**PROIECT DISPOZITIVE ȘI APLICAȚII MOBILE**

**Student: Racoviță Iulia-Elena**

**Grupa: 1082**

**Seria: D**

***Cuprins***

1. **Introducere** 
   1. **Definire problemă**
   2. **Obiective**
   3. **Cadru general**
2. **Descriere soluție**
3. **Implementare soluție**
4. **Concluzie**
5. **Bibliografie**
6. **Introducere**
   1. **Definire problemă**

**În ziua de azi, atât autoturismul personal, cât și telefonul mobil au devenit bunuri aproape indispensabile vieții cotidiene. Deseori, nu observăm că o parte semnificativă din bugetul personal ajunge în diversele achiziții legate de mașina noastră. Eficiența și controlul asupra gestionării banilor devin de multe ori greu de obținut.**

**Altă problemă este posibilitatea apariției unei defecțiuni tehnice la autoturism în timp ce conducem. Cu atât mai neplăcut este atunci când suntem nevoiți să oprim mașina într-un oraș necunoscut și să contactăm în cel mai scurt timp ajutor specializat.**

**1.2. Obiective**

**Aplicația de față își propune să ofere utilizatorilor săi un mod prin care aceștia să poată gestiona rapid și eficient cheltuielile legate de autoturismul personal , cât și rapoarte în care șoferii pot observa exact ce sume au cheltuit și pe ce anume.**

**Rapoartele pot fi descărcate pe dispozitivul mobil și ulterior printate astfel încât șoferii pot avea un istoric al investițiilor legate de mașina personală. De asemenea, oferim clienților și o listă de contacte ale sediilor service (incluzând locația,mailul și numărul de telefon) din diferite orașe, astfel încât aceștia să aibă acces la ajutor specializat, în caz de nevoie.**

**Datele despre alimentarea cu combustibil sau cheltuielile realizate sunt stocate in baza noastra de date, iar utilizatorul poate accesa contul indiferent de dispozitivul mobil pe care îl folosește, prin intermediul mailului și al unei parole stabilite de acesta.**

* 1. **Cadrul general**

**Aplicația noastră poate fi folosită de șoferii care doresc să aibă un control mai bun asupra cheltuielilor legate de autoturismul personal. Interfața prietenoasă și ușor de utilizat face aplicația perfectă pentru toată lumea.**

1. **Descriere soluție – Ghid de utilizare al aplicației**

**Aplicația CarBuddy – Prietenul mașinii tale permite utilizatorilor permite conectarea, cât și înregistrarea datelor personale în baza de date.**

**Pentru utilizatorii noi punem la dispoziție formularul de *Înregistrare,* disponibil prin accesarea butonului „ÎNREGISTREAZĂ-TE” de pe ecran. Apăsând pe buton, noul utilizator va fi redirecționat către formular, unde va putea introduce date precum:**

* **Prenume**
* **Nume**
* **Email ( folosit ulterior pentru a se conecta )**
* **Parola**
* **Modelul autoturismului deținut**
* **Sex**

**În urma completării formularului, datele clientului vor fi salvate în baza noastră de date SQL prin apăsarea butonului cu textul „ÎNREGISTREAZĂ-TE.**

**Utilizatorii existenți se pot conecta prin formularul dedicat, apăsând butonul „CONECTEAZĂ-TE” . Datele de conectare cerute sunt mailul si parola .**

**Odată conectat, utilizatorul va putea vedea *Pagina Principală,* cu cele 4 butoane ale sale.**

1. **Butonul „Mai multe informații” – Acest buton va direcționa clientul către secțiunea de *About* și *Feedback* a aplicației. În *About* acesta se poate familiariza cu aplicația citind despre obiectivele noastre, iar în secțiunea *Feedback* poate adăuga un scor aplicației. De asemenea, tot în *Feedback***

**există butonul “DELETE MY ACCOUNT”, în situația în care acesta dorește să își șteargă contul. Oferim o listă cu contacte service denumită *Info Service* ( Poate fi văzută apăsând pe meniul din colțul dreapta – sus al ecranului). Lista pune la dispoziție date precum numele, locația, mailul și telefonul sediilor service din București.**

1. **Butonul “LOCAȚII SERVICE” – Utilizatorul poate vedea pe Maps locația sediilor service menționate anterior.**
2. **Butonul „Cheltuieli” – va deschide activitatea prin care clientul poate gestiona cu ușurință banii utilizați în diverse achiziții de bunuri/ servicii referitoare la autoturismul personal. În pagina *Cheltuieli* , utilizatorul poate fie să vadă cheltuielile dintr-un fișier JSON, extern, (accesând butonul “ARATĂ CHELTUIELI” ) fie să adauge în baza de date o nouă cheltuială. Acest lucru se poate realiza prin intermediul formularului *Adaugă Cheltuială* , completând câmpurile *Adaugă Cost* și *Adaugă Locație.* Lista de cheltuieli va apărea în activitatea anterioară, în *Istoric Cheltuieli.***
3. **Butonul “Alimentări” – deschide activitatea cu același nume, acesta la rândul ei oferind trei opțiuni/ butoane :**

* ***Adaugă Alimentare –* deschide formularul în care pot fi adăugate :**
* **Data alimentării**
* **Kilometraj**
* **Cost Total**
* **Litri**

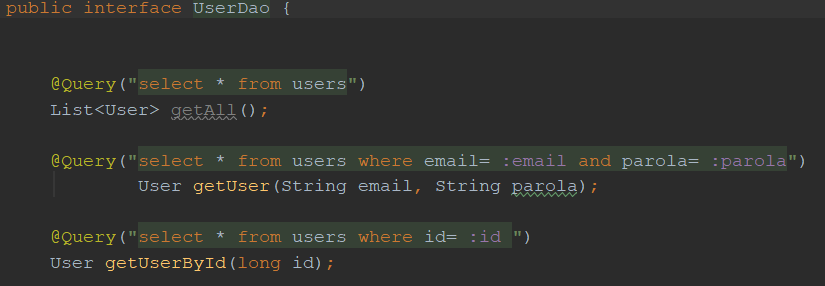
**Odată adăugată, o alimentare este afișată în activitatea alimentări – secțiunea *Istoric Alimentări.* Lista cu alimentări este unică pentru fiecare user.**

* ***Curăță Istoricul –*  Acest buton șterge lista de alimentări din baza de date și din aplicație.**
* ***Vizionează rapoarte* – Această opțiune oferă un control sporit al utilizatorului asupra alimentărilor făcute astfel încât acesta poate vedea exact în ce locație, la ce dată a făcut cele mai costisitoare alimentări, sau cu cantitatea cea mai mare de combustibil. Rapoartele pot fi salvate în format text pe dispozitivul mobil.**

1. **Implementare Soluție**

**Aplicația CarBuddy a fost dezvoltată în Android Studio. Interfața pentru fiecare activitate – mai exact, componentele vizuale a fost realizată prin intermediul fișierelor XML. Acestea se află în directorul res/layout. Imaginile folosite au fost adăugate în res/drawable, iar toate culorile, fonturile și string-urile se află în values ( colors.xml, fonts.xml, strings.xml)**

**Pentru activitățile LogIn și Register s-a folosit o bază de date SQL, în cadrul căreia am creat două tabele, Users – pentru stocarea datelor utilizatorilor și Refs – pentru stocarea datelor legate de alimentări. Operațiile DML utilizate pentru baza de date ( insert/ update/ delete ) se află în interfețele RefuelDao și UserDao din pachetul Dao.**

****

**Pentru o utilizare facilă a bazei de date, am creat clasa abstractă DatabaseManager. Ea face realizează conexiunile dintre aplicație și baza de date. În interiorul acesteia avem numele bazei noastre de date și metodele getInstance, getUserDao și getRefuelDao. Ultimele 2 metode fac legătura dintre baza de date și operațiile pe care le vom utiliza ulterior ( Insert, Update, Delete, Select - @Query ) .**

**public static DatabaseManager getInstance(  
 Context context) {  
 if (*databaseManager* == null) {  
 synchronized (DatabaseManager.class) {  
 if (*databaseManager* == null) {  
 *databaseManager* = Room  
 .*databaseBuilder*(context,  
 DatabaseManager.class,  
 *DB\_NAME*)  
 .fallbackToDestructiveMigration()  
 .build();  
 return *databaseManager*;  
 }  
 }  
 }  
 return *databaseManager*;  
}  
  
public abstract UserDao getUserDao();  
public abstract RefuelDao getRefuelDao();**

**În clasele User.class și Refuel.class sunt definite coloanele tabelelor, cheia primară, cheia secundară ( în cazul tabelei Refs ) . Tot acestea conțin și constructorii cu parametri, metodele get/set , etc. Legătura dintre User și Refuel este One-To-Many ( Un user poate avea mai multe alimentari, dar o alimentare poate fi efectuată de un singur User). Cheia de legătură dintre aceste 2 tabele este reprezentată de id-ul userului.**

**@Entity(tableName = "refs",  
 foreignKeys = @ForeignKey(entity = User.class,  
 parentColumns = "id",  
 childColumns = "userId",  
 onDelete = *CASCADE*))**

**onDelete=ForeignKey.CASCADE - șterge userul odată cu alimentările pe care acesta le-a realizat**

**În cadrul activității *LogIn* verificăm dacă metoda getUser, definită în interfața UserDao ( având ca și parametri mailul și parola introduse de utilizator) returnează un obiect nenul. În această situație, vom reține în fișierul de preferințe id-ul userului ( pentru a putea afișa mai apoi lista de alimentări pe baza userului înregistrat) și deschidem activitatea *Dashboard.***

**În *RegisterActivity* preluam informațiile din componentele vizuale și cream obiectul User prin metoda createUserFromView() . Vom insera acest obiect în baza de date utilizând altă metodă din UserDao, insert.**

**Pentru afișarea datelor pe ecran ( fie preluate din JSON, ca în cazul Istoric Cheltuieli, fie din baza de date SQL – Istoric Alimentări, sau pur și simplu afișarea listei de service-uri ) am folosit adaptori personalizați.**

**public AdapterRefuel(@NonNull Context context, int resource, @NonNull List<Refuel> refuels,LayoutInflater layout) {  
 super(context, resource, refuels);  
 this.context = context;  
 this.resource= resource;  
 this.refuels = refuels;  
 this.layout = layout;  
}  
  
@NonNull  
@Override  
public View getView(int position, @Nullable View convertView, @NonNull ViewGroup parent) {  
 View view = layout.inflate(resource, parent, false);  
 Refuel refuel = refuels.get(position);  
 if(view == null){  
 view = layout.inflate(R.layout.*refuel\_row*, null , true);  
 }**

**Aducerea informațiilor din JSON nu se face în firul principal de execuție, ci se va creea un fir secundar prin intermediul clasei abstracte AsyncTask. Folosind metoda doInBackground se creează o conexiune cu adresa json-ului nostru.**

**Pentru implementarea activității *Mai multe Informații* am folosit TabLayout. Această secțiune conține 2 fragmente, Feedback și About, permite utilizatorului să navigheze cu scroll între ele prin intermediul ViewPager-ului. În fragmentul Feedback, utilizatorul poate adăuga un rating aplicației. Pentru a menține scorul userului la fiecare deschidere a aplicației, am utilizat un fișier de preferințe care reține numărul de stele acordat.**

**În activitatea RefuelHistory afișăm lista de alimentări în funcție de userul logat . Această listă o preluăm din baza de date utilizând query-ul :**

**@Query("select \* from refs where userId= :userId ")  
List<Refuel> getRefuel(long userId);**

**Lista de obiecte returnată se afișează într-un adaptor personalizat. Pentru opțiunea Curăță Istoricul vom utiliza operația Delete, din interfața RefuelDao**

**@Delete  
void deleteAllRefuels(List<Refuel> refs);**

**buttonClear.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 refuelDao.deleteAllRefuels(refuels);  
 }  
});**

**Pentru vizionarea rapoartelor ( afișarea realimentărilor în funcție de cost / litri ) vom folosi query-urile din RefuelDao. Lista de obiecte va fi afișată pe ecran cu ajutorul unui ArrayAdapter. Aceeași listă va fi scrisă într-un fișier text ( care va fi creat în cazul în care nu există – utilizăm FileOutputStream ), utilizând clasa OutputStreamWriter, mai exact, apelând metoda write() .**

**Pentru introducerea, respectiv preluarea din firebase se va utiliza clasa FirebaseDatabase, care face conexiunea între aplicație și firebase și asigură realizarea operatiilor de insert, update sau delete din baza de date. Inserarea și update-ul în baza de date se face prin metoda push() , iar pentru stergere folosim removeValue().**

**În activitatea Maps am creat un obiect Location ce va conține latitudinea, longitudinea și numele service-ului reprezentat pe hartă.**

**Pentru fiecare locație creată se preiau coordonatele ( latitudinea și longitudinea ) și se adaugă un marker pentru a putea fi vizualizate pe hartă.**

@Override  
public void onMapReady(GoogleMap googleMap) {  
 mMap = googleMap;  
 drawCities();  
  
}  
  
private void drawCities() {  
 List<Location> list = new ArrayList<>();  
 list.add(new Location(45.037275, 23.278774, "Targu-Jiu Service"));  
 list.add(new Location(44.476014, 26.057462, "AutoErebus"));  
 list.add(new Location(44.444870, 26.133819, "AutoK9"));  
 list.add(new Location(44.416023, 26.090801, "ContactAutoService"));  
 for (Location location : list) {  
 // Add a marker in Sydney and move the camera  
 LatLng sydney = new LatLng(location.getLatitude(), location.getLongitude());  
 Marker marker = mMap.addMarker(new MarkerOptions()  
 .position(sydney)  
 .icon(bitmapDescriptorFromVector(getApplicationContext(), R.drawable.*carmipmap*))  
 .title(location.getServiceName()));  
  
 marker.showInfoWindow();  
 }  
}

1. **Concluzie**

**În concluzie, CarBuddy – Prietenul mașinii tale este necesar în viața de zi cu zi a șoferilor pentru a gestiona mai bine bugetul, dar și pentru a putea contacta rapid ajutorul în situații critice. Programul Android Studio pune la dispoziție mediul și instrumentele necesare dezvoltării unei astfel de aplicații**

1. **Bibliografie**
2. [**https://medium.com/androiddevelopers/database-relations-with-room-544ab95e4542**](https://medium.com/androiddevelopers/database-relations-with-room-544ab95e4542)
3. [**https://www.survivingwithandroid.com/android-async-listview-jee-and-restful/**](https://www.survivingwithandroid.com/android-async-listview-jee-and-restful/)
4. **https://developers.google.com/maps/documentation/android-sdk/map-with-marker**