TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Andrei Tomba 176287IDDR

Autokulude märkmiku rakenduse arendamine

Diplomitöö

Juhendaja: Nadežda Furs
MBA

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: Andrei Tomba

17.05.2021

Annotatsioon

Antud diplomitöö eesmärgiks on luua auto omamise kulu märkmik lahendust, mis oleks abiks enne auto soetamist, ning reaalsete kulude jälgimiseks. Antud rakendus saab olema märkmiku tüüpi, kuhu on võimalik sisestada mitu autot, ning võrrelda nende kogu kulusid ajavahemiku ja distantsi põhiselt.

Töös lahendatavaks peamiseks probleemiks on luua mugava, mitme parameetreid kaasava ja mitme auto võrdlusvõimalusega autokulude märkmiklahenduse minimaalse töötava lahenduse (MVP), millel oleks arendusvõimaluse potentsiaal.

Analüüsi käigus selgitatakse välja nõutud parameetrite valikut, kaetakse rakenduse funktsionaalseid ja mittefunktsionaalseid nõudeid, selgitatakse välja kasutusjuhud, toetades kasutusmallidele, valitakse sobilike tehnoloogiat, rakenduse arhitektuuri, ning arendusprotsessi tulemuseks valmib veebiteenus ja kliendiliidese rakendus.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 57 leheküljel, 7 peatükki, 11 joonist, 10 tabelit.

Abstract

Development of Vehicle Expenses' Notebook Application

The objective of this diploma thesis is to create a car ownership cost notebook solution that would be helpful before purchasing a car, and to track real costs. This will be a notebook type application, where can be entered several cars and compared their total costs based on time period and distance.

The main problem to be solved in the thesis is to create a convenient minimum viable product (MVP) of a car expenses notebook solution with multiple parameters and an option to compare multiple vehicles, which would also have potential for further development.

The analysis identifies the selection of required parameters, fulfills the functional and non-functional requirements of the application, determines use cases, selects appropriate technologies and application architecture. As a result, a web service and a customer interface application are created.

The thesis is in Estonian language and contains 57 pages of text, 7 chapters, 11 figures, 10 tables.

Lühendite ja mõistete sõnastik

ACID	*Atomicity Consistency Isolation Durability* (andmebaasile seatud piirangute komplekt)
API	*Application Programming Interface* (rakendusliides)
ARM	*Advanced RISC Machines* (kärbitud käsustikuga arvutiarhitektuur)
ASP.NET	*Active Server Pages .NET* (Microsofti raamistik dünaamiliste veebirakenduste ja veebiteenuste loomiseks)
AWS	*Amazon Web Services* (teenus mis pakub tellitavat pilveandmetöötluse platvormi)
BLL	*Business Logic Layer* (äriloogika kiht)
CI/CD	*Continuous Integration / Continuous Delivery* (pidev integratsioon / pidev tarne)
CORS	*Cross-Origin Resource Sharing* (mehhanism, mis võimaldab veebisaidi piiratud ressursse taotleda teiselt domeenilt)
CSRF	*Cross-Site Request Forgery* (saidiülene taotluste võltsimine)
CRUD	*Create Read Update Delete* (loo loe muuda kustuta)
DAL	*Data Access Layer* (andmete ligipääsu kiht)
DBMS	*Database Management System* (andmebaasijuhtimisüsteem)
DevOps	*Development and IT Operations* (arendus ja käitus)
DSL	*Domain Specific Language* (domeenipõhine keel)
DOM	*Document Object Model* (dokumendi objektimudel)
DTO	*Data Transfer Object* (andmeedastusobjekt)
ERD	*Entity Relationship Diagram* (olemi-suhte graaf)
EF	*Entity Framework* (avatud lähtekoodiga ORM raamistik)
ETL	*Extract Transform Load* (protseduur andmete kopeerimiseks)
EURIBOR	*Euro Interbank Offer Rate* (euro üleeuroopaline pankadevaheline intressimäär)
E2E	*End to End* (otsast lõpuni)
GUID	*Globally Unique Identifier* (universaalne unikaalne identifikaator)
JS	*Java Script* (objektorienteeritud programmeerimiskeel)

JSON	*JavaScript Object Notation* (JavaScript andmevahetusvorming)
JWT	*JSON Web Token* (veebitõend)
LAMP	*Linux, Apache, MySQL, PHP/Perl/Python* (komplekt operatsioonisüsteemist ja kolmest vabavaralisest komponendist)
LAN	*Local Area Network* (lokaalvõrk)
MDM	*Master Data Management* (põhiandmete haldamine)
Metadata	*Data that provides information about other data* (Metaandmed)
MSSQL	*Microsoft SQL* (relatsiooniline andmebaasijuhtimisüsteem loodud Microsoft-i poolt)
MVC	*Model View Controller* (mudel vaade kontroller)
MVP	*Minimum Viable Product* (minimaalne töötav toode)
ORM	*Object-Relational Mapper* (objektide relatsiooniline vastendamine)
OS	*Operating System* (Operatsioonisüsteem)
PDF	*Portable Document Format* (Porditav dokumendivorming)
PHP	*Hypertext Preprocessor* (skriptimiskeel)
PK	*Primary Key* (primaarvõti)
QA	*Quality Assurance* (kvaliteedi tagamine)
REST	*Representational state transfer* (tarkvaraarhitektuuri laad)
RESTful	*Service, that implements REST architectural pattern* (API servis)
RoR	*Ruby on Rails* (veebiraamistik, mis on kirjutatud Ruby programmeerimiskeeles)
SPA	*Single Page Application* (üheleherakendus)
SQL	*Structured Query Language* (struktureeritud andmepäringukeel)
IIS	*Internet Information Services* (Microsofti veebiserver)
IP	*Internet Protocol* (Internetiprotokoll)
UC	*Use Case* (kasutusjuhus)
UOW	*Unit Of Work* (tööühik)
UML	*Unified Modeling Language* (ühtne modelleerimiskeel)
UUID	*Universally Unique Identifier* (universaalne unikaalne identifikaator)
XSS	*Cross-Site Scripting* (saidiülene skriptimine)
WEB API	*API for a Web server or a Web browser* (veebiserveri või veebi brauseri rakendusliides)

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeva maailmas inimesed päris tihti vahetavad enda liiklusvahendeid tänu kas sissetuleku suurenemisele, või vähenemisele, rangematele väljaheitegaasi nõuetele, kaasaaegsete autode ressursi piirangule, pereliikmete arvu suurenemisele, seadusandlusele, või mis iganes muude põhjuse pärast.

IHS Markit 2015 aasta uuringu andmetel ameeriklased vahetavad autosid iga 6,5 aasta tagant [1]. Tuginedes Eesti Transpodiameti 2019 aasta esimeste 9 kuu (jaanuar-oktoober) andmetele, sõiduauto (M1) kategoorias on toimunud 107284 omanikuvahetust järelturul, ning sai esmaselt registreeritud 23614 uusi sõidukeid [2], ehk ligi 10% Eesti elanikkonnast said 9 kuuga vahetatud enda sõidukit.

Iga auto omamisega kaasnevad lisakulud, mida oleks vähemalt ligikaudselt selgeks teha enne auto soetamist, kuid mida enamasti ei tehta, või siis tehakse, aga osaliselt. Esinduste- või pangapoolsed arvutuste lahendused näitavad vaid osa kogu kulust, näiteks ainult liisingu kuumakset.

Peale auto liisingu kuumakset on olemas ka muud kulud (mis võivad ületada liisingu makset), nagu näiteks auto odavnevus, kütuse, kasko ja liikluskindlustuse, hoolduste ja remondi kulud. Neid oleks hea enne auto soetamist arvesse võtta, arvutades ostetava auto ligikaudse omamise kogukulu.

Käesoleva töö eesmärgiks on luua märkmik tüüpi autokulude arvutamise lahendust, kus saaks sisestada mitu autod, koos nende prognoositavate kuludega, ning pääseda salvestatule hiljem ligi. Selleks tuleb teha olemasolevate autokulude rakenduste turuanalüüs, parimate võtete väljaselgitamist, autokulude märkmik lahenduse loomist, ning selle majutamist, edaspidise arendamise võimalusega.

Arendatav lahendus saab olema abiks inimestele, kes on huvitatud kulude arvestuses, ning kes soovivad märkmiku kujul lihtsasti võrrelda prognoositava auto omamise kogukulu, ehk peaks aitama leida parima laheduse (kulude vaatest) auto ostu otsuse tegemisel.

Lõputöö jaotub kuue põhilisse peatükki. Esimene peatükk on sissejuhatav. Teises räägitakse probleemist, diplomitöö eesmärgist, skoobist ning tehakse turul olevate lahenduste ülevaadet. Kolmas peatükk lahendab tehnoloogia valiku probleemi. Siin tuuakse välja võimalike tehnoloogiate ülevaadet, kaasates võrdleva analüüsi, ning valiku põhjendust. Neljas peatükk keskendub rakenduse analüüsile. Käsitleb rakenduse nõudeid, kasutajaliidese osa, lahenduse arhitektuurilist struktuuri, räägib turvalisusest, ning valmislahenduse majutamisest. Viiendas peatükis on toodud välja tulemused, ning räägitud ka edasiarenduse võimalustest. Viimane peatükk on kokkuvõttev.

Allikatena on kasutusel enamasti võrgumaterjal, kuid püütud on võimalikult palju tugineda ka teemapõhistele raamatutele, teadusartiklitele ning õpingute käigus läbitud ainetes kasutatud materjalidele.

1.1 Metoodika

Antud lõputöö raames esmalt selgeks tehakse uuritava probleemi, võrreldakse antud valdkonna olemasolevaid lahendusi, selgitatakse välja nende eeliseid, kitsaskohti ja puudujääke. Seejärel leitakse optimaalse tehnoloogilise lahenduse loodava rakenduse funktsionaalsete ning mittefunktsionaalsete nõuete täitmiseks. Peale selle kirjeldatakse kavandatava rakenduse IT lahendust, mis katab olemasolevate lahenduste puudujääke, vastab püstitatud nõetele ning on ehitatud jätkusuutliku arhitektuuriga. Lõpuks antakse hinnang loodud rakendusele ja räägitakse edasiarenduse võimalustest.

Käesolev lõputöö on suunatud: sarnaste lahenduste võrdlevale analüüsile; infosüsteemi skoobi kindlaksmääramisele, sobilike tehnoloogiate leidmisele, kooskõlas vastava arhitektuuriga; tehnoloogilise lahenduse realiseerimisele, rakenduse esmase minimaalse väärtust pakkuva lahenduse (MVP) loomisele, rakenduse turvalisuse tagamisele.

Nõuete kaardistamisel, küsimustikuga lähenemist polnud otsustatud kasutusele võtta, kuna lähtuti olemasolevate sarnaste lahenduste võrdleva analüüsist, ning fookusrühmaks MVP kasutamiseks on lõputöö kirjutaja, ning tema tutvuskond.

2 Probleemi püstitus

Auto ostmine on tõenäoliselt üks suurimatest ostudest, mida inimesed tavaliselt oma elus teevad - see jääb alla ainult oma kodu ostmisele. Enamik inimestest alahindab suuresti autoga seotud kulutusi, ning peab autokuluks ainult igakuist makset. Kuid, kui kõike kulusid arvesse võtta, võib tegelik kulu olla üllatavalt kõrge.

Auto omamise reaalse hinna teadmine on oluline eelkõige seetõttu, et see mõjutab võimalusi teha oma rahaga muid asju - näiteks minna puhkusele, investeerida pensionisse või maksta tagasi vanu laene ja võlgasid.

Autokulud võivad vastavalt autole palju erineda, sõltuvalt sellest kas tegu on kasutatud või uue autoga, eeldatavast hooldusest ja paljust muust. Esimesena mis tuleb meelde on kütusekulu, aga tegelikud kulud hõlmavad amortisatsiooni, kindlustust, hooldust, parkimist, pesemist, rehvivahetuse tasusid, ülevaatust, mõnes riigis ka automaksu, teemaksu ja palju muud. Seega lisaks kuumaksule tuleb arvestada paljude muude kulutustega. Vähe sellest, erinevatel sõidukitel võivad olla väga erinevad kulud - mõnel autol oletub kilomeetri läbimine, teisega võrreldes, kaks või kolm korda kallim.

Sõidukite uurimisel ja võrdlemisel võib kulude arvutamine ja võrdlemine olla hirmutav kogemus. Seepärast kõige tegurite mõistmiseks on oluline autokulude märkmiku rakendust kasutada. See näitaks sõidukiga seotud kulude tegelikku mõju inimese eelarvele ja aitaks leida parima valiku, sellise mida saab endale tõesti lubada. Hetkel turul leidub erinevaid lahendusi, mis ei rahulda antud töö autori nõudmisi, kuid mille võimalusi võiks uurida, ning leida sobiliku lahenduse.

2.1 Diplomitöö eesmärk

Antud diplomitöö eesmärgiks on autokulude märkmiku rakenduse tehniline analüüs ja loomine, ning majutamine edaspidise arendamise võimalusega. Valmis lahendus peab olema lihtsasti ja intuitiivselt kasutatav, ning kättesaadav võimalikult suurele kasutajaskonnale.

Töös peamiseks lahendatavaks probleemiks on luua mugava, mitu parameetre kaasava (turul olevate parimate praktikate baasil), mitme auto võrdlusvõimalusega auto kogukulu

kalkuleeriva märkmiklahenduse minimaalse töötava lahenduse (MVP), millel oleks ka arendusvõimaluse potentsiaal.

2.1.1 Diplomitöö skoop ja autori roll

Antud töö käigus sooritavad tegevused hõlmavad sarnaste lahenduste võrdlevat analüüsi, et leida olemasolevate lahenduste kitsaskohti ja arendusvõimalusi. Lisaks sobilikke tehnoloogiate leidmist, ning rakenduse arendust: andmebaasi disaini, serveripoolse lahenduse loomist, kliendipoolseliidese arendust, kogu lahenduse funktsioneerimist (manuaalne QA testimine) ja majutamist serverile. Loodud MVP lahendus on plaanitud kasutusele võtta peale töö sooritust.

Käesolevas töös on autori roll teostada kogu analüüsi ja luua veebirakendus, mis annab võimaluse sisestada mitme auto prognoositavad kulud, mis arvutab nende põhjal auto omamise kogukulu ajavahemiku ja kilomeetri kohta ning väljundina tekib sisestatud autode arvutatud tulemuste võrdluse võimaluse.

Diplomitöö skoopi ei kuulu automaattestide kirjutamine, pidev integratsioon / pidev tarne (CI/CD) ning serveri turvaprotokollide seadistust.

2.2 Valdkonna ülevaade

Selleks et koguda vajalike sisendeid rakenduse loomiseks esimese asjana on vaja vaadata turul olevate lahenduste võimalusi. Selleks on teostatud valdkonna ülevaade, mis jaotub kahte ossa. Esimene on suunatud pangapoolsete lahenduste võrdlusele, mis enamasti keskenduvad auto liisingumakse arvutusele. Teine osa on suunatud muude lahenduste uurimisele, mis peale liisingumakse arvutamist võtavad arvesse ka muid parameetre. Täpsemalt on toodud järgnevates peatükides.

2.2.1 Panga poolsete lahenduste analüüs

Alguses vaatleme Eesti topp pankade (Swedbank, SEB, Luminor, LHV, Coop Pank) lahendusi [3]:

- Swedpanga poolne lahendus pakub arvutust nii liisingule (sobib uue või vähe kasutatud sõiduki ostuks), kui ka autolaenule (sobib kasutatud sõiduki ostuks) [4].

- SEB [5] ja LHV [6] poolsed lahendused pakuvad arvutust kasutusrenti (sobib nendele, kes soovivad liisingu lepingu lõppedes sõiduki tagastada) ja kapitalirenti (kes soovivad liisingulepingu kehtivuse ajal tasuda kogu sõiduki maksumuse ning saada lepingu lõppedes sõiduki omanikuks) liisingule [5].

- Luminori [7] ja Coop Panga [8] poolsed lahendused pakuvad liisingu arvutust kapitalirenti ja kasutusrenti näol ning auto laenu arvutust.

Järgnev tabel summerib panga poolsete auto liisingu/laenu lahenduste võimalusi.

Tabel 1. Pangapoolsete lahenduste võrdlus

Komponent	Parameeter	Swedbank (liising)	Swedbank (laen)	SEB	Luminor (liising)	Luminor (laen)	LHV	Coop (liising)	Coop (laen)
Sisend	Sõiduki hind	+	+	+	+		+	+	
	Periood	+	+	+	+	+	+	+	+
	Sissemaks	+		+	+		+	+	
	Jääkväärtus	+		+	+		+	+	
	Intressimäär			+	+		+		
	Laenusumma					+			+
Väljund	Lepingutasu	+	+			+			
	Kuumakse	+	+	+	+		+	+	
	Maksegraafik			+			+		
	Võrdlus			+					

Nagu saab antud tabelist näha, põhiline väljund pangapoolsetes lahendustes on kuumakse, mis kujuneb kas liisingu või laenu maksust. Selle sama kuumakse arvutusse ei ole arvesse võetud lepingutasu, mis võib palju erineda.

Peale pangapoolsete lahendusi leidub ka palju alternatiivseid autokulude arvutusi teostavaid lahendusi. Järgmine peatükk räägib nendest täpsemalt.

2.2.2 Alternatiivsete lahenduste analüüs

Alternatiivseid lahendusi saab jaotada viiesse rühma, arvutusmetoodikast lähtudes:

1. Kogukulule põhinevad
2. Liisingu/laenu arvutusele põhinevad
3. Enda sisendandmetel põhinevad
4. Läbisõidule põhinevad

5. Mineviku andmetel põhinevad

Esimesse rühma, mille lahendusi leidub kõige rohkem, kuuluvad kogukulu arvutusele põhinevad kalkulaatorid. Nende seast leidub lahendusi, mis võtavad arvesse palju parameetreid, millel on olemas võrdluse võimalus, arvutavad kogukulu, kulu aasta ja distantsi kohta [9]. Mõned teevad hea graafilise kulude jaotuse, ning raporti allalaadimise võimalust PDF vormingus [10]. On ka lahendusi mis lisaks liisingumakse arvutusele kalkuleerivad ka sõiduki omamise kulu [11] [12]. Leidub ka selliseid lahendusi, mis peale muid sisestavaid parameetreid omavad võimalust seadistada odavnevuse protsenti aastate kaupa [13]. Mõned neist teevad kulude jaotuse aastate kaupa, ning näitavad odavnevuse graafiliselt [14] [15] [16]. Esimesse rühma kuuluvate lahenduste puudujääkideks võib nimetada vaadeldavate autode võrdluse piirangut vaid kuni kahele autole [9] [17] [16] [18] ning antud rühma kõigil lahendustel puudub salvestuse võimalus hilisemaks järele vaatamiseks.

Teisse rühma kuuluvatele lahenduste eeliseks võib tuua kulu arvutuse kuu kohta [19] [20], ning maksegraafiku olemasolu [20]. Puudujääkideks on aga piirang vaid laenu või liisingu arvutusele. Ehk antud lahendused on samaväärsed pangapoolsete lahendustega, millest räägitud peatükis 2.2.1.

Kolmandasse rühma kuuluvate lahenduste eeliseks on lihtne ja kiire sõiduki sisestamise võimalus. Kuid puudujäägiks on see, et arvutus põhineb enda (teenuse pakkuja poolt põhinevatel) andmetel, mida pole võimalik muuta, ning suunatud vaid kindla riiki klientidele [21] [22].

Neljandasse rühma kuuluvad vaid läbisõidul põhinevad lahendused, muud parameetrid on neil enda poolsed, ning neid muuta ei saa. Eeliseks on kasutuse lihtsus, ning võrdluse võimalus, kuigi piiratud sõidukite arvu kohta.

Viiendasse rühma kuulub mineviku andmetel põhinev lahendus, mis annab küll hea ülevaadet kulude jaotusest, kuid tulemuste saamiseks peab kasutaja kõik kulud eelnevalt salvestama (tekitama ajaloolise andmete baasi), ning alles siis tekivad arvutuste väljundid.

Alternatiivsete lahenduste sisend ja väljund parameetrite võrdlustabel on toodud Lisas 2.

2.2.3 Olemasolevate lahenduste analüüsi kokkuvõte

Analüüsi käigus oli lõputöö autori poolt otsustatud mitte arvesse võtta pangapoolsete lahenduste sisendeid, kuna kasutaja saab alati pöörduda pangapoolsete lahenduste juurde, ning teha seal sissemakse, liisingu, ning kuumakse arvutust. Antud parameetrid ei näita auto reaalse omamise kulu, kuna mõni võtab auto liisingusse viieks aastaks, ning selle auto odavnevus võib lihtsasti ületada pangapoolseid kuumakseid, mõni aga võtab laenu mõneks kuuks ning antud juhul selline kuumakse ei pruugi olla võrdne omamise kuluga.

Sõiduki omamise kogukulu, perioodi (€ /Kuu) ja distantsi (€ /km) kohta, arvutamiseks, sisendparameetriteks olid valitud auto andmetega seotud sisendid (mark, mudel, aasta, lisainfo, veebilink auto kuulutusele), kütusega seotud parameetrid (kütuse kulu, kütuse hind, läbisõit), ning regulaarsed kulud (liikluskindlustus, kaskokindlustus, hooldus ja remont, odavnevus ja muud kulud).

Sellise parameetri nagu kütusekulu tüüp (n. linnas, maanteel, jne.) oli esialgu otsustatud mitte kaasata kasutajaliidese vaatesse, kuid andmebaasi olme-suhte diagrammi koostamisel tuleb sellega arvestada, et tulevikus oleks võimalus kütuse kulu sisestamisel kütusekulu tüüpi kasutada.

3 Tehnoloogia valik

Käesoleva töö praktilise osa teostamiseks on vaja leida sobilikud tehnoloogiad MVP arendamiseks. Järgnevad peatükid vaatlevad hulka serveripoolseid, ning kliendipoolseid tehnoloogiaid. Viimases alampeatükis on räägitud ka andmebaasi mootorite valiku võimalustest. Peatükki lõpus on toodud valiku põhjendus.

3.1 Tehnoloogiate ülevaade

Selleks, et luua lõputöö probleemis püstitatud lahenduse on mitu võimalust, näiteks kasutusele võta LAMP või .NET arenduspinud, mis esindavad veebiarenduses kahte domineerivat konkureerivat tehnoloogiat [23].

LAMP on vabavaraline ja avatud lähtekoodiga tarkvara kombinatsioon. See viitab Linuxi operatsioonisüsteemile, Apache HTTP serveri, MySQL andmebaasitarkvara ja programmeerimiskeelte PHP, Perl ja Python esitähtedele [24].

Kuigi LAMP on jõudluse ja lõpptoote poolest võrreldav ASP.NET-iga, on keerukate rakenduste arendamine ja tehniline hooldus LAMP-i abil keerulisem ja problemaatilisem kui tehes sama ASP.NET-iga. Kui LAMP on ideaalne väiksemate projektide jaoks, siis ASP.NET sobib mastaapsete arenduste jaoks paremini [23].

3.1.1 Serveripoolsed tehnoloogiad

Serveri poolsed tehnoloogiad on vajalikud kommunikatsiooniks serveri, rakenduse ja andmebaasi vahel. Laialt levinud programmeerimisraamistikud (keeled) on ASP.NET Core (C#), Laravel (PHP), Ruby on Rails (Ruby), Django (Python) ja Spring Boot (Java) [25]. Järgnevad alampeatükid tutvustavad neid täpsemalt.

3.1.1.1 ASP.NET Core

ASP.NET Core on tasuta, vabavaraline veebiraamistik mis on arendatud Microsoft-i poolt. See võimaldab ehitada tagarakendusi kaasaaegsete veebirakenduste ning ka web APIde jaoks. Programmeerimiskeeleks on kasutatud C#, või mõni muu .NET põhinev keel. See töötab nii Windows kui ka muudel platvormitel.

Raamistiku mõlemad osad - MVC ja WEB API - aitavad kaasaegsete veebirakenduste loomisel. MVC on mõeldud selliste traditsiooniliste veebirakenduste loomiseks, kus

renderdamine toimub serveri pool, kuid see toetab ka kaasaegsete JS-teekide ja kliendipoolse renderdamise integreerimist. ASP.NET pakub paljusid valmiskujul veebiarenduse funktsioone, näiteks turvalisus, andmete valideerimine, juurutamine ja muud. ASP.NET Web API on mõeldud RESTful veebiteenuste loomiseks kaasaegsete kasutajaliideste, mobiilirakenduste ja teiste sõlmede *(end-point)* jaoks [26].

3.1.1.2 PHP

PHP on üldotstarbeline programmeerimiskeel, millel on ülisuur arendajate, raamistike ja teekide ökosüsteem. PHP-l ei ole selliseid reegleid nagu kompileeritud keeltel[1] või rangeid standarte, mida võib näha Pythoni puhul, selle asemel on saadaval arendajaskonnalt pärinevad juhised. Selle tulemusel võib suuremate, ilma range struktuurita projektide lugemine ja hooldus muutuda keeruliseks, tänu „spagetikoodile“ [25].

PHP raamistik Lavarel on mõeldud veebi rakenduste arendamiseks. Oma paindlikkuse, funktsioonide ja mugavate tööriistade tõttu on Lavarel üks kõige tõhusamaid ja populaarsemaid raamistikke. Lavareli kasutamine on kiire ja lihtne. Raamistiku süntaksit peetakse elegantseks ja väljendusrikkaks [27].

3.1.1.3 Python

Teadusringkondades populaarne Python on avatud lähtekoodiga programmeerimiskeel, mis paistab silma koodi loetavuse, lühikese koodi ja ulatuslike teekidega. See sobib rakendustele, mis skaleeruvad horisontaalselt (lisades oma ressursside hulka rohkem masinaid) üle olekuvabade serverite (mis ei hoia serveri teavet ega seansi üksikasju enda teada), mis teeb sellest hea lahenduse pilve kasutavate rakenduste jaoks [25].

Django on Pythoni põhine vaba ja avatud lähtekoodiga veebiraamistik, mis järgib mudel-vaade-controller arhitektuurimustrit ja võimaldab turvaliste ja jätkusuutlike veebisaitide kiiret arendamist [27].

3.1.1.4 Ruby

Ruby on dünaamiline, objektiorienteeritud, üldotstarbeline programmeerimiskeel. Ruby on populaarne programmeerimiskeel DevOpsi raamistikele, nagu näiteks Puppet ja Chef.

---

[1] PHP on interpreteeritud ehk tõlgitud arvutile mõistetavasse keelde [72]

Rubyl on raamistike ja teekide haldamiseks isegi oma süsteem nimega RubyGems. Neil on valikus üle 60 000 teegi ja Rubyl on väga aktiivne arendajate kogukond [25].

Ruby on Rails (RoR) on Ruby abil kirjutatud serveripoolne veebirakenduste raamistik. Selle raamistiku peamine eesmärk on muuta veebisaitide ja rakenduste loomise protsessi lihtsamaks. Arendajatele meeldib Ruby on Rails, sest see vähendab tavapärastele ülesannetele (nagu vormi, menüü või tabeli loomine) kulutatud aega, kuna saab valida valmiskujul lahenduse, mida saab korduvalt kasutada. Seetõttu ei raiska arendajad aega veebisaidi või rakenduse nullist ehitamisele. Sellel on ka puudusi. Näiteks kõik veebimajutuse pakkujad ei toeta RoR-i rakendusi [27].

3.1.1.5 Java

Java on üldotstarbeline programmeerimiskeel, mis on klassipõhine, objektorienteeritud ja spetsiaalset loodud võimalikult väheste rakendussõltuvustega. Tuginedes filosoofiale „kirjuta üks kord ja kasuta kõikjal“ on Java 2021. aastal populaarsuselt teine programmeerimiskeel. Java pikk kasutusiga ja laialdane kasutuselevõtt on loonud tugeva ökosüsteemi dokumentatsioonist, teekidest ja raamistikest, millest paljud on suunatud e-kaubanduse, turvalisuse ja keerukate tehingustruktuuride tegemisele [25].

Spring on Java kõige populaarsem veebirakenduste arendamise raamistik. Miljonid arendajad kõikjal maailmas kasutavad Spring raamistikku suure jõudlusega hõlpsasti testitava ja korduvkasutatava koodi loomiseks. Spring on suuruse ja läbipaistvuse poolest kerge, põhiversioon on umbes 2MB [28].

3.1.2 Kliendipoolsed tehnoloogiad

Kasutajaliidese kirjutamine puhtalt JavaScriptiga on võimalik, varem nii veebisaite arendatigi, kuid aastate jooksul on tekkinud mitmeid kooditeeke, mis kasutavad erinevaid juhtelemente ja võimaldavad koodi korduvkasutatavust. Kasvades on mõned nendest teekidest ühinenud, et luua raamistik, mis aitab arendajatel luua täielikke veebisaite algusest lõpuni, pakkudes samas ka vastuseid paljudele kaasaegsete veebisaitide poolt nõutavatele küsimustele seoses marsruutimise, autentimise, andmete sidumise, oleku juhtimise ja muuga. Kolm tänapäeva kõige populaarsemat JavaScripti raamistikku on: Angular, React ja Vue.js [26]. Järgnevad alampeatükid räägivad nendest täpsemalt.

3.1.2.1 Angular

Angular on TypeScripti põhine avatud lähtekoodiga veebirakenduste raamistik, mida juhib Google'i Angulari meeskond ning üksikisikute ja ettevõtete kogukond. Angular on täielikult ümberkirjutatud sama meeskonna poolt, kes lõid AngularJS-i. Angular kasutab reaalset DOM-i (mida on äärmiselt raske käsitleda), mis mõjutab selle jõudlust (muudab selle aeglaseks) ja võimaldab sellel teha dünaamilisi tarkvararakendusi. Angular on täielik raamistik ja seega pakub kõike vajalikku alates marsruutimisest kuni mallideni. See on rakenduse struktureerimisel paindlik. Koodi kirjutamist saab hõlpsalt alustada kõige vajaliku abil, mida Angulari pakett sisaldab, kuid sellel on järsk õppimiskõver, sest see on täielik raamistik [29].

3.1.2.2 React

React, mis lasti välja 2013. aastal ja mida haldab Facebook, on komponentidele keskenduv JavaScripti SPA raamistik, mis täidab vaatemootori (*view engine*) eesmärki. Vastupidiselt Angularile on Reactil vähem ütlemist selle kohta, kuidas kogu rakenduse koostatake, ning selle õppimiskõver on tagasihoidlikum. React võimaldab ehitada kapseldatud ja korduvkasutavaid komponente JSX-i abil (domeenispetsiifline keel), mis võimaldab kirjutada komponendi vaate JavaScriptis koos komponendi loogilise koodiga. Alates välja tulemisest on React ühtlases tempos populaarsust kogunud, olles juba mõnda aega juhtivaks SPA raamistikuks [26].

3.1.2.3 Vue.js

Vue.js lasti välja 2014. aastal ja selle töötas välja endine Google'i töötaja Evan You. See on võtnud varem käivitatud raamistikest üle kõik positiivsed aspektid, näiteks virtuaalse DOM-i funktsiooni, mis suurendab jõudlust. Võrreldes teistega on Vue.js kõige kergem. Selle väike teek sobib mistahes kergete rakenduste arendamiseks. Vue raamistik töötab samal moel nagu React raamistik. Vue-d on lihtsam õppida, sest see pakub suuremat kohandatavust. Sellel on sama funktsionaalsus ja komponentide kasutamine nagu Angularil ja Reactil [29].

3.1.3 Andmebaasid

Konkreetset tüüpi andmebaasi (relatsiooniline või objektorienteeritud) sobivus võib teatud määral sõltuda selle kasutaja vajadustest. Kui E.F. Codd töötas välja oma relatsiooniliste andmebaaside teooria (mis oli esmakordselt avaldatud 1969 a.), otsis ta lähenemisviisi, mis rahuldaks võimalikult suurt hulka erinevaid kasutajaid ja

kasutusvõimalusi. Seega kujundame relatsioonilist andmebaasi ilma, et üritaksime ette näha selle konkreetseid kasutusi ja ilma, et lisaksime teatud rakendustele eelistavaid kõrvalekaldeid. See ongi vast relatsioonilise lähenemise eripära [30].

Lähtudes DB-Engines Ranking-ule top nelja relatsiooniliste andmebaasimootorite hulka kuuluvad: Oracle, MySQL, Microsoft SQL Server, PostgreSQL [31]. Järgnevates peatükides vaatleme neid lähemalt.

3.1.3.1 Oracle

Oracle on loonud ja seda haldab Oracle Corporation. Praegu toetab see ühes andmebaasis mitut andmemudelit, nagu dokument, graaf, relatsiooniline ja võtmeväärtus. Oma viimastes väljaannetes keskendus ta pilvandmetöötlusele [32].

Oracle keskendub infoturbele (aktiivsele andmekaitsele, sektsioonidele, täiustatud varundamisele ja taastamisele), sel on tugev tehniline tugi ja dokumentatsioon. Lisaks omab suurt mahutavust (Oracle'i mitmemudeliline lahendus võimaldab mahutada ja töödelda tohutul hulgal andmeid) [32].

Oracle-l on ka miinuseid, milleks on kõrge hind (Standard Edition, mis ei sisalda kõiki saadaolevaid funktsioone, maksab 17 500 dollarit ühiku kohta. Enterprise Edition on üle 47 000 dollari ühiku kohta). See on ressursse nõudev, vajab võimsat infrastruktuuri, ning on raske õppimiskõveraga. Selle halduseks peab kaaluma spetsiaalsete ekspertide palkamist [32].

3.1.3.2 MySQL

Algselt vabavaraline lahendus MySQL on praeguseks Oracle Corporation valduses. Tänapäeal on see osa Linuxist, Apache'ist, MySQL-ist ja Perl / PHP / Pythoni pinust. Tuginedes C ja C++ töötab MySQL hästi selliste süsteemiplatvormidega nagu Windows, Linux, MacOS, IRIX ja teised [32].

MySQLi plussideks võib tuua: individuaalseks kasutamiseks mõeldud põhivahendite komplektiga ning tasuta kogukonnaväljaanne (*community edition*), lihtne süntaks ja mõõdukas keerukus (näiteks saab suurema osa ülesannetest täita otse käsureal, vähendades arendusetappe). Sel on hea pilvega ühilduvus, on saadaval sellistel juhtivatel platvormidel nagu Amazon, Microsoft ja teised nagu zone.ee, veebimajutus.ee, jne. [32];

MySQL on varustatud skriptiga, mis aitab andmebaasi turvalisust parandada, seadistades installi parooli turbetaseme, määrates juurkasutajale parooli, eemaldades anonüümsed kontod ja eemaldades testandmebaasid, millele on vaikimisi juurdepääs kõigil kasutajal [33].

MySQLi miinusteks võib tuua: mastaapsuse väljakutsed, kuna MySQL ei ole loodud selle mastaapsust silmas pidades. Sel on osaliselt avatud lähtekood (Kuigi MySQL-il on avatud lähtekoodiga osa, kuulub see enamasti Oracle'i litsentsi alla, mis piirab MySQL-i kogukonda DBMS-i täiustamise osas). Lisaks ei toeta see mõnda standardset SQL-i funktsiooni, ehk probleemid võivad ilmneda siis, kui peab minema üle teistele andmebaasidele [32].

3.1.3.3 Microsoft SQL Server

SQL Server on tõestanud end andmebaasina, mida spetsialistid armastavad kasutada [34]. See tuleb hästi toime relatsiooniandmete tõhusa salvestamise, muutmise ja haldamisega. SQL Serveri andmebaasidega suhtlemiseks kasutavad andmebaasi insenerid tavaliselt keelt Transact-SQL (T-SQL), mis on SQL-i standardi laiendus [32].

MSSQLi miinusteks võib tuua kallidus, kuna on endiselt üks kallimaid lahendusi (Enterprise väljaanne maksab üle 14 000 dollari tuuma kohta, mida müüakse kahe tuuma kaupa). Sel on ka keerukas häälestusprotsess, kus võib päringute optimeerimise ja jõudluse häälestamisega töötamine olla problemaatiline [32].

3.1.3.4 PostgreSQL

PostgreSQL, tuntud ka kui Postgres, positsioneerib ennast kui "maailma kõige arenenum avatud lähtekoodiga relatsiooniline andmebaas" [33]. Seda loodi eesmärgiga olla väga laiendatav ja standarditele vastav. See rakendab enamuse SQL:2011 standardist [35] ja vastab ACID[1]-kriteeriumitele. PostgreSQL töötab enamikel operatsiooni süsteemidel [36].

PostgreSQL plussideks võib tuua vertikaalset skaleeritavust – võimalust suurendada olemasoleva riist- või tarkvara mahtu ressursside lisamise abil. Postgres toetab vaikimisi

[1] ACID - (relatsioonilisele / transakstsioonilisele) andmebaasile seatud piirangute komplekt, mille täitmine tagab, et andmebaasitransaktsioonid on usaldusväärsed (so. terviklikud ja üheselt mõistetavad) [51]

suurt hulka andmetüüpe, näiteks JSON, XML, H-Store, GUID, bool, jt. See on lihtsalt integreeritavad kolmanda osapoolte programmidega, näiteks DB Data Difftective - andmete võrdlemiseks ja sünkroonimiseks, või pgBackRest – andmete varundamis- ja taastamiseks. Ning Postgres on täiesti avatud lähtekoodiga [32].

PostgreSQL miinusteks võib tuua: mittekooskõlaline dokumentatsioon (puudub järjepidevus ja täielikkus); aruandlus- ja auditeerimisvahendite puudumine (alati on oht, et andmebaasi insenerid märkavad riket liiga hilja) [32].

3.2 Valiku põhjendus

Valiku tegemisel lähtuti tehnoloogiate kirjeldusest, projekti nõuetest, autori tehnilisest kompetentsusest, ajapiirangust, rakenduse arengu võimalusest, algse majutamise plaanist, ning turvalisuse tagamise nõuest.

Andmebaasi valik jäi *PostgreSQL* lahenduse poole, kuna tegu on lihtsasti skaleeritava, vabavaralise, toetava *Dotnet*-is kasutatavate andmetüüpe (nagu näiteks GUID ja Boole'i tüüp) ning täiesti avatud lähtekoodiga lahendusega. Lisaks sellele on see vaikimisi sadaval kõigil Debian (Raspberri Pi 4 operatsioonisüsteem) versioonidel.

Tagarakenduse valik jäi ASP.NET Core 5.0 peale, mis on multiplatvormiline, kaasaaegne, võimas, ning vabavaraline lahendus. Antud tehnoloogia põhjal saab mugavalt ehitada RESTful API servist, mida saab tulevikus kasutada näiteks mobiili rakenduste ühendamiseks tagarakendusega. Samas sai kasutusele võtta palju teadmisi omandatud hajussüsteemide aines. Tagarakenduse suhtlemiseks *PostgreSQL* andmebaasiga oli kasutatud Npgsql (ADO.NET[1] raamistiku andmepakkuja *PostgreSQL-i* jaoks), mis on vabavaraline, implementeeritud täielikult C# koodis, tasuta, ning avatud lähtekoodiga [37].

Kliendi poolsete tehnoloogiate seast jäi valik Bootstrap ja Vue.js peale, tänu selle kiirele õpimiskõverale, lihtsusele, ning sobilikkusele ühelehe rakenduste tegemiseks.

---

[1] Andmepakkuja, mis on kasutatud andmebaasiga ühendamiseks, käskude täitmiseks ja andmete hankimiseks [66]

4 Rakenduse analüüs

Rakenduse analüüsi peatükis on räägitud rakenduse nõuetest, minimaalse töötava toote kasutajaliidesest, kogukulu arvutusest, rakenduse arhitektuurist, selle turvalisuse tagamisest ning majutamisest.

4.1 Rakenduse nõuded

Rakenduse loomiseks on vaja infosüsteemi nõudeid välja selgitada. Tarkvara nõuded spetsifitseerivad, milliseid funktsioone peab süsteem realiseerima (funktsionaalsed nõuded) ja kuidas neid funktsioone täidetakse (mittefunktsionaalsed nõuded) [38]. Järgnevad peatükid vaatlevad funktsionaalseid ja mittefunktsionaalseid nõudeid detailsemalt.

4.1.1 Funktsionaalsed nõuded

Lahendus peaks olema veebirakendusena teostatud. See peaks olema interneti võrkust ligipääsetav. Kasutajatel peaks olema õigus isikliku konto registreerimiseks, ning hiljem vajadusel parooli muutmiseks. Peaks olema kasutuse võimalus nii tavakasutajale, kui ka administraatorile ja andmete administraatorile, kellel oleksid isiklikud ligipääsu õigused. Igal kasutajal peale registreerimist peaks olema auto info-, läbisõidu-, kütusekuluandmete, kulude ja muude parameetrite (n. veebilink) sisestamise võimalus.

Peale auto andmete sisestamist peaks olema võimalus võrrelda sisestatud autode kogukulud kilomeetri ja kuu kohta. Peaks olema võimalus salvestada kalkuleeritud andmed hilisemaks järele vaatamiseks, ning võimalus antud andmete muutmiseks. Samuti peaks olema võimalus sorteerida tabelina toodud autode andmed.

Andmete administraatoril peaks olema võimalus saada ligi, ning vajadusel lisada, muuta või kustutada põhiandmeid, sellised nagu näiteks kulu tüübid, ajavahemikud, valuutad, jne. Rakenduse administraatoril peaks olema võimalus kasutajate ligipääsude haldamiseks: rollide haldamiseks, õiguste muutmiseks, ning ka kasutajate kustutamiseks.

4.1.2 Mittefunktsionaalsed nõuded

Kasutajaga seotud andmed peavad olema privaatsed, kättesaadavad vaid kasutajale endale, aga administraatorile nähtav vaid kasutajanimi (paroolid on peidus, ning

andmebaasis nähtav vaid paiskeväärtusena). Rakendus peab olema võimeline teenindama ligi 100 inimest korraga, ilma jõudluse languseta. Rakendus peab olema ligipääsetav erinevatest brauseritest: Chrome, Mozilla, Safari, Edge, jne.

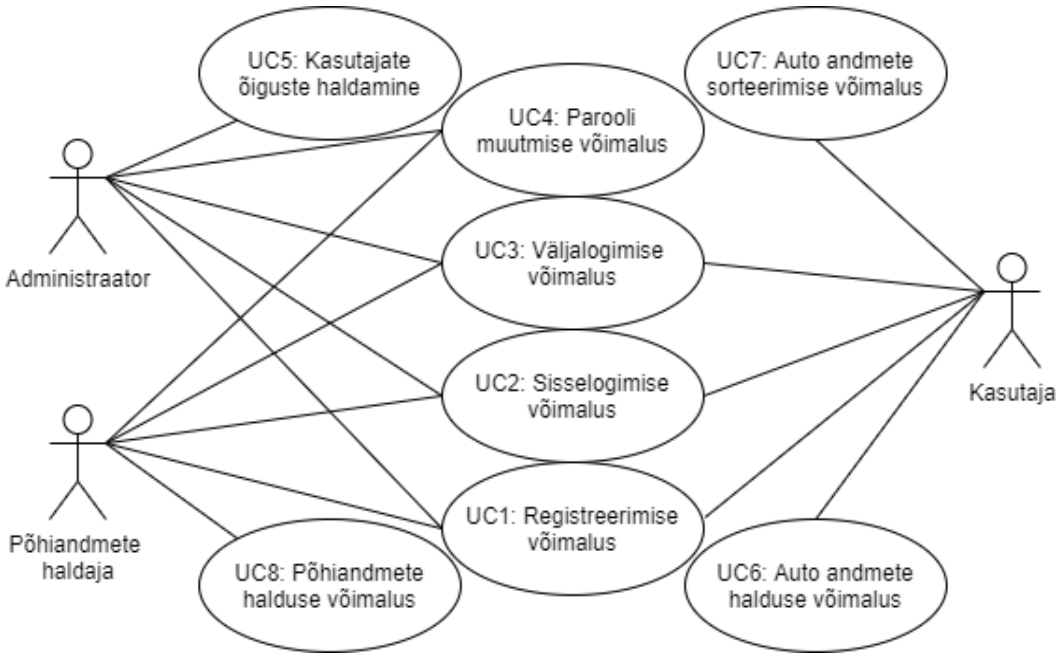
Tagarakendus peab töötama Docker konteineris Raspberry Pi 4 masinas, samas võimalusega seda üle viia Microsoft Azure, Google Cloud, AWS pilve, Heroku või mõne muu pilveteenuse peale. Tarkvara peab olema kirjutatud puhta koodi heade tavade järgi, arendusvõimalusega, ning loetav ka muu keelt kõnevale isikule. Süsteemi seisakud ei tohiks olla pikemad kui viis tundi kuus (ning ühekordne hoolduse aeg ei tohiks ületada kolme tundi). Tagarakendus peab omama dokumenteeritud ning versioonihaldusega APId.

4.2 Kasutajaliides

Selleks et luua mugava, hea kasutatavusega rakenduse kasutajaliidest on peale sisestavate parameetrite (toodud välja peatükis 2.2.3.) vaja välja selgitada ka lõppkasutajate kasutusjuhud. Selleks oli kasutusele võetud UML diagrammi kasutusmallide mudel.

4.2.1 Kasutusmallide mudel

Antud peatükk tutvustab loodud kasutusmallide mudeli, ning toob välja ka ühe kasutusjuhu kirjelduse. Ülejäänud UML-i kasutusjuhud on toodud Lisas 3.



Joonis 1. Kasutusmallide mudel

Rakenduse kasutusjuhud on toodud ülaltoodud joonisel. Kasutajaskond on jaotatud kolme rühma: kasutaja (ehk lõppkasutaja kes hakkab rakenduse funktsionaalsust kasutama), põhiandmete haldaja ning administraator (kõige suuremate võimalustega kasutaja, kes saab teiste kasutajate ligipääsu õigusi hallata).

4.2.1.1 Kasutusjuhtumi ülevaade

Järgnev tabel kirjeldab detailselt ühe kindla kasutusjuhtumi – auto andmete halduse võimalust. Iga kasutusjuhtumi jaoks on toodud välja selle eesmärk, oodatav tulemus, eeltingimused, osalejad, ehk rakenduse kasutajad, ning kindlasti ka peamine stsenaarium, mille läbides peaks olema saavutatud esialgu oodatud tulemus. Olemasolul toodud ka alternatiivne stsenaarium, ning ka järele tingimus, saavutanu kinnituseks.

Tabel 2. UML-i kasutusjuhus 6 – Auto andmete halduse võimalus

Nimi	**UC6 Auto andmete halduse võimalus**
Eesmärk	Sisestada/lugeda/muuta/kustutada auto andmed
Tulemus	Auto andmed sisestatud, kogukulu arvutatud, andmed toodud tabelis välja. Auto andmed loetud, kogukulu arvutatud, andmed toodud tabelis välja. Auto andmed muudetud, kogukulu arvutatud, andmed toodud tabelis välja. Auto andmed kustutatud.
Eeltingimus	Klient on sisse logitud.
Aktorid	Klient
Peastsenaarium	1. Klient soovib sisestada/lugeda/muuta/kustutada auto andmed 2. Klient sisestab/muudab autoga seotud parameetrid(mark, mudel, aasta, kommentaar, veebilink, kütusekulu, kütuse hind, läbisõit), kulud (kasko maksumus, liikluskindlustus, hooldus ja remont, odavnevus, muud kulud), valib kuludele ja läbisõidule ajavahemik (aasta, kuu, üksik kulu) 3. Vajutab nuppu Arvuta/Kustuta 4. Auto kogukulu kilomeetri ja kuu kohta on arvutatud/kustutatud 5. Andmed on salvestatud/kustutatud andmebaasi 6. Sisestatud andmed on/ei ole toodud tabelis välja
Alternatiivne stsenaarium	Auto andmed sisestatud/loetud/uuendatud/kustutatud administraatori poolt otse andmebaasi, kasutades SQLi.
Järeltingimus	Kliendil on võimalus hiljem pääseda salvestatud andmete ligi.

Muud kasutusjuhud on toodud Lisas 5.

4.2.2 Sisendparameetrite vaade

Tuginedes alternatiivsete lahenduste analüüsi väljundi (toodud välja peatükis 2.2.3), sisestavate parameetritele ning võttes arvesse sisendi kasutusjuhtude analüüsist, oli otsustatud luua Bootsrap[1]-i malli järgi klientrakenduse kasutajaliidese vaadet. Selle joonis on toodud Lisas 4.

Sisendparameetreid saab jaotada kolme rühma. Esimesse kuuluksid sisestava sõiduki informatiivsed andmed. Nendeks näiteks sõiduki mark, mudel, sõiduki esmaregistreerimise aasta, kommentaari väli, kus saaks lisada täpsustava informatsiooni sõiduki kohta, näiteks mootori maht, sõiduki müügihind, või kütuse tüüp. Teises osas oleksid kütuse kuluga seonduvad andmed, mille järgi saaks kütuse kulu arvutada. Nendeks on siis sõiduki kombineeritud kütuse kulu, kütuse jooksev hind, ning oletav läbisõit ajavahemiku (kuu või aasta) valiku võimalusega. Kolmandasse ossa kuuluksid kõik ülejäänud kulukohad, ehk liikluskindlustus, kaskokindlustus, hooldus ja remont, odavnevus ja mõned muud kulud, nagu näiteks parkimine, autopesu, jne.

4.2.3 Väljundparameetrite vaade

Lähtudes sisendparameetritele ning peale „Arvuta“ nuppu vajutamist, tekkis koondtabeli vaade, toodud järgneval joonisel.

Insert Date	Make	Model	Year	Remark	Link	Cost/Km	Cost/Month		
27.04.2021	BMW	318i	2017	Diesel, AUT	https://www.auto24.ee/used/3338266	0.26€	319.42€	Edit	Delete
27.04.2021	HONDA	Civic	2012	1.8B, AUT		0.2€	243.82€	Edit	Delete
27.04.2021	LEXUS	IS	2007	2.5B, AUT		0.28€	347.8€	Edit	Delete
27.04.2021	MERCEDES-BENZ	E-Class	2008	E280, Diesel, AUT	http://www.auto24.ee/used/2256949	0.22€	272.75€	Edit	Delete
27.04.2021	KIA	Sportage	2011	B, AUT	http://www.auto24.ee/used/2263398	0.24€	301.53€	Edit	Delete
27.04.2021	VOLVO	V90	2017	Diesel, AUT	https://www.auto24.ee/used/3437165	0.46€	576.33€	Edit	Delete

Joonis 2. Kasutajaliidese väljundparameetrite vaade

Antud vaates saab näha arvutuse sooritamise kuupäeva, sõidukite informatiivseid andmeid, et lihtsasti neid eristada, ning ka auto omamise kogukulu arvutuse tulemust. Auto omamise arvutuse tulemust arvutakse nii läbitud kilomeetri kohta, kui ka kuu kohta.

[1] mallipõhine tööriistakomplekt veebilehtede loomiseks

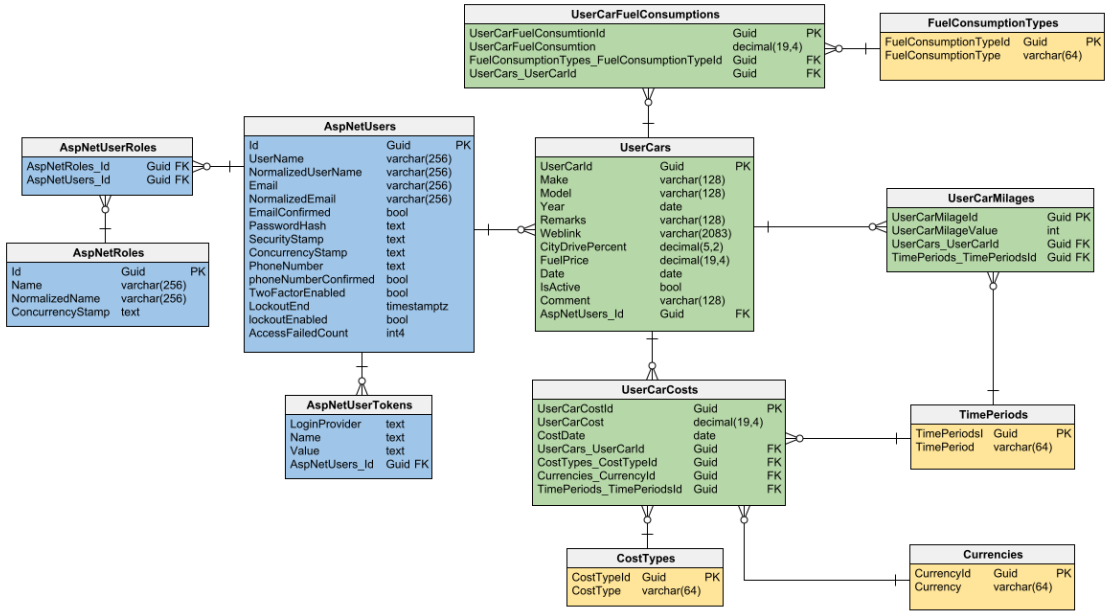
4.3 Arhitektuur

Järgmine peatükk räägib andmebaasi olme-suhte mudelist, tagarakenduse arhitektuurist, mainides RESTful API ja ASP.NET MVC rakendust, ning ka kliendipoolsest rakendusest.

4.3.1 Andmebaasi olme-suhte diagramm

Andmebaas on kirjeldatud olme-suhte diagrammina ja "varese jala" notatsioonina. See koosneb kaheksast põhitabelist, millele lisanduvad kasutajaga seotud tabelid. Rohelisega värvitud olemid on kasutaja poolsete andmete sisestuseks. Kollasega värvitud olemid on ettenähtud põhiandmete halduse jaoks. Sinised on eel-loodud ASP.NET rakendute individuaalse identifitseerimisega tegemisel, kasutaja andmete säilitamiseks.

Igas tabelis on kasutatud primaarvõtmeks, i.k. *Primary Key* (*PK*) automaatselt genereeritud *GUID* väärtus, ning ka muud metaandmed: looja, loomise aeg, uuendaja, uuenduse aeg. Joonisel 3 on toodud andmebaasi olme-suhe diagramm, i.k. *Entity Relationship Diagram* (*ERD*). Loodud olme-suhte diagrammi alusel genereeriti ehitatava andmebaasi, kasutades *Entity Framework ORM*[1]i, ning kood ettepoole lähenemist.



Joonis 3. Andmebaasi ERD skeem

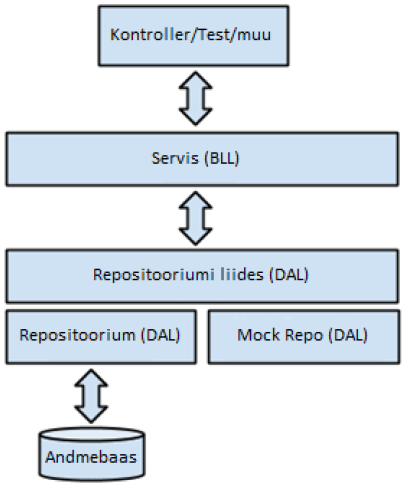
[1] Object–Relational Mapping

Atribuutidele peale vaadates saab märgata, et rahalised väärtused hoitakse andmebaasis decimal(19,4), ehk antud väli saab hoida 19 numbrit ning nende seas 4 numbrit peale koma kohta. Kuupäevi hoitakse *date* vormingus, ning veebilinki jaoks on ettenähtus 2083 tähemärki. Samuti vajab mainimist ka marki ja mudeli hoidmine *UserCar* tabelis. Antud lähenemine on tingitud asjaolust, et marki ja mudeli andmed hangitakse kliendipoolse rakenduse poolt väliselt avalikust rakendusliidesest, USA transpordiosakonna NHTSA[1] organisatsiooni poolt.

4.3.2 Tagarakenduse arhitektuur

Tagarakendus oli loodud .NET raamistikul. Kasutusele oli võetud mitmekihiline arhitektuur (andme-, ligipääsu-, äriloogika- ning esituskiht), üldine representatsioon sellest toodud joonisel 4, detailsem lisas 5 joonisel 10.

Rakenduse iga kiht täidab kindlat ülesannet. See annab kindla eraldamisloogika, mis lihtsustab rakenduse tulevast hooldust ja laiendamist. Mitmetasandilise arhitektuuri eeliseks on, et see pakub tsentraliseeritud ja spetsialiseeritud raamistikku, mis soodustab objektide korduvkasutamist. See tähendab, et koodi muudatusi tuleb rakendada ainult üks kord ja need kajastuvad kogu rakenduses. Näiteks esituskihi sisendi valideerimine delegeeritakse äriloogika objektidele. Kogu esituskihi valideerimise uuendamine nõuab ainult ühe äriloogika objekti uuendamist. Samuti mitmekihilise arhitektuuri rakendusesisest funktsionaalsust on lihtsam jälgida ning tagada turvalisust [39].



Joonis 4. Mitmekihiline arhitektuuri mudel [40]

[1] National Highway Traffic Safety Administration [67], ehk riiklik maanteede liiklusohutuse amet

Järgnev arhitektuuri kihtide lahti kirjutamine on piltlikult esitatud lisas 5, joonisel 8.

Andmebaasi poolt vaadates, algab kõik Domeeni klassidest , i.k. *Domain-ist* – domeeni mudeli klassid, mille põhjal migratsiooni tegemise käigus *Entity Framework* kood ettepoole lähenemisel moodustas andmebaasi objektid (tabelid) õigete olme-suhetega.

Sellele järgneb andmete ligipääsu kiht, i.k. *Data Access Layer* (DAL), kus on realiseeritud repositooriumid ja tööühiku i.k. *Unit Of Work* (UOW) mustrid. Repositoorium võimaldab mälus uuendada andmebaasist saadud andmeid *domain entity*-te kujul, kuna sisaldab kõik selleks vajalikud toimingud: salvestamine, pärimine, muutmine ja kustutamine, ehk CRUD operatsioonid. Antud mustrite kasutus võimaldab lihtsasti testida rakendus komponenditestidega (*unit test*), ühendades kontrolleri võlts andmetega – mälus olevate kollektsioonidega [41].

Repositooriumis küsitud muudatused ei ole koheselt teostatud andmebaasis. Muudatuste tegemiseks on UOW, mis kujutab endast ainsa transaktsiooni, mis konsolideerib mitu CRUD toimingut, ning alles siis täidab neid, tehes muudatusi mälus tehtud toimingutele reaalses andmebaasis. Selline lähenemine suurendab rakenduse jõudlust ja vähendab vasturääkivuste võimalust. Samuti vähendab see tehingute blokeerimist andmebaasitabelites, kuna kõik muudatused tehakse ühe tehinguna, mille päringud saab ORM optimeerida, rühmitades mitu tehingut sama tehingu raames, erinevalt paljudest väikestest ja eraldiseisvatest tehingute täitmistest [41].

Repositooriumi mustrit kasutades on äriloogika täielikult eraldi ja äriobjekte liigutakse edasi-tagasi. Repositoorium hoolitseb selle eest, kuidas ja kuhu andmeid tegelikult salvestatakse.

DAL kihile järgneb äriloogika kiht i.k. *Business Logic Layer (BLL)*, mis koosneb servisitest, vastutab äriloogika ja töövoo käsitlemise eest. Teenuse suhtlus andmebaasiga toimub läbi UOW. Äriloogika ja kasutajaliidese vahelüliks on kontrollerid.

Kasutajaliidese vaated - esituskiht, vastutab andmete korraldamise ja lõppkasutajale liidese edastamise eest. See võimaldab hoida kontrollerid puhtana äriloogikast, kuna kogu keeruline loogika on lükatud BLL-i ja kogu andmetele juurdepääsu kood on lükatud andmetele juurdepääsu kihti.

Igas kihis on oma andmeedastus objekt i.k. *Data Transfer Object* (DTO), näiteks DAL.App.DTO, BLL.App.DTO, PublicApi.DTO.v1, kus säilitatakse andmeid, ning liigutatakse kihtide vahel.

4.3.2.1 RESTful API

Üks tagarakenduse väljunditest on REST arhitektuuri laadil põhinev servis, ehk RESTful API. Antud lahendus töötab vahendajana klient-server suhtlemisel. See teeb andmed kättesaadavaks ressurssidena, ning tänu selle ühtlasele liidesele, erinevad rakendused (Android-, IOS[1]-, veebirakendused), saavad tagarakendusele ligi lõppsõlmedega suhtlemisel [42].

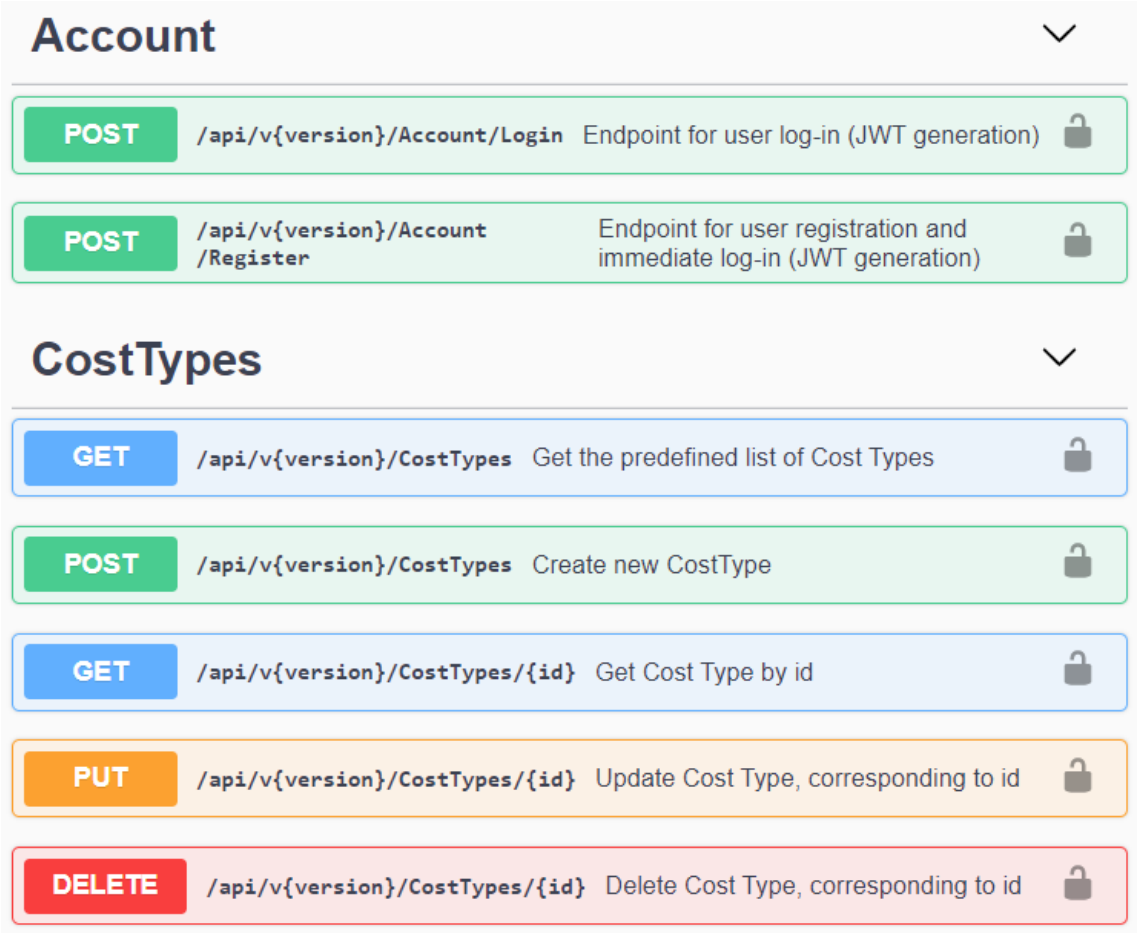
4.3.2.1.1 Dokumentatsioon

RESTful API kirjeldamiseks oli valitud vahendiks Swagger, mis põhineb OpenAPI spetsifikatsioonidel. See võimaldab nii arvutitel kui ka inimestel mõista RESTful API võimalusi ilma otsese juurdepääsuta lähtekoodile, ning vähendab aega korraliku dokumentatsiooni tegemiseks servise kirjeldamisel [43].

Swagger-i dokumentatsiooni kasutamiseks tuli paigaldada Swashbuckle, ning seadistada selle kasutamine *startup*-is. Tagarakenduses tuli kontrollerite sisu XML põhistega annotatsioonidega, mis võimaldaski Swagger kasutajaliidese kaudu dokumentatsiooni näha, ning ka testida iga avaliku API servist.

Swagger-i kasutajaliidese dokumentatsiooni tõmmis on toodud järgneval pildil. Dokumentatsioon on ligipääsetav lisades avalehe aadressile /swagger .

[1] Apple'i arendatav mobiilsete seadmete operatsioonisüsteem



Joonis 5. Swagger UI dokumentatsiooni tõmmis

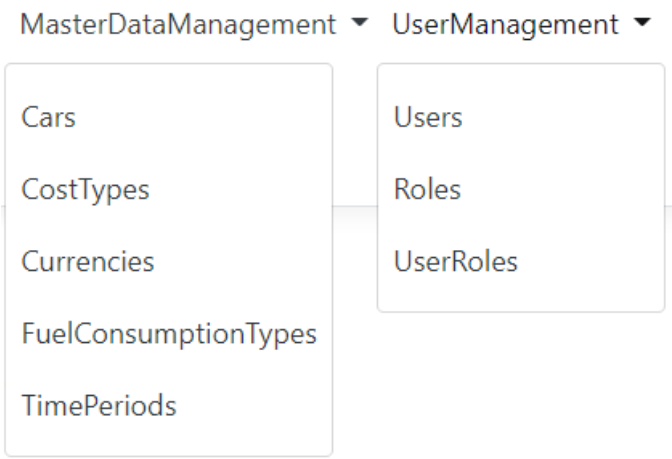
Kasutades *Swagger*-i kasutajaliidest on võimalik näha tabelites hoitud andmetüüpe, ning katsetada päringute saatmist ja vastuvõtmist. Tagarakenduse annotatsiooni põhine dokumenteerimine annab ka tehtud päringutele tagasisidet, tuues välja näiteks erinevad vead 401 (autoriseerimata), 404 (pole leitud), 500 (serveri viga), jne. Siinkohal on võimalik ka JSON Web Token (JWT) autentimine, ning andmete reaalne käsitlemine andmebaasis.

Lisaks API serviste kirjeldamisele, kasutusele oli võetud versioonihaldus, mis tulevikus rakenduse arengu korral võimaldab tekitada uusi versioone RESTful API servistest, jättes tööle eelnevate versioonide omad. Selleks jääb vaid päringute tegemisel mainida kasutuses oleva API versiooni.

4.3.2.2 ASP.NET MVC rakendus

Peatükkist 4.2.1. saab näha et kasutataval rakendusel lõppkliendiks osutub kolm rolli: klient, põhiandmete haldaja ning administraator. Lähtudes administraatori ja põhiandmete haldaja vajadustest, oli otsustatud luua antud rollide jaoks ASP.NET MVC

serveripoolne veebirakendus. Vaateid sai lihtsasti genereeritud kasutades koodigeneraatori. Lisaks sisselogimisele said rakendatud rolli põhised alad, kus saab teostada kõiki andmete halduse CRUD operatsioone.



Joonis 6. Administraatori ja põhiandmete haldaja alad ASP.NET rakenduses

Master Data Management-i ala on ette nähtud põhiandmete haldajale ning *User Management* on ette nähtud Administraatori rolli kasutajale.

ASP.NET Core Identity võimaldas lisada rakendusele sisselogimisfunktsiooni ja tegi sisse logitud kasutajatega seotud andmete kohandamise ja haldamise lihtsaks. *.NET Core Identity* pakus kasutajaandmetega töötamiseks mõningaid vaike klasse või juurutusi, näiteks *UserManager* ja *RoleManager*. *UserManager*-i sai kasutatud kasutaja informatsiooni haldamiseks, nt kasutaja loomiseks, eemaldamiseks, värskendamiseks jms. *RoleManager*-i sai kasutatud aga kasutaja rollide haldamiseks. Siinkohal võib luua uue rolli, eemaldada olemasoleva rolli jms. *RoleManager*-iga sai ka oma rollidele kasutajaid määrata.

Kliendi rolliga kasutajale on ettenähtud Vue.js põhinev kliendipoolne rakendus, millest räägib järgmine peatükk.

4.3.3 Kliendipoolne rakendus

Kliendipoolne rakenduse aluseks oli võetud *Bootstrap*-i šabloon, mis sisaldas vajalike komponente (HTML) ja funktsionaalsust (CSS, JS), et rakendus oleks lihtsasti arendatav, ligipääsetav eri brauseritest, ning näeks ilus välja. JavaScripti raamistikuks sai kasutatud Vue.js teeki, mis vahendab päringuid tagarakenduse RESTful API servisega. Kuna RESTful API ei hoia hetkeseisu klient rakenduse sessioonist serveril, kasutusele oli

võetud *Vuex* hetkeseisu juhtimise muster. Lehtede vahel sirvimiseks oli abiks *Vue Router*. RESTful API-ga suhtlemiseks oli kasutatud *Ajax* (Asynchronous JavaScript And XML) HTTP klient. See võimaldab veebilehtedel JavaScriptiga serverisse tagaplaanil päringuid teha ja andmeid vastu võtta, segamata avatud lehe kuvamist ja olekut [44].

4.4 Rakenduse turvalisus

Rakenduse turvalisuse peatükk räägib autentimisest, autoriseerimisest, erinevatest võimalikest turvalisusega seotud probleemidest ning kuidas neid rakenduse arendamisel hallati.

4.4.1 Autentimine

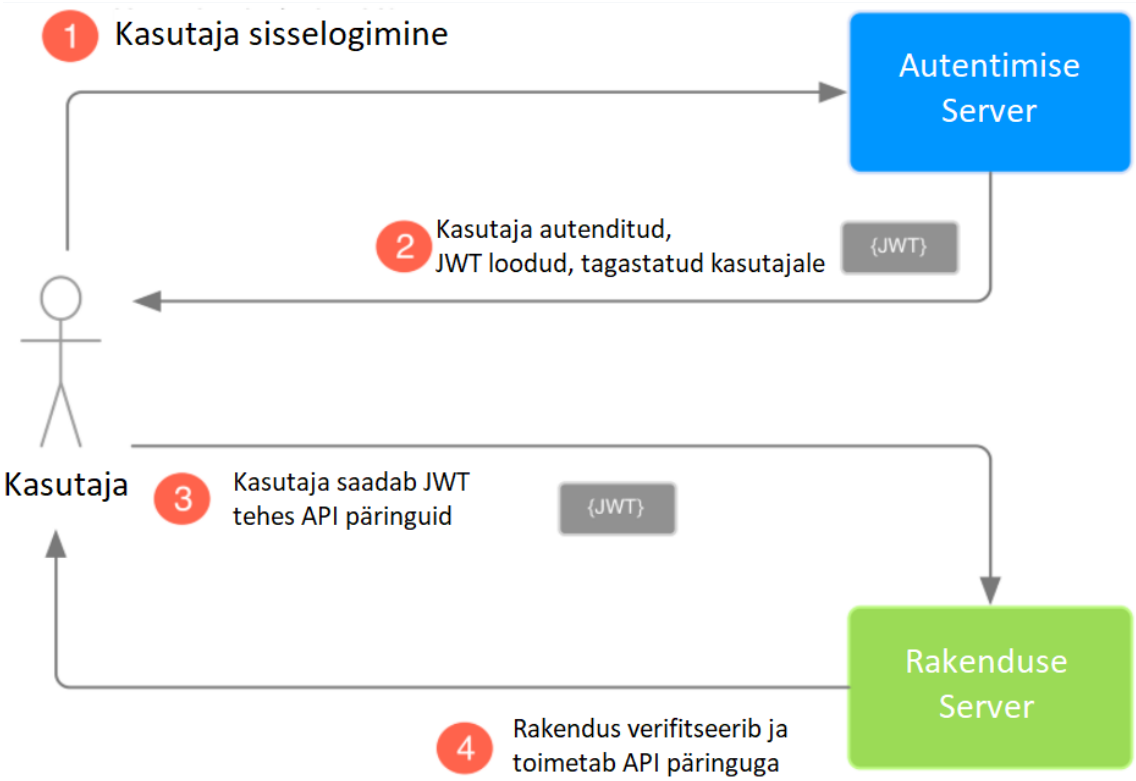
Rakenduse kasutamine algab autentimise protsessist. See on protsess, mille käigus kontrollitakse, kas kasutajal on veebisaidile pääsemise õigus või mitte. Kui kasutaja tuleb veebisaidile esimest korda, registreerub ta saidi kasutajaks. Kõik tema andmed, näiteks e-meili aadress ja parool, salvestatakse andmebaasi. Kui kasutaja sisestab oma e-meili aadressi ja parooli, kontrollitakse andmed andmebaasist üle. Kui kasutaja sisestab e-meili aadressi või parooli, mis ei vasta andmebaasile, kuvab sisselogimisleht teavituse „Probleem kasutajaga/parooliga!“, mis on ka üks turvaohutuse võtetest (tänu sellele, et ei räägi täpselt millega on probleem, kas kasutajaga, või parooliga). Kui kasutaja on sisestanud andmebaasis sisalduvaga ühtiva e-meili aadressi ja parooli, on ta autenditud kasutaja ja suunatakse edasi veebisaidi avalehele.

4.4.1.1 JSON Web token autentimine

Autentimise meetodiks Vue.js rakenduse jaoks oli valitud JSON Web token (JWT). See on laialt levinud autentimise meetod RESTful APIs, kuna luba (*token*) salvestatakse kliendipoolele. Antud lahendus on parem kui seansi küpsised (*session cookie*), järgnevalt siis sellest täpsemalt.

Seansi küpsiste tüüpi autentimismeetodi korral vastutab autentimise eest server ja klient ei tea, mis pärast päringu saatmist serveripoolel juhtub. See võib olla ka probleemiks, kui suur hulk kasutajaid saab ligi rakendusele samaaegselt. JWT aga saadetakse koos iga päringuga ja see sisaldab kogu kasutaja informatsiooni. JWT struktuuri poolest koosneb päisest, paketi sisust, ning allkirja struktuuri kontrollist. Selle kodeeritud ja dekodeeritud kuju on toodud lisas 6.

Järgneval joonisel on piltlikult näidatud autentimise põhimõte. Kui kasutaja saadab sisselogimis andmetega kasutaja autentimise taotluse, loob server krüpteeritud loa JSON-i veebi loa (JWT) kujul ja saadab selle kliendile tagasi. Kui klient saab loa, tähendab see, et kasutaja on autenditud mis tahes toimingu sooritamiseks [45].



Joonis 7. JWT autentimise põhimõte skeem [42]

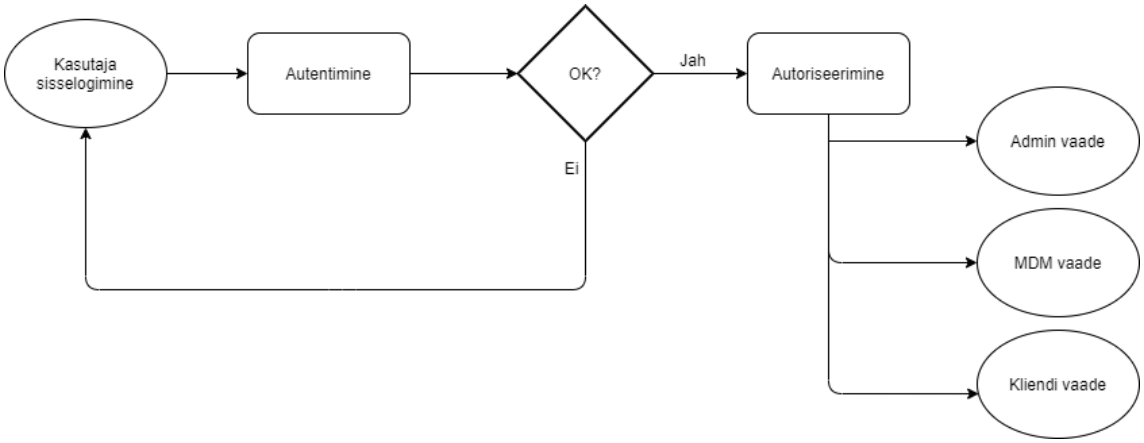
JWT genereerimine toimub serveri poole peal. Selleks oli vaja rakenduse startup-is teha JWT seadistus, ning lisada meetod JWT genereerimiseks ASP.NET poole pealt, mida registreerimisel ja sisselogimisel kutsutakse välja:

```
public static string GenerateJwt(IEnumerable<Claim> claims, string
key, string issuer, string audience,
    DateTime expirationDateTime)
{
    var signingKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
    var signingCredentials = new SigningCredentials(signingKey,
SecurityAlgorithms.HmacSha256);
    var token = new JwtSecurityToken(
        issuer,
        audience,
        claims,
        expires: expirationDateTime,
        signingCredentials: signingCredentials);
    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

Tuleb ka mainida, et ASP.NET MVC rakenduse autentimisel Administraatori ja Põhiandmete haldaja rollidega kasutajatele oli jäetud autentimine läbi seassi küpsiste.

4.4.2 Autoriseerimine

Peale autentimist kasutajale avaneb funktsionaalsus, mis vastab tema rollile. Näiteks MDM rolliga (põhiandmete haldaja) kasutajale tekib võimalus põhiandmete muutmiseks, Admin rolliga kasutajale tekib võimalus ka hallata kasutajate rolle. Tavakasutajal on vaid enda poolsete auto andmete sisestamise võimalus.



Joonis 8. Autentimise ja autoriseerimise skemaatiline representatsioon

Peale autoriseerimist, või ka autentimise käigus võivad tekkida paljud turvaohud, millest räägivad täpsemalt järgnevad peatükid.

4.4.3 Objekti sidumise rünnak

Objekti sidumise rünnakut kasutatakse, kui soovitakse vaates mõningaid andmeid näidata või kui tahetake vaatest mõningaid andmeid saada. Sidumisele võivad olla kaasatud ka seotud objektid, ehk läbi teadaolevate objektide nendega seotud objektide andmete ründamine.

Antud probleemi kõrvaldamiseks PublicApi.DTO.v1 klassid sisaldavad vaid selleks vajalikud andmed, ehk näiteks tehes GET päringu kasutaja kütuse kulu põhitabelisse, võimalik saada vaid kütuse tüübi seotud tabelist tulemuste arvu (palju kütuste tüüpe kokku on), mitte neid tervikuna objektina.

4.4.4 Toore jõu rünnak

Levinud oht, millega veebiarendajad silmitsi seisavad, on parooli ära arvamise rünnak (*brute force attack* ehk toore jõu rünnak). Toore jõu rünnak on katse leida parool, proovides süstemaatiliselt kõiki võimalikke tähtede, numbrite ja sümbolite kombinatsioone, seni kuni avastatakse õige ja toimiv kombinatsioon. Olemas on ka muid paroolide häkkimiseks kasutatavad tehnikaid nagu sõnastiku rünnak, sotsiaalne manipuleerimine (*social engineering*), jne.

Arendatavas .NET Core rakenduses oli kasutusele võetud lukustuse konfiguratsioon:

```
services.Configure<IdentityOptions>(options =>
{
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(30);
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.AllowedForNewUsers = true;
});
```

Seega, kui kasutaja püüab end autentida ja sisestab oma andmed viis korda järjest valesti, lukustatakse kasutaja 30-ks minutiks välja. Seda funktsionaalsust rakendatakse nii vanadele kui uutele kasutajatele. Nii saab olema kindlustanud rakenduse liigsete autentimiskatsete eest.

4.4.5 Saidiülene skriptimine ja saidiülene taotluste võltsimine

Kaks kõige levinumat rünnakut veebisaitide ja veebirakenduste vastu on saidiülene skriptimine, i.k. *Cross Site Scripting* (XSS) ja saidiülene taotluste võltsimine, i.k. *Cross Site Request Forgery* (CSRF).

Saidiülene skriptimine ehk XSS on turvanõrkus, mis võimaldab ründajal paigutada veebilehtedele skripte (kasutades tavaliselt JavaScript-i). Kui kasutajad laadivad mõjutatud lehti, siis käivituvad ründaja skriptid, mis võimaldavad ründajal saada ligi tundlikkele andmetele (nt küpsised, seansitõendid) või isegi olemasolevate andmetega manipuleerimine. Selle rünnaku vältimiseks on andmete kodeerimine, ning kasutaja sisendi valideerimine.

Saidiülene taotluste võltsimine (CSRF) on rünnak, mis sunnib kasutajat veebirakenduses, kus ta parajasti autenditud on, soovimatuid toiminguid tegema [46]. Rakenduses on antud

olukord lahendatud POST ja PUT kontrolleri meetoditel [ValidateAntiForgeryToken] atribuudi abil.

4.4.6 Sama päritolu poliitika ja CORS

Sama päritolu poliitika, i.k. *same-origin policy* on kriitiline turvalisuse mehhanism, mis piirab seda, kuidas ühest päritolust laaditud dokument või skript saab suhelda teise päritolu ressursiga [47], või erinevate domeenide omavahelise suhtluse, et vältida selliseid rünnakuid nagu saidiülene taotluste võltsimine.

Antud rakenduses on tagarakenduseks veebi API ja kasutajaliideseks Vue.js rakendus, seega peab Vue.js omama juurdepääsu API-le. Sel eesmärgil on kasutatud CORS-i, ehk päritoluülest ressursside jagamist, mis toimib nii, et domeenid käsivad brauseril teatud taotlusi lubada. Selleks oli tagarakenduse *startup*-is tehtud järgmine seadistus.

```
services.AddCors(options =>
    {
        options.AddPolicy("CorsAllowAll", builder =>
        {
            builder.AllowAnyHeader();
            builder.AllowAnyMethod();
            builder.AllowAnyOrigin();
        });
    }
);
```

ning app.UseCors("CorsAllowAll");

See tähendab, et nüüd on API-le lubatud igasuguse päritoluga, mistahes päise ja mistahes meetodiga päringud.

4.5 Rakenduse majutamine

Rakenduse majutamiseks oli valitud Raspberry Pi 4 (RPi4) lahendus – ARM protsessoril ühest trükkplaadist koosnev mini server, mis töötab Debian 10.9 operatsioonisüsteemil.

PostgreSQL andmebaas on seadistatud samasse RPi4 peale. Serveripoolne rakendus jookseb Docker (arvutiprogramm, mis teostab operatsioonisüsteemi tasemel virtualiseerimist ehk konteineriseerimist [48]) konteineris. Kliendipoolne lahendus töötab NGINX-i peal, mis on vabavaraline suure jõudlusega HTTP server, mis on ka vähe ressursse kasutav.

Tulemüüriks on kasutusele võetud *Uncomplicated Firewall*, kus on rakenduse kasutamiseks avatud pordid 22 (SSH), 80 (HTTP), 8888 (ASP.NET Core MVC ja RESTful API), 5432 (PostgreSQL andmebaas). Kogu lahenduse ligipääsetavuse internetist tagab RPi4 LAN ühendus, ning ruuteri portide 80, 8888, 5432 edastamine.

Staatilise hosti nimi oli saadud dünaamilisest IP aadressist kasutades No IP servist, mis jälgib pideva IP aadressi muutmise ning teeb lahendust kättesaadavamaks kindlast aadressist.

Rakenduse komplektsuse ja kasutajate arvu kasvu korral saab ümber paigutada rakenduse erikomponendid mõne pilveteenuse platvormile.

5 Tulemused

Lõputöö raames loodud rakendus on täitnud oodatud eesmärke. Minimaalne töötav toode sai valmis, on kasutatav ning vastab püstitatud nõuetele. Funktsionaalsed nõuded, mis olid kirjeldatud kasutusjuhtudena, said enamuses täidetud. Rakendus on praeguseks kasutatav, ning ootab edasiarenduse tegevusi.

Võrreldes turul olevate konkureerivate lahendustega, loodud lahendus lisas paindlikkust ning kasutuse lihtsust tänu sisestavate parameetrite piisavusele nende detailsele kirjeldusele, ning sisestamise võimalusele eri ajavahemiku kohta. Sõidukite võrdluse piirangu probleem sai lahendatuks, et nüüdsest saab lisada võrdlusesse, ning võrrelda niipalju sõiduautosid, kui palju soovi on. Lisaks, muude kulude väli lisab paindlikkust, kuna sinna saab sisestada kõike mida eelnevad kulukohad arvesse ei võta. Sai ka lahendatuks hilisema järele vaatamise probleem, kuna kõik kanded on andmebaasi salvestatud, ning sisse logitud kasutaja saab alati pöörduda enda poolt tehtud arvutuste juurde. Kindlasti saab loodud lahendus abiks olla auto ostjatele, ning auto kulude huvilistele isikutele.

Lisaks loodud rakendusele, rakenduse arendamisel, lõputöö autor täiendas enda teadmisi mitmekihilise arhitektuuri ehitamisel. Samuti sai praktiseeritud Vue.js raamistiku kasutust. Muuhulgas sai palju õpitud andmebaasi seadistamisest ning rakenduse majutamisest serverile, kuna töö raames tuli kokku puutuda NGINX serveri ning Docker konteinerite kasutusega, mis lisas juurde ka DevOps teadmisi.

5.1 Edasiarendus

Edasiarenduse võimalusi võiks jaotada lühiajalisteks, ning pikaajalisteks.

Lühiajaliste plaanide juurde kuuluksid rakenduse:

- Raspberry Pi turvalisuse parendus (Fail2ban[1], Let's Encrypt[2] juurutamisega);
- mitme keelde tõlge;
- mark ja mudeli päringute jõudluse optimeerimine;
- andmete eksportimise võimaluse lisamine, jagamise eesmärgil;
- auto-juurutamise seadistus (pideva integratsiooni näol);
- UI ning tarkvarakomponendi testide kirjutamine;
- vastavusele viimine isikuandmete kaitse määrusele.

Pikaajaliste plaanide juurde kuuluksid:

- mobiili rakenduse arendamine;
- auto reaalsete kulude sisestamise võimaluse funktsionaalsuse lisamine;
- avalikest allikatest sõidukite tehniliste andmete eelsätestamist (n. kütusekulu);
- autode kulu järgi otsimise võimaluse funktsionaalsuse arendamine.

[1] Sissetungi ennetamise tarkvara, mis kaitseb servereid toore jõu rünnakute eest

[2] Vabavaraline sertifitseerimis tarkvara võrguühenduste segmentide krüpteerimiseks transpordikihis

6 Kokkuvõte

Diplomi töö eesmärgiks oli autokulude märkmiku lahenduse loomine, mis lahendaks turul olevate lahenduste puudujääke, ning aitaks sõiduki ostjatele enne ostu otsuse tegemist võrrelda vaatluse all olevate sõidukite kogukulu ajavahemiku ja distantsi kohta.

Töö käigus sai analüüsitud olemasolevaid lahendusi parimate võtete välja selgitamiseks. Said paika pandud funktsionaalsed ja mittefunktsionaalsed nõuded. Said loodud kasutajate vaated, tuginedes kasutusmallide mudelile ja kasutusjuhuste kirjeldusele.

Rakendus sai loodud hajusa süsteemina, mitmekihilise arhitektuurina, mis võimaldab koodi lihtsamat haldamist. Arenduse raames valitud, ning kasutusele võetud tehnoloogiad annavad rakendusele kasvu võimaluse. Kasutades RESTful API-d saab tulevikus ühendada tagarakendust erinevate kasutajaliidestega. Tähelepanu sai pööratud turvalisusele nii tagarakenduse kui Raspberri Pi 4 serveri peal, mis tagab rakenduse ohutut kasutamist.

Töö tulemusena valmis autokulude märkmiku lahenduse minimaalne töötav toode. Veebirakenduse kasutajateks on lõppklient ning administraator ja põhiandmete haldaja, kellele on ette nähtud oma funktsionaalsus. Põhikasutajaks on ettenähtud lõppklient, kes saab nüüdsest kasutada sõidukite omamise kogukulu arvutamiseks märkmik tüüpi lahenduse.

Paika sai pandud projekti lühiajaline ning pikaajaline arendusplaan, mis näeb kliendi vaatest ette rakenduse kättesaadavust mitmes keeles, tulemuste jagamise võimalust, ning mobiilirakenduse arendamist jooksvate kulude mugavaks sisestamiseks.

7 Kasutatud kirjandus

[1] „IHS Markit,“ [Võrgumaterjal]. Available: https://news.ihsmarkit.com/). [Kasutatud 23 03 2021].

[2] Transpordiamet, „Sõidukitega tehtud toimingute statistika,“ [Võrgumaterjal]. Available: https://www.mnt.ee/et/ametist/statistika/soidukid/soidukitega-tehtud-toimingute-statistika#tab-2. [Kasutatud 05 03 2021].

[3] C. F. Institute, „Top Banks in Estonia,“ [Võrgumaterjal]. Available: https://corporatefinanceinstitute.com/resources/careers/companies/top-banks-in-estonia/. [Kasutatud 11 03 2021].

[4] Swedbank, „Auto finantseerimine,“ [Võrgumaterjal]. Available: https://www.swedbank.ee/private/credit/leasing/car?language=EST. [Kasutatud 11 03 21].

[5] SEB, „Autoliisingu kalkulaator,“ [Võrgumaterjal]. Available: https://www.seb.ee/laen-ja-liising/liising/autoliising. [Kasutatud 11 03 2021].

[6] LHV, „Liisingu näidiskuumakse,“ [Võrgumaterjal]. Available: https://www.lhv.ee/et/liising#monthly-payment. [Kasutatud 11 03 2021].

[7] Luminor, „Liisingu kalkulaator,“ [Võrgumaterjal]. Available: https://luminor.ee/era/liising. [Kasutatud 11 03 2021].

[8] C. Pank, „Autolaen,“ [Võrgumaterjal]. Available: https://www.cooppank.ee/autolaen. [Kasutatud 21 03 2021].

[9] F. Mentor, „Find The True Cost of Car Ownership,“ [Võrgumaterjal]. Available: https://financialmentor.com/calculator/car-cost-calculator. [Kasutatud 23 03 2021].

[10] „Калькулятор расходов на автомобиль,“ Autocosts, [Võrgumaterjal]. Available: https://autocosts.info/ru. [Kasutatud 23 03 2021].

[11] C. P. Calculator, „Calculate Automobile Operating Costs,“ [Võrgumaterjal]. Available: https://www.carpaymentcalculator.net/calcs/car-cost.php. [Kasutatud 23 03 2021].

[12] „Vehicle Operating Cost Calculator,“ calculator.me, [Võrgumaterjal]. Available: https://calculator.me/vehicle/operating-cost.php. [Kasutatud 23 03 2021].

[13] „Car Operating Cost Calculator,“ Pigly, [Võrgumaterjal]. Available: https://pigly.com/auto/operating-cost.php. [Kasutatud 23 03 2021].

[14] „Car Cost Calculator,“ Motorparks, [Võrgumaterjal]. Available: https://www.motorparks.co.uk/car-cost-calculator. [Kasutatud 23 03 2021].

[15] „How much does it really cost to own a car?,“ Car Cost Calculator, [Võrgumaterjal]. Available: https://car-cost-calculator.com/. [Kasutatud 23 03 2021].

[16] „Car Cost Calculator - New Car vs. Old,“ The Engineering Toolbox, 2003. [Võrgumaterjal]. Available: https://www.engineeringtoolbox.com/car-costs-old-new-d_339.html.

[17] „Car Cost Comparison Calculator,“ Clearpoint, [Võrgumaterjal]. Available: https://www.clearpoint.org/tools/car-cost-comparison-calculator/. [Kasutatud 23 03 2021].

[18] „Car Buying Calculator to Calculate Car Buying Comparisons,“ Free Online Calculator Use, [Võrgumaterjal]. Available: https://www.free-online-calculator-use.com/car-buying-calculator.html. [Kasutatud 23 03 2021].

[19] „Car Loan Calculator,“ Cars.com, [Võrgumaterjal]. Available: https://www.cars.com/car-loan-calculator/. [Kasutatud 23 03 2021].

[20] „Auto Loan Calculator,“ Caclulator.net, [Võrgumaterjal]. Available: https://www.calculator.net/auto-loan-calculator.html. [Kasutatud 23 03 2021].

[21] „Driving Cost Calculator,“ CAA, [Võrgumaterjal]. Available: https://carcosts.caa.ca/. [Kasutatud 23 03 2021].

[22] „Cost of Car Ownership,“ Edmunds, [Võrgumaterjal]. Available: https://www.edmunds.com/tco.html. [Kasutatud 23 03 2021].

[23] C. Crampton, „Literature review: A Comparative Analysis of LAMP and Microsoft.NET,“ [Võrgumaterjal]. Available: https://www.cs.ru.ac.za/research/g01C1073/images/litreview_final.pdf. [Kasutatud 01 04 2021].

[24] A. M. Z. Hanin M. Abdullah, „Frontend and Backend Web Technologies in Social Networking Sites: Facebook as an Example,“ *3rd International Conference on Advanced Computer Science Applications and Technologies,* nr 10.1109/ACSAT.2014.22, pp. 85-89, 2014.

[25] „Choosing the Right Back-end Technology for your Business,“ Maruti Techlabs, [Võrgumaterjal]. Available: https://marutitech.com/back-end-technology/. [Kasutatud 01 04 2021].

[26] A. Z. S. F. Tamir Dresher, Hands-On Full-Stack Web Development with ASP.NET Core, Packt Publishing Ltd, 2018.

[27] „Backend Web Development Technologies in 2021-2022,“ Cyber Craft, [Võrgumaterjal]. Available: https://cybercraftinc.com/blog/backend-web-development-technologies-in-2021-2022. [Kasutatud 01 04 2021].

[28] „Spring Framework - Overview,“ [Võrgumaterjal]. Available: https://www.tutorialspoint.com/spring/spring_overview.htm. [Kasutatud 15 03 2021].

[29] A. Khirale, „Comparison Between Angular Vs React Vs Vue,“ Angular Minds, 08 09 2020. [Võrgumaterjal]. Available: https://www.angularminds.com/blog/article/comparison-between-angular-vs-react-vs-vue.html. [Kasutatud 01 04 2021].

[30] H. Darwen, „An Introduction to Relational Database Theory,“ 2014. [Võrgumaterjal]. Available: https://bookboon.com/premium/books/an-introduction-to-relational-database-theory. [Kasutatud 25 03 2021].

[31] „DB-Engines Ranking,“ 03 2021. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking. [Kasutatud 25 03 2021].

[32] „Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch and others,“ Altexsoft, [Võrgumaterjal].

Available: https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/. [Kasutatud 25 03 2021].

[33] M. D. ostezer, „SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems,“ The Digital Ocean Community, 19 03 2019. [Võrgumaterjal]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems. [Kasutatud 25 03 2021].

[34] A. H. D. N. J. R.-J. A. S. D. R. a. B. W. Kellyn Gorman, „Introducing Microsoft SQL Server 2019,“ 2019. [Võrgumaterjal]. Available: https://info.microsoft.com/ww-landing-introducing-sql-server-2019-content.html. [Kasutatud 25 03 2021].

[35] „SQL Conformance,“ PostgreSQL, [Võrgumaterjal]. Available: https://www.postgresql.org/docs/13/features.html.

[36] F. S. Martin Lindblom, „Performance analysis and improvement of PostgreSQL,“ *the Department of Computer Science, Lund University,* nr ISSN 1650-2884, p. 45, 2015.

[37] „Npgsql - .NET Access to PostgreSQL,“ [Võrgumaterjal]. Available: https://www.npgsql.org/. [Kasutatud 15 03 2021].

[38] K. Viik, „Mittefunktsionaalsete nõuete määratlemine turvalise tarkvaraarenduse hankimiseks Eesti avalikus sektoris,“ 2017. [Võrgumaterjal]. Available: http://webcache.googleusercontent.com/search?q=cache:AW-Q0Y1-38YJ:www.cs.tlu.ee/teemaderegister/get_file.php%3Fid%3D605+&cd=1&hl=en&ct=clnk&gl=ee. [Kasutatud 31 03 2021].

[39] C. Crampton, „A Comparative Analysis of the LAMP and Microsoft.NET Frameworks,“ 2005.

[40] „Isolating your BLL from your DAL and unit testing it,“ 30 08 2014. [Võrgumaterjal]. Available: https://www.gaui.is/isolating-your-bll-from-your-dal-and-unit-testing-it/.

[41] „Design the infrastructure persistence layer,“ Microsoft, 10 08 2018. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design. [Kasutatud 02 03 2012].

[42] A. Käver, „Building Distributed Systems course material,“ [Võrgumaterjal].

[43] R. S. Christoph Nienaber, „ASP.NET Core web API documentation with Swagger / OpenAPI,“ Microsoft, 29 10 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-5.0. [Kasutatud 04 04 2021].

[44] „Ajax,“ [Võrgumaterjal]. Available: https://et.wikipedia.org/wiki/Ajax. [Kasutatud 21 03 2021].

[45] S. Ghate, „Using Session Cookies Vs. JWT for Authentication,“ Hackernoon, 08 06 2020. [Võrgumaterjal]. Available: https://hackernoon.com/using-session-cookies-vs-jwt-for-authentication-sd2v3vci. [Kasutatud 24 03 2021].

[46] KirstenS, „Cross Site Request Forgery (CSRF),“ The OWASP Foundation, [Võrgumaterjal]. Available: https://owasp.org/www-community/attacks/csrf. [Kasutatud 11 04 2021].

[47] „Same-origin policy,“ MDN Web Docs, [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. [Kasutatud 07 03 2021].

[48] V. Ratan, „Docker: A Favourite in the DevOps World,“ OpenSourceForU, 08 02 2017. [Võrgumaterjal]. Available: http://www.opensourceforu.com/2017/02/docker-favourite-devops-world/. [Kasutatud 18 04 2021].

[49] N. A. Philip Reed, „What Is the Total Cost of Owning a Car?,“ Nerdwallet, 28 06 2019. [Võrgumaterjal]. Available: https://www.nerdwallet.com/article/loans/auto-loans/total-cost-owning-car. [Kasutatud 23 03 2021].

[50] „Cost Calculator,“ Enterprise Car Club, [Võrgumaterjal]. Available: https://www.enterprisecarclub.co.uk/gb/en/about/calculator.html. [Kasutatud 23 03 2021].

[51] „Car running cost calculator,“ RACV , [Võrgumaterjal]. Available: https://www.racv.com.au/on-the-road/buying-a-car/car-running-costs.html. [Kasutatud 23 03 2021].

[52] „Alternative Fuels Data Center,“ U.S. department of energy, [Võrgumaterjal]. Available: https://afdc.energy.gov/calc/. [Kasutatud 23 03 2021].

[53] „Vehicle Cost Calculator,“ Efficiency Maine, [Võrgumaterjal]. Available: https://www.efficiencymaine.com/evehicles/vehicle-cost-calculator/. [Kasutatud 23 03 2021].

[54] „MPG and Cost Calculator and Tracker,“ Spritmonitor, [Võrgumaterjal]. Available: https://www.spritmonitor.de/. [Kasutatud 23 03 2021].

[55] D. R. M. F. E. H. R. M. R. S. Martin Fowler, Patterns of Enterprise Application Architecture, Addison Wesley, 2002.

[56] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, First Edition, Pearson, 2017.

[57] Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley Professional, 2003.

[58] T. Dykstra, „Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application,“ Microsoft, 30 07 2013. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application. [Kasutatud 02 03 2021].

[59] S. Shanu, „ASP.NET MVC 5 Security And Creating User Role,“ 06 02 2016. [Võrgumaterjal]. Available: https://www.c-sharpcorner.com/UploadFile/asmabegam/Asp-Net-mvc-5-security-and-creating-user-role/. [Kasutatud 17 03 2021].

[60] „[ITS] IT terministandardi sõnastik,“ Eesti Keele Instituut, [Võrgumaterjal]. Available: http://www.eki.ee/dict/its/. [Kasutatud 2021].

[61] „Relatsiooniliste andmemudelite koostamise juhend,“ Riigi Infosüsteemi amet, 2015. [Võrgumaterjal]. Available: https://www.ria.ee/sites/default/files/content-editors/publikatsioonid/relatsiooniliste_andmemudelite_koostamise_juhend_ver._1.0.pdf. [Kasutatud 25 03 2021].

[62] P. Raspel, „Andmebaaside alused kursuse materjal,“ [Võrgumaterjal]. Available: https://enos.itcollege.ee/~priit/1.%20Andmebaasid/1.%20Loengumaterjalid/. [Kasutatud 25 03 2021].

[63] „PopularitY of Programming Language,“ PYPL , [Võrgumaterjal]. Available: https://pypl.github.io/PYPL.html. [Kasutatud 01 04 2021].

[64] „What is the difference between DAO and Repository patterns?,“ stackowerflow, [Võrgumaterjal]. Available: https://stackoverflow.com/questions/8550124/what-is-the-difference-between-dao-and-repository-patterns. [Kasutatud 24 03 2020].

[65] „Repository vs DAO,“ Best practices, 01 11 2012. [Võrgumaterjal]. Available: https://blog.sapiensworks.com/post/2012/11/01/Repository-vs-DAO.aspx. [Kasutatud 24 03 2020].

[66] „ASP.NET API Versioning,“ Microsoft, [Võrgumaterjal]. Available: https://github.com/microsoft/aspnet-api-versioning. [Kasutatud 11 04 2021].

[67] „Django introduction,“ MDN Web Docs, [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction.

[68] „ADO.NET Framework Data Providers,“ [Võrgumaterjal]. Available: https://www.javatpoint.com/ado-net-data-providers. [Kasutatud 15 03 2021].

[69] N. H. T. S. Administration, „Vehicle API,“ United States Department of Transportation, [Võrgumaterjal]. Available: https://vpic.nhtsa.dot.gov/api/. [Kasutatud 03 04 2021].

[70] R. Anderson, „XSRF/CSRF Prevention in ASP.NET MVC and Web Pages,“ Microsoft, 14 03 2013. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/xsrfcsrf-prevention-in-aspnet-mvc-and-web-pages. [Kasutatud 21 03 2021].

[71] „What is a "State Management Pattern"?,“ Vuex, [Võrgumaterjal]. Available: https://vuex.vuejs.org/#what-is-a-state-management-pattern. [Kasutatud 21 03 2021].

[72] „Programmeerimise alused õpilastele,“ Tartu Ülikool arvutiteaduse instituut, [Võrgumaterjal]. Available: https://courses.cs.ut.ee/2016/eprogalkool/fall/Main/ErinevadProgKeeled. [Kasutatud 07 03 2021].

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

Mina, Andrei Tomba

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Autokulude märkmik rakenduse arendamine“, mille juhendaja on Nadežda Furs
   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.

17.05.2021

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 - Alternatiivsete lahenduste võrdlustabel

Tabel 3. Konkureerivate lahenduste omaduste võrdlus

	Parameeter	[9]	[11]	[49]	[13]	[12]	[14]	[15]	[17]	[16]	[18]	[50]	[10]	[19]	[20]	[21]	[51]	[22]	[52]	[53]	[54]
Sisend	Sõiduki hind	+	+		+	+		+	+	+	+		+	+	+				+		
	Periood	+	+		+	+	+		+	+	+			+	+				+		
	Sissemakse	+	+		+	+			+		+			+	+						
	Jääkväärtus										+										
	Intressimäär	+	+		+	+			+		+			+	+						
	Laenusumma																				
	Muud kulud	+	+	+	+	+	+		+	+	+	+	+		+						+
	Kindlustus	+	+	+	+	+	+	+	+	+	+	+	+								+
	Läbisõit	+	+		+	+	+	+	+	+	+					+			+	+	+
	Kütusekulu	+	+		+	+	+	+			+		+							+	+
	Kütuse hind	+	+		+	+	+	+	+				+			+			+		+
	Kütuse tüüp																			+	+
	Kütuse kulu (rahas)			+						+		+									
	Sõiduki aasta	+									+										
	Hooldus ja remont	+	+	+	+	+		+	+	+	+	+	+								+
	Sõiduki andmed															+	+	+	+		+
	Linnasõidu suhe															+				+	+
	Liisingumakse			+				+		+		+	+								
	Odavnevus							+		+											
Väljund	Muud kulud				+											+					
	Kindlustus		+		+	+					+					+		+			
	Hooldus ja remont		+			+										+		+			
	Odavnevus	+	+		+	+		+			+					+		+			
	Lepingutasu																				
	Liisingu kuumakse				+	+					+				+						
	Maksegraafik														+			+			
	Võrdlus (tk)	2							2	2	2								5+		5+
	Kütuse kulu (rahas)		+		+		+	+	+		+		+					+		+	+
	Kogu omamise kulu	+	+		+	+		+	+	+	+		+					+			
	Kulu perioodi kohta	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+		+		+
	Kulu distantsi kohta	+	+		+	+					+		+						+		+
	Rühm	1												2		3			4		5

Lisa 3 – *Unified Modelling Language*-i kasutusjuhud

Tabel 4. UML-i kasutusjuhus 1 – Registreerimise võimalus

Nimi	**UC1 Registreerimise võimalus**
Eesmärk	Registreerida kasutaja andmed, et saaks rakendusele ligi eri ligipääsuvõimalustega.
Tulemus	Kasutaja registreeritud.
Eeltingimus	Kasutajal on e-posti aadress, hüüdnimi, parool.
Aktorid	Klient, andmete administraator, administraator
Peastsenaarium	1. Kasutaja soovib registreerida 2. Kasutaja suundub registreerimis lehele, vajutades nupule Registreeri 3. Kasutajal on võimalik sisestada e-posti aadress, hüüdnimi, parool 4. Kasutaja kinnitab parooli teises väljas 5. Kasutaja vajutab nuppu Registreeri 6. Kasutaja andmed on salvestatud andmebaasi.
Alternatiivne stsenaarium	Kasutaja andmed on sisestatud andmebaasi läbi SQLi. Kasutaja andmed on süstitud sisse rakenduse juurutamise käigus.
Järeltingimus	Kasutajal tekib sisselogimise võimalus

Tabel 5. UML-i kasutusjuhus 2 – Sisselogimise võimalus

Nimi	**UC2 Sisselogimise võimalus**
Eesmärk	Logida sisse, et pääseda endale vajalike vaadete juurde.
Tulemus	Kasutaja sisse logitud, pääs nõutud vaadete juurde on olemas.
Eeltingimus	Klient on registreeritud UC1, ning autoriseeritud
Aktorid	Klient, andmete administraator, administraator
Peastsenaarium	1. Kasutaja soovib sisestada auto võrdlemiseks 2. Kasutaja autendib ennast 3. Kasutaja saab ligi nõutud vaadete juurde.
Alternatiivne stsenaarium	Klient saab parooli meeldetuletuse e-posti kaudu.
Järeltingimus	Sisselogimis sessiooni aegumine.

Tabel 6. UML-i kasutusjuhus 3 – Väljalogimise võimalus

Nimi	**UC3 Väljalogimise võimalus**
Eesmärk	Välja logida rakendusest.
Tulemus	Kasutaja on väljalogitud.
Eeltingimus	Kasutaja on sisse logitud UC2.
Aktorid	Klient, andmete administraator, administraator
Peastsenaarium	1. Kasutaja soovib välja logida 2. Kasutaja vajutab nuppu Logi välja 3. Kasutaja on rakendusest väljalogitud.
Alternatiivne stsenaarium	Sessioon on aegunud.
Järeltingimus	Ligipääs sisse logituna nähtavatele vaadetele on puudu. Kasutajal on hiljem võimalus uuesti sisse logida.

Tabel 7. UML-i kasutusjuhus 4 – Parooli muutmise võimalus

Nimi	**UC4 Parooli muutmise võimalus**
Eesmärk	Vahetada olemasolev parool uue vastu.
Tulemus	Parool on vahetatud, kasutaja saab uuesti sisse logides kasutada uut parooli.
Eeltingimus	Kasutaja on sisse logitud UC2
Aktorid	Andmete administraator, administraator
Peastsenaarium	1. Kasutaja soovib vahetada parooli 2. Kasutaja vajutab enda kasutajanime peale 3. Kasutaja jõuab kasutaja andmete juurde 4. Kasutaja vajutab nuppu Parool 5. Kasutaja sisestab olemasoleva parooli 6. Kasutaja sisestab uue parooli 7. Kasutaja kinnitab uue parooli 8. Kasutaja vajutab nuppu Uuenda Parool 9. Andmed on andmebaasi salvestatud
Alternatiivne stsenaarium	Administraator muudab kasutaja parooli kasutajate haldus süsteemis UC6
Järeltingimus	Kasutajal on võimalus sisse logida uue parooliga.

Tabel 8. UML-i kasutusjuhus 5 – Kasutajate õiguste haldamine

Nimi	**UC5 Kasutajate õiguste haldamine**
Eesmärk	Võimalus hallata kasutajate ligipääsuõigusi
Tulemus	Lubada/muuta/kustutada ligipääsu õigusi
Eeltingimus	Kasutaja on logitud sisse Administraatorina
Aktorid	Administraator
Peastsenaarium	1. Administraator soovib hallata teiste kasutajate õigusi 2. Administraator logib sisse administraatori õigustega 3. Valib kasutaja rollide halduse menüü 4. Ilmub olemasolevate rollide list 5. Administraatoril on võimalik antud vaates lisada uue rolli 6. Vajutades nupule kasutajate nimekiri avaneb kasutajate list 7. Valitud kasutaja real võimalik vajutada Kasutaja õigused nupule ja avaneb kasutaja õiguste halduse vaade 8. Administraatoril on võimalus lisada, eemaldada ligipääsuõigusi kasutajalt 9. Peale Salvesta nupu vajutamist kasutaja saab uusi õigusi
Alternatiivne stsenaarium	Kasutaja õigused muudetud administraatori poolt otse andmebaasis, SQLi abil.
Järeltingimus	Kasutaja kellele muudeti õigusi on vajalik korraks välja logida ja sisse logida uuesti.

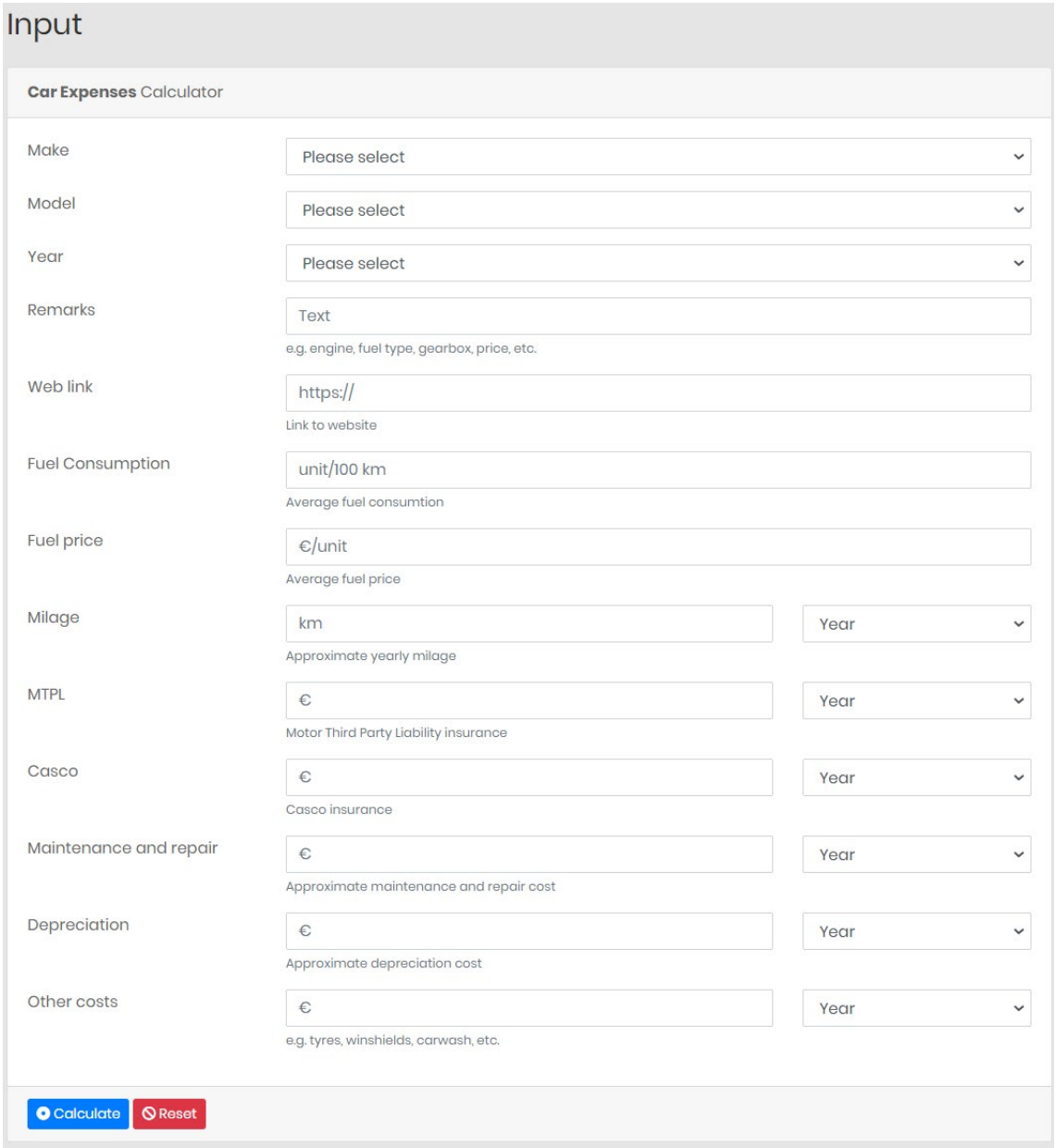
Tabel 9. UML-i kasutusjuhus 7 – Auto andmete sorteerimise võimalus

Nimi	**UC7 Auto andmete sorteerimise võimalus**
Eesmärk	Sorteerida autode andmed vastavalt nõutud väljale
Tulemus	Autode andmed sorteeritud
Eeltingimus	Klient on sisse logitud UC2, ning on sisestanud vähemalt kahe auto andmed UC6
Aktorid	Klient
Peastsenaarium	1. Klient soovib sorteerida sisestatud autode andmed 2. Näeb listi sisestatud autodest 3. Kliendil on võimalik sorteerida andmed iga veeru järgi
Alternatiivne stsenaarium	puudub
Järeltingimus	puudub

Tabel 10. UML-i kasutusjuhus 8 – Põhiandmete halduse võimalus

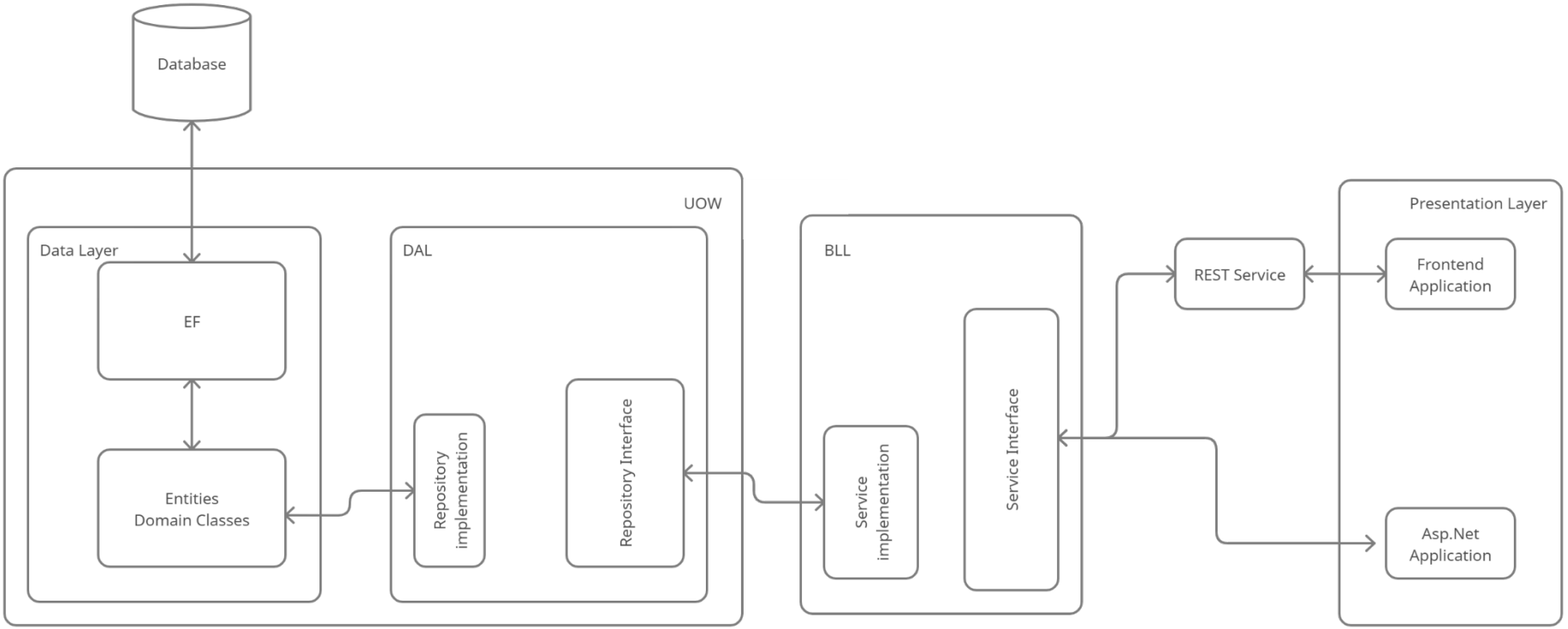
Nimi	**UC8 Põhiandmete halduse võimalus**
Eesmärk	Lisada, muuta, kustutada põhiandmeid
Tulemus	Põhiandmed modifitseeritud vastavalt püstitatud nõuetele
Eeltingimus	Põhiandmete administraatori õiguste olemasolu, sisse logitud UC2
Aktorid	Andmete administraator, administraator
Peastsenaarium	1. Andmete administraator soovib lisada, muuta, kustutada põhiandmeid(kulude tüüp, valuuta, kütuse kulu tüüp, kütuse tüüp, käigukast, auto mark, auto mudel, ajavahemik) 2. Põhiandmete menüüst on võimalik selekteerida konkreetse põhiandme menüü 3. Avaneb alammenüü koos listiga olemasolevatest kirjetest 4. Iga kirje taga on olemas muutmise, detailide vaatamise ja kustutamise võimalus 5. Vajutades alammenüüle avaneb andmete halduse vaade 6. Vastavalt püstitatud nõuetele andmed on kas sisestatud/uuendatud/kustutatud
Alternatiivne stsenaarium	Andmete sisestamine/muutmine/kustutamine administraatori poolt otse andmebaasist, kasutades SQLi
Järeltingimus	Muudatused on salvestatud andmebaasi

Lisa 4 – Sisend parameetrite klientrakenduse kasutajaliides



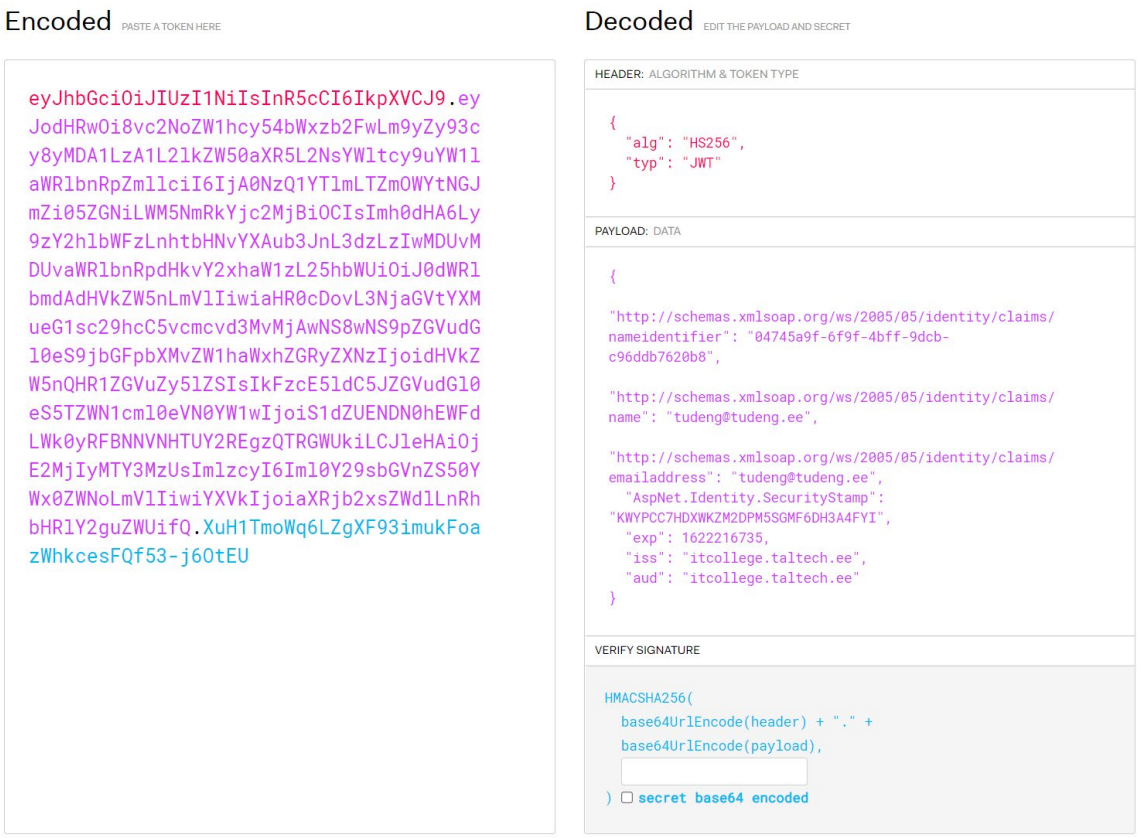
Joonis 9. Kasutajaliidese sisendparameetrite vaade

Lisa 5 – Rakenduse arhitektuuri skemaatiline representatsioon



Joonis 10. Rakenduse arhitektuuri visioon

Lisa 6 – *JSON Web Token* kodeeritud ja dekodeeritud kujul



Joonis 11. JWT kodeeritud ja dekodeeritud kujul