TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Emil Roland Peetsalu
155412IAPB

TTÜ LINNAKU MOBIILSE PARKIMISE RAKENDUS

Bakalaureusetöö

Juhendaja: Jüri Vain
PhD
Professor

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Emil Roland Peetsalu

20.05.2018

Annotatsioon

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 3 peatükki ja 16 joonist. Antud töö eesmärk on analüüsida parkimisolukorra parandamise võimalusi TTÜ-s ja pakkuda välja lahendus. Töös käsitletakse TTÜ parklate probleemi, Eestis kasutusel olevaid parkimislahendusi ja autori pakutud lahendust probleemi parandamiseks. Töös tuuakse välja, et ilma mõõduka finantsinvesteeringuta on probleemi raske parandada, kuid mobiilse parkimisrakenduse ja olemasolevate tõkkepuuga parklate targaks muutmisel on võimalik olukorda paremaks muuta.

Abstract

TTÜ campus mobile parking application

The thesis is in Estonian and contains 30 pages of text, 3 chapters and 16 figures. The purpose of this work is to analyze improvement possibilities for the parking situation in TTÜ and to propose a solution. The author's work describes the parking problem in TTÜ, parking solutions used in Estonia and the author's solution for improving the parking situation. Author's work states that it is difficult to improve the problem without a moderate investment, however it is possible to slightly improve the parking situation by using a mobile application and modifying current parking lots entries and exits with automatic or magnetic card controlled barriers.

Sisukord

Jooniste loetelu

Sissejuhatus

Eestis oli 2012. aasta seisuga liikluses 1000 elaniku kohta 456 autot. 2016. aastal oli 1000 elaniku kohta 534 autot [1]. Autode arvu kasvuga liikluses läheb tarvis rohkem parkimiskohti nende autode parkimiseks. Parkimiskohtade puudus tekitab probleemi.

TTÜ parklates on teatud hetkedel parkimiskohtade puudus. Näiteks argipäevade lõunaajal tiirutab TTÜ parklatest läbi palju autosid, kes otsivad endale vaba parkimiskohta. Kõigepealt otsitakse parkimiskohta endale kõige mugavamast parklast, mis on autojuhi sihtkohale kõige lähemal. Suures parklas ei saa autojuht teada vaba parkimiskoha olemasolust enne kui ta sõidab autoga läbi parkla. Kui esimeses parklas vaba parkimiskohta ei eksisteerinud, siis sõidab autojuht üldiselt läbi järgmise kõige lähema autoparkla kuni leiab endale vaba koha. Tulemuseks on parklates ebaefektiivne autoliiklus ja parkimisreeglite rikkumised, kui pargitakse näiteks parkimisjoonte kõrvale (vt. joonis 1 ja joonis 2).

Joonis 1. Parkimiskohtade puudus TTÜ ühiselamu parklas

Teine faktor on see, kui TTÜ-sse saabuv autojuht ei arvesta parkimiskohtade puudusega ja kulutab kaua aega vaba koha leidmisele, siis võib autojuht hakata parklas kiirustama. See kujutab endast ohtu teistele parklas viibivatele autodele ja jalakäijatele.

Joonis 2. Parkimiskorra rikkumine Tudengimaja kinnises parklas

Käesoleva töö eesmärgiks on uurida parkimisprobleemi lahendamise võimalusi TTÜ parklates aidates autojuhtidel kiiremini leida sobiv parkimiskoht ja seeläbi vältida parkimiskorra rikkumisi ja parklates kiirustamist.

Antud töö tulemusena valmib parkimisprobleemi lahenduste analüüs ning autori pakutud lahenduse prototüüp

Töö esimeses punktis analüüsib autor olemasolevaid parkimislahendusi Eestis, probleemi parimat lahendusviisi olemasolevate vahenditega, esitab oma lahenduse ja võrdleb pakutud lahendust analoogsete lahendustega.

Töö teises punktis kirjeldab autor pakutud lahenduse arhitektuuri, komponente, liideseid ja funktsionaalsust.

Töö kolmandas punktis esitleb autor parkimisrakenduse programmikoodi tähtsamaid osasid, tarkvara testimiseks kasutatud programmikoodi ja kirjeldab esitatud programmikoodilõike.

1 Analüüs

Käesolevas peatükis analüüsitakse parkimisprobleemi lahendamiseks disainitud erinevaid süsteeme. Analüüs on jagatud kolmeks osaks.

Olemasolevate lahenduste all uuritakse Eestis kasutusel olevaid parkimisprobleemi lahendusi.

Nõuete teostatavuse analüüsis vaadeldakse probleemi lahendusvõimalusi ja otsitakse nende parandamise võimalusi. Teostatavuse analüüs hõlmab majanduslikku, tehnilist ja ajalist teostatavust ning sotsiaalset mõju.

Alternatiivse lahenduse peatükis toob autor välja enda pakutud lahenduse TTÜ parkimisprobleemi parandamiseks. Pakutud lahenduse kohta on teostatud SWOT-analüüs ja võrdlus analoogsete süsteemidega.

SWOT-analüüsis hinnatakse pakutud lahenduse tugevusi, nõrkuseid, keskkondlikke võimalusi ja ohte.

Viimases alapeatükis võrreldakse pakutud lahendust analoogsete lahendustega. Lisaks hinnatakse lahenduse funktsionaalsust ja originaalsust.

1.1 Olemasolevad lahendused

Parkimisprobleemide lahendamiseks on maailmas kasutusel mitmeid meetodeid. Enamik neist keskenduvad majanduslikule kasule ning on disainitud kasutamiseks tasulistes parklates või kaubanduskeskustes, mille äri sõltub otseselt klientide parkimisvõimalustest poe lähedal. Probleemile saab läheneda ka teise nurga alt – ehitades linnadesse munitsipaal parklaid juurde, kuid antud töös analüüsib autor tehnilisi lahendusi, mida saaks rakendada valmisolevatele parklatele. Järgnevad neli lahendust on Tallinnas töö kirjutamise hetkel kasutusel.

Esimeseks näiteks on Ülemiste kaubanduskeskuse parkimismaja, mis kasutab vabade parkimiskohtade leidmiseks parkimiskoha andureid. Kaubanduskeskuse külalisi teavitatakse parkimiskohtade arvust numbritabloode abil. Parkimismajal on mitu korrust ja iga korrus on jagatud omakorda osadeks. Iga korruse ja iga korruse osa sissepääsule

eelneval tablool on näha vabade parkimiskohtade arv antud parklaüksuses. Parklasse sisenemisel on tablool näha kogu hoones olevate vabade parkimiskohtade arv. Iga parkimiskoha kohale on paigaldatud kohaandur, mis annab süsteemile teada parkimiskoha staatuse muutusest, kui anduri alla on jõudnud sõiduk või vastupidi, kui sõiduk on koha pealt lahkunud. Kohaanduri staatus muutub teatud hilistumisega mõne sekundi jooksul, et vältida kohtade arvu väärat muutumist, kui mõni sõiduk kasutab vaba kohta manööverdamiseks, ent tegelikult ei pargi antud koha peal. Kõikide parkimiskohtade kohal on ka indikaator, mille abil on kaugelt vaba kohta näha. Roheline tuli tähendab, et koht on vaba, punane tuli tähendab, et koht on hõivatud. See tõstab liiklemise efektiivsust parklates. Parkimiskoha staatuse muutumisel jõuab info süsteemi, mis uuendab parkimiskohtade numbrit vastavatel tabloodel. Antud parklas parkimine on kasutajale tasuta. Parkimishoone asub kaubanduskeskuse kõrval ja on mõeldud keskuse külastajatele.

Eespool kirjeldatud süsteemi analoog on kasutusel Solarise kaubanduskeskuse maa-aluses parklas. Erinevalt Ülemiste keskuse parkimishoonest on Solarise keskuse parkla tasuline. Solarise parkla puhul on märkimisväärne, et vabade parkimiskohtade arv on kajastatud veebilehel *www.kontserdimaja.ee,* mida on mobiiltelefoniga mugav vaadata enne autoga parklasse sõitmist.

Pisut erinev lahendus on kasutusel Viru keskuse parkimismajas. Vabade parkimiskohtade arv on kirjas tabloodel parklasse sisenemisel ning iga korruse ja korruse osa eel. Erinevuseks eelnevalt kirjeldatud lahendusega võrreldes on üksikute kohaandurite puudumine. Vabade parkimiskohtade arv arvutatakse tõkkepuude avanemiskordade abil. Parklasse sisenemine või parklast väljumine toimub läbi vastavasuunaliste tõkkepuude. Teades parkimishoone parkimiskohtade koguarvu liidetakse sellele parklasse sisenev auto või lahutatakse sellest arvust parklast väljuv auto. Antud lahendus on odavam, kui kohaanduritega parkla, sest kohtade lugemiseks kasutatakse ainult tõkkepuid. Antud süsteemi nõrkuseks on ebaefektiivne ajakulu sellistes parklates, sest juhtidel on raskem leida vaba parkimiskoha asukoht.

Kolmas lahendus on eestlaste idufirma Barking poolt arendatud mobiilirakendus *Barking*, mis võimaldab eraparkla omanikel välja rentida parkimiskohti oma parklates [2]. Rakendus annab kasutajale linnakaardi peal teada *Barking* süsteemiga parklate asukoha ja hinnakirja. *Barking* süsteemi kasutavate eraparklate tõkkepuud on liidetud Barking

modemiga, mis võimaldab mobiilirakenduse kasutajal tõkkepuud avada. Tõkkepuu avamisel nõustub kasutaja krediitkaardi või mobiiliarvega parkimise eest tasuma. Teades parkla kogu parkimiskohtade arvu saab vabade kohtade arvu arvutada vastavalt parkimisteenust tasuvate inimeste arvule. Lisavõimalusena saab teenusepakkuja poolt määratud tasu eest sellistes parklates broneerida vaba koha. Selline süsteem on automatiseeritud ja nõuab vähe inimsekkumist. Nõrkusena ei ole *Barking* rakenduses kajastatud vabade parkimiskohtade arv eri parklates. Rakendus annab infot ainult selle kohta, kas parkla on aktiivne või mitte. Seega võib parkla kohale jõudmise ajaks muutuda hõivatuks. Samuti on võimalik süsteemi petta, kui avada tõkkepuu, sõita autoga parklasse ning seejärel rakendusest parkimine lõpetada. Siiski on tegemist väikeste parkimistasudega ning mõne üksiku indiviidi poolt süsteemi petmine ei tähenda *Barking* rakenduse ja eraparkla omanikele suurt kahju. Sellise süsteemi tugevusteks on see, et *Barking* aitab eraparklate omanikel tõsta oma parkla kasumlikkust rentides välja tühjasid parkimiskohti. Samuti on *Barking* parklates väiksem parkimistasu, kui ümbritsevates parklates. Eriti on seda märgata südalinna ja vanalinna parklate puhul.

1.2 Nõuete teostatavuse analüüs

Antud peatükis otsime TTÜ jaoks parimat lahendust parkimissituatsiooni parandamiseks. Vaatlused hõlmavad majanduslikke, tehnilisi ja ühiskondlikke tegureid.

Kõige ideaalsem lahendus TTÜ külastajale vaba parkimiskoha kättejuhatamiseks oleks kasutada kohaandurite süsteemi nagu on Ülemiste keskuse parkimismajas. Selline viis võimaldab tabloode ja indikaatorite abil autojuhile näidata ette vaba parkimiskoha täpse asukoha. Majanduslikust vaatenurgast on aga see kõige kallim variant.

Odavam võimalus oleks parklate sisse- ja väljapääsudesse seada tõkkepuud. Teades parklates kogu parkimiskohtade arvu ja tõkkepuu avanemiste kordi, võimaldaks selline lahendus määrata umbkaudse vabade parkimiskohtade arvu parklas. Peab arvestama võimalusega, et igal ajahetkel võib parklas peatuda autosid, mis toovad inimesi kooli või laevad maha kaupa. Seega, kui parklas on näiteks 200 parkimiskohta ning parklasse on sisenenud 200 autot aga ükski neist pole väljunud, ei saa olla kindel, et kõik 200 parkimiskohta on kasutusel.

Otsides lahendust parkimisprobleemile TTÜ ülikoolilinnakus, tuleb arvestada, et ülikooli eelarve on eelkõige mõeldud hariduse edendamiseks ning kooli infrastruktuuri haldamiseks. Samuti on arvestatav osa ülikooli tudengitest ja töötajatest jalakäijad, või ühiselamu kasutajad, kes ei mängi TTÜ parklate autoliikluses suurt rolli. Ülikooli parklatesse investeerimine ei ole seetõttu juhatuse jaoks prioriteet.

TTÜ haldusdirektori sõnul on ainus võimalus kohaanduritesse või tõkkepuudesse investeerimiseks minna üle tasulisele parkimisele. See aitaks ülikoolil teenida tagasi parkimissüsteemi arendusele kulunud raha [3]. See vähendaks kindlasti ka parklat kasutavate autode arvu. Teisalt raskendaks TTÜ-s tasulise parkimiskorra tekitamine tudengite elu, kellest enamusel on õpingute ajal madal sissetulek.

Arvestades ülalkirjeldatud suhtumist , on TTÜ-s võimalik lahendada parkimisprobleem minimaalsete kulutustega ning rakendada kohalugemise süsteem olemasolevatele tõkkepuuga parklatele, mida kasutavad TTÜ töötajad.

Tehniliselt on võimalik tõkkepuuga parklate puhul jälgida parklasse sissesõidul tõkkepuu avamisi ning arvestada nende arv maha kogu parkla parkimiskohtade arvust. Probleemiks on siin see, et erinevalt Viru kaubanduskeskuse parklast, kus sisse- ja väljapääsu jaoks on eraldi tõkkepuud, on TTÜ töötajate jaoks mõeldud parklad ühe tõkkepuuga. Ilma andurita ei saa ühe tõkkepuu avanemise korral teha kindlaks, kas auto sisenes parklasse või väljus parklast. Töötava lahenduse jaoks on tarvis määrata tõkkepuu alt läbi sõitva auto suund – sisse või välja. Üks võimalus on rakendada parkla sissepääsule liikumisandurid, mis jälgivad tõkkepuu avanemise korral sõiduki liikumise suunda. Teine võimalus on kasutada olemasolevaid kaardilugejaid, mille abil avatakse tõkkepuu, kas parklast seest- või väljastpoolt.

Teades tõkkepuuga parkla parkimiskohtade koguarvu ning sisenemisi ja väljumiseid saame määrata ligikaudse vabade parkimiskohtade arvu saame kajastada seda infot parkla sissepääsu kõrval tablool või Internetis. Viimase puhul tuleb tõkkepuu liidestada modemiga, et võimaldada info saatmist serverile.

Kuna eesmärk on vähendada TTÜ parklates ebaefektiivset autoliiklust, siis peaks autojuhtidel olema võimalus teada parkla olukorda enne, kui nad sisenevad parklasse. Sel juhul oleks otstarbekas hoida parkimiskohtade infot Internetis. Kuna vabade parkimiskohtade arv on suurtes parklates tihti muutuv, peaks võimaldama autojuhil

vaadelda parkimiskohtade arvu mõni minut enne parklasse sisenemist. Seega on mõistlik kasutada info kajastamiseks mobiilirakendust.

Otstarbeka mobiilirakenduse loomisel tuleks järgida järgmisi printsiipe. Teenuse katkestusi selle tarbimise ajavahemikel peaks olema minimaalselt. Kui see peaks mingil põhjusel juhtuma, siis tõkkepuud avanevad ja autod peavad saama parklast väljuda. Kasutajaliides peab olema intuitiivne ning lihtsasti kasutatav. Kasutajaliides peab infot esitlema selgesti loetavalt, sarnaselt navigeerimisrakendustele (Google Maps, Waze), mis on disainitud autojuhile sõidu ajal jälgimiseks. Rakendus ei tohi kajastada aegunud infot.

Kuna enamik TTÜ parklate kasutajaid pargib avalikus parklas, võiks mobiilirakendus sisaldada neile kasulikku funktsionaalsust. Arvestades, et avatud parklas ei saa vabade parkimiskohtade arvu jälgida ilma kalli tehnilise lahenduseta, peaks mobiilirakendus aitama TTÜ-d külastavatel autojuhtidel näha võimalike parklate asukohti. Kuna me arendame mobiilirakendust, saame selle kasutajale pakkuda parkimislogide ja parklate asukoha info põhjal sihtkohale lähima parkla.

1.3 Alternatiivne lahendus

Autor pakub TTÜ parkimisprobleemi lahendamiseks välja mobiilirakenduse, mis küsib selle kasutajalt tema soovitud sihtkoha TTÜ-s ning tagastab kasutaja sihtkohale lähimate parklate asukohad. Tagastatud parklate asukohad on reastatud alustades lähimast parklast ja lõpetades kaugeima parklaga. Sihtkoha saab kasutaja märkida Google Maps API kaardil ja parklate soovitused tagastatakse kasutajale samuti Google Maps kaardil. Lisaks on kasutajal võimalus märkida kõrgendatud parkimisõigused ehk ligipääs kinnistesse parklatesse. Kõnealustele kasutajatele tagastatakse parklate soovituses ka TTÜ tõkkepuuga parklad, millele on lisatud vabade parkimiskohtade arvu hinnang. Iga parkla soovituse puhul on kasutajaliideses kajastatud parkla kaugus kasutaja märgitud sihtkohast linnulennult. Samuti vahemaa kõndimisele kuluv aeg, eeldusel, et inimene kõnnib lühikest vahemaad keskmise kiirusega 5 km/h. Vahemaa hinnang ei arvesta tee peale jäävate takistustest möödumist.

Tõkkepuuga parklaid on TTÜ-s kahte tüüpi. Esimene parklatüüp kasutab parklasse sisenemiseks ja tõkkepuu avamiseks kaardilugejat ning parklast väljumiseks on asfalti sisse paigutatud magnetkontuur, mis tunnetab auto lähenemist tõkkepuule [4]. Teine

parklatüüp kasutab parklasse sisenemiseks ja parklast väljumiseks eraldi tõkkepuid kaardilugejatega. Mõlemal juhul tuleb kaardilugejate ja magnetkontuuride töö edastada modemile, mis teatab serverit tõkkepuu avanemise puhul. Server arvestab ligikaudset vabade parkimiskohtade arvu tõkkepuuga parklates ning vastab mobiilirakendusest tehtud päringutele ja saadab tagasi vabade parkimiskohtade arvu.

Täielik süsteem koosneb mobiilirakendusest, serverist, modemitest ja liikumisanduritest.

Autor pakub välja TTÜ informaatika eriala magistriõppe tudengitele tarkvaraarenduse meeskonnaprojektiaine raames arendada välja pakutud lahenduse tarkvara ja veebiteenus ning lisada täpsed liikumisjuhised parklast rakenduse kasutaja sihtkohani, mis arvestab teele jäävaid takistusi.

Edasiarendusena võiks kaardistada kõik Eesti parklad lastes seda teha kasutajatel. Kasutaja saaks veel kaardistamata parklas asudes saata rakenduse abil serverile parkla koordinaadid, mis jõuavad rakenduse parklate andmebaasi. Et vältida rakenduse puhul ebasobilike parklate soovitamist kasutajale, saavad teised rakenduse kasutajad hinnata märgitud parklat – kas sobib või mitte. Taoline kasutaja poolt lisatud teavituste hindamissüsteem on kasutusel *Waze* navigeerimisrakenduses, mida kasutatakse kaardile liiklusohu või politseinike märkimiseks.

1.3.1 Pakutava lahenduse SWOT-analüüs

Tugevused
Teostuse ja hallatavuse lihtsus ja mugavus.
Mugav ja lihtne kasutajaliides.
Uudsed lisafunktsioonid: kasutaja sihtkohale lähima parkla soovitamine.
Võimalus jälgida nutiseadmes vabade parkimiskohtade arvu.
Lisaks tõkkepuuga parklatele on soovitustes ka avalikud parklad.
Süsteem on täiendatav konkreetsete parkimiskohtadele viitamise funtsionaalsusega.

Nõrkused
Rakendus sisaldab vähe funktsionalsust.
Peatuvate autode tõttu ei ole võimalik tõkkepuuga parklas jälgida täpset vabade parkimiskohtade arvu.
Suurte avalike parklate puhul on vabade parkimiskohtade arvust ja asukohast teavitamiseta keskuse regulaarsele külastajale väheinformatiivne.
Võimalikud ehitustööd parklates ja hoonetes nõuavad aktiivset andmebaasi uuendamist.
Vabade parkimiskohtade määramise süsteem pole reaalselt implementeeritud.
Tõkkepuuga parklate liikumisandurite reaalset rakendamist pole uuritud.
Serveri ülalhoidmine on pidev kulu.

Võimalused
Süsteemi on rakendatav ka muudes keskustes peale TTÜ
Parklate info andmebaasiga võiks integreerida kaubanduskeskuste parkimisinfo, et Eesti parklate info oleks ühest kohast kättesaadav.

Ohud
Eraparklate omanikud ei pruugi olla nõus kohalugemisandurite paigaldamise või nende haldamisega seotud kuludega.

1.3.2 Võrdlus analoogidega

Võrreldes teiste parkimisrakendustega on selle rakenduse puhul uudne sihtkohta jõudmiseks lähimate parklate soovitamine koos liikumisjuhistega. Populaarseimad parkimisrakendused Eestis nagu *Barking* ja *PARGI.EE* näitavad ette kuidas liikuda autoga kindlasse parklasse. Autori poolt pakutud rakendus näitab ette, kuidas jõuda parklast sihtkohani.

Barking rakendus sisaldab analoogset kohalugemise süsteemi, kuid selle puhul pole teada vabade parkimiskohtade arvu. Rakendus teatab, kas parklas on vaba koht või mitte. See olukord võib lühikese aja jooksul muutuda ning kasutaja peab seda võimalust ise meeles pidama. *Barking* rakendus näitab sihtkohale lähimaid tasulisi *Barking* süsteemiga parklaid. Autori rakendus soovitab ka tasuta parklaid.

PARGI.EE rakendus teatab kasutajale tema kauguse lähimast parklast ning võimaldab linnades tasulise parkimistsoonides parkimise eest tasuda. Kõnealuse rakenduse puhul peab kasutaja endale sobiva parkla ise leidma.

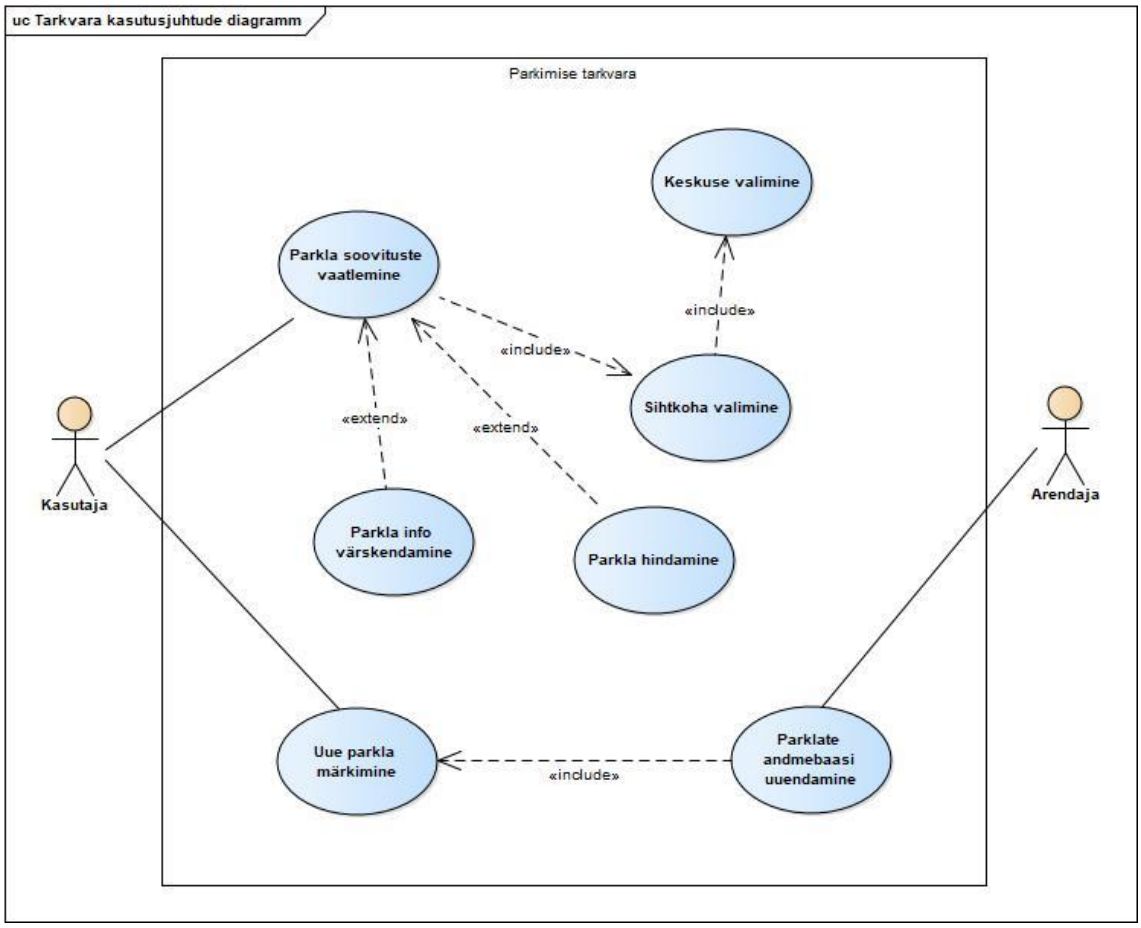
2 Parkimissüsteemi projekteerimine

Järgnevates peatükkides kirjeldatakse pakutud süsteemi põhilisi projekteerimissamme, tuuakse välja süsteemi mobiilirakenduse arhitektuur, liideste ning funktsionaalsuse kirjeldused.

2.1 Kasutuslood

Kasutusjuhtude esialgne loend on järgmine:

- Parkimissoovituste vaatlemine
    - Parkla soovituste vaatlemine
    - Keskuse valimine
    - Sihtkoha valimine
    - Parklate info värskendamine
    - Parkla hindamine
- Parklate nimekirja uuendamine
    - Parklate andmebaasi uuendamine
    - Uue parkla kaardistamine kasutaja poolt



Joonis 3. Kasutusjuhtude diagramm

Diagrammil olevate kasutusjuhtude lühikirjeldused on järgmised:

Parkla soovituste vaatlemine:

Kasutaja on parkimisrakenduse avanud, valinud keskuse, märkinud kaardil sihtkoha ja soovib näha parklate soovitusi. Parklate soovitused ja Kasutaja valitud sihtkoht märgistatakse Google Maps-i kaardil ning kuvatakse ühekaupa. Iga parkla soovituse puhul on rakenduses kirjas vaadeldava parkla kaugus sihtkohast, aeg kõndimiseks, parkla tüüp (avalik või eraparkla) ning parkla kohtade koguarv avaliku parkla puhul ja vabade parkimiskohtade arv eraparkla puhul. Parklad on reastatud lähimast parklast kõige kaugeimani ja soovituste vahel saab liikuda ’EELMINE’ ’JÄRGMINE’ nuppude abil.

Keskuse valimine:

Kui Kasutaja avab parkimisrakenduse, siis on tal võimalus valida keskus, mille ümber kuvatakse hiljem Kasutajale parklate soovitusi.

Sihtkoha valimine:
Kui Kasutaja on kinnitanud keskuse, mille lähedale ta soovib parkida, siis avatakse rakenduses Google Maps-i kaart kinnitatud keskuse asukohalt. Kasutaja saab ekraanil kaarti liigutades valida omale sobiliku sihtkoha. Täpse sihtkoha määrab kaardi keskel asuv tähis.

Parklate info värskendamine:
Kui Kasutaja vaatleb parklate soovitusi, siis eraparkla puhul saab Kasutaja värskendada antud parkla vabade parkimiskohtade arvu. Sel juhul saadetakse serverile päring vaadeldava eraparkla vabade parkimiskohtade arvu saamiseks.

Parkla hindamine
Kasutajal on võimalus hinnata soovitatud parklaid. Hinnanguteks on 'Hea' või 'Halb'.
See aitab vältida olukorda, et parklate soovitustes esineb palju kasutajate poolt lisatud valeparklaid.

Parklate andmebaasi uuendamine:
Võimalusel, et keskuste ümber olevaid parklaid suletakse mõnikord haldus- ja ehitustöödeks või lisatakse juurde, on võimalik rakenduse parklate andmebaasi uuendada. See aitab vältida olukorda, kus Kasutajale soovitatakse reaalselt suletud parklat.

Uue parkla märkimine:
Kui Kasutaja asub parklas, mis pole veel rakenduse parklate andmebaasi lisatud, siis on Kasutajal võimalik uues parklas asudes saata serverile oma telefoni asukoha GPS koordinaadid.

2.1.1 Olulisemad funktsionaalsed ja mittefunktsionaalsed nõuded

Mobiilirakendus peab olema realiseeritud Javas ning toetatud Android operatsioonisüsteemide poolt.

Rakendus peab omama minimalistlikku graafilist kasutajaliidest

Rakendus peab olema testitud ning ei tohi omada kriitilisi vigu koodis, mille tagajärjel rakendus kokku jookseb.

Rakendus peab kasutaja soovitud sihtkoha sisestamisel tagastama parklate soovitused 2 sekundi jooksul.

Rakendus koosneb kasutajaliidesest, mis jookseb mobiiltelefonil ja kaugpöördutavast serverist.

Kasutajaliides võimaldab kaardile märkida sihtkoha, kuhu kasutaja soovib parkimiseks minna.

Rakendus peab soovitama kasutaja sihtkohale geograafiliselt lähima vabade parkimiskohtadega parkla.

Rakendus peab kasutama kindlas formaadis kaardifaile.

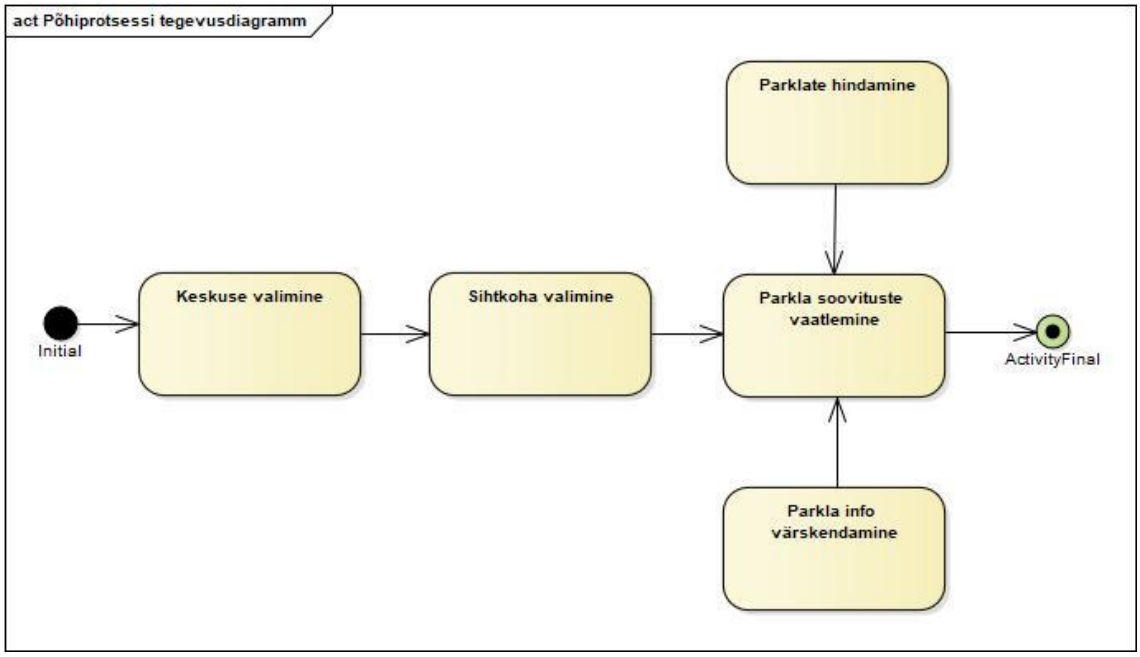
Rakendus tagastab jalutamistee pikkuse parklast sihtkohani

Rakendus tagastab parklast sihtkohani jalutamise ajakulu hinnangu eeldusel, et liikumiskiirus on 5 km/h.

Süsteem demoversioon sisaldab kaarti TTÜ hoonete ja parklatega.

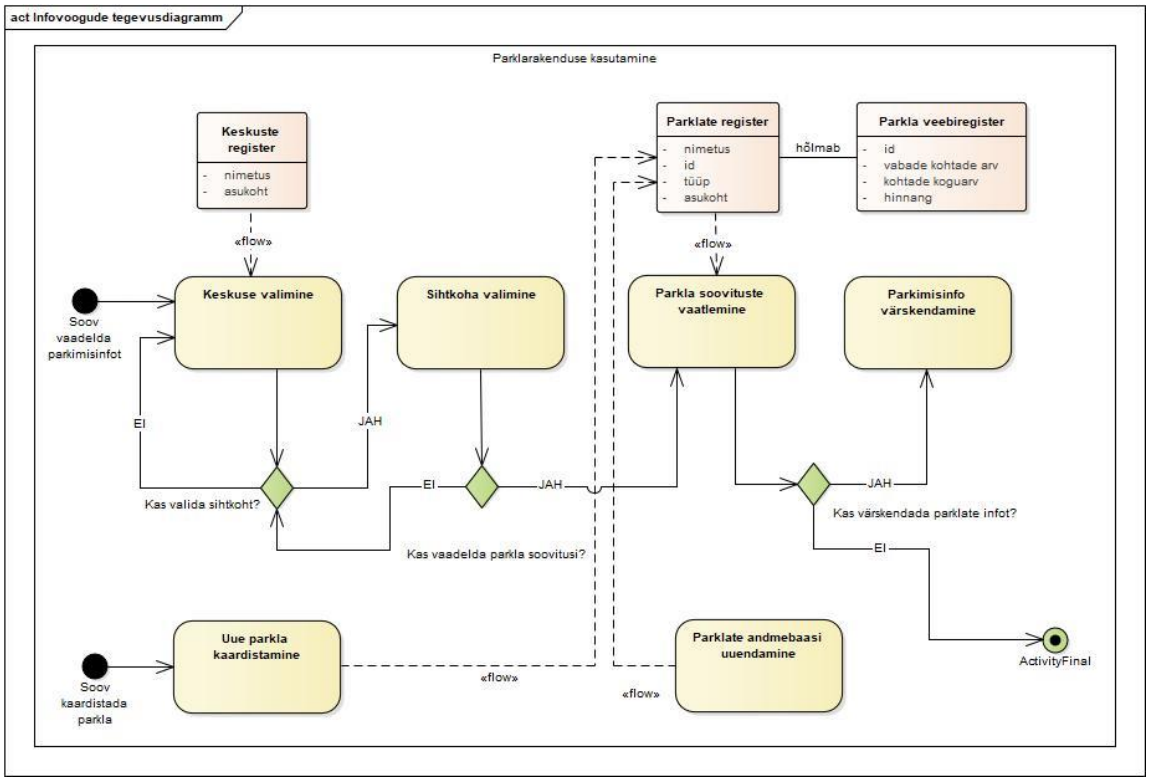
Rakenduses peab saama parkla soovitusi filtreerida eraparklate ja vabakasutusparklate vahel.

2.1.2 Tegevusdiagrammid



Joonis 4. Põhiprotsessi tegevusdiagramm

Sellel tegevusdiagrammil näidatakse parkimisrakenduse põhiprotsessi töövoogu.

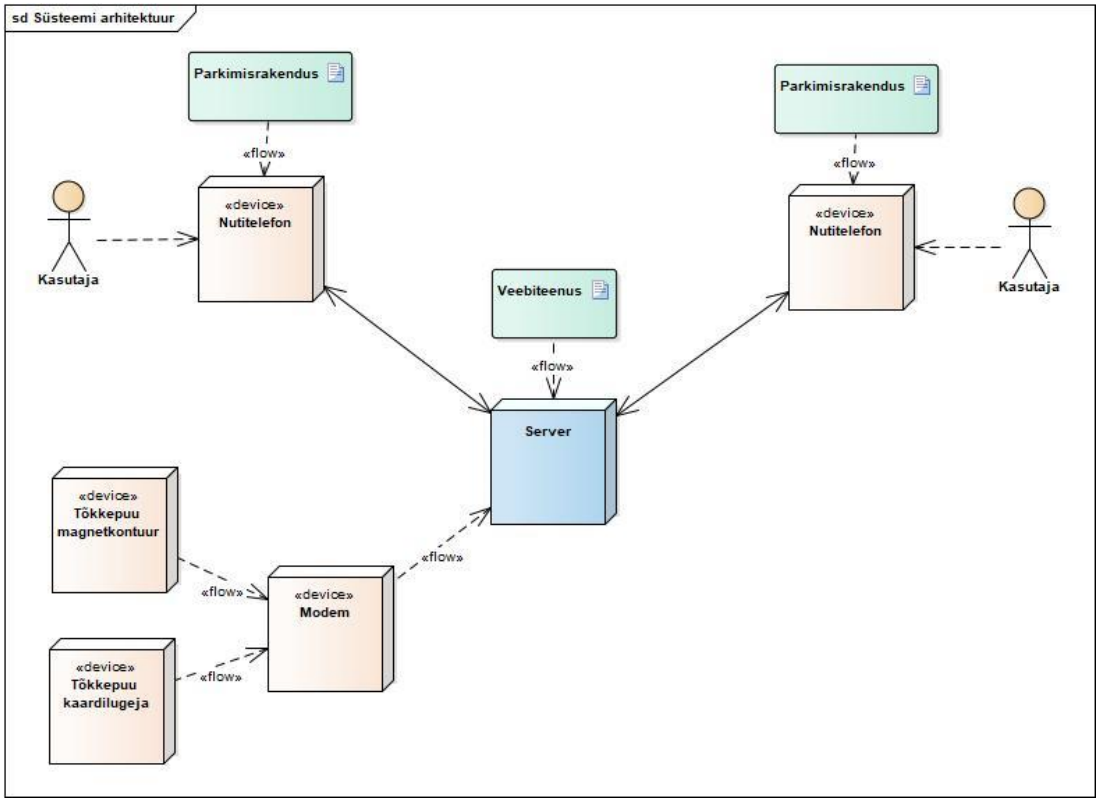


Joonis 5. Infovoogude tegevusdiagramm

Sellel tegevusdiagrammil näidatakse töövoogusid koos infovoogude ja otsustega.

2.2 Süsteemi arhitektuur

Autori pakutud tarkvarasüsteem parkimisprobleemide lahendamiseks kasutab klient-server arhitektuuri. Klient on parkimisrakendust kasutav nutitelefon, mis pärib serveri veebiteenuselt infot parklate kohta. Server saadab kliendile info, mis sisaldab parkla hinnanguid, avalike parklate parkimiskohtade koguarvu ja eraparklate puhul vabade parkimiskohtade arvu. Modemite abil liigub serverisse info tõkkepuude kasutuse kohta kaardilugejatest ja magnetkontuuridest, mis on kasutusel TTÜ kinnistes parklates.



Joonis 6. Süsteemi arhitektuur

Süsteemi kuuluvate komponentide lühikirjeldused on järgmised:

Nutitelefon:

Nutitelefon (ingl. smartphone) on seade, mis võimaldab käivitada Android operatsioonisüsteemil põhinevaid rakendusi ja saata neist päringuid veebiteenusele.

Parkimisrakendus:

Parkimisrakendus tuleb paigaldada Kasutaja nutitelefoni. Parkimisrakendus on tarkvara, mis annab lisaks parklate soovitustele ka juhised nutitelefonile serverilt info pärimiseks ja ekraanil saadud vastuse kuvamiseks.

Server:

Tõkkepuu avanemise korral jõuab serverisse info parklasse sisenenud või parklast väljunud auto liikumissuuna kohta. Server vastab nutitelefonide päringutele.

Veebiteenus:

REST tüüpi veebiteenus võimaldab antud süsteemis nutitelefonide(Klient) ja serveri-vahelt suhtlust.

Tõkkepuu magnetkontuur:
Eraparklatest väljumisel tajub asfalti paigutatud magnetkontuur sõiduki lähenemist tõkkepuule. Tõkkepuu avanemise korral jõuab info sellest modemile.

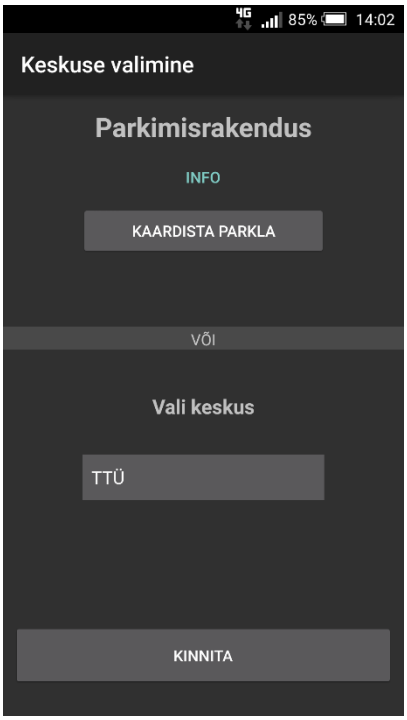
Tõkkepuu kaardilugeja:
Eraparklatesse sisenemiseks kasutatakse TTÜs kaardilugejaid. Info kaardilugeja kasutusest jõuab modemile. Erandina toimub ka parklatest väljumine kaardilugeja abil.

Modem:
Modem saadab tõkkepuu avanemisel kaardilugejalt või magnetkontuurilt saabunud info üle Interneti serverile.

2.3 Liidesed

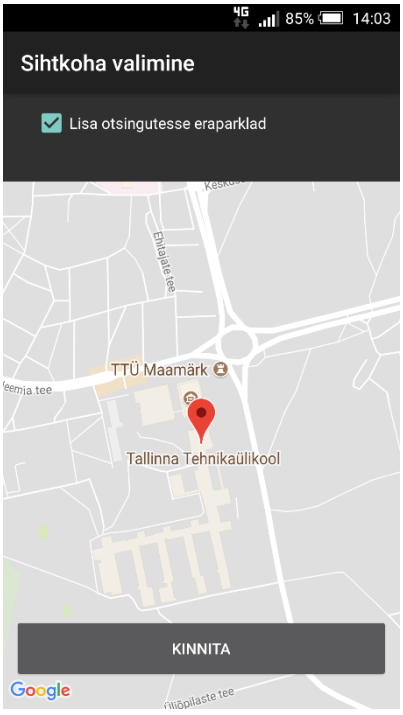
Parkimisrakendus põhineb Google Maps API teenustel, mis võimaldavad kasutada Google Maps kaarti ja sellega seotud funktsioone [6]. Google Maps API lubab parkimisrakendusel lisada kaardil kindlale positsioonile tähiseid (ingl. Markers), jooni (ingl. Polylines) ja piirkondi (ingl. Polygons).

Kasutaja juhib parkimisrakendust kasutajaliidese abil. Esimene vaade kasutajaliideses (vt. joonis 7) tekib parkimisrakenduse avamisel. Teksti 'Vali kaart' all paikneva kasti peale vajutades avaneb saadaval olevate keskuste nimekiri. Alternatiivina saab kasutaja vajutada nupule ’KAARDISTA PARKLA’, mille peale salvestab rakendus kasutaja telefoni GPS koordinaadid. Funktsiooni kasutamiseks peab kasutaja telefonis olema aktiveeritud GPS tugi. Parkimisrakenduse kasutamiseks avaneb ekraanil juhend, kui vajutada teksti ’INFO’ peale.



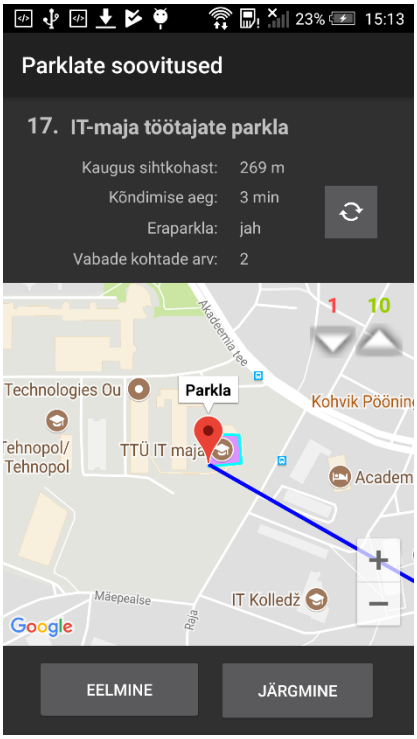
Joonis 7. Kasutajaliidese esimene vaade

Peale keskuse valimist ja 'KINNITA' nuppu vajutamist avaneb kasutajaliideses teine vaade (vt. joonis 8), milleks on sihtkoha valimine. Google Maps kaardil saab vaadet liigutada nõnda, et kaardi keskel olev tähis märgistab kasutaja soovitud sihtkoha. Märkides kasti linnukese on võimalik parkla soovitustesse lisada eraparklad.



Joonis 8. Kasutajaliidese teine vaade

Peale sihtkoha valimist ja 'KINNITA' nuppu vajutamist avaneb kasutajaliideses kolmas vaade, milleks on parkimissoovituste vaatlemine. Kolmas vaade märgib Google Maps kaardil kasutaja sihtkohale lähimad parklad koos lisainfoga. Parklaid kuvatakse ühekaupa. Ekraani all servas asuvate ’EELMINE’ ’JÄRGMINE’ nuppude peale vajutades saab liikuda erinevate parklate vaate vahel. Iga soovitatud parkla puhul saab kasutaja hinnata antud parklat vajutades kaardi servas asuvate noolte peale. Eraparkla soovituse puhul ilmub parkla info kõrvale värskendus nupp, millele vajutades värskendatakse soovitatud parkla vabade parkimiskohtade arv.



Joonis 9. Kasutajaliidese kolmas vaade

Teises ja kolmandas vaates saab igal ajahetkel vajutada Android telefoni tagasinoolt, et minna eelmise vaate juurde.

Serveri veebiteenus kasutab RESTful API-t, mis võimaldab kiirelt teha JSON formaadis andmevahetust kliendi ja serveri vahel [5]. Iga parkla soovituse kuvamisel saadetakse serverile JSON formaadis päring, mis sisaldab soovitatava parkla id-d. Serveri pool võtab päringu vastu ja kontrollib kas serveri andmebaasis on vastava id-ga parklat. Õnnestunud päringu korral saadab server kliendirakendusele tagasi parkla vabade parkimiskohtade arvu, parkla kohtade koguarvu ja hinnangud.

2.4 Funktsionaalsused

Parkimisrakenduse avamisel laetakse kasutajaliidesesse esimene vaade koos selle juurde käiva funktsionaalsusega (activity_first.xml ja FirstActivity.java). Esimese vaate ajal saab kasutaja valida keskuse, mille juurde ta soovib parkida, vaadata rakenduse kasutusjuhendit või kaardistada uus parkla. Kui esimeses vaates valitakse keskus ning vajutatakse nupule 'KINNITA', siis käivitatakse kasutajaliideses teine vaade, millele antakse lisainfona kaasa valitud keskuse nimetus.

Käivitatud teine vaade koos funktsionaalsusega (activity_second.xml ja SecondActivity.java) otsib keskuste registrist keskust, mille nimi on võrdne esimesest vaatest saadetud keskuse nimega ja kirjutab mällu selle keskuse koordinaadid. Seejärel avatakse Google Maps kaart, mille keskpunkt asub valitud keskuse kohal. .Kasutaja saab liigutada kaarti nõnda, et kaardi keskel tähis märgistab kasutaja sihtkohta. Lisaks saab kasutaja panna linnukese kasti ’Lisa otsingutesse eraparklad', mille järel kaasatakse parkla soovituste otsingusse ka tõkkepuuga parklad. Tähistades sihtkoha ja vajutades nupule 'KINNITA' salvestatakse kaardil märgitud sihtkoha koordinaadid ning arvutatakse parklate keskpunkti kaugus sihtkohast. Keskuse parklad sorteeritakse listi, alustades lähimast parklast kõige kaugeima parklani. Käivitatakse kolmas vaade, millele antakse kaasa sorteeritud parklate list, valitud sihtkoha koordinaadid ja tõeväärtus, mis tähistab kas otsing põhineb ainult avalikel parklatel või lisanduvad ka eraparklad.

Seejärel laetakse rakenduse kasutajaliidesesse kolmas vaade koos selle juurde kuuluva funktsionaalsusega (third_activity.xml ja ActivityThird.java). Parklate listist võetakse esimene ParkingLot tüüpi objekt ja tähistatakse kaardil koos kasutaja valitud sihtkoha ja nende kahe punkti vahelise joonega. Parkla kohta kuvatakse ka info: parkla nimetus, parkla kaugus sihtkohast, sirgjooneline kõndimise aeg parklast sihtkohta, parkla tüüp, avaliku parkla puhul parkimiskohtade koguarv või eraparkla puhul vabade parkimiskohtade arv. Iga parkla soovituse puhul päritakse serverilt antud parkla parkimiskohtade koguarv, vabade parkimiskohtade arv ning hinnang (poolthääletuste arv ja vastuhääletuste arv). Vajutades vaate all servas asuvate nuppude ’EELMINE’ ja ’JÄRGMINE’ peal, saab liikuda sihtkohale järgmise kõige lähema parkla soovituse peale või eelmise soovituse peale. Vaadeldes eraparkla soovitust tekib vaatele värskendusnupu ikoon, mille vajutamisel saadetakse serverile uus JSON päring vaadeldava parkla id-ga ja uuendatakse ekraanil asuvad arvud, mis vastavad vastusest tulnud väärtuste tüübile. Kui

kasutaja hindab soovitatud parklat, siis saadetakse serverile JSON formaadis päring hinnatava parkla id koos tõeväärtusega, mis määrab kas hinnati parkla poolt või vastu, vastavalt *true* või *false.*

Eraparklate kaardilugejad ja magnetkontuurid on ühendatud modemiga, mis muundab tõkkepuu avamise puhul kummalgilt seadmelt tuleva info digitaalsignaaliks ja saadab üle Interneti serverile.

Serveris töötab REST API stiilis veebiteenus, mis sisaldab parklate registrit, modemitelt saabuva info ja JSON formaadis kliendipäringute töötlemise funktsionaalsust. Parklate registris on võtmeteks parkla id-d, mille väärtusteks olev list sisaldab antud parkla parkimiskohtade koguarvu, vabade parkimiskohtade arvu ja parkla hinnangut, mis on jagatud poolt- ja vastuhäälteks.

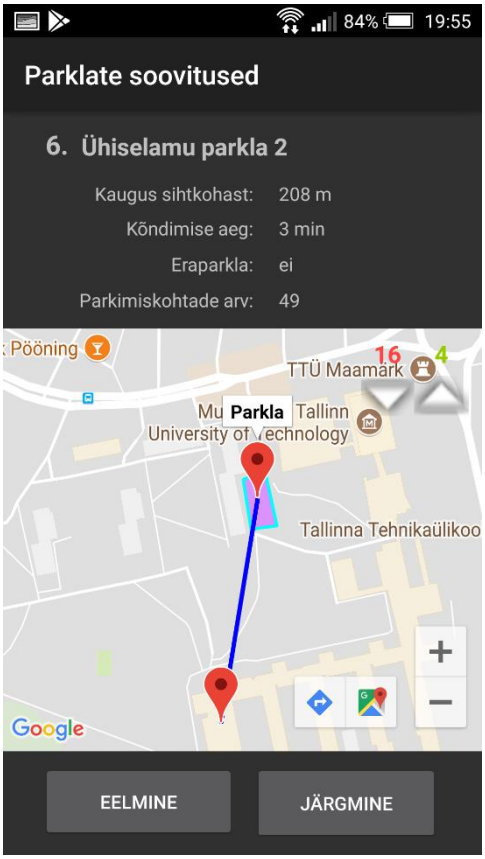
Kui modemilt saabub info serverile eraparkla tõkkepuu kasutuse kohta, siis server liidab või lahutab antud parkla parkimiskohtade koguarvust arvu 1 ja uuendab registris antud parkla vabade parkimiskohtade arvu.

Kui parkimisrakendus saadab serverile päringu ainult parkla id-ga, siis saadab server rakendusele vastuse parklate registris oleva vastava võtme väärtuse, milleks on parkla parkimiskohtade koguarv, vabade parkimiskohtade arv ja poolt- ning vastuhääletuste arvud.

Kui parkimisrakendusest saadetakse serverile päring parkla id ja tõeväärtusega, siis *true* väärtuse puhul liidab server parklate registris vastava id-ga parkla poolthääletustele arvu 1. *False* väärtuse puhul liidab server arvu 1 antud parkla vastuhääletustele.

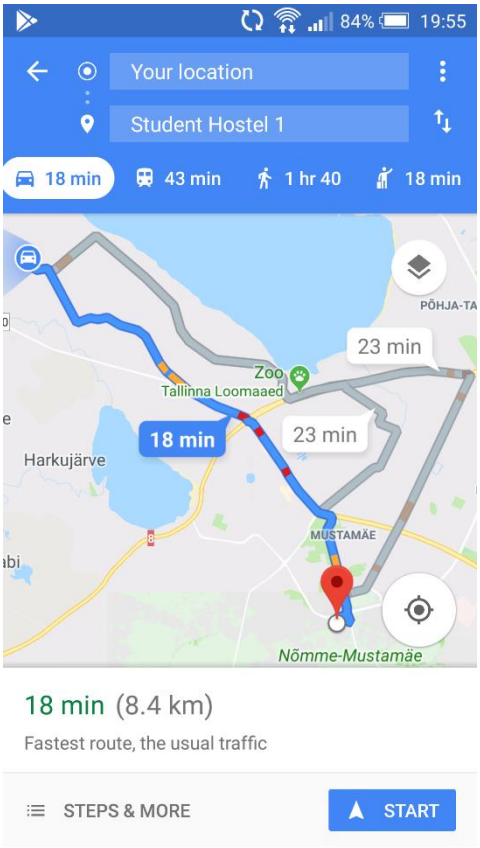
2.5 Lisafunktsionaalsused

Google Maps API võimaldab parkimisrakenduses kaardi tähiste peale vajutades avada Google Maps rakenduses liikumisjuhised kasutaja nutitelefoni asukohast antud tähiseni. Nutitelefoni asukoha määramiseks on tarvis aktiveerida GPS või internetiühendus [7]. Liikumisjuhiste nupp avaneb tähisele vajutades kaardi alumises paremas nurgas kaardi suurendamis- ja vähendamisnuppudest vasakul pool (vt. joonis 10).



Joonis 10. Liikumisjuhiste nupu välimus

Liikumisjuhiste avamisel ja kasutaja nutitelefoni asukoha tuvastamise võimalusel kuvatakse ekraanil võimalikud kiireimad teekonnad nutitelefoni asukohast antud tähiseni koos liikumisaegadega erinevate liikumisvahendite puhul (vt. joonis 11). Pakutavate liikumisvahendite hulgas on autosõit, ühistransport, jalgsi liikumine ja taksoteenus. Ühistranspordi valimisel pakutakse Google Maps rakenduses ühistranspordigraafikud, koos väljumis- ja saabumisaegadega. Taksoteenuse valimisel pakutakse Uber takso tellimis võimalust koos sõidukulu hinnanguga.



Joonis 11. Liikumisjuhised Google Maps rakenduses

3 Realiseerimine

Järgnevates peatükkides kajastatakse Android mobiilirakendustele omaseid programmeerimistehnikaid ja teste rakenduse töökindluse tagamiseks.

3.1 Koodid

Järgnevas programmikoodis on konstruktor, millega luuakse rakenduses parklate objekte. Objekti nimeks on *ParkingLot*, mis sisaldab parkla nimetust, keskust, kuhu alasse parkla kuulub, tõeväärtusi, mis määravad kas parkla on hetkel aktiivne ja parkla tüüpi (*true* kui on eraparkla, *false*, kui on avalik parkla), parkla keskpunkti ja nurkade koordinaadid eraldi väärtustena *double*-tüüpi numbrina, mis lubab salvestada komakohtadega arvu. Autor otsustas parkla koordinaadid salvestada eraldi numbritena, sest *LatLng* objektitüüp, mis sisaldab geograafilise punkti laius- ja pikkuskraadi, ei ole rakenduse vaadete vahel lihtsasti edastatav.

```
public ParkingLot(String parkingLotName, int ParkingLotID, double
parkingLotLatitude, double parkingLotLongtitude,
                  boolean activeLot, boolean privateLot, String
parkingCenter, double cornerLat1, double cornerLon1,
                  double cornerLat2, double cornerLon2, double
cornerLat3, double cornerLon3,
                  double cornerLat4, double cornerLon4) {
    name = parkingLotName;
    id = ParkingLotID;
    latitude = parkingLotLatitude;
    longtitude = parkingLotLongtitude;
    active = activeLot;
    isPrivate = privateLot;
    centre = parkingCenter;
    cornerLatitude1 = cornerLat1;
    cornerLongtitude1 = cornerLon1;
    cornerLatitude2 = cornerLat2;
    cornerLongtitude2 = cornerLon2;
    cornerLatitude3 = cornerLat3;
    cornerLongtitude3 = cornerLon3;
    cornerLatitude4 = cornerLat4;
    cornerLongtitude4 = cornerLon4;
}
```

Joonis 12. ParkingLot objekti konstruktor

Järgnev programmikood kuulub teise vaate funktsionaalsuse alla programmis SecondActivity.java ja kajastab funktsiooni findClosestParkingLots(LatLng destination). Funktsioonile antakse kaasa kasutaja kaardil valitud sihtkoht *LatLng* objektina, mis sisaldab valitud sihtkoha laius- ja pikkuskoordinaate. *LatLng* objekt teisendatakse ümber *Location* objektiks, mis võimaldab arvutada vahemaad erinevate geograafiliste punktide vahel kasutades funktsiooni *distanceTo()*. Parklate register itereeritakse läbi ning leitakse iga parkla keskpunkti kaugus meetrites kasutaja valitud sihtpunktist. Erinevate geograafiliste punktide vahel arvutatud kaugus meetrites arvutatakse komakohaga ja lisatakse *TreeMap*-i võtmeks, mille väärtuseks on *ParkingLot* objekte sisaldav list, juhuks kui mõni parkla asub kasutaja sihtkohast samal kaugusel. Kui kasutaja ei märgi linnukest kasti „Lisa otsingusse eraparkla“, siis lisatakse *TreeMap*-i ainult ainult avalikud parklad. Vastasel juhul lisatakse *TreeMap*-i kõik itereeritud parklad.

```
public TreeMap<Integer, ArrayList<ParkingLot>>
findClosestParkingLots(LatLng destination) {
    Location parkingLotLocation = new Location("parking lot
location");
    Location destinationLocation = new Location("destination");
    destinationLocation.setLatitude(destination.latitude);
    destinationLocation.setLongitude(destination.longitude);
    TreeMap<Integer, ArrayList<ParkingLot>> parkingLotMap = new
TreeMap<>();
    int distance;

    for (ParkingLot parkingLot : parkingLotList) {
        parkingLotLocation.setLatitude(parkingLot.latitude);
        parkingLotLocation.setLongitude(parkingLot.longtitude);
        distance = (int)
destinationLocation.distanceTo(parkingLotLocation);
        ArrayList<ParkingLot> parkingLotListInMap = new ArrayList<>();

        // if the same distance-key exists in parkingLotMap
        if (parkingLotMap.containsKey(distance)) {
            parkingLotListInMap = parkingLotMap.get(distance);
        }

        if (!checkPrivate & !parkingLot.isPrivate) {
            parkingLotListInMap.add(parkingLot);
            parkingLotMap.put(distance, parkingLotListInMap);
        } else if (checkPrivate) {
            parkingLotListInMap.add(parkingLot);
            parkingLotMap.put(distance, parkingLotListInMap);
        }
    }

    return parkingLotMap;
}
```

Joonis 13. Lähimate parklate sorteerimine TreeMap-i SecondActivity.java programmis

Järgnevas programmikoodis on funktsioon, mis saadab parklate sorteeritud nimekirja, kasutaja valitud sihtkoha ja parklate otsingutüübi tõeväärtuse kolmandasse vaatesse. Autor kasutab uue vaate käivitamiseks ja vaadete vahel andmete edastamiseks *Intent*-i. *Intent* on passiivne andmestruktuur, mis kirjeldab rakendusele täitmist vajavaid operatsioone [8]. Andmete edastamine *Intent*-ga toimub võti-väärtus paaride abil.

```
public void setDestination(View view) {
    LatLng destination = marker.getPosition();
    double[] destinationArray = new double[]{destination.latitude,
destination.longitude};
    TreeMap<Integer, ParkingLot> closestParkingLots =
findClosestParkingLots(destination);
    Intent intent = new Intent(this, ThirdActivity.class);
    Bundle bundle = new Bundle();
    bundle.putSerializable("parking lots", closestParkingLots);
    intent.putExtras(bundle);
    intent.putExtra("destination", destinationArray);
    intent.putExtra("show private", checkPrivate);
    startActivity(intent);
}
```

Joonis 14. Andmete saatmine rakenduse teisest vaatest kolmandasse

Järgnev programmikood on kasutusel kolmanda vaate funktsionaalsuses ThirdActivity.java. Kolmanda vaate käivitanud *Intent* objekt kopeeritakse ja sellega kaasnevad andmete väärtused lisatakse muutujatele. Android raamistiku iseärasuste tõttu muutub *TreeMap* objekt vaadete vahel edastusel *HashMap* tüüpi objektiks [9].

Kolmandas vaates muudetakse vastuvõetud *HashMap* uuesti *TreeMap*-ks, et parklate kaugust kirjeldavad võtmeväärtused oleks automaatselt reastatud kasvavas järjekorras.

```
// get data from the intent
Intent intent = getIntent();
Bundle bundle = intent.getExtras();
assert bundle != null;
HashMap<Integer, ArrayList<ParkingLot>> parkingLotHashMap =
(HashMap<Integer, ArrayList<ParkingLot>>)
bundle.getSerializable("parking lots");
closestParkingLots.putAll(parkingLotHashMap);
```

Joonis 15. Andmete vastuvõtt eelmisest vaatest ThirdActivity.java programmis

3.2 Testid

Autor kasutas rakenduse loomiseks *Android Studio* töökeskkonda (IDE), mis pakub lähtekoodi katalooge kahe erinevat tüüpi testide jaoks – lokaalsed ühiktestid ja instrumenteeritud testid. Lokaalsete ühiktestid käivituvad arvuti lokaalse Java Virtuaalmasina peal (JVM) ning selle abil testitakse rakenduse loogikat, mis ei sõltu Android-i raamistikust. Instrumenteeritud testid käivituvad riistvara seadme või Android-i emulaatori peal ning see annab võimaluse testida rakenduse kontekstipõhist loogikat [10].

Kuna rakenduse tegevus sõltub suuresti kasutaja sisendist eelnevates vaadetes ja seda loogikat on keeruline ühiktestide abil kontrollida, siis on autor rakendust testinud valdavalt käsitsi.

Järgnev programmikood sisaldab instrumenteeritud testi loogikat, millega testitakse rakenduse kasutajaliidese toimimist. *Espresso* raamistiku abil on salvestatud kasutajaliidese nuppude vajutusi. Edukalt läbitud instrumenteeritud test kontrollib parklate soovituste, kaardi vaate ja hinnangu- ning värskendusnuppude töökindlust.

```
package com.example.parkingapplication;


import android.support.test.espresso.ViewInteraction;
import android.support.test.rule.ActivityTestRule;
import android.support.test.runner.AndroidJUnit4;
import android.test.suitebuilder.annotation.LargeTest;
import android.view.View;
import android.view.ViewGroup;
import android.view.ViewParent;

import org.hamcrest.Description;
import org.hamcrest.Matcher;
import org.hamcrest.TypeSafeMatcher;
import org.junit.Rule;
import org.junit.Test;
import org.junit.runner.RunWith;

import static android.support.test.espresso.Espresso.onView;
import static android.support.test.espresso.action.ViewActions.click;
import static
android.support.test.espresso.matcher.ViewMatchers.isDisplayed;
import static
android.support.test.espresso.matcher.ViewMatchers.withClassName;
import static
android.support.test.espresso.matcher.ViewMatchers.withContentDescript
ion;
import static
android.support.test.espresso.matcher.ViewMatchers.withId;
import static
```

```
android.support.test.espresso.matcher.ViewMatchers.withText;
import static org.hamcrest.Matchers.allOf;
import static org.hamcrest.Matchers.is;

@LargeTest
@RunWith(AndroidJUnit4.class)
public class InstrumentedTestMap {

    @Rule
    public ActivityTestRule<FirstActivity> mActivityTestRule = new
ActivityTestRule<>(FirstActivity.class);

    @Test
    public void InstrumentedTestMap() {
        ViewInteraction appCompatButton = onView(
                allOf(withId(R.id.confirm), withText("Kinnita"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                0),
                        isDisplayed()));
        appCompatButton.perform(click());

        ViewInteraction appCompatCheckBox = onView(
                allOf(withId(R.id.checkBox), withText("Lisa
otsingutesse eraparklad"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                2),
                        isDisplayed()));
        appCompatCheckBox.perform(click());

        ViewInteraction appCompatButton2 = onView(
                allOf(withId(R.id.button1), withText("Kinnita"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                1),
                        isDisplayed()));
        appCompatButton2.perform(click());

        ViewInteraction imageView = onView(
                allOf(withContentDescription("Zoom in"),
                        childAtPosition(
                                childAtPosition(

withClassName(is("android.widget.RelativeLayout")),
                                        2),
                                0),
                        isDisplayed()));
        imageView.perform(click());

        ViewInteraction imageView2 = onView(
                allOf(withContentDescription("Zoom out"),
                        childAtPosition(
                                childAtPosition(
```

```
withClassName(is("android.widget.RelativeLayout")),
                                        2),
                                1),
                        isDisplayed()));
        imageView2.perform(click());

        ViewInteraction appCompatButton3 = onView(
                allOf(withId(R.id.next), withText("Järgmine"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                1),
                        isDisplayed()));
        appCompatButton3.perform(click());

        ViewInteraction appCompatButton4 = onView(
                allOf(withId(R.id.next), withText("Järgmine"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                1),
                        isDisplayed()));
        appCompatButton4.perform(click());

        ViewInteraction appCompatButton5 = onView(
                allOf(withId(R.id.next), withText("Järgmine"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                1),
                        isDisplayed()));
        appCompatButton5.perform(click());

        ViewInteraction appCompatButton6 = onView(
                allOf(withId(R.id.previous), withText("Eelmine"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                2),
                        isDisplayed()));
        appCompatButton6.perform(click());

        ViewInteraction appCompatButton7 = onView(
                allOf(withId(R.id.previous), withText("Eelmine"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                2),
                        isDisplayed()));
        appCompatButton7.perform(click());

        ViewInteraction appCompatImageButton = onView(
                allOf(withId(R.id.refresh),
withContentDescription("0"),
                        childAtPosition(
                                childAtPosition(
```

```
                                        withId(android.R.id.content),
                                        0),
                                13),
                        isDisplayed()));
        appCompatImageButton.perform(click());

        ViewInteraction appCompatImageButton2 = onView(
                allOf(withId(R.id.refresh),
withContentDescription("0"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                13),
                        isDisplayed()));
        appCompatImageButton2.perform(click());

        ViewInteraction appCompatImageButton3 = onView(
                allOf(withId(R.id.refresh),
withContentDescription("0"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                13),
                        isDisplayed()));
        appCompatImageButton3.perform(click());

        ViewInteraction appCompatImageButton4 = onView(
                allOf(withId(R.id.refresh),
withContentDescription("0"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                13),
                        isDisplayed()));
        appCompatImageButton4.perform(click());

        ViewInteraction appCompatImageView = onView(
                allOf(withId(R.id.dislikeButton),
withContentDescription("0"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                14),
                        isDisplayed()));
        appCompatImageView.perform(click());

        ViewInteraction appCompatImageView2 = onView(
                allOf(withId(R.id.dislikeButton),
withContentDescription("0"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                14),
                        isDisplayed()));
        appCompatImageView2.perform(click());
```

```
        ViewInteraction appCompatButton8 = onView(
                allOf(withId(R.id.previous), withText("Eelmine"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                2),
                        isDisplayed()));
        appCompatButton8.perform(click());

        ViewInteraction appCompatButton9 = onView(
                allOf(withId(R.id.next), withText("Järgmine"),
                        childAtPosition(
                                childAtPosition(
                                        withId(android.R.id.content),
                                        0),
                                1),
                        isDisplayed()));
        appCompatButton9.perform(click());

    }

    private static Matcher<View> childAtPosition(
            final Matcher<View> parentMatcher, final int position) {

        return new TypeSafeMatcher<View>() {
            @Override
            public void describeTo(Description description) {
                description.appendText("Child at position " + position
+ " in parent ");
                parentMatcher.describeTo(description);
            }

            @Override
            public boolean matchesSafely(View view) {
                ViewParent parent = view.getParent();
                return parent instanceof ViewGroup &&
parentMatcher.matches(parent)
                        && view.equals(((ViewGroup)
parent).getChildAt(position));
            }
        };
    }
}
```

Joonis 16. Instrumenteeritud test

Kokkuvõte

Lõputöö eesmärgiks oli uurida TTÜ üliõpilaslinnaku parkimisprobleemi lahendamise võimalusi, et aidata autojuhtidel lühikese ajaga ja võimalikult mugavalt leida sihtpunktile lähim parkimiskoht ja vähendada sellega parklates liigset liikluskoormust.

Käesoleva töö käigus analüüsis autor Eestis kasutuselolevaid parkimislahendusi, pakkus oma lahenduse parkimisolukorra parandamiseks ning arendas tehtud analüüsi põhjal üliõpilaslinnaku jaoks sobiva parkimisrakenduse prototüübi.

Loodud prototüübi testimisel kasutas autor käsitsi testimist ja Espresso abil salvestatud instrumenteeritud teste. Lõputöös kavandatud ja teostatud lahendused on kasutatavad väikeste modifikatsioonidega ka suuremate tehniliste võimalustega linnaparklates pakkudes võrreldes olemasolevate lahendustega mitmeid uuendusi.

Kasutatud kirjandus

[1] „Sõiduautode arv 1000 elaniku kohta | Eesti Statistika,“ [Võrgumaterjal]. Available: stat.ee/34300. [Kasutatud 16 04 2018].

[2] „Barking - Pane parkimiskohad teenima,“ [Võrgumaterjal]. Available: https://barking.ee/pane-parkimiskohad-teenima/. [Kasutatud 04 05 2018].

[3] M. Leivo, Interviewee, [Intervjuu]. 12 10 2017.

[4] A. Kallaste, Tallinn, 2018.

[5] „Soap vs. REST Comparison: Differences in Performance, APIs & More,“ [Võrgumaterjal]. Available: https://stackify.com/soap-vs-rest/. [Kasutatud 05 05 2018].

[6] „Overview | Maps SDK for Android | Google Developers,“ [Võrgumaterjal]. Available: https://developers.google.com/maps/documentation/android-sdk/intro. [Kasutatud 07 05 2018].

[7] „geolocation - How does Google Maps estimate my location without GPS? - Android Enthusiasts Stack Exchange,“ [Võrgumaterjal]. Available: https://android.stackexchange.com/questions/15009/how-does-google-maps-estimate-my-location-without-gps/15010#15010. [Kasutatud 11 05 2018].

[8] „Intent | Android Developers,“ [Võrgumaterjal]. Available: https://developer.android.com/reference/android/content/Intent. [Kasutatud 10 05 2018].

[9] „The mysterious case of the Bundle and the Map - The WTF files - Medium,“ [Võrgumaterjal]. Available: https://medium.com/the-wtf-files/the-mysterious-case-of-the-bundle-and-the-map-7b15279a794e. [Kasutatud 25 04 2018].

[10] „Test Your App | Android Developers,“ [Võrgumaterjal]. Available: https://developer.android.com/studio/test/. [Kasutatud 13 05 2018].

Lisa 1 – Android Manifest File

```
<?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="com.example.parkingapplication">

    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />

    <application
        android:allowBackup="false"
        android:icon="@mipmap/ic_launcher_car"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat"
        tools:ignore="GoogleAppIndexingWarning">
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />

        <!-- The API key for Google Maps-based APIs. -->
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />

        <activity
            android:name=".FirstActivity"
            android:label="@string/app_name"
            android:screenOrientation="portrait">
            <intent-filter android:label="@string/app_name">
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".SecondActivity"
            android:label="Sihtkoha valimine"
            android:screenOrientation="portrait" />
        <activity
            android:name=".ThirdActivity"
            android:label="Parklate soovitused"
            android:screenOrientation="portrait" />
    </application>

</manifest>
```

Lisa 2 – ParkingLot.java

```
package com.example.parkingapplication;

import java.io.Serializable;

public class ParkingLot implements Serializable {

    String name, centre;
    int id;
    boolean active, isPrivate;
    double latitude;
    double longtitude;
    double cornerLatitude1;
    double cornerLongtitude1;
    double cornerLatitude2;
    double cornerLongtitude2;
    double cornerLatitude3;
    double cornerLongtitude3;
    double cornerLatitude4;
    double cornerLongtitude4;


    public ParkingLot(String parkingLotName, int ParkingLotID, double
parkingLotLatitude, double parkingLotLongtitude,
                      boolean activeLot, boolean privateLot, String
parkingCenter, double cornerLat1, double cornerLon1,
                      double cornerLat2, double cornerLon2, double
cornerLat3, double cornerLon3,
                      double cornerLat4, double cornerLon4) {
        name = parkingLotName;
        id = ParkingLotID;
        latitude = parkingLotLatitude;
        longtitude = parkingLotLongtitude;
        active = activeLot;
        isPrivate = privateLot;
        centre = parkingCenter;
        cornerLatitude1 = cornerLat1;
        cornerLongtitude1 = cornerLon1;
        cornerLatitude2 = cornerLat2;
        cornerLongtitude2 = cornerLon2;
        cornerLatitude3 = cornerLat3;
        cornerLongtitude3 = cornerLon3;
        cornerLatitude4 = cornerLat4;
        cornerLongtitude4 = cornerLon4;
    }
}
```

Lisa 3 – FirstActivity.java

```
package com.example.parkingapplication;

import android.content.Intent;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;


public class FirstActivity extends AppCompatActivity implements
AdapterView.OnItemSelectedListener {

    private String[] centersArray = {"TTÜ","Rotermanni"};
    private String selectedParkingCenter;

    /**
     *
https://www.tutorialspoint.com/android/android_spinner_control.htm
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_first);
        setTitle("Keskuse valimine");

        // get spinner
        Spinner spin = (Spinner) findViewById(R.id.spinner);
        spin.setOnItemSelectedListener(this);
        ArrayAdapter aa = new ArrayAdapter(this,
android.R.layout.simple_spinner_item, centersArray);

aa.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_it
em);
        spin.setAdapter(aa);
    }

    /**
     * https://stackoverflow.com/questions/6264694/how-to-add-message-
box-with-ok-button
     */
    public void toggleInfo(View view){
        AlertDialog.Builder dlgAlert  = new AlertDialog.Builder(this);
        dlgAlert.setMessage("Kasutate rakendust, mis soovitab Teie
sihtkohale lähimaid parklaid.\n\n" +
                "1. Samm\n\nValige keskus, mille ümber plaanite
parkida ning vajutage 'KINNITA' nupule.\nHetkel sisaldab rakenduse
andmebaas ainult TTÜ parklaid.\n\n'KAARDISTA PARKLA' funktsionaalsus
on hetkel realiseerimata." +
                "\n\n2. Samm\n\nLiigutage kaarti nõnda, et kaardi
keskel asuv tähis paikneb Teie sihtkoha peal. Kui soovite näha
eraparklaid, siis pange linnuke vastavasse kasti. Seejärel vajutage
'KINNITA' nupule." +
                "\n\n3. Samm\n\nRakendus kuvab Teile parkla soovitusi
```

```
koos infoga. \nVajutades nuppudele 'JÄRGMINE' või 'EELMINE', saate
näha teiste parklate asukohta.\nVajutades nupule 'VÄRSKENDA',
värskendatakse eraparklate vabade kohtade arv ning hinnangud." +
                "\n\nVajutades kaardil asuva tähise peale ja seejärel
kaardi all paremas nurgas avanenud ikoonide seast sinise noole peale,
saab Google Maps rakenduses avada liikumisjuhised Teie telefoni
asukohast antud tähiseni." +
                "\n\nIgal ajahetkel saab rakenduses eelmise sammu
juurde liikuda vajutades oma telefonil 'tagasi' nuppu.");
        dlgAlert.setTitle("Info");
        dlgAlert.setPositiveButton("OK", null);
        dlgAlert.setCancelable(true);
        dlgAlert.create().show();
    }

    @Override
    public void onItemSelected(AdapterView<?> arg0, View arg1, int
position, long id) {
        selectedParkingCenter = centersArray[position];
    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {

    }

    public void openMap(View view) {
        Intent intent = new Intent(this, SecondActivity.class);
        if (selectedParkingCenter.equals("TTÜ"))  {
            intent.putExtra("selectedParkingCenter",
selectedParkingCenter);
            startActivity(intent);
        } else Toast.makeText(FirstActivity.this, "Tuleb varsti!",
Toast.LENGTH_SHORT).show();
    }

    public void mapANewParkingLot(View view) {
        Toast.makeText(FirstActivity.this, "Tuleb varsti!",
Toast.LENGTH_SHORT).show();
    }
}
```

Lisa 4 – SecondActivity.java

```
package com.example.parkingapplication;

import android.content.Intent;
import android.location.Location;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.CheckBox;
import android.widget.Toast;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

import java.util.ArrayList;
import java.util.TreeMap;

/**
 * An activity that displays a Google map with a marker (pin) to
indicate a particular location.
 */
public class SecondActivity extends AppCompatActivity
        implements OnMapReadyCallback {

    private ArrayList<ParkingLot> parkingLotList = new ArrayList<>();
    private Marker destinationMarker;
    private CheckBox checkBox;
    private String parkingCenter;
    private boolean checkPrivate;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // retrieve the content view that renders the map.
        setContentView(R.layout.activity_second);
        // get the SupportMapFragment and request notification when
the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);


        // create public parking lot objects
        ParkingLot ttü1 = new ParkingLot("Ühiselamu parkla 1",1,
59.395843, 24.666550, true, false, "TTÜ", 59.395513, 24.666284,
59.395569, 24.666879,59.396196, 24.666756, 59.396138, 24.666187);
        ParkingLot ttü2 = new ParkingLot("TIPI parkla",2, 59.395832,
24.672459, true, false, "TTÜ", 59.395242, 24.672279, 59.395291,
24.672979,59.396480, 24.672507, 59.396391, 24.671756);
        ParkingLot ttü3 = new ParkingLot("IT ja Loodusteaduste maja
parkla",3, 59.397377, 24.662829, true, false, "TTÜ" , 59.397198,
24.662236, 59.397200, 24.663405, 59.397526, 24.663370, 59.397524,
```

```
24.662224);
        ParkingLot ttü4 = new ParkingLot("IT maja parkla",4,
59.396671, 24.662430, true, false, "TTÜ",59.396731,
24.662220,59.396649, 24.662233,59.396658, 24.662756,59.396729,
24.662764 );
        ParkingLot ttü5 = new ParkingLot("Küberneetikainstituudi
parkla",5, 59.397917, 24.661991, true, false, "TTÜ",59.398169,
24.661371, 59.397629, 24.661360, 59.397700, 24.662392, 59.398182,
24.662341);
        ParkingLot ttü6 = new ParkingLot("Rauakooli parkla",6,
59.394389, 24.672394, true, false, "TTÜ",59.394778, 24.671978,
59.393949, 24.672321, 59.394011, 24.672833, 59.394832, 24.672551);
        ParkingLot ttü7 = new ParkingLot("Raamatukogu parkla",7,
59.396855, 24.671588, true, false, "TTÜ",59.397045, 24.671344,
59.396596, 24.671539, 59.396625, 24.671831,59.397075, 24.671675 );
        ParkingLot ttü8 = new ParkingLot("Majandusmaja parkla",8,
59.397029, 24.669980, true, false, "TTÜ", 59.397024, 24.669207,
59.396855, 24.669276, 59.397074, 24.671218, 59.397248, 24.671132);
        ParkingLot ttü9 = new ParkingLot("Ühiselamu parkla 2",9,
59.396135, 24.669229, true, false, "TTÜ", 59.396289, 24.668995,
59.395882, 24.669121, 59.395910, 24.669539, 59.396373, 24.669340);
        ParkingLot ttü10 = new ParkingLot("Majandusmaja parkla",10,
59.396519, 24.669463, true, false, "TTÜ", 59.396683, 24.669294,
59.396285, 24.669479, 59.396296, 24.669629, 59.396693, 24.669516);
        ParkingLot ttü11 = new ParkingLot("Ühiselamu/Hostel
parklad",11, 59.396902, 24.663952, true, false, "TTÜ", 59.397187,
24.664161, 59.396699, 24.663624,59.396650, 24.663847 , 59.397168,
24.664362);
        ParkingLot ttü12 = new ParkingLot("Ühiselamu/Hostel
parklad",12, 59.397022, 24.664593, true, false, "TTÜ", 59.397015,
24.664255 , 59.396763, 24.665356, 59.396868, 24.665616, 59.397168,
24.664385);
        ParkingLot ttü13 = new ParkingLot("IT Kolledži parkla",13,
59.395028, 24.662891, true, false, "TTÜ", 59.395240, 24.662253,
59.394807, 24.662226, 59.394988, 24.663873, 59.395101, 24.663983);
        ParkingLot ttü14 = new ParkingLot("Ühiselamu parkla 3",14,
59.396344, 24.665620, true, false, "TTÜ", 59.396229, 24.665226,
59.396273, 24.665883, 59.396492, 24.665862, 59.396482, 24.665578);
        ParkingLot ttü15 = new ParkingLot("Ühiselamu parkla 4",15,
59.396214, 24.664060, true, false, "TTÜ", 59.396248, 24.663204,
59.395966, 24.664593, 59.396177, 24.664706,59.396416, 24.663432 );
        ParkingLot ttü16 = new ParkingLot("Ühiselamu parkla 5",16,
59.396750, 24.667174, true, false, "TTÜ", 59.396595, 24.666246,
59.396774, 24.668150 , 59.396917, 24.668136, 59.396792, 24.666215);
        ParkingLot ttü17 = new ParkingLot("U06 parkla",17, 59.395204,
24.668355, false, false, "TTÜ", 59.395402, 24.668033 , 59.394808,
24.668371, 59.394836, 24.668720, 59.395465, 24.668312);
        ParkingLot ttü18 = new ParkingLot("Üliõpilaste tee parkla",18,
59.392642, 24.672974, true, false, "TTÜ", 59.392733, 24.673816 ,
59.392475, 24.671187, 59.392383, 24.671643, 59.392674, 24.673823);
        ParkingLot ttü19 = new ParkingLot("Tekstiilimaja parkla",19,
59.392175, 24.679284, true, false, "TTÜ", 59.392309, 24.679026,
59.392026, 24.678983, 59.392026, 24.679761, 59.392338, 24.679729);
        ParkingLot ttü20 = new ParkingLot("Spordihoone parkla 1",20,
59.393628, 24.679493, true, false, "TTÜ", 59.393386, 24.679386,
59.393040, 24.679692,59.393878, 24.679681 , 59.393887, 24.679182);
        ParkingLot ttü21 = new ParkingLot("Spordihoone parkla 2",21,
59.393929, 24.678120, true, false, "TTÜ", 59.393878, 24.677299,
59.393682, 24.677353, 59.393882, 24.678994, 59.394069, 24.678881);

        // create private parking lot objects
```

```
        ParkingLot ttü22 = new ParkingLot("IT-maja töötajate
parkla",22, 59.396765, 24.662349, true, true, "TTÜ", 59.396743,
24.662354, 59.396775, 24.662968, 59.397045, 24.662925 , 59.397052,
24.662354);
        ParkingLot ttü23 = new ParkingLot("Ühiselamu parkla maa-
all",23, 59.395846, 24.666570, true, true, "TTÜ", 59.395563,
24.666221, 59.395628, 24.666849, 59.396203, 24.666876, 59.396155,
24.666007);
        ParkingLot ttü24 = new ParkingLot("Tudengimaja parkla",24,
59.395742, 24.670814, true, true, "TTÜ", 59.395829, 24.671189,
59.395751, 24.670344, 59.395616, 24.670392, 59.395710, 24.671234);
        ParkingLot ttü25 = new ParkingLot("U05 ja U06 vaheline
parkla",25, 59.394192, 24.669247, true, true, "TTÜ", 59.394417,
24.668862, 59.393896, 24.669060, 59.393922, 24.669489, 59.394480,
24.669306);
        ParkingLot ttü26 = new ParkingLot("U04 ja U05 vaheline
parkla",26, 59.394261, 24.669923, true, true, "TTÜ", 59.393975,
24.669816, 59.394044, 24.670213, 59.394580, 24.670052, 59.394524,
24.669601);
        ParkingLot ttü27 = new ParkingLot("U03 ja U04 vaheline
parkla",27, 59.394293, 24.670907, true, true, "TTÜ", 59.393860,
24.670995, 59.393884, 24.671142, 59.394631, 24.670841, 59.394629,
24.670696);
        ParkingLot ttü28 = new ParkingLot("U02 ja U03 parkla",28,
59.394338, 24.671527, true, true, "TTÜ", 59.394681,
24.671194,59.393915, 24.671516 , 59.393957, 24.671913, 59.394761,
24.671570);
        ParkingLot ttü29 = new ParkingLot("Mektory parkla",29,
59.394370, 24.661692, true, true, "TTÜ", 59.394379, 24.661190,
59.394238, 24.661265, 59.394329, 24.662037, 59.394476, 24.662006);
        ParkingLot ttü30 = new ParkingLot("Majandusmaja parkla maa-
all",30, 59.396663, 24.670036, true, true, "TTÜ", 59.396533,
24.670540, 59.396471, 24.669824, 59.396796, 24.669625, 59.396868,
24.670397);
        ParkingLot ttü31 = new ParkingLot("Raamatukogu parkla maa-
all",31, 59.396631, 24.671084, true, true, "TTÜ", 59.396550,
24.670889, 59.396572, 24.671141, 59.396737, 24.671122, 59.396724,
24.670808);
        ParkingLot ttü32 = new ParkingLot("Raamatukogu eraparkla",32,
59.396436, 24.671175, true, true, "TTÜ", 59.396508, 24.671470,
59.396460, 24.671011, 59.396408, 24.671032, 59.396440, 24.671509);

        // additional public parking lot objects
        ParkingLot ttü33 = new ParkingLot("Puidumaja parkla",33,
59.397357, 24.656565, true, false, "TTÜ", 59.397515, 24.656605,
59.397227, 24.656401, 59.397198, 24.656688, 59.397487, 24.656827);
        ParkingLot ttü34 = new ParkingLot("Laborihoone parkla",34,
59.394990, 24.659537, true, false, "TTÜ", 59.394919, 24.659664,
59.394905, 24.659558, 59.395040, 24.659425, 59.395068, 24.659504);

        // add parking lots to presorting list
        ArrayList<ParkingLot> preSortList = new ArrayList<>();
        preSortList.add(ttü1);
        preSortList.add(ttü2);
        preSortList.add(ttü3);
        preSortList.add(ttü4);
        preSortList.add(ttü5);
        preSortList.add(ttü6);
        preSortList.add(ttü7);
        preSortList.add(ttü8);
        preSortList.add(ttü9);
```

```
        preSortList.add(ttü10);
        preSortList.add(ttü11);
        preSortList.add(ttü12);
        preSortList.add(ttü13);
        preSortList.add(ttü14);
        preSortList.add(ttü15);
        preSortList.add(ttü16);
        preSortList.add(ttü17);
        preSortList.add(ttü18);
        preSortList.add(ttü19);
        preSortList.add(ttü20);
        preSortList.add(ttü21);
        preSortList.add(ttü22);
        preSortList.add(ttü23);
        preSortList.add(ttü24);
        preSortList.add(ttü25);
        preSortList.add(ttü26);
        preSortList.add(ttü27);
        preSortList.add(ttü28);
        preSortList.add(ttü29);
        preSortList.add(ttü30);
        preSortList.add(ttü31);
        preSortList.add(ttü32);
        preSortList.add(ttü33);
        preSortList.add(ttü34);

        // remove inactive parking lots
        for (ParkingLot parkingLot : preSortList) {
            if (parkingLot.active) {
                parkingLotList.add(parkingLot);
            }
        }

        // get selected parking center from intent
        parkingCenter =
getIntent().getStringExtra("selectedParkingCenter");

        // get checkBox
        checkBox = (CheckBox) findViewById(R.id.checkBox);
        addListenerOnCheckBox();
    }

    /**
     * https://www.mkyong.com/android/android-checkbox-example/
     */
    public void addListenerOnCheckBox(){
        checkBox.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if (checkBox.isChecked()) {
                    checkPrivate = true;
                } else {
                    checkPrivate = false;
                }
            }
        });
    }

    /**
     * Manipulates the map when it's available.
     * The API invokes this callback when the map is ready to be used.
```

```
     * This is where we can add markers or lines, add listeners or
move the camera. In this case,
     * we just add a marker near Sydney, Australia.
     * If Google Play services is not installed on the device, the
user receives a prompt to install
     * Play services inside the SupportMapFragment. The API invokes
this method after the user has
     * installed Google Play services and returned to the app.
     *
     * https://developers.google.com/maps/documentation/android-
sdk/map-with-marker
     */
    @Override
    public void onMapReady(final GoogleMap googleMap) {
        double TTÜcenterLatitude = 59.395573;
        double TTÜcenterLongtitude = 24.666472;
        LatLng currentCenter = new LatLng(0, 0);

        // assign map center coordinates
        if (parkingCenter.equals("TTÜ")) {
            currentCenter = new LatLng(TTÜcenterLatitude,
TTÜcenterLongtitude);
        } else {
            Intent intent = new Intent(this, FirstActivity.class);
            startActivity(intent);
        }

        // add a marker in map's center and move the camera to the
same location
        destinationMarker = googleMap.addMarker(new
MarkerOptions().position(currentCenter));
        float zoomLevel = 15.0f;

googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(currentCenter,
zoomLevel));
        googleMap.setOnCameraMoveListener(new
GoogleMap.OnCameraMoveListener() {
            @Override
            public void onCameraMove() {
                // center marker while map is moved

destinationMarker.setPosition(googleMap.getCameraPosition().target);
            }
        });
    }

    public void setDestination(View view) {
        LatLng destination = destinationMarker.getPosition();
        double[] destinationArray = new double[]{destination.latitude,
destination.longitude};
        TreeMap<Integer, ArrayList<ParkingLot>> closestParkingLots =
findClosestParkingLots(destination);
        Intent intent = new Intent(this, ThirdActivity.class);
        Bundle bundle = new Bundle();
        bundle.putSerializable("parking lots", closestParkingLots);
        intent.putExtras(bundle);
        intent.putExtra("destination", destinationArray);
        intent.putExtra("show private", checkPrivate);
        startActivity(intent);
    }
```

```
    public TreeMap<Integer, ArrayList<ParkingLot>>
findClosestParkingLots(LatLng destination) {
        Location parkingLotLocation = new Location("parking lot
location");
        Location destinationLocation = new Location("destination");
        destinationLocation.setLatitude(destination.latitude);
        destinationLocation.setLongitude(destination.longitude);
        TreeMap<Integer, ArrayList<ParkingLot>> parkingLotMap = new
TreeMap<>();
        int distance;

        for (ParkingLot parkingLot : parkingLotList) {
            parkingLotLocation.setLatitude(parkingLot.latitude);
            parkingLotLocation.setLongitude(parkingLot.longtitude);
            distance = (int)
destinationLocation.distanceTo(parkingLotLocation);
            ArrayList<ParkingLot> parkingLotListInMap = new
ArrayList<>();

            // if the same distance-key exists in parkingLotMap
            if (parkingLotMap.containsKey(distance)) {
                parkingLotListInMap = parkingLotMap.get(distance);
            }

            if (!checkPrivate & !parkingLot.isPrivate) {
                parkingLotListInMap.add(parkingLot);
                parkingLotMap.put(distance, parkingLotListInMap);
            } else if (checkPrivate) {
                parkingLotListInMap.add(parkingLot);
                parkingLotMap.put(distance, parkingLotListInMap);
            }
        }

        return parkingLotMap;
    }
}
```

Lisa 5 – ThirdActivity.java

```
package com.example.parkingapplication;

import android.content.Intent;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;
import android.widget.RelativeLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.UiSettings;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.maps.model.Polygon;
import com.google.android.gms.maps.model.PolygonOptions;
import com.google.android.gms.maps.model.Polyline;
import com.google.android.gms.maps.model.PolylineOptions;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.TreeMap;

public class ThirdActivity extends AppCompatActivity implements
OnMapReadyCallback {

    private TreeMap<Integer, ArrayList<ParkingLot>> closestParkingLots
= new TreeMap<>();
    private ArrayList<Integer> distances = new ArrayList<>();
    private LatLng temporaryLatLng;
    private LatLng destinationLatLng;
    private Marker parkingLotMarker = null;
    private GoogleMap map;
    private float zoomLevel = 16f; // this goes up to 21
    private int currentLotNumber;
    private int currentList;
    private int currentLotInList;
    private ParkingLot currentParkingLot;
    private int likeNr;
    private int dislikeNr;
    private TextView distance;
    private TextView timeToWalk;
    private TextView freeSpotsText;
    private TextView freeSpotsValue;
    private TextView totalParkingSpotsText;
    private TextView totalParkingSpotsValue;
    private TextView description;
    private TextView queueNumber;
    private TextView privateLot;
```

```
    private TextView likes;
    private TextView dislikes;
    private Polyline line;
    private Polygon polygon;
    private ImageButton refresh;
    private MockWebService mockService;
    private RelativeLayout.LayoutParams layoutParams;
    private boolean[] likeGiven;
    private int pixels;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // retrieve the content view that renders the map.
        setContentView(R.layout.activity_third);
        // get the SupportMapFragment and request notification when
the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

        // get data from the intent
        Intent intent = getIntent();
        Bundle bundle = intent.getExtras();
        assert bundle != null;
        HashMap<Integer, ArrayList<ParkingLot>> parkingLotHashMap =
(HashMap<Integer, ArrayList<ParkingLot>>)
bundle.getSerializable("parking lots");
        closestParkingLots.putAll(parkingLotHashMap);
        distances = new ArrayList<>(closestParkingLots.keySet());

        // get user destination
        double destinationArray[] =
intent.getDoubleArrayExtra("destination");
        destinationLatLng = new LatLng(destinationArray[0],
destinationArray[1]);

        // get elements on view
        distance = findViewById(R.id.distanceValue);
        timeToWalk = findViewById(R.id.timeToWalkValue);
        freeSpotsValue = findViewById(R.id.freeSpotsValue);
        freeSpotsText = findViewById(R.id.freeSpotsText);
        queueNumber = findViewById(R.id.queueNumber);
        description = findViewById(R.id.parkingLotName);
        privateLot = findViewById(R.id.parkingLotTypeValue);
        likes = findViewById(R.id.likes);
        dislikes = findViewById(R.id.dislikes);
        totalParkingSpotsText =
findViewById(R.id.totalParkingSpotsText);
        totalParkingSpotsValue =
findViewById(R.id.totalParkingSpotsValue);
        refresh = findViewById(R.id.refresh);
        refresh.setVisibility(View.INVISIBLE);

        // convert dp into pixels needed to move freeSpotsText element
        // https://android--code.blogspot.com.ee/2015/05/android-
textview-layout-margin.html
        // https://stackoverflow.com/questions/5255184/android-and-
setting-width-and-height-programmatically-in-dp-units
        layoutParams = (RelativeLayout.LayoutParams)
```

```
freeSpotsText.getLayoutParams();
        float scale =
getApplicationContext().getResources().getDisplayMetrics().density;
        pixels = (int) (132 * scale + 0.5f);

        // start mock Web Service
        mockService = new MockWebService();
        mockService.initialize();

        // create scoring history
        likeGiven = new boolean[distances.size()];
    }

    /**
     * Manipulates the map when it's available.
     * The API invokes this callback when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or
move the camera. In this case,
     * we just add a marker near Sydney, Australia.
     * If Google Play services is not installed on the device, the
user receives a prompt to install
     * Play services inside the SupportMapFragment. The API invokes
this method after the user has
     * installed Google Play services and returned to the app.
     */
    public void onMapReady(final GoogleMap googleMap) {
        // add a marker and move the map's camera to the same
location.
        map = googleMap;
        map.addMarker(new
MarkerOptions().position(destinationLatLng).title("Sihtkoht")).showInf
oWindow();
        UiSettings ui = map.getUiSettings();
        ui.setZoomControlsEnabled(true);

        // show first parking lot recommendation
        currentLotNumber = 1;
        currentList = 0;
        currentLotInList = 0;
        currentParkingLot =
closestParkingLots.get(distances.get(currentList)).get(currentLotInLis
t);
        temporaryLatLng = createNewLatLng();
        if (currentParkingLot.isPrivate) infoForPrivateLot();
        else infoForPublicLot();

        // create a parking lot marker, polyline & move camera to
marker
        parkingLotMarker = map.addMarker(new
MarkerOptions().position(temporaryLatLng).title("Parkla"));
        parkingLotMarker.showInfoWindow();
        line = map.addPolyline(new
PolylineOptions().add(temporaryLatLng,
destinationLatLng).color(Color.BLUE).geodesic(true));

map.moveCamera(CameraUpdateFactory.newLatLngZoom(temporaryLatLng,
zoomLevel));
        getStats();
        updateParkingSpots();
    }
```

```
    public boolean multipleParkingLotsOnSameDistance() {
        return
closestParkingLots.get(distances.get(currentList)).size() > 1;
    }

    public boolean nextParkingLotExistsInList() {
        return currentLotInList <
closestParkingLots.get(distances.get(currentList)).size() - 1;
    }

    public boolean previousParkingLotExistsInList() {
        return currentLotInList > 0;
    }

    public LatLng createNewLatLng() {
        return new LatLng(currentParkingLot.latitude,
currentParkingLot.longtitude);
    }

    public void getNextLot(View view) {
        if (currentList < closestParkingLots.size() - 1) {
            if (multipleParkingLotsOnSameDistance() &
nextParkingLotExistsInList()) currentLotInList += 1;
            else {
                currentList += 1;
                currentLotInList = 0;
            }

            currentLotNumber += 1;
            currentParkingLot =
closestParkingLots.get(distances.get(currentList)).get(currentLotInLis
t);
            temporaryLatLng = createNewLatLng();

            if (currentParkingLot.isPrivate) infoForPrivateLot();
            else infoForPublicLot();
            updateMarkerAndLine();
            updateParkingSpots();
            getStats();
        }
    }

    public void getPreviousLot(View view) {
        if (currentLotNumber > 1) {
           if (currentLotInList == 0) currentList -= 1;

            if (multipleParkingLotsOnSameDistance()) {
                if (currentLotInList == 0) currentLotInList =
closestParkingLots.get(distances.get(currentList)).size() - 1;
                else if (previousParkingLotExistsInList()) {
                    currentLotInList -= 1;
                }
            } else {
                currentLotInList = 0;
            }

            currentLotNumber -= 1;
            currentParkingLot =
closestParkingLots.get(distances.get(currentList)).get(currentLotInLis
t);
            temporaryLatLng = createNewLatLng();
```

```
            if (currentParkingLot.isPrivate) infoForPrivateLot();
            else infoForPublicLot();
            updateMarkerAndLine();
            getStats();
            updateParkingSpots();
        }
    }

    public void getStats() {
        // update distance
        int kaugusInt = distances.get(currentList);
        String kaugusString = String.valueOf(kaugusInt) + " m";
        distance.setText(kaugusString);

        // update approximate walking time
        BigDecimal kaugusDecimal = new BigDecimal(kaugusInt);
        BigDecimal timeToWalkDecimal = kaugusDecimal.divide(new
BigDecimal(1.38889), 0, RoundingMode.HALF_UP);
        String timeToWalkString;
        int timeToWalkInt = timeToWalkDecimal.intValue();
        int timeToWalkMinutes;
        int timeToWalkSeconds;

        // if time is longer than an hour
        if (timeToWalkInt >= 3600) timeToWalkString = "> 1 h";
        else if (timeToWalkInt >= 60) {
            // else if time is over a minute
            timeToWalkMinutes = timeToWalkInt / 60;
            timeToWalkSeconds = timeToWalkInt % 60;
            if (timeToWalkSeconds >= 30) timeToWalkMinutes += 1;
            timeToWalkString = String.valueOf(timeToWalkMinutes) + "
min";
        }
        // if time is less than a minute
        else timeToWalkString = "< 1 min";

        this.timeToWalk.setText(timeToWalkString);

        // update parking lot number
        String queueNumberText = String.valueOf(currentLotNumber) +
".";
        queueNumber.setText(queueNumberText);

        // update description
        description.setText(currentParkingLot.name);

        // add polygon
        LatLng corner1 = new LatLng(currentParkingLot.cornerLatitude1,
currentParkingLot.cornerLongtitude1);
        LatLng corner2 = new LatLng(currentParkingLot.cornerLatitude2,
currentParkingLot.cornerLongtitude2);
        LatLng corner3 = new LatLng(currentParkingLot.cornerLatitude3,
currentParkingLot.cornerLongtitude3);
        LatLng corner4 = new LatLng(currentParkingLot.cornerLatitude4,
currentParkingLot.cornerLongtitude4);

        polygon = map.addPolygon(new PolygonOptions().add(corner1,
corner2, corner3, corner4));
        polygon.setStrokeWidth(7);
        polygon.setStrokeColor(Color.CYAN);
```

```
polygon.setFillColor(getResources().getColor(R.color.magneta));

        // update parking lot type
        String no = "ei";
        String yes = "jah";

        if (currentParkingLot.isPrivate) privateLot.setText(yes);
        else privateLot.setText(no);

        // update likes and dislikes
        likeNr =
mockService.parkingLotData.get(currentParkingLot.id)[3];
        dislikeNr =
mockService.parkingLotData.get(currentParkingLot.id)[2];
        likes.setText(String.valueOf(likeNr));
        dislikes.setText(String.valueOf(dislikeNr));
    }

    public void updateParkingSpots() {
        // get free spot count & total parking spot number from the
web service
        int totalSpots =
mockService.parkingLotData.get(currentParkingLot.id)[1];
        int freeSpots =
mockService.parkingLotData.get(currentParkingLot.id)[0];
        freeSpotsValue.setText(String.valueOf(freeSpots));
        totalParkingSpotsValue.setText(String.valueOf(totalSpots));
    }

    public void updateMarkerAndLine() {
        parkingLotMarker.setVisible(false);
        line.setVisible(false);
        polygon.setVisible(false);
        parkingLotMarker = map.addMarker(new
MarkerOptions().position(temporaryLatLng).title("Parkla"));
        parkingLotMarker.showInfoWindow();

map.moveCamera(CameraUpdateFactory.newLatLngZoom(temporaryLatLng,
zoomLevel));
        line = map.addPolyline(new
PolylineOptions().add(temporaryLatLng,
destinationLatLng).color(Color.BLUE).geodesic(true));
    }

    public void infoForPrivateLot() {
        refresh.setVisibility(View.VISIBLE);
        freeSpotsText.setVisibility(View.VISIBLE);
        freeSpotsValue.setVisibility(View.VISIBLE);
        totalParkingSpotsText.setVisibility(View.GONE);
        totalParkingSpotsValue.setVisibility(View.GONE);
        layoutParams.setMargins(0,pixels, 0 , 0);
        freeSpotsText.setLayoutParams(layoutParams);
    }

    public void infoForPublicLot() {
        refresh.setVisibility(View.INVISIBLE);
        freeSpotsText.setVisibility(View.GONE);
        freeSpotsValue.setVisibility(View.GONE);
        totalParkingSpotsText.setVisibility(View.VISIBLE);
        totalParkingSpotsValue.setVisibility(View.VISIBLE);
```

```
    }

    public void refresh(View view) {
        mockService.refresh();
        updateParkingSpots();
    }

    public void like(View view) {
        // if current parking lot is not scored yet
        if (!likeGiven[currentParkingLot.id]) {
            likeNr += 1;
            likeGiven[currentParkingLot.id] = true;
            int[] values =
mockService.parkingLotData.get(currentParkingLot.id);
            values[3] = likeNr;
            mockService.parkingLotData.put(currentParkingLot.id,
values);
            likes.setText(String.valueOf(likeNr));
        } else Toast.makeText(ThirdActivity.this, "Olete seda parklat
juba hinnanud", Toast.LENGTH_SHORT).show();
    }

    public void dislike(View view) {
        // if current parking lot is not scored yet
        if (!likeGiven[currentParkingLot.id]) {
            dislikeNr += 1;
            likeGiven[currentParkingLot.id] = true;
            int[] values =
mockService.parkingLotData.get(currentParkingLot.id);
            values[2] = dislikeNr;
            mockService.parkingLotData.put(currentParkingLot.id,
values);
            dislikes.setText(String.valueOf(dislikeNr));
        } else Toast.makeText(ThirdActivity.this, "Olete seda parklat
juba hinnanud", Toast.LENGTH_SHORT).show();
    }
}
```

Lisa 6 – Strings.xml

```
<resources>
    <string name="app_name">Parkimisrakendus</string>
    <string name="confirm">Kinnita</string>
    <string name="chooseCenter">Vali keskus</string>
    <string name="privateLotsCheckBoxText">Lisa otsingutesse
eraparklad</string>
    <string name="next">Järgmine</string>
    <string name="previous">Eelmine</string>
    <string name="freeParkingLotsText">Vabade kohtade arv:</string>
    <string name="timeToWalkText">Kõndimise aeg:</string>
    <string name="distanceText">Kaugus sihtkohast:</string>
    <string name="totalParkingSpotsText">Parkimiskohtade arv:</string>
    <string name="number">arv</string>
    <string name="parkingLot">parkla</string>
    <string name="first">1.</string>
    <string name="descriptionText">Kirjeldus:</string>
    <string name="mapAParkingLot">Kaardista parkla</string>
    <string name="parkingLotType">Eraparkla:</string>
    <string name="or">VÕI</string>
    <string name="info">INFO</string>
    <string name="zero">0</string>
    <string name="title_activity_maps">Map</string>
</resources>
```

Lisa 7 – activity_first.xml

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:id="@+id/confirm"
        android:layout_width="343dp"
        android:layout_height="59dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="26dp"
        android:onClick="openMap"
        android:padding="8dp"
        android:text="@string/confirm"
        android:textColor="#ffffff" />

    <Spinner
        android:id="@+id/spinner"
        android:layout_width="217dp"
        android:layout_height="41dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="195dp"
        android:background="?attr/colorButtonNormal" />

    <TextView
        android:id="@+id/chooseCenter"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:gravity="center_horizontal"
        android:text="@string/chooseCenter"
        android:textSize="18sp"
        android:textStyle="bold" />

    <Button
        android:id="@+id/mapAParkingLot"
        android:layout_width="222dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="98dp"
        android:onClick="mapANewParkingLot"
        android:text="@string/mapAParkingLot" />

    <TextView
        android:id="@+id/title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="12dp"
        android:text="@string/app_name"
```

```
        android:textSize="24sp"
        android:textStyle="bold" />

    <View
        android:id="@+id/divider"
        android:layout_width="match_parent"
        android:layout_height="18dp"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="211dp"
        android:background="?android:attr/listDivider" />

    <TextView
        android:id="@+id/or"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/divider"
        android:layout_centerHorizontal="true"
        android:text="@string/or" />

    <Button
        android:id="@+id/info"
        style="@style/Widget.AppCompat.Button.Borderless.Colored"
        android:layout_width="71dp"
        android:layout_height="35dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="54dp"
        android:onClick="toggleInfo"
        android:text="@string/info" />

</RelativeLayout>
```

Lisa 8 – activity_second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <fragment
xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="504dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        tools:context="com.example.mapwithmarker.SecondActivity" />

    <Button
        android:id="@+id/button1"
        android:layout_width="341dp"
        android:layout_height="58dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="26dp"
        android:onClick="setDestination"
        android:padding="8dp"
        android:text="@string/confirm"
        android:textColor="#ffffff" />

    <CheckBox
        android:id="@+id/checkBox"
        android:layout_width="304dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="11dp"
        android:text="@string/privateLotsCheckBoxText" />

</RelativeLayout>
```

Lisa 9 – activity_third.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <fragment
xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="318dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentStart="true"
        android:layout_marginBottom="78dp"
        tools:context="com.example.mapwithmarker.ThirdActivity"
        android:layout_alignParentLeft="true" />

    <Button
        android:id="@+id/next"
        android:layout_width="140dp"
        android:layout_height="60dp"
        android:layout_alignParentEnd="true"
        android:layout_alignTop="@+id/previous"
        android:layout_marginEnd="32dp"
        android:onClick="getNextLot"
        android:text="@string/next"
        android:layout_alignParentRight="true"
        android:layout_marginRight="32dp" />

    <Button
        android:id="@+id/previous"
        android:layout_width="146dp"
        android:layout_height="58dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginBottom="11dp"
        android:layout_marginLeft="29dp"
        android:layout_marginStart="29dp"
        android:onClick="getPreviousLot"
        android:text="@string/previous" />

    <TextView
        android:id="@+id/distanceText"
        android:layout_width="139dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="52dp"
        android:layout_toLeftOf="@+id/next"
        android:layout_toStartOf="@+id/next"
        android:gravity="end"
        android:text="@string/distanceText" />
```

```
    <TextView
        android:id="@+id/timeToWalkText"
        android:layout_width="116dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="78dp"
        android:layout_toStartOf="@+id/next"
        android:gravity="end"
        android:text="@string/timeToWalkText"
        android:layout_toLeftOf="@+id/next" />

    <TextView
        android:id="@+id/freeSpotsText"
        android:layout_width="137dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="159dp"
        android:layout_toStartOf="@+id/next"
        android:gravity="end"
        android:text="@string/freeParkingLotsText"
        android:layout_toLeftOf="@+id/next" />

    <TextView
        android:id="@+id/distanceValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignTop="@+id/distanceText"
        android:layout_marginEnd="113dp"
        android:gravity="start"
        android:text="@string/zero"
        android:layout_alignParentRight="true"
        android:layout_marginRight="113dp" />

    <TextView
        android:id="@+id/timeToWalkValue"
        android:layout_width="46dp"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/distanceValue"
        android:layout_alignStart="@+id/distanceValue"
        android:layout_alignTop="@+id/timeToWalkText"
        android:gravity="start"
        android:text="@string/zero" />

    <TextView
        android:id="@+id/freeSpotsValue"
        android:layout_width="46dp"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/distanceValue"
        android:layout_alignStart="@+id/distanceValue"
        android:layout_alignTop="@+id/freeSpotsText"
        android:gravity="start"
        android:text="@string/zero" />

    <TextView
        android:id="@+id/queueNumber"
        android:layout_width="32dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="11dp"
```

```
        android:layout_toStartOf="@+id/distanceText"
        android:layout_toLeftOf="@+id/distanceText"
        android:gravity="end"
        android:text="@string/first"
        android:textSize="20sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/parkingLotName"
        android:layout_width="291dp"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/queueNumber"
        android:layout_alignParentEnd="true"
        android:layout_marginEnd="10dp"
        android:gravity="center_vertical"
        android:text="@string/descriptionText"
        android:textSize="18sp"
        android:textStyle="bold"
        android:layout_alignParentRight="true"
        android:layout_marginRight="10dp" />

    <TextView
        android:id="@+id/parkingLotTypeText"
        android:layout_width="113dp"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/refresh"
        android:layout_toStartOf="@+id/next"
        android:gravity="end"
        android:text="@string/parkingLotType"
        android:layout_toLeftOf="@+id/next" />

    <TextView
        android:id="@+id/parkingLotTypeValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/distanceValue"
        android:layout_alignStart="@+id/distanceValue"
        android:layout_alignTop="@+id/parkingLotTypeText"
        android:gravity="start"
        android:text="@string/zero" />

    <ImageButton
        android:id="@+id/refresh"
        android:layout_width="46dp"
        android:layout_height="46dp"
        android:layout_alignEnd="@+id/next"
        android:layout_alignRight="@+id/next"
        android:layout_alignTop="@+id/timeToWalkText"
        android:background="?attr/colorButtonNormal"
        android:contentDescription="@string/zero"
        android:onClick="refresh"
        app:srcCompat="@android:drawable/presence_online" />

    <ImageView
        android:id="@+id/dislikeButton"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:layout_alignTop="@+id/likeButton"
        android:layout_marginEnd="0dp"
        android:layout_toStartOf="@+id/likeButton"
        android:contentDescription="@string/zero"
```

```
        android:onClick="dislike"
        app:srcCompat="@android:drawable/arrow_down_float"
        android:layout_marginRight="0dp"
        android:layout_toLeftOf="@+id/likeButton" />

    <ImageView
        android:id="@+id/likeButton"
        android:layout_width="41dp"
        android:layout_height="40dp"
        android:layout_alignEnd="@+id/parkingLotName"
        android:layout_alignParentTop="true"
        android:layout_marginTop="194dp"
        android:contentDescription="@string/zero"
        android:onClick="like"
        app:srcCompat="@android:drawable/arrow_up_float"
        android:layout_alignRight="@+id/parkingLotName" />

    <TextView
        android:id="@+id/dislikes"
        android:layout_width="40dp"
        android:layout_height="wrap_content"
        android:layout_above="@+id/dislikeButton"
        android:layout_alignLeft="@+id/dislikeButton"
        android:layout_alignStart="@+id/dislikeButton"
        android:gravity="center"
        android:text="@string/zero"
        android:textColor="@android:color/holo_red_light"
        android:textSize="18sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/likes"
        android:layout_width="41dp"
        android:layout_height="wrap_content"
        android:layout_above="@+id/dislikeButton"
        android:layout_toEndOf="@+id/dislikeButton"
        android:gravity="center"
        android:text="@string/zero"
        android:textColor="@android:color/holo_green_light"
        android:textSize="18sp"
        android:textStyle="bold"
        android:layout_toRightOf="@+id/dislikeButton" />

    <TextView
        android:id="@+id/totalParkingSpotsText"
        android:layout_width="165dp"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="132dp"
        android:layout_toStartOf="@+id/next"
        android:gravity="end"
        android:text="@string/totalParkingSpotsText"
        android:layout_toLeftOf="@+id/next" />

    <TextView
        android:id="@+id/totalParkingSpotsValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/distanceValue"
        android:layout_alignStart="@+id/distanceValue"
        android:layout_alignTop="@+id/totalParkingSpotsText"
```

```
        android:gravity="start"
        android:text="@string/zero" />

</RelativeLayout>
```

Lisa 10 – MockWebService.java

```
package com.example.parkingapplication;


import java.util.Map;
import java.util.Random;
import java.util.TreeMap;

/**
 * Mock Web Service to imitate querying dynamic parking lot data from
the server
 */
public class MockWebService {

    TreeMap<Integer, int[]> parkingLotData = new TreeMap<>();

    public void initialize() {
        parkingLotData.put(1, new int[]{0,81,3,7});
        parkingLotData.put(2, new int[]{0,160,15,50});
        parkingLotData.put(3, new int[]{0,116,10,32});
        parkingLotData.put(4, new int[]{0,10,14,3});
        parkingLotData.put(5, new int[]{0,150,5,54});
        parkingLotData.put(6, new int[]{0,151,6,62});
        parkingLotData.put(7, new int[]{0,20,4,14});
        parkingLotData.put(8, new int[]{0,27,6,12});
        parkingLotData.put(9, new int[]{0,49,16,4});
        parkingLotData.put(10, new int[]{0,10,2,3});
        parkingLotData.put(11, new int[]{0,18,3,4});
        parkingLotData.put(12, new int[]{0,44,5,3});
        parkingLotData.put(13, new int[]{0,103,1,10});
        parkingLotData.put(14, new int[]{0,13,0,2});
        parkingLotData.put(15, new int[]{0,30,2,1});
        parkingLotData.put(16, new int[]{0,30,4,3});
        parkingLotData.put(17, new int[]{0,30,0,1});
        parkingLotData.put(18, new int[]{0,25,2,5});
        parkingLotData.put(19, new int[]{0,25,0,1});
        parkingLotData.put(20, new int[]{0,43,1,0});
        parkingLotData.put(21, new int[]{0,40,0,1});
        parkingLotData.put(22, new int[]{34,34,1,10});
        parkingLotData.put(23, new int[]{177,177,3,10});
        parkingLotData.put(24, new int[]{23,23,0,11});
        parkingLotData.put(25, new int[]{68,68,0,15});
        parkingLotData.put(26, new int[]{47,47,0,16});
        parkingLotData.put(27, new int[]{41,41,0,13});
        parkingLotData.put(28, new int[]{89,89,0,19});
        parkingLotData.put(29, new int[]{35,35,0,19});
        parkingLotData.put(30, new int[]{89,89,0,19});
        parkingLotData.put(31, new int[]{16,16,0,19});
        parkingLotData.put(32, new int[]{10,10,0,19});
        parkingLotData.put(33, new int[]{7,7,0,19});
        parkingLotData.put(34, new int[]{5,5,0,19});
    }

    /**
     * Update parking lots free spot number with a random number
     * ceiling is the total number of spots in a given parking lot
     */
    public void refresh() {
```

```
        Integer key;
        int[] values;
        Random random = new Random();

        for (Map.Entry<Integer,int[]> entry :
parkingLotData.entrySet()) {
            key = entry.getKey();
            values = entry.getValue();
            values[0] = random.nextInt(values[1]);
            parkingLotData.put(key, values);
        }
    }
}
```