TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Valter-Kaspar Karolin 200812IADB

Metaprint AS-i komponenditehase tootmisinfosüsteemi arhitektuuri uuendamine

Bakalaureusetöö

Juhendaja: Märt Kalmo
MSc

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: Valter-Kaspar Karolin

28.12.2022

Annotatsioon

Tarkvara arengu üks paratamatustest on süsteemide vananemine ning vahel ka aegumine. Põhjuseid on mitmeid, mis enamjaolt tulenevad riistvaralistest muudatustest, äriliste nõuete pidevast muutumisest, kasutusel olevate tehnoloogiate toe kadumisest või uuenemisest. Hästi toimiva tarkvarasüsteemi ülalpidamine nõuab pidevat arendust ning kaasajastamist.

Sarnast kaasajastamist vajab ka Metaprint AS komponenditehase tootmisinfosüsteem. Kasutuslugu on aastate jooksul oluliselt muutunud. Olemasolev tehnoloogiline lahendus ei võimalda ressursitõhusalt muutunud nõudeid lahendada. Efektiivse arendamise peamiseks probleemiks on olemasoleva lahenduse arhitektuur. Antud bakalaureusetöö käsitleb tarkvara arhitektuuri ja koodibaasi uuendamist antud ettevõtte näitel.

Arendusuuringu käigus selgitatakse olemasoleva tarkvarasüsteemi puudused, analüüsitakse võimalikke lahendusviise, leitakse sobilikke tööriistu ja luuakse prototüüp mis lahendab käsitletud puudused ning võimaldaks tulevikus hõlpsasti integreerida ülejäänud ettevõtte infosüsteemi koos erinevate osakondade moodulitega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, viis peatükki, üheksa joonist, üks tabel.

Abstract

Updating Production Information System Architecture of the Component Plant for Metaprint AS

One of the side effects of software development is the inevitable aging of systems and sometimes also obsolescence. There are several reasons, mostly due to hardware changes, constant changes in business requirements, loss of support or renewal of the technologies in use. Maintaining a well-functioning software system requires constant development and modernization.

The production information system of the Metaprint AS components factory also needs a similar modernization. The use case has changed significantly over the years. The existing technological solution does not allow to solve the changed requirements in a resource-efficient manner. The main problem for effective development is the architecture of the existing solution. This bachelor's thesis deals with software architecture and updating the code base on the example of this company.

During the development study, the shortcomings of the existing software system are explained, possible solutions are analyzed, suitable tools are found, and a prototype is constructed that will solve the discussed shortcomings and enable easy integration of the rest of the company's information system with modules from different departments in the future.

The thesis is in Estonian and contains 27 pages of text, five chapters, nine figures, one table.

Lühendite ja mõistete sõnastik

Apache HTTP Server	avatud lähtekoodiga HTTP-server.
API	*Application Programming Interface* ehk rakendusliides.
CSS	*Cascading Style Sheets* on küljendamisel kasutatav märgistuskeel milles määratletakse üles peamiselt veebilehtede kujundust.
Excel	Microsoft Excel on tabelarvutus- ja tabeltöötlusprogramm.
HTML	*HyperText Markup Language* on hüperteksti märgendite keelt kasutatakse veebilehtede loomiseks.
HTTP	*Hypertext Transfer Protocol* on protokoll teabe edastamiseks arvutivõrkudes.
JavaScript	Objektorienteeritud programmeerimiskeel, peamiselt kasutatud veebiarenduseks.
MVC	*Model-View-Controller* ehk Mudel-Vaade-Kontroller on tarkvara ülesehituse arhitektuur.
PHP	Populaarne üldotstarbeline skriptikeel, mis sobib hästi veebiarendusteks.
Polling / Long Polling	Perioodiline andmete kogumise meetod.
Pärandkood	Lähtekood, mis on tavaliselt tehnoloogiliselt või struktuuriliselt vananenud ning seda on riskantne muuta ilma võimalike vigade tekitamiseta.
Regex	*Regular Expression* ehk regulaaravaldis on informaatikas sümbolitest koosnev otsingumuster.
SSE	Server-Sent Events – andmevahetusprotokoll kus server saadab kliendile andmeid aga mitte vastupidi.
URI	*Uniform Resource Identifier* - on ühtne ressursi-indikaator, mis identifitseerib ressurssi nime alusel.
URL	*Uniform resource locator* - ühene aadress, mida kasutatakse infoallikate viitamiseks ja leidmiseks internetis.
Webpack	Tasuta ja avatud lähtekoodiga JavaScripti moodulite komplekteerija.
WebSockets	Täisdupleks andmevahetusprotokoll.

Sisukord

Jooniste loetelu

Tabelite loetelu

Sissejuhatus

Tarkvaraarenduse vältimatu osa on olemasoleva tarkvara või tarkvarasüsteemi hooldamine ja uuendamine. Pärandkoodiks muutumine ei sõltu alati ainult tehnoloogia vananemisest. Rolli mängivad uute nõuete pidev täiendamine ja integreerimine, keerukuse kasvamine, erinevate inimeste panus, programmeerimise oskuste ja erinevate tavade kasutamine, arenduse käigus vajalike tehnoloogiate olemasolu ning nende kasutusoskus.

Halvasti koostatud või keeruliseks kasvanud tarkvara uuendamine on piinarikas protsess, kus mängivad rolli ajalised viited, ebakindlad tulemused, võimetus planeerida tänu ootamatutele tõrgetele uuenduste käigus kui ka tarkvara eripärasustele.

Teema on aktuaalne, sest tänapäeval leidub kindlasti palju ettevõttesiseseid süsteeme mis on sarnaste vahenditega arendatud või kannatavad sarnaste probleemide all ja on muutunud pärandvaraks. Töö võib pakkuda huvi neile, kes puutuvad kokku pärandsüsteemidega ja kaaluvad nende uuendamist. Töös käsitletakse pärandsüsteemist migratsiooniga seotud probleeme, lahenduste otsimist ja ettevõetud samme.

Antud bakalaureusetöös lahendatakse uuendamistega seotud probleeme. Kirjeldatakse rakenduse kitsaskohti, analüüsitakse võimalikke lahendusi, luuakse prototüüp, mis lahendaks olemasolevad probleemid ja võimaldaks tulevikus kiiremat arendust ning rakenduse uuendamist.

1 Ülesande püstitus ja taust

Käesolevas peatükis antakse ülevaade olemasolevast süsteemist, sõnastatakse probleemid ja kirjeldatakse probleemidest vabanemiseks võetud samme ning metoodikat.

1.1 Olemasoleva süsteemi ülevaade

Ettevõttes Metaprint AS, kus autor töötab, toodetakse trükitud kolmeosalisi aerosoolpudeleid. Ettevõtte meeskonda kuulub üle 400 inimese erinevates tootmisüksustes Eestis, Venemaal ja Hollandis. Üks põhilisi protsesse on erinevate aerosoolpudelite komponentide tootmine (kettakujulised otsad ja põhjad). Komponentide tootmistehases on mitu füüsilist liini kuhu sisestatakse pinnatöötlusega toorplekk. Plekist valmistatakse stantsimise teel komponendid, mis iga liini lõpus pakitakse alustele, markeeritakse ja saadetakse erinevatesse tootmisüksustesse pudelite tootmiseks. Erineva pinnatöötlusega komponente on tänaseks üle 50 erineva tüübi ning nende tootmise planeerimine on keeruline protsess. Olemasolev veebirakendus võimaldab erinevatel kasutajatel:

- tootmisplaanis erinevate komponenditüüpide tootmise planeerimist ja materjalikulu arvestamist;
- operaatori vaates materjali registreerimist, tootmistulemuste registreerimist ja mõõtmistulemuste sisestamist;
- toodangu markeerimiseks vajalike etikettide tekitamist;
- transpordikoormate moodustamist ja saatelehtede tekitamist;
- raportite loomist kogutud andmete põhjal.

Tänu pidevalt arenevatele tootmisprotsessidele vajab arendust ka toetav tarkvara.

1.2 Probleemi kirjeldus

Olemasolev tarkvara lahendus töötati komponenditehase tööks välja 2013 aastal, mil tehases oli kaks tootmisliini. Tarkvaralahenduse väljatöötamisel eeldati, et tehase operatsioonid suurel määral ei kasva. 2022 aastaks on tehasesse lisandunud kolm tootmisliini, olles nüüdseks viie tootmisliiniga tehas.

Komponenditehase protsesside muutuste ja arenguga sammu pidava tarkvaralahenduse arendus on muutunud tülikaks ning ei ole kuluefektiivne. Uuenduste planeerimine ja rakendamine on töömahukas ja aeganõudev protsess ning tulemused ei ole alati garanteeritud. Tänu sellele on osad vajalikud tööriistad uuendamata ja tööd tehakse paralleelselt *Exceli* abiga. *Microsoft Excel* on tabelarvutus- ja tabeltöötlusprogramm. *Exceliga* tehtud tööle kulub palju aega, inimressurssi ja andmetele puudub tsentraalne ligipääs ning neid ei ole võimalik operatiivselt kasutada ettevõtte töö planeerimisel. Uuendamiste või arenduste käigus on tekkinud rikkeid rakenduse erinevates punktides.

Pideva veebilehitsejate arenduse käigus vananenud või eemaldatud funktsionaalsuste tõttu juhtub aina enam situatsioone, kus lehitseja uuenduste tõttu lakkavad töötamast osad funktsionaalsused, takistades veebirakenduse korrektset toimimist. Nende probleemide lahendamine on aeganõudev tegevus, kuna rakendusele ei ole võimalik süsteemselt läheneda ja lahendus on tihti ajutine.

Administraatori jooksutatud haavatavuse raportite põhjal on probleemiks ka turvalisus.

1.3 Eesmärk

Käesoleva töö eesmärgiks on Metaprint AS komponenditehase infosüsteemi uuendamine. Vajalik on analüüsida tarkvarasüsteemi, tuvastada probleemsed kohad, leida probleemsetele kohtadele võimalikud lahendused ja luua antud lahenduste põhjal sobiv prototüüp.

Loodud prototüüp peab võimaldama integratsiooni ettevõtte strateegilisse infosüsteemi, reaalajalist andmete sünkroniseerimist kasutajaliidestes ja võimaldama konfigureerimist, modulaarset edasiarendamist ja uuendamist.

1.4 Metoodika

Arendusuuringu käigus sõnastatakse probleem, analüüsitakse käesolevat rakendust ja määratletakse süsteemi kitsaskohad. Probleemi lahendamiseks analüüsitakse ja võrreldakse erinevate tarkvaraarhitektuuride mustreid ning valitakse sobivaim. Järgmiseks uuritakse sobilikke tööriistu ja tehnoloogiaid millega rakenduse prototüüp luua. Viimaks võrreldakse vastavust seatud eesmärkidega.

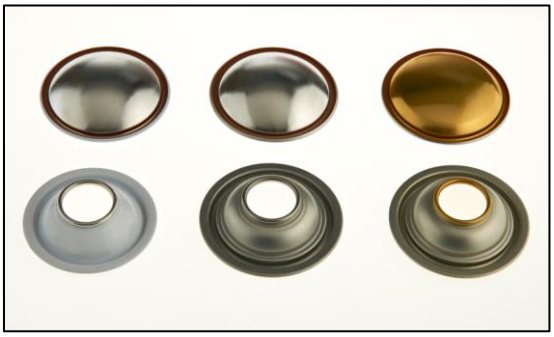
2 Olemasoleva süsteemi kirjeldus ja vigade analüüs

Tarkvara disainimise juures on esmalt selgeks vaja teha milliseid nõudeid peab rakendus ja selle arhitektuur rahuldama. Antud peatükis analüüsitakse rakenduse käitumist koos nõuetega.

2.1 Rakenduse põhifunktsionaalsus

Rakenduse eesmärk on luua uusi ja uuendada vanu võimalusi manuaalse töömahu vähendamiseks ning veakindluse tõstmiseks. Rakenduse peamine funktsioon on hõlbustada tootmise planeerimist, toodangu registreerimist ja parendada kvaliteedimõõtmiste käsitlust. Algselt lihtsakoeliselt disainitud rakenduses ei osatud eeldada, et tegu on keerulise protsessiga. Süsteemi projekteerimisel tehtud valearvestuste tõttu on hilisemalt vaja teha palju arhitektuurilisi muudatusi [1].

Algselt planeeriti tootmiseks kaks liini, millest ühes sai toota otsa komponente ja teises põhja komponente. Joonisel 1 on esitatud ülemises reas põhjad ja alumises reas otsad.



Joonis 1. Komponendid - otsad ja põhjad.

Tootearenduse käigus on lisandunud erinevate diameetrite, tüüpide, paksuste ja pinnatöötlustega komponendid. Mõned näited:

- NI – *necked-in*, SS – *straight side* ja hilisemalt lisandus HT – *high top;*
- erinevad diameetrid: 65mm, 57mm, 52mm;
- erinevad paksused: 0.15mm , 0.18mm ja 0.205mm;
- erinevad pinnatöötlused: disainitrükk koos erinevate lakkide variatsioonidega.

Kasutades empiirilist lähenemist selgitati küsitluste käigus peamised rakenduse probleemid. Selgusid järgmised punktid:

1. Uute liinide lisamine vanasse arhitektuuri on keeruline, kuna:
    - Olemasolevas lahenduses puudub toodete parametriseerimine. Tarkvara identifitseerib tooteid otsides toote nimetusest võtmesõnu, mis eeldab kindla süntaksi järgimist toodete nimetamisel.
    - Puudub liinide parametriseerimine. Identifitseeritud on ainult üks otsaliin ja üks põhjaliin. Lahendus ei võimalda identifitseerida kolme erinevat otsaliini ja kahte erinevat põhjaliini.
2. Oluliselt on edasi arenenud ka arusaamine toote kvaliteedist ja selle mõõtmisest:
    - „Ümmarguse“ komponendi mõõtmisel peab lisaks diameetrile hindama ka selle ovaalsust (kolmest erinevast kohast mõõdetud tulemuse haare).
    - Seoses uute liinide erineva konfiguratsiooniga on suurenenud ka valimite hulk. Kui eelnevalt on olnud igal liinil kaks pastapüstolit millest mõlemast peaks proovi võtma, siis ühel uuel liinil on pastapüstolite arv 8. See nõuab oluliselt teistsugust kasutajaliidese disaini andmete sisestamiseks.
    - Liinil tehtavate mõõtmiste hulk on kasvanud – nende andmete registreerimise jaoks tuleb muuta andmemudelit ja kasutajaliidest.

Tulevikus ei ole välistatud uute liinide lisandumine või olemasolevate vahetumine.

2.1.1 Tootmisplaan

Tootmisplaan on tabel, mille peamisteks ülesanneteks on võimaldada tootmisjuhil lisada ja vastavalt tähtaegadele planeerida ning järjestada töid. Tabelis iga rida on ühte tüüpi komponendi tootmise jaoks vormistatud töökäsk, mis sisaldab toote nimetust, sobilikku tootmise materjali, kogust, tähtaega ja muud tööga seonduvat informatsiooni.

Antud vaate puhul on probleemiks uue töö lisamine nimekirja. Süsteemi registreeritud artiklite kohta tehakse sõnetöötlusega päring, kus otsitakse näiteks kõik artiklid mille nimetuse väljas esineb kusagil sõne osa „OTS“. See on halb lähenemine, mille tõttu tekib erinevaid vigu. Esiteks eeldab see, et artiklite sisestusel ollakse hoolas ja sisestatakse

nimetused korrektselt vastavalt kokkulepetele. Sarnaselt toimub ka seostamine pinnatöötlusega, näiteks sõneosade „VALGE“ või „HALJAS“ järgi. Kui sisestaja artikli nime registreerimisel teeb kirjavea, ei teki sobilik toode nimistusse ja toodet ei saa tootmisesse lisada.

Kasutusjuhtu annab parandada toodete parametriseerimisega. Iga toote artiklikaardile lisatakse tüüp, diameeter, seotud materjalid ja võimalikud pinnatöötlused. Need lepitakse eelnevalt kokku ja artikli registreerimisel käib omaduse lisamine nimekirja rippvaliku põhjal vastupidiselt vabas tekstis kirjutamisele.

2.1.2 Operaatori vaade

Operaatorite igapäevaseks tööks ettenähtud vaade, mille eesmärk on kuvada valitud töö sooritamiseks vajalik informatsioon. Lisaks peab vaade võimaldama operaatoril registreerida töösse materjale. Kuna siiani on puudunud võimalus kontrollida sisestatud materjali sobivust tootmises oleva komponendiga, on raske tootmisvigu ennetada või hallata. Eraldi on loodud administraatorile vaade, kus saab sisestatud vigu parandada. Selline lähenemine on reaktiivne mitte ennetav ja ebaefektiivne. Seetõttu vajab vaade refaktoreerimist koos kontrollide sätestamisega.

Operaatori kohustuste hulka kuuluvad lisaks mõõtmised. Algne mõõtmiste protsess eeldas, et iga erineva valmis toodetud komponendi aluse kohta tehakse üks mõõtmine. Komponendi mõõtmine tähendab, et liinist võetakse töö käigus komponent ja mõõdetakse tema erinevaid omadusi spetsiifiliste tööriistadega. Erinevaid mõõtmispunkte on umbkaudu 15, olenevalt komponendist. Vormi sisestatakse erinevad väärtused, mis salvestatakse andmebaasi. Ühes reas on valmistoodangu aluse number, 15 parameetrit ja kuupäev.

Hilisemalt otsustati, et mõõtmiste hulk ei ole piisav ja mõõtmisi tuleks läbi viia tihedamalt. Lisandunud on ka mõõtmiste arv punkti kohta. Kuna komponent ise on ümmargune, ei saa serva kõrgust mõõta ainult ühest kohast, vaid tuleb mõõta kolmest kohast ja vastavalt tulemustele saab arvutada keskmise, mille abil saab märgata kriitilisi erilisusi tootmisprotsessis. Praeguses süsteemis sellele tugi puudub.

2.2 Sündmuste haldamine

Üks murekohtasid olemasolevas rakenduses on andmete uuendamine. Enamus vaadetes on kasutusel *Polling* ehk teisisõnu kindla intervalliga päringute tegemine andmete uuendamiseks [2]. Osades vaadetes uuendatakse tervet lehte, teistes osaliselt vajalikku informatsiooni.

Eelneva meetodi probleem on selles, et osad andmed uuenevad harva ja mõned päringud võivad olla mahukad või keerulised ja nende tühipaljas pärimine teeb rakenduse kasutamise kogemuse ebameeldivaks ning aeglaseks. Keeruliseks teeb see vaadete sünkroniseerimise. Juhul kui kahes erinevas vaates on elemendid, mis sõltuvad üksteisest ja peaksid muutma oma staatust või kujundust vastavalt kindlale olekule, võib juhtuda *Polling* meetodi puhul nii, et halva ajastuse tõttu ei uuendata vajalikku elementi õigel ajal. Sellest võib tuleneda vääri andmesisestusi või andmete kuvamisi.

Komponenditehasesse lisandusid hiljuti füüsiliste liinide juurde sisend- ja väljundkioskid. Kioskid on puutetundliku ekraaniga arvutid. Sisendkioski peamine eesmärk on kuvada ühte vaadet, kus võimaldatakse kasutajatel registreerida materjalikulu või valmistoodangut. Väljundkioskis saab töö staatuse seada lõpetatuks, mille puhul peab olema sisendkioskis keelatud antud töö juurde lisamaterjali registreerimine.

Käime läbi järgmise stsenaariumi. Esimene kiosk küsib *Polling* meetodiga serverilt uuendusi. Järgmisel minutil tehakse teises kioskis tegevus, näiteks suletakse töö. Kui esimese kioski andmete uuendamiste intervall on seatud kuueks minutiks, ei pruugi töö uuenemise info sinna jõuda enne seitsmendat minutit - mis puhul ei uuendata kioski andmeid või elemente õigeaegselt. Sarnaseid juhtumeid võib kohata kui tööde nimekirja lisatakse olemasolevate andmete põhjal töö. Operaator avab töö ja talle kuvatakse staatiline leht töö infoga. Kui tootmisjuht avastab, et tegi sisestamisel vea ja teeb töökäsus korrektuurid aga tootmist on juba alustatud, ei uuene operaatoril informatsioon ja seetõttu ei ole võimalik kontrolle ja hoiatusi rakendada kuna võrdlused tehakse vanade andmete põhjal.

Pollingu alternatiiviks oleks *Long Polling*, mis erineb eelmisest selle poolest, et server ei sulge ühendust pärast kliendilt päringu saamist, vaid server vastab ainult siis, kui saadaval on mõni uus sõnum või saavutati ajalõpu piir. Antud meetodiga väheneb päringute hulk aga lisanduvad mitmed probleemid. Antud lahendus ei skaleeru ja loob sarnaselt

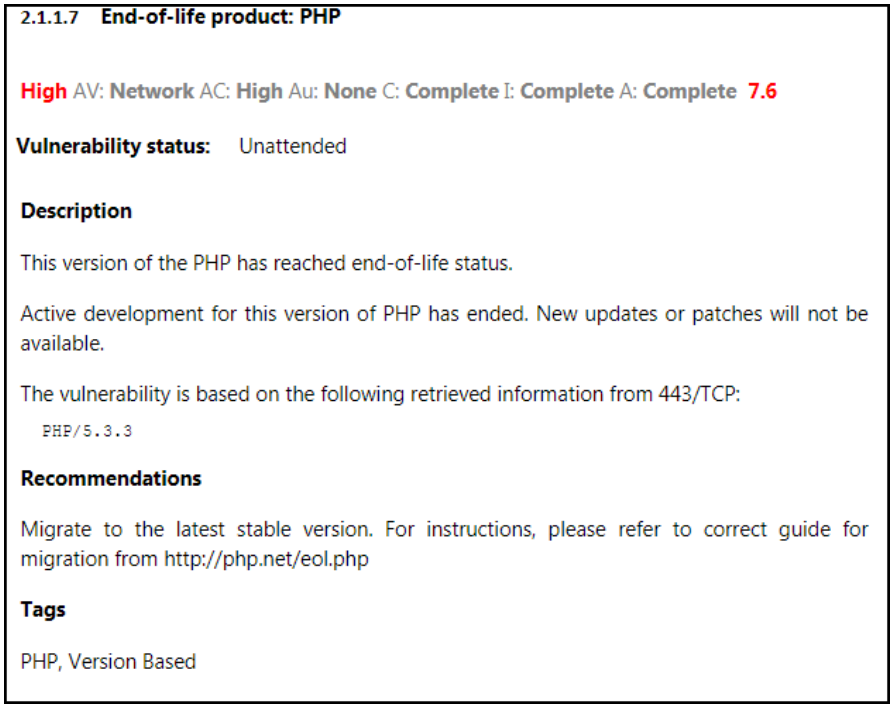
esimesele iga päringu puhul uue ühenduse [3]. Jätkuvalt on sünkroniseerimise probleem erinevate vaadete vahel.

Üks erineva käitumismustriga lahendusviis on WebSockets protokoll, mis ühenduse loomisel võimaldab kahesuunalist liiklust. Ühendust ei suleta ning kumbki pool ootab järgmist sõnumit. Selline ühendus annab võimaluse serveril saata uuendusi kliendile ja vastavalt uuenduste saamisele saab klient uuendada vaadet või valitud elemente ning nendes sisalduvat informatsiooni. Sarnaselt saab klient sama ühendust kasutades saata sõnumeid serverile.

SSE ehk *Server-Sent Events* lubab sarnaselt WebSocketile serveril saata sõnumeid kliendile aga vastupidine kommunikatsioon ei ole võimalik. Antud lahendus võib osutuda probleemiks kuna lubab kuus ühendust korraga brauseri kohta. Kui kasutajatel on mitmeid veebilehitseja vahekaarte lahti, võib tekkida probleeme [4].

2.3 Turvalisus

F-Secure Radar on turvaaukude kontrollimise ja haldamise platvorm, mis võimaldab tuvastada ja hallata rakenduse sisemisi ja väliseid ohte [5]. Joonisel 2 on esitatud näide ühest haavatavusest.


2.1.1.7 **End-of-life product: PHP**

High AV: **Network** AC: **High** Au: **None** C: **Complete** I: **Complete** A: **Complete** **7.6**

Vulnerability status: Unattended

Description

This version of the PHP has reached end-of-life status.

Active development for this version of PHP has ended. New updates or patches will not be available.

The vulnerability is based on the following retrieved information from 443/TCP:

```
PHP/5.3.3
```

Recommendations

Migrate to the latest stable version. For instructions, please refer to correct guide for migration from http://php.net/eol.php

Tags

PHP, Version Based

Joonis 2. F-Secure Radar raporti üks punktidest.

Administraatori läbi viidud F-Secure Radar riskianalüüsi raportis esineb erinevaid haavatavusi. Kokku 9 kõrge, 24 keskmise ja 22 madala riskitasemega probleemi. Osad on tingitud sama põhjuse tõttu ja seetõttu saab need kokku grupeerida.

Kriitilistest haavatavustest on peamised vananenud operatsioonisüsteem, vananenud Apache HTTP veebiserveri tarkvara ja vananenud kasutusel olev PHP programmeerimisekeele versioon. Sellises olukorras ei piisa vaid rakenduse refaktoreerimisest, vaid vajalik on muuta ka rakenduse jooksutamiseks vajalikke aluskihte, mis omavahel mõnel määral seotud on. Operatsioonisüsteemi ja Apache veebiserverit saaks uuendada jooksvalt, luues uue virtuaalserveri, paigaldades sinna uuema veebiserveri ja tõestes üle tarkvara failid. Probleeme tekitab PHP versiooni uuendamine, mida ei ole võimalik teha ilma tõrgeteta rakenduse töös.

2.4 Arhitektuuri ülevaade

Algselt ühe arendaja kõrvaltööna valminud PHP veebirakendus sai alguse MVC raamistiku “Kohana” põhjal, mille tugi praeguseks on aegunud. Mudel-vaade-kontroller (MVC) on tarkvaraehituse arhitektuur, mille kihid täidavad erinevaid ülesandeid. MVC rakenduses on URL aadressidele saabuvad nõuded suunatud mitte serveris olevatele failidele, vaid kontrolleri toimingule ehk toimingumeetodile. Kontroller on klass, mis sisaldab toimingumeetodeid (action methods) ehk toiminguid (actions). Need toimingud vastutavad autoriseerimise, mudelitest andmete pärimise ja lõppandmete vaadetele suunamise eest. Vaates tekitatakse andmete põhjal malliga tavaliselt HTML kood mida siis veebilehitsejale tagastatakse kuvamiseks. [6]

Peatüki esimeses punktis kirjeldatud tootearenduste muudatuste käigus on muutnud andmemudel ja muudatusi on keeruline kasutajaliidesesse sisse viia. Kuna puudub dokumentatsioon ja rakenduse algne autor ei ole projekti kallal töötanud üle 8 aasta, peab süsteemi uurima läbi küsitluste ja tegema visuaalse uuringu.

Autori uuringu tulemusena selgub, et raamistiku paigaldamise käigus seatud faili- ja kaustastruktuuri kõrvale on loodud eraldiseisev kaust, milles sisalduvad mõned alamkaustad ja hulk PHP faile. Need failid on struktureerimata ja tihti sisaldavad segadust tekitavaid nimesid. Kui faile hakata korrastama ja paigutama moodulite kaupa gruppidesse, rakenduse funktsionaalsus katkeb, kuna failidele viitamine käib failitee

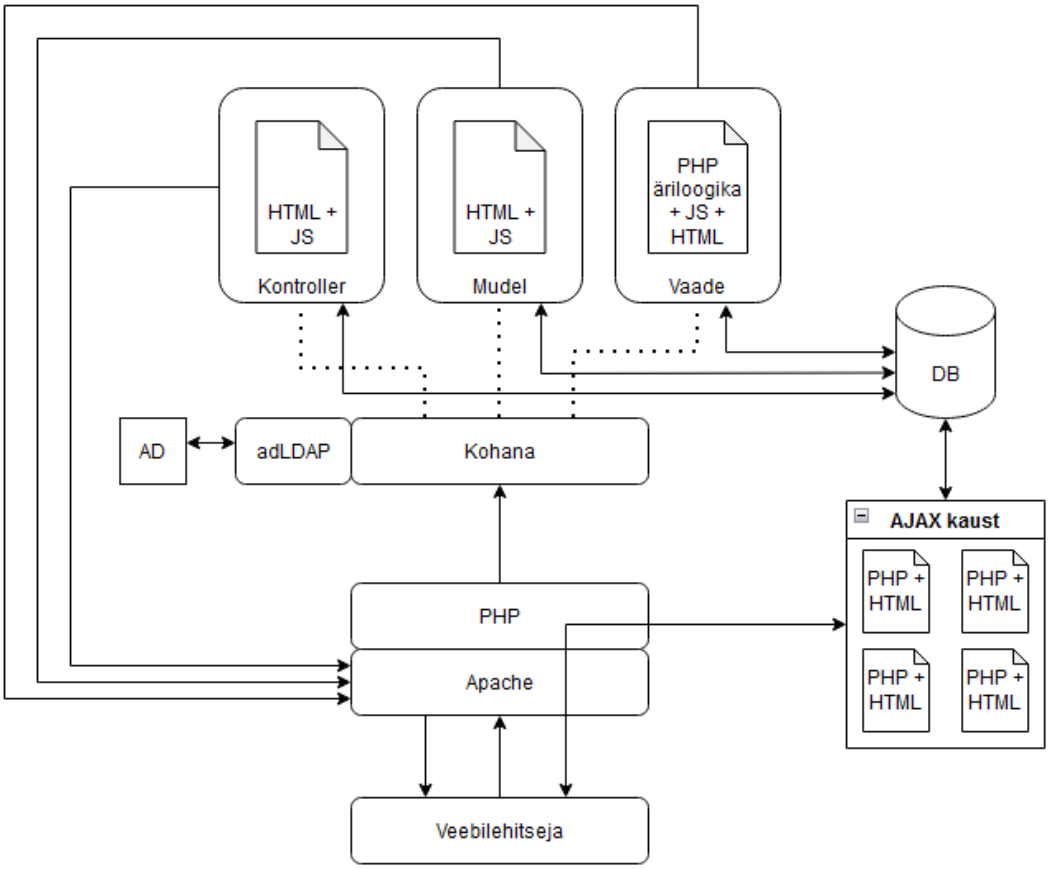
kaudu. Tegemist on suhtelise viitamisega ja seetõttu faile ja kaustasid hõlpsasti organiseerida ei anna. Sellise monoliitse koodibaasi puhul on arendajate või meeskondade vaheline koostöö pärsitud kuna muudatused mõjutavad tihti kõiki korraga ja ühtses repositooriumis muudatuste liitmine võib muutuda pidevaks ning tülikaks tegevuseks. Rakenduse keerukuse kasv on juba praeguseks tekitanud pudelikaelu monoliitse koodibaasi hallatavuses.

Suurimaid vaadete arendamise ja refaktoreerimisega seotud muresid on segunenud äriloogika ja esitluskihi lähtekood. Paljude vaadete kuvamiseks või toimingute tegemiseks võetakse ühe faili sisuks olev funktsioon. Funktsioonis tehakse päring andmebaasi, võetakse tulemus ja genereeritakse veebilehitsejale HTML kood kuhu süstitakse vahele päringutulemused. Sellist koodi hallata on tülikas kuna ei saa eraldi uuendada kuvamisega seotud osa. JavaScripti osa on eraldi failidesse kirjutatud ja kui päringu tagastatava HTML struktuuri osa peaks mingil põhjusel muutuma, ei saa olla kindel, et JavaScript jätkab toimimist puuduvate HTML elementide tõttu. Lisaks on sellest raske aru saada ja moodultestide puudumise tõttu võivad rakenduse eri osad uuenduste käigus kergesti katki minna. Tagarakenduses loodud staatilist HTMLi ja dünaamilisuse jaoks vajalikku JavaScripti eraldiseisvalt on väga keeruline arendada.

„Assets“ kaustas leidub palju JavaScripti teeke ja abistavaid faile. Paljud nendest on vanad. Osasid saab uuendada aga enamus on toe kaotanud ammu. Palju sarnast funktsionaalsust on käesoleva töö kirjutamise ajaks veebilehitsejate API kaudu lahendatud või uuemates raamistikes soliidselt lahendatud ja mugavdatud. Kasulikum oleks vanadest teekidest võimalusel lahti öelda kuna need on turvariskid ja tihti omavad aegunud funktsionaalsust, mis tekitab probleeme rakenduse töös. Kahjuks ei ole võimalik neid eemaldada teadmata, kas ja kus on sellel mõju rakenduse tööle.

Jätkates rakenduse ülesehituse ja käitumise uurimist leiab autor, et vaadete genereerimiseks kasutatakse küll kontrollerite klasse mis kontrollivad kas kasutaja on sisse logitud ja kui ei ole, suunatakse ta autentimise lehele aga puudub pidev kasutaja autoriseerimine toimingute jaoks. Osade meetodite alguses tehakse kontroll aga mitte alati. Kontrollide puudumine piirab rollide põhist rakenduse kasutamist ja annab võimaluse ilma õigusteta inimesele tahtlikult või kogemata teha autoriseerimata tegevusi.

Erinevate põhjuste tõttu nagu ajalised piirangud, kasvav keerukus, raamistiku ja arhitektuuri etteseatud tavade ignoreerimine, jms – on koodibaas muutunud sihtmeetmeliste lahenduste kogumiks, kus puudub kindel süsteemne lähenemine ja raamistiku poolt pakutavate võimaluste kasutus. Palju püsikodeeringut, kus puudub selge jaotus andmete visualiseerimise ja funktsionaalsuse vahel (Joonis 3). Raamistiku pakutavate turvalisusega seotud mugavusmeetodite ja kihtide kasutama jätmise tõttu on suur oht ka tänu turvariskidele.



Joonis 3. Rakenduse arhitektuuriline käitumine.

Kuna rakenduses on aktiivses kasutuses ka teisi tööriistu ja vaateid erinevate osakondade jaoks, leiab autor, et kõige mõistlikum on rakendust uuendada järkjärgult, uuendades esmalt rakenduse arhitektuuri ja etapphaaval refaktoreerida koodibaasi. Olemasolevat koodi on palju ja selle lõplik ümberkirjutamine on ajaliselt mahukas ning aja jooksul võib ilmneda probleeme või eripärasid millega olemasoleval hetkel arvestada ei osata.

3 Lahenduse prototüübi planeerimine

Komponenditehase arhitektuur on puudulik, ei võimalda edasiarendust ega süsteemi ja kasutatud vahendite uuendamist. Arvestades olemasoleva süsteemi analüüsi käigus väljatoodud puudusi, tehakse võimaliku lahenduse analüüsi käigus otsused olukorra parandamiseks. Otsuste tegemisel lähtutakse skaleeruva lahenduse eesmärgist ja kaasaegsetest metoodikatest.

3.1 Lähtetingimused

Autor täidab ettevõttes eesrakenduse arendaja rolli. Sellest tulenevalt teeb autor soovitusi ja otsuseid võttes arvesse meeskonnakaaslaste arvamusi ning kogemusi.

Tarkvarasüsteemis on kasutusel MySQL andmebaas, mille vahetuseks vajadus puudub.

Kuna tegemist on veebirakendusega, tuleb kindlasti arvestada veebiarenduse populaarseima keelega – JavaScript [7]. Ettevõttes on teiste erinevate rakenduste jaoks kasutusel JavaScripti teek Webix, mis hõlbustab vajalike elementide loomist nagu vormid, tabelid, nupud, graafikud ja palju muud. Teeki soovitakse edaspidiselt kasutada kuna ühtse välimuse saavutamine on tähtis hea kasutajakogemuse saavutamiseks [8].

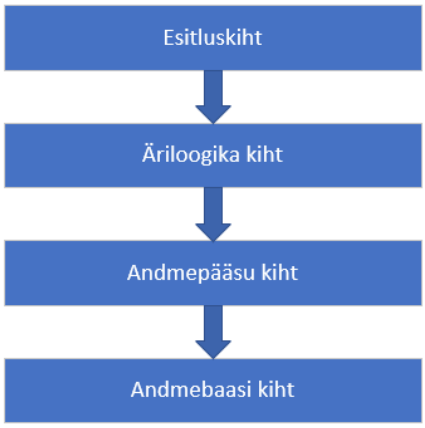
3.2 Struktureerimine

Raamatus *Software Architecture in Practice* kirjutatakse järgmist: „Süsteemi tarkvaraarhitektuur on struktuuride kogum, mis on vajalik süsteemi kirjeldamiseks, mis sisaldab tarkvaraelemente, nendevahelisi seoseid ja mõlema omadusi“ [9].

Kuna antud töö käigus lahendatakse konkreetseid probleeme, võetakse lühidalt eesmärgid kokku mida arhitektuuris muuta on vaja:

- eraldada äriloogika ja esitlusloogika;
- võimaldada erinevate struktuuride või elementide vahetamist, vajadusel uuendamist;
- parandada kasutajaliideste käitumist, modulaarsust ja veakindlust.

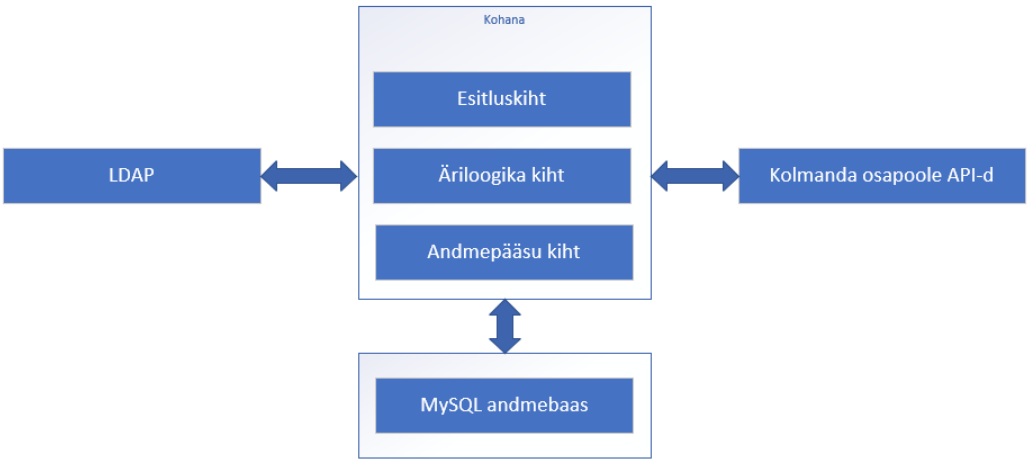
Üks levinumaid klient-server veebirakenduse arhitektuurimustreid on kihiline loogika jaotus. Kihte võib olla rakenduses vastavalt vajadusele - tüüpilise veebirakenduse puhul kasutatakse kolme- kuni neljakihilist lähenemist, kus igal kihil on oma konkreetne roll (Joonis 4).



Joonis 4. Kihilise arhitektuuri kihid.

Kuna monoliitses kihilises süsteemis on sõltuvused tihedalt seotud, on rakendus raskesti ning ainult horisontaalselt skaleeruv [10].

Vaatamata sellele on võimalik koodibaas korrastada ja toimingud kihiti korrektselt jagada. Joonis 5 demonstreerib kuidas olemasolev süsteem võiks välja näha töös käsitletud probleemideta.



Joonis 5. Ideaalseisus rakendus.

Monoliitse kihilise lähenemise juures on paar puudust. Esiteks ei eraldu eesrakenduse osa piisavalt. Erinevad projektid (eesrakendus ja tagarakendus) võiksid olla ka keeleliselt eraldatud. See ei ole ilmtingimata vajalik aga võimaldaks arendajatel selgelt jagada vastutusala ja luua eraldatud repositooriumid. Teine puudus on erinevate komponentide ja raamistike vahetamine. Kui kõik failid on ühe projekti alla koondatud, on hiljem koodibaasi raskem uuendada. Kuna MVC raamistikud kasutavad malli mootorit vaadete loomiseks, peab kõikides vaadetesse lisatud skripti faile haldama ja uuendatud rakenduses siduma.

Kuna eesmärgiks on lahutada rakendus osadeks parema arenduse ja hallatavuse saavutamiseks, sobib antud juhul N-astmeline arhitektuur, mis võimaldab jagada eesrakenduse ja tagarakenduse eraldi projektideks. Eelised mida sellega saavutada:

- skaleeruvus – saab suurendada esi- ja tagarakenduse ressursse eraldiseisvalt;
- kergem rakenduse uuendamine – rakendused uuendatakse sõltumatult üksteisest;
- eraldatud vastutusalad – koodi on hõlpsam siluda. Probleemide ilmnemisel on põhjuse leidmine ja korrigeerimine lihtsustatud.

Eelnimetatud põhjused pole ainukesed, neid leidub veel [11]. Lisaks on võimalik kasutada eesrakenduste raamistikke või teeke, mis võimaldavad eesrakendust luua SPA (*Single-page application*) ehk üheleherakendusena, mis vähendab tunduvalt päringute hulka ja parandab kasutajamugavust muutes rakenduse töö sujuvamaks [12].

Alternatiiviks on mikroteenuste arhitektuur, kus jagatakse rakenduse osad funktsionaalsuse alusel ning tehakse üksteisest sõltumatuks. Selline lähenemine lihtsustab hallatavust ja võimaldab tihedamat evitamise tsüklit kuna arendatavad tükid on väiksemad ja isoleeritud üksteisest. Funktsionaalsuste määramine peab olema konkreetne ja hästi läbi mõeldud. Suure monoliitse rakenduse tükkideks jagamine võib tekitada palju lisakeerukust. Lisaks, mikroteenuste arhitektuuris kasutavad erinevad teenused tihti eraldi andmebaase, mis nõuaks täiendavaid mudeli ja andmebaaside muudatusi [13]. Ettevõtte protsessid on tihedalt seotud, et selline lähenemine optimaalne oleks. Vaated ja protsessid on vaja lahutada üksteises aga piisab nende moodulitesse paigutamisest ehk osakonna või protsessi järgi grupeerimisest.

3.2.1 Esitluskiht

Esitluskihi eesmärk on teha kasutajatele võimalikult lihtsaks ja mugavaks erinevate andmetega töötamine. Kuna olemasolev rakendus koostab HTML-i vaadete jaoks tagarakenduses kasutades malli mootorit ja vormistades staatilisi vaateid, puudub võimalus efektiivselt kontrolle seadistada. Eesrakenduse elementide loomise hõlbustamiseks saab kasutusele võtta teegi.

Arvestama peab erinevate kontrollidega:

- kas kasutaja on sisse logitud;
- kas kasutajal on õigus erinevaid toiminguid teha;
- kas sisend on õige või lubatud.

Viimase kontrolli saab teha kahes etapis. Kui operaator skaneerib plekipakki töösse, peab süsteem jätkamiseks veenduma, et antud plekk sobib komponentide tootmiseks. Väljade eelvalideerimiseks saab kasutada Regex-it. Regex ehk regulaaravaldis on informaatikas sümbolitest koosnev otsingumuster. Kontrollida tuleb enne salvestamist, kas vajalikud väljad on täidetud või vaba vormi sisendid sisaldavad sobilikku teksti. Sisendi lõpliku õigsust peab kontrollima API kaudu kuna eesrakenduses puudub vajalik informatsioon otsuse tegemiseks. Lisandunud toodete parameetrite järgi saab tagarakendus teha kontrolli toote vastavuses.

3.2.2 Äriloogika kiht

Kuna kolmeastmelises arhitektuuris on eraldatud eesrakendus, ärikiht ja andmebaas, võimaldab ärikiht valideerida ja autoriseerida andmevahetust. Selle saavutamiseks on võimalik äriloogika kiht kirjutada RESTful API-ks.

REST (*representational state transfer*) on tarkvaraarhitektuuri laad, mis võimaldab universaalselt salvestada või küsida ressursse pöördudes kindla otspunkti ehk URI poole. Otspunktide poole pöördumiseks kasutatakse HTTP andmeedastus protokolli, kus andmevorming ei ole määratud, mis annab süsteemi ehitamisel vabad käed kasutada XML, JSON või vaba teksti andmevahetuseks. HTTP päises on vajalikud kirjed metaandmete, volituste, URI, vahemälu kasutuse, küpsiste kasutuse ja olekukoodide kohta. [14]

REST nõrkusteks võib lugeda ka tema tugevusi. Näiteks olekuta suhtluses on vajalik eesrakenduses pidada oleku haldust ning andmed kuhugi salvestada, mis muudab programmeerijate jaoks keeruliseks serverivärskenduste juurutamise ilma kliendipoolset päringut tegemata. Selle probleemiga puutusime kokku 2.2 punktis. Õnneks tänapäeva eesrakenduse raamistikud ja teegid on rakendanud mugavad viisid andmete olekute haldamiseks.

3.3 Tagarakenduse lahendus

Kuna olemasolevasse rakendusse on ehitatud ka teisi vaateid ja tööriistu nagu mittevastavuste haldamine ja tootejälgimine, ei ole optimaalne lahendus lihtsalt uus tagarakendus luua. Koodi ümberehitus nõuab palju aega ja ka teiste vaadete otstarve ja mudel võivad olla aja jooksul muutunud, mis omakorda pikendab rakenduse ümberehitust. Antud töö raames neid parandusi ei ole plaanitud teha, vaid arvestatakse tulevase vajadusega.

Olemasolev rakendus on ehitatud MVC arhitektuuril põhineva Kohana 2.0 raamistikuga, mis tuli kasutusse aastal 2007 ja kasutab PHP 5 leksikat. Kuna antud töö kirjutamise ajal on saadaval juba PHP 8.1 versioon, on mõistagi seda kõvasti arendatud ning tulenevalt parandatud palju turvalisusega seotud eripärasid mille kohta F-Secure Radar hoiatas punktis 2.3.

Kasutusel olevat raamistikku on sellest ajast edasi arendatud ning 2009 aastal tuli välja Kohana 3.0, millel on PHP 7 tugi [15]. Kohana projekt lõpetati algsete arendajate meeskonna poolt ära aga kuna migratsioon teiste raamistike peale oli üpris keeruline, otsustasid osad arendajad avatud lähtekoodi projektina jätkata ja edasi arendada. Kohanast kasvas välja järgmine haru mida arendatakse tänaseni, nimega Koseven [16].

Võttes arvesse, et koodi mida migreerida ja refaktoreerida on palju, loeb autor kõige kasulikumaks kirjutada tagarakenduse osa RESTful API-ks samas raamistikus, uuendada raamistiku versiooni ning võimalik, et hilisemalt üle minna Koseven harule või lausa raamistikku vahetada. Selline iteratiivne arendamine võimaldab varajasemas faasis avastada puudusi või ebakõlasid, mille põhjal teha paremaid otsuseid tagarakenduse migreerimisel. Kuna sobilik on lahutada ja korrastada algkoodi ning sellest saab API esimene iteratiivne versioon, ei loeta mõistlikuks rakendada teenuse kihti. Autentimine ja

autoriseerimine toimub kontrolleri tasemel ja kontroller pöördub mudeli poole andmete pärimiseks.

Kuna RESTful API puhul on äriloogika kiht eraldatud, on võimalik see vajadusel hiljem mõnda teise raamistikku ümber migreerida või lausa teises programmeerimisekeeles kirjutada. Populaarseimad nendest on Java koos Spring raamistikuga, C# koos ASP.NET raamistikuga ja Node.js koos Express raamistikuga [7].

3.4 Eesrakenduse tehnoloogia valik

Eesrakenduse raamistiku valikul tuleks lähtuda jätkusuutlike ja kergelt integreeritavate lahendustega, mis täidaks rakendusele esitatavad nõuded. Kuna antud töö puhul on ettevõttes kasutusel Webix JavaScripti teek, mis hõlbustab vaadetes elementide loomise ja nendele funktsionaalsuse lisamise ning seda soovitakse kindlasti edaspidi kasutada, muutub see peamiseks piiranguks. Üldiselt soovitatakse vaadata populaarsemaid teeke ja raamistikke kuna nende kasutajaskond on suur, mis suurendab tõenäosust, et lahendatavatele probleemidele leiab kergemini lahenduse.

Webix dokumentatsioonis leidub nimekiri tehnoloogiatest ja integratsiooni näidetest [17], mille hulka kuuluvad tuntud teegid ja raamistikud nagu Angular, Backbone, Jquery, React, Vue. Lisaks soovitatakse proovida nende endi mikroraamistikku Webix Jet.

Kuna autoril puuduvad süvakogemused antud tehnoloogiatega, peab tegema võrdluse teiste kasutajate arvamuste põhjal. Selleks luuakse tabel ja võrreldakse neid vajalike kriteeriumite põhjal (Tabel 1).

Antud juhul huvitavad tehnoloogiad, mis võimaldavad luua SPA eesrakenduse ja võimaldaksid mõlemasuunalise andmepäringute rakendamise. Kuna Websockets ja SSE on veebilehitseja põhised API-d, ei pea nende pärast muretsema. Populaarsus koosneb mitme allika keskmisest [7], [18], [19].

Tabel 1. Eesrakenduse tööriistade võrdlustabel.

Nimi	Tüüp	Populaarsus	Õppimise keerukus	SPA
Angular	Raamistik	keskmine	suur	jah
React	Teek	suur	suur	jah
Vue	Teek	keskmine	keskmine	jah
Backbone	MVC raamistik	väike	keskmine	jah
Webix Jet	Mikroraamistik	puudub	väike*	jah

* autori hinnang

Antud tabeli järgi jääb mulje, et sobivad Angular, React ja Vue. Parema ülevaate saamiseks loob autor iga tehnoloogiaga seotud prooviprojekti ja katsetab toimimist.

Selgub, et kuna React ja Vue on rohkem pühendunud kasutajaliideste loomisele ja oleku haldamisele ning sama eesmärk on ka Webix teegil, tekivad sisemised komplikatsioonid erinevate elementidega. Siin võib mängida rolli see, et dokumentatsioonis viidatud repositooriumid on uuendatud umbes viis aastat tagasi. Sel ajal olid aktuaalsed Vue2, kuigi praeguseks on kasutusel Vue3. Reacti kirjutati klassi põhiste komponentidega ja viimastes versioonides on üle mindud funktsionaalsetele komponentidele [20]. Kuna eelnimetatud tehnoloogiate õpe algaks suhteliselt algtasemest, jätab autor need hetkel kõrvale.

Kui võrrelda Angulari ja Webix Jet raamistikke, on tunduvalt lihtsam teha otsus, mida eelistada. Angular on täiemahuline raamistik, mis seab kindlad piirid ja suunised projekti ning komponentide loomiseks. Lisaks on Angular arvamuslik (opinionated) raamistik, kus on rangelt soovituslik järgida kindlat stiili rakenduse loomisel. Dokumentatsiooni järgi on Angulari ja Webixi integratsioonil limitatsioonid, sest Webixi elemendid ei ühildu ngIf ja muude DOM-i mutatsioonidirektiividega. ngIf direktiivi kasutatakse Angularis kõige sagedamini tekstisisese malli tingimuslikuks kuvamiseks.

Testimisjärgse hinnangu järgi sobib tegelikult kõige paremini eesmärkide täitmiseks Webixi enda mikroraamistik. Webix Jet võimaldab luua SPA rakenduse, kus on lahendatud marsruutimine (*routing*) ning on suur eelis, et ka süntaks ja objektide käitumine on teegi omale väga sarnane, kiirendades arendust kuna autor ja ka autori meeskonnakaaslased juhinduvad selles palju paremini.

4 Lahenduse teostamine

Antud peatükis teostatakse eelnevate otsuste ja valikute põhjal rakenduse prototüübi loomine plaanitavale arhitektuurile.

4.1 Ümberstruktureerimisega seotud probleemid

Kihtide eraldamiseks oli vajalik esmalt lahutada äriloogika ja esitlusloogika kood. Selleks oli mõistlik produktsiooni kood kopeerida arenduskeskkonda.

Antud juhul on korraga tegu nii refaktoreerimisega kui ka koodi ümberkirjutamisega. Paljud allikad soovitavad enne koodi muudatusi teha eelnevalt funktsionaalsustele testid [21], [22], [23]. Kahjuks ei ole võimalik adekvaatseid teste kirjutada, sest nagu eelnevalt sai välja toodud, paljud funktsioonid teevad mitmeid asju korraga ja tagastatav tulemus ei ole see mida tegelikult soovitakse. Testid kirjutatakse tavaliselt funktsiooni väljundile aga kuna antud juhul on vajalik andmete objekt ja mitte HTML, ei saa sellises olukorras teste vanale koodibaasile luua.

4.1.1 Tagarakenduse refaktoreerimine

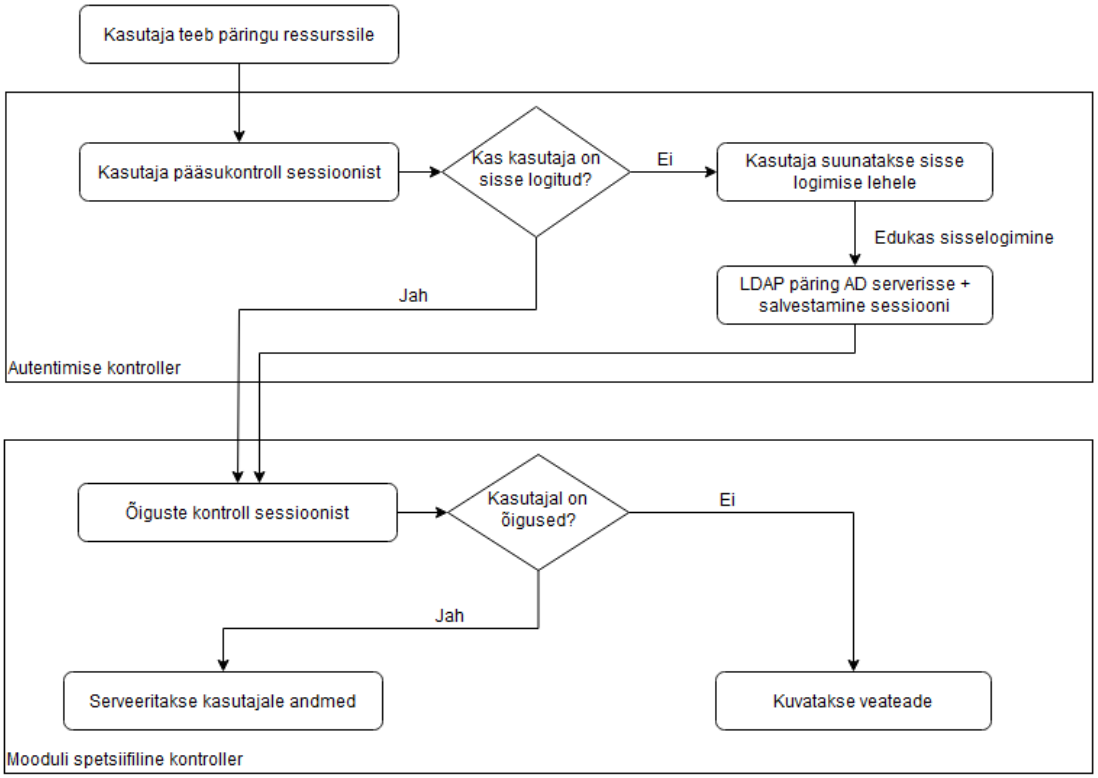
Tagarakenduse muutmisel API-ks oli vajalik rakendada järgmised sammud:

- määrata lõpp-punktid mida rakendus hakkab kasutama;
- määrata andmemudel ja lõpp-punktide tagastatavad andmed;
- refaktoorida meetodid korrektselt nimetatud kontrolleritesse.

Arvestada tuli ka turvalisusega, mis olemasolevas rakenduses on puudulik. Selleks tuleb luua laiendatud kontroller kus kontrollitakse, kas kasutaja on sisse logitud või peab teda suunama sisse logimise lehele. Kontroller vastutab sessiooni haldamise eest, kuhu kasutaja informatsioon on salvestatud (Joonis 6).

Joonis 6. Laiendatud kontrollerid.

Kasutaja suunatakse algselt sisselogimise lehel, kus end kasutajanime ja parooliga tõendab. Õnnestunud sisenemisel salvestataks kasutajaga seotud informatsioon sessiooni muutujasse. Rakendus küsib iga päringu puhul esmalt sessioonist, kas kasutaja on ikka veel sisse logitud. Autentimiseks mõeldud kontrolleri klassi laiendatakse edasi mooduli spetsiifiliseks. Klass võimaldab küsida sessioonilt kasutaja õigustega seotud andmeid ja nende põhjal edasiseid toiminguid teha või keelata (Joonis 7).



Joonis 7. Päringu õiguste kontrolli skeem.

Alternatiivselt tuli luua ka autoriseerimata laiendatud kontrolleri klass nendeks juhtudeks kui andmeid päritakse seadetest kus kasutaja ei pea olema sisse logitud. Näiteks tootmises olevad kioskid, mille funktsionaalsus on limiteeritud ja turvaoht madalam ning kasutajate autoriseerimine puudub.

4.1.2 Eesrakenduse loomine

Analüüsi käigus tehti otsus, et kasutusele võetakse mikroraamistik Webix Jet. Integreerimiseks tuli raamistik alla laadida ja lahti pakkida projektikausta. Tegemist on lähtekoodiga, mida saab käivitada Node.js abil. Node.js on avatud lähtekoodiga mitmeplatvormne JavaScripti käitussüsteem, mille abil saab arendada mitmesuguseid rakendusi [24]. Programm kasutab JavaScripti interpreteerimiseks Google V8 JavaScripti mootorit.

Algses koodistruktuuris on olemas kaust „sources“, mille sees on veel neli kausta „locales“, „models“, „styles“, ja „views“ ning myapp.js fail, mille autor nimetas ümber lihtsalt app.js-iks. Node_modules kausta tekitab Node kõik sõltuvused mida on vaja, et rakendus toimiks (Lisa 2).

Autor lisas „config“ kausta kuhu lisatakse konfiguratsiooni failid, mis sisaldavad rakenduse spetsiifikat nagu näiteks lokatsioon, mille järgi saab määrata lokalisatsiooni ja valida milliseid vaateid rakendus kuvab. Komponente toodetakse ainult Eestis, mis kunagi võib muutuda. Teiste osakondade vaadete migreerimisel on kindlasti vajalik määrata asukoht, kuna erinevate maade tegevused ja vaated erinevad. „Locales“ hoiustab lokalisatsiooni faile ja sinna lisanduvad tõlkefailid, mis on tavalised JavaScripti failid ja sisaldavad objekti võti-väärtus paaridega tõlgetest. „Images“ alla paigutatakse kõik vajalikud pildid ja ikoonid.

Kaustas „views“ asuvad vaated ehk rakenduse komponendid. Autor leiab, et põhifunktsionaalsus, mis vaadetes ei erine, võiks asuda „base“ kaustas ja ülejäänud failid struktureeritud vastavalt osakondadele, kus neid vaateid kasutatakse. Erinevates raamistikes võib sellise kausta nimi ka „modules“ või „components“ olla. Antud juhul Webix Jet otsib „views“ kaustast klasse mille nimed ühtivad otsitavate URL otspunktides olevate sõneosadega.

Node.js ja projektifailidega kaasa tulnud Webpack moodulite komplekteerijaga saab rakenduse pakendada üheks JavaScripti failiks ja üheks CSS failiks, mis siis serverisse paigutatakse.

4.1.3 CORS ja Apache seadistus

Rakenduse evitamisel samasse serverisse aga erineva pordi peale tekitas probleeme kuna päringud ei olnud edukad. Põhjuseks CORS (*Cross-Origin Resource Sharing*) ehk allikatevaheline ressursside ühiskasutus mehhanism, mis võimaldab veebilehel jagada ressursse teiste domeenidega. Vaikimisi on tavaliselt turvalisuse eesmärkidel keelatud. Serveris on kasutusel Apache veebiserver, mille httpd.conf sättete failis tuleb VirtualHost direktiivi lisada „Header set Access-Control-Allow-Origin“ ning väärtustada domeeni URL-i ja pordiga, mis lubab erineva pordi kaudu toimival eesrakendusel päringuid teha tagarakendusele.

Parameetri võib ka projektikausta .htaccess faili lisada aga osadel juhtudel mõjutab see serveri töökiirust. „RewriteRule“ direktiiv kasutab Regex-it, mis reegli konfiguratsiooni faili lisamisel kompileeritakse server käivitamisel ja muudel juhtudel iga päringu puhul [25].

Kuna tegemist on arenduskeskkonnaga, on antud juhul eesrakendusele määratud port 6969. Antud reeglite järgi otsitakse URL ressursse. Kui ressursid puuduvad, suunatakse päring index.html peale, kus rakenduse ruuter käsitleb URL-ga seotud süntaksianalüüsi (Joonis 8).

```
Listen 6969
<VirtualHost *:6969>
  DirectoryIndex index.html
  DocumentRoot "/var/www/jetProject"
  <Location "/">
        RewriteEngine On
        # If an existing asset or directory is requested go to it as it is
        RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -f [OR]
        RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -d
        RewriteRule ^ - [L]
        # If the requested resource doesn't exist, use index.html
        RewriteRule ^ /index.html
  </Location>
</VirtualHost>
```

Joonis 8. Apache konfiguratsioon eesrakendusele.

4.2 Ruuteri seadistamine

Webix Jet pakub kasutamiseks neli erinevat olemasolevat erineva funktsionaalsusega ruuterit:

- StoreRouter – ei kuva URL aga salvestab selle sessiooni.
- EmptyRouter – ei kuva URL ja ei salvesta seisu.
- HashRouter – domeeni ja ülejäänud URL vahele tekib # märk, mis aitab vältida lehe taaslaadimist ja võimaldab märgist alates süntaksianalüüsi.
- UrlRouter – üks traditsioonilisemaid viise veebis navigeerimiseks. URL kuvatakse aadressiribale.

Kuna paljud kasutajad on salvestanud endale enimkasutatavad URL-id veebilehitseja kiirpääsuribale ja harjunud nendega tihti toimetama või erinevatele vaadetele või töödele viitama, on üpris tähtis, et selline käitumine ja funktsionaalsus hoitakse alles. Tänu sellele kaks esimest ruuterit antud projekti ei sobi. Ka HashRouter ei sobi tänu oma lisandunud # märgile ja jääb üle vaid UrlRouter. Ruuteri korrektseks käitumiseks on vaja suunata päringud alati index.html peale. Seda saab teha Apache konfiguratsioonis (Joonis 9).

Arenduse käigus selgus, et kuna Webix Jet kuvab vaateid liitstruktuuris (*Nested structure*) on alati esimene vaade „top“ vajalik ja see esineb ka URL-is. „Top“ sees saab kuvada näiteks staatilist päist, menüüd ja jalust. Ja menüü ja jaluse vahele asetatakse kohatäite element. Kui kasutaja valib menüüst või sisestab aadressiribale soovitud vaate URL-i, asendatakse see kohatäite element vajaliku vaate elemendiga ülejäänud lehte uuendamata.

See tekitas probleemi kohustusliku „/top“ lisandumise tõttu mida eelnevas rakenduses polnud. Probleemi vältimiseks tuli kirjutada kohandatud ruuter, kus „get“ ehk aadressiribalt väärtust küsivas meetodis eemaldatakse sõne „/top“ ja „set“ ehk aadressiribale ja veebilehitseja ajalukku väärtust lisav meetod paneb „/top“ väärtuse vajadusel tagasi.

4.3 Vaadete sünkroonne uuendamine

Õnneks on Websocket ja SSE rakendamine lihtne. Klientrakenduse poolel tehakse uus WebSocket ühendus kuhu sisestatakse protokoll ja URL millega ühenduda ning tagarakendusest sõnumi saamisel uuendatakse antud näites tabelit uute andmetega (Joonis 9). Tagarakenduse jaoks on abiks PHP teek nimega Ratchet.

```
const socket = new WebSocket('ws:host');

socket.onmessage = (e) => {
   $$('productionPlan').clearAll();
   $$('productionPlan').load(JSON.parse(e.data));
}
```

Joonis 9. Websockets rakendamine eesrakenduses.

Andmete uuenemisel saadab tagarakendus ühendatud klientidele sõnumid teavitades andmete muudatustest, mille põhjal saavad klientrakendused kuva uuendada.

4.4 Lõpptulemus

Uuendatud arhitektuuris (Lisa 3) on eraldatud kohustused ja vastutused. Projekt on jagatud kihilisteks osadeks, mille uuendamine ja vahetamine on tunduvalt lihtsam. Klient suhtleb eesrakendusega, mis omakorda küsib vajalikke andmeid API käest. Eesrakenduse vaated on jagatud moodulite kaupa ja tagarakenduse äriloogika on jagatud mooduliteks ettevõtte protsesside järgi tulevaseks hõlpsamaks integratsiooniks. Eesrakenduse ruuter haldab sissetulevaid päringuid ja vastutab vaadete kuvamise eest. Vaadetes vastavalt vajadusele luuakse Websocket ühendus tagarakendusega, mis võimaldab sünkroonseid uuendusi andmete uuenemisel. Suur osa uuendusi siiski tehakse tegevuste tagajärjel, näiteks nupu vajutuse või avanenud modulaarse akna kinnipanekul. Vaadete moodulid on struktureeritud kooskõlas tagarakenduse moodulitega vastutuste ja õiguste paremaks haldamiseks. Igal tagarakenduse moodulil on spetsiifilised kontrollerid ja mudelid aga vaade mis vahendab tagastatavaid andmeid on üks. Lisaks on Kohana spetsiifilised moodulid mis võimaldavad autentimist ja mugavat suhtlust andmebaasiga. Eesrakenduse mooduleid on võimalik taaskasutada, sisestades neid testesse vaadetesse kui komponente.

5 Kokkuvõte

Käesoleva töö käigus kirjeldati kuidas rakendus toetab tootmisprotsesse ja mida peaks kasutajatele võimaldama. Analüüsiti probleeme miks rakendust on raske uuendada või muuta ning milliseid probleeme see endaga kaasa toob. Selgus, et pärandkood oli tihedalt seotud ja kuvas tihti staatilisi lehti, mille kuvamiseks mõeldud kood loodi äriloogika arvutuste käigus. Leidus palju teeke, mis olid vanad ja raske oli määrata nende kasutust rakenduses.

Arendusuuringu käigus analüüsiti võimalusi probleemide lahendamiseks. Kõige sobilikumaks peeti rakenduse kihtide korrastamist ja astmeliseks tegemist. Selgus, et kuna rakenduses on teisi tähtsaid tööriistu mis leiavad kasutust, oleks kõige sobilikum minna üle uuele arhitektuurile etapp haaval. Ümberkirjutamise käigus jagati rakendus kihtideks ja eraldati äriloogika ja eesrakenduse kood, mis võimaldab selgemat arendust ja kergemaid muudatusi.

Teostamise käigus valmis prototüüp, mis lahendab algsed kirjeldatud probleeme ja võimaldab vaadete loomist loogiliste struktuuride alusel moodulite kaupa. Lisaks võimaldab eesrakenduse ja tagarakenduse eraldiseisvaid uuendusi.

Kasutatud kirjandus

[1] „Wayback Machine“, 10. september 2016. https://web.archive.org/web/20160910002130/http://worrydream.com/refs/Brooks-NoSilverBullet.pdf (vaadatud 13. oktoober 2022).

[2] B. Ganesan, „Polling vs SSE vs WebSocket— How to choose the right one“, *Medium*, 19. september 2020. https://codeburst.io/polling-vs-sse-vs-websocket-how-to-choose-the-right-one-1859e4e13bd9 (vaadatud 25. oktoober 2022).

[3] „System Design: Long polling, WebSockets, Server-Sent Events (SSE)“, *DEV Community*. https://dev.to/karanpratapsingh/system-design-long-polling-websockets-server-sent-events-sse-1hip (vaadatud 24. oktoober 2022).

[4] „Using server-sent events - Web APIs | MDN“. https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events (vaadatud 7. november 2022).

[5] „Radar user guide“. Vaadatud: 26. oktoober 2022. [Võrgumaterjal]. Saadaval: https://help.f-secure.com/data/pdf/radar-3.0-userguide-eng.pdf

[6] „ASP.NET MVC“. Vaadatud: 11. november 2022. [Võrgumaterjal]. Saadaval: https://digiarhiiv.ut.ee/Ained/Doc/VFailid/Ptk5-7.pdf

[7] „Stack Overflow Developer Survey 2022“, *Stack Overflow*. https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022 (vaadatud 1. november 2022).

[8] A. AlTaboli ja M. R. Abou-Zeid, „Effect of Physical Consistency of Web Interface Design on Users' Performance and Satisfaction“, *Human-Computer Interaction. HCI Applications and Services*, Berlin, Heidelberg, 2007 Saadaval: https://www.researchgate.net/profile/Ahamed-Altaboli/publication/221098252_Effect_of_Physical_Consistency_of_Web_Interface_Design_on_Users%27_Performance_and_Satisfaction/links/5f9ac6da92851c14bcf0c4c7/Effect-of-Physical-Consistency-of-Web-Interface-Design-on-Users-Performance-and-Satisfaction.pdf (vaadatud 3. november 2022)

[9] L. Bass, P. Clements, ja R. Kazman, *Software Architecture in Practice*

[10] „What is monolithic architecture in software?“, *WhatIs.com*. https://www.techtarget.com/whatis/definition/monolithic-architecture (vaadatud 3. november 2022).

[11] M. Martin, „N Tier(Multi-Tier), 3-Tier, 2-Tier Architecture with EXAMPLE“, 1. märts 2020. https://www.guru99.com/n-tier-architecture-system-concepts-tips.html (vaadatud 7. november 2022).

[12] „The Pros & Cons of Single Page Applications (SPAs)“, *iTechArt*. https://www.itechart.com/blog/pros-cons-of-single-page-applications/ (vaadatud 8. november 2022).
[13] „To Micro or Mono – Pros and Cons of Both Service Architectures - DZone Microservices“, *dzone.com*. https://dzone.com/articles/to-micro-or-mono-pros-and-cons-of-both-service-arc (vaadatud 3. november 2022).
[14] „What is a REST API?“ https://www.redhat.com/en/topics/api/what-is-a-rest-api (vaadatud 7. november 2022).
[15] „Kohana: The Swift PHP Framework“. https://kohanaframework.org/ (vaadatud 2. november 2022).
[16] „Koseven/koseven“. Koseven, 1. oktoober 2022. Vaadatud: 2. november 2022. [Võrgumaterjal]. Saadaval: https://github.com/koseven/koseven
[17] „Integration with Other Frameworks, Guides Webix Docs“. https://docs.webix.com/desktop__third_party_integration.html (vaadatud 8. november 2022).
[18] „The Best JS Frameworks for Front End“. https://rubygarage.org/blog/best-javascript-frameworks-for-front-end (vaadatud 8. november 2022).
[19] „Front-end frameworks popularity (React, Vue, Angular and Svelte)“ https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190 (vaadatud 8. november 2022).
[20] T. Stepro, „React: Migrating from Class to Functional Components (with hooks)“, *Medium*, 29. juuni 2020. https://timothystepro.medium.com/react-migrating-from-class-to-functional-components-with-hooks-1bde8916ca1f (vaadatud 8. november 2022).
[21] „Code Refactoring Tips & Best Practices | Built In“. https://builtin.com/software-engineering-perspectives/code-refactoring (vaadatud 13. november 2022).
[22] „7 Code Refactoring Techniques in Software Engineering“, *GeeksforGeeks*, 22. september 2020. https://www.geeksforgeeks.org/7-code-refactoring-techniques-in-software-engineering/ (vaadatud 13. november 2022).
[23] „PHP code refactoring – practical tips with code examples“, 20. juuli 2022. https://tsh.io/blog/php-code-refactoring/ (vaadatud 13. november 2022).
[24] Node.js, „Node.js“, *Node.js*. https://nodejs.org/en/ (vaadatud 13. november 2022).
[25] „Does .htaccess slow site down?“, *Programmer and Software Interview Questions and Answers*. https://www.programmerinterview.com/apache-interview-questions/does-htaccess-slow-site-down/ (vaadatud 22. november 2022).

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

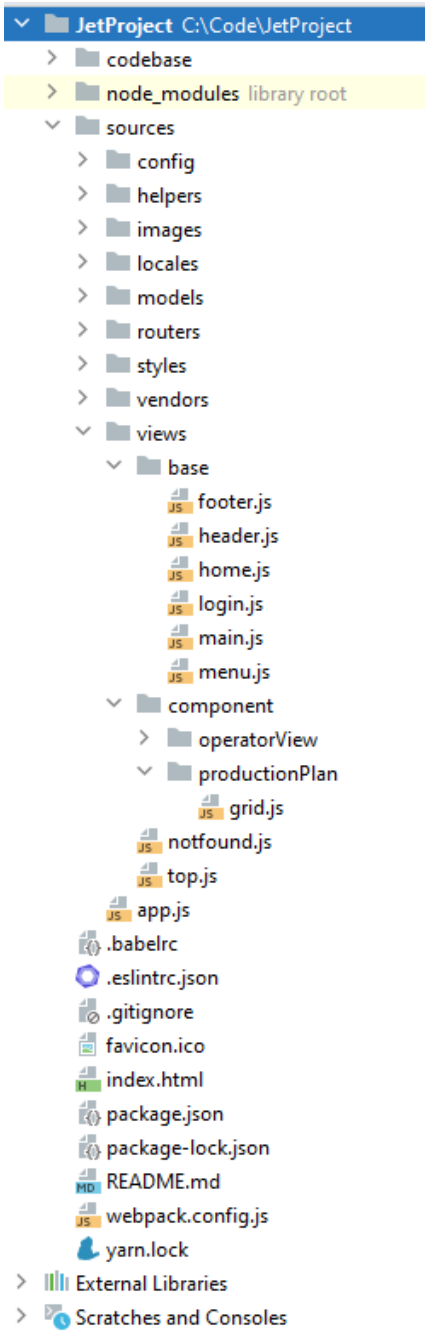
Mina, Valter-Kaspar Karolin

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Metaprint AS-i komponenditehase tootmisinfosüsteemi arhitektuuri uuendamine", mille juhendaja on Märt Kalmo
   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.

28.12.2022

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 – Eesrakenduse failistruktuur



Lisa 3 – Uus arhitektuur