TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Ranno Rajaste 185564IADB

Äritarkvara Directo dokumendisüsteemi veebipõhise eesrakenduse loomine

bakalaureusetöö

Juhendaja: Meelis Antoi
Magister

Tallinn 2022

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: Ranno Rajaste

07.01.2022

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on luua veebipõhine eesrakendus äritarkvara Directo tarbeks, mis suudaks kuvada kõiki äritarkvara dokumendivaateid kasutades selleks võimalikult suurel määral ühtset programmikoodi. Loodav rakendus peaks pakkuma kasutajatele pärandsüsteemiga võrreldes paremat kasutuskogemust, vähendades seejuures süsteemi haldamiseks kuluvat ressurssi ning tõstes tarkvara vea- ja tulevikukindlust, turvalisust ja kasutatavust.

Töös käsitletavateks põhiprobleemideks on pärandsüsteemi funktsionaalsuse ja probleemide kaardistamine ning nende põhjal kasutajanõuete kogumine ja analüüsimine, erinevate dokumendivaadete ühisosa leidmine ja selle põhjal lahenduse modelleerimine, eesrakenduse tehnoloogiate valimine ning dokumentatsiooni ja testimise kavade koostamine.

Bakalaureusetöö tulemusena valmis veebipõhine eesrakendus, mis suudab kuvada äritarkvara dokumendivaateid ühtse programmikoodi alusel ning mis on adekvaatselt dokumenteeritud ja testitud ning kasutab kaasaaegse tarkvaraarenduse parimaid praktikaid ja tehnoloogiaid. Valminud ühtse süsteemi baasil loodi kaks dokumendivaadet, mis on kasutusel äritarkvara orgaaniliste osadena. Valminud süsteemi arendus jätkub ning tulevikus tuuakse uuele süsteemile üle kõik äritarkvara dokumendivaated.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 60 leheküljel, 5 peatükki, 1 joonist, 2 tabelit.

Abstract

Creating a Front-End Web Application for Directo Enterprise Resource Planning Software Document System

The aim of the given thesis is to create a front-end web application for Directo enterprise resource planning software which could display all currently in-use document views using common program code as extensively as possible. Compared to the legacy system, the planned solution should offer better user experience while reducing required resource for software maintenance as well as increase the software's overall resistance to fault, future proofness, security and usability.

Given thesis deals mainly with problems related to uncovering functionality and issues related to the legacy system as well as constructing and analyzing user requirements based off of them, finding common logic between different document views and modeling a solution synthesized from that, choosing technological solutions for the front-end application and creating documentation and testing plans.

As a result of this thesis a front-end web application was created which could display all currently in-use document views using uniform program code as well as being adequately documented and tested while employing the best practices and technological solutions of modern software development. Two document views were created using the developed unified system, which were adopted as organic components of Directo business software. Development of the created system will continue and all of the business software's document views will be recreated using the new unified system in the coming future.

The thesis is in Estonian and contains 60 pages of text, 5 chapters, 1 figures, 2 tables.

Lühendite ja mõistete sõnastik

CRUD operatsioonid	Püsiliku andmestu, eeskätt püsiliku andmebaasi põhioperatsioonid: - *Create*: loomine - *Read/Retrieve*: lugemine/võtt - *Update*: värskendus (ajakohastus) - *Delete/Destroy*: kustutus/hävitus
Eesrakendus	Veebilehele ilmuv kasutajaliides, mis võimaldab veebisaidi külastajal kahepoolselt suhelda saidi dünaamiliste osadega nagu andmebaasid ja veebiliidesed.
Kasutajalugu	Kasutaja vajadusi ja käitumist eelduslikult kirjeldav stsenaarium arendustöös ja tootehalduses.
Kasutusmall	Süsteemi käitumisele ja kasutamisinteraktsioonile esitatavate nõuete kirjeldus.
Läbistustestimine	Sissetungirünnete imiteerimine turvameetmete toimivuse kontrollimiseks.
Modaalaken	Algses aknas avanev sekundaaraken, blokeerib algse akna juhtimise ja nõuab kasutaja sekkumist
Olekuhaldussüsteem	Tarkvara osa, mille ülesandeks on rakenduse oleku andmete talletamine, haldamine ja käitlemine. Inglise keeles *state management system*.
Oskuskasutaja	Rohkete rakendusprogrammide ja nende rohkete võimaluste intensiivne kasutaja.
Pärandsüsteem	Kasutuselolev süsteem, mida ei täiustata ega uuendata, sest tulevane on juba saadaval või väljatöötamisel.
SaaS	Tarkvara teenusena pakkumine. Inglise keeles lühend fraasist *software as a service.* Tarkvara tarnimise mudel, mille kohaselt saab klient tasu eest limiteeritud ajaks ligipääsu teenuseandja pakutavatele rakendustele.
Tagarakendus	Lõppkasutajale nähtamatu veebirakenduse osa, mille ülesandeks on andmeid töödelda, talletada ning käidelda.
TDD	Lühend inglisekeelsest fraasist *test driven development*. Tarkvaraarenduse metoodika, mille põhimõte näeb ette tarkvara nõuete põhjal testide kava koostamist enne vastavate omaduste tarbeks programmikoodi valmis kirjutamist.
Tööjärg	Tootenõuete ja lõpetamisele kuuluvate saaduste prioriteediloend, järgmisel iteratsioonil väljatöötamisele kuuluvate tarkvara

	erijoonte loetelu
Üheleherakendus	Veebirakendus, mis suhtlusel kasutajaga kirjutab käesoleva veebilehe dünaamiliselt üle, selle asemel et serverist tervet uut lehte alla laadida.
XSS	Koodisüsti meetod, mis kasutab ära veebisaitide nõrkusi (näiteks sisestusvälja turvakontrolli puudumist) ning ründab otse või teiste veebisaitide kaudu viies kasutaja brauserisse HTML-, JavaSript-vm kahjurkoodi või kahjulikke veebilinke.
Sessiooni küpsis	Väike tekstisõne, mille kirjutab veebiserver kliendi brauserisse ja mille esitab brauser igal järgmisel pöördumisel kasutaja autentimiseks ja järgmiste pöördumiste hõlbustamiseks
REST	Lühend inglisekeelsest fraasist r*epresentational state transfer.* Veebiteenuste programmeerimisel andmete hajustöötluse paradigma, mis taotleb veebi nüüdisnõuetele sobivat arhitektuuristiili ja ühtset liidest.
Moodultestimine	Üksikprogrammide või -moodulite testimine veendumiseks, et neis pole analüüsi- või programmeerimisvigu.
Integratsioonitestimine	Tarkvara testimise viis, mis testib korraga mitme erineva tarkvaraosise koos töötamist.
JSON	Lühend inglisekeelsest fraasis *JavaScript object notation.* Lihtne andmevahetusvorming, mis põhineb JavaScripti alamhulgal ning on hõlbus inimlugemiseks ja -kirjutuseks.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Antud bakalaureusetöö loomise hetkel töötas autor ettevõttes Directo OÜ, mis arendab ja pakub tasulise teenusena oma klientidele samanimelise äritarkvara kasutamise võimalust vastavalt *SaaS* mudelile. Directo äritarkvara on täielikult pilvepõhine ning seda kasutatakse läbi veebilehitseja.

Töö kirjutamise hetkel olid ettevõttes kasutusel olevad tehnoloogiad iganenud ning ei võimaldanud arhitektuuriliste ja tehnoloogiliste puudujääkide tõttu enam kasutajatele pakkuda soovitud funktsionaalsust, lisaks olid tugevalt raskendatud varasema süsteemi ülalhoidmine, edasi arendamine ja haldamine ning süsteemis esines märkimisväärseid turvariske. Sellest tulenevalt otsustas ettevõte rakenduse ümber kirjutada kasutades modernsemaid tehnoloogiaid ning üheleherakenduse arhitektuuri. Ümbertegemist otsustas ettevõtte juhtkond alustada dokumendivaadetest, mille tarbeks ühtse ja paindliku eesrakenduse loomise analüüsi ja arenduskäiku antud bakalaureusetöös käsitletakse.

Bakalaureusetöö eesmärk oli analüüsida ja välja töötada eesrakendus, mis võimaldaks lõppkasutajatele serveerida kõiki töö kirjutamise hetkel tarkvaras kasutusel olnud dokumendivaateid ning mis toetaks tulevikus ka samal põhimõttel uute dokumendivaadete arendust ja loomist. Töö olulisteks eesmärkideks on lisaks eelpool mainitule veel dokumendivaadete dokumentatsiooni parendamine, turvanõrkuste leevendamine, automaattestide kaetavuse suurendamine ja koodi korduvuse vähendamine ning läbi selle süsteemi hallatavuse tõstmine.

Töös käsitletav probleem ja selle lahendamine on ennekõike vajalik pakkumaks klientidele mugavamat ja ohutumat kasutajakogemust, mis omakorda tõstaks ettevõtte pakutava teenuse konkurentsivõimet turul.

Bakalaureusetöö on jaotatud neljaks suuremaks sisupeatükiks. Esimeses neist on kirjeldatud sügavamalt lahendatavat probleemi ennast ning selle tausta, olemust, lähtetingimusi ning oodatavat eesmärki. Teises peatükis on keskendutud probleemi

analüüsile, mis sisaldab olemasoleva süsteemi funktsionaalsuse, kasutatavuse probleemide ja turvanõrkuste kaardistamist, vaadete ühisosa leidmist ning lisafunktsionaalsuse sidumist olemasolevaga. Kolmandas peatükis on käsitletud lahenduse loomist, sh nõudeid lahendusele ning tehnoloogiate ja strateegiate valikut, automaattestimise ja dokumenteerimise kava välja töötamist, süsteemi üldist struktuuri ja ülesehitust ning lahenduse realiseerimise faasi. Neljandas ja viimases peatükis tuuakse välja töö tulemused koos projekti tulevikuvaadete ja retrospektiiviga.

2 Probleemi püsitus ja projekti eesmärk

Alljärgnevalt kirjeldatakse lähemalt diplomitöös käsitletavat probleemi ning selle püstitust koos probleemi tausta ja lähtetingimustega, sõnastatakse projekti ja diplomitöö eesmärgid, pannakse paika projekti skoop ning antakse ülevaade planeeritavast metoodikast.

2.1 Taust

Töö loomise hetkel töötas autor ettevõttes Directo OÜ, mis pakub oma klientidele samanimelist äritarkvara. Tarkvara oli loodud kasutades peamiselt staatiliste serveri pool genereeritud veebilehekülgede arhitektuuri (inglise keeles *server side rendering*). Antud arhitektuur limiteerib märkimisväärselt kasutuskogemust olukordades, kus kasutaja muudab tihti leheküljel olevaid andmeid ning kus muudatuste peale peab lehekülg koheselt uuenema ning värsket infot kuvama, kuna iga muudatuse peale tuleb terve lehekülg uuesti laadida [1] . Äritarkvarade juures on taoline kasutusviis aga väga levinud, kuna suur osa praegustest vaadetest põhineb CRUD operatsioonidel.

Ettevõte oli eelpool mainitust tingituna alustanud äritarkvara moderniseerimist ja ümber kirjutamist kasutades varasema arhitektuuri asemel modernsemat ühelehe veebirakenduse arhitektuuri, kus on üksteisest eraldatud andmebaas, tagarakendus ja eesrakendus. Enne diplomitöö loomist oli algust tehtud mõne üksiku prototüüpse vaate loomisega kasutades uut arhitektuuri. Kuna prototüübid olid loodud ilma laiema analüüsi ja kavandamiseta, polnud need eriti edukad.

Diplomitöö autor asus tööle uue süsteemi eesrakendust arendavas meeskonnas ning tema eesmärgiks määrati töötada välja lahendus, mis suudaks kuvada kasutajale äritarkvara erinevaid dokumendivaateid.

2.2 Probleemi püstitus

Äritarkvara jaguneb üldises plaanis kaheks suureks komponendiks: dokumendid ja nendel kajastuva info põhjal genereeritud aruanded. Dokumente on väga mitut erinevat liiki ning nendes sisalduv info võib sõltuvalt dokumendivaatest olla radikaalselt erinev. Siiski on dokumendivaadetel väga suur ühisosa ning suures pildis on nad enam vähem sarnase struktuuriga, mistõttu oleks otstarbekas kasutada erinevate dokumendivaadete näitamiseks ühtset süsteemi. Pärandrakenduses vaadete tehnoloogiline ühisosa sisuliselt puudub ning sellest tingituna on koodi korduvus ja veaohtlikus kõrge ning hallatavus raskendatud.

Kuna äritarkvara kasutavaid ettevõtteid on palju ning erinevate ettevõtete jaoks on nende ärimudelist ja ettevõtte spetsiifikast tulenevalt vajadus dokumendivaadetele erinev, on tähtis, et dokumendivaated oleksid lõppkasutaja jaoks võimalikult paindlikud ja kergesti kohandatavad, kuna üks fikseeritud kujundus ja lahendus ei rahulda kindlasti kõikide ettevõtete vajadusi ega paku parimat võimalikku kasutajakogemust. Lisaks võivad dokumendivaadete mõned osised olla erinevate kasutajate jaoks nende õigustest ja häälestusest sõltuvalt peidetud. Pärandrakenduses on antud probleem lahendatud suuresti kasutades kliendi jaoks tellimustööna valminud *JavaScript* faile, mis muudavad vastavalt kliendi soovidele dokumendivaateid ning arvutavad ringi erinevate väljade väärtusi kasutades vastavalt kliendi vajadustele loodud kohandatud äriloogikat. Antud lahendus ei ole aga optimaalne, kuna hõlmab pikka ja kulukat tellimustööd ning on arhitektuurilisel tasemel kehvasti realiseeritud ja sellest tulenevalt veaohtlik ja kohati ebaturvaline.

Lisaks pakub äritarkvara kasutajatele võimalust liidestada tarkvaraga erinevaid väliseid süsteeme, millest lähtuvalt on oluline, et võimalikult suur osa rakenduse äriloogikast paikneks tagarakenduses ja eesrakendus võimalusel ise äriloogikat ja sellest tulenevaid arvutusi ei teostaks, kuna antud lähenemise juures oleks võimalik ühe äriloogika alusel serveerida nii eesrakendust kui ka muid liidestatud programme ja süsteeme. Pärandsüsteemis paikneb äriloogika tasandiliselt erinevalt, tihti leiab seda ka eespool mainitud tellimustööna valminud *JavaScript* failidest, mis aga toob endaga kaasa äriloogika korduvuse, kuna liidestel pole ligipääsu pärandsüsteemis kasutaja veebilehitsejas asuvates *JavaScript* failides peituvale äriloogikale.

Pärandsüsteem loodi 2000. aastate alguses ning kuni tänaseni on selle arhitektuur ja kasutamise mustrid olnud suhteliselt muutumatud. Peaaegu kahekümne aastaga on tehnoloogiad ja internet ise aga äratundmatuseni muutunud ning koos sellega on muutunud ka kasutajate nõuded ja soovitud töövood. Kuna pärandsüsteem ei võimalda eelpool mainitud tehnoloogilistest piirangutest tingituna uusi kasutajanõudeid täita, on areng antud aspektis olnud limiteeritud ning kasutajad on pidanud olude sunnil hakkama saama kohati mitteoptimaalse kasutuskogemusega. Kuna uus süsteem põhineb ühelehe veebirakenduse arhitektuuril ning võimaldab seega oluliselt suuremat paindlikkust kasutajaliideses, on uue süsteemi juures oluline välja selgitada kasutatavuse kitsaskohad ning koguda kokku uuenenud kasutajanõuded.

Eelpool mainitud asjaoludest lähtuvalt on diplomitöös käsitletav põhiprobleem järgmine: kuidas luua veebipõhine eesrakendus, mis suudaks kuvada kõiki äritarkvara dokumendivaateid kasutades selleks võimalikult suures ulatuses sama koodi ja samu komponente, võimaldades seejuures kasutajal võimalikult kergelt ja paindlikult dokumendivaateid ja nende kasutajaliideseid kohandada ning mis pakuks kasutajale kaasaaegsemat ja meelepärasemat kasutajakogemust ning mis sisaldaks võimalikult vähesel määral äriloogikat, kuid toetaks siiski kasutajate jaoks tellimustööna loodud erilahendusi.

2.3 Lähtetingimused

Ettevõtte koodivaramu ei ole avalik ning programmikoodi ei tohi mis tahes viisil või vormil väljaspool süsteemi ennast levitada, mistõttu antud bakalaureusetöö programmikoodi ennast ega selle katkendeid ei sisalda ning piirdub vaid süsteemi üldise tehnoloogilise ja arhitektuurilise kirjeldusega.

Bakalaureusetöö autori ja äritarkvara pakkuva ettevõte vahel sõlmitud konfidentsiaalsuse lepingu tingimustest lähtuvalt ei ole autoril õigust töös käsitleda äritarkvara klientide andmeid ega nende jaoks tellimustööna loodud lahenduste spetsiifikat. Bakalaureusetöös rakenduse turbega seotud osades käsitletakse turvet võimalikult üldjooneliselt ning ei kajastata võimalikke turvanõrkuste üksikasju ega ründe viise, isegi kui turvanõrkused ei ole praktikas enam realiseeritavad või pole seda kunagi olnud.

Autoril on lubatud esitada äritarkvara kasutajanõudeid, kasutajalugusid ning muid üksikasjalikult äritarkvara olemust ja funktsionaalsust kirjeldavaid andmeid üksnes katkenditena. Kasutajalugude ja -nõuete komplektide täies mahus esitamine on keelatud.

2.4 Eesmärk

Diplomitöö eesmärk on välja töötada tehnoloogiline lahendus äritarkvara Directo dokumendivaadete süsteemi jaoks, mis suudaks suhelda süsteemi tagarakendusega ning sealt saadud andmete põhjal kuvada kasutajale erinevaid dokumendivaateid.

Uus lahendus peaks olema pärandsüsteemiga võrreldes tunduvalt veakindlam ning vähendama süsteemi haldamisele kuluvat ressurssi, seda eelkõige läbi koodi ja loogika korduvuse vähendamise kasutades vaadete kuvamiseks ühtset loogikat ja komponente ning omades seejuures korralikku automaattestide kaetavust.

Väljatöötatud lahenduse kasutajaliides peaks olema kasutaja vajaduste piires hõlpsasti kohandatav ning toetama erinevate ettevõtete töövooge ja bakalaureusetöö käigus välja selgitatud kasutajanõudeid.

Lahendus peab olema heal tasemel dokumenteeritud ning tehnoloogilise poole juures kasutama modernse tarkvaraarenduse parimaid praktikaid ja arhitektuurilisi mustreid. Lisaks peab lahendus toetama töös välja selgitatud tulevikuvaateid ning edasisi arendussuundasid.

Bakalaureuse töö lõpptulemusena peab olema valminud lahendus, mis täidab eelpool mainitud nõudeid ning mille baasil on loodud vähemalt üks dokumendivaade, mis on läbinud kasutajatestimise, arendusjuhi ja tootejuhi poolt heaks kiidetud ning mis on äritarkvara orgaanilise osana päris klientide käes kasutusel ning millest tulenev kasu süsteemile tervikuna on selgesti määratletav.

Diplomitööga soovib autor anda ülevaate kaasaaegse äritarkvara tarbeks eesrakenduse loomise protsessist kui tervikust koos ettetulevate probleemide ja kaalutluste kirjelduste ja lahenduste strateegiatega. Diplomitöö eesmärk ei ole anda detailidesse laskuvat ammendavat kirjeldust loodud süsteemile ja selle spetsiifikale, vaid käsitleda probleemi

laiemalt ning kirjeldada arendusprotsessi sellisel viisil, mis võimaldaks bakalaureusetöös välja selgitatud tulemusi ja järeldusi rakendada teiste sarnaste projektide juures.

2.5 Skoop

Bakalaureusetöö skoop piirdub dokumendivaadete süsteemi eesrakenduse välja töötamisega ning sellele eelneva vajaliku analüüsi ja pärandsüsteemi funktsionaalsuse ja probleemide kaardistamisega. Lisaks kuulub skoopi uute kasutajanõuete välja selgitamine koostöös kasutajaliidese disainerite ja tootejuhtidega.

Peale analüüsi ja teostamise kuulub skoopi veel eesrakenduse automaattestimise strateegia välja töötamine ning lahenduse dokumenteerimine koos tõenäoliste tulevikuvaadete ja arendussuundade välja selgitamisega.

Bakalaureusetöö skoop ei sisalda ei süsteemi tagarakenduse analüüsi ega arendust. Samuti ei kuulu skoopi ettevõtte arendusmeeskondade töövoogude ja süsteemi evitusstrateegia analüüs ja võimalik optimeerimine.

Lähtudes töö eesmärkide all kirja pandud soovitud tulemustest kuulub skoopi loodud süsteemi alusel ühe reaalse dokumendivaate kasutajatestimine, tulevikuvaated ja retrospektiiv koos analüüsiga.

2.6 Metoodika

Kuna planeeritav lahendus sisaldab suuresti juba olemasolevale süsteemile parema alternatiivi loomist, on esmajoones tähtis läbi viia pärandsüsteemi erinevate dokumendivaadete analüüs, et selgitada välja juba olemasolev funktsionaalsus. Analüüsi käigus kogutakse kokku pärandsüsteemi kasutajanõuded, mille põhjal luuakse vastavad kasutajalood koos süsteemi üldist loogikat kirjeldavate skeemidega, mille eesmärk on anda parem ülevaade süsteemi hetkelisest ülesehitusest ning kasutusvoogudest.

Koostöös kasutajaliidese disainerite, tootejuhtide ja arendusmeeskonnaga prioritiseeritakse loodud kasutajalood ning filtreeritakse välja need, mis on tänaseks

juba ebavajalikud, mis ei oma erilist kasu süsteemil kasutatavusele või mille jaoks on võimalik välja töötada tõhusam alternatiiv.

Antud kasutajalugude kõrvale kogutakse koostöös samade inimestega ning lisaks klientidelt tulnud tagasisidele toetudes kokku kasutajanõuded uuele süsteemile, mida pärandsüsteem töö kirjutamise hetkel ei toetanud. Nende nõuete põhjal luuakse juurde kasutajalugusid, mida analüüsitakse, prioritiseeritakse ja filtreeritakse sarnaselt pärandsüsteemi nõuetega ning lõpptulemusena valminud kasutajalugude komplekti põhjal hakatakse planeerima uut süsteemi.

Süsteemi planeerimise juures analüüsitakse kasutajanõuetele toetudes erinevate dokumendivaadete ühisosade suurust ning töötatakse välja süsteem, mis võimaldaks ühisosa võimalikult suurel määral ühtse süsteemi abil kirjeldada ja realiseerida.

Uue süsteemi planeerimisel võrreldakse omavahel erinevaid tehnoloogiaid ja lahendusviise ning langetatakse põhjendatud valik. Valikust ja eelpool kogutud kasutajalugudest lähtuvalt planeeritakse uue süsteemi üldine struktuur. Peale struktuuri loomist ning arendusjuhi, kasutajaliidese disainerite ja tootejuhtide heakskiidu saamist alustatakse lahenduse välja töötamist.

Paralleelselt lahenduse välja töötamisega pannakse paika automaattestimise ja dokumenteerimise kavad ja üldised suunitlused, millest lähtuvalt luuakse süsteemile seda heal tasemel kirjeldav dokumentatsioon ja automaattestid.

Valminud lahendus antakse testida valitud hulgale päris kasutajatele. Kasutajatestimisest kogutud tagasiside põhjal luuakse uued kasutajalood, mida taaskord analüüsitakse ning vastavalt tulemustele muudetakse ja parendatakse süsteemi, peale mida evitatakse süsteem laiemale kasutajabaasile ning lahendus saab äritarkvara orgaaniliseks osaks.

Viimase etapina selgitatakse välja projekti tulevikuvaated ja edasised arendussuunad ning tehakse kindlaks lahenduse reaalne kasu ja efekt süsteemile tervikuna

3 Probleemi analüüs

Järgnevates peatükkides keskendutakse probleemi analüüsile ning kaardistatakse pärandsüsteemi funktsionaalsus, kasutatavuse probleemid ja turvariskid eesmärgiga saavutada ülevaade pärandsüsteemi olemusest. Lisaks selgitatakse välja äritarkvara dokumendivaadete süsteemi ühisosa koos soovitud lisafunktsionaalsusega.

3.1 Olemasoleva süsteemi kaardistamine

Parima võimaliku lahenduse välja töötamiseks oli esmajoones mõistlik analüüsida juba olemasolevat süsteemi. Kuna pärandsüsteem on väga vana ning selle dokumentatsioon ei olnud töö loomise ajal rahuldaval määral kättesaadav, tuli süsteemi kaardistamise juures lähtuda suuresti katse-eksituse meetodist.

3.1.1 Funktsionaalsuse kaardistamine

Suur osa dokumendivaadete funktsionaalsusest õnnestus kaardistada vähese vaevaga, peamiselt läbi eelpool mainitud katse-eksituse meetodi, mille käigus sai läbi proovitud erinevad võimalikud kasutamise viisid ning tulemuste vaatlemise ja koodi lugemise põhjal tehtud järelduste baasil sai formuleeritud potentsiaalsed kasutajanõuded. Lisaks oli suureks abiks tootejuhtide ja teiste arendusmeeskonna liikmetega suhtlemine, kelle teadmised süsteemi ja selle kasutamise viiside kohta olid oluliselt sügavamad.

Kuna äritarkvara kasutajaskond on lai ning erinevad ettevõtted vajavad oma ärimudelitest lähtuvalt erinevaid töövooge, sisaldavad dokumendivaated tihti üsna suurel määral peidetud oskuskasutajate jaoks loodud funktsionaalsust, mida kasutab väga väike, kuid äri poole jaoks oluline osa kasutajaskonnast. Antud funktsionaalsuse kaardistamine ja selle põhjal mõistlike kasutajanõuete loomine osutus oodatust keerulisemaks, kuna oskuskasutajate töövooge oli raske ratsionaliseerida ilma isikliku vastava erialase raamatupidamise kogemuseta. Siinkohal oli samuti palju abi tootejuhtidega läbirääkimisest, kes suutsid antud töövooge ja funktsionaalsust paremini

lõppkasutaja vaatepunktist seletada ning võimalusel efektiivsemaid alternatiive välja pakkuda.

Kogutud kasutajanõuded osutusid väga infotihedaks, pakkudes üsna head ülevaadet sellest, mida süsteem tegema peaks, kuid ei andnud ette head pilti miks antud nõuded lõppkasutaja jaoks tähtsad on või milline on nende efekt kasutuskogemusele. Sellest lähtuvalt ei olnud võimalik puhtalt kasutajanõuete komplekti põhjal suuremaid arhitektuurilisi otsuseid langetada ning nende edasine töötlemine oli vajalik.

Tabel 1. Katkend kogutud kasutajanõuetest töötlemata kujul

Nr	Nõue
1	Dokumendivaade peab sisaldama nuppu manuste lisamiseks.
2	Dokumendivaade peab võimaldama näidata kasutajale veateateid.
3	Dokumendivaatel peab olema nupp uue dokumendi loomiseks.
4	Dokumendivaatel peab saama näha antud dokumenti viimati muutnud isiku kasutajanime ja muutmise kuupäeva koos ajaga.
5	Dokumendivaatel peab olema võimalik kuvada seotud manuseid.

Kasutajanõuete hõlpsama haldamise meetodeid on tarkvaraarenduse maailmas väga palju. Nendest ühed populaarseimad ja bakalaureuse töö autorile ja tema arendusmeeskonnale kõige tuttavamad olid luua kasutajanõuetele tuginedes kas kasutajalood või kasutusmallid.

Kasutajalood annavad parema ülevaate lõppkasutaja vajadustest ning on kergemini loetavamad, seda eriti inimeste jaoks, kes pole tarkvaraarendajad, kuid kelle ekspertiis on lahenduse välja töötamise juures äärmiselt vajalik. Lisaks aitavad kasutajalood keskenduda lõppkasutajale ning tema töövoogudele ning annavad selgema ülevaate sellest, miks üks või teine lõppkasutaja kindlat aspekti süsteemi juures vajab. Mainitud plussid on aga kasutajalugude juures ka arendaja perspektiivist miinusteks, kuna nad on kasutajakesksed ning ei paku seejuures head ülevaadet süsteemi tehnilise poole kohta. Kasutusmallid seevastu pakuvad sügavamat ja detailsemat ülevaadet lahendusest endast ja selle osiste tehnilisest kooskõlast läbi lahenduse konkreetsete käitumisviiside kirjeldamise, mis aitab arendusmeeskondadel kergemini süsteemi tema nõuete ümber disainida ja arendada. Kasutusmallide miinuseks on aga, et nad kirjeldavad väga hästi

süsteemi nõutud käitumisviise, kuid ei anna seejuures head ülevaadet soovitud käitumisviiside tagamaadest ning võivad arenduse fookuse viia lõppkasutaja vajadustest liigselt kaugele [2] .

Antud asjaolusid arvesse võttes otsustati kogutud kasutajanõuete põhjal luua kasutajalood, kuna lahenduse üks eesmärkidest on parendada kasutuskogemust ning kasutajalood aitavad süsteemile läheneda oluliselt rohkem lõppkasutaja vaatepunktist. Lisaks on lahenduse välja töötamise juures tähtis koostöö tootejuhtide ja kasutajaliideste disaineritega, mistõttu on vajalik, et ka nemad nõuetest kergelt aru saaksid ning suudaksid nendele adekvaatselt hinnanguid anda ning neid vajadusel prioritiseerida.

Tabel 2. Kasutajanõuete katkendi põhjal loodud kasutajalood

Nr	Kellena	tahan...	selleks, et saaksin...
1	Raamatupidaja	lisada dokumendile manuseid	hiljem näha dokumendiga seotud väliseid andmeid, nagu ostuarved või tarnepaberid.
2	Raamatupidaja	näha dokumendil vea- ja hoiatusteateid	kohest tagasisidet vääralt sisestatud või sisestamata jäänud andmete kohta.
3	Raamatupidaja	võimalust dokumendivaatel uue dokumendi loomiseks	mugavalt ilma registri lehitsejat uuesti avamata luua uut dokumenti.
4	Vastutav isik	näha dokumendi viimati muutmise aega ja muutnud kasutaja nime	kindlaks teha, kes ja millal viimati antud dokumenti muutnud on.
5	Raamatupidaja	näha dokumendi vaatel seotud manuse dokumente	manustel sisalduvat infot dokumendile mugavamalt kanda või dokumendil olevate andmete õigsust kinnitada

Koostatud kasutajalugude komplektile andsid erinevad osapooled hinnangud ning vajadusel muudeti kasutajalugusid või loodi neid juurde eesmärgiga saavutada võimalikult hea kirjeldus hetkel toimivast süsteemist, kuna mõne süsteemi tähtsa funktsiooni mitte märkamine ja sellega süsteemi disaini juures mitte arvestamine võib sügavaid probleeme põhjustada tulevastes faasides.

3.1.2 Kasutatavuse probleemide kaardistamine

Funktsionaalsuse kaardistamisega saavutati süsteemist piisavalt hea ülevaade, et hakata selle põhjal analüüsima süsteemi kasutatavust üldiselt ning sellega kaasnevaid kitsaskohti. Kasutatavuse analüüsi juures tugineti peamiselt kasutajaliidese ja kasutajakogemuse disainerite erialastele teadmistele, millele arendusmeeskond pakkus omalt poolt juurde nõu probleemide süsteemse keerukuse ja lahenduste loomise kulukuse hindamisega.

Protsessi käigus tõstatas arendusmeeskond enda jaoks olulised leitud probleemid ning koostöös eriala spetsialistidega analüüsiti neid ning töötati välja lahendusi. Antud etapist parema ülevaate loomiseks on alljärgnevalt välja toodud üks oluline kaardistatud kasutatavuse probleem ja selle lahendamise protsess.

Probleem: dokumendivaadete väliste osiste avamine (manuste haldamise vaade, dokumendi meilimise ja välja trükkimise vaated) avab need täiesti uues hüpikaknas. Antud käitumisviis on problemaatiline, kuna hüpikaknate arv suurendab segadust kasutaja infoväljas ning selle haldamine nõuab kasutajalt suuremat kognitiivset pingutust. Hüpikaknate avamine võib põhjustada kasutajas kimbatust, kuna uue akna avanemine ei pruugi alati olla kergelt tabatav, hüpikaknate avamine piirab veebilehitseja „tagasi" nupu käitumist, need on raskesti kasutatavad mobiilseadmetes ning ei sulgu alati juurakna sulgemisega, mis jätab kasutaja infovälja üleliigset prahti [3] .

Antud käitumismustri positiivsete külgedena toodi välja asjaolu, et uute hüpikaknate avamine võimaldab kasutajal korraga näha originaalvaadet koos sellel kajastuvate andmetega ning originaalvaate kõrval avatud lisainfoga vaadet.

Alternatiivse lahendusena pakuti välja modaalaknate kasutamist. Modaalaknate eelisteks on võimalus avada põhivaatele kaasnev lisa tegevuse vaade, näiteks dokumendi manuste haldamise vaade, samas aknas, kui põhitegevuse vaade ise, mis kaotab ära hüpikaknatega kaasnevad probleemid. Lahenduse miinuseks on asjaolu, et modaalaken blokeerib ära originaalvaate ning kahte vaadet ei saa korraga kasutada. Kuna valdav enamus dokumendivaate sees hetkel hüpikaknas avanevaid lisavaateid ei vaja põhivaate andmetest otsest ülevaadet, otsustati modaalaknate kasutamise kasuks.

Kasutatavuse probleemide kaardistamise ja analüüsimise järel loodi kogutud andmete ja võimalike lahendussuundade põhjal juurde kasutajalugusid, eesmärgiga kirjeldada muutunud töövooge. Loodud kasutajalood lisati projekti tööjärge.

3.1.3 Turvanõrkuste kaardistamine

Enne uue süsteemi välja töötamist oli oluline kaardistada ja analüüsida kõik võimalikud turvanõrkused praeguses süsteemis ning teha nende põhjal järeldused ning lähtuvalt tulemustest välja selgitada võimalikud programmi osised, mille arendamise juures on oht turvanõrkuste tekkimiseks ning millega seoses tuleb rakendada lisa ressurssi võimalike turvanõrkuste juba arendusstaadiumis testimiseks ja leevendamiseks.

Arendusmeeskond eesotsas bakalaureusetöö autoriga teostas rakenduse turvatestimise tuginedes kollektiivsele asjatundlikkusele veebirakenduste turvalisuse valdkonnas. Turvatestimine viidi läbi kasutades peamiselt läbistustestimise metoodikat selleks spetsiaalselt ettevalmistatud arenduskeskkonnas, kus süsteemi andmetena kasutati ainult testandmeid, millel polnud seost süsteemi päris klientide ega nende ettevõtete andmetega, seda ennekõike vältimaks turvatestimise käigus delikaatsete andmete lekkimist või nende kättesaadavaks muutumist volitamata isikutele.

Turvatestimise juures võeti aluseks sihtasutuse OWASP veebiturvalisuse testimise juhend [4] , kuna see on vabalt kättesaadav ning sisaldab antud projekti jaoks piisaval määral tarkvaraarenduse kogukonna poolt heaks kiidetud parimaid praktikaid ja juhiseid. Turvatestimist viidi läbi äripoole jaoks vastuvõetavas mahus ning juhendis keskenduti punktidele, mis puudutasid kõige enam antud projekti võimalikke turvanõrkuseid, seda eesmärgiga optimeerida turvatestimisele kuluvat ressurssi.

Turvatestimise käigus pandi erilist rõhku XSS tüüpi turvanõrkuste avastamisele, mis on oma olemuselt eriti ohtlikud, kuna ainuüksi ühe taolise turvaaugu ära kasutamine võib võimaldada ründajal realiseerida ka palju muid tüüpi rünnakuid, näiteks võib see võimaldada ründajal varastada ära kasutaja sessiooni küpsised, mille abil on võimalik kaaperdada kasutaja sessiooni ning seeläbi võtta üle tema konto, mis omakorda

võimaldab juba ligipääsu delikaatsetele andmetele ning nende võimalikku võltsimist. Lisaks loob XSS turvanõrkus võimaluse laadida sisse veebileheküljele välist sisu, mille abil on võimalik muuta märkimisväärsel määral kasutajaliidest ning suunata kasutaja enese teadmata ohtlikele välistele veebisaitidele [5] . Antud asjaoludest lähtuvalt võttis äripool bakalaureusetöö autori soovitusel vastu otsuse rakendada nulltolerantsi kõigile võimalikele XSS ründevektoritele, mida võeti arvesse ka uue süsteemi arendusel.

Turvatestimise tulemused analüüsiti ning nende põhjal tehti vastavad järeldused seoses uue süsteemi arendamisega, lisaks tehti turvatestimise tulemused teatavaks arendusjuhile, kes rakendas vastavad meetmed leitud turvanõrkuste lahendamiseks pärandsüsteemis.

3.2 Vaadete ühisosa leidmine

Peale pärandsüsteemi põhjalikku analüüsi ja selle erinevate aspektide kaardistamist oli bakalaureusetöö autori arvates saavutatud piisavalt hea ülevaade pärandsüsteemist ja selle osistest ning järgmise etapina asuti lahendama üht antud projekti põhiprobleemidest – erinevate dokumendivaadete ühisosa leidmine eesmärgiga töötada välja ühtne dokumendivaadete süsteem.

Kuna Directo äritarkvaraga konkureerivad sarnased lahendused on peaaegu täiel määral suletud lähtekoodiga ning ei avalda oma lahenduste detaile, ei ole antud probleemi jaoks lahenduse välja töötamise juures võimalik analüüsida teisi sarnaseid lahendusi. Lisaks on Directo oma süsteemi olemuselt turul küllaltki ainulaadne, mistõttu konkureerivate lahenduste analüüsimine ei aitaks eriti palju uue lahenduse süsteemse poole probleemide lahendamise juures. Mainitud asjaoludest lähtuvalt tuli probleemile läheneda *ad hoc* stiilis.

Vaadete ühisosa leidmisel tugineti varem kogutud kasutajalugude komplektile, mille analüüsimise tulemusena koguti kokku need osad süsteemist, mille käitumine erinevate dokumendivaadete vahel on piisavalt sarnane ning mida oleks võimalik realistlikult kuvada ühtsete süsteemi komponentidega.

Ühisosa leidmise juures arvestati erinevate piirjuhtudega, mille korral võivad teatud osad süsteemist käituda teatud tingimustes ebatavaliselt, seda ennekõike

oskuskasutajatele mõeldud keeruliste kasutusvoogude tõttu. Siinkohal võeti arvesse varem läbi arutatud võimalikud kasutatavust puudutavad muudatused ning kasutusvood, mis takistasid süsteemi ühtlustamist ning mille kasutajaskond oli äärmiselt väike või mille jaoks oli olemas parem alternatiiv, kaotati ühtlustamise eesmärgil ära.

Ühisosa väljatöötamise tulemusena leidis autor, et praegused dokumendivaated jagunevad oma olemuselt suuremateks ühtseteks osadeks, mida hakati nimetama sektsioonideks. Sektsioonid ise võivad sisaldada ka alamsektsioone. Sellest lähtuvalt kujundati dokumendivaatest sektsioonide puu, kus juurtipuks oli dokumendivaade ise ning alamtippudeks olid dokumendivaate erinevad sektsioonid ja nendes sisalduvad alamsektsioonid. Puu lehtedeks olid sektsioonides paiknevad elemendid, mis enamasti olid kas sisendväljad või dokumendi tegevuste nupud.

Dokumendi sektsioonid jaotas autor kahte suuremasse kategooriasse. Esimest neist otsustati nimetada pakendsektsioonideks. Need olid sektsioonid, mis ise sisendvälju ega tegevusi ei sisaldanud ning mille ülesanne oli pelgalt ülejäänud sektsioone pakendada ning vastavalt sektsiooni olemusele neid erineval kujul vaates välja kuvada. Seda tüüpi sektsioonideks olid näiteks sakksektsioonid, mis olid dokumendi sektsioonipuu osad, mille ülesandeks oli oma alamsektsioonide pakendamine ja vaates erinevate sakkide all kuvamine. Pakendsektsioonid võimaldasid seega erinevaid sektsioone gruppidesse jagada ning defineerida, kuidas mingid dokumendivaate osised paiknevad.

Teise suure kategooria alla jaotati sektsioonid, mis sisaldasid endas reaalseid andmeid ja/või dokumendiga seotud tegevuste nuppe. Sellised sektsioonid alamsektsioone ei sisaldanud, küll aga olid nende järglasteks sektsioonipuus puulehed ehk sektsiooni elemendid. Selliseid sektsioone otsustati nimetada põhisektsioonideks. Põhisektsioonideks olid enamasti kas sisendvälja grupid, mida dokumendivaadetes nimetati äritarkvara siseselt enamasti dokumendi päiseks, ning dokumendi tabelid, mida tarkvara siseselt nimetati dokumendi ridadeks. Põhisektsioonid sisaldasid erinevalt pakendsektsioonidele päris andmeid ning sellest sõltuvalt moodustasid need ka dokumendivaadete põhiosa.

Väljatöötatud sektsioonipuu süsteem võimaldas peaaegu täiel määral ära kirjeldada kõikide praeguste dokumendivaadete olemuse ja sisemise loogika (vt. lisa 2) ning võimaldas teoorias välja töötada iga sektsiooni jaoks eraldi komponendi, mis võtaks

sisse sektsiooni koos selle andmetega ning kuvaks vastavalt andmetele kasutajale selles defineeritud vaate osist. Antud lähenemine lahendas lisaks dokumendivaadete ühtlustamise probleemile ka dokumendivaadete kohandamise ja tellimustööna tehtud erikujunduste probleemid, kuna vaate kujunduse muutmiseks piisaks ainult sektsioonipuus olevate andmete muutmisest ning sektsioone ennast kujutavad süsteemi komponendid jääksid seejuures muutmata.

Sektsioonipuu süsteemi suurimaks miinuseks on selle osistevahelise äriloogika defineerimise keerukus. Kuna antud lahenduse juures on lahenduse eesmärkidest ja probleemi püstitusest tulenevalt tähtis hoida loodavas eesrakenduses võimalikult vähe äriloogikat ning eesrakendus ise peaks olema andmete seisukohalt võimaluse korral kõrvalmõjudeta, sobib sektsioonipuu lahendus soovitud eesmärkidega väga hästi ning äriloogika defineerimise keerukus ei tohiks lõpplahendusele erilist mõju avaldada.

Ülalpool kirjeldatut arvesse võttes leidis bakalaureusetöö autor, et sektsioonipuu süsteem on kooskõlas püstitatud eesmärkide ja nõuetega ning lahendab väga hästi dokumendivaadete ühtlustamise, nende kohandamise ja ühtsete komponentidega kuvamise probleemid.

Sektsioonipuule ja selle osistele koos nendes sisalduvate andmetega ning nende mõjule dokumendivaadetes antakse täpsem ülevaade bakalaureusetöö peatükis 4.5.

3.3 Lisafunktsionaalsuse olemasolevaga sidumine

Pärandsüsteemi pika kasutusea jooksul oli laekunud palju tagasisidet nii tootejuhtidelt kui ka päris kasutajatelt seoses süsteemi kasutatavuse ja soovitud lisafunktsionaalsusega. Suurt osa soovitud muudatustest ei olnud võimalik pärandsüsteemis selle tehnoloogiliste ja arhitektuuriliste piirangute tõttu realiseerida. Kuna uus süsteem on oma olemuselt oluliselt paindlikum ning uute kasutusvoogude ja suuremate muudatuste sisse viimiseks on kahe süsteemi vaheline üleminekuperiood soodne aeg, otsustati enne uue süsteemi praktilise osa loomise juurde asumist kokku koguda ja analüüsida soovitud lisafunktsionaalsus ning analüüsi tulemuste põhjal leida viis, kuidas need siduda olemasoleva süsteemiga ning ressursi olemasolu korral ka soovitud funktsionaalsus lahenduse juures täide viia.

Arendusmeeskond ega bakalaureusetöö autor ise kasutajate tagasiside haldamise ega selle põhjal lisafunktsionaalsuse kogumisega ei tegelenud ning uute kasutajanõuete kogumine ja nende tähtsuse ja vajalikkuse välja selgitamine jäeti projektijuhtide ja arenduskomitee otsustada. Viimaste käest kogutud kasutajanõuete põhjal lõi arendusmeeskond juurde kasutajalugusid, arutas läbi nende täideviimise keerukuse ning mõju loodavale süsteemile tervikuna ning lisas uued kasutajalood projekti tööjärge.

Uued kasutajanõuded olid suuresti kooskõlas loodavale süsteemile püstitatud eesmärkide ja nende täitmisega seotud lahendust vajavate probleemidega ning enamusele neist leiti võrdlemisi lihtne ja soodne lahendus, mis ei vajanud juba planeeritud süsteemi ringi disainimist. Alljärgnevalt on kogu protsessi paremaks illustreerimiseks välja toodud üks süsteemi kujundamisele ja selle kasutatavusele suurimat mõju avaldanud uus kasutajanõue ning selle süsteemiga sidumise strateegia koos kaasnenud probleemide ja nende lahendamisega.

Üheks tähtsamaks uueks kasutajanõudeks oli kasutajate soov dokumendi muutmisel nö mustandi loomiseks ehk sooviti võimalust muuta dokumendi andmeid sedasi, et see algdokumendi koheselt ei muudaks, vaid kannaks muudetud andmed algdokumendile alles peale mustandi salvestamist. Tagarakendust ja andmebaasi arendavate arendusmeeskondadega läbirääkimise tulemusena leiti probleemile esmane võimalik lahendus, mis hõlmas endas kahe erineva andmekomplekti loomist: esimene neist oleks andmed, mis moodustavad algdokumendi, ning teine algdokumendi andmete pealt loodud mustandi andmete komplekt. Dokumendile tehtud muudatused kajastuksid seega mustandi andmetekomplektil säilitades seejuures algandmete terviklikkuse. Mustandi salvestamisel kirjutataks algandmed üle mustandi andmetega, peale mida mustandi andmed kustutataks ning asendatakse uuesti algandmetega salvestatud dokumendi järjekordsel muutmisel. Selline lähenemine võimaldaks kasutajale võrdlemisi intuitiivset kasutuskogemust ning ei eeldaks juba pärandsüsteemis eksisteerivate dokumentide andmete keerukat üle toomist, kuna algandmete komplekt ning selle talletamise struktuur süsteemis jääks muutmata.

Küll aga eeldaks selline lähenemine, et eesrakendus teab, mis olekus dokument hetkel on (kas mustand või salvestatud dokument) ning vastavalt olekule küsiks erinevat andmekomplekti tagarakenduse liidese erinevatest otspunktidest, mis läheks aga

osaliselt vastuollu püstitatud eesmärgiga hoida eesrakendus võimalikult äriloogika vaba. Lisaks võimaldaks antud lahendus iga dokumendi kohta ainult ühte mustandit, mis tekitab probleeme olukordades, kus üks kasutaja muudab mustandit, kuid ei salvesta seda. Kuna dokumendil saab olla ainult üks mustand korraga, ei saaks sellises olukorras teised kasutajad antud dokumenti muuta, kuna sellel juba eksisteerib salvestamata mustand. Probleemi lahendaks mustandi sidumine mitte ainult dokumendiga, vaid ka kindla kasutajaga, mis tagaks iga kasutaja jaoks ühe unikaalse mustandi dokumendi kohta. Antud lahendus kaotaks aga ära võimaluse teistel kasutajatel mugavalt loodud mustandit näha, kuna mustand oleks lisaks dokumendile ka kasutajapõhine. Lahendus eeldaks keeruka kuvatava mustandi valimise süsteemi välja töötamist, mis tõstaks oluliselt süsteemi arenduseks kuluvat ressurssi ja süsteemi sisemist keerukust ning mõjutaks negatiivselt kasutuskogemust.

Eelpool mainitud arvesse võttes otsustati luua kompromiss – rakendatakse kahe andmekomplekti süsteemi, kuna see sobib väga hästi kokku praeguse süsteemi andmete talletamise struktuuriga ning on võrdlemisi lihtne realiseerida nii uues planeeritavas eesrakenduses kui ka süsteemi tagarakenduses, kuid seejuures võimaldatakse iga dokumendi kohta ainult üks mustand ning pakutakse kasutajale võimalust vajaduse korral teiselt kasutajalt tema loodud mustand üle võtta. See lahendab probleemi, kus kasutajad saaksid blokeerida dokumendi muutmist nende poolt loodud mustandi salvestamata jätmisega. Eesrakendusse äriloogika viimist mustandi oleku haldamise näol ei peetud suureks probleemiks, kuna liidestel puudub vajadus antud loogikale ning oleku haldus tagarakenduses eeldaks selle arhitektuuris olulisi ja kalleid muudatusi. Lisaks otsustas äripool, et vajadus rohkem kui ühe mustandi järgi dokumendi kohta on äärmiselt ebatavaline ning antud tarkvaraomaduse realiseerimine nõuaks ebamõistlikult palju ressurssi, pakkudes vastu kaduvväikest tõusu süsteemi kui terviku kasutatavusele.

Antud lahendusega paralleelselt kaaluti ka alternatiivi, mis hõlmaks endas mitterelatsiooniliste andmebaaside kasutamist dokumendi muudatuste salvestamiseks, kuid mis tooks endaga kaasa jällegi suured ja kallid muudatused tagarakenduse arhitektuuris, mistõttu jäeti kirjeldatud alternatiiv kõrvale. Mustandisüsteemi loogikat illustreerib lisa 3.

4 Lahenduse loomine

Peale olemasolevast pärandsüsteemist ning planeeritava süsteemi nõuetest ja eripäradest põhjaliku ülevaate loomist koos erinevate dokumendivaadete ühisosa välja selgitamisega alustati lahenduse enda välja töötamist ning sellega kaasnevate probleemide lahendamist.

4.1 Nõuded lahendusele

Enne loodava lahenduse tarbeks tehnoloogiate valimist otsustati esmalt välja selgitada tehnoloogilised nõuded planeeritud lahendusele.

Loodav eesrakendus peaks kasutama ühelehe veebirakenduse arhitektuuri, eesmärgiga võimaldada kõrgemat interaktiivsust võrreldes serveripoolel loodud staatiliste veebilehtedega. Sellest tulenevalt peaks loodav lahendus kasutama mõnda laialt levinud ühelehe veebirakenduste loomiseks mõeldud raamistikku, kuna raamistike kasutamine aitab vähendada arendamiseks kuluvat ressurssi, tõsta rakenduse jõudlust, veakindlust ning projekti jätkusuutlikust tulevikus [1] .

Loodav lahendus peaks olema piisavalt hea jõudlusega ning võimaldama vajaduse korral välja kuvada kuni 100 000 realisi andmetabeleid. Tarkvaras eksisteerib dokumendivaateid, nagu näiteks palga dokumendid, kus on äriloogikast tulenevalt mõnes olukorras väga palju ridu ning ridade hulk võib ulatuda kümnetesse tuhandetesse. Praegune süsteem selliste dokumentide vaatamist kasutajatele ei võimalda, kuid märkimisväärse osa jaoks äritarkvara kasutajaskonnast on selliste mahukate dokumentide vaatamine oluline.

Loodav lahendus peaks võimaldama muuta kõigis dokumendivaate sisendväljade gruppides ja dokumendivaadete tabelites väljade järjekorda ja väljade nähtavust. Peale selle peaks dokumendivaadete tabelites olema võimalik tulpade järgi ridu sorteerida, tulpade laiust muuta, tulpasid suurte horisontaalselt keritavate tabelite korral vasakule ja paremale vaate serva külmutada ning ridu nende sisu järgi filtreerida. Lisaks peaks

vaates tehtud kohandused võimalusel salvestuma rakenduses koheselt kasutaja ja vaate põhiselt, seda kõike eesmärgiga pakkuda kasutajatele nende spetsiifilistest ärilistest vajadustest lähtuvalt võimalust vaadete mugavaks ja ulatuslikuks kohandamiseks.

Viimase olulise punktina peaks eesrakendus olema kasutatav erinevate ekraanisuuruste korral ning erinevates seadmetes, sh mobiilseadmetes, kuna äritarkvara kasutajaskond on jällegi lai ning paljud kliendid kasutavad tarkvara erinevaid osiseid paljude erinevate seadmetega.

4.2 Tehnoloogiate valik

Lahenduse tehnoloogiate valik on süsteemi välja töötamise juures väga suure tähtsusega, kuna valitud tehnoloogiad on kogu süsteemi nurgakivideks ning ebasobivalt langetatud valik võib tähendada eesmärkide täitmata jäämist ning projekti ebaõnnestumist. Kuna planeeritud süsteem on oma mahult suur ning selle täielik välja arendus kestab hinnanguliselt mitu aastat, on süsteemi planeerimise juures parimate võimalike tehnoloogiate valimine äärmiselt tähtis. Alljärgnevalt on välja toodud süsteemi kahe olulisima tehnoloogia valiku protsess.

4.2.1 Kasutajaliidese põhiraamistiku valik

Kaasaaegses tarkvaraarenduses eksisteerib kasutajaliidese loomiseks väga palju erinevaid lahendusi, mistõttu on oluline raamistiku valimisel langetada põhjendatud ja läbimõeldud valik, mistõttu lähtuti kasutajaliidese loomise tarbeks raamistiku valimisel kolmest peamisest asjaolust:

1) Raamistiku populaarsus ja saadavalolev tehniline dokumentatsioon ning selle kvaliteet – mida populaarsem on raamistik, seda paremini on ta üldjuhul kogukonna poolt testitud ning seda parem on üldine tehniline tugi ja raamistiku jätkusuutlikkus, kvaliteetne ja kättesaadav dokumentatsioon vähendab arendusele kuluvat ressurssi.

2) Arendusmeeskonna teadmised raamistikust ning raamistiku kasutamiseks vajaliku lisaõppe maht – mida kõrgemad on arendusmeeskonna juba olemasolevad teadmised raamistikust ning mida väiksem on vajalik lisaõppe

maht, seda kiirem on arendusprotsess ning seda vähem ressurssi süsteemi välja töötamine nõuab.

3) Raamistiku jõudlus ja koodi maht – parem jõudlus tagab lõppkasutaja jaoks parema kasutatavuse ning raamistiku koodi mahu liiasus piirab rakenduse laadimise kiirust, seda eriti aeglaste võrguühenduste korral.

Tuginedes veebikeskkonna Stackoverflow meeskonna rohkem kui 80 000 osaleja seas läbiviidud uurimusele, milles küsiti, milliseid raamistikke on osalejad viimase aasta jooksul tarkvaraarenduses laialdaselt kasutanud, on 2021. aasta seisuga populaarseimad veebiraamistikud React.js (40.14%), jQuery (34.42%), Express (23.82%), Angular (22.96%) ja Vue.js (18.97%) [6] . Kuna jQuery populaarsus on võrreldes sama meeskonna poolt 2020. aastal läbiviidud uurimusega langenud 8.88% (populaarsus 2020. aastal 43.3%) [7] ning võrreldes 2019. aastal läbiviidud uurimusega langenud 14.28% (populaarsus 2019. aastal 48.7%) [8] , oli töö loomise hetkel jQuery populaarsus selgelt kahanemise trendis, mistõttu jäeti jQuery edasises võrdluses kõrvale. Kuna Express on Node.js serveripoole raamistik ning pole loodud ühelehe veebirakenduste loomiseks [9] , jäeti ka see raamistik edasises võrdluses kõrvale ning järgnes React, Angular ja Vue raamistike omavaheline võrdlus.

Eesrakenduse arendusmeeskonna teadmiste kaardistamisel React, Angular ja Vue raamistike osas selgus, et meeskonna liikmetel on parimad teadmised React raamistikust ning seda oldi varem ka ettevõtte siseselt arenduses kasutatud. Teadmised Vue ja Angular raamistikest olid piiratud, mistõttu osutus antud kriteeriumis selgeks favoriidiks React raamistik.

Viimaseks võrreldi kolme raamistiku jõudlust ja nende toimimiseks üle võrgu kliendi arvutisse saadetava vajaliku koodi mahtu. Võrdlusel tugineti 2019. aastal läbiviidud uurimusele, milles võrreldi omavahel React, Vue ja Angular raamistike jõudlust ja koodi mahtu. Uurimuse tulemustest selgus, et Vue raamistik oli testitud oludes viis korda kiirem kui Angular ja kaks korda kiirem kui React. Koodi mahu osas osutus võitjaks React, teiseks tuli Vue ja kolmandaks Angular [10] .

Kõigis kolmes kriteeriumis jäi Angular selgesti alla React ja Vue raamistikele, mistõttu Angular eemaldati võrdlusest. Eelpool leitud tulemuste põhjal on React oluliselt

populaarsem, arendusmeeskonna poolt palju paremini tuntud ja väiksema koodi mahuga. Ainuke miinus Vue raamistiku ees on jõudlus, mis autori hinnangul polnud piisav React raamistiku eeliste üle kaalumiseks, mistõttu valiti kasutajaliidese põhiraamistikuks React.

4.2.2 Olekuhaldussüsteemi valik

Veebipõhiste eesrakenduste arenduses on äärmiselt tähtis rakenduse oleku adekvaatne haldamine ning sellest tulenevalt ka sobiva olekuhaldussüsteemi valik. Veebirakenduste oleku haldamiseks eksisteerib väga palju erinevaid vahendeid, alustades React raamistiku enda poolt pakutud Context liidesest ning lõpetades suure hulga populaarsete väliste olekuhaldusteekidega nagu Redux, MobX ja Recoil.

Kuna loodav eesrakendus on planeeritud sisaldama vastavalt läbiviidud analüüsile üsna suurt hulka keerulist rakenduse olekust sõltuvat kasutajaliidese loogikat ning suurt hulka andmeid, otsustati olekuhaldussüsteemi valimisel võtta arvesse süsteemi paindlikkust, jõudlust suurte ja tihti muutuvate andmekomplektide korral ning arendusmeeskonna üldisi teadmisi antud tehnoloogiast ja selle arhitektuurilistest mustritest. Kuna kasutajaliidese põhiraamistikuks valiti React, oli suure tähtsusega olekuhaldussüsteemi kokkusobivus React raamistikuga.

Esmalt selgitati välja, kas väliste olekuhaldussüsteemide kasutamine on üldse vajalik või rahuldaks süsteemi vajadusi ka mõni Reacti sisseehitatud võimalus, kuna väliste sõltuvuste lisamine tõstab süsteemi keerukust ja üldist koodi mahtu. Reacti ametlik dokumentatsioon soovitab olekuhalduseks Context liidest [11] . Context liidesest üksi jääb aga väheseks, kuna see võimaldab pelgalt ainult väärtuste salvestamist ja komponentideni tarnimist, kuid ei sisalda ühtegi sisseehitatud tööriista andmete ega süsteemi oleku haldamiseks. Lisaks ei ole Context liidest soovitatav kasutada rakenduste juures, mille olek muutub tihti, kuna vastasel juhul võib probleeme tekkida jõudlusega [12] . Antud asjaoludest lähtuvalt leiti, et Context liides ei ole loodava süsteemi juures piisav ning tuleb kasutada mõnda välist võimekamat lahendust.

Kuna olekuhaldussüsteemide populaarsuse ja kasutatavuse võrdlemiseks töö kirjutamise hetkel sobivaid laiapõhiseid objektiivseid uurimusi ei leitud, otsustati valiku juures lähtuda arendusmeeskonna oma kogemustest erinevate olekuhaldussüsteemidega.

Võimalikuks kandidaadiks pakkus töö autor välja Redux tarkvarateegi, kuna antud teek on tarkvaraarenduse maailmas väga populaarne (kasutusel üle 1.8 miljonis erinevas arendusprojektis [13]) ning sellest tulenevalt võib eeldada, et antud lahendus on ennast tõestanud ning on käesoleva projekti vajaduste jaoks piisavalt hästi dokumenteeritud. Redux tarkvarateegi olemuse, sobivuse, võimekuse, jõudluse ja paindlikkuse analüüsil tugineti arendusmeeskonna varasemale kogemusele Redux'i kasutamisel teistes sarnastes projektides ning Redux'i ametlikule dokumentatsioonile [14] . Analüüsi tulemustena leidis autor, et antud tarkvarateek sobib väga hästi püstitatud nõuetega. Ühtegi argumenti Redux'i mittesobivuse kohta ei suudetud leida. Kõike mainitud arvesse võttes otsustatigi antud tarkvarateek loodava lahenduse olekuhaldussüsteemina kasutusele võtta.

4.3 Automaattestimine

Üheks antud projekti tähtsaks eesmärgiks oli tõsta olulisel määral äritarkvara veakindlust. Pärandsüsteemi arenduse ja haldamise juures kulus märkimisväärne ressurss tarkvararegressiooni vältimise ja parandamise peale. Tarkvararegressiooni all peetakse antud kontekstis silmas tarkvaravigu, mille korral lakkab töötamast varem toiminud funktsionaalsus. Lisaks halvemale kasutuskogemusele ja sellest tingitud tarkvara konkurentsivõime langusele toob tarkvararegressioon endaga kaasa ka suurenenud töökoormuse kasutajatoele ja arendajatele endile ning aeglustab seetõttu ka olulisel määral arenduse tempot, mistõttu on tarkvararegressiooni esinemise miinimumini viimine antud projekti raames äärmiselt tähtis.

Pärandsüsteemi analüüsi käigus selgitati välja, et üheks peamiseks tarkvararegressiooni esinemise põhjuseks on pärandsüsteemis ebapiisav automaattestimise kava, mistõttu ei ole võimalik regressioonivigu õigeaegselt avastada. Sellest lähtuvalt otsustati enne uue eesrakenduse programmikoodi kirjutamise juurde asumist paika panna selge automaattestimise strateegia.

Automaattestimise strateegia detailide välja töötamise juures lähtuti erinevate tarkvaraarenduse ettevõtete peal läbi viidud uurimustest automaattestimise strateegiate ja filosoofiate tõhususe ja tagajärgede osas. Üheks neist oli ettevõtte IBM sees läbi viidud uurimus [15] , milles leiti, et pärast TDD arendusmetoodika kasutusele võtmist

langes tarkvaradefektide esinemise sagedus ligikaudu 50% ning uurimuse läbiviijate hinnangul tõusis oluliselt tarkvara osiste paindlikkus. Positiivsete külgede kõrval tõdeti aga, et TDD arendusmetoodika kasutusele võtmine langetas mõnevõrra arendajate tootlikkust.

Kuigi IBM ja Directo OÜ ei ole oma olemuselt eriti sarnased, leiti siiski, et uurimuses kirjeldatud varasem arendusmetoodika ning sellest tulenevad probleemid langesid märkimisväärses mahus kokku ettevõtte Directo OÜ siseste protsesside ja probleemidega, mistõttu peeti kahe ettevõtte kõrvutamist TDD kasutusele võtmise kasu ja tagajärgede osas põhjendatuks.

Ühe suure probleemina automaattestimise kava välja töötamise juures nähti võimalust, et automaattestide kirjutamise peale kulub liiga palju ressurssi ja testidest tulenev kasu on liiga väike, mistõttu pole võimalik saavutada projektile seatud eesmärke. Sellest lähtuvalt otsustati mitte seada meelevaldseid sihte automaattestide kaetavusele ning kaetavuse määra piisavuse hindamine jäeti arendusmeeskonna liikmetele nende enda erialaste oskuste ja kogemuste põhjalt otsustada.

Viimase aspektina võrdles autor erinevate automaattestide liikide mõju ja kasulikkust eesmärgiga selgitada välja, millist liiki automaatteste ja millises mahus oleks kõige mõistlikum kirjutada. Peamiselt üritati seada paika vahekorda moodultestide ja integratsioonitestide vahel. Moodultestid võimaldavad testida igat tarkvaraosist eraldi, kuid nende miinuseks on testide arvu kasvuga kiiresti kahanev kasutegur ühe testi kohta, kuna testitavad piirjuhud ja nende esinemine päriselu olukordades mitme komponendi koos toimimisel on väga väikese tõenäosusega. Lisaks kulub moodultestide haldamise peale märkimisväärne ressurss, kuna need tuleb tarkvaraosiste sisemise seadmestuse muutumise korral tihti ringi kirjutada. Integratsioonitestid seevastu testivad tarkvara osiste koostööd palju kõrgemal ning kasutajalähedasemal tasemel, pakkudes seega ühe kirjutatud testi kohta suuremat kasutegurit. Lisaks vajavad integratsioonitestid palju harvemini ringi kirjutamist, kuna tarkvaraosise sisemise seadmestuse muutus ei too endaga kaasa vajadust integratsioonitestide ringi kirjutamiseks eeldusel, et komponentide koostöö loogika ja nende liidestus ei muutunud.

Kõike mainitud arvesse võttes leidis autor, et tarkvararegressiooni vältimise ja modulaarsema arhitektuuri huvides on mõistlik kirjutada nii palju automaatteste kui võimalik ning lähtuda mõistlikkuse piirides TDD metoodikast. Testide kirjutamise juures tuleks kasuteguri optimeerimise huvides kirjutada rohkem integratsiooniteste kui moodulteste. Mainitud leiud ning nende põhjal formuleeritud juhised koos argumentidega tehti teatavaks arendusmeeskonnale ning nendest lähtuti projekti arenduse vältel.

4.4 Lahenduse dokumenteerimine

Pärandsüsteemi analüüsi käigus selgus, et kuigi süsteemi jaoks on loodud küllaltki mahukas ja ammendav viki, mis annab väga hea ülevaate süsteemi üldistest funktsioonidest ja käitumisest, ei ole see piisav, et anda edasi arenduseks vajalikku täit informatsiooni. Dokumentatsiooni piiratus mõjutab oluliselt arendajate omavahelist koostööd puuduva teabevahetusmeediumi tõttu ning aeglustab seeläbi arendusprotsessi, mistõttu otsustati loodavas eesrakenduses paika panna plaan rakenduse süsteemse poole dokumenteerimiseks.

React raamistiku arhitektuur on üles ehitatud taaskasutatavate vaatekomponentide põhimõttel [16] , mistõttu leiti, et kõige mõistlikum on dokumenteerida neid samu vaatekomponente koos nende liideste kirjelduse ja komponendi enda kasutamise näidete ja viisidega.

Komponentide dokumenteerimiseks pakuti välja kaks suunda. Esimene neist hõlmas tarkvaralahenduste kasutamist, mis võimaldaksid komponentide liidestele kirjutatud dokumentatsioonisõnede pealt automaatselt luua dokumentatsioonivaateid, mis pakuksid arendajatele kiiret ja head ülevaadet komponendi parameetritest, liidesest ning kasutusviisidest koos vastavate näidetega. Antud lahenduse miinuseks oleks aga vajadus allutada suuremat hulka ressurssi vastavate tarkvaralahenduste leidmisele, võrdlemisele ja seadistamisele, lisaks ei pruugi valmis lahendused alati vastata soovitud nõuetele. Teiseks lahenduseks pakuti välja dokumenteerimisvaadete ise käsitsi koostamist. Antud lahenduse eeliseks oleks suurim võimalik paindlikkus. Miinusteks aga on dokumentatsiooni loomisele kuluv kõrgendatud ajaline ressurss ning komponentide muutmisel oleks tarvis dokumentatsioon käsitsi muudatustega vastavusse viia.

Mainitut arvesse võttes langetati otsus dokumentatsiooni automaatselt loovate tarkvaralahenduste kasutamise kasuks, seda eelkõige lähtuvalt asjaolust, et kuigi antud variant nõuab alguses rohkem ressurssi, võimaldab see tulevikus oluliselt kokku hoida aja ja arendaja töömahu arvelt. Vastavate tarkvaralahenduste võrdlusi ning välja valimise protsessi bakalaureusetöös ei käsitleta.

4.5 Süsteemi andmevoog ja tagarakendusega suhtlus

Tulenevalt projektile püstitatud eesmärgist hoida võimalikult palju vaadete äriloogikat süsteemi tagarakenduses, oli oluline paika panna täpne viis, kuidas tagarakendus suudaks kõige hõlpsamalt vajalikud andmed eesrakenduseni tarnida ning kuidas eesrakendus vajalikke andmeid tagarakendusse tagasi saadaks.

Kuna tagarakendus oli erinevatel antud töös mittekäsitletavatel kaalutlustel juba varem disainitud ja loodud kasutama REST arhitektuuril põhinevaid otspunkte info sisse võtmiseks ning välja saatmiseks, tuli ka eesrakendus disainida kasutama REST arhitektuurist tulenevaid mustreid.

Dokumendivaadete ühtse süsteemi tarbeks välja töötatud sektsioonipuu (vt sisupeatükk 3.2) struktuuri jaoks oli vajalik, et tagarakendus suudaks eesrakendusele ette anda korrektse sektsioonipuu, milles oleks ära kirjeldatud kõik vajalik informatsioon eesrakenduse kindla dokumendivaate toimimiseks. Sellest lähtuvalt kujundati sektsioonipuule vastav JSON formaadis andmestruktuur, mida tagarakendus suudaks päringu peale eesrakendusse läbi oma REST otspunktide saata ning mille iga sektsiooni juurde kuulus vajalik teave sektsioonis paiknevate andmete pärimiseks ning sektsiooni elementide üldine struktuur koos neid kirjeldava vajaliku informatsiooniga. Loodud andmestruktuuri hakati kutsuma vaate skeemaks.

Vaate skeema erinevate sektsioonide välja kuvamiseks loodi sektsioonidele vastavad vaate komponendid, mis võtavad sisse konkreetse sektsiooni tüübile vastava info vaate skeemast ning kuvavad selle põhjal skeemas kirjeldatud viisil välja vastavat vaate osist. Vaate skeema järgi vaate välja joonistamine algas juurkomponendist, mis kutsus rekursiivsel viisil mööda vaate skeema puud liikudes välja igale skeema sektsioonile vastava komponendi, mis omakorda sektsiooni välja kuvas. Antud lahendus osutus üsna võimekaks ning tõestas sektsioonipuu süsteemi toimimist erinevate vaadete ühisosade

kokku toomiseks ning viis autori hinnangul erinevate dokumendivaadete loomiseks vajaliku koodi mahu ja ajalise ressursi miinimumi.

```
def kuvaSektsioon(sektsioon):
   sektsiooniKomponent.kuva(sektsioon)
   if sektsioon.alamsektsioonid != null:
      for alamsektsioon in sektsioonid.alamSektsioonid:
          kuvaSektsioon(alamSektsioon)

kuvaSektsioon(sektsioonipuu.juurtipp)
```

Joonis 1: Sektsioonipuu põhjal vaate rekursiivselt vaatekomponentide abil kuvamise loogika pseudokood.

Kuna üheks tähtsaks eesmärgiks oli lisaks veel dokumendivaadete võimalikult kõrge paindlikkuse ja kohandatavuse saavutamine, oli oluline lisada juurde vaate skeemale iga sektsiooni juurde sektsiooni üldist häälestust kirjeldavad andmed nagu näiteks sektsiooni elementide järjekord, nende laiused, elementide nähtavus vaates (peidetud või mitte) ning palju muid sektsiooni spetsiifilisi andmeid. Sektsiooni ringi kohandamisel saatis eesrakendus uued kohandusandmed päringuga tagasi tagarakendusse, kus need andmebaasi salvestati. Vaate skeemast ja selles paiknevatest andmetest annab ülevaate bakalaureusetöö lisa 4.

Järgmise olulise etapina oli tähtis paika panna strateegia, kuidas dokumendivaade sektsioonide sisendväljades paiknevad ja dokumendi moodustavad vajalikud andmed tagarakenduse õigetest otspunktidest õigetel viisidel päriks ning kuidas toimuks nende kasutajale välja kuvamine ja muutmine. Dokumendi sektsioonides paiknevaid andmeid otsustati hoida eraldi vaate skeemast, kuna skeema ise on mahu poolest võrreldes sektsioonis kuvatavate andmetega võrdlemisi väike ning sel viisil oleks võimalik kasutajale välja kuvada korrektset vaate struktuuri koos laadimisanimatsiooniga kuniks mahukamate dokumentide korral mitmetesse megabaitidesse küündivaid sektsiooni andmeid päritakse, mis tagab meeldivama kasutuskogemuse ja väiksemad tajutavad laadimisajad [17] .

Vaate skeema igas sektsioonis oli vastavalt eelpool mainitule kirjeldatud ära sektsiooni andmete pärimiseks sektsiooni andmete lätte ID, mille eesrakendus välja luges ning vastavalt millele korrektne päring õigesse otspunkti tehti. Peale kõikide andmete kätte saamist peatati laadimisanimatsioonid ning kasutajale kuvati terviklikku vaadet.

Sektsiooni andmete muutmisel st dokumendi mustandi või dokumendi enda muutmisel saadeti tehtud muudatused tagarakendusse, kus läks käima vastav äriloogika. Kuna äriloogika jooksutamine võis võtta olenevalt dokumendist ja selle andmete ja äriloogika mahust märkimisväärselt palju aega, otsustati ooteaegade vähendamiseks ja vaate reageerimisvõime suurendamiseks esmalt muudatused sisse viia vaate enda olekuhaldussüsteemis talletatud andmete peal, pärast mida saadeti muudatused tagarakendusse, kus peale äriloogika protsesside lõppemist saadeti tagasi vastus, mis sisaldas teavet muutunud väljade ning nende uute väärtuste kohta. Antud lähenemine võimaldas täita eesmärki hoida võimalikult palju äriloogikat tagarakenduses pakkudes samas maksimaalset võimalikku vaate reageeritavust. Kirjeldatud protsessist annab ülevaate bakalaureusetöö lisa 5.

Kokkuvõttes disainiti dokumendi üldine andmevoog järgmiselt: lehekülje laadimise peale teostab eesrakendus esmalt vaate skeema päringu, peale mida kuvatakse skeema põhjal kasutajale dokumendivaade koos andmete laadimise animatsioonidega. Samal ajal kogub eesrakendus kokku kõikide sektsioonide andmete lättepunktide teabe ning teostab vastavad päringud asünkroonselt. Peale päringute vastuste kohale jõudmist peatatakse laadimisanimatsioonid ning kasutajale kuvatakse terviklikku dokumenti. Loodud süsteemi andmevoogu kirjeldab bakalaureusetöö lisa 6.

Lisaks eelpool mainitule kuulus dokumendivaadete juurde veel hulganisti lisaandmeid, nagu näiteks dokumendi ajaloo info, dokumendi metaandmed, dokumendi manuste info ja manuste failid, dokumentidega seotud sõnumid ning aktiivsed vea- ja hoiatusteated, mis oma üldises struktuuris päriti sarnaselt sektsiooni andmetele ning mida antud bakalaureusetöös detailsemalt ei kirjeldata.

4.6 Lahenduse realiseerimine

Peale põhjalikku analüüsi ja planeerimise faasi lõpule viimist alustati lahenduse realiseerimisega. Tarkvara arendusprotsess toimus vastavalt ettevõtte sisestele metoodikatele ning välja kujunenud protseduuridele, mida bakalaureusetöös lähtetingimustes kirjeldatud piirangute tõttu detailselt ei kirjeldata ega analüüsita.

Bakalaureusetöö autoril oli lahenduse realiseerimise juures tähtis roll ning ta juhtis arendusprotsessi vältel eesrakendust arendanud meeskonda. Valminud programmikoodi,

testide ning dokumentatsiooni mahust üle 80% oli bakalaureusetöö autori poolt kirjutatud ning sageli jäi suuremate arhitektuuriliste otsuste kooskõlastamine ning põhjendatult langetamine bakalaureusetöö autori kohustuseks.

Arendusprotsessi juures tegi eesrakendust loonud meeskond eesotsas bakalaureusetöö autoriga tihedat koostööd tagarakendust arendava meeskonnaga. Lisaks oli tihe koostöö ka erinevate arendusmeeskondadesse mitte kuuluvate spetsialistidega nagu kasutajaliidese disainerid ja tootejuhid. Arendusprotsess oli oma üldiselt olemuselt iteratiivne ning iga iteratsiooni lõpus andis autor valminud tarkvaraosistest, ettetulnud probleemidest ning tekkinud ettepanekutest koosolekute vältel teistele arendusprotsessi kaasatud isikutele ülevaate, kellega koostöös lahenduse olemust või käitumist korrigeeriti.

Arendusprotsess kujunes edukaks, seda bakalaureusetöö autori hinnangul ennekõike tänu läbimõeldud analüüsile, korrektsele arhitektuurile ning õigesti valitud tehnoloogiatele. Lisaks hindas bakalaureusetöö autor kõrgelt sujuvat suhtlust erinevate arendusmeeskondadesse mitte kuuluvate eriala spetsialistidega ning tõdes, et nende poolt pakutud nõu ja perspektiiv aitasid lahenduse realiseerimise vältel õiget kurssi hoida ning vältida tarkvaralahenduse lõppkasutajast kaugenemist.

Tarkvara välja töötamise ajaline raam pikenes olulisel määral võrreldes projekti alguses paika pandud hinnanguliste tähtaegadega, seda autori hinnangul ennekõike vajaka jäänud arendusressursi ja arendusprotsessi käigus lisandunud uute nõuete tõttu, mis nihutasid algselt seatud eesmärke. Lisaks mängis autori hinnangul olulist rolli teatud funktsionaalsuste realiseerimise keerukuse alahindamine planeerimise faasis.

Peale toimiva tarkvaralahenduse välja töötamise koos korrektse dokumentatsiooni ning automaattestidega ning arendusjuhi poolt heakskiidu saamist järgnes lahenduse kasutajatestimine valitud hulga äritarkvara klientide juures. Kasutajatestimise jooksul kogutud tagasiside põhjal tehti vastavad järeldused ning vajalikud muudatused loodud lahendusele. Üldises plaanis osutus kasutajatestimine väga edukaks ning märkimisväärseid probleeme ei esinenud. Kasutajate tagasiside oli igati konstruktiivne ning aitas enne loodud lahenduse laiemat evitamist saavutada kindlust selle toimimises.

5 Tulemused

Alljärgnevates peatükkides antakse ülevaade loodud lahendusest, saavutatud eesmärkidest, lahenduse üldisest kasust süsteemile tervikuna ning tuuakse välja retrospektiiv, milles autor analüüsib arendusprotsessi ning toob välja aspektid, mida tagasivaadatates oleks võinud lahendada teistmoodi.

5.1 Loodud lahendus

Valminud eesrakendus koos ühtse dokumendivaadete kuvamise mootoriga viis erinevate dokumendivaadete arenduseks ja haldamiseks kuluva ressursi miinimumini, kuna võimaldas sama koodi ja samade tarkvaraosistega kuvada välja mõningaste mööndustega kõiki äritarkvara dokumendivaateid. Kui varem oli iga dokumendivaate välja kuvamiseks vajalik olenevalt vaatest mitu sada kuni mitu tuhat rida koodi, mis viis kogu äritarkvara dokumendivaateid kuvava koodi mahu mitmekümnetesse tuhandetesse, siis uus süsteem vähendas koodi mahtu vähemalt kümme korda pakkudes seejuures olulisel määral uusi funktsionaalsusi. Kuna koodi maht on reeglina võrdelises suhtes selle haldamiseks kuluva ressursiga, võib väita, et uus süsteem vähendas haldamisele kuluvat ressurssi samuti mitmetes kordades. Lisaks vähendab ühtne süsteem tulevikus kõiki dokumendivaateid puudutavate muudatuste realiseerimiseks vajalikku ressurssi, kuna muudatused tuleb sisse viia ainult ühes kohas ühel ühtsel süsteemil.

Loodud lahenduses realiseeritud lisafunktsionaalsus võimaldas pakkuda äritarkvara laiale kasutuskonnale märkimisväärselt paremat kasutuskogemust läbi uute võimaluste ja mugava kohandatavuse, mis omakorda võimaldas erinevatel ettevõtetel kujundada dokumendivaateid vastavalt nende vajadustele ning aitas seeläbi optimeerida nende töövooge, mis omakorda mõjus positiivselt tarkvara üldisele konkurentsivõimele turul.

Olemasolevad funktsionaalsused ja töövood kaardistati ning realiseeriti uues lahenduses, mis aitas ära hoida kasutajate töövoogude häirimise märkamata jäänud kasutusvoogude või realiseerimata jäänud funktsionaalsuste tõttu, mis tähendab, et

ülemineku protsess vanadelt vaadetelt uutele oli kasutajate jaoks võimalikult sujuv ning programmi üldine kasutatavus üheski valdkonnas märgatavalt ei vähenenud.

Välja töötatud sektsioonipuu ja vaate skeema süsteem võimaldas täita eesmärgi pakkuda kasutajatele vastavalt nende nõudmistele võimaluse korral tellimustööna kohandatud vaateid ja äriloogikat. Lisaks võimaldasid antud lahendused viia äriloogika eesrakendusest välja, mis omakorda täitis eesmärgi hoida võimalikult suurel määral rakenduse äriloogikat tagarakenduses.

Tänu läbimõeldud automaattestimise kavale ning ulatuslikule testide kaetavusele suurendati programmi veakindlust ning tulevikus loodavate arenduste kiirust tänu vähendatud riskile regressioonivigade tekkimiseks. Tulevikuarendusteks kuluvat ressurssi aitas lisaks vähendada lahendusele loodud dokumentatsioon, mis võrreldes pärandsüsteemiga oli palju detailsem ja sisaldas rohkem vajalikku teavet.

Loodud lahendus kasutas läbimõeldud arhitektuuri ning modernse tarkvaraarenduse parimaid praktikaid ja tehnoloogiaid, mis tagas projekti jätkusuutlikkuse tulevikus. Lisaks võimaldas uus arhitektuur ja kasutatud tehnoloogiad paremat jõudlust ning aitas välja kuvada suure andmemahuga vaateid, mida varem polnud võimalik näidata, kuid mille kasutamine on teatud klientide jaoks olulise tähtsusega.

Valminud dokumendivaadete mootori baasil on töö kirjutamise hetkel loodud kaks terviklikku dokumendivaadet, mis on läbinud kasutajatestimise ning saanud heakskiidu ettevõtte juhtkonnalt ning on päris klientide juures tarkvara orgaanilise osana kasutusel. Lisaks on töö kirjutamise hetkel valmimas veel kuus dokumendivaadet.

Negatiivsete külgedena soovib autor välja tuua asjaolu, et uus lahendus suurendas üle võrgu kantava andmemahu hulka, mis omakorda vähendas programmi vaadete laadimise kiirust. Tajutava laadimisaja vähendamiseks sai dokumendivaadetele disainitud laadimisanimatsioonid, mis toimisid autori hinnangul efektiivselt ning täitsid oma eesmärki, millest tulenevalt rakenduse tajutav reageeritavus ja laadimise kiirus autori hinnangul oluliselt määral ei kannatanud.

Kuigi loodud lahenduse välja töötamise protsess ületas olulisel määral sellele analüüsi etapis seatud ajalisi tähtaegasid, suudeti siiski kõik projektile seatud eesmärgid autori

hinnangul rahuldaval või väga heal tasemel täita ning üldises plaanis tõstis loodud lahendus märkimisväärselt äritarkvara kui terviku kasutatavust, veakindlust, konkurentsivõimet ning vähendas edasiseks arenduseks kuluvat ressurssi. Kuna lahendus on töö kirjutamise hetkel kasutuses olnud vaid loetud kuud ning kõik dokumendivaated ei ole veel üle viidud, selgub täielik saavutatud kasu koos muude kõrvalmõjudega alles tulevikus.

5.2 Projekti tulevik

Loodud lahenduse arendus kestab hinnanguliselt veel mitu aastat ning järk järgult hakatakse pärandsüsteemi dokumendivaateid välja vahetama valminud süsteemil põhinevate uuema põlvkonna dokumendivaadetega, mistõttu üle toodud pärandsüsteemi vaadete aktiivne arendamine peatatakse.

Loodud lahenduse perspektiivikust analüüsitakse võttes arvesse arenduse faasi jooksul ette tulnud võimalikke uusi tulevikusuundi, probleeme ning kitsaskohti, millele tuginedes viiakse sisse vastavad süsteemsed muutused eesmärgiga saavutada kõrgem tulevikukindlus.

Edasise arendustsükli jooksul pannakse suurt rõhku loodud lahenduse jõudluse testimisele ning optimeerimisele, kuna vaadete jõudlus on lõppkasutajatele antud kontekstis väga tähtis ning bakalaureusetöö autor leiab, et loodud lahendust saaks oluliselt optimeerida nii vaate laadimise kiiruse kui ka üldise reageeritavuse osas, mida ajalistest piirangutest tingituna töö kirjutamise hetkel teha ei olnud võimalik.

Loodud lahendus andis tõestust äritarkvara üldise uue hajutatud süsteemidel põhineva arhitektuuri toimimise kohta ning valminud dokumendivaadete mootori näitel hakatakse tulevikus looma ka aruannete kuvamise mootorit, mis on äritarkvara teiseks suureks osiseks.

Iga uuele süsteemile üle toodud dokumendivaade suurendab loodud süsteemi haldamiseks ning edasi arendamiseks kuluvat ressurssi, mistõttu on oluline laiendada tulevikus uut süsteemi arendavat meeskonda.

Tänu projekti jooksul rakendatud kaasaaegsetele arhitektuurilistele mustritele ning sobivalt valitud tehnoloogiatele on bakalaureusetöö autori hinnangul projekti tuleviku kindlus süsteemse poole pealt kõrge ning hetkel selles valdkonnas tõenäolisi probleeme tulevikus ei nähta.

Loodud lahenduse edasise arenduse kiiruse ja veakindluse tõstmiseks on plaanis läbi rääkida ja võimalusel läbi viia lahenduse võimalik üle toomine praeguselt JavaScript keelelt TypeScript keelele, mida antud töö kirjutamise hetkel erinevatest töös mitte käsitletavatest põhjustest tingituna veel tehtud pole, kuid millest tulenev võimalik kasu on arendusmeeskonna hinnangul märkimisväärne.

5.3 Retrospektiiv

Loodud lahenduse analüüsi, planeerimise ning lahenduse realiseerimise faasidele tagasi vaadates oli bakalaureusetöö autori hinnangul valitud metoodika igati õigustatud ning korrektselt valitud, lisaks sujus lahenduse loomine ilma suuremate probleemideta ning langetatud arhitektuurilisi ja tehnoloogilisi otsuseid hiljem tagantjärele muuta ei olnud vaja.

Küll aga oleks autori hinnangul saanud paremini planeerida projektile kuluvat ajalist ressurssi ning anda täpsemaid hinnanguid tähtaegade tõsielulisusele, seda ennekõike lahenduse realiseerimise faasis, mille jooksul oli autori hinnangul sageli hetki, kus suunati ressurssi rohkem uute harva kasutatavate funktsionaalsuste realiseerimisele põhifunktsionaalsuste loomise asemel, mistõttu pikenes märgatavalt projekti valmimisele seatud esialgne ajaline tähtaeg. Antud viga oleks autori hinnangul aidanud ära hoida kasutajalugude sügavam ja täpsem prioritiseerimine ning töötlemine, mis oleks võimaldanud selgitada välja erinevate funktsionaalsuste realistlikuma ajalise hinna ning pakutava väärtuse suhte.

Lisaks oleks autori hinnangul võinud kasutajatestimise faasis testida loodud lahendust suurema kasutajaskonna peal ning valida kliente, kelle ärilised vajadused ja nendest tulenevad kasutusvood oleksid olnud rohkem üksteisest erinevad. Projekti jooksul valitud kasutajaskond jäi autori hinnangul liiga kitsaks ning kasutajate üldine aktiivsus lahenduse testimisel ja tagasiside andmisel oleks võinud olla parem.

Kasutajanõuete kaardistamisel ning olemasolevate funktsionaalsuste kasutatavuse analüüsil oleks autori hinnangul võinud teha suuremat koostööd äritarkvara päris klientidega ning uurida sügavamalt nende igapäevasel äritarkvara kasutamisel päriselt ettetulevaid probleeme. Kuigi antud protsess ei kuulunud autori vastutuste hulka ning seda bakalaureusetöös pikemalt ei käsitletud, leiab autor siiski, et mainitud protsess oleks võinud olla parem, mis omakorda oleks aidanud lahenduse loomisel nõudeid paremini prioritiseerida.

Autori hinnangul oleks projekti planeerimise faasis välja selgitatud keerukuse ja mahu tõttu võinud projekti realiseerimisel kaaluda ning võimalusel kasutada JavaScript keele asemel TypeScript keelt, mis pakub JavaScriptiga võrreldes staatilist tüüpimist ning tüübiturvalisust, mis oleks autori hinnangul aidanud vältida arenduse faasis ette tulnud dünaamilisest tüüpimisest tulenevaid komistuskive ning toota veakindlamat koodi.

Viimase mainimisväärse aspektina leiab autor, et projekti realiseerimise faasi lõpu poole avastatud ressursi puudujäägi kompenseerimiseks suunatud lisa inimjõud ei tõstnud olulisel määral projekti arenduse kiirust ning võis isegi omada negatiivset efekti lisandunud abijõu koolitamise ja haldamisega kaasnevate kuludele tõttu. Autori hinnangul oleks saanud projekti planeerimise faasis hinnata täpsemalt projekti realiseerimisele kuluvat tööjõudu ning vajalik ressurss oleks tulnud eraldada enne projekti realiseerimise faasi alustamist.

Kokkuvõte

Käesoleva bakalaureusetöö põhieesmärgiks oli luua Directo äritarkvara kaasajastamise tarbeks veebipõhine eesrakendus, mis suudaks kuvada kõiki äritarkvara dokumendivaateid kasutades seejuures võimalikult suurel määral ühtset loogikat ja programmikoodi ning mis tõstaks võrreldes pärandsüsteemiga programmi veakindlust ja hallatavust ning langetaks dokumendivaadete arendusele kuluvat ressurssi.

Töö jooksul kaardistati pärandsüsteemi funktsionaalsus, kasutatavuse ja turvalisuse probleemid, koguti kokku kasutajanõuded ning nende põhjal viidi läbi analüüsid, mille tulemuste põhjal valiti projekti jaoks tarvilikud tehnoloogiad ning langetati arhitektuurilised otsused. Järgnes arenduse faas, mille alguses pandi paika loodava süsteemi automaattestimise ja dokumenteerimise kava ning peale mida alustati süsteemi loomist. Toimiva lahenduse välja töötamisele järgnes kasutajatestimise faas, peale mida loodud lahendus evitati ning see sai Directo äritarkvara orgaaniliseks osaks.

Loodud lahendus täitis autori hinnangul rahuldaval või väga heal tasemel kõik projektile seatud eesmärgid ning lahendas püstitatud probleemid. Projekti tulemusena töötati välja React raamistikul põhinev ühelehe veebirakendus äritarkvara dokumendivaadete ühtsel süsteemil alusel kuvamiseks. Loodud lahendus vähendas tarkvara dokumendivaadete koodi mahtu üle kümne korra ning koos sellega vähendas ka haldamiseks ning edasiseks arenduseks kuluvat ressurssi. Lisaks pakkus loodud eesrakendus pärandsüsteemiga võrreldes paremat veakindlust ning võimaldas kasutajatele funktsionaalsust, mida pärandsüsteemis polnud võimalik realiseerida, mis omakorda suurendas äritarkvara konkurentsivõimet turul. Töö kirjutamise hetkel on väljatöötatud lahenduse baasil loodud eesrakendusse kaks dokumendivaadet, mis on äritarkvara orgaaniliste osadena tarkvara klientide käes kasutusel ning arenduses on lisaks veel kuus uut dokumendivaadet.

Loodud lahendus võimaldab pakkuda klientidele oluliselt paindlikumat kasutajakogemust ning lubas klientidel erinevaid dokumendivaateid kohandada

vastavalt nende ärilistele vajadustele, mis omakorda tõstis veelgi pakutava toote konkurentsivõimet.

Projekti arendus jätkub tulevikus ning plaanis on tuua loodud süsteemile üle kõik äritarkvara praegused dokumendivaated. Tähtsamate edasiste arendussuundadena keskendutakse loodud süsteemi jõudluse parendamisele ning kasutajate tagasiside kogumisele ja selle põhjal edasisele kasutatavuse parendamisele.

Kasutatud kirjandus

[1] S. Ivanova, G. Georgiev, “Using modern web frameworks when developing an education application: a practical approach”, uurimus, Institute of Electrical and Electronics Engineers (IEEE), 2019, https://ieeexplore.ieee.org/abstract/document/8756914, vaadatud 27.11.2020.

[2] A. Stellmann, “Requirements 101: User Stories vs. Use Cases”, O'Reilly autorite J. Greene ja A. Stellman ametlik veebilehekülg, 2009, https://www.stellman-greene.com/2009/05/03/requirements-101-user-stories-vs-use-cases/, vaadatud 07.11.2021.

[3] J. Nielsen, A. Kaley, “Opening Links in New Browser Windows and Tabs”, Nielsen Norman Group akadeemilised artiklid kasutajaliidese disainist, 2020, https://www.nngroup.com/articles/new-browser-windows-and-tabs/, vaadatud 07.11.2021

[4] “Project web security testing guide”, sihtasutus OWASP, 2021, https://owasp.org/www-project-web-security-testing-guide/stable/, vaadatud 10.11.2021.

[5] S. Kristen, “Cross site scripting”, sihtasutus OWASP, 2021, https://owasp.org/www-community/attacks/xss/, vaadatud 10.11.2021.

[6] “2021 Developer Survey”, veebilehe Stackoverflow meeskonna poolt läbiviidud uurimus tarkvaraarenduse tehnoloogiate populaarsuse ja kasutamise osas aastal 2021, 2021, https://insights.stackoverflow.com/survey/2021, vaadatud 14.11.2021.

[7] “2020 Developer Survey, veebilehe Stackoverflow meeskonna poolt 2020 aastal läbiviidud uurimus tarkvaraarenduse tehnoloogiate populaarsuse ja kasutamise osas, 2020, https://insights.stackoverflow.com/survey/2020, vaadatud 14.11.2021.

[8] “Developer Survey Results 2019 ”, veebilehe Stackoverflow meeskonna poolt 2019 aastal läbiviidud uurimus tarkvaraarenduse tehnoloogiate populaarsuse ja kasutamise osas, 2019, https://insights.stackoverflow.com/survey/2019, vaadatud 14.11.2021.

[9] Express raamistiku ametlik dokumentatsioon, 2021, https://expressjs.com/, vaadatud 14.11.2021.

[10] E. Saks, “JavaScript frameworks: Angular vs React vs Vue”, bakalaureusetöö, Haaga-Helia ülikool, Soome, 2019, https://www.theseus.fi/bitstream/handle/10024/261970/Thesis-Elar-Saks.pdf?sequence=2, vaadatud 14.11.2021.

[11] “React Context”, React raamistiku ametlik dokumentatsioon, 2021, https://reactjs.org/docs/context.html, vaadatud 19.11.2021.

[12] M. Erikson, “Why React Context is Not a "State Management" Tool (and Why It Doesn't Replace Redux)”, Redux tarkvarateegi haldaja Mark Eriksoni arendusblogi, 2021, https://blog.isquaredsoftware.com/2021/01/context-redux-differences/, vaadatud 19.11.2021.

[13] Redux'i ametlik koodirepositoorium, GitHub, https://github.com/reduxjs/redux, vaadatud 20.11.2021.

[14] Redux'i ametlik dokumentatsioon, https://redux.js.org/, vaadatud 20.11.2021.

[15] E.M Maximilien, L. Williams, "Assessing test-driven development at IBM", juhtumiuurimus, Institute of Electrical and Electronics Engineers (IEEE), 2003, https://ieeexplore.ieee.org/abstract/document/1201238, vaadatud 22.11.2021.

[16] "Components and props", React raamistiku ametlik dokumentatsioon, 2021, https://reactjs.org/docs/components-and-props.html, vaadatud 23.11.2021.

[17] K. Sherwin, "Progress Indicators Make a Slow System Less Insufferable", Nielsen Norman Group akadeemilised artiklid kasutajaliidese disainist, 2014, https://www.nngroup.com/articles/progress-indicators/, vaadatud 24.11.2021.

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

Mina, Ranno Rajaste

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Äritarkvara Directo dokumendisüsteemi veebipõhise eesrakenduse loomine" mille juhendaja on Meelis Antoi

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

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

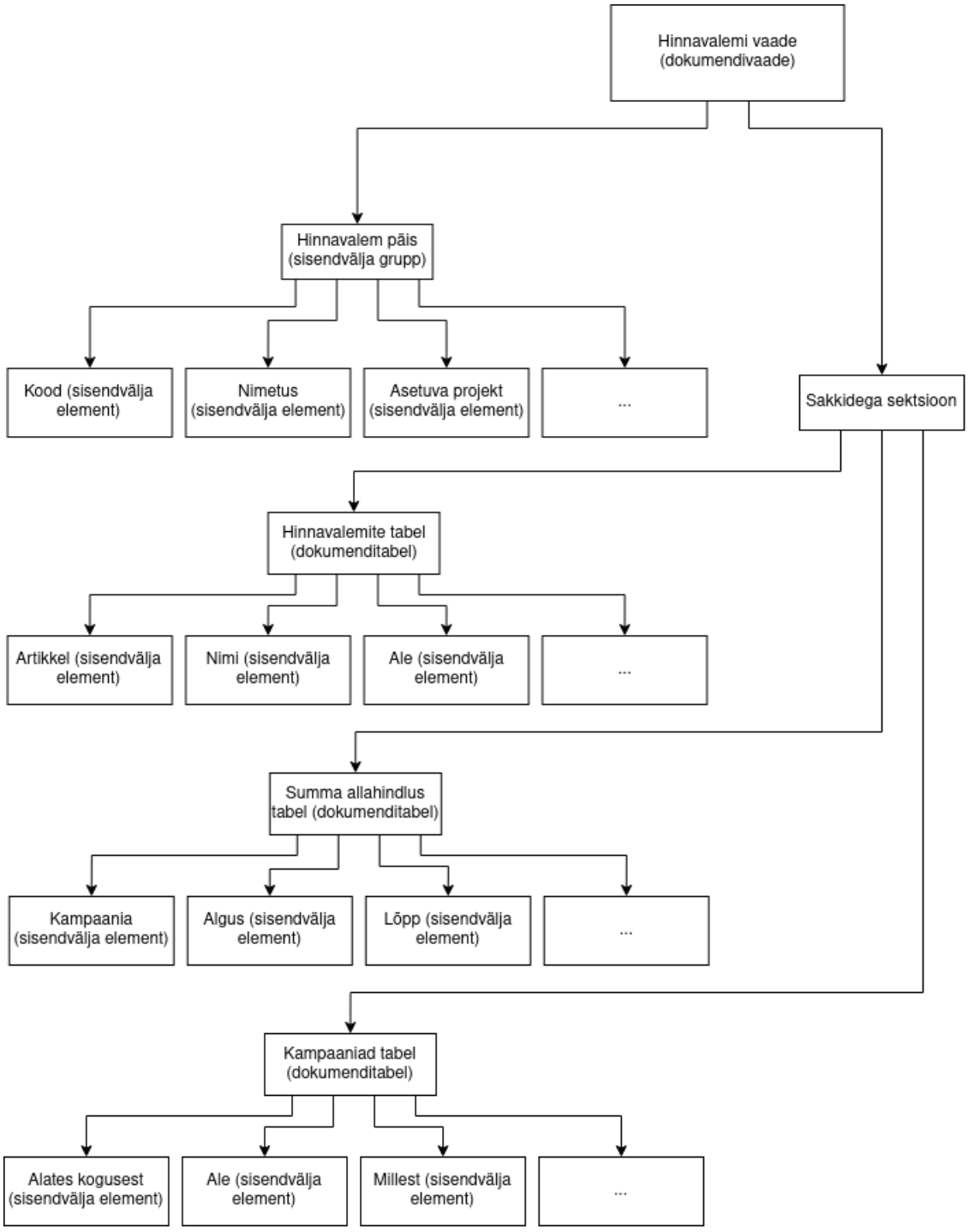
2 Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

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

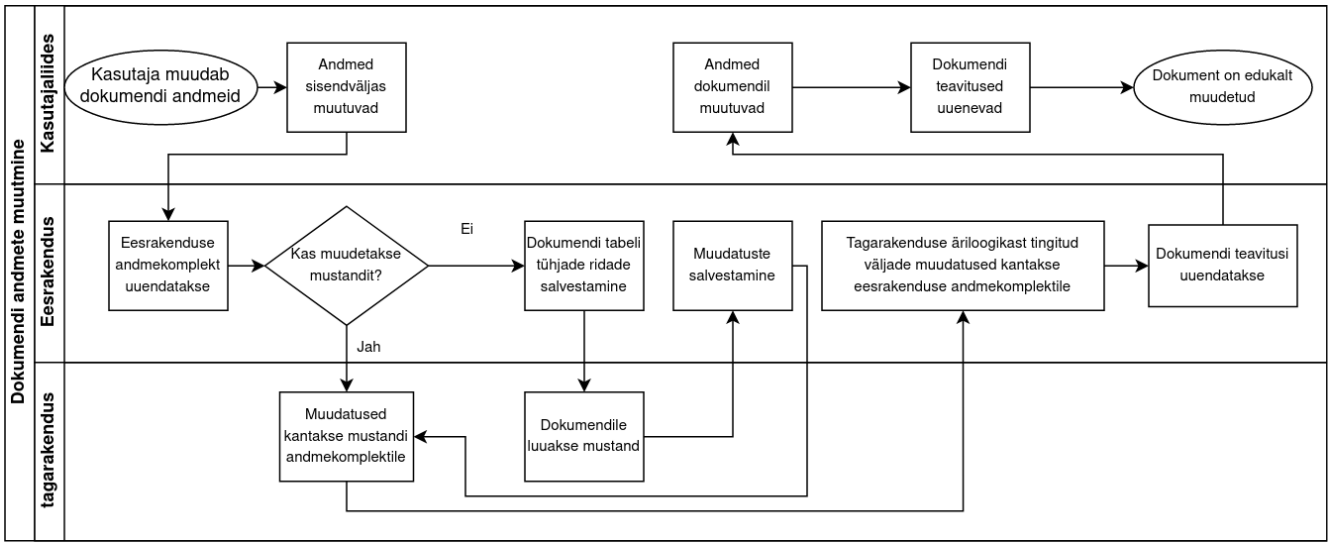
07.01.2022

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 – „Hinnavalem” dokumendivaate sektsioonipuu süsteemi abil kujutamine (lihtsustatud)



Lisa 3 – Mustandisüsteemi loogika skeem dokumendi muutmise korral



Lisa 4 – “Hinnavalem” vaate skeema näide[1]

```
{
    "sections": [
      {
        "id": "inputs_document",
        "type": "input_group",
        "label": "",
        "dataSourceInfo": {
          "id": "priceformula_header",
          "type": "single"
        },
        "collapsible": false,
        "elements": [
          {
            "id": "code",
            "type": "primary_text",
            "size": "medium",
            "label": "Kood",
            "maxTextLength": 32,
            "minTextLength": null,
            "editable": false,
            "hideable": false,
            "visible": true,
            "freezeState": 0,
            "filterType": 0
          },
          {
            "id": "name",
            "type": "text",
            "size": "medium",
            "label": "Nimi",
            "maxTextLength": 255,
            "minTextLength": null,
            "editable": true,
            "hideable": true,
            "visible": true,
            "freezeState": 0,
            "filterType": 0,
            "width": 0
          },
          {
            "id": "object",
            "type": "search_text",
            "size": "medium",
            "label": "Asetuv objekt",
            "registry": "object",
            "maxTextLength": 255,
            "minTextLength": null,
            "editable": true,
```

1 Näites on ruumi kokkuhoiu eesmärgil iga sektsiooni all kujutatud maksimaalselt kolme sektsiooni elementi.

```
      "hideable": true,
      "visible": true,
      "freezeState": 0,
      "filterType": 0,
      "width": 0
    }
  ],
  "actions": [],
  "collapsed": false,
  "activeTabId": null,
  "lineBreakIndexes": [5]
},
{
  "id": "document_rows",
  "type": "tabbed_group",
  "label": "",
  "collapsible": false,
  "childrenSections": [
    {
      "id": "table_priceformulas",
      "type": "table",
      "label": "Valemid",
      "dataSourceInfo": {
        "id": "priceformula_rows",
        "type": "rows"
      },
      "collapsible": false,
      "elements": [
        {
          "id": "item_class",
          "type": "search_text",
          "label": "Klass",
          "registry": "item_class",
          "maxTextLength": 32,
          "minTextLength": null,
          "editable": true,
          "hideable": true,
          "visible": true,
          "freezeState": 0,
          "filterType": 0,
          "width": 67,
          "shortcut": false
        },
        {
          "id": "item",
          "type": "search_text",
          "label": "Artikkel",
          "registry": "item",
          "maxTextLength": 32,
          "minTextLength": null,
          "editable": true,
          "hideable": true,
          "visible": true,
          "freezeState": 0,
          "filterType": 0,
          "width": 0,
          "shortcut": false
        },
        {
          "id": "item_datafield",
```

```
        "type": "search_text",
        "label": "Art.lisaväli",
        "registry": "extra_field",
        "constraints": [
          {
            "type": 1,
            "constraintParams": {
              "klass": "Artikkel"
            }
          }
        ],
        "maxTextLength": 32,
        "minTextLength": null,
        "editable": true,
        "hideable": true,
        "visible": false,
        "freezeState": 0,
        "filterType": 0,
        "width": 144,
        "shortcut": false
      }
    ],
    "actions": [
      {
        "id": "bulk_placer_action",
        "type": "bulk_placer_action",
        "label": "Massasetaja"
      },
      {
        "id": "add_all_classes_action",
        "type": "custom_action_mutate",
        "label": "Lisa kõik klassid"
      },
      {
        "id": "add_all_in_use_classes_action",
        "type": "custom_action_mutate",
        "label": "Lisa kõik kasutuses klassid"
      },
      {
        "id": "copy_from_first_row",
        "type": "custom_action_mutate",
        "label": "Kopeeri esimeselt realt"
      }
    ],
    "requiredFieldsToSaveRow": [
      "item",
      "item_class",
      "supplier",
      "item_datafield",
      "item_datafield_value"
    ],
    "viewDensity": 0
  },
  {
    "id": "sum_discounts",
    "type": "table",
    "label": "Summa allahindlus",
    "dataSourceInfo": {
      "id": "priceformula_sum_discounts_rows",
      "type": "rows"
```

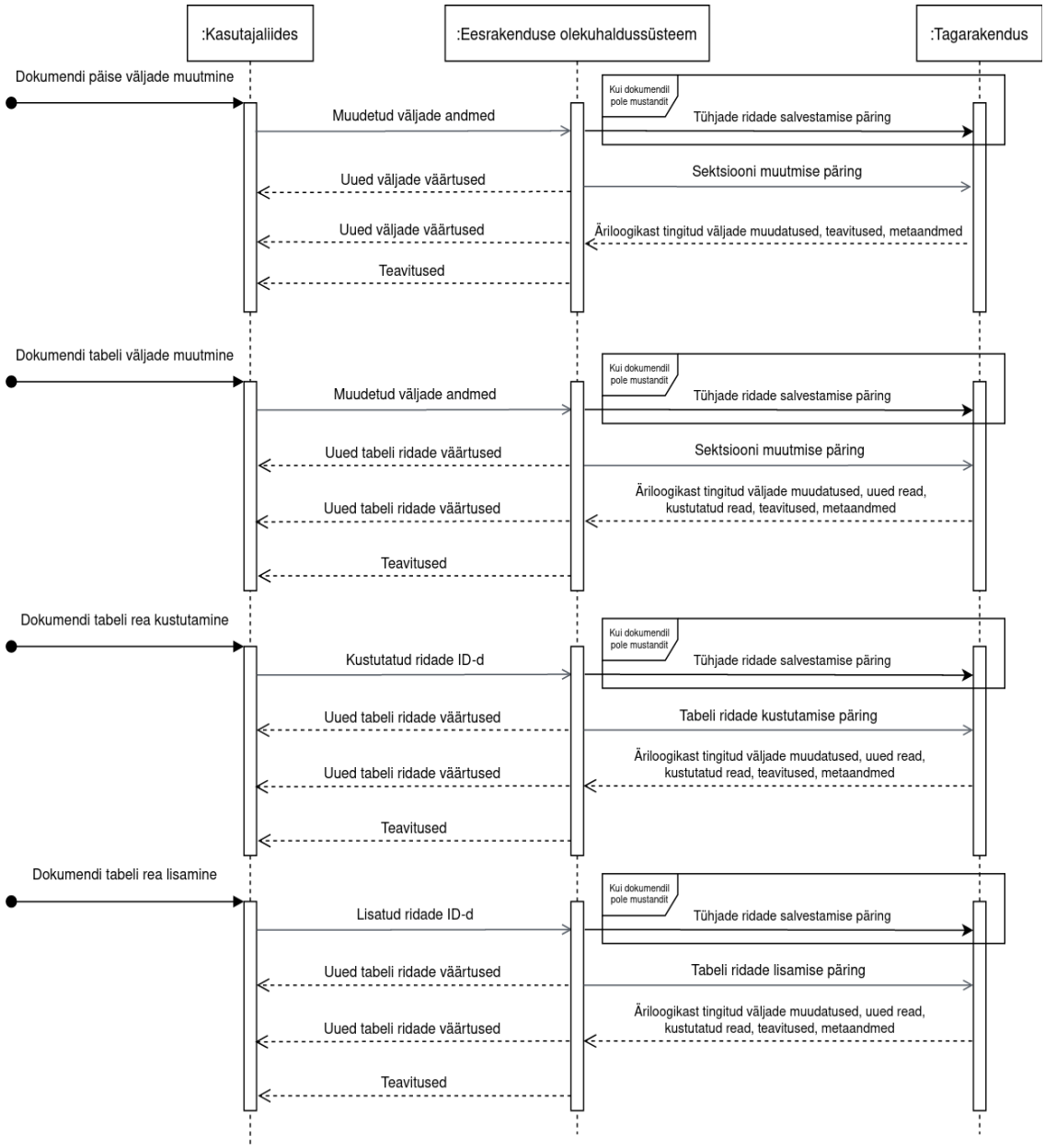
```
},
"collapsible": false,
"childrenSections": [],
"elements": [
  {
    "id": "from_what",
    "type": "select_text",
    "label": "Millest",
    "editable": true,
    "hideable": true,
    "options": [
      {
        "value": "",
        "label": ""
      },
      {
        "value": "kokku",
        "label": "Kokku"
      },
      {
        "value": "tasuda",
        "label": "Tasuda"
      }
    ],
    "visible": true,
    "freezeState": 0,
    "filterType": 0,
    "width": 0,
    "shortcut": false
  },
  {
    "id": "from_sum",
    "type": "number",
    "label": "Alates",
    "precision": 4,
    "min": -1000,
    "max": 1000,
    "editable": true,
    "hideable": true,
    "visible": true,
    "freezeState": 0,
    "filterType": 0,
    "width": 0,
    "shortcut": false
  }
],
"actions": [
  {
    "id": "bulk_placer_action",
    "type": "bulk_placer_action",
    "label": "Massasetaja",
    "confirmationMessage": "",
    "cancelButtonText": "",
    "confirmButtonText": "",
    "disabledWhenStateIs": null,
    "tooltip": "",
    "tooltipDisabled": "",
    "color": null,
    "confirmIsDangerous": true,
    "params": null
```

```
      }
    ],
    "requiredFieldsToSaveRow": [
      "from_what",
      "from_sum",
      "affects",
      "date_start",
      "date_end",
      "time1",
      "time2",
      "project"
    ],
    "viewDensity": null,
    "collapsed": null,
    "activeTabId": null,
    "sectionWidth": null,
    "lineBreakIndexes": null
  },
  {
    "id": "campaigns",
    "type": "table",
    "label": "Kampaaniad",
    "dataSourceInfo": {
      "id": "priceformula_campaigns_rows",
      "type": "rows"
    },
    "collapsible": false,
    "elements": [
      {
        "id": "campaign",
        "type": "search_text",
        "label": "Kampaania",
        "registry": "campaign",
        "maxTextLength": 32,
        "minTextLength": null,
        "editable": true,
        "hideable": true,
        "visible": true,
        "freezeState": 0,
        "filterType": 0,
        "width": 0,
        "shortcut": false
      },
      {
        "id": "campaign_name",
        "type": "text",
        "label": "Nimi",
        "maxTextLength": 255,
        "minTextLength": null,
        "editable": false,
        "hideable": true,
        "visible": true,
        "freezeState": 0,
        "filterType": 0,
        "width": 0,
        "shortcut": false
      },
      {
        "id": "project",
        "type": "search_text",
```

```
            "label": "Projekt",
            "registry": "project",
            "maxTextLength": 32,
            "minTextLength": null,
            "editable": true,
            "hideable": true,
            "visible": true,
            "freezeState": 0,
            "filterType": 0,
            "width": 0,
            "shortcut": false
          }
        ],
        "actions": [
          {
            "id": "bulk_placer_action",
            "type": "bulk_placer_action",
            "label": "Massasetaja"
          }
        ],
        "requiredFieldsToSaveRow": [
          "campaign",
          "date_start",
          "date_end",
          "time1",
          "time2",
          "project"
        ]
      }
    ],
    "collapsed": false,
    "activeTabId": "table_priceformulas"
  }
],
"toolbarContent": [
  {
    "id": "new_action",
    "type": "new_action",
    "label": "Uus"
  },
  {
    "id": "save_action",
    "type": "save_action",
    "label": "Salvesta"
  },
  {
    "id": "copy_action",
    "type": "copy_action",
    "label": "Kopeeri"
  },
  {
    "id": "discard_action",
    "type": "discard_action",
    "label": "Jäta"
  },
  {
    "id": "delete_action",
    "type": "delete_action",
    "label": "Kustuta"
  },
```

```
    {
      "id": "attachments_action",
      "type": "attachments_action",
      "label": "Manused",
      "params": {
        "unit": "hinnavalem"
      }
    },
    {
      "id": "settings_action",
      "type": "settings_action",
      "label": "Seaded"
    }
  ],
  "sidebar": {
    "messages": null,
    "attachments": null,
    "historyInfo": null,
    "width": 26.83,
    "visible": true
  }
}
```

Lisa 5 – dokumendil kuvatavate andmete muutmise protsessi skeemi katkend



Lisa 6 – dokumendivaate mootori andmevoo skeemi katkend