Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Edin Beganović

Mobilna aplikacija za hitrejšo dostavo pošiljk

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Ljubljana, 2018

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Edin Beganović

Mobilna aplikacija za hitrejšo dostavo pošiljk

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V času, ko prodaja izdelkov preko interneta raste, je hitra dostava pošiljk vse bolj aktualna. Analizirajte proces hitre dostave pošiljk. Pri tem se fokusirajte na odkrivanje tistih elementov, kjer bo uvedba mobilne aplikacije pozitivno vplivala na večjo učinkovitost procesa. Na podlagi ugotovitev analize zasnujte mobilno aplikacijo in jo tudi razvijte.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Edin Beganović, z vpisno številko 63090348, sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za hitrejšo dostavo pošiljk

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Roka Rupnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 09. marca 2018 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Roku Rupniku za strokovno pomoč pri izdelavi diplomskega dela. Prav tako se zahvaljujem svoji družini, punci in vsem, ki so me spodbujali in podpirali v času izdelave diplomskega dela.

”Do not lose hope nor be sad.”

Kazalo

Slike

Seznam uporabljenih kratic

kratica	angleško	slovensko
ADT	Android Development Tools	Razvojna orodja za Android
API	Application Programming Interface	Aplikacijski programski vmesnik
GPS	Global Positioning System	Globalni sistem pozicioniranja
GUI	Graphical User Interface	Grafični uporabniški vmesnik
IDE	Integrated Development Environment	Integrirano razvojno okolje
SMS	Short Message Service	Kratko sporočilo
SQL	Structured Query Language	Strukturiran povpraševalni jezik za delo s podatkovnimi bazami
UML	Unified Modeling Language	Poenoten vzorčni jezik
URL	Uniform Resource Locator	Enolični krajevnik vira
XML	eXtensible Markup Language	Razširljiv označevalni jezik

Povzetek

Naslov: Mobilna aplikacija za hitrejšo dostavo pošiljk

Avtor: Edin Beganović

V okviru diplomskega dela je bila razvita mobilna aplikacija za hitrejšo dostavo pošiljk v kurirskih službah. Zaradi pomanjkljive tehnološke podpore so procesi dostave pošiljk na željene naslove časovno potratni. V današnjem življenjskem slogu in visoki zahtevnosti strank je za dobro in uspešno poslovanje podjetja potrebno, da so storitve opravljene hitro, učinkovito in brez nepotrebnih napak. Z razvojem aplikacije, ki je opisana v diplomskem delu, smo se posebej posvetili reševanju omenjene problematike. Aplikacija omogoča hiter pregled lokacij, posledično hitrejši transport in nenazadnje tudi lažjo komunikacijo s stranko. Aplikacija torej v prvi vrsti omogoča večjo hitrost transporta, večjo preglednost in vpogled v statistiko delovne uspešnosti uporabnika, brez nakupa dodatne strojne opreme.

Ključne besede: mobilna aplikacija, pošiljka, dostava, kurir, Google Maps.

Abstract

Title: Mobile application for faster shipment delivery

Author: Edin Beganović

This diploma thesis focuses on the development of a mobile application for faster shipment delivery in courier services. Due to lack of technological equipment, the processes of delivering shipments to the desired addresses are time consuming for the courier. In today's lifestyle and high customer complexity, it is necessary for a good and successful business of this kind to provide services quickly, efficiently and without unnecessary errors. With the development of the application, which is described in the following diploma thesis, we devoted special attention to solving this problem. The application allows quick tracking of client addresses, consequently faster transport and last but not least, easier communication with the customer. Foremost, the application enables faster transport speeds, greater transparency, and easier traceability of work efficiency, without the purchase of additional hardware.

Keywords: mobile application, package, delivery, courier, Google Maps.

Poglavje 1

Uvod

1.1 Motivacija za aplikacijo

Ob naročilu paketov iz različnih kurirskih služb sem srečal različne kurirje. Kako poteka njihov delovni dan in kakšne težave imajo mi nikoli ni vzbujalo posebne pozornosti, dokler se moj prijatelj ni zaposlil pri kurirski službi. Med pogovorom mi je razložil potek dneva kurirja. Zanimanje mi je vzbudilo dejstvo, da ob iskanju naslova stranke vsakič vpišejo podatke stranke v navigacijo. Odločil sem se, da bom znanje, ki sem ga pridobil pri študiju, uporabil za razvoj rešitve njihove težave in jim s tem olajšal delo.

1.2 Cilji

Primarni cilj aplikacije je bil narediti mobilno aplikacijo, ki bi omogočila kurirju lažji prikaz podatkov na zemljevidu, hitrejšo dostavo pošiljke in posledično hitreje opravljeno delo. Sekundarni cilj je bil omogočiti lažjo komunikacijo s stranko preko direktnega klica, SMSa in elektronske pošte. Terciarni cilj je bil razvoj dodatnih funkcionalnosti, kot so uvoz datoteke XML, izpis statistike kurirja za posamezni delovni dan in pošiljanje podatkov nazaj kurirski službi.

1.3 Pregled področja podobnih aplikacij

Podobnih aplikacij, kot je aplikacija v diplomskem delu, ni v prosti uporabi ampak jih imajo podjetja za izključno interno uporabo. Aplikacije, ki so podobne naši aplikaciji so večinoma v lasti podjetij, ki se ukvarjajo s kurirskim delom.

Ob pregledu Google Play trgovine smo našli aplikacije kot jih imajo podjetja Pošta Slovenije, Fedex, DPD, GLS itd. Omenjene aplikacije so namenjene strankam za prosto uporabo in omogočajo funkcionalnosti kot so pošiljanje pošiljk, iskanje najbližjih poslovalnic ter sledenje pošiljki. Takšne funkcionalnosti niso v skladu z naši zastavljeni cilji.

1.4 Struktura diplomske naloge

V prvem, že prebranem poglavju so predstavljeni uvod v diplomsko nalogo, motivacija, cilji, ki smo si jih zadali ob začetku izdelave diplomske naloge, ter pregled področja podobnih mobilnih aplikacij.

V drugem poglavju je opisano področje dostave pošiljk, kdo jih dostavlja in na kakšen način.

V tretjem poglavju so opisane tehnologije in orodja, ki smo jih uporabili med razvojem aplikacije.

V četrtem poglavju je opisana analiza in načrt delovanja aplikacije.

V petem poglavju je podrobno opisan razvoj aplikacije.

V šestem poglavju je podrobno opisano delovanje aplikacije.

V zadnjem poglavju so predstavljene ideje za možne nadgradnje mobilne aplikacije.

Poglavje 2

Opis področja dostave pošiljk

2.1 Kaj so pošiljke

Pošiljke so objekti, ki se prevažajo preko kurirskih ali poštnih služb od pošiljateljev do prejemnikov [9].

Slika 2.1: Vrste pošiljk.

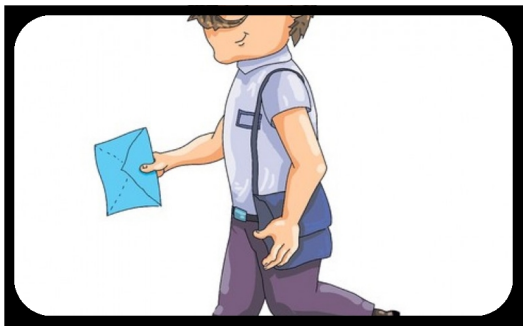
Vrste pošiljk:

- Pisemske pošiljke:
  - standardizirano pismo,
  - sodna pisma,
  - dopisnica,

  - tiskovina,
  - odtisi za slepe.
- Paket.
- Poštna in telegrafska nakaznica.
- Telegrafsko sporočilo.

2.2 Kdo je kurir

Kurir je oseba, ki dostavlja in prevzema različne vrste pošiljk. Naloga kurirja je, da pošiljko čim hitreje in nemoteno transportira iz enega v drug kraj in pri tem pazi, da pošiljke ne poškoduje, izgubi ali zameša. Pravilo je, da mora poslano pošiljko dostaviti v takem stanju, kot jo je prejel. Kurir ob osnovnem delu skrbi za urejenost prevoznega sredstva in njegovo najavo za morebiten servis [8].



Slika 2.2: Slika kurirja.

2.3 Opis procesa dostave pošiljk

Proces dostave pošiljk poteka tako, da kurir pri stranki prevzame pošiljko in jo prenese v servis kurirske službe, kjer jo ustrezno označijo ter dokumentirajo. Kurirska služba ima organizirano verigo kurirjev, ki med seboj sodelujejo in spremljajo pošiljko pri transportu po kopnem, vodi ali zraku. Strankam zagotavlja tudi nekatere druge storitve, kot na primer: sledenje

pošiljki med transportom, nudenje informacij o cenah in storitvah, ki jih nudi kurirska služba, dostava embalaže za pošiljke, ureditev carinske dokumentacije in drugo. Ko se pošiljka ustrezno označi in dokumentira, jo kurir prevzame in dostavi na želeni naslov [8].

2.4 Težava pri dostavi in rešitev težave

Težava pri dostavi nastane, ko kurir prevzame vse pošiljke od kurirske službe. Vse pošiljke namreč mora posebej pogledati in jih najti na seznamu. Iz seznama mora prebrati naslov vsake stranke, naslov ročno vpisati v navigacijo in sam odrejati kje začeti dostavljati pošiljke. Ta proces je časovno zelo potraten.

Težavo odpravimo z mobilno aplikacijo, tako, da dobi kurir seznam pošiljk od kurirske službe, že naložen v mobilno aplikacijo. Mobilna aplikacija se poveže s strežnikom in naloži datoteko XML, ki vsebuje podatke o vseh strankah. Omenjene podatke prikaže na zemljevidu in kurirju omogoča izbiro najbližje stranke.

Poglavje 3

Uporabljene naprave in programska oprema

3.1 Mobilna naprava Samsung Galaxy Note 8

Na sliki 3.1 je mobilna naprava, ki smo jo uporabili za testiranje aplikacije. To je največji zaslon na napravah Note doslej in omogočil nam je odličen prikaz aplikacije na napravi [10].

Slika 3.1: Mobilna naprava Samsung Galaxy Note 8.

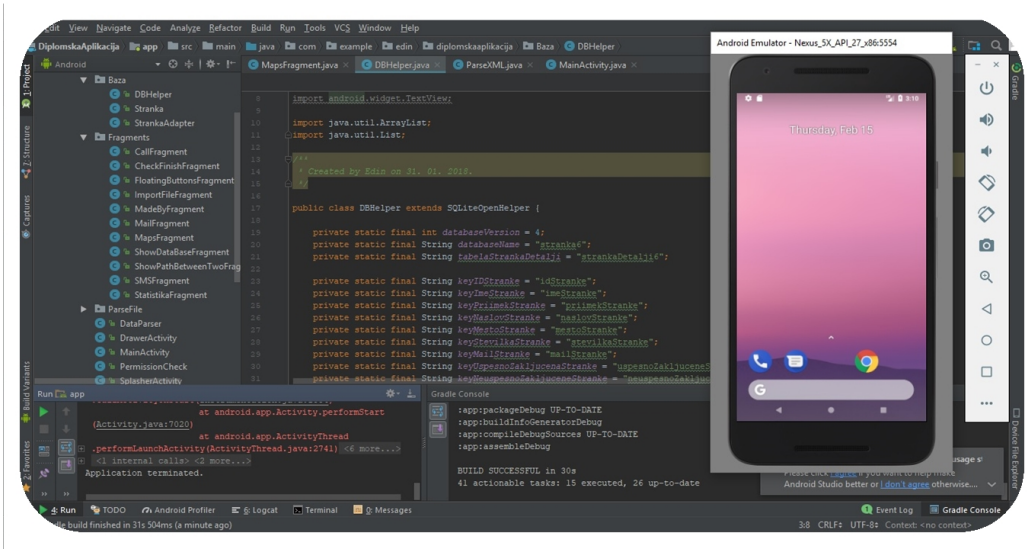
3.2 Operacijski sistem Android

Android je mobilni operacijski sistem, ki ga je razvil Google. Temelji na spremenjeni različici jedra Linuxa in druge odprtokodne programske opreme in je zasnovan predvsem za mobilne naprave na dotik, kot so pametni telefoni in tablični računalniki [2]. Poleg tega je Google dodatno razvil Android TV za televizorje, Android Auto za avtomobile in Android Wear za zapestne ure, od katerih ima vsak poseben uporabniški vmesnik. Različice Androida se pojavljajo tudi na igralnih konzolah, digitalnih fotoaparatih, osebnih računalnikih in drugi elektroniki. Operacijski sistem Android je dosegel več množičnih izdaj, pri čemer je bila trenutna različica 8.1 "Oreo", objavljena decembra 2017.

Android je od leta 2011 najbolj prodajan OS po vsem svetu na pametnih telefonih, na tabličnih računalnikih pa od leta 2013. Od maja 2017 ima več kot dve milijardi mesečno aktivnih uporabnikov in Google Play trgovina shranjuje več kot 3,5 milijona aplikacij.

3.3 Android studio

Android Studio je uradno integrirano razvojno okolje (ang. IDE) za Googlov operacijski sistem Android, ki temelji na programski opremi JetBrains in je zasnovan posebej za razvoj Androida [3]. Na voljo je za prenos na operacijskih sistemih Windows, MacOS in Linux. Android Studio je brezplačen in na voljo vsem uporabnikom. Služi kot zamenjava za razvojno okolje Eclipse, saj nudi podporo za razvojna orodja Android in ima preglednejši uporabniški vmesnik. Android Studio je bil objavljen leta 2013 na Googlovi konferenci. Prva stabilna zgradba je bila objavljena decembra 2014, začenši z različico 1.0. Sedanja stabilna različica je 3.0 objavljena oktobra 2017. Razvojno okolje Android Studio smo uporabili za programiranje naše mobilne aplikacije.



Slika 3.2: Razvojno okolje Android Studio.

3.3.1 Java

Java je objektno usmerjeni, prenosljivi programski jezik, ki ga je razvil James Gosling s sodelavci v podjetju Sun Microsystems [6]. Projekt, ki se je v začetku imenoval Oak (hrast), je bil razvit kot zamenjava za C++. Jave ne smemo zamenjevati z jezikom JavaScript, ki ima podobno ime, ter podobno, Cjevsko skladnjo. Javo vzdržuje in posodablja Oracle.

Poznamo 3 vrste jave:

- J2SE - standardna različica jave za osebne računalnike.
- J2ME - različica jave za mini naprave.
- J2EE - poslovna različica jave.

Z javo se da programirati tudi aplikacije na mobilnih telefonih (J2ME) in pametnih telefonih z operacijskim sistemom Android.

3.4 Google Maps API

Google je junija 2005 začel uporabljati Google Maps API, kot je razvidno na sliki 3.3, da bi razvijalcem omogočil integracijo Google Maps na njihove

Slika 3.3: Primer mobilne verzije Google Maps API.

spletne strani. Storitev je brezplačna in trenutno ne vsebuje oglasov, vendar Google v pogojih uporabe navaja, da si pridržuje pravico do prikazovanja oglasov v prihodnosti. Čeprav je bil na začetku samo API za programski jezik JavaScript, je bil API razširjen tako, da je vključeval storitev za pridobivanje statičnih slik zemljevida in spletnih storitev za izvajanje geokodiranja. Več kot milijon spletnih strani uporablja API Google Maps, zaradi česar je to najbolj uporaben API za razvoj spletnih aplikacij. Google Maps API je brezplačen za komercialno uporabo, pod pogoji, da je spletno mesto, na katerem se uporablja, javno dostopno in ne zaračunava dostopa ter ne ustvarja več kot 25000 dostopov do zemljevida na dan. Spletna mesta, ki ne izpolnjujejo teh zahtev, lahko kupijo Google Maps API za podjetja. Za uporabo APIja moramo najprej pridobiti brezplačen ključ za API, s čimer Google beleži uporabo svojih storitev. Google Maps API smo uporabili za

prikaz strank na zemljevidu [5].

3.5 SQLite

SQLite je vgrajena SQL baza podatkov. Za razliko od večine drugih zbirk podatkov SQL, SQLite ne potrebuje ločenega podatkovnega strežnika, saj je sestavni del aplikacije. Celotna zbirka SQL z več tabelami, indeksi, sprožilci in pogledi je v eni datoteki diska. Datoteko baze podatkov lahko prosto kopirate med 32-bitnimi in 64-bitnimi sistemi. Te funkcije omogočajo, da je SQLite priljubljena izbira kot oblika datotečnega programa.

```
blic class DatabaseHelper  extends SQLiteOpenHelper{
    private final static int DATABASE_VERSION = 1;
    private final static String DATABASE_NAME = "Rcontacts.db";
    private final static String TABLE_NAME = "Rcontacts";
    private final static String COLUMN_ID = "id";
    private final static String COLUMN_NAME = "name";
    private final static String COLUMN_USER = "user";
    private final static String COLUMN_PASS = "pass";
    SQLiteDatabase db;
    private final String TABLE_CREATE = "create table "+TABLE_NAME +
            "(id integer primary key not null , " +
            "name text not null , user text not null , pass text not null)";// , phone text not null)";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);}
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(TABLE_CREATE);
        this.db=db;}
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        String query = "DROP TABLE IF EXISTS" + TABLE_NAME;
        db.execSQL(query);
        this.onCreate(db);
    }
    public void insertContact(RContact c){...}
    public String searchPass(String uname){
        db = this.getReadableDatabase();
        String query = "select user and pass from "+TABLE_NAME;
        Cursor cursor = db.rawQuery(query, null);
        String u ,p;
        p = "not found";
        if(cursor.moveToFirst()){
            do{
                u = cursor.getString(0);
                Log.i("username: ", u);
                if(u.equals(uname)){
                    p = cursor.getString(1);
                    Log.i("entered if statement", "p: " + p);
                    break;
                }
            }while(cursor.moveToNext());
        }
        cursor.close();
```

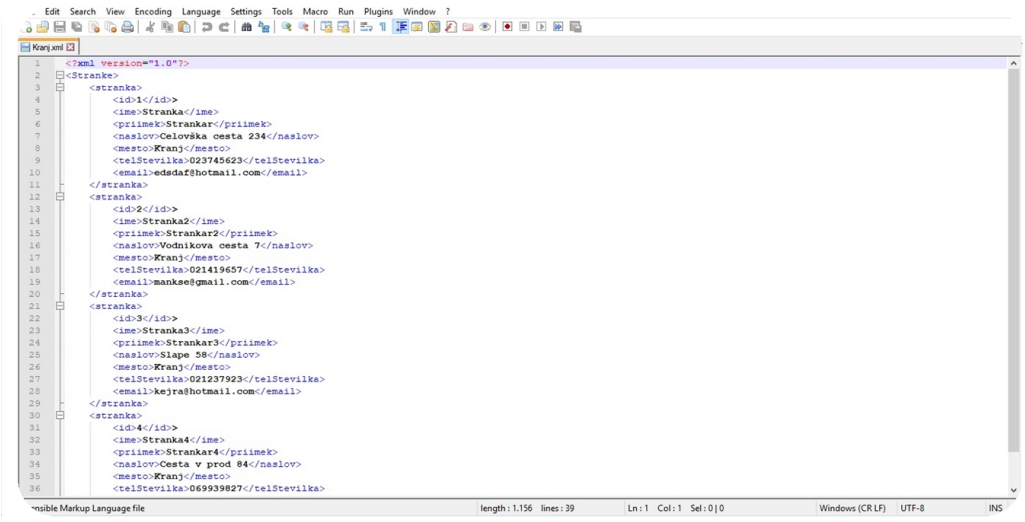
Slika 3.4: Primer kode za SQLiteOpenHelper.

Pomislite, da SQLite ni nadomestilo za Oracle podatkovne baze, temveč

kot nadomestilo za funkcije za odpiranje datotek. Razred SQLliteOpenHelper (slika 3.4) smo uporabili za kreiranje naše lokalne podatkovne baze [11].

3.6 Notepad++

Notepad++ (slika 3.5) je urejevalnik kode. Ponuja poudarjanje sintakse, prepogibanje kode in omejeno samodokončanje za programske, skriptne in označevalne jezike, ne pa tudi inteligentnega zaključevanja kode ali preverjanja sintakse. Kot tak lahko pravilno označuje kodo, zapisano v podprti shemi, vendar ni mogoče preveriti sintakse. Notepad++ označuje skladenjske elemente raznovrstnih jezikov, kot so Java, LaTex, XML. Notepad++ urejevalnik smo uporabili za sestavljanje XML datotek, katere smo uvozili v mobilno aplikacijo [7].



Slika 3.5: Urejevalnik kode Notepad++.

Poglavje 4

Analiza in načrt aplikacije

4.1 Analiza

4.1.1 Diagram primerov uporabe

Na sliki 4.1 je prikazan UML diagram primera uporabe mobilne aplikacije za hitrejšo dostavo pošiljk. Akter je v tem primeru kurir, ki izvaja določene akcije, katere ga pripeljejo do potrebnih rezultatov.

Ko kurir požene aplikacijo se mu prikaže izbira lokacije dostavljanja. Po izbrani lokaciji, se aplikacija v ozadju poveže na strežnik in prevzame datoteko XML. Podatki se shranijo v podatkovno bazo v mobilni aplikaciji. V primeru, da povezava ne uspe, mora kurir ročno uvoziti datoteko XML, ki vsebuje seznam dostav.

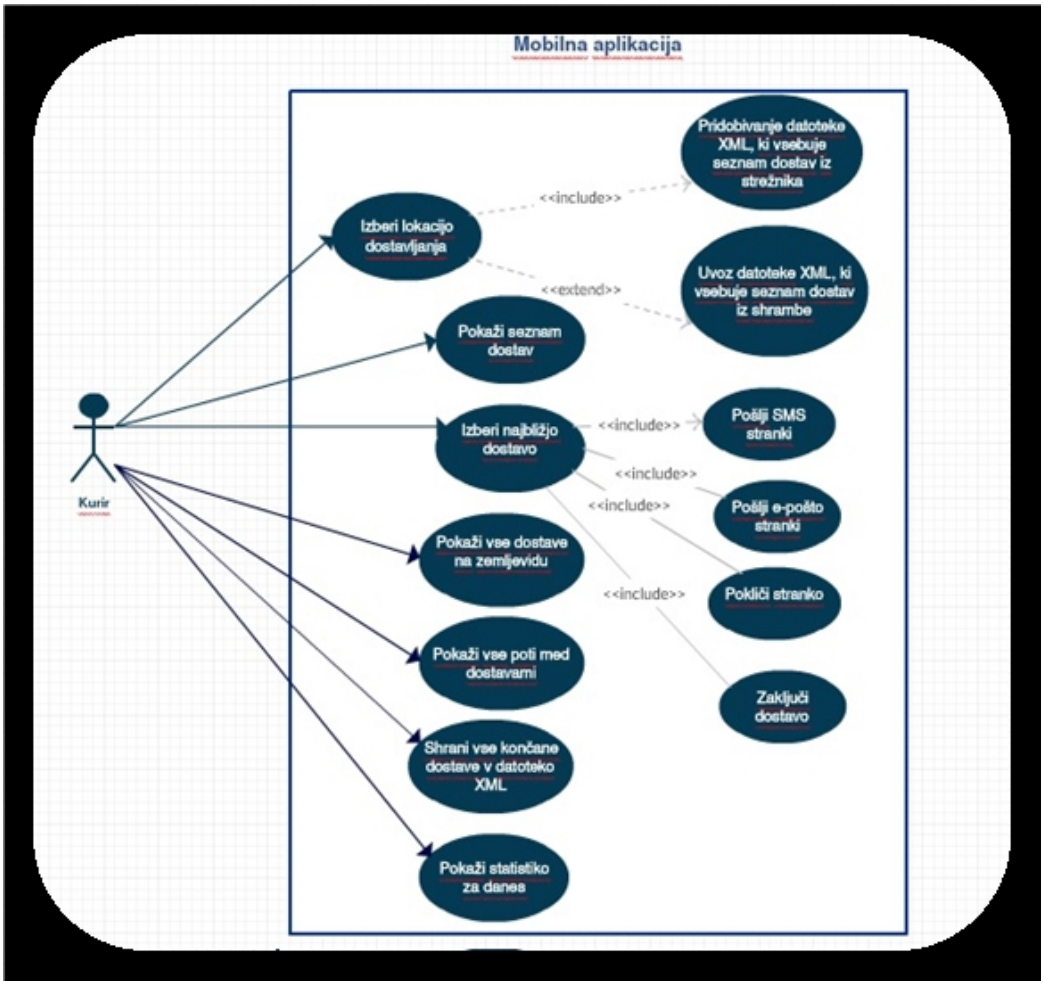
Kurir ima nato možnost izbire drugih akcij. Lahko pogleda seznam vseh dostav. Lahko si prikaže vse dostave na zemljevidu. Lahko pa pokaže vse poti med dostavami.

Kurir ima nato možnost izbire najbližje dostave, ki mu izriše najbližjo pot do stranke na zemljevidu. Ob izbrani najbližji dostavi dobi možnosti, kot so pošiljanje kratkega sporočila, pošiljanje elektronske pošte in klicanje te stranke. Pri trenutni dostavi ima še možnost zaključitve dostave.

Ko opravi vse dostave, lahko podatke o uspešno in neuspešno dostavljenih

pošiljkah shrani v datoteko XML in pošlje na strežnik.

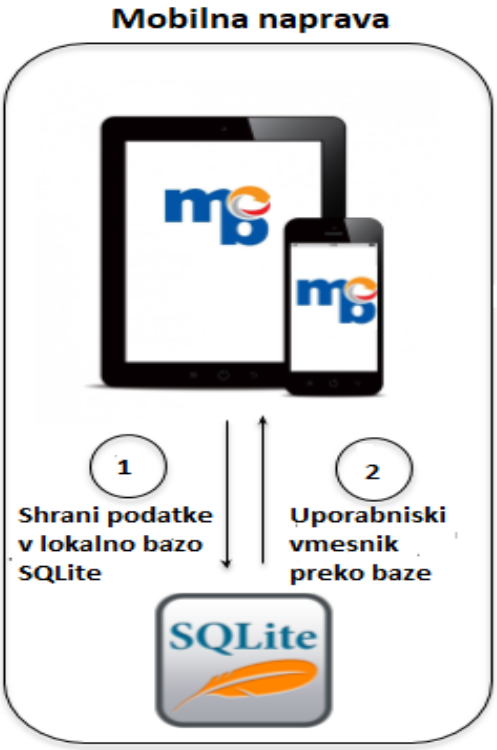
Kurir ima tudi možnost vpogleda v statistiko končanih dostav.



Slika 4.1: Diagram primerov uporabe.

4.1.2 Arhitekturni diagram

Na sliki 4.2, smo prikazali kako poteka povezava med komponentami mobilne aplikacije.



Slika 4.2: Arhitekturni diagram.

4.2 Načrtovanje dizajna aplikacije

Samega izgleda aplikacije si v začetku razvijanja, nismo predstavljali. Načrtovanja smo se lotili z risanjem grafičnega vmesnika na papirju. Skicirali smo vse možne zaslone. Pri skiciranju smo ugotovili, da hočemo čimbolj minimalističen izgled.

Hoteli smo, da ima aplikacija tudi prepoznavno ikono in logotip ter smo slednja skrbno izbrali.

Ko smo določili število zaslonskih mask, smo tudi identificirali funkcionalnosti za vsako zaslonsko masko.

Na koncu smo določili še barve za grafične komponente, ki so se uporabljale za funkcionalnosti.

Poglavje 5

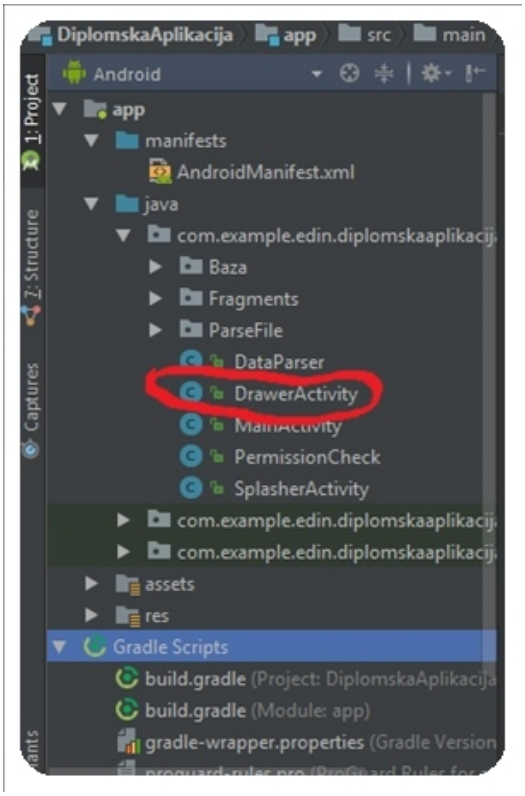
Razvoj aplikacije

5.1 Mobilna aplikacija

Ob končanem načrtovanju, smo se lotili razvoja aplikacije. Mobilna aplikacija je namenjena kurirjem, ki opravljajo terensko delo. Zaradi tega mora imeti aplikacija omogočen dostop do interneta preko mobilnih podatkov ali preko brezžične tehnologije, da se lahko vzpostavi povezava s strežnikom. Aplikacijo ni možno uporabljati brez interneta, ker moramo na začetku vzpostaviti povezavo s strežnikom, za prevzem datoteke XML z vsebino, ki jo uporabljamo v aplikaciji. V primeru, da povezava s strežnikom ni možna, prejme kurir datoteko XML po elektronski pošti, zakar je prav tako potreben internetni dostop. Datoteko XML prevzame iz elektronske pošte in jo shrani v shrambo. Iz shrambe jo kasneje uvozi v aplikacijo. Internetno povezavo potrebujemo tudi v primeru Google Maps API, ki nam omogoča izris zemljevidov. Nenazadnje pa je kurirju potrebna povezava z internetom ob zaključku izmene, da lahko pošlje rezultate svojega dela nazaj na strežnik.

5.2 Začetek projekta

V Android Studiu smo uporabili predlogo začetnega projekta za mobilne aplikacije s predlogo aktivnosti navigacijskega predala (ang. Navigation Drawer



Slika 5.1: Začetni projekt v Android Studiu.

Activity). Tako je bil avtomatsko ustvarjen del izvorne kode za zagon aplikacije (slika 5.1).

5.3 Opis aktivnosti mobilne aplikacije

Aktivnosti so eden od temeljnih gradnikov aplikacij pri platformi Android. Aktivnost predstavlja zaslonsko stran v aplikaciji Android. Aplikacija je običajno sestavljena iz več aktivnosti. Aktivnosti vsebujejo elemente grafičnega uporabniškega vmesnika (ang. GUI), kot so gumbi, seznami ali polja za vnos besedil. Tehnično je vsaka aktivnost primerek razreda android.app.Activity. V posameznem razredu je treba definirati življenjski cikel aktivnosti in definirati, katera postavitev (ang. layout) se bo uporabila za prikaz grafičnega vmesnika. Vsaka aktivnost aplikacije mora biti prijavljena v manifestni da-

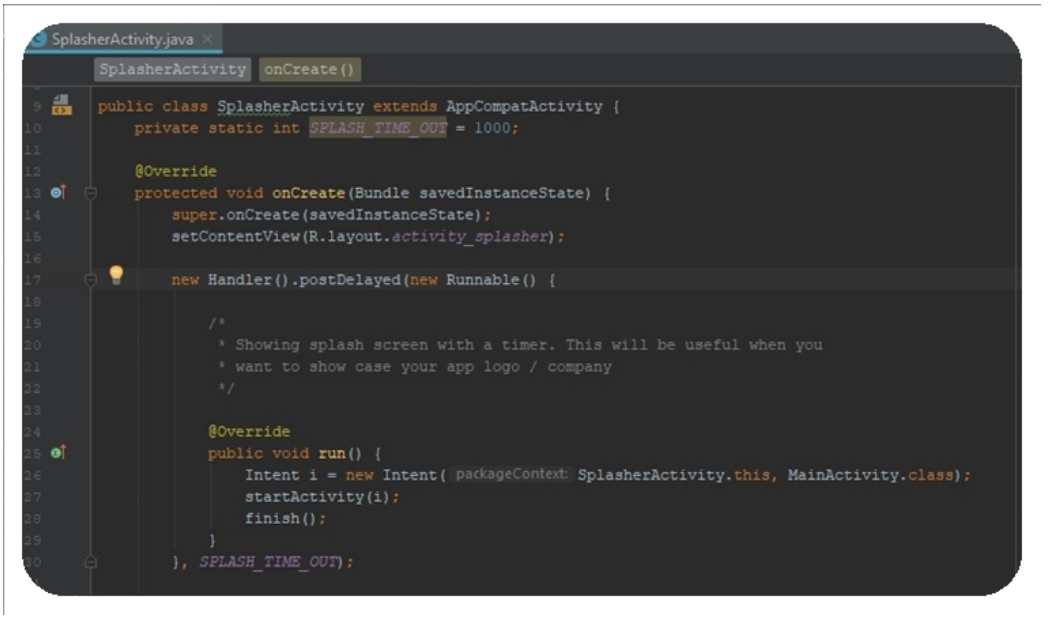
toteki aplikacije (ang. AndroidManifest.xml) [1].

Aktivnosti, ki smo jih programirali v naši aplikaciji so:

- SplasherActivity.java, podpoglavje[5.3.1].
- MainActivity.java, podpoglavje[5.3.2].
- DrawerActivity.java, podpoglavje[5.3.3].

5.3.1 Aktivnost SplasherActivity.java

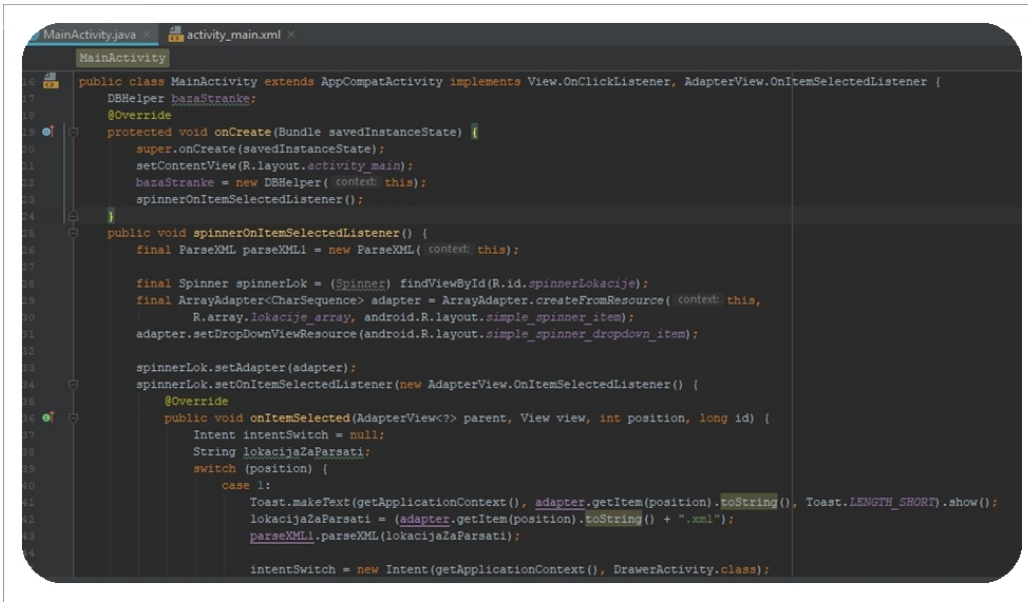
V aktivnosti "SplasherActivity.java" (slika 5.2), smo programirali prikaz prve postavitve (ang. layout), ki se pojavi ko odpremo našo aplikacijo. V tej postavitvi, ki jo imenujemo "activity_splasher.xml" smo postavili en gradnik ImageView za prikaz slike aplikacije in en gradnik TextView za ime aplikacije. Cela aktivnost traja 1 sekundo, ker je to predstavitev logotipa aplikacije, ki ga razvijalec določi. Grafični prikaz aktivnosti (slika 6.1b).



Slika 5.2: Slika aktivnosti SplasherActivity.java.

5.3.2 Aktivnost MainActivity.java

V aktivnosti "MainActivity.java" (slika 5.3), smo sprogramirali prikaz druge postavitve, ki se pojavi kot prvi zaslon, pri katerem imamo izbiro med ponujenimi možnostmi. V tej postavitvi, ki jo imenujemo "activity_main.xml" smo postavili en gradnik TextView za prikaz teksta in en gradnik Spinner iz katerega izberemo eno od možnosti. Grafični prikaz aktivnosti (slika 6.2).

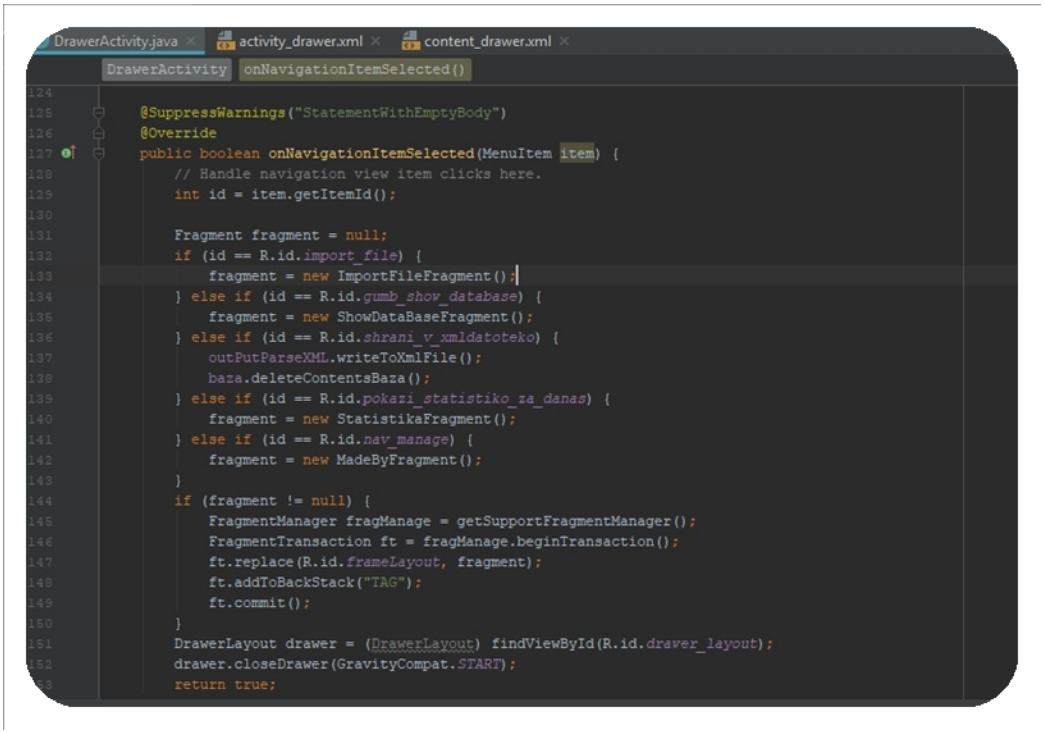


Slika 5.3: Slika aktivnosti MainActivity.java.

5.3.3 Aktivnost DrawerActivity.java

V aktivnosti "DrawerActivity.java" smo programirali prikaz končne postavitve, ki se pojavi kot tretji zaslon po izbiri iz aktivnosti "MainActivity.java". V tej postavitvi, imenovani "content_drawer.xml", smo postavili en gradnik RelativeLayout za prikaz fragmenta "MapsFragment.java". V sledečem podpoglavju, 5.4.3 bomo razložili kaj natančno počne ta fragment. Uporabili smo še gradnik FrameLayout, ki ga uporabljamo za prikaz fragmentov ki so izbrani v nadaljnjem postopku. Ta gradnik je neviden dokler ne aktiviramo kakšnega od fragmentov. V aktivnosti "DrawerActivity.java" (slika

5.4), imamo ponujene različne fragmente, za različna dela. Ko izberemo enega od njih se prikaže ta fragment na zaslonu in ponuja svoje možnosti. Grafični prikaz aktivnosti (slika 6.3).



Slika 5.4: Slika aktivnosti DrawerActivity.java.

5.4 Opis fragmentov mobilne aplikacije

Fragment je blok za večkratno uporabo v aplikaciji Android. Fragmenti so bili predstavljeni v verziji 3.0 Honeycomb, ki je bila prva različica za Android s podporo za aplikacije tabličnih računalnikov. Fragmenti se lahko uporabljajo za prilagajanje aplikacije različnim velikostim zaslona. V istem času bi lahko na zaslonu tabličnega računalnika prikazali več fragmentov v eni aktivnosti, vendar na manjšem zaslonu pametnega telefona le en fragment v različnih aktivnostih. Tehnično je fragment primerek objekta android.app.Fragment [4].

Fragmenti, ki smo jih programirali v naši aplikaciji so:

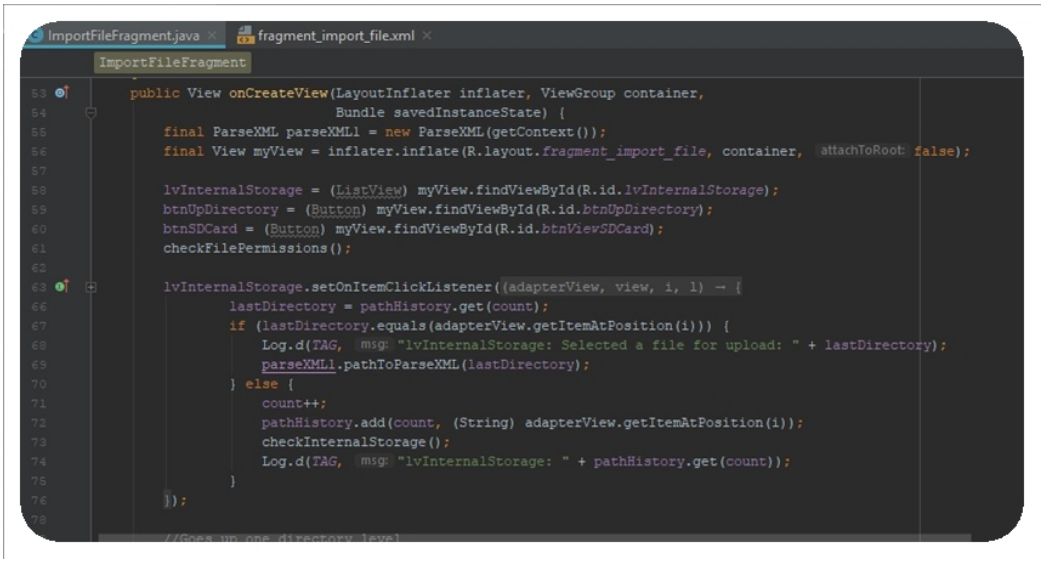
- ImportFileFragment.java, podpoglavje[5.4.1].
- StatistikaFragment.java, podpoglavje[5.4.2].
- MapsFragment.java, podpoglavje[5.4.3].
- FloatingButtonsFragment.java, podpoglavje[5.4.4].
- SMSFragment.java, podpoglavje[5.4.5].
- MailFragment.java, podpoglavje[5.4.6].
- CallFragment.java, podpoglavje[5.4.7].
- CheckFinishFragment.java, podpoglavje[5.4.8].
- ShowPathBetweenTwoFragment.java, podpoglavje[5.4.9].

5.4.1 Fragment ImportFileFragment.java

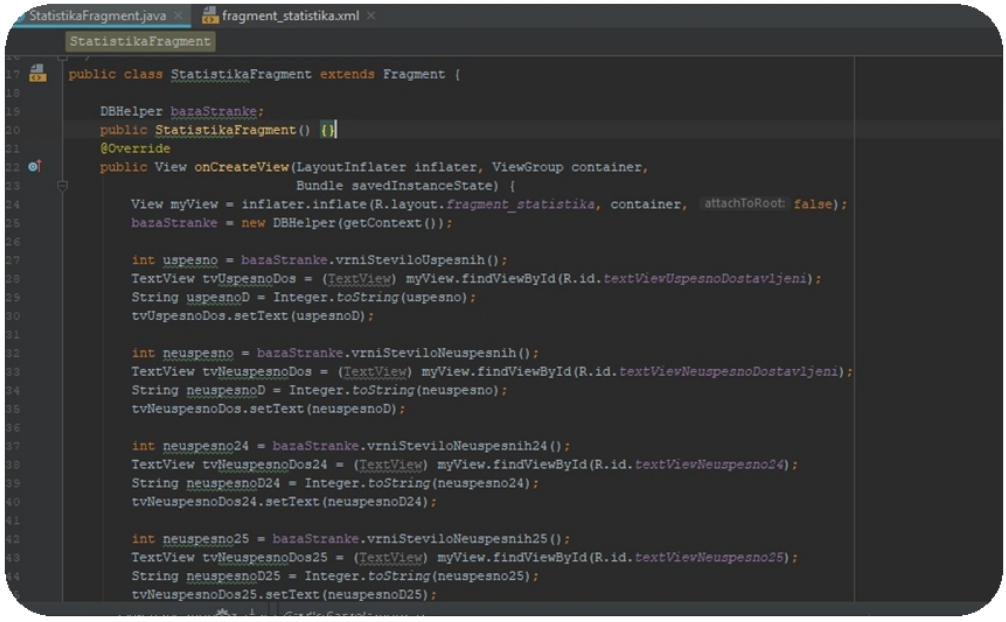
V fragmentu "ImportFileFragment.java" (slika 5.5), smo programirali prikaz postavitve fragmenta, ki vsebuje vse gradnike za prikaz map v shrambi mobilne naprave. V postavitvi, ki jo imenujemo "fragment_import_file.xml", smo postavili 2 gradnika Button, gradnik TextView in gradnik ListView. S pritiskom na prvi gradnik Button se napolni gradnik ListView s seznamom map iz shrambe. Na vsako mapo gradnika ListView lahko pritisnemo in se nam odpre seznam podmap te mape. S pritiskom na drugi gradnik Button se vrnemo en korak nazaj v hierarhijo map naše shrambe. Po seznamu map se premikamo dokler ne pridemo do datoteke, ki jo hočemo uvoziti v našo aplikacijo. Grafični prikaz fragmenta (slika 6.4a).

5.4.2 Fragment StatistikaFragment.java

V fragmentu "StatistikaFragment.java" (slika 5.6), smo programirali prikaz postavitve fragmenta, ki vsebuje vse gradnike za prikaz statistike kurirja v danem dnevu. Celotno smo vstavili 14 gradnikov TextView. 7 gradnikov TextView smo uporabili za izpis informacij o statistikah. 7 gradnikov TextView smo uporabili za izpis številk, ki smo jih pridobili po izračunu statistik. Grafični prikaz fragmenta (slika 6.4b).



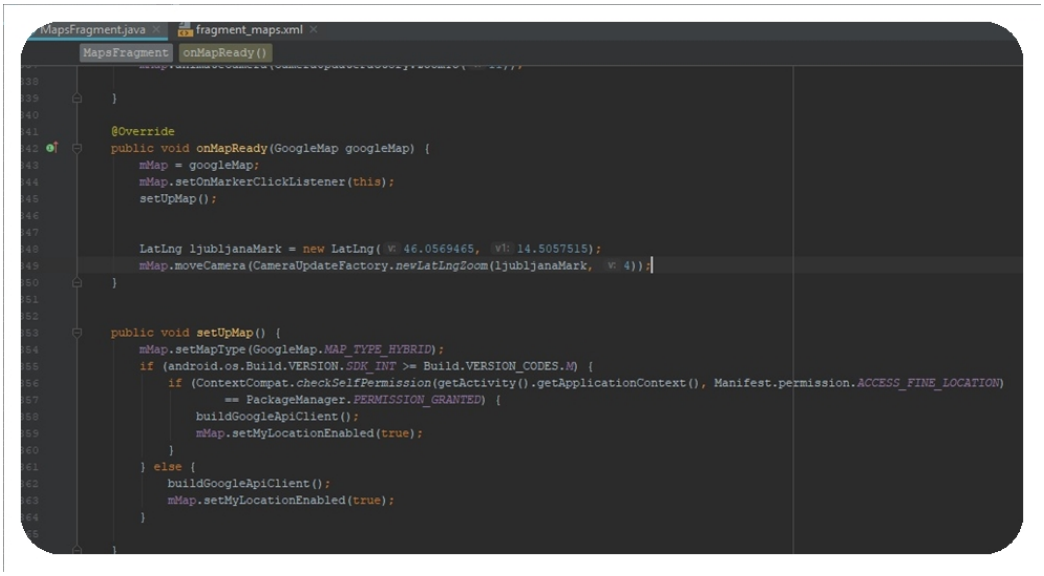
Slika 5.5: Slika fragmenta ImportFileFragment.java.



Slika 5.6: Slika fragmenta StatistikaFragment.java.

5.4.3 Fragment MapsFragment.java

V fragmentu "MapsFragment.java" (slika 5.7), smo programirali prikaz postavitve fragmenta, ki vsebuje vse gradnike za prikaz Google Maps APIja in ostalih elementov v tej postavitvi. V postavitvi, ki jo imenujemo "fragment_maps.xml", smo postavili en gradnik FrameLayout za prikaz Google Maps APIja. Uporabili smo še en gradnik LinearLayout, ki ga uporabljamo za prikaz 4 gradnika FloatingActionButton na zemljevidu. S pritiskom na enega od njih se sprožijo določene funkcije, ki izpišejo svoje rezultate na zemljevidu. V postavitvi imamo še en gradnik FrameLayout, ki ga uporabljamo za prikaz fragmenta "FloatingButtonsFragment.java" 5.4.4, ta gradnik se sproži s pritiskom na označevalec (ang. marker) na zemljevidu. Grafični prikaz fragmenta (slika 6.3a).



Slika 5.7: Slika fragmenta MapsFragment.java.

5.4.4 Fragment FloatingButtonsFragment.java

V fragmentu "FloatingButtonsFragment.java" (slika 5.8), smo programirali prikaz postavitve fragmenta za prikaz gumbov in njihovih funkcionalnosti.

V tej postavitvi smo postavili gradnik LinearLayout, ki vsebuje 4 gradnike FloatingActionButton. Postavili smo tudi gradnik FrameLayout.

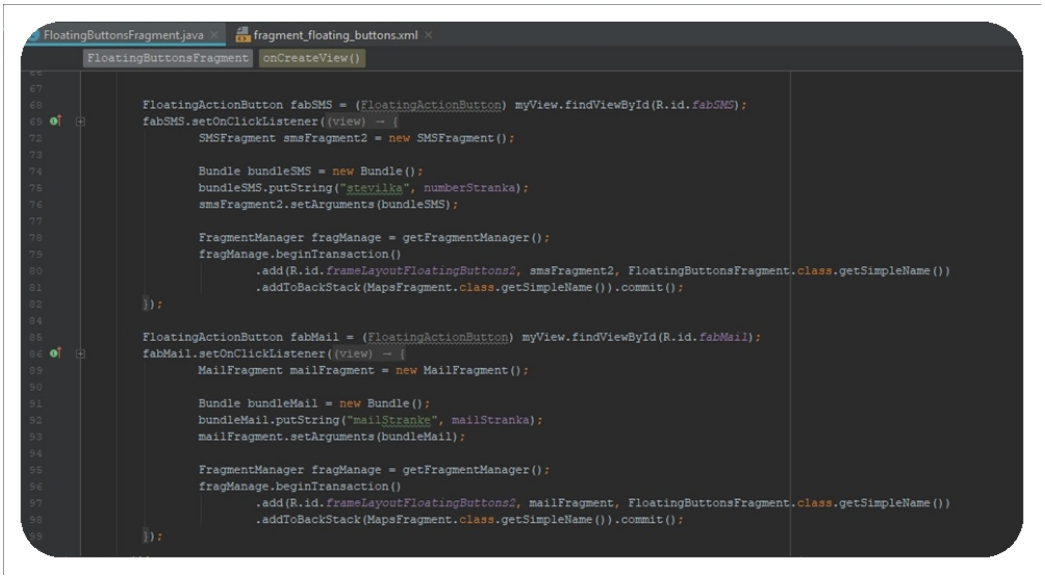
S pritiskom na prvi gradnik FloatingActionButton se nam v gradniku FrameLayout odpre vsebina fragmenta "SMSFragment.java" 5.4.5.

S pritiskom na drugi gradnik FloatingActionButton se nam v gradniku FrameLayout odpre vsebina fragmenta "MailFragment.java" 5.4.6.

S pritiskom na tretji gradnik FloatingActionButton se nam v gradniku FrameLayout odpre vsebina fragmenta "CallFragment.java" 5.4.7.

S pritiskom na četrti gradnik FloatingActionButton se nam v gradniku FrameLayout odpre vsebina fragmenta "CheckFinishFragment.java" 5.4.8.

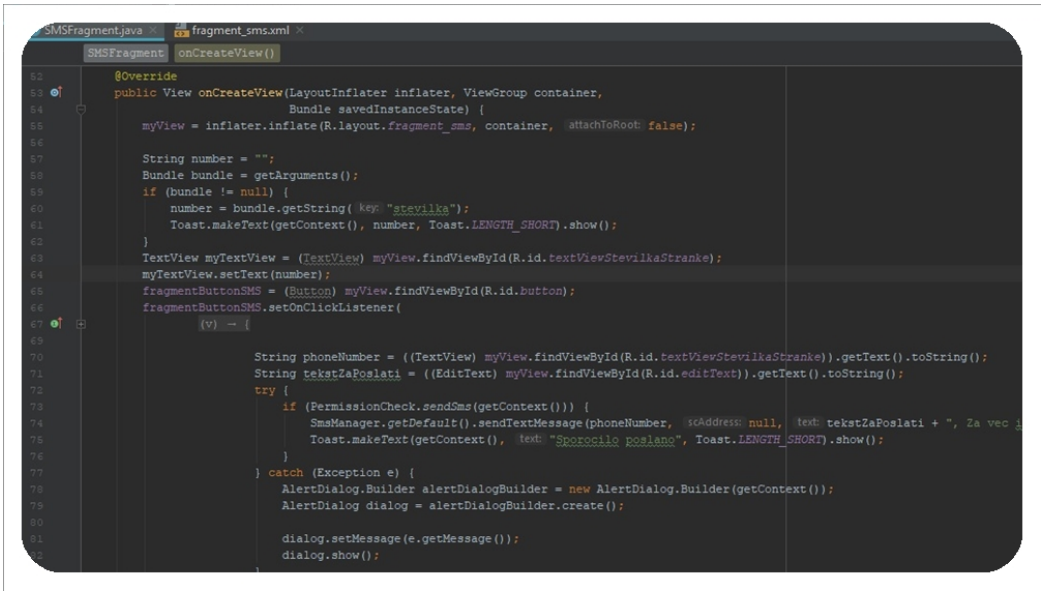
Vsi navedeni fragmenti so v nadaljevanju diplomskega dela tudi opisani.



Slika 5.8: Slika fragmenta FloatingButtonsFragment.java.

5.4.5 Fragment SMSFragment.java

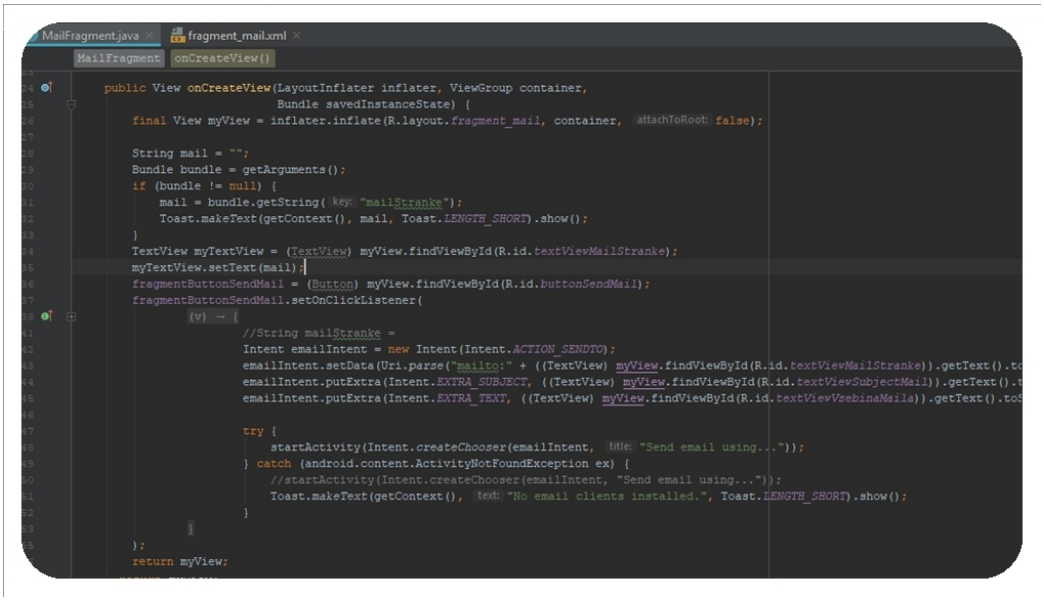
V fragmentu "SMSFragment.java" (slika 5.9), smo programirali prikaz postavitve fragmenta za gradnike, ki se uporabljajo za pošiljanje kratkih sporočil. V tej postavitvi smo uporabili 3 gradnike TextView, gradnik EditText in gradnik Button. Grafični prikaz fragmenta (slika 6.6a).



Slika 5.9: Slika fragmenta SMSFragment.java.

5.4.6 Fragment MailFragment.java

V fragmentu "MailFragment.java" (slika 5.10), smo programirali prikaz

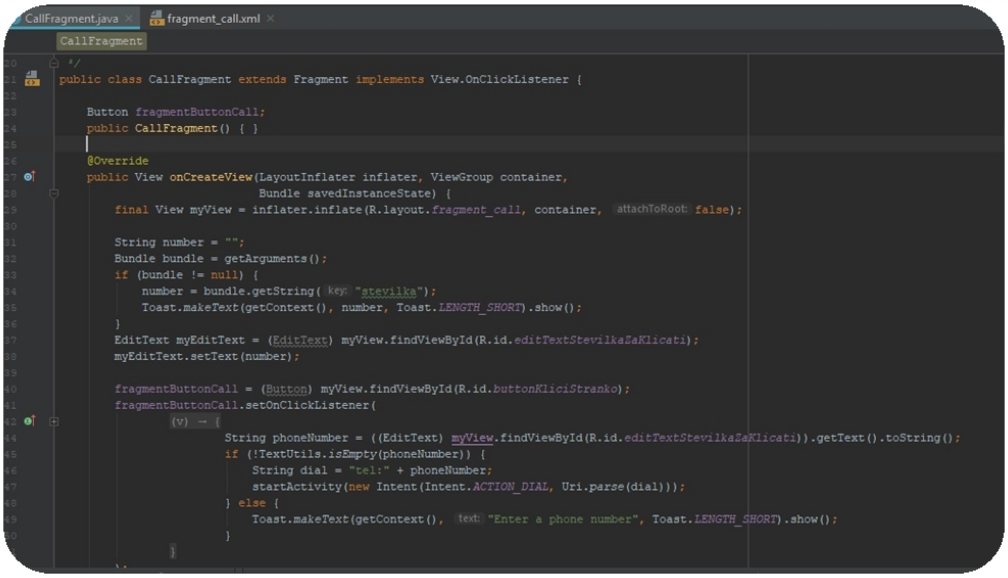


Slika 5.10: Slika fragmenta MailFragment.java.

postavitve fragmenta za gradnike, ki se uporabljajo za pošiljanje elektronske pošte. V tej postavitvi smo uporabili 3 gradnike TextView in gradnik Button. Grafični prikaz fragmenta (slika 6.6b).

5.4.7 Fragment CallFragment.java

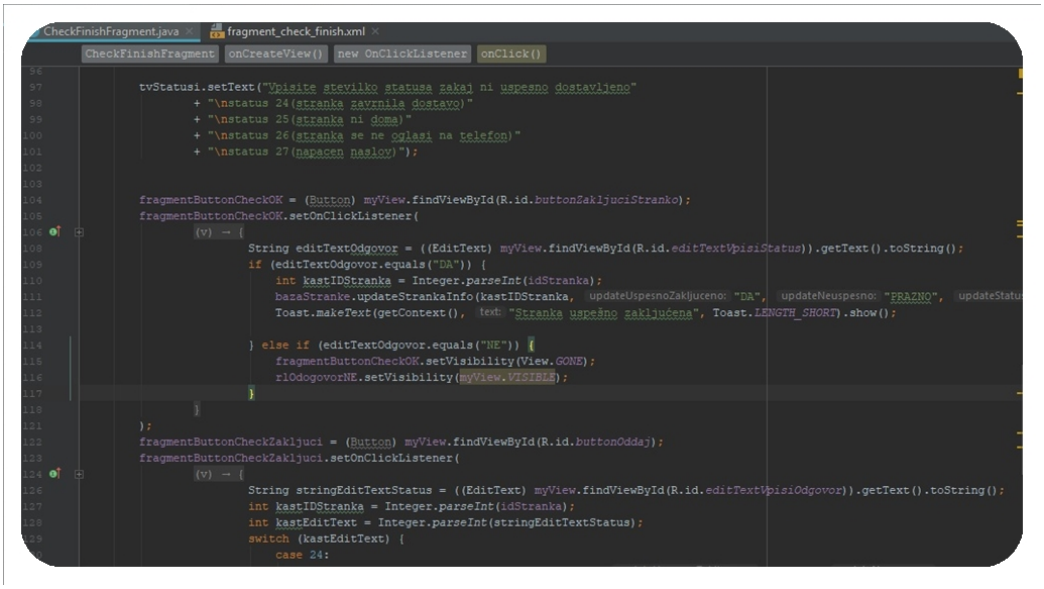
V fragmentu "CallFragment.java" (slika 5.11), smo programirali prikaz postavitve fragmenta za gradnike, ki se uporabljajo za klicanje. V tej postavitvi smo uporabili gradnik TextView, gradnik EditText in gradnik Button. Grafični prikaz fragmenta (slika 6.6c).



Slika 5.11: Slika fragmenta CallFragment.java.

5.4.8 Fragment CheckFinishFragment.java

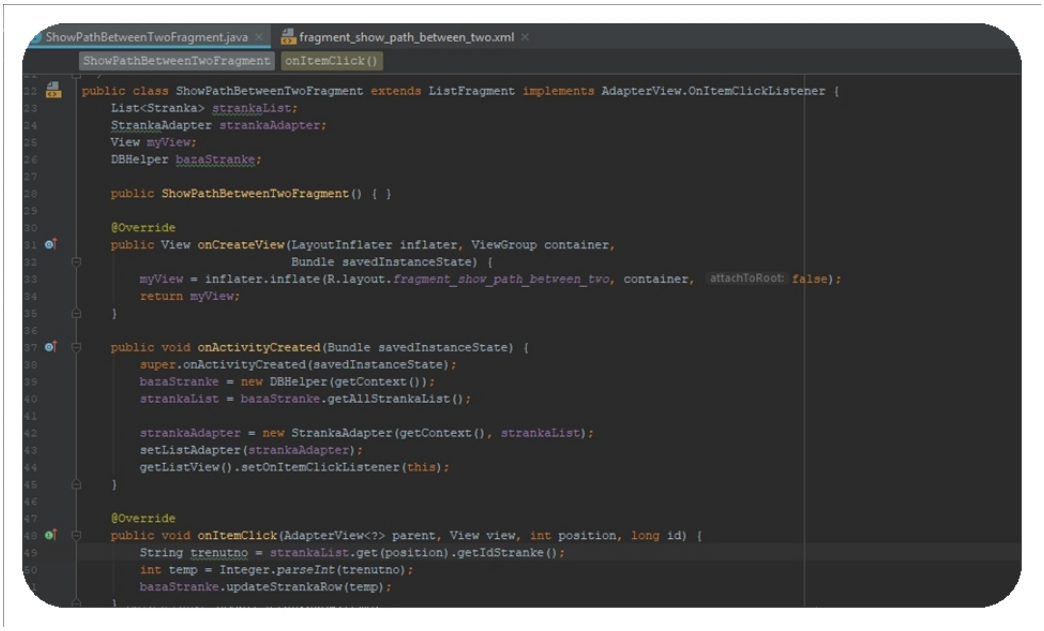
V fragmentu "CheckFinishFragment.java" (slika 5.12), smo programirali prikaz postavitve fragmenta za gradnike, ki se uporabljajo za zaključek dostave. V tej postavitvi smo uporabili 3 gradnike TextView, 2 gradnika EditText in 2 gradnika Button. Grafični prikaz fragmenta (slika 6.6d).



Slika 5.12: Slika fragmenta CheckFinishFragment.java.

5.4.9 Fragment ShowPathBetweenTwoFragment.java

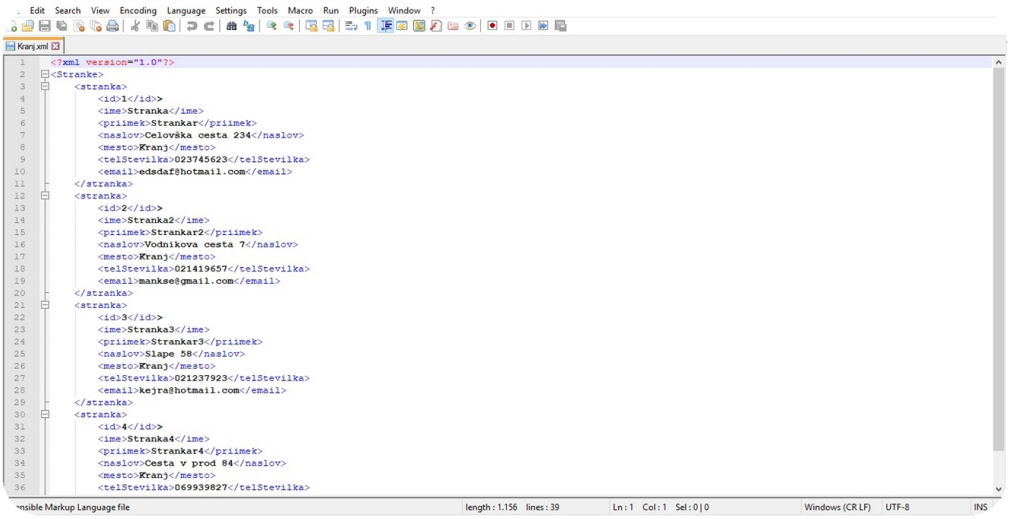
V fragmentu "ShowPathBetweenTwoFragment.java" (slika 5.13), smo programirali prikaz postavitve fragmenta za gradnike, ki se uporabljajo za izbiro stranke, ki jo bo kurir prvo obiskal. V tej postavitvi smo uporabili 2 gradnika TextView in gradnik ListView. Gradnik ListView se napolni s strankami, ki jih mora kurir obiskati. S klikom na izbrano stranko se mu na zemljevidu izriše najbližja pot od njegove trenutne lokacije do lokacije željene stranke. Grafični prikaz fragmenta (slika 6.5d).



Slika 5.13: Slika fragmenta ShowPathBetweenTwoFragment.java.

5.5 Datoteka XML

Ustvarili smo datoteko XML (slika 5.14), iz katere smo pozneje brali vsebino.



Slika 5.14: Pogled v datoteko XML iz katere smo brali vsebino.

Prebrano vsebino smo razčlenili in naprej uporabljali v naši aplikaciji.

5.5.1 Branje iz datoteke XML, ParseXML.java

Prebrano vsebino smo razčlenili s pomočjo pomožnega razreda XmlPullParser na oznake <stranka>, ki so uporabljene v XML datoteki. Oznake <stranka> smo naprej razčlenili na oznake (slika 5.15):

- ID.
- Ime.
- Priimek.
- Naslov.
- Mesto.
- Telefonska številka.
- Elektronska pošta.

Pridobljene oznake smo zapisovali v našo lokalno podatkovno bazo.

ParseXML QUERY_URL

```
85      public void procesParsing(XmlPullParser parser) throws XmlPullParserException, IOException {
86          ArrayList<Stranka> aListaStranka = new ArrayList<>();
87          int eventType = parser.getEventType();
88          Stranka trenutnaStranka = null;
89
90          while (eventType != XmlPullParser.END_DOCUMENT) {
91              String elementName = null;
92
93              switch (eventType) {
94                  case XmlPullParser.START_TAG:
95                      elementName = parser.getName();
96
97                      if ("stranka".equals(elementName)) {
98                          trenutnaStranka = new Stranka();
99                          aListaStranka.add(trenutnaStranka);
100                     } else if (trenutnaStranka != null) {
101                         if ("id".equals(elementName)) {
102                             trenutnaStranka.setIdStranke(parser.nextText());
103                         } else if ("ime".equals(elementName)) {
104                             trenutnaStranka.setImeStranke(parser.nextText());
105                         } else if ("priimek".equals(elementName)) {
106                             trenutnaStranka.setPriimekStranke(parser.nextText());
107                         } else if ("naslov".equals(elementName)) {
108                             trenutnaStranka.setNaslovStranke(parser.nextText());
```

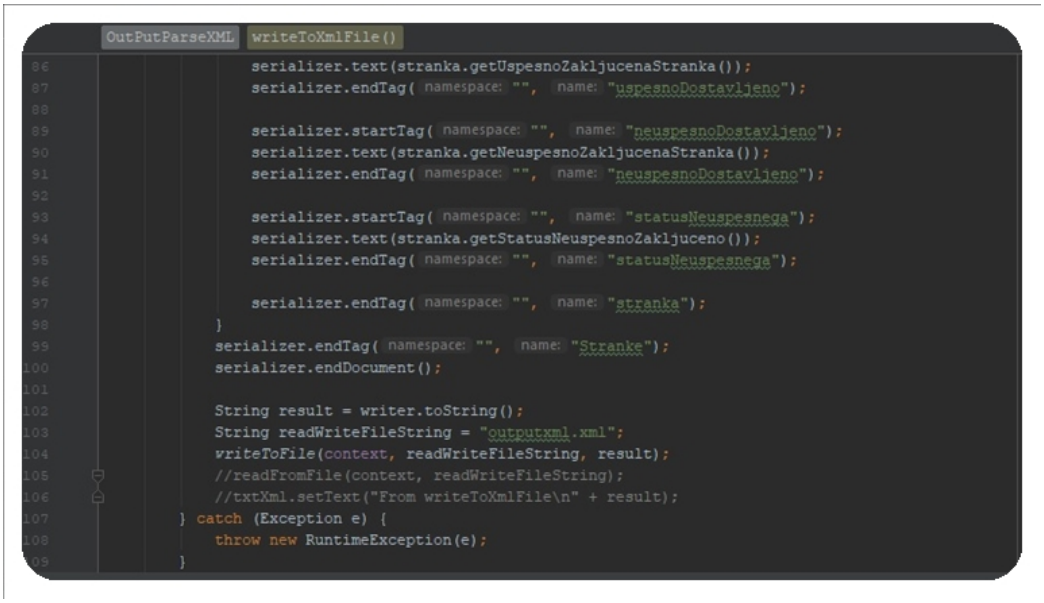
Slika 5.15: ParseXML.java, razred za razčlenjevanje datoteke XML.

5.5.2 Pisanje v XML datoteko, OutPutParseXML.java

Za pisanje v XML datoteko smo uporabili pomožni razred XmlSerializer, ki v datoteko zapisuje po strukturi XML oziroma oznakah. Oznake, ki smo jih zapisovali v XML datoteko za vsako oznako <stranka> so (slika 5.16):

- ID.
- Ime.
- Priimek.
- Naslov.
- Mesto.
- Telefonska številka.
- Elektronska pošta.
- Uspešno dostavljena pošiljka.
- Neuspešno dostavljena pošiljka.
- Status neuspešno dostavljene pošiljke.

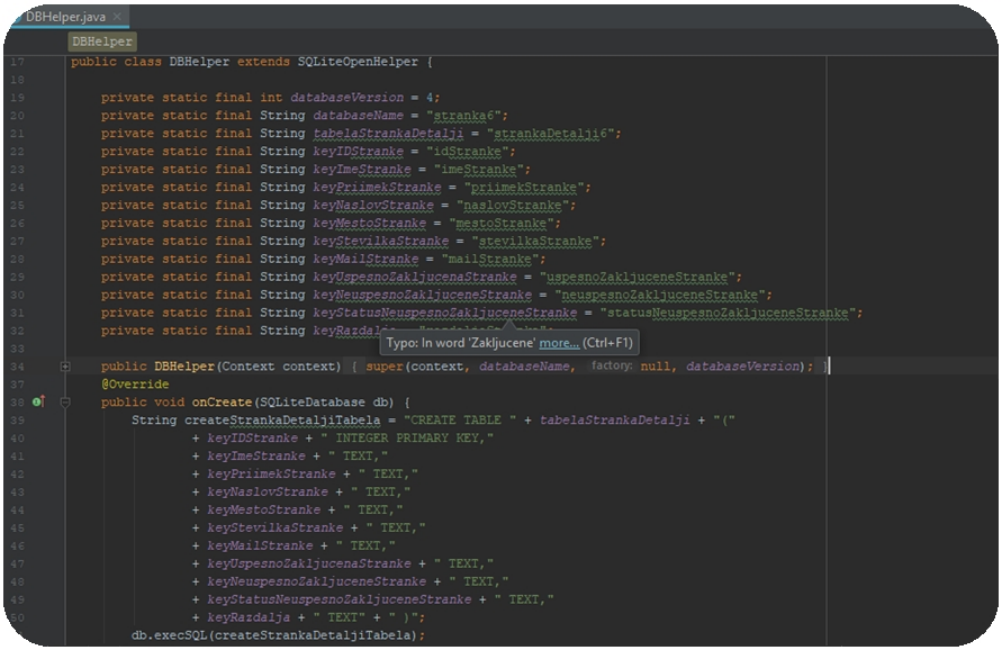
Grafični prikaz zapisa v datoteko (slika 6.4c).



Slika 5.16: OutPutParseXML.java, razred za pisanje v datoteko XML.

5.6 Podatkovna baza

Za podatkovno bazo smo ustvarili razred “DBHelper.java”, ki razširja pomožni razred SQLiteOpenHelper. Razred “DBHelper.java” vsebuje strukturo naše podatkovne baze (slika 5.17).



Slika 5.17: Pregled razreda DBHelper v katerem se nahaja podatkovna baza.

5.6.1 Pisanje v podatkovno bazo

V podatkovni bazi smo ustvarili tabelo (slika 5.17), ki je imela enako število stolpcev kot je imel objekt komponent. V tabelo smo zapisovali vrednosti metode getter klicanega objekta (slika 5.18). Objekti so bili tipa Stranka (slika 5.20).

```
77    public void addNewStranka(Stranka stranka) {
78
79        SQLiteDatabase db = this.getWritableDatabase();
80
81        ContentValues values = new ContentValues();
82
83        //values.put(keyIDStranke, stranka.getIdStranke());
84        values.put(keyImeStranke, stranka.getImeStranke());
85        values.put(keyPriimekStranke, stranka.getPriimekStranke());
86        values.put(keyNaslovStranke, stranka.getNaslovStranke());
87        values.put(keyMestoStranke, stranka.getMestoStranke());
88        values.put(keyStevilkaStranke, stranka.getStevilkaStranke());
89        values.put(keyMailStranke, stranka.getMailStranke());
90        values.put(keyUspesnoZakljucenaStranke, "");
91        values.put(keyNeuspesnoZakljuceneStranke, "");
92        values.put(keyStatusNeuspesnoZakljuceneStranke, "");
93
94        // Inserting Row
95        db.insert(tabelaStrankaDetalji,  nullColumnHack: null, values);
96        db.close(); // Closing database connection
97    }
```

Slika 5.18: Pregled zapisovanja vrednosti v podatkovno bazo.

5.6.2 Branje iz podatkovne baze

Podatke iz baze smo brali s pomočjo pomožnega razreda Cursor (slika 5.19), ki omogoča branje po stolpcih. Vrednosti smo zapisovali v seznam.

```
DBHelper.java
DBHelper  getAllStrankaList2()
290
291     public List<Stranka> getAllStrankaList2() {
292         List<Stranka> strankaLista = new ArrayList<Stranka>();
293
294         // Select All Query
295         String selectQuery = "SELECT  * FROM " + tabelaStrankaDetalji;
296         SQLiteDatabase db = this.getWritableDatabase();
297         Cursor cursor = db.rawQuery(selectQuery,  selectionArgs: null);
298
299         // looping through all rows and adding to list
300         if (cursor.moveToFirst()) {
301             do {
302                 Stranka stranka = new Stranka();
303                 stranka.setIdStranke(cursor.getString( columnIndex: 0));
304                 stranka.setImeStranke(cursor.getString( columnIndex: 1));
305                 stranka.setPriimekStranke(cursor.getString( columnIndex: 2));
306                 stranka.setNaslovStranke(cursor.getString( columnIndex: 3));
307                 stranka.setMestoStranke(cursor.getString( columnIndex: 4));
308                 stranka.setStevilkaStranke(cursor.getString( columnIndex: 5));
309                 stranka.setMailStranke(cursor.getString( columnIndex: 6));
310                 stranka.setUspesnoZakljucenaStranka(cursor.getString( columnIndex: 7));
311                 stranka.setNeuspesnoZakljucenaStranka(cursor.getString( columnIndex: 8));
312                 stranka.setStatusNeuspesnoZakljuceno(cursor.getString( columnIndex: 9));
313                 stranka.setRazdalja(cursor.getString( columnIndex: 10));
314                 // Adding contact to list
315                 strankaLista.add(stranka);
316             } while (cursor.moveToNext());
317         }
318         // return list
319         return strankaLista;
320     }
```

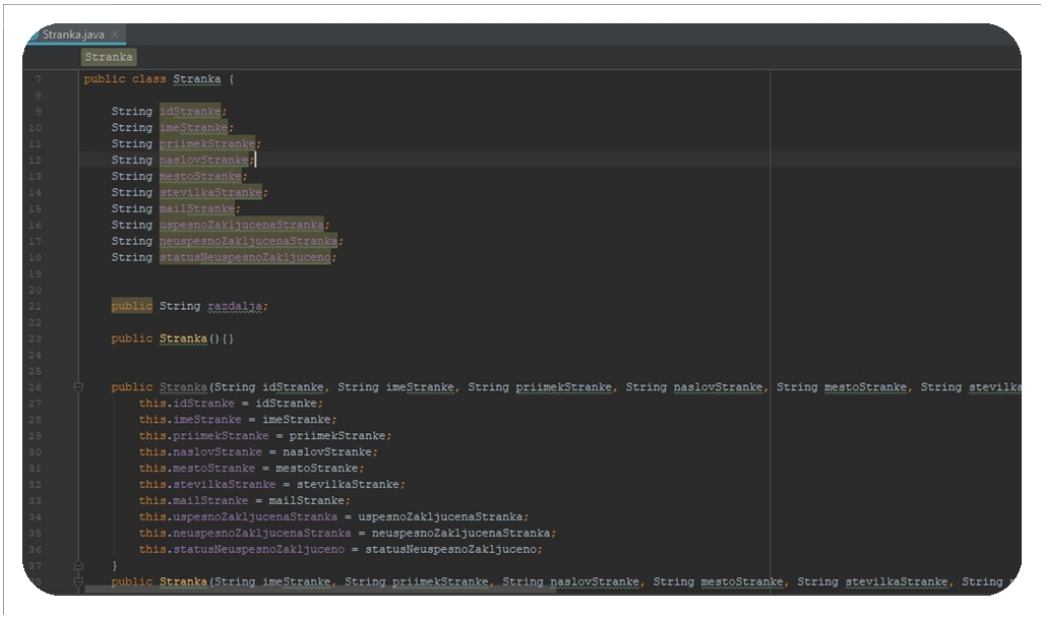
Slika 5.19: Pregled branja vrednosti iz podatkovne baze.

5.7 Opis dodatnih razredov

Programirali smo tudi pomožni razred, ki smo ga uporabili v ostalih razredih.

5.7.1 Razred Stranka

V tem razredu smo ustvarili vse atribute, metode getterjev in setterjev. Getter je metoda, ki omogoča branje atributa. Običajno ime: getImeAtributa (primer: getImeStranke()). Setter je, nasprotno, metoda, s katero lahko kontrolirano nastavimo vrednost atributa. Običajno ime: setImeAtributa (primer: setImeStranke()).



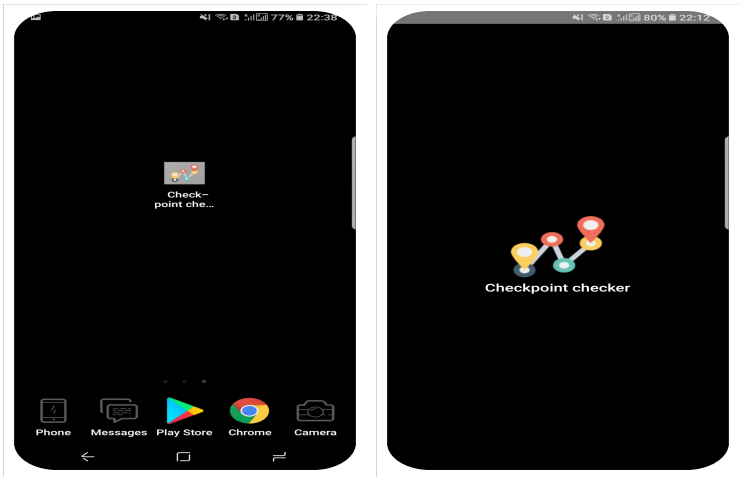
Slika 5.20: Pregled razreda Stranka.

Poglavje 6

Delovanje aplikacije

6.1 Zagon aplikacije

Za našo aplikacije smo izbrali ikono (slika 6.1a). Ob odprtju aplikacije se pokaže logotip naše aplikacije (slika 6.1b). Omenjene stvari so ustvarjene bolj za zabavo proizvajalca, ker aplikacija deluje tudi brez njih. V diplomskem delu smo se za ta korak odločili, ker s tem bistveno izboljšamo izgled naše aplikacije.



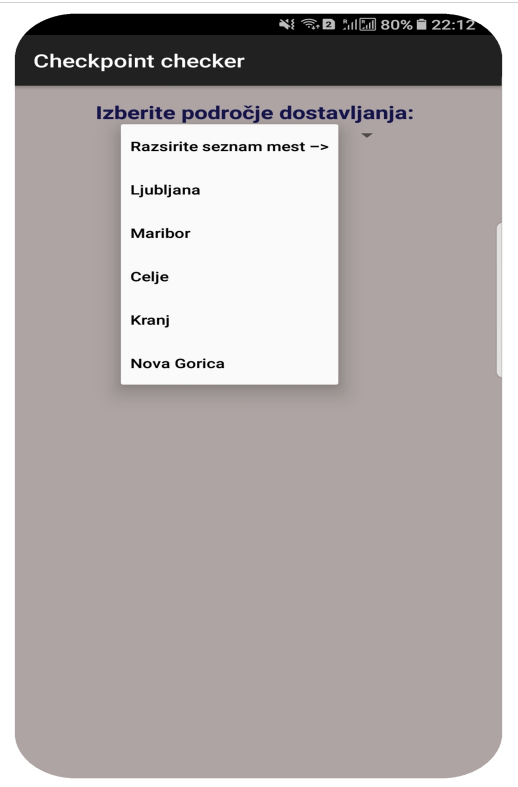
(a) Ikona naše aplikacije. (b) Logotip naše aplikacije.

Slika 6.1: Zaslonska maska SplasherActivity.java

6.2 Uporabniški vmesnik aplikacije

6.2.1 Zaslonska maska MainActivity.java

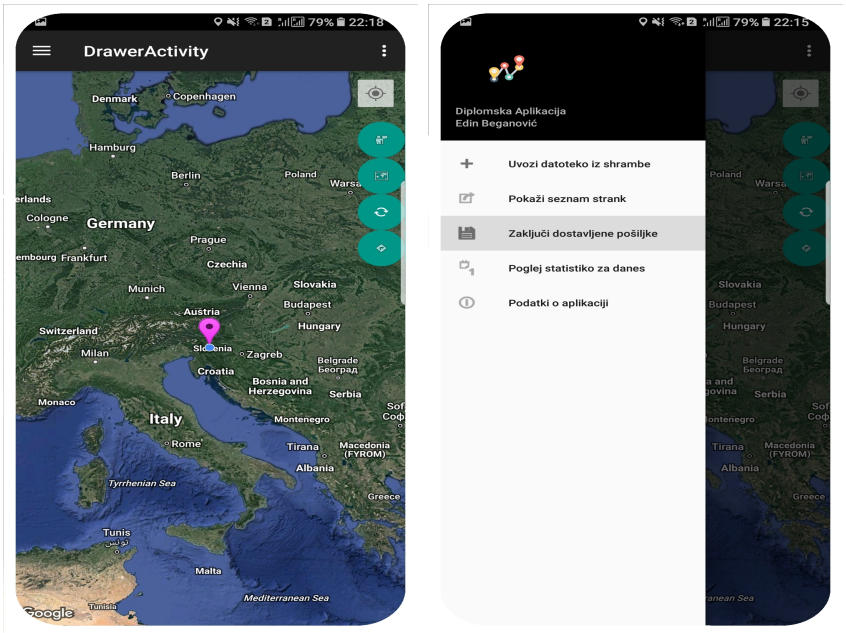
Prva zaslonska maska, ki jo dobi uporabnik na vpogled (slika 6.2). Tukaj imamo možnost izbire področja dostavljanja. Pri vsaki možnosti izbire se nam odpre ista zaslonska maska. Po končani izbiri, aplikacija v ozadju izbiro procesira in na podlagi izbranega področja, pošlje zahtevo strežniku za datoteko "Področje.xml". Če povezava s strežnikom uspe se nam avtomatsko naložijo podatki iz datoteke "Področje.xml" v aplikacijo. Če povezava ne uspe moramo ročno uvoziti datoteko "Področje.xml" iz naše shrambe in datoteko v tem primeru dobimo po elektronski pošti.



Slika 6.2: Zaslonska maska MainActivity.java.

6.2.2 Zaslonska maska DrawerActivity.java

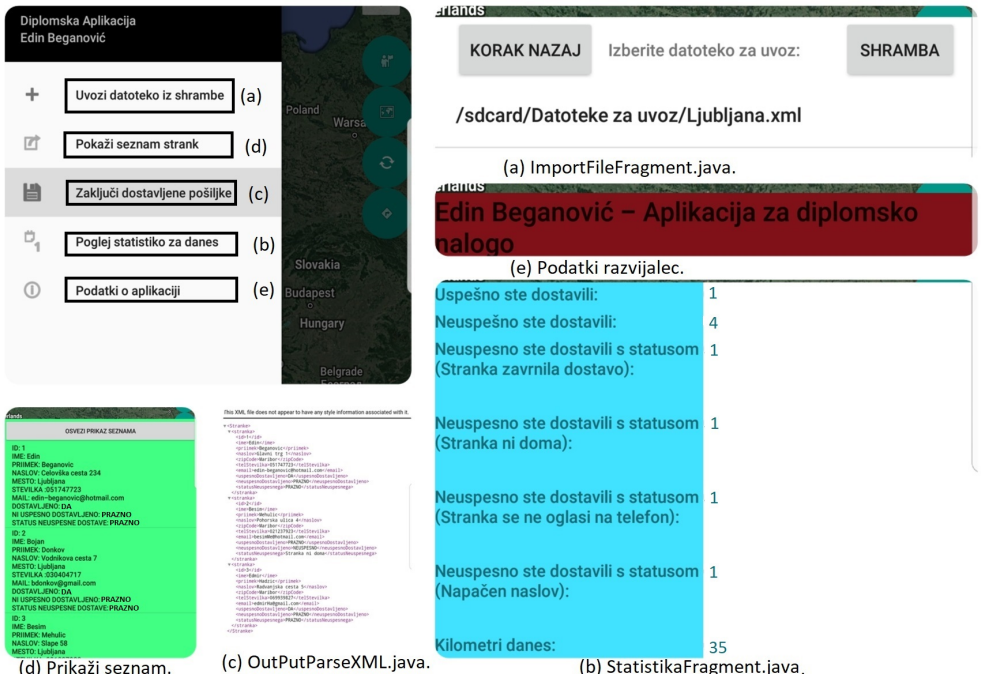
Zaslonska maska DrawerActivity.java vsebuje Google Maps API (slika 6.3a). Po razširitvi predalnika pridobimo 5 možnosti (slika 6.3b), s pritiskom na:



(a) Google Maps API in gumbi. (b) Predalnik možnosti.

Slika 6.3: Zaslonska maska DrawerActivity.java

- "Uvozi datoteko iz shrambe", dobimo zaslonsko masko iz slike 6.4a, ročni uvoz datoteke XML v aplikacijo.
- "Pokaži seznam strank", dobimo zaslonsko masko iz slike 6.4d, to nam pokaže vse dostave kurirja, ki jih ima.
- "Zaključi dostavljene pošiljke", dobimo zaslonsko masko iz slike 6.4c, shrani dostavljene pošiljke kurirja v datoteko "output.xml".
- "Poglej statistiko za danes", dobimo zaslonsko masko iz slike 6.4b, izpiše nam statistiko za vse uspešno in neuspešno dostavljene pošiljke.
- "Podatki o aplikaciji", dobimo zaslonsko masko iz slike 6.4e, dobimo informacije o razvijalcu.



Slika 6.4: Zaslonska maska vsake možnosti iz predalnika.

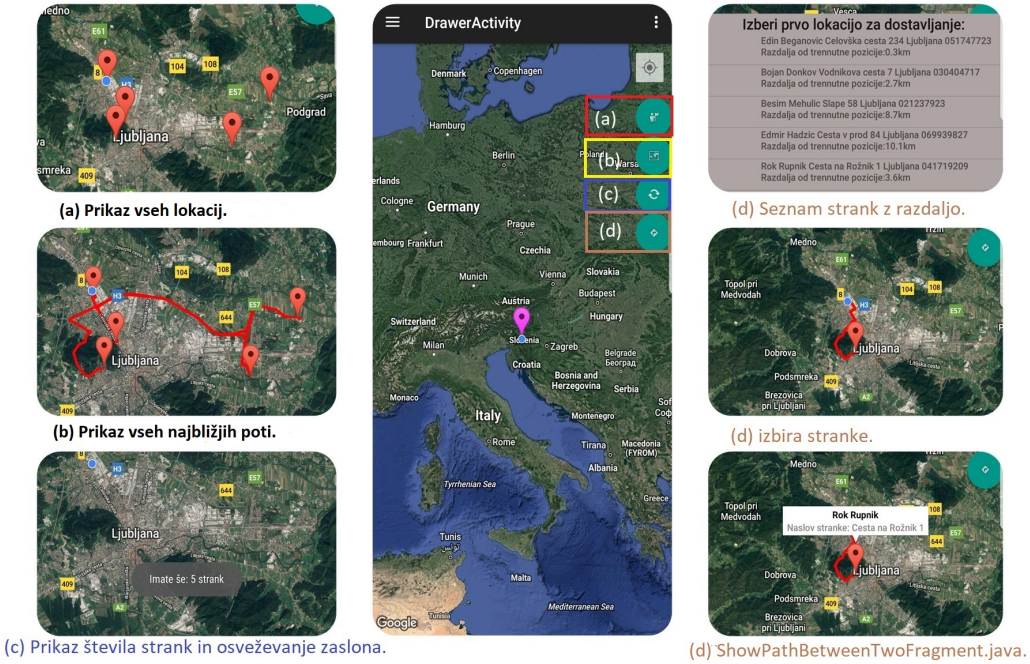
6.3 Scenarij uporabe aplikacije

V zaslonski maski DrawerActivity.java (slika 6.5), imamo 5 gumbov. Prvi gumb desno zgoraj, uporabljamo za globalni sistem pozicioniranja (ang. GPS), da nam pokaže našo trenutno lokacijo.

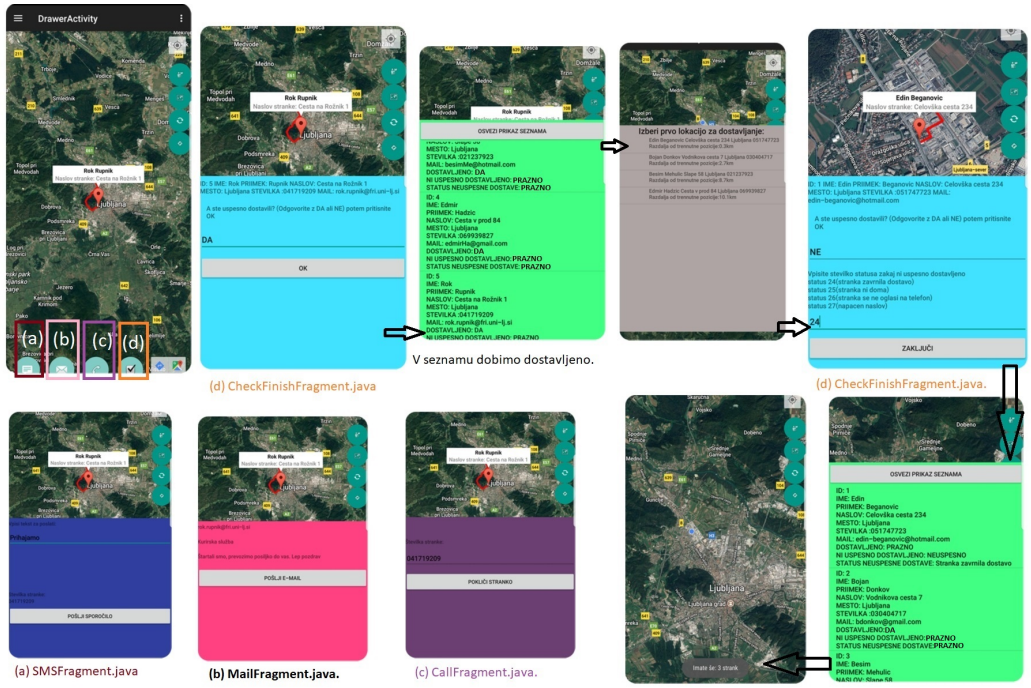
Drugi gumb uporabljamo za prikaz vseh dostav na zemljevidu, pokaže vse dostave v obliki označevalcev (slika 6.5a).

Tretji gumb uporabljamo za prikaz vseh najbližjih poti med dostavami (slika 6.5b).

Četrti gumb uporabljamo za osveževanje zaslona in prikaz števila dostav (slika 6.5c). Peti gumb uporabljamo za prikaz seznama, vseh podatkov o dostavi in za razdaljo med našo trenutno lokacijo ter lokacijo dostave. V seznamu imamo možnost izbire najbližje dostave ali katerikoli druge dostave. Po izbiri nam na zemljevidu izriše najbližjo pot od trenutne lokacije do izbrane dostave. Grafični prikaz imamo v sliki 6.5d.



Slika 6.5: Scenarij uporabe aplikacije.



Slika 6.6: Nadaljevanje scenarija uporabe aplikacije.

Naš scenarij uporabe se nadaljuje s pritiskom na označevalec izbrane dostave (slika 6.6). Dobimo 4 nove gumbe, s sledečimi možnostmi.

Gumb z oznako (a), nam ponuja pošiljanje SMSa stranki, ki se nahaja na tem naslovu (slika 6.6a).

Gumb z oznako (b), nam ponuja pošiljanje elektronske pošte stranki, ki se nahaja na tem naslovu (slika 6.6b).

Gumb z oznako (c), nam ponuja klicanje stranke, ki se nahaja na tem naslovu (slika 6.6c).

Gumb z oznako (d), nam ponuja zaslonsko masko katera vsebuje vprašanje "A ste uspešno dostavili?". Če vpišemo odgovor "DA" in pritisnemo gumb "OK", se dostava zaključi, stranko shranimo v seznam in smo končali s to dostavo (slika 6.6d). Če vpišemo odgovor "NE", dobimo novo vprašanje "Vpišite številko statusa zakaj ni uspešno dostavljeno?". Na izbiro dobimo več statusov, kot so npr. status 24, ki pomeni da je stranka zavrnila dostavo ali status 25, ki pomeni da stranka ni doma. Ko vnesemo številko statusa in pritisnemo gumb "Zaključi" se zaključi dostava, stranko shranimo v seznam in smo končali s to dostavo (slika 6.6d).

To je scenarij uporabe naše aplikacije.

Poglavje 7

Sklepne ugotovitve

V okviru diplomske naloge je bila razvita mobilna aplikacija, ki je namenjena kurirjem za hitrejšo dostavo pošiljk. Glavne funkcionalnosti, ki smo jih hoteli poudariti, so načini pridobivanja datoteke ".xml" s strežnika, ločevanje podatkov o stranki iz datoteke ".xml", prikaz podatkov o stranki na zemljevidu, možnost komuniciranja s stranko preko kratkih sporočil ali klica ali e-pošte, izris najbližje poti do stranke na zemljevidu, pošiljanje obravnavanih podatkov nazaj na strežnik in prikaz statistike uspešnosti kurirja za tisti dan. K razvoju smo pristopili, ker nismo zasledili nobene druge mobilne aplikacije, ki bi izpolnjevala naše zadane kriterije.

7.1 Možne nadgradnje

Razvito aplikacijo je možno v prihodnje še nadgraditi in ji dodati nove funcionalnosti kot so:

- stran za prijavo, da se lahko kurir prijavi s svojim uporabniškim imenom in geslom,
- pošiljanje podatkov o dostavljenih pošiljkah direktno na strežnik,
- skeniranje "QR-code" z novo prevzetih pošiljk in pošiljanje podatkov na strežnik,
- Izdelava podobne aplikacije za stranko, da lahko le-ta pridobi informa-

cije, da je lahko v kontaktu s kurirjem in da lahko sledi kurirju na svoji aplikaciji, ko kurir označi, da bo dostavljal tej stranki.

Literatura

[1] Aktivnost Android. Dosegljivo: https://www.droidwiki.org/wiki/Activity, 2018. [Dostopano: 24. 2. 2018].

[2] Operacijski sistem Android. Dosegljivo: https://en.wikipedia.org/wiki/Android_(operating_system), 2018. [Dostopano: 22. 2. 2018].

[3] Razvojno okolje Android Studio. Dosegljivo: https://en.wikipedia.org/wiki/Android_Studio, 2018. [Dostopano: 23. 2. 2018].

[4] Fragment Android. Dosegljivo: https://www.droidwiki.org/wiki/Fragment, 2018. [Dostopano: 24. 2. 2018].

[5] Google Maps. Dosegljivo: https://en.wikipedia.org/wiki/Google_Maps, 2018. [Dostopano: 22. 2. 2018].

[6] Opis Java. Dosegljivo: https://sl.wikipedia.org/wiki/Programski_jezik_java, 2018. [Dostopano: 23. 2. 2018].

[7] Notepad++. Dosegljivo: https://en.wikipedia.org/wiki/Notepad%2B%2B, 2018. [Dostopano: 22. 2. 2018].

[8] Poklic kurir. Dosegljivo: https://www.ess.gov.si/ncips/cips/opisi_poklicev/opis_poklica?Kljuc=2361&Filter=, 2018. [Dostopano: 20. 2. 2018].

[9] Vrste pošiljk. Dosegljivo: https://www.uradni-list.si/glasilo-uradni-list-rs/vsebina/24897, 2018. [Dostopano: 20. 2. 2018, 6. člen].

[10] Opis mobilne naprave. Dosegljivo: http://www.telekom.si/zasebni-uporabniki/mobiteli-in-naprave/mobiteli/samsung-galaxy-note8, 2018. [Dostopano: 22. 2. 2018].

[11] SQLite. Dosegljivo: https://sqlite.org/about.html, 2018. [Dostopano: 22. 2. 2018].