ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

INGEGNERIA E SCIENZE INFORMATICHE

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Campus di Cesena

Tesi di Laurea Sperimentale in

Sistemi Distribuiti

MONITORAGGIO DINAMICO DI ENTITÁ GEO-LOCALIZZATE: UN'APPLICAZIONE WEB MODULARE

DYNAMIC MONITORING OF GEO-LOCALIZED ENTITIES: A MODULAR WEB APPLICATION

Relatore:
Prof. Andrea Omicini
Co-Relatore:
Prof. Stefano Mariani

Candidato:
Michele Donati
Matr. 0000796431

ANNO ACCADEMICO 2019/2020

Indice

Capitolo 1

Introduzione

Sommario

1.1 Nascita del progetto

Il progetto nasce come re-ingegnerizzazione ed estensione di un applicativo realmente usato dai medici di una clinica di Barcellona nell'ambito del progetto europeo CONNECARE ormai concluso [1]. CONNECARE è nato con l'intento di fornire un efficace supporto alle decisioni per medici e operatori sanitari, i quali devono occuparsi di pazienti con malattie e disturbi cronici. L'obiettivo ultimo, è quello di semplificare la coordinazione tra gli specialisti, i medici di base, i servizi sociali, e ovviamente i pazienti, al fine di diminuire drasticamente i tempi di reazione in caso di emergenza e quindi, di conseguenza, anche i costi che ne derivano. Il progetto svolto nell'ambito di questa tesi, prevede la re-ingegnerizzazione del software CONNECARE attraverso le più moderne tecnologie di sviluppo, e allo stesso tempo eseguire un ampliamento delle sue funzionalità e, soprattutto, del suo dominio applicativo, in un'ottica altamente dinamica in termini di configurabilità dell'applicazione da parte dell'utente, rendendolo uno strumento utilizzabile in molteplici contesti e scenari.

1.2 Obiettivo

L'obiettivo primario di questo progetto di tesi è quello di realizzare uno strumento grazie al quale un utente senza approfondite conoscenze informatiche, ad esempio un medico, possa riuscire, nel modo più semplice e configurabile possibile, a monitorare ed analizzare il comportamento e/o le caratteristiche di entità geo-localizzate su una mappa.
Il progetto, infatti, si pone lo scopo di fornire uno strumento per il monitoraggio dinamico di entità geo-localizzate, sfruttando le più moderne tecnologie per lo sviluppo di applicazioni web modulari, come *Express* [2], *Vue.js* [3], *Node.js* [4] e servizi esterni. L'obiettivo principale di questa applicazione è di permettere ad un suo utilizzatore di poter scegliere la tipologia di entità che desidera monitorare (e.g. veicoli, pazienti di un ospedale), configurando i criteri e i parametri per i quali queste entità vengono monitorate e/o evidenziate a schermo. La scelta di questi elementi, in fase di configurazione, permette anche, in maniera automatica e dinamica, la costruzione di un'interfaccia utente perfettamente e strettamente adattata alle esigenze espresse dall'utilizzatore. L'applicazione permette anche l'inserimento, in maniera arbitraria, di *geo-fences* [5], ovvero triggers geo-localizzati sensibili al posizionamento ed ai movimenti delle entità sulla mappa, le quali vengono continuamente geograficamente constestualizzate tramite un processo di *reverse-geocoding* [6].

Il motivo che ha condotto lo sviluppo di questo progetto è quello di fornire un supporto alle decisioni efficace ed intuitivo per qualsiasi tipologia di utente. Questa tipologia, per esempio, può andare dal medico all'interno di una struttura ospedaliera che deve occuparsi di determinati pazienti, fino ad arrivare anche ad agenti delle forze dell'ordine che devono gestire un alto tasso

di traffico veicolare. In questo periodo particolare, segnato dalla pandemia di COVID-19 [7], un caso d'uso interessante è sicuramente quello relativo al monitoraggio degli assembramenti in aree sensibili. Gli scenari applicativi sono limitati soltanto dal dominio applicativo.

1.2.1 Panoramica strutturale

L'applicazione ha una struttura costituita da un *frontend*, un *backend* e da *entità* simulate, in questo caso *Veicoli* di ogni tipologia e *Pazienti* ospedalieri.
Nel frontend l'utente ha la possibilità di configurare a piacere, conoscendo il dominio applicativo e tramite un file di configurazione, la tipologia di entità che desidera visualizzare, gli indici caratteristici e i relativi criteri per i quali le entità possono essere evidenziate, e i triggers geo-localizzati per il monitoraggio di attività in determinati punti sulla mappa. Alla fine di questa fase preparatoria l'utente può visualizzare su una mappa la posizione geografica delle entità tramite dei markers, inclusi pop-ups contenenti informazioni aggiuntive. Mediante una serie di bottoni è possibile selezionare il tipo di indice caratteristico delle entità, visualizzabile tramite il cambiamento di colore dei markers, con lo scopo di evidenziare all'occhio una specifica caratteristica dell'entità, ad esempio il livello di carburante dei veicoli o la saturazione d'ossigeno dei pazienti. Si può inoltre visualizzare una tabella in cui sono elencati i dati relativi alle entità e i valori degli indici, e delle aree di testo in cui vengono elencate le entità che attivano i triggers geo-localizzati.

Capitolo 2

Dominio Applicativo

Sommario

2.1 Entità

Per lo sviluppo dell'applicazione e delle sue funzioni è stato necessario disporre di dati relativi alle entità da modellare. In questo caso specifico sono state scelte le entità *Veicolo*, inteso come un qualsiasi tipo di veicolo su ruote motorizzato e omologato per la guida, e *Paziente*, inteso come un qualsiasi tipo di persona in cura (locale o domiciliata) presso un generico servizio sanitario, ospedale pubblico o privato accreditato.

Il motivo per il quale sono state scelte tali entità deriva, nel caso dei Pazienti, dall'influenza che questo lavoro ha avuto dal progetto *CONNECARE* [1], che prevede la mappatura di pazienti e la loro area abitativa in modo da avere una loro geo-localizzazione, arricchita dal loro stato clinico e dal livello di allerta calcolato automaticamente dal sistema. La differenza è che nel caso CONNECARE, i pazienti sono statici, mentre in questo progetto possono essere anche in continuo movimento.

La scelta di modellazione delle entità in Veicoli è nata anche per questo motivo, ovvero il fatto che effettivamente i pazienti possono essere considerati come unità statiche, nel caso non si avessero a disposizione le loro coordinate gps in real-time. I veicoli, invece vengono considerati esclusivamente come entità mobili, andando a completare un possibile scenario applicativo d'uso per l'utente, offrendogli la possibilità di interagire con entità molto diverse tra loro, non solo nelle caratteristiche intrinseche e descrittive ma anche nel comportamento.

Le informazioni caratteristiche di ciascuna entità non sono state ottenute in modo diretto da una base di dati esistente, quindi le informazioni utilizzate dall'applicazione non sono reali ma simulate tramite librerie JavaScript in modo realistico e pseudo-casuale.

Dati pre-esistenti sono stati trovati solo per quanto riguarda il comportamento geo-spaziale delle entità, ovvero il loro movimento. A tal proposito, sono state prelevate numerose tracce GPS da un database Microsoft, registrate da 182 utenti di Pechino, in Cina, che dal periodo aprile 2007-agosto 2012, hanno partecipato ad una iniziativa di ricerca chiamata *Microsoft Research Asia Geolife Project* [2].

Una traiettoria GPS di questo set di dati è rappresentata da una sequenza di punti con data e ora, ciascuno dei quali contiene le informazioni di latitudine, longitudine e altitudine. Questo set di dati contiene 17.621 traiettorie con una distanza totale di circa 1,2 milioni di chilometri e una durata totale di oltre 48.000 ore. Queste traiettorie sono state registrate da diversi registratori GPS e telefoni GPS e hanno una varietà di frequenze di campionamento. Il 91 percento delle traiettorie viene registrato in una rappresentazione densa, ad esempio ogni 1 - 5 secondi od ogni 5 - 10 metri per punto. Questo set di dati ha ricodificato un'ampia gamma di movimenti all'aperto degli utenti, inclusi non solo routine di vita come tornare a casa e andare al lavoro, ma anche alcuni intrattenimenti e attività sportive, come shopping, visite turistiche, ristoranti, escursioni e ciclismo. Questo set di dati può essere utilizzato in molti campi di ricerca, come il mining di modelli di mobilità, il riconoscimento dell'attività dell'utente, i social network basati

sulla posizione, la privacy della posizione e la raccomandazione sulla posizione [3] [4] [5].

La distribuzione delle distanze e delle durate delle traiettorie sono presentate in Figura 2.1 ed in Figura 2.2.

Figura 2.1: Distribuzione delle traiettorie rispetto alla distanza

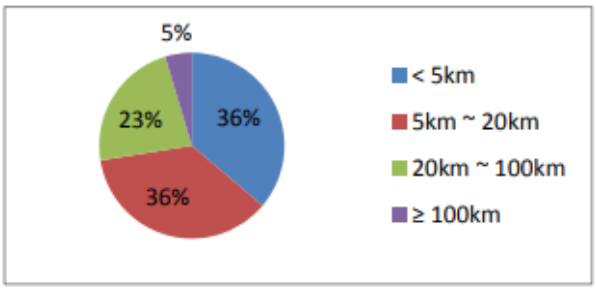
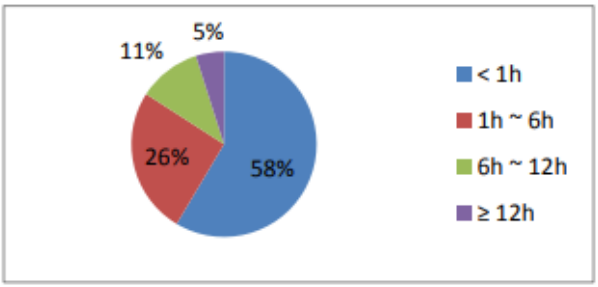
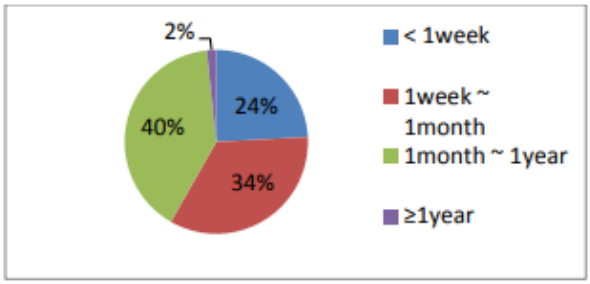


Figura 2.2: Distribuzione delle traiettorie rispetto alla durata



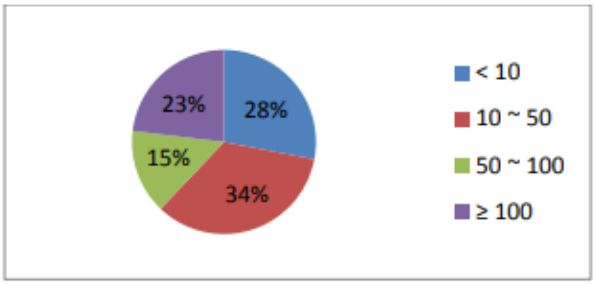
La distribuzione del periodo di tempo di permanenza degli utenti all'interno del progetto e la distribuzione delle traiettorie raccolte da ciascun utente sono evidenziate in Figura 2.3 ed in Figura 2.4

Figura 2.3: Distribuzione della permanenza degli utenti nel progetto



Ogni singolo repository di questo dataset ha in memoria i file GPS di un singolo utente, convertiti nel formato PLT. Ogni file PLT contiene una singola traiettoria ed è identificata dal suo tempo di inizio. Per evitare possibili equivoci in relazione alla time-zones, viene usato GMT nelle proprietà di data/tempo di ciascun punto. Di seguito, in Figura 2.5, si mostra nel dettaglio

Figura 2.4: Distribuzione delle traiettoria raccolte da ciascun utente



il formato di ciascun file PLT e un esempio concreto, presi dalla documentazione disponibile [2].

Figura 2.5: Formato dei file di traiettoria

PLT format:

Line 1…6 are useless in this dataset, and can be ignored. Points are described in following lines, one for each line.

Field 1: Latitude in decimal degrees.

Field 2: Longitude in decimal degrees.

Field 3: All set to 0 for this dataset.

Field 4: Altitude in feet (-777 if not valid).

Field 5: Date - number of days (with fractional part) that have passed since 12/30/1899.

Field 6: Date as a string.

Field 7: Time as a string.

Note that field 5 and field 6&7 represent the same date/time in this dataset. You may use either of them.

Example:

39.906631,116.385564,0,492,40097.5864583333,2009-10-11,14:04:30

39.906554,116.385625,0,492,40097.5865162037,2009-10-11,14:04:35

Di seguito vengono descritte più nel dettaglio le singole tipologie di entità modellate scelte per questo progetto.
Ogni entità e le sue caratteristiche intrinseche sono generate a run-time in modo realistico e pseudo-randomico tramite l'utilizzo della libreria JavaScript *Faker.js* [6].
Si tratta di una libreria che mette a disposizione delle API JavaScript per dare la possibilità di generare una massiccia quantità di dati simulati relativi a diversi ambiti, tra cui:

- Indirizzi
- Date
- Immagini
- Nomi
- Numeri telefonici ed e-mails
- Veicoli

- ... e numerosi altri.

Faker.js permette inoltre la *generazione di dati localizzata*, ovvero la possibilità di generare dati realistici sulla base di una nazionalità. Al momento sono disponibili oltre venti localizzazioni diverse, e dato che li scenari d'uso di questo progetto sono ambientati a Pechino, a casua delle tracce GPS associate, sarà utilizzata la localizzazione cinese.

2.1.1 Veicoli

L'entità *Veicolo* è stata modellata in modo tale che ad ognuno di essi siano associati:

- Identificativo
- Tipo di veicolo
- Modello
- Casa Manifatturiera
- Targa
- Percentuale di carburante
- Percentuale di rumore generato
- Percentuale di vibrazioni generate
- Percentuale di ergonomia

Le prime cinque caratteristiche sono state generate tramite la libreria *faker.js*, mentre le rimanenti sono state generate con API separate e implementate appositamente.
Le politiche di implementazione di queste API sono state dettate dagli studi effettuati sulla documentazione ufficiale relativa alle percentuali di rumore generate dai veicoli [7] [8], alle percentuali di vibrazioni generate dai veicoli [9] [10] e alle percentuali di ergonomia [11].

Rumore

Il motivo per il quale è stato scelto il *rumore* come caratteristica da modellare all'interno dell'entità veicolo è il fatto che il continuo aumento dei rumori prodotti dalla civiltà moderna è giunto ad un punto tale da interagire con lo stato di salute dell'uomo [7].
I rumori dovuti al sempre più intenso traffico stradale ed aereo, dalle industrie o anche solo dagli elettrodomestici, contribuiscono ad una forma di inquinamento ambientale di vaste proporzioni. Inoltre il rumore è causa di precoce invecchiamento in 30 casi su 100 e che nelle grandi città esso riduce la vita media dell'uomo di alcuni anni.
Si individuano 3 classi di fenomeni acustici:

- *fenomeni desiderati*: attraverso i quali l'organo uditivo acquisisce informazioni come la parola, la musica o anche meno gradevoli come gli allarmi;
- *fenomeni non desiderati*: generano fastidio nell'organo uditivo (rumori in genere o suoni che non si è intenzionati ad ascoltare):
- *fenomeni neutri*: campi acustici naturali (vento, mare, pioggia) o artificiali che generano rumore di fondo.

Per definizione, il *rumore* è una qualunque emissione sonora che provochi sull'uomo effetti indesiderati, disturbanti o dannosi o che determini un qualsiasi deterioramento dell'ambiente. I danni prodotti dal rumore possono essere anche a livello di disturbi del sistema nervoso, cardiovascolare e neuro-psichico. Di seguito si mostra la Figura 2.6 dove sono elencate linee guida a questo proprosito.

Figura 2.6: Intensità ed effetti del rumore

INTENSITÀ MEDIA dB(A)	EFFETTI
da 30 a 65 dB(A)	fastidio
da 50 a 85 dB(A)	disturbo
oltre 80 dB(A)	danno

Questi sono i motivi che hanno portato alla scelta di questo elemento come indice di monitoraggio delle entità nell'applicazione, ovvero dare la possibilità all'utente di monitorare i livelli di rumore delle entità veicolari al fine di evitare, per esempio, zone di traffico troppo intense con alti livelli di disturbo acustico. Uno strumento di questo tipo sarebbe perfetto in casi d'uso che prevedono il coinvolgimento di persone con alta sensibilità uditiva o con problemi/patologie all'organo uditivo.
Di seguito, in Figura 2.7, si mostrano degli esempi di intensità di rumore prodotto da diversi elementi.

Figura 2.7: Sorgenti e intensità di rumore

Sorgente	Intensità acustica	Livello (dB)
Soglia di udibilità	1	0
Respiro normale	10	10 (appena udibile)
Stormire di foglie	10^2	20
Voce bisbigliata	10^3	30 (molto quieto)
Ristorante tranquillo	10^4	40
Ufficio silenzioso	10^5	50
Conversazione tra 2 persone	10^6	60
Interno di ufficio rumoroso	10^7	70 (disturbante)
Traffico stradale rumoroso	10^8	80
Autotreno (a 15 m)	10^9	90 (pericolo esposizioni prolungate)
Metropolitana	10^{10}	100
Complesso rock	10^{11}	110
Martello pneumatico	10^{12}	120 (soglia del dolore)
Fuoco di mitragliatrice	10^{13}	130
Decollo di un piccolo aviogetto	10^{14}	140
Galleria aerodinamica	10^{15}	150
Decollo di un grande aviogetto	10^{17}	170

All'interno dell'applicazione le soglie di rumore sono simulate al punto da avere veicoli con emissioni di disturbi acustici differenti e che rispecchino in modo realistico le soglie sopra presentate.

Vibrazioni

Il motivo per il quale sono state scelte le *vibrazioni* come caratteristica da modellare all'interno dell'entità veicolo è il fatto che esse fanno parte di quei rischi che tendono ad essere sottovalutati perchè non producono menomazioni immediate, ma manifestano i loro effetti sulla salute solo dopo diversi anni e con differente intensità da soggetto a soggetto [9].
In particolare vengono prese in considerazione le vibrazioni trasmesse al corpo intero, ovvero le vibrazioni meccaniche che comportano rischi per la salute e la sicurezza. L'esposizione a questo tipo di vibrazioni si riscontra in lavorazioni a bordo di mezzi di movimentazione usati in industria e in agricoltura, mezzi di trasporto e, in generale, macchine vibranti. L'esposizione a vibrazioni è tipicamente associata alla guida del mezzo e quindi avviene mediante il contatto con il sedile.
A(8) è il descrittore che fornisce una stima dell'energia a cui viene sottoposta la persona, cumulata nell'intera giornata.
Esistono tre diverse classi di rischio:

- $A(8) \leq 2{,}5$ m/s^2 - Rischio minimo: controlli sanitari non necessari;
- $2{,}5 < A(8) \leq 5$ m/s^2 - Rischio presente: controlli sanitari necessari;
- $A(8) > 5$ m/s^2 - Rischio inacettabile: misure immediate.

Questi sono i motivi che hanno portato alla scelta di questo elemento come indice di monitoraggio delle entità nell'applicazione, ovvero dare la possibilità all'utente di monitorare i livelli

di vibrazioni delle entità veicolari al fine di evitare, per esempio, mezzi di trasporto che ne generano una quantità eccessiva. Uno strumento di questo tipo sarebbe perfetto in casi d'uso che prevedono il coinvolgimento di persone con problemi/patologie legate alla forte e duratura esposizione a questo elemento.
All'interno dell'applicazione le soglie di vibrazioni sono simulate al punto da avere veicoli con emissioni differenti e che rispecchino in modo realistico le soglie sopra presentate.

Ergonomia

Il concetto di *ergonomia* è nato e si è diffuso come l'obiettivo di raggiungere la massima efficienza dell'ambiente, del prodotto o del sistema garantendo la massima efficienza e il massimo benessere dei suoi utilizzatori [11].
Gli obiettivi dell'ergonomia possono essere sintetizzati in:

- *obiettivi operativi di base*: riduzione degli errori, incremento della sicurezza, incremento delle prestazioni del sistema;
- *obiettivi relativi alla affidabilità, alla durabilità e all'utilità*: incremento dell'affidabilità e della durata dei sistemi, riduzione delle richieste agli operatori e della necessità di training;
- *obiettivi relativi agli utenti e agli operatori*: miglioramento dell'ambiente di lavoro, miglioramento del comfort, incremento della facilità d'uso, dell'accettabilità psicologica, della gradevolezza estetica, riduzione della fatica e dello stress fisico, riduzione della monotonia;
- *altri obiettivi*: incremento dell'economia di produzione, riduzione degli sprechi di tempo e di risorse.

Non essendoci un'unità di misura universale adottata per indicare un determinato livello di ergonomia, in questo caso di un veicolo, è stato scelto di utilizzare una semplice notazione in percentuale, che rispecchi quanto gli obiettivi sopra citati sono soddisfatti in relazione a quella determinata entità.
Questi sono i motivi che hanno portato alla scelta di questo elemento come indice di monitoraggio delle entità nell'applicazione, ovvero dare la possibilità all'utente di monitorare la percentuale di ergonomia delle entità veicolari al fine di poter scegliere, per esempio, i mezzi di trasporto che più lo soddisfano.

2.1.2 Pazienti

L'entità *Paziente* è stata modellata in modo tale che ad ognuno di essi siano associati:

- Identificativo
- Nome

- Cognome
- Numero di telefono
- E-mail
- Lace
- Charlson
- GMA
- ASA
- Barthel

Le prime cinque caratteristiche sono state generate tramite la libreria *faker.js*, mentre le rimanenti sono state generate con API separate e implementate appositamente.
Le politiche di implementazione di queste API sono state dettate dagli studi effettuati sulla documentazione ufficiale relativa agli indici di rischio *Lace* [12] [13], *Charlson* [14], *Barthel* [15], *GMA* [16] [17] ed *ASA* [18].

Lace

La letteratura medica fa riferimento alle riammissioni ospedaliere come eventi ”comuni, costosi e spesso prevenibili” per quanto riguarda trattamenti medici di breve durata. Al giorno d'oggi sono ampiamente considerati come parametro della qualità dell'assistenza. Tuttavia, i tassi di riammissione in ospedale sono entrati a far parte della classifica relativa ai tassi di mortalità e complicanze nel mondo delle ”misurazioni dei risultati della qualità dell'assistenza”. Collettivamente, sono indicatori facili da valutare e forniscono una base per confrontare le misure delle prestazioni ospedaliere.
Per affrontare questo problema, gli ospedali hanno avviato strategie di riduzione della riammissione. Una di queste, promossa dall'*Institute of Health Improvement*, consiste nell'utilizzare uno strumento di stratificazione del rischio per identificare riammissioni prevenibili. L'indice ”LACE” è uno di questi strumenti ampiamente utilizzati.
L'indice LACE identifica i pazienti a rischio di riammissione o morte entro trenta giorni dalla dimissione. Come si può osservare dalla Figura 2.8, esso incorpora quattro parametri.

Figura 2.8: Indice Lace per la quantificazione del rischio di decesso o re-ricovero non pianificato entro 30 giorni dalla dimissione ospedaliera

Variabile	Valore	Punteggio
Durata della degenza, Length of stay "L"	<1	0
	1	1
	2	2
	3	3
	4-6	4
	7-13	5
	≥14	7
Ammissione tramite PS, Acute emergent admission "A"	Si	3
Comorbilità, Charlson Comorbidity Index Score "C"	0	0
	1	1
	2	2
	3	3
	≥4	5
Accessi al PS nei precedenti 6 mesi, Emergency "E"	0	0
	1	1
	2	2
	3	3
	≥4	4

Un punteggio da 0-4 indica un rischio basso di riammissione; da 5-9 moderato; maggiore o uguale a 10 è un rischio alto di riammissione.
Il motivo per il quale è stato scelto questo elemento come indice di monitoraggio delle entità nell'applicazione è il fatto di dare la possibilità ad un utente, per esempio un operatore sanitario, di monitorare in tempo reale questo indice di rischio per prevenire costose riammissioni in ospedale di tali pazienti.

Charlson

L'Indice di Comorbidità di Charlson è stato sviluppato allo scopo di classificare le comorbidità che potrebbero alterare il rischio di mortalità. Si definisce con il termine comorbidità "l'esistenza o il verificarsi di una qualunque entità aggiuntiva durante il decorso clinico di un paziente con una malattia" in poche parole la presenza nello stesso momento di più malattie che possono avere differenti livelli di gravità ed effetti differenti sulla salute dell'individuo [14].
Nella forma classica sono considerate 19 condizioni mediche (16 malattie di cui 3 sono stratificate secondo la gravità) a cui viene attribuito un peso variabile da 1 a 6 con un punteggio finale da 0 a 33. Il punteggio attribuito è correlato alla gravità della condizione, in particolare alla probabilità di morte entro un anno: la somma dei punteggi delle singole comorbidità permette di valutare l'impatto delle stesse sulla gravità della malattia. In seguito l'indice è stato aggiustato considerando anche l'età tra i fattori di rischio ottenendo così un indice combinato età-comorbidità da 0 a 37.
Il motivo per il quale è stato scelto questo elemento come indice di monitoraggio delle entità nell'applicazione è il fatto di dare la possibilità ad un utente, per esempio un operatore sani-

tario, di monitorare in tempo reale questo indice per poter controllare lo stato di salute e il comportamento di pazienti a rischio, intervenendo tempestivamente in caso di bisogno.

Barthel

L'indice Barthel viene valutato sia all' ingresso che alla dimissione dall'ospedale. In esso vengono considerati 10 punti relativi al movimento, alla deambulazione, all'igiene personale, alla capacità di alimentarsi, alla continenza intestinale ed urinaria. Il punteggio che ne deriva esprime il grado di assistenza che le condizioni del paziente richiedono nelle attività quotidiane. Il valore zero indica un paziente totalmente dipendente, mentre il valore 100, che rappresenta il massimo, indica un paziente pienamente autonomo.
La valutazione costituisce la base di discussione del team di lavoro cui spetta il compito di impostare il programma riabilitativo.
La scala di valutazione è, inoltre, strumento fondamentale per la valutazione del livello di performance della struttura riabilitativa anche in confronto con altre [15].
All'interno dell'applicazione le soglie degli indici sono simulati al punto da avere pazienti con punteggi differenti e che rispecchino in modo realistico le soglie sopra presentate.

GMA

GMA (Adjusted Morbidity Groups) è uno strumento per la valutazione del rischio per la salute basata sulla popolazione sviluppato in Catalogna nel 2015 [19], e consente alla popolazione di essere classificata in 6 gruppi di morbilità e, a sua volta, divisa in 5 livelli di complessità, insieme a un gruppo di popolazione sana. Nella stratificazione vengono presi in considerazione elementi come: mortalità, rischio di ospedalizzazione, e visite in pronto soccorso.
Successivamente, la popolazione è suddivisa in 31 categorie suddivise in 7 macro categorie:

- Persone sane;
- Malattie acute;
- Malattie croniche al primo stadio;
- Malattie croniche al secondo stadio;
- Malattie croniche al terzo stadio;
- Malattie croniche al quarto stadio o più;
- Cancro attivo;

Può essere utilizzato per stratificare la popolazione e identificare delle popolazioni target, in particolare vengono considerati questi elementi per la stratificazione finale:

- Età;

- Sesso;
- Morbilità;
- Fattori sociali;
- Fattori economici;
- Accessibilità

Infine, si può affermare che GMA ha buoni risultati esplicativi e predittivi nell'uso degli indicatori delle risorse sanitarie.

ASA

In passato i farmaci per l'anestesia potevano determinare effetti nocivi su diversi organi come il fegato, il rene o sull'apparato cardiovascolare. Fortunatamente la tecnologia e la chimica hanno fatto notevoli progressi e oggi si ha a disposizione una quantità di farmaci oltremodo sicuri [18]. La sicurezza dipende non solo dai farmaci e dagli strumenti elettromedicali utilizzati in ospedale, ma naturalmente dallo stato clinico del paziente. L'età anagrafica, invece, incide in misura molto minore: un paziente anziano può presentarsi in ottime condizioni a differenza di un giovane adulto con patologie di vario genere.
Universalmente è riconosciuta la classificazione redatta dalla Società americana di anestesiologia (ASA – American Society of Anesthesiologists), secondo cui i rischi sono classificati in:

- ASA 1: Paziente sano;
- ASA 2: Paziente con malattia lieve senza limitazioni funzionali;
- ASA 3: Paziente con malattia grave con modica limitazione;
- ASA 4: Paziente con malattia grave con limitazione importante.

All'interno dell'applicazione le soglie degli indici sono simulati al punto da avere pazienti con classi ASA differenti e che rispecchino in modo realistico quelle sopra presentate.

2.2 Scenari

Allo stato attuale un utente medio potrebbe utilizzare il sistema tramite compilazione di un file di configurazione. La traduzione del suddetto file in un'interfaccia utente di configurazione più intuitiva è sicuramente nella visione globale dell'intero progetto, ma lasciata a sviluppi futuri. Infatti il nucleo pulsante di questo lavoro è considerato come la dinamicità e la libertà d'uso, ovvero il fatto di dare la possibilità all'utente di scegliere in qualunque momento *cosa* visualizzare e *come* visualizzarlo, dando a quest'ultima opzione il massimo della libertà possibile.

Presumendo che un utente compili il file di configurazione, esso potrebbe, per esempio, scegliere di visualizzare la regione geografica che desidera (e.g. Pechino, Cina), le entità che desidera monitorare (e.g. Veicoli), gli indici di selezione/evidenziazione di queste entità (e.g. Rumore, Vibrazioni, Gasolio, Ergonomia) e i loro criteri (e.g. terzili, quartili, o espressioni del tipo

$$carburante \leq 30\%$$

). Potrebbe inoltre indicare quali caratteristiche delle entità visualizzare nella tabella dell'interfaccia (e.g. Tipo, Modello, Casa Manifatturiera, Targa, ecc.). Tutto questo, per esempio, al fine di monitorare lo stato del traffico della città e decidere di conseguenza quali percorsi evitare.

Un altro possibile scenario applicativo legato alle entità *Pazienti*, potrebbe essere quello relativo ad un operatore sanitario che configura l'applicazione per monitorare il luogo in cui si trovano i pazienti più a rischio, se sono troppo vicini a luoghi e/o altre persone in cui vi sono probabilità che la loro patologia possa peggiorare.

Capitolo 3

Dominio Tecnologico

Sommario

3.1 Framework Vue

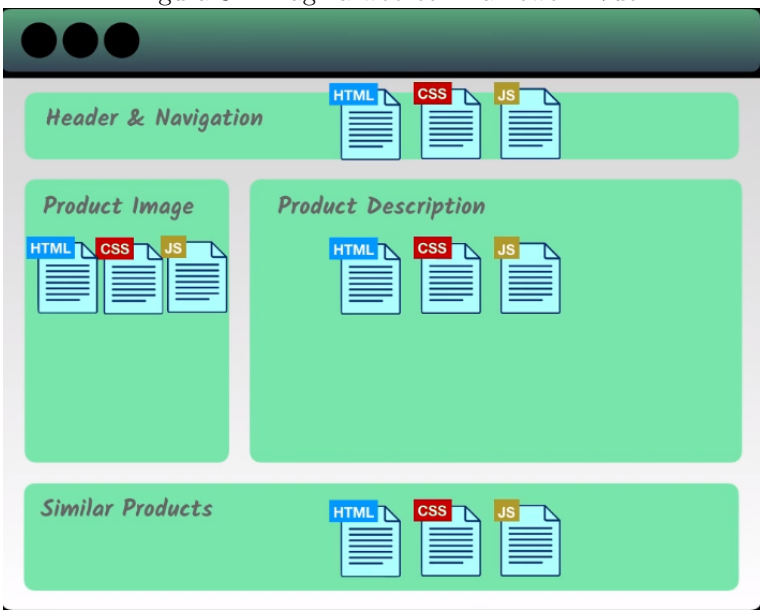
Negli ultimi dieci anni le nostre pagine web sono diventate più dinamiche e potenti grazie a *JavaScript*. Molto del codice che normalmente si trovava sul lato server è stato trasferito sul lato browser, connesso con vari files *html* e *CSS* privi di un'organizzazione formale. Questo è il motivo che ha portato molti sviluppatori ad utilizzare frameworks come *Angular*, *React* o *Vue*.

Vue.js si differenzia dagli altri framework sopra citati per [1]:

- *performance*, in quanto implementa una modalità particolare di rendering basato sui *document fragments*;
- *leggibilità*, soprattutto confrontato con i template JSX che non utilizza html standard;
- *flessibilità*, per via della natura progressiva di Vue.js soprattutto confrontato con framework monolitici come Angular.

Vue è un framework JavaScript accessibile, versatile e performante che permette di creare una base di codice più manutenibile e verificabile. Si tratta di un framework JavaScript progressivo, stando ad indicare che, se si ha un'applicazione esistente lato server, è possibile collegare Vue anche a solo una parte di tale applicazione che ha bisogno di un'esperienza più interattiva e ricca.
Alternativamente, se si vuole costruire più business logic nel frontend fin dall'inizio, Vue possiede le librerie di core e l'ecosistema adatto per la scalabilità. Come altri framework di frontend, Vue consente di prendere una pagina web e dividerla in componenti riutilizzabili, ognuno con i propri HTML, CSS e JavaScript necessari per il rendering di quella parte della pagina (Figura 3.1).

Figura 3.1: Pagina web con framework Vue



La caratteristica principale di Vue è il *Rendering Dichiarativo*, ovvero la possibilità di inserire gli elementi dell'interfaccia definiti come markup, direttamente all'interno dell'HTML della pagina. Tuttavia, il concetto chiave che rende Vue incredibilmente utile consiste nel fatto che Vue è un framework *Reattivo*: quando i dati all'interno di un componente Vue cambiano, il framework si prende cura di aggiornare tutti i punti, all'interno dell'applicazone web, in cui quei dati vengono utilizzati. Esempi di questa proprietà possono essere osservati nel Capitolo 5 di questo documento, o direttamente nella documentazione del framework [2].

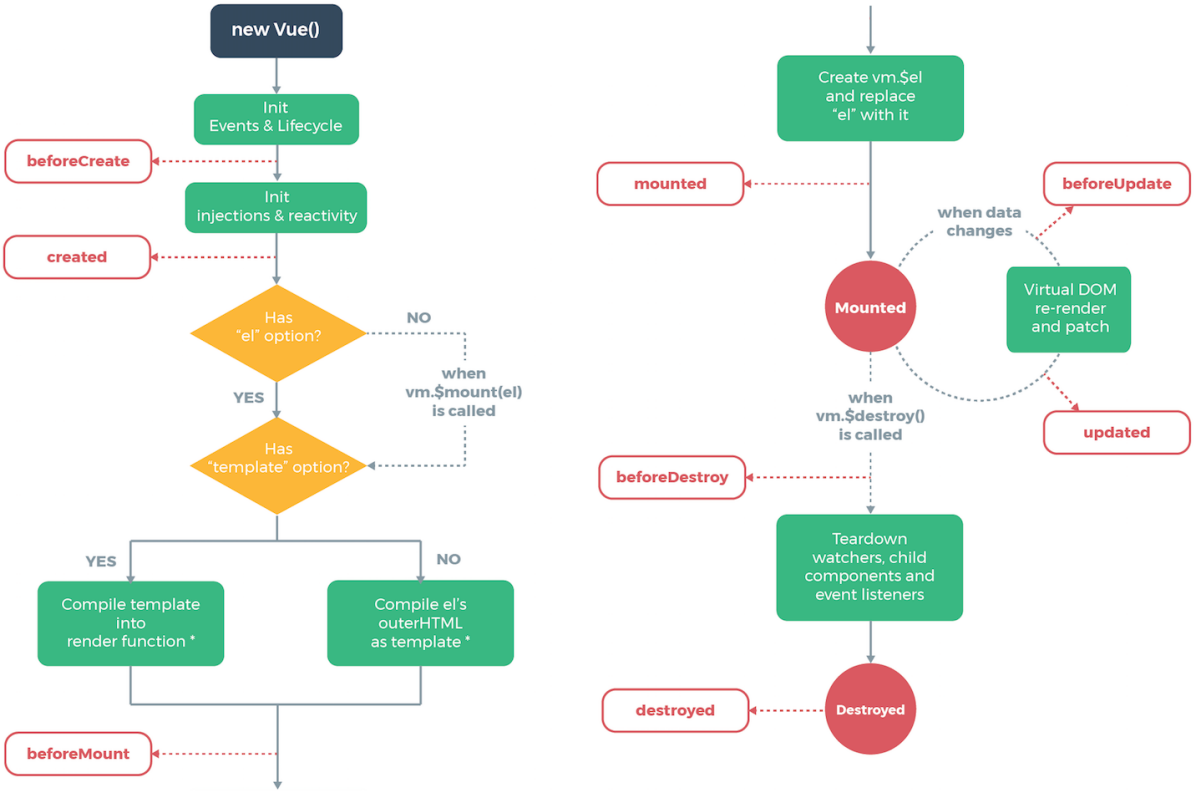
3.1.1 Il ciclo di vita di Vue

Ciascuna istanza di Vue presente nella pagina segue un ciclo di vita ben preciso ed offre la possibilità di definire degli hooks, tramite callbacks JavaScript, per ciascun cambiamento di stato dell'istanza. Come si nota dalla Figura 3.2, vi sono diversi tipi di hooks, tra cui:

- *beforeCreate*, chiamato in modo sincrono immediatamente dopo che l'istanza è stata inizializzata, prima dell'osservazione dei dati e della configurazione degli eventi.
- *created*, chiamato in modo sincrono dopo la creazione dell'istanza. In questa fase, l'istanza ha terminato l'elaborazione delle opzioni, con l'impostazione dei seguenti elementi: osservazione dei dati, proprietà di calcolo, metodi, callbacks eventi. Tuttavia, la fase *mounted* non è stata avviata e la proprietà $el non sarà ancora disponibile.
- *beforeMount*, chiamato subito prima dell'inizio della fase *mounted*: la funzione di render sta per essere chiamata per la prima volta. Questo hook non viene chiamato durante il rendering lato server.

- *mounted*, chiamato dopo che l'istanza è stata montata. Da notare che *mounted* non garantisce che siano stati montati anche tutti i componenti figlio. Questo hook non viene chiamato durante il rendering lato server.
- *beforeUpdate*, chiamato quando i dati cambiano, prima che il DOM venga aggiornato. Questo è un buon punto del lifecycle per accedere al DOM esistente prima di un aggiornamento, ad esempio per rimuovere listeners di eventi aggiunti manualmente. Questo hook non viene chiamato durante il rendering lato server, perché solo il rendering iniziale viene eseguito lato server.
- *updated*, chiamato dopo una modifica dei dati, il DOM virtuale viene nuovamente renderizzato e aggiornato. Questo hook non viene chiamato durante il rendering lato server.
- *beforeDestroy*, chiamato subito prima che l'istanza di un componente venga smontata. In questa fase l'istanza è ancora completamente funzionante.
- *destroyed*, chiamato dopo che un'istanza del componente è stata smontata. Quando viene chiamato questo hook, tutte le direttive dell'istanza del componente sono state sbloccate, tutti i listener di eventi sono stati rimossi e anche tutte le istanze del componente figlio sono state smontate.

Figura 3.2: Ciclo di vita di Vue



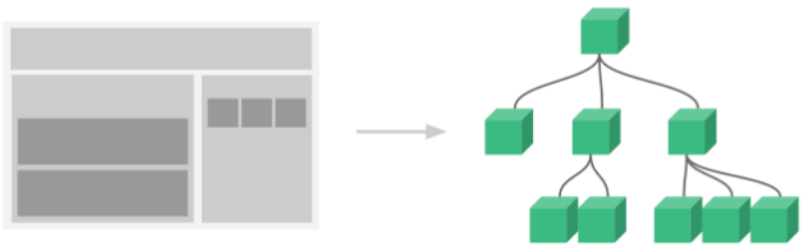
3.1.2 Template e Data-Binding

Vue utilizza un sistema di template basato su HTML e su particolari attributi chiamati *direttive*, che permettono di definire il comportamento del layer di presentazione. Il framework è in grado di interpretare il template e di compilarlo in un Virtual DOM che permette a Vue non solo di offrire reattività dei contenuti, ma anche di effettuare cambiamenti alla pagina solo se davvero necessari, evitando quindi sprechi di risorse e di tempo.
Una delle funzionalità più interessanti di Vue è il cosiddetto data-binding, ovvero la possibilità di associare un elemento presentazionale con una particolare variabile o oggetto Javascript e di essere certi che eventuali cambiamenti vengano propagati alla view esclusivamente agendo sulla variabile.
Esempi di questa proprietà sono osservabili nel Capitolo 5 di questo documento o direttamente nella documentazione ufficiale del framework[2].

3.1.3 Composizione di Componenti

Se si volesse costruire un'applicazione di grandi dimensioni, si potrebbe suddividerla in molteplici componenti e files. A tal proposito risulta utile introdurre uno dei concetti chiave di Vue, la *Composizione di Componenti*.
Il sistema dei componenti è un'astrazione che consente di creare applicazioni su larga scala composte da piccoli componenti autonomi e spesso riutilizzabili. Quasi ogni tipologia di interfaccia può essere astratta in un albero di componenti (Figura 3.3) [2].

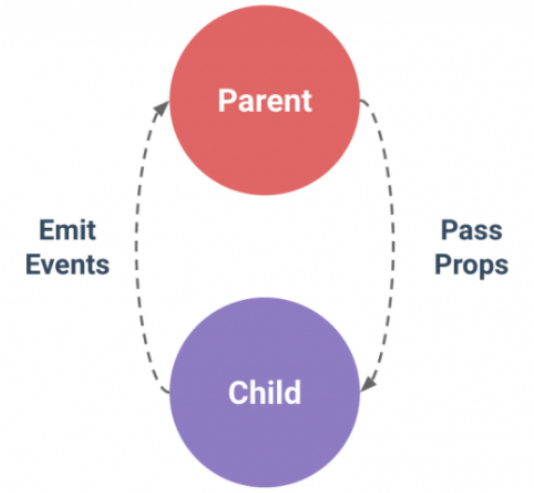
Figura 3.3: Sistema dei componenti



Il motore di Vue implementa il *one way data flow* come modalità con la quale i componenti comunicano tra di loro. In particolare, questo significa che le proprietà esterne che permettono di configurare un componente (chiamate *props*), passate ad uno di essi, possono essere modificate dall'esterno, ma non dal componente stesso, in quanto il flusso di dati è monodirezionale, dall'alto al basso.
Tuttavia, grazie alla definizione di *eventi* personalizzati, è possibile comunicare qualcosa a partire da un componente verso l'esterno. Ogni componente presenta due metodi: $on ed $emit, rispettivamente per mettersi in ascolto o per generare un evento.

Il pattern suggerito dagli sviluppatori è definito come *props down, events up* in quanto la comunicazione verso il basso è implementata tramite passaggio di props, mentre la comunicazione opposta tramite eventi (Figura 3.4).

Figura 3.4: Comunicazione tra componenti non correlati

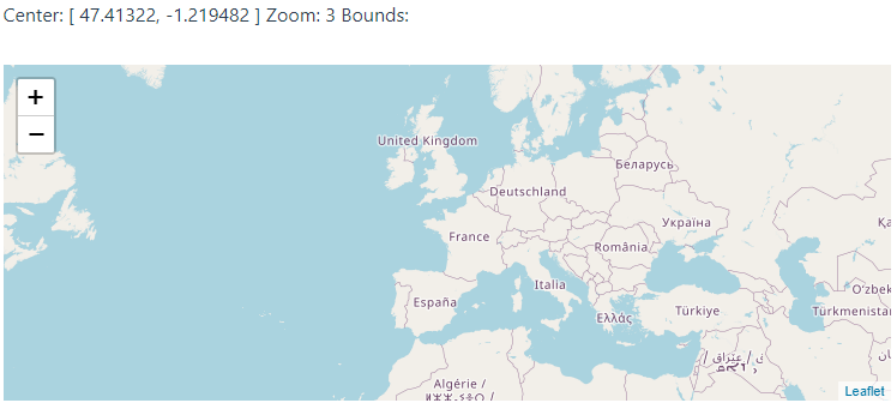


Un esempio diretto dell'utilizzo di eventi può essere osservato nel Capitolo 5 di questo documento.

3.2 Vue2Leaflet

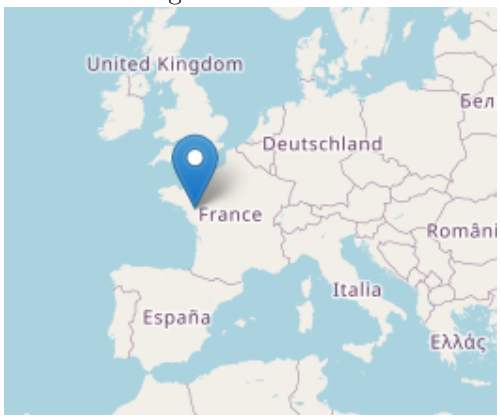
Leaflet è una libreria JavaScript, con API ben documentate, ampiamente utilizzata per la creazione di mappe interattive per applicazioni web [3].
Vue2Leaflet è una libreria wrapper per Leaflet, realizzata per incapsulare molte delle sue funzionalità in una serie di componenti Vue [4].
Le funzionalità di questa libreria permettono di configurare la mappa come componente interattivo, impostando la vista di default e lo zoom iniziale al caricamento della pagina web (Figura 3.5).

Figura 3.5: Centro e Zoom



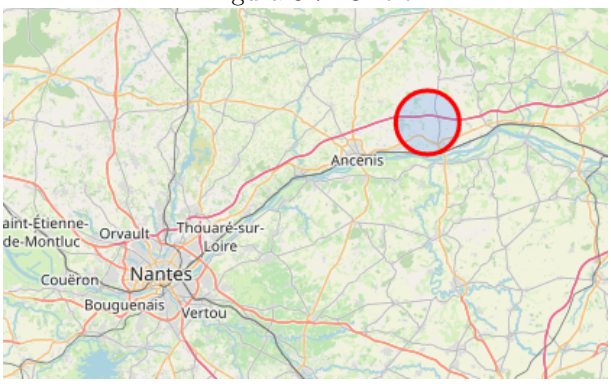
I componenti che si possono utilizzare e manipolare sono molteplici, i più importanti ed utilizzati nell'applicazione sono: *LMarker*, *LCircle* e *LPopup*. Si possono vederne gli esempi rispetivamente in Figura 3.6, Figura 3.7 ed in Figura 3.8.

Figura 3.6: Marker



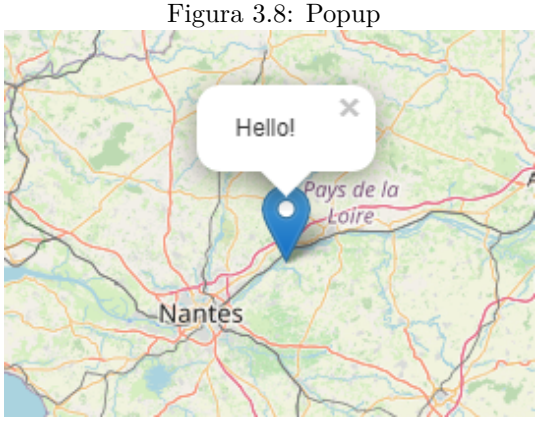
I markers sono icone associabili a determinate coordinate geografiche. Sono personalizzabili sia nella forma che nei colori e sono utili per identificare luoghi od entità importanti sulla mappa. Nel caso dell'applicazione qui descritta, i markers sono utilizzati per identificare a schermo, in tempo reale, le entità in movimento nelle zone di interesse.

Figura 3.7: Circle



I cerchi sono invece strutture geometriche di cui si devono specificare il centro e il raggio, e sono indicate per individuare zone più ampie piuttosto che un punto specifico. Nell'applicazione qui descritta, essi sono utilizzati per evidenziare le aree inerenti ai triggers geo-localizzati (geo-fences), che si attiveranno nel momento in cui dei markers delle entità vi entreranno.

Figura 3.8: Popup



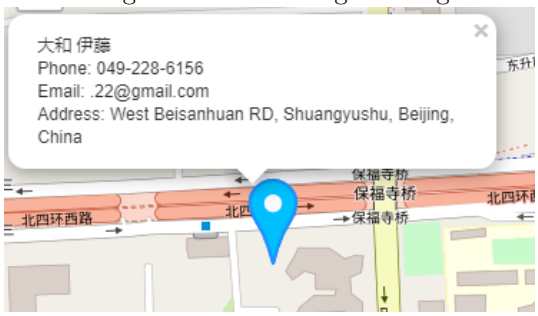
I popup sono finestre di dialogo che compaiono se si seleziona un elemento come i markers. Esse permettono di visualizzare una qualsiasi linea di testo, che può essere, come nel caso di questa applicazione, una descrizione dell'entità (Nel caso di un paziente: nome, cognome, e-mail, numero di telefono). Una funzionalità interessante di questa applicazione è quella di poter mostrare nei popup di ciascun marker l'indirizzo fisico e reale nel quale l'entità sta transitando in quel momento. Questo è possibile grazie ad un servizio di *reverse geocoding* che verrà descritto nella prossima sezione.

3.3 Reverse geocoding - HERE Developer

Il servizio scelto per implementare il reverse-geocoding è *HERE Developer* [5]. HERE è un'impresa in comproprietà delle aziende automobilistiche tedesche Audi, BMW e Daimler e fornisce servizi e tecnologie di dati di tipo geografico e di mappatura per il settore automobilistico,

consumer e aziendale.
Vi sono numerosi enti che offrono questo tipo di servizio, ma la scelta è ricaduta su questo in quanto completamente gratuito e privo di limitazioni d'uso, offrendo una gamma di REST API in grado si soddisfare le esigenze di questo progetto.
La sua funzione, come si può osservare dalla Figura 3.9, all'interno del sistema, è quella di ricevere richieste di reverse-geocoding (traduzione di coordinate geografiche in indirizzi fisici reali) e di inviare i risultati al sistema, che si occuperà di tenere aggiornate in tempo reale le informazioni dei markers delle entità con gli indirizzi fisici nei pressi dei quali esse si trovano.

Figura 3.9: Reverse geocoding



Un esempio di implementazione di reverse geocoding è possibile osservarlo nel Capitolo 5 di questo documento.

3.4 Geo-fences - Tile38

Il servizio scelto per implementare i triggers geo-localizzati è *Tile38* [6]. Si tratta di un servizio open-source che offre un database geo-spaziale e un server geo-fancing per implementare funzionalità in tempo reale.
Nel dettaglio si può dire che Tile38 dispone di un engine di indicizzazione spaziale ad alte prestazioni, e supporta un'ampia varietà di tipi di oggetti:

- *punto di latitudine/longitudine*, l'oggetto base, con campo opzionale che può essere usato per indicare dati ausiliari come altitudine o timestamp;
- *riquadri di delimitazione*, costituiti da due punti. Il primo è il punto più a sud-ovest e il secondo è il punto più a nord-est;
- *riquadri xyz*, area rettangolare rappresentata da coordinate X e Y, e livello di zoom Z;
- *geohash*, rappresentazione di un punto come stringa. Con la lunghezza della stringa come indicazione della precisione del punto;

- *geojson*, un formato standard industriale per rappresentare una varietà di tipi di oggetto come un punto, un multipunto, un poligono, un multipoligono, una collezione geometrica, proprietà, e collezioni di proprietà.

Tile38 fornisce, inoltre, notifiche di eventi geospaziali per applicazioni *mission-critical* (sistemi da cui dipende il business d'impresa), accoppiandosi con webhook esterni e code di eventi. Fornisce, infine, un supporto integrato per gli strumenti più popolari, come:

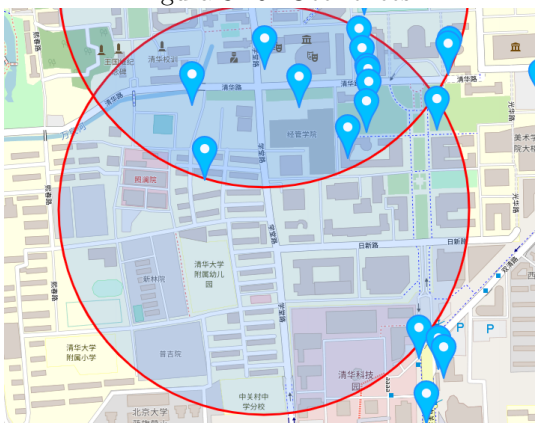
- Apache Kafka;
- Amazon SQS;
- Redis;
- RabbitMQ;

Tile38 supporta la ricerca di oggetti e punti che si trovano all'interno e/o intersecano altri oggetti. È possibile cercare tutti i tipi di oggetto, inclusi Poligoni, MultiPoligoni e Collezioni Geometriche.

- *Within*: raggruppa in una collezione gli oggetti che sono completamente contenuti all'interno di un'area di delimitazione specificata.
- *Intersects*: raggruppa in una collezione gli oggetti che intersecano un'area di delimitazione specificata.
- *Nearby*: raggruppa in una collezione gli oggetti che intersecano un raggio specifico.

La sua funzione, come si può osservare dalla Figura 3.10, all'interno del sistema, è quella di tenere traccia, tramite gli aggiornamenti che l'applicazione gli dispone, della posizione in tempo reale di tutte le entità presenti sulla mappa e di monitorare se queste entità entrano od escono dalle aree triggers scelte in fase di editing del file di configurazione. Quando questo accade, un evento viene scaturito dal server Tile38 e raccolto dal sistema, che aggiornerà l'interfaccia utente mostrando a schermo i dettagli delle entità che hanno attivato i triggers geo-localizzati. Il tutto in tempo reale e in maniera completamente asincrona grazie all'utilizzo delle callbacks.

Figura 3.10: Geo-fences



Capitolo 4

Web-app

Sommario

4.1 Requisiti e Analisi

L'obiettivo primario di questo progetto è quello di realizzare uno strumento grazie al quale un utente medio possa riuscire, nel modo più semplice e dinamico possibile, a monitorare ed analizzare il comportamento e/o le caratteristiche di entità geo-localizzate su una mappa.

4.1.1 Requisiti Funzionali

Di seguito si riportano i requisiti funzionali del progetto:

- Il punto cardine dell'applicazione, allo stato attuale, deve essere un file di configurazione grazie al quale è possibile indicare: la tipologia di *entità* che si desidera monitorare, gli *indici* e i *criteri* con i quali monitorare e/o evidenziare le entità, le *caratteristiche* intrinseche delle entità che si vogliono visualizzare nell'interfaccia utente (e.g. per un veicolo possono essere: modello, tipo, casa manifatturiera, ergonomia, gasolio, ecc.), e gli eventuali *triggers geo-localizzati* con le loro caratteristiche (e.g. coordinate di posizionamento e raggio d'azione).
- L'applicazione deve presentare un'interfaccia utente semplice e intuitiva, contenente una mappa per la visualizzazione in tempo reale delle entità tramite markers, un tabella contenente le loro informazioni e degli indici di evidenziazione selezionabili.
- L'applicazione deve essere un'applicazione web modulare realizzata con le più recenti tecnologie, in modo da garantire, partendo dal file di configurazione, una perfetta dinamicità per la costruzione dell'interfaccia utente.
- Non avendo a disposizione dati reali, è necessario simularli.
- L'applicazione deve poter garantire un servizio di *geo-fencing*, quindi poter creare dei triggers geo-localizzati per analizzare il comportamento e il movimento delle entità.
- L'applicazione deve poter garantire un servizio di *reverse-geocoding*, quindi poter tradurre in tempo reale le coordinate geografiche delle entità in indirizzi fisici reali.
- L'applicazione deve poter garantire l'utilizzo di un *linguaggio espressivo* per formulare queries atte alla personalizzazione dei parametri di visualizzazione delle entità.

La traduzione del suddetto file in un'interfaccia utente di configurazione più intuitiva è sicuramente nella visione globale dell'intero progetto, ma lasciata a sviluppi futuri. Il focus di questo progetto è infatti la dinamicità e la libertà d'uso che l'utente deve percepire utilizzando l'applicazione.

L'interfaccia utente sarà costruita dinamicamente a run-time, sulla base delle informazioni

date dall'utente in fase di configurazione, grazie alle proprietà di data-binding e di rendering dinamico del framework Vue.

I servizi di geo-fencing, reverse geo-coding e imlementazione di un linguaggio espressivo per queries personalizzate, sono funzionalità molto onerose dal punto di vista implementativo, e quindi verranno inglobate nel progetto grazie allo sfruttamento di servizi esterni.

4.1.2 Requisiti Non Funzionali

Alcuni requisiti non funzionali sono i seguenti:

- La comunicazione via rete tramite i componenti del sistema deve essere tale da non generare tempi di latenza eccessivi.
- Come tecnologia si vuole utilizzare uno stack MEVN (MongoDB, Express, Vue.js, Node.js), ma a causa dell'assenza di dati reali, si è evitato di instaurare un database fisico.
- Le entità e le loro caratteristiche sono simulate di volta in volta a run-time, in quanto non è necessario l'utilizzo di dati persistenti.

L'obiettivo della bassa latenza nell'esperienza d'uso dell'utente è dato dal fatto di voler garantire all'utilizzatore dell'applicazione un'esperienza più rapida e priva di tempi d'attesa inutili. Il giusto utilizzo dei protocolli di comunicazione odierni può fare una notevole differenza a questo proposito.

Dato che questa sarà una fase iniziale dell'intera visione globale del progetto, non si vuole dare troppo peso alla questione di manutenzione e reperibilità di una banca dati reale e consistente da utilizzare. Come già affermato, lo scopo di questo progetto è altro, quindi l'implementazione di un base di dati contenente veri dati consistenti è lasciata a sviluppi futuri.

Il gap d'astrazione tra le tecnologie scelte e i problemi da risolvere non è eccessivo, in quanto sono state scelte appositamente le più recenti in ambito di programmazione di applicazioni web modulari. Un esempio lampante è la proprietà di *data-binding* di Vue.js che permette un eccellente dinamicità e modularità nella costruzione dell'interfaccia utente e della generale programmazione lato client.
Il gap più elevato riscontrato è sicuramente quello relativo alle tecnologie di simulazione di dati realistici relativi a persone e veicoli, in quanto le librerie open source accessibili non sono evolute a tal punto da coprire l'intera gamma di caratteristiche descrittive che si vogliono modellare nelle entità.

4.2 Progettazione

4.2.1 Struttura dell'applicazione

L'architettura del sistema, come si nota in Figura 4.1, è composta da:

- *Frontend* realizzato con *Vue* come framework lato client e *Node.js*. Questo componente dell'architettura gestisce tutta l'interazione dell'utente con l'interfaccia (mappa, markers, tabella dei dati, ecc.), occupandosi di costruirla in base alle scelte dell'utilizzatore editate sul file di configurazione. Si occupa inoltre di comunicare, tramite chiamate HTTP REST, con il *backend* al fine di ottenere i dati richiesti dall'utente. L'ultima sua funzione è quella di instaurare un canale di comunicazione con il HERE-Developer Server per ottenere un servizio di *reverse-geocoding* gratuito. La scelta di interfacciarsi a questo servizio lato frontend è dovuta ad una motivazione legata al traffico dati, in quanto il backend si ritroverebbe con ulteriori comunicazioni da gestire, aumentando l'indice di latenza. Si tratta, inoltre, di una soluzione affine alla politica della richiesta a polling del frontend verso il backend per i cambiamenti delle posizioni delle entità. Nel caso in cui si presenti un cambiamento nella posizione di un marker, il nuovo eventuale indirizzo sarebbbe subito evidenziato.

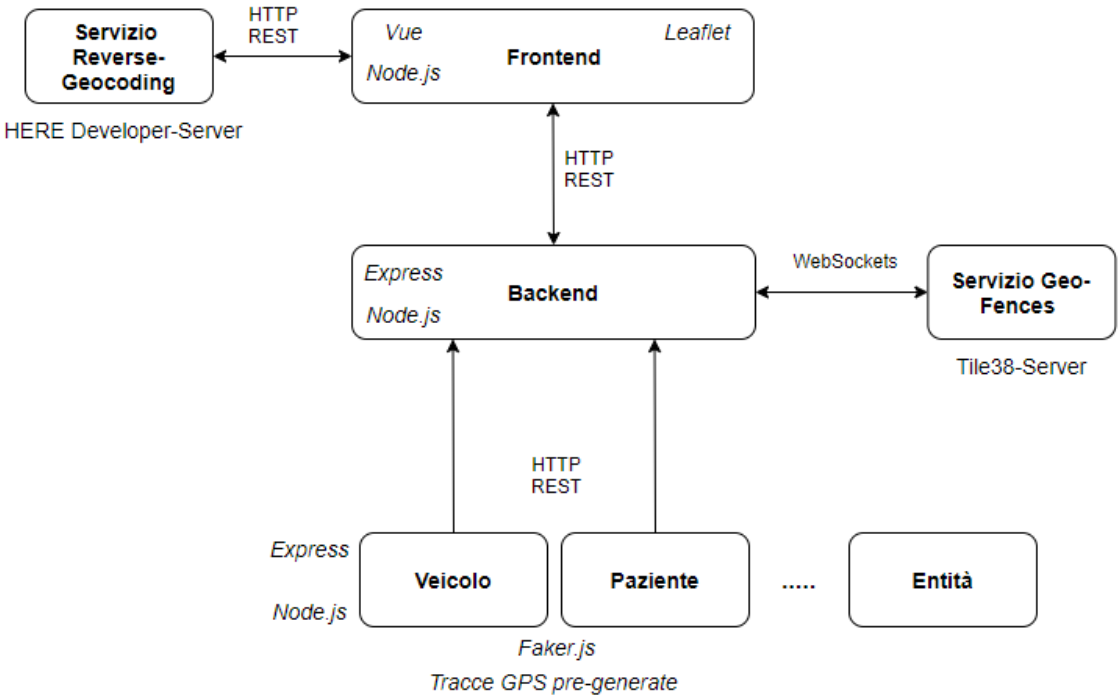
- *Backend* realizzato con *Express* e *Node.js*. Questo componente è il punto cardine della comunicazione tra tutti gli elementi del sistema. Esso infatti riceve i dati simulati in tempo reale dalle entità e le richieste da parte del frontend. Si occupa inoltre di gestire la maggior parte del carico di calcolo necessario per il corretto funzionamento delle funzionalità del sistema, come il processamento del linguaggio espressivo e il calcolo delle condizioni di selezione/monitoraggio espresse dinamicamente dall'utente.
  Il motivo per il quale è stato scelto questo componente come gestore della maggior parte dell'attività di calcolo è stato il fatto di volere mantenere un client il più leggero possibile, oltre alla presa in considerazione delle possibilità di scalabilità di un componente software come il backend, soprattutto, anche se questo non è il caso, se si effettua la scelta di operare in termini di cloud computing.

- *Entità* realizzate con *Express* e *Node.js*. Questi componenti sono elementi indipendenti, il cui unico scopo è quello di simulare le attività/spostamenti delle entità con l'utilizzo di librerie come *Faker.js* e tracce gps pre-generate, e di inviare i dati al componente *backend*.

- *Servizio Geo-Fences* è un servizio offerto da Tile38, un progetto costruito appositamente per garantire funzionalità legate a triggers geo-localizzati. Nel caso di questo progetto è stato sufficiente scaricare dal loro repository un programma eseguibile che si occupa di instaurare un server locale che comunica tramite Web Sockets.

- *Servizio Reverse-Geocoding* è un servizio offerto da *HERE Developer*, completamente gratuito. Il reverse-geocoding consiste nel tradurre coordinate geografiche in indirizzi fisici reali. La comunicazione tra il frontend e questo servizio avviene tramite chiamate HTTP REST.

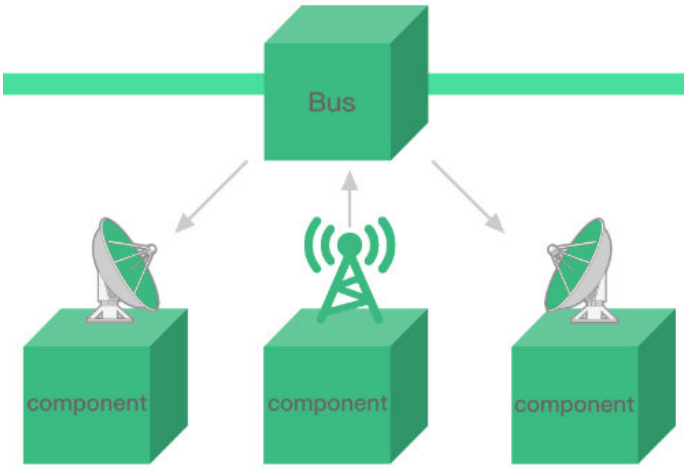
Figura 4.1: Architettura del sistema



4.2.2 Frontend

Come si può evincere osservando la Figura 4.3, il frontend è costituito da diversi componenti *.vue*, sfruttando la proprietà di *composizione di componenti* di Vue, fattore che permette di costruire l'interfaccia utente e tutte le sue funzionalità in maniera totalmente modulare, grazie alle proprietà gerarchiche dei componenti e alla proprietà di *data-binding* del framework stesso. Quest'ultimo consente la comunicazione interna tra i componenti grazie ad un *event bus*, che propaga eventi scatenati dai componenti a tutti gli altri. Come si nota osservando la Figura 4.2, ogni evento generato può essere raccolto e gestito da tutti gli altri componenti, indipendemente dal fatto che qualcuno di essi lo abbia fatto prima. Vi è quindi un'ottima gestione del bus degli eventi a basso livello, permettendo allo sviluppatore di non doversene preoccupare, salvo rare eccezioni nelle quali si vuole direttamente agire sulle politiche di gestione degli eventi.

Figura 4.2: Event Bus Globale



I dati prelevati dalla compilazione del file di configurazione da parte dell'utente vengono utilizzati da vari componenti:

- *App.vue* li utilizza per instanziare l'ambiente globale dell'applicazione, verificando la tipologia di entità che l'utente ha selezionato.
- *Page.vue* li utilizza per instanziare dinamicamente le quantità e le tipologie di componenti figli per costruire l'interfaccia utente.
- *Map.vue* li utilizza per instanziare dinamicamente le quantità e le tipologie di componenti figli per costruire la mappa dell'interfaccia utente e per determinare la logica dei dati che invia/riceve al/dal server.
- *Table.vue* li utilizza per la costruzione della tabella dei dati da mostrare nell'interfaccia utente.

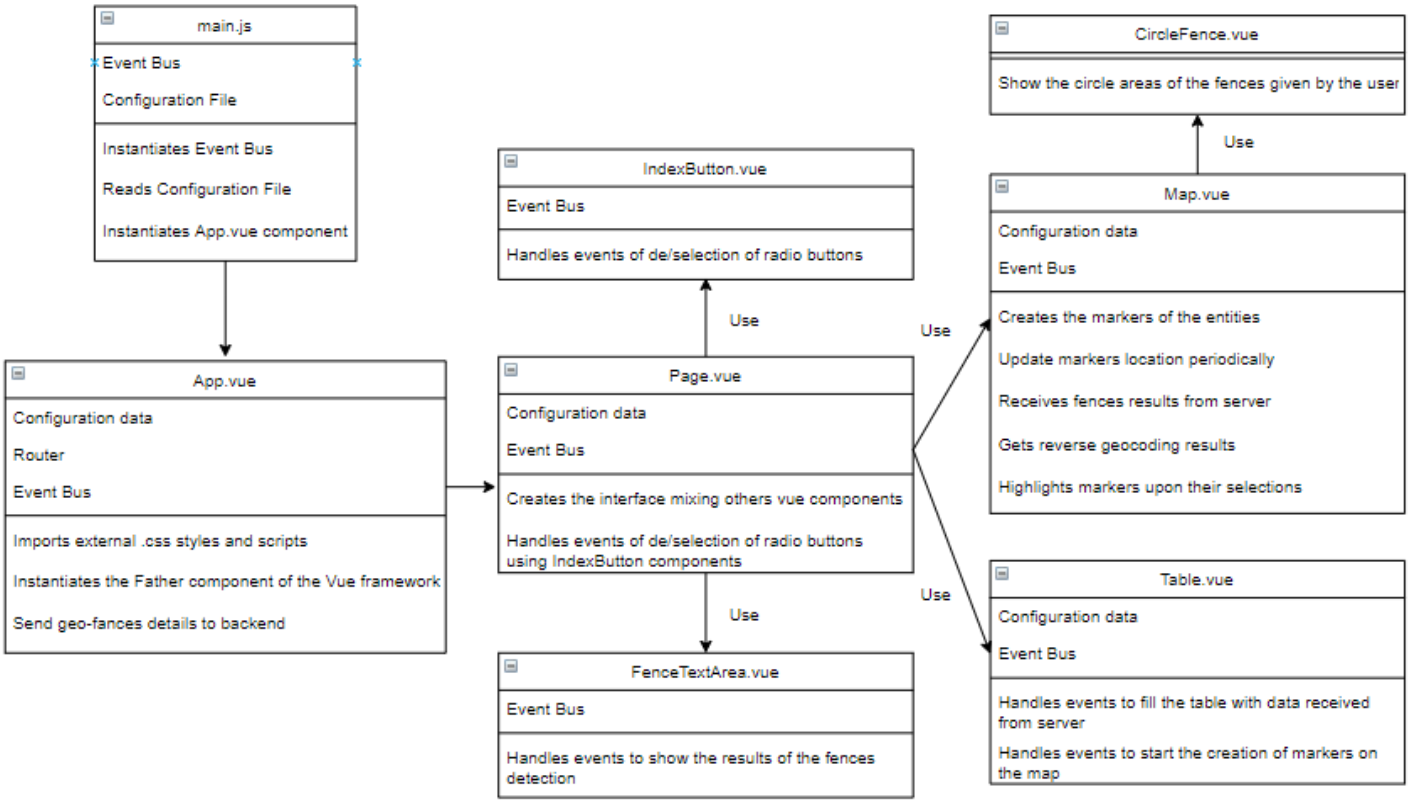
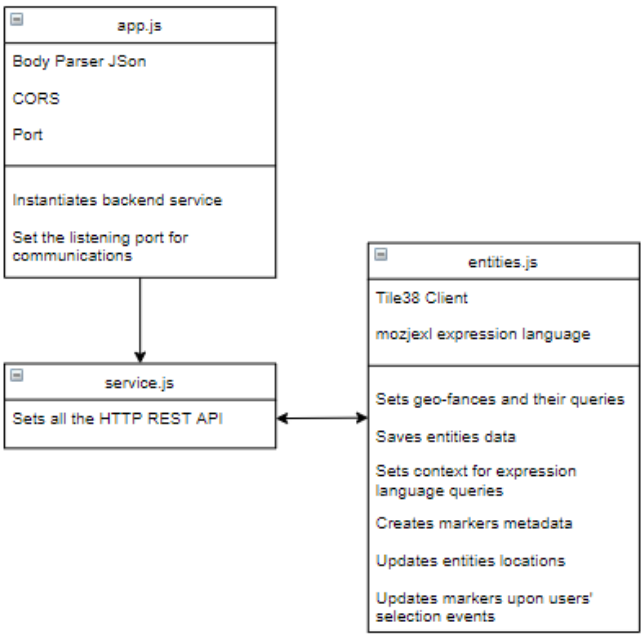


Figura 4.3: Frontend modellato

4.2.3 Backend

Come si può notare osservando la Figura 4.4, il backend è costituito da un file *app.js* che si occupa di configurare la porta per la comunicazione HTTP REST e di istanziare il file *service.js*. In questo file sono presenti e configurate tutte le API che gli altri componenti dell'architettura chiamano per comunicare con il backend e sfruttarne le funzionalità implementate.
La scelta di far ricadere su questo componente del sistema la maggior parte delle funzionalità di calcolo è dovuta al fatto, oltre al volere mantenere un frontend il più leggero possibile, che è possibile effettuare dei potenziamenti sia in *scale up* sia in *scale out*.

Figura 4.4: Backend modellato



Ma la possibilità più vantaggiosa da prendere in considerazione, per quanto riguarda la scalabilità del sistema, è quella di implementare un supporto in termini di *cloud computing*. Questo permetterebbe un potenziale di scalabilità virtualmente infinito, oltre che ad alleggerire la gestione delle risorse di sistema. Esistono diversi tipi di servizi di cloud computing già avviati da sfruttare, come *Google Cloud Platform* [1], *Amazon AWS* [2] o *Microsoft Azure* [3], che permetterebbero così:

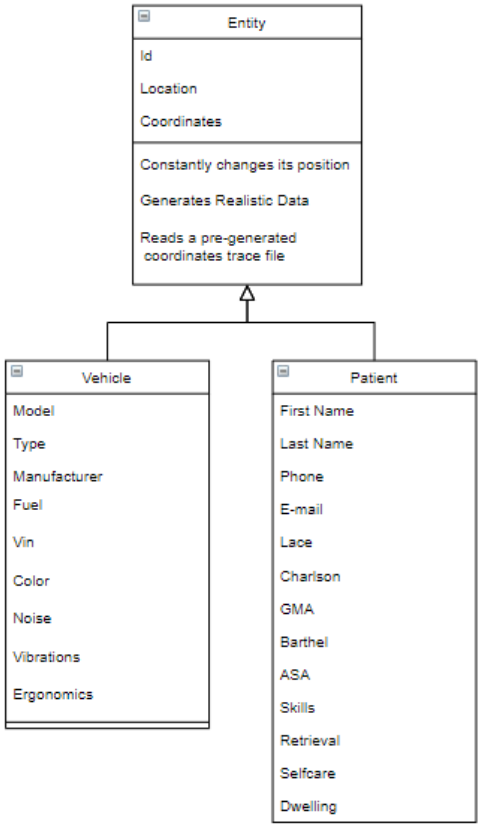
- Uno sviluppo più rapido, evitando vincoli o problemi relativi alla gestione sistemica.
- Di avere un protocollo di sicurezza dei dati già implementato.
- Di avere un possibile e più rapido accesso ad ulteriori funzionalità già presenti nel loro sistema.

Questo è sicuramente un punto da prendere in considerazione per quanto riguarda gli sviluppi futuri del progetto.

4.2.4 Entità

L'entità è l'elemento centrale della struttura del progetto, in quanto tutto è costruito per poter visualizzare e monitorare le sue caratteristiche e comportamenti a schermo. Come si può vedere in Figura 4.5, ogni entità dispone di un identificativo (Id), di un punto di partenza (Location), e di una serie di coordinate pre-generate e realistiche (Coordinates).

Figura 4.5: Entità modellate



Le entità specializzate sono i Veicoli e i Pazienti.
I dati di queste entità sono interamente generati automaticamente e pseudo-randomicamente a run-time, utilizzando la libreria *Faker.js*, come già evidenziato nel Capitolo 2.

4.3 Interazione

Le possibili scelte prese in considerazione per la comunicazione sono le *API REST* e le *WebSockets*. HTTP e WebSocket sono entrambi protocolli di comunicazione utilizzati nella comunicazione client-server.

Protocollo HTTP

HTTP è un protocollo unidirezionale in cui il client invia la richiesta e il server invia la risposta. Come si può notare dalla Figura 4.6, il suo funzionamento è il seguente [4]:

1. Il client invia una richiesta al server sotto forma di HTTP o HTTPS.
2. Dopo aver ricevuto la richiesta, il server invia la risposta al client (ogni richiesta, quindi è associata ad una risposta corrispondente)

3. Dopo aver inviato la risposta, la connessione viene chiusa (ogni richiesta HTTP o HTTPS stabilisce una nuova connessione al server, che viene terminata dopo la risposta).

Figura 4.6: Protocollo HTTP

HTTP è un protocollo stateless eseguito sulla parte superiore del TCP che è un protocollo orientato alla connessione che garantisce il trasferimento dei pacchetti di dati utilizzando i metodi di handshaking a tre vie e ritrasmette i pacchetti persi.
HTTP può essere eseguito su qualsiasi protocollo affidabile orientato alla connessione come TCP o SCTP. Quando un client invia una richiesta HTTP al server, viene aperta una connessione TCP tra il client e il server e dopo aver ottenuto la risposta la connessione TCP viene terminata. Ogni richiesta HTTP apre una connessione TCP separata al server, quindi se, ad esempio, il client invia dieci richieste al server, verranno aperte dieci connessioni HTTP separate che verranno chiuse dopo aver ottenuto la risposta/fallback.
Le informazioni del messaggio HTTP sono codificate in ASCII e ogni messaggio di richiesta è composto da metodi HTTP (GET/POST/PUT/FETCH), headers (tipo di contenuto, lunghezza del contenuto), informazioni sull'host e dal corpo del messaggio. Gli headers HTTP variano da 200 byte a 2KB, ma la dimensione generica risulta solitamente intorno ai 700-800 byte. Il payload degli headers può essere notevolmente ridotto dall'eventuale utilizzo, da parte del client, di strumenti come i cookies.

Protocollo WebSocket

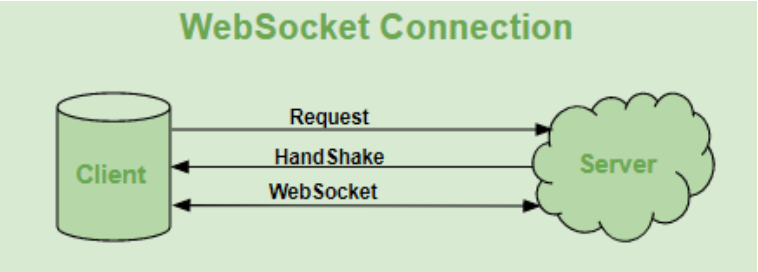
Questo protocollo è bidirezionale, utilizzato nello stesso scenario di comunicazione client-server. A differenza di HTTP è un protocollo statefull, il che significa che la connessione tra client e server rimane attiva finchè una delle due parti non la termina, determinando la chiusura anche dalla parte opposta. Come si può notare dalla Figura 4.7, il suo funzionamento è il seguente [4]:

1. Quando il client decide di iniziare una nuova connessione con il server, avviene un processo di *handshacking*, che terrà aperta la connessione finchè non verrà chiusa da una delle due parti.

2. Quando la connessione è stabilita, la comunicazione avviene tramite lo stesso canale finchè non viene terminata.

3. Lo scambio di messaggi in questo canale è bidirezionale e persistente (WebSocket).

Figura 4.7: Protocollo WebSocket

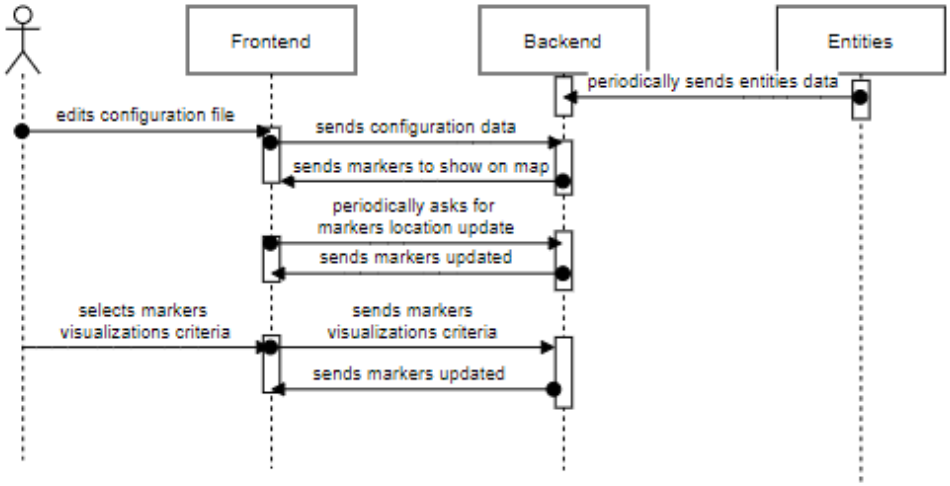


Scelta del protocollo e interazione

La tecnologia di comunicazione principale scelta è HTTP REST ma vi è anche l'utilizzo del protocollo WebSocket per la comunicazione con il servizio di geo-fences.
Come si può osservare dalla Figura 4.8 l'*utente* compila il file di configurazione, i cui dati vengono elaborati dal *frontend* per la costruzione dell'interfaccia utente e inviati al *backend*, il quale risponderà con i markers da mostrare a schermo, completi con i metadati delle entità scelte dall'utente.
La posizione geografica dei markers viene aggiornata periodicamente dal *backend* a fronte degli aggiornamenti che le *entità* simulate gli inviano, mentre lato *frontend* avviene un polling per richiedere questi dati in modo da aggiornare la posizione dei markers sulla mappa. Questa scelta è stata fatta a fronte del basso carico dati che ogni richiesta esige, in quanto si tratta di semplici coordinate geografiche. La scelta alternativa sarebbe stata quella di utilizzare le web sockets, ma a meno di far si che il backend istanziasse un server figlio per la gestione della comunicazione, sarebbero state bloccanti. Considerato il fatto che il backend è il nucleo centrale di calcolo del sistema e data la bassa mole di dati coinvolta, inserire una meccanica bloccante non è sembrata la scelta più efficiente. Inoltre, la comunicazione tra Entità e Backend è unidirezionale.

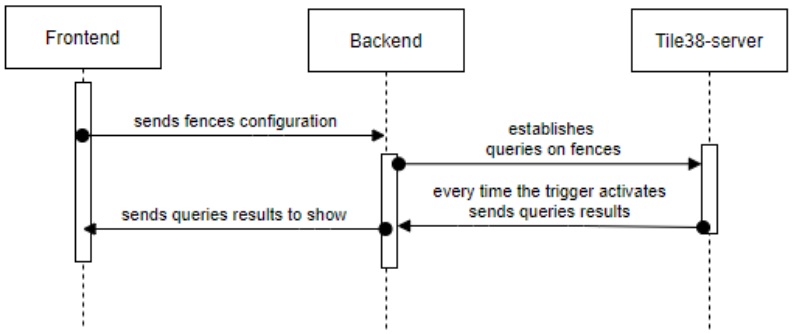
Test sull'efficienza della struttura di comunicazione sono stati effettuati e i suoi risultati verranno mostrati nella prossima sottosezione.

Figura 4.8: nterazione Utente-Frontend-Backend-Entità



Come si osserva dalla Figura 4.9, l'interazione con il Server Tile38 avviene a fronte dell'invio, da parte del frontend, dei dati di configurazione relativi alle geo-fences scelte dall'utente al backend, il quale si occupa della costruzione delle queries che vengono poi periodicamente eseguite sul server del servizio. Tile38 notifica così il backend con i risultati delle queries, che vengono poi inviati al frontend per mostrarli a video. Il protocollo di comunicazione in questo caso è quello delle WebSockets, in quanto l'aggiornamento delle attivazioni dei triggers geo-localizzati deve essere estremamente veloce e performante, e il carico di dati nel messaggio di notifica è di discrete dimensioni.

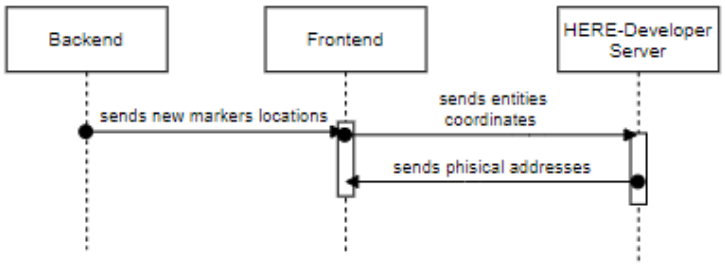
Figura 4.9: Interazione Frontend-Backend-Tile38Server



Come si osserva dalla Figura 4.10, l'interazione con il *Server HERE-Developer* per il reverse-geocoding avviene periodicamente a partire dal *frontend*, il quale, a seguito dell'aggiornamento delle coordinate dei markers, invia una richiesta al servizio per tradurre le nuove coordinate in indirizzi fisici reali. Il protocollo di comunicazione qui è HTTP in quanto non è richiesto un tempo di aggiornamento tale da rendere vitale la politica del real-time, inoltre il carico di dati

del messaggio è di dimensioni ridotte, e la comunicazione unidirezionale.

Figura 4.10: Interazione Frontend-HEREServer



4.3.1 Test di Efficienza

I test effettuati sono stati incrementalmente lineari, partendo da un carico di venti entità coinvolte e finendo con un centinaio di elementi a schermo attivi contemporaneamente.
Come si può notare in Figura 4.11, relativa alla comunicazione tra frontend e backend con il protocollo HTTP, al crescere lineare delle entità si è verificato un aumento lineare dei tempi di latenza della comunicazione, che è quello che ci si aspettava. Inoltre, i tempi risultanti sono considerati più che accettabili dal punto di vista dell'efficienza.

Figura 4.11: Test sull'efficienza HTTP

20 Entità	10-12 millisecondi	
40 Entità	20-24 millisecondi	
60 Entità	44-64 millisecondi	
80 Entità	62-80 millisecondi	
100 Entità	81-96 millisecondi	

Come si può notare, invece, in Figura 4.12, relativa alla comunicazione tra frontend e backend con il protocollo WebSocket, si presenta lo stesso incremento lineare, ma leggermente superiore in termini di millisecondi necessari, rispetto al protocollo HTTP, a causa processo di *handshacking* e instaurazione di un server figlio per la gestione asincrona di ogni comunicazione aperta. Leggermente svantaggioso, rispetto alla quantità di dati trasportati nel corpo di ogni messaggio.

Figura 4.12: Test sull'efficienza WebSocket

20 Entità	11-12 millisecondi	
40 Entità	22-25 millisecondi	
60 Entità	49-67 millisecondi	
80 Entità	71-89 millisecondi	
100 Entità	91-104 millisecondi	

4.4 Analisi sulla Distribuzione

4.4.1 Multi-utenza

Per quanto riguarda la possibilità di avere più utenti connessi, al momento è possibile avere più utenti connessi sulla stessa macchina, ma con una limitazione. Il backend risponde alle chiamate multiple (nel caso avessi più frontend aperti) alla stessa macchina e quindi, gli eventuali risultati chiesti da un client vengono visualizzati anche sull'altro.
Questo risultato è dovuto dalla mancanza di un protocollo di identificazione (di utente e/o di macchina) che rende unica una sessione d'uso.
A questo proposito, all'interno del sistema, allo stato attuale, viene utilizzata una libreria JavaScript denominata *Axios.js*, che permette di realizzare le chiamate HTTP REST dal frontend verso il backend. Tale libreria, oltre a permettere di costruire le chiamate in modo intuitivo e semplice, conferisce anche la possibilità di indicare, nei campi opzionali della chiamata, un codice identificativo come un *token di autenticazione utente* (come previsto, d'altronde, dallo stesso protocollo HTTP REST).
Utilizzare token di autenticazione in questo modo permette di costruire un'interfaccia di rete personalizzata per ciascun utente, garantendo l'utilizzo esclusivo di ogni istanza dell'applicazione. Questo risultato è possibile, tuttavia, a fronte di un'eventuale implementazione di un servizio di login/autenticazione dell'utenza, che per ora rimane previsto per gli sviluppi futuri, considerando il fatto che a tal proposito sarebbe necessaria l'installazione di un database in cui memorizzare i dati degli utenti.

4.4.2 Multi-istanza

Un'osservazione può essere fatta anche dal punto di vista della possibilità di avere più istanze del backend su macchine diverse. Allo stato attuale è possibile e come già affermato precedente-

mente l'unico vincolo risiede nell'indirizzo fisico da indicare al frontend, in quanto al momento non è presente un protocollo di scelta dinamica di un'istanza del backend più appropriata, per esempio dal punto di vista geografico per la minimizzazione della latenza.
Si può inoltre precisare che sarebbe possibile disporre di istanze del backend relative ad aree geografiche differenti, in quanto il componente, allo stato attuale non ha alcun vincolo geografico, dovuto al fatto dell'assenza di un controllo dinamico sulla localizzazione geografica delle entità. Tuttavia, non disponendo di un database fisico associato al backend, e non essendoci un'associazione con l'area geografica scelta dinamicamente dall'utente nel frontend, non risulta prettamente fattibile allo stato attuale.

Una nota da precisare in maniera critica è la seguente: *il servizio Tile38 è utilizzato tramite l'avvio di un applicativo locale che instaura un server in* ***localhost****, la funzionalità che permette di costruire le geo-fences non sarebbe disponibile nel caso non fosse installato sulla stessa macchina dove risiede il backend.*
Negli sviluppi futuri di questo progetto vi è sicuramente la ricerca di un servizio differente che possa garantire la funzionalità dei triggers geolocalizzati non in localhost, in maniera tale che gli utenti dell'applicazione abbiano anche la possibilità di usufruire delle sue funzionalità tramite dispositivi mobili.

4.4.3 Utilizzo su dispositivi mobili

Per quanto riguarda la possibilità di utilizzo dell'applicazione tramite dispositivi mobili, si può affermare che è possibile usufruire delle funzionalità del sistema anche da dispositivi mobili quali smartphones o tablets.
Per testare questa possibilità è stato installato il *backend* sulla piattaforma Heroku, cambiato l'URL delle chiamate nel frontend e nelle entità simulate, e avviato l'applicazione da un smartphone.
Il risultato, dal punto di vista estetico e di intuitività e semplicità d'uso dell'interfaccia utente, è accettabile. I test iniziali sono stati effettuati a partire da un numero esiguo di entità (pari a dieci) e ne sono stati effettuati aumentando gradualmente il loro numero. Testando l'applicazione con un numero di entità sempre crescente, fino a cento, si notano i limiti della mancanza di scalabilità orizzontale del server sul quale è installata l'applicazione, che evidenziano una latenza eccessiva negli aggiornamenti delle posizioni delle entità sulla mappa. I dettagli su questo punto sono mostrati ed esplicati nella sottosezione successiva.

4.4.4 Distribuzione su piattaforme cloud

L'applicazione, in un'ottica di testing, è stata installata sulla piattaforma cloud Heroku [5]. Il risultato, dopo un testing effettuato da dispositivo fisso, così da includere le funzionalità di geo-

fencing con l'applicativo locale di Tile38, è stato insoddisfacente in termini di latenza del traffico dati. Il test è stato effettuato con *cento* entità simulate, cominciando a risultare più in linea con gli standard di accettazione per le performance intorno alla decina di entità. Questo a causa dell'impossibilità di scalare l'applicazione orizzontalmente all'interno del servizio, fatto dovuto ai limiti della licenza gratuita, non adatti per applicazioni *high-traffic real-time*. Nel caso in cui si volesse, negli sviluppi futuri, esplorare più in profondità l'opzione di un'implementazione incentrata sul cloud-computing sarebbe necessaria una valutazione costi-benefici sull'eventuale progetto, in quanto le terze parti offrono il servizio necessario per questa applicazione previo pagamento periodico. Oltre ad avere funzionalità di deployment automatico e monitoring dell'efficienza del sistema del sistema, si otterrebbero anche delle eccellenti funzionalità riguardo alla sicurezza dei dati (e.g. crittografia, monitoraggio dei pacchetti dati).

La piattaforma Heroku basa il suo funzionamento su specifici containers chiamati *Dynos*. Si tratta di containers Linux isolati e virtualizzati progettati per eseguire codice in base ad un comando specificato dall'utente. Un'applicazione può scalare verso qualsiasi numero specificato di dynos in base alla necessità di risorse. Le funzionalità di gestione dei containers di Heroku forniscono un modo semplice per scalare e gestire il numero, le dimensioni e il tipo di dynos di cui l'applicazione potrebbe aver bisogno in un dato momento. Inoltre il router mantiene un buffer di 1MB per le risposte del dyno, per ogni connessione.

Le applicazioni in Heroku utilizzano un singolo file di configurazione, il *"Procfile"*, per specificare le tipologie di processi necessari per eseguirle. Ogni tipologia di processo rappresenta un comando che deve essere eseguito dal *Dyno Manager* quando viene eseguito un dyno. Ci sono tre principali tipi di processi:

- *Web Dynos*: ricevono il traffico HTTP dai routers ed eseguono dei web servers;
- *Worker Dynos*: eseguono i jobs di background, i sistemi di coda, e i jobs in scadenza;
- *One-off Dynos*: sono temporanei ed eseguono comandi di breve durata, possibilmente collegati a un terminale locale, e sono in genere utilizzati per gestire attività amministrative come l'esecuzione di una shell REPL per eseguire migrazioni di database o jobs occasionali in background.

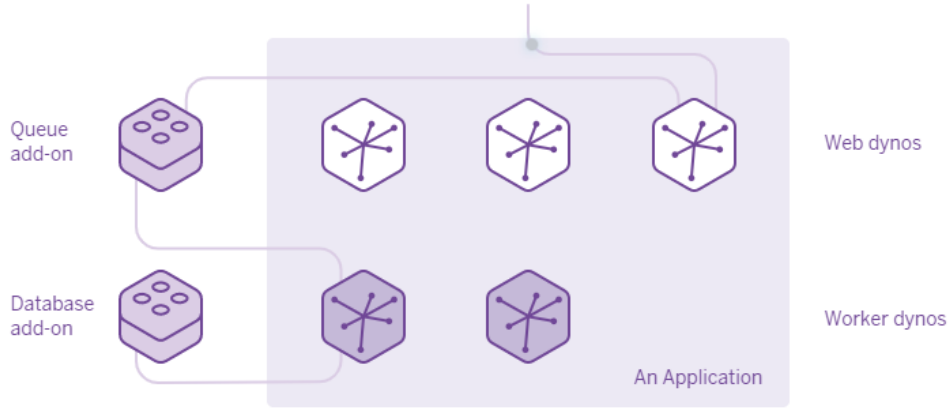
In Figura 4.13 si può notare il funzionamento specifico dei Dynos di Heroku. In particolare il flusso può essere esplicato come segue:

1. L'applcazione riceve una richiesta via web;
2. La richiesta è spedita ad un randomico web dyno;
3. La richiesta è inserita in una coda e il web dyno ritorna un messaggio di successo all'utente;

4. Un worker dyno raccoglie la richiesta dalla coda e svolge il lavoro da eseguire;

5. Il worker dyno può far persistere il risultato del lavoro svolto in un database, il quale può essere poi ritornato all'utente in una richiesta differente;

Figura 4.13: Funzionamento dei Dynos su Heroku



Più nello specifico sono riportati alcuni dettagli tecnici:

Tipologia Dyno	RAM	CPU Share	Compute
free	512 MB	1x	1x-4x
hobby	512 MB	1x	1x-4x
standard-1x	512 MB	1x	1x-4x
standard-2x	1024 MB	2x	4x-8x
performance-m	2.5 GB	100%	12x
performance-l	14 GB	100%	50x

Di seguito sono indicati nello specifico i dettagli tecnici delle varie licenze disponibili sulla piattaforma:

Free

RAM	512MB
Deploy da Git	Si
Aggiornamenti automatici del SO	Si
Logs unificati	Si
Numero di processi	2
Domini personalizzati	Si
Cripting dei dati	No
Scaling orizzontale	No
Preboot	No
Metriche applicazione	No
Metriche linguaggio run-time	No
Allarmi su triggers	No
Autoscaling	No
VPN	No
Network access controls (VPC peering, Internal routing)	No
Prezzo	0 dollari/mese

Hobby

RAM	512MB
Deploy da Git	Si
Aggiornamenti automatici del SO	Si
Logs unificati	Si
Numero di processi	10
Domini personalizzati	Si
Cripting dei dati	Si
Scaling orizzontale	No
Preboot	No
Metriche applicazione	Si
Metriche linguaggio run-time	No
Allarmi su triggers	No
Autoscaling	No
VPN	No
Network access controls (VPC peering, Internal routing)	No
Prezzo	7 dollari/mese

Standard

RAM	512MB-1GB
Deploy da Git	Si
Aggiornamenti automatici del SO	Si
Logs unificati	Si
Numero di processi	Infiniti
Domini personalizzati	Si
Cripting dei dati	Si
Scaling orizzontale	Si
Preboot	Si
Metriche applicazione	Si
Metriche linguaggio run-time	Si
Allarmi su triggers	Si
Autoscaling	No
VPN	No
Network access controls (VPC peering, Internal routing)	No
Prezzo	25-50 dollari/mese

Performance

RAM	2.5-14GB
Deploy da Git	Si
Aggiornamenti automatici del SO	Si
Logs unificati	Si
Numero di processi	Infiniti
Domini personalizzati	Si
Cripting dei dati	Si
Scaling orizzontale	Si
Preboot	Si
Metriche applicazione	Si
Metriche linguaggio run-time	Si
Allarmi su triggers	Si
Autoscaling	Si
VPN	No
Network access controls (VPC peering, Internal routing)	No
Prezzo	250-500 dollari/mese

Oltre questa fascia di prezzo si avrebbero a disposizione anche servizi come VPN, VPC peering e Internal routing.

Capitolo 5

Dettagli Implementativi

Sommario

5.1 File di Configurazione

In questa sezione si vuole mostrare lo stato attuale del file di configurazione, ovvero un semplice file *.json*, contenente, come mostrato in Figura 5.1:

- *center*: il punto geografico in cui la mappa deve essere centrata. Nel caso di questo progetto è il centro di Pechino (Cina). In sostituzione delle coordinate geografiche, può essere possibile inserire, più intuitivamente, un indirizzo fisico reale che successivamente viene tradotto in coordinate geografiche da un servizio di geocoding. Tuttavia si tratta di una features non ancora implementata e lasciata a sviluppi futuri.

- *entity*: la tipologia di entità che si scegli di visualizzare. Nel caso di questo progetto sono i Veicoli (*vehicles*) ma potrebbero essere anche Pazienti (*patients*).

- *indexes*: gli indici per i quali l'utente può cambiare i criteri di visualizzazione dei markers sulla mappa. Per ogni indice che si vuole visualizzare è necessario specificare il componente .Vue (*component*), in questo caso *IndexButton*, a causa della proprietà di data-binding e rendering dinamico del framework. Negli sviluppi futuri del progetto si vorrà "nascondere" questo tecnicismo di basso livello sotto un'interfaccia grafica di configurazione molto più intuitiva. Risulta necessario specificare quindi la tipologia di indice (*text*) e il criterio di selezione (*tertile*, *quartile*, o arbitrario utilizzando la semantica data dal linguaggio espressivo).

- *values*: i valori/caratteristiche che l'utente vuole vedere nella tabella dei dati su schermo. Risulta necessario specificare la tipologia di valore (*text*) e la sua etichetta (*value*) per fornire la possibilità di utilizzo delle proprietà di data-binding e rendering dinamico di Vue. Anche in questo caso si vorrà nascondere questo tecnicismo in sviluppi futuri.

- *fences*: il numero e la tipologia di triggers geo-localizzati che l'utente desidera, sia in termini di posizione geografica, sia in termini di raggio d'azione. Risulta necessario specificare il componente .Vue (*component*), il componente grafico del trigger (*circle*), un identificativo arbitrario (*id*), le coordinate di latitudine e longitudine in cui sarà centrato (*lat* e *lng*) e il raggio d'azione espresso in metri (*radius*).

Figura 5.1: File di configurazione

```
"center": [39.984702,116.318417],
"entity": "vehicles",
"indexes": [
  { "component": "IndexButton","text": "Noise", "criterion": "tertile" },
  { "component": "IndexButton", "text": "Vibrations", "criterion": "quartile" },
  { "component": "IndexButton", "text": "Fuel", "criterion": "vehicles[.fuel < 30]" },
  { "component": "IndexButton", "text": "Ergonomics", "criterion": "vehicles[.ergonomics < 30]" }
],
"values": [
  { "text": "Vehicle", "value": "vehicle" },
  { "text": "Manufacturer", "value": "manufacturer" },
  { "text": "Model", "value": "model" },
  { "text": "Type", "value": "type" },
  { "text": "Vin", "value": "vin" },
  { "text": "Noise (db/A)", "value": "noise" },
  { "text": "Vibrations (m/s^2)", "value": "vibrations" },
  { "text": "Fuel %", "value": "fuel" },
  { "text": "Ergonomics %", "value": "ergonomics" }
],
"fences": [
  { "component":"FenceTextArea", "circle":"CircleFence", "id": "Tsinghua Science and Technology Park",
    "lat": "39.994", "lng": "116.326", "radius": "500" },
  { "component":"FenceTextArea", "circle":"CircleFence", "id": "Southern Daoxiangyuan Community",
    "lat": "39.981", "lng": "116.300", "radius": "500" },
  { "component":"FenceTextArea", "circle":"CircleFence", "id": "Qinghua Road",
    "lat": "39.999", "lng": "116.314", "radius": "500" },
  { "component":"FenceTextArea", "circle":"CircleFence", "id": "East Zhongguancun Road",
    "lat": "39.983", "lng": "116.328", "radius": "500" }
]
```

5.2 Proprietà di data-binding e rendering dinamico

In questa sezione si mostra il funzionamento della proprietà di data-binding e rendering dinamico del framework Vue per la costruzione delle aree di testo in cui vengono mostrate le informazioni delle entità che attivano i triggers geo-localizzati.
Come viene mostrato in Figura 5.2, nella parte HTML del componente padre viene dichiarata la volontà di inserire nell'interfaccia utente un numero di componenti *block.component* (osservando la Figura 5.1 questi componenti sono le *FenceTextArea*), per ogni elemento presente nell'array *fences* del file.

Figura 5.2: Data-binding componente padre

```
<template v-for="block in myjson_fences">
  <component :is="block.component" :block="block" :key="block.id"></component>
</template>
```

Nel componente figlio, *FenceTextArea.vue* si indica nella sezione HTML il tipo di componente utilizzato. Come si nota dalla Figura 5.3 si tratta di una semplce v-card con un titolo dinamico,

scelto dall'utente in fase di configurazione (*block.id*) e i una variabile dinamica *textResults* che verrà aggiornata ogni volta che vi saranno novità dal punto di vista dei risultati delle geo-fences presi dal backend.

Figura 5.3: Data-binding componente figlio

```
<template>
  <div>
    <v-card>
      <v-card-title class="headline grey lighten-2">
        {{block.id}}
      </v-card-title>
      <v-card-text>
        {{textResults}}
      </v-card-text>
    </v-card>
  </div>
</template>
```

Come si può vedere, a tal proposito, dalla Figura 5.4, per dare vita a queste proprietà è necessario specificare: le proprietà esterne *props*, in questo caso specifico l'array *fences* del file di configurazione, comunicato a questo componente dal suo componente padre, e la variabile *textResults* per l'aggiornamento dinamico delle informazioni.

Figura 5.4: Data-binding componente figlio

```
export default {
  props: {
    block: Object,
  },
  name: "FenceTextArea",
  data: () => ({
    textResults: '',
```

Successivamente, come si nota dalle informazioni cerchiate in rosso nella Figura 5.5, a seguito dell'evento *bus.$on('fencesResults')* la variabile viene aggiornata con le nuove informazioni (*this.textResults*).

Figura 5.5: Aggiornamento dinamico della variabile

```
bus.$on( event: 'fencesResults', callback: (data) => {
  for (let i = 0; i < data.length; i++) {
    if(this.block.id == data[i].id){
      for (let j = 0; j < data[i].results.length; j++) {
        if (!(this.fenceResults.includes(data[i].results[j]))) {
          this.bench.unshift(data[i].results[j]);
          this.fenceResults.unshift(data[i].results[j]);
          let splitting = data[i].results[j].split(':');
          let id = splitting[0];
          let firstField = splitting[1];
          let secondField = splitting[2];
          this.textResults += '' + id + ', ' + firstField + ', ' + secondField + '  |  ';
```

5.3 Linguaggio espressivo per queries personalizzate

Il linguaggio espressivo utilizzato per implementare una personalizzazione dei criteri di evidenziazione e selezione delle entità dell'applicazione è stato ricavato dallo sfruttamento della libreria JavaScript *mozjexl* [1].

Il suo funzionamento è molto semplice, e consiste nel costruire un contesto (*context*) all'interno del quale inserire le informazioni che verranno valutate e scandagliate tramite le queries. Un esempio di contesto si può osservare in Figura 5.6.

Figura 5.6: Constesto mozJexl

```
var context = {
    name: {first: 'Sterling', last: 'Archer'},
    assoc: [
        {first: 'Lana', last: 'Kane'},
        {first: 'Cyril', last: 'Figgis'},
        {first: 'Pam', last: 'Poovey'}
    ],
    age: 36
};
```

Alcuni esempi di possibili queries per l'interrogazione del contesto sono riportati in Figura 5.7 per il filtraggio di un array; In Figura 5.8 per operazioni matematiche; In Figura 5.9 per la concatenazione dei criteri di selezione. Altri esempi sono ampiamente mostrati nella documentazione ufficiale [1].

Figura 5.7: Filtraggio di un array

```
// Filter an array
jexl.eval('assoc[.first == "Lana"].last', context).then(function(res) {
    console.log(res); // Output: Kane
});
```

Figura 5.8: Operazione matematica

```
// Do math
jexl.eval('age * (3 - 1)', context, function(err, res) {
    console.log(res); // Output: 72
});
```

Figura 5.9: Concatenazione di criteri di selezione

```
// Concatenate
jexl.eval('name.first + " " + name["la" + "st"]', context).then(function(res) {
    console.log(res); // Output: Sterling Archer
});
```

Nel sistema, il contesto viene creato dinamicamente dal backend una volta che le entità sono state create, come si può notare in Figura 5.10. Sono mantenuti in memoria due constesti separati, uno per i veicoli e uno per i pazienti, per alleggerire il carico di calcolo per le queries.

Figura 5.10: Creazione del contesto delle entità Veicoli

```
contextVehicles = {
    vehicles: vehiclesArray
};
```

Successivamente, come si nota in Figura 5.11, si esegue la query desiderata sul contesto desiderato. La query è data dall'utente stesso in fase di editing del file di configurazione (*criterion*), e il risultato viene ritornato tramite callback.

Figura 5.11: Creazione del contesto delle entità Veicoli

```
return await mozjexl.eval(criterion, contextVehicles).then(function (res) {
```

5.4 Reverse geocoding

In questa sezione viene mostrata l'implementazione della funzione JavaScript per l'interrogazione del servizio di *reverse geocoding* fornito dal server HERE Developer. Il servizio fornisce un codice *apiKey* da utilizzare all'interno della funzione come chiave di autenticazione, previa registrazione come utente sul server HERE [2], per potere avere accesso alla funzionalità desiderata.

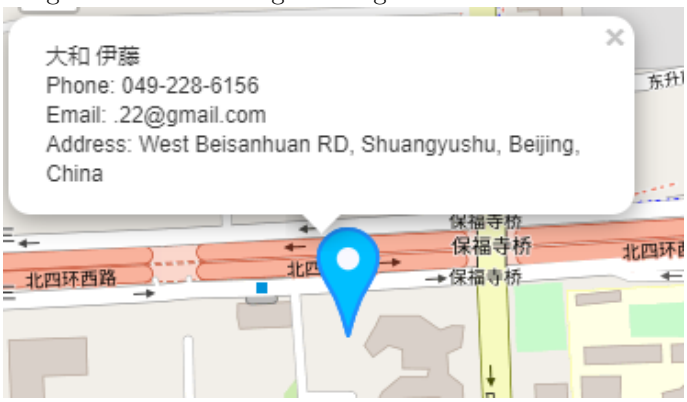
Come si può notare dalla Figura 5.12, l'*apiKey* viene impostata come parametro nel costruttore dell'istanza della piattaforma del servizio. Nel caso in cui l'autenticazione vada a buon fine, al servizio viene inviato un messaggio HTTP per ogni coppia latitudine-longitudine delle entità presenti nel sistema. Si tratta di un'operazione asincrona, che continuamente aggiorna le posizioni di tutti i markers sulla mappa.

Figura 5.12: Funzione JavaScript per l'interrogazione del servizio di reverse geocoding

```
reverseGeocoding() {
  var platform = new H.service.Platform( options: {
    apikey: [this.apiKey]
  });
  var service = platform.getSearchService();
  for (let i = 0; i < this.markers.length; i++) {
    service.reverseGeocode( params: {
      at: '' + this.markers[i].location.lat + ',' + this.markers[i].location.lng
    }, onResult: (result) => {
```

In Figura 5.13 si può osservare distintamente il popup del marker che mostra a video delle informazioni descrittive dell'entità e la posizione geografica tradotta in indirizzo reale.

Figura 5.13: Reverse geocoding di un'entità in real-time



5.5 Geo-fences

In questa sezione viene mostrata l'implementazione di una live geo-fence, il cui ruolo è quello di notificare il sistema, in questo caso il backend, ogni volta che un'entità soddisfa i requisiti per l'attivazione del trigger geo-localizzato. Inizialmente, come si può notare dalla Figura 5.14, inerente all'entità veicoli, si inserisce nel sistema di Tile38 ogni entità attiva, utilizzando il seguente format: *set(key, id, locationObject, fields, options)*. Osservando la figura si vengono indicati:

1. *entity*, si tratta della *key* alla quale associare l'entità che si sta inserendo in Tile38. Si tratta di un dato arbitrario, ma per meglio comprendere il suo ruolo si può affermare, per esempio, che se volessi inserire un veicolo, potrei denominare questa key "flotta". Utilizzando quindi keys diverse posso generare gruppi distinti di entità, in maniera totalmente arbitraria. In questo caso specifico (Figura 5.14), la key sarà denominata come l'entità scelta dall'utente in fase di configurazione;

2. *vehicle.id*, si tratta dell'identificativo della singola entità che vado ad inserire nel gruppo della key corrispondende. In questo caso sarà un *id* che viene generato pseudo-randomicamente nel momento della creazione delle entità;

3. *vehicle.vehicle* e *vehicle.manufacturer*, si tratta, in questo caso specifico, di informazioni descrittive della tipologia di veicolo e della casa manifatturiera del veicolo che sto inserendo nel sistema Tile38. Queste informazioni vengono concatenate come stringhe all'*id* in quanto, all'interno dei campi opzionali che il format prevede, non è possibile inserire dati che non siano numerici. Quindi il desiderio di inserire dati descrittivi dell'entità in forma di stringa è possibile grazie a questo artificio tecnico. Risulta necessario, se desidero ottenere tali informazioni in seguito, effettuare un parse dell'*id*. Ovviamente, in questa maniera, è possibile inserire qualsiasi tipo di informazione o dato descrittivo dell'entità che si desideri;

4. *vehicle.location.lat* e *vehicle.location.lng* fanno parte del campo *locationObject*, ovvero le coordinate attuali dell'entità. Essendo solo le coordinate attuali, questa operazione (client.set...) viene effettuata ogni volta che le coordinate di posizione di un'entità attiva cambiano. Effettuare nuovamente la stessa operazione, con la stessa *key* e lo stesso *id*, risulta in una sovrascrittura dell'entità nel sistema Tile38, prevenendo un eventuale eccesso di elementi.

Figura 5.14: Aggiunta di un'entità nel server Tile38

```
client.set(entity, vehicle.id + ':' + vehicle.vehicle + ':' + vehicle.manufacturer,
    [vehicle.location.lat, vehicle.location.lng]).catch(err =>
    console.log(err) // id not found
);
```

Successivamente, si possono costruire le *queries* che andranno a determinare le condizioni di attivazione delle geo-fences. Come si può notare dalla Figura 5.15, nella fase di costruzione di una query è necessario indicare la tipologia di condizione di attivazione [3]:

- *withinQuery*, come in questo caso specifico, la condizione di attivazione consiste nell'identificazione di un'entità all'interno del raggio della geo-fence;

- *intersectsQuery*, la condizione di attivazione consiste nell'identificazione di un'entità che interseca il confine della geo-fence;

- *searchQuery*, la condizione di attivazione consiste nell'identificazione di un'entità che rispetti alcuni vincoli, come ad esempio un *id* specifico, all'interno del raggio della geo-fence.

- *nearbyQuery*, la condizione di attivazione consiste nell'identificazione di un'entità che si trovi nei pressi di una geo-fence in una misura arbitraria.

Risulta possibile, inoltre, scegliere una sub-condizione di attivazione del trigger (*detect*) e la forma della geo-fence (*circle*), indicando in maniera arbitraria le coordinate del punto di origine (*.lat* e *.lng*) ed il raggio d'azione del trigger (*.radius*).
La condizione *detect* specifica un'ulteriore vincolo di attivazione della geo-fence. In questo caso particolare (Figura 5.15) viene indicato *inside* per fare in modo che il trigger si attivi ogni volta che l'entità *entra* nella geo-fence, ed *outside* per fare in modo che il trigger si attivi anche quando l'entità *esce* dall'area del trigger. Questa è un possibilità che Tile38 offre per espandere le opzioni di monitoraggio, fornendo così anche la possibilità di implementare un'allerta, per esempio, se l'area di una geo-fence si sta spopolando, avvertendo l'utente che una determinata strada è meno trafficata o che un luogo è meno popolato e quindi risulta meno rischioso in termini di esposizione e contagio di malattie.

```
let query = client.withinQuery(fenceKey).detect('inside', 'outside').circle(fenceToDo.lat, fenceToDo.lng, fenceToDo.radius);
```

Figura 5.15: Costruzione di una query

Infine, è necessario eseguire la live geo-fence tramite la query appena creata. Come si nota in Figura 5.16, viene eseguito il comando di esecuzione sulla query e inserita una sub-condizione di attivazione del trigger nel caso in cui le condizioni di monitoraggio vengano rispettate.
Ogni volta che le condizioni di attivazione delle geo-fences sono rispettate, il sistema viene notificato dal server Tile38 in maniera asincrona tramite WebSockets (si noti la callback *results* in Figura 5.16).

```
let fence = query.executeFence((err, results) => {
    if (err) {
        console.error('Query ' + fenceToDo.id +': something went wrong! ' + err);
    } else {
        if(results.detect == 'inside'){
            if(!(fenceResult.includes(results.id))){
                console.log(fenceKey+' entered the Fence ' + fenceToDo.id + ': '+ results.id);
                fenceResult.unshift(results.id);
                let result = {
                    id: fenceToDo.id,
                    results: fenceResult
                }
                fencesResults.unshift(result)
```

Figura 5.16: Esecuzione di una live geo-fence

Per quanto riguarda la logica architetturale, le posizioni delle entità all'interno del server di Tile38 vengono aggiornate dal backend a seguito dell'aggiornamento che le entità simulate

inviano al backend. Questo avviene con la stessa operazione mostrata in Figura 5.14, andando a sostituire la vecchia entità con una con posizioni aggiornate (la sostituzione avviene in quanto la nuova entità inserita possiede *key* e *id* identiche a quella precedente). Ogni volta, quindi, che il backend viene notificato di un aggiornamento della posizione delle entità, in maniera asincrona esso sostituirà l'entità nel server Tile38 per aggiornarne la posizione. Tile38 non prevede un metodo diverso per l'aggiornamento dei metadati delle entità.

Capitolo 6

Esempi d'Uso

Sommario

In questo capitolo vengono mostrati due casi di studio inerenti alle diverse entità modellate in questo progetto.
Per quanto riguarda l'entità *Veicolo* si mostra un esempio d'uso dell'applicazione per monitorare lo stato del traffico in una città, configurando delle geo-fences in incroci stradali strategici, che si attivano e notificano l'utente nel caso in cui nei pressi di quella zona si presenti un elevato tasso di traffico veicolare, risultando quindi in un tratto stradale da evitare.
Per quanto riguarda l'entità *Paziente* si mostra un esempio d'uso dell'applicazione per monitorare le zone di rischio assembramento per pazienti a rischio elevato di contrarre una malattia. Si configurano quindi delle geo-fences in luoghi importanti per i pazienti (centri commerciali, piazze, negozi) che si attivano e notificano l'utente di evitare le zone colme di persone che potrebbero aggravare la condizione del paziente trasmettendogli una malattia.

6.1 Traffico veicolare intenso

Il file di configurazione per questo caso di studio è riportato in Figura 6.1, dove si possono notare:

- *center*, le coordinate inserite corrispondono ad un quartiere di Pechino (Cina);
- *entity*, sono stati scelti i veicoli;
- *indexes*, sono stati inseriti gli indici di rumore con associato un criterio di visualizzazione in terzili; Vibrazioni con associato un criterio di visualizzazione in quartili; Carburante con associato un criterio di visualizzazione in linguaggio espressivo per veicoli con carburante al di sotto del 30 per cento; Ergonomia con associato un criterio di visualizzazione in linguaggio espressivo per veicoli con indice di ergonomia al di sotto del 30 per cento;
- *values*, i valori da visualizzare nella forma tabellare in fondo alla schermata.
- *fences*, l'identificativo, la forma, le coordinate di centratura e il raggio d'azione dei triggers geo-localizzati (500 metri dal punto centrale).

Figura 6.1: File di configurazione

```
"center": [39.984702,116.318417],
"entity": "vehicles",
"indexes": [
  { "component": "IndexButton","text": "Noise", "criterion": "tertile" },
  { "component": "IndexButton", "text": "Vibrations", "criterion": "quartile" },
  { "component": "IndexButton", "text": "Fuel", "criterion": "vehicles[.fuel < 30]" },
  { "component": "IndexButton", "text": "Ergonomics", "criterion": "vehicles[.ergonomics < 30]" }
],
"values": [
  { "text": "Vehicle", "value": "vehicle" },
  { "text": "Manufacturer", "value": "manufacturer" },
  { "text": "Model", "value": "model" },
  { "text": "Type", "value": "type" },
  { "text": "Vin", "value": "vin" },
  { "text": "Noise (db/A)", "value": "noise" },
  { "text": "Vibrations (m/s^2)", "value": "vibrations" },
  { "text": "Fuel %", "value": "fuel" },
  { "text": "Ergonomics %", "value": "ergonomics" }
],
"fences": [
  { "component":"FenceTextArea", "circle":"CircleFence", "id": "Tsinghua Science and Technology Park",
    "lat": "39.994", "lng": "116.326", "radius": "500" },
  { "component":"FenceTextArea", "circle":"CircleFence", "id": "Southern Daoxiangyuan Community",
    "lat": "39.981", "lng": "116.300", "radius": "500" },
  { "component":"FenceTextArea", "circle":"CircleFence", "id": "Qinghua Road",
    "lat": "39.999", "lng": "116.314", "radius": "500" },
  { "component":"FenceTextArea", "circle":"CircleFence", "id": "East Zhongguancun Road",
    "lat": "39.983", "lng": "116.328", "radius": "500" }
]
```

Il risultato a schermo si può osservare in Figura 6.2, dove si può notare la mappa centrata nelle coordinate indicate, la presenza di cento markers relativi alle entità veicolari simulate, le aree cerchiate di rosso corrispondenti alle geo-fences indicate nel file, e sulla sinistra della schermata vi sono gli indici di monitoraggio e selezione delle entità e le aree di testo in cui vengono indicati i dettagli delle entità che attivano i triggers geo-localizzati. In questo caso specifico le geo-fences sono posizionate all'altezza di strade e di incroci molto trafficati e con alto tasso di incidenti.

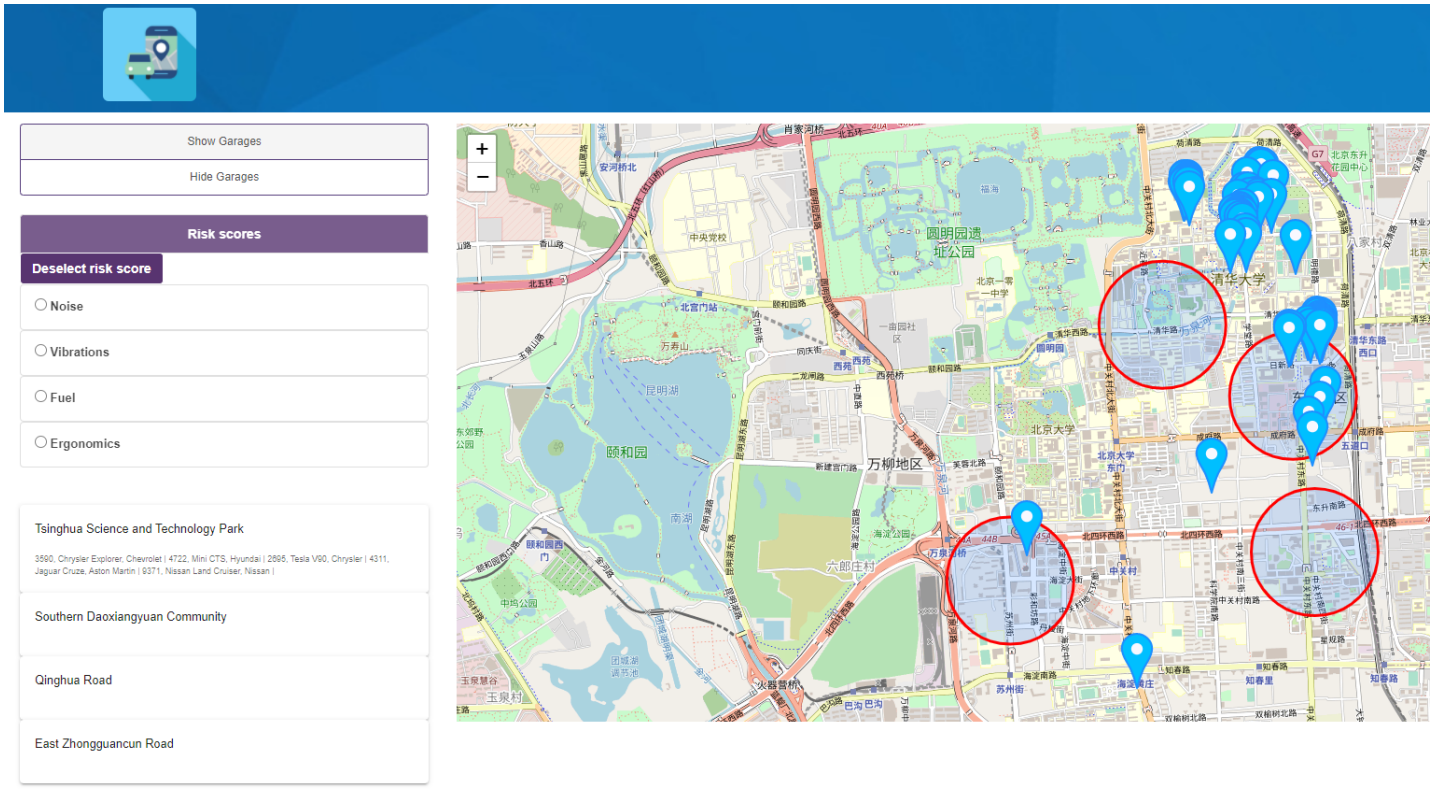


Figura 6.2: Mappa con i markers

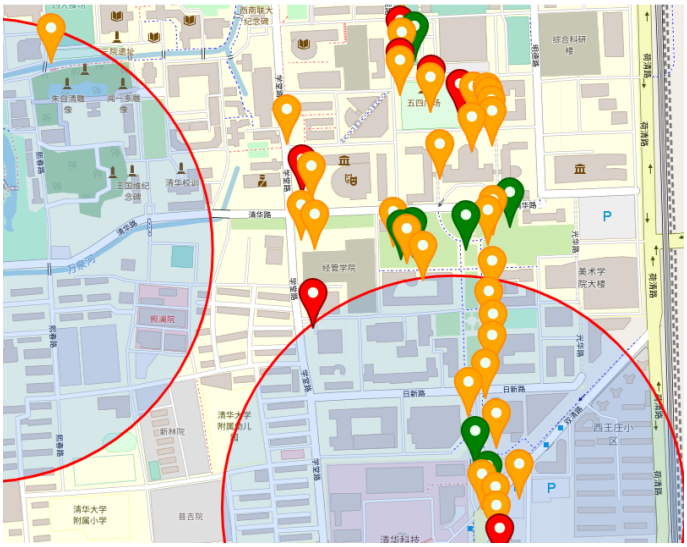
In Figura 6.3 si possono notare i dati descrittivi delle cento entità a schermo in una rappresentazione tabellare. All'interno di questa forma è possibile digitare in maiuscolo delle lettere per una ricerca rapida di un'eventuale entità di interesse spicifica, e/o selezionare singolarmente o in maniera collettiva più entità. Questa selezione cambierà il colore del marker dell'entità selezionata, centrando la visuale della mappa su di essa.

Figura 6.3: Tabella dei dati

Nel caso in cui seleziono un indice di visualizzazione, i markers delle entità vengono eviden-

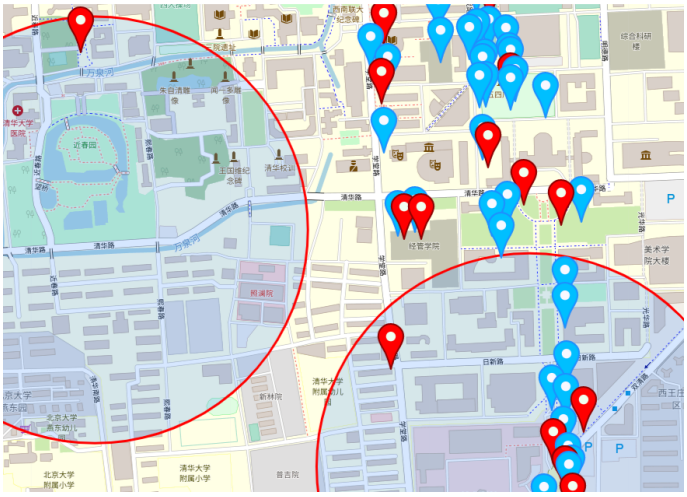
ziati, tramite un cambiamento del colore dell'icona, secondo il criterio scelto in fase di editing del file di configurazione. Per esempio, come si nota dalla Figura 6.4, selezionando l'indice *Noise*, che è associato ad un criterio in terzili, vengono evidenziati tutti i markers colorando di verde i veicoli meno rumorosi, di arancione quelli mediamente rumorosi, e di rosso quelli più rumorosi. In questa maniera, già da questa prima visuale, un utente è in grado di evitare tratti di strada più rumorosi di altri.

Figura 6.4: Indice di rumore selezionato in terzili



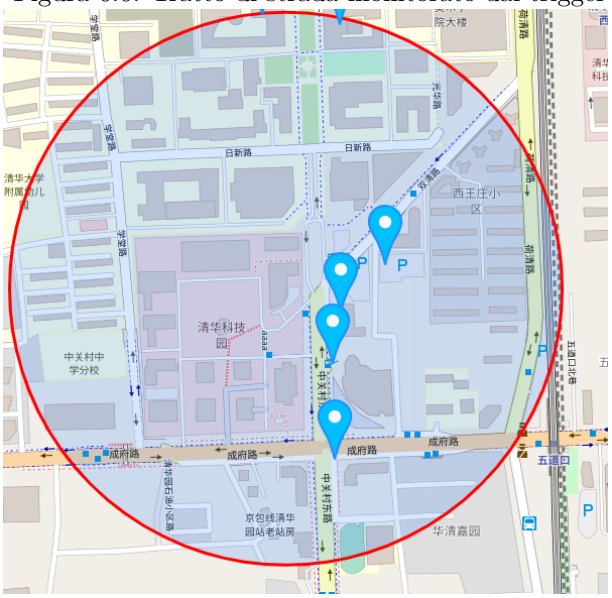
Si mostra inoltre, in Figura 6.5, la differenza di evidenziazione dei markers tra un criterio in terzili/quartili e uno scritto in linguaggio espressivo. I markers che corrispondono al criterio vedono la loro icona colorata di rosso, come nel caso qui mostrato, dove al selezionamento dell'indice di carburante, vengono evidenziati effettivamente solo i veicoli con una percentuale di carburante al di sotto del 30 per cento.

Figura 6.5: Indice di carburante selezionato in linguaggio espressivo



Nella fase iniziale dell'esperimento, è stata scelta un'area che, per esempio, può essere etichettata come altamente trafficata e con alti tassi di incidenti stradali quando il tasso di traffico veicolare è intenso. Quindi è stata posizionata una geo-fence per monitorare la situazione, come si può notare in Figura 6.6.

Figura 6.6: Tratto di strada monitorato dal trigger



Nel momento in cui i veicoli all'interno dell'area monitorata determinano, secondo la politica scelta, uno stato di intenso traffico (Figura 6.7), l'utente riceve un messaggio di allerta da parte del sistema, notificando lo stato del traffico in quel tratto stradale e invitandolo a non percorrerlo (Figura 6.8).

Figura 6.7: Tratto di strada monitorato dal trigger, veicoli numerosi

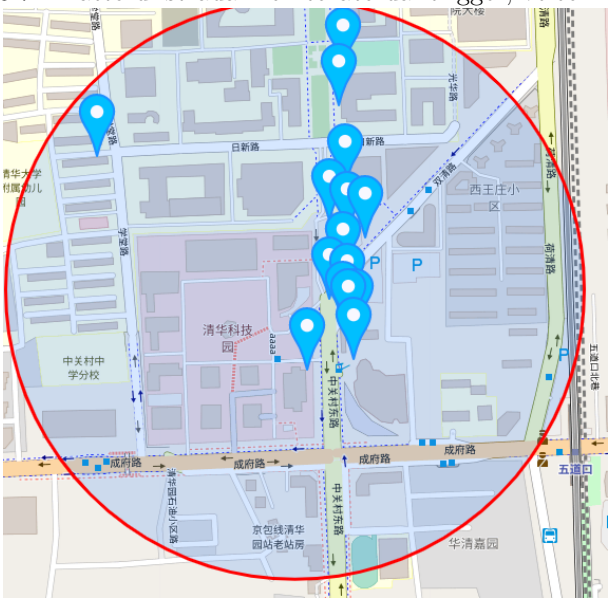
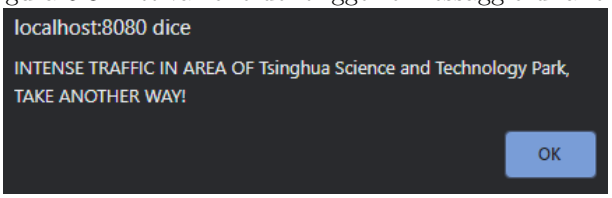


Figura 6.8: Attivazione del trigger e messaggio di allerta



6.2 Zona di assembramento epidemico

Il file di configurazione per questo casodi studio è riportato in Figura 6.9, dove si possono notare:

- *center*, le coordinate inserite che corrispondono ad un quartiere di Pechino (Cina);
- *entity*, sono stati scelti i pazienti;
- *indexes*, sono stati inseriti tutti gli indici di rischio, tra cui LACE con associato un criterio di visualizzazione in terzili; Charlston con associato un criterio di visualizzazione in quartili; ASA con associato un criterio di visualizzazione di filtraggio per i soli pazienti ASA 1; GMA con associato un criterio di visualizzazione di filtraggio per i soli pazienti con l'indice maggiore di zero; Barthel con associato un indice di visualizzazione in terzili;
- *values*, i valori da visualizzare nella forma tabellare in fondo alla schermata;
- *fences*, l'identificativo, la forma, le coordinate di centratura e il raggio d'azione dei triggers geo-localizzati (150 metri dal punto centrale).

Figura 6.9: File di configurazione

```
"center": [39.984702,116.318417],
"entity": "patients",
"indexes": [
  { "component": "IndexButton","text": "Lace", "criterion": "tertile" },
  { "component": "IndexButton", "text": "Charlston", "criterion": "quartile" },
  { "component": "IndexButton", "text": "ASA", "criterion": "patients[.asa == 'I']" },
  { "component": "IndexButton", "text": "GMA", "criterion": "patients[.gma > 0]" },
  { "component": "IndexButton", "text": "Barthel", "criterion": "tertile" }
],
"values": [
  { "text": "Name", "value": "firstName" },
  { "text": "Surname", "value": "lastName" },
  { "text": "Retrieval", "value": "retrieval" },
  { "text": "Selfcare", "value": "selfcare" },
  { "text": "Dwelling", "value": "dwelling" },
  { "text": "Career", "value": "career" },
  { "text": "LACE", "value": "lace" },
  { "text": "Charlson", "value": "charlson" },
  { "text": "Barthel", "value": "barthel" },
  { "text": "ASA", "value": "asa" },
  { "text": "GMA", "value": "gma" }
],
"fences": [
  { "component":"FenceTextArea", "circle":"CircleFence", "id": "Wudaokou",
    "lat": "39.99933924950308", "lng": "116.32642656808596", "radius": "150" }
]
```

Il risultato a schermo si può osservare in Figura 6.10, dove si può notare la mappa centrata nelle coordinate indicate, la presenza di cento markers relativi alle entità dei pazienti simulate, l'area cerchiata di rosso corrispondente alla geo-fence indicata nel file, e sulla sinistra vi sono gli indici di monitoraggio e selezione delle entità e l'area di testo in cui vengono indicati i dettagli delle entità che attivano il trigger geolocalizzato. In questo caso specifico, la geo-fence è posizionata all'altezza di un parco molto frequentato a Pechino, quindi a rischio di sovraffollamento e assembramento epidemico.

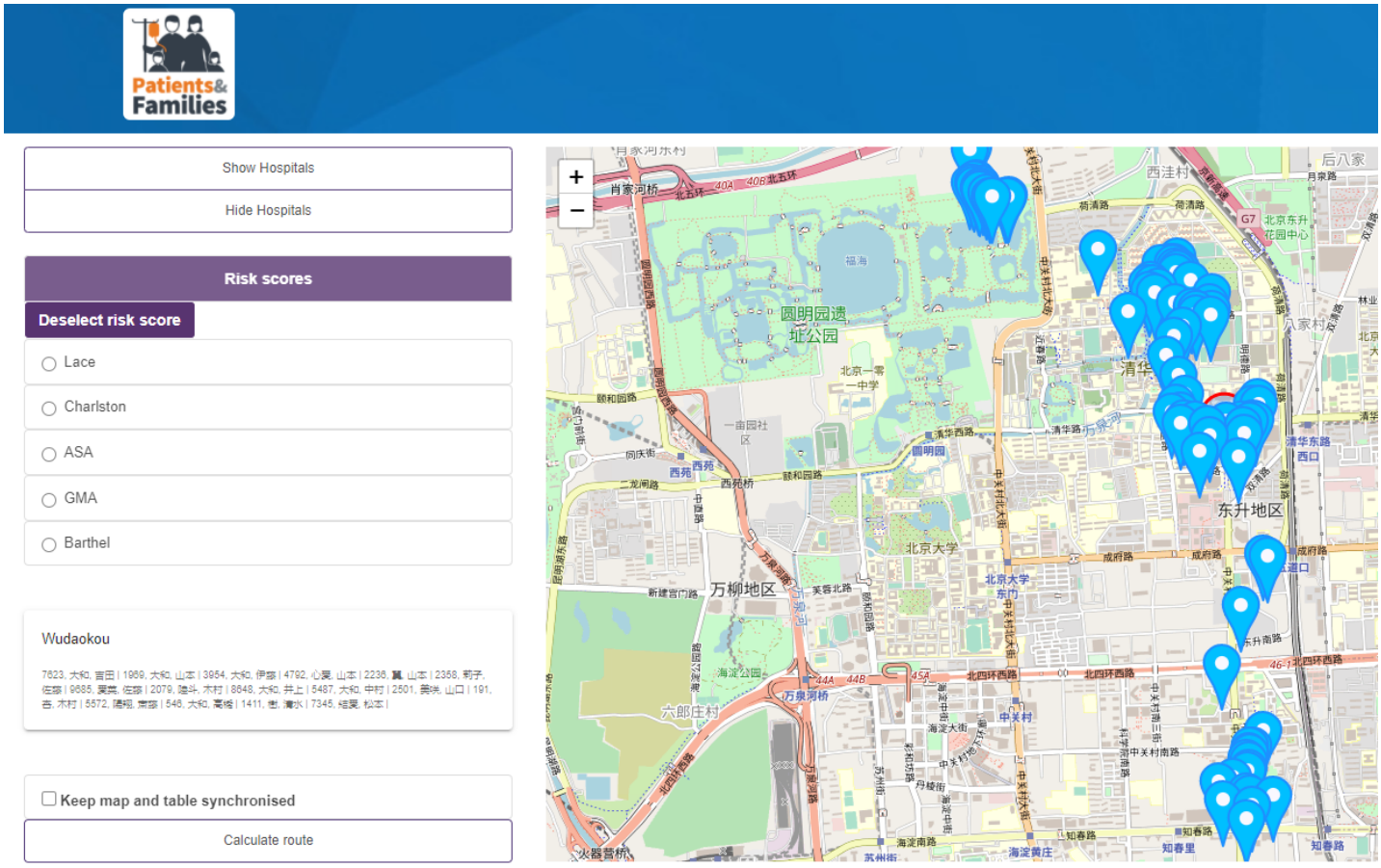


Figura 6.10: Mappa con i markers dei pazienti

In Figura 6.11 si possono notare i dati descrittivi delle cento entità a schermo in una rappresentazione tabellare. All'interno di questa forma è possibile digitare in maiuscolo delle lettere per una ricerca rapida di un'eventuale entità di interesse spicifica, e/o selezionare singolarmente o in maniera collettiva più entità. Questa selezione cambierà il colore del marker dell'entità selezionata, centrando la visuale della mappa su di essa. Da notare come la libreria JavaScript *faker.js* abbia permesso di rendere più realistici i nominativi con la localizzazione cinese dei nomi e dei cognomi dei pazienti.

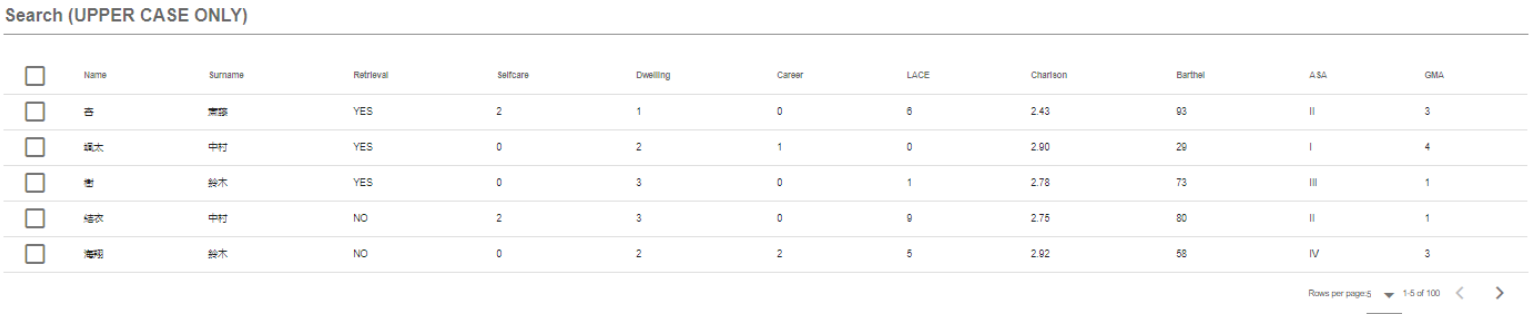
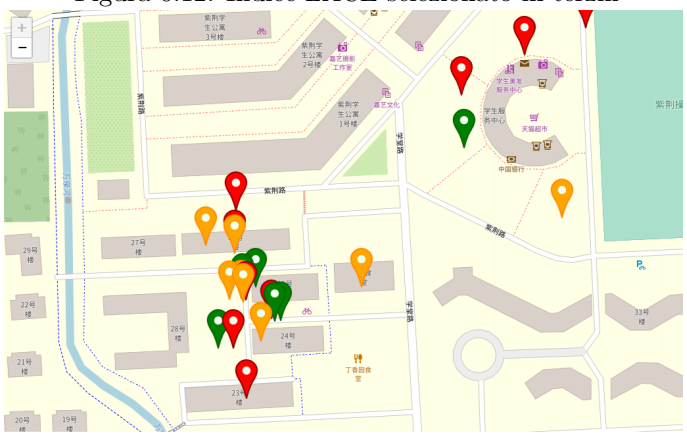


Figura 6.11: Tabella dei dati dei pazienti

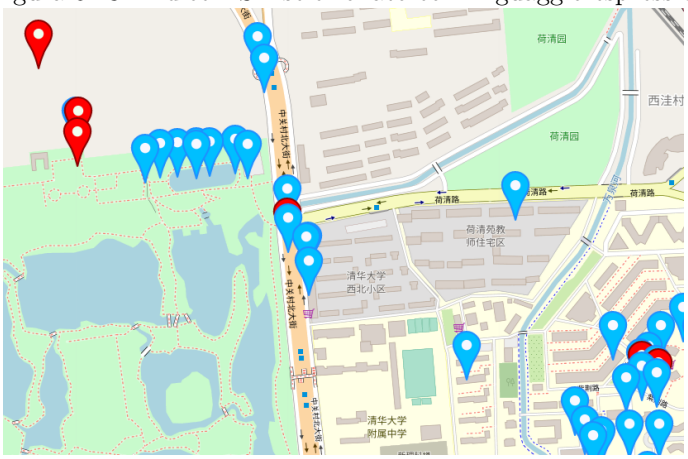
Nel caso in cui seleziono un indice di visualizzazione, i markers delle entità vengono evidenziati, tramite un cambiamento del colore dell'icona, secondo il criterio scelto in fase di editing del file di configurazione. Per esempio, come si nota dalla Figura 6.12, selezionando l'indice *LACE*, che è associato ad un criterio in terzili, vengono evidenziati tutti i markers colorando di verde i pazienti con un rischio di ri-ospedalizzazione minore, di arancione quelli con un indice medio, e di rosso quelli con un alto rischio di ri-ospedalizzazione. In questo modo, già da questa prima visuale, un utente, per esempio un medico, è già in grado di osservare quali pazienti ad alto rischio effettuano azioni o vanno in luoghi che aumenterebbero ulteriormente il loro indice.

Figura 6.12: Indice LACE selezionato in terzili



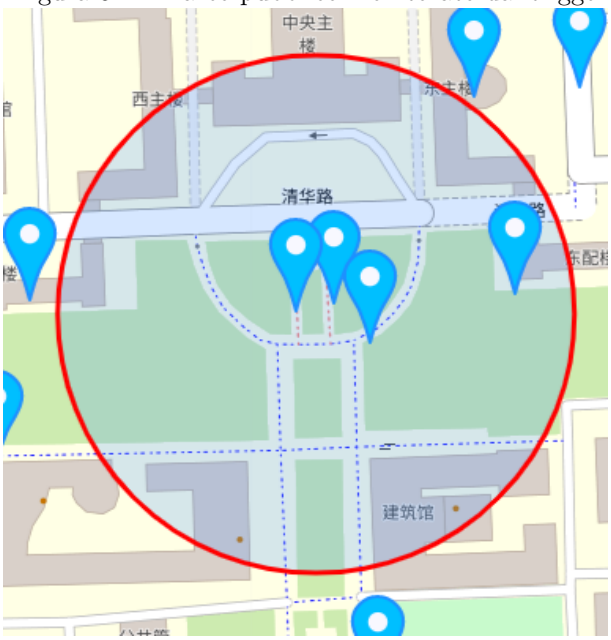
Si mostra inoltre, in Figura 6.13, la differenza di evidenziazione dei markers tra un criterio in terzili/quartili e uno scritto in linguaggio espressivo. I markers che corrispondono al criterio vedono la loro icona colorata di rosso, come nel caso qui mostrato, dove al selezionamento dell'indice di ASA, vengono evidenziati effettivamente solo i pazienti con l'indice pari a 1.

Figura 6.13: Indice ASA selezionato con linguaggio espressivo



Nella fase iniziale dell'esperimento, è stato scelto un luogo che, per esempio, può essere etichettato come pericoloso e rischioso per quanto riguarda l'indice di sovraffollamento e assembramento epidemico per i pazienti più a rischio. Quindi è stata posizionata una geo-fence per monitorare la situazione, come si può notare in Figura 6.14.

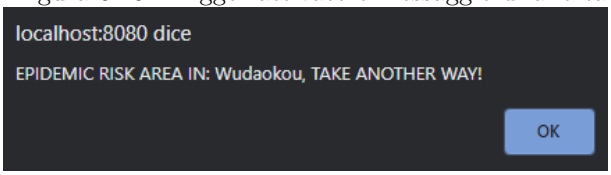
Figura 6.14: Parco pubblico monitorato dal trigger



Nel momento in cui i pazienti all'interno dell'area monitorata determinano, secondo la politica scelta, uno stato di sovraffollamento e rischio di assembramento epidemico (Figura 6.15), l'utente riceve un messaggio di allerta da parte del sistema, notificando lo stato de popolamento di quello specifico parco pubblico, e invitandolo ad evitarlo (Figura 6.16).

Figura 6.15: Parco pubblico monitorato dal trigger, assembramento

Figura 6.16: Trigger attivato e messaggio di allerta



Note

Alcune note che risulta necessario specificare:

- Se lo si desidera è possibile, sempre tramite il file di configurazione, inserire e rimuovere geo-fences anche a run-time.
- Se lo si desidera, è possibile cambiare il contenuto del file di configurazione a run-time e ricaricando la pagina web è possibile vederne i risultati.

Capitolo 7

Conclusioni

Sommario

7.1 Obiettivi raggiunti

Concludendo, il lavoro svolto è stato sicuramente soddisfaciente per quanto rigurada il raggiungimento degli obiettivi proposti. In particolare si possono elencare i seguenti punti:

- Sviluppo di un'applicazione web modulare ed altamente configurabile tramite l'utilizzo delle più moderne tecnologie open-source.
- Perfetta integrazione di servizi esterni tramite API e applicativi locali.
- Applicazione altamente dinamica dal punto di vista dell'utilizzo da parte dell'utente. Si tratta, infatti, di un'applicazione in grado di adattarsi all'utilizzo di un utente specializzato per un monitoring specifico (e.g. un medico) od un utente senza alcun tipo di preparazione o specializzazione.
- Progettazione dell'applicazione in maniera compatibile agli eventuali sviluppi futuri.
- Performance accettabili in termini di latenza della comunicazione.
- Funzionalità di monitoring e visualizzazione altamente personalizzabili, soprattutto in termini di arbitrarietà nei confronti dei criteri di visualizzazione, tramite il linguaggio espressivo, e nei confronti dei triggers geolocalizzati, manipolabili nelle loro posizioni, forme, raggio d'azione e persino comportamento, anche in real-time.

Chiaramente vi sono ancora numerose migliorie, soprattutto estetiche, da poter effettuare e funzionalità da implementare. Esse sono indicate nella prossima sezione.

7.2 Sviluppi futuri

Tra i lavori futuri si possono inserire i seguenti:

- Costruzione di un'interfaccia utente di configurazione che nasconda la complessità ora richiesta, nella compilazione del file, con semplici menu interattivi;
- Allo scopo del punto di cui sopra, implementazione della funzionalità di geocoding per la traduzione di indirizzi fisici reali in coordinate geografiche;
- Implementazione di un servizio di login/autenticazione degli utenti in modo tale da rendere possibile la multi-utenza nel sistema.
- Esplorare altri possibili servizi di geo-fencing che possano essere intallati su server raggiungibili, rendendo possibile usufruire delle funzionalità dell'applicazione anche tramite dispositivi mobili.
- Esplorazione di ulteriori tecniche di geo-fencing, tra cui le fences mobili;

- Ampliamento delle tipologie di entità gestite dal sistema;
- Ritrovamento o generazione di dati reali da inserire in un database fisico per completare lo stack MEVN e usufruire di dati persistenti. Questo porterebbe all'ininfluenza delle entità simulate, anche se in alcuni esperimenti potrebbero comunque risultare utili.
- Considerare l'opzione di spostarsi verso un'implementazione *cloud computing*, portando benefici sia in termini di efficienza sia in termini di sicurezza, anche se ad un costo.
- Effettuare una valutazione costi/benefici per il punto di cui sopra.

Bibliografia

Capitolo 1

[1] Stefano Mariani et al. “Deliver intelligence to integrate care: the Connecare way.” In: *International Journal of Integrated Care (IJIC)* 19 (2019).

[2] Azat Mardan. *Express. js Guide: The Comprehensive Book on Express. js*. Azat Mardan, 2014.

[3] Olga Filipova. *Learning Vue. js 2*. Packt Publishing Ltd, 2016.

[4] Stefan Tilkov e Steve Vinoski. “Node. js: Using JavaScript to build high-performance network programs”. In: *IEEE Internet Computing* 14.6 (2010), pp. 80–83.

[5] Michael A Sheha e Angie Sheha. *Method and system for identifying and defining geofences*. US Patent 8,019,532. Set. 2011.

[6] Matthew Zarem, Eric Vuillermet e John Deaguiar. *Intelligent reverse geocoding*. US Patent 8,731,585. Mag. 2014.

[7] Abdullah A Balkhair. “COVID-19 pandemic: a new chapter in the history of infectious diseases”. In: *Oman medical journal* 35.2 (2020), e123.

Capitolo 2

[1] Stefano Mariani et al. “Deliver intelligence to integrate care: the Connecare way.” In: *International Journal of Integrated Care (IJIC)* 19 (2019).

[2] Microsoft. *Download GeoLife GPS Trajectories from Official Microsoft Download Center*. URL: `https://www.microsoft.com/en-us/download/details.aspx?id=52367&from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fdownloads%2Fb16d359d-d164-469e-9fd4-daa38f2b2e13%2F`.

[3] Yu Zheng et al. “Mining interesting locations and travel sequences from GPS trajectories”. In: *Proceedings of the 18th international conference on World wide web*. 2009, pp. 791–800.

[4] Yu Zheng et al. “Understanding mobility based on GPS data”. In: *Proceedings of the 10th international conference on Ubiquitous computing*. 2008, pp. 312–321.

[5] Yu Zheng, Xing Xie, Wei-Ying Ma et al. “Geolife: A collaborative social networking service among user, location and trajectory.” In: *IEEE Data Eng. Bull.* 33.2 (2010), pp. 32–39.

[6] Marak. *Faker.js - generate massive amounts of fake data in the browser and node.js.* URL: https://github.com/marak/Faker.js/.

[7] Andrea Nicolini - Università degli Studi di Perugia. *Indici di Valutazione del Rumore.* URL: http://www.ciriaf.it/ft/File/Didattica/lezioni/nicolini_IAA/D_indici_valutazione_rumore.pdf.

[8] Directorate-General for Internal Policies. *Sound Level of Motor Vehicles.* URL: https://www.europarl.europa.eu/document/activities/cont/201312/20131205ATT75547/20131205ATT75547EN.pdf.

[9] INAIL - Istituto Nazionale per l’Assicurazione contro gli Infortuni sul Lavoro. *100 misure di vibrazioni in ambiente lavorativo.* URL: https://www.inail.it/cs/internet/comunicazione/pubblicazioni/catalogo-generale/100-misure-di-vibrazioni-in-ambiente-lavorativo.html.

[10] INAIL - Istituto Nazionale per l’Assicurazione contro gli Infortuni sul Lavoro. *La valutazione del rischio vibrazioni.* URL: https://www.inail.it/cs/internet/comunicazione/pubblicazioni/catalogo-generale/pubbl-valutazione-del-rischio-vibrazioni.html#:~:text=Il%20volume%20si%20propone%20come,tecniche%20e%20scientifiche%20in%20materia..

[11] Politecnico di Milano. *L’Ergonomia nella Normativa.* URL: http://www.laboratoriopoliziademocratica.org/SALUTE/studi%20ergonomici.pdf.

[12] Salvatore Speciale. *La mia esperienza come burocrate intelligente della Geriatria.* URL: http://www.grg-bs.it/usr_files/eventi/journal_club/21_10_2011_speciale.pdf.

[13] Besler. *How to calculate the LACE risk score.* URL: https://www.besler.com/lace-risk-score/#:~:text=The%20LACE%20index%20identifies%20patients,within%20thirty%20days%20of%20discharge.&text=is%20widely%20used.-,The%20LACE%20index%20identifies%20patients%20that%20are%20at%20risk%20for,stay%20of%20the%20index%20admission..

[14] Centrostudi Gised. *Calcola l’Indice di Comorbidità di Charlson (CCI).* URL: https://www.centrostudigised.it/calcola_indice_charlson.html.

[15] Servizio Sanitario Regionale Emilia Romagna. *GLI STRUMENTI DI VALUTAZIONE DEL PAZIENTE: LA SCALA DI BARTHEL.* URL: http://www.asmn.re.it/gli-strumenti-di-valutazione-del-paziente-la-scala-di-barthel#:~:text=L'indice%20di%20Barthel%20verr%C3%A0,che%20alla%20dimissione%20dall'Ospedale..

[16] David Monterde, Emili Vela, Montse Clèries et al. “Adjusted morbidity groups: A new multiple morbidity measurement of use in Primary Care”. In: *Atencion primaria* 48.10 (2016), pp. 674–682.

[17] Jaime Barrio-Cortes et al. “Adjusted morbidity groups: Characteristics and comorbidities in patients with chronic conditions according to their risk level in Primary Care”. In: *Atencion primaria* 52.2 (2020), pp. 86–95.

[18] Montallegro. *Rischio Anestesiologico e classificazione ASA*. URL: https://www.montallegro.it/servizi/anestesiologia-e-rianimazione/procedure/rischio-anestesiologico-e-classificazione-asa/#:~:text=ASA%201%3A%20Paziente%20sano%3B,malattia%20grave%20con%20limitazione%20importante..

[19] Ivan Dueñas-Espín et al. “Proposals for enhanced health risk assessment and stratification in an integrated care scenario”. In: *BMJ open* 6.4 (2016), e010301.

Capitolo 3

[1] Alberto Bottarini. *Vue.js un'introduzione*. URL: https://www.html.it/pag/63947/vue-js-unintroduzione/.

[2] Vue.js. *The Progressive JavaScript Framework*. URL: https://vuejs.org/.

[3] Shajia Abidi. *Building an interactive map with Vue and Leaflet*. URL: https://blog.logrocket.com/building-an-interactive-map-with-vue-and-leaflet/.

[4] Vue Leaflet. *Introduction*. URL: https://vue2-leaflet.netlify.app/components/.

[5] HERE Deveoper. *Build apps with HERE Maps API*. URL: https://developer.here.com/.

[6] Tile38. *Installation - Tile38*. URL: https://tile38.com/topics/installation.

Capitolo 4

[1] Google. *Cloud computing services*. URL: https://cloud.google.com/.

[2] Amazon. *Amazon Web Services (AWS)*. URL: https://aws.amazon.com/it/.

[3] Microsoft. *Microsoft cloud services*. URL: https://azure.microsoft.com/it-it/.

[4] ArpitAsati. *What is web socket and how it is different from the HTTP?* URL: https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/.

[5] Heroku. *Heroku apllication cloud service.* URL: https://www.heroku.com/.

Capitolo 5

[1] mozjexl. *mozJexl is a fork of Jexl for use at Mozilla, specifically as a part of SHIELD and Normandy.* URL: `https://www.npmjs.com/package/mozjexl`.

[2] HERE Deveoper. *Build apps with HERE Maps API.* URL: `https://developer.here.com/`.

[3] Tile38. *Command examples.* URL: `https://github.com/phulst/node-tile38`.