SAPIENZA
UNIVERSITÀ DI ROMA

Progettazione della funzionalità “Lascia posto” nell’applicazione Android GeneroCity

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

Omar Bayoumi
Matricola 1747042
[redacted]

Relatore
Emanuele Panizzi
[redacted]

A.A. 2019-2020

Introduzione

GeneroCity è un'applicazione mobile il cui scopo principale è quello di permettere agli automobilisti di cercare e scambiare un parcheggio in modo immediato.

L'argomento del tirocinio che ho svolto, si è basato sullo sviluppo della funzionalità "Lascia Posto". Tale funzione è stata sviluppata in parallelo ad un'altra ("Cerca Posto") che è stata svolta in collaborazione con un collega. Si è reso necessario questo procedimento dal momento che, le due funzioni, sono interdipendenti. Per questo motivo verranno mostrate successivamente parti di codice e interfacce relative sia a "Lascia posto" che "Cerca posto".
La prima fase del tirocinio si è basata inizialmente con il risolvere alcuni semplici bug presenti nell'applicazione, la pulizia del codice, modificandolo e seguendo lo stile standard Google di Java [1]. Successivamente, ho realizzato un elemento nell'interfaccia relativa alla creazione di un'auto, in particolare il tasto per scegliere il colore della stessa. Dopo questa prima fase, mi sono occupato di ricreare, all'interno del FrameWork dell'applicazione, il database che gestisce le auto create, eliminate o modificate.

La seguente relazione è stata suddivisa in quattro capitoli. Nel primo capitolo ci sarà una vista generale dell'applicazione mostrando funzionalità e problematiche che tenta di risolvere. Nel secondo, invece, verranno mostrati gli obiettivi del tirocinio con le relative funzionalità da implementare. Nel terzo si scenderà nel dettaglio mostrando le parti di codice usate per implementare la funzionalità "Lascia posto" e "Cerca posto": spiegando le classi e i metodi. Infine, si analizzeranno gli sviluppi futuri che riguarderanno questa due feature.

Capitolo 1

GeneroCity

1.1 App GeneroCity

Figura 1: logo GeneroCity

GeneroCity è un'applicazione mobile sviluppata per sistemi Android e iOS dal Gamification Lab del Dipartimento di Informatica dell'ateneo La Sapienza di Roma con lo scopo principale di permettere agli utenti di ricercare in modo semplice un parcheggio.

> GeneroCity mette in contatto chi cerca un posto auto con chi sta per lasciarlo. L'app è basata sulla gamification e permette agli iscritti di scambiarsi informazioni sui posti auto liberi per risparmiare tempo nella ricerca del parcheggio. Lo scambio viene stimolato tramite il "pagamento dell'informazione" sui posti che si liberano, mediante punti. Gli iscritti a GeneroCity potranno sapere in anticipo dove e quando si libererà un posto e occuparlo. Quando a loro volta libereranno il posto, potranno generosamente utilizzare l'app per indicarlo. La generosità degli utenti che liberano un posto verrà stimolata facendo uso della gamification. Infatti, per poter sapere dove si libera un posto, sarà necessario disporre di punti, che si

> ottengono informando la community quando si lascia il proprio posto. Un'oculata strategia di gestione dei punti stimolerà a "lasciare" più posti auto di quanti se ne cerchino (mediante l'app). Da qui il nome dell'applicazione, che cerca di incoraggiare una filosofia del dare. Oltre al servizio principale sopra descritto, verranno oerti altri servizi correlati [. . .]. [2]

Il problema di cercare un posto in Francia, secondo uno studio fatto da Le Fauconnier e Gantelet, farebbe perdere mediamente 70 milioni di ore e circa 700 milioni di euro [3].

1.1.1 Lascia un parcheggio

Un utente può decidere di lasciare un parcheggio diventando Giver. Come si diventa Giver? L'utente che ha un'auto parcheggiata deve cliccare sul tasto "Lascia posto" per poter successivamente scegliere l'orario in cui vuole lasciare il posto. Una volta scelto l'orario questo utente diventerà Giver ed avrà dichiarato di voler lasciare un parcheggio.

1.1.2 Cerca un parcheggio

Un utente può decidere di cercare un parcheggio diventando Taker. Per diventare Taker, l'utente deve cliccare sul tasto "Cerca posto" per poter accedere alla schermata in cui può scegliere il luogo in cui cercare. Scelto il luogo, potrà scegliere un orario o dichiarare di cercare non appena arriva nel luogo scelto.

1.1.3 Effettua lo scambio di un parcheggio

Lo scambio di parcheggio avviene tra Taker e Giver che hanno dichiarato rispettivamente di cercare e lasciare il posto nella stessa zona e allo stesso orario creando così un Match[1]. Il Giver dovrà aspettare il Taker che arrivi prima di lasciare posto, altrimenti non riceverà i suoi GeneroCoins.

Il Taker, invece, dovrà andare verso il Giver ed infine, quando arriverà al parcheggio, premere sul tasto "Sono qui" per concludere il Match e permettere al Giver di andarsene. Dopo questa procedura al Taker verranno sottratti dei GeneroCoins. Questo significa che la procedura per cercare un posto è possibile solo se si hanno un numero sufficiente di GeneroCoins.

Sia il Taker che il Giver potranno decidere di annullare questa procedura in qualsiasi momento.

[1] *Match: Evento in cui un Giver e un Taker decidono di scambiare un posto per la stessa ora e posto.*

1.1.4 Gestione delle automobili

Nella schermata principale, l'utente ha la possibilità di creare delle auto da utilizzare nell'applicazione. All'utente gli verrà richiesto come prima cosa di inserire la targa oppure di utilizzare uno dei suoi dispositivi bluetooth associati (Figura 2), per poi inserire il resto delle informazioni (Figura 3):

- nome dell'auto;
- marca;
- modello;
- targa;
- dimensione;
- colore.

All'utente gli sarà poi permesso, con apposito tasto nella schermata principale, di modificare questi dati in qualsiasi momento. Inoltre, avrà la possibilità di eliminare un'auto.

Figura 2: schermata per inserire la targa o usare un dispositivo bluetooth

Figura 3: schermata per inserire i dati di un'auto

Capitolo 2

Integrazione della funzionalità “Lascia posto”

2.1 La funzionalità “Lascia posto”

La funzionalità “Lascia posto” è il fulcro principale dell’applicazione insieme alla controparte “Cerca posto”. Questa funzionalità permette a un utente di lasciare il posto in cui ha parcheggiato la propria auto, in un determinato orario ad un altro utente al fine di guadagnare GeneroCoins necessari per cercare un posto in futuro.

2.1.1 Come iniziare la procedura per lasciare il parcheggio

Per poter lasciare il parcheggio bisogna cliccare sul tasto “Lascia posto” nella card relativa all’auto (Figura 4). Però, prima di poter lasciare il parcheggio, l’applicazione si assicura che l’auto sia effettivamente parcheggiata rendendo non cliccabile il tasto in caso non lo fosse (Figura 5).

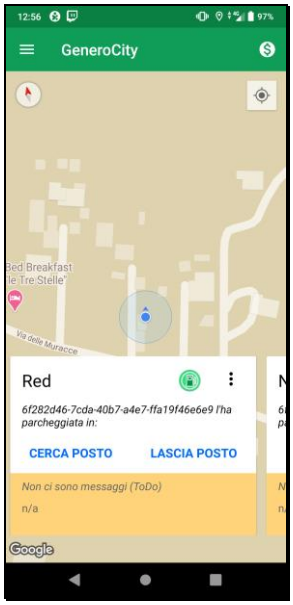

Figura 4: card in caso di auto parcheggiata

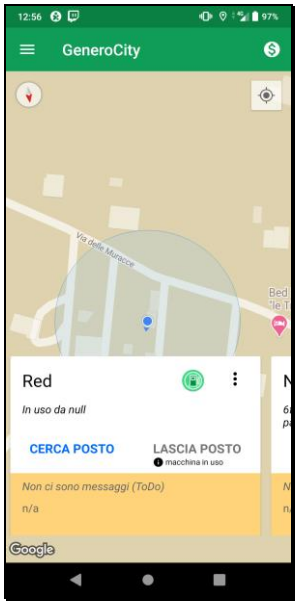

Figura 5: card in caso di auto non parcheggiata

2.1.2 La scelta dell’orario

Dopo aver cliccato sul tasto “Lascia posto” verrà mostrata un’interfaccia composta da un primo tasto in alto che ti permette di dichiarare di voler lasciare un posto adesso, ovvero al primo schedule[2] disponibile e subito sotto un elenco di schedule disponibili calcolati dall’orario corrente (Figura 6). Affianco a uno schedule potrebbero comparire i nomi di altri utenti. Questo sta a significare che per quel determinato schedule un altro utente ha dichiarato di cercare un posto nella stessa zona in cui è stata parcheggiata l’auto. Questo dà la possibilità all’utente di scegliere uno schedule in cui saprà già che avverrà un Match, riducendo così i tempi di attesa alla ricerca di un Taker.

Scelto lo schedule in cui lasciare, verrà fatta una chiamata all’API (Application Programming Interface: unità di elaborazione che definisce le interazioni tra più intermediari software [4]) createGiver trasformando l’utente in un Giver dando luogo a tutto il meccanismo dei Match (2.2).

Figura 6: schermata per la scelta di uno schedule

[2] *Schedule: Orario in cui si vuole lasciare o cercare un posto diviso a scaglioni di 15 minuti.*

2.2 Il meccanismo dei Match

Un Match tra un Giver e un Taker è quell’evento che permette ai due di scambiarsi di posto nel luogo e nell’ora scelti.

Questo meccanismo è diviso in vari stati: “waiting” (2.2.1); “running” (2.2.2); “unsuccess-giver” (2.2.3); “unsuccess-taker” (2.2.4); “expired” (2.2.5); “success” (2.2.6).

2.2.1 Lo stato di “waiting”

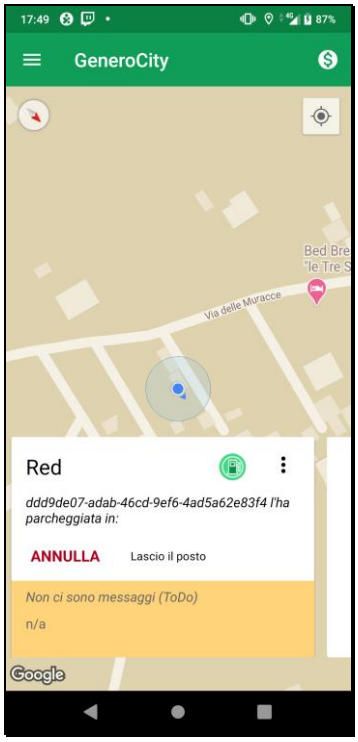

Figura 7: lsacia il posto adesso

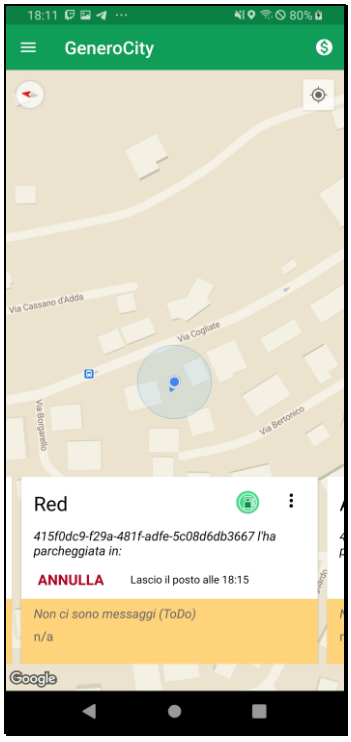

Figura 8: lascia il posto a un orario specifico

Lo stato di “waiting” si ottiene diventando Giver (o Taker) e si è in attesa di trovare un match. Nella Figura 7 è mostrata la card che appare quando si clicca sullo schedule “Adesso” mentre nella Figura 8 quando si clicca su uno schedule specifico (in questo caso alle 18:15). La card contiene un tasto annulla, che annullerà immediatamente la ricerca di un Match attraverso la chiama API deleteGiver, e una descrizione sul quando si vuole lasciare il posto.

2.2.2 Lo stato di “running”

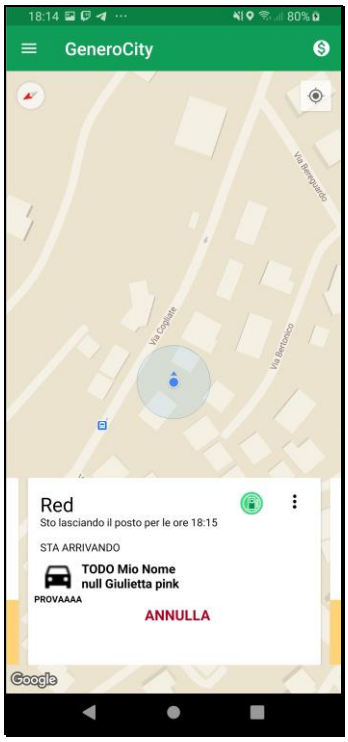

Figura 9: card nello stato di "running"

Lo stato di “running” è uno stato in cui il Match tra Giver e Taker è stato trovato e si è in attesa dello scambio nell’ora prefissata. Nella Figura 9 è mostrata la card relativa a questo stato. È presente un’etichetta in alto che mostra l’orario di quando si vuole lasciare il posto e al di sotto le informazioni dell’auto con cui si deve scambiare ottenute con la chiamata API getCarByCid, come: nome dell’auto; nome del proprietario; modello; targa e colore. Il tasto annulla compie la stessa azione spiegata nella sezione precedente (2.2.1) con la differenza che porta il Taker nello stato di “unsuccess-giver” (2.2.3) poiché non ha più un Giver con cui scambiarsi.

2.2.3 Lo stato di “unsuccess-giver”

Lo stato di “unsuccess-giver” è uno stato del Taker che si ottiene quando il Giver annulla un Match trovato. In questo stato il Taker effettua la chiamata API removeDriverHistory per eliminare questo Match dalla sua lista dei Match. Quindi viene lanciata, se lo schedule non è passato rispetto all’orario attuale, una nuova chiamata API di createTaker per tornare di nuovo nello stato di “waiting” con lo stesso schedule e zona scelta.

2.2.4 Lo stato di “unsuccess-taker”

Lo stato di “unsuccess-taker” si ottiene se dallo stato di “running”, il Taker con cui c’è stato il Match clicca sul tasto annulla chiamando l’API deleteTaker. Per il Giver, analogamente allo stato di “unsuccess-taker” (2.2.3), verrà eliminato questo Match dalla sua lista dei Match con la chiamata API removeDriverHistory. Successivamente, se lo schedule non è passato rispetto all’orario attuale, l’utente ritornerà Giver con una nuova chiamata API di createGiver, tornando nello stato di “waiting” (2.2.1) con lo stesso schedule.

2.2.5 Lo stato “expired”

Lo stato “expired” si verifica se il Giver (stessa cosa per il Taker) si trova in “waiting” (2.2.1) o “running” (2.2.2) da più di 15 minuti. In questo stato, viene fatta una chiamata API a removeDriverHistory per eliminare questo Match dalla sua lista dei Match per poi tornare alla schermata di default (Figura 4, Figura 5).

2.2.6 Lo stato di "success"

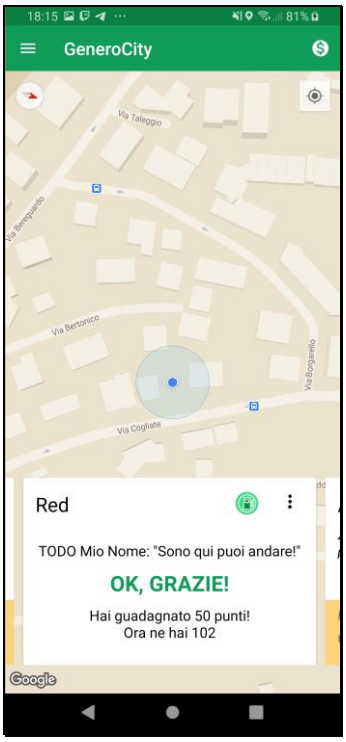

Figura 10: card nello stato di "success"

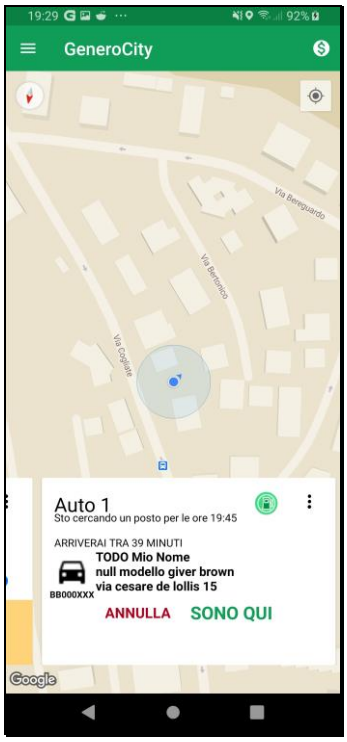

Figura 11: card nello stato di "running" per il Taker

Lo stato di "success" è lo stato che si ottiene nella fase finale del match, quando tutto va a buon fine: il Giver lascia il proprio posto al Taker che lo occupa. Per arrivare in questo stato, il Taker deve confermare di essere arrivato con il tasto "sono qui" (presente in Figura 11) chiamando l'API completeMatch. In Figura 10 è illustrata la schermata di "success" per il Giver con i seguenti elementi:

- Un messaggio del Taker che avverte il Giver che può andar via;
- Un tasto "ok, grazie!" che esegue la chiamata API removeDriverHistory per eliminare il Match dalla sua lista dei Match. Successivamente, verrà mostrata la schermata di default (Figura 4, Figura 5);
- In basso è presente la quantità dei GeneroCoins e il suo totale (Figura 10). Tale informazione è ottenuta tramite l'utilizzo della chiamata API getMeGcoins.

Capitolo 3

Implementazione

3.1 Ambiente di sviluppo

L'applicazione GeneroCity per Android è sviluppata usando Java, un linguaggio orientato agli oggetti [5], per il codice e XML nella versione 1.0, un metalinguaggio per la definizione di linguaggi markup [6], nell'ambiente di sviluppo di Android Studio [7] con un livello di API massimo supportato pari a 21.

3.2 Le classi sviluppate

Per realizzare ciò che è stato detto nel capitolo precedente (2) e in concomitanza alle argomentazioni dette nell'introduzione (intro.), è stata necessaria la modifica di alcune classi già esistenti e aggiungerne di nuove, relative sia a "Cerca posto" che "Lascia posto". Inoltre, sono state implementate quattro nuove interfacce mentre una è stata modificata, il tutto seguendo le direttive dettate nel material design di Android [8].

3.2.1 Le classi aggiunte

Sono state aggiunte in tutto sei nuove classi di cui quattro adattate sia per il "Cerca Posto" che per il "Lascia posto" e due solo per il "Cerca posto".

3.2.1.1 LeaveAndSearchParkScheduleFragment

La classe LeaveAndSearchParkScheduleFragment implementa il codice relativo all'interfaccia della selezione degli schedule detta in precedenza (Figura 6). Gli attributi di interesse di questa classe sono:

- giver: un attributo boolean che indica se è stato un fatto accesso a questa schermata da un potenziale Giver (true) o un potenziale Taker (false);
- location: una stringa che indica il luogo in cui il Taker vuole cercare un posto;
- car: un oggetto di tipo Car che corrisponde all'auto con la quale si vuole cercare o lasciare il posto;
- scheduleList: è una ListView che permette di rappresentare tutti gli schedule, con affianco un elenco di Taker o Giver disponibili per ogni schedule (metodo seScheduleListAdapter()), in una vista scorrevole (Figura 13);
- hourNow e minuteNow: che indicano ora e minuti attuali nel momento in cui viene aperto questo Fragment normalizzati al primo schedule (Figura 12);
- today: è un boolean. Il suo utilizzo principale è quello di indicare se il Fragment è stato aperto a partire dalle 23:45 in modo da permettere la visualizzazione degli schedule del giorno dopo dalle 00:00 alle 23:45 (Figura 12).

```
if (hourNow == 23 && minuteNow >= 45) {
    hourNow = 0;
    minuteNow = 0;
    today = false;
} else {
    minuteNow = (minuteNow / 15) * 15 + 15;
    if (minuteNow == 60) {
        minuteNow = 0;
        hourNow++;
    }
}
```

Figura 12: inizializzazione di minuteNow, hourNow e today

```
if (isGiver()) {
    title.setText("Quando vuoi lasciare?");
    nowButton.setText("Adesso");
    if (!today) {...} else {...}
    app.api().getTakerList(car.getCid().toString(), GCConstants.DELTA_LAT_LONG,
            GCConstants.DELTA_LAT_LONG).enqueue(new Callback<List<Giver>>() {
        @Override
        public void onResponse(@NonNull Call<List<Giver>> call, @NonNull Response<List<Giver>> response) {
            setScheduleListAdapter(response, scheduleArray, rootView);
        }

        @Override
        public void onFailure(@NonNull Call<List<Giver>> call, @NonNull Throwable t) {...}
    });
} else {
    title.setText("Quando vuoi cercare?");
    nowButton.setText("Appena arrivo in zona");
    String nowButtonString = nowButton.getText().toString() + " " + location;
    nowButton.setText(nowButtonString);
    if (!today) {...} else {...}
    RecentListUtils recentListUtils = new RecentListUtils(getContext());
    app.api().getGiverList(car.getCid().toString(), (float) recentListUtils.getValue(location).latitude,
            (float) recentListUtils.getValue(location).longitude, GCConstants.DELTA_LAT_LONG,
            GCConstants.DELTA_LAT_LONG).enqueue(new Callback<List<Giver>>() {
        public void onResponse(@NonNull Call<List<Giver>> call, @NonNull Response<List<Giver>> response) {
            setScheduleListAdapter(response, scheduleArray, rootView);
        }

        @Override
        public void onFailure(@NonNull Call<List<Giver>> call, @NonNull Throwable t) {...}
    });
}
```

Figura 13: inizializzazione elenco di Giver o Taker affianco agli schedule

La classe disponde di due metodi:

- onCreateView(): metodo che inizializza tutte le variabili e gli elementi grafici di interfaccia per poterne programmare il contenuto anche in base al valore dell'attributo giver. Inoltre, vendono anche inizializzati tutti i clickListener dei tasti:
  - Il listener del tasto "Adesso" (riferimento sempre a Figura 6), qualora fosse utilizzato da un Giver, esegue la chiamata API createGiver passando come valori di deltaLat e deltaLon, una costante definita nella classe GCConstants (3.2.2.4) e come valore di schedule il primo schedule disponibile dato dal metodo statico firstSchedule() di DateTime (3.2.2.3). All'interno della risposta vengono gestiti i valori da inserire nella descrizione della card in stato di "waiting" con l'uilizzo della classe SchedulePositionUtils (3.2.1.6);
  - Il listener del tasto back sulla toolbar per tornare indietro alla schermata precedente.
- setScheduleListAdapter(): questo metodo è chiamato sia dal Giver che dal Taker. Il suo scopo è quello di impostare l'elenco dei Taker o Giver diponibili per ogni schedule sulla base della risposta API di getTakerList o getGiverList. Ogni elemento schedule-elencoGiver/Taker è un oggetto del tipo Schedule Item (3.2.1.2). Dopo aver impostato la lista di SchduleItem correttamente, viene inizializzato uno ScheduleListAdapter passandogli questo array (3.2.1.5).

```
private void setScheduleListAdapter(Response<List<Giver>> response,
                                    List<ScheduleItem> scheduleArray, View rootView) {
    if (response.body() != null) {
        List<Giver> takerOrGiverList = response.body();
        for (Giver takerOrGiver : takerOrGiverList) {
            DateTime schedule = new DateTime(takerOrGiver.getSchedule());
            String userNickname = takerOrGiver.getUsernickname();
            int scheduleHour = schedule.getHour();
            int scheduleMinute = schedule.getMinute();
            int scheduleItemIndex = 4 * (scheduleHour - hourNow) + (scheduleMinute - minuteNow) / 15;
            if (scheduleItemIndex >= 0) {
                scheduleArray.get(scheduleItemIndex).setUsedBy(userNickname);
            }
        }
    }
    ScheduleListAdapter adapter = new ScheduleListAdapter(getContext(), R.layout.list_schedule_item,
            scheduleArray, rootView, today, giver, app, car, getFragmentManager(),
             leaveAndSearchParkScheduleFragment: this, location);
    scheduleList.setAdapter(adapter);
}
```

Figura 14: inizializzazione dello ScheduleListAdapter per la visualizzazione degli schedule

3.2.1.2 ScheduleItem

La classe ScheduleItem (Figura 15) è una classe che definisce il formato del singolo schedule (Figura 16). Ogni schedule avrà un ora nel formato "hh-mm" (dove hh indica due cifre per l'ora e mm due cifre per i minuti) seguito da un elenco di Taker (se sei Giver) o Giver (se sei Taker) impostati dal metodo setUsedBy() che altro non fa che concatenare un Giver/Taker separandoli con ", ".

```
public class ScheduleItem {
    String schedule;
    String usedBy;

    public ScheduleItem(String schedule, String usedBy) {
        this.schedule = schedule;
        this.usedBy = usedBy;
    }

    public ScheduleItem(String schedule) { this(schedule, usedBy: ""); }

    public String getSchedule() { return schedule; }

    public String getUsedBy() { return usedBy; }

    public void setUsedBy(String usedBy) {
        if ("".equals(this.usedBy)) {
            this.usedBy = usedBy;
        } else {
            this.usedBy = this.usedBy + ", " + usedBy;
        }
    }
}
```

Figura 15: classe ScheduleItem

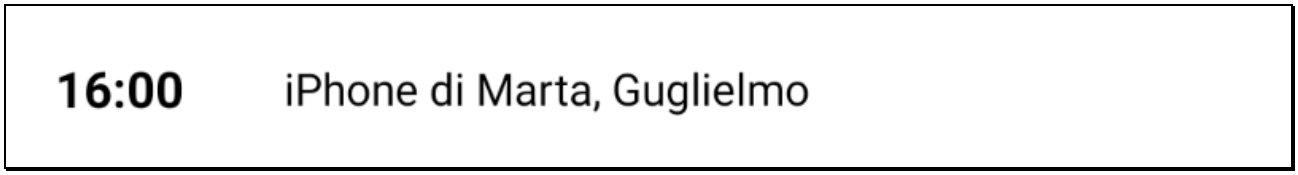

Figura 16: layout dello schedule

2.2.1.3 CompleteMatchTakerUtils

La classe CompleteMatchTakerUtils è una classe che gestisce un booleano attraverso uno sharedPreferences. Lo sharedPreferences è costituito dalla coppia (cid (car id), true/false). Il valore assegnato dalla key cid è true qualora l'auto a cui corrisponde questa chiave è un Taker che ha appena completato un match con la chiamata API completeMatch. Invece, se è false, quel cid non è presente all'interno dello sharedPreferences. Questa classe viene utilizzata principalmente per la visualizzazione della card nello stato di "success" per il Taker.

```
public class CompleteMatchTakerUtils {

    SharedPreferences sharedPreferences;

    public CompleteMatchTakerUtils(Context context) {
        sharedPreferences = context.getSharedPreferences( s: "completeMatchTaker", Context.MODE_PRIVATE);
    }

    public void setCompleteMatchForTakerCar(String key) {
        sharedPreferences.edit().putBoolean(key,  b: true).apply();
    }

    public boolean isCompleteMatchForTakerCar(String key) {
        return sharedPreferences.getBoolean(key,  b: false);
    }

    public void removeCompleteMatchForTakerCar(String key) {
        sharedPreferences.edit().remove(key).apply();
    }
}
```

Figura 17: classe CompleteTakerUtils

2.2.1.4 RecentListUtils

Figura 18: lista dei recenti

La classe RecentListUtils si occupa di mostrare le ultime dieci ricerche fatte da un utente in ordine cronologico (Figura 18). Per fare ciò utilizza uno sharedPreferences che salva gli ultimi dieci inserimenti fatti nell'editText di ricerca. Lo sharedPreferences è definito con una coppia (posizione scelta, coordinate della posizione). I metodi di questa classe sono:

- setSearchedPosition() (Figura 19): se lo sharedPreferences non ha superato i dieci elementi, allora viene inserito come valore: latitudine + "," + longitudine; come chiave: findGreaterIndex() + posizione. Se invece ha superato i dieci elementi viene eseguito il metodo updateIndex();

```
public void setSearchedPosition(String position, double latitude, double longitude) {
    if (sharedPreferences.getAll().keySet().size() < 9) {
        String greaterIndex = findGreaterIndex();
        sharedPreferences.edit().putString( s: greaterIndex + position, s1: latitude + "," + longitude)
                .apply();
    } else {
        updateIndex("1" + position, latitude + "," + longitude);
    }
}
```

Figura 19: metodo setSearchedPosition()

- getKeys() (Figura 20): restituisce la lista delle chiavi ordinata eliminando l'indice posto all'inizio della stringa per una visualizzazione pulita;
- orderList() (Figura 20): questo metodo ordina e restituisce la lista delle chiavi. L'ordinamento è stato fatto attraverso un selection sort perché il livello di api massimo richiesto era insufficiente per supportare il metodo di List, sort(). È stato scelto il selection sort per semplicità di implementazione e perché comunque anche se ha un costo di $O(n^2)$, n non sarà mai più grande di 10 eseguendolo così in modo istantaneo;

```
public List<String> getKeys() {
    List<String> list = orderList();
    List<String> listFinal = new ArrayList<>();
    for (String key : list) {
        listFinal.add(key.substring(1));
    }
    return listFinal;
}

public List<String> orderList() {
    String[] keys = Arrays.copyOf(sharedPreferences.getAll().keySet().toArray(),
            sharedPreferences.getAll().keySet().toArray().length, String[].class);
    for (int i = 0; i < keys.length - 1; i++) {
        int maxValue = Integer.parseInt(keys[i].substring(0, 1));
        int maxIndex = i;
        for (int j = i + 1; j < keys.length; j++) {
            if (maxValue < Integer.parseInt(keys[j].substring(0, 1))) {
                maxValue = Integer.parseInt(keys[j].substring(0, 1));
                maxIndex = j;
            }
        }
        String temp = keys[maxIndex];
        keys[maxIndex] = keys[i];
        keys[i] = temp;
    }
    return new ArrayList<>(Arrays.asList(keys));
}
```

Figura 20: metodi getKeys() e orderList()

- updateIndex() (Figura 21): questo metodo inserisce un nuovo elemento (chiave, valore) all'interno dello sharedPreferences con l'accortezza di eliminare l'elemento più vecchio. L'elemento più vecchio è identificato dal numero più alto posto all'inizio di ogni chiave;

```
public void updateIndex(String key, String value) {
    List<String> list = orderList();
    list.add( k 0, key);
    String removed = list.remove( k list.size() - 1);
    sharedPreferences.edit().remove(removed).apply();
    sharedPreferences.edit().putString(key, value).apply();
    List<String> values = new ArrayList<>(list);
    for (String k : sharedPreferences.getAll().keySet()) {
        values.remove(list.indexOf(k));
        values.add(list.indexOf(k), sharedPreferences.getAll().get(k).toString());
        sharedPreferences.edit().remove(k).apply();
    }
    List<String> updateIndexList = new ArrayList<>();
    for (int i = 1; i <= list.size(); i++) {
        int index = list.size() - i + 1;
        updateIndexList.add(index + list.get(i - 1).substring(1));
    }
    for (int i = 0; i < updateIndexList.size(); i++) {
        sharedPreferences.edit().putString(updateIndexList.get(i), values.get(i)).apply();
    }
}
```

Figura 21: metodo updateIndex()

- getValue(): restituisce il valore associato a una chiave;
- findGreaterIndex(): restituisce l'indice successivo a quello più vecchio. Se non sono presenti elementi, allora restituisce 0;

3.2.1.5 ScheduleListAdapter

La classe ScheduleListAdapter è la classe che riempie il contenuto della ListView descritta nella classe LeaveAndSearchParkScheduleFragment (3.2.1.1) con dei ScheduleItem (3.2.1.2). Per ognuno di questi item (Figura 16), viene impostato un clickListener che si occupa di trasformare l'utente in un Giver (o un Taker) per lo schedule selezionato. In particolare, se l'attributo isGiver è true, verrà fatta la chiamata API di createGiver e nella risposta verrà impostato il valore che poi sarà mostrato all'interno della card in stato di "waiting" attraverso l'utilizzo della classe ShedulePositionUtils (3.2.1.6). Successivamente, viene fatto un update delle cards con l'utilizzo del metodo statico CarUtils.CarListUpdated() e infine chiudere il fragment e mostrare le cards. La classe, inoltre, ha un metodo chiamato checkSchedule() che si occupa di verificare che lo chedule scelto non sia passato rispetto all'orario attuale. Esempio: sono le 13:43 e voglio diventare un Giver per le 13:45. Per qualche motivo ritardo il click alle 13:46. Verrà mostrato uno SnackBar che avvisa l'utente che lo schedule è passato.

```
if (isGiver) {
    app.api().createGiver(car.getCid().toString(), new Position(
            GCConstants.DELTA_LAT_LONG, GCConstants.DELTA_LAT_LONG, schedule))
            .enqueue(new Callback<String>() {
        @Override
        public void onResponse(@NonNull Call<String> call, @NonNull Response<String> response) {
            Log.e( tag: "ScheduleListAdapter",  msg: "Create giver onResponse " + response.code());
            String article;
            if (currentDateTime.getHour() == 1) {
                article = "all'";
            } else {
                article = String.format(Locale.getDefault(),  format: "%s ", "alle");
            }
            String firstPart = String.format(Locale.getDefault(),  format: "%s " + article,
                    "Lascio il posto");
            String hour = String.format(Locale.getDefault(), format: "%02d", currentDateTime.newDateAddMinute(1).getHour());
            String minute = String.format(Locale.getDefault(), format: "%02d", currentDateTime.newDateAddMinute(1).getMinute());
            String sharedSchedule = hour + ":" + minute;
            new SchedulePositionUtils(getContext()).setSharedPosition( key: schedule + "~" + car.getCid().toString(),
                    firstPart, sharedSchedule,  secondPart: "",  position: "");

            //update graphics
            CarUtils.carListUpdated(app);

            fragmentManager.beginTransaction().remove(leaveAndSearchParkScheduleFragment).commit();
        }

        @Override
        public void onFailure(@NonNull Call<String> call, @NonNull Throwable t) {
            Log.e( tag: "ScheduleListAdapter",  msg: "Create giver onFailure");
        }
    });
```

Figura 22: API createGiver per un determinato schedule

3.2.1.6 SchedulePositionUtils

La classe SchedulePositionUtils utilizza uno sharedPreferences per salvare i valori da inserire nella card in stato di "waiting". In particolare, come chiave è stata utilizzata la stringa: schedule + "~" + cid; mentre per valore una serie di informazioni separate da virgola. La descrizione contenuta nella card segue è il risultato della concatenazione degli attributi della classe:

1. firstPart: descrive la prima parte della frase. I valori che assumerà, a seconda della situazione, saranno: "Lascio il posto"; "Cerco un posto appena arriverai in zona"; "Lascio il posto alle"; "Lascio il posto all'"; "Cerco un posto per l'"; "Cerco un posto per le".
2. schedule: rappresenta lo schedule scelto nel formato "hh:mm".
3. secondPart: descrive la seconda parte dellafrase. Il valore che assumerà, come per firstPart, dipende dalla situazione e saranno i valori: "in zona" oppure "";
4. location: rappresenta la zona scelta dal Taker.

Ognuno di questi attributi potrebbe aver valore "" e questo sta a indicare che quella parte non è necessaria ai fini della costruzione della frase. I metodi di questa classe sono:

- setSharedPosition() (Figura 23): il metodo prende in input una chiave e quattro stringhe. Quello che fa è inserire un nuovo elemento nello sharedPreferences, con chiave quella presa in input e con valore la concatenazione delle stringhe separate dal carattere "~". Se una delle stringhe prese in input è la stringa vuota, questa verra sostituita da una composta da un singolo spazio per permettere lo split.

```
public void setSharedPosition(String key, String firstPart, String schedule, String secondPart, String position) {
    DateTime currentDateTime = new DateTime(key.split( regex: "~")[0]);
    DateTime currentDateTimePlusOne = currentDateTime.newDateAddMinute(1);
    int currentHour = currentDateTimePlusOne.getHour();
    int currentMinute = currentDateTimePlusOne.getMinute();

    String scheduleKey = key.substring(0, 11)
            + String.format(Locale.getDefault(), format: "%02d", currentHour) + ":"
            + String.format(Locale.getDefault(), format: "%02d", currentMinute)
            + ":00"
            + key.substring(23);
    if ("".equals(firstPart)) {
        firstPart = " ";
    }
    if ("".equals(schedule)) {
        schedule = " ";
    }
    if ("".equals(secondPart)) {
        secondPart = " ";
    }
    if ("".equals(position)) {
        position = " ";
    }
    String value = firstPart + "~" + schedule + "~" + secondPart + "~" + position;
    sharedPreferences.edit().putString(scheduleKey, value).apply();
}
```

Figura 23: metodo setSharedPosition()

- setPositionUtilsAttribute() (Figura 23): il metodo prende in input una chiave tramite la quale verrà ricavato il valore dallo sharedPreferences. Seguendo lo schema descritto in Figura 23, sarà possibile accedere ai valori di: firstPart, schedule, secondPart e location facendo uno split con regex uguale a "~". Una volta presi i valori, questi saranno impostato come attributi della classe;

```
public void setPositionUtilsAttribute(String key) {
    String[] sharedPosition = sharedPreferences.getString(key, s1: " ~ ~ ~ ").split( regex: "~");
    firstPart = " ".equals(sharedPosition[0]) ? "" : sharedPosition[0];
    schedule = " ".equals(sharedPosition[1]) ? "" : sharedPosition[1];
    secondPart = " ".equals(sharedPosition[2]) ? "" : sharedPosition[2];
    location = " ".equals(sharedPosition[3]) ? "" : sharedPosition[3];
}
```

Figura 24: metodo setPositionUtilsAttribute()

- setPositionUtilsWhenArriveAtDestination() (Figura 25): questo metodo esegue le stesse cose di setPositionUtilsAttribute() con la differenza che la chiave in input è solo il cid dell'auto. Questo perché quando un utente dichiara di voler cercare appena arriva in zona, non è effettivamente un Taker ma comunque dovrà vedere la card relativa allo stato di "waiting". Quindi quello che fa il metodo è cercare all'interno dello sharedPreferences una corrispondenza tra chiave in input e la sottostringa corrispondente al cid. Una volta trovata, verranno impostati gli attributi come in setPositionUtilsAttribute();

```
public void setPositionUtilsWhenArriveAtDestination(String key) {
    for (String s : sharedPreferences.getAll().keySet()) {
        if (key.equals(s.split( regex: "~")[1])) {
            String[] sharedPosition = sharedPreferences.getString(s, s1: " ~ ~ ~ ").split( regex: "~");
            firstPart = " ".equals(sharedPosition[0]) ? "" : sharedPosition[0];
            schedule = " ".equals(sharedPosition[1]) ? "" : sharedPosition[1];
            secondPart = " ".equals(sharedPosition[2]) ? "" : sharedPosition[2];
            location = " ".equals(sharedPosition[3]) ? "" : sharedPosition[3];
            break;
        }
    }
}
```

Figura 25: metodo setPositionUtilsWhenArriveAtDestination()

- cleanSharedPreferences(): questo metodo prende in input una chiave ed elimina dallo sharedPreferences ogni sua occorrenza.

3.2.2 Le classi modificate

Per realizzare le funzionalità di "Lascia posto", "Cerca posto" è stato necessario modificare numerose classi già esistenti. Sono state modificate in tutto sei classi.

3.2.2.1 MainScreenCarSharingFragment

La classe MainScreenCarSharingFragment è una delle classi più importanti dell'applicazione che permette di mostrare le cards all'interno dell'interfaccia principale. La card associata a un'auto varia la sua struttura interna a seconda dello stato del Match in corso. Tuttavia, alcuni stati posseggono elementi di interfaccia comuni. Questo ha portato a una ristrutturazione completa del metodo updateCar(), con uno nuovo chiamato updateCardStandard(). Quest'ultimo contiene la struttura che precedentemente era all'interno del ciclo for di updateCar() che itera sulle auto possedute. All'interno di updateCardStandard() è stato suddiviso e gestito ogni elemento grafico in un metodo a parte cosi da poterne riutilizzare il contenuto. Il codice è strutturato con un ciclo for con al suo interno sei if in sequenza per determinare in quale stato ci troviamo (Figura 35):

I - if: (Figura 26 e Figura 27) se l'auto si trova in un Match con lo stato di "waiting" oppure un Taker ha dichiarato di cercare appena arriva in zona, il metodo isDestinationSetForCar() di WhenArriveAtUtils restituisce true e verrà mostrata la card relativa a tale stato con il metodo updateCardWaiting();

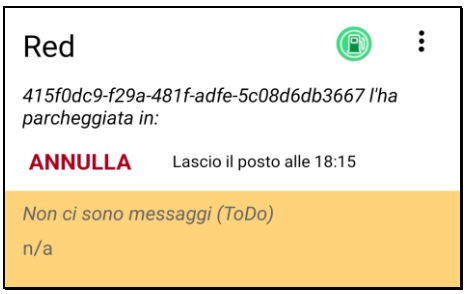

Figura 26: card "waiting" per il Giver

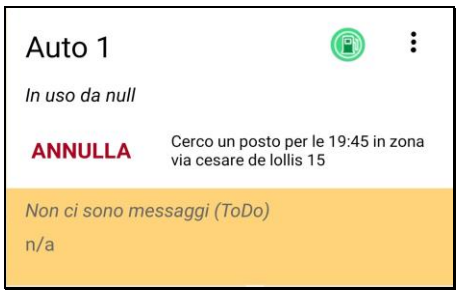

Figura 27: card "waiting" per il Taker

II - if: (Figura 28, Figura 29) se l'auto si trova in un Match con lo stato di "running", verrà mostrata la card relativa a tale stato con il metodo updateCardRunning;

Figura 28: card "running" per il Giver

Figura 29: card "running" per il Taker

III - if: (Figura 30, Figura 31) se l'auto si trova in un Match con lo stato di "success" oppure un Taker ha completato un match cliccando sul tasto "sono qui" dell'interfaccia "running" (Figura 29), il metodo isCompleteMatchForTakerCar() di CompleteMatchTakerUtils (2.2.1.3) restituisce true e verrà mostrata la card relativa a questo stato con il metodo updateCardSuccess();

Figura 30: card "success" per il Giver

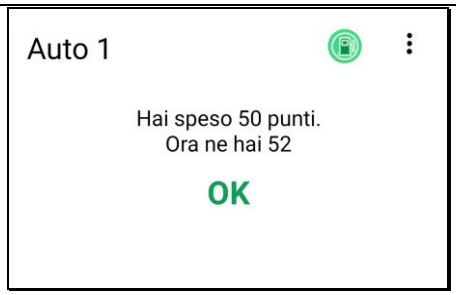

Figura 31: card "success" per il Taker

IV - if: (Figura 32) se l'auto si trova in un Match con lo stato di "unsuccess-giver", verrà rimosso tale Match dalla sua lista dei match con la chiamata API removeDriverHistory. Nella risposta di questa API, se lo schedule del Match rimosso non è passato rispetto all'orario attuale, l'utente torna Taker con stesso schedule e posizione utilizzando la chiamata API createTaker;

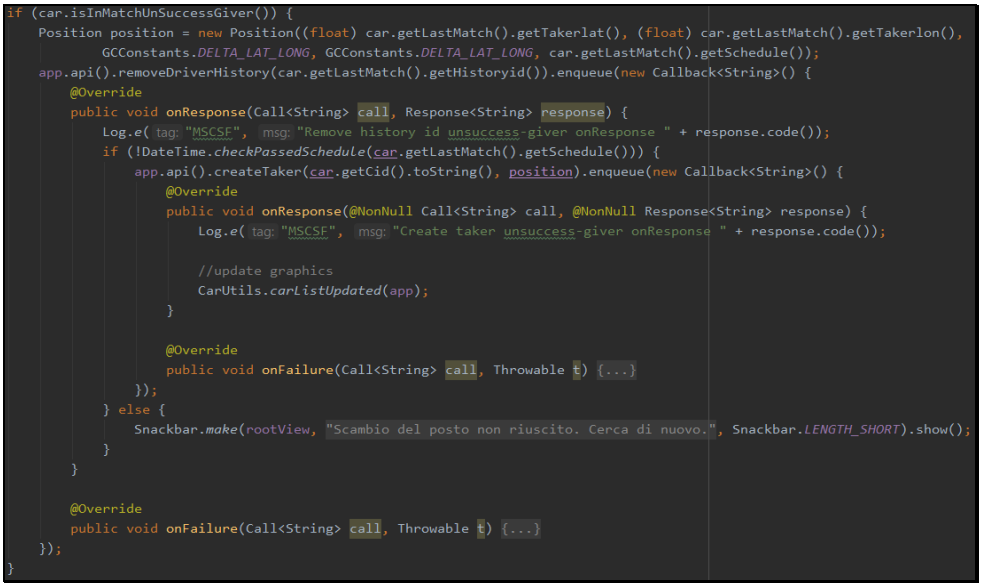

Figura 32: codice che gestisce l'"unsuccess-giver"

V - if: (Figura 33) se l'auto si trova in un Match con lo stato di "unsuccess-taker", accadranno le stesse cose descritte nel precedente if ma con il Taker al posto del Giver;

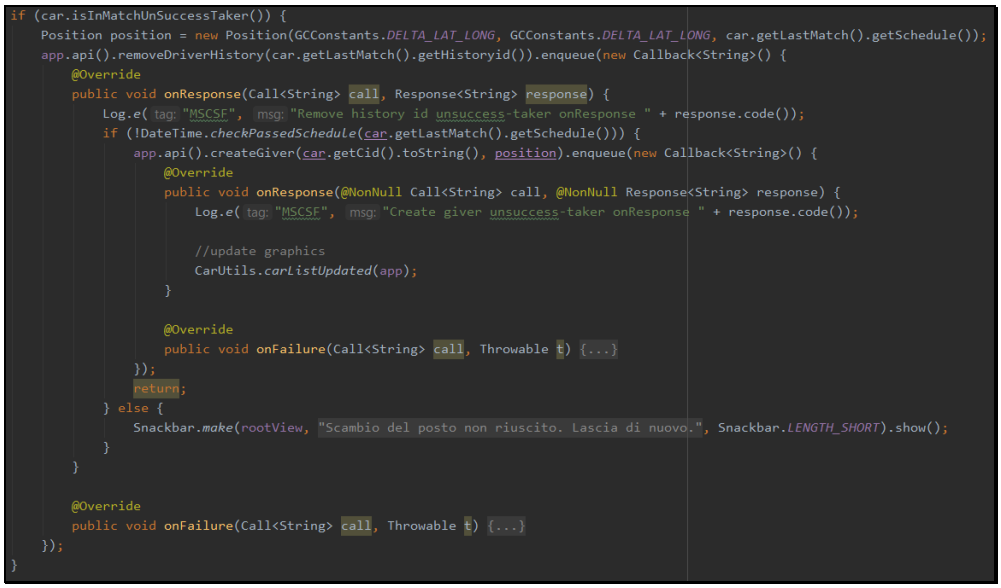

Figura 33: codice che gestisce l'"unsuccess-taker"

VI - if: se l'auto si trova in un Match con lo stato di "expired", allora questo Match verrà rimosso dalla sua lista dei Match con l'API removeDriverHistory.

Se ognuno di questi if fallisce, allora verrà mostrata la card standard con updateCardStandard() (Figura 34) e verrà eliminata ogni informazione necessaria alle card di "waiting" e "running" attraverso il metodo cleanSharedPreferences() di SchedulePositionUtils (3.2.1.6).

Figura 34: card standard

```
//for every car in local memory
for (final Car car : me.getAllCars()) {
    //updateCardSuccess(car, cardWidth, cardSpace);
    if (car.isInMatchWaiting() || new WhenArriveAtUtils(getContext())
            .isDestinationSetForCar(car.getCid().toString())) {
        updateCardWaiting(car, cardWidth, cardSpace);
        continue;
    }
    if (car.isInMatchRunning()) {
        updateCardRunning(car, cardWidth, cardSpace);
        continue;
    }
    if (car.isInMatchSuccess() || new CompleteMatchTakerUtils(getContext())
            .isCompleteMatchForTakerCar(car.getCid().toString())) {
        updateCardSuccess(car, cardWidth, cardSpace);
        continue;
    }

    if (car.isInMatchUnSuccessGiver()) {...}

    if (car.isInMatchUnSuccessTaker()) {...}


    if (car.isInMatchExpired()) {
        app.api().removeDriverHistory(car.getLastMatch().getHistoryid())
                .enqueue(new Callback<String>() {...});
    }
    new SchedulePositionUtils(getContext()).cleanSharedPreferences(car);
    updateCardStandard(car, cardWidth, cardSpace);
}
```

Figura 35: codice del ciclo for che gestisce gli stati delle card

3.2.2.2 Match

La classe Match è un tipo che contiene gli attributi di un Match. In questa classe sono stati inseriti dei getter() per alcuni attributi (Figura 36).

```
public String getStatus() { return status; }

public String getSchedule() { return schedule; }

public String getTakerCid() { return takercid; }

public String getTakerNickname() { return takernickname; }

public String getGivernickname() { return givernickname; }

public double getTakerlat() { return takerlat; }

public double getTakerlon() { return takerlon; }

public double getGiverlat() { return giverlat; }

public double getGiverlon() { return giverlon; }

public String getCreated() { return created; }

public Integer getHistoryid() { return historyid; }

public String getGivercid() { return givercid; }
```

Figura 36: getter() inseriti nella classe Match

3.2.2.3 DateTime

La classe DateTime gestisce le date per mantenere dei formati utili alle chiamate API. In particolare il metodo toString() farà un parse per restituire la data nel formato RFC 3339. La RFC 3339 definisce un profilo dell'ISO 8601 che può essere usato nella suite di protocolli Internet [9].

Nella classe sono stati aggiunto un costruttore e dei metodi di utilità:

- Il costruttore senza parametri che inizializza un DateTime con la data attuale (Figura 37)

```
public DateTime() {
    this(new Date());
}
```

Figura 37: costruttore senza parametri di DateTime

- Il metodo checkPassedSchedule() che prende in input uno schedule e ritorna true se questo è passato rispetto all'orario attuale, false altrimenti (Figura 38).

```
/**
 * Return true if the schedule is old compared to now
 * @param schedule schedule to compare
 */
public static boolean checkPassedSchedule(String schedule) {
    DateTime scheduleTime = new DateTime(schedule);
    DateTime timeNow = new DateTime();
    return scheduleTime.getValue() < timeNow.getValue();
}
```

Figura 38: metodo checkPassedSchedule()

- Due metodi getHour() e getMinute() che ritornano rispettivamente l'ora e i minuti dell'oggetto DateTime (Figura 39).

```
/**
 * return the hour of this {@Link DateTime}
 */
public int getHour() {
    return Integer.parseInt(this.toString().substring(11, 13));
}

/**
 * return the minute of this {@Link DateTime}
 */
public int getMinute() {
    return Integer.parseInt(this.toString().substring(14, 16));
}
```

Figura 39: metodi getHour() e getMinute()

- Un metodo statico firstSchedule() che restituisce il primo schedule disponibile sulla base dell'ora attuale (Figura 40).

```
/**
 * return the first schedule considering current hour
 */
public static String firstSchedule() {
    DateTime dateTimeNow = new DateTime();
    int hourNow = dateTimeNow.getHour();
    int minuteNow = dateTimeNow.getMinute();
    minuteNow = (minuteNow / 15) * 15 + 14;
    return dateTimeNow.toString().substring(0, 11)
            + String.format(Locale.getDefault(), format: "%02d", hourNow) + ":"
            + String.format(Locale.getDefault(), format: "%02d", minuteNow)
            + dateTimeNow.toString().substring(16);
}
```

Figura 40: metodo firstSchedule()

- Due metodi newDateAddMinute() e newDateSubtractMinute() che prendono in input un numero di minuti e ritornano un nuovo DateTime con l'attributo value a cui vengono aggiunti o sottratti i minuti in input (Figura 41).

```
public DateTime newDateAddMinute(int numberOfMinute) {
    return new DateTime( value: value + 60000 * numberOfMinute);
}

public DateTime newDateSubtractMinute(int numberOfMinute) {
    return new DateTime( value: value - 60000 * numberOfMinute);
}
```

Figura 41: metodi newDateAddMinute() e newDateSubtractMinute()

3.2.2.4 GCConstants

GCConstants è una classe che ha al suo interno tutte le costanti dell'applicazione. Quello che è stato aggiunto è una costante relativa a deltalat e deltalon, delle chiamate API createTaker e createGiver, nominata DELTA_LAT_LONG con valore di 0.005f.

3.2.2.5 IGeneroCityAPI

Questa classe ha al suo interno tutte le chiamate API usate nell'applicazione ed è stata aggiunta la definizione per la chiamata removeDriverHistory.

3.2.2.6 Car

Car è il tipo che descrive un'auto all'interno dell'applicazione. Sono stati aggiunti alcuni metodi di utilità come:

- isInMatchWaiting(): ritorna true se il Match più recente ha lo stato di "waiting", false altrimenti;
- isInMatchRunning(): ritorna true se il Match più recente ha lo stato di "running", false altrimenti;
- isInMatchSuccess(): ritorna true se il Match più recente ha lo stato di "success", false altrimenti;
- isInMatchUnsuccessGiver(): ritorna true se il Match più recente ha lo stato di "unsuccess-giver", false altrimenti;
- isInMatchUnsuccessTaker(): ritorna true se il Match più recente ha lo stato di "unsuccess-taker", false altrimenti;
- isInMatchExpired(): ritorna true se il Match più recente ha lo stato di "expired", false altrimenti;
- getLastMatch(): se non sono presenti Match ritorna null, altrimenti ritorna il Match più recente, ovvero, il Match con il valore di DateTime più alto (Figura 42);

```
public Match getLastMatch() {
    if (match == null) {
        return null;
    }
    Match max = match.get(0);
    DateTime dateMax = new DateTime(max.getCreated());
    for (Match m : match) {
        DateTime dateM = new DateTime(m.getCreated());
        if (dateM.getValue() > dateMax.getValue()) {
            dateMax = dateM;
            max = m;
        }
    }
    return max;
}
```

Figura 42: metodo getLastMatch()

3.2.3 Le classi modificate per il FrameWork

Per la realizzazione del lavoro sul FrameWork descritto nell' introduzione (intro.) è stato necessario ricostruire da zero la classe CarManager.

3.2.3.1 CarManager

CarManager è una classe del Framework che si occupa di gestire le auto salvate localmente sul dispositivo. Quello che è stato fatto è utilizzare SQLite per creare un database locale. La scelta di questo sistema è anche dovuta dal risultato di alcuni benchmark, i quali hanno constatato che SQLite è più veloce del 35% su piccoli BLOB rispetto al filesystem [10].

Il database è stato strutturato con due entità:

- CAR. Rappresenta una normale auto ed è composto da un solo attributo che è anche chiave primaria: BLUETOOTHMAC. Associati a CAR ci sono i metodi per gestire: un INSERT, una DELETE, un UPDATE e una query SELECT *;
- BLACK_LISTED_CAR. Un'entità che rappresenta una nuova auto connessa al dispositivo in cui l'utente ha selezionato l'opzione "Non chiedermelo più". Come per CAR, anche BLACK_LISTED_CAR ha dei metodi per gestire: un INSERT, una DELETE, un UPDATE e una query SELECT *.

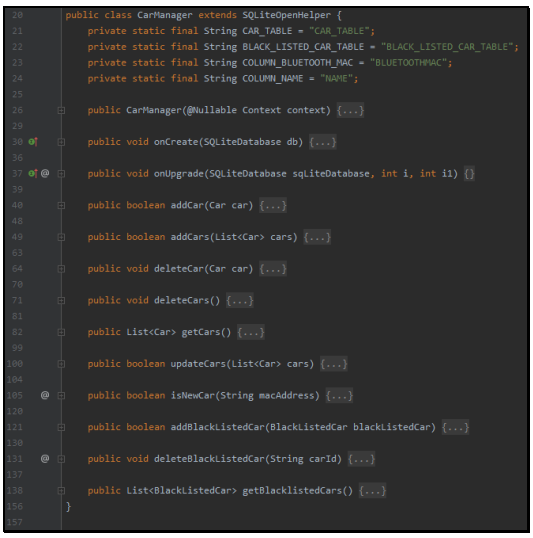

Figura 43: CarManager nel FrameWork

Capitolo 4

Conclusioni

4.1 Sviluppi futuri

L'applicazione Android di GeneroCity ha subito una pesante ristrutturazione che ha causato la mancanza di molte funzionalità di base come: il parcheggio, il cerca posto, il lascia posto e altro ancora. In questa relazione è stato trattato in particolare le funzionalità di "Lascia posto" e "Cerca posto" che per essere ultimate avrebbero bisogno di altri elementi quali:

- TimePicker[3]: nella schermata in cui ora è presente la lista degli schedule (Figura 6) si è pensato in futuro di sostituirla con un TimePicker, in modo tale da renderla conforme allo standard Android;

- L'utilizzo di un'immagine al posto del Nome del potenziale Giver o Taker all'interno dell'interfaccia poiché la loro identità è ininfluente (Figura 6);

- Mostrare sulla mappa il percorso che il Taker dovrà eseguire per raggiungere il Giver e la visualizzazione della posizione in tempo reale del Taker sulla mappa;

- L'update delle cards in tempo reale grazie all'utilizzo delle notifiche FireBase.

[3] *TimePicker: Orologio per la scelta di ore utilizzato nelle app android.*

4.2 Sommario

In conclusione, abbiamo visto come il parcheggio sia diventato uno dei più grandi problemi nelle città di oggi, dal punto di vista sociale ed economico. L'Applicazione di GeneroCity si presenta con l'obiettivo di far risparmiare tempo e denaro ai suoi utenti in cerca di parcheggio.

Attraverso le interfacce dell'applicazione è stato mostrato tutto il meccanismo alla base di uno scambio di parcheggio con l'utilizzo dell'app attraverso le due funzionalità principali di "Lascia posto" e "Cerca posto".

Abbiamo visto, inoltre, come la funzionalità di "Lascia posto" sia strettamente collegata a "Cerca posto". Per rendere possibile la loro implementazione all'interno dell'applicazione GeneroCity, sono state mostrate le nuove classi implementate e quelle che hanno avuto bisogno di modifiche.

Bibliografia

[1] Google Java Style Guide,
URL: https://google.github.io/styleguide/javaguide.html;

[2] Emanuele Panizzi. Caratteristiche del progetto GeneroCity. 2017.

[3] Le Fauconnier e Gantelet, The time looking for a parking space: strategies, associated nuisances and stakes of parking management in France. European Transport Conference (ETC), 2006, URL: https://trid.trb.org/view/846283;

[4] API (Application Programming Interface),
URL: https://en.wikipedia.org/wiki/API;

[5] Java, URL: https://en.wikipedia.org/wiki/Java_(programming_language);

[6] XML (Extensible Markup Language), URL: https://en.wikipedia.org/wiki/XML;

[7] Android Developers per l'utilizzo di Android Studio,
URL: https://developer.android.com/

[8] Material design per Android, URL: https://material.io/;

[9] G. Klyne, Clearswift Corporation, C. Newman, Sun Microsystems, Date and Time on the Internet: Timestamps, July 2002,
URL: https://tools.ietf.org/html/rfc3339;

[10] SQLite, 35% Faster Than The Filesystem, 2011,
URL: https://www.sqlite.org/fasterthanfs.html