UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

Facilitarea găsirii mașinii personale în cadrul parcărilor subterane: Find My Car

propusă de

Sebastian – Dragoș Ursulescu

Sesiunea: *iunie, 2016*

Coordonator ştiinţific

Asist. Dr. Vasile Alaiba

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

FACULTATEA DE INFORMATICĂ

Facilitarea găsirii mașinii personale în cadrul parcărilor subterane: Find My Car

Sebastian – Dragoș Ursulescu

Sesiunea: *iunie, 2016*

Coordonator ştiinţific

Asist. Dr. Vasile Alaiba

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul „*Find My Car*" este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;

- reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;

- codul sursă, imaginile etc. preluate din proiecte *open-source* sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;

- rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi, *27.06.2016*

Absolvent Sebastian – Dragoş Ursulescu

_____

(semnătură în original)

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licenţă cu titlul „*Find My Car*”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi, *27.06.2016*

Absolvent Sebastian – Dragoş Ursulescu

_____

(semnătură în original)

Rezumat

În această lucrare se descrie aplicația Android numită Find My Car împreună cu toate componentele folosite în realizarea acesteia. Find My Car este o aplicație pentru platforma Android care ajută la orientarea utilizatorului în cadrul parcării subterane a complexului Palas Iași, specificând de asemenea și suma aproximativă pe care o are de plătit utilizatorul la automatele de plată sau la ghișeurile amplasate la ieșirile din parcare. Aceasta reține prin scanarea unei inscripționări (Ex: D40) de pe stâlpii din apropiere, sau prin introducere manuală, locul unde mașina utilizatorului a fost parcată. Când acesta se va întoarce în parcare, va putea prin aceleași două metode descrise mai sus să „comunice" aplicației locația în care se află, iar în urma acestei comunicări aplicația va trasa drumul cel mai scurt până la locul de parcare înregistrat mai devreme și de asemenea va afișa o serie de sugestii pentru a ajuta la orientarea utilizatorului în cadrul parcării.

Cuvinte cheie

Android, parcare, aplicație, Palas, scanare, orientare, sugestii, automate, ghișeuri

Cuprins

Introducere

O serie de produse indispensabile oamenilor au fost aduse pe piață în urma progresului continuu a domeniului IT si al Telecomunicațiilor. Două dintre aceste produse sunt smartphone-urile si tabletele. În urmă cu ceva timp, aceste „gadget-uri" erau foarte rare și în același timp foarte scumpe, dar odată cu progresul tehnologiei, acestea au devenit din ce în ce mai accesibile unei categorii mai variate de persoane, prețul și specificațiile oferite fiind într-o relație de inversă proporționalitate. Astfel, astăzi putem găsi smartphone-uri sau tablete foarte ieftine cu specificații de top. În plus, avem și o varietate mult mai mare de firme de unde putem alege.

În ziua de azi, smartphone-urile joacă un rol foarte important în viața oricărei persoane care deține un astfel de dispozitiv. Pe lângă acțiunile clasice care pot fi efectuate cu acestea, precum apeluri telefonice, SMS-uri, notițe, alarme, smartphone-urile dispun și de camere foto/video de rezoluție înaltă, difuzoare de calitate maximă, o multitudine de senzori(accelerometru, temperatură, gravitație, giroscop, lumină ambientală, câmp magnetic, proximitate, umiditate) precum și un modul GPS pentru localizare și un modul Wi-Fi folosit la conexiunea utilizatorului la internet. Unele smartphone-uri dispun de procesoare și memorii RAM mai performante până și față de unele laptop-uri sau PC-uri, rezultând astfel că partea software ramâne la latitudinea și imaginația dezvoltatorului de aplicații pentru sistemul de operare care rulează pe dispozitiv.

În funcție de firma de la care provine smartphone-ul, se distinge o serie de sisteme de operare pentru platformele mobile: Android, iOS, Windows, Blackberry OS, FirefoxOS, Sailfish, etc. Cele mai întâlnite și folosite sisteme de operare sunt Android, iOS, Windows și Blackberry OS.

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
2015Q2	82.8%	13.9%	2.6%	0.3%	0.4%
2014Q2	84.8%	11.6%	2.5%	0.5%	0.7%
2013Q2	79.8%	12.9%	3.4%	2.8%	1.2%
2012Q2	69.3%	16.6%	3.1%	4.9%	6.1%

Figura 1: Tabelul popularității sistemelor de operare pentru platformele mobile în perioada 2012-2015[1]

[1] *http://www.idc.com/prodserv/smartphone-os-market-share.jsp*

După cum se poate observa din tabelul precedent, cele mai populare sisteme de operare sunt Android și iOS, acestea ocupând în anul 2015 82.8%, respectiv 13.9% din piață. Pentru dezvoltarea aplicației a fost ales sistemul de operare Android deoarece:

- Este cel mai răspândit sistem de operare pentru telefoanele mobile rulând pe sute de milioane de smartphone-uri și tablete
- Documentația acestuia este foarte bine pusă la punct și se poate găsi toată într-un singur loc (http://developer.google.com), iar problemele standard se pot găsi foarte ușor printr-o simplă căutare pe google, deci nu se pierde foarte mult timp când vine vorba de rezolvarea problemelor din cod
- Datorită faptului că mediul de dezvoltare Android Studio este gratuit și se poate rula pe Windows, Linux și Mac OSX, iar contul pentru Google Play este doar 25$, Android a devenit sistemul de operare favorit al dezvoltatorilor de aplicații mobile

Pentru platforma Android există mai multe surse și modalități de distribuire a aplicațiilor, principala fiind Google Play.

1. Tehnologii folosite

1.1 Android API

Aplicațiile destinate platformei Android sunt create cu ajutorul API-ului Android, acestea rulând pe mașina virtuală Dalvik. API-ul Android este scris în limbajul de programare Java de către dezvoltatorii platformei Android, acesta fiind public și gratuit, având de asemenea și o pagină web dedicată documentației API-ului plus câteva „unelte” suplimentare (tutoriale, „best practices” în development precum și design, aplicații de test, etc.) pentru a ajuta dezvoltatorii de astfel de aplicații. Există mai multe versiuni de Android API întrucât acesta se actualizează la fiecare versiune nouă de Android. Astfel, Android API 1 corespunde primei versiuni de Android, însă prima versiune de API facută publică este Android API 3 ce corespunde versiunii Android 1.5. După statisticile efectuate de Google, compania care a construit sistemul de operare pentru platformele mobile, repartiția versiunilor de Android pe toate dispozitivele care rulau acest sistem de operare la data de 6 iunie 2016 este următoarea:

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.9%
4.1.x	Jelly Bean	16	6.8%
4.2.x		17	9.4%
4.3		18	2.7%
4.4	KitKat	19	31.6%
5.0	Lollipop	21	15.4%
5.1		22	20.0%
6.0	Marshmallow	23	10.1%

Data collected during a 7-day period ending on June 6, 2016.
Any versions with less than 0.1% distribution are not shown.

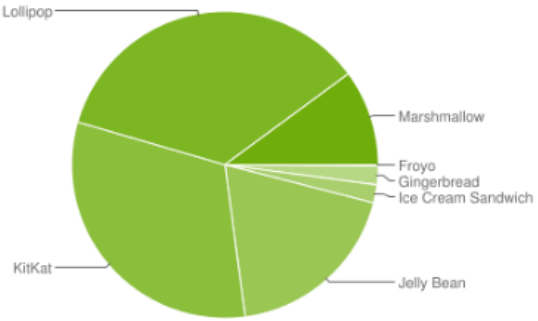


Figura 2: Diagrama popularității versiunilor de Android la data de 6 Iunie 2016[2]

[2] https://developer.android.com/about/dashboards/index.html?hl=ko

Aplicația Find My Car este dezvoltată în așa fel încât să ocupe aproape jumătate din dispozitivele pe care rulează sistemul de operare Android. Astfel, versiunea minimă pe care poate fi rulată aplicația este 5.0 (Android API 21), iar versiunea maximă fiind cea mai nouă 6.0 (Android API 23) urmând ca spre sfârșitul anului să apară noua versiune de Android și anume 7.0, precum și noul Android API 24, iar odată cu apariția acestora se va putea face o actualizare a versiunii maxime pe care va putea rula aplicația.

(Fun fact: Se poate observa faptul că primele litere ale versiunilor de Android sunt în ordine alfabetică, iar versiunea 7.0 mai este cunoscută printre fanii Android ca și Android N, din moment ce versiunea 6.0 a fost Marshmallow. Un zvon care circulă printre dezvoltatori susține că următoarea versiune de Android va avea numele de cod „Nutella”.)

1.2 Android Studio

Mediul de dezvoltare recomandat de Google pentru platforma Android este Android Studio. Acesta a fost anunțat pe data de 16 Mai 2013 la conferința anuală Google I/O din respectivul an. Mediul de dezvoltare a intrat în „early access preview” cu versiunea 0.1 în luna Mai a anului 2013, după care a intrat în „beta stage” începând cu versiunea 0.8 în Iunie 2014. Prima versiune stabilă a acestui software a fost lansată în Decembrie 2014 începând de la versiunea 1.0 și continuând până în ziua de azi, în prezent fiind la versiunea 2.1.2. Acest mediu de dezvoltare este bazat pe software-ul „IntelliJ IDEA” aparținând companiei de software JestBrains și a fost creat special pentru dezvoltarea aplicațiilor pentru platforma Android.

Android Studio, odată cu apariția sa, a înlocuit ADT-ul (Android Development Tools) mediului de dezvoltare Eclipse, adăugând în același timp o interfață mult mai prietenoasă pentru dezvoltatori, precum și unelte mai ușor accesibile și mai ușor de folosit. Pe lângă acestea, a fost adăugat și un „build automation system” numit Gradle cu ajutorul căruia se pot configura build-urile facute pentru aplicațiile Android mult mai simplu și mai intuitiv. De asemenea cu Gradle se pot descărca biblioteci specifice Android-ului printr-o simplă linie de cod și un „click” pe butonul de „Sync Project with Gradle Files”.

Din moment ce acum ne aflăm la al doilea mediu de dezvoltare pentru platforma Android și au intervenit multe schimbări odată cu trecerea de la Eclipse la Android Studio, se poate preciza faptul că acum trebuie respectată o altă structură de directoare și fișiere față de cea din primul mediu de dezvoltare:

- *build.gradle(project)* este fișierul care reprezintă configurațiile care se aplică la toate modulele aplicației
- Numele-modulului/
    - *build/* este directorul care conține „the build outputs”
    - *libs/* este directorul care conține Bibliotecile private
    - *build.gradle(module)* este fișierul care reprezintă configurațiile pentru build-ul modulului
    - *src/* este directorul care conține tot codul și toate resursele pentru modulul din care face parte
        - *androidTest/* conține codul pentru testele automate care ruleaza pe device-ul cu Android
        - *main/* conține toate fișierele sursă principale
            - *Android Manifest.xml* „descrie natura aplicației și toate componentele sale”
            - *java/* conține toate fișierele sursă Java
            - *jni/* conține toate fișierele de cod nativ folosind „Java Native Interface (JNI)”
            - *gen/* conține fișierele Java generate de către Android Studio cum ar fi R.java sau interfețele create din fișierele AIDL
            - *res/* conține toate resursele care sunt folosite pentru crearea aplicației cum ar fi layout-uri sau „UI strings”
            - *assets/* conține fișierele care ar trebui incluse în fișierul *.apk*
        - *test/* conține codul pentru testele locale care rulează în JVM

1.3 Optical Character Recognition

Optical character recognition (OCR) este „conversia mecanică sau electronică a imaginilor cu text scris la tastatură, scris de mână sau printat în text *machine-encoded* fie dintr-un document scanat, o poză a unui document, a unui semn sau a unei pancarde. Acesta este folosit pentru a prelua date din documente printate, pașapoarte, facturi, mail-uri, sau orice fel de documentație. Este o metodă de a aduce textul printat în format digital astfel încât să poată fi editat, interogat, salvat mai compact și folosit pentru traducere electronică, *text mining* și multe altele”[3].

Primele versiuni ale acestei tehnologii au fost antrenate cu câte o imagine a fiecărui caracter și funcționau doar pentru un singur tip de font. Acum, odată cu progresul tehnologiilor, se pot găsi „sisteme mai avansate capabile sa producă un grad de recunoaștere cu acuratețe mult mai mare pentru majoritatea font-urilor existente și cu suport pentru mai multe extensii de fișiere de tip imagine”[3].

Această tehnologie este folosită în cadrul aplicației Find My Car prin intermediul bibliotecii *tess-two* descrisă mai jos la secțiunea **2. Biblioteci Folosite**, această bibliotecă fiind mai mult un „wrapper” peste biblioteca *Tesseract* creată de aceeași companie care a creat sistemul de operare Android și anume Google.

[3] *https://en.wikipedia.org/wiki/Optical_character_recognition*

2. Biblioteci folosite

2.1 Biblioteca de OCR tess-two

Tess-two este „un *fork* a uneltei Tesseract pentru Android peste care s-au adăugat câteva funcționalități. Unealta Tesseract pentru Android este reprezentată de un set de API-uri Android pentru Bibliotecile care se ocupă cu procesarea de imagini și anume *Tesseract OCR și Leptonica*”[4].

Această bibliotecă funcționează pentru versiunea 3.5.00dev de Tesseract, 1.73 de Leptonica, libjpeg 9b și libpng 1.6.20. Codul sursă a acestor dependențe se află în directorul *tess-two/jni* din proiectul care poate fi descărcat accesând link-ul din bibliografie de la punctul **7**. Modulul tess-two conține *tool-uri* pentru compilarea Bibliotecilor Tesseract și Leptonica pentru a putea fi folosite cu ușurință în cadrul unui proiect Android. De asemenea, tot la același link din biliografie se pot găsi testele automate făcute pentru biblioteca tess-two care se află în directorul *tess-two-test*.

Biblioteca tess-two este compatibilă de la versiunea de Android 2.3 în sus, iar pentru antrenarea acesteia va fi necesar un fișier de antrenare pentru versiunea 3.02+. Acest fișier va trebui situat într-un subdirector cu numele *tessdata*, iar părintele acestui director dat ca și parametru la funcția *init()* a bibliotecii.

Pentru a folosi această bibliotecă într-un proiect Android va trebui editat fișierul *build.gradle* a modulului unde va fi folosită biblioteca în felul următor:

```
dependencies{
      compile 'com.rmtheis:tess-two:6.0.1'
}
```

În cadrul aplicației Find My Car directorul *tessdata* pentru antrenare a fost inserat în directorul *assets* pentru a putea fi preluat cu ajutorul clasei *AssetManager* din Android API și copiat în spațiul rezervat aplicației din dispozitivul care rulează Android. După copierea fișierului, a fost apelată funcția *init()* a bibliotecii cu cei doi parametri obligatorii care sunt *path-ul* unde se află directorul *tessdata* (DATA_PATH) și limba la care ar trebui sa se aștepte biblioteca atunci cand va face procesarea imaginilor (lang). După acest apel a fost setată o variabilă (VAR_CHAR_WHITELIST) pentru a restricționa caracterele care vor fi recunoscute în imaginea procesată.

```
tessBaseAPI.init(DATA_PATH, lang)
tessBaseAPI.setVariable(TessBaseAPI.VAR_CHAR_WHITELIST,"ABCDEFGHIJKLMNOPQRSTUVW\n")
```

[4] https://github.com/rmtheis/tess-two

Biblioteca *tess-two* este folosită atunci când utilizatorul alege să scaneze o imprimare de pe un stâlp al parcării în locul introducerii manuale oferite și astfel în urma fotografiei făcute semnului, în care se găsește o literă aflată în intervalul A-W și un număr sub acea literă, se obține o structură de date de tip *Bitmap* care este dată ca și parametru funcției *setImage()* din cadrul bibliotecii, iar apoi se extrage textul aflat în imagine prin apelarea funcției *getUTF8Text()* care va returna un String în care se află textul recunoscut.

```
tessBaseAPI.setImage(bitmap);
tessBaseAPI.getUTF8Text();
```

2.2 Biblioteca SugarDB

Biblioteca SugarDB este un Object-Relational Mapping (ORM) care ușurează relaționarea dezvoltatorului cu baza de date prin faptul că elimină necesitatea de a scrie cod SQL pentru a interacționa cu o bază de date SQLite, se ocupă de crearea bazei de date, își construiește singură relațiile între obiecte și oferă un API simplu pentru manipularea obiectelor stocate.

Ea se include foarte ușor în cadrul unui proiect Android prin editarea fișierului *build.gradle* a modulului în care va fi folosită biblioteca, introducând între acoladele de la *dependencies* următoarea linie:

```
dependencies{
      compile 'com.github.satyan:sugar:1.4'
}
```

După includerea bibliotecii în cadrul proiectului se poate configura baza de date prin editarea fișierului *AndroidManifest.xml* pentru a schimba numele fișierului bazei de date, versiunea acesteia, a selecta dacă dorim sau nu să se scrie într-un fișier codul SQL generat pentru instrucțiunile de SELECT și a pune un nume de pachet unde se află toate clasele noastre *model* pentru a ușura crearea tabelelor.

API-ul oferit de această bibliotecă este unul foarte simplist și intuitiv. Obiectele java create în care se vor stoca informații vor trebui derivate din clasa *SugarRecord* care este clasa de bază a unui obiect din baza de date, această clasă având funcții precum *save()* sau *delete()* care ușurează interacțiunea cu informațiile reținute pe disc. Astfel, după ce stabilim toate informațiile pe care le dorim salvate într-un obiect, pentru a funcționa versiunea *1.4* a bibliotecii va trebui făcut un constructor fără parametri pentru a putea fi creat tabelul, altfel va da eroare

și de asemenea va trebui oprită funcționalitatea de *Instant Run* din mediul de dezvoltare Android Studio, deoarece încă nu a fost rezolvată această problemă deși este cunoscută, iar dezvoltatorii bibliotecii încearcă să o rezolve în următoarea versiune.

Această bibliotecă este folosită în cadrul aplicației Find My Car pentru a stoca graful orientat folosit în calcularea drumului de cost minim folosind algoritmul Dijkstra precum și informații despre locurile de parcare și locurile salvate până atunci pentru a le putea afișa în lista din ecranul principal. Cu ajutorul API-ului simplist salvarea unui obiect sau modificarea acestuia este foarte ușoară și se poate efectua prin modificarea variabilelor alese mai apoi apelând funcția *save()* care va returna id-ul din baza de date a obiectului salvat.

Aceasta este clasa care reprezintă informațiile unui loc de parcare:

```
public class ParkingSpot extends SugarRecord{
      private String text;
      private Color color;
      private long timestampOfParking;

      public ParkingSpot(){}

      public ParkingSpot(String text, Color color){
            this.text = text;
            this.color = color;
            this.timestampOfParking =DateTime.now().getMillis();
      }
}
```

Salvarea unui nou obiect se face în felul următor:

```
new ParkingSpot(text, Color.findById(Color.class,color.save)).save()
```

Listarea tuturor obiectelor salvate de un anumit timp se poate face astfel:

```
ParkingSpot.listAll(ParkingSpot.class);
```

După cum se poate observa API-ul oferit este unul foarte simplu și ușor de folosit în ciuda faptului că este mai lent decât unele ORM-uri pentru platforma Android care se găsesc la momentul actual, dar în acest caz se poate accepta un compromis când vine vorba de viteza de căutare, întrucât baza de date nu este atât de amplă, aceasta conținând mai puțin de 1000 de elemente.

2.3 Biblioteca Joda-Time-Android

API-ul Android are în componența sa clase care se ocupă de informațiile temporale, deci de ce anume ar dori un dezvoltator să folosească o bibliotecă și să mărească dimensiunea fișierului *.apk*?

Într-adevăr sunt oferite clase pentru aceste informații temporale, dar API-ul oferit nu este unul foarte prietenos, iar dacă se dorește să se efectueze operații mai complicate pe aceste date, dezvoltatorul va trebui să fie foarte atent și va depune mult efort atunci când va încerca să rezolve aceste probleme pentru a putea fi folosite peste tot.

Joda-Time este o bibliotecă care rezolvă aceste probleme, aceasta oferind un API foarte prietenos, multe funcții care ajută la manipularea informațiilor temporale dorite precum și operații între două obiecte de tip *DateTime*. „Pentru dezvoltatorii de aplicații Android, această bibliotecă rezolvă o problemă foarte critică și anume *stale timezone data*. Informațiile despre timezone care sunt deja încorporate în Android se actualizează doar atunci când se actualizează și sistemul de operare”, iar dacă acest lucru se întâmplă destul de rar la dispozitivele sub marca Nexus pe care se rulează sistemul de operare pur, pe dispozitivele care se află sub alte mărci se întâmplă mult mai rar și de la un anumit moment chiar deloc.

Această bibliotecă se poate adăuga foarte simplu într-un proiect al unei aplicații Android prin editarea fișierului *build.gradle* aparținând modulului unde se va folosi biblioteca prin adăugarea codului de mai jos:

```
dependencies{
      compile 'net.danlew:android.joda:2.9.3.1'
}
```

După adăugarea bibliotecii, pentru a putea lucra cu api-ul oferit de aceasta trebuie apelată funcția *init()* în cadrul funcției *onCreate()* din clasa care extinde clasa *Applicaton*, iar această clasă trebuie declarată în fișierul *AndroidManifest.xml* ca și aplicația default.

```
public class MyApp extends Application{
      @Override
      public void onCreate(){
            super.onCreate();
            JodaTImeAndroid.init(this);
      }
}
```

Această bibliotecă este folosită în cadrul aplicației Find My Car pentru a avea un reper temporal a momentului în care a fost salvat locul de parcare a mașinii, precum și pentru a face toate operațiile necesare între momentul în care utilizatorul intră în aplicație și momentul în care a fost salvată locația mașinii. Astfel, atunci când se creează un nou rând în baza de date cu informațiile unui nou loc de parcare se setează în variabila de tip *long* a clasei ParkingSpot timestamp-ul curent:

```
this.timestampOfParking = DateTime.now().getMillis();
```

Atunci când se afișează lista cu locurile de parcare salvate, se calculează, după criteriile impuse de Palas Iași, suma aproximativă care trebuie plătită pentru parcare la automatele de plată sau la ghișeurile situate la ieșirile din parcare. Astfel, se calculează numărul de ore care au trecut de la momentul parcării și se adaugă o sumă de 2RON dacă ora este în intervalul orar 09:00-23:00 și o sumă de 1RON dacă ora care a trecut este în intervalul orar 23:00-09:00. Motivația folosirii acestei Biblioteci este faptul că API-ul oferit de aceasta este unul foarte simplist și astfel calculele explicate precedent se pot efectua foarte simplu deoarece accesul la informațiile temporale necesare este foarte direct, minimizând codul scris de dezvoltator.

2.4 Biblioteca AdMob

AdMob este o companie pentru „*mobile advertising* fondată de Omar Hamoui. Numele companiei vine de la *advertising on mobile* și a fost construită în luna Aprilie a anului 2006, în timp ce fondatorul era încă la universitatea *Wharton*. Această companie a fost achizoționată de Google pe data de 27 Mai 2010 pe suma de 750 de milioane de USD”[5].

Admob este una dintre cele mai mari platforme de *mobile advertising* din lume, aceasta încărcând mai mult de 40 de miliarde de *banner*-e și reclame text pe lună, însumate de pe toate site-urile web si aplicațiile mobile.

Folosirea ei este relativ simplă. După crearea unui cont pe site-ul AdMob de la punctul **11. AdMob** din bibliografie, se urmărește un tutorial simplu pentru a crea o „aplicație” pentru a obține un App ID care va fi folosit în aplicație pentru a inițializa biblioteca. După crearea „aplicației”, se poate urmări cu ușurință tutorial-ul de pe pagina web de la punctul **12. Getting started with Firebase in Android Studio** din bibliografie pentru a prelua fișierul *google-services.json*, după care va trebui editat fișierul *build.gradle* al aplicației pentru a introduce între acoladele de la *dependencies* classpath-ul bibliotecii:

```
classpath 'com.google.gms:google-services:3.0.0'
```

[5] https://en.wikipedia.org/wiki/AdMob

După aceste operațiuni, în cadrul fișierului *build.gradle* al modulului se va introduce sus plugin-ul pentru a-l putea folosi, precum și biblioteca pentru reclame:

```
apply plugin: 'com.google.gms.google-services'
dependencies{
      compile 'com.google.firebase:firebase-ads:9.0.2'
}
```

Pentru a include aceste dependențe trebuie apăsat butonul de *Sync Gradle*, iar apoi va trebui introdus în fișierul strings.xml id-ul reclamei create care se poate găsi în contul platformei Firebase. După toate aceste operațiuni putem introduce *View*-ul pentru reclame numit *AdView* în fișierul layout al activității noastre, adăugând proprietățile *adSize* și *adUnitId* care folosesc la recunoașterea reclamei pe server-ul AdMob.

```
<com.google.android.gms.ads.AdView
                  android:id="@+id/av_activity_main"
                  android:layout_width="match_parent"
                  android:layout_height="wrap_content"
                  android:layout_alignParentBottom="true"
                  app:adSize="BANNER"
                  app:adUnitId="@string/banner_ad_unit_id"/>
```

După introducerea *View*-ului, trebuie inițializată biblioteca în cadrul funcției *onCreate()* din activitatea principală și încărcată reclama folosind un *AdRequest* care va fi dat ca și parametru funcției *loadAd()* care aparține clasei *AdView*:

```
protected void onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      MobileAds.initialize(getApplicationContext(),
                           "ca-app-pub-4102134113002162~2908220331");
      AdView adView = (AdView) findViewById(R.id.av_activity_main);
      AdRequest adRequest = new AdRequest.Builder().build();
      adView.loadAd(adRequest);
}
```

2.5 Bibliotecă de suport pentru Material Design

Versiunea de Android 5.0 a fost foarte importantă din punct de vedere al progresului de care a dat dovadă Google în momentul în care a fost instaurat conceptul de Material Design.

Acest concept a fost unul care a „reîmprospătat toată experiența" de pe dispozitivele cu Android. Asfel, pe data de 29 Mai 2015 s-a „născut" biblioteca de suport pentru design-ul din Android care aduce animații diverse specifice design-ului Material, precum și o multitudine de *View*-uri noi care respectă „material guidelines", cum ar fi *Navigation View, TextInputLayout, FloatingActionButton, Snackbar, TabLayout* și multe altele care se pot găsi la link-ul de la punctul **13. Android Design Support Library** din bibliografie.

În cadrul aplicației Find My Car, această bibliotecă a fost folosită pentru a oferi o notă de noutate design-ului creat, dar mai ales pentru a fi în temă cu standardul de *Material Design*, întrucât acesta devine din ce în ce mai popular pe zi ce trece.

Din componentele bibliotecii a fost folosit *FloatingActionButton*, poziționat deasupra listei de locuri de parcare salvate și deasupra hărții care reprezintă parcarea de la Palas Iași, această poziție fiind una clasică conform „material guidelines" și chiar una ușor accesibilă pentru utilizator, precum și plăcută ochiului.

Această bibliotecă se include foarte simplu în cadrul unui proiect Android modificând fișierul *build.gradle* care aparține modulului din care se dorește să facă parte bibliotecă, adăugând între acoladele de la *dependencies* următoarea linie:

```
compile 'com.android.support:design:23.4.0'
```

După această modificare a fișierului se apasă pe butonul de SyncGradle în urma căruia biblioteca este descărcată și inclusă în proiect. Acum se pot folosi *View*-urile care fac parte din această bibliotecă, fiecare având proprietățile sale specifice și care pot fi folosite în diferite feluri.

FloatingActionButton este un simplu buton circular, de regulă cu o imagine de tip *Material* care reprezintă acțiunea pe care o efectuează atunci cand este apăsat. În cadrul aplicației acesta are doua funcționalități. În ecranul principal acesta are scopul de a deschide dialog-ul folosit pentru identificarea locului de parcare, în urma căreia se adaugă acel loc în listă, iar în ecranul de detalii a unui loc de parcare, este folosit pentru a căuta poziția utilizatorului în parcare pentru a-i putea oferi drumul cel mai scurt până la mașina acestuia. El se folosește în cod ca orice alt buton din API-ul Android, acesta fiind mai întâi inclus în fișierul layout al activității:

```
<android.support.design.widget.FloatingActionButton
                    android:id="@+id/fab_activity_main_add"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_above="@+id/av_activity_main"
                    android:layout_alignParentEnd="true"
                    android:layout_margin="16dp"
                    android:src="@drawable/ic_plus_sign"
                    app:backgroundTint="@color/palas_blue"
                    app:elevation="4dp" />
```

Se pot observa cu ușurință proprietățile speciale *backgroundTint* și *elevation* care fac parte din stilul pe care îl abordează design-ul Material. După ce se introduce în layout, acesta este preluat în clasă prin intermediul id-ului setat ca și proprietate și setată o acțiune pe care să o efectueze atunci cand este apăsat, adică i se setează un *OnClickListener* prin apelarea metodei *setOnClickListener* care aparține clasei *View* de bază pentru toate *View*-urile existente în Android:

```
FloatingActionButton fab_activity_main_add = (FloatingActionButton)
                              findViewById(R.id. fab_activity_main_add);
fab_activity_main_add.setOnClickListener(new View.OnClickListener(){
      @Override
      public void onClick(View v){
            //Ce va face butonul atunci când va fi apăsat
      }
});
```

2.6 Bibliotecă pentru Material Dialogs

În cadrul API-ului Android există clase care să producă *dialog*-uri, iar de la versiunea 5.0 *dialog*-urile native sunt sub „material guidelines” și oferă destul de multe funcționalități și metode de a le manipula, deci de ce am avea nevoie de o bibliotecă?

Această bibliotecă este construită de utilizatorul cu numele *afollestad* de pe github, iar prima versiune a fost distribuită pe github în urmă cu 2 ani. De atunci dezvoltatorul a actualizat această bibliotecă adăugând ceea ce cerea comunitatea, sau rezolva bug-urile care erau semnalate. El a dezvoltat această bibliotecă în ideea de a ușura munca dezvoltatorilor de aplicații Android atunci cand venea vorba de a face un *dialog* mai special decât cele oferite nativ de API-ul Android. De asemenea, biblioteca a fost creată și pentru a putea folosi design-ul tip Material care abia apăruse și nu putea fi folosit decât pe cele mai recente versiuni,

versiunile de la 4.4 în jos neavând acest tip de design care era foarte nou și plăcut. Astfel, odată cu apariția acestei Biblioteci, dezvoltatorii nu au mai trebuit să urmeze toate specificațiile design-ului Material pentru a crea de la zero un astfel de dialog, ci pur și simplu prin apelarea unor simple funcții din clasa builder oferită de API-ul bibliotecii care rezolvau problema.

Biblioteca de *Material Dialogs* nu este singura de pe piață, dar este cea mai ușor de folosit și cea mai diversificată. Aceasta se poate include foarte simplu într-un proiect Android modificând fișierul *build.gradle* al modulului din care se dorește să se folosească biblioteca adăugând următoarea linie între acoladele de la *dependencies*:

```
compile 'com.afollestad.material-dialogs:commons:0.8.5.9'
```

După modificarea fișierului trebuie apăsat butonul de *Sync Gradle* pentru a descărca biblioteca și a-i putea folosi clasele în cadrul proiectului. Aceasta se folosește foarte simplu și este foarte flexibilă. Câteva exemple se pot găsi la link-ul de la punctul **14. Material Dialogs** din bibliografie.

Biblioteca a fost folosită în cadrul aplicației Find My Car pentru a menține design-ul în aceeași temă și pentru a ușura crearea și modificarea lor. Astfel, pasul scanării sau a introducerii manuale se efectuează cu ajutorul dialog-urilor din această bibliotecă. Un exemplu de utilizare ar putea fi:

```
public static void showMainDialog(final AppCompatActivity activity){
       new AlertDialog.Builder(activity)
              .setTitle("SAVE PARKING SPOT")
              .setMessage("Scan a pillar or submit manually")
              .setPositiveButton("SUBMIT MANUALLY"
                     , new DialogInterface.OnClickListener(){
                            @Override
                            public void onClick(DialogInterface dialog, int which){
                                   showSubmitManuallyDialog(activity);
                            }
                     }
              ).show();
}
```

2.7 Biblioteca pentru Material Progress

În nevoia de a face câteva operațiuni asincrone, în cadrul unei aplicații android se poate folosi un „spinner” pe ecran care să indice acest lucru. Pentru a menține nota de noutate a design-ului s-a folosit biblioteca *Material-ish Progress* care oferă un astfel de *View* foarte simplu de inclus și de folosit.

Această bibliotecă se include prin modificarea fișierului *build.gradle* care aparține modulului în care se vrea folosită biblioteca adăugând între acoladele de la *dependencies* următoarea linie:

```
compile 'com.pnikosis:materialish-progress:1.7'
```

După acest pas trebuie apăsat butonul de *Sync Gradle* în urma căruia Android Studio va descărca biblioteca și o va include în proiect pentru a putea fi folosită. *View*-ul care este oferit în cadrul acestei Biblioteci este *ProgressWheel*, iar pentru a-l folosi acesta trebuie inclus în fișierul layout al activității. Se dorește să se blocheze contactul utilizatorului cu interfața grafică a aplicației și astfel acest *View* va fi inclus într-un *FrameLayout* care se va întinde pe tot ecranul și i se va da un negru transparent ca și culoare de fundal pentru a sugera faptul că nu se poate interacționa cu nimic din interfață:

```
<FrameLayout
      android:id="@+id/fl_activity_main_progress"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:background="#44000000"
      android:clickable="true"
      android:visibility="gone">

      <com.pnikosis.materialishprogress.ProgressWheel
            android:id="@+id/progress_wheel"
            android:layout_width="80dp"
            android:layout_heigth="80dp"
            android:layout_gravity="center"
            wheel:matProg_barColor="#0A33C5"
            wheel:matProg_progressIndeterminate="true" />
</FrameLayout>
```

Se pot observa cu ușurință proprietățile speciale *matProg_barColor*, care setează culoarea cercului care se va învârti și *matProg_progressIndeterminate*, care stabilește dacă *View*-ul se va roti la infinit sau nu. După ce a fost inclus în layout, tot ce mai trebuie făcut este afișarea acestuia atunci cand se începe o operație asincronă și ascunderea lui atunci când aceasta se termină.

În cadrul aplicației Find My Car, această bibliotecă s-a folosit pentru a ușura munca depusă pentru a face un „spinner” de la zero respectând specificațiile oferite de Material Design. Astfel, cu ajutorul bibliotecii, problema care ar fi durat o perioadă pentru a fi rezolvată a fost rezolvată în câteva minute cu succes și fără bug-uri.

3. Analiză și proiectare

3.1 Analiza problemei și găsirea soluției

Localizarea autovehiculului în cadrul parcării complexului Palas Iași reprezintă o provocare pentru mulți dintre clienții ansamblului, iar pentru angajații complexului reprezintă de multe ori o provocare zilnică. Astfel, rezolvarea acestei probleme este obligatorie întrucât singurele metode de orientare în parcare sunt hărțile tipărite și amplasate pe pereții parcării sau solicitarea ajutorului unui angajat al parcării care se află la unul dintre ghișeurile amplasate la ieșiri.

În urma unui studiu efectuat asupra metodelor de orientare în cadrul unui spațiu destul de amplu și subteran, am observat că principala problemă este lipsa semnalului GPS sau a conexiunii la internetf. Soluția cea mai folosită într-un astfel de caz este amplasarea de *beacon*-uri la poziții fixe, răspândite pe toată aria spațiului. Acestea funcționează pe bază de *Bluetooth Low Energy* și oferă informații spațiale clare ale dispozitivului care este în conexiune cu ele. Acestea au o baterie încorporată care rezistă foarte mult întrucât *Bluetooth*-ul folosit consumă foarte puțină energie, iar conexiunea dispozitivelor cu aceste *beacon*-uri se efectuează foarte ușor întrucât API-ul Android oferă o serie de clase cu care se poate stabili o conexiune, transmite informații, etc.

Din nefericire, în parcarea din cadrul ansamblului Palas Iași nu sunt instalate astfel de *beacon*-uri, deci ca și rezultat singura soluție va fi orientarea offline pe baza hărții și a reperelor vizuale, cum ar fi culoarea sau textul din componența simbolurior fiecărui loc de parcare sau stâlp, întrucât în unele locații un simbol semnifică mai multe locuri de parcare.

O altă necesitate ar fi obținerea hărților parcării și transformarea acestora în format digital. Acesta a fost un proces destul de îndelungat întrucât a fost nevoie de mai multe aprobări pentru a le putea deține. Prima dată am discutat cu Directoarea de Marketing al complexului căreia i-am explicat ce doresc să obțin prin această aplicație. Aceasta mi-a comunicat faptul că îmi va trebui o cerere care va fi înaintată Directorului Ansamblului Palas Iași. După redactarea cererii și trimiterea acesteia prin e-mail, a urmat o saptamână de așteptare a unui răspuns. În cele din urmă am primit din nou telefon de la Directoarea de Marketing care mi-a comunicat faptul că sunt foarte încântați de idee și doresc neapărat să mă ajute sa o duc la final. Pentru acest lucru mi-a trebuit o recomandare de la profesorul meu coordonator Asist. Dr. Vasile Alaiba care a fost înaintată domnului Iulian Dascălu, patronul ansamblului Palas Iași. După încă o saptamână de așteptare, am primit din nou telefon de la Directoarea de Marketing a complexului care mi-a confirmat faptul că proiectul a fost aprobat, iar hărțile au trebuit obținute

de la directorul parcării Palas Iași. Acesta mi-a oferit un fișier în format *.pdf* cu ajutorul căruia am reușit să transpun hărțile în format *.svg* pentru a le putea include în aplicație.

3.2 Proiectarea aplicației

În urma analizei și a găsirii soluției am stabilit cerințele aplicației Find My Car. Astfel, ca și funcționalități principale vom avea:

1. Detectarea locului de parcare a mașinii și a locului în care se află utilizatorul
2. Aflarea celui mai scurt drum până la mașina parcată
3. Orientarea cu ajutorul hărții și a unor informații ajutătoare

Pe lângă aceste funcționalități principale am mai introdus salvarea istoricului locurilor de parcare precedente, precum și afișarea unei aproximări a sumei pe care utilizatorul o va plăti la automatele de plată sau la ghișeurile aflate la ieșirile din parcare.

Detectarea locului de parcare a mașinii și a locului în care se află utilizatorul se face prin două metode, prima fiind introducerea manuală a textului de pe pilonul cel mai apropiat de acesta, iar a doua fiind detectarea automată a textului cu ajutorul tehnologiei OCR și a bibliotecii *tess-two* în urma unei fotografii capturate cu textul de pe un pilon al parcării. După detectare sau introducere manuală, i se vor afișa utilizatorului informațiile preluate, iar acesta va trebui să confirme dacă ele sunt corecte. De asemenea, se vor face verificări pentru ca utilizatorul să nu poată introduce un loc de parcare inexistent afișând un *dialog* în care se va specifica faptul că sistemul a întâmpinat o eroare.

Aflarea celui mai scurt drum până la mașina parcată se face cu ajutorul unui graf orientat pe care se aplică algoritmul lui Dijkstra de aflare a drumului de cost minim. Fiecare nod al grafului reprezintă un punct de pe harta parcării în jurul căruia se află mai mulți piloni însemnați, muchiile grafului reprezintă legaturile dintre acele puncte, iar costul de pe muchii este calculat ca distanța dintre două puncte dintr-un sistem de coordonate cu două axe.

Orientarea prin parcare va fi ușurată cu ajutorul hărții care are posibilitatea de a se roti la 360 de grade și posibilitatatea de a scala pentru a putea observa mai exact drumul care trebuie urmărit. De asemenea, deasupra hărții sunt afișate o serie de indicii ajutătoare semnificând pilonii pe lângă care va trece utilizatorul în drumul său spre mașina parcată. Utilizatorul va putea selecta un anumit punct din cele afișate, iar în urma acestei acțiuni, drumul se va actualiza cu locația lui curentă.

4. Arhitectura aplicației Android

Partea practică a lucrării constă într-o aplicație Android care încorporează toate funcționalitățile prezentate mai sus, precum și o metodă de stocare a grafului orientat și a locurilor de parcare. Conexiunea la GPS sau la internet este inexistentă, deci se poate deduce foarte simplu imposibilitatea de a face o conexiune cu un server de orice fel. Astfel, singurele două entități care vor comunica vor fi aplicația Android și baza de date inclusă în aplicație prin intermediul bibliotecii *SugarDB*. Mai jos se poate observa diagrama simplă a legăturii dintre aplicație și baza de date încorporată:

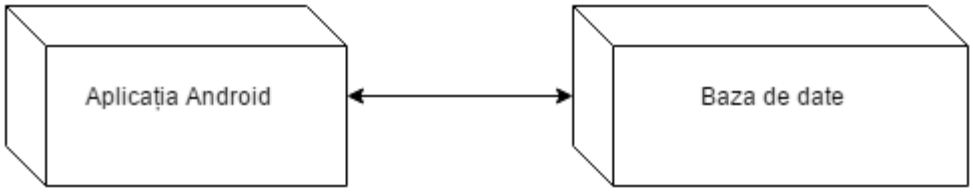


Figura 3: Diagrama legăturii dintre aplicație și baza de date

Aplicația Find My Car dezvoltată pentru platforma Android constituie subiectul principal al lucrării de licență. Aceasta este dezoltată în Android Studio, software destinat dezvoltării de aplicații pentru platforma Android.

Structura aplicației corespunde cu structura nouă recomandată de Android Studio, astfel componentele aplicației sunt:

- fișierele sursă Java ce se află în directorul */src/main/java* structurate pe pachete în dependență de funcția și scopul acestora, cum ar fi activități, adapter-e, holder-e de *View*-uri, clase utilitare, modele bazei de date, clasele ce ajută algoritmului Dijkstra sau clasele ce ajută la afișarea hărților în format svg împreună cu rotirea și scalarea acestora
- fișierele din directorul */src/main/res* care sunt fișiere de layout folosite pentru definirea interfeței grafice, fișiere de valori pentru string-uri care sunt împărțite pe două limbi și anume engleză și română, fișiere de dimensiuni, imagini în format *.svg* și icon-ul aplicației aflat în directorul *mipmap*
- fișierele din directorul */src/main/assets* care sunt reprezentate de datele de intrare pentru graful orientat, hărțile nivelelor de la parcare în format *.svg* și fișierul de antrenare pentru biblioteca *tess-two*

Activitatea în cadrul unei aplicații Android este componenta care oferă un ecran prin intermediul căruia utilizatorul interacționează cu aplicația. Fiecare activitate primește un fișier de tip layout care reprezintă interfața grafică a acelei activități. O aplicație pentru platforma Android conține de regulă mai multe activități dintre care una este activitatea principală care este afișată atunci cand se deschide aplicația, iar dacă aplicația se află în background se va deschide ultima activitate afișată. Activitatea principală se specifică în fișierul *AndroidManifest.xml* printr-un set de *tag*-uri speciale:

```
<activity
      android:name=".activities.SplashScreen"
      android:screenOrientation="portrait">
      <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="anroid.intent.category.LAUNCHER" />
      </intent-filter>
</activity>
```

Fiecare activitate poate crea o altă activitate, iar în acel moment în care se creează o activitate nouă, cea care a deschis-o este oprită și memorată de sistem într-o stivă de activități numită „back-stack" pentru a se putea întoarce la ea dacă este nevoie. Pe parcursul rulării, activitatea trece prin mai multe stări, iar trecerea de la o stare la alta poate fi urmărită prin intermediul callback-urilor oferite de clasa *Activity*. Aceste callback-uri sunt *onCreate()* aceasta fiind primul callback apelat de activitate, după care urmează callback-urile *onStart()* și *onResume()*, iar după acestea activitatea este în starea de „running". Atunci când se închide o activitate sunt apelate pe rând callback-urile *onPause()* și *onStop()*, iar după aceste două apeluri activitatea se află în „back-stack". Aceste stări pot fi vizionate pe diagrama ciclului de viață a unei activități:

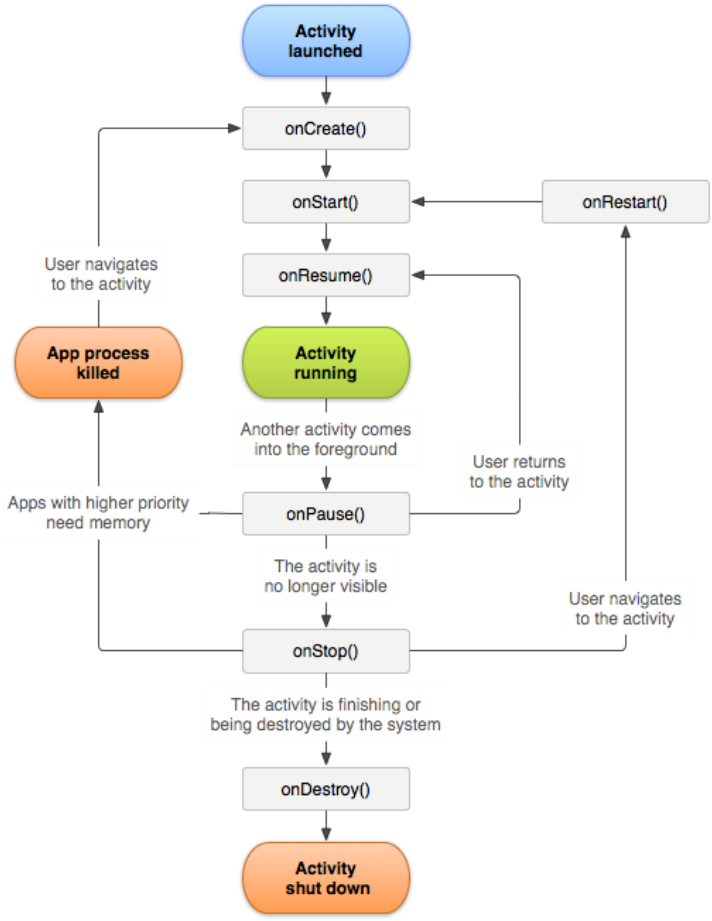


Figura 4: Diagrama ciclului de viață a unei activități[6]

Baza de date folosită în cadrul acestei aplicații este creată prin intermediul bibliotecii *SugarDB*, aceasta fiind o bază de date SQLite la care se fac interogări pentru a extrage informațiile necesare aplicației cum ar fi nodurile și muchiile din graful orientat folosit pentru algoritmul lui Dijkstra de aflare a drumului de cost minim între două noduri. Modelele acestei baze de date se află în pachetul *com.licenta.sebi.findmycar.database.models*, acestea reprezentând tabelele care vor fi create în baza de date, legăturile dintre acestea creându-se automat în funcție de variabilele din cadrul modelelor. Index-urile fiecărui tabel se creează și ele automat, ajutând la rapiditatea preluării datelor din cadrul bazei de date, în același timp ușurând munca depusă de către dezvoltator pentru construcția bazei de date. Folosirea acestei Biblioteci ușurează munca dezvoltatorului și atunci când vine vorba de salvarea sau ștergerea rândurilor precum și preluarea și manipularea datelor din baza de date. În următoarea imagine se pot observa relațiile dintre tabelele care sunt create în baza de date:

[6] *https://developer.android.com/reference/android/app/Activity.html*

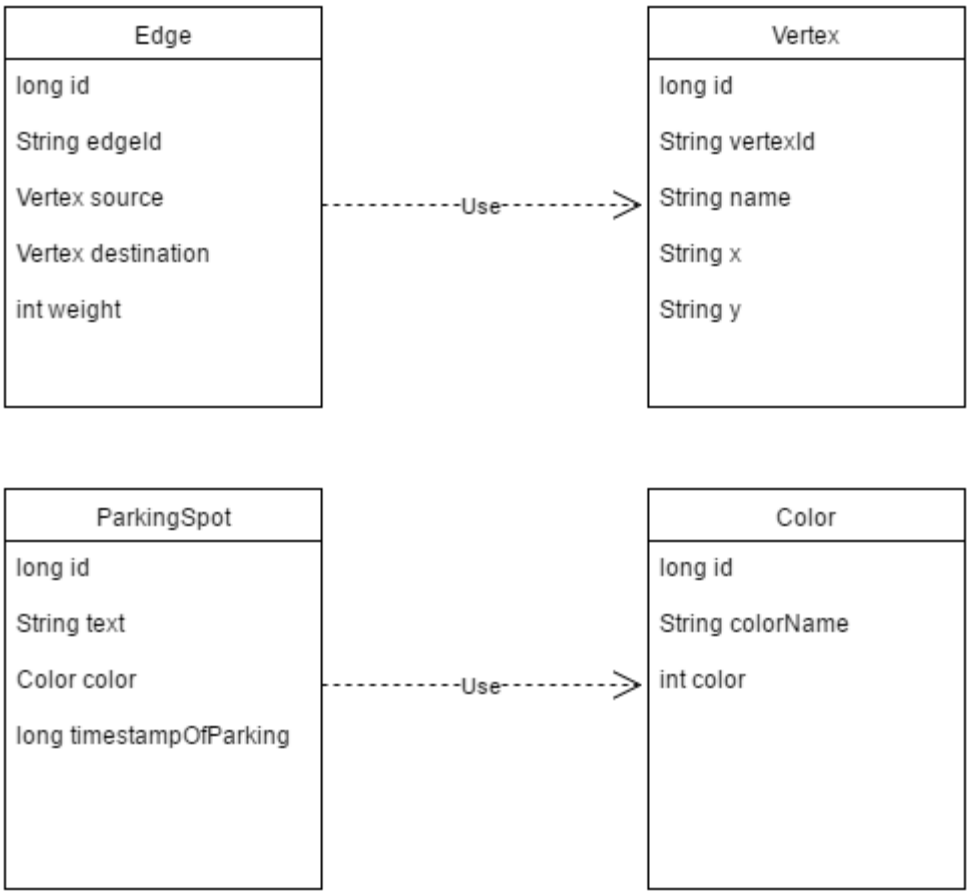


Figura 5: Diagrama bazei de date și relațiile între tabele

În pachetul *com.licenta.sebi.findmycar.dijkstra* se află clasa *DijkstraAlrotihm* cu ajutorul căreia se efectuează căutarea celui mai scurt drum între nodurile date ca sursă și destinație.

În pachetul *com.licenta.sebi.findmycar.models* se află clasa *Graph*. Această clasă este creată cu ajutorul nodurilor și muchiilor salvate în baza de date și este dat ca și parametru la clasa algoritmului Dijkstra.

În pachetul *com.licenta.sebi.findmycar.utils* se află clasele utilitare care ajută la salvarea culorilor, afișarea dialog-urilor, efectuarea pozei, afișarea hărților și procesarea pozelor cu ajutorul bibliotecii *tess-two.*

În pachetul *com.licenta.sebi.findmycar.viewholders* se află clasele de tip „holder” care ajută la afișarea informațiilor în lista din ecranul principal.

5. Implementarea aplicației Android

5.1 Implementarea activităților

La rularea aplicației Android este creată activitatea *SplashScreen*. Primul lucru care se face în această activitate este să se verifice dacă utilizatorul a aprobat permisiunile de scris pe disk și permisiunea de internet pentru a putea rula aplicația. Dacă aceste două permisiuni nu au fost aprobate de către utilizator se cere aprobarea lor, iar în caz contrar se continuă cu inițializarea aplicației:

```
if(ActivityCompat.checkSelfPermission(this,
              Manifest.permission.WRITE_EXTERNAL_STORAGE)
                     != PackageManager.PERMISSION_GRANTED){
       ActivityCompat.requestPermissions(this,
                     new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE,
                                   Manifest.permission.INTERNET},
                     PERMISSION_REQUEST_CODE);
} else {
       initApp();
}
```

După cererea permisiunilor și aprobarea acestora, se continuă cu inițializarea aplicației. Această activitate are în componența sa o interfață Java cu ajutorul căreia se semnalează atunci când s-a terminat de inițializat aplicația:

```
private interface OnAppInitFinishedCallback{
       void onAppInitFinished();
}
```

Aceasta se dă ca și parametru funcției *initializeApplication()*, care construiește un obiect de tip *Runnable*, transmis ca și parametru unui *Thread* pentru a face inițializarea asincronă, întrucât nu se dorește blocarea interfeței grafice pentru a afișa emblema Palas Iași.

```
initializeApplication(new OnAppInitFinishedCallback(){

	@Override
	public void onAppInitFinished(){
		startActivity(new Intent(SplashScreen.this, MainActivity.class));
		finish();
	}
})
```

În cadrul funcției *run()* a obiectului de tip Runnable, se inițializează biblioteca de reclame, se antrenează biblioteca *tess-two* pentru a o putea folosi în procesarea imaginilor dacă va fi cazul, se inițializează obiectul de tip *HashMap* în care se țin referințele la culori grupate după litere, după care se citește fișierul de inițializare a grafului orientat pentru a salva nodurile și muchiile, folosindu-le apoi la crearea grafului și a algoritmului Dijkstra pentru drumul de cost minim. După cum se poate observa, odată ce inițializarea aplicației s-a terminat, se pornește următoarea activitate, care este *MainActivity* și se oprește activitatea curentă *SplashScreen* folosind funcția *finish()*. În această activitate primul care se efectuează este preluarea referințelor la *View*-urile din interfața grafică pentru a avea posibilitatea de a le manipula:

```
AdView av_activity_main = (AdView) findViewById(R.id.av_activity_main);
RecyclerView rv_activity_main =(RecyclerView)findViewById(R.id.rv_activity_main);
FloatingActionButton fab_activity_main_add = (FloatingActionButton)
                                        findViewById(R.id.fab_activity_main_add);
fl_activity_main_progres = (FragmeLayout)
                                    findViewById(R.id.fl_activity_main_progress);
```

După acest pas se inițializează *View*-ul pentru reclame, după care se populează lista cu locurile de parcare salvate până în acel moment, iar apoi se atribuie un obiect de tip *OnClickListener* butonului de adăugare a unui loc de parcare care atunci când este apăsat va afișa *dialogul* principal cu cele două opțiuni de recunoaștere: scanare pe bază de poză și introducere manuală. Această activitate mai suprascrie două funcții ale clasei *AppCompatActivity* care face parte din API-ul Android, acestea fiind *onRequestPermissionsResult()* și *onActivityResult()*.

Prima funcție suprascrisă se apelează atunci cand se obține un rezultat care poate fi bun sau rău, în urma cererii permisiunilor pentru folosirea camerei din componența dispozitivului, iar în urma apelului său este apelată funcția *onRequestPermissionsResult* care face parte din clasa *PhotoUtils* și care se ocupă cu preluarea datelor transmise ca și parametru la apelul funcției din activitate pe care le verifică și acționează în funcție de rezultatul primit.

```
@Override
public void onRequestPermissionRestult
             (int requestCode, String[] permission, int[] grantResults){
       super.onRequestPermissionsResult(requestCode, permissions,
                                                                grantResults);
       PhotoUtils.onRequestPermissionsResult(this, requestCode,
                                                                grantResults);
}
```

A doua funcție suprascrisă este apelată atunci când s-a finalizat capturarea unei fotografii folosind camera dispozitivului, iar în urma apelului său este apelată funcția *onActivityResult()* din cadrul clasei *PhotoUtils* care se ocupă cu preluarea datelor trimise ca și parametru la apelul funcției din activitate, pe care le verifică și acționează în funcție de rezultatul primit.

```
@Override
protected void onActivityResult
       (int requestCode, int resultCode, final Intent data){
       super.onActivityResult(requestCode, resultCode, data);

       PhotoUtils.onActivityResult(this, resultCode,

       fl_activity_main_progress);
}
```

La apăsarea unui element din lista de locuri de parcare salvate se deschide activitatea *ParkingSpotDetailsActivity*. Această activitate se ocupă cu afișarea hărții și a drumului determinat în urma algoritmului Dijkstra. În cadrul funcției *onCreate()* sunt preluate *View*-urile din cadrul fișierului layout al activității pentru a putea fi manipulate:

```
ll_parking_spot_details_hints = (LinearLayout)
                                  findViewById(R.id.ll_parking_spot_details_hints);
svgMapView = (SVGMapView) findViewById(R.id.svgMapView);
fab_parking_spot_details_activity_search = (FloatingActionButton)
                    findViewById(R.id. fab_parking_spot_details_activity_search);
fl_parking_spot_details_activity_progress = (FrameLayout)
                    findViewById(R.id.fl_parking_spot_details_activity_progress);
```

După acest pas se apelează funcția *drawVertexOnSVG* care face parte din clasa *SVGUtils* care desenează locul de parcare pe care a intrat utilizatorul pe hartă și se atribuie butonului de căutare a unei poziții în parcare un obiect de tip *OnClickListener*, iar atunci când acest buton va fi apăsat se va afișa *dialog*-ul principal cu cele două opțiuni de recunoaștere a poziției în parcare, adică scanare pe bază de poză și introducere manual. Această activitate mai suprascrie funcțiile *onRequestPermissionsResult()* și *onActivityResult* care aparțin clasei *AppCompatActivity* din cadrul API-ului Android. Acestea efectuează aceleași apeluri ca și cele din *MainActivity*. În cadrul activității mai există funcția statică *getActivityIntent()* care returnează un obiect de tip *Intent* pentru a ajuta la pornirea acestei activități.

```
public static Intent getActivityIntent(Context context, long
                                                        parkingSpotId){
       Intent intent = new Intent(context, ParkingSpotDetailsActivity.class);
       intent.putExtra(PARKING_SPOT_ID, parkingSpotId);

       return intent;
}
```

După recunoașterea unui loc de parcare se apelează funcția *drawRoadOnSVG()* care face parte din clasa *SVGUtils* și care caută asincron poziția utilizatorului, iar apoi apelează funcția de căutare a celui mai scurt drum din cadrul clasei *DijkstraAlgorithm,* iar după ce determină drumul îl desenează pe hartă și setează indiciile.

5.2 Adaptere

Aplicația Find My Car folosește componente de interfață complexe pentru a afișa un set de date mai mare. Pentru a face legătura dintre setul de date și componentele de interfață se implementează o serie de adaptere. În principiu adapter-ele creează și transmit obiectelor de interfață complexe componentele repetitive din cadrul lor cu ajutorul unui fișier de tip layout căruia i se face „inflate” și a unei liste de elemente ce trebuiesc afișate. În această aplicație a fost implementat un adapter pentru lista de tip *RecyclerView* din ecranul principal pentru a afișa toate locurile de parcare salvate. Adapter-ul *ParkingSpotsAdapter* este o clasă Java derivată clasei *RecyclerView.Adapter<T>* care ajută la afișarea listei de elemente pe interfața grafică. Acesta suprascrie funcțiile *onCreateViewHolder(), onBindViewHolder()* și *getIntemCount()* pentru a face posibilă această afișare.

Prima funcție suprascrisă, și anume *onCreateViewHolder()*, se apelează atunci când reciclarea elementelor de interfață nu mai este posibilă, și astfel este necesară crearea unui nou obiect de tip *RecyclerView.ViewHolder* pentru a avea o referință la *View*-urile din interfață:

```
@Override
public ParkingSpotViewHolder onCreateViewHolder(ViewGroup parent, int viewType){
       return new ParkingSpotViewHolder(
                     inflater.inflate(R.layout.parking_spot_item), parent, false);
}
```

A doua funcție suprascrisă, și anume *onBindViewHolder()*, se apelează pentru fiecare element afișat în parte sau atunci când acesta se schimbă, iar adapter-ul este notificat de schimbarea acestui element. În această funcție se atribuiesc toate proprietățile necesare View-urilor unui element grafic pentru a fi afișat, cum ar fi culori, text, acțiuni la apăsarea unor butoane, etc. În cadrul acestei funcții, în aplicația Find My Car, se ia obiectul de tip *ParkingSpot* din lista de obiecte pentru a folosi informațiile pe care le deține acesta, după care se atribuie un obiect de tip *OnClickListener* întregului element vizual pentru ca atunci când elementul este atins de utilizator să se pornească activitatea pentru detaliile locului de parcare selectat. În continuare se atribuie un obiect de tip *OnClickListener* imaginii din elementul grafic care reprezintă ștergerea elementului pentru a-l elimina din listă și din baza de date la nevoie. După acești pași se atribuie textul și culoarea de fundal al elementului grafic pentru text pentru a-i sugera utilizatorului elementul pe care urmează să apese, apoi se fac calculele pentru suma de plată care va trebui plătită la automatele de plată sau la ghișeurile aflate la ieșirile din parcare și se afișează tot într-un element grafic special pentru text:

```
@Override
public void onBindViewHolder(final ParkingSpotViewHolder holder, int position){
       final ParkingSpot parkingSpot = items.get(position);
       holder.root.setOnClickListener(new View.OnClickListener(){
              @Override
              public void onClick(View v){
                     activity.startActivity(
                     ParkingSpotDetailsActivity.getActivityIntent(activity,
                                                   parkingSpot.getId()));
              }
       });
       . . .
       //restul de cod care se ocupă de afișarea elementului
}
```

5.3 Fișierele Java utilitare

În cadrul aplicației Android Find My Car au fost implementate o serie de clase utilitare pentru a structura funcțiile pe „pachete”. Astfel, din pachetul *com.licenta.sebi.findmycar.utils* fac parte 6 clase fiecare având în componența lor mai multe funcții care se ocupă cu returnarea de informații necesare aplicației sau efectuarea unor acțiuni sau verificări pe un set de date dat ca și parametru.

5.3.1 ColorUtils

Prima clasă utilitară este *ColorUtils* care a fost implementată din punct de vedere a programării orientată pe obiecte ca un *Singleton*, instanța statică fiind *colorUtilsInstance*. Această clasă are în componența sa o variabilă de tip *HashMap<Character, Color>* în care se reține perechea formată de un caracter și o culoare de tip *Color* din cadrul bazei de date, pentru a putea prelua mai apoi, în funcție de textul unui loc de parcare, culoarea de fundal a simbolului total pentru a sugera utilizatorului în listă elementul pe care urmează să îl acceseze, precum și zona în care se află sau în care trebuie să meargă. Această clasă are o funcție numită *findColorByCharacter()* care primește un caracter ca și parametru și returnează culoarea corespunătoare acestuia:

```
public Color findColorByCharacter(Character character){
      return lettersToColorsMap.get(character);
}
```

5.3.2 DialogsUtils

A doua clasă care face parte din pachetul utilitar este *DialogsUtils* care se ocupă cu seria de *dialog*-uri care ar trebui afișate atunci când utilizatorul dorește salvarea sau căutarea unei locații din cadrul parcării. Astfel, această clasă are în componența sa 4 funcții și o interfață Java care este folosită pentru a returna textul identificat, precum și culoarea identificată la finalul procesului de căutare a locației:

```
public interface OnParkingSpotDeterminedCallback{
      void onParkingSpotDetermined(String text, Color color);
}
```

Cele 4 funcții ale acestei clase sunt *showMainDialog(), showSubmitManuallyDialog(), showConfirmationDialog()* și *showErrorDialog()*. Prima funcție se ocupă cu afișarea *dialog*-ului principal în care utilizatorul va trebui să aleagă între cele două metode de identificare a locului de parcare, și anume scanarea textului unui stâlp al parcării sau introducere manuală. În cazul în care utilizatorul alege introducerea manuală este apelată funcția *showSubmitManuallyDialog()* din cadrul aceleiași clase, iar în cazul în care utilizatorul alege opțiunea de a face o poză se apelează funcția *scanWithCamera()* din cadrul clasei *PhotoUtils*.

A doua funcție, și anume *showSubmitManuallyDialog()* se ocupă cu afișarea unui *dialog* în care se află un *View* de tip *EditText* în care utilizatorul poate introduce textul aflat pe stâlpul parcării, după care va trebui să apese pe butonul de „SUBMIT" pentru verificarea textului. Dacă textul are lungimea corectă se apelează funcția *showConfirmationDialog()* din cadrul aceleași clase, iar în caz contrar se apelează funcția *showErrorDialog()* tot din cadrul clasei *DialogsUtils.*

```
public static void showSubmitManuallyDialog(final AppCompatActivity activity){
       new MaterialDialog.Builder(activity)
              .title("Introduceți textul")
              .input("Ex: D40", "", false, new MaterialDialog.InputCallback(){
                     @Override
                     public void onInput(MaterialDialog dialog,
                                                          CharSequence input){
                            if(input.length() >= 3)
                                   showConfirmationDialog();
                            else
                                   showErrorDialog();
                     }
              }).show();
```

A treia funcție *showConfirmatonDialog()* se ocupă cu afișarea dialogului final în care utilizatorului i se prezintă textul identificat sau introdus, culoarea de fundal a acestuia și 3 opțiuni de unde poate alege, acestea fiind opțiunea de a introduce din nou manual, de a face din nou o poză sau de a confirma faptul că datele sunt corecte. La apăsarea butonului de confirmare se face verificarea locului de parcare și se apelează funcția *onParkingSpotDetermine()* daca locul de parcare este în regulă și a fost găsit cu succes, iar în caz contrar se apelează funcția *showErrorDialog()* care se ocupă cu afișarea unui dialog de eroare corespunzător erorii întâlnite pe parcursul procesului de identificare a locului de parcare sau a poziției utilizatorului.

5.3.3 DijkstraUtils

A treia clasă din pachetul de clase utilitare este clasa *DijkstraUtils* care se ocupă de rularea algoritmului Dijkstra de căutare a drumului de cost minim. Această clasă a fost implementată din punctul de vedere al programării oriantă pe obiecte ca un *Singleton*, instanța unică a acestei clase fiind reprezentată de variabila *dijkstraUtilsInstance* din cadrul clasei. Această clasă are două funcții care rulează asincron și două interfețe Java pentru a semnala finalizarea acțiunilor derulate în cadrul funcțiilor.

```
public interface OnRoadComputedCallback{
      void onRoadComputed(List<Vertex> road);
}
public interface OnVertexFoundCallback{
      void onVertexFound(Vertex vertex);
}
```

Prima funcție a acestei clase este *getRoad()* care se ocupă cu căutarea asincronă a drumului de cost minim între o sursă și o destinație, ambele fiind date ca și parametru, cu ajutorul algoritmului Dijkstra, iar după găsirea acestui drum se apelează funcția *onRoadComputed()* pentru a trimite lista de noduri callback-ului respectiv.

Cea de-a doua funcție a acestei clase este *getVertexByText()*, iar aceasta se ocupă cu gasirea nodului care are ca și etichetă textul trimis ca și parametru, iar în cazul în care s-a găsit nodul respectiv se va apela funcția *onVertexFound()* a callback-ului trimis ca și parametru de tip *OnVertexFoundCallback.*

5.3.4 PhotoUtils

Clasa utilitară *PhotoUtils* se ocupă cu operațiile care ar trebui efectuate cu ajutorul camerei dispozitivului. Această clasă are în componența sa 4 funcții și anume *scanWithCamera(), openCamera(), onActivityResult(), onRequestPermissionsResult()*. Prima funcție este funcția principală a acestei clase și se ocupă cu verificarea faptului că utilizatorul a aprobat permisiunea de folosire a camerei dispozitivului, continuând în acest caz cu apelarea funcției *openCamera()* din cadrul aceleiași clase, iar în caz contrar se apelează funcția *requestPermissions()* pentru a cere aprobarea utilizatorului:

```
public static void scanWithCamera(AppCompatActivity activity){
      if(ActivityCompat.checkselfPermission(activity,
      Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED){
            ActivityCompat.requestPermissions();
      }else{
            openCamera();
      }
}
```

Cererea aprobării permisiunilor apelează funcția *onRequestPermissionsResult()* din cadrul aplicației curente, iar în acel callback se apelează funcția *onRequestPermissionsResult()* din cadrul clasei *PhotoUtils* folosind variabilele primite ca și parametru pentru a stabili dacă utilizatrul a aprobat folosirea camerei sau nu.

Funcția *openCamera()* creează fișierului în care se va salva poza rezultată temporar și se ocupă cu deschiderea aplicației de cameră din cadrul dispozitivului cu ajutorul unui intent creat cu o acțiune standart și cu fișierul dat ca și informație extra:

```
private static void openCamera(AppCompatActivity activity){
      takenPhotoPath=newFile(Environment
            .getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES),
                                    „PHOTO_TAKEN.png”).getAbsolutePath();
      Uri outputFileUri = Uri.fromFile(new File(takenPhotoPath));

      Intent intent = new Intent(„android.media.action.IMAGE_CAPTURE”);
      intent.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);

      activity.startActivityForResult(intent, 0);
}
```

După efectuarea fotografiei se apelează funcția *onActivityResult()* din cadrul activității curente, iar în acel callback se apelează funcția *onActivityResult()* din cadrul clasei *PhotoUtils*. În această funcție se preia fișierul cu fotografia, se rezolvă rotirea imaginii , se convertește obiectul de tip *Bitmap* la configurația *ARGB_8888* după care se extrage textul din imagine și se afișează dialogul corespunzător.

5.3.5 SVGUtils

Clasa utilitară *SVGUtils* se ocupă cu tot ce ține de fișierele în format *.svg* cu nivelele parcării complexului Palas Iași. Aceasta are în componența sa numeroase funcții pentru a ajuta la desenarea hărților pe interfața grafică.

Un exemplu de astfel de funcție îl reprezintă *drawRoadOnSVG()* care este funcția principală a clasei *SVGUtils*. Această funcție primește ca și parametri textul sursei, textul destinației, *View*-ul pentru încărcarea fișierului *.svg* și *View*-ul de tip *LinearLayout* pentru crearea indiciilor care îl vor ghida pe utilizator. Textele sursei și ale destinației sunt folosite pentru a găsi nodurile din graful orientat salvat în baza de date, iar cu ajutorul acestei liste se modifică fișierul *.svg* corespunzător pentru a desena drumul pe hartă. După desenarea drumului se creează indiciile ajutătoare și se încarcă harta în *View*-ul corespunzător.

```
public static void drawRoadOnSVG(final Context, final String sourceText,
             final String destinationText, final SVGMapView svgMapView,
                    final LinearLayout linearLayout){
       //codul care se ocupă cu desenarea și încărcarea hărții
}
```

5.3.6 TesseractUtils

Ultima clasă utilitară este *TesseractUtils* care se ocupă cu instanțierea bibliotecii *tess-two* și care are în componența sa două funcții, dintre care una este pentru returnarea obiectului de tip *TessBaseAPI* din biblioteca specificată. În constructorul acestei clase se verifică dacă fișierul de antrenare pentru limba dată este prezent în memoria rezervată aplicației, iar dacă nu este prezent acesta este adus din directorul *assets* și pus într-un director cu numele *tessdata*(acest lucru se întâmplă doar prima dată când este rulată aplicația sau dacă au fost șterse toate fișierele scrise de această aplicație). Cu ajutorul acestei clase, toată aplicația are acces la aceeași instanță a obiectului de tip *TessBaseAPI*, acesta fiind folosit pentru a extrage textul dintr-o imagine obținută cu camera foto.

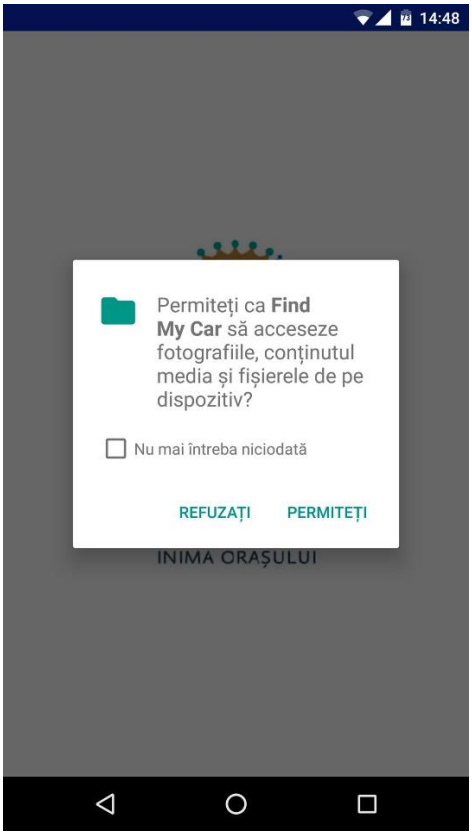
5.4 Interfața

Interfața în cadrul unei aplicații Android este definită în mare parte prin intermediul unor fișiere în format *.xml* și stocate în directorul */res/layout*. Fiecare activitate sau element al unei liste are nevoie de un astfel de fișier pentru a avea o interfață grafică. Astfel cele trei activități și elementele din lista care aparține ecranului principal au fiecare câte un fișier de acest tip. Secțiunea următoare de cod arată un exemplu de un astfel de fișier folosit pentru elementele din listă:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:id="@+id/root"
    android:paddingEnd="16dp"
    android:layout_marginBottom="8dp"
    android:paddingStart="16dp">

    <TextView
        android:id="@+id/tv_parking_spot_item_text"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:background="@color/colorPrimary"
        android:gravity="center"
        android:lineSpacingMultiplier="0.8"
        android:text="D\n40"
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/tv_parking_spot_item_pay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_toStartOf="@+id/iv_parking_spot_item_delete"
        android:text="~2RON"
        android:layout_marginEnd="6dp"
        android:textSize="20sp"
        android:textStyle="bold" />

    <ImageView
        android:id="@+id/iv_parking_spot_item_delete"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:contentDescription="@null"
        android:src="@drawable/ic_delete" />
</RelativeLayout>
```

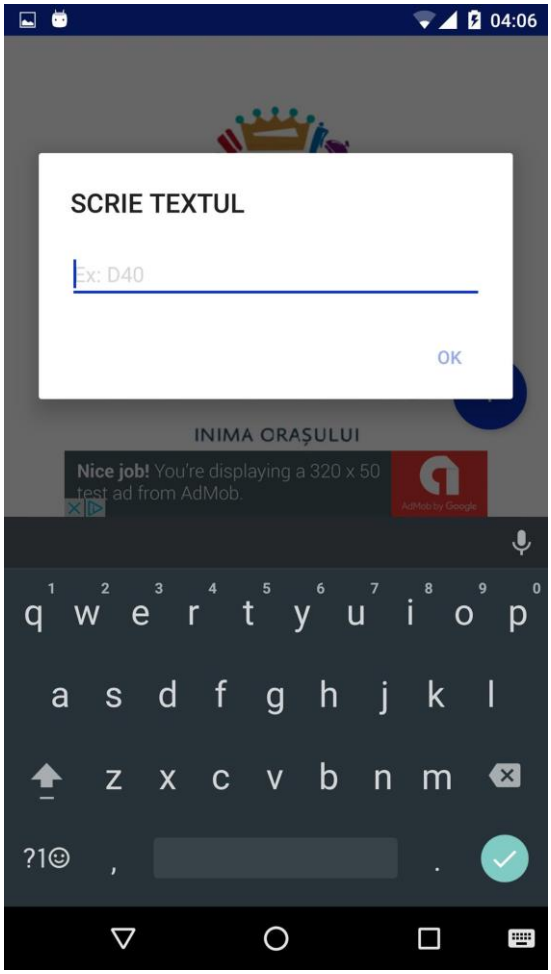
6. Manual de utilizare

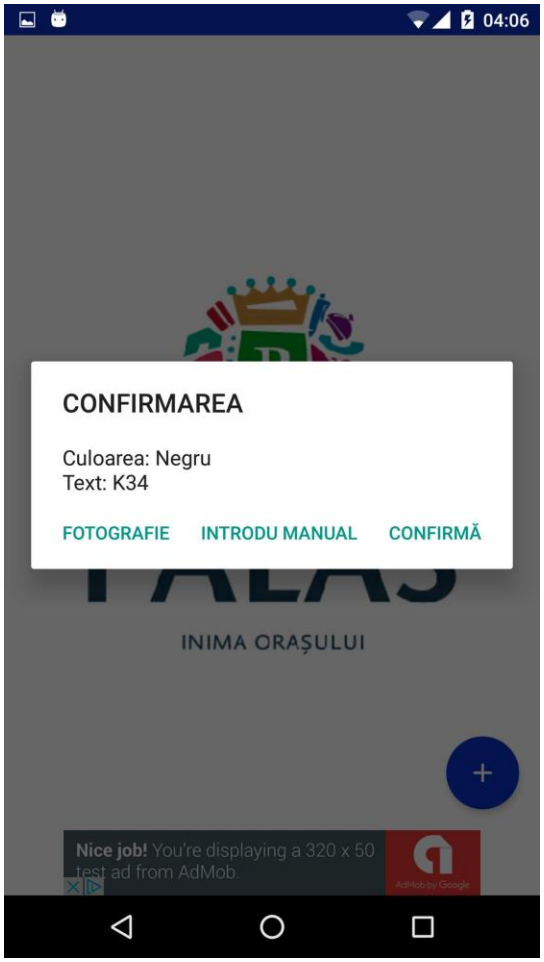
La rularea aplicației se afișează utilizatorului ecranul numit *Splash Screen* în care se află amplasată pe interfața grafică sigla complexului Palas Iași și în care se inițializează toate componentele aplicației. De asemenea în acest ecran se verifică dacă utilizatorul a aprobat permisiunea ca aplicația să poată scrie în spațiul rezervat acesteia, iar în funcție de caz de afișează un *dialog* în care se cere aprobarea permisiunii.



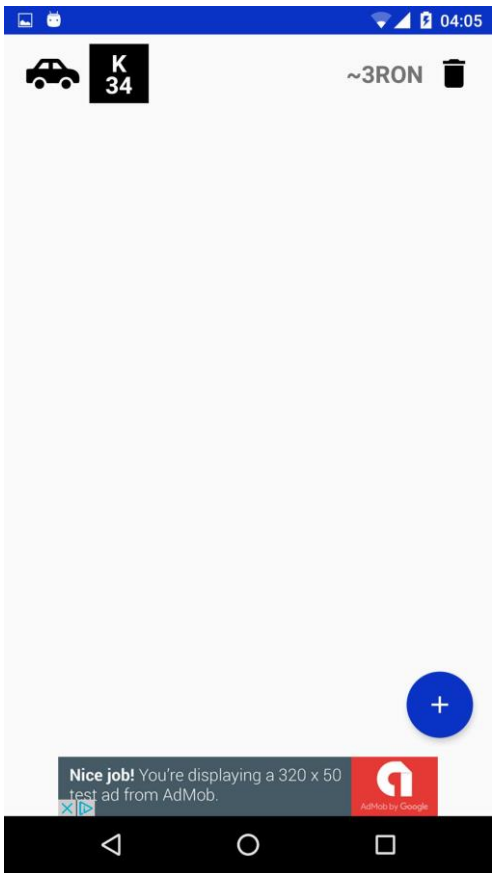
După efectuarea tuturor verificărilor i se afișează utilizatorului ecranul principal în care se află o reclamă la baza paginii, butonul de adăugare a unui loc de parcare, iar pe fundal sigla Palas Iași. La apăsarea butonului de adăugare a unui loc de parcare se deschide un *dialog* în care utilizatorul poate face alegerea metodei cu care va detecta locul de parcare. Astfel la apăsarea butonului „FOTOGRAFIAZĂ” se va deschide aplicația standard a dispozitivului folosit cu ajutorul căreia se va face o fotografie care va fi mai apoi analizată și extras textul din aceasta. Pentru ca aplicația Find My Car să poată deschidă aplicația de camera, utilizator va trebui să-și dea acordul pentru această permisiune, iar dacă acesta nu este de accord nu va putea folosi această funcționalitate.

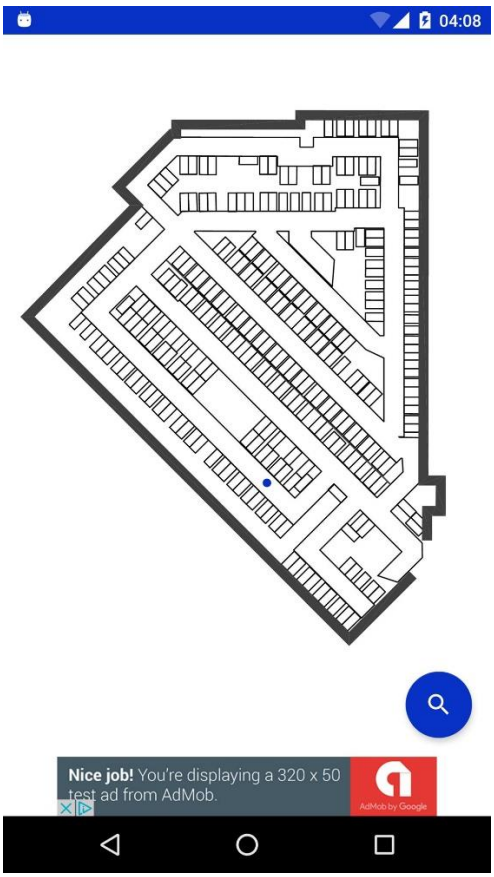
La apăsarea butonului „INTRODU MANUAL” i se va afișa utilizatorului *dialog*-ul în care se va putea introduce textul scris pe simbolul stâlpului, iar după apăsarea butonului „OK” se afișează *dialog*-ul de confirmarea locului de parcare în care utilizatorul poate confirma datele introduse sau extrase din poză, poate scana din nou prin intermediul unei poze sau poate reintroduce textul în cazul în care acesta este greșit.





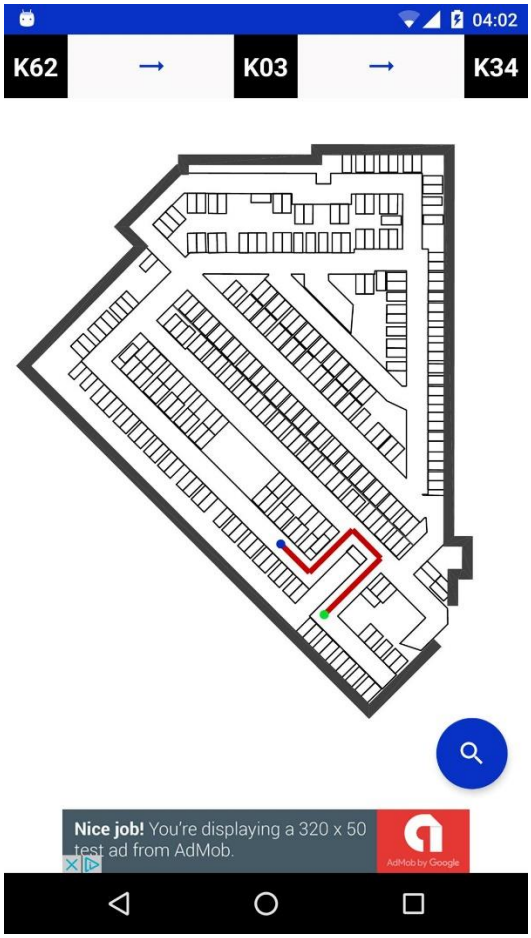
După confirmarea locului de parcare ecranul principal se modifică prin ascunderea siglei complexului Palas Iași și afișarea unei liste cu noul loc de parcare salvat. În cazul în care utilizatorul a salvat deja un loc de parcare și intră în aplicație, în ecranul principal nu va mai fi prezentă sigla Palas Iași aceasta fiind înlocuită de lista locurilor de parcare salvate. La selectarea unui element din listă utilizatorul va fi redirecționat către ecranul de „detalii” a locului de parcare selectat în care se află o reclamă la bază, un buton cu ajutorul căruia se va acționa căutarea poziției curente a utilizatorului și harta nivelului de parcare la care se află mașina acestuia cu un punct albastru reprezentând locația mașinii pe acel nivel.

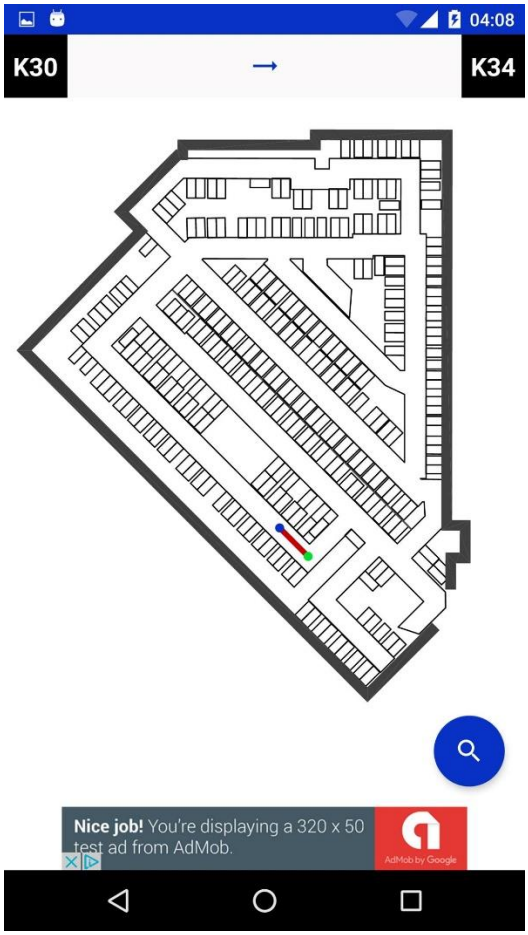




La apăsarea butonului de căutare a poziției curente se va deschide un *dialog* care va avea același comportament ca și cel din ecranul principal cu care se salva un loc de parcare. Utilizatorul va putea alege din aceleași opțiuni, adică scanarea prin intermediul unei fotografii sau introducerea manual a textului de pe stâlp.

După ce utilizatorul a urmat procesul de recunoaștere a poziției sale se construiește drumul cel mai scurt de la poziția sa până la mașina parcată precum și o serie de indicii ajutătoare cu care utilizatorul se va orienta vizual prin parcare. Poziția utilizatorului este reprezentată pe hartă cu un punct verde, poziția mașinii parcare cu un punct albastru, iar drumul este trasat cu roșu. Aici ramâne la latitudinea utilizatorului să apese indiciile oferite atunci când acesta ajunge în dreptul lor, acțiune ce va reface drumul pe hartă și care va ajuta la orientarea utilizatorului în cadrul parcării subterane.





Concluzii

Această lucrare descrie aplicația Find My Car care pune la dispoziția utilizatorilor săi o metodă mult mai accesibilă de orientare în cadrul parcării subterane a complexului Palas Iași. La moment orientarea este offline și se bazează mai mult pe indicii vizuale cum ar fi locurile de parcare pe lângă care va trece utilizatorul sau culoarea zonei în care se află sau în care va trebui să ajungă. Această aplicație suportă momentan două limbi diferite, Română și Engleză, care sunt detectate în funcție de limba setată în cadrul dispozitivului pe care îl folosește utilizatorul.

O posibilă direcție de dezvoltare a aplicației pe viitor ar fi amplasarea de *beacon*-uri pe toată suprafața parcării pentru a elimina orientarea pe bază de indicii vizuale și implementarea unui sistem asemănător dispozitivelor GPS. Astfel, utilizatorul se va deplasa într-o direcție, iar aplicația îi va indica acest lucru în timp real, eliminând confuzia direcției de mers și nevoia utilizatorului de a verifica harta în mod constant pentru a-și da seama în ce direcție ar trebui să se deplaseze.

Pentru a fi accesibilă și mai multor persoane care nu cunosc bine nici una dintre limbile suportate, s-ar putea include mai multe limbi în componența aplicației cum ar fi Franceză, Italiană, Germană, etc.

O actualizare interesantă al acestei aplicații, ar fi implementarea unui sistem care să afișeze pe hartă locurile ocupate și locurile libere din cadrul parcării. Astfel, utilizatorul va ști întotdeauna în ce parte a parcării sa se deplaseze pentru a-și parca mașina eliminând căutarile, uneori frustrante, a unui loc liber de parcare.

Bibliografie

[1] Smartphone OS Market Share, 2015Q2

http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[2] Android Platform Versions

https://developer.android.com/about/dashboards/index.html?hl=ko

[3] Android Studio

https://en.wikipedia.org/wiki/Android_Studio

[4] Gradle

https://en.wikipedia.org/wiki/Gradle

[5] The Android Project View

https://developer.android.com/studio/projects/index.html

[6] Optical Character Recognition

https://en.wikipedia.org/wiki/Optical_character_recognition

[7] Tess-two

https://github.com/rmtheis/tess-two

[8] Sugar ORM

http://satyan.github.io/sugar/index.html

[9] Joda-Time-Android

https://github.com/dlew/joda-time-android

[10] AdMob Wiki

https://en.wikipedia.org/wiki/AdMob

[11] AdMob

https://www.google.com/admob/

[12] Getting Started with Firebase in Android Studio

https://firebase.google.com/docs/admob/android/quick-start

[13] Android Design Support Library

http://android-developers.blogspot.ro/2015/05/android-design-support-library.html

[14] Material Dialogs

https://github.com/afollestad/material-dialogs

[15] Materialish Progress

https://github.com/pnikosis/materialish-progress

[16] Bluetooth Beacons

http://www.infoworld.com/article/2608498/mobile-apps/what-you-need-to-know-about-using-bluetooth-beacons.html

[17] Activity Life Cycle

https://developer.android.com/reference/android/app/Activity.html

[18] Dijkstra's Algorithm

http://www.vogella.com/tutorials/JavaAlgorithmsDijkstra/article.html