TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Mathias Kivi 193815IADB

Veebirakenduse loomine elektriautode laadimiseks külalisena Eleporti laadimispunktides

Bakalaureusetöö

Juhendajad: Meelis Antoi
Magistrikraad

Ivari Horm
Bakalaureusekraad

Tallinn 2022

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: Mathias Kivi

16.05.2022

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua veebirakendus, mis võimaldab kasutaja-kontot loomata laadida elektriautot Eleporti laadimispunktides ühekordse kaardimaksega. Veebirakendus on mõeldud asendama Soome laadimisplatvormi Virta senist lahendust, et saavutada kontroll arendusprotsessi üle.

Kaardimaksete tegemiseks kasutatakse veebirakendust, kuna pikemas perspektiivis läheksid traditsioonilised kaardimaksed rohkem maksma: vaja oleks soetada makse-terminalid, millele lisanduksid kuumaksed.

Käesoleva bakalaureusetöö teema tekkis seoses Virta laadimisplatvormi vahendustasude kallinemise ja vähese arendustööga. Võimetus arendustöö aktiivsust mõjutada takistab kasutusel olevat lahendust kiiresti kasutajate tagasiside põhjal täiustada. Viimane on otsene oht Eleporti konkurentsivõimele.

Bakalaureusetöö eesmärk sai täidetud. Töö käigus valmis olemasoleva lahendusega võrdväärne veebirakendus, millele lisaks on loodud ka keelevalik. Enne kasutusel oleva lahenduse asendamist tuleks veebirakendust veelgi põhjalikumalt testida, et veenduda makselahenduse turvalisuses ja järjekindluses nii enne laadimisseansi alustamist kui ka selle ajal.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 7 peatükki, 17 joonist, 3 tabelit.

Abstract

Creating a Web Application for Charging Electric Cars as a Guest at Eleport Charging Stations

The aim of this bachelor's thesis is to create a web application that allows charging an electric car at Eleport charging stations with a single card payment without creating a user account. The web application is intended to replace the current solution that is based on the Finnish charging platform Virta in order to gain control over the development process.

Card payments are being made using a web application, because in the long run the traditional card payments would start to cost more: it would be necessary to purchase payment terminals to which monthly payments would be added.

The topic of this bachelor's thesis arose due to the increase in transaction fees and the low development work on the Virta charging platform. The lack of influence on development activity prevents the current solution from being quickly improved based on user feedback. The latter is a direct threat to Eleport's competitiveness.

The goal of the bachelor's thesis was accomplished. In the course of the work, a web application equivalent to the existing solution was developed, in addition to which localization was added. The web application should be tested further before replacing the existing solution, in order to ensure the security and reliablity of the payment solution both before and during the charging session.

The thesis is in Estonian and contains 29 pages of text, 7 chapters, 17 figures, 3 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusprogrammi poolt määratud reeglistik teiste rakendusprogrammidega suhtlemiseks [1]
binaarfail	kahendvormingus salvestatud fail [1]
BSON	*Binary JSON*, binaarfail andmete salvestamiseks JSON-vormingus [2]
CSS	*Cascading Style Sheets*, laadileht märgistuskeelse dokumendi välisilme kirjeldamiseks [3]
HTML	*Hypertext Markup Language*, veebilehtede loomiseks mõeldud märgistuskeel [1]
HTTP	*Hypertext Transfer Protocol*, andmevahetusprotokoll dokumentide ülekandmiseks Internetis [1]
interpreteerima	programmi ridahaaval käitama: iga lähtekoodi rida transleeritakse masinakeelde ja seejärel täidetakse [1]
JSON	*JavaScript Object Notation*, standard andmevahetuseks JavaScripti objektide kujul [3]
kahendvorming	viis sisu esitamiseks ainult kahe numbri 0 ja 1 abil [1]
kompileerima	kõrgkeeles kirjutatud programmi masinakeelde transleerima [1]
kõrgkeel	võimaldab kirjutada programmi nii, et see ei sõltu oluliselt konkreetse arvuti tüübist [1]
masinakeel	keel, millest arvutid aru saavad: kogu info on esitatud ainult nullide ja ühtede jadana [1]
NoSQL	*Not only SQL*, mitterelatsiooniliste andmebaaside kontseptsioon [3]
REST	*Representational State Transfer*, arhitektuuristiil API loomiseks [3]
SQL	*Structured Query Language*, relatsiooniliste andmebaaside haldamiseks ja kasutamiseks mõeldud päringukeel [3]
teek	valmiskompileeritud alamprogramm, mida teine programm saab kasutada [1]
teenusetaseme leping	leping teenusepakkuja ja kasutaja vahel, kus on kirjas lepingu kehtivusaja kestel oodatav teenusekvaliteet [1]
transleerima	ühest programmikeelest teise tõlkima [1]
XML	*Extensible Markup Language*, andmete struktureerimiseks mõeldud märgistuskeel [1]

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Elektrisõidukite levik Eestis on olnud aastate jooksul võrdlemisi väike. Aeglase leviku peamisteks põhjusteks on olnud nende oluliselt kõrgem hind, lühem sõiduulatus ja aeglane laadimine. Tänu tehnoloogia arengule on aga viimase 10 aastaga nimetatud probleemid oluliselt leevenenud ning elektrisõidukite arv on hakanud Eestis kiiresti kasvama.

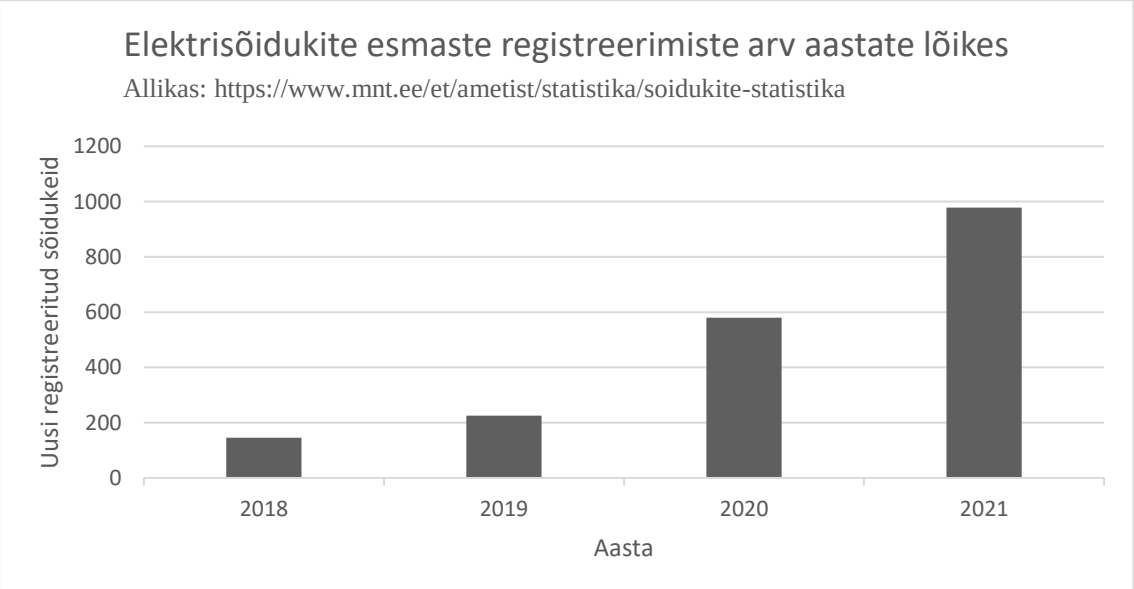
Uute kasutajate tulekuga on oodata üha rohkem tagasisidet ning konkurentsis püsimiseks tuleb muudatused võimalikult kiiresti ellu viia. Üle kolme aasta tegutsenud elektriautode laadimistaristu arendaja Eleport OÜ lõi tütarettevõtte Eleport Innovations OÜ, mille eesmärgiks on lisaks taristu ehitamisele arendada ka tarkvara elektriautode laadijate haldamiseks. Seni on Eleport ostnud sarnast tarkvara kolmandalt osapoolelt, kuid selle vähene arendustöö ja elektriauto kasutajate eeldatav kiire kasvu jätkumine on tekitanud Eleportil soovi saada kontroll arendusprotsessi üle.

Käesoleva lõputöö eesmärgiks on luua veebirakendus, mis võimaldaks laadida elektriautosid Eleporti laadimispunktides kasutajakonto olemasoluta. Hetkel kasutatakse Soome laadimisplatvormi Virta poolt pakutavat lahendust, kuid sellest soovitakse vabaneda peamiselt kahel põhjusel: vahendustasusid tõstetakse viiekordselt ja pakutavat teenust ei arendata piisavalt aktiivselt.

Töö käigus tutvustatakse elektrisõidukite laadimisega seotud probleemi, tuuakse välja levinumad laadimisplatvormid laadimisteenuse osutamiseks, esitatakse töö skoop ning määratakse funktsionaalsed ja mittefunktsionaalsed nõuded. Samuti põhjendatakse veebirakenduse väljatöötamiseks kasutatud tehnoloogiaid ja esitatakse nende võimalikud alternatiivid. Lõpetuseks tuuakse välja tehtud töö tulemused, sealhulgas testimine ja võimalikud edasiarendused.

2 Probleemi kirjeldus

Viimastel aastatel on hakatud Eestis järjest enam soetama traditsiooniliste fossiilkütustega sõidukite asemel elektrisõidukeid (vt Joonis 1). Näiteks aastal 2018 registreeriti vaid 146 uut elektrisõidukit, kuid aastal 2021 oli see arv juba 978. Selline trend viitab, et praegune entusiastide osakaal elektrisõidukite kogukonnas jääb peagi alla tavainimestele, mistõttu on oluline tagada, et laadimisprotsess oleks võimalikult kiire, lihtne ja mugav.



Joonis 1. Elektrisõidukite esmaste registreerimiste arv aastate lõikes

Euroopa Parlamendi ja nõukogu direktiivi kohaselt peaksid kõik üldkasutatavad laadimispunktid võimaldama elektrisõidukite kasutajatele ka ühekordset laadimisvõimalust ilma asjaomase elektritarnija või käitajaga lepingut sõlmimata [4]. Eestis leidub aga laadimisteenuse pakkujaid, kes nõuavad kasutajakonto registreerimist. See on tingitud Eesti seadusandlusest, mis kohustab järgima vaid ühtseid ohutusnõudeid ning autentimis-, kasutus- ja maksetingimused võivad seejuures erineda [5].

Eleport pakub anonüümse tasumise võimalust, kuna Tartu linnavalitsuse korraldatud riigihanke raames rajatud laadimispunktides oli nõutud avalikult kättesaadav ja lihtsalt kasutatav võimalus tasuda laadimisteenuse eest pangakaardiga. Rakendust kasutatakse ka

ettevaatusabinõuna, sest kohustuslikke maksevõimalusi reguleerivad seadused võivad välisturgudel erineda.

Probleem on Eleporti praegune võimetus kasutusel oleva lahenduse arendust mõjutada ja seega kasutajate tagasiside põhjal ei saa seda täiustada. Töö käigus valmib olemasolevaga võrdväärne lahendus, luues võimaluse uut veebirakendust edasi arendada täisväärtusliku elektriautode laadimisteenuse osutamiseks ka tulevikus.

2.1 Eksisteerivad lahendused

Järgnevalt tuuakse välja mõned levinumad tähistamata tooteid (ingl. k. *white-label products)* pakkuvad ettevõtted, mida Euroopa elektrisõidukite laadimistaristu ehitajad kasutavad. Tähistamata toode on ühe ettevõtte loodud toode või teenus, mida teised ettevõtted saavad osta koos oma visuaalse identiteediga, et see näeks välja nagu nemad oleksid selle loonud [6]. Selline lähenemine võimaldab pakkuda toodet või teenust lahendusega seotud taristu ja tehnoloogia haldamise pärast muretsemata.

2.1.1 has·to·be

has·to·be on 2013. aastal asutatud Austria laadimisplatvormi arendusettevõte, olles selle ala üks suuremaid tegijaid maailmas. Nende valmislahendust oma laadimisjaamade haldamiseks kasutavad näiteks Lääne-Euroopa laadimistaristu arendaja IONITY [7] ja eestimaine energiaettevõte Alexela [8]. Samuti tehakse koostööd autotootjatega nagu Audi, MAN ja Porsche, et integreerida laadimispunktide asukohad elektrisõidukite navigatsioonisüsteemidesse, püüdes saavutada Tesla Superchargeri võrgustikuga sama-väärset tulemust.

Tabel 1. has·to·be lahenduse (emobility.community) eelised ja puudused

Eelised	**Puudused**
Euroopa suurima ühisvõrgu (ingl. k. *roaming network*) haldamine [9]	Veebirakendus ei võimalda vaadata lehekülgi eesti keeles
Koostöös suurte autotootjatega integreeritakse laadimispunktid navigatsioonisüsteemidesse	Veebirakendus ei näita sisestatud tähise põhjal potentsiaalseid vasteid
	Veebirakendus nõuab tasumist ettemaksuna laetud kilovatt-tundide asemel
	Puudub veebirakendus kaardil laadimispunktide vaatamiseks

2.1.2 Virta

Virta on veel üks paljudest laadimisplatvormi arendusettevõtetest, mis asutati Soomes 2013. aastal. Tegemist on Euroopas suuruselt teise ühisvõrku pakkuva ettevõttega, ühendades enam kui 180 000 erinevate ettevõtete laadimispunkti [10]. Nende äri-klientideks on näiteks Eesti elektriautode laadimistaristut arendav Eleport ja Euroopa üks suurimaid energiaettevõtteid E.ON.

Tabel 2. Virta lahenduse (lae.ee) eelised ja puudused

Eelised	**Puudused**
Financial Timesi andmetel viimase kolme aasta jooksul olnud Euroopas kõige kiiremini kasvav elektrisõidukite laadimisplatvormi arendusettevõte [11]	Veebirakendus ei võimalda valida keelt
Veebirakendus salvestab ajutiselt eelnevalt lisatud maksekaardi	Veebirakendus näitab laadimisseansi ajal ainult laetud vatt-tunde
	Eraldi veebirakendused kaardil laadimispunktide vaatamiseks ja valimiseks

2.1.3 Driivz

Viimasena on välja toodud 2013. aastal asutatud Iisraeli laadimisplatvormi arendusfirma Driivz. Nende eripäraks on asjaolu, et Eesti suurim elektrisõidukite kiirlaadimisvõrgustik Enefit VOLT hakkas nende lahendust kasutama 2021. aasta maist, asendamaks 2013. aastast pärit aegunud lahendust [12].

Tabel 3. Driivzi lahenduse (account.evgo.com/findCharger) eelised ja puudused

Eelised	**Puudused**
Ühe veebirakendusega on võimalik kaardilt otsida ja valida sobiv laadija	Veebirakendus kasutab Google Maps otsingumootorit, mistõttu peale laadijate näidatakse ka laadija nimega kattuvaid asulaid
Veebirakendus näitab laadija kasutamise tipptunde	Ühisvõrgu lisateenus on ebapopulaarne, mistõttu tuleb välismaal reisides kasutada mitmeid kohalike teenusepakkujate veebirakendusi
Veebirakendus võimaldab operaatoril lisada laadijast pilte	

2.2 Uue lahenduse skoop

Eksisteerivad laadimisplatvormid näivad olevat hea valik alustavatele laadimisteenust pakkuvatele ettevõtetele, kuid populaarsuse ja klientide ootuste kasvuga muutuvad nad kiiresti piiravaks. Elektriautode valdkond on endiselt uus ja muutuv ning täielikult sobivat lahendust olemasolevate teenusepakkujate hulgas ei ole. Seetõttu on Eleport otsustanud uuele laadimisplatvormile ülemineku asemel luua nullist sobiv lahendus.

Tagamaks võimalikult sujuv üleminek kasutuses olevalt lahenduselt uuele, on käesoleva töö eesmärgiks luua vähim elujõuline toode, mis toetaks kohanduvat disaini ja võimaldaks teha toiminguid nagu laadija valimine, maksekaardi lisamine ja maksete sooritamine, laadimise alustamine ja lõpetamine ning kviitungite edastamine kliendi e-postile. Praeguse lahenduse täiendusena lisatakse ka keelevalik, mis võimaldab vaadata lehekülgi eesti, inglise, läti, leedu ja vene keeles. Madala taseme suhtlus laadijatega toimub eraldi tagarakenduse läbi, mis ei ole antud lõputöö skoobis.

3 Lahenduse analüüs

Olles paika pannud uue lahenduse skoobi, tuleb järgmise sammuna ära määratleda eesmärgi saavutamiseks vajalikud nõuded, sobivaim andmete hoiustamine ning rakenduse loomisel kasutatavad tehnoloogiad. Enne töö alustamist oli juba loodud veebi-rakendus kasutajakonto registreerimiseks, mistõttu hakkavad need rakendused kasutama ühist andmebaasi ja tagarakendust. Sellest tulenevalt keskendutakse andmebaasi ja tehnoloogiate valimise alapeatükkides rohkem sellele, kas valitud tehnoloogiad on mõistlikud või tuleks kaaluda nende asendamist.

3.1 Nõuete määramine

Nõuete määramise alapeatükis tuuakse välja vähima elujõulise toote arendamiseks vajalikud funktsionaalsed ja mittefunktsionaalsed nõuded. Nõuded selgitati välja ettevõtte tehnoloogiajuhi ja ärianalüütikuga koostöös. Uue lahenduse esialgsed nõuded olid määratletud põhimõttel, et kasutusel oleva lahendusega oleks vähemalt võrdväärne võimekus.

Parema ülevaate saamiseks on kasutatud MoSCoW prioritiseerimise meetodit [13], mille käigus liigitatakse nõuded nende olulisuse järgi. Jaotamiseks kasutatakse nelja kategooriat: peab olema (ingl. k. *must have*), peaks olema (ingl. k. *should have*), võiks olla (ingl. k. *could have*) ja ei hakka olema (ingl. k. *won't have*). Selle töö raames luuakse rakendus, mis vastab kategooriate "peab olema" ja "peaks olema" nõuetele. Kategoorias "ei hakka olema" nõudeid lähiajal välja ei töötata, kuid pole välistatud, et neid tulevikus üldse ei arendata.

Peab olema:

- Kasutajana soovin muuta keelt, et mõistaksin veebilehel kuvatavat sisu;
- Kasutajana soovin valida laadimispunkti tähise alusel, et saaksin valida oma elektrisõidukile lähima laadija;
- Kasutajana soovin valida laadimispistiku, et saaksin ühendada oma elektrisõiduki sobiva pistikuga;

- Kasutajana soovin lisada maksekaardi, et laadimise eest tasuda;
- Kasutajana soovin kustutada oma lisatud maksekaardi, et saaksin lisada teise maksekaardi;
- Kasutajana soovin alustada laadimist, et saaksin laadida oma elektrisõidukit;
- Kasutajana soovin lõpetada laadimise, et saaksin laadida sobiva summa eest;
- Veebirakendust saab kasutada ka aeglase mobiilside korral;
- Veebirakenduse töökiirus ei vähene märgatavalt tipptunni ajal;
- Veebirakendus on hea kättesaadavusega (töövõimeaeg 99%).

Peaks olema:

- Kasutajana soovin saada kviitungid e-postile, et saaksin vajadusel tõendada arve tasumist;
- Kasutajana soovin kasutada rakendust nii arvutis kui telefonis, et vajadusel saaksin alustada laadimisseansi ka oma tuttavale;
- Veebirakendus alustab laadimisseansi maksimaalselt 30 sekundi jooksul;
- Veebirakendus võtab kasutaja maksekaardilt raha ettemaksuna.

Võiks olla:

- Kasutajana soovin valida laadimispunkti QR-koodi alusel, et saaksin soovitud laadimispunkti lihtsamalt valida;
- Veebirakendus alustab laadimisseansi maksimaalselt 15 sekundi jooksul;
- Veebirakendus on väga hea kättesaadavusega (töövõimeaeg 99,9%).

Ei hakka olema:

- Kasutajana soovin vaadata kaardil lähedalasuvaid laadimispunkte, et saaksin vajadusel kiiresti minna teise laadimispunkti;
- Kasutajana soovin saada tõuketeavitusi (ingl. k. *push notifications*), et saaksin laadimisolekutega kursis olla.

3.2 Andmebaasi valik

Andmebaasid jagunevad kõige üldisemalt relatsioonilisteks ja mitterelatsioonilisteks andmebaasideks. Relatsioonilised andmebaasid kasutavad andmete käsitlemiseks struktuuripäringukeelt (ingl. k. *Structured Query Language*) ja eelnevalt koostatud andmeskeemi. [14] Andmeskeemi koostamine on aga märkimisväärne eeltöö ning ka hilisem muutmine on aeganõudev, keeruline ja vastutusrikas ülesanne, eriti juurutatud keskkonnas olevate andmete korral.

Elektrisõidukite laadimistaristu on pidevalt uuenev, mistõttu ei saa välistada vajadust andmeskeemi mitmekordseks muutmiseks. Lisaks on relatsioonilised andmebaasid vertikaalse skaleeruvusega ehk andmete ja töökoormuse kasvades on vaja mitme vähem võimsama serveri asemel ühte aina võimsamat serverit [15]. Koormuse jagamata jätmine mitme serveri vahel soodustab aga väiksemat töövõimeaega, sest ühe võimsa serveri rikke korral lakkab töötamast kogu süsteem. Eelneva põhjal võib järeldada, et relatsioonilised andmebaasid pole parim valik.

Mitterelatsioonilised ehk NoSQL (ingl. k. *Not only SQL*) andmebaasid said tuntuks ligi 15 aastat tagasi pilveteenuste, suurandmete ning veebi- ja mobiilirakenduste tulekuga. Tänapäeval on antud andmebaasi tüüp oma jõudluse, skaleeritavuse ja kasutuslihtsuse tõttu aina enam eelistatud. [16] Seda kinnitavad ka Stack Overflow Insights küsitlused: kui aastal 2017 kasutas elukutseliste arendajate seas MongoDB andmebaasi 18,5% [17], siis 2021. aastal oli see osakaal juba 28,03% [18].

Mitterelatsioonilisi andmebaase on neli peamist tüüpi: võti-väärtus paar, dokument, graaf ja lai-veerg [19]. Igal kategoorial on omad eelised ja piirangud ning leiavad kasutust erinevatel juhtudel, mistõttu on põhjendatud otsuse tegemiseks vaja mõista kõiki variante.

Võti-väärtus paar on kõige lihtsamat tüüpi andmebaas, kus igal ainulaadsel võtmel on mingi väärtus. Väärtused võivad olla mis tahes objekti kujul: arv, tekst või isegi uus võti-väärtus paar, kuid viimane muudab andmebaasi struktuuri palju keerulisemaks ja mõistlikum oleks selliste andmete korral kasutada teist andmebaasi tüüpi. Sellise andmebaasi suurimaks puuduseks on päringute tegemine vaid võtmete alusel: väärtuste hulgast sobiva vaste otsimine ei ole võimalik. [20] Menukamad andmebaasid on Redis, Memcached ja Oracle Berkeley DB.

Kõige sagedamini kasutatakse dokumendipõhiseid andmebaase, mis hoiavad andmeid dokumentides võti-väärtus paaridena. Ühise sisuga dokumendid jagatakse täiendavalt kogumikesse. Kogumikud on oma olemuselt samaväärsed relatsiooniliste andmebaaside tabelitega. Andmeid hoiustatakse dokumentides tavaliselt mõnes standardvormingus või -kodeeringus nagu JSON, XML või BSON [21]. Levinumad andmebaasid on MongoDB, Amazon DynamoDB ja Microsoft Azure Cosmos DB.

Üheks NoSQL andmebaasi tüübiks on ka graaf, kus kirjed esitatakse sõlmedena ja nendevahelised seosed servadena. Erinevalt relatsiooniliste andmebaasidega, kus tabelid on sõltumatud ja seosed tuleb hiljem arvutada, talletatakse seosed otse andmebaasi, mistõttu on seoste tuvastamine väga kiire. Seda tüüpi andmebaase on mõistlik kasutada, kui olemite vahel on palju seoseid, näiteks sõprussuhted sotsiaalvõrgustikes [22]. Tuntud andmebaasid on Neo4j, Virtuoso ja ArangoDB.

Lai-veerg on viimane laiemalt levinud mitterelatsioonilise andmebaasi tüüp. Andmete hoiustamiseks kasutatakse sarnaselt relatsioonilistele andmebaasidele tabeleid, veerge ja ridu, kuid suure erinevusena võib veeru nimi ja andmetüüp ridade lõikes erineda [23]. Lai-veerg tüüpi andmebaasid sobivad olukordades, kus on vaja hoiustada rohkelt andmeid, näiteks logimine või aegreapõhised nagu temperatuuri jälgimine [24]. Enam-levinud andmebaasid on Cassandra, HBase ja Google Cloud Bigtable.

Tulenevalt elektrisõidukite laadimise valdkonnas käsitletavatest andmetest peaks andmebaas toetama andmete vaheliste seoste salvestamist ka otse andmebaasi. Näiteks laadimispistik saab olla seotud vaid ühe laadijaga ja ei saa eksisteerida ilma laadijata. Samuti toimub suhtlus taga- ja eesrakenduse vahel REST API arhitektuuri järgides ehk JSON-vormingus. Neid asjaolusid arvestades on põhjust eeldada, et kõige mõistlikum andmebaasi tüüp elektrisõidukite laadimisplatvormi loomiseks on dokument.

Lõpetuseks võrreldakse Microsoft Azure Cosmos DB ja MongoDB dokumendipõhiseid andmebaase. Microsoft Azure Cosmos DB valiti elujõuliseks alternatiiviks Azure pilveteenuse tõttu. Hetkel majutatakse andmebaasi ja tagarakendust erinevates pilve-teenustes, mistõttu peavad andmebaas ja tagarakendus suhtlema üle avaliku võrgu, soodustades väiksemat töövõimeaega, lisades viivitust ja on ka turvarisk.

Praegu on kasutusel MongoDB andmebaas ja omal ajal valiti peamiselt kolmel põhjusel. Esiteks ei toetanud Cosmos DB programmeerimiskeelt Go, mida on kasutatud laadijatega

suhtleva tagarakenduse loomiseks. Teiseks töötab Cosmos DB ainult Microsoft Azure pilveteenuses, mida sel ajal taristu majutamiseks ei kasutatud. Viimaseks oli määrav ka oluliselt kallim hind (algne hinnapoliitika oli kulukam). MongoDB on tugev kandidaat ka oma tuntuse tõttu: suur kasutajaskond võib olla abiks keerulisemate päringute väljatöötamisel.

- **Microsoft Azure Cosmos DB** on dokumendipõhine andmebaas, mis ilmus 2017. aastal. Cosmos DB kasutab struktuuripäringukeelt ja eelneva kogemuse olemasolul on võimalik valida ka MongoDB andmebaasi päringukeeleks. Cosmos DB suudab jälgida andmebaasi töökoormust ning eraldada vastavalt vajadusele rohkem võimsust ja uusi masinaid, mistõttu on skaleeruvus väga hea. Cosmos DB on ka väga paindlik, kuna andmebaasi tüübiks on võimalik valida peale dokumendi ka võti-väärtus paar, graaf ja lai-veerg. Suurim puudus on hind: maksta tuleb nii andmebaasi mälukasutuse kui ka päringute eest [25].

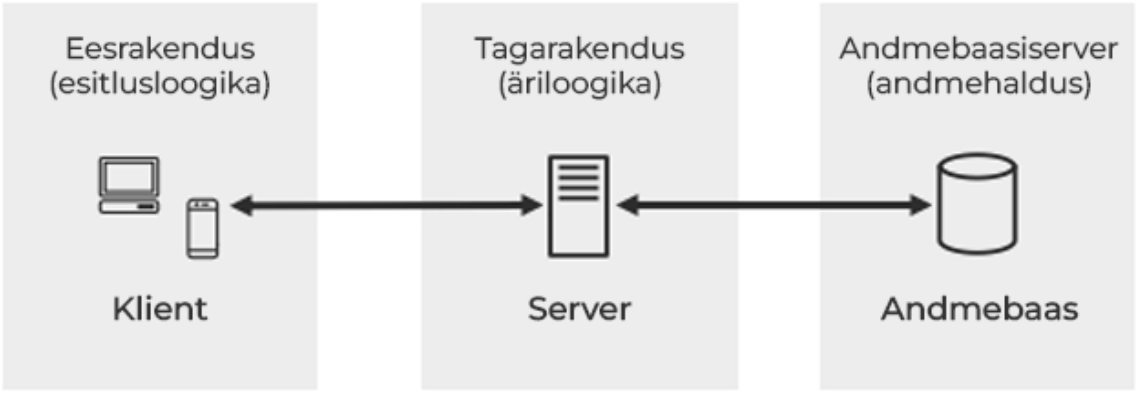
- **MongoDB** on enim kasutatav mitterelatsiooniline andmebaas, mille esmaväljalase oli 2009. aastal. MongoDB on samuti väga skaleeruv, kuid automaatseks skaleeruvuseks vajab täiendavat seadistamist, mis erineb pilveteenuspakkujate lõikes. MongoDB on tasuta kasutamiseks isegi äriliseks otstarbeks, seega rohkete päringute korral on pikemas perspektiivis odavam, kuna maksta tuleb vaid majutuse eest. Ühe puudusena MongoDB kasutab endale omast päringukeelt, mida peab eelneva kogemuse puudumisel õppima.

Microsoft Azure Cosmos DB ei toeta siiani ametlikult Go programmeerimiskeelt [26], kuid laadijatega suhtlev tagarakendus on plaanis ümber kirjutada C# programmeerimiskeeles, mistõttu on töö autori arvates põhjust kaaluda tulevikus MongoDB asendamist Cosmos DB vastu peamiselt kahel põhjusel. Esiteks tagab ühele pilveteenusele oluliste taristu osade majutamine väiksema viivituse, parema turvalisuse ja suurema töövõimeaja. Teiseks pakub Cosmos DB teenusetaseme lepingut, mis hõlmab olulisi andmebaasi omadusi nagu läbilaskevõime, reageerimisaeg ja käideldavus [27].

3.3 Tagarakenduse raamistiku valik

Hajussüsteemi loomiseks kasutatakse klient-server arhitektuuri, mis koosneb kolmest eraldiseisvast kihist: esitlusloogika, äriloogika ja andmehaldus (vt Joonis 2) [28]. See

mudel sobib kõige paremini, kuna elektrisõidukite laadimisplatvormi realiseerimine eeldab klientide ehk eesrakenduste loomist erinevatele platvormidele, mis kasutavad sama äriloogikat. Seetõttu on juurdepääs äriloogikale ja andmetele ühe liidese kaudu eelistatud.



Joonis 2. Kolmetasemeline klient-server arhitektuur

Peamiselt tuntakse REST ja SOAP printsiipe tagarakenduse API loomiseks [29]. Suure erinevusena põhineb REST arhitektuuristiil soovituste kogumil, kuid SOAP protokoll määrab teatud reeglid: andmevahetuseks XML-vormingu kasutamine ei ole soovitatav, vaid kohustuslik. Samuti ei salvestata veebilehitseja vahemällu SOAP API vastuseid, seega andmeid pärides tehakse tagarakendusele iga kord uus päring. [30] Nii MongoDB kui ka Microsoft Azure Cosmos DB ei toeta andmete salvestamist XML-vormingus [31] [32], mistõttu oleks vaja andmeid lugedes ja kirjutades neid täiendavalt teisendada. Eelnevast tulenevalt on REST API parem valik.

Tagarakenduse raamistiku valimiseks tuleb välja selgitada nõuded programmeerimiskeelele. Programmeerimiskeeled saavad olla kas dünaamiliselt (ingl. k. *dynamically typed*) või staatiliselt tüübitud (ingl. k. *statically typed*). Dünaamiliselt tüübitud keeltes nagu JavaScript, PHP ja Python pole kompileerimise ajal (ingl. k. *compile-time*) muutujate andmetüübid teada, mis võib põhjustada ootamatuid tõrkeid rakenduse töötamise ajal (ingl. k. *run-time*). [33] Näiteks mitterelatsioonilistel andmebaasidel puudub andmete käsitlemiseks andmeskeem ehk andmebaasiserver ei halda andmetüüpe. Sellest tulenevalt on andmetüüpide täideviimise vastutus rakendusel ja staatiliselt tüübitud programmeerimiskeeltele omane staatiline analüüs (ingl. k. *static analysis*) on väga kasulik, kuna vastuolust andmemudelis määratletud andmetüüpidega teavitatakse varakult. Eelneva põhjal võib järeldada, et dünaamiliselt tüübitud programmeerimis-

keeltel põhinevad tagarakenduse raamistikud nagu Django, Express.js ja Laravel ei ole kõige sobivamad valikud.

Staatiliselt tüübitud programmeerimiskeeltel põhinevate raamistike hulgas on kaks jätkusuutlikku valikut: ASP.NET Core ja Spring Boot. Järgnevalt tutvustatakse mõlemaid raamistikke ja tehakse järeldus, kas praegu kasutusel olev ASP.NET Core on asjakohane või tuleks kaaluda üleminekut alternatiivile.

- **ASP.NET Core** on C# programmeerimiskeelel põhinev raamistik, mis on loodud Microsofti poolt. Esmakordselt ilmus see 2002. aastal **ASP.NET** nime all ja toetas ainult Windowsi operatsioonisüsteemi. Aastal 2016 ilmus tasuta, avatud lähtekoodiga ja platvormiülene **ASP.NET Core** raamistik, mis põhjustas olulise kasutajaskonna kasvu. Tegemist on armastatud raamistikuga veebirakenduste loomiseks, kuna selle kasutuselevõtu on Microsoft muutnud võimalikult lihtsaks. Näiteks ametlik Visual Studio arenduskeskkond on tasuta kasutamiseks nii isiklikuks kui ka äriliseks otstarbeks[1], võimaldades arendustöö alustamiseks vajalikud sammud nagu tarkvara allalaadimine sooritada kiiresti ja mugavalt otse arenduskeskkonnast. Samuti pakub ASP.NET Core mitmeid valikuid ees-rakenduse tegemiseks [34], mis võimaldab kasutada ühist programmeerimiskeelt ja tavasid.

- **Spring Boot** on Java programmeerimiskeelel põhinev raamistik veebirakenduste loomiseks, mille esmaväljalase oli 2002. aastal. Spring Booti hea omadus on võimalus projekti eelseadistada veebipõhises tööriistas Spring Initializr, kus saab valida soovitud teegid vastavalt planeeritud rakendusele, mis aitab vältida eba-vajalike sõltuvuste rakendusse lisamist ja seeläbi vähendada rakenduse suurust. Suurim nõrkus on ametliku arenduskeskkonna puudumine.

Töö autor ei näe vajadust vahetada ASP.NET Core raamistik välja peamiselt kahel põhjusel. Esiteks pakub Spring Boot raamistikuga võrreldes ASP.NET Core kiiremat reageerimisaega, tarbides samal ajal ka vähem ressursse nagu protsessor ja mälu [35]. Teiseks võimaldavad mõlemad valikud rakendada tähtsamaid programmeerimis-

[1] Organisatsioonides, kus on vähem kui 250 arvutit või mille aastatulu on alla ühe miljoni USA dollari, saavad Visual Studio Community integreeritud arenduskeskkonda kasutada kuni viis inimest [36].

tehnikaid nagu objekt-relatsiooniline vastendus (ingl. k. *object-relational mapping*) ja sõltuvuste süstimine (ingl. k. *dependency injection*).

3.4 Eesrakenduse raamistiku valik

Nutiseadmele mõeldud eesrakendus on üldjuhul mobiilirakenduse vormis. Eesrakenduse loomine mobiilirakendusena on õigustatud, kui on täidetud vähemalt üks tingimus: rakenduse kasutamise vajadus on eeldatavalt järjepidev või on plaanis kasutada seadmepõhiseid toiminguid nagu asukoha määramine ja teadete vastuvõtmine [37]. Kumbki kriteerium ei kehti loodavale rakendusele, seega pole eesrakenduse loomine mobiilirakendusena põhjendatud.

Hoolimata sellest on veebirakenduse sihtrühmaks mobiilseadmete kasutajad, mistõttu soovitakse mobiilirakendusega võimalikult sarnase kasutajakogemuse saavutamiseks luua üheleherakendus (ingl. k. *single-page application*). Traditsioonilise veebirakendusega võrreldes võimaldab üheleherakendus uuendada veebilehe sisu osaliselt ehk veebilehte sirvides laetakse terve veebilehe asemel alla vaid tükk veebilehest. Selline lähenemine tagab väiksemad andmemahud, kiiremad laadimisajad ja sujuvama kasutuskogemuse. Hetkel on eesrakenduste loomiseks kasutusele võetud Blazor WebAssembly eesrakenduse raamistik. Järgnevalt uuritakse, millised võimalused on üheleherakenduse loomiseks ja kas Blazor WebAssembly kasutamist tasub jätkata.

Üheleherakenduse loomiseks kasutatakse üldiselt raamistikke nagu Angular, React või Vue, mis põhinevad JavaScripti programmeerimiskeelel. JavaScript on interpreteeritud programmeerimiskeel ehk lähtekoodi tõlgendamine ja käivitamine sõltub veebilehitseja interpretaatorist, mis võib põhjustada veebilehitsejate lõikes erinevusi [38]. Erisuste vähendamiseks on loodud ECMAScripti eristuskirjad, mis kirjeldavad versioonile vastavat JavaScripti võimekust. Tegelikkuses juurutavad veebilehitsejad erinevaid ECMAScripti versioone ja laiahaardelist rakendust luues ei ole võimalik uusimaid JavaScripti tavasid kasutada. Sellise veebilehitsejate vahelise ebajärjekindluse ja alapeatükis 3.2 mainitud probleemide tõttu ei ole JavaScripti programmeerimiskeele kasutamine eesrakenduse loomiseks parim valik.

Üheleherakendust on võimalik luua ka WebAssembly tehnoloogial. WebAssembly on 2017. aastal loodud programmeerimiskeel, mida kaasaegsed veebilehitsejad suudavad

käivitada. Erinevalt JavaScriptist on WebAssembly kompileeritud kood, mis tagab suurema jõudluse ja parema järjepidevuse veebilehitsejate lõikes.

Kõige tuntum WebAssembly eesrakenduse raamistik on Blazor, mille esmaväljalase oli 2018. aastal. Tegu on täisväärtusliku lahendusega üheleherakenduste loomiseks, mis võimaldab lisaks tavapärastele veebitehnoloogiatele nagu CSS (ingl. k. *Cascading Style Sheets*), HTML (ingl. k. *Hypertext Markup Language*) ja JavaScript kasutada ka C# programmeerimiskeelt. Samuti on toetatud programmeerimistehnikad nagu sõltuvuste süstimine, muutes olekuhalduse (ingl. k. *state management*) väga lihtsaks.

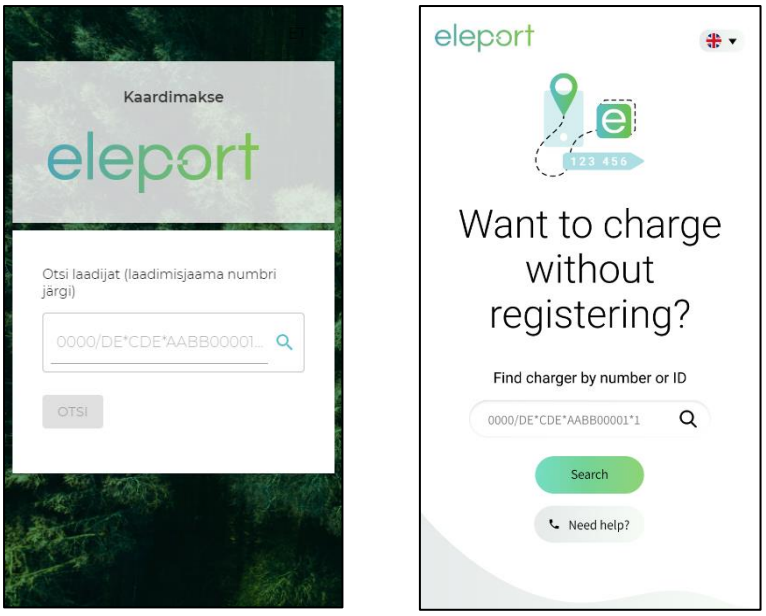
Töö autori hinnangul on mõistlik jätkata Blazor WebAssembly eesrakenduse raamistiku kasutamist. Lisaks eelnevalt mainitud eelistele võimaldab Blazori kasutamine jagada tagarakendusega ühist programmeerimiskeelt, andmemudelit ja koodibaasi (ingl. k. *code base*). Viimane lubab taaskasutada näiteks sisendi õigsuse kontrollimiseks kasutatud koodi.

4 Veebirakenduse disain

Veebirakenduse kasutajaliidese kujundus oli Figma veebirakenduses ettevõtte poolt loodud prototüübina, mille üldine ülesehitus põhines olemasoleval lahendusel. Üheks kriteeriumiks oli kohanduv disain, seega olid kavandid esitatud nii arvuti- kui ka mobiilivaates. Selles peatükis keskendutakse mobiilivaadetele, kuna veebirakenduse sihtrühmaks on mobiilikasutajad.

Kujunduse loomiseks kasutatakse ainult kaskaadlaadistikke. Raamistikke nagu Bootstrap ja Tailwind ei kasutata kahel põhjusel. Tailwind ei ole ahvatlev, kuna suure hulga klassidega risustatakse HTML dokumente. Bootstrap pole jätkusuutlik, kuna kujundus-elementidel on teatud välimus ja kohandatud kujunduse saavutamiseks on ikkagi vaja kasutada kaskaadlaadistikke.

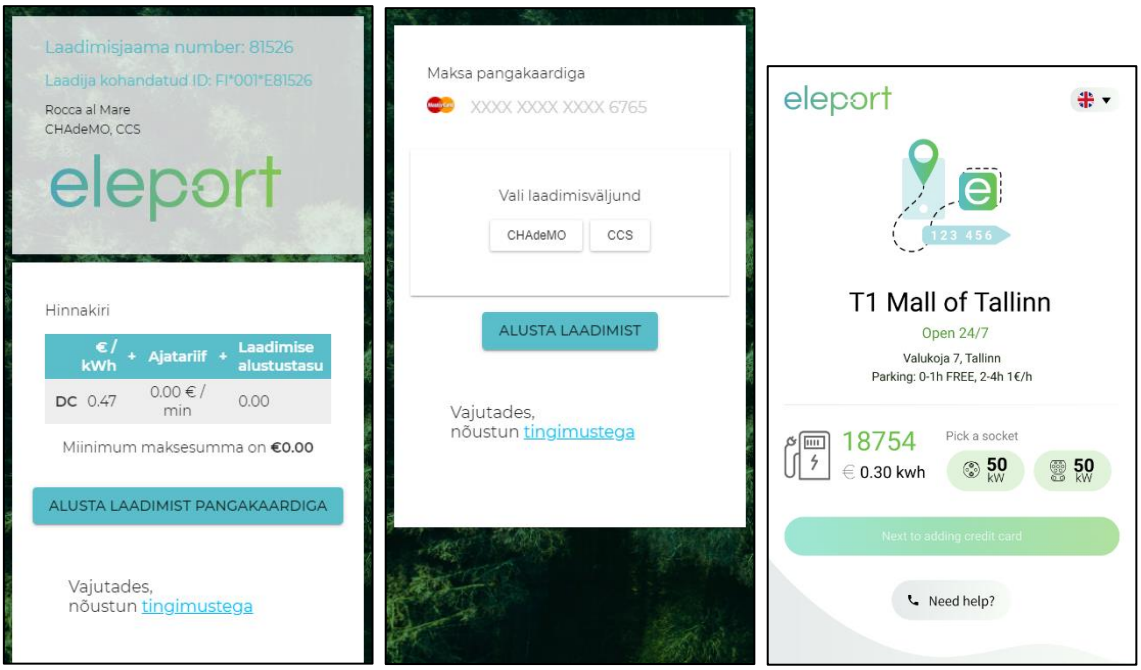
Järgnevalt võrreldakse vana ja uue lahenduse olulisemaid vaateid ning tuuakse välja kitsaskohad olemasoleva lahenduse kasutajakogemuses. Avalehel on näha olemasoleva lahendusega võrreldes kahte täiendust: keele valimine ja viide kasutajatoele (vt Joonis 3). Arvestades elektriautode ja nende laadimise uudsust, on küsitav klienditeeninduse viite puudumine olemasolevas lahenduses. Uus lahendus järgib ka Eleporti üldist värvi-lahendust, aidates arendada Eleporti kaubamärki.



Joonis 3. Laadija valimine praeguses (vasakul) ja uues lahenduses (paremal)

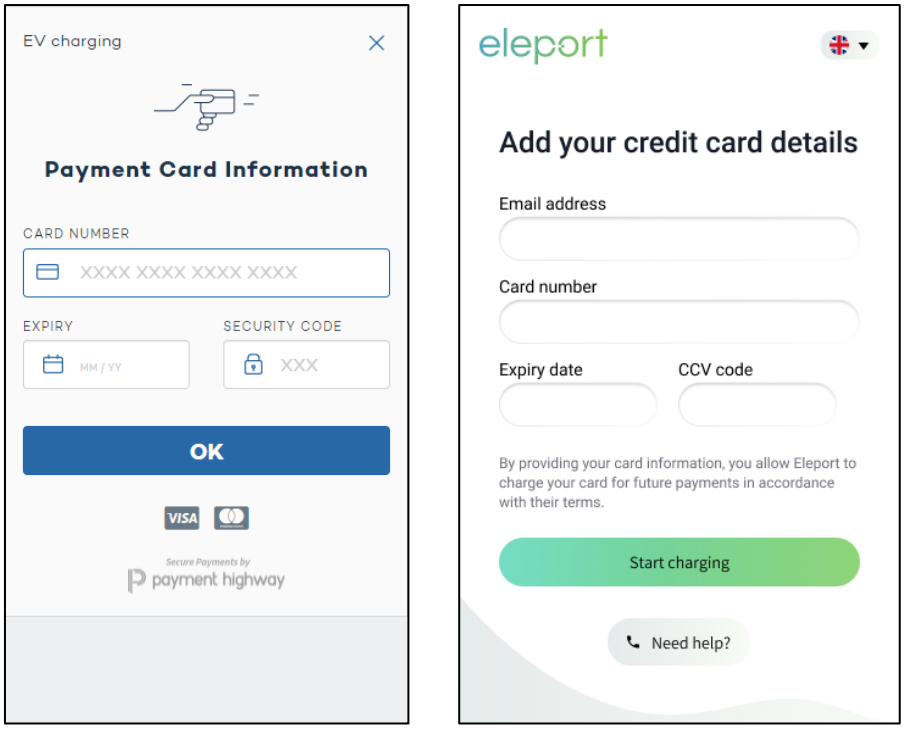
Laadimispistiku valimise lehekülgede võrdluses on hästi näha, kuidas on võimalik hoida kokku ruumi ja vältida ebavajalikke lisasamme (vt Joonis 4). Teave nagu laadija asukoht, hinnakiri ja saadaval olevad pistikud võiksid olla kõik ühel leheküljel, et saada ühtne ülevaade.

Praeguses lahenduses eristatakse laadimispistikuid vaid teksti põhjal. Inimesed, kellel on elektriautodega vähe või üldse mitte kogemust, ei suuda tõenäoliselt CCS ja CHAdeMO pistikutel vahet teha, mistõttu on uuel lahendusel pistikud piltidena kujutatud.



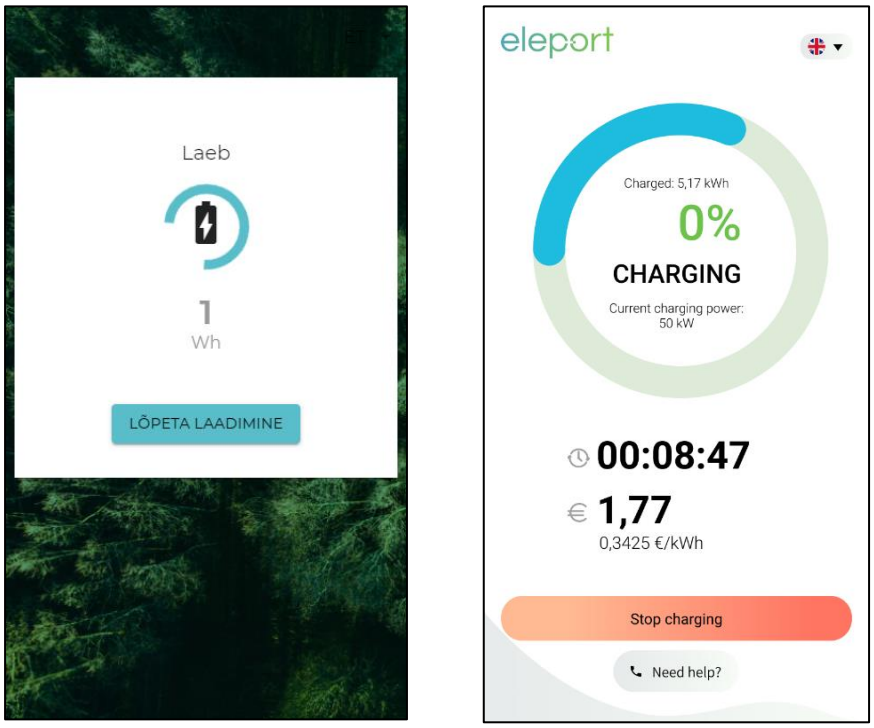
Joonis 4. Laadimispistiku valimine praeguses (vasakul ja keskel) ja uues lahenduses (paremal)

Järgnevalt on toodud vaated maksekaardi andmete lisamiseks (vt Joonis 5). Olulise muudatusena ei suunata enam maksekaardi andmete lisamiseks eraldi veebilehele. Selline lähenemine loob kasutajas suurema turvatunde ja näeb ka meisterlikum välja. Tulevikus kviitungite saamiseks määrab kasutaja maksekaardi lisamisel ka oma e-posti aadressi.



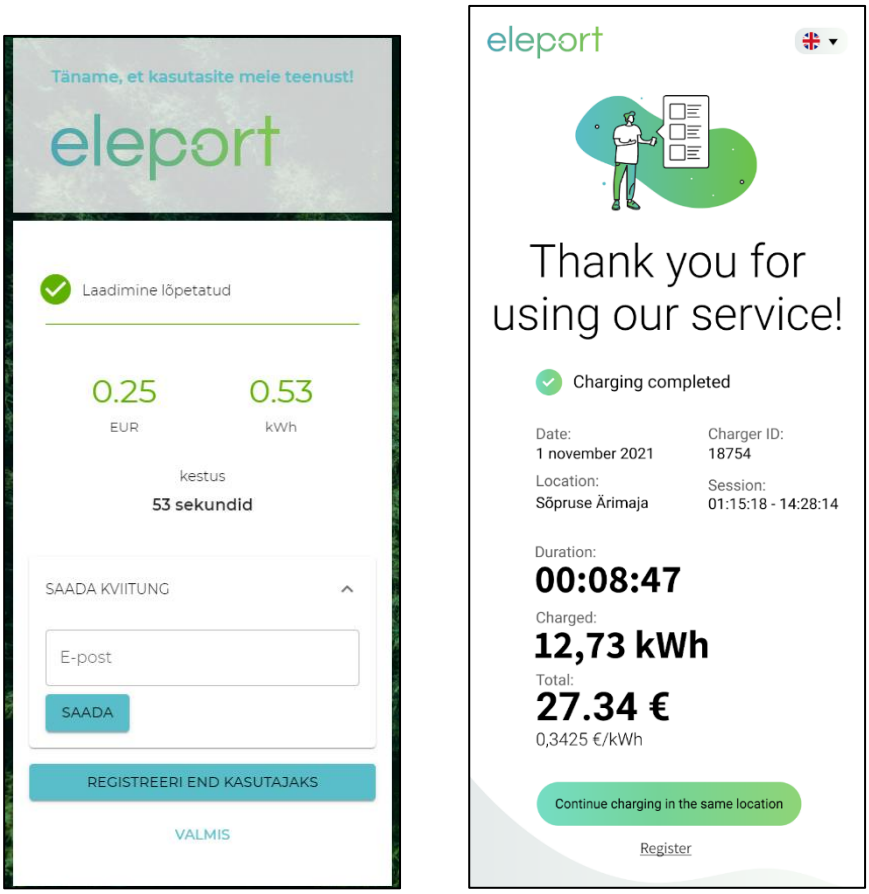
Joonis 5. Maksekaardi lisamine praeguses (vasakul) ja uues lahenduses (paremal)

Laadimisseansi vaadet muudeti samuti märkimisväärselt (vt Joonis 6). Lisaks laetud vattitundide arvule hakatakse tulevikus näitama ka hetkelist laadimiskiirust, laadimisseansi pikkust, eeldatavat maksumust ja võimalusel ka auto akutaset, kui laadija seda edastab. Praeguses versioonis hakkab ringil olema ketramise animatsioon, kuid võimalik oleks tulevikus auto akuteabe olemasolul kasutada ringi aku täituvuse näitamiseks.



Joonis 6. Laadimisseansi vaade praeguses (vasakul) ja uues lahenduses (paremal)

Kokkuvõtte vaade jäi suures osas puutumata (vt Joonis 7). Suurim erinevus on kviitungi saatmise komponendi eemaldamine – kviitung saadetakse automaatselt maksekaardi lisamise etapis määratud e-posti aadressile. Laadimisseansi kohta on ka pisut rohkem teavet, näiteks kus ja millal laadimist alustati ning mis oli kilovatt-tunni hind.



Joonis 7. Kokkuvõtte vaade praeguses (vasakul) ja uues lahenduses (paremal)

Vaatamata oma pikale ajaloole on Virta laadimisplatvorm siiani üsna lakooniline nii väljanägemiselt kui funktsionaalsuselt. Lahenduse loomisel on kahtlemata oluline selle toimimine, kuid pisidetailid jäetakse sageli tähelepanuta. Ettevõtte lehekülgede järjekindel väljanägemine ja meeldiv kasutajakogemus on aga peamised tegurid, mis aitavad tekitada kasutajas usaldust.

5 Veebirakenduse arendus

Selles peatükis selgitatakse täpsemalt veebirakenduse arenduse käigus välja töötatud andmemudelit ning taga- ja eesrakenduse ülesehitust. Andmemudel on koostatud veebirakendusega diagrams.net, mis on tasuta avatud lähtekoodiga veebirakendus erinevate jooniste tegemiseks [39]. Serveri- ja kasutajapoolne lahendus on loodud integreeritud arenduskeskkonnas Visual Studio 2022 Community, mis on Microsofti poolt loodud tasuta[1] ja ametlik lahendus .NET raamistikul põhinevate rakenduste tegemiseks [40].

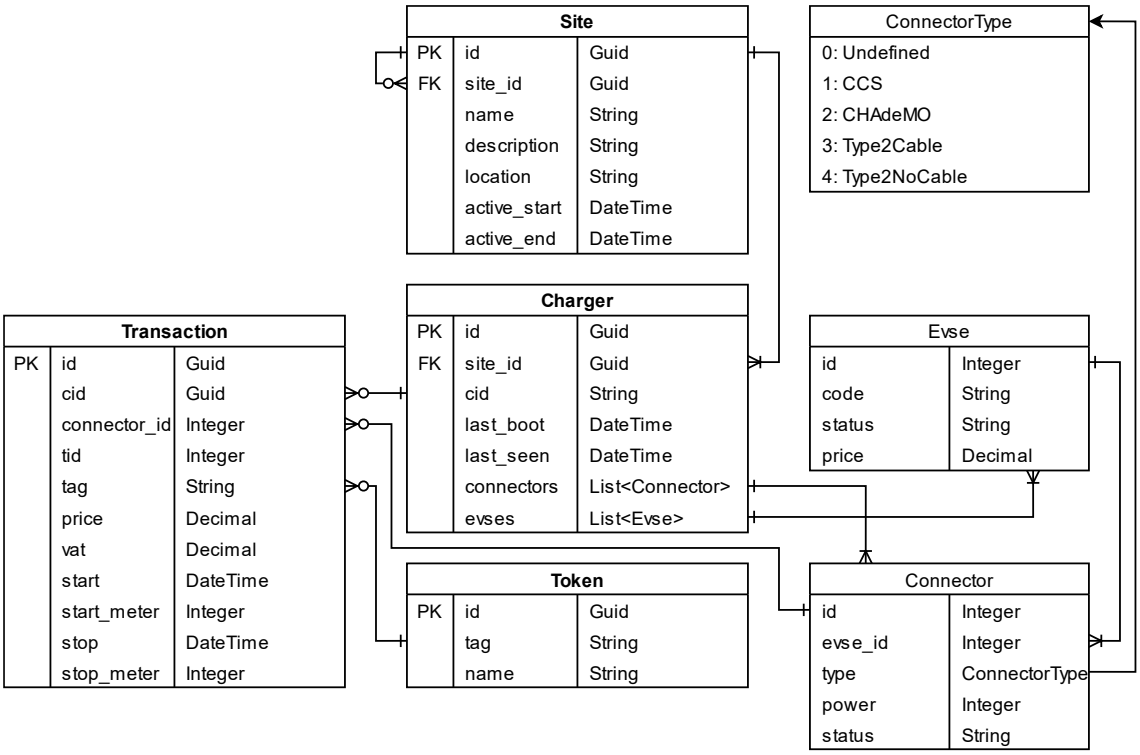
5.1 Andmemudeli struktuur

Vaatamata sellele, et mitterelatsioonilised andmebaasid ei nõua andmeskeemi [41], on sellegipoolest kasulik koostada andmemudel. Läbimõeldud andmemudel aitab varakult tuvastada vajalikud andmed ja nendevahelised seosed. Samuti on uutel töötajatel lihtsam aru saada arendatavast infosüsteemist.

Järgnevalt on välja toodud lahendusega seotud andmebaasi andmemudel (vt Joonis 8). Erinevalt relatsioonilistest andmebaasidest, kus andmeskeemil kujutatakse tabeleid ja veerge, on dokumendipõhise andmebaasi andmemudelil dokumendid ja väljad. Segaduse vähendamiseks on välja jäetud selle tööga mitteseonduvad dokumendid, nagu kasutajad ja lepingud.

Laadimisseansi haldamiseks kasutatakse kahte tagarakendust. Üks suhtleb otse laadijaga (väljaspool lõputöö skoopi) ja teine rakendab äriloogikat, kuid mõlemad kasutavad sama andmebaasi. Sellest tulenevalt on dokumentides esindatud mõlemale tagarakendusele vajalikud väljad. Näiteks laadija (ingl. k. *Charger*) ja tehingu (ingl. k. *Transaction*) dokumentidel on kaks tähise (ingl. k. *identifier*) välja. Väljad *cid* ja *tid* on tekitatud laadijate püsivara poolt ning *id* on andmebaasi poolt genereeritud tähis.

[1] Organisatsioonides, kus on vähem kui 250 arvutit või mille aastatulu on alla ühe miljoni USA dollari, saavad Visual Studio Community integreeritud arenduskeskkonda kasutada kuni viis inimest [36].



Joonis 8. Kuvatõmmis veebirakenduse andmebaasi andmemudelist

Dokumendid võivad andmebaasis olla normaliseeritult ja mittenormaliseeritult. Mittenormaliseeritud ehk manustatud (ingl. k. *embedded*) on dokumendid, mis saavad olla seotud ainult ühe dokumendiga ja ei saa eksisteerida üksi. Näiteks laadimispistikud on manustatud otse laadija dokumenti, kuna nad saavad olla seotud vaid ühe laadijaga ega saa eksisteerida ilma laadijata, seega laadija dokumenti kustutades eemaldatakse ka vastavad laadimispistikud.

Paljudes üks-mitmele suhetes esinevad dokumendid on normaliseeritud. Näiteks mitme laadijaga asukohti (ingl. k. *Site*) on tõenäoliselt mitu tükki. Seetõttu on normaliseeritud asukoha dokument, et iga laadija ei peaks sisaldama kogu asukoha teavet, vaid saab viidata ainult asukoha *id* väljale (vt Joonis 9).

Üheks põhidokumendiks on *Charger*, mis esindab ühte füüsilist laadijat. Laadija on omakorda jagatud üheks või enamaks virtuaalseks laadijaks, mida esindab *Evse* (ingl. k. *Electric vehicle supply equipment*) dokument. Igal laadijal on ka nimekiri füüsilistest laadimispistikutest (ingl. k. *connectors*), millest igaüks on seotud ühe virtuaalse laadijaga. Laadimispistikud võivad olla erinevat tüüpi, näiteks tuntumad on Euroopa CCS ja Jaapani CHAdeMO standardid. Seetõttu on *Connector* dokumendil väli *type*, mis sisaldab täisarvu. Täisarvude tähendus on esitatud *ConnectorType* dokumendiga.

Mittenormaliseeritud

Charger									
id	*site_name*	*site_location*	*site_active_start*	*site_active_end*	*cid*	*last_boot*	*last_seen*	*connectors*	*evses*
1	T1 Tallinn	Peterburi tee 2, ...	00:00	00:00	F9OK7L	2022-02-18 23:56	2022-03-29 19:32	{ ... }	{ ... }
2	Rocca al Mare	Paldiski mnt 102, ...	21:00	06:00	A4N5MT	2022-02-22 22:22	2022-03-29 19:29	{ ... }	{ ... }
3	T1 Tallinn	Peterburi tee 2, ...	00:00	00:00	RVE30J	2022-03-11 15:31	2022-03-29 19:44	{ ... }	{ ... }

Normaliseeritud

Site				
id	*name*	*location*	*active_start*	*active_end*
1	T1 Tallinn	Peterburi tee 2, ...	00:00	00:00
2	Rocca al Mare	Paldiski mnt 102, ...	21:00	06:00

Charger						
id	*site_id*	*cid*	*last_boot*	*last_seen*	*connectors*	*evses*
1	1	F9OK7L	2022-02-18 23:56	2022-03-29 19:32	{ ... }	{ ... }
2	2	A4N5MT	2022-02-22 22:22	2022-03-29 19:29	{ ... }	{ ... }
3	1	RVE30J	2022-03-11 15:31	2022-03-29 19:44	{ ... }	{ ... }

Joonis 9. Mittenormaliseeritud ja normaliseeritud asukoha dokument

Füüsiline laadija peab ka kuskil asuma, mistõttu asukohta esindamiseks on loodud *Site* dokument, millega *Charger* dokument on seotud. Asukohal võib olla ka alamasukohti, näiteks parkimismaja erinevad korrused. Ühte laadimisseansi ehk tehingut esindab *Transaction* dokument. Tehingu alustamiseks kasutatakse žetooni, mis on esindatud *Token* dokumendina. Žetoon saab olla füüsiline nagu RFID-kaart või virtuaalne, mis on tagarakenduse poolt loodud.

Järgmistes alapeatükkides selgitatakse serveri- ja kasutajapoolset lahendust. Mõlemad lahendused kasutavad andmemudeli täitmiseks ühiseid andmemudeli klasse. Klassid on loodud vastavalt andmemudelis esindatud dokumentidele, väljadele ja väljade andmetüüpidele. Samuti on väljadele lisatud valideerimisreeglid (vt Joonis 10).

```
public class Charger {
    [BsonId]
    20 references
    public Guid Id { get; set; }

    [Required]
    [BsonElement("site_id")]
    16 references
    public Guid SiteId { get; set; }

    [Required]
    [BsonElement("cid")]
    1 reference
    public string ChargerId { get; set; }

    [ReadOnly(true)]
    [BsonElement("last_boot")]
    2 references
    public DateTime LastBoot { get; set; }

    [ReadOnly(true)]
    [BsonElement("last_seen")]
    3 references
    public DateTime LastSeen { get; set; }

    [BsonElement("connectors")]
    10 references
    public List<Connector> Connectors { get; set; } = new();

    [BsonElement("evses")]
    6 references
    public List<Evse> Evses { get; set; } = new();

    0 references
    public override string ToString() {
        return ChargerId;
    }
}
```

Joonis 10. Andmemudeli klass, mis võimaldab rakendustel teostada andmemudelit

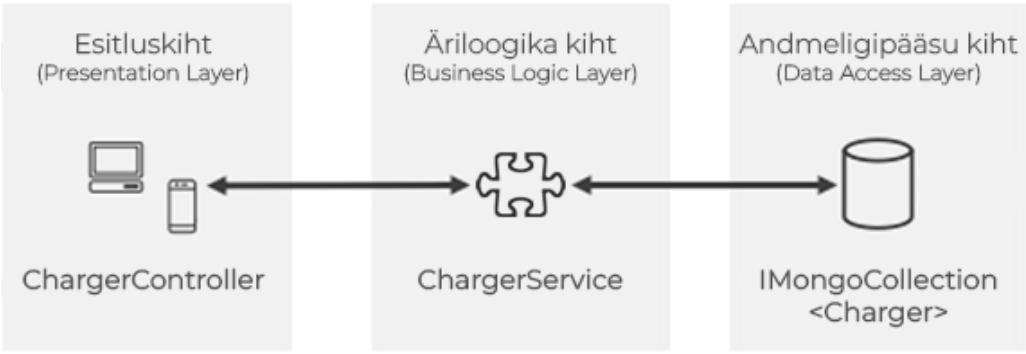
5.2 Serveripoolne lahendus

Veebirakenduse serveripoolse lahenduse peamisteks ülesanneteks on andmehaldus ja rakendusliidese päringute töötlemine. Lahendus on loodud C# programmeerimiskeelel põhineva ASP.NET Core veebiraamistikuga, mis on .NET raamistiku edasiarendus veebirakenduste loomiseks. Põhiliste serveripoolsete talitluste arendamiseks on vajalikud tööriistad ja teegid kas sisseehitatud või hõlpsasti paigaldatavad NuGet paketihalduriga.

5.2.1 Struktuur

Modulaarsuse saavutamiseks on tagarakendus jaotatud kolmeks kihiks (vt Joonis 11):

- Esitluskiht (ingl. k. *Presentation Layer*);
- Äriloogika kiht (ingl. k. *Business Logic Layer*);
- Andmeligipääsu kiht (ingl. k. *Data Access Layer*).



Joonis 11. Tagarakenduse kolmekihiline arhitektuur koos näidisklasside nimedega

Kolmetasandilises arhitektuuris on iga kiht sõltumatu ja täidab teatud ülesannet. Iga kihti saab täiendada teisi kihte muutmata, mis võimaldab rakendust kiiremini arendada, pakub rohkem paindlikkust ja muudab koodi paremini hallatavaks [42]. Kihtidevaheline sõltumatus saavutatakse sõltuvuste süstimisega (ingl. k. *dependency injection*), mis põhineb abstraktsioonide vastu programmeerimise kontseptsioonil. Abstraktsioon saavutatakse jõustades (ingl. k. *implement*) madalama kihi liidest (ingl. k. *interface*), mis võimaldab kasutada madalama kihi meetodeid, kuid ei avalikusta nende lahendust. [43]

Kõige madalam tase on andmeligipääsu kiht, mille eesmärgiks on võimaldada suhtlus andmebaasiga. Andmeligipääsu kihis on kogumikud (ingl. k. *collections*), mis sisaldavad andmehalduseks vajalikke põhitoiminguid ehk andmete sisestamist, lugemist, muutmist ja kustutamist. Kogumik on klass MongoDB andmebaasi teegis, millest luuakse objekt äriloogika kihis teenuste poolt, kasutades andmemudeli klassi (vt Joonis 12).

```
public class ChargerService {
    private readonly IMongoCollection<Charger> _chargers;

    public ChargerService(IDatabaseSettings settings) {
        var client = new MongoClient(settings.ConnectionString);
        var database = client.GetDatabase(settings.DatabaseName);
        _chargers = database.GetCollection<Charger>(
            settings.ChargerCollectionName
        );
    }

    . . .
}
```

Joonis 12. Kogumike loomine äriloogika kihis teenuste poolt

Andmeid töötleb edasi äriloogika kiht, milles on teenused (ingl. k. *services*) ja käitlejad (ingl. k. *handlers*). Teenused on kasutusel andmebaasiga suhtlemiseks, hõlmates andmete sisestamisel õigsuse kontrollimist ja andmete pärimisel sobivasse vormi viimist. Näiteks

laadija pärimisel seotakse ka vastav asukoha dokument, mis võimaldab lugeda laadija asukoha lahtiolekuaegu. Käitlejad on loodud kolmandate osapooletega suhtlemiseks, näiteks maksete tegemiseks Stripe maksete töötlemise platvormi kasutades.

Esitluskiht on kolmetasandilise arhitektuuri kõrgeim kiht, milles on kontrollerid ees-rakendustega suhtluse võimaldamiseks. Kontrollerid töötlevad sissetulevaid REST API päringuid, kasutades andmehalduseks äriloogika kihi teenuseid või käitlejaid. Igale päringule saadetakse vastuseks HTTP olekukood (ingl. k. *status code*) koos äriloogika kihi poolt tagastatud tulemusega.

5.2.2 REST arhitektuuristiil

REST on arhitektuuristiil rakendusliideste loomiseks, mis kasutab andmete käsitlemiseks HTTP päringuid. Andmeedastus toimub tavaliselt JSON-vormingus, kuid võimalikud on ka XML- ja HTML-vormingud. Kasutusel olev vorming on määratud HTTP päise (ingl. k. *header*) väljal *Content-Type*, mida nii serveri- kui kasutajapoolne lahendus kasutab andmete lugemiseks.

Andmete käsitlemiseks tehakse HTTP päring kindlale lõpp-punktile koos HTTP verbiga:

- *GET* (saama): andmete lugemiseks, vastused hoitakse võimalusel vahemälus;
- *POST* (postitama): andmete sisestamine;
- *PUT* (panema): andmete uuendamine;
- *DELETE* (kustutama): andmete kustutamine.

5.2.3 Turvalisus

Tagarakenduse turvalisuse parandamiseks on kasutatud HTTPS (ingl. k. *Hypertext Transfer Protocol Secure*) protokolli, mis on krüpteeritud andmeedastusega HTTP edasiarendus [44]. HTTPS protokoll võimaldab kasutajatel ohutult edastada tundlikke andmeid nagu pangakaardi üksikasju. HTTPS protokollist on aegamööda saanud uus standard, kuna see tagab ka andmete terviklikkuse [45].

Volitamata laadimisseanside tagamiseks on kasutatud võtmesüsteemi. Esmalt saadetakse laadimisseansi alustamiseks tagarakendusele päring koos Stripe maksetöötlusplatvormi maksemeetodi tähisega (maksekaardi lisamise etapis hangitud), misjärel tehakse maksekaardile broneering. Ettemakse õnnestumisel luuakse ainulaadne võti, mis seostatakse

maksemeetodi tähisega. Seejärel tagastatakse võti kasutajapoolsele lahendusele, mille alusel on võimalik laadimisseanss lõpetada. Tulevikus oleks vaja lisada serveripoolsele lahendusele talitlus, mis võimaldaks maksekaardi nelja viimase numbri alusel laadimisseansse hallata ka teistes seadmetes.

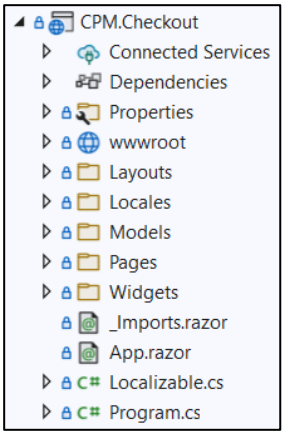
5.3 Kasutajapoolne lahendus

Kasutajapoolse lahenduse peamiseks ülesandeks on võimaldada elektriautode kasutajatel Eleporti ärilahendust tarbida ilma kasutajakontota. Lisas 2 on kujutatud plokkskeemina eesrakenduse üldine kasutusvoog. Lahendus on loodud veebipõhise eesrakendusena, mille välja töötamiseks on kasutatud C# programmeerimiskeelel põhinevat Blazor veebiraamistikku. Blazor on osa ASP.NET Core veebiraamistikust kasutajaliideste loomiseks, seega saavad taga- ja eesrakendus kasutada sama koodibaasi ning rakendada ühiseid tehnoloogiaid ja andmemudeli klasse.

5.3.1 Struktuur

Eesrakenduse loomiseks on kasutatud kahte projekti. Projektis *CPM.UI* hoiustatakse komponente, mida võivad kasutada ka teised veebipõhised eesrakendused. Konkreetselt käesolevaks kasutajapoolseks lahenduseks on *CPM.Checkout* projekt, mis sisaldab faile nagu tõlked (ingl. k. *locales*) ja erinevaid projektipõhiseid komponente (vt Joonis 13): paigutused (ingl. k. *layouts*), leheküljed (ingl. k. *pages*) ja vidinad (ingl. k. *widgets*).

Kasutajaliidese jagamine erinevateks osadeks võimaldab neid taaskasutada ja määrata nende välimus ühes kohas (ingl. k. *single source of truth*). Sel viisil tagatakse kasutusel olevatel komponentidel ühesugune väljanägemine, kuna välimuse korrigeerimiseks pole vaja muuta kõiki lehekülgi.



Joonis 13. Kasutajapoolse lahenduse projekti struktuur

Komponendid on loodud Razor süntaksiga (vt Joonis 14), mis kasutab Razor märgistuskeelt ning C#, CSS ja HTML tehnoloogiaid. HTML elementidega on kirjeldatud kasutajaliidese ülesehitus. Samuti on võimalik elementide vahele lisada näiteks tingivaid lauseid (ingl. k. *if sentences*), mis muudavad ülesehituse loomise paindlikumaks. CSS faile on kasutatud elementide välimuse määramiseks.

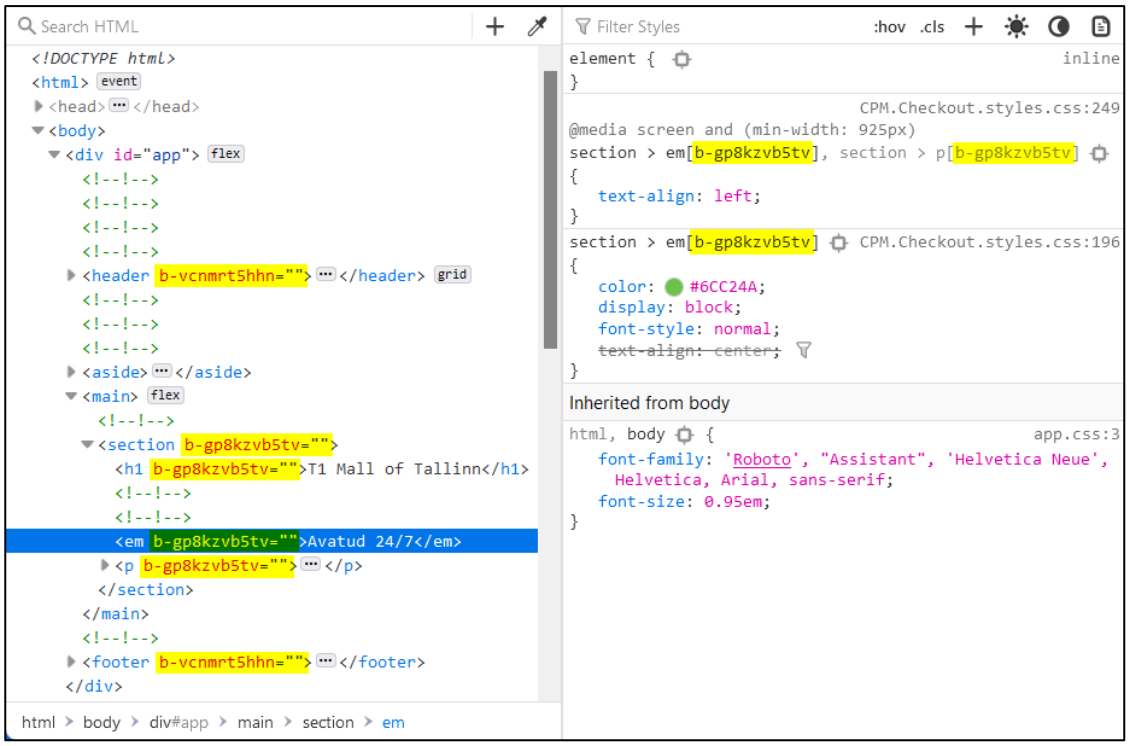
LanguageSelector.razor

```
@inject II18N L
@inject NavigationManager _navigationManager

<section>
    <button><Icon Name=@($"flags/{Language.Locale}.svg") /></button>
    <div>
        @foreach (var language in L.Languages) {
            <NavLink href="@GetLink(language)"><Icon Name=@($"flags/{language.Locale}.svg") /></NavLink>
        }
    </div>
</section>

@code {

    [Parameter]
    public PortableLanguage Language { get; set; }

    private string GetLink(PortableLanguage language) {
        var path = _navigationManager.ToBaseRelativePath(_navigationManager.Uri);
        var position = path.IndexOf('/');
        return $"/{language.Locale}{(position > 0 ? path.Substring(position) : null)}";
    }

}
```

Joonis 14. Kasutajaliidese komponentide loomiseks kasutatav Razor süntaks

Kasutajaliidesele talitluse lisamiseks on kirjutatud C# programmeerimiskeeles Razor komponendi lõppu kood. HTML elementide ja C# koodi ühendamiseks on kasutatud Razor märgistust, mis on mallide (ingl. k. *templates*) loomiseks mõeldud märgistuskeel. Selle abil on võimalik käivitada C# programmeerimiskeeles kirjutatud koodi vastavalt kasutaja toimingutele, näiteks nupu vajutamine või vormi sisendvälja muutmine. Razor märgistuskeel võimaldab ka komponentidele parameetreid kaasa anda, mis muudab kasutajaliidese loomise väga tõhusaks.

Eesrakenduse kujundamiseks on kasutatud CSS eraldatust (ingl. k. *CSS isolation*) ehk välimus lisatakse ainult konkreetses käsitlusalas (ingl. k. *scope*) olevatele elementidele. Selline lähenemine võimaldab hoida eesrakenduse lähtekoodi puhtana, kuna HTML elementide tähistamiseks pole vaja neile lisada näiteks rakendusüleseid *class* atribuute. CSS eraldatuseks on kasutatud rakenduse, paigutuste, lehekülgede ja vidinate käsitlusalasid, mis luuakse Blazor veebiraamistiku poolt HTML elementidele ainulaadsete märgete lisamisega (vt Joonis 15).



Joonis 15. CSS eraldatuseks loodud suvalised märgised

Staatiliste andmetega eesrakendus oleks küllaltki kasutu, mistõttu on tagarakendusega suhtlus vajalik. Serveripoolse lahendusega suhtlemiseks on kasutatud Blazori veebi-raamistikku sisseehitatud HttpClient klassi, mis võimaldab teha REST API päringuid. HttpClient objekt luuakse rakenduse käivitamisel *Program* klassis, kus lisatakse viide tagarakenduse veebiaadressile.

6 Veebirakenduse testimine

Veebirakenduse toimimises veendumiseks oli vaja ka lahendust testida. Testimine viidi läbi käsitsi erinevaid tööriistu kasutades. Esmalt kontrolliti tagarakendust programmiga Insomnia, mis võimaldab koostada REST API päringuid. Testimise eesmärgiks oli veenduda, et REST API lõpp-punktid tagastavad õigeid HTTP olekukoode ja vastuseid.

Järgnevalt testiti eesrakenduse väljanägemist Google Chrome, Mozilla Firefox ja Safari veebilehitsejates, kuna veebilehitsejate esitlusmootorid võivad HTML elemente kuvada erinevalt [46]. Rakendust ei ole põhjalikult testitud teistes Chromium-põhistes veebilehitsejates nagu Microsoft Edge ja Opera, kuna need kasutavad Google Chrome veebilehitsejaga sama esitlusmootorit.

Veebirakenduse välimust võib mõjutada ka mobiilseadmete erinevad ekraani eraldusvõimed. Erinevate ekraani eraldusvõimete toetamiseks kasutatakse kohanduvat disaini ja selle toimimist kontrolliti veebilehitsejatesse sisseehitatud tööriistadega, mis võimaldasid rakendust proovida erinevate ekraanisuuruste ja jõudlustega. Viimane oli eriti kasulik, kuna võimaldas rakendust proovida aeglast internetiühendust jäljendades.

Kaardimaksete sooritamiseks kasutatakse Stripe maksete töötlemise platvormi, mis pakkus testkeskkonnas makselahenduse katsetamiseks kindlate tulemustega maksekaarte. Eelnevalt määratletud tulemus nagu ebaõnnestunud makse või kaardiomaniku täiendava nõusoleku vajamine võimaldas katsetada erinevaid stsenaariume nii enne laadimisseansi alustamist kui ka laadimisseansi ajal.

Laadimisseansi alustamise ja lõpetamise testimiseks oli kontoris testlaadija ja autot jäljendav seade (vt Joonis 16). Laetud kilovatt-tundide testimiseks oli autot jäljendava seadme külge võimalik ühendada näiteks veekeedukann voolutarbimise tekitamiseks.

Joonis 16. Testlaadija (vasakul) ja autot jäljendav seade koos veekeedukannuga (paremal)

6.1 Testimise tulemused

Analüüsi käigus püstitatud funktsionaalsed nõuded said täidetud. Olemasoleva lahendusega on loodud samaväärne võimekus, millele lisaks on loodud ka keelevalik. Enne olemasoleva lahenduse asendamist tuleks veebirakendust veelgi ulatuslikumalt testida, et veenduda makselahenduse turvalisuses ja järjekindluses nii enne laadimisseansi alustamist kui ka selle ajal.

Rahuldatud said ka mittefunktsionaalsed nõuded. Veebirakendus võtab tasu ettemaksuna ja kasutajakogemust ohverdamata on võimalik veebirakendust kasutada nii nutitelefonis kui ka arvutis.

6.2 Edasiarenduse võimalused

Mõistliku lõputöö skoobi töömahu säilitamiseks loodi vähim elujõuline toode. Vaatamata sellele, et loodud veebirakendusel on küllaltki kitsas kasutusala, on selle täiustamiseks sellegipoolest olemas mõned edasiarenduse võimalused.

Esimese täiustusena saaks lisada nii laadija kui laadimispistiku valimise ühte QR-koodi skaneerides. Kasutajakogemust aitaks oluliselt parandada ka märguanded laadimisseansi kohta, näiteks kui laadimine on ootamatult katkenud või auto aku on saanud täis. Mõlema funktsiooni lisamiseks tuleks täiendavalt uurida, kas ja millised võimalused on Blazor veebiraamistikul kasutada seadmepõhiseid toiminguid nagu kaamera kasutamine ja tõuketeavituste saamine.

7 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua veebirakendus, mis võimaldaks laadida elektriautot Eleporti laadimispunktides ilma kasutajakontota, asendamaks Virta laadimisplatvormi. Virta puudusteks on vahendustasu hinnakasv ja vähene arendustöö. Viimane mõjutab otseselt Eleporti, kuna kasutajate tagasiside põhjal ei ole võimalik lahendust täiustada, mis ohustab konkurentsivõimet.

Analüüsi peatükkides kirjeldati probleemi seoses entusiastide osakaalu vähenemisega elektriautode kasutajate seas, toodi välja levinumad laadimisplatvormid koos eeliste ja puudustega, püstitati lõputöö eesmärk ning ettevõtte tehnoloogiajuhi ja ärianalüütikuga koostöös selgitati välja funktsionaalsed ja mittefunktsionaalsed nõuded. Samuti analüüsiti ettevõttes kasutusel olevate tehnoloogiate jätkusuutlikkust. Analüüsi põhjal soovitab töö autor mõelda MongoDB andmebaasi väljavahetamisele Microsoft Azure Cosmos DB vastu, kuid taga- ja eesrakenduste loomiseks kasutatavad raamistikud on autori hinnangul õigustatud.

Praktilises osas võrreldi vana ja uue lahenduse väljanägemist, tõstes esile nii olemasoleva lahenduse kitsaskohad kui ka uue lahenduse võimalikud täiustused kasutajakogemuses. Järgnevalt kirjeldati välja töötatud andmemudelit ning taga- ja eesrakenduse ülesehitust. Praktilises osas teostati ka valminud lahenduse testimine, misjärel anti ülevaade testimise tulemustest. Praktilise osa lõpetuseks pakkus autor välja edasiarenduse võimalused.

Töö käigus valmis nõuetele vastav lahendus ja eesmärk sai täidetud. Rakendusel on eeldused kasutusel oleva lahenduse asendamiseks, kuid põhjalikuma testimise puudumise tõttu ei ole veel avalikus kasutuses. Ettevõttel on soov juurutada ka lõputöö skoobist välja jäänud edasiarenduse võimalused, mistõttu jätkab töö autor koostööd ettevõttega projekti edasiarendamiseks.

Kasutatud kirjandus

[1] *„e-Teatmik: IT ja sidetehnika seletav sõnaraamat“*. [Võrgumaterjal]. Loetud aadressil: http://vallaste.ee. Kasutatud: 23.04.2022.

[2] *„JSON and BSON“*. [Võrgumaterjal]. Loetud aadressil: https://www.mongodb.com/json-and-bson. Kasutatud: 23.04.2022.

[3] *„AKIT - Andmekaitse ja infoturbe leksikon“*. [Võrgumaterjal]. Loetud aadressil: https://akit.cyber.ee. Kasutatud: 23.04.2022.

[4] EUR-Lex, *„Euroopa Parlamendi ja nõukogu direktiiv 2014/94/EL“*, 2014. [Võrgumaterjal]. Loetud aadressil: https://eur-lex.europa.eu/legal-content/ET/TXT/PDF/?uri=CELEX:32014L0094. Kasutatud: 05.03.2022.

[5] Riigi Teataja, *„Alternatiivkütuste taristu kasutuselevõtule esitatavad ohutusnõuded“*, 2017. [Võrgumaterjal]. Loetud aadressil: https://www.riigiteataja.ee/akt/101122017014. Kasutatud: 05.03.2022.

[6] *„White Label Product“*. [Võrgumaterjal]. Loetud aadressil: https://www.investopedia.com/terms/w/white-label-product.asp. Kasutatud: 12. 03.2022.

[7] has·to·be, *„has·to·be software for fast charging network“*, 2019. [Võrgumaterjal]. Loetud aadressil: https://has-to-be.com/en/press/has-to-be-provides-emobility-software-for-european-fast-charging-network. Kasutatud: 12.03.2022.

[8] Alexela, *„Today, Alexela opened its charging stations for electric vehicles. To celebrate this, charging electric cars is free of charge until the end of the month“*, 2021. [Võrgumaterjal]. Loetud aadressil: https://www.alexela.ee/en/about-alexela/today-alexela-opened-its-charging-stations-electric-vehicles-celebrate-charging. Kasutatud: 12.03.2022.

[9] has·to·be, *„has·to·be gmbh's European Roaming Network Reaches 200,000 Charge Points“*, 2021. [Võrgumaterjal]. Loetud aadressil: https://has-to-be.com/en/general/european-roaming-network-reaches-200000-charge-points. Kasutatud: 12.03.2022

[10] *„Taking charge of tomorrow – Virta“*. [Võrgumaterjal]. Loetud aadressil: https://www.virta.global/company. Kasutatud: 13.03.2022.

[11] Virta, *„Virta leads the EV charging industry for the third year in a row on the Financial Times FT 1000 list“*, 2022. [Võrgumaterjal]. Loetud aadressil: https://www.virta.global/news/virta-leads-the-ev-charging-industry-for-the-third-year-in-a-row-on-the-financial-times-ft-1000-list. Kasutatud: 13.03.2022.

[12] Driivz, *„Enefit VOLT Transforms into a New Charging Network Using Driivz Technology“*, 2021. [Võrgumaterjal]. Loetud aadressil: https://driivz.com/news/enefit-volt-chooses-driivz. Kasutatud: 13.03.2022.

[13] *„MoSCoW or Kano Models - how do you prioritize?“*. [Võrgumaterjal]. Loetud aadressil: https://www.hotpmo.com/management-models/moscow-kano-prioritize. Kasutatud: 13.03.2022.

[14] *„NoSQL vs SQL Databases“*. [Võrgumaterjal]. Loetud aadressil: https://www.mongodb.com/nosql-explained/nosql-vs-sql. Kasutatud: 18.03.2022.

[15] J. Pokorny, *„NoSQL databases: a step to database scalability in web environment“*, 2013. [Võrgumaterjal]. Loetud aadressil: http://faculty.washington.edu/wlloyd/courses/tcss562/papers/Spring2017/team7_NOSQL_DB/NoSQL%20databases-%20a%20step%20to%20database%20scalability%20in%20web%20environment.pdf. Kasutatud: 18.03.2022.

[16] *„SQL vs. NoSQL Databases: What's the Difference?“*. [Võrgumaterjal]. Loetud aadressil: https://www.ibm.com/cloud/blog/sql-vs-nosql. Kasutatud: 19.03.2022.

[17] *„Stack Overflow Developer Survey 2017“*. [Võrgumaterjal]. Loetud aadressil: https://insights.stackoverflow.com/survey/2017#technology-_-databases. Kasutatud: 19.03.2022.

[18] *„Stack Overflow Developer Survey 2021“*. [Võrgumaterjal]. Loetud aadressil: https://insights.stackoverflow.com/survey/2021#most-popular-technologies-database-prof. Kasutatud: 19.03.2022.

[19] *„Understanding the Different Types of NoSQL Databases“*. [Võrgumaterjal]. Loetud aadressil: https://www.mongodb.com/scale/types-of-nosql-databases. Kasutatud: 19.03.2022.

[20] *„What is a Key-Value Database?“*. [Võrgumaterjal]. Loetud aadressil: https://redis.com/nosql/key-value-databases. Kasutatud: 19.03.2022.

[21] C. O. Truica, F. Radulescu, A. Boicea ja I. Bucur, *„Performance evaluation for CRUD operations in asynchronously replicated document oriented database“*, 2015. [Võrgumaterjal]. Loetud aadressil: https://arxiv.org/pdf/1806.04761.pdf. Kasutatud: 19.03.2022.

[22] I. Mapanga ja P. Kadebu, *„Database Management Systems: A NoSQL Analysis“*, 2013. [Võrgumaterjal]. Loetud aadressil: https://www.researchgate.net/profile/Innocent-Mapanga/publication/258328266_Database_Management_Systems_A_NoSQL_Analysis/links/00b49527d04f2f19ed000000/Database-Management-Systems-A-NoSQL-Analysis.pdf. Kasutatud: 20.03.2022.

[23] *„Wide Column Stores“*. [Võrgumaterjal]. Loetud aadressil: https://db-engines.com/en/article/Wide+Column+Stores. Kasutatud: 20.03.2022.

[24] *„Wide-column Database“*. [Võrgumaterjal]. Loetud aadressil: https://www.scylladb.com/glossary/wide-column-database. Kasutatud: 20.03.2022.

[25] *„Azure Cosmos DB pricing“*. [Võrgumaterjal]. Loetud aadressil: https://azure.microsoft.com/en-us/pricing/details/cosmos-db. Kasutatud: 26.03.2022.

[26] *„Microsoft Azure Cosmos DB System Properties“*. [Võrgumaterjal]. Loetud aadressil: https://db-engines.com/en/system/Microsoft+Azure+Cosmos+DB. Kasutatud: 26.03.2022.

[27] *„SLA for Azure Cosmos DB“*. [Võrgumaterjal]. Loetud aadressil: https://azure.microsoft.com/en-gb/support/legal/sla/cosmos-db/v1_4. Kasutatud: 26.03.2022.

[28] *„1.6.3.1 Mitmekihiline arhitektuur“*. [Võrgumaterjal]. Loetud aadressil: https://eopearhiiv.edu.ee/e-kursused/eucip/arendus/1631_mitmekihiline_arhitektuur.html. Kasutatud: 27.03.2022.

[29] O. Dospinescu ja M. Perca, *„Web Services in Mobile Applications“*, 2013. [Võrgumaterjal]. Loetud aadressil: https://revistaie.ase.ro/content/66/02%20-%20Dospinescu,%20Perca.pdf. Kasutatud: 27.03.2022.

[30] *„REST vs. SOAP“*. [Võrgumaterjal]. Loetud aadressil: https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest.
Kasutatud: 27.03.2022.

[31] B. Moschetti, *„Why XML-in-RDBMS != MongoDB“*, 2018. [Võrgumaterjal]. Loetud aadressil: https://www.mongodb.com/blog/post/why-xml-in-rdbms-is-not-mongodb. Kasutatud: 27.03.2022.

[32] *„Working with JSON in Azure Cosmos DB“*. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/azure/cosmos-db/sql/sql-query-working-with-json. Kasutatud: 27.03.2022.

[33] L. Tratt, *„Dynamically typed languages“*, 2009. [Võrgumaterjal]. Loetud aadressil: https://eprints.mdx.ac.uk/5921/1/Trapp_-_dynamically_typed_languages.pdf. Kasutatud: 02.04.2022.

[34] *„Choose an ASP.NET Core web UI“*. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/aspnet/core/tutorials/choose-web-ui. Kasutatud: 02.04.2022.

[35] H. K. Dhalla, *„A Performance Comparison of RESTful Applications Implemented in Spring Boot Java and MS. NET Core“*, 2021. [Võrgumaterjal]. Loetud aadressil: https://iopscience.iop.org/article/10.1088/1742-6596/1933/1/012041/pdf. Kasutatud: 02.04.2022.

[36] *„License Terms | Microsoft Visual Studio Community 2022“*. [Võrgumaterjal]. Loetud aadressil: https://visualstudio.microsoft.com/license-terms/vs2022-ga-community. Kasutatud: 02.04.2022.

[37] *„Should I Build a Mobile Website or a Mobile App?“*. [Võrgumaterjal]. Loetud aadressil: https://www.cleart.com/build-mobile-website-mobile-app.html. Kasutatud: 03.04.2022.

[38] S. Bose, *„How to resolve JavaScript Cross Browser Compatibility Issues“*, 2021. [Võrgumaterjal]. Loetud aadressil: https://www.browserstack.com/guide/resolve-javascript-cross-browser-compatibility-issues. Kasutatud: 03.04.2022.

[39] *„About diagrams.net“*. [Võrgumaterjal]. Loetud aadressil: https://www.diagrams.net/about. Kasutatud: 09.04.2022.

[40] *„Visual Studio 2022 Community Edition“*. [Võrgumaterjal]. Loetud aadressil: https://visualstudio.microsoft.com/vs/community. Kasutatud: 09.04.2022.

[41] D. Taylor, *„NoSQL Tutorial: What is, Types of NoSQL Databases & Example“*, 2022. [Võrgumaterjal]. Loetud aadressil: https://www.guru99.com/nosql-tutorial.html. Kasutatud: 10.04.2022.

[42] *„Three Tier System Architecture For Business Applications“*. [Võrgumaterjal]. Loetud aadressil: https://www.codeauthority.com/Blog/Entry/three-tier-architecture. Kasutatud: 16.04.2022.

[43] J. P. S. Boodhoo, *„Layered Architecture, Dependency Injection, and Dependency Inversion“*, 2020. [Võrgumaterjal]. Loetud aadressil: https://www.codemag.com/article/0705071/Layered-Architecture-Dependency-Injection-and-Dependency-Inversion. Kasutatud: 16.04.2022.

[44] *„What is HTTPS?“*. [Võrgumaterjal]. Loetud aadressil: https://www.cloudflare.com/en-gb/learning/ssl/what-is-https. Kasutatud: 17.04.2022.

[45] E. Schechter, *„A milestone for Chrome security: marking HTTP as 'not secure'“*, 2018. [Võrgumaterjal]. Loetud aadressil: https://blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure. Kasutatud: 17.04.2022.

[46] A. Betts, *„How Do Browsers Display Web Pages, and Why Don't They Ever Look the Same?“*, 2015. [Võrgumaterjal]. Loetud aadressil: https://www.makeuseof.com/tag/how-do-browsers-display-web-pages-and-why-dont-they-ever-look-the-same. Kasutatud: 23.04.2022.

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

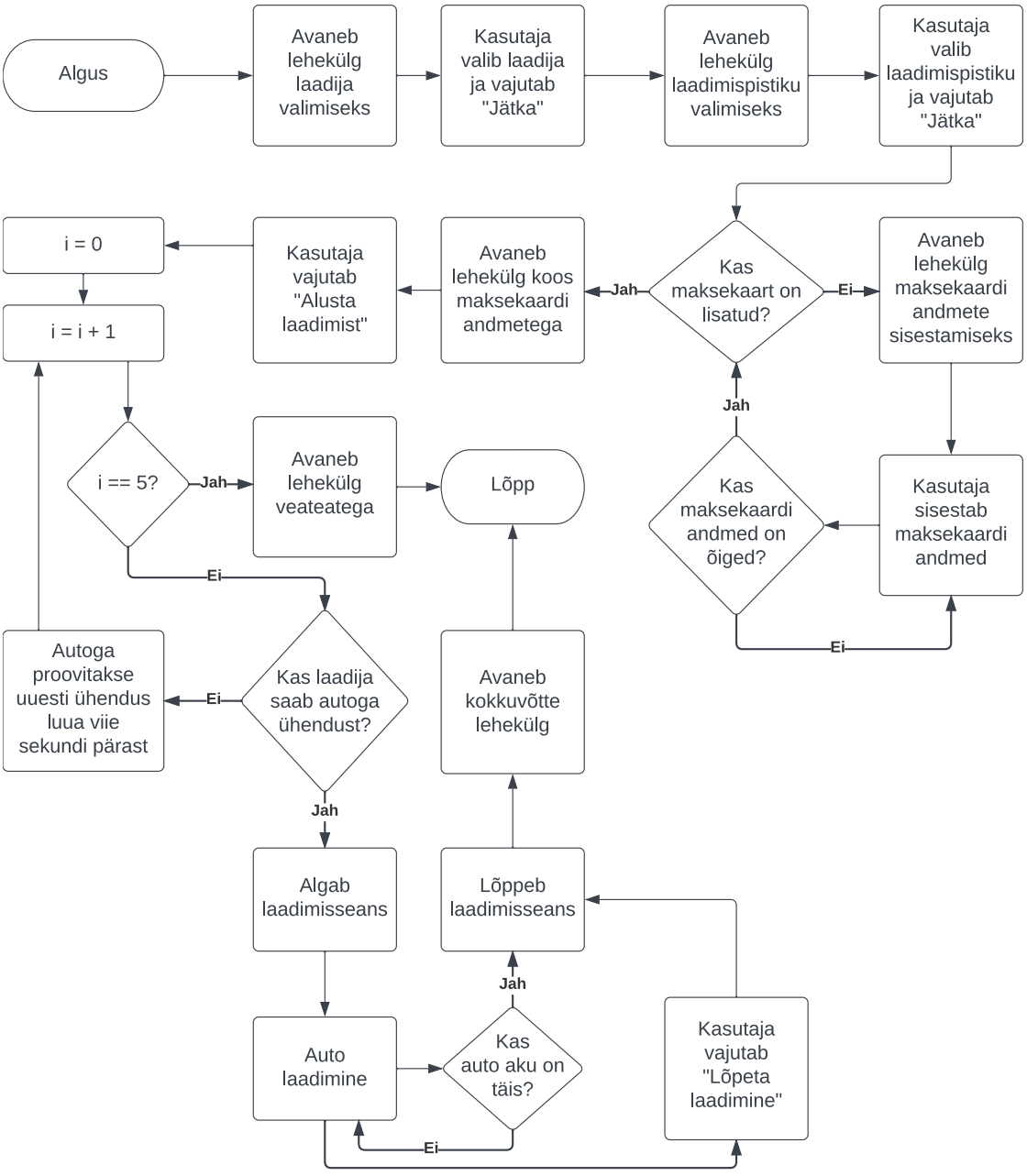
Mina, Mathias Kivi

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Veebirakenduse loomine elektriautode laadimiseks külalisena Eleport laadimis-punktides“, mille juhendajad on Meelis Antoi ja Ivari Horm
   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.

16.05.2022

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 – Plokkseem kasutajapoolse lahenduse üldise kasutusvoo kujutamiseks



Joonis 17. Plokkskeem eesrakenduse kasutusvoo kujutamiseks