TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Siim Melles 185348IADB

LOGISTIKAETTEVÕTTE HOOLDUSLOGI VEEBISÜSTEEMI ARENDAMINE

Bakalaureusetöö

Juhendaja: Kristiina Hakk

PhD

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud 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: Siim Melles

13.05.2021

Annotatsioon

Logistikaettevõttes kasutusel olev paberipõhine hoolduslogide süsteem on tekitanud mitmeid probleeme andmete pidevuses ning informatsiooni kättesaadavuses. Selleks, et neid probleeme vältida, otsustati asendada paberlehtedel põhinev süsteem digitaalse lahendusega. Bakalaureusetöö eesmärk on luua hoolduslogide haldamist ja vaatamist võimaldav veebirakendus, mis omaks sama funktsionaalsust kui eelnev süsteem, kuid oleks kõikjal kättesaadav.

Töö esimeses peatükis kirjeldatakse lähemalt ettevõtte tegevusvaldkonda ning digitaalse lahenduse vajadust ja olulisust nende äris. Sellele järgnevas analüüsi osas kaardistatakse rakenduse funktsionaalseid nõuded, mida on kirjeldatud kasutajalugude kaudu. Seejärel analüüsitakse ülesande lahendamiseks sobilikke tehnoloogiaid ning valitakse välja, mida lahenduse loomisel kasutama hakatakse.

Kolmandas peatükis selgitatakse rakenduse arhitektuuri. Autor seletab lahti andmebaasimudeli ja kuidas see väljendub rakenduse äriloogikas ning kirjeldab loodava ees- ja tagarakenduse ülesehitust.

Töö lõpus on analüüs tehtud töö tulemustest ning autori pakutud lahendused ja variandid töö edasiarenduseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 32 leheküljel, nelja peatükki, 13 joonist, kahte tabelit.

Abstract

Development of Maintenance Log System for Logistics Company

The logistics company at hand is currently using a paper-based maintenance log system which has been causing many problems, for example data loss due to lost papers or poor reachability due to there being a single source of information. The decision was made to replace the old system with a new digital solution to solve these problems. The aim of this bachelor's thesis is to create a web application, that would maintain the same functionality as the paper system but would be reachable from anywhere.

The first chapter describes the logistics company and their specific field in more detail and explains why the system is necessary for their business needs. In the following chapter the author describes the distinctive functionality that the web application must possess. The functional requirements are described through user stories. Following that, the author analyses which technologies could be considered to develop the application and chooses the technology stack appropriate for the task.

The third chapter describes the architecture of the application. The author explains database model and how it translates to the business logic of the application, as well as the structure of both the back-end and front-end applications.

In the final chapter of the thesis, the author analyses the application that was completed as a result of this thesis and points out possibilities for further development.

The thesis is in Estonian and contains 32 pages of text, four chapters, 13 figures, 2 tables.

Lühendid ja mõisted

ACID	*Atomicity, Consistency, Isolation, Durability*, andmebaasiomaduste kogum
API	*Application Programming Interface*, rakendusliides
BLL	*Business Logic Layer*, äriloogika kiht mitmekihilise arhitektuuri mustris
CRUD	*Create, Read, Update and Delete*, neli funktsiooni, mis on andmebaasiga rakenduse loomiseks vajalikud – loomine, lugemine, uuendamine ja kustutamine
DAL	*Data Access Layer*, andmetele ligipääsu kiht mitmekihilise arhitektuuri mustris
DBMS	*Database Management System*, andmebaasi haldussüsteem
DTO	*Data Transfer Object*, andme-edastusobjekt
GUID	*Globally-Unique Identifier*, globaalselt unikaalne identifikaator
HTML	*HyperText Markup Language*, hüperteksti märgistuskeel
HTTP	*Hypertext Transfer Protocol* – protokoll teabe edastamiseks arvutivõrkudes
ISO	*International Organization for Standardization*, Rahvusvaheline Standardimisorganisatsioon
JSON	*JavaScript Object Notation*, andmevahetusvorming
JWT	*JSON Web Token*, autentimistooken
LINQ	*Language-Integrated Query*, Microsofti loodud programmeerimiskeeltesse integreeritud andmeallikatest päringute tegemise keel
MVC	*Model-View-Controller*, mudel-vaade-kontroller – rakendustes kasutatav arhitektuur
NoSQL	*Not only SQL*, mitterelatsiooniline andmebaas
REST	*Representational State Transfer*, tarkvara arhitektuuri laad, mis seab veebirakenduse loomisele kindlad piirid

SQL	*Structured Query Language*
UOW	*Unit Of Work*, tarkvara disaini muster
URL	*Uniform Resource Locator*, internetiaadress

Sisukord

1 Sissejuhatus 11
2 Hetkeolukorra kirjeldus 12
2.1 Taust 12
2.2 Probleem 12
2.3 Eesmärk 13
3 Rakenduse analüüs 14
3.1 Rakenduse nõuded 14
3.2 Rakenduse tehnoloogia valik 15
3.2.1 Tagarakenduse programmeerimiskeele valik 16
3.2.2 Tagarakenduse raamistiku valik 18
3.2.3 Eesrakenduse programmeerimiskeele valik 19
3.2.4 Eesrakenduse raamistiku valik 20
3.3 Rakenduse andmebaas 23
3.3.1 Andmebaasi struktuur 23
3.3.2 Andmebaasi haldussüsteemi valik 24
4 Logistikaettevõtte hoolduslogi rakenduse loomine 26
4.1 Rakenduse andmebaas 26
4.1.1 Primaarvõtme valik 26
4.1.2 Andmebaasimudeli kirjeldus 27
4.2 Tagarakenduse arendus 29
4.2.1 Tagarakenduse arhitektuur 29
4.2.2 REST API 31
4.2.3 API kontrollerite turvalisus 32
4.2.4 Administraatoriliides 33
4.3 Eesrakenduse arendus 33
4.3.1 Eesrakenduse projekti struktuur 33
4.3.2 Eesrakenduse kasutajaliidese struktuur 35
4.3.3 Kasutajaliides ettevõtte töötajatele 36
4.3.4 Kasutajaliides ettevõtte klientidele 39
5 Tulemused 41

6 Kokkuvõte 42
Kasutatud allikad 43
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 47
Lisa 2 – Nõuete kirjeldus epikute ja kasutajalugude kaudu 48

Jooniste loetelu

Joonis 1. Programmeerimiskeelte trend platvormil Stack Overflow [13]. 17
Joonis 2. JavaScripti raamistike trend platvormil Stack Overflow [26]. 22
Joonis 3. Andmebaasi olemi-suhte diagramm. 27
Joonis 4. Tõlkeobjekti tõlke kaasamine LINQ avaldises hoidla päringus. 31
Joonis 5. Ettevõtte andmete kuvamise lõpp-punkti näide. 32
Joonis 6. Eesrakenduse koodibaasi failistruktuur. 34
Joonis 7. Veebirakenduse esi- ja sisselogimislehed. 35
Joonis 8. Sisselogitud kasutaja esileht ja navigatsiooniriba. 36
Joonis 9. Laohoonete ja üksuste nimekirja, laoüksuse hoolduslogi ning sissekande lisamise lehed. 37
Joonis 10. Hoolduslogi sissekande detaile sisaldav hüpikaken. 38
Joonis 11. Hoolduslogide nimekirja ning hoolduslogi sissekannete lehed. 38
Joonis 12. Laotehnika hoolduslogi sissekande loomise leht ning sissekande detailide hüpikaken. 39
Joonis 13. Külaliskonto esileht ning navigatsiooniriba. 40

Tabelite loetelu

Tabel 1. Tagarakenduse programmeerimiskeelte võrdlus. 18
Tabel 2. Eesrakenduse raamistike võrdlus. 22

1 Sissejuhatus

Logistikaettevõtte tänapäevaste digitaalsete süsteemide kõrval kasutatakse endiselt ladude ja tehnika hooldustööde jälgimiseks paberkandjal olevaid logiraamatuid, mida laooperaatorid peale igat hooldust täidavad. Selle süsteemiga kaasneb mitmeid puudujääke ja kitsaskohti, mida oleks süsteemi digitaliseerimisega võimalik lahendada.

Lõputöö eesmärk on luua logistikaettevõttele hoolduslogide digitaalne süsteem. Töös kirjeldatakse esmalt klientettevõtte tausta ja tegevusala, kuidas tekkis idee luua uus digitaalne süsteem ning miks on töö tulemusena loodav rakendus üldse vajalik. Lühidalt selgitatakse, kuidas näeb välja praegune süsteem ning tuuakse välja, milliseid probleeme hetkel eksisteeriva süsteemi puudujääkide tõttu esineb. Autor kirjeldab eesmärki ehk millised on üldised nõuded rakendusele ning mida uue lahendusega saavutada tahetakse.

Järgnevas analüüsi peatükis kirjeldab autor täpsemalt rakenduse nõudeid kasutajalugude vaatepunktist ning toob välja, millised on erinevad kasutajagrupid ja nende õigused süsteemis. Seejärel analüüsib autor, milliste tehnoloogiate valik oleks optimaalne antud rakenduse loomiseks ning valib välja sobiliku programmeerimiskeele ning raamistiku nii taga- kui ka eesrakenduse loomiseks. Samuti kirjeldatakse, millise struktuuriga andmebaas rakenduse andmete talletamiseks valida ning millise andmebaasihaldussüsteemi abil seda tehakse.

Rakenduse loomise peatükis kirjeldatakse esmalt andmebaasi – milline on andmebaasimudel, millist primaarvõtit andmebaasis kasutatakse ning mis põhjustel seda tehakse. Seejärel kirjeldab autor tagarakenduse arhitektuuri ning kuidas tagarakendus toimib. Viimasena kirjeldatakse peatükis eesrakenduse loomist – millise struktuuriga on eesrakenduse projekt ning kuidas on üles ehitatud eesrakenduse kasutajaliides.

Tulemuste peatükis analüüsitakse lõputöö käigus loodud süsteemi. Tuuakse välja, mis lõputöö raames loodi ning kuidas rakenduse arendusega edasi minna.

2 Hetkeolukorra kirjeldus

Logistikaettevõtte, millele hoolduslogide veebirakenduse süsteem luuakse, on Katoen Natie Eesti AS. Tegu on ettevõttega, mis tegeleb peamiselt toiduainetele (kakaooad, kohvioad ja muud kakaotooted) kui ka muudele kaupadele ladustamisteenuse osutamisega. Ettevõtte klientuuri kuuluvad paljud šokolaadi ja maiustuste tootjad – ettevõtte bränd on logistikas globaalselt tuntud oma kvaliteedi ja kõrgete standardite poolest [1]. Sellest tulenevalt on ettevõttel oma klientide ees vastutus tagada kauba korralik ladustamine, mistõttu on oluline omada ülevaadet ladude ning tehnika korrashoiust. Samuti on ettevõte sertifitseeritud ISO 9001:2015 sertifikaadiga, mis tähendab kindlat kvaliteedistandardit, mille ettevõte tagama peab [2].

2.1 Taust

Idee luua digitaalne süsteem hoolduslogide haldamiseks tekkis autoril 2020. aasta suvel, kui ettevõttes hooajalisel tööl olles märkas ta, et ettevõtte tänapäevaste digitaalsete süsteemide kõrval kasutatakse ladude ja tehnika hooldustööde jälgimiseks paberkandjal olevaid logiraamatuid, mida laooperaatorid peale igat hooldust täidavad. Selle süsteemiga kaasnes mitmeid puudujääke, mis olid põhjustanud probleeme ettevõtte tegevuses ning kvaliteedi ja andmete järjepidevuses. Arutelus ettevõtte direktoriga pakkus autor, et loob ettevõttele digitaalse logiraamatute süsteemi, mis omaks baastasandil sama funktsionaalsust, kuid ka lisaks juurde mõningaid abistavaid funktsioone vajalike andmete kiiremaks leidmiseks, ning mis oleks kasutuskogemuse poolest sama mugav kui paberkandjal logiraamatu täitmine.

2.2 Probleem

Süsteemi digitaliseerimist ajendasid mitmed paberkandjal esinevad probleemid. Kuna iga laoüksuse logiraamatut hoitakse laos seinal, siis on kiire ülevaate saamine tihti keeruline – tuleb lattu minna ning seal logiraamatut vaadata. Paberkandjal logiraamatud kipuvad tihti ka määrduma. See võib klientidele hooldusandmete esitamisel jätta ebaprofessionaalse mulje. On ka juhtumeid, kus logiraamatu lehed olid kadunud ning info

oli sellest tulenevalt puudulik. Viimase probleemina on logiraamatuid kohati raske lugeda, sest laooperaatorite käekirja loetavus varieerub.

2.3 Eesmärk

Eesmärgiks on luua süsteem, mis ennekõike eemaldaks eelmainitud probleemid, kuid ka looks ettevõttele lisaväärtust. Rakenduse loomisel on prioriteet kasutuskogemus. Rakenduse eesmärk on hõlbustada hoolduslogide täitmist ning jälgimist, mis tähendab, et rakenduse visuaalne ülesehitus peab olema kergesti jälgitav ning üheselt mõistetav. Rakenduse kasutajaskonna nutioskused varieeruvad, mistõttu on oluline luua kerge menüüsüsteem ning lihtsalt täidetavad ankeedid. Võrdväärselt lihtne peab olema laoüksuse ülevaate kuvamine, et näha, kus on hooldustööd tehtud ning kus veel tegemata. Parema läbipaistvuse ning kiirema suhtluse tagamiseks on võimalik anda klientidele vaatamisõigustega ligipääs rakendusele, et neil oleks võimalik näha, kuidas nende kaupu ladustavaid laoruume hooldatud on. Samuti oleks süsteemi lisatud laotehnika hoolduse logiraamatud, millel on praegu samade probleemidega raskusi. Laotehnika logi andmete täielikkus on oluline, et tagada töötajate ning töökeskkonna kvaliteet ja ohutus.

Selleks, et kogu ettevõtte süsteem konfigureerida, on rakendusel olemas ka administraatoriliides, kus on võimalik hallata ettevõtte andmeid – laoüksuseid ja neis ladustatavaid kaupu, hooldustööde ja kaupade liike, hoolduslogisid ja nende sissekandeid ning ettevõtte töötajate kasutajaid ning nende rolle ja õiguseid.

3 Rakenduse analüüs

Rakenduse analüüsis kirjeldatakse rakenduse nõudeid ning valitakse võrdluse abil välja nii taga- kui ka eesrakenduse programmeerimiskeel ja raamistik. Samuti analüüsitakse, kas valida relatsiooniline või mitterelatsiooniline andmebaas, ning seejärel analüüsitakse vastava tüübi andmebaasi variante ja valitakse välja sobilik lahendus.

3.1 Rakenduse nõuded

Loodav veebirakendus vastab eelnevalt välja toodud kirjeldusele. Nõuded koosnevad ettevõtte soovidest ning autoripoolsetest ettepanekutest. Nõuete realiseerimiseks loob autor hajusveebirakenduse, mille teostamiseks luuakse nii taga- kui ka eesrakendus.

Nõuete kirjeldamiseks on kasutatud agiilse arenduse metoodikat. Töö on jaotatud epikuteks, mis kirjeldavad üldisemat funktsionaalsust, ning epikud omakorda kasutajalugudeks, mis kirjeldavad iga funktsionaalsuse erinevaid aspekte [3]. Kasutajad jagunevad nelja rolli: peaadministraator, administraator, töötaja ja külaline. Peaadministraatori rollis on autor ise. Peaadministraatorile jäävad õigused süsteemi ettevõtteid lisada ning nende andmeid hallata. Administraatori rollis on ettevõtte töötaja, kes saab oma ettevõtte andmeid hallata. Töötaja rollis on ettevõttes töötavad laooperaatorid, kes hoolduslogisid täidavad. Külalise rollis on ettevõtte kliendid, kes soovivad rakenduses vaatamisõiguseid.

Epikute ja kasutajalugude kaudu kirjeldatud nõuded on välja toodud lisas 1.

Kuna ettevõte avaldas soovi süsteem kasutusele võtta ka teistes riikides, peab olema võimalik valida nimekirjast sobiv ettevõte ning selle süsteemi sisse logida. Ettevõtte süsteemis on kasutajatel võimalik vaadata kõiki laohooneid ning nende alla kuuluvaid laoüksuseid. Iga laoüksuse kohta peab olema võimalik näha selles laoüksuses tehtud hooldustöid – kes on tööd teinud, mis liiki tööd on tehtud, millal töö on tehtud ning milliseid märkmeid sissekandele lisatud on. Samuti peab olema võimalik näha, mis tooteid laoüksuses hetkel ladustatakse.

Sarnaselt laoüksustele peab kasutajal olema võimalik näha ettevõtte süsteemi kuuluvaid laotehnika hoolduslogisid ning nende sissekandeid. Sissekannetel on sarnased andmed. Igal laotehnika hoolduslogi real märgib kasutaja linnukesega, kas tegevus sai tehtud või mitte, ning lisab vajadusel kommentaari. Samuti salvestatakse sissekandele hooldustöö tegija nimi ja kuupäev.

Kasutajatel peab olema võimalik täita eelmainitud välju ning luua sissekandeid mõlemasse logiraamatusse. Kasutajakogemuse mugavdamiseks peab olema võimalikult palju informatsiooni täitmisel automatiseeritud (täitja nimi, kuupäev).

Administraatoriliideses peab ettevõtte administraatoril olema võimalik lisada, uuendada ning kustutada ettevõtte laohooneid ja nendes sisalduvaid üksuseid, ettevõttes ladustatavaid tooteid, laotehnika hoolduslogisid ning ettevõtte koristusliike. Samuti peab administraatoriliidese kaudu võimalik olema logiraamatute sissekannete muutmine ja kustutamine. Süsteemis puudub avalik registreerimine ning iga ettevõtte süsteemis peab kasutajate lisamine ja eemaldamine käima administraatori kaudu. Lisaks sellele peab administraator saama vajadusel muuta ka kasutajate andmeid ja rolle.

Süsteemi peaadministraatoril peavad olema kõik tavaadministraatori õigused, kuid lisaks sellele peab ta saama süsteemi ettevõtteid lisada ning neile administraatoreid määrata.

3.2 Rakenduse tehnoloogia valik

Tänapäeval on veebirakenduste loomisel programmeerimiskeele ning keelel põhinevate raamistike valik lai, kuid enamjaolt kasutatakse kümmet populaarseimat keelt [4]. Aja kokkuhoiu eesmärgil ning valimi kitsendamiseks on autor taga- ja eesrakenduse programmeerimiskeele valikute hulka arvanud vaid need keeled, millega tal varasem kokkupuude ja kogemus on. Uue keele ning omakorda uue raamistiku õppimine kujuneks aja mõttes üleliia kulukaks. Sellest tulenevalt on tagarakenduse programmeerimiskeelte valikus Java, C#, JavaScript, PHP ja Python ning eesrakenduse valikus JavaScript ja TypeScript.

3.2.1 Tagarakenduse programmeerimiskeele valik

Enne, kui on võimalik valida programmeerimiskeelele loodud raamistik, tuleb langetada otsus, milline programmeerimiskeel valida. Programmeerimiskeele valikul tuleks kaaluda keele keerukust, õppimiskõverat, kommuuni suurust, populaarsust, jõudlust ning selle loomise eesmärki [5].

Java

Java on laialdaselt kasutatud objektorienteeritud programmeerimiskeel, mis loodi 1995. aastal Sun Microsystemsi poolt [6]. Java tugevusteks on võimalus jooksutada oma koodi igal platvormil ning selle lihtsasti loetav süntaks, mis on tuletatud C++ keelest. Java kasutab mitmekeermelist (*multi-threaded*) veebiserverit, mis teeb Javast väga hea valiku protsessorile toetuvate rakenduste arendamisel. 20-aastane staaž ning laialdane omaksvõtt arendajate poolt tähendab, et Java keelele on loodud mitmeid raamistikke ning erinevaid teeke, mis arendusprotsessi mugavamaks teevad [7].

C#

C# on 2001. aastal Microsofti poolt välja lastud objektorienteeritud programmeerimiskeel. Sarnaselt Javale on C# edasiarendus programmeerimiskeelest C++. Keele loomisel oli eesmärgiks luua lihtsasti õpitav objektorienteeritud keel, mida saaks rakendada erinevate projektide loomiseks. C# abil on võimalik luua nii veebi-, mobiili- kui ka töölauarakendusi ning peale .NET Core raamistiku (praegune .NET) väljalaset toetab see samuti rakenduste jooksutamist igal platvormil [8], [9], [10].

JavaScript

JavaScript loodi 1995. aastal objektorienteeritud skriptimiskeelena, mida kasutatakse peamiselt veebilehtede arendamiseks – selle abil on võimalik muuta veebilehtede funktsionaalsust ning struktuuri, et luua interaktiivset ja seadmetundlikku disaini. Tänu sellistele tehnoloogiatele nagu Node.js ja Express on nüüd võimalik JavaScripti abil luua ka tagarakenduse projekte [8], [11].

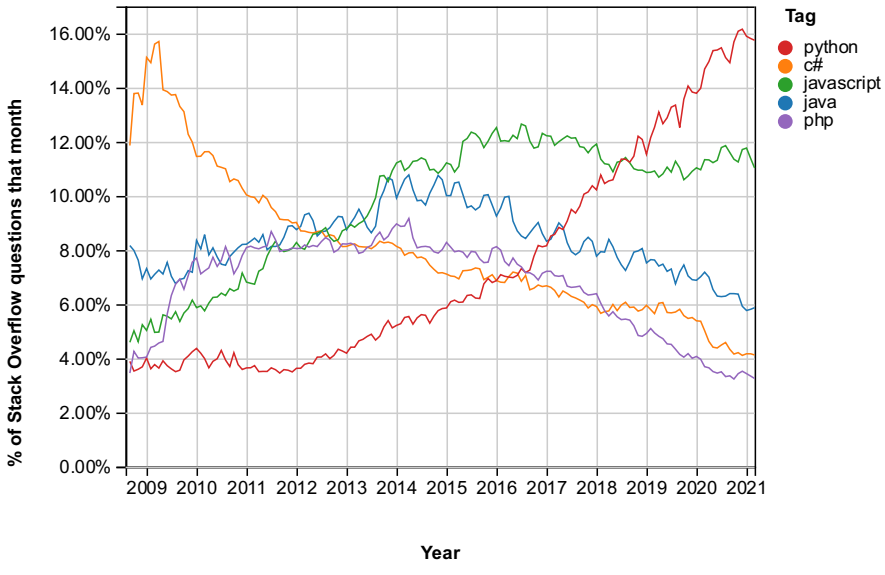
PHP

PHP on 1994. aastal loodud mitmeotstarbeline programmeerimiskeel, mis on kujunenud üheks enimkasutatud serverrakenduste loomise keeleks. PHP populaarsuse tagab tema lihtsus ning laialdane toetus erinevatele teekidele. PHP rakendusi on samuti võimalik kõikidele platvormidele paigaldada [8].

Python

Python on 1991. aastal loodud mitmeotstarbeline programmeerimiskeel, mille abil on võimalik luua nii objektorienteeritud rakendusi kui ka lühikesi käsurea skripte. Pythoni populaarsuse taga on kergesti loetav süntaks ning lai valik teeke erinevateks vajadusteks. Kuigi Python on algajate seas populaarne, ei ole see üldisemalt veebiarenduses kasutusel aeglasema kiiruse ja skaleeritavuse tõttu [8], [12].

Tagarakenduse programmeerimiskeele valimiseks on autor kaalunud eelnimetatud aspekte ning autori enda kogemust programmeerimiskeele kasutamisel tagarakenduse loomisel. Samuti on autor arvesse võtnud programmeerimiskeele õppimisele kuluvat aega ning kättesaadavate materjalide hulka programmeerimisküsimuste lehel Stack Overflow saadaval oleva statistika põhjal, mis näitab mitu protsenti kogu postituste hulgast on tehtud selle keele kohta. Allpool on toodud postituste statistika (joonis 1) ning tabel iga programmeerimiskeele kohta käivate andmetega (tabel 1).



Joonis 1. Programmeerimiskeelte trend platvormil Stack Overflow [13].

Tabel 1. Tagarakenduse programmeerimiskeelte võrdlus.

Keel	**Autori kogemus tagarakenduse kasutamisel**	**Hinnanguline õppimise keerukus**	**Levik postituste statistika põhjal**	**Jõudlus**
Java	Hea	Keskmine	Keskmine	Hea
C#	Väga hea	Keskmine	Keskmine	Väga hea
JavaScript	Hea	Keskmine	Hea	Keskmine
PHP	Puudulik	Madal	Keskmine	Keskmine
Python	Keskmine	Madal	Väga hea	Puudulik

Nende andmete põhjal on valikust väljas Python ja PHP, mis nii jõudluse kui ka autori kogemuse poolest jäävad teistele keeltele alla. See jätab valikusse Java, C# ning JavaScripti. Kahe olulisema aspektiga, st autori kogemuse ja jõudlusega, arvestades langeb valikust taas välja Javascript ning autoril jääb valida Java ja C# vahel. Kuigi keelte levik Stack Overflow statistika põhjal on samal tasemel ning jõudlus mahukamate veebirakenduste puhul on Java rakendustes parem, siis arvestades töö jaoks loodava rakenduse mahu ning autori kogemusega tagarakenduse arendamiskeeltega, sobib C# paremini. Nendest aspektidest lähtudes otsustas autor tagarakenduse programmeerimiskeeleks valida C#.

3.2.2 Tagarakenduse raamistiku valik

C# keeles rakenduse loomisel kasutatakse enamasti Microsofti enda loodud .NET Frameworki ja uuemat .NET (endine .NET Core) raamistikku. Kahe raamistiku erinevus on platvorm, millel neid paigaldada võimalik on – .NET Framework raamistiku abil loodud rakendusi on võimalik paigaldada ainult Windowsi süsteemides. .NET raamistiku abil loodud rakendused on platvormiülesed ning samuti parema jõudluse ning skaleeritavusega kui .NET Framework rakendused tänu võimalusele juurutada rakendust *live* keskkonda, kasutades Dockeri virtuaalkonteinerit [14].

Nii .NET Framework kui ka .NET raamistik toetavad veebirakenduste loomist vastavate raamistike abil. .NET Framework raamistiku jaoks loodud ASP.NET veebirakenduste raamistik võimaldab luua serveripõhiseid veebirakendusi Windowsi

operatsioonisüsteemile. .NET raamistiku veebirakenduste raamistik ASP.NET Core on aga platvormiülene, pakub paremat jõudlust, on paindlikum ning ühildub rohkemate tehnoloogiatega [15].

Selleks, et tagada paindlikkus tagarakenduse paigaldamisel, tuleks valida .NET raamistik, sest see tagab nii platvormiülesuse kui ka võimaluse kasutada Dockeri virtuaalkonteinereid. Samuti on .NET puhul tegu uuema raamistikuga, mida aktiivselt toetatakse ja arendatakse. .NET Framework raamistikule rohkem uuendusi ei tule, küll aga säilib loojate toetus ja ühilduvus [14], [15], [16].

Arvestades neid aspekte ning nõudeid loodavale rakendusele otsustas autor tagarakenduse raamistikuna kasutada .NET raamistikku ja ASP.NET Core veebirakendusraamistikku.

3.2.3 Eesrakenduse programmeerimiskeele valik

Tagarakenduse veebiraamistikuna ASP.NET Core'i valimine jätab võimaluse kasutada lisaks REST (*Representational State Transfer*) API (*Application Programming Interface*) liidesele ka MVC (*Model-View-Controller*) struktuuri eesrakenduse loomisel. Rakenduse veebiliidese loomisel on plaan kasutada nii MVC struktuuri kui ka REST API't – ASP.NET Core MVC abil luuakse rakenduse adminstraatoriliides, kus on võimalik süsteemi hallata, ning ASP.NET Core API kontrollerite kaudu luuakse REST lõpp-punktid, mille abil on võimalik rakendust kasutada [17], [18].

Kuna planeeritav administraatoriliides vajab ainult CRUD (*Create, Read, Update, Delete*) operatsioonide võimalust, siis on ASP.NET Core MVC struktuur selle jaoks sobiv, sest tänu koodigenereerimistööriistadele saab kergelt luua veebikontrollerid ja baasdisainiga veebilehed, mis neid operatsioone võimaldavad. Administraatoriliidese madalama kasutusaktiivsuse tõttu on genereeritud lehed hea viis liidese loomiseks, sest see aitab kokku hoida väärtuslikku aega, mis muidu kuluks andmete haldamist võimaldava koodi kirjutamisele [19].

Kasutajatele mõeldud liidese loomiseks kasutab autor eraldiseisvat eesrakendust. Eraldi eesrakendus tagab projekti modulaarsuse. Vajadusel on võimalik eesrakendust uuendada või juurutada, ilma tagarakendust mõjutamata. Samuti jätab ees- ja tagarakenduse lahus

hoidmine võimaluse luua erinevaid liideseid eesrakenduse realiseerimiseks – lisaks veebirakendusele on võimalik luua ka mobiilirakendust. Viimasena tagab eraldi eesrakenduse ja tagarakenduse hajussüsteem parema skaleeritavuse, sest rakendusi on võimalik paigaldada erineva võimekusega serveritesse, üksteisest sõltumata [20].

Eesrakenduse loomisel on autoril võimalik valida kahe keele vahel – JavaScript ja TypeScript. JavaScript on objektorienteeritud skriptimiskeel, mida kasutatakse peamiselt veebilehtede arendamiseks. TypeScript on JavaScripti edasiarendus ning programmeerimiskeel, mis saavutab sama funktsionaalsuse, kuid lisab koodile staatilise tüüpimise. TypeScript muudab arendusprotsessi ning koodisilumist kiiremaks ning mugavamaks, sest erinevalt JavaScriptist toob TypeScript koodivead esile juba kompileerimisel, mitte programmi jooksutamisel. Kuid kuna TypeScript põhineb JavaScriptil, siis kulub TypeScripti koodi kompileerimisele rohkem aega [21]. Sellegipoolest otsustas autor sujuvama ning stabiilsema koodikirjutamise nimel valida eesrakenduse programmeerimiskeeleks TypeScripti.

3.2.4 Eesrakenduse raamistiku valik

Eesrakenduse raamistiku valikul on autor taas valikusse võtnud ainult need raamistikud, millega tal varasem kokkupuude ja kogemus on. See toob lõplikusse valikusse kolm raamistikku: React.js, Vue.js ning Aurelia. Kuigi kõik need raamistikud on loodud JavaScriptiga kasutamiseks, toimivad nad ka TypeScripti kasutades.

React.js

React.js on Facebooki arendustiimi poolt loodud JavaScripti raamistik, mille abil on mugavalt võimalik luua interaktiivseid ja seadmetundlikke veebirakendusi. React.js'i komponentidel põhinev arhitektuur lihtsustab koodi taaskasutamist ning vähendab koodibaasi mahtu. React.js'i kasutavad mitmed suured tehnoloogiaettevõtted oma veebilehtede loomiseks ning see on üks populaarsemaid programmeerimiskeeli arendajate seas. React.js'i tugevusteks on selle jõudlus ning skaleeritavus [22].

Vue.js

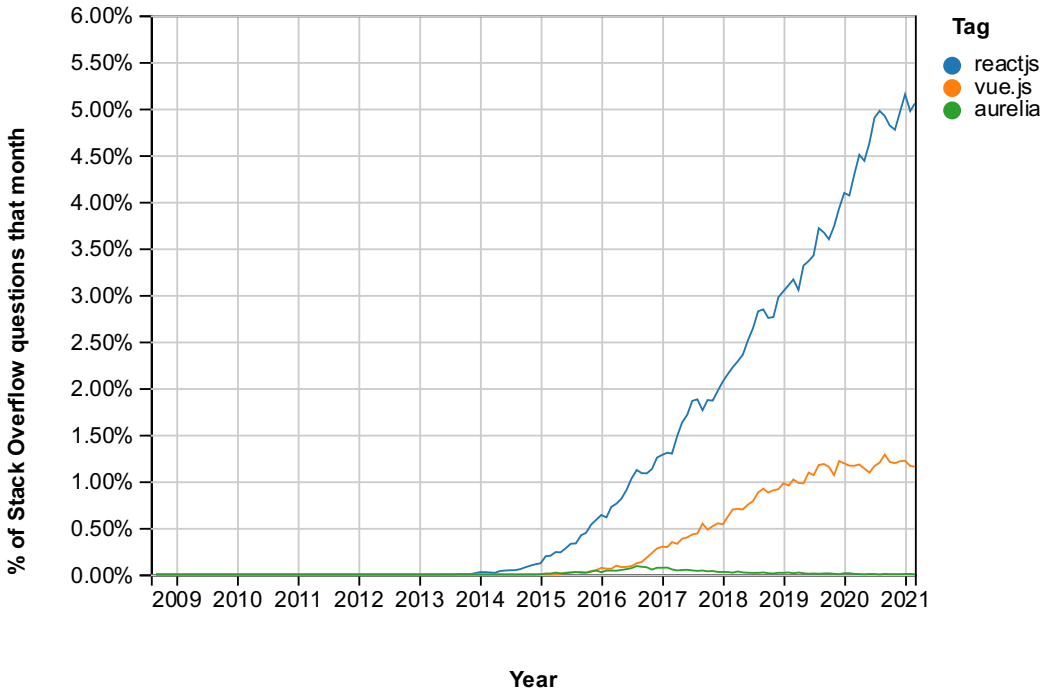
Vue.js on Evan You loodud JavaScripti raamistik, mis kasutab lehtede loomiseks MVC struktuuri. Iga lehe puhul on ühte faili koondatud eesrakenduse funktsionaalne osa, kasutajale kuvatava lehe mall ning mallile rakendatavad kujundusreeglid. Vue.js on samuti väga suure kogukonnaga ning programmeerijate seas populaarne raamistik. Vue.js'i tugevusteks on selle lihtne õppimiskõver, ulatuslik dokumentatsioon ja hea jõudlus [22], [23].

Aurelia

Aurelia on kolmest raamistikust kõige vähem tuntud. Aurelia struktuur sarnaneb Vue.js'ile – iga lehe kuvamiseks on olemas mall ning mallile andmeid lisav funktsionaalsuse osa, kuid erinevalt Vue.js'ist on need eraldatud eri failidesse. Aurelia vähese populaarsuse tõttu on raske leida arendajate lahendusi ning õppematerjale, kuid seda korvab sisukas ametlik dokumentatsioon ning raamistiku üldine lihtsus. Aurelia tugevuseks on vähese konfigureerimise vajadus ning koodi loetavus tänu HTML-mallide (*HyperText Markup Language*) ning äriloogika komponentide eraldatusele [22], [24].

Sarnaselt tagarakenduse programmeerimiskeele ja raamistiku valimisele kehtivad eesrakenduse raamistiku valimisel samad tingimused: populaarsus, õppematerjalide kättesaadavus, tunnused, funktsionaalsus ning suures osas ka autori eelistus ja oskused raamistiku kasutamisel [25].

Autor on taas arvesse võtnud lehel Stack Overflow saadaval olevat statistikat, mis näitab mitu protsenti iga kuu postitustest on olnud vastava raamistiku teemal. Allpool on toodud postituste statistika (joonis 2) ning tabel iga raamistiku kohta käivate andmetega (tabel 2).



Joonis 2. JavaScripti raamistike trend platvormil Stack Overflow [26].

Tabel 2. Eesrakenduse raamistike võrdlus.

Raamistik	**Autori kogemus raamistiku kasutamisel**	**Hinnanguline ajakulu raamistiku õppimiseks**	**Levik postituste statistika põhjal**	**Jõudlus**
React.js	Väga hea	Keskmine	Väga hea	Väga hea
Vue.js	Keskmine	Madal	Keskmine	Väga hea
Aurelia	Puudulik	Madal	Kehv	Väga hea

Tabelis esitatud andmete põhjal jääb esimesena valikust välja Aurelia – autori kogemus raamistiku kasutamisel on piiratud ning kuigi raamistiku jõudlus on hea ja kasutama õppimise ajakulu madal, ei ole vähese leviku tõttu väljaspool dokumentatsiooni informatsiooni leida ning seetõttu võib arendusprotsess lihtvigade taha seisma jääda. Valikusse jäävad React.js ja Vue.js. Raamistikud on jõudluselt võrdsed aga õppimise ajakulus on Vue.js üle. Samas on React.js statistika põhjal märkimisväärselt rohkem levinud ning kõige suuremat kaalu omav tegur, st autori kogemus, on React.js'i puhul kõige ulatuslikum.

Kuigi Vue.js on võrdväärne kandidaat nii raamistiku lihtsuse, hea leviku kui ka samaväärse jõudlusega, on aja säästmise ning mugavuse huvides autori valik React.js. Eesrakenduse disainimiseks on kasutatud nii Bootstrap kui ka MaterialUI disainiraamistikke, mis kasutajaliidese disaini hõlbustavad ning arendusprotsessi kiirendavad.

3.3 Rakenduse andmebaas

Selleks, et rakenduse andmeid talletada, on vajalik valida rakendusele sobiv andmebaas. Valiku tegemisel tuleb valida andmebaasi struktuur ning seejärel vastava struktuuriga andmebaasi haldussüsteem. Andmebaasi haldussüsteemi valimisel tuleb samamoodi nagu muude tehnoloogiatega kaaluda populaarsust, materjalide kättesaadavust, jõudlust, funktsionaalsust ning hinda.

3.3.1 Andmebaasi struktuur

Veebirakenduse andmebaasi valimisel tuleb esmalt otsustada, kas valida relatsioonilise või mitterelatsioonilise (NoSQL ehk *Not only SQL*) struktuuriga andmebaas. Mõlemal valikul on omad tugevused ja nõrkused ning valik peaks sõltuma kasutusjuhust.

Relatsioonilised andmebaasid võeti programmeerimises kasutusele 1970ndatel aastatel. Sellest ajast saadik on need erinevates rakendustes laialdaselt levinud ning nende jõudlust ja vastupidavust tõestab nende populaarsus läbi aastate. Relatsioonilisi andmebaase kirjeldab andmete struktureeritus omavahel seoses olevates tabelites ning ACID (*Atomicity, Consistency, Isolation, Durability*) omadused andmebaasi transaktsioonides [27].

ACID omadused transaktsioonides tähendab [28]:

- Atomaarsus – andmed salvestatakse andmebaasi ainult siis, kui kõikide salvestatavate andmete õigsuses on veendutud.
- Järjepidevus – salvestatavad andmed ei saa rikkuda andmebaasi terviklikkust. Vastasel juhul andmete muutmise operatsioon katkestatakse ning andmebaasi eelnev seisund taastatakse.

- Isoleeritus – andmebaasioperatsioonide puhul jälgitakse, et mitme samaaegse operatsiooni korral oleks tulemusena andmebaasi olek samaväärne, kui operatsioonid oleks täidetud järjestikuselt.
- Vastupidavus – isegi süsteemirikke puhul lähevad kinnitatud operatsioonid läbi.

NoSQL ehk mitterelatsiooniliste andmebaaside puhul on tegemist hilisema tehnoloogiaga, mis on hoogu kogunud viimasel sajandil. NoSQL andmebaasides võivad andmed olla talletatud erineval kujul – võti-väärtus kogumina või JSON (*JavaScript Object Notation*) dokumentidena. NoSQL andmebaaside tugevuseks on nende skaleeritavus ning kiiruse püsivus suure koormusega süsteemides. Negatiivse küljena puuduvad NoSQL andmebaasides tihti ACID omadused [27], [29].

Arvestades loodava veebirakenduse olemust on autor otsustanud andmebaasi haldussüsteemide valimisse võtta relatsioonilise struktuuriga andmebaasid. Rakenduse kasutushulk ei saavuta niivõrd kõrget mahtu, et tuleks kaaluda NoSQL andmebaasi, ning samuti on tegemist struktureeritud andmetega, mis teeb relatsioonilisest andmebaasist sobiva valiku.

3.3.2 Andmebaasi haldussüsteemi valik

Andmebaasi haldussüsteemi valimisel on autor piirdunud nelja populaarsema relatsioonilise andmebaasi haldussüsteemi analüüsimisega: SQL Server, PostgreSQL, MySQL ning OracleDB.

SQL Server on Microsofti loodud andmebaasi haldussüsteem, mida on võimalik kasutada nii kohapealsetes serverites kui ka pilvepõhise teenusena. SQL Server toimib nii Linuxi kui ka Windowsi süsteemides ning ühildub hästi Microsofti enda loodud tehnoloogiatega. See on kiire ja usaldusväärne andmebaasi haldussüsteem, mille nõrgaks küljeks on selle kõrge maksumus väljaspool SQL Express prooviversiooni [30], [31].

PostgreSQL on valikutest kõige uuem – aastal 1997 avalikustatud vabavaraline haldussüsteem on tuntud nii oma rohkete omaduste kui ka hea võimekuse ning skaleeritavuse poolest. Samuti on PostgreSQL'i andmebaasis tugev rõhk andmete terviklikkusel – andmebaasi transaktsioonid on ACID nõuetele vastavad. PostgreSQL'i negatiivseks pooleks on staažist tulenev madalam populaarsus võrreldes teiste

haldussüsteemidega. See tähendab, et dokumentatsiooni ning veebimaterjalide leidmine võib osutuda keerukamaks kui teiste süsteemide puhul. Vabavaraline andmebaasi haldussüsteem tähendab, et PostgreSQL'i on võimalik paigaldada erinevatele operatsioonisüsteemidele [30], [31].

MySQL on samuti vabavaraline andmebaasi haldussüsteem ning seda peetakse üheks populaarseimaks andmebaasisüsteemiks arendajate seas. MySQL on töökindel ning üldkasutuseks sobilik andmebaasi haldussüsteem, mis jääb küll omaduste poolest teistele alla, kuid heastab selle oma kasutajalihtsuse ning andmebaasi lugemisoperatsioonide kiirusega. Kompromiss selle võimekuse saavutamiseks on kehvem jõudlus komplekssete päringute tegemisel. MySQL sobib kasutamiseks nii Windowsi, Linuxi kui ka macOS'i süsteemides [30]-[33].

Oracle on tuntud oma kvaliteetsete andmebaasisüsteemide poolest ning nende andmebaasi haldussüsteem OracleDB just seda ongi. OracleDB puhul on tegu kõige arenenuma ning vastupidavama andmebaasitehnoloogiaga, mille abil on võimalik talletada miljardeid kirjeid, jõudlust ohverdamata. Uusim versioon 19c on loodud pilvepõhise teenusena kasutamiseks ning sisaldab rohkem omadusi kui alternatiivsed variandid. OracleDB suurimaks puuduseks on hind – andmebaasi haldussüsteem on loodud eeskätt suurettevõtetele kasutamiseks [30], [31].

Autor valib andmebaasi haldussüsteemiks MySQL'i. Haldussüsteem sobib, sest rakenduse peamine koormus andmebaasile saab olema lugemisoperatsioonide teostamisel ning populaarsus ja laialdane kasutus arendajate seas tagab, et vajadusel on võimalik leida abistavaid veebimaterjale. Samuti on tänu MySQL'i platvormiülesusele mugav seda lokaalse arenduse jaoks kasutada.

4 Logistikaettevõtte hoolduslogi rakenduse loomine

Rakenduse arendusprotsess jaguneb kolme osasse – andmebaasimudeli koostamine, tagarakenduse loomine ning eesrakenduse loomine.

4.1 Rakenduse andmebaas

Analüüsi peatükis valis autor andmebaasi haldussüsteemiks MySQL'i, kuid enne tagarakenduse kirjutamisega alustamist tuleb otsustada, mida kasutada andmebaasi tabelites primaarvõtmena, ning luua andmebaasimudel, mis kirjeldab tabeleid ning nende veerge.

4.1.1 Primaarvõtme valik

Rakenduse andmebaasi loomisel on oluline kaaluda, millist andmetüüpi kasutada olemite primaarvõtmena. Primaarvõti, sageli andmebaasitabelites märgitud kui „ID" veerg, on tabeli atribuutide ehk veergude kogumik, mis võimaldab tuvastada tabeli igat rida. Iga rida peab omama primaarvõtit ning võti peab olema oma tabelis unikaalne [34].

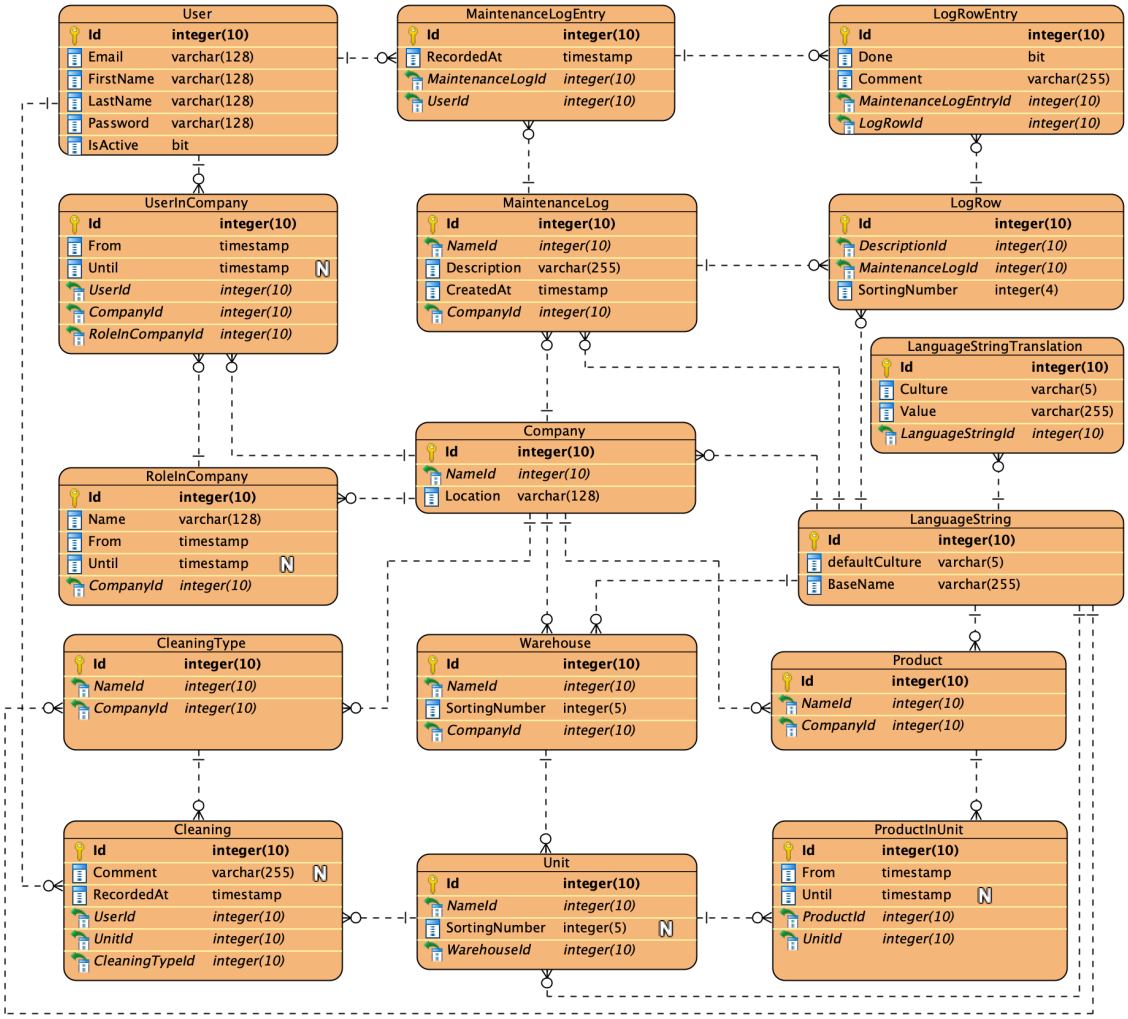
Täisarv ehk *integer* on lihtsaim primaarvõtme andmetüüp, mille tugevuseks on selle väike maht mälus – täisarv tüüpi primaarvõtmed on suuruselt 4 baiti. See tähendab, et andmebaasi päringute tegemine on tänu võtmete järjestusele kiire. *Integer* tüüpi võtmete negatiivseks küljeks on nende piiratud kogus igas tabelis, sest 4-baidise suuruse tõttu on võimalik talletada ainult üle kahe miljardi võtme, mis võib tekitada probleeme suurtes süsteemides. Selle lahenduseks on küll võimalik suurendada primaarvõtme andmetüüpi *bigint* andmetüübile, mis on enamikul juhtudel piisav [35], [36].

GUID (*Globally Unique Identifier*) on 16 baidi suurune andmestruktuur, mis koosneb 32-st kuueteistkümnendsüsteemi numbrist. GUID-ide suurim tugevus on nende unikaalsus – tõenäosus, et kaks suvaliselt genereeritud GUID-i on identsed, on nullilähedane. Unikaalsete GUID primaarvõtmete hulk on 2^{128}. See tähendab, et need on hea lahendus massiivsete hajussüsteemide jaoks, sest primaarvõtmeid on võimalik genereerida väljaspool andmebaasi, kartmata kasutusel oleva võtme loomist. Kuid andmetüübi suurem suurus tähendab ka kõrgemat koormust andmebaasile [35], [37].

Kuigi GUID on kahest valikust suuremõõtmeliste süsteemide puhul loogiline valik, siis ei ole töö raames loodava rakenduse ulatus niivõrd lai, et see õigustaks GUID primaarvõtme kasutamist. Seetõttu valib autor andmebaasi primaarvõtme andmetüübiks täisarvu. Juhul, kui rakendus saavutab tulevikus *integer* andmetüübi ülempiiri, on võimalik üle minna suuremale andmetüübile.

4.1.2 Andmebaasimudeli kirjeldus

Rakenduse andmebaasimudel on kujutatud olemi-suhte diagrammi kaudu. Diagramm kirjeldab andmebaasis olevaid tabeleid ning nendevahelisi seoseid. Samuti on iga tabeli kohta kirjeldatud selle sisemine struktuur – veerud ning nende andmetüübid. Rakenduse diagramm (joonis 3) koosneb 16 olemist.



Joonis 3. Andmebaasi olemi-suhte diagramm.

Süsteemi keskmes on *Company* ehk ettevõtte olem, millel on nimi ning asukoht, mis võib olla vabas vormis tekst. *Company* juurde kuuluvad vahetabeli *UserInCompany* kaudu ettevõtte süsteemis olevad kasutajad (*User*) ning nende rollid (*RoleInCompany*). Vahetabel tagab võimaluse luua kasutajaid, kellel on ligipääs mitme ettevõtte süsteemile, nagu näiteks administraator.

Ettevõtte olemiga on seotud laotehnika hoolduslogid (*MaintenanceLog*). Laotehnika hoolduslogid on mõeldud eri masinate ning tööriistade hooldamise jälgimiseks. Iga hoolduslogi omab nime, mis tähistab, mille hooldamist kirja pannakse, kirjeldust, mis on vabas vormis tekst, kus võib selgitada täitmise graafikut, ning kuupäev, mis märgib hoolduslogi loomist. Iga hoolduslogi alla kuuluvad selle read (*LogRow*), millel on kirjeldus ning sorteerimisnumber, et tagada ridade korrektne kuvamise järjekord eesrakenduses. Hoolduslogi ridu luues tuleb meeles pidada, et hoolduslogi täites on võimalik lisada jah/ei vastus ning kommentaar, nii et kirjeldus peab olema vastav.

Iga hoolduslogi ning selle ridade kohta on võimalik kasutajatel luua sissekanne (*MaintenanceLogEntry* ja *LogRowEntry*). Sissekandel märgitakse loomise kuupäev ning iga rea täitmisel on kasutajal võimalik märkida, kas vastav hooldustöö osa sai tehtud või mitte, ning vajadusel saab märkida juurde kommentaari, juhul kui seadmega esineb probleeme. Iga sissekanne on seotud kasutaja ja täidetava hoolduslogiga.

Company juurde kuuluvad laohooned (*Warehouse*). Iga laohoone omab nime ning sorteerimisnumbrit, mis on kasutusel, et tagada laohoonete soovitav kuvamise järjekord eesrakenduses. Iga laohoone alla kuuluvad veel omakorda selle laoüksused (*Unit*), mis omavad samuti nime ning sorteerimisnumbrit. Laoüksuse juurde kuuluvad ka selle laoüksuse hooldused ehk koristused (*Cleaning*). Iga koristuse juures tuleb märkida, mis laoüksust koristati, ning koristusliik (*CleaningType*), mis omab kirjeldavat nimetust tehtava töö kohta. Iga koristus on seotud kasutajaga ning koristuse juurde on võimalik märkida kommentaar, juhul kui laooperaatoritel esineb koristamisel probleeme. Iga koristuse lisamisel salvestatakse ka koristuse kuupäev.

Viimasena kuuluvad ettevõtte alla tooted (*Product*). Igal tootel on kirjeldav nimi. *ProductInUnit* vahetabelis märgitakse, mis tüüpi toodet on kindlas laoüksuses hoitud ning mis on hoiustamise alg- ja lõppkuupäev. Ladudesse toodete lisamine annab ka ülevaate,

kui palju ning kui tihti koristustöid laoüksuses tegema peaks, sest ettevõttel on erinevaid tooteid ladustavatele laoruumidele erinevad koristusgraafikud.

Paljud tabelid kasutavad sõne andmetüübi asemel viidet *LanguageString* olemile, millel omakorda on üks mitmele seos *LanguageStringTranslation* olemiga. See tähendab, et sõne asemel hoitakse väljal viidet tõlkeobjektile, mis omab sellele sõnele vastavaid tõlkeid eri keeltes. Igal tõlkel on tõlgitud sõne väärtus ning tõlke keele kood (*culture*), mis koosneb ISO 639 keelekoodist ning ISO 3166 riigikoodist [38].

4.2 Tagarakenduse arendus

Tagarakenduse loomisel on rakenduse peamise kasutuse jaoks loodud API ehk rakendusliides. Samuti on tagarakenduse küljes administraatoriliides, mis on loodud kasutades MVC mustrit. Tagarakenduse loomiseks on kasutatud C# programmeerimiskeelt ning .NET raamistikku ja selle alla kuuluvat ASP.NET Core veebiraamistikku.

4.2.1 Tagarakenduse arhitektuur

Tagarakenduse loomisel on järgitud mitmekihilise arhitektuuri mustrit. Mitmekihilise arhitektuuriga rakendused sisaldavad reeglina nelja eristatavat horisontaalset kihti: esitluskiht (*presentation layer*), äriloogikakiht (*business logic layer* ehk BLL), andmetele juurdepääsu kiht (*data access layer* ehk DAL) ning andmebaasikiht [39]. Tihti viidatakse sellele kui kolmekihilise arhitektuuri mustrile (esitlus-, äriloogika- ja andmekiht), käsitledes andmebaasi eraldi süsteemina [40]. Mitmekihilise arhitektuuri kasutamine aitab luua loetavamat ning kergemini hallatavat koodi ning lihtsustab eri kihtide testimist. Igal kihil on süsteemis kindel eesmärk ning see teeb vajadusel eri kihtide funktsionaalsuse muutmise ning väljavahetamise lihtsamaks [41]. Selle saavutamiseks kasutatakse abstraktsiooni (*abstraction*) – iga kiht kasutab madalama kihi liidest (*interface*), mis avaldab madalamas kihis olevate klasside meetodid, kuid mitte nende sisu.

Süsteemi keskmes on andmebaasikiht. Nimest tulenevalt asub siin rakenduse andmebaas. Andmebaas koosneb rakenduse domeeniobjektidele vastavatest tabelitest ning see vastutab rakenduse andmete hoiustamise eest.

Järgmine kiht on DAL ehk andmete juurdepääsu kiht. DAL kihti kasutatakse andmebaasi kirjete lugemiseks, sisestamiseks, uuendamiseks või kustutamiseks ning see võimaldab hoida kogu andmebaasi operatsioonide funktsionaalsus ühes kohas. Kihis on kasutatud hoidlamustrit (r*epository pattern*), mille eesmärgiks on DAL kihi abstraktsioon. Iga domeeniobjekti kohta on üks hoidla klass, mis sisaldab andmebaasile juurdepääsuks vajalikku loogikat [42]. Hoidlas on kasutatud *Entity Framework* nimelist ORM-i (*object-relation mapper*), mis on vahelüli hoidlate ja andmebaasi vahel ning mis aitab vältida andmebaasiga suhtlemiseks vajaliku madala tasandi koodi kirjutamist kui ka võimaldab vajadusel mugavalt muuta rakenduses kasutatavat andmebaasi [43].

Andmete juurdepääsu kihile järgneb äriloogika kiht ehk BLL. Nimele kohaselt vastutab BLL rakenduse äriloogika eest. See tähendab, et äriloogika kihis on igale domeeniobjektile vastav teenuseklass (*service*), mis vastutab kõikide vajalike arvutuste ning funktsionaalsete toimingute tegemise eest enne andmete andmebaasi salvestamist või presentatsioonikihile edastamist. BLL kihi eraldatuse eesmärk on hoida presentatsioonikihi veebikontrollerites võimalikult vähe äriloogikat ja parandada koodi loetavust. Tihti võib ühe domeeniobjekti teenuseklassis vaja minna ka teiste domeeniobjektide hoidlaid. Selle lahendamiseks kasutavad teenuseklassid tööüksust ehk UOW-d (*Unit of Work*). UOW klass sisaldab kõikide domeeniobjektide hoidlaid, mis tagab võimaluse neid erinevates teenuseklassides kasutada. Samuti võimaldab UOW koondada kõikide hoidlate andmebaasipäringud ühte transaktsiooni, mis aitab vähendada koormust andmebaasile ning parandab rakenduse jõudlust [44].

Viimane ja kõige kasutajapoolsem kiht süsteemis on esitluskiht. Siin asuvad kõik veebikontrollerid, mis kasutajate pöördumisel serveri poole sobiva vastuse tagastavad. Iga domeeniobjekti kohta on eraldi kontroller, mis vastab selle objekti funktsionaalsusega seotud päringutele. Kasutajale andmete tagastamiseks kutsuvad kontrollerid välja domeeniobjekti teenuseklassi meetodeid.

Igas kihis on igale domeeniobjektile vastav andme-edastusobjekt ehk DTO (*data transfer object*). DTO-de kasutamiseks on mitmeid põhjuseid. Andmebaasist andmete pärimisel ning ORM-i abil nende domeeniobjektideks kaardistamisel võib objekt omada välju, mida ei taheta kliendile rakenduses näidata või ei pruugi need olla näitamiseks vajalikud.

Samuti võib tekkida andmebaasikirjete otse domeeniobjektideks kaardistamisel rekursiivseid objektiviiteid. Selle vältimiseks kaardistatakse peale andmebaasipäringut andmed vastava kihi DTO objektiks. See aitab vähendada objektide suurust ning vajadusel võimaldab see mitme päringu andmed komplekteerida ühte objekti, vähendades seeläbi kihtide vahel edastatavate objektide hulka [45], [46].

Nagu andmebaasimudeli kirjelduses selgitati, kasutavad paljud andmebaasi tabelid lihtsõnede asemel tõlkeviidetega objekte. Tõlgete saamiseks on DAL kihi hoidla päringutes vaja LINQ (*Language Integrated Query*) avaldise abil kaasata tõlkeviitadega objekt ning omakorda sellele kuuluvate tõlgete kogum. DAL kihi DTO objektil on nimi sõne andmetüüpi ning päringu andmete DAL kihi DTO-ks kaardistamisel kutsutakse välja tõlkeobjekti *toString()* meetod, mis kontrollib päringu keelekoodi ning otsib tõlgete nimekirjast keelekoodile vastav tõlge. Tõlke leidmisel see tagastatakse, vastasel juhul tagastatakse tõlkeobjekti vaikimisi väärtus. Vaikimisväärtuseks on objekti loomisel määratud sõne, mis on objekti loomise päringus määratud keelekoodis. Allpool on toodud näide (joonis 4), kuidas LINQ avaldises *Company* klass objektile tõlgete kaasamine toimib.

```
query = query
    .Include(c => c.Name)
    .ThenInclude(l => l.Translations);
var dbEntites = await query.ToListAsync();
var result = dbEntites.Select(e => Mapper.Map(e));
```

Joonis 4. Tõlkeobjekti tõlke kaasamine LINQ avaldises hoidla päringus.

4.2.2 REST API

Selleks, et eesrakendusel oleks võimalik tagarakenduselt kätte saada vajalikud andmed, on tagarakenduses loodud REST API kontrollerid, mis vastavad kindlatele REST lõpp-punktidele. Lõpp-punktide nimetused tulenevad kontrollerite nimedest. Igal kontrolleril on API lõpp-punktid GET, POST, PUT ja DELETE päringute jaoks ning kontrollerid vastavad päringutele JSON formaadis. Selleks, et tulevikus API-st sõltuvates süsteemides probleeme ei tekiks, on rakenduses API versioonimine. Eesrakendus saab kasutada kindla versiooniga lõpp-punkti ning olla kindel, et formaat ja funktsionaalsus ei muutu. Seega

on lõpp-punktide muster /api/<versioon>/<kontrolleri nimetus>. API kontrollerite loomiseks on kasutatud *aspnet-codegenerator* käsurea tööriista, mis võimaldab kiirelt genereerida algsed kontrollerid, mida on võimalik lõpplahenduse loomiseks kasutada [47].

REST API kaudu on võimalik andmeid pärida ka kliendi valitud keelde tõlgituna, mis võimaldab luua eesrakenduses mitmekeelset veebilehte. Selleks, et päringutes küsitud andmed oleks tõlgitud, tuleb lisaks muudele päringu parameetritele täpsustada ka eelnevalt mainitud keelekoodi, mis on tõlke sihtkeeleks. Ettevõtete andmete lõpp-punkti mustri näide on toodud allpool (joonis 5).

```
/api/<versioon>/companies?id=<ettevõtteId>&culture=<keelekood>
```

Joonis 5. Ettevõtte andmete kuvamise lõpp-punkti näide.

4.2.3 API kontrollerite turvalisus

Andmete juurdepääsu turvalisuse tagamiseks on REST API veebikontrollerites kasutusel JWT (*JSON Web Token*) autentimine. JWT on standard, mis defineerib kompaktse ja mugava viisi, kuidas saata informatsiooni osapoolte vahel JSON objekti kujul. JWT tähis on digitaalselt allkirjastatud väljastajapoolse võtmega [48]. JWT tähis tagastatakse peale registreeritud kasutaja edukat sisselogimist e-posti, parooli ning ettevõtte ID kaudu. Tähise väljastamisel on sellele määratud ka kehtivusperiood, peale mida tuleb serverist pärida uus tähis. Järgnevatel päringutel serveri poole pöördudes tuleb serveri kliendil JWT võti kaasata HTTP (*Hypertext Transfer Protocol*) päringu päises.

Selleks, et tagada ettevõttele kuuluvate andmete turvalisus, on iga ettevõtte privaatset informatsiooni käsitlevatel kontrolleritel lisakontroll. Igal päringul kontrollitakse, kas kasutaja kuulub ettevõtte alla või mitte. Tähtsamate andmete kontrollerites, nagu näiteks laohooned, laoüksused, ettevõtte koristusliigid jne, kontrollitakse ka kasutaja rolli, et tavakasutajad ei saaks ettevõtte olulisi andmeid muuta või kustutada. Külaliskasutajatel on süsteemis ainult lugemisõigus, töötajatel osaline kirjutusõigus ning administraatoritel täielik kirjutusõigus.

4.2.4 Administraatoriliides

Administraatoriliides on loodud *aspnet-codegenerator* käsurea tööriista abil, mille abil on võimalik genereerida kontrollereid ja vaateid. Administraatoriliidesele on juurdepääs ainult peaadministraatoril, kes saab kõikide ettevõtete andmeid hallata, ning ettevõtte administraatoritel, kes saavad hallata nende ettevõtete andmeid, kuhu nad kuuluvad.

Administraatoriliideses on ettevõtte administraatoril võimalik lisada, uuendada ning kustutada ettevõtte laohooneid ja nendes sisalduvaid üksuseid, ettevõttes ladustatavaid tooteid, laotehnika hoolduslogisid ning ettevõtte koristusliike. Samuti on administraatoril võimalik muuta või kustutada nii laoüksuste ja laotehnika hoolduslogide sissekandeid kui ka laoüksustes hoiustatavat kaupa. Viimasena on administraatoril võimalik hallata ettevõtte süsteemis olevaid kasutajaid ning nende rolle.

4.3 Eesrakenduse arendus

Töö teostamiseks luuakse tagarakenduse peale ka kasutajaliides eesrakenduse kujul, mis on peamine viis rakenduse kasutamiseks. Rakenduse kasutajaliidese loomiseks kasutatakse TypeScripti programmeerimiskeelt ning React.js'i raamistikku. Kasutajaliidese disainimise hõlbustamiseks on kasutatud Bootstrapi ning MaterialUI disainiraamistikke, mis võimaldavad kiirelt luua stiliseeritud HTML-i elemente ning neid rakendusele vastavalt kohandada.

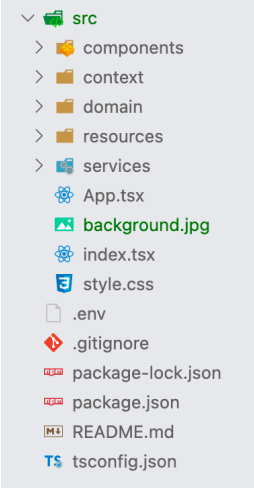
4.3.1 Eesrakenduse projekti struktuur

Eesrakenduse struktuur on ülesehituselt lihtsam kui tagarakendus. Rakenduse projekti kuuluvad välisteeke sisaldav kaust (*node_modules*), üldkasutatavaid ressursse sisaldav kaust (*public*) ning koodibaasi sisaldav kaust (*src*), kuhu lisatakse kõik autori poolt loodavad failid. Samuti sisaldab projekt loomisel automaatselt genereeritud konfiguratsioonifaile, mis täpsustavad rakenduse seadistust.

Rakenduse koodibaasi kausta (joonis 6) kuuluvad:

- Komponentide kaust (*components*), mis sisaldab veebilehtede kuvamiseks kasutatavaid komponente.

- Rakenduse oleku faili sisaldav kontekstikaust (*context*).
- Rakenduse domeeniobjekte kirjeldav kaust (*domain*).
- Tõlkematerjalide kaust (*resources*), mille abil luuakse eesrakenduse mitmekeelsus.
- Teenuseklasside kaust (*services*), milles sisalduvaid klasse kasutatakse komponentides tagarakendusega suhtlemiseks ning andmete pärimiseks.
- Eesrakenduse sisenemispunkti index.tsx fail ning App.tsx komponent.
- Veebilehe esilehel kasutatav taustapilt ning style.css HTML-i elementide kujundusfail.



Joonis 6. Eesrakenduse koodibaasi failistruktuur.

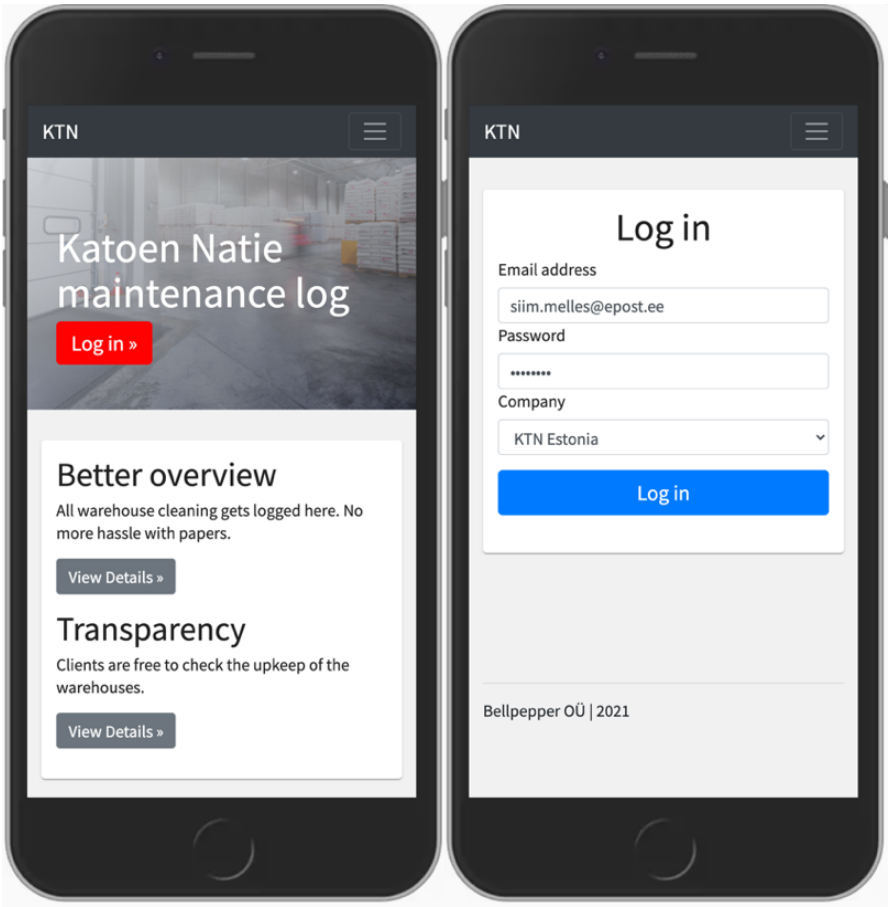
Eesrakenduse sisenemispunktis, index.tsx failis, renderdab rakendus App.tsx komponendi. *App* komponent on eesrakenduse alus. Siin hoitakse rakenduse olekut, mille väljad on defineeritud konteksti kaustas olevas klassis. Selle abil on võimalik kasutajasessiooni konteksti salvestada erinevaid muutujaid nagu JWT tähis, sessiooni keelekood, kasutaja ees- ja perekonnanimi jne.

App komponendis on kasutusel *ReactRouter* komponent, mis vastavalt URL (*Uniform Resource Locator*) mustrile kuvab sobiva *Route* alamkomponendi. Tagamaks, et autentimata ning piiratud õigustega kasutajad ei saaks ligi komponentidele, mis kuvavad ettevõtete privaatset informatsiooni, on loodud eraldi *PrivateRoute* alamkomponent, mis võtab parameetritena valideerimise predikaadi ning ümbersuunamise sihtpunkti. Lugemisõigusega klientidele näidatavate lehtede puhul on predikaadiks JWT tähise olemasolu ning tähise puudumise korral suunatakse kasutaja ümber sisselogimise lehele.

Ettevõtte privaatsemat informatsiooni sisaldavatel lehtedel kontrollitakse nii JWT tähist kui ka kasutaja rolli ettevõttes. Ligipääsu puudumisel suunatakse kasutaja esilehele.

4.3.2 Eesrakenduse kasutajaliidese struktuur

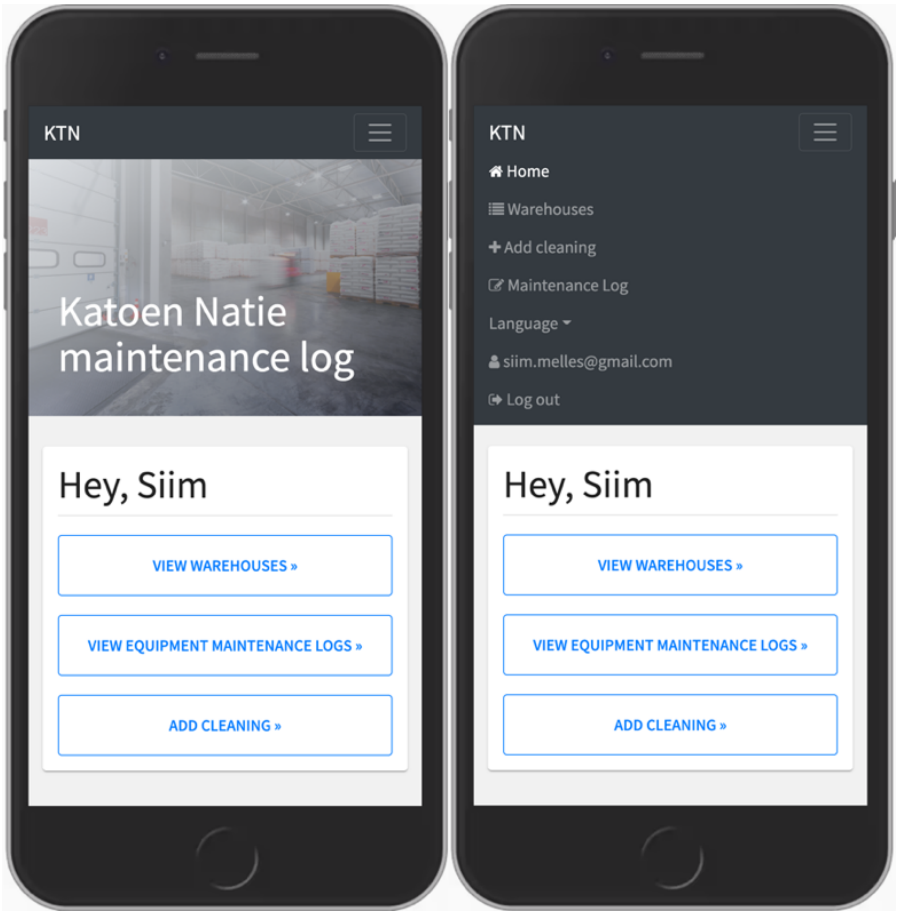
Veebilehe poole pöördudes kuvatakse kasutajale tutvustav esileht (joonis 7), kus on nupp, mis viib sisselogimisvormile (*Log in*). Akna üleval ääres on ka navigatsiooniriba mille abil erinevate lehtede vahel navigeerida, kuid sisselogimata kasutajale kuvatakse seal ainult esilehe, sisselogimise ning keele muutmise valikuid. Sisselogimise lehel (joonis 7) tuleb registreeritud kasutajal sisestada e-posti aadress ja salasõna ning valida nimekirjast ettevõte, mille süsteemi soovitakse sisse logida. Ebakorrektsete sisselogimisandmete korral näidatakse kasutajale veateadet. Eduka sisselogimise järel tagastab tagarakendus JWT tähise, kasutaja andmed ning ettevõtte ID. Need väärtused salvestatakse sessiooni konteksti.



Joonis 7. Veebirakenduse esi- ja sisselogimislehed.

4.3.3 Kasutajaliides ettevõtte töötajatele

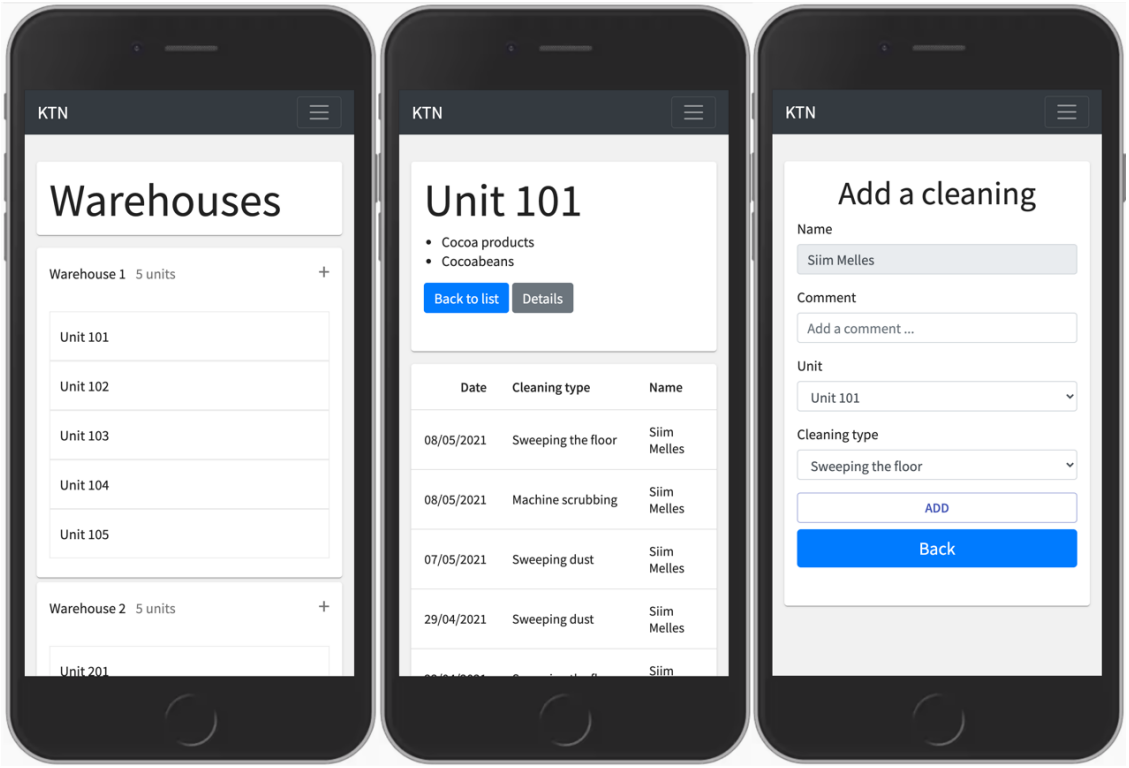
Sisselogitud kasutaja suunatakse tagasi esilehele, kus nüüd kuvatakse erinevaid linke (joonis 8). Samuti on autenditud kasutaja navigatsiooniriba menüüs mitmeid erinevaid lisavalikuid võrreldes anonüümse kasutajaga (joonis 8). Kasutajatel on võimalik menüü kaudu muuta rakenduse keelt, vaadata enda kasutaja profiili informatsiooni ning soovi korral välja logida.



Joonis 8. Sisselogitud kasutaja esileht ja navigatsiooniriba.

Nii esilehe kui ka navigatsiooniriba kaudu on sisselogitud kasutajal võimalik ka navigeerida ettevõtte laohooneid ning nende alla kuuluvaid laoüksuseid kuvavale lehele (joonis 9). Laoüksuse peale vajutades suunatakse kasutaja selle üksuse informatsiooni sisaldavale lehele. Siin kuvatakse hetkel laoüksuses hoitavaid kaupasid ning tabelit, milles on reastatud hooldustööde sissekanded sisestuskuupäeva järgi järjestatuna, kuvades uuemad sissekanded eespool (joonis 9). Ühtlasi on kasutajal esilehe ja navigatsiooniriba kaudu võimalik liikuda hooldustööde laoüksustesse lisamise lehele,

mille kaudu hooldustööde sissekandeid luuakse (joonis 9). Hooldustöö lisamisel tuleb kasutajal lisamisvormil valida laoüksus, kus hooldustööd tehti, ning hooldustöö liik, mida tehti. Vajadusel on kasutajal võimalik hooldustöö sissekandele lisada ka kommentaar. Kuupäev ning töö tegija nimi lisatakse vormile automaatselt. Valiidse vormi esitamisel suunatakse kasutaja ümber laoüksuse hoolduslogi lehele, millele ta hooldustöö lisas. Laoüksuse hoolduslogi lehelt on võimalik edasi liikuda ka laoüksuse detailide lehele, kus lisaks laoüksuse informatsioonile on administraatoril võimalik lisada ja eemaldada laoüksuses ladustatavaid tooteid.

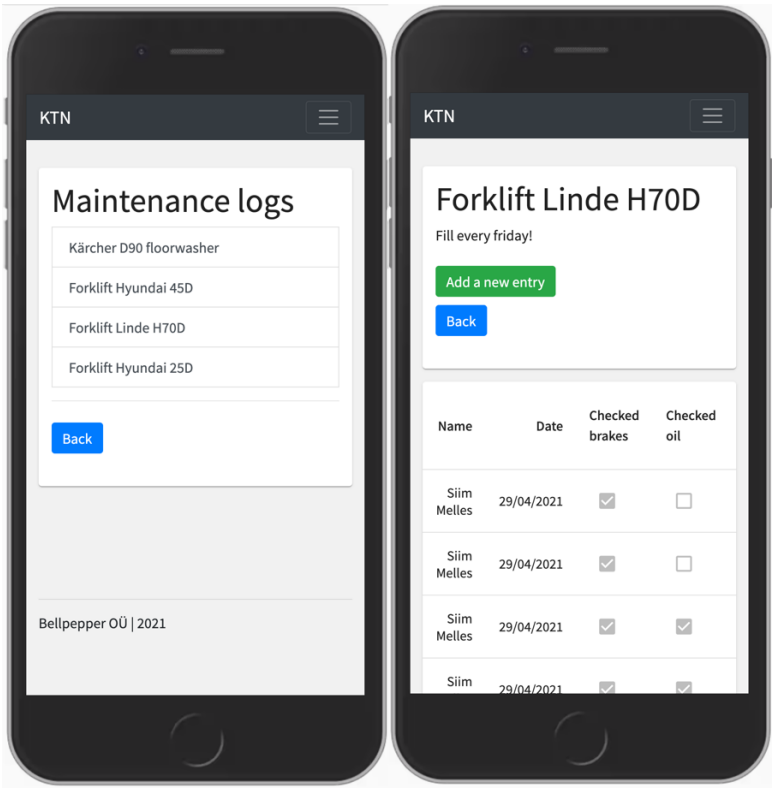


Joonis 9. Laohoonete ja üksuste nimekirja, laoüksuse hoolduslogi ning sissekande lisamise lehed.

Selleks, et tagada veebilehe kasutamismugavus mobiiliseadmetes, ei ole tabeli veergude hulgas hoolduslogi sissekande kommentaari. Mobiilseadme ekraani väiksema vaatamisakna tõttu oleks kommentaari veerg tabelis liialt kokku surutud ning see teeks kommentaari lugemise keeruliseks. Selle lahendamiseks on võimalik tabeli ridadele klikkida, et avada hüpikaken, milles on kuvatud hoolduslogi täpsemad andmed (joonis 10).

Joonis 10. Hoolduslogi sissekande detaile sisaldav hüpikaken.

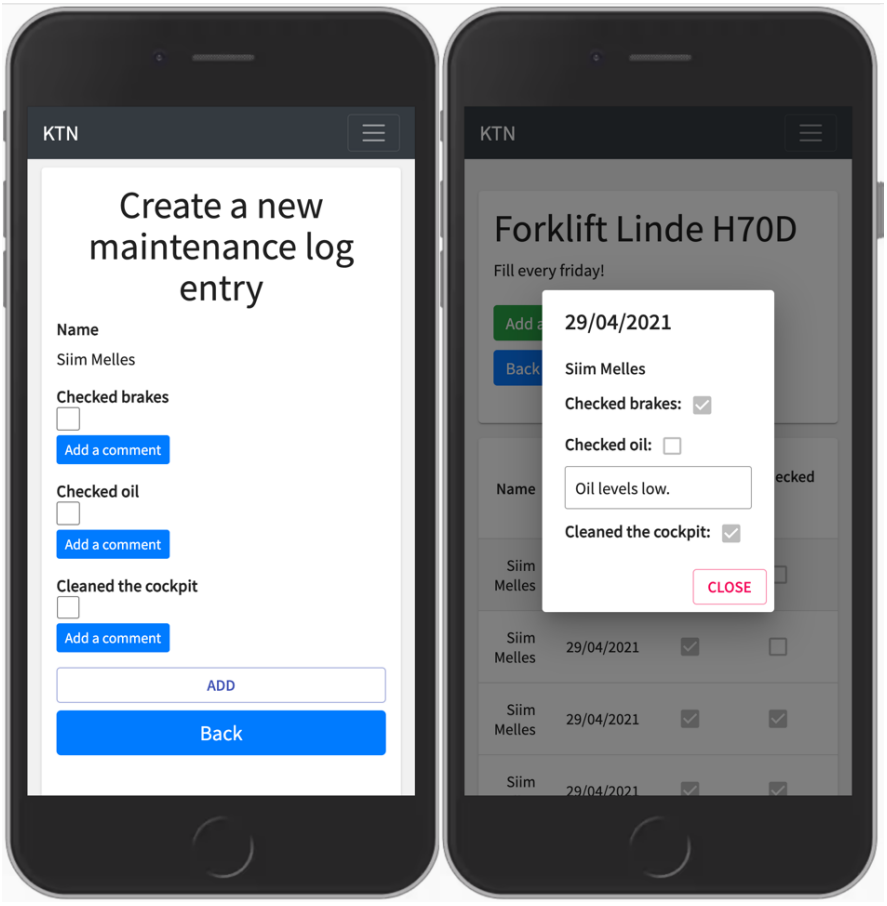
Samuti on nii navigatsiooniriba menüü kui ka esilehe kaudu võimalik liikuda laotehnika hoolduslogide lehele, kus kuvatakse kõik ettevõtte süsteemis olevad laotehnika hoolduslogid (joonis 11). Laotehnika hoolduslogi nime peale vajutades suunatakse kasutaja lehele, kus kuvatakse hoolduslogi kirjeldus ning tabel kuupäeva järgi järjestatud hoolduslogi sissekannetega (joonis 11).



Joonis 11. Hoolduslogide nimekirja ning hoolduslogi sissekannete lehed.

Laotehnika hoolduslogi lehel on ka nupp, mis viib kasutaja sissekande loomise lehele (joonis 12). Sissekande loomise lehel kuvatakse kasutajale kõik laotehnika hoolduslogi read, mis kirjeldavat hoolduse korras teostatavat kontrolli või tööd. Kasutajal on võimalik igal real märkida, kas tegevus sai tehtud, ning vajadusel on võimalik lisada kommentaar. Valiidse sissekande korral suunatakse kasutaja tagasi vastava hoolduslogi lehele.

Sarnaselt laoüksustele ei ole laotehnika hoolduslogi tabelis kommentaaride veergusid. Tabel on küll horisontaalselt keritav, kuid loetavuse huvides ning veebilehe mobiilseadmes kasutamise hõlbustamiseks on need peidetud ning kuvatakse tabeli reale vajutades hüpikaknas (joonis 12).

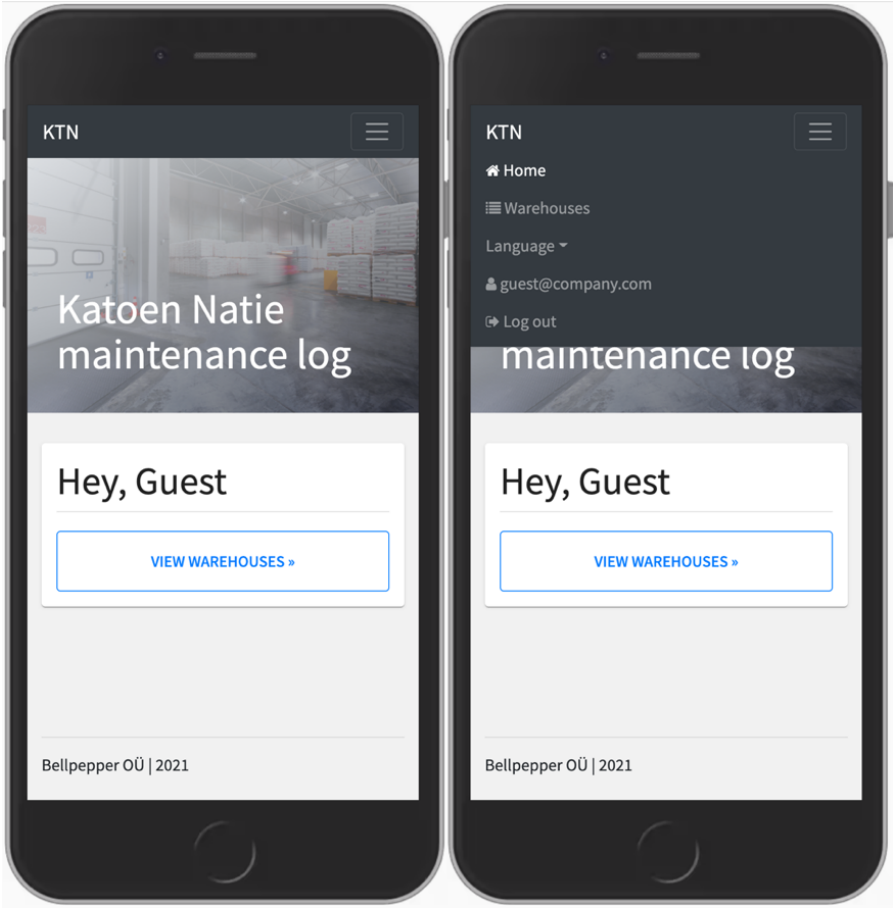


Joonis 12. Laotehnika hoolduslogi sissekande loomise leht ning sissekande detailide hüpikaken.

4.3.4 Kasutajaliides ettevõtte klientidele

Ettevõtte klientidele loodavate külaliskontodega rakendusse sisselogides on vaadetest eemaldatud paljud valikud. Nii esilehel kui ka navigatsiooniribal puuduvad lingid laotehnika hoolduslogide nimekirja ning laoüksustesse hoolduste lisamise lehtedele.

Esilehelt on võimalik navigeerida ainult ettevõtte laohooneid ning nende alla kuuluvaid laoüksuseid kuvavale lehele (joonis 13). Navigatsiooniriba kaudu on lisaks eelnevale valikule võimalik muuta lehe keelt, vaadata kasutajaprofiili andmeid ning logida rakendusest välja (joonis 13).



Joonis 13. Külaliskonto esileht ning navigatsiooniriba.

5 Tulemused

Lõputöö tulemusena loodi ettevõtte esialgsetele nõuetele vastav veebirakendus, mis koosneb kasutajaliidesena kasutatavast eesrakendusest ning funktsionaalset loogikat hõlmavast tagarakendusest, mille külge on andmete hoiustamiseks lisatud andmebaas. Rakenduse loomise lõppfaasis ning kliendile seni valminud töö näitamisel selgus, et kuigi klient oli rakendusega rahul ning see lahendas nende jaoks esimeses peatükis kirjeldatud probleemid, siis sooviks nad rakenduses veel mitmeid lisaomadusi ning funktsionaalseid võimalusi. Arvestades töö niigi laialdast skoopi, jäid need hetkel esialgse lahenduse arendamisest välja, kuid annavad võimaluse rakenduse edasiarendamiseks, et need järgmises versioonis kaasata.

Kliendi üheks funktsionaalseks sooviks on võimalus luua igale laoüksusele selles laos tegemist vajavate hooldustööde tabel, mida laoüksuse hoolduslogis kuvataks. Tabel näitaks, kui palju on kindlat liiki hooldustöid jooksval kuul tehtud ja mis on hooldustööde koguse eesmärk. Selle lisamiseks oleks vaja lisada tabeleid andmebaasi, luua vastav loogika tagarakenduses ning kuvada andmed eesrakenduses.

Samuti soovis klient otsingufunktsiooni, et kuvada laoüksustes tehtud hoolduseid kasutaja määratud filtrite põhjal. See nõuaks tagarakenduses täiendavat API lõpp-punkti ning eraldi vaadet eesrakenduses.

Viimasena tuleks nii laoüksuste kui ka laotehnika hooldamiste lehel kuvada korraga limiteeritud kogus andmeid ning jaotada need andmed eri lehtedele, mille vahel kasutaja navigeerida saaks. Tegemist ei ole suure funktsionaalse nõudega ning see puudutab ainult eesrakendust. Funktsionaalsuse lisamine aitaks parandada kasutajakogemust suure hulga andmete kuvamisel.

Lisaks funktsionaalsetele nõuetele on rakendus vaja paigaldada serverisse, et seda oleks võimalik kasutada. Selleks tuleks luua automatiseeritud pideva avalikustamise süsteem, kus koodi üleslaadimisel jooksutatakse automaatteste ning seejärel juurutatakse uuendatud rakendus serverisse. Pideva avalikustamise süsteemi loomine jäi autoril kogemuse puudumise ning töö skoopi mittemahtumise tõttu tegemata, kuid projektiga edasi minnes oleks süsteem vajalik.

6 Kokkuvõte

Lõputöö eesmärk oli luua logistikaettevõttele digitaalne hoolduslogide haldamist ja vaatamist võimaldav süsteem, mis lahendaks kasutusel oleva paberipõhise süsteemi puudujäägid, nagu näiteks andmete puudulikkus kadunud lehtede tõttu või halb loetavus segase käekirja tõttu. Hoolduslogide terviklikkus on ettevõtte jaoks vajalik nii heade kliendisuhete kui ka üldise kvaliteedi tagamiseks.

Rakenduse spetsiifilised nõuded kaardistati vestlusel ettevõtte tegevjuhiga ning rakenduse üldine nõue oli tagada sama funktsionaalsus nagu paberipõhisel süsteemil. Rakenduse analüüsi osas kirjeldati ja analüüsiti loomiseks kaalutud tehnoloogiaid ning langetati otsus, mida valida rakenduse realiseerimiseks. Rakenduse loomisel osutus autori valitud tehnoloogiapinuks C# ja .NET tagarakenduses, TypeScripti ja React.js eesrakenduses ning MySQL andmebaasina.

Praktilises osas kirjeldas autor esmalt andmebaasimudelit ning kuidas see rakenduse äriloogikas väljendub. Seejärel kirjeldas autor tagarakenduse arhitektuuri. Rakenduse valmistamiseks loodi REST API tagarakendus, mille külge loodi ka MVC mustrit kasutades administraatoriliides. Praktilise osa viimases peatükis kirjeldas autor lühidalt eesrakenduse projekti struktuuri ning kasutajaliidese ülesehitust.

Töö lõpus analüüsis autor lühidalt loodud rakenduse arendusprotsessi ning kliendi tagasisidet. Analüüsist selgus, et klient jäi rakendusega rahule ning see lahendab eelnevas süsteemis esinenud kitsaskohti. Arutelust selgus ka, et lisaks olemasolevale funktsionaalsusele sooviks klient rakendusse lisanduvaid funktsioone. Töö suure algse mahu tõttu jäid need esmasest arendusest välja, kuid annavad võimaluse rakendusega tööd jätkata. Samuti on töö edasiarendamiseks ette nähtud rakenduse juurutamine serverisse.

Lõputöö tulemusena loodi ettevõtte esialgsetele nõuetele vastav veebirakendus, millel on kasutajaliides eesrakenduse kujul ning tagarakendus koos andmebaasiga, mis hõlmab rakenduse funktsionaalset loogikat. Kuna rakendus ei ole veel paigaldatud serverisse ning kliendil esines lisasoove, siis jätkab autor rakenduse arendamist ning koostööd ettevõttega.

Kasutatud allikad

[1] Katoen Natie, „Katoen Natie Services,“ [Võrgumaterjal]. Available: https://generalcargocommodities.katoennatie.com/services/. [Kasutatud 20 04 2021].

[2] ISO, „ISO 9000 Quality Management,“ [Võrgumaterjal]. Available: https://www.iso.org/iso-9001-quality-management.html. [Kasutatud 20 04 2021].

[3] M. Rehkopf, „Epics, stories, themes, and initiatives,“ [Võrgumaterjal]. Available: https://www.atlassian.com/agile/project-management/epics-stories-themes. [Kasutatud 20 04 2021].

[4] I. Kleshnin, „How to choose a programming language?,“ [Võrgumaterjal]. Available: https://mkdev.me/en/posts/how-to-choose-a-programming-language. [Kasutatud 20 04 2021].

[5] L. Velázquez, „How to Choose the Right Backend Framework (after years of experience),“ 14 01 2021. [Võrgumaterjal]. Available: https://dev.to/levivm/how-to-select-a-backend-framework-after-years-of-experience-4fej. [Kasutatud 21 04 2021].

[6] Tutorialspoint, „Java - Overview,“ [Võrgumaterjal]. Available: https://www.tutorialspoint.com/java/java_overview.htm. [Kasutatud 21 04 2021].

[7] Scand, „Why use Java for Back-end Development?,“ 15 05 2020. [Võrgumaterjal]. Available: https://scand.com/company/blog/why-use-java-for-back-end-development/. [Kasutatud 21 04 2021].

[8] Back4App, „Top 10 Backend Programming Languages,“ [Võrgumaterjal]. Available: https://blog.back4app.com/top-10-backend-programming-languages/. [Kasutatud 21 04 2021].

[9] Wikipedia, „.NET Core,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/.NET_Core. [Kasutatud 21 04 2021].

[10] M. Chand, „What Is C#,“ 07 03 2020. [Võrgumaterjal]. Available: https://www.c-sharpcorner.com/article/what-is-c-sharp/. [Kasutatud 21 04 2021].

[11] T. DeGroat, „The History of JavaScript: Everything You Need to Know,“ 19 08 2019. [Võrgumaterjal]. Available: https://www.springboard.com/blog/history-of-javascript/. [Kasutatud 21 04 2021].

[12] O. Romanyuk, „NodeJS vs Python: How to Choose the Best Technology to Develop Your Web App's Back End,“ 14 01 2020. [Võrgumaterjal]. Available: https://www.freecodecamp.org/news/nodejs-vs-python-choosing-the-best-technology-to-develop-back-end-of-your-web-app/. [Kasutatud 21 04 2021].

[13] Stack Overflow, „Stack Overflow Trends: Java, C#, JavaScript, Python, PHP,“ 25 04 2021. [Võrgumaterjal]. Available:

https://insights.stackoverflow.com/trends?tags=java%2Cc%23%2Cjavascript%2Cpython%2Cphp. [Kasutatud 25 04 2021].

[14] Microsoft, „.NET Core/5+ vs. .NET Framework for server apps,“ 20 04 2021. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server. [Kasutatud 23 04 2021].

[15] Microsoft, „Choose between ASP.NET 4.x and ASP.NET Core,“ 12 02 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/choose-aspnet-framework?view=aspnetcore-5.0. [Kasutatud 23 04 2021].

[16] Microsoft, „.NET Framework versions and dependencies,“ 17 01 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/framework/migration-guide/versions-and-dependencies. [Kasutatud 23 04 2021].

[17] Microsoft, „Create web APIs with ASP.NET Core,“ 20 07 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-5.0. [Kasutatud 23 04 2021].

[18] Microsoft, „Overview of ASP.NET Core MVC,“ 12 02 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0. [Kasutatud 23 04 2021].

[19] M. Mendieta, „Creating Views and Controllers In a .NET Core App With ASP.NET Code Generator,“ 15 07 2019. [Võrgumaterjal]. Available: https://mapadevelopment.com/community/tutorials/generate-views-and-controller-in-dotnet-core-with-codegenerator. [Kasutatud 24 04 2021].

[20] V. Joshi, „Seven Reasons Why A Website's Front-End And Back-End Should Be Kept Separate,“ 19 07 2018. [Võrgumaterjal]. Available: https://www.forbes.com/sites/forbestechcouncil/2018/07/19/seven-reasons-why-a-websites-front-end-and-back-end-should-be-kept-separate/. [Kasutatud 24 04 2021].

[21] JavatPoint, „Difference between JavaScript and TypeScript,“ [Võrgumaterjal]. Available: https://www.javatpoint.com/javascript-vs-typescript. [Kasutatud 24 04 2021].

[22] L. Harkushko, „7 Best JavaScript Frameworks and Libraries for Front-end Web Development in 2021,“ [Võrgumaterjal]. Available: https://yalantis.com/blog/javascript-frameworks-how-to-make-your-choice/. [Kasutatud 24 04 2021].

[23] M. Nowak, „Vue vs React in 2021: Which Framework to Choose and When,“ 06 08 2020. [Võrgumaterjal]. Available: https://www.monterail.com/blog/vue-vs-react-2021. [Kasutatud 24 04 2021].

[24] H. Emekoma, „Comparing the best new JavaScript frameworks to React,“ 02 03 2021. [Võrgumaterjal]. Available: https://blog.logrocket.com/comparing-the-best-new-javascript-frameworks-to-react/. [Kasutatud 24 04 2021].

[25] G. Bakradze, „How to Choose the Best Front-end Framework,“ [Võrgumaterjal]. Available: https://www.toptal.com/javascript/choosing-best-front-end-framework. [Kasutatud 25 04 2021].

[26] Stack Overflow, „Stack Overflow Trends: ReactJS, Vue.js, Aurelia,“ 25 04 2021. [Võrgumaterjal]. Available: https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Caurelia. [Kasutatud 25 04 2021].

[27] Microsoft, „Relational vs. NoSQL data,“ 19 01 2021. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data. [Kasutatud 26 04 2021].

[28] EssentialSQL, „SQL ACID Database Properties Explained,“ [Võrgumaterjal]. Available: https://www.essentialsql.com/sql-acid-database-properties-explained. [Kasutatud 26 04 2021].

[29] GeeksforGeeks, „Introduction to NoSQL,“ 05 09 2019. [Võrgumaterjal]. Available: https://www.geeksforgeeks.org/introduction-to-nosql/. [Kasutatud 26 04 2021].

[30] D. Tobin, „Which Modern Database Is Right for Your Use Case?,“ 10 06 2020. [Võrgumaterjal]. Available: https://www.xplenty.com/blog/which-database/. [Kasutatud 26 04 2021].

[31] L. M., „SQL Database Management System: Which One Should You Choose?,“ 05 01 2021. [Võrgumaterjal]. Available: https://www.bitdegree.org/tutorials/sql-database-management-system/https://www.bitdegree.org/tutorials/sql-database-management-system/. [Kasutatud 26 04 2021].

[32] K. Hristozov, „MySQL vs PostgreSQL - Choose the Right Database for Your Project,“ 19 07 2019. [Võrgumaterjal]. Available: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres. [Kasutatud 26 04 2021].

[33] M. SmallCombe, „PostgreSQL vs MySQL: The Critical Differences,“ 26 05 2020. [Võrgumaterjal]. Available: https://www.xplenty.com/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/. [Kasutatud 26 04 2021].

[34] J. Rabelo, „What is a Primary Key?,“ 14 08 2020. [Võrgumaterjal]. Available: https://www.techopedia.com/definition/5547/primary-key. [Kasutatud 27 04 2021].

[35] M. Jones, „Integers vs GUIDs - The Great Primary Key Debate,“ 19 11 2015. [Võrgumaterjal]. Available: https://exceptionnotfound.net/integers-vs-guids-the-great-primary-key-debate/. [Kasutatud 27 04 2021].

[36] E. Pollack, „How to solve the SQL Identity Crisis in SQL Server,“ 14 11 2017. [Võrgumaterjal]. Available: https://www.sqlshack.com/solve-identity-crisis-sql-server/. [Kasutatud 27 04 2021].

[37] 8bitmen, „UUID GUID Oversimplified – Are they Really Unique?,“ [Võrgumaterjal]. Available: https://www.8bitmen.com/uuid-guid-oversimplified-are-they-really-unique/. [Kasutatud 27 04 2021].

[38] Microsoft, „CultureInfo Class,“ [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.globalization.cultureinfo?view=net-5.0. [Kasutatud 28 04 2021].

[39] P. Walpita, „Software Architecture Patterns — Layered Architecture,“ 09 07 2019. [Võrgumaterjal]. Available: https://priyalwalpita.medium.com/software-architecture-patterns-layered-architecture-a3b89b71a057. [Kasutatud 27 04 2021].

[40] Microsoft, „Three-Layered Services Application,“ 17 03 2014. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648105(v=pandp.10). [Kasutatud 27 04 2021].

[41] J. Verreckt, „Layered Architecture,“ 05 07 2018. [Võrgumaterjal]. Available: https://www.thinktocode.com/2018/07/05/layered-architecture/. [Kasutatud 28 04 2021].

[42] DevIQ, „Repository Pattern,“ [Võrgumaterjal]. Available: https://deviq.com/design-patterns/repository-pattern. [Kasutatud 28 04 2021].

[43] Microsoft, „Entity Framework Core,“ 20 09 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/ef/core/. [Kasutatud 28 04 2021].

[44] Microsoft, „Design the infrastructure persistence layer,“ 08 10 2018. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design. [Kasutatud 28 04 2021].

[45] Microsoft, „Create Data Transfer Objects (DTOs),“ 16 06 2014. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5. [Kasutatud 28 04 2021].

[46] J. Strumpflohner, „Lessons Learned: Don't Expose EF Entities to the Client Directly,“ 24 10 2012. [Võrgumaterjal]. Available: https://juristr.com/blog/2012/10/lessions-learned-dont-expose-ef-entities-to-the-client-directly/. [Kasutatud 28 04 2021].

[47] Microsoft, „dotnet aspnet-generator,“ 16 11 2020. [Võrgumaterjal]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/tools/dotnet-aspnet-codegenerator. [Kasutatud 28 04 2021].

[48] JWT, „Introduction to JSON Web Tokens,“ [Võrgumaterjal]. Available: https://jwt.io/introduction. [Kasutatud 28 04 2021].

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

Mina, Siim Melles

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Logistikaettevõtte hoolduslogi veebisüsteemi arendamine“, mille juhendaja on Kristiina Hakk
   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.

13.05.2021

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 – Nõuete kirjeldus epikute ja kasutajalugude kaudu

Epik 1: Ettevõtte hoolduslogi rakendusse sisselogimine.

- Kasutajana tahan valida süsteemis olevatest ettevõtetest enda oma, et sisse logida.
- Kasutajana tahan sisse logida e-posti aadressi ja parooli kaudu.

Epik 2: Ettevõtte ladude hooldusloglde vaatamine.

- Kasutaja/külalisena tahan näha kõiki laoüksuseid, mis ettevõttele kuuluvad.
- Kasutaja/külalisena tahan vaadata töid, mis on igas laoüksuses tehtud.
- Kasutaja/külalisena tahan näha, mis hooldusliiki tööd tehtud on.
- Kasutaja/külalisena tahan näha, mis kuupäeval hooldustöid tehtud on.
- Kasutaja/külalisena tahan näha, mis kaupa laoüksuses hoitakse.

Epik 3: Ladude hooldamiste sissekandmine.

- Kasutajana tahan lisada oma nime sissekandele.
- Kasutajana tahan lisada kommentaari sissekandele.
- Kasutajana tahan lisada kuupäeva sissekandele.
- Kasutajana tahan valida laoüksust, millele hooldamise sissekannet lisan.
- Kasutajana tahan valida hooldusliiki, mida laoüksuses teen.

Epik 4: Ettevõtte tehnika hooldusloglde vaatamine.

- Kasutajana tahan näha kõiki tehnika hooldusiogisid, mis ettevõttele kuuluvad.
- Kasutajana tahan näha sissekandeid, mis on igasse tehnika hooldusiogisse lisatud.
- Kasutajana tahan näha, mis kuupäevadel sissekanded loodud on.
- Kasutajana tahan näha sissekannetele lisatud kommentaare.

Epik 5: Tehnika hooldusiogidesse sissekannete lisamine.

- Kasutajana tahan täita logiraamatu välju (jah/ei vastus).
- Kasutajana tahan valida igale väljale kommentaari.
- Kasutajana tahan lisada oma nime sissekandele.
- Kasutajana tahan lisada kuupäeva sissekandele.

Epik 6: Administraatoriliideses ettevõtte töötajatele/klientidele kasutajakontode lisamine ja haldamine.

- Administraatorina tahan luua töötajale töötajakonto.
- Administraatorina tahan luua kliendile kliendikonto.
- Administraatorina tahan muuta ettevõtte kasutajate õiguseid.
- Administraatorina tahan inaktiveerida kasutajaid.
- Administraatorina tahan muuta töötaja/kliendi konto andmeid.
- Administraatorina tahan muuta töötaja/kliendi konto parooli.

Epik 7: Administraatoriliideses ettevõtte ladude haldamine.

- Administraatorina tahan lisada süsteemi laohooneid ettevõtte alla.
- Administraatorina tahan lisada süsteemi laoüksuseid iga laohoone alla.
- Administraatorina tahan märkida, mis tooteid laoüksuses ladustatakse.
- Administraatorina tahan eemaldada tooteid laoüksusest.
- Administraatorina tahan muuta laohoonete nimesid.
- Administraatorina tahan muuta laoüksuste nimesid.
- Administraatorina tahan eemaldada laoüksuseid süsteemist.
- Administraatorina tahan eemaldada laohooneid süsteemist.

Epik 8: Administraatoriliideses laoüksuse hoolduslogide haldamine.

- Administraatorina tahan näha kõiki sissekandeid iga laoüksuse hoolduslogist.
- Administraatorina tahan muuta laoüksuse hoolduslogi sissekande andmeid.
- Administraatorina tahan eemaldada sissekandeid laoüksuse hoolduslogist.

Epik 9: Administraatoriliideses laotehnika hoolduslogide haldamine.

- Administraatorina tahan näha kõiki sissekandeid iga seadme hoolduslogist.
- Administraatorina tahan muuta seadme hoolduslogi sissekande andmeid.
- Administraatorina tahan eemaldada sissekandeid seadme hoolduslogist.
- Administraatorina tahan muuta seadme hoolduslogi nime ja kirjeldust.
- Administraatorina tahan muuta seadme hoolduslogi ridu.
- Administraatorina tahan eemaldada seadme hoolduslogist ridu.
- Administraatorina tahan lisada seadme hoolduslogile ridu.

- Administraatorina tahan eemaldada ettevõtte süsteemist laotehnika hooldusloogisid.

Epik 10: Administraatoriliideses ettevõtte andmete haldamine.

- Administraatorina tahan muuta ettevõtte nime.
- Administraatorina tahan lisada ettevõtte süsteemi kuuluvaid laoüksuste hooldusliike.
- Administraatorina tahan muuta ettevõtte süsteemi kuuluvaid laoüksuste hooldusliikide nimesid.
- Administraatorina tahan eemaldada ettevõtte süsteemi kuuluvaid laoüksuste hooldusliike.
- Administraatorina tahan lisada ettevõttes ladustatavaid tooteliike.
- Administraatorina tahan muuta ettevõttes ladustatavate tooteliikide nimesid.
- Administraatorina tahan eemaldada ettevõttes ladustatavaid tooteliike.