TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Ranar Leisson 206545IADB

Telia Eesti ASi tellimuste töötlemiste süsteemi tehnilise kasutajaliidese moderniseerimine

Bakalaureusetöö

Juhendaja: Meelis Antoi
Magistrikraad
Kaasjuhendaja: Rain Vink
Magistrikraad

Tallinn 2023

Autorideklaratsioon

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

Autor: Ranar Leisson

15.05.2023

Annotatsioon

Antud bakalaureusetöö eesmärk on leida lahendus Telia Eesti AS tellimuste töötlemise rakenduse tehniline kasutajaliides ajakohastamiseks ning realiseerida see. Olemasolevas kasutajaliideses on mitme aastaga kujunenud tehniline võlg, mis pärsib rakenduse hooldamist ja suuremate arhitektuuriliste üleminekute teostamist tellimuste töötlemise süsteemis kui tervikus.

Eesmärgi saavutamiseks teostas autor olemasolevale kasutajaliidesele analüüsi, et välja selgitada selle funktsionaalsus, liidestused äriloogika kihi serverrakendustega, suurimad puudused ja tehnoloogiline seisukord. Nende tulemuste põhjal otsustas autor täiesti uue rakenduse loomise kasuks.

Arendusprotsessi käigus loodi uue tehnilise kasutajaliidese MVP (*Minimal Viable Product*) veebirakenduse kujul, mis pakub väikest alamhulka kogu funktsionaalsusest.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 7 peatükki, 12 joonist, 3 tabelit.

Abstract

Modernizing of Telia Eesti AS Order Processing System Technical User Interface

The goal of this thesis is to find a solution to update the technical interface of the order processing application of Telia Eesti AS and to implement it. The existing user interface has accumulated a large technical debt over several years, which hinders the maintenance of the application and the implementation of major architectural changes in the order processing system as a whole.

In order to achieve the goal, the author carried out an analysis on the existing technical interface to identify its functionality, interfaces with business logic layer server applications, major shortcomings and technological state. On the basis of these results, the author decided to create a completely new application.

During the development process, an MVP (*Minimal Viable Product*) of a new technical user interface was created in the form of a web application that provides a small subset of the overall functionality.

The thesis is in Estonian and contains 30 pages of text, 7 chapters, 12 figures, 3 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakenduse programmerimise liides
Active Directory	Microsofti loodud tsentraliseeritud kataloogiteenus autentimise ja autoriseerimise haldamiseks.
BFF	*Backend for frontend*, arhitektuurne muster, mis hõlmab iga klientrakenduse jaoks eraldi serverrakenduse arendamist
bundle	Optimeeritud ja tihendatud JavaScript koodifailide kogumik
CD	*Continuous deployment*, pidev paigaldus
CI	*Continuous integration*, pidev integratsioon
controller	Tarkvarakomponent, mis käsitleb ja haldab API päringuid ja vastuseid.
CSS	*Cascading Style Sheets*, märgistuskeel, mida kasutatakse veebilehe kujunduse kirjeldamiseks
DDD	*Domain Driven Design*, tarkvara projekteerimise lähenemisviis, kus tarkvara kujundatakse valdkonnale vastavalt
DTO	*Data Transfer Object*, objekt andmete liigutamiseks süsteemi erinevate kihtide vahel
full-stack	mõiste kirjeldamaks veebirakenduse lahendust, mis hõlmab endas nii klient- kui serverrakendust
HTML	*HyperText Markup Language*, märgistuskeel veebidokumentide ülesehituse kirjeldamiseks
JHipster	Avatud lähtekoodiga veebirakenduste arendusplatvorm, mis kasutab Spring Boot'i
JWT	*Json Web Token*, avatud standard teabe turvaliseks edastamiseks JSON-objektina

LTS	*Long Term Support*, tarkvara või riistvara pikaajalise toe versioon
MVC	*Model-View-Controller*
MVP	*Minimum Viable Product*, vähim elujõuline toode
NPM	*Node Package Manager*, käsureapõhine paketihaldur JavaScriptile
OWASP	*The Open Worldwide Application Security Project*
pod	Kuberneteses, pod on üks jooksva konteineri instants klastris.
REST	*Representational state transfer,* veebirakenduste suhtlusstandard
SPA	*Single Page Application*, üheleheline veebirakendus, mille sisu dünaamiliselt laetav sisu on navigeeritav ilma lehe täieliku värskendamiseta
SSO	*Single sign-on*, autentimismehhanism, mis võimaldab mitme rakenduse kasutamist ühe kasutaja kaudu
TMForum	*TeleManagement Forum*, telekommunikatsioonieттevõtete tööstusliit, mis töötab välja digitaalteenuste standardeid
ToTeKa	*Toodete ja Tellimuste Kataloog*, Telia Eesti tellimuste töötlemise süsteemi nimi

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tellimuste töötlemise süsteem on tooteid ja teenuseid digitaalselt vahendavale ettevõttele üks olulisemaid osi, kuna see juhib klientide tellimuste täitmise protsessi kui tervikut. Selline süsteem hõlmab hulka tihedalt integreeritud rakendusi, mis omavahel koordineerides automatiseerivad tellimuste haldamise protsessi erinevaid aspekte, näiteks toote varude haldamist, saatmist, arveldamist ja kliendisuhtlust. [1]

Turul on palju ettevõtteid, mis pakuvad sellist süsteemi kui terviklahendust teenusena, nagu näiteks IBM [2], Ericsson [3] ja Shopify [4]. Kuid antud lahendused kas ei suuda täita Telia Eesti AS (edaspidi Telia) praegusi ärilisi nõudmisi või on majanduslikult ebapraktilised. Telia praegune eesmärk on välja arendada vajadustele kohandatud süsteem ettevõttesiseselt.

Telias eelnimetatud süsteemi üheks osaks on tellimuste töötlemise rakendus, mis haldab tellimust, tellimuse komponente ning nende olekuid. Selleks, et arendajad, peakasutajad ja teenindustugi saaksid kiirelt ja mugavalt tellimusi hallata on loodud rakendusele oma haldusliides, mis on käesoleva bakalaureusetöö põhifookuses.

Lõputöö eesmärk on tellimuste töötlemise rakenduse haldusliidese ajakohastamine. Aastate jooksul tekkinud tehniline võlg pärsib süsteemis suuremate arhitektuuriliste üleminekute teostamist. Töö käigus analüüsitakse olemasolevat haldusliidest ning selle puuduseid. Lisaks uuritakse võimalikke lahendusi nende kitsaskohtade kõrvaldamiseks. Sõltuvalt analüüsi tulemustest valitakse lähenemisviis süsteemi ajakohastamiseks. Vastavalt lähenemise mastaapsusele paneb autor paika antud töö skoobi ja realiseerib selle.

Teises peatükis tutvustatakse lähemalt lõputöös lahendatavat probleemi ning selle tausta. Selleks esitatakse süsteemi arhitektuuri tervikvaade ning kirjeldatakse tellimuse töötlemise süsteemi peamisi funktsionaalsusi.

Kolmandas peatükis tutvustatakse olemasolevat haldusliidest. Teostati koodi analüüs, et määrata kindlaks kasutatud tehnoloogiad ja integratsioonid teiste süsteemide/rakendustega. Lisaks testiti olemasolevat lahendust, et teha kindlaks pakutav funktsionaalsus ja selle puudused. Peatüki lõpus võetakse kokku analüüsi tulemused ning valitakse parim lähenemine tehnilise lahenduse moderniseerimiseks.

Neljandas peatükis, tuginedes analüüsi käigus tekkinud tulemustele, määratakse uue lahenduse funktsionaalsed ja mittefunktsionaalsed nõuded. Lisaks selgitatakse välja kasutusele võetavad tehnoloogiad ja äriloogika kihis teostatavad täiendused.

Viiendas peatükis kirjeldatakse analüüsile tuginedes uue lahenduse realiseerimist. Välja on toodud ka kasutatava arendusmetoodika kirjeldus ning töövahendite valik varasemalt valitud tehnoloogiate utiliseerimiseks.

Kuuendas ja viimases peatükis antakse ülevaade valminud lahendusest. Tutvustatakse loodud lahenduse funktsionaalsust ja kasutajaliidest. Lisaks kirjeldatakse võimalusi projekti edasiarenduseks ja tulevikku planeeritud arendusi.

2 Probleem ja eesmärk

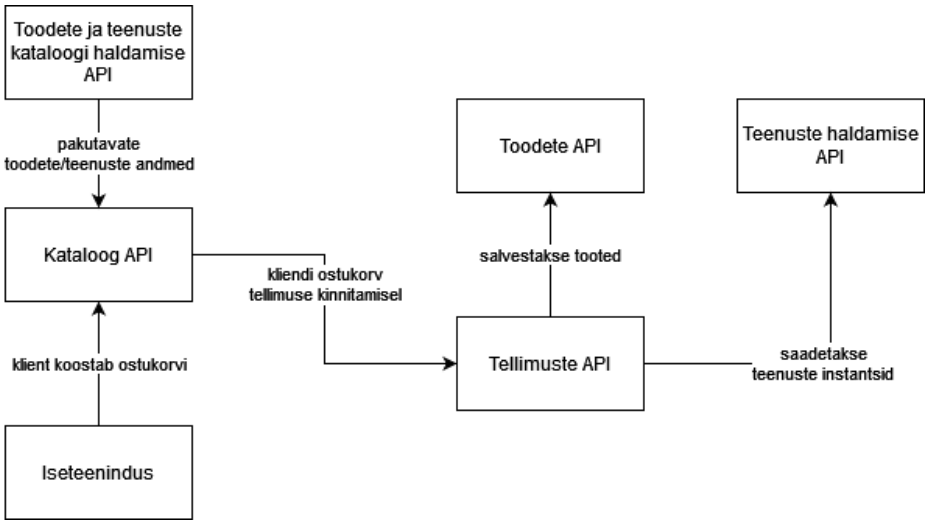
Telekommunikatsiooni ettevõttel Telia kui Eestis on oluline olla paindlik ning pakkuda oma klientidele innovaatilist ja parimat kasutajakogemust. Uute arenduste tarneaja minimeerimiseks on oluline, et kõik individuaalsed rakendused oleksid paindlikud ning kogu lahenduse arhitektuur komponendipõhine. Ka üksik rakendus, mis kogu süsteemi vaates mängib näiliselt ebaolulist rolli, võib osutuda takistuseks.

2.1 Probleemi taust

Telia arendab hetkel ettevõttesiseselt välja teenuste tellimise- ja tarneprotsesside platvorm, koodnimega ToTeKa (*Toodete ja Teenuste Kataloog*), mille eesmärk on kogu ettevõtte tellimuste ja tarneprotsessi haldamine, konfigureerimine ja seire. See koosneb suurest hulgast klient-server rakendustest, mis moodustavad omavahel ühtse terviku.

Joonis 1 kirjeldab kliendi tellimuse teekonda läbi ToTeKa rakenduste. Kliendi teekond hakkab iseteeninduses t, mis tuginedes kataloog API-le (*Application Programming Interface*) paneb kokku ostukorvi varasemalt kirjeldatud toodete ja teenuste põhjal. Need kirjeldused tulevad kataloogi haldamise APIst, mis pakub tehnilistele ja ärilistele kasutajatele võimekust seadistada ja omavahel siduda erinevaid pakkumisi, hindu ja muid tunnuseid, mida on vaja mõne toote või teenuse puhul. Tellimuse kinnitamisel saadetakse kliendi loodud ostukorv tellimuse töötlemise rakendusse (edaspidi tellimuste API), kus algab selle töötlemine. Tellimuste API on kriitiline osa kogu ToTeKast, täites nelja peamist ülesannet:

- Tellimuste täitmise protsessi kui terviku juhtimine.
- Tellimuste käsitlemine ning nende staatuste haldamine.
- Tellimuste kohta info vahendamine osapooltele.
- Toodete instantside haldamine koostöös teiste süsteemidega.



Joonis 1. Kliendi tellimuse kulg läbi erinevate ToTeKa rakenduste

Tellimuste API on kirjutatud Java programmeerimiskeeles kasutades Spring raamistikku ning on üles ehitatud REST (*Representational State Transfer*) veebirakenduste arhitektuuri põhimõtetel. REST on veebiteenuste loomise arhitektuuriliste printsiipide kogum, mille eesmärk on pakkuda standardiseeritud viisi süsteemide vaheliseks suhtlemiseks. [5]

Tellimuste API koosneb tegelikkuses kahest eraldi rakendusest: tellimuste lugemise API ja kirjutamise API. Tellimuste API ülesanne on tellimuste ja nendega seotud andmete vastuvõtmine ja töötlemine. Lugemise API ülesanne on liita kokku tellimuse infot erinevatest keskkondadest ning olla selle teabe peamiseks vahendajaks lõppkasutajatele. Nende võimekuste poolitamine kaheks eraldiseisvaks rakenduseks aitab hajutada koormust.

Tellimuste APIsse jõuab esimene sisend kliendi soovidest struktureeritud ostukorvina JSON-vormingus (*Javascript Object Notation*). Niipea, kui rakenduseni jõuab ostukorv, saab sellest tellimus. Olenevalt andmete sisust, käivitatakse rakenduse poolt vastavad protsessid, mis hõlmavad endast näiteks andme kontrolli ärireeglitele vastavusele, suhtlust kolmandate teenustega või staatuste muutmisi. Kõik andmed ja nende muudatused sünkroniseeritakse jooksvalt lugemise APIsse.

Ettevõte kasutab paralleelselt teisi analoogseid tellimuste töötlemise süsteeme, kust aktiivselt migreeritakse teenuseid uuele platvormile. Lõppeesmärgiks on kogu Telia toodete ja teenuste kataloogi tellimuste haldamine uues süsteemis. Peamiseks keerukuseks selles ülemineku protsessis on andmestruktuuride erisused ning toodete ja

teenuste paljusus. Kõige otsesem lahendus vanade andmete teisendamine sobivasse vormingusse, kuid Telia on andmete vormingu osas võtnud pikaajalisema eesmärgi, milleks on TMForumi poolt välja töötatud telekom-ettevõtetele mõeldud API standardile üleminek. Kõigi uute rakenduse puhul on üks nõuetest selle andmevormingu kasutamine. Üks rakendustest, mis seda nõuet ka juba järgib, on tellimuste lugemise API.

2.2 Probleemi püstitus

Tellimuste API peale loodud haldusliides omab kogu ToTeKa süsteemi vaates olulist rolli. See on peamine koht, kus tehnilised kasutajad saavad tellimusi ning nendega seotud protsesse jälgida ja hallata ilma, et tuleks suubuda otse rakenduste andmebaasidesse. Rakendus suudab suuremas osas täita sellele seatud funktsionaalseid nõudeid.

Samal ajal, kui ülejäänud ToTeKa süsteemis on aastatega toimunud palju muudatusi, on tellimuste API haldusliides jäänud suuresti samaks. Seni on nõudnud enamik arendusi väikseid parandusi. Aastate jooksul kogunenud tehnilise võla tõttu on selle ülalpidamine ja edasiarendamine muutunud ebaotstarbekaks ja ajamahukaks. Nendest probleemidest tulenevalt on jäänud teostamata mitmeid arendussoovid rakenduse peakasutajatelt. Rakendus on ka suure tõenäosusega saamaks takistuseks suuremate edasiste arenduste planeerimisel, nagu näiteks TMF standardi laiemalt omaks võtmine. Rakenduse puuduseid kirjeldatakse põhjalikumalt järgmises peatükis.

3 Olemasoleva lahenduse analüüs

Antud peatükis on analüüsitud olemasolevat haldusliidest. Analüüsil oli viis peamist eesmärki:

- Teha kindlaks pakutav funktsionaalsus.
- Luua põhjalik ettekujutus rakenduse koodi arhitektuurilisest ülesehitusest.
- Tuvastada rakenduse loomiseks kasutatud tehnoloogiad.
- Sõnastada rakenduse peamised puudused.
- Valida lähenemisviis rakenduse moderniseerimisele.

3.1 Funktsionaalsus

Praegusel tellimuste API haldusrakendusel on neli peamist funktsionaalsuste komplekti:

- Tellimuste haldamine - Kasutajad saavad otsida tellimusi erinevate parameetrite järgi ning vaadata nende kohta üksikasjalikku teavet. Detailinfo vaates tervet andmestikku tellimuse ning sellega seotud alamolemite s.h. läbitud töötlusprotsesside kohta ja muudatusi tellimuste andmestikus. Samuti on võimalus tellimuse peal teha erinevaid toiminguid, nagu näiteks tellimuse tühistamine või tellimuse töötlemisprotsessi taaskäivitamine.
- Toodete haldamine - Kasutajatel on võimalik hallata tellimuste ridadega seotud tooteid. Näiteks otsida neid erinevate parameetrite abil, vaadata üksikasju ning otsida välja seoseid teiste toodete, tellimuste ja hinnaridadega. Sarnaselt tellimustele, on ka toote detailvaates võimalus uuesti saata andmeid teistele süsteemidele.

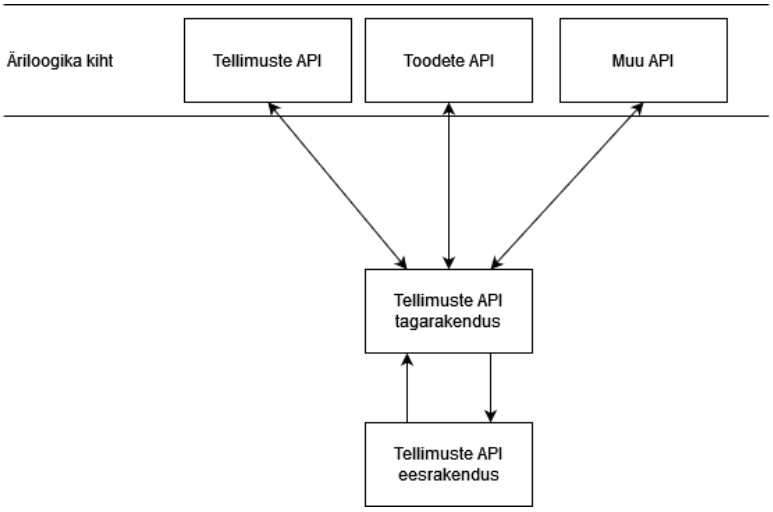
- Toote hinnaridade haldamine - Rakendus võimaldab kasutajatel otsida ja vaadata üksikasjalikke andmeid toote hinnaridade kohta, sealhulgas seotud hindu, nende ajalugu, kehtivaid soodustusi ja arveldussüsteemi teavet.

- Seire - Haldusliideses on põhjalik koondpaneel kõigi käsitletavate olemite staatuse jälgimiseks. Kasutajal on võimalus jälgida vigaseid hinnaridu, tellimusi, tooteid ja protsesse. Kasutajatel on võimalus jätta kommentaare vigaste olemite kohta.

Kuna eelnevalt välja toodud funktsionaalsus on liidetud kokku mitme erineva rakenduse poolt, siis sellest tulenevalt on haldusrakendusel võrdlemisi palju integratsioone erinevate äriloogika kihi serverrakendustega.

3.2 Rakenduse tutvustus

Tellimuste API haldusrakendus koosneb klient-server rakendusetest ning põhineb *backend for frontend* (lühendatult *BFF)* arendusmustril. *BFF* on arhitektuurne muster, mille peamise põhimõtte kohaselt on igal klientrakendusel eraldi serverrakendus, mis on kohandatud vastavalt klientrakenduse vajadustele. Selle ülesanne on olla vahelüliks kasutajaliidese ja tarbitavate teenuste vahel, pakkudes ühtset liidest mitmele äriloogika kihi serverrakendusele. Ainukesed pöördumised, mida klientrakendus teeb, on enda serverrakenduse poole, mis omakorda pärib infot äriloogika kihi serverrakendustelt. [6] Joonisel 2 on näha selle mustri rakendamist haldusliidese näitel.



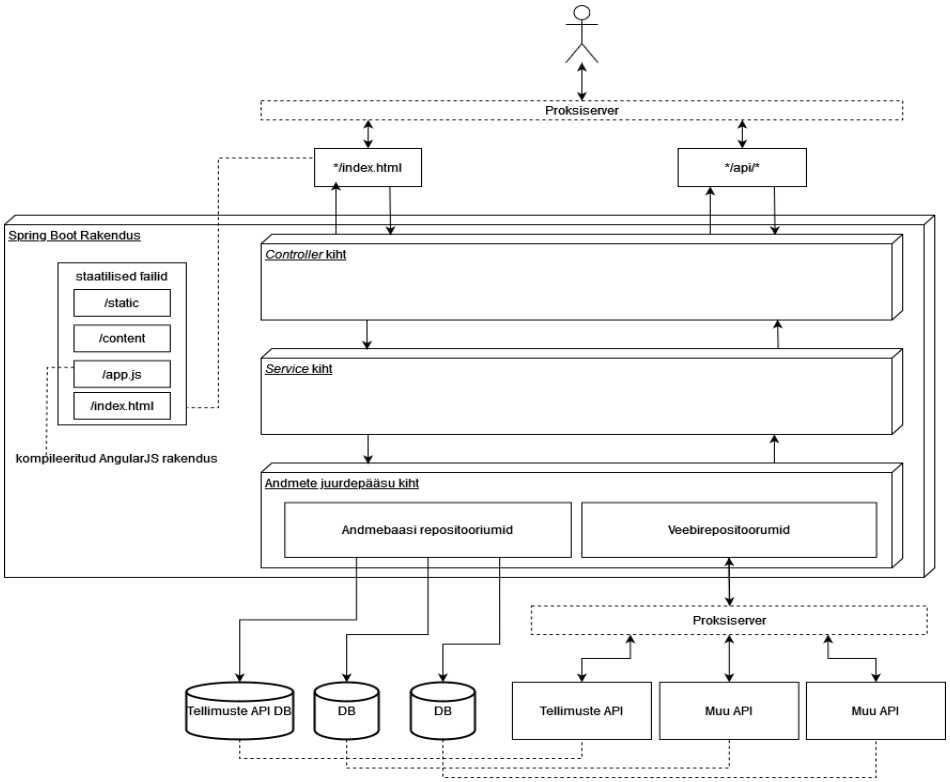
Joonis 2. *BFF* muster haldusliidese näitel

Tänaseks on haldusrakendus Telia poolt kuulutatud vananenuks. Klient- ja serverrakendus on omavahel tihedalt seotud üheks tervikuks ja kogu rakenduse funktsionaalsus on ehitatud ühtsesse koodibaasi. Lõpptulemus on Spring Web MVC (*Model-View-Controller*) rakendus, mis serveerib AngularJS klientrakendust ja ühtlasi täidab ka serverrakenduse rolli. (Joonis 3)

Haldusrakenduse genereerimiseks on kasutatud JHipster-it. JHipster on tasuta ja avatud lähtekoodiga tööriist, mis aitab arendajatel kiiresti luua *full-stack* veebirakendusi. JHipsteri rakenduse genereerimisel on võimalik kasutada JDL-i (*JHipster Domain Language*), kus on võimalik kirjeldada rakendused, nende olemid ja nende olemite vahelised seosed. Vastavalt JDLis kirjeldatule genereeritakse eelkonfigureeritud ja kohandatav projekti šabloon, kus on realiseeritud MVC muster ning hulk elementaarset funktsionaalsust nagu autentimine, andmebaasiühenduvus ja mitmed API teenused. Serveripoolseks tehnoloogiaks on Java koos Spring Boot raamistikuga ning klientrakenduse puhul saab valida Reacti, Angulari ja Vue vahel. [7] [8] Eesrakenduse raamistiku valikuks oli Google poolt arendatud originaalne AngularJS, mis on tänaseks kuulutatud mittesoovitatavaks. [9]

Haldusrakendusel on oma andmebaasi skeem tellimuste API andmebaasi koosseisus, mis on vajalik JHipsteri poolt genereeritud kasutajate haldusega seotud funktsionaalsuse jaoks. Kuid see ja ka enamik muust JHipsteri poolt pakutavast baasfunktsionaalsusest on jäänud kasutamata. Vastavalt ettevõttesisestele turvanõuetele, toimub autentimine ja autoriseerimine organisatsiooni sisevõrgus asuva proksiteenuse kaudu, mis varustab haldusrakendust sessiooni võtme loomiseks vajaliku informatsiooniga.

Infot küsitakse serverrakendustest kahel erineval viisil: osa andmeid küsitakse läbi REST-teenuste, kuid suur osa info pärimisest toimub otse vastavate rakenduste andmebaasidest (Joonis 3).



Joonis 3. Haldusliidese arhitektuuriline ülesehitus

3.3 Puudused

Järgmisena on sõnastatud rakenduses tähele pandud tehnoloogilised ja arhitektuursed puudused.

3.3.1 Arhitektuursed puudused

Analüüsi käigus selgus, et valdav osa andmeid päritakse otse serverrakenduste andmebaasidest, mitte selleks ettenähtud teenustelt. DDD (*Domain Driven Design*) arhitektuuripraktika kohaselt hoitakse domeene lahus eraldi andmebaasides ning ligipääs andmetele käib läbi vastava rakenduse, kus on rakendatud domeeniga seotud ärireeglid tagamaks andmete terviklikkust. Põhjused on järgnevad:

- Jõudlus - Andmebaasi skeem ja indekseerimine on optimeeritud just ühe rakenduse nõudmiste jaoks.
- Isoleeritus – Olukorras, kus andmebaasi on vaja uuendada või muul põhjusel ajutiselt peatada, siis mõjutab see vaid ühte rakendust. Lisaks ei saa viga ühes rakenduses kogemata mõjutada mitme rakenduse poolt kasutatavaid andmeid.

- Paindlikkus - Andmete skeemi muutmisel on tuleb domeeni objekte muuta ainult ühe rakenduse koodis. See nõuab lisaressurssi, ning aeglustab arendusprotsessi.

- Turvalisus – Erinevaid turvalisuse ja seire alaseid meetmeid on lihtsam rakendada, kui vaid üks rakendus saab andmebaasile ligi. Kui teistel süsteemidel on neid andmeid vaja, tehakse selle jaoks rakendusse vastav teenus. [10]

Selle praktika peamine puudus esineb olukordades, kus süsteemidel on vaja omavahel andmeid jagada. Sellisel juhul tuleb panustada rohkem integratsioonide loomisele, kas siis veebiteenuste või muude kanalite kaudu. ToTeKa süsteemi loomise hetkel olid mitmed, nüüdseks eraldiseisvad rakendused, põimitud kokku suuremateks monoliitrakendusteks. Haldusrakendus ehitati otse tellimuste API andmebaasi peale. See lähenemisviis oli kiire, mugav ja arhitektuuriga antud ajahetkel kooskõlas.

Teine probleem on seotud haldusrakenduse enda ülesehitusega. Hetkel on klient- ja serverrakendus omavahel tihedalt seotud, kuna tegemist on monoliitrakendusega. Tihe seotus muudab muudatuste läbiviimise keerukamaks ning vähendab paindlikkust, skaleeritavust ja hooldatavust. [11] Lihtne muudatus klientrakenduses ei tohiks nõuda kogu rakenduse uuesti ehitamist.

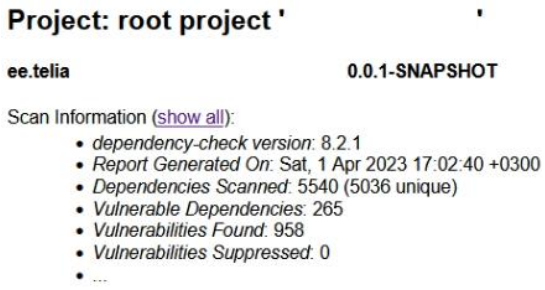
3.3.2 Tehnoloogilised puudused

Kasutatud tehnoloogiate valik ning nende vanus toob endaga kaasa mitmeid probleeme, millest olulisim on aegunud AngularJS versioon 1.6.4. Antud versioon on nüüdseks peaaegu 6 aastat vana [12] ning kogu raamistik on kuulutatud mittesoovitatavaks. [9]

Iganenud on ka kõik alamsõltuvused nii klient- kui serverrakenduses. Kõik vananenud teegid on järk-järgult kaotamas tuge rakendust ümbritsevate tehnoloogiate jaoks. Näiteks klientrakenduse kui terviku testimiseks kasutatav Chromedriver ja Node.js käitluskeskkond on samuti vanad, 3 ja 2 aastat vastavalt, ning neid ei olegi hetkel võimalik enam uuendada AngularJS sõltuvuste tõttu. Hiljuti uuendati tellimuste API andmebaasimootori versiooni, mis nõudis arendusressurssi ka haldusliideses andmebaasi teegi uuendamiseks. Need on vaid mõned näited aja jooksul esinenud muredest. Tehnoloogilised puudused nõuavad ebaproportsionaalselt palju aega rakenduse

hooldamiseks ja see probleem ajaga ainult süveneb. Hetkel piirdub rakenduse arendus ainult selle hooldamisega. Vähem oluline ei ole ka fakt, et kasutajate arendussoove ei ole võimalik täita

Vananenud tehnoloogiad kujutavad endast ohtu ka rakenduse turvalisusele. Kuna enamik sõltuvusi on jäänud uuendamata, on aastatega neis leitud avalikustatud turvaaugud üsna arvukad. Selle tõestamiseks käivitas autor rakenduse peal OWASP Dependency-check käsurea tööriista, mis loendab rekursiivselt üles kõik rakenduse sõltuvused ja kontrollib neid vastu avalikke turvanõrkuste andmebaase. [13] Kui raporti tulemus lühidalt kokku võtta, võib öelda, et pea igas sõltuvuses on rohkem kui 1 vähemalt keskmise kriitilisusega turvaauk. Raporti tulemuste lühikokkuvõtet on näha joonisel 4.



Joonis 4. Turvanõrkuste kontrolli tulemuste lühikokkuvõte

Oluline on rõhutada, et üheski sellise skoobiga rakenduses ei olegi võimalik saavutada haavatavuste arvuks absoluutset nulli.. Selleks on mitmeid põhjuseid, kuid peamine on see, et paljudel haavatavustega sõltuvusel ei olegi alati võtta LTS (*Long Term Support*) versiooni, mis ühilduks kõigi teiste sõltuvustega. Kuna tegemist on sisevõrgu rakendusega, millele puudub ligipääs avalikust veebist, on turvanõrkuste ärakasutamise ohud piiratud.

3.4 Lähenemine rakenduse moderniseerimisele

Ühe rakenduse moderniseerimiseks on kaks valikut: kas refaktoreerimine või rakenduse nullist uuesti üles ehitamine.

Antud juhul on vaja kõrvaldada lisaks kasutusel olevatest tehnoloogiatest tulenevatele probleemidele ka mitmed arhitektuursed puudused. Haldusliides koosneb sisuliselt kahest rakendusest: klient- ja serverrakendus.

Esmalt tuleks need lahku lüüa kaheks eraldi projektiks, vähendamaks seotust nende vahel. See nõuab uute koodihoidlate loomist, eraldiseisvaid pideva integratsiooni ja paigalduse konveiereid ning rakenduste omavahelist liidestamist läbi Telia sisevõrgu.

Serverrakenduses on vaja oluliselt muuta integratsioone teiste äriloogika kihi rakendustega, kõrvaldades infovood, kus haldusliides pärib andmeid otse andmebaasist. See omakorda nõuab panust neid rakendusi haldavatelt meeskondadelt uute teenuste loomise näol. Lisaks on koodibaasis võrdlemisi palju algselt JHipsteri poolt genereeritud baasfunktsionaalsust, mis tuleks eemaldada, kuna reaalsuses on see jäänud kasutamata või siis kasutatakse ainult osaliselt.

Java ja Spring Boot raamistiku osas on meeskonnas pädevus väga kõrge. Siiski nõuaks suurte hüpete tegemine raamistiku ning muude sõltuvuste versioonides ja kasutamata funktsionaalsuse eemaldamine/refaktoreerimine koodibaasist omajagu tööd.

Klientrakenduse puhul on vaja kogu raamistik välja vahetada. Angular on täielik ümberkirjutus AngularJS-ist, mis kasutab JavaScripti asemel TypeScripti, toetab täiesti teisi komplekteerijaid ning oluliselt erinevat koodi arhitektuuri. [14] Kuna erinevused nende kahe vahel on nii mastaapsed, kujutab see endast klientrakenduse täielikku ümber kirjutamist.

Lisaks kõigele eelnevale, selle töö käigus tuleks teha haldusrakenduses üleminek ka uuele andmete kujule, milleks on TMF-i standard.

Tuginedes analüüsi tulemustele otsustas autor koos süsteemiarhitektiga täiesti uue lahenduse loomise kasuks, kuna kahe eraldiseisva rakenduse kapitaalne refaktoreerimine/muutmine on antud olukorras raskem kui uue lahenduse loomine. Uue lahenduse loomisega saab võtta kasutusele tehnoloogiad, mis on organisatsiooni poolt aktsepteeritud, pakkudes süsteemi arendusele perspektiivikamat tulevikku.

3.4.1 Skoop

Uus haldusliides peab oma funktsionaalsuselt olema vähemalt võrdväärne praeguse rakendusega. Kuna ajaliselt ei ole võimalik realiseerida töö lõppeesmärk täielikult, piirdutakse vaid kõige olulisemaga. Lõputöö skoop hõlmab ainult tellimustega seotud funktsionaalsust.

Käesoleva lõputöö skoobiks võtab autor uue haldusliidese MVP loomise veebirakenduse näol. Töö hõlmab ka tellimuste töötlemise rakenduste täiendamist uute teenustega ning nende samade teenuste kasutusele võtmist loodavas veebirakenduses.

4 Uue lahenduse analüüs

Antud peatükis määratletakse loodava veebirakenduse funktsionaalsed ja tehnilised nõuded, kirjeldatakse süsteemi arhitektuur ning valitakse realiseerimiseks kasutatavad tehnoloogiad. Analüüsi käigus luuakse veebirakenduse esimene kujunduse prototüüp, mille pealt kogutakse tagasisidet kasutajatelt. Lisaks kaardistatakse lahenduse realiseerimiseks vajaminevad juurdearendused äriloogika kihi serverrakendustes.

4.1 Nõuded

Esmalt määrab autor uue rakenduse nõuded. Nõuete kirjapanekul on autor võtnud arvesse järgmist:

- Ettevõttesiseselt seatud tehnoloogilisi nõudmisi.
- Tellimuste töötlemise süsteemi tuleviku arhitektuurilisi eesmärke.
- Meeskonna süsteemiarhitekti ja DevOps inseneri soovitusi.

4.1.1 Funktsionaalsed nõuded

Funktsionaalsed nõuded kirjeldavad rakenduse tahtlikku käitumist, mis väljenduvad teenuste, ülesannete või funktsioonidena, mida rakendus täidab. [15] Antud töö skoopi jääb tellimuste haldamisega seotud funktsionaalsus. Töö raames loodava haldusliidese MVP funktsionaalsed nõuded on järgmised:

1. Kasutaja saab otsida tellimusi kõigi tellimust kirjeldavate väljade järgi.
2. Kasutaja saab vaadata tellimuse detailandmeid.
3. Kasutaja saab tühistada tellimust.
4. Kasutaja saab algatada tellimuse andmete sünkroniseerimist tellimuste API-st tellimuste lugemise API-sse.

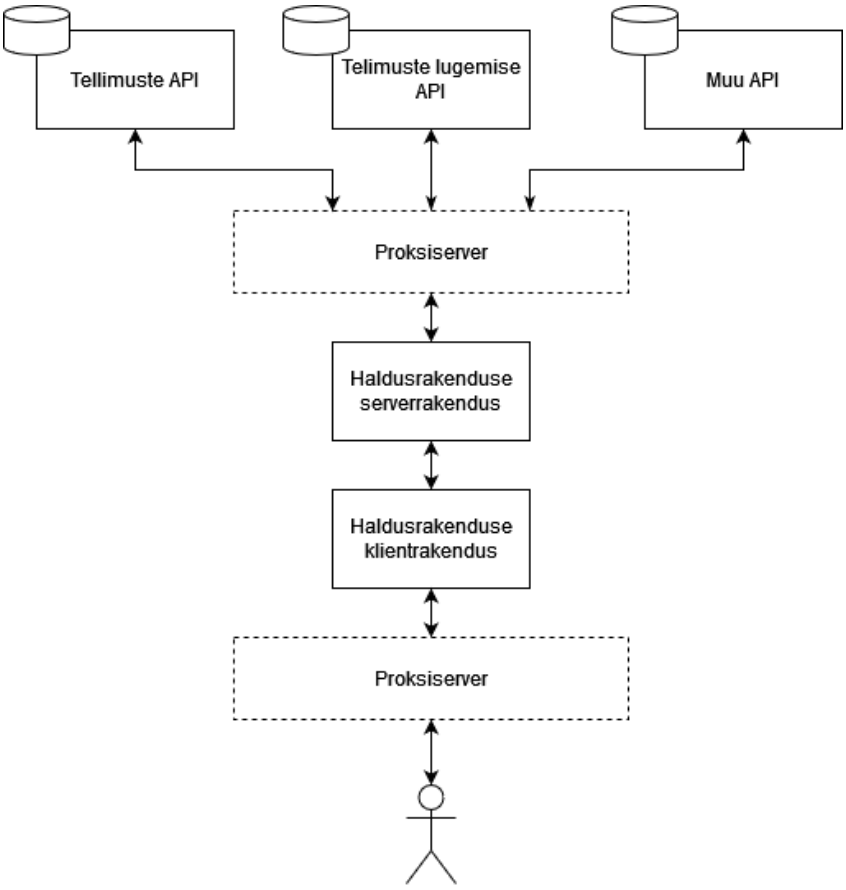
4.1.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded kirjeldavad rakenduse mittekäitumuslikke aspekte ning jäädvustavad omadusi ja piirangud, mille alusel süsteem peab toimima. [16] Need nõuded lõputöö raames loodavale MVPle on järgmised:

1. Haldusrakendus pärib informatsiooni ainult selleks ettenähtud teenuste kaudu.
2. Haldusrakenduse klientrakenduse kujunduslik pool peab kasutama organisatsiooni kujundusraamistikust.
3. Haldusrakenduses peab lõppkasutaja autentimiseks kasutama ettevõttesisesest SSO (*Single sign-on*) lahendust.
4. Haldusrakendusel toimib pidev integratsioon ja paigaldus.
5. Kirjutatav kood ning selle struktuur peab kasutama ettevõttes määratud parimaid praktikaid.
6. Töötavale rakendusele teostada seiret.

4.2 Arhitektuur

Uue haldusliidese puhul saab olema tegemist klient- ja serverrakendusega. Klientrakenduse ja äriloogika kihi vahele jääva vahelüli ainus eesmärk on rakendada kahe kihi vahel saadetavatele päringutele lisatöötlust. Väikese serverrakenduse olemasolu on hädavajalik, sest teenustele ligipääs läbi proksiserveri nõuab lisa autentimist ja autoriseerimist. Loodava lahenduse puhul oleks serverrakendusel omaette proksiserveri roll, mis paneb kõigile päringutele külge haldusliidese tehnilise kasutaja mandaadi. Klientrakenduses ei saa seda lihtsasti realiseerida turvalisuse kaalutlustel, kuna see nõuaks nende andmete vahendamist lõppkasutaja masinasse. Rakenduse struktuuri on kujutatud joonisel 5.



Joonis 5. Uue haldusrakenduse arhitektuur

Kuna uus serverrakendus saab olema oma olemuselt piiratud funktsionaalsusega, on see praktikas teostatav vähese hulga koodiga. Sellest tulenevalt loob autor uue rakenduse koodibaasi monorepositooriumina ning käsitleb rakendust paigaldamisel ühtse komponendina.

4.3 Tehnoloogiate valik

Lahenduse skoobis on uue veebirakenduse loomine, mille realiseerimiseks kasutatavate tehnoloogiate potentsiaalne valik on väga suur. Antud võrdluses langetab autor need otsused sõltuvalt enda ja arendusmeeskonna kogemustest, rakenduse olemusest, ettevõtte sisestest suunistest ning funktsionaalsusest tulenevatest nõuetest.

4.3.1 Klientrakenduse programmeerimiskeel

Kaasaegsed veebirakendused koosnevad kolmest põhikomponendist: HTML, CSS ja JavaScript. HTML on vaid märgistuskeel, mis kirjeldab veebilehe struktuuri ja sisu, CSSi

kasutatakse sisu kujundamiseks ja paigutamiseks ning JavaScript muudab ühe veebirakenduse interaktiivseks ja dünaamiliseks.

Üks variant oleks kirjutada puhast JavaScripti. Aastate jooksul on välja kujunenud mitmeid programmeerimiskeeli, mis varieeruvad erinevas mahus JavaScriptist, pakkudes kas lisafunktsionaalsust või koguni täiesti teistsugust süntaksi. Kuid lõppkokkuvõttes kompileeritakse need kõik puhtaks JavaScriptiks. [17] Selliseid kõrgema taseme keeli, mis kompileeruvad teiseks kõrgema taseme keeleks nimetatakse transkeelteks. [18]

CoffeeScript on üks esimesi selliseid JavaScripti edasiarendusi. Loodud aastal 2009 Jeremy Ashkenas'e poolt, lisab see JavaScriptile juurde mitmeid kasulikke abstraktsioone, pakub lühemat süntaksi, sisukamaid veakoode ja muud. [19]

Üks populaarsemaid selliseid keeli on TypeScript. TypeScript-i peamine eelis on JavaScriptile valikulise staatilise tüüpimise lisamine. Sellele lisaks on TypeScriptis olemas liidesed (*interfaces*), klassid ja moodulid, mis lihtsustavad suuremahuliste rakenduste kirjutamist ja hooldamist. [20]

Tabelis 1 on hinnatud eelnevalt välja toodud programmeerimiskeeli nelja kriteeriumi järgi: autori kogemus, kasutus Telia projektides, õppimiskõver ja keele ühilduvus JavaScript teekidega.

Tabel 1. Javascripti ja levinud Javascripti transkompileeritud keelte võrdlus

	JavaScript	TypeScript	CoffeeScript
Autori kogemus	Keskmine	Keskmine	Puudub
Kasutamine Telia projektides	Keskmine	Suur	Puudub
Õppimiskõver	Lauge	Lauge	Lauge
Ühilduvus JavaScript teekidega	-	Kõrge	Kõrge

Juhindudes TypeScripti ulatuslikust kasutamisest teistes Telia projektides, ning antud keele poolt pakutud lisafunktsionaalsusest, on autori hinnangul parim variant TypeScript.

4.3.2 Klientrakenduse teek/raamistik

Valdav enamik JavaScriptis loodud tänapäeva veebirakendusi on ehitatud tuginedes mõnele teegile või raamistikule. JavaScripti teek on eelnevalt kirjutatud JavaScripti koodi kogum, mis on loodud eesmärgiga hõlbustada uute rakenduste loomist. Teegid pakuvad arendajatele eelvalmistatud funktsioone ja mooduleid, mida arendaja saab kasutada laialt levinud ülesannete lihtsustamiseks. [21]

Raamistik on samuti oma olemuselt lihtsalt eelnevalt kirjutatud koodi kogum, mille eesmärk on lihtsustada arendaja elu. Kuid erinevalt teekidest, pakuvad raamistikud rakenduste loomisel palju struktueeritumat lähenemist. Raamistiku kasutamine hõlmab endas enamasti mõne arhitektuurilise mustri kasutamist ja eelnevalt defineeritud suuniste järgimist. [22] Antud analüüsis on võrreldakse kolme erinevat tehnoloogiat: Vue, Angular ja React.

Vue, Angular ja React on kõik populaarsed JavaScripti raamistikud kasutajaliideste loomiseks. Vue on tuntud oma lihtsuse ja kasutusmugavuse poolest. React on populaarne valik suuremate projektide jaoks ning pakub suurt teekide ökosüsteemi. Angular pakub terviklikku lahendust ja on orienteeritud just suuremahuliste rakenduste loomiseks. [23] Populaarsuse poolest on enim kasutatud suure edumaaga React, millele järgneb Angular ning viimasel kohal on Vue. [23], [24]

Tabel 2. Teekide ja raamistike võrdlus

	Vue	Angular	React
Autori kogemus	Madal	Madal	Kõrge
Kasutus Telia projektides	Väike	Keskmine	Suur
Õppimiskõver	Lauge [25]	Järsk [25]	Keskmine [25]
Populaarsus	Keskmine [24]	Keskmine [24]	Kõrge [24]

Tuginedes tabelile 2, on selge valik antud juhul React. Teegiga on autoril suur kogemus ning seda kasutatakse ulatuslikult paljudes projektides üle maailma, sealhulgas ka enamikes Telia projektides. Seoses nende asjaoludega, ei ole vaja pühendada aega uue

tehnoloogia õppimisele. Lisaks on probleemide tekkimisel võimalik leida abi kolleegidelt kui ka veebist.

4.3.3 Serverrakenduse programmeerimiskeel

Lahenduse raames on vaja luua ka lihtne serverrakendus, mille ainus ülesanne on olla vahelüliks klientrakenduse ja äriloogika kihi vahel. Võrreldi nelja levinumat programmeerimiskeelt..

Java, mida on juba antud töö raames mainitud, on üks võimalik valik. Javat kasutatakse tavaliselt suuremate, ettevõtte tasandi rakenduste, sealhulgas API-de arendamiseks. Laialt on levinud raamistikud ja teegid nagu näiteks Spring Boot ja Hibernate, mis lihtsustavad API-de loomist ja hooldamist.

Teine valik on Node.js, populaarne JavaScripti käituskeskkond, mida samuti kasutatakse laialt veebirakenduste ja API-de loomiseks. Node.js pakub suurt ökosüsteemi moodulitest ja pakettidest, mis võimaldavad väikese vaeva ja koodi hulgaga luua varieeruva skoobiga veebiteenuseid ja rakendusi.

C# on Microsofti poolt välja töötatud objektorienteeritud programmeerimiskeel, mis sarnaneb oma olemuselt Javale. Raamistikud .NET ja ASP.NET on ulatuslikult kasutatud valikud veebirakenduste ja API-de loomiseks.

Python on mitmekülgne kõrgetasemeline programmeerimiskeel, mida kasutatakse sageli veebirakenduste arenduseks ja andmeanalüüsiks. Pythoni keeles on saadaval mitmeid raamistikke, näiteks Flask ja Django, mis on suunatud veebiteenuste/-rakenduste loomisele.

Tabel 3. Serverrakenduse programmeerimiskeelte võrdlus

	Java	Javascript	C#	Python
Autori kogemus	Kõrge	Keskmine	Keskmine	Madal
Kasutus Telia projektides	Suur	Keskmine	Väike	Väike
Õppimiskõver	Keskmine	Lauge	Keskmine	Lauge
Populaarsus	Kõrge [26]	Kõrge [26]	Keskmine [26]	Kõrge [26]

Tabel 3 alusel tundub kõige parem valik Java, millele järgneb JavaScript. Kuid autori hinnangul muudab Java kasutamine koos sõltuvustega nagu Spring ja/või Hibernate nõutava tüüpkoodi hulga ja üldise üldkulu seoses rakenduse ehitamise ja hooldamisega ebaproportsionaalselt suureks. Seda juhul kui võtta arvesse loodavat funktsionaalsust. Sellise skoobiga rakendust on võimalik Node.js keskkonnas realiseerida paari failiga, mida pole vaja paigaldamiseks isegi kompileerida. On ka tehtud katsetusi, millest on selgunud, et selliste väikeste rakenduste puhul on süsteemi protsessori ja mälu kasutus väiksem. [27] Java ei ole mõeldud selliste rakenduste loomiseks. Nagu eelnevalt välja toodud, on Java populaarne valik robustsete, ettevõtte taseme APIde loomiseks ja sellest tulenevalt on sinna kohe kaasa pakitud lai valik tööriistu, millest valdavat enamikku hetkeolukorras ei lähe tegelikult vaja. Nendest asjaoludest lähtuvalt otsustas autor siiski JavaScripti kasuks, ühtlustamaks kogu uue lahenduse tehnilist pinu.

4.4 Juurdearendused äriloogika kihi serverrakendustes

Uue lahenduse skoopi jäävad juurdearendused äriloogika kihi serverrakendustes. Selleks, et kõrvaldada praeguses haldusrakenduses kasutusel olevat praktikat, kus andmeid päritakse otse andmebaasist, on vaja luua mitmeid eriotstarbelisi teenuseid, mis tagastaksid sama struktuuriga andmeid, kuid millele rakendub sarnastest teenustest erinev autoriseerimise mehhanism. Nende teenuste tarbeks loob autor rakendustesse uued moodulid, kuhu konsolideerub kõik töö raames ja ka edaspidi kirjutatud haldusrakendusega seotud kood.

Tellimuste API-sse on vaja luua kaks teenust, mis teevad järgnevat:

- Tellimuste sünkroniseerimise teenus, mis taaskäivitaks tellimuste API-st kindla tellimuse info saatmise tellimuste lugemise API-sse.
- Tellimuse tühistamise teenus, mis viib kindla tellimuse lõppolekusse.

Tellimuste lugemise API-sse on vaja luua järgmised teenused:

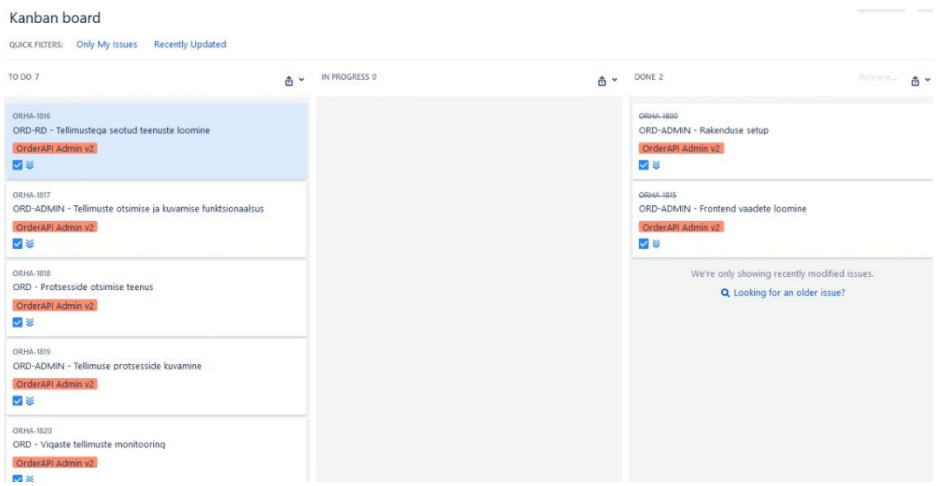
- Tellimuste pärimise teenus, mis tagastaks vastavalt filtrile loendi kõigist kriteeriumitele vastavatest tellimustest.
- Üksiktellimuse pärimise teenus, kust on võimalik pärida konkreetse tellimuse andmeid, koos antud tellimuse kõigi alamolemitega, mis on salvestatud teistesse äriloogika kihi rakendustesse.

5 Realisatsioon

See peatükk kajastab uue haldusrakenduse MVP loomise arendusprotsessi. Esmalt on eraldi välja toodud kasutatud arendusmetoodika ja arendustööde teostamiseks kasutatud töövahendid. Seejärel on kirjeldatud äriloogika kihi serverrakendustesse sisse viidud täiendusi ja kogu uue veebirakenduse loomise arendusprotsessi.

5.1 Arendusmetoodika ja töövahendid

Uue rakenduse loomisl kasutas autor agiilset arendusmetoodikat. Arendusprotsess algas kogu planeeritud funktsionaalsuse loomiseks seotud tööde piletiteks kirjeldamisega ning nende järjekorda seadmisega. Tööde visuaalseks jälgimiseks kasutati kolme defineeritud etapiga Kanban tahvlit (vt Joonis 6). Iga töötlemine sisaldas arendamist, testimist ning töö sisulist valideerimist koos meeskonnaliikmete või tehnilise juhendajaga.



Joonis 6. Kanban tahvel koos kavandatud töödega

Telias on kasutusel Atlassiani poolt pakutud töövahendid. Tööde planeerimiseks kasutatakse Jira-t, koodirepositooriumi ja versioonihaldustarkvaraks Bitbucketit. Arendustööde läbiviimisel serverrakenduste puhul kasutati JetBrains IntelliJ integreeritud arenduskeskkonda, ning klientrakenduse arendamiseks Microsofti Visual Studio Code koodiredaktorit.

5.2 Äriloogika kihi rakenduste täiendamine

Vältimaks vanas rakenduses esinevat arhitektuurset viga, kus haldusliides küsib andmeid otse vastavate rakenduste andmebaasidest, on vaja luua äriloogika kihi serverrakendustesse vastavad REST-teenused. Realiseerimaks töö skoobis olevat funktsionaalsust, tuli täiendada tellimuste ja tellimuste lugemise API-t.

Rakendustesse loodi uued alammoodulid, eraldamaks haldusliidesega seonduvat koodi ülejäänud rakendusest. Vastavatesse moodulitesse lõi autor uued teenused *controllerite* näol ja nende teenuste poolt tagastatavate ressursside DTO-d.

Mõlemas eelnevalt mainitud rakenduses olid varasemast juba olemas vajalikud andmete juurdepääsu kihid, mille kaudu sai andmebaasist pärida andmeid. Tuginedes nendes kihtides realiseeritud repositooriumidele oli autoril triviaalne kätte saada vajalik info, vastavalt päringus kirjeldatud otsingukriteeriumitele.

Loodud teenustele tuli luua ka eriotstarbeline autoriseerimine. Päringuid saavad teha vaid ettenähtud Active Directory (edaspidi AD) õiguste grupiga kasutajad. Kontroll toimib klientrakenduse poolt edastatud JWT sisus kirjeldatud AD kasutajanime järgi, mille põhjal valideeritakse pöörduja kasutaja kuulumist vastavasse AD gruppi.

5.3 Uue lahenduse arenduskäik

Töö käigus valmis uus haldusrakendus, mis koosnes klientrakendusest ja serverrakendusest. Järgnevad alapeatükid kirjeldavad rakenduste erinevaid arenduse etappe nende realiseerimise järjekorras.

5.3.1 Klientrakenduse algseadistamine

Esimene samm oli luua rakenduse põhi. Autor kasutas selle jaoks Reacti haldajate endi poolt loodud käsurearakendust *create-react-app*. See on saadaval Node.JS paketihalduris NPM (*Node Package Manager*) ning genereerib arendajale minimaalse eelseadistatud veebirakenduse arenduskeskkonna, kus on olemas rakenduse ehitamiseks vajalikud sõltuvused, lihtne struktuur ja arendusserver.

Järgmiseks paigaldas autor rakendusele veel mitmeid teeke, mida läks vaja rakenduse realiseerimiseks. Need olid React Router, React Redux, Axios, *sso-client, telia-front-*

react. React Router lisab React-ile dünaamilise URL-põhise navigeerimise erinevate komponentide vahel. React Redux aitab Reacti rakendustes hallata rakenduse globaalseid olekuid, pakkudes võimalust lihtsasti ühendada igat komponenti ühte tsentraliseeritud rakenduse seisu hoidlasse. Axios võimaldab saata ja vastu võtta HTTP-päringuid JavaScripti rakendustes.

Viimased kaks teeki on Telia ettevõttesisesed. Esimene neist, *sso-client*, hõlbustab veebirakenduse integreerimist SSO lahendusega. Teine on kujundusraamistik, mis pakub eelnevalt stiliseeritud kasutajaliidese komponente. Antud raamistiku kasutamine lubab arendajatel kergesti luua rakendusi, mis vastavad organisatsioonis kasutusel olevale kujunduskeelele ja brändi identiteedile. Need raamistikud on saadaval vaid ettevõtte privaatses repositooriumis.

Rakenduse algseadistamiseks kasutatud käske on näha joonisel 7.

```
npm create-react-app my-app --template typescript
npm install react-router-dom
npm install react-redux
npm install axios

// sisevõrgu repositooriumi määramine arenduskeskkonnas
npm config set registry https://$REPOSITOORIUMI_URL
npm install sso-client
npm install telia-front-react
```

Joonis 7. Klientrakenduse loomiseks kasutatud käsurea käsklused

5.3.2 Vaadete loomine

Üks mittefunktsionaalsetest nõuetest oli ettevõttesisese kujundusraamistiku kasutamine. Tuginedes eelnimetatud raamistiku pakutud komponentidele, oli autori järgmine samm luua programselt kõikide vaadete prototüübid, mida rakendus peab eelnevalt defineeritud funktsionaalsete nõuete raames pakkuma. Vaated on staatilised, ehk päringuid serverrakendusele veel ei tehta, kuid kõik interaktiivsed elemendid ja suunamised toimivad.

Antud rakenduse loomisel on autor nii disaineri kui arendaja rollis. Prototüübi realiseerimiseks ettenähtud tööriistade kasutamise asemel, nagu näiteks Figma, asuti kohe lähtekoodi kirjutama. Sellisel lähenemisel on mitu eelist:

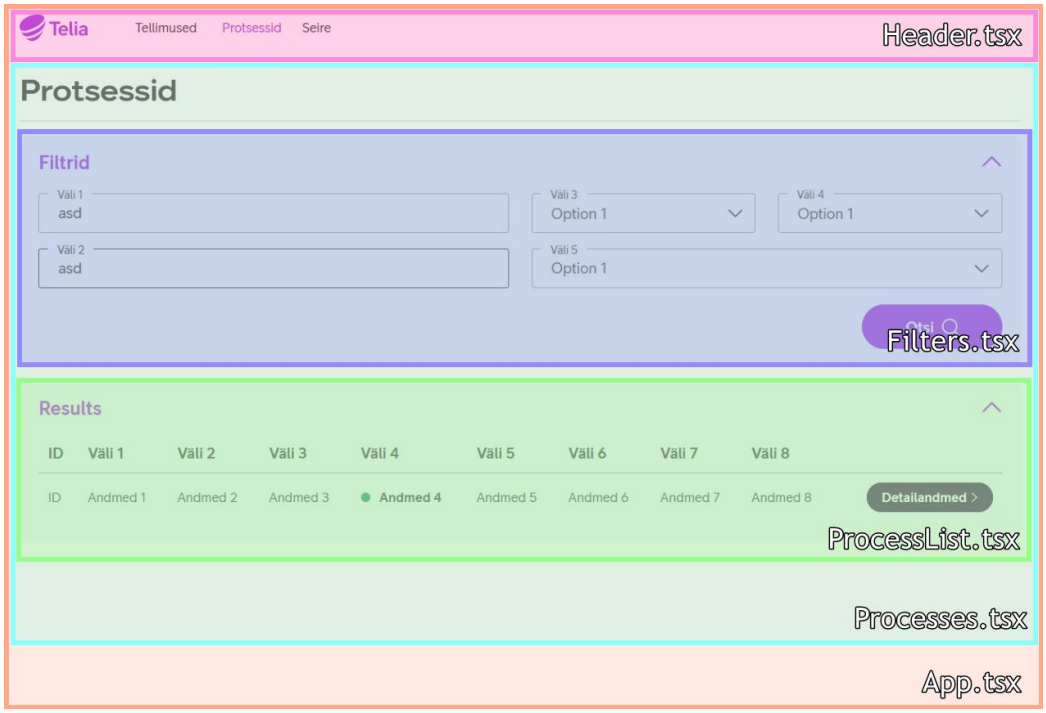
- Vähene kogemus prototüüpimiseks mõeldud tööriistadega oleks nõudnud aega õppimiseks.
- Peale loodud kujunduse valideerimist saab kohe tegeleda funktsionaalsete töödega kuna kõik on juba lähtekoodis olemas.
- Rakenduse kasutajad saavad parema tunnetuse, kuidas päris rakendus veebilehitsejas käitub võrreldes mõnes prototüüpimise tööriistas loodud lahendusega.
- Haldusrakendus on mõeldud ennekõike tehnilistele kasutajatele ja sellepärast ei ole vajadust teha rakendusele UI/UX alast analüüsi.

Enne rakenduse edasi arendamist küsiti kasutajatelt tagasisidet peamiste vaadete kohta. Tagasiside oli positiivne, kuid mainiti ka mõningaid kaebusi. Üks kaebus oli, et vaadetes oleksid elemendid rohkem hajutatud, et neid oleks lihtsam lugeda ja vajutada. Teine ettepanek oli seoses veebilehitseja ajaloos navigeerimisega. Kui detailvaatest tagasi minna otsingu vaatesse, võiks olla otsingu tulemused ja parameetrid salvestatud. Lähtuvalt tagasisidest viidi sisse parandused, mida oli võimalik antud faasis teostada. Vaadete prototüübid, mis ühtlasi peegeldavad ka rakenduse lõplikku seisu on näha töö lisas 2.

5.3.3 Komponentide hierarhia ja rakenduse struktuur

Peamine eesmärk prototüübina loodud programselt realiseeritud vaadete loomisel oli tagada optimaalne kuvakvaliteet ja kiirus. Selleks loodi lihtsuse mõttes ühe vaate kohta vaid üks komponent. Tagamaks rakenduse edaspidise hooldatavuse, on oluline luua väikseid ja fokuseeritud komponente, mis täidavad korraga võimalikult vähe erinevaid ülesandeid. [28] Väiksemad komponendid on lihtsamini testitavad, taaskasutatavad ning skaleeritavad.

Prototüübi arendamise käigus loodud monoliitkomponendid said laiali võetud loogilisteks väikesteks juppideks. Joonisel 8 on näha komponentide ülesehitust protsesside otsimise vaates.



Joonis 8. Komponentide hierarhia protsesside otsingu vaates

Veebirakenduse puhul on tegemist üheleherakendusega ehk *SPA*-ga. SPA (*Single Page Application*) on veebirakendus, mis laeb ühe HTML-lehekülje ja uuendab sisu dünaamiliselt, kui kasutaja lehega suhtleb, ilma et oleks vaja kogu lehekülge uuesti laadida. [29] Komponent „*App*“ on juurkomponendiks, mis kuvab kogu veebirakendust ning selle alamlehti läbi react-router teegi.

5.3.4 Serverrakendus

Järgmise sammuna algseadistati serverrakendus. Vastavalt analüüsi tulemustele, loodi rakendus JavaScriptis Node.js käitluskeskkonda kasutades. Rakenduse realiseerimiseks tugineti Express raamistikule. Express on populaarne JavaScripti raamistik, mida kasutatakse veebirakenduste ja teenuste loomiseks. See pakub lihtsat liidest HTTP-päringute käsitlemiseks ja veebiserverite ehitamiseks. [30]

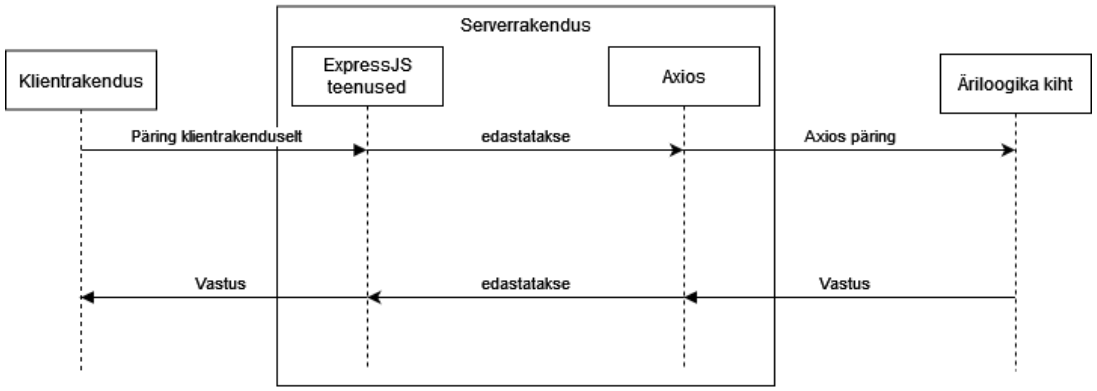
Rakenduse algseadistamiseks loodi esmalt pakett, mis hõlbustaks rakenduse käsitlemist pideva integratsiooni ja paigalduse käigus. Seejärel paigaldati vajalikud teegid, Express, Axios ja dotenv.(Joonis 9) Viimane teek, dotenv, millest pole veel juttu olnud lubab rakendusele faili näol kaasa anda keskkonnamuutujaid. Nende kaudu saab pideva

integratsiooni ja paigalduse käigus anda kaasa andmed, mis on vastavas keskkonnas(arendus, test, toodang) vajalikud.

```
// npm paketi loomine vaikimisi seadetega
npm init --yes
npm install express
npm install axios
npm install dotenv
```

Joonis 9. Serverrakenduse loomiseks kasutatud käsurea käsklused

Nende teekidega on võimalik realiseerida rakenduse põhifunktsionaalsus, mis on kujutatud joonisel 10. Klientrakendusest tulevale päringule tehakse autoriseerimisega seotud lisatöötlust, ning saadetakse päring edasi äriloogika kihile kasutades Axiost. Päringu vastus edastatakse töötlemata kujul otse tagasi klientrakendusele.

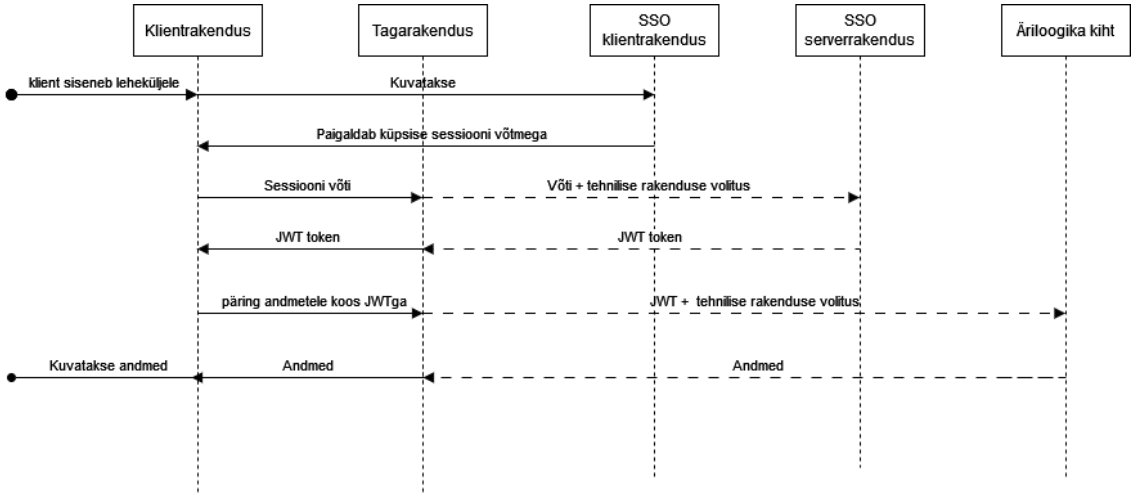


Joonis 10. Haldusrakenduse info liikumise järjestusdiagramm

5.3.5 Autentimine ja autoriseerimine

Üheks rakenduse mittefunktsionaalsetest nõuetest oli ettevõttesisese SSO valmislahenduse kasutamine lõppkasutaja autoriseerimiseks. (vt Joonis 11) Antud lahendus koosneb kasutajaliidesest ja lisafunktsionaalsust pakkuvatest REST-teenustest. Kasutajaliides pakub mugavat võimalust kõikide võimalike toetatud meetoditega sisselogimiseks. Sisselogimisel seatakse kasutaja veebilehitsejasse hulk küpsiseid, millest üks on sessiooni võti. Küpsis kehtib domeenil „telia.ee“ ning tema kõikidel alamdomeenidel. Sessiooni võtme alusel on võimalik SSO teenustest pärida kasutaja infot ja JWT(*Json Web Token*) võtit, mille alusel äriloogika kihis toimub autoriseerimine.

Haldusrakendusse sisenemisel tehakse kontroll, kas kasutajal on veebilehitsejas olemas küpsis. Küpsise alusel kontrollitakse SSO teenuste kaudu, kas sessioon on kehtiv ning kas kliendil on vajalik õiguste grupp rakenduse kasutamiseks. Vastasel juhul kuvatakse modaalis *iframe* kujul SSO klientrakendus, kus teenindaja saab sisse logida.



Joonis 11. Autentimise ja autoriseerimise järjestusdiagramm

Eduka autentimise puhul päritakse SSO teenustelt sessiooni võtme alusel JWT võti. See salvestatakse veebirakenduse seisuhoidlasse, ning pannakse igale järgnevale API päringule kaasa. Selle võtme alusel toimub äriloogika kihis autoriseerimine.

5.3.6 Pidev integratsioon ja paigaldus

Toimiv pidev integratsioon oli samuti üks uue haldusrakenduse mittefunktsionaalsetest nõuetest. Nende nõuete realiseerimiseks, koostöös meeskonna DevOps inseneriga, integreeriti uus rakendus olemasolevasse infrastruktuuri. Struktuuriüksuses on kasutusel avatud lähtekoodiga CI/CD server Jenkins ja rakendused paigaldatakse Kubernetesesse.

Autor kirjutas esialgse rakenduse Docker konteineri ehitamise skripti ja rakenduse ehitamise/paigaldamise konveierite skriptid. DevOps insener valideeris ning tegi mõningaid täiendusi. Näiteks lisas logimisega seotud funktsionaalsuse ja tegi koodi parandusi vastavalt rakenduse/keskkondade eripäradele. Paigaldus realiseeriti tuginedes varasemalt loodud baasfunktsionaalsusele, mis peitis kogu Kubernetesega seotud keerukuse abstraktsioonide taha.

Rakendusel on kokku neli konveierit: rakenduse ehitamine alam- ja juurharust, rakenduse juurutamine arendus-, test- ja toodangukeskkonda. Rakenduse ehitamise konveieri tööd algatab Bitbucket. Selleks kirjeldati Bitbucketisse päästikud, mis kutsuvad välja Jenkinsis

defineeritud veebihaake. Näiteks käivitatakse ehitamine iga koodi uuenduse (*commit*) korral. Rakenduse ehitamise konveieri sammud (vt Joonis 12) on järgmised:

- Vastavast koodirepositooriumi harust rakenduse koodi kloonimine.
- Vajalike NPM sõltuvuste paigaldamine.
- Paigalduseks sobiliku optimeeritud JavaScript komplekti (*bundle*) loomine.

Alamharu ehitamise kasulikkus seisneb selles, et see aitab valideerida koodi enne selle haru liitmist juurharuga. Juurharu ehitamise konveier läbib samu samme, aga sisaldab ka mitmeid lisategevusi:

- Rakenduse konteineri tõmmise ehitamine ja salvestamine Docker registrisse.
- Rsyslog konteineri tõmmise ehitamine ja salvestamine Docker registrisse.
- Rakenduse juurutamine ettevõtte taristusse.

Haldusrakenduse klient- ja serverrakendus on konteinerdatud samasse Dockeri tõmmisesse. Klientrakendust serveeritakse eelnevalt ehitatud komplektina läbi Nginx veebiserveri staatiliste failidena ning paralleelselt käivitatakse taustaprotsessina Node.js serverrakendus.

Rsyslog on logimissüsteem, mis on eriline oma suure jõudluse, skaleeritavuse ja paindlikkuse poolest. Seda kasutatakse konteineri logisõnumite edastamiseks ettevõttes kasutatavale logimissüsteemile Graylog.

Joonis 12. Edukalt läbitud haldusrakenduse juurharu ehitamise Jenkins konveier

Pidevjuurutuse konveier kasutab ehitamise käigus hoidlasse salvestatud Dockeri tõmmiseid ning annab käsu, sõltuvalt keskkonnast, vastavatele Kubernetese klastritele nende ehitamiseks. *Pod*-is käivitatakse paralleelselt rakenduse konteinerile ka Rsyslog konteiner, mis edastab logikirjeid. Ühtlasi on Rsyslogi kasutamisega täidetud ka logimise nõue.

6 Tulemus

Järgnev peatükk sisaldab ülevaadet töö käigus loodud rakendusest ja selle funktsionaalsusest. Lisaks kirjeldatakse rakenduse tulevikku ja sinna plaanitud arendusi.

6.1 Uus haldusliides

Töö käigus loodi uue haldusliidese MVP, mis kujutab endast klient- ja serverrakendusest koosnevat veebirakendust. Rakendus vastab peatükis 3 seatud funktsionaalsetele ja mittefunktsionaalsetele nõuetele.

Uues haldusliideses ei esine samu puuduseid, mis vana liidese analüüsi käigus välja tulid. Uus lahendus pärib infot selleks spetsiaalselt loodud teenustelt, millel puudub keerukas serverrakendus ja mis on realiseeritud tuginedes ettevõttes laialt kasutusel olevatele tehnoloogiate hiljutistele versioonidele. Samuti on kasutusele võetud TMF standard.

Seoses eelnevalt nimetatud edasiminekutega, on uus haldusliidest lihtsam arendada ja seeläbi odavam hooldada. Uue haldusliidese lõplikke vaateid on näha töö lisas 2.

6.2 Planeeritud edasiarendused

Kuna rakenduses realiseeritud funktsionaalsus on vaid väike alamhulk kogu olemasoleva haldusliidese funktsionaalsusest, ei vasta rakendus praegusel kujul kõigile kasutajate ootustele, mistõttu on vaja vana rakendust ka mõningat aega veel arendada ja toetada. Lõppeesmärgiks on täielik üleminek uuele haldusliidesele, mistõttu tuleb tulevikus jätkata funktsionaalsuse üle toomisega.

Esimene samm on lõpuni välja arendada kogu tellimustega seotud funktsionaalsus. Seejärel tuleb hakata koostöös teiste arendusmeeskondadega üle tooma teiste domeenidega seotud funktsionaalsust. Vana rakenduse saab sulgeda siis, kui uus rakendus on oma funktsionaalsuse poolest võrdväärne. Peale seda tekib võimalus realiseerida ka uut funktsionaalsust, mis võtab arvesse lõppkasutajate ootusi. Lisaks sellele tuleb rakendust täiendada automaattestidega, mille realisatsioon jäi lõputöö skoobist välja.

7 Tulemus

Lõputöö eesmärk oli moderniseerida Telia tellimuste töötlemise rakenduse haldusliides. Eesmärgi saavutamiseks viis autor esmalt läbi analüüsi olemasolevale rakendusele, kaardistades funktsionaalsuse, kasutatavad tehnoloogiad ning puudused.

Tulenevalt analüüsi tulemustest, valiti sobivaimaks moderniseerimise lähenemisviisiks uue rakenduse loomine. Töö mastaapsuse tõttu seati skoobiks luua uue rakenduse MVP, mis pakuks vaid väikest alamhulka kogu planeeritud funktsionaalsusest, kuid see-eest vastaks antud hetkel heakskiidetud ettevõttesisestele tehnoloogilistele normidele. Selleks, et kõrvaldada mõningaid puuduseide, mis olemasoleva rakenduse analüüsi käigus ilmsiks tulid, võttis autor töö skoopi ka mõningaid arendustöid äriloogika kihi serverrakendustesse. Need olid haldusliidesesse vajaliku funktsionaalsuse loomiseks möödapääsmatud.

Enne arendustegevust tuli paika panna rakenduse funktsionaalsed ja mittefunktsionaalsed nõuded. Kirjeldades ära rakenduse tulevase kuvandi arhitektuuri vaates, valiti realiseerimiseks kasutatud tehnoloogiad. Lõpuks pandi paika ka äriloogika kihi rakendustesse juurde tehtavad arendused.

Töö tulemusena valmis uus haldusliidese MVP veebirakenduse näol, mis pakub väikest alamhulka kogu vana haldusliidese funktsionaalsusest. Rakendusel on toimiv pidev integratsioon ja paigaldus ning on kättesaadav vaid Telia sisevõrgus selleks ettenähtud tehnilistele kasutajatele.

Autor plaanib jätkata ka peale lõputöö esitamist süsteemi edasiarendamist igapäevaste tööülesannete raames, lõppeesmärgiga täielikult asendada vana haldusliides.

Kasutatud kirjandus

[1] IBM, „How does order management work?," IBM, [Võrgumaterjal]. Available: https://www.ibm.com/topics/order-management. [Kasutatud 19 03 2023].

[2] IBM, „IBM Sterling Order Management," IBM, [Võrgumaterjal]. Available: https://www.ibm.com/products/order-management. [Kasutatud 23 03 2023].

[3] Ericsson Group, „Order Care," Ericsson Group, [Võrgumaterjal]. Available: https://www.ericsson.com/en/portfolio/cloud-software--services/digital-bss/order-care. [Kasutatud 19 03 2023].

[4] Shopify, „Shopify Plus," Shopify, [Võrgumaterjal]. Available: https://www.shopify.com/plus. [Kasutatud 19 03 2023].

[5] A. Rodriguez, „Restful web services: The basics," *IBM developerWorks,* nr 33, p. 18, 2008.

[6] S. Newman, „Pattern: Backends For Frontends," Sam Newman & associates, 18 10 2015. [Võrgumaterjal]. Available: https://samnewman.io/patterns/architectural/bff/. [Kasutatud 25 03 2023].

[7] „JHipster - Full Stack Platform for the Modern Developer!," JHipster, [Võrgumaterjal]. Available: https://www.jhipster.tech/. [Kasutatud 01 04 2023].

[8] „JHipster Domain Language," JHipster, [Võrgumaterjal]. Available: https://www.jhipster.tech/jdl/intro. [Kasutatud 16 04 2023].

[9] M. Thompson, „Discontinued Long Term Support for AngularJS," Google, 12 01 2022. [Võrgumaterjal]. Available: https://blog.angular.io/discontinued-long-term-support-for-angularjs-cc066b82e65a. [Kasutatud 01 04 2023].

[10] Amazon Web Services, „Database-per-service pattern," Amazon Web Services Inc, [Võrgumaterjal]. Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/database-per-service.html. [Kasutatud 01 04 2023].

[11] M. R. N. F. Z. D. Pramod Sadalage, Software Architecture: The Hard Parts : Modern Trade-off Analysis for Distributed ArchitecturesPramod Sadalage, Mark Richards, Neal Ford, Zhamak Dehghani, O'Reilly Media, 2021.

[12] Microsoft, „NuGet Gallery | Angular 1.6.4," Microsoft, [Võrgumaterjal]. Available: https://www.nuget.org/packages/angularjs/1.6.4. [Kasutatud 01 04 2023].

[13] OWASP, „OWASP Dependency-Check," OWASP, [Võrgumaterjal]. Available: https://owasp.org/www-project-dependency-check/. [Kasutatud 01 04 2023].

[14] Google, „Angular - Upgrading from AngularJS to Angular," Google, 28 02 2022. [Võrgumaterjal]. Available: https://angular.io/guide/upgrade. [Kasutatud 01 04 2023].

[15] R. Malan ja D. Bredemeyer, *Functional requirements and use cases,* Bredemeyer Consulting, 2001.

[16] L. Chung ja J. C. S. do Prado, „On non-functional requirements in software engineering," %1 *Conceptual modeling: Foundations and applications: Essays in honor of john mylopoulos,* Springer Berlin, 2009, pp. 363-379.

[17] C. Song, „TypeScript vs JavaScript: What's the difference?,“ [Võrgumaterjal]. Available: https://www.educative.io/blog/typescript-vs-javascript-whats-the-difference. [Kasutatud 02 04 2023].

[18] B. Michel, „What is a transpiler ?,“ Bam.tech, 29 04 2015. [Võrgumaterjal]. Available: https://www.bam.tech/article/what-is-a-transpiler. [Kasutatud 02 04 2023].

[19] A. MacCaw, The Little Book on CoffeeScript, O'Reilly Media, 2012.

[20] Microsoft, „TypeScript: Documentation - TypeScript for JavaScript programmers,“ Microsoft, [Võrgumaterjal]. Available: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html. [Kasutatud 02 04 2023].

[21] B. O'Grady, „What is a JavaScript library?,“ Codeinstitute, [Võrgumaterjal]. Available: https://codeinstitute.net/global/blog/what-is-a-javascript-library/. [Kasutatud 08 04 2023].

[22] R. Toal, „What is a JavaScript framework?,“ CodeInstitute, [Võrgumaterjal]. Available: https://codeinstitute.net/global/blog/javascript-framework/. [Kasutatud 08 04 2023].

[23] M. Sharapova, „Choosing a Javascript framework: Top JS frameworks comparison,“ Gearheart, 27 04 2022. [Võrgumaterjal]. Available: https://gearheart.io/articles/how-to-choose-the-best-javascript-framework-comparison-of-the-top-javascript-frameworks/. [Kasutatud 08 04 2023].

[24] tkrotoff, „Front-end frameworks popularity (React, Vue, Angular and Svelte),“ 08 04 2023. [Võrgumaterjal]. Available: https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190. [Kasutatud 08 04 2023].

[25] T. Nyugen, „What's the easiest front-end framework for beginners?,“ Frontendmag, 18 01 2023. [Võrgumaterjal]. Available: https://www.frontendmag.com/insights/easiest-front-end-framework. [Kasutatud 15 04 2023].

[26] L. S. Vailshery, „Most used languages among software developers globally 2022,“ Statista, 09 08 2022. [Võrgumaterjal]. Available: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/. [Kasutatud 15 04 2023].

[27] M. Choubey, „Express vs Springboot: Hello world performance comparison,“ 23 02 2023. [Võrgumaterjal]. Available: https://medium.com/deno-the-complete-reference/express-vs-springboot-hello-world-performance-comparison-dd066bf53858. [Kasutatud 19 04 2023].

[28] Meta, „Thinking In React,“ Meta, [Võrgumaterjal]. Available: https://react.dev/learn/thinking-in-react. [Kasutatud 09 04 2023].

[29] V. Gavrilă, L. Băjenaru ja C. Dobre, „Modern single page application architecture: a case study,“ *Studies in Informatics and Control,* nr 28.2, pp. 231-238, 2019.

[30] „Express - A Node.js web application framework,“ OpenJS Foundation, [Võrgumaterjal]. Available: https://expressjs.com/. [Kasutatud 15 04 2023].

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

Mina, Ranar Leisson

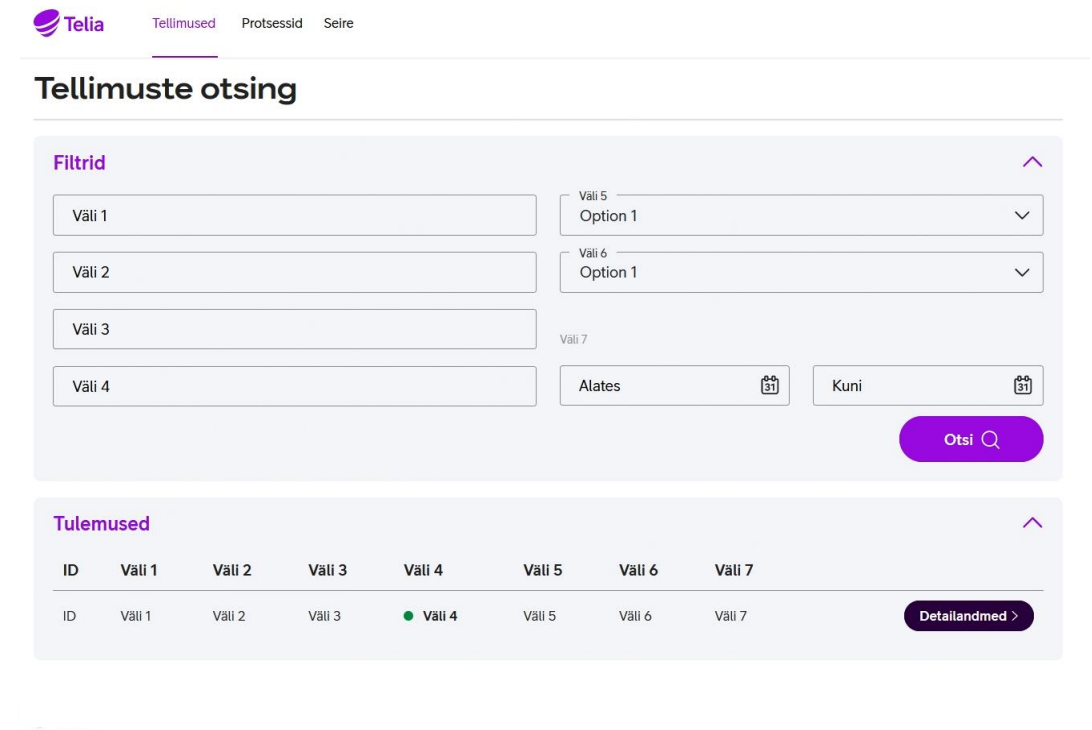
1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Telia Eesti AS tellimuste töötlemiste süsteemi tehnilise kasutajaliidese moderniseerimine“, mille juhendajateks on Meelis Antoi ja Rain Vink
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, 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.

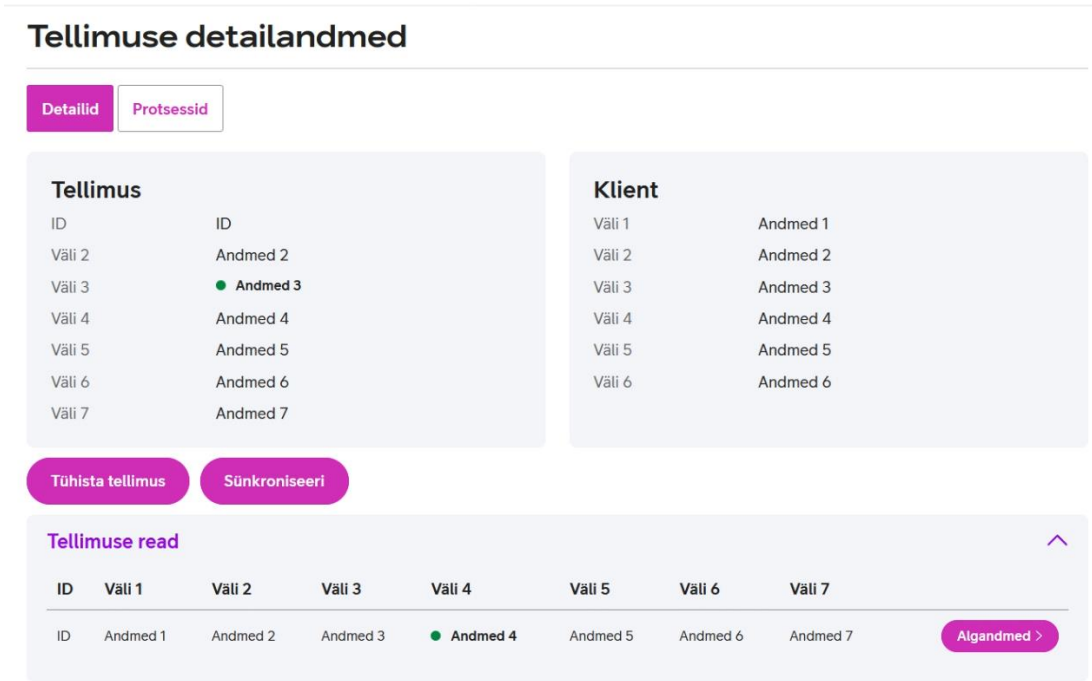
15.05.2023

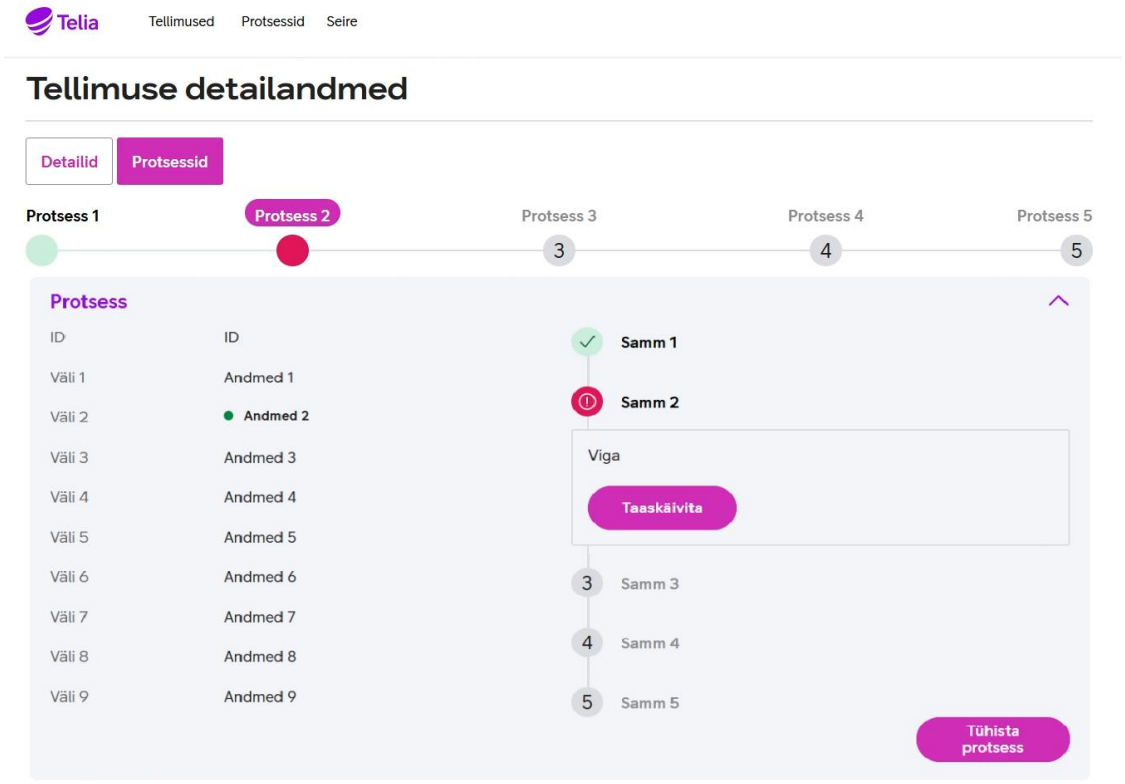
1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või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

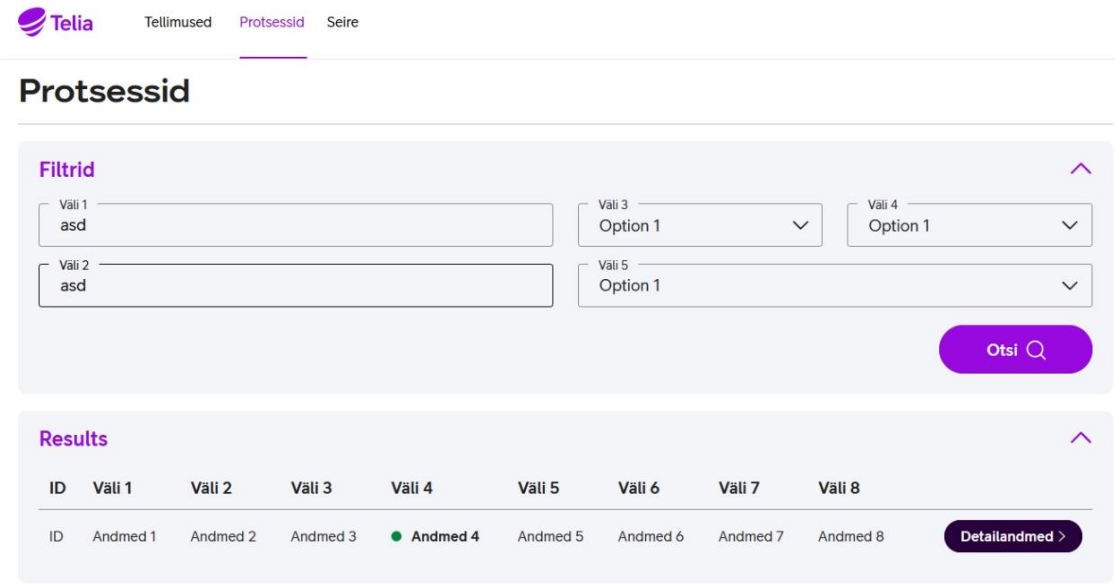
Lisa 2 – Uue haldusrakenduse vaated

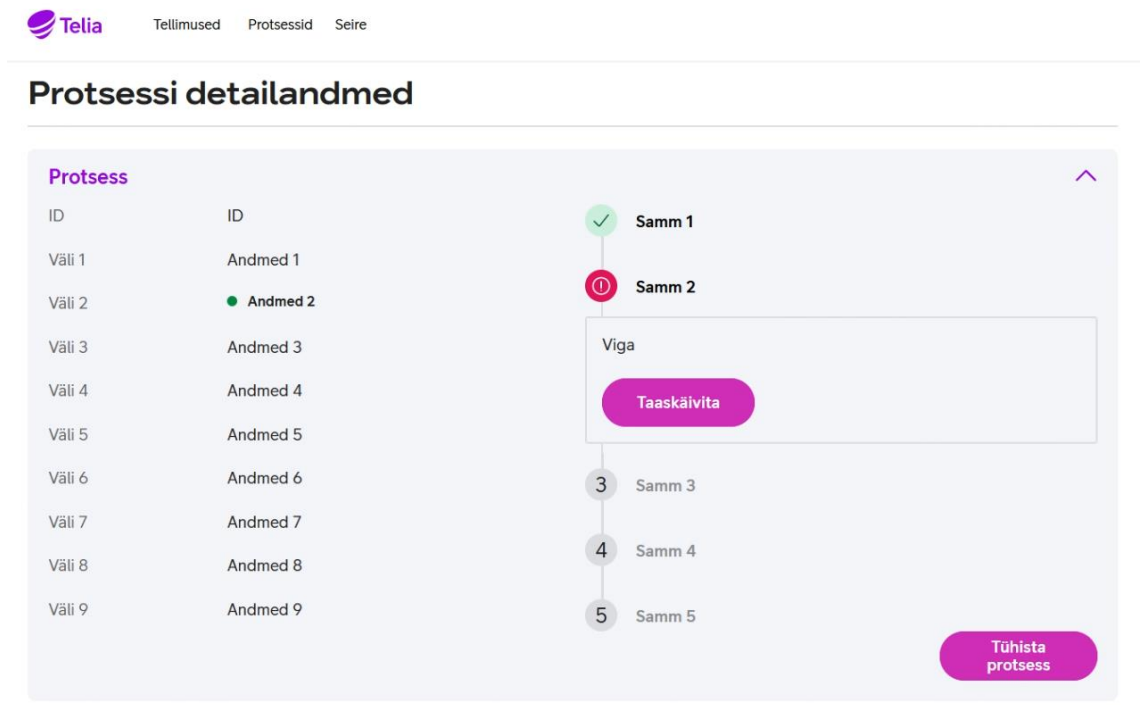
Antud näidetes on andmed anonüümseks tehtud ning nende kuju peidetud.

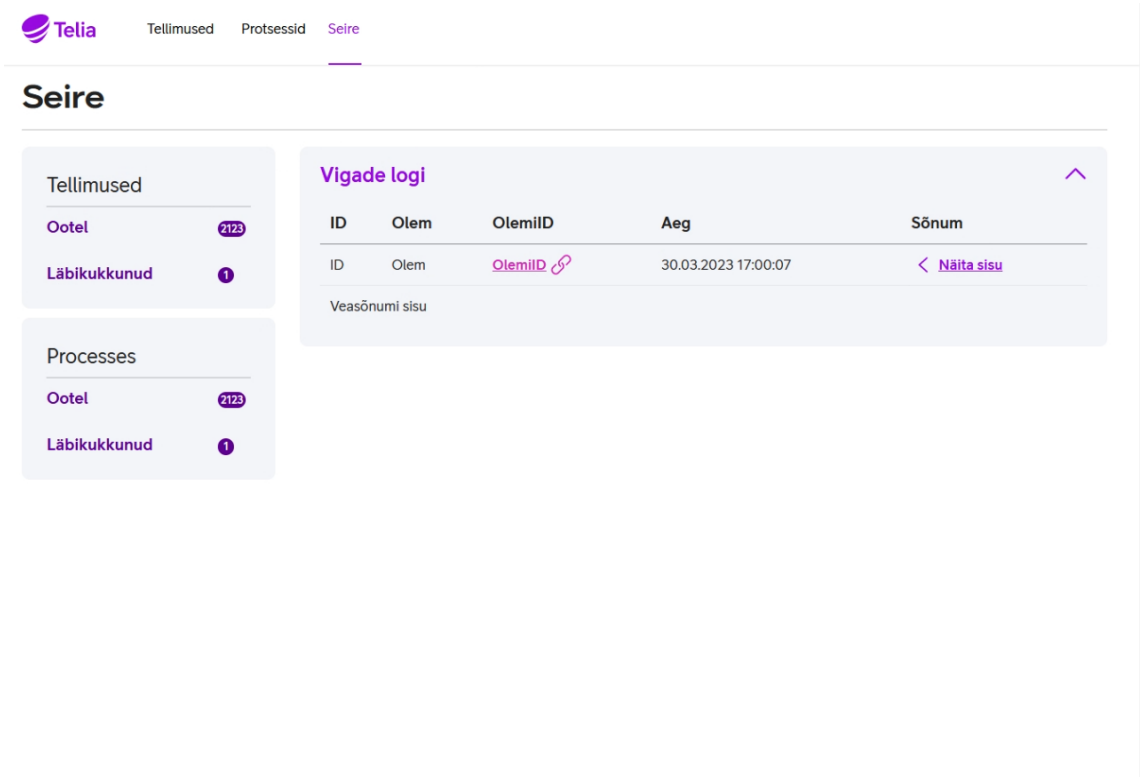





Telia
Tellimused
Protsessid
Seire
Tellimuse detailandmed
Detailid
Protsessid
Protsess 1
Protsess 2
Protsess 3
Protsess 4
Protsess 5
3
4
5
Protsess
ID
ID
Väli 1
Andmed 1
Väli 2
Andmed 2
Väli 3
Andmed 3
Väli 4
Andmed 4
Väli 5
Andmed 5
Väli 6
Andmed 6
Väli 7
Andmed 7
Väli 8
Andmed 8
Väli 9
Andmed 9
Samm 1
Samm 2
Viga
Taaskäivita
3
Samm 3
4
Samm 4
5
Samm 5
Tühista protsess


Telia
Tellimused
Protsessid
Seire
Protsessid
Filtrid
Väli 1
asd
Väli 3
Option 1
Väli 4
Option 1
Väli 2
asd
Väli 5
Option 1
Otsi
Results
ID
Väli 1
Väli 2
Väli 3
Väli 4
Väli 5
Väli 6
Väli 7
Väli 8
ID
Andmed 1
Andmed 2
Andmed 3
Andmed 4
Andmed 5
Andmed 6
Andmed 7
Andmed 8
Detailandmed >


Telia
Tellimused
Protsessid
Seire
Protsessi detailandmed
Protsess
ID
ID
Väli 1
Andmed 1
Väli 2
Andmed 2
Väli 3
Andmed 3
Väli 4
Andmed 4
Väli 5
Andmed 5
Väli 6
Andmed 6
Väli 7
Andmed 7
Väli 8
Andmed 8
Väli 9
Andmed 9
Samm 1
Samm 2
Viga
Taaskäivita
3
Samm 3
4
Samm 4
5
Samm 5
Tühista protsess


Telia
Tellimused
Protsessid
Seire
Seire
Tellimused
Ootel
2123
Läbikukkunud
1
Processes
Ootel
2123
Läbikukkunud
1
Vigade logi
ID
Olem
OlemiID
Aeg
Sõnum
ID
Olem
OlemiID
30.03.2023 17:00:07
Näita sisu
Veasõnumi sisu