TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Erik Priis 155196IAPB

OLEMASOLEVA PLAGIAADISÜSTEEMI PROTOTÜÜBI JA KASUTAJALIIDESE REALISEERIMINE JAVAS

Bakalaureusetöö

Juhendaja: Ago Luberg
MSc

Tallinn 2018

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: Erik Priis

20.05.2018

Annotatsioon

Antud töö raames luuakse serveri- ja veebirakendused, mis ühendatakse erinevate plagiaadisüsteemidega.

TTÜ-s on kasutusel olemasolev prototüüp Julia API, mille kaudu saab kontrollida üliõpilaste koodi kattuvust. Süsteem on ilma kasutajaliideseta ning kirjutatud PHP-s.

Töö eesmärgiks on kirjutada olemasolev liidese prototüüp ümber Java platvormile ning arendada kasutajaliides, mille kaudu saab serveriga mugavalt suhelda. Kasutajaliides peab võimaldama kontrollida erinevate lahenduste sarnasust. Töös uuritakse, mille abil on võimalik teha Producer-Consumer rakenduse arhitektuuri mustrit Java-s, mida kasutatakse asünkroonsete sõnumite vahetamiseks. Samuti võrreldakse eelised ja puudused erinevate raamistike vahel nii serveri- kui veebirakenduste kirjutamiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 26 leheküljel, 6 peatükki, 17 joonist, 0 tabelit.

Abstract

Realization of the existing plagiarism system prototype and user interface in Java

The purpose of this thesis is creating a server and web applications that are going to be connected to different plagiarism systems.

The TUT has the Julia API's existing prototype interface, which allows to control students work for the plagiarism. The system has no user interface and is written in PHP.

The basic objective of this thesis is to rewrite the existing interface prototype to Java platform and develop a user interface that can be used to communicate with the server conveniently, which must also be able to control the similarity of the plagiarism based on various works. This thesis is researching the possibilities of using Producer-Consumer pattern in Java, which is used in asynchronous messaging. It also compares the advantages and disadvantages of using a different frameworks for server and web applications.

The thesis is in Estonian and contains 26 pages of text, 6 chapters, 17 figures, 0 tables.

Lühendite ja mõistete sõnastik

AMQP	*Advanced Message Queuing Protocol*, sõnumijärjekorra protokoll
API	Tarkvara, mille liides on disainitud suhtlema teiste programmidega
Docker	Arvutiprogramm, mille kaudu tehakse virtualiseerimist ehk konteineriseerimist
FTP	*File Transfer Protocol*, arvutivõrgu protokoll, mida kasutatakse failide vahetamiseks ja muutmiseks
Git	Hajutatud versioonihaldussüsteem
HTML	*HypetText Markup Language*, keel, mille abil on võimalik luua veebilehte
HTTP	*HyperText Transfer Protocol*, protokoll andmete edastamise aluseks veebis
JDBC	*Java DataBase Connectivity*, API, mis annab kliendile võimaluse üheneda andmebaasiga
JGit	Plagiaadisüsteem
JMS	*Java Message Service*, API klientide sõnumite saatmiseks
JPA	*Java Persistence API*, annab võimaluse salvestada Java objektid andmebaasi
JWT	*JSON Web Token*, autentimisviis
MOSS	Plagiaadisüsteem
MVC	*Model-View-Controller*, veebirakenduse disainimuster
MVP	*Minimum Viable Product*, toode minimaalse ning piisava funktsionaalsusega
ORM	*Object-Relational Mapping*, virtuaalobjektide andmebaas
POM	*Project Object Model*, Maven-i põhiüksus
REST	Tarkvaraarhitektuuri stiil
Socket	Sideprotokoll arvutivõrgus, mille eesmärk on ühenduse loomine kahe või rohkem arvutite vahel ning andmete edastamine nende vahel.
SSH	*Secure Shell*, krüptograafiline protokoll, mille abil luuakse turvaline ühendus
SVN	*Apache Subversion*, versioonide kontrollsüsteem
Token	Unikaalne autentimise tõend, mis genereeritakse serveris ja antakse kliendile
Tomcat	Veebiserver
WAR	*Web Application Resource*, faili formaat, kirjeldab kuidas veebirakendus pakitakse JAR failina
XML	*eXtensible Markup Language*, üldotstarbeline märgistuskeel

Sisukord

Jooniste loetelu

1 Sissejuhatus

Alates 2016. aastast on TTÜ-s on kasutusel selline süsteem nagu Julia API, mille realiseeris Artur Luik [1]. Süsteem võimaldab kontrollida erinevate koodifailide sarnasust. TTÜ-s on erinevad ained, mis kasutavad erinevaid programmeerimiskeeli. Selleks, et hinnata üliõpilaste töid, peaks enne plagiaadi kontrolli tegema, mille kaudu tehakse otsus, et kas töö on teinud üliõpilane ise või tehtud teise üliõpilase pealt maha.

Kasutades Julia API-t on võimalik käivitada erinevaid plagiaadisüsteeme nagu JPlag, MOSS ja Sherlock ühe liidese kaudu. See ei ole mitte ainult mugavam ja abstraktsem, vaid ka praktilisem, kuna iga plagiaadisüsteem kasutab omaenda liidest, mille kaudu suhtlemine REST API ja süsteemide vahel on keeruline ja mittepraktiline. Liideste ühendamine lihtsustab suhtlust REST API ja plagiaadisüsteemide vahel.

Töö eesmärgiks on kirjutada olemasolev liides ümber Java platvormile ning arendada kasutajaliides, mille kaudu saab suhelda serveriga mugavalt ning võrrelda kattuvate tööde sarnasust.

2 Analüüs

Antud peatüki eesmärgiks on tutvuda juba olemasoleva plagiaaditeenuste ja andmeallikate ühendatud liidese prototüübiga ning selle tehnoloogiate valikuga.

2.1 Julia API

Julia – on universaalne plagiaadi kontrollija, mis võimaldab kontrollida plagiaati, muretsemata plagiaaditeenuste või ressursside pakkujate tehniliste omaduste üle.

Rakendus on kirjutatud täielikult PHP-s, kuid kasutajaliides puudub, mille tulemusena juhtimine ja suhtlemine süsteemiga toimub läbi API, mis on teostatud erinevate HTTP päringute kaudu. Rakendus kasutab 3 erinevat plagiaadisüsteemi: JPlag[1], MOSS[2], Sherlock[3]. Ressursside (ehk näiteks tudengi koodi) alla laadimiseks on kasutusel nii Git kui ka SVN. Rakendus on ehitatud Customer-Producer disainimustri arhitektuuril, mis annab võimaluse kasutada järjekorda ja asünkroonset suhtlemist plagiaadisüsteemide, ressursside teenuste ja lõpptulemuste vahel, mis salvestatakse lõpuks andmebaasi. Andmebaasina kasutatakse SQL andmebaasi, mida juhistakse läbi Eloquent ORM'i.

Rakenduses on salvestatud migratsiooni skriptid, mille abil on võimalik vahetada SQL andmebaasi ilma koodi ümberkirjutamata. Eloquent ORM on PHP Laravel programmeerimistehnoloogia, mis ühendab andmebaasid objektorienteeritud programmeerimiskeelte mõistetega, luues niinimetatud "virtuaalobjektide andmebaasi" [2].

Rakenduse uurimisel ja analüüsimisel tuvastati mõned arhitektuuriprobleemid. Rakendus salvestab ebakorrektselt Git-i URL-i, mis saadeti plagiaadisüsteemi. See viga on seotud räsi kokkupõrkega. Süsteem genereerib omaenda räsi võrreldes alla laaditud allika faili

[1] https://jplag.ipd.kit.edu/

[2] http://theory.stanford.edu/~aiken/moss/

[3] http://rp-www.cs.usyd.edu.au/~scilect/sherlock/

nimega. Seega, kui kaks failid sama nimega laetakse erinevatest allikatest, siis annab süsteem neile samasuguse räsi, mille tagajärjel kaob seos autorsuse ja URL-i vahel, kust iga fail oli alla laaditud. Selle vea parandamiseks on vaja kas keelduda või muuta failide räsimise funktsiooni, aga kuna viga tuleb rakenduse arhitektuurist, siis selle parandamise on väga keeruline, sest rakenduse arhitektuur on vaja täielikult ümber kirjutada.

See viga ei võimalda antud süsteemi edasiarendamist ning sellele kasutajaliidese loomist. Lisaks oli autoril plaanis ühtse üliõpilase plagiaadiprofiili loomine, mis võimaldaks õppejõududel eelnevalt hinnata, millises mahus on üliõpilane plagiaadiga seotud. Eelnimetatud vea tõttu ei suutnud autor sellega tegeleda ning seetõttu vahetas autor nii eesmärki kui ka tööülesanded.

Kuna autor ei ole PHP-s tugev, siis võeti plaani selle prototüübi ümber kirjutamine mõnda teise programmeerimiskeelde, millega autor oli tuttav.

2.2 Uue programmeerimiskeele valimine serverirakenduse arendamiseks

Selle töö üheks ülesandeks on olemasoleva rakenduse ümber kirjutamine. Selles peatükis vaadeldakse mitut programmeerimiskeelt, mille peal on rakendust võimalik ümber kirjutada. Oma valikul tugines autor järgmistele punktidele:

- Programmeerimiskeele raamistik peab toetama MVC mustrit, mis lihtsustab veebirakenduse tuge ja arendamist
- Ülikoolis peab olema aine, kus õpetatakse seda programmeerimiskeelt selleks, et tudengid saaksid tulevikus seda süsteemi edasi areneda
- Programmeerimiskeel peab olema tuttav autorile

Programmeerimiskeelena oli valitud kolm peamist keelt: Java, Python, C#. Kõiki kolme keelt õpetatakse ülikoolis ning kõigil on MVC raamistik, mis vastab antud tingimustele nende valimiseks.

Java-s on Spring raamistik, mis kombineerib paljusid erinevaid mooduleid kiireks ja lihtsaks arendamiseks ning veebirakenduse toeks. Annotatsioonide abil saab rakendust seadistada ilma püsiprogrammeerimiseta (*hard code*). Samuti kasutatakse Spring-is java

serialiseeritavaid standardklasse (*bean*), mis võimaldavad ühendada mooduleid sõltuvuste sisestamisega (*dependency injection*), seega mooduleid on võimalik automaatselt ühendada ning neid hallata, lihtsustades programmeerija elu [3].

C# keeles on nii Java Spring raamistiku analoog Spring.NET kui ka populaarsem ASP.NET. Ülikoolis õpitakse C# keelt äriinfotehnoloogi erialal, kuid informaatika erialal seda keelt ei õpita.

Python-is on Django raamistik, kus erinevalt MVC standardist, kasutatakse MVT-i (model-view-template) mustrit, mis on tehniliselt sama nagu MVC. Django ei ole nii populaarne kui Spring või ASP.NET, seega nii kolmandate osapoolte väljatöötatud dokumentatsioon kui ka pakutavate moodulite arv on väiksem [4].

Autor on tuttav Java Spring-iga, mis võimaldab mitte keskenduda uue raamistiku õppimisele. Samuti on autor pigem tuttav Java-ga kui C#-ga. Kõik eespoole toodud raamistikud ei erine üksteisest oluliselt, et luua vajalikku veebirakendust, sest nad kõik kasutavad analoogset MVC mustrit. Samas süsteemi edasi arendamist on jätkamas suurem tõenäosusega informaatika eriala üliõpilane, seega sai valitud Java Spring raamistik serverirakenduse arendamiseks.

2.3 Uue raamistiku valimine veebirakenduse arendamiseks

Antud osas esitatakse erinevat tüüpi veebisaitide tehnoloogiate eelised ja puudused ning kaalutakse ka reaktiivse programmitöö paradigmat. Osa lõpus loetletakse praegused reaktiivsed programmeerimisraamistikud ja nende seas valitakse kas üks või teine raamistik, millega antud rakendus arendatakse [5].

2.3.1 Staatilise, dünaamilise ja reaktiivse veebirakenduse võrdlus

Staatiline veebisait on veebileht, mis koosneb staatilistest HTML lehtedest, mis moodustavad ühe terviku. Kõik saidi muudatused tehakse saidi lehtede lähtekoodile, mille muutumise jaoks on vaja saada liigipääsu veebiserverisse nendele failidele.

Eelised:

- Veebilehe loomiseks ei ole vaja teada programmeerimiskeelt
- Hea lehe vahemällu salvestamine

- Lehekülje kiire laadimine
- Minimaalsed veebiserveri nõuded ja selle minimaalse koormuse mõju
- Lihtne üle kanda teise serverisse või kohalikku arvutisse
- Võimalus vaadata faile otse brauseris vahetarkvara kasutamata

Puudused:

- Võimetus dünaamiliselt luua HTML-i sisu
- HTML-i sisu muutmiseks on vaja saada liigipääsu failidele
- Piiratud funktsionaalsus

Dünaamiline veebisait on veebileht, mis koosneb dünaamilistest lehtedest – mallid, sisu, skriptid jm. Kasutajale kuvatav leht genereeritakse serverirakenduses. Server genereerib veebilehe, mis võimaldab erinevate päringute kaudu saada erinevaid andmeid [6].

Eelised:

- Veebilehtede arvu suurenemisega on dünaamilise saidi hallatus võrreldes staatilistega üsna lihtne, sest kui on vaja teha sarnaseid muudatusi, siis koodi muutmine on võimalik teha ühes kohas, kuid seda muudatust rakendatakse kõikidel lehtedel
- Erinevad andmed võimaldavad suhelda kasutajaga
- Mugavam navigatsioon
- Rohkem funktsionaalsuse võimalusi

Puudused

- Võrreldes staatiliste lehtedega on alla laadimise kiirus väiksem
- Aeglasem arendamine

Reaktiivne veebisait on veebileht, mis põhineb reaktiivse programmeerimise paradigmal, mis võimaldab muuta reaalajas andmeid. Nende erinevus seisneb selles, et kui dünaamilised veebisaidid luuakse tavaliselt serveripoolel, siis reaktiivse lahenduse korral luuakse see kliendi pool reaalajas, mis võimaldab mitte ainult vähendada serveri koormust, vaid ka vähendada kliendilt serverisse saadetavate päringute arvu.

2.3.2 Reaktiivse programmeerimise raamistikud

Siin peatükis vaadeldakse kolme kõige populaarsemat reaktiivse programmeerimise raamistikku – need on Angular, React ja Vue. Kõige populaarsem neist on React, Angular on kõige funktsionaalsem ning Vue on kõige noorem, kuid sarnase funktsionaalsusega nagu React. Oluline erinevus Vue-i ja React-i vahel on selles, et Vue on ehitatud HTML-i märkimise baasil, mis ei kohusta põhjalikult raamistiku meetodeid uurima. Selline meetod lihtsustab rakenduse arendamist ning vähendab sellele kuuluvat aega. Angular-i aluseks on direktiivid, mis muudavad koodi aeg-ajalt keerukamaks, eriti väikeses rakenduses, nagu on antud projekt. Vue on React-iga võrreldes lihtsam ning praegu arendatakse kolmanda osapoole mooduleid kiiremini, kuna Vue on muutumas populaarsemaks. See tähendab seda, et React-il ei ole eeliseid võrreldes Vue-ga [7].

2.4 Andmebaasi valik

On olemas kolme erinevat tüüpi andmebaase: SQL, NoSQL, Graph.

Julia API kasutab relatsioonilist andmebaasi. Sealne andmebaasiskeem on toodud lisas 1. Selles on palju seoseid, mida tegelikult rakenduses ei kasutata, ehk mis ei ole realiseeritud. Antud töös realiseeritakse vaid vajalikud tabelid ja seosed. Näiteks kasutajate ja gruppidega seonduv jääb käesolevas realisatsioonis ära (seda ei kasutatud ka vanas süsteemis). Päringud süsteemile saadetakse JSON-formaadis, seega realisatsioonis keskendutakse JSON-tüüpi andmete salvestamisele.

Kuna Graph põhineb seostel, siis seda peaks kasutama mudeliga, kus kasutatakse palju seoseid. Selles töös on vaja ainult kahte seost: accountId ja checksuiteId, muudel juhtudel on võimalik ühendustest loobuda, mis muudab Graph tüüpi andmebaasi mõttetuks.

SQL-i puhul võiks öelda seda, et seda tüüpi andmebaasi on NoSQL-i ja Graph-i vahel, kus kasutatakse mõlemat: nii dokumenditüüpi kui ka objektide vahelist seost. Selles töös

keskendutakse rohkem dokumendiandmetele, kus peamiselt JSON-i andmetüübi jaoks ei ole SQL kasutamine mõttekas.

NoSQL on selle töö jaoks suurepärane, kuna see kasutab dokumenditüüpilist andmete salvestamist JSON-is, mis on mugav objekti serialiseerimiseks ja deserialiseerimiseks ilma täiendavate objektide seoste loomiseta.

Kõige populaarsem NoSQL-tüüpi andmebaas on MongoDB.

2.5 Rakenduse nõuetest

Rakendus realiseeritakse lähtudes MVP printsiibist, mis tähendab, et realiseeritakse minimaalne, aga piisav funktsionaalsus, et rahuldada lõpptarbijat. Siin tuuakse välja vastavad nõuded:

- Kasutada 2 plagiaadisüsteemi: JPlag ja MOSS. Sherlock-it pole seni kordagi kasutatud ning sellel ei ole eelised teiste plagiaadisüsteemide ees.
- Kasutada ainult Git-i ressursside alla laadimiseks. SVN on juba vana süsteem, mida pole seni kordagi kasutatud.
- Realiseerida Docker-i abil konteineriseeritud rakendus, millega saab rakendust juurutada kergema vaevaga.
- Kasutajaliideses peavad olema realiseeritud autentimise funktsioon ning vaated:
    - Checksuite vaade, kus kuvatakse nimekiri kõikidest lisatud kontrollidest
    - Check detailsem vaade, kus võib vaadata failide ja nende ridade sarnasust
    - Uue kontrolli lisamine
    - API dokumentatsioon

3 Arendamisel kasutatud tehnoloogiad

3.1 Spring Boot framework

Spring-i raamistik pakub mitmekülgset arendamis- ja konfiguratsioonimudelit Java ärirakendustele, erinevatel platvormidel. Spring-i põhielement on rakenduse tasemel infrastruktuuri tugi: keskendutakse ärirakenduste seosele, seega arendajad saavad keskenduda äriloogikale ilma vajaduseta tegeleda seadistamisega sõltuvalt töö keskkonnast [9].

Võimalused:

- Sõltuvuse sisestamine
- Aspekt-orienteeritud programmeerimine, sh deklaratiivne tehingute haldamine
- Spring-i MVC veebirakenduste ja REST veebiteenuste loomine
- JDBC, JPA, JMS esialgne tugi

Spring Boot raamistik võimaldab luua täiemahulisi Spring rakendusi, mille seadistamine ja käivitamine on lihtsamaks tehtud. See sisaldab enamlevinud kolmandate osapoolte tarkvara ning seadistamine on tehtud nii mugavaks, et enamik Spring Boot-i rakendused vajavad väga väikest konfiguratsiooni või töötavad isegi ilma konfiguratsioonita [10].

Võimalused:

- Täieõiguslik Spring-i rakenduse loomine
- Sisseehitatud Tomcat (WAR failide installimist pole vaja)
- Pakub esialgseid POM-e, mis lihtsustavad Maven-i konfiguratsiooni.
- Võimaluse korral on automaatne Spring-i konfiguratsioon

- Pakub selliseid võimalusi nagu meetrika, staatuse jälgimine ja laiendatud konfiguratsioon
- Absoluutselt ilma koodi genereerimiseta ja XML konfiguratsiooni kirjutamata

3.2 Vue framework

Vue on raamistik, mis võimaldab kasutada reaktiivse programmeerimise paradigmat. Selline paradigma võimaldab kliendipoolsete andmete reaalajas muutmist, viidates serverile minimaalse arvu kordi. Kogu lehekülje kuvamine toimub kliendi poolel, mis aitab vähendada serveri koormust, erinevalt dünaamilisest veebisaidist, kus nii serveris kui ka kliendi poolel toimuvad töötlemised.

Vue raamistikku on võimalik installida ka käsurealiidese (CLI) kaudu, mis hõlbustab projekti sõltuvuste rakendamist NPM pakettide kaudu. NPM paketid võimaldavad kasutada modulaarset arhitektuuri, jagades rakenduse erinevate moodulite vahel, midagi sarnast objektorienteeritud programmeerimise paradigmaga.

Saadaval on mitmeid valmislahendused kasutajaliidese jaoks, näiteks @Websanova/Vue-Auth, mis võimaldab kasutada JWT-autentimise viisi. Bootstrap Vue võimaldab kasutada Twitter-i Bootstrap 4 versiooni, mis põhineb Vue alusel, mitte jQuery-il, mis annab ka võimalusi kasutada mõnda reaktiivse programmeerimise elementi, näiteks tabel, kus andmete muutmine toimub kliendil reaalajas [11][12].

Autor on tuttav Vue raamistikuga, mille tagajärjel React-i, Angular-i ja Vue-i hulgast valiti Vue, sest selle funktsioonide hulk on React-iga sarnane, kuid raamistik on lihtsam ja arusaadavam, kuna kasutab HTML-iga peaaegu identset markeeringu koodi.

3.3 MongoDB

MongoDB kasutab erinevalt tavalisest relatsioonilisest andmebaasi süsteemist teistsugust struktuuri, kus ei ole tabeleid, skeeme, SQL-päringuid, välisvõtmeid, ja palju teisi asju, mis on omavahel seotud objektide-relatsioonandmebaasidega [13]. Tavaliselt kasutatakse kõigi andmete säilitamiseks relatsioonandmebaase (MSSQL, MySQL, Oracle, PostgreSQL). Tihti relatsiooniliste andmebaaside puhul üldse ei mõelda, kas need on kõige sobivamad rakenduse andmete salvestamiseks. Erinevalt relatsioonandmebaasidest

pakub MongoDB dokumendil põhinevat andmemudelit, mis teeb MongoDB kiiremaks ja seda on lihtsam kasutamises teatud tüüpi andmete puhul.

Kui relatsioonilistes andmebaasides salvestatakse andmed tabelitesse, siis MongoDB puhul salvestatakse andmed dokumentides ehk kollektsioonides. Erinevalt ridadest, mis kasutatakse tabelite puhul, võivad dokumendid salvestada keeruka struktuuriga andmeid. Dokumenti saab esitada võtmete ja väärtuste hoidlana. Võti kujutab endast lihtsat märki, millega on seotud konkreetne andmehulk.

Kuid kõigi erinevuste puhul on üks sarnasus, mis toob MongoDB ja relatsioonilised andmebaasid üksteisele lähemale. Relatsioonandmebaasisüsteemides on selline asi nagu primaarvõti. See mõiste kirjeldab teatud veergu, milles on unikaalsed väärtused. MongoDB-s on iga dokumendi jaoks unikaalne tunnus nimega *_id*. Kui selle väärtust ei määrata, siis MongoDB loob selle jaoks väärtuse automaatselt.

Iga võtme jaoks on määratud konkreetne väärtus. Kuid siin peame arvestama ka ühe võimalusega: kui relatsioonandmebaasis on selgelt välja toodud struktuur, kus mõni väli ei ole kohustuslik, siis (sõltuvalt konkreetse andmebaasi seadetest) saab välja väärtuseks määrata NULL. MongoDB-ga on see erinev. Kui võtme väärtust ei seostata, siis see võti lihtsalt jäetakse dokumendist välja ja seda ei kasutata.

3.3.1 Kollektsioon

Kui traditsioonilises SQL-i maailmas on tabeled, siis on MongoDB- maailmas kollektsioonid. Kui relatsioonandmebaasis salvestatakse andmeid jäigalt struktureeritud struktuuriga, siis kollektsioon võib sisaldada erinevaid objekte, millel on erinev struktuur ja erinev omaduste komplekt.

3.3.2 Replikatsioon

MongoDB andmesalvestussüsteem kujutab endast koopiate komplekti. Selles komplektis paikneb üks peamine sõlm ning teised sõlmede komplektid. Kõik sekundaarsed sõlmed säilitavad terviklikkuse ja uuendatakse automaatselt koos põhiosa värskendusega. Kui põhiosa mingil põhjusel ebaõnnestub, siis muutub üks sekundaarsetest sõlmedest peamiseks.

3.3.3 Lihtne kasutamisel

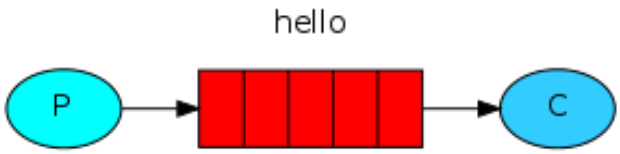
Konkreetse andmebaasi skeemi puudumine ja seega vajadus selle skeemi taastamiseks koos andmesalvestuse muutumisega muudavad MongoDB andmebaasiga töötamise palju lihtsamaks. Lisaks säästavad arendajad aega - nad ei pea enam mõtlema andmebaasi taastamisele ja keerukate päringute loomise ajakulule.

Kuna projektis on enamus objekte esitatud JSON andmetüübina, siis on otsustatud kasutada NoSQL-i dokumendi andmebaasi. Selliste seas on MongoDB kõige populaarsem, kuid MongoDB päringud erinevad tavalisest SQL-ist

3.4 RabbitMQ, Producer-Consumer realisatsioon

RabbitMQ on sõnumivahendaja, mis hoolitseb sõnumite vastuvõtmise, suunamise ja väljastamise eest. Sõnumid koosnevad kehast ja päisest [14]. Marsruudi aluseks on kommunikatsioonivahetus ja järjekord. Järjekord on tüübi järgi jagatud nii prioriteetseks kui tavaliseks.

Selle rakenduse arhitektuur põhineb asünkroonsel Publish-Subscribe mustril, mis omakorda võimaldab rakendada Producer-Consumer mustrit.



Joonis 1. Producer-Consumer RabbitMQ realisatsioon

PHP Laravel-is on olemas tööd (*jobs*), kuid Java Spring-is selline realisatsioon puudub. Antud töös ei hakatud looma eraldi realisatsiooni vaid võeti kasutusele RabbitMQ, kuna AMQP protokolli on lihtsam rakendada.

3.5 Docker

Docker on avatud platvorm rakenduste arendamiseks, tarnimiseks ja kasutamiseks [15]. Docker on loodud rakenduste kiireks käivitamiseks. Docker-i abil on võimalik oma rakendust eraldada infrastruktuurist ja hallata seda kui eraldi süsteemi.

Docker-i abil saab koodi kiiremini hallata, kiiremini katsetada, rakendust kiiremini juurutada (deploy), kulutada vähem aega koodi kirjutamise ja koodi käivitamise vahel. Docker teeb seda kergete konteinerite virtualiseerimisplatvormiga, kasutades protsesse ja teenuseid, mis aitavad rakendusi hallata ja üles laadida.

Tegelikult on Docker sarnane virtuaalmasinaga, kuid erinevus seisneb selles, et Docker on väikse mahuga ja ei vaja külalismasinas täismahus operatsioonsüsteemi installeerimist erinevalt virtuaalse masinast. Külalis operatsioonisüsteemide puudumise tõttu suureneb ka rakenduse kiirus, mis jällegi ei lähe virtuaal masinate võrdlusse. Rakendus Docker-is on isoleeritud ehk sisesed rakendused ei saa ligi väljapoole, mis teeb konteinerid ideaalseks tooteserveriks. Kõige olulisem eelis on selles, et sõltuvuste installeerimine tehakse Dockerfile-i kaudu ning selle rakenduse käivitamine tehakse ühe käsuga, kus kasutajal ei ole vaja kirjutada eraldi skripte, sest kõik on automatiseeritud.

4 Serverirakenduse arendamine

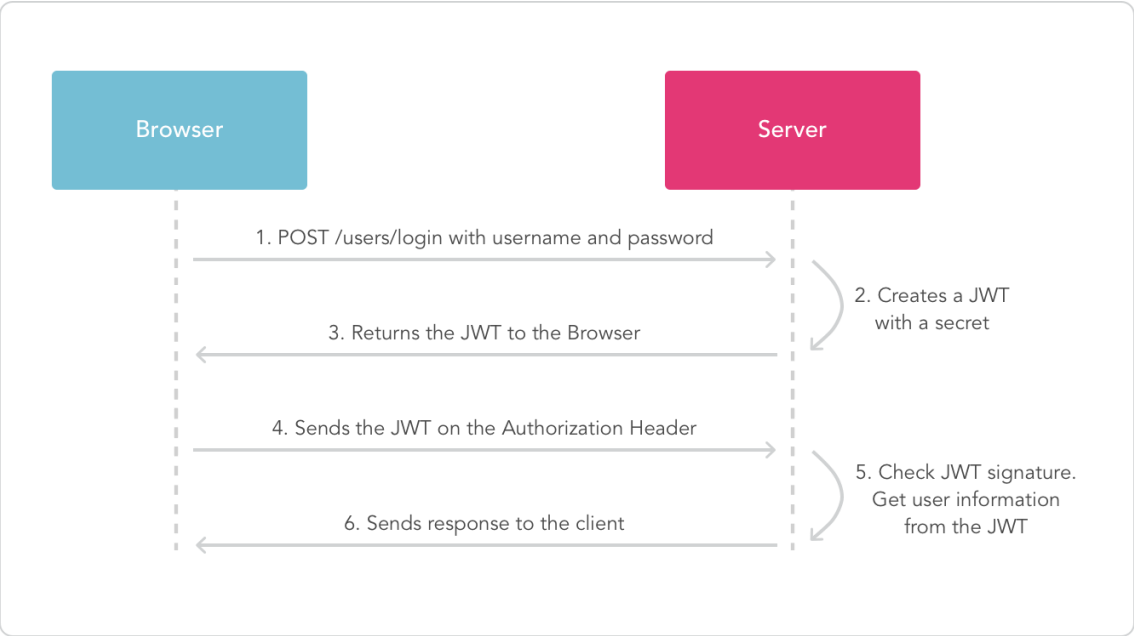
Selles osas käsitletakse süsteemi arendamist ja juurutamist.

4.1 Autentimine

Spring-i abil on võimalik autentida tava autentimise (Basic Auth) meetodiga [16]. See on tavaline kasutajanime ja parooliga sisselogimine. Selline meetod on ebamugav sisselogimiseks REST API kaudu, kuid antud rakendus kasutab just REST API-t. Kuna sisselogimine võib olla tehtud ka päringute kaudu, siis on vaja kasutada sellist autentimisviisi, millega on võimalik kliendi saadetud päringute kaudu autentida.

Julia API-s on kasutusel just päise kaudu autentimine. Päringu saatmisel lingile */api/v1/user/login* toimub autentimine ning kliendile tagastatakse *token*, mis salvestatakse ka andmebaasi. *Token* oli lõputu elueaga (*lifetime*), seega kui korra oli autentimine tehtud, ei olnud seda rohkem vaja teha. Autentimise meetod oli kirjutatud ilma olemasoleva tehnoloogia baasita ehk oli eraldi loodud selle projekti jaoks. Ühekordne autentimine on ebaturvaline ning sellist *token*-it võib kompromiteerida lihtsasti.

Uue autentimisviisina on valitud JWT tehnoloogia, mis võimaldab seda paindlikult kasutada ja konfigureerida serveripoolseid teatud parameetreid [17]. Java ümbrisena on kasutusel *io.jsonwebtoken:jjwt:0.9.0*, mida on mugav konfigureerida. Selle tehnoloogia kasutamise tööprintsiip on esitatud Joonisel 2.



Joonis 2. JWT tokeni tööprintsiip

1. Autentimise korral saadab brauser HTTP päringu kaudu kasutajaandmed: kasutajanimi ja parool
2. Server loob *token*'i selle alusel
3. Server tagastab klindile *token*'i
4. Klient saadab suvalise päringu koos *token*'iga päises
5. Server kontrollib *token*'i kehtivust, kasutajaandmed on salvestatud *token*'i sees
6. Kehtivuse korral vastab server kliendile HTTP vastusega 200 ja saadab nõutud andmed, aga kui *token* on kehtetu, siis saadab server kliendile HTTP vastusega 401 ning nõutud andmeid ei saadeta enne, kui kasutaja on uuesti autoriseeritud kehtiva *token*'iga

Praegu on rakenduses realiseeritud mitte perfektne *token*'i uuendamise funktsioon, mis genereerib iga päringu jaoks uue *token*'i. Ideaalses lahenduses tuleks realiseerida uuendamine eraldi serveris, aga antud MVP puhul seda lõputöö raames ei realiseerita.

4.2 Ressursside alla laadimine

Eelmises rakenduses kasutatakse ressursside alla laadimiseks kahte erinevat allikat: Git ja SVN. Kuna projekti ülesanne on rakendada MVP-d, siis realiseeritakse antud rakenduses vaid ühendus Gitiga, mida kasutatakse palju rohkem kui SVN-i.

Git-is on olemas 4 autentimistüüpi [18]:

1. Ilma autentimiseta läbi HTTPS protokolli
2. Autentimine läbi HTTPS protokolli, kasutades kasutajanime ja parooli
3. Autentimine läbi FTP protokolli
4. Autentimine privaatvõtmega SSH protokolli kaudu

Git-i realisatsioon peab toetama 2. ja 4. autentimistüüpi. *gitlab.cs.ttu.ee* URL-is ei ole võimalik ilma autentimiseta alla laadida tudengite koodi, seega 1. tüüp jääb antud töös realiseerimata. Samuti jääb realiseerimata FTP autentimine.

Git-i Java realisatsioonidest oli valitud selline teek nagu JGit (*org.eclipse.jgit:org.eclipse.jgit:4.11.0.201803080745-r*), mis võimaldab Git-iga suhelda otse Java keeles.

Realisatsioon ilma autentimiseta on näidatud Joonisel 3.

```
CloneCommand cloneCommand = Git.cloneRepository()
       .setURI(configuration.getRepository())
       .setDirectory(…);
```

Joonis 3. JGit kloonimine ilma autentimiseta

`SetURI` määrab Git-i repositooriumi lingi, `setDirectory` omakorda määrab lokaalse hoidla asukoha, kuhu failid paigutatakse peale kloonimist.

Realisatsioon kasutajanime ja parooliga autentimisega on esitatud Joonisel 4.

```
cloneCommand.setCredentialsProvider(new UsernamePasswordCredentialsProvider(
             configuration.getUsername(), password));
```

Joonis 4. JGit autentimine kasutades kasutajanimi ja parool

Sisselogimise vahendajaks kasutatakse Java-s sisseehitatud vahendajat, mille abil parameetreid edastatakse serverisse.

Kolmas autentimisviis on SSH ja privaatvõtme kaudu, mis on kujutatud Joonisel 5.

```
String passphraseValue = Optional.ofNullable(
       configuration.getPassphrase()).orElse("");
byte[] passphrase = null;
final byte[] privateKey = gitCipher.decodeToByte(
       configuration.getPrivateKey());

if (!passphraseValue.isEmpty()) {
       passphrase = gitCipher.decodeToByte(configuration.getPassphrase());
}

final byte[] finalPassphrase = passphrase;
cloneCommand.setTransportConfigCallback(new TransportConfigCallback() {
       @Override
       public void configure(Transport transport) {
              SshTransport sshTransport = (SshTransport) transport;
              sshTransport.setSshSessionFactory(
                           new JschConfigSessionFactory() {
                     @Override
                     protected void configure(
                                  OpenSshConfig.Host hc, Session session) {
                           session.setConfig("StrictHostKeyChecking", "no");
                     }

                     @Override
                     protected JSch createDefaultJSch(FS fs) throws
                     JSchException {
                           JSch jSch = super.createDefaultJSch(fs);
                           jSch.addIdentity(null, privateKey, null,
                           finalPassphrase);
                           return jSch;
                     }
              });
       }
});
```

Joonis 5. JGit autentimine SSH kaudu

Nagu on näha koodist, isiklikud andmed nagu privaatvõtmed, mis on salvestatud andmebaasis kodeeritult, dekodeeritakse kasutades AES-128 standardit. Peale seda kontrollitakse paroolifraasi olemasolu ning selle tuvastamiseks see dekodeeritakse. Seejärel konfigureeritakse SSH ühendus transpordi tasandil, milles kasutatakse dekodeeritud privaatvõtit ja paroolifraasi, mis oli eelnevalt kodeeritud bait massiivina.

Tunneli konfiguratsioonis võib näha ka seda, et hostid määratakse negatiivse väärtusega. See on vajalik selleks, et iga uue hosti lisamisel ei esitata küsimust nende kontrollimiseks, vaid lihtsalt lisatakse ilma küsimuseta. Kuna git-i repositooriumite alla laadimine on piiratud ainult üliõpilaste koodiga ning alla laaditud faile ei kasutata süsteemi käivitamisel, siis otsustati rakenduse lihtsustamiseks vastav kontrollimine eemaldada.

4.3 Plagiaadi tuvastamise keskkonnad

Julia API-s kasutatakse kolme erinevat plagiaadisüsteemi: JPlag, MOSS, Sherlock. Viimast kasutatakse väga vähe, sest kaks esimest teevad sama töö ära ning Sherlock ei paku mingeid lisasid, millega saaks teistest erineda. Selle tagajärjel otsustati see MVP-st välja jätta. Antud rakenduses on tehtud kõik selleks, et uue plagiaadisüsteemi lisamine oleks mugav ning selleks ei pea koodi ümber kirjutama. Vaja on luua uus klass ja siduda see abstraktse klassiga, kuhu peab edastama andmeid plagiaadisüsteemist.

4.3.1 JPlag

JPlag on spetsiaalselt Java jaoks kirjutatud. Seda saab otse Java koodist välja kutsuda. Antud plagiaadisüsteem suudab otsida sarnasusi ainult Java failides, seega ei saa ta teisi keeli sellega kontrollida. JPlag kasutamine koodis on näidatud Joonisel 6.

```
Collection<File> sourceFiles = FileUtils.listFiles(
             new File(source).getAbsoluteFile(), new String[]{"java"}, true);
if (sourceFiles.size() == 0) throw new FileSystemNotFoundException();

Process ps = Runtime.getRuntime().exec(
       new String[] {"java", "-jar", "bin/jplag/jplag-2.11.9.jar", "-s",
                     source, "-r", destination, "-l", "java17"});

if (!ps.waitFor(10, TimeUnit.MINUTES)) {
       ps.destroy();
       throw new InterruptedException("JPlag timeout exceeded");
}

if (ps.exitValue() != 0) {
       String msg = new BufferedReader(
              new InputStreamReader(ps.getErrorStream()))
                     .lines()
                     .parallel()
                     .collect(Collectors.joining("\n"));
       throw new IOException("Exit value = " + ps.exitValue() + "\n" + msg);
}
```

Joonis 6. JPlag koodi realiseerimine

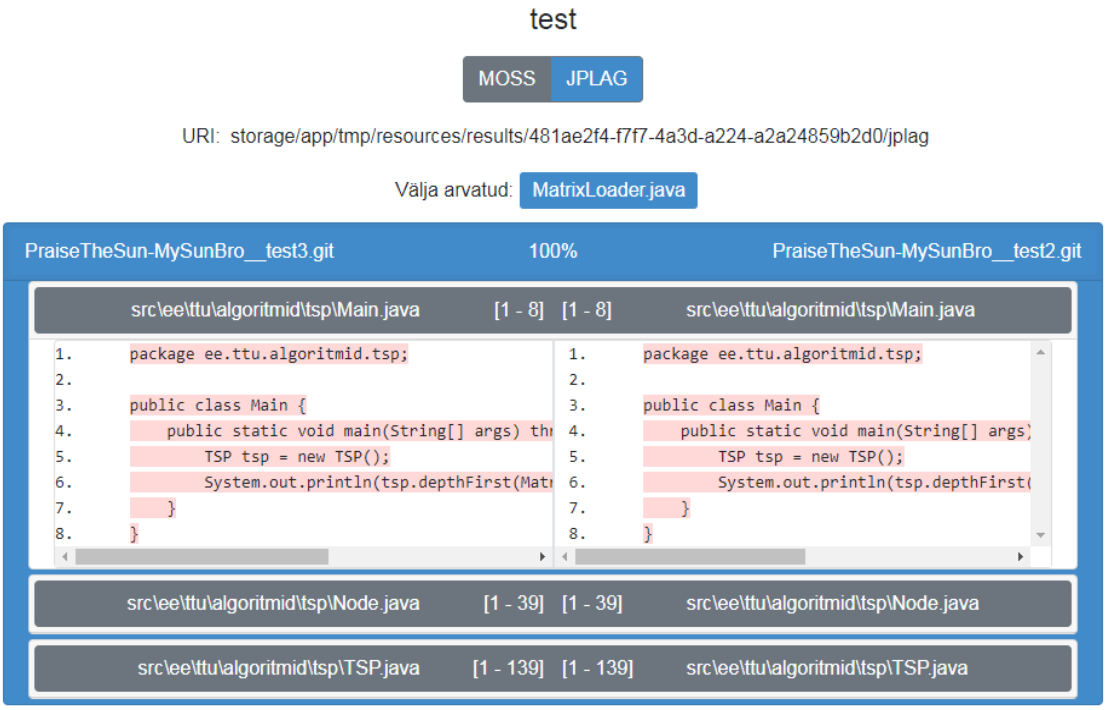
Realiseeritud sammud:

1. Esmalt otsitakse .java laiendiga failid, et neid saata plagiaadisüsteemile

2. Peale seda määratakse taimer 10 minutiga, mis annab vea taimeri lõppemise korral ning kirjutab selle vea andmebaasi

3. Lõpuks, kui tagastatud kood ei olnud võrdne 0-ga, saadetakse viga andmebaasi

JPlag näitab ka failide võrdlust, mida saab veebirakenduses vaadata.



Joonis 7. JPlag sarnasuse kontrolli kuvamine

Tulemus, mis saadetakse JPlag-ist, on ebamugaval kujul – HTML formaadis. Tulemused salvestatakse HTML failidena. Selleks, et saada täpsem tulemus sarnasuse kohta, on vaja HTML-i parsida.

4.3.2 MOSS

MOSS-i realisatsioon Java-s *(it.zielke:moji:1.0.2)* on kirjutatud juba 2014. aastal. Plagiaadisüsteem võimaldab kontrollida mitte ainult .java tüüpi faile, vaid ka teisi. See süsteem on universaalne, kuid selles projektis realiseeritakse ainult Java failide tugi MVP tõttu. Tulevikus on võimalik lisada teised programmeerimiskeeled, mida saaks kontrollida selle süsteemiga.

Tulemuse saamiseks on vaja saata HTTP päring, kus on andmed failide kohta, samuti unikaalne kasutajakood.

```
Collection<File> studentFiles = FileUtils.listFiles(
       new File(
              String.format("%s/%s", gitProperty.getClonePath(),
              checksuite.getCheck().getId()))
       .getAbsoluteFile(), new String[]{"java"}, true);
 if (studentFiles.size() == 0) {
       log.error("MOSS not found any files for plagiarism check");
       return false;
}

URL result = sendAndReturnUrl(studentFiles);
if (result == null) return false;
```

Joonis 8. MOSS-i tulemuse saamine 1

Esialgselt otsitakse failid üles (praegu ainult Java failid). Kui ei leita ühtegi faili, tekib viga. See koodiosa on näidatud Joonisel 8.

```
private URL sendAndReturnUrl(Collection<File> studentFiles) {
       SocketClient socketClient = new SocketClient();
       socketClient.setUserID(mossProperty.getKey());
       try {
              socketClient.setLanguage("java");
              socketClient.run();
              for (File file : studentFiles) {
                     socketClient.uploadFile(file);
              }

              socketClient.sendQuery();
       } catch (MossException | IOException e) {
              e.printStackTrace();
              return null;
       }

       URL result = socketClient.getResultURL();
       socketClient.close();
       return result;
}
```

Joonis 9. MOSS-i tulemuse saamine 2

Kuna kasutatakse juba MOSS-i valmis realisatsioon, siis kõik suhtlemised toimuvad serveriga mitte HTTP, vaid socket-i kaudu. Joonisel 9 on näidatud vastav kood, mille sammud on:

1. Luuakse ühendust serveriga
2. Määratakse kasutaja unikaalse kood
3. Määratakse plagiaadisüsteemi kasutatav programmeerimiskeel

4. Laaditakse üles üliõpilaste failid, mida on vaja süsteemil kontrollida

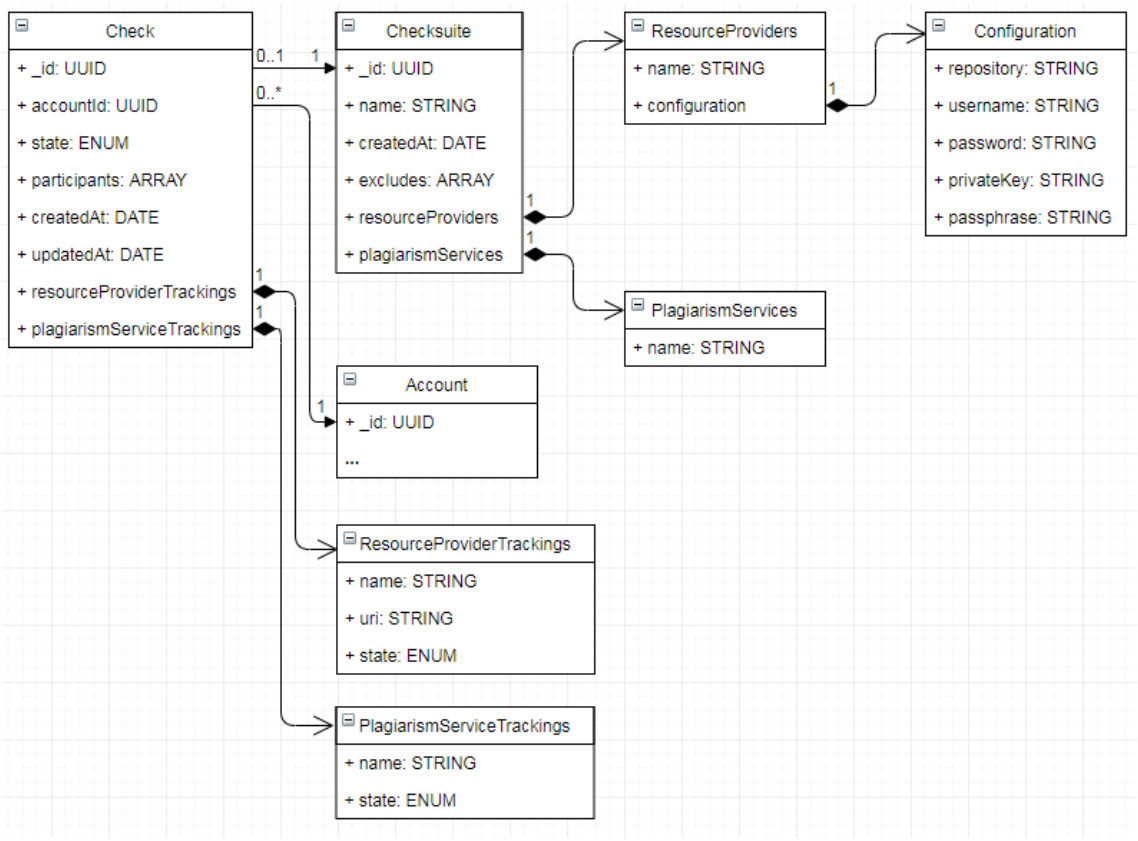
5. Serverist saadakse tulemused üliõpilaste sarnasuste kohta

MOSS-i puhul puudub võimalus vaadata failide sarnasusi veebilehel, kuna MOSS-i serverist tulevad sarnasuse andmed mitte iga failide kohta.

4.4 Andmebaas

Kuna kasutusel on NoSQL andmebaas, siis kogu loogika, mis on kirjeldatud lisas 1, oli vaja ümber kirjutada. Nüüdseks, seoseid ja kollektsioone on vähem, seega andmebaasi struktuur on lihtsam ja arusaadavam.

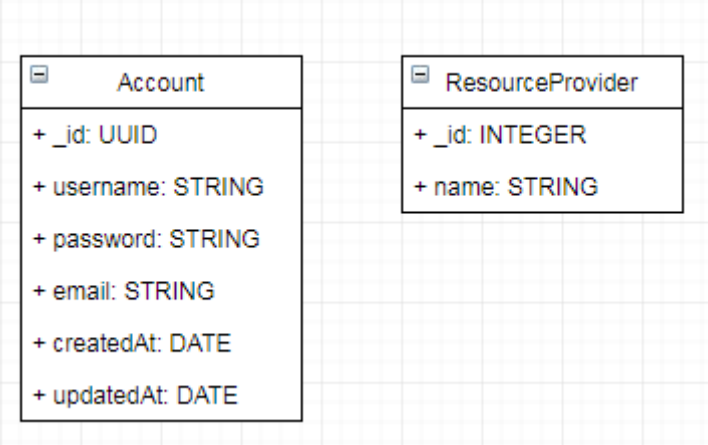
Joonis 10 näitab andmebaasiskeemi koos seostega.



Joonis 10. Andmebaasistruktuur

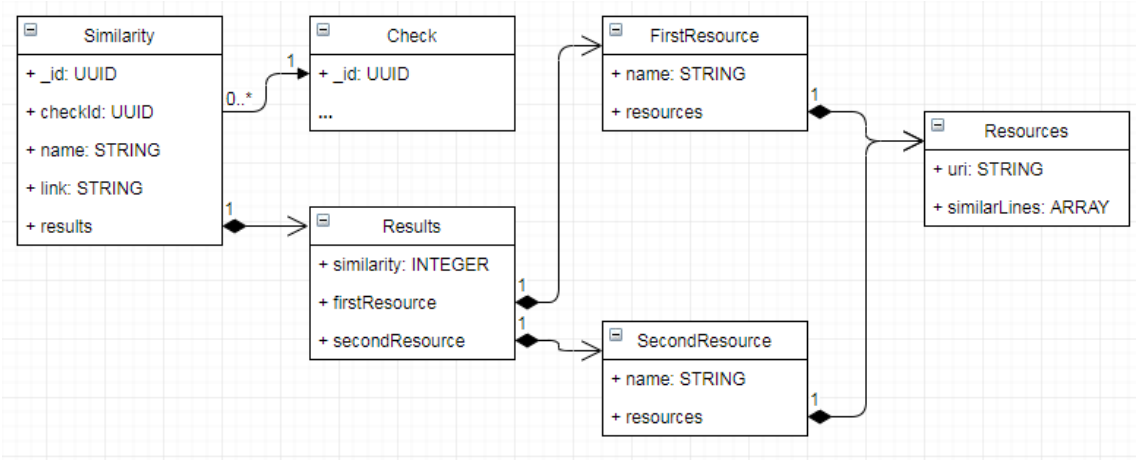
Nagu on näha joonisel, seoseid kasutatakse ainult Check - Checksuite vahel kasutades *_id* ning Check - Account vahel kasutades *accountId*. Ülejäänud põhineb kollektsioonide kompositsioonidest, mis teeb andmebaas paindlikumaks kui Julia API variant. Skeem on räsidest vabanenud, sest need ei ole enam vajalikud. Tänu sellele on

vabanetud ka privaatvõtmete salvestamisest eraldi kollektsioonis, sest nüüd nad salvestatakse ResourceProvider konfiguratsioonis.



Joonis 11. Andmebaasi klassifikaatorid

Andmebaasis on ainult 2 klassifikaatorit, need on Account ja ResourceProvider. ResourceProvider-is salvestatakse plagiaadisüsteemide nimed, mida kasutatakse antud rakenduses.



Joonis 12. Andmebaasi sarnasuse kollektsioonid

Similarity kollektsioon salvestab esimese ja teise ressursside tulemused ehk URL ja failides sarnaste ridade vahemikud, kus 0 indeks on algus, 1 on lõpp, jne (näiteks võib olla [5, 8, 13, 17], kus 5 on algus, 8 on lõpp ning 13 on teine algus ja 17 on teine lõpp)

4.5 Asünkroonsete sõnumite edastamise süsteem

RabbitMQ kasutab järjekorda ja sõnumite vahetamise tehnoloogiat, mille kaudu on võimalik suhelda programmi ja serveri vahel. Rakenduse töötamiseks oli vaja luua 3 erinevat järjekorda (Joonis 13):

- Järjekord, mis laadib andmed alla Git-i repositooriumist ehk GitClone järjekord
- Plagiaadi kontrollimise järjekord, st kontrollib, millal plagiaadisüsteemid alustavad tööd ehk PlagiarismCheck järjekord
- Järjekord lõpliku tulemuse saamiseks, st plagiaadisüsteemidest saadud tulemus ehk Similarity järjekord

Joonis 13. RabbitMQ järjekorrad

Koodi vaatepunktist see on lihtsasti realiseeritud, vt Joonis 14.

```
@Service
@RabbitListener(queues = "GitClone")
public class GitListener {
	@RabbitHandler
	private void gitClone(@Payload String json) {
		String id = gson.fromJson(json, String.class);
		...
	}

	...

	private void startPlagiarismCheck(Checksuite checksuite) {
		rabbitTemplate.convertAndSend(
		"PlagiarismCheck", gson.toJson(checksuite.getId()));
	}
}
```

Joonis 14. RabbitMQ kuulaja näide

Annotatsioonidega määratakse järjekorrad, mida kuulatakse ja suunatakse selle callback-ina. Peale seda määratakse ise callback, millesse pärinevad serialiseeritud andmed. Edasi RabbitMQ muutuja kaudu on võimalik saata päringuid edasi järjekorras.

4.6 Docker

Docker-it kasutatakse rakenduses selleks, et lihtsustada juurutamist (*deploy*) ning omavahel siduda erinevad teenused. Docker võimaldab jagada rakenduse erinevateks osadeks, mida edaspidi nimetatakse konteineriteks. Konteineris on võimalik installeerida minimalistlik operatsioonsüsteem, mis kasutab vähe ressursse.

Docker seadistamiseks on vaja kasutada kas Dockerfile või docker-compose fail, kus kirjeldatakse erinevate konteinerite seoseid ja seadeid.

```
version: '3'

services:
  mongodb:
    image: mongo:latest
    ports:
      - 27017:27017
    volumes:
      - shareStorage:/shared:ro
```

Joonis 15. Docker-compose faili näide

Docker-compose-i töölepanemiseks on vaja käivitada selline käsk:
`docker-compose up -d`, kus parameeter –d tähendab taustal (*background*) käivitumist.

5 Kasutajaliidese arendamine

Arendamise etapil kasutati Vue koos Bootstrap Vue raamistikega. Enamik elemente on loodud niinimetatud akordioni elemendi alusel. See võimaldab mitte ainult peita mittevajalikud elemendid, vaid aitab ka kiiresti vaadata sisud.

Kõik lokaalsete muudatuste või praeguse sessiooni kohta käivad andmed salvestatakse Vuex-i sõnastikku, mis võimaldab andmeid hõlpsalt hallata ning nendega töötada.

Antud sõnastik laetakse serverist lehe esmakordsel laadimisel. Kasutaja andmed salvestatakse sessiooni, kust neid edaspidi saab kasutada, mistõttu pole vaja andmebaasi ühendust pidevalt kasutada.

Antud realisatsioon on jagatud erinevateks mooduliteks. Rakenduses on üks index.js fail, kust moodulid laaditakse rakendusse ning neid saab kasutada globaalse muutuja kaudu.

5.1 Autentimine

Rakendus kasutab kolmanda osapoole teeki @Websanova/Vue-Auth, mis võimaldab tegeleda päistega ja *token*-itega ilma lisa toiminguteta.

Lisaks on võimalik kontrollida ka juurdepääsu komponentidele ehk mitte autenditud kasutaja saab ligi ainult kas sisselogimise vormile või API dokumentatsioonile. Muude lehtede jaoks peab kasutaja olema autenditud.

Seda saavutatakse kasutades metaandmed, mis on kirjeldatud ruuteri komponendis (Joonis 16).

```
{
       path: '/',
       name: 'checksuites',
       component: Checksuites,
       meta: {
           auth: true
       }
    },
    {
       path: '/auth',
       name: 'auth',
       component: Auth,
       meta: {
            auth: false
       }
    }
}
```

Joonis 16. Vue-Auth autentimise metaandmed

Sisselogimine on tehtud ühe funktsiooniga, mis võimaldab ära jätta päise kontrollimise ning mugavalt ja abstraktselt kirjutada koodi (Joonis 17).

```
onLogin () {
       this.errorAlert = false
       let app = this
       this.$auth.login({
              data: {
                     username: app.credentials.username,
                     password: app.credentials.password
              },
              success: (res) => {
                     ...
                     app.$store.dispatch('fetchUser')
              },
              error: (err) => {
                     ...
              },
              rememberMe: true
       })
}
```

Joonis 17. Vue Auth sisseloogimise funktsioon

5.2 API dokumentatsioon

Dokumentatsioon on kirjeldatud .yml failis.

Dokumentatsiooni kuvamiseks on kasutusel kolmanda osapoole tarkvara ReDoc, mis kasutab Swagger-i teeki [19].

```
export default {
  mounted () {
    let node = document.getElementById('api-doc')
    let element = document.createElement('script')
    element.src = 'https://cdn.jsdelivr.net/npm/redoc@next/' +
              'bundles/redoc.standalone.js'
    node.appendChild(element)
    node.insertAdjacentHTML('afterbegin',
       `<redoc spec-url="${axios.defaults.baseURL}/api/doc"></redoc>`)
  },
  destroyed () {
    let node = document.getElementById('api-doc')
    while (node.firstChild) {
      node.removeChild(node.firstChild)
    }
  }
}
```

Joonis 16. API dokumentatsiooni kuvamine

Kuna ReDoc-i realisatsioon Vue-s puudub, siis oli vaja see sisestada nagu sõltuvus DOM elementidesse. Seega igakord on vaja lisada ja kustutada dünaamiliselt elemendid, kui kasutaja tuleb sellele lehele või lahkub lehelt.

6 Kokkuvõte

Antud töö peamiseks eesmärgiks oli teha kaks rakendust. Üks rakendus on serveripoolne rakendus, mis oli juba kirjutatud eelnevalt PHP-s ning mille nimi on Julia API. Teine rakendus on kasutajaliides sellele serverirakendusele, mille kaudu on võimalik suhelda ja kasutada REST API ühendust.

Töö tulemusena valminud rakendused vastavad kõikidele püstitud eesmärkidele.

Serverirakendus on ümber kirjutatud Java-sse ning tehtud modulaarsemaks. Rakendus on turvalisem, kuna nüüd privaatvõtmed salvestatakse andmebaasi krüpteeritult.

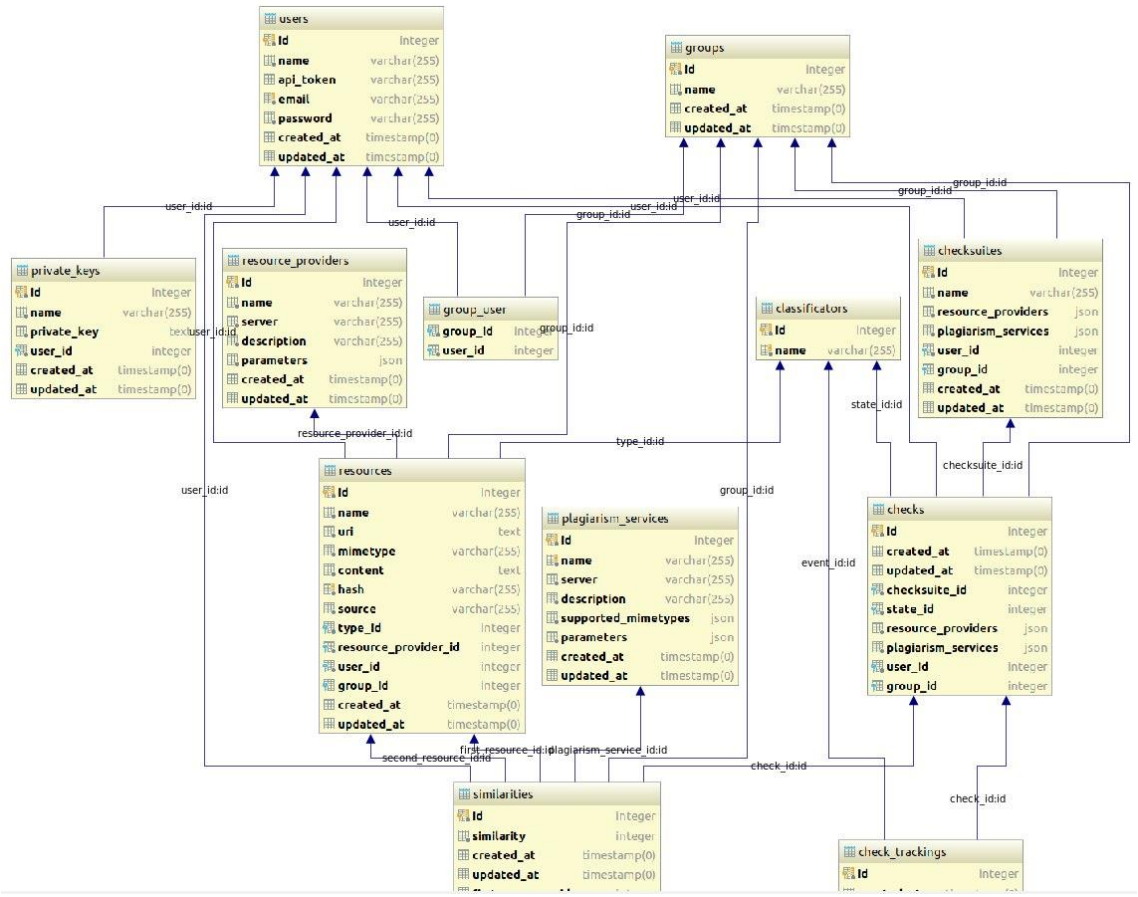
Kasutajaliides on tehtud Vue baasil, kus kasutatakse reaktiivse programmeerimise paradigmat. See tähendab seda, et andmed muutuvad reaalajas ning päringuid serverisse on vähem. Disain on iseenesest mugav ja ei vaja kasutaja õpetamist.

Kasutajaliidese suhtlemine serverirakendusega toimub REST API kaudu, mis tähendab seda, et serverirakendusega võib ühendada ka kolmanda osapoole tarkvara. Edasiarendusena realiseerida ühenduse Moodle API-iga.

Kasutatud kirjandus

[1] Luik, A. (2016). Olemasolevate plagiaaditeenuste ja andmeallikate ühendatud liidese prototüüp. Tallinna Tehnikaülikool. [Bakalaureusetöö]. Tallinn.
[2] Laravel dokumentatsioon [WWW] https://laravel.com/docs/5.6/eloquent (18.05.2018)
[3] Spring raamistik [WWW] https://spring.io/ (18.05.2018)
[4] Django raamistik [WWW] https://www.djangoproject.com/ (18.05.2018)
[5] Staatiline veebisait. Wikipedia. [WWW] https://en.wikipedia.org/wiki/Static_web_page (18.05.2018)
[6] Dünaamiline veebisait. Wikipedia. [WWW] https://en.wikipedia.org/wiki/Dynamic_web_page (18.05.2018)
[7] Neuhaus, J. (2017). Angular vs React vs Vue: A 2017 comparison [WWW] https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176 (18.05.2018)
[8] DB-engine võrdlus [WWW] https://db-engines.com/en/ranking/document+store (18.05.2018)
[9] Spring raamistik [WWW] https://projects.spring.io/spring-framework/ (19.05.2018)
[10] Spring Boot raamistik [WWW] https://projects.spring.io/spring-boot/ (19.05.2018)
[11] @Websanova/Vue-Auth [WWW] https://github.com/websanova/vue-auth (19.05.2018)
[12] Bootstrap Vue [WWW] https://bootstrap-vue.js.org (19.05.2018)
[13] MongoDB dokumentatsioon [WWW] https://docs.mongodb.com/ (19.05.2018)
[14] RabbitMQ [WWW] https://www.rabbitmq.com/ (19.05.2018)
[15] Docker [WWW] https://www.docker.com/what-docker (19.05.2018)
[16] Spring baasautentimine [WWW] https://spring.io/guides/gs/securing-web/ (20.05.2018)
[17] JWT token [WWW] https://jwt.io/ (20.05.2018)
[18] Git kloonimine info [WWW] https://git-scm.com/docs/git-clone (20.05.2018)
[19] ReDoc Swagger [WWW] https://github.com/Rebilly/ReDoc (20.05.2018)

Lisa 1 – Julia API andmebaasi seosed

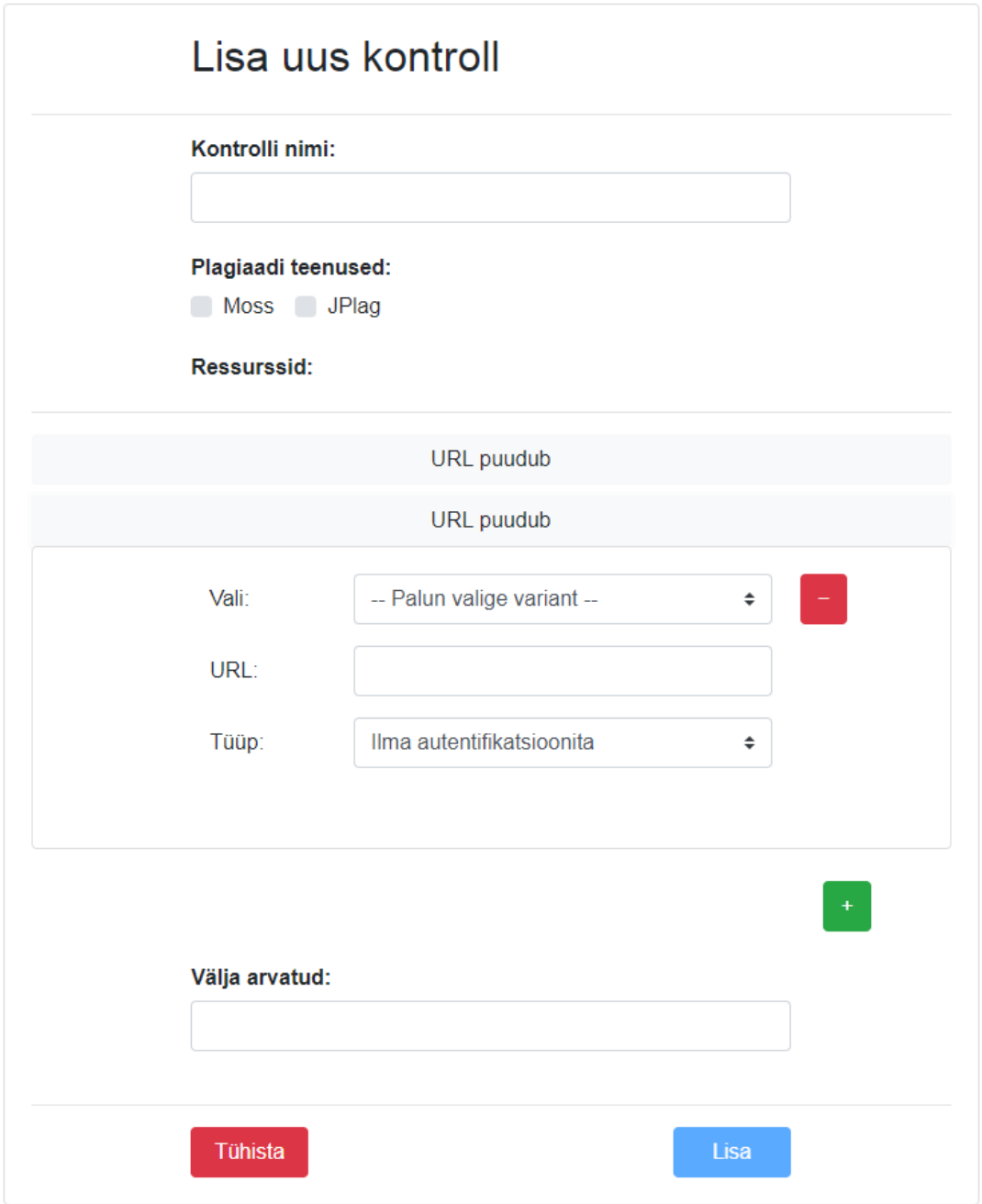


Lisa 2 – Kasutajaliides: kontrollide vaade

Kood	Nimi	Lisatud	Tegevused	
35fb2432-76ad-4ff3-9947-2ec035d833a3	Testme	07.05.2018 21:33	FAILED	DETAALID >
01e6082b-eae7-4a56-b499-e6757af9d29d	Testme	07.05.2018 21:23	ABORTED	DETAALID >
4538c522-1dab-4eaf-aab7-c9e7a8dd7f09	Testme	07.05.2018 20:29	FAILED	DETAALID >
1d6c76e8-83c7-4df4-bf8f-f1c23dee9e6f	Testme	07.05.2018 20:26	FAILED	DETAALID >
93269231-d200-48f6-99c9-62b4048814dc	Testme	07.05.2018 20:21	FAILED	DETAALID >
2ab75bb2-8772-41f9-9df7-ed5eb4338abc	sadas	04.05.2018 15:47	FINISHED	DETAALID >
19cd4394-f035-452b-baa0-06d56016d508	dsada	04.05.2018 15:17	ABORTED	DETAALID >
437ccac5-55c7-4f6e-b91a-0ac7e53b5f4d	testmee	04.05.2018 15:10	ABORTED	DETAALID >
64917982-e834-4d27-8adc-ad05ee15aee9	Testme	04.05.2018 14:53	ABORTED	DETAALID >
481ae2f4-f7f7-4a3d-a224-a2a24859b2d0	test	29.04.2018 11:07	FINISHED	DETAALID >

« ‹ 1 2 3 4 › »

Lisa 3 – Kasutajaliides: kontrolli lisamine



Lisa 4 – API dokumentatsioon