



UNIVERSITÀ DI PARMA

DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA
CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA,
ELETTRONICA E DELLE TELECOMUNICAZIONI

PROGETTAZIONE E SVILUPPO DI UNA UI
BASATA SU WEB PER IL MONITORAGGIO
DELLA QUALITÀ DELL'ARIA IN SCENARI
ETEROGENEI TRAMITE IOT

DESIGN AND DEPLOYMENT OF A WEB-BASED UI FOR
IOT-ENABLED AIR QUALITY MONITORING IN
HETEROGENEOUS SCENARIOS

Relatore:

Chiar.mo Prof. Ing. LUCA DAVOLI

Tesi di Laurea di:
EDOARDO SICHELLI
Matricola:
308342

ANNO ACCADEMICO 2022/2023

Agli amici di una vita.

Indice

Introduzione	v
1 Analisi e monitoraggio della qualità dell'aria	1
1.1 Piattaforme di visualizzazione dati esistenti	1
1.1.1 Airindex EEA	2
1.1.2 Waqi	4
2 Strumenti e tecnologie utilizzate	6
2.1 Nodo Sensoriale IoT	6
2.1.1 Raspberry PI 4	7
2.1.2 Sensore Adafruit MiCS5524	8
2.1.3 Sensore Sensirion SCD30	9
2.1.4 Sensore Sensirion AG SPS30	10
3 Architettura Software Del Sistema	12
3.1 Database relazionali e non relazionali	13
3.1.1 Database Relazionale MySQL	13
3.1.2 Database MongoDB	17
3.2 Web Application	19
3.2.1 Backend	19
Uploader	27
3.2.2 Frontend	28
Dashboard	29
Grafici	29

3.2.3 Gestione dei dati all'interno del grafico	31
Dato predetto	38
4 Conclusioni e Sviluppi Futuri	39
Bibliografia	43

Elenco delle figure

1.1	AirIndex EEA Italia.	3
1.2	Waqi.	4
2.1	Nodo sensoriale IoT.	6
2.2	Raspberry PI 4.	7
2.3	Sensore Adafruit MiCS5524.	8
2.4	Sensore SCD30.	9
2.5	SCD30 CO2.	10
2.6	Sensore Adafruit MiCS5524.	10
2.7	Sensirion AG SPS30.	10
2.8	Scheda Tecnica Sensiron AG SPS30.	11
3.1	Andamento del parametro ads1115 voltage.	30
3.2	Andamento del parametro ads1115 voltage clean.	31
3.3	Andamento del parametro scd30 co2.	33
3.4	Andamento del parametro scd30 co2 clear.	34
3.5	Andamento del parametro scd30 co2.	35
3.6	Andamento del parametro scd30 co2 small.	36
3.7	Andamento del parametro scd30 co2 1min end.	37
3.8	Andamento del parametro scd30 co2 1min beginning.	38

Introduzione

L'aria è una miscela di sostanze aeriformi (gas e vapori), la cui composizione varia in base alla quota altimetrica a cui ci si trova. Al suolo la sua composizione si aggira intorno al 78% di azoto (N_2), un 21% di ossigeno (O_2) e circa 1% di argon (Ar) [9]. Scendendo più nel dettaglio, è possibile identificare altri parametri da analizzare per capire in modo più dettagliato cosa effettivamente compone quello che respiriamo tutti i giorni. Prendendo come riferimento quello che si andrà ad analizzare nel presente lavoro di Tesi, è possibile utilizzare, fra i tanti, anche sistemi sensoriali basati sul paradigma dell'internet of things (IoT) in grado di raccogliere e analizzare vari parametri che saranno dettagliati nel seguito del lavoro di Tesi. L'anidride carbonica CO_2 , l'umidità dell'aria, e vari valori relativi ai PM presenti nell'aria. In dettaglio, il termine PM indica le "particulate matter" (anche note come "particle pollution"), ovvero un insieme di particelle di polvere, fumi e inquinamento. Si differenziano in base alla dimensione delle singole particelle, ad esempio, le PM10 sono particelle che hanno un diametro pari a 10 micrometri o minore, mentre le PM2.5 sono quelle particelle con un diametro di 2.5 micrometri o minore. A titolo di paragone, un capello umano ha un diametro di circa 70 micrometri, e questo fa intuire quanto queste particelle siano piccole. Da questo ne deriva la necessità di poter monitorare in modo quanto più corretto e puntuale possibile tali inquinanti, è fondamentale comprendere i danni provocati da queste particelle, poiché, essendo estremamente sottili, possono depositarsi nei polmoni e persino entrare nel flusso sanguigno, causando gravi problemi di salute. [3]. Secondo l'organizzazione Mondiale della Sanità, per il particolato non è possibile definire un valore limite al di sotto del quale non si verificano nella popolazione effetti sulla salute: per questo motivo la concentrazione di PM10 e PM2.5 nell'aria dovrebbe

essere mantenuta al livello più basso possibile. [8]

In questo lavoro di Tesi, i dati raccolti relativi ai PM sono PM0.5, PM1, PM2.5, PM4 e PM10.

Inoltre, per questo lavoro di Tesi, è stato sviluppato un sistema per il monitoraggio della qualità dell'aria, che integra anche una dashboard per la visualizzazione e il monitoraggio dei dati analizzati.

Il presente lavoro di Tesi è organizzato come segue:

- Nel Capitolo 1 vengono introdotti i concetti base relativi alla qualità dell'aria, come PM, umidità e anidride carbonica, e alcune dashboard esistenti per il monitoraggio della qualità dell'aria.
- Nel Capitolo 2 vengono introdotti i concetti base relativi alla dashboard sviluppata per il monitoraggio della qualità dell'aria, come le componenti che sono servite per realizzarla e le tecnologie utilizzate.
- Nel Capitolo 3 viene descritto lo sviluppo sperimentale della dashboard per il monitoraggio basato sui dati raccolti dal nodo IoT.
- Infine, nel Capitolo 4 si riportano alcune conclusioni e possibili spunti per sviluppi futuri.

Capitolo 1

Analisi e monitoraggio della qualità dell'aria

L'aria che respiriamo è come lo spazio che ci circonda: invisibile ma presente, essenziale ma spesso ignorato. Dobbiamo imparare a riconoscere il suo valore e a proteggerlo, come facciamo con il nostro ambiente naturale.

ChatGPT, 2023

1.1 Piattaforme di visualizzazione dati esistenti

La possibilità di poter monitorare e visualizzare i dati raccolti tramite sensori, unitamente alle possibili elaborazioni sugli stessi in tempo reale e tramite tecniche differenti, risulta essere di fondamentale importanza, anche per il cittadino comune che potrebbe non avere tutti gli strumenti, e non avere la conoscenza, per poter valutare tali informazioni senza un aiuto esterno. Vista l'importanza di tali dati, nel presente lavoro di Tesi è stato deciso di creare un sistema software che permetta di visualizzare in modo semplice e *user-friendly* queste informazioni, in modo da poter essere utilizzato da tutti.

Esistono già piattaforme che permettono di visualizzare dati sulla qualità dell'aria, ma la maggior parte di queste risultano essere poco specifiche e manchevoli di molte informazioni, come ad esempio la possibilità di visualizzare i dati in tempo reale, o di

poter organizzare i dati per scenario o per *use-case*, potendo selezionare pertanto una determinata località o punto di raccolta in cui uno specifico nodo IoT per la raccolta di dati sia stato installato, avendo un periodo specifico per la visualizzazione dei dati. Nelle prossime sezioni verranno descritte alcune di queste piattaforme commerciali per la visualizzazione dei dati.

1.1.1 Airindex EEA

AirIndex EEA airindex.eea.europa.eu è un servizio della comunità europea che consente di avere accesso ad una visualizzazione dei dati di qualità dell'aria nelle varie località europee. Purtroppo questo servizio consente di visualizzare i dati sulla qualità dell'aria solamente per le 48 ore precedenti, e non consente di scegliere un punto specifico, ma visualizzare dei marker colorati su varie regioni d'Italia che non permettono di comprendere in dettaglio in che modo si distribuiscono i dati. Inoltre nel servizio mancano molti punti di cui non si hanno informazioni a riguardo.

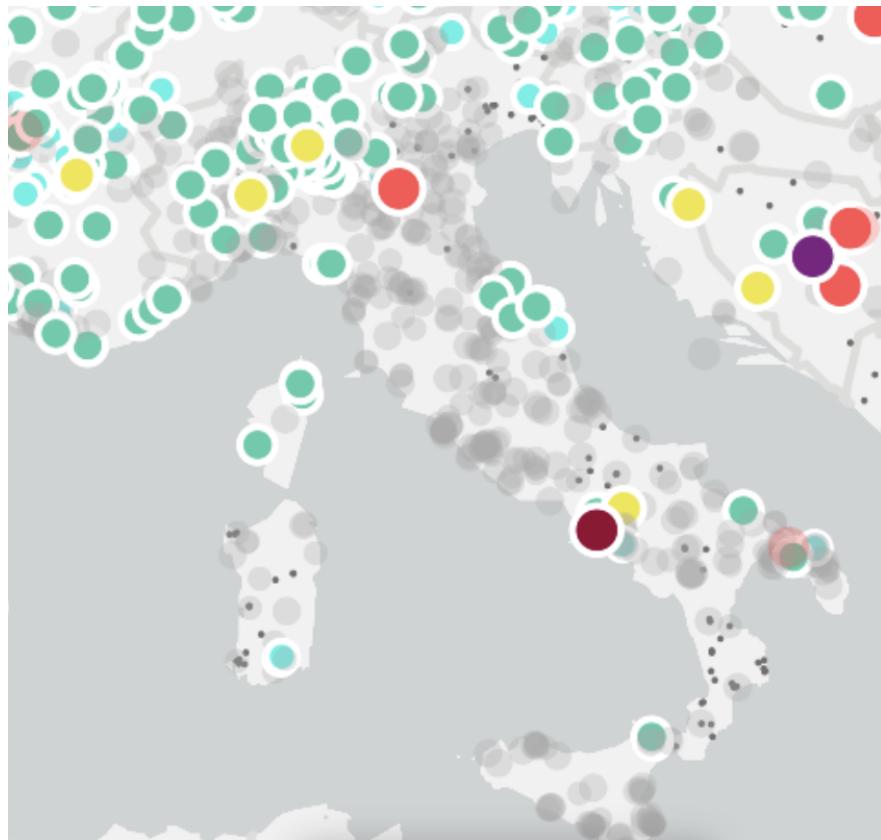


Figura 1.1: AirIndex EEA Italia.

A livello generale questa mappa rappresenta comunque un valido ausilio per comprendere (a macro livello) quali siano le zone più inquinate e quali lo siano meno. E' però possibile notare come in corrispondenza dei molteplici marker grigi corrisponde una mancanza di dati, e questo rappresenta un "problema", in quanto non è possibile visualizzare i dati in maniera completa.

Un vantaggio di questa piattaforma è legata alla disponibilità di rappresentazioni aggiuntive nelle zone in cui fornisce i dati, (una breve descrizione del motivo per il cui il servizio cataloga la zona indicata come inquinata o come non inquinata). A titolo esplicativo, la selezione del marker rosso in alto a destra in Figura 1.1 che rappresenta il comune di Schivenoglia, che fornisce un dato dettagliato con la descrizione "Air quality index: Poor (due to PM2.5)". Viene quindi specificato il motivo per il quale la zona selezionata sia stata catalogata come inquinata, però non viene fornito nessun

dettaglio sul momento in cui tale anomalia sia stata rilevata tramite la misurazione e quali siano i valori esatti.

1.1.2 Waqi

Waqi waqi.info rappresenta un altro servizio web che consente di visualizzare i dati sulla qualità dell'aria in maniera più approfondita rispetto a *Airindex EEA*. Inoltre, rispetto al servizio descritto nella sezione 1.1.1, Waqi è in grado di restituire dati e informazioni aggiuntive come mostrato in Figura 1.2.



Figura 1.2: Waqi.

Come anticipato in precedenza, entrambe queste piattaforme web mancano della possibilità di definire degli scenari (*use-case*) composti da una moltitudine di dispositivi di monitoraggio della qualità dell'aria, unitamente alla possibilità di visualizzare i dati in maniera personalizzata. Nei capitoli successivi verranno pertanto presentate le caratteristiche principali del nodo IoT prototipale utilizzato come dispositivo sensoriale per la raccolta dei dati sulla qualità dell'aria, unitamente alle tecnologie e alle librerie utilizzate per lo sviluppo della dashboard web per la visualizzazione di tali dati, oggetto del presente lavoro di Tesi.

Capitolo 2

Strumenti e tecnologie utilizzate

2.1 Nodo Sensoriale IoT

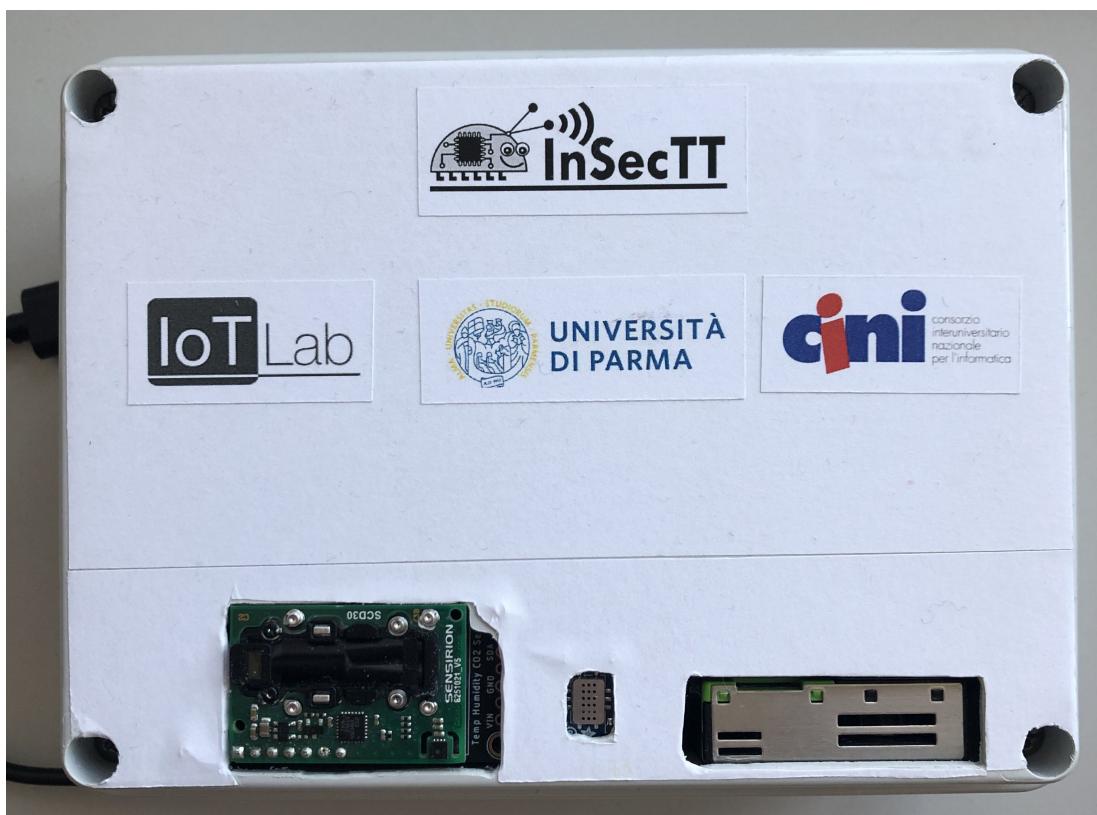


Figura 2.1: Nodo sensoriale IoT.

Come nodo per il monitoraggio dei parametri di qualità dell'aria, nel presente lavoro di Tesi è stato adottato il nodo sensoriale mostrato in figura. In dettaglio, ogni nodo IoT utilizzato per la raccolta dei dati è composto da un *Raspberry PI 4*, utilizzato come *Single Board Component SBC*, un *Adafruit MiCS5524* per il sensing della concentrazione di gas in modo “cumulativo”, perchè purtroppo non restituisce la singola concentrazione di ogni gas, un *Sensirion SCD30* per il sensing di temperatura ed umidità dell'aria e della concentrazione della CO₂ e infine un *Sensirios SPS30* per la rilevazione delle PM.

2.1.1 Raspberry PI 4

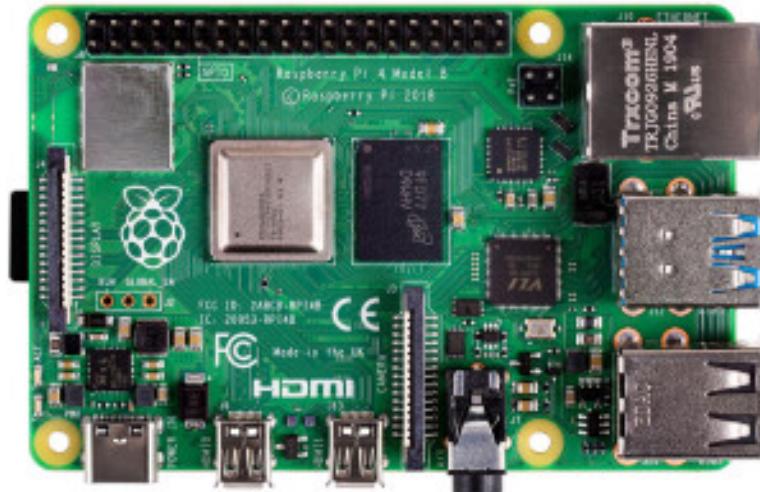


Figura 2.2: Raspberry PI 4.

Come anticipato, come SBC all'interno del nodo sensoriale IoT è stato scelto un Raspberry PI 4 con funzione di dispositivo di supporto, ovvero quello che fa funzionare tutti gli altri dispositivi e garantisce un corretto comportamento degli apparati fra di loro.

2.1.2 Sensore Adafruit MiCS5524

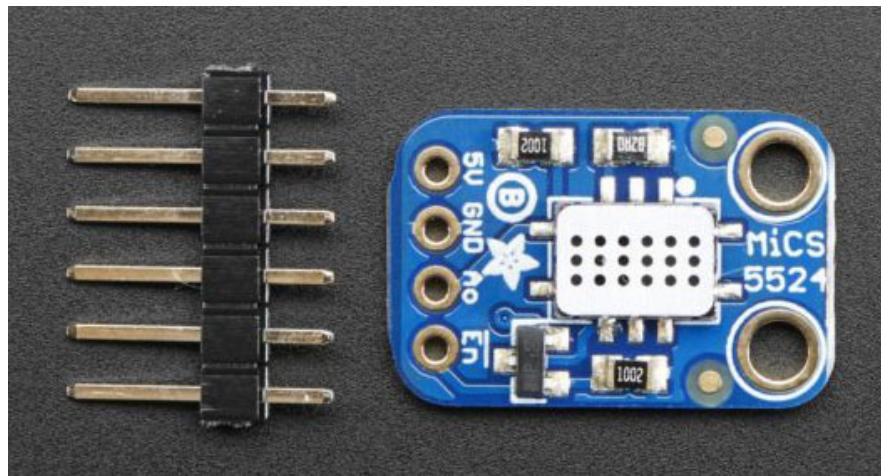


Figura 2.3: Sensore Adafruit MiCS5524.

Il sensore Adafruit MiCS5524, mostrato in Figura 2.3 è utilizzato per la raccolta delle informazioni sui gas che sono presenti nell'aria, ed è in grado di restituire un valore aggregato di concentrazione di gas presenti nell'ambiente monitorato dal nodo IoT. Non essendo in grado di restituire i valori dei singoli gas a cui è sensibile, anche analisi successive legate a tali dati dovranno tenere conto di tali caratteristiche.

2.1.3 Sensore Sensirion SCD30

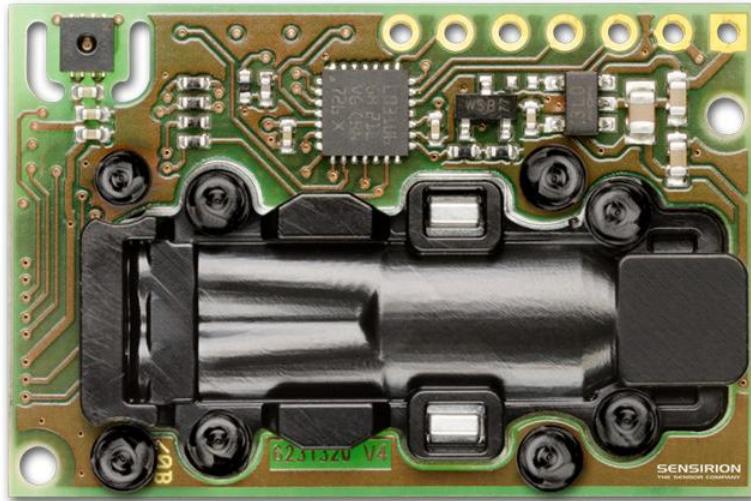


Figura 2.4: Sensore SCD30.

Il sensore Sensirion SCD30, mostrato in Figura 2.4, è stato integrato dal nodo IoT con la funzione di raccolta di valori puntuali relativi a CO₂ ed umidità dell’aria.

Per completezza, nelle Figure 2.5 e 2.6, si riportano alcuni estratti del datasheet del sensore, contenenti le relative caratteristiche principali.

CO₂ Sensor Specifications

Parameter	Conditions	Value
CO ₂ measurement range	I2C, UART PWM	0 – 40'000 ppm 0 – 5'000 ppm
Accuracy ²	400 ppm – 10'000 ppm	± (30 ppm + 3%MV)
Repeatability ³	400 ppm – 10'000 ppm	± 10 ppm
Temperature stability ⁴	T = 0 ... 50°C	± 2.5 ppm / °C
Response time ⁵	$\tau_{63\%}$	20 s
Accuracy drift over lifetime ⁶	400 ppm – 10'000 ppm ASC field-calibration algorithm activated and SCD30 in environment allowing for ASC, or FRC field-calibration algorithm applied.	± 50 ppm

Figura 2.5: SCD30 CO2.

Humidity Sensor Specifications⁷

Parameter	Conditions	Value
Humidity measurement range	-	0 %RH – 100 %RH
Accuracy ⁸	25°C, 0 – 100 %RH	± 3 %RH
Repeatability ⁹	-	± 0.1 %RH
Response time ⁵	$\tau_{63\%}$	8 s
Accuracy drift	-	< 0.25 %RH / year

Table 2: SCD30 humidity sensor specifications

Figura 2.6: Sensore Adafruit MiCS5524.

2.1.4 Sensore Sensirion AG SPS30

Figura 2.7: Sensirion AG SPS30.

Il sensore Sensirion AG SPS30, mostrato in Figura 2.7, è utilizzato per la raccolta dati relativi ai livelli di concentrazione delle PM, in particolare, come anticipato, relativamente a PM0.5, PM10, PM2.5, PM4 E PM10.

Come mostrato in sezione 2.1.4, anche per questo sensore si riportano le principali caratteristiche, tratte dal datasheet ufficiale.

Parameter	Conditions	Value	Units
Mass concentration range	-	0 to 1'000	$\mu\text{g}/\text{m}^3$
Mass concentration size range	PM1.0	0.3 to 1.0	μm
	PM2.5	0.3 to 2.5	μm
	PM4	0.3 to 4.0	μm
	PM10	0.3 to 10.0	μm
Mass concentration precision ^{1,2} for PM1 and PM2.5 ³	0 to 100 $\mu\text{g}/\text{m}^3$	± 10	$\mu\text{g}/\text{m}^3$
	100 to 1000 $\mu\text{g}/\text{m}^3$	± 10	% m.v.
Mass concentration precision ^{1,2} for PM4, PM10 ⁴	0 to 100 $\mu\text{g}/\text{m}^3$	± 25	$\mu\text{g}/\text{m}^3$
	100 to 1000 $\mu\text{g}/\text{m}^3$	± 25	% m.v.
Maximum long-term mass concentration precision limit drift	0 to 100 $\mu\text{g}/\text{m}^3$	± 1.25	$\mu\text{g}/\text{m}^3 / \text{year}$
	100 to 1000 $\mu\text{g}/\text{m}^3$	± 1.25	% m.v. / year
Number concentration range	-	0 to 3'000	#/ cm^3
Number concentration size range	PM0.5	0.3 to 0.5	μm
	PM1.0	0.3 to 1.0	μm
	PM2.5	0.3 to 2.5	μm
	PM4	0.3 to 4.0	μm
	PM10	0.3 to 10.0	μm
Number concentration precision ^{1,2} for PM0.5, PM1 and PM2.5 ³	0 to 1000 #/ cm^3	± 100	#/ cm^3
	1000 to 3000 #/ cm^3	± 10	% m.v.
Number concentration precision ^{1,2} for PM4, PM10 ⁴	0 to 1000 #/ cm^3	± 250	#/ cm^3
	1000 to 3000 #/ cm^3	± 25	% m.v.
Maximum long-term number concentration precision limit drift ²	0 to 1000 #/ cm^3	± 12.5	#/ $\text{cm}^3 / \text{year}$
	1000 to 3000 #/ cm^3	± 12.5	% m.v. / year
Sampling interval	-	1±0.04	s

Figura 2.8: Scheda Tecnica Sensiron AG SPS30.

Capitolo 3

Architettura Software Del Sistema

Il presente lavoro di Tesi ha previsto una prima fase di progettazione legata all’analisi degli strumenti software migliori per la rappresentazione dei dati e delle informazioni. La soluzione migliore è risultata essere una *Web Application*. I motivi legati a questa scelta sono differenti, fra i quali i principali risultano essere l’elevata disponibilità di librerie software per la rappresentazione di grandezze eterogenee e la possibilità di disporre di un software online per l’utenza finale senza dover prevedere pre-requisiti in termini di caratteristiche della piattaforma. Terminata la fase di analisi delle modalità di visualizzazione, si è resa necessaria un’altra fase di analisi relativa alla simulazione dei dati da calcolare e analizzare. Nel dettaglio la simulazione è stata generata mediante dei dati di prova per avere un’idea più precisa sulla struttura dei dati che l’applicativo dovrà elaborare, ossia tre file con circa 40000 righe di dati ciascuno, ognuno dei quali contenente 17 colonne diverse. I dati di prova totali con cui è stata modellata la dashboard sono stati quindi circa 2 milioni, la cui gestione è risultata quindi essere essenziale all’interno di un database per poi essere usati dall’applicazione. Quest’ultima fase ha quindi previsto l’esecuzione di alcuni *benchmarks* su alcuni tipi di database per valutare quale si sarebbe prestato meglio (in termini di prestazioni) per la gestione dei dati.

3.1 Database relazionali e non relazionali

3.1.1 Database Relazionale MySQL

L'attività di *benchmark* è stata eseguita tra due tipi di database, uno SQL e uno NoSQL, in modo tale da poter disporre di un confronto operativo e prestazionale fra le due più grandi classi di database note. Per i *benchmarks* è stato utilizzato uno script sviluppato in linguaggio Python come componente integrativo del presente lavoro di Tesi opportunamente per questo scopo. Il funzionamento dello script consiste nel generare due liste contenenti i valori che verranno usati per testare le prestazioni dei database. Per la versione dello script che valuta le prestazioni del database MySQL si è reso necessario l'utilizzo delle liste di tuple, in quanto *mysql.connector* (libreria utilizzata per la connessione con il database MySQL) richiede i dati sotto forma di tuple. Nel dettaglio la prima lista contiene una tupla composta solamente da tre valori, in modo da testare il database con una quantità bassa di informazioni per richiesta, mentre l'altra lista contiene tuple contenenti, a loro volta, 17 valori di tipo *String*, *Date* e *Integer*. Le liste vengono generate quindi con i medesimi valori ma in numero variabile, eseguendo i test con 50.000 tuple per lista.

Come primo passaggio, lo script Python definito per le attività di benchmarking verifica la connessione al server che si vuole testare. Nel caso in cui la connessione vada a buon fine, lo script prova a connettersi effettivamente con il database e se anche questo ha esito positivo esegue i primi test.

Il primo test che viene effettuato consiste nell'inserimento (mediante un *statement* di tipo `INSERT`) di 50000 dati con il seguente payload:

```
( "John", 35, 1.80 )
```

In questo specifico test, però, l'`INSERT` viene eseguito *one-by-one*, ovvero viene eseguita ricorsivamente una operazione di `INSERT` alla volta per un numero di occorrenze pari a *nvalues*, come dettagliato nel pezzo di codice seguente.

Ne consegue come questo test non sia molto indicato per verificare la velocità massimo di `INSERT` di un database; per questo non risulta essere molto utile qualora si debbano caricare un numero elevato di dati in contemporanea all'interno di un data-

```
while i < nvalues do
    mycursor.execute(sql, mytuple);
    i += 1;
end
```

base, mentre risulta essere utile, oltre che come effettivo calcolo delle prestazioni del database sui singoli INSERT, anche come caso reale. Questo perchè quando arrivano nuovi dati e questi debbano essere caricati per la prima volta su un database, allora l'utilità svanisce, mentre quando dovranno essere elaborati singolarmente e, dopo una eventuale elaborazione, dovranno essere aggiornati sul database, probabilmente si renderà necessario un caricamento singolo, quindi è utile conoscere anche le prestazioni delle singole INSERT all'interno del database MySQL.

Il secondo test, invece, prevede l'INSERT della medesima quantità di dati e del medesimo *payload* ma con un altro tipo di approccio, ovvero non si procede nell'inserimento dei dati *one-by-one*, ma tutti simultaneamente. Nel dettaglio non viene usato un ciclo while come nel primo caso ma viene sfruttata una funzione (`executemany()`) a cui viene passata come primo parametro la query SQL priva di valori, e come secondo parametro la lista di valori da inserire. La funzione effettuerà tutte le INSERT in modo automatico, finalizzando quindi il tutto mediante la funzione `commit()`. Questo processo, come poi verrà confermato dai risultati del *benchmark*, consente una velocità molto maggiore rispetto all'INSERT *one-by-one*, quindi ha una maggiore utilità nel momento in cui si debbano caricare elevate quantità di dati all'interno del database: un esempio è rappresentativo dal momento in cui vi sia la necessità di caricare tutti i dati estratti sul database per poi essere elaborati dalla Web Application.

Il terzo ed il quarto test sono identici rispettivamente al primo e al secondo, con una variazione del payload. Questo è stato fatto per testare le prestazioni del database anche con un payload con dimensione maggiore, in modo non solo da rendere più veritiero il test per quanto riguarda la casistica reale del presente lavoro di Tesi, ma anche per valutare quanto la dimensione del singolo payload influisca sulle prestazioni del database MySQL. In questo caso il payload è costituito da una tupla contenente ben 17 dati, molto simile alla struttura dei dati provenienti, come spiegato nel capitolo

2, dai nodi IoT.

```
( "Y13vRk5zFoh6wjkr", "Y13vRk5zFoh6wjkrY13vRk5zFoh6wjkr",
  1234567, date, "Y13vRk5zFoh6wjkr",
  "Y13vRk5zFoh6wjkrY13vRk5zFoh6wjkr",
  date, "Y13vRk5zFoh6wjkr",
  "Y13vRk5zFoh6wjkrY13vRk5zFoh6wjkr", date ,
  "Y13vRk5zFoh6wjkr", "Y13vRk5zFoh6wjkrY13vRk5zFoh6wjkr",
  date,
  "Y13vRk5zFoh6wjkr", "Y13vRk5zFoh6wjkrY13vRk5zFoh6wjkr",
  date )
```

dove il campo `date` viene sostituito dal `datetime` corrente di ogni acquisizione. A livello logico, a monte della valutazione dei risultati reali, è lecito aspettarsi che l'ordine dei vari test di performance in ordine di velocità dal più veloce al più lento, siano il seguente:

1. test #2, payload ridotto e utilizzo della funzione `executemany()`
2. test #4, payload elevato e utilizzo della funzione `executemany()`
3. test #1, payload ridotto e insertion *one-by-one*
4. test #3, payload elevato e insertion *one-by-one*

Per essere realistici i test devono essere eseguiti su piattaforme il più simili possibili. Per questo motivo si è scelto di utilizzare Docker come piattaforma su cui eseguire i database, così da avere sempre le stesse caratteristiche dell'emulatore del database. La versione di Docker al momento dei test era Docker Desktop 4.12.0 (85629) [1], mentre il database MySQL era aggiornato all'ultima versione disponibile, ovvero MySQL 8.0.0. A livello operativo, il test per la valutazione delle performance di MySQL ha previsto l'esecuzione di Docker MySQL con il comando:

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD=password -p
3306:3306 -v mysql_data:/var/lib/mysql -d mysql:latest
```

Per mezzo di questo comando viene creato un database MySQL esposto sulla porta 3306 (default) con l’ultima versione di MySQL e la password dell’account *root* impostata a *password*. Una volta creato il database è possibile procedere con l’effettivo test, lanciando lo script per il test tramite il seguente comando:

```
python3 db_benchmark_sql.py
```

A valle dell’esecuzione dello script Python, i risultati ottenuti sono riportati in seguito:

	Total elapsed time	About time per operation
Test #1	0:00:41.912810	0:00:00.000838
Test #2	0:00:00.774050	0:00:00.000015
Test #3	0:00:53.679391	0:00:00.001074
Test #4	0:00:04.112547	0:00:00.000082

Ad una prima analisi dei risultati appare chiaro come l’ordine dei risultati in base alla velocità rispetti le aspettative; analizzandoli meglio è possibile notare alcune cose.

Si può notare come il Test #1 e il Test #3, che si differenziano solamente per la dimensione del payload (mantenendo immutato il tipo di inserimento, *one-by-one*, per entrambi), hanno una differenza di tempo pari al 22%, ovvero il Test #1 è il 22% più veloce rispetto al Test #3. Mentre analizzando il Test #2 e il Test #4, che differiscono come nel caso precedente solamente nel tipo di payload e hanno come tipo di inserimento la funzione `executemany()`, si nota una differenza in termini di velocità del 433,77%. Quindi il fattore che incide principalmente sulla velocità dell’inserimento risulta essere non tanto la dimensione del payload, ma quanto più il tipo di funzione utilizzata. Questo può essere confermato confrontando anche i valori degli altri Test con eguale payload ma differente metodo di inserimento. Confrontando il Test #1 e #2 che hanno entrambe il payload ridotto ma differenti metodi di inserimento, la differenza fra i due è pari al 5342%. Quello che prevede l’esecuzione della funzione `executemay()` è più veloce del 5342%. Successivamente, confrontando anche il Test #3 e il Test #4 che hanno entrambi il payload maggiore e differente metodo di inserimento, anche in questo caso la differenza risulta essere importante, nell’ordine del 1225%. I risultati vengono rafforzati dal rapporto che c’è fra le rispettive percentuali di velocità, siccome la funzione `executemay()` risulta essere circa 4,3 volte più efficiente del inserimento *one-by-one*.

3.1.2 Database MongoDB

Completato il *benchmark* sul database MySQL, sono stati eseguiti esperimenti simili sul database non relazionale MongoDB. Lo script utilizzato risulta essere similare, con alcune variazioni legate al fatto che MongoDB richiede i dati con una struttura differente (sotto forma di *dictionary* e non di *tuple*). In modo similare a *mysql.connector* anche *pymongo* (libreria Python per la connessione con MongoDB) dispone di due funzioni per l'INSERTION proprio come per MySQL. La prima è *insert_one()* che si comporta come l'*insertion one-by-one* di MySQL:

```
while i < nvalues do
    db_collection.insert_one(mydict);
    i += 1;
end
```

Invece, la funzione equivalente alla funzione *executemany()* di MySQL è *insert_many()* che, come per MySQL, richiede una lista come parametro contenente dei *dictionary*. I due payload utilizzati sono i medesimi adattati per il *benchmark* sul database MySQL. Anche per questo test è possibile “prevedere” (basandosi anche sui test precedenti) quale sarà l'ordine finale relativo alla velocità:

1. test #2, con payload ridotto e utilizzo della funzione *insert_many()*
2. test #4, con payload elevato e utilizzo della funzione *insert_many()*
3. test #1, con payload ridotto e insertion *insert_one()*
4. test #3, con payload elevato e insertion *insert_one()*

Come per i test con il database MySQL, anche nel caso delle valutazioni di performance di MongoDB è stato adottato il gestore di container Docker, nella versione Docker Desktop 4.12.0 (85629) [1]. Il database MongoDB era aggiornato all'ultima versione disponibile, Mongo_version 6.0.1. Similmente a quanto fatto per MySQL, anche nel

caso di MongoDB il container è stato lanciato con il seguente comando:

```
docker run --name mongodb -d -p 27017:27017 mongo:latest
```

Grazie al comando viene creato un database MongoDB esposto sulla porta 27017 (default) con l'ultima versione di MongoDB e la password dell'account *root* impostata a *password*. Una volta creato il database, è stato possibile procedere con l'effettivo test. Lanciando lo script Python per il test con il comando:

```
python3 db_benchmark_mongodb.py
```

I risultati dell'esecuzione dei differenti test sono riportati in seguito.

	Total elapsed time	About time per operation
Test #1	0:00:46.441054	0:00:00.000929
Test #2	0:00:00.384103	0:00:00.000008
Test #3	0:00:47.638675	0:00:00.000953
Test #4	0:00:01.186850	0:00:00.000024

Come era lecito attendersi, l'ordine corrisponde con quanto descritto in precedenza, ma permane la necessità di valutare l'effettiva velocità dei singoli test e se vi sia una differenza rispetto a MySQL. Il test #1, ovvero quello con il payload ridotto e la funzione *insert_one()*, risulta essere più veloce su MySQL di circa l'11%. Confrontando i test #2, ovvero quelli con payload ridotto e *insert_many()*, in questo caso invece risulta essere più veloce il test su MongoDB per circa il 102%. Passando al test #3, ovvero quello con payload elevato e la funzione *insert_one()* risulta essere più veloce quello con MongoDB di circa il 12%. Passando al test #4, ovvero quello con il payload elevato e la funzione *insert_many()*, risulta essere più veloce quello con MongoDB di circa il 248%. A parte il test #1, in cui l'esecuzione risulta essere più veloce su MySQL, per tutti gli altri test MongoDB si rivela essere la soluzione più rapida. Considerando che, in termini di tipologia di dati trattati nel presente lavoro di Tesi, il caso comune è quello con un payload elevato è stato fatto appositamente per essere il più simile possibile a quello che verrà poi utilizzato nella Web Application, risulta opportuno valutare in base a questo specifico test. Andando a confrontare come è stato fatto il rapporto che c'è fra i due test con il payload grande e scegliendo chiaramente quello con l'`INSERT`

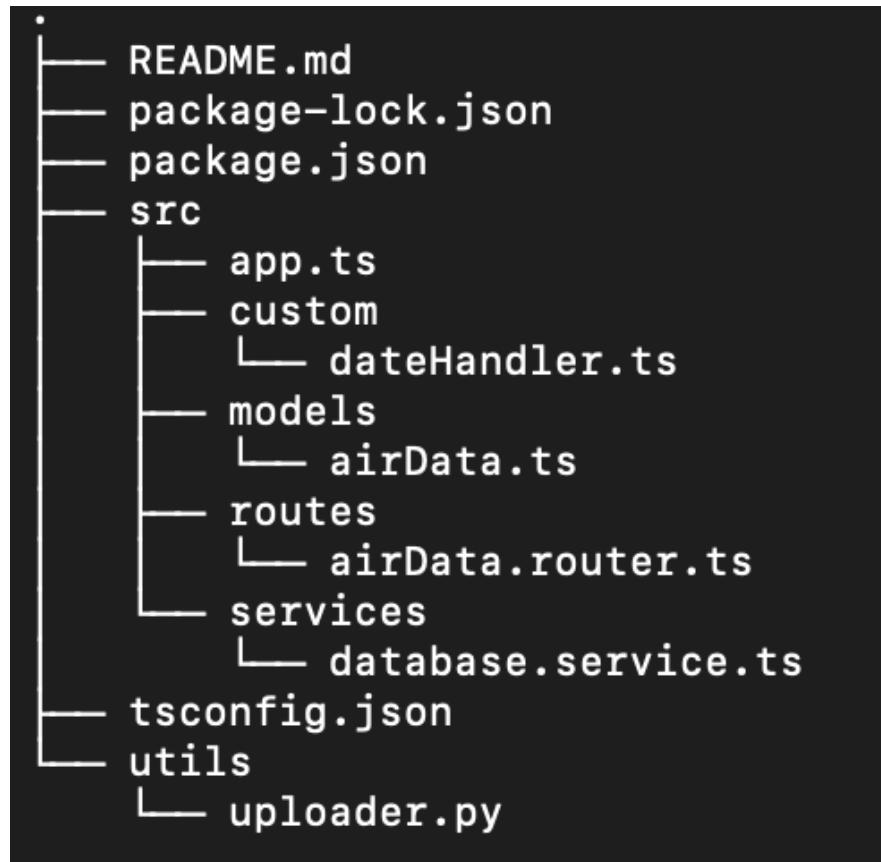
più veloce, utilizzare MongoDB risulta essere la scelta migliore, restituendo una performance in termini di velocità pari al 248%.

3.2 Web Application

La Web Application sviluppata nel presente lavoro di Tesi e che consentirà di elaborare e analizzare i dati forniti dai rilevatori della qualità dell'aria si baserà su un backend realizzato in linguaggio Typescript [7]. La scelta di Typescript è dovuta al fatto che è una delle migliori soluzioni attualmente disponibili per lo sviluppo delle Web Application, ed il motivo principale per cui viene utilizzata in modo così diffuso sta nel fatto che è essenzialmente Javascript. Nel dettaglio, una volta compilato, Typescript è Javascript quindi non presenta problemi di integrazione con i browser moderni, in più garantisce un controllo sui tipi delle variabili di cui Javascript non dispone. Inoltre, in Typescript è possibile dichiarare il tipo della variabile così da garantire una consistenza maggiore nel codice e evitare problemi con la gestione del codice stesso.

3.2.1 Backend

Il seguente codice, sviluppato nel presente lavoro di Tesi per la parte di backend si compone di una struttura ad albero formata come segue:



Come anticipato nel capitolo 2, le librerie utilizzate per la definizione e l'implementazione della componente di backend del presente lavoro di Tesi sono le seguenti:

1. express [4]
2. mongodb [5]
3. dotenv [2]
4. nodemon [6]

In dettaglio, la connessione dal database MongoDB viene effettuata da `mongoDB.MongoClient` all'interno del file `database.service.ts`

```
const client: mongoDB.MongoClient = new
    mongoDB.MongoClient(process.env.MONGODB_URL || '');
```

```
await client.connect();
const db: mongoDB.Db =
  client.db(process.env.MONGODB_DATABASE);
```

Il listato di codice precedente mostra come venga creata una nuova variabile *client* di tipo *mongoDB.MongoClient* che stabilisce la stringa di connessione con il server, poi successivamente, tramite una funzione asincrona *client.connect()* viene effettuata la connessione. L'ultima riga di codice serve per la connessione effettiva con il database. Per garantire una maggiore sicurezza sui dati inseriti nel database, viene utilizzato uno schema di controllo *jsonSchema* (mostrato nel listato successivo) che controlla che i dati sul database rispettino le linee guida impostate dentro tale schema, in modo da non permettere formati di dati non corretti.

```
$jsonSchema: {
  bsonType: "object",
  required: ["id_measure", "ads1115_value",
    "ads1115_voltage", "scd30_co2", "scd30_temp",
    "scd30_hum", "sps30_pm05_count",
    "sps30_pm1_count", "sps30_pm1_ug",
    "sps30_pm25_count", "sps30_pm25_ug",
    "sps30_pm4_count", "sps30_pm4_ug",
    "sps30_pm10_count", "sps30_pm10_ug",
    "sps30_pm_typ", "ts_insertion", "node"],
  additionalProperties: false,
  properties: {
    _id: {},
    id_measure: {
      bsonType: "string",
      description: "'id_measure' is required and is a
        string"
    },
    ads1115_value: {
      bsonType: "string",
```

```
        description: "'id_measure' is required and is a
                      string"
    },
    ads1115_voltage: {
        bsonType: "string",
        description: "'id_measure' is required and is a
                      string"
    },
    scd30_co2: {
        bsonType: "string",
        description: "'id_measure' is required and is a
                      string"
    },
    scd30_temp: {
        bsonType: "string",
        description: "'id_measure' is required and is a
                      string"
    },
    scd30_hum: {
        bsonType: "string",
        description: "'id_measure' is required and is a
                      string"
    },
    sps30_pm05_count: {
        bsonType: "string",
        description: "'id_measure' is required and is a
                      string"
    },
    sps30_pm1_count: {
        bsonType: "string",
        description: "'id_measure' is required and is a
                      string"
    },
}
```

```
sps30_pm1_ug: {
    bsonType: "string",
    description: "'id_measure' is required and is a
                  string"
},
sps30_pm25_count: {
    bsonType: "string",
    description: "'sps30_pm25_count' is required and
                  is a string"
},
sps30_pm25_ug: {
    bsonType: "string",
    description: "'sps30_pm25_ug' is required and is
                  a string"
},
sps30_pm4_count: {
    bsonType: "string",
    description: "'sps30_pm4_count' is required and
                  is a string"
},
sps30_pm4_ug: {
    bsonType: "string",
    description: "'sps30_pm4_ug' is required and is a
                  string"
},
sps30_pm10_count: {
    bsonType: "string",
    description: "'sps30_pm10_count' is required and
                  is a string"
},
sps30_pm10_ug: {
    bsonType: "string",
    description: "'sps30_pm10_ug' is required and is
```

```
        a string"
    },
    sps30_pm_typ: {
        bsonType: "string",
        description: "'sps30_pm_typ' is required and is a
                     string"
    },
    ts_insertion: {
        bsonType: "string",
        description: "'ts_insertion' is required and is a
                     string"
    },
    node: {
        bsonType: "string",
        description: "'node' is required and is a string"
    }
}
```

Lo schema di controllo (definito in formato JSON e mostrato in seguito) viene integrato con il modello della collezione di MongoDB, che è la struttura della collezione sul database riportata sul sistema di backend.

```
export default class AirData {
    constructor(public id_measure: String, public
                ads1115_value: String, public ads1115_voltage: String,
                public scd30_co2: String, public scd30_temp: String,
                public scd30_hum: String, public sps30_pm05_count:
                String, public sps30_pm1_count: String, public
                sps30_pm1_ug: String, public sps30_pm25_count: String,
                public sps30_pm25_ug: String, public sps30_pm4_count:
                String, public sps30_pm4_ug: String, public
                sps30_pm10_count: String, public sps30_pm10_ug: String,
                public sps30_pm_typ: String, public ts_insertion:
```

```
String, public node: String, public id?: ObjectId) {}  
}
```

Le definizioni precedenti corrispondono alla parte più “nascosta” del backend, ovvero quella che esegue i controlli senza essere veramente “esposta”. La parte che espone le API operative si trova nel file `airData.router.ts`. A questo riguardo, sono stati definiti diversi endpoint accessibili, il primo dei quali è quello che fornisce le chiavi del *dictionary* della collezione su MongoDB, raggiungibile al seguente endpoint:

```
/getKeys
```

Questo endpoint restituisce tutte le chiavi del *dictionary* utilizzati nella collezione in formato JSON con codice di risposta HTTP 200:

```
res.status(200).send(JSON.stringify(keys));
```

Un altro endpoint è il seguente, corrispondente alla root delle API:

```
/
```

gestito dal meccanismo di routing delle richieste come segue:

```
airDataRouter.get("/", async (req: Request, res: Response)  
  => {  
    const from = req?.query?.from;  
    const to = req?.query?.to;  
    let limit = req?.query?.limit;  
  
    if (limit === undefined){  
      limit = "30"; // to not overload the system  
    }  
    if (from! > to!){  
      res.status(500).send('error');  
      return;  
    }  
  })
```

```
try {

    if (from === undefined && to === undefined) {
        const airData = await collections.airData!.find({}),
            { projection: { _id: 0 }};
        ).limit(Number(limit)).toArray() as unknown as
        AirData[];
        res.status(200).send(airData);
    }
    else{
        const airData = await
            collections.airData!.find({ts_insertion: {$gt:
                from, $lte: to}}, { projection: { _id: 0
            }}).limit(Number(limit)).toArray() as unknown as
            AirData[];
        res.status(200).send(airData);
    }
}

} catch (error) {
    res.status(500).send('error');
}
});
```

Nel dettaglio, i tre parametri `from`, `to` e `limit` rappresentano rispettivamente la data iniziale, la data finale e il numero massimo di dati richiesti all'interno risposta dell'API. Inoltre, nel listato di codice dell'API viene impostato il `limit` pari a 30 se non viene esplicitamente impostato nella richiesta HTTP in modo da non sovraccaricare il client con una quantità troppo elevata di dati. Se le date non vengono inserite, invece, vengono restituiti i primi dati, mentre con le date inserite vengono restituiti quelli all'interno delle date selezionate. Gli altri endpoint sono simili, in quanto restituiscono tutti i dati con lo stesso principio e formato, ognuno dei quali però con una specifica tipologia di dati restituita. Per ogni singola key esiste un endpoint che filtra i dati con i filtri inseriti dall'utente e li restituisce.

Uploader

All'interno della cartella `utils` del codice sorgente della componente di backend del presente lavoro di Tesi si trova un file denominato `uploader.py`, che viene utilizzato per effettuare l'upload dei file `csv` nel database MongoDB. Nel dettaglio, il codice prevede due argomenti aggiuntivi, il primo corrisponde al percorso del file `.csv` da elaborare, mentre il secondo argomento corrisponde al nome del nodo IoT per il monitoraggio della qualità dell'aria da cui sono stati raccolti i dati. Basandosi su questi argomenti, l'uploader estraе i dati dal file e, come effettuato dagli script di benchmark, crea dei *dictionary* che aggiunge a una lista, che successivamente carica sul database MongoDB. Il processo di caricamento dei dati risulterebbe scomodo se dovesse essere fatto collegandosi al server di backend tramite SSH per poi, tramite un altro comando, caricare dal proprio PC al server il file in formato `csv` e infine eseguire lo script Python passandogli come parametro il file `csv`. Per ovviare a questo problema nel frontend di questo lavoro di Tesi verrà implementata una dashboard che consentirà, tra le sue funzionalità, di caricare i file in modo semplice.

```
let buffer = req.files myfile as
    fileUpload UploadedFile;
buffer.name = uuidv4();

buffer.mv('./uploads/' + buffer.name);
const spawn = require("child_process").spawn;
const pythonProcess =
    spawn('python3', ['./utils/uploader.py',
        './uploads/' + buffer.name, req.body.node]);
pythonProcess.stdout.on('data', (data: any) => {
    console.log(data.toString());
});
res.send({
    status: true,
    message: 'File is uploaded',
    data: {
```

```
    name: buffer.name  
}  
});
```

Il file viene caricato sul server e letto nel modo seguente:

```
let buffer = req.files myfile as  
fileUpload UploadedFile;
```

In dettaglio, il codice si occupa di lanciare lo script Python prendendo come argomenti il file caricato dall'utente (salvato all'interno della cartella `./uploads/`) e aggiungendovi `buffer.name` (ovvero il nome del file appena caricato) e il nome del nodo. Come è possibile notare nella seconda riga di codice dopo avere preso il file, `buffer.name` viene modificato dal nome originale del file caricato dall'utente con un nuovo identificativo, più precisamente un codice alfanumerico generato dalla funzione per la generazione di sequenze casuali `uuidv4()`. Questo viene fatto per due motivi principalmente: quello meno importante e meno rischioso per il sistema è che se per errore venissero caricati due file con lo stesso nome, il sistema potrebbe leggere quello non desiderato ma caricato precedentemente duplicando i dati. Mentre nel secondo caso, alla riga 6 si nota facilmente una *path traversal* negli argomenti passati al comando Python. Alla path `./uploads/` viene aggiunto `buffer.name`: essendo questo modificabile dall'utente e non sanificato, potrebbe portare a un rischio di sicurezza per il sistema. Probabilmente non ci sarebbero grossi danni al sistema perché lo script di upload dei dati si interromperebbe senza consentire l'esecuzione di codice arbitrario.

3.2.2 Frontend

La componente di frontend sviluppata nel presente lavoro di Tesi è stata realizzata con ReactJS, un framework molto utilizzato per lo sviluppo frontend dato che permette un comodo riutilizzo dei componenti. In ReactJS esiste la possibilità di creare dei componenti (quali, ad esempio, pulsanti, form, o qualsiasi altra parte di vista web) che poi potranno essere utilizzati in più viste. Questo consente di evitare la riscrittura di

codice ridondante in ogni singola pagina semplicemente richiamando il componente che verrà renderizzato sulla pagina visualizzata dall'utente in quel momento.

Dashboard

La vista principale della Web Application è la dashboard, che, come anticipato, permette di caricare un file .csv per importare i dati su database MongoDB, come dettagliato in sezione 3.2.1.

Grafici

Unitamente alla funzionalità che consente la selezione e l'upload di un file csv contentente i dati di qualità dell'aria campionati dai nodi IoT, la dashboard consente anche la gestione dei dati sulla qualità dell'aria e la loro visualizzazione sotto forma di grafici. Questo principalmente per riuscire a visualizzare e comprendere a prima vista quale sarà l'andamento della qualità dell'aria in determinati luoghi e avere la visualizzazione anche in base all'orario e al giorno di come cambiano questi dati. La libreria che è stata utilizzata per la gestione dei grafici è [@syncfusion/ej2-react-charts](#). Questa libreria si compone di molti componenti utili per la creazione di grafici di qualsiasi tipo. Nel presente lavoro di Tesi quelli più indicati sono il grafico a linee e quello a barre, i grafici a torta non sono utili perchè è importante vedere come cambiano i dati dello stesso tipo nel tempo, mentre non ha senso comparare dati di tipologia diversa con un grafico a torta.

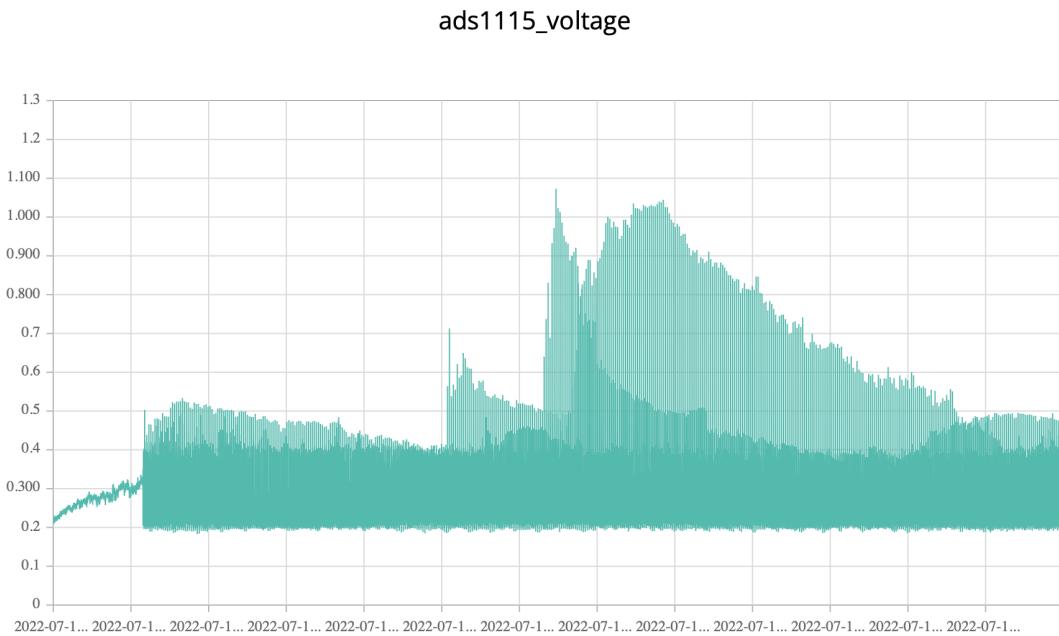


Figura 3.1: Andamento del parametro ads1115 voltage.

A titolo esemplificativo, in Figura 3.1 è riportato il grafico a linee che mostra i dati raccolti dai dispositivi dalle 7:50 alle 8:50. Tuttavia, è evidente che il grafico risulta difficile da comprendere a causa della consistenza dei dati. In particolare, si nota che all'inizio del grafico i dati sono comprensibili, ma successivamente diventano illeggibili. Questo problema è dovuto al fatto che i dati raccolti non sono presi a intervalli di tempo regolari. Fino alle 8:20 circa, i dati sono rilevati ogni due secondi, mentre successivamente vengono raccolti ogni pochi millisecondi. Ciò provoca un'eccessiva densità di dati nel grafico, rendendoli troppo vicini per poter essere compresi facilmente. Per affrontare questa problematica, si potrebbe considerare una scrematura dei dati prima di inserirli nel database, definendo un intervallo regolare di secondi e eliminando i dati eccedenti. Un'alternativa potrebbe essere gestire questa ottimizzazione nella parte di frontend, eliminando i dati in eccesso per rendere il grafico più chiaro. Queste soluzioni mirano a migliorare la leggibilità del grafico senza compromettere l'utilizzo dei dati nel backend per eventuali analisi più approfondite.

3.2.3 Gestione dei dati all'interno del grafico

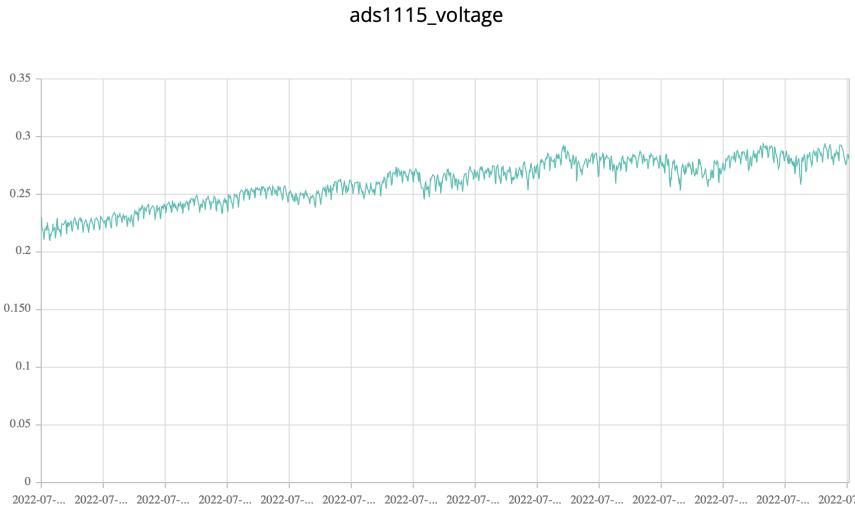


Figura 3.2: Andamento del parametro ads1115 voltage clean.

In Figura 3.2 è riportato il risultato del filtraggio effettuato dal frontend a monte del caricamento dei dati nel grafico, (mantenendo immutato l'intervallo temporale del grafico precedente). Il metodo adottato prevede di considerare massimo un dato ogni secondo, così da rendere i dati più uniformi e avere una visione chiara sul grafico.

```

const start = json[0].ts_insertion.substring(0,19);
const end = json[json.length-1].ts_insertion.substring(0,19);
let prec = start;
const newData = [];
newData.push(json[0]);
const jl = json.length;
for i = 1; i < jl; i ++
  if json[i].ts_insertion.substring(0,19) !== prec then
    newData.push(json[i]);
    prec = json[i].ts_insertion.substring(0,19);
  end
end
return newData;
  
```

Questa funzione chiamata `filterData` riceve come parametro `json` che è l'array di oggetti restituito dal backend. L'algoritmo prende due sottostringhe `start` e

end di 19 caratteri (rispettivamente la prima e l'ultima data restituite dal backend rimuovendo la parte dopo i secondi, ovvero i millisecondi). Quindi, all'interno della variabile `prec` viene salvato il valore di `start` e crea un nuovo array `newData` e inserisce come primo elemento il primo elemento di `json`. Successivamente per evitare di doverlo ricalcolare ad ogni giro mette in `l_j` la lunghezza dell'array `json` passato come parametro. All'interno del ciclo viene controllato se la data della misura fino al secondo è diversa da quella precedente, se è così allora aggiunge in `newData` quell'oggetto, altrimenti continua finché non trova un dato con una data differente. In questo modo viene selezionato al massimo un dato ogni secondo.

Per ora sono stati utilizzati solamente grafici a linea, che rendono bene l'idea dell'andamento, ma questa applicazione fornisce anche la possibilità di utilizzare i grafici a barre. Eseguendo a titolo esemplificativo una ricerca con i parametri seguenti:

```
Dataname: scd30_co2
From date: 2022-07-15 07:54:52.487
To date: 2022-07-15 10:54:52.487
Limit: 10000
Node: node01
```

Il risultato, in termini di plot a barre, visualizzato all'interno della dashboard è come in figura 3.3:



Figura 3.3: Andamento del parametro scd30 co2.

Anche in questo caso la comprensione dei dati richiede un'attenta analisi. Una soluzione potrebbe essere quella di adattare una strategia simile ma con un grafico a linee, rimuovendo i dati in eccesso senza rovinare o modificare il risultato finale, ma migliorandolo. Applicando la stessa “patch” applicata al grafico a linee il risultato appare in figura 3.4:



Figura 3.4: Andamento del parametro scd30 co2 clear.

Come è possibile notare osservando la Figura 3.4, il risultato ottenuto è migliore rispetto a quello precedente (però non ancora perfetto), in quanto l'estremità a destra sarebbe possibile di ulteriori miglioramenti. Per garantire la consistenza dei dati, è necessario utilizzare un intervallo di tempo uniforme per ogni singolo dato. Una possibile soluzione potrebbe consistere nell'utilizzare un dato ogni almeno due secondi. In questo modo, si otterebbe una regolarità nell'acquisizione dei dati e si eviterebbe la sovrapposizione e l'eccessiva densità nel grafico. Questo approccio consentirebbe una migliore comprensione dei dati e una visualizzazione più chiara delle tendenze nel grafico.

Eseguendo, in termini di miglioramento, un filtraggio con un dato ogni due secondi, il risultato ottenuto è mostrato in Figura 3.5:

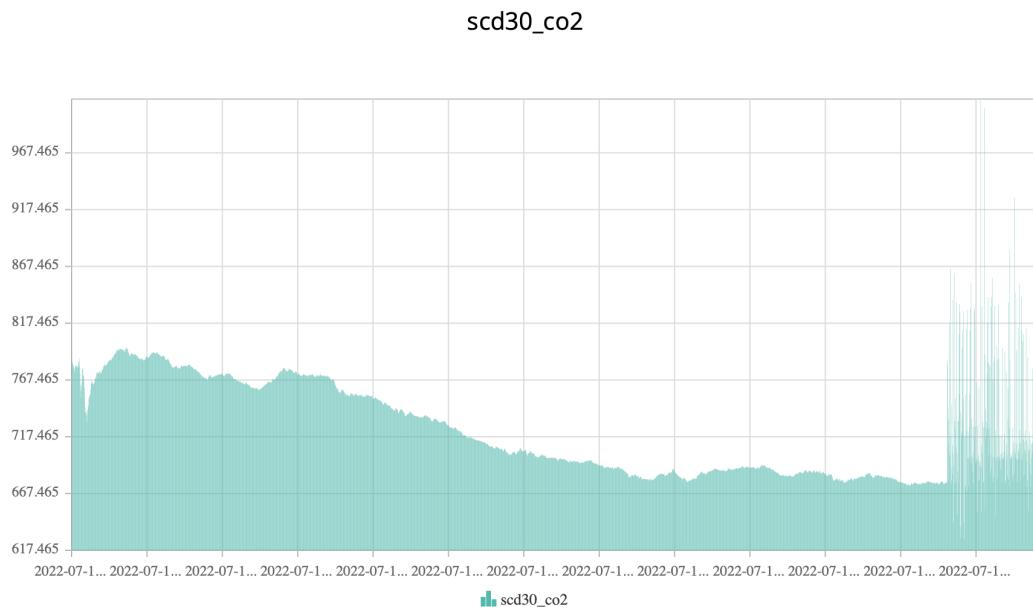


Figura 3.5: Andamento del parametro scd30 co2.

Il risultato sembra essere esattamente lo stesso, riguardo l'estremo destro del grafico sembra esserci ancora un problema. Andando ad analizzare i dati un po' più in profondità e dissezionando il grafico in intervalli temporali più ridotti, e più precisamente quello corrispondente alla parte finale del grafico, è possibile notare un comportamento particolare dei dati.

```
Dataname: scd30_co2
From date: 2022-07-15 09:40:52.487
To date: 2022-07-15 09:49:52.487
Limit: 10000
Node: node01
```

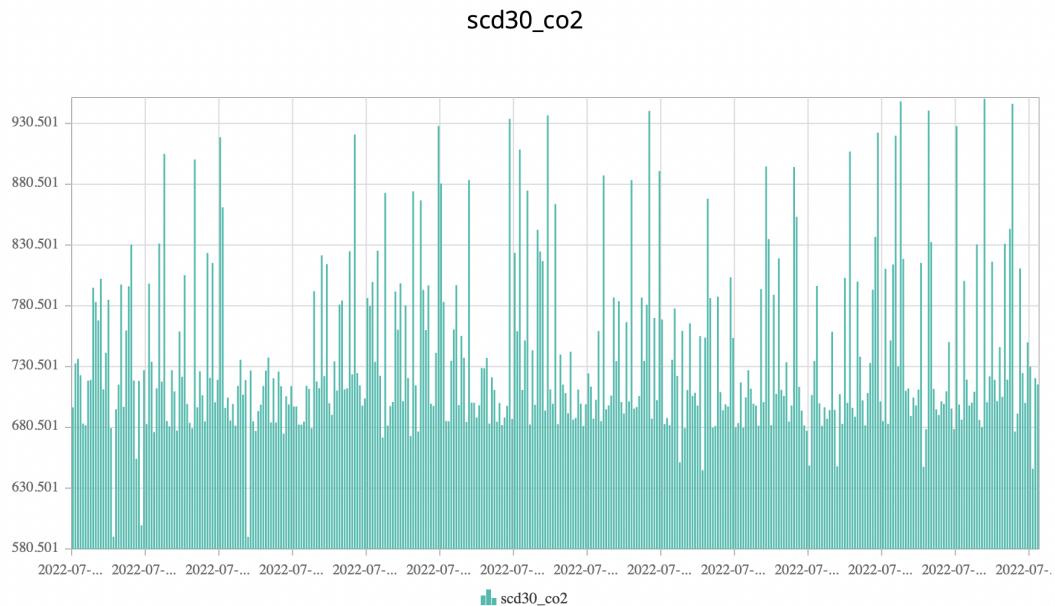


Figura 3.6: Andamento del parametro scd30 co2 small.

In Figura 3.6 sono mostrati gli ultimi 10 minuti del grafico precedente, è possibile notare come i valori richiedono analisi ulteriori, anche se il *gap* temporale è solo un secondo.

Provando a fare un confronto fra i dati di un solo minuto all'inizio e alla fine per vedere se effettivamente risulta essere così sempre o solo in alcuni momenti, ed utilizzando i valori seguenti all'interno dei filtri di selezione

Come dati e intervallo temporale sono stati utilizzati questi:

```
Dataname: scd30_co2
From date: 2022-07-15 09:48:52.487
To date: 2022-07-15 09:49:52.487
Limit: 10000
Node: node01
```

Il risultato che si ottiene è mostrato in Figura 3.7:

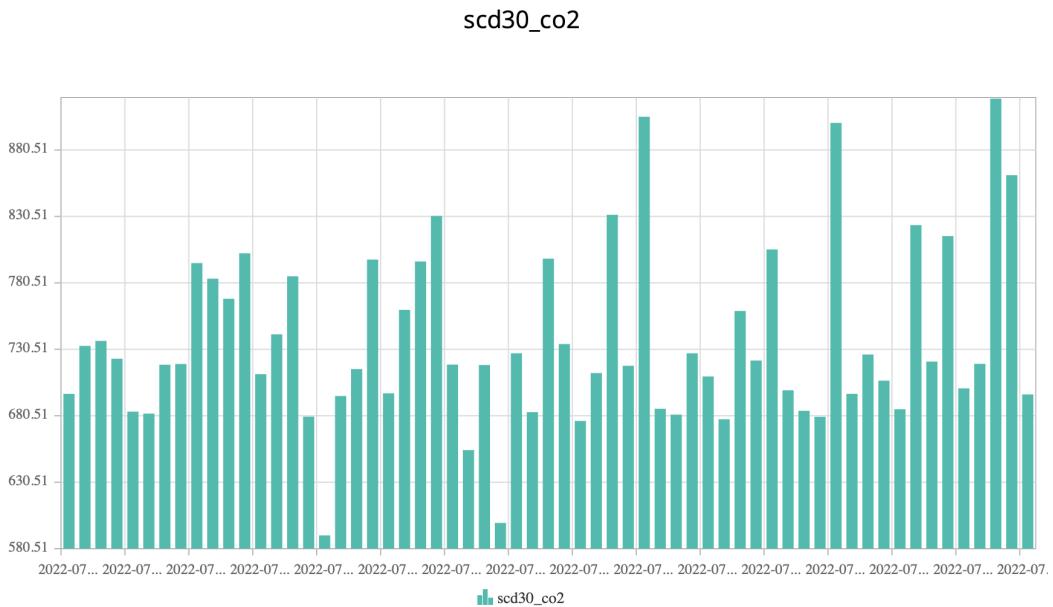


Figura 3.7: Andamento del parametro scd30 co2 1min end.

Si ha la conferma di come i dati siano irregolari, pensando anche che ogni singola barra corrisponde sempre a un secondo di differenza dalla barra precedente. In questo grafico il dato minore ha come valore pari a 590, mentre il dato massimo supera i 920. Si nota una differenza significativa di oltre 330 punti tra i valori. Inoltre, analizzando il dato con valore 590, si osserva che il valore precedente è 680, mentre quello successivo si avvicina a 700. Allo stesso modo, considerando il dato con valore 920, il valore precedente è 720, mentre quello successivo è 860. Evidentemente, ci sono degli errori nei dati che necessitano di correzione per garantire la coerenza e l'accuratezza delle informazioni.

```
Dataname: scd30_co2
From date: 2022-07-15 08:00:52.487
To date: 2022-07-15 08:01:52.487
Limit: 10000
Node: node01
```

Il risultato che si ottiene è mostrato in Figura 3.8:

scd30_co2

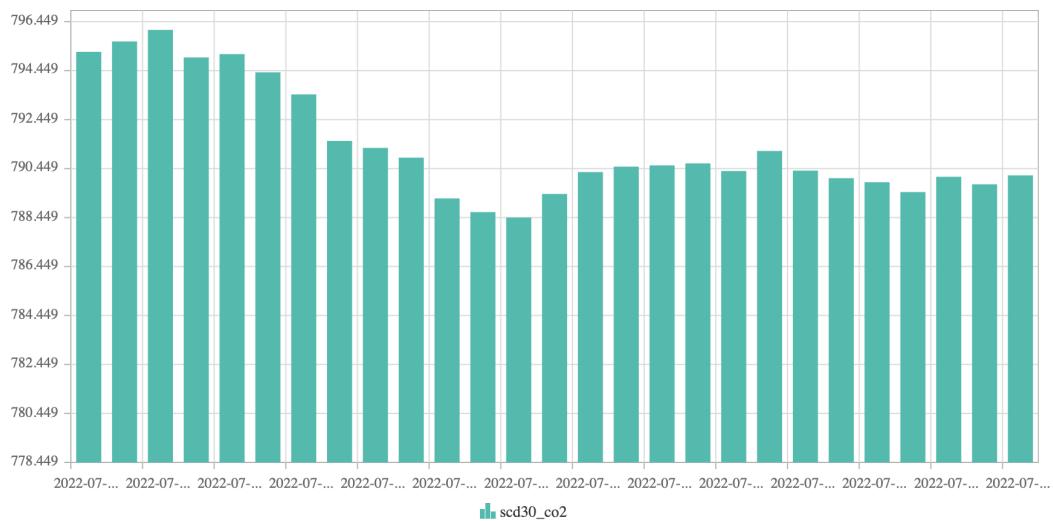


Figura 3.8: Andamento del parametro scd30 co2 1min beginning.

In questo caso, invece, i dati risultano essere maggiormente eterogenei, con assenza di picchi in aumento e diminuzione. Il dato minore ha valore pari a 788 mentre quello maggiore si attesta attorno al valore 796.

Dato predetto

Infine la dashboard è stata predisposta per essere in grado di visualizzare una serie aggiuntiva, corrispondente ai valori predetti da un algoritmo di AI, più precisamente un algoritmo di *machine learning* basato su reti neurali (al momento una *Recurrent Neural Network, RNN*) che predice in modo parametrizzabile sulla base di time series. Attualmente, l'algoritmo è in grado di predire il valore di un sensore in base ai valori precedenti, questa visualizzazione sarà pertanto utile anche per meglio sfruttare come la qualità dell'aria potrà evolvere.

Capitolo 4

Conclusioni e Sviluppi Futuri

Anche se il prodotto del presente lavoro di Tesi è già funzionante e offre nuove funzionalità rispetto ai prodotti già esistenti, essendo ancora in fase di sviluppo, il progetto presenta tuttora alcune lacune che verranno colmate in futuro. Inoltre, non solo sarà possibile migliorare le funzionalità già presenti in modo da renderle più efficienti e performanti, ma si potranno anche introdurre nuove funzionalità che permetteranno di rendere il prodotto più completo e competitivo.

Potranno essere integrate nuove funzionalità per quanto riguarda la gestione dei dati, come ad esempio, la possibilità di inserire nuovi dati, modificarli o eliminarli anche singolarmente e non solamente in gruppo come per la funzione di inserimento.

Introducendo anche algoritmi più avanzati di intelligence artificiale, si potrà migliorare la qualità delle previsioni e poterle fare anche su dati che ancora non sono in grado di essere analizzati dal sistema.

Un'altra possibile iniziativa potrebbe essere quella di portare il software, o una parte più ridotta, sotto forma di applicazione mobile, in modo da poter essere utilizzata anche da dispositivi mobili come smartphone e tablet, cambiando l'interfaccia grafica in modo da renderla più adatta a questo tipo di dispositivi.

Un importante sviluppo futuro per il progetto riguarda l'implementazione di un sistema di monitoraggio in tempo reale della qualità dell'aria. Attualmente, il prodotto del presente lavoro di Tesi fornisce informazioni sulla qualità dell'aria basate su dati pas-

sati e previsioni, consentendo agli utenti di prendere decisioni informate. Tuttavia, un sistema di monitoraggio in tempo reale consentirebbe di fornire aggiornamenti costanti sulla qualità dell'aria e di inviare notifiche tempestive agli utenti in caso di condizioni ambientali preoccupanti. L'implementazione di un sistema di monitoraggio in tempo reale per la qualità dell'aria richiederebbe l'uso di sensori avanzati collegati tra loro per raccogliere dati in tempo reale sulle condizioni ambientali. Questi sensori sarebbero posizionati strategicamente in diverse aree urbane e ambienti sensibili per ottenere una copertura completa delle condizioni dell'aria. Tramite una connessione costante, i sensori invierebbero costantemente informazioni a un sistema centrale, consentendo agli utenti di accedere a dati aggiornati in tempo reale. Per analizzare i dati in tempo reale, sarebbe necessario utilizzare potenti algoritmi di analisi e modelli di machine learning. Questi algoritmi esaminerebbero i dati in tempo reale per identificare rapidamente cambiamenti o tendenze significative nella qualità dell'aria. Sarebbe anche possibile impostare avvisi personalizzati in base a determinati parametri e inviare notifiche direttamente ai dispositivi mobili degli utenti o attraverso altri mezzi di comunicazione. Inoltre, un sistema di monitoraggio in tempo reale potrebbe essere integrato con altre piattaforme e servizi, come app per la salute o servizi di navigazione, per fornire informazioni contestuali agli utenti. Ad esempio, gli utenti potrebbero ricevere consigli sulla salute basati sulla qualità dell'aria nella loro area di residenza o di destinazione attuale, o potrebbero ricevere suggerimenti su percorsi alternativi per evitare zone inquinate. L'implementazione di un sistema di monitoraggio in tempo reale rappresenterebbe un importante progresso nell'offrire informazioni precise e tempestive sulla qualità dell'aria. Consentirebbe agli utenti di prendere decisioni più consapevoli e di adottare misure preventive per proteggere la propria salute e l'ambiente circostante. Tuttavia, è fondamentale sottolineare che la realizzazione di un sistema di monitoraggio in tempo reale richiede un'infrastruttura tecnologica adeguata, nonché una stretta collaborazione con enti governativi, organizzazioni ambientali e altri soggetti interessati. Il coordinamento e la condivisione dei dati in tempo reale rappresentano una sfida complessa, ma gli sforzi dedicati a questo sviluppo potrebbero portare a importanti benefici per l'intera società. In conclusione, l'implementazione di un sistema di monitoraggio in tempo reale rappresenta un passo significativo per

migliorare la qualità dell'aria. Consentirebbe agli utenti di accedere costantemente a informazioni aggiornate e di adottare misure preventive per mitigare gli effetti negativi dell'inquinamento atmosferico. Anche se richiede ulteriori ricerche, investimenti e collaborazioni, potrebbe rappresentare un contributo significativo per migliorare la salute e la qualità della vita delle persone.

Ringraziamenti

Desidero ringraziare la mia famiglia, i miei amici e i compagni del gruppo Havce per il loro costante supporto durante questi anni. Un ringraziamento speciale al mio Relatore per la sua guida preziosa e l'incoraggiamento che mi ha offerto. Sono profondamente grato a tutti voi per aver reso possibile il mio percorso universitario e avermi aiutato a crescere personalmente e professionalmente. Grazie di cuore!

Bibliografia

- [1] Docker, *Docker desktop release note*, Tech. report, Docker, 2022, <https://docs.docker.com/desktop/release-notes/#4120>.
- [2] dotenv, *dotenv*, <https://www.npmjs.com/package/dotenv>.
- [3] EPA, *Particulate matter (pm) basics*, 2022, <https://www.epa.gov/pm-pollution/particulate-matter-pm-basics>.
- [4] Express, *Expressjs*, <https://expressjs.com>.
- [5] MongoDB, *Mongodb*, <https://www.mongodb.com>.
- [6] nodemon, *nodemon*, <https://nodemon.io>.
- [7] Typescript, *TypeScript*, <https://www.typescriptlang.org>.
- [8] Super User, *Pm10 - particolato atmosferico o polveri sottili*, <https://www.issalute.it/index.php/la-salute-dalla-a-alla-z-menu/p/pm10-particolato-atmosferico-o-polveri-sottili#effetti-sulla-salute>.
- [9] Wikipedia, *Aria*, Oct 2022, <https://it.wikipedia.org/wiki/Aria>.