

Geo-fencing Platform

22/03/2020


Stefano Balla, Federica La Piana, Gianluca Spiller

Introduzione

Il progetto “Geo-fencing Platform” si propone come una piattaforma di disseminazione di messaggi di contenuto informativo, sulla base della posizione utente.

Dal punto di vista tecnico, possiamo descrivere tale progetto come un sistema di creazione e fruizione di geofence composto da un software server dove vengono inserite le informazioni relative ai geofence e un software mobile che permette agli utenti di inviare la propria posizione al server (incluso il tipo di mobilità) e ricevere notifiche relative al luogo in cui stanno circolando.

Con “Geo-fencing” si intende un servizio che attiva un’azione specifica quando una dispositivo entra in un confine virtuale impostato attorno ad un’area geografica.



Il contesto d'uso può essere ad esempio quello delle applicazioni di marketing o di turismo. Lo strumento predefinito per queste applicazioni è lo smartphone, dal momento che è un dispositivo GPS-abilitato che tutti possiedono e portano sempre con sé, fornendo una pletora di dettagli riguardo la propria posizione, direzione e mezzo di locomozione. Infatti, grazie ai sensori integrati su questi dispositivi è possibile tramite un app installata impostare delle azioni in base a come l'utente si muove nello spazio.

Sono previste due categorie nel target del progetto:

- Utente che utilizza l'app mobile
- Amministratore che visualizza uno strumento di analisi di dati degli utenti.


Progettazione

Client mobile

L'applicazione client realizzata è un'applicazione mobile Android con una struttura molto semplice:

- *Main Activity*: Permette l'interazione finale dell'utente con l'app
 - Layout XML: Interfaccia grafica dell'app
 - File Java
- *Constants*: Classe contenente i parametri globali per "affinare" il riconoscimento della mobilità
- *DetectedActivitiesIntentService*: classe che estende *IntentService* e, su richiesta (gestione dell'intent), aggiorna la lista delle attività in corso più probabili. Tali attività vengono poi inviate in broadcast usando un *LocalBroadcastManager*.
- *BackgroundDetectedActivitiesService*: viene eseguita in background facendo attivare le attività in base ad un intervallo temporale.
 - *MyLocationService*: Utilizza le API per la localizzazione dei servizi di Google Play richiedendo, a intervalli di tempo regolari, l'ultima posizione nota del dispositivo

L'applicazione client dialoga occasionalmente con l'infrastruttura server per l'invio e la ricezione dei dati utili inviando richieste HTTP a un endpoint RESTful.



Nello specifico, tramite la libreria “Volley”, il client effettua delle richieste contenenti i dati di posizione e di mobilità, ottenendo come risposta un messaggio che può essere un link da caricare in una webView, qualora si trovi dentro a un geofence, oppure , in caso contrario, una messaggio di negazione.

Librerie

- com.karumi:dexter:6.0.2
- com.cocoahero.android:geojson:1.0.1@jar
- com.android.volley:volley:1.1.0
- com.nabinbhandari.android:permissions:3.8
- com.google.android.gms:play-services-location:11.8.0

Server

Il server è stato concepito prevalentemente come una repository centralizzata delle informazioni relative ai geofences. Le sue funzioni principali sono dunque:

- Gestione dei geofences memorizzati nel database
- Gestione e memorizzazione delle posizioni degli utenti all'interno del database
- Comunicazione con l'app mobile

Il server è scritto in linguaggio *javascript* e utilizza *Express*, ossia un framework di Node.js.

La struttura del Server è la seguente:

- *App.js*: gestisce le chiamate RESTful
- *query.js*: contiene le query utilizzate in *App.js*

Database

Nel server i dati vengono conservati prevalentemente su database *Postgres PostGIS*.

Tale database conserva tutti i dati relativi ai geofences e alle posizioni inviate dalle app mobile. Ciò rende possibile l'utilizzo delle funzioni geospaziali di PostGis, permettendo il calcolo delle inferenze fra geofences e posizioni degli utenti.

Le tabelle principali sono:

- *traiettorieSpaz*: Tiene traccia dei singoli punti di localizzazione (longitudine-latitudine) inviati dalle app mobile con mobilità “walk”.
- *traiettorieBike*: Tiene traccia dei singoli punti di localizzazione (longitudine-latitudine) inviati dalle app mobile con mobilità “bike”.
- *traiettorieCar*: Tiene traccia dei singoli punti di localizzazione (longitudine-latitudine) inviati dalle app mobile con mobilità “car”.
- *geofence_walk*: Tiene traccia dei geofence relativi alla mobilità “walk” memorizzati come poligoni, ossia come insiemi di punti spaziali.
- *geofence_bike*: Tiene traccia dei geofence relativi alla mobilità “bike” memorizzati come poligoni, ossia come insiemi di punti spaziali.
- *geofence_car*: Tiene traccia dei geofence relativi alla mobilità “car” memorizzati come poligoni, ossia come insiemi di punti spaziali.

<div>traiettoriespaz</div> <div><div>id: varcharing</div><div>date: timestamp with time zone</div><div>geom: geometry</div></div>	<div>traiettorieWalk</div> <div><div>id: varcharing</div><div>date: timestamp with time zone</div><div>geom: geometry</div></div>	<div>traiettorieBike</div> <div><div>id: varcharing</div><div>date: timestamp with time zone</div><div>geom: geometry</div></div>
<div>geofenceCar</div> <div><div>id: integer</div><div>message: varcharing</div><div>geom: geometry</div></div>	<div>geofenceWalk</div> <div><div>id: integer</div><div>message: varcharing</div><div>geom: geometry</div></div>	<div>geofenceBike</div> <div><div>id: integer</div><div>message: varcharing</div><div>geom: geometry</div></div>
<div>avetime</div> <div><div>id: varcharing</div><div>d1: timestamp with time zone</div><div>geom: geometry</div></div>		

Front end

Per quanto riguarda il front end, è stato deciso di sviluppare sia un'interfaccia web mediante OpenLayer che un progetto QGIS con vari livelli di visualizzazione.

- *Openlayer*: libreria javascript per visualizzare mappe interattive nel browser web
- *QGIS*: Applicazione Desktop che permette di far confluire dati provenienti da diverse fonti in un unico progetto di analisi vettoriale. I dati vengono divisi in Layer e mostrati in una mappa territoriale personalizzabile dall'utente.

Il motivo del mantenimento di due piattaforme front end risiede nel fatto che, mentre quella implementata con OpenLayer consente maggiore libertà nell'inserimento di moduli aggiuntivi e nell'organizzazione dello spazio, quella su QGIS consente di testare nuove funzionalità senza effettuare troppe operazioni.

Inoltre, il mantenimento della piattaforma su QGIS è un valore aggiunto nel caso in cui l'utente amministratore desideri modificare lo stile dei livelli senza mettere mano al codice o voglia aggiungere funzionalità aggiuntive quali l'applicazione dei numerosi algoritmi di analisi predisposti da QGIS.

Implementazione

Client mobile

All'avvio dell'App, come prima cosa, viene chiesto all'utente di abilitare i permessi per la localizzazione e per il riconoscimento della mobilità. La richiesta delle autorizzazioni viene gestita tramite *Dexter*, una libreria di Android che consente di concedere o negare autorizzazioni durante l'esecuzione dell'app, anziché concederle tutte durante l'installazione dell'applicazione.

Le autorizzazioni in questione sono:

- `ACCESS_COARSE_LOCATION`: L'API può utilizzare il wi-fi o i dati delle celle mobili per determinare la posizione del dispositivo.
- `ACCESS_FINE_LOCATION`: L'API può determinare una posizione quanto più precisa dai provider di posizioni disponibili.

- **ACTIVITY_RECOGNITION**: L'API può determinare l'attività in corso d'esecuzione dell'utente sfruttando i dati ottenuti dai sensori di movimento integrati nel dispositivo (come accelerometro, giroscopio, magnetometro, etc.)

Di seguito vengono elencati i principali metodi coinvolti nei processi di localizzazione e riconoscimento della mobilità:

- *updateLocation()*: Utilizza l'oggetto *FusedLocationApi* che offre il riferimento al Provider di localizzazione e utilizza il metodo *RequestLocationUpdate* per richiedere notifiche al servizio di localizzazione.
- *buildLocationRequest()*: Consente di impostare i parametri per effettuare le richieste al servizio (intervallo ecc).
- *getPendingIntent()*: Poiché il service di localizzazione viene eseguito in background, al fine di aggiornare l'interfaccia utente è stato utilizzato un Broadcast Receiver.
- *onReceive()*: Invocato nella classe *MyLocationService* viene utilizzato per gestire l'aggiornamento di posizione e restituire le coordinate al Main .
- *handleUserActivity()*: controlla l'attività rilevata e il grado di confidenza (probabilità che l'attività in corso effettivamente sia quella rilevata);
- *startTracking()*: crea l'intent relativo al *BackgroundDetectedActivitiesService* e avvia il rispettivo service in esecuzione in background.

Per quanto riguarda la comunicazione con il Server, come accennato in precedenza, è stata utilizzata la libreria *Volley*. Più specificatamente, i dati di posizione vengono modellati in formato geoJSON (utilizzando un'opportuna libreria) e trasmessi al Server tramite una richiesta POST, in cui viene specificato l'URL.

Oltre ai dati di posizione (longitudine e latitudine), il geoJson viene popolato inserendo ulteriori parametri quali:

- Ora esatta della posizione
- Ora esatta dell'ultima notifica ricevuta dal Server
- ID anonimizzato del dispositivo
- Tipo di mobilità



Una volta effettuata la richiesta, il metodo *onResponse* gestisce la risposta:

- Se l'utente si trova all'interno di un geofence viene creata una *WebView* caricando il link ricevuto.
- Se l'utente non si trova nel geofence la risposta contiene una stringa di negazione che viene caricata nella *textView*.

Per quanto riguarda la rilevazione della mobilità, viene utilizzato un service eseguito in background e attivato dal metodo *startTracking()*.

Periodicamente, in base all'intervallo di aggiornamento impostato, il *BackgroundDetectedActivitiesService* controlla le ultime attività rilevate e invia l'aggiornamento.

Le API per l'activity recognition integrano, di default, un'ottimizzazione dal punto di vista energetico. L'intervallo di aggiornamento, quindi, non è seguito alla lettera, bensì verrà effettuato se necessario (o comunque di rado se l'utente mantiene lo stesso tipo di attività per un periodo di tempo prolungato).

L' *ActivityRecognitionClient* è l'oggetto che permette l'interazione con i servizi Google e il conseguente ottenimento dei risultati periodici di aggiornamento. A tal fine, viene utilizzato un *broadcast receiver* in ascolto per gli eventuali aggiornamenti.

Infine, Il broadcast receiver, tramite il metodo *onReceive()*, consente di comunicare al Main l'attività riconosciuta.

Uso delle risorse energetiche

Per quanto concerne l'attenzione all'uso delle risorse energetiche è stato pensato un metodo ad hoc riguardante l'intervallo di tempo trascorso il quale la posizione viene inviata al server. Infatti, adattando questo intervallo di tempo in base al tipo di mobilità, che è fortemente connessa alla velocità del dispositivo, si possono evitare numerosi aggiornamenti della posizione risparmiando risorse.

Per scegliere questi intervalli di tempo, sono state calcolate le velocità medie per ogni tipo di mobilità ed è stato scelto un intervallo spaziale anch'esso associato ad ogni tipo di mobilità. Gli intervalli spaziali sono stati scelti in rispetto della finalità del progetto, ovvero: un sistema di notifiche informative basato sulla posizione.

Inoltre, l'ordine di dimensione dei geofence iniziali è stato un punto di riferimento per questa scelta, infatti, gli intervalli spaziali devono garantire un limite di accuratezza affinché il sistema di notifica sia ragionevolmente preciso rispetto alla posizione.

Non per ultimo è stato preso in considerazione il contenuto delle notifiche per questa scelta, gli intervalli scelti pensiamo soddisfino i requisiti intrinsecamente imposti dal tipo di informazione

contenuta nei messaggi quali ad esempio, il suggerimento di rastrelliere per la bicicletta, colonnine per il parcheggio o siti di interesse turistico.

Walk

Tappe	Distanza (km)	Tempo (min)	Velocità (km/h)
Via Santo Stefano 68- Pz Verdi	1,00	13	4,61
Pz Verdi- Via Indipendenza	0,65	8	4,88
via Indipendenza- Pta S.Donato	1,10	14	4,72
Pt S.Donato- Pt Lame	2,0	25	4,80
Pt Lame- San Petronio	1,50	19	4,73
San Petronio- Parco Montagnola	1,30	15	5,20
Parco Montagnola- Sant'Orsola Malpighi	1,80	23	4,69
Sant'Orsola Malpighi- Teatro Duse	1,40	19	4,42
Teatro Duse - Basilica S.Francesco	1,60	19	5,05
Basilica S.Francesco- Le due Torri	1,10	13	5,07
Le due Torri- Pza Aldrovandi	0,50	6	5,00
Pza Aldrovandi - Pta Maggiore	0,65	8	4,87
Pta Maggiore- Pt San Vitale	0,45	5	5,4
Pt San Vitale- DISI	0,35	5	4,20

$$v_m = 4,83 \text{ km/h}$$


$$= 2.62 \times 10^{-2}$$

In quanto la deviazione standard è molto bassa è stato preso come punto di riferimento la velocità media.

L'intervallo spaziale scelto $i = 100m$ ottenendo quindi un tempo di aggiornamento $t_a = 74s$

Car

Tappe	Distanza (km)	Tempo (min)	Velocità (km/h)
Via Santo Stefano 68- Pz Verdi	3,90	14	16,71
Pz Verdi- Via Indipendenza	3,60	13	16,66
via Indipendenza- Pta S.Donato	1,70	7	14,65
Pt S.Donato- Pt Lame	2,70	9	11,33
Pt Lame - Giardini Margherita	3,70	9	24,66
Giardini MArgherita- via Donati Creti	4,20	10	25,30
Via Donato Creti- Pt San Felice	3,60	9	24,00
Pt San Felice- San Petronio	1,90	9	12,67
San Petronio- Pz Aldrovandi	1,60	8	12,00
Pz Aldrovandi- Sant'Orsola Malpighi	2,60	9	17,33
Sant'Orsola Malpighi- Teatro Duse	2,00	8	15,00
Teatro Duse - Pt Sant'Isaia	2,50	7	21,42
Pt Sant'Isaia- Parco	2,80	11	15,27



Montagnola			
Parco Montagnola - DISI	1,30	6	13,00

$$v_m = 17,90 \text{ km/h}$$

$$= 5,06$$

In questo caso la variazione standard non è bassa come nel caso precedente e questo è giustificato dal fatto che i percorsi scelti passino da strade più o meno scorrevoli e dal fatto che le strade del centro siano più lente. Ma sempre considerando lo scopo informativo delle notifiche è stato scelto di prendere in considerazione la velocità media.

L'intervallo spaziale scelto $i = 250 \text{ m}$ ottenendo quindi un tempo di aggiornamento $t_a = 50\text{s}$

Bike

Per la mobilità relativa alla bicicletta non è stato possibile applicare lo stesso metodo per mancanza di dati. Per questo è stata scelta la parte intera superiore dell'intervallo di velocità compreso fra la stima della velocità di corsa di un runner mediamente allenato $v = 11,99 \text{ km/h}$ e la velocità media della macchina.

L'intervallo spaziale scelto $i = 100 \text{ m}$ ottenendo quindi un tempo di aggiornamento $t_a = 27\text{s}$

Ritardo della notifica

Il calcolo del ritardo della notifica, ovvero quanto tempo intercorre da quando l'utente entra dentro il geofence a quando riceve la notifica è un misuratore importante delle prestazioni del sistema. L'implementazione di questo calcolo è affidata completamente al server utilizzando solo l'orologio del dispositivo mobile per evitare problemi relativi alla sincronizzazione.

L'applicazione Android salva ed invia due parametri, la data corrente (comprensiva di orario) e la data dell'ultima notifica ricevuta. Questa seconda variabile viene assegnata ogni volta che il device riceve una notifica.

E' chiaro che al primo uso dell'applicazione il secondo parametro sarà nullo.

I due parametri *currDate* e *notiTime* arrivano quindi al server sfalsati. Quest'ultimo inserisce i dati per ogni device tramite le funzioni *inserTime()* e *updateTime()*.

I dati sono salvati all'interno della tabella *avetime* con campi *id*, *currDate* e *notiTime*. I primi due costituiscono la chiave primaria mentre l'ultimo è un campo opzionale, questo consente di creare i record per i device che non hanno ancora ricevuto una notifica.

Infine, tramite la funzione *selectAvetime* viene calcolata la media del tempo di ritardo della notifica.

Server

Come accennato precedentemente, il server si occupa di gestire le richieste del client controllando l'eventuale presenza dell'utente all'interno di un geofence. Oltre a questo, memorizza i dati ricevuti ed estrae le informazioni relative ai tragitti degli utenti e ai geofence dal database.

Le funzioni in *App.js* vengono gestite in maniera asincrona, restituendo come risultato una *promessa* piuttosto che il loro valore effettivo. Tramite *await* il codice viene messo in pausa fino a quando la funzione adempie alla promessa, restituendo infine il risultato.

Il file *query.js* contiene i parametri per effettuare la connessione con il database Postgres e la dichiarazione delle query.

Di seguito vengono elencate le query più rilevanti:

- Inserimento della posizione dell'utente nel DB:

```
INSERT INTO traiettoriespaz ("id", "geom", "date") VALUES
('user01', ST_SetSRID(ST_MakePoint(44.4969683, 11.354965), 4326), '2020-03-03
09:19:25+01')
```
- Controllo della presenza dell'utente all'interno del geofence con eventuale selezione del messaggio da visualizzare

```
SELECT message
FROM geofence_walk
WHERE (ST_Within(ST_GeomFromText(POINT(point), 4326), geom))
```
- Selezione delle traiettorie degli utenti con trasformazione da punti a linee

```
SELECT DISTINCT ON (id) id, ST_AsText(ST_MakeLine(geom))
FROM traiettoriespaz
GROUP BY id
```
- Selezione del numero di punti all'interno di un geofence

```
SELECT geofence_walk.id, ST_AsText(geofence_walk.geom),
COUNT(DISTINCT(traiettoriespaz.id)) AS totale
```

```
FROM geofence_walk LEFT JOIN traiettoriespaz ON st_contains(geofence_walk.geom,
traiettoriespaz.geom)
GROUP BY geofence_walk.id,geofence_walk.geom
ORDER BY totale DESC
```

- Esecuzione dell'algoritmo K-mean (k=3) sui punti di posizione degli utenti

```
SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, tt.id, ST_AsText(geom)
FROM traiettoriespaz tt INNER JOIN
(SELECT id, MAX(date) AS MaxDateTime
FROM traiettoriespaz
GROUP BY id) groupedtt ON tt.id = groupedtt.id AND tt.date =
groupedtt.MaxDateTime
```
- Esecuzione dell'algoritmo DbScan sui punti di posizione degli utenti per ottenere suggerimenti su dove posizionare nuovi geofence

```
SELECT ST_ClusterDBSCAN(geom, 0.005, 1) OVER() AS cid, id, ST_AsText(geom)
FROM
(SELECT geom, tt.id, MaxDateTime
FROM traiettoriespaz tt INNER JOIN (SELECT id, MAX(date) AS MaxDateTime
FROM traiettoriespaz GROUP BY id) groupedtt ON tt.id = groupedtt.id AND tt.date =
groupedtt.MaxDateTime
EXCEPT
(SELECT tr.geom, tr.id, date
FROM traiettoriespaz as tr join geofence_walk as geo on st_contains(geo.geom,
tr.geom))) as finale
```

Front end

Per quanto riguarda il front-end, come accennato nella sezione della progettazione, sono stati considerati due approcci diversi: OpenLayer e QGIS.

Il motivo del mantenimento di due piattaforme front end risiede nel fatto che, mentre quella implementata con OpenLayer consente maggiore libertà nell'inserimento di moduli aggiuntivi e nell'organizzazione dello spazio, quella su QGIS consente di testare nuove funzionalità senza effettuare troppe operazioni.

Inoltre, il mantenimento della piattaforma su QGIS è un valore aggiunto nel caso in cui l'utente amministratore desideri modificare lo stile dei livelli senza mettere mano al codice o voglia

aggiungere funzionalità aggiuntive quali l'applicazione dei numerosi algoritmi di analisi predisposti da QGIS.

OPENLAYER

Le richieste al server web, per il reperimento delle risorse presenti nel database, sono state effettuate tramite richieste AJAX specificando:

- url
- type: Tipo di richiesta
- dataType: tipo di dati attesi dal server
- success: Funzione di callback da eseguire se la richiesta ha esito positivo

```
$.ajax({
    url: url,
    type: "POST",
    dataType: "json", // added data type
    async: false,
```

In caso di esito positivo, i dati geometrici vengono trasformati nell' opportuna geometria e inserite in una sorgente (Source).

```
for (var i = 0; i < coord.length; i += 1) {
    //console.log(coord[i]);

    feature.push(format.readFeature(coord[i]));

    feature[i].getGeometry().transform("EPSG:4326", "EPSG:3857");

    vectorSource.addFeature(feature[i]);
}
```

Il layer viene quindi aggiornato con la nuova sorgente e con lo stile determinato da una combinazione di riempimento, tratto, testo e immagine e, infine, caricato sulla mappa.

```
vectorLayer = new ol.layer.Vector({
```

```

source: vectorSource,

style: customStyleFunction

});

map.addLayer(vectorLayer);

```

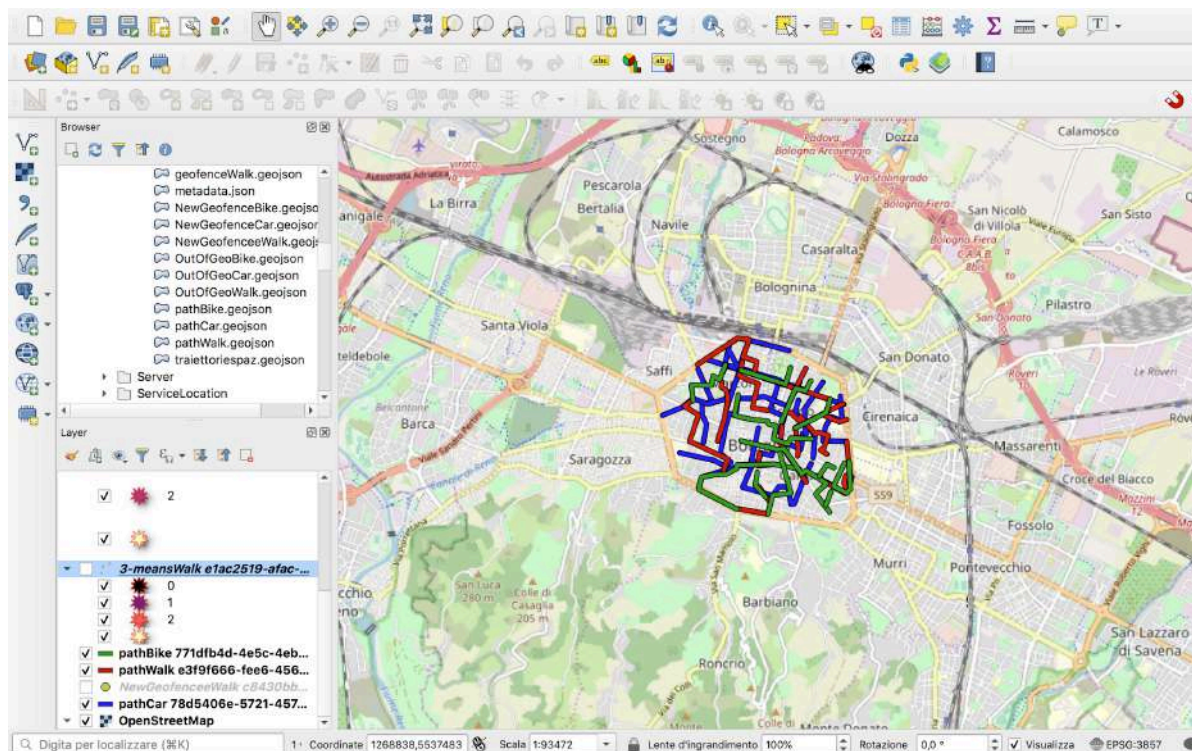
QGIS

Per quanto riguarda QGIS, dopo aver fatto la connessione con il database, è stato sufficiente caricare i layer relativi alle tabelle presenti su PgAdmin.

Dal momento che alcuni dati (es. traiettorie degli utenti per le 3 mobilità) non sono direttamente presenti nel database ma sono il risultato di una manipolazione delle geometrie tramite query, è stato necessario l'utilizzo del Db Manager presente su Qgis.

Le query sono quindi state eseguite e trasformate in layer personalizzati visibili sulla mappa.

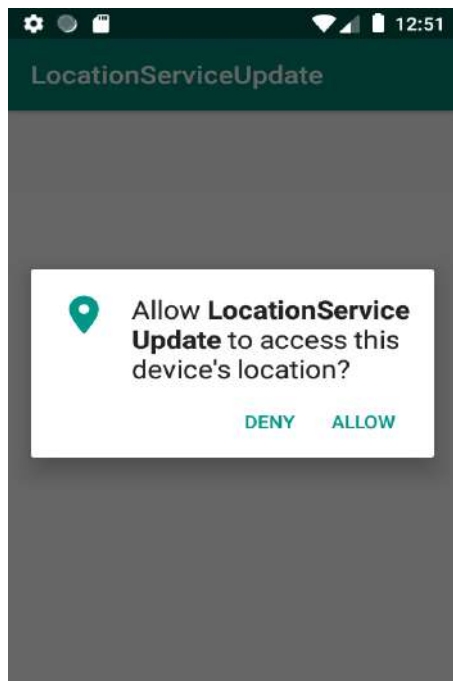
Per quanto riguarda l'utilizzo degli algoritmi DbScan e Kmean, QGIS, al contrario di OpenLayer, offre la possibilità di applicare i classificatori direttamente su una tabella senza dover formulare query.



Risultati

UTENTE GENERICO

Al primo avvio dell'applicazione viene chiesto all'utente di abilitare i permessi di localizzazione. Solo in tal caso l'app sarà in grado di sfruttare il GPS per individuare i dati di posizione

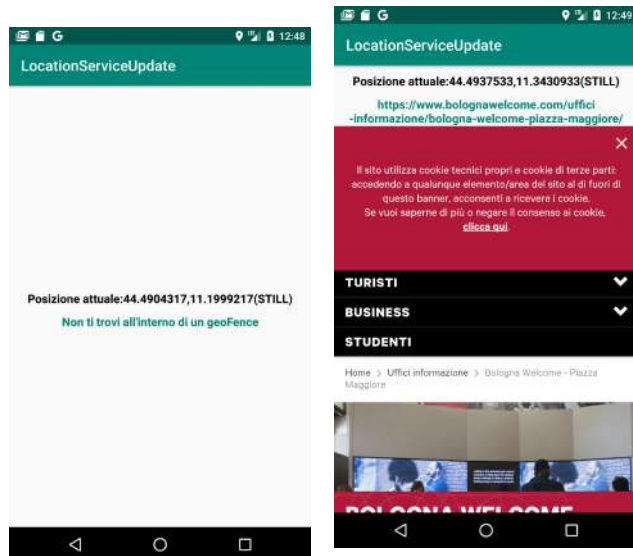


Una volta abilitati i permessi, a seconda della posizione dell'utente, nella parte superiore dello schermo verranno mostrate:

- dati di posizione sotto forma di latitudine e longitudine
- tipo di mobilità riconosciuta

Se l'utente si trova all'interno di un geofence, sullo schermo verrà mostrata una pagina web relativa al luogo in cui si trova (nel caso la mobilità individuata sia "Walk") oppure delle informazioni sui parcheggi o sulla posizione delle rastrelliere (nel caso la mobilità sia "bike" o "car").

Qualora l'utente non si trovi in alcun geofence verrà mostrato un messaggio di testo.



UTENTE AMMINISTRATORE

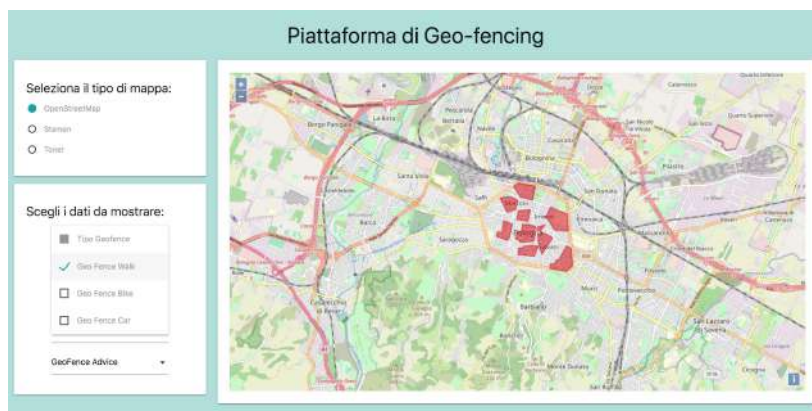
L'interfaccia della piattaforma di geofencing consente di scegliere tra 3 diverse tipologie di mappe sulle quali verranno visualizzati i dati geospaziali.

- OpenLayer: mappa abilitata di default
- Stamen Terrain: mappa del terreno con ombreggiature per le colline e colori naturali della vegetazione
- Stamen Toner: in bianco e nero ad alto contrasto, ideale per mash up dei dati e per l'esplorazione di meandri fluviali e zone costiere



Cliccando sulle drop box a sinistra e abilitando le checkbox, l'utente ha la possibilità di scegliere i dati da visualizzare (anche contemporaneamente) per ciascun tipo di mobilità.

La sezione "geofence" mostra le aree di geofence all'interno della città di Bologna relative a ciascun tipo di mobilità



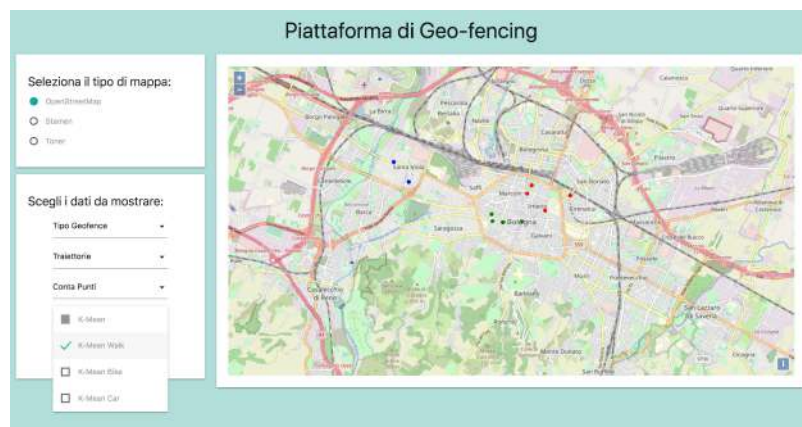
La sezione “traiettorie” mostra i tragitti percorsi dagli utenti



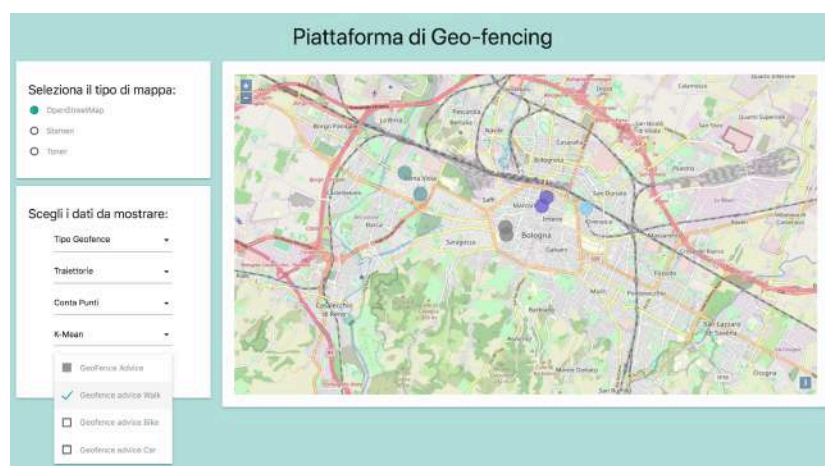
La sezione “Conta punti” mostra le aree di geofence distinte per colore in base al numero di utenti che vi sono entrati. La legenda delle attivazioni è visibile nella parte superiore dello schermo.



La sezione “k-mean” mostra il risultato dell’algoritmo di clustering k-mean, applicato sulle posizioni (punti di coordinate) degli utenti




La sezione “Geofence advice” mostra dei suggerimenti su dove inserire ulteriori geofence in base alle posizioni degli utenti



Conclusioni

Lo sviluppo di queste piattaforma di geofencing è stata una esperienza di sviluppo a tratti complessa, ma certamente molto interessante. Partendo da un concept molto ben definito sul funzionamento del progetto, è risultato naturale impostare l’architettura del software.

Le principali difficoltà sono occorse durante la fase di ricerca e identificazione delle librerie e dei software con cui implementare l’architettura; durante questa fase si è tenuto conto dei requisiti progettuali (ovvero delle indicazioni fornite dal docente), e del know-how dei membri del progetto (anche per una corretta divisione del carico di lavoro).



Le interfacce grafiche sono minimali, in particolar modo quella dell'applicazione mobile, tuttavia, abbiamo preferito concentrarci sulle funzionalità e sul corretto funzionamento della piattaforma piuttosto che sul design.