Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Hacin

Prototip aplikacije za souporabo avtomobilov

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Trendi kažejo, da bo souporaba vozil vse pomembnejši pristop pri prevažanju ljudi, saj je praviloma bolj ekonomična od lastništva vozil. Proučite funkcionalnosti, ki so potrebne za mobilno aplikacijo, ki bi članom omogočala souporabo vozil. Ne pozabite na pravno-formalni del, saj avtomobila ne moremo prepustiti vsakomur, ki ima nameščeno mobilno aplikacijo. Na podlagi tega razvijte prototip mobilne aplikacije za souporabo vozil.

Zahvaljujem se mami, očetu in preostali družini, ki so mi nudili podporo in motivacijo skozi celoten študij.

Zahvala pa pripada tudi vsem prijateljem in sodelavcem, ki so poskrbeli, da mi bodo študentska leta za vedno ostala v lepem spominu.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
REST	Representational State Transfer	arhitektura za izdelavo spletnih storitev
API	Application Programming Interface	vmesnik za programiranje
JSON	JavaScript Object Notation	javascript objektni zapis
HTTP	Hypertext Transfer Protocol	glavna metoda za prenos informacij na spletu
GPS	Global Positioning System	globalni sistem pozicioniranja
JVM	Java Virtual Machine	virtualni stroj, ki poganja javanske aplikacije
XML	Extensible Markup Language	razširljivi označevalni jezik
URL	Uniform Resource Locator	enolični krajevnik vira
LTE	Long Term Evolution	standard na področju mobilnih komunikacij
UI	User Interface	uporabniški vmesnik

Povzetek

Naslov: Prototip aplikacije za souporabo avtomobilov

Avtor: Matej Hacin

Živimo v času, ko so avtomobili čedalje bolj dostopni, zaradi česar se večja mesta srečujejo s težavo prevelikega števila vozil. Eden izmed možnih pristopov do reševanja te težave je souporaba avtomobilov. Cilj te diplomske naloge je razviti prototip mobilne aplikacije, ki bo uporabnikom omogočala prav to. Z aplikacijo bo imel vsak uporabnik enostaven pregled nad prostimi avtomobili in bo katerega koli izmed teh lahko brez problemov rezerviral in upravljal. Pri tem bomo uporabili tehnologije za razvoj mobilnih aplikacij in komunikacijo med različnimi sistemi, ki so povezani preko spletnega omrežja. Našo rešitev bomo analizirali in določili možnosti za izboljšavo in implementacijo dodatnih funkcionalnosti.

Ključne besede: mobilna aplikacija, souporaba avtomobilov, promet, digitalna rešitev.

Abstract

Title: Prototype application for car sharing

Author: Matej Hacin

We live in a time when cars are becoming increasingly affordable, making the major cities face the problem of too many vehicles. One of the possible solutions to this problem is car sharing. The purpose of this thesis is to develop a prototype of a mobile application that will allow users to do just that. With the application, each user will have a simple overview of available cars and any of these can be easily reserved and managed. We will use technologies for mobile application development and network communication. Lastly, we will analyze our solution and determine the possibilities for improving and implementing additional functionality.

Keywords: mobile application, car sharing, traffic, digital solution.

Poglavje 1

Uvod

V Sloveniji imamo povprečno en avtomobil na dva prebivalca, zaradi česar imajo večja mesta ogromno parkirišč, večjo onesnaženost s CO_2 in zelo pogoste prometne zastoje. Poleg tega je vsak avtomobil tudi velik finančen napor za lastnika, saj je potrebno plačevati stroške, kot so menjava gum, zavarovanje, cestnina, popravila in tehnični pregledi, četudi avtomobil večino časa miruje. Mobilna aplikacija za souporabo avtomobilov omogoča uporabnikom, da imajo vedno na voljo avto, ki ni v njihovi lasti, plačajo ga samo takrat, ko ga aktivno uporabljajo [4].

1.1 Poslovna domena

Poslovna domena je izposoja avtomobilov, torej so konkurenčne aplikacije vse tiste, ki na kakršen koli način ponujajo izposojo avtomobilov. Težava klasičnih podjetij, ki tovrstno storitev ponujajo, je, da lahko stranka avtomobil rezervira samo nekaj dni v naprej in za točno določen čas, kar je po navadi več kot en dan. Poleg tega je tudi uporabniška izkušnja slabša, saj mora stranka ob vsaki rezervaciji avtomobila plačati vnaprej in osebno podpisati pogodbo, ko na podjetju pokaže svoje osebne dokumente. Sodobna tehnologija nam omogoča, da si vse podatke o stranki že vnaprej shranimo in dovolimo, da avtomobil plača glede na porabljene minute in kilometre ob

koncu najema. Mobilna aplikacija uporabniku omogoča, da avtomobil najame, upravlja s ključavnico in ob končani uporabi zaključi najem. Celoten postopek izposoje avtomobila se tako izvede brez kakršne koli komunikacije s podjetjem. S tem bomo v Sloveniji ustvarili novo možnost najema avtomobilov, kakršne ne ponuja še nobeno drugo podjetje, zato konkurence efektivno še ni.

1.2 Struktura diplomske naloge

Diplomska naloga se pričenja z definicijo problema in rešitve, nato so opisane uporabljene tehnologije in programska oprema, ki je ključna za razvoj. V nadaljevanju se načrtuje razvoj prototipa, tako da so našteti možni primeri uporabe, zastavljena želena programska arhitektura in opisan potek komunikacije z avtomobili. Naprej so opisani pogledi v aplikaciji, kot so navigacija, izbira lokacije in rezervacija avtomobila. Pri tem so opisani podatki, ki jih aplikacija uporabniku prikazuje. Na koncu so naštete še možnosti za nadaljnji razvoj aplikacije, kamor spadata tudi optimizacija delovanja in dodajanje novih funkcionalnosti.

Poglavje 2

Pregled problema in rešitve

2.1 Problem

V Sloveniji je bilo po podatkih iz leta 2015 registriranih več kot milijon avtomobilov, po čemer lahko sklepamo, da je to več kot en avtomobil na eno zaposleno osebo oziroma en avtomobil na dva prebivalca. Povprečen strošek avtomobila za lastnika skupaj s kupno ceno, zavarovanjem, cestninami in popravili je približno 500 EUR na mesec. Po evropski statistiki je avtomobil 90% časa neuporabljen na mestu, ljudje pa zaradi tega gradimo dodatno infrastrukturo, širše ceste in večja parkirišča. Poleg tega velika večina teh avtomobilov za pogon uporablja gorivo, kot sta bencin in nafta, kar pomeni, da močno prispevajo k onesnaževanju zraka s CO_2. Posamezniki torej v povprečju porabimo veliko denarja za avto, ki potem večino časa samo zavzema prostor, ki bi bil drugače lahko bolje porabljen, na primer s prostorom za pešce ali mestnim parkom [4].

2.2 Rešitev

Glede na to, da je povprečen avtomobil v uporabi približno 10% časa, lahko izkoristek avtomobila znatno povečamo tako, da si ga uporabniki delijo. Kadar avtomobila ne uporablja uporabnik, ga uporablja nekdo drug. Storitev

souporabe prispeva k zniževanju števila lastniških avtomobilov, in sicer se na en avtomobil v souporabi število lastniških zniža v povprečju za 10 [3].

Prvi del naše rešitve je flota električnih avtomobilov, ki so razporejeni na določenih mestih po Sloveniji v bližini stanovanjskih blokov, trgovin in raznih poslovnih stavb. Avtomobili so vedno na voljo za uporabnike naše storitve, ki lahko avtomobil rezervirajo do 15 minut vnaprej brez kakršnega koli dodatnega stroška. Šele ko uporabnik avtomobil prevzame, se začnejo računati minute in kilometrina, kar uporabnik plača ob koncu najema. Najem je lahko dolg do 24 ur, saj smatramo, da uporabnik avtomobila za daljši čas ne potrebuje v aktivnem najemu. Za dodatne stroške, kot je gorivo, ni potrebno skrbeti, saj so avtomobili električni, polnjenje na naših lokacijah je pa povsem brezplačno oziroma vključeno v ceno uporabe storitve.

Drugi del rešitve je mobilna aplikacija, ki jo bomo bolj podrobno opisali v tej diplomski nalogi. Aplikacija uporabnikom omogoča, da se za storitev registrirajo, plačajo enkratno prijavnino in nato uporabljajo vse avtomobile v naši floti. Registrirani uporabnik vidi vse lokacije, na katerih se nahajajo avtomobili. Ko uporabnik avtomobil rezervira, ima 15 minut časa, da ga prevzame in uradno začne najem. Takrat dobi možnost odklepa avtomobila z aplikacijo, brez ključa, ki se že nahaja v avtomobilu. Ko avtomobila ne potrebuje več, ga vrne na katero koli izmed razpoložljivih prevzemno-vračilnih lokacij in z izbrano plačilno metodo plača najem. Poleg enkratne prijavnine in najema avtomobila uporabnik nima nobenih dodatnih stroškov, kot bi jih imel z lastnim avtomobilom. Tudi parkirnine so zastonj, četudi se lokacije nahajajo tudi v centrih večjih mest, kjer je navadno težje najti zastonj parkirišče.

Poglavje 3

Tehnologije in programska oprema

V tem poglavju bomo na kratko opisali vsako izmed uporabljenih tehnologij in programskih jezikov ter opredelili, kje in za kaj se uporablja. S tem bosta struktura in delovanje projekta veliko bolj razumljiva. Začeli bomo z osnovami, kot je glavno razvijalsko okolje za operacijski sistem Android, nato pa z opisom tehnologij in programske opreme nadaljevali vse do tehnologije za komunikacijo s strežnikom in avtomobilom.

3.1 Android Studio

Android Studio je osnovno razvijalsko okolje za operacijski sistem Android, ki ga je izdelal Google. Z istim namenom se lahko uporablja katero koli razvijalsko okolje, ki podpira Javo ali Kotlin, vendar je v tem primeru to nesmiselno, saj je Android Studio za to najbolje optimiziran in ponuja največ orodij za delo z Androidom.

Da v Android Studiu postavimo projekt, enostavno izberemo primarni jezik, v katerem želimo programirati (Java ali Kotlin) in najmanjšo verzijo Android sistema, za katero bo naša aplikacija podprta. V našem primeru sta to Kotlin ter Android verzija 19 [1].

3.2 Java

Java je primarni programski jezik za Android, ki je podprt že od samega nastanka tega operacijskega sistema. Celoten Android sistem poganja Javanski Virtualni Stroj JVM, zato je Java najbolj optimalen jezik za pisanje tovrstnih aplikacij.

Komplet za razvoj programske opreme Android, ki ga ponuja Google, je v večini spisan v Javi, zato ima programiranje v tem jeziku najmanj upora. Naš projekt bo pisan v Javi in Kotlinu, ki bo opisan v poglavju 3.3 [2].

3.3 Kotlin

Kotlin je programski jezik, ki se prav tako poganja na JVM, zato je povsem kompatibilen z Android sistemom. Jezik je sicer starejši, vendar ga je Google leta 2017 poleg Jave uradno podprl kot razvojni jezik v Android Studiu. Kotlin podpira iste funkcionalnosti in lahko v projektu živi skupaj z Javo, kar pomeni, da lahko dostopa do Javanskih funkcij, spremenljivk in razredov, in obratno [6].

Razlog, da se pri Android razvoju zadnje čase raje odločamo za Kotlin kot za Javo, je, da je ta jezik veliko modernejši. Ponuja nam sintakso, s katero lahko na eleganten način napišemo kodo, ki je lažje berljiva ter vzdrževana kot Java. Posledično se piše tudi bolj optimalna koda, saj nam ponuja uporabne funkcionalnosti, kot so na primer:

neobvezne spremenljivke, ki jih lahko varno uporabimo, ne da bi prej preverili, ali imajo nastavljeno vrednost,

vrstične funkcije, ki med drugim omogočajo, da blok kode enostavno pošljemo kot parameter v drugo funkcijo,

razširitve razredov, kar pomeni, da lahko že obstoječim razredom, ki jih nismo izdelali mi, dodamo novo funkcionalnost.

3.4 XML

XML oziroma eXtensible Markup Language je označevalni jezik, ki se v večini uporablja za shranjevanje in prenašanje podatkov. V Android projektu se uporablja za izdelovanje uporabniških vmesnikov.

Android Studio ponuja orodje, ki se imenuje graditelj vmesnikov in nam omogoča, da vizualno zgradimo celoten uporabniški vmesnik, tako da z miško nastavimo postavitev gumbov, slik in drugih gradnikov. Ker je ta metoda velikokrat zamudna in omejena, se v večini primerov odločimo, da bomo uporabniški vmesnik zgradili kar s pomočjo XML kode. Ta jezik je tako enostaven, da se ga lahko naučimo v nekaj minutah, saj ima zelo malo sintaktičnih pravil in omejitev.

Ponuja nam enostavne ukaze za postavitev grafičnih gradnikov, kot so na primer poravnava, relativna pozicija in velikost gradnika. Za gradnike, kot je pisava, nam poleg teh osnovnih funkcij ponuja tudi ukaze za nastavitev velikosti, barve in stila pisave. Ob pisanju vseh teh ukazov nam Android Studio v živo osvežuje predogled naše trenutne postavitve elementov. Prav zaradi tega je neposredno pisanje XML-ja v večini prednostna izbira [7].

3.5 REST

Representational state transfer, na kratko REST, je arhitekturni stil, ki se uporablja za izdelavo spletnih storitev (API). To je skupek pravil, ki jim v večini sledita strežnik in odjemalec, da se lahko med seboj brezhibno sporazumevata.

Komunikacija poteka preko enega izmed prvih protokolov za spletno komunikacijo, HTTP. Podatki se prenašajo v obliki JSON, ki bo opisan v poglavju 3.6.

Vsak zahtevek, ki ga odjemalec pošlje strežniku, je poslan na končno točko, ki je del vnaprej definiranega osnovnega URL-ja. Na vsak zahtevek nato strežnik odgovori z želenimi podatki ali enostavnim odgovorom uspešno/neuspešno. Poleg vsakega odgovora pošlje tudi HTTP statusno

kodo, iz katere lahko razberemo, kje je bila napaka, če se je le-ta pojavila.

Osnovni ukazi, ki nam jih REST ponuja in jih bomo uporabljali tudi mi, so:

GET, s katerim od strežnika zahtevamo določeni podatek ali seznam podatkov,

POST, ki strežniku pošlje podatek in v podatkovni bazi naredi nov vnos,

PUT, ki strežniku pove, kateri obstoječi podatek želimo posodobiti in kako,

DELETE, ki strežniku pove, kateri podatek želimo izbrisati iz podatkovne baze.

Če zahtevek pošljemo na pravilni URL s podatki v pravilni obliki, nam bo strežnik vrnil odgovor s statusno kodo 200 in želenim odgovorom [8].

3.6 JSON

JavaScript Object Notation ali JSON je format, ki se pogosto uporablja za izmenjavo podatkov med različnimi sistemi, kot sta v našem primeru strežnik in odjemalec. Oba sistema uporabljata drug programski jezik in delujeta na povsem drugačen način, vendar se lahko med seboj sporazumevata s podatkovno obliko, ki jo znata oba serializirati in deserializirati.

JSON ima zelo malo enostavnih sintaktičnih pravil in je enostavno berljiv. Podpira osnovne tipe podatkov, kot so niz, številka, objekt, seznam in boolean (resnica/neresnica). Poleg teh podatkov vsebuje minimalno število drugih znakov, zato je eden izmed najprimernejših formatov za prenos podatkov [8].

3.7 Git

Git je sistem kontrole verzij, primeren za manjše in večje projekte. Poleg tega služi tudi kot baza za shrambo našega projekta, in omogoča, da imajo

do njega dostop tudi drugi, ki niso nujno razvijalci.

Ko se datoteka spremeni za vrstico ali več, kar sami določimo ob razvijanju, se v sistemu Git vidi točna sprememba in datum spremembe. To pomeni, da imamo dostop do celotne zgodovine razvoja projekta od samega začetka. Če se v kodi pojavi napaka ali nas enostavno zanima, kakšna je bila koda nekaj časa nazaj, lahko kadar koli pogledamo zgodovino projekta in vidimo, kje so bile spremembe in kakšne so bile.

Sistem Git je odprtokoden, kar pomeni, da ga lahko poganja kateri koli strežnik. Postavljenega lahko imamo na lastnemu strežniku ali pa za to uporabimo kakšno izmed obstoječih storitev, ko so GitHub ali BitBucket. V našem primeru bomo imeli Git postavljen na svojem strežniku [5].

Poglavje 4

Analiza in načrtovanje

4.1 Diagram primerov uporabe

Pred izdelavo aplikacije je potrebno definirati osnovne primere uporabe, da je potem lažje zasnovati uporabniški vmesnik in arhitekturo aplikacije. V našem primeru je glavni primer uporabe rezervacija avtomobila, poleg tega pa ima aplikacija še druge stranske funkcionalnosti, ki so lahko ravno tako pomembne za posameznega uporabnika.

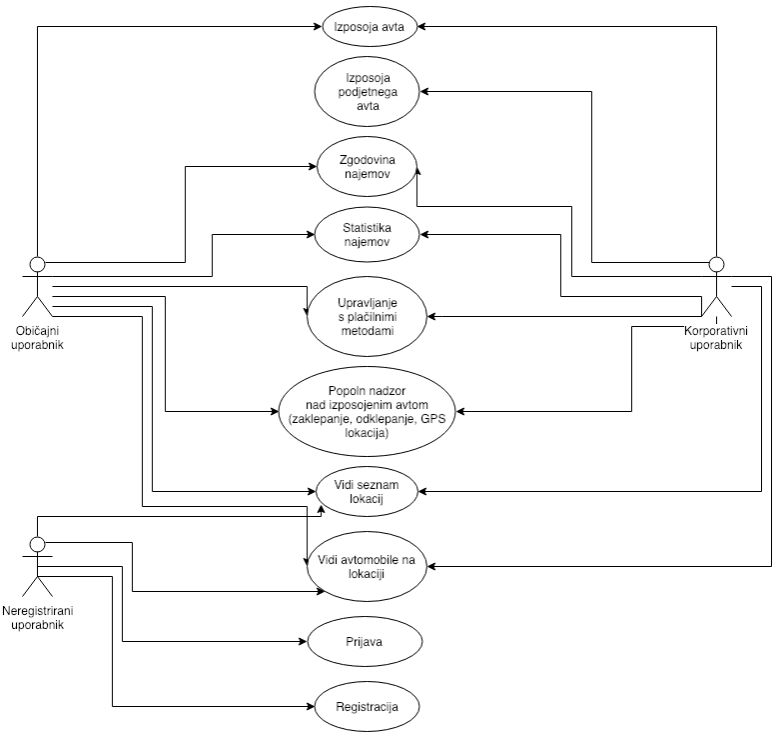
Imeli bomo tri vrste uporabnikov, ki so:

neregistrirani uporabnik, ki je ravnokar namestil aplikacijo, zato ima najbolj omejeno funkcionalnost,

običajni uporabnik se je že registriral in plačal prijavnino, zato lahko najame vse javne avtomobile in dostopa do glavnih funkcionalnosti aplikacije, in

korporativni uporabnik, ki spada pod določeno partnersko podjetje, ki je zakupilo določeno število avtomobilov na določeni lokaciji podjetja, do katerih lahko ta uporabnik dostopa poleg drugih, javnih.

Slika 4.1 prikazuje primere uporab s strani vseh treh vrst uporabnikov.

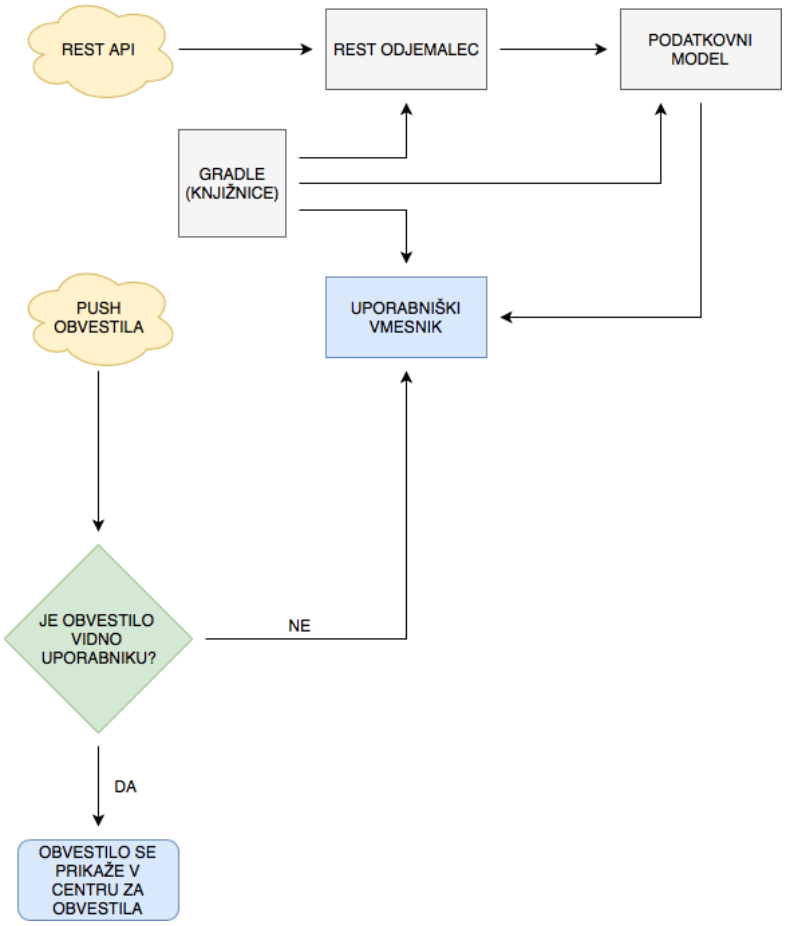


Slika 4.1: Diagram primerov uporabe

4.2 Arhitektura mobilne aplikacije

Naša celotna aplikacija bo s tehnološkega vidika zelo kompleksna, saj mora končnemu uporabniku omogočiti rezervacijo in upravljanje fizičnega avtomobila. To pomeni, da bomo morali poskrbeti za brezhibno komunikacijo med našo aplikacijo in strežnikom (REST API). Projekt mora biti prilagodljiv za prihodnje posodobitve, zato moramo natančno zasnovati arhitekturo, ki mora biti skalabilna in vzdržljiva.

Prva funkcija, ki jo mora arhitektura podpirati, je komunikacija z REST API, ki nam bo serviral vse podatke in nudil operacije nad avtomobili, s katerimi aplikacija nikoli ne komunicira neposredno – več o tem v poglavju 4.4. Pri tovrstni komunikaciji moramo poskrbeti, da odjemalec podpira osnovne operacije, kot so GET, POST, PUT in DELETE. Preko teh zahtevkov nam bo API serviral podatke v serializirani obliki JSON, ki jih bomo nato pre-



Slika 4.2: Shema komunikacije med sloji aplikacije

tvorili v lokalni podatkovni model oziroma Kotlin objekt.

Druga, redkejša oblika komunikacije s strežnikom, so poslana obvestila, ki aplikacijo opozorijo na določene dogodke, ki so se zgodili na strežniški strani, kot so na primer rezervacija avtomobila, zaključek najema ali denarna transakcija iz uporabnikove denarnice. Obvestilo je lahko poslano kadar koli, če je aplikacija uporabniku vidna ali ne, zato se moramo za vsako obvestilo posebej odločiti, kako ga bomo prikazali. Lahko ga prikažemo v Android centru za obvestila, kar je najpogostejša rešitev, ali pa znotraj aplikacije, tako da se posodobi uporabniški vmesnik. Za to obliko komunikacije se uporablja

Googlova storitev Firebase, ki deluje kot posrednik obvestil med strežnikom in odjemalcem.

Kompleksnejše rešitve, kot so na primer HTTP komunikacija, določene grafične komponente, animacije in deserializacija podatkov, so že implementirane v obliki knjižnic, ki jih v projekt vključimo preko orodja Gradle.

Srce aplikacije, v katerem se vse zgoraj naštete komponente arhitekture združijo v glavno uporabniško izkušnjo, je uporabniški vmesnik. To so vsi pogledi v aplikaciji, ki uporabniku prikazujejo podatke s strežnika in mu dovoljujejo vnaprej določene akcije. Kot je razvidno iz sheme na sliki 4.2, ima uporabniški vmesnik neposreden dostop do podatkovnega modela, ne ve pa, od kod so ti podatki prišli. Lahko jih dobimo preko REST odjemalca ali iz pomnilnika, v katerem so podatki začasno shranjeni. Ta del arhitekture je zelo pomemben, saj nam omogoča, da zamenjamo podatkovni ali komunikacijski sloj, ne da bi morali posodobiti grafični vmesnik, in obratno.

4.3 Podatkovni model

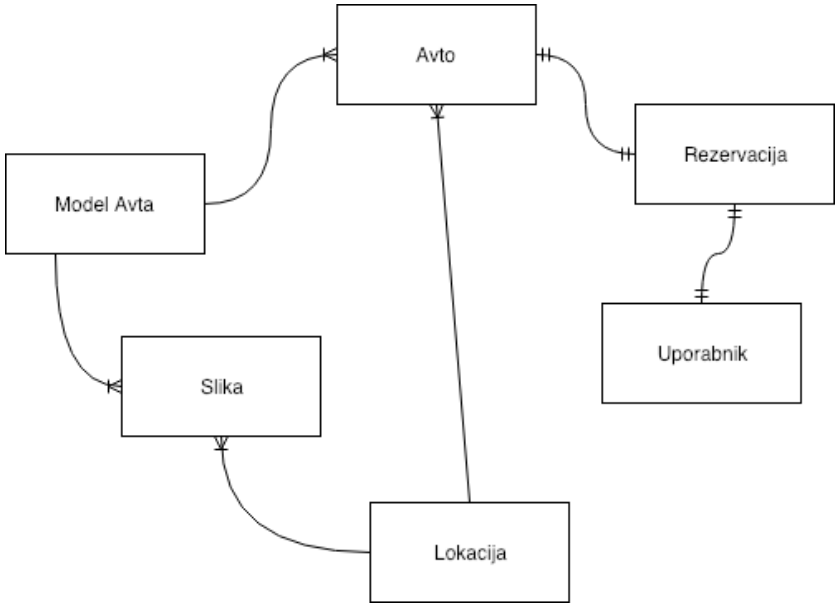
Aplikacija je zasnovana na način, da prikazuje samo žive podatke, ki jih v danem trenutku pridobi preko HTTP zahtevka, zato ni potrebe po podatkovni bazi. V začasnem pomnilniku aplikacije se v serializirani obliki JSON preprosto shranjujejo podatki, kot so:

podatki uporabnika, kot so ime, naslov in e-pošta,

osnovne nastavitve, ki se vsakič ob zagonu aplikacije preberejo iz strežnika in shranijo za kasnejši lažji dostop – to so določena pravila za delovanje aplikacije,

privzet jezik, ki ga lahko uporabnik spreminja v nastavitvah aplikacije.

Vsi ostali podatki, na primer podatki o avtomobilih, lokacijah in rezervacijah, so shranjeni v obliki Kotlin objektov, ki so inicializirani ob pretvorbi JSON besedila, ki ga dobimo preko HTTP zahtevka. Na sliki 4.3 je prikazana relacija med objekti, ki so v naši aplikaciji. Te relacije niso iste kot na



Slika 4.3: Podatkovni model

strežniku, saj na mobilnem odjemalcu hranimo manjšo količino podatkov. Medtem ko ima objekt avta na strežniku lahko referenco na več rezervacij (aktivne in neaktivne), nas na mobilnem odjemalcu zanima le ena rezervacija (aktivna), zato imamo relacijo 1:1. Vsak avto ima referenco na natanko eno izmed mnogih lokacij ter modelov, kateri imajo pa lahko referenco na več slik. Objekt uporabnika ima referenco do trenutne aktivne rezervacije in obratno. Teh referenc se seveda ne da omejiti s pravili, kot lahko to storimo z relacijo v podatkovni bazi, zato se je potrebno pravil držati oziroma sproti v programski kodi preverjati, ali je referenca ustvarjena. Kadar je ni, je potrebno ustrezno ravnati, sicer se lahko aplikacija sesuje ali pa uporabnik opazi manjkajoči podatek, kar smatramo za programskega hrošča.

Podatki se iz oblike JSON v Kotlin objekte in obratno pretvarjajo s pomočjo Googlove knjižnice Gson. Gson je deserializator, ki polja v JSON obliki preko ključev poveže s polji v Kotlin objektu. Če se ključa ujemata, podatkovna tipa pa ne, bo deserializacija neuspešna in se bo aplikacija sesula, zato je potrebno poskrbeti, da se podatkovni model v aplikaciji ujema

s podatkovnim modelom na strežniku.

4.4 Prenos podatkov med avtomobilom in zalednim sistemom

Aplikacija tako zaradi varnosti kot lažje implementacije nikoli ne komunicira neposredno z avtomobilom. V našem primeru se zaledni sistem (v nadaljevanju strežnik) obnaša kot posrednik med avtomobilom in aplikacijo.

Vsak avtomobil je opremljen s posebno strojno opremo, ki strežniku posreduje podatke, ki so nujno potrebni za delovanje sistema. To so na primer stanje odometra, stanje motorja (vključen/izključen), stanje ključavnice (odklenjen/zaklenjen) in GPS lokacija avtomobila, ki je potrebna, da se lahko avtomobil v aplikaciji prikaže na pravilni lokaciji.

Poleg zgoraj naštetih osnovnih podatkov ima strežnik dostop tudi do določenih osnovnih funkcionalnosti avtomobila. To sta v našem primeru samo odklep in zaklep ključavnice, zato da lahko uporabnik s pomočjo aplikacije prevzame in vrne avto.

Naša aplikacija bo torej strežniku predala zahtevo za podatek ali akcijo, počakala, da strežnik kontaktira avtomobil in opravi svoje delo, nato pa bo strežnik aplikaciji posredoval želeni podatek ali novo stanje. Takšen način komunikacije je optimalen za delovanje aplikacije in varnost, saj aplikacija ne more sama upravljati z avtomobilom. Vsako akcijo mora avtorizirati in izvesti strežnik, ki ga je veliko težje zlorabiti kot aplikacijo.

Poglavje 5

Ključne funkcionalnosti in pogledi

V tem poglavju bomo podrobno opisali ključne funkcionalnosti aplikacije. Začeli bomo pri navigaciji in pregledu nad vsemi lokacijami, na katerih se nahajajo avtomobili, ter rezervaciji avtomobila, za katerega se odločimo. Nato bomo opisali postopek začetka najema, upravljanje avtomobila ter na koncu še zaključek najema.

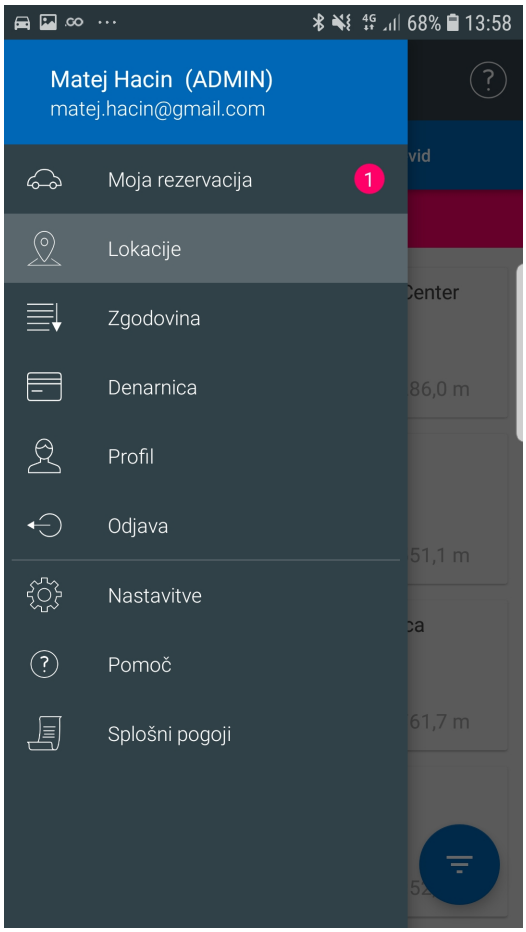
5.1 Navigacija

Če želimo svojim uporabnikom zagotoviti enostavno uporabniško izkušnjo, moramo glede na funkcije, ki jih nudi aplikacija, uporabiti ustrezno vrsto navigacije. Android nam ponuja že vgrajeno vrsto navigacije, ki se imenuje navigacijski predal in je prikazan na sliki 5.1.

Google je določil približne kriterije, ki jim mora aplikacija ustrezati, da lahko upravičeno uporabi to vrsto navigacije. To so:

- aplikacija ima pet ali več pogledov ki morajo biti uporabniku enostavno dostopni,
- aplikacija ima dve ali več stopenj navigacijske hierarhije,

- aplikacija mora podpirati hitro navigacijo med nepovezanimi komponentami.



Slika 5.1: Navigacijski predal

Naša aplikacija ustreza vsem trem kriterijem. Imamo osem višjenivojskih destinacij, za katere je pomembno, da so uporabniku dostopne ne glede na to, kje v aplikaciji se trenutno nahaja. Te destinacije so:

Moja rezervacija pripelje uporabnika na pogled, kjer lahko vidi trenutno aktivno rezervacijo in z njo upravlja.

Lokacije so glavni predel navigacije, ki uporabniku omogoča dostop do vseh

lokacij in rezervacijo avtomobila. Ta pogled je privzeto izbran ob zagonu aplikacije.

Zgodovina uporabniku omogoča hiter dostop do pregleda vseh preteklih rezervacij in statistike.

Denarnica uporabniku nudi pregled nad plačilnimi metodami in omogoča dodajanje nove plačilne metode.

Profil prikazuje vse uporabniške podatke in uporabniku omogoča menjavo gesla.

Nastavitve omogočajo uporabniku menjavo privzetega jezika v aplikaciji.

Pomoč je hitra dostopna točka do tehnične podpore,če uporabnik naleti na težavo.

Splošni pogoji so povezava do splošnih pogojev za uporabo aplikacije.

Gumba za odjavo ne uvrščamo med navigacijo, saj služi kot bližnjica za odjavo uporabnika iz aplikacije in uporabnika ne preusmeri na kak drug predel aplikacije.

Vsaka izmed zgoraj naštetih destinacij ima tudi svojo vrsto navigacije, ki uporabnika pripelje na novi pogled, smiselno povezan s trenutnim. To vrsto navigacije imenujemo navigacija druge stopnje, ki je tudi eden izmed zgoraj naštetih kriterijev.

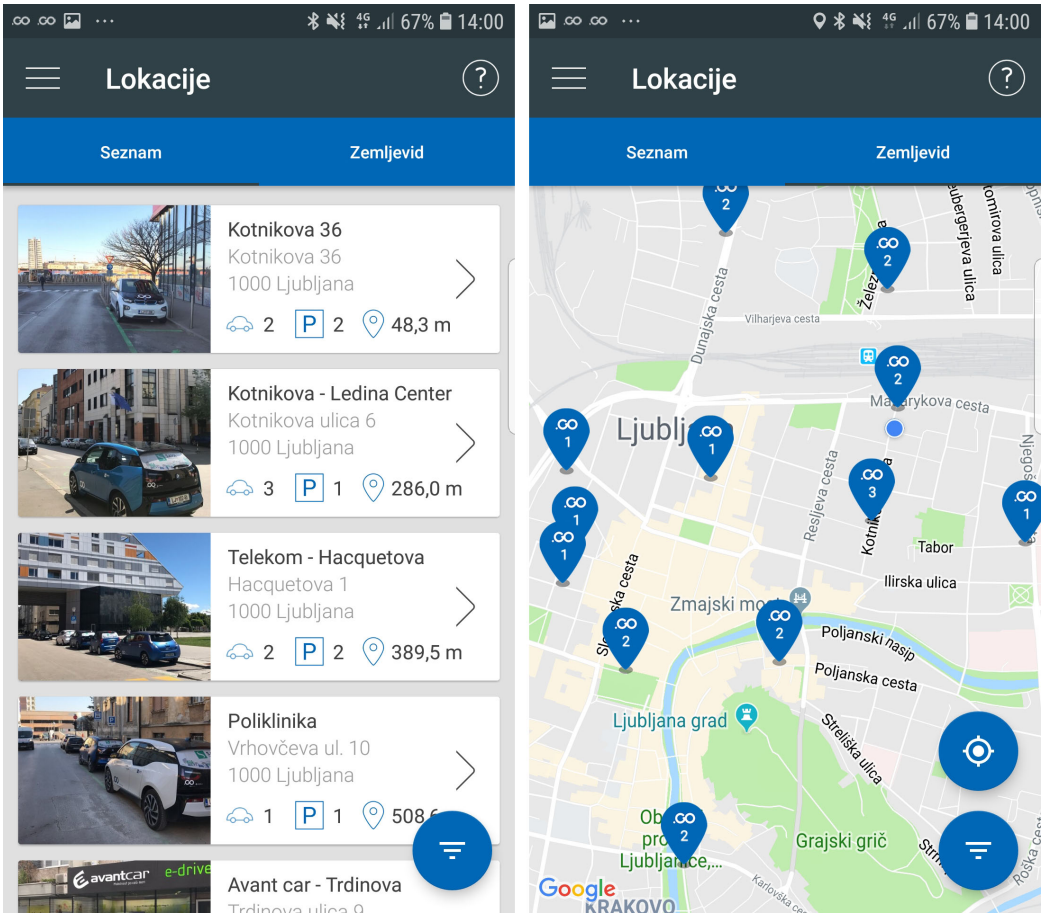
5.2 Pregled lokacij

Glavna uporabniška izkušnja se začne s pogledom, ki prikazuje vse lokacije, na katerih so parkirani avtomobili. Tu prikazujemo najpomembnejše informacije o lokaciji, kot so:

- naslov, na katerem se lokacija nahaja,

- prosta parkirna mesta, na katerih lahko uporabnik pusti rezervirani avto,
- prosti avtomobili, ki jih je možno rezervirati,
- geolokacija (na zemljevidu) ali oddaljenost od lokacije (na seznamu).

Imamo dve možnosti prikaza lokacij, vsak je namenjen drugačni vrsti uporabe.



Slika 5.2: Levo: seznam lokacij, desno: lokacije prikazane na zemljevidu

Seznam, ki ga vidimo levo na sliki 5.2, prikazuje lokacije vertikalno eno pod drugo, tako da se lahko uporabnik hitro pomika po seznamu. Na tem

pogledu vzamemo uporabnikovo geolokacijo, za katero se je prej strinjal, da jo deli z aplikacijo, zato da lahko lokacije razvrstimo glede na njegovo oddaljenost. Oddaljenost v metrih ali kilometrih je na seznamu jasno prikazana, da uporabnik takoj dobi občutek, koliko približno je oddaljen od lokacije. Ta vrsta prikaza je namenjena uporabnikom, ki lokacije že poznajo, zato vedo, kje jih lahko pričakujejo, ali uporabnikom, ki so trenutno v avtomobilu in potrebujejo informacijo o prostih parkirnih mestih za najbližjo lokacijo.

Druga vrsta prikaza lokacij, ki ga vidimo desno na sliki 5.2, je prikaz na zemljevidu. Tudi tu lahko uporabnik hitro dostopa do najbližje lokacije, saj ima gumb, ki zemljevid animira na trenutno uporabnikovo geolokacijo. Kljub temu je zemljevid namenjen bolj raziskovanju novih lokacij in hitrejšemu dostopu do bolj oddaljenih lokacij, saj v prvotnem pogledu prikazuje veliko manj podatkov kot seznam. Če uporabnika zanimata naslov lokacije ali število prostih parkirnih mest, mora pritisniti na označevalec na zemljevidu, zaradi česar je pregled lokacij v tem načinu pogleda veliko bolj zamuden.

Za lažje iskanje lokacij lahko uporabimo filtre, kot je vidno na sliki 5.3. Filtrira se lahko glede na:

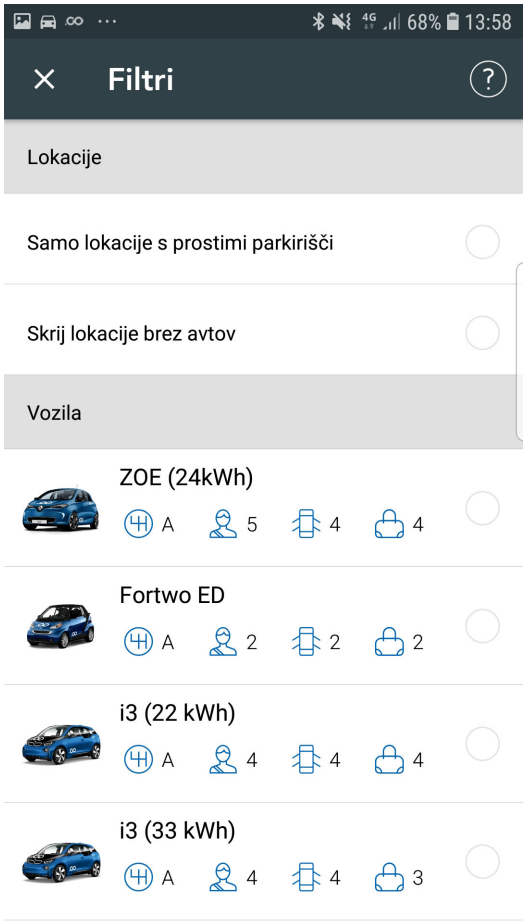
lokacije s prostimi parkirišči za lažje iskanje najbližjega parkirnega mesta,

lokacije z avtomobili za lažje iskanje najbližje lokacije, na kateri lahko prevzamemo avtomobil,

model avtomobila, če si želimo sposoditi specifični avtomobil.

5.3 Izbira avtomobila

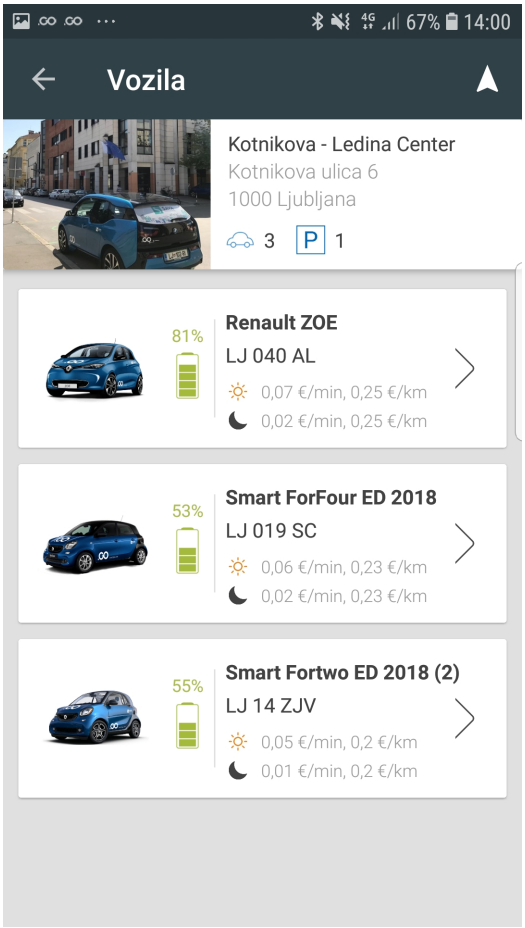
Ko izberemo lokacijo, pristanemo na seznamu avtomobilov, ki se tam nahajajo. Na sliki 5.4 vidimo, kako so le-ti prikazani. Uporabnik vidi le najbolj osnovne podatke, ki ga glede na določeni model avtomobila zanimajo. To so



Slika 5.3: Filtriranje lokacij

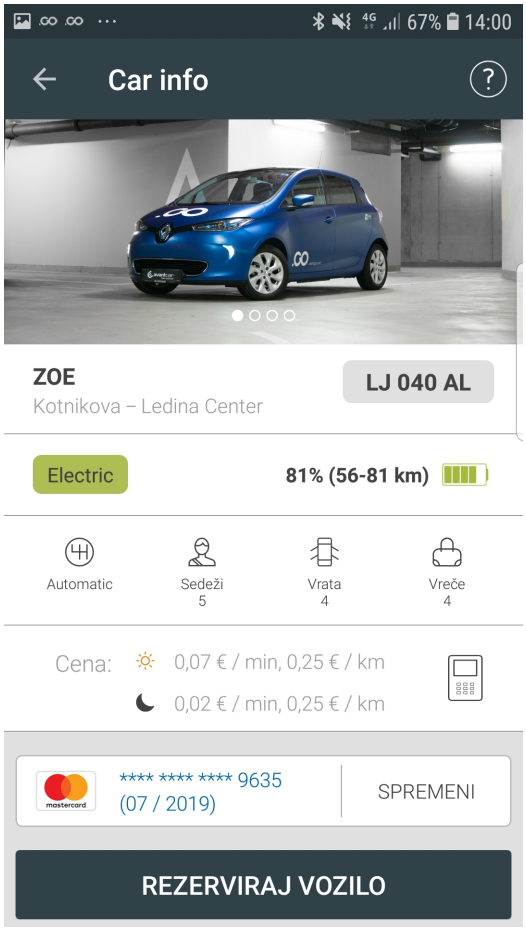
slika, ime, trenutno stanje baterije, registrska številka in cena. Stanje o bateriji je na tem mestu zelo pomembno, da lahko uporabnik glede na ta podatek udobno načrtuje svojo pot. Registrska številka je v tem koraku pomembna samo v primeru, če je uporabnik že fizično prisoten na lokaciji in si izbira avtomobil glede na trenutno vizualno stanje. Cena se seveda razlikuje glede na model avtomobila, zato je prav tako pomembna.

Ko si izberemo avtomobil, se nam prikaže še zadnji pogled pred rezervacijo, na katerem je vidnih še več podatkov o avtomobilu, kot so domet v kilometrih, vrsta menjalnika, število sedežev in število vrat. Najbolj po-



Slika 5.4: Seznam avtomobilov

membni funkciji tega vmesnega pogleda, ki ga vidimo na sliki 5.5, sta izbor plačilne metode in potrditev rezervacije. Uporabniki lahko dodajo več različnih plačilnih kartic, poleg tega naš sistem podpira tudi korporativne uporabnike, ki lahko najamejo avtomobil na račun podjetja. Ti uporabniki bodo pogosto menjavali plačilno metodo med osebno in korporativno. S pritiskom na gumb za rezervacijo uporabnik rezervira izbrani avtomobil za natanko 15 minut.



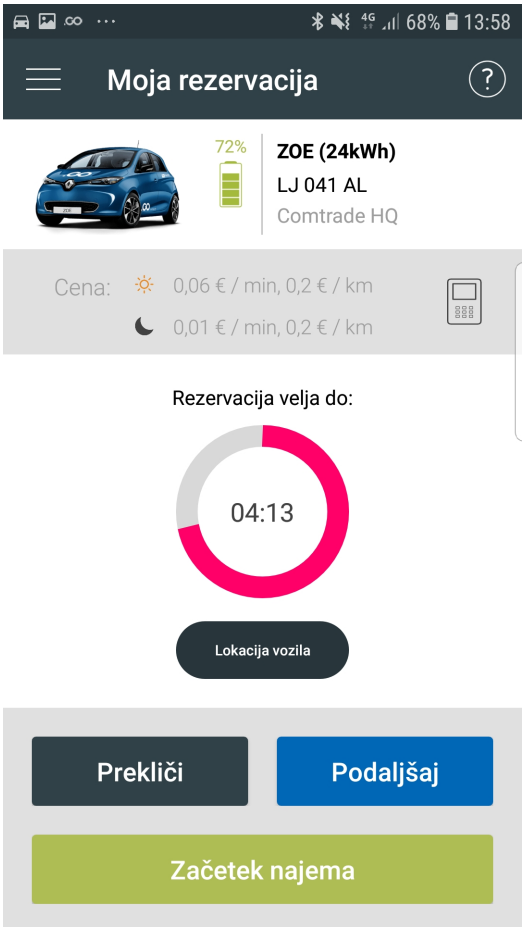
Slika 5.5: Zadnji korak pred rezervacijo avtomobila

5.4 Najem in upravljanje avtomobila

Celoten postopek najema avtomobila je razdeljen na tri korake: rezervacija, najem in zaključek najema. Vsi trije pogledi bojo prikazani in opisani v tem podpoglavju.

Takoj ko rezerviramo avtomobil, pristanemo na pogledu aktivne rezervacije. Tu vidimo, koliko časa še imamo, preden se naša rezervacija izteče, poleg tega imamo na voljo štiri gumbe, ki so vidni na sliki 5.6. To so gumbi za:

lokacijo avtomobila, ki odpre zemljevid z natančno prikazano geolokacijo



Slika 5.6: Pogled aktivne rezervacije

rezerviranega avtomobila,

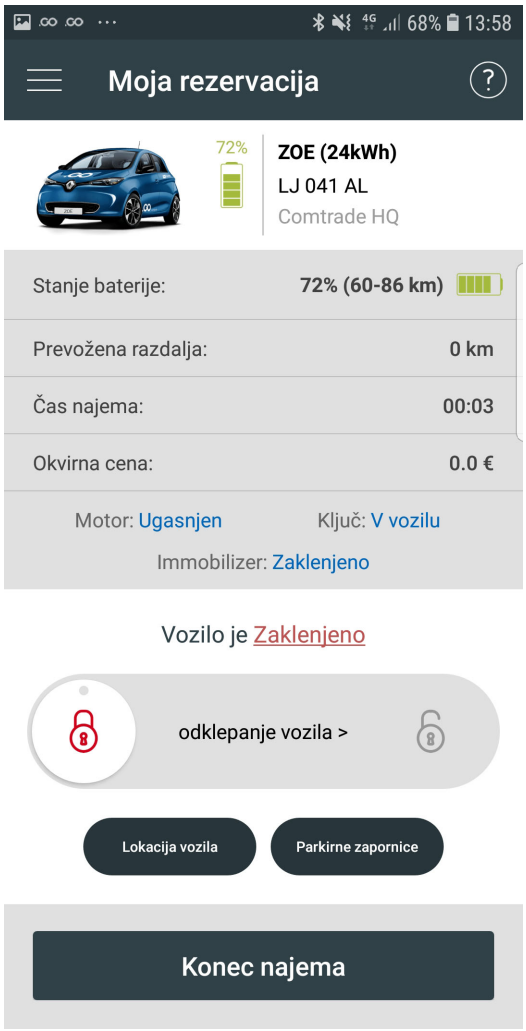
preklic za primer, če si premislimo in želimo rezervacijo preklicati brez dodatnih stroškov,

podaljšanje v primeru, da 15 minut ni dovolj in želimo rezervacijo podaljšati za dodatnih 15 minut, ter

začetek najema, ko smo pri avtomobilu in želimo začeti z najemom.

S pritiskom na gumb za začetek najema pristanemo na pogledu, ki ga vidimo na sliki 5.7, v katerem lahko začnemo upravljati z avtomobilom. Na tem

pogledu vidimo podatke o trenutnem stanju baterije, številu prevoženih kilometrov, času najema, približni dosedanji ceni najema ter statusu ključavnice in motorja. Ti podatki se vsako minuto osvežujejo, tako da če ostanemo na tem pogledu, ves čas vidimo dokaj posodobljeno stanje našega najema.



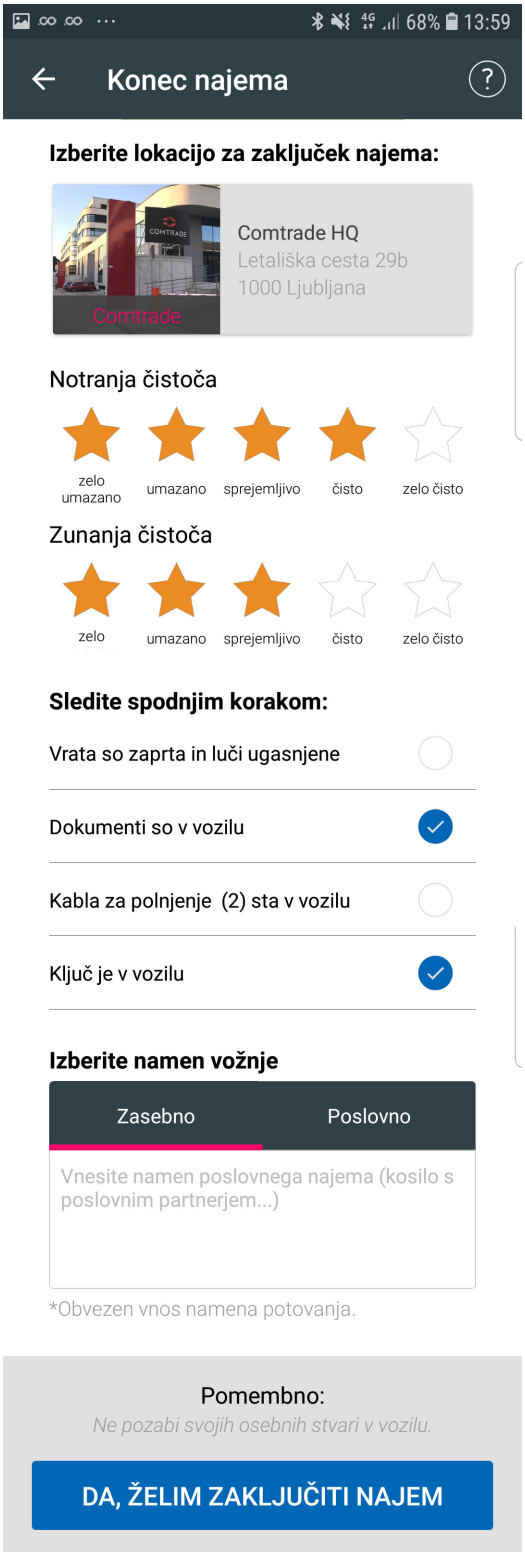
Slika 5.7: Pogled aktivnega najema

Za odklep avtomobila s prstom premaknemo gumb s ključavnico z leve proti desni. S tem pošljemo zahtevek za odklep našemu API-ju, ki zahtevek preusmeri nazaj na strojno opremo v avtomobilu, povezano z omrežjem

preko LTE. S tem je poskrbljeno za varnost, da telefon ne more sam, brez avtorizacije strežnika, odkleniti avtomobila.

Ker so nekatere lokacije zaprte z zapornicami, imamo na tem pogledu tudi gumb, ki odpre seznam parkirnih zapornic v bližini, ki so prav tako povezane z omrežjem. Klik na gumb odpre določeno zapornico in lahko začnemo z vožnjo.

Ko z vožnjo zaključimo in želimo avtomobil vrniti, pritisnemo na gumb za zaključek najema in pridemo na tretji pogled, ki je viden na sliki 5.8. Na tem pogledu je lokacija vrnitve avtomobila že avtomatsko izbrana glede na uporabnikovo geolokacijo, tako da uporabnik samo preveri in potrdi, da je izbrana lokacija pravilna. V nasprotnem primeru jo izbere sam. Nato samo še ocenimo čistočo avtomobila, potrdimo, da je avtomobil v stanju, pripravljenem za zaključek najema, ter izberemo način vožnje – poslovno ali zasebno. Ob pritisku na gumb za zaključek najema se vožnja obračuna, avtomobil pa se ponovno pojavi na lokaciji in je na voljo drugim uporabnikom.



Slika 5.8: Pogled pred zaključkom najema

Poglavje 6

Nadaljni razvoj

Testno obdobje je pokazalo, da aplikacija zadostuje vsem trenutnim potrebam uporabnikov, kljub temu pa imamo možnosti za izboljšave. V tem poglavju bomo na kratko opisali, kako bomo v prihodnosti izboljšali aplikacijo s tem, da pohitrimo delovanje in dodamo dodatne funkcionalnosti, ki uporabniku prinesejo dodatno vrednost.

6.1 Optimizacija delovanja

6.1.1 Testiranje enot kode

Testov za preverjanje posameznih enot kode še nismo pisali, saj to do zdaj ni bilo potrebno, poleg tega se je zaradi tega pohitril razvoj. Ker bomo v prihodnosti dodajali nove funkcionalnosti, katerih implementacija lahko negativno vpliva na trenutno delovanje aplikacije, bodo testi nepogrešljivi.

Poznamo več vrst testov, za nas bosta prišli v poštev dve:

testi enot kode, ki se spišejo za posamezno funkcijo ali za specifični del kode, ki ga nato kličejo s testnimi parametri in pričakujejo natančno določen rezultat, ter

UI testi, ki simulirajo uporabnikovo interakcijo z aplikacijo in ob tem preverjajo, če se uporabniški vmesnik pravilno posodablja.

Kodna baza aplikacije je zelo velika in se kljub previdno postavljeni arhitekturi velikokrat prepleta na različnih mestih. To pomeni, da lahko manjša sprememba na enem predelu aplikacije povzroči programskega hroščka na navidezno drugem predelu – temu pravimo regresija. S pisanjem testov regresije ne bomo preprečili, lahko pa tovrstne težave občutno zmanjšamo, kar pomeni, da bomo našim uporabnikom dostavili bolj stabilen produkt.

6.1.2 Shranjevanje podatkov v pomnilnik

Vse podatke, ki jih prikažemo uporabniku, dobimo preko REST klica na strežnik, kar pomeni, da mora uporabnik na vsak podatek čakati od nekaj milisekund do nekaj sekund. Izjeme so podatki o uporabniškem profilu, ki so obvezni za delovanje aplikacije, dokler je uporabnik prijavljen. V našem primeru je namreč izjemno pomembno, da uporabnik vidi trenutno stanje in ne zastarelega. Lokacije morajo prikazovati pravilne avtomobile, avtomobili pravilno stanje baterije in rezervacija pravilno stanje prevoženih kilometrov.

Kljub temu bi lahko na določenih predelih uporabili shranjevanje podatkov v pomnilnik:

Slike lokacij in avtomobilov so stalne oziroma se zelo redko spreminjajo, kar je idealen primer za shranjevanje v pomnilnik, da ni potrebno vsake slike naložiti vsakič, ko se aplikacija na novo zažene.

Zgodovina se nikoli ne spreminja, kar pomeni, da lahko v pomnilnik shranimo vse pretekle rezervacije in vsakič ob odprtju od strežnika zahtevamo samo nove.

Statistika se prav tako kot zgodovina redko spreminja, kar pomeni, da lahko trenutno stanje shranimo v pomnilnik, da ga uporabnik naslednjič vidi nemudoma.

Shranjevanje teh podatkov v predpomnilnik v našem primeru brez dvoma ni nujno, vendar bi aplikacijo v določenih primerih za uporabnika občutno pohitrilo, kar pa pomeni boljšo uporabniško izkušnjo.

6.2 Sistem proste souporabe

Sistem trenutno omogoča prevzetje avtomobila na vnaprej določenih lokacijah. S tem se pojavijo tako logistične težave (polne ali prazne lokacije) kot tudi slabša uporabniška izkušnja za ljudi, ki živijo na lokacijah, kjer v bližini ni lokacij za souporabo, ali pa se na takšne lokacije prevažajo.

Sistem proste souporabe bi omogočal uporabniku, da avtomobil vrne kjerkoli znotraj nekega vnaprej določenega območja. Avtomobili bi se tako po mestu bolj enakomerno razporedili, kar bi olajšalo logistiko prestavljanja avtomobilov, saj se lokacije ne bi več tako hitro polnile. Prav tako bi uporabniki, ki živijo daleč od najbližje parkirne lokacije, imeli potencialno lažji dostop do avtomobilov ter možnost vračila avtomobila pred svojim domom [3].

Poglavje 7

Zaključek

Cilj te diplomske naloge je bil predstaviti izdelavo prototipa mobilne aplikacije za souporabo avtomobilov, ki uporabnikom omogoča enostavno izkušnjo od rezervacije avtomobila do vračila na drugem mestu. Naloge smo se lotili tako, da smo najprej načrtovali arhitekturo mobilne aplikacije in si nato ogledali tehnologije, ki bodo uporabljene za implementacijo rešitve. Cilj naloge je uresničen, saj nam končni produkt omogoča zelo enostavno rezervacijo avtomobilov in deluje brezhibno. Pregledali in opisali smo tudi možnosti za optimizacijo in nadaljnji razvoj, na kar je aplikacija zaradi pravilno zasnovane arhitekture pripravljena.

Literatura

[1] Android Studio. Dosegljivo: `https://developer.android.com/studio/`, 2018. [Dostopano: 21. 11. 2018].

[2] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[3] Car Sharing in Europe. Dosegljivo: `https://git-scm.com/`, 2018. [Dostopano: 21. 11. 2018].

[4] The Cost of Car Ownership Over Time. Dosegljivo: `https://www.consumerreports.org/car-maintenance/the-cost-of-car-ownership/`, 2018. [Dostopano: 10. 12. 2018].

[5] Git. Dosegljivo: `https://git-scm.com/`, 2018. [Dostopano: 21. 11. 2018].

[6] Kotlin. Dosegljivo: `https://developer.android.com/kotlin/`, 2018. [Dostopano: 21. 11. 2018].

[7] Jason Morris. *Android User Interface Development: Beginner's Guide*. PACKT publishing Ltd, 2011.

[8] Leonard Richardson and Sam Ruby. *RESTful web services*. "O'Reilly Media, Inc.", 2008.