Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

FOG ESCAPING:
Un'applicazione di mobile data Crowd-Sourcing per il supporto alla mobilità veicolare

Tesi di Laurea in Basi di Dati e Sistemi Informativi

Relatore:
Chiar.mo Prof.
MARCO DI FELICE

Presentata da:
FEDERICO MACCAFERRI

Sessione III
Anno Accademico 2012-2013

Part of the inhumanity of the computer is that,
once it is competently programmed and working smoothly,
it is completely honest...
I.Asimov

Indice

Elenco delle figure

Capitolo 1

Introduzione

È chiaro oramai, come sia quasi impossibile nel terzo millennio, vivere lontano dai nostri smartphone, perché rappresentano in ogni momento, il punto d'unione tra noi, la nostra vita privata, quella intima, sociale e culturale o lavorativa che sia. Alcuni paradigmi che stanno evolvendo come il Participatory Sensing e il Data Crowd-Sourcing, possono rendere questi device sempre più utili. I progressi tecnologici che in continuazione migliorano gli smartphone, hanno spianato le strade a un nuovo paradigma per i rilevamenti su larga scala, conosciuto come Participatory Sensing. L'idea dietro il Participatory Sensing è di sensibilizzare e responsabilizzare i cittadini, in modo che essi possano raccogliere dati dagli ambienti circostanti e renderli disponibili alla collettività. È possibile quindi creare una rete dove il cittadino stesso crea i contenuti utili che saranno utilizzati per la soddisfazione di un bisogno collettivo. I telefoni cellulari, anche se non costruiti appositamente per il rilevamento, sono muniti di svariati sensori che possono essere utilizzati per raccogliere dati dall'ambiente e che quindi aprono agli sviluppatori, un mondo di possibili e plausibili innovazioni tecnologiche.

Gli smartphone diventano sempre più intelligenti grazie all'utilizzo di nuove tecnologie integrate o nella rete. Iniziano a capire alcuni pattern della nostra vita, imparano il nostro stato di salute e ci aiutano durante la giornata. Gli smartphone diventeranno sempre più cognitivi [1]. Nel futuro sarà

sempre più indispensabile che gli smartphone conoscano e prevedano i modi in cui gli utenti utilizzano le applicazioni. La previsione di utilizzo delle applicazioni potrebbe dare benefici come il pre-caricamento, riducendo drasticamente i tempi di apertura, rendendoli quasi istantanei [2].

L'idea alla base di questa Tesi di laurea è creare un'applicazione semplice, intuitiva e rapida che permetta di risolvere un problema molto sentito nel nostro territorio: *La circolazione in caso di scarsa visibilità per colpa delle nebbie.* Questa problematica è stata sollevata dal Responsabile dell'Area di Progettazione Sistemi Informativi, il Sig.Cattani Stefano di ARPA (Agenzia Regionale Per l'Ambiente)

In questa Tesi si affrontano tutte le idee e i passaggi che hanno permesso la creazione dell'applicazione *Fog Escaping*. Nel secondo capitolo sono analizzati i paradigmi di Data Crowd-Sourcing e Participatory Sensing, mostrando una panoramica delle Applicazioni esistenti. Nel terzo capitolo è descritto il problema riguardante lo sviluppo dell'applicazione *Fog Escaping*, attraverso i ragionamenti che hanno portato alla sua realizzazione. Nel quarto capitolo è descritta l'implementazione del Sistema; sono specificati i dettagli implementativi riguardanti il Server, il Database e il Client. Nel quinto capitolo sono descritti gli studi effetuati sulla Image Recognition; sono specificati i dettagli implementativi riguardanti il riconoscimento di oggetti in movimento e i metodi di analisi delle immagini. Nel sesto capitolo sono analizzate le prestazioni dell'applicazione, confrontandola con altre simili; sono inoltre presenti alcuni screenshot dell'applicazione *Fog Escaping* in esecuzione. Nel settimo e ultimo capitolo è presente un riepilogo del lavoro realizzato e una discussione critica sulle difficoltà riscontrate e sulle possibili implementazioni future.

Capitolo 2

Stato dell'Arte

In questo capitolo s'introduce il paradigma di Participatory Sensing; ovvero la raccolta di dati dalle persone che possiedono dispositivi mobili (Data Crowd-Sourcing). Si discuteranno i vantaggi, gli svantaggi e si farà una panoramica sulle tegnologie attualmente sviluppate.

2.1 Participatory Sensing

La crescente presenza della connettività internet e di potenti sensori sui nostri device mobili, ha aperto le porte a un nuovo paradigma per il monitoraggio del territorio conosciuto come Participatory Sensing [3], [4]. Più di 5 bilioni di persone in tutto il mondo hanno accesso ai telefoni cellulari. I progressi raggiunti nel campo delle tecnologie mobili, e la loro ubiquità, hanno permesso la rilevazione dei dati su larga scala. L'idea dietro al Participatory Sensing si basa sul concetto della raccolta e della condivisione dei dati negli ambienti circostanti da parte dei cittadini attraverso i dispositivi mobili. I telefoni cellulari non sono stati creati per specifiche rilevazioni, ma possiedono una serie di sensori che se utilizzati nella maniera corretta possono generare dati affidabili.

2.1.1 Vantaggi

Sono molti i vantaggi derivanti dai sistemi di rilevazione cooperativa. Innanzitutto si utilizzerebbero rilevatori già esistenti (i telefoni cellulari). La comunicazione può essere fatta tramite la rete Wifi o Cellulare; quindi i costi d'installazione sono virtualmente zero. Come secondo vantaggio si ha una copertura spaziotemporale che non ha precedenti; questo tipo di copertura sarebbe capace di rilevare eventi altrimenti impossibili da riscontrare (come l'inquinamento localizzato). Come terzo vantaggio si hanno le economie di scala dovute dall'utilizzo dei cellulari invece di sofisticati sensori. Come ultimo, ma non meno importante, abbiamo la presenza di sistemi di sviluppo e di reti di distribuzione che permettono una pubblicazione veloce e semplice delle applicazioni.

2.1.2 Problematiche

Invio di Dati Incompleti

La raccolta dei dati arriva direttamente dalle persone che contribuiscono alle rilevazioni; questo limita la misurazione ai soli posti in cui queste persone sono presenti. Inoltre è possibile che gli utenti vogliano preservare l'uso del dispositivo mobile per altre applicazioni, o potrebbero scegliere di raccogliere dati solo quando hanno sufficiente energia. Questi dati essendo tipicamente distribuiti randomicamente nello spazio e nel tempo, sono incompleti. È quindi un obiettivo importante riuscire ad utilizzare tecniche capaci di analizzare questi dati e trarne conclusioni affidabili [5].

Dedurre Contenuti e Attività dell'Utente

È importante che i dispositivi mobili diventino sempre più cognitivi [1], e che comprendano e apprendano le tipologie di attività svolte da un utente in un determinato momento. Sono quindi necessarie tecniche di machine learning che raccolgano una mole di dati dai sensori e ne ricavino una serie di attività [6].

Preservare la Privacy dell'utente

I dati nel Participatory Sensing sono raccolti su larga scala, è quindi necessario che siano protetti, evitando intrusioni nella privacy dell'utente. Finché i dati saranno raccolti direttamente dagli utenti, si avrà una certa riluttanza da parte delle persone. Il rischio è che questi dispositivi si trasformino in microspie poiché possono conoscere se siamo svegli, dove siamo, se stiamo facendo una conversazione e con chi. Sono quindi state studiate applicazioni capaci di prelevare i dati in modo anonimo tramite la rilevazione opportunistica [7], dove non è il dispositivo a inviare autonomamente i dati ma è utilizzata una strategia di delegazione per la raccolta.

Valutare l'Affidabilità dei dati

È molto importante che si raccolgano dati su larga scala; purtroppo è possibile che ci siano persone che non collaborano correttamente, inviando dati non affidabili o errati. È quindi una sfida importante riuscire a capire chi sta inviando dati corretti e chi no. Sono quindi stati proposti alcuni sistemi di reputazione [8], dove gli utenti hanno un punteggio che riflette l'affidabilità dei dati che hanno inviato e quelli ancora da inoltrare.

Risparmio Energetico

Gli utenti si sentono realizzati nel contribuire alla raccolta dei dati solo se questo non compromette notevolmente la durata della batteria del dispositivo. In questo momento i dispositivi mobili vengono ricaricati una volta al giorno, questo però non deve essere un presupposto per un consumo maggiore di energia. Al contrario è necessario che i dispositivi trattengano la carica a lungo, affinché il processo di rilevazione dei dati sia il meno costoso possibile. Alcuni sensori come il GPS, consumano molta più energia degli altri, ma la necessità di sapere dove è localizzato l'utente ha permesso lo sviluppo del Participatory Sensing. È quindi necessario utilizzare questi sensori in modo conservativo. Sono stati studiati dei sistemi che permettono di passare dalla

rilevazione accurata ma costosa del GPS, alle rilevazioni meno accurate ma più efficienti, tramite la localizzazione Wifi o tramite rete Cellulare [9].

Anche le piattaforme di sviluppo come *XCode* di Apple, permettono di richiedere la posizione dell'utente nel modo più conservativo possibile, passando dalla rilevazione GPS, all'identificazione tramite vicinanza a una rete Wifi, oppure alla triangolazione delle reti Cellulari. Ovviamente non tutte le applicazioni possono utilizzare questa politica conservativa, infatti, quelle di navigazione consumano molta energia per la necessità di utilizzo di dati precisi provenienti dal GPS.

Stress delle Infrastrutture

La proliferazione dei dispositivi mobili ha reso disponibile la possibilità di raccogliere dati su larga scala tramite il Crowd-Sourcing. Le soluzioni correnti, dipendendo dalla grandezza della folla, creano imprevedibili stress sulle infrastrutture e la dipendenza di energia dei device può causare una rilevazione non accurata del comportamento delle persone [10]. Sono state considerate le Olimpiadi di Londra del 2012. Mentre era possibile prevedere la frequentazione di un particolare evento, non c'era modo di prevedere come le persone si muovessero, mettendo in stress molte risorse della città. Crowd-Watch [10] è un sistema collaborativo basato sugli smartphone; esso permette a servizi di monitoraggio e di feedback, di monitorare il comportamento e le dinamiche delle persone all'interno della folla. Anche se Londra è nota per il sistema di fotocamere, le informazioni raccolte con esso richiedono una complessa e costosa elaborazione delle immagini [11] e quindi non può essere utilizzato per reagire agli eventi in tempo reale. Il Crowd-Sourcing [12] e il Crowd-Computing [13][14] sono utilizzati per sfruttare le informazioni raccolte dagli individui all'interno della folla, ma in momenti di emergenza o sovraffollamento [15] l'accesso al server tramite la rete cellulare, potrebbe essere molto inaffidabile e quindi questi dati potrebbero non essere resi disponibili ai server.

CrowdWatch riesce a raggiungere l'efficienza energetica e un monitoraggio di folla distribuito, sfruttando tre osservazioni fondamentali. In primo luogo le folle tendono a concentrarsi in gruppi, pertanto non è necessario che tutti gli utenti raccolgano dati in ogni momento. In secondo luogo considerando la dinamicità delle affluenze, si è notato che evolvono in continuazione; pertanto non è necessario avere un'istantanea perfetta della folla in ogni momento. In terzo luogo, la maggior parte dei dispositivi mobili attuali possiedono più sistemi radio come Wifi e Bluetooth.

CrowdWatch costruisce una gerarchia di dispositivi, sfruttando la lontananza Wifi per la creazione di una dorsale dinamica tra la folla e la connettività Bluetooth, per sostenere il campionamento e la raccolta dei dati.

Attualmente i ricercatori, sono in fase di progettazione di applicazioni che utilizzeranno l'architettura CrowdWatch; esse permetteranno di studiare le dinamiche di interazione interpersonale in una scala senza precedenti.

2.2 Participatory Sensing Application

Possiamo suddividere le tipologie di applicazioni di Participatory Sensing in tre filoni: incentrate sulle persone (people-centric), incentrate sull'ambiente (environment-centric) e miste. Le applicazioni incentrate sulle persone, si focalizzano sulla documentazione di attività (viaggi, attività sportive) e sulla comprensione del comportamento degli individui (disordini alimentari, diete). Le applicazioni incentrate sull'ambiente, raccolgono informazioni e parametri dall'ambiente circostante (qualità dell'aria, inquinamento acustico). Mentre le applicazioni miste uniscono alcuni aspetti dalle due categorie precedenti. In questa sezione sono presentate alcune applicazioni rappresentative per ognuna delle due categorie.

2.2.1 Applicazioni Incentrate sulle Persone

Le applicazioni incentrate sulle persone, utilizzano i sensori integrati nei dispositivi mobili, per la raccolta di informazioni che riguardano l'utente.

Monitoraggio dello Stato di Salute

Nel monitoraggio dello stato di salute, i dispositivi mobili sono utilizzati per monitorare lo stato psicologico e di salute dei pazienti/partecipanti. I dispositivi utilizzano sensori integrati o esterni. (es. accelerometro indossabile, o sensore d'inquinamento).

DietSense [16] assiste i partecipanti che vogliono perdere peso. Si monitorano i piatti dell'utente, sistemando il dispositivo mobile attorno al collo dei partecipanti. Il dispositivo raccoglie automaticamente immagini riguardo ai piatti che l'utente si appresta a mangiare. Le immagini raccolte permettono di estimare il peso del cibo e la parte non mangiata. Inoltre il dispositivo mobile raccoglie informazioni riguardo al contesto del pasto registrando l'ora, le coordinate geografiche e i suoni registrati per dedurre relazioni fra i partecipanti e il comportamento nel contesto. I dati sono inseriti in un archivio personale, dove l'utente può rivederli, selezionarli, eliminarli oppure condividerli con i propri medici e nutrizionisti.

Bewell [17] è un'applicazione android che utilizza il modello BES [18]. Il modello BES stima la durata del sonno dell'utente, sfruttando i dati di utilizzo del telefono e le statistiche raccolte dagli stessi utenti tramite Bewell. In particolare, BES tiene traccia su base giornaliera, della durata complessiva di tempo, in cui il telefono è in uno di questi stati: bloccato, spento, in carica, al buio o in un ambiente silenzioso. Tutti i dati necessari, sono raccolti esclusivamente da sensori incorporati nel telefono, senza intervento dell'utente. BES presume che vi sia un rapporto statistico, tra la durata del sonno dell'utente e il periodo che il telefono rimane al buio. BES tenendo conto di tutte le caratteristiche descritte in precedenza, dedurrà la durata del sonno dell'utente. Esistono altre applicazioni per monitorare il sonno; esse utilizzano sensori esterni, come ad esempio JawBone UP [19], che si serve di un bracciale connesso al dispositivo tramite bluetooth.

PEIR (Personal Environmental Impact Report) [20] è un sistema che permette agli utenti di utilizzare il proprio telefono cellulare, per determinare l'esposizione agli agenti inquinanti nell'ambiente. Utilizzando le informazioni sulla posizione dell'utente e sulle modalità di spostamento, il server fornisce agli utenti le informazioni sull'impatto ambientale del loro viaggio, in termini di emissione di particelle e agenti inquinanti. Raccoglie inoltre dati aggiuntivi, dalle stazioni metereologiche e dai servizi di traffico.

Monitoraggio delle attività sportive

BikeNet [21] è un sistema di monitoraggio degli utenti durante le attività ciclistiche. Esso traccia il profilo di ogni ciclista, tramite la misurazione della posizione e della risposta galvanica della pelle. Sono utilizzati una moltitudine di sensori per la raccolta di informazioni: microfono, magnetometro, sensore di velocità della pedalata, inclinometro, lateral tilt, GSR stress monitor e sensori di concentrazione di anidride carbonica. I sensori periferici comunicano tramite una connessione senza fili e i dati catturati possono essere rivisti, uniti con altri partecipanti o combinati con parametri addizionali come la qualità dell'aria e le condizioni del traffico, in modo da poter costruire mappe per la comunità dei ciclisti.

Runtastic [22] è un'applicazione che monitora il percorso dell'utente e permette di salvare e condividere questi percorsi. È inoltre possibile incitare le persone che stanno facendo attività sportiva. Runtastic può essere collegato a sensori esterni per il monitoraggio del battito cardiaco.

RunKeeper [23] è un'applicazione che aiuta a monitorare un percorso, permette di scegliere dei piani di allenamento, e persino di "noleggiare" un trainer. RunKeeper è associabile al braccialetto di JawBone [19] per il monitoraggio dei valori corporei.

Monitoraggio dei Viaggi

EveryTrail [24] è un'applicazione che permette all'utente di cercare o creare guide per viaggi. Le guide create con quest'applicazione comprendono la direzione precisa del viaggio e le foto che l'utente creatore ha deciso di scattare durante la registrazione del percorso. È quindi possibile seguire un percorso, visionando le foto e i commenti; alcuni percorsi sono paragonabili a vere e proprie guide turistiche. EveryTrail utilizza un sistema di reputazione gestito direttamente dagli utenti tramite il classico sistema a cinque stelle.

Figura 2.1: EveryTrail per iOS7

Miglioramento dei Social Media

Sono molte le applicazioni che utilizzano i dati acquisiti dai sensori per arricchire i contenuti che sono condivisi nei social. CenceMe [6] raccoglie informazioni dai dispositivi mobili (accelerazione, campioni audio, immagini, posizione), questi dati sono poi utilizzati per dedurre informazioni sull'attività svolta in un determinato ambiente, sull'umore e le abitudini. L'informazione dedotta è poi tradotta e condivisa come rappresentazione dei partecipanti nel mondo virtuale.

Raccolta dei Prezzi

PetrolWatch [25] è un sistema che automatizza la raccolta dei prezzi, usando le telecamere dei dispositivi mobili. Il dispositivo è montato nel seggiolino del

passeggero e viene rivolto verso la strada, per fotografare automaticamente i prezzi nei cartelli delle stazioni di rifornimento, quando il veicolo è nelle vicinanze (GPS e GIS). L'immagine è inviata al server che la elabora tramite algoritmi di computer vision, che estrapolano il prezzo. Gli utenti possono quindi richiedere al database, quale sia il distributore più vicino con prezzo inferiore.

Monitoraggio delle Stazioni di Ricarica

PlugShare [26] permette di cercare le stazioni di ricarica più vicine, permette inoltre di impostare la tipologia di motore che si deve ricaricare, in modo che possa essere cercata una stazione adeguata. Gli utenti possono anche aggiungere stazioni di ricarica sia pubbliche sia private (nelle abitazioni). PlugShare notifica l'utente, non solo della presenza delle stazioni, ma anche della loro disponibilità.

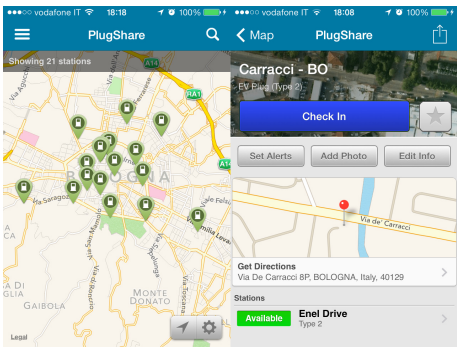


Figura 2.2: PlugShare per iOS7

2.2.2 Applicazioni Incentrate sull'Ambiente

Negli scenari incentrati sull'ambiente, i dispositivi mobili catturano le informazioni tramite sensori integrati e sensori esterni per l'analisi dell'ambiente circostante. A differenza dello scenario incentrato sulle persone, i dati acquisiti sono utilizzati principalmente su scala comunitaria, ad esempio per

monitorare alcuni parametri ambientali come la qualità dell'aria, l'inquinamento acustico, le condizioni di traffico o per rilevare eventi socialmente interessanti.

Monitoraggio Qualità dell'Aria

HazeWatch [27] è un'applicazione dove i dispositivi mobili sono collegati con sensori d'inquinamento esterni per misurare la concentrazione del monossido di carbonio, ozono, diossido di zolfo e diossido di azoto nell'aria. Comparati alle stazioni metereologiche, i dispositivi mobili possono raccogliere dati meno accurati; è però possibile raccogliere dati inaspettati come l'inquinamento accidentale. I dati sono salvati insieme al timestamp e alla geo-localizzazione, sono poi aggregati con dati affidabili (stazioni metereologiche) e resi disponibili a tutti gli utenti.

Air Quality Egg [28] è principalmente una comunità che condivide la passione per il rilevamento degli agenti inquinanti. Ognuno degli utenti possiede un insieme di sensori a forma di uovo che connesso a internet, invia i dati direttamente alla comunità. Nonostante non siano molti i dispositivi attualmente connessi, l'idea è certamente interessante perché se si riuscisse ad attivare il mondo delle persone interessate, si potrebbero unire i dati rilevati dalle stazioni adibite a quelli provenienti dagli utenti affidabili.

Monitoraggio dati Ambientali

Weendy [29] è un'applicazione che permette agli utenti, di notificare le condizioni del vento nella zona in cui si trovano. È utilizzata principalmente da chi pratica sport estremi come windsurf e surf; la comunità è molto presente anche in Italia.

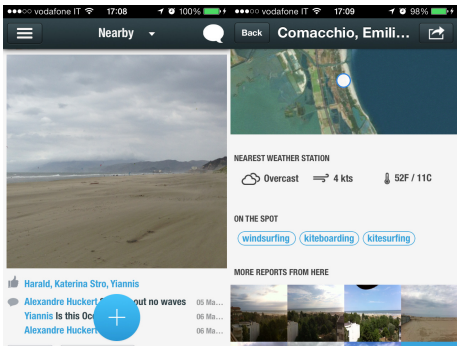


Figura 2.3: Weendy per iOS7

La cosa che rende quest'applicazione affidabile, è il collegamento con alcune stazioni di rilevazione; si unisce quindi l'affidabilità all'ubiquità. Weendy utilizza un sistema di reputazione gestito direttamente dagli iscritti; se un utente notifica una buona condizione di vento per fare attività sportiva, gli altri possono decidere di votare questa segnalazione e far guadagnare punti a quell'utente. Inoltre è possibile seguire un determinato utente, in questo modo a ogni sua segnalazione di vento si verrà notificati.

Monitoraggio del Rumore

I microfoni all'interno dei dispositivi mobili, permettono di monitorare il livello di rumore e raccogliere informazioni sull'ambiente.
EarPhone [5], utilizza i microfoni per monitorare l'inquinamento acustico che può influenzare le capacità di ascolto e il comportamento. È creata una mappa, resa successivamente accessibile agli specialisti per capire le relazioni fra l'esposizione al rumore e i problemi comportamentali delle persone.

Monitoraggio della Strada e delle Condizioni di Traffico

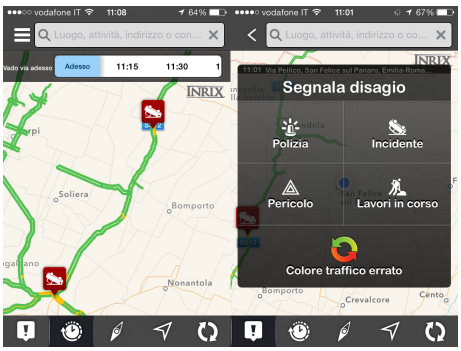


Figura 2.4: INRIX Traffic per iOS7

I dispositivi mobili possono essere utilizzati, per analizzare i problemi della strada e del traffico.

Nericell [30] utilizza i sensori integrati come accelerometro, microfono e sistemi di posizionamento (GPS, GSM, Wifi) per analizzare e localizzare le condizioni del traffico e del manto stradale. L'applicazione integra le informazioni di traffico, della superfice stradale e del rumore, per renderle disponibili alle persone.

INRIX Traffic [31] è un'applicazione che unisce l'affidabilità delle rilevazioni effettuate con telecamere per il monitoraggio della viabilità, alle segnalazioni inviate direttamente dagli utenti. Gli utenti che utilizzano INRIX per navigare o per programmare una partenza, aiutano la comunità a evitare condizioni di traffico insostenibili. INRIX utilizza un algoritmo dinamico che muta insieme alle condizioni del traffico. Con INRIX è possibile segnalare un disagio nelle condizioni di traffico come un incidente, un pericolo, dei lavori in corso oppure un colore di traffico errato in base alla reale condizione. INRIX utilizza un sistema di reputazione, dove si guadagnano punti quando le segnalazioni sono confermate da altri utenti della comunità.

Monitoraggio delle Reti

OpenSignal [32] è un'applicazione che permette di visualizzare i dati di ricezione della rete cellulare. Contiene i dati sulla posizione delle celle, e in background rileva i dati di ricezione direttamente dall'applicazione, senza avvalersi dell'utente. I dati sono abbastanza aggiornati e realistici, è però possibile notare che la maggior parte delle informazioni arriva dalle strade; questo fa pensare che il sistema utilizzi anche altri sistemi di rilevazione (Figura 2.5b).

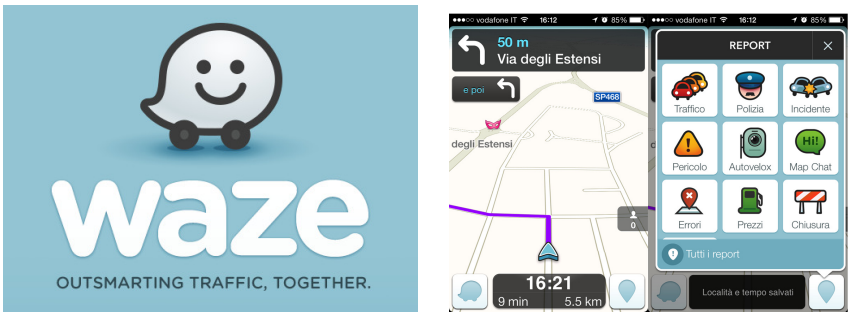
(a) (b)

Figura 2.5: OpenSignal per iOS7.

2.2.3 Applicazioni Miste

È stato necessario creare una terza sezione per quest'applicazione che include funzionalità incentrate sia sull'ambiente sia sulle persone.

Waze [33] è un applicazione di navigazione stradale dove gli utenti si scambiano informazioni. Gli utenti possono inviare dei report specifici sulla condizione della navigazione stradale: Traffico, Polizia, Incidente, Pericolo sulla strada, Autovelox, Prezzi del Carburante, Chiusura di una strada e messaggi di chat per gli altri utenti. L'applicazione è multipiattaforma e ben ottimizzata (Cap. 6.2.1).



(a) (b)

Figura 2.6: Waze per iOS7.

L'utente si può registrare tramite Facebook, può vedere gli amici connessi e in alternativa può invitarli. Una volta registrato al Sistema, l'utente è rappresentato dal suo *Waze*: una sorta di avatar a forma di automobile stilizzata (Figura 2.6a). L'utente appena registrato è limitato in alcune opzioni, come ad esempio l'invio di un report sui prezzi del carburante. Il proprio *Waze*, in altre parole ciò che siamo all'interno l'applicazione, cresce man mano che l'utente naviga e così facendo si sbloccano funzionalità.
Waze non è solo un applicazione, ma un vero e proprio Sistema, accedibile anche tramite computer. Tramite L'Editor Web è possibile fare modifiche alle mappe, limitando l'utente alle zone dove ha guidato. L'editor permette di inserire una strada, una rotonda, un parcheggio, una stazione di servizio o un generico landmark.
Quindi riassumendo Waze è: un social, è un modo per evitare il traffico ed è anche un modo per segnalare una serie di informazioni che sono prettamente incentrate sulle persone.

Tabella 2.1: Tabella Riepilogativa delle Applicazioni

NOME	TIPOLOGIA	DESCRIZIONE
DietSense [16]	P	Permette agli utenti interessati al dimagrimento, di monitorare le calorie dei propri piatti.
Bewell [17]	P	Stima la durata del sonno degli utenti. https://bewellapp.org/
JawBone UP [19]	P	Utilizza un braccialetto per monitorare il battito cardiaco dell'utente; può stimare la durata del sonno. https://jawbone.com/up
PEIR [20]	P	Determina l'esposizione degli utenti agli agenti inquinanti nell'ambiente.
BikeNet [21]	P	Utilizza sensori integrati e esterni, per il monitoraggio degli utenti durante le attività ciclistiche. http://www.tacx.com/en/experience/bikenet
Runtastic [22]	P	Monitora il percorso dell'utente durante le attività fisiche. Può essere associata a sensori esterni. https://www.runtastic.com/it/
RunKeeper [23]	P	Monitora il percorso dell'utente e aiuta durante le sessioni di allenamento, tramite i consigli di trainer specializzati. http://runkeeper.com/
EveryTrail [24]	P	Permette di creare guide per viaggi e renderle accessibili agli altri utenti della comunità. http://it.everytrail.com/
CenceMe [6]	P	Raccoglie informazioni dai dispositivi mobili e li analizza per dedurre le attività degli utenti. http://metrosense.cs.dartmouth.edu/
PetrolWatch [25]	P	Automatizza la raccolta dei prezzi tramite la fotocamera dei dispositivi.
PlugShare [26]	P	Ricerca le stazioni di ricarica per le autovetture con motore elettrico. http://www.plugshare.com/
HazeWatch [27]	A	Raccoglie informazioni sugli agenti inquinanti utilizzando sensori esterni. http://www.pollution.ee.unsw.edu.au/
Air Quality Egg [28]	A	Rileva la qualità dell'aria tramite un dispositivo a forma di uovo contenente sensori. http://airqualityegg.com/
Weendy [29]	A	Permette di ricevere e inserire notifiche riguardanti le condizioni di vento. http://www.weendy.com/
EarPhone [5]	A	Rileva il livello di rumore tramite il microfono integrato del dispositivo.
Nericell [30]	A	Analizza le condizioni del manto stradale e del traffico, utilizzando i sensori interni del dispositivo. http://research.microsoft.com/en-us/projects/nericell/
INRIX Traffic [31]	A	Permette di evitare il traffico stradale; utilizza i dati provenienti dalle telecamere per il monitoraggio della viabilità e le segnalazioni ricevute dagli utenti. http://www.inrixtraffic.com/
OpenSignal [32]	A	Permette di visualizzare i dati di ricezione della rete cellulare. http://opensignal.com/
Waze [33]	M	È un applicazione di navigazione stradale dove gli utenti si scambiano informazioni riguardanti la viabilità. https://www.waze.com/it/

P (Incentrata sulle Persone) - A (Incentrata sull'Ambiente) - M (Mista)

Capitolo 3

Problema Affrontato e Metodologia Utilizzata

Questo capitolo si pone come obiettivo, la presentazione dei problemi e delle soluzioni che sono state utilizzate per lo sviluppo del Sistema.

Si voleva realizzare un Sistema di monitoraggio della situazione delle nebbie, che offrisse agli Utenti le seguenti funzionalità:

- Segnalazione dello stato di Nebbia tramite Geo-Tag;
- Creazione di un percorso alternativo.

3.1 Segnalazione della Visibilità

L'Utente deve poter scegliere uno stato di visibilità da segnalare; successivamente l'applicazione si occupa di richiedere al device la posizione; in caso di successo il dato di posizione e il dato ambientale di nebbia sono inviati al server. Il Client in seguito attende la risposta dal Server e ne comunica il responso all'Utente.

3.2 Ricerca Percorso Alternativo

L'Utente sceglie una posizione di partenza, che può essere l'attuale e una posizione di arrivo. Il Client richiederà lo stato delle nebbie in una zona limitata e dopo aver ricevuto questi dati dal Server, li analizzerà per generare un percorso alternativo.

La ricerca di un percorso alternativo è in realtà un macro problema. Per trovare un buon percorso alternativo, è stato utilizzato un approccio *Greedy*; sono quindi utilizzate alcune soluzioni approssimate.

3.2.1 Richiesta Percorso Ottimale

Il Client innanzitutto richiede il percorso ottimale direttamente tramite l'oggetto *MKDirections*, che fornisce indicazioni stradali dai server di Apple. È possibile utilizzare le istanze di questa classe per ottenere informazioni di viaggio in termini di tempo o di direzione. L'oggetto passa poi la richiesta ai server Apple e restituisce in modo asincrono le informazioni richieste in oggetti chiamati *directions*.

3.2.2 Griglia dei Punti: Costruzione di un Grafo

Una volta che la direzione ottimale è stata calcolata, è necessario creare una griglia di punti, che ci permetta in seguito di calcolare il percorso alternativo. Immaginiamo la superficie della mappa come a uno spazio in due coordinate x e y, che rappresentano rispettivamente longitudine e latitudine; è necessario avere dei riferimenti nello spazio che ci permettano di trovare un cammino per evitare le nebbie. Purtroppo non abbiamo un grafo completo di tutte le direzioni esistenti nello spazio reale (Mappa), quindi l'idea è di creare una griglia con punti equidistanti fra loro. In una prospettiva di risparmio di risorse, questa griglia non può essere infinitamente grande, ma deve essere delimitata da alcuni punti che comprendano il percorso che è stato in precedenza calcolato (percorso ottimale).

Figura 3.1: Inserimento dei quattro margini esterni.

A questo punto si creano quattro punti nello spazio delle coordinate latitudine e longitudine, che rappresentano gli estremi superiori e inferiori di un rettangolo che farà da contenitore per la griglia dei punti (Figura 3.1). Sulla base dei quattro punti sul piano, è creato uno strato di punti a intervalli regolari, che ricopre tutto il rettangolo. (Figura 3.2).

Un *Punto di Controllo* è un punto geo-localizzato al quale è associato un valore di riga e di colonna, in modo da essere riconoscibile sia come punto nello spazio, sia come elemento di una griglia. Un *Punto di Controllo*, ha determinate proprietà di adiacenza con altri *Punti di Controllo*.

Quest'astrazione permette di separare il concetto di richiesta del dato ambientale per un determinato punto geografico, dal concetto di grafo costruito tramite la griglia.

Figura 3.2: Inserimento dei *Punti di Controllo*.

Questo processo non è privo di vizi, infatti, i punti della griglia non corrispondono quasi mai a punti appartenenti a una strada e nemmeno ai punti di partenza e arrivo che sono stati inseriti. Il primo problema è risolto con la correzione del percorso (Cap. 3.6) mentre il secondo problema è risolto inserendo i punti di partenza e arrivo, tramite un processo di sostituzione per vicinanza a uno degli altri punti appartenenti alla griglia.

A questo punto abbiamo costruito il nostro grafo di *Punti di Controllo* , con il quale sarà generata una richiesta al Server.

Figura 3.3: Processo di sostituzione con i punti di Partenza e Arrivo

3.3 Richiesta dei Dati Ambientali

In questa fase il Server riceve una richiesta di dati ambientali, contenente tutti i punti della griglia; per ognuno di questi punti interroga il Database circa l'esistenza di segnalazioni di visibilità nelle vicinanze. Il parametro di vicinanza è a discrezione del proprietario del Server, ma di default è impostato a 1000 metri. Il Database restituirà solo un valore di visibilità, quello più alto fra i trovati.

A questo punto il Server provvederà a generare una risposta contenente tutti i punti della griglia, dove a ognuno di essi è associato un valore di visibilità che va da 0 a 5:

0. **Limpido**: Nessuna presenza di nebbie;
1. **Caligine**: Visibilità superiore ai 10 Km;

2. **Foschia**: Visibilità compresa fra 1 e 10 Km;
3. **Nebbia spessa**: Visibilità fino a 200 mt;
4. **Nebbia fitta**: Visibilità compresa fra 30 e 50 mt;
5. **Nebbia densa**: Visibilità inferiore a 30 mt.

3.4 Ricerca nel Grafo : Dijskra

In questa fase il Client riceve una griglia di punti, dove a ognuno di essi è associato un valore di visibilità che sarà utilizzato per calcolare il percorso ottimale attraverso i *Punti di Controllo*.

Algoritmo di Dijskra

L'algoritmo di Dijskra è utilizzato per cercare i cammini minimi in un grafo con o senza ordinamento, ciclico e con pesi non negativi sugli archi [34]. Quest'algoritmo permette in ogni caso, di trovare almeno una soluzione. È però necessaria un'astrazione per utilizzarlo in un contesto di un cammino con *"nebbia minima"*.

Dato un insieme di *Punti di Controllo* PC, dati due nodi $\alpha, \beta \in PC$, dato il peso sull'arco di connessione $P(\alpha, \beta)$ e i rispettivi valori di visibilità $V(\alpha)$ e $V(\beta)$ avremo:

$$P(\alpha, \beta) = max(V(\alpha), V(\beta)) \text{ [Figura 3.4]}$$

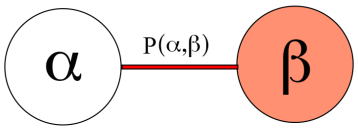


Figura 3.4: Valutazione del parametro di visibilità.

Siamo quasi a buon punto, ma manca ancora un fattore: *Il peso per la distanza*. Bisogna quindi riformulare leggermente l'assunzione precedente, considerando che il peso di un nodo deve essere influenzato dalla distanza.

Dato un insieme di *Punti di Controllo* PC, dati tre nodi $\alpha, \gamma, \delta \in PC$, dati i pesi sugli archi di connessione $P(\alpha, \gamma), P(\alpha, \delta)$, i loro rispettivi valori ambientali $V(\alpha), V(\gamma), V(\delta)$. Assumendo che la distanza fra α e $\gamma = 1$, che i punti siano dislocati su una griglia, che γ sia diagonale rispetto a α, e che δ sia alla destra di α possiamo quindi calcolare i pesi degli archi di connessione:

Utilizziamo la formula della diagonale di un quadrato per calcolare la distanza tra il punto α e il punto γ: $D(\alpha, \gamma) = 1 * \sqrt{2} \Rightarrow 1.4142$

quindi:

$$P(\alpha, \gamma) = max(V(\alpha), V(\gamma)) + 1.4142$$
$$P(\alpha, \delta) = max(V(\alpha), V(\delta)) + 1$$

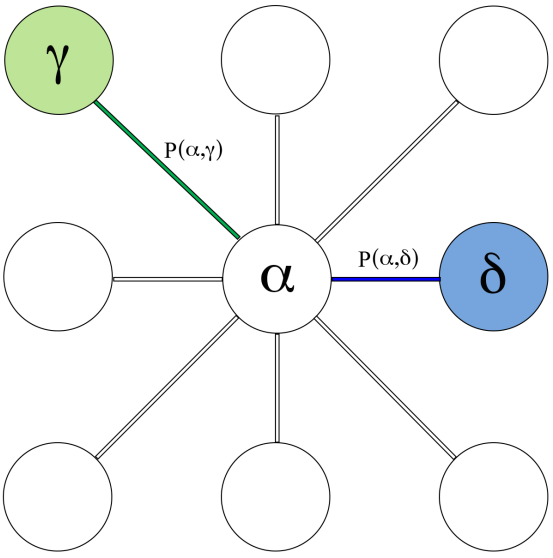


Figura 3.5: Punti di controllo sulla griglia.

Innanzitutto ogni *Punto di Controllo* è inserito come nodo facente parte del grafo di soluzione, poi per ognuno di questi nodi sono calcolati i pesi degli archi di connessione ai nodi adiacenti con il metodo appena trovato.

Ora abbiamo tutti i dati necessari per usare l'algoritmo di Dijskra in modo corretto.

3.5 Correzione della Soluzione

In questa fase è necessaria una correzione dei punti di soluzione trovati dall'algoritmo di Dijskra. Dato che i *Punti di Controllo* non sono creati su un grafo stradale preciso, è necessario eliminare il comportamento di *"Allontanamento"* che si è verificato in fase di sviluppo (Figura 3.6).

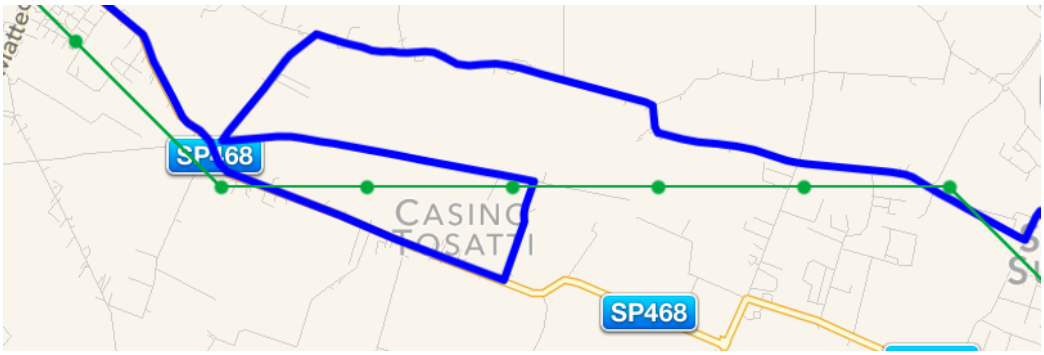
Figura 3.6: Allontanamento parziale dalla meta.

Fin dal primo momento in cui si è verificato l'errore, era chiaro come le posizioni dei *Punti di Controllo* fossero il vero problema; ma com'è possibile risolverlo? Bisogna innanzitutto, distinguere le due tipologie di errore riscontrate: *"Allontanamento"* e *"Inversione"*:

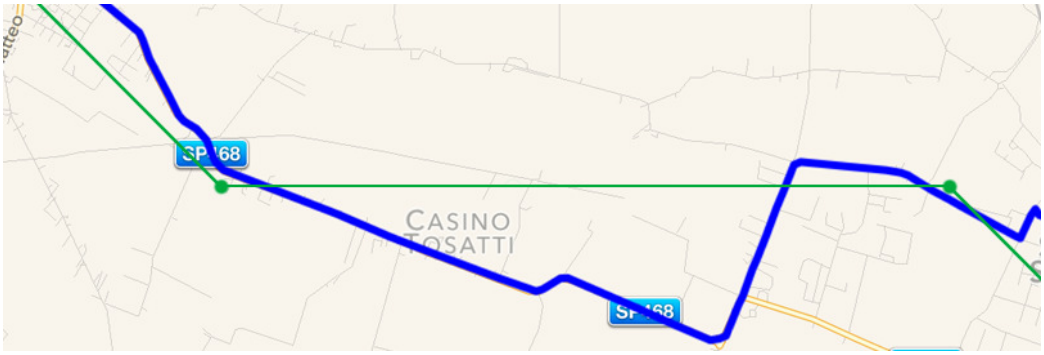
Allontanamento : per allontanamento si definisce un comportamento di variazione di direzione, che non è conforme alla soluzione, né tantomeno rispetto la direzione di arrivo.

Inversione : per inversione s'intende una variazione di direzione, che viene interrotta per ritornare sul percorso precedente (*inversione a U*).

Una soluzione è composta di un insieme di *Punti di Controllo*, che collegati tra loro, formano delle rette di direzione. I punti che contengono le rette di direzione, causano il problema dell'Allontanamento, questo perché nelle richieste di *directions* dai server Apple (Cap. 3.2.1), si richiede di passare per ogni punto che compone la retta (Figura 3.7a). La soluzione sta nell'eliminare tutti i punti che sono *"inutili"* per la soluzione, lasciando solo il punto iniziale e finale della retta di direzione (Figura 3.7b).



(a) Direzione con tutti i punti di soluzione della griglia.



(b) Direzione con i punti di soluzione interni alle rette eliminati.

Figura 3.7: Comparazione delle direzioni, prima e dopo la correzione

La direzione calcolata con meno punti di soluzione, è non solo più corretta, ma anche più veloce per le minor richieste effettuate ai server Apple. Come

si può vedere nell'immagine comparativa (Figura 3.7), dopo aver inserito l'algoritmo di eliminazione dei punti interni alla retta, il problema è stato arginato. La soluzione risulta anche più realistica "in termini automobilistici"; per via del percorso maggiormente ottimizzato in termini di distanza e tempo di percorrimento.

3.6 Correzione del Percorso

In questa fase le direzioni fra i punti della soluzione, sono state ricevute dal server Apple, ed è necessario correggere un ulteriore problema sempre dovuto alla natura dei punti della soluzione: Il problema dell'*"Inversione"* (Cap. 3.5). Per risolvere il problema, è necessario capire come funziona il Sistema di direzione dei server Apple.

Una volta inviate le richieste per ogni coppia di punti soluzione, il Server Apple risponde asincronamente con oggetti chiamati *directions*, in questi oggetti sono contenute, oltre alle informazioni utili per la navigazione, anche le coordinate dei punti necessari a disegnare la direzione sulla mappa. Dato che le richieste delle direzioni sono asincrone, è necessario attendere che tutte le soluzioni siano arrivate, e ricomporre la strada in ordine, partendo dalla partenza fino all'arrivo.

Ora è possibile notare che nel momento in cui avviene un'Inversione, alcune di queste coordinate sono doppie, bisogna quindi eliminarle dall'insieme dei punti di direzione pre-calcolati.

Immaginiamo un cammino composto da 6 nodi, e immaginiamo che il cammino passi per i nodi in quest'ordine: 1-2-3-4-5-4-3-6 (Figura 3.8). Notiamo subito che il cammino sarebbe più corto, se si evitasse di passare per i nodi 4 e 5. In questo modo otteniamo un insieme di coordinate corrette, che possono essere usate per disegnare sulla mappa, una strada completamente priva d'inversioni.

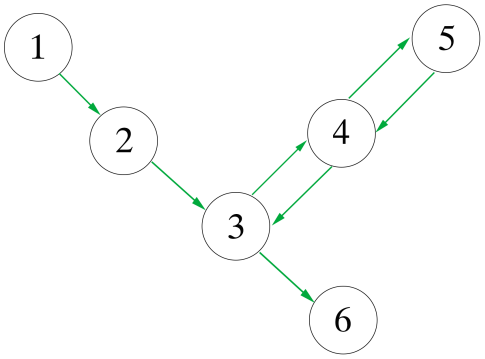


Figura 3.8: Esempio di inversione.

3.7 Struttura del Sistema

Il Client è composto dall'applicazione *Fog Escaping*, che permette ai dispositivi iPhone e iPad, di comunicare con il Server tramite protocollo HTTP. Il Database contiene tutte le informazioni riguardanti gli utenti e le loro segnalazioni. Il Server riceve le richieste e interrogando il Database, risponde ai dispositivi.

Figura 3.9: Il Sistema Fog Escaping.

Capitolo 4

Implementazione

In questo capitolo saranno introdotti i dettagli dell'implementazione del Sistema. Nella prima sezione sarà introdotta l'architettura di sistema e il protocollo di comunicazione. Nella seconda sezione saranno introdotti i dettagli d'implementazione del Server HTTP e del Database. Nella terza e ultima sezione saranno introdotti i dettagli d'implementazione del Client.

4.1 Architettura del Sistema

Il Sistema presenta un'architettura Client-Server; il Client è un applicativo per dispositivi Apple sia iPhone che iPad, mentre il Server è composto da un componente web sviluppato in PHP che s'interfaccia a un Database MySQL per la richiesta e la memorizzazione dei dati.

Il Client può operare sul Server solo se l'Utente completa correttamente il processo di *Registrazione* e *Login*. Una volta *autenticato*, il Client potrà utilizzare i servizi per segnalare e richiedere i dati ambientali di Nebbia.

4.1.1 Client

L'applicazione che è stata sviluppata per il progetto di Tesi, si chiama *Fog Escaping*. Il nome permette immediatamente d'intuire il compito principale dell'applicazione; cioè *"fuggire dalla nebbia"*. *Fog Escaping* permette a un

Utente di registrarsi, loggarsi, inviare rapidamente una segnalazione di nebbia e calcolare un percorso alternativo da una posizione di partenza a una di arrivo. Le funzionalità di Fog Escaping, gli permettono in via teorica, di autosostenersi; infatti, il problema principale delle applicazioni con paradigma Data Crowd-Sourcing, è *la massa critica*: la presenza di sufficienti utenti attivi, che permettano al sistema di possedere abbastanza dati capillari da farlo funzionare in modo corretto. *Fog Escaping* ha alcuni requisiti di funzionamento legati alla tecnologia utilizzata:

1. Dispositivo iPhone 4 o superiori
2. Dispositivo iPad 3a generazione o superiore
3. Sistema operativo iOS7

L'ambiente di sviluppo utilizzato per la parte Client, è *XCode 5*, software proprietario di Apple Inc.

4.1.2 Server

La parte PHP, è composta da alcuni servizi web, che rispondono a determinate richieste, e da un componente che gestisce le connessioni a una base di dati MySQL.

Il Server è attivo al seguente URL:

http://fogescaping.esy.es/request.php?service=nomeServizio*
Il nomeServizio* può essere uno dei quattro resi disponibili dal Server:

- registration;
- login;
- insertFog: Invio Dato Ambientale di Nebbia;
- checkForFogValues: Richiesta Dati di Nebbia;

I requisiti della parte server sono:

1. Server HTTP Apache, versione 2.2.25 o superiore;
2. PHP versione 5.5.3;
3. PDO drivers, in particolare per MySQL 5.5.33 o superiore;
4. MySQL 5.5.33 o superiore

Gli ambienti di sviluppo utilizzati per la parte Server, sono MySQLWorkbench e PHPStorm, rispettivamente per la base di dati e il sorgente PHP.

4.1.3 Protocollo di Comunicazione

Tra il Client e il Server, esiste uno scambio bidirezionale di dati in formato JSON, attraverso il protocollo di comunicazione HTTP. Si può testare il funzionamento di ogni servizio, utilizzando uno strumento aggiuntivo di Chrome, chiamato Advanced Rest Client. Le richieste HTTP, sono progettate secondo l'architettura REST, e per ognuna di esse è indicato:

- il metodo HTTP da utilizzare;
- la struttura dell'URL, i parametri e gli esempi;
- il Body della richiesta;
- l'Header e il Body della risposta.

Risposta Generica di Errore Dati

Il Server provvede a inviare un JSON di risposta generico per Body con dati mancanti o errati.

```
{
  message: "tutti i campi sono richiesti",
  request_time: 1392212902
}
```

Listing 4.1: JSON Generico di Errore.

Registrazione

La registrazione richiede all'Utente, l'inserimento di alcuni dati che possano riconoscerlo univocamente: eMail, Nome, Cognome e Password. Il Server, attuando un controllo sull'e-mail, si assicura che l'Utente non sia già registrato nel Database, e in seguito provvede a inserirlo.

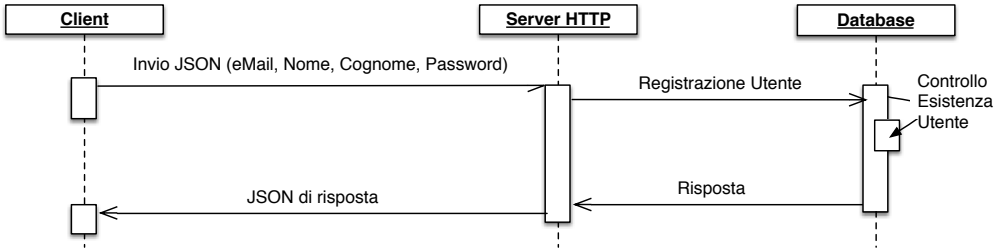


Figura 4.1: Diagramma di Sequenza: Registrazione Utente.

Richiesta:

Metodo: Post

URL: http://fogescaping.esy.es/request.php?service=registration

Body: JSON di Registrazione (Listing 4.2)

Header: Accept: application/json | Content-type:application/json

```
{
  "email":"federico.maccaferri2@studio.unibo.it",
  "name":"Federico",
  "lastname":"Maccaferri",
  "password":"federico"
}
```

Listing 4.2: JSON di Registrazione.

I parametri devono tutti essere presenti e non nulli. Nel caso che l'Utente sia già registrato presso il Database, il Server provvederà a inviare un JSON

adeguato (Listing 4.4). Nel caso di errore nei dati, il Server restituirà un JSON di errore generico (Listing 4.1).

Risposta:

Body: JSON di Registrazione (Listing 4.3-4.4)

Header: Content-type:application/json; charset=utf-8

Il parametro più importante del JSON di risposta è user_id; esso contiene le informazioni riguardanti l'ID di registrazione presso il Database.

La registrazione non avviene nel caso alcuni parametri non siano corretti, o quando l'Utente è già registrato. Un ritorno uguale a -1 significa che l'Utente è già registrato nel Database (Listing 4.4), altrimenti sarà restituito un JSON di corretta registrazione con l'user_id dell'Utente (Listing 4.3).

```
{
  message: "",
  request_time: 1392209379,
  user_id: "2"
}
```

Listing 4.3: JSON di Registrazione Corretta.

```
{
  message: "",
  request_time: 1392210112,
  user_id: "-1"
}
```

Listing 4.4: JSON di Registrazione Errata.

Login

Il Login è il processo che permette all'Utente di autenticarsi, e che permette al Client di attivare le opzioni di Segnalazione Nebbia e Ricerca Percorso Alternativo. Sono richiesti due parametri: eMail e Password.

Il Server, dopo la richiesta di Login, restituirà l'ID di registrazione dell'Utente al Database.

Il Client provvederà a salvare l'ID di Login, che verrà utilizzato per le operazioni di Segnalazione Nebbia e Calcolo Percorso Alternativo.

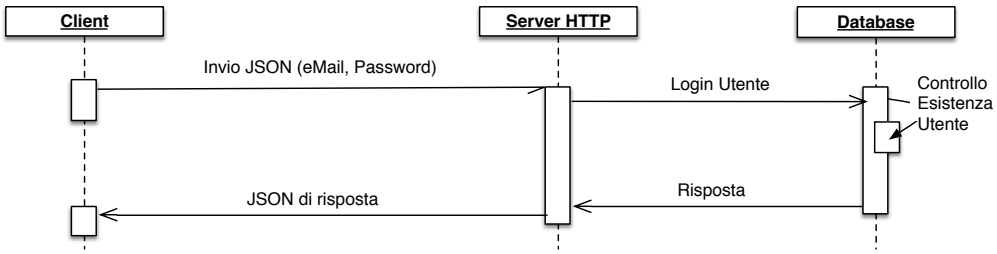


Figura 4.2: Diagramma di Sequenza: Login Utente.

Richiesta:

Metodo: Post

URL: http://fogescaping.esy.es/request.php?service=login

Body: JSON di Login (Listing 4.5)

Header: Accept: application/json | Content-type:application/json

```
{
  "email":"federico.maccaferri2@studio.unibo.it",
  "password":"federico",
}
```

Listing 4.5: JSON di Login.

I parametri devono tutti essere presenti e non nulli. Nel caso l'Utente sia già registrato presso il Database, il Server provvede a inviare un JSON di risposta adeguato (Listing 4.7). Nel caso di errore nei dati, il Server restituirà un JSON di errore generico (Listing 4.1).

Risposta:

Body: JSON di Risposta (Listing 4.6-4.7)

Header: Content-type:application/json; charset=utf-8

Il parametro più importante del JSON di risposta è user_id; esso contiene le informazioni riguardanti l'ID di registrazione presso il Database. Se questo parametro ritorna un valore uguale a -1, l'Utente non è registrato presso il Database (Listing 4.7), altrimenti è correttamente registrato (Listing 4.6).

```
{
  message: "",
  request_time: 1392211556,
  user_id: "2"
}
```

Listing 4.6: JSON di Login Corretto.

```
{
  message: "",
  request_time: 1392211913,
  user_id: "-1"
}
```

Listing 4.7: JSON di Login Errato.

Invio Dato Ambientale di Nebbia

L'Utente tramite l'interfaccia grafica del Client, seleziona la situazione di visibilità attuale (vedi Cap. 3.3), in seguito il Client richiederà la posizione dell'Utente e la segnalazione sarà inviata.

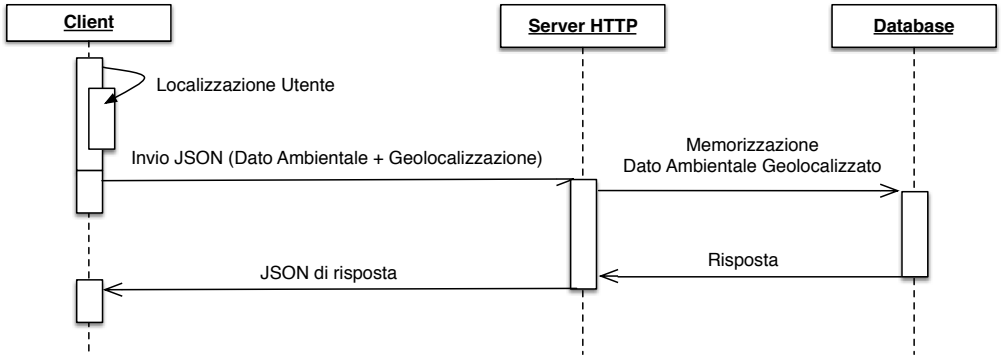


Figura 4.3: Diagramma di Sequenza: Invio Segnalazione.

Richiesta:

Metodo: Post

URL: http://fogescaping.esy.es/request.php?service=insertFog

Body: JSON d'Inserimento Dato Ambientale di Nebbia (Listing 4.8)

Header: Accept: application/json | Content-type:application/json

Com'è possibile vedere nel JSON di Segnalazione della Nebbia, i dati sono aggregati assieme al dato di posizione dell'Utente, alla condizione di visibilità (valore da 0 a 5) e all'ID dell'Utente, precedentemente restituito dal processo di Login.

```
{
  "userID":"1",
  "lat":"44.506484",
  "lng":"11.341968",
  "fogValue":"5"
}
```

Listing 4.8: JSON Segnalazione Nebbia.

I parametri devono tutti essere presenti e non nulli. Nel caso in cui la Segnalazione vada a buon fine, il Server restituirà un JSON adeguato (Listing 4.9).

Nel caso di errore nei dati, il Server restituirà un JSON di errore generico (Listing 4.1).

Risposta:

Body: JSON di Risposta (Listing 4.9)

Header: Content-type:application/json; charset=utf-8

Il parametro più importante del JSON di risposta è result; esso contiene true se l'inserimento è andato a buon fine, false se l'inserimento non è stato effettuato; quest'ultimo errore potrebbe avvenire per qualche errore d'inizializzazione del Server HTTP o del Database.

```
{
  message: "",
  request_time: 1392215028,
  result: true
}
```

Listing 4.9: JSON Risposta Segnalazione Nebbia.

Richiesta Dati di Nebbia

La richiesta dei dati di Nebbia, è strettamente legata alla funzionalità del Client di calcolare un percorso alternativo. Il Client invia un insieme di punti che rappresentano la griglia dei *Punti di Controllo* (Cap. 3.2) e attende i valori di visibilità dal Server, dopo che esso ha interrogato per ognuno dei *Punti di Controllo* il Database.

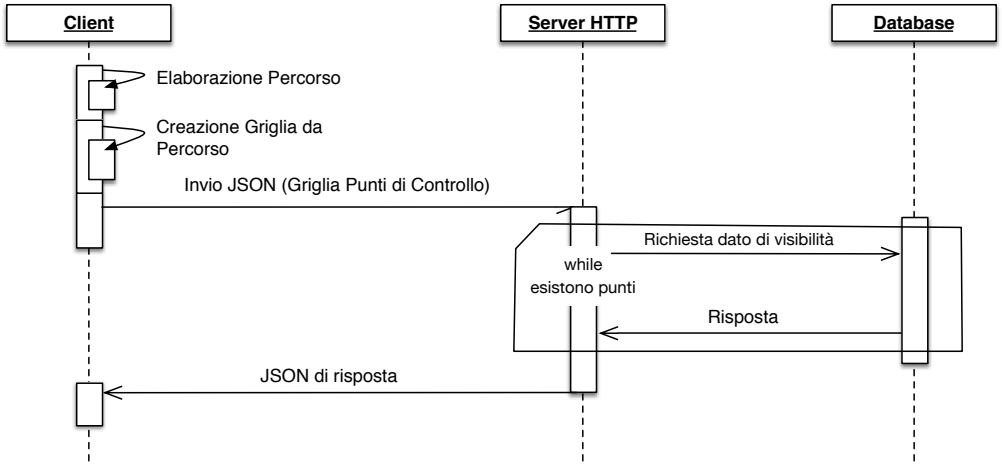


Figura 4.4: Invio Griglia di *Punti di Controllo*

Richiesta:

Metodo: Post

URL: http://fogescaping.esy.es/request.php?service=checkForFogValues

Body: JSON di Richiesta Dati di Nebbia (Listing 4.10)

Header: Accept: application/json | Content-type:application/json

```
{
  "points":[
    {
      "lat":44.838008,
      "lng":11.152922,
      "point_id":"punto1"
    },
    {
      "lat":44.748008,
      "lng":11.152922,
      "point_id":"punto2"
    }
  ]
}
```

Listing 4.10: JSON di richiesta dati ambientali.

Risposta:

Body: JSON di Risposta (Listing 4.11)

Header: Content-type:application/json; charset=utf-8

Una volta processati i punti e associato ad ognuno di essi un valore di visibilità, quello che si ottiene è un JSON così formattato:

```
{
  "message": "",
  "request_time": 1392112251,
  "points": [
    {
      "lat": 44.838008,
      "lng": 11.152922,
      "point_id": "punto1",
      "fog_value": 5
    },
    {
      "lat": 44.748008,
      "lng": 11.152922,
      "point_id": "punto2",
      "fog_value": 0
    }
  ]
}
```

Listing 4.11: JSON di risposta dati ambientali.

4.2 Server

Il Processo di implementazione lato Server, è iniziato con la definizione delle funzionalità che lo stesso doveva implementare. Si è così preferito iniziare dallo sviluppo del Database MySQL, che rappresenta uno dei punti più importanti del sistema, poiché ad esso sono delegate, la maggior parte delle

responsabilità. La struttura del Database è molto semplice, poiché rappresenta solo uno strumento di appoggio *momentaneo*, e in sviluppi futuri dovrà essere ampliato con funzionalità che permettano la gestione della sicurezza, di controllo sull'affidabilità dei dati ecc.. (Cap. 6.3.5). Il vero cuore quindi, non è nella struttura ma nelle funzioni che esso ricopre; infatti, sono implementate alcune stored procedure, che permettono un'analisi veloce dei dati, direttamente nella sua struttura originaria. Le stored procedure, sono pensate proprio in visione di un ampliamento della struttura del Database, poiché rispetto alle query, permettono una modifica della funzionalità lato Database e non lato Server, in cui è necessario concatenare una moltitudine di stringhe, per ottenere il medesimo risultato. Inoltre le stored procedure permettono, a tutti gli effetti, di programmare la base di dati, precompilando metodi che saranno poi richiamati, da un qualsiasi componente che utilizzi i Driver MySQL; nel nostro caso PHP con il componente PDO con Driver MySQL. Il nome della base di dati, per motivi di hosting, è *u572804960_fog*.

4.2.1 Implementazione Stored Procedure

Le stored procedure rappresentano a tutti gli effetti, i servizi al quale il Server risponde. Le stored procedure create per il progetto di Tesi sono:

- registration
- login
- insertFogGeoTag
- checkForFogValue

Registrazione

La procedura *registration* prende in ingresso: la mail dell'Utente, il Nome, il Cognome e una password.

```
1  CREATE PROCEDURE registration(IN mail varchar(255), IN
       firstName varchar(100), IN secondName varchar(100), IN
       pswd varchar(32))
```

Sono dichiarate alcune variabili, che saranno utili al funzionamento della procedura.

```
2  BEGIN
3    DECLARE isPresent INT DEFAULT 0;
4    DECLARE aFirstName varchar(20) DEFAULT LOWER(TRIM(firstName
       ));
5    DECLARE aSecondName varchar(20) DEFAULT LOWER(TRIM(
       secondName));
6    DECLARE eMail varchar(255) DEFAULT LOWER(TRIM(mail));
7    DECLARE idUtente INT DEFAULT -1;
```

Viene fatto un controllo sull'esistenza dell'Utente nella base di dati; se l'Utente esiste, viene inserito il suo ID nella variabile *isPresent*.

```
8      SELECT COUNT(U.idUser) INTO isPresent
9      FROM User AS U
10     WHERE U.Mail = eMail;
```

Se l'Utente è già inserito, la procedura restituirà -1, altrimenti l'ID appena inserito.

```
11     IF isPresent = 0 AND aFirstName != '' AND aSecondName !=
       '' AND pswd != '' AND eMail != '' THEN
12       INSERT INTO  `u572804960_fog`.`User` (`FirstName` ,`
       SecondName` ,`Mail` ,`Password`)
13         VALUES (aFirstName,  aSecondName,  eMail,  pswd);
14       SET idUtente = LAST_INSERT_ID();
15     END IF;
16   SELECT idUtente;
17 END
```

Login

La procedura *login* prende in ingresso la mail dell'Utente e la password.

```
CREATE PROCEDURE `u572804960_fog`.`login`(IN Mail varchar
    (255), IN pswd varchar(32))
```

Se viene trovato qualche record corrispondente, nella variabile aIdUser sarà contenuto l'ID dell'Utente trovato. Sarà restituito -1 se l'Utente non è registrato nella base di dati.

```
BEGIN
  DECLARE aIdUser INT DEFAULT -1;
  SELECT U.idUser INTO aIdUser
  FROM User AS U
  WHERE U.Mail = Mail AND U.Password = pswd;
  IF aIdUser > -1 THEN
    SELECT aIdUser AS idUser;
  ELSE SELECT aIdUser AS idUser;
  END IF;
END
```

Inserimento Dato Ambientale di Nebbia

La procedura *insertFogGeoTag* prende in ingresso: l'ID dell'Utente, Latitudine, Longitudine e il valore ambientale di Nebbia.

```
CREATE PROCEDURE insertFogGeoTag(IN idUser INT, IN latitude
    FLOAT(12,8), IN longitude FLOAT(12,8), IN fogValue INT)
```

La procedura restituisce 1 se il record è stato inserito, 0 se qualche dato in ingresso è nullo.

```
  BEGIN
      IF idUser != '' AND latitude != '' AND longitude != ''
      AND fogValue >= 0 AND fogValue <=5 THEN
      INSERT INTO FogGeoTag(`idUser`,`Date`,`Latitude`,`
   Longitude`,`FogValue`)
```

```
6       VALUES (idUser, NOW(), latitude, longitude, fogValue);
7       SELECT 1 AS inserted;
8     ELSE SELECT 0 AS inserted;
9     END IF
10    END
```

Controllo Dato Ambientale di Nebbia

La procedura *checkForFogValue*, è sicuramente la più importante, perché permette di trovare segnalazioni di nebbia adiacenti a un *Punto di Controllo*. Prende in ingresso Latitudine, Longitudine, un valore di distanza in metri e un massimo di tempo passato. Gli ultimi due parametri sono importantissimi, poiché permettono di impostare i metri di diametro e di massima validità di una segnalazione. Questi valori sono impostati in un file di configurazione PHP, in modo da poter essere modificati con facilità (Cap. 4.2.2).

```
1 CREATE PROCEDURE checkForFogValue(IN latitude FLOAT(12,8),IN
    longitude FLOAT(12,8), IN maxDistanceInMeters INT, IN
    maxElapsedInHours FLOAT)
```

La procedura utilizza una formula per il calcolo della distanza fra due coordinate geografiche, e restituisce il valore massimo di nebbia riscontrato, assieme alla distanza in metri della segnalazione scelta.

```
2   BEGIN
3       SELECT MAX(C.FogValue) AS FogValue, C.Latitude, C.
    Longitude, C.distance_in_meters AS Distance ,C.
    elapsed_time_in_hours AS ElapsedTime
4   FROM (
5   SELECT
6     F.Latitude, F.Longitude, F.FogValue,
7     6371 * 2 * ASIN(SQRT(POWER(SIN(RADIANS(latitude - ABS(F.
    Latitude))), 2)
8     + COS(RADIANS(latitude)) * COS(RADIANS(ABS(F.Latitude)))
9     * POWER(SIN(RADIANS(longitude - F.Longitude)), 2)))
10    *1000 AS distance_in_meters,
```

```
11      (TIME_TO_SEC(TIMEDIFF(NOW(), F.Date))/60)/60 AS
        elapsed_time_in_hours
12    FROM FogGeoTag AS F
13    HAVING distance_in_meters < maxDistanceInMeters AND
        elapsed_time_in_hours < maxElapsedInHours
14    ORDER BY FogValue, distance_in_meters
15    ) AS C;
16      END
```

4.2.2 Implementazione PHP

L'Implementazione PHP, permette la creazione dei servizi web, accedibili tramite il protocollo HTTP.

config.php

È il file di configurazione; in esso sono salvate le informazioni che devono essere presenti, per un corretto funzionamento del Sistema. Fra le variabili configurabili, sono presenti i dati di accesso alla base di dati, come il nome dell'host, il nome del Database, il nome Utente e la password.

```
1   if(!defined("db_host")) define ("db_host", "mysql.hostinger
      .it");
2   if(!defined("db_name")) define ("db_name", "u572804960_fog"
      );
3   if(!defined("db_username")) define ("db_username", "
      u572804960_fog");
4   if(!defined("db_password")) define ("db_password", "
      fogescaping");
```

Sono inoltre molto importanti questi ultimi valori, poiché rappresentano i criteri per la validità spaziotemporale delle segnalazioni di Nebbia. Di default, le segnalazioni hanno un diametro di 1000 metri e 6 ore di validità.

```
5   if(!defined("MAX_DISTANCE_IN_METERS")) define ("
     MAX_DISTANCE_IN_METERS", 1000);
6   if(!defined("MAX_TIME_IN_HOURS")) define ("
     MAX_TIME_IN_HOURS", 6);
```

request.php

Questa è la classe che si occupa di gestire le richieste HTTP, attivando i servizi necessari per l'utilizzo del Sistema. Viene prima instanziato l'oggetto *HttpResponder*, che si occuperà di gestire la creazione delle risposte HTTP.

```
1 require_once('lib/HttpResponder.php');
2 $myHR = new HttpResponder();
```

Per ognuno dei servizi, viene controllato che il metodo HTTP utilizzato sia POST, e che il contenuto del Body abbia tutti i dati necessari. Nel caso in cui il metodo HTTP utilizzato sia diverso da POST, sarà inviata una risposta *405 Method Not Allowed*; nel caso in cui il JSON inviato sia errato, si provvederà a inserire un messaggio *"tutti i campi sono richiesti"* nel JSON di risposta.

Per ognuno dei quattro servizi creati, il codice PHP è molto simile; riporto qui solo il primo servizio implementato: *registration*.

```
3 if($_GET['service'] == 'registration'){
4     $input = file_get_contents('php://input');
5     $data = json_decode( $input, TRUE );
6
7     if($_SERVER['REQUEST_METHOD'] != "POST"){
8         $myHR->methodNotAllowed("Metodo ".$_SERVER['
    REQUEST_METHOD']." non supportato da questa risorsa!");
9     }else{
10        if(($data['email'] != "") AND ($data['name'] != "")
    AND ($data['lastname'] != "") AND ($data['password'] != ""
    )){
```

```
11              $myHR->registration($data['email'],$data['name'],
    $data['lastname'],$data['password']);
12          }else{
13              $myHR->requestDenied("tutti i campi sono
    richiesti");
14          }
15      }
16  }
```

HttpResponder.php

La classe HttpResponder, si occupa di gestire la creazione delle risposte HTTP. Essa importa la classe Controller.php, che si occupa di fare da *"centro di smistamento"* per le richieste al Database. I metodi implementati sono: requestDenied(), methodNotAllowed(), registration(), login(), insertFogGeoTag(), checkForFogValue(). HttpResponder implementa anche un semplice metodo chiamato setHeadersFor200OK(); esso crea un Header adatto alla risposta HTTP 200, che verrà utilizzato nella maggior parte dei casi.

```
1       public static function setHeadersFor200OK(){
2           header('Content-type: application/json; charset=utf-8
    ');
3           if($_SERVER['SERVER_PROTOCOL'] != "HTTP/1.1"){
4               header('Connection: close');
5               header("HTTP/1.0 200 OK");
6           }else{
7               header("HTTP/1.1 200 OK");
8           }
9       }
```

Il metodo *requestDenied* ha il compito di notificare al Client, che una richiesta effettuata, non può essere presa in carico per mancanza di dati o per errori negli stessi. Esso prende in ingresso una stringa, che sarà poi aggiunta nel campo *messaggio* del JSON di risposta (esempio nel Listing 4.1).

```
10      public static function requestDenied($cause){
11          date_default_timezone_set("UTC");
12          self::setHeadersFor200OK();
13          $response = array(
14              "message" => $cause,
15              "request_time" => time()
16          );
17          $json = json_encode($response);
18          echo $json;
19      }
```

Il metodo *methodNotAllowed*, ha il compito di notificare al Client che il metodo HTTP utilizzato per la richiesta, non è permesso. Tutti i servizi attivi sul Server HTTP, rispondono solo al metodo POST, quindi ogni altro metodo come GET o DELETE, ritorneranno una risposta di codice 405. Com'è possibile notare, tra riga 24 e 28, il metodo è compatibile sia con il protocollo HTTP 1.1, che HTTP 1.0.

```
20      public static function methodNotAllowed($cause){
21          date_default_timezone_set("UTC");
22
23          header('Content-type: application/json; charset=utf-8
    ');
24          if($_SERVER['SERVER_PROTOCOL']!="HTTP/1.1"){
25              header('Connection: close');
26              header("HTTP/1.0 405 Method Not Allowed");
27          }else{
28              header("HTTP/1.1 405 Method Not Allowed");
29          }
30
31          $response = array(
32              "message" => $cause,
33              "request_time" => time()
34          );
35          $json = json_encode($response);
36          echo $json;
```

```
37      }
```

Il metodo *registration*, ha il compito di notificare al Client, il responso riguardante la registrazione. Il metodo, come i successivi che elencherò, richiamano la classe Controller.php, per richiedere la registrazione dell'Utente con i dati a disposizione. Prende in ingresso email, nome, cognome e password dell'Utente, e restituisce l'ID dell'Utente appena inserito nel Database. L'ID di risposta, sarà invece uguale a -1 quando l'Utente è già registrato presso il Database.

```
38  public function registration($email, $name, $lastname,
        $password){
39          date_default_timezone_set("UTC");
40          $userID = Controller::registration($email, $name,
        $lastname, $password);
41
42          $response = array(
43              "message" => "",
44              "request_time" => time(),
45              "user_id" => $userID
46          );
47
48          $json = json_encode($response);
49          self::setHeadersFor200OK();
50          echo $json;
51      }
```

Il metodo *login* ha il compito di notificare al Client, il responso riguardante il "Login" dell'Utente; in realtà, considerata la natura del Sistema, non è un vero e proprio Login, poiché non viene memorizzata nessuna sessione riguardante il Client. Il metodo prende in ingresso email e password dell'Utente e restituisce l'ID di registrazione presso il Database, che sarà poi memorizzato all'interno del Client e utilizzato nelle richieste di *insertFogGeoTag* e *checkForFogValues*.

```
52 public function login($email, $password){
53         date_default_timezone_set("UTC");
54         $userID = Controller::login($email,$password);
55 
56         $response = array(
57             "message" => "",
58             "request_time" => time(),
59             "user_id" => $userID
60         );
61 
62         $json = json_encode($response);
63         self::setHeadersFor200OK();
64         echo $json;
65     }
```

Il metodo *insertFogGeoTag*, ha il compito di notificare al Client, il responso riguardante l'inserimento di un dato ambientale di Nebbia. Prende in ingresso l'ID dell'Utente restituito dal processo di Login, Latitudine, Longitudine e il valore di Nebbia dichiarato dalla segnalazione. Restituisce *true*, se l'operazione è andata a buon fine, *false* altrimenti.

```
66     public function insertFogGeoTag($userID,$lat,$lng,
   $fogValue){
67         date_default_timezone_set("UTC");
68         $inserted = Controller::insertFogGeoTag($userID,$lat,
   $lng,$fogValue);
69 
70         ($inserted == 1) ? $result = true : $result = false;
71 
72         $response = array(
73             "message" => "",
74             "request_time" => time(),
75             "result" => $result
76         );
77 
78         $json = json_encode($response);
79         self::setHeadersFor200OK();
```

```
80          echo $json;
81      }
```

Il metodo *checkForFogValue*, ha il compito di generare una risposta, dove per ogni *Punto di Controllo* appartenente alla griglia, viene associato un valore di visibilità richiesto al Database. Ogni *Punto di Controllo* deve quindi mantenere le sue proprietà principali, di latitudine, longitudine e id; questo per la corretta ricostruzione della griglia lato Client.

```
82      public function checkForFogValue($points){
83          date_default_timezone_set('UTC');
84
85          $fogPoints = array();
86          foreach($points as $point){
87              $pointData = Controller::checkForFogValue($point[
    'lat'],$point['lng']);
88              $newPoint = array(
89                  "lat" => $point['lat'],
90                  "lng" => $point['lng'],
91                  "point_id" => $point['point_id'],
92                  "fog_value" => (int)$pointData[0]['FogValue']
93              );
94
95              array_push($fogPoints,$newPoint);
96          }
97
98          $response = array(
99              "message" => "",
100             "request_time" => time(),
101             "points" => $fogPoints
102         );
103         $json = json_encode($response);
104         self::setHeadersFor200OK();
105         echo $json;
106     }
```

Controller.php

La classe Controller.php, si occupa di smistare ogni richiesta diretta al Database. Tutti i metodi utilizzano la classe DBHandler.php, che si occupa di generare le richieste al Database per le stored procedure. I metodi implementati sono: registration(), login(), insertFogGeoTag(), checkForFogValue(). Ognuno di questi metodi richiama la funzione relativa dall'oggetto *singleton* DBHandler.

Metodo di Registrazione

```
1     public static function registration($email, $name,
    $lastname, $password)
2     {
3         $dbHandler = DBHandler::singleton();
4         $result = $dbHandler->callProcedure(array($email,
    $name,$lastname,$password),"registration");
5
6         return $result[0]['idUtente'];
7     }
```

Metodo di Login.

```
8  public static function login($email, $password)
9      {
10         $dbh = DBHandler::singleton();
11         $result = $dbh->callProcedure(array($email, $password
    ),"login");
12
13         return $result[0]['idUser'];
14     }
```

Metodo d'Inserimento dato ambientale di Nebbia.

```
15    public static function insertFogGeoTag($userID, $lat,
    $lng, $fogValue)
16    {
```

```
17          $dbh = DBHandler::singleton();
18          $result = $dbh->callProcedure(array($userID, $lat,
    $lng, $fogValue),"insertFogGeoTag");
19
20          return $result[0]['inserted'];
21      }
```

Metodo di richiesta dato ambientale di Nebbia. Esso utilizza i valori MAX_DISTANCE_IN_METERS e MAX_TIME_IN_HOURS, entrambi provenienti da Config.php.

```
22      public static function checkForFogValue($lat,$lng){
23          $dbh = DBHandler::singleton();
24          $result = $dbh->callProcedure(array($lat,$lng,
    MAX_DISTANCE_IN_METERS,MAX_TIME_IN_HOURS),"
    checkForFogValue");
25
26          return $result;
27      }
```

DBHandler.php

La classe DBHandler.php rappresenta la vera e propria connessione con il Database. La classe è implementata con il pattern *Singleton*, in modo che per ogni Utente, ci sia una sola istanza di questo oggetto.

```
1  class DBHandler {
2      private static $instance;
3      private function __construct(){}
4      public static function singleton()
5      {
6          if (!isset(self::$instance)) {
7              //instantiating class if istance doesn't exists
8              //$className = __CLASS__;
9              self::$instance = new DBHandler();
10         }
11         //returning class instance
```

```
12          return self::$instance;
13      }
14      private function __clone(){}
```

Il metodo *callProcedure*, prende in ingresso una array di valori e il nome della procedura che dovrà essere richiamata. Qualora al posto dell'array sia passata una stringa, essa sarà interpretata come un array di un solo elemento.

```
15      public function callProcedure($values, $procedureName) {
16          if(sizeof($values)>0 and (is_array($values) or
    is_string($values))){
17              try{
```

Viene poi richiamato l'oggetto PDO, nel quale sono inseriti i dati provenienti da Config.php essenziali per la connessione dal Database.

```
18      //initializing PDO and establishing DB connection
19                  $db = new PDO("mysql:host=" .db_host. ";
    dbname=".db_name.";charset=utf8" ,db_username,db_password,
    array(
20                      PDO::MYSQL_ATTR_INIT_COMMAND => "SET
    NAMES utf8"));
21                  $query = "CALL ".db_name."."$procedureName."
    (";
```

In seguito, i valori contenuti nell'array, sono concatenati per comporre la richiesta per una determinata stored procedure.

```
22      if(is_array($values)){
23    //if i = 0 => first iteration
24      $i = 0;
25      foreach ($values as $value) {
26        if($i == 0){
27        $query = $query . "?";
28      }else{
29      $query = $query . ",?";
```

```
30      }
31    $i++;
32    }
33    $query = $query . ");";
34  }
```

A questo punto è necessario preparare il Database per l'accoglimento della stored procedure.

```
35    $stmt = $db->prepare($query);
36    if(is_array($values)){
37    //using the "j" index to bind the parameter
38    $j = 0;
39    foreach($values as $value){
40      //DO NOT USE $value instead of $values[$j]. PDO doesn't
      work this way even if $value should be correct
41      $stmt->bindParam($j+1, $values[$j], PDO::PARAM_STR);
42      $j++;
43    }
44  }
```

Viene eseguita la stored procedure, e viene salvato il contenuto della risposta all'interno della variabile *$result*.

```
45    //executing procedure
46    $rs = $stmt->execute();
47    if(!$rs){$error_code = $stmt->errorCode();
48      if($error_code == '42000'){
49                          //return "<p class=\"error\">query
      syntax error: " .$error_code. "</p>";
50                          return("Query syntax error, error
      code: 42000:\n"); exit();
51                      }
52    }
53    $result = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Infine viene interrotta la connessione al Database, e viene restituita la variabile *$result*.

```
54    $db = null;
55    return $result;
56    }catch(PDOException $ex){
57                  //die('db error');
58      echo("ERROR:\n".$ex->getMessage()); exit();
59    }
60  }}
```

4.3 Client

L'implementazione della parte Client, è un applicazione pensata per funzionare sulle piattaforme mobili Apple, sia iPhone che iPad. Si è preferito sviluppare direttamente sui dispositivi invece che su un simulatore, questo per garantire la qualità del prodotto.

La piattaforma *XCode* di Apple, permette di sviluppare sia la parte grafica che il codice, inoltre gli strumenti messi a disposizione, aiutano lo sviluppatore a trovare facilmente gli errori, *inevitabili* durante la scrittura.

L'applicazione *Fog Escaping* è sviluppata secondo il pattern MVC (Model-View-Controller), che aiuta a tenere separati i componenti grafici dalla logica, inoltre che a garantire l'alta riusabilità dei componenti.

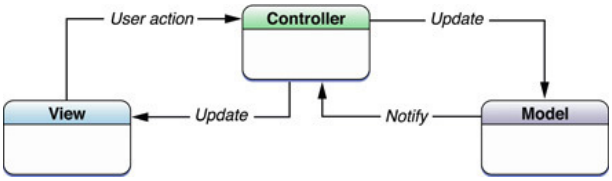


Figura 4.5: Pattern Model-View-Controller.

L'implementazione è iniziata con lo studio dei sistemi di direzionamento di Apple; l'obiettivo era capire quale livello di precisione si sarebbe potuto raggiungere. Purtroppo non è possibile accedere ai dati sui nodi stradali, quindi era necessario trovare un metodo alternativo (Cap. 3.2).

Fog Escaping ha due interfacce grafiche distinte, una per iPhone e una per iPad; sono quindi differenti i metodi con il quale si devono gestire gli spostamenti tra una View e un'altra. Per iPhone si è preferito utilizzare una navigazione modale, che permette di far comparire le opzioni dal basso, mantenendo quindi una gerarchia dove dalla View principale, ci si può spostare verso ognuna delle altre (Figura 4.6).

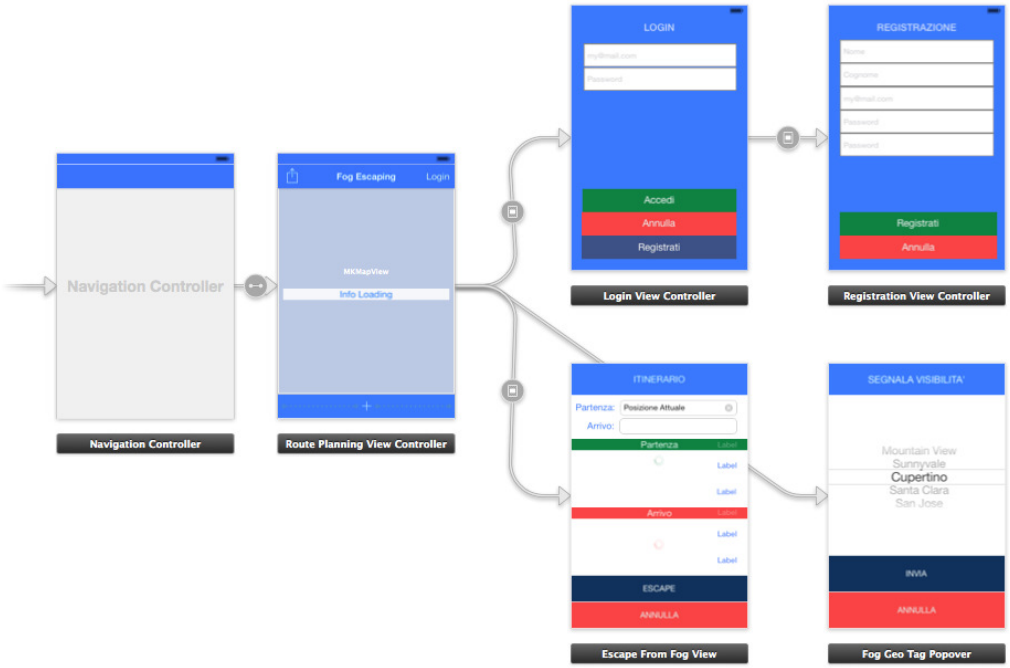


Figura 4.6: Storyboard iPhone.

Per iPad la metodologia è decisamente diversa, poiché per le opzioni basilari, come la segnalazione del dato ambientale di Nebbia e la ricerca di un percorso alternativo, si è preferito utilizzare i Popover; essi permettono la visualizzazione del contenuto, senza mai uscire dalla View principale (Figura 4.7).

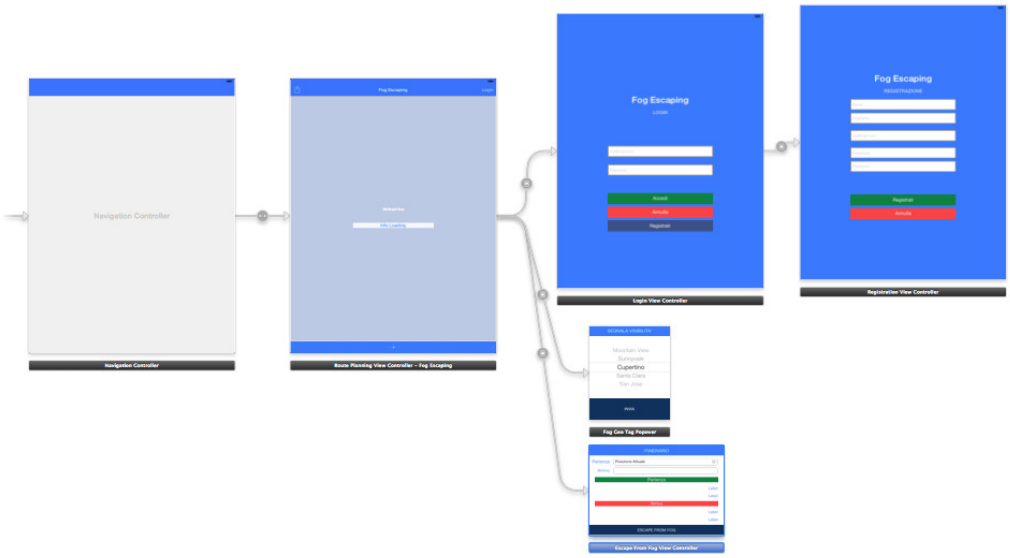


Figura 4.7: Storyboard iPad.

4.3.1 Notification

Data la natura asincrona delle richieste, è stato necessario gestire alcuni passaggi, tramite le *Notification* di Apple. Esse permettono di notificare a uno o più Observer, il cambiamento di uno stato; nell'applicazione sono inserite per gestire i JSON di ritorno dal Server e per effettuare operazioni. Ad esempio quando il Client ha inserito i dati di Login, prima di variare uno stato grafico, è necessario attendere il JSON di risposta del Server e successivamente decidere se permettere il Login, oppure far notificare all'Utente che i dati non sono corretti.

4.3.2 RoutePlanningViewController

La classe RoutePlanningViewController ha lo scopo di gestire l'interfaccia grafica principale del sistema. Essa comprende: la mappa in primo piano,

in alto i pulsanti di Login/Logout e ricerca percorso alternativo, in basso il pulsante di segnalazione dello stato ambientale di Nebbia.

Interfaccia

L'interfaccia definisce alcune *Property*, necessarie alla gestione dei componenti grafici e una variabile ZOOM_QUARTIER che servirà per il corretto zoom iniziale della mappa.

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>

#define ZOOM_QUARTIER 10000

@interface RoutePlanningViewController : UIViewController
@property (strong, nonatomic) IBOutlet MKMapView *mapView;
@property (nonatomic, strong) CLLocationManager *locationManager;
@property (strong, nonatomic) IBOutlet UIBarButtonItem *loginLogoutButton;
@property (strong, nonatomic) IBOutlet UIBarButtonItem *fogGeoTagButton;
@property (strong, nonatomic) IBOutlet UIBarButtonItem *escapeButton;
@property (strong, nonatomic) IBOutlet UIActivityIndicatorView *
    mapActivityIndicator;
@property (strong, nonatomic) IBOutlet UILabel *infoLoadingLabel;
@end
```

viewDidLoad:

Quando la View sta per essere caricata, il sistema richiama il metodo viewDidLoad, nel quale sono fatte le inizializzazioni delle variabili d'istanza.
In seguito è inizializzato il *CLLocationManager*; esso si occupa di richiedere la posizione dell'Utente al dispositivo.

```
- (void)viewDidLoad
{
    [super viewDidLoad];
     if(!_locationManager) _locationManager = [[CLLocationManager alloc]init
    ];
    _locationManager.delegate = self;
    _locationManager.desiredAccuracy = kCLLocationAccuracyKilometer;
    [self showUserLocation];
    readyToRemoveInversion = NO;
    _infoLoadingLabel.text = @"";
```

```
10      _mapView.delegate = self;
11      logout = [self isAlreadyLogged];
12      if(logout) {
13          _loginLogoutButton.title = @"Logout";
14      }
15      [self initComponentGraphicsModification];
16  }
```

In questa fase la classe RoutePlanningViewController, si mette in ascolto su alcune *Notification*, che permetteranno di riflettere coerentemente lo stato della richiesta effettuata, con lo stato grafico necessario.

```
17      [[NSNotificationCenter defaultCenter] addObserver:self selector:
        @selector(sendFogValue:) name:@"fogValue" object:nil];
18      [[NSNotificationCenter defaultCenter] addObserver:self selector:
        @selector(escapeFromFog:) name:@"escape" object:nil];
19      [[NSNotificationCenter defaultCenter] addObserver:self selector:
        @selector(receiveSolutionFromGrid:) name:@"solution-ready" object:nil];
20      [[NSNotificationCenter defaultCenter] addObserver:self selector:
        @selector(receiveGridWithFog:) name:@"grid-ready" object:nil];
21      [[NSNotificationCenter defaultCenter] addObserver:self selector:
        @selector(connectionError:) name:@"connection-error" object:nil];
22      [[NSNotificationCenter defaultCenter] addObserver:self selector:
        @selector(routeReady:) name:@"route-ready" object:nil];
```

sendFogValue:

Il metodo *sendFogValue* viene richiamato al ricevimento della *Notification* relativa all'inserimento del dato ambientale di Nebbia. Utilizza la classe FogRequest (Cap. 4.3.10) per la gestione della richiesta HTTP, riguardante il servizio *insertFogValue* (Cap. 4.1.3).

```
23  - (void) sendFogValue:(NSNotification *) notification {
24      NSNumber *fogValue = [notification object];
25      FogRequest *req = [[FogRequest alloc]init];
26      [req insertFogDataWithCoordinate:_locationManager.location.coordinate
        andFogValue:[fogValue intValue]];
27      [_fogGeoTagPopover dismissPopoverAnimated:YES];
28  }
```

escapeFromFog:

Il metodo *escapeFromFog*, viene richiamato al ricevimento della *Notification*, relativa al calcolo di un percorso alternativo. Come specificato nelle sezioni precedenti, prima di generare la richiesta HTTP per i dati di nebbia, è necessario richiedere il percorso ottimale (Cap. 3.2.1), e la Griglia dei *Punti di Controllo* (Cap. 3.2.2). Viene quindi richiamato il metodo di classe *loadStandardRouteWithStartLocation:andArriveLocation*, che si occupa di generare la sequenza degli eventi che porteranno alla presentazione della richiesta HTTP. *checkForFogValues*.

```
29 - (void) escapeFromFog:(NSNotification*) notification {
30     _infoLoadingLabel.text = @"Processamento Dati";
31     [_infoLoadingLabel setHidden:NO];
32     [self initComponentAndVariabiles];
33     [self resetAllMapContents];
34     [self startMapLoading];
35     NSArray *locations = [notification object];
36     CLLocation *start = locations[0];
37     CLLocation *arrive = locations[1];
38     [self loadStandardRouteWithStartLocation:start andArriveLocation:arrive
       ];
39 }
```

loadStandardRouteWithLocation:andArriveLocation:

Il metodo *loadStandardRouteWithLocation:andArriveLocation*, richiamerà il metodo *findDirectionsFrom:to*, che ha il compito di richiedere ai Server Apple il percorso ottimale tra un punto di partenza e un punto di arrivo. Il percorso calcolato sarà poi passato al metodo, *useDirections:*.

```
40 MKDirectionsRequest *request = [[MKDirectionsRequest alloc] init];
41     [request setSource:source];
42     [request setDestination:destination];
43     request.requestsAlternateRoutes = NO;
44 
45     MKDirections *directions = [[MKDirections alloc] initWithRequest:request
       ];
46     //make network indicator visible
47     [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:
       YES];
```

```
48      __weak RoutePlanningViewController *weakSelf = self;
49      [directions calculateDirectionsWithCompletionHandler:
50       ^(MKDirectionsResponse *response, NSError *error) {
51           if (error) {
52               NSLog(@"Error is %@",error);
53               NSString *errorString = @"Richiesta di Direzione non
    disponibile al momento";
54               if(error.code == 5){
55                   errorString = @"Direzione non disponibile al momento.";
56               }
57               UIAlertView *alert = [[UIAlertView alloc]initWithTitle:@"Fog
    Escaping" message:errorString delegate:nil cancelButtonTitle:@"OK"
    otherButtonTitles: nil];
58               [alert show];
59               //activate the program for other actions
60               [self stopMapLoading];
61           } else {
62               _infoLoadingLabel.text = @"Processamento Dati di Direzione";
63               [weakSelf useDirections:response];
64           }
65       }];
66  }
```

useDirections:

Il metodo *useDirections*, si occupa principalmente di salvare il percorso ottimale generato dai servizi Apple, e infine di creare l'oggetto *Grid* che rappresenta la griglia dei *Punti di Controllo* . Sarà direttamente l'oggetto *Grid* ad occuparsi della richiesta HTTP, e invierà una *Notification* quando tutti i dati di visibilità, saranno ritornati dal Server.

```
67  - (void)useDirections:(MKDirectionsResponse *)response
68  {
69    //Lazy instantiation for routes array of points
70    if(!_routes){
71      _routes = [[NSMutableArray alloc]initWithCapacity:[response.routes count
      ]];
72    } else {
73     [_routes removeAllObjects]; //remove all the precedent routes from the
      array
74    }
75
76    MKRoute* route = response.routes[0];
77    //estract the polyline from route
```

```
78    MKPolyline *line = route.polyline;
79    //extract all the points from the polyline
80    NSMutableArray *array = [[NSMutableArray alloc]initWithCapacity:[line
        pointCount]];
81    for(int i = 0; i < line.pointCount; i++) {
82      CLLocationCoordinate2D coord = MKCoordinateForMapPoint(line.points[i]);
83      CLLocation *location = [[CLLocation alloc]initWithLatitude:coord.
        latitude longitude:coord.longitude];
84      //print test for route points
85      RoutePoint *point = [[RoutePoint alloc]init];
86      point.location = location;
87      [array addObject:point];
88    }
89    [_routes addObject:array];
90
91    //insert intermediate point to A-B segment too long
92    [self correctRoutesForLongStraightDistance];
93    _infoLoadingLabel.text = @"Richiesta Dati Ambientali";
94    _grid = [[Grid alloc]initGridWithRoute:_routes[0]];
95  }
```

receiveGridWithFog:

Quando la Griglia è stata completamente costruita, viene richiamato il metodo *receiveGridWithFog*, che si occupa di cambiare la label delle info di caricamento e di inviare la Griglia al metodo che troverà il *"cammino di nebbia minima"*.

```
96  - (void) receiveGridWithFog:(NSNotification*) notification {
97      dispatch_async(dispatch_get_main_queue(), ^{
98          [[UIApplication sharedApplication]
        setNetworkActivityIndicatorVisible:NO];
99          _infoLoadingLabel.text = @"Processamento Dati Ambientali";
100     });
101     [_grid gridReady:[notification object]];
102 }
```

receiveSolutionFromGrid:

Quando la soluzione sarà stata calcolata, essa dovrà ripassare per il *RoutePlanningViewController* per essere corretta e poi visualizzata sulla mappa.

Un passaggio molto importante, è la correzione della soluzione tramite il metodo *correctSolution*, che si occupa di eliminare i punti interni alla retta come specificato nel Capitolo 3.5. Successivamente sono numerati i punti di soluzione, in modo che possano essere calcolati correttamente i percorsi fra un punto e l'altro.

```
103 - (void) receiveSolutionFromGrid:(NSNotification*) notification {
104     NSArray* solutionAndGrid = [notification object];
105     NSArray* solution = [self correctSolution: solutionAndGrid[0]];
106     _solutionPoints = solution;
107     NSArray*gridPoints = solutionAndGrid[1];
108     //numering the route points
109     for (int i = 0; i<[_routes[0]count]; i++){
110         RoutePoint *p = _routes[0][i];
111         p.number = [NSNumber numberWithInt:i];
112     }
```

Sono caricate sulla mappa, le informazioni riguardanti lo stato delle Nebbie, in modo da avere un paragone visivo.

```
113 _routeSolutionForAlternative = [solution copy];
114     for(int i = 0; i<[gridPoints count] ; i++){
115         ControlPoint *point = gridPoints[i];
116         CLLocation *loc = point.location;
117         if ([point.levelOfFog intValue]>=1){
118             fogValueToShow = [point.levelOfFog intValue];
119             MKPointAnnotation *annotation =[self
    createMapAnnotationForCoordinate:loc.coordinate andTitle:text
    andFogValue:[NSNumber numberWithInt:fogValueToShow]];
120             [_mapView addAnnotation:annotation];
121         }
122     }
```

Infine sono visualizzate sulla Mappa, le direzioni fra un *Punto di Controllo* e un altro, in modo da creare un percorso alternativo completo.

```
123     allRouteToCount = [solution count];
124     if(!_alternativeRoute) _alternativeRoute = [[NSMutableArray alloc]init];
125     [_alternativeRoute removeAllObjects];
126
127     for (int i = 0; i<[solution count]; i++) {
128         ControlPoint *p1 = solution[i];
129         if ([solution count]>i+1){
```

```
130             ControlPoint *p2 = solution[i+1];
131             //itering until destination point
132             [self findDirectionsFromStartPoint:p1.location.coordinate
    andArrive:p2.location.coordinate from:i to:i+1];
133         }
134     }
135 }
```

4.3.3 LoginViewController

LoginViewController, è la classe imputata alla gestione grafica di Login dell'Utente. Nell'interfaccia grafica, è presente il pulsante Registrazione; se premuto, aziona un *modal segue*, che caricherà la View di Registrazione (Cap. 4.3.4). L'interfaccia contiene le *Property* necessarie alla gestione dei componenti grafici.

```
136 #import <UIKit/UIKit.h>
137
138 @interface LoginViewController : UIViewController
139 @property (strong, nonatomic) IBOutlet UITextField *emailTextField;
140 @property (strong, nonatomic) IBOutlet UITextField *passwordTextField;
141 @property (strong, nonatomic) IBOutlet UIActivityIndicatorView *
    activityIndicator;
142 @property (strong, nonatomic) IBOutlet UILabel *infoLogin;
143 @end
```

viewDidLoad

In questa fase si inizializzano i componenti grafici e si imposta la classe come Observer sulle *Notification* di Login: loginAccepted e loginFailed.

```
1 - (void)viewDidLoad
2 {
3     [super viewDidLoad];
4     _emailTextField.delegate = self;
5     _passwordTextField.delegate = self;
6     [[NSNotificationCenter defaultCenter] addObserver:self selector:
      @selector(loginSuccess:) name:loginAccepted object:nil];
7     [[NSNotificationCenter defaultCenter] addObserver:self selector:
      @selector(loginError:) name:loginFailed object:nil];
8 }
```

requestAccess:

Sicuramente è uno dei metodi più importanti della classe; esso si occupa di controllare i dati inseriti dall'Utente, richiamare il metodo di Login, oppure notificare eventuali errori d'inserimento.

```
9  - (IBAction)requestAccess:(id)sender {
10     [_activityIndicator startAnimating];
11     if(![_emailTextField.text isEqualToString:@""] && ![_passwordTextField.
       text isEqualToString:@""]) {
12         NSString *email = _emailTextField.text;
13         NSString *password = _passwordTextField.text;
14         FogRequest *req = [[FogRequest alloc]init];
15         [req loginWithEmail:email andPassword:password];
16         [self hideKeyboard];
17     } else {
18         NSLog(@"Errore login: Dati mancanti");
19         _infoLogin.text = @"Username o Password Mancanti";
20         [_activityIndicator stopAnimating];
21     }
22 }
```

cancel:

Il metodo *cancel:* ha il semplice compito di chiudere la View e ritornare alla precedente. Questo metodo è necessario solo per la versione iPhone.

```
23 - (IBAction)cancel:(id)sender {
24     [self dismissViewControllerAnimated:YES completion:nil];
25 }
```

loginSuccess:

Quando il Login viene effettuato con successo, viene richiamato il seguente metodo dall'Observer. Esso si occupa di gestire i componenti grafici richiamandoli direttamente dal Thread principale; in questo modo è possibile modificarli all'istante. Successivamente si ritorna alla View precedente, notificando l'Utente con un Alert adeguato (Figura 6.15a).

```
26 - (void) loginSuccess:(NSNotification*) notification {
27     //essential for istant message on main queue
28     dispatch_async(dispatch_get_main_queue(), ^{
29         _infoLogin.text = @"Login Effettuato.";
30         [_activityIndicator stopAnimating];
31         UIAlertView * alert = [[UIAlertView alloc]initWithTitle:@"Fog
       Escaping" message:@"Login Effettuato." delegate:nil cancelButtonTitle:@"
       OK" otherButtonTitles: nil];
32         [alert show];
33     });
34     [self dismissViewControllerAnimated:YES completion:nil];
35 }
```

loginError:

Quando il Server risponde con un JSON di Login Errato, è necessario notificare all'utente che i dati inseriti non sono corretti. Viene quindi inserito un messaggio di testo direttamente nella View, in questo modo l'Utente può decidere di inserire nuovi dati di Login (Figura 6.15a).

```
36 - (void) loginError:(NSNotification*) notification {
37     //essential for istant message on main queue
38     dispatch_async(dispatch_get_main_queue(), ^{
39         _infoLogin.text = @"Username o Password Errati";
40         [_activityIndicator stopAnimating];
41     });
42 }
```

4.3.4 RegistrationViewController

La classe *RegistrationViewController*, si occupa di gestire la Registrazione dell'Utente. L'interfaccia dichiara i componenti grafici e le relative *Property*

```
1 #import <UIKit/UIKit.h>
2
3 @interface RegistrationViewController : UIViewController
4 @property (strong, nonatomic) IBOutlet UITextField *name;
5 @property (strong, nonatomic) IBOutlet UITextField *lastName;
6 @property (strong, nonatomic) IBOutlet UITextField *mail;
```

```
@property (strong, nonatomic) IBOutlet UITextField *password1;
@property (strong, nonatomic) IBOutlet UITextField *password2;
@property (strong, nonatomic) IBOutlet UIActivityIndicatorView *activity;
@property (strong, nonatomic) IBOutlet UILabel *registrationInfo;

@end
```

viewDidLoad:

Il metodo inizializza i componenti grafici e aggiunge la classe come Observer sulle seguenti *Notification*: registrationAccepted e registrationFailed.

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    _mail.delegate = self;
    _name.delegate = self;
    _lastName.delegate = self;
    _password1.delegate = self;
    _password2.delegate = self;
    [[NSNotificationCenter defaultCenter] addObserver:self selector:
    @selector(registrationSuccess:) name:registrationAccepted object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:
    @selector(registrationError:) name:registrationFailed object:nil];
}
```

registration:

Il metodo è collegato alla pressione del pulsante Registrazione; esso si occupa di controllare tutti i valori in ingresso e in seguito di notificare l'avvenuta registrazione, oppure di notificare gli errori d'inserimento dati (Fig. 6.13a-6.13a).

```
- (IBAction)registration:(id)sender {
    BOOL vName = ![_name.text isEqualToString:@""];
    BOOL vLastName = ![_lastName.text isEqualToString:@""];
    BOOL vMail = ![_mail.text isEqualToString:@""];
    BOOL vPassword1 = ![_password1.text isEqualToString:@""];
    BOOL vPassword2 = ![_password2.text isEqualToString:@""];
    BOOL vEqualPassword = ([_password1.text isEqualToString:_password2.text
    ]) && (vPassword1);
    NSLog(@"Registrazione");
```

```
20      [_activity startAnimating];
21
22      if(vName && vLastName && vMail && vPassword1 &&vEqualPassword) {
23          NSString *name = _name.text;
24          NSString *lastName = _lastName.text;
25          NSString *email = _mail.text;
26          NSString *password = _password1.text;
27          FogRequest *req = [[FogRequest alloc]init];
28          [req registrationWithEmail:email name:name lastName:lastName
    andPassword:password];
29          [self hideKeyboard];
30      } else {
31          UIColor *lightYellow = [UIColor colorWithRed:1 green:0.925 blue
    :0.478 alpha:1];
32          NSLog(@"Errore login: Dati mancanti");
33          NSMutableString *advice = [[NSMutableString alloc]init];
34          [advice appendString:@"Inserire: "];
35          if (!vName) {
36              [_name setBackgroundColor: lightYellow];
37              [advice appendString:@"Nome"];
38          } else {
39              [_name setBackgroundColor: [UIColor whiteColor]];
40          }
41          if (!vLastName) {
42              [_lastName setBackgroundColor: lightYellow];
43              [advice appendString:@" Cognome"];
44          } else {
45              [_lastName setBackgroundColor:[UIColor whiteColor]];
46          }
47          if (!vMail) {
48              [_mail setBackgroundColor: lightYellow];
49              [advice appendString:@" Mail"];
50          } else {
51              [_mail setBackgroundColor:[UIColor whiteColor]];
52          }
53          if (!vPassword1) {
54              [_password1 setBackgroundColor: lightYellow];
55              [advice appendString:@" Password"];
56          } else {
57              [_password1 setBackgroundColor:[UIColor whiteColor]];
58          }
59          if (!vPassword2) {
60              [_password2 setBackgroundColor: lightYellow];
61          } else {
62              [_password2 setBackgroundColor:[UIColor whiteColor]];
63          }
64          [advice appendString:@".\n"];
```

```
65          if (!vEqualPassword) {
66              [_password1 setBackgroundColor: lightYellow];
67              [_password2 setBackgroundColor: lightYellow];
68              [advice setString: @" Password Diverse."];
69          } else {
70              [_password1 setBackgroundColor:[UIColor whiteColor]];
71              [_password2 setBackgroundColor:[UIColor whiteColor]];
72          }
73          _registrationInfo.text = advice;
74          //_infoRegistration.text = @"Username o Password Mancanti";
75          [_activity stopAnimating];
76      }
77  }
```

registrationSuccess:

Il metodo viene richiamato dall'Observer nel momento in cui viene notificata l'avvenuta Registrazione dell'Utente. Esso si occupa di chiudere la View e notificare l'Utente con un Alert adeguato (Figura 6.12b).

```
78  - (void) registrationSuccess : (NSNotification*) notification {
79      //essential for istant message on main queue
80      dispatch_async(dispatch_get_main_queue(), ^{
81          _registrationInfo.text = @"Registrazione Effettuata.";
82          [_activity stopAnimating];
83          UIAlertView * alert = [[UIAlertView alloc]initWithTitle:@"Fog
        Escaping" message:@"Registrazione Effettuata." delegate:nil
        cancelButtonTitle:@"OK" otherButtonTitles: nil];
84          [alert show];
85      });
86      [self dismissViewControllerAnimated:YES completion:nil];
87  }
```

4.3.5 registrationError:

Il metodo viene richiamato dall'Observer, nel momento in cui viene notificato un errore di Registrazione. Esso si occupa di far visualizzare direttamente nella View, un messaggio di errore Registrazione.

```
88  - (void) registrationError : (NSNotification*) notification {
89      //essential for istant message on main queue
90      dispatch_async(dispatch_get_main_queue(), ^{
```

```
91          NSLog(@"Errore registration");
92          _registrationInfo.text = @"Errore di Registrazione";
93          [_activity stopAnimating];
94      });
95  }
```

4.3.6 FogGeoTagViewController

FogGeoTagViewController è la classe imputata alla gestione grafica dell'inserimento del dato ambientale di Nebbia. Nell'interfaccia grafica è presente il pulsante Invia; se premuto, richiamerà il metodo per l'inserimento della Segnalazione. L'interfaccia contiene la *Property* necessaria alla gestione del Picker.

```
1  #import <UIKit/UIKit.h>
2
3  @interface FogGeoTagViewController : UIViewController
4  @property (strong, nonatomic) IBOutlet UIPickerView *visibilityPicker;
5
6  @end
```

viewDidLoad:

viewDidLoad si assicura, che nel momento del caricamento della View, vengano inseriti tutti i valori di visibilità nel Picker.

```
1  - (void)viewDidLoad
2  {
3      [super viewDidLoad];
4      _visibilityPicker.dataSource = self;
5      _visibilityPicker.delegate = self;
6      visibilityName = @[@"Limpido", @"Caligine >10Km",
7      @"1Km< Foschia <10Km",@"Nebbia Spessa (max 200mt)",
8      @"30mt< Nebbia Fitta <50mt",@"0mt< Nebbia Densa <30mt"];
9  }
```

sendFogValue:

sendFogValue, si occupa di richiamare il metodo per l'inserimento del dato ambientale di Nebbia. Nel caso che il device sia iPhone, viene chiusa la View e si ritorna alla precedente.

```
- (IBAction)sendFogValue:(id)sender {
    NSNotification *fogValue = [NSNotification notificationWithName:@"
    fogValue" object:[NSNumber numberWithInt:selected]];
    [[NSNotificationQueue defaultQueue] enqueueNotification:fogValue
    postingStyle:NSPostNow];
    //If iPhone, make dismiss for modal segue
    if ([[UIDevice currentDevice].model rangeOfString:@"iPhone"].location !=
     NSNotFound) {
        //Device is iPhone
        [self dismissViewControllerAnimated:YES completion:nil];
    }
}
```

4.3.7 EscapeFromFogViewController

EscapeFromFogViewController, è la classe che si occupa di gestire la grafica per la richiesta di un percorso alternativo. L'interfaccia dichiara le *Property* necessarie per la gestione dei componenti grafici e un metodo pubblico che verrà utilizzato per fare il *Geocoding* della posizione attuale dell'Utente, nel momento del caricamento della View.

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>

@interface EscapeFromFogViewController : UIViewController
@property (strong, nonatomic) IBOutlet UITextField *startTextField;
@property (strong, nonatomic) IBOutlet UITextField *arriveTextField;
@property (strong, nonatomic) IBOutlet UILabel *startLabel1;
@property (strong, nonatomic) IBOutlet UILabel *startLabel2;
@property (strong, nonatomic) IBOutlet UILabel *startLabel3;
@property (strong, nonatomic) IBOutlet UILabel *arriveLabel1;
@property (strong, nonatomic) IBOutlet UILabel *arriveLabel2;
@property (strong, nonatomic) IBOutlet UILabel *arriveLabel3;
- (void) setUserPositionPlaceMark:(CLLocation*) location;
@property (strong, nonatomic) IBOutlet UIActivityIndicatorView *
    startActivityIndicator;
```

```
15 @property (strong, nonatomic) IBOutlet UIActivityIndicatorView *
       arriveActivityIndicator;
16 @end
```

Variabili d'istanza

Sono necessarie alcune variabili d'istanza, che sono definite nell'implementazione dell'interfaccia.

```
1  #import "EscapeFromFogViewController.h"
2
3  @interface EscapeFromFogViewController () <UITextFieldDelegate>
4  @property (nonatomic, strong) NSMutableArray *placeResults;
5  @property (nonatomic, strong) CLPlacemark *userPlacemark;
6  @property (nonatomic, strong) CLLocation *startLocation;
7  @property (nonatomic, strong) CLLocation *arriveLocation;
8  @end
```

viewDidLoad:

Al caricamento della View, sono impostate le label che saranno utilizzate per visualizzare le informazioni, sul punto di partenza e di arrivo. Viene inoltre impostata la classe come *Delegate* per le TextField, dove saranno inseriti i punti di partenza e arrivo.

```
9  - (void)viewDidLoad
10 {
11     [super viewDidLoad];
12     _startTextField.delegate = self;
13     _arriveTextField.delegate = self;
14     _arriveLabel1.text = @"";
15     _arriveLabel2.text = @"";
16     _arriveLabel3.text = @"";
17     _startLabel1.text = @"";
18     _startLabel2.text = @"";
19     _startLabel3.text = @"";
20 }
```

escape:

Il metodo *escape*, controlla i dati inseriti dall'utente e ne verifica la correttezza; nel caso superi il controllo, sarà inviata una *Notification* contenente i punti di partenza e arrivo. La *Notification* sarà poi presa in carica, da *RoutePlanningViewController*.

```
21 - (IBAction)escape:(id)sender {
22
23     if(_startLocation && _arriveLocation) {
24         if([_startLocation distanceFromLocation:_arriveLocation] < 55000){
25             //escape from fog
26             NSArray *locations = [NSArray arrayWithObjects:_startLocation,
       _arriveLocation, nil];
27             NSNotification *escape = [NSNotification notificationWithName:@"
       escape" object:locations];
28             //Posso scegliere anche una queue diversa
29             [[NSNotificationQueue defaultQueue] enqueueNotification:escape
       postingStyle:NSPostWhenIdle];
30             //If iPhone, make dismiss for modal segue
31             if ([[UIDevice currentDevice].model rangeOfString:@"iPhone"].
       location != NSNotFound) {
32                 //Device is iPhone
33                 [self dismissViewControllerAnimated:YES completion:nil];
34             }
35         } else {
36             UIAlertView * alert = [[UIAlertView alloc]initWithTitle:@"Fog
       Escaping" message:@"Distanza maggiore di 55Km, non permesso." delegate:
       nil cancelButtonTitle:@"OK" otherButtonTitles: nil];
37             [alert show];
38         }
39     } else {
40         UIAlertView * alert = [[UIAlertView alloc]initWithTitle:@"Fog
       Escaping" message:@"Errore\nInserisci tutti i campi!" delegate:nil
       cancelButtonTitle:@"OK" otherButtonTitles: nil];
41         [alert show];
42     }
43 }
```

cancel:

Il metodo si occupa di chiudere la View, per ritornare alla precedente. Questo metodo è utilizzato solo dai device iPhone.

```
45 - (IBAction)cancel:(id)sender {
46     [self dismissViewControllerAnimated:YES completion:nil];
47 }
```

textFieldShouldReturn:

Rappresenta uno dei metodi *Delegate* della TextField. Viene richiamato alla pressione del pulsante invio, sulla tastiera del Sistema Operativo. Il metodo si occupa di creare delle descrizioni per il punto di partenza e arrivo, da inserire direttamente nella View come controllo visivo per l'Utente. Verrà quindi richiamato il metodo di classe *geocodeAddress:forDestination*, che si occuperà di generare queste informazioni.

```
48 - (BOOL)textFieldShouldReturn:(UITextField *)textField {
49     NSString *destination = nil;
50     if([textField isEqual:_startTextField]){
51         //start text field is editing
52         destination = @"start";
53     } else {
54         //arrive text field is editing
55         destination = @"arrive";
56     }
57     NSLog(@"Text should change");
58     [self geocodeAddress:textField.text forDestination:destination];
59     NSLog(@"Text should return");
60     [textField resignFirstResponder];
61     return YES;
62 }
```

textFieldShouldClear:

Rappresenta il secondo metodo *Delegate* della TextField. Viene richiamato nel momento in cui si preme la x a lato della TextField. Il metodo si occupa di ripristinare il contenuto delle label riguardanti l'inserimento del punto di partenza o di arrivo.

```
63 - (BOOL)textFieldShouldClear:(UITextField *)textField {
64     NSLog(@"Text should clear");
65     if([textField isEqual:_startTextField]){
```

```
66          //start text field clearing
67          _startLabel1.text = @"";
68          _startLabel2.text = @"";
69          _startLabel3.text = @"";
70      } else {
71          //arrive text field clearing
72          _arriveLabel1.text = @"";
73          _arriveLabel2.text = @"";
74          _arriveLabel3.text = @"";
75      }
76      return YES;
77  }
```

geocodeAddress:forDestination:

Il metodo viene utilizzato per ricavare le informazioni di localizzazione, su una stringa passata in ingresso. Viene effettuata la richiesta all'oggetto *CLGeocoder*, che restituirà le descrizioni relative.

```
78  - (void) geocodeAddress:(NSString*)address forDestination:(NSString*)
        destination {
79      if(!_placeResults) _placeResults = [[NSMutableArray alloc]init];
80      [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:
        YES];
81      if([destination isEqualToString:@"arrive"]){
82          [_arriveActivityIndicator startAnimating];
83      } else {
84          [_startActivityIndicator startAnimating];
85      }
86      [_placeResults removeAllObjects];
87      CLGeocoder *geocoder = [[CLGeocoder alloc]init];
88      [geocoder geocodeAddressString:address
89                      completionHandler:^(NSArray* placemarks, NSError* error){
90                          for (CLPlacemark* aPlacemark in placemarks)
91                          {
92                              [_placeResults addObject:aPlacemark];
93                          }
94                          //load results into Lable
95                          if([_placeResults count]>0){
96                              CLPlacemark *place = _placeResults[0];
97                              [self printPlaceMarkIntoView:place andDestination
        :destination];
98                          }
99                      }];
100 }
```

printPlaceMarkIntoView:

Il metodo si occupa di prendere in ingresso gli oggetti *CLPlacemark* rappresentanti le informazioni riguardanti il punto di partenza e di arrivo. Le informazioni sono quindi inserite all'interno delle label relative, e visualizzate all'interno della View.

```
102 - (void) printPlaceMarkIntoView:(CLPlacemark*) place andDestination:(
        NSString*)destination{
103     CLLocationCoordinate2D loc = place.location.coordinate;
104     NSDictionary *dict = place.addressDictionary;
105     NSArray *formattedAddress = dict[@"FormattedAddressLines"];
106     NSMutableString *addressFormatted = [@"" mutableCopy];
107     if([formattedAddress count]>0){
108         addressFormatted = [formattedAddress[0]mutableCopy];
109         for (int i=1; i<[formattedAddress count]; i++){
110             [addressFormatted appendString:[NSString stringWithFormat:@", %@
        ",formattedAddress[i]]];
111         }
112     }
113     if([destination isEqualToString:@"arrive"]){
114         [_arriveActivityIndicator stopAnimating];
115         _arriveLabel1.text = addressFormatted;
116         _arriveLabel2.text = place.addressDictionary[@"State"];
117         _arriveLabel3.text = geoInfo;
118         _arriveLocation = place.location;
119     } else {
120         [_startActivityIndicator stopAnimating];
121         _startLabel1.text = addressFormatted;
122         _startLabel2.text = place.addressDictionary[@"State"];
123         _startLabel3.text = geoInfo;
124         _startLocation = place.location;
125     }
126     [[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:NO
        ];
127 }
```

4.3.8 Grid

L'Oggetto *Grid*, rappresenta la griglia che viene creata per la ricerca di un percorso alternativo. La sua interfaccia dichiara i metodi pubblici necessari al funzionamento.

```
128 @interface Grid : NSObject
129 - (id) initGridWithRoute:(NSArray*)route;
130 - (NSArray*) getGridPoints;
131 - (NSDictionary*) getGridDictionary;
132 - (NSArray*) findSolutions;
133 - (void) gridReady:(NSDictionary*)grid;
134 @end
```

Variabili d'Istanza

Sono importate molte Librerie, tra le quali è presente PESGraph, per il calcolo dell'algoritmo di Dijskra. Sono inoltre dichiarate le variabili necessarie per la configurazione dei vari stati, che può assumere la classe.

```
135 #import "Grid.h"
136 #import "RoutePoint.h"
137 #import "ControlPoint.h"
138 #import "FogRequest.h"
139 #import <CoreLocation/CoreLocation.h>
140 #import <MapKit/MapKit.h>
141
142 #import "PESGraph.h"
143 #import "PESGraphNode.h"
144 #import "PESGraphEdge.h"
145 #import "PESGraphRoute.h"
146 #import "PESGraphRouteStep.h"
147
148 @interface Grid()
149 @property (nonatomic, strong) NSMutableArray *controlPointsInRoutes;
150 @property (nonatomic, strong) NSMutableDictionary *gridDictionary;
151 @property (nonatomic, strong) CLLocation *startLocation;
152 @property (nonatomic ,strong) CLLocation *arrivalLocation;
153 @property (nonatomic, strong) NSMutableArray *solutionSteps;
154 @end
155
156 @implementation Grid {
157     int MAX_DISTANCE_BETWEEN_POINTS;
158     int MULTIPLIER;
159     int maxRow;
160     int maxColumn;
161     int distanceCounter;
162     int actualKeyNumber;
163     NSArray *allKeys;
164     NSString *arrivalPointInGrid;
```

```
165     NSString *startPointInGrid;
166 }
```

initGridWithRoute:

Viene creato un metodo per l'inizializzazione delle variabili necessarie all'oggetto. Viene richiamato il metodo di classe *calculateBoundsPointsWithRoute*, che calcolerà i punti rappresentanti i margini esterni della griglia (Figura 3.1); viene inoltre richiamato il metodo *createControlPointsGridWithBounds*, che si occupa di creare tutti i *Punti di Controllo* all'interno dei margini esterni (Figura 3.2). Dopo aver creato la griglia dei *Punti di Controllo*, viene richiamato il metodo *requestFogData*, che si occupa di richiedere i dati sulle Nebbie per i *Punti di Controllo*.

```
168 - (id) initGridWithRoute:(NSArray *)route {
169     self = [super init];
170     MAX_DISTANCE_BETWEEN_POINTS = 2000; //2km standard distance
171     MULTIPLIER = 1;
172     maxRow = -1;
173     maxColumn = -1;
174     distanceCounter = 0;
175     _controlPointsInRoutes = [[self calculateBoundsPointsWithRoute:route]
        mutableCopy];
176     _controlPointsInRoutes = [[self createControlPointsGridWithBounds:
        _controlPointsInRoutes andRoute:route]mutableCopy];
177     allKeys = [_gridDictionary allKeys]; //important here
178     [self requestFogData];
179     return self;
180 }
```

requestFogData:

Il metodo richiama l'oggetto *FogRequest* che si occuperà di generare una richiesta HTTP per la griglia dei *Punti di Controllo*.

```
181 - (void) requestFogData {
182     FogRequest *fogReq = [[FogRequest alloc]init];
183     [fogReq requestDensityOfFogWithGrid:_gridDictionary];
184 }
```

gridReady:

Quando il Server risponderà alla richiesta effettuata tramite l'oggetto *FogRequest*, verrà richiamato il seguente metodo. Esso si occupa di inserire i dati ambientali di Nebbia ricevuti, all'interno della libreria PESGraph per il calcolo della soluzione.

4.3.9 insertAllGridIntoDijskraAlgorithm

In questo metodo, viene fatto il lavoro più delicato; cioè inserire i pesi degli archi diretti ai nodi adiacenti. Viene seguito il ragionamento presente nel Capitolo 3.4.

Per il calcolo dell'algoritmo di Dijskra, viene utilizzata la libreria PESGraph; essa prende in ingresso il nome dell'arco (riga 195), e il peso dello stesso (riga 205). Quando per ogni nodo sono salvate le informazioni sul peso degli archi diretti ai nodi adiacenti, inizia il calcolo della soluzione (riga 213).

```
185 - (void) insertAllGridIntoDijskraAlgorithm {
186     //insert the grid
187     PESGraph *graph = [[PESGraph alloc] init];
188     //add all the weight to the graph's point
189     for (NSString *key in allKeys){
190         ControlPoint *cp = _gridDictionary[key];
191         //find distance from up point
192         if (([cp.rowNumber intValue]-1) >= 0){
193             NSString *upKey = [NSString stringWithFormat:@"%d-%d",[cp.
    rowNumber intValue]-1,[cp.colNumber intValue]];
194             ControlPoint *up = [_gridDictionary objectForKey:upKey];
195             NSString *bidirectional = [NSString stringWithFormat:@"%@ <--> %
    @",key,upKey];
196             int fogLevel = 0;
197             if([cp.levelOfFog intValue]>[up.levelOfFog intValue]) {
198                 fogLevel = ([cp.levelOfFog intValue]*fogMultiplier)+
    routeMultiplier;
199             } else {
200                 fogLevel = ([up.levelOfFog intValue]*fogMultiplier)+
    routeMultiplier;
201             }
202             cp.upFogValue = [NSNumber numberWithInt:fogLevel-1];
203             PESGraphNode *n1 = [PESGraphNode nodeWithIdentifier:key];
204             PESGraphNode *n2 = [PESGraphNode nodeWithIdentifier:upKey];
```

```
205             [graph addBiDirectionalEdge:[PESGraphEdge edgeWithName:
    bidirectional andWeight:[NSNumber numberWithInt:fogLevel]] fromNode:n1
    toNode:n2];
206         }
207        .... (Molto simile per ognuno dei punti di adiacenza)
208     }
209     PESGraphNode *startNode = [graph nodeInGraphWithIdentifier:
    startPointInGrid];
210     PESGraphNode *arrivalNode = [graph nodeInGraphWithIdentifier:
    arrivalPointInGrid];
211     if(arrivalNode && startNode) {
212         PESGraphRoute *route = [graph shortestRouteFromNode:startNode toNode
    :arrivalNode];
213         NSArray *steps = route.steps;
214         _solutionSteps = [[NSMutableArray alloc]initWithCapacity:[steps
    count]];
215         for (PESGraphRouteStep *graphStep in steps){
216             PESGraphNode *stepNode = graphStep.node;
217             [_solutionSteps addObject:_gridDictionary[stepNode.identifier]];
218         }
219     }
220 }
```

4.3.10 FogRequest

L'oggetto *FogRequest*, è imputato alla generazione delle richieste HTTP verso il Server. *FogRequest* ha la particolarità d'implementare il pattern Singleton; esso permette di avere un'unica istanza della classe, con benefici sulle prestazioni. Descriverò solo i metodi per l'inserimento della segnalazione di Nebbia e di ricerca del percorso alternativo, questo perché per il login e la registrazione, non c'è nulla di sostanzialmente diverso.

L'interfaccia descrive i metodi pubblici che sono implementati.

```
1 #import <Foundation/Foundation.h>
2 #import <CoreLocation/CoreLocation.h>
3
4 @interface FogRequest : NSObject
5 - (void) requestDensityOfFogWithGrid:(NSMutableDictionary*) grid;
6 - (void) registrationWithEmail:(NSString *)email name:(NSString*)name
    lastName:(NSString*)lastName andPassword:(NSString*)password;
7 - (void) loginWithEmail:(NSString *)email andPassword:(NSString*)password;
8 - (void) insertFogDataWithCoordinate:(CLLocationCoordinate2D)point
    andFogValue:(int) fogValue;
```

```
9  + (FogRequest *)sharedInstance; //singleton
10 @end
```

Variabili d'Istanza

Oltre alle variabili necessarie per la memorizzazione della griglia e della soluzione, viene definita la stringa che sarà utilizzata per le richieste HTTP secondo il protocollo di comunicazione (Cap. 4.1.3).

```
11 #import "FogRequest.h"
12 #import "ControlPoint.h"
13 #import "RoutePoint.h"
14
15 #define SERVER_PATH @"http://fogescaping.esy.es/request.php?service="
16
17 @interface FogRequest()
18 @property (nonatomic, strong) NSMutableDictionary *gridDictionary;
19 @property (nonatomic, strong) NSMutableDictionary *routeDictionary;
20 @end
```

Implementazione Singleton

I metodi sottostanti garantiscono una corretta implementazione del pattern *Singleton*.

```
21 #pragma mark - shared instance for singleton
22 + (FogRequest *)sharedInstance {
23     static dispatch_once_t pred;
24     static FogRequest *shared = nil;
25     dispatch_once(&pred, ^{
26         shared = [[super alloc] init];
27     });
28     return shared;
29 }
30
31 - (id)copyWithZone:(NSZone*)zone {
32     return self;
33 }
```

requestDensityOfFogWithGrid:

Il metodo prende in ingresso un Array rappresentante la griglia dei *Punti di Controllo*, per il quale fare la richiesta dei dati ambientali. Successivamente viene costruito un JSON conforme al protocollo di comunicazione (Listing:4.10) e viene richiamato il metodo di classe *getFogDataForGridJson*.

```
34 - (void)requestDensityOfFogWithGrid:(NSMutableDictionary*) grid {
35     _gridDictionary = grid;
36     NSArray *keys = [grid allKeys];
37 
38     NSMutableArray *gridArray = [[NSMutableArray alloc]init];
39     //create a grid json with control point information
40     for (NSString *key in keys){
41         ControlPoint *cp = grid[key];
42         NSMutableDictionary *dict = [[NSMutableDictionary alloc]init];
43         [dict setObject:[NSNumber numberWithFloat:cp.location.coordinate.
       latitude] forKey:@"lat"];
44         [dict setObject:[NSNumber numberWithFloat:cp.location.coordinate.
       longitude] forKey:@"lng"];
45         [dict setObject:key forKey:@"point_id"];
46         [gridArray addObject:dict];
47     }
48     NSDictionary *gridToSend = [[NSDictionary alloc]initWithObjectsAndKeys:
       gridArray,@"points", nil];
49     NSData *jsonData = [self transformDictionaryIntoJson:gridToSend];
50     [self getFogDataForGridJson:jsonData];
51 }
```

getFogDataForGridJson

Il metodo permette di inviare una richiesta asincrona al servizio *checkForFogValues*. Quando arriverà la risposta, essa sarà gestita dal *completionHandler* che invierà la *Notification* adeguata. La *Notification* sarà gestita dalla classe *RoutePlanningViewController*.

```
52 - (void) getFogDataForGridJson:(NSData*)requestData {
53     NSString *urlAsString = [NSString stringWithFormat:@"%@checkForFogValues
       ",SERVER_PATH];
54     NSURL *url = [NSURL URLWithString:urlAsString];
55     NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
```

```
56      [request setHTTPMethod:@"POST"];
57      [request setValue:@"application/json" forHTTPHeaderField:@"Accept"];
58      [request setValue:@"application/json" forHTTPHeaderField:@"Content-Type"
        ];
59      [request setValue:[NSString stringWithFormat:@"%lu", (unsigned long)[
        requestData length]] forHTTPHeaderField:@"Content-Length"];
60      [request setHTTPBody: requestData];
61
62      NSOperationQueue *queue = [[NSOperationQueue alloc] init];
63
64      [NSURLConnection sendAsynchronousRequest:request queue:queue
        completionHandler:^(NSURLResponse *response,
65
                 NSData *data, NSError *error) {
66          if ([data length] >0 && error == nil){
67              [self setFogDataInGrid:data];
68          }
69          else if ([data length] == 0 && error == nil){
70              //send a notification when server response is empty
71              NSNotification *serverError = [NSNotification
        notificationWithName:@"server-error" object:nil];
72              [[NSNotificationQueue defaultQueue] enqueueNotification:
        serverError postingStyle:NSPostNow];
73          }
74          else if (error != nil){
75              //send a notification for connection error
76              NSNotification *connectionError = [NSNotification
        notificationWithName:@"connection-error" object:nil];
77              [[NSNotificationQueue defaultQueue] enqueueNotification:
        connectionError postingStyle:NSPostNow];
78          } }];
79  }
```

insertFogDataWithCoordinate:andFogValue:

Il metodo permette di inviare una richiesta asincrona al servizio *insertFog*. Viene composto un JSON conforme al protocollo di comunicazione (Listing:4.8) e viene gestita la risposta proveniente dal Sever.

```
80  - (void) insertFogDataWithCoordinate:(CLLocationCoordinate2D)point
        andFogValue:(int) fogValue{
81
82      //get userID from NSUSerDefault
83      NSDictionary *login = [[NSUserDefaults standardUserDefaults]
        dictionaryForKey:@"login-options"];
```

```
84      NSString * userID = [login objectForKey:@"userID"];
85      NSString *jsonRequest = [NSString stringWithFormat:@"{\"userID\":\"%@
        \",\"lat\":\"%f\",\"lng\":\"%f\",\"fogValue\":\"%d\"}",userID, point.
        latitude,point.longitude, fogValue];
86      NSData *requestData = [NSData dataWithBytes:[jsonRequest UTF8String]
        length:[jsonRequest length]];
87      NSString *urlAsString = [NSString stringWithFormat:@"%@insertFog",
        SERVER_PATH];
88      NSURL *url = [NSURL URLWithString:urlAsString];
89      NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:url];
90      [request setHTTPMethod:@"POST"];
91      [request setValue:@"application/json" forHTTPHeaderField:@"Accept"];
92      [request setValue:@"application/json" forHTTPHeaderField:@"Content-Type"
        ];
93      [request setValue:[NSString stringWithFormat:@"%lu", (unsigned long)[
        requestData length]] forHTTPHeaderField:@"Content-Length"];
94      [request setHTTPBody: requestData];
95
96      NSOperationQueue *queue = [[NSOperationQueue alloc] init];
97
98      [NSURLConnection sendAsynchronousRequest:request queue:queue
        completionHandler:^(NSURLResponse *response,
99
                 NSData *data, NSError *error) {
100         if ([data length] >0 && error == nil){
101             dispatch_async(dispatch_get_global_queue(
        DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
102                 dispatch_async(dispatch_get_main_queue(), ^{
103                     UIAlertView * alert = [[UIAlertView alloc]initWithTitle:
        @"Fog Escaping" message:@"Segnalazione Inviata con Successo" delegate:
        nil cancelButtonTitle:@"OK" otherButtonTitles: nil];
104                     [alert show];
105                 });
106             });
107         }
108         else if ([data length] == 0 && error == nil){
109             NSLog(@"Nothing was downloaded."); }
110         else if (error != nil){
111             //send a notification for connection error
112             NSNotification *connectionError = [NSNotification
        notificationWithName:@"connection-error" object:nil];
113             [[NSNotificationQueue defaultQueue] enqueueNotification:
        connectionError postingStyle:NSPostNow];
114         } }];
115 }
```

4.3.11 CustomAnnotationPoint

L'oggetto *CustomAnnotationPoint*, rappresenta una annotazione sulla mappa; è stato necessario crearlo, per permettere a una annotazione di avere alcuni valori interni che ne variassero l'aspetto grafico. Viene utilizzato per i punti della griglia che contengono un valore di Nebbia.

L'interfaccia dichiara la *Property* necessaria per il valore di Nebbia; è inoltre necessario, per la corretta visualizzazione, che la classe sia del tipo *MKPointAnnotation*.

```
#import <MapKit/MapKit.h>

@interface CustomPointAnnotation : MKPointAnnotation

@property (copy) NSNumber *fogValue;

@end
```

4.3.12 ControlPoint

L'oggetto *ControlPoint* rappresenta un *Punto di Controllo* (Cap. 3.2.2). Il *ControlPoint* è essenziale per la ricerca di un percorso alternativo, possiede quindi molte variabili che saranno utilizzate in tutto il processo.

```
#import <Foundation/Foundation.h>
#import <CoreLocation/CoreLocation.h>

@interface ControlPoint : NSObject
@property (nonatomic, assign) int distanceFromArrival;
@property (nonatomic, assign) int distanceFromStart;
@property (nonatomic, assign) int distanceFromPoint;
@property (nonatomic, assign) BOOL alreadyPassed;
@property (nonatomic, strong) NSNumber* upFogValue;
@property (nonatomic, strong) NSNumber* downFogValue;
@property (nonatomic, strong) NSNumber* rightFogValue;
@property (nonatomic, strong) NSNumber* LeftFogValue;
@property (nonatomic, strong) NSNumber* upRightCornerFogValue;
@property (nonatomic, strong) NSNumber* upLeftCornerFogValue;
@property (nonatomic, strong) NSNumber* downRightCornerFogValue;
@property (nonatomic, strong) NSNumber* downLeftCornerFogValue;
@property (nonatomic, strong) NSNumber* rowNumber;
```

```
@property (nonatomic, strong) NSNumber* colNumber;
@property (nonatomic, strong) CLLocation* location;
@property (nonatomic, strong) NSNumber *isPresentFog;
@property (nonatomic, strong) NSNumber *levelOfFog;
@end
```

L'oggetto possiede anche un metodo d'inizializzazione, che imposta i valori prestabiliti delle *Property*.

```
#import "ControlPoint.h"

@implementation ControlPoint
-(id) init {
    self = [super init];
    _levelOfFog = [NSNumber numberWithInt:0];
    _upFogValue = [NSNumber numberWithInt:0];
    _downFogValue = [NSNumber numberWithInt:0];
    _rightFogValue = [NSNumber numberWithInt:0];
    _LeftFogValue = [NSNumber numberWithInt:0];
    _downLeftCornerFogValue = [NSNumber numberWithInt:0];
    _downRightCornerFogValue = [NSNumber numberWithInt:0];
    _upLeftCornerFogValue = [NSNumber numberWithInt:0];
    _upRightCornerFogValue = [NSNumber numberWithInt:0];
    _alreadyPassed = NO;
    return self;
}
@end
```

4.3.13 Routepoint

L'oggetto *RoutePoint* rappresenta uno dei punti di direzione restituiti dai servizi di direzionamento Apple. Viene utilizzato per garantire ulteriori funzionalità: sapere il livello di nebbia presente e sapere se fa parte di una inversione a U.

```
#import <Foundation/Foundation.h>
#import <CoreLocation/CoreLocation.h>

@interface RoutePoint : NSObject
@property (nonatomic, strong) NSNumber* number;
@property (nonatomic, strong) CLLocation *location;
@property (nonatomic, strong) NSNumber* isPresentFog;
@property (nonatomic, strong) NSNumber* levelOfFog;
```

```
@property (nonatomic, strong) NSNumber* fogStopAtRoutePointNumber;
@property (nonatomic, assign) BOOL isAnInversionPoint;
@end
```

Possiede un metodo d'inizializzazione che imposta i valori prestabiliti delle *Property*.

```
#import "RoutePoint.h"

@implementation RoutePoint

- (id) init {
    self = [super init];
    _location = nil;
    _isPresentFog = nil;
    _levelOfFog = nil;
    _fogStopAtRoutePointNumber = nil;
    _isAnInversionPoint = NO;
    return self;
}

@end
```

4.3.14 AlternativeRouteSegment

L'oggetto rappresenta l'insieme dei punti che fanno parte di un percorso tra un punto di soluzione e un altro. Data la natura asincrona delle richieste di direzionamento, era necessario tenere traccia di ognuna di queste soluzioni e successivamente unirle, per creare il percorso alternativo definitivo. L'interfaccia contiene i punti del segmento e un numero, *startSolutionPoint*, che rappresenta l'ordine dello stesso rispetto al percorso totale.

```
#import <Foundation/Foundation.h>

@interface AlternateRouteSegment : NSObject
@property (nonatomic, strong) NSArray *segmentPoints;
@property (nonatomic, strong) NSNumber *startSolutionPoint;
- (id) initWithPoints:(NSArray*) points andStartSolutionPoint:(int)
    startSolutionPoint;
@end
```

Possiede anche un metodo d'inizializzazione che verrà richiamato per generare ognuno di questi segmenti.

```
8  - (id) initWithPoints:(NSArray*) points andStartSolutionPoint:(int)
       startSolutionPoint {
9      self = [super init];
10     _segmentPoints = points;
11     _startSolutionPoint = [NSNumber numberWithInt:startSolutionPoint];
12     return self;
13 }
```

4.3.15 PESGraph

Per il progetto di Tesi era richiesto l'utilizzo dell'algoritmo di Dijskra, si è voluto utilizzare una libreria molto conosciuta e ottimizzata chiamata *PESGraph*. La libreria permette di creare Nodi, aggiungere Archi pesati di connessione fra i Nodi e successivamente di calcolare il cammino minimo con il Grafo creato dall'insieme dei Nodi.

Gli oggetti che sono presenti nella libreria sono:

- PESGraph
- PESGraphNode
- PESGraphEdge
- PESGraphRoute
- PESGraphRouteStep

La parte d'implementazione utilizzata per l'applicazione *Fog Escaping* è presente nella classe *Grid* (Cap. 4.3.8).

PESGraph

La classe permette di creare un Grafo fatto di Nodi *PESGraphNode* e collegati da Archi *PESGraphEdge*. Possiede il metodo *shortestRouteFromNode:toNode:* che calcola il cammino minimo all'interno del Grafo.

PESGraphNode

Rappresenta un Nodo all'interno di un Grafo. Possiede un identificatore e un titolo. Viene creato con il seguente codice:

```
1  PESGraphNode *n1 = [PESGraphNode nodeWithIdentifier:key];
```

PESGraphEdge

Rappresenta l'arco pesato tra un Nodo e un altro. Può essere inizializzato come unidirezionale o bidirezionale. Per l'implementazione si è utilizzata l'impostazione bidirezionale.

```
1  NSString *bidirectional = [NSString stringWithFormat:@"%@ <--> %@",key,
     upKey];
2  [PESGraphEdge edgeWithName:bidirectional andWeight:[NSNumber numberWithInt
     :fogLevel]] fromNode:n1 toNode:n2];
```

PESGraphRoute e PESGraphRouteStep

Rappresenta l'oggetto di ritorno dal metodo *shortestRouteFromNode:toNode*; è quindi un insieme di passi *PESGraphRouteStep*, rappresentanti il cammino minimo all'interno del Grafo. *PESGraphRouteStep* rappresenta uno degli step di soluzione, contenente i nomi dei due nodi di passaggio. Nell'implementazione i *PESGraphRouteStep* sono scorsi uno a uno, e inseriti nel Dictionary delle soluzioni.

```
1  PESGraphRoute *route = [graph shortestRouteFromNode:startNode toNode:
     arrivalNode];
2        NSArray *steps = route.steps;
3        _solutionSteps = [[NSMutableArray alloc]initWithCapacity:[steps
     count]];
4        for (PESGraphRouteStep *graphStep in steps){
5            PESGraphNode *stepNode = graphStep.node;
6            [_solutionSteps addObject:_gridDictionary[stepNode.identifier]];
7        }
```

Capitolo 5

Studi sull'Image Recognition

Durante la fase di sviluppo, sono stati condotti alcuni test per la futura implementazione di un sistema, capace di riconoscere automaticamente la densità della Nebbia, in base ad alcuni scatti effettuati con i dispositivi mobili di Apple: iPhone e iPad.

L'idea è di utilizzare tecniche di *Object Recognition* e *Image Analysis*, assieme alla *geo-localizzazione* dell'Utente, per generare un valore affidabile di visibilità. L'Utente quindi scatta una foto a un oggetto a distanza ravvicinata, poi si allontana fino a che l'applicazione non gli dice di fermarsi e scattare un'altra foto. Il Sistema tramite la distanza, il riconoscimento dell'oggetto e la comparazione dei livelli di colore, può trovare un valore di visibilità affidabile.

5.1 OpenCV

Durante la fase di test sulle tecnologie è stata utilizzata la libreria *OpenCV* con licenza open source BSD. La libreria offre la maggior parte delle utility necessarie, per raggiungere gli scopi di *riconoscimento oggetti* ed *analisi dei fotogrammi*. La libreria è nata nel 1999 come progetto di ricerca di Intel, ed essendo cross platform, permetterebbe future implementazioni sui principali sistemi operativi mobili (iOS e Android) oltre che su piattaforme desktop. La

libreria permette con poche righe di codice di collegarsi ai sistemi di raccolta video, e tramite alcuni filtri è possibile riconoscere e seguire oggetti come semplici cerchi rossi durante uno stream video.

5.2 Object Recognition

I test sono stati eseguiti, per capire se era possibile fare il tracking di un oggetto in movimento sui dispositivi mobili.

Riconoscimento di un oggetto in movimento

Per seguire un oggetto in movimento, è necessario utilizzare alcuni filtri che permettono di isolare un determinato colore; per l'implementazione è stato scelto il rosso. Successivamente viene utilizzato il metodo *HoughCircles* che permette di identificare i cerchi all'interno dello stream video filtrato. Quando un cerchio viene identificato, viene contornato da un bordo bianco.

```
1  - (void)processImage:(Mat&)image;
2  {
3      //red filter
4      cv::Mat threshold_image;
5      cvtColor(image, threshold_image, CV_BGRA2BGR);
6      //dal rosso al rosa
7      cv::inRange(threshold_image, cv::Scalar(0, 0, 150), cv::Scalar(135, 110,
        255), threshold_image);
8      bitwise_not(threshold_image, threshold_image);
9
10     GaussianBlur( threshold_image, threshold_image, cv::Size(3, 3), 2, 2 );
11     vector<Vec3f> circles;
12     //Apply the Hough Transform to find the circles
13     HoughCircles( threshold_image, circles, CV_HOUGH_GRADIENT, 1,
       threshold_image.rows/8, 200, 40, 0, 0 );
14
15     NSMutableArray* circleArray = nil;
16     /// Draw the circles detected
17     for( size_t i = 0; i < circles.size(); i++ )
18     {
19         if(!circleArray) circleArray = [[NSMutableArray alloc]
       initWithCapacity:circles.size()];
```

```
20          cv::Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
21          int radius = cvRound(circles[i][2]);
22          // circle outline
23          circle( image, center, radius, Scalar(255,0,0), 3, 8, 0 );
24
25          //create the mask for cropped circle
26          cv::Mat mask = cv::Mat::zeros( image.rows, image.cols, CV_8UC1 );
27          circle( mask, center, radius, Scalar(255,0,0), -1, 8, 0 ); //-1
    means filled
28          Mat dst;
29          image.copyTo( dst, mask ); // copy values of img to dst if mask is >
     0.
30          //transform BGR to RGB
31          cvtColor(dst, dst, CV_BGR2RGB);
32          UIImage *img = [self UIImageFromCVMat:dst];
33          [circleArray addObject:img];
34      }
35  }
```

Com'è possibile vedere nella Figura 5.1, sono riconosciuti la maggior parte dei cerchi rossi presenti nello stream video.

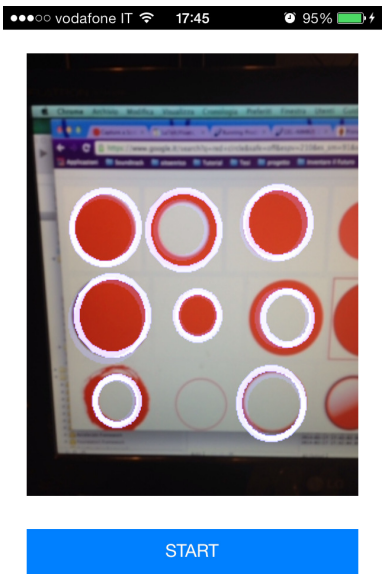


Figura 5.1: Test di Riconoscimento Oggetti.

5.3 Image Analyzer

L'idea è di comparare due immagini e successivamente, dopo aver tarato adeguatamente il dispositivo, trovare in modo automatico un valore di visibilità.

Il metodo *analyzeImages* carica due foto diverse, in seguito divide le foto in tre livelli distinti, uno per colore (red-green-blue). Successivamente carica gli istogrammi e provvede a comparare i livelli con i quattro metodi a disposizione:

- Correlation
- Chi-Square
- Intersection
- Bhattacharyya distance

```
- (void) analyzeImages {
    Mat imageoneHSV, imageTwoHSV;
    UIImage *uiImage1 = [UIImage imageNamed:@"fog.jpg"];
    UIImage *uiImage2 = [UIImage imageNamed:@"fog_ultra.jpg"];
    _image1.image = uiImage1;
    _image2.image = uiImage2;
    Mat imageOne = [self cvMatFromUIImage:uiImage1];
    Mat imageTwo = [self cvMatFromUIImage:uiImage2];

    //create vector for splitted rgb
    vector<Mat> bgr_planes1;
    vector<Mat> bgr_planes2;
    //Split the image in 3 places (B,G and R)
    split( imageOne, bgr_planes1 );
    split( imageTwo, bgr_planes2 );

    /// Establish the number of bins
    int histSize = 256;
    /// Set the ranges ( for B,G,R) )
    float range[] = { 0, 256 } ;
    const float* histRange = { range };

    bool uniform = true; bool accumulate = false;

    /// Histograms
```

```
26      Mat b_hist1;
27      Mat g_hist1;
28      Mat r_hist1;
29      Mat b_hist2;
30      Mat g_hist2;
31      Mat r_hist2;
32
33      /// Compute the histograms:
34      calcHist( &bgr_planes1[0], 1, 0, Mat(), b_hist1, 1, &histSize, &
        histRange, uniform, accumulate );
35      calcHist( &bgr_planes1[1], 1, 0, Mat(), g_hist1, 1, &histSize, &
        histRange, uniform, accumulate );
36      calcHist( &bgr_planes1[2], 1, 0, Mat(), r_hist1, 1, &histSize, &
        histRange, uniform, accumulate );
37      calcHist( &bgr_planes2[0], 1, 0, Mat(), b_hist2, 1, &histSize, &
        histRange, uniform, accumulate );
38      calcHist( &bgr_planes2[1], 1, 0, Mat(), g_hist2, 1, &histSize, &
        histRange, uniform, accumulate );
39      calcHist( &bgr_planes2[2], 1, 0, Mat(), r_hist2, 1, &histSize, &
        histRange, uniform, accumulate );
40
41      // Draw the histograms for B, G and R
42      int hist_w = 512; int hist_h = 400;
43      int bin_w = cvRound( (double) hist_w/histSize );
44
45      Mat histImage1( hist_h, hist_w, CV_8UC3, Scalar( 0,0,0) );
46      Mat histImage2( hist_h, hist_w, CV_8UC3, Scalar( 0,0,0) );
47
48      /// Normalize the result
49      normalize(b_hist1, b_hist1, 0, histImage1.rows, NORM_MINMAX, -1, Mat() )
        ;
50      normalize(g_hist1, g_hist1, 0, histImage1.rows, NORM_MINMAX, -1, Mat() )
        ;
51      normalize(r_hist1, r_hist1, 0, histImage1.rows, NORM_MINMAX, -1, Mat() )
        ;
52      /// Normalize the result
53      normalize(b_hist2, b_hist2, 0, histImage2.rows, NORM_MINMAX, -1, Mat() )
        ;
54      normalize(g_hist2, g_hist2, 0, histImage2.rows, NORM_MINMAX, -1, Mat() )
        ;
55      normalize(r_hist2, r_hist2, 0, histImage2.rows, NORM_MINMAX, -1, Mat() )
        ;
56
57      /// Draw for each channel
58      for( int i = 1; i < histSize; i++ )
59      {
60          line( histImage1, cv::Point( bin_w*(i-1), hist_h - cvRound(b_hist1.
```

```
    at<float>(i-1)) ) ,
61              cv::Point( bin_w*(i), hist_h - cvRound(b_hist1.at<float>(i)) ),
62              Scalar( 255, 0, 0), 2, 8, 0  );
63          line( histImage1, cv::Point( bin_w*(i-1), hist_h - cvRound(g_hist1.
    at<float>(i-1)) ) ,
64              cv::Point( bin_w*(i), hist_h - cvRound(g_hist1.at<float>(i)) ),
65              Scalar( 0, 255, 0), 2, 8, 0  );
66          line( histImage1, cv::Point( bin_w*(i-1), hist_h - cvRound(r_hist1.
    at<float>(i-1)) ) ,
67              cv::Point( bin_w*(i), hist_h - cvRound(r_hist1.at<float>(i)) ),
68              Scalar( 0, 0, 255), 2, 8, 0  );
69      }
70      for( int i = 1; i < histSize; i++ )
71      {
72          line( histImage2, cv::Point( bin_w*(i-1), hist_h - cvRound(b_hist2.
    at<float>(i-1)) ) ,
73              cv::Point( bin_w*(i), hist_h - cvRound(b_hist2.at<float>(i)) ),
74              Scalar( 255, 0, 0), 2, 8, 0  );
75          line( histImage2, cv::Point( bin_w*(i-1), hist_h - cvRound(g_hist2.
    at<float>(i-1)) ) ,
76              cv::Point( bin_w*(i), hist_h - cvRound(g_hist2.at<float>(i)) ),
77              Scalar( 0, 255, 0), 2, 8, 0  );
78          line( histImage2, cv::Point( bin_w*(i-1), hist_h - cvRound(r_hist2.
    at<float>(i-1)) ) ,
79              cv::Point( bin_w*(i), hist_h - cvRound(r_hist2.at<float>(i)) ),
80              Scalar( 0, 0, 255), 2, 8, 0  );
81      }
82
83      _hist1.image = [self UIImageFromCVMat:histImage1];
84      _hist2.image = [self UIImageFromCVMat:histImage2];
85
86      //print result for red compare
87      printf("************************** Red Compare\n");
88      for( int i = 0; i < 4; i++ )
89      { int compare_method = i;
90          double one_one = compareHist( r_hist1, r_hist1, compare_method );
91          double one_two = compareHist( r_hist1, r_hist2, compare_method );
92
93          printf( " Method [%d] r_one-r_one, r_one-r_two : %f, %f \n", i,
    one_one, one_two);
94      }
95      //print result for green compare
96      printf("************************** Green Compare\n");
97      for( int i = 0; i < 4; i++ )
98      { int compare_method = i;
99          double one_one = compareHist( g_hist1, g_hist1, compare_method );
100         double one_two = compareHist( g_hist1, g_hist2, compare_method );
```

```
101
102         printf( " Method [%d] g_one-g_one, g_one-g_two : %f, %f \n", i,
    one_one, one_two);
103     }
104     //print result for blue compare
105     printf("************************* Blue Compare\n");
106     for( int i = 0; i < 4; i++ )
107     { int compare_method = i;
108         double one_one = compareHist( b_hist1, b_hist1, compare_method );
109         double one_two = compareHist( b_hist1, b_hist2, compare_method );
110
111         printf( " Method [%d] b_one-b_one, b_one-b_two : %f, %f \n", i,
    one_one, one_two);
112     }
113     printf("************************* STOP");
114 }
```

Per ogni livello di colore, viene inserito sempre un campione di test, in modo da verificare che la comparazione fra lo stesso livello della medesima foto, dia un valore neutrale. Il secondo valore invece rappresenta il risultato della comparazione tra le due immagini. Com'è possibile notare, in tutti i casi e in tutti i livelli di colore, le comparazioni danno risultati soddisfacenti. Sarebbe quindi possibile utilizzare uno dei seguenti metodi, per comparare la prima immagine con la seconda e capire quale livello di visibilità esiste.

```
1  ************************* Red Compare
2   Method [0] r_one-r_one, r_one-r_two : 1.000000, -0.303090
3   Method [1] r_one-r_one, r_one-r_two : 0.000000, 3202443.748974
4   Method [2] r_one-r_one, r_one-r_two : 9907.996529, 63.392060
5   Method [3] r_one-r_one, r_one-r_two : 0.000000, 0.969744
6  ************************* Green Compare
7   Method [0] g_one-g_one, g_one-g_two : 1.000000, -0.290273
8   Method [1] g_one-g_one, g_one-g_two : 0.000000, 741814.764506
9   Method [2] g_one-g_one, g_one-g_two : 15948.307584, 170.305205
10  Method [3] g_one-g_one, g_one-g_two : 0.000000, 0.960473
11 ************************* Blue Compare
12  Method [0] b_one-b_one, b_one-b_two : 1.000000, -0.343736
13  Method [1] b_one-b_one, b_one-b_two : 0.000000, 808483704680.074097
14  Method [2] b_one-b_one, b_one-b_two : 13589.429940, 155.564171
15  Method [3] b_one-b_one, b_one-b_two : 0.000000, 0.963065
16 ************************* STOP
```

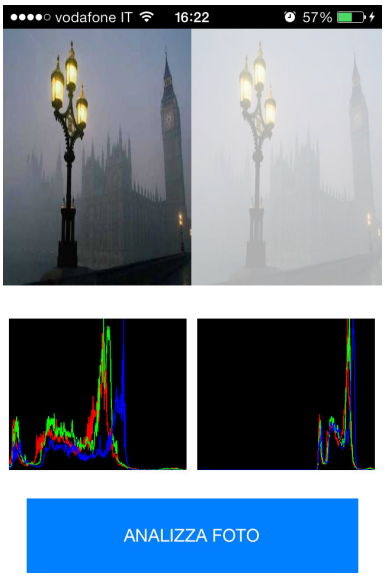


Figura 5.2: Test di Comparazione Immagini.

Capitolo 6

Validazione

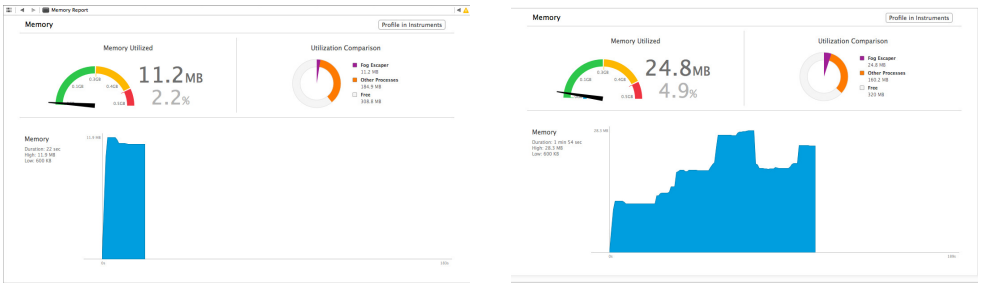
In questo capitolo si tratteranno le metodologie utilizzate per la validazione del progetto di Tesi. Nella prima sezione si prenderanno in considerazione le prestazioni del sistema e nella seconda saranno effettuate alcune comparazioni con applicazioni simili. Infine saranno presentati alcuni screenshot che rappresentano le funzionalità implementate nell'applicazione.

6.1 Prestazioni

Le prestazioni del sistema sono state monitorate in fase di debug, tramite la piattaforma *XCode 5*, e successivamente tramite *Instrument*. Instrument è un'applicazione che fa parte degli strumenti messi a disposizione da Apple per lo sviluppo; permette di verificare alcuni parametri come il consumo di batteria, la quantità di CPU utilizzata in un determinato momento e la quantità di pacchetti scambiati con la rete.

6.1.1 Utilizzo di Memoria

All'avvio l'applicazione utilizza circa 11.2 MB di memoria, pari al 2% sulla memoria totale. Durante la fase più impegnativa, ovvero il caricamento dei componenti grafici sulla mappa, l'applicazione arriva a utilizzare circa 25 MB, pari al 4.9% sulla memoria totale.



(a) Apertura Applicazione. (b) Caricamento Grafica.

Figura 6.1: Timeline dell'Utilizzo di Memoria.

6.1.2 Utilizzo CPU

In fase di avvio l'applicazione carica i componenti grafici per visualizzare le mappe, raggiunge quindi un utilizzo di CPU pari al 52.4%; successivamente l'applicazione si stabilizzerà, rimanendo circa sul 25%. La fase più importante, inizia con la richiesta di un percorso alternativo e raggiunge il suo punto massimo nel momento del calcolo dell'algoritmo di Dijskra, che fa registrare un consumo dell'80% sul totale.

CPU Activity

Time	Total Activity	Foreground App Activity	Audio Processing	Graphics
00:00 - 00:01.244	52.4%	68.3%	0.5%	23.2%
00:01.244 - 00:04.228	26.8%	15.1%	0.6%	12.1%
00:04.228 - 00:07.228	24.3%	0.6%	0.5%	13%
00:07.228 - 00:10.228	32.2%	13.7%	0.6%	16.6%
00:10.228 - 00:13.287	52.4%	10.1%	0.9%	28.7%
00:13.287 - 00:16.228	50.6%	5.3%	1.1%	14.9%
00:16.228 - 00:19.228	68.1%	58.5%	0.5%	18.7%
00:19.228 - 00:22.228	34.9%	14%	0.6%	21.3%
00:22.228 - 00:25.228	28.8%	16.1%	0.6%	15.3%
00:25.228 - 00:28.228	81.2%	90.6%	0.4%	7.2%
00:28.228 - 00:31.228	80.8%	90.7%	0.4%	6.2%
00:31.228 - 00:34.228	80.8%	90.9%	0.4%	6%
00:34.228 - 00:37.228	72.7%	92%	0.4%	5.9%
00:37.228 - 00:40.229	47.7%	28.9%	0.5%	12.3%
00:40.229 - 00:43.228	20.3%	1.5%	0.5%	1.7%

Figura 6.2: Test di Utilizzo CPU su iPhone.

In Figura 6.3 vi è un elenco dei thread attivi durante la ricerca di un percorso alternativo. I thread più importanti sono:

Thread 1 Creazione della Griglia dei *Punti di Controllo*;

Thread 18 Computazione dell'Algoritmo di Dijskra.

Thread 19 Richieste Asincrone HTTP;

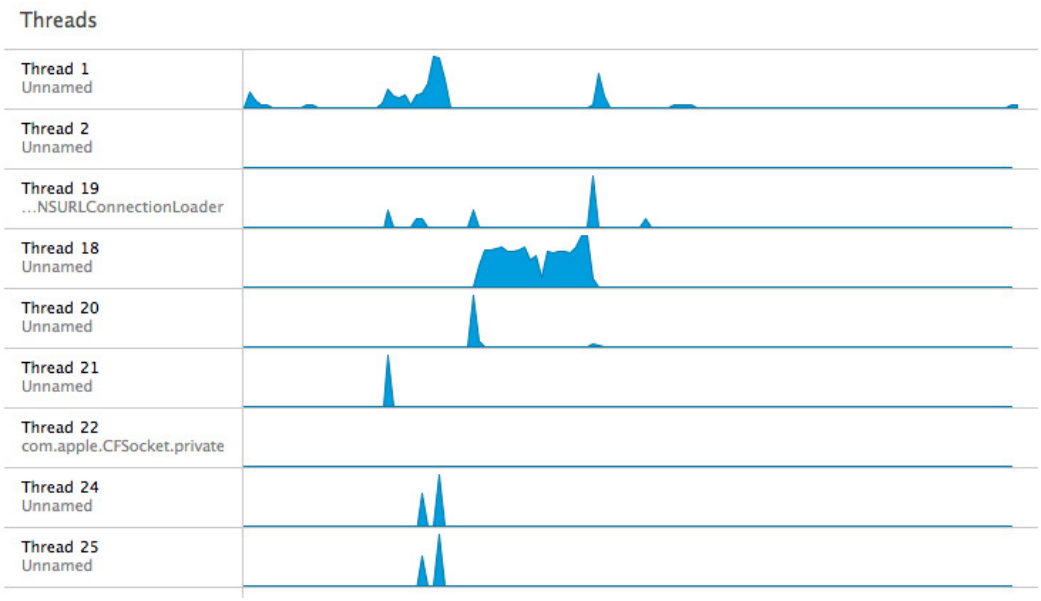


Figura 6.3: Timeline di Vita dei Thread.

6.1.3 Consumo di Energia

Per quanto riguarda il consumo di energia, bisogna differenziare l'utilizzo dell'applicazione con rete Wifi dall'utilizzo con rete cellulare. Come si può notare in Figura 6.4, durante l'utilizzo della rete Wifi, solo in un punto la richiesta di energia raggiunge un consumo di 1/20; mentre durante l'utilizzo della rete cellulare un consumo di 1/20 si ha nella maggior parte dei casi, e in un punto si registra un consumo di 8/20.

Il risultato non stupisce per nulla; infatti, sia la localizzazione dell'Utente, sia lo scambio di dati, sono più onerosi durante l'utilizzo della rete Cellulare.

Energy Consumption

Time	Energy Usage Level
00:14.292 - 00:15.285	0/20
00:15.285 - 00:16.285	0/20
00:16.285 - 00:17.989	0/20
00:17.989 - 00:18.989	8/20
00:18.989 - 00:19.989	1/20
00:19.989 - 00:20.268	1/20
00:20.268 - 00:21.268	0/20
00:21.268 - 00:22.268	0/20
00:22.268 - 00:23.268	0/20
00:23.268 - 00:24.268	0/20
00:24.268 - 00:25.316	0/20
00:25.316 - 00:26.316	1/20
00:26.316 - 00:27.316	1/20
00:27.316 - 00:28.316	1/20
00:28.316 - 00:29.316	1/20
00:29.316 - 00:30.339	1/20
00:30.339 - 00:31.339	1/20
00:31.339 - 00:32.339	1/20

(a) Uso di rete Cellulare.

Energy Consumption

Time	Energy Usage Level
00:10.937 - 00:11.937	1/20
00:11.937 - 00:12.937	0/20
00:12.937 - 00:13.938	0/20
00:13.938 - 00:15.079	0/20
00:15.079 - 00:16.079	0/20
00:16.079 - 00:17.079	0/20
00:17.079 - 00:18.079	0/20
00:18.079 - 00:18.933	0/20
00:18.933 - 00:19.933	0/20
00:19.933 - 00:20.933	0/20
00:20.933 - 00:21.933	0/20
00:21.933 - 00:22.933	0/20
00:22.933 - 00:23.939	0/20
00:23.939 - 00:24.939	0/20
00:24.939 - 00:25.939	0/20
00:25.939 - 00:26.939	0/20
00:26.939 - 00:27.939	0/20
00:27.939 - 00:28.939	0/20
00:28.939 - 00:29.939	0/20

(b) Uso di rete Wifi

Figura 6.4: Paragone del Consumo Energetico tra Rete Wifi e Cellulare.

6.1.4 Timeline di Esecuzione

In Figura 6.5 si possono benissimo notare i momenti diversi dell'esecuzione dell'applicazione:
Al secondo 15, l'applicazione invia la richiesta dei dati ambientali di Nebbia al Server. Circa dal secondo 37, l'applicazione riceve la risposta dal Server e inizia a calcolare l'Algoritmo di Dijskra. Circa al secondo 39, l'applicazione richiede ai servizi di direzionamento di Apple, il passaggio tra un punto e l'altro della soluzione.

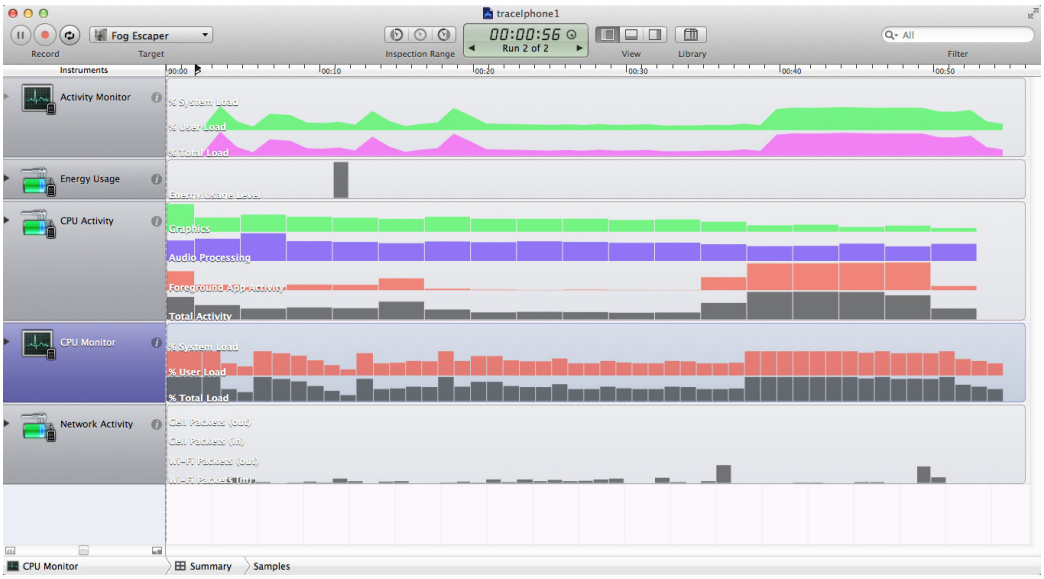


Figura 6.5: Test prestazioni su iPhone con rete wifi.

6.2 Comparazioni

Fog Escaping è stato confrontato con alcuni sistemi che utilizzano GPS, mappe e sistema di navigazione. Bisogna però far notare che ognuna delle applicazioni che è stata confrontata, utilizza un sistema di direzionamento completamente lato Server; mentre Fog Escaping, crea un percorso utilizzando un algoritmo che è a carico quasi solamente del device.

La comparazione è stata eseguita con l'applicazione Instrument tramite il dispositivo iPhone 4s. Sono lanciate le applicazioni una alla volta e per ognuna di esse viene richiesta una navigazione o una visualizzazione di dati sulla mappa.

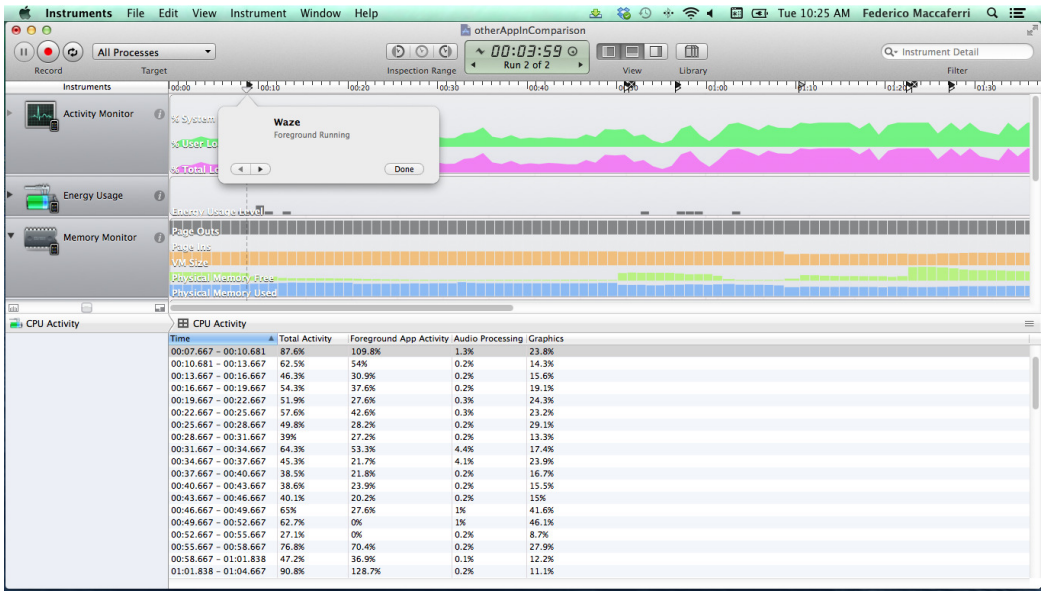


Figura 6.6: Comparazione con Instrument.

6.2.1 Comparazione con Waze

Waze è un'applicazione con paradigma Data Crowd-Sourcing, dove gli utenti si scambiano informazioni riguardanti il traffico. L'utilizzo di CPU di Waze è in media più oneroso di *Fog Escaping*; ma mentre *Fog Escaping* raggiunge un picco dell'80%, Waze rimane stabile tra il 40-60%. Questo è sicuramente imputabile alla grafica più onerosa e a un sistema di direzionamento che opera lato Server.
Durante l'esecuzione non si sono verificati *Memory Warning* (Sovraccarichi della memoria).

CPU Activity

Time	Total Activity	Foreground App Activity	Audio Processing	Graphics
00:07.667 - 00:10.681	87.6%	109.8%	1.3%	23.8%
00:10.681 - 00:13.667	62.5%	54%	0.2%	14.3%
00:13.667 - 00:16.667	46.3%	30.9%	0.2%	15.6%
00:16.667 - 00:19.667	54.3%	37.6%	0.2%	19.1%
00:19.667 - 00:22.667	51.9%	27.6%	0.3%	24.3%
00:22.667 - 00:25.667	57.6%	42.6%	0.3%	23.2%
00:25.667 - 00:28.667	49.8%	28.2%	0.2%	29.1%
00:28.667 - 00:31.667	39%	27.2%	0.2%	13.3%
00:31.667 - 00:34.667	64.3%	53.3%	4.4%	17.4%
00:34.667 - 00:37.667	45.3%	21.7%	4.1%	23.9%
00:37.667 - 00:40.667	38.5%	21.8%	0.2%	16.7%
00:40.667 - 00:43.667	38.6%	23.9%	0.2%	15.5%
00:43.667 - 00:46.667	40.1%	20.2%	0.2%	15%
00:46.667 - 00:49.667	65%	27.6%	1%	41.6%
00:49.667 - 00:52.667	62.7%	0%	1%	46.1%

Figura 6.7: Utilizzo CPU di Waze.

6.2.2 Comparazione con OpenSignal

OpenSignal è un'applicazione con paradigma di Data Crowd-Sourcing, dove gli Utenti possono segnalare lo stato di ricezione della rete cellulare. L'utilizzo di CPU di OpenSignal è decisamente maggiore di *Fog Escaping*, infatti, raggiunge un picco del 95%; inoltre durante questo picco viene notificato un *Memory Warning*, che indica chiaramente come il programma non sia stato ottimizzato.

CPU Activity

Time	Total Activity	Foreground App Activity	Audio Processing	Graphics
00:55.667 - 00:58.667	76.8%	70.4%	0.2%	27.9%
00:58.667 - 01:01.838	47.2%	36.9%	0.1%	12.2%
01:01.838 - 01:04.667	90.8%	128.7%	0.2%	11.1%
01:04.667 - 01:07.699	72.3%	60.8%	0.2%	35%
01:07.699 - 01:10.750	95.2%	122.7%	0.1%	15%
01:10.750 - 01:13.674	82.2%	85.5%	0.2%	18.2%
01:13.674 - 01:16.666	73.2%	69.8%	0.1%	29.1%
01:16.666 - 01:19.667	43.8%	24.5%	0.2%	19.7%
01:19.667 - 01:22.667	75.2%	0%	0.6%	51.1%
01:22.667 - 01:25.667	58.4%	0%	0.2%	36.8%

Figura 6.8: Utilizzo CPU di OpenSignal.

6.2.3 Comparazione con Weendy

Weendy è un'applicazione con paradigma di Data Crowd-Sourcing, dove gli Utenti possono segnalare lo stato delle condizioni climatiche; viene utilizzato sopratutto dai surfisti che cercano le condizioni ottimali per cavalcare l'onda. L'utilizzo di CPU di Weendy è molto simile a quello di *Fog Escaping* e non sono stati registrati *Memory Warning* durante l'esecuzione.

CPU Activity

Time	Total Activity	Foreground App Activity	Audio Processing	Graphics
01:22.667 - 01:25.667	58.4%	0%	0.2%	36.8%
01:25.667 - 01:28.669	59.1%	54.3%	0.7%	17.4%
01:28.669 - 01:31.667	46.8%	28.9%	0.2%	8.9%
01:31.667 - 01:34.667	39.8%	19.4%	0.2%	9.3%
01:34.667 - 01:37.667	84.2%	70.6%	0.1%	8.3%
01:37.667 - 01:40.667	50%	32.5%	0.2%	18.2%
01:40.667 - 01:43.669	45.8%	32.8%	1.3%	21.2%
01:43.669 - 01:46.667	81.1%	107.7%	0.6%	14.1%
01:46.667 - 01:49.668	76%	66.3%	0.7%	25.4%
01:49.668 - 01:52.666	76.5%	70.3%	0.6%	22.8%
01:52.666 - 01:55.667	39.4%	27.9%	0.7%	21.3%
01:55.667 - 01:58.667	39.5%	28.5%	0.7%	21.9%
01:58.667 - 02:01.667	38.5%	26.9%	0.7%	21.1%
02:01.667 - 02:04.667	21.2%	0.5%	0.5%	13.7%
02:04.667 - 02:07.667	51.7%	50%	3.8%	21.4%
02:07.667 - 02:10.667	62.9%	69.5%	0.5%	10.7%
02:10.667 - 02:13.680	58.3%	6.4%	0.7%	35.6%

Figura 6.9: Utilizzo CPU di Weendy.

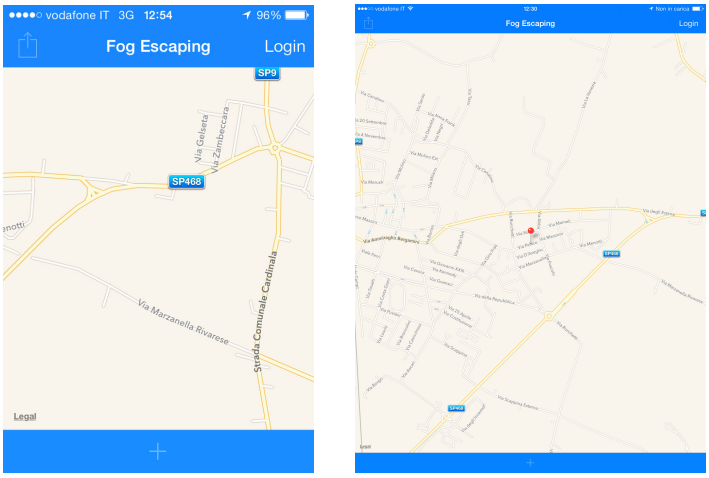
6.3 ScreenShot di Fog Escaping

In questa sezione, sono illustrate le varie fasi di utilizzo dell'applicazione *Fog Escaping*, tramite i device iPhone e iPad: Stato iniziale d'avvio, registrazione, login, segnalazione nebbia, ricerca percorso alternativo.

Figura 6.10: Icona di Fog Escaping.

6.3.1 Stato Iniziale

Al primo avvio dell'applicazione, sono disabilitate le funzionalità di segnalazione dello stato ambientale di nebbia e di ricerca di un percorso alternativo. L'Utente per abilitare le funzionalità dovrà obbligatoriamente utilizzare la funzionalità di login.

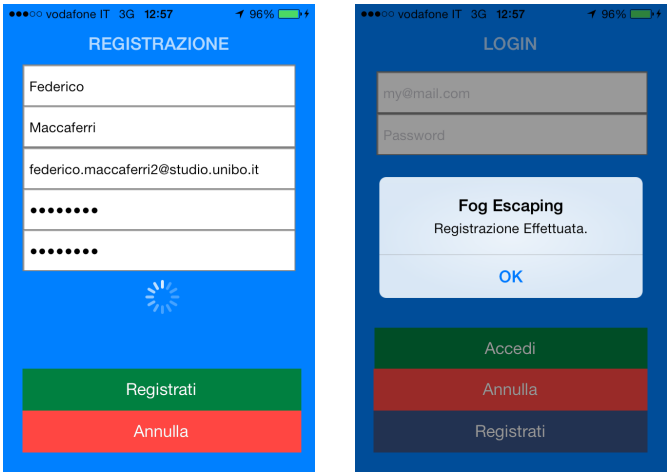


(a) Iphone (b) Ipad

Figura 6.11: Stato di primo avvio dell'applicazione *Fog Escaping*

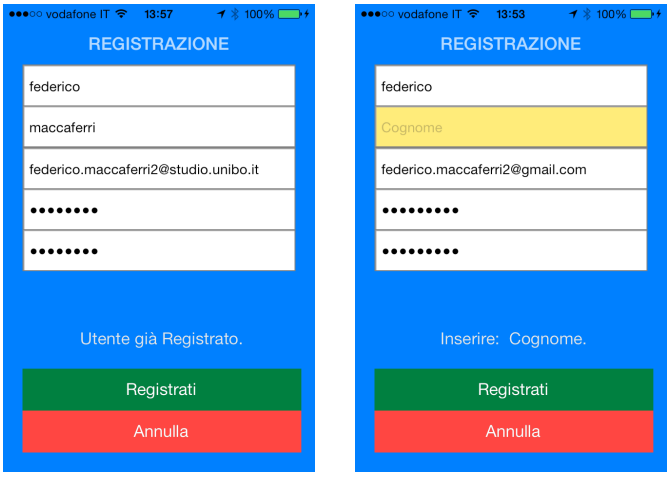
6.3.2 Registrazione

Se i dati inseriti sono corretti, l'Utente viene notificato della corretta registrazione tramite un alert di notifica (Fig. 6.12b). Se i dati inseriti sono errati, le label relative si colorano di giallo e nella parte bassa della View, compare l'indicazione di cosa deve essere inserito (Fig. 6.13b).



(a) Inserimento Dati. (b) Alert di Risposta.

Figura 6.12: Processo di Registrazione.



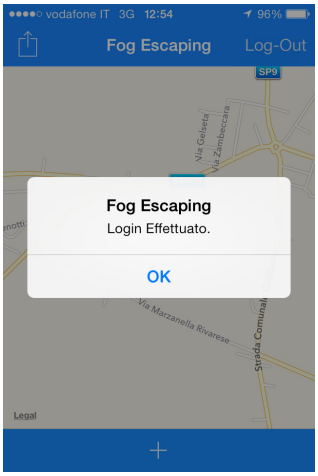
(a) Utente già presente. (b) Dati Mancanti.

Figura 6.13: Errore di Registrazione.

6.3.3 Login

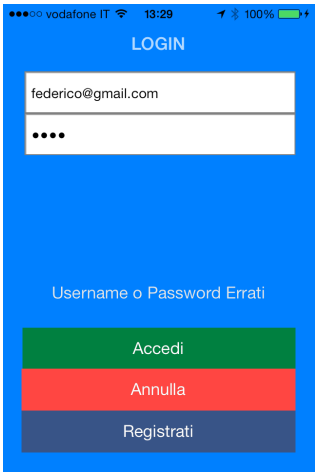
La View di Login permette l'inserimento di una e-mail e di una password. Se i dati inseriti sono corretti, la View viene deallocata e compare un alert di notifica (Fig. 6.14b). Se i dati inseriti non sono corretti, viene notificato il problema direttamente nella parte bassa della View (Fig. 6.15b).

(a) Inserimento Dati.

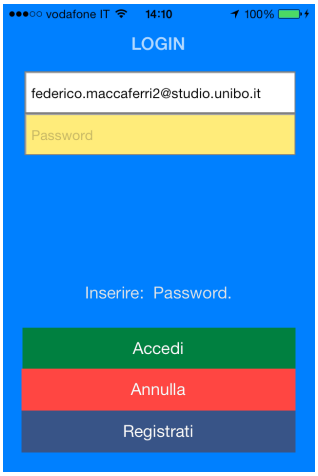


(b) Login Effettuato.

Figura 6.14: Processo di Login.



(a) Errore Dati.

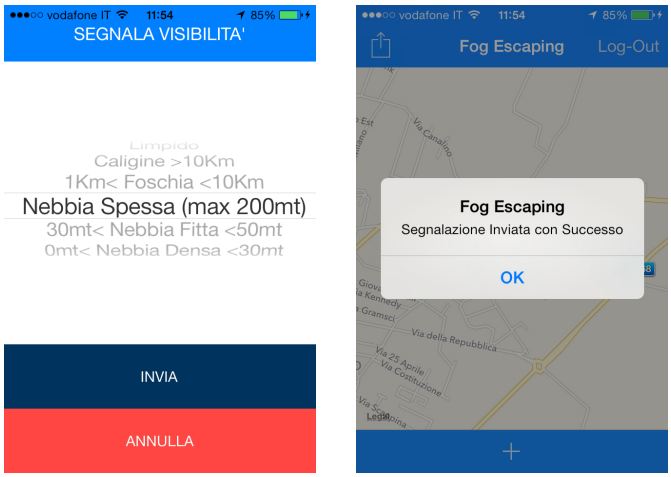


(b) Mancanza Dati.

Figura 6.15: Errore di Login.

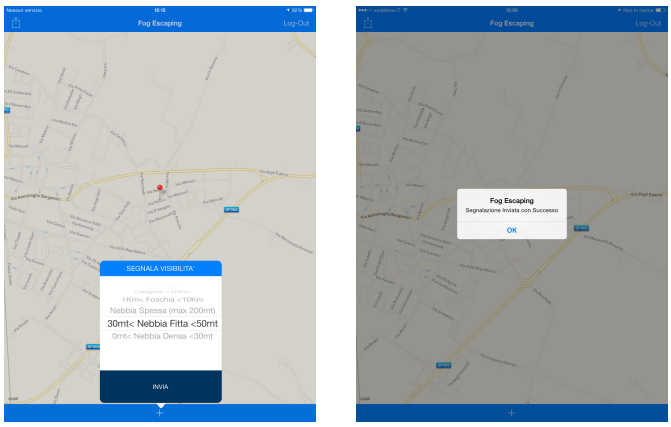
6.3.4 Segnalazione Nebbia

Una volta selezionato il valore di visibilità è possibile inviare la segnalazione di Nebbia premendo il pulsante Invia. La grafica nei dispositivi iPad utilizza una particolare tipologia di View chiamata *Popover* (Fig. 6.17a).



(a) (b)

Figura 6.16: Processo di Segnalazione tramite iPhone.



(a) (b)

Figura 6.17: Processo di Segnalazione tramite iPad.

6.3.5 Ricerca Percorso Alternativo

In questa View è possibile scegliere una posizione di partenza e arrivo. Al caricamento viene fatto il *geocoding* automatico sulla posizione attuale dell'Utente. Alla pressione del pulsante Escape inizia il processo di ricerca del percorso alternativo che si conclude con la visualizzazione del percorso e delle icone rappresentanti lo stato delle nebbie.

Figura 6.18: Ricerca Percorso Alternativo su iPhone.

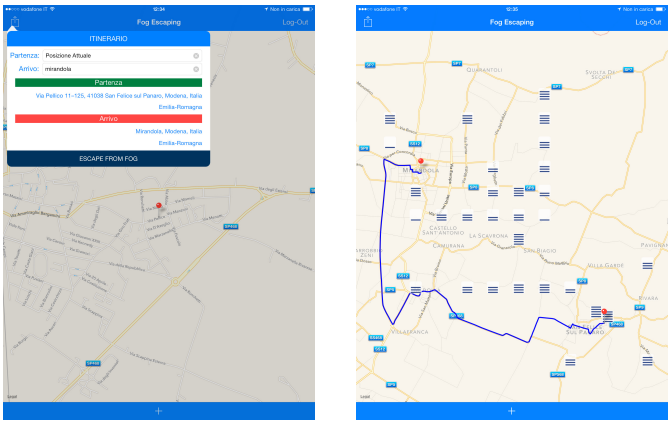


Figura 6.19: Ricerca Percorso Alternativo su iPad.

Conclusioni

Riepilogo

In questa Tesi di laurea, si è affrontato il problema della mobilità veicolare in caso di nebbie. Si è quindi sviluppato un prototipo con architettura Client-Server, che si è soffermato maggiormente sull'analisi dei dati per la creazione di un percorso alternativo. Si è preso in considerazione il sistema operativo mobile di Apple, iOS7 che rappresenta uno dei Sistemi Operativi mobili maggiormente presenti sul mercato oggigiorno e che possiede un buon bacino di utenze.
La parte Server è stata sviluppata secondo l'architettura REST; è presente un Server HTTP che riceve richieste e risponde in modo adeguato ai Client tramite lo scambio bidirezionale di dati in formato JSON. Nella parte Server è inclusa la base di dati: un componente molto importante poiché implementa al suo interno, parte della logica di Sistema tramite stored procedure.
La parte Client è un'applicazione per dispositivi iPad e iPhone chiamata *Fog Escaping*; essa è stata sviluppata secondo il pattern MVC (Model-View-Controller). *Fog Escaping* implementa un algoritmo *Greedy* di ricerca del percorso alternativo, che può essere utilizzato per diverse tipologie di applicazioni, che descriverò nella sezione sulle estensioni future (6.3.5).

Difficoltà emerse

Sono emerse svariate difficoltà al momento della realizzazione.
In primo luogo i servizi di Apple per la gestione delle mappe, possiedono alcuni piccoli "problemi" che è stato necessario arginare; infatti l'oggetto *MKDirections*, che rappresenta il percorso ottimale generato dai Server Apple, non è completamente accedibile tramite librerie ad alto livello.

All'inizio dello sviluppo era stata presa in considerazione la libreria di Google Maps, ma dopo attenta analisi, nessuno dei due sistemi (GoogleMap e iOS7) permetteva una navigazione attraverso una moltitudine di punti di soluzione. Si è quindi deciso di utilizzare il sistema più stabile fra i due; quindi la scelta è ricaduta sulle mappe integrate nel sistema iOS7.

Lo sviluppo della grafica, è iniziato su dispositivo iPad; gli strumenti messi a disposizione da *XCode 5*, hanno permesso in tempi brevi, il *porting* su iPhone.

La parte più critica del progetto, è stata sicuramente la creazione di un algoritmo *greedy*, che permettesse di evitare le nebbie. Prima della decisione finale, ovvero la creazione della Griglia dei *Punti di Controllo*, sono stati implementati svariati metodi e algoritmi, che però fallivano sempre in casi particolari. In seguito è stato necessario correggere alcuni problemi che sono stati generati dal metodo della Griglia, ma lavorando ininterrottamente, è stato possibile ottenere un buon risultato, sia in termini grafici sia in termini computazionali. L'implementazione della parte Server, non ha richiesto particolari sforzi, siccome durante i vari progetti del corso di studi, soprattutto dell'ultimo anno, erano già state sviluppate le tecnologie PHP e MySQL che sarebbero state rivisitate per il progetto di Tesi.

Implementazioni Future dell'Algoritmo

L'applicazione *Fog Escaping*, è nata per cercare un percorso alternativo in caso di nebbia, ma l'algoritmo per la ricerca di un percorso alternativo, può essere utilizzato per svariate tipologie di applicazioni.

Ad esempio date le recenti alluvioni nel territorio dell'Emilia-Romagna, si potrebbe creare un'applicazione, che fa visualizzare un elenco di tutti i ponti aperti e chiusi nella zona e creare un percorso alternativo che passi per i soli ponti aperti. Attualmente i cittadini devono rimanere in attesa di queste informazioni, quando disponibili, direttamente visionando il sito internet di ogni comune.

Un'altra idea per sfruttare l'algoritmo creato, è quella di utilizzare i dati ambientali d'inquinamento, per sensibilizzare il cittadino che è intenzionato a fare una passeggiata o un'attività sportiva, in determinati punti della città. Questi dati sono raccolti tutti i giorni, e verranno resi disponibili in formato GeoJSON da ARPA [35]. L'idea è di utilizzare l'algoritmo per consigliare percorsi che possano, in un qualche modo, far respirare aria migliore.

Estensioni Future

L'applicazione *Fog Escaping*, è stata sviluppata in modo da poter essere ampliata; in maniera specifica si sono analizzati alcuni contesti di espansione, che potrebbero essere sviluppati nell'immediato futuro.

Gestione dell'Affidabilità

Innanzitutto è necessario risolvere il problema dell'affidabilità dei dati; questo può essere fatto implementando un sistema simile a quello studiato nel Capitolo 5, ovvero l'analisi automatica della visibilità, tramite i sensori integrati nel dispositivo. In questo modo si otterrebbero dei valori di nebbia quantomeno affidabili e dipendenti dai sensori e non da un click sullo schermo. Purtroppo è risaputo che non basta far rilevare il dato ai sensori del dispositivo; è necessario un sistema di reputazione, che permetta agli altri utenti di confermare o meno lo stato di nebbia.

Raccolta e Reputazione

Per incoraggiare la raccolta dei dati di nebbia, nel momento in cui una persona porta a termine una segnalazione, sarà richiesto ai cittadini vicini se la condizione di nebbia è realmente esistente. In cambio della nuova segnalazione l'Utente guadagnerà punti di reputazione. Ovviamente chi segnala in modo "sbagliato", verrà punito con una diminuzione della reputazione. Il sistema quindi, dopo aver capito che esiste una condizione di nebbia, richiederà ad alcuni Utenti che sono man mano più distanti dalla prima segnalazione, i dati riguardanti la visibilità riscontrata. In questo modo si richiede la partecipazione di un campione di utenti limitrofi, eventualmente interrogando i client di quelli più affidabili.

Utilizzo dei Nodi Stradali

Per un algoritmo preciso, sarebbe necessario utilizzare una mappatura completa del grafo stradale. Questi dati sono disponibili dalle librerie di *OpenStreetMap*, richiedendo i nodi che sono all'interno di un quadrato, formato da due latitudini e due longitudini [36]. In questo modo i *Punti di Controllo* sarebbero esattamente parte della strada. In seguito basterebbe inserire i dati ambientali di visibilità e utilizzare l'algoritmo di Dijskra così com'è stato pensato.

Epilogo

Secondo i sostenitori del movimento Open data, i dati andrebbero trattati come *beni comuni* [37].

La cooperazione è la parola "chiave" che permette l'evolversi della comunicazione in modo rapido. Le tecnologie ci aiutano e ci assistono, ma questo accade solo se queste tecnologie sono programmate per farlo. È inoltre necessario che i dati raccolti siano divulgati; così facendo chiunque potrà fare la sua parte per risolvere i piccoli e i grandi problemi di ogni giorno.

Bibliografia

[1] Andrew Campbell and Tanzeem Choudhury. From smart to cognitive phones. *IEEE CS*, 2012.

[2] Ye Xu, Mu Lin, Hong Lu, Giuseppe Cardone, Nicholas D. Lane, Zhenyu Chen, Andrew Campbell, and Tanzeem Choudhury. Preference, context and communities: A multi-faceted approach to predicting smartphone app usage patterns. *Dartmouth College, Intel Labs, University of Bologna, Microsoft Research Asia, Cornell University*, 2013.

[3] Andrew T. Campbell, Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, and Ronald A. Peterson. People-centric urban sensing. *Computer Science, Dartmouth College Computer Science, Electrical Engineering, Columbia University*, 2006.

[4] J. Burke, D. Estring, M. Hansen, A Parker, N. Ramanathan, S. Reddy, and M.B. Srivastava. People-centric urban sensing. *Proceedings of Second Annual International Wirless Internet Conference (WICON)*, pages 2–5, August 2006.

[5] R. Rana, C.T. Chou, S. Kanhere, N. Bulusu, and W. Hu. Ear-phone: An end-to- end participatory urban noise mapping system. *Proceedings of ACM/IEEE IPSN, Stockholm, Sweden*, April 2010.

[6] E. Miluzzo, N. Lane, K. Fodor, R. Peterson, S. Eisenman, H. Lu, M. Musolesi, X. Zheng, and A. Campbell. Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme ap-

plication. *Proceedings of ACM SenSys, Raleigh, NC, USA*, November 2009.

[7] N. Kapadia, A.and Triandopoulos, C. Cornelius, D. Peebles, and D. Kotz. Anonysense: Opportunistic and privacy-preserving context collection. *Indulska, J., Pat- terson, D.J., Rodden, T., Ott, M. (eds.) PERVASIVE 2008*, 2008.

[8] K. Huang, S.S. Kanhere, and W. Hu. Are you contributing trustworthy data? the case for a reputation framework in participatory sensing. *Proceedings of ACM MSWiM, Bodrum, Turkey*, October 2010.

[9] S. Gaonkar, J. Li, and R.R. Choudhury. Micro-blog: Sharing and querying content through mobile phones and social participation. *Proceedings of ACM Mo- biSys, Breckenridge, CO, USA*, June 2008.

[10] Robin Kravets, Hilfi Alkaff, Andrew Campbell, Karrie Karahalios, and Klara Nahrstedt. Crowdwatch: Enabling in-network crowd-sourcing. *University of Illinois and Dartmouth College*, 2013.

[11] I. Saleemi, K. Shafique, , and M. Shah. Probabilistic modeling of scene dynamics for applications in visual surveillance. *Proc. of IEEE Transactions on Patterns Recognition and Machine Intelligence*, 2009.

[12] U. Crowdsourcing. Proceedings of 2nd workshop on ubiquitous crowdsourcing, 2011. *http://www.personal.psu.edu/u1o/crowdsourcing/program.html.*, 2011.

[13] S. Bisker and F. Casalegno. Digital volunteerism during disaster: Crowdsourcing information processing.
http://metamanda.com/crowdcomputing/subs/bisker.pdf, 2011.

[14] D. G. Murray, E. Yoneki, J. Crowcroft, and S. Hand. The case for crowd computing. *ACM MobiHeld?10.*

[15] K. Starbird. Digital volunteerism during disaster: Crowdsourcing information processing. *ACM CHI?2011 Workshop*, 2011.

[16] S. Reddy, A. Parker, J. Hyman, J. Burke, D. Estin, and M. Hansen. Image brows- ing, processing and clustering for participatory sensing: Lessons from a dietsense prototype. *Proceedings of the Workshop on Embedded Networked Sensors (Em- NetS), Cork, Ireland*, June 2007.

[17] bewell. *http://www.bewellapp.org.*

[18] Zhenyu Chen, Mu Lin, Fanglin Chen, Nicholas D. Lane, Giuseppe Cardone, Rui Wang, Tianxing Li, Yiqiang Chen, Tanzeem Choudhury, and Andrew T. Campbell. Unobtrusive sleep monitoring using smartphones. *Dartmouth College, Chinese Academy of Sciences, Microsoft Research Asia, Cornell University,University of Bologna, Beijing Key Laboratory of Mobile Computing and Pervasive Device*, 2013.

[19] jawbone up. *https://jawbone.com/up.*

[20] M. Mun, S. Reddy, and et al. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. *Proceedings of ACM MobiSys, Krakow, Poland*, June 2009.

[21] S. Eisenman, E. Miluzzo, N. Lane, R. Peterson, G. Ahn, and A. Campbell. The bikenet mobile sensing system for cyclist experience mapping. *Proceedings of ACM SenSys, Sydney, Australia*, November 2007.

[22] Runtastic. *https://www.runtastic.com/it.*

[23] Runkeeper. *http://runkeeper.com/.*

[24] Everytrail. *http://it.everytrail.com/.*

[25] Y. Dong, S.S. Kanhere, C.T. Chou, and N.: Bulusu. Automatic collection of fuel prices from a network of mobile cameras. *Proceedings of IEEE DCOSS 2008, Santorini, Greece*, June 2008.

[26] Plugshare. *http://www.plugshare.com/.*

[27] J. Carrapetta, N. Youdale, A. Chow, and V. Sivaraman. Haze watch project. *http://www.pollution.ee.unsw.edu.au.*

[28] Air quality egg community. *http://airqualityegg.com/.*

[29] Weendy. *http://weendy.com/.*

[30] P. Mohan, V. Padmanabhan, and R. Ramjee. Nericell: Rich monitoring of road and traffic conditions using mobile smartphones. *Proceedings of ACM SenSys, Raleigh, NC, USA*, November 2008.

[31] Inrix traffic. *http://www.inrixtraffic.com/.*

[32] Opensignal. *http://opensignal.com/.*

[33] Waze. *https://www.waze.com/it/.*

[34] E. W. Dijskra. A note on two problems in connexion with graphs. *Numerishe Mathematik 1*, pages 269–271, 1959.

[35] L'acceeso ai dati di arpa emilia-romagna. *http://forges.forumpa.it/assets/Speeches/9932/co_29_cattani.pdf.*

[36] Openstreetmap node request. *http://wiki.openstreetmap.org/wiki/API_v0.6.*

[37] Open data. *http://it.wikipedia.org/wiki/Dati_aperti#Diritti_fondamentali.*

Ringraziamenti

In ordine completamente casuale vorrei ringraziare:

Il Professor Marco Di Felice per l'aiuto nello svolgimento della tesi.
Graziano e Daniela per l'immensa pazienza, consigli e il supporto morale. Gaia per avermi supportato e per avermi dato la spinta necessaria per ripartire. Iria e Ezio per avermi aiutato nel momento del bisogno. Ermes e Franca per le cene irripetibili. Enrico e David per tutto il tempo passato assieme. Richy D. per le tante risate e Eleonora per gli interminabili giri in roller.
Ilaria per la pazienza con David e Alice per la pazienza con Enrico.

Buch per la creatività e le passioni condivise. Alessia per la pazienza e la capacità di giudizio. Gabri per la dzattera, i flip e il futuro motore a reazione. Garro per la frase : "Gli amici prima di tutto". Richy B. per le sue doti di comunicazione. Nello per la forza d'animo. Chiara per la sua competenza e il suo aiuto. Casti e Frà per il magnifico accompagnamento. Cesare per aver addolcito molte giornate amare. Sara per la sua aura di felicità. Corra per avermi spiegato come funziona l'essere umano. Giorgio per le spiegazioni di analisi. Biondo e Pier per la compagnia durante gli anni migliori.

Chris per l'appoggio nelle idee e nello studio. Jack che con il suo modo di fare è capace di farti sorridere sempre. Marika per la sua pazza visione della vita. Steve per le sue perle e comete. Vale per l'attitudine da killer. Marti per la Giulietta incorporata. Sonia per la pazienza nel sopportare i compagni.

Luca per la passione nei fagioli. Marco per avermi fatto compagnia in treno e per i magnifici fumetti. Maggy per il sostegno nei secoli. Silvia per la sua passione per la natura. Pizzi, Samini, Piwi per gli interminabili discorsi. Tolsimir, Erundil, Kennerd, Luthien, Astarte, Bolobo, Beyron per i viaggi meravigliosi. Il Pantegana per i momenti magnifici assieme. L'Alchimista per il controllo. Pecorè per la perseveranza. Gracca per l'amicizia e la sua sapienza musicale. War Machine per la preparazione della battaglia contro il BOSS finale. Griglia, Pinza, Cunsa, Fuoco per il gustosissimo lavoro.
Dasy per i tanti brindisi.

Anna per la forza d'animo e per avermi spronato.

Tutti i professori per le sfide.
Tutte le persone che hanno allietato i viaggi in treno.
Tutti i personaggi che sognando regalano nuove speranze.
Tutti gli altri amici che mi hanno sostenuto in questo percorso.