



**UNIVERSITÀ DEGLI STUDI DI PARMA**

---

Dipartimento di Ingegneria dell'Informazione  
Corso di Laurea in Ingegneria Informatica elettronica e delle  
telecomunicazioni

# Sviluppo di un App per la mobilità sostenibile tramite Gamification

Development of an App for sustainable mobility through  
Gamification

Relatore:

Prof. Andrea Prati

Correlatore:

Ing. Aldo Musci

Tesi di Laurea di:

Federico Mastrini

---

ANNO ACCADEMICO 2017-2018

Ai miei genitori e fratelli.

*“Vivi come se dovessi morire domani.  
Impara come se dovessi vivere per sempre.”*  
*Gandhi*

# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Background</b>	<b>4</b>
1.1 Il Mobility Manager ed il Piano Spostamento Casa-Lavoro . . .	4
1.2 Il Car Pooling . . . . .	6
1.3 La Gamification . . . . .	7
<b>2 Caratteristiche del sistema</b>	<b>10</b>
2.1 Registrazione e configurazione degli utenti . . . . .	11
2.2 Archiviazione dei tragitti e calcolo delle routine lavorative . .	12
2.3 Matching . . . . .	14
2.4 Gamification: livelli utente e statistiche . . . . .	18
<b>3 Applicazione Android</b>	<b>21</b>
3.1 Architettura dell'App . . . . .	22
3.1.1 Domain Layer . . . . .	22
3.1.2 Data Layer . . . . .	23
3.1.3 Presentation Layer . . . . .	23
3.2 Activity recognition e geolocalizzazione utente . . . . .	25
3.3 Networking e sincronizzazione . . . . .	31
3.4 Design, interfaccia utente (UI) ed esperienza d'uso (UX) . . .	33
Home . . . . .	34
Classifiche . . . . .	34
Match . . . . .	35

INDICE	ii
<hr/>	
Profilo . . . . .	36
3.5 Condivisione dell'auto . . . . .	37
<b>4 Conclusioni e sviluppi futuri</b>	<b>42</b>
<b>Riferimenti</b>	<b>44</b>

# Introduzione

Ogni giorno milioni di italiani si recano sul proprio posto di lavoro utilizzando la propria auto andando così a contribuire alla produzione di traffico e quindi all'inquinamento (anche sonoro). Ciò porta di conseguenza al crearsi dei così detti **costi della mobilità**, costi che possono essere suddivisi in individuali, che ricadono sull'utilizzatore del veicolo, ed ambientali e sociali, che ricadono sulla collettività e che non sono sostenuti da chi li ha generati. Alcuni esempi di questi ultimi possono essere:

- L'inquinamento atmosferico, che causa danni alla salute della popolazione, agli edifici e ai monumenti, ai boschi e all'agricoltura.
- Le perdite di tempo, dovute alla congestione del traffico, che hanno anche un impatto sui costi in termini di ore-lavoro perse.
- I danni sanitari, psichici e relazionali dovuti ad eventuali incidenti stradali.
- Il rumore.

Seppur negli ultimi anni lo sviluppo tecnologico dell'auto abbia contribuito a ridurre consumi ed inquinamento e il potenziamento dei trasporti pubblici abbia ridotto la congestione del traffico, il problema della **mobilità non sostenibile** è ancora attuale e significativo, e la necessità di porvi rimedio è abbastanza evidente ed urgente.

Con il Decreto del 27/03/98, il Ministero dell'Ambiente ha stabilito che le aziende con almeno 300 dipendenti siano tenute ad individuare un Responsabile della mobilità aziendale (Mobility manager) e ad adottare il **Piano**

**spostamento casa lavoro (PSCL)** dei propri dipendenti.

Con queste premesse, l'obiettivo di questo lavoro di tesi è quello di realizzare un'applicazione per smartphone in grado di ottimizzare gli spostamenti casa-lavoro dei dipendenti delle aziende, così da (i) poter far fronte alle esigenze nate dal decreto del 27/03/98 del Ministero dell'Ambiente, in un'ottica di sensibilizzazione sociale; (ii) spingere la collettività verso un mondo ecosostenibile ed, infine, (iii) venire in aiuto delle amministrazioni locali e/o gruppi di aziende che grazie all'app potranno individuare nuovi piani eco-sostenibili per lo spostamento dei dipendenti.

L'applicazione deve quindi essere in grado di:

- Raccogliere i dati sullo spostamento casa-lavoro dei dipendenti, così da poter tracciare un profilo delle abitudini lavorative dell'utente.
- Effettuare un match tra utenti con abitudini lavorative simili, in modo tale da poter suggerire altri dipendenti con cui poter condividere la propria auto oppure a cui poter chiedere un passaggio.
- Offrire un sistema basato sulla *Gamification* così da poter incentivare i dipendenti ad utilizzare l'applicazione.

Inoltre, il sistema deve offrire semplicità di utilizzo, affidabilità e un consumo di risorse dello smartphone minimo, offrendo un'applicazione leggera, *user friendly* e performante.

Lo scopo di questa tesi è quello di descrivere lo sviluppo dell'applicazione e l'architettura del sistema su cui si basa. Il tutto verrà diviso in differenti capitoli.

Lo scopo di questa tesi è quello di descrivere lo sviluppo dell'applicazione e l'architettura del sistema su cui si basa. Nel primo capitolo verranno introdotti alcuni concetti fondamentali per comprendere le tematiche trattate e si discuterà delle soluzioni già esistenti, mettendone in luce i limiti. Nel

---

secondo verrà descritto il sistema nel suo complesso, analizzando le caratteristiche principali e l'architettura adottata. Nel terzo capitolo si entrerà nel dettaglio dello sviluppo dell'applicazione Android, parte centrale del lavoro. Infine, nel quarto, verranno mostrati i risultati ottenuti ed alcuni esempi di *use case* dell'applicazione sviluppata.

# Capitolo 1

## Background

### 1.1 Il Mobility Manager ed il Piano Spostamento Casa-Lavoro

Come già accennato precedentemente nell'introduzione, la figura del *Mobility Manager* è stata introdotta con il Decreto Interministeriale “Mobilità Sostenibile nelle Aree Urbane” del 27/03/1998, e si applica ad ogni organizzazione (sia essa una azienda o un ente pubblico) con più di 300 dipendenti per “unità locale” o, complessivamente, con oltre 800 dipendenti.

Il decreto prevede che le organizzazioni interessate debbano individuare un responsabile della mobilità del personale, definito, per l'appunto, *Mobility Manager*. Anche se, come detto, tale figura va nominata non solo nelle aziende, ma anche negli enti pubblici, è usuale individuarla in tutti i casi con l'espressione “Mobility Manager aziendale”.

Il Mobility Manager aziendale ha il compito di individuare le strategie e gli interventi dell'azienda in materia di mobilità attraverso la redazione e il monitoraggio di un documento programmatico, il **Piano Spostamento Casa Lavoro (PSCL)**, mirato a conseguire una mobilità sempre più sostenibile favorendo l'utilizzo di modi alternativi all'uso individuale dell'auto privata.



Il Decreto del Ministero dell'Ambiente del 20/12/2000 ha poi definito la funzione del Mobility Manager di area, figura di supporto e di coordinamento dei Mobility Manager aziendali, istituita presso l'Ufficio Tecnico del Traffico dei Comuni più grandi.

Il Mobility Manager di area elabora le strategie orientate alla gestione della mobilità casa-lavoro nel suo complesso. La sua è un'importante funzione di coordinamento e di intermediario tra tutte le differenti parti coinvolte. Si concentra sullo studio dei comportamenti degli utenti e sulla domanda di trasporto a livello aggregato, in modo da individuare e dimensionare le possibili azioni applicabili in ciascuna impresa.

Gli obiettivi del Mobility Manager di Area possono essere raggiunti attraverso la realizzazione e l'adozione del PSCL dei dipendenti per l'azienda di cui fa parte o l'assistenza alle aziende tenute a redigere il PSCL che sono ricomprese nell'area di sua competenza (Comune). Il Piano Spostamenti Casa Lavoro (PSCL) analizza le abitudini di mobilità dei dipendenti al fine di individuare i problemi, le cause che li generano e le possibili soluzioni in modo da aumentare il benessere degli stessi e allo stesso tempo contribuire alla sostenibilità ambientale.

Infatti, gli spostamenti sistematici rappresentano una quota assai rilevante della mobilità in campo urbano. Pertanto ogni azienda è tenuta a fare il possibile per minimizzare l'impatto sull'ambiente generato dai suoi dipendenti, contribuendo attivamente al comune obiettivo di ridurre l'inquinamento atmosferico e le emissioni di gas serra.

In questi anni si è molto parlato delle iniziative che possono essere adottate dai Mobility Manager per favorire un minor uso del mezzo privato negli spostamenti casa-lavoro. Purtroppo, molte di esse sono poco adottate, non solo a causa dei costi, ma anche a causa della scarsa sensibilità e responsabilità dei dirigenti aziendali su questo tema. L'obbligo di legge viene adempiuto con la nomina del Mobility Manager, che deve essere necessariamente una fi-

gura interna all'azienda, e con la redazione del PSCL, per cui invece ci si può far supportare da consulenti esperti della materia. Il resto (collaborazione con in Mobility Manager di Area, monitoraggio periodico delle abitudini di mobilità, revisione delle misure adottate, ecc.) è lasciato alla buona volontà della dirigenza aziendale, ma non è materia sottoposta a controllo o sanzioni. Nella pratica, l'efficacia di un Piano Spostamento Casa-Lavoro in termini di shift modale (abbandono del mezzo privato a favore del mezzo pubblico o del trasporto condiviso), rischia di essere molto scarsa, ed il successo dell'iniziativa è affidato alla buona volontà degli stessi dipendenti.

## 1.2 Il Car Pooling

Il *car pooling* è uno degli ambiti di intervento della mobilità sostenibile e consiste nella condivisione di automobili private tra un gruppo di persone, con il fine principale di ridurre i costi del trasporto.

Nel car pooling, in genere, uno o più soggetti coinvolti mettono a disposizione il proprio autoveicolo, mentre i passeggeri contribuiscono coprendo parte della spesa sostenuta.

L'utilizzo di questa pratica porta ad un miglioramento della congestione del traffico e ad una riduzione dell'inquinamento, dovuto ad un ridotto numero di veicoli in circolazione. Porta, inoltre, ad un risparmio economico in termini di costo pro-capite di carburante, olio, pneumatici, pedaggi, costi di parcheggio, ecc. . . e ad un miglioramento dei rapporti sociali tra le persone.

La condivisione dell'automobile riduce però il grado di flessibilità proprio dell'uso individuale di un veicolo, rendendosi necessario un accordo preventivo da parte dei passeggeri sui tempi di viaggio e sui percorsi previsti.

La pratica del car-pooling ha visto negli ultimi anni un incremento notevole, grazie alla diffusione di numerose piattaforme web e applicazioni per smartphone, che consentono a chi cerca un passaggio ed a chi ne offre uno

di incontrarsi e definire al meglio i dettagli organizzativi del viaggio, che sia esso di natura occasionale o continuativa.

Queste piattaforme consentono, in particolare, di superare uno dei principali ostacoli della diffusione della pratica del car-pooling, dato dalla naturale diffidenza nel condividere un viaggio in auto con degli sconosciuti.

Non sono però provviste di un sistema che consente a chi cerca un passaggio o a chi lo offre, di trovare persone compatibili con i propri spostamenti in modo tale da avere un contatto rapido e semplificato con altri passeggeri o persone disposte a condividere l'auto, lasciando quindi il compito all'utilizzatore di trovare queste persone, non sempre con esiti positivi. Mancano, inoltre, effetti incentivanti, come quelli descritti nella prossima sezione.

## 1.3 La Gamification

Il termine *gamification* è stato introdotto per la prima volta in pubblico nel febbraio 2010 da Jesse Schell, un famoso game-designer americano.

Il termine, com'è facile intuire, deriva dalla parola "Game", cioè gioco, anche associato al semplice divertimento senza scopi particolari. La Gamification tuttavia non è semplicemente questo, non solo: traendo vantaggio dall'interattività concessa dai mezzi moderni ed ovviamente dai principi alla base del concetto stesso di divertimento, la Gamification rappresenta uno strumento estremamente efficace in grado di veicolare messaggi di vario tipo, a seconda delle esigenze, e di indurre a comportamenti attivi da parte dell'utenza, permettendo di raggiungere specifici obiettivi, personali o d'impresa. Al centro di questo approccio va sempre collocato l'utente ed il suo coinvolgimento attivo.

Possiamo definire la Gamification come un insieme di regole mutate dal mondo dei videogiochi, che hanno l'obiettivo di applicare meccaniche ludiche ad attività che non hanno direttamente a che fare con il gioco; in questo modo è possibile influenzare e modificare il comportamento delle persone,

favorendo la nascita ed il consolidamento di interesse attivo da parte degli utenti coinvolti verso il messaggio che si è scelto di comunicare, sia questo relativo all'incremento di performance personali o più in generale alle performance d'impresa.

Tali meccaniche stanno decisamente evolvendo negli ultimi anni, di pari passo con le recenti innovazioni portate dagli studi in materia di game design. Le meccaniche di base, però, sono composte da concetti semplici, utilizzati per assicurare un'esperienza irresistibile per l'utente/giocatore, in grado di aumentare l'interesse, spingendo alla partecipazione e all'impegno.

Le meccaniche di gioco basilari sono i punti, i livelli, le sfide, i beni virtuali e le classifiche.

Ogni meccanica è strettamente legata e soddisfa una determinata dinamica di gioco, influenzandone anche altre in maniera minore. Le dinamiche coinvolte nella gamification sono la ricompensa, lo stato, la conquista di un risultato, l'espressione di sé e la competizione.

Esploriamo in dettaglio ogni meccanica, associata alla dinamica corrispondente.

- *Punti/Crediti – Ricompensa*: il collezionismo di punti è una meccanica molto potente poiché è in grado di motivare le persone. Anche se non c'è un valore reale associato ai punti gli utenti continuano ad accumularli e possono anche venir suddivisi in diverse categorie, in modo da spingere verso interazioni di tipo differente o comportamenti particolari. I punti si possono poi scambiare per ottenere ricompense che forniscono all'utente la sensazione di investire in modo profittevole il proprio tempo ed energie, dando l'idea di guadagnare qualcosa.
- *Livelli – Stato*: i livelli rappresentano una segmentazione della base d'utenza e riflettono numerosi contesti reali: gli ambienti sociali, lavorativi e d'affari sono spesso basati su differenti classi ordinate in modo gerarchico.

I livelli forniscono un sistema per introdurre traguardi da raggiungere e che possono essere condivisi ed evidenziati nello stato dell'utente. La struttura sottostante può comunque basarsi su punti che ognuno guadagna per passare di livello, garantendo l'accesso a nuovi contenuti e possibilità inedite.

- *Sfide – Obiettivi*: le sfide sono le “missioni” che gli utenti possono intraprendere all'interno del gioco. Forniscono una ragione per continuare a partecipare e motivano gli utenti a raggiungere dei risultati sotto forma di trofei o obiettivi da sbloccare. La vera essenza degli obiettivi, comunque, risiede nella possibilità di mostrarli ad altri utenti, con una struttura che incoraggia il confronto e spinga alla competizione.
- *Beni virtuali o reali – Espressione di sé*: l'economia di gioco costruita sull'opportunità di ottenere dei punti non può durare a lungo senza qualcosa che l'utenza possa acquistare, guadagnare e, in determinati casi, consumare. La presenza di beni virtuali può far aumentare l'interesse dei giocatori che possono comprare o aggiudicarsi una vasta scelta di oggetti, dal vestiario alle armi o abilità, utilizzati per creare un'identità propria nell'ambiente sociale. In questo modo, ad esempio, ognuno può esprimere sé stesso personalizzando il proprio avatar e mostrandolo ad amici e colleghi.

Nel caso della mobilità sostenibile, l'utilizzo della Gamification può essere uno strumento per incentivare i dipendenti ad applicare il Piano Spostamento Casa-Lavoro attuato dall'azienda, in cambio di riconoscimenti all'interno dell'azienda oppure veri e propri premi.

Accumulare punti e migliorare il proprio profilo con statistiche e grafici va a creare un senso di competizione nel quale, alla fine, vanno a guadagnarci sia l'azienda che il dipendente stesso.

## Capitolo 2

# Caratteristiche del sistema

Il sistema sviluppato è costituito da un server cloud, responsabile della gestione dei dati, salvataggio e matching dei tragitti, e da un client Android, necessario per il tracciamento dei percorsi casa-lavoro e utilizzato come mezzo di interazione con il sistema da parte degli utenti.

Tenendo conto di un elevato numero di utenti che utilizzeranno l'app e le funzionalità che sfruttano algoritmi di matching e di archiviazione dei tragitti, per ospitare il server si è considerato necessario utilizzare un Datacenter che soddisfi i seguenti requisiti:

- Affidabilità
- Elevato calcolo computazionale
- Elevata disponibilità di archiviazione
- Scalabilità
- Sicurezza
- Attività di Audit
- GDPR compliance

Per soddisfare quanto sopra si è scelto di ospitare la piattaforma su Data-center Azure, il quale mette a disposizione funzionalità avanzate per gestire piattaforme finalizzate alla raccolta di Big Data e Data Analytics.

Il server è stato sviluppato in ASP.NET e C#. Questa scelta è stata dovuta ad una questione di comodità piuttosto che da esigenze particolari: l'ambiente di sviluppo messo a disposizione da Visual Studio, IDE ufficiale per lo sviluppo con .NET, in aggiunta al deploy automatizzato su piattaforma Azure, ha reso la realizzazione del lato server molto rapido e semplificato rispetto ad altri linguaggi disponibili.

Il web service fornisce delle API realizzate secondo i principi dell'architettura REST, che mettono a disposizione le risorse al client attraverso richieste HTTP.

Per accedere alle API è necessaria una chiave privata, pertanto non sono pubblicamente accessibili ma solamente da client autorizzati come, in questo caso, l'applicazione Android ufficiale ed altri applicativi realizzabili in futuro.

## 2.1 Registrazione e configurazione degli utenti

Per poter utilizzare l'applicazione ogni utente deve avere un account registrato e configurato. Registrazione e configurazione avvengono via web e non sono integrate direttamente nell'applicazione, che può essere utilizzata solo con un account attivo.

La registrazione di un account può avvenire in due modi:

- Via integrazione con il sistema di gestione delle risorse umane dell'azienda. L'azienda registra manualmente gli account tramite il caricamento via SFTP di un file con le anagrafiche dei dipendenti. Il sistema provvederà alla popolazione dei dati nel database e alla creazione degli

account. Gli utenti potranno quindi accedere all'applicazione tramite l'e-mail e la matricola come password.

- Via registrazione manuale da parte dei dipendenti. Tramite l'apposita pagina web si potrà inserire i propri dati anagrafici e creare quindi un nuovo account. L'utente potrà quindi accedere con e-mail e password scelti al momento della registrazione.

Dopo la creazione di un account è necessaria la sua configurazione.

In questa fase l'utente dovrà specificare l'indirizzo di partenza ed arrivo al lavoro, oltre che agli orari abituali. Sebbene il sistema utilizzi i dati rilevati sugli spostamenti casa-lavoro degli utenti, la configurazione iniziale permette di poter trovare fin da subito dei match con altri utenti, in attesa di dati più accurati tramite l'analisi dei percorsi rilevati. Una volta che l'utente avrà utilizzato abbastanza a lungo l'applicazione, così da poter permettere al sistema di creare un profilo degli spostamenti casa-lavoro, questi dati non saranno più presi in considerazione.

## **2.2 Archiviazione dei tragitti e calcolo delle routine lavorative**

Per poter calcolare i match tra utenti che condividono abitudini lavorative simili, viene utilizzato un algoritmo che confronta i tragitti casa-lavoro che vengono percorsi abitualmente. Il rilevamento dei tragitti avviene tramite il client Android che ad intervalli regolari invia al server i dati registrati. Per quanto riguarda il modo in cui le posizioni vengono registrate dal client se ne parlerà nel capitolo successivo, mentre ora ci si soffermerà su come questi dati sono gestiti dal server.

I dati inviati contengono un array in formato JSON composto da:

- Coordinate geografiche della posizione registrata.



- Data e ora del rilevamento.
- ID dell'utente.

```
[  
  {  
    "userId": 5,  
    "latitude": 44.24569120,  
    "longitude": 10.1524220,  
    "date": "2018-10-21T00:10:23"  
  },  
  {  
    "userId": 5,  
    "latitude": 44.1459120,  
    "longitude": 10.14576220,  
    "date": "2018-10-21T00:10:23"  
  },  
  ...  
]
```

Un esempio dei dati inviati dal client al server

Una volta ricevuti, i dati vengono poi archiviati in un database MS-SQL.

Al fine di ridurre i calcoli necessari dall'algoritmo di matching, da queste informazioni viene ricavata una “routine” di spostamento casa-lavoro per ogni utente. Questa routine contiene le informazioni necessarie per sapere gli orari in cui lavora e gli spostamenti che effettua ogni utente, da cui poi si potranno ricavare match con altri utenti.

Per calcolarla vengono prese le posizioni registrate dell'utente e vengono filtrate quelle relative alla partenza ed arrivo dalla propria abitazione e quelle relative alla partenza ed arrivo al lavoro. In questo modo è possibile rimuovere eventuali spostamenti straordinari che causerebbero rumore nei risultati. Da queste posizioni vengono poi calcolati, facendone una media, gli orari re-

lativi alla partenza da casa, arrivo al lavoro, partenza dal lavoro ed arrivo a casa. Gli orari relativi alle partenze vengono arrotondati per difetto, mentre quelli di arrivo vengono arrotondati per eccesso.

La routine di ogni dipendente è aggiornata settimanalmente e tiene conto delle posizioni delle ultime 4 settimane.

## 2.3 Matching

Una volta che le routine lavorative sono state calcolate, l'algoritmo di matching calcola i match tra i vari utenti presenti nel sistema.

Un match è una coppia di utenti che condividono approssimativamente le stesse abitudini lavorative, quindi un potenziale ospite (*guest*) ed un ospitante (*host*) per una condivisione dell'auto.

Inizialmente l'algoritmo ideato teneva conto dell'intero tragitto casa-lavoro, calcolando anche match parziali derivanti da percorsi condivisi anche parzialmente o che si intersecavano in uno o più punti. Il risultato era un lavoro eccessivo anche su pochi percorsi e la presenza di falsi positivi. Per poter ridurre quindi il carico di lavoro e la presenza di errori si è scelto di limitare il calcolo al solo punto di partenza ed arrivo, trascurando quindi il percorso che li collega.

Il risultato è stato comunque soddisfacente, avendo ridotto i tempi di esecuzione e le risorse utilizzate, senza ridurre l'efficacia dei match ottenuti.

L'algoritmo si basa su di un sistema a punteggio suddiviso in tre fattori: distanza del punto di partenza, distanza del punto di arrivo e differenza di orario. Per ognuno dei tre fattori viene calcolato un punteggio indipendente dagli altri, ma che insieme vanno a costituire il punteggio finale.

Se uno dei tre punteggi è nullo, quindi senza alcun riscontro, il punteggio finale è automaticamente anch'esso nullo, stando quindi a significare che non vi è match. Prendiamo ad esempio due persone che abitano nella stessa zo-

na, lavorano agli stessi orari ma in zone completamente diverse, oppure due persone che abitano nella stessa zona e lavorano nella stessa azienda ad orari completamente differenti. In questi due esempi non vi può essere un match, seppur due dei tre fattori hanno un punteggio pieno.

```
private static int GetDistanceScore(  
    double guestLat, double guestLng,  
    double hostLat, double hostLng,  
    double minDistance, double maxDistance)  
{  
    var distance = MapUtils.Haversine(guestLat, guestLng, hostLat,  
        hostLng);  
  
    var score = (int) ((distance - maxDistance) / (minDistance -  
        maxDistance)) * 100;  
  
    score = score > 100 ? 100 : score;  
    score = score < 0 ? 0 : score;  
  
    return score;  
}
```

Funzione per il calcolo del punteggio partendo dalla distanza

Per quanto riguarda il punteggio attribuito alla distanza di partenza ed arrivo, la funzione utilizzata per il calcolo è la medesima e la si può vedere nello snippet di codice qui sopra. Tiene conto delle posizioni di partenza ed arrivo degli utenti (guest ed host), una distanza massima oltre il quale il punteggio è automaticamente 0 (*maxDistance*) ed una distanza minima entro il quale il punteggio è automaticamente il massimo (*minDistance*). Entrambe le distanze sono configurabili esternamente alla funzione e possono essere regolate in base alle esigenze.

La distanza viene calcolata utilizzando la formula di Haversine, formula che consente il calcolo della distanza tra due coordinate terrestri. Date le coppie di coordinate geografiche  $\phi$  e  $\theta$  :

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_{lat} - \theta_{lat}}{2}\right) + \cos(\phi_{lat}) \cos(\theta_{lat}) \sin^2\left(\frac{\phi_{lon} - \theta_{lon}}{2}\right)}\right)$$

dove  $d$  è la distanza tra le due coordinate date e  $R$  il raggio terrestre.

```
public static double Haversine(double lat1, double lon1, double
    lat2, double lon2)
{
    var R = 6372.8; // In kilometers
    var dLat = toRadians(lat2 - lat1);
    var dLon = toRadians(lon2 - lon1);
    lat1 = toRadians(lat1);
    lat2 = toRadians(lat2);

    var a = Math.Sin(dLat / 2) * Math.Sin(dLat / 2) + Math.Sin(dLon
        / 2) * Math.Sin(dLon / 2) * Math.Cos(lat1) * Math.Cos(lat2);
    var c = 2 * Math.Asin(Math.Sqrt(a));
    return R * c;
}
```

Implementazione della formula di Haversine in C#

Una volta calcolata la distanza ne viene estrapolato un punteggio. Se quest'ultimo è negativo, significa che le due posizioni sono più distanti della massima distanza consentita ed il punteggio restituito è 0. Se la distanza è maggiore di 100 il punteggio restituito è 100.

```
private static int GetTimeScore(
    TimeSpan guestStartTime, TimeSpan guestEndTime,
    TimeSpan hostStartTime, TimeSpan hostEndTime,
    TimeSpan minTimeDelta, TimeSpan maxTimeDelta)
{
    var startTimeDelta =
        guestStartTime.Subtract(hostStartTime).Duration();
    var endTimeDelta =
        guestEndTime.Subtract(hostEndTime).Duration();

    var startScore =
        (startTimeDelta.Subtract(maxTimeDelta).Milliseconds /
        minTimeDelta.Subtract(maxTimeDelta).Milliseconds) * 100;
    var endScore =
        (endTimeDelta.Subtract(maxTimeDelta).Milliseconds /
        minTimeDelta.Subtract(maxTimeDelta).Milliseconds) * 100;

    startScore = startScore > 100 ? 100 : startScore;
    startScore = startScore < 0 ? 0 : startScore

    endScore = endScore > 100 ? 100 : endScore;
    endScore = endScore < 0 ? 0 : endScore;

    if (startScore == 0 || endScore == 0) return 0;
    return (startScore + endScore) / 2;
}
```

Funzione per il calcolo del punteggio in base alla differenza di orario

Per quanto riguarda il calcolo del punteggio dell'orario la funzione è simile alla precedente per il calcolo del punteggio sulla distanza, ed è possibile vederla nello snippet di codice precedente. La differenza è che al posto di utilizzare le coordinate della posizione, il calcolo è effettuato sull'ora di arrivo e partenza

da lavoro del guest e del host, in forma di *TimeSpan*, una struttura del .NET framework che rappresenta un intervallo di tempo.

Anche in questo caso vengono passati due valori per determinare il tempo minimo entro il quale il punteggio è automaticamente il massimo (*minTimeDelta*) e il tempo massimo oltre il quale il punteggio è automaticamente 0 (*maxTimeDelta*). Questi valori sono ovviamente configurabili all'esterno della funzione per effettuare eventuali regolazioni.

I match considerati quindi validi saranno quelli con un punteggio uguale o superiore ad un punteggio minimo indicato al momento del calcolo. Ad esempio, si può impostare di non considerare match con un punteggio inferiore a 50, che indicherebbe un match “scarso”.

## 2.4 Gamification: livelli utente e statistiche

Nei capitoli precedenti si è parlato di Gamification e di come questa tecnica possa portare a risultati positivi all'interno di iniziative dove la partecipazione degli utenti è di fondamentale importanza.

In questo caso si è scelto di introdurre il concetto come all'interno di un gioco di ruolo tramite un exp system: l'utente compie azioni ed in cambio ottiene punti esperienza, denominati punti Karma.

I punti Karma vengono attribuiti ogni qual volta un utente completa una condivisione della propria auto o utilizza l'auto condivisa di un altro utente. I punti guadagnati dipendono da quanti chilometri sono stati percorsi e il ruolo dell'utente durante la condivisione. Il ragionamento di fondo è che un utente deve guadagnare tanti punti quanti i chilometri che non sono stati percorsi grazie alla condivisione effettuata.

$$punti_{guest} = Km * C$$

Se invece l'utente è host della condivisione, quindi ha messo a disposizione la propria auto, i punti guadagnati saranno dati dalla somma dei chilometri percorsi da tutti gli ospiti della condivisione moltiplicati per una costante  $C$ .

$$punti_{host} = \sum_{i=0}^{guest} Km_i * C$$

La costante  $C$  è attualmente impostata a 10, in questo modo gli utenti guadagneranno 10 punti ogni chilometro condiviso.

In base al punteggio ad ogni utente viene attribuito un livello che andrà ad indicare il prestigio e la dedizione dell'utente che, a discrezione dell'azienda a cui appartiene, gli garantirà premi settimanali o mensili, in base alle proprie prestazioni nel periodo indicato.

Il livello utente viene calcolato a partire dall'ammontare di punti Karma guadagnati ed utilizza una proporzione quadratica inversa, modello matematico che permette di richiedere un numero sempre maggiore di punti per progredire tra i livelli.

$$livello = 1 + \sqrt{\frac{p + T}{T}}$$

dove  $p$  è l'ammontare di punti dell'utente, mentre  $T$  una costante che permette di regolare la velocità con cui gli utenti possono progredire tra i livelli.

L'aggiunta di 1 garantisce un livello iniziale per ogni utente di 1.

Oltre ad un exp system è stato introdotto un sistema di statistiche in modo tale che gli utenti, oltre a migliorare il proprio profilo visibile da altri utenti, possano entrare in una sorta di competizione tra loro.

Le statistiche raccolgono dati sulla quantità di chilometri condivisi, condivisioni completate, media di condivisioni mensili e record personali, come la quantità massima di condivisioni effettuati nell'arco di un mese oppure la condivisione più lunga mai registrata.

In aggiunta alle statistiche vi è un sistema di classifiche. Le classifiche danno un resoconto dei migliori utenti della settimana, del mese oppure di sempre, sia su base globale, andando ad includere tutti gli utenti indipendentemente dall'azienda di appartenenza, oppure ristrette all'azienda di appartenenza. In questo modo i migliori utenti possono essere riconosciuti e premiati dall'azienda.

Questo sistema fa sì che gli utenti vengano spinti a condividere l'auto con l'idea di guadagnare punti, migliorare le proprie statistiche e ricevere premi, comportando direttamente un calo del traffico, dell'inquinamento e un maggior rispetto del PSCL.



## Capitolo 3

# Applicazione Android

Dopo aver parlato del lato server nei capitoli precedenti, in questa parte della tesi verrà descritto il client del sistema sviluppato.

Il requisito fondamentale per il client era quello di aver la possibilità di installarlo su un dispositivo che fosse portatile e sempre in possesso del dipendente, in modo tale da poter tenere traccia dei suoi spostamenti casa-lavoro in modo preciso ed efficace.

Oltre a ciò, il client doveva risultare: (i) semplice da installare da parte dell'azienda e/o del dipendente, (ii) facile nell'utilizzo grazie ad una interfaccia utente semplice ed user friendly ed, infine, (iii) non avido di risorse, in modo tale da non compromettere il dispositivo su cui veniva installato, accorciandone la durata della batteria o rendendolo inutilizzabile.

Per le motivazioni sopracitate, e per ovvie ragioni, si è scelto di realizzare un'applicazione per smartphone.

L'applicazione realizzata è per dispositivi **Android** e richiede come requisito minimo la versione 5.0 con supporto alle API 21.

Il linguaggio utilizzato per lo sviluppo è **Kotlin**, linguaggio di programmazione moderno ed open source, progettato per essere pienamente operativo con l'ambiente Java e tutto il suo enorme ecosistema, superando però le limitazioni e le criticità del linguaggio Java stesso.

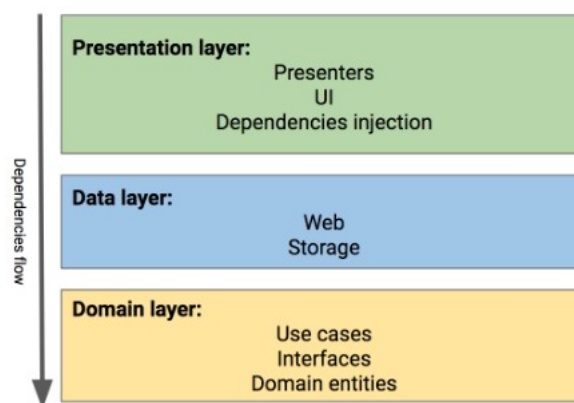
## 3.1 Architettura dell'App

Per la progettazione e lo sviluppo dell'applicazione sono stati seguiti i principi della *Clean Architecture*, architettura che consente di realizzare un'applicazione con codice altamente riutilizzabile e semplice da modificare in caso di cambi di specifiche e requisiti nel corso dello sviluppo

I principi fondamentali della Clean Architecture sono:

- Il codice dell'applicazione è suddiviso in strati (*layer*): ogni strato ha un solo e preciso scopo.
- Gli strati seguono una rigida legge di dipendenza: ogni strato vede e comunica soltanto con gli strati sottostanti.
- Più ci si avvicina al fondo, più il codice diventa generico e meno specifico: lo strato sul fondo detta regole e metodi, lo strato superiore si occupa della loro implementazione.

Il risultato di tale approccio è stato quindi un'applicazione suddivisa in tre moduli o strati: Domain, Data e Presentation.



### 3.1.1 Domain Layer

Questo layer è totalmente indipendente da Android ed è costituito da codice molto generico. Non è a conoscenza della presenza degli altri due moduli

presenti nel progetto e non sa della presenza dell'SDK Android. Non si preoccupa di come i dati verranno rappresentati, ma definisce cosa è e cosa può fare l'applicazione.

E' costituito dalle *Entity*, classi che descrivono i dati trattati e che fungono da contenitori di informazioni, dagli *UseCase* o iteratori, che descrivono cosa fa l'applicazione, ed infine dalle *Interface*, “contratti” con il quale il Domain layer indica cosa fare ai layer superiori.

Queste astrazioni consentono di garantire un corretto funzionamento del sistema indipendentemente dalle modifiche che avvengono nella loro implementazione.

### 3.1.2 Data Layer

Appena sopra il Domain layer vi è il Data layer. La sua funzione principale è quella di fornire all'applicazione tutti i dati di cui ha bisogno per funzionare. Non vi è più codice generico ma vi sono delle implementazioni concrete di data providers. Conosce i dati e sa in che modo reperirli grazie al Domain layer, ma non conosce come saranno rappresentati dal Presentation Layer.

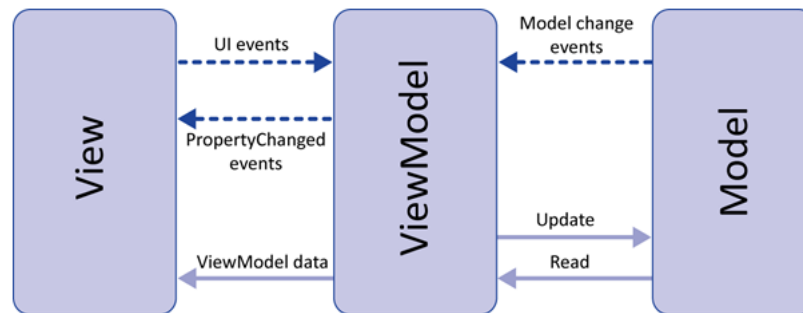
Il Data layer implementa tutte le funzioni necessarie per comunicare con le API REST del server e la gestione dello storage locale. Per quanto riguarda la comunicazione con le API REST è stata utilizzata la libreria Retrofit, la quale permette di creare client REST partendo da delle interfacce le quali definiscono i metodi HTTP necessari.

### 3.1.3 Presentation Layer

Infine, al di sopra di tutti i layer, vi è il Presentation layer.

Qui vi troviamo le View (*Activity* e *Fragment*), tramite le quali l'utente potrà interagire con l'applicazione.

Esse sono implementate utilizzando il pattern MVVM (*Model-View-ViewModel*) il quale permettere di suddividere in differenti parti con responsabilità diverse le View, garantendo un ulteriore isolamento del codice.



La View, come detto in precedenza, si occupa di rappresentare i dati nella UI e di ricevere gli input dell'utente. Il Model si occupa di tenere le informazioni di cui ha bisogno la view e funziona come contenitore di informazioni. Il ViewModel è il ponte di comunicazione tra le due parti precedentemente citate. Si occupa di leggere e scrivere nel model, elaborare i dati e servirli alla View.

Il Presentation layer ha un ulteriore compito fondamentale che è quello della *Dependency Injection* che, per progetti di questa portata, è di importanza critica.

Solitamente, ad ogni istanza di una classe si tende a creare all'interno di essa nuove istanze delle classi di cui ha bisogno. In questo modo però una modifica in una di quest'ultime può portare a cambiamenti inaspettati di behavior nelle classi che le utilizzano. Inoltre, in caso di testing (tramite Unit Testing o test Android), la modifica manuale di ogni classe con classi di mockup può portare a perdere molto tempo. La Dependency Injection non è altro che il passaggio di istanze delle dipendenze direttamente nel costruttore della classe. Per semplificare il processo ci vengono in aiuto le Factory Class, che ci permettono di "costruire" le classi di cui abbiamo bisogno passando le dipendenze tramite Interface e UseCase.

Una volta definite le Factory, si implementano le interfacce necessarie tramite dei Provider. Nel progetto è stato utilizzato *Dagger 2* [1], nota libreria sviluppata da Google per la Dependency Injection, che tramite l'utilizzo di annotazioni nel codice consente di definire dipendenze e providers, generando automaticamente tutto il codice necessario.

Nel Presentation layer, oltre alle View, vi sono poi i service necessari per il funzionamento della geolocalizzazione dell'utente, del quale si parlerà nel capitolo successivo.

## 3.2 Activity recognition e geolocalizzazione utente

La funzione principale dell'applicazione è quella di registrare gli spostamenti casa-lavoro degli utenti e sincronizzare le posizioni registrate con il server.

Il rilevamento degli spostamenti avviene tramite l'esecuzione di un servizio in background che resta in esecuzione fino a quando l'utente non effettua il logout dall'applicazione.

Per limitare il dispendio energetico sono state utilizzate alcune strategie per rendere intelligente la registrazione delle posizioni e limitare l'utilizzo della batteria. Una strategia rivelatasi vincente è stata quella di utilizzare le API Android per l'*Activity Recognition* [2].

L'Activity Recognition consente di avere una stima dell'attività dell'utente in corso (camminata, corsa, bicicletta, fermo, in piedi, in auto, ecc...) attraverso una misurazione dello stato del dispositivo dai vari sensori di cui è munito, come sensore di prossimità e giroscopio.

Nell'app sviluppata questa API viene utilizzata per sapere quando l'utente inizia uno spostamento in auto e, non appena viene rilevata, viene registrata la sua posizione. Viceversa, se l'utente è in auto e la successiva misurazione dell'attività rileva che l'utente sta camminando, possiamo dire che la guida è terminata. Si è volutamente scelto di contare anche eventuali soste durante

il tragitto (ad esempio, per un rifornimento di carburante) poiché non rappresenta un problema per la stima delle routine lavorative.

Il sistema messo a punto viene implementato da un **Service** denominato **LocationService** che viene eseguito non appena l'utente effettua l'accesso oppure, se l'utente ha già effettuato l'accesso in precedenza, all'avvio del dispositivo.

Non appena avviato, il client effettua alcune operazioni:

- Controlla lo stato di autenticazione dell'utente: nello specifico, controlla che l'access token generato dal server sia ancora valido e non sia necessaria una nuova autenticazione.
- Connette i client Google necessari per il funzionamento delle API per la Geolocalizzazione e il rilevamento dell'attività.
- Crea una notifica persistente che informa dello stato di esecuzione del servizio, necessario per far sì che il sistema operativo Android non termini il servizio.
- Avvia il servizio di rilevamento dell'attività. Il rilevamento è configurato per effettuare un controllo sulle attività ogni 10 secondi.
- Termina la configurazione e passa nello stato di *foreground*, in attesa di attività da parte dell'utente.

A questo punto, non appena l'Activity Recognition rileva nuove attività, viene avviato un **IntentService** denominato **ActivityRecognizedService** che ha il compito di analizzare i dati rilevati.

Il rilevamento è composto da un array delle differenti attività possibili con un valore di affidabilità associato: più quest'ultimo valore è elevato, più è probabile che l'attività rilevata sia quella in corso. L'array viene filtrato e vengono prese soltanto le attività che superano un soglia di 75 (valore deciso a seguito di test che hanno evidenziato un minor numero di errori e falsi

positivi).

```
private fun handleDetectedActivities(probableActivities:
    List<DetectedActivity>) {
    probableActivities
        .asSequence()
        .filter { it.confidence >= ACTIVITY_TRESHOLD }
        .forEach {
            when (it.type) {
                DetectedActivity.IN_VEHICLE -> {
                    startDrivingActivity()
                }
                DetectedActivity.ON_FOOT,
                DetectedActivity.ON_BICYCLE,
                DetectedActivity.RUNNING -> {
                    stopDrivingActivity()
                }
                DetectedActivity.STILL -> {
                    increaseStillStatus()
                }
            }
        }
    }
}
```

Funzione per l'analisi delle attività rilevate

La funzione nello snippet di codice qui sopra filtra le attività che superano il livello di soglia impostato ed analizza le rimanenti attività comunicando al `LocationService` l'operazione da effettuare. A questo punto, in base all'attività rilevata, si possono verificare differenti scenari.

```
private fun startDrivingActivity() {  
    if (isDriving) { return }  
  
    startService(intentFor<LocationService>(EXTRA_ACTIVITY to  
        DrivingActivity.STARTED_DRIVING))  
  
    isDriving = true  
    stillStatusCounter = 0  
}
```

Funzione per l'avvio dell'attività di guida

Viene rilevata l'attività di guida: se il flag per lo stato di guida è impostato su `False` (e quindi l'utente non era già in uno stato di guida) viene impostato su `True` e viene comunicato al `LocationService` di registrare la posizione.

```
private fun increaseStillStatus() {  
    if (!isDriving) { return }  
  
    stillStatusCounter++  
    if (stillStatusCounter >= MAX_STILL_STATUS_COUNT) {  
        stopDrivingActivity()  
    }  
}
```

Funzione per la gestione dei rilevamenti dell'attività STILL

Viene rilevato che l'utente è fermo: non viene terminata la guida, poiché se il dispositivo è fermo non possiamo sapere se è perché l'utente ha smesso di guidare o perché è fermo in auto (in coda nel traffico, ad esempio). Viene quindi incrementato il contatore `stillStatusCounter` che, una volta raggiunto un valore massimo, terminerà automaticamente lo stato di guida.



```
private fun stopDrivingActivity() {  
    if (!isDriving) { return }  
  
    startService(intentFor<LocationService>(EXTRA_ACTIVITY to  
        DrivingActivity.STOPPED_DRIVING))  
  
    isDriving = false  
    stillStatusCounter = 0  
}
```

Funzione per l'interruzione dell'attività di guida

Infine, se viene rilevata una qualsiasi altra attività, possiamo dire che l'utente non stia più guidando e quindi viene impostato su False il flag dello stato di guida, azzerato il contatore `stillStatusCounter` e comunicato al `LocationService` la fine dell'attività di guida.

Il service `ActivityRecognizedService` comunica con il `LocationService` attraverso degli Intent con il quale passa le informazioni sull'attività rilevata. Quest'ultimo riceve tali informazioni nel metodo `onStartCommand` e, se le informazioni sono valide e contengono un risultato, registra la posizione dell'utente.

La posizione viene poi salvata in un database SQLite in attesa della sincronizzazione con il server.

Il `LocationService` ha anche il compito di registrare l'ultima posizione dell'utente, indipendentemente dalla sua attività corrente. Questo avviene tramite la registrazione di un *listener* il quale, non appena rileva che la nuova posizione dell'utente soddisfa determinati requisiti, comunica la posizione corrente con il server.

```
if (ActivityCompat.checkSelfPermission(this,
    Manifest.permission.ACCESS_FINE_LOCATION) ==
    PackageManager.PERMISSION_GRANTED) {

    val locationRequest = LocationRequest.create()
    locationRequest.priority =
        LocationRequest.PRIORITY_LOW_POWER
    // Voglio la posizione non prima di uno spostamento di
        almeno 1 km dall'ultima posizione nota
    locationRequest.smallestDisplacement = 1000f
    // Se altre app hanno richiesto l'aggiornamento della
        posizine la ottengo anche io ma non prima che siano
        passati almeno 5 minuti
    locationRequest.fastestInterval = 5 * 60 * 1000
    // Se non disponibile, richiedo io un aggiornamento di
        posizione ogni 30 minuti
    locationRequest.interval = 30 * 60 * 1000

    mFusedLocationProviderClient?
        .requestLocationUpdates(locationRequest,
            lastLocationCallback, Looper.myLooper())

}
}
```

Registrazione del listener per l'aggiornamento della posizione utente

Tali requisiti sono una differenza di almeno 1000 metri e 30 minuti dall'ultima posizione. Nel caso in cui un altro servizio presente nel dispositivo abbia già richiesto un aggiornamento della posizione ed è quindi disponibile una posizione aggiornata, la differenza di tempo scende a 5 minuti. Questo perché non sarà necessario un fix della posizione poiché già disponibile e quindi non si andrà a consumare ulteriore batteria, permettendo quindi un aggiornamento

più frequente.

### 3.3 Networking e sincronizzazione

Oltre a rendere la localizzazione dell'utente efficiente, richiedendo il minor numero possibile di fix della posizione pur avendo una posizione dell'utente precisa e sempre aggiornata, era necessario far sì che la sincronizzazione di tali posizioni gravasse anch'essa il meno possibile sulla batteria.

La sincronizzazione delle posizioni utilizza infatti il `JobScheduler` di Android, API che permette di gestire dei job chiamati `JobService` e avviarli al verificarsi di determinate circostanze [3]. Il job responsabile della sincronizzazione è il `SyncJobService` e viene gestito da una classe helper denominata `JobScheduler`, responsabile dello scheduling dei var job dell'applicazione.

```
fun scheduleSyncJob(userId: Long) {  
    val extras = PersistableBundle()  
    extras.putLong(SyncJobService.KEY_USER_ID, userId)  
  
    val builder = JobInfo.Builder(SyncJobService.ID,  
        ComponentName(context, SyncJobService::class.java))  
    builder.setRequiredNetworkType(JobInfo.NETWORK_TYPE_ANY)  
    builder.setPeriodic(TimeUnit.HOURS.toMillis(6))  
  
    builder.run {  
        setRequiresDeviceIdle(true)  
        setRequiresCharging(true)  
        setPersisted(true)  
        setExtras(extras)  
    }  
}
```

```
val jobScheduler =
    context.getSystemService(Context.JOB_SCHEDULER_SERVICE) as
    JobScheduler
val jobs = jobScheduler.allPendingJobs
if (!jobs.any { it.id == SyncJobService.ID }) {
    jobScheduler.schedule(builder.build())
}
}
```

#### Pianificazione del job per la sincronizzazione

Nello snippet di codice precedente vi è la configurazione del job presente nella classe `JobScheduler`. Per la sincronizzazione delle posizioni si è scelto di avviare la sincronizzazione delle posizioni ogni 6 ore, ma solo se il dispositivo si trova in stato di *idle* ed in carica. Quindi, passate 6 ore dall'ultima sincronizzazione, prima di avviarne una nuova il job attenderà che il dispositivo venga messo in carica. Nel caso in cui il dispositivo non venga caricato per più di 24 ore, viene avviato un sync forzato, qualsiasi siano le condizioni attuali.

Per quanto riguarda la sincronizzazione dell'ultima posizione dell'utente, posizione che può arrivare ad essere aggiornata fino a 12 volte ogni ora, la sincronizzazione è istantanea, poiché i dati da comunicare sono ridotti e non rappresentano un problema per la batteria o le performance del dispositivo.

Una volta stabilito quando effettuare la sincronizzazione, essa avviene in due fasi: prima si controlla che vi siano effettivamente posizioni in attesa di sincronizzazione e, nel caso in cui vi siano, vengono inviate al server. Una volta che l'invio si è concluso con successo, e quindi avendo ricevuto risposta positiva dal server, le posizioni presenti sul dispositivo vengono eliminate.

## 3.4 Design, interfaccia utente (UI) ed esperienza d'uso (UX)

Una volta descritto il back-end dell'applicazione, è il momento di parlare di come è stata sviluppata l'interfaccia utente (UI), cioè il front-end, per garantire un'esperienza d'uso all'altezza dell'applicazione.

Essendo un'applicazione Android nativa, per la realizzazione della UI sono state seguite le linee guida Google per il **Material Design** [4], creando un'interfaccia semplice, intuitiva e che potesse al contempo dare più informazioni e mezzi di interazione possibile. Il colore principale scelto è il verde chiaro con i dettagli (Bottoni, Switch, Badge) invece blu chiaro. Questa combinazione ha garantito all'applicazione un aspetto che richiama molto la natura e l'ecosostenibile.

La semplicità è stato un fattore determinante per la realizzazione della UI, poiché la probabilità che l'utente utilizzi l'applicazione con continuità viene ridotta se il suo utilizzo risulta difficile o poco chiaro.

All'avvio l'applicazione la prima cosa che viene mostrata è, ovviamente, la schermata di accesso, dove è possibile effettuare il login tramite l'inserimento di e-mail e password.

Una volta effettuato l'accesso, l'applicazione è composta principalmente da un'unica Activity suddivisa in 4 sezioni navigabili attraverso una barra di navigazione nella parte bassa dello schermo.

Le 4 sezioni sono, in ordine, Home, Classifiche, Match e Profilo. Ulteriori Activity sono una per la chat tra gli utenti ed una per la gestione delle impostazioni e preferenze dell'utente.

## Home

La **Home** rappresenta la sezione social dell'applicazione. Permette di visualizzare cosa sta succedendo all'interno della comunità (che può essere l'azienda o una determinata regione geografica), interagire con altri utenti ed accedere alla messaggistica privata.

Le informazioni vengono visualizzate come in un classico feed, ordinate per data, e possono mostrare chi sta condividendo l'auto in quel momento, chi invece ha concluso un'applicazione, l'aggiornamento dello stato di un utente, le informazioni da parte della propria azienda ed i risultati degli utenti ottenuti nel mese appena concluso.

In questa sezione, tramite l'apposita icona nella barra nella parte superiore dello schermo, è possibile accedere alla lista delle conversazioni. La messaggistica privata implementata nell'applicazione supporta, al momento, conversazioni tra massimo due utenti e permette lo scambio di soli messaggi di testo.



Figura 3.1: Home con il feed delle ultime news

## Classifiche

Nella sezione **Classifiche** sono visualizzate tutte le classifiche costantemente aggiornate. È possibile selezionare tramite dei menu a tendina il criterio utilizzato per la classifica, come le condivisioni effettuate o i chilometri condivisi, ed il periodo di riferimento, ovvero giornaliera, settimanale, mensile oppure totale.

Le classifiche visualizzate si riferiscono ai soli utenti della propria azienda. È comunque possibile visualizzare le classifiche globali aprendo la sezione dedicata tramite l'apposito pulsante nella action bar.












Classifica aziendale				← Classifica globale			
Condivisioni		Assoluto		Condivisioni		Assoluto	
1		Giacomo Pinto Lunipaper S.R.L.	8	1		Federico Mastrini Campus UNIPR	12
2		Samuela Portoghese Lunipaper S.R.L.	7	2		Giacomo Pinto Lunipaper S.R.L.	8
3		Oscar Conti Lunipaper S.R.L.	0	3		Samuela Portoghese Lunipaper S.R.L.	7
4		Ferdinando Loggia Lunipaper S.R.L.	0	4		Viola Lettiere Ospedale Sant'Antonio Abate	2
				5		Anna Bianchi Campus UNIPR	1
				6		Luca Rossi Campus UNIPR	1
				7		Oscar Conti Lunipaper S.R.L.	0

Figura 3.2: Classifica aziendale e globale

## Match

In **Match** troviamo la lista di tutti i match ordinati secondo il punteggio ottenuto. Selezionando un match vengono mostrati informazioni aggiuntive, come la provenienza dell'utente match e le informazioni sul punteggio, e la possibilità di contattare l'utente tramite messaggio.

Nel caso in cui non siano disponibili dei match (ad esempio perché l'utente è iscritto da poco e non ci sono ancora abbastanza informazioni sulle abitudini lavorative), in questa sezione vengono mostrati gli utenti che possono rappresentare potenziali match, come persone che abitano nella stessa zona e che lavorano in zone adiacenti al luogo di lavoro dell'utente.

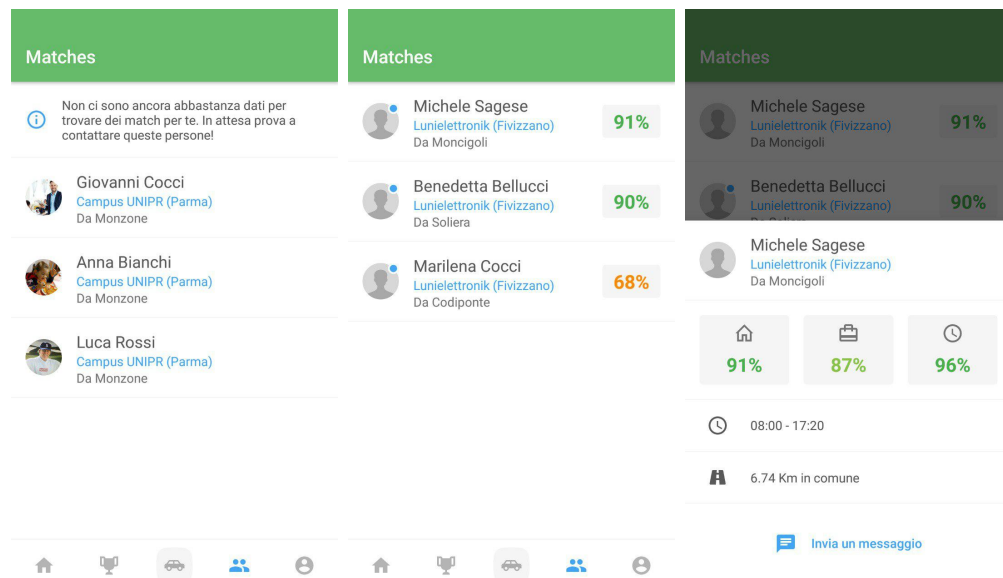


Figura 3.3: Lista dei match potenziali, match trovati e dialog con informazioni match

## Profilo

Il **Profilo** rappresenta la sezione personale dell'utente. È la parte centrale dell'implementazione della Gamification nell'applicazione.

Qui vengono mostrati i progressi dell'utente, quindi il suo livello attuale, i punti Karma e il progresso verso il livello successivo, le statistiche dettagliate e l'andamento personale degli ultimi 6 mesi. Questa parte sarà pubblicamente visibile, quindi qualsiasi utente potrà visitare il proprio profilo potendo mostrare la propria storia personale all'interno dell'applicazione.

Il livello utente è stato pensato ispirandosi ai classici giochi di ruolo, con colore e stile che varia con il progredire dei livelli.

In questa sezione, tramite l'apposito pulsante situato nella action bar, è possibile accedere alle impostazioni, dove è possibile modificare le preferenze riguardanti le notifiche, il tracking della posizione (scegliendo, ad esempio, di attivarlo solo in determinati momenti della giornata) ed attivare la modalità vacanza, modalità tramite la quale è possibile disattivare ogni funzione



dell'applicazione senza dover però effettuare l'accesso ed allo stesso momento informare gli altri utenti di non essere disponibile per eventuali condivisioni.

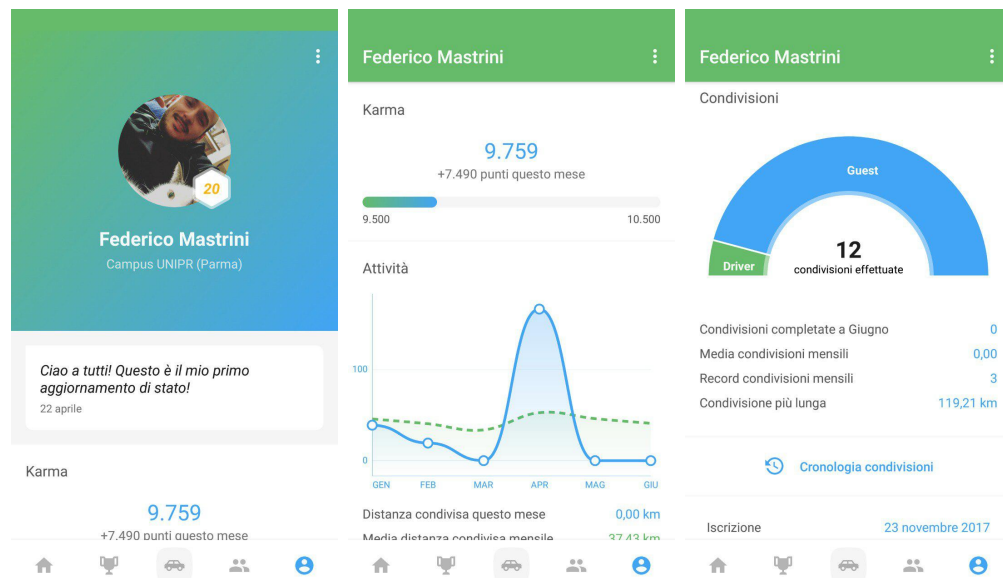


Figura 3.4: Profilo utente con le statistiche sulle condivisioni

## 3.5 Condivisione dell'auto

Il sistema di condivisione dell'auto, per poter raccogliere informazioni utili alle statistiche e assegnazione dei punti karma, doveva avere un sistema di consuntivazione che potesse verificare la correttezza delle condivisioni.

La soluzione sviluppata prevede l'utilizzo di QR Code univoci per la creazione di condivisioni e la geolocalizzazione degli utenti per la verifica della loro posizione. In questo modo, gli utenti sono accoppiati tramite l'assegnazione di un ID e, per l'accoppiamento, è necessario che essi siano insieme.

La stessa cosa avviene anche per il completamento della condivisione, la quale richiederà nuovamente la scansione dell'ID per verificare che si tratti dello stesso guest/host della partenza. In più, la verifica della posizione, consente di calcolare l'effettiva distanza condivisa. Il calcolo della distanza avviene

prendendo il percorso più breve tra i due punti registrati, questo per evitare che qualcuno possa “accumulare punti” allungando volutamente il tragitto: questo andrebbe contro ogni principio dell’applicazione. Il procedimento di creazione e completamento di una condivisione è molto semplice. Di seguito verrà descritta prendendo come esempio due utenti, chiamati utente A ed utente B.

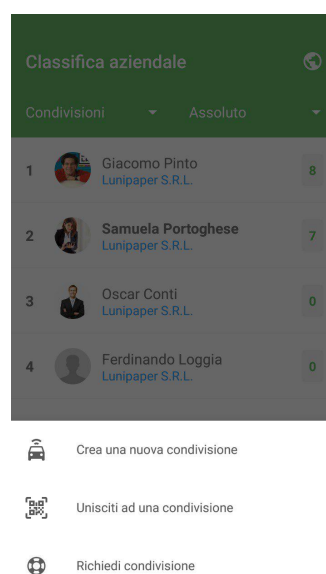


Figura 3.5: Menu per la gestione delle condivisioni

L’utente A decide di condividere la propria auto (host), quindi crea una nuova condivisione tramite l’apposita funzione accessibile premendo il pulsante centrale della barra di navigazione. Una volta che la condivisione è stata creata, gli verrà attribuito un ID unico il quale potrà essere mostrato sotto forma di QR Code.

L’utente B vuole utilizzare il passaggio concesso dall’utente A (guest), quindi, tramite la funzione accessibile sempre dal pulsante centrale della barra di navigazione, sceglie di partecipare ad una condivisione. A questo punto verrà richiesta la scansione dell’ID della condivisione alla quale unirsi.

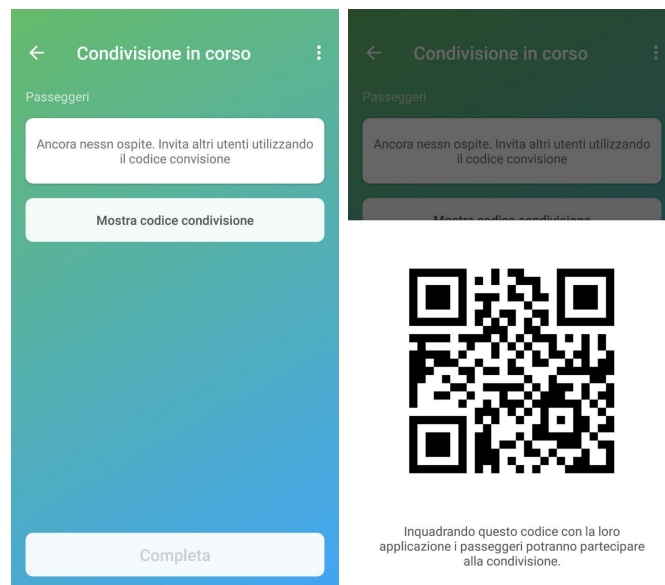


Figura 3.6: Dashboard per la gestione della condivisione (host) e QRCode generato

Per evitare eventuali aggiramenti del sistema (ad esempio, l'utente A crea condivisioni fittizie ed invia il QR code ad altri utenti per generare punti), una volta che il codice è scansionato, viene effettuata la verifica della posizione dei due utenti. Se si trovano vicini allora tutto è corretto e l'utente B viene aggiunto alla condivisione.

A questo punto non sono richieste ulteriori azioni per il proseguimento della condivisione. Ulteriori guest possono unirsi alla condivisione tramite il medesimo procedimento appena descritto.

Una volta che l'utente A o B sono arrivati a destinazione, il guest dovrà segnalare la condivisione come completata premendo sull'apposito pulsante nella dashboard della condivisione. Verrà richiesto nuovamente la scansione del codice QR e la verifica delle posizioni per accertarsi che i due utenti siano ancora insieme. Se tutto va a buon fine, la condivisione viene completata e, in base alla distanza condivisa, verranno assegnati i punti Karma ottenuti.

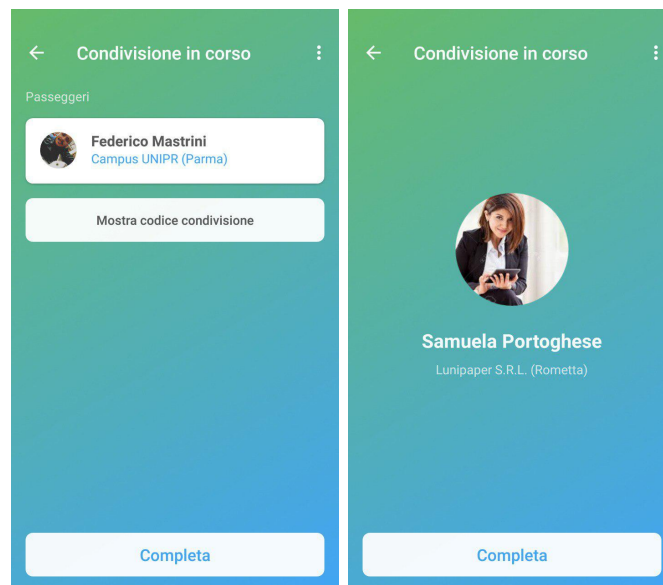


Figura 3.7: Ciò che vede l’host (sinistra) e il guest (destra) una volta completata l’unione alla condivisione

L’host, una volta che tutti i guest avranno segnalato il completamento della condivisione, potrà concluderla e ricevere i punti guadagnati. Nel caso in cui uno o più utenti non abbiano ancora segnalato il completamento, l’host non può concludere la condivisione.

Per evitare che un guest dimentichi di completare la condivisione mettendo quindi a rischio il punteggio degli altri utenti, l’host avrà la possibilità di espellere un guest dalla condivisione.

Un’ultima funzione dell’applicazione è quella di ricerca di emergenza o SOS. Tramite questa funzione sarà possibile ricercare utenti in zona disponibili per un passaggio. Questa funzione può risultare utile nel caso in cui il nostro passaggio dell’andata abbia avuto imprevisti e ci abbia lasciato senza altre soluzioni, oppure per qualsiasi altro tipo di emergenza. Il funzionamento è semplice e sfrutta l’ultima posizione registrata degli utenti. Avviata la ricerca, verranno mostrati gli utenti in zona che potrebbero fornire un passaggio, poiché residenti vicino alla propria abitazione oppure di passaggio. Una vol-

---

ta trovato l'utente sarà possibile accordarsi per una condivisione dell'auto tramite messaggistica privata.

## Capitolo 4

### Conclusioni e sviluppi futuri

In questo lavoro di tesi si è descritta la realizzazione e sviluppo di un'applicazione per la mobilità sostenibile che sfrutta il concetto di gamification per aumentare il coinvolgimento degli utenti e migliorare le loro performance.

L'inquinamento ed il traffico sono problemi a cui è necessario porre un rimedio, ed il sistema che si doveva andare a sviluppare doveva fornire uno strumento efficace a tale scopo.

Il risultato è un'applicazione che, tramite un sistema di ricompense per gli utenti che scelgono di non utilizzare la propria auto o usarla per accompagnare altri colleghi, semplifica la realizzazione ed adozione di un PSCL da parte delle aziende e, di conseguenza, riduce la formazione di traffico ed inquinamento.

L'applicazione sviluppata nasce dalla partecipazione al contest Mob App Awards 2017 organizzato dall'Università di Parma, nel quale si è aggiudicata il primo premio per il bando Infor S.r.l. [5], nel quale era richiesto lo sviluppo di un app che potesse raccogliere i dati sullo spostamento casa-lavoro e mettere in contatto le persone che condividevano tratti comuni o simili. Per la realizzazione si è, quindi, trovato nella gamification un ottimo alleato che potesse aiutare il sistema ad essere adottato.



La collaborazione con Infor S.r.l è continuata e dal progetto è nata **Home2Work** [6], applicazione che ha avuto l'interessamento di diverse aziende che hanno visto in essa un potenziale prodotto per il raggiungimento degli obiettivi posti dal PSCL.

Attualmente pensata per soli dispositivi Android, in futuro verrà sviluppata una versione iOS per essere resa disponibile su più dispositivi possibile. Vista la rapida adozione dei dispositivi wearables, sarebbe interessante svilupparne anche una versione per smartwatch, in modo da semplificare la consuntivazione delle condivisioni e il tracking degli utenti.

Un aspetto importante dell'applicazione è anche il lato social, il quale fa sì che il coinvolgimento tra gli utenti aumenti e di conseguenza aumenti l'efficacia del sistema. Tale aspetto potrebbe essere sviluppato in modo da creare una sorta di social network del car pooling, coinvolgendo sia gli utenti che le aziende stesse.

Il sistema di matching degli utenti si è rivelato un ottimo strumento per semplificare la ricerca di potenziali partner di condivisione, seppur attualmente supporti soltanto alcuni parametri come partenza, destinazione ed orari. Un suo potenziamento, ad esempio tramite l'estensione dei match ad anche tratti comuni condivisi, può rendere il sistema ulteriormente efficace.

Tale sistema, unito alle possibilità offerte dalla gamification e le potenzialità del social, costituisce uno strumento importante verso un'ottica di sensibilizzazione alla mobilità sostenibile, riducendo quindi l'inquinamento ambientale e migliorando la qualità della vita e del lavoro di ogni persona.

# Riferimenti

- [1] Dagger. *Github*. <https://google.github.io/dagger/>.
- [2] Activity recognition. *Google Developers*. <https://developers.google.com/location-context/activity-recognition/>.
- [3] Schedule jobs intelligently. *Google Developers*. <https://developer.android.com/topic/performance/scheduling>.
- [4] Material design. *Material.io*. <https://material.io/design/>.
- [5] Mob app awards 2017. *Università di Parma*. <https://www.unipr.it/notizie/11-luglio-ingegneria-consegna-del-mob-app-award>.
- [6] Home2work - la app per gli spostamenti casa-lavoro dei dipendenti. <http://home2work.it/>.