TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Ragnar Rebase 164036IAPB

LÄHTEKOODI SARNASUST TUVASTAVA SÜSTEEMI ARENDAMINE

Bakalaureusetöö

Juhendaja: Ago Luberg, MSc

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Ragnar Rebase

05.05.2019

Annotatsioon

Antud töös luuakse veebirakendus, mis võimaldab mugavalt kontrollida plagiaati Tallinna Tehnikaülikoolis toimuvatel programmeerimise kursustel. Olemasolev protsess nõuab lisatööd ja pole ülevaatlik. Puudub igasugune võimalus kasutajaid hallata ja tulemusi organiseerida.

Töö eesmärgiks on lihtsustada ja kiirendada õppejõudude tööd plagiaadi kontrollimisel. Selleks kogutakse automaatselt GitLabist kokku tudengite esitused ja saadetakse need koodi sarnasust tuvastavasse süsteemi Moss. Saadud tulemused seotakse konkreetse tudengi, projekti, kursuse ja ülesandega. Tuvastatakse korduvalt plagiaadiga vahele jäänud tudengeid nii ülesannete kui ka kursuste lõikes. Võimaldatakse erinevate õiguste tasemetega kasutajate loomist. Tavakasutajatel on ligipääs ainult nendele kursustele, kus kasutaja on märgitud õppejõuks.

Töö tulemust valideeritakse selle reaalse kasutamisega 2019. aasta kevadel toimuval kursusel "Programmeerimise põhikursus" (ITI0202). Veebirakendus on kättesaadav aadressil https://plagiaat.cs.ttu.ee.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 25 leheküljel, 6 peatükki, 5 joonist, 1 tabel.

Abstract

Development of a System for Detecting Software Similarity

The aim of this thesis is to develop a plagiarism detection solution that can be comfortably used for Tallinn University of Technology programming courses. The current solution requires extra work and does not give a good overview. There is a complete lack of application user management.

The web application will automatically gather the student solutions from GitLab and send them to Moss, which is a system for detecting software similarity. It gives results ordered by similarity, which is parsed by the application. The matches are connected with a specific student, project, course and assignment. It will be possible to identify students who have been caught with plagiarism multiple times across different assignments and courses. The application provides user management capabilities, which allows giving limited course specific read-only access to teaching assistants.

The application is developed using popular frameworks Django and React while focusing on maintainability and best practices by using the best available tools known to the author. All of the dev-ops is automated and documented making deploys fast and easy. The project is dockerised, which means that all services run inside of Docker containers and are orchestrated using Docker Compose. Making it incredibly easy to set up a local development regardless of the platform. Moving the project between servers requires little effort – users, permissions, security settings and underlying infrastructure is automatically set up using Ansible and Docker provides compatibility.

The developed solution will be validated by using it in the 2019 spring course “Computer programming” (ITI0202). The web application is available at https://plagiaat.cs.ttu.ee.

The thesis is in Estonian and contains 25 pages of text, 6 chapters, 5 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application programming interface*, rakendusliides
DOM	*Document Object Model*
Git	Versioonihaldustarkvara
HTML	*Hyper Text Markup*
HTTP	*Hypettext Transfer Protocol*
JSON	*JavaScript Object Notation*
JSX	*JavaScript Extensible Markup Language*
LTS	*Long Term Support*, pikaajaline tugi
MVT	*Model-View-Template*
ORM	*Object-relational mapping*
OS	*Operating System*, operatsioonisüsteem
PEP	*Python Enhancement Proposal*
SPA	*Single-page application*
SQL	*Structured Query Language*, struktuurpäringukeel
SSH	*Secure Shell*, turvakest
SVG	*Scalable Vector Graphics*

Sisukord

1 Sissejuhatus 11
2 Analüüs 12
2.1 Süsteemi vajadused 12
2.2 Andmebaas 13
2.2.1 Mudelid 13
2.2.2 Päringud 14
3 Rakenduse realisatsioon 16
3.1 IT-taristu ja selle haldus 16
3.1.1 Ansible 16
3.1.2 Fabric 17
3.1.3 Nginx 17
3.1.4 Docker 17
3.1.5 GitLab CI 17
3.2 *Back-end* raamistik ja pakid 17
3.2.1 Django 18
3.2.2 Django REST framework 18
3.2.3 Pytest 18
3.2.4 WebSocket 18
3.2.5 Celery 19
3.2.6 Redis 19
3.2.7 PostgreSQL 19
3.3 *Front-end* raamistik ja pakid 19
3.3.1 React 19
3.3.2 Formik 20
3.3.3 Redux 20
3.3.4 Babel 20
3.3.5 ESLint 20
3.3.6 SCSS 20
3.3.7 Bootstrap 20

3.3.8 Webpack 21
4 Rakenduse realisatsioon 22
4.1 Funktsionaalsus 22
4.1.1 Uni-ID-ga autentimine 22
4.1.2 Kursuste loomine, muutmine ja kustutamine 22
4.1.3 Ülesannete loomine, muutmine ja kustutamine 23
4.1.4 Projektide vaade 23
4.1.5 Logide vaade 24
4.1.6 Ülesande ülevaade 24
4.1.7 Plagiaadikontrolli vastete tabel 25
4.1.8 Tudengi profiili vaade 25
4.2 Struktuur 25
4.2.1 Dot-failid ja konfiguratsioonifailid 25
4.2.2 Django struktuur 26
4.2.3 Reacti struktuur 27
4.3 GitLabi kasutamine 29
4.3.1 Ülesanded 29
4.3.2 Harud 29
4.4 Plagiaadi kontrollimine 29
4.4.1 Eeldused 30
4.4.2 Protsess 30
5 Arengusuunad 34
5.1 Alternatiivse plagiaaditeenuse kasutamine 34
5.2 Plagiaadi otsimine veebist ja eelnevate aastate lahendustest 34
5.3 Sõltumatus GitLabist 34
6 Kokkuvõte 36
Kasutatud kirjandus 37
Lisa 1 – Plagiaadi ja kursuse rakenduse mudelid 38
Lisa 2 – Kõikide rakenduste mudelid 39
Lisa 3 – Kursuste vaade 40
Lisa 4 – Kursuse loomise vorm 41
Lisa 5 – Kursuse ülesannete vaade 42
Lisa 6 – Ülesande loomise vorm 43
Lisa 7 – Kursuse GitLabi projektid 44

Lisa 8 – Kursuse logid 45
Lisa 9 – Ülesande ülevaatlik vaade 46
Lisa 10 – Kursuse õppejõudude muutmise modaalaken 47
Lisa 11 – Plagiaadikontrolli tulemuste vaade 48
Lisa 12 – Vaste rakendusliidese vaatekomplekt 49

Jooniste loetelu

Joonis 1. Django ORM *query* näide. 15
Joonis 2. Kursuse ülesannete vaade. 23
Joonis 3. Ülesande ülevaatlik vaade 24
Joonis 4 Lihtsustatud näide *reducer*'ite kombineerimisest 28
Joonis 5 Plagiaadi kontrolli etapid. 31

Tabelite loetelu

Tabel 1. Järjestikuse ja paralleelse kloonimise ajaline võrdlus 32

1 Sissejuhatus

Plagiaat on levinud probleem, kus inimene esitab kellegi teise tööd enda nimel. Plagiaadi tuvastamine on keeruline ja ajakulukas protsess. Antud probleem esineb lisaks kirjatöödele ka programmeerimistöödes.

Tallinna Tehnikaülikoolis toimub iga semester palju programmeerimiskursuseid, mis vajavad plagiaadi kontrollimist. Olemasolev süsteem on vähese funktsionaalsusega ja selle kasutamine on ajakulukas ning ebamugav. Puudub igasugune võimalus kasutajaid hallata ja õigusi jagada.

Töö käigus luuakse veebirakendus, mis võimaldab õppejõududel mugavalt ja automaatselt kontrollida plagiaati kasutades selleks koodi sarnasust tuvastavat süsteemi Moss. Selle kasutamist võimaldatakse kõikides TalTechi programmeerimiskursustes.

Rakendus laeb automaatselt tudengite tööd määratud GitLabi grupist, töötleb need sobivale kujule ja kuvab ülevaatlikul kujul. Hoitakse meeles tudengeid, kes on jäänud korduvalt plagiaadiga vahele. Kasutajatele saab anda piiratud õigusi ja ligipääs on ainult kursustele, kus kasutaja on märgitud õppejõuks.

Töö käigus realiseeritakse dokumenteeritud rakendusliides, kuhu Moodle hindamissüsteem saab saata kaitstuks märgitud tudengi GitLabi salve ajahetke identifikaatori, et tagada õppejõule esitatud koodi kontrollimist.

Rakenduse *back-end* osas kasutatakse Django raamistiku ja *front-end* osas Reacti raamistiku. Stabiilsust aitavad tagada testid ja pidevkooste (*continuous integration*).

Loodud rakendus on arusaadav ja täiendatav ka igale teisele arendajale sisaldades projekti salves vajaliku dokumentatsiooni nii koodi toimimise kohta kui ka serverisse uue projekti püsti panemiseks ja olemasoleva täiendamiseks ning juurutamiseks.

2 Analüüs

Antud peatükis analüüsitakse loodava süsteemi vajadusi.

2.1 Süsteemi vajadused

Loodav plagiaadikontrolli võimaldav veebirakendus peab olema mugav ja lihtsasti kasutatav kõikidele õppejõududele. Kontrollid peavad olema struktureeritud ehk seotud kindla kursuse, ülesande, projekti ja tudengiga.

Rakenduses saab kursuse loomisel märkida GitLabi grupi, kust kloonitakse tudengite projektid. Kontrolli käivitamisel saadetakse vastava ülesande failid tudengite salvedest koodi sarnasust tuvastavasse süsteemi Moss [1]. Saadud tulemused töödeldakse ja salvestatakse andmebaasi. Kasutajaliideses saab muuta Mossi seadeid.

Õppejõud saab kontrolli vasteid märkida vastavalt plagiaadiks või aktsepteeritavaks. Esialgu on vasted olekus uus. Vasted, mis jäävad alla määratud piiri, märgitakse automaatselt aktsepteeritavaks ehk mitte plagiaadiks.

Tudengi profiili vaates näeb korduvaid plagiaadiga vahele jäämisi ülesannete ja kursuste lõikes. Superkasutajal on võimalus Django administreerimisliidese kaudu näha rohkem informatsiooni.

Kontrolli kulgu saab jälgida veendumaks, et kõik toimub ootuspärasel viisil. Juhul kui esineb tõrkeid, siis need on käsitletud ja kuvatakse sobivat veateadet. Ebaõnnestunud kontrolli saab uuesti käivitada.

Kursustel kaasatake tihti abilisi, kellele saab õppejõud võimaldada ligipääsu piiratud kujul. Selleks luuakse kolm õiguste taset, mis on järgnevad: abiline, õppejõud ja superkasutaja. Abiline ja õppejõud näeb ainult kursusi, kus ta on märgitud õppejõuks. Superkasutajal on seevastu kõik õigused ja ligipääs administreerimisliidesele. Õppejõud saab luua uusi kursusi ja seal tegutseda. Abilisel on kursuse raames ainult lugemise õigus

ja puudub õigus vaadata tudengi profiili vaadet. Kirjeldatud õigused on valideeritud nii kasutajaliideses kui ka serveris.

2.2 Andmebaas

Andmete hoidmiseks kasutatakse rakenduses PostgreSQL andmebaasi. Andmebaas defineeritakse Django mudelite kaudu. Reeglina vastab igale mudelile üks andmebaasi tabel.

2.2.1 Mudelid

Rakenduses luuakse järgnevad mudelid: `User`, `Course`, `Assignment`, `Student`, `StudentDefenseCommit`, `Submission`, `GitlabProject`, `Match` ja `MicrosoftAccount`. Lisaks sellele on andmebaasis Django poolt autentimisega seotud tabelid, mis hoiavad andmeid kasutajate sessioonide kohta ning tabel rakendusliidese võtmete hoidmiseks. Lisas 2 on näha kõikidest mudelitest loodud klassidigrammi.

Kasutaja (`User`) mudel on vajalik rakenduse kasutajate hoidmiseks. Sinna salvestatakse Microsofti autentimisest saadud inimese nimi, meiliaadress ja seos saadud Microsofti kasutaja identifikaatoriga. Olemas on väljad, mis määravad ära kas kasutajal on abilise, õppejõu või superkasutaja (*superuser*) õigused.

Kursuse (`Course`) mudel hoiab nime, *slug*'i ehk inimloetavat võtmesõna aadressiribal, kasutatavat programmeerimise keelt, õppejõude, Moodle'i hindamissüsteemi identifikaatorit, GitLabi grupi identifikaatorit, projekti nime filtreerimiseks regulaaravaldist, projektide asukohta, logide infot, järjekorra numbrit ja loomise aega. Lisaks on väljad faililaiendite, maksimaalsete läbipääsude ja vastete kuvamiste jaoks, mis on vaikeväärtusteks ülesannete loomisel.

Ülesande (`Assignment`) mudel hoiab sarnaselt kursusele nime, *slug*'i, Moodle'i hindamissüsteem identifikaatorit, logide infot, järjekorra numbrit ja loomise aega. Lisaks on olemas staatus ja välisvõti kursusele. Staatus on üks järgnevatest väärtustest: uus, kontrollimas, valmis või ebaõnnestunud.

Tudengi (`Student`) mudel hoiab tudengi nime ja Uni-ID-d, mis peab olema andmebaasis unikaalne. See võimaldab luua tudengi profiili ehk seostada omavahel erinevates kursustes toimuvaid plagiaadi vasteid.

Tudengi kaitstud kehtestuse (`StudentDefenseCommit`) mudel esindab Moodle'i hindamissüsteemist saadetud tudengi kaitstud kehtestusi. Seda kasutatakse ülesannete kontrollimisel konkreetse kehtestuse seisuga tudengi projekti kloonimiseks.

Esituse (`Submission`) mudel on seotud välisvõtme kaudu ülesandega ja GitLabi projektiga. Lisaks on väli kehtestuse räsi ja faili sisu (*file content*) jaoks.

GitLabi projekti (`GitlabProject`) mudel hoiab GitLabi projekte. See on välisvõtme kaudu seotud kursuse ja tudengiga. Väljad on ka lisainfo ja statistika jaoks.

Vaste (`Match`) mudel on seotud kahe esitusega. Sellel on staatus, mis saab omada järgnevaid väärtusi: uus, plagiaat, aktsepteeritav. Selle külge läheb kõikvõimalik info, mida saab kätte töödeldes Mossi tulemusi.

2.2.2 Päringud

Andmebaasi päringute tegemiseks kasutatakse Django ORM-i. Joonis 1 demonstreerib ORM-is kirjeldatud päringut ja sellele vastavat SQL lauset. Antud päring valib `Match` tabelist need kattuvused ja välisvõtme kaudu seotud esituse (*submission*) kehtestuse räsid, mille vaste (*match*) staatus on uus (*new*) ning kattuvusi (*lines matched*) üle 50. Tulemused on järjestatud kahanevalt kattuvuse järgi.

Django ORM-i kasutamine kiirendab ja lihtsustab oluliselt andmebaasi andmete käsitlemist. Väldib vigu, mis tekivad pikkade SQL lausete kirjutamisel. Lisaks PostgreSQL andmebaasile on olemas hea tugi MySQL'ile ja SQLite'ile. See võimaldab põhiandmebaasina kasutada PostgreSQL'i, aga testandmebaasiks kiiret mälu režiimis töötavat SQLite'i.

```
from plagiarism.models import Match

matches = Match.objects.filter(
       status='new',
       lines_matched__gt=50,
).order_by('-lines_matched').values('submission__commit_sha')

# Generated SQL query
print(str(matches.query))

SELECT "plagiarism_submission"."commit_sha" FROM "plagiarism_match" INNER
JOIN "plagiarism_submission" ON ("plagiarism_match"."submission_id" =
"plagiarism_submission"."id") WHERE ("plagiarism_match"."lines_matched" > 50
AND "plagiarism_match"."status" = new) ORDER BY
"plagiarism_match"."lines_matched" DESC
```

Joonis 1. Django ORM *query* näide.

3 Rakenduse realisatsioon

Antud töös realiseeritakse veebirakendus kasutades parimaid autorile teadaolevaid meetodeid. Selles peatükis tutvustatakse arenduseks kasutatud meetodeid, raamistike ja tehnoloogiaid.

3.1 IT-taristu ja selle haldus

Loodav veebirakendus vajab serverit, kus projekt töötama hakkab. Samuti peab server olema seadistatud järgides parimaid eelkõige turvalisusega seotud tavasid ja omama projekti püsti panemiseks vajalike pakke.

Arenduse käigus kasutatakse kahte serverit. Üks serveritest, mis paikneb DigitalOceani pilveserveris, on *staging* serveri rollis ja teine, mis paikneb TalTechi serveris, on toodanguserveri rollis. Mõlemasse serverisse paigaldatakse operatsioonisüsteem Ubuntu 18.04. Domeenid on keskkondadel vastavalt https://plagiarism.staging.rrebase.com/ ja https://plagiaat.cs.ttu.ee/.

3.1.1 Ansible

Serverite halduseks kasutatakse Ansible'it, mis võimaldab suure hulga ülesandeid (*task*) kirja panna korduvalt käivitatavateks *playbook*'ideks. See tagab serveris Django projektiks vajaliku konfiguratsiooni ja annab hea ülevaate tehtud muudatustest. Uue serveri ülesseadmiseks kulub ainult minuteid ja organiseeritus on tagatud.

Antud rakenduse puhul jooksevad kõik vajalikud teenused Dockeri konteinerites. Turvalise HTTPS ühenduse jaoks on vaja seadistada SSL sertifikaadid. *Staging* serveris on sertifikaadid Let's Encrypti poolt, kuid toodanguserver kasutab TalTechi Terena poolt väljastatud sertifikaati.

Ülesandeid luuakse lokaliseerimise, tulemüüri, portide ja kasutajate seadistamiseks ning Dockeri paigaldamiseks.

3.1.2 Fabric

Juurutamiseks kasutatakse Pythoni pakki Fabric, mis on disainitud kesta käskude jooksutamiseks üle SSH. See ühendab vastavalt *staging* või toodanguserverisse ja sisestab vajalikud käsud uue versiooni püsti panemiseks.

3.1.3 Nginx

Nginxi kasutatakse veebirakenduses veebiserverina. See tegeleb koormuse jaotusega, liikluse jälgimisega, SSL sertifikaatide haldamisega ja üldise turvalisusega.

Konfiguratsioonis on kirjeldatud kiiruse piirangud sisse logimise lehele, staatiliste failide serveerimine ja vigade käsitlemine.

3.1.4 Docker

Kõik rakenduse teenused jooksevad Dockeris. See tagab ühesuguse arenduskeskkonna igal platvormil ja vähendab peavalu erinevate teenuse konfigureerimisel. Iga teenus eraldi konteineris suurendab turvalisust ja potentsiaalset skaleeritavust. Dockerit toetavad kõik tuntumad pilveteenused ja seetõttu on rakenduse liigutamine ühest kohast teise võrdlemisi lihtne.

3.1.5 GitLab CI

Rakenduse realiseerimise jooksul kasutatakse GitLabi pidevkoostet. See käivitatakse iga mestimispäringu tegemisel ja `master` harusse üleslaadimisel. Antud juhul on pidevkooste seadistatud kontrollima koodi kvaliteeti ja jooksutama teste. Igal käivitamisel paigaldatakse loodavas Dockeri konteineris kõik vajalikud pakid nullist tagamaks, et kõik süsteemi osad töötaks ootuspäraselt. Lisaks arvutatakse ja kuvatakse koodi testidega katvuse protsent (*code coverage*).

Pidevkoostest teostab GitLab Runner, mis on üles seadistatud eraldi serverisse. Lisaks on planeeritud igapäevased konveierid `master` ja `live` harul, mis püüavad kinni võimalike pakkide paigaldamisega seotud probleeme.

3.2 *Back-end* raamistik ja pakid

Rakenduse realiseerimiseks on kasutusel mitmed populaarsed raamistikud. Selles peatükis tehakse nendest ülevaade.

3.2.1 Django

Django on Pythoni baasil ehitatud avatud lähtekoodiga *high-level* veebiraamistik, mis järgib MVT mustrit. See soodustab kiiret ja efektiivset arendust. Peamisteks tugevusteks on hea turvalisus ja *out-of-the-box* administreerimisliides.

3.2.2 Django REST framework

Antud rakendus on põhiliselt ehitatud SPA-na ehk suur osa andmetest jõuab kasutajani läbi rakendusliidese. Django REST framework on rohkete võimalustega töörist rakendusliideste ehitamiseks. See võimaldab rakendusliidest läbi brauseri otse lehitseda lihtsustades arendaja tööd. Olemas on tugi andmete serialiseerimiseks ORM-i andmeallikatest, millele Django on üles ehitatud. Seetõttu on lihtsalt võimalik andmebaasist viia vajalik info JSON kujule, et seda kasutada *front-end'*is andmete töötlemiseks ja kuvamiseks.

3.2.3 Pytest

Pytest on Pythoni testimisraamistik. See on oluliselt parem sisseehitatud *unittest* raamistikust.

Pytest võimaldab parametriseerimist ehk sama testi jooksutamist mitme konfiguratsiooniga ilma, et peaks selleks kirjutama tsüklit. Iga testi jooksutatakse eraldi ja arendaja näeb täpselt ära, mis sisenditega test ebaõnnestus ja millistega mitte. Seejuures annavad *assert*'id ebaõnnestumisel palju infot.

`Fixture` on funktsionaalsus, mis võimaldab korduvaid testobjekte taaskasutada mitmes testis. Pytest suudab `fixture` automaatselt üles leida samast kaustast ja lubab need sisestada testfunktsiooni parameetriks seal, kus vaja.

3.2.4 WebSocket

Websocket on kahesuunaline reaalajas kliendi-server suhtluseks loodud tehnoloogia. See võimaldab kuvada kliendile reaalajas uusi andmeid ilma, et klient peaks tegema uut HTTP päringut. Antud rakenduses kasutatakse seda reaalajas logide ja staatuse uuenduste saatmiseks.

3.2.5 Celery

Celery on asünkroonne järjekorra põhine süsteem tausta ülesannete jooksutamiseks. Seda kasutatakse antud rakenduses plagiaadikontrolli teostamiseks ja GitLabis olevate tudengite projektide kohta info pärimiseks. Mõlemad tegevused on reeglina ajakulukad võttes mõnel juhul mitmeid minuteid. Celery võimaldab jooksutada taustal mitmeid kontrolle samaaegselt.

3.2.6 Redis

Redis on avatud lähtekoodiga mälupõhine võti-väärtus andmebaas. Seda kasutatakse sõnumivahendajana. Celery kasutab seda järjekorra hoidmiseks. Rakenduses kasutatakse seda veel kulukate operatsioonide lühiajaliseks puhverdamiseks.

3.2.7 PostgreSQL

Andmebaasiks on antud rakenduses PostgreSQL. See on hästi toetatud Django poolt. Tabelid kirjeldatakse mudelina ja andmeid käsitletakse läbi Django ORM-i. Andmebaasis hoitakse tabeleid kursuste, ülesannete, kasutajate, tudengite, vastete, sessioonide ja muu sellise kohta.

3.3 *Front-end* raamistik ja pakid

Front-end osa ehitamiseks kasutatakse tuntud raamistiku React. Antud peatükis kirjeldatakse selle kasutust antud rakenduses.

3.3.1 React

React on JavaScripti raamistik kasutajaliideste loomiseks. See võimaldab tükeldada koodi taaskasutatavatest komponentideks. Opereeritakse virtuaalse DOM-i peal, mis on efektiivne ja teeb uuendusi ainult siis, kui on toimunud reaalseid muudatusi. Selline lähenemine lubab deklaratiivset Reacti rakendusliidest. Arendaja saab defineerida kasutajaliidese ja React vastutab selle eest, et vastav olek läheks kokku reaalse DOM-iga [2].

3.3.2 Formik

Vormide ehitamiseks Reactis kasutatakse Formikut. See lihtsustab vormi oleku, valideerimise ja vigade käsitlemist. Antud rakenduses on see kasutuses kursuse loomise, muutmise, ülesande loomise, muutmise ja seadete muutmise vormides.

3.3.3 Redux

Suuremad rakendused muutuvad erinevate kasutajaliideses olevate olekute tõttu keeruliseks väga kiiresti. Teadete edastamine omavahel mitte seotud komponentide vahel on tülikas ja tulemuseks on raskeski jälgitav kood. Seda probleemi lahendab Redux hoides kogu rakenduse olekut ja loogikat tsentraliseeritult ühes kohas. Testimine ja silumine on tänu Reduxile lihtsustatud.

3.3.4 Babel

Babel on JavaScripti kompileerija. See võimaldab kasutada uuema generatsiooni JavaScripti koodi teisendades koodi vanemate brauseritega ühilduvaks koodiks. Rakenduses kasutatakse muuhulgas funktsioonide kirjeldamiseks ES2015 *arrow function'*it, mille Babel muudab vanema generatsiooni funktsiooni kujule. Oluline koht on Babelil ka JSX süntaksi konverteerimisel, mida kasutatakse Reactis kasutajaliidese kirjeldamiseks.

3.3.5 ESLint

JavaScripti ja JSX koodi analüüsitakse ESLint abil, mis tagab parimate tavade kasutuse ja aitab vältida vigu isegi enne koodi käivitamist.

3.3.6 SCSS

SCSS on täiendus tavalisele CSS-ile. See võimaldab kirjutada stiile paremini loetavalt ja taaskasutatavalt. SCSS tuleb kompileerida CSS-iks.

3.3.7 Bootstrap

Bootstrap on populaarne raamistik mobiilisõbralike veebilehtede koostamiseks. See sisaldab kindla stiiliga kõike levinud komponente alustades nuppudest lõpetades rippmenüüdega. Rakenduses on vaikestiilid ülekirjutatud muutes vastavaid Bootstrapi SCSS muutujaid.

3.3.8 Webpack

Erinevate sõltuvustega SCSS, piltide ja JavaScripti moodulite kokku pakendamise eest vastutab Webpack. Rakenduses on kasutusel `babel-loader`, `css-loader`, `sass-loader`, `url-loader`. Enne lõpliku pakendamist toimub optimeerimine minimeerides JavaScripti ja CSS faile, eemaldades kommentaare ja taaskasutades sarnaseid alamosi. Kohaliku arenduse jaoks minimeerimist ei toimu, et võimaldada selgemat silumist.

4 Rakenduse realisatsioon

Antud peatükis kirjeldatakse rakenduse realisatsiooni käigus loodavat struktuuri ja olulisemate tegevuste protsessi.

4.1 Funktsionaalsus

Kirjeldatakse loodavat funktsionaalsust plagiaadi kontrollimiseks.

4.1.1 Uni-ID-ga autentimine

Süsteemi sisenemiseks on vajalik autentimine. Antud rakenduse puhul on sobilik selleks TalTechi meiliaadressi kasutamine. TalTechi meiliga autentimist võimaldab Microsofti poolt loodud Azure auth.

Superkasutajale antakse ka alternatiivne võimalus luua kasutaja, kes saab sisse logida meili ja parooliga.

4.1.2 Kursuste loomine, muutmine ja kustutamine

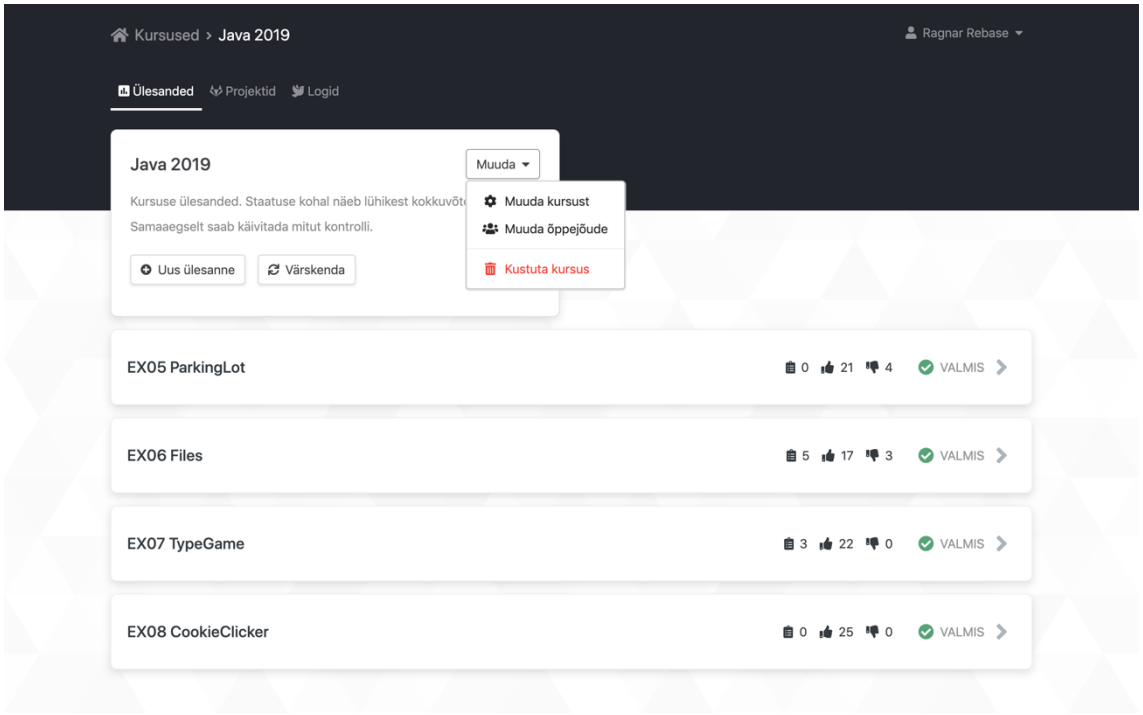
Plagiaadi kontrollide paremaks organiseerimiseks on hea neid jagada vastava kursuse alla. Selleks on vaja luua kursuseid. Kursuste loomiseks on olemas vorm, mida saavad kasutada õppejõu või superkasutaja õigust omavad kasutajad. Kursuse loomisel peab kasutajal olema määratud GitLabi juurdepääsuluba (*access token*). Juhul, kui seda pole tehtud, kuvatakse vormis vastavat hoiatust. Vormis on igal väljal valideerimine, mis tagab korrektse info sisestamise ja saatmise lisaks *back-end* kontrollidele.

Kursuse muutmise vormile saab liikuda kursuse detailvaatest. Sealt leiab rippmenüü erinevate kursusega seotud valikutega. Vajutades muutmise nupule avaneb modaalaken vormiga, kus on võimalik teha muudatusi ja neid salvestada.

Kursusega seotud õppejõudude lisamiseks ja eemaldamiseks leiab rippmenüüst vastava nupu. Sellele vajutades avaneb modaalakend olemasolevate õppejõudude tabeliga, kus on näha õigusi ja on võimalus uute lisamiseks ning olemasolevate eemaldamiseks.

Kursuse kustutamiseks on rippmenüüs vastav nupp ja selle vajutamisel küsitakse kinnitust enne reaalset kustutamist.

Joonisel 2 on näha avatud rippmenüüga kursuse ülesannete vaadet koos eelnevalt kirjeldatud valikutega. Kasutajatel, kellel pole õigust kursust hallata, näevad nuppe keelatud olekus. Vastav õiguste kontroll on olemas ka serveri poolel.



Joonis 2. Kursuse ülesannete vaade.

4.1.3 Ülesannete loomine, muutmine ja kustutamine

Kursustele lisaks saab luua igale kursusele ülesandeid. Selleks on analoogselt kursustele olemas vastavad loomise ja muutmise vormid. Ülesannete kontrollimiseks on vaja igal kursusel esmalt pärida projektid, mis lähevad kontrollimisele.

4.1.4 Projektide vaade

Nägemaks, et kõik tudengite projektid on korrektselt kätte saadud ja andmebaasi salvestatud, on kasutajaliideses realiseeritud tabel. Sealt saab otsida, seda saab sorteerida ja see on jagatud mitmeks leheküljeks, sest projekte võib olla palju.

4.1.5 Logide vaade

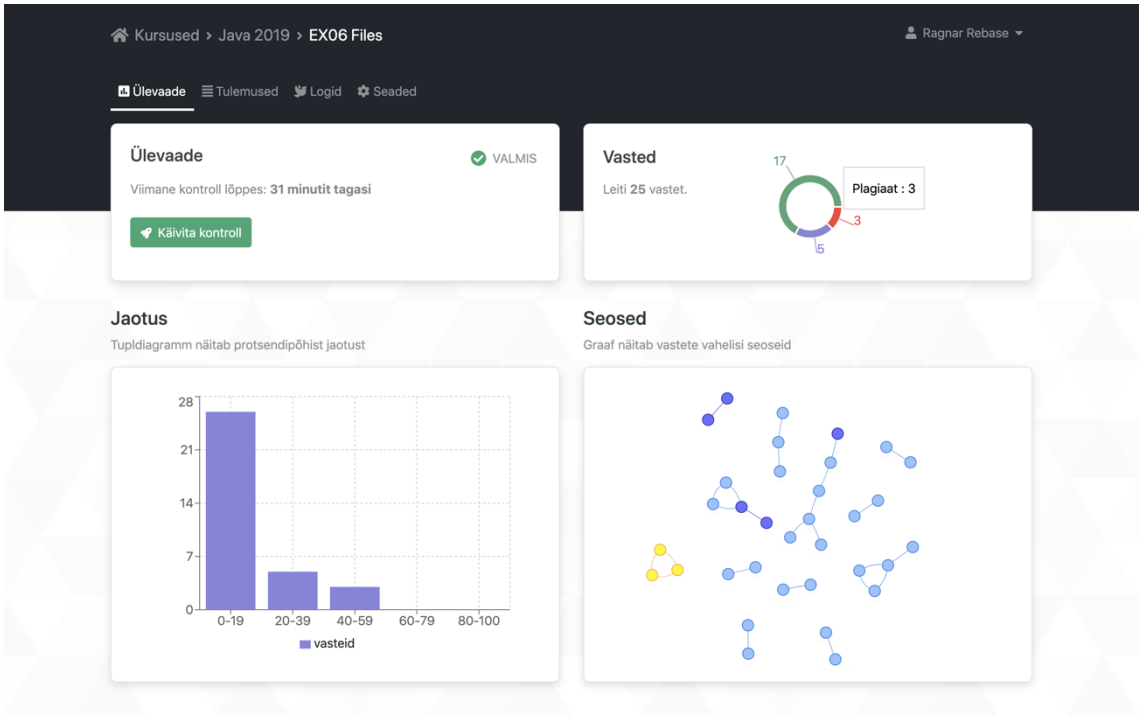
Rakenduses on olemas logide vaade nii kursuse projektide laadimise juures kui ka iga ülesande juures, mis näitab reaalajas WebSocketite abil serveri logisi käimasoleva plagiaadikontrolli kohta.

Logide vaade võimaldab jälgida plagiaadikontrolli protsessi kulgu. Kui midagi läheb valesti, siis logidest on näha, mis täpselt juhtus. Levinud veaks võib olla vigase ülesande teekonna sisestamine.

4.1.6 Ülesande ülevaade

Joonisel 3 on näha ülesande ülevaatliku vaadet, kuhu suunatakse ülesande detailvaate avamisel. Seal kuvatakse tulpdiagrammina vastete protsendipõhist jaotust ja seoste võrgustikku. Sektordiagrammil näeb vastete jaotust staatuse järgi.

Juhul, kui kasutajal on GitLabi juurdepääsuluba (*access token*) määramata ja tal on õigus kontrolli alustada, siis kuvatakse vastav hoiatus. Staatuse oleku muutumise kohta tuleb informatsioon läbi WebSocketite ehk uuendust näeb reaalajas ja lehe värskendamine pole vajalik.



Joonis 3. Ülesande ülevaatlik vaade

4.1.7 Plagiaadikontrolli vastete tabel

Kontrolli tulemusel saadud vasted kuvatakse tabelina, milles saab otsida ja sorteerida. Lisaks on igal vastel võimalik märkida see vaste kas aktsepteerituks või plagiaadiks.

4.1.8 Tudengi profiili vaade

Õppejõul ja superkasutajal on võimalik vaadata tudengi profiili vaadet. Seal on näha mitmel korral on tudeng plagiaadiga vahele jäänud. Superkasutajal on võimalus läbi administreerimisliidese kindlaks teha kõik need plagiaadid, millega tudeng on vahele jäänud.

4.2 Struktuur

Projekti struktuur on jagatud konfiguratsioonifailideks, juurutamise failideks, *back-end* osa ehk Django failideks ja *front-end* osa ehk Reacti failideks. Projekti esialgne skelett tuli Thorgate Django projekti mallist [3].

4.2.1 Dot-failid ja konfiguratsioonifailid

Projektis on mitu dot-faili, mis on peamiselt mõeldud erinevate tööriistade konfigureerimiseks. `editorconfig` tagab järjepideva koodistiili defineerides ära reeglid faili tüüpide kaupa nagu taanete stiili, suurused ja kodeeringu, mida suudavad ära kasutada enamik koodiredaktoreid. `gitignore`, `dockerignore` ja `eslintignore` sisaldavad nimekirja failidest või kaustadest, mida vastavalt Git, Docker ja ESLint peavad ignoreerima. `gitlab-ci.yml` defineerib ära konfiguratsiooni ja sammud GitLab pidevkooste jaoks. `babelrc` määrab ära seadistuse ja pistikprogrammid JavaScripti kompileerimiseks, et oleks võimalik kirjutada uuema generatsiooni JavaScripti. `prospector.yaml` on Pythoni lintimise paki konfiguratsioonifail, kus täpsustatakse PEP8 ja Pylint kontrollitavaid reegleid.

Projekti juurkaustast leiab mitu Dockeriga seotud faili. Kohalikuks arenduseks on `docker-compose.yml`, mis orkestreerib Dockeri konteinereid. See sisaldab `django`, *`node`*, *`postgres`*, *`celery`*, *`celery_beat`* ja *`redis`* teenuseid.

Sealjuures põhineb Django `Dockerfile-django` dockerfailil, mille baasiks on ametlik Pythoni tõmmis. Lisaks paigaldatakse *gettext* ja Git, mis on vajalikud vastavalt

tõlgete genereerimiseks ja tudengite salvede kloonimiseks. Seejärel laetakse alla kõik `Pipfile` failis defineeritud pakid, defineeritakse *volume*'s olev töökaust ja käivitatakse arendusserver pordil 8000.

Front-end osa eest vastutav *node* teenus põhineb `Dockerfile-node` dockerfailil. See baseerub node ametlikul viimasel LTS tõmmisel. Esimese sammuna määratakse *volume*'s olev töökaust. Seejärel paigaldatakse `package.json` failis olevad pakid ja käivitatakse arendusserver.

Andmebaasi teenus `postgres` baseerub tõmmisel `postgres:11`. Django konteiner saab andmebaasiga otse suhelda, olles samas sisemises võrgus ja seetõttu andmebaasi porti avama ei pea. Mitmed teenused kasutavad *bash'*i skripti `wait-for-it.sh`, mis ootab kuni ühendus andmebaasiga on saadaval.

Celeryle lisaks leiab teenuse `celery_beat`, mis ainuüksi vastutab perioodiliste tegevuste planeerimise eest. Nimetatud teenused kasutavad andmete hoidmiseks kergekaalulist võti-väärtus andmebaasi Redis.

4.2.2 Django struktuur

Django soovitab projektid jagada taaskasutavateks rakendusteks [4]. Antud projekt on jagatud järgnevateks rakendusteks: `accounts`, `courses`, `moss`, `plagiarism` ja `microsoft_auth`. Iga rakendusega on seotud andmebaasi mudelid, migratsioonid, testid ja vaated. Kõik projekti tõlkefailid loodakse kausta `locale` vastava keele alamkausta. Kaust `info` sisaldab endas WebSocketite tarbijaid (*consumers*) ja marsruute (*routing*).

Django sätted leiab `settings` kaustast. Seal on Pythoni sätete fail iga keskkonna jaoks. Kõik keskkonnad baseeruvad failil `base.py`, mis sisaldab ühiseid sätteid. Kohalikuks arenduseks minevad lisasätted jõuavad faili `local.py`, mis luuakse projekti püsti pannes `local.py.example` näitel. Testserver kasutab säteteks faili `staging.py` ja toodanguserver `production.py`. Kusjuures toodanguserveri sätted põhinevad testserveri sätetel, mis omakorda põhinevad baassätetel.

Arenduse käigus muudetavad staatilised failid leiab kaustast *`static`*. Seal on projektis kasutatavad SVG-d ja *favicon*. Samuti paikneb seal `styles-src`, mis sisaldab suurt

osa stiilidest. Stiilid on organiseeritud kasutades Atomic Design mustrit [5]. See tähendab, et Sassi failid on grupeeritud aatomiteks, molekulideks, organismideks ja vaadeteks. Imporditakse ka Bootstrapi stiilid, mille muutujaid kasutatakse stiilifailides ning kirjutatakse vajadusel üle tagamaks ühtset joont.

Kaustast `template` leiab rakenduste kaupa grupeeritud HTML failid, mis on kirjutatud kasutades Django malli keelt. Need sisaldavad muutujaid ja vähesel määral loogikat, et lihtsal ja loetaval viisil genereerita sobilik HTML fail. Antud projektis on neid kasutatud võrdlemisi vähe, sest suur osa rakendusest on ehitatud Reactis SPA-na ja mitte Django *template engine* põhjal.

Rakendusliidesed leiab iga Django rakenduse kaustast `api`, kus on defineeritud vaatekomplektid, serialiseerijad ja marsruudid. Lisa 12 on näide ühest vaatekomplektist.

4.2.3 Reacti struktuur

Kasutajaliidese kood paikneb Django projekti `app` kaustas. Pakendamise eest vastutava Webpacki konfiguratsioonifailid on kaustas `webpack` ja JavaScripti failid kaustas `src`. Sisenemispunktiks on fail `main.js`, mis initsialiseerib React *router*'i, Redux *store*'i, määrab ajatsooni ja kasutatavad Font Awesome'i ikoonid.

Komponendid on jagatud vaadete kaupa kaustadesse. Reacti komponendid on kirjutatud nii, et need oleks loetavad ja korduvkasutatavad. Igasse komponendi kausta on pandud fail `index.js`, mis ekspordib komponendi `export {default} from './CourseDetail'`. See võimaldab komponente mugavamalt lühema teega importida, täpsustades ainult kausta nime, mitte kogu teekonda komponendini.

Keerulisema infovahetusega komponendid on ühendatud Reduxiga. Seda on näha siis, kui komponendi eksportimisel on näga järgnevat koodjuppi `connect(mapStateToProps, mapDispatchToProps)(CourseDetail)`. Nendega kaasnevad funktsioonid `mapStateToProps` ja `mapDispatchToProps`, kus on vastavalt defineeritud Reduxi olekute (*state*) ja tegevuste (*action*) ülekandmised komponendi külge.

Reduxi *reducer*'id kombineeritakse kokku `ducks` kaustas olevatest väiksematest *reducer*'itest. Joonis 4 demonstreerib lihtsustatud näitega, kuidas igas `duck` failis on

kasutatud omakorda kombineerimist *combineReducer*s'i abil. Kasutatud on mustrit, kus iga oleku jaoks on üks *reducer*, see annab hea loetavuse kogudes ühte kokku kõik olekut mõjutavad tegevused. Nii on oleku muudatused hallatavad ka siis, kui projekti koodibaas suureneb.

```
export const STATE_KEY = 'MATCH';

const RECEIVE_MATCHES = `${STATE_KEY}/RECEIVE_MATCHES`;
...other action types

const initialState = {
    match: {},
    ...other state,
};

const matchReducer = (state = initialState.match, action) => {
    switch (action.type) {
        case RECEIVE_MATCHES:
            return action.match.reduce((result, match) => ({
                ...result,
                [match.id]: match,
            }), state);
        default:
            return state;
    }
};

...other reducers

export default combineReducers({
    match: matchReducer,
    ...other reducers,
});
```

Joonis 4 Lihtsustatud näide *reducer*'ite kombineerimisest

Projektis on komponendid `MatchTable` ja `ProjectTable`. Mõlemad võimaldavad tabelis olevaid andmeid vaadata lehekülje kaupa, sorteerida ja otsida. Seetõttu on nende olekut haldavad *reducer*'id samasugused. Kuna programmeerimises on oluline DRY printsiip, siis analoogsete olekute haldamiseks kasutatakse *High-Order Reducer*'eid. Need on funktsioonid, mis võtavad *reducer* funktsiooni argumendiks ja tagastavad tulemusena uue *reducer*'i [6]. Antud juhul lisatakse argumendina vastava *duck*'i eesliide (*prefix*), mis lubab *reducer*'ite loogikat taaskasutada.

Rakenduse suurenedes saab oluliselt vältida vigu teostades tüübikontrolli. Antud projektides on seda tehtud kasutades pakki PropTypes. See lubab määrata andmetele tüübid. Seejärel toimub andmete valideerimine iga kord, kui need komponendist läbi liiguvad.

Rakenduses kasutatakse kõige uuemas Reacti versioonis välja tulnud Hooks rakendusliidest, mis lubab olekut ja muud Reacti funktsionaalsust kasutada klassi kirjutamata [7]. See võimaldab taaskasutada olekutega komponentide loogikat ja vähendada klassidest tulenevat komponentide keerukust.

4.3 GitLabi kasutamine

Antud peatükis antakse ülevaade GitLabi kasutamisest arenduse vältel.

4.3.1 Ülesanded

Nõutud funktsionaalsust kirjeldatakse GitLabi ülesannete abil. Igale ülesandele määratakse võimalusel silt (*label*), mis selle olekut hästi kirjeldab. See võimaldab ülesandeid mugavalt grupeerida, otsida ja analüüsida. Esialgu luuakse järgnevad sildid: *Bug, Design, DevOps, Docs, Doing, Feature, Moss, To Do*.

4.3.2 Harud

Arenduse käigus luuakse reeglina haru igale *Feature* sildiga ülesandele. Harude nimetamine toimub järgneva mustri järgi: <*issue-nr*>-<*short-version-of-issue-title*>. Nii on arendajal kohe võimalik omavahel seostada iga haru konkreetse ülesandega. GitLab tekitab vastavad viited automaatselt ja loob mestimispäringu.

Funktsionaalsuse valmimisel mestitakse muudatused `master` harusse ja *staging* serverisse juurutatakse `master` haru uusim versioon. Lisaks luuakse eraldiseisev `live` haru, mille põhjal juurutatakse toodanguserverit. Sinna jõuab ainult stabiilne ja *staging* serveris varem valideeritud funktsionaalsus.

4.4 Plagiaadi kontrollimine

Antud peatükis kirjeldatakse plagiaadi kontrolli protsessi.

4.4.1 Eeldused

Enne plagiaadikontrolli teostamist on vajalik täita mõned sammud. Kõigepealt peab süsteemi kasutaja olema autenditud. Õiguste kontroll toimub igal sammul nii serveri poolel kui ka kasutajaliideses. Kontrolli saab alustada siis, kui on loodud kursus ja ülesanne. Seda saavad teha kasutajad, kellel on haldamise õigus. Kursust näevad ainult need kasutajad, kes on märgitud kursuse õppejõuks. Erandiks on superkasutajad, kes näevad kõiki kursusi. Ülesande loomisel peab olema määratud korrektne ülesande teekond, sest vastasel juhul plagiaadikontroll ebaõnnestub.

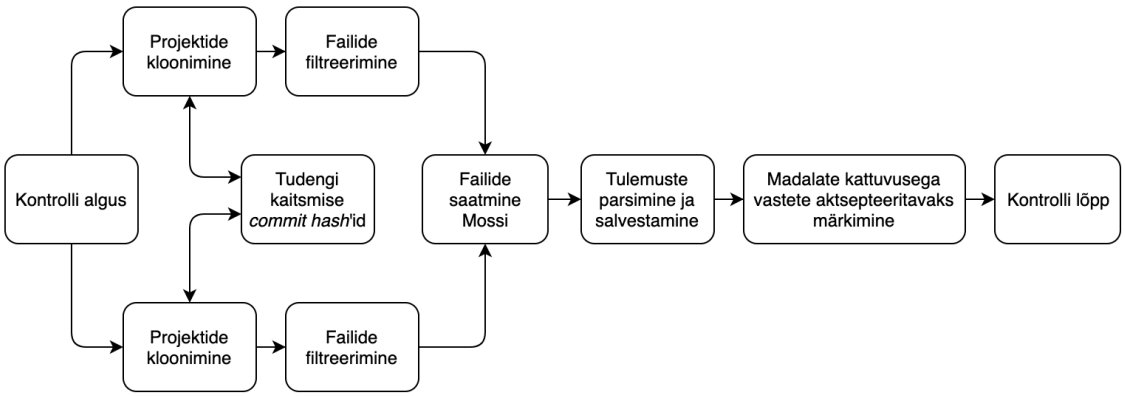
Kursuse loomise järel ja enne kontrolli käivitamist on vaja uuendada andmebaasis olevaid tudengite GitLabi projekte. Seda saab teha kursuse detailvaates, kus on võimalik jälgida ka selle kulgu logide kaudu. Kasutaja, kes soovib kontrolli käivitada, peab olema eelnevalt määranud oma kasutaja seadetes GitLab'i personaalne juurdepääsuluba (*access token*). Seda kasutatakse projektide kloonimiseks. Seega peab kasutajal olema vastava GitLabi grupile ligipääs. Kui luba on määramata, siis kuvatakse vastavat hoiatust.

Realiseeritakse *API endpoint*, kuhu on võimalik saata tudengite kaitstud kehtestuste räsisid, mida kasutatakse võimalusel ülesannete kontrollis. See väldib olukorda, kus tudeng esitab kaitsmise ajaks ühe versiooni koodist ja pärast kehtestab teise versiooni. Selleks on loodud leht dokumentatsiooniga, mis on avalikult kätte saadav aadressil https://plagiaat.cs.ttu.ee/docs/. See on eelkõige suunatud Moodle'i hindamissüsteemi arendajatele, kes saavad integreerida automaatse kaitsmiste saatmise. Dokumentatsioon on avalikult kätte saadav, aga rakendusliidest saab kasutada ainult võtmega autentimisel, mille saab genereerida superkasutaja (*superuser*) Django administreerimisliideses.

4.4.2 Protsess

Joonis 5 näitab lihtsustatud kujul kontrolli etappe. Kontrolli alustab vastavate õigustega kasutaja saates serverisse päringu ülesande *slug*'iga, mis alustab Celery taustprotsessi `send_assignments_to_moss`. Kontrolli käigus on iga olulisema sammu juures välja kutsutud funktsiooni `log_check_progress`, mis võtab argumentideks ülesande ja sõnumi. See funktsioon salvestab praeguse ajahetke, lisab selle logide järjendisse koos sõnumiga ja kutsub välja funktsiooni `send_ws_assignment_info_update`. Viimane uuendab andmebaasis ülesande logide välja ja saadab ülesande Websocketi

kanalisse kõigile kuulajatele sõnumi tüübiga `assignment_info`, mis sisaldab uuendatud logisi ja staatust.



Joonis 5 Plagiaadi kontrolli etapid.

Esimese sisulise sammuna luuakse `MossClient` objekt. Selle objekti külge lisatakse kõik tudengite esitused. Ignoreeritakse neid faile, mille faili suuruseks on null baiti. Tühjaks failiks on tihti `__init__.py`. Esitused saadetakse korraga Mossi läbi sokli. Järgneb GitLabi projektide kloonimine, mis võib olla aeglane protsess sõltuvalt projektide hulgast ja salvede suurustest. Kuna iga projekti kloonimine on üksteisest sõltumatu tegevus, siis on võimalik kasutada `ThreadPoolExecutor`'it paralleelselt salvede kloonimiseks. Tabelis 1 on näha kuidas 400 projekti näitel kiirendab paralleelne kloonimine oluliselt kogu kontrolliks kuluvat aega. Kloonimise autentimiseks kasutatakse kontrolli alustanud kasutaja küljes olevat GitLab juurdepääsuluba (*access token*). Seejärel tehakse andmebaasi päring ülesande identifikaatori, kursuse identifikaatori ja tudengiga, et leida kaitstud kehtestus. Juhul kui selline on olemas, siis tehakse *checkout* sellele kehtestusele ja Mossi saadetakse sellel ajahetkel salves olnud failid. Juhul, kui salv on varasemalt juba kloonitud, siis toimub muudatuste alla laadimine. Tõrke puhul jäetakse tudengi koodi kloonimine vahele ja selle kohta logitakse sõnum. Eduka kloonimise järel käiakse salves oleva ülesande teekonnaks märgid kaust rekursiivselt läbi. Juhul kui failil on lubatud faililaiend, siis see lisatakse `MossClient` objekti `solution_files` järjendisse. Kusjuures seal märgitakse kõik ühe tudengi esituse failid unikaalse eesliitega, milleks on projekti `id`. Selle abil oskab Moss tulemusi kuvada grupeerituna samamoodi tudengi kaupa.

Kloonimise viis	**Ajakulu (s)**
Järjestikune	342.2
Paralleelne 2 *worker*'iga	192.3
Paralleelne 4 *worker*'iga	98.7

Tabel 1. Järjestikuse ja paralleelse kloonimise ajaline võrdlus

Järgneva sammuna avatakse Mossiga sokkel ühendus ja saadetakse konfiguratsiooni väärtused, milleks on programmeerimise keele määramine, vastete arvu piiramine, korduvate ridade piiri määramine ja kausta formaadi kasutamine. Seejärel laetakse sokli kaudu kõik salvedest kogutud tudengite esitused ja jäädakse vastust ootama.

Seejuures on oluline käsitleda tekkivaid vigu. Võimalikud vead, millega tuleb arvestada:

- Moss on *offline*
- Moss ei anna määratud aja jooksul vastust
- Kasutaja sisestatud ülesande teekond on vigane
- Tudeng on pannud Giti väga suure faili
- Tudeng on muutnud Giti ajalugu ja kehtestuse räsi pole korrektne
- Juhtub ootamatu seni käsitlemata viga

Osad eelnimetatud vead on vältimatud ja on oluline kasutaja teavitamine tekkinud veast nii, et oleks võimalik sellele reageerida.

Mossilt saadakse sokli kaudu tulemuseks veebiaadress, kus on võimalik näha tulemusi. Need on HTML tabeli formaadis. Selleks, et andmed andmebaasi sisestada, tuleb lehte töödelda. Paraku on Mossi tulemuste lehel HTML väga vigaselt kirjutatud ja ükski HTML töötleja seda korrektselt lugeda ei suuda. Seetõttu peab kohati kasutama regulaaravaldist, et tulemusi kätte saada. Töötlemise käigus luuakse `MossMatch` objektid, mis hoiavad ühe vastega seotud koodi plokke, protsente ja ridade arvu.

Tulemuste parsimise järel luuakse andmebaasi vasted ja esitused iga tulemuse rea kohta. Juhul, kui selliste esituste vahel on vaste juba olemas, siis uut ei looda. Kahte esitust ühe tudengi poolt peetakse erinevaks siis, kui koodis on tehtud muudatus. Vahele jäetakse vasted, kus mõlemad esitused on sama tudengi poolt. See võib juhtuda siis, kui tudeng on

teinud salvest koopia. Vasted, mille protsent on väiksem määratud konstandist, märgitakse automaatselt aktsepteeritavasse staatusesse ehk nende puhul on protsent nii väike, et suure tõenäosusega pole tegemist plagiaadiga.

Kontrolli lõpus kuvatakse kontrolliks kulunud aeg ja juhul, kui protsessi käigus tekkis viga, siis see on samuti kasutajaliideses nähtav. Igasuguse vea tekkimisel märgitakse ülesande kontrolli olek ebaõnnestunuks ja see edastatakse kõigile kanali kuulajatele WebsSocketi kaudu.

5 Arengusuunad

Antud peatükis kirjeldatakse võimalike arengusuundi.

5.1 Alternatiivse plagiaaditeenuse kasutamine

Moss plagiaadituvastussüsteem pole alati saadaval, sest server on tihti maas ja ei vasta päringutele. Seetõttu on kasulik kaaluda paralleelset mitme teenuse kasutamist, nende kombineerimist, optimeerida olemasolevaid avatud lähtekoodiga süsteeme või tulla välja enda lahendusega. Sõltuvus ühest süsteemist ei ole hea pikas perspektiivis.

5.2 Plagiaadi otsimine veebist ja eelnevate aastate lahendustest

Hetkel tuvastatakse ainult plagiaadiolukordi, kus kaks tudengit on esitanud GitLabi sarnase koodi. Samas esineb olukordi, kus tudeng kopeerib koodi kuskilt veebist. Kontrolli tasuks lisada koodijuppe GitHubi, Stackover Overflow ja Pastebini platvormidelt. Selleks on võimalik teha vastavad otsingud kasutades ülesandes nõutud funktsiooni sõrmejälge. Üldjuhul on funktsiooni nimi ja argumendid piisavalt unikaalne kombinatsioon leidmaks sobivaid vasteid.

Programmeerimiskursustes taaskasutatakse tihti eelnevate aastate ülesandeid ja siis on oht, et tudengit esitavad eelnevate aastate töid enda pähe. Selle lahenduseks saab luua andmebaasi kõikidest eelnevate aastate lahendustest ja lisada need võrdlustesse.

5.3 Sõltumatus GitLabist

Antud rakendus teeb mugavaks plagiaadikontrolli juhul, kui tudengite projektid asuvad GitLabi keskkonnas. Muul juhul on süsteemi kasutamine keeruline. Selleks tuleb lisada tugi failide otse üleslaadimiseks. Kusjuures rakenduse esimene prototüüp loodi sellel viisil. Seega tuleb antud funktsionaalsus Giti ajaloost taastada ja integreerida olemasolevasse süsteemi.

Analoogselt GitLabile on võimalik lisada tugi teistele Giti platvormidele, kuid üldist lahendust, mis kõikides automaatselt toimib, ei ole. Iga platvormi rakendusliides on

erinevate võimalustega ja seetõttu on vajalikud kohandused. Samas ühisosa on suur ja seega töömaht pole suur.

6 Kokkuvõte

Töö tulemusena loodi veebirakendus, mis võimaldab mugavalt kontrollida plagiaati Tallinna Tehnikaülikooli programmeerimise kursustel. Plagiaadikontrollid on seotud kursuste, ülesande, tudengi ja GitLabi projektiga.

Tudengite tööd kogutakse automaatselt GitLabi salvedest ja vastavad ülesanded saadetakse koodi sarnasust tuvastavasse süsteemi.

Projekti seadistamine on dokumenteeritud. Koodibaasis on järgitud parimaid tavasid ja stabiilsust tagavad testid ning pidevkooste. Kontrolli etappe saab reaalajas jälgida logide kaudu. Õppejõul on ülevaade kontrollide olekust.

Loodi dokumenteeritud rakendusliides, kuhu Moodle'i hindamissüsteem saab saata kaitsmiste märkimise ajahetke räsi tudengi GitLabi salvest.

Veebirakendus on saadaval aadressil https://plagiaat.cs.ttu.ee ja koodibaas https://gitlab.cs.ttu.ee/rareba/iapb.

Tulemust on valideeritud selle kasutamisega 2019. aasta kevadel toimuval kursusel "Programmeerimise põhikursus" (ITI0202).

Kasutatud kirjandus

[1] „Moss,“ [Võrgumaterjal]. Available: http://theory.stanford.edu/~aiken/moss/. [Kasutatud 7. mai 2019].

[2] „React Virtual DOM and Internals,“ [Võrgumaterjal]. Available: https://reactjs.org/docs/faq-internals.html. [Kasutatud 2. mai 2019].

[3] „Thorgate Django project template,“ [Võrgumaterjal]. Available: https://github.com/thorgate/django-project-template. [Kasutatud 2. mai 2019].

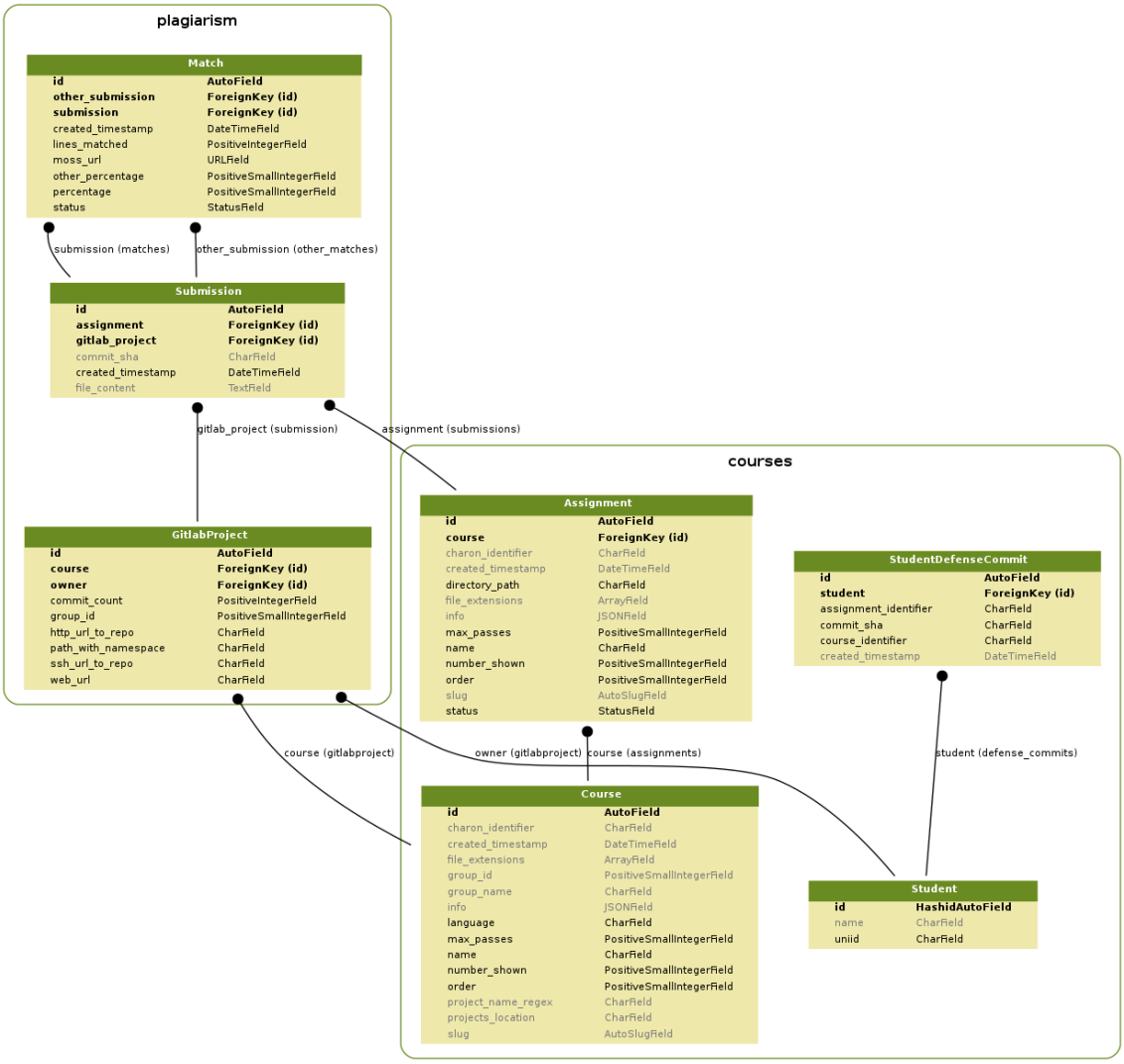
[4] „Django reusable-apps,“ [Võrgumaterjal]. Available: https://docs.djangoproject.com/en/2.2/intro/reusable-apps/. [Kasutatud 5. mai 2019].

[5] „BEM & Atomic Design,“ [Võrgumaterjal]. Available: https://www.lullabot.com/articles/bem-atomic-design-a-css-architecture-worth-loving. [Kasutatud 3. mai 2019].

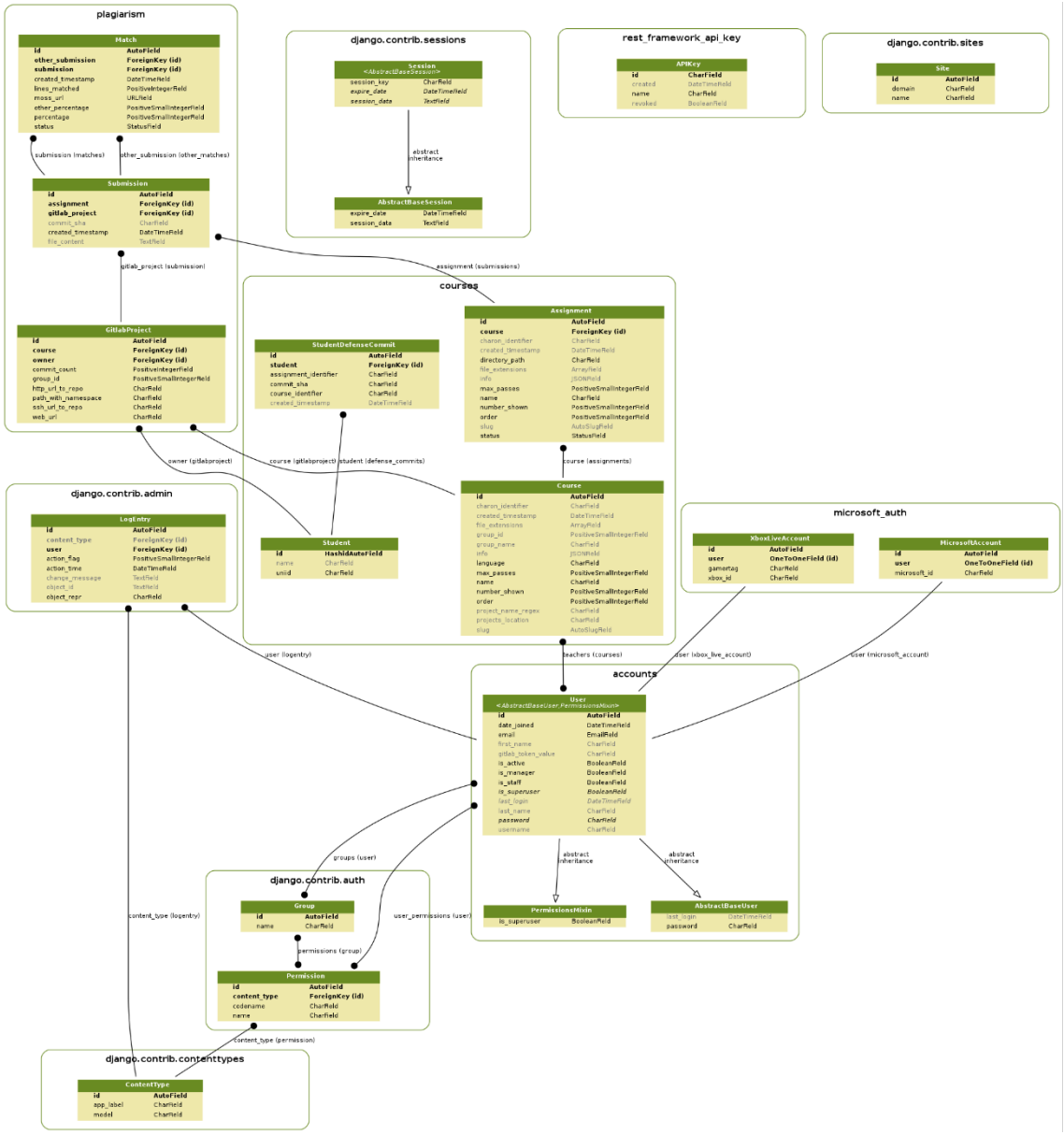
[6] „Redux Reusing Reducer logic,“ [Võrgumaterjal]. Available: https://redux.js.org/recipes/structuring-reducers/reusing-reducer-logic. [Kasutatud 5. mai 2019].

[7] „React hooks,“ [Võrgumaterjal]. Available: https://reactjs.org/docs/hooks-reference.html. [Kasutatud 5. mai 2019].

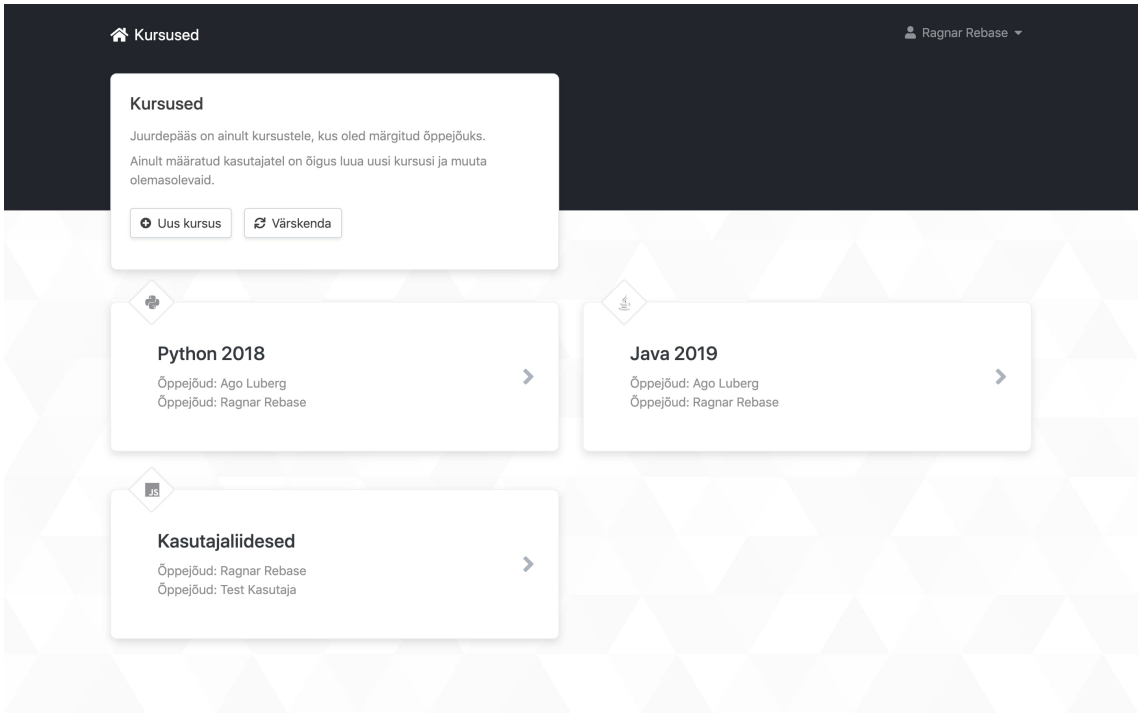
Lisa 1 – Plagiaadi ja kursuse rakenduse mudelid



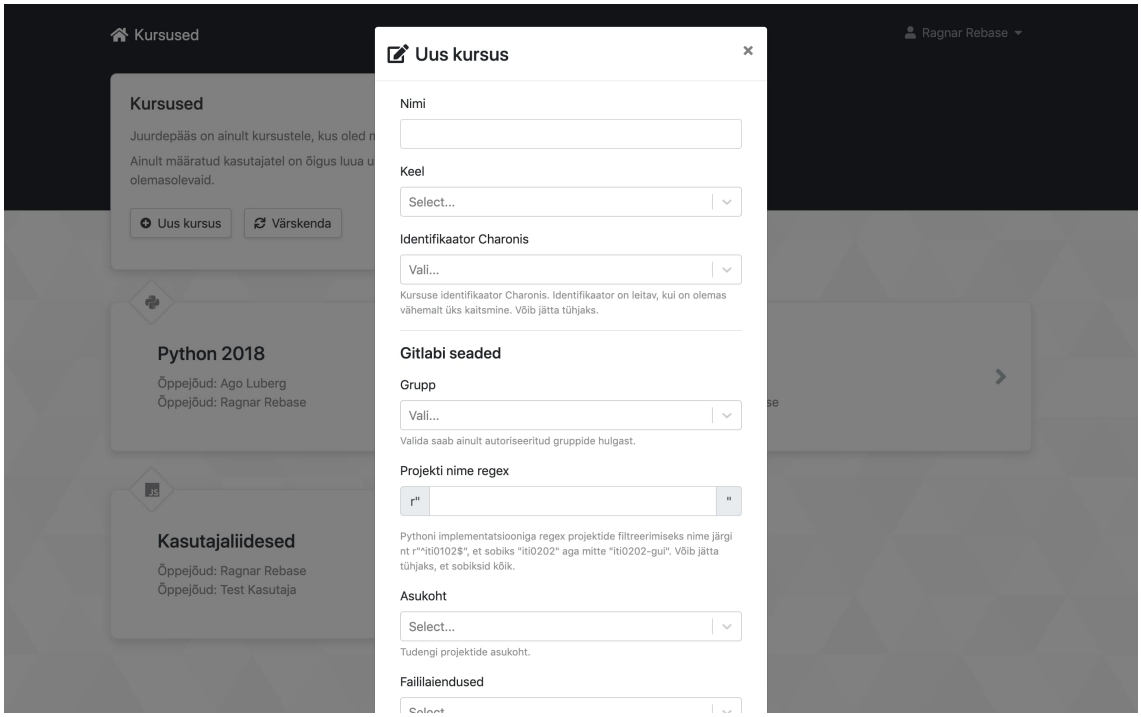
Lisa 2 – Kõikide rakenduste mudelid



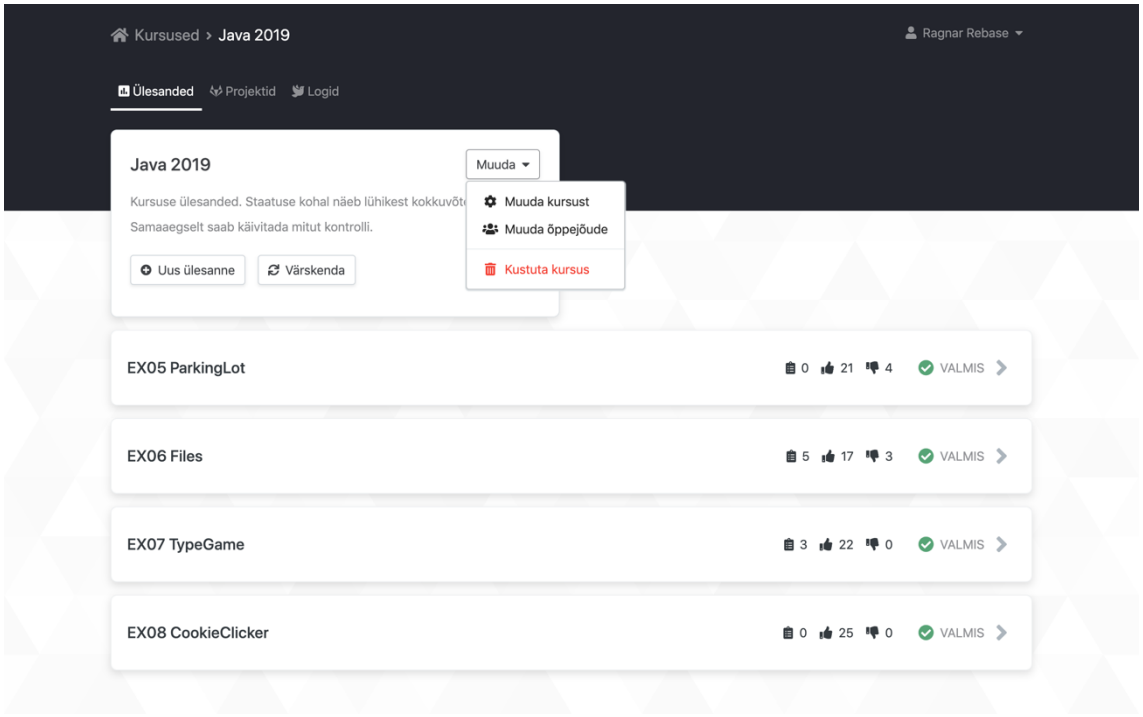
Lisa 3 – Kursuste vaade



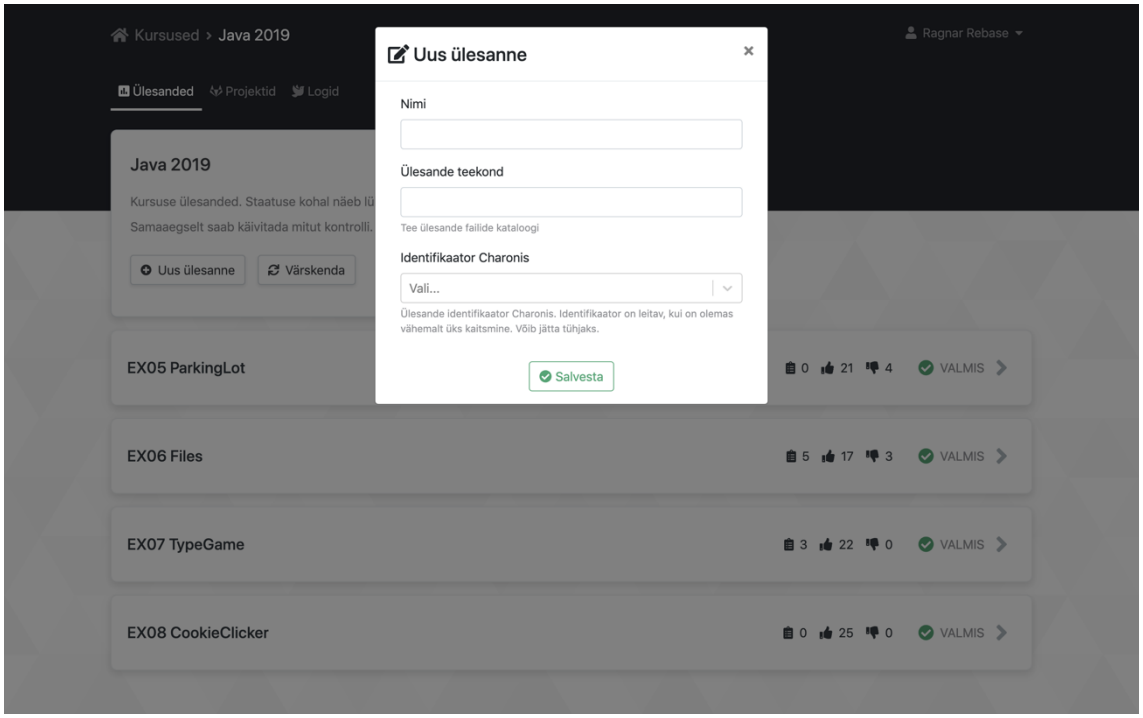
Lisa 4 – Kursuse loomise vorm



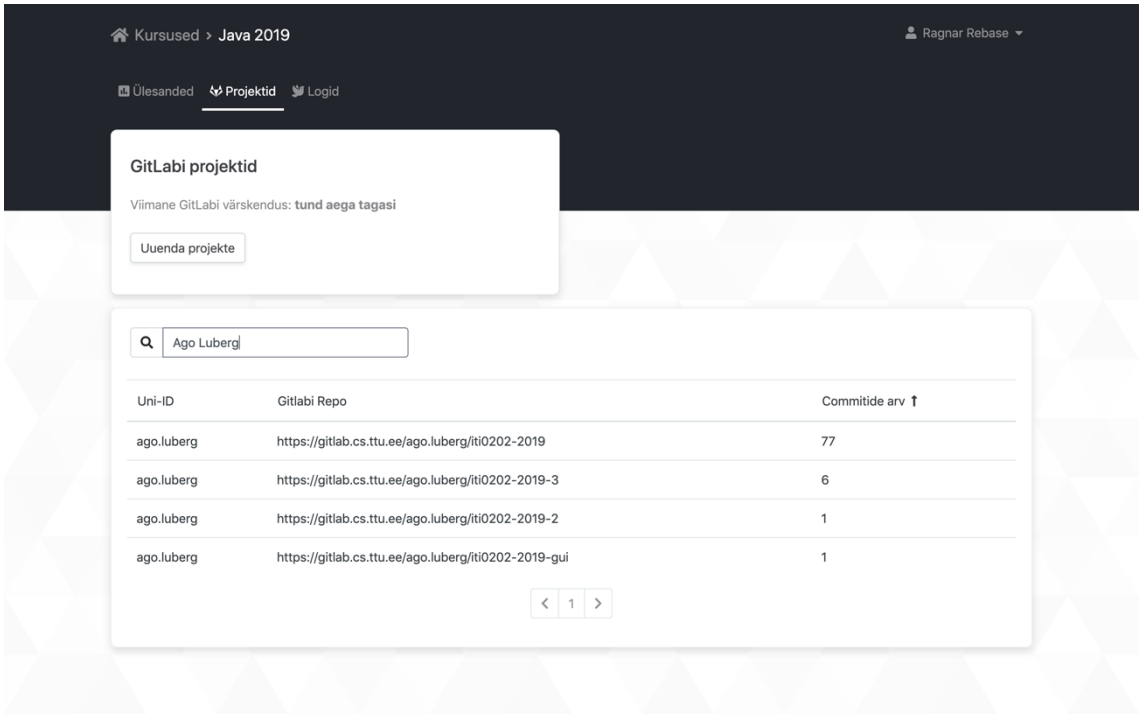
Lisa 5 – Kursuse ülesannete vaade



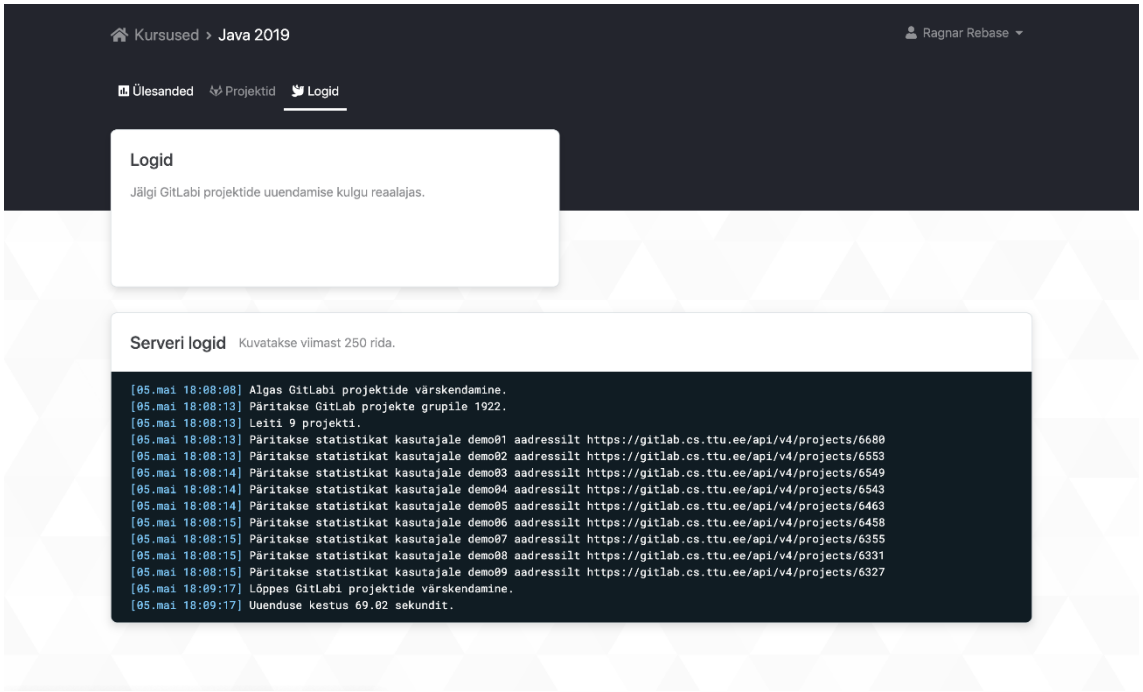
Lisa 6 – Ülesande loomise vorm



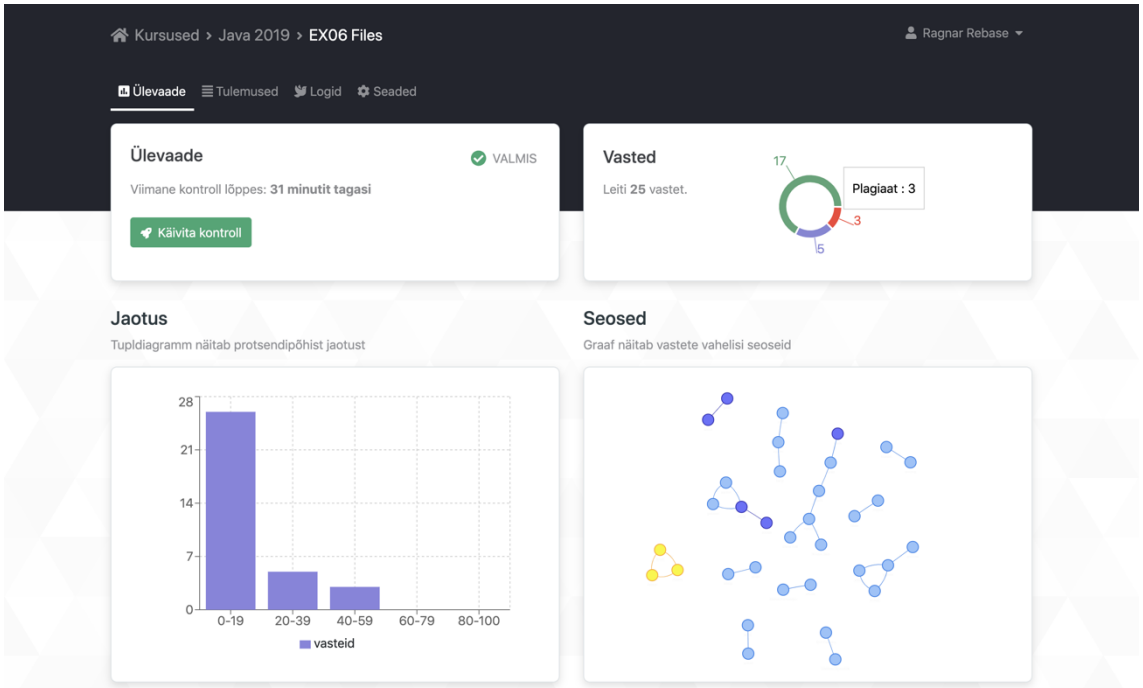
Lisa 7 – Kursuse GitLabi projektid



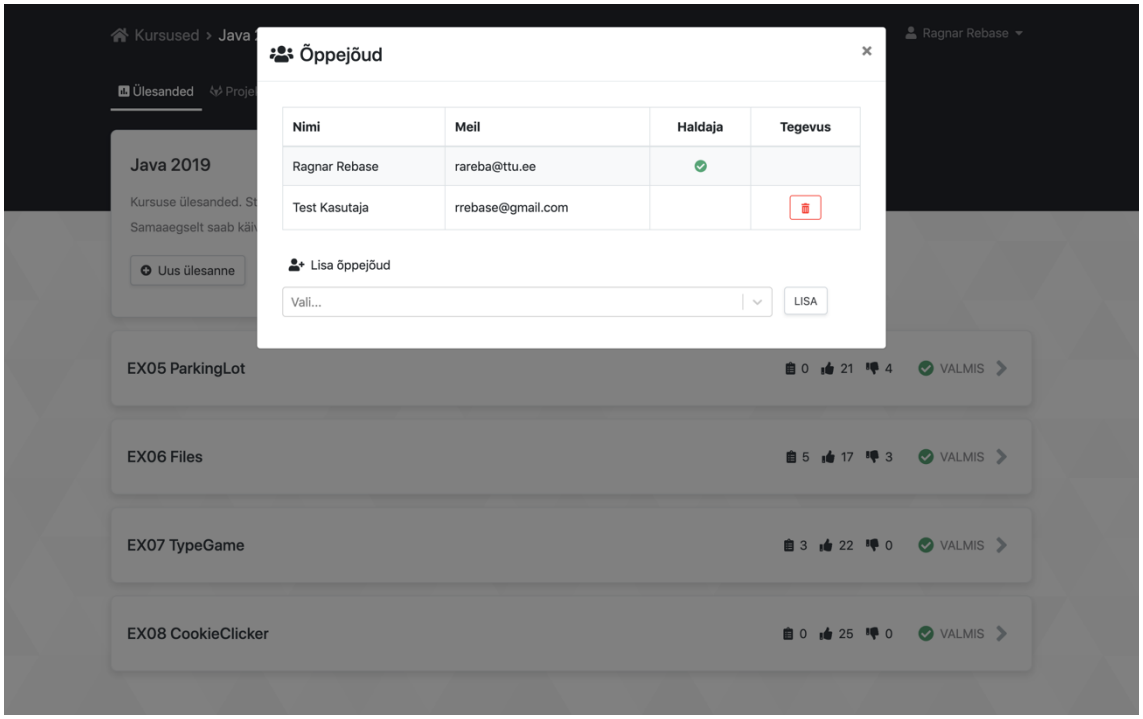
Lisa 8 – Kursuse logid



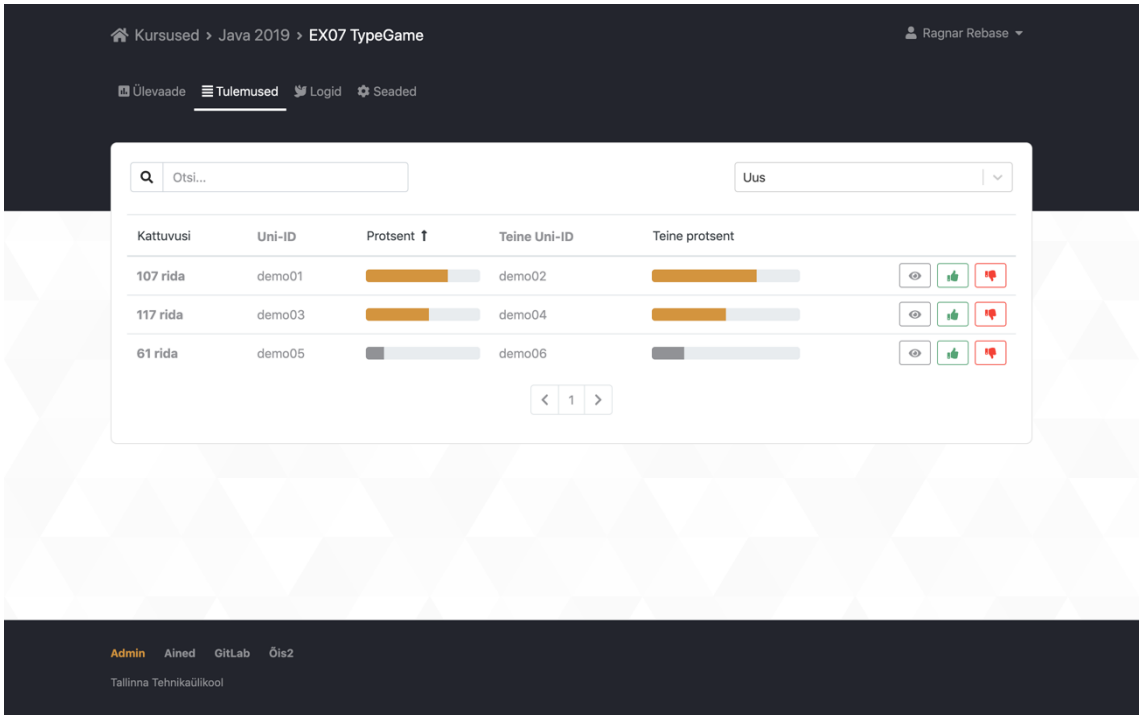
Lisa 9 – Ülesande ülevaatlik vaade



Lisa 10 – Kursuse õppejõudude muutmise modaalaken



Lisa 11 – Plagiaadikontrolli tulemuste vaade



Lisa 12 – Vaste rakendusliidese vaatekomplekt

```
class MatchViewSet(ModelViewSet):
    queryset = Match.objects.all()
    permission_classes = (IsAuthenticated, AssistantReadOnly)
    serializer_class = SimpleMatchSerializer
    pagination_class = StandardResultsSetPagination
    filter_backends = (DjangoFilterBackend, SearchFilter, OrderingFilter)
    search_fields = (
        'submission__gitlab_project__owner__uniid',
        'other_submission__gitlab_project__owner__uniid',
    )
    filter_fields = (
        'submission__assignment__slug',
        'status',
    )
    ordering_fields = (
        'lines_matched',
        'percentage',
        'other_percentage',
        'submission__gitlab_project__owner__uniid',
        'other_submission__gitlab_project__owner__uniid',
    )

    def get_queryset(self):
        qs = super().get_queryset()
        user = self.request.user

        qs = qs.prefetch_related(
            'submission__gitlab_project__owner',
            'other_submission__gitlab_project__owner',
        )

        if user.is_superuser:
            return qs
        return qs.filter(submission__assignment__course__teachers=user)

    def get_serializer_class(self):
        if self.action == 'retrieve':
            return MatchSerializer
        return super().get_serializer_class()

    @action(detail=True, methods=['post'])
    def mark_plagiarism(self, *args, **kwargs):
        match = self.get_object()
        match.status = Match.STATUS.plagiarism
        match.save()
        return Response(status=status.HTTP_200_OK)
```