



UNIVERSITÀ DEGLI STUDI DELL'AQUILA
DIPARTIMENTO DI
INGEGNERIA E SCIENZE
DELL'INFORMAZIONE E MATEMATICA



CORSO DI LAUREA IN
INGEGNERIA DELL'INFORMAZIONE

**Un'applicazione mobile geolocalizzata cross-platform per
contesti di disaster-management**

Relatrice:

Prof.ssa Laura Tarantino

Candidato:

Fabio Di Sabatino

Co-relatore:

Prof. Giovanni De Gasperis

Matricola:

219202

ANNO ACCADEMICO 2015–2016

*Dedico il presente lavoro
a tutte le persone che mi sono
state vicine durante questi anni*

Indice

| | |
|---|-----------|
| Sommario | 1 |
| Introduzione | 2 |
| 1 Lo scenario applicativo | 4 |
| 1.1 La vulnerabilità | 4 |
| 1.2 Le fasi di emergency management | 7 |
| 1.3 I luoghi comuni sui disastri | 10 |
| 1.4 Il contesto d'uso | 15 |
| 2 OpenStreetMap | 18 |
| 2.1 Cos'è OpenStreetMap ? | 18 |
| 2.2 Perchè il mondo ha bisogno di OSM | 21 |
| 2.3 OpenStreetMap vs Google Maps | 23 |
| 3 L'applicazione lato utente | 28 |
| 3.1 I concetti principali | 28 |
| 3.2 Requisiti di sistema | 29 |
| 3.2.1 Requisiti funzionali | 29 |
| 3.2.2 Requisiti non funzionali | 33 |
| 3.3 L'interfaccia grafica | 36 |
| 4 L'implementazione | 47 |
| 4.1 Le applicazioni cross-platform | 47 |
| 4.2 PhoneGap | 49 |
| 4.3 Ratchet | 53 |
| 4.4 Leaflet | 56 |
| 4.5 La griglia trasparente | 58 |

| | | |
|-----------------------|---|-----------|
| 4.5.1 | L'algoritmo Santiago | 59 |
| 4.5.2 | Il codice | 64 |
| 4.6 | La struttura architetturale | 66 |
| 5 | Conclusioni e prospettive future | 73 |
| Ringraziamenti | | 74 |
| Bibliografia | | 75 |

Elenco delle figure

| | |
|---|----|
| 1.1 Ciclo di fasi per il disaster-management | 7 |
| 1.2 Contesto d'uso del sistema all'interno del disaster-management cycle | 15 |
| 2.1 Il logo di OpenStreetMap | 19 |
| 2.2 Scala lineare di utenti iscritti ad OpenStreetMap | 20 |
| 2.3 Risultato ricerca "colazione" | 22 |
| 2.4 Polo universitario di Coppito-L'Aquila | 24 |
| 2.5 Mappa OSM di Pourt-au-Prince 6 ore dopo il terremoto del 2010 . . | 26 |
| 2.6 Mappa OSM di Pourt-au-Prince 48 ore dopo il terremoto del 2010 . | 27 |
| 3.1 Dashboard | 37 |
| 3.2 Tap bottone " <i>How are you?</i> " | 38 |
| 3.3 Lista predefinita di stati selezionabili | 38 |
| 3.4 Tap del bottone " <i>Signal event</i> " | 39 |
| 3.5 Lista predefinita dell'emergenze | 39 |
| 3.6 Tap " <i>locate on the map</i> " | 40 |
| 3.7 Mappa mostrata dal sistema | 40 |
| 3.8 Tap da fare per accedere alla lista dei FOI | 41 |
| 3.9 Lista dei FOI ordinata per default in base alla gravità | 41 |
| 3.10 Tap per aprire il menu a comparsa | 42 |
| 3.11 Tap da fare per ordinare la lista secondo i tre parametri | 42 |
| 3.12 Tap per visualizzare i FOI sulla mappa | 43 |
| 3.13 Mappa centrata sul FOI più vicino | 43 |
| 3.14 Due markercluster contenenti rispettivamente due e tre FOI | 44 |
| 3.15 Tap su un markercluster contenente cinque FOI | 44 |
| 3.16 Tap da fare per accedere ai propri POI | 45 |
| 3.17 Lista dei POI ordinata per default in base al più recente | 45 |
| 3.18 Tap sul marker di un FOI | 46 |

| | |
|--|----|
| 3.19 Tap sul marker di un POI | 46 |
| 4.1 Utilizzo dei vari OS su scala mondiale | 47 |
| 4.2 Logo del framework PhoneGap | 49 |
| 4.3 Logo del framework PhoneGap | 50 |
| 4.4 Architettura di un'applicazione realizzata con PhoneGap | 51 |
| 4.5 Le API e il loro supporto per le varie piattaforme | 52 |
| 4.6 Il logo del framework Ratchet | 53 |
| 4.7 Il componente tab-bar con cinque icone | 55 |
| 4.8 Il logo della libreria Leaflet | 56 |
| 4.9 Risultato grafico con il codice precedente | 57 |
| 4.10 Sistema di riferimento dell'utente | 59 |
| 4.11 Troncamento delle coordinate alla quarta cifra decimale posta a zero | 60 |
| 4.12 Alcune celle evidenziate in blu | 61 |
| 4.13 Proiezione delle distanze utilizzate per il calcolo della cella | 62 |
| 4.14 Posizione ottenuta riconvertendo la tupla di dati vista nel passo 3 dell'algoritmo | 63 |
| 4.15 Rappresentazione semplificata dell'architettura | 66 |
| 4.16 Rappresentazione della sequenza di azioni per la richiesta dei propri FOI e POI | 67 |
| 4.17 Rappresentazione semplificata dell'architettura | 70 |

Sommario

Lo scopo di questa tesi è lo sviluppo di un'applicazione mobile cross-platform geolocalizzata, per contesti di disaster management.

Gli utenti possono segnalare la posizione delle emergenze e il proprio stato fisico; queste informazioni vengono inviate ad un main-server, il quale sarà utilizzato dai soccorritori per gestire il disastro e coordinare al meglio le operazioni di salvataggio. Inoltre attraverso l'applicazione è possibile visualizzare lo stato dei propri familiari, dei luoghi d'interesse e quello generale dell'area colpita.

Per realizzare l'applicazione si sono utilizzati i seguenti framework: phonegap per renderla cross-platform, Ratchet per realizzare l'interfaccia e Leaflet per la mappa. Le posizioni degli utenti e degli eventi sono mappate in una griglia trasparente all'utente, utilizzando un algoritmo appositamente ideato.

Abstract

The purpose of this thesis is the development of a cross-platform geolocalized mobile application for disaster management.

Users can signal the position of emergencies and send their physical status; these information are sent to a main-server that will be used by rescuers to manager and optimize the rescue operations. Through this application users can also check the status of their family and other point of interest, moreover they can visualize the general state of the area affected by disaster.

Introduzione

Durante una lezione del corso di progettazione di sistemi interattivi, tenuto dalla prof.ssa Laura Tarantino, fu chiesto a tutti gli studenti cosa ritenevano più tecnologico tra un libro e il proprio laptop. Una domanda apparentemente banale alla quale tutti gli studenti, me compreso, hanno risposto: laptop.

Ci venne spiegato che incoscientemente siamo portati a ritenere un oggetto tecnologico se questo è stato inventato dopo la nostra nascita o comunque in epoca recente.

Il fatto di non essere riuscito a rispondere correttamente a questa domanda mi ha fatto riflettere, come può un ingegnere non sapere per certo cosa sia tecnologico? Quando guardando la pubblicità dell'ultimo *smartwatch* la risposta mi è parsa ovvia: negli ultimi decenni, il concetto di tecnologia è stato manipolato, trasformato e usato a fini puramente commerciali, inculcando nelle nostre menti un significato sbagliato di questo nobile concetto. Non ho nulla contro questi accessori e condivido la necessità di realizzare profitti; piuttosto contesto l'allocazione totale delle risorse e della forza lavoro nella sola ricerca/produzione di **tecnologia superflua**.

La tecnologia è progresso, è l'insieme di tutti gli oggetti e studi che in qualche modo migliorano significativamente la nostra vita; come ingegnere (aspirante), credo di dover contribuire a questo progresso (o almeno provarci).

Concludo dicendo che, non ho accettato questa tesi al fine unico di conseguire il titolo bensì con la speranza che il mio lavoro possa essere una **tecnologia** per il disaster-management e che più in generale contribuisca, anche se in maniera infinitesimale, al progresso tecnologico.

La presente tesi è così strutturata:

Capitolo 1: In questo capitolo viene esposto il concetto di vulnerabilità e il contesto d'uso del sistema.

Capitolo 2: In questo capitolo vengono illustrate le questioni etiche e i motivi tecnici affrontati nella scelta del map-provider.

Capitolo 3: In questo capitolo vengono riportati i requisiti di sistema e alcune schermate significative dell'interfaccia grafica.

Capitolo 4: In questo capitolo si riportano i framework utilizzati e alcuni dettagli implementativi dell'applicazione.

Capitolo 1

Lo scenario applicativo

In questo capitolo si illustra brevemente il concetto di vulnerabilità e l'equazione concettuale che esprime il legame tra l'impatto di un disastro ed essa. Viene quindi spiegato un framework ciclico per il disaster-management e si sfatano i luoghi comuni legati al contesto di questa tesi. Contesto che viene infine definito nell'ultimo paragrafo.

1.1 La vulnerabilità

La parola vulnerabilità [1] deriva dal latino *"vulnerare"* che significa ferire. Principalmente si riferisce all'esporre persone, beni e arrività a potenziali danni e/o perdite.

La vulnerabilità è un concetto astratto, ma comunque reale, difficilmente misurabile. La sua esistenza si percepisce successivamente ad un evento, verificando l'impatto del disastro stesso. Dunque è una proprietà latente o intrinseca.

Uno dei grandi risultati di studi disastri nella seconda metà del XX secolo è stato quello di stabilire che la vulnerabilità è la componente principale del rischio (Hewitt 1983). In formulazioni più estreme, il pericolo (l'altro principale componente) è considerato come la probabilità che accada un fenomeno dannoso e la vulnerabilità la propensione a subire danni. E' stata quindi formulata un'equazione concettuale che lega questi concetti, ovvero:

$$\text{Pericolo} * (\text{Vulnerabilità} * \text{Esposizione}) = \text{Rischio} \rightarrow \text{Impatto} \quad (1.1)$$

Per **Pericolo** si intende la probabilità che un certo fenomeno colpisca una certa area mentre per **Esposizione** si intende il lasso di tempo nel quale, una persona, o una risorsa è minacciata da un particolare rischio. Dato il ruolo di *Esposizione*, è importante notare che la vulnerabilità non è un concetto "tutto o niente".

Molti studi di rischio sono basati sulla propensione alle perdite totali. Questo, naturalmente, presuppone una totale incapacità di resistere all'impatto del disastro. E' necessario quindi introdurre un altro concetto: la **Resilienza**. La resilienza deriva dal comportamento fisico dei materiali, e si riferisce alla capacità di una sostanza (o in questo caso della società) di assorbire e resistere al trauma di un disastro. È, ovviamente, l'inverso della vulnerabilità; possiamo quindi raffinare l'equazione precedente:

$$\text{Pericolo} * (\text{Vulnerabilità} * \text{Esposizione}) / \text{Resilienza} = \text{Rischio} \rightarrow \text{Impatto}$$

Quindi, la **Vulnerabilità** può essere anche parziale. Se è quantificabile può essere espressa come un indice o una percentuale relativa alla perdita totale. Se può essere stimata si possono usare varie metriche, come la gravità dei danni rispetto alla letalità oppure i gradi di perdita dell'integrità strutturale di un edificio rispetto al suo collasso totale.

La possibilità di disaggregare la vulnerabilità in varie componenti indica che questa può assumere diverse forme, tuttavia queste non sono indipendenti le une dalle altre. Una possibile interpretazione della vulnerabilità è che **può essere definita rispetto alle circostanze che la generano**. Il seguente modello la scomponete in funzione del contesto (Alexander 1997 Özerdem e Jacoby 2006):

- **Vulnerabilità totale:** la vita è generalmente precaria perché poco o nulla è stato fatto per ridurre le fonti e i potenziali rischi di impatti. Questa condizione tende a manifestarsi soprattutto nelle società più povere ed emarginate che non dispongono delle risorse per proteggersi.
- **Economici:** le persone non hanno un'adeguata occupazione, quindi la vulnerabilità si riferisce alla precarietà delle loro attività produttive e alle fonti di reddito.
- **Tecnologia:** causata dalla pericolosità della tecnologia o dal modo in cui viene usata.
- **Delinquenti:** causata dalla corruzione, negligenza o attività criminali che mettono in pericolo le persone e i beni.

- **Appena generato:** causato dal cambiamento delle circostanze , per esempio a causa di rischi emergenti .

Data l'etereogenità delle moderne società , tali categorie non sono mutue esclusive. Se accettiamo che la vulnerabilità può assumere diverse forme, allora dobbiamo tener conto delle possibili interazioni tra le varie componenti. Possiamo definire:

- **Vulnerabilità primaria:** il prodotto diretto tra causa ed effetto. Ad esempio, se un terremoto squote una casa, la scarsa qualità delle murature potrebbe causare il crollo totale della costruzione.
- **Vulnerabilità secondaria:** è il risultato dell'interazione tra le cause o il verificarsi di coincidenze. Ad esempio, un edificio può resistere alle scosse di un terremoto ma non all'inondazione causata dal collasso di una diga a monte (Disastro del Vajont).
- **Vulnerabilità complessa:** si presenta quando le complicate interazioni tra le componenti aumentano complessivamente la vulnerabilità. Gli effetti economici ramificati di un forte terremoto, in una città metropolitana, ne sono un chiaro esempio.

La vulnerabilità non è statica, se in questo momento qualcuno è "vulnerabile", questo non implica che lo sarà futuro; analogamente vale il viceversa, le persone potrebbero diventare vulnerabili a causa di forze o processi come l'invecchiamento o la malattia che sono eventi indipendenti dai disastri. Quindi **l'analisi della vulnerabilità può essere vista come un'istantanea di un processo dinamico** e come tale ha una validità temporale limitata.

La vulnerabilità può essere misurata o stimata direttamente come il potenziale danno o perdita oppure, in modo indiretto misurando la non resilienza. Queste misure richiedono però di ipotizzare il potenziale impatto di un probabile evento non avvenuto, di conseguenza la presenza di errori è inevitabile.

In conclusione le misure prese per ridurre la vulnerabilità devono essere sostenibili; inoltre devono essere locali, supportate dalla comunità, ben integrate nella legislazione e pianificate al fine di rendere la vita degli abitanti, della zona in questione, più "*resistente*".

1.2 Le fasi di emergency management

Per aumentare la "resilienza" (vedi 1.1) della comunità è necessario prepararsi in tempi di pace, saper gestire i disastri e cercare di prevenirli quando possibile. Tutte queste attività prendono il nome di **Emergency management**.

Il termine "emergency management" è usato per indicare tutte quelle attività, condotte da agenzie private o pubbliche, che hanno lo scopo di fornire supporto e assistenza ai territori colpiti da disastri ambientali e/o umani.

La gestione di tali cataclismi, può essere divisa in quattro fasi, come illustrato in Figura 1.1 :



Figura 1.1: Ciclo di fasi per il disaster-management

Le definizioni di queste fasi sono svariate e variano a seconda dell'agenzia che le fornisce; a grandi linee possono essere così descritte [3]:

- **Mitigation:** Include tutte quelle attività svolte per ridurre le perdite di vite e proprietà a causa di un disastro naturale e/o umano: *“un’azione continua che riduce o elimina il rischio a lungo termine per le persone e le proprietà da pericoli naturali e dai loro effetti”*. L’attuazione di strategie di Mitigation è una parte della fase di Recovery se applicata dopo il verificarsi di un disastro. Le misure di Mitigation possono essere:

- **Strutturali**, forniscono soluzioni tecnologiche, come l’ampliamento degli argini di un fiume.
- **Non strutturali**, forniscono soluzioni legislative, come il divieto di costruzione su terreni palustri.

- **Preparednes:** Ha lo scopo di preparare al meglio la comunità attraverso attività formative.

Questa fase è un continuo ciclo di pianificazione, organizzazione, prove e valutazioni; in questo modo l’emergency manager sarà in grado di fornire la miglior soluzione.

Nella fase di preparednes si sviluppano piani d’azione per gestire e contrastare i rischi e si costruiscono le risorse necessarie per attuare tali piani. Alcune misure comuni sono:

- Comunicazione dei piani con terminologia e metodi comprensibili.
- Corretta manutenzione dei servizi d’emergenza.
- Sviluppo ed esercizio di metodi di avviso emergenza per la popolazione.
- Creazione di rifugi e piani di evacuazione

- **Response:** Include la mobilitazione dei servizi di emergenza necessari e primi interventi nell'area colpita; è probabile che includano una prima ondata di servizi base come pompieri, polizia e ambulanze.

In particolare in questa fase si attuano i piani precedentemente ideati nella fase di preparednes e le attività dovrebbero includere:

- Ricerca e soccorso.
- Distribuire viveri e medicinali
- Valutazione dei danni
- Riparo per le vittime
- Estinzione di eventuali incendi

- **Recovery:** Include tutte quelle attività che hanno l'obiettivo di riportare le persone alla loro "normalità". Riparazione, sostituzione e ricostruzione sono esempi tipici.

Lo scopo della fase di Recovery è appunto ripristinare l'area colpita allo stato precedente la catastrofe, tuttavia le azioni devono essere eseguite soltanto quando i bisogni della comunità sono stati soddisfatti e la fase di Response è quindi terminata; a quel punto si possono avviare le attività di ripristino con lo sforzo di "ricostruire indietro meglio", cioè provando a ridurre i rischi, precedenti al disastro, presenti nella comunità e nelle infrastrutture.

La fase di Recovery può essere divisa in due periodi. La fase a breve termine, in genere, dura da sei mesi almeno ad un anno e prevede la fornitura di servizi immediati alle imprese. La fase a lungo termine, che può durare anche decenni, richiede una pianificazione strategica efficente per affrontare gli impatti più gravi e permanenti di un disastro.

Le comunità devono accedere e distribuire una vasta gamma di risorse pubbliche e private per consentire la ripresa economica a lungo termine.

Un utilizzo ottimale di questo framework è in grado di fornire alla comunità le risorse necessarie affinché un disastro possa essere affrontato in maniera efficace, o meglio, si possono ridurre tutte quelle morti causate dalla negligenza di "involucri organici", privi di qualsiasi etica e morale erroneamente chiamati professionisti.

1.3 I luoghi comuni sui disastri

Se ci venisse chiesto di immaginare lo scenario immediato ad un disastro ambientale o umano, nella maggior parte dei casi, potremmo scrivere una buona trama per un film apocalittico. Questo succede a causa di alcuni luoghi comuni diffusi nella nostra società [2], ciò che ci sembra ovvio infatti potrebbe essere soltanto un mito inculcato nella nostra mente:

1. **I disastri sono eventi rari**, sono una parte normale della vita quotidiana e in molti casi sono eventi ripetitivi.
2. **I disastri naturali sono inevitabilmente il risultato della furia di madre natura**, le cause di questi eventi sono sì naturali, infatti non possiamo fare niente per eliminare i terremoti, le inondazioni o le tempeste tropicali. Tuttavia, il disastro è quasi sempre causato dalle persone e dalle comunità stesse che si mettendo in condizioni di rischio e non vi è nulla di molto naturale o inevitabile in tutto questo.
3. **I disastri provocano una grande quantità di caos e non possono essere gestiti in modo sistematico**, ci sono ottimi modelli teorici su come disastri funzionano e come gestirli. Dopo più di 75 anni di ricerca nel campo, gli elementi generali del disastro sono ben noti, e tendono a ripetersi da un disastro all'altro.
4. **I disastri uccidono le persone indipendentemente dal loro status sociale o economico**, le persone povere ed emarginate sono molto più a rischio di morte di quelle ricche e benestanti.
5. **I terremoti sono comunemente responsabili di bilanci molto elevati di morte.**, gli edifici che crollano sono i veri responsabili. Come già detto non è possibile fermare i terremoti, è possibile però costruire edifici antisismici e organizzare attività umane in modo tale da minimizzare il rischio di morte.
6. **Quando avviene il disastro il panico è la reazione più comune**, la maggior parte delle persone mantiene un comportamento razionale. Il panico non è comunque da escludere.
7. **Un gran numero di persone fuggirà dall'area colpita**, di solito c'è un "reazione di convergenza" e la zona si riempie di persone. Alcuni dei sopravvissuti se ne andranno, altri saranno evecuati ma durerà comunque poco.

8. **Dopo il disastro i sopravvissuti tendono ad essere storditi e apatici**, i sopravvissuti tendono a mettersi al lavoro per "ripulire". L'attivismo è molto più comune della rassegnazione (questa è la cosiddetta "comunità terapeutica"). Nel peggiore dei casi solo il 15-30 per cento delle vittime mostrano reazioni passive.
9. **Le persone tendono a prendere decisioni sbagliati a meno che non siano guidati dalle autorità**, le persone prendono decisioni sulla base delle informazioni che sono in grado di ottenere e di interpretare. All'interno di questa "bussola", la maggior parte del processo decisionale può essere giudicato razionale.
10. **I disastri di solito danno luogo a diffuse manifestazioni spontanee di comportamento antisociale**, in generale i comportamenti sono caratterizzati da una grande solidarietà sociale, generosità e sacrificio di sé stessi.
11. **Il saccheggio è un fenomeno comune**, il fenomeno dei saccheggi è raro e di portata limitata. Si verifica soprattutto quando ci sono presupposti forti, come quando una comunità è già profondamente divisa.
12. **Le persone ricorrono alla violenza per proteggere i propri interessi**, La 'comunità terapeutica' è comune, infatti le persone hanno una maggiore tendenza ad aiutarsi a vicenda in situazioni d'emergenza che in tempi normali.
13. **La legge marziale deve essere imposta dopo il disastro al fine di evitare che la società si rompa del tutto**, L'imposizione della legge marziale dopo il disastro è estremamente raro, ad ogni modo il suo utilizzo implica che i normali meccanismi di governo non sono mai stati efficaci.
14. **I cadaveri non seppelliti costituiscono un pericolo per la salute**, Nemmeno la decomposizione costituisce un significativo rischio per la salute. Seppellire in modo avventato demoralizza i sopravvissuti e sconvolge le modalità di certificazione di morte, i riti funebri, e, ove necessario, l'autopsia.
15. **Le epidemie sono risultati quasi inevitabili delle distruzioni e delle scarse condizioni salutari causate dalla maggior parte dei disastri**, in generale, il livello di sorveglianza epidemiologica e l'assistenza sanitaria nella zona del disastro è sufficiente per fermare ogni possibile epidemia. Tuttavia, il tasso

di diagnosi di malattie può aumentare con una migliore assistenza sanitaria, soprattutto in aree molto povere.

16. **Una grande quantità e assortimento di medicinali dovrebbe essere inviato**, gli ospedali da campo sono solitamente installati troppo tardi per curare i feriti, finiscono per fornire assistenza generale e continuità delle cure. Poiché il trasporto e il funzionamento di questi tende ad essere costoso e logisticamente impegnativo, in alcuni casi può essere più efficiente per tentare di ripristinare o aumentare ospedali esistenti nel settore, anche se sono significativamente danneggiati.
17. **A seguito di un disastro la vaccinazione di massa è un ottima strategia per fermare la diffusione di malattie**, considerando che la vaccinazione mirata di gruppi specifici (ad esempio, bambini, medici e infermieri) può essere una strategia efficace, la vaccinazione di massa indiscriminata è invece uno spreco.
18. **I cadaveri, i sopravvissuti, strade, macerie e altre cose devono essere irrorati con un disinfettante per fermare la diffusione della malattia**, questa misura comune e popolare sposta grandi quantità di disinfettante e non fa assolutamente nulla per la salute pubblica.
19. **Di solito c'è una carenza di risorse quando si verifica un disastro e questo impedisce di gestirlo in modo efficace**, le carenze se si verificano, sono quasi sempre molto provvisorie. In realtà ci sono più problemi nella buona distribuzione e utilizzo in modo efficiente rispetto all'acquisto vero e proprio.
20. **Qualsiasi tipo di aiuto è utile dopo il disastro purché sia fornito abbastanza rapidamente**, iniziative avventate e frettolose tendono a creare il caos. Inoltre saranno necessari solo alcuni tipi di assistenza tecnica, beni e servizi; infatti non tutte le risorse utili preesistenti al disastro saranno distrutti. La donazione di materiali inutilizzabili o manodopera consuma risorse di organizzazione e di alloggio che potrebbero più proficuamente essere utilizzate per ridurre il bilancio del disastro.
21. **Al fine di gestire al meglio un disastro è necessario accettare qualsiasi aiuto venga offerto**, è molto meglio limitarsi ad accettare donazioni di beni e servizi che sono effettivamente necessari nella zona del disastro.

22. **C’è un fenomeno noto come ”terremoto tempo”**, la credenza popolare che i terremoti si verifichino quando vi è un clima afoso non ha alcuna base di fatto. Numerosi studi scientifici hanno cercato di individuare le condizioni atmosferiche precursorie ad un terremoto, ma l’unica osservazione riguarda il rilascio di alogenzi nell’atmosfera.
23. **Gli animali sono in grado di percepire il terremoto prima che accada**, in realtà gli animali si comportano in modo insolito prima di un terremoto, tuttavia non è un modo affidabile di sapere se tale fenomeno sta per accadere.
24. **Siamo ben organizzati ad affrontare una pandemia o un attacco nucleare**, nella maggior parte dei paesi, compresi quelli più ricchi e più grandi, la fase di Preparedness è quella più trascurata e carente.
25. **In caso di un attacco terroristico biologico siamo in grado di confinare l’epidemia in modo efficace**, le scorte di anticorpi e vaccini sono insufficienti così come, i reparti di isolamento, le unità di risposta sul campo, le unità di decontaminazione, e la formazione per i soccorritori e medici. Inoltre potrebbe anche essere difficile da individuazione tempestivamente l’agente patogeno o la tossina in questione.
26. **Il panico e comportamenti irrazionali sono inevitabili conseguenze di un attacco terroristico nucleare**, in disastri di ogni genere maggior parte delle persone si sforzano di comportarsi razionalmente e prendere decisioni razionali . Questo è antitetica al panico. Tuttavia, se le persone non hanno informazioni adeguate, il loro processo decisionale può sottrarsi all’analisi razionale.
27. **I soccorritori non si presenterebbero in caso di disastro, poiché impegnati a proteggere la propria famiglia**, non è comune l’assenteismo di massa tra i soccorritori. In generale, le persone tendono ad avere un maggiore senso del dovere.
28. **I disastri accadono sempre a qualcun altro**, la sindrome dell’ invulnerabilità personale, induce erroneamente le persone a ritenere che siano in qualche modo immune da disastri. Non è così.
29. **Scorte di sangue ed emoderivati devono essere inviati nei paesi stranieri colpiti**, ci sono ragioni patologiche e logistiche per le quali è meglio acquisire sangue ed emoderivati da donatori dello stesso paese colpito.

30. **Quando si verificano disastri gli adulti normodotati dovrebbero volontariamente aiutare**, l'era del volontariato spontaneo è finita. I volontari non organizzati portano più problemi di quanti ne risolvino. La risposta è quella di creare, in tempi di pace, associazioni di volontari addestrati ed equipaggiati, che possano essere integrati nel sistema di protezione civile per legge o secondo regole ben definite che definiscono i propri compiti e responsabilità.
31. **Vittime della carestia di solito muoiono di fame**, il più delle volte muoiono a causa di effetti collaterali della carestia, come la malnutrizione, diarrea, il tifo o il colera.
32. **Solo la preparazione porta ad agire**, produrre i mezzi per ridurre il rischio di catastrofi (un sistema di allarme, una mappa di pericolosità, un avanzamento delle tecniche costruttive antisismiche, di un regolamento edilizio aggiornato, ecc), non implica che verranno utilizzati; infatti la mancanza di leadership, cattive intenzioni, paralisi politica e la mancanza di fondi sono alcune possibili ragioni per cui questi mezzi potrebbero non essere utilizzati.

1.4 Il contesto d'uso

Questa tesi si colloca nel contesto di un lavoro di ricerca portato avanti da diversi anni da alcuni ricercatori del dipartimento DISIM.

Il team di ricerca ha condotto studi approfonditi sull'utilizzo dei sistemi ICT a supporto del disaster-management e prototipato diversi sistemi in questo ambito, basandosi anche sull'esperienza diretta del terremoto che colpì L'Aquila nel 2009.

Una delle attività più recenti di questo contesto è relativa ad un sistema che nella prima fase di *Response* (vedi 1.2), possa supportare i soccorritori (e le vittime) nell'esplorazione del territorio colpito da un disastro.

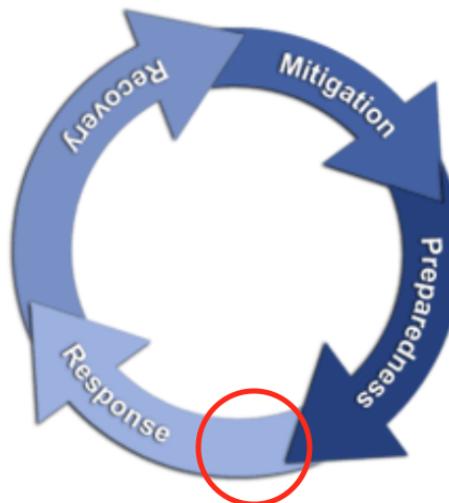


Figura 1.2: Contesto d'uso del sistema all'interno del disaster-management cycle

La soluzione proposta dal team integra l'attività di pianificazione e la **conoscenza dinamica del territorio** all'interno di un sistema multiagent-oriented per poter guidare le attività dei soccorritori (per maggiori dettagli si consiglia la visione del documento: "Disaster Response: a Multi-Agent based Approach" [4]).

La conoscenza dinamica è una caratteristica cruciale per sistemi di questo dominio applicativo; infatti un disastro naturale come il terremoto, può cambiare profondamente la morfologia del territorio rendendo gli strumenti OGB (Only-GPS-Based)

non adatti a fornire soluzioni efficaci. I soccorritori, soprattutto quelli non locali, possono avere molte difficoltà nel raggiungere le vittime.

L’acquisizione dinamica d’informazioni, benché utile alla conoscenza morfologica del territorio, è rilevante soprattutto per il coordinamento generale dei soccorritori, i cui piani d’azione non possono basarsi soltanto sulle considerazioni fatte nella fase di *Preparedness* (vedi 1.2).

Le strategie adottate dipendono dall’analisi preventiva della vulnerabilità (vedi 1.1), che a sua volta, dipende dalla stima del potenziale impatto di un disastro.

E’ chiaro che, in questo contesto, la disponibilità di dati dinamici permetterebbe ai soccorritori di rielaborare tali strategie a seconda del reale impatto. L’utilizzo di un MAS (multy agent system) presenta altri vantaggi, tra questi i più significativi sono: la possibilità di distribuire il carico computazionale e l’aumento della robustezza globale del sistema per mezzo di funzioni avanzate come l’automonitoraggio e l’autodiagnosi.

L’acquisizione dinamica delle informazioni può avvenire tramite droidi, sensori, robot e operatori umani incluse le stesse vittime; quest’ultime infatti utilizzando l’applicazione sviluppata in questa tesi, possono comunicare informazioni molto semplici e dirette tipiche del contesto (esempio ”immediato aiuto qui”) ai soccorritori. L’applicazione mobile risulterebbe uno strumento valido ed utilizzato dalle vittime: in accordo con i miti sfatati da D. Alexander (vedi in particolare i miti 6-8-10 in 1.3), è allora più che ragionevole utilizzare l’enorme quantità di informazioni che possono provenire dai sopravvissuti. Si noti che nelle prime 12-24 ore dopo il disastro le persone tendono, con o senza il sistema, a trasmettersi brevi messaggi significativi come ”sono vivo”, ”la strada è bloccata qui”, ”c’è urgente bisogno di aiuto” ect permettendo la realizzazione di un sistema sia semplice da utilizzare anche da persone con poca familiarità con la tecnologia e anche in condizioni critiche caratterizzanti del contesto, che in grado di limitare, per quanto possibile, il numero di informazioni trasmesse.

In conclusione l’implementazione di tale sistema contribuirebbe ad aumentare la *Resilienza* della comunità (vedi 1.1), ovvero *l’impatto* (vedi l’equazione 1.1) e quindi le vittime.

Capitolo 2

OpenStreetMap

In questo capitolo si illustra brevemente il progetto OpenStreetMap, vengono sollevate alcune questioni etiche nella scelta di un map-provider e infine si espongono i motivi tecnici che hanno portato a scegliere OSM come map-provider per l'applicazione sviluppata nell'ambito di questa tesi.

2.1 Cos'è OpenStreetMap ?

OpenStreetMap (OSM) è **un progetto cartografico collaborativo, nato per creare una mappa mondiale gratuita e libera**. Gli utenti iscritti possono visualizzare, aggiungere e modificare in ogni momento la mappa, secondo un approccio analogo a quello di Wikipedia. I dati infatti sono distribuiti sotto la licenza ODbL[5]: *"Sei libero di copiare, distribuire, trasmettere e adattare i nostri dati, finché lo attribuiisci a OpenStreetMap e ai suoi contributori. Se alteri o ti basi sui nostri dati, puoi distribuire il risultato sotto la stessa licenza [...]"*.



Figura 2.1: Il logo di OpenStreetMap

Si potrebbe erroneamente pensare che l'esistenza di una mappa gratuita non sia una novità; in realtà i dati geografici [6], nella gran parte del mondo (Italia ed Europa incluse), non sono gratuiti. In linea di massima l'onere della realizzazione di mappe è delegata ad agenzie nazionali che poi le rivendono a privati o aziende e ne ricavano finanziamenti. Gli Stati Uniti d'America sono l'unica eccezione macroscopica: qui, infatti, le leggi sul copyright delle agenzie nazionali rendono questi dati di pubblico dominio.

In altre parole se si vive in uno di questi paesi, si pagano le tasse perché vengano realizzate le mappe e si paga nuovamente per avere copie di esse o meglio "fotocopie". Di fatto si tratta di vere e proprie fotocopie poiché non possono essere modificate, sebbene spesso le mappe contengano errori intenzionali (chiamati in gergo easter eggs). Si tratta solitamente di strade inesistenti o mancanti, oppure indicazione di edifici che in realtà non esistono. Se si tenta di realizzare una mappa partendo da questi dati, le ditte od enti che le hanno realizzate potranno dire di avervi beccato semplicemente controllando se sono presenti i loro errori.

La comunità di OSM è in forte crescita: dal 2004, anno in cui nasce da un'idea di Steve Coast ,ad oggi, il numero di utenti iscritti e i loro contributi sono cresciuti anno dopo anno, come illustrato in Figura 1.2 .

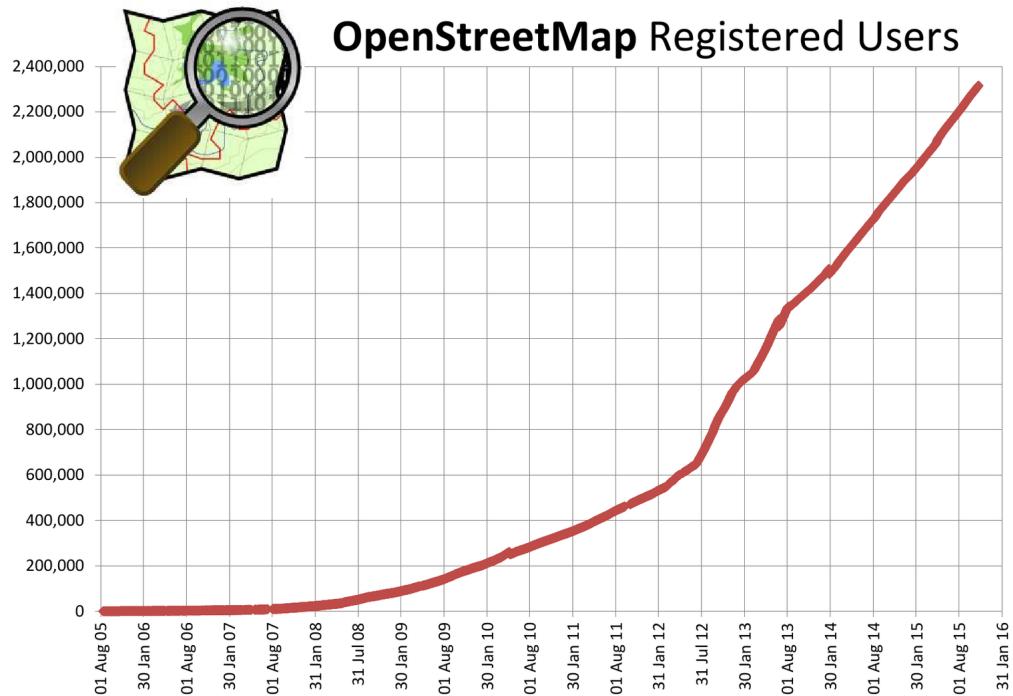


Figura 2.2: Scala lineare di utenti iscritti ad OpenStreetMap

Nel prossimo paragrafo si illustreranno meglio i motivi dell'esistenza di un progetto come OSM.

2.2 Perchè il mondo ha bisogno di OSM

Finora quello di OSM sembrerebbe un altro progetto di nobile causa ma fine a se stesso. Per comprendere meglio la sua importanza bisogna fare un passo indietro nella storia e tornare nel 1800 [7] .

In quel periodo uno dei tanti problemi era costituito dal tempo, non in termini di tempo a disposizione, ma di che ora fosse. Gli orologi esistevano già, ma ogni città aveva il suo "tempo locale". Viene da sé come anche prendere un banale appuntamento con l'amico del paese vicino, comportasse una grande difficoltà. Successivamente l'adozione di uno standard comune ha reso il tempo **universale, libero e di tutti**. L'equivalente attuale del dilemma del tempo è la posizione geografica, e diversi soggetti stanno cercando di diventare il riferimento assoluto (Google spende un miliardo di dollari l'anno per mantenere le proprie mappe). Dunque perché il mondo ha bisogno di OSM? La risposta è semplice, perché in una società nessuna azienda dovrebbe avere il monopolio su qualcosa. I luoghi sono un bene comune, e dando ad una o poche entità questo potere gli viene dato non solo il potere di dirti la tua posizione, ma anche di poterla manipolare. Ci sono tre aspetti da analizzare:

- Cosa viene visualizzato
- Dove dovresti andare
- Privacy personale

Cosa viene visualizzato: Chi decide cosa debba essere visualizzato su una mappa di Google? Ovviamente Google. Se pensiamo ad un servizio pubblico, questo deve essere il più imparziale e trasparente possibile, come potrebbe esserlo se utilizza un mappa di Google?.

Il punto è, nel momento in cui si sceglie un provider di mappe, gli viene dato il potere di decidere quali siano gli elementi a cui dare risalto, o quali non debbano essere proprio mostrati.

Dove dovresti andare: La seconda questione riguarda il posizionamento. Chi decide cosa sia più vicino a me? Ancora una volta la risposta è banale. Non è un caso che scrivendo su Google Maps la parola "*colazione*" vengano mostrate le grandi catene come McDonald's piuttosto che il bar sotto casa.

Nell'immagine successiva viene mostrato uno screenshot della ricerca "*colazione*"

effettuata su Google Maps all'interno della biblioteca di scienze umane dell'università dell'Aquila.

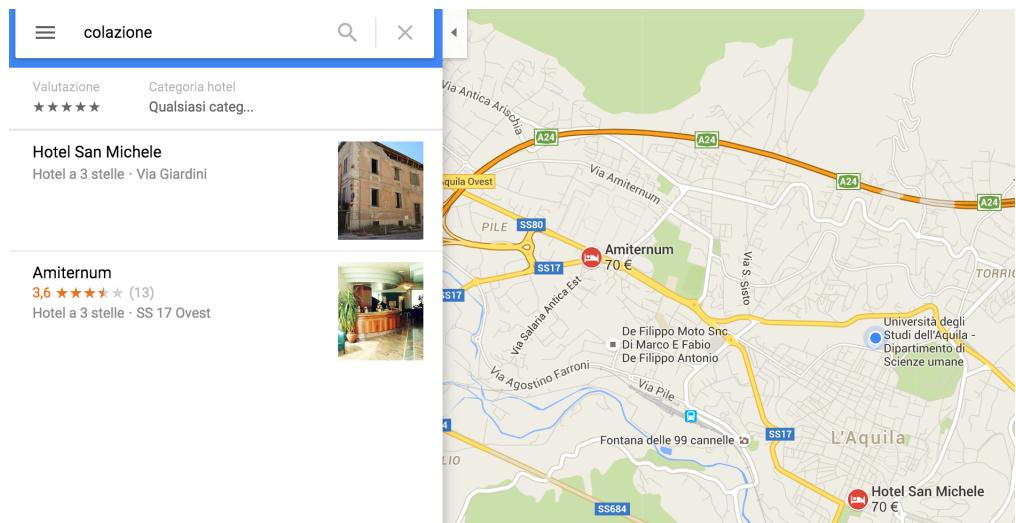


Figura 2.3: Risultato ricerca "colazione"

Come possano notare vengono mostrati soltanto due hotel locali, che probabilmente hanno pagato il provider, nonostante ci sia un piccolo bar appena fuori la struttura.

Privacy personale: L'ultima questione riguarda la privacy. Sia Google che Apple raccolgono una quantità smisurata di informazioni sulla posizione degli utenti che utilizzano le loro API. E' evidente che non si possono ignorare le implicazioni sociali che comporta la disponibilità di così tanti dati in mano ad una singola azienda, indipendentemente da quanto si dichiari benevola. Aziende come Foursquare utilizzano il mezzo della "gamification" per coprire quello che di fatto è un'opera di acquisizione di dati, e anche Google è entrata nella partita della "gamification" con *Ingress*, un gioco che sovrappone un mondo virtuale a quello reale e porta gli utenti a raccogliere foto e informazioni stradali con l'obiettivo di combattere, o favorire, un'invasione aliena.

2.3 OpenStreetMap vs Google Maps

Tralasciando le questioni etiche e al fine della realizzazione dell'applicazione, sono stati analizzati i seguenti topic per la scelta del map-provider:

- Accuratezza
- Costo e Download
- Scalabilità

Accuratezza: Poiché le mappe fornite da OSM sono il frutto di lavori "amatoriali", si è indotti a pensare che queste non rispecchino la realtà. Come in ogni progetto in stile wiki, non c'è alcuna garanzia riguardo l'accuratezza dei dati. C'è da dire, però, che quasi nessuna mappa "commerciale" dà alcuna garanzia di accuratezza. In fondo, gli errori intenzionali, sono per l'appunto, errori. [8]

L'essenza stessa dei processi in stile wiki è che gli utenti stessi, tutti gli utenti, hanno un ruolo nell'accuratezza dei contenuti. Se qualcuno dovesse inserire dati errati, per errore o con intenzione, tutti gli altri possono accorgersene e correggere l'errore o semplicemente eliminarlo. La presenza di una larghissima maggioranza di utenti benintenzionati garantisce che gli errori restino entro un limite accettabile.

L'esperienza degli altri progetti basati su wiki ci insegna, comunque, quanto sia agevole raccogliere dati di buona/ottima qualità e quanto, invece, possa essere complicato scovare gli inevitabili errori. Attualmente non sono stati realizzati processi o meccanismi che rendano semplice questo genere di controllo, tuttavia una comunità attiva come quella di OSM garantisce un controllo qualità sufficiente.

La domanda che dobbiamo porci è chi meglio di noi conosce il quartiere dove abitiamo? O la strada che percorriamo tutti i giorni o il parco dove portiamo il nostro animale a passeggiare? La risposta è **NOI**. Ed è proprio questo sottile concetto a fare la differenza tra una mappa OSM e una mappa proprietaria.

Utilizzando uno dei tanti "map-compare" sulla rete possiamo fare degli esempi concreti. Nella Figura 1.4 abbiamo una duplice visualizzazione dell'area universitaria di Coppito (L'Aquila): a sinistra vediamo la mappa ottenuta tramite dati OSM mentre a destra la stessa mappa fornita da Google.

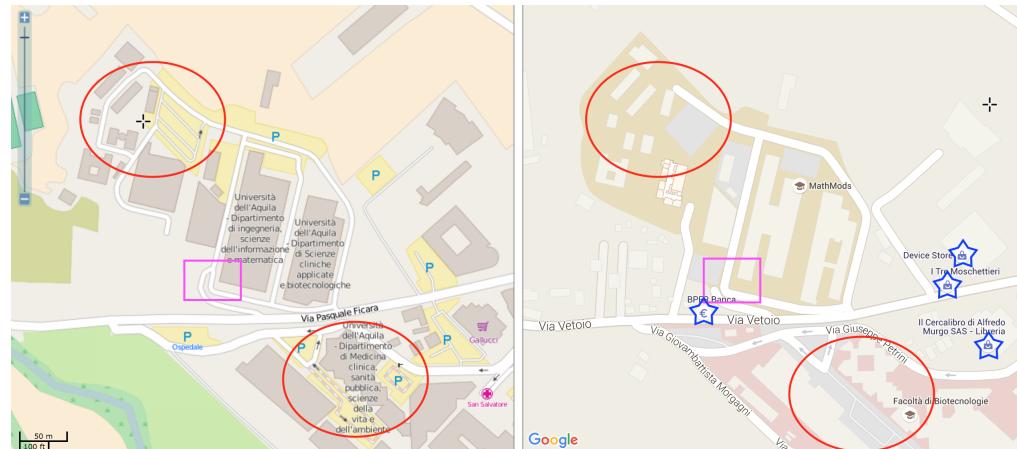


Figura 2.4: Polo universitario di Coppito-L'Aquila

Soffermiamoci quindi soltanto sulle porzioni di mappa racchiuse nelle diverse figure geometriche:

- **Cerchi rossi:** notiamo come nella mappa di Google siano assenti diverse strade interne alla struttura che collegano i diversi edifici tra di loro; questi sono percorribili da un veicolo d'emergenza o più semplicemente un'autovettura.
- **Rettangolo viola:** Presenza di un errore (possibile easter eggs) da parte di Google: la strada non si interrompe in quel modo
- **Stelle blu:** Presenti solo nella mappa di Google, indicano tutte attività commerciali, sarà un caso?

Infine osservando la mappa fornita da Google non si percepisce di avere davanti una grande struttura universitaria, mentre sulla mappa OSM sono indicati perfino i nomi dei dipartimenti presenti nei diversi edifici.

Costo e download: Volendo realizzare un'applicazione per dispositivi mobili completamente gratuita, priva di pubblicità o di qualsiasi altra forma di lucro, di questo fattore non si può non tenere conto. Per quanto riguarda Google, l'API javascript è gratis fino ad un massimo di venticinquemila richieste giornaliere per novanta giorni consecutivi. Superata questa soglia si ha un costo di \$ 0,50 ogni mille richieste [9].

Oltre al costo vengono applicate le seguenti restrizioni:

- **Area limitata:** è possibile scaricare una porzione di mappa la cui dimensione massima non superi i centoventimila chilometri quadrati [10].
- **Scadenza:** le mappe saranno disponibili in assenza di rete per un totale di 30 giorni, dopodiché si dovrà effettuare l'accesso alla rete.

Per quanto riguarda OSM, essendo un progetto open-data, non vi è alcuna restrizione. Infatti, è possibile scaricare l'intera mappa del mondo o porzioni di essa in totale libertà e conservarle a tempo indeterminato [11].

Scalabilità: Quest'ultimo topic ha segnato di fatto il punto decisivo per la scelta di OSM come map-provider. L'associazione no-profit HOT (Humanitarian Open-streetmap Team), ha lo scopo di fornire un valido supporto sul campo al mapping di aree colpite da disastri ambientali e non.

Nel sito ufficiale si possono visualizzare i rapporti delle principali emergenze a cui l'associazione ha partecipato [12] , consideriamo quindi il terremoto di Haiti.

Il 12 gennaio 2010 un violento terremoto di magnitudo 7.0 Mw (quello che colpì L'Aquila nel 2009 fu di 6.3 Mw) con epicentro localizzato a circa 25 chilometri in direzione ovest-sud-ovest della città di Port-au-Prince, capitale dello Stato caraibico di Haiti, colpì tutta l'area circostante.

Il numero di vittime è stato stimato al 24 febbraio 2010 in 222.517. Secondo la Croce Rossa Internazionale e l'ONU, il terremoto avrebbe coinvolto più di 3 milioni di persone [13].

Sei ore dopo il disastro la mappa disponibile di Port-au-Prince era la seguente:

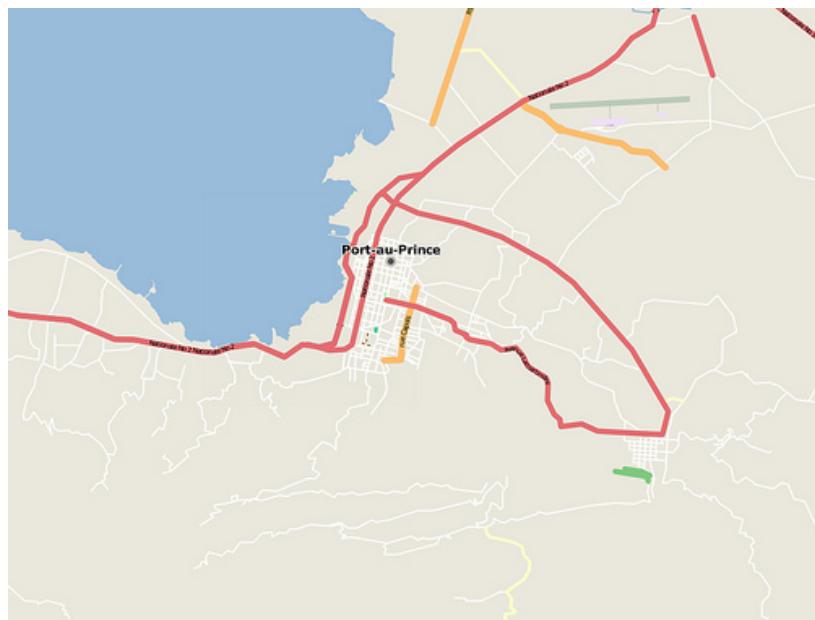


Figura 2.5: Mappa OSM di Pourt-au-Prince 6 ore dopo il terremoto del 2010

Nelle ore successive, numerose aziende di geo-data (Geo-eye, Google, Yahoo...) resero pubbliche le proprie immagini satellitari, in modo tale che i volontari potessero mappare l'area colpita dal proprio pc in qualsiasi parte del mondo.

Quello che avvenne fu qualcosa di straordinario, come disse Jeffery Johnson *"What we did in Haiti changed disaster response forever"*.

Il contributo di più di seicento volontari portò ad avere dopo appena 48h dal disastro la seguente mappa:

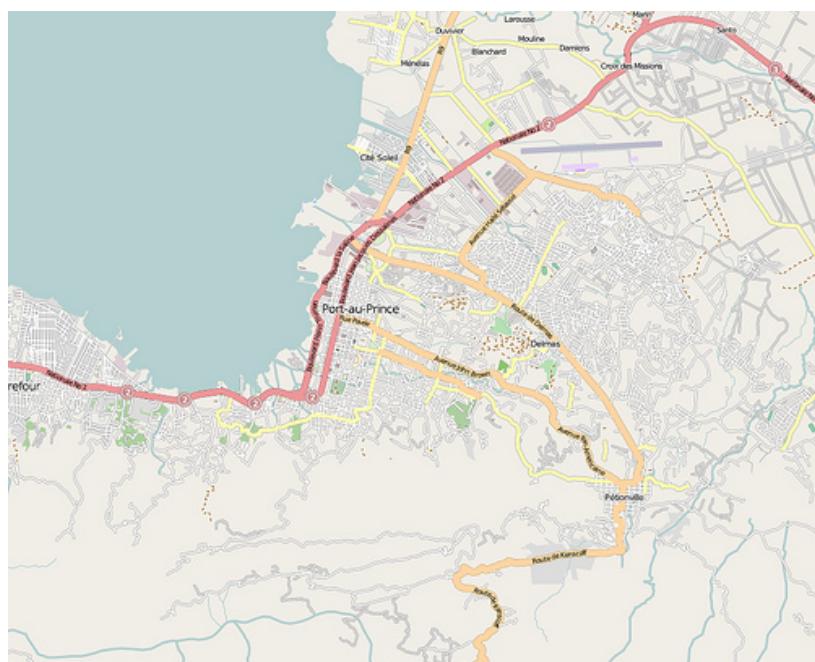


Figura 2.6: Mappa OSM di Pout-au-Prince 48 ore dopo il terremoto del 2010

La mappa nelle settimane successive continuò ad essere aggiornata e raffinata, tutto questo ha facilitato il coordinamento delle operazioni umanitarie di soccorso e approvvigionamento, salvando di fatto innumerevoli vite.

In conclusione OSM garantisce un'ottima scalabilità, caratteristica che come abbiamo visto, nei contesti di disaster-management è di fondamentale importanza.

Capitolo 3

L'applicazione lato utente

Nella prima parte di questo capitolo, vengono illustrati i requisiti di sistema che hanno guidato lo sviluppo dell'intera applicazione. Mentre nella parte finale verranno mostrati i task principali e le relative schermate dell'interfaccia realizzata con il framework Ratchet (vedi 4.3).

3.1 I concetti principali

La progettazione dell'applicazione è stata guidata, oltre allo studio di numerosi documenti riguardanti il contesto d'uso, dalla definizione di alcuni concetti chiave.

Nel contesto d'uso, le persone sono interessate a conoscere lo stato dei propri cari e di alcuni luoghi d'interesse. Questa osservazione è stata implementata nei concetti di Friends Of Interest e Point Of Interest; i FOI e i POI rappresentano quindi l'insieme delle persone e dei luoghi per i quali l'utente ritiene rilevante conoscere lo stato in uno scenario di disastro ambientale e/o umano.

Un'altra questione importante è l'attendibilità delle informazioni mostrate; per questo motivo ogni stato, che sia un FOI o un POI, è correllato dalla data e l'ora in cui è stato "prodotto". Per i POI l'attendibilità è rafforzata dal numero di segnalazioni di tale emergenza; infatti dato che chiunque, tramite una procedura standard, potrebbe segnalare un'emergenza in quello specifico luogo ha senso considerarlo come un buon indice di attendibilità dell'informazione, senza trascurare però quei luoghi (come le case indipendenti) che avranno un numero relativamente più basso di segnalazioni.

Per i FOI oltre allo stato, si è pensato di utilizzare una sorta di "ping"; il dispositivo perdiadicamente e in base al numero degli utenti intorno a lui, invia un "bit" al

sistema centrale, se allo scadere di un certo timeout, non si hanno notizie del dispositivo verrà considerato come spento. In questo modo gli utenti possono sapere se il dispositivo di un loro caro è spento, acceso o se egli stia usando l'applicazione.

3.2 Requisiti di sistema

La stesura dei requisiti di sistema è un processo imprensindibile per la progettazione, nel nostro caso è il risultato di un processo iterativo basato sulla valutazione di un esperto.

I requisiti sono divisi in:

- **Requisiti funzionali** indicano quello che il sistema deve fare
- **Requisiti non funzionali** vincoli sul sistema e il suo sviluppo

3.2.1 Requisiti funzionali

1. Interazione mappa

- *Identificativo:* RF-1
- *Descrizione:* Il sistema deve permettere all'utente di interagire con la mappa, compiendo le azioni basilari quali: zoom In, zoom Out, CCW (Change Center View), click.

2. Impostazione POI

- *Identificativo:* RF-2
- *Descrizione:* Il sistema deve permettere all'utente di impostare una specifica porzione di mappa come un POI (point of interest).
- *Razionale:* In questo modo l'utente può applicare un filtro sulla mappa (vedi RF-7) e visualizzare rapidamente lo status dei luoghi d'interesse.

3. Impostazione FOI

- *Identificativo:* RF-3
- *Descrizione:* Il sistema deve permettere all'utente di impostare altri utenti del sistema come FOI (Friends Of Interest).
- *Razionale:* l'utente può in questo modo applicare un filtro (vedi RF-7) e visualizzare in modo rapido lo status delle persone d'interesse.

4. Segnalazione evento

- *Identificativo:* RF-4
- *Descrizione:* l'utente deve poter segnalare la posizione di un certo evento (vedi RNF-1). La procedura standard di segnalazione deve avvenire sia cliccando su un punto della mappa sia tramite una schermata dedicata. Nel caso di rete congestionata o assente il sistema deve provvedere alla bufferizzazione delle richieste e avvisare di tale situazione l'utente stesso.
- *Razionale:* In questo modo gli utenti contribuiscono all'aggiornamento dello status generale del territorio colpito.

5. Aggiornamento status

- *Identificativo:* RF-5
- *Descrizione:* L'utente può cambiare il suo status (vedi RNF-4). Il sistema quindi deve comunicare immediatamente al server tale aggiornamento.
- *Razionale:* In questo modo gli utenti contribuiscono a fornire informazioni dinamiche sul territorio colpito e su se stessi.

6. Trusty data

- *Identificativo:* RF-6
- *Descrizione:* Il sistema deve informare l'utente sul grado di aggiornamento delle informazioni visualizzate, ovvero:
 - Eventi
 - Status dei POI
 - Status dei FOI
 - mappe offline

L'attendibilità degli eventi è data dal numero di segnalazioni di tale evento nella relativa cella (vedi RNF-1) e dall'orario in cui è stata generata l'ultima segnalazione.

- *Razionale:* Nel contesto d'uso, la rete potrebbe collassare o più semplicemente gli utenti potrebbero non utilizzare il sistema per un certo periodo, in questo modo si garantisce la totale trasparenza delle informazioni fornite.

7. Filtra mappa

- *Identificativo:* RF-7
- *Descrizione:* Il sistema deve permettere all'utente di filtrare le informazioni visibili sulla mappa in base a:
 - propri POI
 - tipo eventi
 - propri FOI
- *Razionale:* La mappa visualizzata dall'utente potrebbe contenere un numero elevato di informazioni.

8. Modalità offline

- *Identificativo:* RF-8
- *Descrizione:* Il sistema deve salvare porzioni di mappa visualizzate dall'utente attraverso l'interazione base (vedi RF-1) nella memoria temporale, inoltre l'utente deve poter scaricare una specifica center view su diversi livelli di zoom. Il sistema quindi deve permettere all'utente

di utilizzare la modalità offline, in questo caso la rete verrà utilizzata solamente per inviare e ricevere aggiornamenti riguardo:

- POI e NPOI
- FOI e NFOI
- Eventi
- *Razionale:* L'utente potrebbe voler utilizzare la modalità offline per risparmiare dati o per la pessima connessione

9. Markercluster

- *identificativo:* RF-9
- *Descrizione:* Il sistema per livelli di zoom, sufficientemente bassi, deve raggruppare i marker in un unico markercluster; inoltre deve mostrare la quantità di elementi inglobati.

10. Aggiornamento dati persistenti

- *identificativo:* RF-10
- *Descrizione:* Il sistema deve periodicamente richiedere al server l'aggiornamento dei dati persistenti (vedi RNF-2). Inoltre l'utente può richiedere l'aggiornamento in qualsiasi momento

11. Salta a

- *Identificativo:* RF-11
- *Descrizione:* Il sistema deve permettere all'utente di spostare la propria center view al NPOI (nearest point of interest), al NFOI (nearest family of interest) o al riferimento di una notifica d'allerta (vedi RF-11) cliccando su di essa.
- *Razionale:* L'utente potrebbe voler prestare soccorso al famigliare o visualizzare lo status del punto d'interesse più vicino a lui.

12. Notifiche di allerta

- *Identificativo:* RF-12
- *Descrizione:* Il sistema deve notificare l'utente sull'aggiornamento dello status di un FOI o della segnalazione di un evento all'interno di un POI.

3.2.2 Requisiti non funzionali

1. Dati griglia

- *Identificativo:* RNF-1
- *Descrizione:* la mappa è divisa da una griglia con celle di 22x16 mt (vedi 4.5.1).
- *Razionale:* Più utenti vicini potrebbero utilizzare il sistema, senza tolleranza una porzione di mappa potrebbe essere saturata dalla visualizzazione di eventi omogenei.

2. Dati persistenza

- *Identificativo:* RNF-2
- *Descrizione:* il sistema deve memorizzare in modo permanente, attraverso un database locale, i seguenti dati:
 - FOI
 - POI
 - eventi visualizzati nella center view
 - porzioni di mappa

3. Dati aggiornabili

- *Identificativo:* RNF-3
- *Descrizione:* I dati memorizzati (vedi RNF-2) sono aggiornabili secondo le modalità precedentemente descritte (vedi RF-9).
- *Razionale:* Le informazioni cambiano dinamicamente, alcune emergenze potrebbero essere state risolte, altre potrebbero nascere successivamente.

4. Dati tipi

- *Identificativo:* RNF-4
- *Descrizione:* L'utente può scegliere il tipo di evento da segnalare (vedi RF-5) tra i seguenti:
 - edificio crollato
 - incendio
 - allagamento
 - strada interrotta
 - persona intrappolata
 - persona ferita
 - persona non autosufficiente

Inoltre può scegliere il suo status, che verrà notificato agli utenti che lo hanno impostato come FOI, tra:

- sto bene
- ferito lieve
- ferito
- intrappolato
- intrappolato e ferito

5. Ambientale tecnico

- *Identificativo:* RNF-5
- *Descrizione:* Il sistema è un'applicazione per dispositivi mobili cross platform, si utilizza il framework phonegap e le tecnologie proprie alla programmazione web:
 - HTML
 - CSS
 - Javascript

E' richiesto l'accesso alla rete internet e l'utilizzo del gps integrato del dispositivo

6. Ambientale sociale

- *Identificativo:* RNF-6
- *Descrizione:* I dati sono prodotti dagli utenti che attraverso il sistema cooperano per arricchire il server di informazioni vitali sul territorio colpito. Essendo quindi un processo distribuito particolare attenzione deve essere posta all'controllo dell'attendibilità degli eventi segnalati per tutti gli utenti (vedi RNF-1)

7. Ambientale fisico

- *Identificativo:* RNF-7
- *Descrizione:* Il sistema potrebbe essere utilizzato in svariati contesti d'uso, di seguito i più comuni:
 - assenza di luce
 - presenza di folla
 - Scarsa visibilità
 - forti raffiche di vento
 - allagamenti
 - incendi
- *Razionale:* Essendo un sistema il cui obiettivo principale è fornire un supporto utile ai soccorritori nel caso di eventi catastrofici, i contesti d'uso dell'applicazione possono essere imprevedibili. Tuttavia si può rispondere al meglio attraverso una buona progettazione del sistema, utilizzando i principi dell'interaction design.

8. Utenti

- *Identificativo:* RNF-8
- *Descrizione:* la fascia di utenti del sistema è molto ampia, non è possibile stabilirne con certezza gli estremi.

9. Usabilità

- *Identificativo:* RNF-9
- *Descrizione:* Il sistema deve essere il più usabile possibile, nel contesto d'uso gli utenti potrebbero essere spaventati o perdere lucidità

3.3 L'interfaccia grafica

Nella realizzazione dell'interfaccia si sono utilizzati pattern mobile ormai consolidati come: le liste per i FOI e i POI, i segment controll per la scelta della loro visualizzazione e la title-bar con icona per tornare alla home.

Le gestures non possono essere complicate, infatti come detto nei requisiti la fascia di utenti è molto ampia e alcuni di essi potrebbero avere poca familiarità nell'utilizzo dei dispositivi mobili. Di fatto l'unica gestures presente è il classico "tap" su display touchscreen, scelta che impone un "comportamento a bottone" di tutti gli elementi grafici.

In questo paragrafo verranno quindi illustrati i task principali e le relative schermate mostrate dall'applicazione.

Dashboard: è la home del sistema, da qui l'utente può accedere a tutte le sue funzionalità. E' costituita da cinque "blocchi logici", il primo include la title-bar (presente in ogni schermata dell'applicazione), il secondo è composto da un container con le informazioni sul FOI più grave, il terzo include i due buttoni per i task "*how are you?*" e "*signal event*", il quarto blocco è composto dall'ultima notizia ufficiale comunicata dai soccorrittori e infine il quinto blocco è costituito da tre buttoni per l'accesso ai FOI, POI e alla mappa.

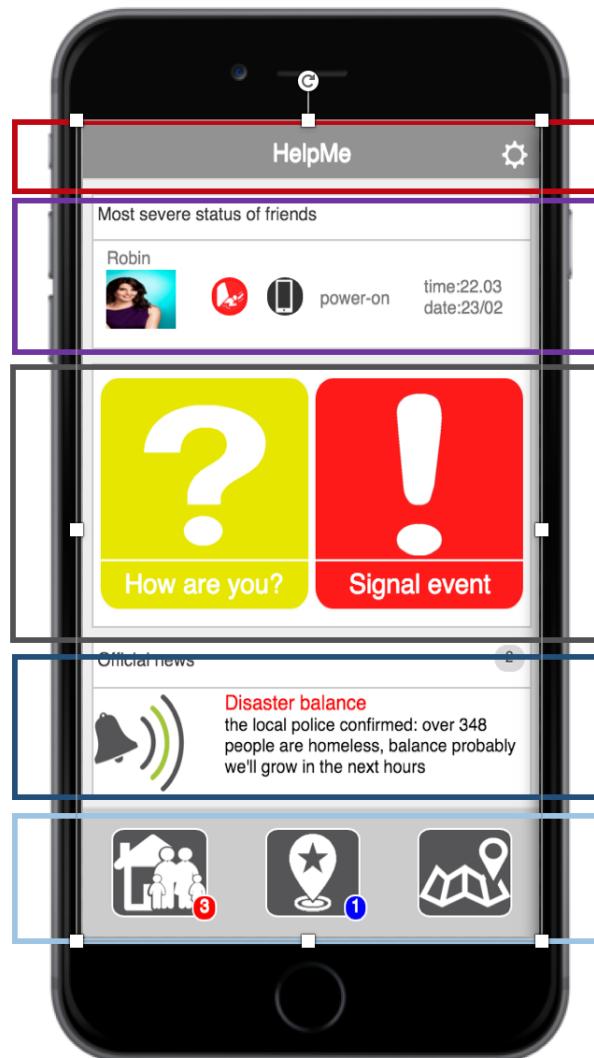


Figura 3.1: Dashboard

How are you?: Per aggiornare il proprio stato bisogna cliccare sul bottone giallo, come in Fig 3.2, quindi tappare su un elemento della lista (Fig 3.3). Per rendere effettivo il nostro aggiornamento basterà cliccare sul tasto in alto a destra: "Publish".

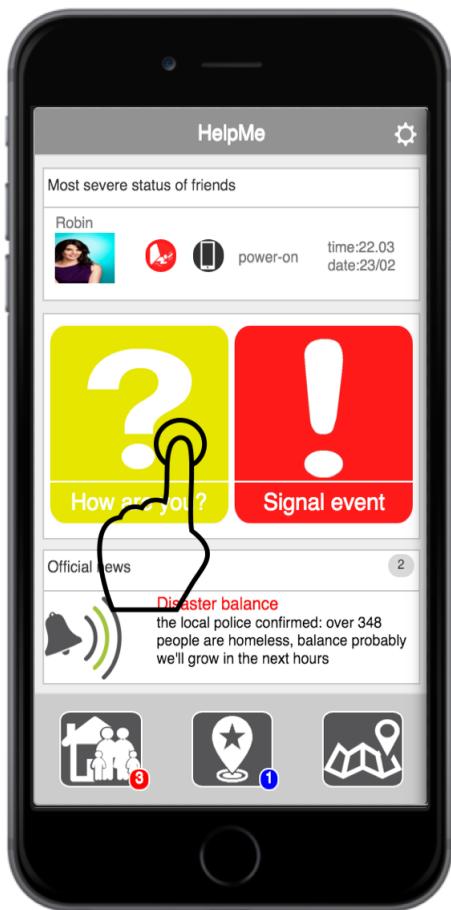


Figura 3.2: Tap bottone "How are you?"

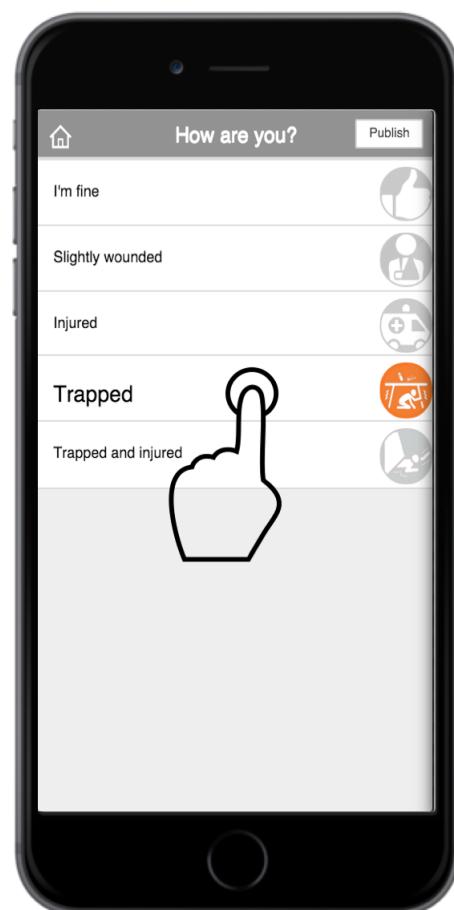


Figura 3.3: Lista predefinita di stati selezionabili

Signal event: Per segnalare un'emergenza bisogna cliccare, nella dashboard, sul bottone rosso come in Fig 3.4. Come per gli stati possiamo scegliere l'emergenza da segnalare da una lista predefinita(Fig 3.5).

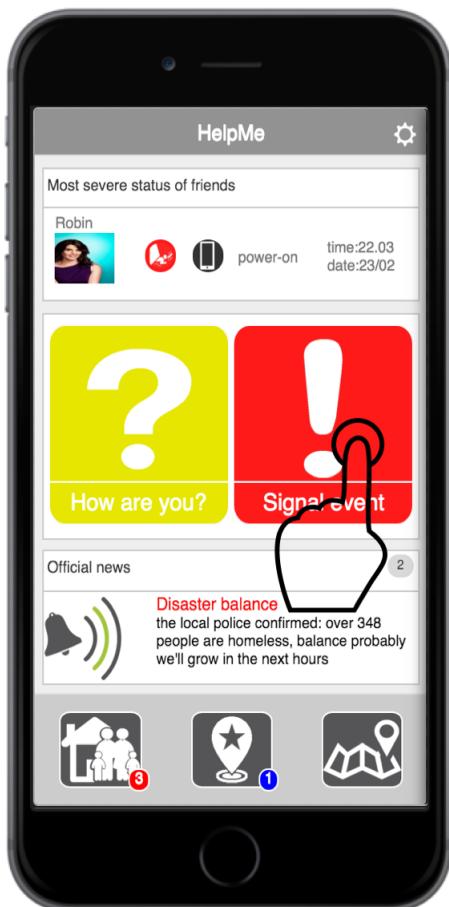


Figura 3.4: Tap del bottone "Signal event"

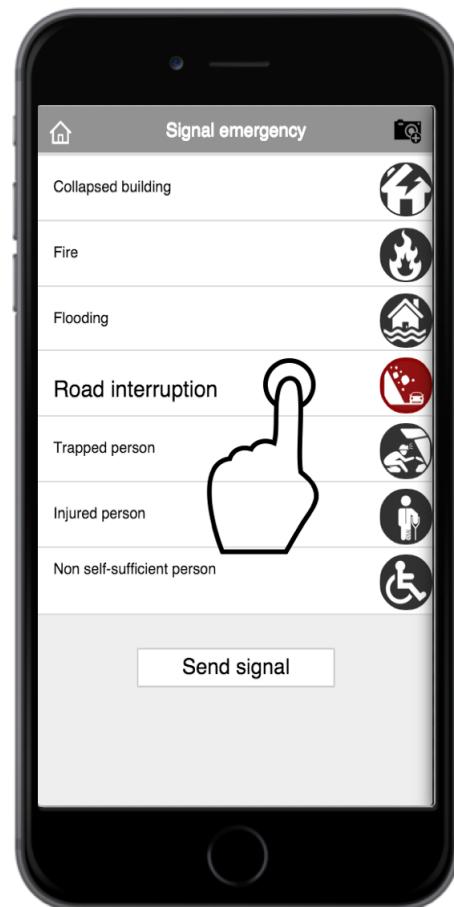


Figura 3.5: Lista predefinita dell'emergenze

Quindi premendo sul bottone "Send signal", avremo la possibilità di usare la nostra posizione oppure di utilizzare la mappa (Fig 3.6).

Where?: se si decide di utilizzare la mappa per segnalare la posizione dell'emergenza, tappando come in Fig 3.6, allora si dovrà cliccare su un punto della mappa che il sistema ci mostrerà (Fig 3.7).



Figura 3.6: Tap "locate on the map"

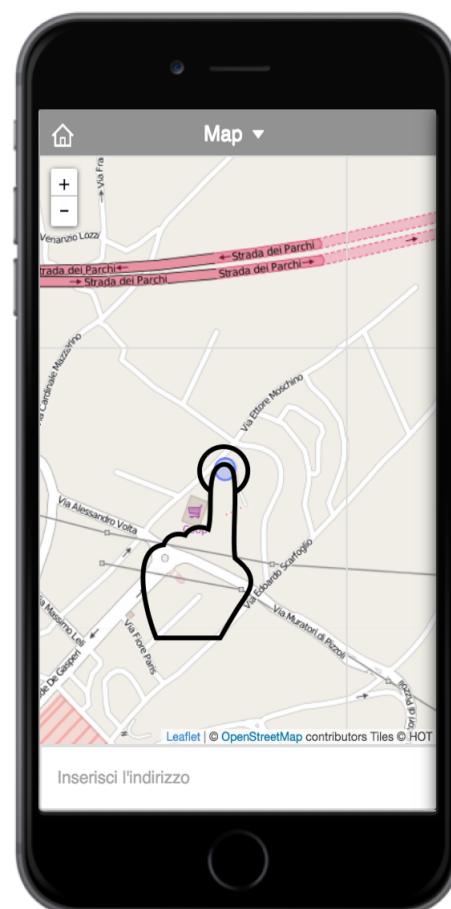


Figura 3.7: Mappa mostrata dal sistema

Per confermare la posizione scelta, basterà infine cliccare sul tasto "Done" e segnalare così l'emergenza.

Visualizza FOI: per visualizzare lo stato dei nostri cari bisogna tappare, nella schermata della dashboard, il bottone in basso a sinistra come in Fig 3.8. La lista sarà ordinata per default in base alla gravità del loro stato.

Come possiamo vedere in Fig 3.9, ogni elemento (FOI) è composto da un'icona del suo stato correlata dall'ora e la data in cui egli ha eseguito l'aggiornamento, lo stato del dispositivo ottenuto mediante un "ping" del sistema centrale e infine dalla distanza tra lui e l'utente dell'applicazione.

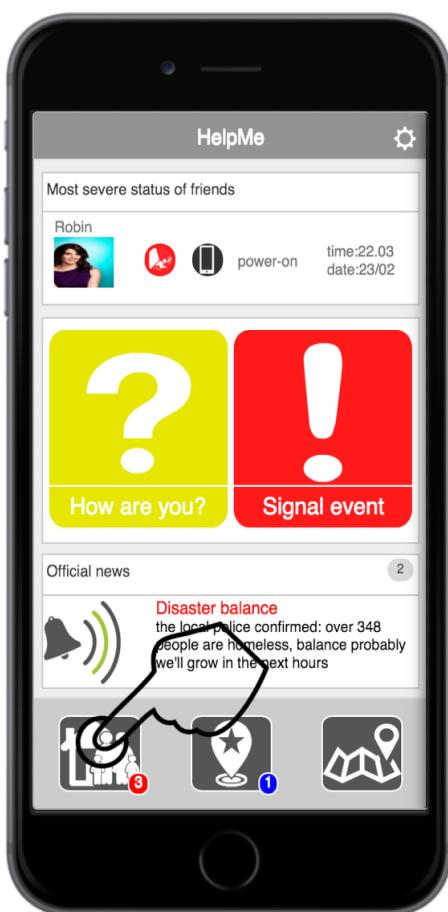


Figura 3.8: Tap da fare per accedere alla lista dei FOI

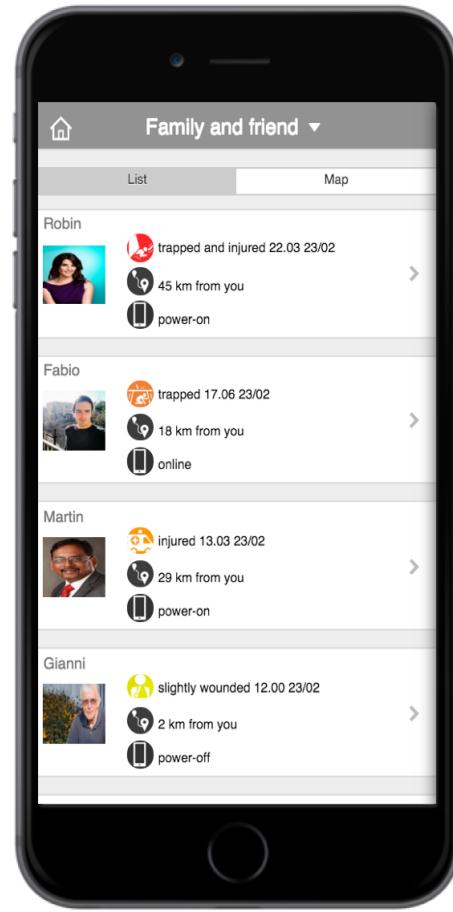


Figura 3.9: Lista dei FOI ordinata per default in base alla gravità

Ordina FOI: come detto, la lista dei nostri FOI sarà ordinata inizialmente in base alla gravità del loro stato, tuttavia è possibile ordinarla anche a seconda della vicinanza o del più recente aggiornamento semplicemente cliccando prima sull'icona del menu a comparsa, Fig 3.10, poi sulla relativa etichetta (Fig 3.11).

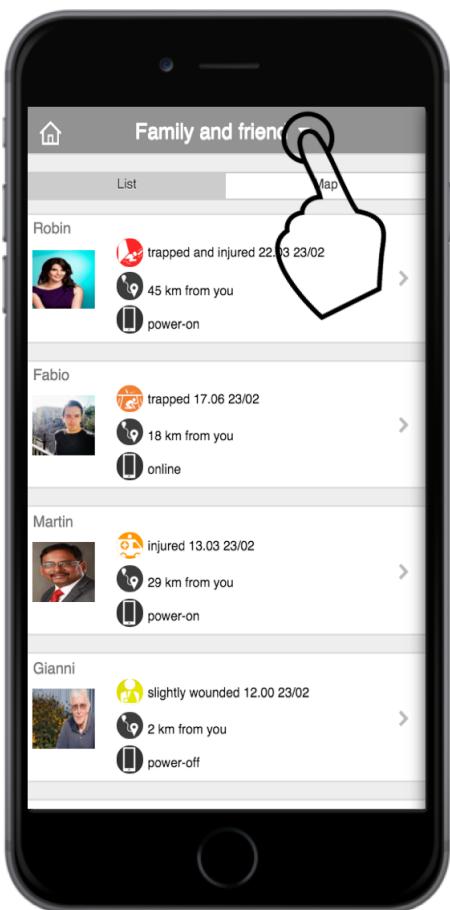


Figura 3.10: Tap per aprire il menu a comparsa

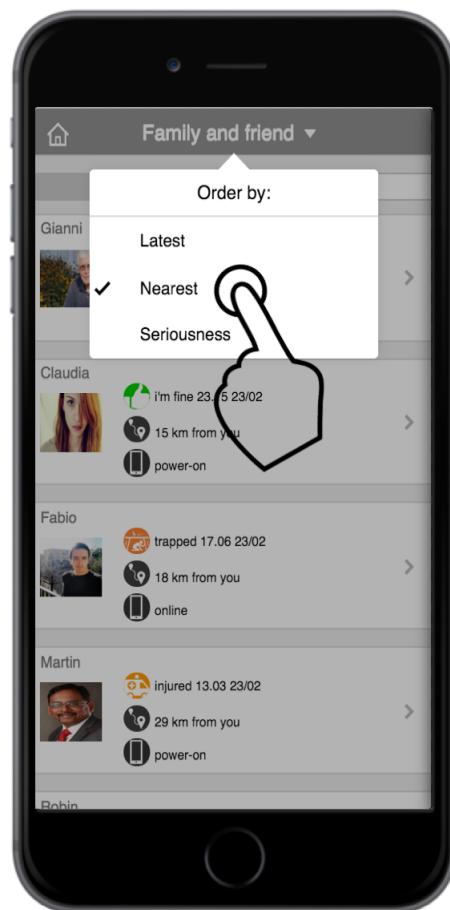


Figura 3.11: Tap da fare per ordinare la lista secondo i tre parametri

Visualizza FOI sulla mappa: oltre alla lista, è possibile visualizzare i propri FOI sulla mappa. Nella schermata precedente basta cliccare sul bottone "Map", come in Fig 3.12. La mappa verrà quindi centrata sul primo FOI della lista, nell'esempio di Fig 3.13 la mappa era stata ordinata per "vicinanza".

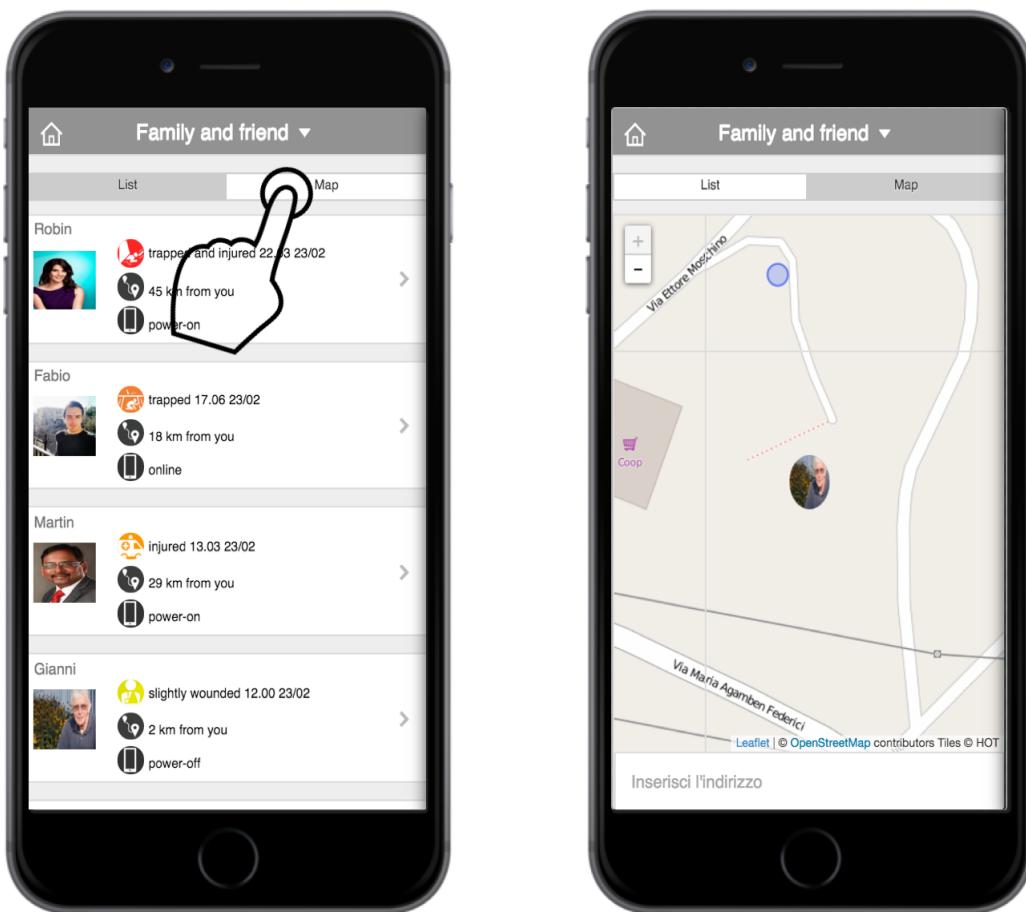


Figura 3.12: Tap per visualizzare i FOI sulla mappa

Figura 3.13: Mappa centrata sul FOI più vicino

E' possibile comunque, con le stesse modalità della lista, centrare la visuale della mappa in base al FOI più vicino, più grave o che abbia aggiornato più recentemente il suo stato

Clustermarker FOI: come specificato nei requisiti, i marker dei FOI (e dei POI) devono essere raggruppati quando lo zoom della mappa è tale da farli "toccare" (Fig 3.14). Inoltre cliccando il clustermarker è possibile vedere da quali FOI è composto (Fig 3.15).



Figura 3.14: Due markercluster contenenti rispettivamente due e tre FOI



Figura 3.15: Tap su un markercluster contenente cinque FOI

POI: le modalità di interazione e le schermate sono analoghe a quelle viste per i FOI. Per accedere alla lista (Fig 3.19), bisogna cliccare sul bottone in fondo alla dashboard, come mostrato in Fig 3.16.

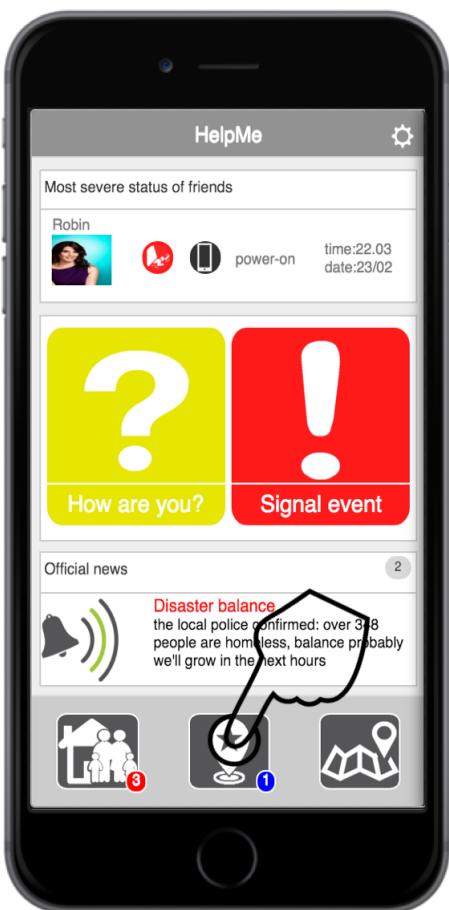


Figura 3.16: Tap da fare per accedere ai propri POI

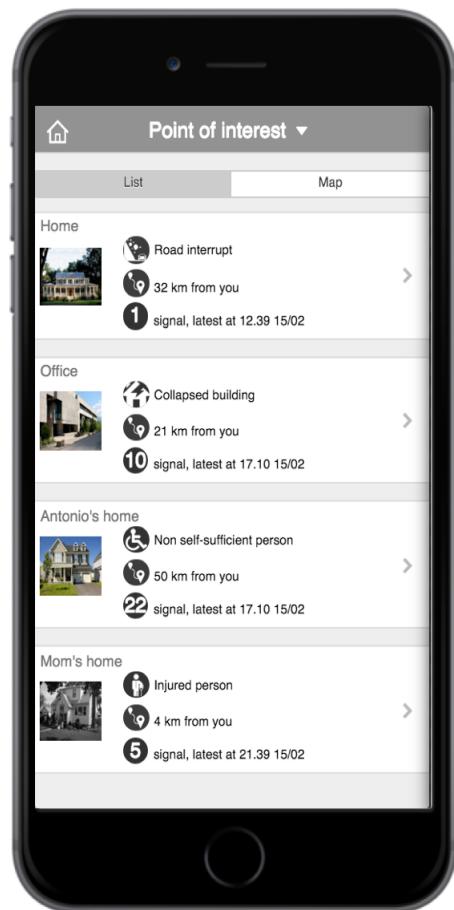


Figura 3.17: Lista dei POI ordinata per default in base al più recente

POI: Tappando su un marker, indipendentemente che sia un FOI o un POI, è possibile visualizzare le informazioni principali (Fig 3.18).

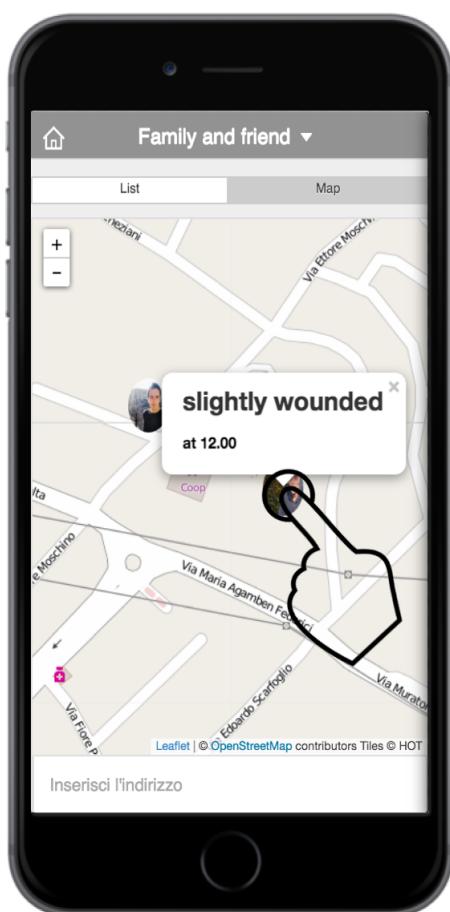


Figura 3.18: Tap sul marker di un FOI

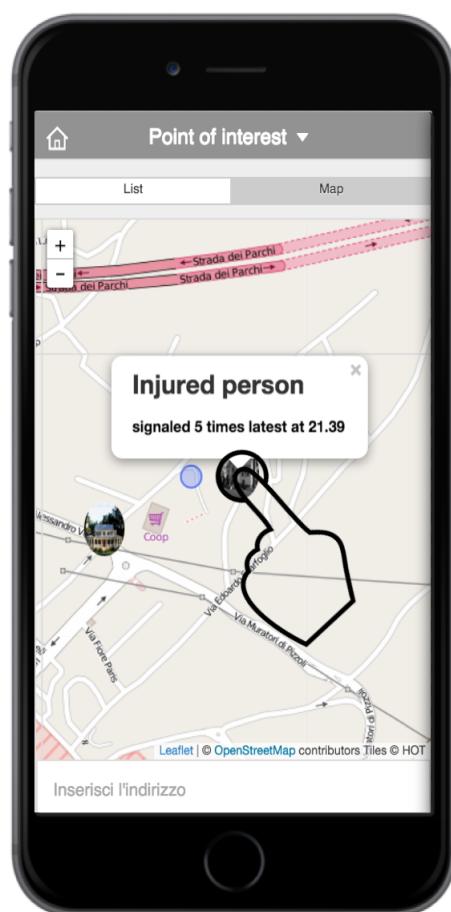


Figura 3.19: Tap sul marker di un POI

Capitolo 4

L'implementazione

In questo capitolo verranno illustrati i framework utilizzati per l'implementazione dell'applicazione e l'algoritmo ideato per la creazione di una griglia sulla mappa con il relativo codice Javascript.

4.1 Le applicazioni cross-platform

Negli ultimi anni, i dispositivi mobili sono diventati sempre più parte integrante della nostra vita, grazie al progresso tecnologico e all'abbattimento dei prezzi, oggi costituiscono un bene alla portata di tutti.

La possibilità di scegliere tra una vasta gamma di smartphone diversi per caratteristiche e produttore, ha reso più complicata la vita delle aziende informatiche che sono costrette a considerare l'eterogeneità dei vari sistemi operativi in circolazione.

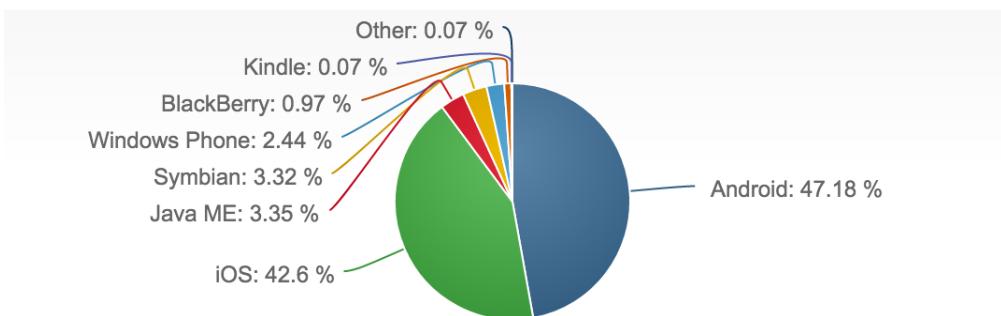


Figura 4.1: Utilizzo dei vari OS su scala mondiale

Chi sviluppa applicazioni mobili, può scegliere tra due approcci:

1. **Scegliere uno degli OS mobile** e implementare l'applicazione utilizzando il linguaggio nativo.
2. **Scegliere un framework** che, utilizzando un meta-linguaggio, sia in grado di generare diverse versioni dell'applicazione per i relativi OS.

I diversi approcci forniscono rispettivamente pro e contro. Per il primo i vantaggi sono una maggiore velocità del sistema, la possibilità di creare un'interfaccia rispettando il look and feel nativo e pochi problemi di compatibilità. Lo svantaggio è invece legato alla **bassa portabilità**, l'applicazione andrà riscritta completamente per dispositivi con diversi OS.

Il principale vantaggio del secondo approccio, è invece la possibilità di riutilizzare lo stesso codice per generare varie versioni dell'applicazione per diversi OS, questo a discapito di una minore velocità del sistema se l'applicazione fosse sviluppata nel linguaggio nativo.

Considerando il contesto d'uso del sistema (vedi 1.4), è chiaro che la nostra applicazione dovrebbe poter essere installata su qualsiasi dispositivo (ndr l'idea di salvare vite con un certo OS non è delle più nobili), di conseguenza si è adottato tale approccio.

La filosofia "*Write once, run anywhere*", non è un concetto nuovo nell'informatica, lo slogan fu ideato dalla Sun Microsystems per descrivere un linguaggio (Java) in grado di essere eseguito su diverse macchine. Nel corso degli anni diverse software-house hanno realizzato dei framework in grado di fare questa "magia", sostanzialmente ne esistono due tipi:

- **I Cross-Compiling:** si scrive l'applicazione in un certo linguaggio, successivamente un framework riesce a compilarlo per le diverse piattaforme (Appcelerator Titanium1, Corona SDK2, Xamarin Monotouch3); Solitamente sono framework professionali e a pagamento.
- **I Browser-Embedding:** più recenti, consentono di sviluppare il codice con le tecnologie del web (HTML5, CSS3, Javascript), l'applicazione verrà avviata nel dispositivo mobile all'interno di un browser (PhoneGap, Apache Cordova).

Ancora una volta è stato scelto il secondo approccio, nello specifico il framework PhoneGap.

4.2 PhoneGap

Phonegap è un framework di sviluppo mobile prodotto da Nitobi e acquistato successivamente da Adobe Systems. Permette ai programmati software di creare applicazioni per dispositivi mobili utilizzando esclusivamente HMTL, CSS e Javascript invece di utilizzare linguaggi nativi come Object-C e C++. Il risultato sono applicazioni ibride, nel senso che non sono né del tutto native (perché tutto il rendering del layout è fatto tramite viste web e non tramite i framework UI delle piattaforme native) né del tutto web-application (perché non sono applicazioni solo web, ma hanno anche accesso alle funzioni interne dei device tramite le API).

Il software alla base di PhoneGap è *Apache Cordova* ed è open source.



Figura 4.2: Logo del framework PhoneGap

Presentato la prima volta in un evento iPhoneDevCamp a San Francisco, PhoneGap ha vinto il premio People’s Choice Award alla conferenza O'Reilly Media Web nel 2009. Il framewotk PhoneGap viene utilizzato da molte piattaforme di sviluppo software come ViziApps, Worklight, Convertico e appMobi.

Il 4 ottobre 2011 Adobe annuncia ufficialmente l’acquisizione della Nitobi Software. In concomitanza il codice PhoneGap ha contribuito con la Apache Software Foundation ad iniziare un nuovo preocetto chiamato Apache Cordova.

PhoneGap non è altro che un modulo software che permette agli sviluppatori di incorporare le loro applicazioni web all'interno di applicazioni native di diverse piattaforme. Le applicazioni che utilizzano questo framework sono scritte in HTML5, CSS3 e Javascript e il collegamento con le librerie native, specifiche di ogni piattaforma, è fatto tramite le API che mette a disposizione il framework. PhoneGap fa quindi da ponte tra il sistema operativo e la web application realizzata dallo sviluppatore. Indipendentemente dalla piattaforma sottostante esisterà un modo per invocare le API native mediante funzioni Javascript.

Il programmatore quindi utilizzerà le API fornite da PhoneGap per accedere alle risorse hardware e software del device in modo da aggiungere le funzionalità di base al motore Javascript e renderle facilmente utilizzabili come se fossero vere e proprie metodi di una libreria. Dopo la compilazione, che avviene tramite il servizio cloud **PhoneGap Build** (Figura 4.3), viene generato un pacchetto internamente composto da due elementi principali con differenti responsabilità che però cooperano tra loro per fornire delle funzioni a valore aggiunto. In pratica esiste un runtime basato su WebKit4 in cui vengono iniettate le componenti statiche; il risultato sarà un pacchetto composta da due elementi principali con differenti responsabilità che però cooperano tra loro per fornire delle funzioni a valore aggiunto. Nel caso specifico il runtime si occupa di dialogare direttamente con il dispositivo e le parti statiche offrono l'interfaccia verso l'utente. L'uso di Javascript e Ajax consente poi di rendere le applicazioni più complesse e dinamiche.

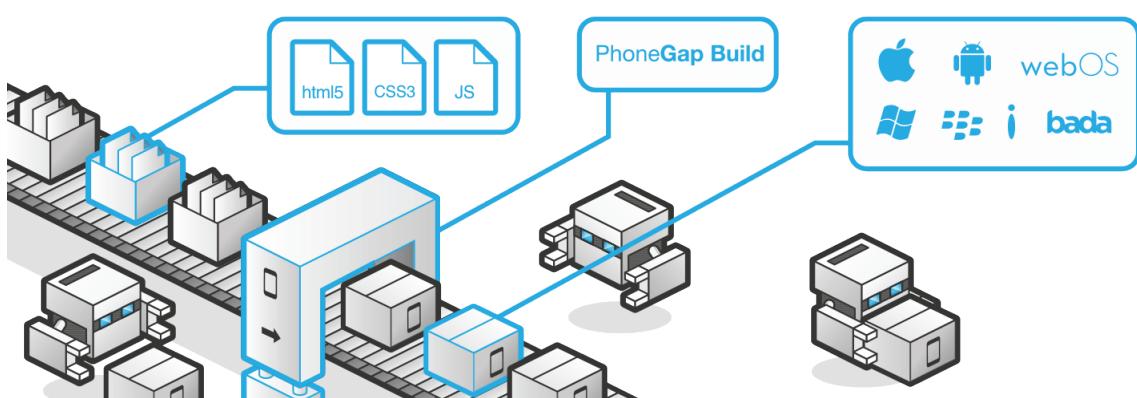


Figura 4.3: Logo del framework PhoneGap

Ora illustriamo mediante la Figura 4.4 come è strutturata l'architettura PhoneGap. Partendo dal basso verso l'alto, si può notare che la parte in blu è quella del sistema operativo della piattaforma nativa che si trova sui vari device. Al livello immediatamente successivo troviamo il framework PhoneGap (in grigio) che ci viene fornito insieme alle API Javascript. Da notare il rettangolo arancione che rappresenta l'embedded browser che incapsula la web application e le API. Di default il browser viene aperto esternamente all'applicazione. Di conseguenza ad ogni richiesta di una pagina web avviene l'apertura di un bottone fornito dal browser. In pratica si rende impossibile notare che si tratta di un browser e quindi di una pagina web. Ciò rende l'applicazione più simile ad un applicazione nativa.

Infine, in verde è evidenziata la parte a carico dello sviluppatore che è la vera e propria web application, realizzata all'interno del framework PhoneGap.

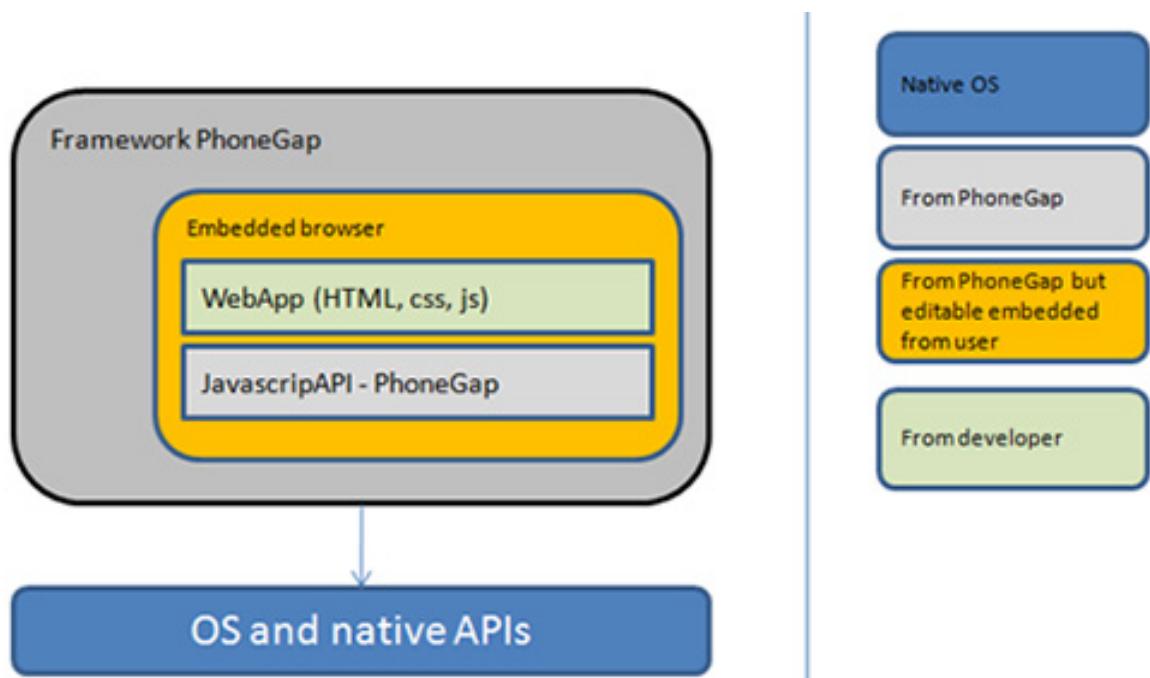


Figura 4.4: Architettura di un'applicazione realizzata con PhoneGap

Come detto PhoneGap mette a disposizione del programmatore, una serie di API per l'accesso all'hardware nativo del device. Bisogna tenere conto però che non tutte le piattaforme hanno a disposizione gli stessi sensori e che occorre prestare molta attenzione nel caso in cui il building dell'applicazione è fatto su piattaforme

diverse. Il framework, infatti, offre supporto per ogni piattaforma degna di essere presa in considerazione ovvero: Android, iOS, Symbian, WebOS, Blackberry etc. Non tutte le piattaforme sono però supportate allo stesso modo. La seguente tabella (Fig 4.5) riassume lo stato dell'arte riguardo la compatibilità e l'accessibilità che il framework offre verso i sensori presenti nei diversi device, equipaggiati da diversi OS.

| | amazon-fireos | android | blackberry10 | Firefox OS | ios | Ubuntu | wp8 (Windows Phone 8) | windows (8.0, 8.1, 10, Phone 8.1) |
|--------------------------|-----------------------|-----------------------|---|-----------------------|-----------------|----------|---|---|
| cordova CLI | ✓ Mac, Windows, Linux | ✓ Mac, Windows, Linux | ✓ Mac, Windows | ✓ Mac, Windows, Linux | ✓ Mac | ✓ Ubuntu | ✓ Windows | ✓ |
| Embedded WebView | ✓ (see details) | ✓ (see details) | X | X | ✓ (see details) | ✓ | X | X |
| Plug-in Interface | ✓ (see details) | ✓ (see details) | ✓ (see details) | X | ✓ (see details) | ✓ | ✓ (see details) | ✓ |
| Platform APIs | | | | | | | | |
| Accelerometer | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| BatteryStatus | ✓ | ✓ | ✓ | ✓ | ✓ | X | ✓ | ✓ * Windows Phone 8.1 only |
| Camera | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Capture | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ |
| Compass | ✓ | ✓ | ✓ | X | ✓ (3GS+) | ✓ | ✓ | ✓ |
| Connection | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ |
| Contacts | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | partially |
| Device | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Events | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ |
| File | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ |
| File Transfer | ✓ | ✓ | ✓ * Do not support onprogress nor abort | X | ✓ | X | ✓ * Do not support onprogress nor abort | ✓ * Do not support onprogress nor abort |
| Geolocation | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Globalization | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ |
| InAppBrowser | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ | uses iframe |
| Media | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ |
| Notification | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ |
| Splashscreen | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ |
| Status Bar | X | ✓ | X | X | ✓ | X | ✓ | ✓ Windows Phone 8.1 only |
| Storage | ✓ | ✓ | ✓ | X | ✓ | ✓ | ✓ localStorage & indexedDB | ✓ localStorage & indexedDB |
| Vibration | ✓ | ✓ | ✓ | ✓ | ✓ | X | ✓ | ✓ * Windows Phone 8.1 only |

Figura 4.5: Le API e il loro supporto per le varie piattaforme

4.3 Ratchet

Il solo utilizzo del framework PhoneGap non è sufficiente per realizzare un’applicazione degna di nota. L’interfaccia grafica costituisce un’elemento fondamentale nelle moderne applicazioni mobili. Inoltre, se ben progettata, contribuisce ad aumentare la user experience [14] e in generale l’usabilità del sistema stesso. A tale proposito è stato utilizzato il framework opens source **Ratchet** [<http://goratchet.com/>].



Figura 4.6: Il logo del framework Ratchet

I componenti grafici offerti da questo strumento sono responsive e appositamente progettati per gli smartphone; il loro utilizzo è tanto semplice quanto efficace, analogamente ad altri progetti di successo, come Bootstrap, i componenti possono essere inclusi nell’interfaccia aggiungendo dello specifico codice HTML e vari stili CSS (costumizzabili a proprio piacimento). A partire dalla seconda versione è stata integrata una classe di icone con immagini di uso comune nelle applicazioni mobili. La combinazione di questo tool insieme ad altri per l’accesso al DOM (Document Object Model) dell’applicazione (vedi 4.2), nel nostro caso *jQuery*, permette di realizzare applicazioni con un’interfaccia accattivante e molto simile a quelle native.

Ad esempio per aggiungere la ”classica” tab-bar basta inserire il seguente codice HTML:

```
<nav class="bar bar-tab">
  <a class="tab-item active" href="#">
    <span class="icon icon-home"></span>
    <span class="tab-label">Home</span>
  </a>
  <a class="tab-item" href="#">
    <span class="icon icon-person"></span>
    <span class="tab-label">Profile</span>
  </a>
  <a class="tab-item" href="#">
    <span class="icon icon-star-filled"></span>
    <span class="tab-label">Favorites</span>
  </a>
  <a class="tab-item" href="#">
    <span class="icon icon-search"></span>
    <span class="tab-label">Search</span>
  </a>
  <a class="tab-item" href="#">
    <span class="icon icon-gear"></span>
    <span class="tab-label">Settings</span>
  </a>
</nav>
```

Ed ottenere il seguente risultato:

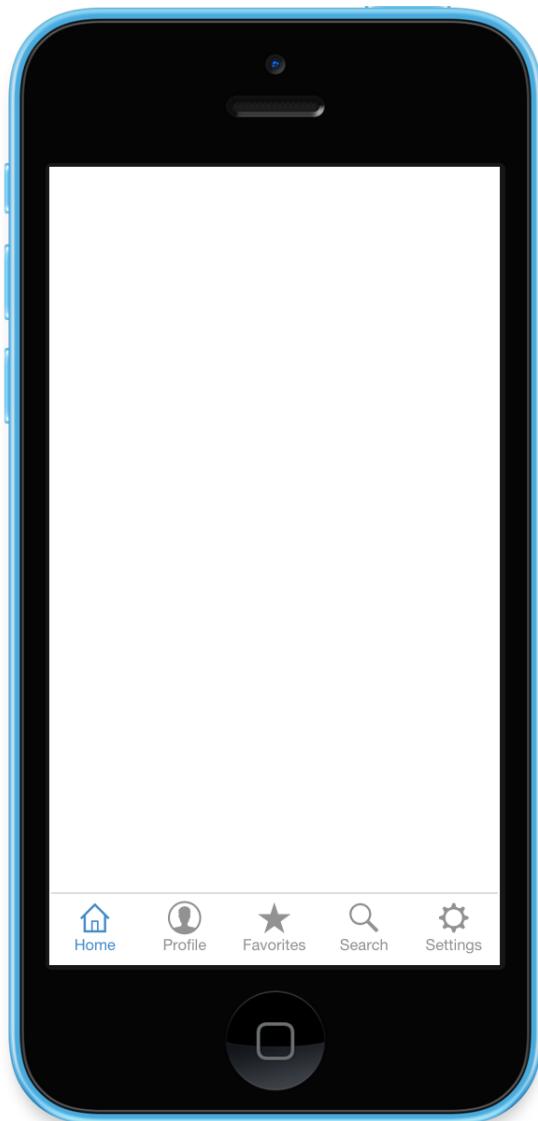


Figura 4.7: Il componente tab-bar con cinque icone

4.4 Leaflet

Per la mappa interattiva si è utilizzata la libreria **Leaflet** [15]. Leaflet è una libreria open source JavaScript per le mappe interattive mobili-friendly. Con un peso di soli circa 33 KB di JS, ha tutte le caratteristiche e le maggiori funzioni necessarie agli sviluppatori.

Leaflet è stato progettato per avere alte prestazioni ed facilmente utilizzabile. Funziona in modo efficiente su tutte le principali piattaforme desktop e mobile, può essere estesa con un sacco di plugin e la documentazione è ben fatta.



Figura 4.8: Il logo della libreria Leaflet

Il codice per inserire una mappa all’interno di un *div* con *id=’map’* è semplicemente:

```
var map = L.map('map').setView([51.505, -0.09], 13);

L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png',
{
    attribution: '&copy; <a href="http://osm.org/
        copyright">OpenStreetMap</a> contributors'
}).addTo(map);
//aggiunge un marker alle coordinate [51.5,-0.09]
L.marker([51.5, -0.09]).addTo(map)
    .bindPopup('A pretty CSS3 popup.<br> Easily
        customizable.')
    .openPopup();
```

Con il seguente risultato:

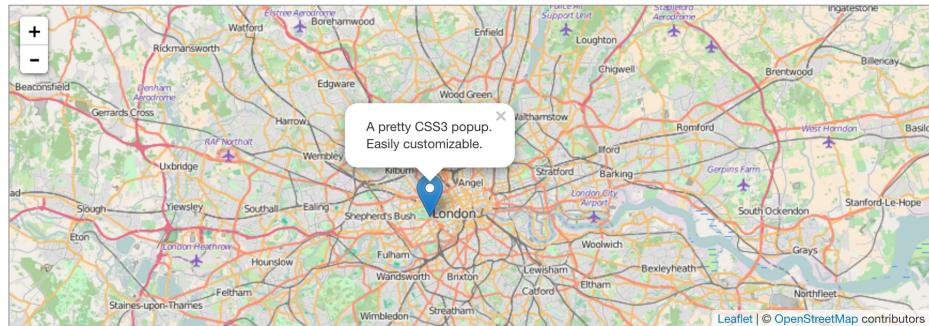


Figura 4.9: Risultato grafico con il codice precedente

Le coordinate sono implementate attraverso l'oggetto latLng, ed è possibile crearne delle nuove nel seguente modo:

```
var latlng = L.latLng(50.5, 30.5);
```

Come detto la libreria dispone di una grande quantità di funzioni e metodi utili, ad esempio l'evento sulla geolocalizzazione:

```
function onLocationFound(e) {  
    console.log("localizzato!!")  
};
```

L'oggetto "e" contiene al suo interno, oltre alle coordinate, altri oggetti utili come la velocità, l'orario, l'altitudine ect.

Esistono altre librerie per utilizzare i dati ottenuti da un server OpenStreetMap, tuttavia questa si è rivelata essere molto efficiente e semplice; merito di una curva di apprendimento buona e della ricca e chiara documentazione fornita. Inoltre è stato utilizzato un plugin che permette di salvare, in locale al device, la mappa visualizzata dall'utente su diversi livelli di zoom.

4.5 La griglia trasparente

Come specificato nei requisiti di sistema, l’utente e gli eventi devono essere associati in modo univoco ad una cella di grandezza $(22 * 16)mt^2$; per fare ciò è necessario che il sistema sia in grado di ”costruire” dinamicamente una griglia. Questa deve essere del tutto invisibile agli utenti che non hanno alcun interesse a conoscere o comprendere tale dettaglio implementativo.

Prima di spiegare la soluzione a questo problema occorre fare un passo indietro, sin dai tempi più remoti l’uomo ha cercato di creare delle mappe cartografiche. Le più antiche testimonianze [16] conosciute di qualcosa che assomigli ad una rappresentazione cartografica non riguardano la terra, ma il cielo, così come appare di notte. Sui muri delle grotte di Lascaux sono stati infatti osservati dei puntini dipinti databili al 16.500 a.C. che rappresentano il cielo notturno ed in cui si possono riconoscere Vega, Deneb e Altair (il cosiddetto Triangolo estivo), nonché le Pleiadi. Da allora la cartografia ha fatto passi da gigante(vedi cap 2).

Tornando al nostro problema, la soluzione trovata si basa sul calcolo della distanza tra due coordinate geografiche, il cui problema è il calcolo stesso. Non avendo la Terra una forma lineare, è difficile trovare una rappresentazione geometrica perfetta di essa.

Gli algoritmi più utilizzati per il calcolo della distanza tra due punti, utilizzano le formule di Haversine e Vincenty, il primo si basa sull’approssimazione della terra ad una sfera perfetta, mentre il secondo la modelizza in uno sferoide oblato.

E’ bene dire che mentre il risultato ottenuto tramite la formula di Haversine può avere un’errore dello 0,3% quello di Vincenty è molto più accurato, con un errore di 0.5mm circa; questa precisione si ha a discapito di un costo computazionale più elevato. Per brevi distanze i metodi tendono invece a convergere.

4.5.1 L'algoritmo Santiago

Con entrambi gli algoritmi è stato verificato che una variazione unitaria di 10^{-4} gradi in latitudine corrisponde ad una distanza di 11 mt, 8 mt se in longitudine. Partendo da questo risultato si è così ideato il seguente algoritmo.

1. Per prima cosa, si considera l'utente posto all'interno di un sistema di riferimento calcolato troncando le sue coordinate alla terza cifra decimale. La grandezza fisica di questi sistemi è quindi di 9680mt^2 . L'origine corrisponde alle coordinate dell'utente troncate alla 10^{-4} , con l'ultima cifra decimale uguale a zero.

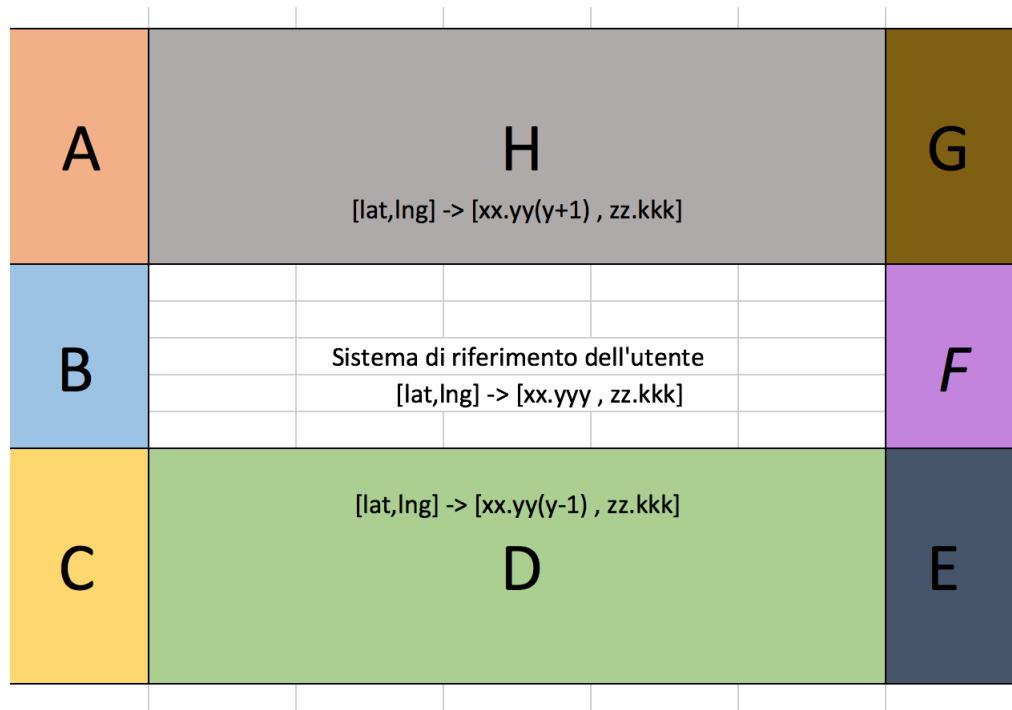


Figura 4.10: Sistema di riferimento dell'utente

2. Le coordinate dell’utente (o dell’evento) vengono quindi troncate alla quarta cifra decimale che viene posta a zero, questo significa mappare tutte le persone, all’interno di un area rettangolare di $88m^2$, nell’angolo in basso a sinistra dello stesso rettangolo (ndr Santiago de Chile si trova in direzione sud-ovest dall’Italia, da cui il nome dell’algoritmo). Di seguito un’immagine per chiarire meglio il funzionamento.

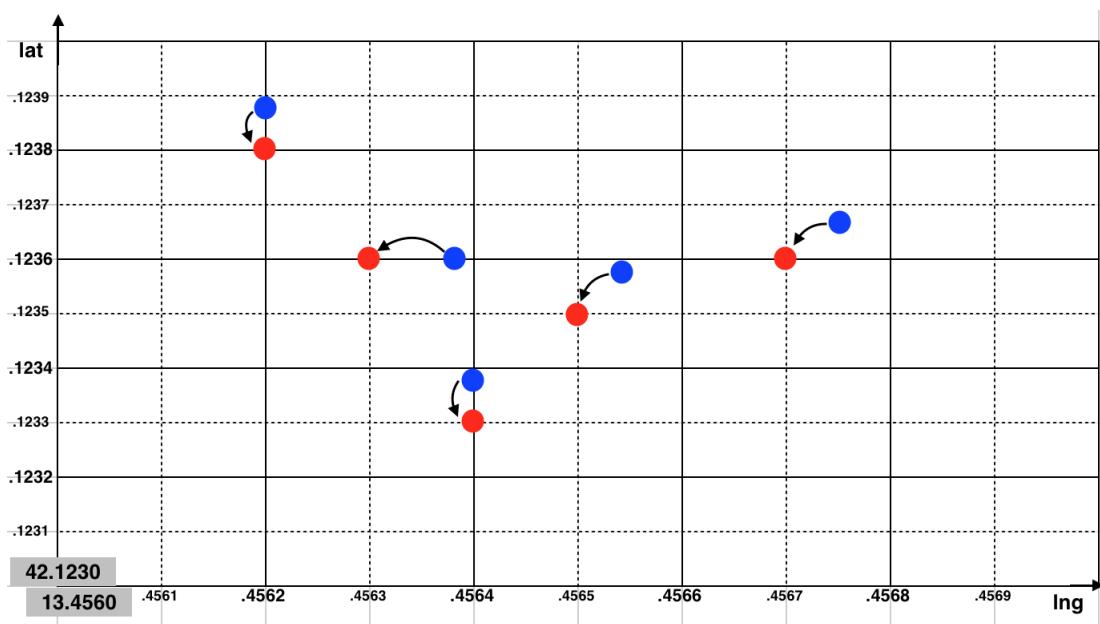


Figura 4.11: Troncamento delle coordinate alla quarta cifra decimale posta a zero

I cerchi blu rappresentano le posizioni di diversi utenti, mentre i cerchi rossi rappresentano le posizioni dopo il troncamento. E’ bene osservare che i punti sui confini in alto e a destra vengono mappati nella cella (o sistema per quelli ai confini) successiva.

3. Le specifiche impongono celle di grandezza doppia rispetto a quelle ottenibili semplicemente troncando le coordinate alla quarta cifra decimale, questo vuol dire che quattro rettangoli di grandezza $(11 * 8)mt^2$ compongono una cella della nostra griglia; così facendo ogni sistema è composto da 25 celle, come possiamo vedere nella figura successiva.

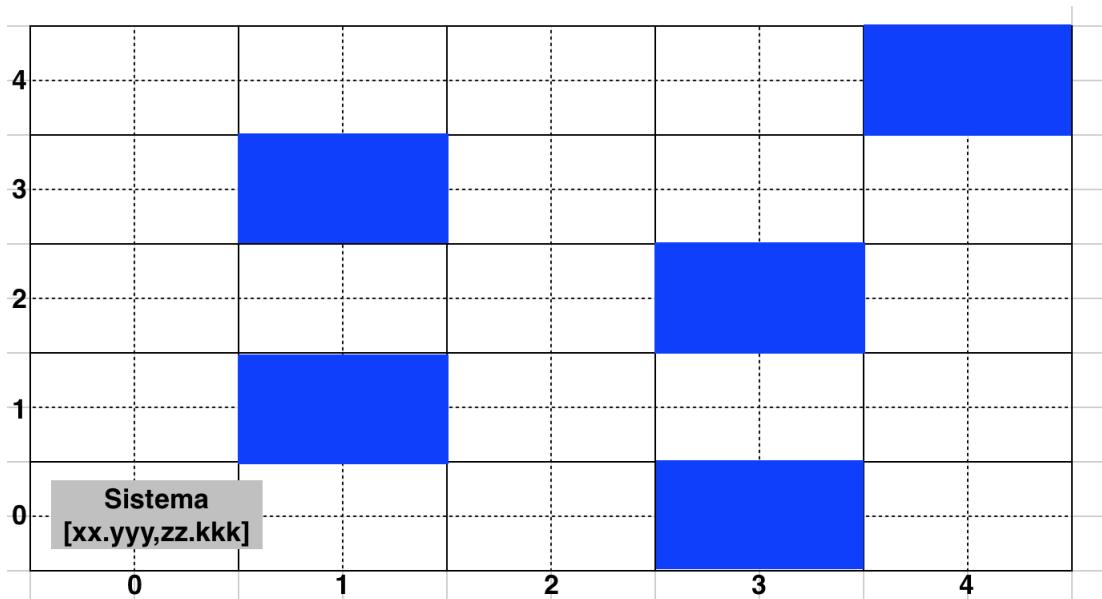


Figura 4.12: Alcune celle evidenziate in blu

Quindi gli utenti/eventi sono **identificati senza ambiguità in tutto il globo** dalla seguente ennupla:

- Sistema di riferimento espresso in coordinate troncate alla terza cifra decimale $[xx.yyy, zz.kkk]$
- Coppia di interi indicante il numero di cella all’interno del sistema $[lat_y; lng_x]$

4. A questo punto bisogna ottenere la coppia di interi $[lat_y; lng_x]$, il sistema calcola la distanza dagli assi del sistema, dividendo in modulo per 22 quella con l'asse della latitudine e dividendo in modulo per 16 quella con l'asse longitudinale.

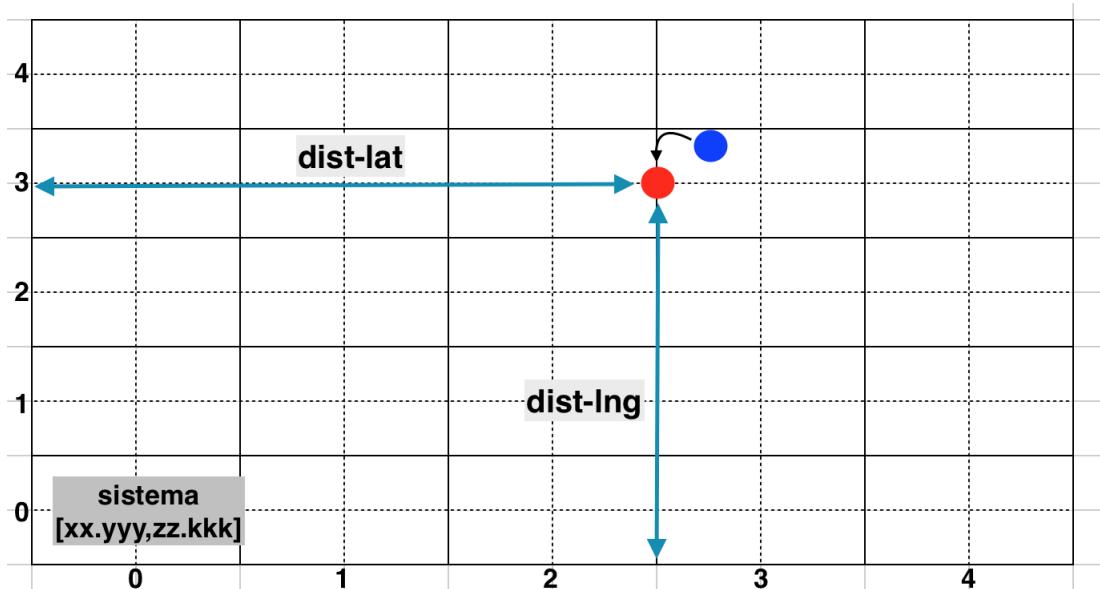


Figura 4.13: Proiezione delle distanze utilizzate per il calcolo della cella

Poichè la distanza massima si ha in corrispondenza della cella $[4; 4]$ ed equivale a $110mt$, l'errore massimo di calcolo sarà pari a $33cm$. Questo è il motivo della scelta di un sistema con queste dimensioni, infatti troncando alla seconda cifra decimale avremo sistemi di $0.968km^2$ e un'errore di calcolo massimo pari a $3.3mt$, senza considerare l'aumento del costo computazionale richiesto.

Il procedimento inverso consiste nella seguente formula:

$$[xx.yyy; zz.kkk] + 0.0001 + ([lat_y; lng_x] * 0.0001 * 2) \quad (4.1)$$

Così facendo tutti gli eventi/persone vengono mappati al centro delle varie celle, con un errore massimo dalla posizione reale (trascurando l'errore inevitabile del GPS) di 13mt.

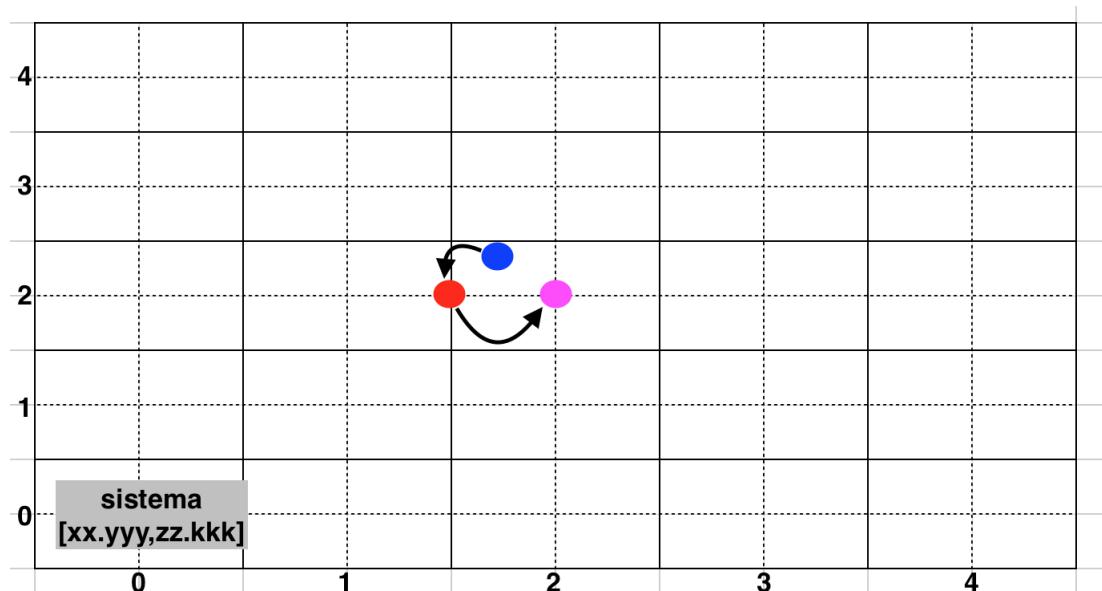


Figura 4.14: Posizione ottenuta riconvertendo la tupla di dati vista nel passo 3 dell'algoritmo

4.5.2 Il codice

Vediamo quindi il codice *Javascript* utilizzato per implementare l’algoritmo di Santiago. La prima funzione calcola il sistema e la posizione dell’evento o della persona all’interno di esso.

```
latLng->{lat:Number,lng:Number}

calcCell:function(latLng) {
    var lat=latLng.lat;
    var lng=latLng.lng;
    var cut=Math.pow(10,4);
    //coordinate origine del sistema
    var coordinates=L.latLng(Math.floor(lat*cut)/cut,Math
        .floor(lng*cut)/cut);
    //coordinate perpendicolari lat e lng
    var perpendicular_lat=L.latLng(Math.floor(coordinates
        .lat*1000)/1000,coordinates.lng);
    var perpendicular_lng=L.latLng(coordinates.lat,Math.
        floor(coordinates.lng*1000)/1000);
    //richiama un metodo per il calcolo della distanza
    //utilizzando l’algoritmo di Haversine
    var dist_lat=this.calcDist(coordinates,
        perpendicular_lat).haversine;
    var dist_lng=this.calcDist(coordinates,
        perpendicular_lng).haversine;
    var dist={dist_lat,dist_lng};
    //divisione in modulo per trovare la cella all’interno
    //del sistema
    var cell_number_lat=Math.floor(dist_lat/22);
    var cell_number_lng=Math.floor(dist_lng/16);
    var cell={cell_number_lat,cell_number_lng};
    return ({zero:L.latLng(perpendicular_lat.lat,
        perpendicular_lng.lng),cell}),
}
```

E ora il codice per l’equazione inversa (4.1):

```
data-> {zero:{latLng},cell:{cell_number_lat (Number),  
cell_number_lng (Number) }}  
  
coordFromCell:function (data) {  
  
    return L.latLng(data.zero.lat+0.0001+(data.cell.  
    cell_number_lat*0.0001*2),data.zero.lng  
    +0.0001+(data.cell.cell_number_lng*0.0001*2))  
    ;  
  
} ,
```

4.6 La struttura architetturale

Come detto l’applicazione deve comunicare con un main-server, quindi l’architettura è quella classica delle applicazioni web, ovvero client-server. Tuttavia si è pensato di utilizzare anche il concetto di REST.

REST [17] definisce un insieme di principi architetturali per la progettazione di un sistema. Rappresenta uno stile architetturale, cioè non si riferisce ad un sistema concreto e ben definito né si tratta di uno standard stabilito da un organismo di standardizzazione.

Le risorse sono gli elementi fondamentali su cui si basano i Web Service RESTful, a differenza dei Web Service SOAP-oriented che sono basati sul concetto di chiamata remota. Per risorsa si intende un qualsiasi elemento oggetto di elaborazione. Per fare un parallelo con la programmazione ad oggetti possiamo dire che una risorsa può essere assimilata ad una istanza di una classe.

Il principio che stiamo analizzando stabilisce che ciascuna risorsa deve essere identificata univocamente. Essendo in ambito Web, il meccanismo più naturale per individuare una risorsa è dato dal concetto di URI.

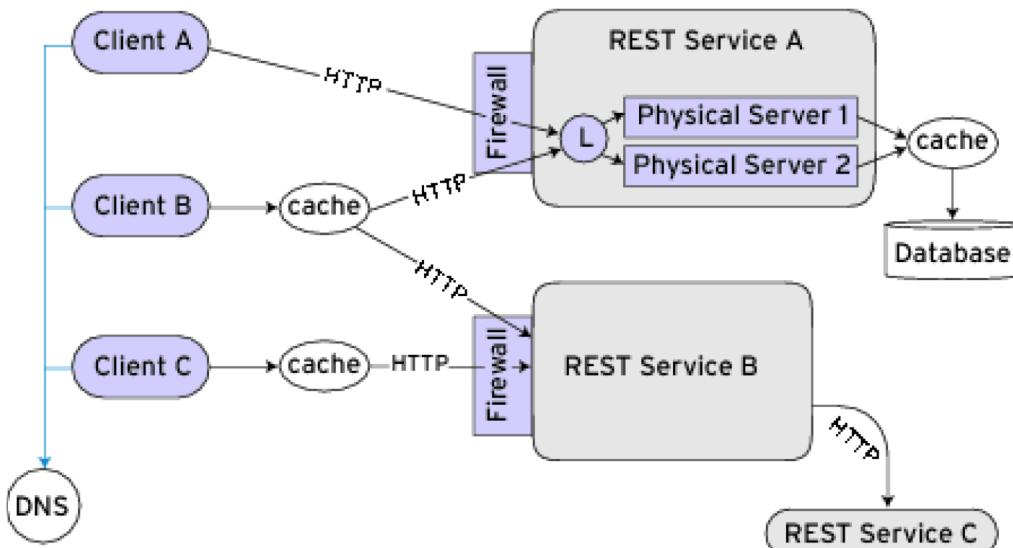


Figura 4.15: Rappresentazione semplificata dell’architettura

Così facendo, non si deve implementare un metodo del tipo `"getDati(idutente)"`, ma basterà eseguire una chiamata HTTP con metodo GET alla risorsa `"http://www.myapp.com/dati/idutente"`. Questo approccio permette quindi di definire un'interfaccia tra il client e il server, rendendo quest'ultimo più scalabile. La realizzazione del server non è argomento di questa tesi, tuttavia è necessario definire la struttura dei dati trasmessi e ricevuti dall'applicazione, in modo tale che in futuro il progettista del server possa implementare il REST service come meglio crede purché rispetti il formato dei dati stabilito.

Di seguito i sequence diagram relativi alla richiesta e la trasmissione di informazioni tra il client e il server, per ognuni pacchetto verrà mostrato il formato stabilito.

Richiesta dei propri FOI e POI: In questa fase l'applicazione deve ricevere i dati sui POI e i FOI dell'utente. Il formato della risorsa dipenderà dal successo o meno della richiesta. Il pacchetto ErrorFP è lo stesso generato dall'applicazione in caso di rete assente.

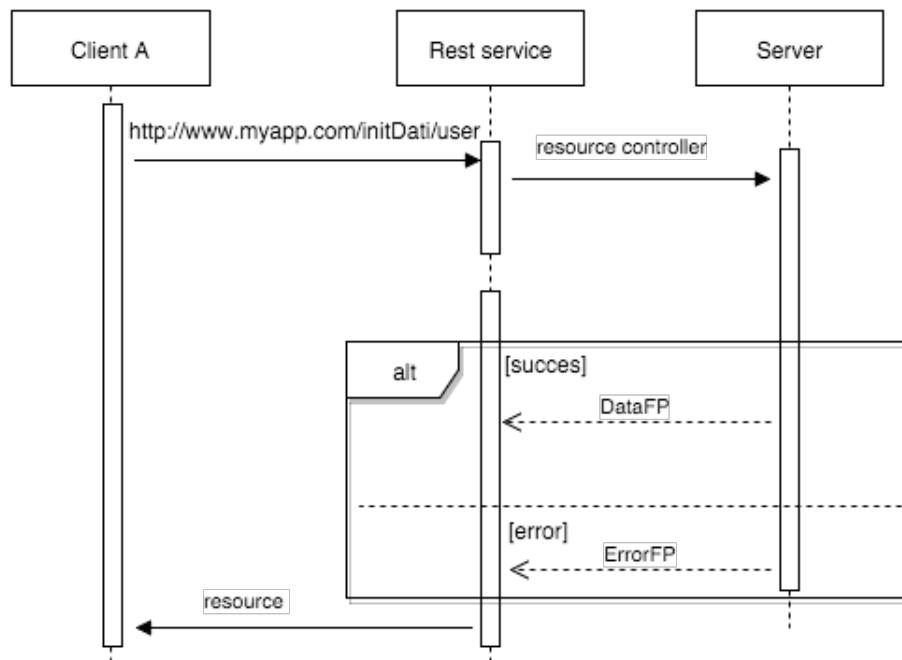


Figura 4.16: Rappresentazione della sequenza di azioni per la richiesta dei propri FOI e POI

- `GET http://www.myapp.com/initDati/user`, è l'URI della risorsa contenente le informazioni sui POI e i FOI dell'utente. Il parametro `user` serve per identifi-

care l’utente che sta facendo la richiesta.

- *DataFP*, in caso di successo la resource è nel seguente formato:

```
{  
  poi: [  
    {  
      nome: String,  
      emergency: String,  
      number: Integer,  
      ultimo_ora: String,  
      ultimo_data: String,  
      position:  
        {  
          zero:{  
            lat: Number,  
            lng: Number  
          },  
          cell:{  
            cell_number_lat:  
              Number,  
            cell_number_lng:  
              Number  
          }  
        }  
    },  
    .  
    .  
    .  
  ],  
  foi: [  
    {  
      nome: String,  
      stato: String,  
      ora_stato: String,  
    }]
```

```
        data_stato: String,
        dispositivo: String,
        ora_dispositivo: String,
        position:
        {
            zero:{ 
                lat: Number,
                lng: Number
            },
            cell:{ 
                cell_number_lat:
                    Number,
                cell_number_lng:
                    Number
            }
        }
    },
    .
    .
    .
]
```

- *ErrorFP*, in caso di errore la resource è semplicemente una stringa:

```
{
    error: String
}
```

Aggiornamento status: di seguito si illustrano le azioni legate all'aggiornamento dello status di un'utente (client A). La sequenza si compone di una prima fase, nella quale l'utente chiede al server di modificare la risorsa corrispondente al suo status, è una seconda fase nella quale un altro utente (client B), che ha tra i suoi FOI il precedente, richiede al server l'aggiornamento dei dati (l'algoritmo legato ai tempi e i modi di aggiornamento dei propri dati non è stato sviluppato, andrebbe fatto tenendo conto del contesto e dello stato della rete).

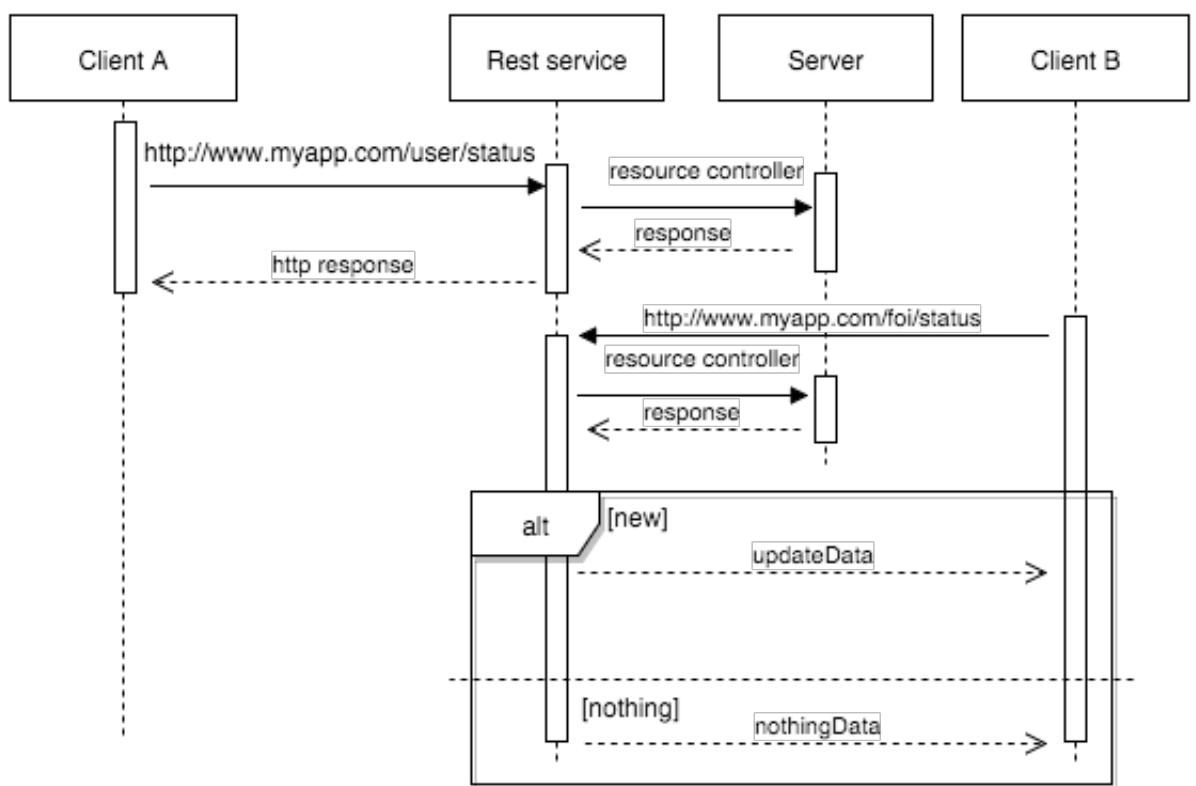


Figura 4.17: Rappresentazione semplificata dell'architettura

- `PUT http://www.myapp.com/iduser/status`, è l'URI della risorsa contenente le informazioni sull'utente "user", questo parametro serve ad identificarlo. I dati inviati sono nel seguente formato:

```
{
    stato: String,
    cod_emergency: Integer,
```

```
    ora_stato: String,  
    data_stato: String,  
}
```

Per il successo dell’operazione è sufficiente verificare la response http del server direttamente nell’applicazione.

- *GET* *http://www.myapp.com/iduser/foi/status*, è l’URI della risorsa contenente tutte le informazioni dei FOI di un relativo utente (parametro *iduser* per identificare l’utente che richiede la risorsa).
- il pacchetto ”*updateData*”, contiene tutti e soli i FOI con un nuovo status. Il formato è:

```
{  
  foi: [  
    {  
      nome: String,  
      stato: String,  
      ora_stato: String,  
      data_stato: String,  
      dispositivo: String,  
      ora_dispositivo: String,  
      position:  
      {  
        zero:{  
          lat: Number,  
          lng: Number  
        },  
        cell:{  
          cell_number_lat:  
            Number,  
          cell_number_lng:  
            Number  
        }  
      }  
    },  
    .
```

```
    .  
    .  
],  
}
```

- nel caso in cui non ci siano aggiornamenti, il server restituisce il pacchetto “nothing”:

```
{  
  message: nothing  
}
```

Capitolo 5

Conclusioni e prospettive future

Ringraziamenti

Bibliografia

- [1] Theoretical Notes on Vulnerability to Disaster <http://emergency-planning.blogspot.it/2009/01/theoretical-notes-on-vulnerability-to.html>
- [2] Fifty-six Common Misconceptions About Disaster <http://emergency-planning.blogspot.it/2008/12/forty-four-common-misconceptions-about.html>
- [3] Phases of Emergency Management <http://www.cheltenhamtownship.org/mobile/pview.aspx?id=4161&catid=29>
- [4] Disaster Response: a Multi-Agent based Approach, http://www.sapienzaapps.it/chitaly2015/wp-content/uploads/2015/09/02_chitaly2015_Costantini_et_al.pdf
- [5] Open Data Commons
<http://opendatacommons.org/licenses/odbl/summary/>
- [6] wiki.openstreetmap, “Perché state realizzando OSM?”
<http://wiki.openstreetmap.org/wiki/IT:FAQ>
- [7] Why the world needs OpenStreetMap, “Perché il mondo ha bisogno di OSM?” <http://www.theguardian.com/technology/2014/jan/14/why-the-world-needs-openstreetmap>
- [8] wiki.openstreetmap, “Com’è possibile che un progetto del genere porti a mappe accurate?”
<http://wiki.openstreetmap.org/wiki/IT:FAQ>

- [9] Google Developers , “JavaScript API Usage Limits” <https://developers.google.com/maps/pricing-and-plans/>
- [10] Google Support , “Download e utilizzo di aree offline” <https://support.google.com/gmm/answer/6291838?hl=it>
- [11] Wiki OSM, “Planet.OSM” <https://wiki.openstreetmap.org/wiki/Planet.osm>
- [12] Humanitarian Openstreetmap Team, “Disaster Mapping” <https://hotosm.org/projects/disaster-mapping>
- [13] Terremoto di Haiti del 2010 https://it.wikipedia.org/wiki/Terremoto_di_Haiti_del_2010
- [14] User Experience https://it.wikipedia.org/wiki/User_Experience
- [15] Leaflet: an open-source JavaScript library for mobile-friendly interactive maps https://it.wikipedia.org/wiki/User_Experience
- [16] Storia della cartografia https://it.wikipedia.org/wiki/Storia_della_cartografia
- [17] I principi dell’architettura RESTful <http://www.html.it/pag/19596/i-principi-dellarchitettura-restful>