Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Triennale

Un'applicazione mobile per il monitoraggio della qualità del servizio di trasporto ferroviario

Tesi di Laurea in Architettura degli elaboratori

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Luca Rinaldi

Sessione II
Anno Accademico 2012/2013

Introduzione

La scarsa qualità del servizio di trasporto ferroviario, in Italia, è sotto gli occhi di tutti i viaggiatori. Sono state fondate vere e proprie associazioni di pendolari per protestare contro il servizio offerto ai cittadini. Queste lamentele riguardano soprattutto i continui ritardi (e soppressioni), il sovraffollamento e la scarsa pulizia dei treni regionali.

Sono un viaggiatore pendolare da qualche anno e ho un'esperienza diretta sull'argomento. Mi è capitato spesso di vedere treni semi-vuoti alternati a treni sovraffollati, in cui era difficile persino salire. Mi è anche capitato di assistere a ritardi annunciati solo all'ultimo minuto, costringendo i viaggiatori a rimanere in attesa sulla banchina. Questi disagi possono convincere i viaggiatori ad utilizzare altri mezzi, spesso meno economici e più inquinanti. In questa tesi cerco di utilizzare l'informatica per realizzare uno strumento di supporto al viaggiatore ferroviario.

Il progetto che illustro in queste pagine è una applicazione mobile per Android, della quale descrivo: gli obiettivi che cerca di soddisfare (cap. 1), gli strumenti utilizzati per realizzarla (cap. 2), la sua progettazione (cap. 3), la sua implementazione (cap. 4), la guida per utilizzarla e le prove sperimentali effettuate (cap. 5).

Indice

Elenco delle figure

Capitolo 1

Scenario e obiettivi

Scenario

CUSTOMER SATISFACTION DELLA PERMANENZA A BORDO
(percentuale di passeggeri soddisfatti)

Permanenza a bordo	Media lunga percorrenza	Trasporto regionale
Comfort	83	66
Pulizia	76	46
Puntualità	83	64
Informazioni a bordo	87	64
Personale	95	81

Figura 1.1: Sondaggi Trenitalia sulla qualità della permanenza a bordo

Analizzando il sondaggio[1] in figura 1.1 si può notare come la soddisfazione dei viaggiatori ferroviari verso il servizio offerto sia piuttosto scadente, soprattutto per quanto riguarda il trasporto regionale (e quindi i treni che utilizzano prevalentemente i *pendolari*). Questo si può anche osservare dalla presenza di numerosi comitati locali e associazioni di pendolari (i pendolari in Italia sono circa 2 milioni), nati per protestare contro la scarsa qualità del servizio ferroviario italiano (bib. [2]).

[1] Vedi bibliografia [1].

Questa tesi è stata fatta per cercare di migliorare la qualità del servizio ricevuto, offrendo ai viaggiatori dei dati in tempo reale sulla situazione del treno che vorrebbero utilizzare. Prima di avanzare è meglio offrire una panoramica sui principali problemi riscontrati dai viaggiatori ferroviari:

- I treni sono spesso in ritardo e questo, nella maggior parte dei casi, viene segnalato solo pochi minuti prima dell'orario ufficiale di arrivo del treno. Un treno può essere in ritardo per i seguenti motivi:
  - Scarsa organizzazione del personale;
  - Condizioni meteorologiche particolari (neve, freddo, grandine, ...);
  - Presenza di altri treni in ritardo sulla linea;
  - Incidente sulla linea;
  - Malfunzionamento del treno;
  - Attesa eccessiva dovuta alla discesa e alla salita dei viaggiatori sul treno.

  In alcuni casi, un eccessivo ritardo del treno può causare la sua soppressione. Naturalmente il ritardo improvviso del treno causa non pochi disagi ai pendolari, che sono costretti ad arrivare in ritardo ad appuntamenti o esami.

- La scarsa pulizia della carrozza e dei bagni, oltre che al fattore igienico, può dare fastidio a molti viaggiatori (soprattutto quelli occasionali) e far preferire l'utilizzo di altri mezzi (come l'automobile) per arrivare a destinazione.

- Il sovraffollamento dei treni è molto frequente in certe fasce orarie, soprattutto nel fine ed inizio settimana. Non solo il viaggiatore non trova il posto a sedere, ma in certi casi può fare addirittura fatica ad entrare nel treno (una delle cause del ritardo del treno) per via dell'eccessivo numero di persone presenti a bordo. Viaggiare in queste condizioni non è solo poco gradevole, ma è anche pericoloso. Il sovraffollamento

è dovuto soprattutto alla chiusura di certe carrozze del treno e ad un numero non previsto di passeggeri.

Obiettivo principale

L'obiettivo principale di questa tesi è creare una applicazione mobile per Android che, attraverso l'utilizzo dei sensori presenti nel dispositivo, valuti automaticamente la qualità del servizio offerto al viaggiatore ferroviario.

Obiettivo secondario

L'obiettivo secondario è offrire agli utenti la possibilità di valutare personalmente e manualmente la qualità il viaggio. Attraverso questa funzionalità è possibile offrire agli utenti un rapporto sulle qualità non valutabili automaticamente, come ad esempio la pulizia del treno.

Utilità

La qualità del servizio viene trattata sotto diversi punti di vista:

Comfort del viaggiatore Durante la sua permanenza sul treno, specialmente per quanto riguarda i viaggi di lunga percorrenza, il viaggiatore si dovrebbe sentire a proprio agio. Il comfort del cliente è più alto se trova un *posto a sedere* e se l'ambiente è *pulito*.

Puntualità Il viaggiatore preferisce scegliere un treno *puntuale* per il suo viaggio. Per quanto riguarda i treni regionali, i ritardi vengono spesso segnalati all'ultimo minuto e il pendolare non ha alcun modo di saperlo in anticipo, se non mettendosi in contatto telefonico con chi è già sul treno. Se un treno rimane fermo a lungo o ha una *velocità media* inferiore al solito, allora è probabile che questo sia la causa di un eventuale ritardo del treno o che sia la conseguenza di un suo malfunzionamento.

Numero di persone a bordo Un grande problema del viaggiatore pendolare è il sovraffollamento dei treni. Può essere utile alla società ferroviaria capire in dettaglio in quali giorni il *numero delle persone* a bordo del treno supera notevolmente la sua effettiva capacità. In questo modo la società può decidere eventualmente di aumentare il numero di carrozze del treno o di sostituirlo con un altro solo in quegli orari.

I dati ottenuti dal monitoraggio hanno un duplice scopo:

1. Permettere al viaggiatore di essere informato sulla qualità dei treni disponibili per effettuare un determinato viaggio, in modo da poter scegliere quello che offre un servizio di qualità maggiore.

2. Incentivare la società ferroviaria all'aumento della qualità del servizio offerto, tenendo anche conto dei giudizi espressi dai propri clienti.

Aumentare la qualità del servizio di trasporto ferroviario potrebbe inoltre spingere il cittadino all'utilizzo dei mezzi pubblici, riducendo di conseguenza il traffico e l'inquinamento dovuti all'utilizzo di automobili.

Capitolo 2

Strumenti di sviluppo

In questo capitolo vengono trattati in dettaglio gli strumenti software utilizzati per lo sviluppo dell'applicazione.

Per la realizzazione dell'applicazione ho utilizzato:

- Eclipse Juno (versione 4.2);
- Plugin ADT (versione 22.0.5) per Eclipse;
- Apache HTTP Server (versione 2.22).
- Modulo PHP5 per Apache, con estensione PDO.
- MySQL (Oracle).

La tesi è si può dividere in due principali sezioni. La prima sezione è la parte client, che è quella relativa all'applicazione per Android (vedi par. 2.1); la seconda sezione è quella relativa al server, ovvero la gestione del database (vedi par. 2.2). La rete creata è composta da un server centrale e da numerosi client (smartphone, tablet, ...), che comunicano con esso.

2.1 Sviluppo lato client

In questo paragrafo vengono analizzati i principali strumenti di sviluppo utilizzati a lato client.

2.1.1 Android

Android è un sistema operativo di Google per dispositivi mobili. Si distingue dagli altri per il fatto di essere *open source* (Licenza Apache[1]) e di basarsi sul kernel 2.6 di Linux (kernel 3.x da Android 4.0 in poi).

Ho scelto di sviluppare l'applicazione su Android per puntare ad una fetta di utenti più alta: infatti, come si può notare dalla figura 2.1, in Italia (e anche nella maggior parte degli altri Paesi) le vendite di smartphone Android sono in continua crescita, mentre la concorrenza subisce dei rallentamenti.

Italy	3 m/e June 2012	3 m/e June 2013	% pt. Change
iOS	16.7	16.4	-0.3
Android	54.6	70.7	16.1
BlackBerry	5.5	2.5	-3.0
Symbian	12.3	2.3	-10.0
Windows	8.3	7.8	-0.5
Other	2.5	0.4	-2.1

Figura 2.1: Vendite degli smartphone in Italia nel mese di giugno (bib. [4])

Programmare in Android

Lo strumento principale che si utilizza è *Eclipse* , un ambiente di sviluppo integrato (IDE[2]) utilizzato per lo sviluppo di software con diversi linguaggi di programmazione.

[1]La Licenza Apache consente agli utenti di distribuire e modificare il software, utilizzandolo per ogni possibile scopo.

[2]Integrated Development Environment

Eclipse è incentrato sull'utilizzo di *plugin* . Quello relativo ad Android si chiama ADT (Android Developer Tools, bib. [3]) e offre funzionalità in più rispetto a quelle normalmente offerte da Eclipse, che permettono il test e lo sviluppo di applicazioni Android. Tra queste troviamo:

- IDE Java completo, che comprende:
  - Template di base per creare progetti e componenti Android;
  - Editor XML migliorato e navigazione integrata tra JAVA e le risorse XML;
- Possibilità di modificare l'interfaccia grafica attraverso il semplice "drag and drop";
- Possibilità di emulare un dispositivo virtuale (personalizzabile) per eseguire il test dell'applicazione;
- Possibilità di eseguire il test dell'applicazione su un dispositivo reale, attraverso un cavo USB.

Oltre al plugin ADT è necessario installare anche *Android SDK* , ovvero le librerie API e gli strumenti necessari per lo sviluppo di applicazioni Android.

Per iniziare a scrivere una applicazione Android è necessario conoscere le basi di Java e XML. Una applicazione Android è infatti formata principalmente da due elementi: il codice e le risorse. Il codice è scritto in Java e le risorse[3] sono composte nella maggior parte dei casi da file XML.

La divisione tra codice e risorse è presente per tre motivi: il primo motivo è separare il *come* dal *cosa*, ovvero separare la presentazione dei dati (layout) dalla loro effettiva gestione.

Il secondo motivo è sicuramente più funzionale: dare la possibilità di scegliere tra diverse risorse in base alle configurazioni del dispositivo in cui è installata l'applicazione. Nell'applicazione che ho sviluppato, ad esempio,

[3]Risorse: tutto ciò che non è codice.

sono presenti due file *string.xml*, uno contenente le stringhe in inglese e l'altro contenente le stringhe in italiano; verrà selezionato automaticamente (a runtime) il file giusto in base alla lingua impostata nel dispositivo (in caso di altre lingue selezionate verrà selezionata di default quella inglese).

Il terzo e ultimo motivo della divisione tra codice e sorgente è il fatto che la ricompilazione del codice viene fatta solo quando questa risulta strettamente necessaria. Se per esempio volessi aggiungere una terza lingua alla mia applicazione, potrei creare un nuovo file *string.xml* e metterlo nel percorso */res/values-xx/* dell'apk [4] , dove *xx* è il codice di due lettere corrispondente al linguaggio.

Altri esempi di risorse della la mia applicazione sono i dati relativi agli orari dei treni e i dati relativi alle coordinate delle tratte ferroviare, dei quali parleremo nel capitolo 4.2.1.

2.2 Sviluppo lato server

In questo paragrafo vengono analizzati i principali strumenti di sviluppo utilizzati per il lato server.

Per gestire e memorizzare i dati che ogni client invia al server, faccio uso di *MySQL* . Oracle MySQL è un RDBMS (Relational Database Management System) che comprende un client con interfaccia a riga di comando e un server, che permette l'interrogazione del database da parte di altre applicazioni. Nel paragrafo 4.1.1 è spiegato in dettaglio l'utilizzo di MySQL nello sviluppo del mio progetto.

Un altro strumento che utilizzo è *Apache HTTP Server* con il modulo PHP5 . Apache è un web server in grado di gestire le richieste HTTP provenienti dalla rete ed in particolare la mia tesi gestisce le richieste POST tra applicazione e server. Al modulo PHP ho aggiunto inoltre l'estensione PDO (PHP Data Objects) che mi offre un'unica interfaccia per l'accesso alle basi di dati. In questo modo non sono comunque vincolato all'utilizzo di MySQL

[4] Android Package. Può essere visto come il prodotto finale del processo di sviluppo.

per la gestione del database, potrei infatti decidere di cambiare DBMS senza comunque modificare gli script PHP.

Grazie alla combinazione di questi strumenti riesco a creare la rete in figura 2.2, dove il senso delle frecce corrisponde al senso delle richieste.

Figura 2.2: Diagramma del server

L'accesso diretto al DBMS diventa dunque disponibile solo agli script PHP sul server Apache, nei quali sono presenti i controlli per evitare eventuali SQL Injection.

Capitolo 3

Progettazione

In questo capitolo è illustrata la fase di progettazione dell'applicazione. Si parte dunque dall'idea iniziale fino ad arrivare ai dettagli più tecnici.

3.1 Idea iniziale

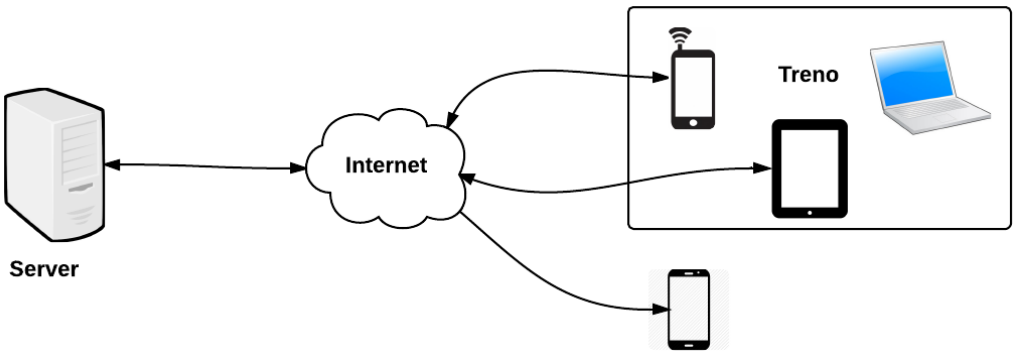

Figura 3.1: Schema della rete globale

Analizziamo la figura 3.1: a sinistra è presente un server (PC) che si occupa della gestione dei dati; a destra sono presenti i client, ovvero smartphone, tablet e altri dispositivi mobili con il sistema operativo Android; sul treno possono esserci anche altri dispositivi Bluetooth, comunque utili per il nostro scopo. Il verso delle frecce indica chi può mandare e chi può ricevere

i dati rilevati *automaticamente*. Come si può notare tutti i dispositivi mobili possono ricevere dati, ma solo quelli sul treno possono monitorarli e inviarli al server.

Ecco le prime domande che mi sono posto:

1. Quali strumenti ho a disposizione?
2. Quali tra questi possono essere utili al mio scopo?
3. Quali dati riesco effettivamente a monitorare?
4. Come capisco se un dispositivo è su un treno?

La risposta alla domanda numero 1 può essere riassunta con "non si sa". Non so su quale dispositivo l'utente installerà l'applicazione e ogni dispositivo mobile può anche non avere un determinato strumento utile al mio scopo. Devo dunque costruire un'applicazione che si adatti al dispositivo sul quale è installata, utilizzando di conseguenza solo gli strumenti effettivamente disponibili. Quindi mi devo chiedere: "quali strumenti *posso* avere a disposizione?".

Per rispondere a questa domanda ho realizzato una applicazione di prova che visualizza sullo schermo una lista di tutti i possibili strumenti e sensori disponibili all'interno del dispositivo. Provando l'applicazione su più dispositivi di recente uscita ho rilevato i seguenti strumenti: Bluetooth, GPS, sensore di gravità, giroscopio, accelerometro, sensore di rotazione, sensore di luminosità, sensore di prossimità e sensore di orientamento.

Passo dunque a rispondere alle domande numero 2 e 3: uno strumento che certamente mi può essere utile è il Bluetooth, che mi consente di scansionare l'ambiente circostante in cerca di altri dispositivi e fare dunque una stima sul numero di persone presenti nella carrozza. Tra i sensori disponibili posso invece utilizzare il sensore di prossimità e l'accelerometro: combinandoli si ottiene una stima sulla posizione del viaggiatore (in piedi o seduto).

Un altro strumento particolarmente utile è il ricevitore GPS , che mi calcola la velocità del treno e la posizione in coordinate. Utilizzando il GPS

riesco a rispondere anche alla domanda numero 4. Considerando il fatto che non posso inviare dati al server se non sono sicuro che questi dati siano stati presi sul treno, allora il GPS diventa un prerequisito per avere la possibilità di monitorare automaticamente l'ambiente circostante.

I dettagli più approfonditi sull'utilizzo degli strumenti del dispositivo all'interno dell'applicazione sono al paragrafo 3.3.

3.2 Caratteristiche

Requisiti e caratteristiche dell'applicazione:

- Il livello di API minimo richiesto dall'applicazione è il 16, che corrisponde ad Android 4.1 Jelly Bean.
- L'applicazione utilizza solamente i sensori effettivamente disponibili nel dispositivo. Nel caso questi sensori non dovessero essere disponibili, l'applicazione può essere comunque utilizzata per consultare i dati degli altri utenti.
- L'applicazione supporta sia la lingua inglese (di default) che la lingua italiana.

Permessi richiesti:

L'applicazione, prima di essere installata nel dispositivo, richiede i seguenti permessi:

- Accesso alla rete internet;
- Accesso al Bluetooth;
- Accesso alla lettura dei dati proveniente dai sensori all'interno del dispositivo;
- Accesso alla lettura dello stato del dispositivo;
- Avvio del service al termine del processo di boot del dispositivo.

3.3 Funzionamento

L'applicazione si basa sul rapporto tra i client (applicazioni Android) e il server (database). I dati devono essere raccolti in modo automatico, senza l'intervento dell'utente e per fare questo l'applicazione si divide in 2 parti: ciò che l'utente vede (foreground o activity) e ciò che l'utente non vede (background o service). Nei paragrafi successivi sono elencati i compiti delle due parti.

Funzionamento del service

Il service è la parte dell'applicazione che opera "all'insaputa" dell'utente e si occupa di eseguire un monitoraggio in automatico ogni 20 minuti (ma può comunque essere disattivato manualmente dall'utente). La rilevazione dei dati deve essere effettuata anche quando l'utente ha chiuso l'applicazione (o non l'ha aperta), in modo che il processo di monitoraggio venga avviato anche in assenza di un esplicito consenso da parte dell'utente. Il service viene attivato automaticamente all'accensione del dispositivo, senza richiedere ogni volta il permesso dell'utente (il permesso per avviare il service al termine del processo di boot viene richiesto una sola volta, prima dell'installazione). Una volta che il service è stato attivato, avvia la procedura di monitoraggio e inizia a verificare la disponibilità di una connessione internet (sia WiFi che 3G). Nel caso la rete non sia disponibile, il service termina il monitoraggio.

Se la rete è disponibile, il service controlla lo stato del ricevitore GPS (o di un dispositivo di geolocalizzazione equivalente). Se il GPS non è attivo, il service interrompe il monitoraggio. Se il GPS è attivo, il service esegue 3 geolocalizzazioni a distanza di 2 secondi l'una dall'altra. Se tutte e 3 le scansioni rilevano delle coordinate all'interno di una tratta ferroviaria (vedi paragrafo 4.2.1) , allora il service invia la velocità massima rilevata al server e controlla lo stato degli altri strumenti da utilizzare (Bluetooth, sensore di prossimità e accelerometro).

- Se il Bluetooth è presente ed è attivo, allora il service effettua in modo asincrono una scansione Bluetooth (tempo medio di 12 secondi). Al termine della scansione, per ogni dispositivo rilevato, invia la coppia <MAC dispositivo utente, MAC dispositivo rilevato> al server.

- Se il sensore di prossimità e l'accelerometro sono presenti ed attivi, allora il service calcola se il dispositivo è in tasca e se l'utente è seduto o in piedi. Lo stato dell'utente viene inviato al server. Questa rilevazione viene effettuata solamente nel caso che il treno sia in movimento (quindi con velocità maggiore di zero).

Al termine del monitoraggio il service mostra all'utente, tramite notifica, quali dati sono stati raccolti e inviati al server. Nel paragrafo 4.2.2 ci sono maggiori dettagli sull'implementazione del service.

Funzionamento dell'activity

L'activity è quella parte dell'applicazione con cui l'utente può interagire in modo diretto. A differenza del service, l'activity si avvia solo quando l'utente apre l'applicazione e si interrompe quando l'utente esce dall'applicazione. In realtà l'applicazione è composta da più di una activity: ogni schermata dell'applicazione è una activity a parte; l'applicazione ha un totale di 5 schermate e quindi di 5 activity.

La prima schermata, quella che compare subito dopo avere avviato l'applicazione, è il *menu iniziale*. In questa activity l'utente può attivare o disattivare il service (attivo di default), ovvero scegliere di non voler avviare il monitoraggio automatico ogni 20 minuti. Sempre in questa activity l'utente può visualizzare le statistiche relative al dispositivo in uso (ad esempio da quanti treni sono stati raccolti i dati o quanti giudizi manuali sono stati inseriti). Per accedere all'activity successiva (quella relativa ai viaggi) l'utente deve selezionare il codice del treno sul quale vuole ricevere informazioni. È possibile anche scegliere una data (nel caso si volessero osservare i dati passati), di default è comunque selezionata la data attuale. Se è presente una

connessione internet, vengono richiesti al server i dati relativi a quel treno e viene visualizzata la schermata successiva.

La seconda schermata contiene una lista dei *viaggi* relativi al treno selezionato. Per ogni viaggio sono indicati: stazione di partenza, stazione di arrivo, orario del viaggio e numero totale di dati raccolti dalla rete. Da questa activity è possibile passare all'activity di valutazione per valutare un viaggio sulla lista (solo nel caso che siano stati raccolti dati relativi a quel viaggio). Per ogni viaggio è possibile passare anche all'activity di visualizzazione dei dati.

La terza schermata, quella della *valutazione*, consente all'utente di dare una propria valutazione manuale ad certo viaggio. In questa schermata è possibile inviare al server una valutazione (di massimo 3 stelle) su pulizia del treno, disponibilità di posti a sedere e puntualità del treno. È inoltre possibile lasciare un commento sul viaggio. Confermando la valutazione, i dati vengono inviati al server e l'applicazione torna all'activity iniziale (menu principale).

La quarta schermata e la quinta schermata sono dedicate alla *visualizzazione dei dati*. È possibile passare da una schermata all'altra semplicemente trascinando l'activity a destra o a sinistra. Nell'activity di sinistra è presente una lista con i dati monitorati automaticamente:

- Velocità media: media delle velocità giornaliere per quel viaggio ricevute dal server, in chilometri all'ora.

- Numero di persone presenti sul treno: numero delle persone rilevate attraverso il Bluetooth, contando solo gli indirizzi MAC distinti dalle coppie <MAC1, MAC2> presenti sul server. Se il numero degli indirizzi MAC rilevati è inferiore al numero dei dispositivi che hanno inviato i dati al server, allora viene visualizzato quest'ultimo dato.

- Numero di persone in piedi.

- Numero di persone sedute.

Nell'activity di destra è possibile visualizzare la media delle valutazioni globali relative a quel viaggio, compresi gli eventuali commenti rilasciati dagli utenti.

Nel paragrafo 4.2.3 sono presenti i dettagli dell'implementazione delle activity.

3.4 Scelte progettuali

Avvio automatico del service

Per scegliere l'intervallo di tempo che passa tra un monitoraggio e quello successivo si deve tenere conto di alcuni requisiti:

- Meglio non scegliere un intervallo di tempo troppo basso (da 1 a 10 minuti):

  Utente sul treno Nella maggior parte dei casi la situazione sul treno non cambia in pochi minuti e risulta inutile inviare al server dei dati simili a quelli inviati pochi minuti prima.

  Utente non sul treno Un monitoraggio continuo causerebbe un inutile spreco di risorse. Inoltre è improbabile che un viaggio duri solo pochi minuti, quindi un'eventuale salita su un treno si potrebbe rilevare anche con un intervallo di tempo non troppo basso.

- Meglio non scegliere un intervallo di tempo troppo alto (40 o più minuti):

  Utente sul treno La situazione del treno (ad esempio il numero di persone) spesso cambia tra una fermata e l'altra. Scegliendo un intervallo di tempo troppo alto si rischia di non monitorare alcuni cambiamenti essenziali. Per esempio, supponiamo di avere 4 stazioni in successione: A,B,C e D. Il tragitto $\overline{AB}$ dura 30 minuti e il tragitto $\overline{BC}$ dura 20 minuti. Prendiamo 40 minuti come intervallo di tempo tra un monitoraggio e l'altro. Supponiamo che l'utente

voglia partire dalla stazione A per arrivare alla stazione D e che il primo monitoraggio si avvii dopo 20 minuti dalla partenza (quindi al server vengono inviati i dati del tragitto $\overline{AB}$). Il successivo monitoraggio avviene dopo 40 minuti e il successivo tragitto non viene considerato: 10 minuti per arrivare a B, 5 minuti di attesa nella stazione B e 20 minuti per arrivare a C. Notiamo come in questo caso il monitoraggio non sia avvenuto nel tratto $\overline{BC}$ e che quindi gli utenti nella stazione C abbiano ricevuto dei dati probabilmente sbagliati (ad esempio non viene calcolato il numero di viaggiatori salito in B, che può essere elevato).

Utente non sul treno Un gran numero di pendolari fa dei viaggi in treno piuttosto brevi; tenendo l'intervallo troppo alto si rischia quindi di non monitorare interi viaggi. Riprendendo l'esempio precedente: supponiamo che l'utente voglia andare dalla stazione A alla stazione B e che primo monitoraggio parta 10 minuti prima della partenza (rilevando l'utente *non sul treno*). Il successivo monitoraggio avviene dopo 40 minuti, ma la durata del viaggio è di soli 30 minuti, quindi il viaggio fatto dall'utente non viene monitorato.

Tenendo presenti i fattori appena elencati ho tenuto come intervallo il tempo medio di un viaggio (su un treno regionale) tra una stazione e quella successiva: 20 minuti.

Accensione automatica dei sensori

Nelle applicazioni Android, richiedendo i relativi permessi, è possibile abilitare in modo automatico gli strumenti presenti nel dispositivo (GPS, Bluetooth, ...) e la rete internet (connessione dati o WiFi). Ho scelto di non seguire questa strada per due principali motivi:

1. Abilitare in modo automatico tutti i sensori ogni 20 minuti causa un calo significativo del livello di batteria. Questo può essere fastidioso

per l'utente e può portarlo a disinstallare l'applicazione;

2. Anche se l'applicazione viene eseguita in background, deve essere comunque l'utente ad avere un controllo globale su di essa. Se per esempio l'utente non ha attivato la connessione dati perché non vuole spendere soldi, l'applicazione non può attivarla senza il suo consenso.

Per questi motivi lascio che l'applicazione si adatti al dispositivo sul quale è installata, utilizzando solamente gli strumenti effettivamente a disposizione.

3.5 Algoritmi utilizzati

Per calcolare l'intensità delle vibrazioni ricevute dal treno utilizzo i dati ricavati dalle rilevazioni dell'accelerometro.

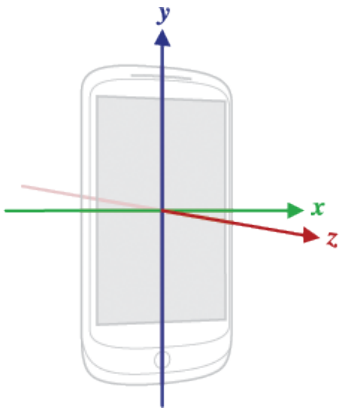

Figura 3.2: Sistema delle coordinate nelle API Android

Un accelerometro misura le accelerazioni applicate al dispositivo sugli assi (indicati in figuta 3.2), inclusa quella di gravità. Quando, per esempio, il dispositivo è appoggiato su un tavolo, l'accelerometro rileva una accelerazione verso il basso di 9.81 m/s^2 (accelerazione gravitazionale). Quando invece il dispositivo è in caduta libera, l'accelerazione verso il basso diventa quindi di 0 m/s^2. Per questo motivo, per calcolare la *reale* accelerazione subita dal dispositivo, devo sottrarre l'accelerazione gravitazionale alle accelerazioni rilevate.

Il mio obiettivo è calcolare l'intensità delle vibrazioni del treno rilevate dal dispositivo, ovvero la differenza di accelerazione tra un rilevamento e quello successivo. Chiamo x_t, y_t e z_t le tre accelerazioni in figura 3.2 rilevate all'istante t. Il valore assoluto dell'accelerazione del dispositivo all'istante 0 e all'istante t sarà data da

$$a_0 = |g| \approx 9.81 m/s^2$$

$$a_t = \sqrt{x_t^2 + y_t^2 + z_t^2}$$

Nelle accelerazioni a_t è sempre inclusa l'accelerazione gravitazionale. Calcolo ora la differenza tra l'accelerazione all'istante t e l'accelerazione all'istante $t-1$:

$$\Delta_t = a_t - a_{t-1}$$

Si può notare come $\Delta_1 = a'_1$ sia la reale accelerazione del dispositivo nel primo istante, dopo aver sottratto l'accelerazione gravitazionale a_0. Per le accelerazioni successive utilizzo la tecnica del *high-pass filter* , ovvero:

$$a'_t = \Delta_t + a'_{t-1} * \alpha \text{ con } 0 \leq \alpha \leq 1$$

dove α è la costante del high-pass filter (il suo valore è circa 0.9) che dà più importanza agli improvvisi cambiamenti di accelerazione, utile per individuare scossoni e/o vibrazioni del treno. Le prove sperimentali sull'accelerometro sono riportate nel paragrafo 5.2.

Capitolo 4

Implementazione

Questo capitolo contiene i dettagli dell'implementazione della tesi e le scelte implementative. Nel paragrafo 4.1 ci si concentra sulla parte relativa al server, ovvero quella parte che riguarda la memorizzazione dei dati e la gestione delle richieste provenienti dai client. Nel paragrafo 4.2 ci si concentra invece sull'implementazione dell'applicazione Android e sulla gestione delle risposte del server.

4.1 Programmazione server

La programmazione del server comprende l'implementazione del database (MySQL) e la scrittura degli script PHP che gestiscono le richieste provenienti dai client.

4.1.1 Database

Nel database devo memorizzare solo i dati effettivamente importanti, senza occupare spazio per dati inutili e senza rischiare di perdere dati utili a fini statistici.

```
1 CREATE TABLE VELOCITA (Train VARCHAR(15),
2  Travel VARCHAR(10),
3  Device VARCHAR(100),
```

```
4  TodayDate VARCHAR(15),
5  Speed VARCHAR(100) NOT NULL,
6  PRIMARY KEY(Train, Travel, Device, TodayDate));
```

La tabella *VELOCITA* deve contenere la velocità (Speed) rilevata dal dispositivo (Device) durante il viaggio (Travel) di oggi (TodayDate) sul treno (Train). Se un utente in un giorno rileva più di una velocità durante lo stesso viaggio, allora devo tenere la velocità più recente, sovrascrivendo quella precedente.

```
1  CREATE TABLE BLUETOOTH(Train VARCHAR(15),
2  Travel VARCHAR(10),
3  Mac1 VARCHAR(20),
4  Mac2 VARCHAR(20),
5  TodayDate VARCHAR(15),
6  PRIMARY KEY(Train, Travel, Mac1,Mac2, TodayDate));
```

La tabella *BLUETOOTH* contiene gli indirizzi (Mac2) di tutti i dispositivi rilevati dal dispositivo dell'utente (Mac1) sul treno (Train). La scansione è relativa ad un determinato viaggio (Travel), perché tra un viaggio e l'altro il numero di passeggeri può cambiare. Se un dispositivo effettua più scansioni durante lo stesso viaggio, allora devo tenere memorizzati gli indirizzi delle scansioni precedenti e aggiungere eventuali indirizzi nuovi rilevati dall'ultima scansione.

```
1  CREATE TABLE STATO(Train VARCHAR(15),
2  Travel VARCHAR(10),
3  Device VARCHAR(100),
4  TodayDate VARCHAR(15),
5  Status VARCHAR(30) NOT NULL,
6  PRIMARY KEY(Train, Travel, Name, TodayDate));
```

La tabella *STATO* è costruita per contenere la posizione (Status) del viaggiatore (Device) durante un viaggio (Travel) sul treno (Train). Se sono state effettuate più rilevazioni in un giorno (TodayDate) sullo stesso viaggio, allora si tiene la rilevazione più recente, sovrascrivendo quelle passate.

```
1 CREATE TABLE VALUTAZIONE(Train VARCHAR(15),
2  Travel VARCHAR(10),
3  Device VARCHAR(100),
4  Seating VARCHAR(5) NOT NULL,
5  Cleaning VARCHAR(5) NOT NULL,
6  Punctuality VARCHAR(5) NOT NULL,
7  Comment VARCHAR(300),
8  PRIMARY KEY(Train, Travel, Device));
```

La tabella *VALUTAZIONE* si occupa di mantenere i dati immessi manualmente dall'utente (Device). Un utente può inserire i dati su un viaggio (Travel) solo se ha raccolto precedentemente altri dati relativi a quel viaggio. L'utente può valutare la qualità del treno (Train) in base alla pulizia (Cleaning), ai posti a sedere (Seating) e alla puntualità (Punctuality). Inoltre può anche aggiungere un commento (Comment) per evidenziare eventuali lati positivi o negativi del viaggio. Ogni utente può inserire una sola valutazione per ogni viaggio (indipendentemente dalla data), ma può sempre modificare le valutazioni fatte (ad esempio togliendo il commento).

4.1.2 PHP

Gli script PHP gestiscono le richieste dei client e, prelevando i dati dal database, inviano una risposta in formato JSON. Per il collegamento con il DBMS si utilizza l'estensione PDO .

Il collegamento con il DBMS viene fatto nel seguente modo:

```
1 header("content-type:application/json;charset=UTF-8");
2 $response = array(); //preparo la risposta
3 $db = new PDO("host;nome database;username;password");
4 $db->setAttribute(PDO::ATTR_ERRMODE,
5                    PDO::ERRMODE_EXCEPTION);
6 $db->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

Come si può notare, utilizzando PDO il codice dello script viene scritto indipendentemente dal DBMS utilizzato.

Inserimento di dati nel database

```
1  //Controllo la correttezza della richiesta del client
2  if (isset($_POST["devname"])
3      and isset($_POST["speed"])
4      and ...)){
5      //Prelevo i dati dalla richiesta POST
6      $devname = $_POST["devname"];
7      $travel_id = $_POST["travel_id"];
8      $code = $_POST["train_code"];
9      ...
10     try{
11       $statement = $db->prepare("REPLACE INTO
12                                 tabella VALUES (?,?,...)");
13       $statement->execute(array($code,$travel_id,...));
14       $response["success"]="true";
15     }
16 /*In caso di errore invio la sua descrizione
17 con  $response["success"]="false" */
18 ...
19 echo json_encode($response);
```

I dati da inserire vengono passati al server attraverso la richiesta POST di HTTP. Lo script utilizza PDO per inserire questi dati nel database e risponde al client con un file JSON così formato:

```
1  {
2   "success" : <"true" o "false">
3  }
```

Richiesta di dati al database

```
1    if (isset($_POST["train"])
2    and isset($_POST["date"])){
3    try{
4      $train = $_POST["train"];
```

```
5       $date = $_POST["date"];
6       $stmSpeed = $db->prepare("SELECT * FROM VELOCITA
7                                 WHERE Train=? AND Date=?");
8       $stmSpeed->execute(array($train,$date));
9       $rowsSpeed = $stmSpeed->fetchAll(PDO::FETCH_ASSOC);
10      $response["speed_table"] = $rowsSpeed;
11      //SELECT da altre tabelle
12      ...
13      $response["success"]="true";
14      //Controllo sugli errori e stampa del risultato
15      ...
```

Il client invia al server il codice del treno e la relativa data in una richiesta POST. Il server deve rispondere con un file JSON così formato:

```
1  {
2   "speed_table" : [
3     {
4       "Train" : <codice del treno>,
5       "Device" : <id del device>,
6       "TodayDate" : <data della rilevazione>,
7       ...
8     },...
9    ],
10   "status_table" : [...],
11   "bt_table" : [...],
12   "valuation_table" : [...],
13   "success" : "true"
14 }
```

Nella risposta del server sono contenuti tutti i dati relativi al codice del treno e alla data passati come parametri.

4.2 Programmazione client

La programmazione del lato client (applicazione) comprende la gestione dei metadati (orari dei treni e coordinate delle linee ferroviarie), l'implementazione del service (per gestire il monitoraggio) e la struttura delle activity (parte grafica).

Struttura dell'applicazione

Elenco di seguito le principali classi realizzate per l'implementazione dell'applicazione:

- Activity:

  MenuActivity Activity iniziale, adibita all'attivazione manuale del service e alla richiesta dei dati al server;

  TravelListActivity Activity che visualizza la lista dei viaggi del treno selezionato;

  InfoListActivity FragmentActivity composta da due activity: la prima activity visualizza le informazioni rilevate automaticamente relative al viaggio selezionato, la seconda activity visualizza quelle inserite manualmente dagli utenti;

  ValuationActivity Activity che consente all'utente di inserire una propria valutazione manuale.

- Service:

  MyService Service che si occupa del monitoraggio automatico eseguito in background.

- Altro:

  BluetoothReceiver Receiver che gestisce la scansione Bluetooth e invia i dati al server;

JSONParser Classe che si occupa del parsing dei file JSON presenti nelle risorse o ricevuti come risposta dal server;

MySensorListener Listener che si occupa della gestione dei sensori (prossimità e accelerometro) e invia i dati al server;

InsertXX e GetXX AsyncTask utilizzate per interfacciarsi con il server;

Travel Classe che contiene i dati relativi ad un viaggio;

Train Classe che contiene i dati relativi ad un treno (compresi i viaggi);

TravelInformation Classe che contiene le informazioni monitorate relative al viaggio selezionato dall'utente.

4.2.1 Metadati

I dati riguardanti le coordinate delle linee ferroviarie e i dettagli di ogni treno sono inseriti nell'apk come risorse e sono divisi in due file *JSON*: *mappa.json* e *treni.json*. Il primo file contiene le coordinate dei percorsi tra ogni 2 stazioni. Per la struttura di questo file mi sono ispirato ai dati offerti dalle API di Google Maps (bib. [5]), dove per ogni percorso tra due punti vengono presi in considerazione tutti i segmenti che compongono quel percorso. Il file ha quindi la seguente struttura:

```
1  {
2   "routes" : [
3     {
4       "id" : <id_route>,
5       "end_location" : {
6         "name": <nome stazione di arrivo>,
7         "lat" : <latitudine stazione arrivo>,
8         "lng" : <longitudine stazione di arrivo>
9       },
10      "start_location" : {
11      "name": <nome stazione di partenza>,
```

```
12        "lat" : <latitudine stazione partenza>,
13        "lng" : <longitudine stazione di partenza>
14        },
15        "steps" : [
16          {
17            "end_location" : {
18              "lat" : <latitudine punto di arrivo>,
19              "lng" : <longitudine punto di arrivo>
20            },
21            "start_location" : {
22              "lat" : <latitudine punto di partenza>,
23              "lng" : <longitudine punto di partenza>
24            }
25          }, ...
26        ]}, ...
27    ]}
```

La struttura è una lista di *route*. Ogni route è descritta da una stazione di partenza e una stazione di arrivo. Il percorso relativo ad un route è formato da una lista di *step*, cioè dai segmenti che lo compongono. Ogni step è descritto dalle coordinate dei due punti estremi del segmento che rappresenta. Ad esempio, se un route va da una stazione nel punto A ad una stazione nel punto D, allora gli steps che lo compongono saranno: $\overline{AB}, \overline{BC}$ e $\overline{CD}$, dove B e C sono i punti dove il percorso cambia direzione (ad esempio nei pressi di una curva).

Il secondo file (*treni.json*) contiene le informazioni su ogni treno ed ha la seguente struttura:

```
1  {
2    "trains" : [
3      {
4        "code" : <codice treno>,
5        "travels": [{
6          "travel_id" : <codice univoco del viaggio>;
```

```
7           "min_hour" : <ora di partenza>,
8           "max_hour" : <ora di arrivo>,
9           "route" : <tratta ferroviaria percorsa>
10        }, ...
11      ]} ,...
12  ]}
```

La struttura è una lista di *train*. Ogni train è descritto da un proprio codice univoco e da una lista di viaggi *travel* che quel treno effettua regolarmente. Ogni travel è descritto da un orario di partenza, da un orario di arrivo, da un id univoco e dall'id del route corrispondente nel file *mappa.json*.

4.2.2 Service

Come è stato illustrato nel capitolo 3.3, il compito del service è utilizzare i sensori presenti nel dispositivo per effettuare il monitoraggio dei dati da inviare al server. Il service viene chiamato in automatico ogni 20 minuti e deve essere avviato al termine del boot del dispositivo. Per fare questo devo inserire il seguente codice nel *AndroidManifest.xml*[1]:

```
1 <receiver
2   android:name=".BootReceiver"
3   android:enabled="true"
4   android:exported="false" >
5   <intent-filter>
6     <action android:name=
7     "android.intent.action.BOOT_COMPLETED" />
8   </intent-filter>
9 </receiver>
```

In questo modo viene registrato un *receiver* che ha il compito di gestire gli intent di tipo *BOOT COMPLETED* (inviati al termine del processo di boot). Una volta che il receiver ha ricevuto questo intent deve avviare il

[1]File che definisce i contenuti e il comportamento dell'applicazione.

service e programmare l'avvio del monitoraggio ogni 20 minuti a partire da quell'istante.

Considerando che un service può essere automaticamente arrestato da Android in caso di mancanza di risorse e che il service che devo costruire si deve attivare anche con l'applicazione chiusa, mi è risultato impossibile utilizzare un TimerTask per programmare l'avvio del monitoraggio ogni 20 minuti. Ho optato dunque per l'utilizzo della classe AlarmManager , che offre uno scheduling degli eventi utilizzando i service del sistema, in questo modo il service diventa indipendente dall'applicazione stessa (che può anche essere chiusa).

Per quanto riguarda l'implementazione del monitoraggio, la prima cosa da fare è assicurarsi che il dispositivo possa connettersi alla rete:

```
1 public static boolean checkConnection(Context ctx){
2     ConnectivityManager cm = (ConnectivityManager)
3     ctx.getSystemService(Context.CONNECTIVITY_SERVICE);
4     NetworkInfo ni = cm.getActiveNetworkInfo();
5     if ((ni!=null) && (ni.isConnected()))
6       return true;
7     else
8       return false;
9 }
```

Se il metodo checkConnection restituisce true, allora il service può inviare i dati al server e può iniziare dunque il processo di monitoraggio.

Il prossimo passo è trovare il miglior strumento di geolocalizzazione presente nel dispositivo. Se ad esempio il GPS non è presente o non è attivato, allora posso fare uso del WiFi o di uno strumento meno accurato. Per scegliere lo strumento migliore scrivo queste righe:

```
1 //LM = Location Manager
2 Criteria crit = new Criteria();
3 crit.setAccuracy(Criteria.ACCURACY_FINE);
4 //Prendo un provider attivo con la migliore accuratezza
5 String bestProv = LM.getBestProvider(crit, true);
```

```
6 if (LM.isProviderEnabled(bestProv)){
7   LM.requestLocationUpdates(bestProv, 2000,0,
8     new LocationListener(){...
```

Se il provider (GPS) esiste ed è attivo, allora inizio a monitorare, in modo asincrono, l'aggiornamento dei suoi valori (quando cambiano le coordinate) con un intervallo minimo di 2 secondi tra una scansione e l'altra.

Prelevando le coordinate dagli step del file *mappa.json*, il service verifica che le coordinate rilevate dal GPS facciano parte di uno degli step. In caso negativo interrompo la scansione del GPS e aspetto la prossima chiamata del service, mentre nel caso positivo attendo la prossima scansione del GPS. Alla terza scansione, se le coordinate rilevate dal GPS fanno ancora parte di uno step, devo inviare la velocità rilevata al server. Come è stato detto nel paragrafo 4.1.1, il database (oltre che alla velocità) ha bisogno di: un id univoco del dispositivo, il codice del treno, l'id del viaggio e la data di rilevazione.

```
1 Calendar actual_date = Calendar.getInstance();
2 //Prendo la data di oggi nella forma gg/mm/aaaa
3 String dateString = DateFormat.format("dd/MM/yyyy",
4                     actual_date).toString();
5 //Prendo l'ora attuale nella forma hh:mm
6 String hourString = DateFormat.format("kk:mm",
7                      actual_date).toString();
8 //ID univoco del device (TM = TelephonyManager)
9 String dev_id = TM.getDeviceId();
```

Successivamente calcolo l'id del *route* sapendo in quale step mi trovo. Conoscendo l'orario e l'id del route riesco a trovare i relativi codice del treno e id del viaggio nel file *treni.json*.

Se il Bluetooth è disponibile ed attivo, avvio in modo asincrono il relativo processo di scansione. Il processo di scansione invia in broadcast degli *intent* di tipo "ACTION FOUND" per ogni altro dispositivo rilevato. Al termine della scansione, mediamente della durata di 12 secondi, il processo

invia in broadcast un intent di tipo “ACTION DISCOVERY FINISHED” e ferma l’utilizzo del Bluetooth. Per ricevere gli intent devo costruire un BroadcastReceiver che in caso di “ACTION FOUND” mi aggiunga il MAC del dispositivo rilevato in una lista e in caso di “ACTION DISCOVERY FINISHED” invii le coppie <MAC dispositivo utente, MAC dispositivo rilevato> al server.

Se la velocità rilevata è maggiore di zero e i sensori sono disponibili e attivi, registro un Listener (handler asincrono) in ascolto sul cambiamento dei valori del sensore di prossimità. Il sensore di prossimità può assumere due valori: 0 (prossimità vicina) e 5 (prossimità lontana). Appena ci si mette in ascolto sul sensore, il valore restituito di default è *sempre* 5, anche se il dispositivo è in tasca (quindi con prossimità vicina). Il valore 0 può essere restituito solo come secondo valore, nel caso che il dispositivo sia in tasca. Per risolvere questo problema creo un TimerTask della durata di 2 secondi (entro cui sicuramente può arrivare il valore 0): se entro tale tempo il valore 0 è arrivato, allora il dispositivo è in tasca, altrimenti non è in tasca.

```
1  is_near = false;
2  ...
3  Timer t = new Timer();
4  Calendar now = Calendar.getInstance();
5  now.add(Calendar.SECOND, 2);
6  t.schedule(new TimerTask(){
7    @Override
8    public void run() {
9      if (!is_near)
10       //Dispositivo non in tasca, termino qui
11   }
12 }, now.getTime());
13 if (event.values[0] != 5)
14   is_near = true;
15   //Dispositivo in tasca, registro un Listener
16   //per il prossimo sensore e termino qui
```

Se il dispositivo viene rilevato come "in tasca", registro un Listener in ascolto sui cambiamenti dei valori dell'accelerometro e faccio un totale di 20 rilevamenti. Considerando l'algoritmo illustrato nel paragrafo 3.5 e le prove sperimentali presentate nel paragrafo 5.2, posso supporre che se l'intensità dell'accelerazione rilevata in *ognuno* dei 20 rilevamenti è inferiore ad 1, allora l'utente è in piedi, altrimenti è seduto.

```
1  float accel = 0.00f;
2  float accelCurrent = SensorManager.GRAVITY_EARTH;
3  float accelLast = SensorManager.GRAVITY_EARTH;
4  int i = 0;
5  ...
6  public void onSensorChanged(SensorEvent event) {
7    i++;
8    accelLast = accelCurrent;
9    float x = event.values[0];
10   float y = event.values[1];
11   float z = event.values[2];
12   accelCurrent = (float)Math.sqrt(x*x + y*y + z*z);
13   float delta = accelCurrent - accelLast;
14   accel = accel * 0.9f + delta;
15   if (accel>1)
16     //Invio al server la posizione ''seduto''
17     //e termino
18   else if (i >= 20)
19     //Invio al server la posizione ''in piedi''
20     //e termino
21 }
```

Dove 0.9 è la costante del *high-pass filter*.

Rapporti con il server

Per interagire con il server faccio uso della classe AsyncTask, che mi permette di eseguire una parte di codice in modo asincrono. Come è stato

detto nel paragrafo 4.1.2, il server si aspetta dal client delle richieste HTTP di tipo POST. Queste richieste devono contenere i parametri di ricerca o di inserimento, utilizzati poi dal server per eseguire le relative query sul database.

```
1  List<NameValuePair> values =
2      new ArrayList<NameValuePair>();
3  values.add(
4    new BasicNameValuePair("devname",parametro1));
5  ...
6  DefaultHttpClient httpClient = new DefaultHttpClient();
7  HttpPost httpPost = new HttpPost(url);
8  httpPost.setEntity(new UrlEncodedFormEntity(values));
9  HttpResponse httpResponse =
10   httpClient.execute(httpPost);
```

Nell'ultima riga il client si mette *in attesa* (sincrona) della risposta del server, per questo motivo le parti relative all'interazione con il server devono essere eseguite in modo *asincrono*, a differenza del resto del codice.

Parsing dei file JSON

Le risposte del server arrivano al client in formato JSON e inoltre, come illustrato nel paragrafo 4.2.1, il client deve essere in grado di leggere le risorse in formato JSON presenti nell'apk. Per fare il parsing dei file JSON utilizzo le classi JSONArray (JA) e JSONObject (JO). Ad esempio, prendiamo il seguente file JSON:

```
1  {
2    "array" : [...]
3  }
```

Il file intero è un JO che comprende un JA chiamato "array".

Il file *mappa.json* è un JO che contiene un JA chiamato "routes" , che contiene a sua volta una serie di JO (i *route*). Ognuno di questi JO con-

tiene: una stringa "id", due JO ("end_location" e "start_location") e un JA chiamato "steps".

Una risposta del server è un JO che contiene una stringa "success" e tanti JA quante sono le tabelle nel database. Ogni JA contiene un JO per ogni riga di quella tabella restituita dalla query.

4.2.3 Activity

L'implementazione delle activity si divide principalmente in due parti: la parte del *layout* e la parte della gestione degli eventi. Il layout di una applicazione Android viene scritto in XML ed è formato da *widget* . Per esempio, per inserire un bottone in una activity devo inserire nel relativo layout il seguente codice:

```
1 <Button
2         android:id="@+id/btnShowInfo"
3         style="@style/myBtnStyle"
4         android:layout_width="wrap_content"
5         android:layout_height="wrap_content"
6         android:layout_alignParentBottom="true"
7         android:layout_centerHorizontal="true"
8         android:layout_marginBottom="38dp"
9         android:text="@string/showInfo" />
```

In questo caso, per inserire il bottone nell'interfaccia grafica ho specificato:

id Ogni widget ha un id univoco ("btnShowInfo") che mi consente di riferirmi a lui in ogni parte dell'applicazione.

style In questo caso ho voluto cambiare lo stile standard del bottone, creandone uno rosso. "myBtnStyle" è l'id dello stile corrispondente, creato in un'altra risorsa XML, di nome "style".

layout I parametri di layout servono per definire la dimensione e la posizione del bottone nell'interfaccia grafica.

text Il testo contenuto dal bottone è contenuto nella risorsa *string.xml* ed ha id "showInfo".

Per collegare l'activity (*MenuActivity.java*) al layout corrispondente (*activity_menu.xml*) devo inserire la seguente riga nell'activity:

```
1 setContentView(R.layout.activity_menu);
```

In questo modo riesco a gestire il bottone ricercandolo tramite il suo id:

```
1 Button btnShowInfo =
2   (Button)findViewById(R.id.btnShowInfo);
```

L'activity chiamata all'avvio dell'applicazione è definita nel file *AndroidManifest.xml* con il seguente codice:

```
1 <activity
2   android:name="com.project.MenuActivity"
3   android:label="@string/app_name" >
4   <intent-filter>
5     <action android:name="android.intent.action.MAIN"/>
6     <category
7       android:name="android.intent.category.LAUNCHER"/>
8   </intent-filter>
9 </activity>
```

Dopo aver definito il layout, posso procedere con la gestione degli eventi.

Come è stato detto nel paragrafo 3.3, il service può essere interrotto manualmente dall'utente. Per fare questo devo creare una relazione tra activity e service, in modo che l'activity possa sapere in qualsiasi momento quando il service è attivo. Tra i possibili modi per realizzare questo legame, ho scelto di utilizzare la classe SharedPreferences, che consente di memorizzare in modo *persistente* (e condiviso tra activity e service) delle coppie <chiave,valore> formate da dati primitivi. Durante la creazione del service (metodo "Create"), prima dell'avvio del monitoraggio, memorizzo una SharedPreference <"service_attivo",true>. Nel metodo "Destroy", invocato subito prima della

chiusura del service (sia volontaria che involontaria), modifico la SharedPreference in <"service_attivo",false>. La persistenza della SharedPreference garantisce all'activity la possibilità di leggere in qualsiasi momento il valore corrispondente alla chiave "service_attivo" e, quindi, di verificare quando il service è attivo.

Per la gestione dei dati provenienti dalle risorse o dalle risposte del server, ho creato le classi Travel, Train e TravelInformation. La creazione di queste classi mi permette di ricavare i dati una sola volta e passarli tra una activity e l'altra, attraverso l'utilizzo degli intent. La classe Travel contiene le informazioni relative ad un singolo viaggio: l'id del viaggio, gli orari, l'id della route corrispondente e le stazioni di partenza e di arrivo. La classe Train contiene le informazioni relative ad un treno: il codice del treno e la lista di Travel che può effettuare. La classe TravelInformation contiene le informazioni monitorate durante un viaggio: tutte le velocità effettuate, gli indirizzi MAC dei dispositivi Bluetooth rilevati, la posizione degli utenti rilevata dai sensori e le valutazioni manuali fornite dagli utenti. Il codice che segue si riferisce alla richiesta fatta dal client appena l'utente conferma il codice del treno e la data inseriti.

```
1  //Chiamata asincrona al server
2  new AsyncGetInfo(){
3    //La chiamata mi restituisce un oggetto
4    //di tipo TravelInformation
5    protected void
6      onPostExecute(TravelInformation result){
7      if (result != null){
8        //Costruisco l'intent per chiamare l'activity
9        //che contiene la lista dei viaggi
10       Intent intent =
11         new Intent(this, TravelListActivity);
12       //Aggiungo gli oggetti di tipo TravelInformation
13       //e Train agli extra dell'intent
14       intent.putExtra("TravelInfo", result);
```

```
15          intent.putExtra("Train", selected_train);
16          //Avvio l'activity corrispondente
17          startActivity(intent);
18      }
19    }
20  //Al server passo il codice del treno e la data
21  }.execute(traincode,selected_date);
```

Nella classe AsyncGetInfo invio al server il codice del treno e la data, inserendo i dati della risposta in un oggetto di tipo TravelInformation. Nelle activity successive non ho più bisogno di richiedere i dati al server, basta passare gli oggetti di tipo Train e TravelInformation da una activity all'altra, attraverso gli intent.

L'elaborazione dei dati ricevuti dal server avviene nell'activity adibita alla visualizzazione dei dati, subito dopo aver selezionato il relativo viaggio.

Capitolo 5

Valutazione

5.1 Guida all'uso

Attivare il service

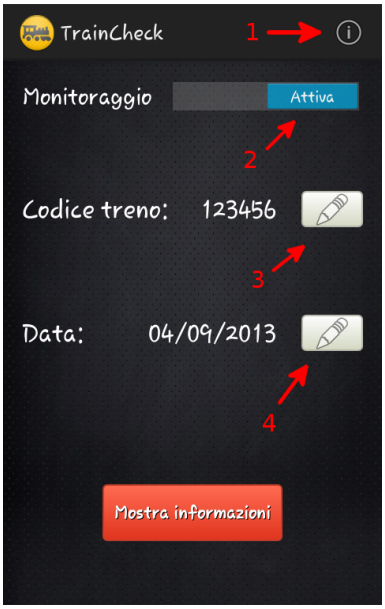

Figura 5.1: Guida - menu iniziale

Per attivare o disattivare il service è sufficiente cliccare o trascinare il bottone illustrato nella figura 5.1 (2). Quando si attiva manualmente il service,

parte subito un primo processo di monitoraggio, utile nel caso l'utente voglia rilevare "manualmente" i dati. Quando un monitoraggio è terminato con successo, viene visualizzata una notifica con indicati i relativi dati rilevati.

Leggere le informazioni del dispositivo

Per leggere le informazioni del dispositivo si deve selezionare la voce "Informazioni sul dispositivo" dal menu, come illustrato in figura 5.1 (1). Per fare questo il dispositivo deve essere connesso alla rete internet. Una volta selezionata l'opzione nel menu, verrà visualizzata una schermata come in figura 5.2.

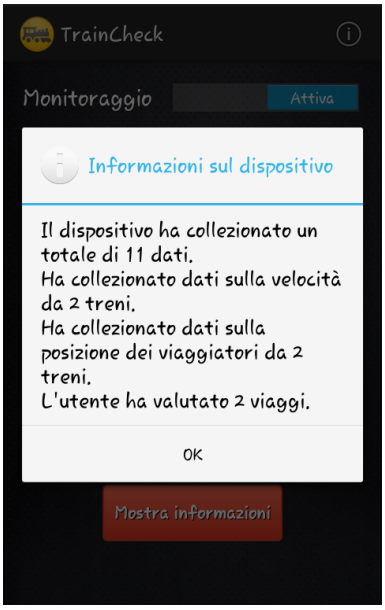

Figura 5.2: Guida - informazioni dispositivo

Leggere le informazioni relative ad un viaggio

Per leggere le informazioni relative ad un viaggio il dispositivo deve essere connesso alla rete internet. Si devono scegliere il codice del treno (3) e la data (4), come illustrato in figura 5.1. Una volta selezionati i filtri di ricerca, premere sul bottone "Mostra informazioni".

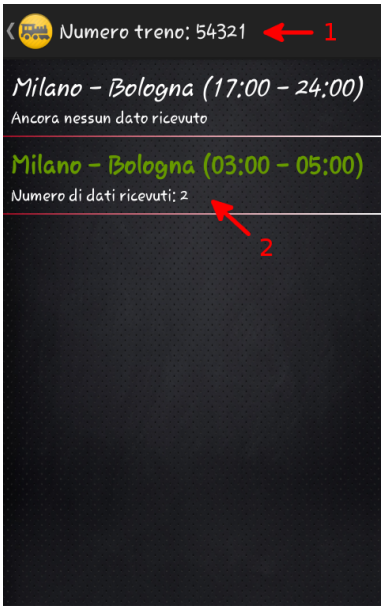

Figura 5.3: Guida - lista dei viaggi

La schermata che appare (figura 5.3) si presenta con una lista dei viaggi che quel treno può effettuare. Il titolo della schermata (1) indica il numero del treno scelto e selezionandolo si torna alla schermata precedente. Un viaggio si presenta nella forma "Partenza - Arrivo (Ora partenza - Ora arrivo)" e per ogni viaggio sono anche indicati quanti dati, sia automatici che manuali, sono stati raccolti per quel viaggio nella data selezionata. Quando il viaggio è scritto in corsivo vuol dire che l'utente non hai mai monitorato dati per quel viaggio, altrimenti viene scritto in grassetto (2).

Selezionando un viaggio appare la schermata in figura 5.4. Questa schermata contiene le informazioni relative al viaggio e alla data selezionati in precedenza. Nel campo "Velocità media" (1) è indicata la media (in km/h) delle velocità rilevate da tutti gli utenti che hanno monitorato quel viaggio in quella data. Nel campo "Numero di passeggeri rilevati" (2) è indicata una stima dei passeggeri rilevati a bordo. Questo numero è il massimo tra il numero di dispositivi che hanno effettuato il monitoraggio e il numero di dispositivi rilevato attraverso il monitoraggio Bluetooth. I campi "Persone in piedi" (3) e "Persone sedute" (4) si riferiscono ai dati rilevati attraverso

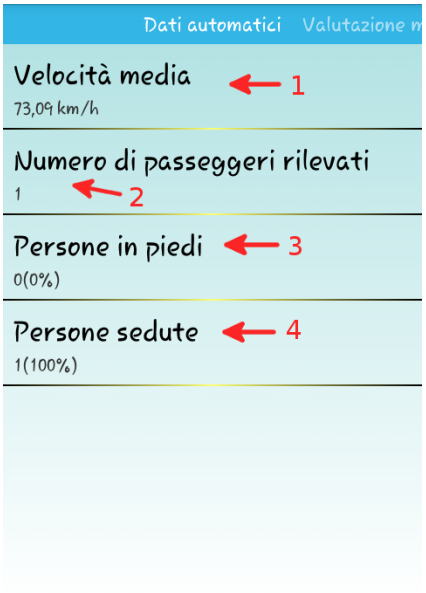

Figura 5.4: Guida - visualizzazione dati automatici

il sensore di prossimità e l'accelerometro. La percentuale visualizzata a lato del numero si riferisce al numero di passeggeri dati dalla somma delle persone sedute e le persone in piedi, non al numero di passeggeri rilevati dal Bluetooth (che può anche essere diverso). Selezionando uno dei campi precedenti compaiono delle informazioni relative al campo scelto, come ad esempio il metodo di rilevazione del dato.

Per passare alla schermata dei "risultati manuali" (figura 5.5) basta far scorrere il dito da destra a sinistra. In questa schermata (a scorrimento verticale) sono presenti la media delle valutazioni e tutti i commenti rilasciati dagli utenti relativi a quel viaggio, indipendentemente dalla data scelta (le valutazioni manuali riguardano il viaggio in generale, non solo quello giornaliero). Le valutazioni di "Pulizia", "Posti a sedere" e "Puntualità" sono espresse con *3 stelle* e il punteggio di ciascuna valutazione assume un valore da 0 a 6 (si può dare anche un punteggio di *mezza stella*). Per visualizzare la lista dei commenti degli utenti riguardanti il viaggio si deve premere il bottone "Visualizza commenti". Per tornare alla schermata precedente si deve far scorrere il dito da sinistra a destra.

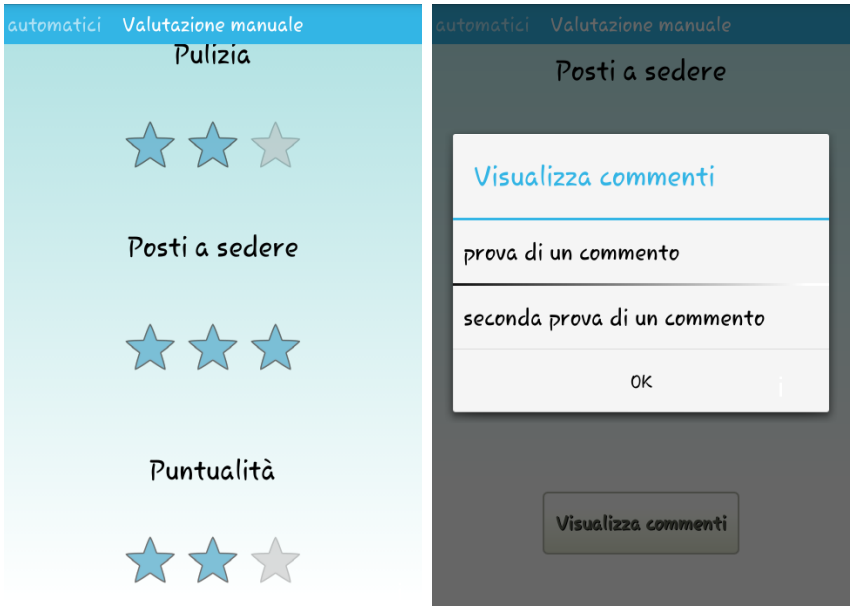

Figura 5.5: Guida - visualizzazione dati manuali

Valutare un viaggio

Per valutare un viaggio il dispositivo deve essere connesso alla rete e si deve tornare alla schermata in figura 5.3. I viaggi visualizzati in verde e grassetto sono *da valutare*, quelli in corsivo invece non si possono valutare, dato che l'utente non ha effettuato il viaggio (o non l'ha monitorato). Nella figura 5.6 sono rappresentati i vari passaggi per valutare un viaggio in treno. Dalla lista dei viaggi, tenendo premuto su un viaggio in grassetto, compare l'opzione "Valuta". Selezionando questa opzione compare una schermata a scorrimento verticale dove è possibile dare un punteggio da 0 a 6 (in *3 stelle*) ad ognuna delle 3 voci viste precedentemente. Inoltre è possibile inserire un commento relativo al viaggio, con massimo 300 caratteri (il commento può anche non essere inserito, semplicemente lasciando vuoto il relativo box). Terminate le modifiche, per confermare la valutazione premere sul bottone "Valuta". Ogni viaggio che è stato valutato correttamente viene segnato in azzurro e grassetto nella lista dei viaggi e sarà possibile modificare in qualsiasi momento la valutazione data.

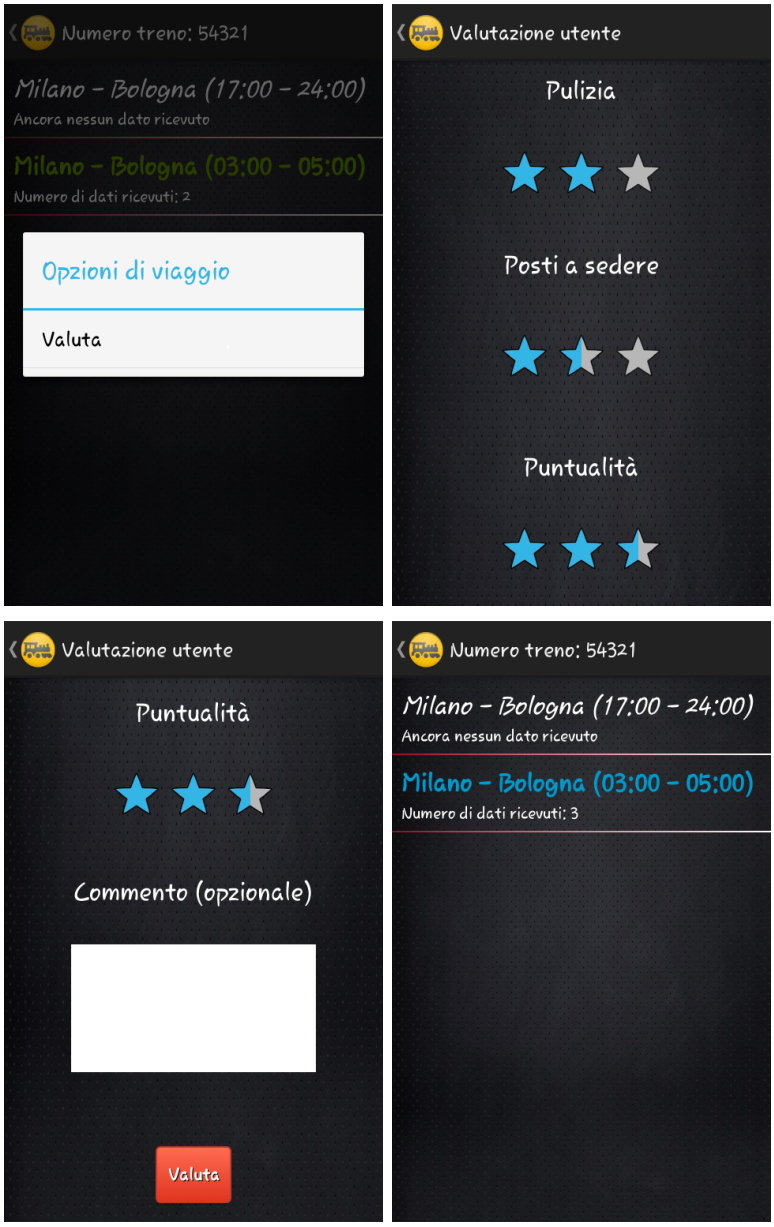

Figura 5.6: Guida - valutazione manuale

5.2 Prove sperimentali

Per calibrare il rilevamento automatico della posizione dell'utente (seduto o in piedi) ho dovuto fare alcune prove sperimentali. Ho realizzato dunque una applicazione Android con lo scopo di memorizzare in un database locale (in SQLite) i rilevamenti dell'accelerometro presente nel dispositivo. L'applicazione effettua un monitoraggio ogni 5 minuti e ogni monitoraggio rileva 20 variazioni di accelerazione. Provando l'applicazione in un reale viaggio in treno (con lo smartphone in tasca), ho effettuato un totale di 20 monitoraggi (10 mentre ero seduto e 10 mentre ero in piedi) e le vibrazioni ricevute dal treno in movimento avevano intensità diverse a seconda della mia posizione. L'accelerometro rileva un'intensità di accelerazione maggiore quando l'utente è seduto sul treno, mentre quando è in piedi, l'accelerazione rilevata ha un'intensità inferiore. Questo può essere dovuto al fatto che quando l'utente è seduto, tutto il corpo è a contatto con il treno; invece, quando l'utente è in piedi, la maggior parte delle vibrazioni viene assorbita dal corpo (a partire dai piedi) prima di arrivare alla posizione del dispositivo (tasca, zaino, ...). Per fare un confronto accurato dei due possibili casi ho scelto e realizzato due grafici che rappresentano rispettivamente un monitoraggio con la più bassa intensità di accelerazione rilevata da seduto (figura 5.7) e un monitoraggio con la più alta intensità di accelerazione rilevata in piedi (figura 5.8). Quello che si può subito notare è che la differenza tra le due diverse situazioni rimane accentuata anche cercando di scegliere due monitoraggi (un monitoraggio per situazione) con risultati molto vicini tra di loro.

Non avendo a disposizione i reali dati delle coordinate delle tratte ferroviarie, ho provato l'applicazione completa sia in automobile che in autobus, inserendo manualmente i dati delle relative coordinate. I dati ottenuti hanno rispettato le mie aspettative, trovando circa 5 viaggiatori in meno rispetto ai viaggiatori reali.

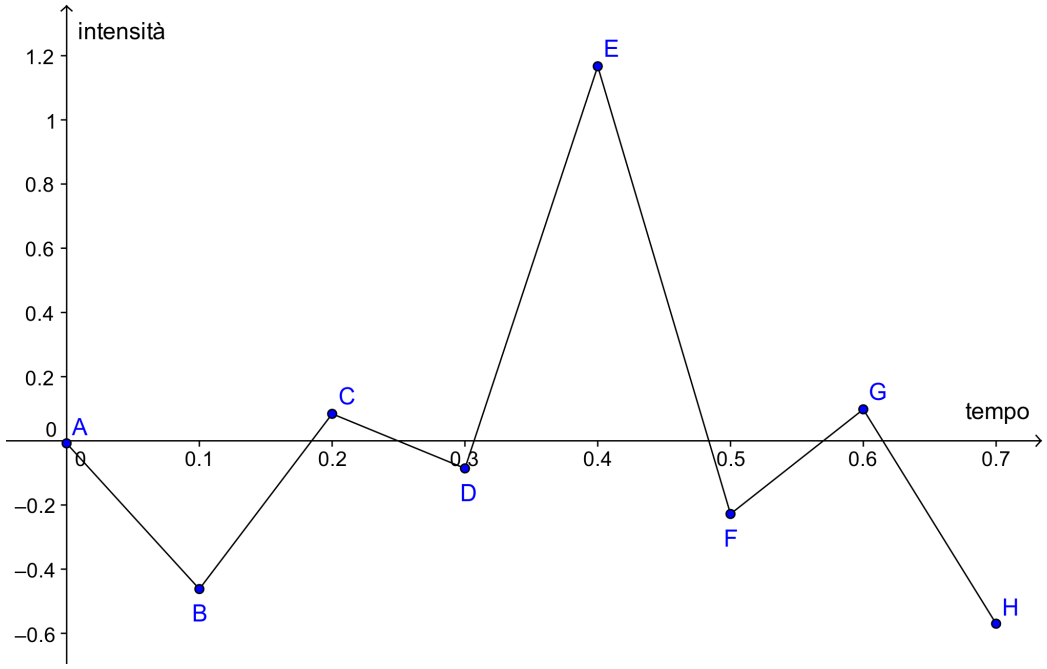

Figura 5.7: Monitoraggio da seduto con i valori più bassi

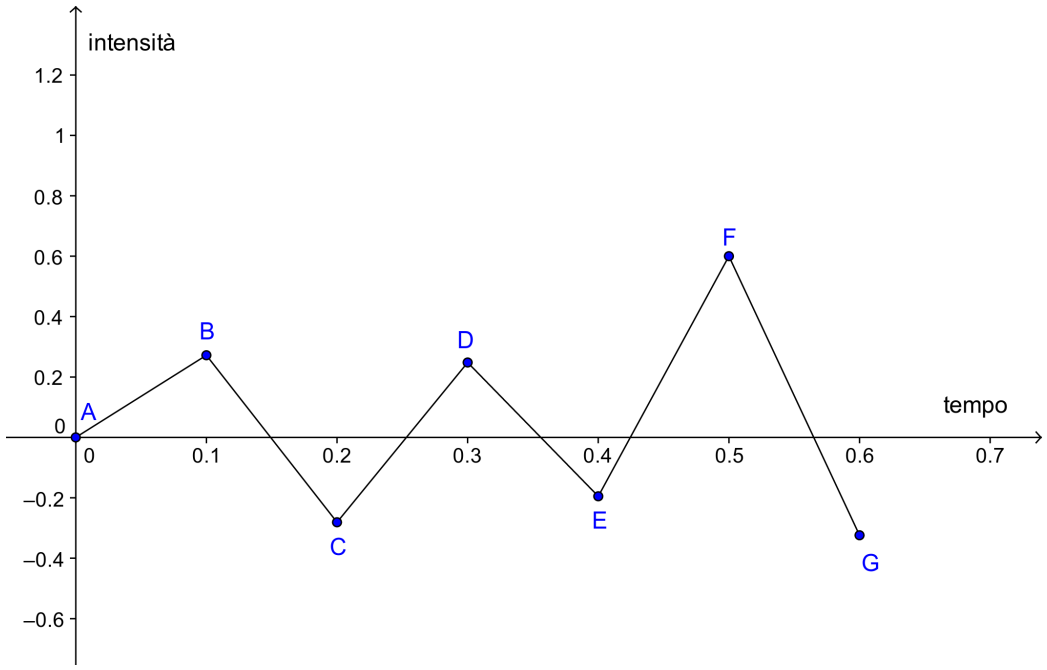

Figura 5.8: Monitoraggio da in piedi con i valori più alti

Conclusioni e sviluppi futuri

Il progetto che è stato presentato soddisfa gli obiettivi che mi sono posto nel capitolo 1 e le prove sperimentali effettuate sono state soddisfacenti, anche se l'applicazione realizzata, per essere effettivamente utile al suo scopo, deve essere utilizzata da un gran numero di viaggiatori. Rendere l'applicazione più *user-friendly* può essere un modo per incentivare l'utilizzo dell'applicazione anche da parte degli utenti con poca esperienza nell'utilizzo dei dispositivi mobili. Il progetto che è stato presentato in questa tesi può essere visto come uno scheletro per lo sviluppo di altri progetti e, ad esempio, può essere ampliato nei seguenti modi:

- Sviluppare un *Social Network* tra gli utenti che utilizzano l'applicazione. Ecco alcune caratteristiche che potrebbe avere:
  - Registrazione degli account;
  - Classifica degli utenti, ognuno con un relativo punteggio (in base al numero di monitoraggi effettuati);
  - Sistema di amicizie basato sul Bluetooth o sul NFC[1].
- Progettare un servizio automatico per la gestione degli orari e delle tratte ferroviarie, offrendo alle società ferroviarie l'opportunità di collaborare attraverso la condivisione delle informazioni.

[1]Near Field Communication

- Rendere l'applicazione compatibile anche con altri sistemi operativi per dispositivi mobili, come iOS e Windows Phone, in modo che il tipo di dati monitorati sia indipendente dal sistema utilizzato.

- Fare uno studio approfondito sui dati raccolti da un campione abbastanza grande di utenti, in modo da poter fornire una valutazione statistica sulla qualità del servizio di trasporto ferroviario italiano.

Bibliografia

[1] Sondaggi trenitalia del 2011: http://www.trenitalia.com/cms-file/allegati/trenitalia/area_clienti/Relazionesullaqualitadelservizio.pdf

[2] http://www.legambiente.it/contenuti/articoli/i-principali-comitati-dei-pendolari-italiani

[3] http://developer.android.com/tools/sdk/eclipse-adt.html

[4] Statistiche mensili di Kantar: http://www.kantarworldpanel.com/

[5] https://developers.google.com/maps/documentation/geocoding/

Indice analitico