensori visualizzati



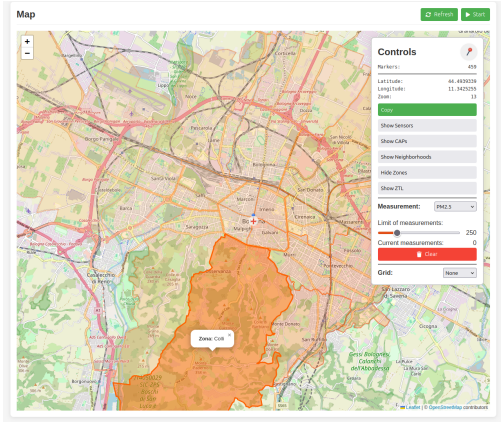
(b) Pannello controllo espanso con sensori nascosti



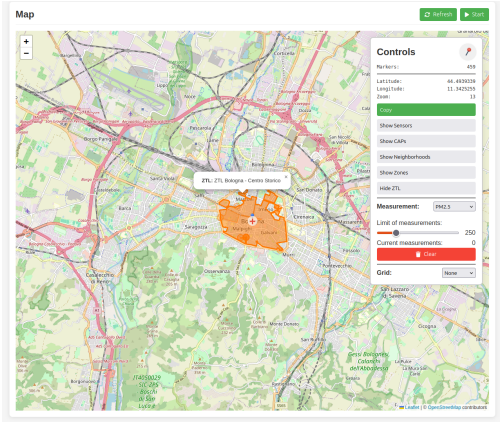
(c) Livello a CAP



(d) Livello a quartieri

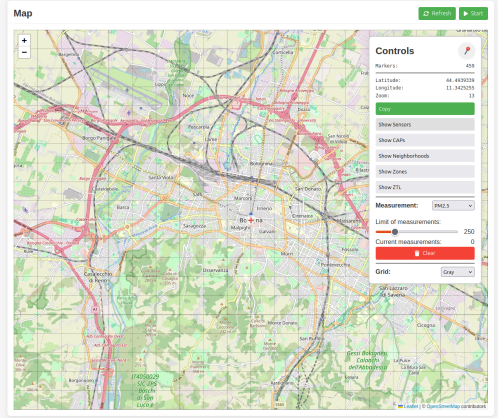


(e) Livello a zone

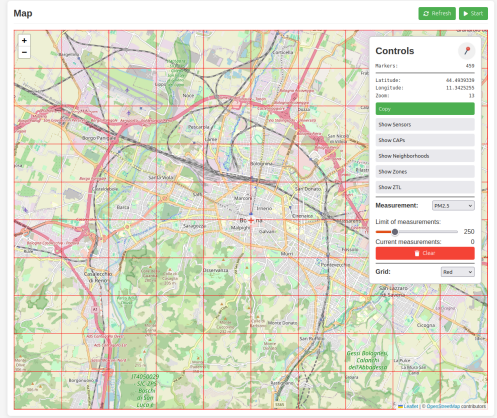


(f) Livello ZTL

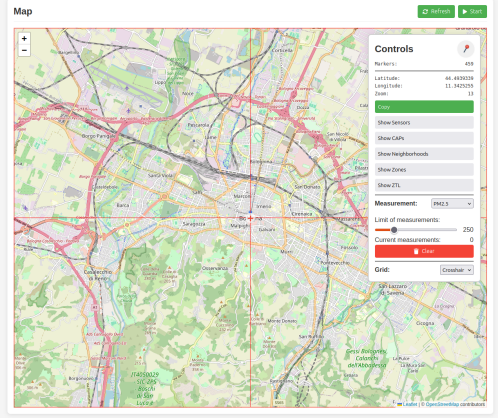
Figura 3.10: Livelli mappa



(a) Griglia grigia



(b) Griglia rossa



(c) Griglia crosshair

Figura 3.11: Griglie mappa

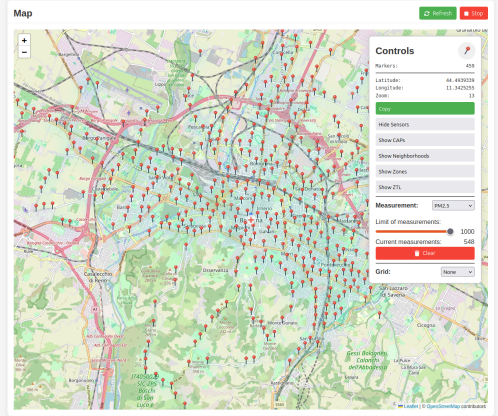
Heatmap Una heatmap (mappa di calore) è una rappresentazione grafica bidimensionale di dati in cui i valori individuali, contenuti in una matrice, sono rappresentati attraverso colori [75]. Questa tecnica di visualizzazione permette di identificare rapidamente pattern, correlazioni e anomalie all'interno di dataset complessi mediante l'uso di una scala cromatica che associa intensità di colore a valori numerici [76]. Il suo utilizzo aiuta a rendere più intuitiva la distribuzione dei dati poiché a maggior concentrazione risulta un colore più intenso, facilitandone la comprensione ed attirando l'attenzione sui principali focolai.

Formalmente, data una matrice $M \in \mathbb{R}^{m \times n}$ dove M_{ij} rappresenta il valore nella posizione (i, j), una heatmap è una funzione di mappatura $f : M_{ij} \to C_{ij}$ dove C_{ij} è il colore corrispondente al valore M_{ij} secondo una scala cromatica predefinita.

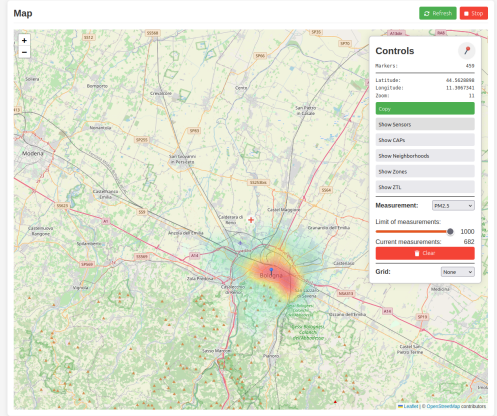
La scelta della palette di colori è cruciale per l'efficacia comunicativa della heatmap. È fondamentale utilizzare scale cromatiche che rispettino principi di accessibilità e che siano percettivamente uniformi [77]. Tali colori sono infatti quelli forniti dall'European Environment Agency (EEA) come indicato nella tabella 1.1;

Nelle immagini seguenti viene mostrata la heatmap relativa alle misurazioni di PM2.5. Partendo dalla vista default in immagine 3.12a, passiamo ad una versione con zoom inferiore senza sensori (figura 3.12b) e con sensori (figura 3.12c). Successivamente diminuiamo ulteriormente lo zoom senza sensori (figura 3.12d), per poi riavvicinarci (figura 3.12e), visualizzarle la ZTL (figura 3.12f) ed i quartieri (figura 3.13a). Infine, visualizziamo una versione con concentrazioni più elevate (figura 3.13b) rispetto alle prime, essendo passato più tempo ed avendo quindi un numero maggiore di misurazioni, e le altre misurazioni disponibili (figura 3.13c).

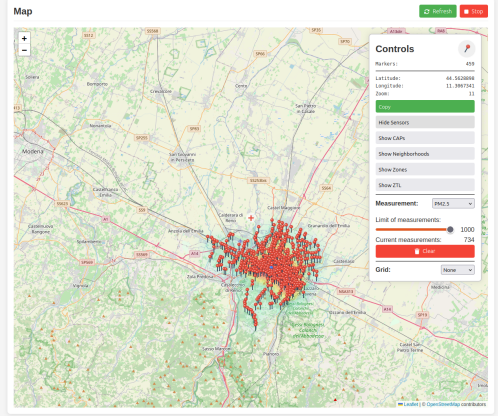
Come anticipato, allontanarci dal centro della mappa porterà a concentrazioni maggiori, mentre avvicinarci il contrario. Anche il numero di misurazioni influisce sulle concentrazioni della mappa di calore, essendo direttamente correlate.



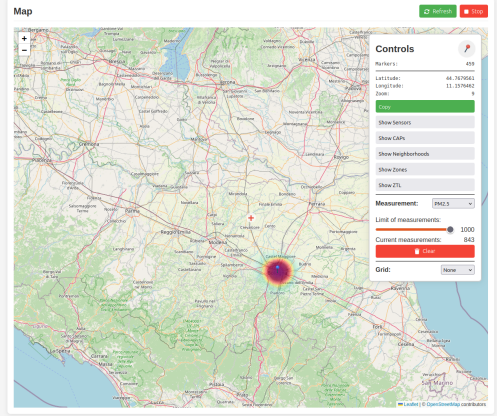
(a) Heatmap default



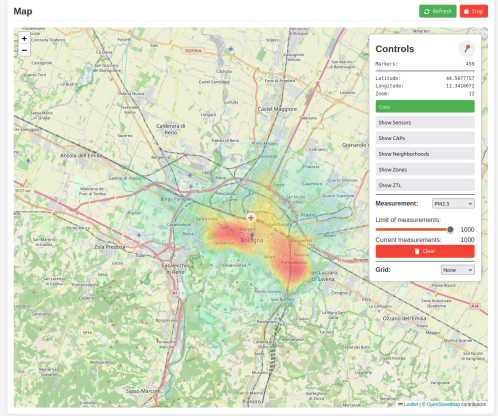
(b) Heatmap zoom 11 senza sensori



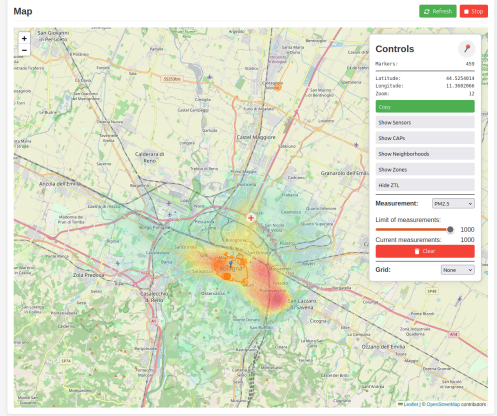
(c) Heatmap zoom 11 con sensori



(d) Heatmap zoom 9 senza sensori

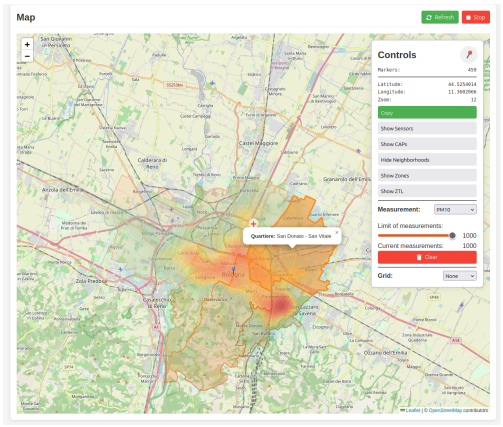


(e) Heatmap zoom 12 senza sensori

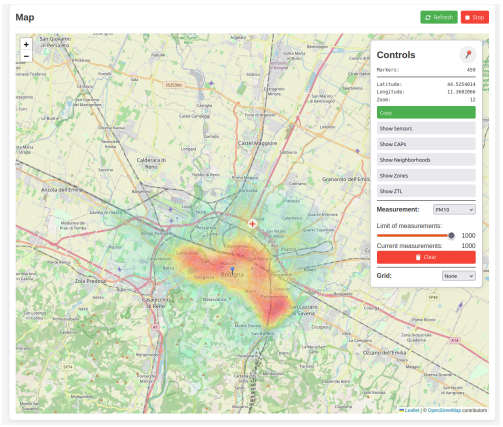


(f) Heatmap zoom 12 senza sensori ZTL

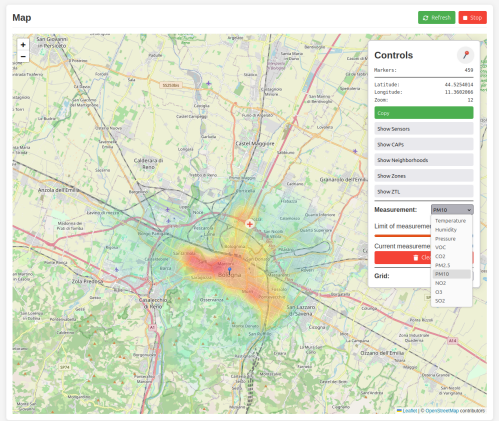
Figura 3.12: Heatmap - parte 1



(a) Heatmap zoom 12 senza sensori con quartieri



(b) Heatmap zoom 12 senza sensori, concentrazione maggiore



(c) Heatmap zoom 12 senza sensori, apertura lista misurazioni disponibili

Figura 3.13: Heatmap - parte 2

Ultime misurazioni La tabella delle misurazioni illustrata nella figura 3.14 offre una visualizzazione organizzata delle ultime 50 rilevazioni acquisite dal sistema. Ogni misurazione viene presentata su una riga separata, contenente informazioni essenziali come l'identificativo del sensore, il momento preciso dell'acquisizione e tutti i parametri rilevati con le corrispondenti unità di misura. L'interfaccia è progettata per facilitare la navigazione: selezionando una qualsiasi riga della tabella, l'utente viene automaticamente reindirizzato alla vista mappa centrata sul sensore responsabile di quella specifica rilevazione. Questa funzionalità permette di collegare rapidamente i dati numerici alla loro posizione geografica, offrendo una comprensione più completa delle informazioni raccolte. La struttura tabulare garantisce una consultazione rapida e ordinata dei dati più recenti, mantenendo sempre visibili le informazioni più aggiornate del sistema di monitoraggio.

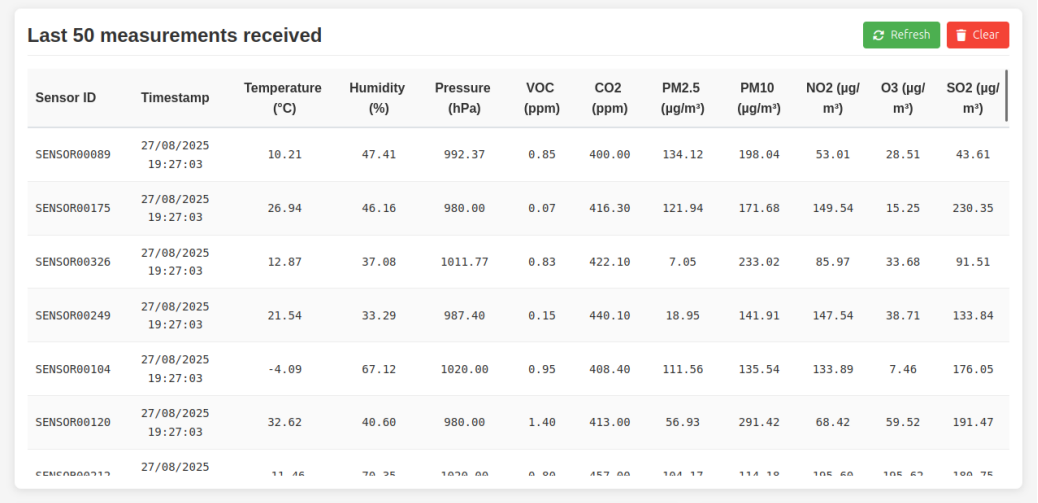


Figura 3.14: Tabella ultime misurazioni

Statistiche La tabella delle statistiche riportata nell'immagine 3.15 fornisce un'analisi completa dei dati per ogni tipologia di misurazione ambientale raccolta dal sistema. Il processo di elaborazione calcola automaticamente gli indicatori statistici fondamentali: vengono determinati valore medio e mediana per identificare la tendenza centrale, mentre i valori minimo e massimo evidenziano gli estremi registrati. Il range di variazione, ottenuto dalla differenza tra questi due estremi, quantifica l'ampiezza delle oscillazioni rilevate. Oltre agli indicatori numerici, il sistema integra una valutazione qualitativa dell'aria specifica per ciascun tipo di inquinante monitorato. Questa analisi applica criteri di classificazione standardizzati riportati nella tabella 1.1, i quali traducono i valori numerici in categorie comprensibili, accompagnate da un sistema di colori intuitivo che permette di identificare immediatamente il livello di qualità raggiunto. L'approccio combinato tra dati statistici e valutazione qualitativa offre agli utenti sia una comprensione tecnica dettagliata che un'interpretazione immediata dello stato ambientale monitorato.

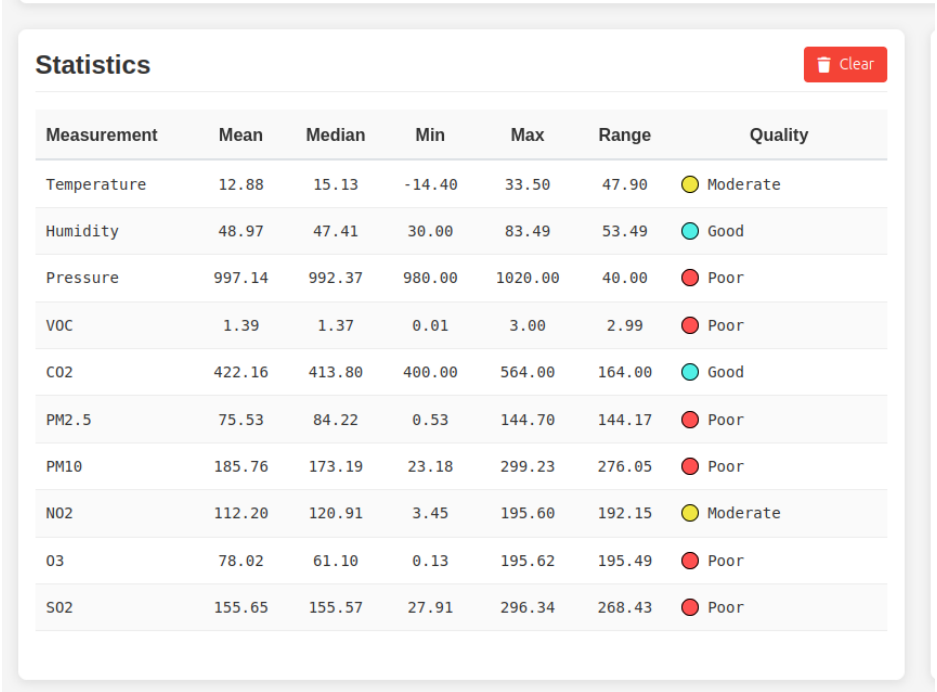


Figura 3.15: Tabella statistiche

Log di sistema L'applicazione mantiene un registro completo delle attività utente attraverso una tabella di log che documenta tutte le interazioni, sia quelle effettuate tramite interfaccia grafica che attraverso chiamate API, come si può vedere nell'immagine 3.16. Il sistema traccia diversi tipi di eventi: l'arrivo di nuove misurazioni dai sensori, i clic sui dispositivi nell'interfaccia, le operazioni di registrazione dei sensori e l'attivazione o disattivazione della ricezione dati. Per ottimizzare la visualizzazione, il log raggruppa automaticamente gli eventi identici che si verificano nel medesimo secondo, mostrando il numero di occorrenze (in grigio) accanto alla voce principale. Questa funzionalità evita la duplicazione di informazioni e mantiene il registro più leggibile. Il sistema limita la visualizzazione a un massimo di 25 righe per garantire prestazioni ottimali e una consultazione agevole. Quando questo limite viene raggiunto, le voci più recenti sostituiscono automaticamente quelle più datate, assicurando che il log mostri sempre le attività più attuali del sistema.

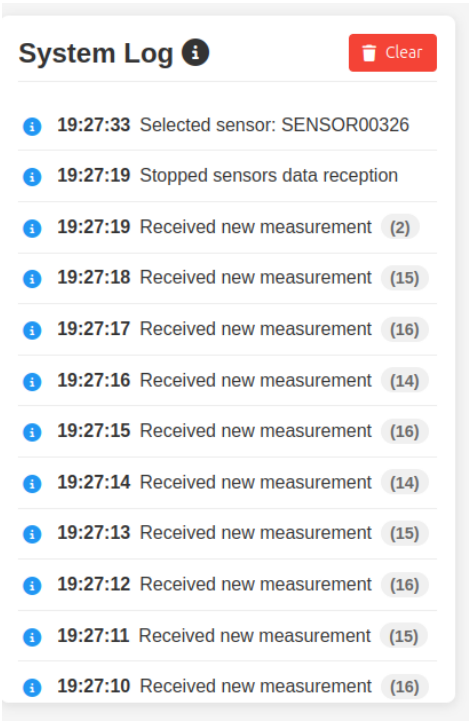
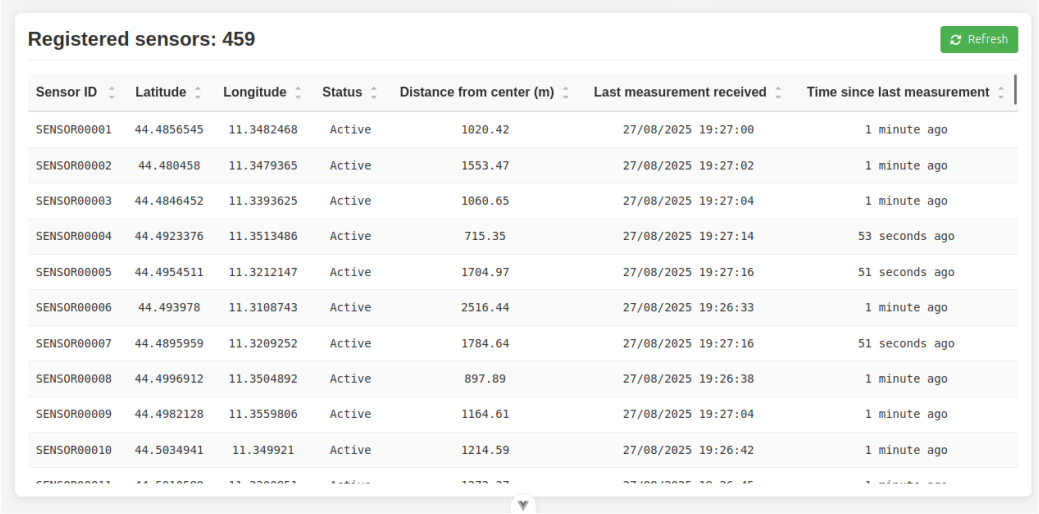


Figura 3.16: Tabella log di sistema

Tabella dei sensori registrati La parte finale della pagina principale riporta la tabella dei sensori registrati.

Questa tabella è formata da diverse colonne, quali l'id del sensore, latitudine, longitudine, stato, distanza dal centro della mappa, data ultima misurazione registrata e relativo tempo trascorso da essa.



Registered sensors: 459 — Refresh

Sensor ID	Latitude	Longitude	Status	Distance from center (m)	Last measurement received	Time since last measurement
SENSOR00001	44.4856545	11.3482468	Active	1020.42	27/08/2025 19:27:00	1 minute ago
SENSOR00002	44.480458	11.3479365	Active	1553.47	27/08/2025 19:27:02	1 minute ago
SENSOR00003	44.4846452	11.3393625	Active	1060.65	27/08/2025 19:27:04	1 minute ago
SENSOR00004	44.4923376	11.3513486	Active	715.35	27/08/2025 19:27:14	53 seconds ago
SENSOR00005	44.4954511	11.3212147	Active	1704.97	27/08/2025 19:27:16	51 seconds ago
SENSOR00006	44.493978	11.3108743	Active	2516.44	27/08/2025 19:26:33	1 minute ago
SENSOR00007	44.4895959	11.3209252	Active	1784.64	27/08/2025 19:27:16	51 seconds ago
SENSOR00008	44.4996912	11.3504892	Active	897.89	27/08/2025 19:26:38	1 minute ago
SENSOR00009	44.4982128	11.3559806	Active	1164.61	27/08/2025 19:27:04	1 minute ago
SENSOR00010	44.5034941	11.349921	Active	1214.59	27/08/2025 19:26:42	1 minute ago

Figura 3.17: Tabella sensori registrati

Il calcolo per ottenere la distanza di un sensore dal centro della mappa è realizzato attraverso la formula di Haversine [78], ovvero una funzione usata per calcolare la distanza ortodromica (linea retta sulla superficie terrestre) tra due punti. Essendo i sensori dotati di coordinate spaziali quali latitudine e longitudine, come il centro della mappa, la formula è risultata l'ideale per calcolarne la distanza.

Formula di Haversine Di seguito la formula in termini matematici 3.18, con relativa legenda 3.4.2, e la versione di codice Javascript 3.12 realmente utilizzata dall'applicazione.

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{3.4}$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \tag{3.5}$$

$$d = R \cdot c \tag{3.6}$$

Figura 3.18: Formula matematica di Haversine

Simbolo	Significato
φ	latitudine (in radianti)
φ_1, φ_2	latitudine del punto 1 e punto 2
$\Delta\varphi$	differenza di latitudine ($\varphi_2 - \varphi_1$)
λ	longitudine (in radianti)
λ_1, λ_2	longitudine del punto 1 e punto 2
$\Delta\lambda$	differenza di longitudine ($\lambda_2 - \lambda_1$)
R	raggio terrestre medio ($\sim$ 6371 km)
d	distanza tra i punti (in metri o km)
c	distanza angolare (in radianti)
a	termine intermedio

Listing 3.12: Formual di Haversine in codice Javascript

```
1  // Function to calculate the distance between two
       geographic points (Haversine formula)
2  calculateDistance(lat1, lon1, lat2, lon2) {
3    const R = 6371000; // Earth radius in meters
4    const dLat = ((lat2 - lat1) * Math.PI) / 180;
5    const dLon = ((lon2 - lon1) * Math.PI) / 180;
6    const a =
7      Math.sin(dLat / 2) * Math.sin(dLat / 2) +
8      Math.cos((lat1 * Math.PI) / 180) *
9      Math.cos((lat2 * Math.PI) / 180) *
10     Math.sin(dLon / 2) *
```

```
11        Math.sin(dLon / 2);
12      const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 -
            a));
13      return R * c;
14    }
```

Conclusioni

Il progetto AirQualityInsight ha dimostrato la fattibilità e l'efficacia di un approccio integrato al monitoraggio ambientale urbano, combinando tecnologie moderne di sviluppo web, architetture distribuite e tecniche avanzate di visualizzazione dati. L'implementazione del sistema ha permesso di validare le scelte architetturali adottate e di confermare l'importanza della user experience nella progettazione di strumenti per il monitoraggio ambientale destinati al pubblico generale.

Uno degli aspetti più significativi emersi durante lo sviluppo è stata l'importanza della geolocalizzazione strategica dei sensori. L'utilizzo delle query Overpass per identificare le intersezioni stradali più rilevanti e la successiva applicazione di algoritmi di filtraggio spaziale hanno permesso di ottimizzare la copertura territoriale, bilanciando efficacemente densità di monitoraggio e utilizzo delle risorse. La scelta di concentrarsi sui punti di maggiore criticità del traffico urbano si è dimostrata una strategia interessante per massimizzare l'utilità informativa del sistema.

L'implementazione delle heatmap dinamiche ha evidenziato le potenzialità delle visualizzazioni geografiche nel rendere accessibili informazioni tecniche complesse. La capacità di tradurre concentrazioni di inquinanti in rappresentazioni cromatiche intuitive ha trasformato dataset numerici in strumenti di comunicazione efficaci e rappresentativi, facilitando la comprensione immediata delle condizioni ambientali e dei loro pattern spaziali. L'integrazione con gli standard europei EAQI ha inoltre garantito la coerenza con i sistemi di monitoraggio ufficiali, aumentando la credibilità e l'utilità pratica del

sistema.

Il progetto ha tuttavia evidenziato anche alcune limitazioni e aree di possibile miglioramento. La natura simulata dei dati, pur permettendo lo sviluppo e il testing del sistema, non può sostituire completamente l'integrazione con sensori reali per una validazione completa dell'efficacia operativa. Inoltre, l'attuale implementazione si concentra esclusivamente sul territorio bolognese, limitando la valutazione della scalabilità geografica del sistema.

Le fondamenta architetturali e tecnologiche di AirQualityInsight aprono numerose prospettive per estensioni e miglioramenti futuri, ciascuna delle quali potrebbe significativamente ampliare l'utilità e l'impatto del sistema. Di seguito si propongo degli sviluppi futuri che potrebbero arricchire il sistema realizzato in modo da accrescerne il potere informativo e migliorare l'esperienza utente con l'aggiunta di nuove funzionalità.

Le evoluzioni principali riguardano l'implementazione di funzionalità di analisi retrospettiva con controlli temporali granulari per visualizzare l'evoluzione della qualità dell'aria su diverse scale temporali, dall'analisi oraria a quella mensile. Questa funzionalità richiederebbe algoritmi di compressione e aggregazione dei dati storici per garantire prestazioni ottimali.

Un sistema di notifiche proattivo trasformerebbe il sistema da strumento passivo a piattaforma di allerta attiva, implementando soglie personalizzabili e meccanismi di notifica multicanale. Gli algoritmi predittivi potrebbero anticipare il superamento delle soglie critiche, integrando servizi di messaggistica esterni per garantire l'affidabilità delle comunicazioni.

Il modulo di reporting avanzato beneficerebbe enti pubblici e ricercatori attraverso la generazione automatica di grafici statistici e report personalizzabili, trasformando i dati grezzi in strumenti di supporto decisionale. L'implementazione includerebbe librerie di visualizzazione avanzate e template conformi agli standard normativi europei.

I servizi di prossimità geografica fornirebbero informazioni sulla qualità dell'aria nel raggio di distanze personalizzabili, utilizzando API di geolocalizzazione e algoritmi di interpolazione spaziale per calcolare indici medi in

tempo reale, risultando particolarmente utili per applicazioni mobile.

L'evoluzione più ambiziosa riguarda lo sviluppo di capacità predittive tramite modelli di machine learning che integrino variabili meteorologiche, pattern di traffico e dati storici per fornire previsioni affidabili a breve e medio termine.

L'integrazione con l'ecosistema urbano includerebbe l'interoperabilità con sistemi smart city esistenti attraverso API standardizzate, mentre l'espansione geografica oltre Bologna richiederebbe meccanismi automatizzati per l'acquisizione di dati via OpenStreetMap. Infine, l'evoluzione verso un monitoraggio collaborativo integrerebbe sensori citizen science e dispositivi IOT domestici, democratizzando l'accesso alle informazioni ambientali.

La modularità del sistema e l'adozione di standard aperti facilitano un'implementazione incrementale che bilanci innovazione tecnologica e sostenibilità operativa.

Bibliografia

[1] Chimica Online. Che cos'è l'equatore, 2025.

[2] Topgeometri. Concetti base sul sistema wgs84 del gps, 2022.

[3] European Environment Agency. European air quality index, map and charts, 2025.

[4] AccuWeather. Accuweather website, 2025.

[5] European Environment Agency. European environment agency, 2025.

[6] Agenzia Europea dell'Ambiente and Commissione Europea. Indice europeo della qualità dell'aria: informazioni aggiornate sulla qualità dell'aria a portata di mano, 2017.

[7] Google. Visualizzare le informazioni sulla qualità dell'aria in google maps, 2025.

[8] International Association of Oil & Gas Producers. Geomatics guidance note 7-1: Using the epsg geodetic parameter dataset. Technical Report 373-7-1, IOGP, London, 2019.

[9] V. Ashkenazi, S.A. Crane, G.T. Johnston, and S. Millard. Coordinate systems and map projections for offshore surveying. *International Hydrographic Review*, 61(2):7–23, 1984.

[10] EPSG Official Database. Wgs 84 - wgs84 - world geodetic system 1984, used in gps - epsg:4326. EPSG Official Database, 2022. Definizione ufficiale completa con parametri WKT e Proj4.

[11] EPSG Official Database. Wgs 84 / pseudo-mercator - spherical mercator, google maps, openstreetmap, bing, arcgis, esri - epsg:3857. EPSG Official Database, 2020. Definizione completa Pseudo-Mercator.

[12] EPSG Official Database. Wgs 84 / utm zone 33n - epsg:32633. EPSG Official Database, 2022. Definizione WGS84/UTM zone 33N.

[13] EPSG Official Database. Monte mario / italy zone 1 - epsg:3003. EPSG Official Database, 2021. Definizione Monte Mario zone 1.

[14] GPS.GOV. The world geodetic system 1984 (wgs84) and its updates. GPS.gov, 2008.

[15] P. Neis and A. Zipf. Openstreetmap. *International Journal of Interactive Communication Systems and Technologies*, 2(1):69–78, 2012.

[16] C.C. Fonte, V. Antoniou, L. Bastin, J. Estima, J.J. Arsanjani, J.C.L. Bayas, L. See, and R. Vatseva. Osm science—the academic study of the openstreetmap project, data, contributors, community, and applications. *ISPRS International Journal of Geo-Information*, 11(4):230, 2022.

[17] OpenStreetMap Wiki. History of openstreetmap. OpenStreetMap Wiki, 2024. Accesso: 2024.

[18] Steve Coast. First street entered. OpenStreetMap mailing list, December 2004.

[19] OpenStreetMap Foundation. About openstreetmap. OpenStreetMap Wiki, 2006. Fondazione istituita il 22 agosto 2006.

[20] Sukhjit Singh Sehra, Jaiteg Singh, and Hardeep Singh Rai. Assessment of openstreetmap data - a review. *International Journal of Computer Applications*, 76(8):1–8, 2013.

[21] J.E. Vargas-Muñoz, S. Srivastava, D. Tuia, and A.X. Falcão. Openstreetmap: Challenges and opportunities in machine learning and remote sensing. *IEEE Geoscience and Remote Sensing Magazine*, 8(3):44–55, 2020. Review sulle applicazioni ML e remote sensing con OSM.

[22] Sterling Quinn. Using openstreetmap for your research: leveraging a massive global geographic database that emphasizes local knowledge. SFU Library Scholarly Publishing, 2022.

[23] OpenStreetMap Wiki. Elements. OpenStreetMap Wiki, 2024. Documentazione del modello dati OSM: nodi, percorsi e relazioni.

[24] FLOSS Manuals. The openstreetmap data model. FLOSS Manuals, 2024. Guida completa al modello dati OpenStreetMap.

[25] OpenStreetMap Wiki. Relation. OpenStreetMap Wiki, 2024. Documentazione delle relazioni OSM.

[26] OpenStreetMap Wiki. Map features. OpenStreetMap Wiki, 2024. Sistema di tag e caratteristiche delle mappe OSM.

[27] Sterling Quinn. Using openstreetmap for your research: leveraging a massive global geographic database that emphasizes local knowledge. SFU Library Scholarly Publishing, 2022. Guida per ricercatori sull'uso di OSM.

[28] OpenStreetMap Wiki. Overpass api. OpenStreetMap Wiki, 2024. API per interrogazioni avanzate sui dati OSM.

[29] Web based data mining tool. overpass turbo. Web-based data mining tool, 2024. Interfaccia web interattiva per Overpass API.

[30] Geoapify Documentation. 3 ways to get openstreetmap(osm) data. Geoapify Documentation, May 2024. Metodi per ottenere e utilizzare dati OSM.

[31] Evan You. Vue.js: The progressive javascript framework, 2014.

[32] Misko Hevery and Adam Abrons. Angularjs: Superheroic javascript mvw framework, 2010.

[33] Pete Hunt, Jordan Walke, and React Team. React: A javascript library for building user interfaces, 2013.

[34] MDN Web Docs Contributors. Single-page applications, 2024.

[35] Vue.js Core Team. Vue cli: Standard tooling for vue.js development, 2018. Strumento da riga di comando per scaffolding e gestione progetti Vue.js.

[36] Evan You and Vite Contributors. Vite: Next generation frontend tooling, 2021. Build tool veloce per applicazioni web moderne.

[37] Eduardo San Martin Morote Posva and Vue.js Contributors. Vue router: The official router for vue.js, 2016. Libreria ufficiale per il routing in applicazioni Vue.js.

[38] Evan You and Vue.js Contributors. Vuex: Centralized state management for vue.js, 2016. Libreria per la gestione centralizzata dello stato nelle applicazioni Vue.js.

[39] Eduardo San Martin Morote Posva and Pinia Contributors. Pinia: The vue store that you will enjoy using, 2021. Store di nuova generazione per Vue.js, successore ufficiale di Vuex.

[40] Vue.js Core Team. Vue devtools: Browser extension for debugging vue.js applications, 2016. Estensione browser per il debugging e l'analisi di applicazioni Vue.js.

[41] Vladimir Agafonkin. Leaflet: An open-source javascript library for mobile-friendly interactive maps, May 2011.

[42] Leaflet Contributors. Leaflet - npm package, 2024.

[43] Tomislav Capan. Why the hell would i use node.js? a case-by-case tutorial, August 2013.

[44] Node.js Documentation Team. Node.js event loop, timers, and process.nexttick(), 2023.

[45] Express.js Team. Express.js: Fast, unopinionated, minimalist web framework for node.js, 2025.

[46] Socket.IO Team. Socket.io: Bidirectional and low-latency communication for every platform, 2025.

[47] MongoDB Inc. Mongodb: The developer data platform, 2025.

[48] Redis Ltd. Redis: The open source, in-memory data store, 2025.

[49] W3C WoT Working Group. Web of things (wot) thing description. Technical report, World Wide Web Consortium, April 2020. W3C Recommendation.

[50] ETSI Technical Committee SmartM2M. Saref: the smart applications reference ontology, 2020. ETSI TS 103 264.

[51] W3C WoT Working Group. Web of things (wot) architecture. Technical report, World Wide Web Consortium, April 2020. W3C Recommendation.

[52] Dominique Guinard, Vlad Trifa, and Erik Wilde. The web of things: challenges and opportunities for the internet of things. *Computer*, 44(4):30–37, 2011.

[53] Guido van Rossum. *Python Reference Manual*. CWI, Amsterdam, Netherlands, 1995.

[54] Tim Peters. Pep 20 – the zen of python, 2004. Accessed: 2025-08-21.

[55] Python Software Foundation. Python package index, 2023. Accessed: 2025-08-21.

[56] Django Software Foundation. Django: The web framework for perfectionists with deadlines, 2023. Accessed: 2025-08-21.

[57] Armin Ronacher. Flask: A lightweight wsgi web application framework, 2023. Accessed: 2025-08-21.

[58] Neha Narkhede, Gwen Shapira, and Todd Palino. *Kafka: The Definitive Guide*. O'Reilly Media, 1st edition, 2017.

[59] Nishant Garg. *Apache Kafka*. Packt Publishing, 2013.

[60] Ben Stopford. *Designing Event-Driven Systems: Concepts and Patterns for Streaming Services with Apache Kafka*. O'Reilly Media, 2018.

[61] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: a distributed messaging system for log processing. *Proceedings of the NetDB Workshop*, 2011. Republished and expanded 2014.

[62] Kristina Chodorow. *MongoDB: the definitive guide*. O'Reilly Media, 2nd edition, 2013.

[63] Kyle Banker. *MongoDB in action*. Manning Publications, 2011.

[64] Eelco Plugge, Tim Hawkins, and Peter Membrey. *The definitive guide to MongoDB: the noSQL database for cloud and desktop computing*. Apress, 2010.

[65] Rick Harrison. *MongoDB Applied Design Patterns*. O'Reilly Media, 2015.

[66] Peter Membrey, Eelco Plugge, and Tim Hawkins. *The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB*. Apress, 3rd edition, 2014.

[67] Brad Dayley. *MongoDB and Python: Patterns and processes for the popular document-oriented database*. Addison-Wesley Professional, 2014.

[68] Adrian Mouat. *Using Docker: Developing and Deploying Software with Containers*. O'Reilly Media, 2015.

[69] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[70] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015.

[71] Martin Fowler and James Lewis. Microservices: a definition of this new architectural term. `https://martinfowler.com/articles/microservices.html`, 2014. Accessed: 2024-08-25.

[72] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. *Present and Ulterior Software Engineering*, pages 195–216, 2017.

[73] Chris Richardson. *Microservices Patterns: With examples in Java*. Manning Publications, 2018.

[74] Eberhard Wolff. *Microservices: Flexible Software Architecture*. Addison-Wesley Professional, 2016.

[75] Leland Wilkinson. *The Grammar of Graphics*. Springer Science & Business Media, New York, 2nd edition, 2009.

[76] William S. Cleveland. *Visualizing Data*. Hobart Press, Summit, NJ, 1993.

[77] Colin Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, Boston, MA, 3rd edition, 2012.

[78] Chris Veness. Calculate distance, bearing and more between latitude/longitude points. `https://www.movable-type.co.uk/scripts/latlong.html`, 2002-2022.

[79] European Environment Agency. Air pollution, 2025.

[80] European Environment Agency. European air quality index., 2025.

[81] European Petroleum Survey Group. Epsg:3857 - wgs 84 / pseudo-mercator. Technical Report Guidance Note 7-2, European Petroleum Survey Group, 2008. Disponibile online su epsg.org.

[82] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle. *GNSS – Global Navigation Satellite Systems: GPS, GLONASS, Galileo, and more*. Springer, 2007.

[83] S.E. Battersby, M.P. Finn, E.L. Usery, and K.H. Yamamoto. Implications of web mercator and its use in online mapping. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 49(2):85–101, 2014.

[84] Google Inc. *Google Maps API Documentation*, 2005. Documentazione tecnica originale sulla implementazione Web Mercator.

[85] OpenStreetMap. Copyright and license. OpenStreetMap, 2024.

[86] History of Information. Openstreetmap begins. History of Information, 2024.

[87] Algorithms for Geographic Data. Using openstreetmap data. Algorithms for Geographic Data, 2024. Tutorial tecnico per l'utilizzo dei dati OSM.

[88] Sandeep Pandey. Openstreetmap: Access, data mining and data manipulation. Personal Blog, October 2020. Tutorial su accesso e manipolazione dati OSM con Python.

[89] Sandeep Pandey. Open source routing: Exploring open route service (osr). Personal Blog, February 2025. Setup e utilizzo di Open Route Service.

[90] Nikolai Janakiev. Loading data from openstreetmap with python and the overpass api. *TDS Archive, Medium*, May 2023. Tutorial Python per accesso dati OSM via Overpass API.

[91] Itinero Documentation. Openstreetmap data model. Itinero Documentation, 2024. Documentazione del modello dati OSM per sviluppatori.

[92] Overpass API Documentation. The data model of openstreetmap. Overpass API Documentation, 2024. Documentazione tecnica del modello dati per Overpass API.

[93] S. Auer, J. Lehmann, and S. Hellmann. Linkedgeodata: Adding a spatial dimension to the web of data. ResearchGate, 2009. Architettura tecnica OpenStreetMap e componenti.

[94] Vladimir Agafonkin. Leaflet: An open-source javascript library for mobile-friendly interactive maps, 2011.

[95] Leaflet Contributors. Leaflet documentation, 2023.

[96] Vladimir Agafonkin and Leaflet Contributors. Leaflet, 2011. Open-source JavaScript library, Version 1.9.x.

[97] Vue.js Contributors. Vue.js documentation, 2024.

[98] Evan You and Vue.js Contributors. Vue.js, 2014. Progressive JavaScript framework, Version 3.x.

[99] Vue.js Contributors. Vue.js api reference, 2024. Official API documentation.

[100] Node.js Foundation. Node.js official documentation, 2023.

[101] Dylan Palino and Rajini Sivaram. *Kafka in Action*. Manning Publications, 2022.

[102] Gwen Shapira. Building real-time data pipelines with apache kafka. Technical report, Confluent Inc., 2016.

[103] Docker Inc. Docker compose overview. `https://docs.docker.com/compose/`, 2023. Accessed: 2024-01-15.

[104] Martin Fowler and James Lewis. Microservices. *ThoughtWorks*, 2014. Accessed: 2024-01-15.

[105] Sam Newman. *Building microservices: designing fine-grained systems*. O'Reilly Media, 2015.

[106] Liang Chen, Babar Ali, and Muhammad Bilal. From monolith to microservices: a dataflow-driven approach. In *Proceedings of the 24th Asia-Pacific Software Engineering Conference*, pages 466–475. IEEE, 2017.

[107] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. Definition, benefits and challenges of microservices: A systematic mapping study. *Journal of Systems and Software*, 144:61–79, 2018.

[108] Nuha Alshuqayran, Nour Ali, and Roger Evans. A systematic mapping study in microservice architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 44–51. IEEE, 2016.

[109] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[110] Flavio Battaglia et al. The role of ontologies in enabling dynamic interoperability. *Future Generation Computer Systems*, 76:371–382, 2017.

[111] Matthias Kovatsch, Marius Lanter, and Zach Shelby. A restful runtime for the web of things. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 10–17. IEEE, 2012.

[112] Vlad Stirbu. Towards a restful plug & play experience in the web of things. *Semantic Web*, 1(1-2):105–110, 2010.

Ringraziamenti

Vorrei ringraziare Miriana, che mi sostiene ed accompagna ogni giorno nelle difficoltà e negli impegni della vita che assieme ci stiamo costruendo.

Ringrazio i miei genitori, Anna e Domenico, mia sorella Benedetta, mia cugina Valeria e la mia famiglia tutta.

Ringrazio i miei suoceri, Immacolata e Marco, che mi hanno sin da subito accolto nella loro vita e fatto sentire parte della famiglia, ma che sopratutto mi hanno permesso di scegliere Miriana come la mia compagna.

Ringrazio Luca, che prima di essere stato il mio mentore e collega è stato mio amico.

Ringrazio Daniele, nel quale ho trovato un amico sincero e leale.

Ringrazio il Professor Olaiya, che mi ha seguito in questo lavoro con attenzione e disponibilità, senza il quale non sarebbe stato possibile realizzare quanto prodotto.