Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica per il Management

Progettazione e Implementazione di una piattaforma di Geofencing per il Context-Aware Advertising

Relatore:
Chiar.mo Prof.
Marco Di Felice

Presentata da:
Filippo Boiani

Sessione II
Anno Accademico 2015/2016

A chi mi ha messo al mondo,

a chi l'ha reso un posto interessante...

Introduzione

La diffusione di smartphone e dispostivi mobili a cui si è assistito nell'ultima decade ha portato con sè lo sviluppo di nuovi sistemi e tecnologie basate sulla localizzazione. Questi sistemi vengono chiamati *Location Based Systems* (LBS) [1] ed il loro successo è stato reso possibile dai sensori come GPS, antenna WiFi e accelerometro che hanno permesso agli sviluppatori di creare contenuti e servizi basati sulla posizione dell'utente.

Una delle tecnologie su cui si basano i LBS è chiamata *Geofencing* e consiste nella creazione di aree virtuali (dette *geofence*) per delimitare luoghi di interesse (*Point of Interests*) e notificare l'utente quando entra, esce o si trova nelle vicinanze di una delle aree delimitate. Questa caratteristica viene utilizzata soprattutto per realizzare applicazioni che ricordano all'utente di svolgere delle azioni, per esempio un'app di promemoria che ricorda all'utente di comprare il latte quando si trova vicino ad un supermercato.

Dal punto di vista economico, uno degli utilizzi più promettenti è il cosiddetto *Context Aware Advertising*: i possessori di dispositivi mobili che si trovano a camminare nelle vicinanze di un negozio o un centro commerciale possono essere considerati possibili clienti e ricevere delle notifiche con pubblicità o questionari [2].

Esistono altre applicazioni tra cui: il monitoraggio delle flotte di camion, in controllo di animali domestici o il monitoraggio dei comportamenti delle

folle [3, 4]. Oltre alla posizione, le suddette applicazioni hanno bisogno di conoscere altre informazioni contestuali come tipo di attività, ora, velocità, età o sesso, per poter inviare notifiche di interesse per l'utente.

Gli esempi illustrati corrispondono a sistemi detti *pro-attivi* in quanto non è l'utente a richiedere esplicitamente le informazioni ma sono le applicazioni ad inviare delle notifiche in base a suoi spostamenti. Siccome gli spostamenti devono essere tracciati in maniera continua emergono delle problematiche legate alla privacy dell'utente e al consumo di batteria dei dispositivi.

Nel seguente lavoro verrà analizzato nel dettaglio cos'è il *Geofencing*, le sue applicazioni e le tecnologie sulla quale si basa. Verranno presi in considerazione alcuni problemi relativi all'utilizzo delle tecnologie di posizionamento, con particolare attenzione ai consumi di batteria.

Verrà inoltre descritta la progettazione e l'implementazione di una piattaforma client-server composta da un servizio web — sviluppato in stile RESTful — accessibile da un sito web e un'applicazione mobile sviluppata in iOS. Il sito web permette ai possessori di attività commerciali di creare e gestire dei POI nonchè di monitorare gli spostamenti dei possibili clienti. Dopo aver installato l'applicazione mobile sul proprio smartphone, i clienti potranno ricevere notifiche una volta oltrepassati i confini di un *geofence*.

L'applicazione mobile verrà testata e analizzata in termini di prestazioni, consumo di batteria e frequenza di aggiornamenti di posizione con riferimenti a valutazioni simili reperibili in letteratura.

La trattazione è strutturata come segue: all'inizio (capitolo 1) verrà introdotto il concetto di geofencing e le sue applicazioni. Nel capitolo 2 verranno analizzate le tecnologie di posizionamento, le soluzioni al problema di consumo della batteria ed il supporto dei database. Nel capitolo 3 e 4 verrà descritto il sistema che è stato sviluppato e i suoi dettagli implementativi.

Il documento termina con l'analisi del sistema e delle tecnologie di posizionamento (capitolo 5) e si conclude con un breve sommario in cui vengono specificate le difficoltà affrontate e i possibili sviluppi futuri.

Indice

Introduzione **i**

1 Il Geofencing **1**

1.1 Cos'è il Geofencing? 1

1.2 Differenze tra Geofencing e Beacons 2

1.3 Le applicazioni 4

1.3.1 Promemoria 4

1.3.2 Pubblicità basate sulla localizzazione 4

1.3.3 Coupon, sconti e fedeltà della clientela 5

1.3.4 Gestione di grandi eventi ed emergenze 6

1.3.5 Trasporti 7

1.3.6 Sicurezza e droni 7

2 Le Tecnologie del Geofencing **9**

2.1 Tecnologie di Localizzazione 10

2.1.1 *Global Positioning System* 10

2.1.2 Posizionamento Tramite WiFi: *Fingerprinting* 12

2.1.3 Posizionamento Basato sulla Rete Cellulare 13

2.2 Gestione della Batteria 14

2.2.1 Confronto tra la varie tecnologie e problemi 14

2.2.2 Soluzioni proposte 15
2.3 Supporto dei Database 17
2.3.1 Sistemi NoSQL 17
2.3.2 Funzionalità 18
2.3.3 Algoritmi 19

3 Progettazione di un sistema di Geofencing per dispositivi mobili 23

3.1 Applicazione Web 24
3.1.1 *Point of Interest* 24
3.1.2 Utenti 25
3.1.3 Funzionalità 26
3.2 Applicazione Mobile 27
3.2.1 Funzionalità 27

4 Dettagli Implementativi 35

4.1 Database 35
4.1.1 POI circolari 36
4.1.2 POI poligonali 38
4.1.3 Utente 40
4.2 Server 41
4.2.1 Routing 41
4.2.2 Servizio di Ricerca dei POI 42
4.3 Client Web 47
4.4 Client Mobile 49
4.4.1 Core Location Framework 50
4.4.2 Monitoraggio del livello di batteria 53

5 Analisi e Valutazioni **55**

5.1 Valutazioni effettuate in letteratura 56

5.2 Test e Valutazioni dell'Applicazione 57

Conclusioni **61**

Bibliografia **63**

Elenco delle figure

1.1 Illustrazione geofence 3

2.1 Triangolazione GPS 11
2.2 Illustrazione R-Tree 20

3.1 Illustrazione piattaforma realizzata 24
3.2 Interfaccia di login e registrazione 28
3.3 Interfaccia visualizzazione POI 29
3.4 Interfaccia di aggiunta POI 30
3.5 Interfaccia dei percorsi utente 31
3.6 Interfaccia dell'applicazione mobile 32
3.7 Altre interfacce dell'applicazione mobile 33

Elenco delle tabelle

2.1 Sistemi di posizionamento 15

5.1 Risultati test 58

Capitolo 1

Il Geofencing

1.1 Cos'è il Geofencing?

Il *Geofencing* è una delle tecnologie su cui si basano i cosiddetti *Location Based Systems* (LBS) e consiste nella creazione di barriere virtuali intorno a delle aree geografiche e dell'uso di queste per svolgere delle azioni quando un utente entra o esce dai suoi confini [1].

La popolarità crescente degli smartphone ha permesso la diffusione di queste tecnologie grazie soprattutto ai sensori fisici o virtuali montati su questi dispositivi. In particolare accelerometro, bussola e GPS riescono a determinare posizione, velocità e direzione di una persona e rilevare quando questa si trova nelle vicinanze di un POI e oltrepassa i suoi confini virtuali. Ad ogni punto di interesse può essere associata un'azione che si traduce in una notifica inviata all'utente con informazioni su prodotti, coupon, storia dei monumenti o questionari.

I fence possono essere di vari tipi e dimensioni: possono essere grandi come una piccola bottega o come un quartiere fino ad arrivare ad intere città o addirittura stati. Le tipologie variano da *fence* di tipo statico, che si riferi-

scono a delle posizioni geografiche fisse come ristoranti, negozi e monumenti, a *fence* di tipo dinamico [6] che si riferiscono ad entità che possono cambiare posizione nel tempo. Alcuni esempi dell'ultima tipologia sono geofence associati a veicoli o ad individui in movimento come nel caso di applicazioni social che permettono di scoprire gli amici nelle proprie vicinanze.

Anche le forme possono variare: nel caso si debba notificare un insieme di persone ad una distanza massima da un determinato punto, la soluzione migliore sarebbe quella di optare per fence di forma circolare, identificati dalle coordinate del centro e dalla lunghezza del raggio. La forma poligonale può essere usata in situazioni in cui si debba delimitare una fiera di strada, un insieme di vie, delle aree coltivabili o i possibili punti di accesso ad un luogo.

Le architetture dei LBS si compongono generalmente di un server web e uno o più client. Il primo si occupa di gestire i dati relativi ai punti di interesse e immagazzinarli all'interno di un database abilitato alla gestione di oggetti spaziali; Il secondo è di solito un'applicazione installata su un client mobile con lo scopo di rilevare la posizione dell'utente e inviare coordinate e altri dati contestuali al server. Il server, a sua volta, effettua una ricerca all'interno del database e restituisce la lista dei geofences in prossimità della posizione specificata. Per ogni geofence viene inviato un messaggio all'utente sotto forma di SMS o notifica.

1.2 Differenze tra Geofencing e Beacons

I Beacons sono un tipo differente di tecnologia su cui si basano i LBS e consiste nell'utilizzo di piccoli trasmettitori bluetooth che si limitano a mandare messaggi in broadcast a tutti i dispositivi mobili che si trovano nel

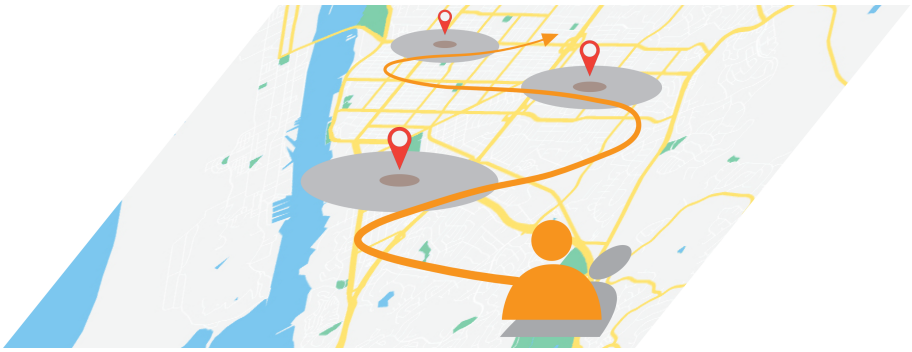


Figura 1.1: Illustrazione di una mappa con punti di interesse delimitati da geofence.

loro raggio di azione.

A differenza del geofencing, questa tecnica non delimita delle aree geografiche tramite coordinate, non è in grado di determinare la posizione e non possiede un raggio d'azione molto ampio. Come ultima caratteristica, i beacons non sono in grado di inviare notifiche con offerte o coupon ma si limitano a spedire alcune informazioni alle applicazioni abilitate alla gestione di questi trasmettitori. Una volta ricevuto il messaggio, le applicazioni possono contattare un server o spedire delle notifiche all'utente.

Le caratteristiche descritte in precedenza, rendono questa tecnologia particolarmente adatta a spazi chiusi e ristretti in cui il GPS non riesce a determinare la posizione, come negozi e centri commerciali. Geofencing e beacons possono essere molto utili se utilizzati in modo complementare all'interno di sistemi di context-aware advertising.

1.3 Le applicazioni

Le applicazioni di questa tecnologia sono innumerevoli e disperante. Si parte dall'advertising e l'invio di coupon per arrivare a sistemi per il monitoraggio di animali domestici e infanti, passando per il check-in/check-out automatico da hotel e strutture turistiche nonchè a guide interattive per musei e punti di interesse all'interno di una città o a sistemi di controllo per detenuti agli arresti domiciliari [1, 7].

Di seguito verrà trattata una serie di possibili applicazioni corredate da esempi di aziende che attualmente utilizzano o hanno intenzione di utilizzare questa tecnologia. Questa trattazione è volontariamente non esaustiva a causa degli innumerevoli campi di applicazione e delle possibilità d'impiego.

1.3.1 Promemoria

Dal 2012, con l'applicazione Remainder (Promemoria) di Apple [8], gli utenti possono creare un promemoria in modo tale che avvisi l'utente quando entra o lascia una determinata zona.

Per fare un esempio, questa funzionalità risulta particolarmente utile nel caso di voglia ricordare a se stessi di compare qualcosa la prossima volta che si passa davanti al supermercato. In altri casi risulta utile quando ci si deve ricordare di chiedere qualcosa ad un conoscente la prossima volta che si va a visitarlo.

1.3.2 Pubblicità basate sulla localizzazione

Uno degli impieghi maggiori del geofencing è sicuramente quello che viene chiamato Local Awareness Advertising [9], cioè pubblicità create ad hoc e spedite ad un target di utenti in base alla loro posizione.

Il colosso dei Social Media, Facebook, ha creato nel 2014 un servizio chiamato *Local Awareness Ad* messo a disposizione dei business per creare delle campagne pubblicitarie con lo scopo di raggiungere tutti gli utenti di Facebook che ti trovino nelle vicinanze dell'attività commerciale. Il gestore dell'attività non deve far altro che indicare la posizione del negozio, specificare un raggio e un target di clientela. In base ai dati introdotti viene calcolata una stima del numero di persone a cui verrà suggerito di visitare il negozio.

Questa soluzione risulta molto efficace dal momento che la maggior parte dei possessori di smartphone ha installato l'applicazione di Facebook e che questo possiede innumerevoli informazioni contestuali sugli utenti permettendo di creare target specifici (informazioni su età e sesso, preferenze, interessi, storia dei posti visitati e del network di conoscenze). Considerando che l'utente tende a ridurre al minimo il numero di applicazioni con il diritto di tracciare la propria posizione, molte aziende si appoggiano al suddetto servizio piuttosto che creare applicazioni apposite.

1.3.3 Coupon, sconti e fedeltà della clientela

Alcune delle metodologie classiche per attirare la clientela verso il proprio negozio sono la creazione di coupons e sconti. Grazie all'impiego di smartphone e geofencing oltre a sfruttare al meglio queste risorse si può aumentare la gamma di servizi offerti e la possibilità di creare (o migliorare, nel caso fosse già presente) un rapporto di fedeltà tra attività commerciale e cliente [2].

Le tecnologie basate sulla localizzazione permettono di notificare qualsiasi possessore di un cellulare che si trovi a camminare o a guidare nelle vicinanze di un'attività commerciale, spedendo dei buoni sconto a scadenza temporale.

Nel caso esaminato in precedenza, Facebook era un esempio di un provider di servizi. Altre aziende hanno optato per la creazione di applicazioni apposite per l'emissione di buoni sconto o hanno deciso di aggiungere funzionalità alle loro applicazioni (come e-commerce) che permettessero di migliorare l'esperienza di acquisto. Per esempio, Walmart ha posizionato dei geofence intorno ad ognuno degli oltre 4500 punti vendita negli Stati Uniti con lo scopo di avvisare i dipendenti quando un cliente sta per entrare nel negozio per ritirare degli articoli [10]. In aggiunta, l'applicazione permette di attivare la modalità di navigazione virtuale all'interno del negozio più vicino una volta oltrepassato il confine definito dal recinto virtuale .

1.3.4 Gestione di grandi eventi ed emergenze

Il *crowdsensing* è la misurazione di un fenomeno risultante dalla combinazione di misurazioni effettuate da dispositivi diversi portati da individui diversi [5, 11]. Queste rilevazioni possono essere utilizzate insieme al geofencing per gestire eventi di grosse dimensioni o situazioni di pericolo pubblico.

Mentre il crowdsensing permette di capire cosa stanno facendo un insieme di persone, il geofencing permette di definire dove lo stanno facendo. Grazie a ciò è possibile indicare percorsi differenti in base allo stato della folla all'entrata di uno stadio [4], monitorare i movimenti di gruppi di persone, individuare comportamenti fuori dall'ordinario e prevenire situazioni di emergenza.

Una volta determinata una situazione critica, in base alla posizione, è possible indicare la via di fuga più veloce o specificarne una alternativa nel caso quella più veloce non fosse percorribile. Tutto ciò mediante notifiche diverse inviate ai dispositivi mobili di ogni persona presenta nella folla.

1.3.5 Trasporti

La delimitazione di aree geografiche virtuali e oggetti risulta incredibilmente utile nell'ambito dei trasporti. Nel caso di aziende di spedizione questa tecnologia viene utilizzata per monitorare l'aderenza ad un percorso prestabilito grazie alla creazione si una regione circolare intorno a furgoni e camion. Queste aree virtuali vengono poste anche intorno ai magazzini per notificare quando un carico sta per arrivare [3, 12].

Nell'abito dei traporti possiamo considerare, a titolo d'esempio, Uber, un'azienda che fornisce servizi di trasporto privato. La sua app si basa su un network che connette passeggeri e autisti. I passeggeri possono richiedere un passaggio e selezionare una posizione in cui farsi prendere.

Uber utilizza il geofencing per molti scopi, uno dei più rilevanti è quello della tutela degli autisti. Infatti gli autisti di Uber non sono a contratto e non hanno un'assicurazione, questo gli impedisce di prendere passeggeri in determinati punti vicino agli aeroporti. Uber aggira il problema creando dei geofences introno a queste zone in modo che gli autisti rispettino le leggi definite dalle autorità aeroportuali [13].

1.3.6 Sicurezza e droni

In generale, nell'aviazione, la sicurezza dei voli dipende dall'addestramento dei piloti, dalle loro esperienze e rispetto delle regole. Al contrario, per quanto riguarda i piccoli aeromobili a pilotaggio remoto, altrimenti detti UAV (Unmanned Air Vehicles), possono essere acquistati da qualsiasi persona senza particolari abilità o addestramento.

Virtualmente, questi aeromobili, fatta particolare attenzione per i droni, posso volare dappertutto senza grosse restrizioni, questo porta con sè svariati

problemi di sicurezza. Basti pensare che un piccolo UAV di un chilogrammo che cade da 50 metri raggiunge il terreno ad una velocità di 30 m/s (approssimativamente 110 Km/h) quanto basta per infortunare o uccidere qualcuno [14].

La maggior parte di questi velivoli possiede un sensore GPS e un computer di bordo. Questo permette di utilizzare il geofencing come tecnologia sulla quale basare meccanismi per evitare ostacoli e per la delimitazione di no fly zones, zone in cui i droni non possono volare.

Le aziende produttrici di droni, come la DJI [15] , mantengono database di no fly zones, soprattutto aeroporti, identificati da geofences tridimensionali, i quali permettono di delimitare aree geografiche e lo spazio aereo sovrastante fino ad una soglia massima. Queste aziende permettono anche la sottomissione di geofences personalizzati, nel caso un individuo non volesse vedere droni volare sopra la propria casa.

Capitolo 2

Le Tecnologie del Geofencing

Attualmente, i sistemi mobili come Android o iPhone non utilizzano solo il GPS come tecnologia di base per la localizzazione. Diversi altri metodi sono stati messi a disposizione, come la geo-localizzazione tramite rete WiFi o tramite le celle radio della rete telefonica cellulare.

Questi sistemi di posizionamento hanno differenti caratteristiche in termini di precisione, consumo di energia, accessibilità, accuratezza e tempo per la prima rilevazione (comunemente detto *time to first fix* o TTFF). Bareth e Küpper in [16] illustrano in dettaglio gli attributi comuni ai vari sistemi:

1. **Accuratezza/precisione**: il possibile errore, espresso in metri, rispetto al posizione reale;

2. **Accessibilità**: facilità di accesso al sistema di localizzazione;

3. **Energia richiesta**: l'ammontare di energia richiesta in media da sistema di localizzazione;

4. **TTFF**: time to first fix, tempo, espresso in secondi, necessario per avere la prima rilevazione della posizione.

Queste proprietà serviranno come metro di paragone per confrontare le diverse tecnologie.

Siccome i dispositivi mobili hanno risorse energetiche limitate, il seguente capitolo trattatterà con particolare attenzione il problema del consumo di batteria e verranno illustrate le soluzioni proposte in letteratura che si pongono l'obiettivo di ottimizzare le risorse energetiche. Infine, verrà analizzato un altro aspetto essenziale dei sistemi basati sulla localizzazione: il supporto fornito dai database ai dati di tipo spaziale.

2.1 Tecnologie di Localizzazione

2.1.1 *Global Positioning System*

Il sistema di posizionamento attualmente più comune è, senza dubbio, il GPS. È stato creato ed è tuttora gestito dal governo degli USA e si compone di tre parti dette *segmenti*. Il *segmento spaziale* è costituto da 31 satelliti che orbitano intorno alla Terra e trasmettono in continuazione segnali radio in broadcast. I sensori GPS integrati in tutti i dispositivi mobili di ultima generazione costituiscono il *segmento utente* e si occupano di riceve i segnali radio trasmessi dai satelliti. Le stazioni di controllo dei satelliti e delle trasmissioni costituiscono infine il *segmento di controllo*.

Il principio di funzionamento si basa su un metodo di posizionamento sferico, che parte dalla misura del tempo impiegato da un segnale radio a percorrere la distanza satellite-ricevitore. Il ricevitore ha bisogno di ricevere almeno 4 segnali provenienti da altrettanti satelliti per calcolare una posizione tridimensionale (latitudine, longitudine e altitudine) e la data corrente. I piani orbitali dei satelliti sono disposti in modo tale che ogni pozione sulla superficie terreste possa ricevere almeno 5 segnali radio differenti [17].

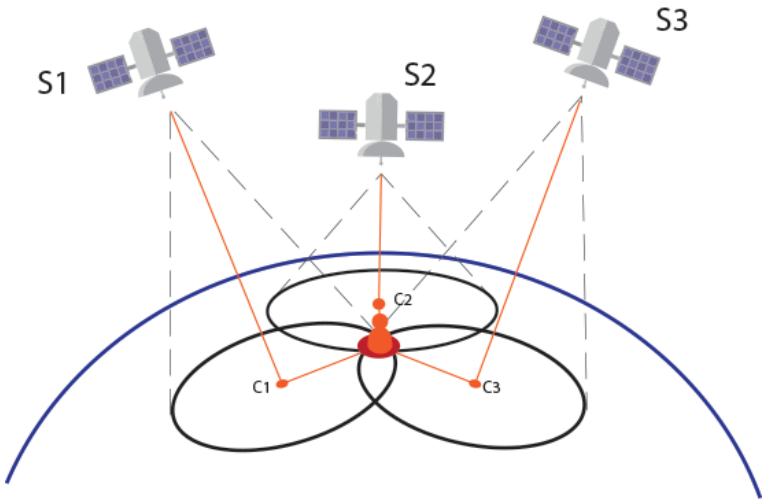


Figura 2.1: Sistema di triangolazione GPS. La linea rossa che parte da S1 e termina in C1 rappresenta la distanza del satellite dalla terra, mente il raggio della circonferenza rappresenta il suo raggio di visibilità. L'utente si trova nel punto/area di intersezione delle tre circonferenze.

Uno degli svantaggi principali — oltre all'accessibilità e al consumo di energia — è il tempo richiesto per la prima localizzazione cioè il TTFF derivante dal tempo necessario per ricavare la lista dei satelliti disponibili che può essere di diversi minuti. Il TTFF è stato ridotto a poche decine di secondi grazie all'introduzione del cosiddetto *Assisted-GPS* che permette allo smartphone di collegarsi, tramite la rete cellulare, ad un Assistance Server fornendogli dati sulla cella a cui il dispositivo è agganciato. L'*Assistance Server* restituisce la lista dei server in vista alla cella specificata supponendo che il terminale mobile veda la stessa lista di satelliti (data la relativa vicinanza alla cella) [1, 16, 17, 18].

Anche con l'ultima generazione di smartphone, capaci di ricevere segnali molto deboli, il GPS non è generalmente accessibile all'interno di edifici o

dei cosiddetti *canyon urbani* come viali notevolmente alberati, vie molto strette o circondate da edifici molto alti. Sebbene i produttori dei chip GPS stiano cerando di ridurre al minimo il consumo di energia, l'utilizzo di questa tecnologia rimane dispendiosa per dispostivi con limitate risorse energetiche. Possiamo dire che l'elevato consumo è il prezzo da pagare per avere una misurazione estremamente precisa e accurata anche in aree rurali e disabitate, in quanto questa tecnologia ha copertura globale.

2.1.2 Posizionamento Tramite WiFi: *Fingerprinting*

Il sistema di posizionamento basato su WiFi più comune consiste nel determinare la posizione del dispositivo mobile rispetto ad uno o più hotspot (a cui non deve essere necessariamente connesso). Per determinare la posizione relativa ci cerca all'interno di un database di posizioni dove ogni entry è formata dall'indirizzo MAC di un access point, usato come identificatore, e una posizione GPS.

Per creare questi database ci sono varie tecniche tra cui il *wardriving* dove veicoli appositi, girando per una città, riempiono il database con posizioni GPS correlate a segnali radio. Quando un dispositivo è collegato ad una rete WiFi utilizza l'indirizzo MAC dell'hotspot per trovare la posizione corrispondente all'interno del database [16].

Per determinare la pozione relativa si può usare il *fingerprinting* che consiste nella misurazione della potenza del segnale ricevuto da diversi hotspot (RSS Received Signal Strenght) e nella successiva trilaterazione di queste informazioni per determinare la posizione e la distanza tra il dispositivo e gli access points.

Qualsiasi sistema o algoritmo si utilizzi si può far affidamento su un consumo inferiore rispetto al GPS e su una buona accuratezza e precisone (nel-

l'ordine dei 100-150 metri) finchè ci si trova in aree urbane con alta densità di access points. Quando ci si trova in aree rurali dove questi sono meno presenti, la localizzazione diventa difficoltosa, se non impossibile [18].

La precisione dipende anche da fattori come il numero di entry all'interno del database delle posizioni e dal grado di aggiornamento degli stessi. Infatti è necessario mantenere costantemente aggiornate le entry del database siccome alcuni hotspot possono essere mobili o disattivabili, si pensi per esempio a smartphone che utilizzano la modalità thetering.

2.1.3 Posizionamento Basato sulla Rete Cellulare

Simile al sistema precedente, il sistema di posizionamento basato su rete cellulare utilizza un identificatore unico della stazione radio base che è composto da: *mobile country code* (MCC), dal *mobile network code* (MNC), l'id della cella radio e l'identificatore dell'area coperta dal segnale (LAI according location area identifier) detta appunto cella radio. Questo id viene correlato ad una posizione GPS [16].

Per calcolare la posizione si analizza la potenza del segnale radio di ogni cella telefonica (in relazione alla rispettiva stazione radio base) collegata con il dispositivo mobile e ne viene determinata la distanza da questa in base alla conoscenza di parametri come l'attenuazione nel mezzo di propagazione.

Il posizionamento tramite rete cellulare è senza dubbio quello più efficiente in termini di consumo energetico, tuttavia risulta anche il meno accurato in quanto si possono avere errori medi nell'ordine dei chilometri. La precisione inoltre è molto variabile, dipende infatti dalla vicinanza delle stazioni radio base in quanto la misurazione è basata sulla rilevazione dei livelli di potenza del segnale tra più stazioni e la successiva triangolazione dei dati.

2.2 Gestione della Batteria

Nei sistemi LBS di seconda generazione basati sul geofencing, viene usato un meccanismo pro-attivo di notifica. Questo significa che una persona non deve occuparsi di cercare i punti di interesse (POI) nelle vicinanze, ma è l'applicazione che notifica l'utente non appena oltrepassa o lascia i confini di un'area geografica virtuale. Per fare ciò è necessario che l'utente abbia installato un'applicazione sul proprio dispositivo che questa sia continuamente attiva in background in modo da tracciare gli spostamenti.

2.2.1 Confronto tra la varie tecnologie e problemi

Avere uno sensore di localizzazione continuamente attivo porta a dei grandi consumi di batteria che dipendono generalmente dal tipo di tecnologia che si sta utilizzando. Nella sezione precedente sono stati analizzati tre diverse tecnologie che possono essere organizzate in ordine gerarchico in base al consumo di batteria e all'accuratezza.

Al primo posto della gerarchia abbiamo il GPS, che rappresenta il sistema di geo-localizzazione più accurato ma, allo stesso tempo, il più dispendioso in termini di consumi energetici a causa del tempo richiesto per la ricerca dei satelliti e della triangolazione della posizione. Come illustrato in [6, 16] questa tecnologia può arrivare ad un grado di precisione di 3-5 metri.

Nell'ultimo livello della gerarchia è presente il sistema di localizzazione basato su celle radio che risulta il più efficiente in termini di consumi energetici ma allo stesso tempo il meno accurato a causa della distanza delle stazioni radio. Nei casi migliori l'accuratezza può arrivare a meno di un chilometro, mente nei casi peggiori diversi chilometri.

In mezzo si colloca il sistema basato su WiFi a cui corrisponde un consumo di energia intermedio e un grado di accuratezza che va dai 50 ai 200 metri [6, 16]. Nella tabella sottostante (tab. 2.1) sono riassunte le principali tecnologie utilizzate.

Una delle sfide più impegnative del geofencing — e più in generale per le applicazioni basate sulla localizzazione — è quella di diminuire il consumo di energia continuando a garantire un grado di accuratezza soddisfacente.

Tecnologia	**Accuratezza**	**Descrizione**
Cell-ID	500-3500 m	Il cell-id è un numero unico associato ad una stazione radio base. Ogni id è associato ad una posizione.
WiFi	50-200 m	I produttori di smartphone mantengono database con indirizzo MAC dell'hotspot e la sua posizione.
GPS	3-50 m	Può essere autonomo o assistito dalla rete cellulare (A-GPS). Nel secondo caso il TTFF si riduce sensibilmente.

Tabella 2.1: Riassunto dei sistemi di localizzazione più comuni in ordine crescente di accuratezza.

2.2.2 Soluzioni proposte

Sono state condotte numerose ricerche in ambito accademico per creare meccanismi che permettessero di localizzare un dispositivo e determinare in modo efficiente dal punto di vista energetico quando questo si trovi all'interno di un fence. Una soluzione proposta da Küpper in [16] consiste nell'utilizzo

delle varie tecnologie illustrate in maniera gerarchica: inizialmente il dispositivo determina se l'area target A_t è all'interno dell'area circolare A_c con centro nella stazione radio più vicina e raggio uguale all'accuratezza. Nel caso A_t fosse contenuta in A_c, viene attivato il successivo sistema in ordine di accuratezza, il WiFi, che determina a sua volta un'area A_w. Solo nel caso in cui A_t fosse contenuto in A_w si attiva il sensore GPS.

Nakagawa e altri [19] utilizzano l'accelerometro per determinare se l'utente è in movimento o fermo. Una volta determinato lo stato di moto, viene calcolata la distanza dal geofence più vicino e la velocità di approccio al target che consiste nella reale velocità dell'utente corretta in base alla direzione. Queste misurazioni servono per determinare quando deve avvenire la successiva localizzazione GPS.

Esistono numerosi studi che si concentrano sulla distanza dal fence più vicino. Garzon [2] utilizza un sistema dove è il server ad indicare al dispositivo mobile la strategia di localizzazione da adottare. Quando il dispositivo si trova lontano dal target si attiva la modalità *safety zone* definita da un raggio nel quale non sono presenti POI e non si ha necessità di avere aggiornamenti della posizione. Quando il dispositivo lascia la safety zone spedisce al server un aggiornamento e riceve da questo una nuova strategia detta *periodic update* nel caso si trovasse nelle vicinanze di un target. Cardone e altri[4] utilizzano un semplice algoritmo che varia l'intervallo di aggiornamento in base alla distanza: nel caso il fence si trovasse ad oltre 750 metri di distanza, l'intervallo viene impostato a 120 secondi; 20 secondi nel caso fosse tra 750-250 metri e 10 secondi sotto ai 250 metri. Questi valori sono determinati partendo dall'assunzione che una persona non corra oltre i 20 km/h.

Siccome il GPS non è affidabile all'interno di edifici, Nakamura e altri [7] hanno proposto un meccanismo che fa uso di un sensore termico per

determinare la differenza di temperatura tra ambiente interno ed esterno all'edificio. Una volta determinata la stagione e la differenza di temperatura, si decide se attivare o disattivare il sensore GPS.

I sistemi proposti utilizzato tutti le tecnologie di localizzazione in modo differente spesso aiutate da altri sensori come giroscopio, accelerometro e termometro, ma una cosa che hanno in comune quasi tutte è l'utilizzo di intervalli di tempo variabile e la diminuzione del numero di rilevazioni. Si può affermare che il consumo di batteria dipende soprattutto dal tipo di accuratezza che si vuole ottenere, la quale dipende a sua volta dalla dimensione dei geofences. In aree urbane ad alta densità di hotspots, viene generalmente utilizzano il WiFi come sistema di posizionamento, che permette di avere un'accuratezza di circa 100 metri perfetta per rilevare fence con diametro superiore ai 150 metri.

2.3 Supporto dei Database

2.3.1 Sistemi NoSQL

Con l'avvento del Web 2.0 l'ammontare dei dati scambiati è aumentato in maniera esponenziale facendo nascere il concetto di Big Data. La necessità di gestire questa enorme mole di dati ha spinto a creare database basati su sistemi distribuiti. Ovviamente, i sitemi relazioni tradizionali (RDBMs) possono essere scalati, tuttavia i costi sono molto elevati e le performance non sono sempre le migliori, di conseguenza sono nati sistemi NoSQL con lo scopo di sorpassare questi problemi.

Nel tempo, la componente spaziale è divenuta sempre più importante grazie alla diffusione di applicazioni basate sulla localizzazione favorendo le basi di dati che erano nativamente ottimizzate per archiviare e cercare dati corre-

lati ad oggetti nello spazio come punti, linee, cerchi e poligoni. Alcuni esempi di questi database sono: CouchDB, MongoDB, BigTable di Google e Neo4j. Altri sistemi (sia relazionali che non) si sono adatti creando librerie che sopportassero oggetti e query spaziali, come PostGIS nel caso di PostgreSQL [20].

2.3.2 Funzionalità

È nata un'organizzazione internazionale chiamata Open Geospatial Consortium con lo scopo di definire specifiche tecniche e standard per i servizi geospaziali e di localizzazione.

Secondo lo standard OGC, la classe `Geometry` dovrebbe avere diverse funzioni che possono essere divise in tre categorie:

funzioni topologiche

funzioni di analisi

funzioni su insiemi

Nella presente trattazione ci concentreremo su MongoDB, un database non relazionale opensource e document-oriented sviluppato dalla compagnia 10gen [21].

Dal punto di vista delle funzionalità MongoDB offre supporto per le seguenti query topologiche:

1. **Inclusione**: è possibile cercare oggetti contenuti interamente all'interno di una forma specificata. La funzione di inclusione utilizza il comando `$geoWithin`;

2. **Intersezione**: permette di cercare oggetti che siano intersecati con una qualsiasi delle forme geometriche permesse da MongoDb attraverso l'operatore `$geoIntersect`;

3. **Prossimità**: con l'operatore `$near` si possono cercare oggetti vicini al altri oggetti.

2.3.3 Algoritmi

Per ottimizzare la ricerca di oggetti come punti, linee e poligoni MongoDB, come gli altri database spaziali, usa degli indici detti *indici geospaziali*. In particolare usa la seguente lista di indici:

1. `2dsphere`: supporta queries relative ad oggetti collocate su superfici sferiche, come quella della terra;

2. `2d`: è relativo a dati conservati come punti su un piano euclideo;

3. `geoHaystack`: è un indice speciale che permette di avere ottimizzazioni nel caso di ricerche su piccole aree in cui si può utilizzare la geometria piana.

. Tutti gli indici di MongoDB sono implementati utilizzando una struttura dati detta B-Tree. Molti altri database utilizzano una struttura più complessa, detta R-Tree, basata su un B-Tree e ottimizzata per l'indicizzazione di oggetti multidimensionali come punti, cerchi e poligoni. La struttura dati è composta da un insieme di *Minimum Bounding Rectangles* (MBR) organizzati in ordine gerarchico (figura 2.2).

In ogni nodo sono contenuti un numero variabile di entry, in ogni entry che non appartenga ad un nodo foglia è contenta un'entità che identifica

l'MBR che chiameremo MBRi e una reference al nodo figlio nel quale sono presenti tutti i MBR contenuti nell'MBRi.

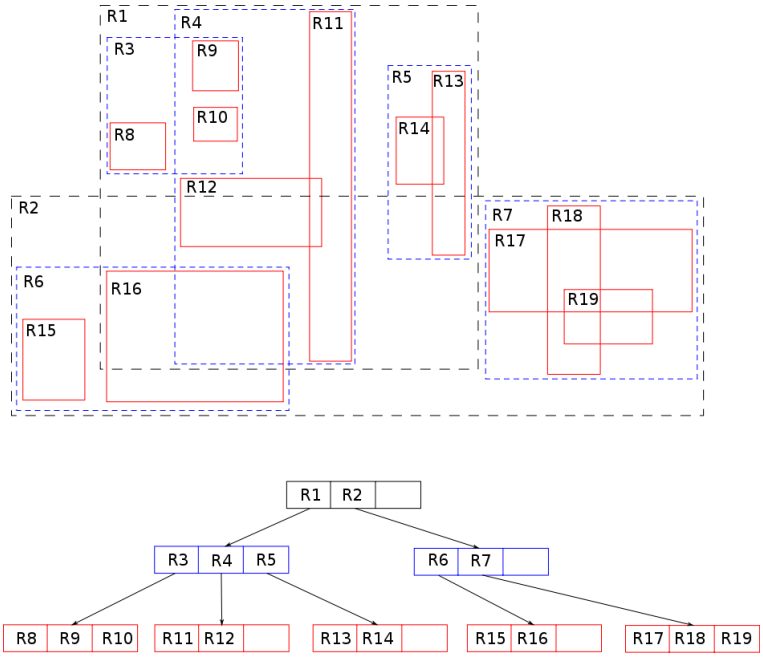


Figura 2.2: Sopra viene riportata la struttura di un R-Tree e l'organizzazione in gruppi di MBR.

In poche parole, questa struttura dati raggruppa in modo bilanciato oggetti vicini e li rappresenta mediante rettangoli. Permette una ricerca efficiente utilizzando i bounding boxes per decidere se cercare o meno all'interno di un sotto-albero. Infatti, se un bounding box non è contenuto all'interno di un altro, si può scartare tutto il sotto-albero radicato nel secondo bounding box.

Una volta identificato il MBR basta verificare se l'oggetto che si sta usando come riferimento, per esempio un punto di coordinate (x,y), è contenuto all'interno del poligono (o qualsiasi altra forma) identificata dal MBR.

Per fare questa verifica si può usare un algoritmo a scelta tra *ray-casting* e *winding number*. Il primo calcola il numero di volte n in cui il raggio con vertice nel punto, con qualsiasi direzione, interseca i lati del poligono. Se n è un numero dispari il punto è contenuto nella figura, in caso contrario è fuori.

Il secondo algoritmo calcola l'*indice di avvolgimento* del punto rispetto al poligono e, se corrisponde a 0 significa che il punto risiede all'interno del poligono. Una tecnica per determinare l'indice è quella di calcolare il numero di angoli sottesi da ogni lato della figura.

Entrambi gli algoritmi devono comparare il punto con ogni vertice del poligono, di conseguenza il costo computazionale della verifica dipende dal numero di vertici della figura. Il costo computazionale della ricerca su una struttura dati di tipo R-Tree è dell'ordine di $\mathcal{O}(\log_M(p)) + \mathcal{O}(v)$ dove M è il numero massimo di figli che un nodo può avere, p è il numero di fence poligonali e v è il numero di vertici di un fence poligonale [23].

Capitolo 3

Progettazione di un sistema di Geofencing per dispositivi mobili

Per investigare le reali potenzialità e criticità del geofencing è stato progettato un sistema composto da un'applicazione web e un client mobile basato su questa tecnologia.

Lo schema della piattaforma e l'architettura del server è mostrata nella figura 3.1. Come si può vedere, il sistema si compone di un server connesso ad un database in MongoDB e due client: uno web e uno mobile. Il client web permette agli utenti di gestire e aggiungere i punti di interesse, mentre quello mobile ha il compito principale di tracciare la posizione degli utenti e chiamare il servizio di ricerca dei POI messo a disposizione dal server.

Di seguito faremo un'analisi dei requisiti concentrandoci in particolar modo sulle funzionalità offerte dalla piattaforma.

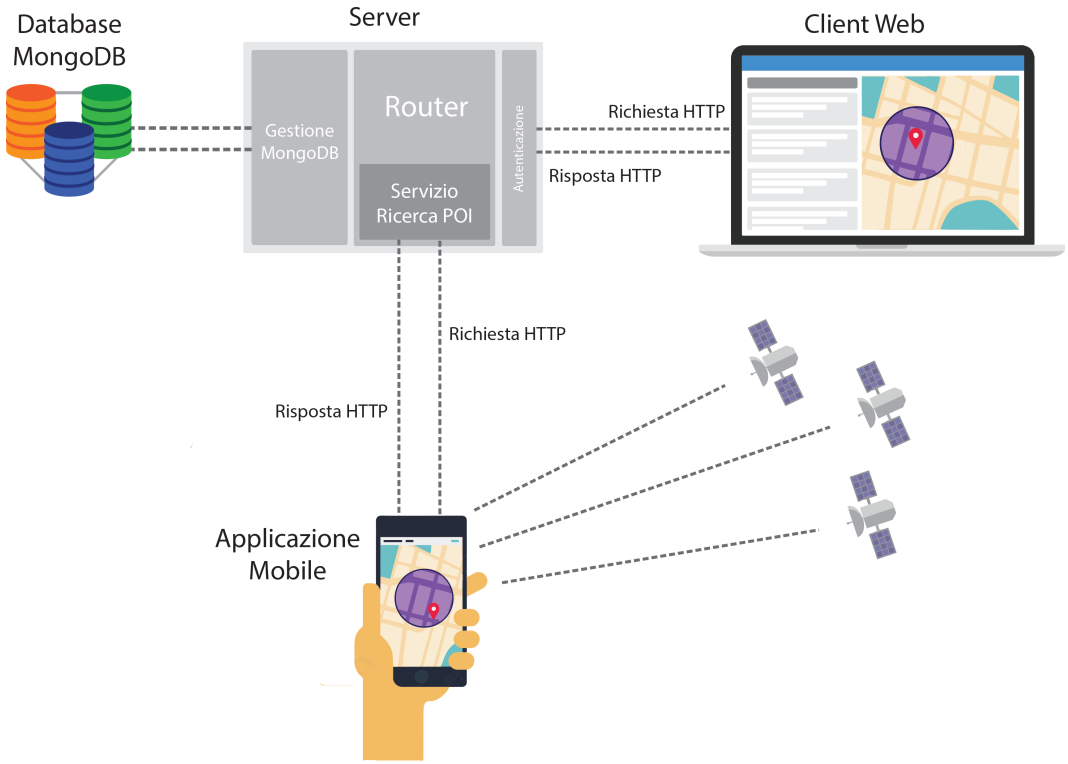


Figura 3.1: Piattaforma client-server. Il server comunica con client web e applicazione mobile tramite richieste HTTP. I POI e le informazioni degli utenti sono salvate all'interno di un database non relazionale (MongoDB).

3.1 Applicazione Web

3.1.1 *Point of Interest*

L'applicazione si basa sul concetto di *Point of Interest* (POI) [24], un luogo di interesse, come un monumento o un edificio, a cui sono collegate delle informazioni. Per rappresentare un POI si utilizza un geofence che può avere sia forma circolare che poligonale; nel primo caso è necessario specificare il centro e il raggio mentre nel secondo è necessario indicare i vari punti che costituiscono i vertici della figura.

Oltre ai dati relativi alla posizione è necessario specificare altre informazioni contestuali in particolare:

1. **Nome**: Il nome del punto di interesse;
2. **Creatore**: Identificatore dell'utente che l'ha creato, nel nostro caso la mail;
3. **Indirizzo**: informazioni relative alla via, all'edifico o al quartiere in cui è presente il poi;
4. **Lista di attività**: le attività corrispondono al tipo di mobilità dell'utente, ad ogni attività corrisponde un messaggio diverso;
5. **Intervallo di validità**: tempo di validità del punto di interesse.

Le informazioni contestuali servono per creare un target e scremare i vari POI per fornire delle informazioni che possano essere rilevanti per l'utente. Per esempio, nel caso del tipo di mobilità, un utente che sta correndo potrebbe essere interessato a pubblicità relative a prodotti per il fitness mentre un utente che sta guidando potrebbe essere interessato a pezzi di ricambio o alla posizione del parcheggio più vicino. Le informazioni contestuali sul tempo sono utili soprattutto nel caso di esercizi commerciali i quali non hanno interesse a offrire coupon o inviti ad entrare nel proprio negozio fuori dagli orari di apertura.

3.1.2 Utenti

L'applicazione prevede tre tipi di utenti: *admin*, *customer* e *business*. Ogni utente è identificato da:

1. **Id**;

2. **Nome**;
3. **Cognome**;
4. **Password**;
5. **Tipologia di utente**;
6. **Sesso**;
7. **Storia**

La storia dell'utente mantiene una lista degli aggiornamenti di posizione ottenuti tramite l'applicazione mobile.

3.1.3 Funzionalità

Le funzionalità offerte dipendono dalla tipologia di utente, in particolare un utente di tipo business può:

1. Effettuare la registrazione (figura 3.2 a);
2. Effettuare il login/logout (figura 3.2 b);
3. Visualizzare la lista dei propri punti di interesse (figura 3.3);
4. Aggiungere un punto di interesse (figura 3.4);
5. Cercare tra i propri punti di interesse (figura 3.3);

In aggiunta un utente admin può:

1. Visualizzare la lista di tutti i POI creati all'interno dell'applicazione (figura 3.3);

2. Visualizzare gli spostamenti degli utenti registrati all'applicazione (figura 3.5);

Nelle figure 3.2 e 3.5 possiamo vedere la lista delle interfacce.

3.2 Applicazione Mobile

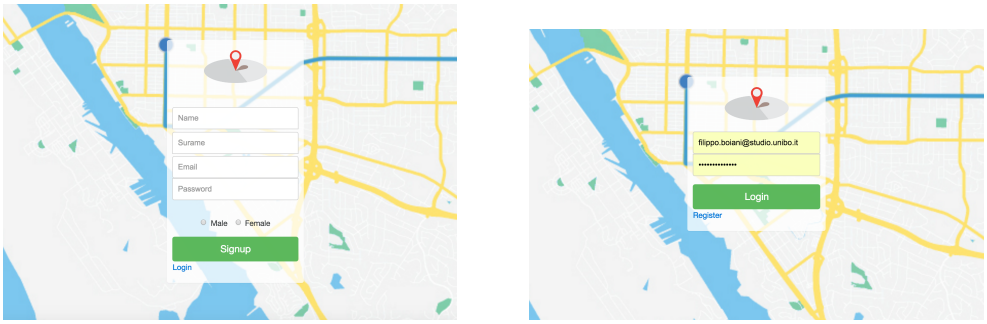
Oltre al programma web, è stata progettata un'applicazione mobile che permette a tutti gli utenti (admin, business, customer) di indicare la propria mail e di tacciare la propria posizione in background. Le coordinate vengono inviate, insieme ai dati sul tipo di mobilità, ad un servizio che restituisce la lista dei POI nelle vicinanze dell'utente.

3.2.1 Funzionalità

Le funzionalità dell'applicazione consistono nel:

1. Tracciare la posizione dell'utente (figura 3.6 a);
2. Inviare le coordinate al servizio REST (figura 3.6 b);
3. Ricevere la lista dei POI (figura 3.6 b);
4. Visualizzare il geofence circolare in cui l'utente si trova (figura 3.7 a);
5. Notificare l'utente dei punti di interesse nelle vicinanze (figura 3.7 b);
6. Mostrare la lista di attività svolte all'interno dell'applicazione (figura 3.6 b).

Nelle figure 3.6 e 3.7 vengono mostrate le funzionalità offerte dall'app.



(a) (b)

Figura 3.2: Nella figura (a) è presente il form per la registrazione dell'utente. Per registrarsi occorre specificare il proprio nome, cognome, indirizzo mail, sesso e password. Nella figura (b) è presente il form per il login: l'utente viene identificato dal suo indirizzo mail.

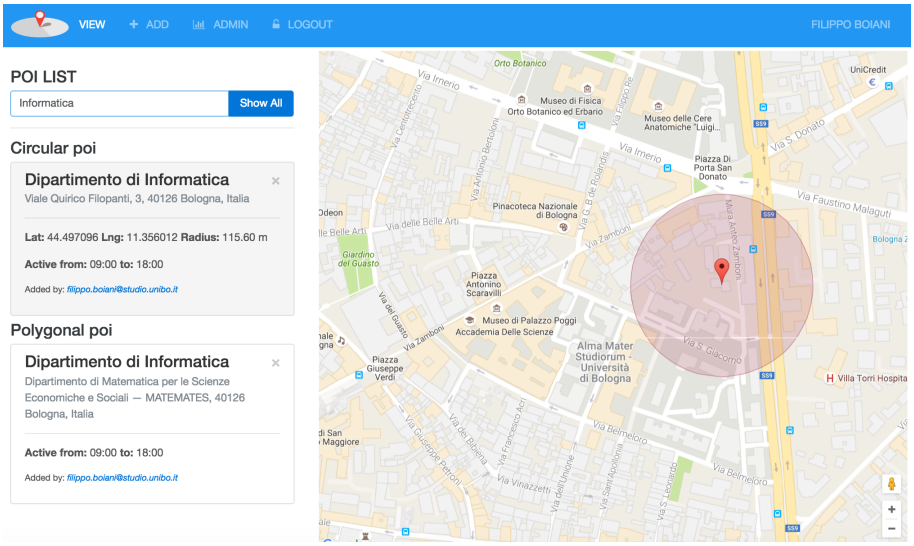


Figura 3.3: Nella figura viene illustrata la pagina di visualizzazione dei POI aggiunti da un utente. La pagina divide i punti di interesse in base alla forma del geofence, in particolare forme circolari e poligonali. I POI possono essere cercati in base al proprio nome ed essere visualizzati tutti sulla mappa per avere una vista d'insieme. L'utente admin è in grado di vedere non solo i propri POI ma anche quelli sottomessi da altri utenti.

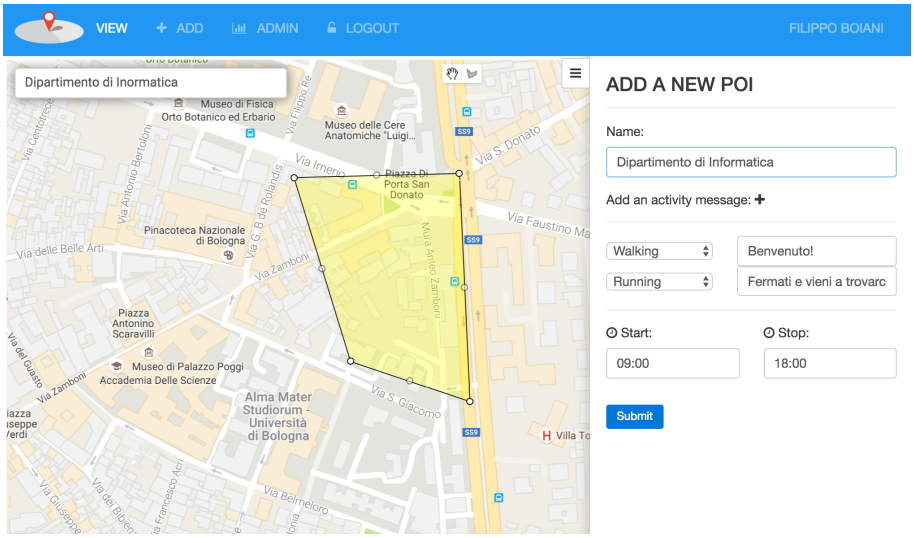


Figura 3.4: L'interfaccia web che permette ad un utente di aggiungere un poi si compone di: una barra di ricerca che permette di specificare un luogo; una mappa di Google che permette la creazione di forme poligonali e cerchi; un form laterale che è possibile nascondere o mostrare a piacimento. Nel form devono essere inseriti i dati relativi al POI, come il nome, e alcuni dettagli contestuali come gli orari di attivazione, i tipi di mobilità supportati e i corrispondenti messaggi.

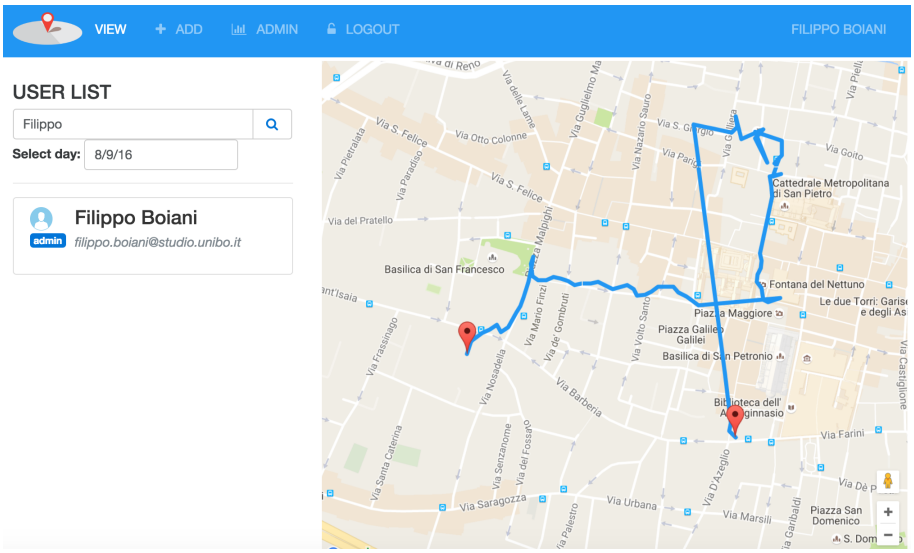
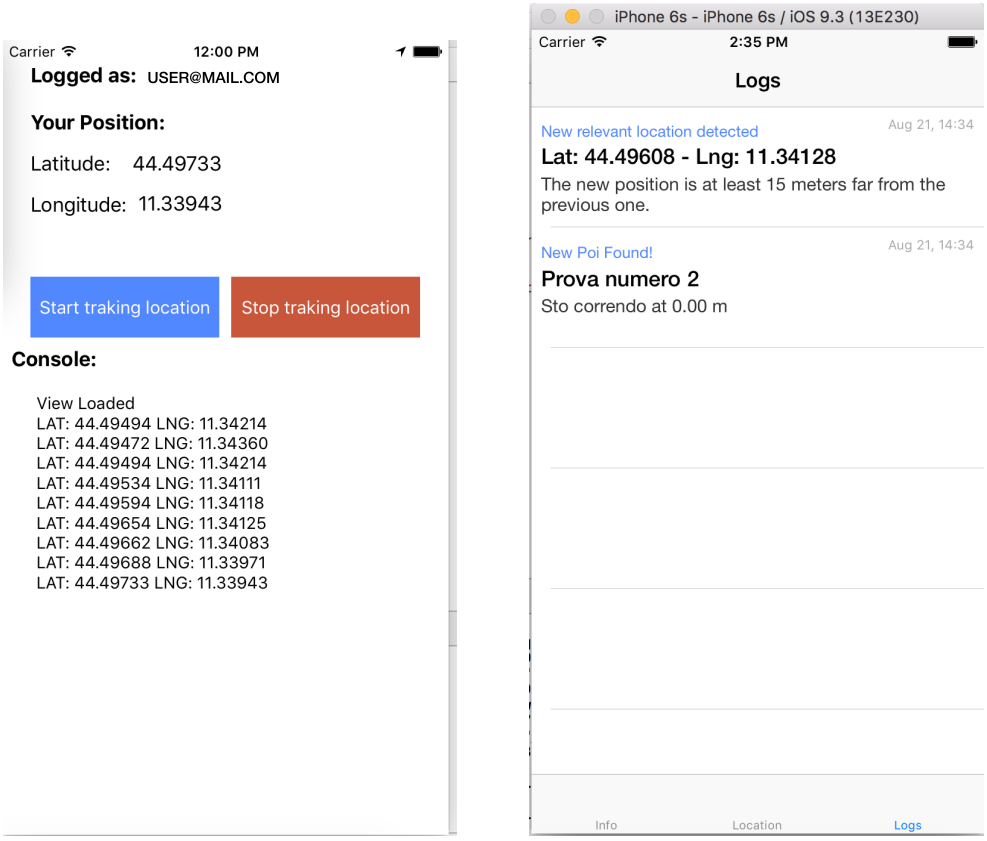
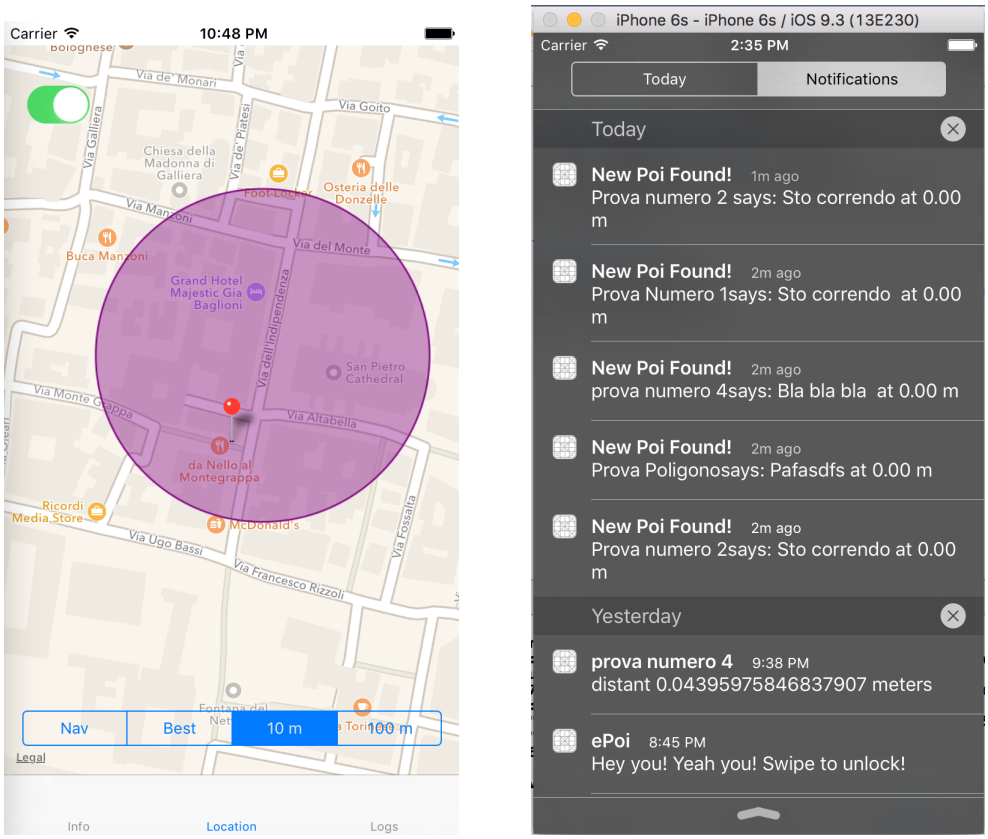


Figura 3.5: La pagina mostra sulla sinistra la lista degli utenti registrati all'applicazione, i quali possono essere cercati grazie alla barra di ricerca. Sulla destra, all'interno della mappa, è mostrato il percorso effettuato dall'utente selezionato nella data specificata. Solo gli utenti admin possono accedere a questa pagina.



(a) (b)

Figura 3.6: Nella figura (a) viene illustrata la prima tab dell'applicazione che si compone di una view con alcune *label* indicanti la posizione dell'utente, il suo id e lo storico delle posizioni. Sono presenti inoltre due bottoni che permettono di attivare/disattivare il sensore GPS. Nella figura (b) è presente la terza tab che corrisponde ad una TableView con la lista delle azioni che avvengono all'interno dell'applicazione. Le azioni sono: l'aggiornamento di posizione, i POI nelle vicinanze dell'utente e le richieste HTTP effettuate. Ogni azione viene salvata in un database interno al dispositivo mobile insieme alle informazioni relative e alla data.



(a) (b)

Figura 3.7: Nella figura (a) si può notare la seconda tab dell'applicazione in cui è presente una MapView. Nella mappa vengono mostrate le posizioni dell'utente e l'ultimo geofence individuato. Oltre a ciò è possibile attivare e disattivare il servizio di localizzazione e modificare l'accuratezza delle rilevazioni GPS che può variare dai 3 ai 100 metri. La figura (b) mostra il centro notifiche del sistema operativo iOS e la lista delle ultime notifiche ricevute dal servizio di ricerca. Quando il servizio ritorna un POI, l'applicazione crea ed invia una notifica che può essere visualizzata anche con lo smartphone in standby.

343. Progettazione di un sistema di Geofencing per dispositivi mobili

Capitolo 4

Dettagli Implementativi

L'applicazione web è stata sviluppata in javascript sia lato client che lato server. In particolare il client è stato implementato utilizzando HTML e AngularJS mentre il server in NodeJS. HTTP e JSON sono stati utilizzati rispettivamente come protocollo di interscambio dati e formattazione dei messaggi. Il client mobile è stato sviluppato in Swift 2.0.

Come già accennato nei capitoli precedenti, per la persistenza si è scelto di utilizzare MongoDB per tre motivi principali:

1. Supporto per i dati spaziali;
2. Ottima integrazione con NodeJS;
3. I documenti sono in formato JSON, lo stesso utilizzato per lo scambio dei dati.

4.1 Database

Il formato JSON permette di rappresentare i dati in maniera più flessibile rispetto ai DBMS relazionali come MySQL a scapito delle relazioni tra entità

che sono di interesse secondario per lo scopo di questa trattazione.

L'integrazione tra Node.js e MongoDB è stata implementata utilizzando Mongoose, una libreria che permette di realizzare un mapping detto Object Data Mapping (ODM) e fornisce un'interfaccia semplificata per la creazione di queries complesse. L'ODM consiste nella traduzione dei documenti all'interno del database in oggetti Javascript in modo che siano immediatamente utilizzabili all'interno dell'applicazione Node.js.

4.1.1 POI circolari

Di seguito viene mostrato il documento JSON rappresentante il poi circolare:

```
POI circolare

{
  "name": "Dipartimento di Informatica",
  "activities": [
    {
        "name": "walking",
        "message": "Benvenuto ad Informatica."
    },
    {
        "name": "driving",
        "message": "Messaggio diretto a chi guida."
    }],
  "sender": "john.doe@mail.com",
  "address": "Viale Quirico Filopanti, 3, 40126 Bologna,
     Italia",
  "location": {
    "type": "Point",
    "coordinates": [11.3459035, 44.4939358]
```

```
19    },
20    "radius": 150,
21    "interval": {
22      "from": 2016-07-14T09:30:00Z,
23      "to": 2016-07-29T23:30:00Z
24    }
25  }
```

Come si può notare, le attività sono rappresentate da un array di oggetti javascript composti da un nome ed un messaggio relativo. Il nome dell'attivtà può essere uno tra i seguenti:

1. Still
2. Walking
3. Running
4. Car
5. Train

Queste attività corrispondono ad alcune di quelle riconosciute dalla classe DetectedActivity all'interno del pacchetto Google Play Services messo a disposizione per Android.

Il campo *sender* corrisponde all'indirizzo mail dell'utente che ha creato il POI mentre l' *address* corrisponde ad una stringa restituita dal servizio di Geocoding offerto da Google. La stringa corrisponde all'indirizzo ricavato dalle coordinate contenute nel campo *location*.

Il campo *location* è conforme al formato standard GeoJSON utilizzato per rappresentare dati di tipo spaziale. L'oggetto *location* è composta da un

campo *type*, che indica il tipo di oggetto GeoJSON, e da un campo *coordinates*. Per convenzione le coordinate sono rappresentate da un array contenente prima la longitudine poi la latitudine.

Nel caso di POI circolari il campo *location* rappresenta il cerchio della circonferenza, mentre il campo *radius* esprime la lunghezza del raggio in metri.

La proprietà *interval* contiene i campi *from* e *to* che esprimono rispettivamente la data di attivazione del geofence e quella di disattivazione. Anche se l'intervallo viene espresso usando l'oggetto *Date* l'unica parte della data utilizzata è quella corrispondente alle ore. La parte non utilizzata rappresenta la data di sottomissione del POI.

4.1.2 POI poligonali

I poi poligonali hanno la seguente forma:

```
1  POI poligonale
2
3  {
4    "name": "Dipartimento di Informatica Poligonale",
5    "activities": [
6      {
7          "name": "walking",
8          "message": "Benvenuto ad Informatica."
9      },
10     {
11         "name": "running",
12         "message": "Messaggio diretto a chi corre."
13     }],
14   "sender": "john.doe@mail.com",
15   "address": "Viale Quirico Filopanti, 3, 40126 Bologna, Italia",
```

```
16   "location": {
17     "type": "Polygon",
18     "coordinates": [
19       [
20           [11.356660, 44.496308],
21           [11.356569, 44.498222],
22           [11.354874, 44.498313],
23           [11.354831, 44.497081],
24           [11.356660, 44.496308]
25       ]
26     ]
27   },
28   "radius": 0,
29   "interval": {
30     "from": 2016-07-14T09:30:00Z,
31     "to": 2016-07-29T23:30:00Z
32   }
33 }
```

Il documento JSON è essenzialmente identico a quello descritto in precedenza, di conseguenza entrambi i documenti vengono salvati all'interno della stessa collezione.

L'unica differenza tra POI circolari e poligonali consiste nell'oggetto GeoJSON. Nel caso dei punti di interesse delimitati da geofence circolari, il campo `location.type` è `"Point"` e le coordinate rappresentano il centro della circonferenza. Nel secondo caso invece si ha `"location.type": "Polygon"` e il campo `coordinates` contiene la lista dei vertici del poligono.

Come si può notare nelle righe 20 e 24, la prima coordinata è identica all'ultima, questo fa si che la figura del poligono sia rappresentata da una linea spezzata chiusa. Siccome il poligono non deve definire un raggio, il campo `radius` ha valore 0.

In generale, le query geospaziali richiedono un indice geospaziale per migliorare le prestazioni della ricerca, di conseguenza è necessario specificare quale campo dovrà essere considerato come indice. Per fare ciò si utilizza il seguente comando:

```
db.pois.createIndex( { location : "2dsphere" } )
```

L'indice utilizzato sarà il campo `location` e sarà del tipo `2dsphere` il quale supporta queries che calcolano geometrie poste su superfici sferiche. Questo indice è perfetto per i poligoni e la circonferenze disegnati per delimitare aree geografiche sulla superficie terrestre.

4.1.3 Utente

La collection *users* contiene una lista di documenti nella seguente forma:

```
User

{
  _id: "john.doe@mail.com",
  name: "John",
  lastname: "Doe",
  pass: "password",
  type: "admin",
  sex: "male",
  history: [
    HistoryItem,
    HistoryItem,
    HistoryItem,
    ...
  ]
}

HistoryItem
```

```
19
20 {
21     "coordinates": {
22       "lat": 44.567893,
23       "lng": 11.293300
24     },
25     "mobility": "walking",
26     "time": "2016-08-10T09:43:42.913Z"
27
28 }
```

I campi sono gli stessi elencati durante la progettazione, in particolare `_id` è l'indirizzo email dell'utente e `history` è la storia rappresentata da una lista di `HistoryItem`.

L'oggetto `HistoryItem` contiene le coordinate relative alla posizione dell'utente, il tipo di mobilità (tra quelli possibili) e la data relativa all'aggiornamento di posizione.

4.2 Server

Il server è stato realizzato seguendo i principi architetturali definiti in REST (REpresentational State Transfer). Questo è stato possibile grazie all'aiuto di Express.js [25], un framework per Node.js con supporto efficiente per routing e middleware.

4.2.1 Routing

Il server espone diverse api che permettono di implementare tutte le funzionalità messe a disposizione dal client web. Le funzionalità si traducono

essenzialmente in azioni su risorse dove i metodi HTTP rappresentano le azioni e gli URI identificano le risorse.

Di seguito vengono elencate le azioni supportate dal server.

Aggiunta di un POI:

```
1 PUT      http://www.host.domain/api/poi
```

Restituire la lista di tutti i POI:

```
1 GET      http://www.host.domain/api/pois
```

Restituisce la lista di tutti gli utenti:

```
1 GET      http://www.host.domain/api/users
```

Restituisce i dati dell'utente autenticato:

```
1 GET      http://www.host.domain/api/user
```

Autenticazione:

```
1 POST     http://www.host.domain/api/authenticate
```

Registrazione:

```
1 POST     http://www.host.domain/api/signup
```

Ogni richiesta HTTP, per essere soddisfatta, deve avere nell'header un token con i dati dell'utente che permettano di identificarlo ed autenticarlo. L'unica eccezione è la richiesta che viene fatta al servizio di ricerca dei POI, che non necessita di un token, ma deve solo specificare un identificativo che permetta di riconoscere l'utente. L'autenticazione in questo caso non è necessaria in quanto non serve essere registrati alla piattaforma per utilizzare l'applicazione.

4.2.2 Servizio di Ricerca dei POI

Il server mette a disposizione un servizio per la ricerca dei POI nelle vicinanze dell'utente. Questo servizio viene utilizzato in particolare dall'ap-

plicazione mobile, ma può essere utilizzato da qualsiasi sito web o client in quanto l'api non necessita autenticazione.

Per richiamare il servizio è sufficiente fare una richiesta HTTP GET al seguente URI:

```
GET http://www.host.dom/api/poi/customer/customerMail/
    mobilityType/latitude/longitude
```

All'interno dell'URI deve essere specificato:

1. l'indirizzo dell'host che ospita il servizio;
2. la mail dell'utente;
3. il tipo di mobilità scelto tra: still, walking, running, car, train;
4. latitudine
5. longitudine

Dopo la validazione dell'URI e degli input, il server spedisce la seguente query al database MongoDB:

```
db.pois.aggregate([
 { "$geoNear": {
   "near": {
       "type": "Point",
       "coordinates": [ user.lng , user.lat ]
   },
   "distanceField": "distance",
   "spherical": true
 } },
 { "$unwind": "$activities"},
 { "$project": {
     "_id": 1,
     "name": 1,
```

```
14        "location": 1,
15        "radius": 1,
16        "distance": 1,
17        "activityName": "$activities.name",
18        "message": "$activities.message",
19        "fromHours": { $hour: "$interval.from" } ,
20        "toHours": { $hour: "$interval.to" },
21        "fromMin": { $minute: "$interval.from" },
22        "toMin": { $minute: "$interval.to" },
23        "within": { "$lte": [ "$distance", "$radius" ] }
24    } },
25    { "$match": {
26      "within": true,
27      "activityName": user.mobility,
28      "fromHours": { "$lte": time.getHours() },
29      "toHours": { "$gte": time.getHours() },
30    } }
31  ]);
```

Nella riga 1 si utilizza l'aggregation framework di MongoDB che permette di processare una pipeline di dati dove ad ogni stadio vengono eseguite delle operazioni.

Il comando `$geoNear` all'inizio della pipeline restituisce una lista di documenti ordinati in base alla distanza da un punto corrispondente alla longitudine e latitudine dell'utente (`user.lng , user.lat`). MongoDB accetta punti nel formato GeoJSON individuato dal parametro `"near"`. Siccome è stato utilizzato un sistema di indicizzazione per oggetti su una superficie sperica (`2sphere` index) il parametro `"spherical"` è impostato a `true`.

I passi successivi della pipeline sono l'unwind dell'array `"activities"` e la projection dei parametri stessi. Con *projection* si intende la selezione e/o creazione dei parametri utili per lo step successivo della pipeline mentre

l'unwind è la divisione delle entry di un array in documenti diversi.

L'ultima operazione consiste nel verificare per ogni POI, tramite il comando `$match`, se il punto è contenuto all'interno di qualche fence e se i dati contestuali come attività e tempo corrispondono.

Una volta ottenuta la lista dei POI sotto forma di array, questa viene restituita al client che ha chiamato il servizio. L'output assume la seguente forma:

```
[
  {
    "_id": "57beb0723e75ac1d02350817",
    "name": "Mercatino dell'usato",
    "radius": 86.63239378409844,
    "location": {
      "coordinates": [
        11.342815160751343,
        44.49584203104033
      ],
      "type": "Point"
    },
    "distance": 47.282298125109925,
    "activityName": "walking",
    "message": "Fermatevi per comprare qualcosa!",
    "fromHours": 7,
    "toHours": 22,
    "fromMin": 0,
    "toMin": 0,
    "within": true
  },

  {
    "_id": "57d56f635f5c5f2a032866eb",
```

```
25      "name": "Dipartimento di Informatica",
26      "radius": 0,
27      "location": {
28        "coordinates": [
29          [
30            [
31              11.35666608941392,
32              44.496308855784534
33            ],
34            [
35              11.356569529889384,
36              44.49822203229276
37            ],
38            [
39              11.354874373791972,
40              44.498313863186304
41            ],
42            [
43              11.354831458447734,
44              44.49708178665108
45            ],
46            [
47              11.35666608941392,
48              44.496308855784534
49            ]
50          ]
51        ],
52        "type": "Polygon"
53      },
54      "distance": 0,
55      "activityName": "walking",
56      "message": "Benvenuto",
57      "fromHours": 9,
```

```
58      "toHours": 19,
59      "fromMin": 0,
60      "toMin": 0,
61      "within": true
62    }
63  ]
```

Bisogna notare che non tutte le informazioni relative al punto di interesse vengo restituite all'utente, in particolare viene restituito solo il messaggio relativo al tipo di mobilità dell'utente invece che la lista di tutti i possibili messaggi. Una volta restituito il risultato, il servizio salva i dati della richiesta nella storia dell'utente.

```
1   /* Save the request in History */
2       User.findByIdAndUpdate(req.params.id,
3         { "$push": {
4           "history": {
5             "coordinates": {
6               "lat": req.params.lat,
7               "lng": req.params.lng,
8             },
9             "mobility": req.params.mobility,
10            "time": time
11          }
12        }
13      }, function(err, numberAffected){
14
15      });
```

Per fare ciò viene utilizzato un metodo che cerca nella collection *users* quello con l' id corrispondente al richiedente e aggiunge al campo `history` un nuovo `HistoryItem`.

4.3 Client Web

Il client web permette agli utenti admin e business di inserire e visualizzare i POI nonchè vedere gli spostamenti degli utenti che utilizzano l'applicazione mobile. Per rendere l'interfaccia utilizzabile in maniera pratica è stata utilizzata la libreria Maps di Google insieme ad AngularJS.

Google Maps offre numerose API in javascript per interagire con la propria mappa, in particolare offre la possibilità di definire circonferenze e figure poligonali corrispondenti ai confini dei geofence che si intendono aggiungere.

```
1  function createPolygon(pointOfInterest) {
2
3      var color = '#'+Math.floor( Math.random()*16777215 ).
           toString(16);
4
5      var path = [];
6      var coordinates = pointOfInterest.location.coordinates
           [0];
7      for(var i=0; i< coordinates.length; i++){
8        var point = { lat: coordinates[i][1], lng: coordinates[
             i][0] }
9        path.push(point);
10     }
11     var polygon = new google.maps.Polygon({
12       paths: path,
13       strokeColor: color,
14       strokeOpacity: 0.5,
15       strokeWeight: 1,
16       fillColor: color,
17       fillOpacity: 0.2
18     });
19
```

```
20      return polygon;
21    }
```

Il listato sopra mostra come creare un poligono utilizzando le coordinate dei vertici contenute nell'oggetto `pointOfInterst` restituito dal server. Oltre alla forma del poligono è possibile definire alcuni dettagli di stile come: colore dei bordi, colore della figura, spessore e opacità. Per aggiungere il poligono ad una mappa definita nel seguente modo

```
1   // Map initialization data
2   var init = {
3     center: new google.maps.LatLng(44.49500, 11.3439),
4     zoom: 17,
5     mapTypeId: google.maps.MapTypeId.ROADMAP,
6     mapTypeControl: false,
7   };
8
9   // Create the map
10  var  map = new google.maps.Map(document.getElementById("
        mapCanvas"), init);
```

è sufficiente scrivere il seguente comando: `polygon.setMap(map)`.

Grazie alle mappe è possibile definire geofence bidimensionali di qualsiasi forma dimensione e in qualsiasi posizione nel mondo

4.4 Client Mobile

L'applicazione mobile è stata implementata per iPhone, utilizzando il linguaggio Swift 2.0 [26] e ha lo scopo principale di testare il servizio di ricerca dei punti di interesse. È composta da tre tab principali che permettono di rispettivamente di: monitorare le rilevazioni del sensore GPS; mostrare le ultime posizioni, il geofence più vicino all'utente su una MapView e scegliere

il grado di accuratezza della rilevazione; mostrare la lista dei POI trovati e altri messaggi di Log.

Il client mobile risulta essere molto leggero in termini di occupazione di memoria in quanto non mantiene dati relativi ai geofences, ma si limita a richiamare il servizio di ricerca dei POI tramite richiesta HTTP GET ogni n secondi. Il valore di n può essere impostato dall'interno dell'applicazione.

4.4.1 Core Location Framework

Il *Core Location framework* permette allo sviluppatore di determinare la posizione associata ad un dispositivo. Utilizza tutti i sensori hardware presenti per determinare coordinate e direzione dell'utente. Attraverso la classe CLLocationManager è possibile:

1. Tracciare cambiamenti di posizione con diversi gradi di accuratezza;
2. Riportare cambiamenti consistenti di direzione grazie alla bussola;
3. Monitorare regioni di interesse e creare notifiche quando si entra o esce dalla regione;
4. Collezionare aggiornamenti di posizione in background.

Per poter determinare la posizione dell'utente è necessario richiedere prima di tutto la sua autorizzazione. Il metodo `checkCoreLocationPermission()` ha il compito di verificare se l'utente ha acconsentito all'utilizzo dei propri dati relativi alla posizione, anche quando l'applicazione si trova in background.

```
1 // MARK: - CLLocationManagerDelegate
2
3     // Check if the app is authorized
4     func checkCoreLocationPermission() {
```

```
5
6           if CLLocationManager.authorizationStatus() == .
                AuthorizedWhenInUse {
7
8               locationManager.startUpdatingLocation()
9               locationManager.requestAlwaysAuthorization()
10
11          }else if CLLocationManager.authorizationStatus() == .
                NotDetermined {
12
13              // The user has not made a choice
14              locationManager.requestWhenInUseAuthorization()
15              locationManager.requestAlwaysAuthorization()
16
17          }else if CLLocationManager.authorizationStatus() == .
                Restricted {
18
19              // This app is not authorized
20              print("Not Authorized")
21          }
22      }
```

Una volta ottenuta l'autorizzazione viene attivato un timer che ogni n secondi richiede la posizione:

```
1       // Location timer
2       var timer = NSTimer()
3       var TIME_INTERVAL: Double = 15 //seconds
4
5       timer = NSTimer.scheduledTimerWithTimeInterval(
            TIMEINTERVAL , target:self, selector: #selector(self.
            checkForPosition), userInfo: nil, repeats: true)
```

L'intervallo di tempo è espresso dal campo `TIME_INTERVAL`. Nell'esempio, ogni 15 secondi viene richiamato il metodo `checkForPosition`:

```
1   @objc private func checkForPosition() {
2         print("Start tracking position...")
3         locationManager.startUpdatingLocation()
4     }
```

Una volta rilevato un cambiamento di posizione viene richiamato un metodo che individua le coordinate e richiama il servizio di ricerca dei POI messo a disposizione del server tramite HTTP.

Non appena il client ottiene la risposta dal server, il JSON e viene elaborato in modo da estrapolare i dati dei Geofence. Con i dati ottenuti si creano degli oggetti Geofence e il più vicino alla posizione dell'utente viene inserito all'interno della MapView.

```
1   var loc: CLLocationCoordinate2D = CLLocationCoordinate2D()
2   if coords.count == 2 {
3       loc.latitude = coords[1] as! CLLocationDegrees
4       loc.longitude = coords[0] as! CLLocationDegrees
5
6       self.geofence = Geofence(id: id, name: name, message:
            message, center: loc, radius: radius)
7
8       print(self.geofence.getCircularRegion())
9       dispatch_async(dispatch_get_main_queue()) {
10
11          if self.geoOverlay != nil {
12              self.mapView?.removeOverlay(self.geoOverlay)
13          }
14          self.geoOverlay = MKCircle(centerCoordinate: self.
                geofence.getCircularRegion().center, radius:
                self.geofence.getCircularRegion().radius)
```

```
15            self.mapView?.addOverlay(self.geoOverlay)
16
17        }
18
19    }
```

Swift offre il supporto per il Geofencing tramite l'utilizzo delle classi `CLCircularRegion` e `CLRegion` che definiscono i confini delle aree che si intende monitorare. Queste aree possono anche essere visualizzate sulla mappa come nella figura 3.6 (a).

4.4.2 Monitoraggio del livello di batteria

Swift offre un supporto parziale al controllo dei livelli di batteria con la classe `UIDevice` che restituisce un Singleton rappresentante il dispositivo mobile dal quale si possono ottenere anche informazioni riguardanti sensori, sistema operativo e altro.

All'interno dell'applicazione è stata creata una semplice classe BatteryMonitor per controllare il livello di batteria:

```
1  import Foundation
2  import UIKit
3
4  class BatteryMonitor {
5
6      var batteryLevel: Float {
7          return UIDevice.currentDevice().batteryLevel
8      }
9
10     var batteryState: UIDeviceBatteryState {
11         return UIDevice.currentDevice().batteryState
12     }
```

```
13
14      func startMonitoring(){
15          UIDevice.currentDevice().batteryMonitoringEnabled =
                true
16          print(batteryState)
17          print(batteryLevel)
18      }
19
20      func suscribeForBatteryUpdate(){
21          // Add an observer to monitor the battery status and
                level
22          NSNotificationCenter.defaultCenter().addObserver(self
                , selector: #selector(BatteryMonitor.
                batteryStateDidChange(_:)), name:
                UIDeviceBatteryStateDidChangeNotification, object:
                 nil)
23          NSNotificationCenter.defaultCenter().addObserver(self
                , selector: #selector(BatteryMonitor.
                batteryLevelDidChange(_:)), name:
                UIDeviceBatteryLevelDidChangeNotification, object:
                 nil)
24      }
25
26      @objc func batteryStateDidChange(notification:
            NSNotification){
27          print("Battery State Changed")
28          print(notification)
29      }
30
31      @objc func batteryLevelDidChange(notification:
            NSNotification){
32          print("Battery Level Changed")
33          print(notification)
```

```
34      }
35  }
```

La classe UIDevice mette a disposizione due metodi per monitorare i livelli di batteria e notificare l'applicazione quando livello e stato della batteria cambiano in maniera significativa. Le variabili `batteryState` e `batteryLevel` rappresentano rispettivamente lo stato della batteria (unknown, charging, unplugged, full) e il livello espresso da un numero tra 1.0 (carica massima) e 0.0 (scarico).

Capitolo 5

Analisi e Valutazioni

Come introdotto nel secondo capitolo, i problemi legati al geofencing dipendono dalle tecnologie utilizzate per la localizzazione. Infatti, per determinare se l'utente entra o esce da dei confini virtuali è necessario tracciare continuamente la sua posizione, questo porta a consumi elevati della batteria degli smartphone e altre problematiche relative alla privacy [1].

La quantità di energia consumata dipende in larga misura dalla frequenza degli aggiornamenti di pozione, dai sensori utilizzati e dalle richieste effettuate sul network. La soluzione più semplice consiste nel diminuire la frequenza degli aggiornamenti aumentando l'intervallo di tempo tra una misurazione e l'altra. Tuttavia non si può aumentare l'intervallo a piacimento poiché si potrebbero perdere dei punti di interesse se gli aggiornamenti fossero troppo distanti. Di conseguenza è necessario trovare la giusta combinazione tra consumo di batteria, numero di misurazioni e quantità di geofence non rilevati.

5.1 Valutazioni effettuate in letteratura

Garzon e altri [2] hanno effettuato delle prove sperimentali considerando diversi intervalli di tempo e mezzi di locomozione. Dalle rilevazioni è risultato che l'intervallo con con un rapporto maggiore tra numero di POI individuati e numero di aggiornamenti di posizione è tra 50 e 70 secondi. Al di sotto di questo range, il grado di precisione aumenta così come i consumi mentre al di sopra del range il numero di geofence non rilevati cresce in misura maggiore rispetto al risparmio di energia che se ne trae.

Altre ricerche hanno portato a questo risultato sperimentale, in particolare in [8] si individua un intervallo di 60 secondi in aree urbane partendo dal presupposto che un individuo cammina ad una velocità di circa 5 km/h percorrendo 80 metri al minuto. Impostando un limite minimo di 100 metri e uno massimo di 500 al raggio del geofence circolare, la probabilità di attivazione dello stesso risultata molto elevata.

Per aggiustare la frequenza di aggiornamenti sono stati sviluppati diversi algoritmi, alcuni dei quali prendono in considerazione la velocità dell'utente corretta in base alla sua direzione rispetto al target [21], altri invece considerano semplicemente la distanza dal geofence più vicino [4, 5, 21]. Questi algoritmi sono tutti basati su scenari reali che prevedono aree geografiche di qualsiasi tipo (rurali o urbane) e devono adattarsi alla presenza o meno di hotspot WiFi e al tipo di mobilità dell'utente.

Tuttavia, quando ci si trova in aree urbane con alta densità di punti di interesse, quasi tutte le strategie proposte tendono ad effettuare misurazioni GPS e WiFi ad intervallo regolare che si aggira intorno al minuto. Queste misurazioni portano a risparmi energetici del 50% [21], fino al 90% [18] rispetto all'utilizzo costante del GPS nonchè ad una diminuzione del numero di aggiornamenti della stessa entità.

La frequenza degli aggiornamenti influisce anche sulle prestazioni del database in quanto ogni richiesta viene salvata nella storia dell'utente. Questo significa che all'aumentare delle rilevazioni aumenta il tempo di ricerca all'interno della collection. Se si considera, ad esempio, un intervallo di 15 secondi, il numero di aggiornamenti in 10 ore risulta 2400 contro le 600 nel caso si estendesse l'intervallo a 60 secondi. Un modo per aggirare il problema sarebbe quello di eliminare le entry più vecchie o inserire solo aggiornamenti di posizione significativi.

Il consumo di batteria non dipende soltanto dalla frequenza ma anche dalla potenza del segnale GPS. In [20] è stata analizzata la correlazione tra la potenza del segnale in termine di SNR (signal to noise ratio) e il tempo per determinare la pozione. Lo studio ha dimostrato che in presenza di segnale disturbato, come nei canyon urbani, il consumo di batteria è del 38% superiore al consumo del sensore GPS in condizioni ottimali.

Un altro problema legato alla continua localizzazione è quello della privacy. Avendo la possibilità di tracciare gli spostamenti di una persona si possono ottenere informazioni sensibili come dati medici o appartenenza a gruppi religiosi e politici. Questo inconveniente può essere risolto con il geofence stesso, in quanto si posso delimitare delle zone in cui la posizione dell'utente non deve essere tracciata oppure si possono introdurre dei meccanismi che permettano di offuscare o rendere anonimi i dati.

5.2 Test e Valutazioni dell'Applicazione

L'applicazione mobile è stata installata su uno smartphone iPhone 6s del 2015 con una batteria da 1715 mAh con lo scopo di effettuare alcuni test e determinare il rapporto tra consumo di batteria e frequenza di aggiornamenti

di posizione.

A tal fine sono stati tracciati due percorsi: il primo pensato per un viaggio in macchina e il secondo per una camminata. Entrambi i percorsi prevedevano una durata di 30 minuti e sul tragitto erano stati disseminati dei geofence circolari di vario raggio (rispettivamente 10 per il primo tragitto e 15 per il secondo).

Prima di ogni test il dispositivo è stato caricato al 100% e alla fine di ogni test venivano registrati il numero di POI individuati e la percentuale di batteria rimanente. Ogni prova differiva dall'altra per il tipo di mobilità (*car* o *walking*) e intervallo di aggiornamento della posizione (15 secondi, 60 secondi, aggiornamento continuo e nessun aggiornamento). I risultati sono mostrati nella tabella 5.1.

	Batteria	**POI Car**	**POI Walking**
App spenta	99%	—	—
Continuo	91%	10/10	15/15
15 sec	94%	9/10	15/15
60 sec	97%	6/10	14/15

Tabella 5.1: Risultati dei test.

I risultati dei test confermano quanto detto in precedenza: all'aumentare dell'intervallo, diminuisce il consumo di batteria e il valore ottimo che permette di mantenere i consumi ridotti e garantire anche una buona percentuale di POI individuati si aggira intorno ai 60 secondi. Con il suddetto intervallo si ha un consumo del solo 2% in più rispetto all'utilizzo del dispositivo con l'applicazione spenta e allo stesso tempo si riescono ad individuare 14 POI sui 15 presenti. Il valore cala drasticamente a 6 su 10 quando il tipo di mobi-

lità passa da camminata a guida in macchina. Questo dipende dal fatto che l'automobile, muovendosi ad una velocità media di 50 km/h, percorre circa 830 metri in 60 secondi e può perdere qualche *geofence* di piccola dimensione.

I risultati dei seguenti test sono fortemente influenzati anche dalla dimensione e distribuzione dei *geofence* sui percorsi scelti. Questo porta ad affermare che, su strade ad alta velocità di percorrenza, sia necessario creare delle aree virtuali di almeno 600m di raggio mentre su strade pedonali e centri cittadini un raggio di 200/300 metri può essere considerato soddisfacente.

Conclusioni e Sviluppi Futuri

Nella seguente trattazione si è affrontato il tema del geofencing all'interno dei servizi basati sulla localizzazione. È stata effettuata un'analisi dei possibili utilizzi e delle tecnologie che rendono possibile la geolocalizzazione dei dispositivi mobili degli utenti.

Dopo aver concluso l'analisi delle tecnologie parlando del supporto offerto dai database (in particolare MongoDB) ai dati di tipo spaziale, ci si è concentrati sulla progettazione e implementazione di un sito web che permettesse ad esercizi commerciali e negozi di creare e gestire dei geofence.

Il sito web, implementato sfruttando la tecnologia REST offriva un servizio per permettere all'applicazione mobile, sviluppata per iPhone, di tracciare gli spostamenti degli utenti, inviando la posizione al servizio e ricevendo in risposta una lista di POI attivi nelle vicinanze.

Le difficoltà maggiori quando si procede allo sviluppo di un'applicazione basata su geofence sono legate soprattutto all'efficienza. Con questo termine si intende, da un lato, l'utilizzo in maniera parsimoniosa dei sensori di localizzazione come il GPS, mentre dall'altro ci si riferisce alla gestione ottimale dei dati. Infatti i dati, come gli aggiornamenti sulla posizione degli utenti, non devono eccedere oltre al necessario per poter permettere di effettuare ricerche in tempi rapidi.

Per raggiungere questo obiettivo è necessario sviluppare un algoritmo che

utilizzi la combinazione gusta di sensori disponibili su uno smartphone per ottenere l'accuratezza necessaria in quel momento e calcoli degli intervalli di tempo per ridurre il numero di aggiornamenti di posizione. L'algoritmo dovrà dipendere da valori come la posizione, la velocità dell'utente, la sua direzione e da alcune caratteristiche dei geofence come dimensione, densità e distanza dall'utente.

In futuro si intende portare avanti questa analisi cercando di aumentare le funzionalità messe a disposizione dall'applicazione web nonchè la creazione di un algoritmo che permetta di ridurre al minimo il consumo di batteria e che mantenga allo stesso tempo un grado di accuratezza elevato ed una buona percentuale di POI rilevati. Si intende inoltre sviluppare il client mobile per Android affiancabile a quello sviluppato in iOS ed un sistema accurato per il rilevamento del tipo di attività dell'utente.

Bibliografia

[1] U. Bareth, A. Küpper, and B. Freese, "Geofencing and Background Tracking - The Next Features in LBS," in Proc. of the 41th Annual Conf. of the Gesellschaft für Informatik e.V. (INFORMATIK 2011), vol. 192. Berlin, Germany: Köllen Druck + Verlag GmbH, Oct 2011.

[2] S. R. Garzon, B. Deva, G. Pilz, S. Medack, "Infrastructure-assisted Geofencing: Proactive Location-based Services with Thin Mobile Clients and Smart Servers" in 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 2015 pp. 61-70.

[3] F. Reclus and K. Drouard, "Geofencing for Fleet & Freight Management," in Proc. of 9th Int. Conf. on Intelligent Transport Systems Telecommunications,(ITST), Oct 2009, pp. 353-356.

[4] G. Cardone, A. Cirri, A. Corradi, L. Foschini, R. Ianniello, and R. Montanari, "Crowdsensing in Urban Areas for City-Scale Mass Gathering Management: Geofencing and Activity Recognition" Published in IEEE Sensors Journal, Vol. 14, NO. 12, Dec. 2014 pp. 4185-4195

[5] I. Carreras, D. Miorandi, A. Tamilin, E. R. Ssebaggala, and N. Conci, "Matador: Mobile task detector for context-aware crowd-sensing

campaigns," in Proc. IEEE PERCOM Workshops, Mar. 2013, pp. 212-217.

[6] A. Greenwald, G. Hampel, C. Phadke, and V. Poosala, "An Economically Viable Solution to Geofencing for Mass-Market Applications," Bell Labs Technical Journal, vol. 16, no. 2, pp. 21-38, Sept 2011.

[7] Y. Nakamura, M. Sekiya, K. Honda, and O. Takahashi, "An effective power saving method in Geo-fencing service using temperature sensors," in Proc. of the Int. Conf. on Consumer Electronics (ICCE-TW), May 2014, pp. 139-140.

[8] https://support.apple.com/en-us/HT205890 Visitato in data: 10 Sept. 2016.

[9] https://www.facebook.com/business/a/local-awareness-ads, Visitato in data: 2 Sept. 2016.

[10] http://www.pcmag.com/article2/0,2817,2493418,00.asp Visitato in data: 3 Sept. 2016.

[11] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: Current state and future challenges," IEEE Commun. Mag., vol. 49, no. 11, pp. 32-39, Nov. 2011.

[12] R. R. Oliveira, I. Cardoso, J. Barbosa, C. da Costa, M. Prado, "An intelligent model for logistics management based on geofencing algorithms and RFID technology", Expert Systems with Applications, Vol. 42, Issues 15-16, Sept. 2015, Pages 6082-6097

[13] https://eng.uber.com/go-geofence/ , Visitato in data 9 Settembre 2016.

[14] T. Gurriet, L. Ciarletta, "Towards a generic and modular geofencing strategy for civilian UAVs", International Conference on Unmanned Aircraft Systems (ICUAS) Arlington, VA USA, June 7-10, 2016, pp. 540-549.

[15] http://www.dji.com/fly-safe/category-mc?www=v1, Visitato in data: 5 Sept. 2016.

[16] U. Bareth and A. Küpper, "Energy-Efficient Position Tracking in Proactive Location-Based Services for Smartphone Environments," in Proc. of the 35th Annual Int. Computer Software and Applications Conference (COMPSAC 2011). IEEE, July 2011, pp. 516-521.

[17] http://www.gps.gov/systems/gps/ Visitato in data: 8 Sept. 2016

[18] L. Tawalbeh, A. Besalamah, R. Mehmood, H. Tawalbeh, "Greener and Smarter Phones for Future Cities: Characterizing the Impact of GPS Signal Strenght on Power Consumtion" in IEEE Access, 10 Feb. 2016, Vol. 4, pp. 858-868.

[19] T. Nakagawa et al., "Variable interval positioning method for smartphone-based power-saving geofencing," in Proc. IEEE PIMRC, Sep. 2013, pp. 3482-3486.

[20] Elaheh Pourabbas, "Geographical Information Systems Trends and Technologies", CRC Press Taylor & Francis Group, 2014.

[21] https://docs.mongodb.com Visitato in data: 4 Agosto 2016.

[22] Y. Hu, S. Ravada, R. Anderson, B. Bamba, "Distance Queries for Complex Spatial Objects in Oracle Spatial" in Proc. SIGSPATIAL 2014, pp 291-300

[23] https://medium.com/@buckhx/unwinding-uber-s-most-efficient-service-406413c5871d#.8s90l9ttw Visitato in data: 10 Sept. 2016.

[24] http://www.w3.org/2010/POI/wiki/Main_Page Visitato in data: 10 Sept. 2016.

[25] https://expressjs.com Visistato in data: 13 Sept. 2016

[26] https://developer.apple.com/ Visitato in data: 13 Sept. 2016

Ringraziamenti

Prima di tutto vorrei ringraziare i miei genitori per tutti i sacrifici che hanno fatto per me.

Ringrazio il gruppo dei 7, che considero la mia seconda famiglia, Nicola per il suo supporto a distanza e Riccardo per avermi sempre motivato a fare meglio.

Infine, ringrazio il prof. Di Felice per avermi dato l'opportunità di trattare questo argomento e per la sua disponibità.