TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Markus Kildemaa 175949IADB

Laotarkvara loomine väikeettevõttele

bakalaureusetöö

Juhendaja: Meelis Antoi
Magistrikraad

Tallinn 2023

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: Markus Kildemaa

06.01.2023

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on autori tööandja poolt esitatud kriteeriumite järgi realiseerida olemasoleva laotarkvara puuduvad funktsionaalsused ettevõttele Attila. Olemasolevaks laotarkvaraks on arvutiprogramm nimega Noom. Noom on Windows operatsioonisüsteemil töötav programm lao- ja müügihalduse tarbeks.

Lõputöö kirjeldab kitsenduste lahendamiseks loodud tarkvara arenduskäiku. Kõigepealt seletatakse lahti probleemi püstitus, seejärel võrreldakse erinevaid lahendusvariante, kirjeldatakse valitud lahenduse arendus kulgu ning viimasena analüüsitakse loodud tarkvara kasutegurit ettevõttes.

Lõputöö on kirjutatud Eesti keeles ning sisaldab teksti 34 leheküljel, 9 peatükki, 17 joonist, 2 tabelit.

Abstract

Creation of inventory management software for small company

The purpose of this thesis is to implement a solution for a small company called Attila to fulfill the shortcomings of the stock software at hand according to the criteria marked down by the employer of the author. Stock software at hand is a computer program called Noom. Noom is a program working on a Windows operating system used to manage stock and sales in a company.

This thesis is describing the development of a software created to implement the missing functionality according to given criteria. Firstly the problem is specified, then different solution variants are compared after which the selected solution's development processes are described. Finally the paper analyzes the changes in the company's workflows' efficiency after commission of the new software.

The thesis is in Estonian language and contains 34 pages of text, 9 chapters, 17 figures, 2 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* ehk rakendusliides [1]
Eesrakendus	klient-server arhitektuuris klientrakendus [2]
ERP	*Enterprise resource planning* ehk ettevõtte ressursside planeerimine [3]
HTTP	*HyperText Transfer Protocol* ehk hüperteksti edastusprotokoll [4]
IP aadress	*Internet Protocol address* ehk arvutivõrgus oleva seadme identifikaator [5]
JSON	*JavaScript Object Notation* ehk lihtsustatud andmevahetusvorming, mis põhineb JavaScript-i programmeerimiskeele alamhulgal [6]
JWT	*Json Web Token* ehk standardile RFC-7519 vastav sõnum turvaliseks andmetevahetuseks [7]
MVP	*Minimum Viable Product* ehk minimaalne töötav toode [8]
SSO	*Single Sign-On* ehk ühekordne sisselogimine ühte või rohkematesse rakendustesse [9]
Tagarakendus	klient-server arhitektuuris serverrakendus [10]
URL	*Uniform Resource Locator* ehk interneti ressursi aadress [11]

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Ladu hoidvad jae- ja hulgimüügiga tegelevad ettevõtted koostavad maailma majandusest märkimisväärse protsendi. Konkurents on tihe, lisaks turundusel, kauba hinnal ja kvaliteedil, on suur roll ka klienditeenindusel ning tarnimise kiirusel, milledele aitab suuresti kaasa tellimuste korrektne haldus ja lao täpne inventuuri seisu hoid, mis hoiab ära potentsiaalsed probleemid tellimuste täideviimisel.

Käesolev lõputöö käsitleb probleemi, kus autori tööandja metallilõikeriistade jae- ja hulgimüügi ettevõttes, nimega Attila, kasutuses oleval ERP *(Ettevõtte ressursside planeerimise)* tarkvaral, nimega Noom, esinevad teatud puudused, mis pärsivad edukat lao- ja müügitegevusega seotud protsesside toimimist, sealhulgas lao inventuuri tegemiseks kasutajasõbraliku liidese puudumine, olematu sissetulevate ning väljaminevate saadetiste haldamine, piiratud programmi liidestamise võimekus ettevõttes olemasolevate tarkvaradega ning kulleripakkide koostamise ja jälgimisega seotud automaatika puudumine.

Antud lõputöö tulemusena loob autor tema tööandja poolt ette antud kriteeriumeid arvestades uue tarkvara.

Antud lõputöö kirjeldab arendusprojekti läbi kaheksa peatüki: sissejuhatus, olemasoleva programmi tutvustus, uue tarkvara nõuded, uue tarkvara valimine, tarkvara arenduse juhtimine, tarkvara arhitektuur, tarkvarade rakendused, tulemus ja kokkuvõte.

Olemasoleva programmi tutvustuse peatükis kirjeldatakse Noom programmi. Uue tarkvara nõuete peatükis on defineeritud kriteeriumid, millele peab uus tarkvara vastama ning Noom programmi vastavused nendele nõuetele. Järgmises peatükis kirjeldatakse erinevaid variante antud kriteeriumite täitmiseks uue tarkvara osas. Tarkvara arenduse juhtimise peatükis on selgitatud erinevad arenduse administreerimisega seotud tegevused. Tarkvara arhitektuur peatükis on välja toodud tarkvara ülesehituse omadused. Tarkvara rakendused peatükis kirjeldatakse laotarkvara arenduse käigus

loodud rakendusi. Tulemus peatükis kirjeldatakse käesoleval hetkel valminud tarkvara ning vaadeldakse selle kasutegurit ettevõttes.

2 Olemasolev programm

Ettevõttes Attila kasutusolev laotarkvara Noom on alates 2008 aastast Astro Baltics OÜ käest renditud Windows programm, mida majutatakse lokaalselt Attila kontoris asuvas serveris.

Noom on lao- ja majandustarkvara, mis võimaldab hallata toodete seisu laos, luua tellimustega seonduvaid dokumente, hoida klientide kontaktandmeid ning kasutada raamatupidamisega seotud operatsioone.

3 Uue tarkvara nõuded

Tabel 1: Uuelt tarkvaralt nõutud funktsioonid.

Kriteerium	**Noom vastavus**
Andmete ja programmi vaadete kiire kuvamine. Iga baastasemel funktsioon ja vaade peavad saama täide viidud vähemalt 500 millisekundi jooksul.	Programmi funktsioonid võtavad sageli aega rohkem kui 500ms.
Laoartiklite lisamine, lugemine, muutmine ja kustutamine.	Võimaldab.
Laoartiklite tabeli filtreerimine kategooria järgi.	Ei võimalda.
Laoartikli kategooria lisamine, lugemine ja muutmine.	Ei võimalda.
Laoartikli hinna ja tarnija lisamine, lugemine ja muutmine.	Võimaldab.
Laoartiklile triipkoodide lisamine ja kustutamine piiramatus koguses.	Triipkoodide lisamine laoartiklile on piiratud kolmele ühikule.
Laoartiklile lao asukohtade lisamine ja kustutamine piiramatus koguses.	Igale laoartiklile saab määrata ainult ühe asukoha.
Toodete automaatselt tellimine tarnijatelt.	Täies ulatuses testimata, tulemus teadmata.
Klientide andmete lisamine, lugemine, muutmine ja kustutamine.	Võimaldab.
Programmi kasutamine väljaspool Attila kontorit.	Võimaldab.
Programmi kasutamine Windows, IOS, MacOS, Linux ja Android seadmetega.	Piiratud Windows operatsioonisüsteemiga seadmetele.
Funktsioonide kasutamise piiramine vastavalt sisselogitud kasutaja rolli õigustele.	Võimaldab.

Kriteerium	**Noom vastavus**
Saadetiste lisamine, lugemine, muutmine ja kustutamine.	Ei võimalda.
Saadetiste tabeli filtreerimine Euroopa Liidu liikmesriikide sisese liikumise järgi.	Ei võimalda.
Skaleeritavuse võimekus ehk uute funktsioonide lisamine.	Võimaldab osaliste piirangutega.
Toodete laoseisu haldamine mobiilse seadmega.	Võimaldab teatud piirangutega lisateenuse tellimisel.
Kullersaadetiste genereerimine Omniva, UPS, DHL ja DPD andmebaasides.	Ei võimalda.
Kullersaadetiste kleebiste printimine.	Ei võimalda.
Kullersaadetise jälgimise info lugemine sealhulgas kullerteenuste Omniva, UPS, DHL ja DPD.	Ei võimalda.
Liidestus Attilas kasutusesolevate programmidega.	Võimaldab osaliste piirangutega.

4 Uue tarkvara valimine

Uuele laoprogrammile etteantud kriteeriumite täitmiseks on tarvis valida variant juba maailmas pakutavate laotarkvarade või uue programmi loomise vahel. Võimalus on samuti Noom programmi pakkuva ettevõtte arendajatega koostöös tellida uute funktsioonide arendust Noom-ile kuid see teenus tuleb etteantud kriteeriumite raames kokku majanduslikult kulukam kui uue tarkvara loomine. Põhjus on, et käesolev Noom programm ei ole loodud hea skaleeritavuse võimekusega, mis tähendab, et lisaarenduste hind on suur. Samuti on seniste lõputöö autori tööandja kogemuste põhjal ette tulnud Noom-i uute arendusteenuste järel olukordi, kus peale mõne uue funktsiooni lisamist Noom programmile on mõni eelnev võimekus lakanud töötamast või esinenud töös vigu. Kasutusesolev Noom programm on üles ehitatud vanade tehnoloogiate peal, mis ei võimalda täita ka koos lisaarendustega ühte kriitilist kriteeriumit milleks on programmi kasutusvõimekus seadmetega, mis ei kasuta Windows operatsioonisüsteemi.

Lõputöö autori tööandja visioon on tulevikus luua üks ühtne ettevõttesisene tarkvarasüsteem mis kombineerib kokku erinevate autori poolt programmeeritud tagarakenduste kasutamise ühe eesrakendusega, seega otsustati kriteeriumite täitmiseks luua uus tarkvara, mida on võimalik kujundada ka tulevikus vastavalt tööandja soovidele. Selle tõttu otsustati maailmas juba olemasolevaid laotarkvarasid mitte kasutusele võtta, sest nende arenduste üle puudub tööandjal piisavalt suur kontroll.

5 Tarkvara arenduse juhtimine

Järgnevalt vaatleme laotarkva arendusega seotud juhtimisprotsesse.

5.1 Arendustsüklid

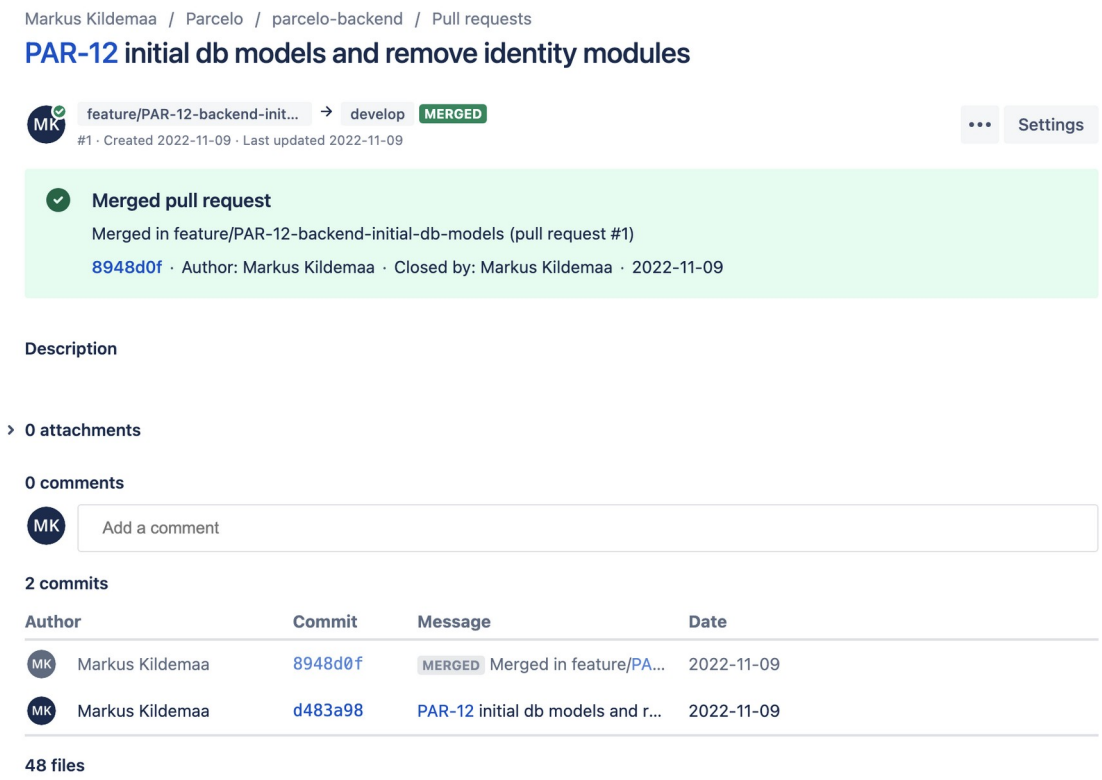
Arenduskäik toimub agiilsel Scrum meetodil arendustsüklite läbi [12], [13], milleks on käesolevas projektis tavaliselt ühe kuni kolme nädalased intervallid mida kutsutakse sprintideks. See võimaldab tuua programmi kasutusvõimalust lõppkasutajatele, antud projektis ettevõtte Attila töötajatele, kiiremalt läbi iteratsioonide, lisades järk järgult uusi funktsioone. Esialgne tarkvara luuakse MVP *(Minimum Viable Product)* ehk minimaalne toimiv programm, mis sisaldab vähimas koguses funktsioone [14], millega saavad ettevõtte töötajad koheselt tarkvara kasutama hakata juba enne kui lõppprodukt valmis on. See võimaldab arenduse käigus saada lõppkasutajatelt tagasisidet ning vajadusel selle järgi kohandada järgmistesse arendustsüklitesse ülesandeid.

5.2 Ülesanded

Arenduskäik on kaardistatud veebikeskkonna JIRA abil. Projektijuhid, kes antud projektis on lõputöö autor ning tema tööandja, koostavad tarkvara lõpptulemuse funktsioone silmas pidades ülesande kaardid, mis lisatakse JIRA veebilehel Backlog nimelisesse nimekirja. Projektijuhid liigutavad iga arendustsükli eel selleks perioodiks valitud ülesanded koos vastava tsüklile määratud versiooninumbriga Kanban nimelise stiiliga ülesannete seinale. Igale ülesandele märgitakse selle täideviija. Arenduse käigus ülesande staatuse muutudes liigutatakse see vastavasse tulpa sõltuvalt kas ülesanne ootab tegema hakkamist, on parasjagu tegemisel, ootab ülevaatamist, ootab testimist või on tehtud.

5.3 Tarkvara koodi haldamine

Programmide koode hoitakse versioonihalduse git repositooriumides ehk koodikogumites kasutades veebiteenust Bitbucket. Iga programmi jaoks on loodud oma repositoorium, mis omakorda koosneb puuoksade kuju kombel harudes. Iga haru hoiab eraldi olukorra jaoks eraldi programmi koodi seisu. See on selleks vajalik, et iga uue arendustsükli järel on vaja enne lõppkasutajatele programmi kasutamiseks andmist inimkäe läbi testida programmi funktsioonid üle, et vältida potentsiaalsete programmi vigade avaldamist. Selle tarbeks on olemas iga arenduse ülesande jaoks oma haru, mille nimeks on ülesande identifitseerimiskood koos nimega ühendatud sidekriipsuga näiteks PAR-12-initial-api-endpoints. Selline ülesannete harudeks jaotamine on vajalik, et vältida erinevate uute funktsioonide paralleelselt arendamisega programmikoodis konfliktseid kattuvusi, millega võib lihtsalt tekkida programmi jooksutamisel sisse vigu. Ülesande raames kirjutatud kood liigutatakse valmis saamise järel ülesande harust arenduse harusse, mille nimeks Develop. Kui arendustsükli raameks planeeritud funktsionaalsused on valmis arendatud ja arenduse harusse liigutatud, testitakse need test programmis läbi, peilides välja kõikvõimalikud vead. Vea tuvastades koostatakse sellest ülesannete seinale uus kaart mis lahendatakse tavapärase arenduskäiguga. Kui testimisel vigasid ei tuvastata, liigutatakse programmikood arenduse harust Master nimega põhiharusse, mille tulemusena saavad lõppkasutajad põhiprogrammis hakata kasutama viimase arendustsükli käigus loodud funktsioone. Kui põhiprogrammis tuvastab keegi lõppkasutaja vea, koostatakse sellest uus ülesanne ning liigutatakse see järgmise arendustsükli ülesannete seinale lahendamiseks.



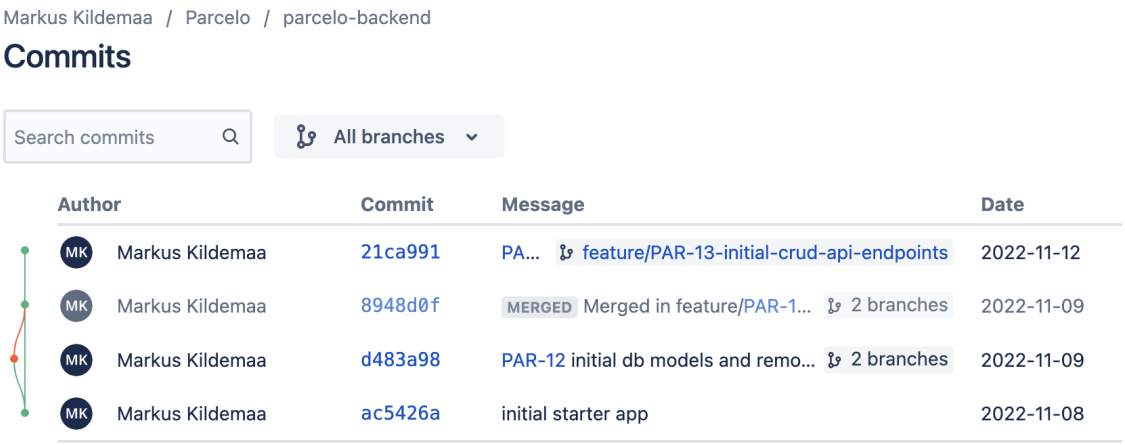
Joonis 1: Haru kombineerimise näide Bitbucket veebiliideses.

Tihtilugu kasutatakse standardmeetodi järgi lisaks ka veel enne põhiharuga kombineerimist iga arendustsükli jaoks eraldi versiooniharu ning kokkuvõtvat eelavalikku haru nimega Staging [15], kuid antud projektis ei ole nende käsitlemisega seotud ajakulu projektijuhtide silmis otstarbekas.

5.4 Tarkvara koodi valideerimine

Kui ülesanne koosneb koodi kirjutamisest, siis ülevaatamine toimub code review meetodil, kus teine arendaja vaatab ülesande raames programmeeritud koodi üle ning kui leiab et on vaja teha muudatusi, lisab vastava kommentaari, mille järel peab ülesande täideviija tegema koodis vajalikud parandused ning esitama uuele ülevaatusele. Kui ülevaataja silmis kõik sobib, siis märgib ta ülevaatuse lehel selle heakskiidetuks, mille järel saab täideviija arendaja koodi liigutada ülesande harust arenduse harusse. Käesolevas projektis on hetkeseisuga ainult üks arendaja seega vaatab antud juhul erandkorras ülesande arendaja omaenda koodi ise üle. Ühe arendajaga on võimalik kiiremaks arenduseks jätta ülevaatamise etapp vahele, kuid antud juhul

tehakse kõik toimingud siiski samamoodi nagu mitme arendajaga, põhjusel et uue arendaja liitumisel saab arendust jätkata sarnaselt ilma töövoogu ümber tegemata ning igast arendustsüklist on samasuguse mustriga jäänud maha jälg repositooriumi ajalukku, mis võimaldab vajadusel ühtset tagasivaadet programmikoodi muudatustele iga ülesande kohta eraldi sõltumata arendajate hulgast projektis erinevatel ajajärkudel.



Joonis 2: Programmikoodi muudatuste ajaloo näide Bitbucket veebiliideses.

5.5 Tarkvara koodi jooksutamine

Et koodist saada opereeritav programm, on vaja seda kuskil arvutiseadmes jooksutada. Käesolevas projektis jooksutatakse kõik taga- ja eesrakendused ühes DigitalOcean virtuaal-pilveserveris. Kaaluti ka varianti paigaldada selleks Attila kontorisse eraldiseisev serverimasin, kuid sellise variandiga kaasnevate suurenenud hooldamisega seotud kohustuste tõttu otsustati selle vastu. DigitalOcean pakub pilveserveri rentimise võimalust, kus selle riistvaraga seonduvad toimingud teostab DigitalOcean.

Virtuaalserveris kasutatakse automaatikat, mis jälgib tarkvara repositooriumides muudatuste kulgu. Kui automaatika tuvastab koodis mõne muudatuse, pakib ta uue koodi käivitatavaks programmiks kokku ning serveerib URL lingi kaudu kättesaadavaks tarkvarateenuseks. Programmid jooksevad virtuaalserveris isoleeritud Docker konteinerites [16]. Kasutades Docker konteinereid, on võimalik serveris luua nii serverist endast, kui ka üksteisest sõltumatud nii öelda alamserverid, mis saavad selle sees jooksvate programmide toimimiseks vajalikud teenuste ressursid installeerida

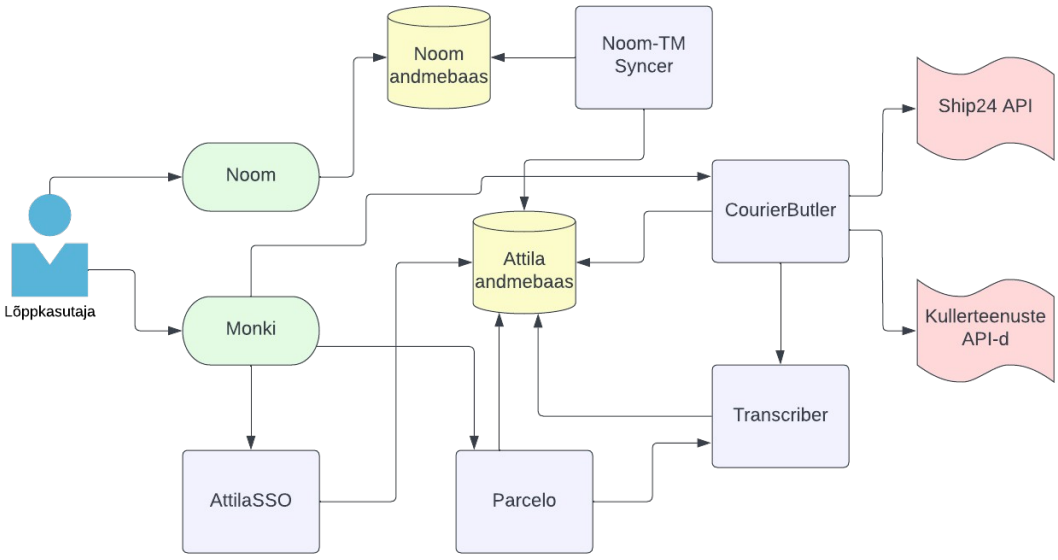
iseseisvalt iseendale. See võimaldab programmi komplektide kiiret ja mobiilset jooksutamist erinevates masinates ilma lisategevusteta inimese poolt.

Jooksutatud taga- või eesrakendust saab teine rakendus või inimene kasutada läbi URL-i, kasutades virtuaalserveri IP *(Internet Protocol)* või serveris konfigureeritud nimelist aadressi.

6 Tarkvara arhitektuur

Tarkvara luuakse klient-server arhitektuuriga, kus andmed liiguvad üle veebi erinevate tagarakenduste ja eesrakenduse vahel. Lõppkasutaja, antud juhul ettevõtte töötajad, opereerivad oma toimingute jaoks eesrakendusega.

Laotarkvara koosneb ühest eesrakendusest, nimega Monki ja viiest tagarakendusest Parcelo, Transcriber, CourierButler, AttilaSSO ja Noom-TM Syncer. Monki rakenduses sisselogimiseks ja tagarakendustes sisselogimise info valideerimiseks kasutatakse autentimise rakendust AttilaSSO.



Joonis 3: Tarkvarasüsteemi skeem.

Tarkvara andmebaasiks valiti ettevõttes olemasolev MySql versioon nr. 8 platvormiga andmebaas. MySql on vabavaraline andmebaasisüsteem, mis kasutab SQL *(Structured Query Language)* programmeerimiskeelt.

Kõik tagarakendused haldavad andmeid andmebaasis kasutades SQL päringuid. Rakendused suhtlevad üksteisega kasutades REST API *(Representational State*

Transfer Application Programming Interface) meetodit. Selle tarbeks on tagarakendustesse programmeeritud lõpp-punktid [17], nii öelda aknad, mille kaudu saab rakendusse saata ning küsida infot, mille abil teeb rakendus vastavad operatsioonid. Eesrakendus Monki manageerib andmebaasi andmeid ainult läbi tagarakenduste ilma otse andmebaasiga suhtlemiseta. Selline variant hoiab eesrakenduse abstraktsena ja sõltumatuna andmebaasi platvormist ja arhitektuurist.

Kõik rakendused on majutatud ühes pilveserveris, et rakenduste omavaheline suhtlus oleks maksimaalselt kiire ning et minimaalne arv API lõpp-punkte oleksid serverist väljastpoolt kättesaadavad, mis vähendab potentsiaalsete küberrünnakute riske. Kõik API lõpp-punktid, mida eesrakendus ei kasuta on piiratud ainuüksi serveris lokaalselt kasutamiseks. Samamoodi on andmebaasi andmetele ligipääs limiteeritud serveris asuvatele rakendustele.

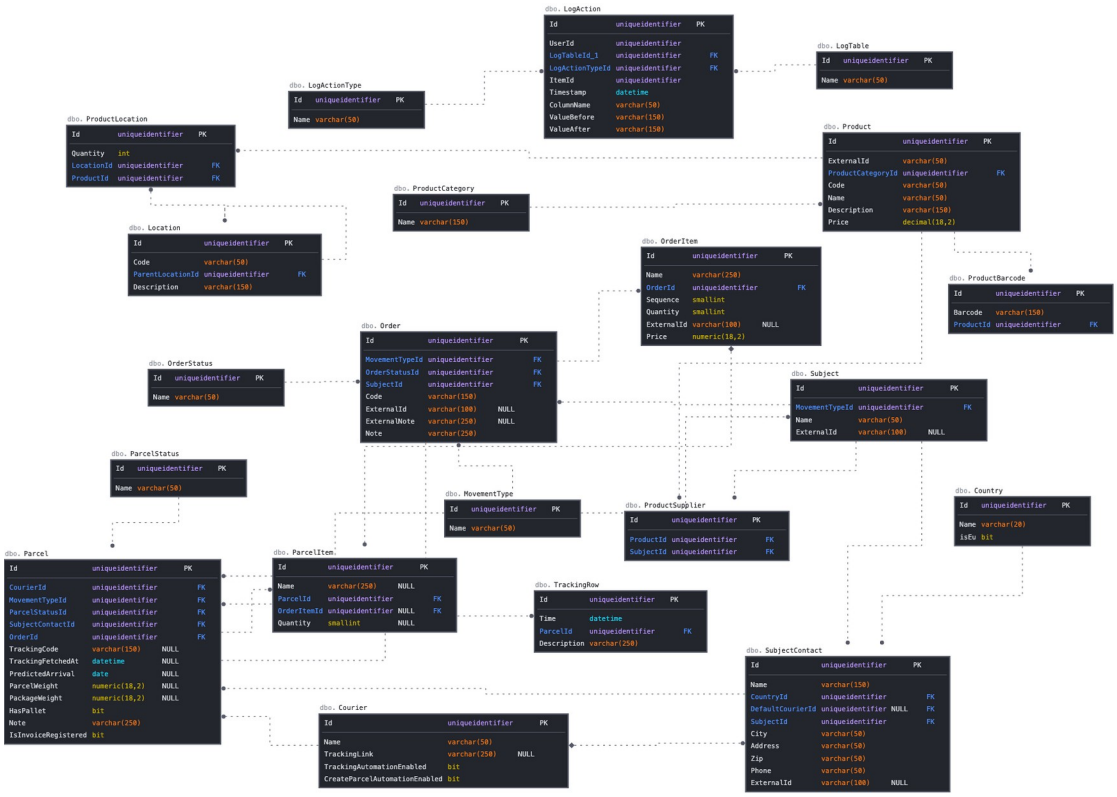
Tarkvara rakendused on loodud olema üksteisest võimalikult sõltumatud [18]. See tähendab, et igat rakendust on võimalik eraldi välja vahetada mõnes teises programmeerimiskeeles ja raamistikus loodud rakenduse vastu või jooksutada teises serveris ilma teistes rakendustes muudatusi tegemata. Rakenduste omavaheliseks suhtlemiseks vajalik info, näiteks API lõpp-punktide aadressid, hoitakse igas rakenduses eraldi ühes standardses failis, mille andmete väärtusi saab defineerida serveris kõigile rakendustele korraga ühes docker-compose nimelises failis. Selline tegumood muudab kogu tarkvarasüsteemi rakenduste kompoti piisavalt mobiilseks, et rakenduste väljavahetamine ja serveri vahetamise protsess võtab märkimisväärselt vähem aega kui igal rakendusel eraldi väärtusi hallata ning samuti vähendab inimvea faktorit.

6.1 Andmebaas

Andmebaas on käesolevas projektis korrastatud infokogum, kus andmed on jaotatud tabelitesse, kus veerud on defineeritud info nimetuse ja andmetüübi järgi. Andmed esinevad tabelites ridadena piiratult iga veeru andmetüübile. Igale tabelile on loodud primaarvõtme veerg nimega ID, millele on lisatud unikaalsuse nõue, mis tähendab, et iga tabelisse lisatud kirje ID väärtus ei tohi kattuda olemasolevaga [19]. See aitab vältida duplikaat andmete tekkimist ning kirjetele viitamine on lihtsustatud. Kui mõne tabeli andmete kirjed on seotud teise tabeliga, luuakse tabelile välisvõtme veerg, mis

viitab teise tabeli primaarvõtme veerule. Juhul, kui mõne tabeli rea seos teise tabeli rea suhtes on mitu-mitmele, kus mõlema tabeli rea suhtes on neil omavahel mitu seost, luuakse selle tarbeks uus vahetabel, kus defineeritakse mõlemale tabelile viitamiseks eraldi välisvõtme veerud. Näiteks kui on olemas laoasukohtade tabel ja toodete tabel ning üks toode võib laos asuda mitmes kohas korraga. Selle tarbeks luuakse vahetabel, kuhu lisatakse iga toote jaoks eraldi kirje koos sellele vastava asukohaga laos viidates välisvõtmete abil nii toote tabeli kui ka laoasukoha tabeli spetsiifilisele reale.

Kõige efektiivsem viis andmete arhitektuuri projekteerimiseks, on disainida arenduse algfaasis tabelite skeem, kus on välja joonistatud kõik tabelid, veerud ja tabelite vahelised seosed. Antud projektis on selle jaoks kasutusel veebiliides nimega SqlDBM [20].



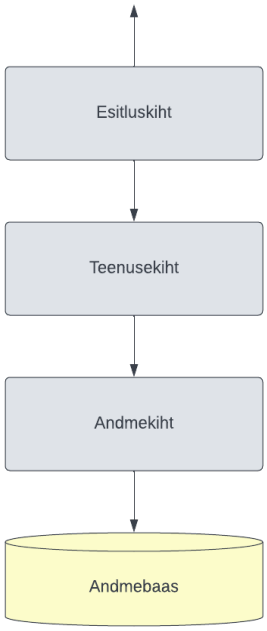
Joonis 4: Andmebaasi skeem veebiliideses SqlDBM.

Käesoleva laotarkvara poolt kasutuses olev andmebaas, serveeritakse DigitalOcean pilveserveris [21] suletud Docker konteineris, millele saavad ligi ainult samas serveris jooksutatud rakendused. Kui arendajal on tarvis pääseda käsitsi ligi andmebaasi andmetele, jooksutatakse ajutiselt selle tegevuse perioodiks serveris PhpMyAdmin

graafiline liides [22], millega pääseb koos kasutajanime ja parooliga sisselogimisel andmeid manipuleerima.

6.2 Tagarakenduse kihid

Tagarakenduste loogikad on projektis igaüks organiseeritud mitme abstraktse kihi vahel. Selline meetod hoiab programmi koodi erinevad funktsioonide kogumikud üksteisest lahus, nii on kood arendajate jaoks paremini taaskasutatav [23], arusaadav ja navigeeritav. Analoogiks saab näitena tuua riiete organiseerimise kapis, kus särgid ja sokid on jaotatud eraldi sahtlitesse.

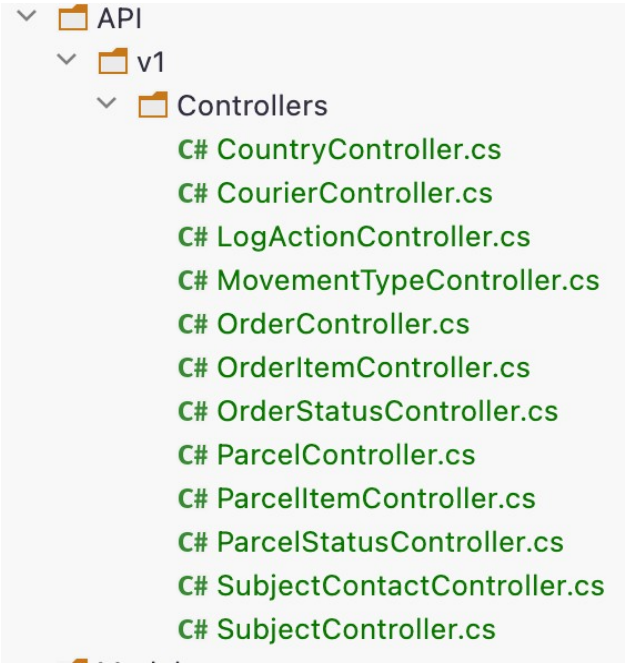


Joonis 5: Rakenduse kihid.

Andmekihis asuvad domeeniobjektid, millega saab andmebaasi andmeid käsitleda koodis kui objektidena [24].

Teenusekihis on kirjeldatud kõik rakenduse äriloogika ja kihtidevahelised suunamised. Selles kihis on kirjutatud kõik programmile iseloomulikud funktsionaalsused ja loogikad näiteks andmetega kalkuleerimine, e-maili saatmine jne. Teenusekiht on kui vahendaja rollis andmekihi ja estitluskihi vahel, suunates esitluskihist saabuvad andmete küsimise päringud andmekihi pihta.

Esitlusloogika kihis asuvad kontrollerid, kuhu on kirjutatud rakenduse API lõpp-punktid, mis võtavad vastu HTTP päringuid GET, POST, PUT ja DELETE meetoditega [25] koos päringu sees asuvate JSON *(JavaScript Object Notation)* formaadis andmetega. Päringute vastuses sisaldavad andmed tagastatakse samuti JSON formaadis.



Joonis 6: Kontrollerifailide näide Parcelo tagarakenduses.

Kontrolleri faili sees kasutatakse ASP.NET kontrolleri teegi, nimega Microsoft.AspNetCore.Mvc, annotatsioone, mis hõlbustavad ilma lisa koodi kirjutamiseta kaardistada API URL aadresse, mille vastu päringuid saates käivitatakse funktsioon, mille kohale on koodis vastav annotatsioon kantsulgusid kasutades kirjutatud. Näiteks v1 nimega kaustas asuv SubjectController failis annotatsioon [Route("api/v{version:apiVersion}/[controller]")] kaardistab kõik selle klassi sees olevad funktsioonid URL aadressile, mis algab domeeninimega, näiteks samas seadmes lokaalselt jooksutatud programmi puhul https://localhost:7069, kus 7069 tähistab porti, mille vastu programm on seadistatud seadmes jooksma ning mille järel on tekst /api/v1/Subject. Annotatsiooni sees asuv tekst {version:apiVersion} tähistab kontrolleri versiooni, antud juhul v1. Versiooni numbri kaardistamist versioonile v1 aitab SubjectControlleri failis leiduv annotatsioon [ApiVersion("1")]. Viimases URL

aadressi näites kirjeldatud tekst Subject on samuti automaatselt ASP.NET teegi poolt kaardistatud, kasutades annotatsioonis asuvat teksti [controller].

```
namespace ParceloApp.API.v1.Controllers
{
    [ApiVersion( "1" )]
    [Route( template: "api/v{version:apiVersion}/[controller]")]
    [ApiController]
    1 usage
    public class SubjectController : ControllerBase
    {
        private readonly ApplicationDbContext _context;
        private readonly IMapper _mapper;

        public SubjectController(ApplicationDbContext context, IMapper mapper)
        {
            _context = context;
            _mapper = mapper;
        }

        // GET:
        [HttpGet]
        public async Task<ActionResult<IEnumerable<App.DAL.DTO.v1.Responses.Common.Subject>>> GetAll()
        {
            var data :List<Subject> = await CommonSubjectDbSet().ToListAsync();
            var res :IEnumerable<Subject> = data
                .Select(subject =>
                    _mapper.Map<App.DAL.DTO.v1.Responses.Common.Subject>(subject));
            return Ok(res);
        }
```

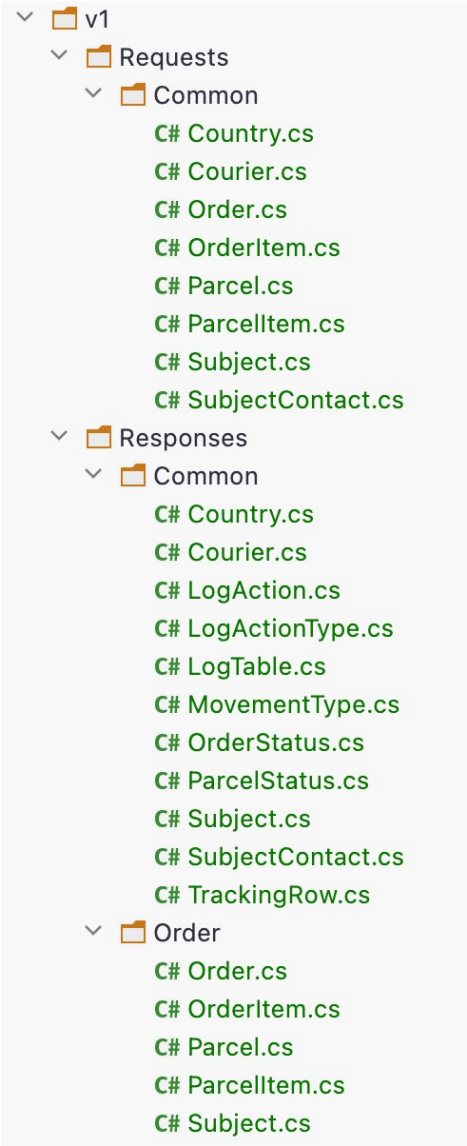
Joonis 7: SubjectController faili sisu näide Parcelo tagarakenduses.

Kontrollerite versioneerimine on vajalik vältimaks vigu, kus mõni teine programm kasutab antud rakenduse API kontrolleri vastu päringute tegemist, milles on hilisemas staadiumis tehtud muudatusi, mille tulemuseks on teise programmi poolt oodatud väljunditest erinev vastus. Sellisel juhul saame peale kontrolleris muudatuste tegemist lisada sellele teistsuguse versiooni numbri enne kui teised rakendused on kohandatud neid muudatusi arvestama. Senikaua kasutab teine programm oma tegevusteks eelmist kontrolleri versiooni. Sellist meetodit kasutades peame kontrollerites säilitama vanadele versioonidele kirjutatud funktsioone nii kaua kuni teised programmid neid veel kasutavad. Lihtsamaks versioonide haldamiseks on kontrolleri failid organiseeritud versiooni numbri järgi eraldi kaustadesse.

Vältimaks kontrollite poolt liigse info tagastamist kasutajale, on programmis loodud esitlusloogika kihis selle tarbeks eraldi objektid, kus on defineeritud ainult need

atribuudid mille väärtusi on tarvis kontrolleritel API päringute vastustes tagastada. See muuhulgas väldib andmebaasis asuvat sensitiivset infot piirata kasutajale vaatamiseks. Objektide failid on sarnaselt kontrollerite failidele organiseeritud versiooni järgi kaustadesse. Väärtuste objektideks ümber konverteerimiseks kasutavad kontrollerid AutoMapper teeki.

Sarnaselt on esitlusloogika kihis kasutuses ka API päringute vastuvõtmise andmekujude defineerimiseks eraldi objektid versiooni numbrite järgi kaustadesse jaotatud Request kaustas.



Joonis 8: Näide API-ga seotud andmekujude failidest Parcelo tagarakenduses.

```
namespace App.DAL.DTO.v1.Responses.Common;

6 usages
public class Subject : DomainEntityId
{
    public MovementType MovementType { get; set; } = null!;
    public string Name { get; set; } = default!;
    public string? ExternalId { get; set; }

    public ICollection<SubjectContact>? SubjectContacts { get; set; }
}
```

Joonis 9: API poolt tagastatava andmekuju objekti näide Parcelo tagarakenduses.

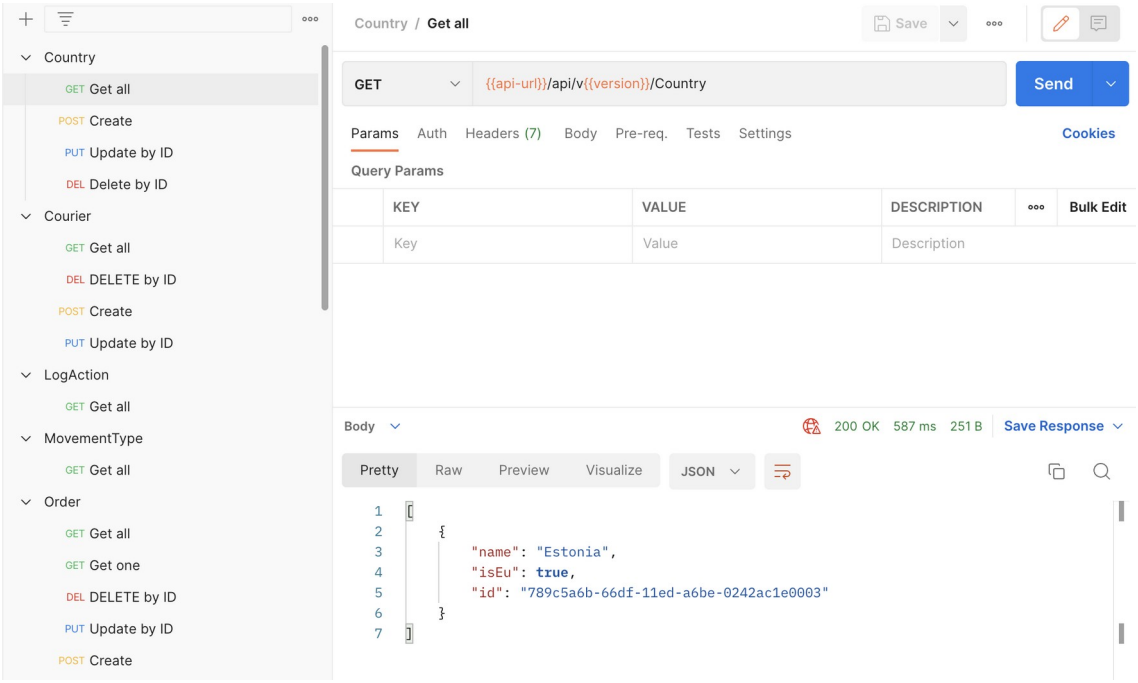
Standardlahenduse järgi kasutatakse API päringutes GET meetodit andmete küsimiseks. Näiteks kõiki andmebaasis talletatud kullerite andmete saamiseks saadab mõni teine programm või inimene käsitsi HTTP päringu GET meetodiga aadressile sufiksiga /api/v1/Courier. Selle tulemusena käivitub kontrolleris asuv funktsioon GetAll, mille kohale on märgitud annotatsioon [HttpGet]. See annotatsioon kaardistab funktsiooni vaikimisi samale URL aadressile, mis on funktsiooni klassi kohal kirjutatud annotatsioonis. Funktsioon GetAll käivitab seejärel teenusekihis asuva kulleritega seonduva teenuse kindla funktsiooni, mis on kontrolleri funktsioonis paika pandud. Teenuse funktsioon kasutab andmekihti jooksutamaks vajalikud andmebaasiga seotud käsklused, mille tulemusena saab andmebaasist kätte nõutud andmed. Antud näite puhul käivitatakse CourierService nimega teenuse failis GetAllCouriers funktsioon, mis kasutab andmekihis asuvat domeeniobjekti nimega Courier ning käivitab EntityFramework funktsiooni ToListAsync, mis tagastab kõik andmebaasis asuvad kullerid. Seejärel tagastab teenus saadud kullerite domeeniobjektide nimekirja kontrollerile. Kontroller konverteerib saadud domeeniobjektid tagastatavateks objektideks ümber ning serveerib need API päringut tegevale kasutajale JSON formaadis.

Sarnaselt eelmisele näitele käituvad ka API meetodid PUT, DELETE ja POST. PUT kasutatakse standardlahendusena andmebaasi andmete muutmise päringutel, DELETE andmerea kustutamisel, ja POST uue andmerea lisamisel. POST API päringute

meetoditel kasutatakse kaasa antud andmete kuju defineerimiseks selle tarbeks loodud andmeobjekti, mis sisaldab nõutud atribuute ja nende andmetüüpe. Uue kulleri lisamiseks andmebaasi API versioon nr. 1 puhul kasutame v1 kaustas asuvat Request kaustas esinevat Common kausta loodud Courier objekti.

Sageli on tarvis API päringuga tagastatavatel andmetel esitada koos baasandmetega sellega seotud alamandmed. Näiteks saadetise info puhul sellega seotud andmete kaasa andmiseks, sealhulgas saadetisele vastav tellimus, kuller, liikumisetüüp jne. Selle tarbeks on loodud funktsioon CommonParcelDbSet, milles on defineeritud andmebaasi liit päringud vastavalt andmebaasis asuvate saadetise tabeliga seotud tabelitele. Lisa päringute tegemiseks kasutatakse EntityFramework Include ja ThenInclude funktsioone, mis genereerivad automaatselt JOIN SQL käsklused saadetisega seotud tabelite pihta. Nii kombineeritakse saadetise andmetega ka sellega seotud tellimuse number, kulleri nimetus jne. CommonParcelDbSet funktsiooni poolt tagastatud saadetise andmeid saab seda välja kutsuv funktsioon veel lisaks filtreerida vastavalt saadetise ID-le või muule kriteeriumile. Antud funktsioon on loodud eesmärgiga vähendada korduvalt esinevat identset koodi, kus päritakse andmebaasi andmeid samade liidetud andmete tulemuste ootustega.

Arendaja saab kasutada API lõpp-punktide lihtsamaks käsitsi testimiseks graafilise liidesega testprogrammi näiteks Postman [26]. Postman-is on võimalik salvestada API päringute aadressid koos kaasa antud andmetega. Nii on erinevate API päringute jooksutamine üheainsa nupuvajutuse kaugusel. Lisaks saab salvestada ka erinevad serveri aadressid ja autentimise võtmed.



Joonis 10: Postman programmi näidispäring.

7 Tarkvara rakendused

Uue laotarkvara tagarakenduste tehnoloogiaks valiti C# programmeerimiskeel koos .NET raamistikuga. Antud tehnoloogia on veebitarkvarade vallas üks enamlevinum [27] ning uuenduste väljaandmine on küllaltki sagedane. Selle populaarsus on kasvutrendis [28]. C# ja tema peamise konkurendi Java vahel ei ole tarkvara lõpptulemuse kontekstis märkimisväärset erinevust, seega langes valik lõputöö autori kogemustepagasit arvestades C# kasuks.

.NET raamistik on aastate jooksul läbinud suuri muudatusi. Alates 2014 aastast ehitati .NET uuesti üles, toetamaks lisaks Windows operatsioonisüsteemile ka Linux ja MacOS peal jooksutamist [29].

Eesrakenduse arenduseks valiti skriptimiskeel JavaScript. JavaScript on maailmas enamlevinud variant veebilehtede arendusel. Arenduse ajakulu vähendamiseks valiti JavaScript-ile samuti enamlevinud raamistik React ning sellele omakorda raamistik Next.js. Next.js hoiab arenduse aega kokku genereerides veebirakenduse URL *(Uniform Resource Locator)* suunamiste funktsioonid veebilehtedele automaatselt vastavalt failide asukohtadele failipuus [30].

JavaScript-iga arendamisel kasutatakse antud projektis TypeScript süntaksit. See nõuab arendaja poolt programmikoodis kõikide muutujate, objektide ja meetodite tüübi defineerimist. See eeldab arendusel vähesel määral suuremat ajakulu aga hoiab ära rakenduse mahu kasvades potentsiaalsed arendajast tulenevad vead, kus koodis erinevates asukohtades kasutatakse samal elemendil erinevat tüüpi, mille tulemusena võib rakenduse töös tekkida vigu.

Projekti eesrakenduse disain ei ole autori tööandja poolt spetsifitseeritud seega disaini elementideks, sealhulgas nupud, tekstilahtrid, tabelid jms osteti olemasolev komponentide komplekt nimega Devias Kit Pro Client & Admin Dashboard [31]. Komplekti pakub MUI veebipood ning valiti Standard Plus pakett maksumusega 129 USD. See hoiab kokku arenduse aega jättes vahele enamiku disainielementide

programmeerimise arendaja poolt. Valik langetati antud komplekti kasuks võttes arvesse selle populaarsust ja tööandja arvamust.

7.1 Tagarakenduste ühised omadused

Järgnevalt vaatleme operatsioone, mille loogika on kõikidel laotarkvara tagarakendustel sama.

7.1.1 Andmebaasiga suhtlus

Kõik lõputöös käsitletud tagarakendused, mis suhtlevad andmebaasiga kasutavad objekt-andmebaas kaardistaja raamistikku EntityFramework. See võimaldab andmebaasiga seotud koodi kirjutamise hulka vähendada kasutades andmebaasi tabelitele viitamiseks domeeniobjekte. Domeeniobjektides on defineeritud andmebaasi tabelite tulbad ja nende andmetüübid koos tabelitevaheliste seosetega.

```
namespace App.Domain;

11 usages  Markus Kildemaa *  4 exposing APIs
public class Subject : DomainEntityId
{
    public Guid MovementTypeId { get; set; }
    1 usage
    public MovementType MovementType { get; set; } = null!;
    public string Name { get; set; } = default!;
    public string? ExternalId { get; set; }

    public ICollection<Order>? Orders { get; set; }
    2 usages
    public ICollection<SubjectContact>? SubjectContacts { get; set; }
}
```

Joonis 11: Domeeniobjekti näidis Parcelo tagarakenduses.

```
dotnet ef migrations add --project App.DAL.EF --startup-project
ParceloApp Initial
```

Joonis 12: Migratsioonifaili loomise käsklus.

Domeeniobjektidega loome sealhulgas võimaluse kasutada Code-First meetodit [32] andmebaasi tabelite ja tulpade genereerimiseks ja muutmiseks kasutades migratsioone. Selleks loome rakenduse loomise algfaasis kõigepealt doomeniobjektid vastavalt sellele kuidas soovime, et andmebaasi tabelid ja tulbad olema peavad. Seejärel käivitame terminalis käskluse, mis kasutades EntityFrameworki loob migratsioonifaili, mis sisaldab SQL keeles käskluseid loomaks andmebaasis tabelid ja tulbad vastavalt kuidas on rakenduse koodis domeeniobjektid defineeritud.

```
dotnet ef database update --project App.DAL.EF --startup-project
ParceloApp
```

Joonis 13: Andmebaasi uuendamise käsklus.

Kui migratsioon loodud, käivitame terminalis andmebaasi uuendamise käskluse mis omakorda käivitab automaatselt andmebaasis tabelite ja tulpade genereerimise.

Kui on tarvis andmebaasi arhitektuuris muudatusi teha, muudame koodis domeeniobjekte ja loome uue migratsiooni. EntityFramework jälgib automaatselt ise millised muudatused on tehtud ning loob selle järgi uute SQL käsklustega migratsioonifaili. Peale selle jooksutame taaskord terminalis andmebaasi uuendamise käsklust ning andmebaas uuendatakse automaatselt.

Code-First meetod eeldab, et arendaja ei muuda iseseisvalt andmebaasis tabelite struktuure vaid kasutab selleks ainult programmikoodi ja migratsioonifaile. Vastasel juhul tekivad migratsioonide jooksutamisel konfliktid, mis enne ei lahene kui käsitsi tehtud muudatused muuta tagasi endisesse olekusse.

Andmebaasiga ühendamisega seotud konfiguratsioone hoitakse appsettings.json failis. Seal on muuhulgas defineeritud kasutusesoleva MySql andmebaasi aadress koos kasutajanime, parooli, nime ja andmebaasi draiveriga.

```
"ConnectionStrings": {
  "DefaultConnection": "DataSource=app.db;Cache=Shared",
  "MysqlConnection": "Server=localhost;Port=5445;Uid=root;Pwd=test;Database=TM"
},
```

Joonis 14: Andmebaasiga ühendamiseks vajalik info appsettings.json konfiguratsioonifailis Parcelo tagarakenduses.

7.1.2 Turvalisus

Kõik tagarakendustele saadetud API päringud, mille vastused on piiratud sisselogitud kasutajatele, peavad päises sisaldama JWT (Json Web Token) väärtust. Päringu päise atribuudi nimetus peab olema Authorization ning selle väärtus algama tekstiga Bearer ja selle järel tühik koos JWT andmega. Pärija on varasemalt saanud JWT väärtuse AttilaSSO tagarakenduse kaudu.

```
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJBIjoiQiJ9.ObdlsIi545nHk9gsZ-
W0GMVXwTDIHFejpJw2fvgVlFA
```

Joonis 15: JWT päise näide.

Rakenduse kontroller, saades päringuga kaasa JWT, valideerib selle kasutades Microsoft.IdentityModel teeki. Valideerimise käigus kontrollitakse kas JWT vastab standardkujule, pole ületanud kehtivusaega, sisaldab korrektset signatuuri ja kas sisseloginud kasutajal on õigus näha päritavat ressurssi. Kui väärtus on tühi, puuduv või valideerimine ei ole edukas, tagastatakse pärijale tühi vastus koos staatuse koodiga 401.

7.2 Transcriber

Transcriber tagarakendus on loodud eesmärgiga logida erinevate rakenduste poolt andmebaasi andmetega manipuleerimised - lisamine, muutmine ja kustutamine. Välised rakendused saadavad peale andmebaasi andmete muutmist API päringu abil Transcriberile sellekohase info ning Transcriber salvestab andmebaasi LogAction tabelisse vastavate andmetega uue rea. See aitab jätta kogu tarkvarasüsteemi sees tehtud kasutajate toimingute kohta maha jälje, olgu selleks uue saadetise koostamine või kulleri kustutamine. Logide andmeid saab kasutaja vaadata eesrakenduses Monki.

Logimiseks eraldiseisva rakenduse kasutamise eesmärk on hoida logi andmete lisamise loogikat ühes kohas. See väldib rakendustes duplikaat koodi tekkimist, mille tulemusena ei pea enamik logimise protsessidega seotud muudatuste tegemiseks muutma kõikide rakenduste koodi eraldi.

7.3 AttilaSSO

AttilaSSO tagarakendus vastutab lõppkasutajate autentimise eest kogu tarkvarasüsteemis. Sisselogimise protsess algab kasutaja poolt eesrakenduse Monki sisenemise veebilehel kasutajanime ja parooli sisestamisega. Seejärel saadab Monki sisestatud kasutajanime ja parooli andmed API päringu abil AttilaSSO tagarakendusele. AttilaSSO kontrollib kas kasutaja andmed vastavad andmebaasis salvestatud andmetele. Kui vastavad, genereeritakse JWT, mis sisaldab selle kehtivusaega, signatuuri ja kasutaja ID-d. Seejärel tagastatakse JWT väärtus pärijale JSON kujul.

AttilaSSO sisaldab sealhulgas API lõpp-punkti, millele JWT väärtust saates, valideeritakse see ning validatsiooni edukalt läbides tagastatakse uus genereeritud JWT, mille kehtivusaega on pikendatud. See on tarvis, et hoida kasutajat eesrakenduses sisselogituna senikaua kui ta seda kasutab, saates perioodiliti API päringuid uue kehtivusajaga JWT saamiseks. Vastasel juhul võib juhtuda olukord, kus eesrakenduse kasutaja logitakse kasutamise keskel välja, selline olukord pärsib tarkvara kasutajasõbralikkust.

7.4 Noom-TM Syncer

Noom-TM Syncer tagarakenduse otstarve on sünkroniseerida omavahel Noom-i ja uue laotarkvara andmebaasi andmed. Projekti MVP raames toimub sünkroniseerimine esialgu ühepoolselt, kus laotarkvara andmeid uuendatakse Noom programmi järgi. See eeldab, et neid laotarkvara andmetabeleid, mis samastuvad Noom tabelitega, muudetakse lõppkasutajate poolt ainult Noom programmis. Kahepoolne sünkroniseerimise võimekus lisatakse laotarkvara versioonis nr. 1.5.

Sünkroniseerimise eesmärgiks on eemaldada vajadus igal tagarakendustel eraldi pärida andmeid otse Noom-i andmebaasist põhjustel, et selle andmebaas on üles ehitatud tänaseks aegunud platvormil Firebird 2.5, andmed ei ole indekseeritud, arhitektuur on kehva kvaliteediga ja ei järgi SQL standardeid, mille tulemusena andmestruktuurid ei ole ühtlaselt defineeritud ning andmete pärimise protsess võtab liigselt kaua aega pärssides tarkvara kasutajasõbralikkust.

Sünkroniseerimiseks pärib esmalt rakendus erinevatest Noom andmebaasi tabelitest andmed, seejärel vaatab kas laotarkvara andmebaasis need andmed juba eksisteerivad või mitte. Kui eksisteerivad, uuendab laotarkvara tabelite kirjed vastavalt, kui mitte siis lisab need. Sünkroniseerimine toimub ööpäevaringselt katkematult joostes.

Kuna Noom andmebaasi andmete tüübid ei ole konkreetselt defineeritud, saavad kirjete väärtused kasutada oma tabelis ühes veerus erinevaid stiile. Näiteks veerg, mille all olevatelt kirjetelt oodatakse binaarset väärtust, tõene või vale, on teatud hulga kirjetel märgitud sümbolitega pluss ja miinus, teisel hulgal on väärtused sõnadega true ja false ning ülejäänutel on väärtused defineeritud numbritega 0 ja 1. Selline andmete ebaühtlus kohustas arendajal luua Noom-TM Syncer-is funktsioonid, mis sünkroniseerimise käigus konverteerivad Noom andmebaasist korjatud andmed ümber, asendades kirjete väärtused vastavalt laotarkvara andmebaasis defineeritud andmetüüpidele. Selle tarbeks pidi arendaja otsima üles kõik erinevad variandid, millena on Noom andmebaasis olemasolevad andmed esindatud.

7.5 Parcelo

Parcelo tagarakendus sisaldab tarkvarasüsteemis laortiklite, tellimuste, kullerite ja saadetiste haldamiseks tarvilikke funktsioone. Peamiselt koosnevad funktsioonid andmebaasi kirjete manipuleerimisega seotud tegevustest: lugemine, lisamine, uuendamine ja kustutamine. Tähtsaim roll Parcelo-l on pakkuda võimekus siduda saadetiste andmed tellimuste infoga. See tähendab lisada ettevõtte müügi- ja ostutellimustele saadetisi, millega on tellitud tooted välja saadetud. Eelnevalt on ettevõtte töötajatel puudunud konkreetne ülevaade millised saadetised on seotud milliste tellimustega.

7.6 CourierButler

CourierButler tagarakenduse eesmärk on kullerteenuseid kasutavate saadetiste automaatne genereerimine ja teekonna jälgimine. Ettevõttes kasutavate kullerteenuste hulgas on DPD, DHL, UPS ja Fedex. Ettevõtte töötajate senine standartne töövoog saadetiste haldamiseks on näinud ette iga kullerteenuse pakkuja veebikeskkonnas paki saatmise tarbeks registreerida saadetis ning paki teekonna jälgimiseks trükkida saadetise

triipkood seal asuvasse lahtrisse. Väljaminevate ja sissetulevate saadetiste üles märkimine on eelnevalt lahendatud Microsoft Excel programmis tabeli täitmisega. Selline töö protsess ei ole iga kullerteenuse pakkuja puhul ühtne ning nõuab märkimisväärses koguses manuaalset tööd, logides igas veebiliideses iga kasutusjärje puhul sisse ja sisestada paki saaja andmed, kopeerides need Noom programmi tellimuste tabelist ümber. CourierButler-i eesmärk on eemaldada andmete käsitsi sisestamise protsess ja tekitada võimalus sooritada need operatsioonid ühes veebirakenduses.

Saadetiste jälgimine programmaatiliselt on lahendatud kasutades Ship24 API teenust [33]. See teenus pakub populaarseimate kullerteenuste pakkide teekonna jälgimist ühtse standardiga API-t kasutades. See likvideerib nõude iga kullerteenuse jaoks eraldi API funktsioonide loomist jälgimistegevuste raames.

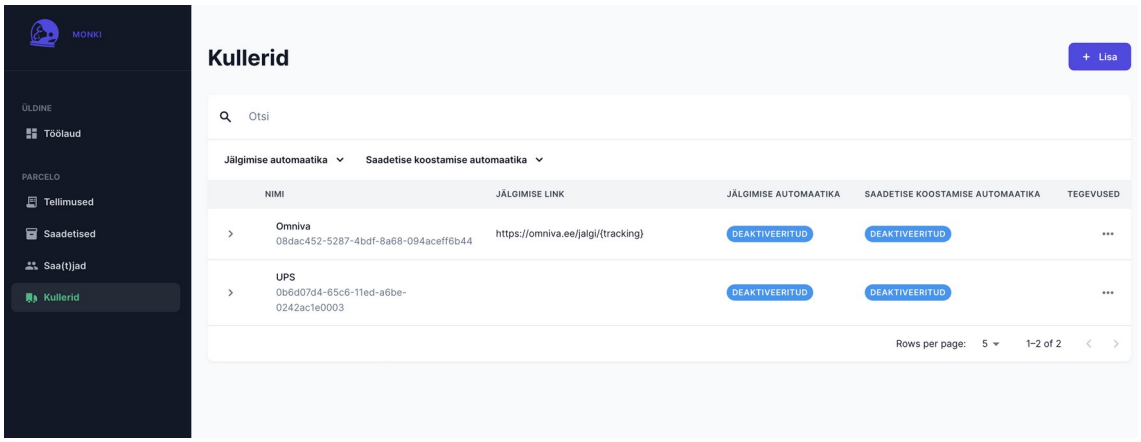
CourierButler käivitab iga teatud perioodi tagant funktsiooni, mis kogub esmalt kokku kõik laotarkvara andmebaasis salvestatud saadetiste jälgimiskoodid ning pärib nende kohta Ship24 API-t kasutades kõik saadetiste teekonnaread ning salvestab need laotarkvara andmebaasis TrackingRow tabelisse. Jälgimisega seotud info laotarkvara andmebaasis hoidmine hoiab Ship24 API-le suunatud päringute koguse minimaalsena, vastasel juhul peab saatma Ship24 API pihta päringu iga kord kui lõppkasutaja soovib eesrakendus Monki-s neid andmeid kuvada.

Saadetiste genereerimise jaoks kullerteenuste andmebaasides on tarvis kasutada kullerfirmade poolt pakutavaid API lõpp-punkte. Autor ei leidnud selle jaoks loodud olemasolevat teenust, mis pakuks ühtset keskkonda sarnaselt Ship24 pakkide jälgimise teenusele seega on vaja kasutada iga kullerfirma API-t eraldi. CourierButler rakendus on projekteeritud sellise arhitektuuriga, mis võimaldab lisada kullerfirmade API lõpp-punktide pärimise funktsioone võimalikult minimaalse ajakuluga hoides neid eraldi kihis isoleerituna ülejäänud rakenduse koodist.

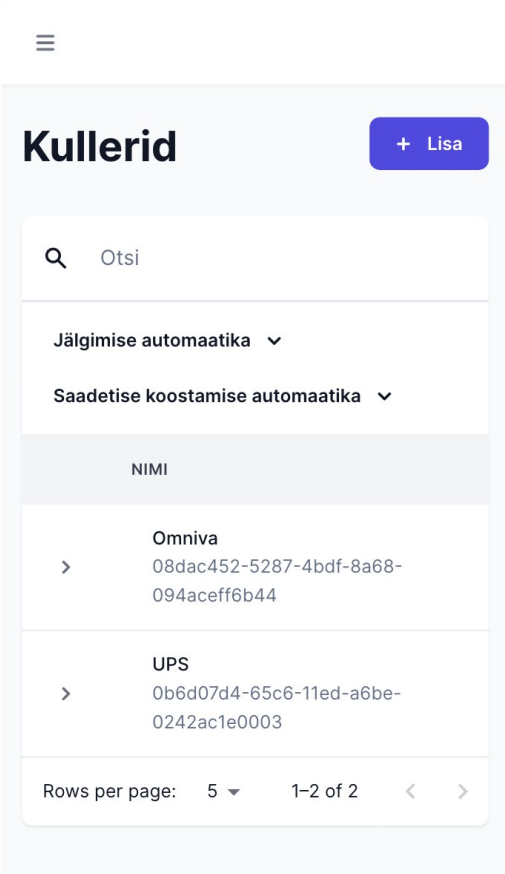
7.7 Monki

Eesrakendus Monki on veebiprogramm brauseris kasutamiseks lõppkasutajate poolt. Kogu lõputöös käsitletud laotarkvara inimkasutus toimub Monki kaudu. Läbi brauseri kasutamine tagab võimaluse opereerida tarkvarat iga seadmega, kuhu saab installeerida veebibrauserit, sealhulgas nutitelefonid ja nutitahvlid. Selle tarbeks on rakendus loodud

kasutades Responsive metoodikat, kus veebilehtede komponendid on arendatud sellistena, mis kohandavad oma suurust ja paigutust automaatselt vastavalt kasutatava seadme ekraani suurusele.



Joonis 16: Eesrakendus Monki kullerite vaade laiekraan seadmes.



Joonis 17: Eesrakendus Monki kullerite vaade nutitelefonis.

Monki eesmärk on anda võimaluse kasutada kõikide autori poolt arendatud rakenduste funktsioone ühes veebiliideses, ka neid, mis on käesoleva lõputöö skoobist väljas, sealhulgas e-mailide, konsignatsiooni laokappide ja töötajate tegevusnimekirja andmete haldamine.

Laotarkvara raames on Monki ettenähtud haldama ostu- ja müügitellimuste, tellimustega seotud saadetiste, kullerite, tarnijate ja klientide andmeid, kuid koostöös autori ja tema tööandjaga on rakenduse lõpptulemuse visioon pidevas muutuses. Monki disaini ja nõutud funktsioonide nimekirja kohandatakse peale igat arendustsüklit.

8 Tulemus

Lõputöös käsitletud arenduse tulemusena sai käesolevaks hetkeks valmis laotarkvara MVP versiooniga nr. 0.9. Kõik lõpptulemuselt eeldatud kriteeriumid ei saanud valminud versioonis täidetud. Ülejäänute funktsionaalsuste arendused on planeeritud järgmistes versioonides. Versioon 1.0 on planeeritud valmima järgneva ühe kuni kahe kuu jooksul ning versioon 1.5 järgmise kolme kuni nelja kuu jooksul.

Tabel 2: Uuelt tarkvaralt nõutud funktsioonide täidetavus.

Kriteerium	**Laotarkvara vastavus**
Andmete ja programmi vaadete kiire kuvamine. Iga baastasemel funktsioon ja vaade peavad saama täide viidud vähemalt 500 millisekundi jooksul.	Võimaldab.
Laoartiklite lisamine, lugemine, muutmine ja kustutamine.	Lugemine võimaldatud. Lisamine, muutmine ja kustutamine versioonis 1.5.
Laoartiklite tabeli filtreerimine kategooria järgi.	Võimaldab.
Laoartikli kategooria lisamine, lugemine ja muutmine.	Võimaldab.
Laoartikli hinna ja tarnija lisamine, lugemine ja muutmine.	Hinna lugemine võimaldatud. Hinna ja tarnija lisamine, muutmine ja kustutamine versioonis 1.5.
Laoartiklile triipkoodide lisamine ja kustutamine piiramatus koguses.	Versioonis 1.5.
Laoartiklile lao asukohtade lisamine ja kustutamine piiramatus koguses.	Versioonis 1.0.
Toodete automaatselt tellimine tarnijatelt.	Versioonis 1.5.
Klientide andmete lisamine, lugemine, muutmine ja kustutamine.	Lugemine võimaldatud. Lisamine, muutmine ja kustutamine versioonis 1.5.

Kriteerium	**Laotarkvara vastavus**
Programmi kasutamine väljaspool Attila kontorit.	Võimaldab.
Programmi kasutamine Windows, IOS, MacOS, Linux ja Android seadmetega.	Võimaldab.
Funktsioonide kasutamise piiramine vastavalt sisselogitud kasutaja rolli õigustele.	Võimaldab.
Saadetiste lisamine, lugemine, muutmine ja kustutamine.	Võimaldab.
Saadetiste tabeli filtreerimine Euroopa Liidu liikmesriikide sisese liikumise järgi.	Võimaldab.
Skaleeritavuse võimekus ehk uute funktsioonide lisamine.	Võimaldab.
Toodete laoseisu haldamine mobiilse seadmega.	Versioonis 1.5.
Kullersaadetiste genereerimine Omniva, UPS, DHL ja DPD andmebaasides.	Omniva saadetiste koostamine võimaldatud. Ülejäänud kullerfirmade puhul versioonis 1.0.
Kullersaadetiste kleebiste printimine.	Versioonis 1.0.
Kullersaadetise jälgimise info lugemine sealhulgas kullerteenuste Omniva, UPS, DHL ja DPD.	Versioonis 1.0.
Liidestus Attilas kasutusesolevate programmidega.	Võimaldab.

Valminud tarkvara üheks kasuteguri hindamiseks valiti tööprotsess, millele saab kasuteguri kontekstis anda konkreetse numbrilise väärtuse, mõõtes ajakulu ilma ja koos laotarkvara kasutamisega. Tööprotsessiks valiti ettevõtte laotöötaja ühe kullersaadetise koostamine ja sellega seonduva info töötlemine. Vanaviisi, enne laotarkvara loomist, kulus töötajal testi tulemusena selleks aega 2 minutit ja 11 sekundit. Laotarkvaras võttis sama väljundiga tööprotsess 6 sekundit. Ühes päevas koostatakse keskmiselt 30 saadetist. Ajavõit ühes päevas on ligikaudu 1 tund. Ülejäänud tarkvara kasutegurid ei ole ajaliselt mõõdetavad vaid väljenduvad andmete manuaalselt sisestamisest tingitud

inimveade likvideerimise, laoliikumiste parema ülevaate ja muude lisafunktsioonide näol.

9 Kokkuvõte

Käesoleva lõputöö eesmärgiks oli lahendada ettevõtte Attila poolt etteantud nõuded, mis kirjeldasid peamiselt kasutuseloleva müügi- ja laoprogrammi Noom puudujääke. Selle tarbeks loodi uus laotarkvara, mille otstarve on täita antud kriteeriumid.

Käesolevaks hetkeks valmis versiooniga nr. 0.9 laotarkvara, mis täidab kahekümnest kriteeriumist kümme.

Tarkvara arenduse käigus loodi viis tagarakendust ja üks eesrakendus. Eesrakendus Monki käitub veebiliidesena, mida saavad tarkvara funktsioonide kasutamiseks opereerida ettevõtte töötajad. CourierButler tagarakendus vastutab kullerteenuste pakkujate saadetiste genereerimise ja teekonna jälgimise eest. Transcriber tagarakendus aitab jätta kõikidest lõppkasutajate tegevustest maha jäljed andmebaasi. Parcelo tagarakendus haldab laoartiklite, tellimuste, tarnijate ja klientide andmeid. Noom-TM Syncer tagarakendus sünkroniseerib perioodiliti omavahel Noom-i ja uue laotarkvara andmebaaside andmed. AttilaSSO tagarakendus käitub tarkvara autentimistega seotud teenusena.

Peale tarkvara versioon 1.5 valmimist, on tulevikus plaan laiendada tarkvara funktsionaalsust veelgi ja hägustada mõttelist piiri laotarkvara ja ülejäänud ettevõttes loodud rakenduste vahel, luues ühtne rakendustest koosnev ettevõttesisene tarkvarasüsteem, milles olevad tagarakendused, koos eesrakendusega Monki, teevad omavahel tihedat programmaatilist koostööd.

Lõputöö autor õppis arenduse käigus palju uusi asju, märkimisväärseimaks õppekohaks osutus andmebaaside vaheline sünkroniseerimine, kus oli vaja arvestada Noom andmebaasi puhul andmete ebakorrapäralise esindatavusega. Andmed olid salvestatud samas tabelis erinevate standarditega ning esines olukordi, kus osa andmetest esines korrapäratult mitmes tabelis korraga. Samuti õppis tundma Firebird andmebaasi platvormi eripärasid.

Kasutatud kirjandus

[1] Rakendusliides - Vikipeedia [Online]
https://et.wikipedia.org/wiki/Rakendusliides (30.11.2022)

[2] Kasutaja:Tomiandrep/Klient-server_arhitektuur - Vikipeedia [Online]
https://et.wikipedia.org/wiki/Kasutaja:Tomiandrep/Klient-server_arhitektuur
(30.11.2022)

[3] Mis on ERP? - Erpcon [Online]
https://erpcon.ee/kasulikku/mis-on-erp (30.11.2022)

[4] Hüperteksti edastuskontroll - Vikipeedia [Online]
https://et.wikipedia.org/wiki/H%C3%BCperteksti_edastusprotokoll (30.11.2022)

[5] IP-aadress – Vikipeedia [Online]
https://et.wikipedia.org/wiki/IP-aadress (30.11.2022)

[6] JSON – Vikipeedia [Online]
https://et.wikipedia.org/wiki/JSON (30.11.2022)

[7] JSON Web Token – Vikipeedia [Online]
https://et.wikipedia.org/wiki/JSON_Web_Token (30.11.2022)

[8] Vähim elujõuline toode – Vikipeedia [Online]
https://et.wikipedia.org/wiki/V%C3%A4him_eluj%C3%B5uline_toode (30.11.2022)

[9] Ühekordne sisselogimine – Vikipeedia [Online]
https://et.wikipedia.org/wiki/%C3%9Chekordne_sisselogimine (30.11.2022)

[10] Frontend and backend – Wikipedia [Online]
https://en.wikipedia.org/wiki/Frontend_and_backend (30.11.2022)

[11] URL – Wikipedia [Online]
https://en.wikipedia.org/wiki/URL (30.11.2022)

[12] What Is Agile Scrum Methodology? [Online]
https://www.businessnewsdaily.com/4987-what-is-agile-scrum-methodology.html
(22.11.2022)

[13] Hein Smith. (2018) Scrum: The Ultimate Beginner's Guide To Learn And Master Scrum Agile Framework

[14] Minimum Viable Product (MVP) – What is a MVP and why is it important? [Online]
https://www.productplan.com/glossary/minimum-viable-product/ (22.11.2022)

[15] Git Branching and Branching Strategy – DEV Community [Online]
https://dev.to/preethamsathyamurthy/git-branching-and-branching-strategy-4mci
(22.11.2022)

[16] James Turnbull. (2014) The Docker Book: Containerization is the new virtualization

[17] Mis on lõpp-punkt? | Microsofti turbeteenus [Online]
https://www.microsoft.com/et-ee/security/business/security-101/what-is-an-endpoint
(22.11.2022)

[18] Microservice Architecture Pattern [Online]
https://microservices.io/patterns/microservices.html (23.11.2022)

[19] Andmebaas – Vikipeedia [Online]
https://et.wikipedia.org/wiki/Andmebaas (28.11.2022)

[20] SQL Database Modeler [Online]
https://sqldbm.com/ (28.11.2022)

[21] DigitalOcean | The Cloud for Builders [Online]
https://www.digitalocean.com/ (28.11.2022)

[22] PhpMyAdmin [Online]
https://www.phpmyadmin.net/ (28.11.2022)

[23] The Importance Of Code Reusability In Software Development [Online]
https://cloudemployee.co.uk/blog/code/the-importance-of-code-reusability-in-software-development (29.11.2022)

[24] Creating Domain-Driven Design entity classes with Entity Framework Core [Online]
https://www.thereformedprogrammer.net/creating-domain-driven-design-entity-classes-with-entity-framework-core/ (29.11.2022)

[25] HTTP Methods for RESTful Services [Online]
https://www.restapitutorial.com/lessons/httpmethods.html

[26] Postman API Platform [Online]
https://www.postman.com/ (29.11.2022)

[27] Top 7 Programming Languages for Backend Web Development [Online]
https://www.geeksforgeeks.org/top-7-programming-languages-for-backend-web-development/ (22.11.2022)

[28] C# popularity surges in Tiobe programming language index [Online]
https://www.infoworld.com/article/3660078/c-sharp-popularity-surges-in-tiobe-programming-language-index.html (22.11.2022)

[29] Understanding cross platform .NET, and why .NET 5 is important [Online]
https://www.fearofoblivion.com/understanding-cross-platform-NET-and-why-NET-5-is-important (22.11.2022)

[30] Routing: Introduction | Next.js [Online]
https://nextjs.org/docs/routing/introduction (22.11.2022)

[31] Devias Kit Pro – Client & Admin Dashboard | MUI Store [Online]
https://mui.com/store/items/devias-kit-pro/ (22.11.2022)

[32] Part 2: Creating the Domain Models [Online]
https://learn.microsoft.com/en-us/aspnet/web-api/overview/older-versions/using-web-api-1-with-entity-framework-5/using-web-api-with-entity-framework-part-2 (29.11.2022)

[33] PARCEL TRACKING – Track Package & Shipment Delivery [Online]
https://www.ship24.com/ (22.11.2022)

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

Mina, Markus Kildemaa

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Laotarkvara loomine väikeettevõttele" mille juhendaja on Meelis Antoi

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.

06.01.2023

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.