MOBILNA APLIKACIJA ZA AUTOMATSKU NAPLATU PARKIRANJA

Širac, Filip

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:672773

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-17**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni preddiplomski studij računarstva

MOBILNA APLIKACIJA ZA AUTOMATSKU NAPLATU PARKIRANJA

Završni rad

Filip Širac

Osijek, 2019.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 22.09.2019.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Filip Širac
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3832, 19.09.2018.
OIB studenta:	99172256542
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Mobilna aplikacija za automatsku naplatu parkiranja
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 1 bod/boda Jasnoća pismenog izražavanja: 1 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	22.09.2019.
Datum potvrde ocjene Odbora:	25.09.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2019.

Ime i prezime studenta:	Filip Širac
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3832, 19.09.2018.
Ephorus podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija za automatsku naplatu parkiranja**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD.....3
   1.1 Zadatak završnog rada.....3
2. IZAZOVI KOD NAPLATE PARKIRANJA.....4
   2.1 Infrastuktura parkiranja.....4
   2.2 Postojeće aplikacije.....8
   2.3 Problemi kod aplikacije.....11
   2.4 Vlastita ideja.....11
3. MODEL MOBILNE APLIKACIJE ZA AUTOMATSKU NAPLATU PARKIRANJA.....12
   3.1 Korisničko sučelje.....12
   3.2 Baza.....13
   3.3 Potencijalni korisnici.....14
   3.4 Geolokacija.....15
   3.5 Postupak naplate.....16
4. PROGRAMSKO RJEŠENJE.....17
   4.1 Okolina i tehnologija razvoja aplikacije.....17
      4.1.1 Android Studio.....17
      4.1.2 Programski jezik Kotlin.....18
      4.1.3 XML.....19
   4.2 Programsko rješenje korisničkog sučelja.....20
   4.3 Opis bitnih djelova programa.....26
      4.3.1 Manifest.....26
      4.3.2 Glavni aktivit.....27
      4.3.3 Dijalog za dodavanje novog vozila.....28
      4.3.4 Programsko rješenje baze podataka.....30
   4.4 Arhitektura mobilne aplikacije.....32
   4.5 Slojevi aplikacije.....35
5. NAČIN KORIŠTENJA I ISPITIVANJE RADA APLIKACIJE.....39
   5.1 Opis kako se koristi aplikacija.....39
   5.2 Scenarij i ispitivanje.....40
   5.3 Analiza.....40
6. ZAKLJUČAK.....41
7. LITERATURA.....42
   SAŽETAK.....44
   ABSTRACT.....45
   ŽIVOTOPIS.....46

PRILOZI.....47

1. UVOD

U doba kada tehnologija uvelike olakšava svakodnevni život svih nas, u cilju je svaku radnju olakšati ili skratiti vrijeme obavljanja iste. Sve prisutnost mobilnih uređaja kod ljudi daje mogućnost da uz pomoć raznih aplikacija stvorimo uvjete za savladavanje raznih aktivnosti u što kraćem vremenu.

U ovom radu bit će opisana android aplikacija za automatsku naplatu parkiranja. Korisnik će kreirati vozilo ili više vozila u aplikaciji, vozila će biti pohranjena u bazi podataka. Nakon toga klikom na određeno vozilo korisniku će se otovriti prozoru u kojemu su integrirane google karte s dva markera koja predstavljaju prakrinu zonu, potom će iskočiti dijalog koji će predložiti broj zone u kojoj se korsinik nalazi, u slučaju da je van označene zone korisnik ručno unosi broj zone. Zatim korisnik otvara SMS aplikaciju u koju će se proslijediti određeni parametri i izvršiti naplata.

U drugom poglavlju opisana je već postojeća infrastuktura parkiranja u gradu Osijeku, prikazane su postojeće aplikacije i njihove mogućnosti, prikazan je pregled problema kod mobilne aplikacije i idejno rješenje aplikacije. Treće poglavlje prikazuje model aplikacije za automatsku naplatu parkiranja u kojoj je opisano korisničko sučelje, potencijalni korisnici, baza, geolokacija i sam postupak naplate. Četvrto poglavlje opisuje programsko rješenje kroz korištenu okolinu i tehnologiju, programsko rješenje korisničkog sučelja, neke binte djelove programa, arhitekture mobilne aplikacije i slojeva aplikacije. Peto poglavlje opisuje način korištenja aplikacije i njeno ispitivanje.

1.1 Zadatak završnog rada

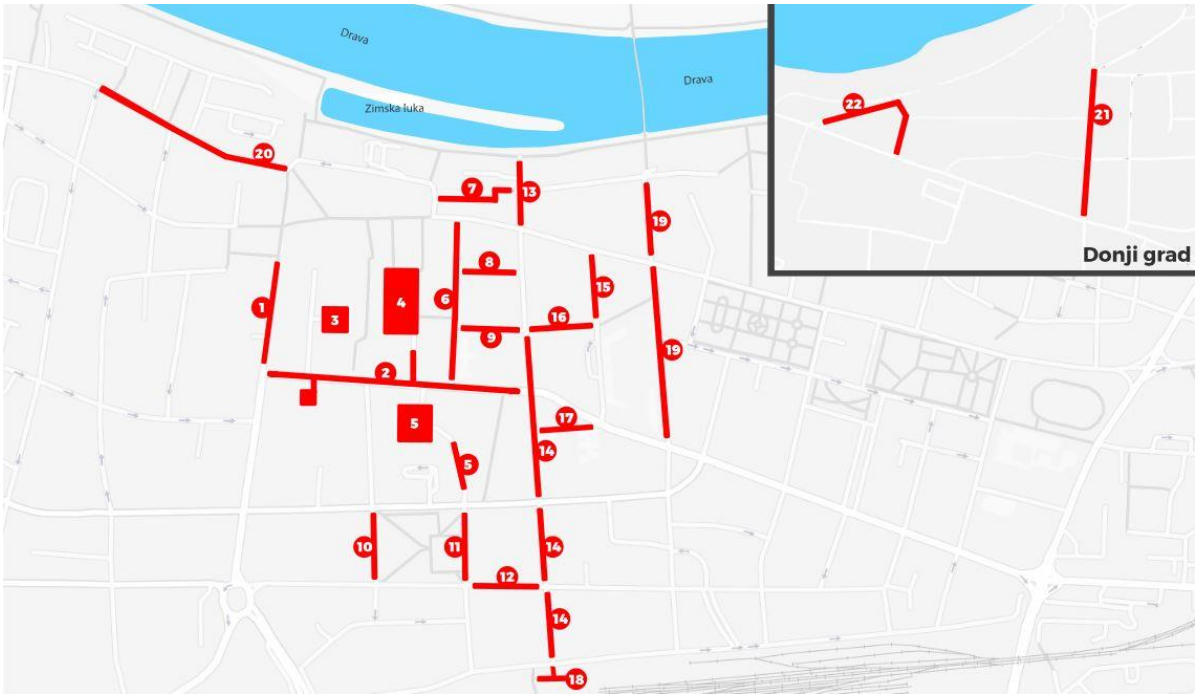
U radu je potrebno opisati preduvjete i korake automatske naplate parkiranja na temelju geolokacije, te razraditi model definiranja načina plaćanja po zonama. Nadalje, treba opisati potrebne programske okoline i tehnologije i programski ostvariti korisničko sučelje, bazu podataka i implementirati izabrani način plaćanja koji uzima u obzir lokaciju parkiranja. Ostvarenu aplikaciju treba ispitati na odgovarajućim primjerima s gledišta korisničkog iskustva i učinkovitosti programskog koda.

2. IZAZOVI KOD NAPLATE PARKIRANJA

Izazovi za lokalnu upravu i komunalne redare su nesmetano kretanje većeg broja gostiju i vozila kako u samom gradu tako i u staroj gradskoj jezgri Tvrđi, kao i u Zračnoj luci Osijek. Također je izazov razvoj prometne infrastukture koja bi zadovoljila potrebe lokalnog stanovništva, urbanističkog plana, potrebe većeg broja gostiju te lokalne institucije. Parkirališni prostor jedan je od ključnih preduvjeta koje treba zadovoljiti kako bi se nesmetano regulirao promet u satima visoke frekvencije vozila.

2.1 Infrastuktura parkiranja

Prema[1], gradsko poglavarstvo grada Osijeka 1993. godine počinje razmatrati model naplate parkiranja na temelju austrijskog modela i iskustva. Godine 2002. tvrtka Elektromodul uvodi naplatu parkiranja mobitelom. Danas se također koristi taj model s podjelom na zone, gdje svaka zona ima određeni broj na koji se pomoću SMS-a izvršava naplata parkiranja. U gradu Osijeku poznajemo tri parkirne zone. Prva zona obuhvaća dvadeset i dvije najprometnije ulice užeg centra grada. Druga zona obuhvaća devetnaest ulica, dok treća zona obuhvaća tek tri ulice. U nastavku ulice prve zone bit će prikazane na slici 2.1. .

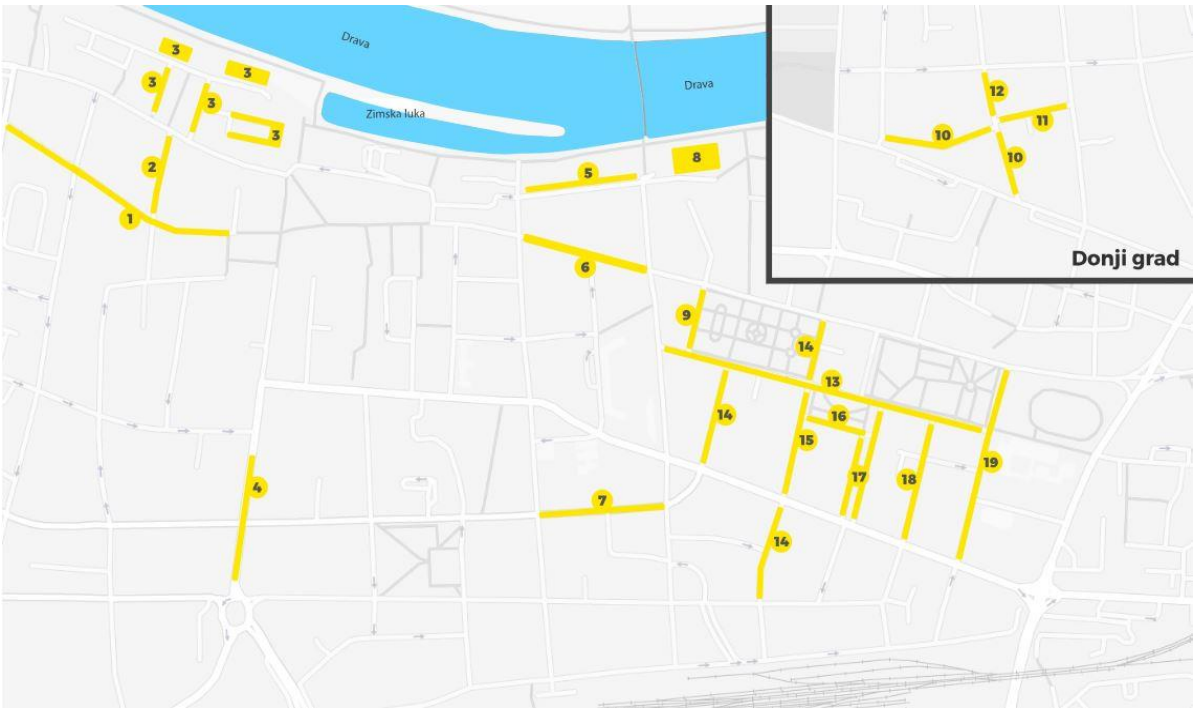


Sl. 2.1. Prva zona parkiranja[1]

Prva zona parkiranja obuhvaća sljedeće ulice:

1. Županijska ulica (od Trga Pape Ivana Pavla II do ulice Hrv. Republike)
2. Ulica Hrvatske Republike (uklj. Pariralište iza kbr. 41 i 43 te Vijenac Jakova Gotovca)
3. Trg Lava Mirskog
4. Vijenac Paje Kolarića
5. Ulica Hrvatske Republike 15-19h (dvorište i istočno od zgrade Gundulićeva 32-40)
6. Ulica Lorenza Jegera
7. Šetalište Petra Preradovića
8. Ulica Ivana Adamovića
9. Školska ulica
10. Zrinjevac zapad
11. Zrinjevac istok
12. Ulica Adama Reisnera
13. Šetalište Vjekoslava Hengla
14. Ulica Stjepana Radića
15. Ulica Dragutina Neumana
16. Trg Ljudevita Gaja – sjever
17. Trg Ljudevita Gaja – jug
18. Trg Lavoslava Ružičke
19. Ulica A. Stepinca (od Šetalište kardinala F. Šepera do Trga baruna F. Trenka)
20. Ulica J. J. Strossmayera
21. Ulica J. Huttlera i Park kraljice K. Kosača (od Ulice cara Hadrijana do Crkvene ulice)
22. Ulica Franje Mückea i Christiana Monspergera

Na slici 2.2. prikazane su ulice druge parkirne zone.



Sl. 2.2. Druga zona parkiranja[1]

Druga zona obuhvaća sljedeće ulice:

1. Ulica P. Pejačevića (od Ulice sv. Roka do Trga Pape I. Pavla II.)
2. Waldingerova ulica (od Ulice P. Pejačevićeva do Strossmayerove ulice)
3. Gornjodravska obala
4. Županijska ulica (od ul. Hrvatske Republike do Reisnerove)
5. Šetalište kardinala F. Šepera
6. Europska avenija
7. Gundulićeva ulica (od Radićeve ulice do Stepinčeve ulice)
8. Šetalište kardinala F. Šepera 1a-1c (parkiralište kod nebodera „Drava“ i HEP)
9. Park kralja P. Krešimira IV. – zapad
10. Trg Bana Jelačića
11. Ulica Ive Tijardovića
12. Ulica Z. J. Jovanovića
13. Ulica kralja Zvonimira
14. Ulica Franje Krežme

15. Zagrebačka ulica (od Ulice Adama Reisnera do Ulice kralja Zvonimira) i Park kralja P. Krešimira IV. – istok
16. Park kneza Branimira
17. Ulica O. Keršovanija (od Vukovarske ulice do Ulice kralja Zvonimira)
18. Ulica Dobriše Cesarića (od Vukovarske ulice do Ulice kralja Zvonimira)
19. Istarska ulica (od Vukovarske ulice do Europske avenije)

Na slici 2.3. prikazane su ulice treće parkirne zone.

Sl. 2.3. Treća zona parkiranja[1]

Treća zona obuhvaća sljedeće ulice:

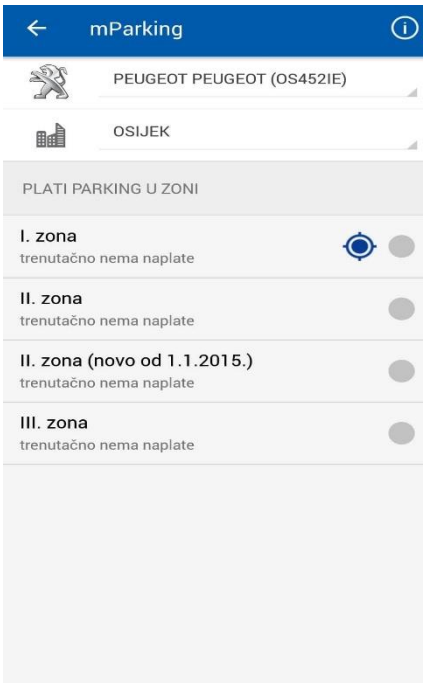
1. Pejačevićeva ulica (prostor bivše trgovine "Zeko")
2. Šamačka ulica - Zimska luka
3. Ulica Ivana Gundulića ispred kbr. 32 do 40

U svakoj ulici određene zone nalazi se automat za naplatu parkiranja. Prema[12], automati za naplatu imaju LCD na prednjoj strani i kratke upute koje vode korisnika kroz jednostavnu naplatu. Na LCD zaslonu jasno piše datum, vrijeme i zona u kojoj je aparat smješten. Naplata se izvršava

kovanicam od 50 lp, 5, 2 i 1 kunu. Automat je napajan uz pomoć solarnog panela, ali ima mogućnost spajanja na električ nu mrežu. Uz određenu cijenu zone može se kupiti karta i za nekoliko sati, parkirna karta može se kupiti i SMS-om, odnosno ako korisnik nije kupio kartu računa se kao da je kupio dnevnu kartu i ostavlja mu se kazna u iznosu cijene dnevne karte od 72 kune.

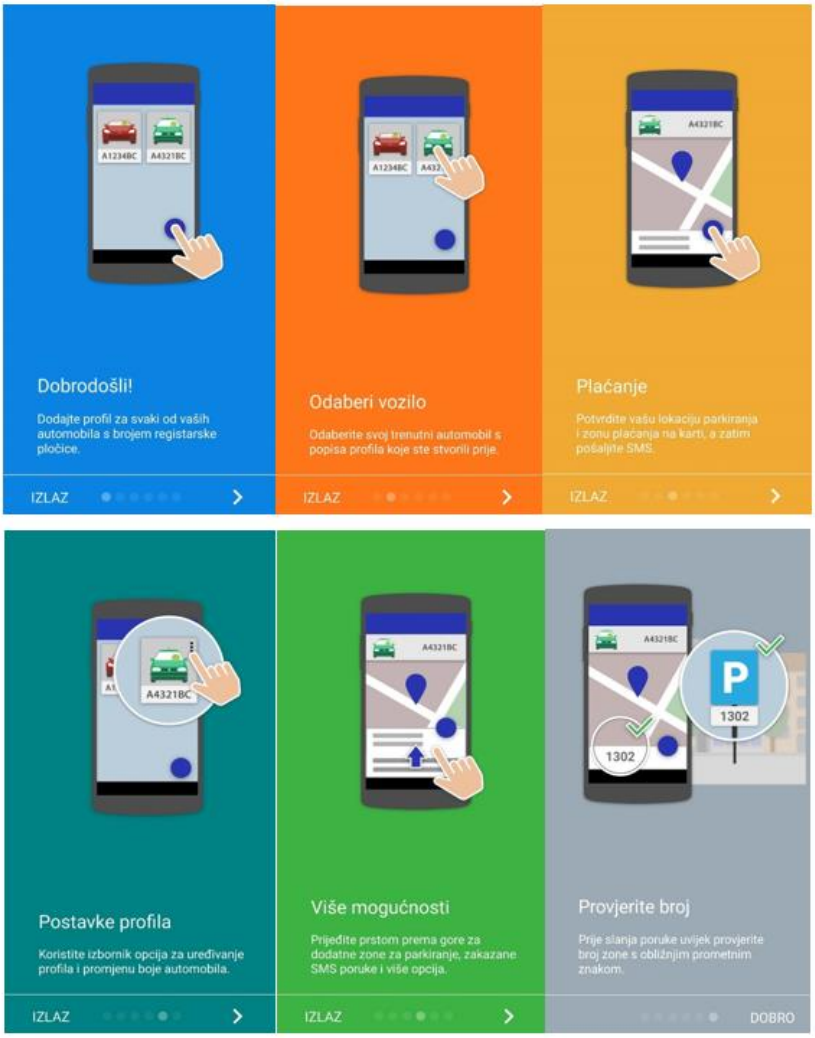
2.2 Postojeće aplikacije

HAK-ova aplikaciju za pametne telefone čija je jedna od uloga „mParking“ za plaćanje parkiranja. Ima mogućnost automatske detekcije grada po GPS lokaciji korisnika, automatsku detekciju parkirne zone u Zagrebu. Aplikacija sadrži informacije o cijeni i trajanju parkirne karte, cijeni dnevne karte, dopuštenom vremenu zadržavanja i ostalim podacima o koncesionaru naplate parkiranja. Naplata parkiranja je samo jedan segment HAK-ove aplikacije za pametne telefone, ona također sadrži popis najbližih benzinski posataja u Hrvatskoj, sveobuhvatan popis interesnih točaka u Hrvatskoj, cijene goriva u Europi, alat za pomoć pri pronalasku svog automobila parkiranog u nepoznatoj sredini „Gdje mi je auto?“, pregled cestarina na kompletnoj mreži autocesta u Hrvatskoj, olakšano kontaktiranje HAK-a i važnih službi, stanje na cestama, slike uživo s više od 150 kamera u Hrvatskoj, popis radarskih kontrola, pomoć na moru, interaktivna karta koja sadrži planer putovanja i većinu funkcionalnosti desktop verzije karte. Sučelje HAK-ove aplikacije nalazi se na sl. 2.4.



Sl. 2.4. Sučelje HAK-ove aplikacije za naplatu parkiranja

Aplikacija „SMS Parking“ ima isključivo funkciju naplate parkiranja. Naplata se odvija tako da možemo stvoriti više profila, odnosno više automobila s različitim registarskim oznakama, zatim odabiremo jedno vozilo s popisa profila nakon toga otvara nam se sučelje s google kartama i ponudi nam se zona. U postavkama postoji usluga za premium korisnika što bi značilo da se za 9,00 HRK može ukloniti sve oglase na godinu dana. Može se kontrolirati jezik i državu, ažuriranja i obavijesti, SMS postavke i osobne podatke. Jedna zanimljiva funkcionalnost kod SMS postavki je ta da nudi mogućnost izrade novog profila na temelju dolaznih SMS poruka s registarskim tablicama. Na slici 2.5. nalaze se neke od funkcionalnosti aplikacije „SMS Parking“.



Sl. 2.5. Mogućnosti aplikacije „SMS Parking“

2.3 Problemi kod aplikacije

Problemi koji su mogući kod aplikacije za automatsku naplatu parkiranja povezani su s geolokacijom. Naime aplikacija kao dio funkcionalnosti ima pronalaženje zone u kojoj se korisnik nalazi i prema tome slijedi naplata po toj zoni i određenom vremenu na lokaciji prilikom naplate. Do gotovo sigurne pogreške može doći kod preklapanja zona ili kada su zone jedna do druge.

Prema[2], negativan utjecaj na točnost GPS-a mogu imati ionosferski i troposferski utjecaja usporavanja signala satelita kroz atmosferu, višestazni prijam koji se događa pri reflektiranju GPS signala od raznih objekata, pogreške sata prijemnika, orbitalne pogreške odnosno netočni podaci o položaju satelita, zasjenjivanje se događa kada su dva satelita na istom pravcu ili su jedan u odnosu na drugoga pod malim kutem, namjerna degradacija satelitskog signala od strane vojske.

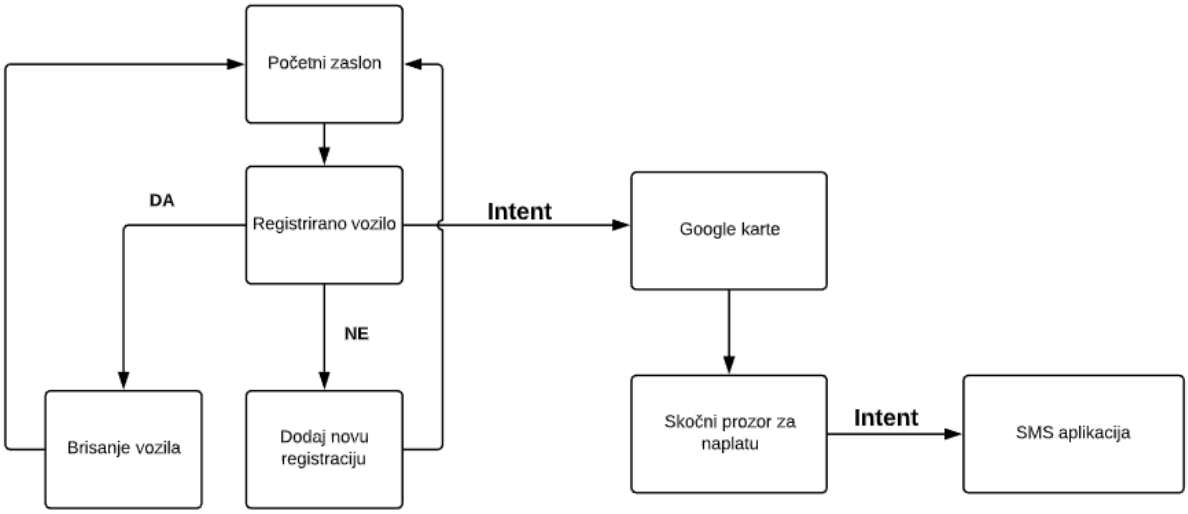
2.4 Idejno rješenje projekta

Ideja mobilne aplikacije za automatsku naplatu parkiranja je da korisnik koji upravlja vozilom ima aplikaciju koja bi ubrzala proces naplate parkiranja i time umanjila potrebno vrijeme za obavljanje te fizičke akcije, pojednostavljenje naplate i ekološki faktor jer ne bi dolazilo do ispisa papira kao kod plaćanja na automatu. Korisniku je prikazana parkirna zona u kojoj se nalazi, on je može potvrditi, ponuđen mu je broj zone u kojoj se nalazi i pripremljena SMS poruka s registracijskim oznakama motornog vozila.

3. MODEL MOBILNE APLIKACIJE ZA AUTOMATSKU NAPLATU PARKIRANJA

3.1 Korisničko sučelje

Korisničko sučelje daje korisniku jednostavan i jasan pristup svim funkcionalnostima koje aplikacija pruža. Jednostavnost aplikacije vrlo je bitna kako bi korisnik u što kraćem vremenu savladao osnovne akcije koje aplikacija nudi. Ono mora biti estetski prihvatljivo kako bi privuklo i zadržalo korisnika. Pruža krajnji komunikacijski kanal između programera koji je razvijao aplikaciju i korisnika koji se istom služi. Sam dizajn korisničkog sučelja može pridonijeti i povećati vrijednost aplikacije kao i pozitivno iskustvo korisnika interakcijom kroz sučelje te jasne akcije i smjer u kojem aplikacija radi. Na sl. 3.1 nalazi se dijagram toka aplikacije.



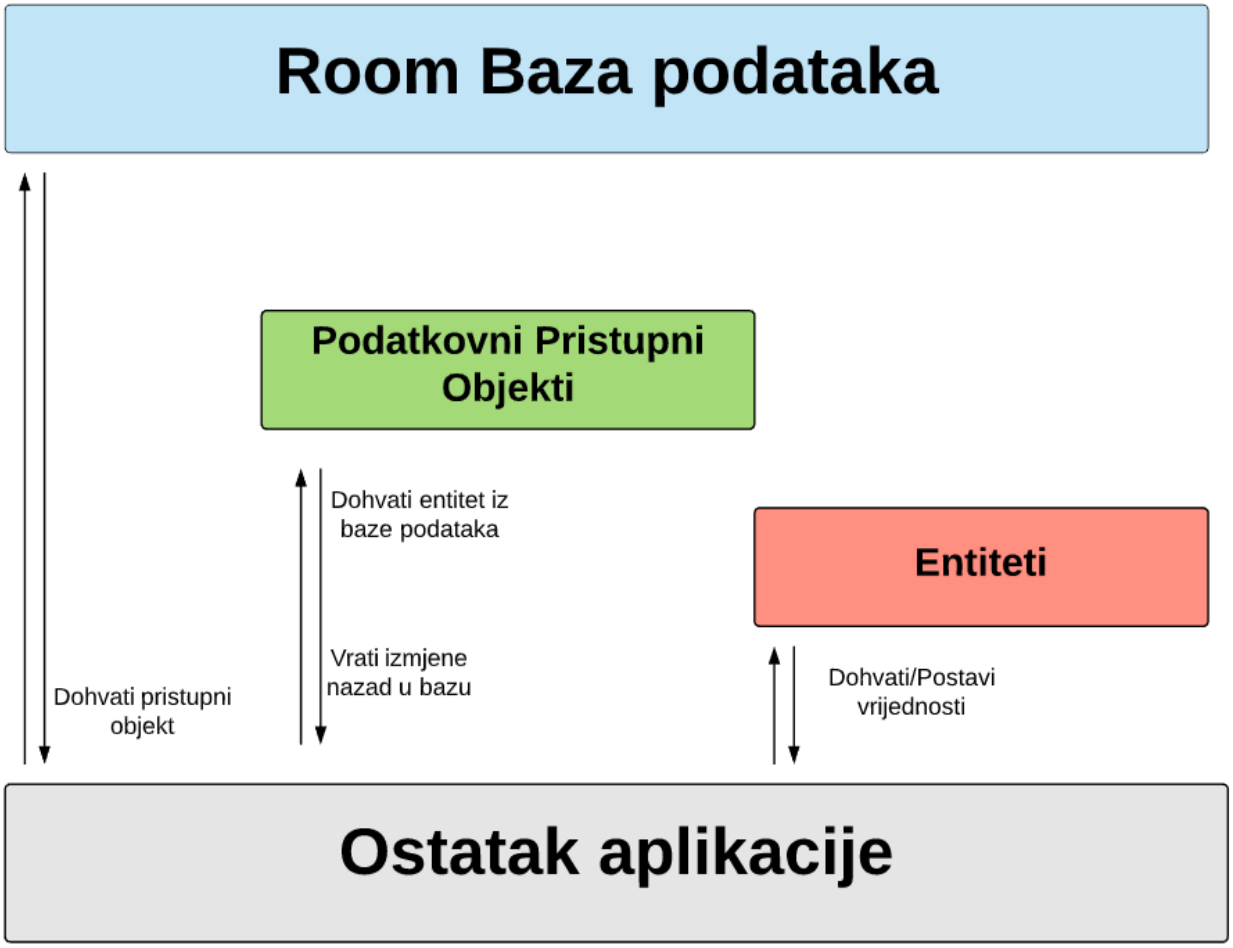
Sl. 3.1 Dijagram toka aplikacije

3.2 Baza podataka

Potreba za bazom podataka kod mobilne aplikacije za automatsku naplatu parkiranja proizlazi iz potrebe da pohranimo određene podatke o registriranom vozilu. Nakon što su ti podaci pohranjeni u bazu i kada je kreirano vozilo, korisnik može vrlo brzo naplatiti parkiranje.

U ovoj andorid aplikaciji imamo mali broj podataka koje trebamo spremiti zato je pogodna Room biblioteka (*engl. library*) koja je zapravo sloj iznad SQLite baze podataka. U slučaju da želimo skalirati aplikaciju tako da dodamo neke od funkcionalnosti za koje bi nam trebala neka komunikacija s mrežom, u slučaju da mobilna aplikacija nema pristup internetu podatke spremi lokalno, a idući puta kada aplikacija uspostavi vezu s internetom ona automatski uskladi promijenjene podatke.

U aplikacija će se koristi Room baza podataka kako bi dobila pristupne objekte ili Podatkovni pristupni objekti (*engl. Data Access Objects,* Dao*)* koji su povezani s bazom podataka. Zatim aplikacija koristi te objekte kako bi dobila pristup entitetima iz baze podataka i spremila izmjene na njima nazad u bazu podataka. Naposljetku aplikacija koristi te entitete kako bi dohvaćala i postavljala podatke u tablici koja odgovara tom entitetu[3]. Na slici 3.2 nalazi se arhitektura Room-a.



Sl. 3.2 Dijagram arhitekture Room-a

3.3 Korisnici

Prema podacima App Anniea tijekom 2017. godine preuzeto je više od 175 milijardi aplikacija, stoga svaka aplikacija ima svog korisnika. Potrebno je dobro analizirati tržište i vidjeti koji su potencijalni korisnici aplikacije, a to su najčešće osobe koje su vlasnici motornih vozila žive i rade u velikim gradovima gdje je potreba za uštedom vremena i parkiranjem od velikog značaja.

3.4 Geolokacija

Geolokacija je proces pronalaska, određivanja i pružanja točne lokacije određenog uređaja ili opreme koja ima pristup internetu. Geolokacija obično koristi Globalni sustav pozicioniranja (*engl. Global Positionig System,* GPS*)* i ostale povezane tehnologije kako bi utvrdila koordinate i mjere određenog uređaja. GPS određuje geografsku širinu i duljinu, koja opisuje položaj uređaja ili korisnika[4]. Svaka adresa pojedinačno je prikazana na karti, a skupno se nalaze na određenom prostoru gdje su grupirani kao država, grad, ulica, zgrada. Kada su tako grupirane određene adrese na karti, geolokacija određenog uređaja ulazi u te skupine te to omogućava potpunu sliku korisnika gdje se nalazi, jer ima referentne točke prema kojima si može vizualizirati svoju lokaciju.

Prema[5], postoje tri mrežna poslužitelja u Androidu, GPS lokacijski poslužitelj, ovaj poslužitelj određuje lokaciju koristeći satelite. Ovisan je o uvjetima, potrebno mu je određeno vrijeme kako bi vratili važu točnu lokaciju. Zahtijeva dopuštenje (*engl. permission)* android.permission.ACCESS_FINE_LOCATION.

Zatim postoji mrežni lokacijski poslužitelj. Ovaj poslužitelj određuje vašu lokaciju na temelju dostupnosti mobilnih tornjeva i WiFi pristupne točke. Zahtjev ili dopuštenje android.permission.ACCESS_COARSE_LOCATION ili android.permission.ACCESS_FINE_LOCATION.

Postoji i hibridni lokacijski poslužitelj koji kombinira GPS i WiFi pristupnu točku. Automatski određuje koji mu je dostupniji i pogodniji za trenutni pronalazak lokacije. U aplikaciji za automatsku naplatu parkiranja korišten je GPS lokacijski poslužitelj. Na sl. 3.3 prikazano je korištenje ACCES_FINE_LOCATION dopuštenja, što nam prema gore opisanome govori da se radi o GPS lokacijskom poslužitelju.

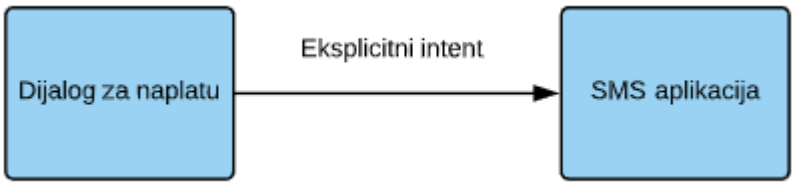
```
private fun setUpMap() {
    if (ActivityCompat.checkSelfPermission(
            context: this,
            android.Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(
            activity: this,
            arrayOf(android.Manifest.permission.ACCESS_FINE_LOCATION), LOCATION_PERMISSION_REQUEST_CODE
        )
        return
    }
```

Sl. 3.3 GPS dopuštenje

Postoje globalni i lokalni sustavi lociranja. GPS globalni sustav pozicioniranja razvijen od strane SAD-a sastoji je od trideset dva satelita lociranih u šest različitih orbitalnih ravnina na visini od dvadeset tisuća kilometara.[2] Točnost ovoga sustava je visoka s odmakom od nekoliko metara što u slučaju mobilne aplikacije za automatsku naplatu parkiranja radi dobro s određenim rubnim slučajevima, kada bismo proširli aplikaciju i mapirali više zona koje su dodirne moguće je da bi došlo do problema pri određivanju točnosti u kojoj se zoni korisnik nalazi, ali zato postoji mogućnost ručne izmjene broja zone. Postoje problemi lociranja u određenim okolinama, u slučaju podzemnih garaža, visokih građevina, tunela, itd. Postoje tri glavne komponente koje čine ovaj sustav, GPS prijemnik, GPS satelit i određeni softver koji izračunava lokaciju korisnika.

3.5 Postupak naplate

Postupak naplate je vrlo jednostavan, koristi se već postojeća SMS aplikacija na mobilnom uređaju. Nakon što korisnik registrira vozilo njegova registracijska oznaka je spremljena u bazu, u idućem koraku aplikacija odredi u kojoj se zoni nalazi korisnik te otvori dijalog u kojemu su zapisane registracijska oznaka i prema lokaciji broj zone. U slučaju da se korisnik ne nalazi niti u jednoj označenoj zoni treba ispuniti polje s brojem zone ručno. Zatim korisnik klikom na „Prepare SMS“ pomoću eksplicitnog intenta otvara SMS aplikaciju u kojoj je automatski postavljen broj zone kao adresa slanja SMS-a i sadržaj SMS poruke je registracijska oznaka što je prikazano na sl. 3.4 dijagramom naplate. Kada korisnik pošalje poruku, dolazi mu obavijest u SMS aplikaciju kako je platio parkiranje i do kada vrijedi.



Sl. 3.4 Dijagram naplate

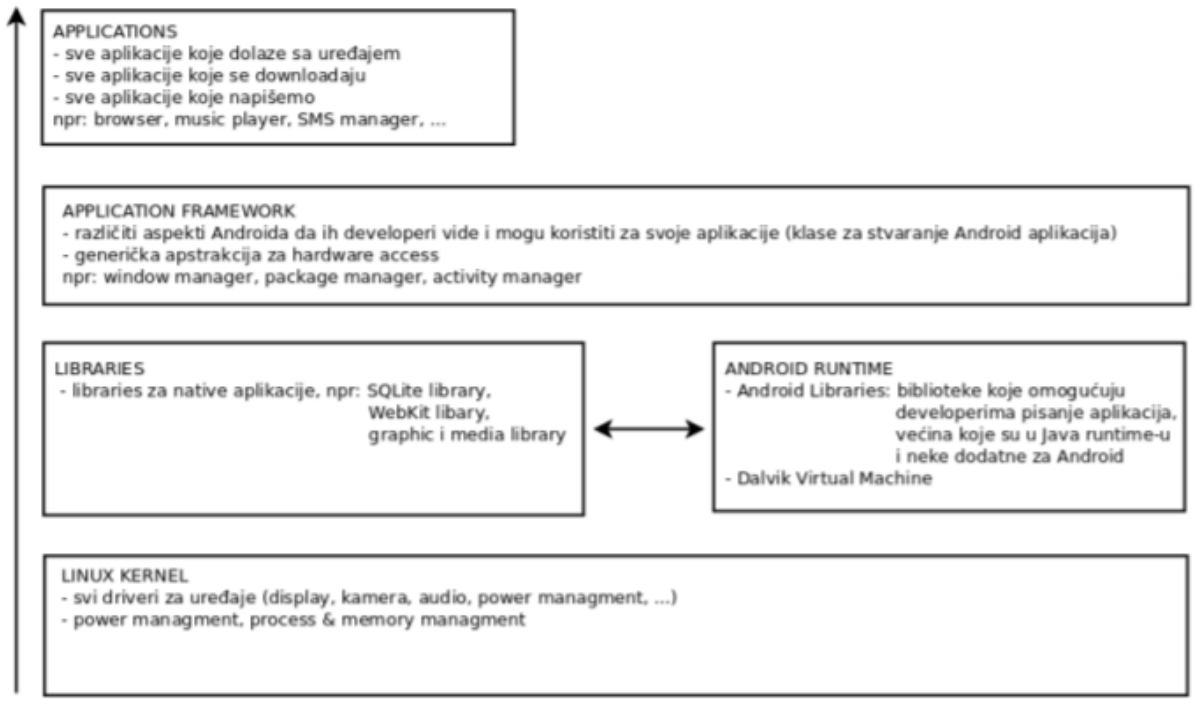
4. PROGRAMSKO RJEŠENJE

4.1 Okolina i tehnologija razvoja aplikacije

Aplikacija za automatsku naplatu parkiranja pisana je na Android operativnom sustavu u jeziku Kotlin. Kako bi se aplikacija napravila potrebno je razumijevanje i predznanje korištenja Andorid Studia kao i Kotlin programskog jezika.

4.1.1 Android Studio

Android Studio je integrirano razvojno okruženje IDE koje se pojavilo kao nasljednik Eclipse ADT. Prezuimanje Android Studia je besplatno, kako bi studio radio potrebno je imati instalaciju Java Development Kit-a, nakon verzije studija 2.3.0 JDK dolazi zajedno. Android operativni sustav je otovrenog koda, što znači da mu svi mogu pristupiti i modificirati, pisan je u Javi na jezgri Linux. Prva stabilna verzija Android Studia objavljena je u prosincu 2014. godine[6]. Prema[7], arhitektura Android sustava podijeljena je na pet djelova u četiri sloja:
1. Linux Kernel - Android koristi Linux za rukovanje procesima, sigurnost, upravljanje memorijom, korištenje snage. Također ima sve drivere za uređaje. Ono je apstraktni sloj između sklopovlja i softvera.
2. Libraries, Android Run Time - Skup biblioteka SQLite, WebKit, grafika, media, LibWebCore. Svaka aplikacija radi sa zasebnom instancom DalvikVM, Dalvik ovisi o nižem sloju Linux Kernel za pomoć s osnovnim funkcijama kao što su najniže upravljanje memorijom i nitima.
3. Application Framework - Omogućuje razvoj aplikacija, daje programeru abstraktan pristup sklopovlju i ostalim aspektima Android sustava. Npr. Upravitelj aktivitijima (*engl. Activity manager)*, Upravitelj paketima (*engl. Package manager)*.
4. Applications - Ovaj sloj sadrži sve aplikacije koje dolaze uz mobilni uređaj, sve aplikacije koje su skinute i sve aplikacije koje programer napiše.



Sl. 4.1 Slojevi Android arhitekture[7]

4.1.2 Programski jezik Kotlin

Prema[8], nakon pojave Kotlina 2011. godine nije ušao u širu primjenu, ali nakon objavljivanja prve stabilne verzije 2016. godine ono doživljava svoju ekspanziju i širu primjenu ne samo za kreiranje Android aplikacija već i za druge platforme. Kotlin je objektno orjentirani jezik, vrlo koncizan i snažan. Nastao je kako bi pružio sigurniju alternativu Javi, IDE Android Studio prepoznat će i sam pretvoriti kod iz Jave u Kotlin, isto tako može se pisati mješoviti kod Jave i Kotlina. Najčešća primjena je razvoj Android aplikacija, ali može razvijati aplikacije i za ostale platforme Kotlin se pokreće pomoću Java virtualnog stroja (*engl. Java Virtual Machine, JVM)*, može biti kompajliran i u Java Script pomoću Virtualnog stroja niske razine (*engl. Low Level Virtual Machine, LLVM)*. Ako je kompajliran na LLVM može se pokrenuti i kao web aplikacija.

U Javi je postojao problem da se za svaki potencijalni null-objekt moralo provjeravati i pisati dosta „boilerplate“ koda, kreiranje objekata pogleda moralo se prvo pristupiti određenom pogledu i zatim na njemu pozvati metoda *findViewById()* preko id-a određenog elementa na pogledu, zatim je još bilo potrebno „kastati“ kako bi se znalo o kojem se elemntu radi. Kotlin odlično rukuje takvim stvarim, ako postavimo da je određeni objekt nulabilan on i ako je „null“ neće se ništa dogoditi jer

kada se kompajlira provjeravaju se takve stvari što pridonosi sigurnosti samog jezika i uvelike olakšava rad programeru. Također ako nam je pogled povezan s određenim aktivitijem ili fragmentom lako se može pristupati i raditi s elementima samo prema njihovom imenu. Dodatne prednosti Kotlina bit će opisane u sklopu programskih rješenja, kako se budu pojavljivale u programu.

4.1.3 XML

Za kreiranje sučelja korišten je jezik za označavanje podataka (*engl. Extensible Markup Language*, XML). XML je jezik vrlo sličan HTMLu, služi za opisivanje podataka. Prema[9], vrlo je jednostavan za čitanje i pisanje kako čovjeku tako i stroju. Osim za kreiranje sučelja postoji još XML datoteka u projektu vrijednosti, stringovi, drawable, dimenzije i manifest. U Manifestu moramo definirati sve što aplikacija koristi, sve *activitye*, dopuštenja i slično. Na sl. 4.2 prikazan je primjer XML rasporeda.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:padding="15dp"
              android:layout_width="match_parent"
              android:layout_height="match_parent">
    <TextView
            android:id="@+id/paymentDialogHeading"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            style="@style/HeadingTextView"
            android:textAllCaps="true"
            android:layout_marginBottom="15dp"
            android:text="Payment"/>
    <EditText
            android:id="@+id/registration"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Enter your registration"
            android:layout_marginBottom="15dp"
            android:inputType="text"
    />
    <EditText
            android:id="@+id/zonePaymentNumber"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/numberDialogHint"
            android:layout_marginBottom="15dp"
    />
```

Sl. 4.2 XML layout

4.2 Programsko rješenje korisničkog sučelja

Svaki zaslon pisan je u jeziku za uređivanje podataka koji nam služi za deklariranje elemenata. Sučelja su napravljena kombinacijom ograničenog rasporeda (*engl. constraint layout*) elemenata i linearnog rasporeda (*engl. linear layout*) elemenata.

Na slici 4.5 prikazan je programski dio kako je izveden početni zaslon i izgled početnog zaslona koji je prikazan odmah nakon otvaranja aplikacije. U njemu se nalazi spremište u koje će biti pohranjena vozila s registarskim oznakama i odabranim imenom vozila nakon registracije. Programski dio sadrži ograničeni raspored (*engl. Constraint Layout*) u kojemu su raspoređeni ostali elementi zaslona. Ograničeni raspored je nasljeđe relativnog rasporeda elemenata, gdje su također elementi relativni jedni u odnosu na druge, ali je puno jednostavnije napraviti kompleksno korisničko sučelje. Koristeći samo četiri lijnije koda može se definirati točno gdje se koji element nalazi u odnosu na roditelja i ostale elemente. Na ovom zaslonu korišteni su elementi *Recycler view, Text view, Progress bar, Floating Action Button. Recycler view* je elemnt koji u sebi prikazuje određenu listu elemenata, ono je napredna verzija *List View-a.* Razlikuju se po tome što *Recycler view* ima mogućnost kreiranja uređenja određenog elementa u listi, taj posebno dizajnirani element u list naziva se držać (*engl. holder*). Ovaj element poravnat je tako da zauzima cijeli prostor zaslona, definiran je pomoću dva svojstva, visine i širine. Za *Recycler view* potrebno je dodati ovisnost (*engl. dependencie*) koji je prikazan na sl. 4.3.

```
//recycler view
implementation 'androidx.recyclerview:recyclerview:1.1.0-beta04'
```

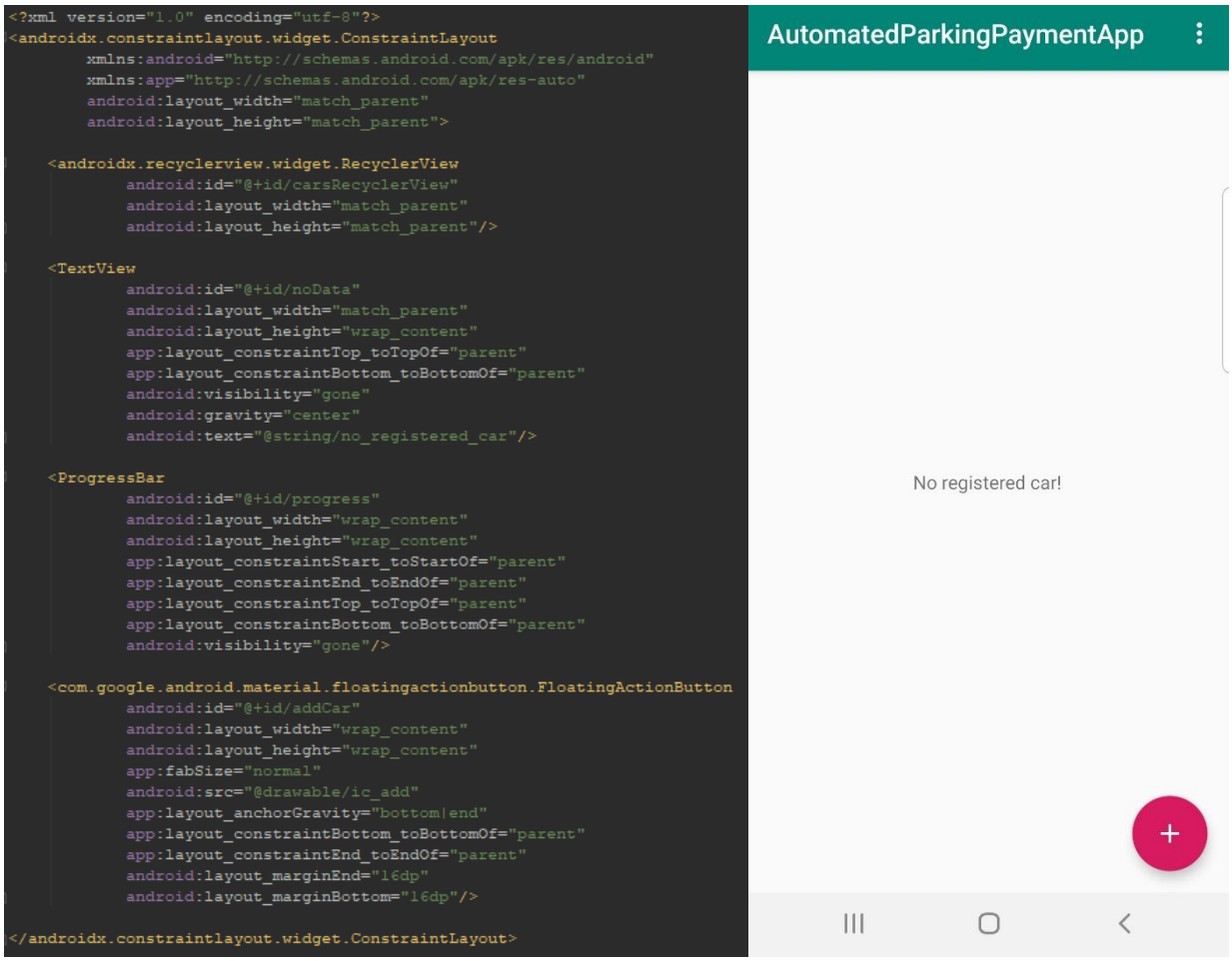
Sl. 4.3 Ovistnost *Recycler View-a*

Sljedeći element je tekstualni pogled (*engl. Text View)* čija je uloga da ispiše na zaslonu kako nema „No registered car!“ što znači da nema registriranog vozila. Definirano je pomoću više svojstava visine, širine, gravitacije koja je orijentirana prema centru zaslona, vidljivosti i dva svojstva koja su dio ograničenog raporeda jedan određuje da je vrh elementa ograničen s vrhom elementa roditelja, a drugi da je dno elementa ograničeno s dnom elementa roditelja. Traka napretka (*engl. Progress Bar)* je element korisničkog sučelja koji govori o napretku neke akcije, ograničen je sa sve četiri strane s roditeljem i ima postavljenu vidljivost. Zauzima prostor onoliko koliko je sami element. Lebdeći

akcijski gumb (*engl. Floating Action Button)* je element koji „lebdi“ nad samim zaslonom, klikom na njega otvara se novi zaslon. Za njega je također potrebno dodati ovisnost u *build.gradle* datoteku, ovisnost je prikazana na Sl. 4.4.

```
//material
implementation 'com.google.android.material:material:1.1.0-alpha10'
```

Sl. 4.4 Ovisnost za *Floating Action Button*

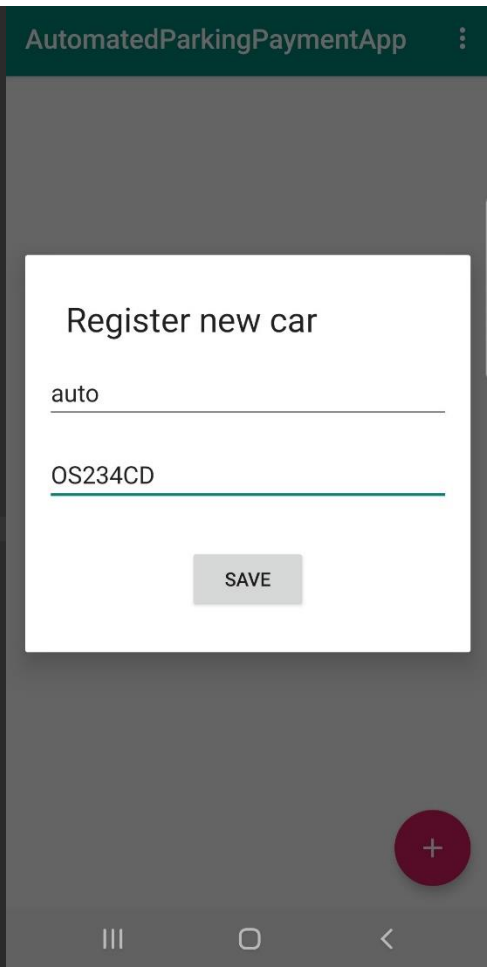


Sl. 4.5 Početni zaslon

Nakon klika na *Floating Action Button* otvara se zaslon za registraciju namijenjen je kako bi korisnik izradio profil vozila. Ime vozila kako bi korisnik znao o kojem je vozilu riječ kada bude htio izvršiti naplatu. Registracijske oznake koje su potrebne prilikom naplate kako bi sustav imao uvid o kojem se vozilu radi te za koje je vozilo uplaćena parkirna karta. Na sl. 4.6 prikazano je sučelje i programsko rješenje dijaloga za dodavanje novog vozila (*engl. fragment dialog new car*). Elementi dijaloga raspoređeni su u linearnom odnosu jedna prema drugome. Linearni raspored (*engl. Linear Layout*) koristi opciju orijentacije zaslona, u slučaju dijaloga za dodavanje novog vozila orijentacija je vertikalna. Tekstualni pogled opisuje naslov i daje korisniku ideju što može očekivati na ovom zaslonu. Korištena su dva nova elementa uređenje teksta (*engl. Edit Text*) koji služi za unos teksta od strane korisnika korišteno je svojstvo nagovještaj (*engl. hint*) kako bi se u razvoju zaslona moglo vidjeti kako će element izgledati. Također upotrebljen je gumb (engl. Button) kao element čija je uloga spremiti unesene podatke u bazu. Ovo je element korisničkog sučelja na koji korisnik može kliknuti kako bi izveo određenu akciju.

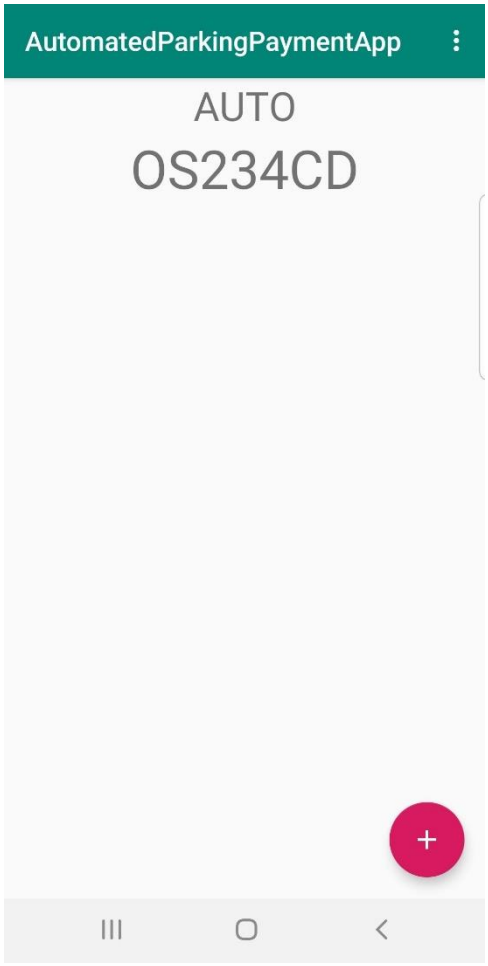
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:padding="15dp"
              android:layout_width="match_parent"
              android:layout_height="match_parent">
    <TextView
            android:id="@+id/newCarHeading"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            style="@style/HeadingTextView"
            android:layout_marginBottom="15dp"
            android:text="Register new car"/>
    <EditText
            android:id="@+id/newCarTitleInput"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Enter cars name"
            android:layout_marginBottom="15dp"
            android:inputType="text"
            />
    <EditText
            android:id="@+id/newCarRegistrationInput"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Enter registration"
            android:layout_marginBottom="15dp"
            android:inputType="text"
            />

    <Button
            android:id="@+id/saveCarRegistration"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="15dp"
            android:layout_marginBottom="15dp"
            android:layout_gravity="center"
            android:text="save"/>
</LinearLayout>
```



Sl. 4.6 Dijalog za registraciju (*engl. fragment dialog new car*)

Nakon što je korisnik izvršio registraciju na početnom zaslonu prikazano je registrirano vozilo ili više njih. Na sl. 4.7 prikazan je početni zaslon s registriranim vozilom.



Sl. 4.7 Početni zaslon s registriranim vozilom

Na sl. 4.8 nalazi se fragment koji je preko konteksta povezan na *activity* u kojemu postavljene google karte na ovaj fragment. Prema[10], na fragment se može gledati kao dio aktivitija koji ima svoj životni ciklus (*engl. Lifecycle*), prima svoje ulazne događaje i može se uništiti za vrijeme dok aktiviti i dalje radi u svom životnom ciklusu. Jedan aktiviti može sadržavati više fragmenata, isto tako fragment može sadržavati još jedan fragment.

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
        xmlns:tools="http://schemas.android.com/tools" xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/map"
        tools:context=".ui.activities.maps.MapsActivity"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        app:layout_constraintHorizontal_bias="1.0"/>
```

Sl. 4.8 *Support Map Fragment*

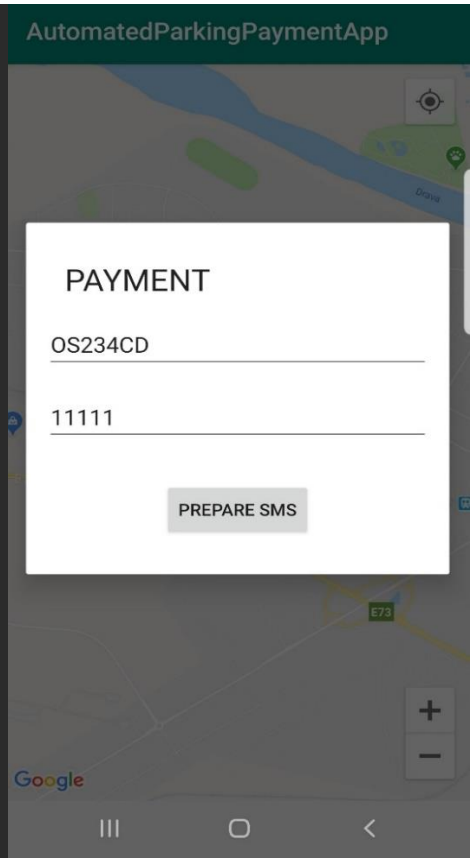
Na sl. 4.9 nalazi se pregled zaslona koji ima integrirane google karte koji će biti prikazan nakon klika na registrirano vozilo. Taj zaslon daje uvid u korisnikovu trenutnu geolokaciju i prikaz dvije označene zone u gradu Osijeku.

Sl. 4.10 Zaslon na kojemu je prikazana google karta

Na zaslonu na kojemu su prikazane google karte iskočit će dijalog koji ima opciju izmjene registracijske oznake te broja zone na koji će biti poslana registracijska oznaka. Ukoliko se korisnik ne nalazi niti u jednoj od označenih zona polje za broj zone ostat će prazno i korisnik će morati unijeti broj ručno. Sučelje opisanog dijaloga i programsko rješenje nalaze se na sl. 4.11. Elementi dijaloga raspoređeni su u linearnom rasporedu (*engl. Linear layout*) jedna prema drugome. Struktura ovog dijaloga je ista strukturi dijaloga za dodavanje novog vozila. Klikom na gumb (*engl. Button*) šalje se eksplicitni intent u već postojeću SMS aplikaciju na mobilnom uređaju u koju su proljeđeni sadržaj poruke odnosno registracija i adresa na koju se šalje poruka odnosno broj zone.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:padding="15dp"
              android:layout_width="match_parent"
              android:layout_height="match_parent">
    <TextView
            android:id="@+id/paymentDialogHeading"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            style="@style/HeadingTextView"
            android:textAllCaps="true"
            android:layout_marginBottom="15dp"
            android:text="Payment"/>
    <EditText
            android:id="@+id/registration"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Enter your registration"
            android:layout_marginBottom="15dp"
            android:inputType="text"
    />
    <EditText
            android:id="@+id/zonePaymentNumber"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Enter payment number manually."
            android:layout_marginBottom="15dp"
    />

    <Button
            android:id="@+id/prepareSMS"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="15dp"
            android:layout_marginBottom="15dp"
            android:layout_gravity="center"
            android:text="Prepare SMS"/>
</LinearLayout>
```



Sl. 4.11 Zaslon s google kartama i pripremnim dijalogom

4.3 Opis bitnih djelova programa

4.3.1 Manifest

Android Manifest datoteka, nalazi se u korijenu same strukture programskog stabla, manifest daje sve osnovne informacije Android gradivnim alatima (*engl. Andorid Build Tools*), Android operativnom sustavu i *Google Play-u*. Prema[11], između mnogo ostalih stvari manifest datoteka zahtjeva deklariranje sljedećih stvari, ime paketa aplikacije, Android gradivni alati koriste ime paketa kako bi odreditli lokaciju programskih entiteta kada grade aplikaciju. Kada se aplikacija pakira gradivni alati zamjenjuju ovo ime s identifikacijom iz *build.gradle* datoteke, ova identifikacija je jedinstvena oznaka u sustavu i na *Google Play-u*. Zatim je potrebno deklarirati komponente aplikacije koje uključuju sve aktivitije, servise, *broadcast recivere* i poslužitelje sadržaja. Svaka komponenta mora sadržavati svoje ime i je li datoteka pisana u Javi ili Kotlinu. Isto tako može se odrediti opis koja konfikuracija mobilnog uređaja može korisiti aplikaciju i koja vrsta namjere(*engl. Intent*) kako bi se znalo na koji način će se određena komponenta pokrenuti. Također potrebno je deklarirati dopuštenja (*engl. permission*) kako bi aplikacija mogla pristupiti zaštićenim djelovima sustava ili aplikacije, deklarira i dopuštenja koje ostale aplikacije moraju imati kako bi pristupile sadržaju ove aplikacije. Ako se korisiti Android Studio IDE, manifest datoteka kreirana je sama i mnoge od ovih komponenti se automatski kreiraju za vrijeme razvoja aplikacije. Na sl. 4.12 prikazana je Manifest ove aplikacije.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools" package="hr.ferit.automatedparkingpaymentapp">

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <application
            android:name=".AppParking"
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="@string/app_name"
            android:roundIcon="@mipmap/ic_launcher_round"
            android:supportsRtl="true"
            android:theme="@style/AppTheme"
            tools:ignore="AllowBackup, GoogleAppIndexingWarning">
        <activity android:name=".ui.activities.main.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>

        <activity
                android:name=".ui.activities.maps.MapsActivity"
                android:screenOrientation="portrait">
        </activity>

        <meta-data
                android:name="com.google.android.geo.API_KEY"
                android:value="AIzaSyC8eaqK6xUBqqtHak2fHwjNK6ajSnfjOj4"/>
        <uses-library android:name="org.apache.http.legacy" android:required="false" />
    </application>
</manifest>
```

Sl. 4.12 Manifest

4.3.2 Glavni aktiviti

Glavni aktiviti (*engl. Main Activity*) u Manifestu je opisan kao glavni i početni prikaz. Iz razloga da svaki aktiviti ne nasljeđuje *AppCompatActivity()* napravljena je klasa *BaseActivity()* koja jedina nasljeđuje AppCompat, a svi ostali aktivitiji nasljeđuju tu klasu. Ona sadrži dvije abstraktne metode s*etUpUi()* koja u sebi koristi funkciju iz *BaseActivity(),* ta funkcija prima fragment koji će biti postavljen na glavni aktiviti u okvirni raspored koji je definirat u XML-u i *getLayoutResourceId()* povezuje aktiviti s određenim sučeljem. U prepisanoj metodi *onCreateOptionMenu()* „napuhujemo“ (*engl. inflate*) meni koji smo kreirali s njegovim opisom u xml-u. Zatim je potrebno postaviti *listener* na taj fragment, a to radimo u metodi *setListeners().* Kada je postavljen *listener* potrebno je odrediti što se događa klikom na određenu opciju, to se postavlja u prepisanoj metodi *onOptionsItemSelected().* U glavnom aktivitiju pojavlju se dva fragmenta to su *RegisteredCarsFragment i AddCarDialogFragment* koji će biti opisani u nastavku. Glavni aktiviti nalaz se sl. 4.13.

```
class MainActivity : BaseActivity() {

    private var listener: MenuMethods.RegistrationMethods? = null
    override fun getLayoutResourceId() = R.layout.activity_main

    override fun setUpUi() {
        showFragment(RegisteredCarsFragment.newInstance())
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        super.onCreateOptionsMenu(menu)
        menuInflater.inflate(R.menu.menu_registered_cars, menu)
        return true
    }

    fun setListeners(fragment: RegisteredCarsFragment){
        listener = fragment
    }

    override fun onOptionsItemSelected(item: MenuItem?): Boolean {
        if(item!!.itemId == R.id.clearAllCars) listener!!.clearAllCars()
        return super.onOptionsItemSelected(item)

    }

}
```

Sl. 4.13 Glavni aktiviti (e*ngl. MainActivity*)

4.3.3 Dijalog za dodavanje novog vozila

Dijalog fragment koji služi za dodavanje novog vozila u listu, dijalog je uređen u MVP arhitekturalnom obrascu. Kreirana je referenca na prezenter kojemu je u konstruktor predana *CarRoomRepository()* koji dalje komunicira s bazom, ali to ovaj pogled odnosno dijalog ne zna. U prepisanoj *onCreate()* metodi postavljaju se parametri dijaloga. U prepisanoj *onResume()* metodi uz pomoć prezentera postavlja se pogled na ovaj dijalog, metoda je definirana u ugovoru o kojemu će biti govora u sljedećem poglavlju. Dijalog mora biti povezan sa svojim odgovarajućim dizajnom u xml datoteci, to se postavlja u *onCreateView()* prepisanoj metodi. U prepisanoj *onStart()* metodi određeno je kako će se ovo sučelje rasporediti u rasporedu. Metoda *initListeners()* postavlja slušalac (*engl. listener)* na gumb spremi (*engl. save*) i izvršava spremanje auta koje će biti opisano u nastavku. Sve ove opisane metode osim *initListeners()* su metode životnog ciklusa fragmenta i u svakom ciklusu potrebno je nešto napraviti, na primjer prilikom kreiranja pogleda potrebno je odrediti s kojom xml datotekom je povezan, zatim nakon što je pogled kreiran potrebno je odraditi postavljanje slušalaca. Prvi dio opisanog dijaloga prikazan je na sl. 4.14.

```
class AddCarDialogFragment : DialogFragment(), AddCarDialogContract.View {

    private var carAddedListener: CarAddedListener? = null
    private val presenter: AddCarDialogContract.Presenter by lazy {
        AddCarDialogPresenter(CarRoomRepository())
    }

    interface CarAddedListener {
        fun onCarAdded(car: Car)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setStyle(STYLE_NO_TITLE, R.style.FragmentDialogTheme)
    }

    override fun onResume() {
        super.onResume()
        presenter.setView(this)
    }

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        return inflater.inflate(R.layout.fragment_dialog_new_car, container)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        initListeners()
    }

    override fun onStart() {
        super.onStart()
        dialog?.window?.setLayout(WindowManager.LayoutParams.MATCH_PARENT, WindowManager.LayoutParams.WRAP_CONTENT)
    }

    private fun initListeners() {
        saveCarRegistration.setOnClickListener{ saveCar()}
    }
```

Sl. 4.14 Prvi dio Dijaloga za dodavanje novog vozila

Kada je postavljen slušalac na gumb za spremanje vozila, potrebno mu je odrediti što radi, a to je ostvarno u metodi *saveCar()*. Prvo se provjerava postoji li uneseni tekst koji se može spremiti, ako ne postoji ispiši poruku kako je potrebno unijeti odgovarajuće parametre kako bi došlo do spremanja, ako je postoji unos uzmi ime vozila i spremi ga u varijablu ime i ako postoji registracija spremi je u varijablu registracija. Nakon toga kreiraj objekt auto u čiju klasu kostruktora prosljedi varijable imena automobila i registracije, kada su postavljene vrijendosti objekta pomoću prezentera predaj metodi a*ddCar()* ovaj objekt. Zatim se poziva metoda *clearUi()* koja čisti dijalog za sljedeću registraciju. Opis drugog dijela dijaloga prikazano je na sl. 4.15.

```
    private fun saveCar() {
        if (isEmpty(newCarTitleInput.text) || isEmpty(newCarRegistrationInput.text)) {
            Toast.makeText(context,"Have to fill both fields.", Toast.LENGTH_SHORT).show()
            return
        }

        val name :String = newCarTitleInput.text.toString()
        val registration :String = newCarRegistrationInput.text.toString()
        val car = Car(name = name, registration = registration)
        presenter.addCar(car)
        onCarAdded(car)
        clearUi()

    }

    private fun clearUi() {
        newCarTitleInput.text.clear()
        newCarRegistrationInput.text.clear()
    }

    fun setCarAddedListener(listener: CarAddedListener){
        carAddedListener = listener
    }

    override fun onCarAdded(car: Car) {
        carAddedListener?.onCarAdded(car)
        dismiss()
    }

    override fun onCarAddFailed() {
        Toast.makeText(context, getString(R.string.FailedToAddCar), Toast.LENGTH_SHORT).show()

    }

    companion object {
        fun newInstance(): AddCarDialogFragment {
            return AddCarDialogFragment()
        }
    }
}
```

Sl. 4.15 Drugi dio dijaloga za dodavanje novog vozila

4.3.4 Programsko rješenje baze podataka

Korišten je Room kao abstraktni sloj SQLite baze podataka koja je opisana u predhodnom poglavlju. Nakon prikazanog modela baze u poglavlju 3.2 sada slijedi implementacija tri glavna dijela Room-a. Entitet, Kotlin podatkovna klasa koja predstavlja tablicu unutar baze podataka. Klasa treba biti označena s anotacijom entitet (*engl. Entity)* kako bi Room znao da se radi o entitetu, svaki entitet mora sadržavati barem jedan primarni ključ (*engl. Primary Key)*. Room automatski stvara stupac u tablici za svako polje, ali to se može izbjeći dodavanjem anotacije ignoriraj (*engl. ignore*) uz određeno polje. Svojstvo Kotlina za razliku od Jave je da postoji podatkovna klasa(*engl. data class*) koja nema potrebu za kreiranje konstruktora, *gettera, settera* već su oni automatski generirani od strane kompajlera. Prema[12], kompajler automatski dostavlja metode s parametrima koji su definirani u konstruktoru, a to su *equals()/ hasCode()* par, *toString()* izgleda „User(name=John, age=42)“, *copy()*. U ovoj podatkovnoj klasi definirane su tri varijable id prirmarni ključ koji će biti sam generiran, registracija i ime. Na sl. 4.16 prikazana je Kotlin podatkovna klasa koja je prethodno opisana.

```
@Entity
data class Car(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,

    val registration: String,
    val name: String
)
```

Sl. 4.16 Kotlin podatkovna klasa, Entitet

Podatkovni pristupni objekt (*engl. data access object*, Dao*)* je sučelje (*engl. Interface)*, koje sadrži metode za pristup bazi podataka. Kako bi Room znao da se radi o pristupnim objektima mora se dodati anotacija Dao. Dao sadrži metode koje daju mogućnost abstraktnog pristupa bazi podataka. Dao također može biti i abstraktna klasa koja u svom konstruktoru prima samo *RoomDatabase* kao svoj jedini parametar [13]. Prvi upit iz baze je dohvaćanje iz baze, listu svih automobila, definirano je anotacijom upit (*engl. Query)* u kojoj je zatražena vrijednost odaberi sve i tablice atuomobil. Slijedi metoda *getAllCars()* koja vraća listu automobila. Zatim uz pomoć anotacije *Insert,* metoda *insertCar()* prima kao parametar objekt automobila koji se umeće u bazu. Zatim metoda *get()* uz pomoć upita odabire sve automobile kojima je identifikacija jednak identifikaciji koju je metoda primila, povratni

tip je automobil, jer je zatražen određeni automobil. Metoda za brisanje automobila *deleteCar()* briše predani automobil uz pomoć *Delete* anotacije. Brisanje može biti ostvareno i *Query* anotacijom kao kod metode *deleteAllCars()* gdje se navodi da se obrišu svi automobili iz tablice. Na sl. 4.17 prikazano je Dao sučelje.

```
@Dao
interface CarDao {
    @Query( value: "SELECT * FROM Car")
    fun getAllCars(): MutableList<Car>

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    fun insertCar(car: Car)

    @Query( value: "SELECT * FROM Car WHERE id= :id")
    fun get(id: Int): Car

    @Delete
    fun deleteCar(car: Car)

    @Query( value: "DELETE from Car")
    fun deleteAllCars()
}
```

Sl. 4.17 Dao sučelje

Dao poslužitelj (*engl. Provider)* je abstraktna klasa koja nasljeđuje *RoomDatabase*. U ovoj klasi definiramo entitete i verziju baze. Kod kreiranja nove instance ove klase treba paziti da se koristi *singleton* oblikovni obrazac iz razloga sto je objekt *RoomDatabase* klase prilično „skup“, a rijetko postoji potreba za više instanci u isto vrijeme[13]. Nešto što se u Javi nazivalo *static* u Kotlinu to obavlja *companion object*, a to znači da se ne mora kreirati instanca objekta kako bi se pristupilo nekoj metodi iz određene klase već samo uz pomoć imena klase imamo njoj pristup. Što to zapravo znači, sada metoda *getInstance()* ne priprada objektu koji je kreiran iz klase *DaoProvider* već samoj klasi. Ova metoda se kreira u memoriji samo onda kada je klasa učitana i ona tamo postoji bez obzira na kreirane objekte. Unutar metode *getInstance()* provjeravamo postoji li već instanca ove klase, ako ne postoji odnosno ako je *null* kreiraj novu instancu pomoću *databaseBuilder()* metode, u suprotnom vrati instancu koja već postoji. Na ovom primjeru se jasno vidi i upotreba nulabilnosti koju donsi Kotlin. Na sl. 4.18 prikazan je Dao poslužitelj koji je prethodno opisan.

```
@Database(entities = [Car::class], version = 1)
abstract class DaoProvider: RoomDatabase() {

    abstract fun carDao(): CarDao

    companion object {

        private var instance: DaoProvider? = null

        fun getInstance(context: Context):DaoProvider{

            if (instance == null){
                instance = Room.databaseBuilder(
                    context.applicationContext,
                    DaoProvider::class.java,
                     name: "CarDb"
                ).allowMainThreadQueries().build()
            }
            return instance as DaoProvider
        }

    }
}
```

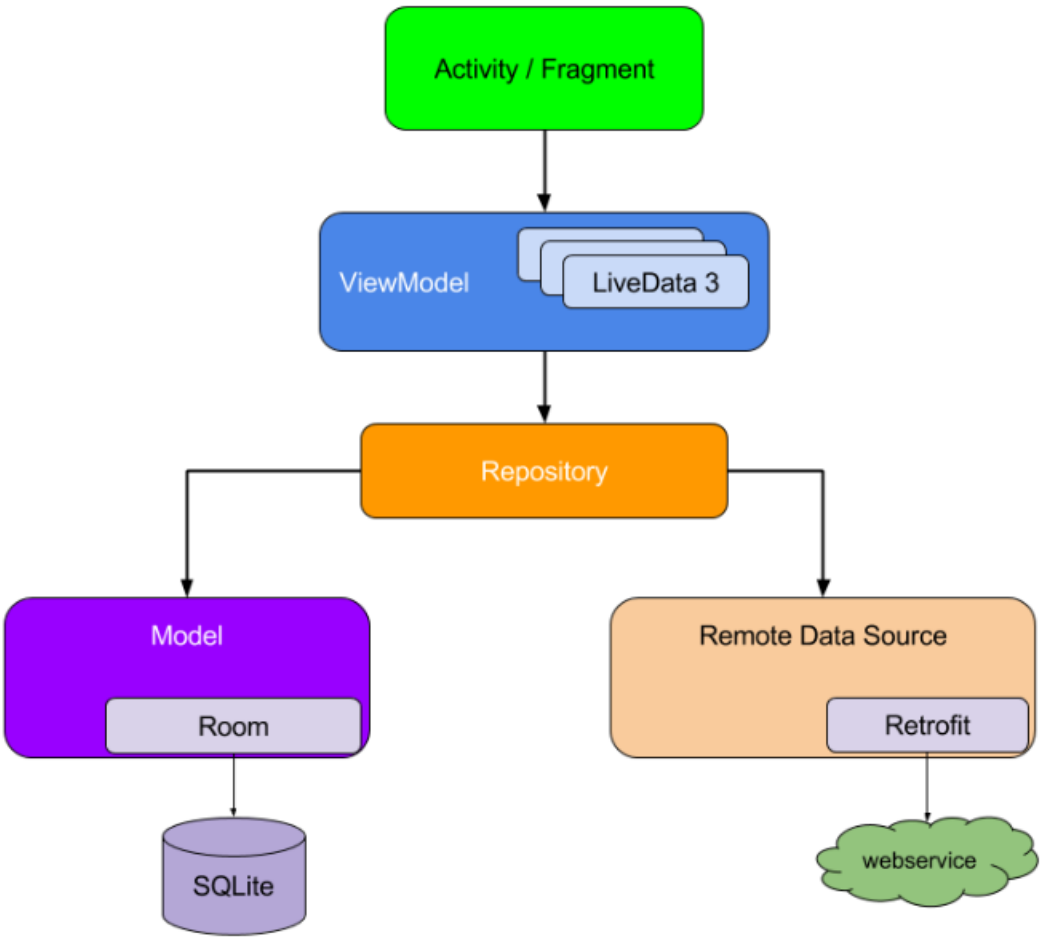
Sl. 4.18 Dao poslužitelj

4.4 Arhitektura mobilne aplikacije

Svaki oblikovni obrazac (*engl. design pattern*) predstavlja rješenje za određeni problem koji se iznova ponavlja svuda oko nas.[14] Oblikovni obrazac opisuje elemente koji pospješuju aplikaciju, njihove odnose, odgovornosti i međusobnu komunikaciju. Rješenje određenog problema u obliku oblikovnog obrasca ne opisuje točnu implementaciju, već predstavlja predložak koji generalno pristupa rješavanju određenih problema. Abstraktno opisuje raspored tih elemenata i podjelu odgovornosti između slojeva. Na prvu povećava broj linija koda što daje dojam kako se sam program komplicira, ali nije tako, uvođenje određene arhitekture dobiva se na apstraktnosti što olakšava testiranje aplikacije, kao i određene izmjene ili nadgradnju postojećeg projekta.

Prema[15], u većini slučajeva desktop aplikacije imaju jednu ulaznu točku i rade kao jedan jednostavan proces s druge strane Android aplikacije imaju puno složeniju strukturu. Tipična Android

aplikacija sadrži brojne komponente uključujući aktivitije, fragmente, servise, itd. Rad svake aplikacije kada je ona u fokusu djelovanja može doći do prekida, na primjer zaprimljenim pozivom. Korisnik se nakon poziva očekuje vratiti aplikaciji koju je koristio. To „app-hopping“ ponašanje je sasvim normalno za mobilne uređaje, zbog takvih uobičajenih situacija ne bi se trebalo držati podatke u komponentama aplikacije, niti komponente trebaju međusobno ovisiti jedna o drugoj. Na sl. 4.19 je prikazan dijagram koji opisuje kako bi moduli trebali komunicirati jedan s drugim nakon što je aplikacija dizajnirana.



Sl. 4.19 Dijagram komunikacije između modula[15]

U aplikaciji za automatsku naplatu parkiranja korišten je arhitekturalni obrazac MVP (*engl. model view presenter).* Kojim ispunjavamo uvjete dijagrama sa slike 4.19. Uloga MVP-a je odvojiti samu logiku od pogleda (*engl. view)* [16]. U ovom slučaju logiku koja radi s dohvaćanjem podataka iz baze i pohranom podataka u bazu. Prednost toga što je logika izvan aktivitija ili fragmenta je ta što se pogled može zasebno testirati, mijenjati i proširivati.

Sl. 4.20 Model-Pogled-Prezenter

Pomoću arhitekturalnog obrazsca ostvareni su neki od SOLID principa. Prema[17], S.O.L.I.D. je pet principa, svako slovo opisuje jednu karakteristiku koju se treba poštivati tijekom pisanja programa. S – *The Single Responsibility Principle (SRP),* jedna klasa treba imati samo jednu odgovornost. O – *The Open-Closed Principle (OCP),* klase, funkcije i moduli moraju biti otovreni za „nasljeđivanje“, ali zatvoreni za mijenjanje. L – *The Liskov Substitution Principle (LSP),* podklasa ne smije „strgati“ roditeljsku klasu nakon prepisivanja naslijeđene metode. I – *The Interface Segregation Principle (ISP),* ovaj princip objašnjava da ako je sučelje prekrcato metodama, radije treba to sučelje podjeliti na više manjih sučelja nego da programer mora nakon što implementira sučelje implementrirati i metode koje mu nisu potrebne. D – *The Dependency Inversion Principle(DIP),* moduli visoke razine ne bi trebali ovisiti o modulima niže razine, već bi obje razine trebale ovisiti o apstrakcijama.

4.5 Slojevi aplikacije

Aplikacija je podijeljena na tri sloja model, pogled i prezentacijski sloj. Svaki sloj ima jasno definiranu zadaću kako bi postigli točno raspoređene odgovornosti po slojevima. Prvo je potrebno kreirati ugovor (*engl. contract*) koji opisuje komunikaciju između pogleda i prezentera. Dobra je praksa pisati ugovor kao sučelje koje u sebi sadrži dva sučelja, jedno za pogled, a drugo za prezenter. Na sl. 4.21 može se vidjeti primjer jednog ugovora iz aplikacije. Iz samih naziva se može zaključiti kakvu odgovornost imaju određena sučelja i njihove metode. Sučelje pogleda samo zaprima obrađene podatke dok sučelje prezentera ima nekakve logičke zahtjeve koje dalje prenosi prema modelu.

```
interface RegisteredCarsContract {

    interface View{

        fun onCarListReceived(cars: MutableList<Car>)

        fun onGetCarsFailed()

    }

    interface Presenter{

        fun setView(view: RegisteredCarsContract.View)

        fun onGetAllCars(): MutableList<Car>

        fun deleteCar(car: Car)

        fun deleteAllCars()


    }
}
```

Sl. 4.21 Sučelje ugovora (*engl. contract interface*)

Zatim se kreira prezenter koji prihvaća podatke iz modela i smislene ih prikazuje pogledu uz pomoć varijable *repostitory*. Metode koje su definirane u ugovoru odnosno sučelju prezentera implementirane su od strane *RegisteredCarsPresenter*. Na sl. 4.22 prikaz je opisani prezenter.

```
class RegisteredCarsPresenter(private val repository: CarRoomRepository): RegisteredCarsContract.Presenter {

    private lateinit var view: RegisteredCarsContract.View

    override fun setView(view: RegisteredCarsContract.View) {
        this.view = view
    }

    override fun onGetAllCars(): MutableList<Car> {
        return repository.getAllCars()
    }

    override fun deleteCar(car: Car) {
        repository.deleteCar(car)
    }

    override fun deleteAllCars() {
        repository.deleteAllCars()
    }
}
```

Sl. 4.22 Prezenter

Pogled je implementiran u Fragmentu koji sadrži slabu referencu na prezenter. Također odgovoran je za kreiranje reference. Pogled služi za prikaz podataka koje je iskomunicirao preko reference prezentera. Dakle pogled ne zna što se događa s tim podatcima, niti logiku kako je do njih došlo, ono samo dobije podatke na prikaz. Na sl. 4.23 prikazan je pogled, kreirani objekt prezentera i prikazano je kako se postavljaju podatci dohvaćeni tim objektom.

```
class RegisteredCarsFragment : BaseFragment(), AddCarDialogFragment.CarAddedListener, RegisteredCarsContract.View,
    MenuMethodes.RegistrationMethods {

    private val adapter : RegistrationAdapter by lazy { RegistrationAdapter({ onItemSelected(it) }, { deleteItem(it) }) }
    private val presenter: RegisteredCarsContract.Presenter by lazy {
        RegisteredCarsPresenter(CarRoomRepository())
    }

    override fun getLayoutResourceId() = R.layout.fragment_registered_cars

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        initUi()
        initListeners()
    }

    override fun onPause() {
        super.onPause()
        refreshCars()
    }

    override fun onResume() {
        super.onResume()
        this.setHasOptionsMenu(true)
        (activity as MainActivity).setListeners(this)
        refreshCars()
    }

    private fun initUi() {
        progress.visibility
        noData.isCursorVisible
        carsRecyclerView.layoutManager = LinearLayoutManager(context)
        carsRecyclerView.adapter = adapter
    }
```

Sl. 4.23 Pogled (*engl. View)*

Zatim sloj modela, njegova je uloga dostavljati podatke prezenteru preko *repository*-a prilagođene pogledu za koji je namijenjen. Na primjer mrežne podatke ili podatke iz baze podataka što je slučaj u ovoj aplikaciji. Ono je korisniku najudaljeniji i predstavalja entitet koji se koristi bazom podataka.

CarRepository je sučelje s metodama za dohvaćanje automobila, dodavanje i brisanje koje je prikazano na sl. 4.24.

```
interface CarRepository {
    fun getAllCars(): MutableList<Car>
    fun addCar(car: Car)
    fun get(id: Int): Car
    fun deleteCar(car: Car)
    fun deleteAllCars()
}
```

Sl.4.24 *CarRepository* sučelje

CarRoomRepository implementira sučelje CarRepository i implementira sve navedene metode. Kreira se podatkovni pristupni objekt (*engl. Data Access Object*) koji je prethodno opisan u ranijim poglavljima, uz pomoć tog objekta pristupa se metodama koje rade upite na bazu, ovisno o upitu predaju se određeni parametri. Na primjer ako želimo obrisati određeni automobil pomoću metode *deleteCar()* njoj se mora predati automobil i sada pomoću *carDao* objekta pristupa se metodi *deleteCar()* koja se nalazi u *CarDao* sučelju i očekuje automobil kao parametar. Opisani repozitorij nalazi se na sl. 4.25.

```
class CarRoomRepository: CarRepository {

    private var db :DaoProvider = DaoProvider.getInstance(AppParking.getAppContext())
    private var carDao :CarDao = db.carDao()

    override fun getAllCars(): MutableList<Car> {
        return carDao.getAllCars()
    }

    override fun addCar(car: Car) {
        carDao.insertCar(car)
    }

    override fun get(id: Int): Car {
        return carDao.get(id)
    }

    override fun deleteCar(car: Car) {
        carDao.deleteCar(car)
    }

    override fun deleteAllCars() {
        carDao.deleteAllCars()
    }

}
```

Sl.4.25 *CarRoomRepository*

5. NAČIN KORIŠTENJA I ISPITIVANJE RADA APLIKACIJE

5.1 Opis kako se koristi aplikacija

Mobilna aplikacija za automatsku naplatu parkiranja vrlo je jednostavna i intuitivna za korištenje. Korisnik ima početni zaslon u kojemu se nalazi leteći akcijski gumb (*engl. floating action button,* FAB*)* u donjem desnom kutu zaslona. Klikom na taj gumb korisniku se otvara dijalog za registraciju u kojemu su ponuđena dva polja za unos teksta, a to su naziv vozila i registracijske oznake. Klikom na gumb (*engl. button)* „save“ sprema se registrirano vozilo i zatvara se dijalog za registraciju. Sada je korisnik ponovno na početnom zaslonu u kojemu je prikazano registrirano vozilo. Korisnik sada ima četiri mogućnosti kao sljedeći korak. Može ponoviti postupak dodavanja novog vozila. Ima mogućnost brisanja registriranog vozila na dva načina jedan je pojedinačno brisanje ili brisanje svih vozila koje je registrirao. Pojedinačno brisanje se izvršava tako što korisnik prstom povuče s desna na lijevo preko dijela zaslona gdje se nalazi određeno registrirano vozilo. Zatim mu se otvori dijalog koji ga upita je li siguran da želi obrisati određeno vozilo, korisniku su ponuđena dva gumba odustani kako bi otkazao akciju ili obriši kako bi izvršio brisanje određenog vozila. Opcija brisanja svih vozila se nalazi u desnom gornjem kutu, izgled gumba su tri vertikalno raspoređene točke koje predstavljaju izbornik (*engl. menu)*. Klikom na izbornik prikazat će se gumb „delete all“ pritiskom na isti obrisat će se sva registrirana vozila. Četvrta mogućnost je klik na samo registrirano vozilo, u tom slučaju otvorit će se novi zaslon u kojemu se nalaze google karte na kojima su označene parkirne zone i korisnikova trenutna geolokacija. Odmah će se automatski izvršiti provjera nalazi li se korisnikova geolokacija u određenoj zoni i otvorit će se dijalog za naplatu (*engl. payment)*. Dijalog za naplatu ima dva polja koja su ispunjena registracijskom oznakom vozila za koje korisnki želi izvršiti kupnju karte i polja s brojem zone u kojoj se korisnik trenutno nalazi, ako se korisnik ne nalazi niti u jednoj od označenih zona to polje će ostati prazno i korisnik mora unijeti broj zone ručno isto tako korisnik ima mogućnost izmjene sadržaja u ova dva polja. Zatim korisnik ima mogućnost klika na gumb „prepare sms“ , nakon klika otvorit će se aplikacija za slanje SMS poruka. Registracijska oznaka bit će sadržaj poruke, a broj zone bit će adresa na koju će biti poslana poruka. Nakon što je poruka poslana dolazi povratna poruka o plaćenoj prakirnoj karti.

5.2 Scenarij i ispitivanje

Korisnik bi trebao registrirati vozilo, zatim po potrebi kada parkira vozilo u nekoj od dvije označene zone ili aplikaciji nepoznatoj zoni odabrati registrirano vozilo. Aplikacija će automatski prepoznati zonu u kojoj se nalazi i prema geolokaciji ponuditi broj prakirne zone. U slučaju da se korisnik nalazi u aplikaciji nepoznatoj zoni ponudit će mu se prazno mjesto za upis broja zone u kojoj se nalazi, odnosno na koji broj će biti poslana njegova registarska oznaka. Nakon tog koraka, adresa na koju se šalje sadržaj poruke je broj zone, a sadržaj poruke je registracijska oznaka prikazana u SMS aplikaciji.

Mobilna aplikacija je ispitana na više lokacija i u obje parkirne zone. Aplikacija se ponaša kao što je i predviđeno, automatski prepoznaje zonu u kojoj se nalazi. Kada nije niti u jednoj zoni nudi mogućnost ručnog upisa broja zone. Također registrirano je više vozila u aplikaciju i za svako vozilo naplata parkiranja je provedena.

5.3 Analiza

Mobilna aplikacija za automatsku naplatu parkiranja sa svojom MVP arhitekturom otvara mogućnost vrlo lakog proširenja. Dodavanje parkirnih zona, skaliranje na ostale gradove. Dakako aplikacija ispunjava svoje osnovne funkcionalnosti, a to su da korisnik može lako i brzo registrirati vozilo te mu zatim aplikacija automatski pronađe u kojoj se zoni nalazi i sam izvrši naplatu. Korištena je SQL baza podataka kojom je komunicirano putem apstraktnog sloja preko Room-a. Zbog malog broja podataka koje je potrebno spremiti, podaci su spremljeni lokalno.

6. ZAKLJUČAK

U ovom završnom radu opisana je aplikacija za automatsku naplatu parkiranja. Pokazano je određeno znanje kreiranja Android aplikacije u Kotlin programskom jeziku, postavljanje arhitekture MVP arhitekturalnim obrazscem, te spremanje određenih podataka o vozilima u bazu podataka preko Room apstraktnog sloja. Korištenje geolokacije preko google karte uz pomoć *Google Maps API*. Najzahtjevniji segment ove aplikacije bilo je postavljanje arhitekture i stvaranje logike za prepoznavanje zona. Nešto što bi obogatilo ovaj rad bilo bi testiranje aplikacije samim pisanjem testova. Način na koji bi se aplikacija mogla poboljšati jest dodavanje svih postojećih zona u gradu Osijeku isto tako skaliranje na ostale gradove. Mogućnost dodavanja registracije korisnika u samu aplikaciju, to bi zahtijevalo postavljanje mrežne komunikacije u samu aplikaciju. Proširenje aplikacije može ići u smjeru dodavanja raznih funkcionalnosti o prometu, koje bi ubrzale i olakšale pristup informacijama samim vozačima.

7. LITERATURA

[1] Elektromodul Osijek, https://elektromodul.hr/parking-osijek, pristupljeno 9.2019.

[2] B. Hofmann-Wellenhof, H. Lichtenegger, E.Wasle, GNSS – Global Navigation Satellite Systems: GPS, GLONASS, Galileo and more, str:277,341, Beč, 2008. 8

[3] Room dokumentacija. Dostupna na: https://developer.android.com/training/data-storage/room#kotlin , pristupljeno 9.2019.

[4] Geolocation, https://www.techopedia.com/definition/1935/geolocation, Techopedia, pristupljeno 9.2019.

[5] Android Location Providers (gps, network, passive), pristupljeno 9.2019.

[6] M.L. Murphy, The Busy Coder's Guide to Android Development, CommonsWare, United States of America, 2014.

[7] Android skripta, https://web.math.pmf.unizg.hr/~karaga/android/android_skripta.pdf, pristupljeno 9.2019. 9

[8] Uvod u Kotlin, https://startit.rs/uvod-u-kotlin-koje-su-prednosti-zvanicnog-jezika-za-razvoj-android-aplikacija/ , Marko Arsić, pristupljeno 9.2019. 17

[9] XML in Android, https://abhiandroid.com/ui/xml, Abhi Android, pristupljeno 9.2019. 16

[10] Fragment dokumentacija, https://developer.android.com/guide/components/fragments , pristupljeno 9.2019. 13

[11] Android Manifest dokumentacija, https://developer.android.com/guide/topics/manifest/manifest-intro, pristupljeno 9.2019. 14

[12] Data Classes, https://kotlinlang.org/docs/reference/data-classes.html , pristupljeno 9.2019. 15

[13] Android Architecture Components Room and Kotlin, https://medium.com/mindorks/android-architecture-components-room-and-kotlin-f7b725c8d1d, Ritesh Singh, pristupljeno 9.2019.

[14] Design Patterns, Elements of Reusable Object-Oriented Software, Addison Wesley, United States of America 1994.

[15] Guide to app architecture, https://developer.android.com/jetpack/docs/guide, pristupljeno 9.2019.

[16] Model-View-Presenter: Android Guideliness, https://medium.com/@cervonefrancesco/model-view-presenter-android-guidelines-94970b430ddf, Francesco Cervone, pristupljeno 9.2019.

[17] Exploring S.O.L.I.D. Principle in Android, https://proandroiddev.com/exploring-s-o-l-i-d-principle-in-android-a90947f57cf0, Anitaa Murthy, pristupljeno 9.2019.

SAŽETAK

Cilj ovog završnog rada bio je kreirati mobilnu aplikaciju za automatsku naplatu parkiranja. Ovo rješenje implementirano je na Android platformi, program je pisan u Kotlinu, sučelje je definirano u XML-u. Aplikacija traži spremanje malog broja podataka lokalno, to je ostvareno pomoću Room baze podataka. U aplikaciji je utvrđena geolokacija korisnika i uspoređena s geolokacijom zone. Nakon određivanja zone izvršena je naplata pomoću već postojeće SMS aplikacije. Korisniku je dostupno lako registriranje vozila, pronalazak vozila u određenoj zoni te naplata parkiranja.

Ključne riječi: baza podataka, korisničko sučelje, mobilna aplikacija, naplata parkiranja

ABSTRACT

Mobile apllication for automated parking payment

The goal of this final paper was to create mobile application for automated parking payment. This solution was implemented on Android platform, program was written in Kotlin, interface was defined in XML. Application need to store small amount of data locally, it was resolved with Room database. The application define geolocation of user and compare it with geolocation of zone. After the zone was established parking payment can be executed by SMS application. User has easy solution for registration of his vehicle, finding his vehicle in certain zone and parking payment.

Keywords: database, user interface, mobile application, parking payment

9. ŽIVOTOPIS

Filip Širac rođen je 21.12.1996. u Osijeku. Živi u Pakracu gdje pohađa osnovnu školu Braće Radića. Nakon završetka osnovne škole upisuje Tehničku školu u Daruvaru smjer Računalstvo. Sudjeluje na projektu „Balon Stellar - stratosfera 30 km“ u sklopu Google Lunar Xprize. Također sudjeluje na smotri radova na Fakultetu Elektrotehnike i Računarstva u Zagrebu sa završnim radom „Samobalansirani robot na dva kotača“. Nakon završetka srednje škole ostvarujue direktno pravo upisa na fakultet temeljem uspjeha ostvarenog tijekom srednjoškolskog obrazovanja. Trenutno studira Računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek.

_____

(Vlastoručni potpis)

PRILOZI (na CD-u)

Prilog 1. Završni rad u docx formatu

Prilog 2. Završni rad u pdf formatu

Prilog 3. Projekt mobilne aplikacije