Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

UNA PIATTAFORMA COLLABORATIVA DI URBAN-SENSING BASATA SU DISPOSITIVI ANDROID

Tesi di Laurea in Basi di dati

Relatore:
Chiar.mo Prof.
MARCO DI FELICE

Presentata da:
FILIPPO TODESCHINI

Sessione II
Anno Accademico 2012/2013

Introduzione

Nel corso degli ultimi anni, si è assistito ad un forte e sempre più crescente sviluppo di dispositivi mobili, come smartphone e tablet, che presentano caratteristiche software e hardware sempre più complesse. A fronte di ciò si sono aperte nuove frontiere e nuove possibilità di creare servizi sia per i singoli utenti che per la comunità intera. In particolare, questa tesi di laurea tratta dello sviluppo di un progetto volto alla realizzazione di una piattaforma collaborativa di urban-sensing, basata sui dispositivi Android. La piattaforma consiste in un sistema client-server in cui il client è rappresentato da un applicazione, esclusivamente per dispositivi Android, che raccoglie i dati dai sensori installati sul dispositivo e li invia ad un server; quest'ultimo, una volta ricevuti i dati, li salva e colleziona su un database interno.

L'idea di fondo sulla quale è stato sviluppato il progetto e la tesi è quella di creare un'applicazione e un sistema informativo nel quale vengono salvati dati relativi ai dispositivi, in modo tale da poter collezionare una gran quantità di dati e dai quali poter ricavare dei servizi utili, sia al singolo utente, che all'intera comunità.

I concetti base su cui si fonda la tesi sono urban-sensing e crowdsourcing. Il primo consiste nell'adottare una visione centralizzata dell'utente, cioè nell'adottare una metodologia in cui gli attributi delle persone, i dintorni immediati della gente, e il modo di interpretare e interagire con l'ambiente circostante diventano importanti ed essenziali nella raccolta dei dati e delle informazioni. Invece, il secondo concetto rappresenta una particolare attività nella quale vengono coinvolti un gruppo di persone per la realizzazione

di un particolare compito. In questo contesto è richiesto, a tutti gli utenti in possesso dell'applicazione del progetto, in maniera assolutamente libera e volontaria, di raccogliere dati ed informazioni, relativi ai sensori del proprio dispositivo e alle interazioni del dispositivo con l'ambiente urbano circostante; l'obiettivo di creare un database ricco di informazioni sia relative alle caratteristiche dei singoli dispositivi, sia relative all'ambiente urbano che ogni utente frequenta e sia alle principali attività e interazioni dell'utente con l'ambiente esterno.

Nei prossimi capitoli verranno fornite ulteriori informazioni riguardanti i concetti di fondo della tesi, dell'applicazione, della sua architettura ed implementazione. In praticolare nel capitolo 1 verranno analizzati i concetti di Crowdsourcing, Urban-Sensing e verrà data una visione generale dei dati che si possono recuperare dai dispositivi android. Nel capitolo 2 verrà analizzata l'architettura e le funzionalità del sistema sviluppato, mentre nel capitolo 3 verrà analizzata l'implementazione prendendo in considerazione alcuni estratti di codice significativi ai fini dell'analisi.

Figura 1: Database & Android

Indice

Introduzione **i**

1 Stato dell'arte **1**

1.1 Crowdsourcing 1

1.1.1 Definizione 1

1.1.2 Vantaggi e svantaggi 3

1.1.3 Evoluzione 4

1.2 Urban-Sensing 5

1.3 Esempi di applicazioni 7

1.3.1 People-Centric Sensing Application 7

1.3.2 Enviroment-Centric Sensing Application 8

1.4 Crowdsourcing coi sensori dei dispositivi Android 10

1.4.1 SensorManager 11

1.4.2 ConnectivityManager 17

1.4.3 LocationManager 18

1.4.4 Altri sensori 18

2 Architettura **19**

2.1 Client 19

2.1.1 Obiettivi, funzionalità principali e requisiti 19

2.1.2 Struttura 20

2.2 Server 22

2.2.1 Obiettivi, funzionalità principali e requisiti di sistema . 22

2.2.2 Struttura 22

3 Implementazione **41**
3.1 Client 41
3.1.1 SensorActivity.java 41
3.1.2 SensorExpandableAdapter.java 44
3.1.3 SensorDetails.java 45
3.1.4 JSONSender.java 47
3.2 Server 51
3.2.1 GestioneDB.php 51
3.2.2 installDB.php 53
3.2.3 sensorMonitorServer.php 54

4 Analisi delle prestazioni **55**

Conclusioni **57**

A Query per la creazione del database **59**

B Casi implementati sul server **71**

Bibliografia **81**

Elenco delle figure

1 Database & Android ii

1.1 Crowdsourcing 2
1.2 Web crowdsourcing 4
1.3 People centric view 5
1.4 BikeNet 8
1.5 HazeWatch 9
1.6 Mobile Sensor 10
1.7 Connectivity 17
1.8 Mobile Sensor 18

2.1 Android 20
2.2 SensorActivity 21
2.3 SensorDetails 21
2.4 Google Server 22
2.5 Diagramma Entità-Relazione 23

3.1 Sessione attiva 42
3.2 Sessione chiusa 42
3.3 Preferenze 43
3.4 Sessione attiva 44
3.5 Sessione chiusa 44
3.6 Sensor 46
3.7 LocationProvider 46
3.8 NetworkInfo 46

3.9 Bluetooth 46
3.10 Sessione attiva 47
3.11 Sessione chiusa 47
3.12 JSON batteria 50
3.13 JSON accelerometro 50
3.14 HTTPRequest 50
3.15 Server 51
3.16 Database 52
3.17 Database2 53
3.18 Server Interaction 54

4.1 Prestazioni base 56
4.2 Prestazioni SensorManager, batteria e microfono 56
4.3 Prestazioni delle network 56
4.4 Prestazioni dei LocationProvider 56
4.5 World 58

Elenco delle tabelle

1.1 Sensor types supported by the Android platform.[6] 11
1.2 Motion sensors that are supported on the Android platform.[7] 14
1.3 Environment sensors that are supported on the Android platform.[9] 16
1.4 Position sensors that are supported on the Android platform.[8] 16

2.1 Diagramma Logico 24
2.2 Dizionario delle Entità 30
2.3 Dizionario delle Relazioni 39

3.1 Metodi di JSONSender.java 48

A.1 Query per la creazione delle tabelle 59

Capitolo 1

Stato dell'arte

1.1 Crowdsourcing

1.1.1 Definizione

Il crowdsourcing si riferisce ad un modello di problem-solving in cui la risoluzione del lavoro, del problema è distribuito ad una folla di utenti di dimensione indefinita. I partecipanti sono scelti secondo il meccanismo delle open call, ovvero un sistema secondo il quale gli utenti che vogliono partecipare al progetto rispondono, in totale libertà, ad un invito divulgato, nella maggior parte dei casi, attraverso il web. Il termine crowdsourcing è stato coniato ed utilizzato per la prima volta da Jeff Howe nell'articolo scritto nel 2006 per la rivista Wired intitolato The Rise of Croudsourcing definendolo:

> Simply defined, crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively), but is also often undertaken by sole individuals. The crucial prerequisite is the use of the open call format and the large network of potential laborers. [10]

In questo articolo, Howe scrive come le potenzialità del crowdsourcing permetteranno di risolvere problemi complessi, di svolgere e di distribuire il carico di lavoro di determinate attività e di contribuire con idee sempre più originali allo sviluppo economico, tecnico e tecnologico del prossimo futuro. Ciò lo si è già visto negli ultimi anni con il forte sviluppo tecnologico che ha portato, innanzitutto, ad una dimunizione dei costi dell'hardware, poi ad una maggiore divulgazione della conoscenza e nello sviluppo del software riducendo notevolmente la distinzione tra professionista e amatore del settore.

Figura 1.1: Crowdsourcing

Le persone che partecipano alle attività di crowdsourcing richieste vengono contattate e scelte, solitamente, con il metodo delle open-call; gli utenti che vedono l'invito possono scegliere, in maniera volontaria ed autonoma se aderire o meno al progetto proposto. Una volta che questi hanno scelto di partecipare, di solito, si riuniscono in comunità online nelle quali possono recuperare i dati e le specifiche dell'attività richiesta, del problema da risolvere o gli spunti dai quali partire per proporre alcune idee. Inoltre, si possono raccogliere i dati richiesti, elaborare le proposte, le idee e condividerle con gli altri partecipanti e chi ha organizzato, richiesto l'attività. Oltretutto in

queste comunità, gli utenti si confrontano l'un l'altro sui risultati ottenuti e magari migliorare la propria partecipazione. Infine queste comunità sono sempre costituite da una vasta gamma di utenti di qualsiasi tipologia: esperti, inesperti, professionisti e amatori del settore. Ciò garantisce ai progetti e a chi li propone una vasta gamma di potenzialità derivate dalle conoscenze, dalla voglia di partecipare e dalle abilità specifiche dei singoli utenti.

1.1.2 Vantaggi e svantaggi

Il crowdsourcing porta a dei benefici, sia per chi propone l'attività sia alla comunità, al gruppo di utenti che usufruiranno degli eventuali servizi. Per primo, si ha che le attività, i problemi vengono risolti ad un costo relativamente basso e in tempi abbastanza brevi. Secondo, di solito non è previsto un pagamento, o se previsto è definito sulla base dei risultati ottenuti, quindi i partecipanti del progetto sono stimolati esclusivamente dalla propria voglia di mettersi in gioco e dal proprio desiderio di ricerca personale, non da un eventuale premio in denaro. Terzo, chi propone l'attività si può relazionare con un numero di esperti superiore a quelli che già conosce e frequenta. Quarto, si possono conoscere direttamente i desideri, le richieste e le necessità degli utilizzatori. Quinto, la possibilità di distribuire meglio il lavoro, sia per quanto riguarda la quantità, sia per quanto riguarda la complessità e le zone in cui andrebbe svolto. Infine, nel caso specifico di una azienda, richiedendo il contributo diretto agli utenti si viene a sviluppare un senso di appartenenza che li fidelizza all'azienda.

Tuttavia, il crowdsourcing presenta anche delle difficoltà e degli svantaggi. In primo luogo, è si una ottima fonte di conoscenza e un ottimo modello per captare le richieste e le idee degli utenti a livello globale, ma non sempre riesce a garantire la qualità dei risultati ottenuti e quindi sono necessari dei costi aggiuntivi per poter selezionare i dati raggiungendo dei risultati soddisfacenti. Secondo, non essendo a conoscenza di quanti risponderanno all'invito di partecipazione, la probabilità del fallimento del progetto è maggiore se c'è una mancanza di risorse, utenti, qualità dei risultati su larga scala. Terzo,

la mancanza di un incentivo economico potrebbe portare a raccogliere meno interesse negli utenti. Quarto, non c'è alcun accordo scritto tra chi propone l'attività e chi collabora, si denota c'è una difficoltà nel mantenere i rapporti tra i collaboratori. Infine, ci potrebbero essere dei collaboratori che volutamente hanno deciso di partecipare al progetto, e producendo soluzioni errate cerchino di causare danno.

1.1.3 Evoluzione

Il crowdsourcing, nato come concetto nel 2006, si è sviluppato prevalentemente sul web. Infatti, le molteplici tecniche, applicazioni e potenzialità di Internet permettevano e tuttora permettono la diffusione veloce delle richieste, semplificano la condivisione dei risultati, facilitano gli utenti ad essere più aperti in quanto non si sentono fisicamente giudicati, infine, permettono una maggiore focalizzazione sul progetto e non sulle persone che sono coinvolte. Tuttavia negli ultimi anni, con lo sviluppo tecnologico sia del web, ma sopratutto dei dispositivi mobili si sono venute a creare nuove potenzialità per quanto riguarda il crowdsourcing; ora, grazie alle applicazioni sviluppate per i dispositivi mobili, c'è la possibilità di sfruttare anche i dati forniti da essi. Da questi dati, si ha la possibilità di conoscere e capire sempre meglio la realtà in cui vivono gli utenti, il contesto urbano in cui sono immersi, come si relazionano ed interagiscono con esso e come potrebbe essere migliorato.

Figura 1.2: Web crowdsourcing

1.2 Urban-Sensing

A fronte delle nuove tecnologie, è necessario utilizzare un approccio diverso da quello tradizionale, in cui la ricerca degli aspetti, delle caratteristiche della realtà, e in particolare dell'ambiente urbano che sta attorno alle persone non va eseguita solo con l'analisi dei dati recuperati dai sensori già presenti sul territorio e dalle informazioni a disposizione negli archivi, ma va effettuata a partire dalle singole persone. Con lo sviluppo delle tecnologie mobili è possibile monitorare lo stato di ogni realtà a sfruttando i dati ricavati dai vari dispositivi.

Figura 1.3: People centric view

Il nuovo approccio si basa su una visione più centrale dei singoli individui, poichè i dati ricavati dai dispositivi mobili, descrivono il contesto urbano, sia nelle sue caratteristiche fisiche sia nelle sue interazioni con l'individuo, vissuto da ogni utente. I singoli individui non sono più degli agenti passivi che subiscono i risultati delle rilevazioni e delle interpretazioni dei dati ottenuti senza, o almeno in parte, essere stati interrogati, ma sono agenti attivi che, partecipando al monitoraggio della realtà, inviando i dati dei propri di-

spositivi che raccontano la quotidianità di ognuno, permettono di migliorare l'interpretazione del contesto urbano vissuto.

L'idea di fondo di questo nuovo approccio è di responsabilizzare i cittadini a raccogliere informazioni da condividere dal proprio ambiente, dal contesto urbano che ciascuno vive. I dispositivi mobili permettono di fare ciò, in particolare con l'analisi dei dati ricavati dai sensori a disposizione è possibile realizzare una buona approssimazione della realtà monitorata. Tuttavia, tutto ciò è utile e realistico se, a monitorare l'ambiente urbano ci siano un buon numero di utenti.

I dati recuperati verrebbero raccolti e analizzati con delle attività di crowdsourcing dove, gli utenti, che hanno deciso liberamente di partecipare, raccolgono i dati e li inviano ad uno o più server che li memorizzano in un database e dai quali è possibile ricavare delle applicazioni, dei servizi che descrivono il contesto urbano e possono rendere più facile la vita quotidiana dei cittadini. La città diventa, quindi, una realtà *socio-technical*[3] nella quale ogni situazione viene monitorata e analizzata.

Questa nuova concezione ha dei vantaggi, infatti permette un'analisi più precisa dei contesti urbani, permette di capire meglio le interazioni dei cittadini con la città ed è più facile di percepire ed interrogare i singoli cittadini sulle loro necessità quotidiane. Un altro vantaggio è dato dal fatto che, il costo computazionale di tali operazioni è distribuito tra più dispositivi.

Tuttavia, presenta alcuni svantaggi e alcune questioni tecniche che ne limitano ancora l'utilizzo. Prima di tutto, c'è la difficoltà di coinvolgere la gran parte dei cittadini, rassicurandoli e costruendo dei meccanismi che mantengano al sicuro la privacy degli utenti. Un altro aspetto tecnico da considerare, è nel costruire un applicazione, un sistema che consumi poca energia; se il sistema creato consumasse molta energia, nei dispositivi mobili, gli utenti sarebbero poco volenterosi nell'utilizzarla perchè gli priverebbe della possibilità di accedere ai loro servizi preferiti. Invece, uno degli svantaggi che si riscontra sta nella non completezza dei dati o nella loro imprecisione che potrebbe influire nel contesto complessivo delle analisi.

1.3 Esempi di applicazioni

Fino ad ora è stato trattato come si sono sviluppate, negli ultimi anni, nuove tecnologie, nuove concezioni e nuovi atteggiamenti volti a migliorare la rappresentazione e la conoscenza della realtà. A fronte di ciò sono create delle applicazioni che, partendo dall'analisi dei dati singoli utenti, offrono dei servizi utili sia alle singole persone, che alla comunità intera.

1.3.1 People-Centric Sensing Application

Una prima serie di applicazioni sviluppate pongono come strumento di analisi la persona.

Ad esempio, *DietSense*[11] è applicazione che monitora lo stato di salute di una persona, in particolare viene usato da chi vuole perdere peso, chi sta seguendo una dieta per documentare le sue attività con immagini e suoni. Il dispositivo viene disposto sul collo dell'utente e, in maniera automatica, fotografa i piatti che gli vengono posti davanti e registra ora, luogo e suoni di quel momento in quel preciso contesto. Le immagini permettono ai partecipanti di vedere cosa e quanto hanno mangiato, in modo da potersi regolare per i pasti futuri e per mantenere un diario della dieta da mostrare, se necessario, al dottore.

Un altro esempio è *PEIR(Personal Enviromental Impact)*[12], è un sistema che permette di determinare l'esposizione degli utenti all'inquinamento ambientale. L'applicazione invia le coordinate e alcuni dati dell'utente al sistema con cui si interfaccia. Questo, recuperando i dati dai sensori ambientali presenti e stimando l'inquinamento prodotto dall'utente, ritorna i valori dell'inquinamento stimati in quella zona in quel preciso momento. Inoltre, ci sono anche applicazioni che possono monitorare il moto degli utenti. In particolare, è bene ricordare *BikeNet*[13] che permette di descrivere l'attività ciclistica dei vari utenti. Questa applicazione misura la posizione, la velocità e le calorie bruciate attraverso il monitoraggio e l'analisi dei dati ricavati

dai sensori del dispositivo e anche da alcuni sensori esterni; i dati vengono condivisi nel web e sono a disposizione di tutti i ciclisti della comunità.

Figura 1.4: Bikenet

Un'altra applicazione simile, è *WAID*[4] che permette d'impostare, a seconda del moto dell'utente, un determinato profilo al telefono. L'applicazione monitora i principali sensori di movimento e, attraverso un algoritmo che riconosce la tipologia del moto dell'utente imposta un profilo precedentemente creato ed assegnato.

Infine, *PetrolWatch*[14], che permette di condividere i prezzi dei distributori di carburante. I partecipanti utilizzano le loro fotocamere per fotografare automaticamente, ogni qual volta ci passano vicino, il prezzo del carburante di un distributore. Le immagini vengono inviate e catalogate anche con delle informazioni relative all'orario e alla posizione dell'utente nel momento in cui sono state scattate. Le informazioni dei prezzi vengono poi condivise sul web dove gli utenti possono vederle e ricercare i distributori con i prezzi inferiori.

1.3.2 Enviroment-Centric Sensing Application

Una seconda serie di applicazioni, invece, prende come oggetto d'analisi l'interazione degli utenti con l'ambiente esterno. A differenza delle precedenti

applicazioni, i dati vengono raccolti ed analizzati non a livello dei singoli utenti, ma a livello della comunità dei partecipanti, interagendo in particolar modo con i sensori esterni.

Un primo esempio è *Haze Watch*[15], dove i dispositivi mobili si interfacciano con i sensori esterni che misurano l'inquinamento atmosferico con lo scopo di ricevere le informazioni da essi. I dispositivi mobili possono arricchire questi dati con le loro misurazioni, anche se queste risultano meno precise, poichè possono monitorare situazioni lontane dai sensori fissi o situazioni accidentali. Tutte queste misurazioni vengono catalogate e salvate su un server centrale; queste possono essere visualizzate da tutti, non solo dai partecipanti.

Figura 1.5: HazeWatch

Invece, un secondo esempio può essere *EarPhone*[16], un'applicazione che misura il livello di inquinamento acustico. I microfoni dei dispositivi mobili possono essere utilizzati per misurare il volume dell'ambiente circostante, successivamente questi dati vengono inviati su un server centrale e condivisi;

analizzando i dati è possibile notare dove gli utenti sono esposti ad un maggiore inquinamento acustico e si possono cercare delle soluzioni per migliorare la salute di tutti.

Infine, ci sono applicazioni che aiutano i cittadini nella mobilità. In particolare, esistono delle applicazioni, come *Nericel*[17], che permettono di visualizzare in tempo reale lo stato del traffico lungo le strade. L'applicazione invia i dati dei sensori dei dispositivi, attraverso i quali riesce a monitorare lo stato del traffico della strada attraversata, ad un server che li colleziona e riesce a mostrare a tutti, in tempo reale, lo stato del traffico delle strade della città.

1.4 Crowdsourcing coi sensori dei dispositivi Android

Negli ultimi anni si è visto un crescente sviluppo di dispositivi mobili. Come abbiamo già detto, questi attraverso i loro sensori ci permettono di descrivere al meglio la realtà che circonda ogni singolo utilizzatore. A seconda del dispositivo troveremo un certo numero di sensori installati e a volte anche di marche diverse. In particolare in questo progetto sono stati analizzati i dispositivi mobili Android.

Figura 1.6: Mobile Sensor

1.4.1 SensorManager

La classe SensorManager permette di accedere ai sensori del dispositivo; più precisamente ci garantisce l'accesso ai dati, alle informazioni dei sensori di movimento, ambientali e di posizione installati sul dispositivo. Nella tabella seguente c'è una panoramica dei sensori supportati in Android.

Tabella 1.1: Sensor types supported by the Android platform.[6]

Sensors	**Type**	**Description**	**Commons Usage**
ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
AMBIENT TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (Â°C). See note below.	Monitoring air temperatures.
GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).

Continued on next page

Tabella 1.1 – *Continued from previous page*

Sensors	**Type**	**Description**	**Commons Usage**
GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.)
LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
LINEAR ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
MAGNETIC FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in $\hat{I}\frac{1}{4}T$.	Creating a compass
ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z).	Determining device position.
PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.

Continued on next page

Tabella 1.1 – *Continued from previous page*

Sensors	**Type**	**Description**	**Commons Usage**
PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
RELATIVE HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
ROTATION VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (Â°C).	Monitoring temperatures.

Di tutti questi sensori, implementando i giusti servizi e metodi, è possibile ricavare le caratteristiche fisiche e i valori che assumono nel corso del tempo. Come si può notare, i sensori elencati sono classificabili in tre categorie: i sensori di movimento, i sensori ambientali e i sensori di posizioni

Sensori di movimento

I sensori di movimento sono l'accelerometro, il sensore di gravità, il giroscopio e quello che misura a rotazione che misurano le forze di accelerazione e rotazione lungo i tre assi.

Tabella 1.2: Motion sensors that are supported on the Android platform.[7]

Sensors	**Sensor event data**	**Description**	**Units**
ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s^2
	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	m/s^2
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	m/s^2
GRAVITY	SensorEvent.values[0]	Force of gravity along the x axis.	m/s^2
	SensorEvent.values[1]	Force of gravity along the y axis.	m/s^2
	SensorEvent.values[2]	Force of gravity along the z axis.	m/s^2
GYROSCOPE	SensorEvent.values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation around the y axis.	rad/s
	SensorEvent.values[2]	Rate of rotation around the z axis.	rad/s

Continued on next page

Tabella 1.2 – *Continued from previous page*

Sensors	**Sensor event data**	**Description**	**Units**
LINEAR ACCELERATION	SensorEvent.values[0]	Acceleration force along the x axis (excluding gravity).	m/s^2
	SensorEvent.values[1]	Acceleration force along the y axis (excluding gravity).	m/s^2
	SensorEvent.values[2]	Acceleration force along the z axis (excluding gravity).	m/s^2
ROTATION VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis (x * sin(θ/2)).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis (y * sin(θ/2)).	Unitless
	SensorEvent.values[2]	Rotation vector component along the z axis (z * sin(θ/2)).	Unitless
	SensorEvent.values[3]	Scalar component of the rotation vector ((cos(θ/2)).	Unitless

Sensori ambientali

I sensori ambientali sono quelli che misurano alcuni parametri ambientali, quali la temperatura ambiente e pressione dell'aria, illuminazione, e umidità. Questa categoria comprende i barometri, fotometri, e termometri.

Tabella 1.3: Environment sensors that are supported on the Android platform.[9]

Sensors	**Sensor event data**	**Units**	**Data Description**
AMBIENT TEMPERATURE	event.values[0]	^{o}C	Ambient air temperature
LIGHT	event.values[0]	lx	Illuminance
PRESSURE	event.values[0]	hPa o mbar	Ambient air pressure
RELATIVE HUMIDITY	event.values[0]	%	Ambient relative humidity
TEMPERATURE	event.values[0]	^{o}C	Device temperature

Sensori di posizione

Questa categoria comprende i sensori di posizione, ovvero quelli che monitorano la posizione fisica del dispositivo; questi sono i sensori di orientamento e i magnetometri.

Tabella 1.4: Position sensors that are supported on the Android platform.[8]

Sensors	**Sensor event data**	**Description**	**Units**
MAGNETIC FIELD	SensorEvent.values[0]	Geomagnetic field strength along the x axis.	μT
	SensorEvent.values[1]	Geomagnetic field strength along the y axis.	μT

Continued on next page

Tabella 1.4 – *Continued from previous page*

Sensors	Sensor event data	Description	Units
	SensorEvent.values[2]	Geomagnetic field strength along the z axis.	μT
ORIENTATION	SensorEvent.values[0]	Azimuth (angle around the z-axis).	Degrees
	SensorEvent.values[1]	Pitch (angle around the x-axis).	Degrees
	SensorEvent.values[2]	Roll (angle around the y-axis).	Degrees
PROXIMITY	SensorEvent.values[0]	Distance from object.	cm

1.4.2 ConnectivityManager

In aggiunta, dai dispositivi mobili è possibile monitorare lo stato della rete a cui sono connessi. In particolare, per quanto riguarda le network e le connessioni in generale, c'è la classe ConnectivityManager che ci permette di gestirle e di recuperane i dati. Tuttavia, per il bluetooth e per il wifi, ci sono altre due classi, BluetoothAdapter e Wifimanager che ci permettono di effettuare più operazioni quali la scansione dei dispositivi o delle reti, recuperare i dati di dispositivi o reti precedentemente connessi e di vedere lo stato del proprio dispositivo.

Figura 1.7: Connectivity

1.4.3 LocationManager

Dai dispositivi mobili è possibile ricavare i dati relativi alla posizione dell'utente. A riguardo ciò, la classe del LocationManager, ci permette, a seconda del provider scelto, di recuperare la posizione, nel sistema di coordinate della latitudine e longitudine, e altri dati relativi ad essa. I LocationProvider che è possibile monitorare sono: il Gps, implementando la relativa interfaccia, il Network e il Passive. Il primo ci permette di ricavare i dati della posizione senza far riferimento alla rete, senza utilizzare internet, senza costi, ma sfruttando i satelliti. Invece, il secondo ci permette di recuperare i dati senza interrogare i satelliti, ma sfruttando la connessione alla rete del dispositivo; la precisione di questo dato dipende dalla qualità della rete a cui il dispositivo è connesso.

Figura 1.8: Mobile Sensor

1.4.4 Altri sensori

Dai dispositivi mobili è possibile ricavare i dati anche da altri sensori. In particolare, in questo progetto, sono stati raccolti anche i dati relativi al microfono del dispositivo, monitorando il valore dell'ampiezza d'onda, alla batteria, monitorando il suo livello e il suo stato.

Ciò nonostante, con il continuo sviluppo sia hardware che software dei dispositivi mobili i sensori sono in continua trasformazione e non è da escludere che, a partire dai prossimi smart-phone e dalle prossime versioni Android, ci siano altri sensori che ci permetteranno di raccogliere altri dati per descrivere al meglio la realtà e per fruire di nuovi servizi.

Capitolo 2

Architettura

Il progetto sviluppato per questa tesi di laurea è un sistema che presenta un'architettura di tipo client-server; il client è composto da un'applicazione per dispositivi Android, mentre la parte server è composta da un database e da alcune pagine web. Queste due componenti comunicano attraverso una pagina web che riceve i dati inviati dal client e li salva sul database interno.

2.1 Client

2.1.1 Obiettivi, funzionalità principali e requisiti

L'applicazione sviluppata per la parte client del sistema si chiama Sensor-Monitor. Questa è stata creata per poter acquisire le informazioni dai sensori dei dispositivi mobili, in particolare da quelli che supportano il sistema Android.

Le principali funzionalità di questa parte del sistema sono: visualizzare quali sensori sono installati sul dispositivo, monitorarne lo stato e i valori assunti in un determinato intervallo di tempo e inviare le informazioni ottenute al server attraverso una richiesta HTTP di tipo POST.

Tuttavia, nonostante l'applicazione sia stata pensata per poter acquisire le informazioni dal maggior numero di dispositivi possibili, c'è un requisito che è necessario soddisfare per poterla utilizzare: il dispositivo sul quale si

vuole installare tale applicazione deve supportare un sistema Android con versione 2.3 o superiori.

Figura 2.1: Android

2.1.2 Struttura

L'applicazione è composta da due activities, le quali presentano alcune funzionalità simili, ma hanno delle implementazioni e dei layout abbastanza differenti.

La prima, quella principale, è stata realizzata in modo tale da permettere all'utente di visualizzare la lista dei sensori installati sul proprio dispositivo; tra questi ci sono tutti quelli di movimento, posizione, ambientali, il microfono, la batteria, le network supportate, il bluetooth e i sensori relativi alla localizzazione del dispositivo. Da questa lista, l'utente potrà selezionare o deselezionare i sensori, e quando viene avviata, oppure è già in corso, una sessione per l'acquisizione e il monitoraggio dei dati, questi vengono aggiunti o rimossi dall'elenco di quelli da cui ricevere gli aggiornamenti.
Inoltre, questa activity, come precedentemente accennato, permette di inviare i dati acquisiti al server. La comunicazione con il server avviene attraverso l'invio di oggetti JSON, contenenti tutte le informazioni raccolte, ad una pagina web sul server tramite una richiesta HTTP di tipo POST. A sua volta il server, una volta ricevuti i dati ed effettuate le relative operazioni per salvarli sul database, invierà un messaggio di successo o insuccesso delle attività svolte.

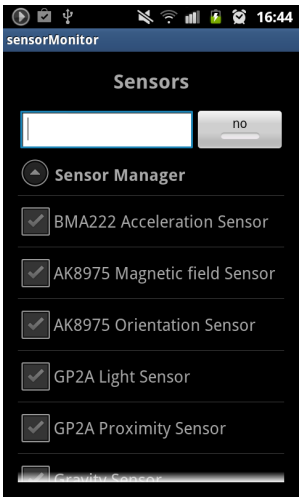

Figura 2.2: SensorActivity

Invece, la seconda activity permette di monitorare lo stato di un singolo sensore, scelto dall'activity principale, visualizzandone tutti i dettagli, i valori assunti nell'intervallo di tempo precedentemente impostato e, in alcuni casi, c'è la possibilità di fare delle operazioni specifiche per alcune categorie di sensori. Questa activity viene aperta una volta che l'utente clicca sul nome di un sensore nell'activity principale.

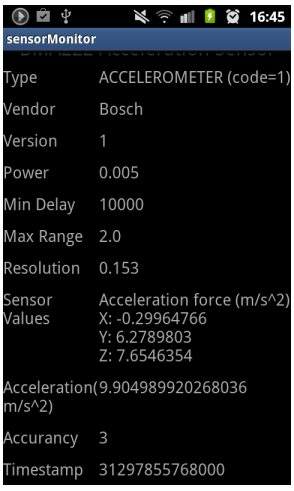

Figura 2.3: SensorDetails

2.2 Server

2.2.1 Obiettivi, funzionalità principali e requisiti di sistema

L'obiettivo del server è quello di raccogliere e memorizzare i dati ricevuti dai client. Per fare ciò, sono state costruite da una serie di pagine php, alcune volte alla gestione ed installazione del database, mentre altre alla ricezione e all'inserimento dei dati, e da un database. Il server deve, quindi, ricevere i dati dai client, elaborarli in maniera tale da poterli salvare sul database e inviare l'esito delle operazioni al client.
Il database utilizzato è MySQL versione 5.5.32, mentre la versione di php utilizzata è PHP5.
Il server di tale sistema si trova all'indirizzo:

http://todeschinitesi.web.cs.unibo.it/ .

Figura 2.4: Google Server

2.2.2 Struttura

Il server, come precedentemente accennato, ha due componenti: una serie di pagine php e un database mysql. Le prime servono per poter gestire al

meglio il database, in particolare possiamo vedere due tipologie di pagine: una prima che serve a gestire, creare e modificare il database, mentre la seconda nell'interazione con il client e nell'esecuzione delle relative operazioni di inserimento dei dati.
Quest'ultima riceve i dati dai client sotto forma di JSON, li decodifica e a seconda del JSON che arriva effettua determinate query per l'inserimento dei dati all'interno del database; se gli inserimenti hanno successo, quindi i dati ricevuti sono corretti, ritorna un messaggio di successo delle operazioni effettuate, con eventualmente dei dati utili ai futuri inserimenti, altrimenti se gli inserimenti falliscono o non viene riconosciuto il tipo di JSON inviato ritorna al client un messaggio di errore.

Per quanto riguarda il database, la sua struttura ed architettura è osservabile nel seguente diagramma Entità-Relazione:

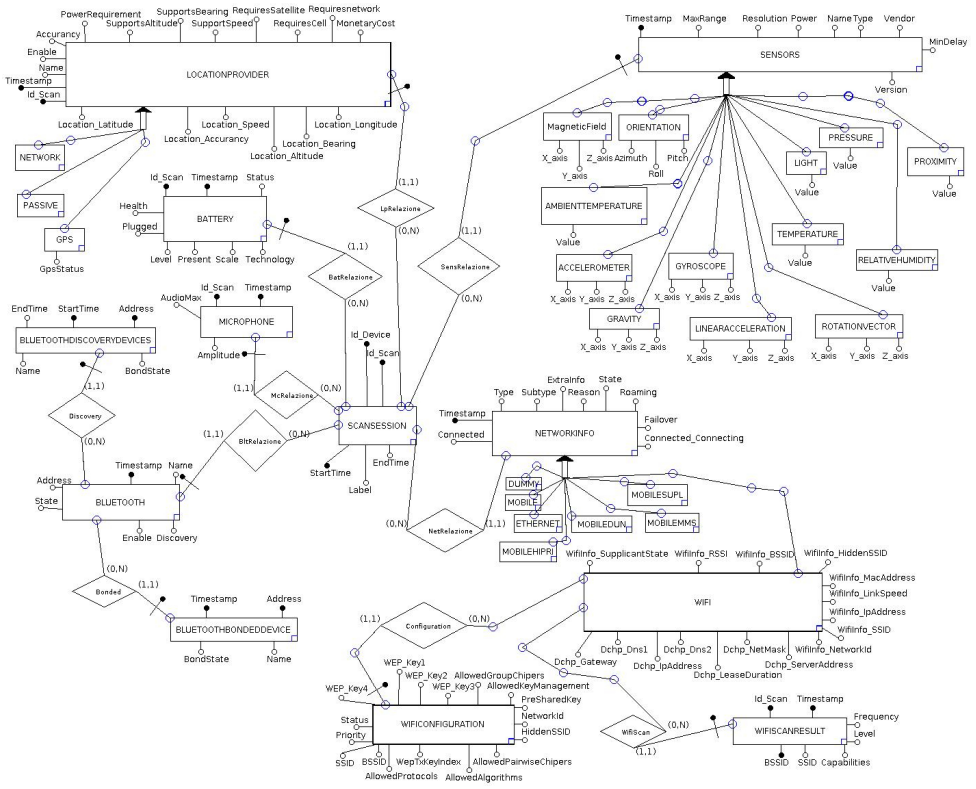

Figura 2.5: Diagramma Entità-Relazione

Da questo diagramma è stata estrapolata successivamente la seguente struttura logica:

Tabella 2.1: Diagramma Logico

Table	**Foreign Key**
SCANSESSION(Id_Scan, Id_Device, (*StartTime*) , EndTime , Label)	
BATTERY(Id_Scan, Timestamp, Status, Health, Plugged, Level, Present, Scale, Technology, Temperature, Voltage)	BATTERY (Id_Scan) ⟶ SCANSESSION (Id_Scan)
MICROPHONE(Id_Scan, Timestamp, AudioMax, Amplitude)	MICROPHONE (Id_Scan) ⟶ SCANSESSION (Id_Scan)
NETWORK(Id_Scan, Name, Enable, Accurancy, PowerRequirement, MonetaryCost, RequiresCell , RequiresNetwork, RequiresSatellite, SupportsAltitude, SupportsBearing, SupportSpeed, Timestamp, Location_Accurancy, Location_Altitude, Location_Bearing, Location_Latitude, Location_Longitude, Location_Speed)	NETWORK (Id_Scan) ⟶ SCANSESSION (Id_Scan)
PASSIVE(Id_Scan, Name, Enable, Accurancy, PowerRequirement, MonetaryCost, RequiresCell , RequiresNetwork, RequiresSatellite, SupportsAltitude, SupportsBearing, SupportSpeed, Timestamp, Location_Accurancy, Location_Altitude, Location_Bearing, Location_Latitude, Location_Longitude, Location_Speed)	PASSIVE (Id_Scan) ⟶ SCANSESSION (Id_Scan)

Continued on next page

Tabella 2.1 – *Continued from previous page*

Table	**Foreign Key**
GPS(Id_Scan, Name, Enable, Accurancy, PowerRequirement, MonetaryCost, RequiresCell , RequiresNetwork, RequiresSatellite, SupportsAltitude, SupportsBearing, SupportSpeed, GpsStatus, Timestamp, Location_Accurancy, Location_Altitude, Location_Bearing, Location_Latitude, Location_Longitude, Location_Speed)	GPS (Id_Scan) ⟶ SCANSESSION (Id_Scan)
ACCELEROMETER(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)	ACCELEROMETER (Id_Scan) ⟶ SCANSESSION (Id_Scan)
MAGNETICFIELD(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)	MAGNETICFIELD (Id_Scan) ⟶ SCANSESSION (Id_Scan)
GYROSCOPE(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)	GYROSCOPE (Id_Scan) ⟶ SCANSESSION (Id_Scan)
LIGHT(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)	LIGHT (Id_Scan) ⟶ SCANSESSION (Id_Scan)
PRESSURE(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)	PRESSURE (Id_Scan) ⟶ SCANSESSION (Id_Scan)

Continued on next page

Tabella 2.1 – *Continued from previous page*

Table	**Foreign Key**
PROXIMITY(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)	PROXIMITY (Id_Scan) ⟶ SCANSESSION (Id_Scan)
GRAVITY(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)	GRAVITY (Id_Scan) ⟶ SCANSESSION (Id_Scan)
LINEARACCELERATION(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)	LINEAR-ACCELETATION (Id_Scan) ⟶ SCANSESSION (Id_Scan)
ROTATIONVECTOR(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)	ROTATIONVECTOR (Id_Scan) ⟶ SCANSESSION (Id_Scan)
ORIENTATION(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Azimuth, Pitch, Roll, Timestamp)	ORIENTATION (Id_Scan) ⟶ SCANSESSION (Id_Scan)
RELATIVEHUMIDITY(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)	RELATIVEHUMIDITY (Id_Scan) ⟶ SCANSESSION (Id_Scan)

Continued on next page

Tabella 2.1 – *Continued from previous page*

Table	**Foreign Key**
AMBIENTTEMPERATURE(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)	AMBIENTTEMPERATURE (Id_Scan) ⟶ SCANSESSION (Id_Scan)
TEMPERATURE(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)	TEMPERATURE (Id_Scan) ⟶ SCANSESSION (Id_Scan)
DUMMY(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)	DUMMY (Id_Scan) ⟶ SCANSESSION (Id_Scan)
ETHERNET(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)	ETHERNET (Id_Scan) ⟶ SCANSESSION (Id_Scan)
MOBILE(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)	MOBILE (Id_Scan) ⟶ SCANSESSION (Id_Scan)
MOBILEDUN(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)	MOBILEDUN (Id_Scan) ⟶ SCANSESSION (Id_Scan)

Continued on next page

Tabella 2.1 – *Continued from previous page*

Table	**Foreign Key**
MOBILEHIPRI(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)	MOBILEHIPRI (Id_Scan) ⟶ SCANSESSION (Id_Scan)
MOBILEMMS(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)	MOBILEMMS (Id_Scan) ⟶ SCANSESSION (Id_Scan)
MOBILESUPL(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)	MOBILESUPL (Id_Scan) ⟶ SCANSESSION (Id_Scan)
WIFI(Id_Scan, Timestamp, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Dchp_Dns1, Dchp_Dns2, Dchp_Gateway, Dchp_IpAddress, Dchp_LeaseDuration, Dchp_Netmask, Dchp_ServerAddress, WifiInfo_BSSID, WifiInfo_HiddenSSID, WifiInfo_SSID, WifiInfo_IpAddress, WifiInfo_LinkSpeed, WifiInfo_MacAddress, WifiInfo_NetworkId, WifiInfo_RSSI, WifiInfo_SupplicantState)	WIFI (Id_Scan) ⟶ SCANSESSION (Id_Scan)

Continued on next page

Tabella 2.1 – *Continued from previous page*

Table	**Foreign Key**
WIFICONFIGURATION(Id_Scan, Timestamp, AllowedAuthAlgorithms, AllowedGroupCiphers, AllowedKeyManagement, AllowedPairwiseCiphers, AllowedProtocols, BSSID, HiddenSSID, NetworkId, preSharedKey, Priority, Status, SSID, WEP_Key1, WEP_Key2, WEP_Key3, WEP_Key4, WepTxKeyIndex) WIFICONFIGURATION(Id_Scan, Timestamp) ⟶ WIFI(Id_Scan, Timestamp)	
WIFISCANRESULT(Id_Scan, Timestamp, BSSID, SSID, Capabilities, Frequency, Level)	WIFI (Id_Scan) ⟶ WIFI (Id_Scan)
BLUETOOTH(Id_Scan, Name, Address, State, Enable, Discovery, ScanMode, Timestamp)	BLUETOOTH (Id_Scan) ⟶ SCANSESSION (Id_Scan)
BLUETOOTHDISCOVERYDEVICES(Id_Scan, StartTime, EndTime, Address, BondState, Name)	BLUETOOTH-DISCOVERY-DEVICES (Id_Scan) ⟶ BLUETOOTH (Id_Scan)
BLUETOOTHBONDEDDEVICE(Id_Scan, Timestamp, Address, BondState, Name)	BLUETOOTH-BONDEDDEVICE (Id_Scan) ⟶ BLUETOOTH (Id_Scan)

Dizionario delle Entità

Tabella 2.2: Dizionario delle Entità

Entità e descrizione	**Attributi**	**ID**
SCANSESSION Sessioni degli utenti nella catalogazione dei dati dei sensori	Id_Scan INT NOT NULL AUTO_INCREMENT, Id_Device VARCHAR(255) NOT NULL, StartTime BIGINT NOT NULL, EndTime BIGINT, Label TEXT	Id_Scan, Timestamp
BATTERY Batteria del dispositivo	Id_Scan INT(11) NOT NULL, Timestamp BIGINT, Status INT, Health INT, Plugged INT, Level INT, Present INT, Scale INT, Technology TEXT, Temperature INT, Voltage INT	Id_Scan, Timestamp
MICROPHONE Microfono del dispositivo	Id_Scan INT(11) NOT NULL, Timestamp BIGINT, AudioMax INT, Amplitude FLOAT	Id_Scan, Timestamp
NETWORK Network provider	Id_Scan INT(11) NOT NULL, Name VARCHAR(10), Enable TINYINT(1), Accurancy INT, PowerRequirement INTEGER, MonetaryCost TINYINT(1), RequiresCell TINYINT(1), RequiresNetwork TINYINT(1), RequiresSatellite TINYINT(1), SupportsAltitude TINYINT(1), SupportsBearing TINYINT(1), SupportSpeed TINYINT(1), Timestamp BIGINT, Location_Accurancy FLOAT, Location_Altitude DOUBLE, Location_Bearing FLOAT, Location_Latitude DOUBLE, Location_Longitude DOUBLE, Location_Speed FLOAT	Id_Scan, Timestamp

Continued on next page

Tabella 2.2 – *Continued from previous page*

Entità e descrizione	**Attributi**	**ID**
PASSIVE Passive provider	Id_Scan INT(11) NOT NULL, Name VARCHAR(10), Enable TINYINT(1), Accurancy INT, PowerRequirement INTEGER, MonetaryCost TINYINT(1), RequiresCell TINYINT(1), RequiresNetwork TINYINT(1), RequiresSatellite TINYINT(1), SupportsAltitude TINYINT(1), SupportsBearing TINYINT(1), SupportSpeed TINYINT(1), Timestamp BIGINT, Location_Accurancy FLOAT, Location_Altitude DOUBLE, Location_Bearing FLOAT, Location_Latitude DOUBLE, Location_Longitude DOUBLE, Location_Speed FLOAT	Id_Scan, Timestamp
GPS GPS provider	Id_Scan INT(11) NOT NULL, Name VARCHAR(10), Enable TINYINT(1), Accurancy INT, PowerRequirement INTEGER, MonetaryCost TINYINT(1), RequiresCell TINYINT(1), RequiresNetwork TINYINT(1), RequiresSatellite TINYINT(1), SupportsAltitude TINYINT(1), SupportsBearing TINYINT(1), SupportSpeed TINYINT(1), GpsStatus INT, Timestamp BIGINT, Location_Accurancy FLOAT, Location_Altitude DOUBLE, Location_Bearing FLOAT, Location_Latitude DOUBLE, Location_Longitude DOUBLE, Location_Speed FLOAT	Id_Scan, Timestamp

Continued on next page

Tabella 2.2 – *Continued from previous page*

Entità e descrizione	**Attributi**	**ID**
ACCELEROMETER Accelerometrodel dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
MAGNETICFIELD Magnetometro del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
GYROSCOPE Giroscopio del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
LIGHT Sensore di luminosità del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
PRESSURE Sensore di pressione del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT	Id_Scan, Timestamp

Continued on next page

Tabella 2.2 – *Continued from previous page*

Entità e descrizione	**Attributi**	**ID**
PROXIMITY Sensore di prossimità del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
GRAVITY Sensore di gravità del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
LINEAR-ACCELERATION Sensore di accelerazione lineare del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
ROTATION-VECTOR Sensore di rotazione del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
ORIENTATION Sensore dell'orientamento del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Azimuth FLOAT, Pitch FLOAT, Roll FLOAT, Timestamp BIGINT	Id_Scan, Timestamp

Continued on next page

Tabella 2.2 – *Continued from previous page*

Entità e descrizione	**Attributi**	**ID**
RELATIVE-HUMIDITY Sensore dell'umidità del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
AMBIENT-TEMPERATURE Sensore di temperatura del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
TEMPERATURE Sensore di temperatura del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT	Id_Scan, Timestamp
DUMMY Network supportata dal dispositivo	Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT	Id_Scan, Timestamp

Continued on next page

Tabella 2.2 – *Continued from previous page*

Entità e descrizione	**Attributi**	**ID**
ETHERNET Network supportata dal dispositivo	Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT	Id_Scan, Timestamp
MOBILE Network supportata dal dispositivo	Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT	Id_Scan, Timestamp
MOBILEDUN Network supportata dal dispositivo	Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT	Id_Scan, Timestamp

Continued on next page

Tabella 2.2 – *Continued from previous page*

Entità e descrizione	**Attributi**	**ID**
MOBILEHIPRI Network supportata dal dispositivo	Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT	Id_Scan, Timestamp
MOBILEMMS Network supportata dal dispositivo	Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT	Id_Scan, Timestamp
MOBILESUPL Network supportata dal dispositivo	Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT	Id_Scan, Timestamp

Continued on next page

Tabella 2.2 – *Continued from previous page*

Entità e descrizione	**Attributi**	**ID**
WIFI Wi-Fi del dispositivo	Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT, Dchp_Dns1 INT, Dchp_Dns2 INT, Dchp_Gateway INT, Dchp_IpAddress INT, Dchp_LeaseDuration INT, Dchp_Netmask INT, Dchp_ServerAddress INT, WifiInfo_BSSID TEXT, WifiInfo_HiddenSSID TINYINT(1), WifiInfo_SSID TEXT, WifiInfo_IpAddress INT, WifiInfo_LinkSpeed INT, WifiInfo_MacAddress TEXT, WifiInfo_NetworkId INT, WifiInfo_RSSI INT, WifiInfo_SupplicantState TEXT	Id_Scan, Timestamp
WIFICONFIGURATION Reti Wi-Fi configurate all'interno del dispositivo	Id_Scan INT(11) NOT NULL, Timestamp BIGINT, AllowedAuthAlgorithms TEXT, AllowedGroupCiphers TEXT, AllowedKeyManagement TEXT, AllowedPairwiseCiphers TEXT, AllowedProtocols TEXT, BSSID TEXT, HiddenSSID TINYINT(1), NetworkId INT, PreSharedKey TEXT, Priority INT, Status INT, SSID TEXT, WEP_Key1 TEXT, WEP_Key2 TEXT, WEP_Key3 TEXT, WEP_Key4 TEXT, WepTxKeyIndex INT	Id_Scan, Timestamp

Continued on next page

Tabella 2.2 – *Continued from previous page*

Entità e descrizione	**Attributi**	**ID**
WIFISCAN-RESULT Wi-Fi trovate con la scansione delle reti dal dispositivo	Id_Scan INT(11) NOT NULL, Timestamp BIGINT, BSSID VARCHAR(255), SSID TEXT, Capabilities TEXT, Frequency INT, Level INT	Id_Scan, Timestamp, BSSID
BLUETOOTH Bluetooth del dispositivo	Id_Scan INT(11) NOT NULL, Name TEXT, Address TEXT, State INT, Enable TINYINT(1), Discovery TINYINT(1), ScanMode INT, Timestamp BIGINT	Id_Scan, Timestamp
BLUETOOTH-DISCOVERY-DEVICES Device trovati dalla scansione del bluetooth del dispositivo	Id_Scan INT(11) NOT NULL, StartTime BIGINT, EndTime BIGINT, Address VARCHAR(255), BondState INT, Name TEXT	Id_Scan, Timestamp, Address
BLUETOOTH-BONDED-DEVICE Bonded device del dispositivo	Id_Scan INT(11) NOT NULL, Timestamp BIGINT, Address VARCHAR(255), BondState INT, Name TEXT	Id_Scan, Timestamp, Address

Dizionario delle Relazioni

Tabella 2.3: Dizionario delle Relazioni

Relazione	Descrizione	Componenti
BltRelazione	Relazione tra la tabella contenente i dati generali delle sessioni e la tabella del bluetooth	SCANSESSION, BLUETOOTH
Discovery	Relazione tra la tabella bluetooth e la tabella contenente i dati dei dispositivi trovati nelle scansioni.	BLUETOOTH, BLUETOOTHDISCOVERYDEVICES
Bonded	Relazione tra la tabella bluetooth e la tabella contenente i dati dei bonded device.	BLUETOOTH, BLUETOOTHBONDEDDEVICE
McRelazione	Relazione tra la tabella contenente i dati generali delle sessioni e la tabella MICROPHONE	SCANSESSION, MICROPHONE
BatRelazione	Relazione tra la tabella contenente i dati generali delle sessioni e la tabella battery	SCANSESSION, BATTERY
LpRelazione	Relazione tra la tabella contenente i dati generali delle sessioni e le tabelle dei location provider	SCANSESSION, GPS, PASSIVE, NETWORK

Continued on next page

Tabella 2.3 – *Continued from previous page*

Relazione	**Descrizione**	**Componenti**
SensRelazione	Relazione tra la tabella contenente i dati generali delle sessioni e le tabelle dei sensori	SCANSESSION, ACCELEROMETER, MAGNETICFIELD, GYROSCOPE, LIGHT, PRESSURE, PROXIMITY, GRAVITY, LINEARACCELERATION, ROTATIONVECTOR, ORIENTATION, RELATIVEHUMIDITY, AMBIENTTEMPERATURE, TEMPERATURE
NetRelazione	Relazione tra la tabella contenente i dati generali delle sessioni e le tabelle delle network supportate	SCANSESSION, DUMMY, ETHERNET, MOBILE, MOBILEDUN, MOBILEHIPRI, MOBILEMMS, MOBILESUPL
Configuration	Relazione tra la tabella contenente i dati dei Wi-Fi dei dispositivi e quella contenente i dati delle reti configurate.	WIFI, WIFICONFIGURATION
WifiScan	Relazione tra la tabella contenente i dati dei Wi-Fi dei dispositivi e quella contenente i dati delle reti scansionate.	WIFI, WIFISCANRESULT

Capitolo 3

Implementazione

3.1 Client

Il client, come già detto nel capitolo precedente, è un'applicazione per dispositivi Android con versione 2.3 o superiore. Nel capitolo precedente è stata data una visione generale dell'architettura dell'applicazione, in questo analizzeremo più nel dettaglio la sua implementazione. Come tutte le applicazioni Android, la parte grafica è stata fatta col linguaggio XML, mentre la parte dei calcoli e delle operazioni con Java. Per quanto riguarda la parte grafica non mi soffermerò molto, in quanto credo che gli screenshot proposti nel corso della spiegazione delle varie activity siano sufficienti per capirne la struttura.

L'applicazione è composta, oltre ai file relativi alla grafica, da due activities (SensorActivity.java e SensorDetails.java), da un file che gestisce la lista espandibile (SensorExpandableAdapter.java) e da un file che crea, gestisce e invia i JSON (JSONSender.java) al server. Nelle prossime sezioni vedremo questi file analizzati nel dettaglio.

3.1.1 SensorActivity.java

Il file contiene la classe SensorActivity; essa estende Activity e implementa SensorEventListener, LocationListener, GpsStatus.Listener. Questa è

l'activity principale, quella che viene lanciata subito appena viene eseguita l'applicazione.

Il layout è composto: dal titolo dell'applicazione, da una label in cui l'utente può scrivere un messaggio relativo alla sessione da monitorare, un togglebutton che permette di far partire o terminare la sessione per la raccolta dei dati e una lista espandibile contenente tutti i sensori implementati dal sistema e installati nel dispositivo raggruppati per tipologia. La struttura e l'implementazione della lista è spiegata più avanti, mentre ora analizzeremo la struttura di questa activity.

L'activity, quando viene creata, carica il layout e invoca il metodo loadSensorList() che serve per recuperare la lista dei sensori implementati ed installati nel dispostivo. Tale metodo, appena viene eseguito va ad invocare a sua volta i metodi di loadSensorManager(), loadConnectivityManager(), loadLocationManager(), loadOther() che permettono di recuperare rispettivamente i sensori che appartengono al SensorManager, ConnectivityManager, LocationManager e quello della batteria e del microfono. Tutti questi sensori vengono messi in stutture dati, come le Liste, e successivamente vengono unite ed organizzate per poterle passare come parametri alla classe che crea e gestisce la lista espandibile.

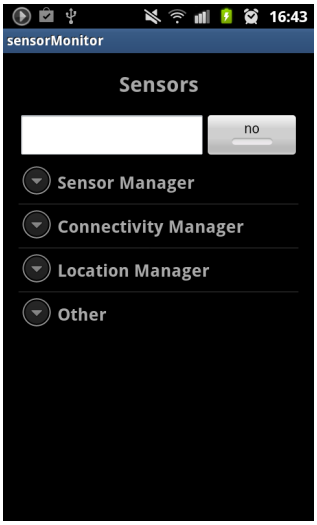

Figura 3.1: Sessione attiva

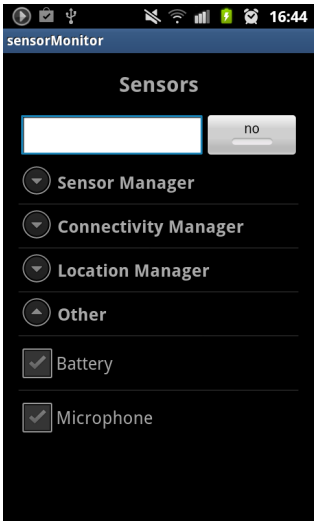

Figura 3.2: Sessione chiusa

Inoltre, l'utente può scegliere da quali sensori ricevere gli aggiornamenti, selezionandoli o deselezionandoli attraverso la checkbox presente in ogni elemento della lista; se selezionati, vengono aggiunti alla lista dei sensori da ascoltare, altrimenti vengono rimossi. Per ricevere gli aggiornamenti dai sensori, questa activity implementa SensorEventListener, LocationListener, GpsStatus.Listener e contiene una classe NetworkReceiverAdapter che estende BroadcastReceiver. Le interfacce, implementando i relativi metodi, servono per ricevere gli aggiornamenti relativi ai sensori del SensorManager, LocationManager e GPS, mentre la classe da quelli relativi al ConnectivityManager, Bluetooth, Wifi e batteria; per questi ultimi, se selezionati, viene aggiunto un nuovo oggetto della classe NetworkReceiverAdapter, viene registrato il relativo receiver ed aggiunto ad un ArrayList che contiene tutti i receiver registrati, invece se deselezionato viene eliminato il receiver precedentemente registrato.

Quando l'utente avvia la sessione per il monitoraggio dei dati, per ogni sensore selezionato verranno inviati i dati relativi agli aggiornamenti attraverso l'invocazione dei metodi statici della classe JSONSender spiegata successivamente. Appena l'utente ferma la scansione, i dati smetteranno di essere inviati. Inoltre, è stato creato un sistema di preferenze attraverso il quale l'utente può impostare l'intervallo di tempo per gli aggiornamenti di alcune categorie di sensori.

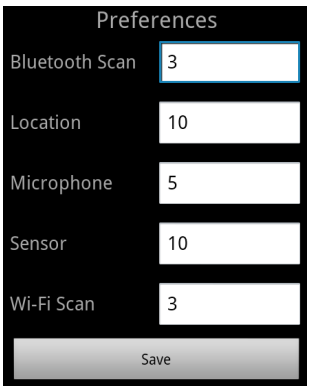

Figura 3.3: Preferenze

3.1.2 SensorExpandableAdapter.java

Il file contiene la classe SensorExpandableAdapter che estende BaseExpandableListAdapter. Questa permette di disegnare e gestire la lista espandibile dell'activity principale. In particolare la classe, necessita come parametri nel costruttore di una HashMap e di una List, che contengono le informazioni relative a ciò che va messo e in che ordine all'interno della lista. Inoltre, oltre a contenere i metodi per recuperare gli elementi della lista contiene un metodo getChildView(int groupPosition, int childPosition, boolean isLastChild, View v, ViewGroup viewGroup) che gestisce il contenuto dei singoli elementi di ogni gruppo della lista.

Di particolare importanza in questo metodo troviamo, oltre alla selezione relativa al caricamento del layout, una serie di istruzioni che permettono di caricare, alcuni dei dettagli dei sensori. Per identificare la categoria del sensore, state utilizzate le eccezioni ClassCastException, cioè per recuperare l'elemento viene invocato il metodo getChild di questa classe e viene effettuato un cast a seconda della tipologia del sensore. Se ciò non genera un'eccezione di ClassCastException significa che l'elemento è un oggetto dello stessa classe di quella utilizzata per il cast, altrimenti significa che le classi sono differenti e, con la stessa logica, vengono controllate le altre classi relative alle altre tipologie di sensori finchè non si trova quella corretta.

Figura 3.4: Sessione attiva

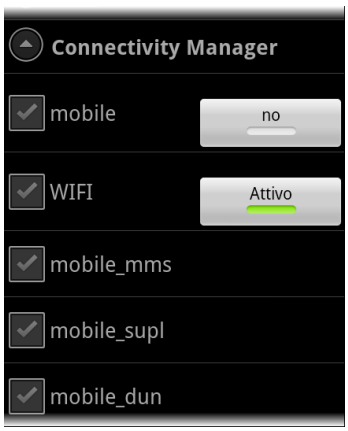

Figura 3.5: Sessione chiusa

Il layout utilizzato per i singoli elementi è composto da una checkbox, che serve per aggiungere o rimuovere il sensore da quelli che si vogliono monitorare, e da una label che contiene il nome o la tipologia del sensore, che, se cliccata, aprirà l'activity contenente i dettagli di quello specifico elemento. Tuttavia, in alcuni casi c'è anche un bottone che permette, a seconda del sensore di abilitarlo o disabilitarlo.

3.1.3 SensorDetails.java

Questo file contiene la classe SensorDetails che estende Activity e implementa SensorEventListener, LocationListener, GpsStatus.Listener. Questa è l'activity che permette di visualizzare sul display dello smartphone i dettagli e i valori assunti dal sensore in quel preciso momento. Questa viene lanciata quando, dall'activity principale si clicca sul nome del sensore di cui se ne vogliono vedere i dettagli.

L'activity, quando viene creata va a ricercare negli extra una variabile che identifica la categoria del sensore selezionato. A seconda della categoria verrà caricato il layout corrispondente e verranno invocati i metodi che permettono di visualizzare le informazioni del sensore scelto. Inoltre, l'activity verrà automaticamente messa in ascolto degli eventi e degli aggiornamenti generati dal sensore, ovvero quando questo avrà un cambiamento nello stato, nei valori assunti o in qualcuna delle sue informazioni verrà aggiornato il contenuto dell'activity.

Come abbiamo precedentemente detto, la classe implementa una serie di interfacce. Queste permettono, implementando i relativi metodi, di ricevere gli aggiornamenti sugli stati, sui valori e sulle informazioni dei sensori. In particolare, implementando SensorEventListener si ottengono gli aggiornamenti di quelli relativi al SensorManager, invece implementando LocationListener si ottengono quelli dei LocationProvider relativi al LocationManager, mentre GpsStatus.Listener aggiorna sullo stato del gps.

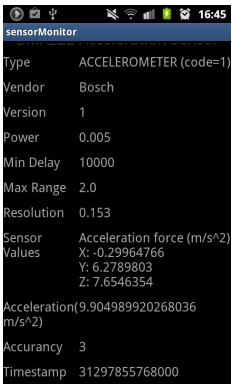

Figura 3.6: Sensor

Figura 3.7: LocationProvider

Tutttavia, le interfacce implementate non bastano per recuperare gli aggiornamenti da tutti i sensori, poichè alcuni necessitano di un BroadcastReceiver. A tal proposito, è stata implementata anche una classe NetworkReceiver che, appunto, estende BroadcastReceiver e permette di recuperare le informazioni degli aggiornamenti dello stato delle networkInfo, della batteria, del bluetooth e del wifi; ogni qual volta questa classe riceve un aggiornamento aggiorna il contenuto del layout dell'activity. Per ricevere gli aggiornamenti bisogna che l'activity sia registrata al receiver con il corretto intent filter, questo accade nel metodo onResume(), mentre quando si mette in pausa o si chiude l'activity non è necessario che i BroadcastReceiver registrati rimangano tali, per ciò viene eseguita l'operazione di unregister.

Figura 3.8: NetworkInfo

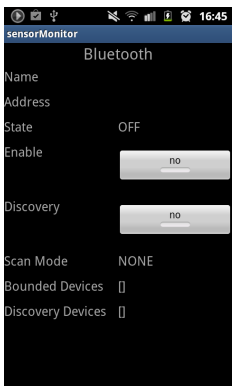

Figura 3.9: Bluetooth

3.1.4 JSONSender.java

Il file continene una classe costituita da una serie di metodi statici e ha come obiettivo quello di inviare, tramite una richiesta HTTP di tipo POST, i dati raccolti, sotto forma di un JSONObject, al server del sistema.

Analizzando più nel dettaglio, ci sono due metodi start e stop che, quando invocati, permettono di avviare o terminare la sessione del monitoraggio dei dati; in questi due metodi, a differenza dei successivi, una volta ricevuta la risposta dal server viene modificato il valore della variabile statica SESSION_ID di SensorActivity.java. Questo perchè se si avvia la scansione l'utente è in grado di sapere quale è l'id della sessione che ha attivato, e una volta che ferma la scansione il valore della variabile viene posto a -1.

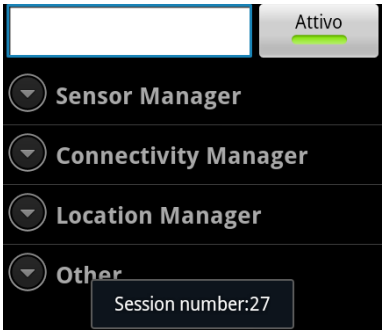

Figura 3.10: Sessione attiva

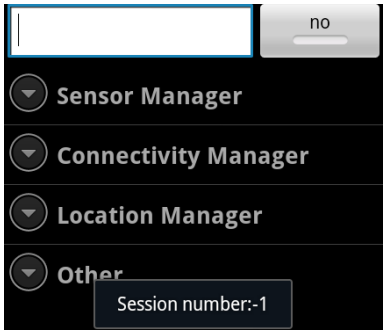

Figura 3.11: Sessione chiusa

Tuttavia, ci sono anche una serie di metodi che permettono di costruire il JSON contenente le informazioni e i dati dei sensori selezionati. I metodi in questione sono descritti nella seguente tabella:

Tabella 3.1: Metodi di JSONSender.java

Metodo	**sensorClass Type**	**Descrizione**
battery(int status, int health, int plugged, int present, int level, int scale, String technology, int temperature, int voltage)	"Battery"	Permette di costruire il json contenente le informazioni della batteria passate come parametri.
microphone(int audioMax, double amplitude)	"Microphone"	Permette di costruire il json contenente i valori del microfono passati come parametri.
sensor(Sensor s, SensorEvent sensorEvent)	"Sensor"	Permette di costruire il json contenente i dati e le informazioni del sensore passato come parametro assieme all'evento che ha generato.
locationProvider(LocationManager lm, LocationProvider lp, Location loc, int statusCode)	"Location Provider"	Permette di costruire il json contenente le informazioni del locationprovider monitorato e della posizione monitorata.

Continued on next page

Tabella 3.1 – *Continued from previous page*

Metodo	**sensorClass Type**	**Descrizione**
network(NetworkInfo ni, WifiManager wm)	"NetworkInfo"	Permette di costruire il json contenente le informazioni della network monitorata. Inoltre, nel caso si stia monitorando il Wifi, al json construito vengono aggiunte altre informazioni più specifiche recuperabili da l WifiManager.
wifiScan(WifiManager wm)	"WifiScanResult"	Permette di costruire il json contenente le informazioni delle reti Wi-Fi scansionate
bluetooth()	"Bluetooth"	Permette di costruire il json contenente le informazioni del bluetooth recuperandole direttamente dal BluetoothAdapter del sistema.
bluetoothScan(ArrayList <BluetoothDevice> bltDeviceList, long bltStartDiscoveryTime)	"Bluetooth Discovery"	Permette di costruire il json contenente le informazioni dei dispostivi bluetooth scansionati conenuti nella lista passata come parametro.

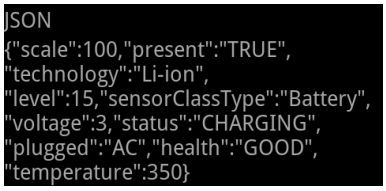

Figura 3.12: JSON batteria

Figura 3.13: JSON accelerometro

Infine, c'è il metodo sendJSON(JSONObject js), che a differenza dei precedenti è privato poichè viene invocato all'interno dei metodi elencati nella tabella precedente. Questo metodo permette di inviare l'oggetto js, passato come parametro, al server del sistema. All'oggetto js vengono aggiunti due valori, IDSession e TimestampSend, che servono al server per identificare la sessione del monitoraggio dei dati attiva e l'istante in cui questi vengono inviati.

L'invio del js avviene, come precedentemente accennato, tramite una richiesta HTTP di tipo POST, in modo da non rendere visibili i dati raccolti. Di seguito c'è uno screenshot della parte di codice relativa all'esecuzione della richiesta.

```
private static void sendJSON(JSONObject js){
    try{
        js.put("IdSession",SensorActivity.getID_SESSION());
        js.put("TimestampSend",(new Date()).getTime());

        try{
            //Sending  Post Request
            HttpClient httpCLient= new DefaultHttpClient();
            HttpPost httpPost=new HttpPost("http://todeschinitesi.web.cs.unibo.it/sensorMonitorServer.php");
            List<NameValuePair> postParameters= new ArrayList<NameValuePair>();
            postParameters.add(new BasicNameValuePair("jsonDataSensor",js.toString()));
            HttpEntity httpEntity=new UrlEncodedFormEntity(postParameters);
            ResponseHandler<String> responseHandler = new BasicResponseHandler();
            httpPost.setEntity(httpEntity);
            String response=httpCLient.execute(httpPost,responseHandler);
            Toast.makeText(SensorActivity.getContext(), "SEND---->\n"+response, Toast.LENGTH_LONG).show();
        }catch(Exception e){Toast.makeText(SensorActivity.getContext(), e.getMessage(), Toast.LENGTH_LONG).show();}
    }catch(JSONException jsExc){}
}
```

Figura 3.14: HTTPRequest

3.2 Server

In questa sezione del capitolo verrà descritto come è stato implementata la parte server del progetto sviluppato per questa tesi di laurea. Come già detto precedentemente, la parte server è composta da una serie di pagine php volte alla creazione e alla gestione di un database, e dal database che contiene tutti i dati collezionati dai dispositivi. Le pagine web implementate per questo progetto sono: GestioneDB.php, installDB.php e sensorMonitorServer.php.

Figura 3.15: Server

3.2.1 GestioneDB.php

Il file contiene la classe GestioneDB che serve per gestire la connessione e disconnessione al database, e l'esecuzione delle istruzioni sql del progetto. Analizzando il codice della classe possiamo trovare tre metodi statici, di cui due privati, connettiDB e disconnettiDB, e uno pubblico eseguiQuery. I metodi sono stati implementati statici perchè così è necessario costruire un oggetto della classe per poterne sfruttare i metodi, inoltre i primi due metodi elencati sono privati perchè vengono, e devono essere, invocati esclusivamente dal metodo pubblico della classe.

Figura 3.16: Database

Il metodo connettiDB() viene invocato ogni qual volta viene eseguito il metodo eseguiQuery, e permette di gestire la connessione con il database. All'interno di questo metodo, viene letto un file json, dbInfo.json, che contiene tutti i dati di configurazione, utili alla connessione con il database, come nome, user, password e indirizzo; queste informazioni vengono salvate in un array associativo. Successivamente viene creato un oggetto mysqli passando come parametri l'indirizzo, lo user e la password del database; una volta creato con successo l'oggetto, viene selezionato il database con il nome precedentemente letto dal file di configurazione e alla fine ritorna l'oggetto mysqli appena creato.

Invece, il metodo disconnettiDB() viene richiamato, anch'esso, dal metodo eseguiQuery, ma solamente volta effettuata l'esecuzione della query, e gestisce la disconnessione dal database; se questa operazione ha successo restituisce il valore booleano true.

Infine, c'è il metodo pubblico eseguiQuery che richiede come parametro una stringa contenente la query da eseguire. Il metodo come prima cosa, invoca connettiDB() recuperando l'oggetto di tipo mysqli attraverso il quale è possibile far eseguire la query. Successivamente viene eseguita la query: se questa genera errore viene inviato un messaggio di errore, altrimenti viene richiamato il metodo disconnettiDB e successivmente vengono inviati i risul-

tati della query. Il metodo, implementato in questa maniera, permette di eseguire qualsiasi tipo di query; nel caso delle SELECT ritornerà i risultati della SELECT, mentre per le query di INSERT, UPDATE E DELETE ritornerà un valore booleano che indica se è stata esguita con successo o no.
.

3.2.2 installDB.php

Il file installDB.php ha come obiettivo quello di installare le tabelle all'interno del database. All'interno del file, ci sono inizialmente una serie di istruzioni attraverso le quali vengono eliminate tutte le tabelle presenti nel database e successivamente vengono create le query volte alla creazione delle tabelle. Queste query vengono salvate all'interno di un array $q, ordinate in modo tale da rispettare i vincoli delle chiavi esterne, e successivamente eseguite come mostrato dal seguente estratto di codice:

```
foreach($q as $query){
$v=GestioneDB::eseguiQuery($query);
}
```

Figura 3.17: Database2

Le query, relative all'installazione delle tabelle del database, sono elencate nell'appendice A.

3.2.3 sensorMonitorServer.php

Il file, a differenza dei precedenti, gestisce l'interazione dei client con il database; a questa pagina i client inviano i dati che verranno successivamente salvati sul database.

Affinchè vengano inseriti sul database, la pagina richiede un parametro POST in ingresso che si chiama jsonDataSensor che deve contenere il JSON con le informazioni da salvare. Una volta verificata l'esistenza di questa variabile, viene letto e decodificato il JSON in ingesso e salvati i dati all'interno di un array associativo denominato json. Successivamente viene, controllato il contenuto di json[sensorClassType] e verrà eseguita, a seconda del contenuto della variabile, la relativa operazione definita all'interno switch; i casi dello switch sono elencati e descritti nell'appendice B.

Figura 3.18: Server Interaction

Capitolo 4

Analisi delle prestazioni

In questo capitolo verrà descritto il modo con cui sono state eseguite le analisi prestazionali dell'applicazione sviluppata lato client. I valori presi in considerazione sono quelli che riguardano il consumo della batteria, sia nell'utilizzo della CPU, sia nell'utilizzo del display; l'applicazione grazie al quale è stata possibile l'analisi è PowerTutor.

Una prima analisi è stata fatta con l'applicazione che, senza essere in ascolto di alcun sensore, procede nella sua esecuzione. Come è possibile vedere nella figura, il consumo della batteria è di 38.7 J; 37.8 J dal LCD e 0.9 J dalla CPU .

Invece, una seconda analisi è stata fatta con l'applicazione in ascolto dei sensori relativi al SensorManager, del sensore della batteria e del microfono (il campionamento dei dati avveniva ogni 5 secondi). A differenza della precedente analisi, il consumo della batteria è di 117.8 J; 110.9 J dal LCD e 6.9 J dalla CPU .

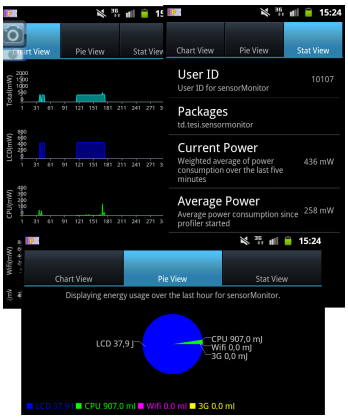

Figura 4.1: Prestazioni base

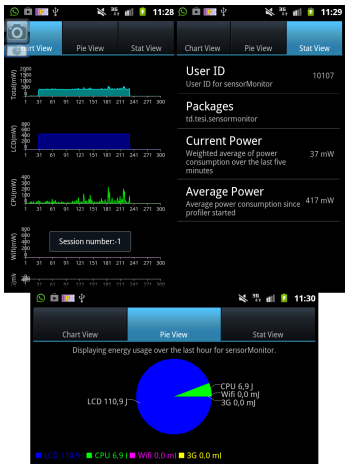

Figura 4.2: Prestazioni SensorManager, batteria e microfono

Inoltre, è stata fatta una terza analisi con l'applicazione in ascolto dei sensori relativi alle network. Qui il consumo della batteria è di 75.2 J; 69.7 J dal LCD, 1.6 dal WiFi e 3.9 J dalla CPU.

Infine,è stata fatta un'ultima analisi con l'applicazione in ascolto dei sensori relativi al LocationManager. Qui il consumo monitorato è stato di 56 J; ; 51.6 J dal LCD e 5.4 J dalla CPU.

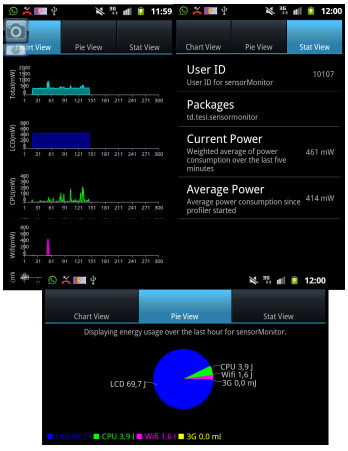

Figura 4.3: Prestazioni delle network

Figura 4.4: Prestazioni dei LocationProvider

Conclusioni

In conclusione, questa tesi di laurea mi sono occupato a sviluppare un sistema client-server in cui gli utenti condividono le informazioni dei sensori dei propri dispostivi Android su un server centrale. Il progetto sviluppato, lato client è costituito da un'applicaazione Android che permette agli utenti di avviare delle sessioni di lavoro in cui monitorando determinati sensori dei prori dispositivi inviano le informazioni e i valori di questi ultimi ad un server centrale. Il server a sua volta, costituito da alcune pagine php e da un database, elaborerà i dati ricevuti per poterli salvare correttamente sul database e alla fine delle varie operazioni di inserimento notificherà all'utente se queste hanno avuto successo o meno. Il sistema implementato è funzionante ed è stato testato su più dispositivi, ma a fronte della notevole quantità e differenza di sensori tra un dispositivo e l'altro non posso garantire con certezza il corretto funzionamento su tutti i dispositivi in circolazione.

Tuttavia il sistema potrebbe essere ancora ulteriormente sviluppato, poichè dai dati raccolti si possono costruire una serie di applicazioni, strumenti e servizi che possono facilitare la vita di tutti i giorni oppure fornire delle statistiche sull'utilizzo dei dispositivi oppure delle statistiche sui contesti urbani monitorati. Inoltre, anche la parte client potrebbe essere migliorata, sia per quanto riguarda l'estetica dell'applicazione che per come è stata implementata; ad esempio potrebbe essere implementato un servizio attraverso il quale l'applicazione possa funzionare anche in background, permettendo quindi agli utenti di fare altre operazioni mentre è attiva una sessione di monitoraggio.

Inoltre, a fronte del continuo sviluppo del software e dei dispositivi bi-

sognerebbe tenere aggiornati, il server, il database e l'applicazione sui nuovi sensori che vengono installati ed implementati nelle prossime release di Android.

Infine, l'applicazione andrebbe distribuita su larga scala per poter ottenere una visione realistica del contesto urbano e dell'interazione dei cittadini con esso. A fronte di questa distribuzione su larga scala si potrebbe pensare di distribuire il carico di lavoro del server su alcuni server slave,sia per quanto riguarda gli accessi alle pagine dei servizi che per quanto riguarda il database.

Figura 4.5: Wordl

Appendice A

Query per la creazione del database

Nella tabella seguente sono elencate le query, ordinate come all'interno dell'array, utilizzate nella creazione delle tabelle del database:

Tabella A.1: Query per la creazione delle tabelle

Query
CREATE TABLE SCANSESSION(Id_Scan INT NOT NULL AUTO_INCREMENT, Id_Device VARCHAR(255) NOT NULL, StartTime BIGINT NOT NULL, EndTime BIGINT, Label TEXT, PRIMARY KEY (Id_Scan, Id_Device, StartTime))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE BATTERY(Id_Scan INT(11) NOT NULL, Timestamp BIGINT, Status INT, Health INT, Plugged INT, Level INT, Present INT, Scale INT, Technology TEXT, Temperature INT, Voltage INT, PRIMARY KEY Timestamp(Id_Scan, Timestamp), FOREIGN KEY (Id_Scan) REFERENCES SCANSESSION (Id_Scan))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE MICROPHONE(Id_Scan INT(11) NOT NULL, Timestamp BIGINT, AudioMax INT, Amplitude FLOAT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE NETWORK(Id_Scan INT(11) NOT NULL, Name VARCHAR(10), Enable TINYINT(1), Accurancy INT, PowerRequirement INTEGER, MonetaryCost TINYINT(1), RequiresCell TINYINT(1), RequiresNetwork TINYINT(1), RequiresSatellite TINYINT(1), SupportsAltitude TINYINT(1), SupportsBearing TINYINT(1), SupportSpeed TINYINT(1), Timestamp BIGINT, Location_Accurancy FLOAT, Location_Altitude DOUBLE, Location_Bearing FLOAT, Location_Latitude DOUBLE, Location_Longitude DOUBLE, Location_Speed FLOAT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE PASSIVE(Id_Scan INT(11) NOT NULL, Enable TINYINT(1), Accurancy INT, PowerRequirement INTEGER, MonetaryCost TINYINT(1), RequiresCell TINYINT(1), RequiresNetwork TINYINT(1), RequiresSatellite TINYINT(1), SupportsAltitude TINYINT(1), SupportsBearing TINYINT(1), SupportSpeed TINYINT(1), Timestamp BIGINT, Location_Accurancy FLOAT, Location_Altitude DOUBLE, Location_Bearing FLOAT, Location_Latitude DOUBLE, Location_Longitude DOUBLE, Location_Speed FLOAT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE GPS(Id_Scan INT(11) NOT NULL , Name VARCHAR(10), Enable TINYINT(1), Accurancy INT, PowerRequirement INTEGER, MonetaryCost TINYINT(1), RequiresCell TINYINT(1), RequiresNetwork TINYINT(1), RequiresSatellite TINYINT(1), SupportsAltitude TINYINT(1), SupportsBearing TINYINT(1), SupportSpeed TINYINT(1), GpsStatus INT, Timestamp BIGINT, Location_Accurancy FLOAT, Location_Altitude DOUBLE, Location_Bearing FLOAT, Location_Latitude DOUBLE, Location_Longitude DOUBLE, Location_Speed FLOAT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE ACCELEROMETER(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE MAGNETICFIELD(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE GYROSCOPE(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE LIGHT(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE PRESSURE(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE PROXIMITY(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE GRAVITY(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE LINEARACCELERATION(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE ROTATIONVECTOR(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, X_axis FLOAT, Y_axis FLOAT, Z_axis FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE ORIENTATION(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Azimuth FLOAT, Pitch FLOAT, Roll FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE RELATIVEHUMIDITY(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE AMBIENTTEMPERATURE(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE TEMPERATURE(Id_Scan INT(11) NOT NULL, Name TEXT, Type INT, Vendor TEXT, Version INT, Power FLOAT, MinDelay INT, MaxRange FLOAT, Resolution FLOAT, Accurancy INT, Value FLOAT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE DUMMY(Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE ETHERNET(Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE MOBILE(Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE MOBILEDUN(Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE MOBILEHIPRI(Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE MOBILEMMS(Id_Scan INT(11) NOT NULL, Timestamp BIGINT, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE MOBILESUPL(Id_Scan INT(11) NOT NULL, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE BLUETOOTH(Id_Scan INT(11) NOT NULL, Name TEXT, Address TEXT, State INT, Enable TINYINT(1), Discovery TINYINT(1), ScanMode INT, Timestamp BIGINT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE BLUETOOTHDISCOVERYDEVICES(Id_Scan INT(11) NOT NULL, StartTime BIGINT, EndTime BIGINT, Address VARCHAR(255), BondState INT, Name TEXT, FOREIGN KEY (Id_Scan)REFERENCES BLUETOOTH (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, StartTime, Address))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE BLUETOOTHBONDEDDEVICE(Id_Scan INT(11) NOT NULL, Timestamp BIGINT, Address VARCHAR(255), BondState INT, Name TEXT, FOREIGN KEY (Id_Scan) REFERENCES BLUETOOTH (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY(Id_Scan, Timestamp, Address))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE WIFI(Id_Scan INT(11) NOT NULL, Timestamp BIGINT, Type INT, Subtype INT, ExtraInfo TEXT, Reason TEXT, State TEXT, Available TINYINT(1), Connected TINYINT(1), Connected_Connecting TINYINT(1), Failover TINYINT(1), Roaming TINYINT(1), NetworkTypeValid TINYINT(1), Dchp_Dns1 INT, Dchp_Dns2 INT, Dchp_Gateway INT, Dchp_IpAddress INT, Dchp_LeaseDuration INT, Dchp_Netmask INT, Dchp_ServerAddress INT, WifiInfo_BSSID TEXT, WifiInfo_HiddenSSID TINYINT(1), WifiInfo_SSID TEXT, WifiInfo_IpAddress INT, WifiInfo_LinkSpeed INT, WifiInfo_MacAddress TEXT, WifiInfo_NetworkId INT, WifiInfo_RSSI INT, WifiInfo_SupplicantState TEXT, FOREIGN KEY (Id_Scan)REFERENCES SCANSESSION (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Continued on next page

Tabella A.1 – *Continued from previous page*

Query
CREATE TABLE WIFICONFIGURATION(Id_Scan INT(11) NOT NULL, Timestamp BIGINT, AllowedAuthAlgorithms TEXT, AllowedGroupCiphers TEXT, AllowedKeyManagement TEXT, AllowedPairwiseCiphers TEXT, AllowedProtocols TEXT, BSSID TEXT, HiddenSSID TINYINT(1), NetworkId INT, PreSharedKey TEXT, Priority INT, Status INT, SSID TEXT, WEP_Key1 TEXT, WEP_Key2 TEXT, WEP_Key3 TEXT, WEP_Key4 TEXT, WepTxKeyIndex INT, FOREIGN KEY (Id_Scan, TimeStamp) REFERENCES WIFI(Id_Scan, Timestamp) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY (Id_Scan, Timestamp))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
CREATE TABLE WIFISCANRESULT(Id_Scan INT(11) NOT NULL, Timestamp BIGINT, BSSID VARCHAR(255), SSID TEXT, Capabilities TEXT, Frequency INT, Level INT, FOREIGN KEY (Id_Scan) REFERENCES WIFI (Id_Scan) ON DELETE CASCADE ON UPDATE CASCADE, PRIMARY KEY(Id_Scan, Timestamp, BSSID))ENGINE=InnoDB DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;

Appendice B

Casi implementati sul server

In questa appendice, vengono elencate le query utilizzate dai singoli casi implementati nel server. I casi, come detto nel terzo capitolo, implementati nello switch del file sensorMonitorServer.php permettono di effettuare operazioni specifiche per ogni categoria di sensori.

startSession

Caso in cui viene fatta partire una nuova sessione per quanto riguarda il monitoraggio dei dati di un dispositivo

INSERT INTO SCANSESSION(Id_Device, StartTime, Label) VALUES ("'. $json['IdDevice']."', ". $json['TimestampSend']."', "'. $json['MainLabel']."');

SELECT Id_Scan FROM SCANSESSION WHERE Id_Device="'. $json['IdDevice']."' AND StartTime=". $json['TimestampSend']." AND Label="'. $json['MainLabel']."';

stopSession

Caso in cui viene terminata una sessione del monitoraggio dei dati.

UPDATE SCANSESSION SET EndTime=". $json['TimestampSend']." WHERE Id_Scan=". $json['IdSession'].";"

Battery

Caso in cui vengono inseriti i dati relativi alla batteria

INSERT INTO BATTERY(Id_Scan, Timestamp, Status, Health, Plugged, Level, Present, Scale, Technology, Temperature, Voltage) VALUES (". $json['IdSession']., ". $json['TimestampSend']., ". $json['status']., ". $json['health']., ". $json['plugged']., ". $json['level']., ". $json['present']., ". $json['scale']., "'. $json['technology']."', ". $json['temperature']., ". $json['voltage'].");

Microphone

Caso in cui vengono inseriti i dati relativi al MICROFONO

INSERT INTO MICROPHONE(Id_Scan, Timestamp, AudioMax, Amplitude)VALUES (". $json['IdSession']., ". $json['TimestampSend']., ". $json['audioMax']., ". $json['amplitude'].")

Sensor

Caso in cui vengono inseriti i dati relativi ai sensori del SensorManager. Al suo interno contine un'altro switch che, a seconda del tipo del sensore, va ad eseguire la relativa query di inserimento. INSERT INTO ACCELEROMETER(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)VALUES (". $json['IdSession']., "'. $json['name']."', ". $json['type']., "'. $json['vendor']."', ". $json['version']., ". $json['power']., ". $json['minDelay']., ". $json['maxRange']., ". $json['resolution']., ". $json['accurancy']., $Xvalue, $Yvalue, $Zvalue, ". $json['timestamp'].");

INSERT INTO MAGNETICFIELD(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)VALUES (". $json['IdSession']., "'. $json['name']."', ". $json['type']., "'. $json['vendor']."', ". $json['version']., ". $json['power'].",

". $json['minDelay'].", ". $json['maxRange'].", ". $json['resolution']."', ". $json['accurancy']."', $Xvalue, $Yvalue, $Zvalue, ". $json['timestamp']."');

INSERT INTO ORIENTATION(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Azimuth, Pitch, Roll, Timestamp)VALUES (". $json['IdSession']."', '". $json['name']."', ". $json['type']."', '". $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $azimuth, $pitch, $roll, ". $json['timestamp']."');

INSERT INTO GYROSCOPE(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)VALUES (". $json['IdSession']."', '". $json['name']."', ". $json['type']."', '". $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $Xvalue, $Yvalue, $Zvalue, ". $json['timestamp']."');

INSERT INTO LIGHT(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)VALUES (". $json['IdSession']."', '". $json['name']."', ". $json['type']."', '". $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $value, ". $json['timestamp']."');

INSERT INTO PRESSURE(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)VALUES (". $json['IdSession']."', '". $json['name']."', ". $json['type']."', '". $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $value, ". $json['timestamp']."');

INSERT INTO TEMPERATURE(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)VALUES (". $json['IdSession']."', '". $json['name']."', ". $json['type']."', '". $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."',

$value, ". $json['timestamp']."');

INSERT INTO PROXIMITY(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp) VALUES (". $json['IdSession']."', "'. $json['name']."', ". $json['type']."', "'. $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $value, ". $json['timestamp']."');

INSERT INTO GRAVITY(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)VALUES (". $json['IdSession']."', "'. $json['name']."', ". $json['type']."', "'. $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $Xvalue, $Yvalue, $Zvalue, ". $json['timestamp']."');

INSERT INTO LINEARACCELERATION(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)VALUES (". $json['IdSession']."', "'. $json['name']."', ". $json['type']."', "'. $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $Xvalue, $Yvalue, $Zvalue, ". $json['timestamp']."');

INSERT INTO ROTATIONVECTOR(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, X_axis, Y_axis, Z_axis, Timestamp)VALUES (". $json['IdSession']."', "'. $json['name']."', ". $json['type']."', "'. $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $Xvalue, $Yvalue, $Zvalue, ". $json['timestamp']."');

INSERT INTO RELATIVEHUMIDITY(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)VALUES (". $json['IdSession']."', "'. $json['name']."', ". $json['type']."', "'. $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $value, ". $json['timestamp']."');

INSERT INTO AMBIENTTEMPERATURE(Id_Scan, Name, Type, Vendor, Version, Power, MinDelay, MaxRange, Resolution, Accurancy, Value, Timestamp)VALUES (". $json['IdSession']."', "'. $json['name']."', ". $json['type']."', "'. $json['vendor']."', ". $json['version']."', ". $json['power']."', ". $json['minDelay']."', ". $json['maxRange']."', ". $json['resolution']."', ". $json['accurancy']."', $value, ". $json['timestamp']."');

LocationProvider

Caso in cui vengono inseriti i dati relativi ai LocationProvider del LocationManager. Al suo interno contine un'altro switch che, a seconda del nome, va ad eseguire la relativa query di inserimento. Inoltre i valori booleani presenti nel json, sono stati trasformati in 0 o 1 per poter essere salvati sul database.

INSERT INTO NETWORK(Id_Scan, Name, Enable, Accurancy, PowerRequirement, MonetaryCost, RequiresCell, RequiresNetwork, RequiresSatellite, SupportsAltitude, SupportsBearing, SupportSpeed, Timestamp, Location_Accurancy, Location_Altitude, Location_Bearing, Location_Latitude, Location_Longitude, Location_Speed)VALUES (". $json['IdSession']."', "'. $json['name']."', $enable, ". $json['accurancy']."', ". $json['powerRequirement']."', $monetaryCost, $requiresCell, $requiresNetwork, $requiresSatellite, $supportsAltitude, $supportsBearing, $supportSpeed, ". $json['TimestampSend']."', ". $location['accurancy']."', ". $location['altitude']."', ". $location['bearing']."', ". $location['latitude']."', ". $location['longitude']."', ". $location['speed']."');

INSERT INTO NETWORK(Id_Scan, Name, Enable, Accurancy, PowerRequirement, MonetaryCost, RequiresCell, RequiresNetwork, RequiresSatellite, SupportsAltitude, SupportsBearing, SupportSpeed, Timestamp)VALUES (". $json['IdSession']."', "'. $json['name']."', $enable, ". $json['accurancy']."', ". $json['powerRequirement']."', $monetaryCost, $requiresCell, $requiresNetwork, $requiresSatellite, $supportsAltitude, $supportsBearing, $supportSpeed, ". $json['TimestampSend']."');

INSERT INTO PASSIVE(Id_Scan, Name, Enable, Accurancy, PowerRequirement, MonetaryCost, RequiresCell, RequiresNetwork, RequiresSatellite, SupportsAltitude, SupportsBearing, SupportSpeed, Timestamp, Location_Accurancy, Location_Altitude, Location_Bearing, Location_Latitude, Location_Longitude, Location_Speed)VALUES (". $json['IdSession']."', "'. $json['name']."', $enable, ". $json['accurancy']."', ". $json['powerRequirement']."', $monetaryCost, $requiresCell, $requiresNetwork, $requiresSatellite, $supportsAltitude, $supportsBearing, $supportSpeed, ". $json['TimestampSend']."', ". $location['accurancy']."', ". $location['altitude']."', ". $location['bearing']."', ". $location['latitude']."', ". $location['longitude']."', ". $location['speed']."');

INSERT INTO PASSIVE(Id_Scan, Name, Enable, Accurancy, PowerRequirement, MonetaryCost, RequiresCell, RequiresNetwork, RequiresSatellite, SupportsAltitude, SupportsBearing, SupportSpeed, Timestamp)VALUES (". $json['IdSession']."', "'. $json['name']."', $enable, ". $json['accurancy']."', ". $json['powerRequirement']."', $monetaryCost, $requiresCell, $requiresNetwork, $requiresSatellite, $supportsAltitude, $supportsBearing, $supportSpeed, ". $json['TimestampSend']."');

INSERT INTO GPS (Id_Scan, Name, Enable, Accurancy, PowerRequirement, MonetaryCost, RequiresCell, RequiresNetwork, RequiresSatellite, SupportsAltitude, SupportsBearing, SupportSpeed, GpsStatus, Timestamp, Location_Accurancy, Location_Altitude, Location_Bearing, Location_Latitude, Location_Longitude, Location_Speed)VALUES (". $json['IdSession']."', "'. $json['name']."', $enable, ". $json['accurancy']."', ". $json['powerRequirement']."', $monetaryCost, $requiresCell, $requiresNetwork, $requiresSatellite, $supportsAltitude, $supportsBearing, $supportSpeed, ". $json['gpsStatus']."', ". $json['TimestampSend']."', ". $location['accurancy']."', ". $location['altitude']."', ". $location['bearing']."', ". $location['latitude']."', ". $location['longitude']."', ". $location['speed']."');

INSERT INTO GPS(Id_Scan, Name, Enable, Accurancy, PowerRequirement, MonetaryCost, RequiresCell, RequiresNetwork, RequiresSatellite, SupportsAltitude, SupportsBearing, SupportSpeed, GpsStatus, Timestamp)

VALUES (". $json['IdSession']."', "'. $json['name']."', $enable, ". $json['accurancy']."', ". $json['powerRequirement']."', $monetaryCost, $requiresCell, $requiresNetwork, $requiresSatellite, $supportsAltitude, $supportsBearing, $supportSpeed, ". $json['gpsStatus']."', ". $json['TimestampSend']."');

NetworkInfo

Caso in cui vengono inseriti i dati relativi alle Network del ConnectivityManager. Al suo interno contine un'altro switch che, a seconda del tipo, va ad eseguire la relativa query di inserimento.

INSERT INTO MOBILE(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)VALUES (". $json['IdSession']."', ". $json['type']."', ". $json['subtype']."', "'. $json['extraInfo']."', "'. $json['reason']."', "'. $json['state']."', $available, $connected, $connected_connecting, $failover, $roaming, $networkTypeValid, ". $json['TimestampSend']."');

INSERT INTO WIFI(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp, Dchp_Dns1, Dchp_Dns2, Dchp_Gateway, Dchp_IpAddress, Dchp_LeaseDuration, Dchp_Netmask, Dchp_ServerAddress, WifiInfo_BSSID, WifiInfo_HiddenSSID, WifiInfo_SSID, WifiInfo_IpAddress, WifiInfo_LinkSpeed, WifiInfo_MacAddress, WifiInfo_NetworkId, WifiInfo_RSSI, WifiInfo_SupplicantState)VALUES (". $json['IdSession']."', ". $json['type']."', ". $json['subtype']."', "'. $json['extraInfo']."', "'. $json['reason']."', "'. $json['state']."', $available, $connected, $connected_connecting, $failover, $roaming, $networkTypeValid, ". $json['TimestampSend']."', ". $dchp['dns1']."', ". $dchp['dns2']."', ". $dchp['gateway']."', ". $dchp['ipAddress']."', ". $dchp['leaseDuration']."', ". $dchp['netmask']."', ". $dchp['serverAddress']."', "'. $json['wifiInfoBSSID']."', $hiddenSSID, "'. $json['wifiInfoSSID']."', ". $json['wifiInfoIpAddress']."', ". $json['wifiInfoLinkSpeed']."', "'. $json['wifiInfoMacAddress']."', ". $json['wifiInfoNetworkId']."', ". $json['wifiInfoRSSI']."', "'. $json['wifiInfoSupplicantState']."');

INSERT INTO MOBILEMMS(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)VALUES (". $json['IdSession']."', ". $json['type']."', ". $json['subtype']."', "'. $json['extraInfo']."', "'. $json['reason']."', "'. $json['state']."', $available, $connected, $connected_connecting, $failover, $roaming, $networkTypeValid, ". $json['TimestampSend']."');

INSERT INTO MOBILESUPL(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)VALUES (". $json['IdSession']."', ". $json['type']."', ". $json['subtype']."', "'. $json['extraInfo']."', "'. $json['reason']."', "'. $json['state']."', $available, $connected, $connected_connecting, $failover, $roaming, $networkTypeValid, ". $json['TimestampSend']."');

INSERT INTO MOBILEDUN(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)VALUES (". $json['IdSession']."', ". $json['type']."', ". $json['subtype']."', "'. $json['extraInfo']."', "'. $json['reason']."', "'. $json['state']."', $available, $connected, $connected_connecting, $failover, $roaming, $networkTypeValid, ". $json['TimestampSend']."');

INSERT INTO MOBILEHIPRI(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)VALUES (". $json['IdSession']."', ". $json['type']."', ". $json['subtype']."', "'. $json['extraInfo']."', "'. $json['reason']."', "'. $json['state']."', $available, $connected, $connected_connecting, $failover, $roaming, $networkTypeValid, ". $json['TimestampSend']."');

INSERT INTO DUMMY(Id_Scan, Type, Subtype, ExtraInfo, Reason, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)VALUES (". $json['IdSession']."', ". $json['type']."', ". $json['subtype']."', "'. $json['extraInfo']."', "'. $json['reason']."', "'. $json['state']."', $available, $connected, $connected_connecting, $failover, $roaming, $networkTypeValid, ". $json['TimestampSend']."');

INSERT INTO ETHERNET(Id_Scan, Type, Subtype, ExtraInfo, Rea-

son, State, Available, Connected, Connected_Connecting, Failover, Roaming, NetworkTypeValid, Timestamp)VALUES (". $json['IdSession']."', ". $json['type']."', ". $json['subtype']."', "'. $json['extraInfo']."', "'. $json['reason']."', "'. $json['state']."', $available, $connected, $connected_connecting, $failover, $roaming, $networkTypeValid, ". $json['TimestampSend']."');

INSERT INTO WIFICONFIGURATION(Id_Scan, Timestamp, AllowedAuthAlgorithms, AllowedGroupChipers, AllowedKeyManagement, AllowedPairwiseChipers, AllowedProtocols, BSSID, HiddenSSID, NetworkId, PreSharedKey, Priority, Status, SSID, WEP_Key1, WEP_Key2, WEP_Key3, WEP_Key4, , WepTxKeyIndex)VALUES (". $json['IdSession']."', ". $json['TimestampSend']."', "'. $localWifiConfList['allowedAuthAlgorithms']."', "'. $localWifiConfList['allowedGroupChipers']."', "'. $localWifiConfList['allowedKeyManagemen']."', "'. $localWifiConfList['allowedPairwiseChipers']."', "'. $localWifiConfList['allowedProtocols']."', "'. $localWifiConfList['BSSID']."', $hiddenSSID, ". $localWifiConfList['networkId']."', "'. $localWifiConfList['preSharedKey']."', ". $localWifiConfList['priority']."', ". $localWifiConfList['status']."', "'. $localWifiConfList['SSID']."', "'. $$wepKeys[0]."', "'. $wepKeys[1]."', "'. $wepKeys[2]."', "'. $wepKeys[3]."', ". $localWifiConfList['wepTxKeyIndex']."');

WifiScanResult

Caso per l'inserimento dei dati delle reti Wi-Fi scansionate

INSERT INTO WIFISCANRESULT(Id_Scan, Timestamp, BSSID, SSID, Capabilities, Frequency, Level)VALUES (". $json['IdSession']."', ". $json['TimestampSend']."', "'. $localScanWifiList['BSSID']."', "'. $localScanWifiList['SSID']."', "'. $localScanWifiList['capabilities']."', ". $localScanWifiList['frequency']."', ". $localScanWifiList['level']."');

Bluetooth

Caso per l'inserimento dei dati relativo al bluetooth del dispositivo

INSERT INTO BLUETOOTH(Id_Scan, Name, Address, State, Enable, Discovery, ScanMode, Timestamp)VALUES (". $json['IdSession'].", "'. $json['name']."', "'. $json['address']."', ". $json['state'].", $enable, $discovery, ". $json['scanMode']., ". $json['TimestampSend']."');

BluetoothDiscovery

Caso per l'inserimento dei dati dei dispositivi bluetooth scansionati

INSERT INTO BLUETOOTHDISCOVERYDEVICES(Id_Scan, StartTime, EndTime, Address, BondState, Name)VALUES (". $json['IdSession']., ". $json['discoveryStartTime']., ". $json['discoveryEndTime']., "'. $localBltDiscovDev['address']."', ". $localBltDiscovDev['bondState']., "'. $localBltDiscovDev['name']."');

Bibliografia

[1] Andrew T. Campbell, Shane B. Eisemann, Nicholas D. Lane, Emiliano Miluzzo, Ronald A.Peterson, People-Centric Urban Sensing.

[2] Salil S. Kanhere, Participatory Sensing: Crowdsourcing Data from Mobile SmartPhones in Urban Spaces, in International Conference on Mobile Data Management, 2011 12th

[3] Franco Zambonelli, Pervasive Urban Crowdsourcing: Vision and Challenges.

[4] Luca Bedogni, Marco Di Felice, Luciano Bononi, By Train or By Car? Detecting the User's Motion Type through Smartphone Sensors Data, 2012.

[5] http://it.wikipedia.org/wiki/Crowdsourcing

[6] http://developer.android.com/guide/topics/sensors/sensors_overview.html

[7] http://developer.android.com/guide/topics/sensors/sensors_motion.html

[8] http://developer.android.com/guide/topics/sensors/sensors_position.html

[9] http://developer.android.com/guide/topics/sensors/sensors_environment.html

[10] Howe, Jeff (June 2, 2006). "Crowdsourcing: A Definition". Crowdsourcing Blog. Retrieved January 2, 2013.

[11] S. Reddy, A. Parker, J. Hyman, J. Burke, D. Estin and M. Hansen, "Image browsing, Processing and Clustering for Parecipatory Sensing:

Lessons from a DietSense Prototype" in Proceedings of the Workshop on Embedeed Networked Sensors(EmNetS), Cork, Ireland, June 2007

[12] M.Mun, S. Reddy, et al. "PEIR, the Personal Enviromental Impact Report, as a Platform for partecipatory Sensing System Research", in Proceedings of ACM MobiSys, Krakow, Poland, June 2009.

[13] S. Eisenman, E. Miluzzo, N. Lane, R. Peterson, G. Ahn and A. Campbell, "The Bikenet Mobile Sensing System for Cyclist Experience Mapping", in Proceedings of ACM SenSys, Sydney, Australia November 2007.

[14] Y. Dong, S. S. Kanhere, C. T. Chou and N. Bulusu, "Automatic Collection of Fuel Prices from a Network of Mobile Cameras", in Proceedings of IEEE DCOSS 2008, Santorini, Greece, June 2008.

[15] J. Carrapetta, N. Youdale, A. Chow and V. Sivaraman, "Haze Watch Project", Online: http://www.pollution.ee.unsw.edu.au.

[16] R. Rana, C.T. Chou, S. Kanhere, N. Bulusu and W. Hu, "Ear-Phone: An End-to-End Participatory Urban Noise Mapping System", in Proceedings of ACM/IEEE IPSN, Stockholm, Sweden, April 2010.

[17] P. Mohan, V. Padmanabhan, R. Ramjee, "Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones", in Proceedings of ACM SenSys, Raleigh, NC, USA, November 2008.

Ringraziamenti

Desidero ringraziare tutti coloro che mi hanno aiutato nella stesura della tesi con suggerimenti, critiche ed osservazioni: a loro va la mia gratitudine, anche se a me spetta la responsabilità per ogni errore contenuto in questo elaborato e nel progetto sviluppato.

Ringrazio anzitutto il professor Marco Di Felice, Relatore, che mi ha permesso, attraverso la sua disponibilità nel propormi il progetto e nel seguirmi durante la fase di sviluppo, di elaborare questa tesi di laurea.

Un ringraziamento particolare va ai colleghi ed agli amici del gruppo Sayan che mi hanno incoraggiato e che hanno speso parte del proprio tempo per leggere, discutere e testare con me gli sviluppi di tale progetto.