TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduste instituut

Vladislav Redkin 120755IAPB

KLIENDIPORTAALI VEEBIRAKENDUSE ARENDAMINE

Bakalaureusetöö

Juhendaja: Gert Kanter
Doktorant

Kaasjuhendaja: Liis Harjo
Magister

Tallinn 2019

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: Vladislav Redkin

18.05.2019

Annotatsioon

Selle bakalaureusetöö eesmärgiks on analüüsi käigus välja selgitada tellija vajadused, ning selle põhjal luua kliendiportaali rakendus mis aitab optimeerida Estiko Plastar AS kliendihaldurite tööd ning suhtlust ettevõtte ja tema klientide vahel.

Töö realiseerimise käigus luuakse veebirakendus, mis võimaldab klientidel vaadata ja kinnitada Estiko poolt loodud toote disaine. Koostada toodete tellimust ja kinnitada Estiko poolt tellimusele tehtud pakkumist. Jälgida tellimuse täitmise progressi, ning vaadata ajakohast laoseisu infot. Süsteem on liidestatud Microsoft Axapta ärijuhtimise tarkvaraga, mille kaudu päritakse kasutajate, disainide, toodete ja tellimuste infot, ning millega suheldakse sündmuste näol.

Töö kirjeldab rakenduse analüüsi, funktsionaalsust, arhitektuuri, suhtlust väliste süsteemidega ja rakenduse testimist. Samuti on välja toodud probleemid, millega on silmitsi seistud rakenduse arendamisel, ning mis on mõjutanud rakenduse arendamise käiku ja mida ei suudetud analüüsimise ajal ette näha.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 6 peatükki, 11 joonist, 1 tabel ja 5 koodinäidet

Abstract

The aim of this thesis is to identify the needs of the client during the analysis, and on this basis create a client portal application that helps to optimize the work of Estiko Plastar AS client managers and communication between the company and its clients.

During the implementation process, a web application will be created that will allow customers to view and approve the design of products created by Estiko. Create an order and confirm the offer made by Estiko to the order. Keep track of your order progress and view inventory state. The system is integrated with Microsoft Axapta's business management software, through which users, designs, products, and order information are synchonized and communicated through events.

The thesis describes application analysis, system functionality, architecture, communication with external systems and application testing. Problems that have been encountered in the development of the application have also been identified, which have affected the development of the application and could not be foreseen during the analysis.

The thesis is in Estonian and contains 35 pages of text, 6 chapters, 11 figures, 1 table and 5 code examples.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
REST	*Representational State Transfer* on programmiarhitektuuri laad kliendi ja serveri vaheliseks suhtluseks
JPA	*Java Persistence API* on Java rakendusliides mis kirjeldab
DAO	*Data Access Object* - on objekt mis võimaldab suhelda andmebaasi või mõne muu info talletamise viisiga
JSON	*JavaScript Object Notation* on lihtsustatud andmevahetusvorming
JWT	*JavaScript Web Token* - standard kompaktseks ja turvaliseks info edastamiseks kahe osapoole vahel kasutades JSON struktuuri [29]
CSV	*Comma Separated Values,* faili formaat, kus väärtused on eraldatud komaga [35]
Microsoft Axapta	Microsoft Axapta on Microsofti poolt loodud kõrgetasemeline ärijuhtimise tarkvara [27]

Sisukord

Jooniste loetelu

Koodinäidete loetelu

Tabelite loetelu

1 Sissejuhatus

Estiko Plastar AS tegeleb klientidele pakendilahenduste disainimise ja toomisega, ning on selles valdkonnas üks juhtivatest ettevõtetest Baltikumis. Ettevõte pakub nii tootearendust pakendi disainimise ja tootmise näol, kui ka terviklahenduse osutamist, alates trükiettevalmistusest lõpetades logistiliste ja laoteenuste pakkumisega [1]. Kuigi tegu on Eesti mastaabis suure ettevõttega, kellel on tuhandeid kliente nii eestis kui välismaal, käib kogu ettevõtte ja kliendi vaheline suhtlus e-maili teel. Selline suhtlusviis oli küllalt tavaline veel viisteist aastat tagasi, kuid on nüüdseks kiire infotehnoloogia arenguda ajale jalgu jäänud, kuna on aeganõudev ja tülikas mõlemale osapoolele. Põhjuseks miks tänapäeval suheldakse nii suures ettevõttes klientidega enamasti e-mail teel on eelkõige olnud sobiva lahenduse puudumine, kuna kogu ettevõtte sisene haldus on üles ehitatud Microsoft Axapta baasil ja tegu ei ole iseseisva süteemiga. Sellest tingituna tekkis vajadus kohandatud lahenduse järele.

Antud töö eesmärgiks saab olema luua Estiko Plastar AS jaoks veebrirakendus, mis teeb kliendi ja ettevõtte vahelise suhtluse oluliselt kiiremaks ja mugavamaks, ning hakkab asendama siiani kasutatavat suhtlust e-maili teel. Rakenduse põhiliseks funktsionaalsuseks on toote disainide kinnitamise protsessi automatiseerimine ja tellimuste esitamise lihtsustamine. Klient peab saama näha kõiki oma disaine ja tellimusi ning nende infot ja üksikasju talle sobival ajal, ja seda kõike sõlumata kliendihalduri tööajast.

1.1 Metoodika

Püstitatud eesmärkide saavutamiseks selgitatakse välja rakenduse funktsionaalsus ning nõuded ja tehakse analüüs. Rakendus realiseeritakse kolmes interatsioonis. Iga iteratsioon hõlmab endas mitut loogiliselt eraldatud osa. Iteratsiooni lõpus esitletakse realiseeritud funktsinaalsust tellijale ning kuulatakse ära võimalikud ettepanekud. Kolmanda iteratsiooni lõpus tarnitakse kliendile valmis rakendus, mille järel algab kogu rakendust hõlmav põhjalik funktsionaalsuse testimine. Kliendiportaali kasutamisel ilmnenud probleemid, parandused ja täiendused lisatakse github'i probleemide haldamise keskkonda (*issue tracker*). Ilmnenud vigade parandandamisega tegeletakse esimesel võimalusel, sõltuvalt probleemi kriitilisusest. Uue funktsionaalsuse soovidele tehakse eelnevalt analüüs ning antakse ajahinnang, mille järel tellija otsustab, kas konkreetne töö võetakse arendusse või mitte.

2 Analüüs

Selles peatükis on kirjeldatud veebirakenduse analüüsi käik, rakenduse erinevate osade funktsionaalsust, funktsionaalsed ja mittefunktsionaalsed nõuded. Tuuakse välja eeldatav tööde teostamise viis ja järjekord ning olemi-suhte diagramm.

2.1 Rakenduse funktsionaalsed nõuded

Rakenduse funktsionaalsed nõuded:

1. Kasutaja saab rakendusse sisse logida.
2. Kasutaja saab rakendusest välja logida.
3. Portaali päises on menüü, mille kaudu toimub lehekülgede vaheline navigeerimine.
4. Kliendihaldur saab näha kasutajate listi ning temaga seotud ettevõtteid.
5. Kasutaja näeb toote disainide listi.
6. Kliendihaldur saab mitteaktiivsele kasutajale liitumiskutse saata.
7. Kliendihaldur saab muuta kasutaja mitteaktiivseks.
8. Kliendihaldur näeb e-kirjade malle.
9. Kliendihaldur saab e-kirjade malle muuta.
10. Kasutaja saab toote disainile kommentaari lisada.
11. Kliendi esindaja saab toote disaini kinnitada.
12. Kliendi esindaja saab toote disaini tagasi lükata.
13. Kliendihaldur saab disaini arhiveerida.
14. Kasutaja näeb ostutellimuste listi.
15. Kliendi esindaja saab koostada ostutellimust.
16. Kliendi esindaja saab esitamata ostutellimust kustutada.
17. Kliendi esindaja saab müügipakkumist kinnitada.
18. Kliendi esindaja saab müügipakkumist tagasi lükata.
19. Kasutaja näeb laoseisu.
20. Kasutaja saab laoseisu eksportida nii CSV kui XLSX formaadis.
21. Kliendi esindaja saab laost tooteid välja kutsuda.
22. Kasutaja näeb kaebuste listi.
23. Kliendi esindaja saab lisada uut kaebust.

24. Kliendi esindaja saab esitamata kaebust kustutada.
25. Kliendi esindaja saab kaebusele kommentaare ja faile lisada.
26. Kliendihaldur saab kaebuse rahuldada.
27. Kliendihaldur saab kaebuse tagasi lükata.

2.2 Rakenduse mittefunktsionaalsed nõuded

Rakenduse mittefunktsionaalsed nõuded:

1. Rakendus on nii eesti- kui ingliskeelne, sõltuvalt kasutaja küljes olevast parameetrist.
2. Kasutajaliides peab olema lihtne kasutamises, arusaadav ja kiire.
3. Rakendus peab töötama Google Chrome, Mozilla Firefox ning Microsoft Edge veebilehitsejate uusimate versioonidega.
4. Rakendus ei ole mõeldud mobiilsetes seadmetes kasutamiseks, seega ei ole mobiilseadmete tugi vajalik.
5. Süsteem peab olema turvaline.
6. Rakendus peab töötama linux serveris.

2.3 Rakenduse funktsionaalsuse kirjeldus

Rakenduse funktsionaalsus on jagatud rollipõhiselt neljaks, moodustades kaks gruppi: kliendi esindaja ja Estiko poolne kasutaja.

2.3.1 Kliendi esindaja funktsionaalsus

Kliendi esindaja näeb tema poolt esindatavate ettevõtete disaine, tellimusi, laoseisu ja kaebusi.

Toote disainide lehel näeb disainide nimekirja, saab disaini kinnitada või tagasi lükata, lisada kommentaare.

Tellimuste lehel saab luua uut tellimust, pakkumist kinnitada või tagasi lükata, vaadata tellimuste nimekirja, nende staatust ja detaile.

Laoseisu lehel on võimalik näha ja eksportida tellimuste pealt genereeritud laoseisu ja tarneinfot.

Kaebuste lehel lisada uut kaebust, näha selle staatust ja töödeldud kaebuse otsust.

2.3.2 Estiko kasutaja funktsionaalsus

Estiko kasutajad on sõltuvalt rollist jagatud kolmeks:

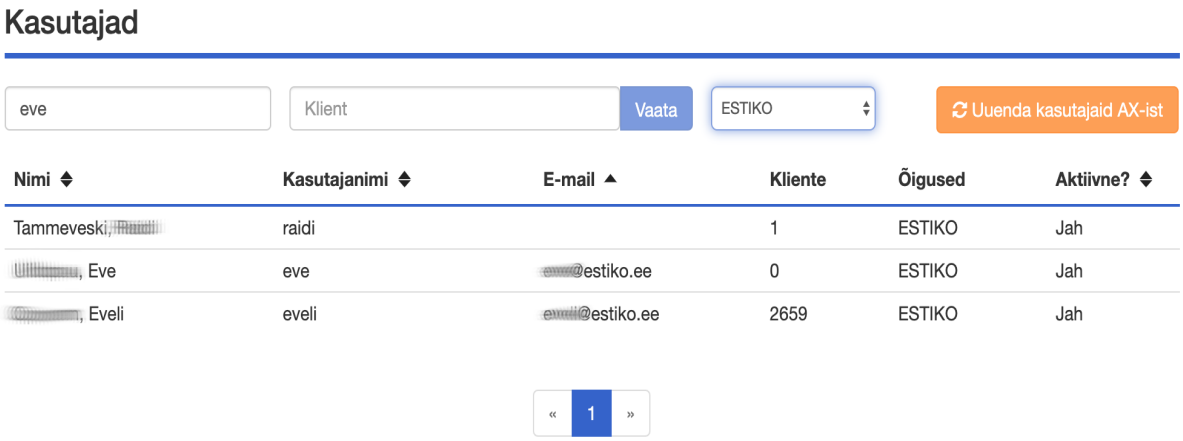
Kliendihaldur näeb endaga seotud klientide disaine, tellimusi, laoseisu ja kaebusi. On ligipääs kasutajate ja ettevõtete nimekirjale. Võimalus lisada disainide alla kommentaare. Saab kliendi kasutajat sulgeda ja suletud kasutajale liitumise kutset saata.

Admin kasutajal on kõik samad õigused mis kliendihalduril, kuid näeb kõikide klientide infot, mitte ainult endaga seotud kasutajate omi. Lisaks on admin kasutajal ligipääs e-kirja mallidele ja võimalus neid muuta. Samuti saab õige staatuse korral kustutada tellimusi ja kaebuseid, ning toote disaine arhiveerida.

Disaineri rolliga kasutajal on ligipääs ainult disainide lehele, kus näeb endaga seotud klientide disaine, ning saab disaini alla kommentaare jätta.

2.3.3 Kasutajate haldus

Kasutajate halduse lehel näeb kõiki süsteemi kasutajaid (Joonis 1). Kasutajad on kuvatud nimekirjana, 25 kasutajat ühel lehel. Kasutajaid on võimalik otsida kasutajanime, emaili, nime ja kasutaja tüübi järgi. Lisaks on olemas klientide nimekiri autocomplete'na, valides valikust kliendi, kuvatakse kõik kasutajad, kellel on olemas seos valitud kliendiga. Antud lehel on ka nupp klientide ja kasutajate käsitsi uuendamiseks Microsoft Axaptast, millest on pikemalt kirjutatud peatükis 3.7.

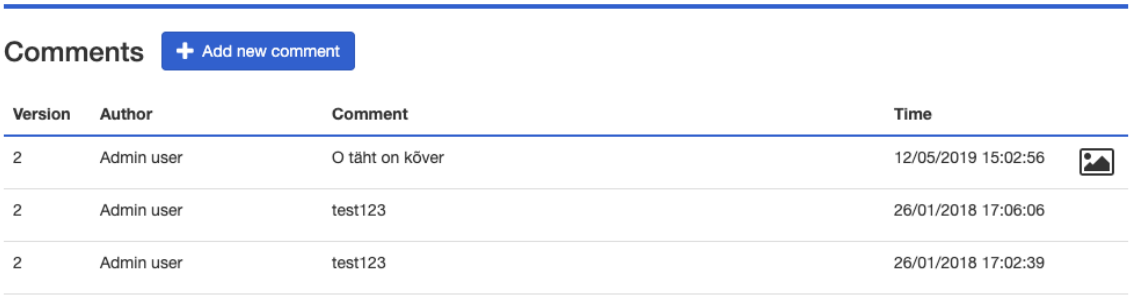


Joonis 1. Kasutajate nimekirja vaade

2.3.4 Disainide leht

Estiko poolt luuaks kliendi tootele kujundus ning saadetakse see portaali kaudu kliendile kinnitamiseks. Kliendi esindaja saab sellekohase e-kirja. Vastava disaini alt näeb kasutaja disaini pilti, mis on kuvatud *canvas*'es (Joonis 2). Pilti saab sisse ja välja suumida, nihutada, valida pildi pealt ala ning selle pealt kommentaari luua (Joonis 4). Samuti on võimalik täpsustavaid kommentaare jätta ainult tekstilisel kujul, pildi peal ala valimata. Kommentaare kuvatakse disaini detailvaates olevas kommentaaride plokis (Joonis 3). Kui kujundus sobib, siis klient kinnitab disaini. Juhul kui kujundus ei sobi, lükkab kasutaja koos põhjusega toote disaini tagasi. Tagasilükatud disain läheb tagasi disaineri töölauale, ning kogu protsess algab otsast peale.

Joonis 2. Toote disaini vaade

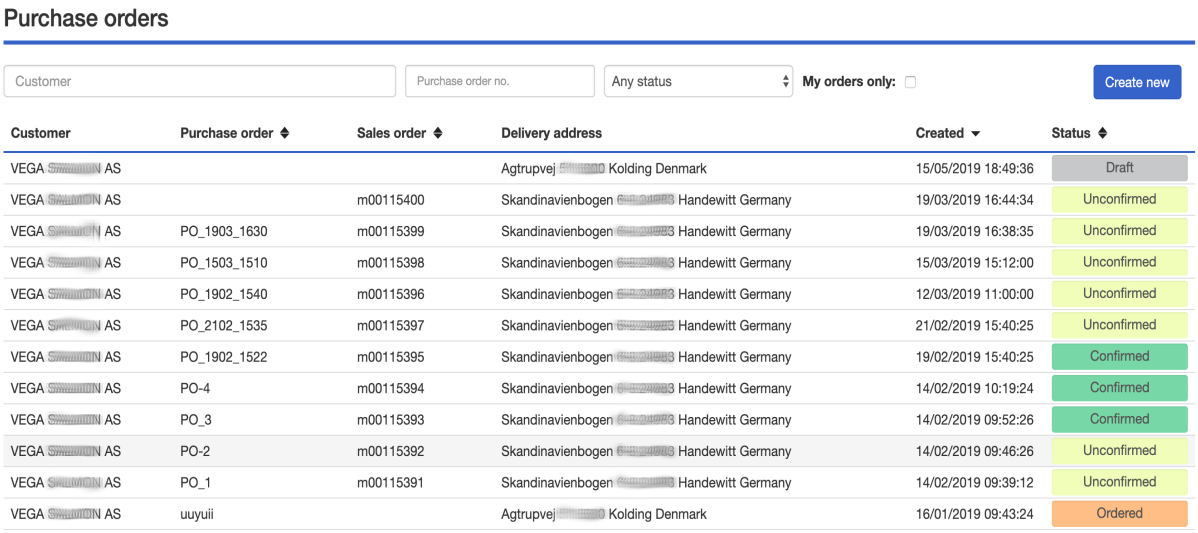


Joonis 3. Toote disaini kommentaaride plokk

Joonis 4. Toote disainile pildiga kommentaari lisamine

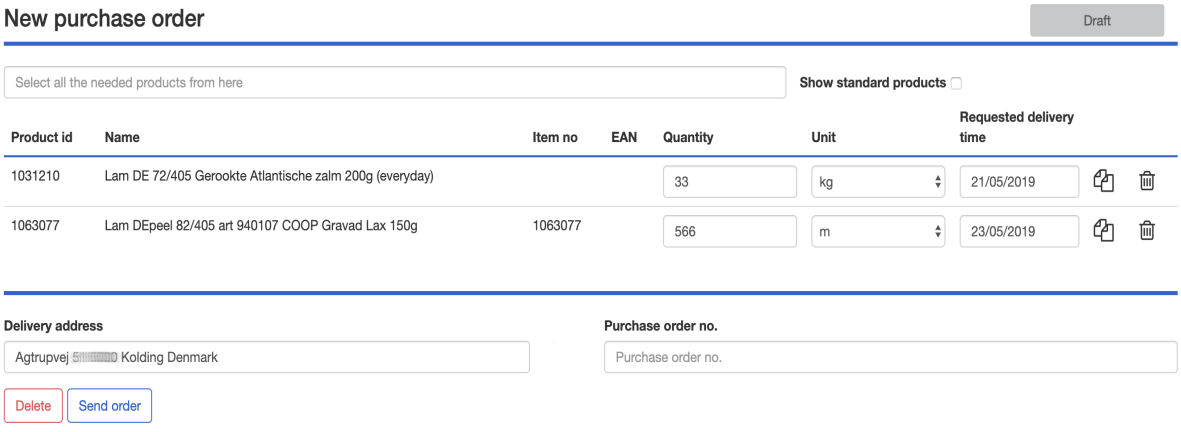
2.3.5 Tellimuste leht

Tellimuste lehel näeb kasutaja tellimuste nimekirja, ning kliendi esindaja saab luua ka uut tellimust. Tellimusi saab otsida kliendi, tellimuse numbri ja staatuse järgi. Näide tellimuste lehest on toodud joonisel 5.



Joonis 5. Tellimuste nimekirja vaade

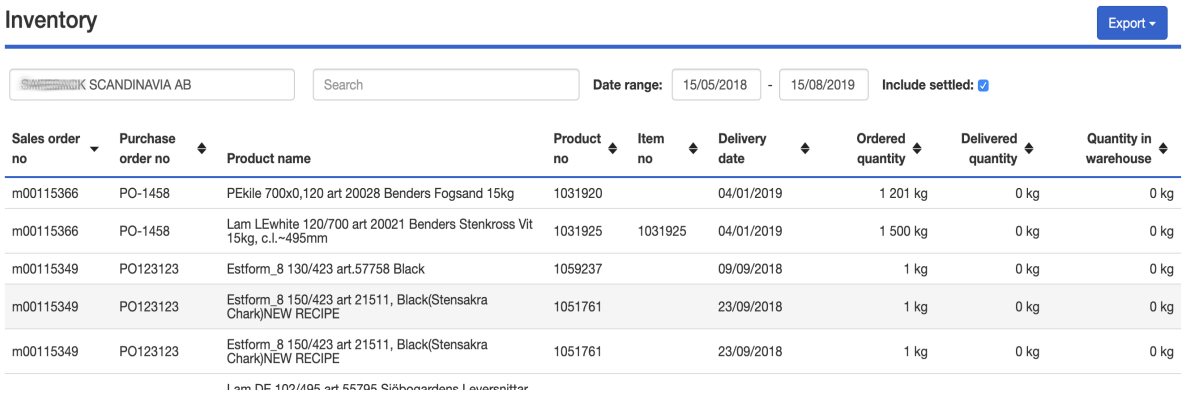
Uut tellimuse luues saab soovitud tooted valida autocomplete väljalt, mille järel lisatakse need tabeli lõppu. Seejärel sisestatakse kogus, ühik ja soovitud tarnekuupäev Samat toodet võib tellimusel olla ka mitu. Näiteks juhul kui soovitakse ühte sama sama toodet saada erinevateks kuupäevadeks. Toote real on kopeerimise nupp, mis teeb reast koopia ja lisab selle tabeli lõppu. Esitatud tellimus saadetakse Estikosse, misjärel koostatakse sellele pakkumine ja saadetakse kliendile tagasi kinnitamiseks. Kinnitatud tellimus läheb automaatselt tootmisse. Tellimuse loomise vaade on kuvatud joonisel 6.



Joonis 6. Uue tellimuse vaade

2.3.6 Laoseis

Kasutaja saab pärida Estiko laos olevate toodete laoseisu, mis genereeritakse kliendi tehtud tellimuste ja käesoleva hetke laoseisu pealt. Laoseisu vastus on grupeeritud tellimuse järgi, mille iga rida kajastab konkreetset toote rida tellimuse peal. Ühel tellimusel saab olla mitu sama tootega rida, millel on erinevad tarne kuupäevad (*Delivery date)* ja/või erinevad kogused, sellel põhjusel kuvatakse tabel tellimuse toote alusel, ning igal real on lisaks tellimuse ja toote infole välja toodud tarne kuupäev (*Delivery date*), tellitud kogus (*Ordered quantity*), tarnitud kogus (*Delivered quantity*) ja laos olev kogus (*Quantity in warehouse*). Kuvatavat tabelit on võimalik eksportida CVS ja XLSX formaati. Joonisel 7 on kuvatõmmis laoseisu leheküljest.



Joonis 7. Laoseisu vaade

2.3.7 Kaebuste leht

Kaebuste lehel saab esitada kaebust, juhul tellitud toode ei vasta nõuetele. Kaebusele lisatakse tellimuse number, artikli kood, ühik ja nõuetele mittevastava toote kogus, kaebuse sisu ja nõue. On ka võimalus lisada faile koos kommentaariga.

Esitatud kaebus saadetakse kliendihalduri töölauale. Haldur vaatab kaebuse üle, vajadusel võtab kliendiga ühendust ja sõltuvalt probleemi põhjusest kas rahuldab kaebuse või lükkab selle tagasi.

2.3.8 E-kirja mallide haldus

Kliendiportaal saadab üsna palju e-kirju, alustades kutsest portaali kasutamiseks ja informeerivad kirjad tellimuse staatuse muutusest ning lõpetades kinnitamise meeldetuletuskirjadega. Kõik e-kirjad on nii eesti- kui ingliskeelsed ja on realiseeritud

mallidena. Mall on HTML kood, koos kohatäitjatega, mis täidetakse kliendipoolse infoga. Malle juurde lisada ei saa, kuid olemasolevaid malle on võimalik redigeerida.

Mallide halduse lehel näeb suvaliste andmetega eeltäidetud malli ning võimalike argumentide listi, mida saab malli sees kasutada. E-kirja malli saab alla laadida, teha vajalikud muudatused, ning uuesti üles laadida. Üles laetud mall valideeritakse. Kui mall valideerub, kuvatakse see eelvaates, mille järel on kasutajal võimalik muudatus salvestada. Malle saavad näha ja muuda ainult admin rolliga kasutajad.

2.4 Töö planeerimine

Analüüsist lähtuvalt sai töö jaotatud kolmeks sprindiks. Iga sprindi pikkuseks oli planeeritud umbkaudu 200 tundi, mille sees arendati mitu loogiliselt eraldiseisvat osa. Sprindi lõpus esitleti tehtud tööd tellijale ja küsiti tagasisidet. Sprindid jagunesid järgnevalt:

Sprint 1:

- Projekti tehniline põhi ja rakenduse püstipanek
- Liidestus Microsoft Axaptaga
- Kasutajaliidese põhi
- Kasutajate haldus
- E-kirjade mallid ja nende haldus
- E-kirjade saatmine
- Autentimine
- Kasutajate info uuendamine Microsoft Axaptast

Sprint 2:

- Kahekeelne tugi
- Toote disainide nimekiri ja detailvaade
- Ostutellimuste nimekiri, koostamine ja detailvaade
- Ostutellimuste saatmine Microsoft Axaptasse
- Müügipakkumiste pärimine Microsoft Axaptast
- Toodete sünkroniseerimine Microsoft Axaptast
- Aadressite sünkroniseerimine Microsoft Axaptast

Sprint 3:

- Laoseisu pärimine Microsoft Axaptast

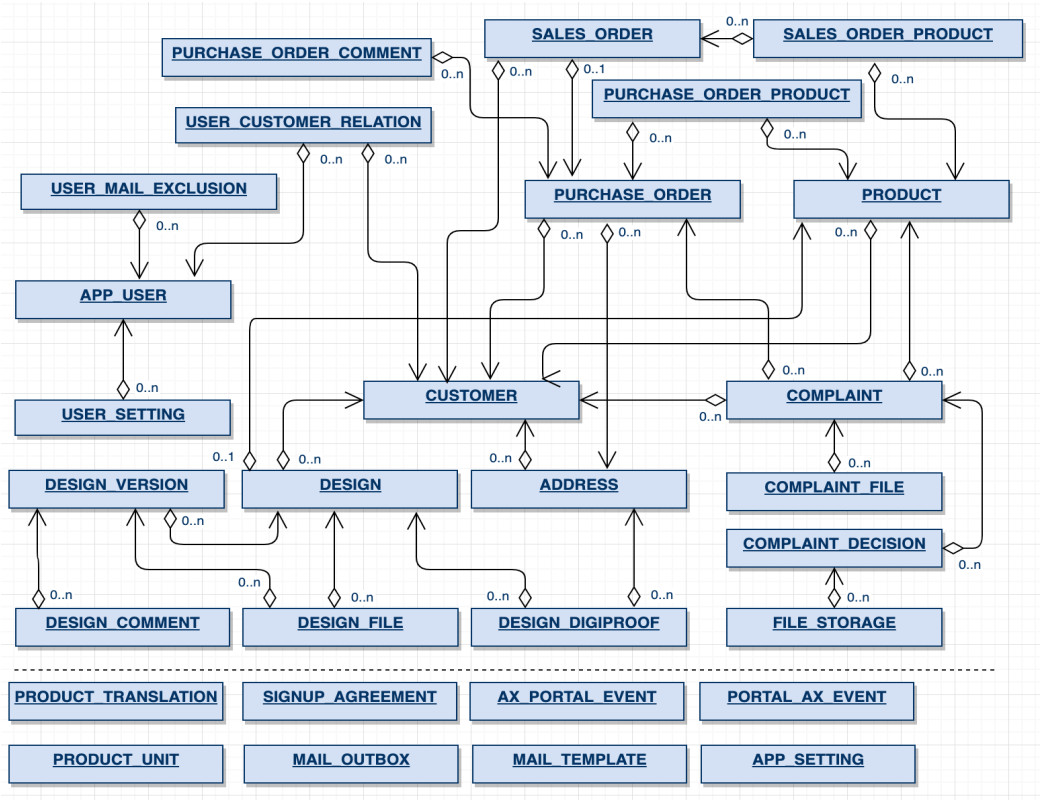
- Laoseisu vaade
- Laoseisu eksport CSV ja XLSX formaati.
- Kaebuste lisamine, detailvaade ja nimekiri
- Toote disaini pildi kuvamine ja joonistamise tarkvara
- Keskkondade seadistamine

Kogu arenduse maht on 700 tundi, ning tegu on *fixed-price* projektiga. See tähendab et projekti arendamise eest makstakse 700 tunni ulatuses. Juhul kui arenduse jaoks kulub aega rohkem kui ette nähtud, siis on see arendaja vastutus ja selle ei ole tasustatud.

2.5 Süsteemi objektid

Süsteemi tsentraalseteks objektideks on klient (customer), kasutaja (app_user) ja toode (product). Joonisel 8 on kujutletud andmebaasi graafilist diagrammi. Kuna tabeleid on palju, siis lihtsuse mõttes on diagrammil kuvatud ainult tabelite nimed ja viited teistele tabelitele.

Peaaegu kõigil tabelitel on created_by ja modified_by väljad, millel küll puudub otsene viide app_user tabelile, kuid kasutab app_user tabeli id väärtuseid. Sellest on põhjalikumalt kirjutatud punktis 3.10.



Joonis 8. Olemi-suhte diagramm

3 Arendus

3.1 Kasutatavad tehnoloogiad ja vahendid

Gradle - Gradle on avatud lähtekoodiga tööriist tarkvara kokkuehitamise automatiseerimiseks ning selle sõltuvuste haldamiseks. Antud projekti kui terviku ja kõigi selle alammoodulite kokkuehitamiseks, automaattestide jookutamiseks ja rakenduse püstipanemiseks kasutatakse gradle'i käske. Gradle on Ant ja Maven kõrval uusim tööriist, mis ühendab endas Ant'i ja Maven'i parimad omadused. [2,3] Näiteks kasutades Gralde'i java pluginat saab java rakenduse kokkuehitamiseks kasutada gradle build ja automaattestide käivitamiseks gradle test käske.

3.1.1 Back-end

Spring Framework on avatud lähtekoodiga Java platformile orienteeritud rakendusraamistik, mis sisaldab endas kogu vajalikku põhifunktsionaalsust Java rakenduse loomiseks. Näiteks REST liides, suhtlus andmebaasiga ja automaattestimine [4].

Spring Boot on sisuliselt Spring Framework laiendus, mis pakub kiiret võimalust eraldiseisva Java rakenduse loomiseks. See peidab endas kogu keerukust ning võimaldab minimaalse koodi kirjutamise ja seadistamisega luua koheselt käivitamiskõlbulikku Java rakendust. [5,6]

Spring Data JPA on üks osa suuremast Spring Data perekonnast. Spring Data toetab suuremat osa SQL ja NoSQL andmebaase ning peidab endas keerukat suhtlust andmebaasiga. JPA (Java Persistance API) on Java ORM-i (Object Relational Mapping võimaldab) standard, mis võimaldab kasutades ainult Java koodi lisada, uuendada ja selekteerida kirjeid andmebaasist [10].

Ldap (Lightweight Directory Access Protocol) on hajutatud kataloogiteenuste sirvimise ja haldamise protokoll mõeldud kasutamiseks üle võrgu. Põhiliseks kasutusvaldkonnaks on kasutajate info, rollide ja õiguste haldamine [11]. Kataloogid võivad sisaldada mistahes infot. Käesoleva projekti raames on LDAP-i kasutatud töötajate autentimiseks.

Lombok ehk Project Lombok on Java teek, mis aitab vähendada geneerilist koodi, nagu näiteks getterid, setterid ja konstruktorid, kasutades selleks lihtsaid annotatsioone.

Enamkasutatavate koodiredaktorite (IDE) jaoks on saadaval Lomboki plugin, mis võimaldab reaalajas, koodi kompileerimata, näha ja kasutada Lomboki poolt genereeritud koodi [28].

Jackson on Java standard teek JSON-i parsimiseks. Jacksoni erinevad moodulid suudavad töödelda lisaks JSON-ile ka Java properties, YAML, XML, CSV ja teisi faile [12]. Antud teeki kasutatakse näiteks Java objekti JSON-iks konverteerimisel ja vastupidi.

JWT (JSON Web Token) on JSON objekt, mida kasutatakse kahe osapoole vahel turvaliseks info jagamiseks. Eelkõige kasutatakse seda kasutaja autentimiseks. JWT token koosneb kolmest osast, päisest (header), kehast (payload) ja allkirjast (signature). Päis sisaldab infot kasutatava algoritmi ja token'i tüübi kohta. Kehas kirjeldatakse parameetritena konkreetset üksust (näiteks kasutaja) ja selle infot. Allkiri koosneb kolmest osast (päis, keha,), mille kõik osad on Base64 kodeeritud ja eraldatud üksteisest punktiga, ning omakorda kindla algoritmiga krüpteeritud. Allkiri võimaldab meil verifitseerida, et token pole osapoolte vahel liigeldes muutunud [29].

3.1.2 Front-end

Node ehk Node.js on 2009 aastal loodud tarkvaraplatvorm, mis võimaldab programmeerida ja jooksutada serveripoolseid JavaScript rakendusi. Node kasutab Google'i V8 JavaScripti mootorit [14].

Npm (Node Package Manager) on 2011 aastal Node.js jaoks loodud tööriist moodulite ja sõltuvuste haldamiseks. Npm kasutab package.json nimelist faili, mille sisu on kirjutatud JSON struktuuris, ning sisaldab minimaalselt mooduli nimetust ning selle versiooni [30]. Sõltuvused laetakse alla vastavast repositooriumist.

TypeScript on Microsofti poolt loodud programeerimiskeel, mis lähtub JavaScripti puudustest suurte rakenduste arendamisel. Typescript võimaldab kasutada tüübikirjeldust, vastab kõikidele ECMAScript 2015 standarditele, ning kompileerub JavaScriptiks [19].

Webpack on front-end moodulite komplekteerija. Webpack käib üle projekti ja koostab kasutatavate sõltuvuste graafi, mida rakendus normaalseks funktsioneerimiseks vajab. Seejärel luuakse uus pakett, mis sisaldab moodulitest ainult minimaalset vajalikku koodi, kõik ülejäänud ja ebavajalik kood jäetakse kõrvale. Selle põhjal luuakse üks kuni mitu kompileeritud bundle'it, mida saab lihtsasti HTML faili importida ja rakenduses kasutada. Lisaks võimaldab Webpack määrata mis vahendiga ja kuidas projektis kasutatavaid faile

töödeldakse [31].

Angular on platvorm mis lihtsustab veebirakenduste loomist. Angular on aastal 2016 oma eelkäija AngularJS täielikult ümber kirjutatud versioon, mis kasutab täiesti uut arhitektuurilist pritsiipi, ning mida soovitatakse kasutada koos TypeScript'iga. Käesolev projekt on kirjutatud kasutades Angulari versioon 6 [17, 20].

SCSS on kujundamiseks kasutatav märgistuskeel. SCSS on CSS'i edasiarendus, võimaldades importida teisi kujundus faile, kasutada muutujaid, üksteise sisse pandud reegleid ja koodi taaskasutamist. See kõik kiirendab oluliselt kujundamist ja koodi kirjutamist, seda eriti mahuka projekti puhul. Ükskõik milline valiidne CSS süntaks on ka samal ajal valiidne SCSS süntaks, kuid mitte vastupidi. Rakenduse kokku ehitamise käigus kompileeritakse SCSS tavaliseks CSS süntaksiks [21].

Bootstrap on avatud lähtekoodiga populaarseim veebilehtede arendamiseks ja prototüüpimiseks mõeldud front-end komponentide teek. Bootstrap sisaldab enimkasutatavaid komponente, nupud, vormid, navigatsioon jne. Teek sai alguse Twitterist, kus see oli mõeldud ettevõttesiseseks kasutamiseks, kuid üsna pea muutus see niivõrd populaarseks et otsustati teha avalikult kättesaadavaks [23].

3.2 Arhitektuur

Back-end rakendus on realiseeritud kolmekihilisena:

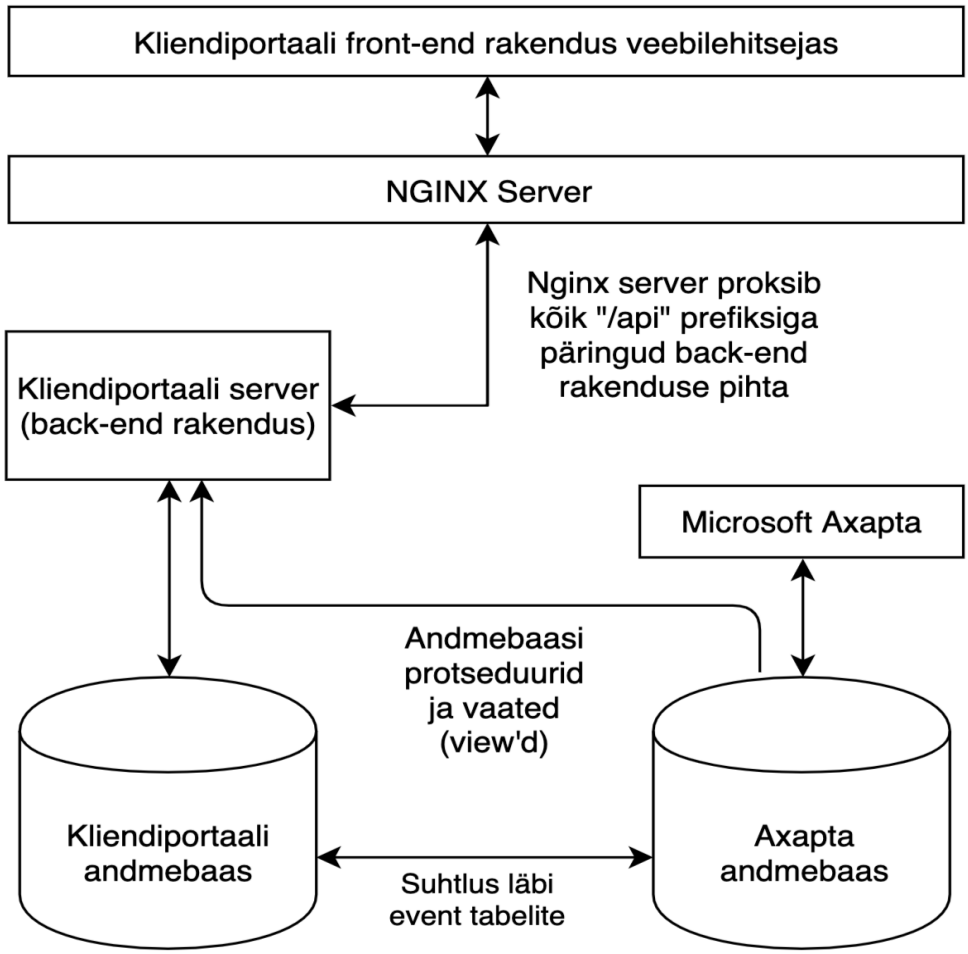
- Andmekiht ehk DAO *Layer*. Andmekiht tegeleb andmete talletamise ja pärimisega. DAO kiht suhtleb andmebaasi või mõne muu andmete talletamise mehhanismiga [10, 13, 26]. Käesolevas rakenduses on suhtleb DAO kiht relatsioonilise andmebaasiga, Microsoft Axaptaga ja virtuaalse kettaga, kus asuvad toote disinide pildid.
- Äriloogika kiht ehk *Service Layer*, mis asetseb ressursi ja andmekihi vahel. Service kiht sisaldab äriloogikat, mis kujundab ümber ja tõlgib andmeid ressursi ja DAO kihtide vahel [8].
- Ressursikiht ehk *Resource layer* on REST teenuste kiht, mis on kättesaadav välistele tarbijatele. Ressursi kiht varjab tarbija eest äriloogikat, ning kasutab sisemiselt Service kihti.

Kogu klient-server suhtlus on *stateless*, see tähendab et server ei talleta mingit infot eelnevate

HTTP päringute kohta, mida kasutaja on teinud. Igat päringut käsitletakse kui uut, ei ole seessioni ega ajalugu. Kogu sessiooni ja oleku info talletatakse kliendi poolel [25].

Joonisel 9 on kujutletud süsteemi diagrammi ning komponentide omavahelist suhtlust. Kliendiportaali back-end rakendus talletab andmeid oma enda andmebaasis. Lisaks oma andmebaasile pärib andmeid Microsoft Axapta andmebaasist läbi vaadete (*view*) ja protseduuride. Ning suhtleb Microsoft Axaptaga kasutades selleks sündmuste tabeleid.

Serveris on paigaldatud NGINX server, mis suunab vastavalt URL-ile kas test või live rakenduse kataloogi, kus asetsevad front-end rakenduse failid. Samuti püüab nginx kinni kõik „/api“ prefiksiga päringud ja suunab need back-end serverisse, mis tegeleb päringute töötlemisega.



Joonis 9. Rakenduse diagramm

3.3 Back-end

Algselt oli rakendus kirjutatud kasutades Spring Boot versiooni 1.4 ja Java 8-t, kuid lõputöö kirjutamise käigus sai rakendus üle viidud uuemale Spring Boot versioonile 2.1 ja Java 11-le. Klassid on paigutatud pakettidesse kasutades *package-by-feature* lähenemist. Iga feature pakett sisaldab omakorda *api*, *interactor* ja *respository* pakette. Nii on kindlat funktsionaalsust arendades lahti võimalikult väike failide puu.

3.4 Front-end

Front-end on rakenduse esitluskiht, mis peidab kasutaja eest äriloogika kihi. Front-end kiht suhtleb veebiteenustega (back-end kihiga) REST päringute vahendusel kasutades selleks JSON struktuuri. Analüüsi käigus sai otsustatud teha front-end single-page veebirakendusena. Single-page veebirakendus on rakendus mis töötab veebilehitsejas ning mis ei vaja kasutamise ajal kogu lehekülge uuesti laadimist [16]. Kuna käesoleva rakenduse funktsionaalsus on suures osas info tabelitena välja kuvamine, mille sisu on suhteliselt harva muutuv, siis single-page rakendus on suurepärane valik, mis võimaldab külastatud lehekülgede infot ja tabelite sisu veebilehitseja poolt mälus. Tabeli laetud lehekülje info hoitakse mälus, ning järgmisel sama lehekülje avamisel võetakse info serveri poole pöördumise asemel mälust. Selline lahendus aitab olulisel määral vähendada rakenduse serveri poole pöördumisi ning tõsta läbi selle kasutusmugavust.

3.5 Andmebaas

Algselt oli plaanis kasutada kahte andmebaasi. Esimene andmebaas rakenduse enda andmete hoidmiseks, selleks oli valitud PostgreSQL. Teist andmebaasi kasutaks portaal view'de kaudu Axaptast andmete sünkroniseerimiseks. PostgreSQL on kõige arenenum avaliku lähtekoodiga relatsiooniline andmebaas. Selle kasuks räägivad rohke funktsionaalsus, kiirus, lihtsus, ulatuslik tugi erinevate teekide poolt, saadavus vabavarana ja samuti tugi erinevate platvormide poolt (Windows, Linux, Mac). Microsoft Axapta kasutab Microsoft SQL Server 2005 andmebaasi, mis on nüüdseks juba 14 aastat vana ning äärmiselt algelise ja piiratud funktsionaalsusega. Vaatamata kõigile Microsoft SQL Server 2005 puudustele PostgreSQL ees, sai andmete sünkroniseerimise probleemide tõttu loobutud kahe andmebaasi kasutamisest ja otsustatud ühe andmebaasi, Microsoft SQL Server 2005 kasuks. Probleemist täpsemalt on kirjutatud punktis 4.1 (Andmete sünkroniseerimine Microsoft Axaptast).

3.6 Andmebaasi skriptid

Andmebaasi skeemi muudatusteks (tabelite loomine, muutmine, andmete sisestamine jne) kasutatakse Liquibase teeki, mis on integreeritud Spring Boot rakendusega. Kõik muudatused on kirjeldatud sql failides mis asetsevad „back/src/main/resources/db/changelog“ kataloogi alamkataloogides. Iga sql fail moodustab *changeset*-i. Changeset on muudatuste grupp, mis võib koosneda ka mitmest muudatusest. Liquibase on konfigureeritud application.properties failis, kus määratakse *changelog* faili asukoht ja nimetus. Antud juhul on selleks „db.changelog-master.xml“ fail, kus hoitakse viiteid muudatuste skriptidele ja määratakse nende paigaldamise järjekord. Liquibase hoiab paigaldatud *changeset*'ide infot DATABASECHANGELOG tabelis. Iga *changeset*'i pealt genereeritakse MD5 räsi, mida hoitakse samuti selles tabelis. Andmebaasi muudatused viiakse sisse Spring Boot rakenduse käivitamise hetkel. Samuti kontrollitakse juba paigaldatud muudatuste vastavust, selleks genereeritakse *changeset*'i pealt uuesti MD5 räsi ja võrreldakse seda andmebaasis olevaga. Juhul kui need ei klapi, siis viitab see muutnud *changeset*'ile. Olenevalt liquibase konfiguratsioonist sõltub, kas *changeset* pannakse uuesti peale või lõpetatakse migratsioon veateatega [32].

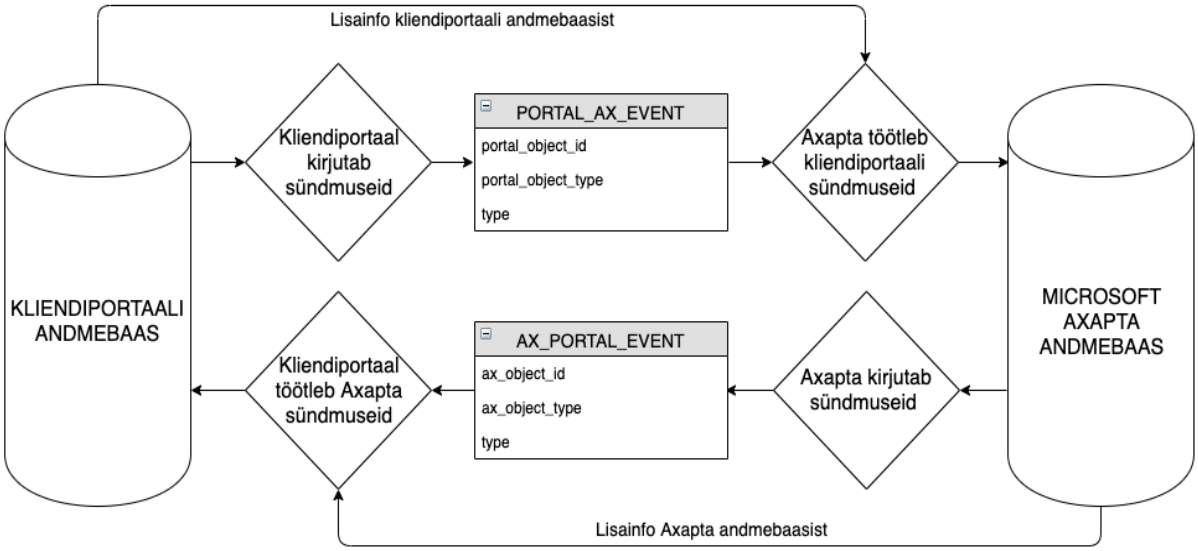
3.7 Liidestus teiste süsteemidega

Lisaks punktis 3.6 kirjeldatud andmete sünroniseerimisele, suhtlevad kliendiportaal ja Microsoft Axapta omavahel sündmuste näol. Analüüsi käigus planeeriti kasutada suhtlemiseks REST teenust ja JSON struktuuri. Kuna aga Microsoft Axapta poolt on JSON struktuuri töötlemise toetus olematu, siis sai kokku lepitud kahe süsteemi vaheline suhtlus realiseerida andmebaasi tasandil. Suhtluse toimimise viis on visualiseeritud joonisel 10.

Kliendiportaali pool toimunud sündmused, mis peavad jõudma Axaptasse, kirjutatakse PORTAL_AX_EVENT tabelisse. Sellesse tabelisse kirjutatud sündmuseid töötleb Microsoft Axapta. Kui töötlemine õnnestub, märgitakse sündmus töödelduks (result=1) ja töötlemise aeg (processed_at) saab väärtuseks hetke aja. Kui töötlemine ebaõnnestub, saab result väli väärtuses “-1” ja result_message väljale kirjutatakse ebaõnnestumise põhjus. Sündmuse tüübi ja selle töötlemise viisi määrab “type” väljal olev eeldefineeritud väärtus, “portal_object_type” tähistab sündmuse tabeli nime ja “portal_object_type” on selles tabelis oleva kirje identifikaator.

Microsoft Axapta kirjutab sündmused, mis peavad portaali jõudma AX_PORTAL_EVENT

tabelisse. Portaalis on sündmuse töötlemiseks eraldi *service* nimega AxPortalScheduler, milles on meetod, mis kasutatud @Scheduled annotatsiooni. See võimaldab meil protsessi teatud perioodilisusega välja kutsuda. Antud juhul iga kolme minuti tagant laetakse tabelist kuni 50 vanimat töötlemata sündmust ja üritatakse neid töödelda. Loogika on täpselt sama mis PORTAL_AX_EVENT tabeli sündmuste töötlemisel, kuid portal_object_type ja portal_object_id veergude asemel on vastavalt ax_object_type ja ax_object_id veerud, mis viitavad Microsoft Axapta vaadetele. Näide südmuste töötlemise protsessist on toodud lisas nr 2.



Joonis 10. Liidestus Microsoft Axaptaga

3.8 Turvalisus ja autentimine

Kõik REST teenused peale sisse ja välja logimise on turvatud, ning neile pääseb ligi vaid sisse loginud kasutaja. Teenuste kasutamise õigust kontrollitakse kolmel tasandil. Kasutatav tase valitakse iga teenuse jaoks eraldi vastavalt vajadusele. Kõik vastu back-end rakendust tehtud päringud püütakse kinni JwtFilter nimelise filtri abil, mis korjab päringu päisest *authorization* väärtuse, milleks on JWT *token*. Seejärel dekodeeritakse ja valideeritakse *token*. *Token*'i seest saadud kasutaja identifikaatori põhjal päritakse kasutaja info andmebaasist ja sisestatakse rakenduse konteksti. Kontekst on päringu põhine, mis tähendab et info säilitatakse ainult alates päringu jõudmisest back-endi kuni back-endi poolt info välja saatmiseni. Juhul kui kasutajal ei ole teenuse kasutamiseks õiguseid, tagastatakse HTTP staatus 401 (Unauthorized).

Esimesel tasandil kontrollitakse kas kasutaja on ennast autentinud.

Teisel tasandi kontrollitakse kasutaja tüüpi, kas kasutaja on klient (CUSTOMER tüüpi), Estiko töötaja (ESTIKO tüüpi) või administraator (ADMIN). Koodinäites nr 1 on selleks "`hasAnyAuthority('ESTIKO', 'CUSTOMER')`", mis valideerib et kasutaja oleks kas ESTIKO või CUSTOMER tüüpi.

Kolmandal tasandil kontrollitakse kas kasutajal on ligipääs kindlale objektile. Kontroll käib objekti tüübi ja selle identifikaatori järgi. Koodinäites nr 1 on selleks "`hasPermission(#purchaseOrderId, 'PURCHASE_ORDER')`" kontroll, mis valideerib kas antud kasutajal on õigus pääseda ligi purchase_order objektile mille identifikaator on päringus parameetrina kaasa antud.

```
@GetMapping(value = "/{purchaseOrderId}/get")
@PreAuthorize("hasAnyAuthority('ESTIKO', 'CUSTOMER') &&
hasPermission(#purchaseOrderId, 'PURCHASE_ORDER')")
public ApiResponseWrapper<PurchaseOrderResource, Void>
getPurchaseOrder(@PathVariable Long purchaseOrderId) {
 return wrap(purchaseOrderInteractor.getPurchaseOrder(purchaseOrderId));
}
```

Kood 1. Kontrolleri teenuse koodinäide

Süsteemi sisenemiseks saadetakse autentimise päring, mille sisuks on kasutajanimi ja parool. Õnnestunud autentimise päringu vastuseks on lisaks kasutaja infole ka JWT token, mis salvestakse *localStorage* objekti, ning mis pannakse järgnevate REST päringute päisesse kaasa. JWT loomisel määratakse selle eluajaks kaheksa tundi. Kaheksa tunni möödudes tuleb kasutajal ennast uuesti autentida. Näide autentimise päringu vastusest on toodud koodinäites 2.

```
{
  "data": {
    "jwtToken":
"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJuakB2ZWdhc2FsbW9uLmRrIiwidXNlcklkIjoyMjI4LCJ0eXBlI
joiQ1VTVE9NRVIiLCJhZG1pbiI6ZmFsc2UsImlhdCI6MTU1ODE4OTgyMiwiZXhwIjoxNTU4MjI1ODIyfQ.8
HlTdjsvZSO-d99o_nVTUVRjei4yW9pyA1melZM1lWWTA6PMX_44dTXZu-
ZoxHDCCdCo6NsO_EsenvZC8juKTg",
    "userId": 2228,
    "username": "nj@.********.dk",
    "name":"Nick Jensen",
    "email": "nj@********.dk",
    "type":"CUSTOMER",
    "isAdmin": false,
    "isDesigner": false,
    "lang": "EN",
    "numberOfCustomers": 1
  },
```

```
  "metadata":null
}
```

Kood 2. Näide autentimise päringu vastusest

Süsteemis on kahte liiki kasutajaid, kliendid ja Estiko töötajad. Kliendid kasutavad sisse logimiseks emaili. Kliendi kasutajate kasutajanimesid ja paroole hoitakse app_user tabelis. Paroolid on krüpteeritud kujul. Krüpteerimiseks kasutatakse BCrypt krüpteerimise funktsiooni, mis on juba Spring Security teeki sisse ehitatud.

Estiko töötajad autentitakse kasutades AD (Active Directory) kontot. See annab võimaluse siseneda süsteemi kasutades sama kasutajatunnust ja parooli mida kasutab töötaja Estiko sisevõrgus arvutisse sisse logimiseks. Juhul kui kasutaja AD konto märgitakse suletuks, ei saa kasutaja ka kliendiportaali siseneda. AD kontoga autentimiseks saadab kliendiportaal AD serveri pihta päringu, kasutades selleks LDAP protokolli. Päringuga antakse kaasa kasutajanimi, mis Estiko AD kasutaja puhul on alati @estiko.sise lõpuga, ning parool. Õnnestunud päringu korral tagastab LDAP server kasutaja grupid, ebaõnnestunud autentimise korral tagastab server *Invalid Credentials* vea [33].

3.9 REST liides

Kõigi REST päringute vastused on nö dekoraatoriga (*wrapper*) dekoreeritud. See tähendab et baasstruktuur on vastustel sama, sisaldades kahte objekti: „data“ ja „metadata“. Data objekti sisu ongi sisuliselt teenuse vastus: objekti pärimisel objekt, listi pärimisel list. Metadata objekti kasutatakse vastusele lisainfo kaasa andmiseks. Näiteks listi pärimisel märgitakse metadatasse listi pikkus. Kui tegu on elementide kogust alamhulga pärimisega, siis sisaldab metadata objekt kogu hulga elementide arvu, alamhulga pikkust, ning kui mitu elementi on enne ja pärast alamhulka.

Selles rakenduses on kasutusel kaks liiki päringuid, GET ja POST päringud. GET päringuid kasutatakse elemendi või listi info pärimiseks. POST päringuid mingi objekti salvestamiseks või muutmiseks. Kuid on ka erandeid, mõned oma loomult GET päringud, millel on palju parameetreid, kasutavad POST meetodit. Nii on võimalik meetoditesse kaasa anda pika parameetrite listi asemel ainult üks objekt, see mõnevõrra lihtsustab koodi lugemist. Tabelis 1 on toodud näide tellimuse (*purchase order*) teenustest.

Kõik REST kontrollerid laiendavad BaseController klassi. BaseController klassis toimub erinevate rakenduse vigade kinni püüdimine ja ühisele kujule teisendamine. Nii välditakse vea

tekkimisel rakenduse *Stack trace*'i välja saatmist. Selle asemel saadetakse alati viga generaliseeritud kujul, mis sisaldab vea tüüpi (nt VALIDATION_ERROR, GENERAL_ERROR, INTERNAL_ERROR jne), võimalusel selle tõlkekoodi, mida saab front-end rakenduses tõlkida, ning ingliskeelset vea lühikirjeldust. Tekinud vea tüübist sõltub ka tagastatav staatuse kood. Sellisel kujul tagastavad vead on front-end rakenduses lihtsamini kinni püütavad ja võimaluse korral tõlgitavad. Õnnestunud päring tagastab staatuseks 200.

Teenus	**Meetod**	**Toming**
api/order/list	POST	Tellimuste listi pärimine
api/order/{purchaseOrderId}	GET	Tellimuse info pärimine id järgi
api/order/{customerId}/create	POST	Uue tellimuse loomine
api/order/{purchaseOrderId}/update	POST	Tellimuse info uuendamine
api/order/{purchaseOrderId}/products	GET	Tellimuse toodete pärimine
api/order/{purchaseOrderId}/products/sales	GET	Pakkumise toodete pärimine
api/order/{purchaseOrderId}/products/current	GET	Toodete tootmisinfo pärimine
api/order/{purchaseOrderId}/product/add	POST	Tellimusele toote rea lisamine
api/order/{purchaseOrderId}/product/{rowId}/update	POST	Tellimuse toote rea uuendamine
api/order/{purchaseOrderId}/product/{rowId}/delete	POST	Tellimuse toote rea kustutamine
api/order/{purchaseOrderId}/decline	POST	Pakkumine tagasilükkamine
api/order/{purchaseOrderId}/confirm	POST	Pakkumise kinnitamine
api/order/{purchaseOrderId}/send-order	POST	Tellimuse saatmine
api/order/{purchaseOrderId}/delete	POST	Tellimuse mustandi kustutamine
api/order/{purchaseOrderId}/delete-by-manager	POST	Tellimuse kututamine Estiko töötaja poolt
api/order/terms/{lang}	GET	Kliendilepingu info pärimine

Tabel 1. Ülevaade mõningatest REST teenustest

3.10 Andmebaasi kirjete auditeerimine

Kõik andmebaasi tabelid sisaldavad *created_at, created_by, modified_at, modified_by* välju. Neid välju kasutatakse kirjete auditeerimiseks, ning on vaja kahel põhjusel. Juhul kui mõnes vaates on vaja objekti loomise või muutmise aega välja kuvada. Või kui on tarvis tõendada kirje muutmise, siis on lihtsasti tuvastatav, kelle poolt ja millal kirjet muudeti.

Kirje loomisel saavad *created_at* ja *modified_at* loomise ajatempli, *created_by* ja

modified_by väärtuseks on kasutaja identifikaator, kes salvestamise välja kutsus. Kirje muutmisel uuendatakse *modified_at* j a *modified_by* praeguse ajatempli ja muutja kasutaja identifikaatoriga vastavalt. Muutmisel jäävad *created_at* ja *created_by* väärtused muutmata.

Kõik tabelite klassid mis on annoteeritud @Entity annotatsiooniga ja mis vastavad üks-ühele tabeli struktuurile, laiendavad abstraktset EntityAudit klassi. See tagab et DAO kihis insert või update meetodit välja kutsudes, väärtustatakse audit väljad automaatselt. @CreatedBy ja @ModifiedBy annotatsiooni poolt tagastatav väärtus on konfigureeritud SpringSecurityAuditorAware klassis. Järgnevalt on koodinäites 3 toodud näide EntityAudit klassist.

```
@Data
@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
public abstract class EntityAudit {

 @CreatedDate
 private Instant createdAt;
 @CreatedBy
 private Long createdBy;
 @LastModifiedDate
 private Instant modifiedAt;
 @LastModifiedBy
 private Long modifiedBy;
}
```

Kood 3. Koodinäide EntityAudit klassist

3.11 E-kirjade saatmine

E-kirjade genereerimine toimub kasutaja kindlate tegevuste peale (nt pakkumise või disaini kinnitamine). Samuti genereerivad e-kirju automaatsed protsessid, mis kindla perioodi tagant kontrollivad andmebaasist kirjete seisu, ning juhul kui need vastavad teatud kriteeriumitele, siis genereeritakse vastava malli põhjal e-kiri ning salvestatakse andmebaasi. E-kirjade saatmise automaatprotsess (MailScheduler) kontrollib iga kolmekümne sekundi tagant e-kirjade andmebaasi tabelit, otsib seal saatmata kirjad ning üritab neid saata. Kui kirja saatmine õnnestus, siis märgistatakse see andmebaasi kirje vastava linnukese ning saatmise kuupäeva ja kellaajaga. Juhul kui saatmine ebaõnnestub ükskõik mis põhjusel, suurendatakse selle kirje ebaõnnestunud saatmiste loendurit. Ebaõnnestunud kirja saatmiseks tehakse kuni viis katset, kusjuures iga järgneva katse aeg on lineaarses sõltuvuses ebaõnnestunud saatmiste arvust. Viimane (viies) katse tehakse üks tund ja viisteist minutit peale esimest ebaõnnestunud katset.

Viie ebaõnnestunud katse järel uusi saatmise katseid ei tehta ja kiri jääb saatmata.

3.12 Tehnilised lahendused

3.12.1 Transaktsioonid

Transaktsioon on muudatuste propageerimine andmebaasi. Transaktsiooni sees andmebaasi mootorile saadetud korraldused lahendatakse nö spekulatiivselt, ehk need muudatused lahendatakse, kuid ei rakendata kohe. Transaktsiooni muudatuste propageerimiseks tuleb muudatused kinnitada (*COMMIT*). Kuid kõik muudatused võib ka tagasi võtta kasutades *ROLLBACK* käsku. Rakenduses kasutatakse transaktsioone ainult andmebaasi kirjete muutmisel (lisamine, uuendamine, kustutamine), andmete pärimine on lahendatud transaktioonivälise lt. Spring raamistik võimaldab transaktsioone juhtida kasutades @Transactional annotatsiooni. Enamasti on seda annotatsiooni kasutatud kas *service* või *repository* kihis, olenevalt vajadusest. Kuna suur osa REST teenustest, mis tegelevad andmete muutmisega, teevad muudatusi korraga mitmes tabelis, siis tuleb tagada andmete õigus ka olukorras kui tekib viga. @Transactional annotatsiooniga märgistatud meetodi sees tehtud muudatused propageeritakse automaatselt kui meetod on oma töö lõpetanud, või keeratakse kõik muudatused tagasi juhul kui on tekkinud viga [22].

3.12.2 Identifikaatorite varjamine

Front-end rakenduses on kasutaja eest andmebaasi identifikaatorid varjatud kasutades selleks "hashids" teeki, mis genereerib identifikaatori numbrist kuueteistkümne märgi pikkuse räsi kasutades selleks krüptograafilist "soola" [34]. Kõik aadressireal kasutatavad identifikaatorid on asendatud räsiga. Iseenesest ei ole identifikaatorite kuvamises turvariski, sest kõik REST teenused on turvatud mitmetasandiliselt, võimaldades ligipääsu andmetele ainult neil kasutajatel, kellel on selleks olemas vastavad õigused. Pigem hoitakse sellega ära ekperimente, kus kasutaja üritab aadressireal olevat numbrit muutes pääseda ligi teiste klientide andmetele. Räsi dekodeeritakse samuti front-endi poolel. Juhul kui dekodeerimine ebaõnnestub, siis back-end rakenduse suunas päringut ei saadeta. Nii hoitakse ära liigsed päringud back-end rakenduse pihta. Tõenäosus et räsi suvalist tähte või numbrit muutes saab üleüldse kätte mingisuguse numbri on väga väike.

Näide kasutaja profiilile viivast URList (.../users/D4Qbre8K01y2R6pO), kus räsi "D4Qbre8K01y2R6pO" on number 36 kodeerutud kujul.

3.12.3 Back-endi ja front-endi ühilduvus

Tingituna asjaolust et back-end ja front-end on sisuliselt erinevad rakendused mida on võimalik uuendada üksteisest sõlttumatult, siis tuli tagada mõlema rakenduse vahel sajaprotsendiline ühilduvus. Selleks oli kirjutatud "api-gen" teek, mis skaneerib Java Reflection API't kasutades üle kõik projekti klassid mis laiendavad BaseController Java klassi, otsib nende sealt üles kõik @GetMapping ja @PostMapping annotatsioonidega teenused ja genereerib nende põhjal api-service.ts faili, kus kirjeldatakse kõik REST teenused, mida kasutab front-end rakendus teenuste poole pöördumiseks [18]. Teenuste sisend ja väljund klassid käiakse läbi rekursiivselt ning nende põhjal genereeritakse api-types.ts fail.

Peale back-end rakenduse koodis muudatuste tegemist käivitatakse "gradlew generateApi" käsk, mis uuendab api-service.ts ja api-types.ts failid vastavalt uuele teenustele ja nende struktuurile. Juhul kui tehtud muudatus ei ühti front-end rakenduse koodiga, annab front-endi rakenduse kompilaator vea. Nii on võimalik tagada, et mõlemad rakendused oleksid omavahel ühilduvad. Näide genereeritud api-service.ts failist koodinäites nr 1.

```
export class AddressOperations {
  constructor(private api: ApiClient) {}

  getCustomerAddresses(customerId: number):
Observable<GetCustomerAddresses$Response> {
    return this.api.execute('GET', 'api/address/' + customerId + '/list',
null);
  }

  saveAddress(customerId: number, saveAddressResource:
SaveAddressResource): Observable<SaveAddress$Response> {
    return this.api.execute('POST', 'api/address/' + customerId + '/save',
saveAddressResource);
  }

  useAddress(customerId: number, addressId: number):
Observable<UseAddress$Response> {
    return this.api.execute('POST', 'api/address/' + customerId + '/use/' +
addressId + '', null);
  }
```

Kood 4. Koodinäide genereeritud typescript teenuste klassist

3.13 Rakenduse ehitamine ja paigaldamine

3.13.1 Back-end'i ehitus

Back-end'i ehitamiseks kasutatakse gradlew build käsku, mis kompileerib java koodi, kopeerib resources kaustas olevad failid ning pakib kogu rakenduse .jar laiendiga faili. Kui back-end on kompileeritud, käivitatakse gradlew generateApi käsk, mis omakorda kompileerib ja käivitab api-gen teegi. Api-gen teegist on põhjalikumalt kirjutatud punktis 3.12.3.

3.13.2 Front-end rakenduse ehitus

Veebirakenduse test ja live keskkonnad asuvad samas serveris kuid erinevatel alamdomeenidel. Rakenduse kokku ehitamiseks kasutatakse serveris olevat shell scripti, mis laeb github'i repositooriumist vastava haru viimase seisu, ehitab back-end ja front-end rakendused, kopeerib genereeritud failid õigesse kausta ning käivitab backend rakenduse. Kõik uued arendused ja parandused mergetakse repositooriumi master harru, selle pealt ehitatakse ka test keskkond. Kui test rakendus on nii arendaja kui tellia poolt üle kontrollitud, mergetakse master haru release harru. Keskkondade ehitamisel ja käivitamisel kasutatakse iga keskkonna jaoks kindlat profiili (dev, test, live). Profiili põhjal laetakse vastav konfiguratsioonifail. Tundlikke andmeid, nagu näiteks andmebaasi kasutaja nimi ja parool, ei hoita rakenduse konfiguratsiooni failis vaid serveri pool ja antakse rakenduse käivitamisel parameetrina kaasa. Nii on võimalik vältida tundlike andmete lekkimist.

Front-end rakendus kompileeritakse *bundle*'teks kasutades selleks AOT (*Ahead-of-Time*) kompileerimist. AOT kompileerimisel on mitmeid eelised võrreldes JIT (Just-in-Time) kompileerimisega [15]:

- AOT kompileerimise käigus tuvastatakse HTML malli ja komponendi vaheliste seoste ebakõlad. See võimaldab tuvastada potentsiaalseid veakohti enne toodangusse jõudmist.
- Kompileeritud failide suurus. JIT kompileerimist kasutades kompileeritakse HTML mallid veebilehitsejas vahetult enne välja kuvamist, mistõttu peab angulari kompilaator failidesse kaasatud olema, mis aga moodustab kogu angulari teegi suurusest umbkaudu poole.
- Kiirus -- AOT'ga kompileeritud koodi ei pea veebilehitseja enam kompileerima,

mistõttu on leht peale rakenduse *bundle* failide allalaadimist kohe väljakuvamiseks valmis.

3.13.3 Rakenduse keskkonnad ja serverisse paigaldus

Arendamiseks kasutatakse dev (*development*) keskkonda, mis jookseb arendaja arvutis ning kasutab lokaalset andmebaasi. Back-end rakenduse käivitamiseks kasutatakse gradlew bootRun käsku, front-end rakenduse käivitamiseks gradlew runDevServer käsku.

3.13.4 Front-end rakenduse uuendamine

Kuna front-end rakendus on realiseeritud *single-page* veebirakendusena, siis laetakse rakenduse kompileeritud kood korraga brauserisse ja rakenduses ringi navigeerides, laetakse ainult selle sisu. Kahtlemata on see kasutusmugavuse poolest väga mugav, kuna lehekülgede vahetamine toimub silmapilkselt, kuid sellega kaasneb ka üks tõsine probleem. Nimelt kui back-endi rakenduse REST teenuse struktuur või selle URL peaksid muutuma niivõrd, et see ei ühti front-end rakenduses kirjeldatuga, siis üsna suure tõenäosusega saab front-end rakendus päringut tehes vea. Selle probleemi lahendamiseks genereerib rakenduste kokkuehitamise shell script unikaalse numbri, mis antakse ehitamise käigus kaasa nii front-end kui back-end rakendustele. Iga REST päringu päises on “app-version” nimeline väli, mille väärtuseks on ehitamise käigus genereeritud number. Kõik front-end’i päringud käivad läbi ApiClient nimelise service’i, mis tegeleb lisaks vigade kontrolli ja päringu vastuse JSON kujule viimisele ka “app-version” parameetri kontrollimisega. Juhul kui back-end’i poolt tagastatav number ei ühti front-end’i poolt talletatuga, kuvab rakendus hoiatava teate, et rakendust on uuendatud ja palub kasutajal lõpetada oma toimingud ning vajutada hoiatusteates olevale lingile, mis laseb serverist uue front-end rakenduse koodi.

3.14 Testimine

Tarkvara testimine on jagatud kolmeks osaks, ühiktestid, alfatestimine ja süsteemitestimine. Iga eelneva testi lõpus liigutakse järgmisse testimise faasi. Kui kõik testimise sammud on läbitud, *merge*’takse testimises olnud haru toodangu (release) haruga, ning edasi toimub live keskkonna uuendamine.

3.14.1 Ühiktestid

Ühiktestid ehk *Unit* testid on kirjutatud väiksemate osade testimiseks, enamasti on selleks

service kihis olev loogika. Selles faasis testitakse meetodite välja kutsumist kindlate parameetritega, ning kontrollitakse kas meetodi sisend ja/või väljund vastab eeldusele. Testide kirjutamiseks ja jooksutamiseks kasutatakse JUnit ja Mockito raamistikke. Mockito'ga saab Java klasse ja meetodeid välja mock'ida, kontrollimaks ainult väljakutsutava meetodi parameetrid ja tagastatavat väärtust meetodi sisest loogikat välja kutsumata. Nii saame näiteks service kihis testida DAO kihi meetodite välja kutsumist ja väärtuste tagastamist ilma andmebaasi poole pöördumiseta.

3.14.2 Alfatestmine

Alfatestimine on arendaja poolt läbi viidav esmane uue funktsionaalsuse füüsiline testimine, mis viiakse läbi arendaja masinas. Kui testimine osutus edukaks, liidetakse (*merge*'takse) antud haru master haruga. Seejärel paigaldatakse arendatud funktsionaalsus test keskkonda, mille ehitus toimub alati master haru pealt. Teine alfatestimine toimub testkeskkonnas samuti arendaja poolt, kuid seekord reaalsete andmetega. Suur osa funktsionaalsust on testitav ainult test keskkonnas, kuna on hulganisti sõltuvusi axapta vaadetest ja keerukatest andmebaasi protseduuridest, mille arenduskeskonnas simuleerimine oleks liialt ajakulukas ning ei kataks kõiki võimalikkue stsenaariume.

3.14.3 Süsteemitestimine

Süsteemitestimise käigus testib tellijapoolne isik tellitud arenduse vastavust nõuetele. Testimise faasi lõpus *merge*'takse arendatav haru githubi release harru, mille pealt käib live keskkonna ehitamine ja paigaldamine.

4 Tehnilised probleemid

Selles punktis on kirjeldatud probleeme mis ilmnesid arendamise käigus.

4.1 Andmete sünkroniseerimine Microsoft Axaptast

Rakenduse realiseerimise käigus selgus, et kahe andmebaasi (PostgreSQL ja Microsoft SQL Server 2005) kasutamine ei ole mõistlik. Algne plaan kasutada info sünkroniseerimiseks *view* poolt tagastatavate kirjete küljes olevat viimase muudatuse kuupäeva ja kellaaega ei osutunud edukaks kolmel põhjusel. Esiteks ei uuendatud neid väärtuseid alati, kuna view väärtus pandi kokku mitme tabeli põhjal ja mingi muudatus Microsoft Axaptas ei pruukinud neid väärtuseid uuendada. Teiseks esines arenduse käigus olukord kui view'sse tuli lisada mõni uus veerg, mis samuti ei kajastunud muutmise kuupäevas ja info jäi sünkroniseerimata. Kolmandaks probleemiks oli andmete uuendamise loogika Java kihis. Tingituna kahe erineva andmebaasi kasutamisest tuli andmed mõlemast andmebaasist pärida Java kihti, konverteerida vastavale kujule ja viia andmed omavahel vastavusse. See aga osutus liialt aja- ja ressursi kulukaks, kuna reaalne kirjete arv ulatus mitmekümne tuhandeni ja sünkroniseerimise ajaks kujunes kümmekond sekundit.

PostgreSQL andmebaasist loobumisega tekkis ka mitmeid probleeme. Kuna arendus toimus macOS operatsioonisüsteemil, siis lokaalset Microsoft SQL Server andmebaasi oli võimalik paigaldada ainult kasutades Docker image'it, mis on saadaval alates 2017 aastast ja kasutatavatest versioonidest on ainult uusimad, 2017 ja 2019. Versioonide erinevusest tingituna (test ja live keskkonnas on kasutusel versioon 2005) esines vahetevahel komplikatsioone, kui lokaalselt töötas rakendus korrektselt, kuid test keskkonda paigaldamisel tekkis viga migratsiooni skriptide käivitamisel või DAO kihis sql päringutes.

SQL Server andmebaasiga ei tööta pagineerimiseks kasutatav Java Persistence API setFirstResult(n) meetod, mis elimineerib sql päringu vastusest *n* esimest kirjet. Seetõttu tuli pagineerimine implementeerida käsitsi, kasutades selleks „`ROW_NUMBER() OVER(ORDER BY ...)`“ funktsiooni.

Samuti puuduvad SQL Server 2005 versioonis järgmised funktsionaalsused:

- „`CREATE OR UPDATE PROCEDURE/FUNCTION ...`“ -- protseduuri või funktsiooni

uuendamiseks puudub *update* lause, selleks tuleb funktsioon või protseduur üles otsida süsteemi tabelist, kustutada ning seejärel uuesti luua.

- Juhul kui andmebaasi tabeli kintsenduse (*constraint*) loomisel jäi selle nimi määramata, genereerib SQL Server nime automaatselt ja selleks on suvaliselt genereeritud väärtus. Kitsendusega tabeli veeru kustutamiseks, tuleb eelnevalt leida kitsendus süsteemi tabelist ning see kustutada. Selleks on üsna keerukas päring, võrreldes näiteks PostgreSql andmebaasiga. Koodinäites 5 on toodud näide design tabelist DEFAULT kitsenduse kustutamiseks.

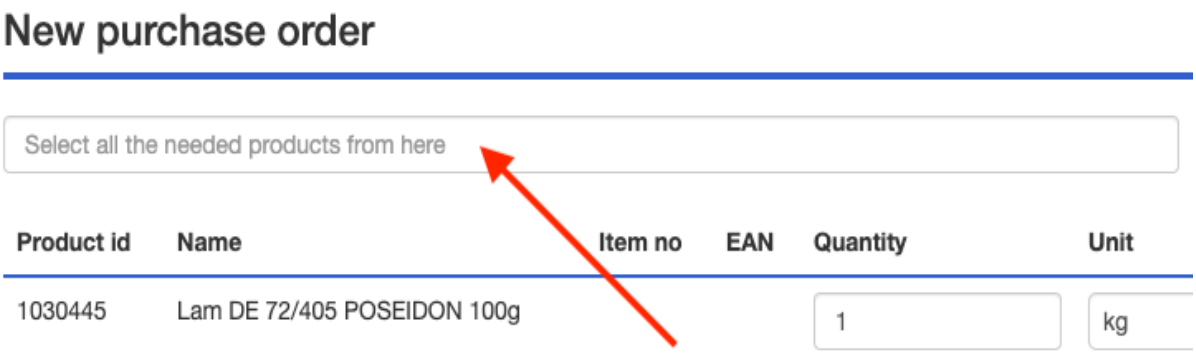
```
DECLARE @ConstraintName nvarchar(200)
BEGIN
  SELECT @ConstraintName = name
  FROM sysobjects WHERE xtype = 'D' and name LIKE 'DF__design__need_%'
  IF @ConstraintName IS NOT NULL
    EXEC('ALTER TABLE design DROP CONSTRAINT ' + @ConstraintName)
END;
```

Kood 5. Koodinäide tabeli veeru kitsenduse kustutamiseks

4.2 Tellimusele toodete lisamine

Tellimuste koostamise lehel on autocomplete toodete valikuga (joonisel 11 märgitud punase joonega), sellele vajutades avaneb nimekiri kliendi toodetest. Toote valimisel lisatakse see tabeli lõppu ja autocomplete elemendi valik tühjendatakse.

Live keskkonnas selgus, et mõned kliendid ei saa aru, et tellimusele on võimalik lisada ka mitu toodet. Tellijaga probleemi arutuades ei leidnud head lahendust. Alternatiivina sai välja pakutud toote lisamine viia eraldi modaali koos põhjalikuma selgitusega, kuid sellest sai loobutud, kuna tellimustel on tavaliselt üsna palju tooteid ja toote modaalist valimine pigem tekitaks klientidele ebamugavust. Jõudsime kokkuleppele et jätame olemasoleva lahenduse ja teavitame kliente eraldi, kellel on sellega probleeme tekkinud. Õnneks ei ole selliseid kliente eriti palju. Tulevikus on plaan lisada leheküljele põhjalikum kasutusjuhend.



Joonis 11. Tellimuse lisamise vaade

4.3 Toote disaini pildina kuvamine

Toote disaini pildi sisse ja välja suumimise, nihutamise ja pildilt ala valimise funktsionaalsus osutus oodatust oluliselt keerukamaks. Keeruliseks tegi asjaolu, et peale suumimist tuli pilti nihutades või pildilt ala valides arvestada muutunud pildi ja *canvas*-e suhtega. Lisas 3 on toodud näide koodist, mis peale suumimist või pildi nihutamist arvutab uuesti pildi mõõtmed, küljesuhted, algus ja lõppkoordinaadid, ning uuendab canvas-e sisu vastavalt transformeeritud pildile.

5 Kokkuvõte

Antud töö põhieesmärk oli analüüsida ja luua veebirakendus, mis võimaldaks Estiko Plastar AS-l pakkuda oma klientidele uut viisi toote disainide ja tellimuste haldamiseks ning tellimuste täitmise progressi jälgimiseks ja reaalajas laoseisu vaatamiseks. Lahendust, mis asendaks seni kasutusel olevat suhtlust kliendi ja ettevõtte vahel e-mailide teel.

Projekti realiseerimise käigus sai kinnitust fakt, et põhjalik analüüs on projekti arendamisel väga oluline, seda eriti *fixed-price* projekti puhul. Põhjalikumat analüüsi tehes oleks võinud mõningaid arendamise käigus tekkinud probleeme ennetada.

Tulemuseks on käesoleva töö kirjutamise hetkeks juba üle aasta aja kasutuses olev veebirakendus, mis on aktiivse kasutusperioodi jooksul saanud juurde ka mitmeid täiendusi ja uut funktsionaalsust.

6 Kasutatud kirjandus

[1] „Estiko Plastar AS üldinfo“, [Võrgumaterjal]. Available: https://plastar.ee/meist/uldinfo [Kasutatud: 10.05.2019]

[2] „Gradle funktsionaalsuse kirjeldus“, [Võrgumaterjal]. Available: https://gradle.org/features/ [Kasutatud: 06.05.2019]

[3] „Ant, Maven ja Gradle võrdlus“, [Võrgumaterjal]. Available: https://www.baeldung.com/ant-maven-gradle [Kasutatud: 06.05.2019]

[4] „Spring Boot kirjeldus“, [Võrgumaterjal]. Available: https://spring.io/projects/spring-boot [Kasutatud: 06.05.2019]

[5] „Spring teegi dokumentatsioon“, [Võrgumaterjal]. Available: https://spring.io/projects/spring-framework [Kasutatud: 10.05.2019]

[6] „Spring ja Spring Boot“, [Võrgumaterjal]. Available: https://www.baeldung.com/spring-vs-spring-boot [Kasutatud: 10.05.2019]

[7] „Sissejuhatus Spring Boot rakendusse“, [Võrgumaterjal]. Available: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm [Kasutatud: 10.05.2019]

[8] „Veebirakendusarhitektuurid“, [Võrgumaterjal]. Available: https://maurus.ttu.ee/ained/IDU0200_2012/doc/36/SissejuhatusRakenduseArhitektuuri_Back_ja_Front.pdf [Kasutatud: 08.05.2019]

[9] „Front End vs Back End Development: Where to start?“, [Võrgumaterjal]. Available: https://www.coursereport.com/blog/front-end-development-vs-back-end-development-where-to-start [Kasutatud: 08.05.2019]

[10] „Java Persistence API tutorial“, [Võrgumaterjal]. Available: https://www.vogella.com/tutorials/JavaPersistenceAPI/article.html [Kasutatud: 03.05.2019]

[11] „LDAP teadmiste omandamise õppematerjal“, [Võrgumaterjal]. Available: https://wiki.itcollege.ee/images/1/1f/LDAP_teadmiste_omandamise_%C3%B5ppematerjal.pdf [Kasutatud: 14.04.2019]

[12] „Jackson Project“, [Võrgumaterjal]. Available: https://github.com/FasterXML/jackson [Kasutatud: 10.04.2019]

[13] „Java Persistence API ja ORM“, [Võrgumaterjal]. Available: https://moodle.taltech.ee/pluginfile.php/128576/mod_resource/content/0/jee10.pdf [Kasutatud: 11.04.2019]

[14] „Node.js tutvustus“, [Võrgumaterjal]. Available: http://minitorn.tlu.ee/~jaagup/kool/java/kursused/14/prog1/Node.docx [Kasutatud: 13.04.2019]

[15] „Angulari AOT kompolaator“, [Võrgumaterjal]. Available: https://angular.io/guide/aot-compiler [Kasutatud: 13.05.2019]

[16] „SPA ja MPA võrdlus“, [Võrgumaterjal]. Available: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58 [Kasutatud: 11.05.2019]

[17] „Best practices for a clean and performant Angular application“, [Võrgumaterjal]. Available: https://medium.freecodecamp.org/best-practices-for-a-clean-and-performant-angular-application-288e7b39eb6f [Kasutatud: 14.05.2019]

[18] [Võrgumaterjal]. Available: https://stackoverflow.com/questions/205573/at-runtime-find-all-classes-in-a-java-application-that-extend-a-base-class [Kasutatud: 15.05.2019]

[19] „Typescript keele ülevaade“, [Võrgumaterjal]. Available: https://www.tutorialspoint.com/typescript/typescript_overview.htm [Kasutatud: 15.05.2019]

[20] „Angulari platvormi dokumentatsioon“, [Võrgumaterjal]. Available: https://angular.io/docs [Kasutatud: 15.04.2019]

[21] „Sissejuhatus SCSS-i“, [Võrgumaterjal]. Available: https://dzone.com/articles/introduction-of-scss [Kasutatud: 16.04.2019]

[22] „Transaktsioonid“, [Võrgumaterjal]. Available: http://enos.itcollege.ee/~priit/E-kursus%20(I%20245)%20AB-de%20alused/11/69670548001_11-5.htm [Kasutatud: 20.04.2019]

[23] „Bootstrap teek“, [Võrgumaterjal]. Available: https://getbootstrap.com/ [Kasutatud: 25.04.2019]

[24] „Pebble Templates dokumentatsioon“, [Võrgumaterjal]. Available: https://pebbletemplates.io/ [Kasutatud: 23.04.2019]

[25] „RESTful arhitektuuri kitsendused“, [Võrgumaterjal]. Available: https://restfulapi.net/rest-architectural-constraints/ [Kasutatud: 11.05.2019]

[26] „Java DAO muster“, [Võrgumaterjal]. Available: https://www.baeldung.com/java-dao-pattern [Kasutatud: 01.05.2019]

[27] „Microsoft Axapta kirjeldus“, [Võrgumaterjal]. Available: https://www.isystems-group.com/solutions/microsoft-dynamics-ax/ [Kasutatud: 15.04.2019]

[28] „Lombok project“, [Võrgumaterjal]. Available: https://projectlombok.org/ [Kasutatud: 17.04.2019]

[29] „JWT dokumentatsioon“, [Võrgumaterjal]. Available: https://jwt.io/introduction/ [Kasutatud: 15.05.2019]

[30] „NPM tutvustus“, [Võrgumaterjal]. Available: https://docs.npmjs.com/about-npm/ [Kasutatud: 13.05.2019]

[31] „Webpack“, [Võrgumaterjal]. Available: https://webpack.js.org/concepts [Kasutatud: 08.05.2019]

[32] „Liquibase dokumentatsioon“, [Võrgumaterjal]. Available: https://www.liquibase.org/documentation/index.html [Kasutatud: 16.05.2019]

[33] „Spring – LDAP autentimine“, [Võrgumaterjal]. Available: https://spring.io/guides/gs/authenticating-ldap/ [Kasutatud: 02.04.2019]

[34] „Hashids teek“, [Võrgumaterjal]. Available: https://hashids.org/javascript/ [Kasutatud: 07.05.2019]

[35] „CSV“, [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Comma-separated_values [Kasutatud: 05.04.2019]

Lisa 1 – Koodinäide AccessPermissionEvaluator test klassist

```
@RunWith(MockitoJUnitRunner.class)
public class AccessPermissionEvaluatorTest {

  @Mock
  private ComplaintPermissionService complaintPermissionService;
  @Mock
  private UserData userData;
  @Mock
  private Authentication authentication;
  @Spy
  @InjectMocks
  private AccessPermissionEvaluator accessPermissionEvaluator;

  @Before
  public void init() {
    when(authentication.getPrincipal()).thenReturn(userData);

    accessPermissionEvaluator.evaluators.put(
      COMPLAINT, new PermissionRule(complaintPermissionService::hasAccess));
  }

  @Test(expected = AccessDeniedException.class)
  public void throwExceptionWhenNotAuthenticated() {
    when(authentication.getPrincipal()).thenReturn(null);

    accessPermissionEvaluator.hasPermission(authentication, 1L, "COMPLAINT");
  }

  @Test(expected = AccessDeniedException.class)
  public void throwExceptionWhenRuleNotFound() {
    accessPermissionEvaluator.hasPermission(authentication, 1L, "RANDOM");
  }

  @Test
  public void throwExceptionWhenHasNoPermission() {
    when(complaintPermissionService
      .hasAccess(1L,authentication))
      .thenReturn(false);

    assertFalse(accessPermissionEvaluator
      .hasPermission(authentication, 1L, "COMPLAINT"));
  }

  @Test
  public void hasPermission() {
    when(complaintPermissionService
      .hasAccess(1L, authentication)).thenReturn(true);

    assertTrue(accessPermissionEvaluator
      .hasPermission(authentication, 1L, "COMPLAINT"));
  }
}
```

Lisa 2 – Microsoft Axapta sündmuste töötlemise koodinäide

```
@Slf4j
@Component
@RequiredArgsConstructor
public class AxPortalScheduler {
 private final AuthService authService;
 private final AxPortalEventRepository axPortalEventRepository;
 private final AxaptaInEventInteractor axaptaInEventInteractor;


 @Scheduled(fixedRate = 180000, initialDelay = 5000)
 public void selectUnprocessedRows() {
   authService.systemUserAuthentication();
   List<AxPortalEvent> list = axPortalEventRepository.selectUnprocessedRows();
   if (!list.isEmpty()) {
     list.forEach(this::processAxEvent);
   }
 }

 void processAxEvent(AxPortalEvent row) {
   if (row == null) {
     return;
   }

   switch (row.getType()) {
     case DESIGN_SEND_TO_CLIENT:
       axaptaInEventInteractor.processDesignNewVersion(row);
       break;
     case DESIGN_DIGIPROOF_SENT:
       axaptaInEventInteractor.processDesignDigiproofSent(row);
       break;
     case QUOTATION_NEW_VERSION:
       axaptaInEventInteractor.processSalesOrderNewVersion(row);
       break;
     case QUOTATION_DELIVERED:
       axaptaInEventInteractor.processSalesOrderDelivered(row);
       break;
     default:
       axaptaInEventInteractor.unknownEvent(row.getId());
   }
 }
}
```

Lisa 3 – Näide toote disaini joonistamiskoponendi koodist

```
draw(center: Point, zoomRatio: number) {
  if (this.context) {
    this.context.clearRect(0, 0, this.canvas.width, this.canvas.height);
    this.context.drawImage(
        this.image, 0, 0, this.image.width, this.image.height,
        this.image.width / 2 - center.x * zoomRatio, this.image.height / 2 -
center.y * zoomRatio,
        this.image.width * zoomRatio, this.image.height * zoomRatio);
  }
}


rescale(previousZoomRatio: number, newZoomRatio: number, center?: Point) {
  if (previousZoomRatio === newZoomRatio || previousZoomRatio === undefined ||
newZoomRatio === undefined || this.canvas == null) {
    return;
  }
  if (center == null) {
    center = {x: this.canvas.scrollWidth / 2, y: this.canvas.scrollHeight / 2};
  }
  const yRatio = center.y / this.canvas.scrollHeight;
  const xRatio = center.x / this.canvas.scrollWidth;
  const prevWidth = this.right - this.left;
  const prevHeight = this.bottom - this.top;
  const scale = previousZoomRatio / newZoomRatio;
  const newWidth = scale * prevWidth;
  const newHeight = scale * prevHeight;
  this.left = this.leftValue + (prevWidth - newWidth) * xRatio;
  this.right -= (prevWidth - newWidth) * (1 - xRatio);
  this.top = this.topValue + (prevHeight - newHeight) * yRatio;
  this.bottom -= (prevHeight - newHeight) * (1 - yRatio);
  this.centerXValue = (this.left + this.right) / 2;
  this.centerYValue = (this.top + this.bottom) / 2;
  this.draw({x: this.centerX, y: this.centerY}, newZoomRatio);
}
```