SAPIENZA
UNIVERSITÀ DI ROMA

Progettazione e sviluppo della visualizzazione dei parcheggi nell'applicazione GeneroCity Android

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Informatica
Corso di laurea in Informatica

Candidata
Jessica Frabotta
Matricola 1758527

Jessica Frabotta

Relatore
Emanuele Panizzi

Emanuele Panizzi

A.A. 2020-2021

Indice

Introduzione

Il problema della ricerca del parcheggio attanaglia milioni di persone ogni giorno nel mondo essendo fonte di dispendio di tempo, carburante e conseguentemente motivo di stress. Inoltre la ricerca del parcheggio ha un forte impatto anche sull'ambiente, oltre che sulla vita dei cittadini. Ed è proprio per risolvere questo importante problema che nasce il progetto GeneroCity.

GeneroCity è un'applicazione mobile che, facendo leva sulla generosità dei suoi utenti, ha come obiettivo principale quello di facilitare lo scambio di parcheggi tra automobilisti, rendendo l'operazione della ricerca del parcheggio più veloce e semplice.

L'obiettivo della mia attività di tirocinio è stata quella di creare nell'applicazione Android uno storico delle soste che potesse consentire all'utente una visualizzazione dei parcheggi effettuati fino a quel momento completa di tutte le informazioni essenziali. Per ogni automobile registrata sull'applicazione è possibile visualizzare, oltre all'ora di inizio e di fine del parcheggio, altre informazioni utili come la tipologia del parcheggio effettuato, la sua posizione, chi lo ha effettuato (l'app prevede di poter condividere una stessa macchina fra più persone) e se è stato ottenuto e/o ceduto attraverso uno scambio.

Prima di arrivare alla scrittura del codice è stato fatto un lavoro preliminare di analisi del problema cercando di capire quali fossero le informazioni importanti da mostrare nello storico delle soste e come dovessero essere disposte all'interno dell'interfaccia affinché fosse comprensibile e facile da usare. Fondamentale in questo senso è stata poi la progettazione grafica dell'interfaccia su carta che è servita da linea guida per l'implementazione. Sono stati fatti nel processo di sviluppo diversi test di usabilità per capire se ci fossero eventuali problemi o difficoltà degli utenti nel comprendere ed utilizzare l'interfaccia. In alcuni casi sono stati proposti diversi prototipi agli utenti per capire quale fosse quello migliore da integrare nell'applicazione.

Il presente elaborato è diviso in 4 capitoli. Nel primo verrà descritta in generale l'applicazione GeneroCity, le sue funzionalità principali e le motivazioni che hanno spinto a svilupparla. Nel secondo verranno presentati gli obiettivi del tirocinio e verranno mostrate le funzionalità sviluppate. Nel terzo capitolo ci si soffermerà sul codice e sui dettagli implementativi, analizzando le classi e i metodi introdotti. Nell'ultimo capitolo verrà fatto un breve sommario e saranno presentati alcuni possibili sviluppi futuri di GeneroCity legati al lavoro svolto.

Capitolo 1

GeneroCity

1.1 Il problema della ricerca di parcheggio

Nelle nostre menti, spesso, vi è un'associazione automatica tra le parole "città" e "traffico". Come testimoniano le classifiche mondiali sui livelli di congestione veicolare, ogni anno gli automobilisti passano un numero impressionante di ore imbottigliati nel traffico: il tempo di guida trascorso nel traffico per un cittadino a Roma nel 2019 è stato di circa 166 ore nell'arco dell'intero anno [1]. Ma c'è qualcosa di ancora più sconvolgente: si stima che il 30% del traffico in città sia causato da automobilisti in cerca di parcheggio [2]. Girare per ore in macchina alla ricerca di un posto non è solo fonte di enorme stress per la persona che sta cercando parcheggio ma ha anche enorme impatto sull'ambiente, sull'economia e in generale sulla vivibilità dei centri urbani. Secondo le indagini ISTAT del 2018 i problemi maggiormente sentiti dalle famiglie riguardo la zona in cui vivono sono stati il traffico (38,8%), l'inquinamento dell'aria (37,8%), e la difficoltà di parcheggio (35,7%) [3]. L'inquinamento ambientale è un problema importante nelle aree urbane densamente popolate, con molti residenti che soffrono di malattie e allergie causate da esso.
Il report State Of Global Air realizzato dall'Health Effect Institute (HEI), dall'Institute for Health Metrics and Evaluation (IHME) e dalla University of British Columbia [4] restituisce dati molto preoccupanti sull'incidenza dell'inquinamento atmosferico sull'aspettativa di vita dei cittadini: nel 2019 esso è il quarto fattore di morte al mondo, preceduto solo da malattie connesse all'obesità, fumo e ipertensione.

1.2 Smart Parking

Lo smart parking è una strategia di parcheggio che combina la tecnologia con l'innovazione umana, nel tentativo di utilizzare il minor numero di risorse possibili (carburante, tempo, spazio) per trovare un parcheggio. I vantaggi dello smart parking sono molteplici:

- Semplifica la ricerca del parcheggio che avviene attraverso app per dispositivi mobili.
- Riduce l'inquinamento atmosferico generato dalla ricerca del parcheggio: secondo una ricerca condotta dall'università di Kaiserslautern in Germania con lo smart parking è possibile risparmiare fino a 900.000 tonnellate di CO2 [5].
- Aiuta a decongestionare il traffico nelle zone urbane.

- Fa risparmiare i cittadini che trovando parcheggio con più facilità spendono meno per il carburante.
- Migliora il benessere sociale ed economico della collettività.

È in questo contesto di sostenibilità che nasce il progetto GeneroCity.

1.3 Il progetto GeneroCity

GeneroCity è un'applicazione mobile di smart parking sviluppata sia per sistemi iOS che Android dal Gamification Lab del Dipartimento di Informatica dell'Università degli Studi di Roma "La Sapienza" sotto la guida del prof. Emanuele Panizzi.

Figura 1. Logo di GeneroCity

Il concetto alla base del funzionamento di GeneroCity è lo scambio: l'app permette ad un utente che sta per lasciare un parcheggio di segnalarlo, in maniera tale che un altro utente interessato a parcheggiare in quella zona e in quella fascia oraria, possa prendere quel posteggio in maniera semplice e senza inutile dispendio di tempo. Il processo di scambio è incentivato da meccanismi di gamification: l'utente che cede un parcheggio guadagna dei GeneroCoins, una valuta virtuale utilizzabile solo all'interno dell'app, che poi reinvestirà per cercare parcheggio quando ne avrà bisogno. In questa maniera più posti un utente lascia più ne potrà cercare. Quindi più un utente sarà generoso e più otterrà benefici dall'app. Da qui nasce il nome dell'applicazione. GeneroCity cerca di incoraggiare, in qualche modo, a considerare gli altri automobilisti come degli alleati nel cercare parcheggio e non necessariamente dei concorrenti.

Affinché possa essere utilizzata l'app chiaramente c'è bisogno di essere connessi ad Internet e concedere i permessi di localizzazione poiché per poter utilizzare le funzionalità offerte da GeneroCity, come la ricerca dei parcheggi, c'è la necessità di conoscere la posizione dell'utente.

1.4 Le funzionalità di GeneroCity

Alla prima installazione di GeneroCity, dopo una serie di schermate introduttive che fungono da tutorial, viene chiesto all'utente di inserire un nickname che servirà ad identificarlo all'interno del sistema. Dopodiché si può accedere a tutte le funzionalità principali.

1.4.1 Creazione e gestione di un'automobile

Per poter cominciare ad utilizzare a pieno l'applicazione è necessario registrare almeno un'automobile. Per farlo bisogna inserire alcune informazioni come la targa, la marca, il modello, la dimensione e il colore. La ragione per cui sono richiesti questi dati è per rendere più agevole il riconoscimento dell'altro utente nel momento in cui si va ad effettuare uno scambio. Si possono inserire nel sistema diverse automobili per lo stesso utente. Una scorciatoia per rendere più immediata la registrazione di un'automobile è aggiungerla dai propri dispositivi Bluetooth associati.
In generale collegare lo smartphone alla propria auto attraverso il Bluetooth è un'operazione fondamentale per l'utilizzo di GeneroCity: l'applicazione riesce a capire automaticamente se l'utente ha effettuato un parcheggio in base al fatto che la connessione Bluetooth tra la macchina e lo smartphone è ancora attiva o meno.
Se la targa inserita nel momento in cui viene creata una macchina è già presente sul database, viene chiesto all'utente di mandare una richiesta di condivisione della macchina alla persona che l'ha inserita in precedenza. L'utente può in ogni caso modificare o eliminare un'automobile dal sistema in qualsiasi momento.
Per renderne più comoda ed immediata la gestione, sulla home viene creata una card per ogni automobile registrata.

1.4.2 Condivisione di un'automobile

Come è stato già introdotto in precedenza è possibile in GeneroCity che più persone (in genere un'utente e i propri familiari) condividano la stessa auto. Nell'app l'insieme degli utenti utilizzatori della stessa automobile vengono indicati proprio con il termine "famiglia". Per entrare a far parte della famiglia relativa ad un'auto basta che l'utente che ha inserito inizialmente una determinata targa accetti la richiesta di condivisione di altri eventuali utenti che registrano un'automobile con quella stessa targa.
Ogni membro della famiglia può vedere chi è stato l'ultimo ad utilizzare la macchina, dove è stata parcheggiata l'ultima volta e lo storico delle soste e degli scambi. I membri della famiglia possono inoltre comunicare tra di loro nell'app attraverso una chat.

1.4.3 Scambio del parcheggio

Lo scambio del parcheggio, come già menzionato, è il fulcro dell'intera applicazione. Nello svolgimento di questa funzionalità possiamo definire due attori: il "giver" ossia l'utente che lascia il parcheggio e il "taker", l'utente che lo vuole prendere.
Il giver per segnalare agli altri utenti dell'app che sta per lasciare il proprio posto clicca il bottone "lascia posto" presente nella card della macchina parcheggiata. In maniera complementare il taker per cercare parcheggio deve cliccare sul bottone "cerca posto" sulla card della macchina che sta usando. A questo punto al giver e al taker vengono chieste le informazioni necessarie per riuscire ad esaudire le loro richieste: il giver comunica al sistema quando ha intenzione di lasciare il posto mentre il taker inserisce le info riguardanti dove e quando intende cercare parcheggio.
Se gli orari e i luoghi delle richieste sono abbastanza vicini, il sistema associa i due utenti. Al giver viene segnalato che qualcuno vuole prendere il suo posto e quindi gli viene chiesto di aspettare. Parallelamente il taker si avvicina al punto in cui il giver lo aspetta per portare a termine lo scambio. Se il match è andato a buon fine, il taker spende GeneroCoins mentre il giver ne guadagna.
Durante l'intero processo, ovviamente, entrambi gli attori possono decidere in qualsiasi momento di annullare lo scambio.

Capitolo 2

Integrazione dello storico delle soste in GeneroCity

2.1 Storico delle soste: a cosa serve e perché integrarlo in GeneroCity Android

Lo storico delle soste consente di visualizzare in ordine cronologico tutti i parcheggi effettuati fino a quel momento e di consultare una serie di informazioni importanti relative alle soste quali l'ora di inizio e di fine, la tipologia del parcheggio effettuato, la sua posizione, chi lo ha effettuato e se è stato ottenuto e/o ceduto attraverso uno scambio. Esso fornisce quindi anche un resoconto dei GeneroCoins guadagnati o spesi per ogni scambio.
Inoltre lo storico può essere visualizzato da tutti i familiari di un'auto che possono in questo modo tracciare i parcheggi fatti dagli altri membri.

Un lavoro simile a questo era già stato portato avanti sulla piattaforma iOS quindi si è voluto riallineare le due versioni. Tuttavia l'interfaccia ottenuta è abbastanza differente per cercare di integrare componenti tipici di Android che gli utenti potessero immediatamente riconoscere.

2.2 Come si effettua un parcheggio

Nell'app, per ora, esistono due modi per effettuare un parcheggio:

- Attraverso il Bluetooth: affinché si possa effettuare un parcheggio con questa modalità, è necessario innanzitutto che l'utente colleghi lo smartphone all'auto tramite il Bluetooth. Quando l'utente si disconnette, l'app capirà che l'utente non sta più utilizzando l'auto e che quindi quest'ultima è parcheggiata.
- Portando a termine uno scambio: il taker spende dei GeneroCoins per prendere il parcheggio di un giver che, invece, per cederlo ne guadagna.

2.3 Progettazione dello storico delle soste

Le prime fasi del lavoro di tirocinio sono state incentrate sull'analizzare e capire quali fossero le informazioni importanti da visualizzare nello storico delle soste e quale fosse la maniera migliore di inserirle nell'interfaccia affinché essa risultasse intuitiva e facile da usare.

In una vecchia versione dell'applicazione la visualizzazione dei parcheggi effettuati fino a quel momento avveniva su un'unica schermata attraverso una ExpandableListView, una tipologia di lista che per ogni suo elemento ha la capacità di espandere e/o comprimere le relative informazioni aggiuntive in base al tocco dell'utente che decide di leggere o meno quelle informazioni.
Nella nuova versione dello storico si è valutato se usare di nuovo questo tipo di struttura ma si è quasi subito compreso che la mole delle informazioni importanti da mostrare avrebbe reso tale lista troppo poco leggibile.

Si è deciso quindi di sviluppare lo storico su due schermate: nella prima c'è una lista con tutti i parcheggi effettuati mostrati in ordine cronologico e cliccando su ogni singolo parcheggio è possibile accedere ad una seconda schermata in cui sono visualizzati tutti i dettagli di quel parcheggio e una mappa che ne mostra la posizione. Nella schermata di dettaglio hanno inoltre trovato collocazione le informazioni riguardanti gli scambi effettuati.
Una volta scelta questa soluzione è stata prima progettata l'interfaccia su carta e poi si è passato alla vera e propria implementazione di essa.

2.3.1 Struttura dello storico

Come anticipato, lo storico delle soste è articolato su due schermate, una che visualizza in una lista tutti i parcheggi effettuati e una di dettaglio.

Si accede allo storico dalla home dell'applicazione: sulle card che rappresentano ogni automobile registrata viene mostrata, a seconda che la macchina sia usata o meno, nel primo caso la stringa "La stai usando" [v. fig. 2], nel secondo caso una stringa in cui viene indicato l'indirizzo del luogo dove la macchina è stata posteggiata [v. fig. 3]. In entrambi i casi se la persona che la sta guidando o che l'ha parcheggiata non è l'utente che in questo momento sta visualizzando la home ma un suo familiare, viene indicato anche il nickname dell'utente in questione.

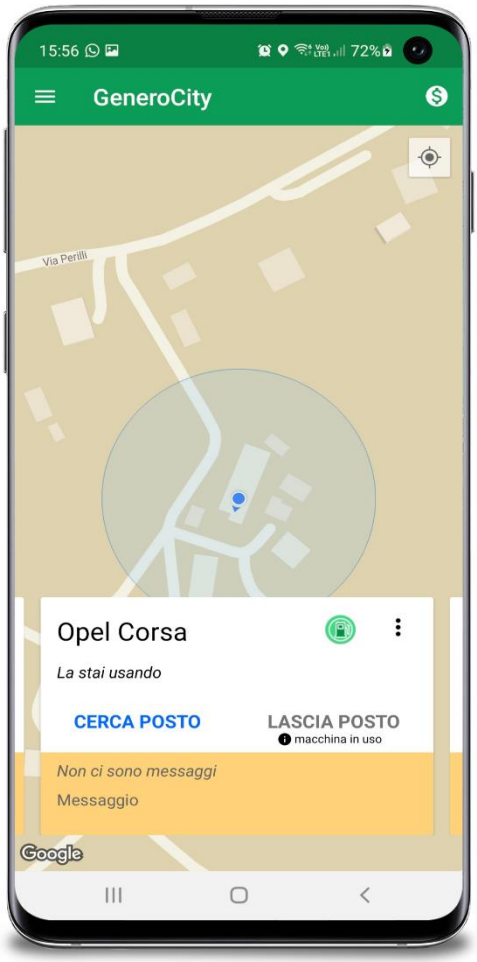

Figura 2. La card quando la macchina è in uso

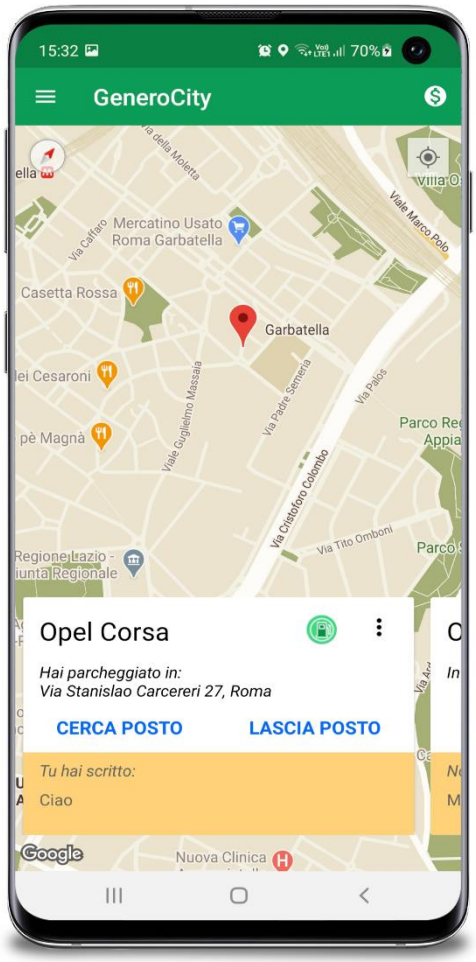

Figura 3. La card quando la macchina è parcheggiata

A seconda che la macchina sia in uso o parcheggiata cambieranno alcune informazioni nella schermata iniziale dello storico: al di sopra della lista dei parcheggi terminati è posta una card che nel caso in cui l'auto è in uso, mostra il nickname del membro della famiglia che la sta usando [v. fig. 4], se l'auto è parcheggiata mostra la posizione di quest'ultima, quando è iniziata la sosta (ma non la data e l'ora di fine poiché quel parcheggio è quello attuale, quindi è ancora in corso) e l'utente che l'ha parcheggiata [v. fig. 5].

Figura 4. Lo storico dei parcheggi quando l'auto è in uso

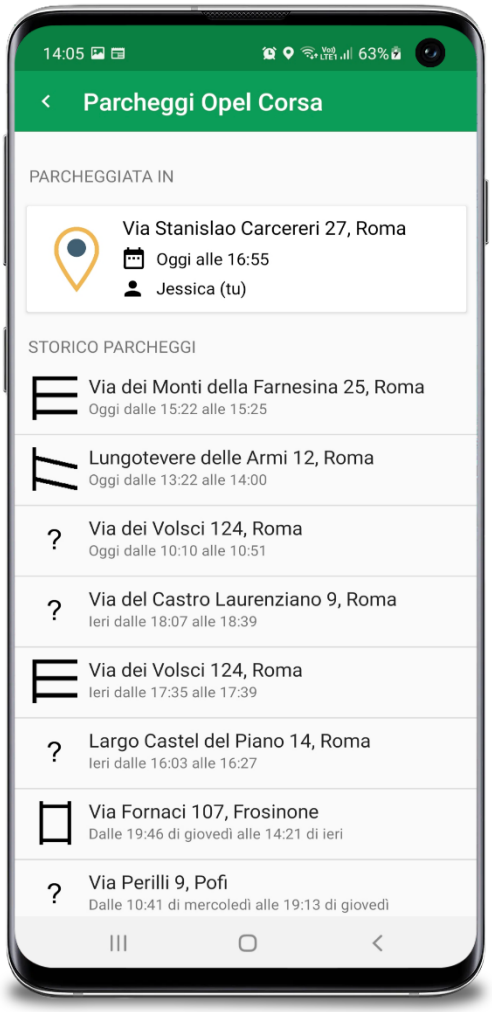

Figura 5. Lo storico dei parcheggi quando l'auto è parcheggiata

I parcheggi elencati nella lista sono tutti parcheggi terminati e sono mostrati in ordine cronologico partendo dal più recente. Le informazioni riportate per ognuno di essi sono l'indirizzo in cui è stata effettuata la sosta, la data di inizio e di fine del parcheggio e la sua tipologia, rappresentata attraverso un'icona [v. fig. 6].

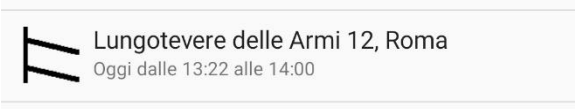

Figura 6. Struttura parcheggio passato

L'indirizzo è formattato in maniera tale da mostrare in primo luogo la via, piazza, viale, corso ecc. in cui è avvenuto il parcheggio seguito dal numero civico e dalla località.

Per quanto riguarda invece la formattazione delle date, si è cercato di renderle più leggibili possibile andando a rispettare i seguenti criteri:

- se il parcheggio è iniziato e finito nello stesso giorno la data verrà mostrata con la formattazione:
  <giorno> dalle <orario di inizio> alle <orario di fine> [v. fig. 7]
- se il parcheggio è iniziato in un giorno e finito in un altro, allora la data verrà mostrata con questo formato:
  Dalle <orario di inizio> del <giorno di inizio> alle <orario di fine> del <giorno di fine> [v. fig. 8].

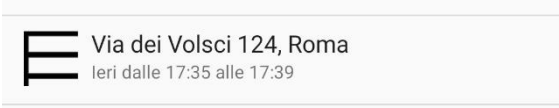

Figura 7. Formattazione della data di una sosta iniziata e finita nello stesso giorno

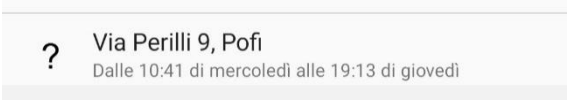

Figura 8. Formattazione della data di una sosta iniziata e finita in giorni diversi

Per rendere in generale le date più leggibili per i giorni più recenti in cui si è effettuato un parcheggio, anziché mostrare la data completa con giorno, mese ed anno, vengono mostrate, quando opportuno, le stringhe "oggi", "ieri" e il giorno della settimana se il parcheggio in questione è avvenuto non più di sei giorni prima.

Come detto, nello storico, accanto ad ogni parcheggio, c'è un'icona che rappresenta la tipologia di parcheggio effettuata. Essa può essere:

- A spina [v. fig. 9]
- Parallelo [v. fig. 10]
- A pettine [v. fig. 11]

Le icone con il punto interrogativo vanno ad indicare che la tipologia di parcheggio non è stata selezionata dall'utente. Nella schermata di dettaglio del parcheggio è possibile selezionare la tipologia di parcheggio se non è stata indicata in precedenza o modificarla se quella già indicata fosse scorretta.

Figura 9. Parcheggio parallelo

Figura 10. Parcheggio a pettine

Figura 11. Parcheggio a spina

Nella parte alta della schermata è presente una app bar [v. fig. 12] in cui viene indicato il nome della macchina di cui si stanno visualizzando i parcheggi. Essa aiuta a fornire un contesto alla schermata. Nell'app bar è implementato anche un bottone di back con cui l'utente può in qualsiasi momento tornare alla home dell'applicazione.

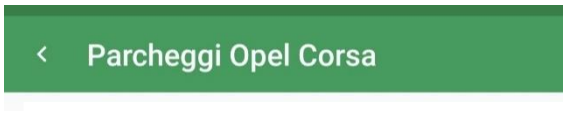

Figura 12. App bar della schermata

Ovviamente non c'è un limite al numero di parcheggi che un utente può effettuare e visualizzare nello storico, per questo motivo effettuare una chiamata al server per restituire tutti i parcheggi può rallentare la visualizzazione e creare inutile sovraccarico. Per questo motivo vengono mostrati nella lista solo i 16 parcheggi più recenti. Se si vuole visualizzarne altri, basta premere il buttone "altri parcheggi" presente in fondo alla lista [v. fig. 13]. Ogni volta che viene premuto questo bottone viene fatta una nuova chiamata al server e vengono mostrati così altri 16 parcheggi. Quando non ci sono più altri parcheggi da visualizzare, il bottone non è più cliccabile e cambia colore per dare un feedback all'utente di tale variazione [v. fig. 14].

Figura 13. Il bottone "Altri parcheggi" e la sua disposizione

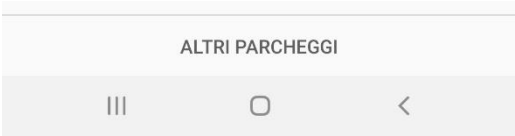

Figura 14. Il bottone "Altri parcheggi" quando non ci sono più parcheggi da mostrare

Se l'utente non ha ancora effettuato dei parcheggi lo storico cambierà di conseguenza mostrando la scritta "non hai effettuato parcheggi". Quella rappresentata nella figura 15 è la schermata che compare ad un utente che cerca di visualizzare lo storico quando ha installato l'app per la prima volta e ha collegato una macchina al Bluetooth ma senza effettuare parcheggi.

Figura 15. Aspetto dello storico se non sono ancora state effettuate soste

2.3.2 Schermata di dettaglio

È possibile accedere alla schermata di dettaglio di un parcheggio cliccando su di esso nella lista.
Questa seconda schermata amplia lo storico perché per ogni parcheggio fornisce informazioni aggiuntive come quelle relative agli scambi e ai GeneroCoins conseguentemente spesi o guadagnati [v. fig. 16].

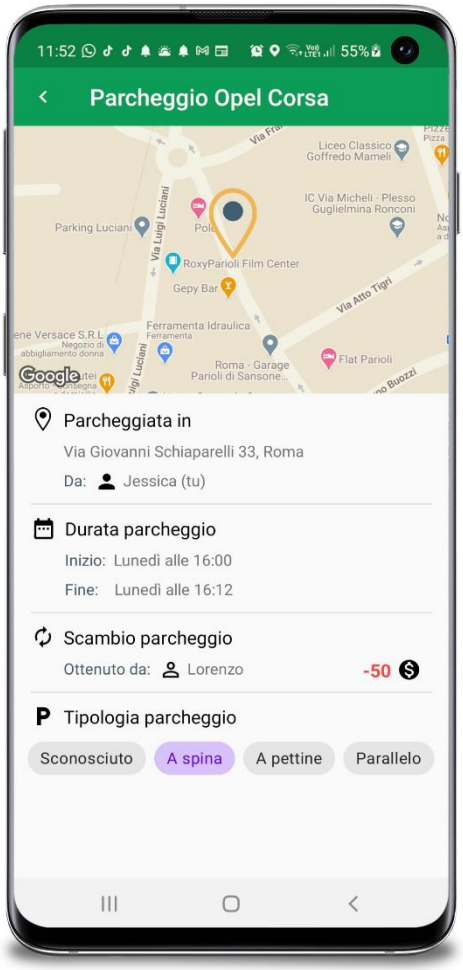

Figura 16. Schermata di dettaglio

Nella parte alta della schermata è posta una app bar simile a quella mostrata nella schermata precedente.
Subito sotto ad essa è presente una mappa statica che permette di visualizzare il punto in cui è stato effettuato il parcheggio [v. fig. 17]. Per complementare questa informazione è presente anche una sezione della schermata in cui viene trascritto l'indirizzo esatto della sosta e il nome dell'utente che l'ha effettuata [v. fig. 18].

Le varie sezioni della schermata sono suddivise da una sottile linea grigia chiamate divider che consentono di raggruppare graficamente le informazioni affini e parallelamente di separare quelle di natura differente.

Figura 17. Mappa

Figura 18. Indirizzo del parcheggio e utente che lo ha effettuato

Nella seconda sezione di informazioni troviamo indicata la durata del parcheggio. La data di inizio e di fine sono disposte su due righe [v. fig. 19].

Figura 19. Inizio e fine parcheggio

Nella terza sezione c'è l'informazione probabilmente più importante tra quelle riportate nella schermata ossia quella riguardante gli scambi. Se il parcheggio è stato ottenuto attraverso uno scambio verrà mostrato il nickname del giver e la quantità di GeneroCoins spesi. Analogamente se il parcheggio è stato ceduto verrà mostrato il nickname del taker e la somma in GeneroCoins guadagnata. In questa sezione vengono tracciati anche gli scambi annullati o in generale non andati a buon fine.

Da notare che per differenziare gli utenti membri della famiglia di un'auto e tutti gli altri utenti, si è scelto, all'interno dell'intero storico, di rappresentarli attraverso icone diverse, rispettivamente una "piena" colorata di nero [v. fig. 20] e una "vuota" con solo la linea di contorno [v. fig. 21].

Figura 20. Icona per i familiari

Figura 21. Icona per gli utenti non appartenenti alla famiglia

Nell'ultima sezione è possibile selezionare la tipologia del parcheggio effettuata, se questa non è stata selezionata in precedenza [v. fig. 22].

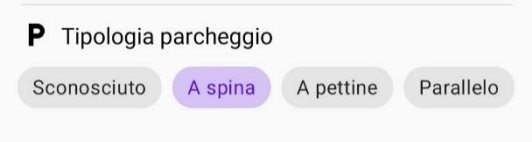

Figura 22. Selettore per la tipologia di parcheggio

Capitolo 3

Implementazione

3.1 Richiesta lista parcheggi

Quando viene aperta la schermata iniziale dello storico viene effettuata una richiesta di GET al server di GeneroCity con lo scopo di ottenere la lista dei 16 parcheggi più recenti effettuati dall'utente con quell'auto. La richiesta al server, come tutte quelle effettuate nell'app, usa la libreria Retrofit.

```
@GET("/car/{cid}/park/")
Call<List<Park>> getParkListByParkId(@Path("cid") String cid, @Query("parkid") Integer parkid);
```

Figura 23. La richiesta di GET effettuata con la libreria Retrofit

Il metodo getParkListById prende in input il Cid, ossia l'id con il quale una macchina viene registrata sul server e un parkId, affinché vengano ritornati i 16 parcheggi precedenti al parcheggio con tale parkId.

```
if (cid != null) {
    app = (GCApplication) getApplication();
    app.api().getParkListByParkId(cid,  parkid: 0).enqueue(new Callback<List<Park>>() {
        @Override
        public void onResponse(Call<List<Park>> call, Response<List<Park>> response) {
            if (response.isSuccessful()) {
                if (response.body() == null) {
                    TextView noParks = findViewById(R.id.no_parks);
                    noParks.setVisibility(View.VISIBLE);
                }
                try {
                    createListItem(response);
                    setCurrentPark(response);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        @Override
        public void onFailure(Call<List<Park>> call, Throwable t) {
            Toast.makeText(app,  text: "Non siamo riusciti a connetterci al server", Toast.LENGTH_LONG).show();
        }
    });
}
```

Figura 24. Metodo per ottenere la lista dei parcheggi

Dalla chiamata viene restituita una lista di oggetti di tipo Park. Se la richiesta va a buon fine ma il body della risposta è vuoto (ossia non ci sono parcheggi effettuati per quella macchina), viene mostrato il layout nella figura 15. Se invece il body non è vuoto i parcheggi ottenuti verranno utilizzati per ricavare le informazioni da porre nella card della sosta corrente e nella lista dei parcheggi passati. Se la richiesta non va a buon fine viene mostrato un toast con un messaggio di errore.

3.2 Creazione della lista dei parcheggi passati

Il metodo createListItem prendendo in input i parcheggi ottenuti dal server sotto forma di lista contenente oggetti di tipo Park, provvede ad inserire ogni sosta in un item della ListView dei parcheggi passati. Per prendere ogni singolo parcheggio viene utilizzato un ciclo for.
La lista è costruita in maniera tale da poter inserire le informazioni su due righe, la prima contenente l'indirizzo della sosta e la seconda l'orario di inizio e fine. Per escludere il parcheggio corrente per quell'auto, viene fatto un controllo sull'istante di fine del parcheggio, se esso esiste, significa che si tratta di un parcheggio passato.
Attraverso i metodi getParkLat e getParklon vengono ottenute le coordinate del parcheggio necessarie per effettuare il reverse geocoding e ottenere la posizione sotto forma di un indirizzo leggibile dall'uomo.

```
public void createListItem(Response<List<Park>> response) throws IOException {
    if (response.body() != null) {
        HashMap<String, String> item;
        for (int i = 0; i < response.body().size(); i++) {
            item = new HashMap<>();
            Geocoder geocoder = new Geocoder( context: this);
            Date start = response.body().get(i).getStarttime();
            Date end = response.body().get(i).getEndtime();
            if (end != null) {
                List<Address> addresses = geocoder.getFromLocation(response.body().get(i).getParklat(),
                        response.body().get(i).getParklon(), maxResults: 1);
                item.put("line1", addresses.get(0).getThoroughfare() + " "
                        + addresses.get(0).getFeatureName() + ", " + addresses.get(0).getLocality());
```

Figura 25. Metodo per inserire le informazioni nella lista

Per l'inserimento della data di inizio e fine il discorso è più complesso, quindi verrà trattato nel prossimo paragrafo.

3.3 Formattazione delle date

Come descritto nel paragrafo 2.3.1, la formattazione delle date varia a seconda che il parcheggio inizi e finisca nello stesso giorno o in giorni diversi. Il metodo setParkDateListItem prendendo in input le date e gli orari di inizio e fine sosta insieme ad un item della lista, effettua vari controlli per riuscire a scrivere le date nel formato voluto, andando anche a visualizzare, per leggibilità, le stringhe "oggi" e "ieri" quando opportuno.

```
    if (!startDay.equals(todayDate) && !startDay.equals(yesterdayDate) && !endDay.equals(todayDate)
            && !endDay.equals(yesterdayDate) && !startDay.equals(endDay)) {
        item.put( k: "line2", v: "Dalle " + startTime + stringDelOrDiStart +
                startParkDayOfTheWeek + " alle " + endTime + stringDelOrDiEnd + endParkDayOfTheWeekDay);
    } else if (!startDay.equals(todayDate) && !startDay.equals(yesterdayDate)
            && !endDay.equals(todayDate) && !endDay.equals(yesterdayDate)) {
        item.put( k: "line2",   v: startParkDayOfTheWeek + " dalle " + startTime + " alle " + endTime);
    } else if (startDay.equals(todayDate) && endDay.equals(todayDate)) {
        item.put( k: "line2", v: "Oggi dalle " + startTime + " alle " + endTime);
    } else if (startDay.equals(yesterdayDate) && endDay.equals(yesterdayDate)) {
        item.put( k: "line2", v: "Ieri dalle " + startTime + " alle " + endTime);
    } else if (startDay.equals(yesterdayDate) && endDay.equals(todayDate)) {
        item.put( k: "line2", v: "Dalle " + startTime + " di ieri alle " + endTime + " di oggi");
    } else if (!startDay.equals(todayDate) && !startDay.equals(yesterdayDate) && endDay.equals(todayDate)) {
        item.put( k: "line2", v: "Dalle " + startTime + stringDelOrDiStart + startParkDayOfTheWeek
                + " alle " + endTime + " di oggi");
    } else if (!startDay.equals(todayDate) && !startDay.equals(yesterdayDate)) {
        item.put( k: "line2", v: "Dalle " + startTime + stringDelOrDiStart + startParkDayOfTheWeek
                + " alle " + endTime + " di ieri");
    }
}
```

Figura 26. Tutti i controlli per ottenere la formattazione delle date

Le stringhe StartParkDayOfTheWeek e EndParkDayOfTheWeek mostrate nella figura 26 vengono restituite dal metodo returnDayOfTheWeek che prende in input una stringa di una data. Nel metodo c'è un ciclo for che va da due a sei perché il suo obiettivo è quello di mostrare correttamente una stringa con il giorno della settimana se il parcheggio in questione è avvenuto in settimana. Il ciclo parte da due perché i casi in cui bisogna visualizzare "ieri" e "oggi" sono già trattati da setParkDateListItem. Dunque viene fatto un confronto tra la data passata in input e la data relativa ad ognuno dei sei giorni precedenti ad oggi.

```java
public String returnDayOfTheWeek(String day) {
    String dayFormat = day;
    for (int i = 2; i <= 6; i++) {
        Calendar cal = Calendar.getInstance();
        cal.set(Calendar.DAY_OF_MONTH, cal.get(Calendar.DAY_OF_MONTH) - i);
        Date dayAgoIesimo = cal.getTime();
        String dateFormat = new SimpleDateFormat( pattern: "dd/MM/yyyy", ITALIAN).format(dayAgoIesimo);
        String dayOfTheWeek = new SimpleDateFormat( pattern: "EEE", ITALIAN).format(dayAgoIesimo);
        dayOfTheWeek = dayOfTheWeek.substring(0, 1).toLowerCase(ITALIAN) + dayOfTheWeek.substring(1);
        if (day.equals(dateFormat)) {
            switch (dayOfTheWeek) {
                case "lun":
                    dayFormat = "lunedì";
                    break;
                case "mar":
                    dayFormat = "martedì";
                    break;
                case "mer":
                    dayFormat = "mercoledì";
                    break;
                case "gio":
                    dayFormat = "giovedì";
                    break;
                case "ven":
                    dayFormat = "venerdì";
                    break;
                case "sab":
                    dayFormat = "sabato";
                    break;
                case "dom":
                    dayFormat = "domenica";
                    break;
                default:
                    dayFormat = day;
                    break;
            }
        }
    }
    return dayFormat;
}
```

Figura 26. Il metodo returnDayOfTheWeek

3.4 Tipologia di parcheggio

Il metodo createListItem si occupa anche di inserire per ogni item della lista nella prima schermata l'icona della tipologia di parcheggio effettuata. Con il metodo getParkType viene ottenuto il parkType che può essere angle, parallel, perpendicular o invalid e viene mostrata l'immagine del tipo di parcheggio per ognuno dei primi tre, mentre se il parkType è vuoto o invalid viene mostrata un'immagine con un punto interrogativo che va ad indicare che quel parcheggio non ha parkType assegnato.

```
String parkType = response.body().get(i).getParkType();
if ("angle".equals(parkType)) {
    item.put("parkTypeIcon", String.valueOf(R.drawable.spina));
} else if ("parallel".equals(parkType)) {
    item.put("parkTypeIcon", String.valueOf(R.drawable.parallelo));
} else if ("perpendicular".equals(parkType)) {
    item.put("parkTypeIcon", String.valueOf(R.drawable.pettine));
} else {
    item.put("parkTypeIcon", String.valueOf(R.drawable.question_mark));
}
```

Figura 27. Assegnazione immagine tipologia di parcheggio

3.5 Stato attuale dell'auto

Come mostrato nelle figure 4 e 5 al di sopra della lista dei parcheggi terminati c'è una card che a seconda dello stato attuale dell'auto, mostra informazioni differenti. Anche le card in cui vengono rappresentate le auto presenti nella home, cambiano in base al fatto che la macchina sia in uso o parcheggiata quindi l'informazione riguardo lo stato dell'auto viene passato tra le classi che gestiscono le due schermate attraverso un Intent. Se l'auto non è parcheggiata viene mostrata la card della macchina in uso, la card del parcheggio attuale altrimenti.

```
    Bundle bundle = getIntent().getExtras();
    boolean isParked = false;
    if (bundle != null) {
        isParked = bundle.getBoolean( key: "isparked");
    }

    if (!isParked) {
        CardView cardCurrentPark = findViewById(R.id.card_current_park_info);
        cardCurrentPark.setVisibility(View.GONE);
        String usedby = getIntent().getStringExtra( name: "usedby");
        TextView textViewParcheggiataIn = findViewById(R.id.textViewParcheggiataIn);
        textViewParcheggiataIn.setVisibility(View.GONE);
        TextView currentUser = findViewById(R.id.current_user);
        currentUser.setText(usedby);
    } else {
        CardView cardUsedBy = findViewById(R.id.card_current_car_user_info);
        cardUsedBy.setVisibility(View.GONE);
        TextView textViewLaStaiUsando = findViewById(R.id.textViewLaStaiUsando);
        textViewLaStaiUsando.setVisibility(View.GONE);
    }
}
```

Figura 28. Meccanismo con cui viene decisa la card dello stato da mostrare

3.6 Caricamento di altri parcheggi nella lista

Come descritto nel capitolo 3.1, inizialmente la lista dei parcheggi passati contiene esattamente 16 parcheggi. Come ultimo elemento della lista è presente il bottone "Altri parcheggi", realizzato secondo lo stile di un Text Button del Material Design.

```
ListView parkListView = (ListView) findViewById(R.id.list);
Button buttonLoadMore = new Button( context: this,  attrs: null, android.R.attr.borderlessButtonStyle);
buttonLoadMore.setTextColor(getResources().getColor(R.color.deep_purple));
buttonLoadMore.setText("Altri parcheggi");
parkListView.addFooterView(buttonLoadMore);
```

Figura 29. Realizzazione del bottone "Altri parcheggi"

Cliccando il bottone si andrà a chiamare di nuovo il metodo getParkListByParkId passando questa volta come input oltre al cid, il valore intero lastParkId.
lastParkId è l'id dell'ultimo parcheggio che si trova nella lista di oggetti Park ottenuta dal server dopo ogni chiamata. Così facendo verranno ritornati altri 16 parcheggi che verranno inseriti nella lista evitando un inutile sovraccarico del server ad ogni apertura dello storico.

```
buttonLoadMore.setOnClickListener(view -> {
    if (cid != null) {
        app = (GCApplication) getApplication();
        app.api().getParkListByParkId(cid, lastParkId).enqueue(new Callback<List<Park>>() {
            @Override
            public void onResponse(Call<List<Park>> call, Response<List<Park>> response) {
                if (response.isSuccessful()) {
                    if (response.body() != null) {
                        parkListView.post(() -> {
                            if (response.body() != null) {
                                numberClickLoadMoreButton++;
                            }
                            parkListView.setSelectionFromTop( position: parkListView.getFirstVisiblePosition()
                                    + 8 * numberClickLoadMoreButton,  y: 0);
                        });
                        try {
                            createListItem(response);
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    } else {
                        buttonLoadMore.setClickable(false);
                        buttonLoadMore.setTextColor(Color.parseColor( colorString: "#616161"));
                    }
                }
```

Figura 30. Chiamata al server per ottenere altri 16 parcheggi

3.7 Mappa nella schermata di dettaglio

La mappa nella schermata di dettaglio è realizzata attraverso un fragment. Per indicare il punto in cui è stato effettuato il parcheggio viene posizionato sulla mappa, prendendo con i metodi getParklon e getParkLat le coordinate geografiche, un custom marker.
La mappa presenta lo stesso stile di quella presente nella home ed è statica, quindi è stato impostato lo zoom su di essa in maniera tale da rendere visibile in quasi tutte le situazioni il nome delle strade e dei luoghi di riferimento principali presenti nelle vicinanze del parcheggio.

```
public void onMapReady(GoogleMap googleMap) {
    GoogleMap parkDetailsMap;
    parkDetailsMap = googleMap;
    int heightMarker = 180;
    int widthMarker = 180;
    BitmapDrawable bitmapdraw = (BitmapDrawable) getResources().getDrawable(R.drawable.placeholder);
    Bitmap bitmap = bitmapdraw.getBitmap();
    Bitmap smallMarker = Bitmap.createScaledBitmap(bitmap, widthMarker, heightMarker, filter: false);
    ParkDetailsActivity parkDetailsActivity = (ParkDetailsActivity) getActivity();
    parkDetailsMap.setMapStyle(new MapStyleOptions("[ { "elementType": "geometry", "stylers": [ { "color": ..."));
    LatLng parkLocation = new LatLng(Double.valueOf(parkDetailsActivity.getParkLat()),
            Double.valueOf(parkDetailsActivity.getParkLon()));
    parkDetailsMap.addMarker(new MarkerOptions().position(parkLocation)
            .icon(BitmapDescriptorFactory.fromBitmap(smallMarker)));
    parkDetailsMap.moveCamera(CameraUpdateFactory.newLatLng(parkLocation));
    parkDetailsMap.getUiSettings().setMapToolbarEnabled(false);
    parkDetailsMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
        @Override
        public void onMapClick(LatLng latLng) { parkMapView.setClickable(false); }
    });
}
```

Figura 31. Caratteristiche della mappa

Capitolo 4

Conclusione

4.1 Sommario

In questa relazione è stato innanzitutto presentato e analizzato il problema della ricerca del parcheggio, che ha un forte impatto sulla vita dei cittadini poiché fonte di stress, inquinamento e inutili sprechi di carburante. È stata anche presentata una possibile soluzione a questo problema: lo smart parking che utilizzato in maniera massiccia condurrebbe a ridurre tanti di questi disagi. In questo panorama è stata presentata l'app GeneroCity, le sue funzionalità e gli obiettivi che si pone di realizzare.

In seguito è stato descritto il lavoro svolto sull'applicazione GeneroCity Android riguardante la progettazione e lo sviluppo della visualizzazione dei parcheggi. Lo scopo di questo lavoro è stato fornire agli utenti delle schermate informative che permettessero il completo controllo degli spostamenti e dei parcheggi effettuati da loro o dai propri familiari.

Nel terzo capitolo è stata descritta l'implementazione, entrando nel dettaglio del codice e mostrandone i frammenti più importanti, come quello che consente di ottenere la lista dei parcheggi effettuati dal server o quello che consente di impaginarli in una lista avente determinate caratteristiche.

4.2 Sviluppi futuri

Nell'applicazione GeneroCity Android sono già state implementate tutte le funzionalità di base ma per rendere il suo utilizzo ancora più piacevole e interessante si potrebbe pensare di introdurre alcune integrazioni per rendere alcune sezioni dell'applicazione più complete.

In particolare le informazioni presenti nello storico delle soste potrebbero essere utilizzate per creare delle statistiche. Esse potrebbero includere informazioni riguardo il numero di persone aiutate a trovare parcheggio o il numero di parcheggi ottenuti facilmente con l'uso dell'app.

Avere delle statistiche dettagliate per ogni utente potrebbe anche aprire le porte ad alcune strategie di Gamification, come il raggiungimento di obiettivi o il completamento di missioni, che possano incentivare l'uso dell'applicazione.

Le statistiche potrebbero anche mostrare altri dati interessanti come il tempo risparmiato nella ricerca dei parcheggi, la quantità di carburante risparmiato e altre informazioni su quanto si è contribuito a migliorare l'ambiente, evitando di rilasciare CO2 nell'aria alla ricerca di un posto.

Bibliografia

[1] Inrix, Inrix 2019 Global Traffic Scoreboard. URL: https://inrix.com/scorecard/

[2] Smart parking systems, Il parcheggio: da problema a risorsa, 2018. URL: https://smartparkingsystems.com/il-parcheggio-da-problema-a-risorsa/

[3] ISTAT, Annuario statistico italiano, 2019. URL: https://www.istat.it/it/files/2019/12/Asi-2019.pdf

[4] Health Effect Institute (HEI), Institute for Health Metrics and Evaluation (IHME), University of British Columbia, State Of Global Air 2020. URL: https://www.stateofglobalair.org/

[5] MSR Traffic, Intelligente Parkleitsysteme von MSR-Traffic auf dem Campus der TU Kaiserslautern. URL: https://www.msr-traffic.de/2020/09/intelligente-parkleitsysteme-von-msr traffic-auf-dem-campus-der-tu-kaiserslautern/