Alma Mater Studiorum
Università di Bologna
Campus di Cesena

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

PROGETTAZIONE E IMPLEMENTAZIONE DI UN SISTEMA DI ROUTE PLANNING CON INTEGRAZIONE DI DATI SULLA QUALITÀ DELL'ARIA

Relatore:
Prof.ssa Silvia Mirri

Presentata da:
Simone Del Gatto

Correlatore:
Dr. Giovanni Delnevo

Sessione III
Anno Accademico 2022-2023

Introduzione

Il progetto Ecosister vede la collaborazione tra l'Università di Bologna e diverse università italiane, e ha come scopo quello di supportare la transizione ecologica del sistema economico e sociale regionale attraverso un processo che coinvolga trasversalmente tutti i settori, le tecnologie e le competenze coniugando transizione digitale e sostenibilità con il lavoro e il benessere delle persone e la difesa dell'ambiente in coerenza con gli obiettivi del Patto per il Lavoro e per il Clima [1], ed integrandosi con programmazioni regionali, nazionali e europee.

Le attivià di ricerca e sviluppo si dividono in sei spoke:

1. Materiali per la sostenibilità e la transizione ecologica
2. Produzione, lo stoccaggio e il risparmio di energia pulita
3. Green manifacturing per un'economia sostenibile
4. **Smart solutions per la mobilità, gli alloggi e l'energia per una società a zero emissioni di carbonio**
5. Circular economy e la blue economy
6. High performance computing e data tecnology

Di particolare interesse per l'elaborato di tesi è lo spoke quattro, che a sua volta si articola in diversi workpackage di ricerca e sviluppo:

- WP1: **Sicurezza dei pedoni e dei ciclisti**, modellazione dei flussi di mobilità, sistemi multimodali e mobilità condivisa, **mobilità cibernetica**

- WP2: Healty and active city, **innovazioni tecnologiche e sociali**, valutazione delle politiche climatiche

- WP3: Riprogettazione urbana nature-based

- WP4: **Qualità dell'aria indoor/outdoor**, dispersione e controllo del PM10, urban heat island, decarbonizzazione urbana, comunità energetiche, NBS

Scopo del progetto di tesi è la progettazione e lo sviluppo di uno degli output previsti dallo spoke 4, un sistema di routes planning per il calcolo di percorsi brevi nell'area di Bologna che tenga conto di diversi fattori di costo oltre alla velocità di percorrenza, quali la qualità dell'aria e la frequenza di incidenti che avvengono sulle strade percorse.

Il sistema dovrà essere accessibile attraverso un'applicazione web responsive mobile-first, che permetta di visualizzare la zona di Bologna coinvolta nel progetto, e i dati sulla qualità dell'aria e sugli incidenti, con una visualizzazione adeguata rispetto allo stato dell'arte delle applicazioni attualmente presenti della stessa tipologia.

Al fine di promuovere aspetti di gamification e coinvolgimento degli utenti, il sistema dovrà prevedere aspetti di personalizzazione e memorizzazione delle informazioni sugli utenti, permettendo agli utilizzatori registrati di salvare e modificare informazioni sul proprio profilo.

Nell'elenco seguente è indicato l'obiettivo di ogni capitolo del volume di tesi.

- Nel primo capitolo sarà riportato l'obiettivo del progetto di tesi, e sarà descritto il background di conoscenze necessario per comprendere il lavoro svolto, in particolare lo studio effettuato sullo stato dell'arte dei

sistemi attuali simili all'elaborato da realizzare, e alcuni concetti fondamentali del dominio applicativo di interesse. Infine, saranno spiegate approfonditamente le funzionalità del sistema.

- Nel secondo capitolo verranno descritte le tecnologie che saranno poi impiegate nella realizzazione del progetto. Trattandosi di un'applicazione web, le tecnologie sono state differenziate in modo da distinguere quelle impiegate lato front-end da quelle impiegate lato back-end.

- Nel terzo capitolo saranno descritte nel dettaglio la progettazione e l'implementazione del sistema. Verrà descritta l'architettura generale, poi si scenderà nel dettaglio della descrizione di ogni singolo componente a partire dai servizi realizzati fino all'applicazione front-end.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Funzionalità del sistema e stato dell'arte

Questo primo capitolo ha due scopi: il primo è presentare l'elaborato di tesi, il suo scope e le sue funzionalità. Il secondo è riportare lo studio preliminare svolto prima di realizzare il progetto. Quindi in primo luogo sarà riportato l'obiettivo del progetto di tesi, in secondo luogo sarà descritto il background di conoscenze necessario per comprendere il lavoro di tesi e lo studio effettuato sullo stato dell'arte dei sistemi attuali simili all'elaborato da realizzare, infine saranno approfondite le funzionalità del sistema.

1.1 Scope del progetto

Scopo del progetto di tesi è la progettazione e lo sviluppo di un sistema di route planning per il calcolo di percorsi brevi nell'area di Bologna tenendo conto di diversi fattori di costo oltre alla velocità di percorrenza, quali la qualità dell'aria e la frequenza di incidenti che avvengono sulle strade percorse.

Il sistema dovrà essere accessibile attraverso un'applicazione web responsive mobile-first, che permetta di visualizzare la zona di Bologna coinvolta nel progetto, e i dati sulla qualità dell'aria e sugli incidenti, con una visualiz-

zazione adeguata rispetto allo stato dell'arte delle applicazioni attualmente presenti della stessa tipologia. [3]

Al fine di promuovere aspetti di gamification e coinvolgimento degli utenti, il sistema dovrà prevedere aspetti di personalizzazione e memorizzazione delle informazioni sugli utenti, permettendo agli utilizzatori registrati di salvare e modificare informazioni sul proprio profilo.

1.2 Sistemi di coordinate Geografiche

Il sistema da realizzare fa parte dei sistemi classificati come GIS, Geographic Information System, cioè tutti quei sistemi che hanno nel loro core-domain la manipolazione di dati relativi alla rappresentazione della Terra.

Per la presentazione del lavoro di tesi risulta utile conoscere alcune specifiche relative ai sistemi di coordinate geospaziali utilizzate in Geodesia, scienza che studia la forma della Terra e le sue dimensioni avvalendosi di misure astronomiche, gravimetriche, trigonometriche[4].

Infatti, esistono diversi sistemi di coordinate geografiche che fanno riferimento a diverse proiezioni della Terra sul piano cartesiano, le quali differiscono in base alle trasformazioni che vengono applicate per riportare la forma tridimensionale del geoide terrestre sul piano cartesiano.

L'EPSG è l'acronimo di "European Petroleum Survey Group", un gruppo europeo che si occupa di standardizzare i sistemi di coordinate geografiche e geodetiche utilizzate nei sistemi GIS (Geographic Information System) e nelle applicazioni geospaziali, classificandoli in specifiche chiamate "EPSG codes" o "EPSG identifiers". Ogni codice EPSG è associato a un sistema di coordinate specifico o una proiezione cartografica utilizzato per identificare in modo univoco un sistema di riferimento spaziale, utilizzabile per una certa zona della Terra [5].

Nei sistemi GIS il tipo di proiezioni viene scelto in base a quella che si adatta meglio alla zona di interesse dell'applicazione.

Si seguito si riportano i sistemi di coordinate utilizzate nello sviluppo dell'elaborato di tesi:

- EPSG 4326 (WGS 84): si tratta di una delle proiezioni più comuni e universalmente accettate. Questo sistema di coordinate utilizza il sistema di riferimento geodetico mondiale (WGS 84), che è ampiamente utilizzato nei sistemi di navigazione satellitare GPS (Global Positioning System). IL sistema rappresenta le coordinate in gradi decimali, dove la latitudine varia da -90° a +90°, lo zero si trovano sull'Equatore, e la longitudine varia da -180° a +180°, lo zero si trova sul meridiano di Greenwich

- EPSG 32632 è una proiezione UTM (Universal Transverse Mercator) basata su EPSG 4326. È una proiezione utilizzata per proiettare le coordinate geografiche in modo che possano essere rappresentate in un sistema di coordinate cartesiano (x, y) esprimendo la loro misura in metri invece che in gradi decimali, in questo modo vengono semplificati, ma anche resi più precisi, i calcoli di distanza tra i punti della Terra. Questa proiezione è specifica per la zona UTM 32N, che copre una parte dell'Europa, in particolare la zona centrale dell'Europa, ed è quindi utilizzabile in applicazioni locali, come sistemi di mappatura e navigazione su scala regionale

1.3 Visualizzazione della qualità dell'aria

In questa sezione saranno approfondite le possibili tecniche utilizzate per rappresentare i dati relativi alla qualità dell'aria.

1.3.1 Air Quality Index

Il primo luogo è necessario specificare cosa si intende con "qualità dell'aria": in questo contesto per "qualità dell'aria" si intende la quantità delle

sostanze inquinanti presenti nell'aria stessa. Più sarà alto il livello di inquinanti più sarà bassa la qualità dell'aria e viceversa. L'**A**ir **Q**uality **I**ndex è un indicatore di sintesi utilizzato per rappresentare la qualità dell'aria aggregando le informazioni relative agli inquinanti presenti in un certo periodo. Di AQI ne esistono tanti e differiscono per metodi di calcolo, inquinanti utilizzati, periodo temporale rappresentato e scala di colori associata.

In genere vengono considerati come sostanze inquinanti

- Particolato (PM_{10}, $PM_{2.5}$)
- Ozono (O_3)
- Diossido di azoto (NO_2)
- Anidride solforosa (SO2)
- Monossido di carbonio (CO)

Tra i tanti indici definiti si descriveranno come esempio l'European Air Quality Index e l'U.S. Air Quality Index.

European Air Quality Index è l'indice utilizzato dagli stati membri della European Environment Agency e rappresenta il livello corrente di qualità dell'aria utilizzando sei livelli diversi: Good (verde acqua), Fair (verde), Moderate (Giallo), Poor (rosso), Very Poor (rosso ciliegia) e Extremely Poor (viola). Vengono utilizzati come inquinanti PM_{10}, $PM_{2.5}$, considerando la loro concentrazione nelle passate ventiquattro ore e O_3, NO_2 e SO_2, considerando la loro concentrazione oraria. Per calcolare l'indice viene calcolato il livello di qualità dell'aria considerando singolarmente i singoli inquinanti come rappresentato nella tabella 1.1. L'indice poi corrisponde al livello di qualità dell'aria più scarso tra quelli calcolati [6] [7].

	Good	Fair	Moderate	Poor	Very poor	Extremely poor
Particles less than 2.5 µm ($PM_{2.5}$)	0-10	10-20	20-25	25-50	50-75	75-800
Particles less than 10 µm (PM_{10})	0-20	20-40	40-50	50-100	100-150	150-1200
Nitrogen dioxide (NO_2)	0-40	40-90	90-120	120-230	230-340	340-1000
Ozone (O_3)	0-50	50-100	100-130	130-240	240-380	380-800
Sulphur dioxide (SO_2)	0-100	100-200	200-350	350-500	500-750	750-1250

Figura 1.1: Livelli di qualità dell'aria per EAQI

U.S. Air Quality Index: è l'indice definito dalla Environmental Protection Agency utilizzato non solo negli stati uniti ma anche in altri paesi. Si tratta di un valore numerico che va da 0 a 500 e più è alto maggiore è il livello di inquinamento. Sono definite sei fasce diverse con sei colori corrispondenti per interpretare il dato: Good (0-50, verde), Moderate (51-100, giallo), Unhealthy for Sensitive Groups (101-150, arancione), Unhealty (151-200, rosso), Very Unhealty(201-300, viola), Hazardous (301-500, rosso ciliegia). Il metodo di calcolo prevede di calcolare l'indice per ogni inquinante (gli stessi utilizzati per l'indice europeo) utilizzando la tabella 1.7 e l'equazione

$$I_p = \frac{I_{Hi} - I_{Lo}}{BP_{Hi} - BP_{Lo}}(C_p - BP_{Lo}) + I_{Lo} \tag{1.1}$$

in cui

- I_p = il valore dell'indice per l'inquinante p

- C_p = la concentrazione troncata dell'inquinante p
- BP_{Hi} = il breakpoint che è maggiore o uguale a C_p
- BP_{Lo} = il breakpoint che è minore o uguale a C_p
- I_{Hi} = il valore AQI corrispondente a BP_{Hi}
- I_{Lo} = il valore AQI corrispondente a BP_{Lo}

Fatto questo il valore dell'indice corrisponde all'indice più alto calcolato [8] [9] [10].

These breakpoints							Equal these AQI's	
O^3 (ppm) 8-hour	O^3 (ppm) 1-hour [1]	$PM^{2.5}$ (µg/m [3]) 24-hour	PM^{10} (µg/m [3]) 24-hour	CO (ppm) 8-hour	SO^2 (ppb) 1-hour	NO^2 (ppb) 1-hour	AQI	Category
0.000-0.054	-	0.0 - 12.0	0-54	0.0-4.4	0-35	0-53	0-50	Good.
0.055-0.070	-	12.1 - 35.4	55-154	4.5-9.4	36-75	54-100	51-100	Moderate.
0.071-0.085	0.125-0.164	35.5 - 55.4	155-254	9.5-12.4	76-185	101-360	101-150	Unhealthy for Sensitive Groups.
0.086-0.105	0.165-0.204	[3] 55.5 - 150.4	255-354	12.5-15.4	[4] 186-304	361-649	151-200	Unhealthy.
0.106-0.200	0.205-0.404	[3] 150.5 - 250.4	355-424	15.5-30.4	[4] 305-604	650-1249	201-300	Very Unhealthy.
0.201-([2])	0.405-0.504	[3] 250.5 - 350.4	425-504	30.5-40.4	[4] 605-804	1250-1649	301-400	Hazardous.
([2])	0.505-0.604	[3] 350.5 - 500.4	505-604	40.5-50.4	[4] 805-1004	1650-2049	401-500	

Figura 1.2: Tabella per il calcolo del U.S. AirQuality Index

1.3.2 Casi di studio

Ai fini della tesi è utile indagare quali sono le applicazioni che utilizzano dati sulla qualità dell'aria, per quali scopo li utilizzano e come li rappresen-

tano, sia per studiare un'interfaccia opportuna per il sistema da realizzare che per trovare una soluzione al problema del calcolo del percorso meno inquinato.

Possiamo individuare in questa prospettiva tre tipologie di applicazioni:

- Applicazioni che hanno come obiettivo principale quello di rendere disponibili agli utenti i dati sulla qualità dell'aria.
- Applicazioni in cui i dati ambientali sulla qualità dell'aria non fanno parte del core-domain dell'applicazione ma che tuttavia vengono utilizzate per arricchire il servizio offerto
- Applicazioni che utilizzano dati sulla qualità dell'aria svolgere i propri task

La seconda tipologia di applicazioni non rappresenta un argomento di studio interessante ai fini dello sviluppo dell'interfaccia di Ecosister. Per la prima e la terza categoria invece si elencheranno alcuni casi d'uso rilevanti in modo da identificare quali sono gli strumenti e le modalità di presentazione dei dati utili in questo genere di applicazioni.

Elemento ricorrente in tutte i software presi in analisi è **la mappa**, la quale occupa di solito quasi tutta l'interfaccia grafica. In genere si tratta di una mappa del globo, centrata e ingrandita sulla zona dove sono compiute le rilevazioni o nel punto della posizione GPS fornita dal device su cui è eseguito software. Sulla mappa è possibile effettuare quelle operazioni che in genere ci si aspetta di poter fare in un'applicazione che presenta questo strumento, in particolare effettuare ingrandimenti e spostarsi sulla mappa stessa. Gli elementi presenti sulla mappa differiranno a seconda della tipologia di applicazione e dallo suo scopo.

Va detto per completezza che ci sono altri software non presi in esame che forniscono dati relativi alla qualità dell'aria senza mostrare una mappa, come il calcolatore del U.S. AQI realizzato da AirNow [11]. Questa tipologia di interfaccia non è stata approfondita, in quanto non permette di evidenziare anche da un punto di vista grafico la stretta correlazione tra il dato sulla

qualità dell'aria e la posizione geografica, e non è adeguata per la realizzazione della funzionalità di navigazione prevista per Ecosister.

Al primo tipo di applicazioni appartengono tutte quelle applicazioni realizzate degli enti che definiscono i vari AQI, più altre dalla scopo divulgativo, informativo e di monitoraggio come *PlumeLabs: Air Report* e *IQAir*.

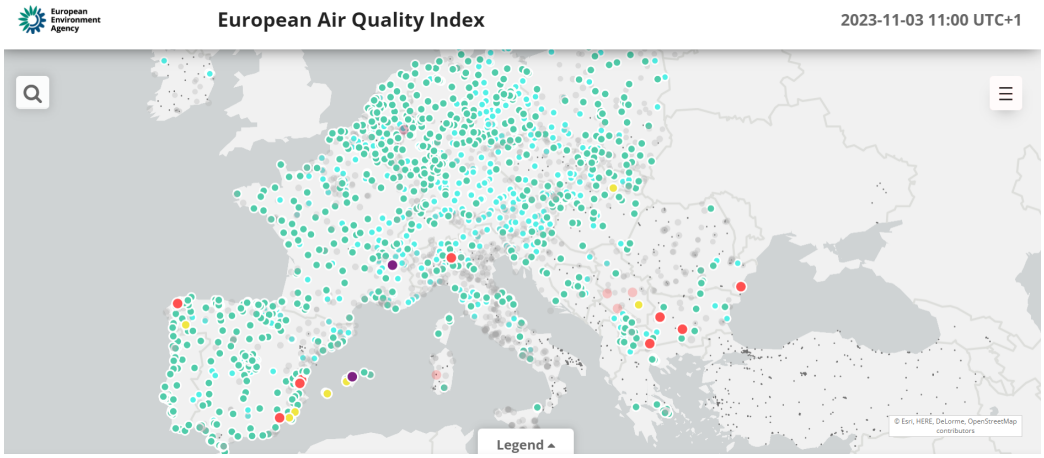

Figura 1.3: Interfaccia di European Air Quality Index con marker colorati

L'applicazione della EEA **European Air Quality Index** ha come scopo quello di rendere disponibile il valore in tempo reale del European AQI nelle zone eurpee monitorate. L'interfaccia dell'applicazione della European Air Quality Index è costituita sostanzialmente da una mappa del globo centrata sull'Europa.

Per rappresentare informazioni relative alla qualità dell'aria si avvale principalmente dell'uso dello strumento dei **marker**: i marker sono dei punti evidenziati su mappa atti ad indicare delle zone di interesse, e possono presentare una certa colorazione o sfruttare un'icona per rendere immediatamente disponibile un'informazione. In questo caso specifico i marker sono colorati in base al colore del Euroepean AQI associato al valore dell'indice calcolato dalla stazione di rilevazione presente in quel punto specifico.

I marker si prestano bene anche per essere usati come elementi "cliccabili", evidenziando che si possono avere ulteriori informazioni in aggiunta all'informazione immediatamente disponibile data dall'icona rappresentante il marker stesso. Nel caso dell'applicazione presa in esame un click sul marker permette di aprire un pop-up che specifica il nome della stazione di rilevazione il valore numerico dell'indice, il principale inquinante, e presenta un link che ridirige a una trattazione più approfondita con una retrospettiva temporale più ampia sui dati campionati. ! SI PUò APPROFONDIRE QUESTO PUNTO SULLE INFORMAZIONI AGGIUNTIVE !

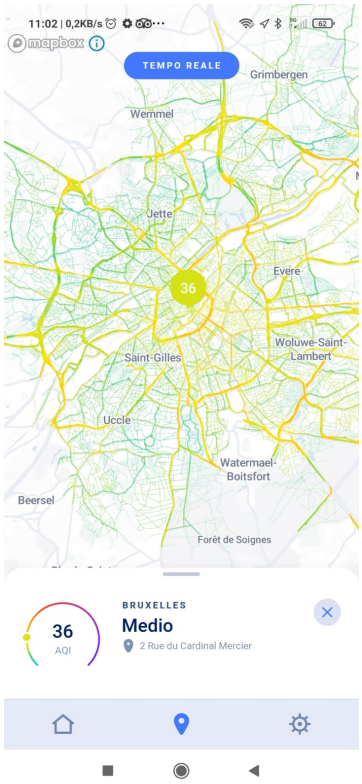

Figura 1.4: Interfaccia di PlumeLabs con rappresentazione street by street

L'applicazione mobile **PlumeLabs: Air Report** [12] nasce con lo scopo di divulgare informazioni in tempo reale sulla qualità dell'aria in modo da aumentare la consapevolezza degli utenti su questa tematica.

L'applicazione presenta un'interfaccia con due schermate per la visualizzazione dei dati: nella prima compare una lista di *card* ognuna rappresentate una posizione sulla mappa salvata dall'utente con una serie di dati sulla qualità dell'aria; nella seconda si trova una mappa tematica con una **visualizzazione** del livello di qualità dell'aria **street-by-street**: ingrandendo sulle città in cui avvengono le rilevazioni, evidenziate da marker di colore blu che una volta cliccato mostra valore numerico e colore del AQI index di quella città, diventa più nitido il reticolo stradale, colorato in base alla qualità dell'aria presente in ogni punto della strada. Questo tipo di visualizzazione permette di evidenziare, utilizzando un opportuno livello di zoom, non solo quanto complessivamente un'area è inquinata, ma soprattutto permette di rappresentare il livello di inquinamento presente in un tragitto da percorrere (anche se non è presente la funzionalità di calcolo di percorsi da un punto di partenza a uno di arrivo).

L'applicazione consente di utilizzare diverse tipologie di AQI e l'interfaccia adatta i colori utilizzati per rappresentare i dati alle specifiche definite per ogni tipologia.

L'applicazione web **IQAir** [13] ha sempre una finalità divulgativa con oggetto la qualità dell'aria a cui vengono aggiunti altri elementi di natura atmosferica che potrebbero influenzare il valore dei dati, come ad esempio il meteo, la temperatura e le informazioni sui venti. Nell'interfaccia compare la mappa del globo di tipo **heatmap** in cui le varie zone hanno una colorazione correlata al U.S. AQI presente in quell'area.

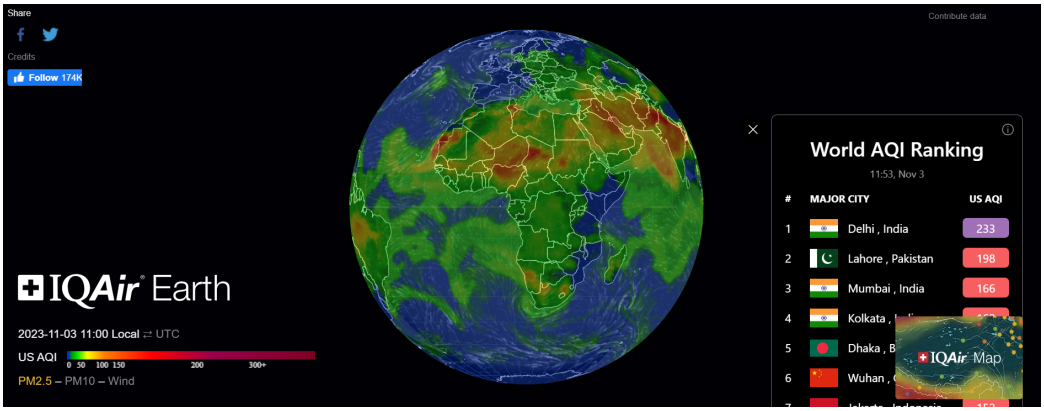

Figura 1.5: Interfaccia di IQAir con rappresentazione dei dati tramite heatmap

Nella sottosezione successiva verrà approfondito come vengono rappresentati i dati utilizzando l'interpolazione.

La terza tipologia di applicazioni sarà invece approfondita successivamente nella sezione 1.4 per remdere più omogenea la trattazione in quanto questo studio risulta maggiormente pertinente alla tematica riguardante il calcolo del percorso meno inquinato.

1.3.3 Metodi di interpolazione

La rappresentazione dei dati sia mediante visualizzazione *street-by-street* che *heatmap* presenta la difficoltà di dover rappresentare il valore della qualità dell'aria in ogni punto della mappa nonostante i sensori e le stazioni di rilevazione siano ovviamente presenti soltanto in certi punti specifici e hanno una copertura limitata. Diventa necessario quindi scegliere un metodo di **interpolazione** geostatistico per interpolare i dati. I metodi geostatistici sono tecniche matematiche il cui scopo è quello di fare previsioni o inferenze su un fenomeno geografico in luoghi non campionati a partire da dati raccolti in determinati punti. Questi metodi sono fondamentali in molte discipline come

la geologia, la meteorologia, l'agricoltura, la forestale, l'ingegneria ambientale e altre scienze della terra.

Uno dei possibili metodi di interpolazione è l'**I**nverse **D**istance **W**eighting [14]. L'idea alla base del metodo è che i punti più vicini ad una località non campionata avranno maggiore influenza sulla previsione rispetto a quelli più lontani.

Il calcolo dell'IDW si basa sul concetto di "peso" assegnato a ciascun punto conosciuto, che diminuisce all'aumentare della distanza dal punto di previsione. La formula generale dell'IDW per un punto di previsione è:

$$P(x) = \frac{\sum_{i=1}^{n} w_i \cdot v_i}{\sum_{i=1}^{n} w_i}$$

dove:

- $P(x)$ è il valore previsto nel punto di interesse.
- n è il numero totale di punti campionati utilizzati per la previsione.
- v_i è il valore misurato nel i-esimo punto campionato.
- w_i è il peso assegnato al i-esimo punto campionato, e di solito è calcolato come $\frac{1}{d_i^p}$, dove d_i è la distanza tra il punto campionato e il punto di interesse, e p è un parametro di potenza che determina quanto rapidamente il peso diminuisce con la distanza.

Il parametro di potenza p è un aspetto critico dell'IDW: un valore più alto dà maggiore importanza ai punti più vicini, mentre un valore più basso rende la superficie interpolata più liscia. Non esiste un valore "ottimale" universale per p; questo può variare a seconda del tipo di dati e dell'applicazione specifica, e può essere determinato sperimentalmente o attraverso la validazione incrociata.

I vantaggi del metodo IDW includono la sua semplicità e il fatto che è deterministico, il che significa che produrrà lo stesso risultato ogni volta che viene utilizzato con lo stesso set di dati. Tuttavia, ci sono anche alcune limitazioni: l'IDW può produrre artefatti come "bolle" attorno ai punti con

valori estremamente alti o bassi, e non tiene conto di eventuali tendenze spaziali o direzionali nei dati.

Un altro metodo di interpolazione è il **Kriging** [15], una tecnica statistica avanzata capace di fornire stime interpolate che tengono conto sia della distanza sia della direzione tra i punti campionati. Quando si parla di "direzione" dei punti campionati nel contesto del Kriging si fa riferimento all'anisotropia spaziale dei dati. L'anisotropia si verifica quando la correlazione spaziale tra i punti di campionamento varia a seconda della direzione e non soltanto a seconda della distanza.

Un esempio classico è l'analisi della topografia di una zona montuosa. La variazione dell'altitudine potrebbe essere molto diversa se ci si muove lungo il pendio di una montagna rispetto a se ci si muove parallelamente al pendio. In questo caso, c'è una chiara anisotropia nei dati perché la direzione del movimento influisce sulla variazione dell'altitudine che si osserva.

L'approccio del Kringing permette di ottenere stime interpolate più accurate in ambienti in cui l'anisotropia è evidente.

Di seguito i passaggi fodamentali per il calcolo dei dati interpolati.

- Calcolo del modello di Variogramma: Il cuore del Kriging è il modello di variogramma, che descrive come la correlazione spaziale dei dati cambia con la distanza e la direzione.

    1. Calcolo delle differenze quadrate: Per ogni coppia di punti campionati, calcoli la differenza quadrata dei loro valori (ad esempio, altezze, concentrazioni di minerali, ecc.). La differenza quadrata per una coppia di punti i e j è data da $(z_i - z_j)^2$, dove z_i e z_j sono i valori dei punti.

    2. Raggruppamento per Bins di Distanza: Le coppie di punti vengono poi raggruppate in base alla loro separazione spaziale. Ad esempio, potresti avere un bin per coppie di punti che sono distanti tra 0 e 10 metri, uno per quelli tra 10 e 20 metri, e così via.

3. Calcolo della Semivarianza: Per ogni bin di distanza, si calcola la semivarianza come la media delle differenze quadrate delle coppie di punti all'interno di quel bin. La semivarianza per un bin è data da $\frac{1}{2N(h)} \sum_{i,j}(z_i - z_j)^2$, dove $N(h)$ è il numero di coppie di punti nel bin a distanza h.

4. Plot del Variogramma Empirico: Questi valori di semivarianza vengono poi tracciati rispetto alla distanza, producendo un grafico che mostra come la semivarianza cambia con la distanza. Questo è il variogramma empirico.

5. Modellazione del Variogramma Teorico: Infine, un modello teorico di variogramma (ad esempio, sferico, esponenziale, gaussiano, ecc.) viene adattato al variogramma empirico. Questo modello teorico fornisce una funzione continua che descrive la correlazione spaziale in tutta l'area di interesse e viene utilizzato nei calcoli di Kriging.

- Calcolo del valore dei punti non campionati: Il primo passo per fare ciò è decidere quali sono i punti campionati da tenere in considerazione per il calcolo. Successivamente viene definito il peso che ogni punto ha nella stima. Questo può essere fatto in vari modi e uno dei possibili è la risoluzione di un sistema lineare come quello seguente.

$$\begin{bmatrix} \gamma(\mathbf{x}_1, \mathbf{x}_1) & \gamma(\mathbf{x}_1, \mathbf{x}_2) & \dots & \gamma(\mathbf{x}_1, \mathbf{x}_n) & 1 \\ \gamma(\mathbf{x}_2, \mathbf{x}_1) & \gamma(\mathbf{x}_2, \mathbf{x}_2) & \dots & \gamma(\mathbf{x}_2, \mathbf{x}_n) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \gamma(\mathbf{x}_n, \mathbf{x}_1) & \gamma(\mathbf{x}_n, \mathbf{x}_2) & \dots & \gamma(\mathbf{x}_n, \mathbf{x}_n) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \\ \mu \end{bmatrix} = \begin{bmatrix} \gamma(\mathbf{x}_1, \mathbf{x}_0) \\ \gamma(\mathbf{x}_2, \mathbf{x}_0) \\ \vdots \\ \gamma(\mathbf{x}_n, \mathbf{x}_0) \\ 1 \end{bmatrix}$$

dove $\gamma(\mathbf{x}_i, \mathbf{x}_j)$ rappresenta la semivarianza tra i punti campionati $\mathbf{x}_i$ e $\mathbf{x}_j$, λ_i sono i pesi di Kriging da determinare per i punti campionati, μ è una variabile lagrangiana che emerge dal vincolo che la somma dei pesi

sia uguale a uno, e $\gamma(\mathbf{x}_i, \mathbf{x}_0)$ è la semivarianza tra i punti campionati e il punto non campionato $\mathbf{x}_0$ per il quale stiamo effettuando la stima.

Per calcolare il valore non campionato basta poi risolvere la seguente equazione:

$$\hat{Z}(\mathbf{x}_0) = \sum_{i=1}^{n} \lambda_i Z(\mathbf{x}_i) \tag{1.2}$$

Dove:

- $\hat{Z}(\mathbf{x}_0)$ è la stima Kriging nel punto non campionato $\mathbf{x}_0$.
- λ_i sono i pesi Kriging calcolati dal sistema di equazioni lineari.
- $Z(\mathbf{x}_i)$ sono i valori campionati nei punti $\mathbf{x}_i$.
- n è il numero di punti campionati utilizzati nella stima.

1.4 Calcolo del percorso

In questa sezione verrà approfondita la tematica del calcolo del percorso da un punto di partenza a un punto di destinazione, riportando anche lo studio effettuato sulle applicazioni di navigazione che utilizzano nel calcolo del percorso anche dati sulla qualità dell'aria.

1.4.1 Applicazioni proprietarie

Il calcolo del percorso minimo è un task che viene svolto da tutte le applicazioni che si propongono di svolgere il servizio di navigazione stradale come Google Maps, Mappe di IOS o Waze. Tutte queste applicazioni sfruttando un'estesa base di dati di informazioni geografiche e stradali, rappresentano la terra come un grafo pesato e poi utilizzano un algoritmo per il calcolo dei percorsi minimi sul grafo per calcolare il miglior percorso da un punto di origine a uno di destinazione. Nelle versioni più moderne vengono integrate anche dati real-time relativi al traffico, informazioni personalizzate relative

all'utente che utilizza l'applicazione, e algoritmi di intelligenza artificiale per ottimizzare i risultati.

Le applicazioni più popolari come quelle sopracitate non rilasciano informazioni particolarmente dettagliate né riguardanti gli algoritmi utilizzati né sulle informazioni stradali sfruttate per il calcolo dei percorsi.

Per quanto riguarda le informazioni utilizzati sia Google, che è propietaria anche del servizio Waze, che Apple, acquistano i dati da fonti di terze parti e hanno i propri servizi di raccolta dati, come ad esempio la flotta di auto Google Street View [16].

Per quanto riguarda gli algoritmi utilizzati sul grafo rappresentante il reticolo stradale vengono sfruttati principalmente due algoritmi:

1. Il noto algoritmo di Dijkstra per il calcolo dei cammini minimi [17]
2. L'algoritmo euristico A* [18]

1.4.2 OpenTripPlanner e OpenStreetMap

Oltre a queste applicazioni popolari con carattere però propretario sono stati studiati anche alcuni strumenti open-source, in particolare il software OpenTripPlanner e OpenStreetMap.

OpenTripPlanner(OTP) [19] è un progetto software open-source che offre servizi di informazioni per passeggeri e analisi delle reti di trasporto. La forza principale di OTP risiede nella sua capacità di pianificare viaggi attraverso vari modalità di trasporto, e di poter sfruttare dati di informazioni geografiche in diversi formati. OTP è in grado di creare itinerari che combinano diversi modi di trasporto, come transito, pedonale, bicicletta e auto, utilizzando reti costruite a partire dai dati di OpenStreetMap e GTFS.

Dal suo lancio nel 2009, OTP è cresciuto fino a supportare una vasta gamma di servizi di pianificazione di viaggi in tutto il mondo, inclusi servizi regionali e nazionali, nonché diverse applicazioni mobili multi-città.

Tra le regioni e i servizi che utilizzano OTP si trovano [20]:

- Norvegia (a livello nazionale)

- Finlandia (a livello nazionale)
- Stato di New York Dipartimento di Stato di Il pianificatore di viaggi di trasporto pubblico dei trasporti fornisce gli itinerari per i sistemi di trasporto pubblico in tutto lo stato in un'unica istanza OTP unificata.
- ViviBus Bologna Bologna, Italia.
- Regione Piemonte, Italia e la Città di Torino costruito su OpenTripPlanner di 5T.
- ViaggiaTrento e ViaggiaRovereto sono stati implementati nell'ambito del Progetto SmartCampus, una ricerca progetto fondato da TrentoRise, UNITN, e FBK.
- OTP Android di CUTR-USF e Vreixo González possono trovare itinerari su molti server OTP diversi tramite un servizio meccanismo di scoperta.

OpenStreetMap (OSM) è un progetto collaborativo che crea mappe libere e modificabili. Fondato nel 2004, che ha rivoluzionato il modo in cui i dati geografici sono raccolti e condivisi. Tra le sue caratteristiche fondamentali:

1. **Dati Geografici Liberi e open-source**: Fornisce dati geografici sotto la licenza Open Database License (ODbL).
2. **Contributi della Comunità**: Gli utenti contribuiscono migliorando e aggiornando i dati attraverso osservazioni sul campo, dati GPS, o interpretazioni di immagini satellitari.
3. **Versatilità e Personalizzazione**: I dati possono essere utilizzati per una vasta gamma di applicazioni, dalla navigazione alla pianificazione urbana.

4. **Aggiornamenti Frequenti e Dinamicità**: La mappa è costantemente aggiornata riflettendo cambiamenti rapidi nel paesaggio urbano e naturale.

5. **Accessibilità Globale**: Copre il globo intero, fornendo dati anche per aree meno mappate.

I dati di OSM trovano applicazioni in vari settori:

1. **Navigazione e Pianificazione del Viaggio**: Creazione di applicazioni di navigazione per diversi mezzi di trasporto.

2. **Analisi Urbana e Pianificazione**: Studio della morfologia urbana e pianificazione dei trasporti.

3. **Gestione delle Emergenze**: Utilizzo dei dati per mappare rapidamente aree colpite da disastri naturali.

4. **Sviluppo di Giochi e Realtà Aumentata**: Impiego nel settore dei giochi basati sulla geolocalizzazione.

5. **Ricerca Ambientale e Conservazione**: Monitoraggio e analisi dell'ambiente naturale.

Nonostante i vantaggi, OSM affronta alcune sfide:

1. **Qualità Variabile dei Dati**: La precisione dei dati può variare a seconda delle regioni.

2. **Manutenzione e Aggiornamenti**: Richiede un impegno costante per mantenere la mappa aggiornata.

3. **Questioni Legali e di Privacy**: Necessità di considerare le implicazioni legali e di privacy nella mappatura di aree sensibili.

Nel capitolo successivo sia OTP che OSM saranno presentati con maggiore dettaglio e pertinenza in quanto utilizzati per la realizzazione dell'elaborato finale.

1.4.3 GreenPath e CleanAirRouteFinder

Studiando lo stato dell'arte delle applicazioni che integrano nel calcolo dei percorsi anche dati sulla qualità dell'aria, appartenenti quindi alla terza categoria descritta nella sezione 1.3.2, sono stati identificate soltanto due applicazioni: **GreenPath** e CleanAirRouteFinder.

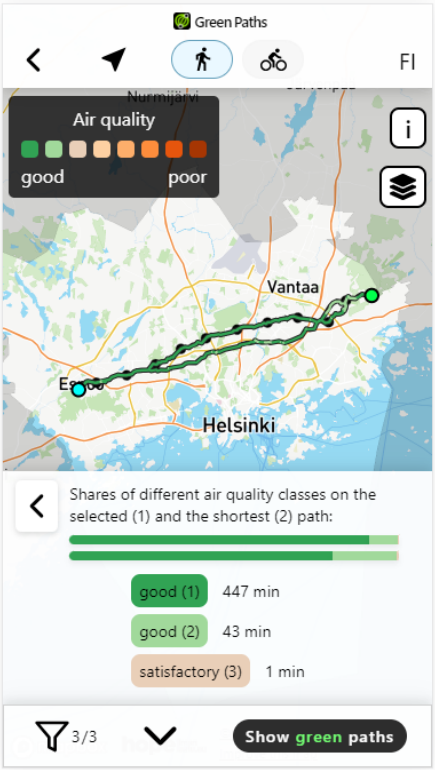

Figura 1.6: Interfaccia di GreenPath

L'applicazione GreenPath, sviluppata dal Digital Geography Lab dell'Università di Helsinki, utilizza una serie di tecnologie avanzate per il calcolo dei percorsi e la visualizzazione delle mappe [21] [22].

Il server di GreenPath calcola percorsi di aria fresca, con meno rumore e più vegetazione per camminate e ciclismo nella regione della capitale di Helsinki. Sfrutta i dati dell'indice di qualità dell'aria sperimentale, AQI 2.0, dal sistema di modellazione FMI-ENFUSER sviluppato dall'Istituto Meteorolo-

gico Finlandese. L'AQI 2.0 si basa su dati orari in tempo reale come misura composita di NO2, PM2.5, PM10, O3, carbonio nero e superficie di deposito polmonare. Inoltre, applica dati di rumore modellati da traffico stradale e ferroviario secondo la Direttiva europea sul rumore ambientale. I dati relativi al verde a livello stradale sono derivati dall'analisi delle immagini di Google Street View e dai dati di copertura del suolo disponibili apertamente forniti da HRI.

GreenPath fornisce diversi spunti per lo sviluppo dell'elaborato di tesi soprattutto da un punto di vista delle tecnologie utilizzate per il calcolo dei percorsi: il servizio sfrutta infatti proprio gli strumenti open-source OTP e OSM per creare un grafo pesato della rete stradale della città di Helsinki e per calcolare i percorsi brevi utilizza l'algoritmo di Dijkstra. Anche l'interfaccia di GreenPath è stata ritenuta interessante in quanto permette di visualizzare per ogni percorso calcolato, per ogni tratto di strada, l'indice di qualità dell'aria e il tempo di percorrenza, in modo che l'utente può sapere per quanto, in termini di tempo e chilometri, sarà esposto a un certo livello di inquinamento.

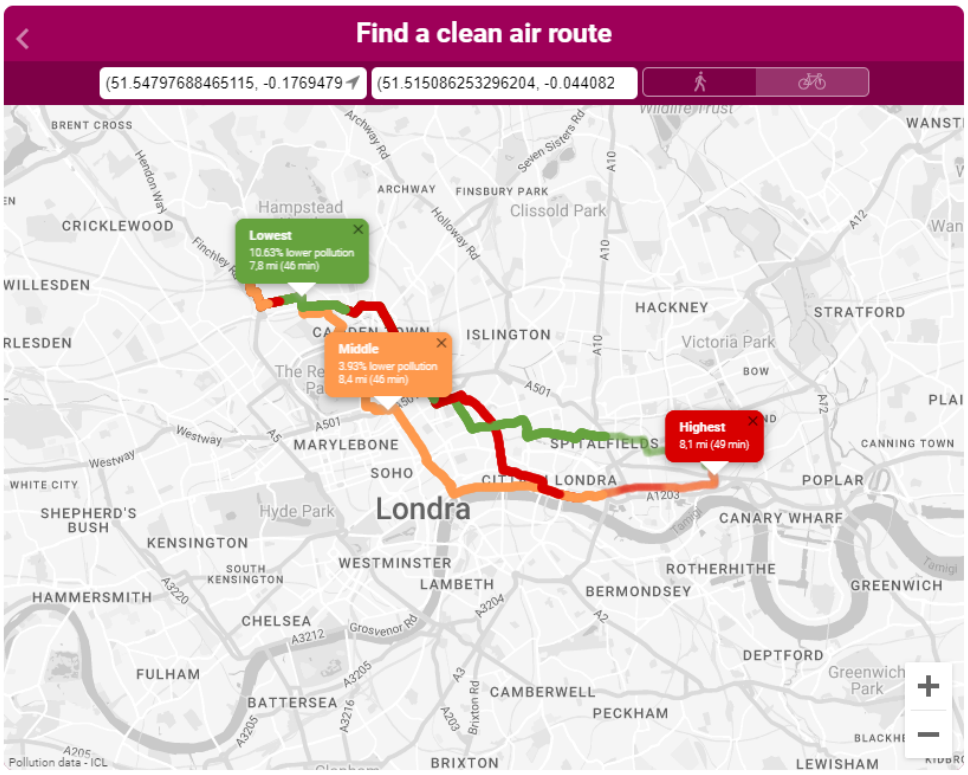

Figura 1.7: Interfaccia di CleanAirRouteFinder

Il Clean Air Route Finder [23] è uno strumento interattivo sviluppato attraverso il Mayor's Air Quality Fund di Londra. Questo servizio permette agli utenti di inserire un percorso e ricevere un'opzione a bassa inquinamento per camminare. Il progetto Cleaner Air Better Business (CABB) ha creato questo strumento, che è stato integrato sui siti web dei suoi partner del Business Improvement District.

Da un punto di vista di funzionalità l'applicazione non si discosta dal software GreenPath e non essendo disponibile codice sorgente, CleanAirRouteFinder è stato studiato principalmente come strumento di confronto con il software GreenPath.

1.5 Funzionalità del sistema

In questa sezione verranno descritti nel dettaglio i requisiti del sistema da realizzare a fronte dello studio effettuato.

Scopo del progetto di tesi è la realizzazione di un'applicazione web che rispetta i requisiti funzionali e non funzionali espressi di seguito.

1.5.1 Requisiti funzionali

- **Visualizzazione della mappa**: l'utente dovrà poter visualizzare una mappa in cui
  - è chiaramente evidenziata l'area di Bologna su cui è attivo il progetto Ecosister
  - sarà possibile visualizzare diverse colorazioni della mappa che esprimono i diversi livelli di inquinamento, in particolare una colorazione *heatmap* e una colorazione *street-by-street*. Per la colorazione e i valori dell'inquinamento dovrà essere utilizzata la specifica dell'indice di qualità dell'aria europeo.
  - sarà possibile visualizzare diverse colorazioni della mappa che esprimono la frequenza di incidenti sulle strade, in particolare una colorazione *heatmap* e una colorazione *street-by-street*.
  - dovrà essere possibile visualizzare i sensori che compiono le rilevazioni del livello di inquinamento. Per ogni sensore dovranno essere resi disponibili l'identificativo del sensore, e le informazioni sull'indice della qualità dell'aria e il livello dei singoli inquinanti registrato.
- **Calcolo del percorso più corto, meno inquinato e meno pericoloso**: l'utente dovrà avere la possibilità di selezionare un punto di partenza e uno di destinazione selezionandoli dalla mappa, e di chiedere successivamente al sistema di calcolare il percorso più veloce, meno inquinato e con meno possibilità di essere coinvolto in un incidente.

Ad ogni strada l'applicazione dovrà assegnare infatti un indice di pericolosità basato sulla frequenza annua di incidenti stradali verificati nella zona e utilizzare il dato nel calcolo del percorso. Con il valore 0 si indicherà una strada poco pericolosa, con il valore 10 una strada molto pericolosa. Per la didascalia coropletrica dell'indice si avrà che il colore verde rappresenta una strada poco pericolosa, con il colore giallo una strada mediamente pericolosa e con il colore rosso si indicherà una strada molto pericolosa.

  - Dovrà essere possibile specificare l'intenzione dell'utente di percorrere la strada a piedi o in bicicletta adattando la risposta in base al parametro selezionato
  - il sistema dovrà mostrare sulla mappa il percorso tra il punto di partenza e di destinazione specificando: lunghezza del percorso, tempo di percorrenza, periodo di esposizione a un certo livello di inquinamento e a un certo indice di pericolosità sia in termini di chilometri da percorrere che in termini di tempo.
  - In caso siano identificati percorsi di durata o caratteristiche simili dovrà essere possibile visualizzare, selezionare e confrontare le diverse possibilità

- **Gestione del profilo utente**: l'utente dovrà avere la possibilità di creare un proprio profilo sull'applicazione e di poter poi effettuare il login. Oltre al nome utente e alla password all'utente che ha effettuato il login sarà data la possibilità di:

  - selezionare sulla mappa una posizione e evidenziarla come posizione *casa*
  - selezionare sulla mappa una posizione e evidenziarla come posizione *lavoro*
  - selezionare sulla mappa un punto di partenza e un punto di destinazione e di salvare il percorso etichettandolo con un nome a

piacere. All'utente sarà poi consentito di visualizzare i percorsi già salvati e di visualizzarli ancora sulla mappa. I percorsi salvati potranno anche essere cancellati.

- **Gestione della lingua** per le didascalie e i contenuti testuali dell'applicazione dovrà essere presente sia una versione in italiano che una versione in inglese.

1.5.2 Requisiti non funzionali

- Lo sviluppo dell'interfaccia dell'applicazione dovrà essere **mobile-first**.
- *Ready-to-start*: il deployment del progetto lato server deve poter essere fatto utilizzando un solo comando. L'installazione di dipendenze necessarie al suo funzionamento deve essere limitato all'installazione di opportuni strumenti di containerizzazione e/o virtualizzazione.
- Usabilità: l'applicazione web deve presentare una interfaccia semplice, intuitiva, e dove possibile minimale, privilegiando l'utilizzo di colori, icone e simboli piuttosto che la forma testuale.
- Estendibilità: la struttura del software dovrà essere estendibile in modo tale da poter prevedere l'utilizzo di altri pesi nel calcolo del percorso oltre al tempo di percorrenza, all'indice di qualità dell'aria e all'indice di pericolosità.

1.6 Challenge affrontate durante lo sviluppo

Le principali difficoltà affrontate per la realizzazione del progetto riguardano:

- la realizzazione di un grafo che rappresenti la mappa basato su dati geografici da utilizzare per calcolare i migliori percorsi in base alla lunghezza del percorso e ai dati sulla qualità dell'aria real-time. Per

questo task sarà necessario prima di tutto identificare una fonte di dati affidabile in un formato standard, ad esempio il formato Open Street Map e definire un processing opportuno per ricavare un grafo per la definizione di un grafo i cui pesi possono essere modificati real-time.

- la visualizzazione della mappa con le coropletrie, heatmap e street-by-street in base ai colori dell'aqi europeo. Per fare questo sarà importante appoggiarsi a una opportuna libreria per la visualizzazione delle mappe, scegliere un metodo di interpolazione che permetta di definire il valore della qualità dell'aria in ogni punto della mappa a partire dai valori dei sensori

- dato che attualmente non esiste un servizio effettivo che fornisce dati sulla qualità dell'aria nel perimetro di Bologna, sarà necessario sviluppare un servizio mock-up che mette a disposizione dati sulla qualità dell'aria come se fossero rilevati da alcuni sensori appartenenti al perimetro di appartenenza. I dati prodotti dal servizio dovranno avere dei valori randomici, variabili nel tempo e tuttavia dovranno appartenere un range di valori verosimili.

Capitolo 2

Tecnologie

In questo capitolo verranno descritte le tecnologie che saranno poi impiegate nella realizzazione del progetto Ecosister. Essendo un'applicazione web le tecnologie sono state differenziate in modo da distinguere quelle impiegate lato front-end che quelle impiegate lato back-end.

2.1 Applicazione front-end

In questa sezione verranno descritte le principali tecnologie utilizzate per sviluppare l'applicazione front-end del sistema Ecosister.

L'applicazione front-end sarà sviluppata in Typescript sfruttando principalmente tre librerie:

- React per lo sviluppo dell'architettura e della struttura generale dell'applicazione
- Redux, una libreria JavaScript open-source ampiamente utilizzata per la gestione dello stato globale in applicazioni front-end, in particolare in applicazioni React
- MapboxGL per la visualizzazione della mappa

Le successive sottosezioni sono dedicate ognuna a una di queste tecnologie.

2.1.1 Typescript

TypeScript [24] è un superset di Javascript open-source sviluppato e mantenuto da Microsoft, annunciato per la prima volta nel 2012, ha guadagnato rapidamente popolarità nella comunità di sviluppatori web. L'obiettivo di TypeScript è fornire una soluzione per affrontare la complessità crescente delle applicazioni web, migliorando la manutenibilità e la scalabilità del codice JavaScript.

La caratteristica principale di Typescript è la tipizzazione statica: in Typescript è possibile definire il tipo di una variabile, i tipo dei parametri delle funzioni e il loro tipo di valore di restituzione in fase di sviluppo. Il linguaggio apporta quindi i vantaggi che caratterizzano la tipizzazione statica, quindi aiuta a prevenire errori di tipo durante la compilazione e offre un'ottima assistenza durante la scrittura del codice, grazie all'autocompletamento e al rilevamento di errori durante la stesura del codice. La tipizzazione statica aiuta anche a rendere il codice più leggibile e comprensibile, facilitando la manutenzione nel tempo.

Typescript offre inoltre diversi meccanismi provenienti dalla programmazione orientata agli oggetti: è possibile definire classi, interfacce e utilizzare concetti come l'ereditarietà e l'incapsulamento, il che facilita la strutturazione del codice in modo organizzato e coeso, migliorando sensibilemente la scalabiltà del codice scritto, la sua riutilizzabiltà e anche la sua potenza espressiva.

TypeScript ha una comunità di sviluppatori in crescita e una vasta gamma di strumenti e librerie a supporto. È supportato da numerosi editor di codice, compresi tra i più famosi Visual Studio Code e WebStorm, e offre un ecosistema di sviluppo attivo.

2.1.2 React

React [25] è una libreria Javascript open-source utilizzata per la creazione di interfacce utente. Il punto di forza della libreria è capacità di React di

rispondere dinamicamente ai cambiamenti dell'applicazione, aggiornando automaticamente l'interfaccia utente in modo efficiente e coerente, permettendo di progettare interfacce interattive e dinamiche. La caratteristica principale di React è quella di basare lo sviluppo dell'interfaccia su quelli che vengono definiti *Componenti*: questo approccio consente agli sviluppatori di scomporre l'interfaccia utente in piccoli pezzi autonomi di cui ognuno rappresenta una parte specifica dell'interfaccia utente, come un pulsante, una barra di navigazione o un modulo di login.

Questo paradigma di sviluppo basato su componenti offre numerosi vantaggi:

- migliora notevolmente la modularità del codice, rendendolo più organizzato e facilmente manutenibile.
- migliora la riutilizzabilità in quanto una volta sviluppato, lo stesso componente può essere riutilizzato più volte sia in punti diversi dello stesso progetto che in altri progetti. Con React è possibile creare una libreria di componenti personalizzati da riutilizzare nei diversi progetti da sviluppare.
- migliora l'incapsulamento: il funzionamento di un componente dipende unicamente della sua definizione che non influenza il funzionamento degli altri componenti.

Ogni componente React è formato da:

- una **descrizione** in JSX degli **elementi inclusi nel componente** con una sintassi simile all'HTML. La sintassi JSX in React è zucchero sintattico che semplifica la dichiarazione degli elementi dell'interfaccia utente all'interno del codice JavaScript.Infatti, oltre ai classici tag HTML, con la stessa sintassi, è possibile anche dichiarare componenti React. All'interno di JSX, è possibile incorporare espressioni JavaScript utilizzando le parentesi graffe, e questo consente di inserire dinamicamente valori, variabili o espressioni JavaScript all'interno

dell'interfaccia utente. L'utilizzo della sintassi JSX migliora notevolmente l'esperienza della programmazione grazie alla sua espressività e alla sua similarità con HTML. Tuttavia è necessaria una transpilazione in javascript valido prima che questo venga eseguito all'interno del browser utilizzando appositi strumenti come Babel, i quali si occupano di trasformare il codice JSX in chiamate a funzione Javascript che creano effettivamente gli elementi desiderati nel DOM.

- uno **stato** cioè un oggetto JavaScript che rappresenta i dati interni del componente e le sue relative informazioni. I valori dello stato di un componente possono variare nel tempo in base alle interazioni con l'utente, eventi o cambiamenti nel ciclo di vita del componente stesso. Questo stato influenza il comportamento dell'utente e come viene rappresentato nell'interfaccia utente. Lo stato di un componente può essere modificato tramite il metodo *setState()* definendo i componenti come classi, o tramite la funzione di aggiornamento restituita da *useState()* in componenti funzionali utilizzando hooks. Modificare lo stato è un'operazione fondamentale per gestire gli aggiornamenti dinamici nell'interfaccia utente. Quando lo stato di un componente cambia, React ricalcola la rappresentazione dell'interfaccia utente del componente.

- delle **proprietà**, che sono un meccanismo fondamentale per passare dati da un componente genitore a uno o più componenti figlio. I dati nello specifico possono essere oggetti, funzioni e variabili. Le proprietà sono oggetti JavaScript immutabili utilizzati per configurare e personalizzare il comportamento o la rappresentazione dei componenti figlio senza che questo possa modificarle. Le proprietà consentono di rendere i componenti configurabili, e ne migliorano la riutilizzabilità grazie proprio al fatto che sfruttando il meccanismo delle proprietà i componenti acquisiscono una natura parametrica che consente di sfruttarli in contesti diversi e con componenti genitori differenti.

Dalla descrizione precedente emerge che, per definire l'interfaccia utente,

i componenti sono organizzati tra di loro andando a formare una gerarchia ad albero, esattamente come succede per gli oggetti che compongono il DOM.

É possibile definire i componenti seguendo due sintassi diverse: utilizzando la sintassi delle classi, adattando la scrittura del codice a uno stile più classico e più vicino alla programmazione ad oggetti, oppure utilizzando i React Hook, utilizzando uno stile più moderno e più vicino alla programmazione funzionale.

Qualunque sia la sintassi scelta per ogni componente è importante definire aspetto, stato, proprietà e cosa deve avvenire nelle diversi fasi del suo ciclo di vita. Infatti, ogni componente React segue un preciso *lifecycle* che definisce cosa deve avvenire dalla sua creazione fino alla sua distruzione, le cui fasi sono descritte nell'elenco sottostante.

- Montaggio (Mounting): è la fase che prevede la creazione di un componente e il montaggio all'interno del DOM. Nel caso il componente sia definito come una classe viene chiamato il suo costruttore in cui in genere vengono inizializzati lo stato iniziale e vengono associati i metodi di istanza. Viene poi chiamato il metodo *render* che restituisce l'elemento UI del componente. dopo il rendering iniziale, viene chiamato il metodo *componentDidMount()* in cui di solito vengono effettuate richieste di dati iniziali e sottoscrizioni a eventi.

  Nel caso il componente sia definito tramite hook viene montato nel DOM quanto definito nello statement return della funzione che definisce il componente stesso, e viene poi chiamato l'hook *useEffect*.

- Aggiornamento (Updating): l'aggiornamento di un componente avviene quando le proprietà o lo stato subiscono un cambiamento. Il cambiamento delle proprietà avviene a partire da un componente e viene propagato a cascata a tutti i componenti figli dei livelli sottostanti di cui il componente è genitore. Il cambiamento dello stato invece avviene all'interno di un singolo componente e riguarda una modifica nei componenti figli solo nel caso lo stato sia utilizzato per definire le proprietà

di questi ultimi. Il cambiamento di stato viene effettuato utilizzando il metodo *setState* nei componenti definiti come classe, e con la chiamata alla seconda funzione restituita dalla chiamata *useState* nei componenti definiti invece con gli hook. Nei componenti React è possibile verificare la necessità di aggiornare effettivamente il cambiamento. Per fare ciò, nei componenti definiti come classi è possibile utilizzare il metodo shouldComponentUpdate, mentre nei componenti definiti come hook, è possibile specificare come secondo argomento della chiamata *useEffect* una lista di proprietà e variabili di stato da prendere in considerazione per avviare una procedura di aggiornamento. Dopo un aggiornamento riuscito, viene chiamato il metodo *componentDidUpdate* nei componenti definiti come classe: è il luogo ideale per interagire con il DOM o effettuare azioni post-aggiornamento. Nei componenti definiti come hook viene invece chiamata ancora la funzione *useEffect*.

- Smontaggio (Unmounting): è la fase in cui un componente viene smontato dal DOM. Nei componenti definiti come classi viene chiamato il metodo *componentWillUnmount*. È il posto migliore per eseguire pulizie o cancellare sottoscrizioni a eventi. Nei componenti definiti invece come hook è possible definire una funzione che viene restituita dalla funzione *useEffect* che viene chiamta prima che il componente venga rimosso dall'albero.

2.1.3 Redux

Redux [26] è una libreria JavaScript che nasce per rispondere alla necessità di gestire in modo efficiente e prevedibile lo stato di un'applicazione JavaScript complessa. La principale ispirazione per Redux è stata l'architettura Flux, sviluppata da Facebook per gestire lo stato nelle applicazioni React. Tuttavia, Redux ha semplificato e raffinato l'innovazione apportata da Flux, rendendolo più accessibile e facile da utilizzare per gli sviluppatori. Redux ha rapidamente guadagnato popolarità ed è diventato uno standard

de facto per la gestione dello stato in applicazioni scritte anche utilzzando React, grazie all'integrazione scritta per questa libreria.

Come principali vantaggi Redux offre:

- Prevedibilità: Redux promuove un flusso unidirezionale dei dati, il che significa che il ciclo di vita dello stato è facile da comprendere e prevedere. Questo rende il debugging e la manutenzione dell'applicazione molto più semplici.

- Centralizzazione dello stato: In Redux, tutto lo stato dell'applicazione è contenuto in un singolo "store" centralizzato. Questo consente di accedere facilmente allo stato da qualsiasi punto dell'applicazione senza doverlo passare manualmente tra i componenti.

- Time Travel Debugging: Redux offre la possibilità di registrare tutte le azioni che modificano lo stato dell'applicazione. Questo permette agli sviluppatori di eseguire il "time travel debugging", ovvero di ispezionare lo stato dell'applicazione in qualsiasi punto del passato.

- Facilità di testing: Grazie alla sua architettura modulare e alla prevedibilità del flusso dei dati, Redux semplifica il testing delle applicazioni React. È possibile testare facilmente le azioni e i riduttori separatamente per garantire che lo stato dell'applicazione si comporti come previsto.

- Redux è ampiamente adottato nella comunità di sviluppatori React, il che significa che esistono numerose risorse, librerie e strumenti disponibili per semplificare lo sviluppo con Redux.

Il funzionamento di Redux può essere sintetizzato descrivendo i suoi componenti:

- Il cuore di Redux è lo *store*, che contiene lo stato dell'applicazione. Questo stato è immutabile, il che significa che non può essere modificato direttamente se non utilizzando le "azioni" e i "riduttori".

- Le *azioni* sono oggetti JavaScript che rappresentano eventi o cambiamenti nell'applicazione. Le azioni vengono inviate allo store per richiedere una modifica dello stato.

- I *riduttori* sono funzioni che descrivono come deve cambiare lo stato dell'applicazione in risposta a un'azione. Ogni riduttore gestisce una parte specifica dello stato globale. Quando viene inviata un'azione, tutti i riduttori vengono chiamati, ma solo quelli interessati all'azione la gestiranno effettivamente.

- Redux segue un flusso unidirezionale dei dati: cioè le azioni vengono inviate allo store, che le passa ai riduttori appropriati, poi i riduttori modificano lo stato, e le modifiche allo stato vengono automaticamente rappresentate dall'interfaccia utente sfruttando i meccanismi di aggiornamento di React

Per collegare Redux a un'applicazione React, si utilizza il pacchetto React-Redux. Questo pacchetto fornisce un componente speciale chiamato *Provider* che fa da wrapper all'applicazione React e rendendo lo store Redux disponibile a tutti i componenti. Inoltre, React-Redux fornisce un'API per collegare componenti specifici allo store e accedere ai dati necessari.

2.1.4 Mapbox GL JS

Mapbox GL JS è una libreria open-source per la creazione di mappe interattive. È parte dell'ecosistema di Mapbox [27] e offre un'ampia gamma di funzionalità per la visualizzazione di dati geografici. Con Mapbox GL JS, è possibile creare mappe che permettono non solo la visualizzazione di strade e punti di interesse, ma mappe in cui è possibile incorporare anche dati complessi con effetti visivi avanzati.

Mapbox GL JS è comunemente utilizzato in applicazioni di navigazione e mappatura interattiva, come app di guida GPS, app di condivisione del tragitto, servizi di mappe online, per la visualizzazione dei dati geografici a

supporto dell'analisi dei dati, applicazioni per il settore immobiliare e applicazioni per la gestione delle risorse naturali. Mapbox GL JS è utilizzato anche per applicazioni del campo della pianificazione urbana e dell'architettura per creare mappe che mostrano l'uso del suolo, il traffico e le infrastrutture.

La libreria è stata sviluppata utilizzando Open GL, che consente di renderizzare grafica 2D e 3D direttamente nel browser. In questo modo le mappe create con Mapbox GL JS possono essere visualizzate senza la necessità di plug-in aggiuntivi o software esterni.

Le mappe Mapbox GL JS vengono renderizzate dinamicamente combinando immagini vettoriali con regole di stile nel browser anziché su server, il che rende possibile modificare lo stile delle mappe e i dati visualizzati in risposta all'interazione dell'utente in maniera molto responsiva.

Tra le principali funzionalità di Mapbox GL JS si trovano:

- Visualizzazione e animazione di dati geografici
- Interrogazione e filtraggio di elementi su una mappa
- Posizionamento dei dati tra i livelli di uno stile Mapbox
- Visualizzazione dinamica e styling dei dati lato client personalizzati su una mappa
- Visualizzazioni e animazioni di dati 3D
- Aggiunta di marcatori e popup alle mappe a livello di codice

La classe Map è la base di ogni progetto Mapbox GL JS. Di seguito, nel listato 2.1 un esempio minimale per aggiungere una mappa Mapbox in un applicazione JS in cui:

- *accessToken*: questo token di accesso a un account Mapbox.
- *container*: l'elemento HTML in cui verrà posizionata la mappa, che nell'esempio è un *¡div¿* con *id* "map".

- *style*: l'URL dello stile della mappa utilizzato per determinare quali set di dati raster o vettoriali include la mappa e lo stile che devono avere. L'esempio sopra utilizza lo stile *Mapbox Streets v12*.

- *center*: Le coordinate della posizione iniziale della mappa, in longitude e latitudine.

- *zoom*: il livello di zoom al quale la mappa deve essere inizializzata. Può essere un numero intero o un valore decimale. [28]

Listing 2.1: Inserimento di una mappa Mapbox

```
mapboxgl.accessToken = '<your access token here>';
const map = new mapboxgl.Map({
    container: 'map', // container ID
    style: 'mapbox://styles/mapbox/streets-v12', //
       style URL
    center: [-74.5, 40], // starting position [lng, lat
       ]
    zoom: 9 // starting zoom
});
```

Le mappe Mapbox GL JS possono essere composte da diversi livelli che forniscono elementi visivi e dati cartografici. Ogni livello fornisce regole su come doverebbero essere disegnati determinati dati nel browser.

Ad ogni livello è associato un *tipo* tra sfondo, riempimento, linea, simbolo, raster, cerchio, riempimento-estrusione, heatmap, hillshade e cielo [29]. Fatta eccezione per il tipo sfondo e il tipo cielo, ogni layer deve essere associato a una certa *sorgente*, la quale può essere una immagine raster, un'immagine vettoriale, un dato di tipo GeoJson (approfondito nella sezione 2.3.2) o un video. [30]. Oltre alla sorgente nella definizione del livello vengono specificati anche alcuni elementi di stile come ad esempio il colore delle geometrie rappresentate. Per fare questo possono essere utilizzate due proprietà: *layout*

che definisce il posizionameto e la visibilità degli elementi, e la cui definizione interessa la prima parte del processo di renderizzazione, e *paint* che riguarda delle caratteristiche secondarie come colore e opacità. Colore e layout possono essere anche modificati facendo si che la mappa sia ancora più dinamica.

Il metodo Mapbox GL JS *addLayer* aggiunge un livello Mapbox allo stile della mappa. L'unico parametro richiesto addLayer un oggetto layer in stile Mapbox. Può essere specificato anche un parametro opzionale*before*, che è l'ID di un livello esistente prima di inserire il nuovo livello. Omettendo questo parametro il livello sarà disegnato sopra la mappa.

Oltre a tutte queste funzionalità utili per modificare la mappa runtime per creare comportamenti che si adattano alla modifica di dati durante l'esecuzione e alle operazioni dell'utente, è possibile definire lo stile di una mappa sfruttando l'ambiente di Mapbox Studio, l'applicazione della piattaforma Mapbox per la gestione dei dati geospaziali e la progettazione di stili di mappa personalizzati. Con Mapbox Studio è possibile progettare una mappa secondo le proprie specifiche caricando e modificando i propri dati geografici nei formati sopracitati, e utilizzando le risorse e le sorgenti messe a disposizione da Mapbox [31].

Un altro punto di forza della piattaforma Mapbox sono proprio i tileset messi a disposizione dell'utente tra cui:

- Mapbox Streets v8, una rappresentazione vettoriale della rete stradale globale, Mapbox Satellite, una rappresentazione raster del globo con vista satellite.
- Mapbox Countries v1, una rappresentazione vettoriale dei confini degli stati
- Mapbox Bathymetry v2, un rappresentazione vettoriale della profondità dei fondali marini
- Mapbox NAIP, un set di immagini raster fornito da Mapbox che include immagini aeree che coprono 48 stati negli Stati Uniti contigui

- Mapbox Terrain v2, set di immagini vettoriali fornito da Mapbox che include funzionalità come ombreggiature delle colline, curve di livello topografiche e dati sulla copertura del territorio
- Mapbox Terrain-DEM v1, una versione ottimizzata di Mapbox Terrain RGB
- Mapbox Terrain-RGB v1, un set di immagini raster fornito da Mapbox contenente dati sul livello del suolo codificati in PNG come valori di colore che possono essere decodificati in altezze grezze in metri
- Mapbox Traffic v1, set di riquadri vettoriali fornito da Mapbox che puoi utilizzare per visualizzare le condizioni del traffico in tempo reale su una mappa
- Mapbox Transit v2, un set di immagini vettoriali fornito da Mapbox che include funzionalità relative al trasporto pubblico

I livelli predefiniti o defini dagli utenti in Mapbox Studio si trovano in posizione remota sulla piattaforma MapBox. In Mapbox GL JS vengono quindi caricati in maniera asincrona, utilizzando spesso l'associazione di eventi per modificare la mappa al momento giusto. Per esempio, può essere utilizzata la chiamata *map.on('load', function() ...* per far si che vengano eseguite determinate azioni solo dopo che le risorse della mappa, incluso lo stile, siano state caricate, come nell'esempio 2.2 in cui viene effettuata l'aggiunta a runtime di un altro livello.

Listing 2.2: Attesa del caricamento della mappa prima di effettuare altre azioni

```
map.on('load', () => {
    map.addLayer({
        id: 'terrain-data',
        type: 'line',
        source: {
```

```
            type: 'vector',
            url: 'mapbox://mapbox.mapbox-terrain-v2'
        },
        'source-layer': 'contour'
    });
});
```

2.2 Back-end

In questa sezione verranno trattate le principali tecnologie che impiegate per l'implementazione nel back-end di Ecosister. Principalmente saranno utilizzati Flask per l'implementazione dei servizi, e OpenTripPlanner per la costruzione della mappa come grafo pesato sfruttando i dati relativi alla zona di interesse nel formato OpenStreetMap.

Come per le tecnologie del front-end, in ognuna delle sottosezioni successive è descritta una di queste tecnologie.

2.2.1 Flask

Flask è un microframework web leggero e flessibile scritto in Python. Creato da Armin Ronacher e rilasciato per la prima volta nel 2010 come progetto open-source, nasce con l'obiettivo di creare un framework web semplice ma potente che potesse essere utilizzato per sviluppare applicazioni web in Python senza l'onere di un framework più completo come Django.

La filosofia di progettazione di Flask si basa sulla semplicità e sulla modularità. Il framework fornisce solo il minimo necessario per creare un'applicazione web, ma è altamente estensibile grazie alla vasta gamma di estensioni e librerie disponibili, permettendo flessibilità e controllo nella creazione delle proprie applicazioni web.

Flask risulta particolarmente adatto allo sviluppo di:

- **Servizi Web di Piccole e Medie Dimensioni**: Flask è particolarmente adatto per la creazione di servizi web di piccole e medie dimensioni, come siti web aziendali, blog personali e portfoli online, grazie alla sua leggerezza e facilità di utilizzo.

- **API RESTful**: Flask è ampiamente utilizzato per creare API RESTful per applicazioni web e servizi. La sua API si adatta particolarmente a questo stile architetturale, e per un adattamento ancora più calzante è possibile utilizzare anche delle sue estensioni come Flask-RESTful [32].

- **Applicazioni Web Real-time**: Flask è utilizzato anche per lo sviluppo di applicazioni web in tempo reale, come chat, giochi online e dashboard interattivi. L'integrazione con librerie come Flask-SocketIO [33] consente di gestire facilmente le comunicazioni real-time.

- **Microservizi**: Flask è una scelta comune per lo sviluppo di microservizi all'interno di un'architettura basata su questo stile architetturale. La sua leggerezza e modularità lo rendono ideale per la creazione di servizi autonomi che possono essere distribuiti in modo indipendente.

E' possibile installare Flask semplicemente utilizzando pip [34], il package manager di default di Python: *pip install Flask*

In poche righe è possibile scrivere un'applicazione server come nell'esempio 2.3 sottostante.

Listing 2.3: Applicazione server Flask minimale

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Nel codice precedente per prima cosa è stata importata la classe Flask.

Successivamente creiamo un'istanza di questa classe la cui creazione richiede di specificare il nome del modulo o del pacchetto dell'applicazione indicato in questo caso con la scorciatoia *__name__* adatta alla maggior parte dei casi. Ciò è necessario affinché Flask sappia dove cercare risorse come modelli e file statici.

Il decoratore *@app.route()* è sicuramente l'elemento più noto del framework, e permette di effettuare il bind tra un URL e una corrispondente funzione che sarà chiamata ogni qualvolta sarà effettuata una chiamata su quel URL. Con Flask è possibile definire degli URL complessi dal contenuto parametrico.

Listing 2.4: Esempio di URL con parametro

```
@app.route('/user/<username>')
def profile(username):
    return f'{username}\'s profile'
```

Come si vede nel listato di codice 2.4, è possibile indicare un parametro all'interno di un URL che può essere utilizzato nella funzione corrispondente.

Listing 2.5: Specifica di metodi HTTP a cui è associata la funzione

```
from flask import request

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_the_login()
    else:
        return show_the_login_form()
```

Il decoratore *@app.route()* consente anche di indicare come secondo parametro a quale o a quali metodi HTTP deve essere utilizzata la funzione associata rendendo poi disponibile nell'oggetto *request* il metodo effettivamente utilizzato come è possibile vedere nel listato di codice 2.5.

L'oggetto *request* è utile anche per accedere ai parametri della query URL utilizzando l'attributo *args* in questo modo *key = request.args.get('key', '')*.

Infine, sempre tramite oggetto *request* è possibile leggere il payload di una richiesta http. Con il metodo *get_data()* è possibile leggere il payload come uno stream di byte. Tuttavia è possibile anche leggere i dati formattati in una maniera specifica a patto che nel body si trovino effettivamente in questo formato: con la funzione *get_json()* è possibile, ad esempio, leggere il body di un messaggio in formato json.

Per completare la suite di utility che rende Flask uno dei framework web-service Python più versatile, esiste la possibilità di inviare, a fronte delle richieste effettuate sul server, risposte in vari formati, funzionalità che consente in maniera semplice l'implementazione di API per servizi web e che risponde anche alle necessità dei più classici server web.

Flask utilizza come risposte alle richieste http il valore di ritorno delle funzioni associate ai vari endpoint:

- Se viene restituito un oggetto risposta del tipo corretto, viene restituito direttamente dalla vista.
- Se è una stringa, viene creato un oggetto risposta con quei dati e i parametri predefiniti.
- Se si tratta di un iteratore o di un generatore che restituisce stringhe o byte, viene trattato come una risposta di streaming.
- Se si tratta di un dict o di un elenco, viene creato un oggetto risposta utilizzando jsonify().
- Se viene restituita una tupla, gli elementi nella tupla possono fornire informazioni aggiuntive. Tali tuple devono essere nella forma.(response, status)(response, headers)(response, status, headers).
- Se ciò che viene restituito non corrisponde a nessuna delle opzioni precedenti, Flask presupporrà che il valore restituito sia un'applicazione WSGI valida e la convertirà in un oggetto di risposta.

- Per inviare un file è possible farlo restituendo il risultato della funzione *send_file()* in questo modo: *return send_file(file_path, as_attachment=True)*

Flask include un motore di templating che permette di creare dinamicamente pagine web utilizzando modelli HTML. Il sistema di templating di Flask si basa su Jinja2[35], un motore di templating molto potente e flessibile[36].

Listing 2.6: Esempio di template HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ title }}</title>
    <link rel="stylesheet" href="{{ url_for('static',
        filename='style.css') }}">
</head>
<body>
    <h1>{{ greeting }}</h1>
    <p>Benvenuto su {{ app_name }}!</p>
    <a href="{{ url_for('about') }}">Vai alla pagina "
        About"</a>
</body>
</html>
```

Listing 2.7: Route che utilizza il template

```
@app.route('/')
def index():
    title = 'Pagina di benvenuto'
    greeting = 'Ciao, utente!'
    app_name = 'La mia applicazione Flask'
    return render_template('template.html', title=title
        , greeting=greeting, app_name=app_name)
```

Una volta definito un template di un file ad esempio html, come nel listato 2.6, è possibile inviarlo come risposta completando le variabili mancanti come si vede in nel listato 2.7

url_for() invece, è una funzione di Flask che consente di generare URL dinamicamente. È particolarmente utile quando è necessario creare link nei propri template in modo dinamico, evitando problemi di hardcoding degli URL

Ultimo aspetto di Flask non meno importante, è che si tratta di un framework supportato da una comunità attiva di sviluppatori e una vasta quantità di documentazione online.

2.2.2 OpenTripPlanner

OpenTripPlanner(OTP), già introdotto nella sezione 1.4.2, è un progetto software open-source che offre servizi di informazioni per i tragitti all'interno di una rete stradale e analisi delle reti di trasporto. La forza principale di OTP risiede nella sua capacità di pianificare viaggi attraverso vari modalità di trasporto, e di poter sfruttare dati di informazioni geografiche in diversi formati. OTP è in grado di creare itinerari che combinano diversi modalità di trasporto, come transito, pedonale, bicicletta e auto, utilizzando reti costruite a partire dai dati di OpenStreetMap e GTFS[37].

Proprio per la sua caratteristica open-source esistono diverse versioni custom del progetto. Nel caso di questo progetto di tesi sarà utilizzata una versione customizzata realizzata dal Digital Geography Lab, come sarà poi approfondito nell'apposita sottosezione sulla costruzione del grafor relativo alla rete stradale 3.2.2

In quanto programma Java, OTP deve essere eseguito all'interno di una Java virtual machine (JVM), fornita come parte di Java runtime (JRE) o Java Development Kit (JDK)[38].

OpenTripPlanner è distribuito come un singolo file JAR eseguibile. Si tratta di un JAR "shaded" contenente tutte le altre librerie necessarie per

il funzionamento di OTP ed è disponibile attualmente nel repository Maven Central[38].

Per costruire un grafo corrispondente alla rete di trasporti pubblici e al reticolo urbano per una certa area interessata sarà necessario fornire per la prima i dati GTFS e per la seconda i data OpenStreetMap corrispondenti. Il formato di dati OSM sarà meglio approfonditi nella sezione . I dati OSM vengono processati da OTP solo se formattati nel formato PBF, un formato di serializzazione binaria utilizzato per la memorizzazione e la condivisione dei dati delle mappe in OpenStreetMap, introdotto per ottimizzare le dimensioni dei dati e migliorare l'efficienza nella gestione delle mappe in alternativa al più classico formato XML[39].

Tramite il jar OTP è possibile svolgere principalmente due funzioni: creare un grafo utilizzando i file OSM e GTFS, e lanciare un server predefinito che rispode a delle API utilizzando il grafo generato.

Per creare un grafo e salvarlo in un file è possibile lanciare il jar utilizzando i parametri della riga di comando *–build* e *–save*.

Il file prodotto sarà un file chiamato *obj* che il server di OTP può caricare utilizzando il flag *–load*.

OTP da la possibilità di creare un grafo e, in un unico passaggio, avviare immediatamente un server, senza salvarlo su disco. Il comando per farlo è con il comando *java -Xmx2G -jar otp-2.4.0-shaded.jar –build –serve /home/username/otp*, dove dove */home/username/otp* dovrebbe essere la directory in cui inserire la configurazione e i file di input OSM e GTSF.

La costruzione del grafo stradale può richiedere molto tempo. Visto che i dati sui trasporti pubblici cambiano in genere più frequentemente rispetto ai dati stradali, OpenTripPlanner dà la possibilità di creare prima il grafo stradale, e quindi di sovrapporre i dati sui trasporti pubblici per creare il grafo finale.

Le API messe a disposizione dal server di OTP forniscono diverse funzionalità per accedere al grafo e alle sue informazioni, ma soprattutto permettono di pianificare e calcolare percorsi e rotte all'interno del grafo. Le API

del server OTP infatti permettono di di configurare vari aspetti del processo di routing, come i timeout per il routing stradale e la durata massima del trasferimento. Queste configurazioni influenzano il modo in cui OTP gestisce le richieste di routing, ottimizzando sia le prestazioni che i risultati. Possono essere modificati i parametri dell'algoritmo RAPTOR utilizzato per il calcolo dei percorsi: ci sono vari parametri che possono essere impostati per affinare la ricerca di itinerari, come il numero massimo di trasferimenti e il tempo di attesa massimo. Altri parametri avanzati includono la soglia per la ricerca binaria degli orari dei viaggi programmati, la dimensione del pool di thread per la ricerca e la dimensione massima della cache per i trasferimenti pre-calcolati.

Nella versione 2 OTP implementa un interfaccia server GraphQL, che sta soppiantando la vecchia interfaccia RESTful. Di seguito l'esempio di una richiesta al server OTP in cui viene chiesto di calcolare un percorso per arrivare da Via Indipendenza alla stazione di Bologna in meno di 40 minuti prendendo non più di 2 autobus.

2.3 Formato dei dati

In questa sezione saranno descritti e approfonditi il formato dei dati utilizzati, scambiati e salvati dal sistema. Ogni sottosezione sarà relativa a uno specifico formato di dati.

2.3.1 OpenStreetMap

I dati OSM sono già stati introdotti nella sezione1.4.2: si tratta di un formato di dati geospaziali open-source che viene utilizzato per memorizzare informazioni geografiche dettagliate, come strade, edifici, punti di interesse e altro ancora. OSM è ampiamente utilizzato per creare mappe e dati geografici in tutto il mondo ed è alla base di molti servizi di mappe online e offline. Nell'ambito di questa tesi saranno meglio dettagliati nella sezione 2.3.1.

I dati OpenStreetMap sono organizzati in modo gerarchico e strutturato per rappresentare il mondo fisico in modo dettagliato. Ecco alcuni dei principali elementi dei dati OSM:

- Nodi (Nodes): I nodi sono i punti geografici elementari definiti da coppie di coordinate latitudine e longitudine. Ogni nodo ha un identificatore unico e può contenere tag che descrivono le caratteristiche del punto, come nome o attributi specifici.

- Vie (Ways): Le vie sono sequenze ordinate di nodi che rappresentano strade, fiumi, binari ferroviari e altre caratteristiche lineari. Possono avere attributi che specificano il tipo di via (ad esempio, strada principale o autostrada) e altre informazioni pertinenti.

- Relazioni (Relations): Le relazioni sono utilizzate per collegare più nodi e/o vie in una struttura più complessa. Possono essere utilizzate per rappresentare caratteristiche geografiche complesse, come incroci stradali, aree di utilizzo misto e confini amministrativi.

- Tag: I tag sono coppie chiave-valore associate a nodi, vie e relazioni. Questi tag forniscono dettagli sulle caratteristiche geografiche, come il nome di una strada, il tipo di edificio o le restrizioni di velocità.

I dati OpenStreetMap [40] sono memorizzati in file di testo strutturati in formato XML o binario, in formato PBF (Protocolbuffer Binary Format). Il formato PBF è una rappresentazione binaria altamente efficiente dei dati OSM, mentre il formato XML è più leggibile per l'utilizzatore umano.

Un esempio di dato OSM in formati XML lo si può trovare nel listato 2.8 sottostante. Nel file viene specificata la versione di OSM a cui fa riferimento il file e vengono definiti due nodi e una strada con le loro caratteristiche.

Listing 2.8: Dati OSM in formato XML

```
<osm version="0.6" generator="OSM editor">
  <node id="1" lat="52.5200" lon="13.4050">
```

```
    <tag k="name" v="Brandenburg Gate" />
  </node>
  <node id="2" lat="52.5123" lon="13.4073">
    <tag k="name" v="Reichstag Building" />
  </node>
  <way id="3">
    <nd ref="1" />
    <nd ref="2" />
    <tag k="highway" v="primary" />
  </way>
</osm>
```

2.3.2 GeoJson

Il formato GeoJSON è un formato aperto e leggero per la rappresentazione di dati geospaziali. La sua ideazione avviene nel 2007[41] ed è diventato uno standard aperto ampiamente utilizzato nello sviluppo di software GIS e nell'ambito delle applicazioni web e mobili basate su mappe.

Prima dell'introduzione di GeoJSON, i dati geospaziali venivano memorizzati in formati più complessi e pesanti, come Shapefile o GML (Geography Markup Language). Questi formati richiedevano dei parser specializzati e risultavano difficili da gestire, specialmente in ambiente web e mobile. Proprio la necessità di un formato più semplice e accessibile ha portato alla definiizione del formato GeoJSON.

Geojson è basato sulla sintassi JSON (JavaScript Object Notation), il quale è un formato di dati molto comune e ampiamente supportato, che ha anche il pregio di risultare particolarmente semplice per la lettura di un utente umano.

In ambito web Mapbox, Google Maps e Leaflet, supportano GeoJSON per la visualizzazione di dati geospaziali per visualizzare punti di interesse, confini geografici, informazioni GPS e altro ancora.

Un file GeoJSON è costituito da un oggetto JSON che rappresenta un singolo oggetto geografico o una collezione di oggetti geografici. I principali tipi di oggetti geografici supportati da GeoJSON sono:

- Punti (*Point*): Rappresentano un singolo punto nello spazio utilizzando le coordinate geografiche latitudine e longintudine. Nei sistemi GIS possono essere utilizzati per rappresentare punti di interesse.

Listing 2.9: Esempio di oggetto "Point"

```
{
  "type": "Point",
  "coordinates": [100.0, 0.0]
}
```

- Linee (*LineString*): Rappresentano una sequenza ordinata di punti sempre utilizzando latitudine e longitudine. Possono essere utilizzati per rappresentare strade e corsi d'acqua.

Listing 2.10: Esempio di oggetto "LineString"

```
{
  "type": "LineString",
  "coordinates": [
    [100.0, 0.0],
    [101.0, 1.0]
  ]
}
```

- Poligoni (*Polygon*): Rappresentano una superficie poligonale chiusa in cui il punto iniziale e finale corrispondono. I Poligoni possono essere utilizzati per rappresentare confini di stati o città e la superficie di laghi e mari, nonchè aree di interesse specifiche.

Listing 2.11: Esempio di oggetto "Polygon"

```
{
  "type": "Polygon",
  "coordinates": [
    [
      [100.0, 0.0],
      [101.0, 0.0],
      [101.0, 1.0],
      [100.0, 1.0],
      [100.0, 0.0]
    ]
  ]
}
```

- Feature: Rappresentano un oggetto geografico con proprietà associate.

Listing 2.12: Esempio di oggetto "Feature"

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [100.0, 0.0]
  },
  "properties": {
    "name": "Luogo di interesse"
  }
}
```

- Feature Collection: Rappresenta una collezione di oggetti Feature.

Listing 2.13: Esempio di oggetto "FeatureCollection"

```
{
  "type": "FeatureCollection",
```

```
    "features": [
      {
        "type": "Feature",
        "geometry": {
          "type": "Point",
          "coordinates": [100.0, 0.0]
        },
        "properties": {
          "name": "Luogo 1"
        }
      },
      {
        "type": "Feature",
        "geometry": {
          "type": "Point",
          "coordinates": [101.0, 1.0]
        },
        "properties": {
          "name": "Luogo 2"
        }
      }
    ]
  }
```

2.3.3 GraphML

[42]GraphML e un formato di file utilizzato per rappresentare grafi diretti o non diretti utilizzando il formato GraphML. Il formato GraphML è un formato di file basato su XML (è quindi è un file di testo) progettato per rappresentare informazioni strutturate sui grafi. È utilizzato principalmente in contesti di teoria dei grafi e nella visualizzazione di dati legati a grafi,

come reti sociali, grafi di strade o altri tipi di grafi. Di seguito le principali caratteristiche di questo formato di file:

- Struttura basata su XML: Il formato GraphML utilizza la struttura XML per organizzare i dati del grafo risultando facilmente leggibili anche da un utente umano.
- Grafi diretti o non diretti: Il formato GraphML è flessibile e consente di rappresentare grafi diretti (in cui le connessioni tra i nodi hanno una direzione) o grafi non diretti (in cui le connessioni sono bidirezionali).
- Nodi e archi: I grafi in formato GraphML sono composti da nodi (vertici) e archi (connettori tra i nodi). Ogni nodo può avere attributi associati, come un identificatore univoco o altre informazioni pertinenti. Gli archi possono anch'essi avere attributi, come un peso o una direzione.
- Supporto per attributi: Una caratteristica potente del formato GraphML è la capacità di allegare attributi personalizzati a nodi, archi o al grafo nel suo complesso. Questi attributi possono essere utilizzati per memorizzare informazioni aggiuntive sul grafo o per scopi specifici dell'applicazione.

Listing 2.14: Esempio di file GraphML

```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
    <graph id="G" edgedefault="directed">
        <node id="n0">
          <data key="v_geom">POINT (688060.0508902381
              4933195.923646035)</data>
        </node>
        <node id="n1">
          <data key="v_geom">POINT (683158.152722773
              4930624.373134296)</data>
```

```
        </node>
        <node id="n18395">
          <data key="v_geom">POINT (687999.63690394
              4933235.873950969)</data>
        </node>
        <edge source="n0" target="n18395">
          <data key="e_ii">852</data>
          <data key="e_geom">LINESTRING
              (688060.0508902381 4933195.923646035,
              688060.5190240045 4933205.564804011,....,
              4933235.873950969)</data>
          <data key="e_geom_wgs">LINESTRING
              (11.366705600000001 44.5275684, 11.366715
              44.527655, 11.366712 44.527723, ...,
              11.3659605 44.5279435)</data>
          <data key="e_l">89.6073</data>
          <data key="e_b_st">0</data>
          <data key="e_b_ab">1</data>
          <data key="e_bsf">3.43</data>
          <data key="e_uv">(0, 18395)</data>
          <data key="e_iw">629</data>
        </edge>
  </graph>
</graphml>
});
```

Nel listato 2.14 è rappresentato un grafo con tre nodi, *n0*, *n1*, *n18395*, e un arco che va da *n0* a *n18395*. Per ogni nodo è stata definito un tag *data* con identificativo *key="v_geom"* che rappresenta una caratteristica del nodo: in questo caso è stata rappresentata la sua geometria come punto all'interno di un sistema di coordinate. Per l'arco invece sono state rappresentate diverse proprietà sempre utilizzando il tag *data* e l'attributo *key*.

2.3.4 JSON Web Token (JWT)

JSON Web Token (JWT) è uno standard aperto (RFC 7519) che definisce un modo compatto e autonomo per trasmettere informazioni tra le parti come un oggetto JSON firmato digitalmente. Queste informazioni possono essere ritenute verificate e affidabili essendo firmate digitalmente: i JWT possono quindi essere utilizzati come strumenti per l'autenticazione all'interno dei sistemi informativi.

Lo standard JWT è stato introdotto per la prima volta nel 2010 come specifica di elaborazione dei token di accesso in OAuth 1.0, ma nel tempo, JWT è diventato uno standard indipendente da OAuth e ha trovato ampio utilizzo in una varietà di scenari di sicurezza e autenticazione, in particolare viene utilizzato per gestire l'autenticazione e l'autorizzazione tra client e server in modo stateless e scalabile. Per fare questo in genere un JWT può essere incluso nell'intestazione di una richiesta HTTP, tipicamente nell'header *Authorization*.

Un JWT è composto da tre parti separate da punti, nel formato *header.payload.signature*. Il *header* contiene il tipo di token (*type*) e l'algoritmo di firma (*alg*). Il *payload* contiene le informazioni desiderate, chiamate affermazioni (claims). La *signature* è il risultato della codifica Base64 dell'header e del payload, concatenati con un punto, e quindi firmati con una chiave segreta.

La firma viene generata applicando un algoritmo crittografico (HMAC o RSA) al token codificato e alla chiave segreta. Questa firma viene quindi aggiunta al token. **??**

2.4 Deployment

In questa sezione verranno descritte le tecnologie che saranno impiegate per il deployment del servizio Ecosister lato back-end in particolare il sistema di containerizzazione Docker e lo strumento Docker-compose.

2.4.1 Docker

Docker[43] è un progetto open-source, supportato principalmente dalla società Docker Inc., che consente la creazione e l'utilizzo di container Linux.

[2] Come gli altri servizi di containerizzazione, Docker non è una modalità innovativa di virtualizzazione. Tramite la virtualizzazione infatti è possibile eseguire più sistemi operativi sulla stessa macchina, utilizzando un'astrazione della stessa detta *hypervisor*. Quando si parla di containerizzazione ci si riferisce invece a un insieme di processi isolati in esecuzione però sulla stessa macchina. Nell'ambiente isolato in cui operano i processi vengono installate tutte le dipendenze utili a creare un contesto che simula quello di un sistema operativo più tutte quelle necessarie per l'esecuzione degli applicativi che sono containerizzati. La specifica di quali componenti vanno installati è specificata in un immagine. I container quindi dividono lo stesso kernel del sistema operativo a differenza delle virtual machine.

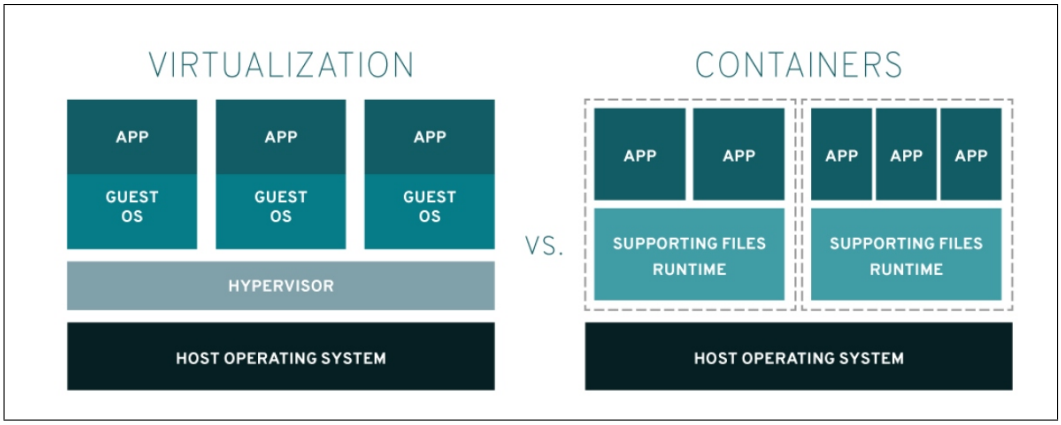

Figura 2.1: Differenza tra virtualizzazione e container [2]

Grazie all'utilizzo di Docker:

- esiste la possibilità creare un ambiente sempre uguale dove eseguire i servizi. Indipendente dal sistema operativo degli sviluppatori e da come sarà effettivamente implementato: il servizio utilizzerà un set preciso di risorse invariante nei diversi contesti

- l'utilizzo dei Dockerfile, in cui vengono specificati tutti i parametri e le azioni per creare il contesto di esecuzione, rende la creazione delle immagini molto semplici
- la distribuzione dei servizi diventa più rapida e più facile
- si evitano la pesantezza e la complessità dell'installazione e del settaggio delle virtual machine
- è possibile accedere alla piattaforma di Dockerhub[44], il servizio di repository remoto di immagini Docker. L'utilizzo del servizio consente una distribuzione semplice delle immagini, di cui viene mantenuto anche uno storico delle diverse versioni, e incentiva un processi di automazione per quanto riguarda la creazione e la distribuzione delle immagini.

2.4.2 Docker-compose

Docker-compose è strumento per semplificare il deployment di applicazioni multi-container Docker. La caratteristica principale di questo strumento è quella di ridurre considerevolmente il numero di comandi necessari per effettuare la creazione e l'avvio di più container, operazioni che se effettuate in maniera classica necessitano di diversi comandi da eseguire per ogni singolo container di cui effettuare il deployment.

Con Docker-compose, è possibile definire l'intera struttura dell'applicazione, compresi i servizi, i volumi, le reti e le variabili d'ambiente, in un unico file di configurazione YAML, in modo completamente dichiarativo.

Listing 2.15: Docker-compose file del progetto Ecosister

```
version: "3.1"

services:
  routes-service:
    image: sdg97/ecosister-routes-service:latest
```

```
    ports:
      - "5000:5000"
    depends_on:
      - aq-service
    environment:
      AQ_SERVICE_INTERPOLATE_API: "http://aq-service
          :5000/interpolate"

  aq-service:
    image: sdg97/ecosister-aq-service:latest
    ports:
      - "5001:5000"

  users-service:
    image: sdg97/ecosister-users-service:latest
    ports:
      - "5003:5000"
    depends_on:
      - db
    environment:
      PROD: "true"
      MONGO_URI: "mongodb://ecosister-username:
          ecosister-password@db:27017/?authMechanism=
          DEFAULT" # Adjust as needed
      SECRET_KEY: "your-prod-secret"
      JWT_SECRET_KEY: "your-prod-jwt-secret"

  db:
    image: mongo:4.0.27
    restart: always
    environment:
```

```
      MONGO_INITDB_ROOT_USERNAME: ecosister-username
      MONGO_INITDB_ROOT_PASSWORD: ecosister-password
    volumes:
      - ./db_init:/docker-entrypoint-initdb.d
      - ./db_init/data:/tmp/data
      - ./db_data:/data/db # Persistent data storage
    ports:
      - "27017:27017"

  nginx:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./build:/usr/share/nginx/html
    depends_on:
      - routes-service
      - aq-service
      - users-service
      - db
```

Il listato di codice 2.15 rappresenta il contenuto di un file docker-compose.yml per il deployment di quattro servizi. Da questo esempio si evince che, come si diceva prima, per ogni sistema è possibile esprimere una lista di servizi che lo compongono. Per ogni servizio è possibile specificare diversi attributi come l'immagine docker del servizio da dover utilizzare, tag *image*, il mapping dei volumi utilizzati dal container e delle porte utilizzando rispettivamente i tag *volumes* e *ports*, assegnare i valori alle variabili d'ambiente del container con il tag *environment*, e infine specificare i rapporti di dipendenza tra i container in modo da stabilire un corretto ordine di avvio di esecuzione utilizzando il tag *depends_on*.

Capitolo 3

Sviluppo dell'elaborato di progetto

In questo capitolo sarà descritto nel dettaglio il sistema che è stato sviluppato per rispondere ai requisiti descritti nella sezione 1.5. Verrà descritta l'architettura generale del sistema, poi si scenderà nel dettaglio della descrizione di ogni singolo componente a partire dai servizi realizzati fino all'applicazione front-end.

3.1 Architettura generale del sistema

In questa sezione verrà descritta l'architettura generale del sistema.

3.1.1 Architettura a microservizi

L'architettura del sistema Ecosister è stata pensata come un insieme di microservizi

Lo stile architettonico a microservizi è un approccio allo sviluppo di una singola applicazione come una suite di piccoli servizi, ciascuno in esecuzione su un processo differente e messi in comunicazione tra loro attraverso meccanismi leggeri, spesso un'API di risorse HTTP.

Difficile trovare una definizione formale dello stile architettonico dei microservizi. Tuttavia, esistono alcuni principi, espressi nella trattazione di Sam Newman *Building Microservices* [45], che definiscono alcune caratteristiche, buone norme di progettazione, e di gestione, per poter costruire opportunamente un sistema con questa prospettiva, evidenziando anche alcuni vantaggi del suo utilizzo.

- **Business focused** Secondo questo principio ogni microservizio deve essere modellato in modo tale che la sua logica applicativa incapsuli uno ed un solo aspetto del modello di business. Il dominio applicativo rappresentato da un singolo microservizio è detto **Bounded Context**. Adottando questo principio si ottiene come risultato:
  - *loose coupling* tra i diversi microservizi.
  - interfacce di comunicazioni più stabili rispetto a quelle modellate intorno a concetti tecnici e che riflettono al meglio il dominio applicativo

- **Incapsulamento** Per massimizzare la capacità di un microservizio di evolvere indipendentemente è necessario nascondere ogni dettaglio implementativo. Nel contesto dei microservizi è importante studiare e osservare quali sono le responsabilità del Bounded Contex definendo opportunamente le risorse che devono essere gestite dal microservizio e in quale modo devono essere utilizzate e rese disponibili all'esterno. L'accesso e l'utilizzo delle risorse del microservizio può avvenire solo se concessa dall'interfaccia che il microservizio presenta. Nascondere i dettagli implementativi ha ovviamente dei vantaggi noti, come la possibilità di modificare totalmente le tecnologie con cui è implementato il servizio senza modificare gli altri elementi del sistema. Nel contesto dei microservizi questo porta a dei risultati anche dal punto di vista del deployment perchè è possibile modificare il servizio quando questo è già in funzione.

- **Decentralizzazione** La decentralizzazione implica l'isolamento del microservizio nella sua totalità, non solo dal punto di vista concettuale, come si è visto precedentemente con il concetto di Bounded Contex, ma anche dal punto di vista:

  1. **tecnico**: i microservizi non devono avere né basi di dati né librerie in comune.

  2. della **comunicazione**:la comunicazione deve essere agnostica utilizzando un messaggio dal formato supportato dalla maggior parte delle tecnologie in modo da non vincolare in nessun modo le scelte implementative.

  3. del **deployment**: ogni microservizio è eseguito in un processo a parte, utilizzando delle risorse allocate unicamente per le sue funzionalità. Questo significa il sistema deve essere realizzato come se ogni servizio, nel caso più estremo, debba essere distribuito su macchine differenti.

  Applicando questo principio si ottene come vantaggio una loose coupling dei microservizi non solo dal punto di vista concettuale ma anche effettivo

Riassumendo quindi l'architettura a microservizi ha come vantaggio quello di poter esprimere anche dal punto di vista architetturale la suddivisione dei vari aspetti del dominio applicativo. Inoltre permette la realizzazione di un sistema resiliente e modulare da un punto di vista concettuale, tecnico e di deployment.

3.1.2 Architettura del sistema

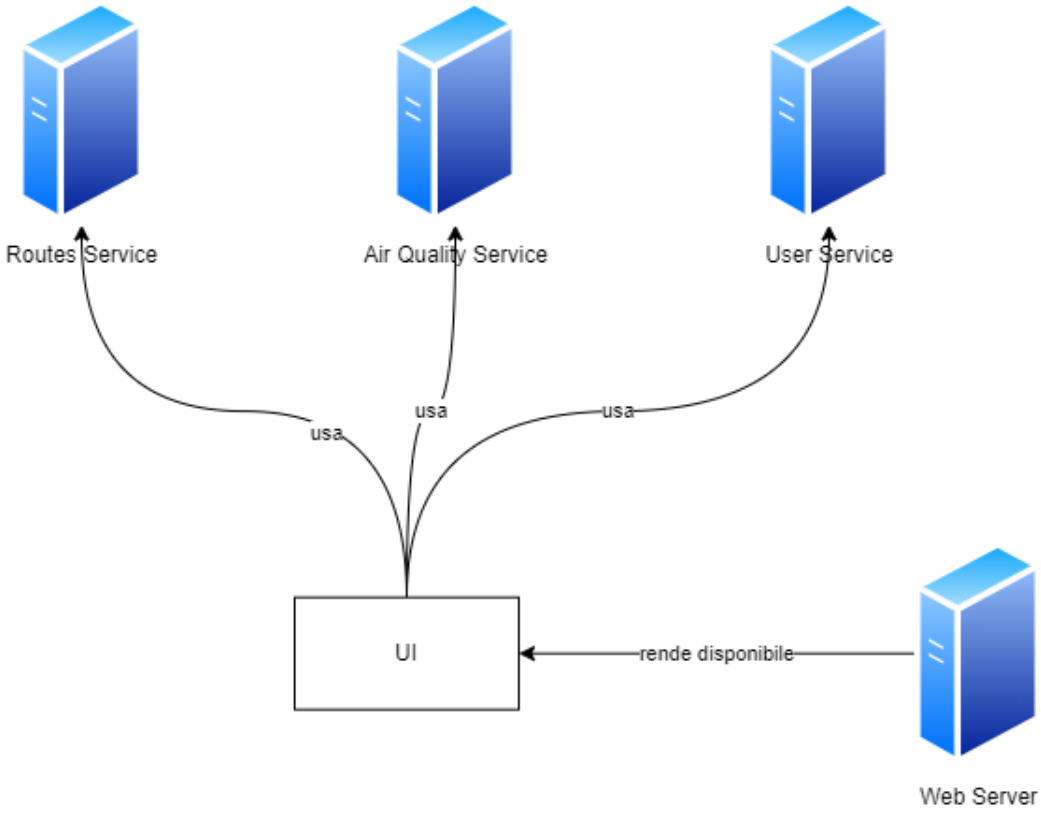

Figura 3.1: Architettura del sistema Ecosister

L'architettura del sistema Ecosister rappresentata in figura 3.1 si compone principalmente di tre componenti più uno:

- **Ecositer Routes Service**: si tratta del servizio che si occupa di gestire le informazioni sul servizio stradale della città. Nello specifico questo servizio si occuperà di:
  - elaborare dati sulla rete stradale al fine di generare un grafo che la rappresenti.
  - tenere aggiornate le informazioni sul grafo, in particolare integrando le informazioni relative alla qualità dell'aria

  - rendere disponibile le informazioni relative ai dati sulla qualità dell'aria associati a ogni strada della città
  - calcolare il percorso minimo tra un punto iniziale e un punto finale, tenendo conto dei dati stradali e dei dati sulla qualità dell'aria

- **Ecositer Air Quality Service**: il si occuperà di elaborare dati mock-up sulla qualità dell'aria, che siano verosimili, abbiano un certo grado di casualità e siano mutevoli nel tempo. In particolare il servizio dovrà:
  - definire e rendere disponibili dati sui sensori mock-up, tra cui posizione all'interno del perimetro di interesse del progetto, la città di Bologna, e identificativo del sensore.
  - per ogni sensore rendere disponibili dei dati sulla qualità dell'aria verosimili, casuali e mutevoli nel tempo
  - offire una funzionalità di interpolaizione tale per cui una volta fornite la latitudine e la longitudine di un punto sia possibile calcolare i dati sulla qualità dell'aria presenti in quel punto interpolando i dati generati per ogni sensore mock-up.

- **Ecosister User Service**: è il servizio che si occuperà di gestire i dati degli utenti che utilizzando il servizio di Ecosister. Dovrà permettere agli utenti di registrarsi attraverso il proprio username e una password scelta da loro. Dovrà consentire di tenere traccia di alcune informazioni scelte dagli utenti. In particolare dovrà:
  - permettere di salvare e modificare una posizione all'interno della mappa come *home*
  - permettere di salvare e modificare una posizione all'interno della mappa come *work*
  - permettere di salvare una posizione di partenza e di arrivo in una lista di preferiti. Dovrà consentire poi di rimuovere le posizioni già salvate nella lista.

- **Ecosister UI**: è l'applicazione web che permette a un utente di interfacciarsi con il servizio di Ecosister. L'interfaccia dell'applicazione sarà principalmente costituita da una mappa interattiva che presenterà tre diverse colorazioni:
  - una colorazione *city*, simile a quella dei più popolari servizi di navigazione.
  - una colorazione *heatmap* che sfutta il servizio Ecosister Air Quality Service, per interpolare i valori della mappa e associargli un'opportuna colorazione in base all'Air Quality Index Europeo.
  - una colorazione *street-by-street* che permette di associare a ogni strada un colore corrispondente all'Air Quality Index Europeo. Per fare ciò sfrutterà la funzionalità messa a disposizione dal Servizio Ecosister Routes Service, che associa ad ogni strada il suo corrispondente Air Quality Index interpolato grazie al servizio Ecosister Air Quality Service.

  Sulla mappa saranno presenti dei markers che rappresenteranno i sensori, la loro posizione e i dati registrati dai sensori real-time. I dati rappresentati, sia relativi ai sensori che alla qualità dell'aria, saranno richiesti al servizio Ecosister Air Quality Service.

  Sulla mappa potranno essere selezionati i punti di partenza e di destinazione per calcolare i propri percorsi utilizzando le funzionalità messe a disposizione da Ecosister Routes Service.

  Per ogni percorso possibile calcolato sarà evidenziato, sia sulla mappa che con opportune didascalie, la lunghezza del percorso, il tempo di percorrenza, il periodo di esposizione a un certo livello di inquinamento sia in termini di chilometri da percorrere che in termini di tempo.

  L'applicazione permetterà all'utente di registrarsi, effettuare il login e il logout sfruttando le funzionalità di Ecosister User Service. Sempre utilizzando questo servizio sarà possibile selezionare un punto sulla mappa

come *home*, selezionare un punto sulla mappa come *work* e, una volta selezionato un punto di partenza e di destinazione sarà possibile salvare il percorso in una lista di percorso preferiti. La lista sarà resa disponibile all'utente in modo che possa selezionare le ricerche e calcolare il percorso sulla mappa, e infine eliminare le coppie partenza-destinazione indesiderate.

- L'ultimo componente del sistema sarà il web server che metterà a disposizione l'applicazione web

3.2 Ecosister Routes Service

In questa sezione verrà descritto nel dettaglio il funzionamento del servizio Routes Service di Ecosister. In primo luogo verrà ripreso e approfondito il suo bounded-context introdotto nella sottosezione 3.1.2. Successivamente si approfondirà come come vengono effettuati il processo per creare il grafo, il calcolo dei percorsi minimi e l'aggiornamento del grafo con i dati relativi alla qualità dell'aria.

3.2.1 Bounded Context

Il servizio Ecosister Routes Service si occupa di:

- elaborare dati sulla rete stradale al fine di costruire un grafo che la rappresenti. Per ogni nodo e arco del grafo saranno memorizzate diverse proprietà utili al calcolo dei percorsi minimi e alla rappresentazione della rete stradale, tra cui, per ogni nodo, coordinate del nodo ed eventuale etichetta nel caso sia un punto di interesse; per ogni arco, nodo di partenza e nodo di arrivo,geometria dell'arco, etichetta per individuare a quale strada appartiene l'arco, lunghezza in metri del percorso stradale rappresentato dall'arco e altre informazioni relative alla percorribilità del percorso rappresentato dall'arco.

- calcolare il percorso minimo tra un punto iniziale e un punto finale, tenendo conto della velocità di percorrenza del percorso e dei dati relativi alla qualità dell'aria. Sarà possibile modificare le opzioni del percorso specificando se il tragitto sarà percorso a piedi o in bicicletta.

- tenere aggiornate le informazioni sul grafo, in particolare integrando le informazioni relative alla qualità dell'aria, memorizzando i dati sugli inquinanti rilevati e l'AirQualityIndex associato.

- fornire un interfaccia RESTful che permetta di accedere ai dati relativi alla qualità dell'aria associati a ogni strada della mappa rappresentata, e che permetta di richiedere il calcolo di un percorso in base ai parametri previsti.

Si noti che il servizio nella sua istanziazione per il software Ecoservice svolge le sue funzionalità utilizzando i dati sulla rete stradale di Bologna. Tuttavia il software è stato sviluppato per poter elaborare anche i dati relativi ad altre città senza dover adottare nessun accorgimento se non quello di effettuare la stessa procedura di costruzione del grafo anche per questi dati.

Ecosister Routes Service è stato sviluppato in Python sia per lo sviluppo del servizio web, utilizzando il framework Flask, che per quanto riguarda lo sviluppo delle funzioni di utilità e della costruzione del grafo: in questo caso sono state utilizzate apposite librerie per la manipolazione di dati Geospaziali e per la manipolazione dei grafi.

Nelle successive sottosezioni saranno descritti approfonditamente tutti i punti dell'elenco.

3.2.2 Costruzione del grafo

Prima di poter avviare il servizio è necessario effettuare il processo che porta alla realizzazione del grafo che rappresenta la rete stradale, su cui sarà poi possibile calcolare i percorsi minimi e associare i dati sulla qualità dell'aria da rendere disponibili.

Il processo di costruzione del grafo prevede la trasformazione del file OpenStreetMap, che nel caso di questo specifico progetto sarà quello relativo alla città di Bologna, il cui formato di file è descritto nella sottosezione 2.3.1, in un grafo in formato GraphML, formato descritto nella sezione 2.3.3. Il grafo così ottenuto sarà poi manipolabile tramite la libreria *igraph* [46], per sviluppare le funzionalità previste.

Per supportare il processo di build del grafo, eccetto per il download dei dati OSM relativi all'area di interesse, è stato scritta un apposita suite di script, in cui per ogni passaggio del processo è stato scritto uno script diverso al fine di lasciare aperta la possibilità di ulteriori manipolazioni intermedie del grafo in vista di sviluppi futuri. Gli script possono essere lanciati utilizzando lo script *graph_build_main.py* che presenta gli script nell'ordine di esecuzione corretto e da all'utente la possibilità di selezionare quale di questi lanciare.

Il primo passaggio da effettuare è **reperire il file OSM** in formato *.pbf* utile per essere elaborata con OpenTripPlanner. Per il progetto di tesi i dati utilizzati sono stati presi dal portale *Estratti Open Street Map Italia* [47]. La scelta del portale è stata effettuata per la funzionalità di poter reperire i dati con un filtraggio specifico sul comune di interesse, per la garanzia di aggiornamento giornaliero dei dati, e per il fatto che il progetto è sostenuto direttamente da OpenStreetMap Italia [48].

Una volta fatto questo, a partire dal file OSM viene generato un file GeoJSON che rappresenta il perimetro dell'aria di Bologna, utilizzando le funzionalità della libreria *Osmium* [49], Pandas [50] e GeoPandas [51], come mostrato nel listato di codice 3.1.

Nel punto *(1)* utilizzando la libreria Osmium e sfruttando la proprietà *admin_level* vengono estratti i dati del perimetro che successivamente vengono convertiti nel perimetro, come si vede nel punto *(2)*.

Listing 3.1: "Script per la generazione del GeoJSON rappresentante i confini"

```
pbf_file_from_osm = 'graph_build/common/bologna.pbf'
output_geojson = 'graph_build/common/bologna.geojson'
```

```
def create_borders_from_xml():
    import osmium
    import shapely.wkb
    import pandas as pd
    import geopandas as gpd

    def merge_two_dicts(x, y):
        z = x.copy()   # start with keys and values of
           x
        z.update(y)    # modifies z with keys and
           values of y
        return z

    class AdminAreaHandler(osmium.SimpleHandler):
        def __init__(self):
            osmium.SimpleHandler.__init__(self)

            self.areas = []
            self.wkbfab = osmium.geom.WKBFactory()
        ## (1) Lettura del perimetro del confine dal
           file PBF
        def area(self, a):
            if "admin_level" in a.tags:

                wkbshape = self.wkbfab.
                   create_multipolygon(a)
                shapely_obj = shapely.wkb.loads(
                   wkbshape, hex=True)

                area = { "id": a.id, "geo": shapely_obj
                    }
```

```
                area = merge_two_dicts(area, a.tags)

                self.areas.append(area)

    handler = AdminAreaHandler()

    # start data file processing
    handler.apply_file(pbf_file_from_osm, locations=
        True, idx='flex_mem')

    # (2) Lettura conversione del file GeoJSON
    df = pd.DataFrame(handler.areas)
    gdf = gpd.GeoDataFrame(df, geometry="geo")

    # Unisci i poligoni in un unico poligono
    merged_polygon = gdf.unary_union

    # Crea un nuovo GeoDataFrame con il poligono unito
    gdf_merged = gpd.GeoDataFrame(geometry=[
        merged_polygon])

    # Esporta il GeoDataFrame in GeoJSON
    gdf_merged.to_file(output_geojson, driver='GeoJSON
        ')

    print(f"Conversione completata. File GeoJSON
        salvato in: {output_geojson}")

def main():
    create_borders_from_xml()
```

```
if __name__ == "__main__":
    main()
```

Successivamente è stata utilizzata una versione custom di OpenTripPlanner [52], realizzata dal Digital Geography Lab [53], in cui è stata modificata la funzione di building del grafo di OTP, descritta nella sottosezione 1.4.2, in modo che anziché produrre un file *.obj* utilizzabile poi solo da OTP, produca due file *.csv*, uno rappresentante i nodi del grafo, di cui un esempio nel listato 3.2, e uno rappresentante gli archi del grafo di cui un esempio nel listato 3.3.

Listing 3.2: Esempio del file CSV relativo ai nodi

```
id_otp;name_otp;vertex_class;traversable_walking;
    traversable_biking;label;traffic_light;free_flowing;
    x;y;geometry
16172;osm:node:255330402;OsmVertex;True;True;osm:node
    :255330402;False;False
    ;11.366705600000001;44.5275684;POINT
    (11.366705600000001 44.5275684) ...
```

Listing 3.3: Esempio del file CSV relativo agli archi

```
id_otp;name_otp;node_orig_id;node_dest_id;length;
    edge_class;street_class;is_stairs;is_no_thru_traffic
    ;permission;allows_walking;allows_biking;
    traversable_walking;traversable_biking;
    bike_safety_factor;geometry
13840;service road;8192;8196;22.376;StreetEdge;5;False;
    False;ALL;True;True;True;True;1.8333333;LINESTRING
    (11.324414200000001 44.5055248, 11.3243521
    44.505328500000005) ...
```

Per ogni nodo del grafo vengono rappresentate diverse proprietà di cui quelle rilevanti per le funzionalità del servizio sono:

- *id_otp*: identificativo assegnato al nodo da OTP.

- *name_otp*: nomenclatura del nodo, un parametro utile soprattutto per identificare dei punti di interesse.
- *traversable_walking*: indica se il punto è attraversabile a piedi
- *traversable_biking*: indica se il punto è attraversabile in bici
- *geometry*: la geometria del dato, in questo caso tutti i nodi sono rappresentati da dei punti, con le coordinate geografiche espresse in WGS4326, sistema di coordiante descritto nella sottosezione 1.2

Per ogni arco del grafo vengono rappresentate invece:

- *id_otp*: identificativo assegnato all'arco da OTP.
- *name_otp*: nomenclatura dell'arco, un parametro utile soprattutto per identificare il nome della strada a cui appartiene l'arco. É importante notare che una via è composta da più archi.
- *node_origin_id*: *id_otp* del nodo di origine dell'arco.
- *node_dest_id*: *id_otp* del nodo di destinazione dell'arco.
- *length*: lunghezza in metri del tratto di strada rappresentato dall'arco.
- *is_no_thru_traffic*: proprietà dal valore booleano descrive se la strada è una strada utilizzata per il transito di auto oppure no
- *geometry*: la geometria del dato, in questo caso tutti gli archi sono sono rappresentati con delle linee, espresse come una sequenza di coordinate in formato WGS4326 1.2

Una volta ottenuti i file CSV, questi possono essere elaborati per generare il file GraphML.

Per operare la generazione del grafo vengono eseguiti questi passaggi:

1. lettura del GeoJSON rappresentante i confini dell'area di Bologna e riproiezione in formato EPSG 32632 [54]. Questa trasformazione è importante perché consente di trasformare le coordinate in un sistema che rappresenta un geoide che si adatta meglio alla conformazione Italia del nord e quindi alla superficie di Bologna.

2. lettura dei dati sui nodi dal file CSV e riproiezione della geometria dei nodi dal sistema di coordinate EPSG 4326 al formato EPSG 32632.

3. lettura dei dati relativi agli archi dal file CSV e riproiezione della geometria degli archi dal sistema di coordinate EPSG 4326 al formato EPSG 32632.

4. filtraggio degli archi non percorribili né a piedi né in bicicletta.

5. ricalcolo della lunghezza degli archi in base al nuovo sistema di coordinate. La trasformazione operata precedentemente risulta utile soprattutto in questo frangente poiché permette di ottenere una lunghezza dei percorsi più veritiera, e quindi permette anche di migliorare anche il calcolo dei percorsi minimi.

6. filtraggio degli archi non intersecanti con l'area di interesse espressa nel GeoJSON riproiettato.

7. eliminazione dei nodi isolati

8. esportazione del grafo nel file *.graphml*

Di seguito lo script per la generazione del file Graphml nel listato 3.4 in cui i commenti sono associati all'elenco precedente in modo che risultino più chiari. Per lo script è stato impiegato l'utilizzo delle librerie Pandas, Geopandas e Shapely.

Listing 3.4: Script per la costruzione del grafo

```
def convert_otp_graph_to_igraph(
```

```
    node_csv_file: str,
    edge_csv_file: str,
    bounds_file: str,
    igraph_out_file: str,
) -> ig.Graph:

    # (1) Riproiezione della geometria dei confini in
       EPSG 32632
    original_geometry = gpd.read_file(bounds_file)['
       geometry'][0]
    bounds = geom_utils.project_geom(original_geometry)

    # (2) Lettura dei nodi dal CSV
    n = pd.read_csv(node_csv_file, sep=';')
    log.info(f'read {len(n.index)} nodes')
    log.info('creating node gdf')

    n[Node.geometry.name] = [
        shapely.wkt.loads(geom) if isinstance(geom, str
           ) else Point() for geom in n[Node.geometry.
           name]
    ]
    n[Node.geom_wgs.name] = n[Node.geometry.name]
    n = gpd.GeoDataFrame(n, geometry=Node.geometry.name
       , crs=CRS.from_epsg(4326))
    log.info('reprojecting nodes to etrs')
    n = n.to_crs(epsg=gp_conf.proj_crs_epsg)
    log.debug(f'nodes head: {n.head()}')

    # (3) Lettura degli archi dal CSV
    e = pd.read_csv(edge_csv_file, sep=';')
```

```
log.info(f'read {len(e.index)} edges')
log.debug(f'edge column types: {e.dtypes}')
log.debug(f'edges head: {e.head()}')
log.info('creating edge gdf')
e[Edge.geometry.name] = [
    shapely.wkt.loads(geom) if isinstance(geom, str
        ) else LineString() for geom in e[Edge.
        geometry.name]
]
e[Edge.geom_wgs.name] = e[Edge.geometry.name]
e = gpd.GeoDataFrame(e, geometry=Edge.geometry.name
    , crs=CRS.from_epsg(4326))
log.info('reprojecting edges to etrs')
e = e.to_crs(epsg=gp_conf.proj_crs_epsg)
log.debug(f'edges head: {e.head()}')

# (4) filtraggio dei degli archi non percorribili
    ne a piedi ne in bicicletta
def filter_df_by_query(df: pd.DataFrame, query: str
    , name: str = 'rows'):
    count_before = len(df.index)
    df_filt = df.query(query).copy()
    filt_ratio = (count_before-len(df_filt.index))
        / count_before
    log.info(f'filtered out {count_before-len(
        df_filt.index)} {name} ({round(filt_ratio *
        100, 1)} %) by {query}')
    return df_filt

e_filt = filter_df_by_query(e, f'{Edge.
    allows_walking.name} == True or {Edge.
```

```
    allows_biking.name} == True', name='edges')
e_filt = filter_df_by_query(e_filt, f'{Edge.
    is_no_thru_traffic.name} == False', name='edges
    ')

log.info('adding nodes to graph')
G = ig.Graph(directed=True)
G.add_vertices(len(n.index))
for attr in Node:
    if attr.name in n.columns:
        G.vs[attr.value] = list(n[attr.name])
    else:
        log.warning(f'node column {attr.name} not
            present in dataframe')

# (5) Ricalcolo della lunghezza degli archi
e_filt[Edge.length.name] = [
    round(geom.length, 4) if isinstance(geom,
        LineString) else 0.0 for geom in e_filt[Edge
        .geometry.name]
]

def get_ig_uv(edge):
    return (ids_otp_ig[edge['node_orig_id']],
        ids_otp_ig[edge['node_dest_id']])

e_filt['uv_ig'] = e_filt.apply(lambda row:
    get_ig_uv(row), axis=1)
e_filt[Edge.id_ig.name] = np.arange(len(e_filt.
    index))
G.add_edges(list(e_filt['uv_ig']))
```

```
for attr in Edge:
    if attr.name in e_filt.columns:
        G.es[attr.value] = list(e_filt[attr.name])
    else:
        log.warning(f'edge column {attr.name} not
            present in dataframe')

# (6) Rimozione degli archi al di fuori di Bologna
blgn_buffered = bounds.buffer(100)

def intersects_hma(geom: Union[LineString, None]):
    if not geom or geom.is_empty:
        return True
    return geom.intersects(blgn_buffered)

e_gdf = ig_utils.get_edge_gdf(G)
log.info('finding edges that intersect with Area')
e_gdf['in_hma'] = [intersects_hma(line) for line in
    e_gdf[Edge.geometry.name]]
e_gdf_del = e_gdf.query('in_hma == False').copy()
out_ratio = round(100 * len(e_gdf_del.index)/len(
    e_gdf.index), 1)
log.info(f'found {len(e_gdf_del.index)} ({out_ratio
    } %) edges outside Area')

before_count = G.ecount()
G.delete_edges(e_gdf_del.index.tolist())
after_count = G.ecount()
log.info(f'deleted {before_count-after_count} edges
    ')
```

```
    # (7) eliminazione dei nodi isolati
    del_node_ids = [v.index for v in G.vs.select(
        _degree_eq=0)]
    log.info(f'deleting {len(del_node_ids)} isolated
        nodes')
    before_count = G.vcount()
    G.delete_vertices(del_node_ids)
    after_count = G.vcount()
    del_ratio = round(100 * (before_count-after_count)
        / before_count, 1)
    log.info(f'deleted {before_count-after_count} ({
        del_ratio} %) nodes')

    # (8) esportazione del grafo
    if igraph_out_file:
        if igraph_out_file and os.path.exists(
            igraph_out_file):
            os.remove(igraph_out_file)

        ig_utils.export_to_graphml(G, igraph_out_file)

    return G
```

L'ultimo passaggio per la creazione del grafo prevede di aggiungere ad ogni arco oltre che un valore per la lunghezza in metri anche un valore per la velocità di percorrenza in bicicletta calcolato come lunghezza espressa in metri dell'arco per un coefficiente che esprime la velocità di percorrenza dell'unità di spazio.

3.2.3 Calcolo dei percorsi minimi

Nell'attuale implementazione del software è possibile calcolare tre tipologie di rotte:

- percorso *fresh* in cui viene tenuto conto della velocità di percorrenza di un tratto di strada e dell'indice di qualità dell'aria
- percorso *no accidents* in cui viene tenuto conto della velocità di percorrenza di un tratto di strada e dell'indice relativo alla frequenza di incidenti
- un percorso *fresh and no accidents* in cui viene tenuto conto sia della velocità di percorrenza di un tratto di strada che di entrambi gli indici

Il software utilizza l'algoritmo di Dijkstra per calcolare il percorso più veloce sfrattando i dati presenti nel grafo in memoria.

Per integrare i dati della qualità dell'aria nel calcolo delle rotte, è stata definita una funzione che permette di calcolare un nuovo costo per l'arco, e che tiene conto, oltre al tempo di percorrenza, anche di un ulteriore costo, dato dalla combinazione tra tempo di viaggio relativo all'arco e il dato sulla qualità dell'aria.

$$C_e = C_t + \sum_{i=1}^{n} C_t \times cf_i \times s$$

Dove C_e è il costo totale (composito) del arco, C_t è il costo di base del arco proporzionale alla distanza di viaggio e alla velocità definita, cf è il fattore di costo da utilizzare (la sommatoria è necessaria in caso ci siano più costi da prendere in considerazione), e s è un coefficiente di sensibilità arbitrario per assegnare un peso applicabile per il componente di costo che esprime quanto peso dovrà avere il costo stesso nel calcolo del percorso.

Il coefficiente di sensibilità s consente di trovare più percorsi alternativi per una singola coppia origine-destinazione, infatti, variando la sensibilità, è possibile trovare sia risultati in cui il calcolo del percorso minimo sia basato

maggiormente sul costo del tempo di percorrenza, in caso di fattore di sensibilità con valore basso, sia risultati in cui il calcolo è basato maggiormente sul parametro della qualità dell'aria, in caso di fattore di sensibilità con valore alto.

Secondo la funzione definita e considerando di utilizzare una lista di valori per la sensibilità di ogni fattore di costo per ogni arco saranno definiti:

- tanti costi quante sono le sensibilità per ogni fattore di costo e le modalità di viaggio

- tanti costi quanti sono gli elementi del prodotto cartesiano delle sensibilità per ogni fattore di costo in caso di costi compositi che tengano conto di più fattori di costo.

Produrre diversi costi è una caratteristica molto interessante poiché è difficile determinare a priori quanto l'utente desideri che un fattore di costo impatti nel calcolo dei percorsi e, in caso siano considerati più fattori di costo, quale si desideri che impatti maggiormente.

L'utilizzo della sensibilità risulta interessante soprattutto nella prospettiva in questa prospettiva: infatti, sfruttando questo parametro e considerando le possibili combinazioni dei valori delle sensibilità per ogni parametro, risulterebbe possibile calcolare un maggior numero di percorsi alternativi, in cui vengono considerati i fattori presi in considerazioni ma con diverse combinazioni di sensibilità, ampliando ulteriormente la possibilità di scelta dell'utente.

Per selezionare i corretti coefficienti di sensibilità sarà necessario ricordare che le variabili ambientali con limitata variazione spaziale richiedono coefficienti di sensibilità più elevati per trovare percorsi alternativi ottimizzati per l'esposizione, e quale range di valori il parametro di costo può assumere, in modo da bilanciare correttamente il suo peso rispetto agli altri.

Per il calcolo dell'algoritmo ci si è appoggiati alla funzione della libreria *igraph get_shortest_paths* che utilizza proprio l'algoritmo di Dijkstra per il

calcolo di percorsi minimi in cui i pesi del grafo sono espressi con valore positivi [55].

Di grande importanza risulta essere quindi l'aggiornamento del valore del peso del grafo. Nella sottosezione successiva verrà descritto questo processo.

3.2.4 Aggiornamento real-time dei pesi del grafo

Per integrare ulteriori fattori di costo, in particolare il fattore di costo relativo alla qualità dell'aria e sulla probabilità di incidenti, è stato necessario implementare una funzionalità che permetta di aggiungere e aggiornare il dato della qualità dell'aria tra gli attributi degli archi del grafo.

L'implementazione della funzionalità prevede l'utilizzo di diversi job:

I primi due job si occupano in senso stretto di mettere in relazione i collegamneti tra i nodi con i fattori di costo: un job utilizza l'API di *Ecosister Air Quality Service GET /interpolate*, descritta nella sezione , e ottiene i dati sulla qualità dell'aria, il secondo invece utilizza un file mockup che indica le aree con più alto indice di incidenti per interpolare il valore di tale indice sugli archi.

Entrambi gli script generano ognuno un file GeoJSON, associando a ogni strada del grafo i corrispondenti dati sulla qualità dell'aria. I file JSON sono poi gli stessi che verranno restituiti dall'API *GET /map-data* descritta nella sottosezione 3.2.5.

Ogni job legge il file relativo al grafo prodotto come descritto nella sottosezione 3.2.2 e elabora gli archi del grafo: dopo aver eliminato eventuali geometrie non valide, per ogni arco viene individuato il punto medio della geometria. Poi viene richiesta l'interpolazione di tutti i punti di campionamento, e infine viene generato un file JSON: come esempio, nel listato **??**, è riportato il risultato del job che si occupa dell'aggiornamento dei dati della qualità dell'aria in cui compaiono i valori relativi agli inquinanti e dell'indice di qualità dell'aria per ogni arco del grafo identificato con il suo id. Nel nome del file salvato compare un timestamp utile per valutare se c'è stato un aggiornamento rispetto a un valore precedente.

Listing 3.5: File prodotto dall'associazione dati sulla qualità dell'aria - archi del grafo


```
"data": [
        {
            "id_ig": 0,
            "aq": {
                "AQI_american": 164,
                "AQI_european": 5,
                "CO": 11.141652328365804,
                "NO2": 170.4748169665242,
                "O3": 197.61698624563795,
                "PM10": 467.1992172344129,
                "PM2.5": 79.93833760861932,
                "SO2": 640.9138778653362,
                "position": [
                    11.324383150000001,
                    44.505426650000004
                ]
            }
        },
        {
            "id_ig": 1,
            "aq": {
                "AQI_american": 68,
                "AQI_european": 2,
                "CO": 2.596631119297742,
                "NO2": 87.23753237797433,
                "O3": 59.88610123021903,
                "PM10": 70.37051314988119,
                "PM2.5": 20.115559156621273,
```

```
                "SO2": 159.46762329392263,
                "position": [
                    11.3725251,
                    44.49632055000001
                ]
            }
        }, ...
```

Gli altri due job leggono ognuno il corrispondente file GeoJSON prodotto e lo utilizzano per aggiornare le proprietà del grafo presenti in memoria.

Il processo periodicamente legge il valore del file salvato, e nel caso in cui il timestamp contenuto nel nome sia successivo al timestamp di una lettura precedente, il job aggiorna i dati presenti sul grafo.

Oltre ad aggiungere ad ogni arco il valore relativo alla qualità dell'aria e all'indice relativo agli incidenti vengono calcolati e aggiunti i costi utilizzabili per il calcolo dei percorsi minimi come descritto nella sottosezione precedente.

3.2.5 Interfaccia RESTful

Per rendere disponibili le sue funzionalità il servizio mette a disposizione un'interfaccia REST composta principalmente da 3 API. Nel listato di codice 3.6 è riportata la loro definizione e nell'elenco successivo saranno descritte singolarmente.

Listing 3.6: "API Messe a disposizione dal servizio"

```
# (1)
GET /paths/<travel_mode>/<routing_mode>/<orig_lat>,<
    orig_lon>/<dest_lat>,<dest_lon>

# (2)
GET /map-data-status/<map_data>
```

```
# (3)
GET /map-data/<map_data>
```

1. La prima API permette all'utente di richiedere il calcolo di un percorso minimo impostando i parametri in base ai parametri di percorso definiti nell'URL.

   - *travel_mode* è il parametro che permette di specificare come verrà compiuto il tragitto: a piedi, valorizzando il parametro con la stringa *walk*, o in bicicletta, valorizzando il parametro con la stringa *bike*.
   - *routing_mode* indica quale devono essere i fattori di costo considerati nel calcolo del percorso. Nell'implementazione attuale è possibile utilizzare solamente i valori *fast*, che permette di includere nel calcolo la velocità di percorrenza, oppure valore *clean*, che permette di considerare anche il valore relativo alla qualità dell'aria.
   - *orig_lat*,*orig_lon*: indicano le coordinate del punto di partenza in formato EPSG 4326
   - *dest_lat*, *dest_lon*: : indicano le coordinate del punto di arrivo in formato EPSG 4326 (non è necessario trasporre le coordinate in EPSG 32632 in quanto servono solo per individuare i possibili archi percorribili per costruire il percorso, ma il peso utilizzato dall'algoritmo per ogni arco è comunque già stato corretto durante la costruzione del grafo stesso)

2. la seconda api è una API di utilità che permette di capire lo stato di aggiornamento dei dati per un dato fattore di costo. Infatti prima di richiedere questi dati, per evitare di caricare dati già presenti è stata pensata questa API che permette di vedere qual'è stato l'ultimo aggiornamento e dà la possibilità a chi la utilizza di decidere se effettuare

la richiesta effettiva della mappa delle strade corredata dai valori sulla qualità dell'aria, o con i dati relativi all'indice di incidenti.

L'api risponde con un JSON con due campi:

- *map_data_available*: un campo booleano che indica che è disponibile una mappa
- *map_data_datetime*: un campo che indica la data della mappa attualmente disponibile

Il parametro *mapdata* permette di selezionare a quali fattori di costo fare riferimento.

3. *u*tilizzando la terza API è possibile reperire il JSON in cui è rappresentato il reticolo stradale come descritto nella sottosezione 3.2.4. Nel caso il file sia pronto l'API lo inoltra sempre in formato JSON, in caso non sia presente l'API risponde con un errore 404.

3.3 Ecosister Air Quality Service

In questa sezione verrà descritto il servizio Ecosister Air Quality Service, in particolare sarà descritta la sua interfaccia e alcuni dettagli sulla generazione dei dati mockup sulla qualità dell'aria.

Il servizio Ecosister Air Quality Service ha come scopo principale quello di produrre dei dati mock-up randomici e variabili nel tempo relativi alla qualità dell'aria.

Anche questo servizio è stato svilupparo in Python utilizzando la libreria Flask.

Le sue API RESTful riportata nel listato 3.7 espone chiaramente le funzionalità del servizio.

Listing 3.7: "API Ecosister Air Quality Service"

```
# (1)
```

```
GET /sensors

# (2)
POST /interpolate
```

- La prima API permette di reperire in formato JSON una lista di sensori e le loro informazioni relative agli inquinanti rilevati in quel momento, con l'EuroAQI e l'USAQI di conseguenza corrispondenti

- La seconda API permette prende in ingresso un oggetto JSON con una chiave "positions" contenente una lista di liste, dove ogni lista interna rappresenta una coppia di valori di longitudine e latitudine, ed esegue l'interpolazione per ogni coppia di valori fornita nella lista "positions", restituendo quindi un oggetto JSON contenente i dati interpolati per ogni posizione: valore degli inquinanti e indici di qualità dell'aria americano e europeo conseguentemente calcolati

Per la generazione dei dati il servizio utilizza un file JSON di configurazione chiamato *sensors.json* che rappresenta una lista di sensori mock-up, con il loro identificativo e la loro posizione che si trova all'interno dei confini di Bologna.

Ad ogni sensore è associata la lista di inquinanti utili per calcolare EuroAQI, come descritto nella sottosezione 1.3.1: per ogni inquinanti è espresso un range di valori da utilizzare per calcolare il valore randomico variabile mockup.

Quando viene richiesta la lista di sensori vengono prodotti i dati mock-up leggendo la lista dei sensori e generando dei valori per gli inquinanti entro i range definiti dal file di configurazione e poi vengono calcolati gli indici di qualità dell'aria.

Quando viene richiesta un'operazione di interpolazione vengono prima generati i dati degli inquinanti per i sensori, poi con il metodo dell'Inverse

Distance Weighting vengno interpolati i valori degli inquinanti per ogni posizione, infine, sempre per ogni posizione, vengno calcolati gli indici di qualità dell'aria.

3.4 Ecosister Users Service

In questa sezione verrà descritto il servizio Ecosister Users Service, in particolare sarà descritta la sua interfaccia.

Il servizio Ecosister Users Service ha come scopo principale la gestione e dei dati relativi agli utenti e la loro memorizzazione persistente.

Il servizio è stato realizzato utilizzando Python e Flask, mentre il database per la memorizzazione persistente è stato realizzato con MongoDB.

Nel listato di codice 3.8 è riportata l'interfaccia RESTful del servizio e la descrizione corrispondente a ogni endpoint:

Listing 3.8: "API Ecosister Users Service"

```
# 1
(POST) /register

# 2
(POST) /login

# 3
(GET) /protected

# 4
(POST) /update\_home

# 5
(POST) /update\_work
```

```
# 6
(POST) /add\_favourite\_path

# 7
(POST) /remove\_favourite\_path
```

1. Permette di registrare un nuovo utente prendendo in ingresso un JSON contenente username e password.

2. Permette di autenticare un utente esistente. L'API ricghiede nel body della richiesta un JSON contenente username e password. In caso di successo: il servizio restituisce un token JWT contenente informazioni sull'utente e codice di stato 200, mentre in caso di errore restituisce un messaggio di errore e codice di stato 401 Unauthorized.

3. API per recuperare i dati personali dell'utente autenticato in formato JSON. Un Token JWT valido nell'header della richiesta è richiesto per effettuare l'operazione.

4. API per l'aggiornamento della posizione *casa* dell'utente autenticato. Un Token JWT valido nell'header della richiesta è richiesto per effettuare l'operazione

5. API per l'aggiornamento della posizione *lavoro* dell'utente autenticato. Un Token JWT valido nell'header della richiesta è richiesto per effettuare l'operazione

6. Permette di aggiungere un percorso preferito nella lista dei percorsi preferiti di un utente autenticato. Un Token JWT valido nell'header della richiesta è richiesto per effettuare l'operazione

7. Permette di rimuovere un percorso preferito nella lista dei percorsi preferiti di un utente autenticato. Un Token JWT valido nell'header della richiesta è richiesto per effettuare l'operazione

3.5 Ecosister UI

In questa sezione verrà descritta l'applicazione frontend realizzata per il progetto Ecosister. Nella prima sottosezione verrà presentata l'interfaccia, e verranno descritte le scelte progettuali che hanno portato al design della stessa. Nella seconda sottosezione verrà descritta nel dettaglio l'implementazione effettiva dell'applicazione.

3.5.1 Progettazione e Design dell'interfaccia

In questa sottosezione verrà presentata l'interfaccia, e verranno descritte le scelte progettuali che hanno portato al design della stessa.

La progettazione dell'interfaccia dell'applicazione è stata effettuata tenendo in considerazione degli obiettivi e delle funzionalità definiti in fase di analisi, dello studio sullo stato dell'arte delle applicazioni attualmente presenti.

Il design dell'interfaccia grafica è stato guidato dai seguenti fattori:

- i requisiti funzionali e non funzionali definiti nel capitolo 1
- tra i requisiti non funzionali in particolare il vincolo di realizzare un'applicazione mobile-first
- gli obiettivi prefissati per il progetto Ecosister espressi nel già citato documento
- il principio *KISS*, orientando l'interfaccia verso uno stile semplice che utilizza strumenti per la comunicazione di tipo non testuali, privilegiando icone, colori e simboli
- lo studio effettuato sullo stato dell'arte dell'applicazioni che utilizzano dati sulla qualità dell'aria riportato nel primo capitolo: si è cercato di sviluppare un'applicazione che avesse una continuità nel design con quelle già esistenti, sia per sfruttare il know-how raggiunto dalle precedenti soluzioni, sia per proporre agli utenti un'interfaccia simile

alle applicazioni che comunemente utilizzano evitando che si trovino spaesati con una user-experience del tutto nuova.

I componenti dell'interfaccia dell'applicazione sono stati divisi in tre gruppi in base alla loro disposizione all'interno dell'interfaccia e in base alle loro funzionalità, sia singole, sia in sinergia con gli altri elementi.

Di seguito saranno presentati i gruppi individuati e i componenti che ne fanno parte.

Mappa

La mappa è il componente più importante dell'applicazione di Ecosister e occupa la maggior parte dell'interfaccia.

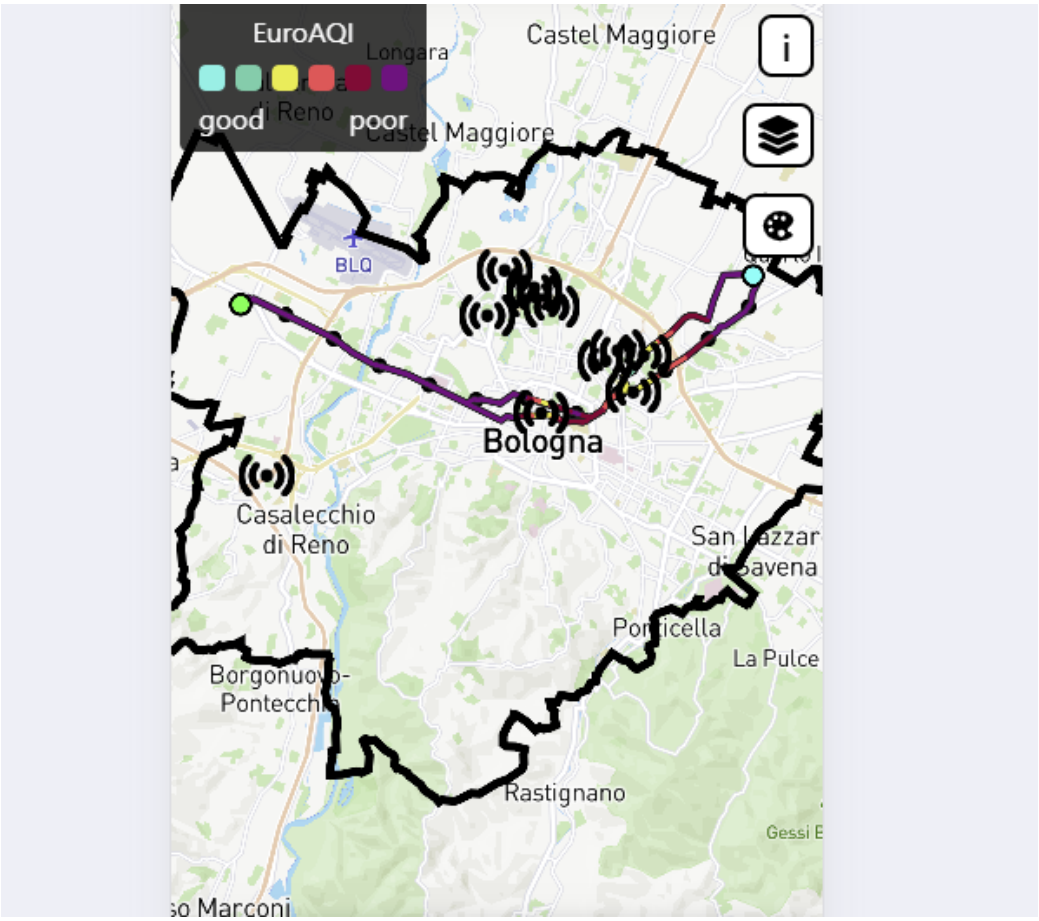

Figura 3.2: Mappa vista *City*

Si tratta di una mappa interattiva e permette all'utente di svolgere tutte quelle funzioni di base che si auspica abbia una mappa interattiva, in particolare, zoom e spostamento del focus sulla mappa.

All'avvio dell'applicazione la mappa si trova centrata su Bologna. Come da requisito sulla mappa compaiono ben evidenziati i confini della città.

Sulla mappa si trovano poi diversi *marker*. Per ogni marker è stata selezionata un'icona che renda immediatamente comprensibile la semantica del marker. I marker presenti sulla mappa sono:

- un marker per evidenziare i sensori

- un marker per indicare il punto di partenza selezionato dall'utente per il calcolo del percorso
- un marker per indicare il punto di destinazione selezionato dall'utente per il calcolo del percorso
- un marker per la posizione *lavoro* se salvata dall'utente loggato
- marker per la posizione *casa* se salvata dall'utente loggato

Per selezionare i punti di partenza e destinazione basterà cliccare sulla mappa e selezionare sul pop-up che si apre come l'applicazione dovrà interpretare il punto indicato, come punto di partenza o di destinazione.

Nel caso l'utente sia loggato all'interno del sistema gli sarà dato anche la possibilità di salvare il punto come punto *casa* o *lavoro* sempre cliccando sull'opzione desiderata nel pop-up che compare cliccandouslla mappa.

Una volta che un utente ha richiesto il calcolo di un percorso questo viene rappresentato sulla mappa in modo che ogni segmento che compone il percorso sia colorato in base al costo del percorso scelto: ad esempio se viene richiesto di calcolare un percorso *aria fresca* i segmenti del percorso calcolato saranno colorati in base alla specifica dell'Indice di qualità dell'aria Europeo che nell'implementazione corrente viene utilizzato per calcolare i percorsi tenendo conto anche della qualità dell'aria. Nel caso sia richiesto di calcolare un percorso tenendo conto di più parametri viene utilizzato un parametro di default per la colorazione della mappa. In caso il calcolo del percorso generi più possibilità queste vengono rappresentate tutte ed è possibile selezionare uno dei percorsi cliccandoci sopra: questo viene evidenziato rispetto agli altri in modo da risultare distinguibile.

Sull'interfaccia compaiono sulla destra anche quattro tasti. Il tasto in cui compare l'icona con più piani impilati e quella della tavolozza di colori sono i tasti che permettono di controllare la visualizzazione della mappa.

Con il primo tasto, il cui design è ispirato al tasto *livelli* di Google Maps, permette di cambiare la visualizzazione della mappa, che come da specifiche presenta tre stili differenti:

- Colorazione *City*: è la colorazione rappresentata dalla figura 3.2. É una rappresentazione classica che permette di visualizzare le principali informazioni geografiche e i punti di interesse del territorio. É anche lo sile che permette di visualizzare meglio il percorso e i marker in quanto questi sono in netto contrasto visivo con tutti gli altri elementi della mappa

- Colorazione *Heatmap*: come da requisiti è possibile visualizzare sulla mappa i valori di alcuni dati con una colorazione a zone heatmap come si vede nella figura 3.3.

- Colorazione *Street-by-street*: come da requisiti è possibile visualizzare sulla mappa i valori di alcuni dati associati ai singoli segmenti che compongno le strade, come si vede nella figura 3.4.

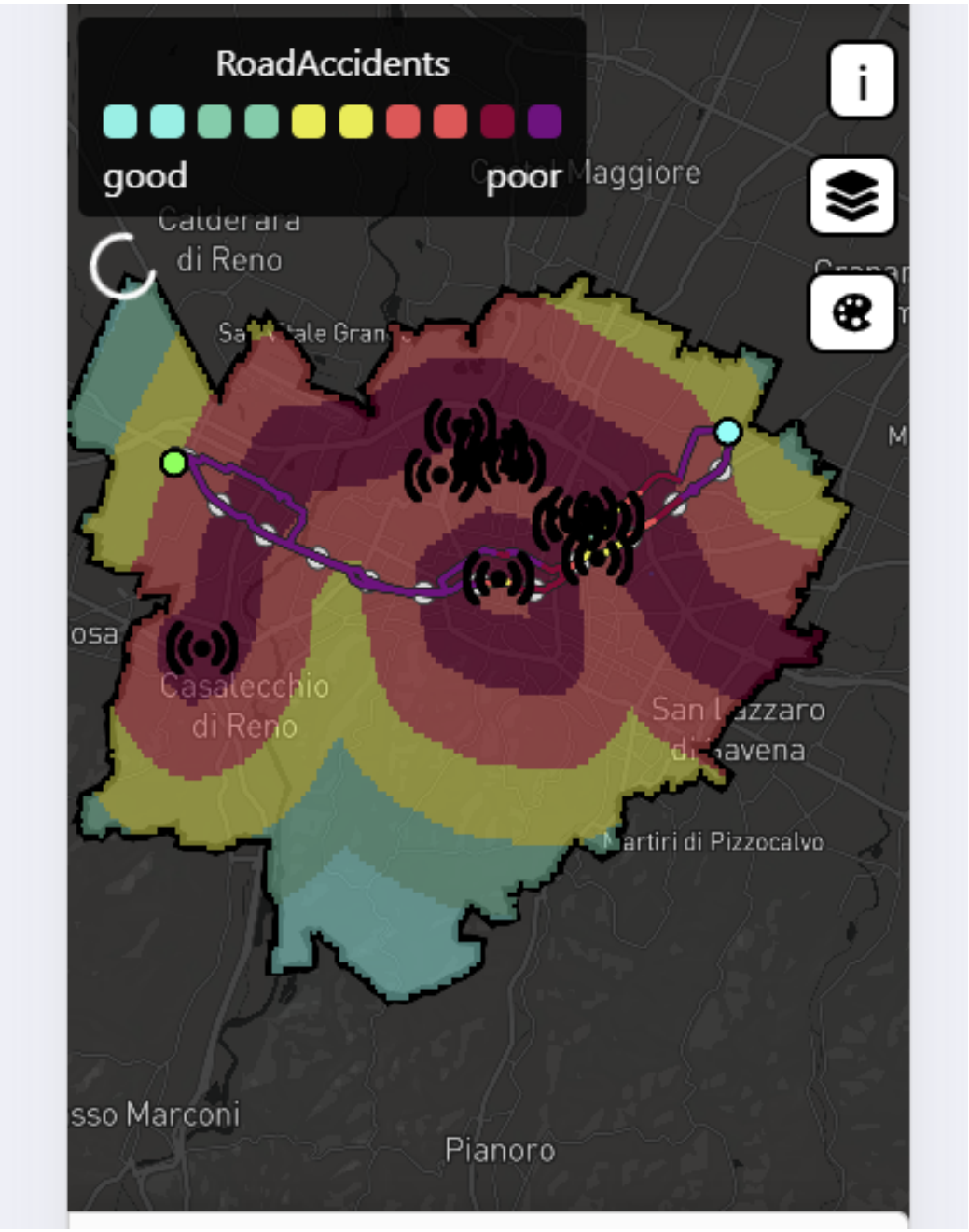

Figura 3.3: Mappa vista *Heatmap*

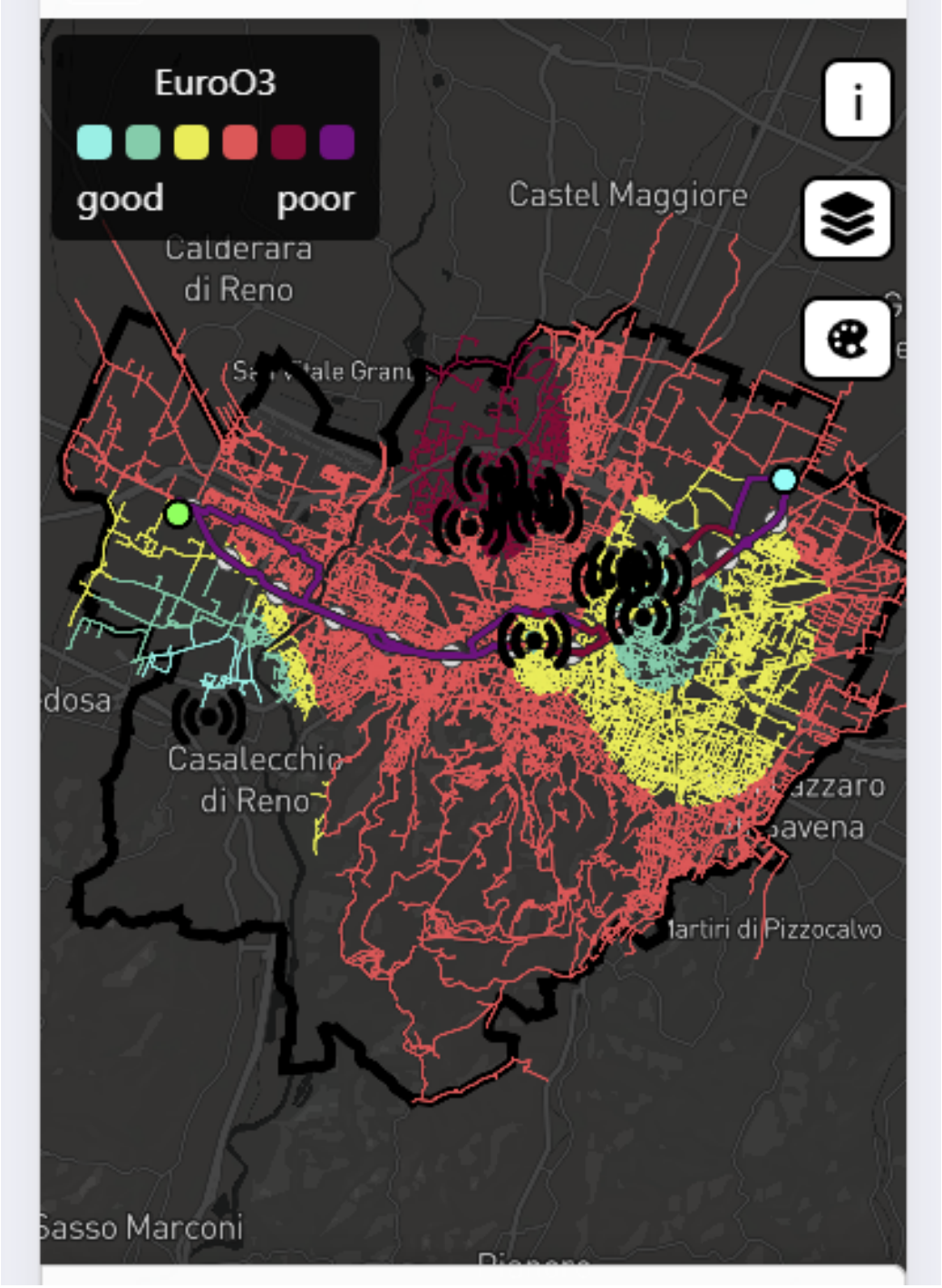

Figura 3.4: Mappa vista *Street-by-street*

Il tasto *tavolozza di colori* risulta utile quando si utilizzano la visualizzazione Heatmap e Street-by-street. Infatti, cliccando sul tasto si apre un popup che permette di scegliere quali dati devono essere utilizzati per colorare la mappa: nell'implementazione attuale è possibile colorare la mappa in base ai valori degli inquinanti utilizzati per calcolare l'European Air Quality Index e alla colorazione definita dalla specifica riportata nella figura 1.1, l'Euro Air Quality Index, utilizzando sempre la colorazione della specifica, e l'indice degli incidenti, la cui colorazione in questa implementazione è frutto di una specifica mock-up.

Per aiutare l'utente a interpretare i valori presenti sulla mappa è stata inserita una legenda.

Quando la mappa viene visualizzata con lo stile *City* la legenda compare solo quando viene visualizzato un percorso mostrando i colori relativi al costo utilizzato per il calcolo del percorso: se il percorso è un percorso *con aria fresca* viene visualizzata la legenda relativa all'European Air Quality Index, se il percorso calcolato è di tipo *non pericoloso* viene utilizzata la legenda relativa all'indice di incidenti, nel caso sia visualizzato un percorso *con aria fresca e non pericoloso* viene mostrato di default comunque la legenda dell'European Air Quality index, la stessa della colorazione del percorso.

Oltre ai tasti per la gestione della mappa compaiono poi anche un tasto che permette di visualizzare delle informazioni generiche relative all'applicazione Ecosister, necessarie per rendere noto all'utente che si tratta di un'applicazione in fase di di sviluppo, e un tasto visibile solo da un utente che abbia effettuato il login, che se cliccato apre un pop-up un cui compaiono due bottoni che danno all'utente la possibilità di:

- salvare il punto di partenza e di destinazione come un percorso *preferiti*.

- visualizzare i percorsi *preferiti* e dare all'utente la possibilità di selezionarne uno e settare le attuali posizioni di partenza e di destinazione in base a quelle corrispondenti al percorso preferito.

TopPanel

Il TopPanel è il pannello di controllo che si trova nella parte alta dell'applicazione.

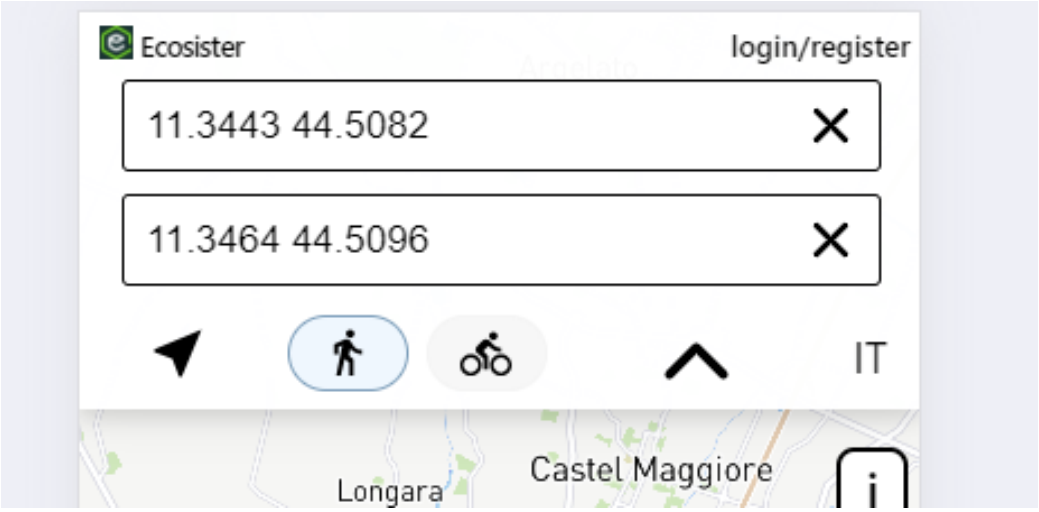

Figura 3.5: Top Panel

I componenti che si trovano in questo pannello danno all'utente la possibilità di:

- effettuare operazioni di login, logout e registrazione
- visualizzare le coordinate dei punti di partenza e destinazione selezionati sulla mappa
- settare la lingua dell'applicazione, con possibilità di scegliere sia italiano che inglese
- visualizzare la posizione corrente dell'utente
- impostare il la modalità di viaggio, in bicicletta o a piedi
- chiudere il pannello stesso in modo da avere una visione più ampia sulla mappa

Essendo un applicazione mobile first il design dei comandi è stato scelto in modo che risultasse intuitivo, semplice e senza fronzoli in modo da avere molte funzionalità in poco spazio e accessibili con pochi tocchi sullo schermo; ad esempio sia per quanto riguarda il tasto di login e logout, sia per quanto riguarda il tasto per il cambiamento della lingua, si tratta di tasti di tipo toggle: questo riduce lo spazio utilizzato sull'interfaccia in cui non devono essere quindi necessariamente mostrate tutte le opzioni che è possibile selezionare.

BottomPanel

I componenti che fanno parte del gruppo BottomPanel sono i componenti che si trovano nella parte bassa della schermata e sono componenti utili per la funzionalità di ricerca del percorso.

Figura 3.6: Bottoni per la ricerca del percorso

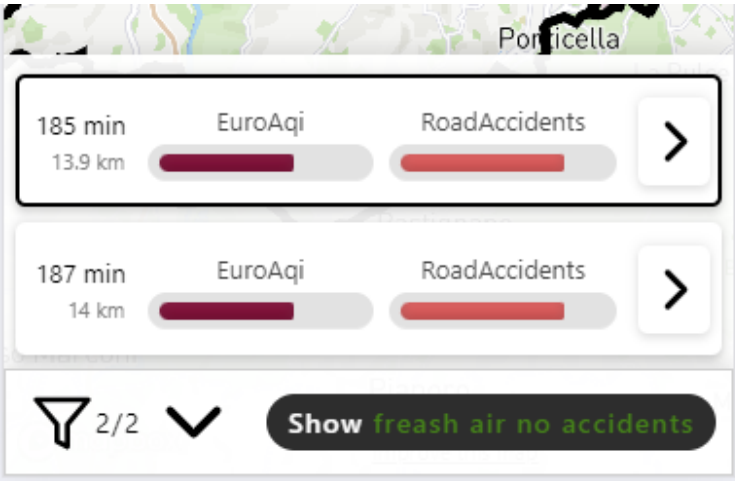

Figura 3.7: Pannello con la lista e le informazioni dei percorsi trovati

I bottoni rappresentati nella figura 3.6 sono i bottoni utilizzabili dall'utente per cercare i percorsi. Con il primo bottone è possibile cercare percorsi che tengano conto della qualità dell'aria, con il secondo è possibile cercare percorsi in modo da evitare strade con un alta frequenza di incidenti, e con il terzo è possibile cercare percorsi che tengano conto di entrambi i fattori. Le diciture indicate sui bottoni sono state scelte per evitare complesse descrizioni relative all'implementazione sottostante, permettendo all'utente di interpretare quale risultato è possibile ottenere facendo riferimento esclusivamente al risultato finale.

Una volta richiesto il calcolo di un percorso con una certa modalità, i bottoni lasciano il posto al pannello visibile in figura 3.7. In questo pannello sono elencati i percorsi trovati a partire da quello con tempo di percorrenza minore.

Cliccando un elemento sulla lista è possibile selezionare un percorso e evidenziarlo sulla mappa.

Per facilitare il confronto tra le varie possibilità trovate, come richiesto dai requisiti, sono stati inserite alcune informazioni per ogni percorso come:

tempo di percorrenza, indice di qualità dell'aria e indice della frequenza di incidenti.

Sempre con l'obiettivo di facilitare un confronto tra le possiblità riscontrate, con il bottone in bassa a sinistra a forma di inbuto è possibile filtrare i percorsi, come suggerisce l'icona, in base alla lunghezza del percorso.

Con il tasto in basso a destra invece è possibile utilizzare il toggle button per cambiare la modalità di ricerca dei percorsi.

3.5.2 Implementazione

In questa sottosezione si parlerà nel dettaglio dell'implementazione dell'applicazione web di Ecosister.

L'applicazione è stata sviluppata utilizzando principalmente le librerie React, Redux e la libreria MapboxGL.

Nelle sottosezioni successive saranno descritti i componenti dell'applicazione: ogni sottosezione fa riferimento a una cartella del progetto, e ogni cartella fa riferimento a uno dei gruppo descritti nella sottosezione precedente che rappresenta una porzione della schermata dell'applicazione e un gruppo di componenti correlati per funzionalità e collocazione.

Mappa

L'elemento centrale dell'applicazione è la mappa interattiva implementata principalmente da due componenti *Map* e *MapControl*.

Map è il componente che funge da contenitore principale per la mappa di Mapbox, gestendo l'inizializzazione, la configurazione e il rendering della mappa.

Il componente si occupa di:

- Caricare layout di base per la mappa o tramite impostazione di default, o scelta dall'utente in base alla selezione dello style *City*, *Heatmap* o *Street-by-street*.
- Aggiungere il layer che rappresenta i confini della città.

- Gestire gli eventi di click sulla mappa per permettere di selezionare i nuovi punti di origine e destinazione, o in caso di utente loggato anche la posizione *casa* e *lavoro*.
- Aggiornare la posizione e lo zoom della mappa in base alle azioni dell'utente o ai cambiamenti di stato dell'applicazione.
- Comunicare con il reducer di Redux per aggiornare lo stato dell'applicazione in base alle interazioni con la mappa.

Il ciclo di vita del componente è il seguente:

- Montaggio:
  - il componente crea una nuova istanza di MapboxGL.
  - il componente aggiunge listener per eventi sulla mappa.
  - il componente aggiunge il perimetro della città come layer sulla mappa.
  - il componente imposta la visualizzazione del centro della mappa

  Configura la finestra della mappa.
- Aggiornamento:
  - Aggiorna la telecamera se la mappa è pronta e non lo era in precedenza.
- Smontaggio: viene rimossa la mappa e i suoi listener

Il componente *MapControl* gestisce il posizionamento automatico iniziale sulla posizione dell'utente, aggiorna la visualizzazione in base a zoom e bbox specificati, e controlla l'ordine di sovrapposizione dei layer della mappa in modo da renderne la visualizzazione ordinata.

Il ciclo di vita del componente è il seguente:

- Aggiornamento:

- il componente controlla se la mappa è disponibile e se ci sono cambiamenti nei dati rilevanti (posizione utente, zoom, basemap, layer caricati). Aggiorna la posizione della mappa in base alla posizione dell'utente se necessario.
- Aggiorna la visuale della mappa in base a bbox specifici, considerando anche le dimensioni dello schermo e la presenza di percorsi visualizzati.
- Aggiorna lo stile della mappa se la visualizzazione passa da *City* a *Heatmap/Street-by-street* e viceversa.
- Riordina i layer della mappa in base a un ordine predefinito che ne consente una visualizzazione corretta.

Heatmap

Questo componente React permette di visualizzare la Heatmap, rappresentando la distribuzione di un costo specifico all'interno dell'area della città.

Per creare il layer relativo alla heatmap il componente crea una griglia di quadrati sovrapposti all'area definita dal poligono che rappresenta i confini della città. Poi, utilizzando *Ecosister Air Quality Servie* o *Ecosister Routes Service* in base ai dati da rappresentare, vengono interpolati i dati del costo da rappresentare. Infine, a seconda del dato associato al quadrato questo viene colorato.

Il componente utilizza lo state di Redux per accedere allo stato dell'applicazione, inclusi informazioni sulla mappa e costo selezionato da rappresentare.

Il componente implementa un meccanismo che permette di aggiornare lo stile della heatmap solo dopo un certo periodo di tempo trascorso dall'ultimo cambio di visualizzazione o ricezione dei dati, evitando aggiornamenti frequenti non necessari.

Di seguito il ciclo di vita del componente:

- Montaggio: viene effettuato il fetch dei dati realativi al costo selezionato

- Aggiornamento: viene effettuato un controllo per verificare se la mappa è selezionata e di conseguenza viene mostrato o nascosto il layer che rappresenta la heatmap sulla mappa. Viene aggioranto lo stato dei dati relativi al costo selezionato in base al fatto che sia stata selezionata la visualizzazione della Heatmap o sia stato selezionato un nuovo costo da rappresentare, si sia verificato uno spostamento della visuale della mappa. In caso i dati siano stati caricati correttamente e non siano previsti ritardi di aggiornamento legati al meccanismo di ottimizzazione dell'aggiornamento dell'interfaccia, viene aggiornato lo stile della heatmap in base ai dati aggiornati.

La procedura per la creazione e la colorazione della groglia è implementata in due funzioni *createGrid* e *assignGradientColor*.

La funzione *createGrid* genera una griglia di quadrati sovrapposti all'area definita da un poligono: prende in ingresso un GeoJSON che rappresenta il poligono di riferimento e un numero che indica la dimensione in gradi decimali di ciascun quadrato della griglia.

1. Viene creata una collezione di features vuota chiamata grid.

2. Si ottengono i limiti del poligono utilizzando la funzione *turf.bbox(polygon)* e si estraggono i valori minimi e massimi di latitudine e longitudine dai limiti del poligono.

3. Si esegue un doppio ciclo for nidificato: il ciclo esterno itera sulla longitudine (x) con incrementi pari alla dimensione desiderata di ogni quadrato presa in ingresso; il ciclo interno itera sulla latitudine (y) con incrementi incrementi pari alla dimensione desiderata di ogni quadrato presa in ingresso.

4. All'interno del ciclo interno viene creato un quadrato utilizzando la funzione *turf.polygon* con le coordinate dei quattro vertici identificati.

5. Si calcola il centro del quadrato utilizzando la funzione *turf.point* e le coordinate del centro vengono calcolate come media delle coordinate dei vertici opposti.

6. Si verifica se il centro del quadrato è all'interno del poligono di riferimento utilizzando la funzione *turf.booleanPointInPolygon*.

7. la funzione restituisce tutti i quadrati generati che si trovano all'interno del poligono di riferimento.

La funzione *assignGradientColors* assegna un colore ad ogni quadrato della griglia in base al valore del costo calcolato per la sua posizione centrale.

La funzione prende in ingresso una collezione GeoJSON contenente i quadrati della griglia e un array di oggetti contenenti i valori del costo calcolati per diverse posizioni, ovvero quelli restituiti dal servizio preposto per l'interpolazione del costo selezionato.

Street-by-street view

Questo componente React è responsabile della gestione della visualizzazione *Street-by-street*.

Il componente utilizza i dati relativi ai costi associati alle strade utilizzando l'apposita API messa a disposizione dal servizio *Ecosister Routes Service* descritta nella sottosezione 3.2.5.

Anche questo componente utilizza Redux per accedere allo stato dell'applicazione, inclusi informazioni sulla mappa selezionata, centro della mappa e costo selezionato.

Il ciclo di vita del componente è il seguente

- Montaggio:
  - Avvia il processo di aggiornamento dei dati del costo.
  - Aggiunge un listener all'evento sourcedata della mappa per rilevare quando i dati sono caricati e avviare l'aggiornamento dello stile con un leggero ritardo.

- Aggiornamento:
  - Nasconde il layer se la mappa selezionata non è *Street-by-street*, o viceversa lo mostra se la mappa selezionata è *Street-by-street*.
  - Aggiorna i dati del costo e lo stato del layer in base a diversi scenari: se la mappa selezionata cambia o il costo selezionato cambia, se i dati non sono già caricati o se il tipo di costo cambia.
  - Cambio di centro mappa: se il centro della mappa cambia e i dati del costo sono già caricati, avvia l'aggiornamento dello stile con un leggero ritardo.
  - Aggiorna lo stile del layer con i dati del costo solo se non ci sono ritardi in attesa e i dati del costo e della mappa sono disponibili.

Una volta caricati i dati del costo selezionato il componente utilizza la funzione *getUniqueFeatureId* per recuperare gli ID univoci delle feature del layer che rappresentano le strade evitando duplicati.

Per ogni id trovato viene cercato il dato del costo selezionato tra i dati caricati e viene così associata alla strada il giusto colore da visualizzare.

Visualizzazione dei percorsi calcolati

Il componente principale che si occupa della visualizzazione dei dati sulla mappa e il componente *PathEdges* che rappresenta sulla mappa tutti i segmenti dei percorsi calcolati.

Il componente riceve i dati di tutti i segmenti dei percorsi dallo store Redux e quando sono presenti dei percorsi da mostrare, carica sulla mappa i percorsi in formato GeoJson impostando lo stile in base alla giusta colorazione da rappresentare in funzione del fattore di costo su cui si basa il calcolo del percorso.

Il componente *pathSelected* evidenzia il percorso selezionato sulla mappa, disegnando sul percorso dei cerchi posti ad intervalli regolari lungo il percorso.

Il componente utilizza lo stato di Redux per capire quale percorso è stato selezionato.

TopPanel

I componenti facenti parte del TopPanel sono collocati pannello di controllo che si trova nella parte alta dell'applicazione.

Componenti per il login e la registrazione

Sono i componenti all'interno del top panel che gestiscono il login e la registrazione degli utenti

- UserStatus: questo componente visualizza lo stato di autenticazione dell'utente, mostrando il nome utente se loggato oppure un link per effettuare il login/registrarsi. Il componente di occupa di recuperare lo stato di autenticazione dallo store Redux e in base a questo visualizza il nome utente loggato (con possibilità di logout in caso di click) oppure un link per accedere o registrarsi permettendo la visualizzazione delle opportune modali.

- UserRecover: questo componente recupera automaticamente i dati dell'utente loggato in precedenza, se presenti, e aggiorna lo stato Redux di conseguenza.

- LoginModal: Questo componente gestisce la modale di login, consentendo agli utenti di inserire le credenziali di accesso e effettuare il login. Per il suo funzionamento il componente sfrutta le funzionalità di *Ecosister-Users-Service* descritte nella sezione 3.4. Il componente viene reso visibile o nascosto in base allo stato Redux che indica se la modale deve essere aperto e aggiorna lo stato Redux in base al risultato del login (successo o fallimento). L'interfaccia presenta un bottone che se premuto consente all'utente di effettuare la registrazione.

- RegisterModal: Questo componente gestisce la modale di registrazione, consentendo agli utenti di inserire le credenziali di accesso e effettuare la registrazione del proprio account sul sistema. Per il suo funzionamento il componente sfrutta le funzionalità di *Ecosister-Users-Service*

descritte nella sezione 3.4. Il componente viene reso visibile o nascosto in base allo stato Redux che indica se la modale deve essere aperto e aggiorna lo stato Redux in base al risultato dell'operazione(successo o fallimento).

I componenti *DestinationInput* e *OriginInput* permettono di visualizzare e modificare le coordinate del punto di partenza e di destinazione presenti nello store Redux.

Il componente *TravelModeSelector* permette di selezionare, e modificare conseguentemente nello store di Redux, i parametri relativi alla modalità di viaggio da selezionare, se in bici o a piedi.

BottomPanel

I componenti che fanno parte del gruppo BottomPanel sono i componenti che si trovano nella parte bassa della schermata e sono componenti utili per la funzionalità di ricerca del percorso.

Il pannello per la visualizzazione dei possibili percorsi è implementato nel componente *PathPanel*.

I bottoni per la ricerca dei percorsi sono invece implementati nel componente *FindPathButtons*.

3.6 Deployment

In questa sezione verrà descritto come è stato progettato e effettuato il deployment del sistema.

Come da requisiti, espressi nel capitolo 1, l'obiettivo che ci si è prefissati è stato quello di creare una modalità di deployment del sistema che prevedesse la messa in produzione del servizio con un solo comando.

Per ogni servizio, *Ecosister Routes Service*, *Ecosister Air Quality Service*, *Ecosister Users Service*, è stato creato un Dockerfile apposito per la creazione di un container adatto all'esecuzione. Per ogni servizio le corrispondenti immagini sono stati pubblicate su Dockerhub.

Nel repository *ecosister-build* si trova file docker-compose riportato nel listato di codice 3.9 che permette di avviare tutto il sistema con un il solo comando *docker-compose up*.

L'avviamento del sistema prevede:

1. Il download delle immagini dei servizi, dell'immagine di nginx [56], e mongodb [57]

2. Il popolamento del db con alcuni dati mockup.

3. L'avvio del servizio *Ecosister Air Quality Service*, del servizio *Ecosister Users Service* e di nginx.

4. L'avvio del servizio *Ecosister Routes Service*.

Listing 3.9: "File Docker-compose per il deployment"

```
version: "3.1"

services:
  routes-service:
    image: sdg97/ecosister-routes-service:latest
    ports:
      - "5000:5000"
    depends_on:
      - aq-service
    environment:
      AQ_SERVICE_INTERPOLATE_API: "http://aq-service
          :5000/interpolate"

  aq-service:
    image: sdg97/ecosister-aq-service:latest
    ports:
      - "5001:5000"
```

```
users-service:
  image: sdg97/ecosister-users-service:latest
  ports:
    - "5003:5000"
  depends_on:
    - db
  environment:
    PROD: "true"
    MONGO_URI: "mongodb://ecosister-username:
        ecosister-password@db:27017/?authMechanism=
        DEFAULT" # Adjust as needed
    SECRET_KEY: "your-prod-secret"
    JWT_SECRET_KEY: "your-prod-jwt-secret"

db:
  image: mongo:4.0.27
  restart: always
  environment:
    MONGO_INITDB_ROOT_USERNAME: ecosister-username
    MONGO_INITDB_ROOT_PASSWORD: ecosister-password
  volumes:
    - ./db_init:/docker-entrypoint-initdb.d
    - ./db_init/data:/tmp/data
    - ./db_data:/data/db # Persistent data storage
  ports:
    - "27017:27017"

nginx:
  image: nginx:latest
  ports:
```

```
    - "5002:5002"
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf:ro
    - ./build:/usr/share/nginx/html
  depends_on:
    - routes-service
    - aq-service
    - users-service
    - db
```

Il database mongodb sopracitato è il database utilizzato dal servizio *Ecosister Users Service* per lo storage dei dati relativi agli utenti.

Il servizio di nginx ha invece una duplice funzione come risulta evidente dal suo file di configurazione riportato nel listato [**?**]: svolge infatti una funzionalità di webserver per rendere pubblico la build dell'applicazione web di Ecosister, e svolge anche una funzione di router per le chiamate dell'applicazione ai vari servizi.

Listing 3.10: "File di configurazione di nginx"

```
events {}
http {
    server {
        listen 5002;
        server_name localhost;

        location /users/ {
            proxy_pass http://users-service:5000/;
        }

        location /routes/ {
            proxy_pass http://routes-service:5000/;
        }
```

```
        location /aq/ {
            proxy_pass http://aq-service:5000/;
        }

        location / {
            root /usr/share/nginx/html;
            try_files $uri /index.html;
        }
    }
}
```

Nel repository sono anche presenti la build dell'applicazione web di Ecosister e i dati mock per il database degli utenti utili per la popolazione in fase di avvio.

Il processo di deployment del servizio prevede:

1. la build delle immagini dei servizi e la pubblicazione su Dockerhub
2. la build dell'applicazione di Ecosister e la copia della cartella *build* ottenuta da questa operazione all'interno del repository *ecosister-build*
3. la pubblicazione del repository con la nuova versione dell'applicazione
4. la clonazione del repository o l'effettuazione di una operazione di pull all'interno di una copia giù esistente
5. l'avvio di tutto il sistema con il comando *docker-compose up*

Conclusioni

Per il progetto di tesi è stato sviluppato un sistema di routes planning per il calcolo di percorsi brevi nell'area di Bologna che tenga conto di diversi fattori di costo oltre alla velocità di percorrenza, quali la qualità dell'aria e la frequenza di incidenti che avvengono sulle strade percorse.

Come previsto dallo scope del progetto e dalle specifiche, il sistema è accessibile attraverso un'applicazione web responsive mobile-first, che permette di visualizzare la zona di Bologna coinvolta nel progetto, e i dati sulla qualità dell'aria e sugli incidenti, con una visualizzazione adeguata rispetto allo stato dell'arte delle applicazioni attualmente presenti della stessa tipologia, e permette all'utente di richiedere il calcolo di percorsi brevi in cui si tenga conto anche della qualità dell'aria e della frequenza degli incidenti.

Il sistema inoltre prevede aspetti di personalizzazione e memorizzazione delle informazioni sugli utenti, permettendo agli utilizzatori registrati di salvare e modificare informazioni sul proprio profilo.

La validazione del sistema rilasciato è stata svolta verificando la coerenza del comportamento dell'applicazione web con i requisiti dichiarati dopo aver effettuato il deployment del sistema.

Nel futuro sono previsti alcune modifiche e miglioramenti del sistema, con l'obiettivo di passare dalla realizzazione di una Proof of Concept a un prodotto vero e proprio.

In particolare, sarà necessario rivedere alcuni aspetti dell'interfaccia web che presentano alcuni elementi di obsolescenza sia nello stile che nel funzionamento, in modo da rendere la UX più disinvolta e più adatta agli utenti

mobile moderni: in primis i tasti di selezione dei costi e della modalità di visualizzazione della mappa devono ricompattare la selezione nel momento in cui viene cliccato un elemento al di fuori della selezione stessa, e deve essere ottimizzata la visualizzazione *heatmap*.

Inoltre, come descritto nel capitolo 3, per il building del grafo viene utilizzata una versione custom di OTP 1.5, e per il calcolo dei percorsi la libreria *igraph* e la sua implementazione dell'algoritmo di Dijkstra.

Essendo OTP un software open-source in continuo sviluppo, altamente performante e utilizzato in tantissimi contesti, si potrebbero estendere alcune caratteristiche del software in modo da sfruttare OTP stesso, nella sua ultima versione, OTP 2, per il calcolo delle rotte e la gestione del grafo.

Il vantaggio di questa decisione è la scelta di affidarsi quasi integralmente a una soluzione come OTP di cui è nota l'affidabilità, e l'efficienza nell'utilizzo dell'algoritmo Multi-criteria Range Raptor algorithm per il calcolo dei percorsi.

Per integrare il costo relativo alla qualità dell'aria e l'indice relativo agli incidenti nel calcolo del percorso di OTP è possibile procedere in due modi:

- sfruttare il modulo *DataOverlay* di OTP: si tratta di un modulo che un fase di building del grafo aggiunge dei pesi a ogni edge in base a dati non direttamente pertinenti al contesto stradale, come possono essere ad esempio i dati sulla qualità dell'aria [58]. La soluzione risulta efficace per il formato di dati utilizzato, JSON e NetCDF, uno dei formati di dati più utilizzati per descrivere dati metereologici tra cui anche la qualità dell'aria, e per la definizione dei costi dichiarativa e parametrica. La soluzione risulta invece problematica nel caso in cui si voglia mantenere il vincolo di utilizzare dati real-time aggiornati con una breve frequenza: sarebbe necessario studiare infatti un modo per riavviare il servizio, con conseguente ricostruzione del grafo con i dati aggiornati, senza intaccare l'availability del sistema.

- creare un'altra implementazione custom di OTP che estenda le funzionalità del sistema prendendo ispirazione dai moduli realizzati per

l'aggiornamento delle informazioni real-time come quelli utilizzati per aggiungere informazioni relative alle auto a noleggio disponibili. La soluzione risulterebbe ottimale per il caso d'uso e permetterebbe di rimanere allineati con le modifiche di OTP nel tempo, avendo un progetto sempre aggiornato, sia in caso si effettui una fork del progetto, sia nel caso si voglia introdurre la feature nel progetto OTP principale: in questo caso si beneficerebbe anche del contributo di tutti gli altri sviluppatori. Unico ostacolo da superare per la realizzazione è la comprensione del progetto OTP 2 che al momento risulta molto esteso e complesso.

Bibliografia

[1] Patto per il lavoro e il clima emilia romagna. https://www.regione.emilia-romagna.it/pattolavoroeclima.

[2] What's a linux container, red hat explanation. https://www.redhat.com/en/topics/containers/whats-a-linux-container,RedHat.

[3] Ecosister website. https://ecosister.it/.

[4] Geodesia, definizione, dizionario treccani. https://www.treccani.it/vocabolario/geodesia/.

[5] Epsg 32632. https://epsg.io/, .

[6] European air quality index. https://airindex.eea.europa.eu/Map/AQI/Viewer/#.

[7] European environment agency. https://www.eea.europa.eu/it.

[8] Us air quality index website. https://www.law.cornell.edu/cfr/text/40/appendix-G_to_part_58.

[9] U.s. environmental protection agency website. https://www.epa.gov/.

[10] 40 cfr appendix g to part 58 - uniform air quality index (aqi) and daily reporting. https://www.law.cornell.edu/cfr/text/40/appendix-G_to_part_58.

[11] Us air quality calculator by airnow. https://www.airnow.gov/aqi/aqi-calculator-concentration/.

[12] Plumelabs website. https://air.plumelabs.com/en/.

[13] Iqair web application and website. https://https://www.iqair.com/.

[14] Inverse distance weighting interpolation method. https://pro.arcgis.com/en/pro-app/latest/help/analysis/geostatistical-analyst/how-inverse-distance-weighted-interpolation-works.htm.

[15] Kringing interpolation method. https://pro.arcgis.com/en/pro-app/latest/tool-reference/3d-analyst/how-kriging-works.htm.

[16] Google street view, how it works. https://www.google.com/intl/it/streetview/how-it-works/.

[17] Dijkstra least cost path algorithm. https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/.

[18] A* algorithm. https://brilliant.org/wiki/a-star-search/.

[19] Opentripplanner. https://www.opentripplanner.org/, .

[20] Opentripplanner deployments. https://docs.opentripplanner.org/en/dev-2.x/Deployments/, .

[21] Greenpath app. https://green-paths.web.app/?map=streets.

[22] Greenpath server github. https://github.com/DigitalGeographyLab/green-paths.

[23] Cleanairroutefinder. https://www.london.gov.uk/programmes-strategies/environment-and-climate-change/pollution-and-air-quality/clean-air-route-finder.

[24] Typescript documentation. https://www.typescriptlang.org/.

[25] React documentation. https://react.dev/learn.

[26] Redux documentation. https://redux.js.org/.

[27] Mapbox platform. https://www.mapbox.com/, .

[28] Mapbox guida introduttiva per lo sviluppo. https://docs.mapbox.com/mapbox-gl-js/guides/.

[29] Guida mapbox, sezione layers. https://docs.mapbox.com/style-spec/reference/layers/, .

[30] Guida mapbox, sezione risorse. https://docs.mapbox.com/style-spec/reference/sources/, .

[31] Mapbox studio, guida. https://docs.mapbox.com/studio-manual/guides/, .

[32] Flask restful, guida. https://flask-restful.readthedocs.io/en/latest/, .

[33] Flask socket-io, guida. https://flask-socketio.readthedocs.io/en/latest/, .

[34] Pip, documentation. https://pip.pypa.io/en/stable/.

[35] Jinga2 templating, guida. https://jinja.palletsprojects.com/en/3.1.x/.

[36] Flask templating, guida. https://flask.palletsprojects.com/en/2.3.x/tutorial/templates/, .

[37] Specifica gtfs. https://gtfs.org/schedule/reference/.

[38] Opentripplanner, guida. https://docs.opentripplanner.org/en/v2.4.0/, .

[39] Specifica pbf. https://wiki.openstreetmap.org/wiki/PBF_Format.

[40] Openstreetmap. https://wiki.openstreetmap.org/w/images/1/19/Libretto_introduzione_a_OpenStreetMap_2017_09.pdf, .

[41] Geojson, archivi. https://web.archive.org/web/20180908194630/http://lists.geojson.org/pipermail/geojson-geojson.org/2007-March/thread.html, .

[42] Graphml, specifica. http://graphml.graphdrawing.org/specification.html.

[43] Docker, documentazione. https://docs.docker.com/, .

[44] Dockerhub, pagina iniziale. https://hub.docker.com/, .

[45] Sam Newman. *Building Microservice*. O'Reilly, 2015.

[46] Igraph library, documentation. https://igraph.org/python/api/0.9.6/.

[47] Portale estratti dati open street map italia. https://osmit-estratti.wmcloud.org/.

[48] Open street map italia, sito web. https://osmit.it/, .

[49] Osmium, libreria python, sito web. https://osmcode.org/pyosmium/, .

[50] Pandas, libreria python, sito web. https://pandas.pydata.org/.

[51] Geppandas, libreria python, sito web. https://geopandas.org/en/stable/index.html, .

[52] Otp, fork by digital geography lab. https://github.com/DigitalGeographyLab/OpenTripPlanner/, .

[53] Digital geography lab, sito web. https://www.helsinki.fi/en/researchgroups/digital-geography-lab.

[54] Epsg 32632. https://epsg.io/32632, .

[55] Igraph library, shortest path documentation. https://igraph.org/python/api/0.9.6/igraph._igraph.GraphBase.html#get_all_shortest_paths.

[56] nginx, website. https://www.nginx.com/.

[57] mongo, website. https://www.mongodb.com/it-it.

[58] Otp 2 documentation, dataoverkay section. https://docs.opentripplanner.org/en/v2.4.0/sandbox/DataOverlay/.

[59] Latex.

[60] Flux documentation. https://facebookarchive.github.io/flux/.

[61] Mapbox studio, guida. https://docs.mapbox.com/help/getting-started/mapbox-data/, .

[62] Flask, guida. https://flask.palletsprojects.com/en/3.0.x/quickstart/, .

[63] Epsg 32632. https://epsg.io/4326, .

[64] rfc7519, specifica. https://datatracker.ietf.org/doc/html/rfc7519.