TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Annabel Sulg 185121
Helena Sultson 185966
Triin Nõmm 185231
Yvonne Pärn 185079

Eesti Energia AS kütusekulu simulaatori rakenduse arendus

Bakalaureusetöö

Juhendajad: Jekaterina Tšukrejeva
MSc.
Kristina Murtazin
MSc.

Tallinn 2021

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: Triin Nõmm

18.05.2021

Annotatsioon

Eesti Energia AS poolt loodi projekt, mille eesmärgiks oli arendada rakendus sisepõlemismootoriga ja elektriauto kütusekulude ning saaste võrdluseks. Projektiga toetati Eesti Energia AS ärilisi eesmärke: aidata kaasa elektriautode populariseerimisele Eesti elanike seas ning olla kaasatud tavasõiduautolt elektrikütusega auto peale ülemineku protsessis. Töö valmis kahes etapis. Esimeses etapis ehk meeskonnaprojekti raames täideti esialgne ettevõtte seatud eesmärk – näidata kasutaja sõiduauto ning valitud elektriauto kütusekulude võrdlust, kuid valminud rakendus oli algelise kasutajaliidesega ning puudulike funktsionaalsustega. Teises etapis ehk lõputöös täiendati rakendust lisafunktsioonidega, kujundati vastavaks Eesti Energia brändi Enefit Volt stiiliga ning korrastati rakenduse kood.

Lõputööna valmis Androidi mobiilirakendus, mis pakub kasutajale võimalust näha võrdlust enda sisestatud sõiduauto ning võrdluseks valitud elektriauto vahel. Võrdluses tuuakse välja erinevused sõidukite kütusekulu ja heite vahel ning näidatakse rahalist säästu elektriautot kasutades. Projekti jooksul korrastati rakenduse kood ning kujundati ümber kasutajaliides, millega parandati ka kasutajakogemuse disaini.

Lõputööna valmis Android mobiilirakendus, mis pakub kasutajale võimalust näha võrdlust sisestatud sõiduauto ning valitud elektriauto vahel. Võrdluses tuuakse välja erinevused sõidukite kütusekulu ja CO_2 heite vahel ning näidatakse rahalist säästu elektriautot kasutades. Projekti jooksul korrastati rakenduse kood ning kujundati ümber kasutajaliides, millega parandati kasutajakogemust.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 68 leheküljel, 5 peatükki, 54 joonist, 5 tabelit.

Abstract

Electric vehicle fuel cost simulator app for Eesti Energia AS

Enefit Volt is a brand of Eesti Energia AS which provides smart electric car charging services to homes, businesses and fleets with over 180 public charging stations across Estonia. Eesti Energia AS provides both charging solutions and the electricity needed.

The readiness to switch a regular car for an electric car in Estonia is very low. As Eesti Energia AS researched during a Design Sprint event, the main argument against an electric car is the initial purchase price. When comparing the cost of fuel for both cars, there is a considerable amount to be saved by using an electric vehicle.

With this in mind, Enefit Volt decided to launch a project to create an additional functionality for the Enefit Volt webpage and app. The aim of the project was the creation of an Android mobile application to compare the user-provided car's fuel cost with a selected electric vehicle's electricity cost.

The project was completed in two stages. The first stage met the main criteria provided by Eesti Energia AS, but lacked the correct user interface and additional functionalities. With this thesis, the second stage of the project was completed with the necessary functions and improvements.

During the second stage of the project, improvements such as multilanguage support, CO_2 calculations and grid fees for calculating the correct electricity cost were added. The second stage also covered correcting and redesigning the application's user interface. In addition, it is now possible to keep track of the application's releases and run the application via an APK file.

The thesis is in Estonian and contains 68 pages of text, 5 chapters, 54 figures, 5 tables.

Lühendite ja mõistete sõnastik

Activity	Klass, mis esindab ühe lehe vaadet kasutajaliidesega
Agile coach	Agiilise arendusmetoodika juhendaja
API	*Application Programming Interface* ehk rakendusliides
APK	*Android Package* ehk paketi failiformaat, mida kasutab Android operatsioonisüsteem rakenduse käivitamiseks
CI/CD	*Continuous Integration/Continuous Delivery* ehk pideva integreerimise ja pideva edastamise tava
CSS	*Cascading Style Sheets* ehk veebilehe küljendamise märgistuskeel
Data scraping	*Data scraping* ehk *web scraping* on tehnika, kus arvutiprogramm impordib andmed veebisaidilt
Entity	Olem, mida kasutatakse eksisteeriva objekti kirjeldamiseks.
Fragment	Taaskasutatav osa rakenduse kasutajaliideses
Instrumented test	Test, millega kontrollitakse Androidi-spetsiifilisi rakenduse osi
Intent	Kavatsus või eesmärk ehk sõnum, mida edastatakse komponentide vahel
JSON	*JavaScript Object Notation* on lihtsustatud andmevahetusvorming, mis põhineb *JavaScript*'i programmeerimiskeele alamhulgal
Locale	*Locale* ehk lokaat on parameetrite kogum, mis määrab kasutaja keele, regiooni ja kõik erilised variantide eelistused, mida kasutaja soovib oma kasutajaliideses näha
OTF	*OpenType Font* ehk skaleeritavate arvuti fontide formaat
Back-end	Rakenduse loogika ja andmete salvestamine
Front-end	Graafiline kasutajaliides
Framework	Raamistik
HEX	*Hexadecimal color* ehk värvikood
HTTP	*Hypertext Transfer Protocol* ehk võrguprotokoll teabe edastamiseks arvutivõrkudes
Model	Koponent, milles hoitakse rakenduse loogikat
MVVM	*Model-View-ViewModel* arhitektuurimuster

Scrum master	*Scrum master* ehk agiilse arendusmeeskonna juhendaja
SRP	*Single-responsibility principle* ehk üksik vastavuse põhimõte
Toast	Hüpikteade
UI	*User Interface* ehk kasutajaliides
URL	*Uniform Resource Locator* ehk üldine infoallika asukohamääraja
View	Komponent milles hoitakse kasutajaliidese loogikat
ViewModel	Suhtluskomponent *Model*'i ja *View*'i vahel
Pipeline	*Pipeline* ehk järjestatud tööde voog
PMD	*Programming Mistake Detector* ehk staatiline lähtekoodi analüsaator

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Enefit Volt on Eesti Energia AS laadimisteenuse bränd, mis pakub nutikaid elektriauto laadimisteenuseid äridele, kodudele ja autoparkidele. Tegemist on Eesti suurima elektriautode avaliku laadimisvõrguga ligi 180 laadimispunktiga üle Eesti. Lisaks pakutakse laadijate paigaldamist ja jooksvat hooldamist äri- ja eraklientidele.

Elektri ja elektriautode laadijate müüjana on ettevõtte kliendibaas piiratud elektriauto omanikega, kellele saab teenust pakkuda. Kaudselt on suurimaks turukonkurendiks hetkel sõiduauto. Turuosakaalu suurendamiseks on tarvis esmalt suuremat nõudlust ehk suuremat elektriautode omajate hulka, kellele pakutav teenus saab kasulik olla.

Ettevõtte poolt 2019. aasta juunis korraldatud disainisprindi raames leiti, et üks põhjustest, miks inimesed eelistavad tavakütusega autot elektriautole on see, et elektriautot peetakse kallimaks. Ettevõtte poolt käsitsi arvutatud kulude võrdluse tulemused näitasid, et kütuse asemel elektriga auto laadimisega kaasneb oluline rahasääst. Eesti Energia AS soovis leida lahenduse, kuidas näidata vastavat võrdlust tavainimesele, kes käsitsi taolisi arvutusi läbi ei vii. Turunduslikel eesmärkidel soovitakse olla inimesega kaasas kogu protsessi vältel – elektriauto valimisest kuni laadijapakkuja valimiseni.

Seetõttu otsustati Enefit Volt mobiilirakenduse ja kodulehe jaoks välja töötada lisafunktsioon, et aidata kaasa elektriautode kasutamise populariseerimisele Eestis. Projekt valmis Tallinna Tehnikaülikooli tudengite Triin Nõmm, Annabel Sulg, Helena Sultson ja Yvonne Pärn ning Eesti Energia AS koostöös kahes etapis.

Esimeses etapis meeskonnaprojekti õppeaine raames valmis rakenduse põhi, mille abil oli võimalik kasutaja sõiduauto andmete põhjal näha kütusekulu võrdlust valitud elektriautoga.

Esimese etapi arendustegevus dokumenteeriti ning kaitsti meeskonnaprojekti aine läbimiseks. Valminud mobiilirakendus täitis Eesti Energia AS poolt seatud põhieesmärgi, milleks oli näidata kliendi sõiduauto ja valitud elektriauto kütusekulude võrdlust.

Teise etapi arendustööks oli projektil mitmeid funktsionaalseid ja mittefunktsionaalseid nõudeid.

Olulised funktsionaalsed nõuded on CO_2 emissiooni näitamine sisepõlemismootoriga auto puhul, elektrihinnale kuutasu ning võrgutasude lisamine, kütusekulude võrdluse viimine kindlale ajaperioodile, rakenduse koodi korrastamine ja rakenduse mitmekeelsus. Rakendusele lisati lisaks inglise keelele ka eesti ja vene keel, et seda saaksid kasutada võimalikult paljud Eesti elanikud.

Selleks, et rakenduse kasutaja näeks elektriauto omamise kasulikkust, ei piisa ainult kütusekulude võrdlusest. Kuna oluline argument elektriauto kasuks on keskkonnasääst CO_2 emissiooni arvelt, oli seda vaja rakenduses kasutajale näidata. Keskkonnasäästlikkust tuli kasutajale rakenduses arusaadavalt illustreerida, sest CO_2 emissiooni arvandmed üksinda võivad olla raskelt hoomatavad.

Mittefunktsionaalne nõue on kasutajasõbralik UI. Rakenduse kasutajaliidese disain ei mahtunud meeskonnaprojekti ajalisse piirangusse, mistõttu lisati rakendusele lõputöö jooksul Enefit Volt brändi ametlik stiil.

Vastavalt Eesti Energia AS projekti püstitusele ning esimese etapi lõpus äriosakonna poolt pakutud lisaideedele olid lõputöö eesmärgid järgnevad:

- näidata kasutajale elektriauto omamise keskkonnasäästlikkust CO_2 emissiooni arvelt ning illustreerida seda kliendile
- arvutada elektriauto ja sisepõlemismootoriga auto kütusekulude võrdlusandmed 5 aasta perioodi peale
- lisada elektrihinnale Eesti Energia AS võrgutasud ja kuutasu
- lisada mitmekeelsus (inglise, eesti ja vene keel)
- ehitada kasutajaliides üles Enefit Volt ametlikus stiilis
- refaktoorida rakenduse *back-end* ja *front-end*

Käesolev lõputöö koosneb neljast põhilisest osast: metoodika, töö tulemused, analüüs ja järeldused ning kokkuvõte.

Metoodikas kirjeldatakse, mis sai nii projekti esimeses kui ka teises etapis tehtud, milliseid tööriistu ja tehnoloogiaid kasutati ning milline nägi välja tööprotsess. Töö tulemustes tuuakse välja projekti tehniline dokumentatsioon ning antakse ülevaade valminud rakendusest. Analüüsi ja järelduste osas põhjendatakse töö tehnilist teostust, antakse ülevaade kirjandusest ja tuuakse välja hinnang projekti teostamise protsessi kohta. Lisades tuuakse välja meeskonnaliikmete isiklik panus ja eneseanalüüs.

2 Metoodika

Järgnevalt antakse ülevaade projekti esimeses ja teises etapis valminud objektist. Lisaks kirjeldatakse kasutatud tööriistu ning tööprotsessi.

2.1 Objekti ülevaade

Eesti Energia AS poolt puudus projektil eelnev arendustöö. Rakendus valmis kahes etapis, meeskonnaprojekti õppeaine raames ja bakalaureusetöö edasiarendusena.

Projekti esimeses etapis on loodud rakenduse põhi andmete pärimiseks ja salvestamiseks, andmebaas ning esialgne funktsionaalsus sisepõlemismootoriga auto ja elektriauto võrdluseks.

Sõiduauto andmete salvestamiseks on loodud rakenduses ühendus Maanteeameti ARIS2 andmevahetusteenusega. Auto registrinumbri sisestamisel pöördub rakendus Maanteeameti infosüsteemi poole, esitab HTTP(s) protokolli alusel POST-tüüpi päringu ning saadud XML dokumendist salvestatakse vajalikud andmed 72 tunniks rakenduse andmebaasi. Andmete salvestamisega vähendatakse päringute arvu, sest päringute arv minutis, tunnis ning ööpäevas on piiratud ja tasustatud.

Kütusehindade saamiseks Fuelo veebilehelt [1] kasutatakse andmete kraapimise (*data scraping*) tehnoloogiat, mis võimaldab avalikult veebilehelt andmeid salvestada. Projektis kasutatakse Jsoup Java teeki, mille abil võetakse veebilehe HTML dokumendist kindla elemendi väärtus ning salvestatakse andmebaasi koos ajatempliga. Andmebaasi salvestatakse nii bensiini kui ka diisli hind 1 liitri kohta, mida rakendus ajatempli kontrollimise abil iga päev automaatselt uuendab. Kütuse hindade andmed salvestatakse ja uuendatakse taustal toimuva protsessina, s.t kasutajaliides ei pea ootama andmete salvestamise järgi.

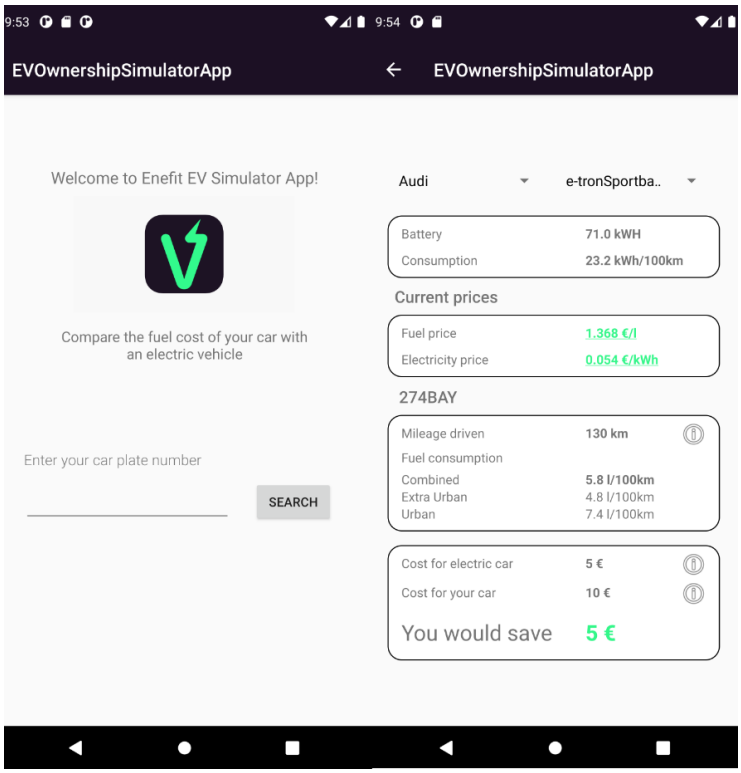
Elektriautode andmed saadakse Eesti Energia AS arendajate poolt loodud API kaudu. Rakenduse käivitamisel kutsutakse HTTP(s) protokolli alusel POST-tüüpi päring ja saadakse tagasi XML dokumendi formaadis kõikide elektriautode andmed.

Andmete salvestamiseks on loodud rakenduse juurde Room andmebaas, mis sisaldab SQLite andmebaasi ja lisafunktsioone. Andmebaasiklass *Database* on rakenduse peamine ühenduspunkt andmebaasiga, kus määratakse andmebaasi ehitamise loogika. Ühtlasi on kasutusel Repository klassid, mis loovad andmetele ligipääsu ümber abstraktsiooni.

Esimeses etapis valminud Android mobiilirakenduses on kasutajal võimalik sisestada esilehel oma sõiduauto registreerimisnumber ning järgneval võrdluse lehel valida meelepärane elektriauto. Valiku põhjal kuvati kasutajale elektriauto ja sisestatud sõiduauto andmed, kütusehinnad ning kütusekulude võrdlus (Joonis 1). Kasutajal oli võimalik valida ka sõiduauto erinevate kütusetarbimise tüüpide vahel linnas, maanteel ja kombineeritult.

Elektriauto kütusekulu leiti rakenduses kasutaja sõiduauto kahe viimase tehnoülevaatuse vahel läbitud vahemaa, elektrihinna ning elektriauto elektrikulu põhjal.

Kasutaja sõiduauto kütusekulu leiti rakenduses kahe viimase tehnoülevaatuse vahel läbitud vahemaa, käesoleva päeva kütusehinna ja auto kütusetarbimise põhjal.



Joonis 1. Rakendus meeskonnaprojekti lõpus

Projekti teises etapis on täiendatud rakendust lisafunktsioonidega, kasutajaliides kujundatud Enefit Volt ametlikus stiilis ja refaktooritud rakenduse *front-end* ning *back-end*. Kogu süsteemi BPNM diagrammi täiendati ning sinna lisati roheliselt funktsionaalsused, mis oli vaja arendada või muuta teise etapi jooksul (Lisa 2).

Rakendus on mitmekeelne. Kasutaja saab valida pealehel inglise, eesti ja vene keele vahel. Kasutajaliides on ümber disainitud vastavalt Enefit Volt brändi stiilile.

Kasutajal on pealehel võimalik võrdluseks valida elektriauto ja sisestada sõiduauto registreerimisnumber. Pealehel kuvatakse vastava elektriauto pilt, et valimist kasutaja jaoks lihtsustada. Järgnevalt avaneb rakenduses võrdluse leht, kus näidatakse sõidu- ja elektriauto andmeid, kütusekulusid ning nende võrdlust.

Kütusekulude arvutused on viidud 5 aasta perioodile, et erinevate autode puhul oleks võrdlus samaväärne. Täpsema 5 aasta keskmise läbisõidu saamiseks võetakse arvutustel aluseks auto esimene ja viimane olemasolev tehnoülevaatuse odomeetri näit.

Elektriautode täpsema kütusekulu arvutamiseks on lisaks elektrihinnale võetud arvesse ka võrgutasud ning võrguteenuse kuumakse.

Lisaks kütusekulude võrdlusele kuvatakse kasutajale sõidukite CO_2 heite võrdlus, et näidata elektriauto keskkonnasäästlikkust.

2.2 Tööriistade kirjeldus

Projekti arendamise keskkond on Android Studio. Rakendus arendati Java programmeerimiskeeles. Rakendus kasutab Android tarkvara 4.1 (Jelly Bean), Java JDK 8 ning Android SDK 16+. Projektis kasutatakse Java versiooni 8, sest see on viimane versioon, mille kasutamist Android SDK täielikult toetab. Arenduse automatiseerimiseks on kasutusel tööriist Gradle.

Arendamistegevuses kasutatakse pideva integratsiooni (*Continuous Integration*) tava, mille kohaselt laetakse kood ühisesse hoidlasse. Lähtekoodihoidlana kasutatakse GitHub veebimajutusteenust, kuhu laetakse kood peale muudatusi üles. Koodi kvaliteedi automaatseks kontrollimiseks kasutati automatiseeritud koodi testimist GitHub Actions tööriista abil. Selleks loodi *Workflow* fail, mis sisaldab ülesandeid koodi puhtuse kontrollimiseks, ühiktestide jooksutamiseks ning rakenduse käivitamiseks *pipeline'*i sees.

GitHub teenuses oli arendusprotsess jagatud ülesanneteks, mille projekti liikmed omavahel ära jagasid. Ülesandeid teostati individuaalselt või paarides.

Kui ülesanne on valmis ning muudatused laetakse *master branch*'i üles, käivitub protsess rakenduse *release*'i valmimiseks. Kõigepealt genereeritakse rakenduse versiooni tähistamiseks unikaalne number, misjärel pannakse lindil kokku rakenduse lähtekood .zip failina. *Pipeline*'i töö tulemusena genereeritakse .apk fail, et rakenduse viimast versiooni oleks nii meeskonnaliikmetel kui ka juhendajatel võimalik kiiresti Android Studio tarkvaraprogrammi omamata tööle panna. APK ehk *Android Package* on Android operatsioonisüsteemis kasutatav failiformaat. Selle abil on rakendus võimalik arvutiga ühendatud Android telefonis või Androidi emulaatoris lihtsasti käima panna.

2.3 Tööprotsessi kirjeldus

Arendusprotsessi efektiivseks kulgemiseks kasutati agiilset arendusmetoodikat. Tööpäevadeks, mille jooksul sai kindlasti tagasisidet mentoritelt ja tooteomanikult, lepiti kokku neljapäevad ning reeded. Iseseisev töö, mis ei vajanud juhendajate abi, toimus muudel päevadel vastavalt meeskonnaliikmete endi soovidele.

Igal kokku lepitud tööpäeva hommikul kell 9:45 neljapäeviti ja 11:45 reedeti toimus *stand-up* koosolek, millest võtsid osa kõik tiimi liikmed, mentorid ning ettevõttepoolne juhendaja. Koosolek toimus Kanban põhimõtetel GitHub keskkonnas loodud virtuaalse tahvli ees, kus ülesanded olid jagatud erinevate veergude vahel: vaja teha (*to do*), tegemisel (*in progress*), tiimisisesel ülevaatamisel (*in team review*), ülevaatamisel (*review in progress*) ja valmis (*done*). Koosolekut viis läbi *agile coach*. Iga tiimiliige kirjeldas lühidalt, millega tegeles eelmisel tööpäeval, millega kavatses tegeleda käesoleval päeval ning millised olid probleemid. Koosoleku eesmärk oli hoida kogu tiimi ning mentoreid meeskonna tegevusega kursis ning saada jooksvalt lahendusi tekkinud probleemidele.

Arendamisel peeti oluliseks meeskonna iseseisvust. Ülesanded olid jaotatud meeskonnaliikmete vahel eraldi või rakendati paarisprogrammeerimist. Kõik meeskonnaliikmed tegelesid projekti jooksul nii *front-end*, *back-end* kui ka muude tehniliste ülesannetega. Iga meeskonnaliikme roll oli olla nii arendaja kui ka testija.

Tiimiliikmed hoidsid üksteist kursis käesolevate töödega demonstreerides valminud koodi. Enne muudatuste üles lükkamist *master branch*'i, teostasid tiimiliikmed üksteise töödele ülevaatusi, kuni tehtud muudatustega rahul oldi. Seejärel esitati muudatused mentoritele ülevaatamiseks.

Probleeme prooviti esialgselt lahendada iseseisvalt, seejärel meeskonnasiseselt ning keerulisemate probleemide puhul pöörduti juhendajate poole.

Arendustegevus jagunes kolme vaheetapi (*milestone*) vahel, millest igaüks kestis ligikaudu ühe kuu. Esimese vaheetapi käigus kujundati kasutajaliides, parandati olemasoleva rakenduse vead ning lisati rakenduse käivitamise alternatiivina APK faili genereerimine. Teise vaheetapi jooksul realiseeriti rakenduses uus kasutajaliides, lisati mitmekeelsus ja muudeti arvutuste loogikat 5 aasta peale. Viimases vaheetapis refaktooriti *front-end* ja *back-end,* lisati avalehele vastava elektriauto pilt, võrdluse lehele CO_2 andmed ja täiendati elektrihinna arvutamist võrgutasudega.

Iga kuu lõpus toimus kokkuvõttev koosolek, kus näidati ettevõttepoolsetele juhendajatele valminud tulemusi ning pandi paika uue kuu tegevuskava. Koosolek koosnes kolmest osast: tagasiside eelmisele kuule ehk retrospektiiv, tulemuste näitamine ja uue kuu planeerimine.

Retrospektiivi raames arutati, mis läks eelmisel kuul hästi meeskonnatöö osas, individuaalselt ja juhendajatega suhtlemisel. Seejärel arutati, mis oleks võinud kuu jooksul paremini minna. Retrospektiivi tulemused märkis üles *agile coach* ning igal meeskonnaliikmel oli kohustus järgmise kuu eesmärkide täitmisel parandada eelmisel kuul esinenud vigu, unustatud kohustusi jms.

Järgmisena toimus tulemuste näitamine (*demo*) ettevõttepoolsetele juhendajatele ja äriosakonna esindajatele. *Demo* eesmärk oli saada regulaarset tagasisidet tehtud tööle. Kogu arendustöö jooksul oli vaja meeles pidada, et kuu lõpus on vaja ettevõttele näidata mingit reaalset tulemust. See sundis ning innustas arendustegevust paremini planeerima ja vajadusel kiiremini meeskonnaliikmetelt või juhendajatelt abi küsima.

Viimasena planeeriti uue kuu eesmärgid ehk vaadati üle uued ülesanded, et kõik tiimiliikmed ja juhendajad saaksid ülesannete nõuetest samamoodi aru, ning lepiti kokku,

kui palju ja milliseid ülesandeid hakkab meeskond tegema. Uue kuu eesmärkide seadmisel lähtuti sellest, et kokku lepitud ülesanded on meeskonna lubadus ettevõtte ees.

3 Töö tulemused

Tulemuste osas antakse ülevaade valminud rakendusest tehnilise dokumentatsioonina, mille osad on: nõuded, rakenduse ülevaade, arhitektuur, disain, kood ja testid.

3.1 Nõuded

Esimese etapi lõpus tegi meeskond Enefit Volt äripoolele ülevaate projektist. Selle käigus esitati meeskonnale teiseks etapiks nõuded, mis on allpool kirjeldatud funktsionaalsete nõuetena. Funktsionaalsed nõuded kirjeldavad programmi ülesandeid ehk kasutajale loodavat väärtust.

Teise etapi alguses täiendasid meeskonnaliikmed juhendajatega tarkvara ja nõuete projekteerimise kriteeriumeid, mis on allpool kirjeldatud mittefunktsionaalsete nõuetena. Mittefunktsionaalsete nõuete alusel sätestatakse süsteemi töökäik.

Funktsionaalsed nõuded:

1. Võrdluses kindlaks määratud aeg: aeg on fikseeritud viiele aastale, millele vastavalt arvutatakse keskmine läbisõit.

2. Elektriautod pealehel: pealehel on võimalik enne sõiduauto registreerimisnumbri sisestamist valida võrreldav elektriauto, millest kuvatakse ka pilt.

3. Elektrihinna täiendus: elektrihinnale on lisatud Elektrilevi OÜ võrgutasud ja võrguteenuse kuutasu.

4. CO_2 emissioon: võrdluses kuvatakse CO_2 emissiooni põhjal elektriauto keskkonnasäästlikkus võrreldes kasutaja sõiduautoga.

Mittefunktsionaalsed nõuded:

1. Kasutajaliides: kasutajaliidese kujundus järgib Enefit Volt stiili ja on vastavuses prototüübiga.

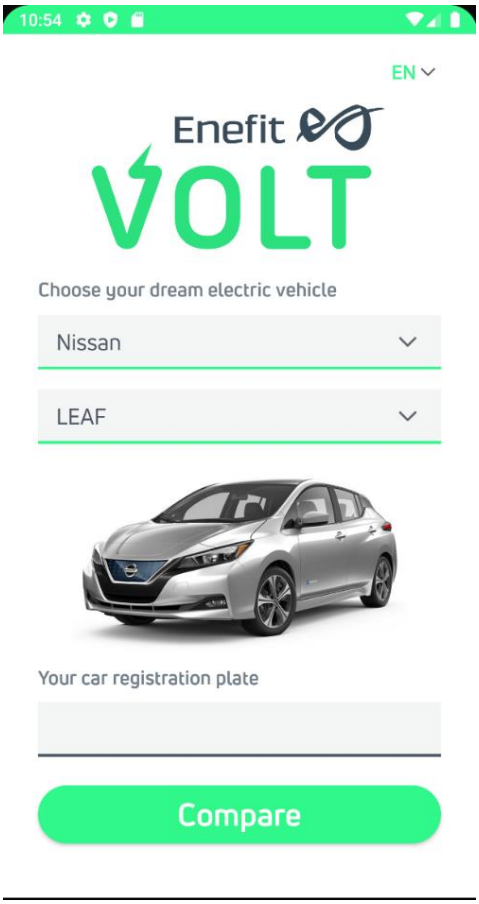
2. Mitmekeelsus: rakenduses on rippmenüü, mille kaudu saab valida rakenduse keele (eesti, vene, inglise).

3. Rakendust saab GitHub'ist APK (*Android Package*) failina alla laadida ning telefonis või emulaatoris käivitada.
4. Projekti arendamisel kasutatakse CI/CD arendustava GitHub Actions abil.
5. Rakenduses jälgitakse puhta arhitektuuri põhimõtteid.
6. Rakenduses kasutatakse MVVM arhitektuurimustrit.
7. Integratsioonitestide koodikattuvus on vähemalt 75%.
8. Ühiktestide koodikattuvus on vähemalt 75%.

Kogu arendustöö ajal oli igal meeskonnaliikmel kohustuslik järgida mittefunktsionaalseid nõudeid. Funktsionaalsed nõuded jagati ülesanneteks, mida hakati realiseerima.

3.1.1 Rakenduse avaleht

Rakenduse käivitamisel avaneb kasutajale esmalt avaleht (Joonis 2). Projekti teises etapis disainiti kogu avaleht ümber vastavalt äripoole nõuetele. Rakendusele lisati keele muutmise ja elektriauto valimise võimalus koos vastava elektriauto pildi kuvamisega.



Joonis 2. Rakenduse avaleht

Vaikimisi avaneb rakendus inglise keeles, kuid kasutajal on soovi korral võimalik rippmenüüst valida ka eesti või vene keel (Joonis 3). Keele muutmisel taaskäivitatakse avaleht ning tekstiväljad uuendatakse valitud keele tõlgetega.

Joonis 3. Keelevaliku rippmenüü avalehel

Edasi on kasutajal võimalik valida elektriauto, millega soovitakse võrdlust läbi viia. Elektriautode andmed saadakse Enefit Volt API kaudu. Saadud andmed salvestatakse andmebaasitabelisse *electric_vehicles* ja tabelit uuendatakse iga kuu kõige uuemate elektriauto andmete saamiseks.

Elektriauto valimiseks loodi kaks rippmenüüd. Esimesest rippmenüüst on võimalik valida erinevate elektriauto markide vahel. Kui kasutaja on valinud endale meelepärase elektriauto margi, laetakse vastava margi mudelid teise rippmenüüsse. Teisest rippmenüüst on kasutajal võimalik valida elektriauto mudel. Mõlemas rippmenüüs on väärtused tähestikulises järjekorras. Vaikeväärtusena on esialgseks elektriauto valikuks määratud Nissan LEAF. Iga valitud elektriauto puhul kuvatakse kasutajale vastava elektriauto pilt rippmenüüde all.

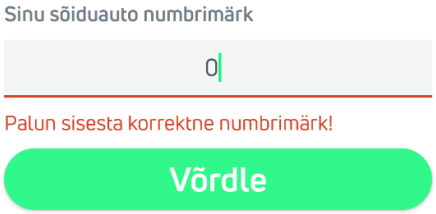
Lehe alumises osas on tekstiväli, kuhu kasutaja saab sisestada oma sõiduauto registreerimisnumbri. Vajutades nupule "Võrdle" pöördub rakendus Maanteeameti infosüsteemi poole ja esitab HTTP(s) protokolli alusel POST-tüüpi päringu sisestatud auto registreerimisnumbri järgi. API-st saadud vastus töödeldakse ja otsitud auto vajalikud tehnilised andmed salvestatakse 72 tunniks andmebaasitabelisse *cars*.

Kui kasutaja on auto kohta juba päringu teinud ning sellest ei ole möödunud rohkem kui 72 tundi, siis sama auto sisestamisel API poole pöördumist ei toimu, vaid andmed laetakse otse andmebaasist.

Kui sisestatud registreerimisnumber vastab Eesti registreerimismärgi nõuetele ning sellele vastav auto eksisteerib, suunatakse kasutaja edasi võrdluse lehele. Avalehel valitud elektriauto margi ja mudeli info ning kasutaja sõiduauto andmed edastatakse avalehelt (*MainActivity*) võrdluse lehele (*ComparisonActivity*), kasutades selleks Androidile omast *Intent*'i.

Kui kasutaja on jätnud sõiduauto registreerimisnumbri sisestamata, kuvatakse kasutajale veateade (Joonis 4). Vigase sisendi korral ehk juhul, kui sõiduauto registreerimisnumber ei vasta nõuetele, kuvatakse kasutajale veateade (Joonis 5). Kui sisestatud sõiduautot Maanteeameti API-s ei eksisteeri, kuid registreerimisnumber vastab nõuetele, kuvatakse kasutajale veateade puuduva sõiduauto kohta (Joonis 6).

Joonis 4. Veateade sõiduauto registreerimisnumbri puudumise korral



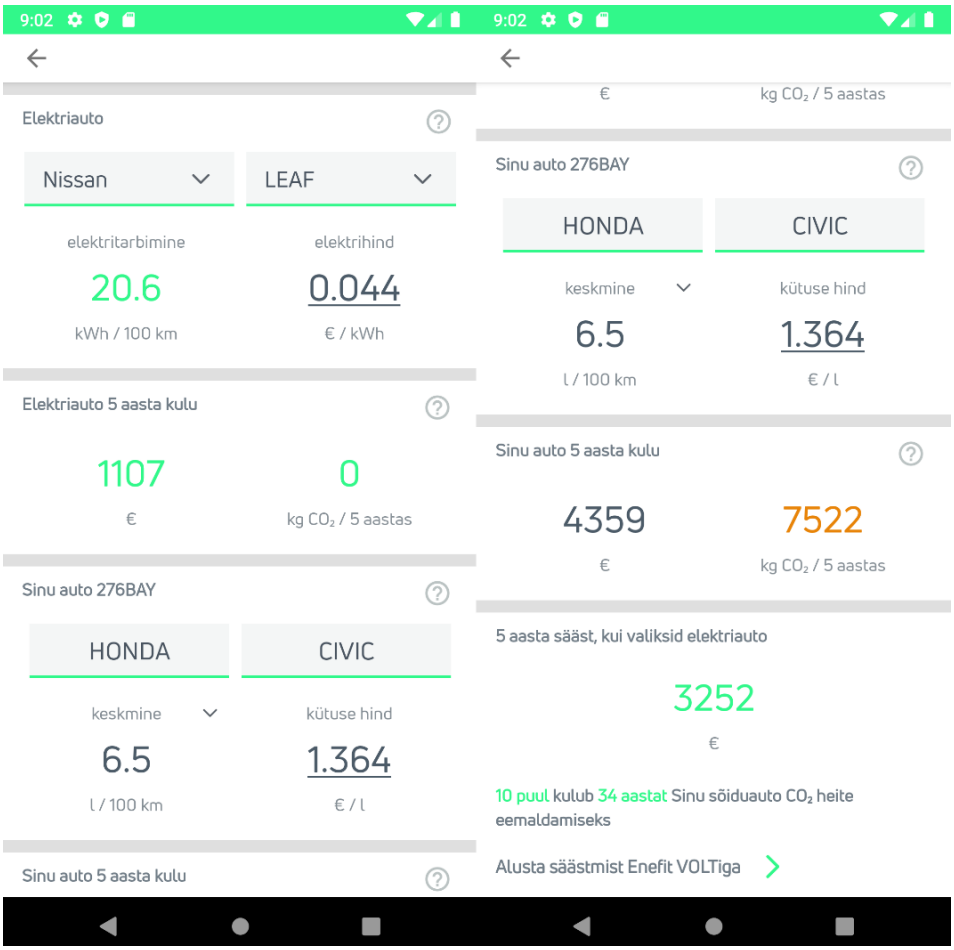
Joonis 5. Veateade vigase sõiduauto registreerimisnumbri korral

Joonis 6. Veateade sõiduauto puudumise korral

Kõikide eelnimetatud juhtude korral edasisuunamist võrdluse lehele ei toimu.

3.1.2 Võrdluse leht

Avalehel korrektse auto registreerimisnumbri sisestamisel ja “Võrdle” nupule vajutamisel suunatakse kasutaja edasi võrdluse lehele (Joonis 7). Projekti teises etapis disainiti kogu võrdluse leht ümber vastavalt äripoole nõuetele ja lisati CO_2 võrdlus. Täpsema võrdluse läbiviimiseks muudeti kõik arvutused 5 aasta perioodile ja elektriauto kütusekulu arvutamisel võeti arvesse ka võrgutasusid.



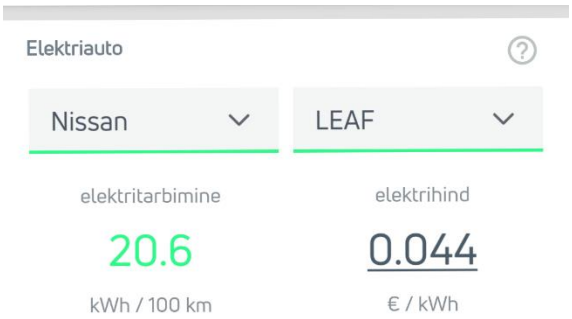
Joonis 7. Võrdluse leht

Võrdluse lehel näeb kasutaja elektrile ja kütusele kuluva rahalise summa ning CO_2 emissiooni koguste võrdlust valitud elektriauto ning sisestatud sõiduauto vahel. Võrdluse leht on jagatud viide sektsiooni: elektriauto info, elektriauto 5 aasta kütusekulu, kasutaja sõiduauto info, kasutaja sõiduauto 5 aasta kütusekulu ja 5 aasta sääst.

Võrdluse lehe ülaosas navigatsiooniribal olevale nooleikooniga nupule vajutades suunatakse kasutaja tagasi avalehele.

3.1.3 Võrdluse lehe esimene sektsioon

Esimeses sektsioonis kuvatakse valitud elektriauto kohta käivad vajalikud tehnilised andmed (Joonis 8).



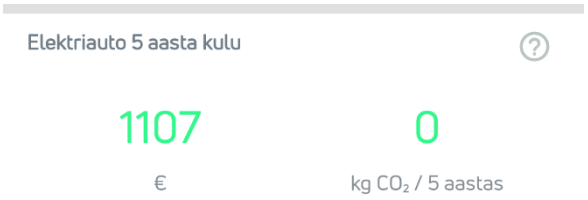
Joonis 8. Elektriauto info sektsioon

Sektsiooni ülemises osas on kaks rippmenüüd – üks elektriauto margi ja teine mudeli valikuks. Kasutajal on soovi korral võimalus valida võrdluseks mõni muu elektriauto, ilma, et peaks selleks tagasi liikuma avalehele. Teise elektriauto valimisel uuenevad kõik arvutuste väärtused automaatselt.

Rippmenüüde all kuvatakse võrdluse läbiviimiseks vajalikke tehnilisi andmeid – vastava elektriauto elektrikulu 100 kilomeetri kohta ning eelmise kuu keskmine elektrihind. Elektrihinna tekstivaatele on külge lisatud link, et kasutajal oleks soovi korral võimalik kontrollida elektrihinna õigsust. Elektrihinna väärtuse peale vajutades suunatakse kasutaja Nord Pool veebilehele [2].

3.1.4 Võrdluse lehe teine sektsioon

Võrdluse lehe teises sektsioonis kuvatakse elektriauto 5 aasta kulu (Joonis 9). Esimene väärtus näitab, kui palju raha kulub 5 aasta lõikes elektri peale elektriautoga sõites.



Joonis 9. Elektriauto 5 aasta kütusekulu sektsioon

Elektriauto kütusekulu leitakse käesoleva kuu keskmise elektrihinna, vastava elektriauto elektrikulu, võrgutasude ja kasutaja sõiduauto 5 aasta keskmise läbisõidu põhjal.

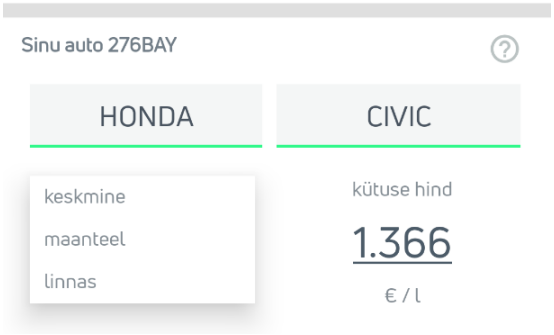
Lisaks elektrile kuluvale summale on teises sektsioonis välja toodud elektriauto CO_2 emissioon. See näitab, mitu kilogrammi süsinikdioksiidi paiskub õhku 5 aasta jooksul, sõites valitud elektriautoga. CO_2 vaikeväärtus on andmebaasis 0 (g/km).

3.1.5 Võrdluse lehe kolmas sektsioon

Kolmandas sektsioonis kuvatakse kasutaja sisestatud sõiduauto vajalikud tehnilised andmed võrdluse läbiviimiseks (Joonis 10). Sektsiooni ülaosas on välja toodud auto üldised andmed – auto registreerimisnumber, mark ja mudel. Auto margi ja mudeli väljade all kuvatakse auto kütusetarbimine ja käesoleva päeva kütuse hind.

Joonis 10. Kasutaja sõiduauto info sektsioon

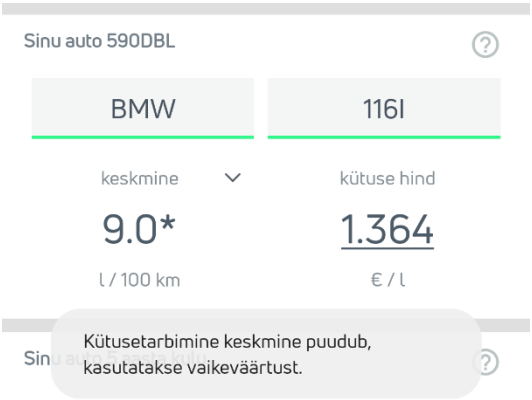
Auto kütusetarbimiseks on vaikimisi määratud keskmine kütusetarbimine. Kütusetarbimise väärtuse kohal paiknevast rippmenüüst on võimalik valida ka muud tüüpi tarbimine, millega võrdlust läbi viia (Joonis 11). Muu tüübi valimisel uuenevad kõik arvutuste väärtused automaatselt.



Joonis 11. Kütusetarbimise valiku rippmenüü võrdluse lehel

Kütuse hinna tekstivaatele on külge lisatud link, mis kütuse hinna peale vajutades suunab kasutaja Fuelo veebilehele [1]. Vastavalt autole kuvatakse kütuse hinna tekstivaates bensiini (A95) või diisli hind.

Kui autol puuduvad kütusetarbimise väärtused, määratakse andmebaasis vaikeväärtused. Kütusetarbimise tekstivaates lisatakse vaikeväärtustele lõppu tärn ning selle peale vajutades kuvatakse kasutajale hüpikteade ehk *Toast* (Joonis 12).



Joonis 12. Kütusetarbimise vaikeväärtus

Kombineeritud ehk keskmise kütusetarbimise vaikeväärtuseks määratakse andmebaasis 9.0, linnas 10.7 ja maanteel 7.4 liitrit kilomeetri kohta.

3.1.6 Võrdluse lehe neljas sektsioon

Neljandas sektsioonis kuvatakse kasutaja sõiduauto 5 aasta kütusekulu (Joonis 13). Esimene väärtus näitab, kui palju raha kulub sisestatud sõiduautoga kütuse peale 5 aasta lõikes. Sõiduauto kütusekulu leitakse käesoleva päeva kütuse hinna, valitud kütusetarbimise tüübi väärtuse ja kasutaja sõiduauto 5 aasta keskmise läbisõidu põhjal.

Joonis 13. Kasutaja sõiduauto 5 aasta kütusekulu sektsioon

Lisaks on neljandas sektsioonis välja toodud kasutaja sõiduauto CO_2 emissioon ehk mitu kilogrammi süsinikdioksiidi paiskub õhku 5 aasta jooksul, sõites sisestatud sõiduautoga.

Kui autol puudub CO_2 väärtus, määratakse andmebaasis vaikeväärtus. CO_2 tekstivaates lisatakse vaikeväärtusele tärn. Vaikeväärtuse peale vajutades kuvatakse kasutajale hüpikteade ehk *Toast*, mis teavitab kasutajat vaikeväärtuse kasutamisest (Joonis 14).

Joonis 14. CO_2 vaikeväärtus

CO_2 vaikeväärtuseks määratakse andmebaasis 150 grammi kilomeetri kohta.

3.1.7 Võrdluse lehe viies sektsioon

Viiendas sektsioonis kuvatakse võrdluse tulemus kasutaja sisestatud sõiduauto ja valitud elektriauto kütusekulude kohta (Joonis 15).

Joonis 15. Viie aasta säästu sektsioon

Esmalt on välja toodud 5 aasta jooksul võimalik säästetav summa, kui kasutaja sisestatud sõiduauto asemel sõidetaks valitud elektriautoga. Säästetav summa leitakse sõiduauto kütusele ja valitud elektriauto elektrile kuluva summa põhjal.

Lisaks kuvatakse kasutajale illustreeriv lause, mis selgitab, mitu aastat kulub 10 puul sisestatud sõiduauto 5 aasta CO_2 heite eemaldamiseks. Teades, et 10 keskmist puud seovad aastas ligikaudu 220 kg süsinikdioksiidi [3], leitakse sõiduauto CO_2 koguse põhjal aastate arv.

Kuna valminud projekti kavatseti Eesti Energia AS poolt kasutada turunduslikel eesmärkidel, oli üks nõuetest, et rakendus peaks võimaldama kasutajal jõuda Enefit Volt kodulehele. Seetõttu on sektsiooni alumises osas nupp “Alusta säästmist Enefit VOLTiga”, millele vajutades suunatakse kasutaja edasi Enefit Volt veebilehele [4].

3.1.8 Hüpikaknad

Üks nõue projektile oli, et kasutaja näeks täpselt, mis andmete põhjal arvutused on tehtud ja saaks soovi korral veenduda arvutuste õigsuses. Seetõttu loodi projekti teises etapis hüpikaknad sektsioonide elektriauto info, elektriauto 5 aasta kulu, kasutaja sõiduauto info ja kasutaja sõiduauto 5 aasta kulu juurde. Sektsioonide paremal ülaosas olevad küsimärgiga ikoonid tähistavad nuppe, millele vajutades avanevad hüpikaknad lisainformatsiooniga. Iga hüpikakna alumises osas on nupp “Sulge” ja ülemisel ribal ikoon “X”, mille peale vajutades suunatakse kasutaja tagasi võrdluse lehele.

Elektriauto info sektsioonis olevale ikoonile vajutades avaneb hüpikaken, kus kuvatakse valitud elektriauto kohta käivad tehnilised andmed (Joonis 16).

Joonis 16. Elektriauto info sektsiooni hüpikaken

Hüpikakna ülemisel ribal kuvatakse elektriauto mark ja mudel. Edasi on välja toodud konkreetse elektriauto sõiduulatus, mis leitakse elektriauto aku mahutavuse ja elektrikulu põhjal. Sõiduulatuse järel kuvatakse kasutajale elektriauto aku mahutavuse ning elektrikulu andmed.

Elektriauto 5 aasta kütusekulu sektsioonis olevale ikoonile vajutades avaneb hüpikaken, kus kuvatakse elektriauto kütusekulu leidmiseks vajalikud tehnilised andmed, et kasutaja saaks soovi korral kalkulatsioonides ise veenduda (Joonis 17).

Joonis 17. Elektriauto 5 aasta kütusekulu sektsiooni hüpikaken

Hüpikaknas kuvatakse järgmised andmed: kütusekulu ehk summa, mis kuluks elektrile, sõites elektriautoga, eelmise kuu keskmine elektrihind, elektrikulu, võrgutasude summa ja kasutaja sõiduauto 5 aasta läbisõit. Võrgutasude andmed saadi Elektrilevi ametlikust võrguteenuste hinnakirjast [5]. Võrgutasude summa leitakse elektrienergia ülekandekulu ja võrgutasude igakuise maksumuse põhjal ning arvutatakse ümber 5 aasta perioodile. Elektrienergia ülekandekulu leitakse valitud elektriauto elektrikulu põhjal.

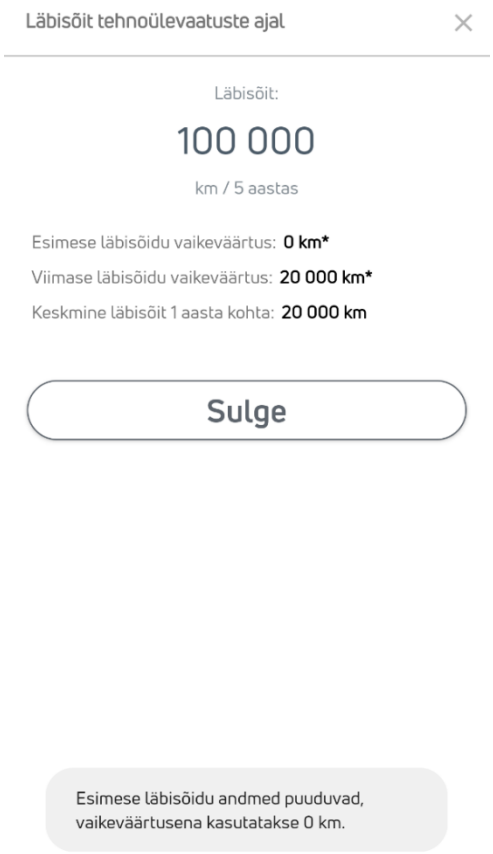
Kasutaja sõiduauto info sektsioonis olevale ikoonile vajutades avaneb hüpikaken, kus kuvatakse sisestatud sõiduauto tehnoülevaatuste ja nende põhjal arvutatud auto läbisõidu andmed (Joonis 18).

Joonis 18. Kasutaja sõiduauto info sektsiooni hüpikaken

Hüpikakna ülaosas kuvatakse sõiduauto 5 aasta keskmine läbisõit, mis leitakse auto esimese ja viimase tehnoülevaatuse ajal registreeritud odomeetri näidu põhjal ning arvutatakse ümber 5 aasta lõikesse.

Läbisõidu järel on välja toodud auto esimese ja viimase tehnoülevaatuse andmed: kuupäev ja odomeetri näit.

Juhul, kui autol on tehtud ainult üks tehnoülevaatus või ei ole tehtud ühtegi, määratakse andmebaasis esimese ja viimase tehnoülevaatuse kuupäevadeks ja läbisõitudeks vaikeväärtused. Läbisõitude vaikeväärtusteks määratakse vastavalt äripoole soovile 0 km ja 20 000 km, mis on standardläbisõidud autoliisingu puhul. Hüpikaknas lisatakse läbisõitude vaikeväärtustele tärn. Vaikeväärtuse peale vajutades teavitatakse kasutajat vaikeväärtuse kasutamisest hüpikteate ehk *Toast* kaudu (Joonis 19).



Joonis 19. Auto läbisõitude vaikeväärtused

Tehnoülevaatuste andmete all kuvatakse kasutajale ka sisestatud sõiduauto keskmist läbisõitu ühe aasta kohta. Tulemus leitakse tehnoülevaatuste läbisõitude põhjal ning arvutatakse kuupäevi arvesse võttes ümber ühe aasta perioodile.

Kasutaja sõiduauto 5 aasta kütusekulu sektsioonis olevale ikoonile vajutades avaneb hüpikaken, kus kuvatakse auto kütusekulu leidmiseks vajalikud tehnilised andmed, et kasutaja saaks soovi korral kalkulatsioonides ise veenduda (Joonis 20).

Joonis 20. Kasutaja sõiduauto 5 aasta kütusekulu sektsiooni hüpikaken

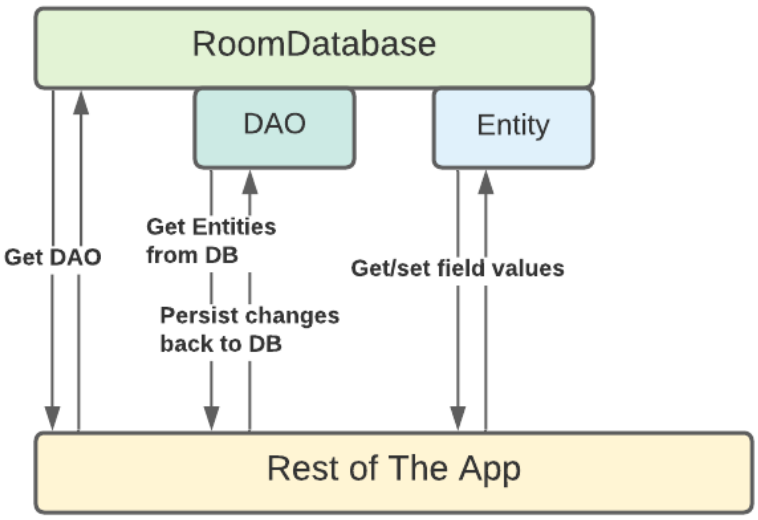
Hüpikaknas kuvatakse järgmised andmed: sõiduauto kütusekulu, kütuse hind, kütusetarbimise väärtus vastavalt valitud tüübile, CO_2 kogus ja auto 5 aasta läbisõit.

3.2 Arhitektuur

Rakenduse arendamisel säilitati ning rakendati puhta arhitektuuri põhimõtteid [6]. Andmebaasi ja äriloogika sõltuvuse vältimiseks kasutatakse *Repository* disainimustrit. Hoidmaks kasutajaliidest eraldi osana, kasutati rakenduses MVVM (*Model-View-ViewModel*) arhitektuurimustrit. Lisaks hoiduti lisateekide ning raamistike kasutamisest.

3.2.1 Rakenduse arhitektuur

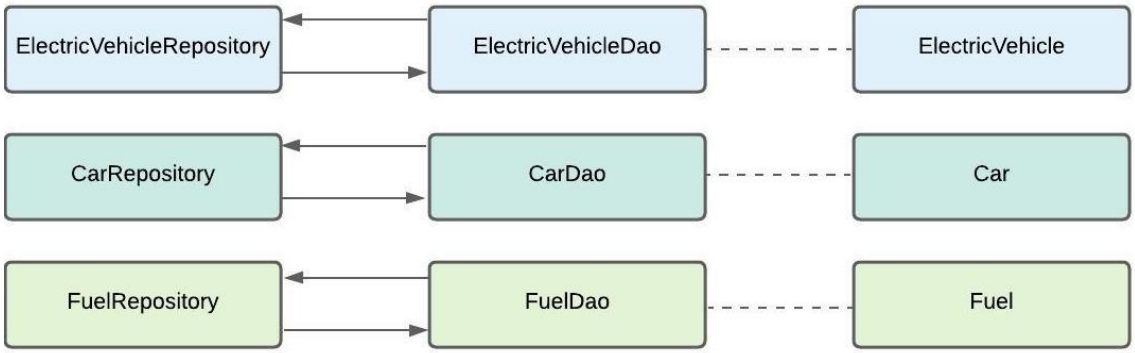
Rakenduses kasutatakse andmete vahemällu salvestamiseks Room abstraktsiooniga SQLite andmebaasi. Room andmebaas on rakenduses järgmiste komponentidena: andmebaas (*database*), andmete objekt (*Entity*) ning andmete ligipääsuks vajalike meetodite liidest (*interface*) DAO (*Data Access Object*) (Joonis 21).



Joonis 21. Room andmebaasi struktuur

Andmebaasi loomiseks kasutatakse rakenduses *Database* klassi. *Car*, *ElectricVehicle* ja *Fuel* on *Entity* klassid ning vastavalt neile on loodud DAO liidesed: *CarDao*, *ElectricVehicleDao* ja *FuelDao*.

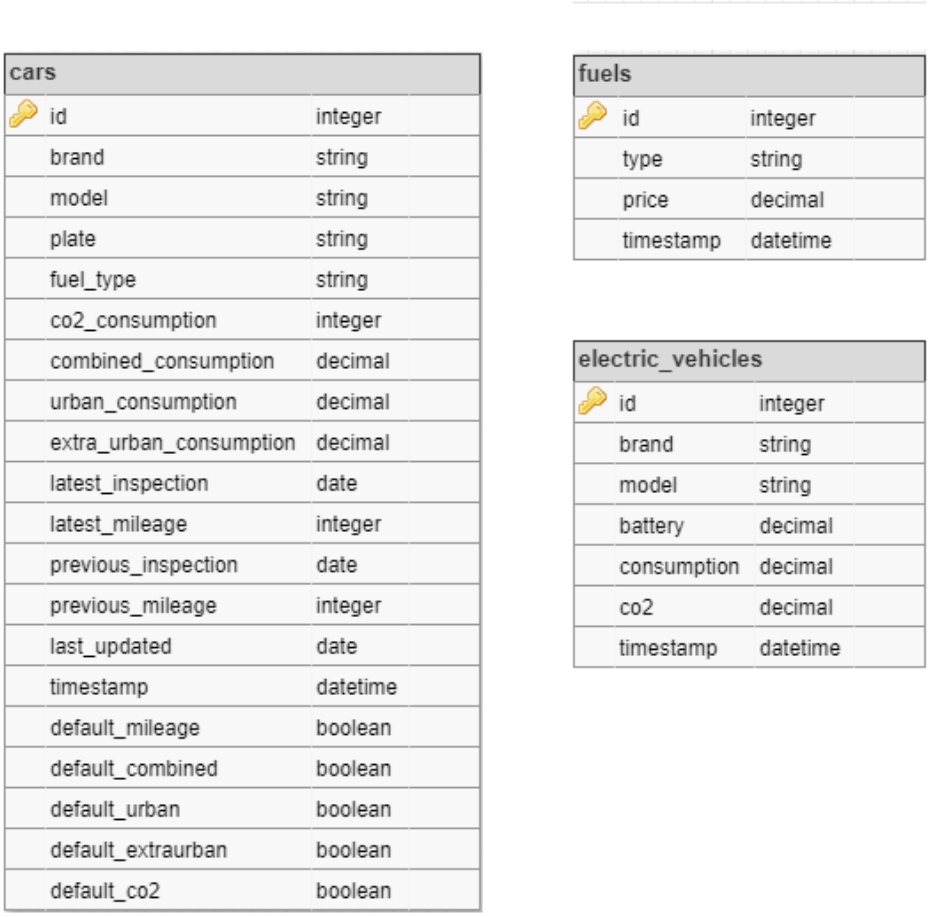
Database andmebaasiklass on rakenduse peamine ühenduspunkt andmebaasiga, kus määratakse selle ehitamise loogika. Andmebaasiga suhtlemisel kasutatakse *Repository* disainimustrit [7]. Mustri rakendamiseks on tehtud eraldi klassid Car*Repository*, *ElectricVehicleRepository* ja *FuelRepository*, mis suhtlevad vastavate DAO liidestega (Joonis 22).



Joonis 22. *Repository* ja DAO suhtlus

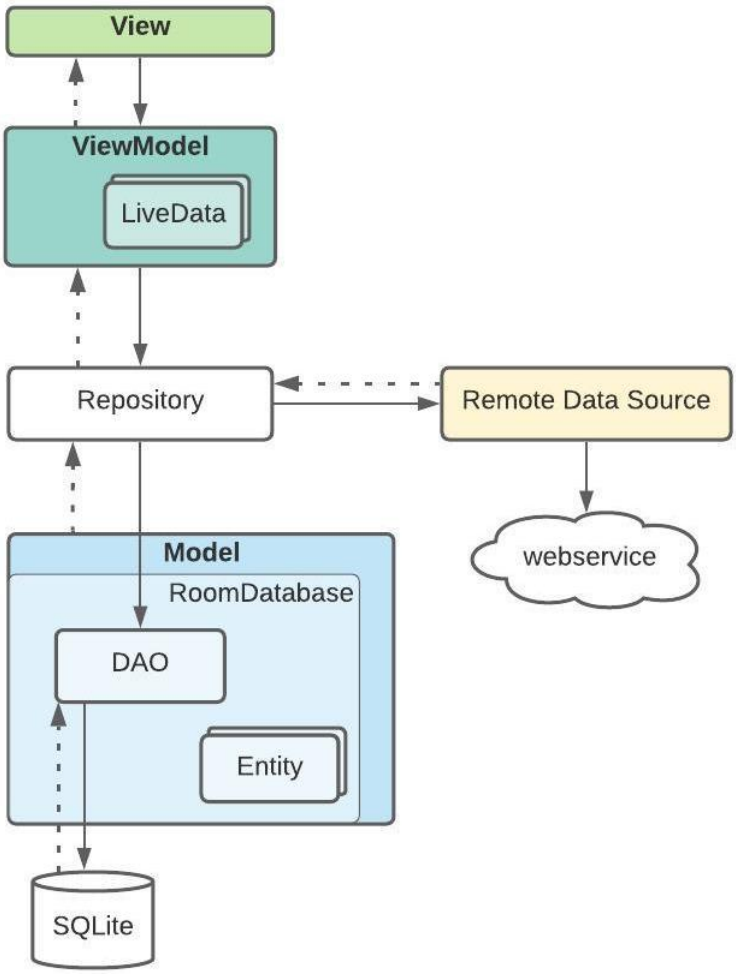
Rakenduse andmebaas koosneb kolmest tabelist: *cars*, *electric_vehicles* ja *fuels*, mis sisaldavad vastavate andmete veerge. Teises etapis ehk bakalaureusetöö raames lisati andmebaasi *cars* tabelisse juurde kaheksa veergu: *brand*, *model*, *co2_consumption*, *default_mileage*, *default_combined*, *default_urban*, *default_extraurban* ning *default_co2*.

Viimased viis välja on *Boolean* tüüpi ning kirjeldavad sõidukile määratud vaikimisi väärtuseid. Lisaks lisati *electric_vehicles* ja *cars* tabelitele juurde CO_2 heite veerud (Joonis 23).



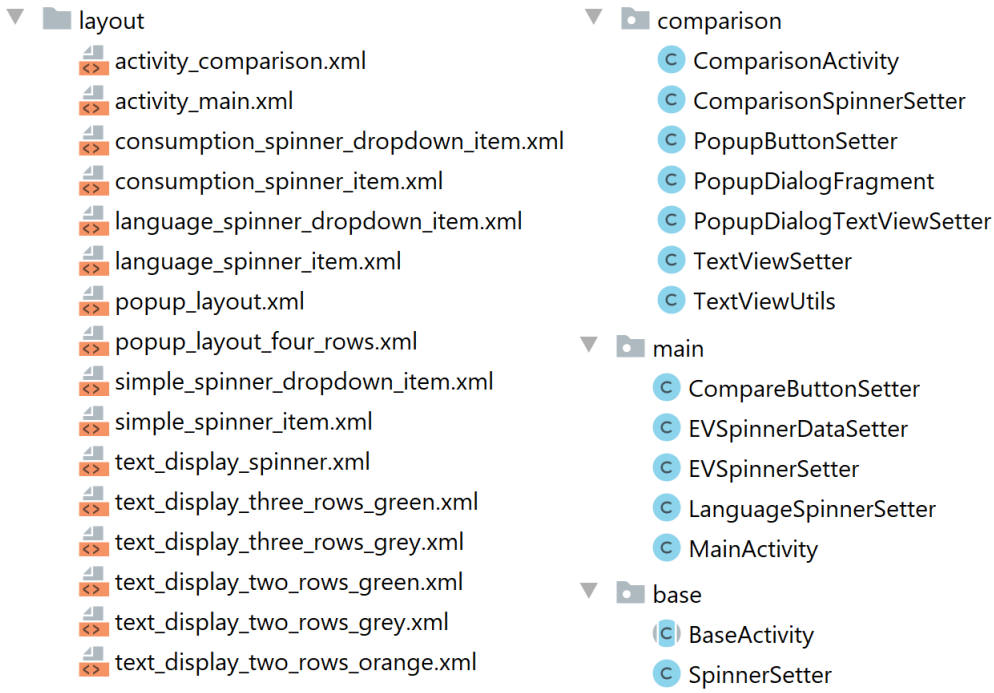
Joonis 23. Andmebaasi mudel

Rakenduse arendamisel jälgiti MVVM (*Model-View-ViewModel*) arhitektuurimustrit. MVVM on arhitektuurimuster, mis eraldab *View* ehk vaate loogika rakenduse loogikast ja võimaldab luua kergesti hallatava ning testitava rakenduse [8] (Joonis 24).



Joonis 24. MVVM arhitektuurimuster rakenduses

View komponent vastutab kasutajale kuvatava vaate eest. *View* komponentide hulka kuuluvad antud rakenduses kasutajaliidese *layout* failid, *Activity* klassid, *Fragment* klass ning nende alamklassid (Joonis 25).



Joonis 25. *View* komponendid

Model komponendis hoitakse rakenduse andmeid ja äriloogikat. Loodud rakenduses on *Model* esitatud DAO liidestena ja *Entity* klassidena.

ViewModel on suhtluskanal *Model* ja *View* komponentide vahel. *ViewModel* teatab *View*'le andmete muudatuste kohta ning vastavalt sellele uuendatakse andmeid kasutajaliideses. Antud rakenduses on igale *Entity* objektile tehtud vastav *ViewModel* klass. Näiteks *Car* objekti *ViewModel*'iks on *CarViewModel* klass.

3.2.2 Koodi arhitektuur

Koodi kirjutamisel jälgis meeskond erinevaid disainimustreid. Need jaotuvad kolme kategooriasse: loomemustrid (*Creational patterns*), struktuurimustrid (*Structural patterns*) ning käitumismustrid (*Behavioural patterns*).

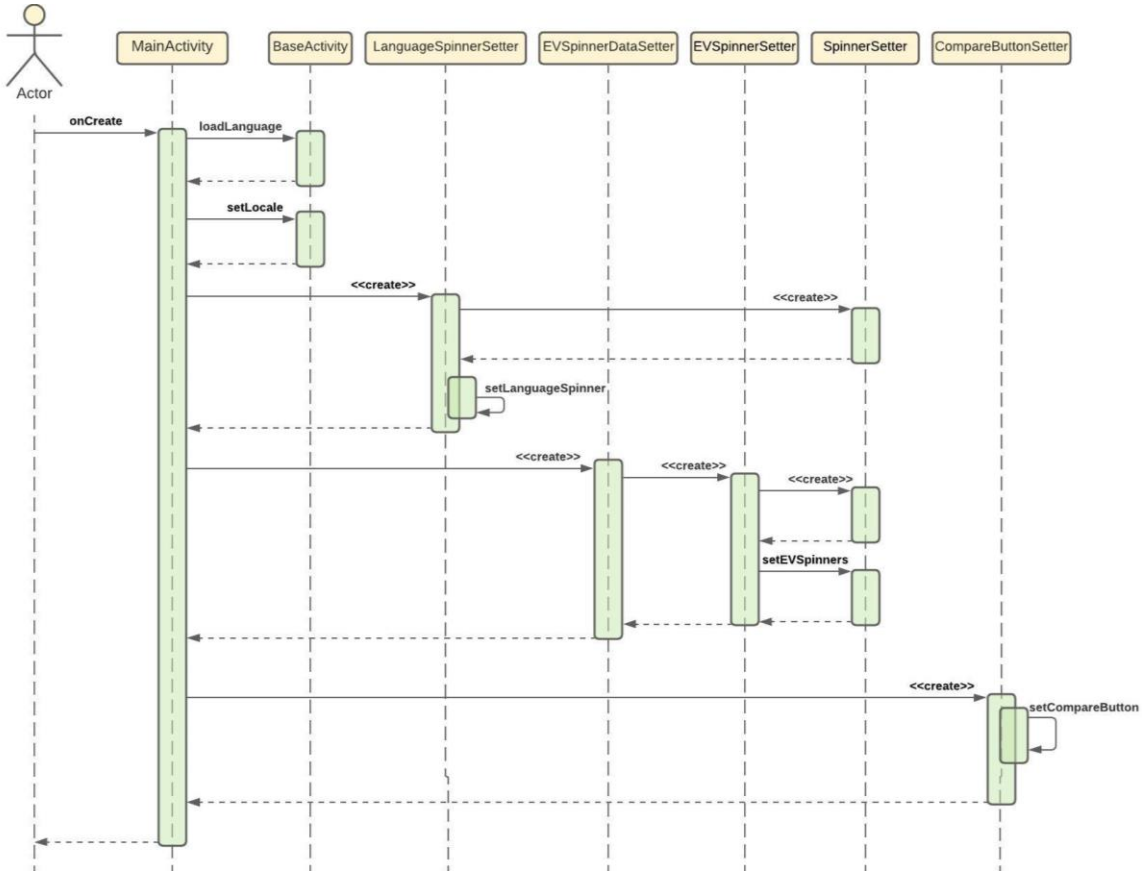
Loomemustreid kasutatakse objektide loomisel. Antud rakenduses on kasutatud järgmiseid loomemustreid: Ehitaja (*Builder)*, Singel (*Singleton*) ning Sõltuvusesüst (*Dependency injection). Builder* mustrit kasutatakse hüpikakende loomisel. *Singleton* mustrit rakendati andmebaasiklassi loomisel. Teisel etapil refaktooriti koodi, mille käigus kasutati *Dependency injection* mustrit. *Dependency injection* muster võeti kasutusele

keelemuutmise funktsionaalsuse lisamisel, kasutades *SharedPreferences* liidest *loadLanguage()* meetodis (Joonis 26).

```
public String loadLanguage(String defaultLanguage) {
    SharedPreferences pref =
        getApplicationContext().getSharedPreferences("MyPref",
Context.MODE_PRIVATE);
    return pref.getString("lan", defaultLanguage);
}
```

Joonis 26. *loadLanguage()* meetod

Struktuurimustreid kasutatakse objektide vaheliste suhete realiseerimiseks. Rakenduses on peamiselt kasutusel Fassaadi (*Facade*) muster, mida kasutatakse arvutusülesannete teostamisel ning vaadete loomisel. Teises etapis kasutati *Facade* mustrit *Activity* klasside refaktoorimisel. *MainActivity* ja *ComparisonActivity* klasse kasutati fassaadina, mille meetod *onCreate()* käivitab vajalikud protsessid ning loob rakenduse vaate (Joonis 27).



Joonis 27. *MainActivity* diagramm

Käitumismustrid määravad objektide vahelise toimimise. Käsk (*Command*) ja Vaatleja (*Observer)* – need on kaks rakenduses peamiselt kasutatud käitumismustrit.

Command mustrit on peamiselt kasutatud *Activity* klassis, mis kutsub välja *onCreate()*, *onClickListener()*, *onItemSelectedListener()* ja muid sarnaseid meetodeid. Lisaks käivitatakse asünkroonne klass *EVSpinnerDataSetter*, mille käigus imporditakse või uuendatakse elektriautode andmed ning tulemusena seadistatakse elektriautode rippmenüüd. Kütuse hindade *data scraping* on samuti asünkroonne protsess, mille tulemusena salvestatakse või uuendatakse need andmebaasis.

Observer muster on projektis kasutusel *LiveData* kaudu, mis teavitab rakendust andmebaasis tehtud muudatuse kohta. *LiveData* on Androidi-põhine andmehoidja klass, mis jälgib muudatusi *View* komponendis ning uuendab aktiivse *View* andmeid (Joonis 28).

```
protected void setEVData(String model) {
    ElectricVehicleViewModel.getEVByModelLiveData(model)
            .observe(
                    lifecycleOwner,
                    ev -> {
                        electricVehicle = ev;
                        this.evConsumption = ev.getConsumption();
                        textViewSetter
                                .setEVConsumption(evConsumption);
                                        textViewSetter
                          .setCostForEV(evConsumption);
                        textViewSetter
                           .setSaving(evConsumption,
                              spinnerConsumptionType);});
                        }
```

Joonis 28. *setEVData*() meetod

Arenduse teises etapis kasutati refaktoorimise käigus Strateegia (*Strategy*) mustrit. *Strategy* mustrit kasutatakse hüpikakende sisu ning rippmenüü adapteri vaate määramisel. Adapteri vaate määramisel valib kood vastavalt rippmenüü objektile selle loomise meetodi (Joonis 29).

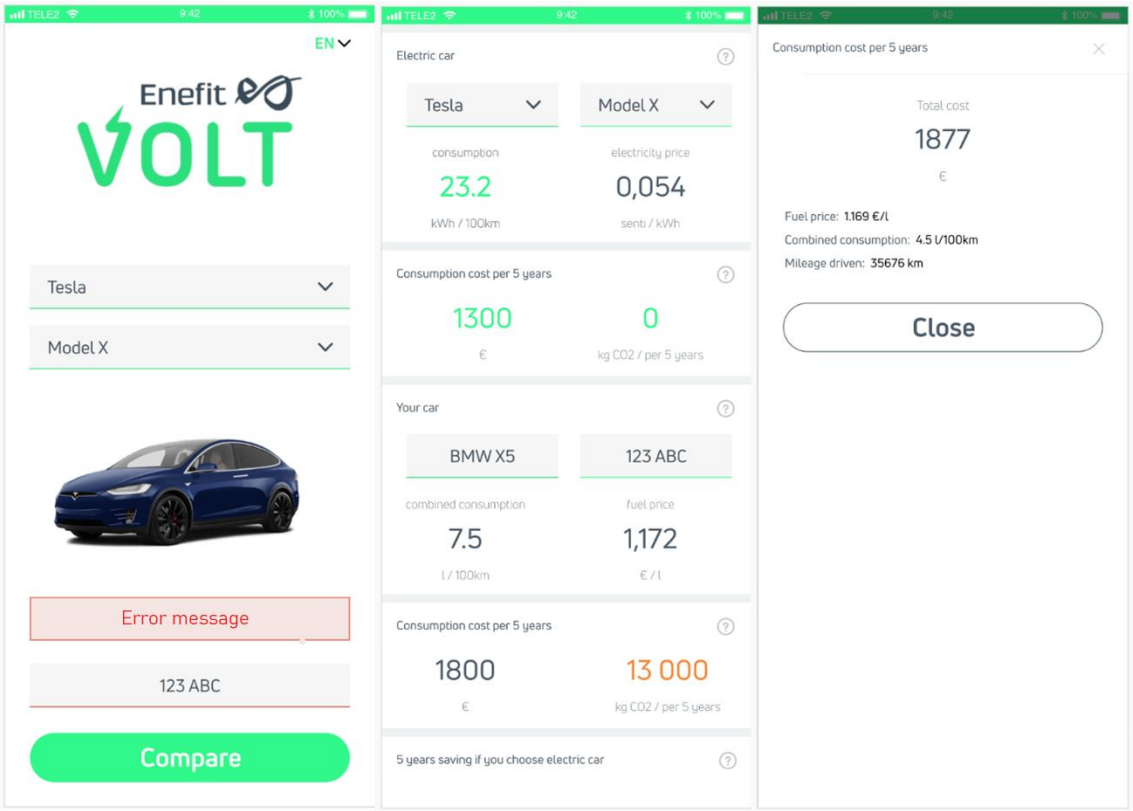
```
protected void setAdapterView(Spinner spinner, int position,
AdapterView<?> parent) {
    if (spinner.equals(spinnerEVBrand)) {
        setEVBrandSpinnerAdapterView(position, parent);
    }
    if (spinner.equals(spinnerEVModel)) {
        setEVModelSpinnerAdapterView(position, parent);
    }
}
```

Joonis 29. *setAdapterView*() meetod

SetAdapterView() meetodis valitakse vastavalt sisendile järgnev meetod, mis seadistab kindla rippmenüü adapteri.

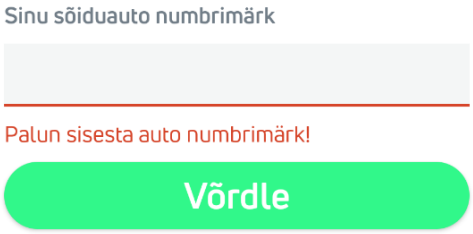
3.3 Disain

Üks projekti nõuetest ja eesmärkidest oli kasutajaliidese ühtlustamine Enefit Volt brändi stiiliga. Meeskonnaliikmed koostasid esialgse prototüübi oma nägemuse järgi ning esitlesid seda ettevõttepoolsetele juhendajatele ja äriosakonna esindajatele. Saadud tagasiside põhjal tehti prototüübis parandusi ning kaasati ettevõttepoolne disainiosakonna esindaja Evgeni Nikolaevski, kes aitas kujundada lõpliku kasutajaliidese prototüübi (Joonis 30).



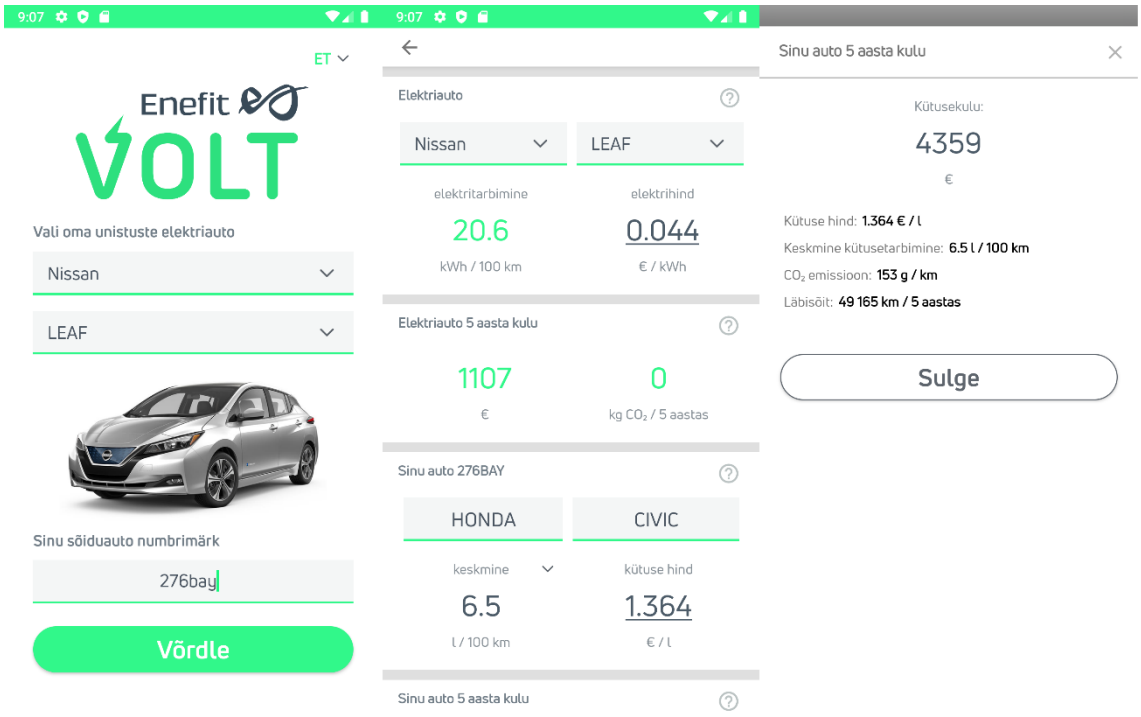
Joonis 30. Prototüüp

Prototüübi osas oli meeskonnal luba teha lõplikus rakenduses mõningaid muudatusi, tingimusel, et need on äripoole esindajatega kooskõlastatud. Näiteks otsustati, et sõiduauto registreerimisnumbri sisestamisel tekkinud veateade kuvatakse tekstilahtri all, mitte üleval (Joonis 31).



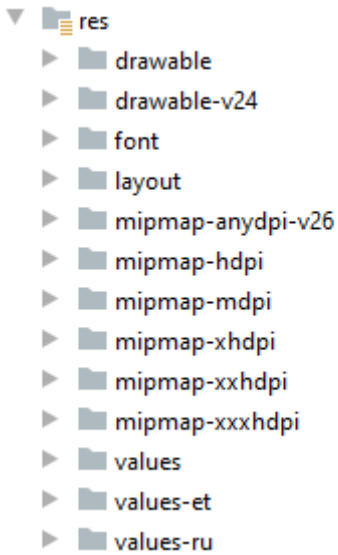
Joonis 31. Veateade

Vastavalt prototüübile ning meeskonnaliikmete ideedele valmis lõplik kasutajaliides (Joonis 32).



Joonis 32. Prototüübile vastav rakendus eesti keeles

Rakenduse kasutajaliidese kujundamine toimus rakenduse *resource* kausta kuuluvate erinevate .xml formaadis failide abil (Joonis 33).



Joonis 33. Resource kaust

Drawable kaust sisaldab kasutajaliideses kasutatavate ikoonide ja erinevate komponentide kujunduse .xml faile. Näiteks määratakse failis *ic_dropdown_icon.xml* ära kõikide rakenduses olevate rippmenüüde nooleikooni kujundus. Failis *dropdown_background.xml* määratakse ära elektriauto margi ning mudeli rippmenüü

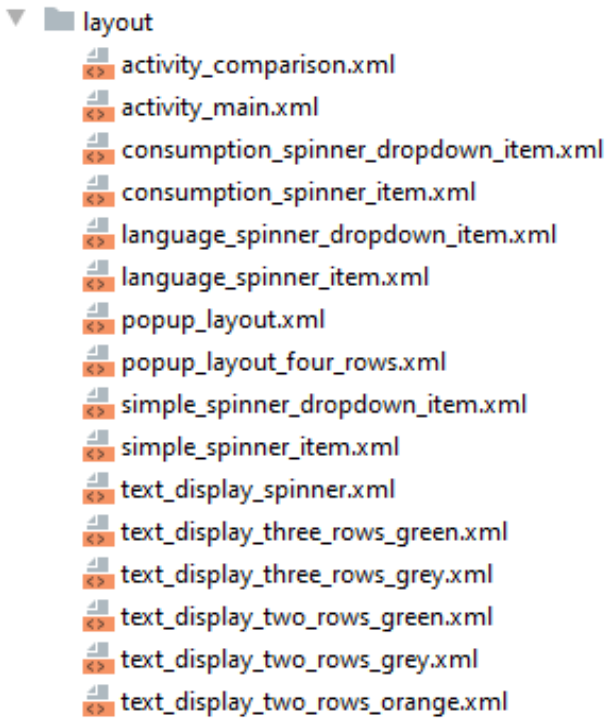
tausta kujundus. Ikoonide ja komponentide kujundus loodi nii prototüübi kui ka Enefit Volt kodulehe analüüsimise põhjal. Lisaks sisaldab antud kaust elektriautode vastavate mudelite fotosid, mida avalehel näidatakse.

Kaustas *font* on .otf formaadis tekstifontide failid. Vastavalt prototüübile ning Enefit Volt ametlikule CSS failile on rakenduses kasutusel font Uni Neue.

Values, *values-et* ning *values-ru* kaustad sisaldavad vastavalt inglise-, eesti- ja venekeelses tõlkes kõiki rakenduses kuvatavaid sõnu, lauseid ning ühikuid. Koodikorduse vältimiseks on *values* kausta loodud spetsiaalne fail *default.xml*, mis sisaldab mittetõlgitavaid vajalikke *String* muutujaid nagu keelekoodid ja erinevad URL aadressid.

Colors.xml fail sisaldab värvide HEX koode, mis leiti vastavalt prototüübis kasutatud värvidele. Kasutajaliidese komponentide (sektsioonide pealkirjad, arvandmed, arvude ühikud, avalehe tekstiväljad jms) ühine stiil määrati nii prototüübi kui ka Enefit Volt kodulehe analüüsimise põhjal failis *styles.xml*.

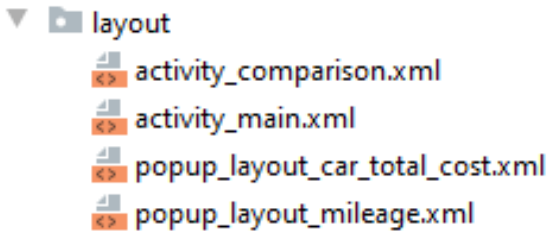
Kausta *layout* loodi kasutajaliidese loomiseks vajaminevate komponentide ja rakenduse vaadete paigutuste .xml failid (Joonis 34).



Joonis 34. Paigutuse .xml failid

Activity_main.xml ja *activity_comparison.xml* failidega määratakse kasutajaliidese kahe vaate (avalehe ning võrdluse vaate) üldine paigutus ja disain. Mõlema faili sees kasutatakse *include* klausliga komponente abifailidest, et tagada puhas kood ning vältida koodikordust.

Projekti esimese etapi, meeskonnaprojekti õppeaine lõpuks sisaldas *layout* kaust vaid 4 faili (Joonis 35). See tähendab, et kogu kasutajaliidese disain sisaldus peamiselt 2 pikas failis. Bakalaureusetöö jooksul jagati kasutajaliidese erinevad komponendid taaskasutatavatesse failidesse. Kui kasutajaliidest oleks vaja täiendada, peaks muutma ainult konkreetset komponenti, mitte avalehe ja võrdluse vaate faili mitmes kohas.



Joonis 35. *Layout* kaust meeskonnaprojekti aine lõpus

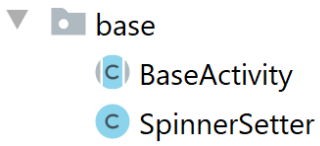
Kõigi kasutajaliidese komponentide väärtustamine konkreetsete andmetega toimub *main* ja *comparison* moodulites Java keeles kirjutatud klassides.

3.4 Kood

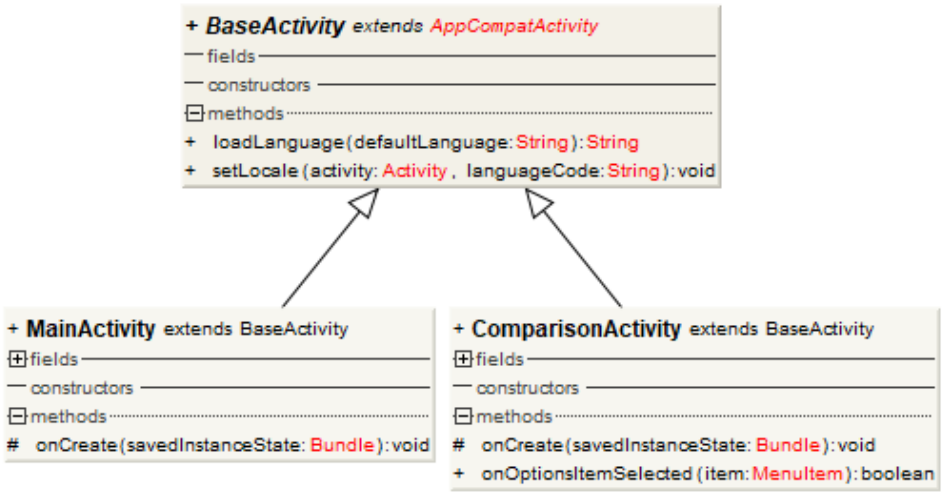
Rakenduse kood jaotub kaheks – *front-end* ja *back-end*. *Front-end* on kirjutatud Java ning XML arenduskeeltes, *back-end* Java arenduskeeles. Koodi puhtuse tagamiseks on kasutatud PMD tarkvara ja Spotless pluginat.

3.4.1 Front-end

Moodulisse *base* loodi abstraktne baasvaate klass (*BaseActivity*), mis sisaldab meetodeid rakenduse keele muutmiseks (Joonis 36). Antud klass laiendab *AppCompatActivity* klassi, mida kasutatakse tegevuste puhul, kus soovitakse kasutada uuemaid Androidi funktsioone vanemate Android seadmetega [9]. Koodikorduse vältimiseks pärivad nii *MainActivity* kui ka *ComparisonActivity* klassid *BaseActivity* klassi, sest mõlemas vaates kasutatakse keele seadistamise meetodeid (Joonis 37).



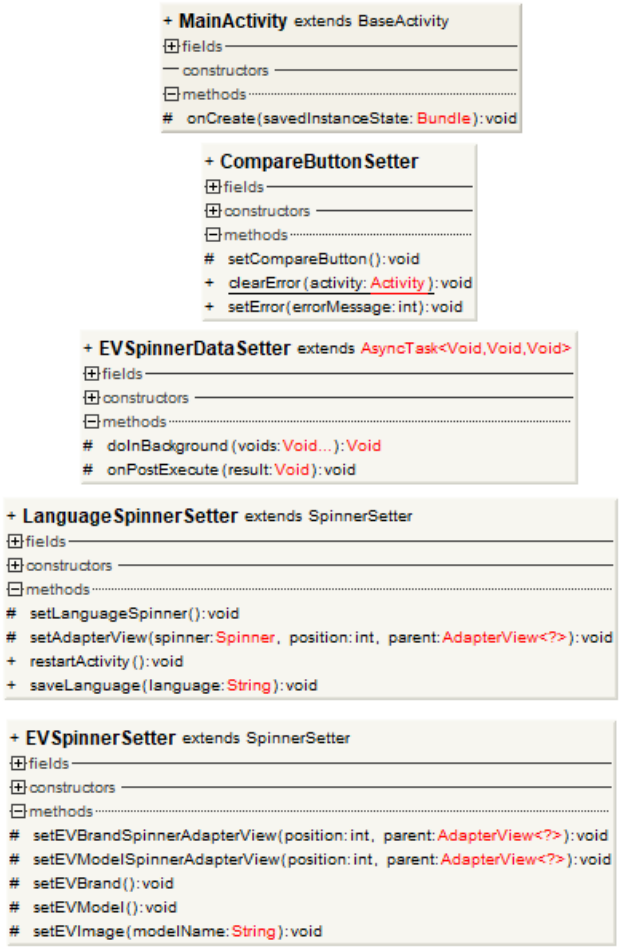
Joonis 36. *Base* moodul



Joonis 37. *BaseActivity* alamklasside klassidiagramm

Projekti esimeses etapis koosnes *main* moodul ühest *MainActivity* klassist, mis sisaldas kogu avalehe vaate seadistamise loogikat. Klassi lähtekood enne refaktoorimist oli 131 rida, mille tulemusena vähendati klass 40 reale. Protsessi käigus tõsteti rippmenüüde seadistamise ja võrdluse lehele liikumise loogika eraldi klassidesse.

Avalehe vaate loomisel kutsutakse klassis *MainActivity* välja *onCreate()* meetod. Meetodis määratakse rakendusele vaikimisi keel ja kutsutakse välja klassid *EVSpinnerDataSetter*, *LanguageSpinnerSetter* ja *CompareButtonSetter* (Joonis 38).



Joonis 38. *Main* mooduli klassidiagramm

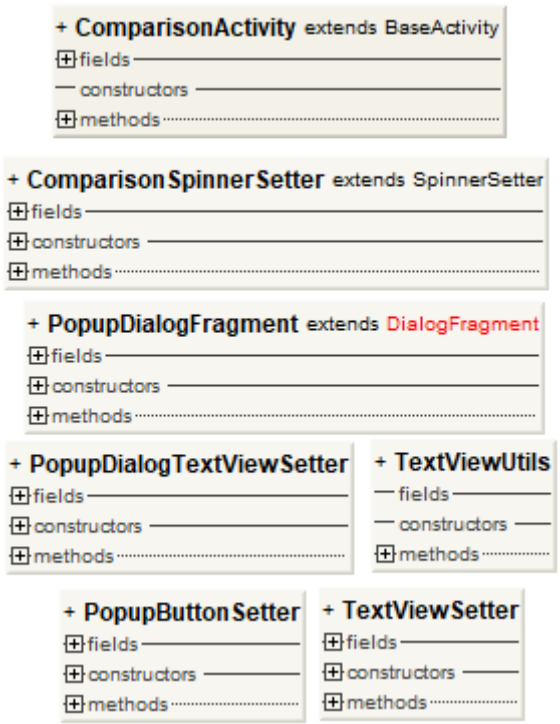
Rippmenüüde loomiseks kasutatakse Androidile omaseid *spinner* komponente. Rippmenüüde seadistamiseks vajalik üldine baasklass on *SpinnerSetter*, mis asub *base* moodulis. Keele, elektriauto margi ning elektriauto mudeli rippmenüüde seadistamiseks kirjutatakse *SpinnerSetter* meetodid üle klassides *EVSpinnerSetter* ja *LanguageSpinnerSetter*, kasutades *@Override* annotatsiooni.

EVSpinnerDataSetter on asünkroonne klass, kus päritakse *back-end*'ilt elektriautode andmed ning seejärel kutsutakse välja *EVSpinnerSetter* klass, mis seadistab elektriautode rippmenüüd eelnevalt päritud elektriautode markide ja mudelitega. Lisaks kutsutakse nimetatud klassis välja *setOnItemSelectedListener()* meetod, kus asub õige elektriauto pildi kuvamise ja margile vastava mudeli rippmenüü uuendamise loogika.

LanguageSpinnerSetter klassis täidetakse keelevaliku rippmenüü. Kui kasutaja soovib muuta rakenduse keelt, kutsutakse välja *setOnItemSelectedListener()* meetod.

Klassis *CompareButtonSetter* asub loogika avalehelt võrdluse lehele liikumiseks. „Võrdle“ nupule vajutades kutsutakse *startActivity()* meetod, millele antakse parameetrina kaasa Androidile omane *Intent*. *Intent*'i abil saadetakse avalehelt (*MainActivity*) andmed võrdluse lehele (*ComparisonActivity*). Vajalikud andmed on: numbrimärgile vastav *Car* objekt, sellele vastav kütuse hind ning valitud elektriauto. *Car* objekt saadakse *CarViewModel* klassi *checkCache()* meetodi abil andmebaasist. Kütuse hinna saamiseks pöördutakse *FuelViewModel* klassi poole, arvestades *Car* objekti kütusetüüpi.

Võrdluse lehe vaate loomine toimub moodulis *comparison*. Projekti esimeses etapis koosnes moodul ühest *ComparisonActivity* klassist, mille pikkus oli 607 rida. Peale refaktoorimist jäi klassi pikkuseks 49 rida. Protsessi käigus tõsteti võrdluse lehe vaate loogika kolme tüüpi klassidesse: tekstivaadete väärtustamise klassid, hüpikakende seadistamise klassid ning rippmenüüde seadistamise klass (Joonis 39).



Joonis 39. *Comparison* mooduli klassidiagramm

Sarnaselt *MainActivity* klassile sisaldab *ComparisonActivity* klass *onCreate()* meetodit, milles seadistatakse vaatele õige keel ja kutsutakse välja järgnevad klassid: *ComparisonSpinnerSetter*, *TextViewSetter* ning *PopupButtonSetter*.

ComparisonSpinnerSetter klassis seadistatakse võrdluse lehel kasutatavad elektriauto margi ja mudeli ning sõiduauto kütusetarbimise valiku rippmenüüd.

TextViewSetter klassis väärtustatakse vastavalt eelnevalt tehtud valikule kõik võrdluse lehel olevad tekstiväljad. Tekstiväljade seadistamiseks kasutatakse klassi *TextViewUtils* abimeetodeid.

Rakenduses on neli hüpikakent, kus kuvatakse kasutajale lisainformatsiooni. Hüpikakende seadistamiseks kasutatakse klasse *PopupDialogTextViewSetter*, *PopupDialogFragment* ja *PopupButtonSetter*.

PopupButtonSetter klassis toimub hüpikakna ikoonile vajutamisele reageerimine ehk *PopupDialogFragment* klassi välja kutsumine. Kõik hüpikaknad ehitatakse samal põhimõttel, seega loodi koodikorduse vähendamiseks *PopupDialogFragment* klass. Tegemist on *DialogFragment* tüüpi klassiga, mis sisaldab *onCreateDialog()* meetodit, millega määratakse hüpikakende ehitamise loogika. *PopupDialogTextViewSetter* klassis väärtustatakse kõigi hüpikakende tekstiväljad, kasutades *TextViewUtils* klassi abimeetodeid.

3.4.2 Back-end

Esimeses etapis loodi rakenduse *back-end*'i andmebaas, API ühenduste klassid ja üldine äriloogika, mida täiendati projekti teises etapis vastavalt nõuetele. *Back-end*'i loodi klass *Calculations*, mis sisaldab meetodeid võrdluse jaoks vajalike arvutuste tegemiseks. Projekti esimeses etapis arvutati sõidukite kütusekulud kasutaja sõiduauto viimase kahe tehnoülevaatuse vahelise aja põhjal. Teises etapis muudeti sõidukite kütusekulude arvutamise loogika 5 aasta perioodi peale. Lisaks arvutati sõidukite CO_2 heite kogused, elektriauto sõiduulatus ja täpne elektriauto elektrikulu, võttes arvesse võrguteenuse tasusid.

Kõikide *Calculations* klassi meetodite tagastustüüp on *BigDecimal*, mis tagastab tulemused murdarvu täpsusega, et vältida arvutustes tekkivaid vigu. Meetodis *getAnnualMileage()* leitakse kasutaja sõiduauto keskmine aastane läbisõit (Joonis 40).

Projekti esimeses etapis tehtud arvutused võtsid aluseks odomeetri näidud sõiduki kahe viimase ülevaatuse ajal, mis võisid olla ka samal päeval tehtud. Täpsema tulemuse saamiseks salvestati projekti teises etapis ülevaatuste andmed võimalikult pika perioodi kohta ehk arvutuste aluseks võeti esimene ning viimane olemasolev odomeetri näit.

```
public BigDecimal getAnnualMileage() {
    Time latest = AppUtils.getTimeFromString(latestInspection);
    Time previous = AppUtils.getTimeFromString(previousInspection);
    // long value of the time between two inspections
    long time = latest.getTime() - previous.getTime();
    // days between the two inspections
    BigDecimal days = BigDecimal.valueOf(time / (1000 * 60 * 60 *
24));
    // mileage per year
    return BigDecimal.valueOf(getDistance())
            .divide(days, 4, RoundingMode.HALF_UP)
            .multiply(new BigDecimal(365))
            .setScale(0, RoundingMode.HALF_UP);
}
```

Joonis 40. Meetod keskmise aastase läbisõidu leidmiseks

Saadud aastane läbisõit korrutatakse viiega ning leitud 5 aasta läbisõit võetakse aluseks nii elektriauto kui ka kasutaja sõiduauto kütusekulude ja CO_2 heite koguse arvutamiseks.

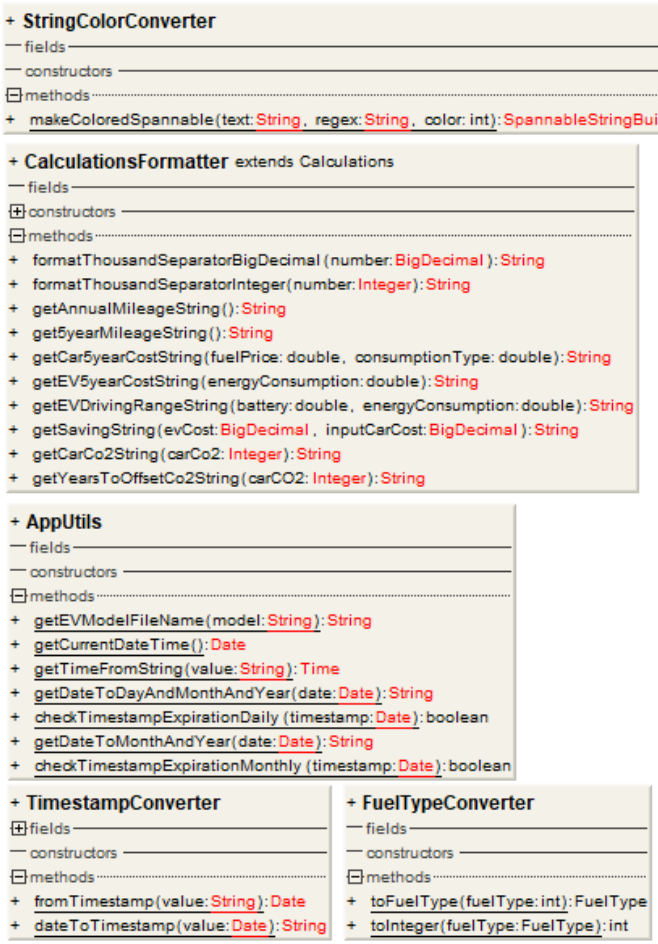
Kasutaja sõiduauto kütusekulu leitakse *getCar5YearCost()* meetodi abil, võttes arvesse läbisõitu, kütusekulu 100 km kohta vastavalt kütusetarbimise tüübile (linnas, maanteel või kombineeritud) ja kütuse hinda.

Lisaks arvutatakse sõiduauto puhul välja tema CO_2 heite kogus antud perioodil. Väärtus leitakse *getCarCo2()* meetodiga, võttes arvesse sõiduauto CO_2 heite kogust grammides kilomeetri kohta ja läbisõitu. Leitud väärtuse põhjal arvutatakse meetodiga *getYearsToOffsetCo2()*, mitu aastat kulub 10 puul sõiduauto poolt eraldatud süsihappegaasi taastamiseks.

Elektriauto kütusekulu summa leidmiseks võetakse aluseks kasutaja sõiduauto läbisõit, elektritarbimine 100 km kohta ning elektrihind, millele lisanduvad elektri võrgutasud ja igakuised maksed. Tarbitud elektri maksumus leitakse *getFuelCost()* meetodi abil, võttes arvesse eelmise kuu elektrihinda ja elektriauto elektritarbimist. Võrgutasud leitakse *getGridServiceFees()* meetodi abil, mis arvutab võrguteenuste maksumuse, võttes arvesse Elektrilevi võrgutasu ning kuumakse. Meetodiga *getEV5yearCost()* leitakse elektriauto kütusekulu ehk nende väärtuste summa.

Elektriauto puhul arvutatakse välja veel tema sõiduulatus laadimiste vahel *getDrivingRange()* meetodiga. Väärtus leitakse elektriauto aku mahutavuse ning elektritarbimise põhjal.

Abimoodulina on loodud *utils* kaust, mis sisaldab kogu rakenduses kasutatavaid abiklasse (Joonis 41).



Joonis 41. *Utils* kausta klassidiagramm

AppUtils on klass, mis sisaldab staatilisi meetodeid, mida saab kasutada kogu rakenduse ulatuses. Meetodid selles klassis on staatilised, et hoida kokku mälukasutust, kuna klass kutsutakse välja ühe korra rakenduse laadimisel. Staatilisi meetodeid saab kasutada klassi objekti ennast välja kutsumata tingimusel, et meetodid ei kasuta muutuvaid välju. Näiteks on kasutusel meetod *getTimeFromString()*, mis tagastab tekstiväärtusest kuupäeva *Time* tüüpi väärtuse.

CalculationsFormatter klass on loodud *Calculations* klassi laiendusena, et tagastada *front-end*'ile arvutuste väärtused tekstiväärtustena ning kuvada tuhandelisi väärtusi õiges vormingus *formatThousandSeparatorInteger()* meetodi abil.

FuelTypeConverter klassi kasutatakse kütusetüübi teisendamiseks, *StringColorConverter* klassi teksti värvi muutmiseks ja *TimeStampConverter* klass sisaldab meetodeid ajaväärtuse teisendamiseks.

3.5 Testid

Meeskond kasutas rakenduse töökindluse tagamiseks dünaamilise testimise meetodeid. Lokaalsete ühiktestidega testiti programmi teksti põhjal (valge kasti testimine). Androidi komponente kasutavate testidega (*instrumented tests*) rakendati funktsionaalset testimist (musta kasti testimine). Pärast iga funktsionaalsuse lisamist testiti rakendus üle ka manuaalselt.

Lokaalsete ühiktestidega kontrolliti rakenduse klasse, mis ei sõltu Androidi-spetsiifilisest loogikast. Ühiktestitud on järgnevad funktsioonid: kõik vajaminevad arvutused, tekstivaadete ja hüpikakende seadistamine, ühenduse loomine API-dega ning elektriauto, sõiduauto, kütuse ja elektri Java objektid. Lisaks testiti erinevate abiklasside loogikat: kuupäevade vormindamine *Date* ja *String* tüüpi, sõiduauto kütusetüübi vormindamine *Integer* ja *Enum* tüübi vahel, elektriauto ja kasutaja sõiduautoga seotud rippmenüüde seadistamine, *String* tekstile värvi lisamine UI jaoks.

```
@Test
public void testBigDecimalFormatter() {
    String expected = "10 000";
    assertEquals(
            expected,
calcFormatter.formatThousandSeparatorBigDecimal(new
BigDecimal(10000)));
}

@Test
public void testBigDecimalFormatterSmallNumber() {
    String expected = "9999";
    assertEquals(
            expected,
calcFormatter.formatThousandSeparatorBigDecimal(new
BigDecimal(9999)));
}
```

Joonis 42. *BigDecimal* tüüpi arvude vormindamise ühiktestid

```
@Test
public void testSetMileagePreviousTextView() {
    TextView textView =
popupTextViewSetter.setMileagePrevious();
    assertEquals("12 300 km", textView.getText());
}

@Test
public void testSetMileageLatestTextView() {
    TextView textView =
popupTextViewSetter.setMileageLatest();
    assertEquals("15 600 km", textView.getText());
}
```

Joonis 43. Tehnoülevaatuste läbisõidu tekstivaate ühiktestid

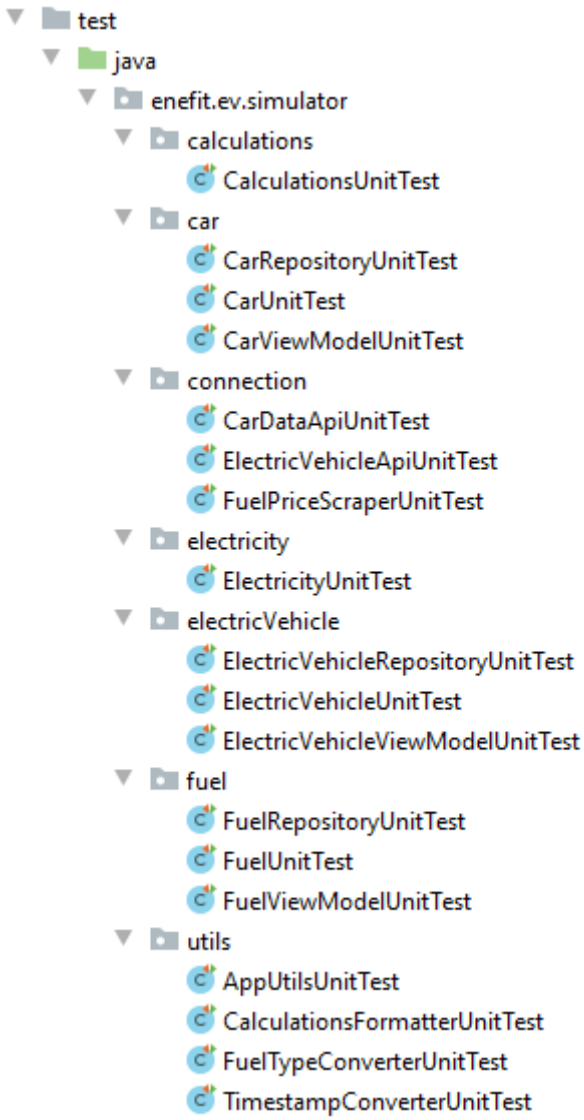
```
@Test
public void testSetCostForCarCombinedConsumption() {
    spinner.setSelection(0);
    TextView textView = textViewSetter.setCostForCar(spinner);
    assertEquals("1307", textView.getText());
}

@Test
public void testSetCostForCarExtraUrbanConsumption() {
    spinner.setSelection(1);
    TextView textView = textViewSetter.setCostForCar(spinner);
    assertEquals("1106", textView.getText());
}

@Test
public void testSetCostForCarUrbanConsumption() {
    spinner.setSelection(2);
    TextView textView = textViewSetter.setCostForCar(spinner);
    assertEquals("1508", textView.getText());
}
```

Joonis 44. Erinevat tüüpi kütusekulude ühiktestid

Lokaalsete ühiktestide klassid on sarnaselt üldisele rakendusele jagatud erinevatesse moodulitesse, et hoida sarnaste funktsioonide testimine ühes asukohas. *Calculations* moodulis testitakse kõiki arvutusi ning võimalikke veakohti. *Connection* moodulis kontrollitakse ühendust Maanteeameti, Enefit Volt API-dega ja *data scraping* tehnikat. *Car*, *electricity*, *electricVehicle* ja *fuel* moodulites testitakse vastavate objektidega seotud meetodeid. *Utils* moodulis testitakse erinevate abiklasside meetodeid (Joonis 45).



Joonis 45. Ühiktestide klassid

Androidi komponente kasutavate testide hulka (*instrumented tests*) kuuluvad kasutajaliidese ja integratsioonitestid.

Kasutajaliidese testimine võimaldab tagada rakenduse vastavuse äripoole nõuetele. Kasutajaliidese testide abil on võimalik kontrollida, kas rakendus käitub ootuspäraselt, kui kasutaja sooritab mõne toimingu või sisestab sisendi.

Integratsioonitestidega kontrollitakse, et rakenduse kahe *Activity* vahel on toimiv andmevahetus ning et andmete salvestamine ja pärimine andmebaasist toimib.

```
@Test
public void testEVModelSelectionView() {
    onView(withId(R.id.spinner_ev_brand)).perform(click());
    onData(allOf(is(instanceOf(String.class)),
is(ev.getBrand()))).perform(click());
    onView(withId(R.id.spinner_ev_model)).perform(click());
    onData(allOf(is(instanceOf(String.class)),
is(ev.getModel()))).perform(click());
    onView(withId(R.id.spinner_ev_model))
    .check(matches(withSpinnerText(containsString("LEAF"))));
}
```

Joonis 46. Elektriauto mudeli valiku UI test

```
@Test
public void testElectricityPriceView() {
    tableLayout =
comparisonActivity.findViewById(R.id.electricity_price);
    real = tableLayout.findViewById(R.id.value);
    assertEquals(
   String.valueOf(calcFormatter.getElectricityPriceRounded()),
            real.getText().toString());
}
```

Joonis 47. Elektrihinna tekstivaate UI test

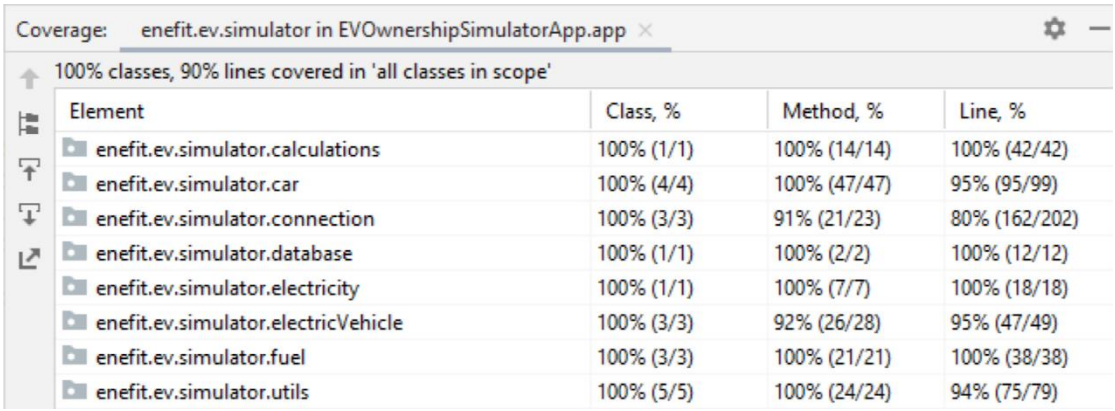
```
@Test
public void testCallToActionButtonOpensEnefitVolt() {
    String url = context.getString(R.string.enefit_volt_url);
    Matcher<Intent> expectedIntent =
allOf(hasAction(Intent.ACTION_VIEW), hasData(url));
    intending(expectedIntent)
            .respondWith(new Instrumentation.ActivityResult(0,
null));
    onView(withId(R.id.call_to_action_button))
            .perform(ViewActions.scrollTo())
            .perform(click());
    intended(expectedIntent);
}
```

Joonis 48. Nupule vajutuse UI test

```
@Test
public void testEVBlockTitlesAreGrey() {
    onView(withId(R.id.ev_caption))
.check(matches(hasTextColor(R.color.enefitGreyBlockTitles)));
    onView(withId(R.id.ev_cost_caption))
.check(matches(hasTextColor(R.color.enefitGreyBlockTitles)));
}
```

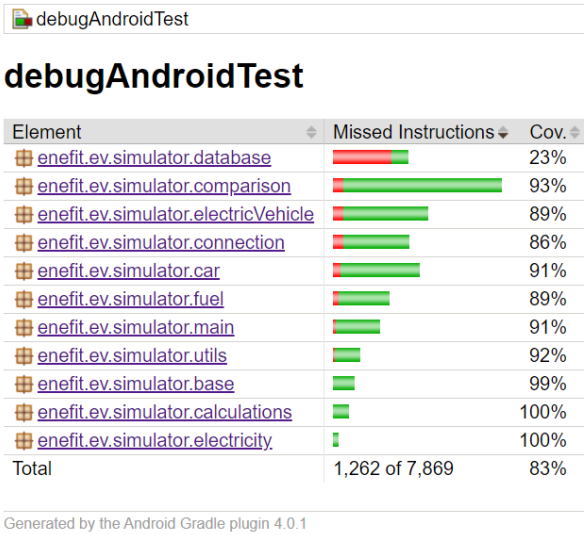
Joonis 49. Pealkirjade värvi UI test

Lokaalsete ühiktestide koodi kattuvus on 90% ridadest ja 98% meetoditest.



Joonis 50. Lokaalsete ühiktestide koodi kattuvus

Androidi komponente kasutavate testide koodi kattuvus on 83% ridadest.



Joonis 51. *Instrumented* testide koodi kattuvus

Kokku on rakenduses 305 testi. Neist 189 on lokaalsed ühiktestid ning 116 on *instrumented* testid.

Lisaks koodis kirjutatud testidele kontrolliti rakenduse töötamist ning võimalikke veakohti ka manuaalselt. Näiteks otsiti Auto24 veebilehelt erinevate sõiduautode numbrimärgid ja kontrolliti brauseris Maanteeameti API-st saadud tulemust. Erinevate tüüpjuhtude kontrollimiseks otsiti numbrimärgid, millel puuduvad kõik, mõned või üks järgnevatest andmetest: kütusetarbimine linnas, kütusetarbimine maanteel, kütusetarbimine kombineeritult, tehnoülevaatuste odomeetri näidud, CO_2 emissioon. Seejärel veenduti, et nende numbrimärkide puhul töötab rakendus ootuspäraselt või annab õige veateate.

4 Analüüs ja järeldused

Järgnevalt põhjendatakse töö tulemuste tehnilist teostust. Lisaks tuuakse välja teaduskirjandus, millele rakenduse loomisel toetuti. Eelviimases osas näidatakse tööks kulunud tundide logisid ja nädalaplaani kõigi liikmete kohta. Viimasena antakse hinnang projekti valmimise protsessile.

4.1 Tehnilise teostuse põhjendus

Järgnevalt kirjeldatakse, miks kasutati töös vastavaid nõudeid, arhitektuuri, kasutajaliidese kujundust, koodi kirjutamise põhimõtteid ning testide tüüpe.

4.1.1 Nõuded

Nõuete sätestamisel analüüsiti neid nii meeskonnasiseselt kui ka tooteomaniku ning juhendajatega. Sel viisil arvestati kõigi ideedega, äripoole nõuete, ajalise ressursi ja tehnilise teostusega. Nõuete koostamisel pöörati rõhku peamiselt neljale aspektile.

Esimeseks aspektiks oli, et rakenduse disain vastaks Enefit Volt disainile, eeldusel, et see lisatakse Enefit Volt mobiilirakendusse. Nõuete koostamisel kasutati ettevõtte disainiosakonna esindaja, Evgeni Nikolaevski, loodud kasutajaliidese prototüüpi.

Teiseks aspektiks oli tõsta kasutajate huvi rakenduse vastu. Selleks tuli laiendada sihtgruppi ning lisada kasutajatele rakenduse kasutamisel väärtust. CO_2 emissiooni funktsionaalsuse lisamisel saab kasutaja aimu sisepõlemismootoriga auto keskkonnasaastest. Kui algselt oli rakendus kasutatav inglise keele valdajatele, siis teise etapi jooksul lisati rakendusse eesti ja vene keel.

Kolmandaks aspektiks oli koodi kvaliteedi kontrollimine. Arendajatel oli kohustus kirjutatud kood testidega katta, et võimalikud veakohad ning piirjuhud tuvastada. Testide automatiseerimiseks kasutati CI/CD *pipeline*'i.

Neljandaks aspektiks oli tagada koodi korduvkasutatavus, et seda oleks lihtne integreerida veebirakendusse, eeldusel, et tulevikus võib äripoolel tekkida soov rakendust kasutada veebilehe komponendina. Selleks kasutati vastavaid arhitektuuri- ja disainimustreid.

4.1.2 Arhitektuur

Projekti arenduse esimeses etapis (meeskonnaprojekti õppeaine raames) kaaluti rakenduse arhitektuuri loomisel MVVM (*Model-View-ViewModel*), MVP (*Model-View-Presenter*) ning MVC (*Model-View-Controller*) mustreid.

Android rakenduse arendamiseks on Androidi ametlikes arhitektuurikomponentides määratud *ViewModel* tüüpi klass. See võimaldab edastada andmebaasist päritud andmeid kasutajaliidesega seotud *View* tüüpi klassidele. MVVM arhitektuurimustrit soovitatakse ametlikus Android dokumentatsioonis [9]. Antud argumentidele toetudes otsustati koostöös juhendajatega, et rakendus järgib MVVM arhitektuurimustrit.

Rakenduses kasutatakse SQLite andmebaasi andmete vahemälus hoidmiseks. SQLite andmebaas on kiirem andmete lugemisel ja kirjutamisel võrreldes kasutaja telefoni kõvakettaga. Samuti on seda väga lihtne kasutada Androidi keskkonnas.

Teise etapi jooksul refaktooriti kogu projekti koodi ning selle käigus kasutati erinevaid disainimustreid. Neid rakendati, et tagada koodi paindlikkus ning taaskasutatavus. Antud omadused tagavad koodi lihtsa lugemise, hallatavatuse ning muutmise.

Keele muutmisel võeti kasutusele Sõltvusesüsti muster, mille eesmärgiks on lihtsamini kasutada keele konfiguratsiooni. Meeskond otsustas kasutada Androidi liidest *SharedPreferences*, mis hõlbustab keelte XML failide kasutamist.

Pärast esimest etappi koosnes *front-end* kahest klassist: *MainActivity* ja *ComparisonActivity*. Meeskond otsustas antud klasside puhul rakendada Fassaad mustrit, eesmärgiga kasutada ühe vaate jaoks ühte põhiliidest, mis käivitab kõik muud keerulisemad protsessid. Tulemusena muutus koodi keerukus väiksemaks, mida on lihtsam hallata.

Rakenduse esmasel käivitamisel esines probleeme elektriautode importimisel ja rippmenüüde laadimisel. Kasutajale kuvati topelt kirjeid elektriautode mudelite rippmenüüs, sest rippmenüüd loodi enne, kui andmed olid andmebaasi salvestatud. Otsustati kasutada *Command* mustrit, mis tagab selle, et rippmenüüd ei laeta enne, kui import on töö lõpetanud.

Kuna esimeses etapis esines palju koodikordust ning koodi loetavus oli halb, otsustati võtta kasutusele Strateegia muster. Selle realiseerimiseks loodi meetodid, mis kutsuvad sisendi järgi välja vastavad meetodid. Peamiselt kasutati Strateegia mustrit selleks, et vastavalt kasutaja sisendile avada kindel hüpikaken või seadistada rippmenüü.

4.1.3 Disain

Kasutajaliidese kujundamisel lähtuti eelkõige äripoole nõuetest ning kasutajasõbralikkusest. Äripoole nõuded olid, et kasutajaliides oleks kasutajale kergesti hoomatav, kajastaks kõiki arvutustes kasutatud andmeid ning ühtiks Enefit Volt stiiliga. Lisaks oli oluline, et rakenduse kasutajaliides koosneks komponentidest, mida oleks arendajatel edaspidi võimalik kerge vaevaga täiendada. Kõik äripoole nõuded kasutajaliidesele said projekti teises etapis täidetud.

Võrdluse leht otsustati jagada sektsioonideks, et tagada andmete parem eristatavus ja et kasutajale oleks arusaadav, mille kohta konkreetsed andmed käivad. Oluliste arvandmete esile tõstmiseks otsustati kasutada suuremat teksti suurust ning erinevaid värve. Näiteks on olulised andmed (nt elektriauto kütusekulu, elektriauto 5 aasta kütusekulu ja CO_2 kogus) rakenduses kuvatud Enefit Volt brändile omase rohelise värviga, et rõhutada elektriautode keskkonnasäästlikkust. Vaikeväärtuste eristamiseks kasutatakse kasutajaliideses tärne.

Kuna äripoole üks nõuetest oli, et kasutaja peab saama veenduda andmete ja arvutuste õigsuses, tuli kasutajaliidese disainimisel mõelda, kuidas suur hulk andmeid paigutada nii, et kasutajaliides jääks võimalikult puhtaks ja selgelt hoomatavaks. Seetõttu otsustati lisainformatsiooni näitamiseks kasutada hüpikaknaid, et peamine informatsioon ei kaoks muu info seas ära. Lisaks otsustati nii elektri- kui ka kütusehinnale külge lisada link, mis viib kasutaja lehele, kust vastavad andmed saadakse. Lingiga väärtused on rakenduses allajoonitud kujul, et kasutaja mõistaks, et tegemist on lingiga.

Meeskonnaprojekti raames valminud rakenduses oli elektriauto valimise võimalus ainult võrdluse lehel. Projekti teises etapis lisati elektriauto valimise võimalus ka avalehele. Selle abil on kasutajal võimalik kohe rakenduse avanedes valida meelepärane elektriauto. Lisaks kuvatakse iga elektriauto juurde selle pilt, et valimist kasutaja jaoks lihtsustada.

4.1.4 Kood

Koodi kirjutamisel on järgitud *clean code* [10] põhimõtteid. Valminud kood on võimalikult lühike ja lihtne. See võimaldab koodi juurde tagasi tulles ja muutuste tegemiseks seda lihtsasti lugeda ning aru saada.

Koodi kirjutamisel on välditud andmete või loogika duplikatsiooni, mille tagamiseks on korduvad read asendatud taaskasutatavate meetoditega ja vajadusel kasutatud klasside laiendamist. Koodi kirjutamisel on eraldatud erinevad komponendid.

Abstraktsiooni on valminud koodis kasutatud võimalikult vähe, sest koodi kirjutati kindlate funktsionaalsuste jaoks, mida pole plaanis kasutada laiemalt teistes projektides. Sellega saavutatakse koodi hea loetavus, mida liigne abstraktsioon võib keeruliseks muuta.

Keerulisemate meetodite juures on kasutatud dokumenteerimist ehk kirjeldavaid kommentaare selgitamiseks. Valminud kood on jooksvalt refaktooritud, et parandada koodi lihtsust ning loetavust.

Kogu arendamise protsessil on järgitud ühtset stiili. Moodulid kutsutakse pakettidest välja täpsete nimedega, et vältida üleliigsete moodulite laadimist. Jälgitud on, et kõik meetodid täidaksid vaid ühte käsku. Klassi väljad (*fields)* on defineeritud privaatselt iga klassifaili alguses ning lokaalsed muutujad (*variables)* vastava kasutusala raamides (meetod või klass).

Nimetamisel on silmas peetud Java programmeerimiskeeles kasutatavaid põhimõtteid: klassid on nimisõnad, meetodid algavad tegusõnaga ning väljad ja muutujad on lühikese nimega. Välditud on ühetähelisi muutujaid ning kõik nimetused viitavad nende kasutuse eesmärgile. Klassidel on kasutatud *UpperCamelCase* vormingut ehk algavad suure tähega. Meetoditel ning muutujatel *lowerCamelCase* vormingut ehk algavad väikese tähega ning sõnaühendite puhul kõik järgnevad sõnad suure tähega [11].

Koodi lihtsaks formaatimiseks kasutatakse Spotless pluginat. Plugin jooksutatakse käsuga *gradlew spotlessApply* lokaalselt Android Studios. *Commit*'i üles laadimisega GitHubi jookseb käsk ka *CI pipeline*'s, mis aitab puhta koodi hoidmist automatiseerida. Sellega eemaldatakse automaatselt ebavajalikud impordid, üleliigsed tühikud koodist, jälgitakse, et ridade taandamisel on kasutatud tühikuid ja iga fail lõppeb uue tühja reaga.

Lisaks kasutati koodi puhtuse jälgimiseks PMD-d, mis on lähtekoodi analüsaator (analüüsiv tehnika). Antud tarkvaras kasutatakse mitmeid reeglite kategooriaid [12], millest on rakenduses kasutusel järgnevad:

- Reeglid katkiste, segaste või koodi jooksutamisel vigadele kalduvate konstruktorite kontrollimiseks. Näiteks *EmptyTryBlock* kontrollib, et koodis poleks tühjasid *try* plokke, kus viga ei saa tekkida.

- Reeglid, mis viitavad probleemidele, kus käsitletakse mitut lõime (*thread*). Näiteks *AvoidThreadGroup* väldib *java.lang.ThreadGroup* klassi kasutamist, kuna see sisaldab meetodeid, mis pole niiditurvalised.

- Reeglid, mis aitavad kasutada üldtunnustatud parimaid tavasid. Näiteks *AvoidUsingHardCodedIP* aitab vältida *hard-coded* IP aadressite kasutamist, kuna see raskendab rakenduse juurutamist.

- Reeglid, mis aitavad kasutada spetsiifilist kodeerimisstiili. Näiteks *UselessParantheses* väldib ebavajalike sulgude kasutamist.

- Reeglid, mis tähistavad mitteoptimaalset koodi. Näiteks *StringToString* väldib juba tekstiväärtusena olevale objektile *toString()* meetodi kutsumist.

- Reeglid, mis aitavad avastada koodi disaini vigu. Näiteks *CollapsibleIfStatements* aitab leida koodist järjestikused *if*-laused, mida on võimalik asendada *Boolean* operaatoriga. Sealhulgas kontrollitakse importide arvu ja meetodite pikkust.

Koodi failide meetrika on välja toodud järgneval joonisel (Joonis 52).

Extension	Count	Lines CODE	Lines	Lines AVG	Lines MIN	Lines MAX	Size AVG
bat *(BAT files)*	1x	61	84	84	84	84	2kB
gradle *(GRADLE files)*	2x	13	14	7	2	12	0kB
java *(Java classes)*	71x	5926	7033	99	17	814	3kB
json *(JSON files)*	1x	251	251	251	251	251	7kB
otf *(OTF files)*	14x	23975	25301	1807	1627	1941	88kB
pro *(PRO files)*	1x	0	21	21	21	21	0kB
properties *(Java properties files)*	3x	10	34	11	6	20	0kB
xml *(XML configuration file)*	48x	2002	2247	46	5	371	2kB
Total:	**141x**	**1 612kB**		**94kB**	**158kB**		**106kB**

Joonis 52. Koodi meetrika ülevaade

Java failid sisaldavad front-end ja back-end klasse, milles on 5926 koodirida. JSON failid sisaldavad andmebaasiga seotud faile ning neis on 251 koodirida. OTF failid sisaldavad tekstifonte. XML failid sisaldavad kasutajaliidese komponente ja moodustavad kokku 2002 koodirida.

4.1.5 Testid

Vastavalt Android rakenduse eripärale on hea tava rakendust testida kahte tüüpi testidega: lokaalsed ühiktestid (*local unit tests*) ning Androidi komponente kasutavad testid (*instrumented tests*) [9].

Rakenduse *back-end*'i testiti lokaalsete ühiktestidega, mille abil on võimalik kontrollida rakenduse üldist loogikat, mis ei sõltu Androidi-spetsiifilistest rakenduse osadest nagu *Activity* ja *Fragment* [9]. Ühiktestid valmisid kasutades Robolectric raamistikku, mille abil on ühikteste võimalik käivitada Java virtuaalmasinal ja kasutada Android API osi emulaatorit käivitamata [13].

Androidi komponente kasutavad testid (*instrumented tests*), mis valmisid kasutades Android JUnit 4 raamistikku, sisaldavad integratsiooni ning kasutajaliidese teste.

Testide abil kontrollitakse, kuidas töötavad koos Androidi-spetsiifilised rakenduse osad (nt erinevad *Activity* ja *Fragment* klassid), kasutajaliides ja *back-end*. Teste on võimalik jooksutada arvutiga ühendatud Android telefoni või virtuaalse Android mobiiltelefoni ehk emulaatori peal [9].

Instrumented testide abil kontrollitakse rakenduse esimest vaadet (*MainActivity*) ja autode võrdluse vaadet (*ComparisonActivity*). Kasutajaliidese testimiseks kasutatakse Espresso raamistikku, mille abil saab simuleerida toiminguid kasutajaliideses [9]. Testide abil kontrollitakse, et UI sisaldab õigeid andmeid (k.a. õiges keeles tõlkeid) ja korrektset stiili (värvid, teksti font, küljendus, paigutus jms). Lisaks veendutakse testide abil, et andmed liiguvad kahe vaate vahel korrektselt ning et andmebaasi andmete salvestamine ja nende pärimine toimib.

Testide automaatseks käivitamiseks ning iga meeskonnaliikme töökvaliteedi kontrollimiseks kasutatakse Github Actions tööriista, mis võimaldab iga koodihoidlasse laetud muudatust CI/CD tava järgi automaatselt testida [14]. Github lähtekoodihoidlasse loodi *android-any-branch.yml* fail (Lisa 3), mille ülesanded käivitati CI *pipeline*'i kaudu

iga koodi üles laadimise korral. Selleks, et teised meeskonnaliikmed ning juhendajad saaksid tehtud muudatused heaks kiita, pidid testid läbima *pipeline*'i vigadeta.

4.2 Kirjanduse ülevaade

Rakenduse arendamiseks uuriti Androidi tarkvaraarenduse põhimõtteid ja üleüldiselt tarkvaraarenduse häid tavasid kirjeldavaid teadusartikleid ning raamatuid. Allikate uurimise eesmärk oli saada kinnitust, et mobiilirakenduse arendamisel lähtutakse õigetest tarkvaraarenduse põhimõtetest. Vajaminevad teadusartiklid leiti *ACM* ning *IEEE Xplore* andmebaasidest. Raamatud leiti *O'Reilly* andmebaasist ja Tallinna Tehnikaülikooli raamatukogust.

4.2.1 Projekti teostamise põhjendus

Projekti arendamise üldist eesmärki põhjendab tõsiasi, et Eesti inimeste valmisolek elektriauto kasutamisele üle minekuks sisepõlemismootoriga autolt on hetkel madal. Transpordiameti sõidukite statistika kohaselt on Eestis 814 875 sõiduautost elektriautosid ainult 1962 tükki [15].

Eesti Energia AS tegeleb Enefit Volt brändi kaudu elektriautode laadimisvõrgu arendamisega Eestis, sh laadijate pakkumise ja elektriautode kasutamiseks vajaliku elektri müümisega üleüldiselt.

Probleem on, et tänasel hetkel ei otsusta Eesti inimesed elektriauto ostmise kasuks peamiselt elektriauto ja sisepõlemismootoriga auto ostuhinna erinevuse tõttu. Eesti Energia AS kaudne eesmärk on suurendada Eesti inimeste teadlikkust elektriauto ja sõiduauto kasutamise kütusekulude võrdlusest. Antud projekt keskendub konkreetselt elektriauto elektrikulu ning sõiduauto kütusekulu võrdlemisele. Läbi üldise teadlikkuse suurendamise oleks Eesti Energia AS-il võimalik paremini täita elektriautodega seotud ärilisi eesmärke: elektriautode kasutamiseks vajaliku elektri müümine ning avaliku laadimisteenuse ja koduste laadijate pakkumine klientidele.

Teise aspektina aidatakse kaasa inimeste teadlikkusele taastuvenergia tarbimise kasulikkusest keskkonnale. Taastuvenergia tarbimise kasv leevendab süsinikdioksiidi heitmeid atmosfääris. Täpsemalt vähendab elektrikütuse tarbimine transpordist tulenevat CO_2 ligi 12 protsenti [16]. Antud projekt aitab kaasa säästva transpordisüsteemi

tugevdamisele, suurendades elanike keskkonnateadlikkust läbi sisepõlemismootoriga ja elektriauto heite võrdluse ning suunates inimesi energiasäästlikele transpordivahenditele.

4.2.2 Rakenduse arhitektuuri põhjendus

Robert C. Martin'i raamatu „Clean Architecture" põhjal otsustati rakenduse arhitektuuri loomisel lähtuda puhta arhitektuuri (*clean architecture*) põhimõtetest [6]. Selle kohaselt saab head tarkvaraarendust iseloomustada järgnevalt:

1. rakendus on iseseisev kasutatud lisateekidest ja raamistikest
2. rakendus on testitav
3. rakenduse äriloogika ja kasutajaliides on sõltumatud
4. rakenduse äriloogika ning andmebaas on sõltumatud

Esimese punkti täitmiseks kasutati rakenduse loomisel nii vähe lisateeke ning raamistikke kui võimalik. Rakenduse *back-end*'is kasutatakse Jsoup avatud lähtekoodiga Java teeki, mille abil salvestatakse Fuelo veebilehekülje HTML elementidest kütuse hinnad. Rakenduse *front-end*'i loomisel eraldi teeke ja raamistikke ei kasutata.

Teise põhimõtte järgimiseks korrastati bakalaureusetöö raames rakenduse *back-end* ja *front-end*. Refaktoorimisel jälgiti, et igal klassil oleks konkreetne eesmärk, mistõttu tõsteti rakenduse loogika võimalusel eraldi klassidesse. Lisaks jälgiti, et iga meetod täidaks kindlat funktsiooni, mistõttu tõsteti varasemalt esinenud pikad meetodid lahku ja muudeti võimalusel korduvkasutatavaks. Tänu puhta arhitektuuri põhimõtete kasutamisele on rakendust võimalik paremini testida: andmebaasi ja API andmeid saab hõlpsalt testklassides imiteerida (*mock*).

Kolmanda ning neljanda põhimõtte alusel hoiti rakenduse *back-end* ja *front-end* võimalikult lahus. *Front-end* klasside suhtlus *back-end*'iga toimub läbi *ViewModel* klasside. Rakenduse *Activity* ja *Fragment* tüüpi klassid ei pöördu seega kordagi otse andmebaasi poole.

4.2.3 Andmebaasi ühendamine äriloogikaga

Rakenduse andmebaasi ühendamine äriloogika osaga toimub sarnaselt Martin Fowler'i poolt kirjeldatud *Table Data Gateway* andmeallika arhitektuurimustrile. *Table Data*

Gateway on liides, milles on kirjeldatud andmebaasist pärimise SQL laused: *select()*, *insert()*, *update()* ja *delete()*. *Table Data Gateway* liides, nagu nimigi viitab, suhtleb andmebaasis konkreetse tabeliga [7].

Antud projektis esindavad *Table Data Gateway* mustrit liidesed *CarDao*, *ElectricVehicleDao* ning *FuelDao*. Liidesed sisaldavad andmebaasist pärimise, kustutamise ning andmebaasi salvestamise meetodeid. Iga meetod kasutab *@Query* annotatsiooni, millele järgneb vastav SQL lause.

Martin Fowler'i kirjeldatud Objekt-Relatsioonvastenduse mustri (*Object-Relational Metadata Mapping Pattern*) implementeerimiseks on rakenduses loodud *CarRepository*, *ElectricVehicleRepository* ja *FuelRepository* klassid [7], milles määratakse DAO liideste meetoditele sisu.

4.2.4 Kasutajaliidese põhjendus

Kasutajaliidese arendamisel järgiti Jessica Thorsby raamatus „Android UI Design" kirjeldatud disaini printsiipe. UI disaini loomisel peeti oluliseks, et iga leht oleks kasutajale esimesel pilgul lihtsasti mõistetav, ilusa väljanägemisega, kiiresti reageeriv ja et kasutaja teekond rakenduses oleks loogiline ning sujuv [17]. Selle eesmärk oli tagada parim kasutajakogemus.

Efektiivne UI tagab piltlikult vestluse kasutaja ja rakenduse vahel [17]. Avalehel tuleb kasutajal sisestada auto numbrimärk. Ebakorrektse sisendi korral kuvatakse informeeriv veateade puuduvatest andmetest või valest vormingust, et aidata kasutajal jõuda korrektse sisendini. Võrdluse lehel, kui sõidukitel on kasutatud vaikeväärtusi, kuvatakse arvväärtustele vajutamisel väike hüpikteade (*Toast*), mis teavitab kasutajat nende kasutamisest.

Kogu rakenduses kasutati ühtset teemat ja stiili, mis loodi vastavalt Enefit Volt ametlikule stiilile. Ekraanikuvandi loomiseks kirjutati välja funktsionaalsused, mida taheti kasutajale näidata ning nende põhjal kujundati paberile ekraaniplaani (*wireframe*) visandid. Visandid anti edasi Eesti Energia disainiosakonna esindajale, kelle poolt loodi lõplik digitaalne prototüüp, mille järgi hakati kasutajaliidest looma.

Kasutajaliides deklareeriti läbi XML failide *resource* kaustas. See tagab eraldatuse UI kujunduse ja koodi vahel, mis rakenduse käitumist kontrollib. Antud viisil hoitakse

mõlemaid koodi osad puhtana ja antakse võimalus muuta UI-d ilma seda kontrollivat koodi muutmata.

Rakenduse vaikimisi keeleks on inglise keel. Rakendus on suuantud Eesti elanikkonnale, kus peamisteks emakeelteks on eesti ja vene keel. Kasutades lokaliseerimist (*locale*), lisati rakendusse peale inglise keele nii eesti kui ka vene keel.

4.2.5 Puhas kood

Selleks, et lähtekoodi oleks kerge hallata, lugeda ning testida, järgiti Robert C. Martin'i raamatu „Clean Code" põhimõtteid [10].

Klasside, meetodite ning parameetrite nimetused pidid olema kirjeldavad ja ühemõttelised. Kogu meeskond kasutas ühtset stiili sarnaste probleemide lahendamiseks. Kood pidi olema arusaadav, et seda lugedes saaks iga meeskonna liige ja juhendaja aru, mida konkreetsed koodiread teevad. Kommenteerimist kasutati üksikutel juhtudel, kui see oli vajalik.

Teises etapis pöörati rõhku klasside ja meetodite suurusele. Jälgiti SRP-d (*single-responsibility principle*), mille kohaselt peab meetod täitma ainult ühte eesmärki ning klass sisaldama sellele kuuluvat loogikat [10]. Lisaks meetodite osadeks jaotamisele eemaldati refaktoorimisel ebavajalikud kommentaarid, meetodid ja parameetrid. Samuti vähendati olulisel määral koodi korduvkasutamist.

Testide kirjutamisel järgiti samuti ülalnimetatud põhimõtteid. Testide klassid olid struktureeritud ning seetõttu oli nende käivitamine ka lihtsustatud [10].

4.2.6 Agiilne arendusmetoodika

Rakenduse arendus põhines agiilsel metoodikal. Robert C. Martin'i raamatu „Clean Agile" alusel lähtuti põhimõtetest, et töö tuleb planeerida, refaktoorida, lähtuda CI/CD tavast ja tagada efektiivne tiimitöö, et saada pidevalt kiiret tagasisidet [18].

Projektile seati kindel ajaline tähtaeg, millal töö peab valmima, et vastata äri ajalistele nõuetele. Samal ajal hoiti funktsionaalsed nõuded muutustele avatuna, kuna tööprotsessi käigus võisid esineda uued tingimused, et projekti peaeesmärki täita [18]. Selline lähenemine võimaldab jooksvalt nõudeid ümber hinnata ja uuesti läbi mõelda, et leida parim lahendus eesmärkideni jõudmiseks.

Projekt jagati kuuajalistesse vaheetappidesse. Esimese vaheetapi käigus pandi kirja kasutajalood ning funktsionaalsused, mis lõpuks täidaksid projekti eesmärke. Nende koostamisel jälgiti, et kasutajalugu kirjeldaks vaid üht ülesannet, looks väärtust, oleks ajaliselt hinnatav ja testitav. Antud elemendid jaotati vaheetappide vahel ja vajadusel muudeti jooksvalt nende sisu. Iga vaheetapp lõppes demonstratsiooniga, kus näidati valminud funktsionaalsusi ja töötavat koodi [18].

Igal tööpäeval peeti ligi 10-minutiliseid püstijala koosolekuid, mis andsid ülevaate tööprotsessi hetkeseisust. Koosolekuid juhtis *agile coach*. Iga meeskonnaliige vastas küsimustele – „Millega tegelesin eelmisel korral?“, „Mida plaanin teha täna?“ ning „Kas ja mis on takistused?“. Kuna arendus toimus enamasti kaugtööna, siis kasutati suhtlemiseks ning koosolekuteks Microsoft Teams suhtlusplatvormi.

Projekti üheks eesmärgiks oli ka olemasoleva koodi refaktoorimine, mis kujutab endast struktuuri parandamist ilma, et programmi käitumine muutuks. Selle käigus nimetati ümber klasse, meetodeid ja muutujate nimesid. Pikad meetodid lahutati väiksemateks ning klassid, mis omasid palju meetodeid, jagati alamklassideks. Tingimuseks oli selle juures, et kõik testid peavad õnnestuma, millega tagatakse programmi käitumise säilimine.

Rakenduse arendamisel kasutati pidevvalmiduse (*Continuous Delivery)* ja pideva integratsiooni tava (*Continuous Integration)* ehk tagati projekti stabiilsus ning pidevalt valmis versioon. Selleks loodi *pipeline*, mis kontrollib iga *commit*’i üleslaadimisel, et kood jookseks algusest lõpuni nagu soovitud. Samuti kontrollitakse ühikteste. Kui muudatused läbivad kontrolli, luuakse APK failina programmi uus väljalase (*release).*

4.3 Teostatud tööde logi

Projekti ajakulu arvestust peeti Toggl tarkvara abil ning on illustreeritud diagrammide abil. Tehtud tööde logi pidas iga liige eraldiseisvalt Excel tarkvara abil, kus märgiti nädala lõikes käsilolevad ning valminud ülesanded.

Tööjaotus on välja toodud allolevas tabelis.

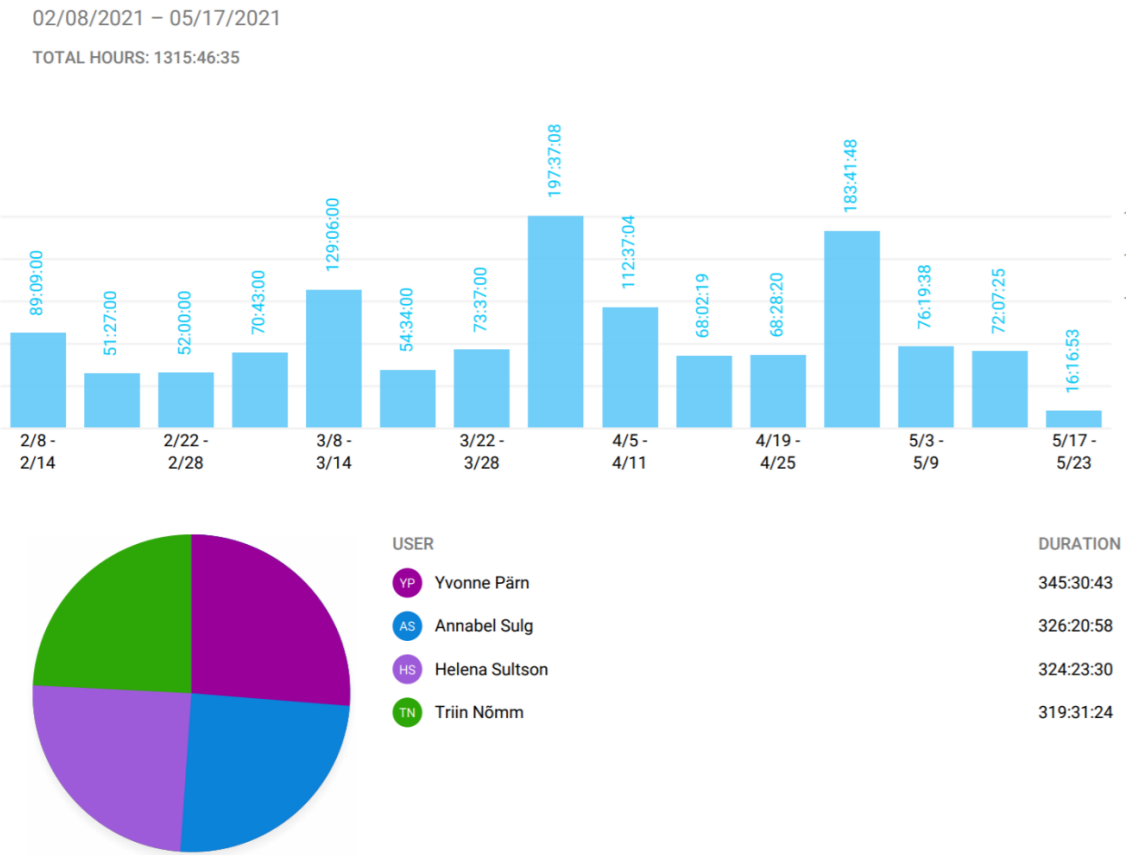
Tabel 1. Tööjaotus

Pealkiri	**Kirjeldus**	**Teostaja**
Rakenduse *release* ja APK	Rakenduse väljalaske ja APK faili genereerimine iga täienduse korral peaharus.	Helena Sultson
Kasutajaliidese uuendamine	Kasutajaliidese jagamine komponentideks ja neile disaini lisamine.	Yvonne Pärn / Triin Nõmm
UI komponendid (esimene leht)	Seadistada esimese lehe UI komponendid vastavalt prototüübile.	Yvonne Pärn
UI komponendid (teine leht)	Seadistada teise lehe UI komponendid vastavalt prototüübile.	Yvonne Pärn
Enefit Volt stiil	Lisada UI komponentidele Enefit Volt stiil (font, värvid, tekstitüüp).	Helena Sultson
Refaktoorimine (esimene leht)	Refaktoorida esimene lehekülg, täiendada ning paranda teste.	Annabel Sulg / Yvonne Pärn
Refaktoorimine (teine leht)	Refaktoorida teine lehekülg, täiendada ning paranda teste.	Yvonne Pärn
Mitmekeelsuse tugi	Lisada rakendusse eesti ja vene keele konfiguratsiooni võimalus.	Helena Sultson
5 aasta arvutused	Võrdluse perioodiks määrata 5 aastat.	Triin Nõmm
Elektriautode pildid	Elektriauto valimisel kuvada vastava auto pilti.	Annabel Sulg
CO_2 kalkulatsioonid	Lisada CO_2 emissiooni põhjal elektriauto keskkonnasäästlikkus võrreldes kasutaja sõiduautoga.	Helena Sultson
Rippmenüüde loogika parandamine	Parandada ripmenüüde loogikas vead.	Annabel Sulg
Võrgutasude lisamine	Arvestada elektrikulu arvutamisel kuu- ning võrgutasu.	Triin Nõmm

Tööjaotuse tabelis on väljatoodud ülesande nimetus, kirjeldus ning teostaja. Teostajaks määrati inimene, kes vastutab tehtud töö eest. Ülesande täitmisel võisid sellest osa võtta ka teised tiimiliikmed.

4.3.1 Ajakulu

Järgnevalt on välja toodud ajakulu kokku ning iga meeskonnaliikme kohta eraldi.

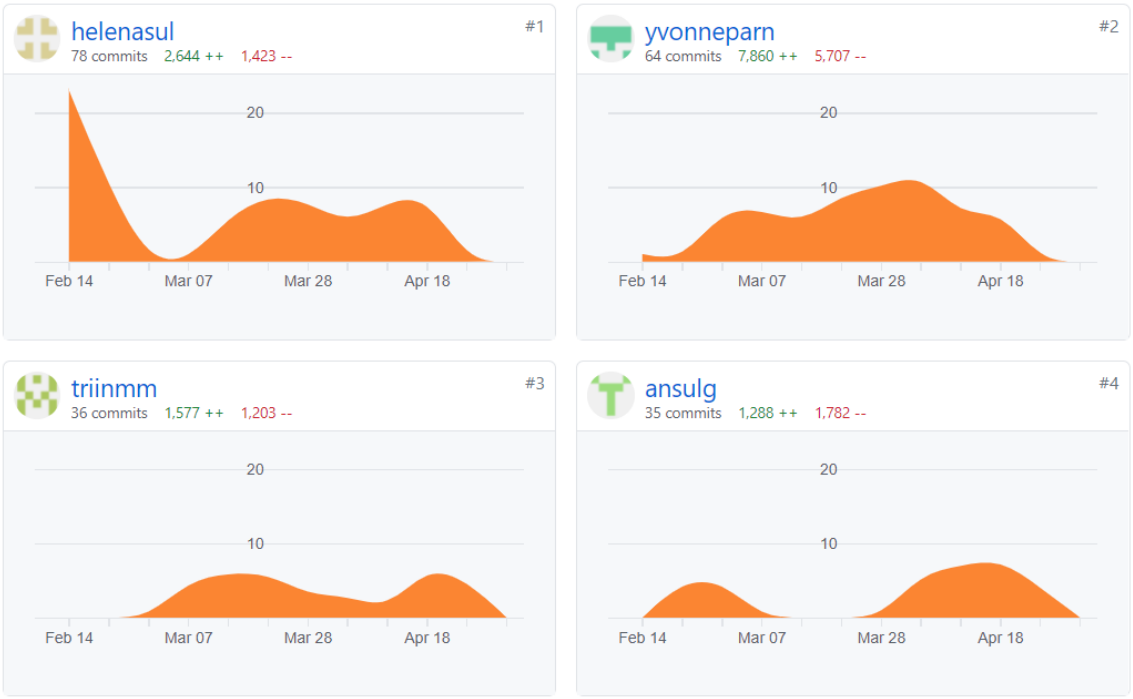


Joonis 53. Ajakulu meeskonnaliikmete kohta

Liige Yvonne Pärn panustas 345 tundi. Liige Annabel Sulg panustas 326 tundi. Liige Helena Sultson panustas 324 tundi. Liige Triin Nõmm panustas 319 tundi. Kokku kulus tiimil projekti valmimiseks ligikaudu 1316 tundi.

4.3.2 Giti commit'id

Järgnevalt on välja toodud *commit*'ide arv meeskonnaliikmete kaupa ajavahemikus 8. veebruar kuni 5. mai.



Joonis 54. *Commit*'ide väljavõte GitHubist

Commit'ide arv on arvestatud GitHubi peaharust, kuhu pandi kõik tiimiliikmete ja mentorite poolt aktsepteeritud muudatused.

4.3.3 Nädalalogid

Alltoodud tabelites on näidatud tiimiliikmete tehtud tööd nädalate lõikes perioodil 08.02-02.05.2021.

Tabel 2. Annabel Sulg nädalalogid

	Annabel
08.02-14.02	Projekti hetkeseisu tutvustamine ettevõttele, Esimese kuu planeerimine.
15.02-21.02	Rakenduse demostratsioon äriosakonnale. Nõuete sätestamine.
22.02-28.02	Piltide lisamine: API'st piltide URL'ide andmebaasi salvestamine.
01.03-07.03	API'st piltide URL'ide andmebaasi salvestamine, piltide kuvamine rakenduses.
08.03-14.03	1. kuu retrospektiiv. Piltide funktsionaalsuse lisamine: erinevate variantidega tutvumine. Tehnoloogiate Glide ja Picasso analüüs: tutvumine, implementeerimine.
15.03-21.03	Tehnoloogiate Glide ja Picasso analüüs: testkeskkonna stabiliseerimine, tehnoloogiate stabiilsuse testimine.

22.03-28.03	Tehnoloogiate Glide ja Picasso analüüs: efektiivsuse ning kasutatavuse testimine.
29.03-04.04	Tehnoloogiate Glide ja Picasso analüüs: dokumentatsiooni täiendamine, testimine. Koosolek juhendajatega, et valida parim tehnoloogia piltide kuvamiseks.
05.04-11.04	Tehnoloogiate Glide ja Picasso analüüs: dokumentatsiooni täiendamine. Piltide töötlemine vastavasse vormingusse ja stiili. Piltide kuvamise funktsionaalsus lisamine.
12.04-18.04	Piltide kuvamise funktsionaalsuse lisamise koodi täiendused ja parandused. Vigade parandamine: Rippmenüüle vaikeväärtuse lisamine, manuaalne testimine.
19.04-25.04	Vigade parandamine: Mudelite duplikatsiooni eemaldamine rippmenüüst, mudelite nimetustes tühikud taastada, manuaalne testimine, automaattestide lisamine. Rippmenüüde loogika refaktoreerimine. Ühiktestide lisamine ning parandamine kogus projektis
26.04-02.05	Kogu projekti retrospektiiv. Ühiktestide lisamine ning parandamine kogus projektis.

Tabel 3. Helena Sultson nädalalogid

	Helena
08.02-14.02	Sisseelamine projekti. Esimese kuu planeerimine ettevõttepoolsete juhendajatega.
15.02-21.02	Kohtumine äriosakonna esindajatega. Projekti edasise plaani paika panemine.
22.02-28.02	APK faili ja rakenduse väljalaske loomine.
01.03-07.03	APK faili ja rakenduse väljalaske loomine.
08.03-14.03	1. kuu retrospektiiv. APK faili logide uurimine vea leidmiseks. Paarisprogrammeerimine Triinuga seoses Maanteeameti API-st tehnoülevaatuste salvestamisega. Mitmekeelsuse ülesandega alustamine.
15.03-21.03	Eesti ja vene keele tõlgete tegemine ning lisamine rakendusse. Tõlgete Strings faili korrastamine, blokkideks jagamine, ebaoluliste väärtuste koodist eemaldamine. Rakenduse keele muutmise rippmenüü lisamine.
22.03-28.03	Rippmenüüle *back-end* loogika lisamine, sh *BaseActivity* klassi loomine, et vältida koodikordust. UI komponentide parandamine, et kõik keeled näeksid samasugused välja. Vene keele tõlgete parandamine.
29.03-04.04	Enefit Volt fonti lisamine rakendusse. Täiendused UI komponentidele Enefit Volt stiilis (värvid, tekstisuurus, küljendus).

05.04-11.04	CO_2 andmete uurimine Maanteeameti API-st. Arutelu meeskonnaga, milliseid CO_2 andmeid ja kuidas kasutajaliideses näidata.
12.04-18.04	CO_2 arvutuste loogika lisamine.
19.04-25.04	CO_2 arvutuste andmete näitamine kasutajaliideses. Kasutajaliidese testide lisamine. Paarisprogrammeerimine Yvonnega seoses *ComparisonActivity* refaktoorimisega. Ühiktestide lisamine.
26.04-02.05	Kogu projekti retrospektiiv. Viimased parandused rakenduses.

Tabel 4. Triin Nõmm nädalalogid

	Triin
08.02-14.02	Tiimi koosolek ja projekti plaani seadmine. Keskkonna seadistamine. Projektiga tutvumine.
15.02-21.02	Demo äriosakonnale. Nõuete paika seadmine. Kasutajaliidese uue disaini visandamine.
22.02-28.02	Kasutajaliidese uue disaini visandamine. Ettevõtte disainiosakonnaga läbirääkimine. Digitaalse prototüübi disainimine.
01.03-07.03	Rakenduse uuendatud kasutajaliidese loomine.
08.03-14.03	1. kuu retrospektiiv. Uue sprindi planeerimine. Avalehe vaate parandamine klaviatuuri avamisel. Auto mudeli pärimine Maanteeameti API-st.
15.03-21.03	Auto mudeli kuvamine võrdluse lehel.
22.03-28.03	Ülevaatuste vahelise perioodi muutmine võimalikult pikale ajale. Andmete puudumisel vaikeväärtuste seadmine. Ebavajalike arvutusmeetodite eemaldamine ning asendamine 5 aasta kütusekulu arvutavate meetoditega. Võrdluse lehe lõpus olevale Enefit Volt lehele viiva nupu seadistamine.
29.03-04.04	5 aasta arvutuste loogika parandamine. Kalkulatsioonide testide muutmine/parandamine.
05.04-11.04	5 aasta arvutuste koodi refaktoorimine. Võrgutasude arvestamine elektriauto hinna arvutamisel.Võrgutasude testide parandamine.
12.04-18.04	Võrgutasude kuvamine kasutajaliideses. Koodi refaktoorimine. Võrgutasude testide kirjutamine/muutmine.
19.04-25.04	Koodi muutmine teiste harudega kattuvaks. Võrgutasude hüpikaknas kuvamise loogika muudatused.
26.04-02.05	Kogu projekti retrospektiiv. Võrgutasude pileti vigade parandamine.

Tabel 5. Yvonne Pärn nädalalogid

	Yvonne
08.02-14.02	Esimese kuu planeerimine juhendajate ja tiimiga.
15.02-21.02	Demonstratsioon äriosakonnale. Projekti plaani paikapanemine. Kasutajaliidese planeerimine ja visandamine.
22.02-28.02	Visandatud kasutajaliidese esitlemine juhendajatele ja Eesti Energia disainerile. Lõpliku kasutajaliidese disaini kooskõlastamine disaineriga.
01.03-07.03	Uue kasutajaliidese realiseerimisega alustamine. Avalehe uute UI komponentide loomine ja Enefit Volt stiili viimine. Elektriautode valimise lisamine avalehele.
08.03-14.03	Esimese vaheetapi retrospektiiv ja uue kuu planeerimine. Võrdluse lehe UI komponentide loomine, korduvkasutatavaks muutmine ja Enefit Volt stiili viimine.
15.03-21.03	UI komponentidele *back-end* loogika lisamine. Uute hüpikakende loomine. Hüpikakende komponentide lisamine ja viimine Enefit Volt stiili.
22.03-28.03	Võrdluse lehele rippmenüü lisamine kütusetarbimise tüübi valimiseks. Loodud rippmenüüle *back-end* loogika lisamine. Veateadete uue disaini loomine.
29.03-04.04	*ComparisonActivity* klassi refaktoorimisega alustamine. Testide korrastamine ja lisamine. *ComparisonActivity* meetodite tõstmine uutesse klassidesse. Hüpikakende jaoks eraldi Fragmenti loomine. Hüpikakende ehitamise jaoks loodud loogika ja meetodite refaktoorimine.
05.04-11.04	Loogika lisamine säästu summa ja auto kütusekulu väärtuse automaatseks uuendamiseks kütusetarbimise tüübi muutmisel. Juhendajate tagasiside põhjal paranduste tegemine refaktoorimise ülesandes.
12.04-18.04	Parandused koodis ja testides juhendajate tagasiside põhjal. Uute loodud klasside testimine nii ühik- kui ka *instrumented* testide näol.
19.04-25.04	Paarisprogrammeerimine Helenaga, et *ComparisonActivity* klassis olevaid meetodeid veel enam lahku tõsta ja koodi puhtamaks muuta. Ava- ja võrdluse lehel olevate rippmenüüde loogika refaktoorimine. Manuaalne testimine. Ühik- ja *instrumented* testide kirjutamine ning parandamine.
26.04-02.05	Kogu projekti retrospektiiv.

Rakenduse arendustöö lõpetati maikuu alguseks, misjärel tegeleti bakalaureusetöö dokumendi koostamisega.

4.4 Hinnang projekti teostamise protsessi kohta

Järgnevalt antakse ülevaade projekti juhtimise ning teostamise protsessist. Lisaks tuuakse välja hinnang nii projekti kitsaskohtadele ja õnnestumistele kui ka projekti üldisele teostamise protsessile.

4.4.1 Projekti juhtimine ja teostamise protsess

Projekti teostati septembri algusest kuni mai alguseni. Projekti esimene etapp, mis toimus meeskonnaprojekti aine raames, kestis septembrist jaanuari keskpaigani. Teise etapi arendustööga alustati 8. veebruaril ning arendustöö lõppes 30. aprillil. Tööpäevadeks lepiti kokku neljapäev ja reede, kus tööd tehti vähemalt 8 tundi. Lisaks tegid meeskonnaliikmed tööd ka töövälistel päevadel vastavalt endi võimalustele.

Projekti ülikoolipoolsed juhendajad olid Tallinna Tehnikaülikooli lektor Kristina Murtazin ja assistent Jekaterina Tšukrejeva. Eesti Energia AS poolt määrati projekti mentoriteks tooteomanik Jaan Ots, arendaja Osama Mohamed Mahmoud Mohamed ja arendaja Vinay Puranik. Lisaks aitasid koosolekuid ning agiilset arendusmetoodikat juhtida juhendajad Ruslan Bjurkland ning Einar Koltšanov.

COVID-19 ning kehtestatud eriolukorra tõttu alates märtsist tavapärast kontoritööd ei toimunud, vaid tööd tuli teha distantsilt. Juhendajatega ja meeskonnavaheliseks suhtluseks kasutati suhtluskanalit Microsoft Teams. Kokkulepitud tööpäevadel olid nii juhendajad kui ka meeskonnaliikmed terve päeva vältel kättesaadavad, et vajadusel üksteist abistada või tekkinud probleemidele lahendusi leida. Iga tööpäeva hommikul toimus *stand-up* koosolek, kus meeskonnaliikmed viisid üksteist ning juhendajaid kurssi, millega nad hetkel tegelevad. Lisaks toimus suhtlus juhendajatega ka GitHubi kaudu, kus anti tagasisidet kirjutatud koodile.

Iga vaheetapi lõpus toimus uue vaheetapi planeerimise koosolek, kus pandi üheskoos paika järgmise kuu eesmärgid ning teostatavad ülesanded. Ülesannete jagamisel meeskonnaliikmete vahel lähtuti eelkõige meeskonnaliikmete eelistustest ja oskustest, kuna teise etapi arendustöö aeg oli piiratud. Jooksvalt abistati ka teisi meeskonnaliikmeid, et kõik planeeritud ülesanded saaksid kokkulepitud ajaks teostatud.

4.4.2 Projekti õnnestumised ja kitsaskohad

Meeskonnaprojekti aine ehk projekti esimese etapi lõpus näidati valminud rakendust äriosakonnale. Meeskond sai head tagasisidet ning uusi ideid, kuidas kevadel rakendust täiendada. Kuna tööprotsess oli kõigile meeskonnaliikmetele juba sügisesest etapist tuttav ja algne projekti edasiarendamise plaan oli paigas, oli bakalaureusetöö raames projekti edasiarendamine üldiselt sujuv protsess.

Projekti teise etapi eesmärkide seadmiseks oli tudengitel võimalik ise lõplikud otsused langetada. Äriosakonna (sh tooteomaniku) poolt pakuti välja ideed, mille teostuse üle oli meeskonnal võimalik ise otsustada. Selline iseseisvus ning ettevõttepoolne usaldus oli meeskonnaliikmete poolt väga hinnatud ja innustas andma endast parimat.

Meeskond töötas väga iseseisvalt. Kõik plaanitud ülesanded ja rakenduse täiendused said õigeaegselt ning korrektselt tehtud. Kohati oli ettevõttepoolne usaldus kevadise etapi jooksul liiga suur, näiteks alates märtsist ei osalenud kõik ettevõttepoolsed juhendajad *stand-up* koosolekutel. Sealjuures tundis meeskond, et ettevõttepoolne järelevalve tudengite tööle oleks võinud olla järjepidevam nagu projekti esimeses etapis.

Kasutajaliidese korrastamine oli algselt planeeritud ühe ülesandena. Tööd tehes selgus, et UI disainimine on keerulisem, kui algselt arvati, sest puudus ettevõttepoolne ühtne Android rakenduse stiili fail. Kasutajaliidese korrastamiseks tuli kasutada eraldi Enefit Volt kodulehe elemente, ettevõttepoolset CSS faili ning ka prototüüpi. Tudengite ettepanekul vaadati kasutajaliidese ülesanne üle ning jagati see neljaks erinevaks väiksemaks ülesandeks, millega said tegeleda kõik meeskonnaliikmed.

Meeskonnatööd mõjutas koroonakriis, mille tõttu suunduti märtsist alates kodukontorisse. Kuigi kõigi liikmete motivatsioon ja enesejuhtimise oskused olid kõrged, vajas kodukontoriga harjumine siiski aega.

4.4.3 Hinnang projekti teostamise protsessile

Meeskond planeeris arendustöödega lõpetada maikuu alguseks, et edasi tegeleda ainult bakalaureusetöö dokumendiga. Ülikoolipoolsete juhendajatega suheldi seoses dokumendi koostamisega pidevalt. Küsimuste tekkimise korral võeti nendega ühendust kirja või veebikõne teel. Keskendudes ühele kindale ülesandele, tagati efektiivne ning hea kvaliteediga töö.

Peamiselt olid meeskonnaliikmed kaugtööl, mis vähendas oluliselt suhtlust nii tiimiliikmete kui ka juhendajate vahel. Hea meeskonnatöö tagamiseks korraldati koosviibimisi, kus jagati üksteisele oma rõõmusid ja muresid. Samuti hoiti ühendust ka töövälisel ajal. Meeskonna arvates on siiski oluline võimaluse korral töötada kontoris. See tagab parema suhtluse ning usalduse tiimiliikmete vahel.

Kogu projekti vältel suheldi ettevõtte juhendajatega veebi kaudu. Sellest tingituna võttis osadele küsimustele vastuse saamine kauem aega. Meeskonnaliikmetel tuli probleeme kirjeldada võimalikult täpselt, et lisaküsimustele ei peaks ajalist ressurssi kulutama. Täpsete kirjelduste koostamine õpetas meeskonnaliikmeid oma tegevusi põhjendama ja probleeme analüüsima.

Samuti räägiti ning leiti lahendusi probleemidele pärast neljapäeva või reede *stand-up* koosolekut. Selles osalesid nii mentorid kui ka tiimiliikmed. Lisaks anti tagasisidet kirjutatud koodi kohta – alguses tiimiliikmete ning pärast seda mentorite poolt. Sellega tagati hea töökvaliteet ning kõikide meeskonnaliikmete kaasamine protsessi.

4.5 Meeskondlik konsensuslik hinnang

Meeskonnatöö kulges sujuvalt ning probleemideta. Kõik tiimiliikmed panustasid rakenduse pidevasse arengusse ja osalesid koosolekutel. Tööajaväliselt tegutseti ülesannetega iseseisvalt või võimalusel koos, sealjuures kogu protsessi ajal kokkuvõtvalt võrdselt. Meeskondlikult hinnatakse üksteise panust võrdselt hindega „0“.

5 Kokkuvõte

Bakalaureuse lõputöö raames valmis ettevõttele Eesti Energia AS rakendus sisepõlemismootoriga ja elektriauto kütusekulude ning saaste võrdluseks. Projektiga toetati Eesti Energia AS ärilisi eesmärke: aidata kaasa elektriautode populariseerimisele Eesti elanike seas ning olla kaasatud tavasõiduautolt elektrikütusega auto peale ülemineku protsessis. Meeskonnaprojekti raames valminud rakendus oli algelise kasutajaliidesega ning puudulike funktsionaalsustega. Seetõttu olid lõputöö eesmärkideks täiendada rakendust lisafunktsioonidega, kujundada vastavaks Eesti Energia elektriauto brändi Enefit Volt stiiliga ning korrastada rakenduse kood.

Töö valmis agiilse arendusmetoodika põhimõttel, mille raames täiendati meeskonnaprojekti funktsionaalsustega ning vajalike muudatustega. Android rakenduse arendus toimus Android Studio keskkonnas Java ja XML arenduskeeltes ning koodi haldamiseks oli kasutusel GitHub lähtekoodihoidla.

Lõputööna valminud rakendus pakub kasutajale võimalust näha võrdlust enda sisestatud sõiduauto ning võrdluseks valitud elektriauto vahel. Võrdluses tuuakse välja erinevused sõidukite kütusekulu ja CO_2 heite vahel ning näidatakse rahalist säästu elektriautot kasutades. Projekti jooksul korrastati rakenduse kood ning kujundati ümber kasutajaliides, millega parandati kasutajakogemust.

Projekti arendamisel lähtuti ettevõtte juhendajate, äriosakonna esindajate ning meeskonna poolt seatud nõuetest ning teaduskirjandusest. Meeskonnatöö kulges sujuvalt ning tekkinud probleemidele leiti konsensuslikult lahendused. Kõik lõputööle seatud eesmärgid said täidetud.

Meeskond tänab Eesti Energia AS projekti püstituse ning juhendajaid suunamise ja abi eest.

Kasutatud kirjandus

[1] Fuelo, *Fuel prices today*, 2021. [Online]. Loetud aadressil: https://m.fuelo.net/m/prices?lang=en&country=ee Kasutatud 27.04.21.

[2] Nord Pool AS, *Day-ahead prices*, 2021. [Online]. Loetud aadressil: https://www.nordpoolgroup.com/Market-data1/Dayahead/Area-Prices/EE/Monthly/?view=table Kasutatud 27.04.21.

[3] U.S. Department of Agriculture, *The Power of One Tree - The Very Air We Breathe*, 2019. [Online]. Loetud aadressil: https://www.usda.gov/media/blog/2015/03/17/power-one-tree-very-air-we-breathe Kasutatud 30.04.21.

[4] Eesti Energia AS, *Enefit Volt*, 2021. [Online]. Loetud aadressil: https://enefitvolt.com/avaleht Kasutatud 28.04.21.

[5] Elektrilevi OÜ, *Elektrilevi võrguteenuse hinnakiri*, 2018. [Online]. Loetud aadressil: https://www.elektrilevi.ee/-/doc/6305157/kliendile/elektrilevi_hinnatariifid_2019km2.pdf Kasutatud 29.04.21.

[6] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*, First Edition. New Jersey: Prentice Hall Press, 2017.

[7] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley Longman Publishing Co., 2002.

[8] T. Szostak, *Windows Phone 8 Application Development Essentials*. Birmingham: Packt Publishing Ltd, 2013.

[9] Google, *Android Developers: Documentation for App Developers*, 2021. [Online]. Loetud aadressil: https://developer.android.com/docs Kasutatud 27.04.21.

[10] R. C. Martin, *Clean Code: Handbook of Agile Software Craftsmanship*. New Jersey: Prentice Hall Press, 2008.

[11] D. Gries, *JavaHyperText and Data Structures*, Cornell University Department of Computer Science, 2017. [Online]. Loetud aadressil: https://www.cs.cornell.edu/courses/JavaAndDS/JavaStyle.html Kasutatud 26.04.21.

[12] PMD Open Source Project, *Java Rules*, 2021. [Online]. Loetud aadressil: https://pmd.github.io/pmd-6.34.0/pmd_rules_java.html#code-style Kasutatud 28.04.21.

[13] Robolectric, *Test-drive your Android code*, 2021. [Online]. Loetud aadressil: http://robolectric.org/ Kasutatud 25.04.21.

[14] GitHub, Inc., *GitHub Docs: About continuous integration*, 2021. [Online]. Loetud aadressil: https://docs.github.com/en/actions/guides/about-continuous-integration Kasutatud 27.04.21.

[15] Transpordiamet, *Sõidukite statistika*, 2021. [Online]. Loetud aadressil: https://www.mnt.ee/et/ametist/statistika/soidukite-statistika Kasutatud 15.05.21.

[16] A. Amin, B. Altinöz ja E. Dogan, „Analyzing the determinants of carbon emissions from transportation in European countries: the role of renewable energy and urbanization,“ *Clean Technologies and Environmental Policy,* kd. 22, nr 1, 2020. doi: 10.1007/s10098-020-01910-2

[17] J. Thornsby, *Android UI Design: Plan, design, and build engaging user interfaces for your Android applications*. Birmingham: Packt Publishing, 2016.

[18] R. C. Martin, *Clean Agile: Back to Basics*. Boston: Pearson Education Inc., 2020.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

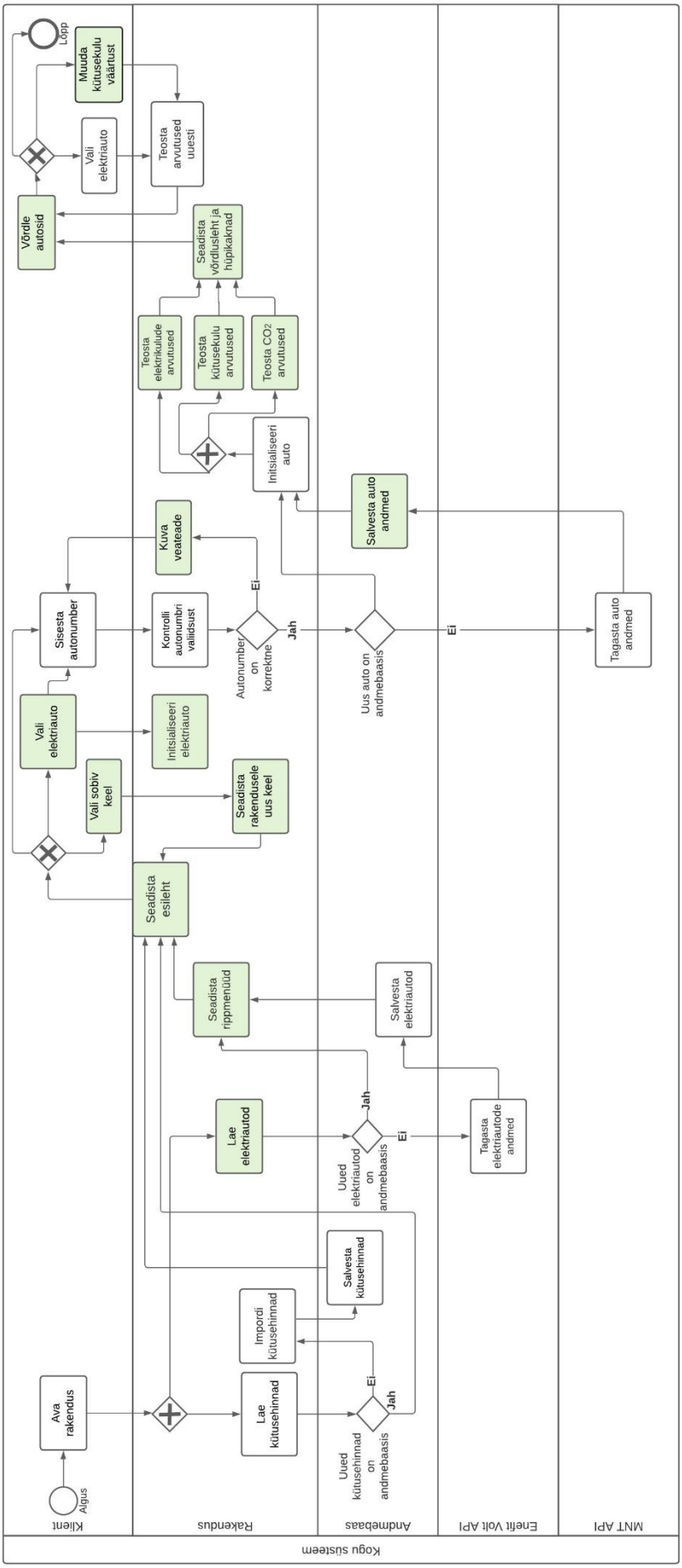
Mina, Triin Nõmm

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Eesti Energia AS kütusekulu simulaatori rakenduse arendus“, mille juhendajad on Kristina Murtazin ja Jekaterina Tšukrejeva
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

10.05.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – BPNM diagramm



Lisa 3 – Github Actions koodi automaatse kontrolli .yml fail

```
name: Test and build Android App

on:
  push:
    branches:
      - '*'

jobs:
  code_analyze_and_create_release:
    runs-on: ubuntu-latest
    steps:
      - name: checkout
        uses: actions/checkout@v2

      - name: Grant execute permission for gradlew
        run: chmod +x ./EVOwnershipSimulatorApp/gradlew

      - name: PMD Source Code Analyzer Action
        uses: sfdx-actions/setup-pmd@v1

      - name: pmd-analyze
        run: pmd -d EVOwnershipSimulatorApp -R
EVOwnershipSimulatorApp/ruleset.xml -f text

      - name: Spotless
        run: ./EVOwnershipSimulatorApp/gradlew -p ./EVOwnershipSimulatorApp
spotlessCheck

      - name: Secrets file
        env:
          MAANTEEAMET_USER: ${{ secrets.MAANTEEAMET_USER }}
          MAANTEEAMET_PASSWORD: ${{ secrets.MAANTEEAMET_PASSWORD }}
        run: echo "<?xml version='1.0' encoding='utf-8'?><resources><string
name='username'>"$MAANTEEAMET_USER"</string><string
name='password'>"$MAANTEEAMET_PASSWORD"</string></resources>" >
./EVOwnershipSimulatorApp/app/src/main/res/values/secret.xml

      - name: Unit tests
        run: ./EVOwnershipSimulatorApp/gradlew -p ./EVOwnershipSimulatorApp clean
test

      - name: Build with Gradle
        run: ./EVOwnershipSimulatorApp/gradlew -p ./EVOwnershipSimulatorApp clean
assembleDebug

      - name: Generate build number
        id: buildnumber
        uses: einaregilsson/build-number@v3
        if: github.ref == 'refs/heads/master'
```

```
    with:
        token: ${{secrets.github_token}}

- name: Print new build number
      if: github.ref == 'refs/heads/master'
      run: echo "Build number is $BUILD_NUMBER"

    - name: Create release and upload apk
      uses: underwindfall/create-release-with-debugapk@v2.0.0
      if: github.ref == 'refs/heads/master'
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
      with:
        tag_name: ${{ steps.buildnumber.outputs.build_number }}
        release_name: Release ${{ steps.buildnumber.outputs.build_number }}
        asset_path: EVOwnershipSimulatorApp/app/build/outputs/apk/debug/app-
debug.apk
        asset_name: EVOwnershipApp.apk
        asset_content_type: application/zip
```

Lisa 4 – Helena Sultson eneseanalüüs

Bakalaureusetöö jooksul oli minu panus töösse põhjalik ning järjepidev. Sain tegeleda nii *front-end*'i, *back-end*'i kui ka ühe tehnilise ülesandega. Lisaks arendustööle panustasin bakalaureusetöö dokumendi kirjutamisse ja ülikoolipoolsete juhendajatega suhtlemisse.

Projekti teises etapis ehk bakalaureusetöö raames olid minu isiklikud ülesanded APK faili loomine GitHubis, mitmekeelsuse lisamine, Enefit Volt stiili täiendamine kasutajaliideses ja CO_2 arvutused.

Bakalaureusetöö arendamine algas veebruaris rakenduse väljalaske (*release*) ja APK faili loomisega. Ülesande valmimiseks täiendasin olemasolevat CI/CD automaatkontrolli .yml formaadis faili unikaalse väljalaske koodi lisamise, APK faili ja rakenduse väljalaske genereerimisega. Ülesande tulemusena tekib peale igat juhendajate poolt peaharusse lisatud muudatust GitHubis rakenduse *release*, mis sisaldab APK faili ning .zip formaadis kogu rakenduse koodi. Antud ülesande näol oli tegemist huvitava väljakutsega, sest projekti esimeses etapis ma CI/CD *pipeline* loomise ja haldamisega ei tegelenud.

Järgmisena tegelesin mitmekeelsuse, täpsemalt vene ja eesti keele lisamisega. Ülesande valmimine koosnes mitmest osast: rakenduses kasutatavate lausete, pealkirjade ja ühikute eesti ning vene keelde tõlkimine, vastavate tõlgete lisamine *resources* kausta failidesse, *back-end*'is rakenduse keele loogika implementeerimine ning seejärel kasutajaliideses keele vahetamiseks vajaliku rippmenüü loomine.

Tagantjärgi võib öelda, et tegemist oli kõige ajamahukama ning keerulisema ülesandega projekti teise etapi jooksul, sest ülesande raames tuli teha mitmed uued aspektid enda jaoks selgeks. Keele tõlgete loomisel sain abi ettevõttepoolsetelt juhendajatelt ning meeskonnaliikmetelt. Näitasin kindla aja tagant meeskonnaliikmetele rakenduse keelte väljanägemist, tänu millele sain tõlkeid jooksvalt parandada.

Peale keelte lisamist täiustasin olemasolevaid uusi kasutajaliidese komponente veel puuduva Enefit Volt stiiliga. Lisasin rakendusse Enefit Volt ametliku fonti, milleks on Uni Neue. Seejärel täiustasin prototüübi alusel rakenduse tekstis, nuppudel ja numbriväljadel kasutatavaid värve, teksti suurust ja tekstitüüpi. Keerulisem osa ülesandest oli rakenduse allosas oleva puude ja aastate arvu, mis kulub sõiduauto CO_2 heite taastamiseks, näitamine. Tegemist on dünaamiliselt muutuva String tüüpi lausega,

millest osa on rohelist, osa halli värvi. Meeles tuli pidada seda, et lause peab olema õigete värvidega nii eesti, inglise kui ka vene keeles.

Viimane iseseisev ülesanne projektis oli CO_2 arvutused. Selle raames tuli langetada otsuseid nii *back-end*'i loogika kui ka kasutajaliideses arvutuste kuvamise kohta. Ülesande tulemusena täiendasin olemasolevat *Calculations* klassi vajaminevate arvutustega ning lisasin arvutuste tulemused kasutajaliidesesse sõiduauto sektsiooni ja hüpikaknasse. CO_2 andmete puudumise korral toimivad arvutused vaikeväärtuse abil ning ka kasutajaliidesest on võimalik kasutajal aru saada, et kasutatakse vaikeväärtust. Kuna selgus, et CO_2 emissioon kilogrammides võib olla suurem arv kui 10 000, täiendasin rakendust veel ka arvude vormindamise abimeetoditega.

Iga ülesande valmimiseks ja kontrollimiseks kirjutasin ka teste.

Kokkuvõtvalt õppisin arendustöö jooksul mõtestatult programmeerima ja koodi kirjutamise häid tavasid kasutama. Õppisin probleemidele ning küsimustele paremini Internetist vastuseid leidma ja oma tehtud tööd teistele selgitama, sh ettevõttepoolsetele mentoritele inglise keeles. Tööde õigeks ajaks valmis jõudmise nimel tuli panustada lisaks kokku lepitud tööpäevadele ka üsna palju omast vabast ajast.

Lisaks iseseisvalt valminud ülesannetele pakkusin end meeskonnaliikmetele probleemide korral appi. Näiteks abistasin tudengit Triin Nõmm tehnoülevaatuste andmete salvestamise korrastamisega ja tudengit Yvonne Pärn suurema refaktoorimise ülesande raames. Meeskonnaliikmeid toetasin nii ametlike tööpäevade jooksul kui ka isiklikust ajast.

Hindan enda panust meeskonnatöö toimimisse kõrgelt. Olin järjepidevalt aktiivne suhtleja ja probleemide lahendaja nii kokku lepitud tööpäevadel kui ka töövälisel ajal. Võtsin initsiatiivi ettevõttepoolsete juhendajatega toimunud koosolekutel, näiteks demonstreerisin meie rakendust veebruari alguses äriosakonnale. Lisaks arendustööle andsin enda panuse projekti bakalaureusetöö kavandi ning dokumendi kirjutamise ja juhendajatega suhtlemise näol.

Lisa 5 – Yvonne Pärn eneseanalüüs

Projekt algas kohtumisest juhendajate ning äripoolega, kus seati paika nõuded ja lepiti kokku teostamisele kuuluvad ülesanded. Bakalaureusetöö raames olid minu suurimad panused koodi puhastamine ning kasutajaliidese ümberdisainimine Enefit Volt stiilis. Projekti käigus tegelesin *front-end* ja *back-end* koodi arendamisega.

Esimese vaheetapi jooksul töötasin meeskonna ning ettevõttepoolse UI disaineriga välja uue kasutajaliidese prototüübi. Prototüübist lähtuvalt hakkasin kogu kasutajaliidest ümber disainima. Minu esimene ülesanne nimega "UI Improvements" sisaldas kogu kasutajaliidese täielikku ümberkujundamist. Esimeses vaheetapis alustasin uute UI komponentide loomisega avalehele ning komponentide viimisega Enefit Volt stiili. Tõin elektriautode valimise võimaluse avalehele ning lisasin loodud rippmenüüdele loogika *back-end*'i. Komponentide Enefit Volt stiili viimiseks analüüsisin Enefit Volt veebilehte.

Esimese vaheetapi lõpus selgus, et kasutajaliidese ümberkujundamine on palju suurem töö, kui algselt planeeritud oli. Seetõttu otsustati ühise arutelu tulemusena jagada ülesanne neljaks väiksemaks osaks – avalehe UI komponentide loomine, võrdluse lehe UI komponentide loomine, rakenduse ühtlustamine Enefit Volt stiiliga ja elektriautode piltide lisamine avalehele.

Teises vaheetapis oli minu peamiseks ülesandeks avalehe, võrdluse lehe ja hüpikakende jaoks korduvkasutavate UI komponentide loomine ning neile loogika külge lisamine. Komponentide loomisel lähtusin prototüübist. UI komponentide korduvkasutatavaks tegemiseks lõin erinevaid *layout* .xml faile, mis sisaldasid komponentide kujundust ja paigutust. Seejärel lisasin need *include* klausliga avalehe, võrdluse lehe ja hüpikakende vaadetele. Komponentide loomisel jälgisin õiget joondust, Enefit Volt stiili ja proovisin leida parimaid lahendusi nö spagetikoodi (*spaghetti code)* vältimiseks. Jooksvalt lisasin kasutajaliidesele külge ka loogika ning kirjutasin ühik- ja *instrumented* teste, et kontrollida, kas rakendus töötab ootuspäraselt.

Kolmandas vaheetapis alustasin võrdluse lehe refaktoorimisega. Kogu võrdluse lehe loogika sisaldus algselt ühes *ComparisonActivity* klassis, mis sisaldas üle 600 koodirea ning ei olnud lähtutud puhta koodi põhimõtetest. Refaktoorimise eesmärk oli *ComparisonActivity* klassis olevate meetodite ümertõstmine uutesse klassidesse, et igal

meetodil ja klassil oleks üks kindel eesmärk ning kood oleks loetavam. Refaktoorimise käigus lõin *comparison* moodulisse uued klassid, mis täidavad ühte kindlat eesmärki, parandasin nimetamise vigu, võtsin kasutusele erinevaid disainimustreid, et vähendada koodikordust, lisasin lisafunktsionaalsuseid võrdluse lehele ning täiendasin võrdluse lehte vajalike andmetega. Näiteks lisasin kütusetarbimise tüübi valimise rippmenüü ja selle loogika, vaikeväärtuste kuvamise loogika ja hüpikakende ehitamise uue loogika. Koodikorduse vähendamiseks lõin uue klassi tekstivaadete seadistamiseks vajalike abimeetoditega ning hüpikakende ehitamise juures võtsin kasutusele strateegia mustri, et kõigi hüpikakende puhul saaks kasutada ühte ja sama meetodit. Ühtlasi korrastasin ja lisasin ühik- ning *instrumented* teste ja puhastasin kogu koodi üleliigsetest osadest.

Pärast võrdluse lehe refaktoorimise lõpetamist alustasime koos meeskonnakaaslase Annabel Sulega avalehe refaktoorimise ülesandega. Tõstsin *MainActivity* klassis sisalduvad meetodid uutesse klassidesse ja puhastasin koodi vastavalt puhta koodi põhimõtetele. Lisaks refaktoorisime kogu rippmenüüde loogika, et vähendada koodikordust. Rippmenüüde loogika refaktoorimisel võtsime kasutusele OOP printsiibid pärimine ja polümorfism ning disainimustri *Strategy*.

Lisaks nimetatud osadele abistasin pidevalt tiimikaaslasi tekkinud probleemidega ning andsin järjepidevalt tagasisidet teiste meeskonnaliikmete kirjutatud koodile. Näiteks abistasin meeskonnakaaslast Helena Sulg keelevaliku rippmenüü loomisel ja tudengit Triin Nõmm arvutuste meetodite korrastamisel.

Antud projektis pean enda suurimaks panuseks kasutajaliidese ümberdisainimist ning koodi puhastamist. Projekti tegemine õpetas mulle meeskonnas töötamist, tööde efektiivset jaotamist ja probleemidele lahenduste leidmist. Lisaks tunnen, et projekti tegemine arendas oluliselt minu tehnilisi teadmisi – õppisin kirjutama puhast koodi ja teadlikult kasutama erinevaid disaimustreid.

Tunnen, et minu panus valminud bakalaureusetöösse oli väga põhjalik ja oluline. Lisaks kokkulepitud tööpäevadele olin meeskonnaliikmetele ning juhendajatele kättesaadav ka töövälisel ajal. Suurema osa tehtud tööst, tegin oma vabast ajast ja andsin endast terve projekti vältel maksimumi.

Lisa 6 – Triin Nõmm eneseanalüüs

Lõputöö raames olid minu ülesanneteks kasutajaliidese disainimine, arvutuste korrigeerimine viie aasta perioodile ja võrgutasude ning võrguteenuse makse lisamine elektrikulu arvestusse. Ülesannete lahendasemiseks tegelesin *front-end*'i ning *back-end*'i arendamisega. Lisaks aitasin meeskonnaliikmetel jooksvalt lahendada probleeme ja pakkusin vajadusel abi.

Suurimateks väljakutseteks olid uus töökeskkond ning uue projektiga liitumine kuna sisseelamine tiimiga, keskonna seadistamine ja rakendusega tutvumine võttis aega. Tingitud Covid-19 olukorrast tehti kogu tööd kodukontoritest, mis oli raskendavaks asjaoluks tiimi kommunikatsioonis ja ka mentoritega suhtluses. Teisest küljest õpetas kaugtöö tiimina olema iseseivam ning lahendama probleeme tiimisiseselt enne, kui pöörduti mentorite poole.

Tööd alustati veebruari kuus ning kuna liitusin uue tiimiga, siis võttis esimene nädal aega keskkonna seadistamiseks ning eelnevalt valminud rakendusega tuvumiseks. Suureks abiks oli meeskonnakaaslaste tugi, kes aitaksid projektiga sujuvalt järjepeale saada.

Järgnevaks ülesandeks sai kasutajaliidese ümberdisainimine. Eelneva rakendenduse kasutajaliides oli algeline ning kasutajakogemuse parandamiseks oli tarvis UI ümber kujundada. Selleks visandasime koos Yvonne Pärnaga mitmed kujundused ning lõime digitaalse prototüübi, mis saadeti ettevõtte disainiosakonnale, mille alusel loodi Enefit Volt stiilile vastav kasutajaliidese prototüüp.

Keerulisimaks ülesandeks oli arvustuste korrigeerimine viie aasta perioodile. Esmalt oli tarvis tehoülevaatuste vaheline periood viia võimalikult pikale ajavahemikule. Kuna eelnevalt võeti arvestusse sõiduauto kaks viimast tehnoülevaatust, võis vahemik jääda vaid mõne päeva pikkuseks, mis ei väljendanud täpselt kasutaja sõiduharjumusi. Selleks muudeti API-st andmete salvestamise loogikat nii, et arvestatakse sõiduauto esimese ning viimase olemasoleva ülevaatuse andmeid. Järgnevalt tuli välja arvutada kasutaja keskmiselt läbitud vahemaa viie aasta perioodil ning selle alusel arvutada nii sõiduauto kui ka elektriauto kütusetarbimise maksumus sellel perioodil.

Elektriauto kütusekulu arvutamisel tuli arvesse võtta ka elektrihinnale lisanduvad võrgu ning teenustasud ja kuvada need kasutajaliidesel. Võrgutasude arvutamisel tuli aluseks

võtta elektriauto tarbitud elekter antud perioodil ning leitud maksumusele tuli lisada igakuised võrgu teenustasud.

Kõige arendavam oli minu jaoks tagasiside tiimilt. Kuna varasemalt polnud ma isiklikult nii efektiivse tiimiga töötanud, oli töökeskkond uus. Õppisin tähepanelikumalt enda tehtud tööd kontrollima ning tiimi ülevaatused aitasid sellele kaasa.

Läbi lõputöö projekti õppisin Android rakenduste arenduse kohta, rakendama puhta koodi põhimõtteid ning tiimiga töötamist. Tiimiga töötamisel oli positiivseks asjaoluks, et omavahel aidatati ja arutati/selgitati üle üksteise tehtud tööd, mis andis hea ülevaate kogu rakendusest ja hoidis ülevaadet kogu projektist. Ennast hinnates tunnen, et õppisin kindlasti kirjutama kvaliteetsemat koodi, tehtud tööd rohkem kontrollima ning tiimisisese suhtluse olulisuse kohta.

Lisa 7 – Annabel Sulg eneseanalüüs

Teise etapi alguses oli raske leida motivatsiooni projekti edasiarendamiseks. Kuid see tekkis juba esimesel nädalal, kui saime koos meeskonnaga kokku, tegime äripoolele rakendusest demonstratsiooni ning analüüsisime koos tiimiga projekti hetkeseisu. Hea meeskond tagas kogu projekti vältel mõnusa töökeskkonna.

Bakalaureusetöös said minu peamisteks ülesanneteks elektriautode piltide lisamine, vigade parandamine, koodi refaktoorimine ning testide parandamine ja lisamine. Enamus neist ei olnud algselt planeeritud, vaid koostati projekti käigus.

Minu esimene ülesanne kujunes välja tiimikaaslase Yvonne ülesandest – lahendasime koos probleemi seoses piltide URL-i pärimisega API-st. Vea põhjuse otsimisel tuli välja, et probleem kujuneb arvatust suuremaks ning lahenduse leidmiseks on vaja teha põhjalik analüüs. Analüüsi käigus uurisin erinevaid tehnoloogiaid, mille abil on võimalik pilte Android rakenduses kuvada. Eeldusel, et piltide kuvamine toimub ainult juhul, kui kasutajal on Internetiühendus, jäid sõelale Glide ja Piccasso lahenduste kasutamine. Alustuseks lugesin mõlema dokumentatsioone ning vastavaid artikleid. Edasi implementeerisin need meie rakendusse ning alustasin keskkonna stabiliseerimisega, et testide tulemused oleksid võimalikud täpsed. Lõpetuseks testisin Glide ja Picasso efektiivsust ning koostasin selle kohta dokumentatsiooni.

Juhendajatega argumenteerimisel leidsime, et kõige parem on kasutada kolmandat lahendust – salvestada pildid lokaalselt rakendusse ning neid sealt kuvada. Kogu protsessi käigus õppisin palju: leidma parimat lahendust probleemile, erinevate tehnoloogiate testimist ja dokumentatsiooni kirjutamist.

Nagu ka esimese ülesande puhul, tuli ka teine ülesanne minule juhuslikult. Rakenduse arendamise käigus olid tekkinud vead, mida tuli lahendama hakata. Peamisteks põhjusteks olid valed kasutusviisid asünkroonsusel ning teksti töötlemisel. Elektriautode importimise ajal, lisati neid rippmenüüdesse, mis tekitas rakenduse esimesel käivitamisel mudelite rippmenüüsse topelt kirjeid. Probleemi lahendasin, kui lisasin asünkroonse tegevuse lõppedes elektriauto rippmenüüde lisamise loogika. Kirjeldatud lahendus lahendas ka vaikeväärtusena kuvatud auto probleemi. Teiseks peamiseks probleemiks oli Maanteeameti API-st andmete pärimisega. Päringuga saadud failist eemaldati alguses

kõik tühikud, teist meetodit kasutades eemaldati ainult tühikud, mis olid *string* väärtuse ees või taga. Selleks, et minu parandused ei tekitaks rohkem vigu, tuli neid analüüsida ning testida erinevate piirjuhtudega.

Teise ülesande lahendamisel sain selgeks vigade põhjuste otsimise, kasutades selleks *debugger*'it. Sain aru, kuidas analüüsida vigu ning neile parimat lahendust leida.

Kolmas ülesanne oli refaktoorida esimese lehekülje *front-end* osa ning kirjutada automaatteste klassidele, mille koodikattuvus oli alla 75%. Kuna refaktoorimisel liigutati erinevad komponendid erinevatesse klassides, ei olnud see väga keeruline. Raskem oli selgeks teha koodi osad, mis oli ammu kirjutatud ning neile teste kirjutada. Antud ülesannet tegin koos tiimikaaslase Yvonnega, koos arutledes ning probleeme lahendades jõudsime kiiresti lõpptulemuseni.

Lõputöö projekti käigus õppisin uurima põhjalikumalt uusi lahendusi ning tehnoloogiaid, mida projekti arendamisel kasutasin. Sain selgeks koodi kirjutamise teoreetilise poole nii testide kirjutamisel kui ka erinevate disainimustrite näol. Kogu protsessi vältel õppisin hindama teiste tiimiliikmete teadmisi, mis arendasid ka minu omasid.