TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Kristjan Hendrik Küngas

Teenuste testimise hõlbustamine Eesti Töötukassa infosüsteemi EMPIS näitel

Bakalaureusetöö (9 EAP)

Juhendaja: Vambola Leping, MSc

Tartu 2021

Teenuste testimise hõlbustamine Eesti Töötukassa infosüsteemi EMPIS näitel

Lühikokkuvõte:
Töös tutvustatakse Eesti Töötukassa infosüsteemi EMPIS ja kirjeldatakse, kuidas süsteemi funktsionaalsuse väiksemateks mooduliteks tükeldamine on suurendanud nendevahelist suhtlust ja vajadust seda testida. Töö eesmärk on kirjeldada süsteemi teenuste testimise jaoks nõudeid, näidata varasemalt kasutatud vahendeid ja luua uus veebirakendus, mis aitab testijatel teenuseid testida, võimaldades teenuste päringuid ja vastuseid seadistada. Töö lõpus antakse ülevaade loodud testimisrakenduse tagasiside kohta.

Võtmesõnad:
Testimine, Vastavustestimine, Teenus, HTTP, Veebirakendus, Angular, Quarkus

CERCS: P170 Arvutiteadus, P175 Informaatika

Simplifying testing of services in Estonian Employment Information System

Abstract:
This bachelor thesis introduces Estonian Employment Information System EMPIS and describes how separating business logic into smaller modules has increased the amount of traffic between services, and the need to test them. The purpose of thesis is to describe requirements for testing of EMPIS services, to show currently used solutions and to create a new web application for simplifying testing of services by allowing user to create requests and mock responses. At the end of the thesis feedback from users is presented.

Keywords:
Testing, Contract testing, Service, HTTP, Web application, Angular, Quarkus

CERCS: P170 Computer science, P175 Informatics

Sisukord

Sissejuhatus

Tarkvaraarenduse maailm on pidevalt muutumas. Kui kümmekond aastat tagasi ehitati infosüsteeme monoliitse arhitektuuriga ehk ühe suure rakendusena, siis tänapäeval liigutakse mikroteenuse arhitektuuri suunas, kus infosüsteem on jaotatud väiksemateks mooduliteks. Selline tükeldamine võimaldab süsteemi funktsionaalsust loogiliselt lahus hoida ja väiksemad osad pakuvad suuremat jõudlust. Samas on moodulid üksteisega järjest tihedamas sõltuvuses, mis teeb nende testimise keerulisemaks.

Nortal on Eesti Töötukassa infosüsteemi EMPIS arendanud alates aastast 2008 ning algselt ehitati see üles kasutades monoliitset arhitektuuri. Infosüsteemi pideva kasvamise tõttu on liigutud alates aastast 2018 mikroteenuste arhitektuuri poole, kus äriloogilised osad on üksteisest selgelt eristatud. See on aga märgatavalt suurendanud infosüsteemi suhtlemist üle teenuste, mis omakorda nõuab rohkem komponentidevahelise integratsiooni testimist. Selliseks testimiseks vajalikud tööriistad on küll olemas, kuid on puudulikud, pole kasutajale mugavad või ei sobi projektis kasutamiseks.

Töö eesmärk on dokumenteerida teenuste testimise jaoks nõuded EMPIS-e projektis, näidata vahendeid, mida testimiseks kasutatakse, ja luua uus lahendus, mis sobiks paremini kokku projekti töökorraldusega, ning näidata loodud veebirakenduse funktsionaalsust, arhitektuuri ja kirjeldada kasutajate tagasisidet.

Esimeses peatükis kirjeldatakse Eesti Töötukassa infosüsteemi EMPIS, Nortali osa selle arendamises ja näidatakse, kuidas süsteemi arhitektuur on aja jooksul muutnud. Seejärel kirjeldatakse teenuste testimise metoodikat ja kuidas süsteemi kasvamine on testimist keerulisemaks muutnud. Teises peatükis kirjeldatakse teenuste testimise nõudeid EMPIS-e projektis, tuuakse välja kasutatavad lahendused ning kirjeldatakse kogemust nendega. Kolmandas peatükis näidatakse töö raames loodud veebirakendust, mis on kohandatud teises peatükis kirjeldatud nõuetele. Samuti kirjeldatakse loodud veebirakenduse arhitektuuri ja kasutatud tehnoloogiaid. Neljandas peatükis tutvustatakse, kuidas rakenduse kohta tagasisidet koguti, arutletakse kasutajate tagasiside ning võimalike edasiarenduste üle.

1. Teenuste testimine Eesti Töötukassa infosüsteemis EMPIS

Järgnevas peatükis kirjeldatakse Eesti Töötukassa infosüsteemi, Nortali osa selle arendamises ja tuuakse välja, kuidas süsteemi arhitektuur on aja jooksul muutnud. Seejärel kirjeldatakse teenuste testimise metoodikat ning tuuakse välja, kuidas süsteemi kasvamine on testimist keerulisemaks muutnud.

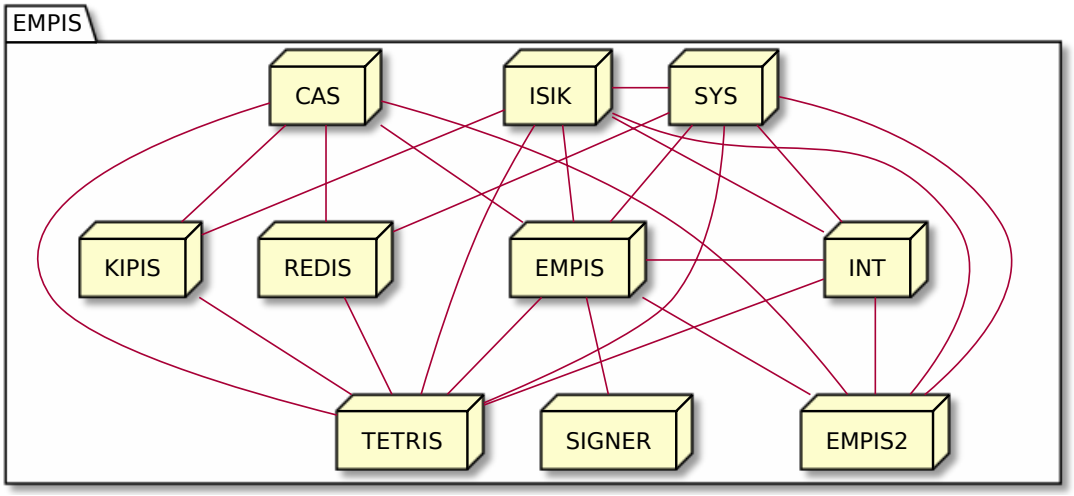
1.1. Eesti Töötukassa infosüsteem EMPIS

Eesti Töötukassa (edaspidi töötukassa) infosüsteem EMPIS (ingl Estonian Employment Information System) ehk Eesti Tööhõive Infosüsteem võeti kasutusele 2009. aastal. 2019. aasta riigihanke dokumentide [1] järgi on selle eesmärgiks hõlbustada rakendust kasutavate konsultantide ja menetlejate tööd, süstematiseerides nende tööprotsesse. Algselt loodi infosüsteem, et toime tulla märkimisväärselt suurenenud töötute arvuga, kui 2008. aasta algusest 2009. aasta lõpuks kasvas see 20 000-lt ligi 100 000-ni. Aastal 2009 avaldatud uudises [2] on kirjas, et esialgu võimaldas infosüsteem peamiselt hallata töötuna arvelevõtmist, töötutoetuse arvestamist, töötuskindlustushüvitise vastuvõtmist ja töötukassasse pöördumisi. EMPIS võimaldas avaldusi ja otsuseid digitaalselt menetleda ning allkirjastada. Uudisest selgub veel, et infosüsteem oli selleks ajaks liidestatud nelja erineva registriga, kust oli võimalik pärida töötukassasse pöördunud isikute kohta andmeid. Riigihanke dokumentides [1] tuuakse välja, et aja jooksul on toetuste ja teenuste arv, mida töötukassa osutab, vähemalt kahekordistunud. Lisaks töötutele suunatud teenustele aidatakse tööd säilitada ja pakutakse karjäärinõustamist nii vähenenud töövõimega, töötavatele kui ka õpingutel olevatele inimestele. Infosüsteem pakub samuti palju asutusesiseseid tugifunktsioone teenuste ja toetuste osutamiseks ning haldamiseks.

Nortal on EUIF (ingl Estonian Unemployment Insurance Fund) projekti raames arendanud töötukassa infosüsteemi EMPIS aastast 2008 [3]. Riigihanke dokumentides [1] on kirjas, et EMPIS on ehitatud monoliitse rakendusena Spring ning Aranea Web Framework raamistikega. Rakenduse esitlus-, teenus-, äri- ja andmebaasikihid on omavahel tihedalt seotud. Aja jooksul on rakendust järjest täiendatud ja praeguseks on see kasvanud üle miljoni koodirea [1]. Alates aastast 2018 on viidud funktsionaalsust üle väiksemateks rakendusteks ehk mooduliteks, et lihtsustada süsteemi arhitektuuri ning selle arendamist. Infosüsteemi majutuse hanke dokumentidest [4] tuleb välja, et tihedalt seotud kihtide asemel liigutakse mikroteenuste arhitektuuri poole, kus äriloogilised osad on omavahel selgelt eristatud ning nendevaheline suhtlus toimib üle teenuste. Näiteks aastal 2020 loodi Isikuandmete moodul (edaspidi ISIK), mille rakendusse ja andmebaasi viidi isikutega seotud äriloogika ja andmed, et isikute andmed ja nendega seotud tegevused oleksid loogiliselt eraldatud ülejäänud süsteemist.

[1] Repositooriumis väljavõtte alusel, mis on välja toodud lisades oleval joonisel 10

Algselt koosnes töötukassa infosüsteem ühest rakendusest, mis suhtles vaid nelja registriga üle andmevahetuskihi X-tee [2]. 2019. aasta infosüsteemi majutuse hanke dokumentidest [4] selgub, et aastal 2020 koosneb kogu töötukassa infosüsteem 20 moodulist, millest kümme on Nortali arenduses ja ülejäänud koostööpartnerite halduses. Nortali arendatavad moodulid ja nendevaheline suhtlus on välja toodud joonisel 1.



Joonis 1. Nortali hallatavad moodulid

Aastal 2020 suhtleb infosüsteem kokku 21 erineva X-tee andmekoguga [5]. Iga moodul ja register võib koosneda kümnetest teenustest, mida võib kasutada mitu rakendust. See tähendab, et kuigi äriloogika on paremini üksteisest eraldatud, siis erinevate komponentide integratsioon on muutunud märkimisväärselt keerulisemaks [4]. X-tee avalike andmete [5, 6] põhjal kasutas töötukassa infosüsteem 2020. aasta novembris 53 erinevat X-tee teenust, tehes nendesse keskmiselt 63 000 päringut päevas. Rakenduste logide põhjal tehti moodulite vahel samal perioodil keskmiselt 1 878 000 päringut päevas [2].

Äriloogika eraldamine mooduliteks lihtsustab nende testimist, sest ühte osa saab teistest sõltumatult testida. Aastal 2011 on Polina Morozova [7] oma magistritöös kirjeldanud testimist EMPIS-e projekti näitel. Kui äriloogika eraldatus on lihtsustanud äriloogika enda testimist, siis rakenduse tükeldamine suurendab teenuste arvu ja seega ka erinevate komponentide vahelise suhtluse ehk integratsiooni testimist. Eriti keerukaks on muutnud mitut moodulit hõlmavate protsesside testimine. Dmitrii Savchenko [8] on välja toonud, et integratsiooni ja tööprotsesside testimise lihtsustamiseks kasutatakse tööriistu, millega on võimalik teenuste päringute vastused asendada näidisvastustega. Sellist testimismetoodikat nimetatakse ka vastavustestimiseks.

[2] Rakenduste logide väljavõte, mis on toodud välja lisades oleval joonisel 11

1.2. Testimismetoodika ja probleemid teenuste testimisel

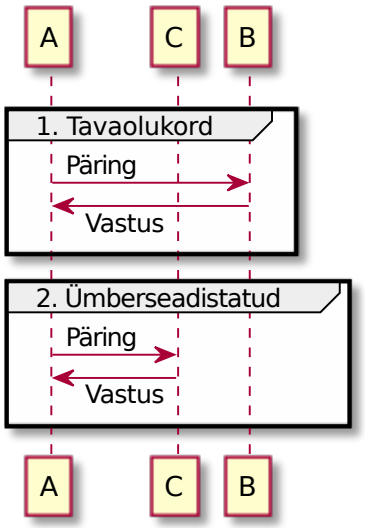
Jyri Lehvä jt [9] on kirjutanud, et vastavus- ehk integratsioonitestimine (ingl *conformance testing*) on testimismetoodika, mida kasutatakse rakendustevahelise suhtlemise kontrollimiseks. Selle metoodika põhieesmärk on teenust pakkuva rakenduse (edaspidi teenusepakkuja) ja teenust tarbiva rakenduse (edaspidi teenusetarbija) vahele luua leping ehk spetsifikatsioon, mis kirjeldab ära teenusetarbija tehtavad päringud ning pakkuja vastused. Kui osapoolte vahel on leping loodud, siis on võimalik nii teenusepakkujat kui ka -tarbijat teineteisest sõltumatult testida. Spetsifikatsioon toimib teenusekihtide eraldajana, mistõttu pole integratsiooni kontrollimiseks vaja kõiki rakendusi ja nende sõltuvusi üles seada. See tähendab ka, et teenusepakkuja ning -tarbija peavad loodud lepingust kinni pidama. Autorid rõhutavad, et kui ühte poolt täiendatakse, siis peavad need muudatused kajastuma ka spetsifikatsioonis ning omakorda teises osapooles. Dmitrii Savchenko [8] on välja toonud, et lepingupõhist testimismetoodikat kasutatakse eelkõige rakendustega, mis on üles ehitatud mikroteenuste arhitektuurile. Autor toob samuti välja, et see võimaldab rakenduste äriloogilisi osi täiendada eraldiseisvalt, ilma et ülejäänud rakendus sellest sõltuks. Kui lisaks äriloogikale täiendatakse pakutavat teenust, siis lepingupõhise testimise kohaselt peab täiendama ka spetsifikatsiooni ja vajadusel teenuse tarbijat [9].

Töötukassa infosüsteemis suhtlevad rakendused üksteisega üle teenuste. X-tee alamsüsteemide lehelt [10] ja 2019. aasta majutuse hanke dokumentidest [4] on näha, et süsteemi moodulid liidestuvad üksteisega peamiselt REST-teenustega, mis koosnevad sünkroonsetest HTTP-päringutest ja vastustest, mille andmete kuju vastab JSON standardile. Joonisel 2 on näha näidet ISIK mooduli isikuandmete päringust ja vastusest. Alternatiivina kasutatakse ka AMQP-liidestust, mis võimaldab rakendustevahelist asünkroonset sõnumitevahetust ja nende sõnumite kuju on samuti JSON-tüüpi. X-tee registritega suhtlus toimub üle SOAP-teenuste, mis sarnaselt REST-teenustele toimub samuti üle HTTP-päringute, kuid andmete kuju vastab XML standardile.

Joonis 2. ISIK mooduli isikuandmete REST päring ja selle vastus

Integratsioonitestimise teeb EMPIS-e projektis keerukaks asjaolu, et alati ei ole olemas täpset spetsifikatsiooni, kuidas teenusele saadetav päring ja tagastatav vastus peaksid välja nägema. REST- ja AMQP-teenuste puhul on olemas analüütiku loodud dokument, mis sisaldab teenuse kirjeldust ja mille järgi peab testija olemasolevate teadmiste põhjal päringu või vastuse looma. SOAP-teenuste puhul on olemas WSDL-spetsifikatsioonid, mille järgi on võimalik genereerida vastuse struktuur, kuid mitte andmeid. Seega alustatakse integratsioonitestimist sellest, et otsitakse rakenduse logist või arendaja käest näidisvastus, mida saab muuta ja seejärel testimisel kasutada. Olemasolevaid lahendusi, millega on võimalik rakendustevahelist suhtlust jälgida, muuta ning näidisvastused välja võtta, kirjeldatakse 2. peatükis.

Integratsioonitestimise efektiivseks ja põhjalikuks läbiviimiseks on vajalik kahe rakenduse, nimetagem neid A ja B, vahel olev liidestus ümber seadistada nii, et tehtav päring liiguks testija ülesseatud testimisrakendusse C, mis vastaks päringule eelseadistatud vastusega, mis oleks sama struktuuriga nagu rakenduse B vastus. Nii on võimalik läbi proovida, kas moodul A sooritab ootuspärase päringu ja suudab eelseadistatud vastust sisse lugeda. Kirjeldatud olukord on näha joonisel 3. Praegu nõuab ümberseadistus konfiguratsioonifaili muudatust moodulis A ja rakenduse taaskäivitamist. Seejärel suunatakse kõik moodulisse B tehtavad päringud ümber testimisrakendusse C. Keeruliseks teeb olukorra see, et protseduuri peab tegema iga rakenduse kohta eraldi ja testija peab oskama konfiguratsioonifaili õigesti muuta. Samuti on rakendus taaskäivitamise ajal kättesaamatu. Kuna testimise ajal suunatakse kõik päringud testimisrakendusse C, siis häirib see ka teiste arendajate ja testijate tööd. Peatükis 3 kirjeldatakse olukorrale lahendust.



Joonis 3. Päringute liikumine moodulite A ja B ning testimisrakenduse C vahel

Töötukassa infosüsteemi testimiseks on projektis üles seatud neli erinevat testkeskkonda, mis on liidestatud vastu X-tee testregistreid ning koostööpartnerite testkeskkondade rakendusi. Mirjam Rauba [11] on välja toonud, et juba aastal 2012 oli kokku kolm testkeskkonda, kus igas testkeskkonnas hoiti arendustsükli seisu või kindlat funktsionaalsust, mida plaaniti kliendile tarnida. Testijad kasutavad ühiseid testkeskkondi, sest kõigi moodulite ülesseadmine, seadistamine ja uuendamine on küllaltki ajakulukas ning ressursimahukas. 2019. aasta majutuse hanke dokumentide [4] järgi on koolituskeskkonna, mis on kõige sarnasem teistele testkeskkondadele, mälunõudlus kuni 64 GB. Testkeskkondade uuendamisega tegeleb paigalduskonveier ning selle haldusega arendajad.

2. Teenuste testimise funktsionaalsed nõuded ja kasutatavad tehnoloogiad

Integratsioonitestimiseks kasutatavaid rakendusi on veebis saadaval mitu. Järgnevates alampeatükkides kirjeldatakse nõudeid, mida sellised vahendid peaksid katma, tuuakse välja neist tuntuimad ja kirjeldatakse kogemust nendega EMPIS-e projektis.

2.1. Teenuste testimise funktsionaalsed nõuded EMPIS-e projektis

Järgnevalt on kirjeldatud funktsionaalsed nõuded EMPIS-e projekti jaoks, mida integratsioonitestimiseks kasutatavad rakendused peavad täitma.

Rakenduses peab olema võimalik seadistada teenuseid, mida kasutaja testima hakkab. See tähendab, et kasutajal peab olema võimalus luua uus teenus, mis on või veel ei ole päriselt olemas, sest moodulite testimine võib alata juba siis, kui kõik osapooled ei ole oma muudatusi veel avalikuks teinud.

Rakenduses peab olema võimalik sooritada HTTP-päringuid. Täpsemalt peavad olema toetatud nii REST- kui ka SOAP-päringud koos manustega. REST-päringute puhul peab saama määrata meetodit, aadressi, selle parameetreid, päiseid ja keha, mis on kas JSON-tüüpi või vabavormis tekst. SOAP-päringuid kasutatakse X-teega liidestamisel ning nende puhul peab saama määrata turvaserveri aadressi, päringu sooritaja asutust ja isikukoodi, päringule vastaja asutust ja teenust ning keha peab olema XML-tüüpi. Lisaks sellele võivad SOAP-päringud sisaldada manuseid, mille olemasolul peab keha vastama MIME standardile.

Rakenduses peab olema võimalik vastata HTTP-päringutele. Täpsemalt peavad olema toetatud nii REST- kui ka SOAP-päringutele vastamine koos manustega. Nii REST- kui ka SOAP-päringute puhul peab saama määrata vastuse staatust, reguleerida vastamise viivitust või jätta päringule üldse vastamata, et oleks võimalik läbi proovida erinevaid veaolukordi. REST-vastuste puhul peab saama määrata staatust, päiseid ning keha, mis on kas JSON-tüüpi või vabavormis tekst. SOAP vastuste korral peab saama määrata staatust ja keha, mis on XML-tüüpi. Manuste korral peab keha vastama MIME standardile.

Rakenduses peab olema võimalik sissetulevaid päringuid edasi suunata ja saadud vastusega esialgsele päringule vastata. Integratsioonitestimisel võib tekkida olukord, kus väline rakendus asendatakse testimisrakendusega, mis koosneb kasutaja seadistatud vastustest. Kasutaja ei pruugi korraga testida ega seadistada kõiki teenuseid, mis välises rakenduses on, seega kui nende pihta tehakse päringuid, siis tuleks vastata päris vastustega. Seega on oluline, et päringud teenustesse, millega kasutaja parasjagu ei tegele, suunataks edasi välisesse rakendusse.

Rakendus peab oma tegevused logima. Kuna rakendust kasutatakse integratsioonitestimiseks, siis on oluline, et see logiks detailselt, millised päringud tulid sisse ja liikusid välja. Logis peavad kajastuma eelnevates punktides kirjeldatud andmed. Lisaks sellele

kasutavad töötukassa rakendused vabavaralist jälitamistarkvara Zipkin, mille jälitamisidentifikaatorid peavad nii logist välja tulema kui ka edasisaadetavatele päringutele kaasa minema. Töötukassa rakendused logivad oma tegevused tsentraalsesse logimishaldurisse Graylog ning kasutajamugavuse eesmärgil peaks seda tegema ka testimisrakendus.

Rakendus peab haldama kasutajate loodud seadistusi. Kasutaja võib ühe tööprotsessi testimiseks seadistada vastuseid mitmele rakendusele ja teenusele, seega need peavad olema kergesti kättesaadavad ja hallatavad.

Rakendust peab saama kasutada isikupõhiselt. Kui kasutaja on testkeskkonnas välise rakenduse asendanud testimisrakendusega, siis see ei tohi mõjutada teiste kasutajate tööd testkeskkonnas.

Integratsioontestimise funktsionaalsust peab olema võimalik kiiresti sisse ja välja lülitada. Praegu nõuab testkeskkonnas ühes moodulis välise rakenduse asendamine testimisrakendusega konfiguratsioonifaili muudatusi, nende rakendamist ja mooduli restarti, mis on keerukas ning ajakulukas.

Rakendus peab võimaldama kasutajal seadistada uusi vastuseid läbikäinud liikluse põhjal. Kui kasutaja ei ole vastust seadistanud, siis testimisrakendus peab päringu edasi suunama välisesse rakendusse ja sealt saadud vastusega sissetulnud päringule vastama. Kasutajal peab olema võimalus sellise sisendi ja väljundi järgi seadistada endale uus päring ning vastus, muutes nendes olevaid parameetreid vastavalt vajadusele. Olemasoleva näite järgi vastuse seadistamine on ohutum kui selle nullist loomine, sest näites on näha, millisel kujul andmed välja näevad.

Rakendus peab suutma pakkuda näidispäringuid ja -vastuseid teenustele, mille päringud on varasemalt juba läbi liikunud või mida on teised kasutajad salvestatud. Kasutajal ei ole alati võimalust testkeskkonnas teha läbi näidispäringut, mille põhjal endale vastus seadistada. Sel juhul peab kasutaja saama vastuse seadistada nullist käsitsi, kuid ta ei pruugi teenuse dokumentatsiooni põhjal teada, kuidas täpselt vastus kokku panna. Rakendus peab võimaluse korral pakkuma näidisvastust, mis tooks välja, mis parameetrid ning väljad on vastuses olemas ja millisel kujul need peaksid olema.

2.2. Kasutatavad tehnoloogiad

EMPIS-e projektis on teenuste testimiseks kasutatud kahte avalikku ja kahte projektisisest vahendit. Nende eesmärk on eelnevalt kirjeldatud probleemi lahendada, kuid kõigil on puudujääke ning need ei kata täielikult vajalikke nõudeid.

2.2.1. SoapUI

Aastal 2005 loodud SoapUI [12] on aktiivselt arenduses ning sellest on saadaval vabavaraline versioon. See on EMPIS-e projektis kõige tuntum tööriist, millega on võimalik REST- ja SOAP-teenuste pihta päringuid teha ja nendele eelseadistatud vastustega vastata. Eelkõige kasutatakse seda X-tee teenuste puhul, sest sellega on võimalik päringute

ja vastusete tüübid genereerida WSDL spetsifikatsioonide abil. SoapUI eeliseks on see, et see on väga põhjalikult seadistatav ja vanematel testijatel on selle kasutuskogemus olemas. Rakenduse kitsaskohaks on sisseelamine ehk uued testijal on seda keeruline kasutusele võtta. Samuti on miinuseks see, et iga testija käitab rakendust omas masinas, mistõttu jagatakse näidispäringuid ja -vastuseid üksteisele käsitsi. Aja jooksul on leitud rakendusest vigu, mistõttu on testijad olnud sunnitud alternatiivseid tööriistu kasutama.

2.2.2. Postman

Postman [13] loodi aastal 2012, on aktiivselt arenduses ning see on suletud lähtekoodiga. Postman on loodud eelkõige REST-päringute sooritamiseks. EMPIS-e projektis on see populaarne, sest seda on lihtsam kasutada kui SoapUI-d. Selle kasutamise kitsaskohaks on see, et testijad jagavad näidispäringuid üksteise vahel käsitsi. Postman võimaldab näidispäringuid üksteisega jagada ja teenustele näidisvastustega vastata, kuid see on tasuline funktsionaalsus. Samuti pole võimalik Postmaniga teenustele vastamist kohalikus masinas üles seada, vaid päringud suunatakse läbi nende enda serveri. Selline lahendus ei ole EMPIS-e projektis sobilik, sest vastused võivad sisaldada privaatseid andmeid.

2.2.3. Rest-mock

Rest-mock [14] on aastal 2018 töö autori loodud rakendus, mis on mõeldud teenuste päringutele näidisvastustega vastamiseks. Erinevalt eelnevalt toodud rakendustest ei ole sellel kasutajaliidest, vaid rakenduse konfigureerimine koosneb JSON konfiguratsioonide ülesseadmisest. See rakendus on eelkõige mõeldud katma Postmani puuduseid. Erinevalt eelmistest tööriistadest on sellel näidisvastused koodirepositooriumis üleval. Rakenduse peamiseks ohukohaks on selle töökindlus, sest seda ei ole niipalju testitud kui eelnevalt väljatoodud tööriistu.

2.2.4. TK-mocktool

TK-mocktool on aastal 2019 testija loodud EMPIS-e projektisisene rakendus, millega on võimalik teenuste päringutele seadistatud vastustega vastata. See on eelkõige mõeldud X-tee teenuste testimiseks asendamaks eelnevalt väljatoodud SoapUI lahendust, kuid sellega on võimalik ka REST-päringutele vastata. Rakendus jookseb kasutaja enda masinas, sellel on kasutajaliides ning näidisvastused on koodirepositooriumis üleval. Rakenduse kasutajaliideses on võimalik eelseadistatud teenustele uusi vastuseid luua ja olemasolevaid muuta, kuid täpsemalt päringute parameetreid muuta ei saa. Samuti ei ole võimalik päringuid teenuste pihta teha, seega ei asenda see Postmani funktsionaalsust.

3. Veebirakenduse ülevaade ja arhitektuur

Järgnevas peatükis kirjeldatakse töö raames loodava veebirakenduse eesmärki, näidatakse kasutajaliidese ja serveripoolset funktsionaalsust, töövooge, kuvasid ning arhitektuuri. Samuti tuuakse välja mittefunktsionaalsed nõuded rakendusele ning kirjeldatakse selle majutust testkeskkonnas.

3.1. Eesmärk

Töö raames loodava veebirakenduse eesmärk on lahendada 1. peatükis kirjeldatud probleemi, kus töötukassa infosüsteemi viiakse järk-järgult monoliitselt arhitektuurilt mikroteenuste arhitektuurile, mis on suurendanud märkimisväärselt moodulitevahelist suhtlust ja vajadust nende integratsiooni testida. Veebirakenduse arendusel on lähtutud 2. peatükis kirjeldatud nõuetest. Kui eelnevalt kirjeldatud lahendused katsid ainult osaliselt EMPIS-e projektis teenuste testimise jaoks vajalikku funktsionaalsust, siis loodav rakendus on kohandatud vastavalt nõuetele. Kuigi loodav veebirakendus asendab olemasolevaid vahendeid, ei saa eeldada, et kasutajad asuvad kohe uut lahendust kasutama. Seega ei tohi selle kasutusele võtmine takistada varasemate tööriistade kasutamist, vaid peab pakkuma mugavamat ja kiiremat kasutajakogemust võrreldes varasemate vahenditega.

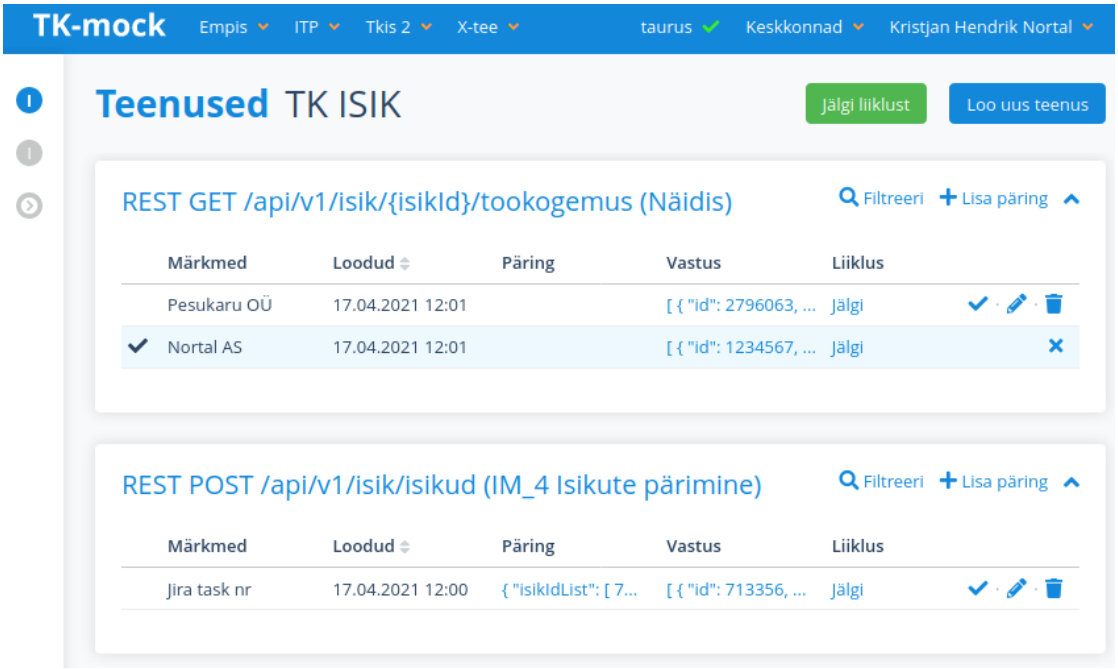
3.2. Kasutajaliidese funktsionaalsuse kirjeldus

Kasutajaliides sisaldab järgmist funktsionaalsust:

Rakenduse kasutamine on isikupõhine. Ühe kasutaja tegevused ei tohi teisi segada ega nende tööd kuidagi häirida. See tähendab, et iga testija või arendaja peab olema testkeskkonda sisse logitud oma kasutajakontoga, mida ei tohi omavahel jagada. Seega suudab rakendus kasutajaid autentida läbi testkeskkonna tsentraalse autentimisteenuse CAS. Erandiks on süsteemne kasutaja, mida töötukassa rakendused kasutavad näiteks taustatöödel päringute sooritamiseks. Selliste olukordade testimise jaoks peab olema võimalik sisse logida ka süsteemse kasutajaga.

Rakenduse funktsionaalsust on võimalik isikupõhiselt sisse ja välja lülitada. Rakenduse kasutamise ajal suunatakse kõik HTTP-päringud läbi serveri, mis võib teha liikluse märgatavalt aeglasemaks või tekitada vigu, mida päriselt ei tohiks tekkida. Seega peab olema võimalus päringute saatmine rakendusse välja lülitada.

Teenused on grupeeritud väliste süsteemide ning X-tee registrite põhiselt. Kui kasutaja testib mõnda funktsionaalsust töötukassa rakenduses, siis analüütiku koostatud dokumentatsioonist tuleb välja, milliste rakenduste teenuseid seal kasutatakse. Seega on päringud jaotatud rakenduste järgi, et neid oleks kerge leida. Rakendused on omakorda jaotatud koostööpartnerite järgi, kes neid arendavad.



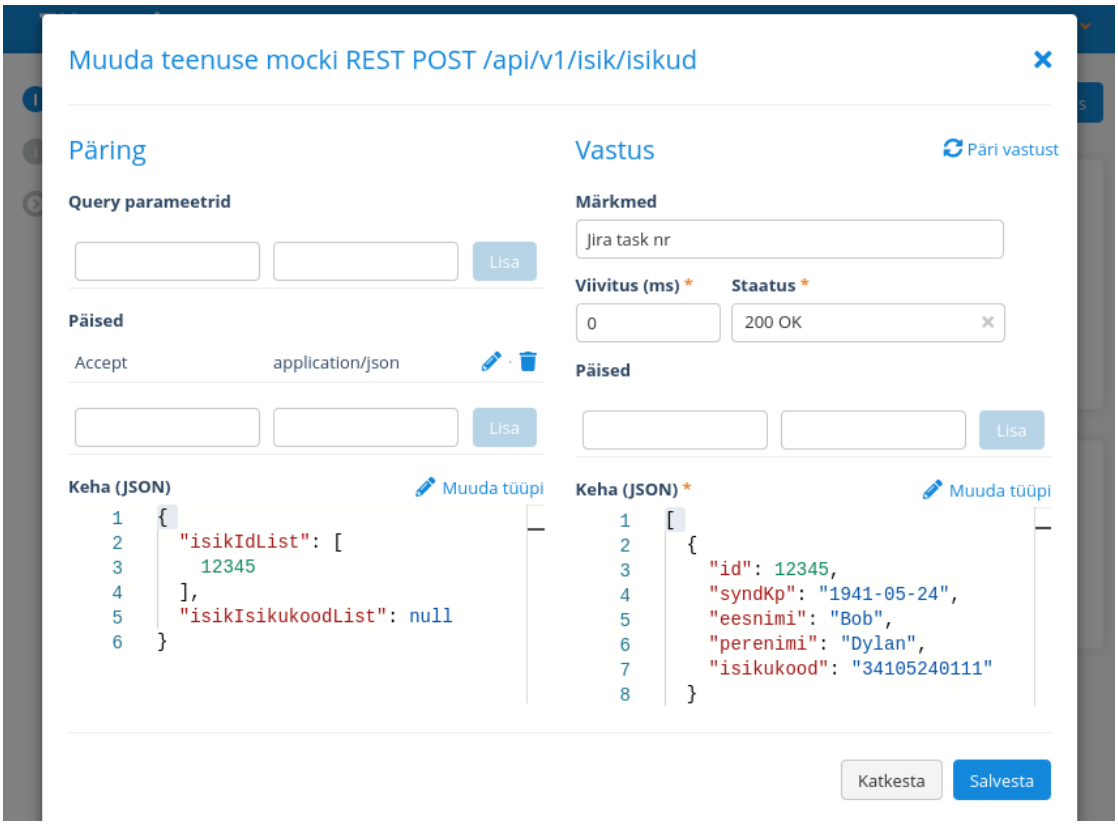
Joonis 4. Teenuste kuva

Päringute ja vastuste seadistamine on grupeeritud teenusepõhiselt. Ühel rakendusel võib olla palju teenuseid ja neid peab saama ükshaaval seadistada. Joonisel 4 on toodud näide teenuste kuvast. Teenusel on selgelt eristav tüüp, HTTP-meetod, aadress ja kasutaja saab lisada enda märkmeid. Kui kasutaja on seadistanud mitu teenust, siis saab neid filtreerida eelnevalt kirjeldatud tunnuste järgi. Kasutaja saab teenuseid juurde lisada, muuta ja kustutada, sest need ei ole fikseeritud, vaid võivad igal hetkel muutuda. Rakendusse saab neid lisada ka enne seda, kui need on päriselt valmis arendatud. Uue teenuse lisamisel ning aadressi valimisel pakub rakendus kasutajale juba teadaolevaid aadresse, millega on võimalik ära hoida väärtuste valesti sisestamist.

Rakenduses saab teenustele seadistada vastuseid, millega rakendus teenuse sissetulevale päringule vastab. Neid saab lisada, muuta ning kustutada. Seadistatud vastust saab sisse ja välja lülitada. Kui vastus on välja lülitatud, saadab rakendus sissetuleva päringu edasi välisesse rakendusse ning vastab sissetulevale päringule sealt saadud vastusega. Ühel teenusel võib korraga olla mitu seadistatud vastust, et kasutaja saaks vastuseid kergelt vahetada. Vastuse juures kuvatakse kasutaja sisestatud märkmeid, loomishetke, lühidalt eeldatava päringu ja vastuse keha. Samuti on võimalik vaadata rakendusest läbi käinud liiklust ühe vastuse kohta.

Vastuse seadistamisel saab kirjeldada sellele eelnevat päringut. Päringul saab seadistada HTTP-aadressi parameetreid, päiseid, keha tüüpi ja selle sisu. Lisaks päringu kirjeldamisele saab vastuse seadistamisel kirjeldada ka vastust ennast. Joonisel 5 on näha näidet päringu ja vastuse seadistamisest. Kasutaja saab vastusele määrata enda märkmeid,

viivitust, HTTP-staatust, -päiseid, keha tüübi ja selle sisu. Nii päringu kui ka vastuse keha tüübiks võib olla vabavormis tekst, JSON või XML ning keha sisu korrektsust valideeritakse vastavalt valitud keha tüübile. Uue päringu ja vastuse loomisel proovib rakendus need automaatselt ära seadistada juhul, kui rakendus on teistel kasutajatel juba näinud sama teenuse liiklust. Kui vähemalt pooltel nähtud päringutest või vastustest on mingi parameetri võti samasuguse väärtusega, täidetakse see automaatselt kasutaja eest ära. Samuti on kasutajal võimalik vastuse seadistamise lihtsustamiseks teha päring välisesse rakendusse, saada sealt näidisvastus päris andmetega ning selle põhjal soovitud vastus seadistada.



Joonis 5. Modaalis päringu ja vastuse seadistamise

Kasutajaliideses on näha ka senine liiklus. Seda näeb nii üle ühe rakenduse, ühe teenuse kui ka ühe seadistatud vastuse. Kuna rakendusest võib palju päringuid läbi käia, siis kuvatakse need nimekirjana ning on filtreeritavad HTTP-aadressi, -staatuse, päringu ja vastuse keha ning kasutaja enda märkmete järgi. Samuti peab iga läbikäinud päringu kohta nägema detaile samasugusel kujul nagu vastuse seadistamisel. Läbi käinud päringu järgi saab seadistada uut vastust, millega rakendus järgmistele päringutele vastaks.

3.3. Serveri funktsionaalsuse kirjeldus

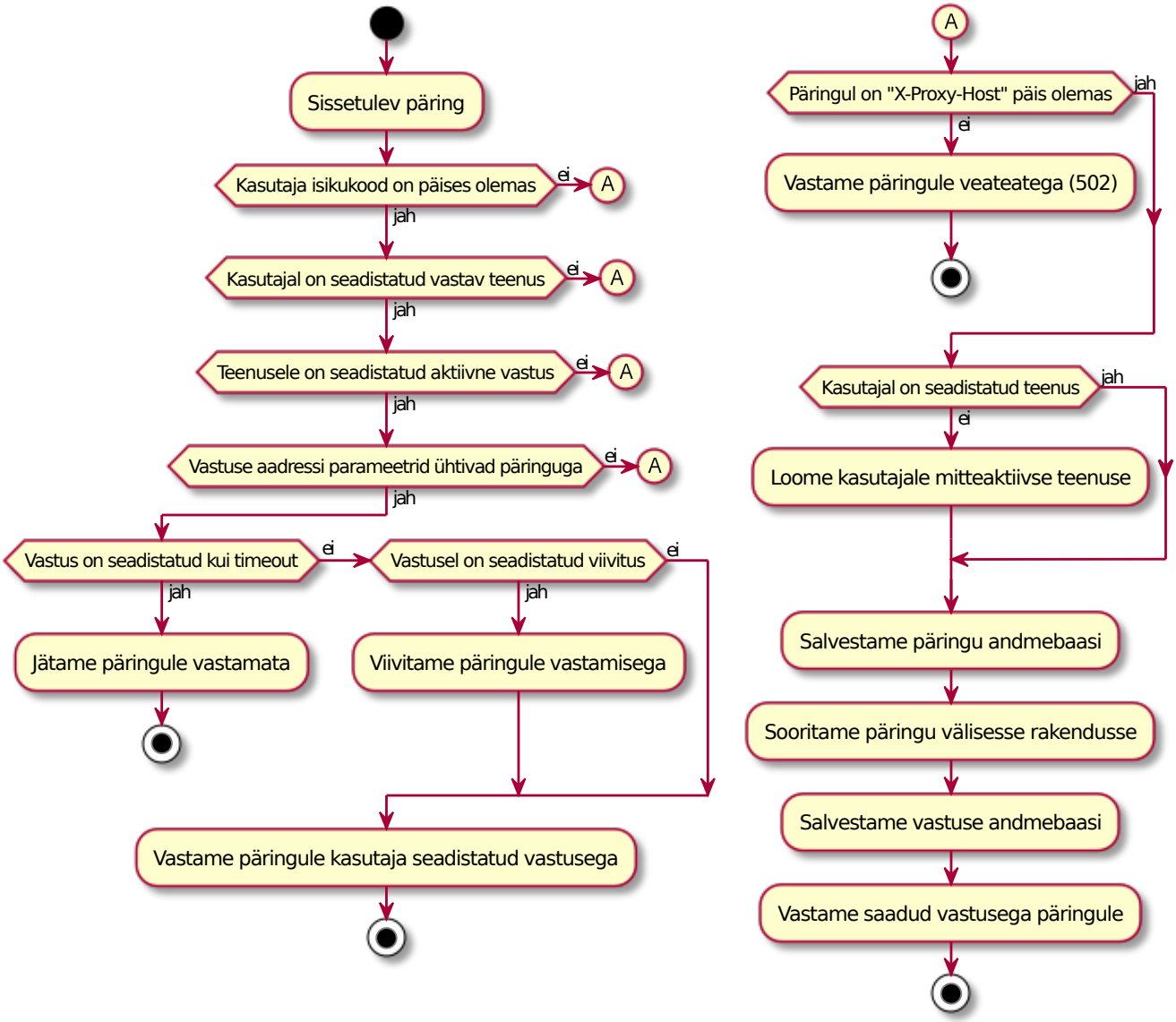
Lisaks kasutajaliidesele sisaldab rakenduse serveri pool järgnevat funktsionaalsust:

Server võimaldab kasutajal ennast autentida läbi tsentraalse autentimisteenuse CAS. See tähendab, et testimisrakendusse ei ole vaja eraldi sisse logida, kui kasutaja on eelnevalt mõnda töötukassa rakendust külastanud. Selline lahendus sarnaneb suures osas OAuth standardile. Autentimise jaoks suunatakse kasutaja CAS-i sisselogimislehele, kus sisse logides suunatakse kasutaja tagasi rakendusse koos piletiga, mille abil saab server CAS-ist kontrollida, kas kasutaja on edukalt sisse loginud. Sellise sisselogimise eeliseks on, et kasutaja ei pea rakendusse oma parooli sisestama, vaid sellega tegeleb CAS ise. Erandiks on süsteemne kasutaja, mille saab kasutajaliidese kaudu valida alles peale tavakasutajana sisselogimist. Kõik serveri teenused, mis pole seotud kasutaja autentimisega, nõuavad kasutaja eelnevat autentimist. Autentimisel päritakse kasutaja nimi töötukassa isikute moodulist, kus hoitakse isikute andmeid.

Server võimaldab testkeskkonna rakenduste päringute suunamist tema pihta sisse ja välja lülitada. Sisselülitamiseks suhtleb rakendus töötukassa süsteemse mooduliga ning määrab süsteemse parameetri *TEST_REROUTE_isikukood* väärtuseks rakenduse enda aadressi. Süsteemne moodul edastab muudatused töötukassa rakendustele üle Hazelcasti klastri, mille abil need tuvastavad, kas antud isiku päringud on vaja suunata ümber testimisrakendusse. Funktsionaalsuse väljalülitamise jaoks määratakse vastav parameeter kehtetuks.

Serveri kaudu on võimalik nii teenuseid kui ka kasutaja seadistatud päringuid ja vastuseid leida, lisada, muuta ja kustutada andmebaasist. Kõik sisendid valideeritakse nii rakenduse kui ka andmebaasi poolel. Teenuste puhul on võimalik soovitusi pärida sisestatud HTTP-aadressi järgi. Kasutaja seadistatud päringute ja vastuste puhul on võimalik saada näidispäringuid ja -vastuseid.

Kui kasutaja on seadistanud vastuse mingile teenuse päringule ja selle sisse lülitanud, siis suudab server vastavale sissetulevale HTTP-päringule sellega vastata. Kui kasutaja on seadistanud päringu, siis oskab server sellega välisesse rakendusse päringut teha ning tagasi saadud vastust salvestada. Kui sissetulev HTTP-päring on hoopis selline, mida kasutaja pole seadistanud, suudab server seda välisesse rakendusse edasi saata ning tagasitulnud vastusega pärijale vastata. Päringule seadistatud vastusega vastamine ja selle edasisuunamise töövood on välja toodud joonisel 6.



Joonis 6. Sissetulevale päringule vastamine või selle edasisuunamine

Eelmises lõigus kirjeldatud tegevuste käigus salvestatakse kõik päringud ja vastused andmebaasi, et neid oleks võimalik hiljem leida. Samuti on oluline need logida nii konsooli kui ka tsentraalsesse logimisrakendusse Graylog, kuhu logivad kõik töötukassa rakendused. Testija töövoog võib läbida mitut rakendust ja on oluline, et logisid vaadates oleksid kõik tegevused esindatud. Kui sissetulevas päringus on kaasas jälitamisidentifikaator, tuleb seda logimisel kasutada ja edasisaadetavate päringutega kaasa panna. Jälitamisidentifikaatori abil saab testija enda töövoogu logidest kergelt üles leida. Logitav päring ja vastus on selge struktuuriga, kust on näha HTTP-meetod, -aadress ja

-parameetrid, kasutaja isikukood, päised ning keha ja vastuse puhul ka staatuse kood. Joonisel 7 on näha päringu ja vastuse logisid.

Joonis 7. Päringu ja vastuse logimine

Server peab koostama korrektseid REST-päringuid ja -vastuseid. Päringute sooritamiseks töötukassa rakendustesse peab rakendus suutma genereerida valiidse JSON Web Tokeni, mida välja kutsutavad rakendused aktsepteerivad. X-tee teenustega suhtlemiseks peab server suutma koostada korrektse SOAP-päringu ja vastuse koos manustega.

3.4. Mittefunktsionaalsed nõuded

Rakendus ei tohi testimist keerulisemaks teha ehk ei tohi tekitada juurde vigu, mida ilma selle kasutamiseta ei esine. Selleks on rakendus kaetud ühik- ja integratsioonitestidega.

Rakendus võib sattuda suure koormuse alla, sest töötukassa rakendused teevad palju päringuid nii üksteise vahel kui ka X-tee registritesse. Kuigi testimisel on korrektsus olulisem kui kiirus, ei tohiks see testija tööprotsessi oluliselt aeglasemaks teha. Selle jaoks on rakendus kaetud koormustestidega, mille väljund on näha lisades jooniselt 12. Testide tulemusest on näha, et testkeskkonnas simuleerides paralleelselt 10-ne kasutaja päringuid suutis rakendus töödelda 201,6 päringut sekundis vastates 95% juhtudel 32,5 millisekundi jooksul.

Rakendus peab olema kasutajale kergelt arusaadav. Rakendust hakkavad kasutama peamiselt arendajad ja testijad. Selle jaoks on kasutajaliides ehitatud samasuguse stiili ja komponentidega nagu töötukassa rakendused.

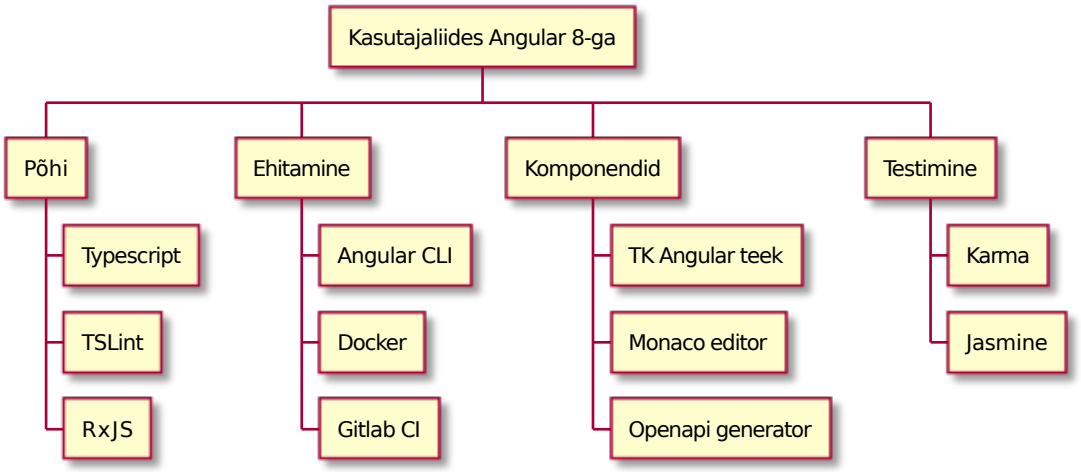
Rakendus peab olema arendajale kergelt edasiarendatav. Selle jaoks on kasutatud kasutajaliidese ja serveri poolel teiste töötukassa rakendustega samu või sarnaseid tehnoloogiaid ja võtteid.

3.5. Rakenduse arhitektuur

Järgnevates alapeatükkides on lähemalt välja toodud tehnoloogiad, millega on kasutajaliides ja serveripoolsed rakendused ehitatud.

3.5.1. Kasutajaliides veebiraamistikuga Angular

Kasutajaliides on ehitatud raamistikule Angular [15]. See on vabavaraline Google'i arendatav veebiraamistik kasutajaliidese loomiseks. Autor kasutas Angulari 8. versiooni. Angular võimaldab rakenduse funktsionaalsust hoida eraldiseisvates komponentides, mis on omakorda jaotatud erinevateks kihtideks. Komponentide ja kihtide eelis on see, et see teeb rakenduse koodi paremini hallatavaks, taaskasutatavaks ja testitavaks. Programmikood on kirjutatud keeles TypeScript, mis põhineb keelel JavaScript, lisades sellele juurde staatilise tüüpimissüsteemi ja funktsionaalsust. Väljakuvatavad elemendid on kirjeldatud HTML-failidena ning stiliseeritud SCSS-failidega. Kasutajaliideses kasutatavad tehnoloogiad on välja toodud joonisel 8.



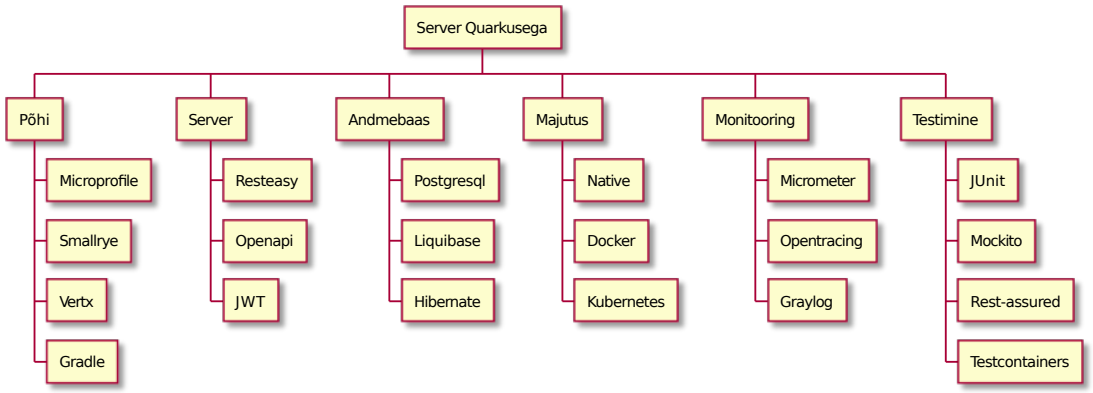
Joonis 8. Kasutajaliideses kasutatavad tehnoloogiad

Üks rakenduse kuva võib koosneda paljudest komponentidest, mis on jaotatud esitlus-, teenuse- ja liidestuskihiks. Esitluskiht koosneb TypeScript- ja HTML-failist ning sisaldab endas elemente, mida välja kuvatakse, ja äriloogikat, kuidas need välja kuvatakse. Liidestuskiht võimaldab REST-päringutega serveripoolsest rakendusest andmeid pärida. Teenuskihti kasutatakse komponentides, kus on vaja liidestuskihist päritud andmeid töödelda või äriloogikat komponentide vahel jagada. Tihtipeale teeb server ära andmete töötluse või komponentide vahel jagatavat osa on vähe, siis see kiht on puudu. Lisades olevatel joonistel 13 ja 14 on näha koodinäiteid kasutajaliidesest.

Kasutajaliides taaskasutab suures osas töötukassa enda rakenduste jaoks mõeldud kuvaelemente. Sellised kuvaelemendid ja stiil on viidud ühisesse teeki, mida kasutavad mitu töötukassa rakendust. Autor otsustas ühist teeki taaskasutada, et rakendus näeks teistele töötukassa rakendustega võimalikult sarnane välja. Sellise lähenemise eeliseks on, et kasutajad on juba tuttavad, kuidas kuvad ja elemendid toimivad. Tulevased arendajad on sarnase koodiga tuttavad ja oskavad vajadusel täiendusi teha. Kuvaelementide taaskasutamine oluliselt kiirendas ning lihtsustas kuvade valmimist, kuid samal ajal kitsendas valikuid, millised need kuvad saaksid välja näha. Eelnevalt väljatoodud joonistel 4 ja 5 on näited kasutajaliidese kuvadest.

3.5.2. Server veebiraamistikuga Quarkus

Veebirakenduse serveripoolne osa on ehitatud raamistikule Quarkus [16]. See on vabavaraline Red Hati sponsoreeritud raamistik veebirakenduse ehitamiseks Java keeles. Autor kasutas Quarkuse 1.13.3. versiooni. Quarkus võimaldab serverit üles ehitada eraldiseisvate komponentide ja kihtidena, kus teenuse-, äri- ja andmebaasikiht on omavahel selgelt eraldatud. Autor valis serveripoolseks raamistikuks Quarkuse, sest see on üles ehitatud olemasolevatele standarditele, seda on kerge üles seada ning võrreldes teiste raamistikega on see märgatavalt kiirem [17]. EMPIS-e projekti seisukohalt on oluline, et loodav rakendus oleks võimalikult sarnane projektis olevate rakendustega, et seda oleks tulevikus võimalik ka teistel arendajatel edasi arendada. Kuigi projektis kasutatavad rakendused on valdavalt ehitatud Spring raamistikule, siis Quarkus on sellele väga sarnane ehk kasutab suures osas samu tehnoloogiaid ja ühildub Springi koodiga. Joonisel 9 on välja toodud serveri poolel kasutatavad tehnoloogiad.



Joonis 9. Serveris kasutatavad tehnoloogiad

Serveripoolne kood on jaotatud teenuse-, äri- ja andmebaasikihtideks ning igas kihis võib olla mitu komponenti. Teenusekiht kirjeldab ära, milliseid REST-päringuid rakendus peab teenindama, tegeleb sisendi valideerimisega ning seejärel annab sissetuleva päringu edasi ärikihile. Teenusekiht on üles ehitatud JAX-RS standardile. Ärikihis on kirjeldatud rakenduse loogika, mis võib näiteks sisendi põhjal andmebaasikihist andmeid pärida, neid töödelda ning teenusekihti tagastada. Kuna serveri pool toimib suures osas CRUD-rakendusena ehk pärib andmeid andmebaasist või salvestab sinna vastavalt sisendile, siis on ärikihi loogikat vähe. Andmebaasikiht on ehitatud Hibernate'i standardile ning võimaldab ärikihil andmebaasist pärida ja sinna salvestada andmeid. Lisades olevatel joonistel 15, 16 ja 17 on näha koodinäiteid teenuse-, äri- ja andmebaasikihtidest.

Lisaks serveripoolsele rakendusele on üles seatud PostgreSQL 12 andmebaas [18]. Autor valis andmebaasiks PostgreSQL-i, sest see on vabavaraline, põhjalikult dokumenteeritud ning EMPIS-e projektis kasutusel. Lisades oleval joonisel 18 on näha rakenduse kasutatav andmebaasi struktuur. Andmebaasi loomiseks on kasutatud Liquibase'i teeki, millega serveri käivitamisel jooksutatakse kaasa pandud SQL-skripte ning luuakse vajalikud andmebaasitabelid.

3.6. Veebirakenduse majutus

Veebirakenduse puhul on kasutajaliidese ja serveri kõrval oluline ka nende majutus keskkonnas. EMPIS-e projektis on rakendused üles seatud Dockeri konteineritena majasiseses Kubernetese klastris, kus majutatakse nelja testkeskkonda. Docker võimaldab luua rakenduse käitamiseks isoleeritud põhja, kus on vajalikud teegid juba paigaldatud ning seadistused tehtud. Projektisiseselt on nii kasutajaliidese kui ka serveri jaoks Dockeri põhjad varasemalt olemas. Kubernetes võimaldab konteinerite käitamist, skaleerimist, võrgusuhtlust ja muid parameetreid seadistada konfiguratsioonifailidega. Sarnaselt projektisisestele rakendustele on töös valminud veebirakenduse kasutajaliides ja server pakendatud Dockeri konteineritena ning need on paigaldatud igasse testkeskkonda Kubernetese klastris. Nelja rakenduse peale on üks ühine andmebaas, mida hoitakse eraldi masinas. Kuigi Kuberneteses on võimalik andmebaasi jaoks kettaruumi tekitada, on selle eraldi masinas hoidmine töökindlam olnud. Rakendused on privaatvõrgus kasutajatele kättesaadavaks tehtud, kasutades Traefiku marsruutimist.

4. Kasutajate tagasiside

Tagasiside kogumise jaoks korraldati arendajatele ja testijatele kaks koosolekut. Arendajatele näidati, kuidas valminud rakendust kasutada, kuid peamine rõhk oli kasutajaliidese ning serveripoolse koodi ülevaatamisel, et näidata, kuidas seda saab edasi arendada ning kuidas kohalikus masinas rakendust käivitada. Testijatele näidati põhjalikumalt kasutajaliidese kasutamist ja erinevaid seadistamisvõimalusi. Seejärel loodi Jira projektihaldamise keskkonda teemapüstitus ning anti kolm nädalat aega rakendust katsetada, välja tuua puudujäägid ja ettepanekud edasiarenduseks. Järgnevad alampeatükid tutvustavad saadud tagasisidet ning rakenduse edasiarendusvõimalusi.

4.1. Tagasiside

Kasutajatele meeldis, et rakenduse kuva oli suures osas neile juba tuttav, kuigi nad nägid seda alles esimest korda. See on sellepärast, et sisselogimine ja kasutajaliideses olevad komponendid on samasugused nagu teistes töötukassa rakendustes. Kuigi komponendid olid tuttavad, jäi kasutajale esialgu segaseks, mida täpselt kasutajaliideses seadistada saab. Peale rakenduse tutvustamist olukord küll lahenes, kuid vaja oleks luua töövoogude kasutusjuhend, et sama mure ei korduks.

Testijad tõid positiivsest küljest välja, et rakendusse sai sisse logida samamoodi nagu teistesse töötukassa rakendustesse ning selle funktsionaalsuse kasutamine ei sega kellegi teise tööd. Samuti meeldis neile, et läbikäinud liikluse põhjal on võimalik seadistada vastuseid ning rakendus logib päringu ja vastuse Graylogi, mida testijad nagunii arenduste testimisel jälgivad. Tagasisides tehti ettepanek, et rakenduses võiks olla võimalus ka teiste loodud vastuseid näha ja endale dubleerida. Praegu on näha vaid kasutaja enda seadistatud vastuseid. Samuti sooviti seadistatud vastuste grupeerimisfunktsionaalsust, sest mingi töövoo testimine võib sisaldada mitmele teenusele vastuse seadistamist ja nende ühekaupa sisse-väljalülitamine on ebamugav.

Lisaks iseseisvale rakenduse katsetamisele, kasutati seda ka kahe arenduse testimiseks. Need koosnesid kaitselüliti funktsionaalsuse lisamisest moodulite vahele ja uue X-tee masspäringu kasutuselevõtmisest. Kaitselüliti funktsionaalsust sai rakendusega edukalt testida ning eelkõige tuli kasuks päringule vastamise viivitus, mida teistes tööriistades seadistada ei saanud. X-tee masspäringu testimisel esines vigu, kus päringuid ei suudetud korrektselt ümber suunata testimisrakendusse. Testijad raporteerisid vead ja testimist jätkati SoapUI tööriistaga. Kuna töötukassa moodulid kasutavad omavaheliseks suhtluseks REST-teenuseid, siis testimisrakenduse arendusel oli rõhk just nende kasutamisel ning SOAP-päring koos manusega pole veel nii töökindel ja vajab parandusi. Seega on testimisrakendusest juba kasu olnud, kuid tulevikus on vaja seda töökindlamaks teha.

4.2. Edasiarendus

Rakenduse arendamisel võeti eesmärgiks võimaldada REST- ja SOAP-päringute sooritamine, päringutele vastamine kasutaja seadistatud vastusega ja päringu edasisuunamine, kui vastus ei olnud seadistatud. Lisaks nendele kasutatakse töötukassa rakenduses ka AMQP-liidestust, mille puhul saaks samuti testimisrakendusest sõnumeid saata ja neid edasi suunata. Praegu AMQP-sõnumitele vastamise funktsionaalsust töötukassa rakendused ei kasuta.

Tänapäeval on olemas rakenduste teenuste kirjeldamiseks OpenAPI standard, mis sisaldab endas rakenduse teenuste spetsifikatsiooni, kus on täpselt kirjas, kuidas nendesse tuleb päringuid teha. Sellist dokumenti saavad kasutada teised koostööpartnerid liidestumiseks. Praegu ükski töötukassa rakendus sellist spetsifikatsiooni ei paku, vaid analüütikud kirjutavad inimloetaval kujul dokumenti teenuste kirjeldused, mida arendajad rakendustes implementeerivad. Töö raames loodud rakendus salvestab andmebaasi läbikäinud päringud ja vastused, mille põhjal oleks võimalik aja jooksul teistele rakendustele OpenAPI spetsifikatsioonid genereerida.

Töö raames arendatud rakendus on mõeldud teenuste manuaaltestimise lihtsustamiseks. Seda saaks kasutada ka rakendustevahelise suhtluse automaatseks testimiseks, sest andmebaasis on olemas kasutajate tehtud näidispäringud ja -vastused, mille põhjal saaksid arendajad luua integratsiooniteste, mis teevad päringuid vastu testimisrakendust.

Kokkuvõte

Töös tutvustati Eesti Töötukassa infosüsteemi EMPIS ja kirjeldati, kuidas süsteem on aastast 2009 järjest suuremaks kasvanud. Seejärel toodi välja Nortali osa infosüsteemi arendamises ja selgitati, kuidas alates aastast 2018 on viidud funktsionaalsust üle väiksemateks mooduliteks. Kuna moodulid suhtlevad üksteisega üle teenuste, siis osadeks tükeldamine on suurendanud nendevahelist andmevahetust üle võrgu ning vajadust seda testida. Seejärel kirjeldati rakenduste integratsioonitestimise metoodikat, teenuste testimisel kasutatavaid tehnoloogiaid, nendega esinevaid probleeme ja kuidas testimisrakendusega teenust testida.

Teiseks koostati nimekiri nõuetest, mis on projektis vajalikud integratsioontestimise jaoks, ning näidati kahte avalikku ja kahte projektisisest vahendit, mida praegu teenuste testimiseks kasutatakse, kuid mis ei kata täielikult eelnevalt kirjeldatud nõudeid.

Kolmandaks kirjeldati töö raames loodud veebirakendust, mille eesmärk on lihtsustada teenuste manuaaltestimist, täites projektis teenuste testimiseks kirjeldatud nõudeid, ning samal ajal pakkuda mugavamat ja kiiremat kasutuskogemust võrreldes varasemate vahenditega. Loodud veebirakendus koosneb kasutajaliidesest ja serveripoolsest osast. Töös kirjeldati funktsionaalsust, mida kasutajaliides ja server pakuvad ning kuidas need on teiste töötukassa rakendustega testkeskkonnas liidestatud. Seejärel näidati rakenduse arhitektuuri ehk tehnoloogiaid, mida kasutajaliides ja server kasutavad ning kuidas on need üles ehitatud. Samuti toodi välja, kuidas rakendus on testkeskkonnas majutatud.

Valminud veebirakendust tutvustati arendajatele ja testijatele ning paluti kolme nädala jooksul tagasisidet anda. Kasutajad tõid tagasisides välja, et rakenduse kuva oli suures osas neile juba tuttav, sest see taaskasutab kasutajaliideses samu komponente, mis teisedki töötukassa rakendused. Kasutajatele meeldis, et testimisrakenduse kasutamine on isikupõhine, seega see ei sega kellegi teise tööd ning rakendus logib oma tegevused tsentraalsesse logimisrakendusse Graylog. Ettepanekutena toodi välja funktsionaalsuse soove, mis muudaksid rakenduse kasutamise veelgi mugavamaks. Lisaks rakenduse katsetamisele prooviti seda kahe arenduse testimisel, kus oli näha, et uuest rakendusest oli kasu, kuid tulevikus on vaja seda töökindlamaks teha.

Kasutatud kirjandus

[1] Eesti Töötukassa. *Raamlepingu eseme tehniline kirjeldus. Riigihanke dokumendid. Eesti Töötukassa tööturuteenuste ja -toetuste infosüsteemi (EMPIS/EMPIS2) jätkuarendus- ja hooldustööd*. 2019. URL: https://riigihanked.riik.ee/rhr-web/#/procurement/1589937/documents/source-document?documentOldId=13298530.

[2] Meelis Paavel. *Uus infosüsteem võimaldab pühendada rohkem aega kliendile. Eesti Töötukassa uudised*. 27. november 2009. URL: https://www.tootukassa.ee/uudised/meelis-paavel-uus-infosusteem-voimaldab-puhendada-rohkem-aega-kliendile (vaadatud 07.12.2020).

[3] Rauno Siimann. „Continuous Delivery põhimõtete rakendamisest Eesti Töötukassa infosüsteemi arendusprotsessis". Bakalaureusetöö. Tartu Ülikool, 2012. URL: http://hdl.handle.net/10062/32898.

[4] Eesti Töötukassa. *Raamlepingu eseme tehniline kirjeldus. Riigihanke dokumendid. Eesti Töötukassa infosüsteemide majutuse teenus*. 2019. URL: https://riigihanked.riik.ee/rhr-web/#/procurement/1620300/documents/source-document?group=B&documentOldId=13292103.

[5] Riigi Infosüsteemi Amet. *X-tee avaandmed. Alamsüsteem TKIS*. URL: https://logs.x-tee.ee/EE/gui/ (vaadatud 30.11.2020).

[6] Kristjan Hendrik Küngas. *X-tee andmete kogumise tööriist*. URL: https://github.com/kristjanhk/xroad-logs-scraper (vaadatud 08.12.2020).

[7] Polina Morozova. „Süstematiseeritud testimise protsessi läbiviimine". Magistritöö. Tartu Ülikool, 2011. URL: http://hdl.handle.net/10062/33033.

[8] Dmitrii Savchenko. „Testing microservice applications". Lappeenranta-Lahti University of Technology LUT, 2019. URL: http://urn.fi/URN:ISBN:978-952-335-415-9.

[9] Jyri Lehvä, Niko Mäkitalo ja Tommi Mikkonen. „Consumer-Driven Contract Tests for Microservices: A Case Study". Teoses: *Product-Focused Software Process Improvement*. Springer Nature Switzerland, 2019, lk. 497–512. ISBN: 978-3-030-35332-2. DOI: 10.1007/978-3-030-35333-9_35.

[10] Riigi Infosüsteemi Amet. *X-tee alamsüsteemide kataloog teenuste ja WSDL kirjeldustega*. URL: https://x-tee.ee/catalogue/EE (vaadatud 07.12.2020).

[11] Mirjam Rauba. „Pideva tarnimise strateegia rakendamise analüüs AS Webmedia Group projektis EMPIS". Magistritöö. Tartu Ülikool, 2012. URL: http://hdl.handle.net/10062/33060.

[12] SmartBear Software. *SoapUI Open Source*. URL: https://www.soapui.org/tools/soapui (vaadatud 08.12.2020).

[13] Inc Postman. *Postman*. URL: https://www.postman.com (vaadatud 08.12.2020).

[14] Kristjan Hendrik Küngas. *Rest-mock*. URL: https://github.com/kristjanhk/rest-mock (vaadatud 08.12.2020).

[15] Google. *Angular*. URL: https://angular.io (vaadatud 08.03.2021).

[16] Red Hat. *Quarkus*. URL: https://quarkus.io (vaadatud 08.03.2021).

[17] TechEmpower. *Web Framework Benchmarks*. URL: https://www.techempower.com/benchmarks (vaadatud 08.03.2021).

[18] The PostgreSQL Global Development Group. *PostgreSQL*. URL: https://www.postgresql.org (vaadatud 08.03.2021).

[19] Tomas Topinka. *Statistic plugin for Intellij IDEA*. URL: https://plugins.jetbrains.com/plugin/4509-statistic (vaadatud 07.12.2020).

[20] k6.io. *Load testing for engineering teams*. URL: https://k6.io/ (vaadatud 28.04.2021).

Lisad

I. Mõisted ja terminid

AMQP	*Advanced Message Queuing Protocol* ehk progressiivne sõnumijärjekorra protokoll – asünkroonse sõnumivahetuse standard. 7, 8, 23
Andmebaasikiht	rakenduse osa, mis tegeleb andmebaasist andmete pärimisega. 5
CAS	*Central Authentication Service* ehk tsentraalne autentimisteenus, millega kasutaja saab ühekordsel autentimisel ligipääsu infosüsteemi rakendustele. 13, 16
CRUD	püsiliku andmebaasi põhioperatsioonid: loomine, lugemine, uuendamine, kustutamine (*Create, Read, Update, Delete*). 21
Docker	tarkvarakomplekt, millega on võimalik tarkvara konteinerdada virtualiseerimisega opsüsteemi tasemel. 21
Esitluskiht	rakenduse osa, mis kuvab kasutajale andmed välja. 5
Hazelcasti klaster	Hazelcast võimaldab rakendusi integreerida andmevõreks ehk võimaldab mingit andmekogu hoida ja uuendada kõigis klastriga liitunud rakendustes korraga. 16
HTML	*HyperText Markup Language* ehk hüperteksti märgistuskeel – veebilehtede loomise vahend, mis võimaldab välja kuvatavaid dokumente vormistada. 19
HTTP	*HyperText Transfer Protocol* ehk hüperteksti edastuse protokoll – veebiteenuste alusstandard, mis määrab sõnumite vormingu ja edastuse. 7, 10, 13–17
JAX-RS	*Java API for RESTful Web Services* – standardne liides veebiteenuste loomiseks Java keeles. 21

JSON	*JavaScript Object Notation* ehk JavaScripti objektide notatsioon – lihtne andmevahetusformaat, mis põhineb JavaScripti alamhulgal ja sobib inimlugemiseks ning -kirjutuseks. 7, 10, 12, 15
JSON Web Token (JWT)	ühene identifikaator juurdepääsu taotleva rakenduse autentimiseks ehk veebitõend, mis kasutab standardset JSON-struktuuri. 18
Kaitselüliti	*circuit-breaker* – teenuse suhtlemise vahel olev mehhanism, mis vigade korral peatab päringute sooritamise ettemääratud ajaks, et parandada rakenduse jõudlust niikaua kui veaolukord laheneb. 22
Kubernetes	konteinerite haldussüsteem, millega automatiseeritakse rakenduste majutamist ja skaleerumist. 21
Mikroteenuste arhitektuur	arhitektuur, milles süsteemi komponendid on jaotatud äriloogiliselt eraldatud osadeks, mis suhtlevad üksteisega üle võrgu. 5
MIME	*Multi-Purpose Internet Mail Extensions* ehk interneti-posti mitmeotstarbelised laiendid – algselt meiliprotokolli SMTP laiendav standard, mis võimaldab mitmeosalisi sõnumikehi edastada manustena. 10
Monoliitne arhitektuur	arhitektuur, milles süsteemi komponendid on omavahel tihedalt seotud ja peavad töötama tervikuna. 5
OAuth	*Open Authorization* ehk avatud volitamine – volitusprotokoll, mis võimaldab kolmandal osapoolel saada ligi kasutaja andmetele ilma, et kasutaja peaks jagama oma parooli. 16
OpenAPI	standard, mis kirjeldab masinloetaval kujul, kuidas REST teenuse spetsifikatsioon peab välja nägema. 23
REST	*Representational State Transfer* ehk esitusoleku siire – veebiteenuste programmeerimise paradigma, mis toetub HTTP standardile. 7, 8, 10–12, 18, 19, 21–23

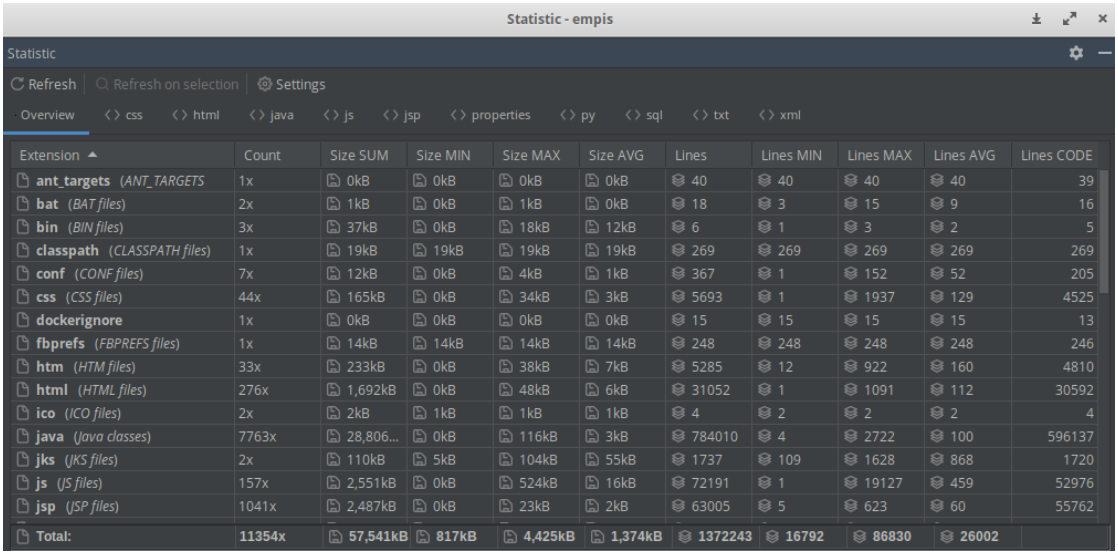
SCSS	*Sassy Cascading Style Sheets* - laiendus CSS-le, mis on märgistuskeel dokumendi välisilmse kirjeldamiseks. 19
SOAP	*Simple Object Access Protocol* ehk lihtne objektipöörduse protokoll – üks veebiteenuste alusstandarditest, mis toetub XML ja HTTP standarditele. 7, 8, 10, 11, 18, 22, 23
SQL	*Structured Query Language* ehk struktureeritud päringute keel – deklaratiivne programmikeel, mida kasutatakse andmebaasist andmete pärimiseks. 21
Traefik	üks pöördproksimise lahendustest, mida kasutatakse Kuberneteses. Pöördproksi on vaheserver, mis vahendab välisklientide päringuid siseserveritele. 21
Vastavustestimine	testimine eesmärgiga välja selgitada, kas süsteem või teenus vastab spetsifikatsiooni või lepingu nõuetele. 6
WSDL	*Web Services Description Language* ehk veebiteenuste kirjeldamise keel – liidesemääratluskeel veebiteenuste kirjeldamiseks, XML-põhine, kasutatakse eelkõige SOAP-teenuste puhul. 8, 12
X-tee	andmevahetuse platvorm, mis võimaldab turvaliselt asutuste vahel teavet pärida ja vahetada. 6, 7, 9–12, 18, 22
XML	*Extensible Markup Language* ehk laiendatav märgistuskeel – platvormist sõltumatu märgistuskeel, mis põhineb avatud standarditel. 7, 10, 15

II. Lähtekood

Töö raames loodud avalikud koodihoidlad asetsevad järgmistel aadressitel:

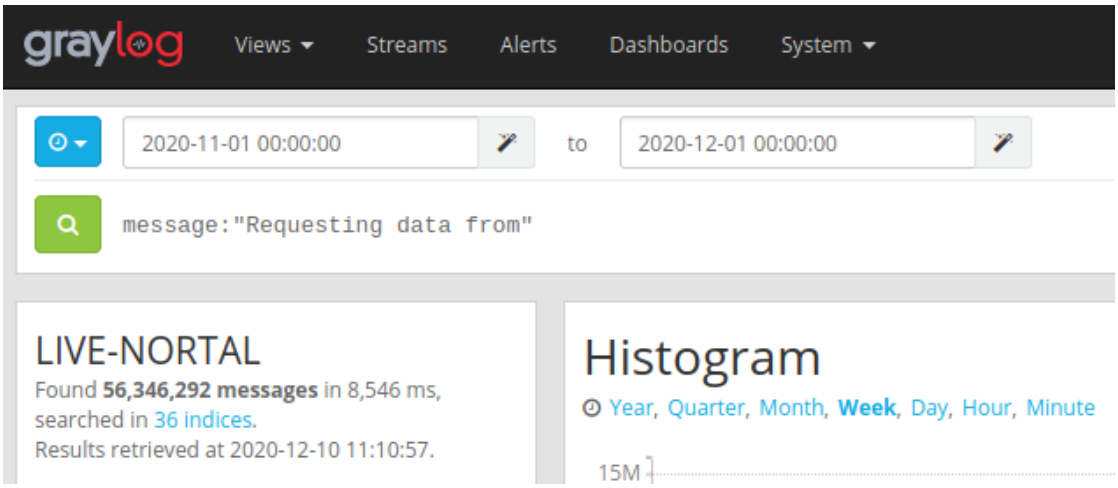
1. `https://gitlab.com/kristjanhk/tk-mock` (Latexi projekt töö dokumendi ehitamiseks)
2. `https://github.com/kristjanhk/xroad-logs-scraper` (X-tee avaandmete kogumise tööriist)

III. Joonised

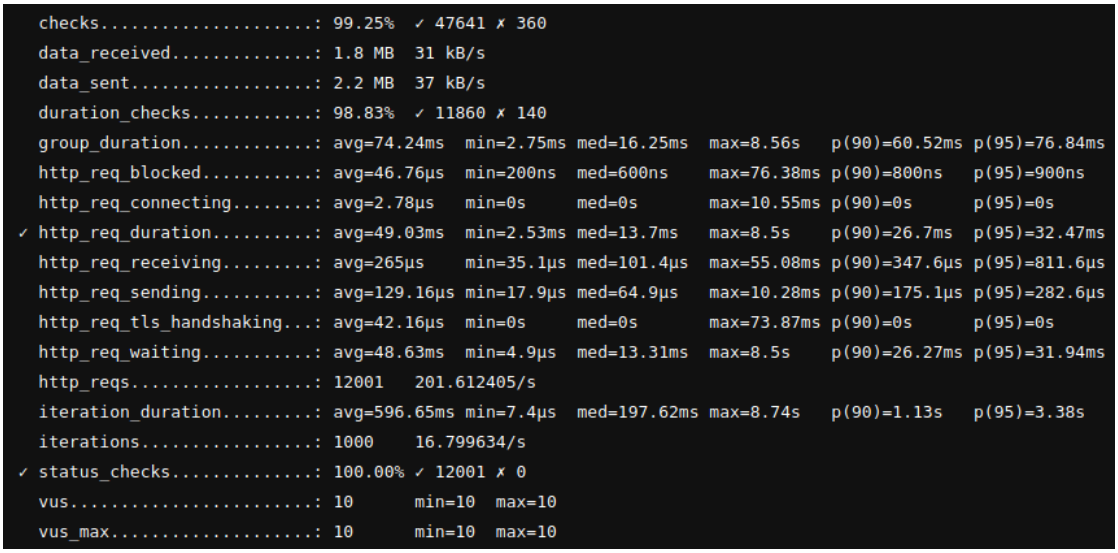


Extension	Count	Size SUM	Size MIN	Size MAX	Size AVG	Lines	Lines MIN	Lines MAX	Lines AVG	Lines CODE
ant_targets (ANT_TARGETS	1x	0kB	0kB	0kB	0kB	40	40	40	40	39
bat (BAT files)	2x	1kB	0kB	1kB	0kB	18	3	15	9	16
bin (BIN files)	3x	37kB	0kB	18kB	12kB	6	1	3	2	5
classpath (CLASSPATH files)	1x	19kB	19kB	19kB	19kB	269	269	269	269	269
conf (CONF files)	7x	12kB	0kB	4kB	1kB	367	1	152	52	205
css (CSS files)	44x	165kB	0kB	34kB	3kB	5693	1	1937	129	4525
dockerignore	1x	0kB	0kB	0kB	0kB	15	15	15	15	13
fbprefs (FBPREFS files)	1x	14kB	14kB	14kB	14kB	248	248	248	248	246
htm (HTM files)	33x	233kB	0kB	38kB	7kB	5285	12	922	160	4810
html (HTML files)	276x	1,692kB	0kB	48kB	6kB	31052	1	1091	112	30592
ico (ICO files)	2x	2kB	1kB	1kB	1kB	4	2	2	2	4
java (Java classes)	7763x	28,806...	0kB	116kB	3kB	784010	4	2722	100	596137
jks (JKS files)	2x	110kB	5kB	104kB	55kB	1737	109	1628	868	1720
js (JS files)	157x	2,551kB	0kB	524kB	16kB	72191	1	19127	459	52976
jsp (JSP files)	1041x	2,487kB	0kB	23kB	2kB	63005	5	623	60	55762
Total:	11354x	57,541kB	817kB	4,425kB	1,374kB	1372243	16792	86830	26002	

Joonis 10. Koodiridade statistika väljavõte projektist EMPIS, kasutades Statistic [19] pluginat koodiredaktorile Intellij IDEA 07.12.2020 seisuga



Joonis 11. Nortali arendatavate Töötukassa moodulite päringute statistika 10.12.2020 seisuga



Joonis 12. K6 [20] tööriistaga mõõdetud serveri koormustestide tulemus

```
<tk-form-panel>
    <tk-form-panel-content [titleAsideEnabled]="true"
                           [collapsible]="true"
                           [collapsed]="false">
        <ng-container contentTitle>
            <tk-button-simple (click)="openRouteModal()">
                {{getTitle()}}
            </tk-button-simple>
        </ng-container>

        <ng-container contentTitleAside>
            <tk-action-list-item>
                <tk-button-simple [isToolbarButton]="true"
                                  (click)="toggleFilter()">
                    <tk-icon [icon]="icons.SEARCH"></tk-icon>
                    Filtreeri
                </tk-button-simple>
            </tk-action-list-item>

            <tk-action-list-item>
                <tk-button-simple [isToolbarButton]="true"
                                  (click)="createMock()">
                    <tk-icon [icon]="icons.ADD"></tk-icon>
                    Lisa päring
                </tk-button-simple>
            </tk-action-list-item>
        </ng-container>

        <m-mock-list [route]="route"
                     [mocks]="mocks"
                     [showFilter]="showFilter">
        </m-mock-list>
    </tk-form-panel-content>
</tk-form-panel>
```

Joonis 13. Kasutajaliidese HTML-faili koodinäide

```
import {Component, EventEmitter, Input, Output, ViewChild} from "@angular/core";
import {Icons, ModalFactory, Util} from "@tk/tk-angular-common";
import {Route} from "../../generated/models/route";
import {MockListComponent} from "../../mock/list/mock-list.component";
import {Mock} from "../../generated/models/mock";

@Component({selector: 'm-route-panel', templateUrl: './route-panel.component.html'})
export class RoutePanelComponent {

    @ViewChild(MockListComponent, {static: true})
    private mockList: MockListComponent;

    @Input() route: Route;
    @Input() mocks: Mock[];
    @Input() collapsed: boolean;
    @Output() onRouteChanged: EventEmitter<void> = new EventEmitter<void>();

    icons: typeof Icons = Icons;
    showFilter: boolean = false;

    constructor(private modalFactory: ModalFactory) {
    }

    getTitle(): string {
        const notes = Util.isPresent(this.route.name) ? ` (${this.route.name})` : '';
        return `${this.route.type} ${this.route.method} ${this.route.path}${notes}`;
    }

    toggleFilter(): void {
        this.showFilter = !this.showFilter;
    }

    createMock(): void {
        this.mockList.createMock();
    }

    openRouteModal(): void {
        //...
    }
}
```

Joonis 14. Kasutajaliidese TypeScript-faili koodinäide

```java
package ee.tk.mock.route.service;

//import ...

@Authenticated
@Tag(name = "route")
@Path("be/route")
public class RouteResource {
  @Inject
  RouteValidator routeValidator;
  @Inject
  RouteService routeService;

  @GET
  @Path("{id}")
  @Produces(APPLICATION_JSON)
  public Optional<Route> getRoute(@PathParam("id") long id) {
    return routeService.get(id);
  }

  @POST
  @Consumes(APPLICATION_JSON)
  @Produces(TEXT_PLAIN)
  @Transactional
  public long createRoute(@NotNull @Valid Route route) {
    routeValidator.validateRouteUniqueByPathAndIdCode(route);
    return routeService.create(route);
  }

  @PUT
  @Consumes(APPLICATION_JSON)
  @Produces(TEXT_PLAIN)
  @Transactional
  public long updateRoute(@NotNull @Valid Route route) {
    return routeService.update(route);
  }
}
```

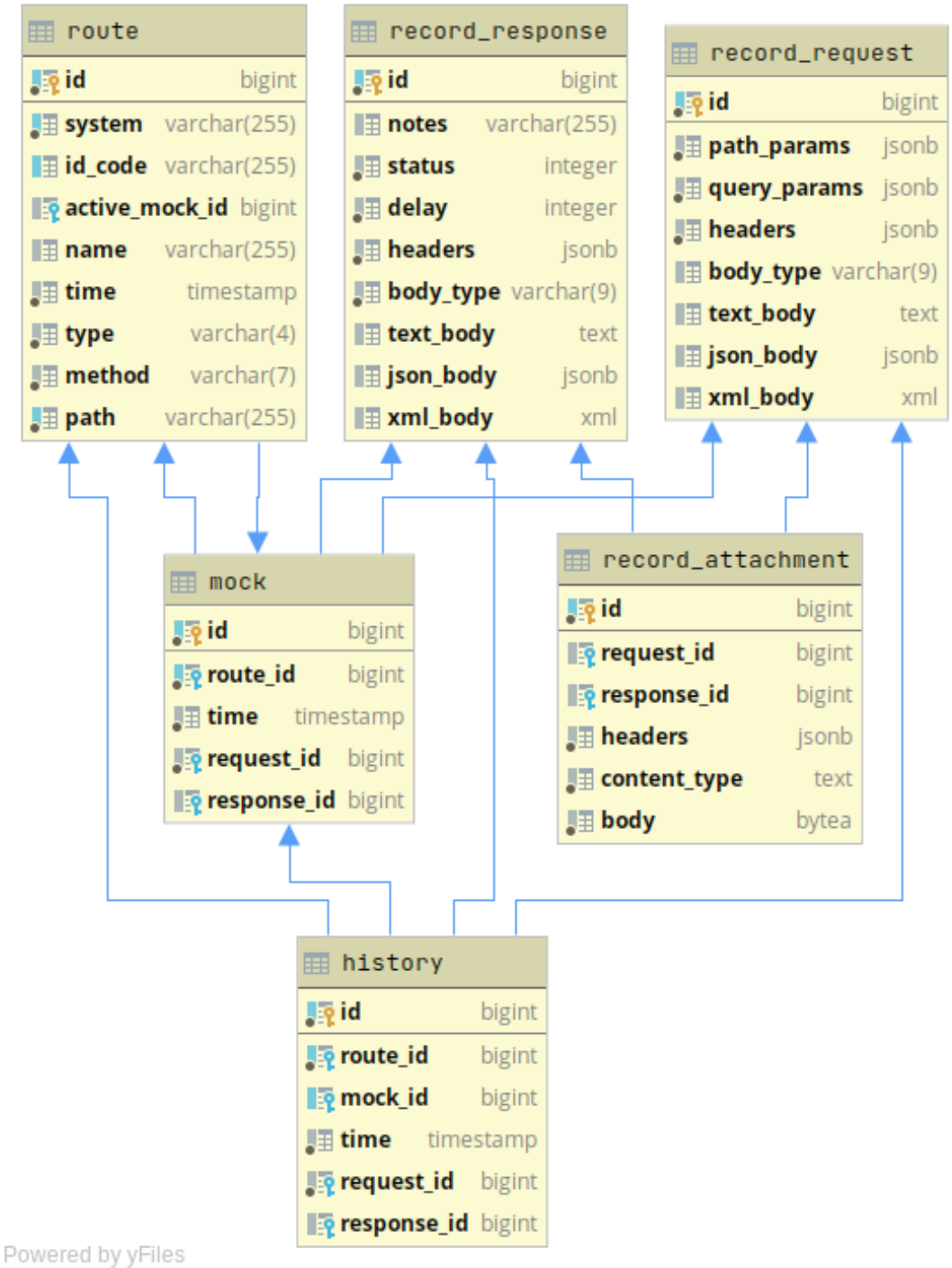
Joonis 15. Serveri teenusekihi koodinäide

```java
package ee.tk.mock.route.service;

//import ...

@ApplicationScoped
public class RouteService {
  @Inject
  RouteRepository routeRepository;
  @Inject
  RouteMapper routeMapper;

  public Optional<Route> get(long id) {
    return routeRepository.findByIdOptional(id)
        .map(entity -> routeMapper.convert(entity));
  }

  public long create(Route route) {
    route.setTime(LocalDateTime.now());
    RouteEntity entity = routeMapper.inverse(route);
    routeRepository.persist(entity);
    route.setId(entity.getId());
    return entity.getId();
  }

  public long update(Route route) {
    route.setTime(LocalDateTime.now());
    RouteEntity entity = routeMapper.inverse(route);
    return routeRepository.getEntityManager().merge(entity).getId();
  }
}
```

Joonis 16. Serveri ärikihi koodinäide

```java
package ee.tk.mock.route.service;

//import ...

@ApplicationScoped
public class RouteRepository implements PanacheRepository<RouteEntity> {

  public List<RouteEntity> findByIdCodeAndSystem(String idCode, String system) {
    return list("idCode = ?1 AND system = ?2 ORDER BY time DESC", idCode, system);
  }

  public List<RouteEntity> findByIds(List<Long> ids) {
    return list("id IN ?1", ids);
  }

  public Optional<RouteEntity> getByMethodAndPathAndIdCode(RouteMethod routeMethod,
                                                           String path,
                                                           String idCode) {
    return find("method = ?1 AND path = ?2 AND idCode = ?3", routeMethod, path, idCode)
        .singleResultOptional();
  }
}
```

Joonis 17. Serveri andmebaasikihi koodinäide



Joonis 18. Andmebaasi struktuur

IV. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Kristjan Hendrik Küngas**,

1. Annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Teenuste testimise hõlbustamine Eesti Töötukassa infosüsteemi EMPIS näitel**, mille juhendaja on Vambola Leping, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Kristjan Hendrik Küngas
07.05.2021