TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kristjan Tärk IADB179517

# SARNASE VÄLIMUSE JA FUNKTSIONAALSUSEGA RAKENDUSTE ARENDUSE ÜHTSUSTAMINE CGI EESTI AS-I NÄITEL

Bakalaureusetöö

Juhendaja: Märt Kalmo
MSc

Tallinn 2020

# 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: Kristjan Tärk

18.05.2020

# Annotatsioon

Ettevõttel võib olla mitmeid sarnase välimuse ja ühise funktsionaalsusega rakendusi. Töö eesmärk on leida tarkvara arenduseks sobiv meetod, mis kiirendab rakenduste arendust, tagades arenduse käigus ühtset välimust ja funktsionaalsust.

Töös esitatakse kahe erineva võimaliku lahenduse analüüs. Lahendusteks on jagatud moodulite loomine ja rakenduste ühisosa sisaldava malli arendus. Analüüsis hinnatakse erinevaid lahenduste väljatöötamise ja kasutuselevõtuga seotud eeliseid ja puuduseid. Tuginedes võrdluse tulemustele valitakse probleemile kõige optimaalsem lahendus.

Töös kirjeldatakse probleemi lahendust malli meetodit kasutades. Lahendus arendatakse AS CGI Eesti vajadustelt lähtuvalt.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 40 leheküljel, 5 peatükki, 3 joonist, 5 tabelit.

# Abstract

Unifying the Development of Applications with Similar Appearance and Functionality on the Example of CGI Eesti AS

Companies can have many web applications that have similar appearance and functionality. The purpose of this thesis is to find a suitable method for developing such applications in more rapid fashion while ensuring consistency of the user interface and shared functionality.

To archive this goal thesis analyses two different methods. One possible solution is to package shared functionality to separately distributed modules, which could be imported to different projects. Another proposed solution is to develop a company specific project template that includes the shared functionality, which could be used for developing new applications.

Analysed methods are compared. Based on the comparison the most optimal solution is chosen. Thesis found template method to be more suitable for CGI's needs.

Thesis shows how to develop a project template on the example of CGI Estonia. The paper describes how to specify requirements for such template and how to develop the template using existing project as a starting point.

The implemented solution fulfils its requirements by easing the development of similar applications. The template is used for developing new applications for the company.

The thesis is in Estonian and contains 40 pages of text, 5 chapters, 3 figures, 5 tables.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| Angular | TypeScript keelel põhinev veebirakenduste loomise raamistik. |
| CSS | *Cascading Style Sheets* ehk kaskaadlaadistik on tehnoloogia märgistuskeelse dokumendi (näiteks HTML dokumendi) vormingu ja ilme kirjeldamiseks. |
| E-arve | Masinloetav arve, mida erinevad tarkvarasüsteemid töötlevad. Nende alusel saavad pangad koostada maksekorraldusi. |
| Git | Versioonihaldustarkvara. |
| Git *fork* | Git repositooriumi kloonimise viis, kus kloonitud repositooriumi sisu saab muuta sõltumatult originaalsest projektist. Originaalis tehtud muudatusi saab kergelt viia koopiasse, samuti saab kloonis tehtud muudatusi viia originaalsesse repositooriumisse. |
| HTML | *HyperText Markup Language* ehk hüpertekst-märgistuskeel on keel, milles märgendatakse veebilehti. HTML dokument koosneb elementidest, mille järgi veebibrauser dokumendi sisu kuvab. |
| JavaScript | Programmeerimiskeel, mida kasutatakse peamiselt interaktiivsete veebilehtede loomiseks. |
| Keycloak | Kasutaja tuvastuseks ja halduseks kasutatav toode. Toodet on võimalik kergesti liidestada erinevatel tehnoloogiatel põhinevate rakendustega. Võimaldab ühe kasutajaga erinevate teenuste kasutamist. |
| Management UI2 | Veebirakendus, kus saab muuta e-arveid töötleva platvormi seadeid. |
| Npm | Laialt levinud JavaScript pakkide haldamise tarkvara. Npm koosneb konsooli rakendusest, npm pakkidest ehk eriliselt pakendatud tarkvarast. Tarkvarapakkide jagamiseks kasutatakse npm registreid. Registritesse saab alla laadida pakke, samuti saab registrisse pakke üles laadida. |
| PIP | *Purchase invoice portal* on veebirakendus, kust väikeettevõtted saavad saata e-arveid. |
| Spring | Raamistik Java-põhiste rakenduste loomiseks. |
| TypeScript | Programmeerimiskeel, mis täiendab JavaScript keelt, lisades sellele staatilise tüübikontrolli. Keeles kirjutatud programmid teisendatakse enne nende käivitamist JavaScript keelde. |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# Sissejuhatus

Ettevõtetes on tihti mitmeid veebilehti, mis jagavad mingit osa funktsionaalsusest või välimusest. Ühtne välimus parandab kasutajakogemust ja vähendab kasutaja tehtud vigade arvu [1].

Arvatakse, et veebilehtede ühtne välimus ja käitumine aitab kasutajal, kes on teise sarnase lehega tutvunud, kergemini uut veebilehte kasutama õppida. Kasutajad suudavad oma eesmärke kiiremini täita, tehes vähem vigu, kuna nad tunnevad sarnast süsteemi. Vähendatud vigade tegemise arv ja kiirem ülesande täitmine tõstab kasutajarahulolu. [2]

Alustades uue toote arendusega, mille kasutajaliides sarnaneb valmisolevate toodetega, tuleb valmis toodetelt välimus ja osa funktsionaalsust üle tuua. Jagatud funktsionaalsus võib olla näiteks rakendusse sisselogimine ja tõlgete süsteem. Funktsionaalsuse ja välimuse ühtlustamiseks tuleb valmisolevast tootest kood üles otsida, uude projekti kopeerida ja vajadusel kohandada. Protsessi tuleb iga projekti algfaasis korrata, mis aga kulutab aega.

Töös analüüsitakse erinevaid võimalikke lahendusi ühisosa jagatavaks tegemiseks. Üks võimalus on koostada ühisosa põhjal jagatavad moodulid, mida saavad erinevad projektid kasutada. Teise lahendusena uuritakse uue projekti malli koostamist, mis sisaldaks projektide ühisosa. Analüüsi käigus leitud lahenduste plussid ja miinused on toodud võrdlemiseks tabelisse, analüüsile tuginedes valitakse probleemi lahendamiseks kõige sobivam meetod.

Analüütilisele osale järgneb lahenduse realiseerimise osa. Ühisosa jagatavaks tegemise probleem lahendatakse, luues ühisosa sisaldav mall. Töös kirjeldatakse malli loomist, lahendamise käigus tekkinud probleeme ja malli arhitektuuri.

Töö viimases osas sõnastatakse töö tulemused, võetakse kokku tööle antud tagasiside ning tuuakse näiteid valmislahendusest.

# 1 Probleemi püstitus ja eesmärk

Peatükis sõnastatakse uuritav probleem. Kirjeldatakse probleemi uurimiseks ja lahendamiseks valitud metoodikat. Peatükis esitatakse töö eesmärgid.

## 1.1 Taust

Ettevõte, kus autor töötab, tegeleb e-arvete platvormi arendusega. Platvorm töötleb erinevat tüüpi dokumente. Dokumente töödeldakse vastavalt kliendi soovile ja dokumendi sisule. Süsteem koosneb erinevatest mikroteenustest, mis täidavad kindlaid ülesandeid, näiteks koostavad teenused XML-dokumendi põhjal arve PDF-kujutise. Süsteem teeb otsuseid konfiguratsiooni põhjal.

Platvormiga on seotud mitmed veebirakendused, mis on suunatud erinevatele kasutajagruppidele.

Süsteemi konfiguratsiooni muutmiseks on kasutusel *Management UI* nimeline veebirakendus, mida kasutavad süsteemi haldajad. Veebirakenduse abil saab muuta olemasolevaid konfiguratsioone ning lisada süsteemi uusi kliente. Veebirakendust kasutatakse erinevate olemasolevate mikroteenuste staatuse jälgimiseks. Dokumendi töötlemisel võib tekkida vigu, rakenduses saab vigaseid dokumente parandada ning vajadusel uuesti töödelda.

Enamik kliente saadavad platvormile dokumente, kasutades erinevaid liidestusi. Liidestused võimaldavad saata dokumente, aga ei tagasta detailset ülevaadet arvete staatuse kohta. Saadetud arvete kohta detailse ülevaate saamiseks on loodud veebirakendus Nessos. Rakenduse abil saavad kliendid dokumentide töötlemist osaliselt seadistada. Näiteks saab veebilehel luua reklaamikampaaniad, mis lisavad arvele erinevaid manuseid või pilte.

Klientidele, kes soovivad veebiliideses arveid koostada, on suunatud *Purchase Invoice Portal* ehk PIP. Lehte saab kasutada e-arvete koostamiseks ja saatmiseks. Ka PIP portaali abil saab arvetele erinevaid manuseid lisada.

Rakendused on arendatud mitmete tehnoloogiate baasil. Rakenduste kasutajaliidesed pole ühtlustatud. Ettevõtte soov on asendada olemasolevad rakendused uutega. Uute rakenduste arenduse eesmärk on ühtlustada kasutatavaid tehnoloogiaid. Teine eesmärk on ühtlustada kasutajaliideste väljanägemist, parandades nii kasutajakogemust.

Töö kirjutamise hetkeks on valitud baastehnoloogiad, mida projektid kasutama hakkavad. Valminud on esimesed kaks uut rakendust, mille välimus erineb üksteisest. Ettevõttel on tekkinud soov leida viise ühtse välimuse tagamiseks.

## 1.2 Probleemi püstitus

Ettevõttele on rakenduste ühtse välimuse ja funktsionaalsuse tagamine oluline, kuna see tagab hea kasutajakogemuse. Ühtne välimus parandab kasutajakogemust ja vähendab kasutaja tehtud vigade arvu [1].

Sarnaste kasutajaliideste tegemisel peab erinevates projektides sarnaseid kasutajaliidese komponente mitmeid kordi looma. Lihtsaim viis on koodi kopeerimine ühest projektist teise. Tihti ei saa koodi täpselt kopeerida, vaid seda tuleb kohanda, mis on aga ajamahukas. Selline lahendusviis tähendab, et ühte probleemi lahendatakse mitu korda.

Autor on kopeerimise meetodit kasutades märganud, et projektides on tihti erinevusi, mis tähendab, et kopeerimine võtab kaua aega. Näiteks HTML koodi kopeerides tuleb kontrollida, et HTML-is kasutatud *class*, *id* ja muudele atribuutidele vastab sobilik CSS stiil. Vajadusel tuleb muuta kopeeritud koodi või täiendada CSS stiili faile.

Probleemi lahendamine kopeerimise meetodil tähendab, et iga projekti algfaasis kulutatakse palju aega kopeerimisele, mitte uue funktsionaalsuse loomisele. See tähendab, et projekti valmistamiseks kulub kauem aega. Näiteks on autor mõlema projekti algfaasis kulutanud aega väliste teekide seadistamisele, sealhulgas tõlgete teegi seadistamisele.

Projektidele esitatud nõuded võivad ajas muutuda, näiteks võib tekkida soov muuta rakenduste välimust, tagades samas ühtne välimus üle rakenduste perekonna. Kui varasemalt on ühtset välimust tagatud läbi kopeerimise, siis on vaja uue välimuse kasutuselevõtuks kõikides projektides ükshaaval välimust muuta. Ka see toob kaasa ajakulu. Suur ajakulu võib tähendada, et ettevõtte loobub ühtse välimuse nõudest. Näiteks

on märgatavalt muudetud autori kahe valminud projekti puhul rakenduse välimusele esitatud nõudeid. Kuivõrd välimuse muutmine võtaks väga kaua aega, on ettevõtte esimese valminud projekti puhul ühtse välimuse nõudest taganenud.

## 1.3 Eesmärk

Töö eesmärk on analüüsida erinevaid võimalusi ühtse välimuse tagamiseks erinevatel veebilehtedel. Sobilike lahenduste leidmiseks määratakse lahendusele esitatavad nõuded. Parima lahenduse leidmiseks hinnatakse erinevate võimalike lahenduste eeliseid ja puuduseid.

Lahenduse puhul on väga tähtis, et see hoiaks kokku arendusele kuluvat aega. Leitud lahendus peab võimaldama jagada funktsionaalsust ja välimust nii, et arenduse ajal kuluks vähem aega koodi kopeerimisele ja kohandamisele.

Lahendus peab arvestama sellega, et tulevikus võivad nõuded välimusele või funktsionaalsusele muutuda. Lahendusega peab olema võimalik kergesti muuta kõikide projektide välimust.

Ettevõtte seatud piirangute tõttu peab lahendus olema kiiresti ja väikse meeskonnaga realiseeritav.

Erinevate lahendusmeetodite analüüsi põhjal otsustatakse, milline lahendusmeetod sobib kõige paremini autori tööle seatud eesmärkide täitmiseks, arvestades ettevõtte pandud piirangutega.

Töö praktilises osas lahendatakse probleem, kasutades analüütilises osas leitud lahendusmeetodit. Lahendusmeetodi rakendamise järgselt saab hinnata, kas analüütilises osas tehtud valikud on õigustatud.

## 1.4 Metoodika

Töös kirjeldatakse esmalt probleemi. Probleemi lahendamiseks esitatakse võimalikele lahendustele nõuded, mida peab valitud lahendus täitma. Lisaks lahenduse üldistele nõuetele tuuakse välja ka kitsendused, mis kehtivad autorile konkreetses ettevõttes probleemi lahendamiseks.

Erinevaid lahendusviise analüüsitakse lähtuvalt lahendusele esitatud eesmärkidest ja kitsendustest. Sobiva lahenduse valikuks koostatakse tabel, kus võrreldakse mitmeid lahendusi. Võrdlustabeli abil valitakse probleemi lahenduseks sobilik meetod.

Enne lahenduseks valitud meetodi realiseerimist analüüsitakse, mis on lahenduse nõutud funktsionaalsus ehk millised nõuded kehtivad kõikidele lahendust kasutama hakkavatele projektidele. Seejärel analüüsitakse, kuidas on kõige otstarbekam lahenduse erinevaid funktsionaalsuseid realiseerida. Seejärel arendatakse valitud lahendus.

Lahenduse valmimisel sõnastatakse tulemused ehk hinnatakse, kui hästi väljatöötatud lahendus toimib.

# 2 Lahendusmeetodi valik

Peatüki eesmärk on hinnata erinevaid lahendusmeetodeid ning leida neist sobivaim.

Parima meetodi leidmiseks on esmalt vaja kirjeldada, milliseid nõudeid peab sobiv lahendus täitma. Lisaks probleemi lahendusele esitatud üldistele nõuetele tuuakse välja ka lisapiirangud, mis kehtivad autori konkreetses projektis oleva probleemi lahendusele.

Esitatud nõuetele tuginedes analüüsitakse erinevaid lahendusi. Peatükis hinnatakse, kuivõrd lahendused vastavad esitatud nõuetele ning millised võivad olla kasutatud meetodi positiivsed ja negatiivsed mõjud rakenduse arenduses.

Viimases alapeatükkis võrreldakse kirjeldatud lahendusi ning leitakse nendest kõige sobivam. Parima lahenduse leidmiseks koondatakse erinevate lahendusviiside eelised ja puudused tabelisse. Tabeli abil leitakse, milline lahendusviis on kõige otstarbekam, arvestades lahendusele esitatud nõudeid ja ettevõtte piiranguid.

## 2.1 Lahendusele esitatavad nõuded

Probleemi lahendusele on autor seadnud mitu nõuet. Nõuded on seatud vastavalt probleemi püstituses esitatud probleemi kirjeldusele.

1. Lahendus peab hoidma kokku uute projektide arenduseks kuluvat aega.
2. Lahendus peab aitama tagada ühtset välimust erinevatel veebirakendustel.
3. Lahendus peab tagama, et projekte on võimalik tulevikus kergesti uuendada, kui nõuded välimusele või jagatud funktsionaalsusele muutuvad.

Autor peab lahendust valides arvestama ka konkreetsetest projektidest tulenevate piirangutega. Ettevõte, kus autor töötab, soovib viia erinevad tooted samadele tehnoloogiatele ning ühtsustada kasutajaliideste välimust.

Valitud tehnoloogia on TypeScript'il põhinev veebiraamistik Angular. Serveripoolse arenduskeelena kasutatakse Javat. Kasutaja tuvastamiseks ja kasutajaõiguste kontrollimiseks kasutatakse Keycloak teenust.

Töö kirjutamise hetkeks on kaks kasutajaliidest uuele platvormile viidud. Tekkinud on esimesed probleemid ühtse välimuse tagamiseks. Projektide tehnoloogiad on mõnevõrra erinevad ning eesmärk on neid tulevikus ühtlustada.

Järgmise kasutajaliidese üleviimiseks on kindlaks tehtud ärilised nõuded. Disainer on loonud esimesed vaated uuele kasutajaliidesele. Planeeritud on ka töö erinevate etappide tähtajad.

Meeskond, mis tegeleb kasutajaliideste uuendamisega, on suhteliselt väike. Meeskonnas on 4 arendajat, kes peavad paralleelselt uute kasutajaliideste arendamisega tegelema ka olemasoleva süsteemi täiendamise ja hooldusega. Seetõttu peab lahendus olema realiseeritav võimalikult väikese ajakuluga.

Alljärgnevalt on välja toodud ettevõttest ja projektidest tulenevad piirangud, millega peab autor sobiva lahenduse valikul arvestama.

1. Lahenduse välja töötamiseks on autoril aega üks kuu. Kuivõrd lahendust soovitakse kuu aja pärast kasutusse võtta, peab lahendus olema suhteliselt kiiresti teostatav.

2. Valminud lahenduse lähtekood ei tohi väljuda ettevõtte serveritest. See tähendab, et lähtekood ei ole avalik ning koodi jagamiseks ei saa kasutada ühtegi teenust, mis ei asu ettevõtte enda serverites.

3. Lahendus peab andma ajalist võitu suhteliselt väheste projektide puhul. Autor näeb, et valminud lahendust kasutatakse tulevikus 2 kuni 4 uue projekti puhul. See tähendab, et lahendus peab aitama vähendada arenduseks kuluvat aega juba kahe projekti puhul.

4. Lahendust kasutusele võtva meeskonna jaoks on tähtis, et valminud lahenduse hoiustamine võtaks minimaalselt ressursse. See tähendab, et valminud lahenduse kasutuselevõtuks ei ole vaja paigaldada serveritesse uusi teenuseid või tehnoloogiaid, mille hooldamine ja töös hoidmine nõuaks aega.

Arvestades, et autoril on mitmeid projektist tulenevaid piiranguid, ei pruugi autori jaoks kõige optimaalsem lahendus olla sobiv teistes projektides. Näiteks võib mõni teine lahendus osutada otsetarbekamaks, kui lahendust kasutatakse rohkemates projektides.

Lahenduse valikul tuleb arvestada, et autori seatud piirangud võivad tulevikus muutuda, mis võib tähendada, et lahenduseks sobilik meetod võib muutuda. Seetõttu tuleb uurida ka ühest lahendusest teisele migreerimise võimalusi.

## 2.2 Ühisosa jagamise viisid

Ühtsustatud välimuse ja jagatud funktsionaalsuse tagamiseks on mitmeid viise. Töös analüüsitakse kahte erinevat meetodit. Üks meetod on jagatud funktsionaalsuse pakendamine jagatavatesse moodulitesse, mida projektid saavad kasutada. Teise võimaliku lahendusena kaalutakse mallipõhist meetodit, kus luuakse mall, mis sisaldab projektide ühisosa.

Probleemi lahendamiseks on ka teisi meetodeid, mida ei analüüsita põhjalikumalt, kuna need ei täida lahendusele püstitatud eesmärke. Näiteks ei täida eesmärke projektide vahel lähtekoodi kopeerimine. Meetodiga ei saa efektiivselt täita seatud eesmärke, kuna meetod ei lahenda projektide ühtse uuendamise probleemi.

Veebipõhiste kasutajaliideste ühtse välimuse tagamiseks võib kasutusse võtta ka kõikide projektide vahel jagatud stiililehe (*CSS*), mida uuendataks tsentraalselt. Lahenduse abil saab luua ühtse välimusega veebilehti. Ühe stiililehe kasutamine aga ei aita ühtsustada veebilehtede funktsionaalsust, näiteks peab sellisel juhul kõikide veebirakenduste jaoks ise leidma lahenduse veebilehe tõlgitavaks tegemisel.

### 2.2.1 Ühisosa jagamine moodulites

Moodul on veebirakenduse osa, mis täidab mingit kindlat eesmärki. Selleks võib olla näiteks kasutajaliidese tõlkemoodul, mis vastutab selle eest, et kasutajale kuvatakse teksti tema valitud keeles. Moodulis loodud funktsionaalsust saavad kasutada teised moodulid. Moodulite kombineerimisel luuakse rakendus. Moodul ei pea tingimata olema kasutatav erinevates projektides, kuid siin peatükis analüüsitakse jagatud mooduleid, mida saab kasutada erinevates projektides.

Angulari raamistiku dokumentatsioon pakub välja, et moodulid on üks võimalik lahendus ühtsustatud kasutajaliidese tagamiseks ja funktsionaalsuse jagamiseks. Funktsionaalsuse moodulitesse viimine tagab, et moodul on iseseisev ja ei ole seotud rakenduse äriloogikaga. Samas on eraldi moodulite loomine keerukam, kui kogu rakenduse funktsionaalsuse ühes moodulis hoidmine. Moodulite loomine ja hooldamine võtab aega. See võib ära tasuda, kui mooduleid kasutatakse piisavalt paljudes rakendustes. [3]

Mooduleid saab jagada funktsionaalsuse järgi osadeks. Näiteks saab luua mooduli, kus on valmis kirjutatud kasutajaliidese komponendid või moodulid, mille abil saab kasutajaid tuvastada. Sellised moodulid on iseseisvad ja nende funktsiooni on lihtne mõista. Igas projektis saab valida, milliseid mooduleid on projektis vaja. Nii saab ebavajalikud moodulid projektist välja jätta.

Selleks et moodul projektis kasutusele võtta, on vaja seda seadistada. Näiteks rakendusele tõlgete süsteemi jaoks mõeldud mooduli lisamiseks peab rakendus kirjeldama, kus asuvad tõlked ja millised keeled on rakenduses toetatud. Rakendus annab moodulile teada, milline keel on valitud ja teavitab moodulit valitud keele muutustest. Mooduli seadistamiseks vajalik kood on igas projektis erinev, kuivõrd konkreetse projekti vajadused võivad olla erinevad.

Moodulit luues on oluline mõista, kuidas seda kasutama hakatakse. Moodulit disainides tuleb otsustada, milline on mooduli väline rakendusliides. Moodulit luues tuleb aru saada, kas mingi osa funktsionaalsusest peab olema konfigureeritav või võib loodud funktsionaalsus püsida konstantsena. Näiteks vajab seadistamist eelpool kirjeldatud tõlgete moodul, samas ei vaja seadistamist moodul, mis valideerib, kas etteantud kontonumber vastab IBAN kontonumbri standardile. Moodulite puhul tuleb arvestada, et need peavad olema piisavalt paindlikud, et vastata kõikide kasutajate vajadustele, sest mooduli kasutaja ei saa muuta mooduli sees olevat koodi. Selleks et teada kõiki mooduli kasutusjuhte, on vaja seda esmalt paaris erinevas rakenduses kasutada. Moodulit kasutades on võimalik mõista, kas selle rakendusliides on kohane ning kas mingi osa moodulist peaks olema muudetav.

Moodulit arendades tuleb tähelepanu pöörata täpsele dokumentatsioonile, kuna mooduli funktsionaalsust kasutaval arendajal on keeruline ligi pääseda mooduli lähtekoodile. Lähtekoodi lugemiseks peab ta leidma mooduli projekti, valima versioonihaldusvahendi

abil õige lähtekoodi versiooni ning alles seejärel saab arendaja tutvuda funktsionaalsusele vastava lähtekoodiga. Kuna lähtekoodi avamise ja lugemise protsess on ajamahukas, siis püütakse seda praktikas vältida. Seega peab mooduli dokumentatsioon olema asjakohane ning kirjeldama täpselt mooduli funktsioonide käitumist ja võimalikke kõrvalmõjusid. Täpse ja ajakohase dokumentatsiooni loomine ja hooldamine on väga ajamahukas.

Mooduli arendamist ja kasutamist hõlbustab näidisrakenduse loomine. See lihtsustab rakenduses jagatava mooduli kasutuselevõttu, kuna arendaja saab mooduli kasutamisel eeskujuks võtta näidises oleva lähtekoodi. Samuti saab näidisrakenduse abil testida, kas moodul töötab nii, nagu arendaja eeldas. Näidisrakenduse puhul tuleb arvestada, et ka selle loomiseks ja uuendamiseks kulub aega.

Moodulit saab lisada projektile, mille arendusega on juba alustatud. Projekti luues ei ole vaja ette teada, milliseid mooduleid projekt kasutama hakkab. Malli kasutades tuleb eelnevalt teada, milliseid mooduleid projektis kasutatakse. See on oluline eelis, kuna kõik nõuded ei pruugi olla ette teada.

Moodulis olevat koodi on võimalik peale valmimist parandada või täiendada. Peale vea parandamist saab luua moodulist uus versiooni, mida saavad teeki kasutavad rakendused kasutama hakata. See tähendab, et vea ilmnemisel on vaja see vaid ühes kohas parandada.

Mooduli täiendamisel tuleb moodulist uus versioon luua ning üles laadida. Seejärel tuleb moodulit kasutavas projektis uuendada mooduli versioon ja uus versioon alla laadida. Mooduli uuendamise järgselt saab testida, kas tehtud muudatused töötavad ootuspäraselt, vastasel juhul tuleb moodulit taaskord uuendada ning üles laadida. Jagatud mooduli uuendamiseks on seda vaja pakendada ja versioniseerida. Seetõttu võtab jagatud mooduli uuendamine ja testimine kauem aega, kui kuluks rakenduse lähtekoodis oleva mooduli muutmiseks.

Selleks et moodulit saaks erinevates projektides kasutada, on mitmeid võimalusi. Erinevatel lahendustel on eelised ja puudujäägid, mida järgnevalt analüüsitakse.

#### 2.2.1.1 Npm registris jagatud moodulid

Moodulite jagamiseks saab nendest luua npm pakid. Npm on laialt levinud JavaScripti pakkide jagamise tarkvara. Npm pakke kasutatakse paljude avalikult jagatavate

JavaScript moodulite jagamiseks. Paki jagamiseks on vaja see üles laadida npm registrisse. Andmebaasist on võimalik pakke alla laadida nime ja versiooni järgi. [4]

Npm registrisse on võimalik üles laadida mooduli erinevaid versioone, mis tähendab, et mooduli koodi muutmisel ei pea klient seda koheselt kasutusse võtma. Moodulit kasutavad projektid saavad uuenduse kasutusele võtta erinevatel ajahetkedel.

Npm kasutab tarkvara erinevate versioonide nimetamiseks semantilist versioniseerimist [4]. Versiooni nimi koosneb kolmest numbrist: suurversioonist, väikeversioonist ja paigast (kujul suurversioon.väikeversioon.paik), näiteks 1.4.3. Suurversiooni (ingl *major*) muutus tähendab, et tarkvara omadused või rakendusliides on muutunud. Väikeversiooni (ingl *minor*) muudatused lisavad rakendusele uusi omadusi täiendades rakendusliidest, kuid olemasolevad tarkvara omadused ja liidesed ei tohi muutuda. Paiga (ingl *patch*) muudatused on kasutusel rakenduste pisivigade parandamiseks. [5], [6] Semantilise versioniseerimise eeliseks on see, et klient saab uuendada pakkide väikeversioone ja paiku, kartmata, et tema rakendus läheks selle tõttu katki.

Paki installimisel registrist on võimalik määrata ligikaudne versioon, näiteks on võimalik öelda, et soovitakse kasutada mooduli kolmandat suurversiooni ning kõige hiljutisemat väikeversiooni. Niimoodi talitledes laetakse registrist alla mooduli kõige uuem lubatud versioon. [4]

Npm registrist on olemas avalik versioon, mille kasutamine on tasuta, kuid arvestada tuleb sellega, et üles laetud pakid on jagatud nii, et need on kõigile nähtavad. Pakkide privaatselt jagamiseks on erinevaid võimalusi. Privaatseks jagamiseks on olemas tasulisi teenuseid, mida kasutades saab luua registreid, milles olevaid pakke näevad vaid selleks loa saanud isikud. Teenuste puhul tuleb arvestada, et kood laetakse teenusepakkuja serveritesse. [4]

Tasuliste teenuste kasutamise alternatiiviks on olemas npm registrid, mida on võimalik paigaldada enda serverisse, näiteks Verdaccio [5]. Selliste registrite puhul saab paigaldaja ise otsustada, kellel on luba pakke üles ja alla laadida. Registri saab paigaldada enda serverisse, mis tähendab, et koodi hoitakse ettevõtte kontrollitud keskkonnas. Npm registri enda serverisse paigaldamisel peab arvestama, et nagu iga tarkvaraga, on ka npm registreid vaja perioodiliselt hooldada ja uuendada. Seega tuleb enda serverisse paigaldatud registri puhul arvestada, et see toob kaasa ajakulu.

#### 2.2.1.2 Npm pakkide jagamine ilma registrita

Npm pakkide alla laadimiseks ei pea pakki tingimata registrisse üles laadima, allalaetavale pakile on võimalik viidata ka gzip arhiivi või Git repositooriumi veebiaadressi kaudu. Git repositooriumit pakina kasutades saab määrata erinevaid mooduli versioone, viidates kindlale Git commit räsile või Git tagile. Viidates privaatsetele Git repositooriumitele on võimalik kasutada pakke nii, et puudub vajadus privaatse npm registri järgi. [4]

Kui projektis viidatakse pakile Git commit räsi või gzip arhiivi aadressi järgi, siis peab versiooni uuendamiseks klient ise kontrollima, kas pakist on olemas uus versioon ning selle olemasolul käsitsi versiooni uuendama. Kui pakile viidatakse Git tagi järgi ning tag kasutab semantilist versioniseerimist, siis saab pakile viidata ligikaudse versiooni järgi. [4]

Angular raamistiku mooduleid pakendades kompileeritakse esmalt lähtekoodiks olev TypeScript kood JavaScript koodiks [8]. Seega ei saa moodulit projektis kasutusele võtta, viidates mooduli lähtekoodi sisaldavale Git repositooriumile. Kompileeritud koodi jagamiseks tuleb teha eraldi Git repositoorium, kuhu kopeeritakse pakendatud mooduli kood ning millele saavad omakorda projektid viidata. Kompileeritud koodi kopeerimine ja kahe erineva repositooriumiga käsitsi töötamine on ajamahukas. Protsessi saab suhteliselt kergesti käsurea skriptide abil automatiseerida.

Kompileeritud mooduleid on kerge jagada arhiivina. Npm käsureaprogrammi abil saab luua sobivas formaadis arhiivi käsuga *pack* [4]. Mooduli projektide vahel jagamiseks tuleb valminud arhiiv veebiserverisse üles laadida. Seejärel saavad projektid arhiivile viidata, kasutades arhiivi veebiaadressi.

#### 2.2.1.3 Monorepositooriumis jagatud moodulid

Npm pakkide koostamise alternatiiv on monorepositooriumi kasutamine. Monorepositoorium on meetod versioonihaldustarkvara kasutamiseks. Repositooriumit kasutatakse nii, et väga paljud projektid asuvad ühes repositooriumis [6]. Monorepositooriumit kasutades saab hoida taaskasutatavaid mooduleid repositooriumis koos projektidega, mis neid mooduleid kasutavad.

Monorepositooriumis on lihtne luua mooduleid. Võrreldes npm pakkidega ei ole moodulite loomiseks vaja eriteadmisi moodulite pakendamisest ja avaldamisest, sest moodulit kasutavad projektid viitavad otse mooduli lähtekoodile.

Monorepositooriumit saab kasutada nii, et moodulist on kasutuses vaid üks versioon. Ainult ühe versiooni olemasolu tähendab seda, et moodulis parandusi tehes on tagatud, et kõikides moodulit kasutatavates programmides on viga parandatud.

Eri versioonide puudumine tähendab, et moodulit muutes tuleb kontrollida, kus ja kuidas moodulit kasutatakse ja millised on muutmise mõjud. Moodulit uuendades peab olema tagatud, et kõik moodulit kasutavad rakendused jätkavad töötamist. Praktikas tähendab see, et moodulis suurte muudatuste tegemisel tuleb arvestada, et kõikide klientide koodi tuleb samuti uuendada, mis võib olla sõltuvalt muudatusest väga ajakulukas.

### 2.2.2 Malli koondatud ühisosa

Teine võimalik meetod ühisosa jagatavaks tegemisel on uue projekti malli koostamine. Projekti malli saab võtta aluseks uue projekti tegemisel. Mallis on implementeeritud jagatud funktsionaalsus. Samuti on mallis seadistatud erinevad projektides kasutatavad pakid.

Erinevalt jagatud moodulitest ei saa projekt kasutada mitut malli korraga. Seetõttu peab mallis olema kogu vajaminev jagatud funktsionaalsus. Funktsionaalsus võib olla siiski jagatud erinevatesse kaustadesse ja moodulitesse. Nii saab projekt, mis teatud funktsionaalsust ei vaja, kustutada ära funktsionaalsusele vastava kausta või mooduli. Mallis võib moodul olla kasutuses mitmes kohas ja selle ümber kirjutamine võtab aega. Mallis olevate moodulite kustutamine võtab rohkem aega, kui projektist puuduoleva paki lisamine.

Mallis on võimalik seadistada ära mitmed välised pakid ja moodulid, mida projektides kasutatakse. Näiteks saab mallis seadistada ära tõlgete mooduli, mis vajab töötamiseks lisainformatsiooni. Moodulit seadistav kood on projektides tihti korduv. Samuti võtab seadistamine aega, kuna selleks tuleb tutvuda vastava mooduli dokumentatsiooniga. Mall aitab kokku hoida moodulite seadistamiseks kuluvat aega.

Mallis olevat koodi on lihtne projektile kohandada. Kui konkreetsele projektile ei sobi mallis ette antud välimus või funktsionaalsus, siis saab projektis malliga kaasa tulnud

koodi ümber kirjutada. Pakendatud moodulit ei saa ümber kirjutada, selle puhul tuleb ette näha, kuidas moodulit kasutama hakatakse. Malli kasutades ei pea nii selgelt ette nägema, milleks malli kasutama hakatakse, kuna selle sisu on lihtsam muuta.

Mall määrab projekti struktuuri. See tähendab, et kõik projektid, mis on loodud malli kasutades, järgivad sarnast struktuuri. Sarnase struktuuriga projektide alustamine on lihtsam, kuna struktuuri ei ole vaja eraldi õppida, arendaja teab, kust leida kindlat funktsionaalsust.

Projektis kasutatavate tööriistade seadistamiseks on projekti juurkaustas mitmeid seadistuseks vajalikke faile. Näiteks on igas projektis olemas failid, mis kirjeldavad, kuidas lähtekoodi kompileerida või millised on lähtekoodi stiilile esitatud nõudmised. Ühe meeskonna projektides on kasulik, kui projektid on seadistatud võimalikult sarnaselt. Sarnane seadistus tähendab, et arendajal on lihtsam eri projektidega tööd teha. Selliseid seadistuse faile saab luua mallis, mis tagab, et projektid kasutavad sarnast seadistust.

Valmis malli kasutuselevõtuks on palju võimalusi. Järgnevalt analüüsitakse erinevate rakendamisviiside eeliseid ja puuduseid.

#### 2.2.2.1 Malli jagamine arhiivina

Kõige lihtsaim viis malli jagamiseks on malli failid arhiveerida, pakkides see näiteks ühtseks zip failiks. Malli kasutuselevõtmiseks pakitakse arhiiv lahti ning asendatakse vajadusel malli failides projekti nime hoidvad kohatäitjad.

Sellise jagamisviisi suurimaks miinuseks on see, et kui kasutusel olevast mallist leitakse viga, ei saa seda korraga kõikides projektides parandada. Samuti ei saa muuta korraga kõikide projektide välimust. Uuendamiseks saab teha mallist uue versiooni, malli põhjal loodud projektidele tuleb uuendused käsitsi üle kanda.

#### 2.2.2.2 Mall Git *fork* meetodi abil

Selleks et mallis tehtud uuendusi olemasolevatesse projektidesse lihtsamalt üle kanda, saab hoiustada malli ja projekti lähtekoodi Git repositooriumis.

Uue projektiga alustades tehakse malli repositooriumist koopia. Loodud koopia põhjal arendatakse uut rakendust, lisades koopiasse projektispetsiifilist koodi. Repositoorium kopeeritakse viisil, kus luuakse uus serveripoolne koopia ning samas säilitatakse ka viide

serveris asuva malli Git repositooriumile. Sellist kloonimist nimetatakse *fork*'imiseks. Niimoodi loodud projekti saab mallis tehtud muudatusi kergelt üle kanda, kasutades Git merge funktsiooni. [8]

Meetod võimaldab malli kasutavate projektide poolautomaatset uuendamist. Rutiinse töö hulk väheneb ning projektide uuendamine on kiirem ja lihtsam.

Uuendamine Git *fork* meetodil eeldab, et projekti ja malli failipuud on suhteliselt sarnased. See tähendab, et mallis loodud faile ei tohiks tõsta teistesse kaustadesse ning projekti lisatakse uut funktsionaalsust peamiselt uutesse kaustadesse ja failidesse. Vastasel juhul on võimalus, et mallis ja projektis tehtud muudatused on konfliktsed ning projekti ja malli sünkroniseerimiseks tuleb käsitsi muudatused üle kontrollida.

Selleks et vähendada konflikte malli ja projekti koodi uuendamisel, tuleb loodavas mallis hoolikalt läbi kaaluda failipuu, et projektis puuduks vajadus faile ümber tõsta. Üheks võimaluseks on failide jagamine funktsionaalsuse järgi eraldi kaustadesse. Läbimõeldud struktuur aitab ka arendades vajalikke faile kergemini üles otsida ning kiirendab seeläbi arendusprotsessi. See tähendab, et malli struktuuri peaks läbi mõtlema ka kõikide teiste malli jagamise meetodite puhul ning piirang ei esita malli arendusele uusi nõudeid.

Meetodi poolautomaatne sünkroniseerimise protsess annab võimaluse arendajal detailselt üle kontrollida, millised muudatused mallist projekti üle kantakse. See tähendab, et arendaja teab, mis on projektis muutunud ning saab testida, kas tehtud muudatused töötavad nii, nagu muudatuse tegija on eeldanud.

#### 2.2.2.3 Projekti generaatorid

Mallist projekti tegemiseks on olemas ka erinevaid projekti generaatoreid. Generaatoreid kasutades on võimalik automatiseerida malli kohandamist konkreetsele projektile. Generaator loob projektipõhja mitmete parameetrite järgi. See suudab näiteks automaatselt ümber nimetada mallis olevaid sõnesid, mis tuleb asendada projekti nimega. Samuti saab generaatori abil teha selliseid malle, kus on võimalik valida, millist funktsionaalsust soovitakse projekti lisada ning milline funktsionaalsus jääb projektist välja.

Generaatorid võimaldavad suurt paindlikkust projekti loomisel. See tähendab aga, et mall, mille põhjal generaator projektipõhja loob, on keerukam ning selle arendus võtab kauem aega.

Malligeneraatorit kasutades kaotatakse ära projektide uuendamise lihtsus. Projektide uuendamine pole selle meetodi puhul kergesti automatiseeritav ning võtab seetõttu rohkem aega kui Git *fork* põhimõttel loodud projektid.

## 2.3 Jagatavaks tegemise meetodi valik

Nii moodulite kui malli lahendus täidavad esitatud eesmärke ehk kiirendavad projekti arenduseks kuluvat aega, aitavad tagada ühtset välimust erinevates rakendustes ja võimaldavad uuendada jagatud koodi peale selle kasutusele võtmist.

Malli kasutamise meetodite valikut piirab kolmas nõue ehk kergesti uuendamise nõue. Seetõttu sobib malli kõikidest kasutusmeetoditest vaid Git *fork* meetod.

Kuivõrd lahendusele esitatud eesmärkidele vastab mitmeid lahendusi, teeb autor valiku lähtuvalt projektist tulenevatest lisapiirangutest, mis on välja toodud peatükis 2.1.

Piirangutele kõige paremini vastava lahenduse väljaselgitamiseks on koostatud tabel (Tabel 1), kus võrreldakse erinevate meetodite sobivust, arvestades piirangutega.

Tabel 1. Ühisosa jagamise meetodite võrdlus.

| | **Ühisosa jagatavate moodulitena** | **Ühisosa mallis** |
|---|---|---|
| **Realiseerimise keerukus** | Realiseerimine on keeruline.<br>Nõuab põhjalikku analüüsi, et otsustada, millist funktsionaalsust moodulitesse panna, kuidas funktsionaalsust moodulite vahel jagada, kuidas hakkab klient mooduleid kasutama, kuidas ja kas peab klient saama mooduli funktsionaalsust konfigureerida.<br>Moodulite pakendamine vajab eriteadmisi.<br>Moodulite monorepositooriumis jagamine tähendab suuri muudatusi kogu koodibaasis. | Realiseerimise on lihtsam kui moodulitega lähenemine.<br>Nõuab samuti analüüsi, milline funktsionaalsus peaks mallis olema ning kuidas seda kaustadesse jagada.<br>Malli loomine ja kasutamine on lihtsam, sest arendajal ei ole vaja malli koostamiseks eriteadmisi. |

| | **Ühisosa jagatavate moodulitena** | **Ühisosa mallis** |
|---|---|---|
| **Valmimiseks vajalik aeg** | Arendus võtab kauem aega, kuna moodulite loomisel on vaja eraldada valmis rakendustest moodul.<br>Mooduli rakendusliidese testimiseks ja demonstreerimiseks on vaja luua erinevaid näidisrakendusi. | Analüüsiks kuluv aeg on sarnane moodulite tegemise meetodiga.<br>Arenduseks kulub vähem aega, kuna malli aluseks saab võtta mõne valminud rakenduse, milles on suur hulk mallile esitatud nõudeid juba täidetud.<br>Rakendusest tuleb eemaldada malli jaoks üleliigne kood ehk rakenduse spetsiifiline kood. |
| **Lahenduse kasutamiseks vajalik infrastruktuur** | Projektile seatud piirangu kohaselt peavad teegid olema jagatavad nii, et need ei välju ettevõtte serveritest.<br>Nõudele vastab Git repositooriumite või arhiivide põhine moodulitele viitamine.<br>Nõudele ei vasta lahendusviis, kus kasutatakse privaatset npm repositooriumit. Hetkel ei ole ettevõttes sellist lahendust kasutuses, seega on vaja teha analüüs, et otsustada, millist tarkvara selleks kasutada. Lisaks tuleb arvestada, et repositooriumi tarkvara peab õppima kasutama ning selleks, et tarkvara töötaks, peab seda hooldama. | Malli on võimalik hoida Git repositooriumis. Git on ettevõttes laialdaselt kasutuses ja ei lisa uusi nõudeid ettevõtte infrastruktuurile. |
| **Kasutuselevõtu keerukus** | Moodulite lisamine projekti on lihtne. Keerukamate moodulite puhul tuleb arvestada moodulite projekti integreerimise ajakuluga.<br>Mooduleid on võimalik lisada ka olemasolevatele projektidele. | Malli kasutamine on uute projektide puhul lihtne. Mallis on erinevad välised moodulid eelnevalt integreeritud ning sellele ei kulu projekti alguses aega.<br>Malli ei saa kasutusele võtta olemasolevatest projektidest. |
| **Parandused peale kasutuselevõttu** | Moodulist saab teha uusi versioone, kus on vead parandatud.<br>Projektis kasutatava mooduli versiooni muutmine on lihtne. | Mallis saab vigu paranda.<br>Git *fork*'i kasutades saab parandusi kasutusele võtta ka malli kasutavatesse projektides. Paranduste sisseviimine on siiski vaid poolautomaatne ning võtab võrreldes mooduli versiooniuuendusega kauem aega. |

Ülaltoodud tabeli (Tabel 1) põhjal saab järeldada, et projektile esitatud lisapiirangute tõttu sobivad kõikidest mooduli jagamise meetoditest vaid Npm pakendamise meetodid, kus ei kasutata Npm registrit.

Võrdlustabelis (Tabel 1) on näha, et malli arendus, jagamine ja kasutuselevõtt uutes projektides on võrrelduna moodulitega lihtsam ning aitab projekti algfaasis aega kokku hoida, kuna erinevaid moodulid on eelnevalt seadistatud. Moodulitega lähenemise eelisteks on see, et mooduleid saab kasutusele võtta ka olemasolevates projektides ning nende uuendamine on lihtsam.

Arvestades, et lahendus on vaja välja töötada suhteliselt lühikese aja jooksul ja see peab andma ajavõidu juba väheste projektide puhul, on mõistlik valida lahendus, mille väljatöötamine ja uuendamine võtab minimaalselt aega. Optimaalse lahenduse leidmiseks hinnatakse erinevate meetodite väljatöötamiseks ja uuendamiseks kuluvat aega. Ajahinnangud koostatakse CGI Eesti e-arvelduse meeskonna näitel.

Jagatavate moodulite arendamisel arvestatakse, et ühine funktsionaalsus jagatakse seitsmeks erinevaks mooduliks. Esimese mooduli puhul arvestatakse erinevate arendusetappide ajakulu suuremaks, kuna autor pole varem pakendatud moodulit loonud ning arvestab seetõttu ajakulu sisse ka uute teadmiste omandamiseks kuluva aja.

Mooduli valmistamiseks kuluv ajakulu jagatakse 4 erineva kategooria vahel.

1. Mooduli arendus. Arenduse käigus täpsustatakse, mis on mooduli nõuded ning kuidas esitatud nõudeid täita. Samuti otsustatakse, milline on mooduli rakendusliides. Etapi lõpuks valmib kood, mis täidab esitatud nõuded.

2. Kooste automatiseerimise ajakulu. Kategooria alla arvestatakse aeg, mis kulub mooduli kooste tegemise, testimise ja jagamiseks vajalikke sammude automatiseerimiseks nii, et see töötaks ettevõtte pidevintegratsiooni (ingl *CI Continuous Integration*) serveris.

3. Mooduli dokumenteerimine. Dokumenteerimise alla kuulub mooduli lähtekoodi dokumenteerivate kommentaaride lisamine. Lähtekoodis olevate kommentaaride abil saab genereerida mooduli rakendusliidest kirjeldava dokumendi. Lisaks tuleb koostada üldine mooduli dokumentatsioon, mis kirjeldab, millist probleemi moodul lahendab ning kuidas moodulit teistesse rakendustesse integreerida.

4. Näidisrakenduse loomine. Võimalikult lihtsa näidisrakenduse loomine, mis demonstreerib mooduli funktsionaalsust. Näidisrakendus toetab mooduli dokumentatsiooni. Näidisrakenduse loomine on oluline ka selleks, et moodulit testida.

Lisaks mooduli arenduseks kuluvale ajale hinnati ka seda, kui kaua võtab mooduli projektis kasutuselevõtt aega.

Mooduli ajahinnangud on välja toodud alljärgnevas tabelis (Tabel 2). Tabelis hinnatakse ajakulu tundides. Lahenduse väljatöötamisele kulub hinnanguliselt 17 tööpäeva, kõikide moodulite projekti lisamiseks ja seadistamiseks kulub ligikaudu poolteist tööpäeva.

Tabel 2. Jagatud moodulitega lahenduse väljatöötamise ajahinnangud tundides.

| **Mooduli nimi** / **Etapp** | **Kokku** | Komponentide moodul | Tõlgete moodul | Autentimise moodul | Konfiguratsiooni moodul | Mitteaktiivse sessiooni tuvastamise moodul | 404 lehekülje moodul | Korduma kippuvate küsimuste moodul |
|---|---|---|---|---|---|---|---|---|
| Arendus | **72** | 30 | 8 | 8 | 8 | 8 | 4 | 6 |
| Automatiseerimine | **20** | 8 | 2 | 2 | 2 | 2 | 2 | 2 |
| Dokumenteerimine | **16** | 6 | 2 | 2 | 2 | 1 | 1 | 2 |
| Näidisrakenduse loomine | **24** | 6 | 3 | 3 | 6 | 2 | 2 | 2 |
| **Väljatöötamine kokku** | **132** | **50** | **15** | **15** | **18** | **13** | **9** | **12** |
| **Projektis kasutuselevõtu aeg** | **12** | **3** | **1** | **2** | **3** | **1** | **1** | **1** |

Malli väljatöötamiseks kuluva aja hindamiseks jagati malli arendus erinevateks etappideks, millele anti eraldi ajahinnangud. Malli väljatöötamiseks kuluva aja leidmiseks liideti erinevate etappide ajahinnangud kokku. Lahenduse väljatöötamise ja projektis kasutuselevõtu ajakulu on välja toodud alljärgnevas tabelis (Tabel 3). Hinnanguliselt kulub lahenduse väljatöötamiseks ligikaudu 11 tööpäeva, malli põhjal uue projekti loomine võtab hinnanguliselt aega 2 tundi.

Tabel 3. Malli lahenduse väljatöötamise ajahinnang tundides.

| **Väljatöötamise etapp** | **Ajahinnang tundides** |
|---|---|
| Baasrakenduse valik | 2 |
| Üleliigse funktsionaalsuse eemaldamine | 4 |
| Refaktoreerimine | 30 |
| Puuduolevate nõuete täitmine | 30 |
| Näidislehe tegemine | 8 |
| Dokumenteerimine | 8 |
| Automatiseerimine | 4 |
| **Väljatöötamine kokku** | **86** |
| **Projektis kasutuselevõtu aeg** | **2** |

Valitud lahendusi tuleb võrrelda ka kopeerimise meetodiga. Kopeerimise meetod on malli arendusega sarnane, sest mõlemad meetodid kopeerivad olemasolevatest projektidest koodi ja kohandavad seda. Meetodeid eristab see, et kopeerimise meetodi puhul kopeeritakse koodi iga uue projekti puhul, kuid malli puhul tehakse seda üks kord. Seega saab kopeerimise meetodile ajahinnangu andmisel eeskujuks võtta malli lahenduse väljatöötamise ajahinnangu. Võrreldes malli meetodiga ei vaja kopeerimine eraldi dokumenteerimist ning näidislehe tegemist. Kopeerimise meetodi puhul hinnatakse, et iga projekti puhul kulub kopeerimiseks ligikaudu 8 tööpäeva.

Lisaks lahenduse väljatöötamise ajakulule tuleb arvestada ka uuenduste tegemiseks kuluva ajakuluga. Ajahinnangud on antud, võttes keskmise mallis tehtava muudatuse sisseviimise ajaks ühe tööpäeva ehk 8 tundi. Mallis ja projektis tehtava muudatuse jaoks kulub võrdselt aega, seega kulub kopeerimise meetodi puhul esimeses projektis muudatuse tegemiseks samuti 8 tundi, järgnevates projektides on muudatust lihtsam teha ning seetõttu hindame muudatuse sisseviimise ajaks 4 tundi. Moodulis on muudatuse tegemisel vaja tähelepanelikult jälgida, et mooduli dokumentatsioon ja näidisrakendus oleksid ajakohased. Seetõttu kulub jagatud mooduli uuendamiseks mõnevõrra kauem aega, hinnanguliselt 9 tundi. Rakendustes, kus moodul on kasutuses, kulub mooduli uuenduseks ligikaudu tund. Tunni aja jooksul uuendatakse rakenduses mooduli versioon ning kontrollitakse, et moodul töötaks nii, nagu on planeeritud. Malli kasutavat projekti

uuendatakse Git *fork* meetodil, mis võtab versiooni uuendamisest mõnevõrra kauem aega. Mallil põhineva projekti uuendamiseks ja testimiseks planeeritakse ligikaudu 2 tundi.

Valem (1) näitab, kuidas hinnata lahenduse kasutuselevõtu kogukulu (valemis KK) ehk aega, mis kulub lahenduse väljatöötamiseks ja kõikides projektides kasutusele võtuks, kui on teada:

- VK ehk lahenduse väljatöötamise ajahinnang,
- RK ehk keskmine lahenduse ühes rakenduses kasutuselevõtu ajahinnang,
- p ehk lahendust kasutavate projektide arv.

$$KK = VK + p * RK \quad (1)$$

Lisaks kasutuselevõtu kogukulule peab arvestama ka lahenduse uuendamisele kuluva ajaga. Valem (2) näitab, kuidas hinnata uuendustele kuluvat aega etteantud perioodil (valemis PUK), kui on teada:

- UK ehk keskmine uuenduse väljatöötamiseks kuluv aeg ehk muudatuste tegemiseks kuluv aeg moodulis või mallis,
- RUK ehk keskmine ühes rakenduses uuenduse kasutuselevõtuks kuluv aeg,
- n ehk perioodi jooksul tehtavate uuenduste arv,
- p ehk projektide koguarv.

$$PUK = \mathrm{n}(\mathrm{UK} + \mathrm{p} * \mathrm{RUK}) \quad (2)$$

Valemite põhjal saab välja hinnata, kui palju aega kulub erinevate lahenduste kasutuselevõtuks ning kui palju kulub aega iga-aastasteks uuendusteks. Tabelis on välja toodud ajahinnangud, mis on arvutatud eeldusel, et lahendust kasutavad 3 erinevat rakendust ning lahendusi uuendatakse aastas 6 korda (Tabel 4).

Tabel 4. Erinevate lahenduste ajahinnangud tundides.

| | **Moodulid** | **Mall** | **Kopeerimine** |
|---|---|---|---|
| Lahenduse väljatöötamise aeg | 132 | 86 | 0 |
| Ühes rakenduses kasutuselevõtu aeg | 12 | 2 | 64 |
| **Kasutuselevõtu ajakulu kokku** | **168** | **92** | **192** |
| Keskmine uuenduse väljatöötamiseks kuluv aeg | 9 | 8 | 8[1] |
| Keskmine uuenduse kasutuselevõtuks kuluv aeg rakenduses | 1 | 2 | 4[2] |
| **Uuendamise aastane ajakulu** | **72** | **84** | **96** |

Tabel 4 välja toodud ajahinnangute põhjal saab järeldada, et malli meetodit on võimalik, võrreldes teiste meetoditega, oluliselt kiiremini kasutusele võtta. Kolme projekti puhul on moodulite meetodi aastane uuendamise kulu malli meetodist ligikaudu poolteist tööpäeva väiksem. Mõnevõrra kiirem uuendamine tähendab, et lahenduste kogukulud võrdsustuksid umbes 7 aasta pärast.

Moodulite meetod aitaks aega kokku hoida, kui loodavaid projekte oleks rohkem ning rakenduste ühisosa on vaja tihti muuta. Kuna töös loodud lahendust kasutatakse vähestes projektides ning uuendatakse suhteliselt harva, siis valitakse lahenduseks malli meetod. See täidab kõiki lahendusele esitatud nõudeid, võimaldades seejuures kiiremat väljatöötamist ja lihtsamat kasutuselevõttu, kuid säilitades ka uuendamise võimaluse.

Malli meetodi puhul tuleb arvestada, et seda ei saa kasutada olemasolevates projektides. Mooduleid kasutades saaks lihtsustada koodi olemasolevates projektides, asendades projektis olev kood jagatud moodulitega. Küll aga saab vajadusel viia malli etappide kaupa eraldi jagatavatesse moodulitesse. Nii saab tulevikus ühtlustada koodibaasi ka olemasolevate projektide puhul. Kui mallil põhinevate projektide uuendamine osutub liiga ajakulukaks, saab tulevikus malli asendada moodulitega.

[1] Kopeerimise puhul mõeldakse siin esimeses rakenduses uuenduse väljatöötamiseks kuluvat aega.

[2] Kopeerimise puhul mõeldakse siin järgnevatesse rakendustesse kopeerimiseks, kohandamiseks ja testimiseks kuluvat aega.

# 3 Lahenduse väljatöötamine

Peatükis kaardistatakse erinevate rakenduste ühisosa ehk leitakse mallile esitatud funktsionaalsed nõuded. Leitud nõuete põhjal valitakse baasrakendus, mille põhjal malli arendatakse. Peatükis analüüsitakse, kuidas on kõige optimaalsem nõudeid täita ning kirjeldatakse malli valmimise protsessi.

## 3.1 Ühisosa kaardistamine

Enne malli arendusega alustamist tuleb leida olemasolevate ja tulevikus tehtavate projektide ühisosa. Leitud ühisosa põhjal saab otsustada, milline peab olema malli funktsionaalsus.

Nõutud funktsionaalsuse leidmiseks moodustatakse nimekiri valmisolevates rakendustes olevatest funktsionaalsustest, mis ei ole seotud konkreetse rakenduse äriloogikaga. Sarnane nimekiri koostatakse ka järgmise arendatava projekti kohta. Nimekirjadest leitakse funktsionaalsused, mis esinevad rohkem kui pooltes nimekirjas olevates punktides. Ühiste punktide alusel koostatakse mallile esitatud nõuete loend.

Leitud ühine funktsionaalsus ehk mallile esitatud nõuded on välja toodud järgnevas punktloendis.

- FN1. Rakendused on välimuselt sarnased.
    - Rakenduste päis, jalus ja menüüd jagavad sarnast välimust. Rakenduste taustavärv ja kasutatavad fondid on samad.
    - Kasutajaliides jälgib Google'i väljatöötatud Material disaini süsteemi. Material välimusele vastavate kasutajaliideste loomiseks kasutatakse Angular Materiali mooduli komponente. Komponendid on kasutajaliidese osad, mis täidavad mingit kindlat rolli. Angular Materiali komponendid on näiteks ühtse stiiliga nupud ja mitmesugused vormielemendid. Angular Materiali komponentide välimust kohandatakse ettevõtte vajadustelt lähtuvalt. Näiteks on muudetud komponentide värve, et rakenduse värvid vastaksid ettevõtte värvidele.

    - Lisaks Angular Materiali komponentidele on rakendustes kasutusel mitmed meeskonna enda loodud komponendid, näiteks komponent, mis kuvab veebilehe jäljerida ehk nn *breadcrumbs* komponent.

    - Rakenduste tunnusikoon ehk *favicon* on kõigil rakendustel samasugune.

- FN2. Rakendusi saab kasutada erinevates keeltes.

    - Kõik rakendused peavad olema inglise ja soome keeles. Osad rakendused vajavad ka rootsi keele tuge. Tõlgete lisamine peab olema lihtne, kuna tulevikus võib lubatud keelte arv olla suurem.

    - Sõnede tõlkimiseks kasutatakse ngx-translate teeki. Teek vajab tõlkimiseks seadistust, seadistada tuleb kasutatavate keelte nimekiri ja tõlkefailide asukoht. Teeki tuleb teavitada ka kasutatavast keelest ning olukorrast, kus kasutaja muudab kasutajaliidese keelt.

    - Komponent, kus kasutaja saab keelt valida, on erinevates rakendustes samasugune.

    - Valitud keel salvestatakse kasutaja seadmesse. See tähendab, et seade jätab meelde valitud kasutajaliidese keele ning kuvab veebilehe taasavamisel sõnesid kasutaja eelnevalt valitud keeles.

    - Rakenduste erinevate teenuste arendusel tuleb arvestada, et kuvatud tekst peab olema tõlgitav. See tähendab, et raamistiku ja väliste teekide pakutud teenuseid ja komponente kasutades tuleb luua vahekiht, mis sõnesid tõlgib. Näiteks tuleb lisada vahekiht, mis tõlgib kasutajale näidatavaid *pop-up* teavitusi.

- FN3. Rakenduste kasutamiseks on vaja kasutajal ennast tuvastada.

    - Kasutaja tuvastamiseks kasutatakse Keycloak teenust. Rakenduse käivitamisel tuleb teenuse käest küsida, kas kasutaja on tuvastatud. Kui kasutaja ei ole tuvastatud, suunatakse kasutaja sisselogimiseks mõeldud leheküljele.

    - Tagarakendus kasutab kasutaja tuvastamiseks igal päringul HTTP *Authorization* päist. See tähendab, et igale päringule lisatakse kliendi tuvastamiseks vajalik informatsioon.

- FN4. Kui sisseloginud kasutaja ei kasuta rakendust kindlal ajaperioodil, logitakse ta automaatselt välja. Enne kasutaja väljalogimist näidatakse kasutajale dialoogi, kus on võimalik väljalogimist edasi lükata.

- FN5. Rakendused kontrollivad, et rakenduse kasutajad on nõustunud kasutustingimustega. Kui tingimustega ei ole nõustutud, siis kuvatakse kasutajale tingimused. Platvormi ei lubata kasutada enne tingimustega nõustumist.

- FN6. Rakenduste seaded sõltuvad keskkonnast. Need on erinevad arendus-, test- ja toodangukeskkonnas. Rakendusest ei tehta nendesse keskkondadesse erinevaid koosteid. See tähendab, et on vaja leida viis, kuidas rakenduse seadeid muuta ilma uut koostet tegemata.

- FN7. Valminud veebirakendus tuleb pakendada jar failina. Selleks kasutatakse Spring raamistikku, mis on seadistatud tagastama päringute peale Angular rakendusest kooste kaustas olevaid faile.

## 3.2 Malliks sobiva baasrakenduse valik

Peale ühisosa leidmist tuleb analüüsida, kuidas on võimalik esitatud nõudeid kõige optimaalsemalt täita. Tuleb otsustada, kas võtta malli tegemise aluseks mõni olemasolev rakendus, millest tuleb üleliigne funktsionaalsus eemaldada, või võtta aluseks Angulari standardne tühi mall, millele tuleb soovitud funktsionaalsust lisada.

Üleliigse funktsionaalsuse kustutamine on lihtsam, kui nõutud funktsionaalsuse kopeerimine ja kohandamine. Funktsionaalsuse ja välimuse üle toomiseks tuleb valmisolevast tootest kood üles otsida, malli kopeerida ja vajadusel kohandada. Tihti on funktsionaalsusega seotud mitmeid erinevaid faile, mis teeb funktsionaalsuse üle toomise veelgi keerulisemaks. Seetõttu on malli arenduse aluseks mõistlik võtta olemasolev rakendus, mis täidab võimalikult palju mallile esitatud nõuetest.

Malli jaoks sobiva olemasoleva rakenduse valikuks koostas autor tabeli (Tabel 5), kus on välja toodud mallile esitatud nõuete olemasolu valmis rakendustes.

Tabel 5. Mallile esitatud nõuete olemasolu valmis rakendustes.

| **Nõue** | ***Purchase invoice portal* (PIP)** | ***Managment UI 2*** |
|---|---|---|
| FN1 | Rakenduse välimus järgib disainerite kavandit. Sarnased kavandid on tehtud ka teistele uutele rakendustele. | Rakenduse välimus erineb teistest rakendustest. |
| FN2 | Rakendus on tõlgitud inglise ja soome keelde. | Rakendus on tõlgitud inglise ja soome keelde. |
| FN3 | Rakenduse kasutamiseks on vajalik kasutaja tuvastamine. Rakendus kasutab selleks Keycloak teenust. Seda hakkavad kasutama ka tulevikus arendatud teenused. | Rakenduse kasutamiseks on vajalik kasutaja tuvastamine. Tuvastamine põhineb teistel tehnoloogiatel ning olemasolevat koodi tuleks oluliselt muuta. |
| FN4 | Funktsionaalsus on olemas. | Funktsionaalsus puudub. |
| FN5 | Funktsionaalsus on olemas. | Funktsionaalsus puudub. |
| FN6 | Rakenduse seadeid saab muuta vaid uusi koosteid tehes. | Rakenduse seaded on muudetavad sõltuvalt keskkonnast. Seadete muutmiseks ei ole vaja teha uut koostet. |
| FN7 | Rakenduse WAR faili ehitamine on seadistaud. | Rakenduse WAR faili ehitamine on seadistatud. |

Ülaltoodud tabeli (Tabel 5) põhjal selgub, et PIP projektis on täidetud kuus erinevat mallile esitatud nõuet ning täitmata on üks nõue. *Management UI 2* projekt vastab kolmele esitatud nõudele, täitmata on neli nõuet. Kuna PIP projekt vastab paremini mallile esitatud nõuetele, võetakse malli arendamisel aluseks PIP projekt.

## 3.3 Malli koostamine

Järgnevalt antakse ülevaade, kuidas koostati uute projektide mall, võttes aluseks olemasolev projekt.

Malli arendust alustati sellega, et eemaldati põhjaks võetud projektist kogu funktsionaalsus, mis on projekti spetsiifiline. Seejärel refaktoreeriti alles jäänud funktsionaalsust nii, et jagatud funktsionaalsust oleks võimalikult lihtne täiendada. Enne malli arendust arutati meeskonnaga põhjaks võetud projekti suuremaid puuduseid. Malli arendades pidas autor mainitud puudusi silmas ning eemaldas need.

Järgnevates alapeatükkides on välja toodud nii arendusele eelnenud arutluse käigus leitud probleemid kui ka malli põhja arenduse käigus tekkinud probleemid ning nende lahendamiseks kasutusele võetud lahendused.

### 3.3.1 Välimuse ühtlustamine

Projektis on märgata ebaühtset stiili, näiteks on rakenduste eri lehekülgedel nuppude välimus teistsugune.

Angulari komponentide stiili saab kohandada, kasutades CSS faile. CSS fail võib olla seotud konkreetse komponendiga, millisel juhul stiili kohandatakse kindlale komponendile, näiteks kasutajatingimustega nõustumise dialoogile. Samas on olemas ka komponentide vahel jagatud CSS fail, milles kirjeldatud reeglid kehtivad kõikidele rakenduste komponentidele. Projekt kasutab kohandatud Angular Materiali komponente, kohandatud on näiteks nuppude värvi. Sellised kohandused on aga tehtud üksikute komponentide CSS failis, mis toob kaasa mitmeid miinuseid. Esiteks sisaldavad CSS failid palju kordusi, teiseks on raske tulevikus stiili muuta, kuna muudatusi on vaja teha erinevate komponentide failides, kolmandaks võivad osad CSS failid jääda muudatuste tegemisel märkamatuks, mis tähendab, et stiil muutub ebaühtlaseks.

Probleemi lahendamiseks pidi malli jäävate komponentide CSS faile kontrollima, et leida, millised stiilikordused olid olemas. Stiilireeglid, mis kehtivad kõikidele komponentidele, viidi jagatud CSS faili. Muudatuse tulemusel kadusid komponentide stiili failidest koodikordused ning mallis olevate nuppude ja teiste kasutajaliideste elementide välimus on erinevatel lehekülgedel samasugune.

Peale stiilide koondamist ühte faili selgus, et suur hulk CSS selektoritest kohandavad erinevaid Angular Materiali komponente ettevõtte värvide kohaselt. Lahendus töötab, kuid autoril on raske veenduda, et kõik Angular Materiali komponendid on soovitud stiiliga. Stiile peab täiendama, kui tulevikus lisandub uusi komponente. Kui kasutajaliidese välimust soovitakse tulevikus muuta, tuleb muudatusi teha küll ühes failis, kuid väga paljudes erinevates CSS selektorites. Probleemile lahendust otsides selgub, et komponentide raamistiku dokumentatsioon soovitab stiili muutmiseks kasutusele võtta CSS eelprotsessor (*CSS preprocessor*) [7]. CSS eelprotsessorid genereerivad CSS faile, mis genereeritakse teiste failide põhjal. Need on kirjutatud mõnes CSSi täiendavas keeles [9]. Eelprotsessorit kasutades saab genereerida Angular Materiali komponentide stiili,

määrates kasutatavad värvid vaid ühes kohas. CSS eelprotsessor on Angulari raamistikku sisse ehitatud ja ei nõua seega projektile ühegi välise sõltuvuse lisamist. Arvestades, et muudatus aitab kaasa loetavusele, tagab ühtlase välimuse ning ei lisa projektile uusi sõltuvusi, võetakse CSS eelprotsessor projektis kasutusele.

### 3.3.2 Modulaarne rakendus

Malli arenduse põhjaks võetud projekt koosneb ühest rakenduse moodulist, kuhu on koondatud kogu rakenduse funktsionaalsus. Sellega kaasneb mitmeid probleeme.

Ühe mooduliga rakenduse puhul laetakse kogu rakendus kasutaja veebilehitsejasse korraga. See tähendab, et veebilehitseja laeb lisaks kuvatud lehele ka need lehed, mida ei ole vaja sessiooni jooksul kuvada. Kasutades erinevate vaadete jaoks eri mooduleid, on võimalik rakendust seadistada nii, et vaate jaoks vajalik kood laetakse alles vaatele navigeerides. Kogu rakenduse korraga laadimine tähendab, et rakenduse esimese vaate kuvamiseks kulub kauem aega. Korraga laetud rakenduse eelis on see, et rakendust kasutades on vaadete vahel navigeerimine kiirem, sest kõik rakenduse moodulid on veebilehitseja mälus. Moodulite vajadusepõhise laadimise korral peab uute vaadete kuvamiseks veebilehitseja laadima mooduleid serverist, mis tähendab lisapäringuid ning ooteaega.

Otsus, milliseid mooduleid laadida rakenduse esmasel avamisel ning milliseid mooduleid laadida jooksvalt, tuleb teha konkreetse rakenduse arendusel, mall ei piira vastava otsuse tegemist. Näiteks võib korraga laadida lehed, mida külastatakse kõige sagedamini.

Mallis seadistatud kolm vaadet laetakse siis, kui nendele navigeeritakse. Vaateid laetakse vajaduspõhiselt, sest antud vaateid külastatakse harva. Mallis defineeritud vaated on: komponente demonstreeriv vaade, mida ei kasutata valminud rakenduses; korduma kippuvate küsimuste vaade ja mitte leitud lehekülje vaade, mis kuvatakse, kui lehekülje aadressile ei vasta ükski lehekülg.

Malli kasutav rakendus ei pruugi vajada kogu malli loodud funktsionaalsust. Rakendusest on mõistlik üleliigne funktsionaalsus kustutada. Ebavajalik kood teeb rakenduse koostet suuremaks, võib vajada hooldust ning võib lisada rakendusele ebavajalikke sõltuvusi väliste teekide näol. Selleks et ebavajalikku funktsionaalsust oleks kerge kustutada, peaks iga funktsionaalsus olema grupeeritud ühte moodulisse, kausta või faili. Selline

grupeerimine aitab nii üleliigset koodi kustutada kui ka otsitavat koodi paremini leida. Malli põhjaks võetud projektis on erinevad klassid jagatud kaustadesse tüübi järgi, mis tähendab, et kõik teenused on teenuste kaustas, komponendid on komponentide kaustas jne. Kaustades on palju faile, mis ei ole omavahel seotud. Õiget faili on raske üles leida.

Malli põhjal loodud projektide navigeerimise ja ebavajaliku koodi kustutamise lihtsustamiseks jagatakse kood mallis funktsionaalsuse järgi erinevateks mooduliteks.

Koodi jaoks sobiva struktuuri leidmiseks moodustatakse funktsionaalsuse põhjal moodulid. Funktsionaalsuse põhjal moodustati järgnevad moodulid:

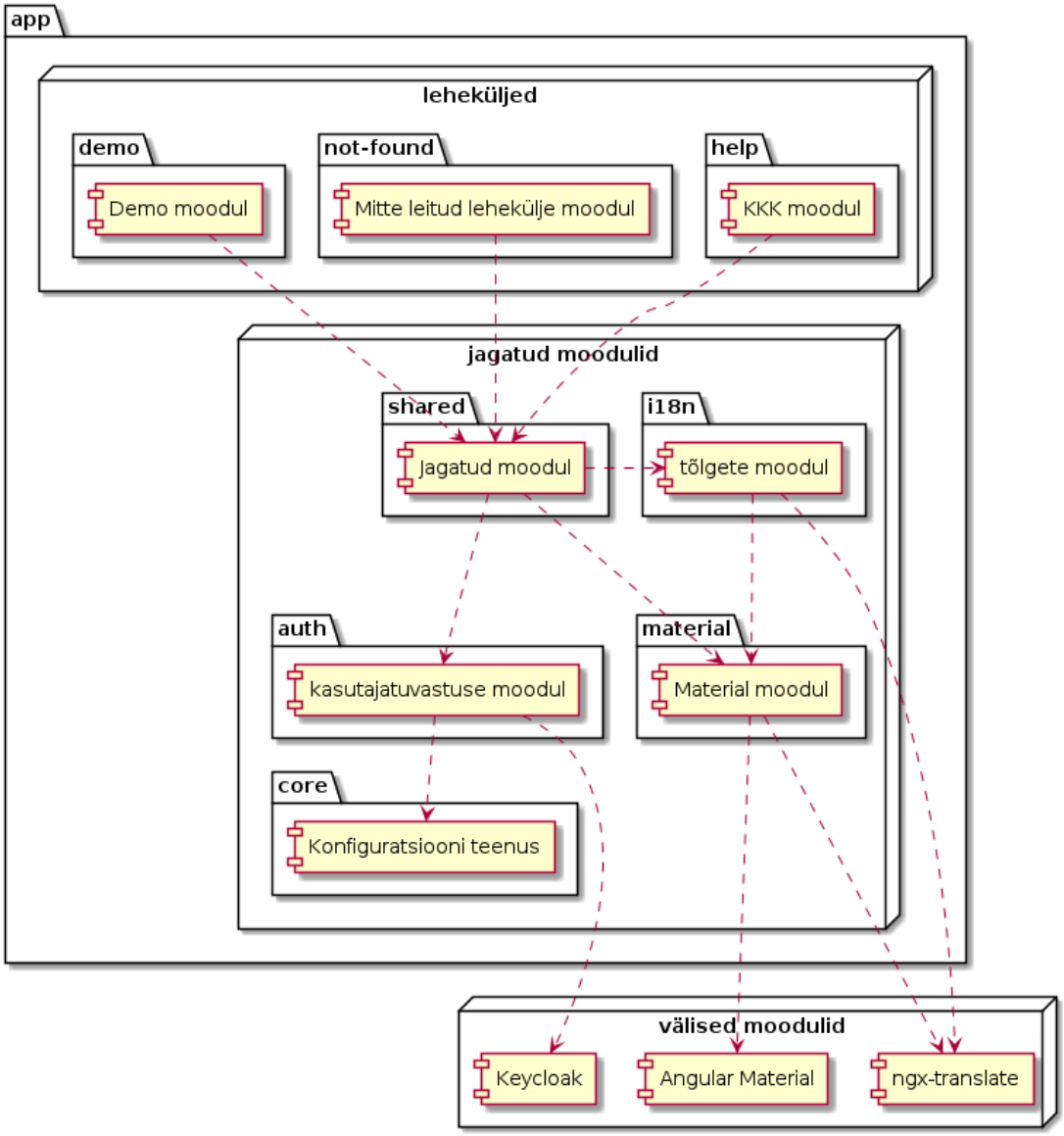
- kasutajaliidest tõlkiv moodul ehk *i18n* moodul;
- Angular Materiali seadistav *material* moodul;
- kasutajatuvastuse eest vastutav moodul ehk *auth* moodul;
- rakenduse konfiguratsiooni jagav moodul ehk core moodul;
- korduma kippuvate küsimuste lehekülje moodul ehk *help* moodul;
- mitte leitud lehekülje moodul ehk *not-found* moodul;
- demo moodul, mis näitab, kuidas erinevaid komponente kasutada, kustutatakse lõplikust rakendusest.

Lihtsam funktsionaalsus, mis koosneb ühest komponendist, jäeti jagatud funktsionaalsuse moodulisse ehk *shared* moodulisse. Selles moodulis on näiteks rakenduse jäljeriba (*breadcrumbs*) kuvav komponent.

Eri mooduleid seob rakenduse põhimoodul ehk *app* moodul. Põhimoodulit kasutatakse ka teiste seadistatavate moodulite seadistamiseks, näiteks seadistab põhimoodul, millised on rakenduses kasutatavad keeled.

Moodulid on jagatud kaheks grupiks: lehekülgede ja jagatud funktsionaalsuse moodulid. Lehekülgede moodulid on moodulid, mis sisaldavad loogikat kindla lehekülje või lehekülgede komplekti kuvamiseks, näiteks reklaamkampaaniate haldamise moodul. Jagatud funktsionaalsuse moodulites hoitakse korduvkasutatavaid komponente või rakenduse ülest funktsionaalsust, näiteks kasutaja tuvastuse loogikat.

Malli erinevate moodulite ja nendevaheliste sõltuvuste kujutamiseks on koostatud joonis. Joonisel olevad nooled on lisatud nii, noole teravik näitab moodulit, millest noole alguspunktis olev moodul sõltub. Lisaks mallis loodud moodulitele on joonisele lisatud välised moodulid, mida erinevad mallis olevad moodulid kasutavad. (Joonis 1)




Joonis 1. Mallis kasutatavad moodulid ja nende vahelised sõltuvused.

Malli põhjal luuakse projektid eelnevalt kirjeldatud Git *fork* meetodil. Sel viisil loodud projektide puhul on võimalik malli uuenemise järgselt projekti uuendused üle tuua. Sellise sünkroniseerimise võimaluse tagamiseks peavad aga mallis ja projektis olevate failide struktuur ja nendes sisalduv kood olema suhteliselt sarnased. Muul juhul leiab Git

malli ja projekti vahel liiga palju erinevusi ning Git ei saa automaatselt mallis tehtud muudatusi projekti üle kanda, vaid muudatused tuleb arendajal käsitsi sisse viia.

Sarnase struktuuri säilitamiseks rakendatakse arendusel järgnevad põhimõtted. Rakenduse uued leheküljed või lehekülgede komplektid lisatakse rakendusse alati uue moodulina. Uus funktsionaalsus, mis ei ole seotud ühe konkreetse leheküljega, vaid kogu rakenduse käitumisega, lisatakse rakendusse samuti moodulina. Väiksema jagatud funktsionaalsuse puhul lisatakse funktsionaalsus *shared* moodulisse, tehes sinna uus kaust koos lisatud funktsionaalsusega.

### 3.3.3 Andmete salvestamine

Malli põhjaks olev rakendus kasutab erineva informatsiooni salvestamiseks Redux teeki. Redux teek võimaldab andmeid salvestada ja pärida teatud mustri järgi. Meeskond on otsustanud, et teegi pakutud mustrid ei sobi kokku meeskonna vajadustega. Mustri ja seega ka teegi kasutamise üle otsustatakse iga projekti puhul eraldi, mis tähendab, et mallis ei tohi olla viiteid sellele teegile. Malli aluseks võetud projekt kasutab teeki näiteks sisselogitud kasutaja nime ja õiguste salvestamiseks. Samuti salvestati teeki kasutades kasutaja valitud keelt veebilehitseja püsimällu (*local storage*), et järgmisel korral saaks lehte kuvada kasutaja valitud keeles.

Redux teegi väljajätmise tõttu tekkis vajadus leida uus viis komponentide vahel jagatavate andmete hoidmiseks ja salvestamiseks.

Rakenduse ühe sessiooni jooksul salvestatavate andmete hoidmiseks soovitab ametlik dokumentatsioon kasutada singleton teenuseid [8]. Singleton on klass, millest luuakse programmi töötamise ajal maksimaalselt üks instants [11]. Singletoni väljadele saab salvestada jagatavaid andmeid, kuna kõik teenust kasutavad programmi osad viitavad samale teenuse instantsile, mis tähendab, et andmetest on vaid üks koopia. Angulari sõltuvuste süstimise süsteemi kasutades saab luua singleton teenuseid, lisades teenuse klassile vastava dekoraatori.

Singletoni klasse on vaja täiendada, et salvestada andmeid, mida on vaja säilitada mitmete sessioonide vahel, näiteks valitud keel. Salvestamaks andmeid erinevate sessioonide vahel on võimalik andmed salvestada veebilehitseja püsimällu (*local storage*) või küpsistena.

Andmete küpsistena salvestamisel peab arvestama mitmete piirangutega. Küpsise maksimaalne pikkus on 4096 baiti. Küpsise pikkuse mõõtmisel liidetakse kokku küpsise nime, väärtuse ja atribuutide pikkus. Küpsised lisatakse igale võrgupäringule, kui server ei kasuta küpsiste väärtusi, siis tähendab see ressursi raiskamist. [9]

Veebilehitseja püsimällu salvestamiseks on võimalik kasutada *window.localStorage* objekti. Objekti saab kasutada ühe domeeni piires andmete püsivaks salvestamiseks. Mällu saab salvestada võti-väärtus paare, seejuures peab arvestama, et nii võtmed kui ka väärtused peavad olema sõned. *LocalStorage* standard soovitab veebilehitsejatel lubada kuni 5 MB andmete püsimällu salvestamist domeeni kohta. Erinevalt küpsistest ei lisata püsimälus olevaid andmeid kõikidele veebipäringutele. [10]

Mõlemal viisil andmete salvestamisel tuleb arvestada, et veebilehitseja võib salvestatud andmed ootamatult kustuda. Andmed võidakse näiteks kustutada juhul, kui kasutaja soovib andmeid kustutada [9], [10]. Mõlemad standardid on toetatud kõikidel kaasaegsetel veebilehitsejatel [11], [12].

Mallis on vaja püsivalt salvestada kasutaja valitud keelt. Seda ei kasutata rakenduse serveris, seega kasutatakse valminud mallis valitud keele püsivalt salvestamiseks *localStorage* objekti.

### 3.3.4 Rakenduse konfiguratsiooni eraldamine koostest

Rakenduse seadeid on vaja muuta vastavalt sellele, millises keskkonnas rakendust käivitatakse. Rakendusel on erinevad seaded arendus-, test- ja toodangukeskkonnas. Näiteks on vaja muuta rakenduse Keycloak teenuse seadistust ja tagarakenduse aadressi.

Angular võimaldab rakenduse seadistuse muutmist, luues keskkondadele eri koosted. Kolme erineva kooste puhul on oht, et keskkonda laetakse üles teise keskkonna jaoks mõeldud kooste. Valminud koostest on väga raske üles leida kasutatavat konfiguratsiooni. Seetõttu on väga raske välja selgitada, millise keskkonna jaoks mõeldud kooste failidega on tegemist. Ettevõttes olevad protsessid uue tarkvara versiooni kasutuselevõtuks on loodud nii, et rakendusest luuakse üks kooste, mis laetakse eri keskkondadesse. Protsessid on automatiseeritud ning vajaksid muudatusi, et toetada keskkondades erinevate koostete kasutamist. Ettevõtte soovib, et igast rakenduse versioonist luuakse üks kooste ning konfiguratsioon oleks koostest sõltumatu.

Meeskonna teised rakendused on loodud Java programmeerimiskeelt kasutades. Rakendused kasutavad erinevaid Spring raamistiku lahendusi rakenduse konfiguratsiooniks. Rakendused kasutavad konfiguratsiooniks seadistuse faile. Failides on kirjeldatud rakenduse konfiguratsioon võti-väärtus paaridena. Faile võib olla mitu tükki ning failile määratakse tähtsus vastavalt faili nimele ja asukohale failipuus. Tähtsamaks hinnatud failis olevad seaded kirjutavad üle vähemtähtsas asukohas olevad seaded. Faili sisu loetakse rakenduse käivitamisel ning faile saab peale kooste tegemist muuta. [13]

Koostest sõltumatud välised seadistuse failid on kasutuses teistes meeskonna loodud rakendusetes. Selleks et malli põhjal valminud rakenduste konfiguratsioon oleks sarnane teiste rakendustega, kasutatakse ka mallis konfiguratsiooni hoidmiseks sarnast faili.

Rakenduse kooste pakitakse jar faili nii, et Spring raamistik tagastab päringute peale Angulari kooste artefakte, mis on oma olemuselt staatilised failid. Kuna rakendus kasutab Spring raamistikku, siis saab rakendusele lisada uusi veebipäringu otspunkte, mille sisu genereerib dünaamiliselt server.

Selleks et kliendi arvutis olev rakendus saaks seadeid küsida, loodi serveripoolsesse rakenduse osasse uus otspunkt, mis tagastab konfiguratsiooni failis seadistatud väärtustest need võti-väärtus paarid, mille nimi algab „public.“ liitega.

Kliendi poolel töötav rakendus peab enne rakenduse muude teenuste käivitamist alla laadima konfiguratsiooni, pärides seda kindlalt otspunkti aadressilt. Konfiguratsiooni alla laadimiseks loodi uus teenus, millel on olemas meetod, mis pärib konfiguratsiooni serverist. Meetod tagastab *promise* objekti ehk objekti, mis kirjeldab asünkroonset tegevust.

*Promise* objekt on proksi objekt ehk mingit teist objekti vahendav objekt [11]. Promise objekt võib vahendada objekti, mida ei ole promise’i loomise hetkel veel olemas. Vahendatav objekt võidakse luua hiljem. Promise objekt saab olla kolmes erinevas olekus:

- ootel, vahendatavat objekti pole veel loodud;
- täidetud, vahendatav objekti on edukalt loodud;

- tõrge, vahendatava objekti loomine ei õnnestunud. [14]

*Promise* objektil on meetod *then*. Meetodi sisendiks on funktsioon, mis käivitatakse, kui *promise* objekti staatus on muutunud täidetud olekusse. Täidetud olekus *promise* objekt käivitab funktsiooni koheselt. Funktsiooni käivitamisel antakse argumendiks *promise'i* vahendatav objekt. Tõrke staatuse tuvastamiseks on *promise* objektil meetod *catch,* mille argument on samuti funktsioon. Funktsioon käivitatakse, kui *promise* objekt on tõrke olekus. [14]

Angular raamistik võimaldab rakendust käivitada kahes etapis, mis võimaldab rakenduse seadistuse alla laadimist enne muu rakenduse käivitamist. Selleks saab seadistada funktsioone, mis käivitatakse enne rakenduse muude osade laadimist. Kui etteantud funktsioonid tagastavad *promise* objekte, siis ootab rakendus edasise laadimisega senikaua, kuni tagastatud objekt muutub täidetud olekusse. Raamistiku käivitatud funktsioonidel võivad olla erinevad parameetrid. Parameetrite tüüpideks võivad olla mitmed raamistiku või arendaja loodud teenuste klassid. Sõltuvuste süstimise raamistik loob vastavate klasside instantsid ning käivitab funktsioonid, kasutades argumentidena teenuste objekte. [15]

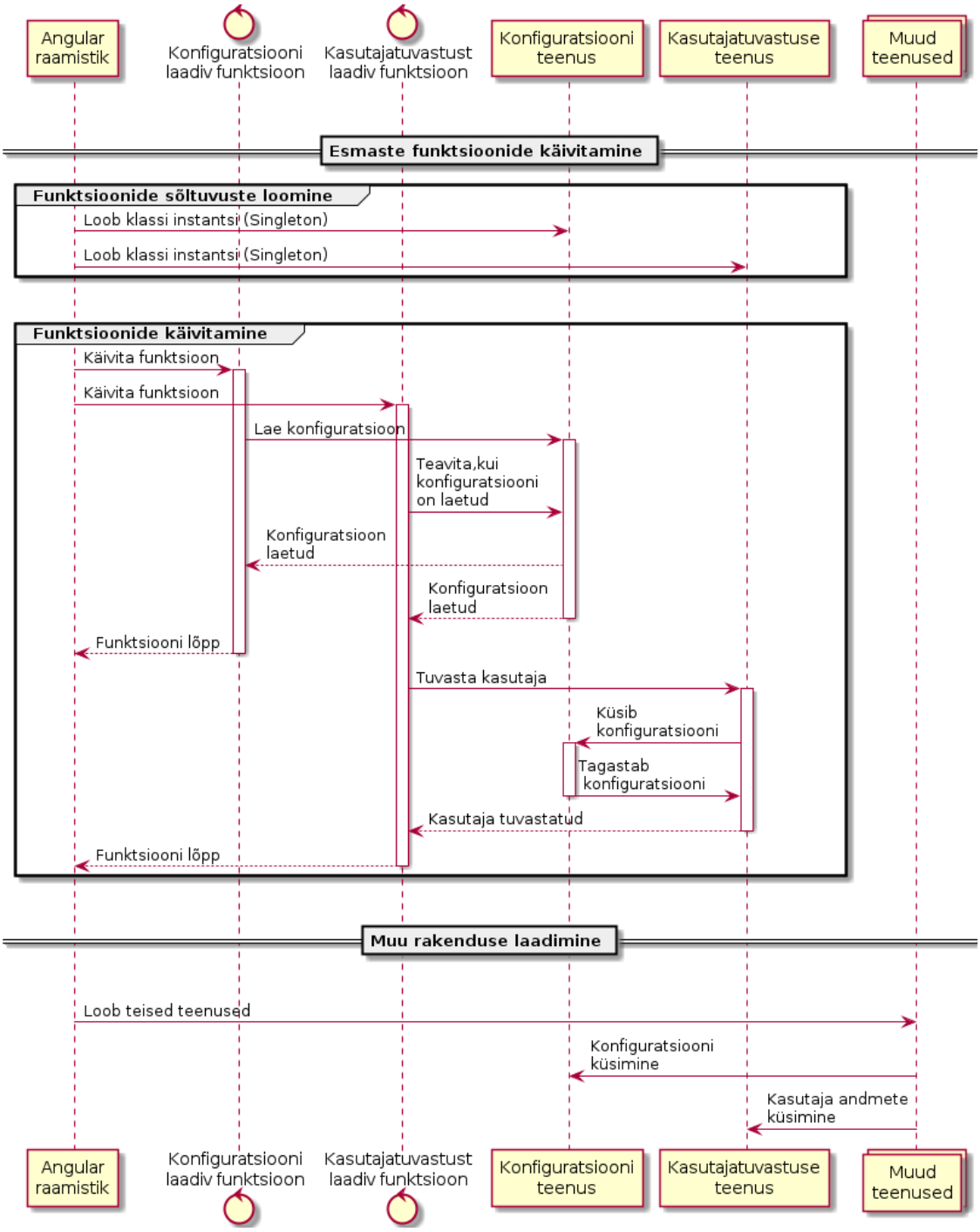
Mall peab rakendust laadima kolmes etapis. Esmalt tuleb laadida rakenduse konfiguratsioon. Teises etapis tuleb seadistada kasutajatuvastuse teenus. Seadistamiseks on vaja luuda Keycloak objekt, mis kontrollib, kas kasutaja on tuvastatud. Kui kasutaja ei ole tuvastatud, siis suunatakse kasutaja rakendusevälisele sisselogimise leheküljele. Keycloak objekti loomiseks on vaja teada rakenduse konfiguratsioonis olevaid parameetreid. Peale konfiguratsiooni laadimist ja kasutaja tuvastamist saab täita kolmanda etapi ehk laadida rakenduse kõik muud osad.

Kuna Angulari raamistik võimaldab rakenduse laadimist vaid kahes etapis, siis tuleb leida alternatiivne lahendus rakenduse laadimiseks mitmes etapis. Selleks jagatakse mallis rakenduse esmase laadimise etapp kaheks eri funktsiooniks. Esimene funktsioon alustab konfiguratsiooni alla laadimist ja tagastab *promise* objekti, mis muutub täidetud olekusse, kui konfiguratsioon on laetud. Teise funktsiooni eesmärk on seadistada kasutajatuvastuse teenus.

Konfiguratsiooni laadimiseks käivitab funktsioon konfiguratsiooni teenuse meetodi, mis laeb konfiguratsiooni tagarakendusest. Meetod tagastab *promise* objekti, mis muutub täidetud olekusse, kui konfiguratsioon on laetud.

Teine funktsioon seadistab kasutajatuvastuse teenuse. Teenuse seadistamiseks on vaja luua Keycloak objekt. Objekti loomise eeldus on rakenduse konfiguratsiooni olemasolu. Seega on kasutajatuvastuse teenuse seadistamise eeldus see, et konfiguratsiooni teenus on eelnevalt seadistatud. Konfiguratsiooni teenuse valmisoleku kontrolliks on konfiguratsiooni teenusele lisatud meetod, mis tagastab *promise* objekti, mis muutub täidetud olekusse, kui konfiguratsioon on laetud. *Promise* objekti *then* meetodit kasutades ootab teine funktsioon senikaua, kuni konfiguratsioon on laetud ning seadistab seejärel kasutajatuvastuse teenuse.

Rakenduse laadimise protsessi kujutamiseks koostati joonis. Joonisel kujutatakse, kuidas seadistatakse konfiguratsiooni ja kasutajaliidese teenused. Joonisel kujutatud pidevate joontega nooled märgivad sünkroonset suhtlust. Punktiirjoontega nooled tähendavad, et meetod või funktsioon tagastab *promise* objekti, mis saab oma väärtuse asünkroonselt. (Joonis 2)




Joonis 2. Rakenduse laadimise protsess.

## 3.4 Eraldiseiva jagatud mooduli loomine

Analüüsis toodi välja, et malli funktsionaalsust on võimalik edaspidi viia eraldi jagatavatesse moodulitesse. Väite kontrollimiseks viib autor peatükis eelnevalt kirjeldatud *shared* moodulis oleva teavituste komponendi eraldi jagatud mooduliks. Moodul pakendatakse npm pakiks ning võetakse mallis uuesti kasutusele, viidates paki gzip arhiivile viitavale aadressile.

Funktsionaalsuse eraldi pakki viimiseks tõstis autor esmalt funktsionaalsuse *shared* moodulist eraldi teavituse moodulisse *alert*. Eraldi pakitav moodul peab olema mallis loodud moodulitest sõltumatu, mis tähendab, et moodul ei saa kasutada mallis loodud funktsionaalsust. Teavituste komponent oli sõltumatu ning seega oli eraldi pakendamiseks sobilik. Kui eraldi pakendatav moodul oleks olnud teistest mallis loodud moodulitest sõltuv, siis peaks leidma võimalusi sõltuvuste eemaldamiseks.

Projektide vahel jagatava mooduli loomiseks on vaja luua uus Angulari projekti kaust. Kausta on omakorda vaja luua jagatava mooduli kaust, mis erineb mõnevõrra üksikus projektis kasutatavast moodulist. Näiteks on jagatavas moodulis fail, kus on kirjas mooduli nimi, versioon, kirjeldus ja autor. Kasutades Angulari raamistikuga kaasnevat koodigeneraatorit, saadi uue projekti ja jagatava mooduli jaoks vajalikud failid käsureal genereerida. Peale koodi genereerimist kopeeriti mallis olevast *alert* moodulist kood uude projekti.

Valminud mooduli jagatavaks tegemiseks ehitas autor sellest kooste. Selle artefaktid pakiti gzip arhiivi, mis laaditi üles ettevõtte veebiserverisse. Seejärel eemaldati mallis olev teavituste moodul ning lisati mallile pakendatud teavituste moodul, viidates paki arhiivi aadressile.

Malli erinevaid mooduleid saab viia vajadusel eraldi pakkidesse. Eraldi pakendamise eelduseks on see, et pakendatava mooduli kood ei tohi sõltuda teistest malli moodulitest.

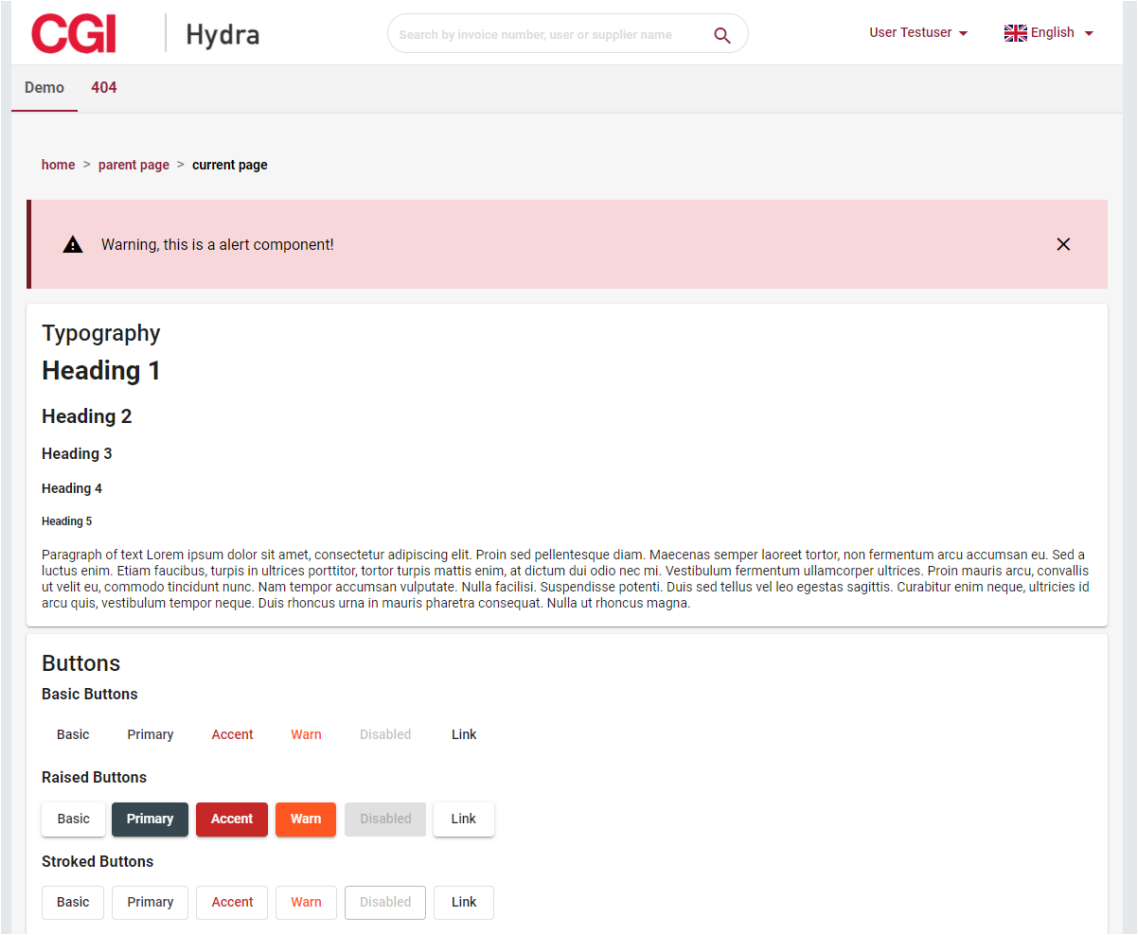
# 4 Tulemused

Peatükk hindab, kas teises peatükis valitud lahendusmeetod ja kolmandas peatükis väljatöötatud lahendus täidavad lahendusele esitatud nõudeid ja lahendavad töös esitatud probleeme. Peatükis on toodud ka näited malli põhjal loodud kasutajaliidesest. Lisaks võetakse kokku kolleegide hinnang lahendusele ning kirjutatakse projekti tulevikust.

## 4.1 Lahendus

Loodud lahendus täidab kolme probleemi põhjal koostatud nõuet. Uute projektide arendusel hoitakse kokku aega, kuna mallis on täidetud mitmed nõuded, mis kehtivad kõikidele ettevõttes arendatavatele projektidele. Projektiga alustades ei ole vaja seadistada erinevaid teeke, näiteks tõlgete või kasutajatuvastuse teeki. Samuti on eelnevalt kujundatud ja loodud põhiliste kasutajaliidese komponentide stiil, näiteks on disainitud nupud, tabelid, menüüd, päis ja jalus. Tänu eelnevalt loodud komponentidele aitab lahendus tagada ühtset välimust erinevatel ettevõtte veebilehtedel. Samas on lihtne komponente projektis muuta, kui selleks peaks vajadus tekkima. Malli kasutamiseks valitud Git *fork* meetodi abil saab projektide välimust kergesti ühtselt uuendada, selleks esmalt malli ning seejärel Git merge käsu abil mallil baseeruvaid rakendusi uuendades.

Mallis on olemas ka erinevaid komponente demonstreeriv lehekülg. Leheküljel on kasutatud malliga kaasa tulevaid kasutajaliidese komponente. Lehekülje lähtekoodi saab kasutada näidisena uute lehekülgede loomisel. Tööle on lisatud näidislehekülje ekraanitõmmis (Joonis 3), kus on näha jagatud komponente, sealhulgas päist, keele valimise menüüd, navigeerimise menüüd, jäljeriba (*breadcrumbs*), teavituse (*alert*) komponenti ja erinevate nuppude näidiseid.




Joonis 3. Malli funktsionaalsust demonstreeriv näidislehekülg.

## 4.2 Tagasiside

Valminud lahendus esitati tagasiside saamiseks teisele meeskonna arendajale. Arendaja veendus, et projektile esitatud nõuded on täitetud. Arendajale meeldis pakutud lahenduse lihtsus. Ta leidis, et lahendust on lihtne kasutusele võtta ning see hoiab kokku uute projektide arendusele kuluvat aega. Tagasisides toodi välja, et mallis olev kood on hästi struktureeritud, malli eri mooduliteks jagamine lihtsustab koodis navigeerimist. Tagasisides paluti, et malli kasutamise kohta koostataks juhised, täpsemalt sooviti juhiseid uue projekti loomiseks ja malli *fork* meetodil uuendamise protsessi kohta. Tagasiside põhjal koostas autor soovitud juhendid.

Lahendus plaanitakse kasutusele võtta vähemalt kahes uues projektis. See võimaldab erinevate väiksemate projektide kiiret arendust, mistõttu soovitakse malli kasutada ka ettevõttesiseste kasutajaliideste arenduseks.

# 5 Kokkuvõte

Töös leiti lahendused sarnaste kasutajaliideste arendusega seotud probleemidele. Põhiprobleem on rakenduste perekonna ühtse välimuse tagamine. Varasemalt saavutati jagatud välimus ja funktsionaalsuskoodi kopeerimisega ühest projektist teise, mis oli ajamahukas töö. Lisaks pidi jagatud välimusele või funktsionaalsusele esitatud nõuete muutumisel tegema käsitsi muudatusi kõikides projektides.

Võimalikule lahendusele esitati erinevad nõuded. Lahendus peab aitama tagada ühtset välimust ja funktsionaalsust, vähendama uute projektide arenduseks kuluvat aega ning nõuete muutumisel peab lahendus võimaldama kerget projektide uuendamist. Lisaks üldistele nõuetele esitati ka konkreetsele arendusele kehtivad piirangud.

Töös analüüsiti erinevaid võimalikke lahendusi. Üheks võimaluseks oli ühisosa jagamine jagatavatesse moodulitesse, mida erinevad projektid saavad kasutada. Teise lahendusena uuriti uue projekti malli koostamist, milles on projektide ühisosa eelnevalt implementeeritud. Sobiva lahenduse valimiseks koondati mõlema lahendusviisi plussid ja miinused võrdlustabelisse.

Analüüsi tulemusel valiti sobivaks lahenduseks malli meetod. See valiti sellepärast, kuna meetod sobitus kõige paremini arendusele ettevõtte seatud piirangutega, lahenduse väljatöötamiseks kulub võrreldes alternatiividega vähem aega ning meetod annab ajalist võitu ka väheste malli kasutavate projektide puhul. Lahendust valides tuleb hinnata projektile kehtivaid piiranguid, kuivõrd teiste projektide puhul võib olla parimaks lahenduseks moodulite meetod. Seda näiteks juhul, kui jagatud funktsionaalsust kasutavad paljud erinevad rakendused.

Töös kirjeldati konkreetse ettevõtte põhjal malli loomist. Malli loomiseks leiti projektide ühisosa, täpsustati mallile esitatud funktsionaalseid nõudeid ja leiti malli arenduseks sobiv baasprojekt. Töös kavandati malli arhitektuuri, kirjeldati malli arendusel tekkinud probleeme ning autori kasutusele võetud lahendusi.

Valminud mall lahendab töös esitatud probleemid. See aitab kokku hoida uute projektide arenduseks kuluvat aega. Valminud lahenduse on autori kolleegid heaks kiitnud ning lahendust rakendatakse ettevõtte uute projektide arenduses.

# Kasutatud kirjandus


[1] A. Altaboli and M. R. Abou-Zeid, "Effect of Physical Consistency of Web Interface Design on Users' Performance and Satisfaction.," *Jacko J.A. (eds) Human-Computer Interaction. HCI Applications and Services. HCI 2007. Lecture Notes in Computer Science.,* vol. 4553, pp. 849-858, 2007.

[2] J. Nielsen, „Coordinating User Interfaces for Consistency," *SIGCHI Bulletin,* p. 63–65, 1989.

[3] Google LLC, „Angular, Overview of Angular libraries," [Võrgumaterjal]. Kättesaadav: https://angular.io/guide/libraries. [Kasutatud 30.03.2020].

[4] npm, Inc., „Npm Documentation," [Võrgumaterjal]. Kättesaadav: https://docs.npmjs.com/. [Kasutatud 19.04.2020].

[5] T. Preston-Werner, „Semver 2.0.0," 16.07.2013. [Võrgumaterjal]. Kättesaadav: https://semver.org/. [Kasutatud 19.04.2020].

[6] Riigi infosüsteemi amet, „Versioneerimine, Riigi infosüsteemi haldussüsteem," [Võrgumaterjal]. Kättesaadav: https://e-gov.github.io/RIHA-Index/Versioneerimine [Kasutatud 11.05.2020].

[7] Verdaccio, „What is Verdaccio?," [Võrgumaterjal]. Kättesaadav: https://verdaccio.org/docs/en/what-is-verdaccio. [Kasutatud 24.04.2020].

[8] Google LLC, „Angular, Creating Libraries," [Võrgumaterjal]. Kättesaadav: https://angular.io/guide/creating-libraries. [Kasutatud 12.05.2020].

[9] P. Hammant, „Trunk Based Development: Monorepos," [Võrgumaterjal]. Kättesaadav: https://trunkbaseddevelopment.com/monorepos/. [Kasutatud 20.04.2020].

[10] Atlassian Corporation Plc, „Forking Workflow," [Võrgumaterjal]. Kättesaadav: https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow. [Kasutatud 11.05.2020].

[11] Google LLC, „Theming your Angular Material app," [Võrgumaterjal]. Kättesaadav: https://material.angular.io/guide/theming. [Kasutatud 19.04.2020].

[12] Mozilla Foundation, „MDN Web Docs, CSS preprocessor," [Võrgumaterjal]. Kättesaadav: https://developer.mozilla.org/en-US/docs/Glossary/CSS_preprocessor. [Kasutatud 24.04.2020].

[13] Google LLC, „Angular, Coding Style Guide," [Võrgumaterjal]. Kättesaadav: https://angular.io/guide/styleguide. [Kasutatud 20.04.2020].

[14] E. Gamma, R. Helm, R. Johnson ja J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Boston, MA, USA: Addison-Wesley, 1994.

[15] A. Barth, „HTTP State Management Mechanism," RFC Editor, 04.2011. [Võrgumaterjal]. Kättesaadav: https://tools.ietf.org/html/rfc6265. [Kasutatud 22.04.2020].

[16] Web Hypertext Application Technology Working Group, „HTML Living Standard, Web storage,“ 22.04.2020. [Võrgumaterjal]. Kättesaadav: https://html.spec.whatwg.org/multipage/webstorage.html. [Kasutatud 22.04.2020].

[17] Mozilla Foundation, „MDN web docs, Window.localStorage,“ [Võrgumaterjal]. Kättesaadav: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage. [Kasutatud 22.04.2020].

[18] Mozilla Foundation, „MDN web docs, Cookie,“ [Võrgumaterjal]. Kättesaadav: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cookie. [Kasutatud 22.04.2020].

[19] P. Webb, D. Syer ja J. Long, „Spring Boot Features, Application Property Files,“ [Võrgumaterjal]. Kättesaadav: https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-external-config-application-property-files. [Kasutatud 22.04.2020].

[20] Mozilla Foundation, „MDN web docs, Promise,“ [Võrgumaterjal]. Kättesaadav: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. [Kasutatud 22.04.2020].

[21] Google LLC, „Angular, Dependency Injection Providers,“ [Võrgumaterjal]. Kättesaadav: https://angular.io/guide/dependency-injection-providers. [Kasutatud 22.04.2020].