Istituto Istruzione Superiore G. Vallauri Fossano

Esame di stato 2016

Autori:

Bergia Stefano Maritano Luca

Classe 5^ D Informatica

Anno scolastico 2015-2016

Progetto Nautibus

Indice

Introduzione

Sono moltissime le persone che si trovano a dover usufruire dei mezzi di linea per raggiungere le località di studio o lavoro ogni giorno. Sebbene esistano molte applicazioni che permettono di consultare orari e ricercare informazioni riguardanti i mezzi pubblici, poche concedono la possibilità di localizzare i mezzi sulla mappa e di conoscere in tempo reale eventuali ritardi/anticipi. Questa è proprio la finalità che la nostra applicazione si propone di raggiungere, oltre a quella di permettere all'utente di ricevere notifiche personalizzate che lo informino preventivamente dell'arrivo degli autobus ad una particolare fermata, in modo da essere sempre a conoscenza sullo stato della linea ed evitare così disagi e seccature varie.

Riassumendo, gli obbiettivi del progetto sono:

- Permettere all'utente di ricercare delle linee in base alle località di partenza e arrivo;
- Permettere all'utente di ottenere informazioni aggiuntive relative alle linee quali orari, autisti e ritardi;
- Permettere all'utente visualizzare il percorso delle linee su una mappa e di conoscere la posizione in tempo reale dei mezzi;
- Permettere all'utente di aggiungere delle linee ai preferiti;
- Permettere all'utente di ricevere delle notifiche push in tempo reale relative al passaggio dell'autobus in fermate a scelta;
- Permettere all'utente di richiedere l'aggiunta di una linea non registrata sul database tramite mail.

Strumenti utilizzati

Di seguito sono elencati gli strumenti utilizzati per lo sviluppo del progetto

- Android Studio 1.5: IDE utilizzato per lo sviluppo dell'applicazione Android e del "Tracker" realizzato per raccogliere le coordinate di percorsi "di esempio".

- Visual Studio Code: semplice editor utilizzato per la scrittura del simulatore di linee (HTML, Javascript) e degli script PHP costituenti il Web Service.

- Altervista: piattaforma gratuita per l'hosting del Web Service e del database MYSQL.

- MYSQL: DBMS per la gestione del database delle linee.

- Google Maps Android API: servizio gratuito offerto da Google per la visualizzazione di mappe geografiche su dispositivi Android, la localizzazione grafica di coordinate e la rappresentazione di "entità geografiche" quali punti e percorsi tramite appositi oggetti grafici. Utilizzata per la visualizzazione del percorso delle linee e della posizione attuale del mezzo.

- Google Distance Matrix Service: servizio web offerto da Google per il calcolo delle informazioni di distanza e tempo di percorrenza tra due coordinate geografiche data una modalità di trasporto.
- Non è completamente gratuito in quanto non permette di effettuare più di 2500 richieste al giorno. Utilizzato per il calcolo del tempo di percorrenza dalla posizione attuale ad una fermata.

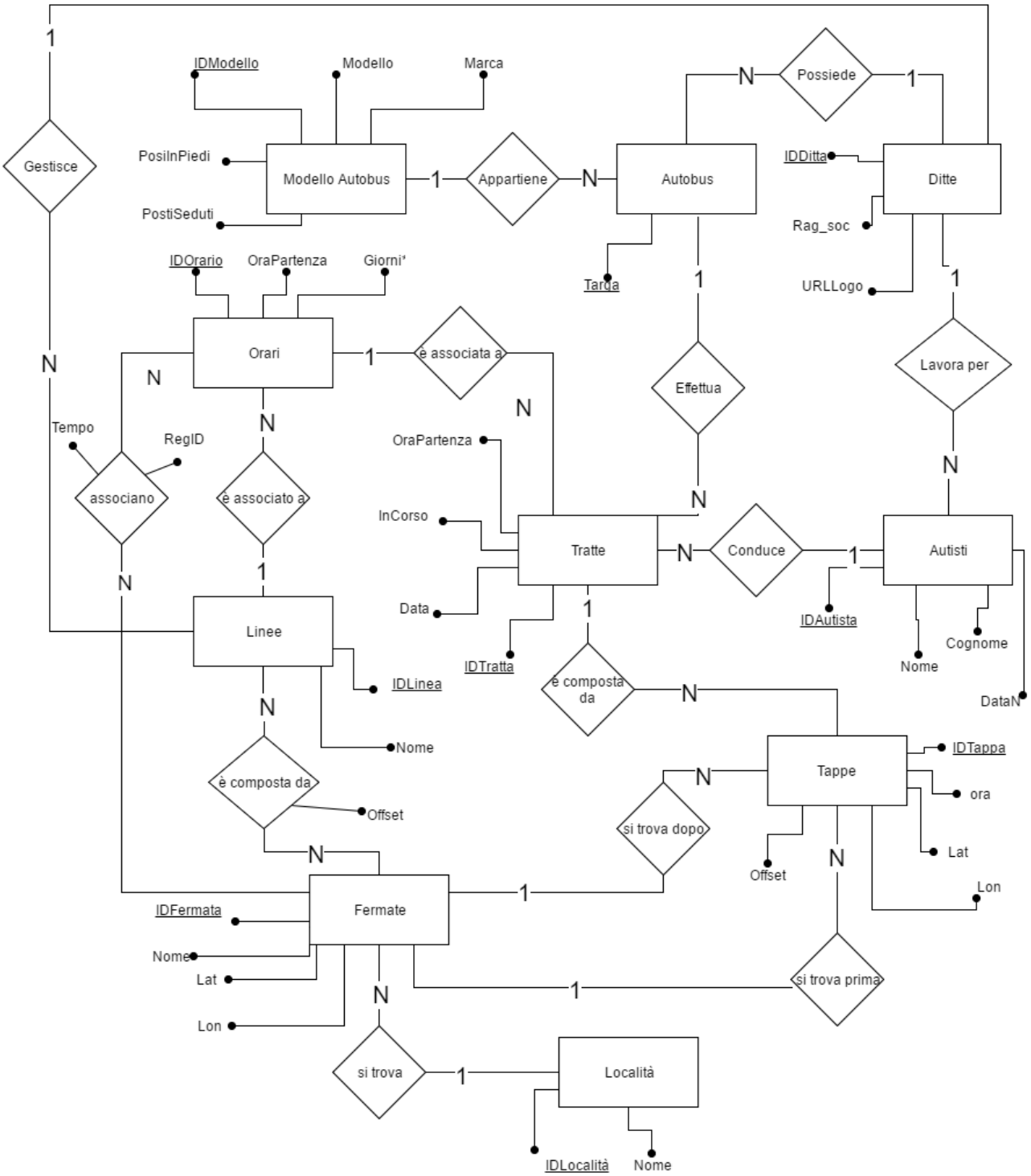
- Google Cloud Messaging API: servizio offerto gratuitamente da Google per l'invio di dati da un server a una serie di client registrati (nel nostro caso le varie installazioni dell'applicazione) e di ricevere informazioni dai client sulla stessa connessione. Il servizio GCM gestisce automaticamente tutti gli aspetti relativi alla coda dei messaggi e al loro invio alle applicazioni in esecuzione sui client degli utenti. Utilizzato per l'invio push delle notifiche alle varie installazioni dell'applicazione sui vari client.

- JQuery: libreria Javascript utilizzata per sviluppare il simulatore.

- Bootstrap: libreria CSS per la formattazione grafica del simulatore

Database

L'intero progetto si basa su database relazionale MYSQL di cui è riportato il modello entity-relationship di seguito



Modello Autobus: Rappresenta i vari tipi di autobus presenti memorizzandone la marca, il modello, il numero di posti in piedi ed il numero di posti seduti.

Autobus: Rappresenta un autobus specifico appartenente ad un dato modello, dotato di una targa che lo identifica.

Autisti: Rappresenta gli autisti di una ditta.

Ditta: possiede un attributo per memorizzare la ragione sociale e uno per il logo (URL di un file presente sul server).

Linee: Rappresenta le linee effettuate da una ditta, memorizzandone il nome.

Fermate: Rappresenta le fermate di una linea, ne memorizza il nome, la latitudine e la longitudine. Linee e fermate sono legate da un'associazione N-N con un attributo offset che rappresenta il numero di minuti necessari i per raggiungere una certa fermata rispetto all'ora di inizio della linea.

Località: rappresenta il comune o la città in cui una fermata è situata, verrà utilizzato per le ricerche.

Orari: Rappresenta gli orari delle linee (per ogni linea ci possono essere più orari). Memorizza l'orario di partenza della linea ed il giorno della settimana in cui si svolge (Lunedì, Martedì…)

Tratte: Rappresenta un singolo servizio che viene effettuato da un'autista e da un autobus (Vincolo di integrità: autisti e autobus devono essere della stessa ditta), ogni tratta è legata ad un orario (e di conseguenza anche ad una linea ed alle relative fermate e località).

Questa entità verrà popolata in automatico alla partenza dell'autobus e andrà a memorizzare la data, l'ora di partenza ed un valore booleano "inCorso" che indica se la tratta si sta svolgendo o si è conclusa.

Tappe: rappresenta una posizione GPS rilevata dall'autobus. Ogni 100 metri percorsi, gli autobus in movimento comunicano con il DB, inviandogli informazioni riguardanti la tratta che stanno effettuando e di conseguenza si potranno conoscere tutte le informazioni ad essa collegate (la relativa latitudine, longitudine e l'ora di registrazione della posizione).

Grazie alle coordinate GPS e alle informazioni sulla linea e fermate, potrà essere individuata la posizione del pullman tra due fermate. Questo è necessario per individuare ritardi o anticipi rispetto agli orari ideali calcolabili tramite l'orario di partenza di una linea e gli offset di tutte le fermate.

Questa operazione verrà effettuata nel seguente modo:

1. Confronto della posizione GPS della tappa con quella della fermata successiva (associazione esistente tra tappe e fermate)

2. Se combaciano (con le appropriate approssimazioni) la tappa precedente assume il valore di quella successiva e quest'ultima punterà alla prossima fermata della linea (questa operazione serve per individuare l'autobus tra due fermate)

3. Ricaviamo il ritardo alla partenza della Tratta rispetto al relativo orario di partenza facendo: *"OffsetTratta = Tratte.OraPartenza - Orari.Orapartenza"*

4. Recuperiamo l'offset ideale della fermata appena superata dall'inizio della linea.

5. Recuperiamo l'offset della tappa rispetto all'inizio della tratta

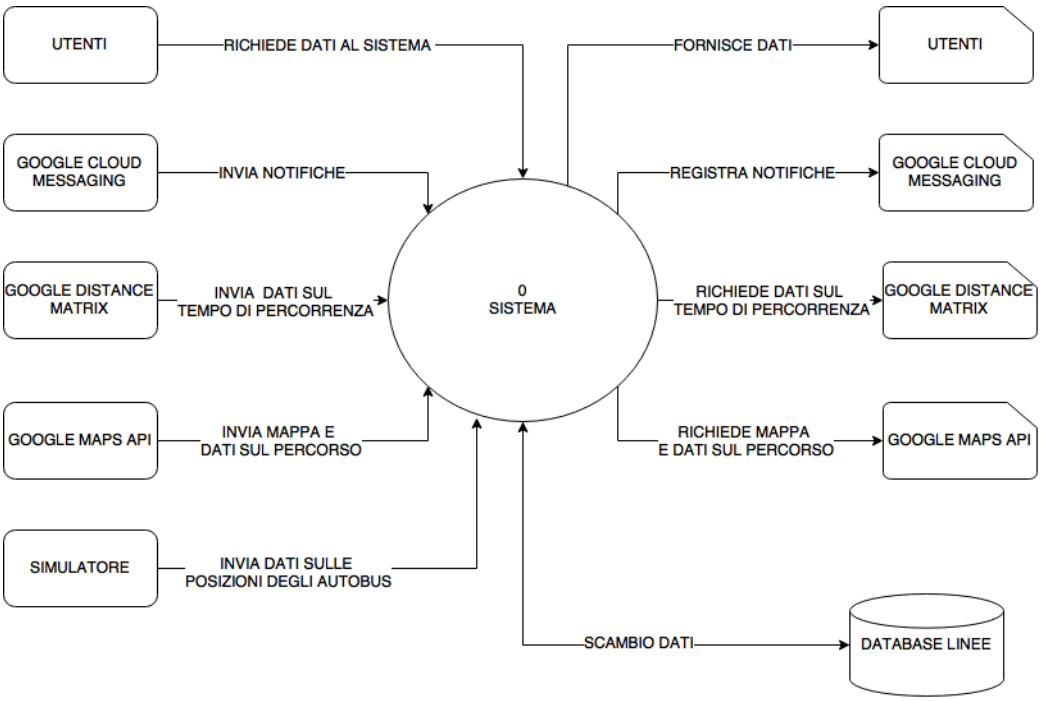
6. Il ritardo sarà uguale a *" OffsetTratta + (OffsetTappa – OffsetFermata) "*

Notifiche: Orari e Fermate saranno collegati da una relazione N-N che rappresenta una notifica registrata da uno specifico utente (attributo RegID) su una specifica linea, per una specifica fermata. Memorizza anche quanto tempo prima del passaggio dell'autobus ad una particolare fermata è necessario inviare la notifica.

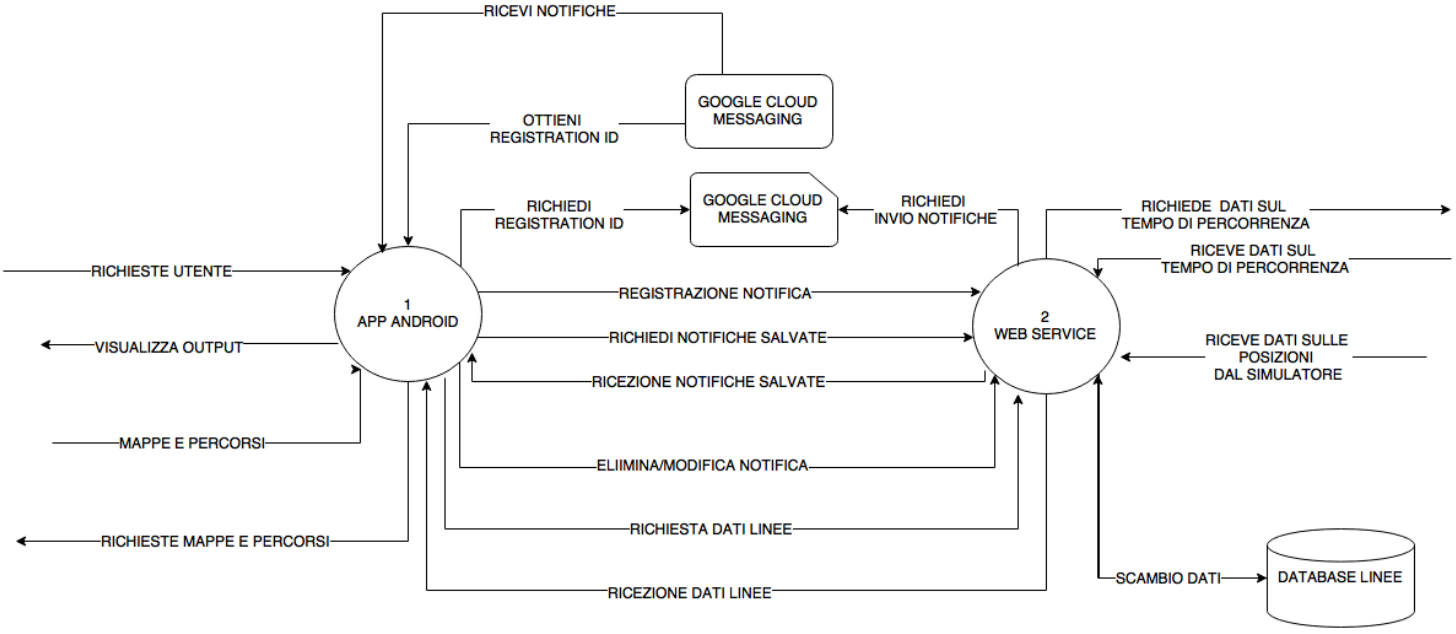
DFD (data flow diagram)

Di seguito riportiamo l'analisi funzionale del progetto, rappresentata utilizzando la notazione DFD in modo da far risaltare i processi e i flussi di dati tra di essi.

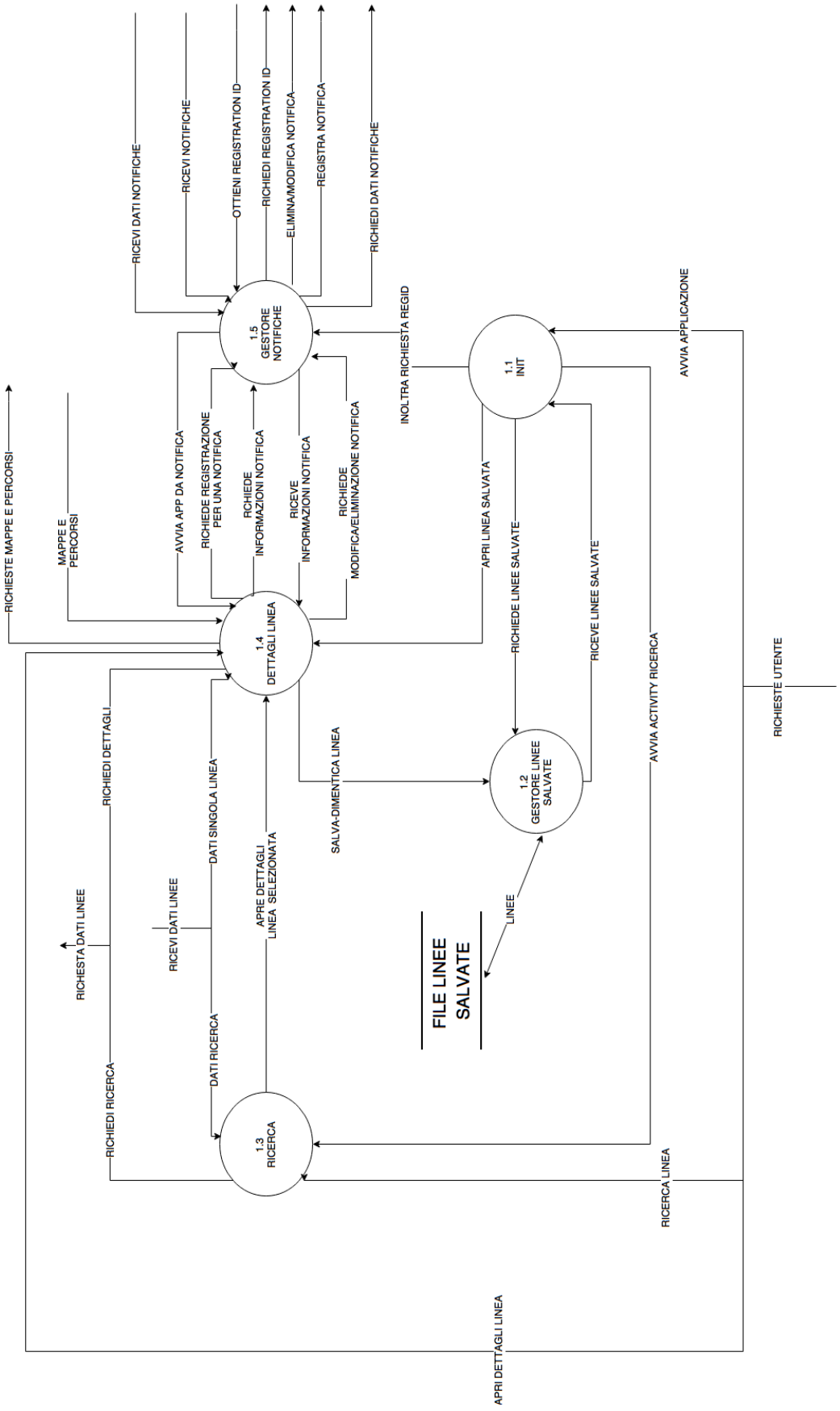
Diagramma di contesto



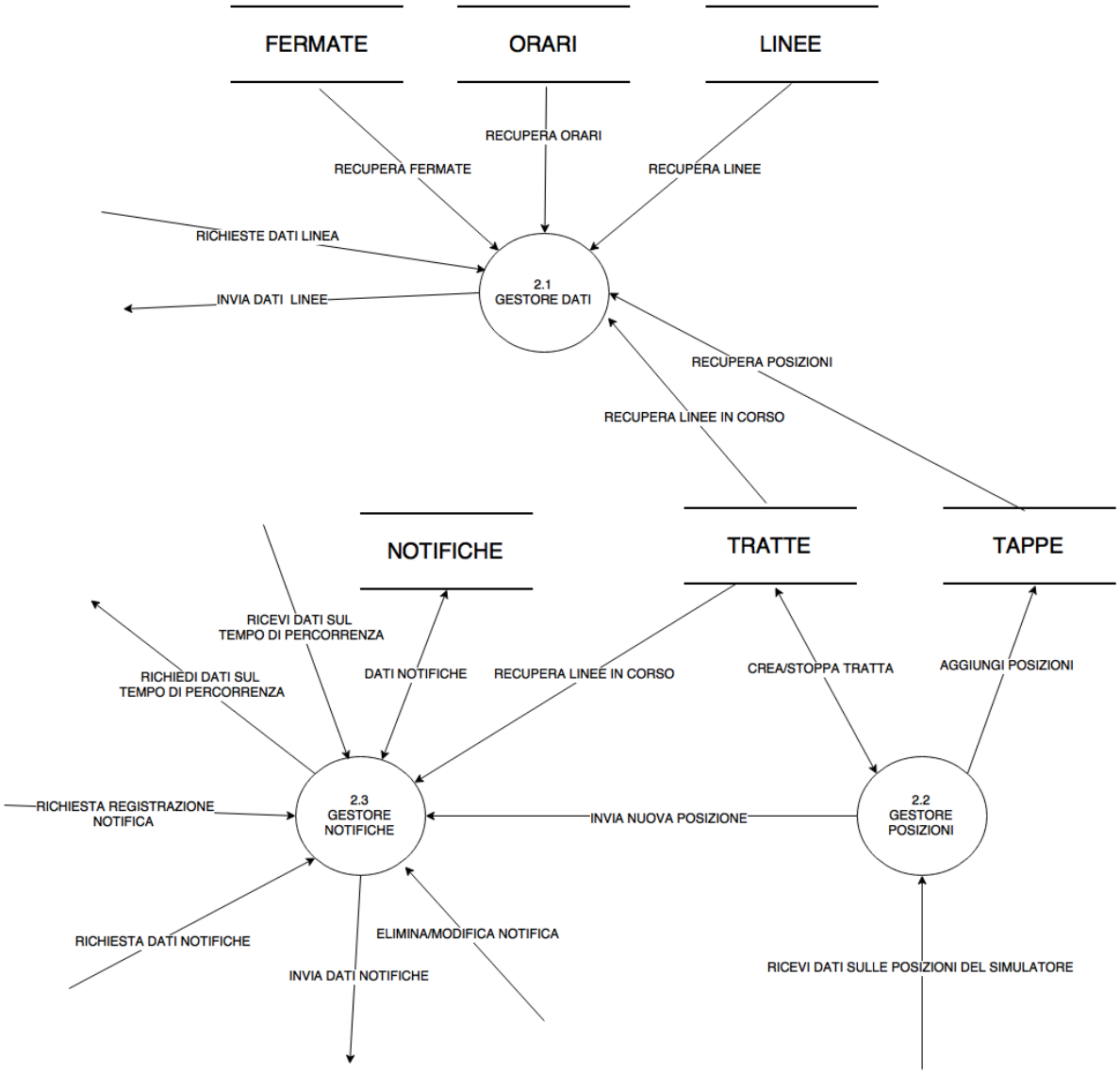
Prima esplosione



Seconda esplosione (parte 1)



Seconda esplosione (parte 2)



Specifiche finali

Nome: SplashScreen

Livello: 1.1.1

Flussi input: richiesta avvio applicazione

Flussi output: avvio Main Activity

Descrizione: punto di partenza dell'applicazione, consiste in una classica splash screen di caricamento in cui viene visualizzato il logo e viene lanciata in background la richiesta di un registration ID a GOOGLE CLOUD MESSAGING, con il quale sarà possibile (in seguito) gestire le notifiche. L'applicazione attende finché non viene ottenuto il registration ID (modulo 1.5.1), successivamente viene lanciata la MainActivity.

Nome: MainActivity

Livello: 1.1.2

Flussi input: interazioni utente

Flussi output: avvio altre componenti del progetto

Descrizione: è l'*activity* principale attraverso la quale l'utente può accedere (direttamente o indirettamente) a tutte le funzionalità dell'applicazione. In particolare visualizza la lista delle linee preferite permettendo di aprirne il dettaglio. Permette di avviare la ricerca di nuove linee, di gestire le notifiche a cui l'utente si è registrato e di condividere l'applicazione.

Nome: Fetcher linee locali

Livello: 1.2.1

Flussi input: richieste da parte degli altri moduli del progetto

Flussi output: elenco delle linee preferite dall'utente

Descrizione: restituisce l'elenco delle linee preferite dall'utente leggendo il relativo file di testo (.txt).

Nome: Salvataggio/eliminazione linee preferite

Livello: 1.2.2

Flussi input: informazioni linea da salvare o rimuovere

Flussi output: assente

Descrizione: aggiunge o rimuove le informazioni dal file delle linee salvate.

Nome: Fetcher linee

Livello: 1.3.1

Flussi input: richiesta ricerca linee (località partenza e località arrivo)

Flussi output: stringa JSON rappresentante le linee trovate

Descrizione: nel momento in cui viene avviata una ricerca il Fetcher avvia un thread in background che invia una richiesta contenente località di partenza e arrivo delle linee da ricercare al modulo 2.1.1 (vedi dopo), quest'ultimo elaborerà la richiesta e restituirà un oggetto JSON rappresentante il risultato della query e contente le informazioni riguardanti le linee trovate. A questo punto la stringa JSON viene passata al 1.3.2 descritto di seguito.

Nome: Gestore layout linee

Livello: 1.3.2

Flussi input: oggetto JSON rappresentante le linee da visualizzare

Flussi output: visualizza informazioni linee trovate

Descrizione: questo modulo riceve una stringa JSON rappresentante delle linee contenute sul DB lato server. Il suo compito è quello di effettuare il parsing della stringa convertendola in una lista di oggetti "Linea" e associarla ad un *adapter* che verrà a sua volta assegnato ad un *recyclerview* in modo che possano essere visualizzati a schermo dei componenti *cardview* contenenti informazioni sulle linee stesse (Nome-Linea, Ora partenza, Ditta e stato della linea). Selezionando una *cardview* verrà aperto il dettaglio della relativa linea.

Nome: Gestore Caselle di ricerca

Livello: 1.3.3

Flussi input: valori "parziali" di località di partenza e arrivo

Flussi output: visualizza possibili completamenti degli input

Descrizione: questo modulo viene richiamato nel momento in cui l'utente inizia a inserire una località di partenza o di arrivo per poter avviare una ricerca. Il suo scopo è quello di suggerire dei possibili valori di località per completare l'inserimento in modo da velocizzarlo e rendere l'utente a conoscenza delle località effettivamente registrate sul database.

Nel momento in cui viene caricata la pagina, viene effettuata una richiesta (su un thread separato) al server, il quale effettuerà una query sul database e restituirà tutti i nomi delle località e delle fermate.
Nel momento in cui viene generato un evento di modifica del contenuto della *AutoCompleteTextView* il sistema mostrerà su un menu a tendina i completamenti più pertinenti con i valori inseriti.

Nome: Fetcher fermate linea

Livello: 1.4.1

Flussi input: ID Linea di cui ottenere le fermate

Flussi output: visualizza lista fermate

Descrizione: nel momento in cui questo modulo viene richiamato, esso effettua una richiesta al server (2.1.2) su un thread separato. L'unico parametro della richiesta sarà l'ID della linea di cui ottenere le fermate. Il server restituirà il prima possibile una stringa JSON contente le informazioni sulle fermate richieste (Nome Fermata, Nome Località e ora di arrivo vengono anche recuperate le informazioni riguardanti la posizione di ogni fermata, esse non verranno visualizzate, ma passate ad il modulo 1.4.4 che provvederà a visualizzarle sulla mappa). A questo punto la stringa verrà parsificata e i dati riguardanti le fermate verranno visualizzati in un *recyclerview* in modo simile a quanto descritto in precedenza per le linee. Effettuando un "long click" su una fermata sarà possibile anche accedere alla gestione delle notifiche per quella fermata su quella specifica linea (moduli 1.5.2 e 1.5.3).

Nome: Fetcher informazioni linea in corso

Livello: 1.4.2

Flussi input: ID Linea di cui conoscere lo stato

Flussi output: visualizza stato della linea

Descrizione: nel momento in cui questo modulo viene richiamato effettua una richiesta al server (2.1.3) su un thread separato. L'unico parametro della richiesta sarà l'ID della linea di cui ottenere le stato. Il server restituirà poi una stringa JSON contente le informazioni sullo stato della linea richiesta. A questo punto la stringa verrà parsificata e se la linea è in corso i dati riguardanti lo stato della tratta (Autobus, Autista e ritardo accumulato) verranno visualizzati in una *"cardview"* sopra le fermate. Lo stato non si aggiorna automaticamente (per risparmiare dati internet), ma l'utente può aggiornarlo attraverso un apposito pulsante nella barra di azione.

Nome: Gestore linee salvate

Livello: 1.4.3

Flussi input: ID Linea da aggiungere/ rimuovere al file delle linee salvate

Flussi output: inoltro richiesta modulo 1.2.2

Descrizione: attraverso un pulsante nella barra di azione l'utente può decidere se aggiungere o rimuovere la linea corrente dai preferiti, si appoggerà sul modulo 1.2.2 per il salvataggio vero e proprio.

Nome: Gestore mappa

Livello: 1.4.4

Flussi input: posizioni fermate della linea da visualizzare e ID Linea attualmente in corso (opzionale)

Flussi output: visualizza percorso della linea e la posizione dell'autobus (se in movimento) sulla mappa

Descrizione: Questo modulo si occupa di disegnare sulla mappa (fornita dalla GOOGLE MAPS API) le fermate che compongono la linea corrente e di tracciare il percorso della linea dalla partenza all'arrivo. Viene inoltre avviato un thread secondario che interroga in continuazione il server (modulo 2.1.4) per verificare lo stato della linea e, se la linea è in corso, recuperare la posizione dell'autobus in modo da poterlo visualizzare sulla mappa, eventualmente aggiornando la posizione precedente.

Nome: Registrazione client su GCM (GOOGLE CLOUD MESSAGING)

Livello: 1.5.1

Flussi input: richiesta GCM (gestita automaticamente dal sistema)

Flussi output: Registration ID

Descrizione: La funzione di questo modulo è quella di fare richiesta ai server GCM di un registration ID univoco attraverso il quale sarà possibile ricevere notifiche push dal server GCM.

Nome: Registrazione notifica

Livello: 1.5.2

Flussi input: Registration ID, ID linea da registrare, ID fermata da registrare, tempo anticipo notifica

Flussi output: inoltro richiesta al modulo 2.3.1

Descrizione: La funzione di questo modulo è quella di richiedere al server (modulo 2.3.3) di registrare una notifica sul database indicando il registration ID del dispositivo (con il quale il GCM sarà in grado di inviare effettivamente la notifica), l'ID della linea su cui la notifica è attiva e la fermata che interesserà il passaggio dell'autobus (in particolare la notifica verrà inviata quando il tempo per raggiungere la fermata sarà inferiore o uguale al tempo specificato come flusso di input).

Nome: Rimozione notifica

Livello: 1.5.3

Flussi input: Registration ID, ID linea registrata, ID fermata registrata

Flussi output: inoltro richiesta al modulo 2.3.2

Descrizione: La funzione di questo modulo è quella di richiedere al server di rimuovere una notifica dal database indicando il registration ID del dispositivo, l'ID della linea su cui la notifica è attiva e la fermata che interessata dal passaggio dell'autobus.

Nome: Aggiornamento notifica

Livello: 1.5.4

Flussi input: Registration ID, ID linea registrata, ID fermata registrata, nuovo tempo

Flussi output: inoltro richiesta al modulo 2.3.3

Descrizione: La funzione di questo modulo è quella di richiedere al server di modificare una notifica dal database indicando il registration ID del dispositivo, l'ID della linea su cui la notifica è attiva e la fermata che interessata dal passaggio dell'autobus.

La modifica non interesserà tutti i campi, ma è possibile solo variare il tempo di "anticipo" della notifica.

Nome: Ricettore Notifica

Livello: 1.5.5

Flussi input: dati della notifica inviati dal GCM

Flussi output: apertura del dettaglio della linea

Descrizione: il compito di questo modulo è quello di visualizzare la notifica ricevuta dal GCM utilizzando un *BroadcastReceive*r e di riprendere il contesto dell'applicazione (in seguito al click dell'utente sul riquadro della notifica) visualizzando l'*activity* di dettaglio della linea a cui l'utente si è registrato (e per cui ha ricevuto la notifica).

Nome: Fetcher linee lato server

Livello: 2.1.1

Flussi input: località partenza e località arrivo

Flussi output: stringa JSON rappresentante le linee trovate

Descrizione: recupera località di partenza e di arrivo dall'header della richiesta HTTP(POST), esegue la query sul database per recuperare i dati sulle linee corrette (IDOrario, Nome-Linea, Ora partenza, Ditta e stato della linea), codifica il risultato in formato JSON e lo restituisce al client.

Nome: Fetcher fermate lato server

Livello: 2.1.2

Flussi input: ID linea di cui ottenere le fermate

Flussi output: stringa JSON rappresentante le fermate trovate

Descrizione: recupera l'ID della linea dall'header della richiesta HTTP(POST), esegue la query sul database per recuperare i dati sulle fermate corrette (IDFermata, Nome Fermata, Nome Località, ora di arrivo, Latitudine, Longitudine), codifica il risultato in formato JSON e lo restituisce al client.

Nome: Fetcher linea in corso lato server

Livello: 2.1.3

Flussi input: ID linea di cui ottenere lo stato

Flussi output: stringa JSON rappresentante lo stato della linea

Descrizione: recupera l'ID della linea dall'header della richiesta HTTP(POST), esegue la query sul database per verificare che la linea sia in corso e in tal caso recuperare i dati sul suo stato corrente (Autobus, Autista e ritardo accumulato), codifica il risultato in formato JSON e lo restituisce al client.

Il ritardo accumulato verrà calcolato facendo:

(OraPartenzaTratta – OraPartenzaLinea) + (OffsetFermataCorrente - OffsetFermataIdeale)

Dove l'ora di partenza della tratta è quella effettiva di partenza dell'autobus (per quel particolare servizio) mentre l'ora di partenza della linea è quella a cui il pullman dovrebbe partire secondo l'orario deciso dalla ditta.

La fermata corrente è quella più vicina (distanza dalla fermata <100 m) all'autobus in movimento.
L'offset della fermata corrente è il tempo trascorso dalla partenza della tratta (cioè *OffsetFermataCorrente*) mentre l'offset ideale della fermata è, come dice il nome stesso, il tempo che dovrebbe trascorrere rispetto all' ora di partenza della linea nel caso in cui non vi sia alcun ritardo.

Nome: Fetcher posizione linea in corso

Livello: 2.1.4

Flussi input: ID linea di cui ottenere lo stato

Flussi output: stringa JSON rappresentante lo stato della linea

Descrizione: recupera l'ID della linea dall'header della richiesta HTTP(POST), esegue la query sul database per verificare che la linea sia in corso e in tal caso recuperare i dati sul sulla posizione corrente dell'autobus (latitudine e longitudine), codifica il risultato in formato JSON e lo restituisce al client.

Nome: Creazione tratta

Livello: 2.2.1

Flussi input: Informazioni tratta (IDOrario, Ritardo, IDautobus, IDAutista)

Flussi output: IDTratta

Descrizione: recupera i dati dall'header della richiesta HTTP(POST), esegue la query sul database per creare una nuova tratta (una tratta è un singolo viaggio di una linea) impostandone lo stato a "in corso". La richiesta viene effettuata dal simulatore nel momento in cui facciamo partire un autobus. Restituisce al simulatore l'id della tratta appena creata (generato automaticamente da MYSQL).

Nome: Terminazione tratta

Livello: 2.2.2

Flussi input: IDTratta

Flussi output: assenti

Descrizione: recupera l'ID della tratta dall'header della richiesta HTTP(POST), esegue la query sul database per settare lo stato della tratta a "non in corso". La richiesta viene effettuata dal simulatore nel momento in cui termina la tratta.

Nome: Aggiunta Tappa

Livello: 2.2.3

Flussi input: IDTratta, Latitudine, Longitudine, Offset

Flussi output: richiama il gestore delle notifiche (2.3.4)

Descrizione: recupera i dati dall'header della richiesta HTTP(POST), esegue la query sul database per creare una nuova tappa. Una tappa rappresenta una posizione (latitudine, longitudine) di un dato autobus che sta percorrendo una tratta (IDTratta) in un determinato momento del tempo (Offset, ovvero il numero di millisecondi trascorsi dall'inizio della tratta al rilevamento della posizione). La richiesta viene effettuata dal simulatore nel momento in cui simula lo spostamento di un autobus. Al termine viene richiamato il gestore delle notifiche che deciderà se inviare o meno delle notifiche ai vari client

Nome: Registrazione notifica lato server

Livello: 2.3.1

Flussi input: Registration ID, ID linea da registrare, ID fermata da registrare, tempo anticipo notifica

Flussi output: assenti

Descrizione: recupera i dati dall'header della richiesta HTTP(POST), esegue la query sul database per creare una entry nella tabella notifiche. Questa tabella sarà consultata dal modulo 2.3.4 per determinare a quali client inviare notifiche.

Nome: Eliminazione notifica lato server

Livello: 2.3.2

Flussi input: Registration ID, ID linea registrata, ID fermata registrata

Flussi output: assenti

Descrizione: La funzione di questo modulo è quella di rimuovere una notifica dal database indicando il registration ID del dispositivo, l'ID della linea su cui la notifica è attiva e la fermata che interessata dal passaggio dell'autobus.

Nome: Aggiornamento notifica lato server

Livello: 2.3.3

Flussi input: Registration ID, ID linea registrata, ID fermata registrata, nuovo tempo

Flussi output: inoltro richiesta al modulo 2.3.3

Descrizione: La funzione di questo modulo è quella di modificare una notifica sul database indicando il registration ID del dispositivo, l'ID della linea su cui la notifica è attiva e la fermata che interessata dal passaggio dell'autobus.

La modifica non interesserà tutti i campi, ma solamente il tempo di "anticipo" della notifica.

Nome: Invio notifiche

Livello: 2.3.4

Flussi input: Posizione (Latitudine, Longitudine) e IDTratta dell'ultima posizione inviata dal simulatore

Flussi output: richiesta ai server GCM di invio notifiche ai client (in base al registration ID)

Descrizione: il compito di questo modulo è quello di determinare a quali client inviare una notifica (attraverso il GCM) in base all'ultima posizione dell'autobus.

Per prima cosa vengono estratte dalla tabella delle notifiche solo quelle che interessano la tratta in corso (IDTratta, inviata dal simulatore). Successivamente per ognuno dei risultati della precedente query viene recuperato l'ID della fermata su cui è registrata quella notifica e, interrogando anche la tabella delle fermate, vengono recuperate latitudine e longitudine di quella fermata. A questo punto viene fatta una richiesta alle GOOGLE DISTANCE MATRIX API per conoscere il tempo di percorrenza tra le coordinate estratte dal database e le coordinate ricevute dal simulatore. Se il tempo calcolato dalle API di Google è minore o uguale a quello salvato nella tabella delle notifiche, allora verrà recuperato il registration ID e memorizzato in un array. Dopo che questo procedimento è stato effettuato per tutti i record interessati, verrà generata un'ultima richiesta ai server di GCM passandogli l'array con tutti i registration ID dei client interessati alla notifica e i dati riguardanti la linea (utilizzati dal modulo 1.5.5 per poter visualizzare i dettagli della linea).

Software di supporto

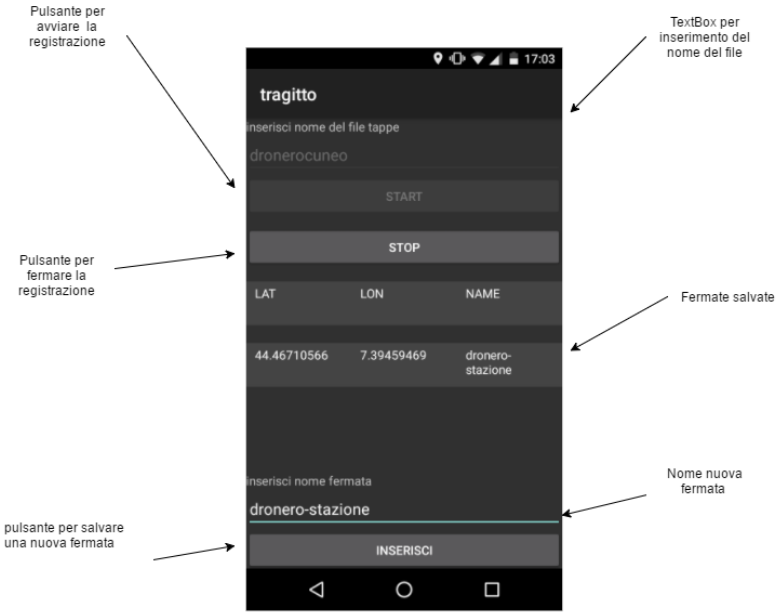
Tracker

Perché il nostro progetto funzioni occorre che sul database siano caricati alcuni dati di esempio. Per rendere il tutto il più realistico possibile, abbiamo realizzato una piccola applicazione android, chiamata tracker, la quale ha il compito di raccogliere i dati attraverso il sensore GPS dello smartphone e di inviarli al server che si preoccuperà di salvarli su dei file testuali per un accesso successivo (vedi simulatore).

L'esecuzione del tracker verrà lanciata all'interno di un autobus in movimento in modo che i dati recuperati rappresentino una linea reale ottenendo così sia una localizzazione geografica corretta, sia una temporizzazione (intesa come tempo di percorrenza da una posizione all'altra) affidabile in quanto relativa a delle linee effettivamente eseguite.

Questa applicazione è completamente slegata dal resto del progetto e ha due funzioni principali:

- Raccogliere informazioni riguardanti le coordinate delle fermate che appartengono ad una specifica linea (ed assegnare un nome alle fermate stesse).
- Raccogliere i dati relativi alle coordinate costituenti l'intero percorso di una linea. Nello specifico l'app permette di avviare un service in background il quale, attraverso l'oggetto di sistema *'LocationManager',* ad ogni spostamento dell'autobus (100 metri) si occuperà di recuperare le coordinate GPS rilevate e di ottenere il tempo trascorso (in millisecondi) dall'inizio della registrazione per poi inviare i dati al server dove verranno salvati su un file .txt. Questo file testuale sarà in seguito fruibile dal simulatore in modo che possa emulare lo spostamento di un autobus.



Simulatore

Come già detto, Nautibus si pone lo scopo di fornire agli utenti un insieme di dati che concernono gli autobus di linea, in particolare quelli riguardanti la posizione di un mezzo in un dato momento. In un'applicazione "reale" del nostro progetto, sarebbe perciò necessario fornire i veicoli di un dispositivo dotato di GPS e di connessione ad internet.

Tale dispositivo deve poter recuperare le coordinate dell'autobus in movimento e inviarle al server perché possano essere elaborate.
Le soluzioni attuabili per fare ciò sono molteplici. Ad esempio si potrebbe pensare di utilizzare il sistema integrato nelle postazioni di gestione della linea a disposizione degli autisti. Questa soluzione, sebbene sia la più "economica" dato che questo sistema è già presente sugli autobus, è tuttavia inattuabile in quanto tale piattaforma è chiusa, cioè a disposizione esclusiva delle aziende di autotrasporti e non integrata con altri tipi di servizi. Un'altra soluzione adottabile potrebbe essere quella di creare un sistema ad-hoc per svolgere soltanto i due compiti richiesti (GPS+internet) utilizzando un microcontrollore (ad esempio arduino) dotato dei moduli hardware e software necessari. La soluzione più semplice è però forse quella di dotare l'autista di uno smartphone su cui installare un'applicazione che svolga le funzioni richieste. Questa applicazione potrebbe tra l'altro essere una versione modificata del TRACKER illustrato in precedenza, in quanto il suo scopo sarebbe solo quello di recuperare le variazioni di posizione dell'autobus e di inviarle al server, gestendo però in maniera più dettagliata la continuità del servizio (di fondamentale importanza per un'applicazione che lavora tramite internet).

Purtroppo la creazione di un sistema del genere richiederebbe parecchio tempo, organizzazione e soprattutto soldi. Per questo motivo abbiamo deciso di realizzare un programma "simulatore" che svolgesse le stesse funzioni di un autobus reale, permettendo quindi di inviare al server una serie di coordinate "fittizie" rappresentanti lo spostamento del veicolo in un'area geografica. L'invio di tali coordinate dovrà però avvenire in maniera sincronizzata, ovvero il tempo trascorso tra l'invio di una posizione e l'invio della successiva dovrà essere uguale al tempo di percorrenza effettivo tra i due punti, in questo modo il simulatore sarà in grado di mimare completamente, sia a livello "geografico" che a livello "cronologico", il comportamento di un autobus.

Una volta ricevuti i dati il server elaborerà le informazioni permettendo all'applicazione android di fornire il servizio prefissato agli utenti.

Ovviamente è di fondamentale importanza che il server non faccia distinzioni a livello di gestione dei dati tra quelli ricevuti dal simulatore e quelli relativi ad una linea reale; ciò viene conseguito dal server esponendo all'esterno un'unica interfaccia per la ricezione delle informazioni. Da questo ne consegue che è possibile utilizzare il simulatore come un'alternativa agli autobus reali, per noi non accessibili sia a livello logistico che a livello espositivo. I dati relativi alle coordinate non potranno ovviamente essere reperiti direttamente da un sensore GPS, ma verranno recuperati leggendo i file .txt scritti dal tracker che, come spiegato in precedenza, contengono le posizioni reperite durante una tratta reale insieme ai tempi di percorrenza da una posizione alla successiva.

Abbiamo deciso di realizzare il simulatore attraverso un'interfaccia web scritta in HTML e javascript (sfruttando anche la libreria JQuery) in modo da evitare problemi di compatibilità tra piattaforme diverse.

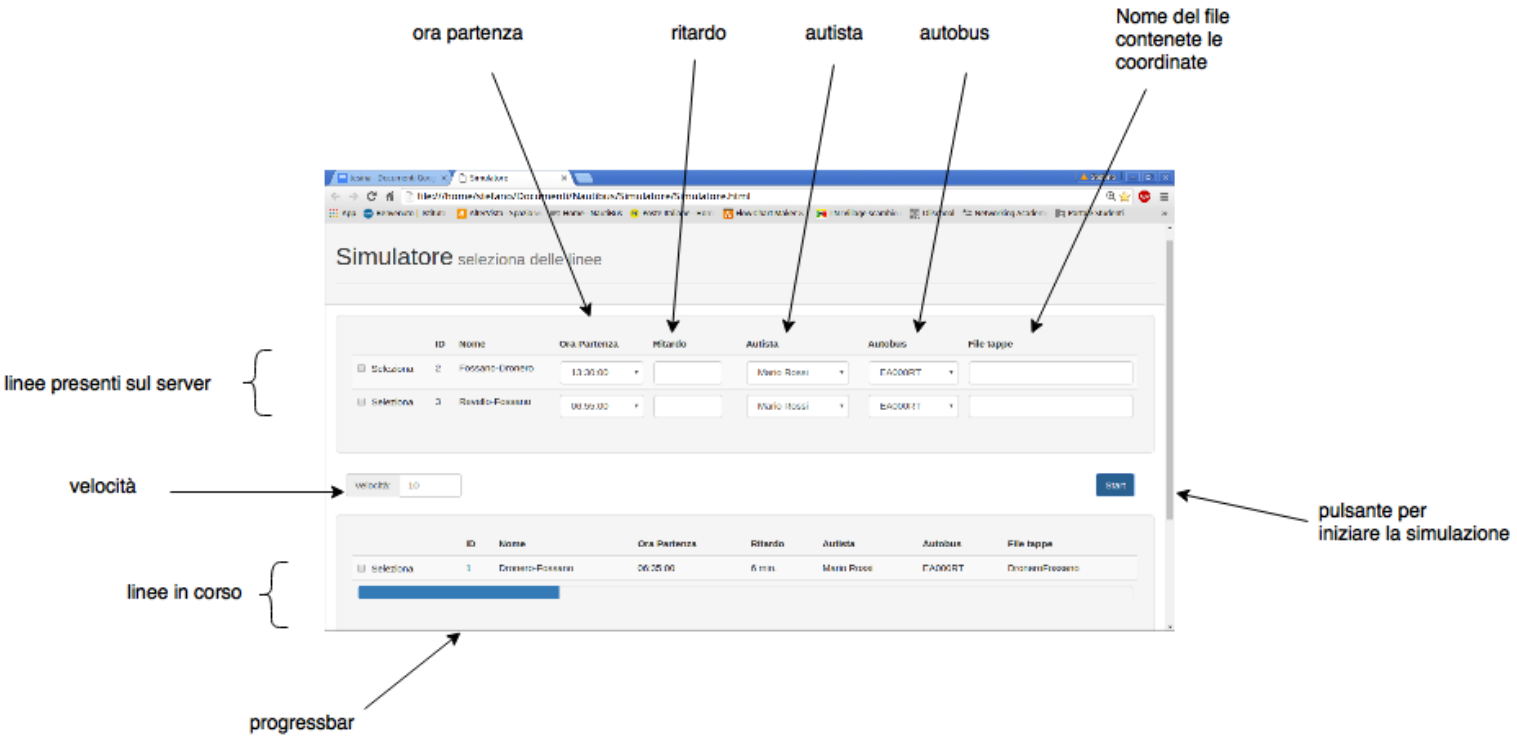
Il funzionamento del simulatore si basa sulla tecnologia AJAX (Asynchronous Javascript and XML) utilizzata per:

- Per reperire in modo asincrono i dati relativi alle linee presenti sul database
- Per inviare le posizioni lette dal file al server senza bloccare l'esecuzione del resto della pagina.

In particolare il simulatore permette di selezionare una o più linee e di "avviarle" specificandone dei parametri quali autista, autobus e ritardo alla partenza. Il parametro più importante è però il nome del file (sul server) da cui reperire le coordinate da inviare al server per simulare lo spostamento dell'autobus. Una volta premuto il pulsante "start" le linee precedentemente selezionate verranno fatte partire e inizierà la simulazione. La pagina tramite AJAX (quindi in modo asincrono rispetto al flusso di esecuzione principale) invierà una posizione alla volta, aspettando tra un invio e l'altro un tempo pari ai millisecondi indicati nel file in corrispondenza della posizione appena inviata. Tale misura di tempo, come già accennato in precedenza, indica il tempo trascorso dal rilevamento di una posizione al rilevamento della successiva. La pagina offre anche la possibilità di avere un feedback sul progresso di una linea attraverso una progressbar indicante il punto della simulazione a cui si è giunti.

Per concludere sono presenti un pulsante per bloccare una tratta in esecuzione e un'opzione per indicare la velocità di simulazione. Quest'ultimo parametro influirà sul tempo di attesa tra l'invio di una coordinata e quello della successiva, rendendo così la simulazione più o meno veloce a seconda delle necessità.

Di seguito è riportato uno screenshot della pagina del simulatore.



Di seguito, a titolo di esempio, riportiamo la porzione più importante del codice del simulatore ovvero la funzione *"StartSimulating"*.

Tale funzione, come si deduce dal nome, ha il compito di simulare lo spostamento dell'autobus inviando al server le coordinate lette dal file scritto utilizzando il tracker.

Una caratteristica importante di questa funzione è la sua ricorsività, ovvero dopo aver inviato una posizione e dopo aver atteso una certa quantità di tempo, essa richiama se stessa fino al raggiungimento della fine del file delle coordinate.

I parametri ricevuti dalla funzione sono:

- Data: vettore contenente i record letti dal file testuale
- progressbar: elemento DOM rappresentante la barra di progresso
- trattaObj: oggetto rappresentante la tratta in corso
- i: indica l'indice del vettore "data" a cui si trova la prossima posizione da inviare (usato nella ricorsione)
- curtime: tempo in millisecondi trascorso dall'inizio della linea. Questo parametro viene incrementato ad ogni ricorsione per tenere traccia del tempo trascorso tra l'invio di una posizione e la successiva.
- Count: questo parametro non interessa direttamente il funzionamento del simulatore, in quanto il suo scopo è solo quello di comunicare al server se è necessario gestire le notifiche. In particolare il server effettua una verifica ogni volta che riceve una nuova posizione (come spiegato in maniera dettagliata a pagina n). Nel processo di determinazione dell'invio vengono interpellate le Google Distance Matrix API per ottenere il tempo di percorrenza dalla posizione inviata a quella relativa alla notifica. Ne deriva che per ognuna di esse vengono fatte tante richieste all'API quante sono le posizioni vengono inviate. A livello teorico ciò non influisce sul funzionamento del servizio, ma il problema si presenta nel momento in cui si superano le 2500 richieste al giorno, in quanto, quando ciò accade, viene bloccato l'accesso all'API. Per risolvere questo problema (dato che il limite delle 2500 richieste viene superato facilmente) abbiamo deciso di verificare l'invio di notifiche non ad ogni posizione inviata ma solo ogni dieci. Per fare questo utilizziamo appunto il parametro count che viene incrementato ad ogni ricorsione; raggiunta quota 10, count viene azzerato e viene comunicato al server di verificare le notifiche.

*Le scritte verdi sono i commenti al codice

```
function StartSimulating(data,i,curtime,progressbar,factor,trattaObj,count){

    // Per prima cosa verifico se la tratta da simulare è stata avviata.
    if(trattaObj.stopped==false && trattaObj!=undefined){

        // Recupero la stringa contenete le coordinate da inviare.
        var row=data[i].split(";");

        // Preparo l'oggetto AJAX FormData per inviare le informazioni al server.
        var pos;
        pos = new FormData();
```

```
            // Aggiungo al FormData l'ID della tratta.
            pos.append('IDTratta', trattaObj.idTratta);

            // Aggiungo al FormData il tempo trascorso dall'inizio della linea.
            pos.append('Offset', curtime);

            // Aggiungo al FormData le coordinate da inviare.
            pos.append( 'Lat',row[0]);
            pos.append( 'Lon', row[1]);

            // Verifico se sono state inviate 10 posizioni.
           if(count==10)
           {
              // Se ciò avviene azzero count e comunico al server di verificare le
              // notifiche.
             pos.append('notification',"OK");
             count=0;
           }

            // Invio i dati al server attraverso una richiesta HTML POST utilizzando
            // AJAX.
            $.ajax({
            type: 'POST',
            url: "http://bustracker.altervista.org/Simulatore/AddTappa.php",
            data: pos,
            processData: false,
            contentType: false,
            dataType:"html",
            async:true,
            success:function(result){
                   console.log(result);
             }
           });

            // Incremento il contatore delle posizioni inviate.
            count++;

            // Recupero il tempo da attendere prima di inviare la prossima posizione.
            var wait=parseInt(row[2])

            // Aggiorno il tempo trascorso dall'inizio della linea.
            curtime+=wait;

            // Recupero la velocità di simulazione.
            var speed=parseInt($("#speed").val());

            if(!isNaN(speed) && speed>0)
                 // Calcolo il tempo di attesa rispetto alla velocità.
                 wait=wait/speed;

            // Aggiorno la progressbar.
            $(progressbar).attr("aria-valuenow",Math.round(factor*curtime));
            $(progressbar).attr("style","width:"+Math.round(factor*curtime)+"%");

            // Incremento indice del record sul file.
            i++;
```

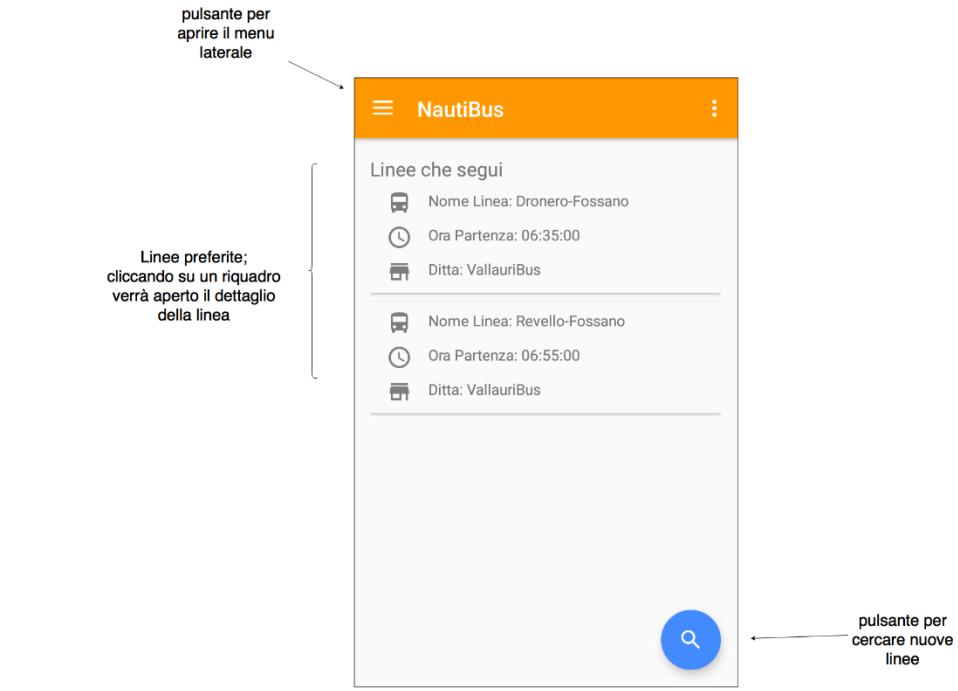
```
            // Verifico se sono arrivato alla fine del file.
            if(i< data.length)
            // Se sono ancora presenti coordinate richiamo ricorsivamente la funzione
             setTimeout(StartSimulating,wait,data,i,curtime, progressbar,factor
                    ,trattaObj,count);
            else
            {
               // Se il file è concluso, effettuo operazioni di reset grafico.

               $.each(tratte, function(i){
                if(tratte[i] === trattaObj) {
                    tratte.splice(i,1);
                    return false;
                }
               });
                $("#tratte tr").each(function(index,value){
                    if(index % 2==0)
                    {
                         if($(this).children().eq(1).children().eq(0)
                                   .text()==trattaObj.id)
                        {
                            $(this).parent().children().eq(index+1).remove();
                            $(this).remove();

                           // Stoppa tratta sul DB.
                            var params;
                            params = new FormData();
                            params.append( 'IDTratta', trattaObj.idTratta);

                           // Al termine comunico al server che la tratta è conclusa.
                            $.ajax({
                                type: 'POST',

                                url:"http://bustracker.altervista.org/Simulatore/
                                StopTratta.php",

                                data: params,
                                processData: false,
                                contentType: false,
                                dataType:"html",
                                async:false
                            });

                            $("#linee").append("<tr>"+trattaObj.row+"<tr/>");
                        }
                    }
                });
            }
        }
    }
```

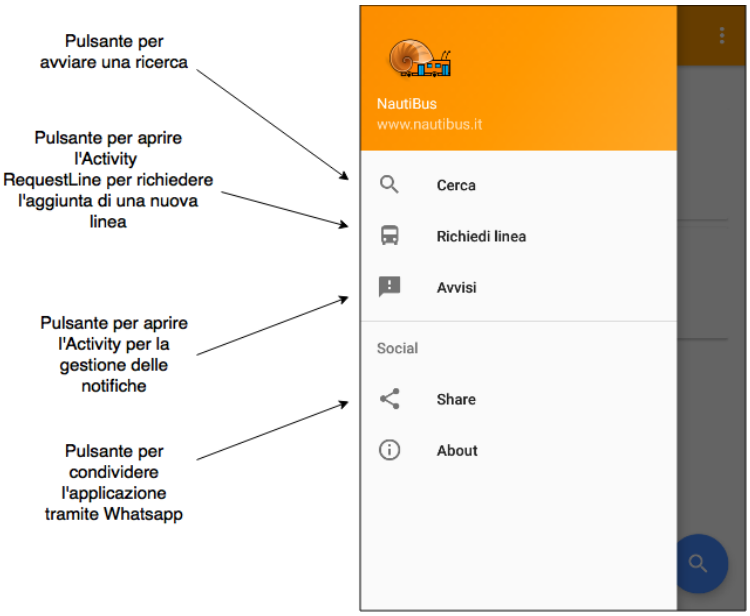
Interfaccia Utente

Di seguito riportiamo una descrizione dell'interfaccia grafica fornita per poter accedere alle funzionalità dell'applicazione. Nella spiegazione viene fatto spesso riferimento alle "Activity", ovvero delle componenti del sistema android che rappresentano una schermata della nostra applicazione, attraverso la quale l'utente può interagire con l'applicazione stessa.

MainActivity

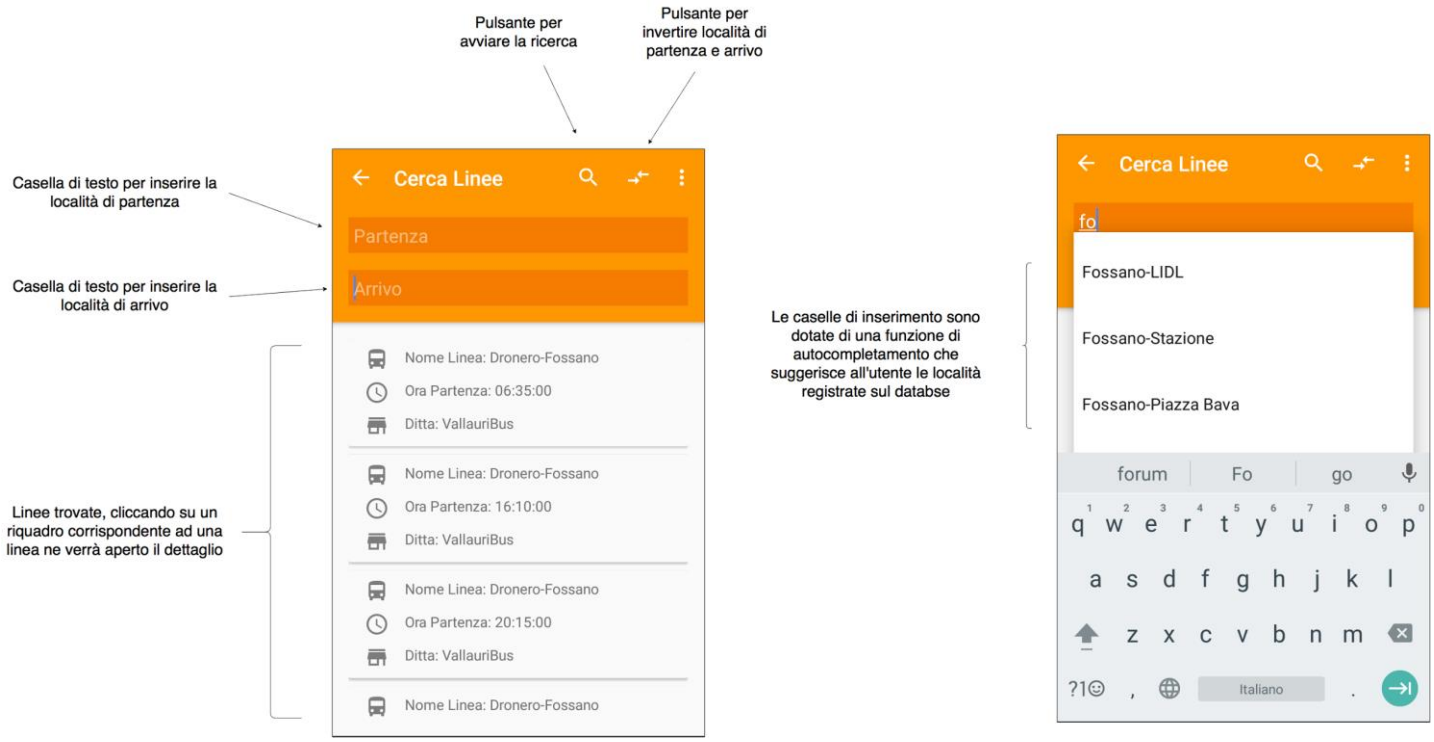
La MainActivity funge da "entry point" della nostra applicazione. Attraverso essa si possono raggiungere tutte le altre funzionalità.





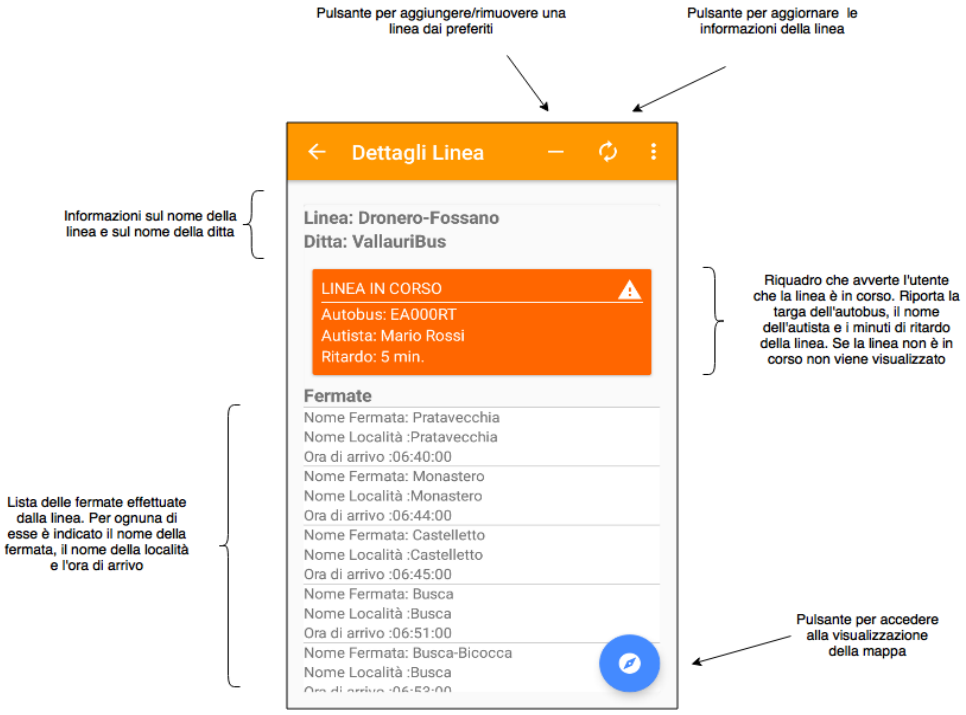
SearchActivity

Attraverso questa activity è possibile ricercare delle linee in base alle località di partenza e di arrivo desiderate. Dopo aver avviato la ricerca verranno elencatele le linee che transitano per le due località indicate; tramite un semplice click sul riquadro di una linea potrà esserne visualizzato il dettaglio.



DetailActivity

Attraverso questa activity è possibile visualizzare il dettaglio di una linea.

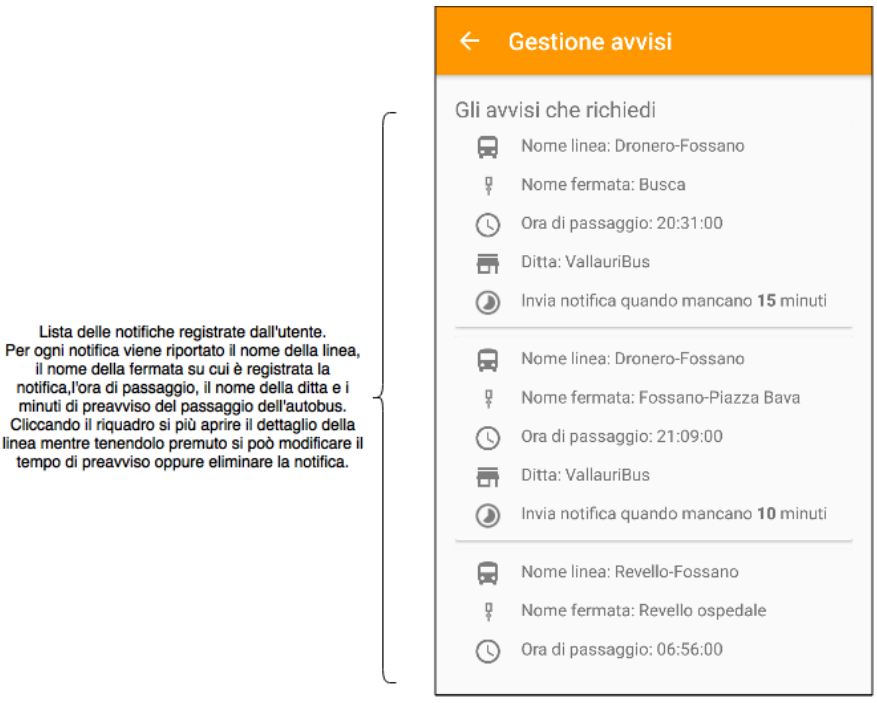


MapActivity

Questa activity permette di visualizzare il percorso di una linea e di localizzare la posizione attuale dell'autobus.

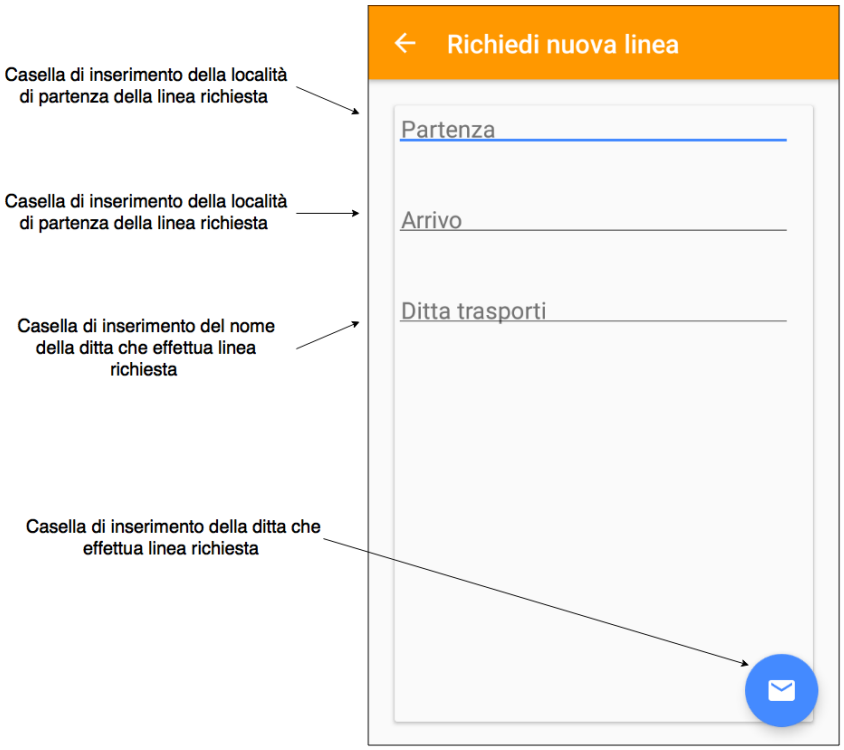
Activity Notifiche

Attraverso questa activity l'utente ha la capacità di gestire le notifiche a cui si è registrato. In particolare le notifiche possono essere visualizzate, eliminate o modificate.



MailActivity

Questa Activity offre la possibilità di richiedere all'amministratore del database l'inserimento di una linea attualmente non presente. La richiesta viene effettuata tramite una mail inviata ad un indirizzo gestito dall'amministratore stesso.



Gestione del percorso

Una delle parti più significative dell'applicazione è indubbiamente quella che si occupa di gestire la mappa e soprattutto di disegnare il percorso della linea sulla stessa. Per fare questo ci serviamo delle GOOGLE MAPS API offerte gratuitamente da Google. Quest'ultime ci permettono appunto di includere delle mappe all'interno della nostra applicazione e di rappresentare entità geografiche su di esse.
Nel nostro progetto accediamo alle suddette API in due frangenti differenti. Nel primo semplicemente carichiamo la mappa e disegniamo i marcatori per indicare la posizione delle fermate, mentre nel secondo accediamo al server Google per recuperare le coordinate del percorso e rappresentarlo in seguito sulla mappa (attraverso una polyline).

Le due funzionalità, sebbene possano essere contenute in un'unica classe, sono suddivise in due classi differenti. Abbiamo deciso di separare il codice in questo modo perché la richiesta del percorso deve essere eseguita su un thread separato in quanto interrogare il server di Google remoto e scaricare i dati potrebbe richiedere parecchio tempo, causando così un rallentamento nell'esecuzione della applicazione.

Una nota molto importante va fatta per la rappresentazione dell'autobus in movimento. Abbiamo già detto che intendiamo rappresentare la posizione dell'autobus attraverso un marker; il problema è che quest'ultimi sono oggetti statici e, una volta posizionati sulla mappa, le loro coordinate non possono essere modificate dinamicamente. Lo scopo di rappresentare l'autobus sulla mappa è però proprio quello di mostrare la sua variazione di posizione nel tempo, cosa che non può essere effettuata con un marker statico.

Per risolvere il problema occorre per cui:

- Ottenere periodicamente dal server la posizione corrente del mezzo
- Aggiornare la posizione dell'autobus sulla mappa eliminando il vecchio marker e creandone uno nuovo nella posizione corretta.

Queste operazioni andranno anch'esse svolte su un thread separato dato che la continua l'attesa di una nuova posizione intaserebbe il thread principale compromettendo l'usabilità dell'applicazione.

Il primo accesso alle Google maps api viene effettuato all'interno della MapActivity.

Di seguito è riportato il codice che carica la mappa e disegna i marker come spiegato sopra.

```
//La funzione OnMapReady viene richiamata automaticamente dopo che la mappa è stata
caricata.
@Override
public void onMapReady(GoogleMap googleMap) {

    //Recupero l'oggetto che rappresenta la mappa.
    mMap = googleMap;

    LatLng pos;


    //Ciclo per ogni fermata che appartiene alla linea richiesta (recuperate
    //in precedenza).
    for(int i=0;i<fermate.size();i++){
```

```
        //Recupero le coordinate della fermata corrente.
        pos=new LatLng(Double.parseDouble(fermate.get(i).getLat()),
                       Double.parseDouble(fermate.get(i).getLon()));

        //Aggiungo il marker per segnalare la fermata su un punto della mappa.
        mMap.addMarker(new MarkerOptions().
             position(pos).
             title(fermate.get(i).getLocalita()).
             snippet(fermate.get(i).getNome() + "\nOra di Arrivo: " +
             fermate.get(i).getOra()).
             icon(BitmapDescriptorFactory.
                   defaultMarker(BitmapDescriptorFactory.HUE_AZURE)));
    }
        // Richiamo l'AsyncTask GetDirections che avrà il compito di disegnare
        // il percorso.
        // Il suo funzionamento è descritto in seguito.
        new GetDirections(this, mMap,fermate).execute();

        //Creo e avvio il thread per disegnare l'autobus in movimento.
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    do {
//Creo una nuova richiesta per il server specificando l'ID della linea del
//cui mezzo si vuole conoscere la posizione.
String data = URLEncoder.encode("IDOrario", "UTF-8")+"=" +
URLEncoder.encode(args.getString("IDOrario"), "UTF-8");

URL url = new URL("http://bustracker.altervista.org/Nautibus/getBus.php");

                        URLConnection cnn = url.openConnection();
                        cnn.setDoOutput(true);

                        OutputStreamWriter sw = new
                        OutputStreamWriter(cnn.getOutputStream());

                        sw.write(data);
                        sw.flush();
                        sw.close();

                        BufferedReader sr = new BufferedReader(new
                        InputStreamReader(cnn.getInputStream()));

                        String supp = "";
                        String ret = "";
                        supp = sr.readLine();
                        while (supp != null) {
                            ret += supp;
                            supp = sr.readLine();
                        }
                        sr.close();

                        //Verifico se la richiesta è stata inviata e se viene
                        //ricevuta una risposta.
                        if (!ret.equals("") && !ret.equals("KO")) {

                            //In tal caso richiamo la funzione Drawbus che aggiorna
```

```
                            //il marker disegnandolo nella posizione corrente.
                            DrawBus(ret);

                            //Metto il thread in pausa per un secondo prima di
                            //aggiornare nuovamente la posizione.
                            Thread.sleep(1000);

                        }else {

                            //Se la linea è terminata (e non è per cui più in
                            //corso) elimino il marker richiamando la funzione
                            //RemoveBus.
                            RemoveBus();

                            //Attendo 10 secondi prima di ricontrollare lo
                            //stato della linea.
                            Thread.sleep(10000);
                        }

                    //Ripeto il ciclo di aggiornamento fino a che non viene
                    //stoppato dall'applicazione stessa.
                    }while(!stopBus);
                } catch (Exception ex) {

                }
            }}).start();
    }
    //Questa funzione elimina il marker relativo all'autobus.
    public void RemoveBus(){
        this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if(busMarker!=null)
                {
                    busMarker.remove();
                    busMarker=null;
                }
            }});
    }
//Questa funzione riceve come parametro le coordinate aggiornate della posizione
//dell'autobus. Il suo compito è quello di eliminare il vecchio marker e di disegnare
//quello nuovo.
    public void DrawBus(final String obj){
        this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                try{
                    JSONObject bus=new JSONObject(obj);
                    LatLng pos=new
                        LatLng(bus.getDouble("Latitude"),bus.getDouble("Longitude"));
                    RemoveBus();
                    busMarker=mMap.addMarker(new MarkerOptions().
                         position(pos).
                         title("Posizione Corrente").
                         icon(BitmapDescriptorFactory.fromResource(R.drawable.bus)));

                }catch(Exception ex){}
            } });
}
```

Di seguito è invece riportato il codice relativo alla classe GetDirections che, come spiegato in precedenza, opera su un thread secondario (GetDirections estende AsyncTask) in modo da non bloccare quello principale. Il compito di questa classe è quindi quello di richiedere al server di Google i dati relativi al percorso e di rappresentarlo sulla mappa attraverso una polyline.

```
public class GetDirections extends AsyncTask<String, String, String> {

    // Vettore contenente le fermate della linea.
    private Vector<Fermata> fermate;

    // Vettore per la memorizzazione delle coordinate del percorso.
    private List<LatLng> polyz;
    private List<LatLng> supppolyz;

    // Oggetto rappresentante la mappa.
    private GoogleMap map;

    // Puntatore alla MapActivity.
    private Activity parent;

    public GetDirections(Activity parent,GoogleMap map,Vector<Fermata> fermate){
        // Inizializzo semplicemente la classe.
        this.parent=parent;
        this.map=map;
        this.fermate=fermate;
        this.polyz=new ArrayList<LatLng>();
    }

    // Codice eseguito in background su un thread secondario
    protected String doInBackground(String... args) {

        if(fermate.size()>=2) {

            // Ciclo per tante volte quanto il numero di fermate della linea
diminuito
            // di uno.
            for(int i=0;i<fermate.size()-1;i++)
            {

                // Recupero la fermata attuale e quella successiva
                Fermata start =fermate.get(i);
                Fermata stop=fermate.get(i+1);

                // Instauro una connessione con i server di google maps ed invio una
                // richiesta HTTP per ottenere il percorso da effettuare per
                // raggiungere la seconda fermata (stop) partendo dalla prima
(start).
                // Questo procedimento verrà ripetuto per ogni coppia di fermate
                // (precedente e successiva) in modo che alla fine possa essere
                // ricostruito il percorso completo

                String stringUrl =
                        "http://maps.googleapis.com/maps/api/directions/json?origin="
                        + start.getLat()
                        + ","
                        + start.getLon()
```

```
                    + "&destination="
                    + stop.getLat()
                    + ","
                    + stop.getLon()
                    +"&sensor=false";

            StringBuilder response = new StringBuilder();
            try {
                URL url = new URL(stringUrl);

                HttpURLConnection httpconn =
                (HttpURLConnection)url.openConnection();

                if (httpconn.getResponseCode() == HttpURLConnection.HTTP_OK) {

                    BufferedReader input = new BufferedReader(
                    New InputStreamReader(httpconn.getInputStream()),8192);

                    String strLine = null;
                    while ((strLine = input.readLine()) != null) {
                        response.append(strLine);
                    }
                    input.close();
                }

                // A questo punto la richiesta è stata effettuata ed è stata
                // ricevuta la risposta in formato JSON che andrà parsificata
                // utilizzando la classe JSONObject.

                String jsonOutput = response.toString();
                JSONObject jsonObject = new JSONObject(jsonOutput);
                JSONArray routesArray = jsonObject.getJSONArray("routes");

                // Se il risultato contiene delle coordinate
                if(routesArray.length()>0) {
                    JSONObject route = routesArray.getJSONObject(0);
                    JSONObject poly = route.getJSONObject("overview_polyline");
                    String polyline = poly.getString("points");

                    // Una volta estratte le coordinate del percorso, esse
                    // andranno ulteriormente decodificate attraverso la funzione
                    // decodepoly, il cui codice è stato tralasciato in quanto di
                    // difficile spiegazione e scarso interesse (si limita
                    // soltanto ad effettuare delle operazioni sulla stringa
                    // estratta dall'oggetto JSON per poter estrarre un array di
                    // coordinate.

                    supppolyz = decodePoly(polyline);

                    // Aggiungo le coordinate al vettore globale.
                    polyz.addAll(supppolyz);

                }else

                    // Se il risultato non contiene delle coordinate decremento
                    // l'indice del ciclo for in modo che alla prossima
                    // iterazione venga effettuata una nuova richiesta tra le
                    // stesse fermate.
```

```
                        i--;

                } catch (Exception e) {}
            }
        }
        return null;
    }

   protected void onPostExecute(String file_url) {

       // Al termine della richiesta, dopo aver recuperato tutti i dati, recupero una
       // alla volta tutte le posizioni che costituiscono il percorso e le aggiungo
       // alla polyline che verrà disegnata sulla mappa.

       int i=0;
       LatLng pos;
       PolylineOptions opt=new PolylineOptions().width(8).color(Color.rgb(68, 138,
255));
       for ( i = 0; i < polyz.size() ; i++) {
           pos = polyz.get(i);
           opt.add(pos);
       }
       map.addPolyline(opt);
   }
}
```

Calcolo del ritardo

Un'altra funzione degna di nota, offerta dalla nostra applicazione, è quella del calcolo del ritardo di una linea in corso. Questa operazione è svolta lato server; l'applicazione android si limita a interrogare il web service per ottenere il valore del ritardo mentre il server, una volta ricevuta una richiesta, effettua il calcolo e restituisce il valore all'applicazione.

La spiegazione del procedimento di determinazione del ritardo è già stato dettagliatamente esposto a pagina n nell'ambito della descrizione della tabella "Tappe" del database.

Di seguito ci limitiamo per cui a riportare e commentare il codice PHP relativo al calcolo del ritardo.

```php
<?php header("Access-Control-Allow-Origin:*");

    // Verifico che sia impostato il parametro POST "IDOrario" che identifica l'ID
    // della linea di cui si vuole calcolare il ritardo.

    if(isset($_POST["IDOrario"])){

        // Recupero il suddetto parametro.
        $idOrario=$_POST["IDOrario"];

        // Apro la connessione con il database.
        $cnn=mysqli_connect("localhost","bustracker","","my_bustracker");

        // Verifico che non si siano verificati errori nell'apertura della
connessione.
        if(!mysqli_errno($cnn)){

                // Viene effettuata la prima query. Il suo scopo è quello di
recuperare
                // tutti i dati relativi ad una linea in corso che elenchiamo di
                // seguito:
                // -IDTratta
                // -Nome Autista
                // -Fermata appena superata

            $res=mysqli_query($cnn,"SELECT Tratte.IDTratta AS ID," .
                    "IDAutobus,Concat(Autisti.Nome,' ',Autisti.Cognome)AS Autista,".
                        "Tappe.IDPrima AS Fermata".

                        "FROM Tratte,Autisti,Tappe".

                        "WHERE Tratte.IDAutista=Autisti.IDAutista AND".
                        "Tappe.IDTratta=Tratte.IDTratta AND".
                        "Tratte.Data=CURDATE() AND".
                        "Tratte.InCorso=1 AND".
                        "Tratte.IDOrario={$_POST['IDOrario']}".

                        "GROUP BY".
                        "Tratte.IDTratta,".
                        "IDAutobus,".
                        "Autisti.Nome,".
                        "Autisti.Cognome,".
                        "Tratte.Ritardo,".
                       "Tappe.IDPrima".
                       "ORDER BY (MAX(Tappe.Offset)+Tratte.Ritardo) DESC LIMIT 1");
```

```
            // Verifico se la query ha restituito almeno un valore.
            if(mysqli_num_rows($res))
            {
               // A questo punto viene effettuata la seconda query che andrà
               // effettivamente a calcolare il ritardo.
               // Innanzitutto viene recuperato il ritardo alla partenza della
tratta.
               // Questo valore viene poi sommato all'offset rispetto all'inizio
della
               // tratta (dell'ultima posizione rilevata) e trasformato in
               // millisecondi.
               // SELECT MAX(Offset)-Tratte.Ritardo*60000
               // Infine viene sottratto a Percorsi.Offset, ovvero l'ora di arrivo
               // ideale alla fermata appena superata, il valore precedentemente
               // calcolato ottenendo così il ritardo complessivo della linea.

                $ritardo=mysqli_query($cnn,"SELECT Percorsi.Offset-" .
                 "(SELECT MAX(Offset)-Tratte.Ritardo*60000".
                 "FROM Tappe,Tratte".
                 "WHERE Tratte.IDTratta=Tappe.IDTratta AND".
                 "Tratte.IDTratta={$dati['ID']} AND".
                 "Tappe.IDDopo={$dati['Fermata']})".
                              "FROM Percorsi,Linee,Orari,Tratte".
                              "WHERE Percorsi.IDLinea=Linee.IDLinea AND".
                              "Linee.IDlinea=Orari.IDLinea AND".
                               "Orari.IDOrario=Tratte.IDOrario AND".
                              "Tratte.IDTratta={$dati['ID']} AND".
                              "Percorsi.IDFermata={$dati['Fermata']}");

                // Recupero il dato calcolato.
                $ritardoVet=mysqli_fetch_array($ritardo);

                // Inserisco il ritardo nell'oggetto da restituire.
                $dati["Ritardo"]=$ritardoVet[0];

                // Restituisco i dati in formato JSON.
                echo "".json_encode($dati);

            }else
                echo "KO";

        }
        mysqli_close($cnn);
    };
?>
```

Google Cloud Messaging

La funzionalità più innovativa del nostro progetto consiste nella possibilità di ricevere delle notifiche che avvisino preventivamente l'utente dell'arrivo di un autobus ad una fermata precedentemente selezionata. Ciò avviene attraverso il servizio GOOGLE CLOUD MESSAGING, una serie di API offerte gratuitamente da Google che permettono al server l'invio di notifiche push ai client che si sono registrati per riceverle.

Le notifiche possono essere gestite sostanzialmente in due modi:

- Il primo metodo è quello di creare una servizio in background lato client che interroghi periodicamente il server per scaricare eventuali messaggi. Questa soluzione è la più semplice da gestire, ma comporta un maggiore consumo di risorse in quanto il servizio di richiesta delle notifiche dovrà sempre essere mantenuto in memoria oltre ad occupare risorse computazionali in maniera continuata.
- La seconda alternativa è quella di adottare la cosiddetta metodologia "push", ovvero non è l'applicazione client a dover richiedere al server le notifiche, ma è quest'ultimo che automaticamente le inoltra ai vari client, senza che alcun modulo sia dell'applicazione sia in esecuzione. A questo punto non occorre più gestire un servizio in background, ma è sufficiente che l'applicazione rimanga in ascolto di eventuali messaggi utilizzando un "*BroadcastReceiver*", attraverso il quale sarà possibile generare la notifica e, una volta che l'utente farà click su quest'ultima, riprendere il contesto di esecuzione dell'applicazione.

Il "*BroadcastReceiver*" è un componente del sistema android che permette alle applicazioni a cui è collegato di effettuare determinate operazioni in seguito al verificarsi di particolari eventi di sistema; ad esempio l'arrivo di un SMS, il cambiamento di stato di carica della batteria, il riavvio del dispositivo oppure, come nel nostro caso, la ricezione di un messaggio proveniente da un server.

Il problema principale dell'implementazione delle notifiche push è che il sistema android non gestisce automaticamente il loro invio da parte di un server per uno specifico gruppo di client; fortunatamente Google è venuta incontro agli sviluppatori offrendo gratuitamente un gruppo di API chiamate GOOGLE CLOUD MESSAGING (GCM).

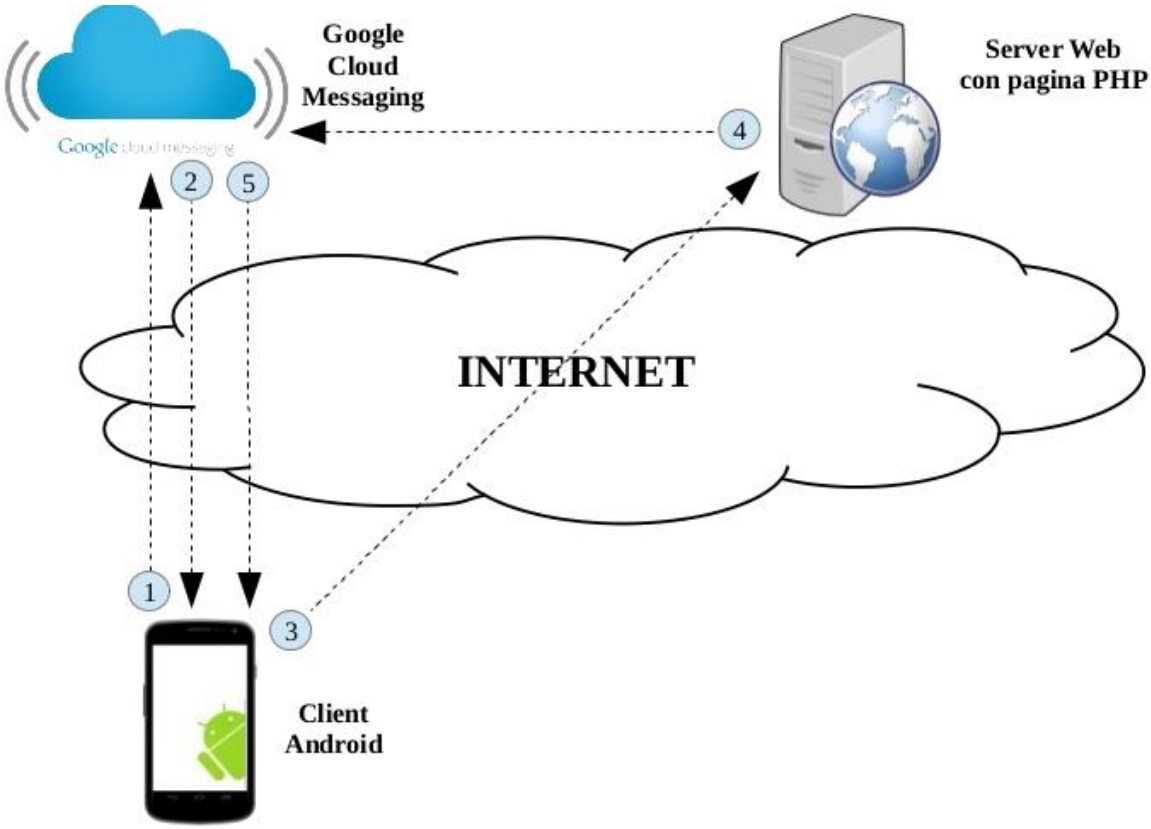
Attraverso le GCM è possibile tenere traccia dei dispositivi su cui la nostra applicazione è installata (ognuno dei quali è identificato univocamente attraverso un "Registration_ID" fornito da Google stessa) e quindi inviare notifiche mirate a ciascun client, appoggiandosi ad un server intermedio gestito da Google.

In pratica, la procedura che permette ad un client di ricevere notifiche push avviene secondo i seguenti passaggi:

1. L'applicazione richiede la registrazione al servizio GCM ai server di Google;
2. Il GCM service assegna un Registration_ID all'applicazione;
3. L'applicazione invia il proprio Registration_ID al nostro server, dove verrà salvato su un Database per un utilizzo successivo;
4. Quando il nostro server vuole inviare una notifica, genera il messaggio e lo inoltra ai server GCM assieme al Registration_ID ricevuto dall'applicazione nel punto precedente;
5. Il servizio GCM si si occuperà infine di individuare il client destinatario della modifica attraverso il Registration_ID (operazione possibile solo attraverso i complessi algoritmi di Google) e di inoltrargli la notifica.

Nota: non solo l'applicazione necessita di un ID, ma anche il server dovrà richiedere una chiave, denominata "*Server_Key*", per poter accedere al servizio GCM

Per completezza riportiamo in seguito uno schema che riassume quanto detto in precedenza.



Gestione Notifiche

Nel capitolo precedente abbiamo spiegato a grandi linee il funzionamento del servizio Google Cloud Messaging; passiamo ora a esporre la gestione dell'intero sistema delle notifiche, dalla registrazione del client, all'elaborazione lato server per finire con la ricezione della notifica dal client stesso.

Registrazione dell'applicazione al servizio GCM

Questa è la prima operazione che viene effettuata dall'applicazione nel contesto della SplashScreen. La MainActivity non verrà visualizzata finché il Registration_ID non è stato ottenuto. In realtà la richiesta del Registration_ID ai server di Google viene effettuata solo al primo avvio dopo l'installazione, in quanto una volta ottenuto esso non cambia per tutto il ciclo di vita dell'applicazione e per cui è sufficiente salvarlo offline per accedervi successivamente

```
// La richiesta di un Registration ID avviene attraverso internet, può per cui
// richiedere parecchio tempo e va quindi eseguita in un thread secondario.
public class RegisterAsync extends AsyncTask<String,Void,String> {

    private String SENDER_ID = "";
    private SplashScreen cx;
    private GoogleCloudMessaging gcm;

    public RegisterAsync(SplashScreen cx,String SENDER_ID) {
        super();
        // SENDER_ID dovrà contenere la server key relativa al nostro web service.
        this.SENDER_ID=SENDER_ID;
        this.cx=cx;
        // Recupero un riferimento all'API GoogleCloudMessaging per poter effettuare
        // la richiesta del REGISTRATION_ID.
        gcm=GoogleCloudMessaging.getInstance(this.cx);
    }

    @Override
    protected String doInBackground(String... params) {
        String regid="";
        String msg = "";
        try {
            // Effettuo la richiesta.
            InstanceID instanceID = InstanceID.getInstance(cx);

            // Recupero il registraton_ID, comunicando al GCM la server_key
            // corrispondente al server che gestirà il registration_ID stesso.
            regid = instanceID.getToken(SENDER_ID,
                    GoogleCloudMessaging.INSTANCE_ID_SCOPE, null);
        }
        catch (IOException ex)
        {
            return null;
        }
        return regid;
    }
}
```

Registrazione notifica

Per la registrazione di una notifica abbiamo invece deciso di non riportare nessun estratto di codice, in quanto essa consiste in una semplice richiesta HTTP POST simile ad altri esempi già esposti in questa tesi. Nel momento in cui l'utente decide di registrare una notifica su una fermata (cliccando su di essa nell'activity di dettaglio di una linea), gli viene offerta la possibilità di scegliere i minuti di preavviso della notifica (5,10,15 minuti) attraverso un menu a scelta multipla. Una volta confermata la registrazione la richiesta viene inviata specificando i seguenti parametri:

- RegID: Registration_ID che identifica il client (Ottenuto nel paragrafo precedente), utilizzato dal GCM per identificare il dispositivo su cui l'applicazione è installata;
- IDOrario: Identifica la linea su cui è registrata la notifica;
- IDFermata: Identifica la fermata (della linea specificata sopra) su cui è registrata la notifica;
- Tempo: Indica il tempo di preavviso della notifica .

Il server, dopo aver ricevuto i parametri POST, si limita semplicemente ad aggiungere un record nella tabella notifiche, popolandolo con i suddetti valori.

Invio notifche

A questo punto tocca al server determinare quando inviare le notifiche precedentemente registrate. Quest'operazione viene eseguita in una serie di passaggi, ripetuti per ogni nuova posizione inviata da qualsiasi autobus.

I Passaggi sono i seguenti:

1. Il Server riceve una nuova posizione da un autobus (dal simulatore);
2. Viene recuperata la linea a cui appartengono le coordinate appena inviate;
3. Vengono recuperate tutte le notifiche registrate sulla linea di cui sopra;
4. Per ciascuna delle notifiche vengono eseguite queste operazioni:
    4.1. Vengono recuperate le coordinate della fermata su cui è registrata la notifica;
    4.2. Viene calcolato il tempo di percorrenza tra le coordinate ricevute dall'autobus e quelle relative alla fermata su cui è registrata la notifica utilizzando le GOOGLE DISTANCE MATRIX API;
    4.3. Viene recuperato il tempo di preavviso della notifica;
    4.4. Se il tempo di percorrenza è minore o uguale al tempo di preavviso la notifica deve essere inviata;
    4.5. Se la verifica del punto precedenza ha un esito positivo viene interpellato il GCM che provvede ad inviare la notifica in base al Registration_ID associato alla notifica;
    4.6. Una volta che la notifica è stata inviata viene rimosso il relativo record sul database di modo che non possa più essere considerato successivamente.

Nota sulle Google Distance Matrix API: Questo servizio di google, a differenza di altri utilizzati nel progetto, non è completamente gratuito in quanto permette di effettuare soltanto 2500 richieste al giorno. Il problema è che effettuando il controllo delle notifiche per ogni posizione inviata, questo

limite viene facilmente superato considerando che una linea media invia circa 300 posizioni. Per questo abbiamo dovuto limitare il numero di richieste per poter fornire un servizio continuo. La soluzione adottata è stata quella di effettuare il controllo delle notifiche non per ogni singola posizione inviata ma soltanto una volta ogni 10 invii. Questo ovviamente influisce negativamente sulla precisione delle notifiche ma ci permette di rientrare nei limiti stabiliti da Google.

Di seguito riportiamo la porzione di codice PHP che effettua quanto descritto sopra

```
// Eseguo la query di inserimento della posizione ricevuta dall'autobus nel
// database.
if(mysqli_query($cnn,"INSERT INTO Tappe(IDTratta,Offset,IDPrima,IDDopo,Lat,Lon)
VALUES({$_POST["IDTratta"]},{$_POST["Offset"]},$prec,$succ,{$_POST["Lat"]},
{$_POST["Lon"]})"));
            {
                // Recupero l'ID della linea a cui appartiene la posizione appena
                // inserita nel database.
              $retIdOrario=mysqli_query($cnn,"SELECT Tratte.IDOrario FROM Tratte
                                        WHERE  Tratte.IDTratta={$_POST["IDTratta"]}");

                $nextID=mysqli_fetch_array($retIdOrario);
                $IDOrario=$nextID[0];
                if(isset($_POST["notification"])){

                 // Richiamo la funzione getNotification per recuperare le notifiche
                 // registrate sulla linea inserita.
                  $notArray=getNoficication($cnn,$_POST["Lat"],$_POST["Lon"],
                  $IDOrario,$succ,$_POST["IDTratta"]);


                  // Istanzio un oggetto GCM il cui scopo è quello di gestire
                  // l'inoltro delle notifiche ai server GCM.
                  $cloud=new GCM();

                  // Inoltro tutte le notifiche al GCM.
                  for($i=0;$i<sizeof($notArray);$i++){
                     $cloud->send_notification($notArray[$i]);
                   }
                }

                echo "OK";
            }
}else
    echo "KO";
```

Funzione getNotification

Il suo compito è quello di reperire le notifiche che devono essere inoltrate al GCM.
I parametri ricevuti sono:

- $conn: oggetto connessione al database Mysql;
- $lat e $lon: latitudine e longitudine dell'ultima posizione;
- $idOrario: ID della linea che si sta verificando;
- $nextf : Id della prossima fermata che l'autobus raggiugerà;
- $idt: ID della tratta.

```
function getNoficication($conn,$lat, $lon,$idOrario,$nextf,$idt){

        // Eseguo la query per recuperare le notifiche registrate sulla linea
corrente
        // ($idOrario).

        $notSupp=mysqli_query($conn,"SELECT REGISTRATION_ID AS RegID,".
                                "Fermate.Lat AS LAT,Fermate.Lon AS LON,".
                                "Notifiche.IDFermata AS IDF,".
                                "Tempo,Fermate.Nome AS Fermata".

                                 "FROM Notifiche,Fermate ".

                                 "WHERE Notifiche.IDFermata=Fermate.IDFermata".
                                 "AND Notifiche.IDOrario=$idOrario");

        $notificationArray=array();
        $tempo=0;
        $temponot=0;
        if(mysqli_num_rows($notSupp)>0)
        {
            // Ciclo per ogni notifica estratta dal database.
            while($not=mysqli_fetch_assoc($notSupp)){

                  // Recupero l'ID della fermata su cui è registrata la notifica.
                  $idf=$not["IDF"];

                     // getPrecFermate è una funzione che restituisce un'array
                     // contenente tutte le fermate di una llinea precedenti a
quella
                     // passata come secondo parametro ($idf).
                  $fermate=getPrecFermate($conn,$idf,$idt);

                     // Verifico che la fermata relativa alla notifica ($idf) non
sia
                     // già stata superata (cioè non si trovi tra la prima fermata e
                     // quella attuale).
                   if(in_array($nextf,$fermate)){

                     // Calcolo il tempo di percorrenza tra la posizione attuale e
                     // quella della fermata relativa alla notifica.
                     $tempo=getTime($lat, $lon, $not["LAT"], $not["LON"]);

                     // Recupero il tempo di preavviso della notifica.
                     $temponot=($not["Tempo"]/1000);
```

```
                    // Inserisco il messaggio della notifica all'interno di un
                    // oggetto apposito.
                    $notificationObj=new Notification($idOrario);
                    $notificationObj->IDFermata=$idf;
                    $notificationObj->Fermata=$not["Fermata"];
                    $notificationObj->Tempo=$temponot;

                    // Verifico se il tempo di percorrenza è minore o uguale del
                    // tempo di preavviso.
                    if($tempo>0 && $tempo<=$temponot+60  ){

                        // In questo caso la notifica può essere inviata, aggiungo
                        // il Registration_Id ai dati da inviare.
                        $notificationObj->addRegID($not["RegID"]);

                        // Elimino la notifica appena esaminata dal databse.
                        mysqli_query($conn,"DELETE FROM Notifiche".
                                    "WHERE REGISTRATION_ID='{$not["RegID"]}'".
                                    "AND IDOrario=$idOrario".
                                    "AND IDFermata={$not["IDF"]}");

                        // Aggiungo la notifica all'array delle notifiche da
inviare.
                        $notificationArray[]=$notificationObj;
                    }
                }
            }
        }
        // Restituisco l'array delle notifiche da inviare.
        return $notificationArray;
    }
```

Funzione getTime

Riceve come parametri le coordinate (latitudine e longitudine) di partenza e le coordinate di arrivo. Il suo compito è quello di effettuare una richiesta al web service delle Google Distance Matrix API per poter restituire il tempo di percorrenza tra le due posizioni

```
function getTime($lat1, $lon1, $lat2, $lon2){

            $from = "$lat1,$lon1";
            $to = "$lat2,$lon2";
            $from = urlencode($from);
            $to = urlencode($to);
            // Effettuo la richiesta HTTP GET e recupero i dati.
            $data =
file_get_contents("https://maps.googleapis.com/maps/api/distancematrix/json?origins=$fr
om&destinations=$to&language=en-EN&sensor=false&key=AIzaSyCn-
cGUDVWln3TdPJci9sDLoCGN11IlIb0");

            // Decodifico i dati.
            $jsondata = json_decode($data);
```

```
            $time=0;


            // Ricostruisco il tempo di percorrenza sommando i singoli tempi relativi
            // ai singoli tratti di strada da percorrere per arrivare a destinazione.
            foreach($jsondata->rows[0]->elements as $road) {
                $time += $road->duration->value;
            }

            // Restituisco il tempo.
            return $time;
}
```

Classe GCM

Per completezza riportiamo anche il codice relativo alla classe GCM, il cui scopo è quello di inoltrare ai server Google Cloud Messaging la notifica da inviare a un client.

```
class GCM {
        // $notData contiene i dati da inviare.
        public function send_notification($notData) {

            // Imposto la Server_key per accedere ai servizi GCM.
            define("GOOGLE_API_KEY", "AIzaSyB-6lTfanauRCvFyIiAaSpbu5HnMq44ksM");

            // Imposto l'url del server GCM.
            $url = 'https://android.googleapis.com/gcm/send';

            // Creo il messaggio da inviare; da notare il campo Registration_Id che
            // verrà utilizzato per individuare il client destinatario della
notifica.
            $fields = array(
                'registration_ids' => $notData->getReGIDs(),
                'data' => array("Ditta" =>$notData->Ditta,
                                "IDOrario" =>$notData->IDOrario,
                                "IDLinea" =>$notData->IDLinea,
                                "Ora" =>$notData->Ora,
                                "NomeLinea" =>$notData->NomeL,
                                "IDFermata" =>$notData->IDFermata,
                                "Fermata" =>$notData->Fermata,
                                "Tempo" =>$notData->Tempo,
                                "RegID" =>$notData->getReGIDs())
            );
            // Intestazioni della richiesta HTTP.
            $headers = array(
                'Authorization: key=' . GOOGLE_API_KEY,
                'Content-Type: application/json'
            );
            // Apro la connessione e imposto i parametri della richiesta tramite CURL
            // (tool a linea di comando per il trasferimento di dati utilizzando vari
            // protocolli).
            $ch = curl_init();

            // Imposto l'URL della richiesta.
            curl_setopt($ch, CURLOPT_URL, $url);

            // Imposto il metodo della richiesta (POST).
            curl_setopt($ch, CURLOPT_POST, true);
```

```
            // Imposto gli header della richiesta.
            curl_setopt($ch, CURLOPT_HTTPHEADER, $headers);
            // Indico al server che dovrà ritornare una stringa e non effettuare
            // l'output direttamente.
            curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
            // Disabilito SSL (altrimenti sarebbe una richiesta HTTPS).
            curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);

            // Inserisco idati da inviare nel corpo della richiesta.
            curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($fields));

            // Eseguo la richiesta.
            $result = curl_exec($ch);
            if ($result === FALSE) {
                die('Curl failed: ' . curl_error($ch));
            }
            // Chiudo la connessione.
            curl_close($ch);
            echo $result; }
 }
```

Ricezione della notifica

Infine, per completare la carrellata sulla gestione delle notifiche, manca solo più la descrizione di come una notifica viene intercettata dal client e di come, facendo click su di essa, viene ripresa l'esecuzione dell'applicazione.
Come già detto in precedenza, per poter intercettare una notifica proveniente da un server, occorre inserire nella nostra applicazione un BroadcastReceiver che resti in ascolto di messaggi di sistema. Una volta intercettato il messaggio occorre stabilirne la natura e, se tale messaggio è una notifica push, verranno recuperati i dati contenuti nel messaggio in modo da generare l'avviso di sistema e poter ricostituire il contesto dell'applicazione una volta che l'utente effettua un click sulla notifica.

Ricapitolando le operazioni effettuate sono le seguenti:

1. Il BroadcastReceiver intercetta il messaggio;
2. Viene richiamata l'API GCM, attraverso la quale si più identificare la natura del messaggio;
3. Se il messaggio viene identificato come notifica push, vengono recuperati i dati del messaggio;
4. Attraverso i dati recuperati si genera l'avviso di sistema, contenente le informazioni da visualizzare all'utente. L'avviso verrà anche collegato ad un "*PendingIntent*", un componente di Android attraverso il quale si può avviare un'activity appartenente ad un applicazione, da un contesto differente da quello dell'applicazione stessa (la notifica viene visualizzata all'esterno dell'applicazione);
5. Quando l'utente fa click sulla notifica viene richiamato il PendigIntent, il quale utilizzarà i dati del messaggio originale per ricreare il contesto dell'applicazione per poter visualizzare l'activity del dettaglio della linea.

A differenza di quanto fatto in precedenza, non riportiamo alcun codice relativo alla ricezione delle notifiche in quanto esso risulta lungo, di difficile comprensione e di scarso interesse espositivo dato che non viene effettuata nessuna operazione differente da quanto descritto sopra.

Sviluppi Futuri

Per finire, concludiamo l'esposizione del progetto con una carrellata di possibili sviluppi e nuove funzionalità che potrebbero essere implementate per migliorare il servizio offerto all'utente, ma che per motivi di tempo e risorse, non sono state realizzate.

Innanzitutto sarebbe utile poter offrire alle aziende che si occupano di trasporto pubblico la possibilità di poter aggiungere o modificare sul database centrale le informazioni riguardanti le linee di cui loro stesse si occupano. In questo modo sarebbe possibile avere delle informazioni sul sistema di trasporto pubblico attendibili e aggiornate in quanto sarebbero le ditte stesse a gestire i dati. Per realizzare ciò sarebbe per cui necessario realizzare un'interfaccia web, accessibile solo dagli utenti dotati di un "Account aziendale", attraverso alla quale le aziende possano accedere ai dati e gestirli.

Un altro miglioramento interessante potrebbe essere quello di gestire degli account utente per l'applicazione in modo che le preferenze relative ad un utente (linee preferite, notifiche, …) possano essere sincronizzate e accessibili tra dispositivi diversi.

Ovviamente ad un aumento delle informazioni raccolte deve anche corrispondere un aumento della sicurezza nella gestione dei dati. Andrebbero per cui implementate sul nostro web service delle politiche di sicurezza in modo da impedire furti di dati e accessi non autorizzati.
I miglioramenti possibili non finiscono qui, si potrebbe infatti migliorare il servizio sotto molteplici aspetti. Ad esempio si potrebbero gestire maggiori informazioni per le linee (posti in piedi, posti seduti, costo del biglietto…), visualizzare un dettaglio delle ditte, rendere più precisa la gestione delle notifiche, rendere più accurato il calcolo del ritardo, gestire modalità di trasporto più complesse come i cambi e le coincidenze, utilizzare i dati raccolti per generare statistiche sulla qualità del servizio (ritardi, numero di passeggeri …).

Infine, una volta che il servizio abbia preso piede e venga effettivamente utilizzato, sarebbe interessante estendere il servizio, adattandolo per altri tipi di trasporto quali treni, tram, taxi e qualunque altra situazione in cui si presi un servizio di localizzazione.