TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Priit Päll 164314 IAPB

HULGIKAUBANDUSETTEVÕTTE TÖÖVÕTETE JA AJAKASUTUSE PARENDAMINE VEEBIRAKENDUSE KASUTUSELEVÕTMISE NÄOL

Bakalaureusetöö

Juhendaja: Mart Roost
MSc

Tallinn 2020

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: Priit Päll

03.08.2020

Annotatsioon

Tootmistööstuse protsessid on keerukad ja hõlmavad suuremahulist info liikumist, mida tuleb pidevalt valideerida. Hulgikaubandusettevõttes on kesksel kohal tellimuse täitmise efektiivsus, millest oleneb otseselt ettevõtte edukus. Efektiivsusel mängib rolli nii tellimuse täitmiseks kuluv aeg, kui ka vigade esinemise sagedus. Üheks viisiks inimlikke vigu vältida on anda valideerimise vastutus üle arvutitele.

Töö eesmärk on luua häid tarkvaraarenduse tavasid järgiv veebirakendus, mis aitab ettevõttel hoida kokku aega ja vähendada vigade tekkimise võimalust. Töö käigus loodi kliendi nõutele vastav veebirakendus, kus hulgikaubandusettevõtte kliendid saavad teha tellimusi ja ettevõtte töötajad saavad hallata ning töödelda loodud tellimusi.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 36 leheküljel, 8 peatükki, 18 joonist, 1 tabelit.

Abstract

Improving the workflow and time management of a wholesale company by the introduction of a web application

The processes of a wholesale company are complicated and prone to error. Both, the process of placing and filling orders contain a lot of product specific information, most of which need to be constantly validated by either the company receiving the order or the client placing the order. Efficiency is of utmost importance and the success of a company depends directly on it. Efficiency itself depends on the time it takes to fill an order and the frequency of errors that occur throughout processing orders. One possible way to minimize the number of occurring errors while also improving the efficiency of order processing is to use computers. Companies can shift the responsibility of validating orders from people to computers.

The aim of this work is to create a web application which follows good software development practices and uses the best possible technological solutions. By introducing and using a web application the number of errors occurring while filling orders should decrease and the time consumed by the processing of orders should be reduced for both, the company and its clients.

In the course of this thesis, a web application was built according to the requirements presented by the client. For the clients of the company, this provided an opportunity to place orders in the web application. Whereas, for the company it introduced the possibility to manage and process orders placed by clients.

The thesis is in Estonian and contains 36 pages of text, 8 chapters, 18 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides, liides kahe programmi omavaheliseks suhtluseks
Back end	Tagakomponent, server
DOM	*Document Object Model*, dokumendi objektimudel
Front end	Eeskomponent, klientrakendus
HATEOAS	*Hypermedia as the Engine of Application State*
HTTP	*Hypertext Transfer Protocol*, protokoll teabe edastamiseks arvutivõrkudes
JSON	*JavaScript Object Notation*, JavaScript keelel põhinev andmevahetusvorm
JWT	*JSON Web Token*, JSON kujul pääsuluba
MVC	*Model-View-Controller*, tarkvararakendust kolmeks omavahel seotud osaks jagav tarkvaraarhitektuurimuster
REST	*Representational State Transfer*, tarkvaraarhitektuuri laad, mis seab veebirakenduse loomiseks kindlad piirid
SOLID	Objektorienteeritud programmeerimise disainiprintsiipide kogum
SQL	*Structured Query Language*, struktuuripäringukeel, domeenispetsiifiline keel haldamaks andmeid relatsioonibaasihaldurites
URI	*Uniform Resource Identifier*, ühtne ressursiidentifikaator
XML	*Extensible Markup Language*, laiendatav märgistuskeel
YAML	*YAML Ain't Markup Language*, konfiguratsioonifailides kasutatav inimloetav keel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tootmistööstuses käib tööpäeva jooksul läbi tohutu kogus infot alates tellimuse tegemisest kuni tootmise ning kohaletoimetamiseni. Roomakardinad on aknakatted, mis on elegantse väljanägemisega – kui tõmmata kardin üles, moodustavad kardinast voldid(Joonis 1). Roomakardina mehhanism koosneb hulgast alamkomponentidest, millel omakorda on lai valik variatsioone. Kardina kett võib olla tehtud plastikust või metallist. Plastikust kett võib värvuselt olla läbipaistev või hall, metallkett võib aga olla värvuselt kroom, antiik-messing või hõbedane. Lisaks sellele saab kardina mehhanism olla kiire või aeglase ülekandega. Nöörirullidel olevad nöörid, mis hoolitsevad kardina ülevalhoidmise ja tõstmise eest, võivad olla erinevates toonides ja nende pikkus oleneb sellest, kui kõrge kardin on. Kogu seda infot peab pidevalt silmas pidama ja valideerima nii roomakardinaid müüv ettevõte, kui ka tellija rollis olev klient. Olukord, kus mitmed osapooled peavad oma peas pidevalt tellimuste infot valideerima, toob kaasa eksimused ja ajakulu.

Joonis 1. Roomakardin [1].

Ettevõtte kliendid kasutavad roomakardinatellimuste tegemiseks enda loodud malle, kus info on kirjas eri viisil. Osad kliendid kasutavad tellimuste tegemiseks tabelil baseeruvat vormi, mõned kirjutavad tellimuse vabas tekstis ning esineb ka kliente, kes teevad jooniseid. Lisaks kindlale tellimuse mallile puudub ka üks kindel kanal, mille kaudu tellimusi edastada. Kliendid edastavad tellimusi vabalt valitud viisil – meili, faksi, sõnumi või telefoni teel. Kogu protsessi vältel peavad mitmed inimesed saadud infot valideerima ja tellimusi enda jaoks õigesti tõlgendama kuna sama termin võib erinevate klientide jaoks omada eri tähendust. Selline protsess toob endaga kaasa vead, mida saaks hõlpsasti ära hoida, luues tellimuste tegemiseks ühtse malli ning andes valideerimise ülesande üle süsteemile, kus tellimusi tehakse.

Käesoleva töö eesmärgiks on luua tootmisettevõttele veebirakendus, kus klient saab tellimusi sisestada ning jälgida ja ettevõte saab tellimuste protsessi juhtida ning omada head ülevaadet hetkeseisust. Tellimuste hoidmine ühes kindlas süsteemis annab ettevõttele palju parema ülevaate hetkeseisust ja aitab paremini planeerida aega ning ressursse. Ühtlasi aitab veebirakenduse kasutusele võtmine vähendada eksimusi – tellimusi luues valideerib süsteem kõik sisestatud andmed juba enne tellimuse esitamist, seega ei jõua vigane tellimus ettevõtteni. Samas aitab veebirakenduse kasutamine hoida klienti kursis uute tootevariatsioonidega – on kliente, kes kasutavad vanemaid tootekatalooge ning ei ole teadlikud uute toodete olemasolust. Veebirakenduses kardinaid tellides kuvatakse kõikvõimalikke detailide variatsioone, mida ettevõte on süsteemi lisanud. Kokkuvõtlikult, põhieesmärgiks on vigade minimaliseerimine ja aja kokkuhoid.

Töö koosneb kuuest osast. Esimene osa tutvustab loodavale rakendusele seatud nõudeid, analüüsib juba turul olevaid valmis tarkvaralahendusi ja põhjendab loodava veebirakenduse vajalikkust. Järgnev osa annab ülevaate töös kasutatud tehnoloogiatest ja toob välja nende valimise põhjused. Kolmas osa tutvustab rakenduse loomisel kasutatud häid tavasid ja printsiipe, tuues välja, miks need vajalikud on ning mida nende kasutamine kaasa toob. Seejärel käsitletakse rakenduse arenduskäiku, kus kirjeldatakse kasutusel olnud arendusviisi ja tutvustatakse loodud vaateid ja funktsionaalsusi. Töö eelviimane osa toob välja tehtud töö tulemused ja kirjeldab rakendusele antud tagasisidet. Viimane osa tutvustab võimalikke rakenduse edasiarendusi.

2 Infosüsteemi analüüs

Ettevõtte probleemide lahendamiseks defineeriti esmalt nõuded rakendusele, mis aitaks neid lahendada. Selle jaoks kaasati klienti, kelle abiga seati rakendusele funktsionaalsed ja mittefunktsionaalsed nõuded ning seatud nõuete põhjal analüüsiti juba turul olevate tarkvaralahenduste sobivust.

2.1 Mittefunktsionaalsed nõuded

Kliendiks oleva ettevõttega konsulteerides seati loodavale rakendusele mittefunktsionaalsed nõuded, mis on nähtavad tabelis 1.

Tabel 1. Rakenduse mittefunktsionaalsed nõuded.

Tüüp	Kirjeldus
Keel	Toodete kirjeldused ja nimetused peavad olema eesti keeles. Kõik süsteemi kasutajale nähtav peab olema eesti keeles s.h. menüü, nupud jne.
Kuupäeva ja kellaja formaat	Kuupäev peab olema formaadis DD.MM.YYYY. Kellaaeg peab olema 24h formaadis HH:MM.
Tegevusi tehes kinnituse küsimine	Rakendus peab kasutajalt küsima kinnitust, kui tehakse kriitilisi operatsioone või kui vastava tegevuse käigus läheks palju tööd kaotsi. Nende tegevuste hulka kuuluvad kustutamised, muudatuste salvestamised ja olukord, kus

	tellimuse tegemine on pooleli ja klient vajutab ekslikult „Ei“ nuppu.
Kasutusmugavus	Rakendus peab olema arusaadav ja kasutajasõbralik.
Turvalisus	Rakendusele pääsevad ligi ainult autenditud kasutajad. Ainult ettevõtte töötajatel on lubatud näha kõiki sisestatud tellimusi. Klientidel on ligipääs vaid enda poolt tehtud tellimustele.
Erinevate seadmete tugi	Rakendus on mõeldud arvutis kasutamiseks, seetõttu ei ole vaja lisarõhku panna mobiilseadmete toele.
Tellimuste valideerimine	Rakendus peab valideerima tellimusi vastavalt kliendi poolt seatud nõutele ja vigade esinemisel tuleb kuvada asjakohane veateade.

2.2 Funktsionaalsed nõuded

Kliendi põhiline soov on tarkvara sõltumatus ettevõttes kasutusel olevast laotarkvarast. Viimase puhul on tegu vana ning väga mitmeid aastaid tagasi soetatud tarkvaraga. Klient ei soovi soetatud tarkvara välja vahetada. Teiseks nõudeks on tarkvara keskendumine konkreetse probleemi (roomakardinatellimuste loomine, haldamine) lahendamisele, mis eeldab rakenduselt kindlat fookust, sisaldades võimalikult vähe lisafunktsioone, mille olemasolu ei ole põhjendatud.

2.2.1 Kasutusjuhud

Kasutusjuhtude kirjutamisel on aluseks võetud Tallinna Tehnikaülikoolis Erki Eessaare loetavas aines Andmebaasid I (IDU0220) kasutusel olnud kasutusjuhtude mall [2].

Kasutusjuht: Tuvasta kasutaja

Tegutsejad: Ettevõtte töötaja, administraator, ettevõtte klient (edaspidi subjekt)

Sündmuste järjestus:

1. Subjekt soovib süsteemi sisenda.
2. Süsteem palub subjektil ennast identifitseerida.
3. Subjekt sisestab oma kasutajanime ja parooli.
4. Süsteem kontrollib, kas andmebaasis on esitatud andmetega kasutaja olemas.
5. Süsteem annab subjektile loa süsteemi kasutada tema õiguste piires.

Laiendus (või alternatiivne sündmuste käik):

5. Kui süsteem ei leia sisestatud kasutajatunnusele vastet või sisestatud parool ei ole kasutajatunnusele vastav, ei saa subjekt süsteemi kasutada ning kuvatakse üldine veateade.

Kasutusjuht: Loo roomakardina tellimus

Tegutsejad: Ettevõtte töötaja, ettevõtte klient (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud ning omab vajalikke õiguseid.

Sündmuste järjestus:

1. Subjekt soovib luua roomakardina tellimuse.
2. Süsteem avab tellimuse loomise vormi, kuhu saab lisada roomakardinaid, lisainfot kommentaari näol ning valida tellimuse täitmise kuupäeva.
3. Subjekt sisestab vajaliku info ja vajutab „Salvesta“ nupule.
4. Kui kõik vajaminev info on korrektne, salvestab süsteem tellimuse andmebaasi.

Laiendus (või alternatiivne sündmuste käik):

4. Kui kohustuslik väli on täitmata või sisestatud info on vigane, kuvatakse sellekohane veateade.

Kasutusjuht: Kustuta roomakardina tellimus

Tegutsejad: Ettevõtte klient, ettevõtte töötaja (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud ja omab vajalikke õiguseid, tellimus on staatuses „SISESTATUD“ või „KINNITATUD“.

Sündmuste järjestus:

1. Subjekt soovib kustutada roomakardina tellimuse.
2. Subjekt avab vormi, kus kuvatakse kõik temale nähtavad tellimused.
3. Subjekt vajutab tellimuse real olevale „Kustuta“ nupule.
4. Süsteem küsib kustutamiseks kinnitust.
5. Subjekt vajutab „Jah“ nupule. Süsteem kustutab tellimuse.

Laiendus (või alternatiivne sündmuste käik):

5. Subjekt vajutab „Ei“ nupule. Tellimust ei kustutata.

Kasutusjuht: Muuda roomakardina tellimust

Tegutsejad: Ettevõtte klient, ettevõtte töötaja (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud ja omab vajalikke õiguseid, tellimus on staatuses „SISESTATUD“ või „KINNITATUD“.

Sündmuste järjestus:

1. Subjekt soovib muuta roomakardina tellimust.
2. Subjekt avab vormi, kus kuvatakse kõik temale nähtavad tellimused.

3. Subjekt vajutab tellimuse real olevale „Muuda“ nupule.
4. Süsteem avab vormi tellimuse muutmiseks
5. Subjekt teeb soovitud muudatused ja vajutab „Salvesta“ nupule.
6. Süsteem küsib kinnitust muudatuste salvestamiseks.
7. Subjekt vajutab „Jah“ nupule. Tehtud muudatused salvestatakse andmebaasi.

Laiendus (või alternatiivne sündmuste käik):

7. Subjekt vajutab „Ei“ nupule ja muudatustest loobutakse.

Kasutusjuht: Lisa roomakardinatellimusele roomakardin

Tegutsejad: Ettevõtte klient, ettevõtte töötaja (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud, asub tellimuse detailvaates ja omab vajalikke õiguseid.

Sündmuste järjestus:

1. Subjekt soovib tellimusele lisada roomakardina.
2. Süsteem avab vormi, kuhu saab sisestada roomakardina tootmiseks vajaminevad andmed.
3. Subjekt täidab vormi ja vajutab „Salvesta“ nupule.
4. Süsteem kontrollib sisestatud infot ja vigade puudumisel lisab antud roomakardina loodavale roomakardinatellimusele.

Laiendus (või alternatiivne sündmuste käik):

4. Süsteem leiab kontrolli käigus vea ja kuvab vastava veateate.

Kasutusjuht: Kustuta tellimusest roomakardin

Tegutsejad: Ettevõtte klient, ettevõtte töötaja (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud, asub tellimuse detailvaates ja omab vajaminevaid õiguseid. Tellimuse on staatuses „SISESTATUD“ või „KINNITATUD“.

Sündmuste järjestus:

1. Subjekt soovib kustutada tellimuselt roomakardina.
2. Subjekt navigeerib vastava tellimuse detailvaatesse.
3. Subjekt vajutab vastava roomakardina real olevale „Kustuta“ nupule.
4. Süsteem küsib kustutamiseks kinnitust.
5. Subjekt vajutab „Jah“ nupule ja süsteem kustutab andmebaasist roomakardina.

Laiendus (või alternatiivne sündmuste käik):

5. Subjekt vajutab „Ei“ nupule ja süsteem ei kustuta andmebaasist roomakardinat.

Kasutusjuht: Muuda roomakardinat

Tegutsejad: Ettevõtte klient, ettevõtte töötaja (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud, asub tellimuse redigeerimise vaates ja omab vajalikke õiguseid. Tellimus on staatuses „SISESTATUD“ või „KINNITATUD“.

Sündmuste järjestus:

1. Subjekt soovib muuta tellimusel olevat roomakardinat.
2. Subjekt vajutab vastava roomakardina real olevale „Muuda“ nupule.
3. Süsteem avab vormi roomakardina muutmiseks.
4. Subjekt teeb vastavad muudatused ja vajutab „Salvesta“ nupule.

5. Tehtud muudatused salvestatakse.

Laiendus (või alternatiivne sündmuste käik):

7. Subjekt vajutab „Ei“ nupule ja muudatusi ei salvestata.

Kasutusjuht: Vaata tellimuste nimekirja

Tegutsejad: Ettevõtte klient, ettevõtte töötaja (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud ja omab vajaminevaid õiguseid.

Sündmuste järjestus:

1. Subjekt soovib vaadata kõiki temale nähtavaid tellimusi.
2. Süsteem avab vormi, kus on näha tellimused, mida vastavale subjektile kuvada võib. Ettevõtte töötaja näeb kõiki aktiivseid tellimusi, klient ainult enda poolt vormistatud aktiivseid tellimusi.

Kasutusjuht: Vaata tellimust

Tegutsejad: Ettevõtte klient, ettevõtte töötaja (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud ja omab vajaminevaid õiguseid.

Sündmuste järjestus:

1. Subjekt soovib vaadata kindlat tellimust.
2. Subjekt navigeerib tellimuste nimekirja vaatesse.
3. Subjekt vajutab vastava tellimuse numbrile.
4. Süsteem avab vastava tellimuse detailvaate.

Kasutusjuht: Muuda tellimuse staatust

Tegutsejad: Ettevõtte töötaja (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud ettevõtte kasutajana ja vajaminevaid õiguseid.

Sündmuste järjestus:

1. Subjekt soovib muuta tellimuse staatust.
2. Subjekt navigeerib tellimuste nimekirja vaatesse.
3. Subjekt vajutab vastava tellimuse real olevale „Muuda staatust " nupule, mille peale avaneb hüpikaken staatuse muutmiseks.
4. Subjekt valib uue tellimuse staatuse ja vajutab nupule „Salvesta".
5. Süsteem valideerib uue staatuse ja seejärel salvestab muudatuse andmebaasis.

Laiendus (või alternatiivne sündmuste käik):

5. Süsteem leiab valideerimisel vea ja kasutajale kuvatakse vastav veateade. Tellimuse staatust ei muudeta.

Kasutusjuht: Lae alla tellimus

Tegutsejad: Ettevõtte klient, ettevõtte töötaja (edaspidi subjekt)

Eeltingimus: Subjekt on süsteemi sisse logitud ja omab vajaminevaid õiguseid.

Sündmuste järjestus:

1. Subjekt soovib alla laadida tellimuse faili.
2. Subjekt navigeerib vastava tellimuse detailvaatesse.
3. Subjekt vajutab detailvaates nupule „Lae alla".

4. Süsteem genereerib tellimuse põhjal Excel (.xlsx) formaadis faili, mis laetakse kasutaja jaoks alla.

2.3 Olemasolevad lahendused

Kuigi turul on mitmeid valmis tarkvaralahendusi, ei õnnestunud leida tarkvara, mis täidaks kõiki kliendi poolt seatud nõudeid. Sellest tulenevalt läks ettevõte oma veebirakenduse teed, mis loodi antud töö raames, lähtudes kliendi poolt seatud nõuetest.

2.3.1 NOOM majandustarkvara

NOOM majandustarkvara on kodumaine tarkvaralahendus. Tarkvara on mõeldud eelkõige ettevõtte aja ja ressursside planeerimiseks ning nende põhjal firma analüüsimiseks. NOOM tarkvara koosneb mitmest moodulist, mida on võimalik omavahel siduda, nende hulgas raamatupidamine, palk ja personal, laohaldus, tootmisarvestus, raamatupidamine ning müük. Tarkvaralahendus on võtnud sihiks koondada kõik ettevõtte osakonnad ühte kohta [3].

Lõputöö kirjutamise hetkel ei pakkunud NOOM majandustarkvara veebipoodi ega iseteeninduskeskkonda. NOOM pakkus API (*Application Programming Interface*) ehk rakendusliidese kasutamise võimalust, üle mille suhelda veebiplatvormidega, kuid taoline lahendus eeldaks eraldi e-poe rakenduse arendamist ja seega hõlmaks mitme eri rakenduse kasutamist, mis kliendi meelest poleks mõistlik.

2.3.2 Eziil

Eziil on tootmistarkvara, mis sisaldab endas ostu-, müügi-, lao- ja tootmise funktsioone. Tarkvara on mitmekülgne – suhtleb vajadusel tarnijatega, loob tootmise kohta raporteid ning võimaldab teha projektimüüki [4].

Antud tarkvara miinuseks on veebipoe osa puudumine. Töö raames võeti ühendust Eziil tarkvara pakkuva ettevõttega täpsustamaks, kuidas tarkvara on üles ehitatud ning millised funktsionaalsused on esindatud. Töö kirjutamise hetkel puudus e-poe standardlahendus kuid ettevõttel oli käsil ühele kliendile erilahendusena liidese ehitamine. Olemasolev protsess eeldas esmalt ettevõttelt müügitellimuse loomist vastavalt kliendi poolt tehtud tellimusele ning seejärel tootmistellimuse koostamist Eziil süsteemis. Üks loodava

rakenduse eesmärke on kõrvaldada ettevõtte poolset tellimuste sisestamist ning valideerimist, andes tellimuse loomise võimaluse kliendile. Ettevõtte jaoks on oluline, et kõik esitatud tellimused järgiksid sarnast malli, mistõttu on vajalik kliendile tellimuse sisestamisel ette anda kindlad valikud. Parimaks lahenduseks siinkohal oleks e-poe lahenduse loomine, mida Eziil tarkvara aga ei paku.

2.3.3 ERPLY

ERPLY on majandustarkvara, mis on suunatud põhiliselt hulgimüügi ja jaemüügi valdkondadele. Tarkvara pakub võimalust valida erinevate komponentide vahel nagu laoarvestus, kassamüük, raamatupidamine ja e-pood [5].

Põhjus, miks antud valmislahendus kliendile ei sobi, on ERPLY sõltuvus laotarkvarast. Kliendi üheks nõudeks on tarkvaralahenduse sõltumatus laotarkvarast. Lisaks eelnevale on tarkvaral hulgaliselt funktsioone, mis antud kliendi puhul rakendust ei leiaks ning oleks seetõttu töötajatele liialt keerukas.

3 Kasutatud tehnoloogiad

See peatükk keskendub töös kasutusel olevate tehnoloogiate tutvustamisele. Iga tehnoloogia kohta antakse lühike ülevaade, võrreldakse teiste sama ülesannet täitvate tehnoloogiatega ja tuuakse välja põhjused, miks just antud tehnoloogia osutus vajalikuks ja valituks.

3.1 Java

Java on platvormist sõltumatu objektorienteeritud tüübikindel programmeerimiskeel. Objektorienteeritus muudab koodi taaskasutatavaks, mis omakorda teeb tarkvaraarenduse vähemkulukaks. Üldjuhul kompileeritakse Java lähtekood baitkoodi, mida saab käivitada Java virtuaalmasinas olenemata sellest, mis platvormiga on tegu. Java hüüdlauseks on „kirjuta korra, käivita ükskõik kus“. Sõltumatus platvormist on üheks põhjuseks, miks Java programmeerimiskeelel on suur toetajaskond. Java sobib nii väiksemahuliste kui ka keerukate veebirakenduse loomiseks. Süntaksi poolest sarnaneb Java C ning C++ keeltele [6].

Antud töös on kasutatud 2019 aasta septembris avaldatud Java 13, mis oli tarkvaraarendust alustades viimane stabiilne versioon. Töö kirjutamise hetkel oli värskeimaks Java versiooniks 2020 aasta märtsis ilmunud Java 14. Java programmeerimiskeel osutus valituks tulenevalt selle stabiilsusest, suurest kasutajaskonnast ja soovist luua rakendus toetudes Spring Boot raamistikule, mis oluliselt kiirendab veebirakenduse arendusprotsessi.

3.2 Spring Boot

Spring on raamistik, mis hõlbustab Java rakenduste arendamist. Raamistiku põhifunktsioone saab kasutada iga Java rakendus. Tegu on väga laialt levinud raamistikuga. Spring raamistikul on mitmed moodulid, milles sisalduvad teenused hõlbustavad oluliselt rakenduse arenduse protsessi [7].

Spring Boot on Spring raamistiku peale ehitatud projekt, mis lihtsustab eraldiseisvate rakenduste loomist, üles seadmist ja käivitamist. Erinevalt Springist, Spring Boot

rakenduse töötamiseks vajaminevad konfiguratsioonid luuakse automaatselt. Tänu sellele puudub vajadus rohkete konfiguratsioonifailide käsitsi kirjutamiseks [8].

Spring moodulitest on töös kasutusel autentimise raames Spring Security ning andmebaasiga suhtlemiseks Spring Data. Otsus kasutada Spring Boot raamistikku tulenes selle omadusest kiirendada ja lihtsustada tarkvaraarenduse protsessi.

3.3 Gradle

Gradle on avatud lähtekoodiga tööriist, mille ülesandeks on tarkvara ehitamise protsessi automatiseerimine. Gradle on kahe varasema tööriista Ant ja Maven järglane ning ühendab kummagi parimaid külgi: Anti konfiguratsiooni paindlikkust ja Maveni konventsioonide järgimist. Erinevalt eelkäijatest, kasutab Gradle konfiguratsioonifailides XML (*Extensible Markup Language*) keele asemel Groovy või Kotlini domeenispetsiifilist keelt, tehes skriptid vähem mahukaks. Gradle töötab Java virtuaalmasinas ja kasutab oma põhifunktsionaalsuse laiendamiseks pluginaid, mis lisavad hulga eeldefineeritud tööülesandeid. Antud projektis on kasutusel Java plugin, mis lisab tööülesanded ühiktestide käivitamiseks, JAR (*Java Archive)* faili loomiseks ja Java lähtekoodi kompileerimiseks. Üheks Gradle ülesandeks on sõltuvuste automaatne haldamine ja seadistamine. Sõltuvused on lisafunktsionaalsust pakkuvad paketid nagu varem mainitud Spring Data ja Spring Security. Sõltuvustele ligi pääsemiseks kasutatakse Ivy ja Maven repositooriume või failisüsteemi. Gradle toetab korraga mitme projekti ehitust, mis on üks selle loomise peamisi põhjuseid [9].

Otsus antud töös kasutada Gradle tööriista tulenes selle võimalusest ehitada rakendus mitmest eri moodulist või projektist, vältides seejuures keerukate XML vormingus konfiguratsioonifailide kirjutamist. Ühtlasi on enamikul juhtudel Gradle suurema jõudlusega, kui Maven [10].

3.4 Vue.js

SPA (*Single Page Application*) ehk üheleherakendus on veebirakendus, mis uude vaatesse minnes ei küsi serverilt vajaminevat lehte vaid kirjutab dünaamiliselt sama veebilehte vastavalt kliendi tegevustele üle. Üheleherakendustes laetakse kõik esikomponendi ressursid kliendi veebibrauserisse ja serverile tehakse päringuid vaid

andmete laadimiseks. Kuna erinevalt mitmeleherakendusest ei laeta iga kasutaja tegevuse korral serverist lehte uuesti, on üheleherakendused kiiremad. Viimaste aastate vältel on üheleherakendused aina enam populaarsust kogunud ning trend on pideval tõusuteel. Üheleherakenduste loomiseks on olemas mitmed JavaScript raamistikud. Käesoleva töö kirjutamise hetkel olid kolm enim tuntud JavaScript raamistikku Angular, Vue.js ja React.

Angular loodi Google'i poolt 2010. aastal ning lisaks Google'ile leiab see kasutust Forbes ja Wix poolt. Angular eristub selgelt teisest kahest raamistikust kuna see põhineb MVC (*Model View Controller*) arhitektuuril ja kasutab virtuaalse DOM-i (*Document Object Model*) asemel päris DOM-i, mis muudab selle teisest kahest raamistikust aeglasemaks. Kuna Angulari rakenduste arhitektuur koosneb rohkematest eri tüüpi komponentidest, on selle õppimine keerukam.

Vue.js on JavaScript raamistik, mis on mõeldud kasutajaliideste ehitamiseks. Vue.js võimaldab tarkvara arendust järk-järgult vastavalt vajadustele. Vue põhiteek keskendub vaid vaate kihile, kuid vajadusel saab hõlpsasti lisada lisafunktsionaalsust pakkuvaid teeke. Kolmest võrreldavast JavaScript raamistikust on Vue kõige väiksema failimahuga ning üldiselt kõige kergemini omandatav tänu väiksemale õppimiskõverale ja detailsele dokumentatsioonile.

React on Facebooki poolt hallatav JavaScript raamistik. Lisaks Facebookile on React kasutusel ettevõtetes nagu Uber, Netflix, Reddit ja PayPal. Reacti abiga on võimalik luua väga suure mahuga rakendusi ja lisaks veebirakendustele ka mobiilirakendusi. React kasutab HTML (*Hypertext Markup Language*) kirjutamiseks JavaScripti laiendust JSX (*JavaScript XML*), mis võimaldab HTML elementide loomise ja DOM-i lisamise JavaScript abil.

Antud töös kasutatakse rakenduses Vue.js raamistikku tulenevalt selle kiiruse eelisest Angulari ees ning HTML-i ja JavaScripti kasutamisest JSX asemel Reactis. Loodav rakendus on kitsa funktsionaalsusega ja väiksemahuline, mistõttu Reacti ega Angulari kasutamine ei olnud otstarbekas.

3.5 Liquibase

Liquibase on andmebaasist sõltumatu avatud lähtekoodiga tarkvara andmebaasi muudatuste jälgimiseks ja haldamiseks. Muudatuste tegemiseks kasutatakse skripte, mida

nimetatakse *changelog*'ideks ning mis koosnevad omakorda üksikutest muudatustest ehk *changeset*'idest. Muudatusi on võimalik kirjutada neljal eri viisil: SQL (*Structured Query Language*), XML, YAML (*YAML Ain't Markup Language*) ja JSON (*JavaScript Object Notation*) [11].

Muudatuste jälgimiseks ja haldamiseks luuakse andmebaasi kaks uut tabelit: „databasechangelog*"* ja „databasechangeloglock*"*. Tabel „databasechangelog*"* sisaldab infot *changeset*'ide kohta – igal real on kirjas muudatuse asukoht muudatuste tegemise järjekorras, faili ja autori nimi, muudatuse id ning indikaator, kas muudatus on käivitatud või mitte. Selleks, et hoiduda konfliktidest, kus mitu eri poolt üritavad teha samaaegselt andmebaasis muudatusi, on kasutusel tabel „databasechangeloglock". Tabeli ülesanne on kindlaks teha, et korraga kasutaks andmebaasi peal Liquibase'i vaid üks osapool.

Rakendust käivitades kontrollib Liquibase, kas kõik olemasolevad muudatused on andmebaasis tehtud ja kui esineb muudatusi, mida ei ole tehtud, käivitatakse vastavate muudatuste tegemine. Liquibase'i kasutamine lihtsustab esialgse andmebaasi üles seadmist, edaspidiste muudatuste tegemist ja andmebaasi kindlasse olekusse viimist. Olukorras, kus andmebaasis on vajalik muudatuste tegemine, tähendaks see reeglina nende tegemist SQL päringute abil otse andmebaasi küljes Kasutades Liquibase'i piisab tarkvaraarendajal vajaminevate *changelog*'ide kirjutamisest ja käivitamisest.

4 Rakenduse arhitektuur

Antud töös püüti tarkvara arendades järgida tarkvaraarenduse häid tavasid, nende hulgas SOLID printsiipe ning REST (*Representational State Transfer*) ja MVC arhitektuuri mustreid. Järgnevalt tutvustatakse loetletud tavasid ja tuuakse välja põhjused nende järgimiseks ning lisaväärtus, mis need tarkvaraarendusele ja tarkvarale toovad.

4.1 SOLID printsiibid

SOLID on lühend kirjeldamaks viite objektorienteeritud programmeerimise disaini printsiipi, mille ülesandeks on parandada koodi loetavust, hooldatavust ja hõlbustada selle edasist arendamist. Robert C. Martin tutvustas SOLID printsiipide teooriat oma 2000. aasta teadusartiklis „Design Principles and Design Patterns“ [12].

4.1.1 Üksiku vastutuse printsiip

Üksiku vastutuse printsiip ütleb, et tarkvaraklassil peab olema vaid üks põhjus muutumiseks. Klass peab täitma kindlat ettenähtud eesmärki. Täites mitut funktsiooni korraga tekib suurema tõenäosusega olukord, kus antud klassis olevad meetodid sõltuvad teineteisest. Tulenevalt meetodite vahelistest sõltuvustest põhjustab vajadus muuta äriloogikat tihti mitmete muudatuste tegemist klassis. Nende muudatuste tegemine on tarkvaraarenduses keerukas ja aeganõudev protsess. Tarkvarakomponentide vähene sõltuvus parandab nende testitavust vähendades teststsenaariumeid. Kompaktsete moodulite ja klasside kasutamine tingib tarkvaraprojekti parema loetavuse ning organiseerituse. Väikeste ja konkreetset ülesannet täitvate klasside haldamine on lihtsam [13].

4.1.2 Avatud-suletud printsiip

Avatud-suletud printsiibi kohaselt peavad tarkvara olemid (klassid, meetodid, moodulid) olema sellise struktuuriga, et neid on võimalik laiendada (lisada funktsionaalsust) ilma olemasolevat koodi muutmata. Koodi defektsus peaks olema ainus põhjus selle muutmiseks. Printsiibi järgimine väldib olukordi, kus töötava koodi muutmisel tekivad uued vead, mis vajavad edasist parandamist [14].

4.1.3 Liskovi asendusprintsiip

Liskovi asendusprintsiip ütleb, et alamklassid tuleb kirjeldada nii, et kui klass A on klassi B alamklass, ei oma kasutaja jaoks tähtsust, kui vahetada klass B alamklassi A vastu – programm peab olema võimeline probleemideta edasi töötama. Kui alamklass oskab teha midagi, mida ülemklass ei oska, on see otsene printsiibi vastu eksimine [15].

4.1.4 Liideste eraldatuse printsiip

Objektorienteeritud tarkvaradisainis luuakse liideste abil abstraktsiooni kihid, mis lihtsustavad koodi ja aitavad hoiduda sõltuvuste sidumisest. Süsteem võib jõuda olukorda, kus see on niivõrd sidus, et enam ei ole võimalik teha muudatusi vaid ühes kohas. Liideste eraldatuse printsiibi kohaselt ei tohi eksisteerida sõltuvusi meetoditest, mida ei kasutata. Palju meetodeid sisaldavad suured liidesed jaotatakse väiksemateks ja spetsiifilisemateks. Printsiibi järgimine aitab hoida süsteemi lahti seotuna, mis hõlbustab tulevikus koodi redigeerimist [16].

4.1.5 Sõltuvuse inversiooni printsiip

Sõltuvuse inversiooni printsiibi kohaselt peaksid tarkvarakomponendid sõltuma abstraktsioonidest, mitte konkreetsetest klassidest ja detailid peaksid sõltuma abstraktsioonidest, mitte vastupidi. Keerukat loogikat sisaldavad kõrgema taseme moodulid peaksid olema kergesti taaskasutatavad ja sõltumatud abifunktsioone pakkuvatest madalama taseme moodulitest. Abstraktsioonide kasutamine hõlbustab komponentide vahetamist [13].

4.2 REST arhitektuuri reeglid

REST on tarkvaraarhitektuuri stiil, mis defineerib reeglid, millele toetudes veebirakendusi luuakse. Reeglitele vastavaid veebirakendusi kutsutakse RESTful veebirakendusteks. REST taotleb ühtset liidest ja püüab seeläbi suurendada jõudlust, muudetavust, töökindlust ja porditavust. Terminit REST tutvustas esmakordselt oma doktoritöös Roy Fielding aastal 2000. Doktoritöös tõi Roy Fielding välja järgmised viis põhilist REST printsiipi [17].

4.2.1 Klient-server printsiip

Klient-server printsiibi eesmärk on vastutusalade jagamine, sarnanedes põhimõttelt SOLID üksiku vastutuse printsiibiga, eraldades kliendi ja serveri. Antud lahendus võimaldab arendada *front end* ja *back end* rakendust eraldi osadena, mis lihtsustab kasutajaliidese teisaldamist eri platvormide vahel. Tänu sellele ei oma tähtsust, mis platvormil *front end* rakendus on realiseeritud. Peale selle toob vastutusalades jagamisest tulenev serveri komponentide lihtsustamine kaasa rakenduse parema skaleeruvuse ehk süsteemi hakkama saamise nii väga suurte kui ka väikeste koormustega ilma tõhususe- ja kvaliteedilanguseta [17].

4.2.2 Olekuta printsiip

Olekuta printsiip eeldab serverist andmete kätte saamiseks tehtavatelt päringutelt kogu vajamineva info olemasolu. Ükski uus päring ei ole sõltuvuses eelnevatest ja serverisse ei talletata infot kliendi poolt tehtud päringute kohta. Tulemuseks on rakenduse suurem jõudlus ja parem skaleeruvus kuna serveri ressursse ei kasutata päringute info talletamiseks [17].

4.2.3 Vahemälu printsiip

Olekuta API (*Application Programming Interface)* ehk rakendusliidese loomusest tulenevalt tehakse palju ühesuguseid päringuid. Tihti vajaminevate päringute vastuste salvestamise võimalus aitab oluliselt hoida aega kokku tehtavate päringute arvult. Vahemälu printsiip nõuab päringu vastuses märgistamist, kas selles sisalduvad andmed talletatakse vahemällu või mitte. Enne iga päringut kontrollib rakendus, kas antud päringu vastus on juba vahemällu salvestatud. Vahemälust vastuse puudumise korral tehakse serverile vajaminev päring [17].

4.2.4 Ühtse liidese printsiip

REST üheks põhiprintsiibiks on ühtse liidese printsiip. Erinevad rakendused peavad saama API-ga suhelda ühte moodi sellest ka nimetus ühtne liides. Kasutusel olev programmeerimiskeel või seade, mis API-le päringuid saadab ei tohiks muuta päringu tegemise viisi. Ühtse liidese saavutamiseks peab täitma mitmeid arhitektuurile seatud reegleid. REST seab liidesele neli kitsendust:

- Ressursside tuvastamine päringus – ressurssidele ligi pääsemiseks kasutatakse URI-sid (*Uniform Resource Identifier*). Näiteks */rest/roman-curtain-orders/3*.

- Esituse järgi ressurssidega manipuleerimine – ressurssidele päringuid tehes saame kindlas formaadis esituse. Enim levinud formaatideks on JSON (JavaScript Object Notation) ning XML. Päring peab sisaldama kirjeldust soovitud andmeformaadi kohta. Omades vajaminevaid õiguseid, on päringu vastuses saadav ressursside esitus küllaldane nendega manipuleerimiseks (muutmine, kustutamine).

- Ennast kirjeldavad sõnumid iga sõnum peab sisaldama selle töötlemiseks vajaminevat infot. Näiteks mis kujul andmeid päringu vastusena oodatakse.

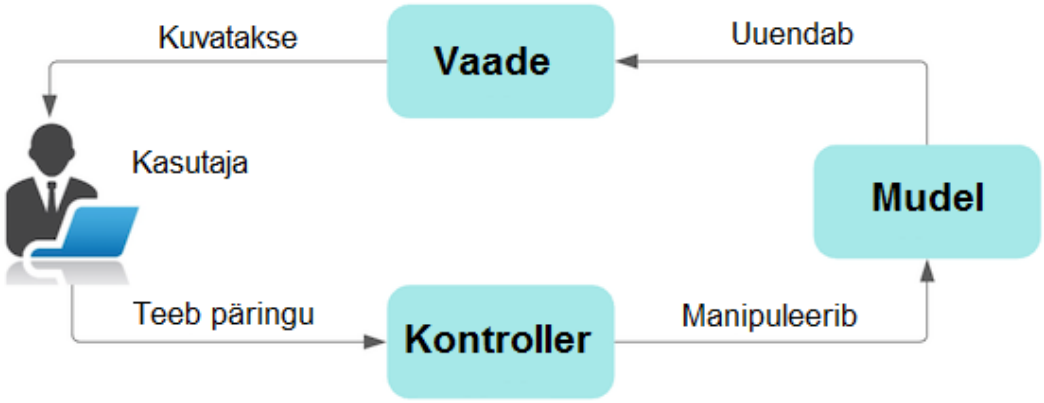
- HATEOAS (*Hypermedia as the Engine of Application State)* lisaks andmetele võivad päringu vastused sisaldada nendega lubatud tegevusi [18].

4.2.5 Kihtide süsteemi printsiip

Kihtide süsteemi printsiip näeb ette API ehitamist hierarhiliste kihtidena, mis suhtlevad vahetult üleval ja all olevate kihtidega Taolist lähenemist kasutades saab hõlpsasti lisada serveri ning kasutajaliidese vahele teisi kihte, näiteks turvalisuse kiht või suuremate rakenduste korral koormuse tasakaalustamise kiht. Üheks enim levinud kihtide arhitektuuriks on MVC [17].

4.3 MVC

MVC on laialdaselt kasutatav tarkvara disaini muster, mis jagab terve rakenduse kolmeks kihiks mudel, vaade ja kontroller (Joonis 2).



Joonis 2. MVC arhitektuur [19].

Mudeli kiht on *back end* rakenduse kõige alumine kiht. See suhtleb andmebaasiga, manipuleerib andmetega, rakendab äriloogikat ja saadab vajalikud andmed kontrolleri kihile. Vaate kiht saadab kontrolleri kihile päringuid ja visualiseerib saadud andmed. Kontrolleri kiht on vahelüli vaate ja mudeli vahel. MVC disainimustri peamiseks eesmärgiks on vastutusalade jagamine, mis võimaldab rakenduse eri kihtide samaaegset arendamist.

5 Rakenduse arenduskäik

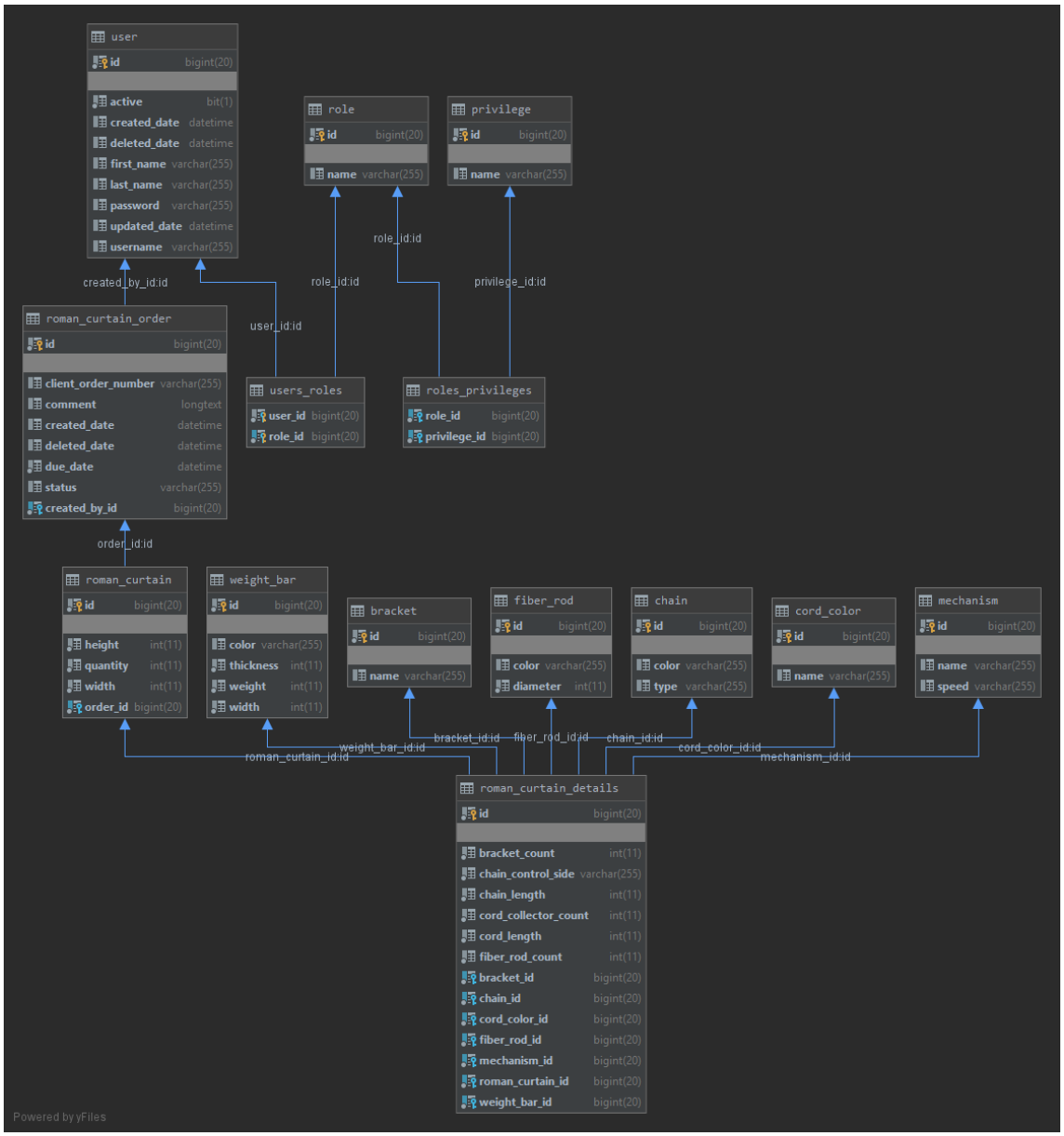
Käesolev peatükk kirjeldab detailselt veebirakenduse arenduskäiku. Tutvustatakse kasutatud arendusmeetodeid, loodud funktsionaalsusi ja nende realiseerimisviisi ning tuuakse visuaalseid näiteid valminud rakenduse eri osadest.

5.1 Üldine arenduskäik

Tarkvaraarenduse versioonihalduseks kasutati GitLab keskkonda. Kliendiga kinnitatud kasutusjuhtude põhjal loodi GitLab keskkonnas ülesanded, mis seati tähtsuse järjekorda. Ülesandeid püüti luua kitsa skoobiga vältimaks liiga pikalt ühe ülesande töös hoidmist. Lahendades ühte ülesannet pikalt on kerge selle põhisisust kõrvale kalduda Tehtava töö haldamiseks kasutati agiilset *Kanban* meetodit, mille keskseks ideeks on töö ja tööjõu balansseerimine. *Kanban* toetub visuaalsele protsesside juhtimisele, kasutades selleks *Kanban* tahvlit, millel on fikseeritud reeglid. Korraga hoitakse töös minimaalne arv ülesandeid ja luuakse reeglid ülesande liikumiseks *Kanban* tahvli ühest veerust teise. Arendusprotsessi käigus tekkis ülesandeid juurde, mis tingis mõningatel juhtudel prioriteetide ümber hindamise ja seega ülesannete täitmise järjekorra muutmise [20].

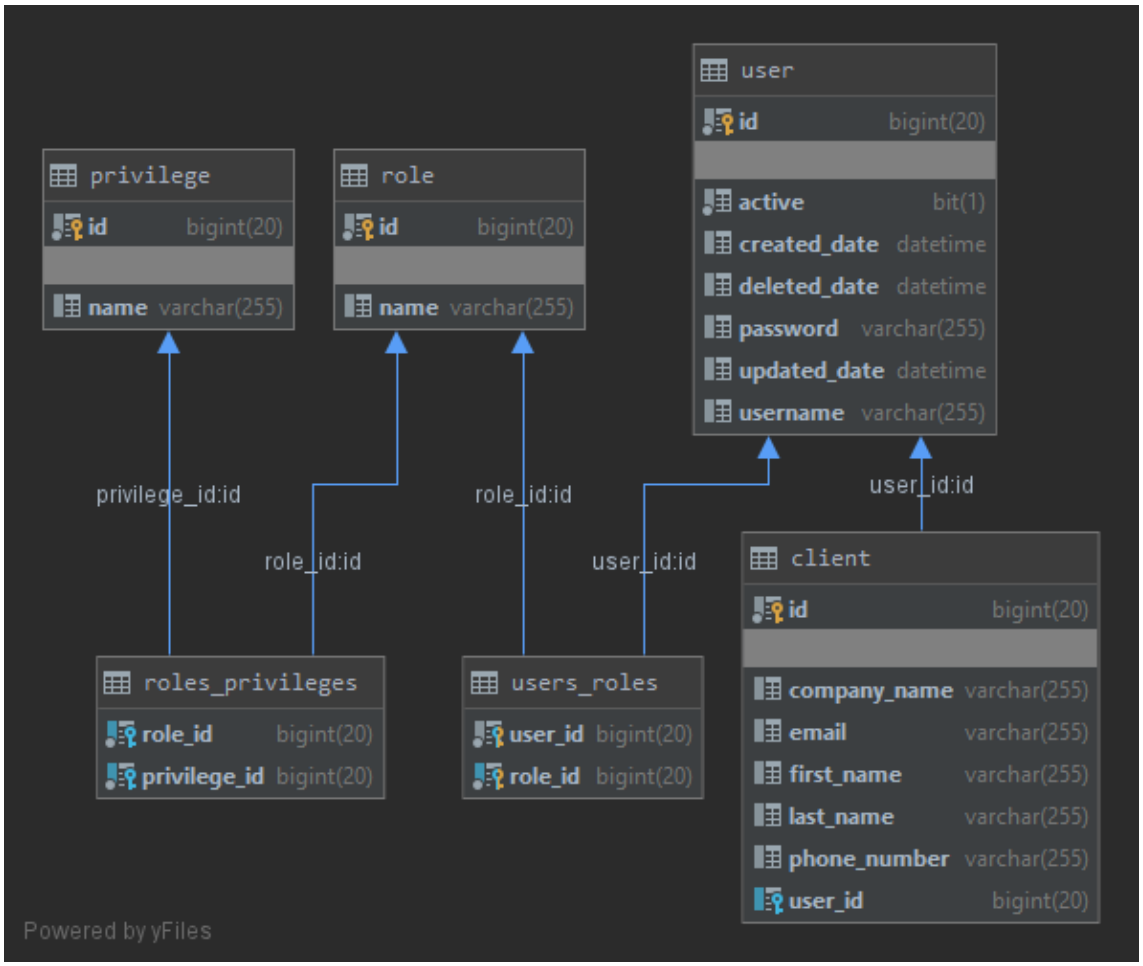
5.2 Andmemudeli loomine ja algandmete andmebaasi sisestamine

Arendusprotsessi esimese etapina loodi rakenduse andmemudel (Joonis 3). Andmemudeli saab näiliselt jagada kaheks: kasutajaga seotud klassid ja roomakardina tellimusega seotud klassid.



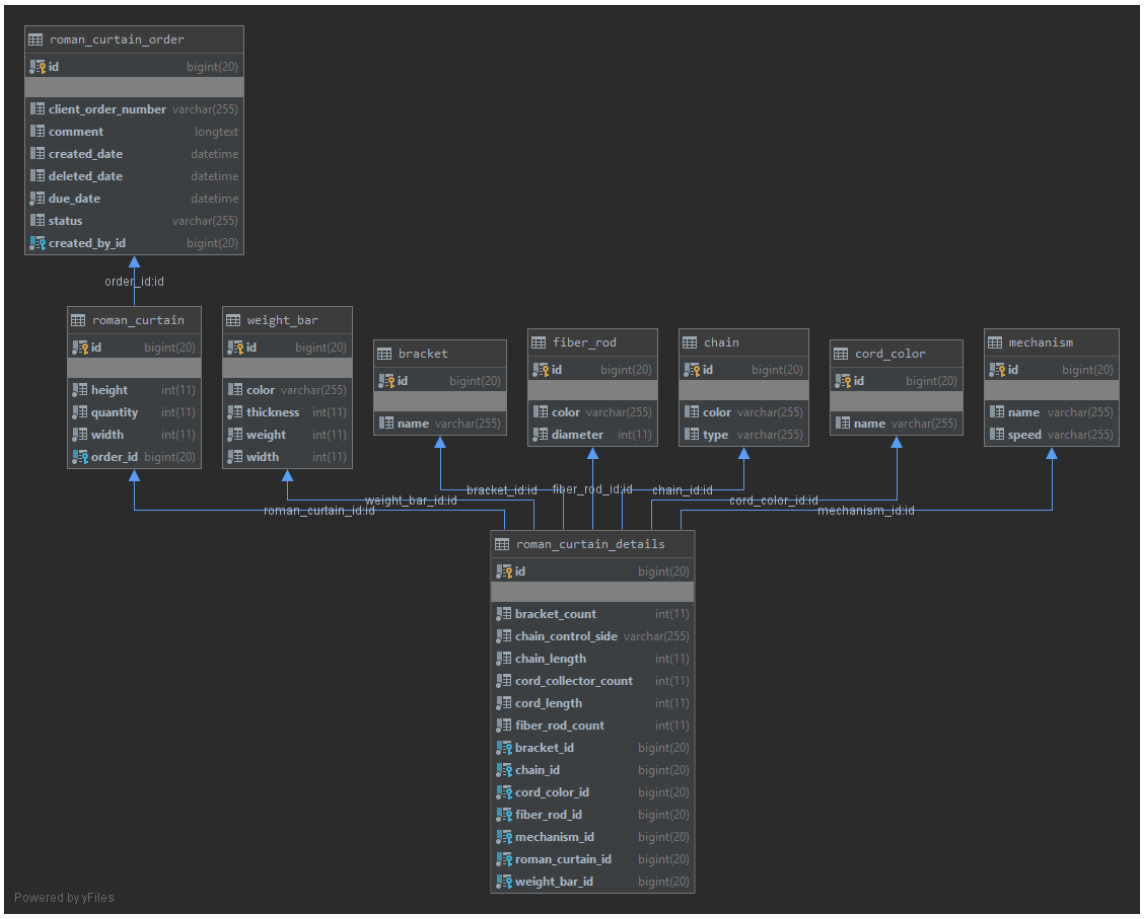
Joonis 3. Rakenduse andmemudel.

Igal kliendil on kasutaja, millel on rollid. Igal rollil on kindlad privileegid. Kliendi ja kasutaja andmed hoitakse kahes eraldi tabelis eraldamaks eri funktsionaalsusega andmeklasse teineteisest. SOLID üksiku vastutuse printsiip ütleb, et iga klass peaks tegema ühte asja. Eraldiseisvate tabelitena on vastutusalad selgelt jagatud – tabel *User* hoiab kasutaja autentimiseks vajalikku infot ja tabel *Client* hoiab kogu ülejäänud infot. Kasutajaga seotud tabelid on nähtavad joonisel 4.



Joonis 4. Kasutajaga seotud tabelid.

Roomakardina tellimuse on seotud vähemalt ühe roomakardinaga. Roomakardina mehhanism koosneb detailidest – juhtimismehhanism, kett, nööririullid, raskuslatt, vahevardad ja kinnitused. Kõiki nimetatud detaile on eri tüüpi ning igal detailil on andmebaasis vastav tabel, mis sisaldab antud detaili kõiki variatsioone. Näiteks keti jaoks on tabel *Chain*, kus igal ketil on kaks välja – keti tüüp ning värvus. Ketid jagunevad tüübi alusel plastik- ja metallkettideks. Andmemudeli tasemel on iga roomakardina tellimus *RomanCurtainOrder* seotud vähemalt ühe roomakardinaga *RomanCurtain*. Roomakardinat esindav klass *RomanCurtain* sisaldab endas roomakardinat puudutavat üldinfot – kardina laius, kõrgus ja kogus. Detailide jaoks on eraldi klass *RomanCurtainDetails*, kus on seosed detailide klassidega ning muu neid puudutav info. Iga roomakardin on seotud täpselt ühe roomakardina detailide klassiga. Mainitud andmemudel on nähtav joonisel 5.



Joonis 5. Roomakardina tellimuse tabel koos temaga seotud tabelitega.

Andmemudeli loomisele järgnes tarkvaraarenduseks vajalike algandmete sisestamine andmebaasi, mille hulka kuulusid roomakardina detailid ning esialgsed klientide, kasutajate, rollide ja privileegide andmed. Andmete sisestamiseks andmebaasi loodi *data* paketis asuv *DataInitializer* klass. Rakendust käivitades pannakse tööle klassi *run* meetod (Joonis 6). Meetod sisaldab teiste klasside meetodite väljakutseid – kasutaja algandmete sisestamiseks on *UserDataInitializer* ja roomakardina detailide lisamiseks *RomanCurtainDataInitializer* klass.

```
@Component
public class DataInitializer implements CommandLineRunner {
    @Autowired
    private UserDataInitializer userDataInitializer;

    @Autowired
    private RomanCurtainDataInitializer romanCurtainDataInitializer;

    @Override
    public void run(String... args) {
        userDataInitializer.initialize();
        romanCurtainDataInitializer.initialize();
    }
}
```

Joonis 6. Algandmeid sisestav klass.

Joonisel 7 on näide roomakardina kinnituste andmete sisestamisest andmebaasi. Meetod *initialize* kirjutab andmebaasi kõigi võimalike detailide andmed. Meetod *createDefaultBracketsIfNotSet* kontrollib, kas tabel *Bracket* on tühi. Kui andmebaasi tabel on tühi, kirjutatakse andmebaasi kõik vajalikud kinnituse tüübid. Sama loogikat rakendatakse ülejäänud roomakardina detailide, klientide, kasutajate, rollide ja privileegide salvestamisel.

```
public void initialize() {
    brackets = createDefaultBracketsIfNotSet();
}

private List<Bracket> createDefaultBracketsIfNotSet() {
    if (bracketRepository.count() > 0) {
        return bracketRepository.findAll();
    }

    List<Bracket> brackets = new ArrayList<>();
    {
        Bracket bracket = new Bracket();
        bracket.setName("laekinnitus");
        brackets.add(bracketRepository.save(bracket));
    }
    {
        Bracket bracket = new Bracket();
        bracket.setName("universaalne kinnitus");
        brackets.add(bracketRepository.save(bracket));
    }

    return brackets;
}
```

Joonis 7. Roomakardina kinnituste sisestamine andmebaasi.

Antud lahendust andmete lisamiseks andmebaasi kasutati vaid algandmete lisamisel, sest andmebaasitabelite loomiseks ja esialgsete andmete sisestamiseks oli see lahendus ajaliselt kiirem. Edaspidisteks muudatuste tegemiseks on kasutusel Liquibase tarkvara, mis muudab muudatuste tegemise tunduvalt lihtsamaks.

5.3 Turvalisus ja autentimine

Süsteemi kasutamiseks tuleb külastajal ennast autentida. Kliendi ja serveri vahelises suhtluses kasutatakse JWT-d (JSON Web Token). JWT on standard, mis defineerib lihtsa ja kompaktse viisi kahe osapoole vahel turvaliselt info edastamiseks. JWT allkirjastatakse digitaalselt, mis muudab selle kontrollitavaks ja usaldusväärseks. Kui klient ei ole keskkonda sisse logitud, kuvatakse sisse logimise vorm (Joonis 8).

Joonis 8. Rakendusse sisse logimise vorm.

Püüdes süsteemi siseneda saadetakse *back end* rakendusele päring. Esmalt kontrollitakse JWT olemasolu päringu päises. Selle puudumisel püütakse kasutajat autentida sisestatud kasutajanime ja parooli alusel. Vastava kasutaja olemasolul genereeritakse kasutajale JWT (Joonis 9).

```
public String generateToken(UserDetails userDetails) {
    Map<String, Object> claims = new HashMap<>();
    return doGenerateToken(claims, userDetails.getUsername());
}

private String doGenerateToken(Map<String, Object> claims, String subject) {
    return Jwts.builder()
            .setClaims(claims)
            .setSubject(subject)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + expiration))
            .signWith(SignatureAlgorithm.HS512, secret)
            .compact();
}
```

Joonis 9. JWT genereerimine serveri poolel.

JWT-d luues allkirjastatakse see serveri poolel asuva võtmega, mis tingib võimaluse selle verifitseerimiseks ja usaldamiseks. JWT genereerimise järel logitakse kasutaja süsteemi sisse ja tagastatakse päringu vastuses genereeritud JWT, mille *front end* rakendus kliendi brauseri vahemällu salvestab (Joonis 10).

```
class AuthService {
    login(user) {
        return axios
            .post(API_URL + "/login", {
                username: user.username,
                password: user.password
            })
            .then(response => {
                if (response.data.accessToken) {
                    localStorage.setItem("user",
JSON.stringify(response.data));
                }
                return response.data;
            });
    }

    logout() {
        localStorage.removeItem("user");
    }
}
```

Joonis 10. Teenus kasutaja sisse ning välja logimiseks front end rakenduses.

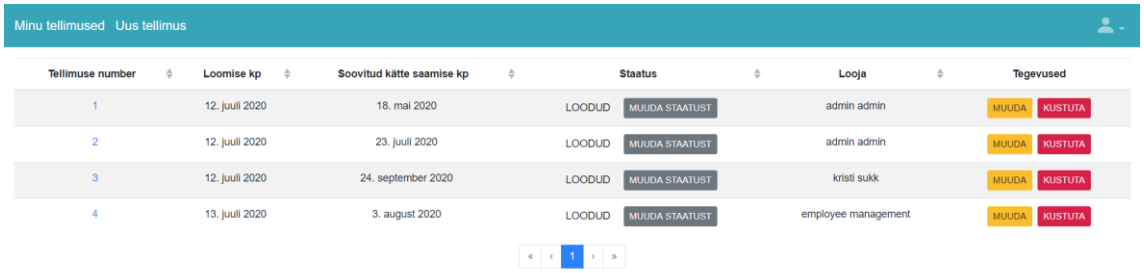
Edaspidi lisatakse genereeritud JWT päringu päisesse(Joonis 11). Selle aegudes genereeritakse uus.

```
getOrder(id) {
    return axios.get(`${API_URL}/${id}`, {headers: authHeader()});
}
```

Joonis 11. Päring, kus päringu päisesse on lisatud JWT.

5.4 Tellimuste nimekirja kuvamine

Rakendusse sisenedes avaneb kasutajale tellimuste vaade (Joonis 12). Firma töötajale on nähtavad kõik aktiivsed tellimused, kliendile vaid enda tehtud aktiivsed tellimused. Kuna tellimuste arvu kasvades muutuvad päringud mahukamaks ja seega aeglasemaks, küsitakse tellimusi *back end* komponendilt lehekülje haaval. Kiirem ja otstarbekam on laadida vaid kuvamist vajavad andmed. Tellimuste tabel on veergude järgi sorteeritav, vajutades vastava veeru pealkirjale. Joonisel 13 on näha, kuidas *back end* rakenduses tellimusi lehthaaval vastavalt kasutaja rollile laaditakse.



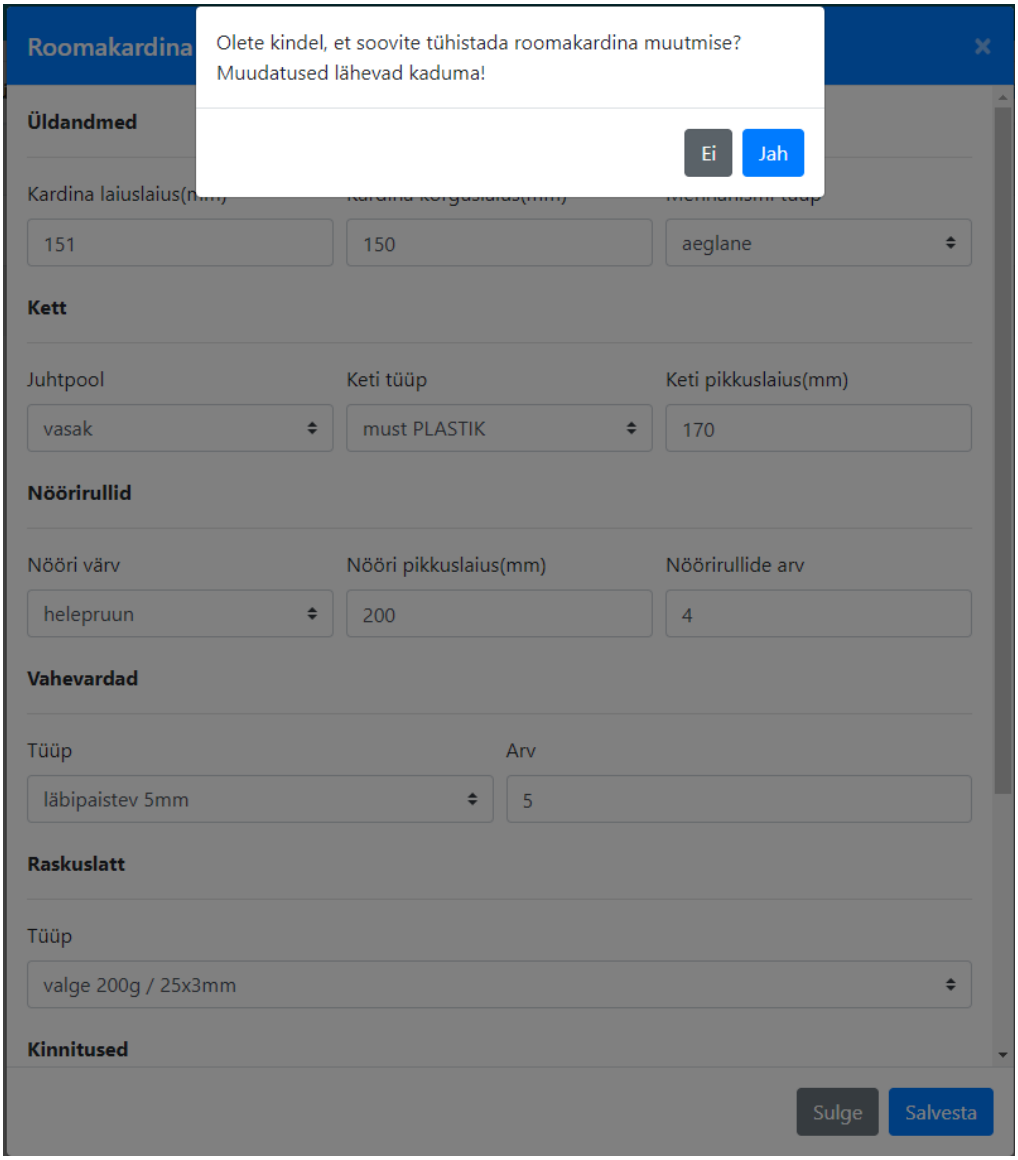
Joonis 12. Tellimuste nimekirja vaade.

```
@Override
public Page<RomanCurtainOrderDisplayDTO> getAll(Pageable pageable) {
    Page<RomanCurtainOrder> romanCurtainOrderPage;
    if (authContextManager.hasRole(
            "ROLE_EMPLOYEE_WAREHOUSE") ||
authContextManager.hasRole("ROLE_EMPLOYEE_MANAGEMENT")) {
        romanCurtainOrderPage =
romanCurtainOrderRepository.findByDeletedDateIsNull(pageable);
    } else {
        User user = authContextManager.getLoggedInUser();
        romanCurtainOrderPage =
romanCurtainOrderRepository.findByDeletedDateIsNullAndCreatedBy(pageable,
user);
    }
    Page<RomanCurtainOrderDisplayDTO> dtoPage =
romanCurtainOrderPage.map(order -> modelMapper.map(order,
RomanCurtainOrderDisplayDTO.class));
    return dtoPage;
}
```

Joonis 13. Teenus kõikide tellimuste laadimiseks.

5.5 Tellimuse muutmine

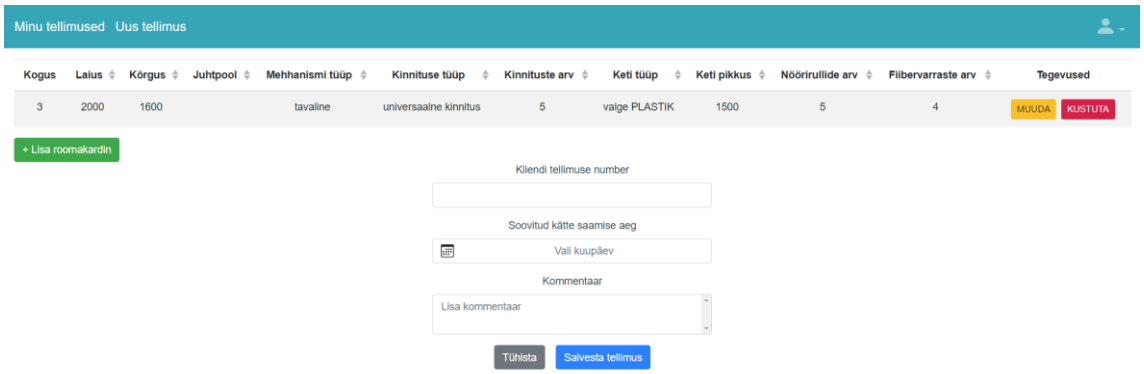
Tellimuste tabeli vaates on igal tellimuse real veerg „Tegevused“, kus on nupud: „MUUDA“ ja „KUSTUTA“. Tellimuse info muutmiseks peab kasutaja vajutama nupule „MUUDA“. Vajutades sellele nupule avaneb tellimuse muutmise vaade, kus saab tellimust muuta. Tellimuses muudatusi tehes valideeritakse muudetud andmed. Üheks kliendi poolt seatud nõudeks on kasutaja kinnituse küsimine suuri tellimusega seotud muudatusi tehes, mida võib näha joonisel 14.



Joonis 14. Muudatustest loobudes kasutajalt kinnituse küsimine.

5.6 Tellimuse loomine

Vajutades veebirakenduse päises asuvale nupule „Uus tellimus“, navigeeritakse kasutaja uue tellimuse loomise vaatesse (Joonis 15). Vaade jaguneb kaheks osaks: tellimuse üldinfo ja roomakardinate tabel.



Joonis 15. Tellimuse loomise vaade.

Üldinfo koosneb kolmest väljast: kliendi tellimuse number, kauba kättesaamise aeg ja kommentaar. Kauba kättesaamise aeg saab olla kas samal päeval või tulevikus. Enamjaolt lisavad kliendid tellimusi tehes oma tellimuse numbri, mis tuleb kaupa pakendades kirjutada pakile. Kuna on erandeid, kus tellimuse number puudub, ei ole vormis antud väli kohustuslik. Kommentaari lahter on mõeldud kliendi poolt sellise lisainfo sisestamiseks, mis aitaks ettevõttel vastavat tellimust paremini täita. Näiteks võib esineda stsenaarium, kus klient tellib kollast värvi nööridega kardina, kuid selle puudumisel on nõus ka valget värvi nööridega. Sellisel juhul lisab klient tellimust tehes vastava kommentaari. See aitab hoida kokku nii kliendi kui ka ettevõtte väärtuslikku aega, sest muidu peaks ettevõte helistama kliendile ja selgitama, et hetkel on nöör otsas ning klient peaks omakorda seletama, et ka teine värv on sobiv.

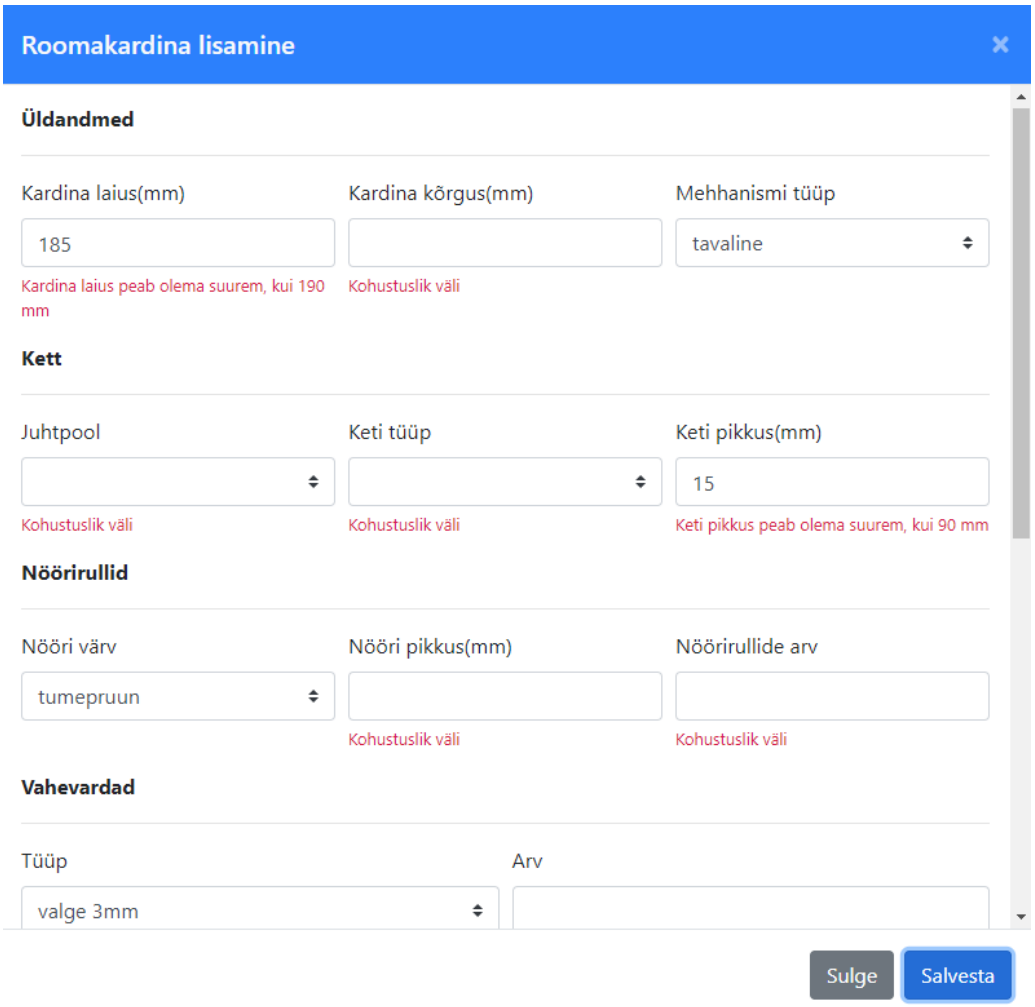
Roomakardinate sisestamise osa koosneb tabelist ja hüpikaknast, mis avaneb vajutades nupule „Lisa roomakardin“. Hüpikakna vaade jaguneb detailide järgi alamosadeks. Valideerimine toimub nii *back end*, kui ka *front end* rakenduses. *Back end* komponendis kasutatakse valideerimiseks sisseehitatud annotatsioonide kaudu töötavat valideerimist. *Front end* kasutab valideerimiseks VeeValidate [21] teeki. VeeValidate pakub funktsionaalsust sisseehitatud reeglite abil põhiliste väljatüüpide valideerimiseks ja annab kasutajale võimaluse kirjutada uusi või muuta juba olemasolevaid valideerimisreegleid.

Roomakardina mehhanismi lisamise vormis on kasutusel sisseehitatud valideerimisreegel, mis on muudetud dünaamiliseks – veateade sisaldab endas valideeritava välja nime ja välja numbrilise väärtuse alampiiri (Joonis 16), mis on valideeritaval väljal defineeritud.

```
extend("positive", {
    validate: (value, args) => {
        return value > Number(args.number);
    },
    params: ["number"],
    message: "{_field_} peab olema suurem, kui {number} mm"
});
```

Joonis 16. VeeValidate reegel numbrilise väärtuse dünaamiliseks valideerimiseks.

VeeValidate teek valideerib vormi sisendeid jooksvalt. „Salvesta“ nupule vajutades kontrollitakse, kas vormis esineb probleeme ja nende puudumisel saadetakse päring *back end* rakendusele. Joonisel 17 on näide vormi valideerimisel kuvatavatest vigadest.



Joonis 17. Roomakardina lisamise hüpikaken.

5.7 Tellimuse staatuse muutmine

Olles sisse logitud kasutajaga, kelle rolliks on *ROLE_EMPLOYEE_WAREHOUSE* või *ROLE_EMPLOYEE_MANAGEMENT*, on võimalik muuta tellimuste staatust, välja arvatud juhul, kui tellimuse staatus on „VALMIS“. Vajamineva rolli olemasolu kontrollimine toimub Spring Security sisseehitatud annotatsiooni *@PreAuthorize* abil. Annotatsioon on kasutav nii kontrolleri kui ka meetodi tasemel. Antud rakenduses kontrollitakse rolli olemasolu juba kontrolleri kihis välistamaks rollide puudumisest tuleneva vea võimalikult varakult.

5.8 Tellimuse detailvaade

Tellimuse detailvaate avamiseks vajutab kasutaja tellimuse nimekirja vaates olles tellimuse numbrile, mille detailvaade soovitakse avada. Detailvaates on vastava tellimuse info muutmatul kujul. Selles vaates saab genereerida tellimuse põhjal faili ja selle alla laadida. Faili genereerimiseks ja alla laadimiseks on kasutusel teek SheetJS [22]. Teegi abil luuakse valitud formaadis fail (toetatud faililaiendid on kirjas dokumentatsioonis), lisatakse sellele uus tühi leht, täidetakse leht vajaminevate andmetega ja laetakse kasutaja brauseris alla (Joonis 18).

```
downloadOrder() {
    // create workbook
    let workbook = XLSX.utils.book_new();

    const fileName = this.generateFileName();
    const orderHeader = this.generateFileOrderHeader();
    const curtains = this.generateFileCurtainTable();

    // create excel sheet with order header
    let sheet = XLSX.utils.json_to_sheet(orderHeader);

    // add roman curtain table data to sheet starting from cell A5
    XLSX.utils.sheet_add_json(sheet, curtains, {origin: "A5"});

    // attach sheet to workbook
    XLSX.utils.book_append_sheet(workbook, sheet, "1");

    // download excel file
    XLSX.writeFile(workbook, `${fileName}.xlsx`);
}
```

Joonis 18. Tellimuse Excel faili genereerimine ja alla laadimine.

6 Tehtud töö tulemus

Töö tulemusena valmis algne roomakardinatellimuste loomise ja haldamise veebirakendus. Rakendus on algne, sest töö kirjutamise hetkeks oli klient avaldanud soovi edasiarendusteks, mida esialgselt seatud nõuetes ei eksisteerinud. Loodud rakenduse kasutusele võtmise tulemusena liigub roomakardinatellimuste valideerimise vastutus inimestelt süsteemile. See aitab vähendada nii kliendi kui ka ettevõtte töötajate poolt tehtavate vigade arvu ja muudab tellimuse töötlemisprotsessi efektiivsemaks. Kõik ettevõttele tehtud tellimused on ühest kohast kättesaadavad, mis tagab parema ülevaate töömahust ja aitab paremini planeerida ettevõtte tegevust. Kõigi tellimuste info hoidmine keskses kohas loob võimaluse tulevikus nende analüüsimiseks ja selle põhjal ettevõtte müügi- ja tootmisprotsessi parandamiseks.

Kliendiks oleva ettevõtte töötajad valideerisid valminud rakendust. Valideerimise tulemusena leiti peamiselt võimalusi kasutusmugavuse parandamiseks. Eelkõige hõlmas see tellimuste tabeli vaadet, kus klient avaldas vajadust parema tabeli filtreerimise ja otsingu funktsiooni järgi. Tellimuste arv kasvab jõudsalt ja nende käsitsi otsimine on tülikas ning aeganõudev. Üldjoones jäi klient valminud rakendusega rahule, sest see pidas kinni seatud nõuetest. Positiivselt poolelt toodi välja, et rakenduse kasutamine on lihtne ja arusaadav ka arvutikaugele inimesele. Märgiti, et roomakardina lisamise hüpikaken on hästi realiseeritud, arvestades info hulka, mida see sisaldab. Kriitikana toodi välja rakenduse välimus, mis kliendi arvates vajab lihvimist.

7 Võimalikud edasiarendused

Valminud rakendust valideerides tekkis nii kliendil kui ka töö autoril ideid, kuidas rakendust paremaks muuta või sellele lisaväärtust anda. Järgnevalt on kirjeldatud ära põhilised võimalikud edasiarenduse suunad.

7.1 Tugi teist tüüpi toodete tellimuste haldamiseks

Kuna töös kirjeldatud probleem esineb ka ettevõtte poolt müüdavate teiste toodetega, on potentsiaali laiendada rakenduse funktsionaalsust, lisades juurde toe teist tüüpi toodete ja tellimuste tegemiseks. Lisatavad tooted on oma olemuselt üpris erinevad ja seega vajaksid teatavaid lisafunktsionaalsusi nagu näiteks rakendusse jooniste failide lisamine või rakenduses endas jooniste tegemine.

7.2 Tootevaliku haldamise funktsionaalsus

Klient on avaldanud soovi tootedetailide haldamiseks rakenduses. Praeguse lahenduse korral peab töö autor ise käsitsi tegema vajaminevad muudatused andmebaasis. Klient tõi välja, et baasrakenduses ei nähtud vajadust antud funktsioonile, sest konkreetse toote (roomakardina) detailide valik ei ole aastate vältel praktiliselt üldse muutunud. Antud funktsionaalsus oleks aga kindlasti vajalik juhul, kui rakendusse lisatakse tugi teist tüüpi toodete tellimiseks, sest nende tootevalikus toimub muudatusi tihemini.

7.3 Rakenduse kasutajakontode haldamise funktsionaalsus

Kuigi infosüsteemi nõudeid kirja pannes leidis ettevõte, et kasutajakontode haldamine ei ole nende jaoks prioriteet, jõuti rakenduse valideerimise käigus järeldusele, et kasutajate haldusega peaks siiski tegelema ettevõte ise, vältimaks sõltuvust ettevõttevälistest inimestest (töö autor). Sellest tulenevalt näeb ettevõte vajadust luua kasutajaliideses vastav funktsionaalsus, kus saab uusi kasutajaid luua, olemasolevaid kasutajaid aktiveerida ja deaktiveerida ning kasutajate õiguseid ja infot muuta.

7.4 Tootmistöötaja tööülesannete täitmine rakenduse abiga

Hetkel kasutavad ettevõtte tootmistöölised tellimuste töötlemiseks välja prinditud versiooni tellimusest. Enesekontrolliks teevad töötajad tellimuslehtedele märkmeid. Suurema tellimuse korral märgitakse ära iga kardin, mis on komplekteeritud ja lugedes suures koguses detaile tehakse abistavaid märkmeid lugemise hõlbustamiseks ja enese kontrollimiseks. Tulevikus soovib ettevõte vähem paberit kasutada ja näeb tootmistöölisi kasutamas paberi asemel arvuteid ning seetõttu vajab rakendus lisaarendusi, mis keskenduksid tootmistöötajale mugava funktsionaalsuse loomisele.

8 Kokkuvõte

Töö eesmärgiks oli parandada ettevõtte roomakardinatellimuste loomise ja tootmise protsessi. Eesmärgi saavutamiseks otsustati kasutusele võtta veebirakendus, mis valideeriks kasutaja sisendi õigsust ja muudaks tellimuse töötlemist sujuvamaks. Selleks defineeriti koos kliendiga rakenduse nõuded ja analüüsiti turul olevaid tarkvaralahendusi ja nende sobivust kliendi poolt defineeritud nõuetega. Tulenevalt sobiva valmislahenduse puudumisest, loodi töö tulemusena roomakardinatellimuste infosüsteem juhindudes headest tarkvaraarendustavadest ja kliendi nõuetest. Loodud veebirakenduses on võimalik kliendil luua, muuta ja jälgida tehtud roomakardinatellimusi. Ettevõte saab veebirakenduse abil valideerida, töödelda ja koondada tellimused kindlal kujul ühte kesksesse kohta. Viimane võimaldab ettevõttel kogutud andmete põhjal analüüsida ja planeerida oma tegevust.

Põhilisteks raskusteks osutusid JWT toe lisamine, andmemudeli loomine ja mitmete uute tehnoloogiate omandamine. Töö koostamisel sai autor hulgaliselt uusi teadmisi tarkvaraarenduse protsessi haldamisest. Lisaks õppis autor häid tarkvaraarenduse tavasid ja sai võimaluse nende koheseks rakendamiseks.

Tööle seatud nõuded said täielikult täidetud ja kliendi rakenduse valideerimisest saadud tagasiside oli positiivne. Rakendust valideerides tulid ilmsiks funktsionaalsused, mis peaksid eksisteerima enne rakenduse täielikku kasutuselevõttu kuid mida esialgsetes nõuetes ei olnud. Pärast vajaminevate lisaarenduste valmimist vajab rakendus veelkord ettevõtte ja seejärel ettevõtte eri klientide poolset testimist ja alles siis saab ettevõte rakenduse kasutusele võtta ja seega senist tööprotsessi parandada.

Kasutatud kirjandus

[1] "Roomakardinad - Aknakate OÜ - Rulood, Kassettrulood, Ribakardinad, Lamellkardinad, Tekstiilkardinad, Turvakardinad, Markiisid, jt. Aknakatted," [Online]. Available: https://www.aknakate.ee/wp-content/uploads/2017/11/4.jpg. [Accessed 16 July 2020].

[2] A. I. (ITI0206). [Online]. Available: https://maurus.ttu.ee/aine_index.php?aine=378. [Accessed 10 June 2020].

[3] "NOOM Majandustarkvara," [Online]. Available: https://noom.ee/. [Accessed 23 May 2020].

[4] "EZIIL," [Online]. Available: https://eziil.com/. [Accessed 23 May 2020].

[5] "ERPLY kassa- ja laotarkvara," [Online]. Available: https://ee.erply.com/. [Accessed 23 May 2020].

[6] "Java (programming language) - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Java_(programming_language). [Accessed 9 June 2020].

[7] "Spring Framework - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Spring_Framework. [Accessed 9 June 2020].

[8] "Spring Boot," [Online]. Available: https://spring.io/projects/spring-boot#overview. [Accessed 9 June 2020].

[9] "The Gradle build system- Tutorial," [Online]. Available: https://www.vogella.com/tutorials/Gradle/article.html. [Accessed 9 June 2020].

[10] "Gradle | Gradle vs Maven: Performance Comparison," [Online]. Available: https://gradle.org/gradle-vs-maven-performance/. [Accessed 28 July 2020].

[11] "Liquibase," [Online]. Available: https://www.liquibase.org/get-started/how-liquibase-works. [Accessed 3 August 2020].

[12] "SOLID - Wikipedia," [Online]. Available: SOLID. [Accessed 7 July 2020].

[13] "A Solid Guide to SOLID Principles | Baeldung," [Online]. Available: https://www.baeldung.com/solid-principles. [Accessed 9 July 2020].

[14] "Open–closed principle - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Open%E2%80%93closed_principle. [Accessed 9 July 2020].

[15] "SOLID — Java materjalid dokumentatsioon," [Online]. Available: https://ained.ttu.ee/javadoc/oop/oop-solid.html. [Accessed 9 July 2020].

[16] "Interface segregation principle," [Online]. Available: https://en.wikipedia.org/wiki/Interface_segregation_principle. [Accessed 21 July 2020].

[17] "Representational state transfer - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed 8 July 2020].

[18] "What Is REST?," [Online]. Available: https://www.restapitutorial.com/lessons/whatisrest.html. [Accessed 22 July 2020].

[19] "Follow Spring MVC architecture to turn static HTML into a Thymeleaf template - Create Web Applications Efficiently With the Spring Boot MVC Framework - OpenClassrooms," [Online]. Available: https://user.oc-

static.com/upload/2019/04/08/1554742446913_Screen%20Shot%202019-04-08%20at%206.51.40%20pm.png. [Accessed 16 July 2020].

[20] "Kanban," [Online]. Available: https://en.wikipedia.org/wiki/Kanban_(development). [Accessed 22 July 2020].

[21] "VeeValidate," [Online]. Available: https://logaretm.github.io/vee-validate/. [Accessed 14 July 2020].

[22] "SheetJS - Home," [Online]. Available: https://sheetjs.com/. [Accessed 14 July 2020].

[23] "What is REST – Learn to create timeless REST APIs," [Online]. Available: https://restfulapi.net/. [Accessed 8 July 2020].