TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Juri Geiman

Veebikoosolekute haldusfunktsionaalsuse refaktoreerimine AS-i Fujitsu Estonia näitel

Bakalaureusetöö

Juhendaja: Kristiina Hakk
PhD
Kaasjuhendaja: Dmitri Lepp
PhD

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: Juri Geiman

15.05.2023

Annotatsioon

Lähtekoodi parandamine ja refaktoreerimine tehnilise võla vähendamiseks on suurte ettevõtete nagu Fujitsu Estonia AS jaoks oluline teema. Käesoleva töö eesmärk on refaktoreerida CaseM-i projekti veebikoosolekute haldusfunktsionaalsust. Antud funktsionaalsus loodi 2014. aastal ja see sisaldab mitmeid arhitektuuriprobleeme, mis raskendavad refaktoreerimise protsessi.

Töö analüütilises osas põhjendati funktsionaalsuse refaktoreerimise aktuaalsust ja kasulikkust, tuvastati olemasoleva lahenduse probleeme ja uuriti nende lahendusviise; sõnastati nõuded refaktoreeritud lahendusele ning valiti kasutatud tehnoloogiad ja lahenduse arhitektuur, koostati refaktoreerimise plaan. Töö praktilises osas arendati veebikoosolekute haldusfunktsionaalsuse mooduli põhi analüütilise osa käigus valitud arhitektuuri ja tehnoloogiate kasutamisega; täideti refaktorerimisplaan lõputöös püstitatud mahus. Töö tulemusena veebikoosolekute haldusfunktsionaalsuse refaktoreerimise vajadus oli põhjendatud ja töö raames plaanitud funktsionaalsuse refaktoreerimine läbi viidud.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 37 leheküljel, 7 peatükki, 9 joonist, 2 tabelit.

Abstract

Meetings Management Functionality Refactoring Based on the Example of Fujitsu Estonia AS

Improving and refactoring source code to reduce technical debt is a crucial topic for large companies like Fujitsu Estonia AS. The focus of this work was to refactor the meetings management functionality in the CaseM project. Given functionality was developed in 2014 and contains several architectural problems, which provides additional challenge for refactoring process.

In the analysis part of the thesis the need for refactoring was demonstrated, and the benefits of refactoring were presented. Then the functionality source code was analysed, and its problems were investigated along with potential solutions. Possible architectures for refactored solution were compared resulting in microservice approach being chosen. Suitable technologies stack was selected. The plan for conducting refactoring process was implemented.

In the practical part of the bachelor's thesis, the author was fulfilling the implemented refactoring plan in scopes of given thesis, solving appearing problems and obstacles in the process. As a result of the work, new microservice module was implemented and integrated into CaseM project and refactoring of the selected part of meeting management functionality was completed.

The thesis is in Estonian and contains 37 pages of text, 7 chapters, 9 figures, 2 tables.

Lühendite ja mõistete sõnastik

ACL	*Access Control List,* pääsupiiramisloend, millega määratletakse, kellel on õigused objektidele ning millised need on
atomic operation	aatomiline toiming — jagamatu toiming, mida viiakse läbi tervikuna. Ekslikul juhul ühte muudatust ei tehta.
AOP	*Aspect-Oriented Programming*, programmeerimisparadigma, mille eesmärk on suurendada programmi moodulipõhisust, lisades käitumist olemasolevale koodile ilma koodi ennast muutmata, vaid määrates eraldi, milline kood muudetakse „*pointcut*“ spetsifikatsiooni kaudu.
API	*Application Programming Interface*, rakendusliides ehk programmiliides
CMS	*Content Management System,* informatsioonisüsteem või arvutiprogramm, mida kasutatakse sisu loomise, toimetamise ja haldamise koostööprotsessi lihtsustamiseks ja korraldamiseks
DAO	*Data Access Object*, andmebaasi juurdepääsu objekt
Documentum	OpenText EMC Documentum, ettevõte sisuhalduse platvorm
DQL	*Documentum Query Language*, spetsialiseeritud SQL-laadne päringukeel Documentum-i pärimiseks.
ECM	*Enterprise Content Management*, ettevõte sisuhalduse süsteem, ehk ettevõte CMS
EJB	*Enterprise JavaBeans* (või *Jakarta Enterprise Beans*), serveripoolne tarkvara komponentide arhitektuur hajusate Java rakenduste ehitamiseks
Java	SUN Microsystems-i 1995. a. loodud klassi- ja objektipõhine platvormist sõltumatu universaalne programmeerimiskeel
JSON	*Javascript Object Notation,* JavaScript-il põhinev andmevahetusvorming
JVM	*Java Virtual Machine,* on abstraktne masin, mis interpreteerib kompileeritud Java baitkoodi arvuti protsessorile arusaadavateks instruktsioonideks
REST	*Representational State Transfer*, on tarkvaraarhitektuuri laad, mis kirjeldab ühtset liidest füüsiliselt eraldiseisvate komponentide (tihti tagarakenduse ja kasutajaliidese paar) vahel
REST-ful API	Programmiliides mis jälgib REST printsiipe

SOAP	*Simple Object Access Protocol*, on arvutivõrkudes kasutatav protokoll, millega veebiteenused vahetavad omavahel struktuurseid (XML formaadis) andmeid
Tehniline võlg	Tarkvara arenduse puhul on „laen“, mis võimaldab koheselt midagi saavutada arvestades, et sellega võivad olla seotud teatud probleeme või riske süsteemi edasiarendamisel [1]
Vaadin	Vaadin 8 Framework, Java põhinev veebirakenduste kirjutamise raamistik, sisemiselt kasutab Google Web Toolkit-i raamistiku Java koodi tõlkimiseks ja kompileerimiseks HTML ja JavaScript koodiks
XML	*Extensible Markup Language*, formaat andmete esitamiseks, salvestamiseks, andmevahetuseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Pärandkoodi kogunemine ning tehnilise võla kasvamine on pikema arendusega tarkvaraprojektides loomulik protsess. Tehnilise võla otsused rakenduvad nii mikrotasandil, kus võime valida uue funktsiooni lisamise keerulise tingimusloogika abil, kui ka makrotasandil, kus teeme arhitektuurilisi kompromisse, et süsteem saaks läbida veel ühe versiooni [2]. Tehnilise võla olemasolu tunnistamine, leides viise, kuidas seda järk-järgult lahendada, on oluline tarkvara hallatavuse säilitamiseks [3].

Käesolevevas lõputöös uuritakse pärandkoodi esinemisest tulenevaid probleeme AS Fujitsu Estonia arenduses oleva projekti CaseM näitel ja kirjeldatakse selle projekti ühe mooduli — veebikoosolekute haldusfunktsionaalsuse refaktoreerimise protsessi. Tarkvara koodi parandamise probleem on aktuaalne AS-i Fujitsu Estonia ettevõttele, ning on oluline osa arendusprotsessist.

Lõputöö eesmärgid on veebikoosolekute haldusfunktsionaalsuse pärandkoodi analüüsimine ja sellest lahti saamine ning iseseisva teenusmooduli arendamine vastavale funktsionaalsusele, kasutades kaasaegseid tehnoloogiaid ning asjakohaseid programmikoodi disainimismustreid.

Lõputöö teoreetilises osas tutvustatakse lühidalt refaktoreerimise mõisteid ning kirjeldatakse rakendust lõputöö skoobis. Analüütilises osas tuvastatakse olemasoleva funktsionaalsuse teostuse probleeme ja uuritakse meetmeid leitud probleemide parandamiseks. Kavandatakse antud funktsionaalsuse refaktoreerimise plaan.

Lõputöö praktilises osas arendatakse vastavalt seatud nõuetele ja tehnoloogiavalikutele veebikoosolekute haldusmooduli põhi, refaktoreeritakse veebikoosolekute dokumentide genereerimise funktsionaalsus, mis hõlmab koodi struktuuri parandamist, päringute optimeerimist ja funktsionaalsuse testidega katmist.

2 Taust

Käesoleva peatüki eesmärk on defineerida ning anda lühike ülevaade antud töö jaoks olulistest mõistetest. Probleemide põhjalikumaks tutvustamiseks antakse ülevaade ettevõttest ja projektist, mille jaoks antud töö teostatakse ning kirjeldatakse käesoleva tööga seotud funktsionaalsust.

2.1 Ettevõte taust

Fujitsu on juhtiv Jaapani päritoluga infotehnoloogia ja kommunikatsiooni teenuseid pakkuv firma. Ettevõttes tegutseb ligi 130 000 töötajat, kes toetavad kliente enam kui 180 riigis. AS Fujitsu Estonia on Fujitsu esindus Eestis, mis on tegutsenud alates 1991. aastast nii tarkvaraarenduse kui ka arvutisüsteemide valdkonnas. Fujitsu Estonia kliendid asuvad peamiselt Soomes ja teistes Põhjamaades ning nende seas on nii suuri kaubanduskette ja pankasid, kui ka tootmisettevõtteid. Eestis ettevõte teeb koostööd avaliku sektori organisatsioonidega. Ettevõttes Fujitsu Estonia tegutseb enam kui 400 töötajat. [4]

2.1.1 Projekti kirjeldus

Projekt, millega on seotud lõputöö, on juhtumi- ja dokumendihalduse rakendus CaseM, mida arendab AS Fujitsu Estonia Dokumentumi meeskond koostöös Fujitsu Finland Oy kolleegidega. CaseM on teabehalduslahendus, mis lisaks juhtumi- ja dokumendihaldusele võimaldab korraldada veebikoosolekuid ja koostööd, avaldada ja allkirjastada dokumente, teostada protsesside ja liigitusreeglite juhtimist ning jälgida protsesside kulgemist reaalajas. [5]

Rakenduse kasutajaliides on tehtud Vaadin 8 raamistikul, tagarakendus kasutab Documentum süsteemi andmete haldamiseks. EMC Documentum on dokumendihaldustarkvara, mis hõlmab endas dokumentide metaandmete, versioonide ja binaarse kuju hoidmist ning võimaldab antud andmete juurde ligipääsu teiste süsteemikasutajate poolt, Dokumentum tagab ka teist funktsionaalsust nagu plaanilised (tausta) tööd (*scheduled job*) [6].

Rakendust kasutavad kümned erinevad organisatsioonid. Organisatsioonide vajaduste rahuldamiseks on rakenduse seadistamine teostatud läbi XML (*Extensible Markup Language*) konfiguratsioonifailide. Antud XML failid võimaldavad seadistada tagarakenduse ja kasutajaliidese käitumist programmikoodi muutmata.

2.1.2 Veebikoosolekute funktsionaalsus

Üks osa CaseM-i projekti funktsionaalsusest on veebikoosolekute haldamine ja läbiviimine. Koosolekute haldus on tehtud läbi kasutajaliidese, mille arendamise eest vastutab Dokumentumi meeskond. Koosolekute läbiviimise keskkond on tehtud SharePoint-i põhjal ja selle arendamise eest vastutab teine meeskond.

Koosoleku olem koosneb arutlusteemades (inglisekeelne nimi projektis on *meeting topic*), mida omakorda iseloomustab sellega seotud põhidokument ning manused (Joonis 1). Teemade põhidokumendid rakendavad lepitud struktuuri (inglisekeelne nimi projektis on *structured document*), mis võimaldab neid toimetada läbi kasutajaliidese ning nende (ning ka koosoleku metaandmete) põhjal genereerida koosolekuga seotud PDF dokumente, nagu päevakorrad ja protokollid.

Joonis 1. Veebikoosoleku kuvatõmmis.

Koosoleku liikmed jagunevad esimeheks (ja tema asendajateks), sekretäriks ja tavaliikmeteks, igal rolli jaoks on oma kasutajaõigused, mis rakenduvad koosolekuga seotud objektidele.

Koosoleku andmeid ja seotud dokumente saab publitseerida, mille eest vastutab publitseerimise töö, mis on osa CaseM taustatööde moodulist (inglisekeelsest nimest *CaseM Job*) ja mis käivitub Dokumentum-i plaanilise tööna.

Veebikoosolekute funktsionaalsus on seadistatav projekti osa ja võetud kasutusele mitmetes organisatsioonides.

2.2 Refaktoreerimine ja optimeerimine

Refaktoreerimine on olemasoleva tarkvara disaini parendamise protsess, mille käigus ei muudeta selle funktsionaalsust. Eesmärk on muuta tarkvara hooldatavamaks, vahetades koodi sisemist struktuuri, kuid hoides välise käitumise vahetatuna. Refaktoreerimine hõlmab väiksemate struktuursete muudatuste tegemist, mis on toetatud testidega, et tagada koodi funktsionaalsuse säilitamist refaktoreerimise protsessi jooksul. Refaktoreerimise käigus teostatud struktuursed muudatused võivad mõjutada jõudlust. [7], [8]

Koodi refaktoreerimiseks on kirjeldatud mitu kasulikku tehnikat, nagu meetodi ekstraheerimine (*Extract Method*), parameetri objekt (*Parameter Object*), või tsüklite jagamine (*Split Loop*). Enamus tehnikatest on universaalsed ehk ei sõltu programmeerimise keelest või kasutatud raamistikkudest. Põhjalik ülevaade refaktoreerimise tehnikatest on võimalik saada Martin Fowler'i raamatust „Refactoring: Improving the Design of Existing Code". [8]

Koodi optimeerimise käigus saavutatakse mingi ressurssi kasutamise paranemine. Enamasti on selleks ressursiks aeg või mälu, ehk programmi tehakse kiiremaks või vähendatakse mälu kasutamist. Sarnaselt refaktoreerimisega koodi optimeerimisel funktsionaalsust ei muudeta. Programmi optimeerimisel ei parandata koodi struktuuri. [7]

2.3 Eesmärk ja metoodika

Käesoleva bakalaureusetöö eesmärk on veebikoosolekute funktsionaalsuse refaktoreerimise vajalikkuse põhjendamine, refaktoreerimise plaani kavandamine ja läbiviimine. Töö raames analüüsitakse kogu koosolekute haldusfunktsionaalsust, kuid praktilises osas refaktoreeritakse teatud osa sellest seoses funktsionaalsuse mahukusega. Refaktoreeritud lahendus peab olema esitatud eraldi projekti moodulina, mida saab kasutada ka teistes projekti moodulites, et saada lahti koodi kordustest ja surnud koodist, ning sellega tõsta koodi kvaliteeti. On oluline katta refaktoreeritud koodi testidega ning võimalusel optimeerida programmi osad.

Lõputöö aktuaalsus on tingitud sellest, et veebikoosolekute funktsionaalsus on oluline osa projektist, mis on kasutusel mitmetel klientidel. Lõputöö eesmärkide saavutamine ehk antud funktsionaalsuse hallatavuse parandamine lihtsustab funktsionaalsuse edasiarendamise protsessi ning teeb CaseM-i rakendust atraktiivsemaks klientidele.

Lõputöö eesmärkide saavutamiseks analüüsitakse olemasolevat programmikoodi ja tuuakse välja selle puudused. Kasutades refaktoreerimise meetodeid parandatakse koodi loetavust, keerukust, hallatavust ja laiendatavust. Luuakse veebikoosolekute haldusmoodul ja kaetakse kood testidega täitmise korrektsuse veendumiseks.

3 Probleemi analüüs

Veebikoosolekute funktsionaalsuse põhi arendati 2014. aastal ja sellest ajast on ilmnenud mitu teostusega seotud probleemi. Käesolevas peatükis on toodud eelnevalt mainitud probleemid ning ka põhjendatud nende lahendamise vajalikkust.

3.1 Põhjused refaktoreerimise vajalikkusele

Erinevates projekti arendamise etappides on veebikoosolekute haldusfunktsionaalsusesse sisse viidud hulk muudatusi, täiustusi ning parandusi. Käesoleva töö kirjutamise ajal funktsionaalsust ei muudeta, kuid see nõuab pidevat tuge ja kasutamisel leitud probleemide parandamist. Probleeme leitakse peamiselt projekti põhiversioonis, mille peale oli migreeritud vaid üks funktsionaalsust kasutavatest klientidest. Teised funktsionaalsust kasutavad kliendid kasutavad töö kirjutamise ajal projekti isiklikke versioone, mis on suures osas vananenud võrreldes põhiversiooniga. On oodata ka nende klientide migreerimist projekti põhiversioonile ja sellest tulenevad probleemid, mis on seotud veebikoosolekute haldusfunktsionaalsusega. Lähtuvalt sellest, funktsionaalsuse refaktoreerimine on kasulik, tehes funktsionaalsuse hooldatavamaks ja lihtsustades kliendi migreerimist pärast osutatud probleemide lahendamist.

Funktsionaalsuskoodi uuendamise lisaargumendiks on see, et selle arendusega on tegelenud peamiselt inimesed, kes käesolevas projektis enam ei tööta. Probleemide lahendamine nõuab lisaaega koodiga tutvumisel, kuna puudub võimalus abi saamiseks pädevatelt kolleegidelt.

Refaktoreerimise vajalikkuse tõestamiseks saab kasutada Adam Tornhill'i poolt pakutud kuumpunkte (*hotspot*), mille kaudu saab tuvastada programmi osasid suurema tehnilise võlaga ning määrata priortiteete nende parandamiseks. Selle meetodi kasutamisel tuvastatakse klasse, mis vajavad refaktoreerimise käigus erilist tähelepanu ja analüüsi. [2]

Programmi kuumpunkt defineeritakse läbi koodi keerukuse ja selle toimetamise tiheduse. Koodi keerukuse arvutamiseks saab kasutada McCabe'i tsüklomaatilise keerukuse ja Halstead'i keerukuse mõõte ning ka koodi ridade arvu, mida pakub Tornhill [2]. Antud

umbkaudne mõõt on siiski korrelatsioonis teiste mainitud keerukuse mõõtudega ehk samamoodi kasulik [2], [9]. Tornhill valib koodi ridade arvu mõõdu selle programmeerimise keelest sõltumatuse pärast. Käesoleva analüüsi jaoks on olulisem selle mõõdu arvutamise ja interpreteerimise lihtsus ja kiirus.

Koodi ridade arvutamiseks on kasutuses avatud lähtekoodiga käsurea tööriist nimetusega *cloc* [10]. Tööristaga saab käsitleda faile erinevates programmeerimiskeeltes. Tulemusena on eraldi antud lähtekoodi-, kommentaari- ja tühjade ridade arv. Käesolevas analüüsis käsitletakse ainult Java faile ja arvutakse ainult lähtekoodi read, ignoreerides kommentaaride ridu ja ka tühje ridu.

Faili toimetamise tihedus annab ülevaadet sellest, kui palju aega sellega tööd tehakse. Kui faili toimetatakse tihti, siis selle refaktoreerimine toob suure tõenäosusega kaasa ka produktiivsuse võidu. Samuti esinevad praktikas kõrge toimetamissagedusega failides kvaliteediprobleemid. [2]

Programmikoodi toimetamise tihedust saab arvutada lihtsa skripti abil rakendades „*git log*“ käsku (Joonis 2) [2]. Skriptis saab määrata meid huvitava projektiskoobi. Käesoleva analüüsi jaoks määratakse skoobiks kasutajaliides, kuna peamine osa veebikoosolekute haldusfunktsionaalsusest on seal teostatud.

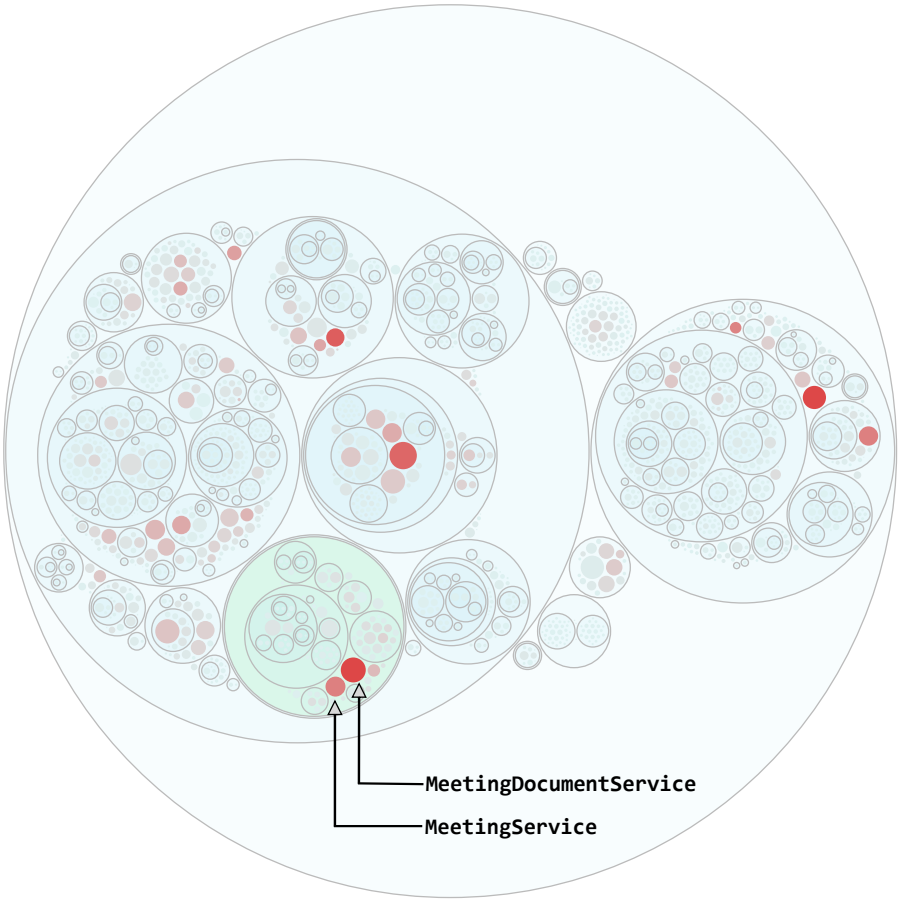
```
PATH_TO_UI="casem-ui/src/main/java"
$ git log --format=format: --name-only | grep "$PATH_TO_UI" | sort \
      | uniq -c | sort -r > ui_changes_frequencies.txt
```

Joonis 2. Skript failide toimetamise tiheduse leidmiseks.

Antud skript ei arvesta failide liikumist ega ümbernimetamist, seega ei ole ideaalne, kuid on piisav antud analüüsi jaoks.

Andmete analüüsimiseks pakub Tornhill andmeid visualiseerida kasutades kas ringi pakitud diagrammi (*circle packing* või *enclosure diagramm*) või *TreeMap* diagrammi. Mõlema diagrammi oluline omadus on hierarhilisus, mis on tagatud projekti struktuuriga. Suurema projekti jaoks on parem, kui diagramm esitatakse interaktiivses keskkonnas, kus saab diagrammi osasid suurendada. Ringi pakitud diagrammi puhul ringi suurus vastab koodi keerukusele (ehk lähtekoodi ridade arv failis) ja ringi punase värvi intensiivsus vastab toimetamise tihedusele (tausta värviga ringid iseloomustavad pakette (*package*) ja sisaldavad teisi ringe). [2]

Saadetud andmete alusel on tehtud kasutajaliidesele kuumpunktide diagramm (Joonis 3). Veebikoosolekute pakett on märgitud helerohelise värviga, ning nooltega on märgitud selle paketti kuumpunktid.



Joonis 3. CaseM kasutajaliidese kuumpunktide visualiseerimine ringi pakitud diagrammi kujul.

Kuna kasutajaliidese mooduli koodibaas on pigem liiga suur, et seda saaks staatilise diagrammiga mugavalt lugeda, andmete järjestamine keerukuse ja toimetamise tiheduste alusel tabelis 1 tuuakse viis esimest tulemusrida, mis on kasutajaliidese kuumkohadeks.

Tabel 1. CaseM kasutajaliidese kuumkohad.

	Faili nimi	Koodiridade arv	Toimetamise tihedus
1	MeetingDocumentService.java	1964	492
2	DocumentImportWindow.java	2364	399
3	CasemUtil.java	1702	490
4	SearchView.java	1053	424
5	MeetingService.java	1255	328

Tulemuste põhjal on näha, et kaks kuumpunkti, mille hulgas on ka kõige prioriteetsem kuumpunkt, on seotud veebikoosolekute funktsionaalsusega. Saadetud tulemuste põhjal võib järeldada, et koosolekute funktsionaalsuse refaktoreerimine on mõistlik ja võib tuua kaasa produktiivsuse võidu. Samuti oleks refaktoreerimise käigus mõistlikum keskenduda MeetingDocumentService.java ja MeetigService.java failidele.

3.2 Arhitektuurist tulenevad probleemid

Projekti kasutajaliideses on mitmeid arhitektuurilisi puudusi, mille hulgas on ka veebikoosolekute teostuse arhitektuur, mis takistavad projekti sujuvat edasiarendamist. Kokkuvõtlikult, kasutajaliides võtab enda peale vastutusalad tagarakenduselt (ja/või mikroteenustelt). Kasutajaliides ise koostab DQL (*Documentum Query Language*) päringuid ja saadab neid tagarakendusele täitmiseks. See põhjustab probleemi nii kasutajaliidese, kui ka tagarakenduse jaoks. Tagarakendusel kasutatud Documentum-i süsteemi asendamine mistahes muu CMS-i (*Content Management System*) süsteemi vastu muutub raskemaks seoses sellega, et Documentum-i päringukeelt kasutatakse ka teistes moodulites. Seega, Documentum-i asendamine nõuaks palju muudatusi kasutajaliideses. Samal ajal, DQL päringe koostamise ja käivitamise programmiplokid (mis võivad olla jagatud mitme klasside vahel ja segatud kasutajaliidese komponentide loomise loogikaga) teevad ka kasutajaliidese raamistiku asendamise raskemaks. Alternatiivne kasutajaliidese kirjutamine nõuaks palju äriloogika dubleerimist ning tõenäoliselt algse kasutajaliidese ümberkujundamist (äriloogika eraldamine kasutajaliidesest teiste teenustesse).

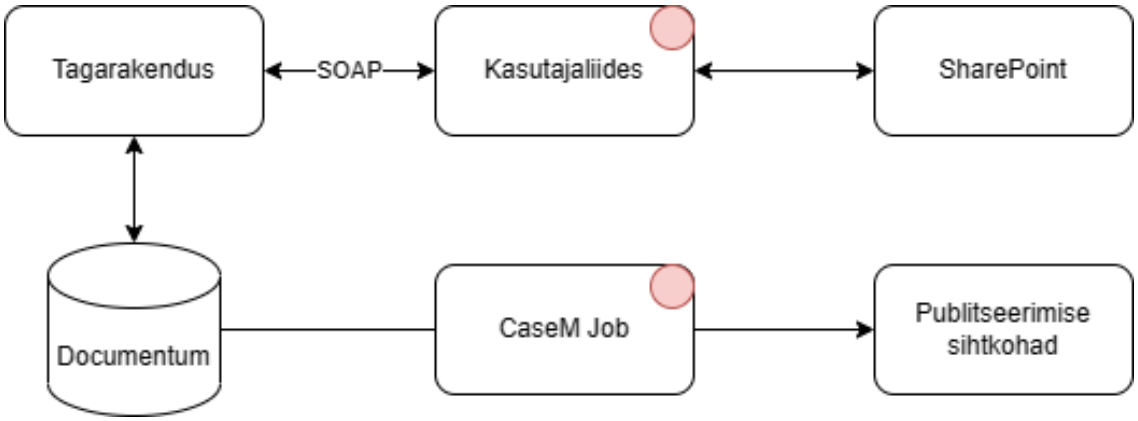
Koosolekute haldusfunktsionaalsuse teostamine kasutajaliideses on problemaatiline ka muudel põhjustel. Näiteks, koosoleku arutlusteemade ja manuste saatmine SharePoint-i ning koosoleku päevakorra genereerimine on kasutaja vaatenurgast aatomiliseks toiminguks (*atomic operation*). Pärast protsessi käivitamist ei tohiks see mingil juhul kasutajaliidesega seotud olla ja ainus asi, millele kasutajaliidesel võib olla juurdepääs, on protsessi edenemise protsent. Kuid kuni viimase ajani operatsioon võis keset tööd katkestada veaga kasutaja ootamatute toimingute pärast (näiteks väljalogimine), ning mingeid andmete tagasipööramist sellisel juhul ei toimunud.

Kasutajaliidese- ja teenuseloogika segunemise kohta võib tuua ka teisi näiteid, mida võib leida mõnest koosolekuhaldusega seotud operatsiooniklassist. Operatsioonideks on Java objektid, mis esindavad kontekstimenüü või akna nupu toiminguid. Ja need toimingud on

mõnikord teenuseloogikaga üle koormatud. Näiteks, klass *PrintMeetingExtractOperation* on nimetuse järgi toiming, mis peaks käivitama *MeetingExtract* dokumendi allalaadimise kasutaja brauseris. Kui dokument pole saadaval, tuleks see enne toimingut genereerida. Oleks õige, kui toiming kasutaks dokumendi ettevalmistamiseks sobivat teenust (praeguse projekti kasutajaliidese puhul on *MeetingDocumentService*). Siiski on palju loogikat rakendatud operatsiooniklassi sees. Antud aspekt muudab operatsiooniklassi keeruliseks ja õigustamatult pikaks.

Samuti kasutajaühendusega laetakse ainult neid koosoleku andmeid, millele kasutajal on juurdepääs. Kuid SharePoint-i integratsiooni jaoks ja dokumentide genereerimiseks kasutatakse administraatori kasutaja ühendust täisandmete laadimiseks. Protsessi jooksul toimub erinevate ühendustega mitu uut andmete laadimist ja lukustamist, mis tekitab jõudlusprobleeme.

Skemaatiline ülevaade projekti moodulite seostuste kohta vaadeldava funktsionaalsuse skoobis on antud järgneval skeemil (vt Joonis 4). Punasega on märgitud moodulid, mis sisaldavad veebikoosolekute haldusfunktsionaalsust, mida käesolevas töös analüüsitakse ja refaktoreeritakse.



Joonis 4. Veebikoosolekute funktsionaalsusega seotud moodulite skeem.

3.3 Programmi lähtekoodi probleemid

Käesolevas peatükis kirjeldatakse olemasoleva koodi probleeme ning uuritakse variante probleemide parandamiseks. Probleemide näidised tulenevad peamiselt *MeetingDocumentService.java* ja *MeetigService.java* failidest kuna refaktoreerimise vajadus oli nendes suurim (peatükk 3.1).

3.3.1 Dubleerimine

Dubleerimine on üks tuntumatest koodilõhnadest, mis käib vastu DRY-printsiipi (akronüüm inglise keelsest väljendist *Don't Repeat Yourself*), mis on formuleeritud Andrew Hunt'i ja Dave Thomas'iga oma raamatus „The Pragmatic Programmer". DRY-printsiibi järgi on koodi kordused lubatud ainult juhul, kui neid iseloomustavad erinevad ideed ehk koodiplokid ei ole loogiliselt seotud. Printsiibi rakendamise põhiväärtus seisneb selles, et ühe süsteemi komponendi muutmisel teised loogiliselt seostamata komponendid jäävad muutmata. DRY koodi muutmine on lihtsam ka sellepärast et, programmeerijatel puudub vajadus erinevate koodiplokkide vahel võrdlemisel veenduda, et need plokid käituvad üheselt. [11]

Vaadeldavas funktsionaalsuses on dubleerimine üks probleemidest, mis tuleneb funktsionaalsuse esialgse planeerimise puudusest (kogu äriloogika teostamine kasutajaliidese moodulis). Veebikoosolekute dokumentide mudeleid genereeritakse kasutajaliideses, peamiselt *MeetingDocumentService* klassis. Hiljem on sama dokumentide genereerimise loogika dubleeritud veebikoosolekute publitseerimistöös, mis on teostatud taustatööde moodulis ja ei ole seotud kasutajaliidese mooduliga. Antud teostused on ilmselt loogiliselt seotud ning funktsionaalsuse muutmine ühes moodulis nõuab ka teise mooduli toimetamist, välja arvatud erireeglite lisamine.

Kuna kasutajaliidese ja taustatööde moodulid ei tohi üksteisest sõltuda, siis ainsaks võimalikuks lahenduseks on kolmanda programmimooduli rakendamine, mida saab ülalmainitud moodulites kasutada, kas läbi koodi sõltuvuse või näiteks, interneti päringute kaudu. Moodul peaks sisaldama nii andmete laadimist kui andmete töötlemisfunktsionaalsust koosoleku dokumentide genereerimiseks. Antud aspekt võib olla keeruline, kuna moodulid kasutavad erinevaid andmeallikaid: kasutajaliidese kasutab CaseM tagarakendust ja taustatööde moodulil on Documentum-ile (ehk andmebaasile) otsene juurdepääs, kuna taustatööd toimetavad Documentum-i serveris. Siiski on antud samm vajalik lähtekoodi ainukese usaldusväärse allika tagamiseks ja funktsionaalsuse hooldatavamaks muutmiseks.

3.3.2 Suur klass

Üldjuhul, klasse ei tehta suurteks. Nemad pigem kasvavad vastavalt muutustele. Enamus uutest tarkvaraomadustest, mida lisatakse programmile, on väiksed ja tihti nõuavad

suhteliselt lühikest koodimuutust. Seoses sellega on lihtsam lisada vajalik kood olemasolevasse meetodisse ja klassi. Kuid põhjalikult koodi uurimisel võib uue klassi lisamine ja vana koodi refaktoreerimine osutuda asjakohasemaks. Pikemas perspektiivis, klassid kasvavad ja muutuvad raskemini hooldavamateks. Suuremates klassides on tihti segatud mitu vastutusala, mistõttu on põhjust klasse sageli redigeerida. Suurte klasside puhul on see aga tülikas. Samuti suuremad klassid kipuvad liigselt koodi kapselduma (*encapsulate*), mistõttu on raskem saavutada koodi head testikatvust. [7]

Mõlemad klassid *MeetingService* (1255 koodirida) ja *MeetingDocumentService* (1964 koodirida) on suured, mõlemal on mitu vastutusala ning mõlemad vajavad refaktoreerimist. *MeetingDocumentService* klassi põhivastutusala on koosolekute dokumentide mudelite genereerimine (edasi mudelid saadetakse PDF genereerimise teenusele valmisdokumentide saamiseks). Nagu sai mainitud peatükis 3.3.1, dokumentide mudelite genereerimise loogika on dubleeritud taustatööde moodulisse. Ehkki selles moodulis on mudelite genereerimise protsess peamiselt sama, on koodi refaktoreeritud kasutades rohkem objektikeskset lähenemist: iga dokumendi mudeli tüübile jaoks loodi vastav klass, millega genereeritakse antud mudel. Ühised omadused ja käitumine on defineeritud ülemklassides. Autori arvates sobib objektikeskne lähenemine dokumendi mudelite genereerimise funktsionaalsuseks ja ta kavatseb antud lähenemist kasutada. Kuid ka refaktoreeritud taustatööde mooduli koodis on veel teatud vastutusalad — andmete pärimine ja andmete töötlemine, mis oleks tarvis klassidesse jagada. Andmepäringute koostamise ja täitmise eraldamine oma klassi, mida antakse edasi sõltuvuse või argumendina, lihtsustab funktsionaalsuse testimist, kuna testi andmeid on lihtsam edasi anda töötlemisele. Samuti eemaldades andmete pärimise mudelite koostamise funktsionaalsusest, ei mõjuta andmeallika muutus dokumendi koostamise loogikat, ehk ka selle testid jäävad kehtivateks.

Teiste *MeetingDocumentService* klassi avalikkude meetodite (millest on kokku 36, konstruktorit, getter ja setter meetodite arvestamata) analüüsimisel tuleb välja, et klassis on funktsionaalsus struktureeritud dokumentide töötlemiseks. Lisaks on meetodid, mis koostavad ja saadavad tagarakendusele päringuid, ning mõned utiliit-tüüpi meetodid. *MeetingDocumentService* klass on ülekoormatud kohustustega, ka need peavad olema jagatud ja suunatud sobivamatesse klassidesse.

MeetingService klassiga peamiselt päritakse andmeid tagarakenduselt, aga klass vastutab *Meeting*, *MeetingParty* ja *MeetingTopic* olemite pärimise ja käsitlemise eest. Kuigi *Meeting* olemi objektiga seotud loogika tihti eeldab, et ka vastavad *MeetingTopic* ja *MeetingParty* olemi objektid oleksid *Meeting* olemis viidatud, on nende olemitega on seotud ka iseseisvad protsessid. On mõistlik jätta *MeetingService* klassi ainult *Meeting* olemile vastav kood ning suunata teiste olemite meetodid oma teenusklassidesse. Iga saadud teenuse puhul tuleks hinnata vajadust selle edasi jagamiseks teenuseks ja hoidlaks, juhul kui päringute moodustamine ja äriloogika on segatud.

Veebikoosolekute funktsionaalsuses enim kasutatud klassidele tuleb uurida klassi jagamise asjakohasust, lähtudes sellest, kui suur on klass ning kui palju kohustusi on klassis segatud.

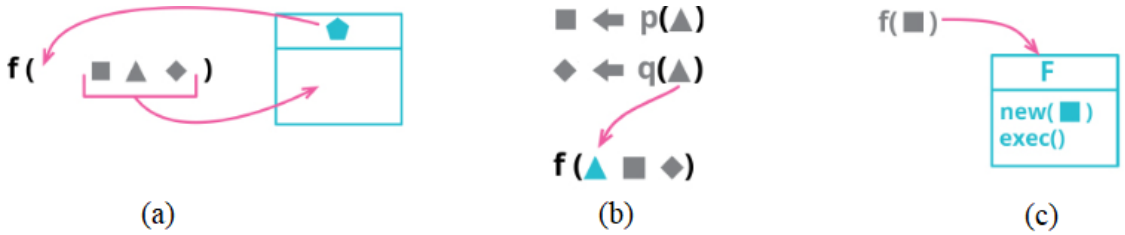
3.3.3 Pikk funktsioon

Raamatus „Clean Code“ kirjutatakse, et meetodid peavad olema lühikesed ja tähendusrikaste nimetustega, peavad teostama ühe asja ilma kõrvaltoimeteta ning sisaldama ainult ühte abstraktsioonitaset. Kui meetod vastab nendele juhistele, siis on meetod ennast kirjeldav ning kergelt hooldatav ja testitav. Järelikult pikkadele meetoditele on ilmne koodilõhn. Neid on raske mõista ja säilitada ning nendes kipuvad olema rikutud ka teised puhta koodi juhised. [12]

Programmi refaktoreerimisel on pikkade meetodite liigendamine kohustuslik. Enamasti tehakse liigendamist meetodi ekstraheerimise võttega (*Extract Function*). Kindlad viited meetodi ekstraheerimise vajadusele on originaalmeetodite sees olevad kommentaariplokid. Samamoodi koodi lugemisel, kui tekib tunne, et midagi vajab kommenteerimist, on näha, et koodiplokki tuleb ekstraheerida uude meetodisse. Selleks, et uue meetodi nimi oleks ennast kirjeldav, peaks koodi eesmärk olema nähtav. [8]

Rakendades meetodi ekstraheerimist on sageli vaja palju argumente ja ka ajutisi muutujaid üle kanda uuele meetodile, meetodi liiane sisendargumentide loend võib samuti vähendada koodi loetavust. Levinud lähenemisviisid argumentide käsitlemiseks on parameetri objekt (*Introduce Parameter Object*), millega ühendatakse argumentide loend ühte klassi, milles on väljad vajalike argumentide jaoks (Joonis 5.a). Mõnikord on sobivam anda argumendina algne objekti tervikuna (*Preserve Whole Object* tehnika), mitte ainult mõned objekti väljade väärtustest (Joonis 5.b). Kui algsel meetodil oli palju

ajutisi muutujaid, mida on vaja ka ekstraheeritud meetodites, saab kasutada meetodi objekti (*Replace Method with Method Object* ehk *Replace Function with Command*) refaktoreerimise tehnikat. Meetodi objekt hakkab sisaldama algse meetodi(te) loogikat, mis jagatakse mitmeks väiksemaks privaatseks meetodiks ning algselt kohalikud muutujad tehakse antud meetod objekti väljadeks (Joonis 5.c). [8]



Joonis 5. Refaktoreerimise tehnikad: (a) *Introduce Parameter Object*, (b) *Preserve Whole Object*, (c) *Replace Method with Method Object* [8].

3.3.4 Surnud kood

Surnud koodiks nimetatakse koodi, mida programmi töös kunagi ei käivitata . Lisaks klassidele ja meetodile, mida programmis enam ei kasutata, võib surnud koodi leida tingimusplokkidest, mille tingimuslause väärtus on alati väär. Aja möödudes surnud kood vananeb ja ei järgi uuemaid süsteemi tavasid ega reegleid, kuigi jätkub edukas kompileerimine. Refaktoreerimise protsessil tuleks surnud kood programmist eemaldada. Antud aspekt vähendab programmi mahtu ning tagab, et arendajad ei raiska aega surnud koodi uurimisele. Juhul, kui tekib vajadus eemaldatud funktsionaalsuse järele, peab see olema versiooni kontrollisüsteemist kättesaadav. [8], [12]

Veebikoosolekute funktsionaalsuse esialgsel teostamisel oli CaseM kasutajaliidese ja SharePoint teenuse suhtlemine korraldatud kasutades SOAP (*Simple Object Access Protocol*) protokolli. Funktsionaalsuse edasises arendamises oli lisatud ka alternatiivne REST (*Representational State Transfer*) teenus. Kasutatav teenus (SOAP või REST) valitakse kliendi konfiguratsioonifailis, kuid praktikas ei ole mitte ühtegi klienti, kellel oleks SOAP teenus valitud. SOAP teostuse koodis on palju skeemiklasse ning need koos vastava koodiga tuleks kasutajaliidese lähtekoodist eemaldada.

3.3.5 Katvus testidega

Ebapiisav testide katvus on oluline probleem arendava projekti jaoks. Testide kirjutamine on viis programmi dokumenteerimiseks. Korralikult testitud programm on oluliselt

paremini hooldatav ja uuendusteks valmis. Samuti, refaktoreerimise protsessi testid on üliolulised vigade koheseks tuvastamiseks. [7]

Pärandkoodi puhul võib korralike testide kirjutamine osutada võimatuks püsiprogrammeeritud sõltuvuste tõttu. Sellistel juhtudel saab rakendada sõltuvustest vabanemise (*dependency-breaking*) tehnikaid. Sellised tehnikad on sisuliselt refaktoreeringud, mis ei pruugi koodi disaini parandada (ning võivad seda isegi rikkuda), kuid kindlasti ei muutu programmi käitumist, vaid võimaldavad koodi testimist, mille tulemusena on kood paremini hooldatav. Sõltuvustest vabanemise tehnikate loend on leitav Michael Feathers'i raamatust. [7]

Testide olemasolul saab rakendada disaini parandavaid refaktoreerimise võtteid. Üldjuhul disaini refaktoreeringu rakendamise algoritmiks on luua uus ajutine meetod (või klass) ja toimetada koodi selles, jättes originaalse meetodi puutumata. Kui kood saab refaktoreeritud, siis käivitamiseks viidatakse talle originaalses meetodis refaktoreerimata koodi asemel. Meetodi signatuuri säilitamisel refaktoreeritud meetodit saab valideerida olemasolevate testidega. Peale edukat testimist tuleb vana kood kustutada ning vajadusel lahendust edasi refaktoreerida, muutes meetodi signatuuri ja teostades vastavad uuendused testkoodis. [8]

On oluline, et testi meetodite kood jälgiks puhta koodi printsiipe analoogselt programmi lähtekoodile, ainuüksi korralikust testidega kaetusest ei piisa. Programmi koodi muutmisel võib tekkida ka testimise koodi toimetamise vajadus, kuid halvasti kirjutatud testide puhul on nende uuendamine aeganõudev. Samuti peab test olema arusaadavalt kirjutatud, et aidata leida võimalikke edasiarendamisel tekkivaid probleeme. Siiski testimiskoodi standardid erinevad programmi koodi standartidest. Testimiskood peab olema lihtne ja väljendusrikas, kuid see ei pea olema samaväärselt tõhus mälu ja protsessori kasutamisel, võrreldes programmi lähtekoodiga. Ollakse arvamusel, et testimiskood võib rikkuda ka mõningaid objekti keskse programmeerimise printsiipe (näiteks kapseldamist) juhul, kui see muudab testide kirjutamise lihtsamaks [7]. Kõige tähtsam on see, et testimiskoodi oleks lihtne mõista ja hooldada. [12]

Koosolekuhalduse funktsionaalsus on ebapiisavalt testitud. Kasutades IntelliJ IDEA arenduskeskkonna *Run with coverage* funktsionaalsust [13], arvutati veebikoosolekute

haldamise funktsionaalsuse testi katvus. Testi katvuse tulemused mõnede funktsionaalsuse olulisemate klasside jaoks on toodud tabelis 2.

Tabel 2. Veebikoosolekute funktsionaalsuse testi katvus.

Klassi nimetus	Meetodid kaetud testidega (%)	Koodi rida, mis on kaetud testidega (%)
MeetingDocumentService	7	8
MeetingService	13	7
MeetingHelper	8	7

Koosolekute funktsionaalsuse refaktoreerimise protsessi alustamisel on oluline katta kood testidega, selleks on abiks sõltuvuste vabanemise tehnikad.

3.4 DQL päringud

Peatükis 3.2 on kirjeldatud projekti arhitektuuri puudusena, et kasutajaliideses päritakse andmeid tagarakendusest kasutades DQL päringuid. Päringulauseid moodustatakse kasutajaliideses, puudub kindel moodul või klasse, milles neid koostada. Samuti ei ole ühist funktsionaalsust päringute moodustamiseks. Kasutatakse kas konstantseid sõnesid, sõnede vormindamist (*String.format*) või ehitamist (nt. *StringBuilder*). Antud pärimise teostuse tõttu on raske Documentum asendada mõne teise päringukeelt kasutatava süsteemiga.

DQL päringute koostamine peab olema üle vaadatud veebikoosolekute funktsionaalsuse refaktoreerimise raames. On oluline tõsta päringulausete ja nende moodustamise eest vastava koodi mõistetavust ja hallatavust. Esimene oluline muudatus, mis on osa peamisest refaktoreerimise protsessist — päringute moodustamise meetodid kogutakse sobivatesse klassidesse (näiteks hoidlad) ning pärimise- ja äriloogika eraldatakse. Teiseks võimalikuks parandusideeks on võtta kasutusele mingi vahend, mis lubaks DQL päringute kirjutamist spetsialiseeritud Java objektide kaudu ehk kasutada päringuehitaja lähenemist. Eeldades, et mingil hetkel loobutakse Documentum kasutamisest (näiteks, finantsalaste tegurite tõttu), võib päringu ehitajalahenduse kasutamine olla eriti kasulik ja aidata vähendada vajalike koodimuudatuste hulka. Sellisel juhul peaks päringuehitaja toetama serialiseerija teostuse muutmist andmeallika tüübi põhjal.

3.5 Staatiliselt tüpiseeritud olemiklassid

Staatiliselt tüpiseeritud klass on klass objektorienteeritud programmeerimises, kus iga muutuja andmetüüp deklareeritakse kompileerimise ajal ja seda ei saa muuta käitusajal. See tähendab, et klassi kompileerimisel kompilaator kontrollib, et kõik muutujate tüübid oleksid õiged ja sobiksid nendega teostatavate operatsioonidega. Tüüpidega seotud vigade tuvastamine toimub enne programmi käivitamist. [14]

CaseM-i rakenduse andmemudel on konfigureeritav ja võib oluliselt erineda erinevatel organisatsioonidel. Selleks, et toetada erinevaid konfiguratsioone, on CaseM tagarakenduse tagasipöördumistes objektid esitatud omaduste loendina, kus iga omadus koosneb identifikaatorist ja väärtuste loendist. Antud omadus võimaldab keerulisemate DQL päringute puhul koostada vastust mitmetest tabelitest.

Projektis puudub ühtne lähenemine sellele, kuidas edasi teisendama tagarakenduse poolt tagastatud objektid. Enamasti teisendatakse need saavad omaduste kujutise sisse, mille puhul võtmed on omaduste identifikaatorid ja kõik väärtused hoitakse *Object* tüübina. Probleemiks on fakt, et edasiseks kasutamiseks vajab *Object* tüüpi väärtus tüübi teisendamist (operatsiooni nimi Java keeles on *casting*), et lubada tüübile vastavate operatsioonide kasutamist. Samuti *Object* tüübi alla saab kirjutada mistahes väärtuse, ehk teoreetiliselt omaduse väärtuse tüüp võib käitusajal muuta. Antud struktuuri kasutamine raskendab andmete edasikasutamist ning lähtekoodi loetavust ja arusaadavust, kuna ei saa kindlalt öelda, mis andmetüübiga on tegu.

CaseM kasutajaliideses omaduste kujutis on mähitud (*wrapped*) vastavasse olemklassi, mis lubab antud klassis kasutada tüpiseeritud ja/või üldist (*generic*) getter ja setter meetodeid. Praktikas tüpiseeritud getter ja setter meetodeid on vähe ning nende lisamine iga omadusele võib (teatud olemite puhul) tekitada juurde sadu koodiridu. Samuti on võimatu *lombok* [15] teegi kasutamine. Üldiste meetodi kasutamine omakorda vajab omaduse identifikaatori teadmist ning omaduse andmetüüpi eeldamist.

Et teha programmi lähtekood arusadavamaks ja veakindlamaks, tuleb uurida viise, kuidas teisendada tagarakenduse tagasipöördumise objektid olemiklassidesse tüpiseeritud väljadega.

3.6 Analüüsi kokkuvõte

Käesolevas peatükis vaadeldi veebikoosolekute funktsionaalsusega seotud probleeme. Üldisemal tasemel sai uuriti, kui tihti funktsionaalsus toimetatakse ning miks selle refaktoreerimine võib olla kasulik CaseM projektile. Koodi tasemel uuriti funktsionaalsuse koodis esinevaid probleeme ning pakuti meetmeid koodi disaini ja funktsionaalsuse arhitektuuri parandamiseks. Analüüsi käigus tuvastatud kõige olulisemad probleemid on:

- Veebikoosolekute dokumentide genereerimise loogika on dubleeritud kasutajaliideses ja taustatööde moodulis.
- Veebikoosolekute funktsionaalsusele vastavate põhiklassid on ülekoormatud erinevate vastutusalade äriloogika plokkidega.
- Kasutajaliidese spetsiifiline loogika on segatud veebikoosolekute haldusloogikaga.
- Lähtekoodi üldine keerukus, mis on põhjustatud puhta koodi printsiipide rikkumisest.
- Funktsionaalsus ei ole testidega kaetud.

Analüüsi põhjal arvab autor, et veebikoosolekute funktsionaalsus vajab refaktoreerimist.

4 Lahenduse kavandamine

Antud peatükis sõnastatakse nõuded lahendusele, valitakse selle arendamisel kasutatav arhitektuur ja tehnoloogiad, kavandatakse veebikoosolekute haldusfunktsionaalsuse refaktoreerimisprotsessi plaan.

4.1 Nõuded lahendusele

Lähtuvalt probleemianalüüsist tuleb funktsionaalsuse refaktoreerimisprotsessis järgida järgnevaid nõudeid:

- Olemasoleva funktsionaalsus tuleb refaktoreerimisel katta testidega. Refaktoreerimise protsess nõuab üksusteste, mida saab muuta ja täiendada koos koodi refaktoreerimisega [8].
- Dubleerivad loogikad ja mittekasutatud kood tuleb kõrvaldada.
- Funktsionaalsuse arhitektuur tuleb üle vaadata ja koodiblokid jagada ja ümber korraldada vastavalt vastutusaladele.

Võimalusel võiks DQL päringute koostamise viisi parandada, kasutades näiteks, päringuehitajat.

Üldised nõuded refaktoreeritud lahendusele on järgmised:

- Lahendus peab olema kättesaadav kasutajaliidese ja taustatööde moodulitest.
- Lahendus peab olema hästi testitud ja hooldatav.
- Kui lahenduse arhitektuuriks on veebirakendus, peaks see vastama projekti kõigile turvanõuetele ja sellel peab olema dokumenteeritud API (*Application Programming Interface*).

4.2 Arhitektuuri valik

Veebikoosoleku haldamise loogika teostus ei peaks sõltuma kasutajaliidesest ega tööde moodulist, kuid peab olema antud moodulites saadaval. Üks võimalik koht koosoleku haldusfunktsionaalsusele on CaseM-i tagarakendus. Sellisel juhul ei teki kasutajaliidesele

ja tasutatööde moodulile uusi sõltuvusi. Teine võimalik koht oleks uus mikroteenuse moodi tehtud rakendus. Viimane lähenemine pakub järgmisi eeliseid, võrreldes tagarakendusega:

- CaseM-i tagarakendus kasutab SOAP-protokolli, kuid mikroteenusel saab rakendada REST veebiliides, mida on lihtsam rakendada ja kasutada mistahes kasutajaliideses, kuna antud edastusandmete andmevormingu on paindlikum, kui SOAP [16].
- Uues mikroteenuses saab kasutada kaasaegsemaid tehnoloogiaid, näiteks Spring Boot-i. Nii ei pea integreerima funktsionaalsust vana EJB-süsteemi (*Enterprise JavaBeans*). Lisaks saab uus teenus omada arhitektuuri, mis sobib selle eesmärgiga kõige paremini. [17]
- Mikroteenust saab paigaldada ainult nendele klientidele, kes tegelikult kasutavad veebikoosolekute funktsionaalsust [17].
- Uut rakendust on lihtsam katta testidega, kui arhitektuur toetab testimist algusest peale.
- Uuel teenusel on parem hallatavus, võrreldes integreeritud koodiga, seda on lihtsam mõista ja muuta, teenuse lähtekood on lühem [17].

Mikroteenuse tüüpi lähenemisel on ka mõningaid puudusi, näiteks suurenenud paigaldamise keerukus [17]. Siiski, võttes arvesse eeliseid, on mikroteenuste rakendamine soodsam, kui funktsionaalsuse integreerimine tagarakendusse. Oluliseks faktoriks on ka see, et lõputöö peamiseks eesmärgiks on funktsionaalsuse refaktoreerimine, mis on uues rakenduses oluliselt lihtsam teostada.

Mikroteenuse parimate tavade hulgas on kasutada igal mikroteenuse jaoks oma eraldiseisvat andmebaasi [18]. Uuel teenusel ei tohi aga oma eraldi andmebaasi olla, vaid selleks peab kasutama tagarakendust ja pärima andmeid sealt. Seda põhjendatakse projekti arhitektuuriga, mille kohaselt kasutab süsteem andmete (sealhulgas koosolekute andmete) säilitamiseks just Documentum süsteemi. Samuti tagarakendus vastutab enamiku üksuste ja seoste valideerimise ning seotud protsesside, nagu ACL-ide (*Access Control List*) värskendamise ja täitmise eest.

4.3 Tehnoloogiate valik

Antud peatükis põhjendatakse mooduli arendamiseks tehtud tehnoloogiate valikuid ja kirjeldatakse valitud tehnoloogiaid.

4.3.1 Teenuse raamistiku valimine

Mikroteenuste arhitektuuri üheks eeliseks on võimalus valida iga mikroteenuse jaoks sobivaim tehnoloogia. Koosolekute haldamise teenuse jaoks valitakse Java 11 programmeerimiskeel, kuna projekti arendus toimub Java arendusmeeskonna poolt. Kuigi antud töö teostamise ajal on Java 17 uusim LTS (*Long Time Support*) versioon, CaseM projektis kasutatakse praegu Java 11 versiooni.

Koosoleku haldamise teenuse arendamiseks kaalutakse järgmisi raamistikke: Spring Web MVC, Spring WebFlux ja Micronaut. Antud valik põhineb autori teadmistel ja tema huvivaldkonnal ning arvestab ka juba olemasolevaid veebiteenuse lahendusi projektis.

Spring Boot koos Spring Web MVC-ga on kasutusel ka teistes projekti veebiteenustes. Neid teenuseid kasutatakse tavaliselt kasutajaliidese või tagarakenduse ja kolmandate osapoolte teenuste või CaseM-i projekti laienduste vaheliseks suhtluseks. Teenustel on REST-ful veebiliides ning tagarakendusega suhtlemiseks on loodud moodul SOAP-kommunikatsiooni käsitlemiseks. Turvaseadistused ühenduste käsitlemiseks on samuti moodulina rakendatud ja kasutavad Spring Security teeki.

Spring WebFlux on osa Spring raamistikust, mis on analoogne Spring MVC-ga, kuid on mitte-blokeeriv veebirakenduste teostus, mis kasutab reaktiivset lähenemist. Peamine eeldatav kasu mitte-blokeerivatest reaktiivsetest rakendustest on võime skaaleeruda väiksema lõimede arvu ja vähema mälukasutusega, mis muudab rakendused koormuse all vastupidavamaks. WebFluxi ja Web MVC võrdlust tulemuste kohta erinevates tingimustes saab uurida, näiteks Valeri Andrejev'i bakalaureusetöös [19]. [20]

Micronaut on avatud lähtekoodiga JVM-põhine (*Java Virtual Machine*) raamistik kergekaaluliste rakenduste ja mikroteenuste loomiseks. Antud raamistik on suhteliselt uus, selle esimene versioon avaldati 2018. aastal. Üks selle peamistest omadustest on sõltuvuste süstimine ja AOP (*Aspect-Oriented Programming*) mudelite loomine kompileerimise ajal. Raamistikud nagu Spring kasutavad sõltuvuste süstimiseks Java *Reflection* (ehk peegeldamist) API-t sõltuvuste süstimiseks, mis nõuab iga rakenduse

kontekstis kasutatava bean-i jaoks refleksioonide andmete laadimist ja vahemällu salvestamist. Micronaut-i lähenemine ei nõua sellist vahemällu salvestamist, mistõttu rakendus kasutab töö ajal vähem serveri mälu. Lisaks on kompileeritud lähenemine kiirem, kui see, mis kasutab peegeldamise API-d. Selleks, et Micronaut-i ja Spring Framework-i võrdlusega paremini tutvuda, soovitab autor tutvuda Kristjan Mäeotsa bakalaureusetööga [21]. [22]

Mikroteenuse raamistiku valimisel tuleks tavaliselt arvestada järgmiste aspektidega [23]:

- raamistiku populaarsus, sealhulgas kvalifitseeritud spetsialistide ja dokumentatsiooni kättesaadavus;
- kogukonna küpsus, mis tavaliselt tähendab raamistiku pidevat tuge, funktsioonide ja vigade parandamise väljalaske sagedust;
- arendamise lihtsus, mis hõlmab ka toetust arenduskeskkonna programmide poolt;
- õppimiskõver, mis näitab, kui lihtne on raamistiku õppida ja kui palju õppematerjale on saadaval;
- arhitektuuri tugi, mis tähendab raamistiku poolt pakutavate ehitusplokkide ja liideste ning sisseehitatud disainimustrite tuge;
- automatiseerimise toetus ehk kuidas raamistik toetab ehitus-, testimis- ja paigaldustööde automatiseerimist;
- sõltumatu paigaldus, mis tähendab, et raamistikul on tugi sõltumatu paigalduse aspektide jaoks: porditavus, taaskasutatavus, ühilduvus.

Autor leiab, et enamikes aspektides on vaadeldud raamistikud võrdsed. Micronaut on toetatud IntelliJ IDEA IDE poolt sama hästi kui Spring Framework [24]. Lisaks, paljud Micronaut-i programmiliidese aspektid on tugevalt inspireeritud Spring raamistikust, nii et selle õppekõver on sarnane Spring-iga. Micronaut on muutunud väga populaarseks, kuid Spring Framework on tunduvalt küpsem tehnoloogia laiema kogukonnaga, seetõttu on turvalisem kasutada Spring raamistikku, kuna saadaval on rohkem spetsialiste ning olemas ka dokumentatsioon, mille hulgas on probleemide kirjeldused ja lahendusviisid. Võib öelda, et enamik mainitud aspekte kehtib Spring Web MVC kohta. Antud raamistik on valitud raamistiku nimekirjast kõige vanem ja sellel on laiem valik erifunktsionaalsuse moodulitest saadaval, kuna ajalooliselt enamik neist on ehitatud blokeerivate tehnoloogiate põhjal [20].

Autori lisatud raamistiku valimise kriteeriumiks on tema teadmised antud raamistikest ja projekti arendusmeeskonna varasem kogemus antud raamistikega. Nende kriteeriumite põhjal valitakse arendava veebiteenuse jaoks Spring Web MVC.

Spring WebFlux lükatakse tagasi peamiselt meeskonna varasema kogemuse põhjal antud raamistikuga. Mõned eelpoolmainitud CaseM projekti REST-teenused loodi algselt Spring WebFlux-iga. Üks teenustest kaotas aeg-ajalt kolmandate osapoolte teenuse vastuseid. Üldine probleem selliste teenustega oli silumine, mis on raskem võrreldes Spring Web MVC teenustega. Samuti, koosolekuhaldamise teenuselt ei oodata sellist koormust, millega reaktiivne lähenemine annaks märkimisväärseid eeliseid. Siiski soovitab Spring kasutada reaktiivset WebClient-i teiste veebiteenuste pöördumiseks [20]. Veebikoosolekute haldamise teenuse ja SharePointi vahelise suhtluse rakendamise võimaluse uurimiseks tuleks uurida WebClient-i kasutamist. CaseM tagarakendusega suhtlemise moodul kasutab hetkel mitte-reaktiivset Spring-WS teeki, millele Spring ei paku reaktiivset alternatiivi.

Micronaut raamistik lükatakse tagasi peamiselt see tõttu, et see kasutab samuti reaktiivset lähenemist. Isegi kui otsustatakse kasutada Micronaut-i, on praegu olemas Spring-i sõltuvused, nagu Spring-WS ja Spring Security, mida tuleks siiski veebikoosolekute haldamise teenuses kasutada. Micronaut-il on alamprojekt *Micronaut for Spring*, mis peaks võimaldama kasutada Micronaut-i sõltuvuste süstimise lahendust Spring-i rakenduses, vähendades seeläbi Spring-i rakenduse mälu kasutust [25]. Seetõttu tasub uurida Micronaut-i süstimise süsteemi integreerimise võimalust.

4.3.2 Muud kasutatud tehnoloogiad

Järgnevalt on kirjeldatud tehnoloogiaid, mida plaanitakse lahenduse arendamisel kasutada ning mida eelnevad peatükid ei katnud.

RabbitMQ — sõnumivahetuse teenus, mis lubab mugavat teenuste vahelist suhtlemist ning muuhulgas lubab efektiivselt ja mugavalt lahendada asünkroonset protsesside töötlust [26].

Lombok — abistav Java teek, mis pakub annotatsioone korduva koodi automaatseks genereerimiseks. Lombok teegi eesmärgiks on vähendada lähtekoodi mahtu ning seeläbi tõsta koodi loetavust, kõrvaldades vajaduse sellise koodi käsitsi kirjutamiseks. [15]

MapStruct — koodigeneraator, mis oluliselt lihtsustab Java tüüpide teisendamist (*mapping*). Loodud kood kasutab otseselt Java meetodeid (ehk ei kasuta peegeldust) ja seetõttu on kiire, tüübiturvaline ja lihtsasti mõistetav. [27]

OpenAPI — spetsifikatsioon määratleb standardse liidese programmide veebiliidestele (API), mis võimaldab nii inimestel kui ka arvutitel mõista teenuse võimalusi ilma juurdepääsuta lähtekoodile, dokumentatsioonile või võrguliikluse inspekteerimise kaudu [28]. Enimkasutatud lahendus spetsifikatsiooni rakendamiseks rakenduses on Swagger [29].

Funktsionaalsuse testimisel oli kasutusel JUnit 5 [30] ning Mockito [31] teegid.

4.4 Refaktoreerimise plaan

Veebikoosolekute haldamise funktsionaalsuse refaktoreerimine koosneb mitmest põhietappidest, milleks on uue teenuse põhja arendamine, koosoleku dokumentide koostamise funktsionaalsuse refaktoreerimine, SharePoint integratsiooni refaktoreerimine ning teiste kasutajaliideses teostatud koosolekutega seotud operatsioone parandamine.

Teenuse põhja arendamise etapil luuakse kõik vajalikud olemiklassid, mis peavad olema tugevalt tüpiseeritud, arendatakse andmete juurderpääsu kihti ning uuritakse viise päringute hallatavuse parandamiseks. Lisaks integreeritakse ja seadistakse uude teenusesse CaseM Security, OpenAPI ja muud vajalikud moodulid.

Koosolekute dokumentide mudelite koostamise funktsionaalsuse refaktoreerimine on võrreldes teistega prioriteetsem, kuna vastavaid klasse toimetatakse sagedamini ja nendega seotud tööülesannete hulk on suur. Nagu peatükis 3.2 kirjeldatud, on antud funktsionaalsus on teostatud nii kasutajaliideses, kui ka taustatööde moodulis, ning see on suuresti dubleeritud. Teostus taustatööde moodulis sobib paremini refaktoreerimise alustamiseks, kuna see on paremini struktureeritud võrreldes kasutajaliidese teostusega. Antud funktsionaalsuse refaktoreerimise protsessi tuleb alustada olemasolevat koodi üksustestidega katmisest, kuna hetkel need puuduvad. Seejärel tuleb eraldada dokumentide mudelite andmeid päriv ja töötlev loogika, mis on oluline osa koodi refaktoreerimisest ning mis teeb funktsionaalsuse liigutamise uude teenusesse lihtsamaks. Kuna antud etapil uuritakse põhjalikult andmepäringuid, siis leitakse ka päringute

optimeerimise võimalusi, mis seisnevad peamiselt päringute arvu vähendamises ühe mudeli koostamise kohta. Refaktoreeritud lahenduse uude moodulisse liigutamisel kohandatakse andmeallikad ja andmeedastusobjektid uue teenuse andmejuurdepääsu kihiga ningvastavalt kohandatakseka teste. Funktsionaalsus, mis oli spetsiifiline kasutajaliidesele jaoks, juurutatakse ja testitakse uues teenuses. Lõpuks tuleb teha sobilikud API meetodid ja mugav API kliendi klass, mida saab kasutada kasutajaliideses ja tausta tööde moodulites. Refaktoreeritud funktsionaalsus tuleb verifitseerida.

Järgmiseks põhietappiks on SharePoint-i integratsiooni koodi viimine kasutajaliidesest uude teenusesse. Antud osa koodist on osaliselt testitud, testimata osad tuleb katta enne liigutamist üksustestidega. Uuritakse, mis osad SOAP protokolli suhtlemise teostuse koodist on kasutusel ning kustutakse antud teostusega seotud surnud kood. Liigutatakse ülejäänud kood uude teenusesse ning kohandatakse vastavalt ka andmete pärimise kood ja testid. Kui leitakse tagarakenduse pöördumised, mida saaks optimeerida, siis tehakse vastavad parandused. Teostatakse API meetodid, refaktoreeritud funktsionaalsus verifitseeritakse.

Viimaseks refaktoreerimise etappiks on täielik CaseM tagarakenduse kasutamise asendamine uue veebiteenusega koosolekute haldusfunktsionaalsuse raames. Kõigepealt keskendutakse asünkroonsetele ja aeganõudvatele operatsioonidele, viimaks aga triviaalseimatel operatsioonidele nagu koosolekute või arutlusteemade loomine, mis võiksid samuti uut veebiteenust kasutada.

5 Lahenduskäik

Järgnevates peatükides kirjeldatakse refaktoreerimise protsessi elluviimist kavandatud plaani põhjal. Kuna kogu veebikoosolekute haldusfunktsionaalsuse refaktoreerimine ja testitega katmine on töömahukas ja aeganõudev, siis piirdub käesoleva töö praktiline osa haldusmooduli põhja arendamise ning veebikoosolekute dokumentide genereerimise funktsionaalsuse refaktoreerimisega ja testidega katmisega. Dokumentide genereerimise funktsionaalsuse refaktoreerimine hõlmab koodi struktuuri parandamist, päringute optimeerimist ja funktsionaalsuse testidega katmist.

5.1 Veebiteenuse põhja arendamine

Rakenduse põhja loomisel kasutatakse Spring Boot-i ning Spring-i sõltuvuste süstimist. Micronaut-i peegeldumisvaba sõltuvuse süstimist ei saa arendatavas programmis rakendada. See on põhjustatud peamiselt *Micronaut for Spring* teekide dokumenteerimise (sealhulgas probleemide lahenduste postitusete) puudulikkusest: autoril ei õnnestunud leida teekide ja nende versioonide kombinatsiooni, mis töötaks rakenduses kasutatava Spring teekide versiooniga.

DQL päringute ehitamiseks valmislahendust autor ei leia. Üks projekt, mida on võimalik kohandada, on *QueryDSL*, mis on avatud lähtekoodiga Java raamistik, mis võimaldab tüübiturvaliste päringute genereerimist. Sellel on tugi erinevate tehnoloogiatele, nagu JPA (*Java Persistence API*), *Lucene*, JDBC (*Java Database Connectivity*) ja muud [32]. Autori arvatest on võimalik arendada laiendus *QueryDSL JDBC* mooduli põhjal, mis toetaks koodi serialiseerimist DQL päringuteks SQL-i asemele ning toetaks DQL-spetsiifilisi käsklusi. Kuna sellise lahenduse arendamine ja testimine on aeganõudev protsess ning ei ole käesoleva töö eesmärk, otsustatakse valida teised võimalikud viisid päringute mõistetavuse ja hallatavuse parendamiseks.

DQL päringulausete loetavuse tõstmiseks kasutatakse IntelliJ IDEA programmeerimiskeskkonna *Language Injection* funktsionaalsus (Joonis 6). Antud funktsionaalsus toetab mitu erinevat tehnilist keelt, nagu SQL päringukeel või

regulaaravaldised ja on vaikimisi kasutatud enimkasutatavates klassides nagu *java.sql.Connection* või *java.util.regex.Pattern*. Funktsionaalsust saab rakendada IntelliJ IDEA seadmetest, spetsialiseeritud kommentaariridа lisamise kaudu, või spetsialiseeritud *@Language* annotatsiooni kaudu, mis on JetBrains *annotations* teegi osa [33]. Kuigi funktsionaalsus ei toeta DQL päringukeelt, on võimalik kasutada SQL keelemudelit antud päringukeelte sarnasuse tõttu.

```
public class LanguageInjectionExample {
    String injectWithComment() {
        // language=SQL
        return "select ORDER_NUMBER from ORDERS where id = 1";
    }

    String injectWithAnnotation() {
        return query( query: "select id, order_number from orders order by order_number");
    }

    1 usage
    private String query(@Language("SQL") String query, Object... args) {
        return String.format(query, args);
    }
}
```

Joonis 6. *Language Injection* funktsionaalsuse rakendamise näidised arenduskeskkonnas.

5.2 CaseM olemi mudel

CaseM olemiklasside tugeva tüpiseerimise tagamiseks arendatakse CaseM olemi mudeli teek (inglisekeelne nimetus *CasemEntityModel*), mis vastutab *CMIS Property* (tagarakenduse poolt tagastatav objekti tunnuste esinduse nimetus) loendi teisendamise eest CaseM olemisse ja tagasi, ning aitab päringulausete koostamisel. Mudel ei kasuta Java peegeldust, vaid kompileerimisajal genereerib koodi spetsialiseeritud annotatsioonide põhjal, kasutades Java annotatsiooni protsessorit. Lahendus on inspireeritud MapStruct teegist, kuid sisuliselt antud teeki ei kasuta ja teegis arendatud funktsionaalsust ei saa asendada MapStruct teegi funktsionaalsusega.

CaseM olemiklass märgitakse *@CasemEntity* annotatsiooniga, mille argumendina antakse ka CaseM-i (või Dokumentum-i) andmetüüp (teatud *CasemTypes* enumeratsioonklassi väärtus). Olemiklassi väljadele antakse *@CasemEntityField* annotatsioon, millega seotakse olemklassi välja ja vastava andmetüübi omadus (sarnane tabeli veeruga, kuid Dokumentum-i andmetüübid koosnevad mitmetest tabelitest).

Ülemklasside väljad, millele on antud sama annotatsioon lisatakse samuti genereeritud mudelisse, mis võimaldab ühiste omadusete ühekordset (ühes ülemklassis) määratlemist. *CasemEntityField* annotatsiooniga määratletakse ka omaduse teisendamise vajadus (mida võib ka välja lülitada), saab seadistada omaduse lisamise olemi loomise ja uuendamise päringutesse, saab määrata, kui omaduse olemasolus on vaja veenduda organisatsiooni konfiguratsioonist. Omaduse tüüp on määratud vastava klassi välja tüüpiga; enim kasutatud Java tüübid on toetatud, enumeratsiooni (*Enum*) tüübid kaasaarvatud. Documentum-i korduvate omadusete (*repeating property*) jaoks saab kasutada loendi (*List*) ja hulga (*Set*) tüüpe. Olemi klassi näidis on toodud järgneval joonisel (vt Joonis 7).

```
@Setter @Getter // lombok on toetatud
@CasemEntity(
    value = CasemTypes.ams_meeting // olemi andmetüüp on ams_meeting
    changesAwareEntity = true // kui soovitakse genereerida
      // ChangesAwareEntity olemi alamtüüpi ning seda kasutada
)
//                  id väli (ehk id omadus) saab pärida ülemtüübilt
public class Meeting extends CasemObjectId {
  @CasemEntityField(
      // omaduse identifikaator
      value = "ams_nativeid",
      // kas omadus on andmetüübil olemas (on andmebaasis)
      persistent = true, // vaikimisi väärtus
      // kas vajab veendumist organisatsiooni konfiguratsioonis
      verifyInOrgConfig = false, // vaikimisi väärtus
      // kas genereeritud mudelisse lisatakse teisendamismeetodid
      // antud omaduse jaoks
      createMapping = true // vaikimisi väärtus
  )
  private String title;

  // korduva omaduse näidis
  @CasemEntityField("example_dates")
  private List<Date> repeatingDates;

  // teised väljad, jne
}
```

Joonis 7. CaseM olemi koodi näidis kommentaaridega.

Olemi teisendamine *CMIS Property* loendisse objekti uuendamise päringuks ei ole triviaalne. Terve olemi teisendamine ei ole soovitatav ega sobilik, kuna mõned omadused võivad olla ainult lugemiseks või omada muid erireegleid, mille pärast neid ei tohiks päringusse lisada. Lihtsamaks lahenduseks on anda vajalike omaduste identifikaatorite

loend teisendamise meetodi lisaargumendina, aga antud lahenduse puuduseks on olemimudeli abstraktsioonitaseme murdmine. Abstraktsiooni säilitavaks lahendamisviisina teostati *ChangesAwareEntity* liides, mida rakendatakse genereeritud olemi alamklassidele. Antud liidese ja alamklasside kaudu tagatakse olemi toimetamise info hoidmine. Autogenereeritud olemite alamklassides kirjutatakse üle omaduste settereid, et salvestada omaduste muutmise infot. Korduva omaduse puhul kasutatakse spetsialiseeritud *ChangesAwareCollection* liidese teostusklasse, mille abil saab teada, et antud omaduse kollektsioontüüpi väärtus oli toimetatud (ilma väärtuse üleskirjutamiseta) ehk see vajab ka uuendamispäringusse lisamist. Väiksemate toimetamiste puhul (tavaline väärtuse lisamine või eemaldamine) saavad ainult kollektsiooni väärtuse uuendused objekti uuendamise päringusse, mis on tagarakenduse poolt toetatud. Genereeritud *ChangesAwareEntity* klassi näidis on toodud järgneval joonisel (vt Joonis 8).

```
@Generated("Generated with CasemEntityAnnotationProcessor")
public class ChangesAwareMeeting extends Meeting
  implements ChangesAwareEntity {
  // olemi uuenduste hoidmiseks kasutatakse kujutist
  private final Map<String, PropertyUpdate> updatesMap =
      new HashMap<>();

  @Override
  public Map<String, PropertyUpdate> getUpdatedProperties() {
    // PropertyUpdate hoiab originaal- ning lõppväärtust.
    // Tulemusse saavad ainult päriselt muudetud väärtustega omadused
    return filtered(updatesMap, PropertyUpdate::hasChanges);
  }

  @Override // Setteri üleskirjutamise näidis
  public void setTitle(String title) {
    updatesMap.compute("ams_title", (propId, update) -> {
      if (update == null) {
        var originalValue = super.getTitle();
        update = new PropertyUpdate(propId, originalValue);
      }
      update.setNewValue(title);
      return update;
    });
    super.setTitle(title);
  }
  // teised setterid, jne.
}
```

Joonis 8 Genereeritud *CahngesAwareEntity* olemi alamklassi koodi näidis kommentaaridega.

Lisaks teisendamise funktsionaalsusele saab olemimudeleid kasutada DQL päringute koostamisel: mudelil on samade nimedega avalikud konstantsed väärtused, nagu olemi väljadel, ning väärtusteks on väljadele vastavad omaduste identifikaatorid. Mugavuseks on ka meetod, mis tagastab kõik olemi omadused ning mida saab kasutada terve olemi laadimiseks.

Arvestades sellega, et teistes projekti moodulites enamasti on tegu omaduste kujutisega, on lisatud ka võimalus teisendada kujutist olemisse. Samuti, peab see lubama ka olemi koostamist näiteks JSON (*Javascript Object Notation*) päringust. Olemi mudeli klassi näidis on toodud järgneval joonisel (vt Joonis 9).

```
@Generated("Generated with CasemEntityAnnotationProcessor")
@org.springframework.stereotype.Component
public class MeetingModel extends CasemEntityModel<Meeting> {

  // väli mis sai päritud CasemObjectId klassist
  public static final String id = "r_object_id";
  // Meeting olemiklassi väljad
  public static final String title = "ams_title";
  public static final String repeatingDates = "example_dates";

  private MeetingModel() {
    super(CasemTypes.ams_meeting);
    addPropertyMappings(
        // kõik vajalikud teisendused defineeritakse konstruktoris
    );
  }

  @Override
  public Meeting newEntity() {
    // kui CahngesAwareEntity polnud kasutatud antud olemi mudeliga,
    // meetod tagastaks tavaline Meeting olem.
    return new ChangesAwareMeeting();
  }

  // meetodit saab kasutada DQL päringulausete koostamisel
  @Override
  public List<String> getQueryFields() {
    return List.of(id, title, repeatingDates);
  }

  // avalikud teisendamismeetodid ja teised abimeetodid
  // päritakse CasemEntityModel ülemklassist
}
```

Joonis 9. Genereeritud CaseM olemi mudeli koodi näidis kommentaaridega.

Loodud lahendus testitakse ja dokumenteeritakse põhjalikkult. Lahenduse kompileerimiseks kasutatakse *maven compiler* laiendust [34]. Funktsionaalsuse kompileerimisosa testimiseks kasutatakse Google arendajate poolt tehtud avatud lähtekoodiga teegi *compile-testing* [35]. Lahendus püütakse teha võimalikult universaalseks, kuid see nõuab, et objektide omaduste tüübid oleksid erinevatel organisatsioonidel sama. Autori arvates on lahendus on sobiv kasutamiseks arendatavas haldusmoodulis, kuna veebikoosolekute haldusfunktsionaalsus on seda kasutatavates organisatsioonides väga sarnane ning kasutusel on piiratud objektide andmetüübid ja nende teatud omadused.

5.3 Veebikoosoleku dokumentide genereerimine

Käesolev peatükk ühendab koosolekutega seotud PDF-dokumentide genereerimise eest vastutava funktsionaalsuse refaktoreerimise etapid. Antud üldine refaktoreerimise protsessi osa ei hõlma dokumentide genereerimisega seotud protsesse, nagu vastavate dokumendiobjektide loomine Documentum-is või teiste seotud objektide metaandmete muutmine, isegi kui kasutaja seisukohalt käsitletakse dokumentide loomist ja vastavate objektide muutmist ühe protsessina. Seotud protsesside refaktoreerimine viiakse läbi järgmistes koosoleku haldusfunktsionaalsuse refaktoreerimise etappides.

5.3.1 Olemasoleva teostuse testimine

Esmased refaktoreerimise sammud taustatööde moodulis on keskendunud koosoleku dokumentide genereerimise funktsionaalsuse testimise võimaldamisele.

Koosoleku dokumentide genereerimise protsess tööde moodulis teostab andmetele juurdepääsu, kasutades Documentum-i teekide klasse. Objektidele juurdepääs toimub nende identifikaatori alusel Documentum-i sessioonist. Andmete keerulisemaks pärimiseks, näiteks objektide loendi pärimise puhul, käivitatakse DQL päringud läbi *QueryManager* klassi. *QueryManager* koosneb peamiselt staatilistest meetoditest. Dokumendi mudeli loomise protsessi testimiseks on vajalik võimalus esitada testimisandmed mõlema andmete pärimise meetodi jaoks.

QueryManager klassi meetodite kutsed vahetati uue liidese *QueryExecutor* kasutamisega. *QueryExecutor* vaikimisi teostus kasutab sisemiselt originaalseid *QueryManager* klassi meetodeid; testimisel kasutatud liidese teostus saab tagastada

etteantud testimisandmeid. Documentum sessiooni ja Dokumentum objektide manipuleerimiseks testides kasutataske *mock*-imise tehnikat, mis on tagatud Mockito teegiga [31].

Klasside kapseldust vähendatakse, tehes rohkem meetodeid avalikuks, lubades nende sõltumatult testimist. Kuigi see ei paranda otseselt koodi kvaliteeti, võimaldab see dokumendi mudeli loomise protsessi paindlikumat ja lihtsamat testimist.

Arendatud testimiskomplekt kasutab JUnit 5 [30] ja Mockito teeke. Testimiskomplekt on suunatud toodetud andmete töötlemise testimisele. Andmete pärimise koodi ei testita, kuna see põhineb peamiselt Documentum-i teegi kasutamisel ning saavutatud teste ei ole võimalik taaskasutada.

5.3.2 Andmete laadimis- ja töötlemisloogika jaotamine

Iga dokumendimudeli generaatoriklassi jaoks luuakse vastav teenuseklass, et viia kogu andmete pärimise funktsionaalsus sinna. Kasutatud objektitüüpide (näiteks koosolek või dokument) jaoks luuakse DAO (*Data Access Object*) klassid nende edasiseks kasutamiseks andmeteenuste klassides. DAO klasside kasutamine tagab, et objektide pärimisloogika pole dubleeritud andmeteenuste klasside vahel. Lisaks DAO klassides on kapseldatud Documentum-i teekide klasside ja suurem osa *QueryManager*-i kasutamised. Antud koodi eraldamise lisasamm aitab andmeteenuseid uude moodulisse kergesti migreerida.

Antud refaktoreerimise sammul analüüsitakse põhjalikult ja optimeeritakse DQL päringute täitmise järjestused. Optimeerimised peamiselt hõlmavad päringute täitmiste arvu vähendamist. Näiteks võis olemasolev teostus kasutajate andmeid mitu korda pärida, kui teatud tingimused on täidetud. Refaktoreeritud lahenduses laaditakse kasutajate andmed ühe päringuga, kui iga vajalik kasutajanimi on teada.

5.3.3 Funktsionaalsuse migreerimine

Refaktoreeritud funktsionaalsus migreeritakse edukalt uude moodulisse. Andmeteenuse klassid kohandatakse kasutamiseks uue andmete juurdepääsu kihiga. Testide üleviimine pole sujuv eelnevalt ebapiisava refaktoreerimise tõttu. Teostatud testide komplekt kontrollitakse pärast andmeteenuste ja generaatorklasside eraldamist, kuid generatsioonifunktsionaalsuse testid tuginevad endiselt madalamal tasemel olevate

andmete juurdepääsu simulatsioonile. Testid oleks pidanud ümber kirjutama nii, et genereerimise testid põhineksid uute andmeteenuste kasutamisel.

Sellel refaktoreerimise sammul integreeritakse kasutajaliidese spetsiifilisi funktsioonid refaktoreeritud lahendusse. Näiteks, lisatakse dokumentide genereerimise edenemisprotsessi jälgimine. Samuti tehakse mitmeid täiendavaid refaktoreerimis- ja optimeerimismeetmeid, mis on inspireeritud kasutajaliidese lahendusest. Näiteks, lisatakse kasutajate salvestamine organisatsioonipõhisesse vahemällu. Struktureeritud dokumentide töötlemiseks asendatakse taustatööde mooduli spetsiifiline lahendus spetsialiseeritud ja universaalse teegi (nimetusega *structured_documents*) kasutamisega.

Koosolekudokumentide genereerimise funktsionaalsus disainitakse asünkroonseks seoses sellega, et genereerimise protsessi aeg olenevalt sõltub sisendandmetest (ehk veebikoosoleku arutlusteemade hulgast). Selle saavutamiseks kasutatakse Spring raamistiku *@Async* annotatsiooni ja ülesandetöötlejat (*Task Executor*).

Uue lahenduse veebiliides koosneb kahest meetodist, mis toetavad koosolekute dokumentide asünkroonset genereerimist. Esimene meetod algatab genereerimisprotsessi ning tagastab protsessi unikaalse identifikaatori. Teist meetodit kasutatakse genereeritud PDF dokumendi pärimiseks, kasutades protsessi unikaalset identifikaatorit. Valmis PDF dokumentide sisu ei hoita serveris, vaid PDF dokumendi päringut vahendatakse (*proxying*) PDF generaator teenusele andes kaasa genereeritud dokumendi mudeli (mida saadakse vahemälust protsessi identifikaatori põhjal).

Genereerimisprotsessi oleku kontrollimiseks kasutatakse kahte lähenemist. Esimene on veebiliidese lisameetod, mis võimaldab klientidel pärida genereerimisprotsessi olekut protsessi unikaalse identifikaatori põhjal. Teine lähenemine kasutab sõnumivahetuse lahendust RabbitMQ [26]. Selles lähenemises teenus avaldab genereerimisprotsessi oleku värskendused protsessijärjekorda (*message queue*), mida saab edasi tarbida protsessi algatanud klient.

Sõnumivahetuse lahenduse kasutamine on tõhusam, kuna kliendiprogramm saab kohe teavituse, kui genereerimise protsess on lõppenud (kas edukalt või veaga). Teine lahendus nõuab kliendi poolt teenusele kindlatel ajavahemikel päringute tegemist, et protsessi olekut kontrollida, mis võib põhjustada viivitust. Kuid teist lahendust saab kasutada teenuse administreerimiseks protsessi oleku verifitseerimiseks.

5.3.4 Funktsionaalsuse verifitseerimine

Refaktoreeritud veebikoosolekute genereerimise funktsionaalsuse automaatseks verifitseerimiseks luuakse ajutine abimoodul. Mooduli ainus ülesanne on genereerida veebikoosoleku dokumente, kasutades vana teostust ning refaktoreeritud lahendust, ja võrrelda, et refaktoreeritud lahenduse poolt tehtud dokumendid on sisuliselt võrdsed vana teostustega tehtud dokumentidega. Antud lähenemine on sarnane musta kasti testimisega, mille puhul, et ole teada, kuidas on funktsionaalsus teostatud, vaid selle toimimise korrektsust hinnatakse teatud sisendi ja väljundi põhjal [36].

Mooduli käivitamisel antakse argumendina koosolekute gruppi identifikaatori, ning selle gruppi koosolekute põhjal genereeritakse dokumente kasutades erinevad võimalikud parameetrite kombinatsioonid. Kui refaktoreeritud funktsionaalsusega genereeritud dokumendi sisu ei ole võrdne vana teostuse genereeritud sisuga, sisendparameetrid salvestatakse edasiseks probleemi analüüsimiseks.

Kasutajaliidese ja taustatööde mooduli genereerimise protsessi käivitamiseks verifitseerimismoodulis on antud vastavad koodisõltuvused. See lubab otsest teostuskoodi vastavate klasside ja meetodite kasutamist ning ei vaja jooksvat kasutajaliidest ega taustatöö käivitamist Documentum serveris. Uut refaktoreeritud lahendust kasutatakse abimoodulis nagu reaalses keskkonnas, ehk läbi mikroteenuse veebiliidest ja sõnumivahenduse teenuse kasutamisega.

6 Tulemuste analüüs

Käesoleva bakalaureusetöö peamise tulemusena said peatükkis 2.3 kirjeldatud eesmärgid täidetud:

- refaktoreerimise vajadus sai põhjendatud;
- refaktoreerimise plaan sai kavandatud ja sammude järjekord põhjendatud;
- refaktoreeritud funktsionaalsuse mooduli põhi sai arendatud ja võimalikud abivahendid ja teegid moodulisse lisatud, et tagada parem struktuur refaktoreeritud funktsionaalsusele ja koodi loetavus;
- veebikoosolekute dokumentide genereerimise funktsionaalsus sai kaetud testidega, refaktoreeritud ja migreeritud uude moodulisse ning verifitseeritud.

Teiste haldusfunktsionaalsuse refaktoreerimise etappide täitmist jätkatakse käesoleva töö raamidest väljaspool.

Antud töö raames arendatud CaseM olemi mudeli teek on näidanud potentsiaali ja kasulikkust oma funktsionaalsuses. See toimib tõhusa vahendina tagarakenduse tagasipöördumiste ja olemklasside teisendamisel. Olemi omadused muudetuste salvestamise võimalus tagab, et vastavasesse päringusse edastatakse ainult muudetud omadused, mis vähendab tarbetut andmeedastust ja lihtsustab olemite uuendamise eest vastavat koodi. Kuigi CaseM olemi mudel on saanud positiivset tagasisidet arendajatelt, tuleb veel tõestada, et seda saab kasutada koos teiste projekti moodulitega ja reaalsete kasutajatega keskkondades. Pakutud funktsionaalsus teeb antud teegi paljutõotavaks tööriistaks, mis võib tõhustada arendusprotsessi.

Veebikoosolekute dokumentide genereerimise pärandkoodi testimine osutus keeruliseks ja aeganõudvaks ülesandeks. Funktsionaalsuse keerukuse ja dokumentatsiooni puudumise tõttu oli raske saavutada piisavat testikatvust ja tõhusat testkoodi disaini. Kuid funktsionaalsuse käitumine on pigem mõistetav: genereeritakse PDF-dokument, mille sisu põhineb sisendobjektil (koosolek või arutlusteema põhidokument) ja täiendavatel parameetritel. Seetõttu refaktoreeritud funktsionaalsuse verifitseerimisel saab alati teada,

millised peavad olema antud sisendi jaoks tulemused olemasolevat funktsionaalsust kasutades.

Autori arvates oleks võinud pärandfunktsionaalsuse testimise etapi vahele jätta. Refaktoreeritud funktsionaalsuse korrektsuse tõestamiseks oleks piisanud testimismoodulist, mis on kirjeldatud peatükis 5.3.4. Refaktoreeritud kood on siiski vaja põhjalikult üksustestidega katta, kuid testide teostamine oleks lihtsam uue mooduli keskkonnas. Samuti pärandfunktsionaalsuse üksustestimisel on raske tagada, et testid oleksid kõikehõlmavad; refaktoreeritud funktsionaalsuse verifitseerimine teadaolevate sisendite ja väljundite komplektiga on endiselt vajalik. Kokkuvõtteks võib öelda, et kirjeldatud lähenemine oleks võinud osutuda tõhusamaks ja tulemuslikumaks refaktoreerimise viisiks, mis samamoodi tagaks, et uus lahendus vastab samadele nõuetele ja käitub õigesti.

Teistpoolt vaadates, algselt valitud lähenemisviis (pärandkoodi testimine) tagab funktsionaalsuse sügavama mõistmise, mis omakorda võib aidata teha refaktoreeritava koodibaasi jaoks sobivamaid arhitektuuri- ja disainivalikuid. Kokkuvõttes on raske öelda, kumb lähenemistest oleks tõhusam, kuid veebikoosolekute dokumentide genereerimise funktsionaalsuse puhul autori arvates refaktoreerimise protsess oleks efektiivsem ilma pärandkoodi üksuste testimiseta.

Tööga seotud edasiste plaanide hulgas on järgmised sammud:

- Jätkata veebikoosolekute haldusfunktsionaalsuse refaktoreerimise protsessi kavandatud plaani järgi.
- Edasi arendada CaseM olemi mudeli teeki, laiendades toetatud omaduste andmetüüpide valikuid. Uurida võimalusi lahenduse kasutamiseks ka teistes projekti moodulites, kus see oleks asjakohalik ning kasulik.
- Edasi uurida võimalust ning asjakohalisust DQL päringute ehitamise lahenduse arendamiseks.

7 Kokkuvõte

Antud lõputöö eesmärk on CaseM projekti veebikoosoleku haldusfunktsionaalsuse refaktoreerimise läbiviimine. Refaktoreerimise protsessi skoop hõlmab olemasoleva funktsionaalsuse koodi uurimist ja probleemide tuvastamist, refaktoreerimise plaani kavandamist ning osalist läbiviimist. Kuna kogu veebikoosolekute haldusfunktsionaalsuse refaktoreerimine ja testidega katmine on töömahukas ja aeganõudev piirdutakse töö praktilises osas haldusmooduli põhja arendamise, veebikoosolekute dokumentide genereerimise funktsionaalsuse refaktoreerimisega ja testidega katmisega

Töö analüütilises osas põhjendati funktsionaalsuse refaktoreerimise aktuaalsust ja kasulikkust, uuriti, millised probleemid esinevad olemasolevas lahenduses ning pakuti välja meetmed antud probleemide parandamiseks. Vastavalt tuvastatud probleemidele ja pakutud lahendusviisidele sõnastati nõuded refaktoreeritud lahendusele ning valiti kasutatud tehnoloogiaid ja lahenduse arhitektuur. Parimaks arhitektuuriks antud funktsionaalsuse jaoks osutus mikroteenuse arhitektuur, lahenduse arendamisel kasutati Spring raamistikku, RabbitMQ sõnumivahetusteenust ja muid abistavaid teeke.

Töö praktilises osas arendati veebikoosolekute haldusfunktsionaalsuse mooduli põhi analüütilise osa käigus valitud arhitektuuri ja tehnoloogiate kasutamisega; koostati refaktoreerimise plaan ja täideti see lõputöös püstitatud mahus.

Töö tulemusena said lõputöö eesmärgid saavutatud: veebikoosolekute haldusfunktsionaalsuse refaktoreerimise vajadus põhjendatud ja töö raames plaanitud funktsionaalsuse refaktoreerimine läbi viidud.

Lõputöö autor jätkab koostatud refaktoreerimisplaani elluviimist oma tööülesannete osana. Uue mooduli arendamisel loodud lahendust olemklasside kirjutamiseks saab edasi arendada ja kasutada ka teistes projekti moodulites ning kirjeldatud lähenemine võib osutada kasulikuks ka teistes pärandsüsteemides.

Kasutatud kirjandus

[1] S. Medijainen, „Tehnilise võla prioritiseerimine SQALE meetodiga ettevõtte näitel,“ 2017. [Võrgumaterjal]. Loetud aadressilt: https://digikogu.taltech.ee/et/Item/46dbb44c-ee55-4606-97a3-fed651979890.

[2] A. Tornhill, Software Design X-Rays, [Online]: O'Reilly Media, 2018.

[3] C. Sterling ja B. Barton, Managing Software Debt: Building for Inevitable Change, [Online]: O'Reilly Media, 2010.

[4] Fujitsu, „Fujitsu Eestis,“ [Võrgumaterjal]. Loetud aadressilt: https://www.fujitsu.com/ee/about. [Kasutatud 17. oktoober 2022].

[5] Fujitsu, „CaseM — Dynamic Information Management in Action,“ [Võrgumaterjal]. Loetud aadressilt: https://www.fujitsu.com/fi/Images/casem.pdf. [Kasutatud 17. oktoober 2022].

[6] M. Trafton, „What is Documentum?,“ [Võrgumaterjal]. Loetud aadressilt: https://argondigital.com/blog/ecm/what-is-documentum/. [Kasutatud 17. oktoober 2022].

[7] M. Feathers, Working Effectively with Legacy Code, [Online]: O'Reilly Media, 2004.

[8] M. Fowler, Refactoring: Improving the Design of Existing Code (Second Edition), [Online]: O'Reilly Media, 2018.

[9] A. Oram ja G. Wilson, Making Software, [Online]: O'Reilly Media, 2010.

[10] AlDanial, „AlDanial/cloc: cloc counts blank lines, comment lines, and physical lines of source code in many programming languages.,“ [Võrgumaterjal]. Loetud aadressilt: https://github.com/AlDanial/cloc. [Kasutatud 22. aprill 2023].

[11] D. Thomas ja A. Hunt, The Pragmatic Programmer: your journey to mastery, 20th Anniversary Edition, 2nd Edition2019, [Online]: O'Reilly Media, 2019.

[12] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, [Online]: O'Reilly Media, 2008.

[13] JetBrains, „Run with coverage | IntelliJ IDEA Documentation,“ [Võrgumaterjal]. Loetud aadressilt: https://www.jetbrains.com/help/idea/running-test-with-coverage.html. [Kasutatud 22. aprill 2023].

[14] Oracle, „Dynamic typing vs. static typing,“ [Võrgumaterjal]. Loetud aadressilt: https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html. [Kasutatud 23 04 2023].

[15] Project Lombok, „Project Lombok,“ [Võrgumaterjal]. Loetud aadressilt: https://projectlombok.org/. [Kasutatud 24. aprill 2023].

[16] A. Stec, „REST vs SOAP,“ [Võrgumaterjal]. Loetud aadressilt: https://www.baeldung.com/cs/rest-vs-soap. [Kasutatud 22. aprill 2023].

[17] C. Richardson, „Microservice Architecture pattern,“ [Võrgumaterjal]. Loetud aadressilt: https://microservices.io/patterns/microservices.html. [Kasutatud 10. november 2022].

[18] C. Richardson, „Pattern: Database per service,“ [Võrgumaterjal]. Loetud aadressilt: https://microservices.io/patterns/data/database-per-service.html. [Kasutatud 22. aprill 2023].

[19] V. Andrejev, „Spring MVC-st Spring WebFlux-le ülemineku otstarbekuse analüüs: juhtumiuuring,“ Tallinna Tehnikaülikool, 2021. [Võrgumaterjal]. Loetud aadressilt: https://digikogu.taltech.ee/et/Item/2abee630-9782-4865-9463-87d7a7a3b82a.

[20] Spring.io, „Web on Reactive Stack (spring.io),“ [Võrgumaterjal]. Loetud aadressilt: https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html. [Kasutatud 13. november 2022].

[21] K. Mäeots, „REST API mikroteenuse disain ja arendamine Micronaut raamistikus ettevõtte näitel,“ Tallinna Tehnikaülikool, 20221. [Võrgumaterjal]. Loetud aadressilt: https://digikogu.taltech.ee/et/Item/4a1e883d-256f-489d-a4ca-86f43f2d8de6.

[22] Micronaut, „Micronaut Framework,“ [Võrgumaterjal]. Loetud aadressilt: https://micronaut.io/. [Kasutatud 13. november 2022].

[23] T. Tymoshchuck, „Choosing the Right Microservices Framework,“ [Võrgumaterjal]. Loetud aadressilt: https://hackernoon.com/choosing-the-right-microservices-framework-gp1235dw. [Kasutatud 13. november 2022].

[24] I. López, „Developing Micronaut applications with IntelliJ IDEA,“ [Võrgumaterjal]. Loetud aadressilt: https://blog.jetbrains.com/idea/2020/09/webinar-summary-developing-micronaut-applications-with-intellij-idea/. [Kasutatud 13. november 2022].

[25] Micronaut, „Micronaut for Spring,“ [Võrgumaterjal]. Loetud aadressilt: https://micronaut-projects.github.io/micronaut-spring/latest/guide/. [Kasutatud 13. november 2022].

[26] RabbitMQ, „RabbitMQ,“ [Võrgumaterjal]. Loetud aadressilt: https://www.rabbitmq.com/. [Kasutatud 23. aprill 2023].

[27] MapStruct, „MapStruct — Java bean mappings, the easy way!,“ [Võrgumaterjal]. Loetud aadressilt: https://mapstruct.org/. [Kasutatud 24. aprill 2023].

[28] SmartBear Software, „OpenAPI Specification,“ [Võrgumaterjal]. Loetud aadressilt: https://swagger.io/specification/. [Kasutatud 10. mai 2023].

[29] SmartBear Software, „OpenAPI Specification and Swagger,“ [Võrgumaterjal]. Loetud aadressilt: https://swagger.io/solutions/getting-started-with-oas/. [Kasutatud 10. mai 2023].

[30] The JUnit Team, „JUnit 5 User Guide,“ [Võrgumaterjal]. Loetud aadressilt: https://junit.org/junit5/docs/current/user-guide/. [Kasutatud 23. aprill 2023].

[31] Mockito, „Mockito Wiki,“ [Võrgumaterjal]. Loetud aadressilt: https://github.com/mockito/mockito/wiki. [Kasutatud 23. aprill 2023].

[32] The Querydsl Team, „Querydsl Reference Guide,“ [Võrgumaterjal]. Loetud aadressilt: http://querydsl.com/static/querydsl/5.0.0/reference/html_single/. [Kasutatud 13. november 2022].

[33] JetBrains, „Annotations | IntelliJ IDEA Documentation,“ [Võrgumaterjal]. Loetud aadressilt: https://www.jetbrains.com/help/idea/annotating-source-code.html. [Kasutatud 14. aprill 2023].

[34] Apache, „Apache Maven Compiler Plugin – Introduction,“ [Võrgumaterjal]. Loetud aadressilt: https://maven.apache.org/plugins/maven-compiler-plugin/. [Kasutatud 14. aprill 2023].

[35] Google, „google/compile-testing: Testing tools for javac and annotation processors,“ [Võrgumaterjal]. Loetud aadressilt: https://github.com/google/compile-testing. [Kasutatud 14. aprill 2023].

[36] R. Patton, Software Testing, Second Edition, [Online]: O'Reilly Media, 2005.

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

Mina, Juri Geiman

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Veebikoosolekute haldusfunktsionaalsuse refaktoreerimine AS-i Fujitsu Estonia näitel“, mille juhendaja on Kristiina Hakk
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

15.05.2023

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