TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Aivar Mägi 184962IADB

Laoressursside halduseks klient-server arhitektuuriga rakenduse arendamine

Bakalaureusetöö

Juhendaja: Meelis Nõmm
Ph.D.
Juhendaja: Kristiina Hakk
Ph.D.

Tallinn 2021

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Aivar Mägi

17.05.2021

Annotatsioon

Käesoleva lõputöö eesmärk on autori tööandja poolt etteantud tingimusi, kitsendusi ja tulevaste klientide nõudeid arvestades klient-server arhitektuuril põhineva rakenduse arendus, mis peab lihtsustama väliste osapoolte ligipääsu ettevõtte ressurssidele ja protsessidele. Lähtetingimustena tuleb kasutada tööandja omanduses olevat laohaldustarkvara IBM ® Sterling Warehouse Management System ning sellega integreeritud rakenduse lihtsat prototüüpi.

Lõputöö käsitleb rakenduse arendusprotsessi kulgu, mille korral on esmalt hinnatud olemasolevat olukorda, seejärel valitud sobilik arendusmetoodika ning teostatud tehniline lahendus.

Tulemuseks on rahvusvahelise kasutajaskonna poolt kasutatav veebipõhine klient-server rakendus, mis sisaldab kõiki algselt planeeritud funktsionaalsusi, täiendavaid muudatusi ning parendustöid. Täidetud on kõik tööandja poolsed lähtetingimused ja kitsendused. Rakenduse veebiliides on vastavalt tagasisidele kasutajate jaoks arusaadav ning piisavalt mugav. Käsitletud arendustööd on teostatud koodi lihtsust, komponentide korduvat kasutust, rakenduse laiendatavust ja tarkvara arenduse parimaid praktikaid arvestades.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 5 peatükki, 1 joonist, 3 tabelit.

Abstract

Developing Client-server Architecture Application for Warehouse Resource Management

The purpose of this thesis is to develop a client-server application, that shall simplify access to corporate resources for external users. There have to be taken into account conditions and restrictions from author's employer and requirements of future end-users. The starting condition is to use warehouse management software IBM ® Sterling Warehouse Management System owned by employer and a lightweight prototype.

The thesis deals with the progress of the application development process, with the existing situation assessed first, followed by the suitable development methodology selection and a description of the technical solution implementation.

The outcome is a web application based on client-server architecture, which is used by international clients. Application contains all planned functionalities, additional changes and improvements. All initial conditions and restrictions are met. User interface of the application is reported to be easy to understand for end-users and is sufficiently comfortable. Development work has been carried out, taking into account simplicity of code, component reuse, extensibility of the application and the best practices of software development.

The thesis is in Estonian and contains 31 pages of text, 5 chapters, 1 figures, 3 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* ehk rakendusliides
CSS	*Cascading Style Sheets* ehk märgistuskeel veebilehtede kujunduse loomisel
CSV	*Comma Separated Values* ehk formaat, milles andmed on üksteistest eraldatud kindlate märkidega, näiteks semikoolonitega või komadega
ERP	*Enterprise Resource Planning* ehk ettevõtte ressursside planeerimine
HTML	*HyperText Markup Language* ehk märgistuskeel veebilehtede loomisel
HTTP	*HyperText Transfer Protocol* ehk hüperteksti edastusprotokoll
JSON	*JavaScript Object Notation* ehk alternatiivne andmevahetusformaat XML-le, põhineb skeemivabal võtme/väärtus paaridel
JWT	*Json Web Token* ehk avalikule standardile RFC 7519 vastav signeeritud JSON formaadis sõnum turvaliseks andmetevahetuseks
Mall	Sterling päringu tegemiseks vajalik sõnumi mall XML formaadis
MVP	*Minimum Viable Product* ehk minimaalne töötav toode
POJO	*Plain Old Java Object* ehk Java spetsifikatsioonile vastav objekt
SASS	*Syntactically Awesome Style Sheets* ehk märgistuskeel, mis laiendab CSS võimekust
SSO	*Single Sign-On* ehk ühekordne sisselogimine paljudesse eri rakendustesse
Tagaserver	*back-end server* ehk klient-server arhitektuuris serverrakendus
URL	*Uniform Resource Locator* ehk internetiaadress
UX	*User Experience* ehk veebiliidese kasutuskogemus
Veebiliides	*front-end server* ehk klient-server arhitektuuris klientrakendus
XML	*eXtensible Markup Language* ehk struktuuriga märgistuskeel info jagamiseks infosüsteemide vahel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

E-kaubandusel on maailma majanduses üha suurem roll. Eesti e-kaubanduse liit toob välja, et Eesti sisene e-kaubanduse käive kasvas 2020. aastal vaatamata e-teenuste ära kukkumisele 15% ehk üle 110 miljoni euro võrra 41.

Uuringutest selgub, et kui 2020. aasta märtsi algus moodustas e-ostjate osakaal kogu kaubagrupi ostjatest enam kui poole viies kaubagrupis, siis 2020. aasta aprillis ületas see 50% piiri juba üheksas kaubagrupis. Lisaks seonduvalt ülemaailmse pandeemiaga ning eriolukorraga kasvas e-kaubandus Euroopa Liidus 27% ning 70% e-ostjatest kavatseb e-ostlemise harjumusega jätkata ka peale eriolukordade lõppu 41.

E-kaubandusega tegelevad ettevõtted vajavad oma äritegevuses laoressursside haldamiseks kergelt arusaadava veebiliidese ja taskukohase hinnaga kasutajasõbralikku ERP (*Enterprise Resource Planning*) tarkvara. Käesoleva lõputöö autori tööandja poolt koostatud uuring toob välja, et peamiselt suurte ettevõtete tarbeks arendatud ERP tarkvara on kulukas omandamise, hooldamise ja töötajate koolitamise poolest. ERP tarkvara võib sisaldada funktsionaalsuseid, mida kasutaja oma äritegevuses ei vaja. Samas võivad puududa funktsionaalsused, mida kasutaja oma äritegevuses vajab. Kliendid on ka leidnud, et baas platvormile täiendavate funktsionaalsuste arendamine on ajamahukas ja kulukas. Autori tööandja on rahvusvaheline logistikaettevõte.

Käesoleva lõputöö probleem seisneb selles, et tööandjal puudub laoressursside haldamiseks kergelt kasutatava veebiliidesega, klientide ootustele vastavate funktsionaalsustega ja taskukohase hinnaga kasutajasõbralik ERP tarkvara.

Käesoleva lõputöö eesmärk on tööandja poolt etteantud tingimusi, kitsendusi ja lõppkasutajate nõudeid arvestades klient-server arhitektuuril põhineva rakenduse MVP (*Minimum Viable Product*) arendus, mis lihtsustab väliste osapoolte ligipääsu ettevõtte ressurssidele ja protsessidele.

Lähtetingimustena tuleb kasutada tööandja omanduses olevat laohaldustarkvara Sterling ja selle API-dega (*Application Programming Interface*) suhtlevat klient-server arhitektuuril põhinevat rakenduse prototüüpi, mis koosneb NodeJs tagaserverist (*back-end server*) ja Angular veebiliidesest (*front-end server*).

Täiendava nõudena on vajalik olemasoleva rakenduse NodeJs tagaserveri koodi ümber kirjutamine Spring Boot raamistikule. Arenduse jooksul tuleb tagada testkeskkonnas testkasutajatele ligipääs ning arvestada tuleb tooteomaniku ja testkasutajate tagasisidest tulnud soovidega.

Tööandja soovib arendatavat rakendust hakata rentima pilvepõhise teenusena e-kaubandusega tegelevatele ettevõtetele, sest arendatav laoressursside haldustarkvara laiendab tööandja teenusteportfelli ning ühtlasi võimaldab pakkuda klientidele tööandja lisateenuseid.

Käesolev lõputöö koosneb viiest peatükist: sissejuhatusest, olemasoleva olukorra kirjeldusest, arendusmetoodika valikust, tehnilisest lahendusest ja kokkuvõttest.

Olemasolevat olukorda kirjeldavas peatükis on tutvustatud olemasolevaid rakendusi. Arendusmetoodika peatükis on käsitletud sobilike metoodikate valikuid. Tehnilises osas on vaadeldud rakenduse arendusprotsessi kulgu ja hinnatud saavutatud tulemit. Kokkuvõtvas osas on toodud välja töö eesmärk, tulemus ja peamised järeldused.

2 Olemasolev olukord

Käesolevas peatükis on kirjeldatud ERP tüüpi laohaldustarkvara IBM ® Sterling Warehouse Management System 41 (edaspidi Sterling) ja selle API-dega suhtleva klient-server arhitektuuril põhinevat rakenduse prototüüpi. Samuti on vaadeldud tööandja teisi infotehnoloogilisi lahendusi, mida on võimalik käesoleva projekti korral kasutada.

2.1 Laohaldustarkvara ja sellega ühenduse loomine

Sterling on ERP tüüpi laohaldustarkvara, mis on loodud lahendamaks erinevate ladude võrgustiku haldamisega seotud probleeme. Ladudeks võivad olla näiteks piirkondlikud jaotuskeskused, peamised jaotuskeskused, täitekeskused, laoruumid, tehaselaod ja remondikeskused. Sterling on kõrge konfigureeritavusega, teenustele orienteeritud arhitektuuriga ning selle eesmärkideks on parendada äriprotsesside automatiseerimisega kliendile vajalikku efektiivsust, automatiseerimist ja paindlikkust 41.

Sterling korral on tegemist nn „paksu kliendi" tüüpi tarkvaraga 41, mis on mõeldud suurettevõtetele. Tänu rohketele funktsionaalsustele ja kõrgele konfigureeritavusele on tegemist keeruka rakendusega. Samas tähendab see omakorda, et rakendusel võib olla mõni järgnevalt loetletud puudustest. Keeruka rakenduse arendamine võib olla ajamahukas töö, mis võib kesta aastaid. Selle tõttu ei pruugi kasutatud tehnoloogiad olla rakenduse valmimisel kaasaegsed või uuenduste väljastamine võib toimuda aeglaselt. Samuti võib olla tarkvara ostuhind kõrge, täiendav konfigureerimine ja kasutajate koolitus võib vajada kõrgelt tasustatud spetsialisti abi. Kliendi poolel käivitatav rakendus võib olla aeglane. Kasutajaliides võib olla kasutajate jaoks keeruline, nad ei vaja suurt hulka funktsionaalsusest või nad ei oska seda kasutada. Võib juhtuda, et kasutajaliides pole kasutajasõbralik.

Käesoleva lõputöö skoobis ei käsitleta olemasoleva Sterling tagaserveri ega veebiliidese arhitektuuri.

Kogu suhtlus Sterling API-de või teenustega toimub ainult läbi ühe URL-i (*Uniform Resource Locator*) ja sellele ei lisata täiendavaid HTTP (*HyperText Transfer Protocol*) parameetreid. Sterling teenusteks on tööandja Sterling meeskonna poolt loodud

laiendatud funktsionaalsus, mida tuleb kasutada sarnaselt API päringutele. Päring on struktuurilt alati ühesugune ja selleks vajalike HTTP päise atribuudid on Tabelis 1.

Tabel 1. Sterling API või teenuse päringuks vajalikud HTTP päise atribuudid.

HTTP päise atribuut	**Selgitus**
YFSEnvironment.userId	Kasutajanimi
YFSEnvironment.userToken	Esmase sisselogimise korral tühi, hiljem Sterling poolt koostatud JWT (*Json Web Token*)
YFSEnvironment.password	Kasutaja parool
InteropApiName	Sterling API või teenuse nimetus
InteropApiData	Andmete mall XML (*eXtensible Markup Language*) formaadis, mille korral kõik XML atribuudid on täidetud, näiteks: <GetCustomerList CallingOrgCode="123" MaxRecords="10"> <BuyerOrganization OrganizationName="cgi" QryType="LIKE"/> </GetCustomerList>
TemplateData	Tagastuva vastuse mall XML formaadis, mille korral kõik lisatud tühjad XML atribuudid on päringu tulemuses soovitud andmed, näiteks: <CustomerList> <Customer CustomerID="" CustomerKey="" OrganizationCode=""> <BuyerOrganization OrganizationCode="" OrganizationKey="" OrganizationName=""> <CorporatePersonInfo Company="" FirstName="" LastName="" /> </BuyerOrganization> </Customer> </CustomerList>
IsFlow	N – API kasutamisel, Y – teenuse kasutamisel

2.2 Klient-server rakendus

Olemasolev klient-server arhitektuuril põhineva rakenduse prototüüp koosneb kahest osast: tagaserver ja veebiliides.

Koodi hoiustamiseks on kasutatud Git versioonihaldussüsteemi ning lähtekood on salvestatud koodihoidlasse eraldiseisvatena. Prototüübi arendustööde käigus lisatud kood on Git-s salvestatud *master* harusse. Olemasolevat prototüüpi on arendatud Git ajaloo järgi peamiselt ühe arendaja poolt ajavahemikul 2016. aasta sügis – 2018. aasta kevad.

Tagaserveri korral on kasutatud NodeJs raamistikku. Peamiselt on koodi kirjutamiseks kasutatud JavaScript skriptimiskeelt. Tagaserveri eesmärgiks on edastada kasutaja tehtud päringuid veebiliidesest Sterling API-dele ja tagastada vastus koos andmetega veebiliidesele. Edastatavad sõnumid tagaserveri ja Sterling API või teenuse vahel on koostatud selliselt, et oleks tagatud vajalik andmestik vastavalt Tabelis 1 kirjeldatule. Tagaserver sisaldab veebiliidese tarbeks vajalike päringute tegemiseks koostatud malle.

Kuna tagaserver on ainult vahendaja rollis veebiliidese ja Sterling API-de vahel, siis ei ole kasutatud andmete salvestamist andmebaasi.

Tagaserver kasutab sisseloginud kasutaja tuvastamiseks vahemälus hoitavat räsi, mis on loodud sisseloginud kasutaja andmete ja Sterling API-lt saadetud JWT põhjal. Vahemälu korral on andmed salvestatud võti ja väärtus paaridena, mille korral on räsi kasutatud võtmena ja ülejäänud andmeid väärtusena. Kasutaja välja logimiseks kustutatakse tagaserveri vahemälust veebiliidesest tulnud räsi järgi leitud võti ja väärtus andmepaar. Sterling ei vaja väljalogimise tegevusest teavitust.

Veebiliidese korral on kasutatud vabavaralist Angular 4 disaini- ja arendusraamistikku. Arendatud on lihtsad vaated, mille abil on võimalik kasutaja sisse logida ja teha toodete, ja ostuarvete otsinguid. Samuti on olemas vaade toodete salvestamiseks, kasutades CSV (*Comma Separated Values*) formaadis sisestatud teksti. Tabelis 2 on kirjeldatud olemasolevad veebiliidese vaated ning antud selgitused vaadete eesmärkidest. Kogu olemasolev funktsionaalsus ja võimalused on marginaalne osa sellest, mida kliendid minimaalselt vajavad.

Tabel 2. Veebiliidese olemasolevad vaated.

Vaate nimetus ja tõlge eesti keelde	**Selgitus, vaate eesmärk**
Login - sisse logimine	Kasutajate sisse logimine
Home - esileht	Vaade peale sisse logimist, ikoonid ja viited ülejäänud otsingu- ja detailvaadetele minekuks (vt. 44)
Item search - toote otsing	Otsing, otsingutulemuste vaade ja navigeerimine toote detailvaatele
Item details - toote detailvaade	Uue toote sisestamine, olemasoleva vaatamine ja muutmine (vt. 46)
Item batch upload - toodete salvestamine	Toodete üleslaadimine, kasutades CSV formaadis andmeid (vt. 48)
Purchase orders search - sisseostu arvete otsing	Otsing, otsingutulemuste vaade ja navigeerimine sisseostu arve detailvaatesse
Purchase order details - sisseostu arve detailvaade	Uue sisseostu arve sisestamine, olemasoleva vaatamine ja muutmine

Veebiliidese komponendid on kirjeldatud samanimeliste failidega, millel on kolm erinevat laiendit. Esimeseks on HTML laiendiga presenteerimisfail, mille sisuks on HTML (*HyperText Markup Language*) ja Angular raamistikust tulenev hübriidsüntaks. Teiseks on komponendi stiilifail, mis on salvestatud SCSS laiendiga ja mille sisuks on SASS (*Syntactically Awesome Style Sheets*) süntaksit kasutav kood. Kood genereeritakse veebirakenduse ehitamisel CSS (*Cascading Style Sheets*) formaati. Kolmandaks on TS laiendiga äriloogikat sisaldav fail, mis kasutab TypeScript süntaksit ning sisaldab viiteid komponendi HTML ja SCSS failile.

Veebiliidese komponentide presenteerimisfailide arenduse lihtsustamiseks on kasutatud ühtset eeldisainitud CSS stiili välisest paketist Bootstrap ja mõningaid eeldisainitud vidinaid välisest paketist NGX-Bootstrap.

Veebiliides kasutab vajalike andmete (klassifikaatorid, sisseloginud kasutaja räsi, valitud ettevõte, valitud keel) hoiustamiseks veebilehitseja kohaliku salvestuspesa funktsionaalsust.

Peale kasutaja sisselogimist salvestatakse tagaserverist saabunud sisseloginud kasutaja räsi veebilehitseja salvestuspessa. Räsi lisamine HTTP päisele enne päringu teostamist tagab kasutaja tuvastuse tagaserveris ning Sterling API päringu teostamise.

Kasutaja väljalogimise korral toimub räsi kustutamine tagaserveri vahemälust ja veebilehitseja salvestuspesast. Kasutaja räsi puudumise korral veebilehitseja salvestuspesast toimub kasutaja automaatne suunamine sisselogimise vaatele.

Veebiliideses on lahendatud veebilehtede tõlkimine kahte keelde, kasutades Angular raamistikus olevat tõlgete funktsionaalsust. Vaadete genereerimisel leitakse tõlkefailidest soovitud tõlked vastavalt tõlke koodile ja valitud keelele. Tõlkefailid on salvestatud JSON (*JavaScript Object Notation*) formaadis.

Angular raamistikus on andmete edastamine lahendatud ülemkomponendilt alamkomponendile või vastupidi. Samas puudub funktsionaalsus andmete saatmiseks kõikidele vaatel kuvatud komponentidele. Selleks on veebiliideses arendatud sisemine andmete edastamise funktsionaalsus, kasutades välist vabavaralist teeki RxJS 41.

RxJS teegi abil on võimalik edastada andmeid vajalikule komponendile. Näiteks rippmenüüst valiku muudatuse korral toimub sõnumi edastamine. Selle tulemusena käivitub loogikafunktsioon komponendi TS loogikafailis, mis asub mitu kihti kõrgemal rippmenüü komponendist. Ilma RxJS andmete edastamise funktsionaalsuseta on vajalik andmete saatmine komponendilt komponendile rohkemates failides. Sellisel juhul on arenduse teostamine keerulisem ja ajamahukam ning väheneb koodi loetavus.

2.3 Tööandja teised infotehnoloogilised lahendused

Projekti dokumentatsiooni haldamiseks ja meeskonnaga jagamiseks saab kasutada tarkvaraettevõtte Atlassian valmistatud pilvepõhist toodet nimega Confluence. Projekti seisu saab jälgida sama ettevõtte poolt loodud pilvepõhise keskkonna Jira Software abil. Samuti saab Jira-sse sisestada arenduspileteid, eepikuid ja hallata sprinte. Koodi versioonihaldus toimub sama ettevõtte poolt loodud keskkonnas Bitbucket, mille puhul on tegemist Git versioonihaldustarkvaral kasutava veebiliidesega.

Peale kohalikus masinas arendatud muudatust on võimalik rakendus käivitada virtuaalserveril põhinevas arendus- või testkeskkonnas. Automaatseks koodi ehitamiseks ja virtuaalserveris käivitamiseks saab kasutada vabavaralist veebirakendust Jenkins. Sprindi lõppedes paigaldatakse uus klient-server rakenduse versioon toodangukeskkonda, kuhu on ligipääsuõigus ainult süsteemiadministraatoril.

3 Arendusmetoodika valik

Projekti käivitamiseks on autori tööandja teinud kokkulepped mitmete tulevaste klientidega ning kaardistanud nende vajadused. Samuti on välja selgitatud vajalike funktsionaalsuste minimaalne hulk. Ärinõuete kitsenduseks tuleb täita kokkuleppeliste klientide vajadused uue tarkvara osas. Klientide soov on samuti jälgida arenduse kulgu, osaledes ise testkasutaja rollis.

Projekti on kaasatud ladude valdkonna spetsialist, kes on käesoleva arenduse jooksul tooteomaniku rollis. Tooteomaniku ülesandeks on meeskonna suunamine tööülesannete täitmisel. Samuti vastutab ta selle eest, et arendustööd pakuvad klientidele maksimaalset väärtust.

Autori ja tema meeskonna ülesandeks on valida arendustööde metoodika. Kasutatav metoodika peab arvestama lähteülesannet, kitsendusi ja klientide tagasisidet. Lisaks metoodikale tuleb välja töötada protsess, kuidas metoodikat kasutades arendustöid teostada. Arenduse käigus tuleb pidevalt suhelda tooteomanikuga ning testkasutajate rollis olevate klientidega.

Vajalikele muutustele ja esile kerkivatele probleemidele tuleb kiirelt reageerida ja leida lahendused. Arvestada tuleb ülesannete võimalike muudatustega ning selle tõttu peab kogu arendustööde protsess olema paindlik. Tarkvara arendamisel tuleb teha omapoolseid ettepanekuid parimate lahenduste leidmiseks või kasutusmugavuse tõstmiseks.

Enne arendustööde alustamist tuleb analüüsida ja otsustada, millist arendusmetoodikat kasutada. Arvestama peab projekti skoobi, lähteülesannete ja nende olulisusega ning meeskonna suuruse ja mõistlike arendustööde tähtaegadega.

Kõik arendusmetoodikad pakuvad erinevaid võimalusi arendustööde läbiviimiseks. Õnnestunud arendusmetoodika valik võimaldab ennetada probleeme ja reageerida muutustele paindlikult. Ebaõnnestunud arendusmetoodika valik võib põhjustada tähtaegade ületamise, mis omakorda tingib suured kulutused või isegi terve projekti läbikukkumise 41.

Autori tööandja arendusmeeskonnad ei kasuta kose meetodit, sest antud meetod ei võimalda enne tööde alustamist ette aimata arenduse käigus tekkivaid probleeme ja muudatusi. Samuti tuleb arendustöid teha klienti protsessi kaasates, kuna kliendi soovid ja vajadused võivad muutuda arendusprotsessi käigus.

Seega käesoleva projekti korral ei ole kose meetod piisavalt paindlik ja kliendi huve arvestav. Seda kinnitab Standish Group statistiline uuring ajavahemikul 1994. aasta – 2018. aasta. Uuring leiab, et kose meetodi kasutamise korral ebaõnnestuvad projektid umbes 1.5 korda enam, kui agiilsete arendusmetoodikate kasutamise korral. Antud uuring toob välja, et agiilsed metoodikad on kose meetodiga võrreldes projekti läbimise vaates umbes kaks korda edukamad 41.

Selle tõttu langeb valik agiilsetele arendusmetoodikatele, mille parimaid praktikaid saab kohandada käesoleva projekti tarbeks. Samas ei ole otstarbekas kasutada ühte kindlat agiilset metoodikat, sest lisaks positiivsetele külgedele on igal metoodikal ka omad negatiivsed küljed.

Näiteks Scrum arendusmetoodika korral ei tohi sprinti kunagi ühtegi tööd juurde võtta, sest see seab sprindile jäigad piirid. Samuti LHV IT-arenduse juht Rainer Tikk leiab, et ülesannete hindamise tulemusena tekkinud stooripunkte hakatakse tõlgendama väärtusena. Selle tulemusena hakatakse madalamate hinnangutega töid eelistama sprintide planeerimisel, mis omakorda toob kaasa toote kvaliteedi vähenemise 41.

Samuti on Kanban arendusmetoodika korral raske jälgida terve projekti hetkeseisu, sest ilma sprintideta puuduvad eesmärgid ja arendajate produktiivsus võib väheneda.

XP (*eXtreme Programming*) arendusmetoodika keskendub pigem koodi kvaliteedile kui projektijuhtimisele ning paaris arendamine võib võtta rohkem aega kui tavaline arendus.

Siiski teeb lõputöö autor meeskonnale ettepaneku kasutada peamiselt Scrum arendusmetoodikat, kohandades seda teiste arendusmetoodikate elementidega.

Enne sprindi algust toimub sprindi planeerimine, mille eesmärk on hinnata varem koostatud arenduspiletite keerukust. Iga järgmise sprindi planeerimisel tuleb eesmärgiks võtta teatud hulga eelnevalt hinnatud arenduspiletite lahendamine. Nii on hiljem võimalik hinnata, millise keerukusega ülesandeid meeskond keskmiselt sprindi jooksul arendada

suudab. Oluline on sprinti võtta erineva keerukuse tasemega ülesandeid, mitte ainult kõige lihtsamaid. Kui keerukuse tõttu võtab ülesande lahendamine rohkem aega kui üks sprint, siis tuleb ülesannet tükeldada.

Autor teeb ettepaneku kasutada kahe nädala pikkuseid sprinte, sest testkasutajad vajavad aega testimiseks ja tagasiside andmiseks. Kindlate intervallidega sprindid tagavad süstemaatilised uuendused ja annavad klientidele hetke olukorrast ülevaate.

Korraldada tuleb igapäevased *standup*-e, sest see annab meeskonnasisese ülevaate kõigi liikmete tegevustest ning võimalikest tekkinud probleemidest. Kiire probleemist teada saamine annab meeskonnale võimaluse leida parim lahendus.

Sprindi lõppedes tuleb korraldada traditsiooniline retro koosolek, vaadata üle senised tööd ja probleemid ning arutada, kas midagi on võimalik parendada.

Ühtlasi leiab autor, et peale ülesannete lahendamist on otstarbekas teha keerulisema koodi tutvustus teistele meeskonnaliikmetele. Tutvustus võimaldab teisel arendajal kontrollida lahenduse vastavust ülesandele ja meeskonnasisesele koodi stiilile. Samuti jagada meeskonnasiseselt loodud lahendust, hinnata testide piisavust ning leida loogikavigu koodis. Tutvustuse vastuvõtu korral paigaldatakse lahendus arendus- või testkeskkonda. Tutvustuse mittevastavuse korral tuleb lahendust täiendada ja uuesti kontrollida.

Kuna olemasoleva prototüübi testimise käigus on kliendid esitanud funktsionaalsuste osas täiendavaid arenduse soove, kuid tööandja soov on arendustöödega alustada võimalikult kiiresti, siis teeb autor ettepaneku arenduspiletite loomisel kasutada eepikuid nii tagaserveri kui ka veebiliidese tarkvaraliste uuenduste tarbeks.

Meeskond arvestab autori ettepanekuid arendusmetoodika ja -protsessi osas ning analüütik edastab metoodika tööandjale. Peale tööandja poolse nõusoleku saamist alustab autor ja tema meeskond arendustöödega.

Arendustööd algavad eepikute loomisega, milles osaleb terve meeskond: analüütik, vanemarendaja, testija ja autor. Arenduspileteid vormistati kokku 123, mis omakorda on jaotatud eepikutesse (vt. Tabel 3).

Tabel 3. Klientide vajadustest koostatud eepikud.

Eepiku nimetus ja tõlge eesti keelde	**Selgitus, eepiku eesmärk**
General improvements - üldised uuendused	Üldised uuendused või kasutusmugavuse parendused üle terve veebiliidese, näiteks käsitsi disainitud rippmenüü asendamine eeldisainitud rippmenüü vastu
Home view improvements - esilehe uuendused	Esilehe uuendused, kuhu kasutaja suunatakse peale edukat sisenemist süsteemi, näiteks dünaamiliste teadete lisamine esilehe jalusesse
Item search view improvements - tooteotsingu vaate uuendused	Toodete otsingu vaate parendused, näiteks otsingutulemuste kuva täiendamine laoseisu, lao jm oluliste andmetega
Item details view improvements - toote detailvaate uuendused	Toote detailvaate parendused, näiteks erinevate väljade lisamised
Item batch upload improvements - toodete üleslaadimise vaate uuendused	Toodete üleslaadimise vaate uuendused, näiteks toodetele valideerimise lisamine
Purchase order search view improvements - sisseostu arve otsingu vaate uuendused	Sisseostu arvete otsingu vaate täiendused, näiteks otsingu parendamine täiendavate valikute järgi
Purchase order details view improvements - sisseostuarve detailvaate uuendused	Sisseostu arve detailvaate parendused, näiteks täiendavate päringute käivitus rippmenüü valiku korral
Sales order search view - müügiarve otsingu vaade	Väljamüügi arvete otsingu vaate lisamine
Sales order details view - müügiarve detailvaade	Väljamüügi arve detailvaate lisamine
Back-end upgrades - tagaserveri uuendused	Tagaserveri uuendused seoses NodeJs ümber kirjutamine Spring Boot raamistikule
Front-end upgrades - veebiliidese uuendused	Veebiliidese uuendused, näiteks Angular või alamteekide versioonide uuendamine

4 Tehniline lahendus

Käesolevas peatükis on kirjeldatud tööandja infotehnoloogilisi lahendusi, mida saab kasutada arendustööde käigus. Välja on selgitatud vajalikud välised laiendused ja teegid ülesande lahendamiseks. Kirjeldatud on nii autori kui ka meeskonna teiste liikmete tegevusi projekti jooksul. Antud on ülevaade arendustööde ajal testkasutajatelt saadud tagasiside põhjal tehtud erinevatest muudatustest. Peatüki lõpus on hinnatud arendustööde tulemust ning analüüsitud valminud rakenduse laiendatavust.

Käesoleva projekti tehnilise lahenduse korral tuleb esmalt valida rakenduse arhitektuur. Traditsioonilise mitmekihilise arhitektuuri korral koosneb infosüsteem kolmest kihist: esitlusloogika kiht, äriloogika kiht ja andmekiht. Käesoleval juhul pole andmekihi vajadust, sest kõik andmed saadakse läbi Sterling API. Selle kaudu loetakse sisse loginud kasutaja tarbeks seadistatud klassifikaatorid, mida kasutatakse näiteks rippmenüüde tarbeks. Kasutaja saab ainult salvestada veebiliideses kasutatava keele, mille väärtus salvestatakse veebilehitseja kohalikku salvestuspessa. Kõik ülejäänud muutujad, mille salvestuseks on vajalik veebilehitseja salvestuspesa, kustutatakse peale kasutaja välja logimist.

Eelnevate aspektidele tuginedes teeb autor meeskonnale ettepaneku kasutada kahekihilist klient-server arhitektuuri, mis koosneb äri- ja esitlusloogika kihtidest.

4.1 Vajalikud välised laiendused ja teegid

Käesoleva lõputöö skoobis ei ole analüüsida tagaserveri ega veebiliidese tehnoloogiate valikut, sest lähteülesandeks on olemasoleva prototüübi tagaserveri ümber kirjutamine Spring Boot raamistikule. Samuti tuleb veebiliidese tehnoloogiaks jätta Angular.

Mõlemad raamistikud on tööandja ettevõttes laialdaselt kasutuses ja seetõttu on need käesoleval juhul kohustuslikud. Vaadeldud on, milliseid täiendavaid tehnoloogiaid, teeke, uuendusi ja vidinaid vajavad nii tagaserver kui ka veebiliides.

4.1.1 Tagaserver

Käesolevaga on eesmärk kasutada tagaserveris nii palju aktiivses arenduses olevaid, põhjalikult testitud, kogukonna poolt palju kasutatud ja heaks kiidetud vabavaralisi

väliseid alamteeke, kui võimalik. Sellega on võimalik vähendada võimalike arendusvigade teket ja ajakulusid programmeerimise osas.

Tagaserveri ehitamiseks, alamteekide halduseks ja pakendamiseks tuleb kasutada kas Maven või Gradle tööriista, sest need võimaldavad luua ja hoida kindlat projekti struktuuri. Antud juhul kasutatakse Maven tööriista, sest projekt ei vaja keerukat seadistust mida Gradle võimaldab. Samuti on Maven arendajate seas enim kasutatud.

Lähteülesandena võetakse tagaserveri arendusel kasutusele vabavaraline raamistik Spring Boot, mille korral toimub arendamine programmeerimiskeeles Java.

Spring Boot võimaldab kasutada nii eraldi alamteeke kui ka alamteekidest koosnevaid kogumikke, mida nimetatakse starteriteks. Käesoleval ajal on erinevaid startereid enam kui 50, mis sisaldavad nii Spring raamistiku alamteeke kui ka väliseid teeke.

Starter kogumike abil saab kergelt lisada rakendusele palju funktsionaalsust vastavalt arendaja vajadustele. Kuna võimalike vigade vähendamiseks on soov seadistada tagaserveri alamteeke võimalikult vähe, siis on otstarbekas kasutada vajaliku funktsionaalsuse saavutamiseks järgnevaid Spring Boot startereid.

Projektis kasutatavad starterid on: spring-boot-starter-web (Spring MVC, REST kontrollerid, veebiserver), spring-boot-starter-security (Spring Security raamistik, AOP), spring-boot-starter-validation (Hibernate Validator valideerimisraamistik), spring-boot-starter-test (Spring Test, Junit, AssertJ, Mockito teegid), spring-boot-starter-actuator (rakenduse jälgimise töövahendid) ja spring-boot-starter-cache (vahemälu).

Domeeniobjektide kirjeldamine on Java puhul tegevus, mille korral tuleb palju kirjutada sarnaseid *get, set, equals, toString* ja konstruktor meetodeid. Traditsionaalse (*boilerplate)* koodi kirjutamise vältimise eesmärgil soovib autor kasutada vabavaralist teeki Project Lombok, mis võimaldab annotatsioone kasutades kirjeldada sama funktsionaalsust. Sellega väheneb käsitsi kirjutatud koodi hulk ja samuti suureneb koodi vea- ja töökindlus, sest palju testitud vabavaralise tarkvara korral on suur hulk vigu juba leitud ja parandatud.

JWT töötlemiseks on tarvis teeki, mis suudab töödelda erinevaid JWT krüpteerimisalgoritme ja on pidevalt uuenev. Võttes arvesse ametlikul JWT veebisaidil

soovitatud JWT töötlemise teeke, otsustab autor kasutada selleks io.jsonwebtoken/jjwt tarkvara.

XML formaadis andmete korral on tarvis andmeid viia Java domeeniobjektide kujule ning selleks kasutatakse com.fasterxml.jackson.dataformat/jackson-dataformat-xml teeki. Selle abil on võimalik annotatsioone kasutades kergelt teisendada XML struktuuris andmeid Java POJO (*Plain Old Java Object*) domeeniobjekti kujule ja vastupidi.

4.1.2 Veebiliides

Veebiliidese korral on samuti eesmärk kasutada arendusvigade ja -töödele kuluva aja vähendamiseks võimalikult palju vabavaralisi, kogukonna poolt testitud ja aktiivselt arendatavaid teeke ja vidinaid.

Vastavalt lähteülesandele jääb veebiliidese tehnoloogia samaks – Angular. Arendustööde käigus tuleb Angular-i versioone ainult uuendada.

Käesoleval juhul on tarvis vähesel määral lisada või uuendada väliseid teeke ja vidinaid, kuna Angular on raamistik, mis juba sisaldab rohkelt funktsionaalsuseid. Olemasolevatest teekidest on vaja uuendada näiteks TypeScript, RxJS, Bootstrap, NGX-Bootstrap, Karma versioone.

Hilisemate tööde käigus lisatakse täiendavalt mitmeid teeke visuaalsete komponentide lisamiseks või välja vahetamiseks (ng-select, ngx-toastr), arendust lihtsustavaid väliseid tööriistu (lodash, moment) või spetsiifilist funktsionaalsust lisavaid tööriistu (powebi-client).

4.2 Meeskonna teiste liikmete tegevused

Lõputöö autori arendusmeeskonda kuuluvad analüütik, vanemarendaja ja testija rollis olevad meeskonnaliikmed. Nendel kõigil on käesolevas projektis oma ülesanded. Liikmed töötavad projektis kas osaliselt või kogu projekti aja jooksul.

Analüütiku rollis oleva meeskonnaliikme eesmärgiks on terve projekti jooksul teha koostööd tooteomaniku ja testkasutajatega. Testkasutajatele tuleb tutvustada uusi arendustöid, dokumenteerida tagasisidet ja ettepanekuid ning vormistada uusi arenduspileteid *backlog*-i. Analüütiku ülesannete hulka kuulub ka testkeskkonda

paigaldatud uute arendustööde testimine, sest tarkvara testija osaleb projektis ainult kontroll- ja ühiktestide loomiseni.

Vanemarendaja rollis oleva meeskonnaliikme eesmärgiks on tagaserveri funktsionaalsuse ümber kirjutamine Spring Boot raamistikule. Vanemarendaja ei panusta projektis täistööajaga, sest osaleb korraga mitme projekti arendustöödes. Git logi väljavõtte järgi on vanemarendaja osalenud projektis alates 2018. aasta sügisest kümne kuu jooksul. Selle ajal toimub rakenduse ning Sterling vahelise autentimise välja töötamine, kuid lahenduse teostamine venib ettevõtte arhitektuursete autentimistehnoloogiate muutuste tõttu. 2019. aasta kevadel lahkub vanemarendaja meeskonnast peale tagaserveri Spring Boot raamistikule üle viimist. 2020. aasta talvel ühineb meeskonnaga uus meeskonnaliige, kes osaleb tarkvara arendajana projektis kahe kuu jooksul.

Tarkvara testija peab välja töötama veebiliidese ühiktestide lahenduse, mis peamiste funktsionaalsuste töökorras oleku kindlaks tegemise eesmärgil automaatselt käivitub peale koodi ehitamist. Samuti on testija eesmärgiks on välja töötada koormustesti loomine. Koormustest selgitab välja rakenduse võimekuse suure kasutajate arvu ja süsteemi koormuse korral. Testija osaleb projektis alates 2019. aasta kevadest kahe kuu jooksul, olles igapäevaselt hõivatud mitme projektiga. Hiljem on autori ülesandeks jätkata ühiktestide täiendamist.

4.3 Autori tegevused projektis

Autor ühineb projektiga 2019. aasta märtsis, kui on valminud väliste kasutajate autentimiseks täiendavad arendustööd Sterling meeskonna poolt. Peale projektiga ühinemist on autorile usaldatud terve projekti arendus, sest meeskonnast on lahkunud vanemarendaja.

Autor on enam kui 1.5 aastase perioodi jooksul ainus rakenduse arendaja ning see annab võimaluse kasutada kõiki kõrgkoolis omandatud teadmisi. Samuti pakub ainsa arendajana projektis olemine väljakutseid, sest tehnilise probleemi korral tuleb ise lahendus leida.

Varasemale töökogemusele tuginedes teeb autor ettepaneku meeskonnasiseselt hakata kasutama ühtset stiili veapileti dokumenteerimisel. Selleks tuleb veapileti kommentaari väljale lisada kirjeldus, milles viga seisnes, mida tehti vea parandamiseks ja kuidas

loodud arendust testida (vt. 43). Antud stiili korral paraneb veapileti lahendusest aru saamine. Täiendavalt on soovituslik kasutada veapileti nimetust või veapileti URL-i Git *commit*-i alguses, mis võimaldab hiljem hiirega klikkimisel veebilehitseja avamist ja kiiret navigeerimist veapileti Confluence lehele.

4.3.1 Igapäevased ülesanded

Arendustöödes on tegevusi, mida tuleb teha või millele tuleb lisaks koodi kirjutamisele tähelepanu pöörata igapäevaselt. Näiteks rakenduse ühiktestide täiendamine, mille eesmärgiks on saada tagasisidet funktsionaalsuste toimimiste osas. Koodi lihtsustamine loetavuse parendamise eesmärgil. Suhtlemine kliendi, tooteomaniku või meeskonnaga. Tööde planeerimisel või teostamisel tuleb välja pakkuda omapoolsed lahendused probleemide realiseerimiseks. Vigade avastamise korral tuleb luua täiendavad piletid nende kõrvaldamiseks olemasolevas või järgnevas sprindis.

Peale projektiga ühinemist leiab autor, et olemasoleva veebiliidese arendamisel ei ole tähelepanu pööratud komponentide korduvkasutusele, sest laialdaselt on kasutatud vana koodi kopeerimist. Autor arvab, et kui sama funktsionaalsus või veebiliidese mallid on kasutuses rohkem kui ühes kohas, tuleb leida lahendus, kuidas panna kood tööle kõikjal, seda ainult ühe korra kirjutades. Sellest tulenevalt võtab autor endale eesmärgiks igapäevastes töödes ja lahenduste teostamisel arvestada koodi korduvkasutusega.

4.3.2 Logimise funktsionaalsuse lisamine

Peale tagaserveri arendust Spring Boot raamistikule on autori esimeste ülesannete seas rakendusele logimise funktsionaalsuse lisamine tagaserveris. Kuna peamiselt toimub tagaserveris päringute edastamine Sterling API-le ja vastupidi, on eesmärgiks luua üldine rakenduse logimine nii, et kõik väljuvad ja tagastuvad päringud oleksid logidest leitavad. Logides peab olema tuvastatavad Tabel 1 näidatud atribuudid.

Logide sisu mõistmiseks on vaja teada kolme päringu parameetrit: milline kasutaja, millise ettevõtte esindaja rollis ja millises veebiliidese vaates. Tagaserveri või tagaserverisse saabunud Sterling API sisemise vea korral tuleb vea pinu näidata logides, kuid ei tohi jõuda veebiliidesele suunduvale päringule.

Väljuva päringu HTTP päisele on juba lisatud sisseloginud kasutaja räsi vastavalt peatükis 14 kirjeldatule ning veebiliidest ei ole vaja muuta. Kasutaja ID ja ettevõtte ID

on varasemalt päisesse lisatud, kasutades Angular *HttpInterceptor* funktsiooni. Samuti veebilehitseja poolt lisatakse automaatselt kõikidele päringutele HTTP päis *Referer*, mille väärtuseks on päringu teostamise ajal olnud aktiivse vaate URL.

Tagaserver kasutab Sterling API päringute tegemiseks Spring raamistikust tulenevat *RestTemplate* mustrit. Sellisel juhul on võimalik rakendusest väljuv *RestTemplate* päringut eelnevalt lugeda, kasutades Spring raamistiku *ClientHttpInterceptor* funktsionaalsust ja kirjutades üle *intercept* meetodi. Selles meetodis on automaatselt teada väljuva päringu andmed, päised ja päringu meetod. Päringu vastuse saabudes on võimalik teha vastuse andmete või vea logimine. Lisaks tuleb üle vaadata olemasolevad päringud, lisada täiendavad kontrollid ning teha muudatused, et vältida vea pinu kuvamist veebiliidese kasutajale.

Autor seadistab edukalt lõppenud päringu logimise *Debug* tasemele ja veaga lõppenud päringu logimise *Error* tasemele. Logimise tase on seadistatav rakenduse peamises konfigureerimisfailis *application.properties*. Seadistada saab logimist kogu rakendusele või pakettide kaupa erinevalt.

Esmalt seadistati rakendus logisid salvestama virtuaalserverisse. Sellisel juhul on vajalik logide nägemiseks või vanade logifailide kustutamiseks virtuaalmasinasse sisse logimine. Selle lihtsustamiseks on administraatorid juurutatud tsentraalse logimise võimekuse, mis kasutab LogStash ja ElasticSearch põhist andmebaasi ja Kibana veebiliidest logide kuvamiseks.

Kuid tsentraalsesse logiserverisse andmete saatmine nõuab vajaliku funktsionaalsuse arendust või olemasoleva teegi kasutamist. Autor leiab teise meeskonna poolt arendatud teegi, mis võimaldab logi ridade väljade teisenduse JSON formaati ning logide saatmise ettevõtte kasutatavasse tsentraalsesse logiserverisse. Peale teegi lisamist ja LogStash URL-i seadistust edastatakse logid krüpteeritud kujul virtuaalserverist tsentraalsesse logiserverisse. Selle tulemusena on võimalik rakenduse logisid läbi Kibana veebiliidese kergelt jälgida ning üle kahe kuu vanad logid kustutatakse automaatselt tsentraalse logiserveri poolt.

Järgnevalt kirjeldab autor arendustööde käigus ette tulnud huvitavamaid ja enim väljakutseid pakkuvaid ülesandeid. Nendeks on dünaamilise andmete valideerimise

loomine veebiliideses, toodete üleslaadimise funktsionaalsuse ümber kirjutamine ja ettevõtte sisesele kasutajale sisselogimise võimekuse arendamine.

4.3.3 Andmete dünaamiline valideerimine

Kuna kõik väljad, mida kasutaja veebiliidese vaates sisestab või muudab, peavad olema valideeritud, siis peab olema takistatud vigaste andmete sattumine tagaserverisse või sealt edasi Sterling keskkonda.

Oluline on ka teada, et veebiliidese kasutaja, kes omab vajalikke oskusi, saab soovi korral muuta tagaserverisse suunduvaid andmeid. Selle tõttu tuleb andmed eelnevalt üle kontrollida esmalt veebiliideses ning leitud välja vea korral tuleb seda näidata koheselt veebiliidese vaates. Tagaserveris välja valideerimisel tekkinud vea korral tuleb veateade edastada veebiliidesele ja veateadet näidata kasutajale.

Käesoleva ülesande esmaseks ajendiks on vajadus teha andmete valideerimine veebiliideses ja tagaserveris nii, et vigaste andmete jõudmine Sterling keskkonda on välistatud.

Tavaliselt teostatakse valideerimine klient-server rakenduse korral selliselt, et väljade nõuded kirjeldatakse eraldi tagaserveris ja veebiliideses. Käesoleval juhul on probleemiks veebiliidese väljade valideerimine, sest iga malli vormi genereerimisel tuleb lisada väljadele täiendavad kontrollid. Väljade rohkuse ja täiendavate kontrollide mahu tõttu võib kõigile veebiliidese mallidele ja vaadetele valideerimise lisamine olla aega nõudev. Lisaks tuleb veebiliideses veateated tõlkida ning tagaserveri väljade muudatuste korral tuleb teha täiendusi, mis omakorda võib tõsta arendaja poolse vea tõenäosust.

Lahenduse esimeses etapis tuleb lisada täiendused tagaserveri andmeobjektide väljadele ja kontrolleritele. Sellisel juhul käivitub hiljem automaatne valideerimine enne kontrolleri meetodi rakendumist. Spring Boot starter spring-boot-starter-validation sisaldab teeki javax-validation, mille korral on lahendatud JSR380 41 spetsifikatsioonile vastav valideerimine.

Kuna Spring Boot raamistikuga seotud starterid on pidevas arenduses ja töökindlad, otsustab autor kasutada spring-boot-starter-validation starterist tulenevat funktsionaalsust tagaserveri probleemide lahendamiseks.

Neid annotatsioone saab kasutada andmeobjektide väljade piirangute kirjeldamisel (nt *@NotNull, @Min, @Email*) ja hiljem kontrolleris (*@Valid*), kus päringuga tulnud objekti valideerimine käivitub automaatselt. Kui kontrolleri poole pöörduvad andmed ei ole sobilikud, tagastab tagaserver vastuse valideerimisvigadega ja kontrolleri meetod ei käivitu. Annotatsioonidele saab lisada sõnumid välja vea korral, kuid neid ei saa veebiliideses kasutada, sest tegemist on keskkonnaga, mille elemente tuleb kahte keelde tõlkida.

Peale esimese etapi lõpetamist tuleb autoril mõte teostada andmete valideerimine veebiliideses dünaamiliselt, kasutades tagaserveris kirjeldatud andmeobjektide omadusi. Autori mõte on kasutada tagaserveri objektide peegelduse funktsionaalsust (*reflection*). Peegeldusega on võimalik tuvastada iga objekti tüübi ja valideerimise annotatsioonid, koostada objektidest koondobjekt ja edastada see peale kasutaja sisselogimist veebiliidesele. Ning veebiliidese vaadetes on võimalik valideerimisandmeid kasutades teostada vaadete väljade valideerimise seadistused hiljem automaatselt.

Lahendus võimaldab edastada tagaserverist objektide nimed, tüübid ja valideerimise annotatsioonid koos parameetritega koos klassifikaatoritega veebiliidesele peale kasutaja sisselogimist. Kuna peegeldusega tuvastatavate objektide sees võivad olla omakorda objektid, siis objektide andmed tuleb töödelda rekursiivselt. Käesolev dünaamiline lahendus peab tagama, et peale uuendatud tagaserveri või veebiliidese käivitust jõuavad valideerimisandmed veebiliideseni.

Peale kasutaja sisselogimist veebiliidesesse saabuvad väljade valideerimisandmed, mis salvestub veebilehitseja kohalikku salvestuspessa. Hiljem, kui kasutaja navigeerib näiteks toodete lehele, siis loetakse salvestuspesast vajaliku malli tarbeks klassifikaatorid ja valideerimisandmed. Valideerimisandmed lisatakse loogikafaili genereerimise käigus veebiliidese malli vormile. Kui kasutaja tegevusena ei saa väli soovitud parameetritega väärtust, näidatakse selle välja komponendi sees veateate tõlget vastavalt välja nimetusele. Seega olenemata valitud keelest või selle vahetusest on alati vaatel näidatav veateade õiges keeles.

Käesolevaga lahendati probleem, mille eesmärgiks on valideerida andmeid nii veebiliideses kui ka tagaserveris. Antud lahendus võimaldab edaspidi andmeväljade uuenemise korral teha muudatusi ainult tagaserveri domeeniobjektides. Veebiliidese

valideerimine uueneb dünaamiliselt peale kasutaja sisselogimist ja peale vaate genereerimist ilma täiendavate muudatusteta. Samal ajal vähendab lahendus võimalike vigade teket.

Nimetatud lahendus on näide, mille korral saab täita ülesande dünaamiliselt.

4.3.4 Toodete üleslaadimise ümber kirjutamine

Teise suure ülesandena tuleb täiendada toodete üleslaadimise funktsionaalsust. Olemasoleva lahenduse järgi tuleb kasutajal tekstina sisestada CSV formaadis toodete andmed, mis võivad pärineda näiteks välise laosüsteemi ekspordi tulemusena genereeritud CSV formaadis failist.

Sisestada saab maksimaalselt 1000 rida ning esimeseks reaks peavad olema toodete väljade nimetused. Väljade nimetused peavad kattuma tagaserveris kirjeldatud andmeobjektide nimetustega, sest vastasel korral Sterling API päring toote salvestus ebaõnnestub kohustuslike väljade puudumise tõttu.

Peale andmete üleslaadimise käivitust genereeritakse veebiliideses ühekaupa toote objektid, mis saadetakse läbi tagaserveri Sterling API-le. Toote eduka salvestuse või ebaõnnestumise korral toimub järgmise toote genereerimine ja edastamine Sterling API-le. Samas ühekaupa 1000 päringu tegemine ei ole otstarbekas ajalises ega ka API-le langeva koormuse osas.

Toodete üleslaadimise ajal näidatakse kasutajale statistikat, millist toodet hetkel üles laetakse ja mitme toote üleslaadimine lõppes edukalt või ebaõnnestunult. Esialgse hinnangu tulemusena on olemasoleva funktsionaalsuse osas võimalik kasutada ainult nimetatud üles laetud toodete statistikat ning ülejäänud osas vajab terve üles laadimise vaade ümber kirjutamist.

Probleemi lahendus on arendada üleslaadimine CSV failist, kasutades selleks palju tooteid ühe päringuga salvestada võimaldavat Sterling API teenust. Päringu käivitudes loob Sterling keskkond uued või kirjutab üle olemasolevad tooted.

Arendustööde käigus tuleb arvestada kasutusmugavust tõstvate elementidega. Näiteks vaatele tuleb lisada täiendav dokument, milles on kirjeldatud toodete üleslaadimise protsess ning toote võimalike väljade nimetused, tüübid, täiendavad nõuded ja

kohustuslikud väljad. Vaatel peab olema näidatud valitud faili nimi, sest see võimaldab kinnitust saada aktiivse faili õigsuses. Kasutaja peab igal ajal saama oma tegevust katkestada ja protsessi korrata.

Peale CSV faili sisestamist peab veebiliides tuvastama faili veerud, mille nimed kattuvad süsteemis olevate väljade nimedega. Kui sisestatud fail sisaldab rohkem ridu kui eelnevalt veebiliideses seadistatud, siis tuleb andmete töötlemine katkestada. Vastasel juhul võib toimuda veebilehitseja hangumine. Faili andmete töötlemise ajal tuleb kasutajale näidata animatsiooni, mis annab märku pooleli olevast toimingust.

Peale andmete töötlemist tuleb kasutajale näidata toodete eelvaate tabelit, mille ridadeks on esimesed failist loetud 5 rida. Osaline eelvaade võimaldab vältida ülemäära paljudest andmetest tingitud veebilehitseja hangumist ja tabeli vertikaalse osa laotust üle kuvari ekraani. Eelvaate veergudeks peavad olema välja nimed, mis on tõlgitud vastavalt kasutaja valitud keelele.

Korrektsete andmete korral peab kasutaja saama käivitada toodete üleslaadimise. Andmete vea korral peab kasutaja aru saama, millises reas ja veerus asub vigane väli. Peale vigase välja tuvastamist vaates ei tohi olla üleslaadimise käivitusnupp aktiivne. Rohkete vigade arvu korral ei ole otstarbeks kõiki vigu näidata.

Kui andmete vigu ei esine, peab kogu vaade mahtuma ühele *Full HD* resolutsiooniga kuvari ekraanile. Kuna sisestatavate toodete välju võib olla üle 50, tuleb toodete eelvaate tabelit kohandada horisontaalselt liikuva kerimisribaga. Kui kasutaja vahetab rippmenüüs ühe veeru teise vastu, tuleb kõik andmed üle kontrollida. Vigade jätkumisel või uute tekkimisel tuleb veateateid näidata või vigade puudumisel olemasolevad veateated eemaldada.

Võimalik peab olema tooteid salvestada teistsuguste väljade nimetuste korral. Sellisel juhul peab kasutaja kokku viima iga failis sisalduva veeru ja toote tegeliku välja nime tagaserveris. Kui faili esimeses reas sisalduvad väljade nimetused on osaliselt või täielikult identsed tegelike toote objekti väljade nimetustega, tuleb need automaatselt tuvastada. Samuti peab enne toodete salvestusprotsessi käivitust olema võimalik andmete väljade nimetusi muuta.

Kasutusmugavuse tõstmiseks tuleb üleslaadimise protsessi jooksul näidata salvestusnupu läheduses animeeritud ikooni. Samuti tuleb tagaserveris enne Sterling API salvestust teha täiendav valideerimine, sest võimaliku vea korral ei tohi salvestus käivituda ning veebiliidesele tuleb edastada vea põhjus. Täiendav valideerimine tagaserveris on vajalik, kuna toote objekt sisaldab alamobjekte ja välju, mille kohustuslikkus sõltub teiste väljade täitmisest.

Järgmise etapina tuleb otsida CSV faili lugemist võimaldav teek. CSV failist andmete lugemine ei ole ainulaadne probleem ning nimetatud võimekus on varem tehtud ja kasutatav vabavarana. Lahendusena leiab autor teegid csv-parser-generator, csv-parse, csv-parser ja papaparse ning otsustab kasutada viimast. Võrdluse hetkel on papaparse funktsionaalsuste hulga ja lugemiskiiruse poolest üle teistest valikus olevatest teekidest. Samuti on papaparse kiirelt arenev ja laialdaselt kasutatav.

Peale teegi lisamist on võimalik näidata eelvaates veerge, kasutades töödeldud faili esimeses reas kirjeldatud päiseid. Veergudes olevate väljade kohustuslikkuse, andmetüübi võrdluse ja andmete parameetrite valideerimisel on võimalik kasutada alampeatükis 27 kirjeldatud lahendust.

Käesoleva arenduse tulemusena tehti ümber toodete üleslaadimise vaade. Varasem lahendus võimaldas CSV formaadis tekstina andmete kleepimist tekstiväljale ja salvestus toimus ühe toote kaupa andmeid valideerimata (vt. 48).

Uus lahendus võimaldab tooteid salvestada või üle kirjutada CSV formaadis failist (vt. 49 ja 50). Lahendus võimaldab kasutajale näidata eelvaadet, mille abil on võimalik kasutajal kokku viia toote väljade veerud vaates ja failis. Korrektse faili esimeses reas sisalduvate andmeväljade nimede korral tuvastab vaade veerud automaatselt. Andmete töötluse käigus avastatud vigade korral kuvatakse need kasutajale. Täiendavatest nõuetest tingitud vigade korral tagastab tagaserver vigade nimekirja, mida näidatakse kasutajale veebiliideses.

Lahendus tagab, et Sterling teenuse salvestamisele lähevad ainult korrektsete andmetega tooted. Muudatus võimaldab salvestada maksimaalselt 5000 toodet, teostades päringud 1000 toote kaupa ning sellega oluliselt vähendada võrguliikluse koormust. Samal ajal

võimaldab lahendus peale vigade tuvastamist hoida kokku andmete parandamiseks kuluvat aega, näidates kasutajale vigase veeru ja rea asukoha.

Autori hinnangul osutus keeruliseks etapiks valideerimise sidumine eelvaates olevate veergude vahetumisega. Keerukus seisnes erinevate väljatüüpide ning väljade omaduste rohkuses ning arvestada tuli erinevate failis olevate väärtuste kombinatsioonidega.

Väljakutset pakkus tagaserveri täiendava valideerimise arendamine, sest Sterling teenuse salvestamise korral tuli arvestada paljude lisanduvate nõuetega. Samuti esines raskusi seoses Angular raamistikuga. Mõnes olukorras ei saanud veebiliides aru, et tarvis on komponente või nende andmeid värskendada. Juhul, kui ametlik raamistiku dokumentatsioon probleemi ei lahendanud, siis tuli abi otsida Angular teemalistest aruteludest. Näiteks Stack Overflow foorumid (https://www.stackoverflow.com).

4.3.5 Ettevõtte sisese kasutaja autentimine

Varasema veebirakenduse sisselogimine on mõeldud ettevõtte välistele klientidele. Ettevõtte protseduuriline korraldus näeb ette, et rakenduse kasutamiseks tuleb läbida mitu etappi. Selle korral sõlmitakse leping, sisestatakse konto väliste kasutajate Active Directory (edaspidi AD) andmebaasi ja luuakse täiendav konto Sterling rakendusse. Sterling rakenduses tuleb seadistada vastavalt sõlmitud lepingule ettevõtte kasutatavad klassifikaatorid ja täiendavad lisateenused (näiteks logistikafirma valik või pakiautomaadi kasutamine).

Arenduse etapis selgub, et ka ettevõtte sisesed kasutajad vajavad ligipääsu käesolevale rakendusele. Ettevõtte sisesed kasutajad on tehnilise toe töötajad, kelle eesmärgiks on aidata tehnilise toe poole pöördunud kasutajat.

Ülesandeks on teostada arendus, mis võimaldab ettevõtte sisesest kasutajat autentida ilma täiendava konto loomiseta väliste kasutajate AD-s. Kasutaja autentimine peab toimuma kindla sisemise AD grupi olemasolul, mis on lisatud ettevõtte tehnilise toe töötajatele. Töötaja ettevõttest lahkumisel suletakse tema AD konto ning hiljem pole võimalik selle kontoga käesolevasse rakendusse sisse logida.

Ülesande lähtetingimus on kasutada uut ettevõtte teenust, mida nimetatakse OneAccount-ks. OneAccount võimaldab siseste kasutajate autentimist kas SAML või OAuth

tehnoloogiate abil. OneAccount omab ka SSO (*Single Sign-On*) funktsionaalsust, kus kasutaja välja logimise korral piiratakse kõigi teiste rakenduste ligipääs, mis kasutab OneAccount autentimist.

Ülesannet lahendama asudes on otstarbekas eelnevalt välja selgitada, millist autentimistehnoloogiat on enim kasutatud ettevõtte teistes Spring Boot rakendustes. Esmalt võimaldab see hoida ühist stiili Spring Boot rakenduste arendamisel ja teiseks võib mõni spetsiifiline autentimisega seotud probleem olla juba varem lahendatud. Uurimise käigus selgub, et OneAccount teenus on äsja käivitunud ning ettevõttes puuduvad analoogilised Spring Boot tehnoloogial põhinevad lahendused.

Peale SAML ja OAuth tehnoloogiate võrdlust otsustab autor lahenduse teostada OAuth tehnoloogiat kasutades, sest eduka sisse logimise korral saab kergelt OAuth JWT edastada läbi tagaserveri Sterling teenusele. SAML tehnoloogia korral toimub peale kasutaja sisselogimist kogu edasine andmevahetus genereeritud lugemiskaitsega sessiooniküpsise abil, mida veebiliides lugeda ei saa.

Arenduse käigus tuleb kokku leppida ettevõtte Sterling meeskonnaga loodava Sterling teenuse ja edastatavate andmete sisus. Samuti tuleb teavitada planeeritavatest töödest ja hoida ülesande seisuga kursis OneAccount meeskonda, kuna tegemist on turvalisuse seisukohalt olulise muudatusega.

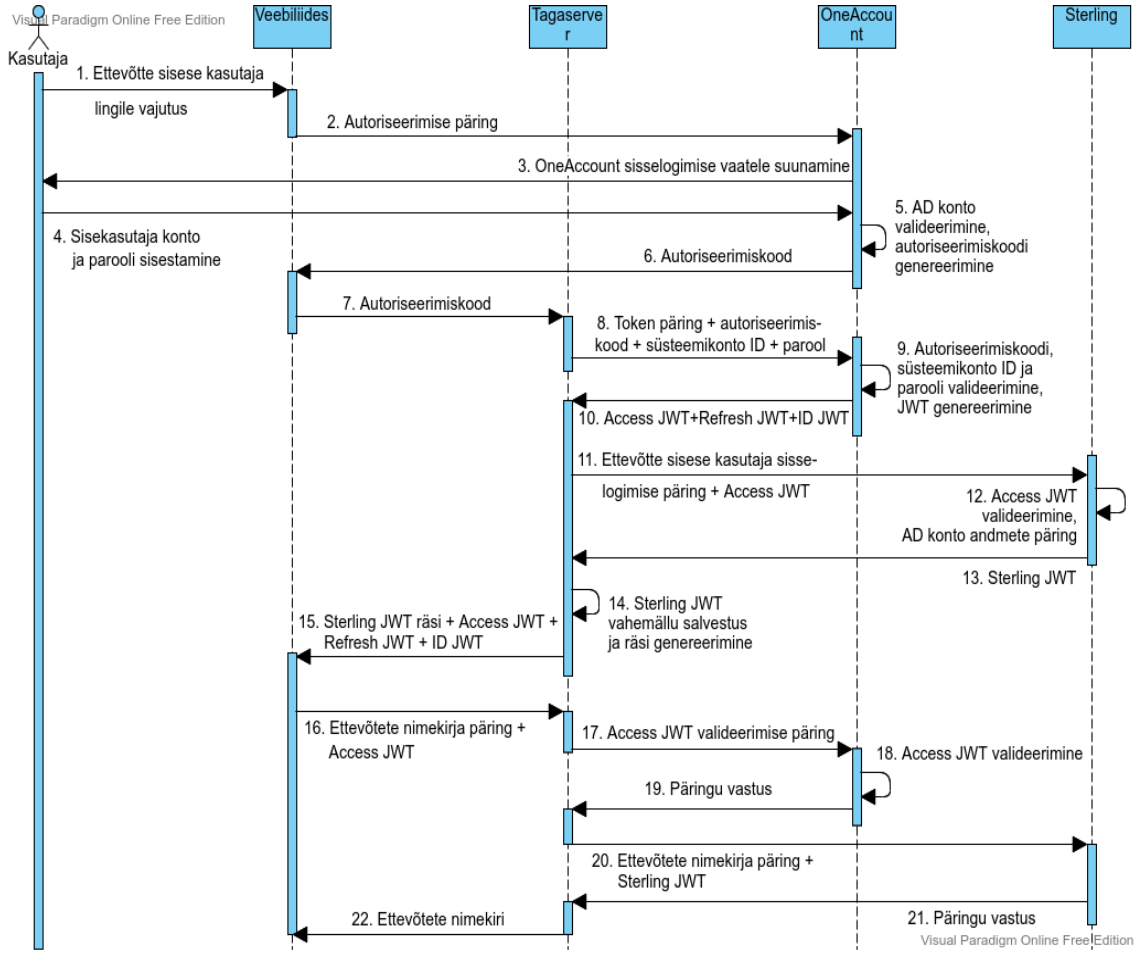
Autoril tuleb sisse viia muudatused nii veebiliideses, kui ka tagaserveris. Veebiliidese korral tuleb lisada viide ettevõtte sisese kasutaja sisselogimiseks, mille korral käivitub automaatne suunamine OneAccount sisselogimise vaatele.

Eduka tuvastuse korral tuleb edastada kasutaja andmed läbi tagaserveri Sterling teenusele. Sterling meeskonna poolt eraldi arendatav teenus peab tagastama tagaserverile Sterling JWT, mis omakorda võimaldab kasutaja veebiliidesesse sisse logida. Sisse loginud tehnilise toe kasutaja tuleb suunata esilehele, kus kasutaja saab valida esindatavat ettevõtet kõikide aktiivsete ettevõtete hulgast.

Sisene kasutaja peab saama veebiliidese vaadetes navigeerida, otsinguid teostada ja salvestatud andmeid vaadata. Samas ei tohi sisene kasutaja saada salvestatud andmeid muuta ega uusi andmeid lisada.

Sterling päringuid tehes tuleb arvestada, et sisemine kasutaja võib olla juba välja loginud mõnest teisest OneAccount autentimist kasutavas rakenduses. Seetõttu tuleb tagaserveris igale päringule eelnevalt teha JWT valideerimine OneAccount poolel ning sõltuvalt tulemusest kas jätkata Sterling päringu teostamisega või logida kasutaja veebiliidesest välja. Samuti tuleb tagaserveris värskendada automaatselt JWT, kui selle kehtivuse aeg on ületatud.

Käesoleva lahendus võimaldab ettevõtte sisestele tehnilise toe töötajatele rakendusele ligipääsu, kasutades olemasolevat platvormi OneAccount ja OAuth tehnoloogiat (vt. Joonis 1).



Joonis 1. Ettevõtte sisese kasutaja sisselogimise jadaskeem.

Arendustööde tulemusena on võimalik ettevõtte tehnilise toe töötajatel osutada paremat teenindust kliendile, kes vajab rakenduse kasutamisel abi. Siseste kasutajate lisamiseks ei ole vaja teostada täiendavat kontode loomise protsessi väliste kasutajate AD-s. Rakendusele ligipääsuks on vaja töötaja kontole siseste kasutajate AD-s lisada vastav

grupp. Samuti töötaja lahkumise korral piisab ainult siseste kasutajate AD-s konto sulgemisest ning sellega on piiratud töötajale hilisem rakendusele ligipääs.

Ülesande lahendamisel osutus probleemseks aeglane info liikumine rahvusvaheliste meeskondade vahel. Võis juhtuda, et möödus 1-2 päeva, enne kui OneAccount või Sterling meeskonnale saadetud kirjas esitatud küsimustele vastused saadi. Selle põhjustas inimeste hõivatus erinevate ülesannete ja koosolekute rohkuse tõttu.

Autor leiab, et suures ettevõttes tuleb aeglase info liikumisega arvestada ning kiirema informatsiooni liikumise tagab korralikult läbimõeldud küsimustega kirjad või varakult planeeritud koosolekud teiste meeskondadega.

4.4 Arenduse käigus teostatud muudatused ja parendused

Arenduse käigus on võimalik, et tööde maht suureneb. Selleks võib olla mitmeid põhjuseid. Näiteks hakkab arendustööde käigus klient paremini mõistma, mida ta täpselt soovib. Kliendi soovid võivad kas muutuda või võib lisanduda täiendavaid soove. Samuti võivad teostatud tööd sisaldada arendusvigu, mis vajavad parandamist.

Arenduse käigus võib tekkida vajadus rakenduse parendamiseks. Näiteks on vajalik koodi üldistada, mis võimaldab paremini komponente korduvalt kasutada. Veebiliidese korral võib tekkida vajadus tõsta selle kasutusmugavust või -kogemust.

Kuna käesoleva projekti korral on võetud eesmärgiks arenduse käigus testkasutajate tagasisidega arvestamine, siis koostati esialgse tööde mahuga võrreldes samas suurusjärgus täiendavaid arenduspileteid.

Testkasutajate muudatuste soovid on peamiselt väikesemahulised, näiteks välja lisamine, asukoha muutmine või mittevajaliku eemaldamine. Leidub ka kasutusmugavuse parendamist vajavaid lisatöid, näiteks detailvaatest tagasi otsingu vaatele minnes sooviti varem teostatud otsingutulemust uuesti näha.

Samuti on arendusprotsessi käigus vaja teha suuremahulisi arendustöid. Näiteks klientide sisestamist võimaldav vaade ning klientide andmetel põhinevad sisseostu- ja väljamüügiarvete vaadete põhjalik muutmine. Samuti ettevõtte müügiraporti vaate

loomine, mis peab sisaldama Microsoft Power BI 41 platvormi poolt genereeritud müügiraportite näitamist.

Arendusmeeskonna poolt koostatud täiendavad piletid olid seotud suures osas kasutusmugavuse parendamisega. Samuti vahetati visuaalseid komponente väliste teekide vidinate või komponentide vastu. Tooteomaniku poolsed soovid olid peamiselt seoses kasutusmugavuse täiendustega. Üks tooteomaniku poolt lisandunud suuremahuline arendustöö oli rakenduse ligipääsu tagamine ettevõtte sisesele tehnilise toe töötajale (vt. alampeatükk 32).

Kokku vormistati täiendavaid arenduspileteid kokku 138, millest umbes 75% moodustasid testkasutajate muudatused ning 25% moodustasid arendusmeeskonna ja tooteomaniku poolsed muudatused ning parendused.

4.5 Tulemus

Käsitletud arendustööde tulemusena valmis 2020. aasta lõpus klient-server arhitektuuril põhinev rakenduse MVP.

Lõputöö autor leiab, et rakendus sisaldab kõiki algselt planeeritud funktsionaalsusi, täiendavaid muudatusi ning parendustöid. Täidetud on kõik ettevõtte poolsed lähtetingimused ja kitsendused.

Tehtud tööde käigus lahendati kokku 261 (100%) arenduspiletit, millest algselt oli planeeritud 123 (47%) ning arendustööde teostamise jooksul lisandus 138 (53%) piletit.

Lisaks tagaserveri Spring Boot tehnoloogiale ümber kirjutamisele lahendas meeskonnas olnud vanemarendaja Git statistika järgi 17 (7%) arenduspiletit. 2020. aasta lõpus meeskonnaga liitunud arendaja lahendas kahe kuu jooksul projektis olles 19 (7%) piletit. Tarkvara testija, kes oli projektis kahe kuu jooksul osalise tööajaga, lahendas testide loomisega 2 (1%) piletit. Ülejäänud 223 (85%) pileti lahendused teostati käesoleva lõputöö autori poolt.

Põhilised arendustööd on seotud veebiliidesega. Vaadete osas on lisatud viis vaadet müügiarvete otsingu, müügiarvete detailvaate, klientide otsingu, klientide detailvaate ja Power BI raportite vaatega ning ümber on tehtud toodete üleslaadimise vaade. Samuti on

muudetud ja täiendatud kõiki ülejäänud vaateid. Rakenduse vaadete näidised enne ja pärast arendustöid on antud 44 - 50.

Tagaserver sisaldas peale Spring Boot tehnoloogiale üle viimist 10 päringut veebiliidese ja Sterling keskkonna vahelise infovahetuse tarbeks. Käsitletud tööde jooksul arendas autor täiendavalt tagaserverisse 27 päringut. Kokku võimaldab tagaserver teostada 37 erinevat päringut.

Arenduse käigus kasutati kahe nädalase kestvusega sprinte. Sprintidele eelnenud töömahtude hindamine paranes aja möödudes tunduvalt, sest hiljem suurenesid projektiga seotud teadmised ning oskused, mis võimaldasid paremini ette näha potentsiaalseid probleeme.

Rakendus on paigaldatud toodangukeskkonda 2021. aasta jaanuaris. Käesolevaks ajaks on rakendus olnud kasutuses neli kuud, mille tõttu ei ole tööandja jõudnud läbi viia kasutajate rahulolu uuringut.

Esmase hinnangu saamiseks on lõputöö autor võtnud ühendust rakenduse tooteomanikuga, kelle üheks ülesandeks on suhelda klientidega ning jälgida rakenduse kasutatavust. Rakenduse tooteomanik arvab, et rakenduse veebiliides on vastavalt tagasisidele kasutajate jaoks arusaadav ning piisavalt mugav. Seda tõestab aktiivne arendatud rakenduse funktsionaalsuste kasutamine. Tuginedes aktiivsele kasutamisele ja klientide esmasele tagasisidele, hindab tooteomanik, et ülesanne lahendati vastavalt püstitatud eesmärgile.

4.6 Rakenduse laiendatavus

Arendustööde tulemusena valminud rakendus on loodud selle edasist arengut silmas pidades. Esiteks on rakendus laialdaselt tuntud klient-server arhitektuuriga, mis on vajaduse korral kergelt aru saadav uuele arendajale. Teiseks on rakenduse tagaserveri ja veebiliidese tehnoloogiad käesoleva lõputöö koostamise ajal enim kasutatavate seas. Kolmandaks on arendamisel arvestatud koodi korduvkasutusega, loetavusega, ühtse stiiliga, testide olemasoluga, kasutusmugavusega ja tarkvara arenduse parimate praktikatega 4141.

Lisaks tehnilisele poolele on võimalik rakendust laiendada täiendavate funktsionaalsustega, sest kasutada on peaaegu 1300 erinevat Sterling API päringumalli. Samuti on võimalik kasutada ettevõte arendanud funktsionaalsuseid laiendavaid teenuseid, mida saab kasutada samas formaadis nagu Sterling API päringuid.

Rakenduse arendusperioodi jooksul osalesid testkasutajad mitmest välisest ettevõttest. Kasutajaid oli kokku umbes kümme, kes olid nõus tagasisidet andma rakenduse paremaks muutmise eesmärgil. Kasutusmugavuse ja kasutuskogemuse osas arvestati kõikide testkasutajate kommentaaridega ning lisanduvate arenduste korral tuli eelnevalt hinnata ajalist mahtu ja võimaluse korral töösse võtta. Kuigi kõiki lisatööde soove ei olnud võimalik teostada ajalise mahu tõttu, koostati nendest täiendavad arenduspiletid järgnevate arendusetappide tarbeks.

On võimalik, et ettevõtted soovivad kasutada erinevaid funktsionaalsusi, mida eelpool nimetatud süsteem võimaldab. Autori hinnangul on rakendust võimalik veelgi laiendada, kui võtta kasutusele moodulite süsteem. Sellisel juhul tuleb rakendust seadistada nii, et erinevaid tegevusi võimaldavad vaated on ühe kindla ettevõtte jaoks kas peidetud või avatud. Kuna tööandja rendib rakendust pilvepõhise teenusena, siis moodulite valik võib soodustada erinevate vajadustega klientide lisandumist.

5 Kokkuvõte

Lõputöö eesmärk on arendada rahvusvahelise kasutajaskonna poolt kasutatav laoressursside halduseks klient-server arhitektuuril põhinev rakenduse MVP, kus lähtetingimusena tuleb kasutada autori tööandja omanduses olevat laohaldustarkvara Sterling ja sellega integreeritud rakenduse lihtsat prototüüpi.

Käsitletud arendustööde tulemusena on valminud klient-server arhitektuuril põhinev rakenduse MVP. Selle arendamiseks kasutati peamiselt Scrum arendusmetoodikat, lõimides seda teiste tarkvara arendusemetoodikate parimate praktikatega. Täidetud on kõik tööandja poolsed lähtetingimused ja kitsendused. Lõputöö autor leiab, et rakendus sisaldab kõiki algselt planeeritud funktsionaalsusi, täiendavaid muudatusi ja parendustöid.

Põhilised arendustööd on seotud veebiliidesega. Vaadete osas on lisatud viis vaadet müügiarvete otsingu, müügiarvete detailvaate, klientide otsingu, klientide detailvaate ja Power BI raportite vaatega ning ümber on tehtud toodete üleslaadimise vaade. Muudetud ja täiendatud on ka kõiki ülejäänud vaateid. Rakenduse vaadete näidised enne ja pärast arendustöid on antud 44 - 50.

Tagaserver on ümber kirjutatud Spring Boot tehnoloogiale. Lisaks sisaldab tagaserver eelnevalt kasutusel olnud 10 päringu asemel täiendavalt 27 päringut veebiliidese ja Sterling keskkonna vahelise infovahetuse tarbeks. Kokku võimaldab tagaserver teostada 37 erinevat päringut.

Autor osaleb rakenduse arendusprotsessis ainsa arendajana üle 1.5 aasta, mille käigus lahendab arenduspiletite kogumahust 261 (100%) 223 (85%) piletit.

Veidi üle kahe aasta arendatud rakendus lahendab e-kaubandusega tegelevate ettevõtete vajadused laoressursside haldamisel, sisaldades kasutajate jaoks ainult soovitud funktsionaalsust. Arendustööd on teostatud koodi lihtsust, komponentide korduvat kasutust, rakenduse laiendatavust ja tarkvara arenduse parimaid praktikaid arvestades.

Rakendus on paigaldatud toodangukeskkonda 2021. aasta jaanuaris. Kuna käesolevaks ajaks on rakendus olnud kasutuses neli kuud ja selle tõttu ei ole tööandja poolt läbi viidud

kasutajate rahulolu uuringut, siis saab autor projekti edukuse hindamisel lähtuda rakenduse tooteomaniku esmasest hinnangust.

Rakenduse tooteomanik arvab, et rakenduse veebiliides on vastavalt tagasisidele kasutajate jaoks arusaadav ning piisavalt mugav. Seda tõestab aktiivne arendatud rakenduse funktsionaalsuste kasutamine. Tuginedes aktiivsele kasutamisele ja klientide esmasele tagasisidele, hindab tooteomanik, et ülesanne lahendati vastavalt püstitatud eesmärgile.

Kasutatud kirjandus

[1] E-kaubanduse käive 2020. https://e-kaubanduseliit.ee/e-kaubanduse-kaive-2020/ (10.05.2021)

[2] Poodide sulgemine tõi kaasa internetiostjate osakaalu hüppelise kasvu nii Eestis kui kogu Euroopas. https://e-kaubanduseliit.ee/poodide-sulgemine-toi-kaasa-internetiostjate-osakaalu-huppelise-kasvu/ (10.05.2021)

[3] IBM Sterling Warehouse Management System Overview. https://www.ibm.com/support/knowledgecenter/en/SS73R8_9.4.0/com.ibm.help.wms.concepts.doc/c_SterlingWarehouseManagementSystemOverview.html (01.03.2021)

[4] Benefits of IBM Sterling Warehouse Management System. https://www.ibm.com/support/knowledgecenter/en/SS73R8_9.4.0/com.ibm.help.wms.concepts.doc/c_BenefitsOfSterlingWarehouseManagementSystem.html (02.03.2021)

[5] Kliendi mõiste erinevad lahendused. https://eopearhiiv.edu.ee/e-kursused/eucip/juhtimine/311_kliendi_miste_erinevad_thendused.html (02.03.2021)

[6] RxJS Introduction. https://rxjs-dev.firebaseapp.com/guide/overview (10.03.2021)

[7] Popular Software Development Methodologies and Frameworks: Full Comparison. https://www.cleveroad.com/blog/software-development-methodologies (11.03.2021)

[8] Project success rates – agile vs waterfall. https://vitalitychicago.com/blog/agile-projects-are-more-successful-traditional-projects/ (11.03.2021)

[9] LHV IT-arenduse juht: miks mulle ei meeldi Scrum. https://www.ituudised.ee/uudised/2020/04/09/lhv-it-arenduse-juht-miks-mulle-ei-meeldi-scrum (30.03.2021)

[10] JSR 380: Bean Validation 2.0. https://jcp.org/en/jsr/detail?id=380 (08.04.2021)

[11] What is Power BI. https://powerbi.microsoft.com/en-us/what-is-power-bi/ (21.04.2021)

[12] Gamma E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. 1st ed. Addison-Wesley Professional, 1994

[13] Martin, Robert C. Clean Code: A Handbook of Agile Software Craftmanship. 1st ed. Pearson, 2008

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

Mina, Aivar Mägi

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Laoressursside halduseks klient-server arhitektuuriga rakenduse arendamine", mille juhendajad on Meelis Nõmm ja Kristiina Hakk

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2 Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3 Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

17.05.2021

[1]Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 - Veapileti kirjelduse ja kommentaari näide

Description

Creating PO: When order is saved without order number, then two error messages are displayed: „Order number is required field" and „Customer PO number is required field".

Creating SO: Error messages are appearing twice as previously described. When error messages are displayed after saving and user fixes some error, then errors should be updated, but aren't.

Comment

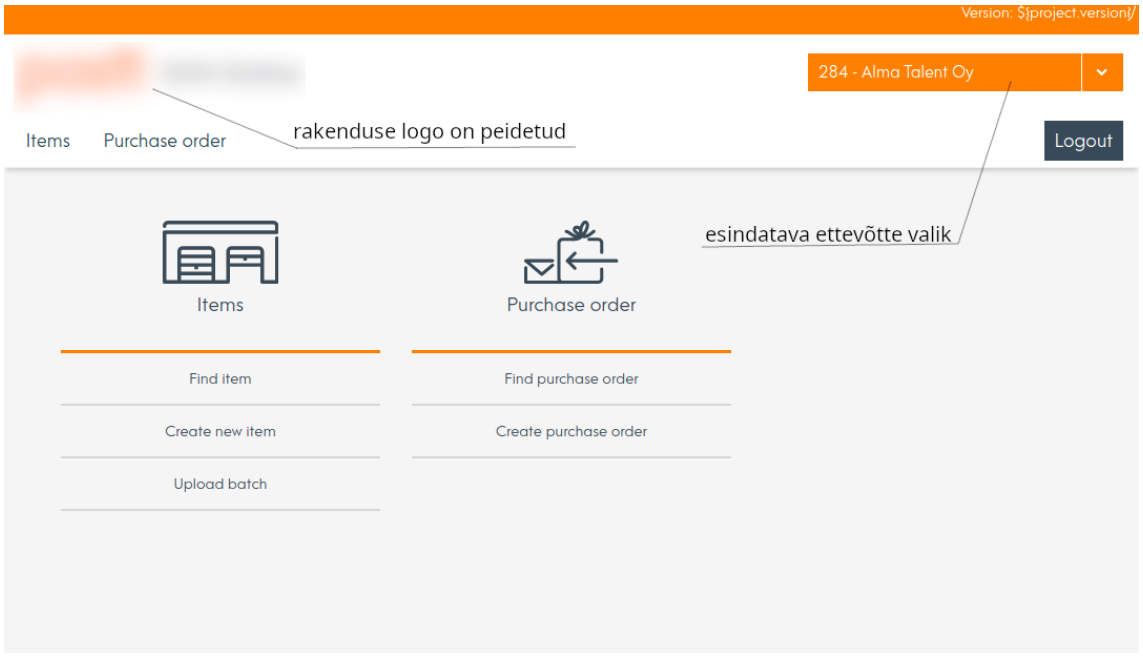
Both fields were validated in back-end and both of them were required fields, although CustomerPoNo value was copied from OrderNo field. So when user didn't enter order number, back-end gave errors from both fields.

Nullable check of CustomerPoNo field in back-end is now removed, because it depends of OrderNo value, and without OrderNo value the order save query to Sterling won't be triggered. Similar logic was added to Item creation page. Also templates of PO, SO, Items creation were changed to remove errors from UI when user corrects them after unsuccessful saving.

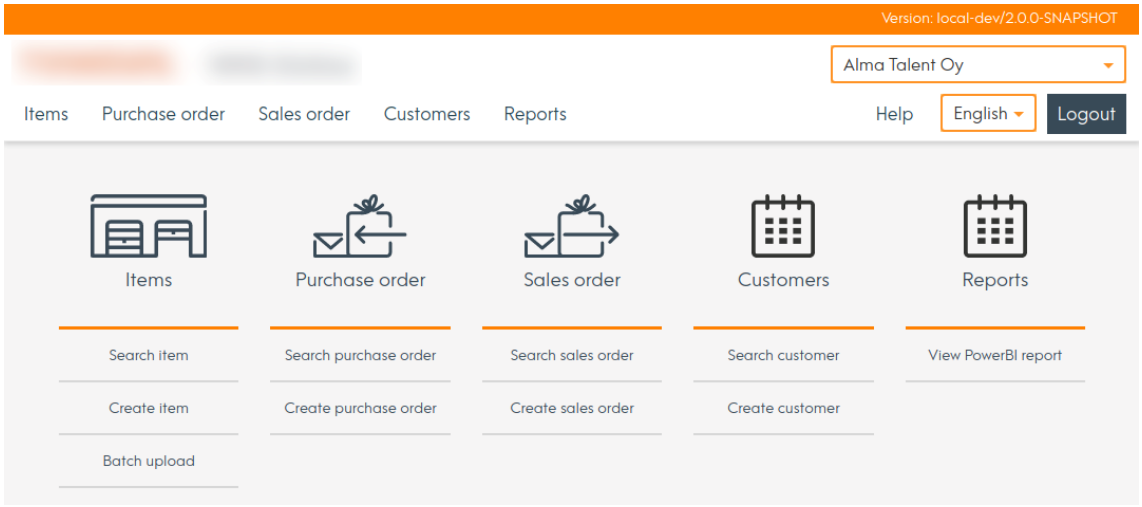
Now when user tries to save order without order number, only one error about missing order number is displayed. When user corrects values after unsuccessful change, specific error is removed.

Overall good practice is to display user errors in UI right after they are made and block form saving when it already has incorrect data values. So proposal is to always disable saving buttons in case of invalid field values.

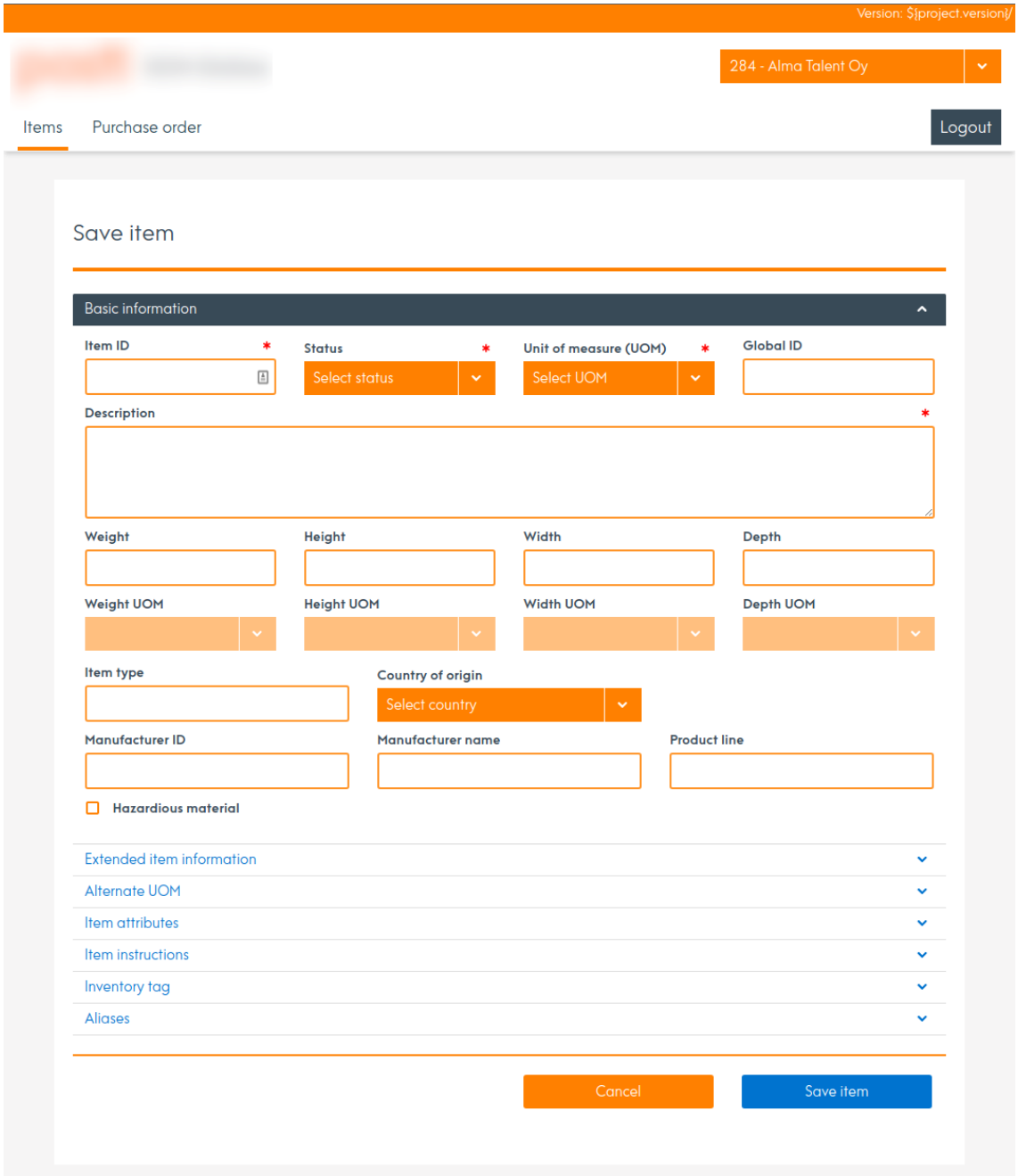
Lisa 3 - Rakenduse prototüüp - esileht



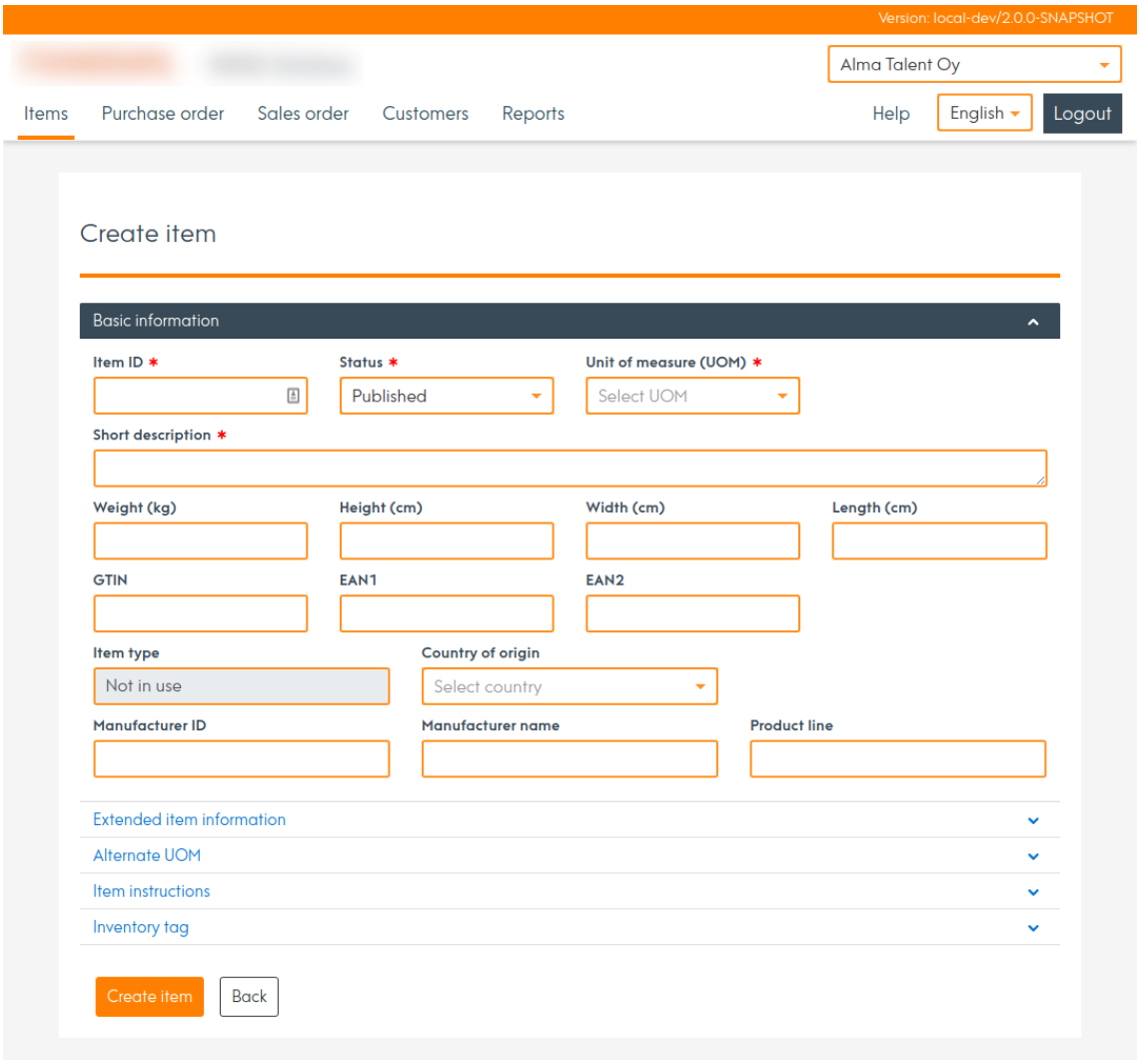
Lisa 4 - Rakenduse lõppversioon - esileht



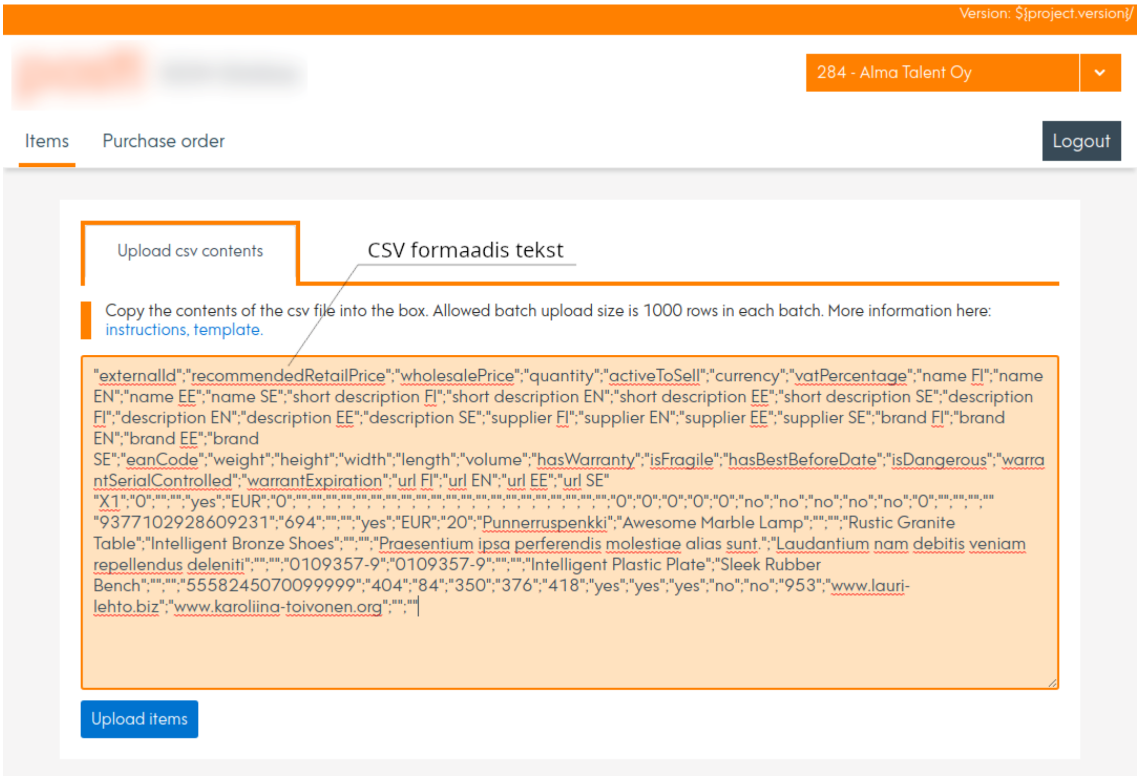
Lisa 5 - Rakenduse prototüüp - toote lisamise vaade



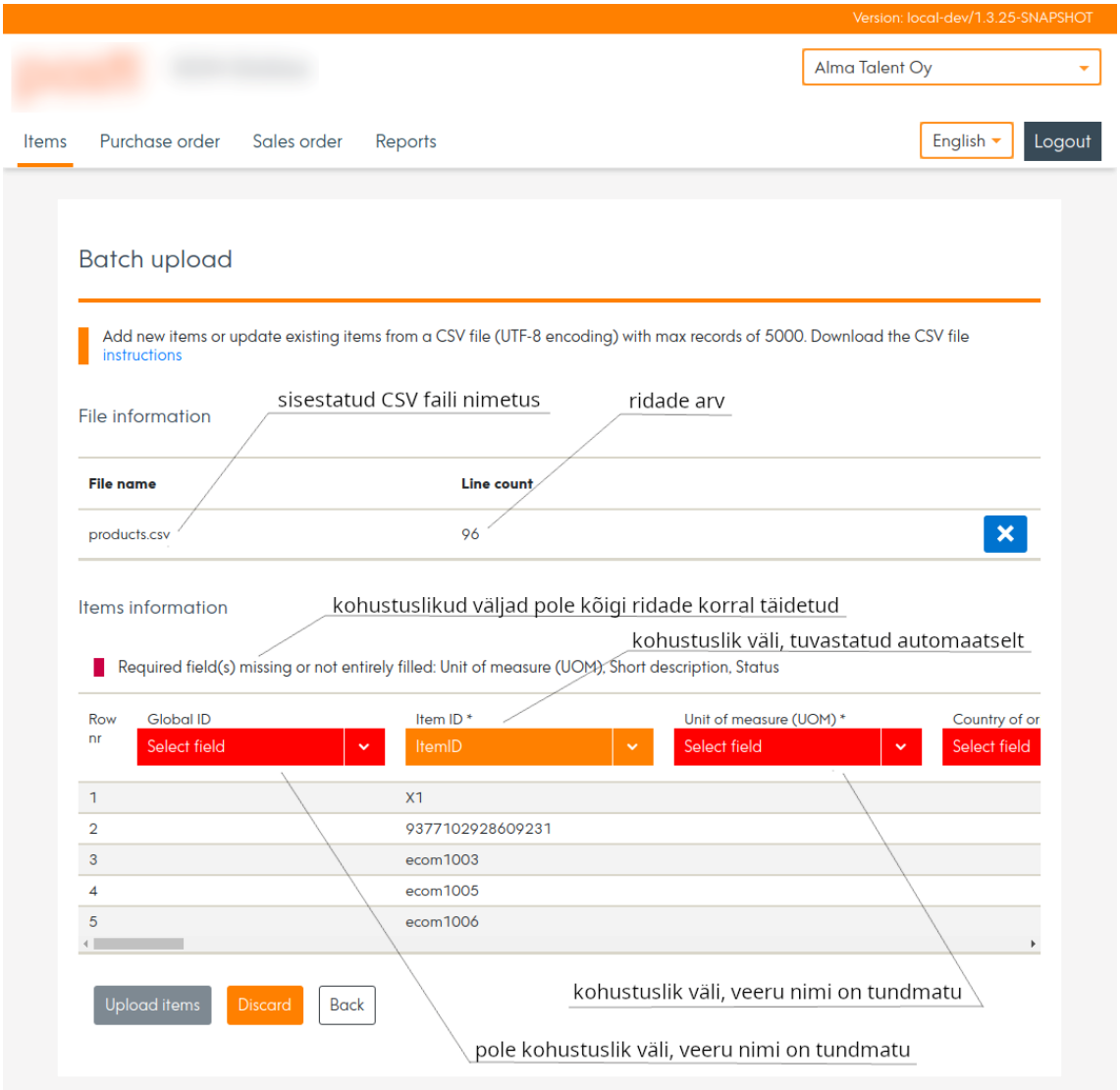
Lisa 6 - Rakenduse lõppversioon - toote lisamise vaade



Lisa 7 - Rakenduse prototüüp - toodete üleslaadimise vaade



Lisa 8 - Rakenduse arenduses olev versioon - toodete üleslaadimise vaade koos olemasolevate komponentidega



Lisa 9 - Rakenduse lõppversioon - toodete üleslaadimise vaade