Politecnico di Torino

Politecnico di Torino

Laurea Magistrale in Ingegneria Informatica

A.a. 2023/2024

Sessione di laurea Aprile 2024

AirSense: un'applicazione mobile per monitorare la qualità dell'aria

Relatori:

Luigi De Russis

Andrea Venditti

Matteo Nisi

Candidato:

Luca Tamburo

Sommario

La crescente preoccupazione per l'inquinamento atmosferico, con le sue implicazioni sulla salute umana e sull'ambiente, ha stimolato l'implementazione di soluzioni pratiche e innovative. Questa tesi si propone di fornire una visualizzazione immediata e accessibile della qualità dell'aria a Torino. Dopo un'analisi approfondita dell'inquinamento atmosferico e degli standard internazionali, la ricerca si concentra sull'implementazione di un'applicazione *mobile*. Quest'ultima consente agli utenti di visualizzare i dati raccolti, il percorso ottimale da seguire basandosi sui livelli di PM10 e una *heatmap* sull'inquinamento. Per supportare tali funzionalità, è stato sviluppato un algoritmo per generare una mappa completa della città. Inoltre, il dispositivo di raccolta dati preesistente basato su Arduino è stato modificato per consentire un'associazione personalizzata del dispositivo a un utente.

La presente tesi offre soluzioni pratiche ed innovative per coinvolgere le persone nel monitoraggio della qualità dell'aria a Torino. Ciò rappresenta un prezioso aiuto per combattere l'inquinamento e promuovere la salute e la sostenibilità ambientale. Allo stesso tempo, si riflette sulle possibili strade future e interventi mirati, sottolineando l'importanza di affrontare l'inquinamento in modo attivo.

"Il successo non è mai definitivo,
il fallimento non è mai fatale;
è il coraggio di continuare che conta."

Winston Churchill

Indice

Elenco delle tabelle

Elenco delle figure

Elenco dei codici

Acronimi

API Application Programming Interface

ARPA Agenzia Regionale per la Protezione Ambientale

GPS Global Positioning System

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

LED Light Emitting Diod

MQTT Message Queuing Telemetry Transport

OMS Organizzazione Mondiale della Sanità

PM Particulate Matter

PWA Progressive Web App

UUID Universally Unique Identifier

Capitolo 1

Introduzione

L'aumento pervasivo dell'inquinamento atmosferico costituisce una sfida cruciale e sempre più complessa che la società contemporanea deve affrontare. Questo fenomeno minaccia non solo la qualità dell'aria che respiriamo, ma incide direttamente sulla salute umana e sull'equilibrio ambientale. Le città, epicentri delle attività umane, sono particolarmente vulnerabili a questa crescente problematica, con effetti avversi che si ripercuotono su milioni di persone. Al fine di affrontare questa sfida, i valori di inquinamento atmosferico sono attentamente monitorati da stazioni sparse sul territorio.

Tuttavia, per affrontare l'impatto dell'inquinamento atmosferico sull'individuo in modo più efficace e completo, è necessario andare oltre la semplice raccolta di dati. Sebbene il monitoraggio delle concentrazioni di inquinanti risulti fondamentale per comprendere l'entità del problema, fornire solamente dati grezzi risulta insufficiente nel proporre soluzioni pratiche e risposte immediate agli individui e alle comunità colpite dall'inquinamento atmosferico. La semplice consapevolezza della presenza di inquinanti nell'aria, sebbene importante, non è sufficiente per consentire alle persone di prendere decisioni per proteggere la propria salute e ridurre l'esposizione a tali agenti nocivi. Pertanto, è essenziale sviluppare uno strumento pratico e *user-friendly* che vada oltre la mera raccolta di dati e che permetta alle persone di interagire con queste informazioni in modo significativo e utile. Tale strumento potrebbe fornire non solo una semplice visualizzazione dei dati raccolti, ma anche una soluzione concreta per guidare gli utenti verso comportamenti più salutari e sostenibili. Inoltre, un'interfaccia intuitiva è fondamentale per garantire che queste informazioni siano comprensibili da tutte le persone, indipendentemente dal loro livello di alfabetizzazione digitale o conoscenza scientifica.

Il contesto urbano, con le sue molteplici fonti di inquinamento e la densità di popolazione, rappresenta uno scenario essenziale per analizzare e intervenire sull'inquinamento atmosferico. La città di Torino, con la sua storia industriale e la sua posizione geografica, offre un campo di studio significativo per comprendere

le dinamiche dell'inquinamento e sviluppare soluzioni innovative per mitigarne gli effetti negativi.

1.1 Obiettivi

Gli obiettivi della presente ricerca sono delineati al fine di sviluppare un'applicazione *mobile* innovativa ed efficace per affrontare l'inquinamento atmosferico, con particolare attenzione al contesto urbano di Torino. Questi obiettivi includono:

Sviluppo di una applicazione *mobile*: L'obiettivo primario è lo sviluppo di un'applicazione *mobile* intuitiva e funzionale che fornisca informazioni sulla qualità dell'aria a Torino. A supporto di tale applicazione, si è fatto uso di un dispositivo Arduino per la raccolta dati. L'applicazione consente agli utenti di visualizzare in modo chiaro e dettagliato i dati raccolti, nonché di interagire con i servizi offerti per acquisire consapevolezza sul fenomeno dell'inquinamento atmosferico e per ridurre l'esposizione agli agenti inquinanti.

Creazione della mappa di Torino: Realizzazione di un algoritmo per la creazione di un grafo che consenta di rappresentare la mappa completa della città di Torino. Tale implementazione è fondamentale per migliorare la precisione dei dati raccolti dal sensore GPS (*Global Positioning System*) posto sul dispositivo di raccolta dati Arduino.

Associazione dispositivo-utente: Sviluppare un sistema che consenta agli utenti di associare il proprio profilo utente al dispositivo di raccolta dati Arduino, facilitando così la visualizzazione e l'analisi dei dati tramite l'applicazione *mobile*.

1.2 Struttura della tesi

La tesi è suddivisa in nove capitoli, ciascuno dei quali si concentra su un aspetto specifico del progetto:

- **Capitolo 2 - Inquinamento dell'aria nelle città**: Fornisce una panoramica dettagliata sull'inquinamento atmosferico, analizzando le sue diverse forme e gli impatti sulla salute umana e sull'ambiente. Saranno esaminati gli standard e le direttive dell'Organizzazione Mondiale della Sanità (OMS), focalizzandosi sugli obiettivi e i limiti imposti. Inoltre, verrà esaminata la situazione attuale dell'inquinamento atmosferico sia a livello globale che nel contesto specifico dell'Italia.
- **Capitolo 3 - Stato dell'arte**: La presente ricerca si basa su una precedente tesi sul monitoraggio dell'inquinamento atmosferico. In questo capitolo, verrà fornita un'analisi approfondita dello stato dell'arte in merito al precedente studio, con un'attenzione particolare rivolta alle implementazioni presentate

nella tesi precedente. L'obiettivo primario è quello di acquisire una comprensione completa del contesto e delle sfide affrontate nella ricerca precedente, fornendo così una solida base per il lavoro attuale.

- **Capitolo 4 - Progettazione del sistema**: Descrive la progettazione dell'intero sistema di monitoraggio, fornendo una descrizione dettagliata delle tecnologie scelte, al fine di garantire una comprensione approfondita dell'intero sistema.

- **Capitolo 5 - Dispositivo di raccolta dati**: Descrive le migliorie *software* apportate al dispositivo di raccolta dati. Sarà analizzato il nuovo processo di inserimento dei dati nel database e il processo di associazione tra il dispositivo e l'utente. Tale funzionalità è fondamentale per due motivi: in primo luogo, rende scalabile l'uso di tale dispositivo, poiché prima non era possibile associare un utente a un percorso; in secondo luogo, consente agli utenti la possibilità di effettuare un'analisi approfondita dei propri percorsi passati.

- **Capitolo 6 - Algoritmo per la creazione della mappa di una città**: Esamina l'algoritmo per la creazione della mappa completa di Torino superando le limitazione della tesi precedente, al fine di consentire un'associazione più accurata delle coordinate fornite dal GPS del dispositivo di raccolti dati.

- **Capitolo 7 - Implementazione dei servizi**: Descrive i servizi implementati nell'applicazione. Tra questi vi sono il recupero dei percorsi precedenti, il calcolo del percorso meno inquinato e il recupero dei valori di inquinamento atmosferico per una determinata area.

- **Capitolo 8 - Implementazione dell'applicazione**: Si concentra sull'implementazione pratica dell'applicazione *mobile*, focalizzandosi sull'autenticazione degli utenti, la visualizzazione dei dati e l'interazione con i servizi offerti.

- **Capitolo 9 - Conclusioni**: Riassume il lavoro svolto e suggerisce possibili direzioni per lo sviluppo futuro del progetto, evidenziando le sfide e le opportunità che si presentano nel campo del monitoraggio dell'inquinamento atmosferico.

Capitolo 2

Inquinamento dell'aria nelle città

Per comprendere appieno l'inquinamento atmosferico, è essenziale fornire una definizione chiara di questo fenomeno. L'inquinamento atmosferico si riferisce alla presenza nell'atmosfera di agenti fisici, chimici e inquinanti biologici che alterano le caratteristiche naturali di quest'ultima. Questi agenti, accumulandosi nel tempo, possono causare danni sia all'ambiente che alla salute umana. Tale problematica, in costante crescita, trova le sue radici principalmente nelle attività umane, caratterizzate da un'intensa emissione di inquinanti atmosferici, tra cui gas e particelle sottili. Tali inquinanti provengono da fonti industriali, impianti energetici, sistemi di riscaldamento e traffico veicolare [1].

2.1 Impatto sulla salute

Tra gli inquinanti atmosferici più dannosi figurano i biossidi di azoto e zolfo, l'ozono, il monossido di carbonio e il particolato. Un'attenzione particolare è rivolta alle diverse polveri sottili presenti nell'aria che l'uomo respira, note come *Particulate Matter* (PM), le quali risultano particolarmente dannose per la salute umana. In particolare, il PM10 fa riferimento a particelle con un diametro inferiore a 10 μm, mentre il PM2.5 si riferisce a particelle con un diametro inferiore a 2.5 μm; infine, il PM1, considerato il più pericoloso, si riferisce alle particelle con un diametro inferiore a 1 μm. La loro pericolosità è legata alla loro dimensione ridotta, che consente loro di superare le difese naturali dell'organismo umano, causando effetti dannosi principalmente a livello polmonare [2]. Per comprendere quanto siano piccole queste particelle rispetto agli oggetti comuni, è sufficiente considerare che un capello umano ha un diametro compreso tra i 50 e i 70 μm, mentre un granello di sabbia misura circa 90 μm. L'esposizione a tali polveri può avere un impatto

immediato sul nostro corpo, manifestandosi con irritazione agli occhi, al naso e alla gola, problemi respiratori, mal di testa, stanchezza e bassa concentrazione. Inoltre, può anche causare malattie gravi che si manifestano nel lungo periodo. Numerosi studi hanno dimostrato come l'esposizione prolungata a tali inquinanti sia correlata all'incremento di malattie cardiache e respiratorie, con un significativo aumento, ad esempio, di ictus e infarti. Nei bambini, può causare una ridotta crescita e funzionalità polmonare, infezioni respiratorie e asma aggravata. Nel 2013, l'Agenzia Internazionale per la Ricerca sul Cancro ha classificato l'inquinamento e i particolati tra i cancerogeni di gruppo 1, ossia agenti sicuramente cancerogeni per l'uomo [3].

2.2 Situazione mondiale e italiana

L'OMS svolge un ruolo chiave nel monitorare e affrontare l'inquinamento atmosferico, fornendo linee guida e raccomandazioni per mitigarne gli impatti sulla salute pubblica. L'organizzazione stima che ogni anno l'esposizione all'inquinamento atmosferico provochi sette milioni di morti premature. Nel 2021, ha rilasciato il documento "WHO Global Air Quality Guidelines (AQGs)", che presenta nuove linee guida più stringenti rispetto al quelle rilasciate nel 2005, riducendo quasi tutti i limiti precedentemente imposti al fine di proteggere la salute delle popolazioni. Queste nuove linee guida stabiliscono limiti annuali inferiori per la concentrazione di polveri sottili nell'aria, riducendo i valori da 10 a 5 μ/m^3 per il PM2.5 e da 20 a 15 $\mu g/m^3$ per il PM10 [4]. La Figura 2.1 mostra i livelli raccomandati di alcuni inquinanti, gli obiettivi intermedi e i valori passati e presenti.

Per valutare la gravità dell'inquinamento atmosferico, è essenziale confrontare i nuovi limiti dell'OMS con i dati effettivi registrati nel 2023. Secondo il report annuale pubblicato da IQAir, solo il 9% dei Paesi nel mondo rispetta tali linee guida, evidenziando una situazione critica. Tra essi, solo 7 Paesi su 134 mantengono una media annua uguale o inferiore a 5 $\mu g/m^3$, ossia Australia, Estonia, Finlandia, Grenada, Islanda, Mauritius e Nuova Zelanda. Al contrario, Paesi come Bangladesh, Pakistan, India, Tajikistan e Burkina Faso mostrano le medie annue più elevate, occupando rispettivamente le prime cinque posizioni con valori di 79.9 $\mu g/m^3$, 73.7 $\mu g/m^3$, 54.4$\mu g/m^3$, 49.0 $\mu g/m^3$ e 46.6 $\mu g/m^3$. L'Italia, pur non figurando nei primi dieci Paesi più inquinati, occupa una preoccupante posizione al 71º posto su 134 Paesi monitorati, con una concentrazione di 15.0 $\mu g/m^3$ [5]. La Figura 2.2 mostra la classifica dei 134 Paesi sotto esame dall'IQAir. La Figura 2.3 mostra la mappa globale colorata in base alla concentrazione media annuale di PM2.5 nel 2023.

In Italia, l'inquinamento atmosferico continua a rappresentare una minaccia significativa, con circa 47.000 decessi prematuri annui attribuibili al PM2.5 e con le regioni settentrionali del Paese a registrare le concentrazioni più elevate. Il

Inquinante	Riferimento temporale	Valori Interim µg/m³				Linee Guida OMS 2021	Linee Guida OMS 2005
		1	2	3	4		
$PM_{2,5}$	Annuale	35	25	15	10	5	10
	24 ore	75	50	37,5	25	15	25
PM_{10}	Annuale	70	50	30	20	15	20
	24 ore	150	100	75	50	45	50
O_3	Valore di picco stagionale	100	70	--	--	60	--
	8 ore	160	120	--	--	100	100
NO_2	Annuale	40	30	20	--	10	40
	24 ore	120	50	--	--	25	--
SO_2	24 ore	125	50	--	--	40	20
CO	24 ore	7 mg/m³	--	--	--	4 mg/m³	--

Figura 2.1: Obiettivi sugli agenti inquinanti redatti dall'OMS

report di Legambiente "Mal'Aria di città 2024" [6] fornisce un approfondimento sulla situazione di alcune città italiane. Tra le 98 città monitorate, ben 18 hanno superato i limiti normativi per gli sforamenti di PM10, rispetto alle 29 del 2022 e 31 del 2021. Frosinone è in testa alla classifica con 70 giorni di sforamento, il doppio rispetto ai valori ammessi, seguita da Torino con 66, Treviso con 63 e Mantova, Padova e Venezia con 62. La maggior parte di queste città fa parte della pianura Padana, che risulta essere l'area più industrializzata e a elevato traffico d'Italia, caratterizzata da significative emissioni di inquinanti atmosferici e dove, a causa della morfologia del territorio, vi è una difficile dispersione delle sostanze inquinanti presenti in atmosfera. Nonostante vi sia un lieve miglioramento rispetto all'anno precedente, attribuibile principalmente a condizioni meteorologiche favorevoli che hanno caratterizzato il 2023, le città italiane, dal Nord al Sud, presentano ancora considerevoli ritardi rispetto ai valori più stringenti proposti dalla revisione della Direttiva europea sulla qualità dell'aria del 2030 (20 μg/m^3 per il PM10, 10 μg/m^3

1	Bangladesh	79.9	46	Mexico	20.1	91	Suriname	10.6
2	Pakistan	73.7	47	South Africa	19.9	92	Lithuania	10.4
3	India	54.4	48	El Salvador	19.5	93	Canada	10.3
4	Tajikistan	49.0	49	Sri Lanka	19.3	94	Russia	10.0
5	Burkina Faso	46.6	50	South Korea	19.2	95	Spain	9.9
6	Iraq	43.8	51	Peru	18.8	96	Japan	9.6
7	United Arab Emirates	43.0	52	Azerbaijan	18.8	97	Panama	9.6
8	Nepal	42.4	53	Chile	18.8	98	Austria	9.6
9	Egypt	42.4	54	Guatemala	18.7	99	France	9.5
10	Dem. Rep. of the Congo	40.8	55	State of Palestine	18.6	100	Belgium	9.4
11	Kuwait	39.9	56	Israel	17.8	101	Argentina	9.2
12	Bahrain	39.2	57	Greece	17.4	102	USA	9.1
13	Qatar	37.6	58	Guyana	17.1	103	Germany	9.0
14	Indonesia	37.1	59	Gabon	16.9	104	Switzerland	8.9
15	Rwanda	36.8	60	Albania	16.7	105	Luxembourg	8.9
16	Zimbabwe	33.3	61	Ivory Coast	16.6	106	Netherlands	8.7
17	Ghana	33.2	62	Georgia	16.4	107	Ukraine	8.6
18	Kyrgyzstan	33.1	63	Togo	16.3	108	Belize	8.3
19	China	32.5	64	Macao SAR	16.2	109	Latvia	8.0
20	Libya	30.4	65	Moldova	15.7	110	Andorra	7.9
21	Laos	29.6	66	Nicaragua	15.7	111	Angola	7.8
22	Vietnam	29.6	67	Romania	15.7	112	United Kingdom	7.7
23	Uzbekistan	28.6	68	Hong Kong SAR	15.6	113	Denmark	7.7
24	Gambia	28.5	69	Maldives	15.3	114	Anguilla	7.4
25	Myanmar	28.2	70	Honduras	15.1	115	Ecuador	7.4
26	Senegal	28.2	71	Italy	15.0	116	Liechtenstein	7.2
27	Bosnia Herzegovina	27.5	72	Slovenia	14.9	117	Jamaica	7.1
28	Uganda	27.3	73	Cyprus	14.3	118	Portugal	6.8
29	Ethiopia	27.0	74	Poland	14.1	119	Ireland	6.3
30	Saudi Arabia	26.5	75	Colombia	14.1	120	Norway	6.3
31	Armenia	26.4	76	Guinea	13.9	121	Costa Rica	6.1
32	North Macedonia	25.2	77	Croatia	13.8	122	Trinidad and Tobago	5.8
33	Zambia	24.1	78	Algeria	13.8	123	Bahamas	5.2
34	Cameroon	24.0	79	Philippines	13.5	124	Sweden	5.1
35	Nigeria	23.9	80	Singapore	13.4	125	Finland	4.9
36	Thailand	23.3	81	Bulgaria	13.2	126	Estonia	4.7
37	Cambodia	22.8	82	Slovakia	13.1	127	Puerto Rico	4.5
38	Malaysia	22.5	83	Brazil	12.6	128	Australia	4.5
39	Mongolia	22.5	84	Bolivia	12.6	129	New Zealand	4.3
40	Kazakhstan	22.2	85	Kosovo	12.1	130	Bermuda	4.1
41	Montenegro	21.3	86	Malta	12.0	131	Grenada	4.1
42	Madagascar	20.6	87	Hungary	12.0	132	Iceland	4.0
43	Serbia	20.5	88	Uruguay	11.7	133	Mauritius	3.5
44	Turkey	20.3	89	Czech Republic	11.5	134	French Polynesia	3.2
45	Taiwan	20.2	90	Kenya	10.6			

Figura 2.2: Classifica dei Paesi più inquinati nel 2023

per il PM2.5). Se il 2030 fosse già qui, il 69% delle città risulterebbe fuorilegge per il PM10, e l'84% per il PM2.5.

Focalizzandoci su Torino, la sua posizione al secondo posto tra le città più inquinate è riconducibile non solo all'intensa attività industriale, ma anche alla sua

Figura 2.3: Mappa globale dei valori medi annui di PM2.5 nel 2023

posizione geografica. Circondata da una grande catena montuosa come le Alpi, la città subisce un accumulo significativo di inquinanti atmosferici, complicato dalla conformazione del territorio che ostacola la dispersione di tali sostanze nell'aria. La topografia agisce come un ostacolo alla purificazione dell'aria da parte dei venti, impedendo loro di allontanare gli agenti inquinanti. A confermare il report fornito da Legambiente, vi è il report dell'Agenzia Regionale per la Protezione Ambientale (ARPA) del 2023, che indica un miglioramento della qualità dell'aria in Piemonte. Tuttavia, vale la pena notare che anche qui viene sottolineato che tale miglioramento è attribuito principalmente a condizioni meteorologiche favorevoli piuttosto che a significativi interventi politici per affrontare l'emergenza smog. Nel dettaglio, il rapporto sottolinea che il 2023 è stato particolarmente positivo per quanto riguarda la diminuzione delle concentrazioni di polveri sottili (PM10 e PM2.5). Tutte le stazioni monitorate hanno rispettato il valore limite medio annuale stabilito dalla normativa, pari a 40 μg/m^3 [7].

In conclusione, l'inquinamento atmosferico rappresenta una sfida critica per la salute umana e l'ecosistema urbano. Affrontare efficacemente questo problema richiede l'adozione di strategie mirate e integrate. Soluzioni come la creazione di zone a traffico limitato, la pedonalizzazione di aree urbane e investimenti in sistemi di trasporto pubblico efficienti possono contribuire a ridurre le emissioni veicolari

e a creare spazi urbani più sicuri e salubri. Tuttavia, la sensibilizzazione della popolazione all'uso di mezzi a emissioni zero, come biciclette e veicoli elettrici, rappresentano la soluzione più efficace. Le biciclette non solo offrono un'alternativa *eco-friendly*, ma promuovono uno stile di vita sano, contribuendo simultaneamente a ridurre le emissioni e migliorare la salute pubblica. Investire nell'infrastruttura ciclabile e promuovere l'uso delle biciclette non solo combatte l'inquinamento atmosferico ma plasmerà un futuro urbano più sostenibile e salutare.

Capitolo 3

Stato dell'arte

Questo capitolo si propone di esplorare il contesto della ricerca pregressa, utilizzando la tesi precedente come punto di partenza fondamentale per comprendere il percorso che ha portato allo sviluppo della presente tesi [8].

3.1 Contesto

La ricerca precedente ha approfondito vari aspetti fondamentali legati all'inquinamento atmosferico. Inizialmente, è stata condotta un'analisi dettagliata comprendente la raccolta e l'elaborazione dei dati provenienti da tre stazioni fisse posizionate nell'area di interesse della precedente tesi: il quartiere di San Salvario e una piccola parte di Nizza Millefiori. Questa analisi ha focalizzato l'attenzione sulla correlazione tra i valori di PM10 e i fattori meteorologici, come pioggia, vento e temperatura nella città di Torino, facendo uso di dati provenienti dal portale Aria della regione Piemonte e da ARPA Piemonte. Lo scopo principale era ottenere un *dataset* finale contenente tutti i dati storici per predire i livelli futuri di PM10 a Torino mediante l'uso di un modello predittivo `ARIMA`[1]. La scelta di focalizzarsi sui valori di PM10 è stata motivata dalla disponibilità di dati più ampi rispetto ai valori di PM2.5. I risultati hanno evidenziato correlazioni significative tra la temperatura e i livelli di PM10, sottolineando l'effetto benefico del vento sui medesimi solo a velocità superiori a 2 m/s, e una moderata influenza della pioggia su di essi. Inoltre, l'analisi ha rivelato un aumento dei livelli di particolato in presenza di basse temperature.

In seguito alle attività di analisi, un aspetto cruciale della ricerca è stata la costruzione di un dispositivo basato su Arduino, che incorpora diversi componenti chiave per il rilevamento della qualità dell'aria e la geolocalizzazione. Il cuore

[1]Il modello ARIMA (acronimo di *AutoRegressive Integrated Moving Average*) è un metodo statistico utilizzato per analizzare e prevedere serie storiche di dati.

del sistema è rappresentato da Arduino UNO WiFi Rev2, un dispositivo versatile ampiamente utilizzato in tutti i progetti IoT (*Internet of Things*) che mirano a costruire una rete di sensori connessa al Wi-Fi o a un dispositivo Bluetooth. Tale modello è stato scelto proprio per la possibilità di connettersi a una rete Wi-Fi. Un secondo dispositivo utilizzato è il sensore digitale PMS7003/G7, che consente di raccogliere dati di PM1, PM2.5 e PM10. Per fornire dati sulla posizione attuale, si è fatto uso del modulo GPS Adafruit PA1616S, potenziato da un'antenna esterna che consente di ottenere un'elevata precisione nella rilevazione della posizione geografica. Le luci LED (*Light Emitting Diod*) svolgono una funzione di segnalazione all'interno del dispositivo, fornendo indicazioni sullo stato della connessione GPS e la trasmissione dei dati. Grazie all'uso di una *breadboard* è stato possibile connettere Arduino agli altri componenti. Per proteggere l'intero sistema dagli agenti atmosferici e fornire un involucro robusto, è stata impiegata una scatola di derivazione esterna IP65[2]. Su di essa sono stati eseguiti diversi fori che garantiscono un flusso d'aria ottimale all'interno della stessa e, di conseguenza, un corretto rilevamento dei valori di PM10. Infine, sarà sufficiente utilizzare un *powerbank* al fine di fornire l'energia necessaria[3]. Grazie alle sue dimensioni ridotte, questo strumento può essere installato su mezzi di trasporto *eco-friendly*, come bici o monopattini elettrici. La Figura 3.1 mostra il dispositivo assemblato.

Figura 3.1: Dispositivo Arduino assemblato

[2]La classificazione IP65 garantisce la protezione totale alla penetrazione di corpi solidi e di liquidi da gocce, vapore, spruzzi e getti d'acqua, che provengono da qualsiasi direzione.

[3]L'uso di un powerbank consente un'opzione flessibile e portatile per alimentare il dispositivo in vari contesti, senza la necessità di una fonte di alimentazione fisso.

Il lavoro è stato svolto in due fasi distinte, la prima riguarda la scrittura del codice su Arduino per ottenere dati sulla posizione e i valori di PM10, mentre la seconda consiste nell'invio in tempo reale dei dati al database . Il codice Arduino è stato sviluppato tramite Arduino IDE (*Integrated Development Environment*), che consente di scrivere codice su un file `.ino` che verrà caricato sul microcontrollore. Il codice è strutturato in due fasi principali: la fase di *setup* e la fase di *loop*.

Nella fase di *setup*, avviata all'accensione del dispositivo, vengono eseguite diverse operazioni chiave. Viene inizializzata e verificata la connettività al Wi-Fi, viene stabilita una connessione con il Broker MQTT (Message Queuing Telemetry Transport). Inoltre, viene inviato un messaggio di *reset* al Broker per identificare univocamente ogni sessione di rilevamento, creando così un nuovo codice univoco per il percorso attuale. Allo stesso tempo, vengono spenti tutti i LED per riportare il sistema a una condizione iniziale standard. La Figura 3.2 mostra la fase di *setup*.

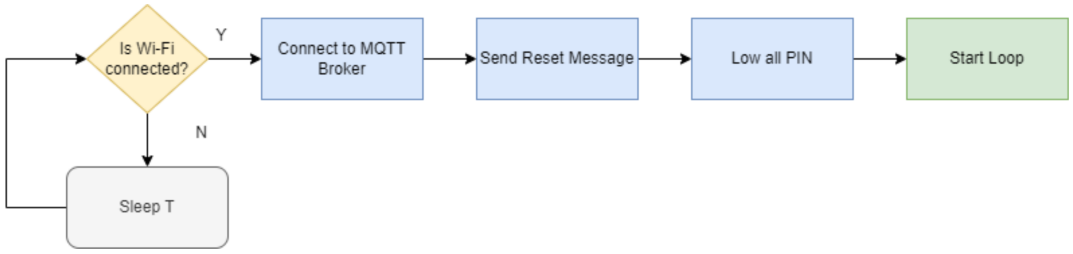

Figura 3.2: Schema della fase di *setup* del codice Arduino

La fase di *loop*, caratterizzata da un ciclo di esecuzione continuo, rappresenta il cuore del funzionamento del dispositivo. A ogni iterazione, viene verificato se è giunto il momento di campionare nuovi dati, controllando se il tempo trascorso dall'ultimo campionamento è maggiore dell'intervallo scelto, ossia 5 secondi. In caso affermativo, si procede con un nuovo campionamento, effettuando la lettura del valore dal sensore del PM10 e dal GPS. La Figura 3.4 mostra la fase di *loop*.

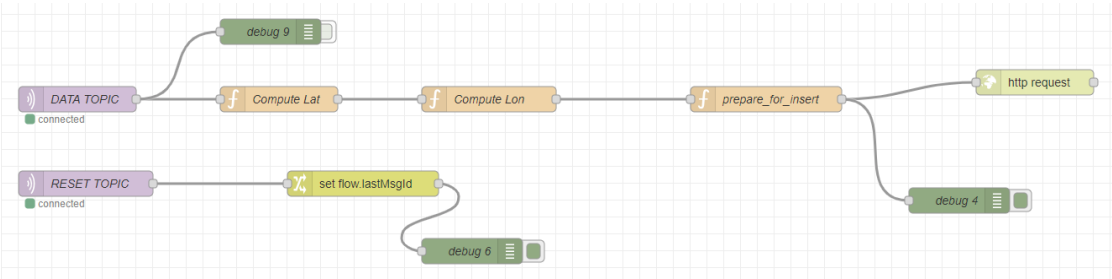

Figura 3.3: Precedente flusso di elaborazione real time in Node-RED

Una volta acquisiti, i dati vengono inoltrati al Broker MQTT, attivando un flusso Node-RED[4] per la preparazione all'inserimento nel database. Le operazioni includono la conversione delle coordinate di latitudine e longitudine in radianti, l'aggiunta di un *timestamp* di ricezione e la strutturazione dei dati nel formato richiesto da CouchDB. La Figura 3.3 mostra il flusso di elaborazione Node-RED sopra citato.

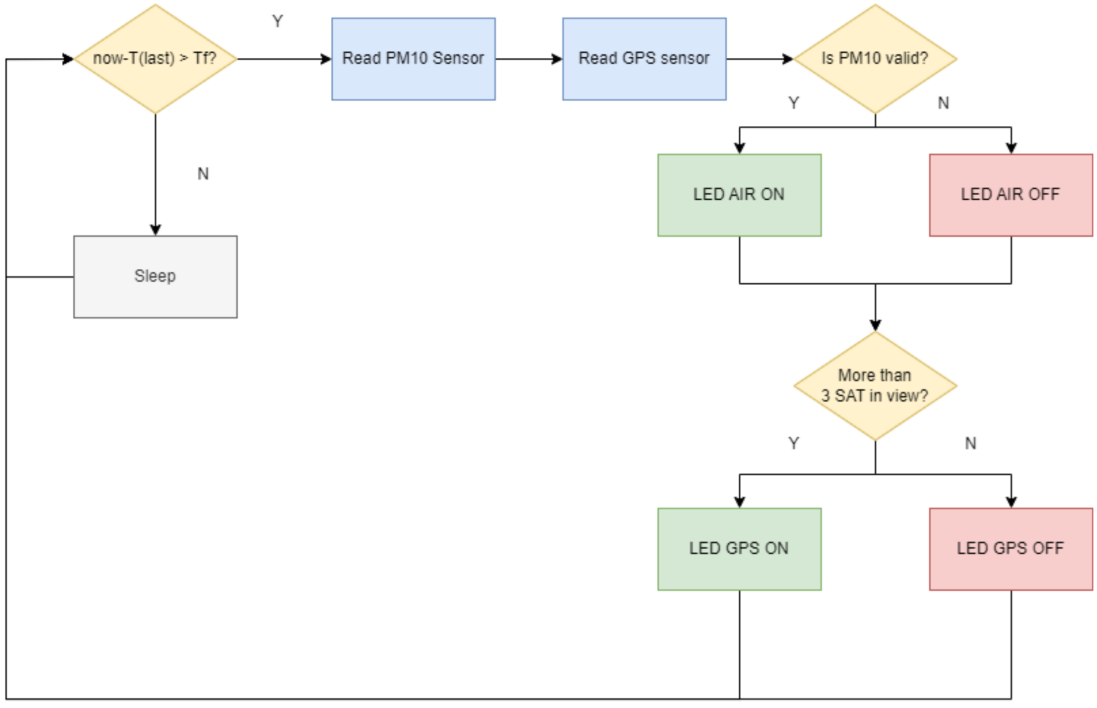

Figura 3.4: Schema della fase di *loop* del codice Arduino

Successivamente, CouchDB[5] viene chiamato tramite API REST per l'inserimento dei dati. La Tabella 3.1 mostra i campi del database CouchDB, comprensivi di una breve descrizione ed un esempio.

[4]Node-RED è un ambiente di sviluppo *open source* basato su *browser* per l'interconnessione di dispositivi *hardware*, API e servizi online. Utilizzando un approccio basato su nodi, Node-RED consente agli sviluppatori di creare facilmente flussi di lavoro automatizzati per l'IoT, l'integrazione di sistemi e altre applicazioni di automazione.

[5]CouchDB è un database orientato a documenti, sviluppato in Apache Software Foundation. È progettato per archiviare, gestire e interrogare dati semi-strutturati in formato JSON tramite un'interfaccia RESTful. La sua architettura distribuita e scalabile lo rende adatto per applicazioni che richiedono una gestione efficiente dei dati su più nodi e la sincronizzazione tra dispositivi.

Campo	Descrizione	Esempio
_id	Identificativo del documento	"5cc441bf521fbe69411c25b3f4c591fb"
_rev	-	"1-a1f52caa73ef2e03285ce5835bcec566"
pm100	Valore di PM10	4
pm25	Valore di PM2.5	4
s	Numero di satelliti (stringa)	"06"
lt	Latitudine in decimali	"4503.5415"
ln	Longitudine in decimali	"00740.8434"
d_lt	Direzione latitudine	"N"
d_ln	Direzione longitudine	"E"
q	Qualità dell'aria	"1"
hd	HDOP	"1.29"
lat	Latitudine in gradi	45.059025
lon	Longitudine in gradi	7.680723333333333
satellites	Numero di satelliti	6
ts	Timestamp rilevazione	1682698933469
tripId	Identificativo del percorso	"TRIP_1682698931799"

Tabella 3.1: Struttura del documento presente nel database CouchDB

Successivamente, l'attenzione si è concentrata sulla creazione di un grafo NetworkX[6] per mappare la zona di San Salvario a Torino. Questo grafo ha agevolato l'associazione dei valori di inquinamento provenienti da stazioni fisse e dal dispositivo Arduino ai nodi della mappa, rappresentanti gli incroci delle strade in San Salvario. Questa fase è stata cruciale per lo sviluppo di un algoritmo che calcola il percorso ottimale basandosi sui livelli di inquinamento, oltre allo sviluppo di servizi per gli utenti. I dati provenienti dal dispositivo Arduino, grazie alla posizione catturata dal GPS, vengono associati a un nodo della mappa della città di interesse.

Infine, sono stati elaborati quattro servizi pratici per gli utenti, utilizzando i dati da stazioni fisse, i dati raccolti dal dispositivo Arduino e la mappa creata su NetworkX. Tali servizi, accessibili tramite un *menu* mostrato nel terminale, consentono agli utenti di ottenere informazioni sulla qualità dell'aria in modo flessibile e predire dati futuri.

- **Servizio 1**: Permette all'utente di inserire una data e un nodo di interesse per ottenere il valore di PM10 corrispondente.
- **Servizio 2**: Simile al precedente, ma l'utente può specificare anche l'ora di interesse.

[6]NetworkX è una libreria Python per la creazione, manipolazione e studio di reti complesse, come grafi e reti di nodi.

- **Servizio 3**: Consente all'utente di inserire data, ora, nodo di partenza e nodo di arrivo per ottenere il percorso con i livelli più bassi di PM10.
- **Servizio 4**: Permette all'utente di inserire data e ora, visualizzando una *heatmap* sulla distribuzione di PM10.

Questi servizi offrono un'esperienza utente completa, integrando dati storici e in tempo reale con previsioni basate su algoritmi avanzati.

3.2 Sviluppi futuri

Questa sezione esplora come alcuni degli sviluppi futuri inizialmente proposti siano stati implementati durante la presente ricerca. In questo elaborato si è deciso di sviluppare le seguenti:

- **Applicazione *mobile***: La tesi precedente proponeva lo sviluppo di un'applicazione *mobile* al fine di integrare i servizi sviluppati. La presente ricerca mira a completare questo obiettivo, progettando e realizzando un'applicazione *mobile* con un'interfaccia utente intuitiva.
- **Estensione del grafo NetworkX**: L'obiettivo di estendere il grafo NetworkX oltre i confini della zona di San Salvario sarà realizzato mediante l'implementazione di un algoritmo che consentirà di costruire una mappa completa dell'intera città. Questo ampliamento fornirà una visione più completa e globale della qualità dell'aria a Torino.

Per una comprensione approfondita di tutti gli aspetti trattati in questo capitolo e per esplorare ulteriormente la ricerca precedente, si consiglia di consultare la tesi completa disponibile all'indirizzo https://hdl.handle.net/20.500.12075/13678.

Capitolo 4

Progettazione del sistema

Il presente capitolo si focalizza sulla progettazione del sistema, offrendo un'analisi delle tecnologie adottate e delle decisioni prese durante la fase di sviluppo. È importante sottolineare che, in azienda, la progettazione del sistema è stata la fase iniziale del processo, durante la quale sono state esaminate diverse alternative per ciascuna soluzione, ottenendo così una visione completa del progetto e delle tecnologie correlate.

Prima di esplorare le tecnologie adottate per lo sviluppo dell'algoritmo di creazione della mappa e del server, è necessario comprendere la motivazione che ha spinto alla transizione da CouchDB a MongoDB. Inizialmente, nella tesi precedente, la scelta di utilizzare CouchDB era motivata dalla sua semplicità e dall'efficace gestione di dati orientati a documenti. Tuttavia, la mancanza di supporto nativo per le *query* geospaziali[1] ha reso necessaria la migrazione a MongoDB. Quest'ultimo è stato scelto non solo per la sua capacità di gestire tali *query*, ma anche per essere un database NoSQL orientato a documenti, analogamente a CouchDB, e soprattutto per i notevoli miglioramenti in termini di flessibilità e scalabilità che offre [9]. Le avanzate funzionalità di indicizzazione e *query* di MongoDB sono risultate essenziali per la realizzazione di servizi basati su *query* geospaziali, consentendo ricerche precise sulla posizione geografica dei dati, sfruttando un indice geospaziale ottimizzato.

[1]Le *query* geospaziali sono un tipo di interrogazione utilizzata nei database per trovare dati basati sulla loro posizione geografica. Questo tipo di *query* consente di cercare dati all'interno di una determinata area geografica, definendo un raggio o un poligono. Esse richiedono un'indicizzazione specifica dei dati basata sulle coordinate geografiche per garantire prestazioni efficienti nelle ricerche di questo tipo. Per ulteriori approfondimenti si veda https://www.mongodb.com/docs/manual/geospatial-queries/.

4.1 Tecnologie per l'algoritmo di creazione della mappa

L'algoritmo per la creazione della mappa di una città rappresenta un componente fondamentale per il successo dell'applicazione. È stato implementato utilizzando il linguaggio di programmazione Python, scelto per le sue caratteristiche versatili e la disponibilità di librerie specializzate nel campo della manipolazione e dell'analisi dei dati geospaziali. In particolare, tre librerie hanno giocato un ruolo cruciale per la realizzazione dell'algoritmo:

- **Shapely**: Specializzata nella manipolazione e nell'analisi di oggetti geometrici planari. Risulta particolarmente utile quando si lavora con dati spaziali, in quanto fornisce un insieme di strumenti per eseguire operazioni geometriche su oggetti come punti, linee o poligoni[2]. Nel contesto dell'algoritmo, è stato fatto uso del modulo `shapely.geometry`, che offre una vasta gamma di classi tra cui `Geometry`, che consentono di rappresentare concetti spaziali in modo intuitivo e di eseguire operazioni su di essi. In particolare, sono state utilizzate le classi `Point` e `Polygon`, che rappresentano rispettivamente un singola coordinata nel formato x,y e un poligono.

- **NetworkX**: Impiegata per la creazione, la manipolazione e lo studio di grafi e reti[3]. Tale libreria è stata impiegata per modellare la rete stradale della città sotto forma di grafo, dove ogni nodo rappresenta un punto geografico, in particolare un incrocio di una strada, mentre gli archi riflettono le connessioni tra questi punti, rappresentando le strade che collegano gli incroci.

- **Geopy**: Specializzata in operazioni di *geocoding*, *reverse geocoding* e calcolo delle distanze tra posizioni geografiche[4]. Nel contesto dell'algoritmo, è stato fatto uso del metodo `great_circle` fornito dal modulo `distance`, il quale gioca un ruolo fondamentale nel calcolo della distanza tra due punti date due coordinate sotto forma di latitudine e longitudine.

[2]Per ulteriori approfondimenti si veda https://shapely.readthedocs.io/en/stable/.

[3]Per ulteriori approfondimenti si veda https://networkx.org/documentation/stable/reference/introduction.html.

[4]Per ulteriori approfondimenti si veda https://geopy.readthedocs.io/en/stable/.

4.2 Tecnologie per il *backend*

Il server svolge un ruolo centrale nella gestione dei dati e delle comunicazioni all'interno del sistema. La scelta del *framework* rappresenta un passo cruciale nella progettazione dell'architettura. In questo contesto, la scelta di adottare NestJS rispetto ad altri *framework*, come Express, è stata motivata dalla sua modularità e dalla struttura organizzata basata sui moduli, che consente una chiara separazione delle responsabilità all'interno dell'applicazione. L'integrazione nativa di TypeScript ha rappresentato un vantaggio significativo in termini di *type safety*, riducendo potenziali errori durante lo sviluppo. Inoltre, si è fatto uso della libreria Geolib[5] per la gestione delle coordinate geografiche e per i calcoli delle distanze.

Per gestire il processo di autenticazione degli utenti, è stato adottato Firebase, una piattaforma di sviluppo di app *web* e *mobile* sviluppata da Google. Firebase offre una vasta gamma di servizi *backend* pronti all'uso, consentendo di concentrarsi sullo sviluppo delle funzionalità dell'applicazione senza dover gestire l'infrastruttura sottostante. Tra i principali servizi offerti da Firebase vi sono l'*hosting*, l'archiviazione dei dati e l'autenticazione, quest'ultima di particolare rilevanza nel contesto della nostra ricerca [10]. Il servizio di autenticazione Firebase fornisce un modo semplice e sicuro per gestire l'accesso degli utenti alle applicazioni. Questo è cruciale per garantire che solo gli utenti autorizzati possano accedere ai dati e alle funzionalità dell'applicazione. Tale servizio è stato scelto per diversi motivi:

- **Sicurezza:** Firebase gestisce in modo sicuro le credenziali degli utenti offrendo funzionalità avanzate di sicurezza, come l'autenticazione a due fattori, garantendo così che l'accesso all'applicazione sia protetto da eventuali minacce esterne.
- **Diverse modalità di accesso:** Firebase supporta diverse modalità di accesso per gli utenti, tra cui l'autenticazione tramite email e password, account Google, account Facebook e molto altro. Questa flessibilità consente di offrire agli utenti diverse opzioni per accedere all'applicazione, migliorando così la comodità e l'esperienza utente.
- **Gestione delle identità:** Firebase fornisce strumenti per la gestione delle identità degli utenti, consentendo di memorizzare e recuperare informazioni aggiuntive sugli utenti, come nome, foto del profilo e altro ancora, personalizzando così l'esperienza dell'utente e offrendo funzionalità aggiuntive basate sul profilo utente.

[5]Per ulteriori approfondimenti si veda https://github.com/manuelbieh/geolib.

4.3 *Frontend*

La realizzazione dell'applicazione è stata suddivisa in tre fasi: la scelta delle tecnologie, la realizzazione dei *mockup* e l'implementazione del codice, la quale verrà trattata nel capitolo 8.

4.3.1 Scelta delle tecnologie

Nel processo di selezione delle tecnologie per l'applicazione, l'analisi iniziale ha considerato lo sviluppo di un'applicazione *web* o *mobile*. Dopo un'attenta valutazione, è emerso che per raggiungere gli obiettivi del progetto era necessario concentrarsi sull'implementazione di un'applicazione *mobile*. Questa decisione è stata guidata dalla natura specifica del progetto, concepito principalmente per l'utilizzo durante gli spostamenti con mezzi di trasporto *eco-friendly*, come biciclette o monopattini elettrici. Di conseguenza, l'adozione di un'applicazione *mobile* si è rivelata la scelta più sensata e adeguata alle esigenze del contesto. A partire da questa scelta, sono state prese in considerazione tre soluzioni per lo sviluppo di un'applicazione *mobile*: applicazioni native, *Progressive Web App* (PWA) e applicazioni ibride [11].

- **Applicazioni native**: Sono sviluppate per funzionare su uno specifico sistema operativo. Ad esempio, un'app nativa per iOS sviluppata con Objective-C o Swift, non potrà essere eseguita su Android. Allo stesso modo, un'app nativa per Android sviluppata con Java o Kotlin, non potrà essere eseguita su iOS. Sebbene offrano elevate prestazioni e un'interazione ottimizzata con le funzionalità offerte dal dispositivo, richiedono lo sviluppo separato per iOS e Android, rendendo il processo più lento e costoso.
- ***Progressive Web App* (PWA)**: Sono studiate per essere simili alle applicazioni native, ma vengono eseguite sul browser del dispositivo senza richiedere un'installazione e possono essere raggiunte dall'utente tramite URL. Consentono di abbattere i costi di sviluppo grazie a un'unica *codebase* e garantiscono un'elevata compatibilità fra tutti i dispositivi. Tuttavia, presentano prestazioni inferiori rispetto alle app native e alcune limitazioni nell'accesso alle funzionalità del dispositivo.
- **Applicazioni ibride (multipiattaforma)**: Combina elementi delle app native e delle PWA, ereditando vantaggi e svantaggi da entrambe. L'utilizzo di una *codebase* unica consente lo sviluppo per entrambe le piattaforme, iOS e Android, con una scrittura del codice singola. Sebbene permettano uno sviluppo più rapido ed economico, le prestazioni potrebbero essere leggermente inferiori rispetto alle app native.

Dopo un'attenta analisi delle diverse opzioni disponibili per lo sviluppo dell'applicazione *mobile*, la scelta è ricaduta sull'adozione di un approccio multipiattaforma. La Figura 4.1 riassume le caratteristiche distintive dei vari tipi di applicazioni, evidenziando le differenze tra di esse.

	NATIVE APPS	PWAs	HYBRID APPS
Language	Java, Kotlin, Swift, Objective-C, C++	HTML5, CSS, JavaScript, Ruby	Ionic, HTML5, Javascript e CSS
Technology	Platform-specific	General	General
Codebase	Distinct repositories for each platform	Single cross-platform codebase	Single cross-platform codebase
Financial investment	High	Low / Medium	Low / Medium
Development	Slow	Fast / Medium	Fast / Medium
Performance	High	Low / Medium	Low / Medium
UI	Native UI	Non-native UI	Semi-native UI
Device features access	Full	Limited	Full (with plugins)

Figura 4.1: Confronto tra *app* native, *Progressive Web App* e *app* ibride

Tra i principali *framework* multipiattaforma *mobile*, sono state prese in considerazione le due soluzioni più diffuse: React Native e Flutter. React Native è un *framework open source* sviluppato da Facebook rilasciato nel 2015, basato sulla libreria React. Una delle sue caratteristiche distintive è la possibilità di scrivere codice JavaScript o TypeScript sfruttando le potenzialità del *framework* React e consentendo la costruzione di applicazioni sia per iOS che per Android. Analogamente a React, l'idea alla base di React Native è il concetto di componente, dove ogni elemento come un bottone, un input o una schermata, viene rappresentato come un componente, contribuendo alla creazione della logica e dell'interfaccia grafica dell'applicazione. Flutter è una *framework open source* sviluppato da Google e rilasciato nel 2018. Utilizza il linguaggio di programmazione Dart, introdotto da Google nel 2011, e segue un approccio differente rispetto a React Native, offrendo un insieme di *widget* che facilitano la creazione di interfacce utente. La Figura 4.2

mostra un confronto fra React Native e Flutter sull'interesse nel tempo dal 2018 al 2024 [12].

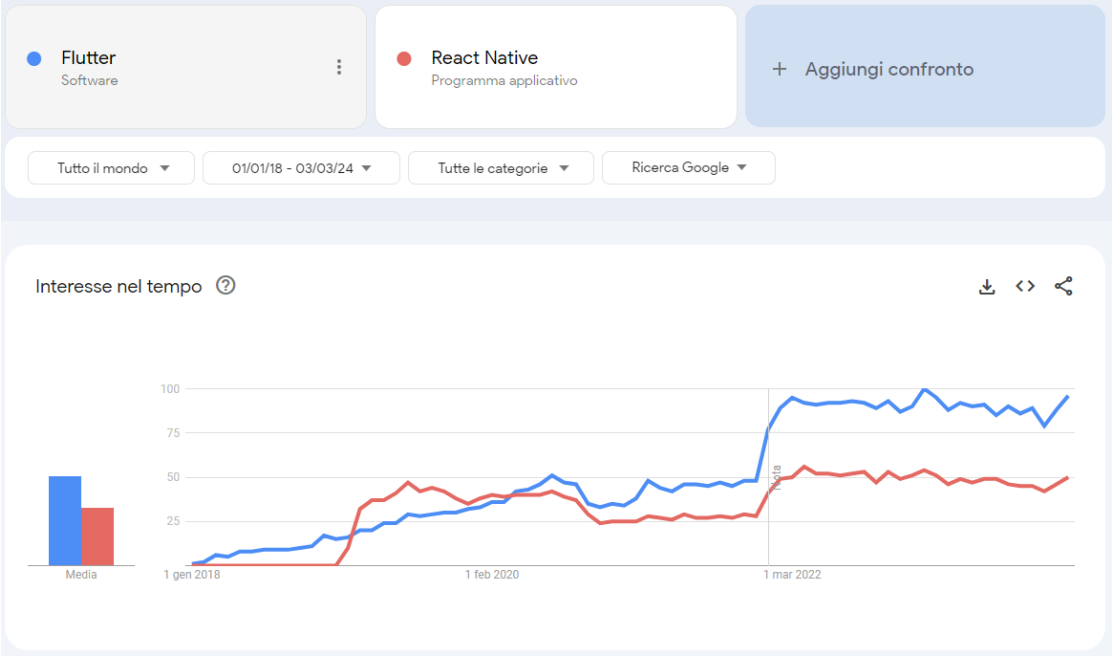

Figura 4.2: Confronto fra React Native e Flutter dal 2018 al 2024

Dopo un'analisi approfondita, la scelta è ricaduta su React Native per lo sviluppo dell'applicazione. Questa decisione è stata guidata dal fatto che React Native è stato rilasciato prima rispetto a Flutter e ha una base di utenti più ampia e stabile. Tuttavia, anziché adottare React Native in modo nativo, si è preferito utilizzare Expo, un *framework* che semplifica notevolmente il processo di sviluppo di app React Native. Expo offre un set completo di strumenti e servizi basati su React Native che consentono di sviluppare, compilare e distribuire app React Native in modo più efficiente e veloce rispetto all'approccio tradizionale [13]. Le funzionalità fornite da Expo includono:

- **Avvio rapido**: Expo offre un'esperienza di sviluppo semplificata che elimina le complessità legate alla configurazione di un progetto React Native. Con un semplice comando, `npx create-expo-app my_app`, è possibile creare rapidamente un progetto funzionante. Al contrario, l'approccio nativo di React Native richiede procedure diverse a seconda del sistema operativo del computer (macOS, Windows o Linux) e del sistema operativo di destinazione (Android, iOS), rendendo il processo di avvio più laborioso e meno immediato.

- **Aggiornamenti Over-the-Air (OTA)**: Con Expo, è possibile distribuire gli aggiornamenti dell'app direttamente ai dispositivi degli utenti senza richiedere loro di scaricare una nuova versione dall'app store. Questo approccio semplifica il processo di distribuzione e garantisce che gli utenti ricevano rapidamente le nuove funzionalità e correzioni di bug.
- **Accesso a librerie precostruite**: Expo fornisce una vasta gamma di librerie precostruite e API (*Application Programming Interface*), semplificando l'aggiunta di funzionalità come la localizzazione, la condivisione di immagini e video, o l'accesso ai sensori del dispositivo come l'accelerometro o il pedometro.

Le librerie principali impiegate durante la fase di sviluppo dell'applicazione sono state due:

- **React Native Firebase**: Fornisce una raccolta di moduli ufficiali per React Native, semplificando il collegamento ai servizi Firebase. Nel contesto di questo progetto, è stato utilizzato il modulo `@react-native-firebase/auth`[6] il quale fornisce il supporto per i servizi di autenticazione, inclusi quelli basati su password, numeri di telefono e *provider social* come Google e Facebook.
- **React Native Maps**: Offre componenti per l'integrazione di mappe interattive nelle applicazioni React Native. Utilizza le API di Google Maps su Android e MapKit su iOS, garantendo una soluzione multipiattaforma per la visualizzazione e l'interazione con mappe geografiche. Questa libreria mette a disposizione diversi componenti, tra cui `MapView`, `Polyline`, `Marker` e `Heatmap`. `MapView` consente di mostrare la mappa e di interagire con essa. `Polyline` consente di disegnare un percorso sulla mappa, utilizzando le coordinate fornite. `Marker` consente di posizionare marcatori sulla mappa per indicare punti di interesse o dati specifici. `Heatmap` permette di generare una *heatmap* basata su dati numerici, offrendo una rappresentazione visiva della distribuzione dei valori in una determinata area[7].

4.3.2 Realizzazione dei *mockup*

Dopo aver selezionato le tecnologie da utilizzare, è stata condotta la fase di realizzazione dei *mockup* al fine di ottenere una chiara rappresentazione grafica di tutte le sezioni dell'applicazione. Le immagini seguenti mostrano i *mockup* delle diverse schermate dell'applicazione.

[6]Per ulteriori approfondimenti si veda https://rnfirebase.io/auth/usage.

[7]Per ulteriori approfondimenti si veda https://github.com/react-native-maps/react-native-maps.

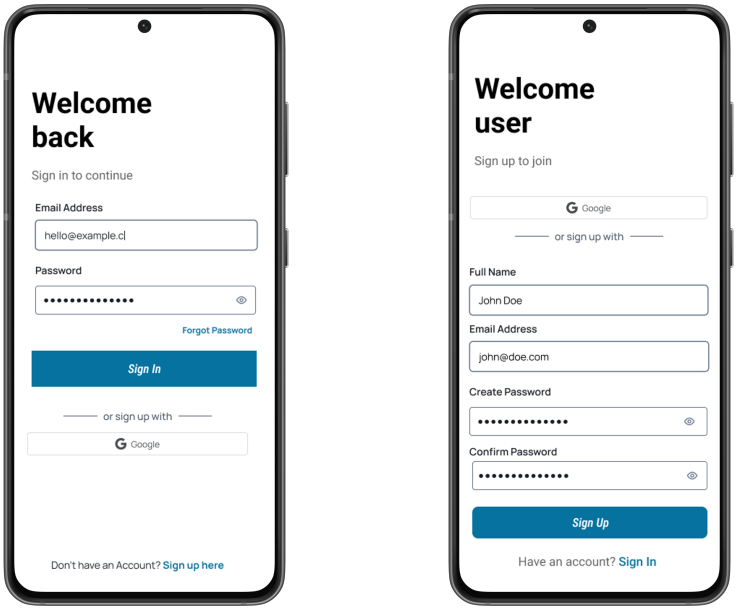

Figura 4.3: *Mockup* delle schermate di accesso e registrazione

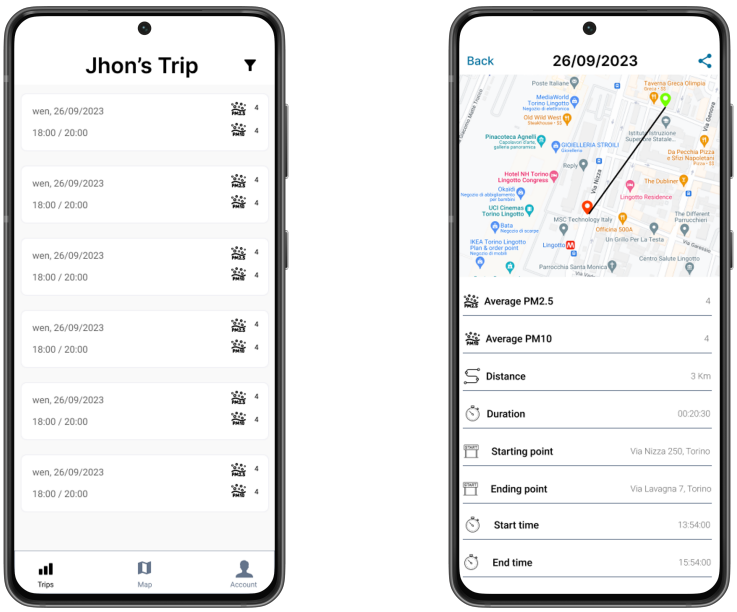

Figura 4.4: *Mockup* delle schermate dei percorsi passati e di un percorso specifico

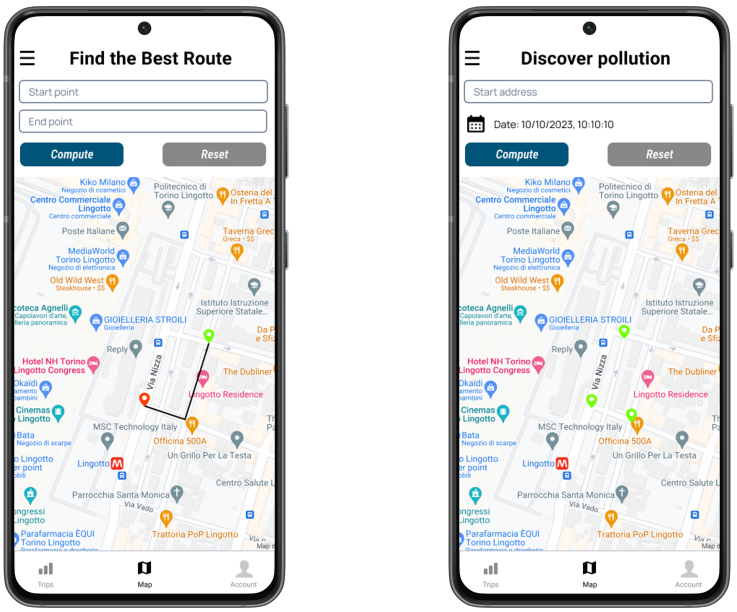

Figura 4.5: *Mockup* delle schermate del percorso meno inquinato e della visualizzazione puntuale dell'inquinamento

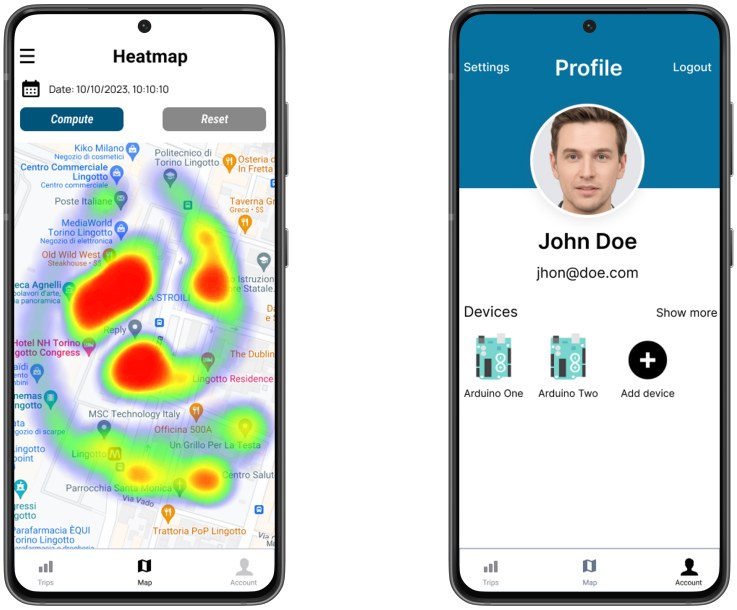

Figura 4.6: *Mockup* delle schermate per la *heatmap* e il profilo utente

Capitolo 5

Dispositivo di raccolta dati

Il presente capitolo analizza l'evoluzione del dispositivo di raccolta dati, concentrandosi sulle migliore *software* apportate. Verranno esaminate le ottimizzazioni introdotte per migliorare l'accuratezza delle coordinate GPS ricevute durante la fase di raccolta dati, oltre all'introduzione di una nuova funzionalità chiave: l'associazione dispositivo-utente. Ogni aspetto verrà analizzato dal punto di vista del dispositivo di raccolta dati, evidenziando le implementazioni specifiche volte a potenziare le capacità di acquisizione e arricchire le funzionalità offerte.

5.1 Evoluzione del flusso di inserimento dei dati

Inizialmente, il processo di inserimento dei dati di un percorso nel database seguiva un approccio differente, come descritto in 3.1. In quel contesto, una volta acquisiti, i dati venivano trasmessi al Broker MQTT, che attivava un flusso Node-RED che si occupava delle operazioni di conversione e preparazione dei dati prima dell'inserimento nel database. Tuttavia, in seguito a un'analisi dei dati ricevuti dal dispositivo, è emerso che la precisione delle coordinate GPS non raggiungeva livelli soddisfacenti. Questa constatazione ha portato a una ristrutturazione del processo di inserimento dati, introducendo un nuovo microservizio `bike-monitoring-location-normalizer`. Questo microservizio è stato progettato per consentire la normalizzazione delle coordinate ricevute dal dispositivo Arduino. Tuttavia, per effettuare tale processo è stato necessario acquisire coordinate che rappresentassero punti di riferimento reali. A tale scopo, è stato sviluppato un algoritmo per la creazione della mappa di una città che consente di ottenere e salvare nel database gli incroci della città in esame (nel nostro caso, Torino). Quest'ultimo verrà discusso nel capitolo 6.

Per garantire una centralizzazione delle operazioni legate all'inserimento dei dati in una singola componente coesa, si è deciso di integrare tutte le operazioni presenti

nel flusso Node-RED, realizzato nella tesi precedente, all'interno del microservizio, fatta eccezione per il recupero del `lastMsgId` il quale è rimasto nel flusso Node-RED per scelta implementativa. Il nuovo processo di inserimento dati inizia con il dispositivo Arduino che invia i dati al Broker MQTT ogni 5 secondi. Il Broker, dopo l'attivazione del flusso Node-RED, recupera il `lastMsgId` e contatta il microservizio all'*endpoint* `POST /v1/location-normalizer/add-trip`, completando il ciclo di acquisizione, normalizzazione e inserimento dei dati relativi a un percorso nel database. I dettagli implementativi del servizio verranno esaminati in 7.1. La Figura 5.1 mostra il nuovo flusso di lavoro di Node-RED.

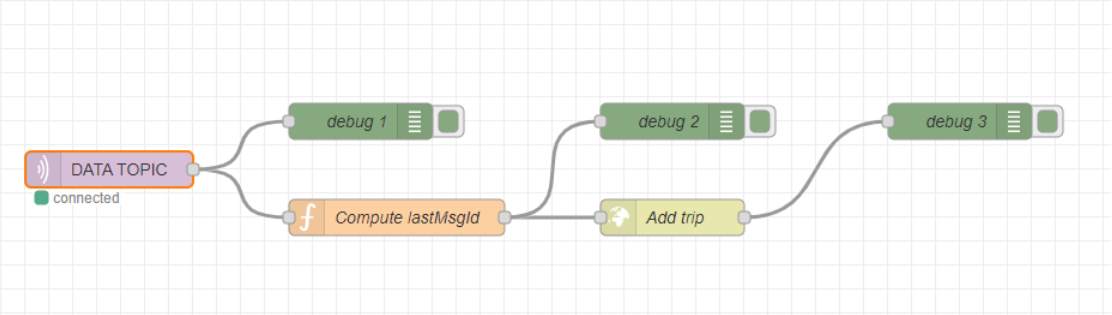

Figura 5.1: Nuovo flusso di elaborazione real time in Node-RED

5.2 Associazione dispositivo-utente

L'implementazione dell'associazione dispositivo-utente ha rappresentato un passo fondamentale per migliorare la flessibilità e la scalabilità del dispositivo di raccolta dati. Questa nuova funzionalità è stata sviluppata con l'obiettivo di consentire a più utenti di acquisire dati simultaneamente tramite dispositivi diversi, superando così una limitazione chiave della precedente implementazione che non permetteva di associare un'utente a un dispositivo.

Il processo di associazione dispositivo-utente è suddiviso in tre fasi: l'invio della richiesta da parte dell'applicazione, la gestione della richiesta sul dispositivo Arduino e il salvataggio dei dati nel database. In questa sezione, verrà esaminata la seconda fase, mentre la prima e la terza fase verranno esaminate rispettivamente nei capitoli 8 e 7.

Il codice è stato sviluppato utilizzando Arduino IDE 2.3.2. Per implementare questa funzionalità, è stata condotta un'analisi del codice esistente per comprendere il suo funzionamento, e successivamente sono state apportate le modifiche necessarie. Il dispositivo, basato sulla scheda Arduino UNO WiFi Rev2, fa uso della libreria

`WiFiNINA.h`[1], che consente di effettuare l'accesso a una rete Wi-Fi o di agire da punto di accesso. Quest'ultima caratteristica è stata sfruttata per creare un punto di accesso alla scheda tramite un'istanza `WiFiServer` sulla porta 80. L'inizializzazione del *server web* viene eseguito nella funzione `setup` attraverso il comando `server.begin()`, come mostrato nel Codice 5.1.

```
#include <WiFiNINA.h>
#include <EEPROM.h>

// Crea un'istanza WiFiServer sulla porta 80
WiFiServer server(80);

// Indirizzo della memoria in cui salvare lo UUID dell'utente
int eepromAddress = 0;
long savedUUID;

// N.B. Genera un nuovo UUID V4 quando il codice viene caricato su una nuova scheda Arduino
String arduinoUUID = "b2049a504a734a02988d66f775eef3f5";

void setup() {
    ...

  // Recupera lo UUID salvato dalla EEPROM
  EEPROM.get(eepromAddress, savedUUID);

  String savedUUIDString = "";
  for (int i = 0; i < 32; ++i) {
      savedUUIDString += char(EEPROM.read(eepromAddress + i));
  }

  // Avvia il server web sulla porta 80
  server.begin();

  ...
}
```

Codice 5.1: Associazione dispositivo-utente: Fase di *setup*

Per gestire le richieste HTTP (*Hypertext Transfer Protocol*) in arrivo dall'applicazione, è stata implementata la funzione `handleHttpRequest`, mostrata nel Codice 5.2. Questa funzione verifica la presenza di nuove richieste HTTP, legge l'intera richiesta e, nel caso in cui sia di tipo `POST`, analizza il corpo della richiesta

[1]Per ulteriori approfondimenti si veda https://docs.arduino.cc/tutorials/communication/wifi-nina-examples/.

per estrarre e salvare il valore del campo UUID[2] (*Universally Unique Identifier*) dell'utente. Successivamente, prepara una risposta da inoltrare all'applicazione seguendo lo standard HTTP, contenente l'informazione relativa allo UUID del dispositivo Arduino. È importante notare che questo valore è statico e che per ogni nuovo dispositivo sarà necessario generare un nuovo UUID v4 per attribuire un nuovo identificativo univoco al dispositivo. Per memorizzare in modo persistente lo UUID dell'utente presente nel corpo della richiesta, si è fatto uso della memoria EEPROM[3] integrata in ogni scheda Arduino, insieme alla libreria `EEPROM.h`. Lo UUID dell'utente viene memorizzato all'indirizzo zero della memoria utilizzando la funzione `EEPROM.write(address, value)`. Durante l'avvio del dispositivo, la funzione `EEPROM.get(address, data)` viene eseguita per leggere il valore dello UUID dalla EEPROM e salvarlo in una variabile. Questo valore viene quindi inoltrato al database insieme alle altre informazioni relative al percorso raccolte dai sensori del dispositivo.

Tale procedura consente di associare in modo univoco i dati raccolti durante un percorso a un utente, permettendogli di visualizzare successivamente le informazioni nella sezione dedicata dell'applicazione.

```
1  void handleHttpRequest() {
2    WiFiClient client = server.available();
3
4    if (client) {
5      Serial.println("Nuova richiesta HTTP in arrivo");
6
7      bool requestProcessed = false;
8      String request = client.readStringUntil('\r');
9      String uuidValue;
10
11     // Analizza il corpo della richiesta
12     if (request.indexOf("POST") != -1 && !requestProcessed) {
13       while (client.available()) {
14         String line = client.readStringUntil('\r');
15         line.trim();
16
17         int newlinePos = line.indexOf('\n');
18         String lineFields = line.substring(0, newlinePos);
```

[2]Uno UUID (Universally Unique Identifier) è un identificatore univoco standardizzato utilizzato nel campo dell'informatica. Si tratta di una sequenza di 128 bit, tipicamente rappresentata come una serie di 32 cifre esadecimali separate da trattini, ad esempio: 550e8400-e29b-41d4-a716-446655440000.

[3]La memoria EEPROM (Electrically Erasable Programmable Read-Only Memory) è un tipo di memoria non volatile utilizzata nei dispositivi Arduino per memorizzare dati in modo permanente, anche quando il dispositivo viene spento o riavviato. Per maggiori approfondimenti si veda https://docs.arduino.cc/learn/programming/eeprom-guide/.

```
19
20          // Controlla se la riga contiene il campo "uuid"
21          if (lineFields.indexOf("\"uuid\":") != -1) {
22              int colonPos = lineFields.indexOf(':');
23              String uuid = lineFields.substring(colonPos + 1);
24              uuid.trim();
25              uuid.remove(0, 1);
26              uuid.remove(uuid.length() - 1, 1);
27              uuidValue = uuid;
28              Serial.println("UUID trovato: " + uuidValue);
29              requestProcessed = true;
30          } else {
31              Serial.println("UUID non trovato!");
32          }
33        }
34
35        if (requestProcessed) {
36            // Creazione del corpo della risposta
37            client.println("HTTP/1.1 200 OK");
38            client.println("Content-Type: application/json");
39            client.println();
40
41            String jsonResponse = "{ \"success\": true, \"message\":
      \"UUID ricevuto con successo\", \"arduinoUUID\": \"";
42            jsonResponse += arduinoUUID;
43            jsonResponse += "\"}";
44            client.println(jsonResponse);
45
46            // Salva lo UUID nella EEPROM
47            for (int i = 0; i < uuidValue.length(); ++i) {
48              EEPROM.write(eepromAddress + i, uuidValue.c_str()[i]);
49            }
50          } else {
51            client.println("HTTP/1.1 400 Bad Request");
52            client.println("Content-Type: application/json");
53            client.println();
54
55            String errorResponse = "{ \"success\": false, \"message
     \": \"Bad Request\" }";
56
57            client.println(errorResponse);
58          }
59      }
60      client.stop();
61    }
62  }
```

Codice 5.2: Associazione dispositivo-utente: Fase di gestione della richiesta HTTP

Capitolo 6

Algoritmo per la creazione della mappa di una città

Il presente capitolo si focalizza sull'algoritmo per la creazione di un grafo NetworkX, finalizzato alla rappresentazione della mappa completa di una città. Questo grafo è fondamentale per implementare i servizi che verranno discussi nel capitolo 7. La sua costruzione consente di superare le limitazioni della tesi precedente, estendendo l'area di analisi oltre San Salvario e fornendo una rappresentazione completa delle strade della città di Torino.

L'algoritmo è stato progettato con un approccio altamente modulare, consentendo una facile adattabilità ad altre città. Per effettuare questa transizione, è sufficiente modificare il valore del campo `name` indicato nel Codice 6.1 e l'identificativo univoco della città assegnato da OpenStreetMap[1] all'interno del parametro `relation` nel Codice 6.2. Questo identificativo può essere trovato utilizzando la funzione di ricerca sul sito ufficiale all'indirizzo https://www.openstreetmap.org.

6.1 Individuazione dei nodi

In questa fase l'obiettivo dell'algoritmo è l'individuazione dei nodi, che consiste nell'identificare un incrocio tra due o più strade. Sono state realizzate due versioni di questa fase, poiché la prima si è rivelata poco efficiente a causa del numero eccessivo di chiamate API necessarie per ottenere i dati desiderati.

[1]OpenStreetMap (OSM) è un progetto collaborativo *open source* che mira a creare e fornire dati geografici liberi e gratuiti al mondo intero. Gli utenti possono contribuire alla mappa aggiungendo, modificando e migliorando le informazioni geografiche, consentendo la creazione di mappe dettagliate e aggiornate globalmente.

Nella prima versione, è stato utilizzato il servizio Overpass API, un'interfaccia di interrogazione che fornisce accesso ai dati di OpenStreetMap. In particolare, inizialmente veniva eseguita una *query* per ottenere tutte le informazioni sulle strade di Torino. Successivamente, venivano estratti e salvati i nomi delle strade in un file. Dopodiché, veniva effettuata un'iterazione su quella lista, cercando di individuare tutte le possibili intersezioni. Questo veniva eseguito effettuando una chiamata API al servizio Overpass per ogni coppia di strade selezionate e veniva applicato un filtro per Torino per essere sicuri di ottenere gli incroci nell'area di interesse. Al termine di tutte le iterazioni, gli incroci trovati venivano salvati in un file. Tuttavia, questo approccio presentava criticità in termini di tempi, poiché una singola chiamata API richiedeva in media circa 15 secondi per restituire una risposta. Considerando il numero di strade coinvolte (circa 2700), l'approccio risultava inefficace e insostenibile. Inoltre, era stata riscontrata l'impossibilità di filtrare correttamente per la città di Torino, poiché tale filtro restituiva informazioni sulla Città Metropolitana di Torino, includendo i paesi limitrofi a Torino. Di fronte a tali problematiche, si è deciso di optare per una soluzione alternativa che seguisse lo stesso ragionamento, ma che fosse più efficiente.

Dopo un'attenta analisi delle criticità riscontrate nella prima versione, è emerso che per superare tali limitazioni era fondamentale ottenere un file contenente tutte le informazioni sulla città di Torino. In questo modo, non sarebbe stata più necessaria alcuna chiamata API per individuare gli incroci. A tale scopo, sono state nuovamente utilizzate le API offerte da Overpass per ottenere un file GeoJSON contenente tutti i punti e le relative informazioni della Città Metropolitana di Torino[2], come mostrato nel Codice 6.1. Il GeoJSON è un formato aperto alla codifica di dati geospaziali basato su JSON[3], che consente di rappresentare informazioni geografiche implementando il supporto a diversi tipi di geometria come `Point`, `LineString` e `Polygon`. In particolare una `LineString` rappresenta una sequenza di segmenti di linea connessi, formati da due punti che definiscono il punto di inizio e di fine, che rappresentano una strada. Mentre, `Polygon` rappresenta un'area chiusa all'interno di una superficie piana, delimitata da una sequenza di coordinate che definiscono i vertici del poligono e rappresentano una zona delimitata [14].

```
1 [out:json];
2 area[name="Torino"]->.a;
3 way(area.a)[highway];
4 out geom;
```

Codice 6.1: *Query* per ricavare le informazioni sulla Città Metropolitana di Torino

[2]La Città Metropolitana di Torino, anche nota come Torino Metropoli, comprende 312 comuni e risulta essere la città metropolitana più estesa d'Italia. Il capoluogo è Torino.

[3]JSON (JavaScript Object Notation) è un formato di scrittura dati leggibile e flessibile, basato su JavaScript, organizzato in coppie chiave-valore per scambiare informazioni strutturate.

La Tabella 6.1 mostra i campi più rilevanti presenti nel file GeoJSON ottenuto dalla *query* precedente.

Campo	Descrizione	Esempio
@id	Identificativo della via	"way/22885550"
highway	Tipo di strada	"tertiary"
name	Nome della strada	"Via Nizza"
geometry	Oggetto geometrico	{"type": "LineString", "coordinates": [[lon, lat]]}

Tabella 6.1: Struttura del file contenente i punti della Città Metropolitana di Torino

La Figura 6.1 mostra un sottoinsieme del risultato del Codice 6.1.

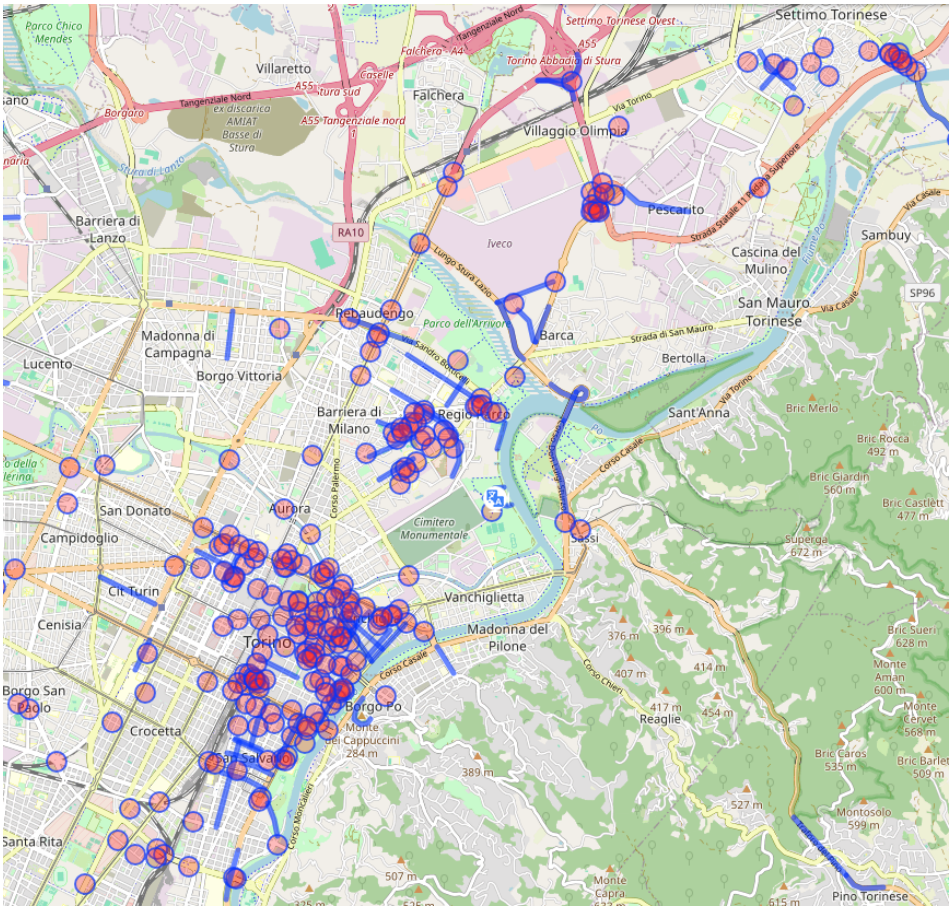

Figura 6.1: Mappa di una sottosezione delle strade della Città Metropolitana di Torino

Tuttavia, come evidenziato dalla Figura 6.1, è emerso nuovamente il problema della presenza di punti esterni alla città di Torino. Per risolvere tale problematica, è stato ricavato il poligono che delimita la città di Torino tramite l'uso delle API offerte da Overpass, come mostrato nel Codice 6.2. Tale poligono viene fornito sotto forma di file GeoJSON e verrà impiegato per eliminare i punti al di fuori di esso.

```
1 [out:json];
2 relation(43992);
3 out geom;
```

Codice 6.2: *Query* per ricavare il poligono della città di Torino

La Figura 6.2 mostra il poligono della città di Torino.

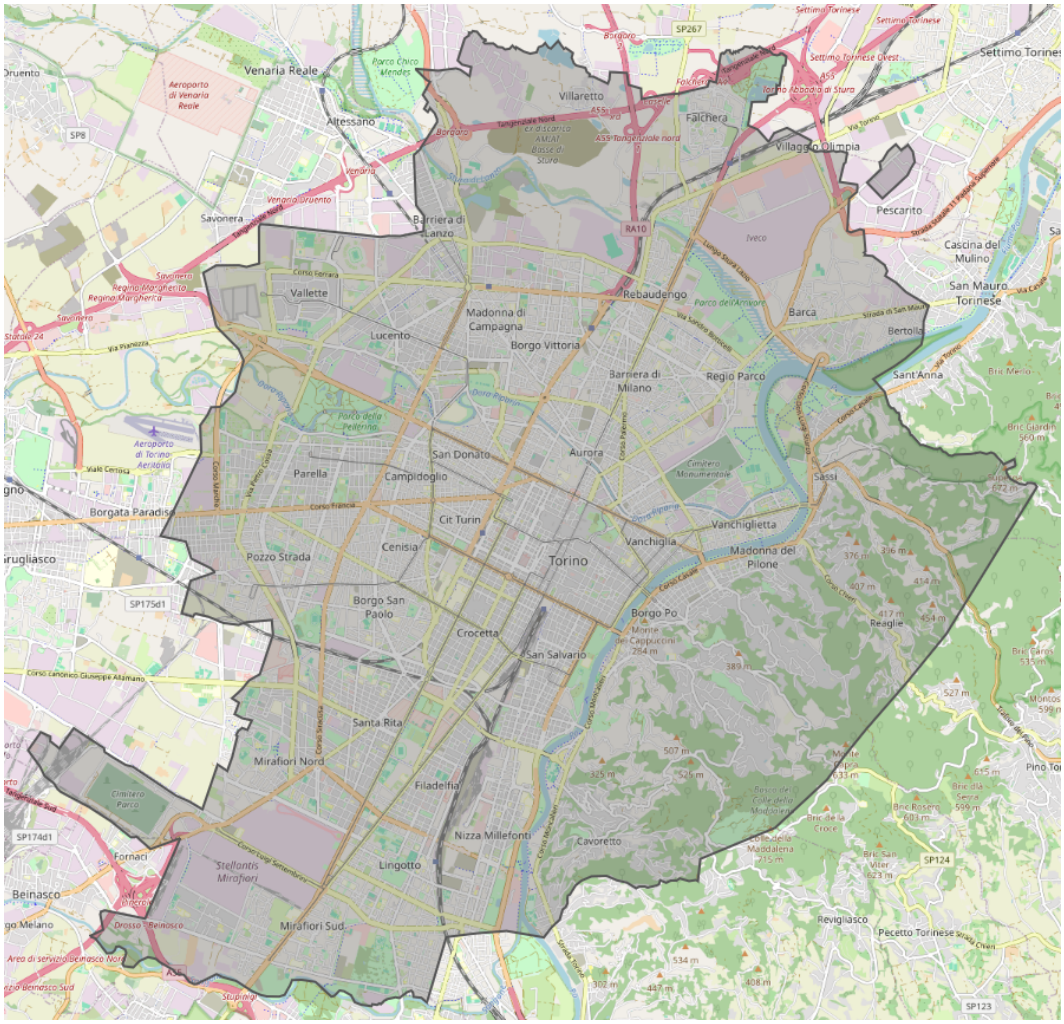

Figura 6.2: Poligono della città di Torino

Dopo aver ottenuto i due file GeoJSON da usare in input, ha inizio la prima fase dell'algoritmo. Inizialmente, vengono caricati i file `turin_geojson` e `turin_polygon`, che rappresenteranno rispettivamente tutte le informazioni della Città Metropolitana di Torino e i punti che formano il perimetro della città di Torino. Per poter fare uso di tale poligono, è stata utilizzata la classe `geometry` fornita dalla libreria `shapely`, la quale definisce il tipo geometrico `Polygon`. Successivamente, si procede all'estrazione dei nomi delle strade e delle relative coordinate che le descrivono. Dopo un'attenta analisi della struttura del file `turin_geojson`, si è deciso di considerare solamente il tipo `LineString`, in quanto rappresenta una via, scartando il tipo `Polygon`, che rappresenta un'area e non è rilevante ai fini dell'algoritmo. Dopo aver ricavato i nomi delle strade e le coordinate dei punti che le compongono, a ciascun punto viene assegnato un identificativo univoco v4 tramite la funzione `generate_unique_uuid`. Ciò è stato necessario in quanto le coordinate presenti nel file `turin_geojson` non dispongono di alcun identificativo univoco. Inoltre, per ogni punto verrà effettuato un conteggio delle sue occorrenze in quanto da analisi è emerso che: se un punto compare almeno due volte, allora esso rappresenta un incrocio. Il Codice 6.3 mostra il processo sopra citato.

```
import uuid

# Funzione per convertire lo UUID in un JSON serializzabile
def serialize_uuid(uuid_obj):
    return str(uuid_obj)

# Funzione per generare un nuovo UUID
def generate_unique_uuid():
    while True:
        new_uuid = uuid.uuid4()
        if new_uuid not in assigned_uuids:
            assigned_uuids.add(new_uuid)
            return new_uuid

# Estrai le strade e i loro punti
for feature in features:
    if feature["geometry"]["type"] == "LineString" and "name" in
   feature["properties"]:
        road_name = feature["properties"]["name"]
        coordinates = feature["geometry"]["coordinates"]

        if road_name not in roads:
            roads[road_name] = []

        for lon, lat in coordinates:
            # Crea un punto della strada con un id univoco
            road_point = {
                "id": serialize_uuid(generate_unique_uuid()),
```

```
28                 "lon": lon,
29                 "lat": lat,
30             }
31             roads[road_name].append(road_point)
32
33             # Conta quante volte una coordinata viene ripetuta
34             coordinate_key = (lon, lat)
35             coordinate_counts[coordinate_key] = (
36                 coordinate_counts.get(coordinate_key, 0) + 1
37             )
38
39 # Filtra le coordinate che si ripetono almeno due volte
40 filtered_coordinates = {
41     key: value for key, value in coordinate_counts.items() if
       value >= 2
42 }
```

Codice 6.3: Individuazione degli incroci della Città Metropolitana di Torino

Una volta individuati tutti gli incroci presenti nella Città Metropolitana di Torino, questi verranno filtrati sulla base del poligono della città di Torino, al fine di escludere i punti al di fuori dei suoi confini. Per fare ciò, è stato fatto uso del file `turin_polygon` e tramite la funzione `is_point_inside_turin_polygon` viene verificato se il punto si trova all'interno del poligono.

```
1  # Funzione per verificare se un punto si trova all'interno del
       poligono di Torino
2  def is_point_inside_turin_polygon(longitude, latitude,
       torino_polygon):
3      point = Point(longitude, latitude)
4      return point.within(torino_polygon)
5
6  # Filtra le strade che si trovano all'interno del poligono
7  filtered_roads_inside_turin = {
8      road: [
9          point
10         for point in points
11         if is_point_inside_turin_polygon(point["lon"], point["lat"
       ], torino_polygon)
12     ]
13     for road, points in filtered_roads_not_empty.items()
14 }
```

Codice 6.4: Filtraggio dei punti all'interno del poligono di Torino

Dopo aver filtrato ed ottenuto tutti gli incroci di Torino, si procede con il loro salvataggio nel database MongoDB, utilizzando la libreria `PyMongo`. Queste informazioni costituiranno la base per l'implementazione dei servizi descritti nel capitolo 7. I dati sui incroci verranno modellati per essere salvati in un formato compatibile con le *query* geospaziali: `{"type": "Point", "coordinates": {"lat", "lon"}}`.

Dopo aver completato l'inserimento dei dati nel database, verranno salvati i seguenti file:

- **street_nodes.json**: Contiene informazioni dettagliate sugli incroci delle strade. Ogni via è associata a una lista di incroci, ciascuno identificato da un identificativo univoco e caratterizzato dalle relative coordinate di longitudine e latitudine. Il codice 6.5 mostra la struttura del file, il quale sarà utilizzato come file di input per la seconda fase dell'algoritmo.
- **street_names_list.json**: Contiene un elenco completo di tutte le vie considerate nell'analisi.

```
{
    "Via Nizza": [
        {
            "id": "25d3b186-401e-47e2-b2fa-6c07ecd18a43",
            "lon": 7.668122,
            "lat": 45.034576
        },
       {
            "id": "b7ddadb5-9786-4753-a129-de7b6548a404",
            "lon": 7.664727,
            "lat": 45.025024
        },
    ],
    "Via Giulio Biglieri": [
        {
            "id": "76ca553b-2841-4d48-a546-54df358e1e9c",
            "lon": 7.668122,
            "lat": 45.034576
        },
    ],
    "Via Lavagna": [
        {
            "id": "84bcebf8-2e43-40b8-9ac7-58d28432974b",
            "lon": 7.667844,
            "lat": 45.033856
        },
    ]
}
```

Codice 6.5: Struttura del file street_nodes.json

6.2 Creazione della mappa

Una volta ottenuta la lista di tutti gli incroci della città di Torino, corrispondenti ai nodi del grafo, la seconda fase dell'algoritmo si concentra sulla creazione di un grafo NetworkX della città. Tale grafo viene costruito, associando un arco tra due nodi, rappresentando così le strade della città.

Questa fase inizia con la creazione di un grafo indiretto utilizzando la libreria NetworkX. Successivamente, viene caricato il file `street_nodes.json` ottenuto in precedenza, vengono estratte le informazioni sugli incroci delle strade e aggiunte come nodi del grafo. Ogni nodo è caratterizzato dall'attributo `node_id` che rappresenta l'identificativo univoco del nodo, e dagli attributi `lat` e `lon` che rappresentano rispettivamente latitudine e longitudine del nodo.

```
# Creazione di un grafo indiretto
G = nx.Graph()

# Aggiunta dei nodi al grafo
for street, nodes in street_nodes.items():
    for node_data in nodes:
        node_id = node_data["id"]
        lat = node_data["lat"]
        lon = node_data["lon"]
        G.add_node(node_id, lat=lat, lon=lon)
```

Codice 6.6: Aggiunta dei nodi della città al grafo NetworkX

La Figura 6.3 mostra la mappa di Torino rappresentata tramite i nodi.

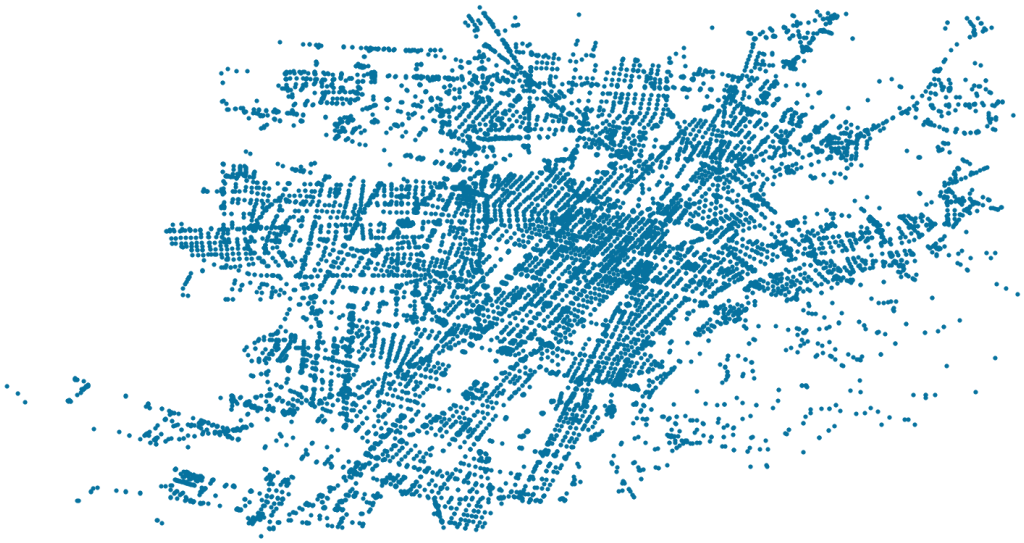

Figura 6.3: Mappa di Torino rappresentata tramite i nodi

Una volta aggiunti tutti i nodi all'interno del grafo si procede con l'associazione di un arco tra due nodi. L'obiettivo è quello di cercare, per ogni strada, i due nodi più vicini lungo la stessa strada e di aggiungere un arco tra di essi nel grafo. La distanza tra due nodi è calcolata tramite la funzione `calculate_distance`, che utilizza le coordinate di latitudine e longitudine e la funzione `great_circle` per calcolare la distanza in metri tra due nodi.

```
# Funzione per calcolare la distanza tra due punti, date le loro coordinate di latitudine e longitudine.
def calculate_distance(node1, node2):
    pos1 = (node1["lat"], node1["lon"])
    pos2 = (node2["lat"], node2["lon"])
    return great_circle(pos1, pos2).meters

# Crea archi tra i nodi della stessa strada
for street1, nodes1 in street_nodes.items():
    for idx1, node1 in enumerate(nodes1):
        min_distance = float("inf")
        nearest_node = None  # Inizialmente None, in quanto non esiste alcun nodo vicino
        for street2, nodes2 in street_nodes.items():
            if street1 != street2:
                continue
            for idx2, node2 in enumerate(nodes2):
                if node1["id"] != node2["id"] and idx2 > idx1:
                    distance = calculate_distance(node1, node2)
                    if distance < min_distance:
                        min_distance = distance
                        nearest_node = node2

        # Aggiungi un arco solo dopo aver trovato il nodo meno distante
        if nearest_node is not None:
            G.add_edge(node1["id"], nearest_node["id"], weight=min_distance)
```

Codice 6.7: Creazione degli archi fra i nodi della mappa

Dopo questa operazione, il grafo è completo e verrà visualizzato con i nodi e gli archi posizionati secondo le coordinate di latitudine e longitudine tramite la funzione `show`. La Figura 6.4 mostra il risultato di una sotto area della città di Torino sovrapposta alla mappa fornita da Google Maps. Come si può evincere, il risultato ottenuto è pressoché identico a quello reale.

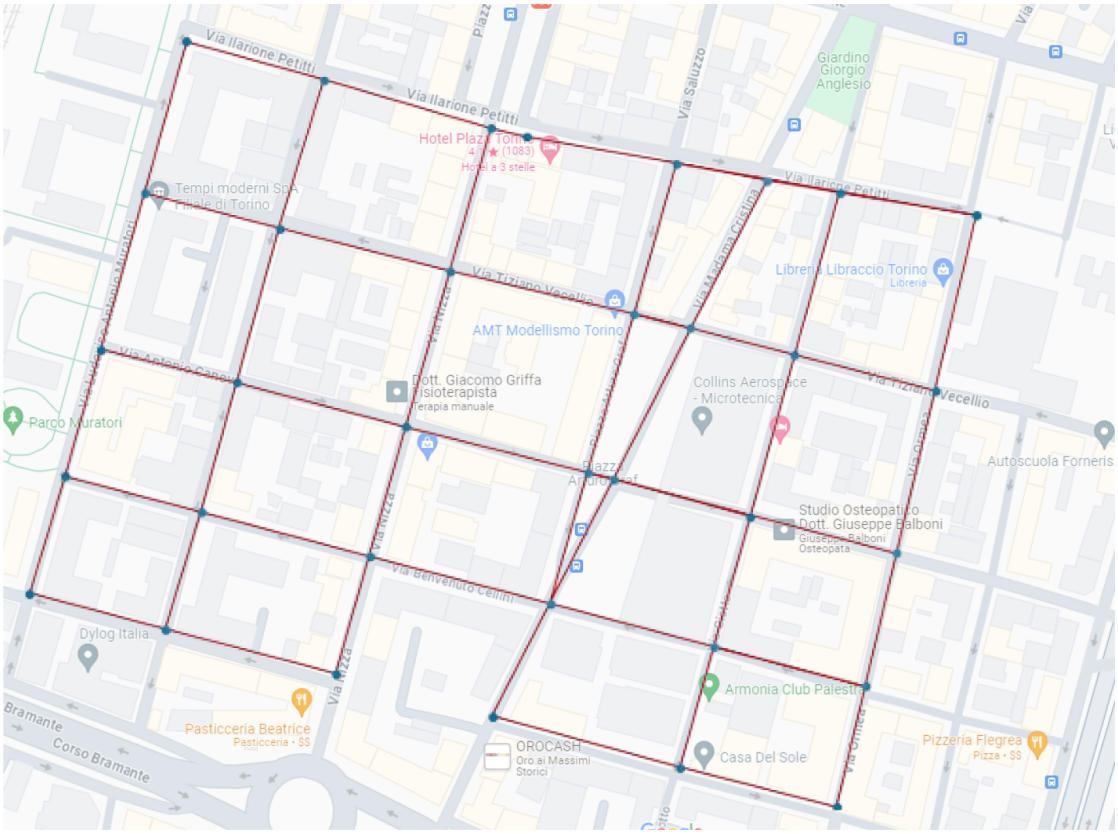

Figura 6.4: Sovrapposizione del risultato dell'algoritmo con Google Maps

Capitolo 7

Implementazione dei servizi

Questo capitolo si propone di esaminare in dettaglio i servizi che rappresentano l'anello di congiunzione tra i dati raccolti dal dispositivo Arduino, le informazioni sui nodi ottenute dalla creazione della mappa della città di Torino e la visualizzazione delle informazioni tramite l'applicazione.

Il primo servizio è fondamentale per migliorare la precisione delle coordinate ricevute dal sensore GPS del dispositivo Arduino. Successivamente, verranno esaminati una serie di servizi progettati per gli utenti, iniziando con la procedura di associazione di un nuovo dispositivo a un utente e proseguendo con il recupero dei dispositivi già associati e la loro rimozione. Saranno inoltre descritti i servizi relativi al recupero di tutti i percorsi dell'utente e di un percorso specifico, nonché il calcolo del percorso ottimale basato sui valori di PM10, offrendo agli utenti uno strumento per una mobilità più consapevole e salutare. Infine, saranno descritti i servizi dedicati al recupero dei valori di PM10 per una specifica area, giorno e ora, e al recupero dei valori di PM10 per una specifica area e giorno per visualizzare una *heatmap*.

7.1 Inserimento dei dati relativi a un percorso

Questo servizio, come discusso in 5.1, è stato realizzato con lo scopo di normalizzare le coordinate acquisite dal sensore GPS installato sul dispositivo Arduino. Senza questa normalizzazione, la precisione delle coordinate non raggiungeva livelli soddisfacenti. Per implementare questo servizio, è stato realizzato un microservizio dedicato denominato `bike-monitoring-location-normalizer`, che ingloba tutte le operazioni presenti nel flusso Node-RED mostrato nella Figura 3.3.

Una volta che il dispositivo Arduino invia i dati al Broker MQTT, viene recuperato il `lastMsgId` e viene contattato il microservizio tramite l'*endpoint* `POST`

`/v1/location-normalizer/add-trip`, avviando così il processo di normalizzazione dei dati ricevuti. La Tabella 7.1 mostra i dati acquisiti e inoltrati al microservizio dal dispositivo Arduino.

Campo	Descrizione	Esempio
_id	Identificativo del documento	"5cc441bf521fbe69411c25b3f4c591fb"
pm100	Valore di PM10	4
pm25	Valore di PM2.5	4
s	Numero di satelliti	"06"
lt	Latitudine in decimali	"4503.5415"
ln	Longitudine in decimali	"00740.8434"
d_lt	Direzione latitudine	"N"
d_ln	Direzione longitudine	"E"
q	Qualità dell'aria	"1"
hd	HDOP	"1.29"
lastMsgId	Timestamp di inizio del percorso	"1682698931799"
UUID	Identificativo dell'utente	"x9J8STAehwUJhKHr6UIDmaIfvBc2"

Tabella 7.1: Struttura dei dati inviati dal dispositivo Arduino

Una volta arrivati al microservizio, i dati vengono elaborati per essere resi compatibili con il formato del documento nel database. Viene effettuato il *parsing* del campo `s` da stringa a intero, viene creato il `tripId` concatenando la stringa `TRIP_` al campo `lastMsgId`, viene calcolato il ***timestamp*** e le coordinate di `latitudine` e `longitudine` utilizzando le funzioni presenti nell'implementazione precedente del flusso Node-RED, come mostrato in Figura 3.3. Inoltre, viene calcolato il ***timeSlot*** a partire dal *timestamp*, definendo così la fascia oraria del giorno in cui il dato è stato ricevuto.

La parte cruciale del processo risiede nella normalizzazione della coordinata ricevuta. La funzione `findNearestPoint`, mostrata nel Codice 7.1, è progettata per individuare il punto più vicino a una coordinata specifica interrogando la collezione `raw_intersections` presente nel database, come descritto in 6.1. Tale collezione contiene tutte le informazioni relative agli incroci della città di Torino, consentendo così di normalizzare il dato con elevata precisione. La funzione restituisce un oggetto di tipo `Points`, la cui struttura è descritta nella Tabella 7.2.

```
1 async findNearestPoint(latitude: number, longitude: number):
    Promise<Points> {
2     try {
3       const nearestPoint = await this.rawIntersectionsCollection.
    findOne({
```

```
4           coordinates: {
5             $near: {
6               $geometry: {
7                 type: 'Point',
8                 coordinates: [latitude, longitude],
9               },
10              $maxDistance: 400,
11            },
12          },
13        });
14
15        const result: Points = {
16          rawCoordinates: {
17            lat: latitude,
18            lon: longitude,
19          },
20          nearestCoordinates: {
21            lat: nearestPoint.coordinates.lat,
22            lon: nearestPoint.coordinates.lon,
23          },
24          coordinatesId: nearestPoint._id.toString(),
25        };
26
27        return result;
28      } catch (error) {
29        throw new Error(
30          `Errore nella ricerca del punto vicino: ${error.message}`
31        );
32      }
33  }
```

Codice 7.1: Funzione per la ricerca del punto più vicino

Campo	Descrizione	Esempio
coordinateId	Identificativo della coordinata normalizzata	"25d3b186-401e-47e2-b2fa-6c07ecd18a43"
rawCoordinates	Coordinata in ingresso	{"lat": 45.0537, "lon": 7.6755}
coordinates	Coordinata normalizzata	{"lat": 45.0536, "lon": 7.6754}

Tabella 7.2: Struttura dell'oggetto `Points`

Dopo aver completato il processo di normalizzazione e aver manipolato i dati per renderli compatibili con il formato presente nel database, essi vengono inseriti nella collezione `data_sensors` attraverso una richiesta `POST`, seguendo la struttura riportata nella Tabella 7.3.

Campo	Descrizione	Esempio
_id	Identificativo del documento	"5cc441bf521fbe69411c25b3f4c591fb"
pm100	Valore di PM10	4
pm25	Valore di PM2.5	4
d_lt	Direzione latitudine	"N"
d_ln	Direzione longitudine	"E"
q	Qualità dell'aria	"1"
hd	HDOP	"1.29"
isSimulated	Indica se il dato è simulato	false
satellites	Numero di satelliti	6
ts	Timestamp	1709554613000
timeSlot	Fascia oraria	1
tripId	Identificativo del percorso	"TRIP_1682698931799"
UUID	Identificativo dell'utente	"x9J8STAehwUJhKHr6UIDmaIfvBc2"
coordinatesId	Identificativo della coordinata normalizzata	"25d3b186-401e-47e2-b2fa-6c07ecd18a43"
rawCoordinates	Coordinata in ingresso	{"lat": 45.0537, "lon": 7.6755}
coordinates	Coordinata normalizzata	{"lat": 45.0536, "lon": 7.6754}

Tabella 7.3: Struttura del documento relativo a un percorso

Dopo aver completato il processo di inserimento dei dati per un percorso, è necessario calcolare i seguenti campi che saranno necessari per i servizi futuri:

- **Durata (`duration`)**: Indica la durata totale del percorso espressa nel formato `HH:MM:SS`.
- **Distanza totale (`totalDistance`)**: Rappresenta la distanza totale del percorso, espressa in metri.
- **Indice di qualità (`quality`)**: Rappresenta l'indice di qualità del percorso utilizzato come criterio per selezionare il percorso meno inquinato.

Per calcolare tali campi, è stato realizzato uno *script* Node.js progettato per elaborare e aggiornare i dati di un percorso specifico. In seguito all'avvio dello

script, all'amministratore del sistema viene richiesto di inserire l'identificativo univoco del percorso (`tripId`) su cui desidera operare. Successivamente, viene stabilita una connessione al database MongoDB per interagire con i dati presenti nel database. Vengono quindi recuperate le coordinate, i valori di PM10 e il *timestamp* del percorso in questione. Partendo da questi dati, lo *script* itera su di essi per ricavare i seguenti campi:

- La durata totale del percorso viene calcolata determinando la differenza tra il *timestamp* successivo e quello attuale per ogni punto del percorso. I valori ottenuti vengono sommati per ottenere la durata complessiva del percorso.
- La distanza totale del percorso viene calcolata tramite la funzione `getDistance` della libreria `geolib`, che consente di calcolare la distanza tra le coordinate geografiche correnti e quelle successive. Sommando tutte queste distanze si ottiene la distanza complessiva del percorso.
- L'indice di qualità viene calcolato moltiplicando la distanza totale del percorso per il valore medio di PM10 registrato durante il percorso. Questo indice è stato progettato per mettere in relazione la lunghezza del percorso con il livello medio di inquinamento registrato durante il percorso.

Tale servizio consente di ottenere una maggiore precisione durante la visualizzazione dei dati, come mostrato in Figura 8.9.

7.2 Interazione dispositivo-utente

I servizi relativi all'interazione tra il dispositivo e l'utente sono gestiti dal microservizio `bike-monitoring-profile`. Questo microservizio offre funzionalità per associare un dispositivo Arduino a un utente, per recuperare tutti i dispositivi associati a un utente e rimuovere un dispositivo tra quelli associati a esso.

7.2.1 Associazione del dispositivo all'utente

Il processo di associazione dispositivo-utente è suddiviso in tre fasi: l'invio della richiesta da parte dell'applicazione, la gestione della richiesta sul dispositivo Arduino e il salvataggio dei dati nel database. In questa sezione, ci concentreremo sulla terza fase, mentre la prima verrà esaminata nel capitolo 8 e la seconda è stata esaminata in 5.2.

Una volta che l'utente ha inoltrato la richiesta HTTP dall'applicazione, essa viene gestita dal dispositivo Arduino, come spiegato in 5.2. Se il dispositivo Arduino fornisce una risposta contenente il proprio identificativo univoco v4, viene

contattato il microservizio tramite l'*endpoint* `POST v1/profile/add-device`, il cui corpo contiene i dati nel formato descritto nella Tabella 7.4.

Campo	Descrizione	Esempio
userUUID	Identificativo dell'utente	"x9J8STAehwUJhKHr6UIDmaIfvBc2"
deviceUUID	Identificativo del dispositivo	"b2049a504a734a02988d66f775eef3f5"

Tabella 7.4: Struttura del corpo della richiesta relativa all'associazione dispositivo-utente

Una volta arrivati al microservizio, viene invocata la funzione `addDevice`, mostrata nel Codice 7.2. Questa funzione verifica se esiste già un documento nella collezione `users`[1] con gli stessi dati presenti nella richiesta. Se non vi è alcun riscontro, viene inserito nella collezione un nuovo documento con i dati del nuovo dispositivo nel formato descritto dalla Tabella 7.5.

```
1  async addDevice(deviceUUID: string, userUUID: string): Promise<
       void> {
2      try {
3        const existingDevice = await this.usersCollection.findOne({
4          deviceUUID: deviceUUID,
5          userUUID: userUUID,
6        });
7
8        if (existingDevice)
9          throw new BadRequestException("Dispositivo associato in
       precedenza");
10
11       const newDevice = {
12         userUUID,
13         isSimulated: false,
14         deviceName: 'Arduino Uno R2',
15         deviceUUID,
16         timestamp: Date.now(),
17       };
18       await this.usersCollection.insertOne(newDevice);
19     } catch (error) {
20       if (error instanceof BadRequestException) {
21         throw error;
22       }
23       throw new Error("Errore durante l'aggiunta di un nuovo
       dispositivo");
24     }
```

[1]Tale collezione contiene le informazioni dei dispositivi Arduino associati agli utenti.

```
25 }
```

Codice 7.2: Associazione dispositivo-utente: Fase di inserimento nel database

Campo	Descrizione	Esempio
_id	Identificativo del documento	"65b7f982e3d2686e9eb2d9c9"
userUUID	Identificativo dell'utente	"x9J8STAehwUJhKHr6UIDmaIfvBc2"
deviceName	Nome del dispositivo	"Arduino Uno R2"
timestamp	Timestamp	1709554613000
isSimulated	Indica se il dispositivo è simulato	false
deviceUUID	Identificativo del dispositivo	"b2049a504a734a02988d66f775eef3f5"

Tabella 7.5: Struttura del documento relativa all'associazione dispositivo-utente

Se l'operazione ha successo, il nuovo dispositivo verrà mostrato nella sezione `Profile` dell'applicazione, come illustrato in Figura 8.10.

7.2.2 Recupero dei dispositivi associati all'utente

Questo servizio consente all'utente di recuperare tutti i dispositivi Arduino a lui associati. Una volta effettuato l'accesso alla sezione `Profile` dell'applicazione, come mostrato in Figura 8.10, viene contattato il microservizio tramite l'*endpoint* `GET v1/profile/devices?userUUID={userUUID}&limit={limit}`. Una volta arrivati al microservizio, viene invocata la funzione `getUserDevices`, mostrata nel Codice 7.3, che accetta due parametri in ingresso: `userUUID` e `limit`. Il primo rappresenta l'identificativo univoco dell'utente, mentre il secondo rappresenta il numero massimo di dispositivi da restituire. Tale funzione restituisce una lista contenente uno o più oggetti `Device`, ordinati dal più recente al meno recente, la cui struttura è riportata nella Tabella 7.6.

```
1 async getUserDevices(userUUID: string, limit: number): Promise<
    Device[]> {
2     try {
3       const query = [
4         { $match: { userUUID: userUUID } },
5         { $sort: { timestamp: -1 } },
6         { $limit: limit },
7       ];
8
9       const queryResult = await this.usersCollection.aggregate(
    query).toArray();
```

```
10
11        const devices: Device[] = queryResult.map((groupedData) => {
12          const device: Device = {
13            deviceUUID: groupedData.deviceUUID,
14            deviceName: groupedData.deviceName,
15          };
16          return device;
17        });
18
19        return devices;
20      } catch (error) {
21        throw new Error(`Errore durante la ricerca dei dispositivi
    associati a un utente: ${error.message}`);
22      }
23  }
```

Codice 7.3: Funzione per il recupero dei dispositivi associati a un utente

Campo	Descrizione	Esempio
deviceUUID	Identificativo del dispositivo	"b2049a504a734a02988d66f775eef3f5"
deviceName	Nome del dispositivo	"Arduino Uno R2"

Tabella 7.6: Struttura dell'oggetto `Device`

7.2.3 Rimozione di un dispositivo associato all'utente

Questo servizio consente all'utente di rimuovere un dispositivo Arduino a lui associato. In seguito alla conferma di rimozione del dispositivo, come mostrato in Figura 8.12, viene contattato il microservizio all'*endpoint* `DELETE v1/profile/delete-device?deviceUUID={deviceUUID}&userUUID={userUUID}`. Una volta arrivati al microservizio, viene invocata la funzione `deleteDevice`, mostrata nel Codice 7.4, che accetta due parametri in ingresso: `userUUID` e `deviceUUID`. Il primo rappresenta l'identificativo univoco dell'utente, mentre il secondo rappresenta l'identificativo univoco del dispositivo. Questa funzione verifica se esiste un documento all'interno della collezione `users` con la stessa coppia di dati ricevuti. Se vi è un riscontro, viene rimosso il documento dalla collezione e viene restituito un messaggio di conferma di rimozione.

```
1  async deleteDevice(deviceUUID: string, userUUID: string): Promise<
       void> {
2    try {
3      // Verifica se esiste un documento con i dati specificati
4      const existingDevice = await this.usersCollection.findOne({
5        deviceUUID: deviceUUID,
```

```
6           userUUID: userUUID,
7         });
8
9         if (!existingDevice) {
10          throw new NotFoundException("Dispositivo o utente non
      trovato");
11        }
12
13        await this.usersCollection.deleteOne({
14          deviceUUID: deviceUUID,
15          userUUID: userUUID,
16        });
17
18      } catch (error) {
19        if (error instanceof NotFoundException) {
20          throw error;
21        }
22        throw new Error("Errore durante l'eliminazione del
      dispositivo");
23      }
24  }
```

Codice 7.4: Funzione per la rimozione di un dispositivo associato a un utente

7.3 Gestione dei percorsi effettuati dall'utente

I servizi relativi ai percorsi sono implementati nel microservizio *bike-monitoring-trips*. Questo microservizio offre funzionalità per recuperare tutti i percorsi effettuati da un utente, ottenere i dettagli di un percorso specifico effettuato dall'utente e calcolare il percorso meno inquinato basandosi sui livelli di PM10.

7.3.1 Recupero di tutti i percorsi

Questo servizio consente all'utente di recuperare le informazioni sui percorsi effettuati in passato. Quando si accede alla sezione `Trip` dell'applicazione, come mostrato in Figura 8.8, viene contattato il microservizio all'*endpoint* `GET /v1/user/{userId}/trips`. Una volta arrivati al microservizio, viene invocata una funzione che accetta come parametro l'identificativo univoco dell'utente (`userId`). Questa funzione inizia verificando se l'utente è presente nel database. Questo controllo è cruciale per garantire che solo gli utenti autorizzati possano accedere alle informazioni dei percorsi. Successivamente, vengono recuperate le informazioni sui percorsi effettuati dall'utente. La funzione dopo aver elaborato e ordinato i dati dal più recente al meno recente, restituisce una lista contenente uno o più oggetto `Trip`, la cui struttura è descritta nella Tabella 7.7.

Campo	Descrizione	Esempio
id	Identificativo del percorso	"TRIP_123456789"
startDate	Data e ora di inizio percorso	"2024-03-15T08:00:00.000Z"
endDate	Date e ora di fine percorso	"2024-03-15T09:30:00.000Z"
pm25Avg	Media dei valori di PM2.5 del percorso	20.87
pm100Avg	Media dei valori di PM10 del percorso	38.20

Tabella 7.7: Struttura dell'oggetto Trip

7.3.2 Recupero di un percorso specifico

Questo servizio consente all'utente di recuperare le informazioni relative a un percorso specifico effettuato in passato. Dopo aver visualizzato la lista dei percorsi nella sezione `Trip` dell'applicazione, è possibile ottenere maggiori dettagli in merito a un percorso specifico, come mostrato in Figura 8.9, cliccando su di esso. Verrà quindi contattato il microservizio all'*endpoint* `GET /v1/user/{userId}/trips/{tripId}`. Una volta arrivati al microservizio, viene invocata una funzione che accetta due parametri in ingresso: `userId` e `tripId`. Il primo rappresenta l'identificativo univoco dell'utente, mentre il secondo rappresenta l'identificativo univoco del percorso. Questa funzione inizia verificando se l'utente è presente nel database. Questo passaggio è cruciale per garantire che solo gli utenti autorizzati possano accedere ai dettagli del viaggio. Se l'utente è presente, vengono recuperate le informazioni relative a quel percorso specifico e vengono restituite in un oggetto `TripDetails`, la cui struttura è descritta nella Tabella 7.8.

Campo	Descrizione	Esempio
id	Identificativo del percorso	"TRIP_123456789"
startDate	Data e ora di inizio percorso	"2024-03-15T08:00:00.000Z"
endDate	Date e ora di fine percorso	"2024-03-15T09:30:00.000Z"
pm25Avg	Media dei valori di PM2.5	20.87
pm100Avg	Media dei valori di PM10	38.20
duration	Durata totale	"00:30:20"
totalDistance	Distanza totale	1400
points	Lista dei punti	[{"date": "2024-03-12T10:30:00Z", "coordinates": {"lat": 45.034576, "lon": 7.668122}}]

Tabella 7.8: Struttura dell'oggetto `TripDetails`

7.4 Ricerca del percorso meno inquinato

Questo servizio si propone di offrire agli abitanti di Torino uno strumento per individuare il percorso meno inquinato. Tale servizio ritorna particolarmente utile nelle giornate più inquinate, dove si sconsiglia di eseguire attività sportive all'aperto. L'obiettivo principale è tutelare la salute dei cittadini, offrendo un percorso ottimale basato sui livelli di PM10. Dopo aver selezionato i punti di partenza e di arrivo nella sezione `Map` dell'applicazione, come mostrato nella Figura 8.5, viene contattato il microservizio all'*endpoint* `GET v1/best-route?startLatitude={startLat}&startLongitude={startLon}&endLatitude={endLat}&endLongitude={endLon}`. Una volta raggiunto il microservizio, viene invocata una funzione che accetta quattro parametri in ingresso: `startLat`, `startLon`, `endLat`, `endLon` rappresentanti le coordinate del punto di partenza e di arrivo. Questa funzione inizia con la validazione delle coordinate fornite, assicurandosi che siano valide e rispettino le linee guida definite nella documentazione di MongoDB per le *query* geospaziali, come riportato in Figura 7.1. Successivamente, vengono recuperati i dati dei punti più vicini entro un raggio di 300 metri dal punto di partenza e di arrivo entro un periodo di 30 giorni. Dopo aver individuato tutti i possibili punti, viene selezionato il punto con il valore di ***quality*** più alto, calcolato secondo il metodo descritto in 7.1, la cui struttura è riportata nella Tabella 7.3. In seguito, viene recuperato l'identificativo del percorso associato al punto selezionato (`tripId`) e vengono recuperati tutti i dettagli di quel percorso specifico. Infine, la funzione restituisce una lista contenente uno o più oggetti `BestRoutePoint` che andranno a comporre il percorso meno inquinato, la cui struttura è riportata nella Tabella 7.9.

Campo	Descrizione	Esempio
id	Identificativo del documento	"5cc441bf521fbe69411c25b3f4c591fb"
tripId	Identificativo del percorso	"TRIP_123456789"
duration	Durata totale	"00:30:20"
totalDistance	Distanza totale	1400
coordinates	Coordinata del punto	{"lat": 45.034576, "lon": 7.668122}
averagePM100	Media dei valori di PM10	38.20

Tabella 7.9: Struttura dell'oggetto `BestRoutePoint`

7.5 Recupero dei valori mediani di PM10

Prima di esaminare i restanti servizi, è fondamentale comprendere il contesto di funzionamento attraverso una premessa. Per ottimizzare la ricerca dei valori di PM10 per ogni nodo in modo rapido ed efficiente, è stata creata la collezione `daily_sensor_data` nel database. Questa collezione contiene i dati giornalieri relativi ai valori mediani di PM10 per ogni nodo, come illustrato nel Codice 7.5. Tale approccio si dimostra vantaggioso poiché calcolare e memorizzare tali valori una volta al giorno, anziché in tempo reale a ogni richiesta, consente di ridurre il carico computazionale dei dati da elaborare. È importante notare che si è deciso di fare uso della mediana dei valori anziché la media, in quanto quest'ultima è sensibile ai valori estremi in un insieme di dati, mentre la mediana risulta più robusta nei confronti di tali valori, essendo basata sul valore centrale.

```
{
  "nodeId": "25d3b186-401e-47e2-b2fa-6c07ecd18a43",
  "coordinates": {
    "lat": 45.034576,
    "lon": 7.668122
  },
  "dailyMedians": {
    "20240311": {
      "medianPm100Daily": 25.75,
      "medianPm100TimeSlot1": 22.1,
      "medianPm100TimeSlot2": 25.9,
      "medianPm100TimeSlot3": 25.6,
      "medianPm100TimeSlot4": 28.3,
      "medianPm100TimeSlot5": 26.7,
      "medianPm100TimeSlot6": 25.2
    },
    "20240312": {
      "medianPm100Daily": -1,
      "medianPm100TimeSlot1": -1,
      "medianPm100TimeSlot2": -1,
      "medianPm100TimeSlot3": -1,
      "medianPm100TimeSlot4": -1,
      "medianPm100TimeSlot5": -1,
      "medianPm100TimeSlot6": -1
    },
  }
}
```

Codice 7.5: Struttura del documento della collezione daily_sensor_data

Per automatizzare l'aggiornamento dei dati giornalieri relativi ai valori di PM10 per ogni nodo, è stata implementata la pianificazione ed esecuzione di richieste API periodiche. In particolare, è stato fatto uso del modulo `cron`, che consente di pianificare l'esecuzione di una funzione in base a un orario specifico o a un intervallo di

tempo. Ogni giorno alle 00:15, viene eseguita la funzione `makeApiRequest`, mostrata nel Codice 7.6, che contatta l'*endpoint* `POST v1/pollution/process-daily-data`.

```
1  const apiUrl = "http://bike-monitoring-pollution:3013/v1/pollution
       /process-daily-data";
2
3  const makeApiRequest = async () => {
4      try {
5        const response = await axios.post(apiUrl);
6        console.log("Richiesta effettuata con successo:", response.
       data);
7      } catch (error) {
8        console.error("Errore durante la richiesta API:", error.
       message);
9      }
10   };
11
12 cron.schedule("0 15 00 * * *", () => {
13     makeApiRequest();
14 });
```

Codice 7.6: Automazione del calcolo dei valori mediani giornalieri di PM10

Dopo che la funzione `makeApiRequest` ha contattato l'*endpoint* `POST v1/pollution /process-daily-data`, viene eseguita una funzione che calcola i valori mediani di PM10 per ogni nodo e li inserisce nella collezione `daily_sensor_data`. L'esecuzione della funzione inizia acquisendo la data di ieri nel formato `YYYYMMDD`. A partire da questa data, viene creato un intervallo di tempo che va dalla mezzanotte alle 23:59:59 della giornata precedente. Successivamente, vengono recuperate le informazioni dal database che ricadono in questo intervallo. Queste informazioni vengono quindi raggruppate per ogni nodo al fine di effettuare calcoli statistici. In particolare, per ogni nodo vengono calcolati i valori mediani di PM10 e i valori mediani di PM10 per ogni fascia oraria, suddivisa in intervalli di 4 ore. Nel caso in cui non venga trovato alcun valore, viene assegnato il valore -1. Una volta calcolati tutti i valori mediani di PM10 relativi alla giornata precedente, vengono inseriti nel database.

7.5.1 Recupero per area, giorno e ora

Questo servizio consente agli utenti di acquisire i dati sui valori mediani di PM10 relativi a una specifica area geografica, giorno e ora. Dopo aver selezionato l'indirizzo o il punto di partenza, un raggio di ricerca, una data e un'ora, tramite la sezione `Map` dell'applicazione, come mostrato nella Figura 8.6, viene inoltrata una richiesta al microservizio `bike-monitoring-pollution` all'*endpoint* `GET v1/pollution?latitude={lat}&longitude={lon}&range={range}&timestamp ={timestamp}`. Una volta arrivati al microservizio, viene eseguita una funzione che accetta quattro parametri in ingresso: la `latitudine`, la `longitudine`, il

`range` e il `timestamp`. Questa funzione inizia con la validazione delle coordinate fornite, assicurandosi che siano valide e rispettino le linee guida definite nella documentazione di MongoDB per le *query* geospaziali, come riportato in Figura 7.1. Successivamente, viene calcolato il campo `timeslot` a partire dal *timestamp* fornito, assegnando il valore 1 se il *timestamp* ricade nell'intervallo dalle 00:00 alle 04:00, e così via. Infine, vengono recuperate tutte le informazioni dei nodi entro il raggio specificato, nella data indicata e nel *timeslot* ottenuto. La funzione restituisce una lista contenente uno o più oggetti di tipo `Point`, la cui struttura è descritta nella Tabella 7.10.

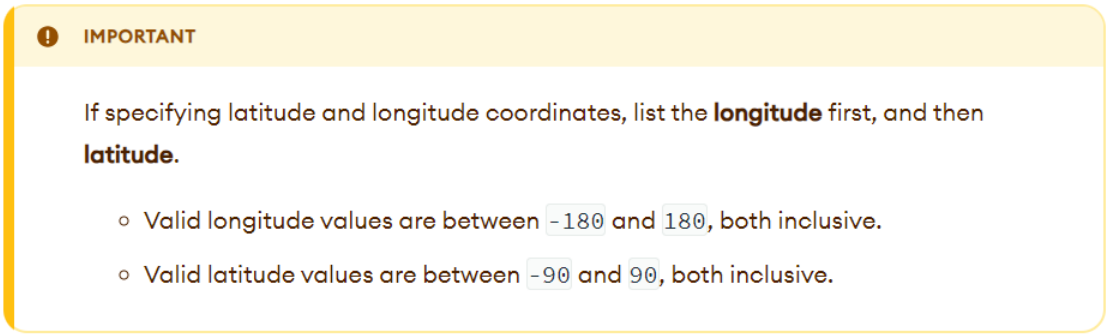

Figura 7.1: Linee guida per l'uso delle *query* geospaziali in MongoDB

Campo	Descrizione	Esempio
nodeId	Identificativo del nodo	"25d3b186-401e-47e2-b2fa-6c07ecd18a43"
pm100Value	Valore del PM10 del nodo	38.20
coordinates	Coordinata del nodo	{"latitude": 45.0536, "longitude": 7.6754}

Tabella 7.10: Struttura dell'oggetto `Point`

7.5.2 Recupero per area e giorno

Questo servizio, consente agli utenti di ottenere dati sui valori mediani di PM10 relativi a una specifica area geografica e giorno. Questo servizio assume grande importanza in quanto, tramite l'utilizzo di una *heatmap* presente nella sezione `Map` dell'applicazione, mostrata nella Figura 8.7, consente agli utenti di individuare le aree con maggiore e minore inquinamento per un dato giorno. Dopo aver inoltrato la richiesta, viene contattato il microservizio `bike-monitoring-pollution` all'*endpoint* `GET v1/pollution/heatmap?latitude={lat}&longitude={lon}&date={date}`. Una volta giunti al microservizio, viene invocata una funzione che accetta tre parametri

in ingresso: la `latitudine`, la `longitudine` e una `data`. Questa funzione inizia con la validazione delle coordinate fornite, assicurandosi che siano valide e rispettino le linee guida definite nella documentazione di MongoDB per le *query* geospaziali, come riportato in Figura 7.1. Successivamente, vengono recuperate tutte le informazioni dei nodi entro un raggio di 1400 metri, a partire dalle coordinate ricevute in ingresso, e nella data specificata. La funzione restituisce una lista contenente uno o più oggetti di tipo `Point`, la cui struttura è descritta nella Tabella 7.10.

Capitolo 8

Implementazione dell'applicazione

Il capitolo si propone di illustrare dettagliatamente le diverse funzionalità offerte dall'applicazione AirSense. Sarà suddiviso in sezioni tematiche, ciascuna dedicata a un aspetto specifico dell'applicazione. Si inizia con una disamina dell'autenticazione, descrivendo il processo di registrazione e accesso all'applicazione e le tecnologie impiegate per garantire un'esperienza sicura e *user-friendly* agli utenti. Successivamente, si esamina la sezione delle mappe, elemento centrale dell'applicazione, che consente agli utenti di interagire con tre servizi distinti: la ricerca del percorso meno inquinato, il recupero dei dati di PM10 per una specifica area, giorno e ora, e la visualizzazione di una *heatmap* basata sui livelli di PM10 per un'area e un giorno determinati. In seguito, viene esaminata la sezione relativa ai percorsi, che fornisce agli utenti informazioni dettagliate sui percorsi passati, con la possibilità di visionare maggiori dettagli su un percorso specifico. Infine, si esamina la sezione del profilo, che permette agli utenti di gestire le proprie informazioni personali e i dispositivi Arduino associati al proprio account.

All'avvio dell'applicazione, agli utenti verrà mostrata la *splashscreen* dell'applicazione seguita immediatamente dalla schermata iniziale, illustrata nella Figura 8.1, che rappresenta il punto di ingresso dell'applicazione stessa.

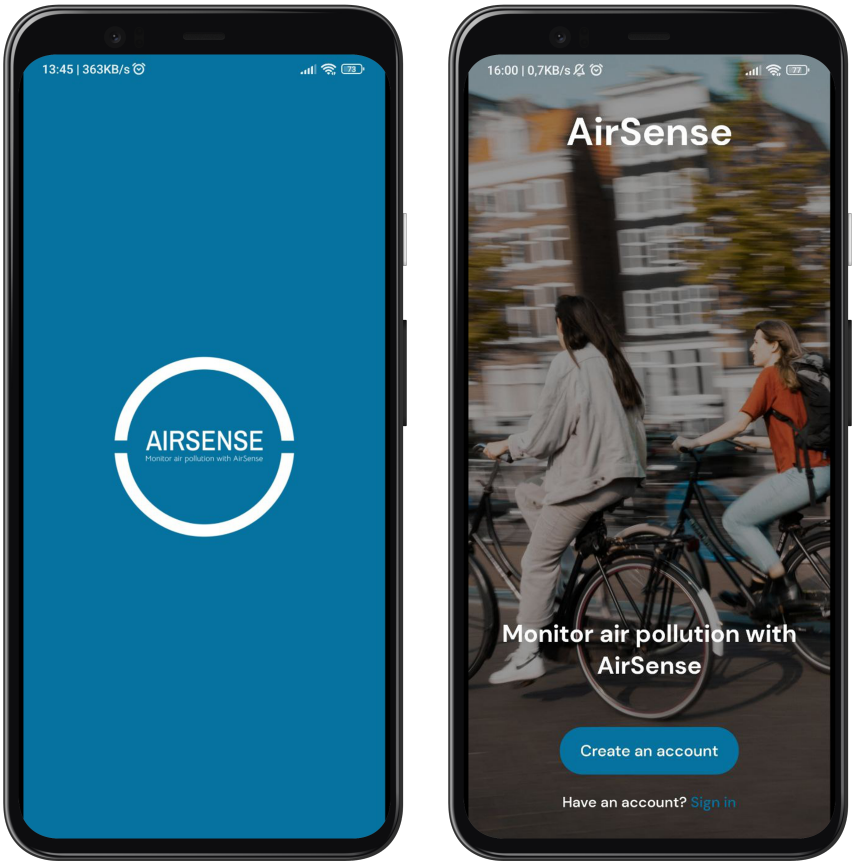

Figura 8.1: Schermata di *splashscreen* e d'ingresso dell'applicazione

8.1 Autenticazione

Per accedere alle funzionalità offerte dall'applicazione, è necessario completare la procedura di autenticazione. Quest'ultima viene gestita tramite Firebase utilizzando la libreria `@react-native-firebase/auth`. Gli utenti hanno la possibilità di autenticarsi utilizzando l'email e la password oppure tramite Google.

Per la gestione della validazione dei dati inseriti durante la fase di registrazione e di accesso, si è fatto uso delle librerie `react-hook-form` e `yup`. La prima offre un modo semplice ed efficiente per gestire i *form* in React Native, mentre la seconda permette di definire regole di validazione personalizzate. Questo consente di garantire che i dati inseriti dagli utenti rispettino il formato desiderato ancor prima di essere elaborati dall'applicazione. Ad esempio, è possibile specificare una lunghezza minima dei caratteri per un determinato campo o verificare la validità di un indirizzo email.

8.1.1 Registrazione

Il processo di registrazione consente agli utenti di creare un nuovo account all'interno dell'applicazione. Durante la fase di registrazione, gli utenti possono scegliere tra due opzioni: utilizzare l'account Google o registrarsi tramite email e password. Nel primo caso, viene visualizzata una lista degli account Google collegati al dispositivo dell'utente, consentendo di selezionarne uno per completare il processo di registrazione e accedere direttamente alla schermata principale dell'applicazione, mostrata nella Figura 8.5. Se l'utente sceglie di registrarsi tramite email e password, dovrà inserire nome, cognome, indirizzo email e dovrà scegliere una password e confermarla. Una volta inseriti correttamente i dati nel *form*, l'utente inoltra la richiesta di registrazione al servizio Firebase, il quale verifica l'unicità dell'email fornita. Se l'email risulta già presente nel database, verrà mostrato un messaggio all'utente. Se la richiesta di registrazione è avvenuta con successo, l'utente viene reindirizzato alla schermata di accesso e riceve un'email di conferma all'indirizzo inserito in fase di registrazione. È importante verificare l'email tramite il *link* di conferma ricevuto, poiché l'accesso all'applicazione non sarà consentito fino a quando l'email non sarà stata verificata. La Figura 8.2 mostra la schermata di registrazione sia vuota che compilata con i dati necessari.

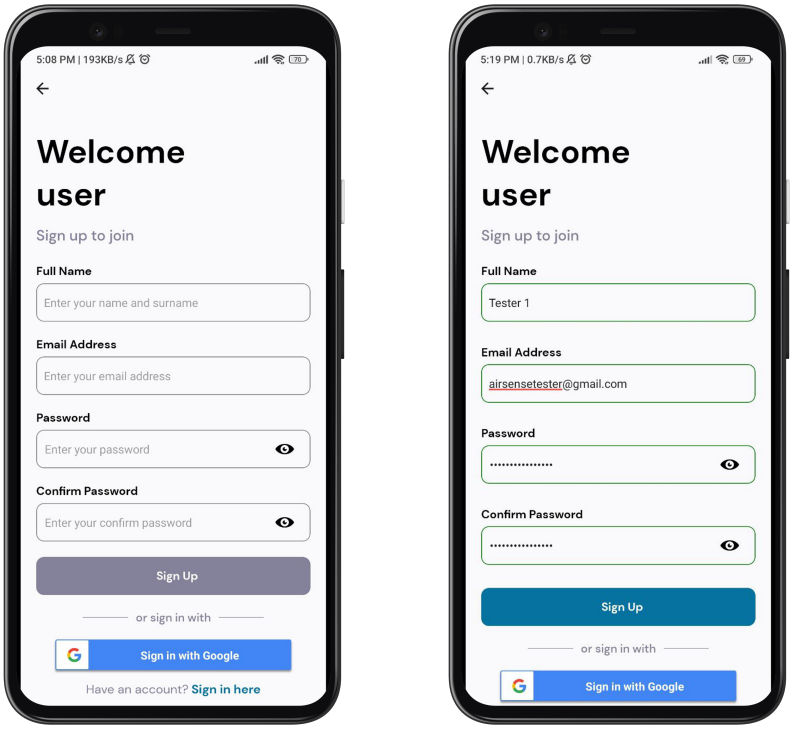

Figura 8.2: Schermata di registrazione vuota e compilata

8.1.2 Accesso in app

Il processo di accesso consente agli utenti di accedere all'applicazione utilizzando le credenziali inserite durante la fase di registrazione. La schermata di accesso offre due opzioni: l'utente può effettuare l'accesso tramite l'account Google oppure inserendo l'email e la password associate al proprio account. Se l'utente sceglie l'opzione di accesso tramite l'account Google, verrà visualizzata una lista degli account Google collegati al dispositivo dell'utente, consentendo di selezionarne uno per completare il processo di accesso. Altrimenti, se l'utente decide di effettuare l'accesso tramite email e password, dovrà compilare i campi del *form* inserendo le credenziali richieste. Una volta inserite correttamente le credenziali nel *form* di accesso, viene inoltrata la richiesta di accesso al servizio Firebase. Il sistema verifica la corrispondenza delle credenziali nel database e, se corrette, permette all'utente di accedere alla schermata principale dell'applicazione, come mostrato nella Figura 8.5. Tuttavia, se le credenziali fornite non sono corrette o l'account associato non è stato ancora verificato, verrà visualizzato un messaggio di errore, notificando all'utente che l'accesso non è possibile.

Inoltre, le informazioni utente restituite da Firebase vengono salvate in un contesto React. Questo contesto permette di condividere dati a livello globale fra tutti i componenti, semplificando la gestione dello stato dell'applicazione [15]. Le informazioni condivise includono l'identificativo univoco dell'utente, il nome completo, l'indirizzo email, l'*URL* della foto profilo e altre informazioni pertinenti, come mostrato nella Tabella 8.1.

Campo	Descrizione	Esempio
UUID	Identificativo univoco dell'utente	"lRptng17fDQdfnsn9ODQxya2de33"
fullName	Nome e cognome	"Tester1"
email	Indirizzo email	"airsensetester@gmail.com"
phoneNumber	Numero di telefono	"+39 123 456 789"
photoURL	URL della foto profilo	"https://example.com/profile.jpg"
createdAt	Data e ora di creazione dell'account	"2024-03-15T22:32:11.455Z"
lastLoginAt	Data e ora dell'ultimo accesso dell'utente	"2024-03-15T22:34:18.201Z'
providerId	ID del provider dell'account	"firebase"

Tabella 8.1: Struttura delle informazioni utente restituite da Firebase

La Figura 8.3 mostra la schermata di accesso sia vuota che compilata con i dati necessari.

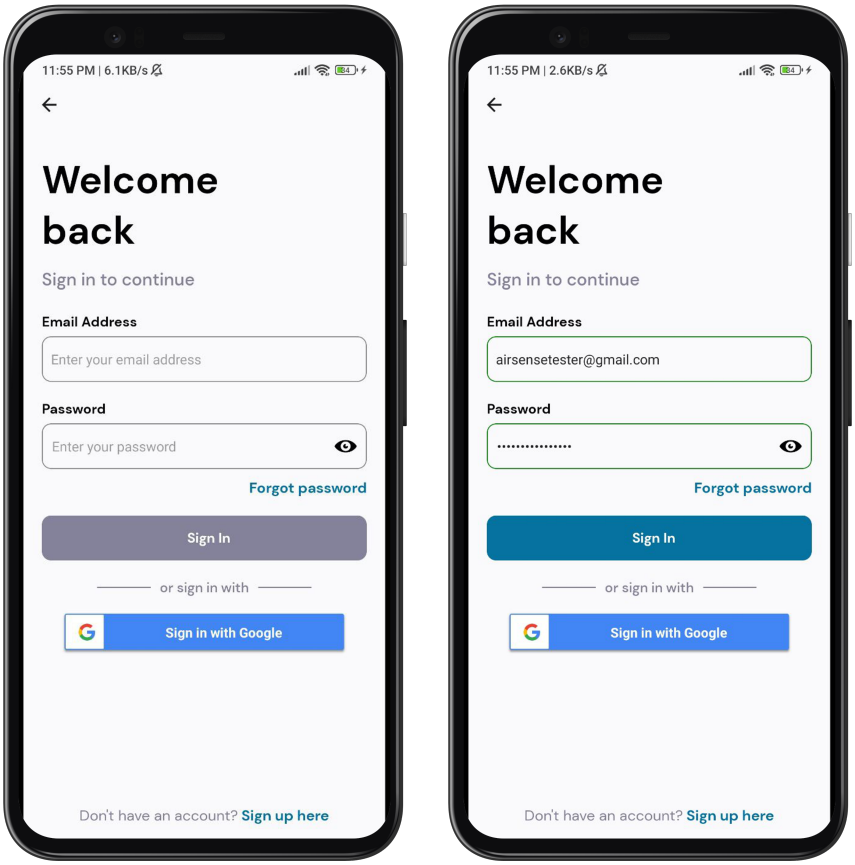

Figura 8.3: Schermata di accesso vuota e compilata

Oltre alla possibilità di effettuare l'accesso all'applicazione, l'utente ha la possibilità di recuperare la password in caso sia stata dimenticata. Selezionando l'opzione *Forgot password*, presente nella schermata di accesso, l'utente viene reindirizzato in una nuova schermata dove può inserire il proprio indirizzo email associato all'account. Successivamente, l'utente riceverà un'email con le istruzioni per il recupero della password. La Figura 8.4 mostra il processo per il recupero della password.

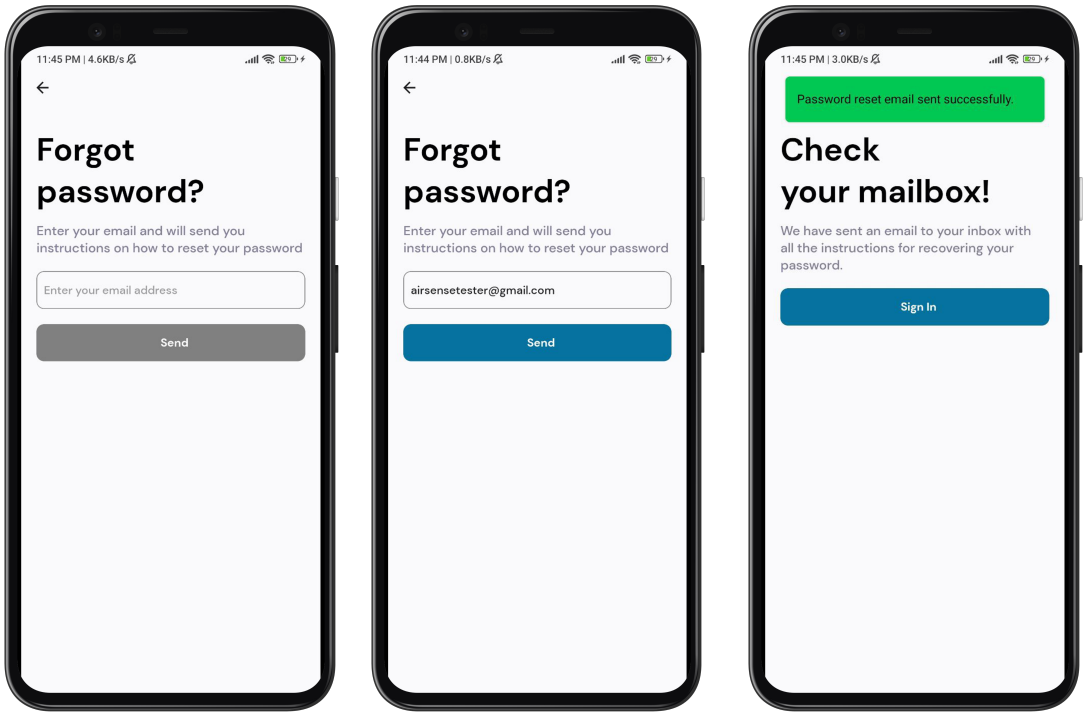

Figura 8.4: Schermate per il recupero della password

8.2 Mappe

La sezione delle mappe costituisce l'elemento centrale dell'applicazione, rappresentando la schermata principale a cui gli utenti accedono dopo aver effettuato l'accesso. Una volta giunti a questa schermata, gli utenti possono scegliere uno dei tre servizi offerti, tramite il *menu* laterale. Questi servizi includono la ricerca del percorso meno inquinato basandosi sui livelli di PM10, il recupero dei valori di PM10 per un'area specifica in un determinato giorno e ora, e la visualizzazione di una *heatmap* dei livelli di PM10 per un giorno e un'area specifici.

Per implementare questa sezione, si è fatto uso della libreria `react-native-maps` insieme ai suoi componenti `MapView`, `Polyline`, `Marker` e `Heatmap`, come descritto nella sezione 4.3.1.

Al fine di offrire un'esperienza interattiva agli utenti, è stato sviluppato un sistema per gestire i permessi di geolocalizzazione. Questo sistema consente all'applicazione di accedere alla posizione corrente dell'utente solo se quest'ultimo ha acconsentito a tale accesso, permettendo così di centrare automaticamente

la mappa sulla sua posizione. Per gestire i permessi di geolocalizzazione e ottenere la posizione corrente, è stata utilizzata la libreria `expo-location`[1] fornita da Expo, la quale semplifica l'accesso alle informazioni di geolocalizzazione e il monitoraggio della posizione corrente. La richiesta dei permessi di geolocalizzazione avviene tramite la funzione `requestForegroundPermissionsAsync`, e una volta ottenuti i permessi, la posizione corrente viene recuperata mediante la funzione `getCurrentPositionAsync`. Nel caso in cui l'utente non conceda i permessi di geolocalizzazione, è stata definita una posizione predefinita situata all'interno della città di Torino. Per condividere queste informazioni tra le varie mappe dell'applicazione, è stato creato un contesto React dedicato. Il Codice 8.1 mostra un esempio di funzione per ottenere la posizione corrente.

```
1  import * as Location from 'expo-location';
2
3  const getLocation = async () => {
4      try {
5          let {status} = await Location.
       requestForegroundPermissionsAsync();
6
7          if (status !== "granted") {
8              setLocationPermission(false);
9              setLocationError("Permesso di accesso alla posizione
       negato");
10             return;
11         }
12
13         let currentLocation = await Location.
       getCurrentPositionAsync({});
14
15         setLocationPermission(true);
16         setLocation(currentLocation);
17
18     } catch (error) {
19         console.error("Errore durante il recupero della posizione:
       ", error);
20     }
21 };
```

Codice 8.1: Funzione per il recupero della posizione corrente

[1]Per ulteriori approfondimenti si veda https://docs.expo.dev/versions/latest/sdk/location/

8.2.1 Percorso meno inquinato

Il servizio di individuazione del percorso meno inquinato è stato progettato per fornire agli utenti uno strumento pratico e utile per pianificare i loro spostamenti all'interno della città di Torino, considerando i livelli di inquinamento atmosferico lungo il tragitto. Per poter usufruire di questo servizio, gli utenti devono inserire gli indirizzi di partenza e di arrivo nel formato `Via Roma 123 City`. Dopo aver avviato il calcolo del percorso meno inquinato, vengono determinate le coordinate geografiche corrispondenti agli indirizzi inseriti attraverso le API gratuite messe a disposizione dal servizio Nominatim[2] inoltrando una richiesta `GET` all'indirizzo `https://nominatim.openstreetmap.org/search?format=json&q={encodeURIComponent(address)}` per ottenere le coordinate desiderate. Una volta ottenute sia le coordinate di partenza che quelle di arrivo, l'applicazione inoltra una richiesta API al microservizio `bike-monitoring-trip` tramite l'*endpoint* `GET v1/best-route?startLatitude={startLat}&startLongitude={startLon}&endLatitude={endLat}&endLongitude={endLon}`, fornendo come parametri la `latitudine` e la `longitudine` delle coordinate di partenza e di arrivo. Questo servizio restituisce le informazioni relative al percorso meno inquinato. Il funzionamento di questo servizio è descritto dettagliatamente in 7.4.

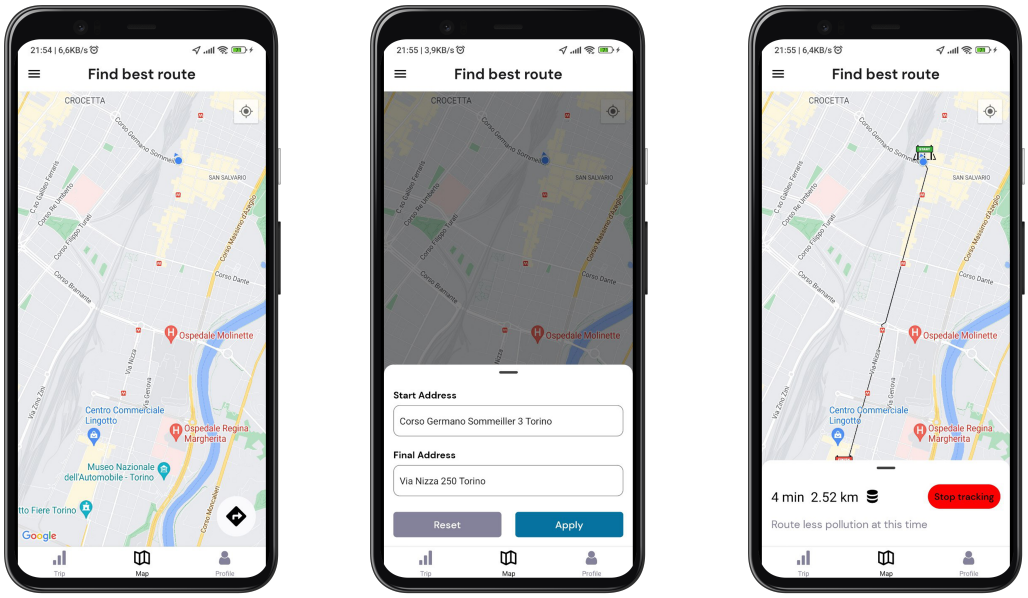

Figura 8.5: Schermate del servizio per il calcolo del percorso meno inquinato

[2]Nominatim è un servizio di geocodifica e ricerca di OpenStreetMap, che consente di trovare le coordinate geografiche di un indirizzo e viceversa.

8.2.2 Visualizzazione puntale sull'inquinamento

Il servizio di recupero dei valori mediani di PM10 consente agli utenti di ottenere dati dettagliati sui valori di PM10 in una specifica area, giorno e ora. Questo servizio è stato implementato per fornire agli utenti informazioni cruciali per valutare la qualità dell'aria in determinate zone e momenti.

Per poter usufruire di questo servizio, gli utenti devono fornire un indirizzo, un eventuale range di ricerca, una data e un'ora di interesse. L'indirizzo può essere fornito in due modalità: tramite l'inserimento manualmente, oppure tramite la selezione di un punto sulla mappa che fungerà da centro per una ricerca circoscritta. Questa doppia modalità di selezione è stata implementata per garantire una maggiore flessibilità agli utenti. Una volta inseriti i dati richiesti, l'applicazione inoltra una richiesta `GET` al microservizio `bike-monitoring-pollution` tramite l'*endpoint* `v1/pollution?latitude={lat}&longitude={lon}&range={range}&timestamp={timestamp}`, fornendo come parametri la `latitudine`, la `longitudine`, il `range` di ricerca e il `timestamp`. Se l'utente ha inserito manualmente l'indirizzo, viene invocata l'API offerta dal servizio Nominatim per convertirlo in coordinate. In questo caso, il valore del range sarà pari a 100 metri. Questo servizio recupera le informazioni sui valori di PM10 nell'area specificata e restituisce i dati nel formato descritto nella Tabella 7.10. Il comportamento di questo servizio è descritto in dettaglio in 7.5.1. Una volta ottenuti i dati, vengono visualizzati sulla mappa attraverso l'uso del componente `Marker`, come mostrato in Figura 8.6, il cui colore varia in base al valore mediano di PM10 ottenuto dalla richiesta. La Tabella 8.2 mostra i criteri di selezione dei colori in base al valori di PM2.5 e PM10.

PM2.5	PM10	Colore	Descrizione
< 15	< 20	Blue	La qualità dell'aria è molto buona e non comporta rischi per la salute
< 30	< 40	Verde	La qualità dell'aria è soddisfacente e comporta rischi significativi per la salute
< 45	< 60	Giallo	La qualità dell'aria è accettabile, ma per le persone a rischio potrebbero esserci impatti sulla salute
< 60	< 80	Arancione	La qualità dell'aria è moderata e potrebbero esserci impatti sulla salute di alcune persone
> 60	> 80	Rosso acceso	La qualità dell'aria è scarsa e ci sono rischi significativi per la salute di tutti

Tabella 8.2: Tabella dei criteri di selezione dei colori per i valori di PM2.5 e PM10

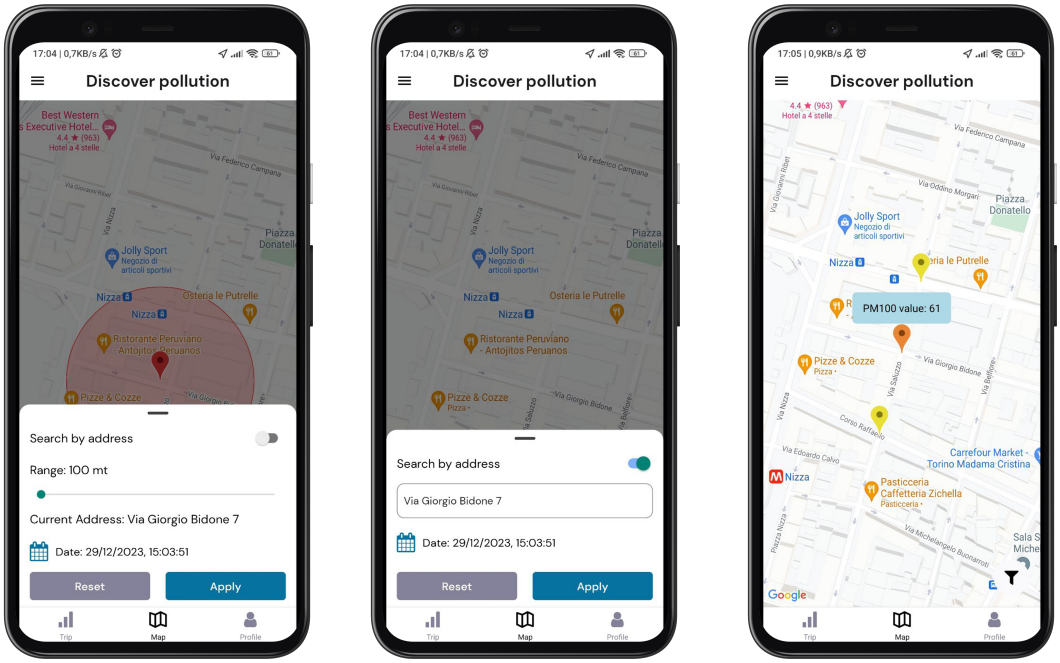

Figura 8.6: Schermate del servizio per il recupero dei valori di PM10

8.2.3 *Heatmap*

Il servizio di generazione della *heatmap* fornisce agli utenti dati dettagliati sui valori mediani di PM10 in un'area e in una data specifica. Questo strumento svolge un ruolo cruciale nell'analisi della qualità dell'aria, consentendo di identificare chiaramente le zone più inquinate all'interno della città. L'*heatmap* rappresenta un prezioso strumento per le autorità locali e i cittadini, agevolando il monitoraggio e la gestione della qualità dell'aria urbana.

Per poter usufruire di questo servizio, gli utenti devono selezionare la data desiderata su cui effettuare l'analisi. A ogni spostamento della mappa, viene effettuata una richiesta `GET` al microservizio `bike-monitoring-pollution` all'*endpoint* `v1/pollution/heatmap?latitude={lat}&longitude={long}&date={date}`, fornendo come parametri la `latitudine`, la `longitudine` e la `data` nel formato `YYYYMMDD`. Questo servizio recupera le informazioni, riguardanti i valori mediani di PM10 nella data specificata. Il comportamento di questo servizio è descritto in dettaglio in 7.5.2. Dopo aver acquisito i dati, viene associato un peso a ciascun valore mediano di PM10 in base alla sua concentrazione, al fine di visualizzare graficamente l'intensità dell'inquinamento. Questa operazione è gestita dalla funzione `getWeightForPm10Function`, mostrata nel Codice 8.2. Il peso assegnato determina

la tonalità del colore utilizzata nella *heatmap* generata dal componente `Heatmap`. La Figura 8.7 mostra una possibile visualizzazione a schermo del risultato.

```
1 function getWeightForPm10(pm10) {
2     if (!pm10) return 0;
3
4     return Math.min(Math.floor(pm10 / 10) + 1, 10);
5 }
```

Codice 8.2: Funzione per ottenere il peso di un punto all'interno della *heatmap*

La Figura 8.7 mostra una possibile visualizzazione a schermo del risultato.

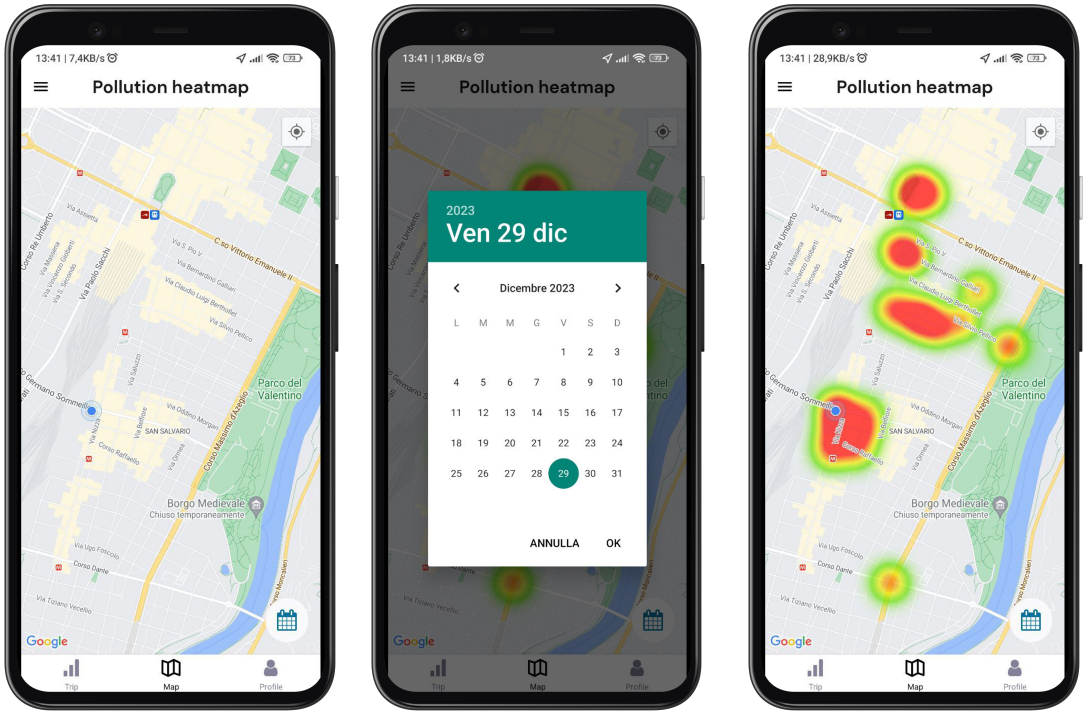

Figura 8.7: Schermate del servizio per la visualizzazione della *heatmap*

8.3 Percorsi passati dell'utente

La sezione dedicata ai percorsi offre agli utenti la possibilità di visualizzare tutti i percorsi effettuati in passato, di applicare filtri su di essi e di accedere a informazioni dettagliate riguardo a un percorso specifico.

8.3.1 Lista dei percorsi

La sezione `Trip` consente agli utenti di visualizzare tutti i percorsi effettuati in passato. Quando si accede a questa sezione, viene effettuata una richiesta verso il microservizio `bike-monitoring-trip`, all'*endpoint* `GET /v1/user/{userId}/trips`, il quale recupera e ordina tutti i percorsi effettuati dall'utente in base alla data, restituendoli dal più recente al meno recente. Il comportamento di questo servizio è descritto in dettaglio in 7.3.1. Se l'utente ha effettuato almeno un percorso in passato, la schermata mostrerà l'elenco di tutti i percorsi passati. La Figura 8.8 mostra la schermata dei percorsi dell'utente.

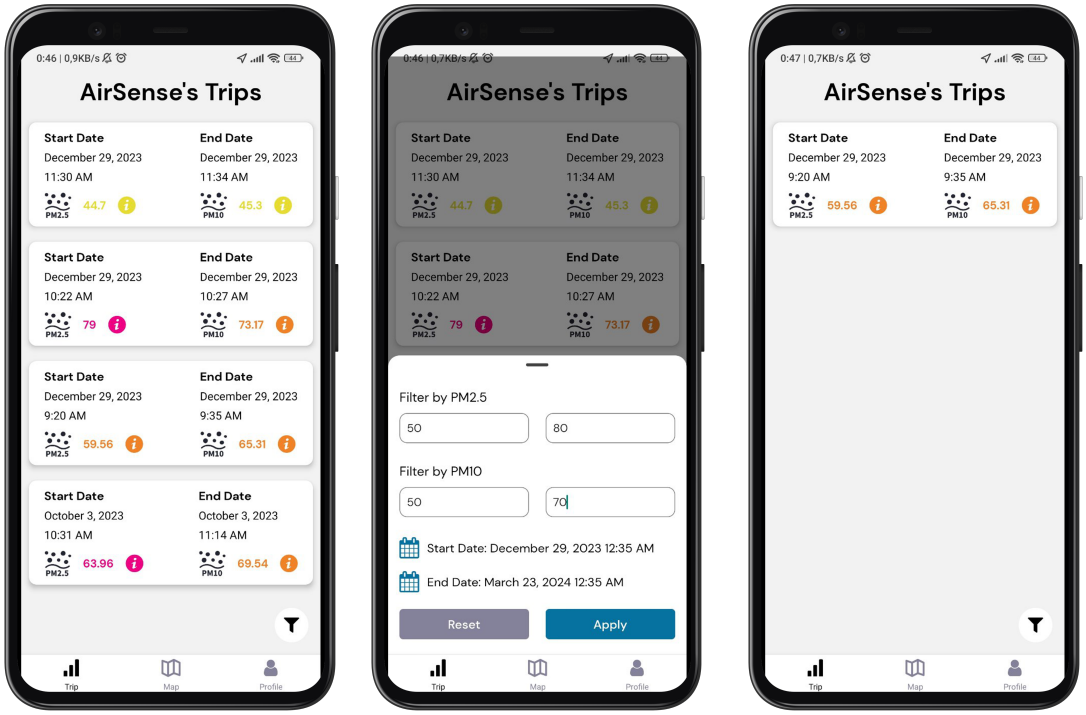

Figura 8.8: Schermate dei percorsi passati dell'utente

Inoltre, per fornire all'utente un'immediata comprensione dei livelli della qualità dell'aria per ciascun percorso, vengono presentate brevi descrizioni associate ai valori medi di PM2.5 e PM10, come mostrato nella Tabella 8.2.

Infine, gli utenti hanno la possibilità di filtrare i percorsi passati in base ai parametri mostrati nella Tabella 8.3.

Campo	Descrizione	Esempio
MinPM2.5	Valore minimo di PM2.5	10.2
MaxPm2.5	Valore massimo di PM2.5	30.8
MinPM10	Valore minimo di PM10	20
MaxPm10	Valore massimo di PM10	60
startDate	Data e ora di inizio del percorso	2024-03-14T22:00:42.411Z
endDate	Data e ora di fine del percorso	2024-03-12T22:00:42.411Z

Tabella 8.3: Parametri di filtraggio per i percorsi passati

Questi parametri consentono agli utenti di personalizzare la ricerca, facilitando la visualizzazione dei percorsi che soddisfino specifici criteri in termini di qualità dell'aria e periodo temporale.

8.3.2 Visualizzazione di un percorso specifico

La schermata dedicata a un percorso specifico offre agli utenti la possibilità di visualizzare informazioni dettagliate relative a un percorso. Al momento dell'accesso a questa schermata, viene effettuata una richiesta verso il microservizio `bike-monitoring-trips`, all'*endpoint* `GET /v1/user/{userId}/trips/{tripId}`. Questo servizio recupera tutte le informazioni riguardanti il percorso specifico richiesto. Il comportamento di questo servizio è descritto in dettaglio in 7.3.2. Una volta ricevuti i dati, vengono mostrati nella schermata, la quale è suddivisa in due sezioni principali: la prima sezione riguarda la mappa, mentre la seconda mostra una legenda dei valori della qualità dell'aria e altre informazioni relative al percorso. Ogni segmento del percorso è evidenziato con un colore che riflette il livello di qualità dell'aria, seguendo le stesse linee guida descritte nella Tabella 8.2. Inoltre, viene fatto anche uso del componente `Marker` per segnalare punti di interesse lungo il percorso, come il punto di partenza e di arrivo. Grazie all'algoritmo per la creazione della mappa della città descritto nel capitolo 6 e al microservizio di normalizzazione delle coordinate descritto in 7.1, è stato risolto il problema di rappresentazione dei punti sulla mappa, ottenendo dei punti normalizzati che riflettono fedelmente gli incroci delle strade, come mostrato nella Figura 8.9. Nella seconda sezione vengono mostrate le informazioni del percorso. Per ottenere gli indirizzi di partenza e di arrivo, è stato fatto uso delle API messe a disposizione dal servizio Nominatim. Per ottenere tali informazioni, verrà effettuata una richiesta all'indirizzo `https://nominatim.openstreetmap.org/reverse?format=json&lat={latitude}&lon={longitude}`.

Inoltre, tramite l'uso dell'API `Share`[3] fornita da React Native, l'utente può condividere i dettagli del percorso.

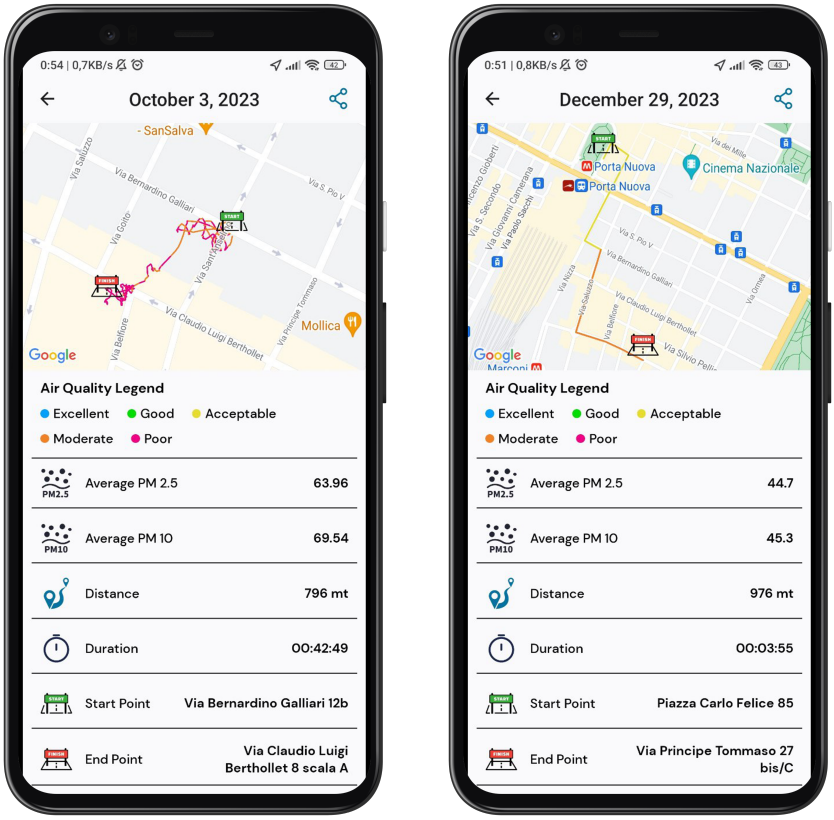

Figura 8.9: Schermata dei dettagli di un percorso prima e dopo il processo di normalizzazione delle coordinate

8.4 Profilo

La sezione `Profile` offre agli utenti la possibilità di visualizzare e modificare le informazioni più rilevanti relative al proprio profilo, nonché di gestire i dispositivi Arduino associati.

In seguito all'accesso a questa schermata, vengono eseguite due operazioni principali. Inizialmente, viene inoltrata una richiesta al microservizio `bike-monitoring-profile` all'*endpoint* `GET v1/profile/devices?userUUID={userUUID}&limit=`

[3]Per maggiori approfondimenti si veda https://reactnative.dev/docs/share?language=javascript.

`{limit}`, per recuperare gli ultimi dispositivi associati all'utente. Qui, il parametro `limit` viene impostato a 4 per visualizzare solo i quattro dispositivi più recenti associati. Il comportamento di questo servizio è descritto in dettaglio in 7.2.1. Inoltre, utilizzando le informazioni precedentemente salvate nel contesto dopo l'accesso, vengono visualizzate le informazioni più rilevanti come nome, email e immagine del profilo dell'utente. La Figura 8.10 mostra la schermata del profilo.

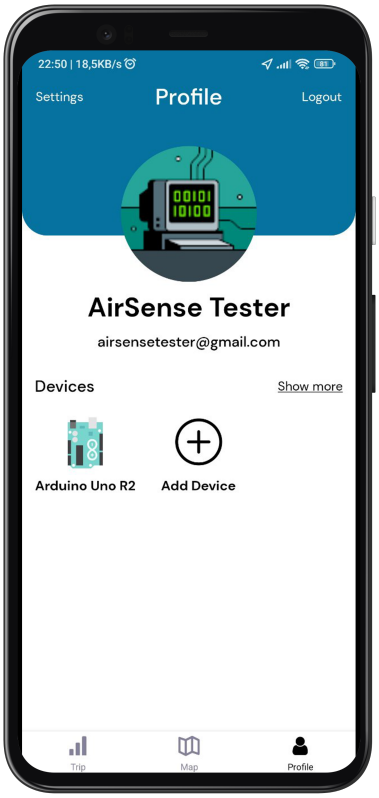

Figura 8.10: Schermata della profilo

8.4.1 Associazione del dispositivo all'utente

Il processo di associazione dispositivo-utente è suddiviso in tre fasi: l'invio della richiesta da parte dell'applicazione, la gestione della richiesta sul dispositivo Arduino e il salvataggio dei dati nel database. In questa sezione, ci concentreremo sulla prima fase, mentre la seconda e la terza sono state rispettivamente esaminate in 5.2 e in 7.2.1.

Tale processo, inizia in seguito alla pressione del pulsante *Add Device* da parte dell'utente. Viene inoltrata una richiesta HTTP al dispositivo Arduino, come

spiegato in 5.2, il quale se risulta essere raggiungibile, restituisce l'identificativo del dispositivo Arduino contattato. Successivamente, come descritto in 7.2.1, il nuovo dispositivo viene aggiunto al database e la lista dei dispositivi associati viene aggiornata.

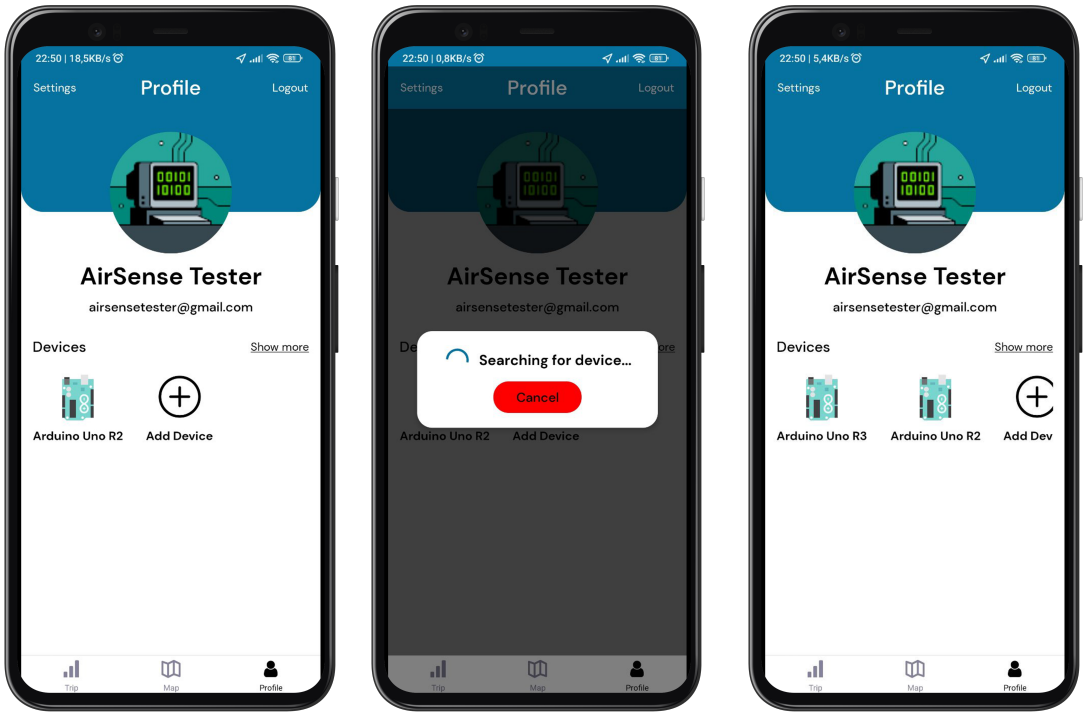

Figura 8.11: Schermate per l'associazione di un dispositivo a un utente

8.4.2 Lista dei dispositivi

Questa funzionalità consente agli utenti di visualizzare tutti i dispositivi associati al proprio account. In seguito all'accesso a questa schermata, viene inoltrata una richiesta al microservizio `bike-monitoring-profile`, all'*endpoint* `GET v1/profile/devices?userUUID={userUUID}&limit={limit}`, senza specificare alcun limite per consentire il recupero di tutti i dispositivi associati all'utente. Una volta ricevuti i dati, vengono mostrati nella schermata come mostrato in Figura 8.12.

Inoltre, gli utenti possono rimuovere un dispositivo presente nella lista tramite l'apposita icona. In seguito alla conferma di tale operazione, viene inoltrata una richiesta all'*endpoint* `DELETE v1/profile/delete-device?deviceUUID={deviceUUID}&userUUID={userUUID}`, rimuovendo dalla lista il dispositivo e disaccoppiandolo.

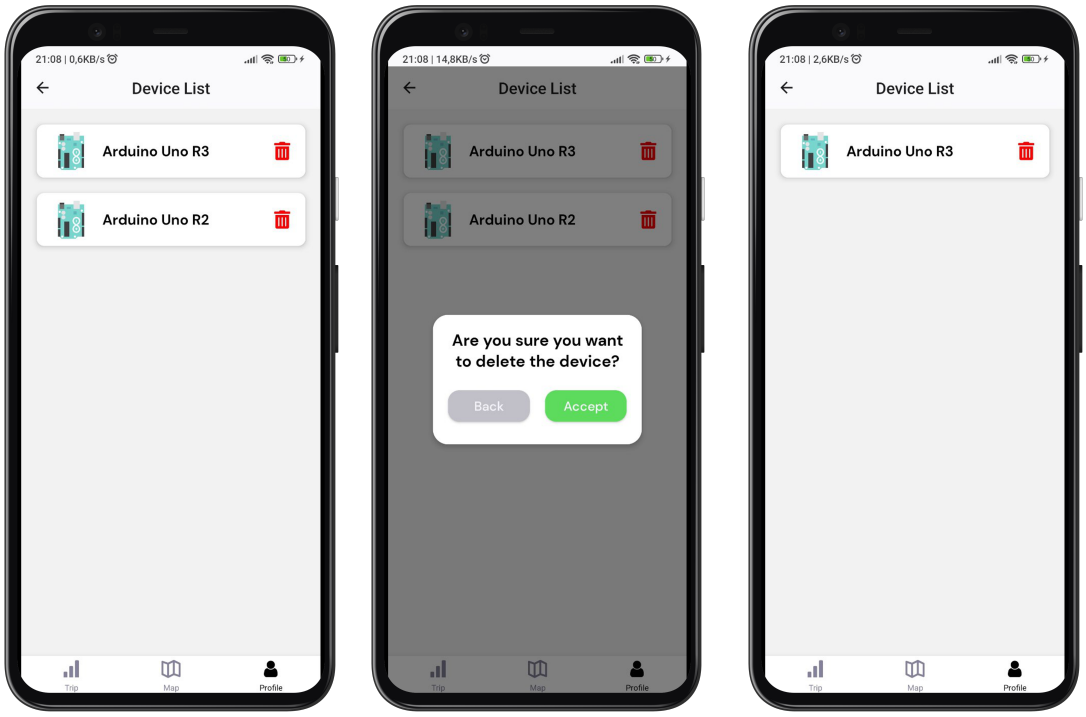

Figura 8.12: Schermate della lista dei dispositivi associati all'utente prima e dopo la rimozione

8.4.3 Impostazioni utente

In questa schermata, gli utenti possono modificare nome, cognome, email e la password tramite l'uso delle API offerte dalla libreria `react-native-firebase/auth`. Come per la fase di registrazione e accesso, si è fatto uso delle librerie `react-hook-form` e `yup` per poter utilizzare i form e validarne i campi. In particolare, per la modifica dell'email, è stata utilizzata la funzione `verifyBeforeUpdateEmail`, che invia all'indirizzo email dell'utente una email contenente le istruzioni per aggiornare l'indirizzo email. In seguito a una modifica, gli utenti verranno reindirizzati alla schermata del profilo. La Figura 8.13 mostra le schermate per le impostazioni utente.

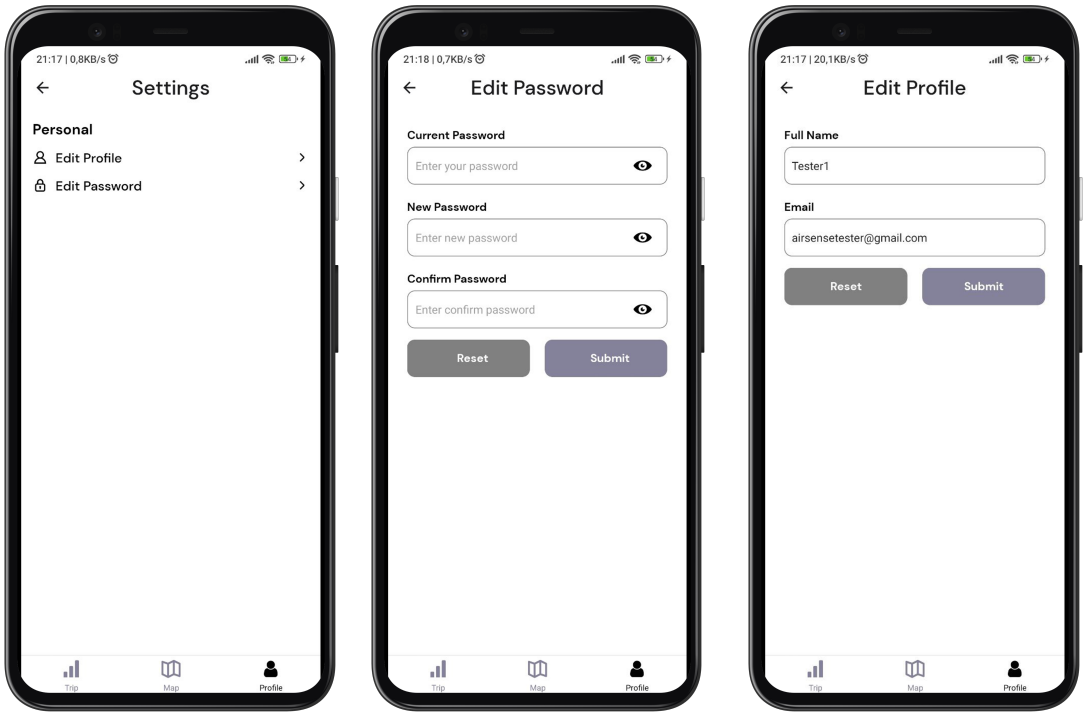

Figura 8.13: Schermate per le impostazioni utente

Capitolo 9

Conclusioni

L'inquinamento atmosferico rappresenta una sfida significativa per le società moderne, con impatti significativi sulla salute umana e sull'ambiente. La presente tesi ha affrontato questa sfida, concentrandosi sull'implementazione di un'applicazione *mobile* per monitorare la qualità dell'aria nelle città. L'obiettivo principale è stato quello di fornire agli utenti uno strumento pratico e intuitivo per monitorare e comprendere l'impatto dell'inquinamento atmosferico sulla loro salute e sul loro ambiente circostante, con un *focus* specifico sulla città di Torino.

Durante il corso di questo lavoro, sono stati raggiunti diversi obiettivi significativi. In primo luogo, l'implementazione dell'applicazione *mobile* ha consentito agli utenti di visualizzare i dati di inquinamento atmosferico raccolti dai dispositivi Arduino distribuiti in diverse aree della città. La possibilità di calcolare il percorso meno inquinato, insieme alla *heatmap* che evidenzia le aree più critiche, ha consentito agli utenti di prendere decisioni informate riguardo ai loro spostamenti e alle attività all'aperto, contribuendo così a migliorare la loro qualità della vita e a evitare le zone più inquinate, riducendo l'esposizione agli agenti inquinanti.

L'associazione del dispositivo di raccolta dati al profilo utente ha consentito agli utenti di tenere traccia dei propri percorsi e di analizzare i dati raccolti in modo più approfondito.

Inoltre, questa ricerca ha portato allo sviluppo di un algoritmo per la creazione di mappe cittadine, ampliando la copertura geografica dei dati raccolti a tutta la città di Torino e garantendo una migliore precisione nella rappresentazione delle informazioni geografiche e nell'associazione dei dati di inquinamento atmosferico a specifici punti della città.

Sono stati inoltre implementati diversi servizi all'interno dell'applicazione, come la gestione dei dispositivi Arduino associati agli utenti, il recupero dei percorsi passati, la ricerca del percorso meno inquinato e il recupero dei valori mediani di PM10 per la visualizzazione di una *heatmap*.

Il raggiungimento di tali obiettivi consente agli utenti di migliorare il loro stile di vita e di acquisire una maggiore consapevolezza riguardo alle decisioni da prendere in relazione ai livelli di inquinamento atmosferico. L'accesso diretto ai dati sull'inquinamento, insieme agli strumenti per interpretarli, offre agli utenti la possibilità di adottare comportamenti più informati e protettivi nei confronti della propria salute e dell'ambiente. La capacità di ottenere percorsi meno inquinati e di evitare le aree critiche contribuisce a ridurre l'esposizione agli agenti inquinanti, migliorando così la qualità della vita e promuovendo abitudini più sostenibili.

9.1 Sviluppi futuri

Nel contesto degli sviluppi futuri dell'applicazione, sono stati individuati diversi ambiti di miglioramento che potrebbero contribuire significativamente a potenziarne le funzionalità e l'utilità per gli utenti:

- **Miglioramento del modello predittivo**: Un miglioramento significativo per lo sviluppo futuro dell'applicazione riguarda l'implementazione di un modello predittivo per stimare i livelli futuri di PM10, al fine di integrarlo con i servizi offerti. Attualmente, tali servizi si basano su dati storici di PM10 presenti nel database, ma l'integrazione di un modello di previsione `ARIMA`, già sviluppato nella tesi precedente, potrebbe consentire previsioni future più accurate e dettagliate dei valori di PM10. Trasformando l'attuale modello in un servizio API, si potrebbe ampliare l'utilizzo dei servizi offerti dall'applicazione, consentendo agli utenti di accedere a informazioni predittive e prendere decisioni più informate riguardo ai propri spostamenti e attività all'aperto.
- **Integrazione di ulteriori sensori nel dispositivo Arduino**: Un'altra area di miglioramento riguarda l'aggiunta di ulteriori sensori al dispositivo Arduino utilizzato per la raccolta dati. L'aumento del numero di sensori consentirebbe di acquisire più informazioni utili non solo per l'utente finale, ma anche per migliorare l'efficacia e la precisione del servizio di calcolo del percorso meno inquinato. Ad esempio, l'aggiunta di sensori per monitorare la temperatura e l'umidità potrebbe fornire una valutazione più completa delle condizioni ambientali, mentre un accelerometro potrebbe rilevare eventuali imperfezioni stradali.
- **Gamification dell'applicazione**: Una caratteristica interessante potrebbe essere l'introduzione di elementi di *gamification* nell'applicazione. Ad esempio, potrebbe essere introdotto un sistema di badge o premi per incentivare comportamenti sostenibili, come il raggiungimento di determinati obiettivi legati al numero di chilometri percorsi o alla segnalazione di punti di interesse.

- **Segnalazione di punti di interesse**: Per coinvolgere gli utenti e migliorare le informazioni fornite dall'applicazione, potrebbe essere utile introdurre un sistema di segnalazione di punti di interesse. Questi punti potrebbero essere elementi non presenti sulla mappa standard, come fontane, panchine o buche sulla strada. Le segnalazioni potrebbero essere inviate direttamente dall'applicazione tramite foto, e potrebbe essere integrato un modello di computer *vision* per verificare automaticamente la validità delle segnalazioni.

- **Notifiche meteo**: Implementare un sistema di notifiche meteorologiche potrebbe ulteriormente migliorare l'utilità dell'applicazione per gli utenti. Consentire agli utenti di pianificare i loro viaggi futuri e ricevere notifiche meteorologiche in tempo reale potrebbe aiutarli a prendere decisioni informate. Ad esempio, l'applicazione potrebbe avvisare gli utenti tramite notifiche *push* se il meteo nella zona del loro viaggio è avverso, suggerendo percorsi alternativi o la possibilità di riprogrammare il viaggio.

Dal punto di vista personale, questa ricerca ha suscitato una profonda consapevolezza sull'importanza cruciale della qualità dell'aria per la salute umana e l'ambiente. Benché avessi conoscenza del serio problema dell'inquinamento, è stato solo dopo aver analizzato e visualizzato i dati relativi all'inquinamento nella città di Torino che ho potuto realmente constatare la gravità della situazione. Questo momento di consapevolezza ha avuto un impatto significativo su di me, spingendomi a riflettere sul mio ruolo e sulla responsabilità individuale nell'affrontare questa sfida globale. Mi ha motivato ad adottare comportamenti più sostenibili e consapevoli, riconoscendo l'importanza di contribuire attivamente alla protezione dell'ambiente e alla salvaguardia della salute pubblica. Inoltre, sono convinto che il progresso nel campo del monitoraggio dell'inquinamento atmosferico e della tecnologia possa portare a miglioramenti significativi nella qualità della vita nelle città e nella salute pubblica. Tuttavia, è necessario un impegno continuo e un investimento significativo da parte delle istituzioni, delle imprese e della società civile per tradurre questi progressi in azioni concrete e risultati tangibili.

In conclusione, questa tesi rappresenta un passo significativo verso la creazione di soluzioni innovative per affrontare la sfida dell'inquinamento atmosferico nelle città. Tuttavia, ci sono ancora molte opportunità di ricerca e sviluppo per migliorare ulteriormente l'applicazione e fornire agli utenti strumenti sempre più efficaci per monitorare e mitigare l'impatto dell'inquinamento atmosferico sulla loro vita quotidiana.

Bibliografia

[1] Il giornale dell'ambiente. *Inquinamento atmosferico in Italia e nel mondo*. Feb. 2024. URL: https://ilgiornaledellambiente.it/inquinamento-ambientale-inquinanti/inquinamento-atmosferico-mondiale-italia/ (cit. a p. 5).

[2] Filair. *Cosa sono le polveri sottili?* Lug. 2021. URL: https://www.filair.it/le-polveri-sottili-cosa-sono-rimedi/ (cit. a p. 5).

[3] Donatella Barus. *Tumori e ictus fra gli effetti dell'inquinamento atmosferico*. Gen. 2019. URL: https://www.fondazioneveronesi.it/magazine/articoli/oncologia/tumori-e-ictus-fra-gli-effetti-dellinquinamento-atmosferico (cit. a p. 6).

[4] Luca Tremolada. «Polveri sottili e inquinamento: perché l'Oms ha inasprito le linee guide sulla qualità dell'aria?» In: *Il Sole 24 Ore* (2021). URL: https://www.infodata.ilsole24ore.com/2021/09/28/perche-loms-inasprito-le-linee-guide-sulla-qualita-dellaria/ (cit. a p. 6).

[5] Michael Montenegro. *2023 IQAir World Air Quality Report*. Mar. 2024. URL: https://www.iqair.com/sa/newsroom/waqr-2023-pr (cit. a p. 6).

[6] Legambiente. *Mal'Aria di città 2024: i nuovi dati sull'inquinamento atmosferico nelle città italiane*. Feb. 2024. URL: https://www.legambiente.it/comunicati-stampa/i-dati-di-malaria-di-citta-2024/ (cit. a p. 7).

[7] ARPA Piemonte. *Prime valutazioni sulla qualità dell'aria nel 2023: trend in miglioramento*. Gen. 2024. URL: https://www.arpa.piemonte.it/news/prime-valutazioni-sulla-qualita-dellaria-nel-2023-trend-in-miglioramento (cit. a p. 9).

[8] Cristian Cingolani. «Un framework per l'analisi della qualità dell'aria e per l'individuazione del percorso meno inquinato in una grande città». Tesi di laurea mag. Univesità Politecnica delle Marche, lug. 2022. URL: https://hdl.handle.net/20.500.12075/13678 (cit. a p. 11).

[9] MongoDB. *Cos'è MongoDB?* 2024. URL: https://www.mongodb.com/it-it/what-is-mongodb (cit. a p. 17).

[10] Jamal Osman. *Che cos'è Firebase?* Feb. 2023. URL: https://appmaster.io/it/blog/cose-firebase (cit. a p. 19).

[11] Luigi Marino. *App native, ibride, web app e PWA: tanti tipi di applicazioni per ogni necessità*. Lug. 2022. URL: https://www.marinoluigi.it/app-native-ibride-web-app-e-pwa-tanti-tipi-di-applicazioni-per-ogni-necessita/ (cit. a p. 20).

[12] Michael Szczepanik. *It's time for the NATIVE mobile development to end*. Feb. 2024. URL: https://medium.com/mobilepeople/its-time-for-the-native-mobile-development-to-end-f677e37be5a8 (cit. a p. 22).

[13] Nandhu_writes. *Expo vs Vanilla React Native*. Ott. 2023. URL: https://blog.stackademic.com/expo-vs-vanilla-react-native-765e6aff5b51 (cit. a p. 22).

[14] Internet Engineering Task Force (IETF). *The GeoJSON Format*. Ago. 2016. URL: https://datatracker.ietf.org/doc/html/rfc7946 (cit. a p. 32).

[15] Meta. *Passing Data Deeply with Context*. Ver. 18.2.0. Giu. 2022. URL: https://react.dev/learn/passing-data-deeply-with-context (cit. a p. 59).

Ringraziamenti

In conclusione di questo percorso accademico, desidero esprimere la mia più profonda gratitudine alle persone che hanno contribuito in modo significativo alla realizzazione di questa tesi e che mi hanno accompagnato nel corso di questi anni universitari.

Innanzitutto, desidero ringraziare il Prof. Luigi De Russis, il mio relatore, per i suoi preziosi consigli e per la sua disponibilità. Grazie per avermi fornito spunti fondamentali nella stesura di questo lavoro e per avermi guidato nei momenti di indecisione.

Un ringraziamento va all'azienda Blue Reply, in particolare al manager Andrea e ai miei colleghi Umberto, Matteo, Gianmarco e Cristian, per aver creato un ambiente di lavoro stimolante e per il loro supporto durante questo percorso. Le loro idee e il loro contributo hanno arricchito enormemente il mio lavoro di ricerca.

Desidero ringraziare di cuore la mia famiglia per il loro amore incondizionato, incoraggiamento costante e sostegno instancabile. Ai miei genitori, Teresa e Vittorio, che sono stati il mio faro nei momenti bui, per avermi sempre sostenuto e spronato a dare il meglio di me stesso. A mia sorella, Valentina, per essere stata al mio fianco in ogni fase della mia vita. E ai miei amati nonni, Concetta e Giacomo, per il loro affetto infinito e il loro costante sostegno morale.

Ai miei cari amici Andrea, Annalisa e Davide, che sono diventati una seconda famiglia negli ultimi due anni, desidero esprimere la mia profonda gratitudine. Grazie per essere stati al mio fianco in questo percorso, per aver condiviso con me gioie e dolori, e per avermi sostenuto in ogni occasione.

Un ringraziamento va anche ai miei amici d'infanzia, che sono stati parte integrante del mio percorso fin dall'inizio. Grazie per aver condiviso con me tanti momenti preziosi e per essere rimasti al mio fianco nel corso degli anni.

Infine, dedico questa tesi a me stesso. La dedico ai sacrifici che pensavo di non essere in grado di sostenere ed alla tenacia che mi ha fatto arrivare in fondo.

Grazie a tutti coloro che hanno reso possibile il completamento di questa tesi magistrale. Sono grato per il vostro sostegno, la vostra fiducia e la vostra ispirazione.