TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Märt Tender

DeepMOOC platvormile tagarakenduse arendamine

Bakalaureusetöö (9 EAP)

Juhendajad: Tõnis Hendrik Hlebnikov
Ahti Põder, PhD

Tartu 2023

DeepMOOC platvormile tagarakenduse arendamine

Lühikokkuvõte:

Tartu Ülikoolis arendatakse uut veebiplatvormi nimega DeepMOOC, mis avardaks programmeerimisainetes tudengite teadmiste kontrollimise viise. Rakenduse rõhk on programmeerimisülesannete automaatkontrollide süsteemil, kuid läbi pistikprogrammide süsteemi hakkab rakendus toetama ka teistsugused ülesandeliike. Käesolev bakalaureusetöö keskendus DeepMOOC platvormi tagarakenduse üles ehitamisele, sealhulgas rakenduse tööks vajalike tehnoloogiate valikule ja andmebaasiskeemi loomisele. Lisaks loodi rakenduse autentimis- ja volitamissüsteem, mitmed tagarakenduse teenused eesrakendusega suhtlemiseks ning integratsioon andmebaasiga. Antud töö raames valmis DeepMOOCi tagarakenduse teine versioon. Töös selgitatakse uue tagarakenduse loomise põhjuseid ning tuuakse paralleele esimese versiooniga.

Võtmesõnad: tarkvaraarendus, tarkvaratehnika, programmeerimisõpe

CERCS: P175 Informaatika, süsteemiteooria

Developing a backend for the DeepMOOC platform

Abstract:

A new web platform called DeepMOOC that aims to broaden the ways of examining students is being developed at the University of Tartu. The application’s emphasis is on automatically verifying solutions to programming tasks but it will also support other kinds of exercises through a planned plug-in system. This bachelor’s thesis focuses on building the backend for the DeepMOOC platform, including selection of technologies and design of the database schema. The application’s authentication and authorization system and several web endpoints were also created in addition to integrating the database with the application. This thesis outlines the creation of the second version of the DeepMOOC backend. The reasons for developing a new backend are explained in the paper and many comparisons to the first version are made.

Keywords: software development, software engineering, programming education

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

Programmeerimisõppe käigus on suur osatähtsus õppurite teadmiste kontrollimisel läbi koodi kirjutamist nõudvate ülesannete. Õppejõudude jaoks on esitatud lahenduste üle vaatamine tihti aeganõudev protsess, mistõttu võetakse selleks appi programmikoodi automaatkontrollid.

Tartu Ülikoolis on lahenduste automaatseks verifitseerimiseks praegu kasutusel Moodle platvormi pistikprogramm VPL ehk Virtual Programming Lab[1]. Antud lahenduse põhilised probleemid on keeruline automaattestide loomise protsess ja puudulik tugi erinevate ülesandeliikide jaoks. Põhjalikuma ülevaate VPLi kitsaskohtadest tegi J. Näks oma bakalaureusetöös [1], mistõttu keskendub käesolev töö juba ainult alternatiivse lahenduse loomisele.

Uus platvorm nimega DeepMOOC saigi alguse soovist luua kasutajasõbralikum ja rohkemate võimalustega automaatkontrollisüsteem, mis asendaks VPLi. Antud töö eesmärk oli luua DeepMOOC platvormile tagarakendus, mis oleks võimeline eesrakendusest vastu võtma ja töötlema kasutajate haldusega seotud päringuid, sealhulgas suhtlema andmebaasiga. Lisaks eelnevale oli sihiks platvormi kasutajate tuvastamiseks ning pääsulubade töötlemiseks autentimis- ja volitamissüsteemi kirjutamine.

Arendustöö DeepMOOC platvormi tagarakenduse kallal algas J. Näks poolt juba 2022 aasta alguses [1]. Käesoleva bakalaureusetöö raames alustati arendustööga otsast peale ehk loodi DeepMOOCile täiesti uus tagarakendus. Uuesti alustamise põhiliseks põhjuseks on eelmises versioonis kasutusel olnud programmeerimiskeele Go puudulik tugi pistikprogrammide (ingl *plug-in*) süsteemi loomiseks. Kuigi antud bakalaureusetöö raames loodud tagarakenduse tarbeks J. Näksi poolt kirjutatud koodist inspiratsiooni ei võetud, siis sellegipoolest tuuakse töös välja mitmeid paralleele varasema ja uue lahenduse vahel.

Töö on jagatud neljaks suuremaks peatükiks. Esimeses osas lahatakse platvormi arhitektuurilisi põhimõtteid ja võrreldakse valitud tehnoloogiaid alternatiividega. Töökäigu peatükis käiakse sammhaaval läbi tarkvaraarenduse protsess ning tuuakse välja mõned esinenud probleemid. Tulemuste osas kirjeldatakse valminud rakendust ning selle

[1] https://moodle.org/plugins/mod_vpl

funktsionaalsusi. Samuti tehakse soovitusi tagarakenduse edasiarendamiseks. Viimases ehk testimise peatükis tuuakse välja tagarakenduse testimispõhimõtted ja loodud automaattestid.

1. Teoreetiline ülevaade

Järgnevates peatükkides seletatakse lahti DeepMOOCi arendusprotsessi teoreetiline pool. Teoreetilise osa eesmärk oli valminud analüüside ja võrdluste toel teha võimalikult sobivaid rakenduse arhitektuuri ja selles kasutatavate tehnoloogiate alaseid valikuid.

1.1 Arhitektuurilised valikud

Enne DeepMOOCi arendamise juurde jõudmist tuli teha mõned otsused rakenduse arhitektuuriliste omaduste osas. Arhitektuuri juures on oma roll kindlasti ka kasutatavatel tarkvarateekidel ja -raamistikel, kuid selleks, et end tehnoloogiliste valikutega mitte nurka mängida, tuli otsused autentimise ja rakenduse modulaarsuse osas eelnevalt ära teha.

1.1.1 Autentimine

Autentimine on protsess, mille käigus rakendus tuvastab kasutaja. Üldjuhul on see tegevus kasutaja poolt vaadates sünonüümne terminiga „sisse logimine“. Töös edaspidi kirjutatakse ka volitamisest, mis on eraldiseisev protsess. Volitamine tähendab kasutajale õiguste andmist. Käesolev peatükk keskendub just DeepMOOCi autentimissüsteemile ehk kasutajate tuvastamisele, analüüsib alternatiive ning selgitab tehtud valikuid.

DeepMOOC rakenduse kasutajasõbralikkuse tagamiseks sooviti kasutajate autentimiseks rakendada ühte kahest Tartu Ülikoolis kasutusel olevatest ainulogimise päringu süsteemidest (ingl *Single Sign-On,* SSO). Need süsteemid võimaldavad tudengitel ja õppejõududel DeepMOOCi sisse logida oma ülikooli kasutajatunnuse ja parooliga ning kaotavad vajaduse uue rakendusepõhise kasutajakonto loomiseks. Juba olemasolevate autentimissüsteemide rakendamine on ka turvalisuse vaatenurgast kindlam variant kui rakendusse täiesti eraldiseisva autentimissüsteemi kirjutamine.

Üks variant oli kasutada OpenID Connect põhist Microsofti poolt pakutavat autentimislahendust, mida kasutab ka Tartu Ülikooli Moodle keskkond. Teine, SAML[2] 2.0 standardile vastav süsteem, on kasutusel näiteks õppeinfosüsteemi sisse logimisel. Järgnevalt võrreldakse OpenID Connect ja SAML protokollide kasutamist DeepMOOC rakenduse tarbeks.

[2] Security Assertion Markup Language

OpenID Connect ja SAML standardeid järgivad autentimissüsteemid on üldjuhul väga sarnased – standardite erinevused on eelkõige tehnilised. Näiteks toimub OpenID Connect põhise lahenduse puhul sisemiselt andmete saatmine JSON formaadis [2], kuid SAML standard näeb ette XML formaadi kasutamise [3]. See aga protokolli kasutuselevõtu juures rolli ei mängi, kuna andmete saatmise eest vastutab protokolli realiseeriv teek.

Kasutaja vaatest toimub autentimine mõlema standardi puhul analoogselt. Lihtsustatud kujul on protsess järgmine:

1) Teenusepakkuja ehk rakendus suunab autentimata kasutaja identiteedipakkuja (ingl *identity provider,* IdP) juurde.
2) Identiteedipakkuja autendib kasutaja.
3) Eduka sisselogimise korral suunatakse kasutaja tagasi teenusepakkuja juurde.

Tartu Ülikooli identiteedipakkujate võimalusi võrreldes tuli sisse üks oluline erinevus. Ülikooli kasutatav OpenID Connect identiteedipakkkuja toetab ainult kasutajatunnuse ja parooli põhist autentimist. Tartu Ülikooli SAML identiteedipakkujal[3] on aga integratsioon Riigi Autentimisteenusega[4], mis tähendab, et lisaks ülikooli kasutajanime ja parooli kasutamisele on seal võimalik sisse logida ka näiteks ID-kaardi, Mobiil-ID või Smart-ID-ga. Nende alternatiivsete sisselogimisvõimaluste olemasolule tuginedes osutuski DeepMOOCi jaoks valituks SAML põhine autentimissüsteem.

1.1.2 Laiendatavus

DeepMOOCi puhul on üks tähtsamaid arhitektuurilisi aspekte platvormi modulaarsus. Selle väljund on tulevane pistikprogrammide süsteem, millega peaks olema võimalik platvormile lisada erinevaid funktsionaalsusi – alustades uutest ülesandeliikidest kuni näiteks alternatiivse autentimissüsteemini.

Käesolev bakalaureusetöö küll ei käsitle pistikprogrammide süsteemi arendust, kuid tagarakenduse arendamisel ja tehnoloogiliste valikute tegemisel oli modulaarsus siiski oluline faktor. Kogu SAML põhine autentimissüsteem sai ehitatud eraldiseisva moodulina ning on seega soovi korral lihtsasti välja vahetatav mõne muu lahenduse vastu. See tuli

[3] https://auth.ut.ee/

[4] https://tara.ria.ee/

kasuks juba ka automaattestide kirjutamisel, kus kasutaja autentimine ei ole oluline ning seega võib SAMLil põhineva süsteemi asendada lihtsamaga.

Modulaarne arhitektuur teeb lihtsamaks ka mõningate arendusel ilmnenud probleemide ületamise. Näiteks pole DeepMOOCi praeguses arendusstaadiumis teada, kuidas hakkab rakendusse jõudma info selle kohta, mis ainetesse ja rühmadesse on iga tudeng registreerunud. Samas on teada, et nende andmete kogumisega hakkab tegelema eraldi integratsioonimoodul. Seega oli praegu vaja lihtsalt implementeerida vastavale moodulile kättesaadavad liidesed, mis võimaldaks õppeainetesse registreerumiste info DeepMOOCi andmebaasi salvestada.

1.2 Valitud tehnoloogiad

Järgnevalt tutvustatakse tehtud tehnoloogilisi valikuid ning selgitatakse nende tagamaid. Valida tuli programmeerimiskeel, kasutatavad tarkvararaamistikud ja andmebaasi haldamise süsteem. Kui andmebaasisüsteemi valik olid võrdlemisi eraldiseisev, siis tarkvararaamistike valimine sai toimuda alles pärast programmeerimiskeele fikseerimist.

1.2.1 Kotlin

Kotlin on ettevõtte JetBrains poolt loodud avatud lähtekoodiga programmeerimiskeel [4]. Üks Kotlini väljaarendamise põhjustest oli soov luua programmeerimiskeel, mis oleks kasutajasõbralikum ja lihtsama süntaksiga kui Java, kuid samas säilitaks võimaluse lihtsasti kasutada ära ja integreeruda juba olemasoleva Java koodiga [5]. Seda võimaldab eelkõige asjaolu, et Kotlinit on võimalik sarnaselt Javaga kompileerida Java baitkoodiks, mis omakorda jookseb Java virtuaalmasina (JVM) peal [5].

Kuigi Kotlin on pigem tuntud keelena, milles on soovitatav luua Androidi rakendusi [6], siis tegelikult on tegu tavalise üldkasutatava programmeerimiskeelega, mis sobib ka veebiarendusel tagarakenduse loomiseks [7].

Varasemalt oli DeepMOOCi tagarakenduse arenduskeeleks Google'i poolt loodud keel Go [1], kuid DeepMOOCi tiimis jõuti arusaamisele, et kuna Gos on programmikoodi dünaamiline laadimine probleemne [8], siis ei ole sellega jätkamine tuleviku arendusplaane vaadates mõistlik. Eelmises peatükis mainitud pistikprogramme peab olema võimalik rakenduse jooksmise ajal DeepMOOCiga integreerida, kuid kompileeritavas keeles nagu

Go on see väga keeruline. Seega toimub sellest õppeaastast kogu DeepMOOC taga-rakenduse arendus keeles Kotlin.

Valituks osutus spetsiifiliselt just Kotlin, kuna see on Java programmeerimise baasilt tulles üsna lihtsasti mõistetav keel. Teine kaalutav programmeerimiskeel oli Python, kuid erinevalt sellest on Kotlin staatilise tüüpimisega [9] ja nullikindlate tüüpidega keel [10], mis oluliselt lihtsustab arendusprotsessi. Need omadused teevad Kotlinis programmeerimise mugavamaks, kuna aitavad arendajal rakendust kirjutades teha vähem vigu.

1.2.2 Jooby

Jooby on Javas ja Kotlinis kasutamiseks mõeldud veebiarenduse tarkvararaamistik, mille loojaks on Edgar Espina [11]. Jooby on küll kirjutatud põhiliselt Java keeles [12], kuid dokumentatsioon on olemas mõlema toetatud keele jaoks, mis tähendab, et Java baas ei ole probleem.

Jooby asemel olid algul kaalumisel ka Spring ja Ktor veebiraamistikud. Nende mõlema eeliseks Jooby ees on suurem populaarsus [13], mis võimaldab lihtsamini leida internetist koodinäiteid ning saada vastuseid arenduse käigus tekkinud küsimustele. Kuid nendel raamistikel oli samas puuduseid, mis nende kasutamise välistasid.

Spring raamistik on küll aasta 2020 seisuga kõige populaarsem Java veebirakenduste tarkvararaamistik [13], kuid meie rakenduse jaoks see ei sobinud. Spring sisaldab endas väga palju erinevaid võimekusi, mis üldjuhul on küll hea, kuid samas on nii suure ökosüsteemiga tutvumine ja selles orienteerumine aeganõudev protsess. Arvestades, et tagarakenduse arendus toimub ka edaspidi tudengite lõputööde raames, siis ei ole nii keerulise raamistiku kasutamine otstarbekas. Jooby on lihtsam raamistik kui Spring ning selles on kõik DeepMOOCi jaoks vajalikud võimalused olemas.

Ktor on tarkvararaamistik, mille arendajaks on Kotlini loonud ettevõte JetBrains ning mille eesmärk on lihtsustada veebirakenduste loomist Kotlinis [14]. Ktor on üks tuntumaid spetsiaalselt Kotlini jaoks mõeldud veebiarenduse tarkvararaamistikke. Ktori välistas DeepMOOCi arenduse jaoks asjaolu, et sellel puudub sisseehitatud SAML autentimise tugi [15]. Miinusena märgiti ka OpenAPI dokumentatsiooni automaatse genereerimise toe puudulikkust.

Erinevalt Ktorist on Jooby'sse OpenAPI spetsifikatsiooni automaatse genereerimise tugi sisse ehitatud [11]. See on vajalik, et lihtsustada eesrakenduse loomist: kas selleks, et eesrakenduse arendaja teaks, mis on iga tagarakenduse teenuse (ingl *endpoint*) kasutamise võimalused, või isegi et automaatselt genereerida see osa eesrakenduse koodist, mis vastutab tagarakendusele päringute saatmise eest.

1.2.3 Komapper

Komapper on teek relatsioonandmebaasi ridade ja Kotlini objektide vastendamiseks [16] (ingl *object-relational mapping*, ORM). Selline teegi kasutamine on vajalik selleks, et rakenduse suhtlust andmebaasiga oleks võimalikult lihtne realiseerida. Valituks osutus just Komapper, kuna tegu on võrdlemisi lihtsa spetsiaalselt Kotlini jaoks loodud teegiga, mis täidab kõik DeepMOOCi vajadused.

Töö käigus tehtud valikutest oli andmebaasi teegi valimine kõige aeganõudvam. Tabelis 1 on välja toodud kõik kaalumisel olnud variandid koos leitud miinustega.

Tabel 1. Andmebaasiteegid ja nende miinused.

Teek	*Miinused*
Exposed	• Ei toeta UPSERT andmebaasiinstruktsiooni [17] • Ei toeta massiivi tüüpi andmebaasiveerge [17]
Hibernate	• Ebavajalikult võimas ja keeruline • Java-, mitte Kotlinipõhine [18]
JDBI	• Kasutab puhast SQLi [19] • Java-, mitte Kotlinipõhine [19]
Jimmer	• Väga väike kasutajaskond [20] • Java-, mitte Kotlinipõhine [20]
jOOQ	• Teegi täisversioon on tasuline [21] • Java-, mitte Kotlinipõhine [21]
Komapper	• Väga väike kasutajaskond [22]
Ktorm	• Aeglane [23]
SQLDelight	• PostgreSQL andmebaasi tugi ei ole stabiilne [24]

Ainsad ametliku Jooby poolse toega variandid on Hibernate ja JDBI [11], kuid nende teekide tabelis 1 loetletud miinused olid piisavad, et tekitada soov mõne sobivama lahenduse otsimiseks. Iga ülejäänud valiku puhul oleks olnud vaja Jooby ja vastava teegi integratsioon ise kirjutada. Ühest küljest on tegu lisamiinusega, kuna teegi kasutamine nõuab rohkem tööd. Samas on integratsiooni ise kirjutamine paindlikum ning võimaldab täpsemalt dikteerida rakenduse ja andmebaasi vahelist suhtlust.

Olles välistanud ametliku Jooby toega Hibernate'i ja JDBI, jõuti kõigi ülejäänud teekide miinuseid arvesse võttes järeldusele, et just Komapper on DeepMOOCi jaoks sobivaim lahendus. Samas lihtne see valik ei olnud kuna ka Komapperil on üks suur puudus: vähene kasutajaskond.

Komapperi madal populaarsus võib DeepMOOCi arendamisel osutuda takistuseks kahel erineval viisil:

1) kui teegi kasutamisel peaks ilmnema probleeme, on väljaspool ametlikku dokumentatsiooni sisuliselt võimatu leida internetist abi ja soovitusi;
2) tõenäosus, et teegi arendamine äkitselt ära lõpetatakse, on palju kõrgem.

Siiski, arvestades Komapperi eeliseid alternatiivsete andmebaasiteekide ees, oldi valmis neid riske aktsepteerima ning Komapper sai DeepMOOCi tagarakenduse ja andmebaasi vahelise suhtluse tarbeks kasutusele võetud.

1.2.4 PostgreSQL

PostgreSQL on avatud lähtekoodiga relatsioonandmebaasi haldamise süsteem [25]. Sama andmebaasi kasutati DeepMOOCi tagarakenduse arenduse juures ka varasemalt [1] ning pole ühtegi põhjust, miks seda muuta oleks vaja – PostgreSQL on võimas ning laialdaselt toetatud andmebaasisüsteem, millel on kõik DeepMOOCi juures kasutamiseks vajalikud võimekused.

Käesoleva töö raames on andmebaas vajalik platvormi kasutajate, kursuste, praktikumirühmade, registreerumiste ja muu sellise info hoidmiseks. Tulevikus on platvormi võimekuste kasvades kindlasti vaja ka andmebaasi skeemi täiendada. Samuti hakkavad edaspidi oma info hoidmiseks andmebaasi kasutama DeepMOOCi pistikprogrammid.

2. Töökäik

DeepMOOCi tagarakenduse loomiseks vajalik töö jagunes kolme etappi: eeltöö, rakenduse arendamine, automaattestide kirjutamine. Järgevalt on kirjeldatud igas etapis tehtud tööd ning ületatud probleeme.

2.1 Eeltöö

Eeltöö DeepMOOCi arendamiseks algas tiimikoosolekuga, kus otsustati lõplikult, et tagarakenduse arenduskeeleks saab Kotlin. Varasema kokkupuute puudumise tõttu oli esimese sammuna enne platvormi arendusega alustamist vaja programmeerimiskeele põhitõed selgeks õppida. Õppimisprotsess koosnes põhiliselt õpetusvideote vaatamisest YouTube'is ja Kotlini dokumentatsiooni lugemisest. Tänu Kotlini ja Java sarnasustele ei osutunud Kotlini õppimine keeruliseks – kõige enam pingutust vajas süntaktiliste erinevustega, eelkõige võtmesõnadega harjumine.

Palju suuremaid raskusi valmistas Jooby veebiraamistiku ära õppimine. Varasem veebirakenduse tagarakenduse arendamise kogemus tuli küll Jooby õppimisel kasuks, kuid siiski nõudis uue raamistiku õppimine lisatööd, kuna Jooby's on mitmete vajalike tegevuste rakendamine põhimõtteliselt erinev enamlevinud süsteemidest nagu Spring. Eriti keeruline oli harjuda Jooby moodulitele toetuva konfiguratsiooni ja sõltuvuste süstimise (ingl *dependency injection*) põhimõtetega.

2.2 Arendusprotsess

Arendusetapis toimus rakenduse põhiline loomine: kirjutati erinevate funktsionaalsuste, sealhulgas autentimise ja volitamise realiseerimiseks vajalik kood ning seati üles andmebaas. Samuti täiendati koodi dokumentatsiooni genereerimiseks vajalike kommentaaride ja annotatsioonidega.

Arendusetapini jõudes oli kõige keerulisem otsustada kust alustada – kogu eelnev DeepMOOCi arendus toimus keeles Go [4], mistõttu ei saanud varem tehtut aluseks võtta. Alustamise tegi aga lihtsamaks Jooby autori poolt loodud tööriist, mis automaatselt

genereerib projekti baasi vastavalt kasutaja vajadustele[5]. Genereeritud kood vajas DeepMOOCi tarbeks küll kohati muutmist, kuid oli siiski kasulik.

Enne rakenduse koodi kirjutamisega alustamist tuli üles seada lokaalne PostgreSQL andmebaas ning luua sinna vajalikud tabelid ja nendevahelised seosed ehk relatsioonid. Selleks oli algselt abiks PostgreSQL installeerimisel kaasa tulev pgAdmin 4[6] rakendus, kuid hiljem selgus, et arenduskeskkonnana kasutusel olnud IntelliJ IDEA Ultimate[7] on andmebaasi konfigureerimiseks oluliselt mugavam. Pärast lokaalse andmebaasi loomist sai Jooby rakenduse konfigureerida seda kasutama.

Esimeseks arendussammuks oli SAML põhise autentimissüsteemi kirjutamine. Selleks tuli luua mõned näidisteenused (ingl *endpoint*) kontrollimaks, et nende kasutamiseks nõutakse kasutajalt autentimist. SAML autentimise realiseerimiseks kasutati teeki pac4j[8], mille dokumentatsioon on küll väga põhjalik, kuid autentimissüsteemi korrektseks seadistamiseks tuli siiski aru saada ka sellest, kuidas SAML protokoll sisemiselt töötab.

Erinevalt autentimisest oli volitamissüsteemi loomine võrdlemisi lihtne. Väga kiirelt sai üles seada süsteemi, kus arendajal on võimalik vastavalt vajadusele iga teenuse puhul määrata, kas sellele pääsevad ligi vaid õppejõu õigustes kasutajad või kõik konkreetsel kursusel osalevad isikud.

Siiamaani oli rakenduse ja andmebaasi vahelise suhtluse võimaldamiseks kasutusel ajutine lahendus Hibernate teegi näol, kuna selle tugi on Jooby'sse sisse ehitatud [11]. Nüüd aga tuli kasutusele võtta välja valitud andmebaasiteek Komapper. See läks võrdlemisi kiiresti tänu teegi lihtsusele, põhjalikule dokumentatsioonile ja ka Komapperi ametlikule näidiste lähtekoodihoidlale[9].

Olles kogu vajaliku eeltöö ära teinud sai hakata tegelema DeepMOOCi rakendusliidese teenuste loomisega. Kuna paralleelselt arendatava eesrakenduse täpsed vajadused ei olnud veel teada, pidi eeldama, milliseid teenuseid vaja võiks minna.

---

[5] https://jooby.io/#getting-started

[6] https://www.pgadmin.org/

[7] https://www.jetbrains.com/idea/

[8] https://www.pac4j.org/

[9] https://github.com/komapper/komapper-examples

OpenAPI dokumentatsiooni automaatse genereerimise sisselülitamine oli arendusprotsessi viimane tegevus. OpenAPI spetsifikatsiooniga on defineeritud standard HTTP rakendusliideste kirjeldamiseks [26] ning sellele vastava dokumentatsiooni genereerimise võimekus on Jooby'sse sisse ehitatud. Selle seadistamise protsess oli üldiselt lihtne – tuli vaid mõned annotatsioonid koodile lisada – kuid genereeritud dokumentatsioon tõi välja paar puudust teenuste ülesehituses, mistõttu kulus veidi aega nende korrastamisele.

2.3 Testide koostamine

Viimane tööprotsessi osa oli rakendusele testide kirjutamine. Eelmisel DeepMOOCi versioonil olid vaid manuaalsed testid [1], mille läbiviimine on palju aeganõudvam töö kui automaattestide jooksutamine. Seega otsustati käesoleva töö raames rakendusele luua teenuste automaattestid.

SAML autentimine tegi rakenduse teenuste testimise keeruliseks, kuna automaattesti veebikliendil pole võimalik minna SAML identiteedipakkuja juurde sisse logima. Üks võimalus selle probleemi lahendamiseks oleks olnud testide jooksutamisel autentimine välja lülitada, kuid sellisel juhul ei oleks olnud võimalik pärida testides andmeid vaid ühe konkreetse sisselogitud kasutaja kohta ning samuti poleks õiguste kontroll ehk volitamine enam toiminud. Nende funktsionaalsuste säilitamiseks tuli testimise tarbeks luua eraldi autentimismoodul, mis asendaks SAML autentimist ja kus testide jooksutamisel autenditakse kasutaja teenuse päringu päises oleva kasutajanime järgi.

Testimise tarbeks oli algselt lootus kasutada andmebaasitehinguid (ingl *transaction*), et andmebaasi tehtud muudatused testi lõppedes tühistada ehk andmebaas lähtestada. Idee aga ei töötanud, kuna testide lõimes (ingl *thread*) alustatud Komapperi tehingud ei suutnud rakenduse lõimedes tehtud muudatusi tagasi keerata. Seega tuli leppida klassikalise lähtestamislahendusega, kus iga test tühistab enne lõppemist enda tehtud muudatused manuaalselt.

3. Tulemused

Järgnevates peatükkides tutvustatakse täpsemalt töö raames valminud tagarakenduse funktsionaalsusi ja ülesehitust. Tuuakse välja valminud autentimis- ja volitamissüsteemi ülesehitus ning kirjeldatakse nendega kaitstud teenuseid. Lõpus mainitakse praeguse versiooni puuduseid ning tehakse soovitusi rakenduse edasi arendamiseks.

3.1 Autentimine

Kasutajate tuvastamiseks kirjutati DeepMOOC rakendusse SAML 2.0 protokolli kasutav autentimismoodul. Järgnevalt kirjeldatakse täpsemalt loodud lahenduse omadusi ning nendest saadavat kasu.

Rakenduse kõikide sisuliste teenuste kasutamiseks on eelnev autentimine nõutud – ainult mõned kõrvalisemad teenused, nagu näiteks OpenAPI dokumentatsiooni vaatamine, ei nõua isikutuvastust. Autentimine toimub esimese päringu tegemisel mõne tagarakenduse veebiliidese teenuse vastu, mispuhul suunatakse kasutaja oma isiku verifitseerimiseks ümber SAML identiteedipakkuja juurde. Pärast edukat sisselogimist suunatakse kasutaja tagasi DeepMOOC rakendusse, kus saab päritud teenus nüüd käivituda.

DeepMOOCi lõppversioonis on küll kavas kasutada Tartu Ülikooli SAML identiteedipakkujat, kuid käeoleva töö käigus võeti selle asemel appi SAMLtest[10] poolt pakutav teenus. SAMLtest pakub võimalust ühendada mingi SAML teenusepakkuja ehk antud juhul DeepMOOC tagarakendus nende testidentiteedipakkujaga ning kasutada seda autentimiseks. Rakenduse arendusstaadiumis on testidentiteedipakkuja kasutamisel mitmeid eeliseid:

- võimalik on tegutseda iseseisvalt ja kiiremini – arendustöö ei jää ülikooli identiteedipakkujat haldava IT-meeskonna taha seisma;
- testkasutajatega opereerimise võimalus – käsitlema ei pea nii öelda pärisandmeid.

Loodud SAML autentimismoodul on lihtsasti seadistatav, kuna võtab kõik oma tööks vajalikud parameetrid rakenduse konfiguratsioonifailist. See tähendab, et tulevikus ülikooli

[10] https://samltest.id/

identiteedipakkuja kasutusele võtmisel tulebki vaid rakenduse seadistust uuendada – koodi muutmine vajalik ei ole.

3.2 Volitamine ja õiguste kontroll

DeepMOOC rakenduse puhul kasutati kaheosalist pääsulubade süsteemi. Esimene osa on volitamine, mille käigus kasutajale antakse õigused mingi kindla hulga teenuste kasutamiseks. Teine pool on õiguste kontroll ehk protsess, mille käigus tuvastatakse, kas kasutajal on päritud teenuse kasutamiseks luba.

Volitamise jaoks on vajalik kasutaja tuvastamine, seega toimub see kohe pärast autentimist. Volitamisel päritakse andmebaasist kursused, millega kasutaja on seotud ning vastavalt tulemile genereeritakse pääsuload ehk õigused. Seega on kasutajal ligipääsuõigus vaid nendele kursustele, millele ta on registreerunud.

Kasutajate pääsuload koosnevad kahest osast: juurdepääsutasemest, mis on kas „tudeng“ või „õppejõud“, ja kursuse koodist. Kursuse koodi sisaldumine lubades on vajalik võimaldamaks olukorda, kus mõni kasutaja on ühel kursusel õppejõud ning teisel tudeng.

Ühe kursuse kohta võib kasutajal olla ka mitu pääsuluba. Volitamise käigus genereeritakse kasutajale iga kursuse kohta load kõige madalamast juurdepääsutasemest kuni andmebaasis määratud juurdepääsutasemeni. Näiteks kui kasutajal on andmebaasis määratud õppejõu juurdepääsutase, saab ta nii õppejõu kui ka tudengi tasemele vastava loa. See tagab, et õppejõu õigustes kasutajal on siiski ligipääs ka madalamat pääsuluba nõudvatele teenustele.

Pääsulubade kontroll toimub vahetarkvara (ingl *middleware*) põhimõttel. See tähendab, et valiidset päringut tehes käivitub esmalt kontrollfunktsioon, mis tuvastab, kas päringu teinud kasutajal on piisavad õigused käesoleva teenuse kasutamiseks. Kontrollfunktsiooni rakendamine toimub automaatselt ning arendajal tuleb vaid määrata teenuse minimaalne juurdepääsutase. Valiidne päring tähendab siinjuures seda, et teenus, mida kasutada üritatakse, on päriselt olemas – kui teenust ei eksisteeri, siis õiguste kontrolli ei toimu.

Vajaliku loa olemasolul käivitatakse teenuse kood ning kasutaja saab oma päringule oodatud vastuse. Kui kasutajal puudub luba teenust kasutada, siis on päringu vastuseks olekukood 403 *Forbidden* ehk keelatud. Sellisel juhul teenuse koodi ei käivitata.

3.3 Kasutajate haldus

Käesoleva bakalaureusetöö raames oli DeepMOOCi veebiliidese (ingl *web API*) arendamisel põhirõhk kasutajate haldusega tegelevatel teenustel. Valminud teenused keskenduvad kasutajate ja kursuste vaheliste seoste loomisele, kustutamisele ning väljastamisele. Järgnevalt kirjeldatakse loodud teenuseid ning toetatud päringuid.

Siin peatükis toodud teenuseid saavad kasutada vaid autenditud kasutajad. See on vajalik, sest enda kohta käiva info pärimiseks peab rakendus teadma, kes päringu tegija on, ning teisi kasutajaid hõlmavate tegevuste puhul on vaja eelnevalt läbi viia õiguste kontroll.

Loodud teenused saab jaotada kaheks. Esimest liiki ehk üldised teenused on kirjeldatud tabelis 2. Nendel teenusel lisanduvaid ligipääsupiiranguid ei ole.

Tabel 2. Üldised teenused.

Selgitus	*Parameetrid*	*Tulem*
Enda info päring	-	Päringu tegija info
Enda kursuste päring	-	Kursused, millega päringu tegija on seotud, koos juurdepääsutasemetega

Tabelis 3 toodud teenused on seotud konkreetse kursusega. Teenust kasutades peab päringu tee (ingl *path*) sisaldama kursuse koodi. Seega lisaks tabelis märgitud parameetritele on ka kursuse kood päringu tegemisel alati vajalik.

Tabel 3. Konkreetse kursusega seotud teenused.

Selgitus	*Parameetrid*	*Tulem*
Enda rühmade päring	-	Käesoleva kursuse rühmad, millesse päringu tegija kuulub
Teise kasutaja rühmade päring	Kasutaja ID	Käesoleva kursuse rühmad, millesse kasutaja kuulub
Registreeritud kasutajate päring	-	Kasutajad, kes on käesoleva kursusega seotud, koos juurdepääsutasemetega

Registreeritud tudengite päring	-	Kasutajad, kes on käesoleva kursusega seotud ja kelle juurdepääsutase on „tudeng“
Kasutajate lisamine kursusele	Lisatavate kasutajate ID-d Juurdepääsutase, mis lisatavatele kasutajatele anda	Kasutajad, kes just registreeriti kursusele, koos juurdepääsutasemetega
Kasutajate eemaldamine kursuselt	Eemaldatavate kasutajate ID-d	-
Kasutajate lisamine rühma	Lisatavate kasutajate ID-d Rühma ID	Kasutajad, kes just lisati rühma
Kasutajate eemaldamine rühmast	Eemaldatavate kasutajate ID-d Rühma ID	-

Kõigil tabelis 3 mainitud teenustel peale esimese on kõrgema taseme juurdepääsupiirang. Kuna operatsioone tehakse teiste kasutajatega, siis on nende teenuste rakendamisel nõutud, et kasutaja oleks vastava kursuse juures vähemalt õppejõu juurdepääsutasemega.

Viimasel kahel tabelis 3 loetletud teenusel on veel kaks lisapiirangut. Esiteks peab ID-ga kirjeldatud rühm kuuluma päringu tees spetsifitseeritud kursuse juurde. Teiseks peab kasutaja olema juba vastavale kursusele registreeritud. Päring ebaõnnestub, kui vähemalt üks nendest tingimustest pole täidetud.

Järgnevalt kirjeldatakse lühidalt olekukoode (ingl *status code*), mis sisalduvad teenuse-päringute vastustes. Eduka päringu vastuse olekukood on sõltuvalt teenusest kas 200 *OK* või 204 *No Content*. Neist teist ehk „sisu pole“ koodi kasutatakse nende tabelis 3 mainitud teenuste puhul, kus tulemiveerg on tühi. Ülejäänud teenuste puhul tähistab edukat päringut kood 200.

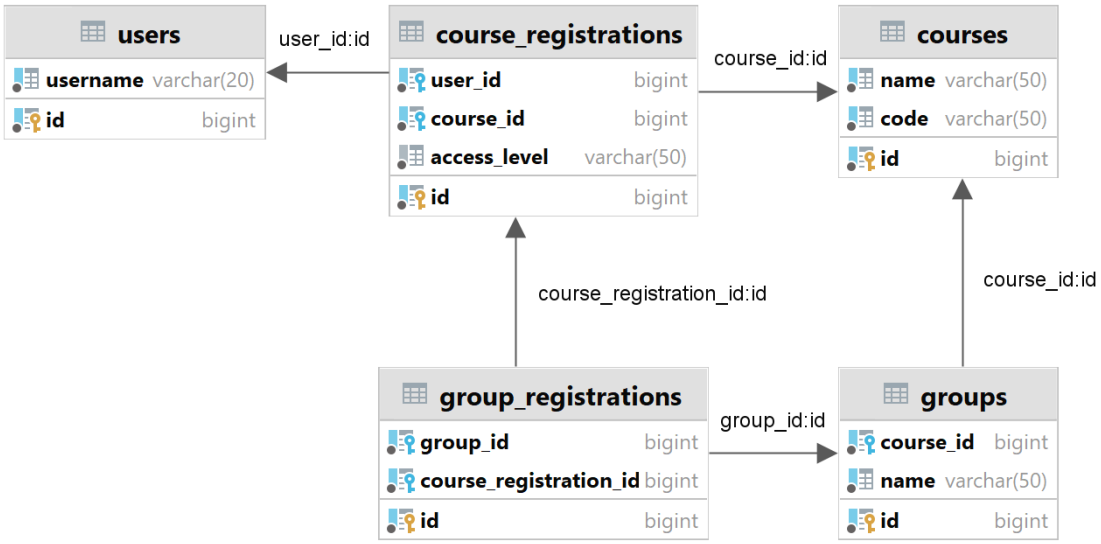
Luhtunud päringu puhul sõltub vastuse olekukood ebaõnnestumise põhjusest:

- 400 *Bad Request* ehk halb päring juhul, kui ebaõnnestumises on süüdi päringu tegija, näiteks mõni nõutud parameeter on puudu või ebasobiv;
- 500 *Internal Server Error* ehk serverisisene viga juhul, kui ebaõnnestumises on süüdi tagarakendus ise – teenuse kasutamisel juhtus midagi ootamatut.

Üks teenus otsustati ka realiseerimata jätta – uue kasutaja loomine. Tegemist oli teadliku valikuga. Kasutajate loomise ainuõigus saab olema tulevikus loodaval integratisooni-moodulil, mis hakkab välisest allikast, näiteks õppeinfosüsteemist, pärima infot tudengite ja nende kursustele registreerumiste kohta. Seega pole eesrakendusele kättesaadavat kasutaja lisamise teenust vaja. Samas tähendab teenuse puudumine rakenduse praeguses arendus-staadiumis seda, et kasutajaid on võimalik luua vaid otse andmebaasis.

3.4 Andmebaasiskeem

DeepMOOCi tarbeks loodud andmebaas koosneb viiest tabelist. Skeem on pigem minimaalne, kuna tabelid sisaldavad hetkel vaid loodud funktsionaalsuste implementeerimiseks vajalikke veerge, mis tähendab, et edasise arenduse käigus on tõenäoliselt vaja nende ülesehitust täiendada. Andmebaasitabelite struktuur ja omavahelised seosed on kujutatud joonisel 1.



Joonis 1. Andmebaasi struktuur.

Järgnevalt on lühidalt selgitatud iga tabeli sisu:

- *users* – info platvormi kasutajate kohta;
- *course_registrations* – kursustele registreeritud kasutajad, sealhulgas nende juurdepääsutase;
- *courses* – kursuste üldinfo;
- *group_registrations* – kasutajate rühmadesse kuuluvus;
- *groups* – praktikumi- ja seminarirühmade üldinfo, sealhulgas viide seotud kursusele.

Iga tabel sisaldab ka arvulist primaarvõtit. Eelmine versioon DeepMOOCi andmebaasist kasutas kasutajate tabeli (*users*) primaarvõtmena kasutajanime (*username*) [1]. Selline lähenemine teeb kasutajanime muutmise protsessi ebavajalikult keeruliseks, mistõttu otsustati nüüd ka kasutajate tabelis üle minna arvulisele primaarvõtmele.

3.5 Dokumentatsioon

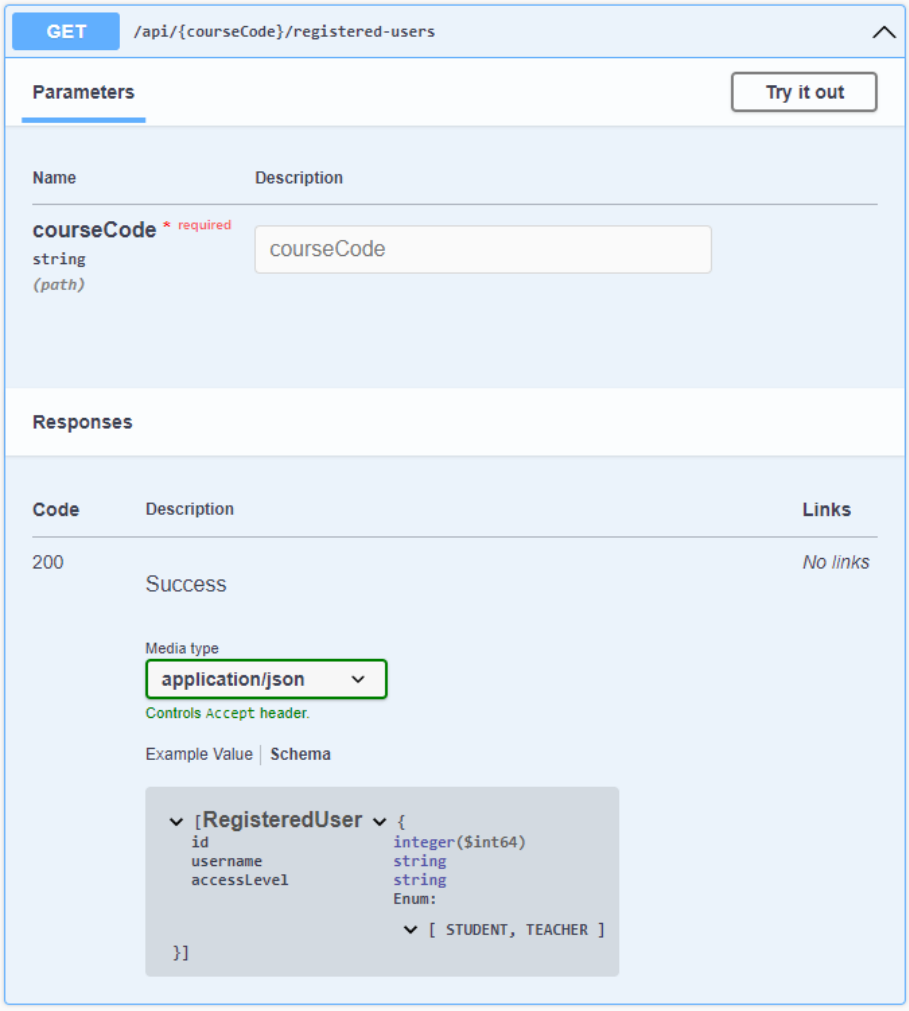
Töö raames genereeriti tagarakenduse teenustele OpenAPI spetsifikatsioonile vastav dokumentatsioon. OpenAPI dokumentatsioonist on eelkõige kasu DeepMOOCi ees-rakenduse arendajatel. Dokumentatsioon on hea koht tutvuda tagarakenduse võimalustega – iga veebiteenuse nõutavate parameetrite ja väljastatavate andmetega – ilma, et oleks vaja süveneda tagarakenduse programmikoodi.

Tänu Jooby ametlikule OpenAPI moodulile[11] toimub rakenduse käivitamisel dokumentatsiooni genereerimine automaatselt. Seega kajastuvad ka tulevikus lisanduvad teenused automaatselt dokumentatsioonis.

Dokumentatsioon sisaldab endas kõikide loodud veebiteenuste struktureeritud kirjeldust, sealhulgas teenuse:

1) teed (ingl *path*);
2) päringu HTTP meetodit, näiteks GET, POST, DELETE;
3) oodatavaid parameetreid ning nende kuju;
4) päringu vastuse võimalikke olekukoode, sealhulgas nendega kaasneva sisu kuju.

[11] https://jooby.io/modules/openapi/



Joonis 2. Teenuse OpenAPI dokumentatsioon Swagger UI-s.

Dokumentatsiooni visualiseerimiseks võeti kasutusele ka tööriist Swagger UI[12]. OpenAPI dokumentatsioon on JSON formaadis, mistõttu pole see puhtal kujul inimeste jaoks kõige kasutajasõbralikum. Swagger UI lahendab selle probleemi, esitades dokumentatsiooni visuaalsel ja interaktiivsel kujul, mida näitlikustab joonis 2. See tähendab, et lisaks OpenAPI dokumentatsioonis kajastuva info edastamisele genereerib Swagger UI ka näidispäringud kõigi tagarakenduse veebiteenuste kasutamiseks ning võimaldab testimiseesmärgil päringuid ka reaalselt käivitada. Need funktsionaalsused teevad DeepMOOCi rakendus-liidese kasutamise tunduvalt mugavamaks, kuna võimaldavad eesrakenduse arendajal lihtsamini teenuseid tundma õppida.

[12] https://swagger.io/tools/swagger-ui/

3.6 Tulevane funktsionaalsus

Mitmed olulised DeepMOOCi tagarakenduse funktsionaalsused jäid ka käesoleva töö raamidest välja. Sinna kuulub nii arhitektuurilises mõttes suuri osi, mis tuleb rakendusele juurde integreerida, kui ka viise olemasoleva koodi parendamiseks.

Kõige kriitilisem täiendus, mida selle töö raames kirjutatud kood vajab, on integratsioon Tartu Ülikooli SAML identiteedipakkujaga – ilma selleta pole võimalik rakendust kasutusele võtta. Kuigi muudatus on oluline, ei ole see tehniliselt kuigi keeruline – tuleb vaid rakenduse seadistust muuta. Muudatusega pole ka kiire, kuna testidentiteedipakkuja kasutamine ei takista ühegi teise DeepMOOCi funktsionaalsuse arendamist. Identiteedipakkuja vahetus ei ole samas ühepoolne tegevus ning selle käigus tuleb kindlasti teha koostööd ülikooli IT-meeskonnaga.

Üks vähemoluline, kuid siiski ära märkimist vääriv aspekt, mis töö käigus tähelepanuta jäi, on logimine. Hetkel logitakse vaid ebakorrektsed sissetulevad teenusepäringud ja käivitatavad andmebaasiinstruktsioonid. Praeguseks on see piisav, kuna rakenduse koodibaas on veel piisavalt väike, et selles on võimalik lihtsasti orienteeruda ja vigade ilmumisel nende allikad leida. Edasise arenduse käigus tuleks aga kasuks rakenduse täiendamine logimiskäskudega.

Ülalmainitud kaks soovitust rakenduse edasi arendamiseks olid otseselt seotud käesoleva töö raames kirjutatud koodiga. Samas on mitmeid kõrvalisi funktsionaalsuseid, mis tuleb ka rakendusse enne selle kasutuselevõttu integreerida. Nendeks on põhiliselt ülesannete lahenduste ehk programmikoodi esitamise ja automaatkonttrollide tugi, pistikprogrammide süsteem ning integratsioonimoodul tudengite info DeepMOOCi importimiseks. Loetletud funktsionaalsuste realiseerimise eest vastutavad aga juba järgmised DeepMOOCi arendajad.

4. Testimine

Loodud rakenduse töökorra tagamiseks on tähtis, et võimalikult suur osa koodist oleks kaetud testidega. Rakenduse testimisel oli fookus veebiliidese korrektsel toimimisel, mistõttu on loodud testide keskmes teenusepäringute tegemine ning saadud vastuste kontrollimine.

4.1 Taust

Rakenduse teenuste korrektsuse verifitseerimiseks kasutati automaatteste. Automaattestide algse ülesseadmise protsess on küll oluliselt keerulisem kui manuaalsete testide puhul, kuid neid on pärast ka palju kiirem ja mugavam läbi viia. Seega on loodud automaattestid kindlasti kasulikuks tööriistaks rakenduse edasise arendustöö käigus.

Automaattestide jooksutamisel ei kasutata rakenduse tavalist, SAML põhist autentimismeetodit. SAML autentimismooduli käikulaskmise asemel lisatakse igasse päringusse spetsiaalne päis enda kasutajanimega. Tegemist pole küll kuigi turvalise autentimismeetodiga, kuid see võimaldab testimisel vältida keerulist ja ebavajalikku SAML autentimisprotsessi. Antud päisepõhise autentimissüsteemi eeliseks on ka see, et erinevate õigustega kasutajatega on testide käitamine väga lihtne.

Automaattestide käitamisel tuleb eelnevalt ka andmebaas initsialiseerida. See toimub samuti automaatselt – enne testide jooksutamist sisestatakse vajalikud andmed andmebaasi. Sellega välditakse olukorda, kus iga test peaks ise looma andmebaasi enda tööks vajalikud kirjed.

4.2 Automaattestid

Igale kasutajate haldusega tegelevale teenusele loodi üks või mitu automaattesti. Loodud testid rakendavad kolme eri tüüpi päringuid:

1) korrektne päring – kõik parameetrid on sobivad, päring õnnestub (vastuse olekukood 200 *OK* või 204 *No Content*);
2) volitamata päring – kõik parameetrid sobivad, kuid kasutajal puudub luba vastava teenuse kasutamiseks, päring ebaõnnestub (vastuse olekukood 403 *Forbidden*);
3) vigane päring – parameetrid on ebakorrektsed või puudu, päring ebaõnnestub (vastuse olekukood 400 *Bad Request*).

Siinjuures tasub märkida, et kuna testimisel kasutatakse alternatiivset autentimismeetodit, siis autentimata päringute testimisest kasu ei oleks ning seega seda ka ei tehta.

Automaattestidega on kaetud kõik tabelites 2 ja 3 loetletud teenused. Järgnevalt kirjeldatakse nende teenusete verifitseerimiseks realiseeritud testjuhute.

Üldised ehk tabelis 2 nimetatud teenused ei nõua ühtegi parameetrit ning nende kasutamisel ei toimu ka õiguste kontrolli. See tähendab, et ebakorrektset ega volitamata päringut pole võimalik nende teenuste jaoks konstrueerida. Mõlema teenuse jaoks realiseeriti seetõttu vaid korrektset päringut teostav testjuht.

Tabelis 3 loetletud ehk konkreetse kursusega seotud teenuste kontrollimiseks oli vaja luua tunduvalt rohkem testjuhte. Nendele teenustele vastavad testjuhud on kirjeldatud tabelis 4.

Tabel 4. Testjuhud konkreetse kursusega seotud teenuste verifitseerimiseks.

Teenus	**Testi kirjeldus**	**Tüüp**
Enda rühmade päring	Kasutaja küsib nimekirja oma rühmadest	Korrektne
Teise kasutaja rühmade päring	Õppejõud küsib nimekirja kasutaja rühmadest	Korrektne
	Tudeng küsib nimekirja kasutaja rühmadest	Volitamata
	Õppejõud küsib nimekirja mitteeksisteeriva kasutaja rühmadest	Vigane
Registreeritud kasutajate päring	Õppejõud küsib nimekirja kursusele registreeritud kasutajatest	Korrektne
	Tudeng küsib nimekirja kursusele registreeritud kasutajatest	Volitamata
Registreeritud tudengite päring	Õppejõud küsib nimekirja kursusele registreeritud tudengitest	Korrektne
	Tudeng küsib nimekirja kursusele registreeritud tudengitest	Volitamata

Kasutajate lisamine kursusele	Õppejõud lisab kasutaja kursusele tudengi juurdepääsutasemega	Korrektne
	Õppejõud lisab kasutaja kursusele õppejõu juurdepääsutasemega	Korrektne
	Õppejõud lisab kursusele kasutaja, kes juba on sellele kursusele registreeritud	Korrektne
	Tudeng lisab kasutaja kursusele	Volitamata
	Õppejõud lisab mitteeksisteeriva kasutaja kursusele	Vigane
	Õppejõud lisab kasutaja kursusele mitteeksisteeriva juurdepääsutasemega	Vigane
	Õppejõud lisab kasutaja kursusele ilma juurdepääsutasemeta	Vigane
Kasutajate eemaldamine kursuselt	Õppejõud eemaldab kursuselt tudengi juurdepääsutasemega kasutaja	Korrektne
	Õppejõud eemaldab kursuselt õppejõu juurdepääsutasemega kasutaja	Korrektne
	Õppejõud eemaldab kursuselt kasutaja, kes pole sinna registreeritud	Korrektne
	Tudeng eemaldab kasutaja kursuselt	Volitamata
	Õppejõud eemaldab mitteeksisteeriva kasutaja kursuselt	Vigane
Kasutajate lisamine rühma	Õppejõud lisab kasutaja rühma	Korrektne
	Õppejõud lisab rühma kasutaja, kes juba on selles rühmas	Korrektne
	Tudeng lisab kasutaja rühma	Volitamata
	Õppejõud lisab rühma kasutaja, kes pole kursusele registreeritud	Vigane
	Õppejõud lisab kasutaja rühma, mis pole käesoleva kursusega seotud	Vigane
	Õppejõud lisab rühma mitteeksisteeriva kasutaja	Vigane

Kasutajate eemaldamine rühmast	Õppejõud eemaldab kasutaja rühmast	Korrektne
	Õppejõud eemaldab rühmast kasutaja, kes pole sinna registreeritud	Korrektne
	Tudeng eemaldab kasutaja rühmast	Volitamata
	Õppejõud eemaldab rühmast kasutaja, kes pole kursusele registreeritud	Vigane
	Õppejõud eemaldab kasutaja rühmast, mis pole käesoleva kursusega seotud	Vigane
	Õppejõud eemaldab rühmast mitteeksisteeriva kasutaja	Vigane

Kõigil konkreetse kursusega seotud teenustel on ka üks ühine testjuht: päring on vigane, kui etteantud kursusekoodile vastavat kursust pole olemas. Samas pole ühelgi kasutajal õigust sellise kursusega seotud teenuseid kasutada, mistõttu on tegelikkuses tegu volitamata päringuga. Kuna see testjuht on iga teenuse puhul samasugune, siis ei olnud kordamiseks põhjust ja see reaaliseeriti ühekordselt.

Kokkuvõte

Bakalaureusetöö eesmärk oli luua DeepMOOC platvormile tagarakendus rõhuasetusega kasutajate haldusel. Selleks loodi tagarakendusele veebiliides kümne teenusega, mis võimaldavad platvormi kasutajate kohta andmeid pärida ning teha muudatusi nende kursustele ja rühmadesse registreerituses. Rakenduse töökindluse tagamiseks loodi ka automaattestid, millega on mugav kontrollida veebiliidese teenuste toimimist.

Teine eesmärk ehk autentimis- ja volitamissüsteemi loomine sai samuti täidetud. Kasutajate autentimiseks loodi SAML standardile vastav lahendus, tänu millele on tulevikus võimalik DeepMOOCi sisse logida Tartu Ülikooli kasutajanime ja parooliga. Rajatud pääsulubade süsteem tagab, et teenustele pääsevad ligi vaid volitatud kasutajad.

Autori hinnangul on loodud tagarakendus tugev baas edasiseks platvormi arendustööks, sealhulgas väliste komponentide, nagu automaatkontrollide süsteem, integreerimiseks. Platvormi edasine arendus toimub tulevaste bakalaureusetööde käigus ning võiks tänu olemasolevale koodile ja loodud dokumentatsioonile olla märgatavalt lihtsam.

Viidatud kirjandus

[1] J. Näks, DeepMOOC platvormile tagarakenduse dispetšeri arendamine, Bakalaureusetöö, Tartu Ülikool, Arvutiteaduse instituut, 2022.

[2] OpenID Foundation, „OpenID Connect FAQ and Q&As,“ https://openid.net/connect/faq/. [Kasutatud 18.04.2023].

[3] OASIS Security Services Technical Committee, „FAQ,“ https://www.oasis-open.org/committees/security/faq.php. [Kasutatud 19.04.2023].

[4] Kotlini lähtekoodihoidla, https://github.com/JetBrains/kotlin/blob/master/ReadMe.md. [Kasutatud 17.04.2023].

[5] D. Jemerov, „Why JetBrains needs Kotlin,“ 2.8.2011. https://blog.jetbrains.com/kotlin/2011/08/why-jetbrains-needs-kotlin/. [Kasutatud 10.01.2023].

[6] C. Haase, „Google I/O 2019: Empowering developers to build the best experiences on Android + Play,“ 07.06.2019. https://android-developers.googleblog.com/2019/05/google-io-2019-empowering-developers-to-build-experiences-on-Android-Play.html. [Kasutatud 12.01.2023].

[7] Kotlini dokumentatsioon, „Kotlin for server side,“ https://kotlinlang.org/docs/server-overview.html. [Kasutatud 12.01.2023].

[8] Go keele tarkvarapaketi "plugin" dokumentatsioon, https://pkg.go.dev/plugin. [Kasutatud 17.04.2023].

[9] Kotlini dokumentatsioon, „Dynamic type,“ https://kotlinlang.org/docs/dynamic-type.html. [Kasutatud 17.04.2023].

[10] Kotlini dokumentatsioon, „Basic syntax,“ https://kotlinlang.org/docs/basic-syntax.html. [Kasutatud 17.04.2023].

[11] Jooby dokumentatsioon, https://jooby.io/. [Kasutatud 19.04.2023].

[12] Jooby lähtekoodihoidla, https://github.com/jooby-project/jooby. [Kasutatud 19.04.2023].

[13] A. Arhipov, „Server-Side Development with Kotlin: Frameworks and Libraries,“ 20.11.2020. https://blog.jetbrains.com/kotlin/2020/11/server-side-development-with-kotlin-frameworks-and-libraries/. [Kasutatud 19.04.2023].

[14] Ktori lähtekoodihoidla, https://github.com/ktorio/ktor/blob/main/README.md. [Kasutatud 10.01.2023].

[15] Ktori dokumentatsioon, „Supported authentication types,“ https://ktor.io/docs/authentication.html#supported. [Kasutatud 19.04.2023].

[16] Komapperi dokumentatsioon, https://www.komapper.org/docs/overview/. [Kasutatud 19.04.2023].

[17] Exposed lähtekoodihoidla, „FAQ,“ https://github.com/JetBrains/Exposed/wiki/FAQ. [Kasutatud 21.04.2023].

[18] Hibernate lähtekoodihoidla, https://github.com/hibernate/hibernate-orm. [Kasutatud 21.04.2023].

[19] JDBI dokumntatsioon, https://jdbi.org/. [Kasutatud 21.04.2023].

[20] Jimmeri lähtekoodihoidla, https://github.com/babyfish-ct/jimmer. [Kasutatud 21.04.2023].

[21] jOOQ koduleht, https://www.jooq.org/download/. [Kasutatud 21.04.2023].

[22] Komapperi lähtekoodihoidla, https://github.com/komapper/komapper. [Kasutatud 21.04.2023].

[23] Jimmeri dokumentatsioon, „Benchmark,“ https://babyfish-ct.github.io/jimmer/docs/benchmark. [Kasutatud 21.04.2023].

[24] SQLDelighti dokumentatsioon, „PostgreSQL (JVM),“ https://cashapp.github.io/sqldelight/1.5.4/jvm_postgresql/. [Kasutatud 21.04.2023].

[25] PostgreSQL koduleht, https://www.postgresql.org/. [Kasutatud 19.04.2023].

[26] OpenAPI spetsifikatsioon, https://spec.openapis.org/oas/latest.html. [Kasutatud 23.04.2023].

[27] TechEmpower, „Web Framework Benchmarks,“ https://www.techempower.com/benchmarks/#hw=ph&test=composite&section=data-r21&l=xan9tr-6bj. [Kasutatud 19.04.2023].

Lisad

I. Lähtekood

Tööga on kaasas loodud rakenduse lähtekood, mille leiab failist „lahtekood.zip“.

Lisaks lähtekoodile sisaldab zip-konteiner faili „README.md“, milles on tagarakenduse käivitamisjuhend. Seal tuuakse välja nii eeldused kui ka sammud andmebaasi ja rakenduse seadistamiseks. Antud juhend on eelkõige suunatud DeepMOOC platvormi tulevastele arendajatele lokaalse arenduskeskkonna ülesseadmiseks.

II. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Märt Tender

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **DeepMOOC platvormile tagarakenduse arendamine**, mille juhendajad on Ahti Põder ja Tõnis Hendrik Hlebnikov, 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.

Märt Tender

05.05.2023