TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Maksim Usmanov 223004IAIB

Aleksandra Tremba 222966IAIB

Katarina Zemljanski 223049IAIB

Automaatse süsteemi arendus LaTeX-põhiste lõputööde keelelise ja tehnilise kvaliteedi kontrollimiseks

Bakalaureusetöö

Juhendaja: Ago Luberg

PhD

Tallinn 2025

Autorideklaratsioon

Kinnitame, et oleme 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.

Autorid: Maksim Usmanov, Aleksandra Tremba ja Katarina Zemljanski

08.06.2025

Annotatsioon

Seoses Tallinna Tehnikaülikooli IT-teaduskonna juhatuse otsusega kirjutada lõputöid kasutades LaTeX tekstikujunduskeelt, tekkis nõudlus lahendusele, mis lihtsustaks selle üleminekuprotsessi. Kuna suurem osa tudengeid pole antud kujundamiskeelt kunagi kasutanud, võib päris palju aega kuluda vormistamisele. Täiendavalt pole eestikeelsete LaTeX-is kirjutatud dokumentide jaoks õigekirja- või stiilikontrolli tarkvara, mis vastaks Tallinna Tehnikaülikooli nõuetele.

Bakalaureusetöö eesmärk on arendada süsteem, mis automaatselt kontrollib ja valideerib bakalaureusetööde faile, et tagada nende vastavus Tallinna Tehnikaülikooli vormindamisjuhendile ja stiilireeglitele. Loodud lahendus peab leidma grammatika-, süntaksi- ja stiilivigu ning aitama tudengitel lõputöid parendada ja õppejõududel keskenduda bakalaureusetööde sisule, mitte grammatika- või stiilivigadele.

Bakalaureusetöö klient on Tallinna Tehnikaülikooli õppejõud Gert Kanter, kelle vastutusalasse kuulub ülikooli lõputööde koostamise ja hindamise protsessi efektiivsuse suurendamine.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 52 leheküljel, 9 peatükki, 52 joonist, 2 tabelit.

Abstract

Development of an Automated Validation System for Evaluating the Linguistic and Technical Quality of LaTeX-based Theses

In accordance with the decision by the management of the IT Faculty at Tallinn University of Technology to mandate the use of LaTeX for writing Bachelor theses, there arose a demand for a solution that would simplify the transition from traditional word processing solutions such as Microsoft Word. As the majority of students have never utilised this typesetting language, a significant amount of time may be consumed by formatting the document. Additionally, there is a lack of software for spell-checking or syntax validation for LaTeX documents written in Estonian language that meets the requirements of Tallinn University of Technology.

The objective of this Bachelor's thesis is to develop a system that automatically checks and validates Bachelor's thesis files to ensure their compliance with the Tallinn University of Technology formatting guidelines and style rules. The developed solution should identify grammar, syntax, and style errors, and assist students in improving their final theses, while enabling supervisors to focus on the content of the theses rather than grammatical or stylistic errors.

The client for the Bachelor's thesis is Gert Kanter, a senior lecturer at Tallinn University of Technology, whose responsibilities include enhancing the efficiency of the process for compiling and evaluating final theses at the university.

The thesis is in Estonian and contains 52 pages of text, 9 chapters, 52 figures, 2 tables.

Lühendite ja mõistete sõnastik

ACID	Atomaarsus, konsistentsus, isoleeritus, püsivus (*Atomicity, Consistency, Isolation, Durability*)
API	Rakendusliides (*Application Programming Interface*)
CSS	Kaskaadilaadistik (*Cascading Style Sheets*)
DOM	Dokumendi objektide mudel (*Document Object Model*)
DTO	Andmesaateobjekt (*Data Transfer Object*)
GUI	Graafiline kasutajaliides (*Graphical User Interface*)
HTML	Hüperteksti märgistuskeel (*Hyper-Text Markup Language*)
HTTP	Hüperteksti edastuse protokoll (*Hyper-Text Transfer Protocol*)
IDE	Integreeritud programmeerimiskeskkond (*Integrated Development Environment*)
JPA	Java püsivusliides (*Java Persistence API*)
JSON	JavaScripti objektide notatsioon (*JavaScript Object Notation*)
JWT	JSON-veebimärgis (*JSON Web Token*)
LaTeX	Dokumentide ettevalmistamise süsteem [1]
Liquibase	Andmebaasiskeemi versioonihaldur
LLM	Suur keelemudel (*Large Language Model*)
MVC	Mudel-vaade-kontroller (*Model-View-Controller*)
NLTK	*Natural Language Toolkit* [2]
ORM	Objekt-relatsioonvastendus (*Object-Relational Mapping*)
PDF	Porditav dokumendivorming (*Portable Document Format*)
POST	"Saada andmeid serverisse töötlemiseks."(*"Post this data to the server."*)
Regex	Regulaaravaldis
REST	Esitusoleku siire (*Representational State Transfer*)
SPA	Üksiklehekülje rakendus (*Single-Page Application*)
SQL	Struktureeritud päringukeel (*Structured Query Language*)
UI	Kasutajaliides (*User Interface*)
URL	Ühtne ressursilokaator (*Uniform Resource Locator*)
UX	Kogemuskujundus (*User Experience*)
XML	Laiendatav märgistuskeel (*eXtensible Markup Language*)
ZIP	Andmete kadudeta pakkimise meetod ja failivorming

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva bakalaureusetöö teema on arendada süsteem, mis võimaldab kontrollida LaTeX-is kirjutatud lõputööde õigekirja ja stiili.

Hetkel puudub antud tekstikujunduskeeles kirjutatud dokumentide kontrolli võimaldav lahendus, mis vastaks Tallinna Tehnikaülikooli vormindusnõuetele. Samuti puudub lahendus, mis kontrolliks nendes eesti keele õigekirja ja kirjastiili.

Töö eesmärk on täiendada sügiseses tellimusprojektis alustatud rakendust, mida lõputöö kirjutajad ja nende juhendajad saavad kasutada oma dokumentide õigekirja, stiili ja vormindamise analüüsimiseks ja tagasiside saamiseks. Juhendajatel peab olema võimalus näha ülevaadet meeskondadest, keda nad juhendavad ja jätta kommentaare. Tudengid võivad üles laadida ZIP-konteineri oma tööst ja teenus näitab, mis testid ja millises dokumendis on läbikukkunud.

Süsteemi eesmärgiks on lihtsustada lõputööde kirjutamise protsessi ja aidata vähendada kirjalikus osas tekkivaid vigu. Üheks eelduseks on see, et tuleb kasutada ametlikku LaTeX malli, mida uuendati tellimusprojekti raames, kuna see automaatselt ennetab lihtsamaid vormindamisvigu. Samuti saavad kasutajad näha, mis muudatused olid tehtud võrreldes teiste üleslaadimistega.

Lõputöö õnnestumiseks analüüsiti olemasolevaid lahendusi, uuriti rakendamiseks kasutatavaid mustreid, teeke ja algoritme. Viidi läbi vestlusi juhendajaga ja kliendiga, et arutada, mida oleks kasulik lisada. Pandi paika rakenduse struktuur ja konsulteeriti tarkvara kvaliteedi osas õppejõududega. Aprilli 3. nädalal alustati alfatestimist ning anti väiksele rühmale rakendust testimiseks.

Rakenduse teenusekihti programmeeriti, kasutades Java Spring Boot raamistikku. Andmebaasihaldurina kasutatakse PostgreSQLi. Esituskihi arenduseks kasutati React teeki ja Tallinna Tehnikaülikooli Storybooki. Autentimiseks kasutati Microsoft Azure teenust.

Testide skript oli kirjutatud Python programmeerimiskeeles. Valminud rakenduse mugavuse ja jõudluse hindamiseks anti see kasutajatele testimiseks.

Töö järgnevates osades antakse detailsem ülevaade projekti teostamisest. Analüüsitakse olemasolevaid lahendusi, käsitletakse projektile seatud skoopi, kirjeldatakse kasutatud tehnoloogiaid ja tööprotsessi. Selgitatakse arhitektuuri- ja disainiotsuseid. Kokkuvõtteks analüüsitakse valminud veebirakenduse vastavust seatud eesmärkidele ja tuuakse välja testimise tulemused. Tuuakse välja edasiarendamise võimalusi.

2 Taust

Tarkvaraarenduse tellimusprojekti aine jaoks plaaniti luua funktsionaalne rakendus, mille eesmärgiks oli kontrollida LaTeX-is kirjutatud lõputööde vastavust TalTechi juhenditega. Meeskonna töö keskendus järgmistele tööülesannetele:

1. Täiustada TalTechi lõputöö jaoks olemasolevat LaTeX-malli, et see vastaks lõputööde koostamise juhendi reeglitele.
2. Määrata üliõpilaste sagedased vead lõputööde kirjutamisel.
3. Töötada välja lihtne rakendus, mis võtab vastu ZIP-arhiivi, milles on lõputööde failid.
4. Arendada Pythoni skript, mis analüüsib sisu ja saadab tulemuse tagarakendusse, mis seejärel edastab andmed veebirakendusele.

Projekti jaoks kasutati mitmeid erinevaid tehnoloogiaid: kontrollija rakendati Pythonis, kasutajaliides Reactis ja tagarakendus Spring Bootis ning kasutati PostgreSQLi andmebaasihaldurina. Tehnoloogia valikut selgitatakse vastavates peatükkides.

Malli struktuur:

- *main.tex* - see on peamine fail, kus dokument koostatakse.
- *thesis.sty* - see fail sisaldab lõputöö vormingu- ja stiiliseadistusi.
- *config.tex* - sisaldab lõputöö muutujaid.
- *chapters* kaust - sisaldab üksikute peatükkide faile.
- *appendices* kaust - sisaldab lõputöö lisasid.
- *misc* kaust - mitmesugused failid tiitellehtede, deklaratsioonide ja litsentsi jaoks.
- *figures* kaust - sisaldab lõputöö jooniseid.

Arvestades võimalust, et lõputööde koostamisel võivad esineda inimlikud vead, suunati ülesanne sellele poolele, et eesmärgiks sai töötada välja täiustatud lahendus, mis võimaldab tuvastada nii LaTeX-spetsiifilisi probleeme kui ka inimlikkusest tulenevaid ebakõlasid ja

kontrollida, kuidas bakalaureusetööd on lõplikult esitatud.

Autorid analüüsisid TalTechi LaTeX-i lõputöö malli ja teostasid muudatused, mille tulemusena mall kohandati viisil, mis vastab paremini TalTechi lõputööde koostamise juhendmaterjalile. Need muudatused hõlmasid teksti vormingu, kujunduse ja muu parandamist, et saavutada võimalikult täpselt juhendmaterjalile vastav dokument. Samuti automatiseeriti mitmeid tehnilisi protsesse, näiteks jooniste, tabelite ja lehekülgede loendamine, mis on nüüd dünaamiliselt uuendatavad.

Pärast edasist kaalumist koostasid autorid nimekirja valideerimistestidest, mille eesmärk on aidata õpilastel kirjutada paremaid lõputöid ja võimaldada õppejõududel tööde vastavust kriteeriumitele usaldusväärselt hinnata.

Kõigi valideerimistestide üksikasjaliku kirjelduse võib leida Lisa 5 – Valideerimistestide kirjeldused.

3 Süsteemi projekteerimine

Arvestades suureneva süsteemi keerukuse, vajaduste ja edasiarendamise võimalustega, pidid autorid ümber mõtlema süsteemi arhitektuuri. Uuendatud tarkvara arhitektuur pidi vastama järgmistele nõuetele:

1. Süsteem on paindlik ja skaleeritav.
2. Süsteemi teenused on isoleeritud.
3. Süsteem on platvormist sõltumatu.

Tarkvaraarenduse projekti raames loodud süsteem osaliselt vastas nõuetele ning autorite eesmärgiks oli analüüsida, parendada ja vajadusel ümber ehitada olemasoleva süsteemi arhitektuuri.

3.1 Süsteemi arhitektuuri muster

Paindliku ja skaleeritava süsteemi loomise esimeseks sammuks on valida sobiv muster, mille järgi arendatakse süsteemi teenuseid. Valitud muster peab toetama arendamisprotsessi ja lihtsustama süsteemi selgust huvipooltele nagu:

- Arendajad
- Administraatorid
- Kasutajad

Võttes arvesse eelnevalt välja toodud projekti nõudeid, analüüsisid autorid tarkvaraarenduses laialt kasutatud mustreid, nende tugevusi, nõrkusi ja sobivust arendatava süsteemi jaoks.

3.1.1 Monoliitne arhitektuur

Monoliitse arhitektuuri all mõeldakse süsteemi, kus üks teenus täidab kõik sellele süsteemile antud ülesanded, näiteks andmete töötlemine, valideerimine ja kasutajaliidese näitamine [3].

Monoliitse süsteemi eriomadus on kõikide komponentide tihe sõltuvus üksteisest. Sellist süsteemi on lihtsam mõista, arendada ja hooldada, sest tarkvaras tavaliselt puudub keeruline andmevahetuse süsteem erinevate osade vahel. Samuti, kuna monoliitsel süsteemil puudub komponentide vahel võrkudevahelise andmevahetuse vajadus, on süsteemi kindlustamine rünnakute vastu kergem [3]. Lisaks on tarkvara kompileerimine ja paigaldamine lihtsustatud puuduvate käigusõltuvuste pärast [4].

Siiski on monoliitsel arhitektuuril puudusi, millepärast lõputöö autorid peavad sellist mustrit ebasobivaks arendatava süsteemi jaoks. Esiteks, vajaduste suurenedes ja koodi keerukusega kasvades, muutub süsteemi parendamis- ja hooldusprotsess asjatult keerulisemaks [3], [4]. Arvestades komponentide omavahelist sõltuvust, võivad muutused ühes komponendis halvasti mõjutada teisi koodiosi. Näiteks, kui muudetakse muutuja tüüpi või nime tuleb arvestada teiste koodiosadega, kus seesama muutuja on aktiivses kasutuses. Seepärast meeskonna liikmetel puudub täielik iseseisvus enda komponendi arendamises [4].

Pidades silmas kõiki eelmainitud aspekte, ei pea lõputöö autorid monoliitset arhitektuuri sobivaks arendatava süsteemi jaoks järgmiste aspektide põhjal:

- Monoliitse arhitektuuri lihtsus ja paindlikkus kaovad, kui koodibaas suureneb.
- Koodiosade tihe sõltuvus üksteisest on vastuolus esitatud arhitektuuri nõuetega.

3.1.2 Mikroteenused

Mikroteenuste arhitektuuri all mõeldakse süsteemi, milles teenused ja nende alamdomeenid on iseseisvad ja üksteistest sõltumatud. Selles süsteemis arendatakse, paigaldatakse ja hooldatakse teenuseid eraldi, välja arvatud ühised teegid. Andmevahetus mikroteenuste vahel toimub valitud kaugpöörduse protokolli abil. Tavaliselt täidab andmevahetuse ülesannet API lüüs [5], [6].

Mikroteenustel on mitu tugevat poolt, mille põhjal sellist mustrit valitakse süsteemi arhitektuuri loomiseks. Esiteks, selles süsteemis iga teenus täidab endale mõeldud ülesannet, millepärast alamdomeenide arv on väike. See lihtsustab mikroteenuste arendamist ja arusaamist [5]. Teiseks, tänu teenuste iseseisvusele tekib võimalus paigaldada teenuste uuendusi reaalajas. Kuna vajadus uuendada kogu süsteemi ühe korraga kaob, siis on võimalik sagedamini ja kiiremini juurutada uuendusi tootmisse ning uuenduste suurus on

samuti väiksem võrreldes monoliitse süsteemiga [6]. Lisaks sellele on vajadusel võimalik alamdomeene jaotada eraldi teenusteks, mis parendab süsteemi turvalisust ja skaleeritavust [6].

Vaatamata kõigele on mikroteenuste arhitektuuril oma märgatavad nõrkused. Alustuseks on mikroteenuste puhul tähtis ettevaatlikult korraldada nende granulaarsust. Funktsionaalsuse liigsus mitmetes teenustes viib kogu süsteemi sõltuvuseni nendest teenustest, seetõttu on keerulisem saada kasu mikroteenuste arhitektuurist [6]. Täiendavalt, kuna igal teenusel on vajadusel oma andmebaas, tuleb mõnede tehingute jaoks rakendada keerulist ja järjekindlat tehingute haldamissüsteemi, mis tagab andmete kooskõlastust ja sünkroniseerimist [5]. Viimaks, mikroteenuste süsteemi üldine jõudlus on madal kordavate tehingute ja arhitektuurimustri jaotatud olemuse pärast [6].

Võttes arvesse ülalmainitud tegureid, peavad lõputöö autorid mikroteenuste arhitektuurimustrit mõeldavaks aluseks arendatava süsteemi jaoks. Arvestades eelantud nõuetega, tagab see arhitektuuri muster paindlikkust, arengu lihtsust ja süsteemi mõistetavust. Kummatigi, mikroteenuste nõuded paigaldatud riistvara jõudlusele ja andmete kooskõlastuse ja sünkroniseerimisega seotud keerukused häirivad lõputöö autoreid.

3.1.3 Kolmekihiline arhitektuur

Kihiline arhitektuur on kõige levinum arhitektuurimuster. Kihilise süsteemi omaduseks on kihtide horisontaalne järjestamine, kus iga kiht täidab oma rolli olevas süsteemis [6]. Tarkvara kihte valitakse vastavalt nõuetele ja projekti suurusele. Tavaliselt koosneb süsteem neljast kihist [6]:

1. Esituskiht (*Presentation Layer*)
2. Talitlusloogikakiht (*Business Logic Layer*)
3. Püsivuskiht (*Persistence Layer*)
4. Andmekiht (*Data Layer*)

Väiksemate süsteemide puhul talitlusloogika- ja püsivuskihi võib ühendada, ning sellist arhitektuuri nimetatakse kolmekihiliseks [6]. Selleks, et lihtsustada kihtide kirjeldust, nimetatakse talitlusloogika- ja püsivuskihi ühendust edaspidi loogikakihiks.

Kolmekihiline arhitektuur eristub teistest tänu kihtide funktsioonide selgusele.

Esituskihi moodustab kasutajaliides (GUI), mille abil suheldakse tarkvaraga [7]. Selle ülesandeks on näidata ja koguda andmeid kasutajatest ning edastada neid allolevale kihile [8].

Loogikakiht tegutseb kasutaja poolt edastatud andmete töötlemisega vastavalt kirjutatud reeglitele. Samuti on loogikakihil ligipääs andmekihile ning sellel kihil on õigused andmeid lisada, muuta ja kustutada [8].

Andmekiht säilitab süsteemi andmeid, mis on edastatud loogikakihilt. Üks tähtsaid nõudeid sellele kihile on isolatsioon teistest kihtidest, välja arvatud loogikakihist, ning andmevahetus teiste kihtide ja andmekihi vahel peab toimuma ainult loogikakihi kaudu [8], [6].

Kolmekihilise arhitektuuri tugevaks omaduseks on süsteemi arendus isolatsioonkihtide (*Layers of Isolation*) mõiste abil. See tähendab, et muutused ühes kihis ei mõjuta komponente teistes kihtides, kuna muutused on isoleeritud nende alamdomeenis [6]. Täiendavalt, kuna muutused on isoleeritud kihtide kaupa, on kihid ja nende vastavad teenused iseseisvad, ning kihi arendus-, uuendamis- ja paigaldamisprotsessid ei sõltu teistest kihtidest, välja arvatud juhtumid, kui andmesaatmisobjekte (DTO) muudetakse [6]. Lõpuks, kolmekihilise süsteemi katsetamine on kergem, kuna on võimalik arendada teiste kihide komponentide makett, et tagada arendatava funktsiooni või liidese täpsus [6].

Antud arhitektuurimustril on ka puudusi. Kolmekihilise arhitektuuri kihtide tiheda granulaarsuse pärast kannatab süsteemi skaleeritavus. Tavaliselt esitus- või loogikakihi sees arendatav teenus on monoliitse arhitektuuriga, mistõttu selle jaotus väikesteks tükkideks on problemaatiline ja kallis [6]. Lisaks tehingu täitmiseks peab iga kihti läbima, millepärast kannatab süsteemi jõudlus ja nõuded riistvarale suurenevad vastavalt tarkvara keerukusele [6].

Võttes arvesse kolmekihilise arhitektuurimustri tugevusi ja nõrkusi, peavad lõputöö autorid seda mustrit kõige rahuldavamaks arendatava süsteemi jaoks. Kolmekihiline süsteem toetab süsteemi mõistetavust, arendamisprotsessi lihtsust ja vastab eelnevalt kirjeldatud nõuetele.

3.2 Tehnoloogiavirn

Enne süsteemi arhitektuuri loomist vastavalt kolmekihilisele arhitektuurimustrile, on lõputöö autorid otsustanud analüüsida olemasoleva lahenduse kasutatud arendusinstrumente ja teeke, et teha kindlaks nende vastavus valitud arhitektuurimustrile ja arendatava süsteemi nõuetele.

3.2.1 Oleva lahenduse tehnoloogiavirna analüüs

Nagu mainiti peatükis 2, kasutab loodud lahendus järgmiseid arendusintrumente ja teeke:

- Python - vigade otsingu tarkvara
- Spring Boot - tagaliides
- PostgreSQL - andmebaas
- React - kasutajaliides

Tellimusprojekti raames valitud tehnoloogiavirn põhineb eeldusel, et süsteemi edasi arendav meeskond võib kasutada mitmekihilise või mikroteenuste arhitektuurimustrit. Tänu sellele mustrile on arendusinstrumentide ja teekide analüüs märgatavalt lihtsustatud, kuna lõputöö autorid saavad vajadusel asendada osa tehnoloogiavirnast ja ei pea ümber kirjutama tervet tarkvara.

Tehnoloogiavirna sobivuse kindlaks määramiseks analüüsitakse antud lahenduse arendusinstrumente ja teeke vastavalt valitud arhitektuurimustri igale kihile.

3.2.2 Esituskiht

Vastavalt peatükile 3.1.3, koosneb esituskiht kasutajaliidesest või UIst (*User Interface*), mille kaudu kasutajad suhtlevad teenusega.

Lõputöö autoritel on järgmised nõuded kasutajaliidese arendamiseks kasutatavale raamistikule:

1. **Lihtsus**. Valitud raamistik või teek peab toetama kiiret ja tingumatu arendust. Kasutajaliidese arendus peab olema komponendipõhine.
2. **Tüübirange keele toetus**. Andmetüüpidega seotud keerukuste vältimiseks kavatsevad lõputöö autorid kasutada TypeScript programmeerimiskeelt [9]. Raamistikul

või teegil peab olema TypeScripti esimese osapoole toetus.

3. **Lihtne integratsioon Tallinna Tehnikaülikooli Storybookiga**. Lõputöö autorid kavatsevad kasutajaliidese rakendamiseks kasutada Tallinna Tehnikaülikooli ametlikku stiiliteeki [10], mis on arendatud React teegi põhjal. Valitud raamistik või teek peab toetama ühilduvust React komponentidega.

Tellimusprojekti raames arendatud lahenduses täidab kasutajaliidese rolli veebirakendus, mis on loodud Reacti peale, kasutades TypeScript programmeerimiskeelt. Reacti alternatiivideks on Vue.js[1] ja Angular[2], mida laialdaselt kasutatakse nii kergemate kui ka keerulisemate veebirakenduste arendamiseks [11].

Lõputöö autorid peavad React teeki rahuldavaks. Järgmises peatükis rõhutatakse Reacti tugevusi ja käsitletakse teiste veebiraamistikute nõrkusi võrreldes valitud teegiga.

Kasutajaliidese arendusinstrumendi valik

React on vaba lähtekoodiga JavaScripti teek. Selle omadusteks on tugev paindlikkus, kolmandate osapoolte teekide valikuvabadus ning veebirakenduse võimestiku rakendamise vabadus, kuna React on vastutav ainult kasutajaliidese esitusvalmenduse eest [12].

Reactis arendatavad kasutajaliidesed rakendatakse, kasutades komponendipõhist programmeerimismudelit ja deklaratiivset esindusvalmendust [12]. Komponente on võimalik pesastada ning nende olekute mällu salvestamiseks ja halduseks pakub React haake (*hooks*), näiteks *useState*, *useRef* ja *useEffect* [13].

Rakendatava kasutajaliidese raames rahuldab React autorite nõudeid täies ulatuses:

1. **Lihtsus ja paindlikkus**. Reacti ainsaks ülesandeks on kasutajaliidest reaktiivselt kuvada, mistõttu teegi ülesannete ulatus väheneb ja teegi põhiomaduste mõistmine on lihtsam. Samuti võimaldab React arendajatel vabalt valida vajalikke pistikmooduleid ja teeke.
2. **TypeScript keele esimese osapoole toetus**. Lõputöö autorite eelistuseks on tüübirange programmeerimiskeele kasutus. React pakub esimese osapoole toetust sellele.
3. **Pärisintegratsioon Tallinna Tehnikaülikooli Storybookiga**. Tallinna Tehnika-

[1] `https://vuejs.org/`

[2] `https://angular.dev/`

ülikooli stiiliteek on rakendatud Reacti peale, mistõttu puudub vajadus kihttõlketarkvara otsimiseks. TalTech stiiliteegi komponenti saab kiirelt ja mugavalt kasutada rakendatavas kasutajaliideses.

Vue.js (edaspidi Vue) on kerge JavaScripti raamistik, mis pakub komponendipõhist programmeerimismudelit kasutajaliidese loomiseks, sõltumata selle keerukusest. Vue põhifunktsioonideks on deklaratiivne esitusvalmendus ja reaktiivsus [14].

Hinnates Vue raamistiku erisusi, peavad lõputöö autorid Vue raamistikku sobimatuks arendatava veebirakenduse jaoks järgmistel põhjustel:

- **Liigne reaktiivsuse paindlikkus**. Vue reaktiivsus võimaldab siduda komponente ja nende seisundeid, kuid rakenduse kasvades muutub ka komponentide ja nendevaheliste seoste haldamine üha keerukamaks. Arendajad peavad alati meeles pidama komponentidevahelisi sidumisi, et vältida jõudlus- ja hooldusraskusi [15], [16].
- **Liigne keerukus React teegi integreerimisega**. Selleks, et integreerida Vue põhjal arendatav veebirakendus Reactiga, on vaja luua sild (*bridge*) nende komponentide vahel. Selleks on võimalik kasutada vuera[3] või Veaury[4] teeki. Nende kasutamiseks on vajalik React teegi paigaldamine ning need pakuvad funktsioone React komponentide reaalajas muutmiseks, mis suurendab asjatult arenduskeerukust.

Angular on veebiraamistik, mis pakub laia arendusinstrumentide, APIde ja teekide komplekti, et lihtsustada ja muuta arendusprotsessi sujuvamaks [17]. Võrreldes teiste raamistikega, saab Angulari kasutada ilma teekide lisamise vajaduseta.

Kõigest hoolimata on selle raamistiku nõrkused märgatavad ja võivad segada kiiret ja kestvat arendust, mistõttu peavad lõputöö autorid Angulari mitterahuldavaks valikuks:

- **Piiratud paindlikkus**. Angular on projekteeritud täpset arendusmeetodit silmas pidades ning piirab arendajate vabadust [18].
- **Suur rakenduse ressursitarve**. Angular on kõikehõlmav veebiraamistik, mistõttu on sellel võrreldes teiste raamistikega suuremad nõuded seadme ressurssidele [18].
- **Reacti ja Angulari töövoo ja arhitektuuri erinevus**. Arvestades eelmainitud

[3]vuera, `https://github.com/akxcv/vuera`

[4]Veaury, `https://github.com/gloriasoft/veaury`

nõrkusi, on Reacti ja Angulari töövoo vahe silmapaistev, mistõttu võivad tulevikus tekkida raskused veebirakenduse uuendamise ja hooldusega.

3.2.3 Loogikakiht

Loogikakiht ühendab talitlusloogika- ja püsivuskihti ning selles kihis rakendatavat tarkvara nimetatakse tagaliideseks. Peatükis 3.1.3 on mainitud, et loogikakihi põhiomadusteks on andmete töötlemine määratud reeglite järgi ning suhtlus andmekihiga.

Sobivat tagarakenduse raamistikku valides lähtusid lõputöö autorid järgmistest nõuetest:

1. Raamistik on lihtne ja paindlik.
2. Raamistik on skaleeritav.
3. Raamistikku arendatakse aktiivselt ja toetatakse kolmandate osapoolte arendajaid.
4. Raamistik on rakendatud Java keeles.
5. Raamistik toetab kiiret ja lihtsat *OAuth* integratsiooni.

Loogikakiht on jaotatud kaheks osaks: tagarakendus ja kontrollija. Olemasolevas tarkvaras on kontrollija loodud Pythoni programmeerimiskeeles. Nagu mainiti peatükis 2, sisaldab kontrollija palju valideerimisreegleid ning selle keerukuse tõttu ei asenda autorid loodud kontrollija tarkvara ja jätkavad arendust sama virna kasutades.

Tellimusprojekti raames arendatud tagarakendus on loodud Spring Boot raamistikku ja Apache Maven tarkvara projekti haldamisinstrumenti kasutades [19]. Lõputöö autorid peavad raamistikku rahuldavaks valikuks ning järgnevalt tuuakse välja Spring Booti tugevused võrreldes teiste raamistikega nagu Quarkus [5] ja Micronaut [6].

Spring Boot on tagarakenduse arendamiseks mõeldud raamistik, mis põhineb Spring Java raamistikul. Spring Booti tugevused toetavad autorite valikut järgmistel põhjustel [20]:

- **Esimese ja kolmandate osapoolte aktiivne toetus**. Spring Boot on arenenud ja aktiivselt toetatud raamistik, mis pakub algusest peale laia valikut esimese ja kolmandate osapoolte arendusinstrumente ja pistikprogramme.
- **Lihtsus**. Antud raamistik on väga hästi dokumenteeritud ning algkonfigureerimiseks

[5] Quarkus, `https://quarkus.io/`
[6] Micronaut, `https://micronaut.io/`

ja käivitamiseks ei ole vaja paljude pistikprogrammide ja arendusinstrumentide teadmisi.

- **Skaleeritavus**. Spring on projekteeritud mikroteenuste arhitektuuri silmas pidades.
- **Turvalisus**. Spring Boot on ühilduv Spring Security raamistikuga, mis on *de-facto* standard Spring tarkvara kindlustamiseks. See toetab JWT (*JSON Web Token*) loomist, valideerimist ja OAuth põhist autentimist.

Quarkus raamistikul on oma tugevused, nagu ehitusoptimeerimine ja reaktiivne tuum [21], kuid selle nõrkused varjutavad raamistiku tugevusi:

- **Quarkus on projekteeritud Kubernetes silma pidades**. See raamistik eeldab tagarakenduse arendamisel Kubernetese kasutamist, mistõttu on selle optimeerimine suunatud seda tarkvara kasutavatele projektidele. Lõputöö autorid ei kasuta Kubernetest [21].
- **Arenev kolmanda osapoole toetus**. Võrreldes Spring Bootiga puudub Quarkus raamistikul samaväärselt aktiivne arendajate toetus, mistõttu on selle pistikmoodulite ja arendusinstrumentide valik piiratud.

Micronaut raamistik on tugev alternatiiv Spring Booti asemel. See pakub ettekompileerimist käivituse optimeerimiseks ja toetab mitmesuguseid kindlustamismehhanisme. Siiski lõputöö autorid peavad Micronaut raamistikku mitterahuldavaks järgmistel põhjustel:

- **Micronaut on liiga uus**. Raamistiku värskuse tõttu puudub sellel Springiga võrreldav kolmandate osapoolte tugi ja uurimiseks on saadaval vähe ressursse. Lisaks on Spring Boot arendajate seas tuntum, mistõttu eelistavad autorid valida tuntuma raamistiku, et vältida võimalikke keerukusi toe leidmisel.

3.2.4 Andmekiht

Andmekiht koosneb andmestruktuurist, mis salvestab, töötleb ja edastab andmeid vastavast domeenist. Andmebaasi päringute täiendamiseks kasutatakse andmebaasihaldussüsteemi, nagu PostgreSQL, MySQL ja SQLite.

Lõputöö autorid analüüsisid ja valisid sobiva andmebaasihaldussüsteemi, lähtudes järgmistest nõuetest:

1. Andmebaasihaldur on skaleeritav.
2. Andmebaasihaldur on vaba lähtekoodiga tarkvara.
3. Andmebaasihaldur pakub turbevahendeid pääsumehhanismi seadistamiseks.
4. Andmebaasihaldur toetab osaliselt või täies ulatuses ISO/ANSI SQL standardit.

Tellimusprojekti raames rakendatud tarkvaras kasutati andmebaasihaldurina PostgreSQLi, mida lõputöö autorid peavad sobivaks valikuks. Järgnevalt põhjendatakse PostgreSQLi valikut ja käsitletakse ka teisi relatsioonadmebaasi mudeliga andmebaasihaldussüsteeme.

Andmebaasihalduse süsteemi valik

PostgreSQL on vaba lähtekoodiga objekt-relatsiooniline andmebaasihaldussüsteem. See andmebaasihaldur on tuntud oma töökindluse, andmete tervikluse, laiendatavuse ja jõudluse poolest [22].

PostgreSQL vastab kõigile autorite seatud nõuetele andmebaasihalduritele:

- **Skaleeritavus**. PostgreSQL toetab konkurrentsust ning suure koormusega säilitab see andmebaasihaldur ndmete usaldusväärsuse tänu selle ACID (*Atomicity, Consistency, Isolation, Durability*) vastavusele [22], [23].
- **Vaba lähtekoodiga tarkvara**. See andmebaasihaldur on vaba lähtekoodiga tarkvara, millel on tugev kogukonna toetus [22].
- **Turbevahendid**. PostgreSQLis on rakendatud pääsu reguleerimise süsteem ning tava- ja multiautentimissüsteem [22].
- **SQL standardi toetus**. Antud andmebaasihaldur vastab SQL standardile ning evitab vähemalt 170 erisust 177-st [22], [24].

Võrreldes PostgreSQLiga, ei vasta MySQL autorite nõuetele täies ulatuses [23], [24]:

- **SQL standardi toetus**. MySQL järgib SQL standardit osaliselt ning sama päring PostgreSQL ja MySQL andmebaasihaldurites võib anda erinevaid tulemusi.
- **Skaleeritavus**. MySQL ei toeta konkurrentsust, mistõttu puudub võimalus asünkroonselt päringuid täita.

Samuti on SQLite puhul märgatavaid puudusi, mis võivad häirida andmekihi haldamist tulevikus:

- **Skaleeritavus**. SQLite salvestab terve andmebaasi ühte kõvaketta faili, mistõttu andmebaasi suurus on piiratud. Lisaks piirab see päringute lugemis- ja kirjutamiskiirust [25].
- **Turvalisus**. Sellel andmebaasihalduril puudub esimese osapoole pääsumehhanism [25, 24].

3.2.5 Kihtide isolatsioon

Kolmekihilise arhitektuurimustri valimisel süsteemi rakendamiseks kavandasid autorid kihid ja nendega seotud teenused üksteisest isoleerituna, kasutades teemade lahususe põhimõtet. Selle põhimõtte kohaselt jagatakse kihid erinevateks isoleeritud domeenideks ning nende vahel rakendatakse "sildändmevahetuse, päringute ja tehingute teostamiseks [26]. Kihtide isoleerimise parandab teenuste sõltumatust, turvalisust ning lihtsustab paigaldust ja hooldust.

Pärast erinevate arendusinstrumentide analüüsi, käsitlevad lõputöö autorid Docker tarkvara kasutamist rakendatavate teenuste seadistamiseks, paigaldamiseks ja hoolduseks. Docker on avatud platvorm rakenduste arenduseks ja käivitamiseks isoleeritud keskkonnas, mida nimetatakse konteineriks. Dockeri tugevad küljed on järgmised [27]:

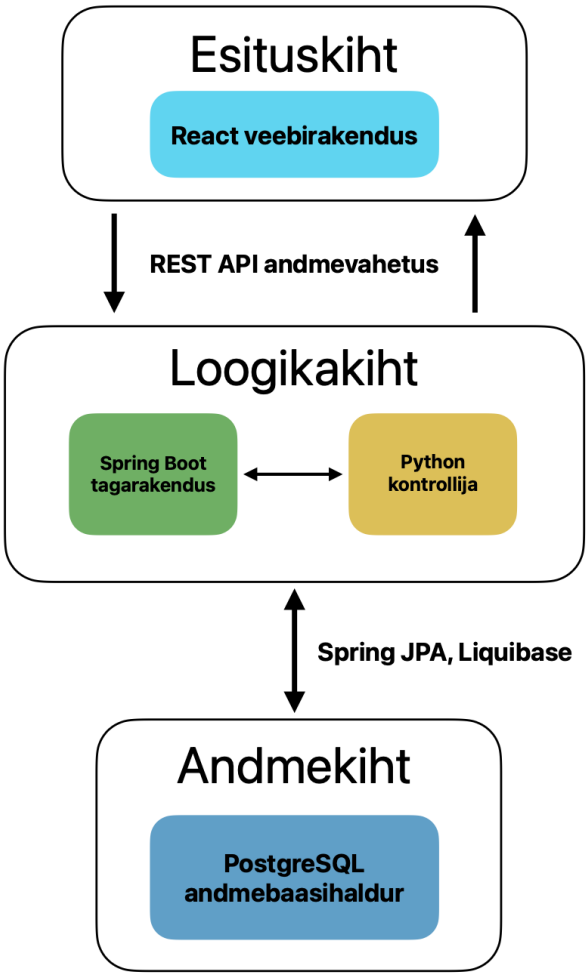
1. Docker on kiire ja kergekaaluline.
2. Docker konteinerid on vaikimisi isoleeritud ja sõltumatud.
3. Docker võimaldab luua konteinerivahelist andmesidevõrku või andmehoidlat.
4. Docker on platvormist sõltumatu.

Hinnates Dockeri platvormi ja selle tugevusi, leiavad autorid, et see sobib kihtide isoleerimiseks ja teenuste jagamiseks alamdomeenidesse. Järgmises peatükis kirjeldatakse, kuidas teemade lahususe põhimõtet rakendatakse kihide isoleerimiseks.

3.3 Docker-põhine süsteemi projekteerimine

Selles peatükis kirjeldatakse projekteeritava süsteemi Docker-põhist arhitektuuri, kasutades jooniseid ja nende selgitusi. Lõputöö autorid lähtuvad peatüki alguses esitatud nõuetest ning arvestavad projekteeritava süsteemi võimaliku ressursikasutuse ja jõudluse suurenemisega.

Kolmekihilise arhitektuurimustri ja teemade lahususe põhimõtte rakendamisel jagasid lõputöö autorid kihtide vastavad teenused eraldi konteineritesse. Konteinerite näitlikustamist on näha Joonisel 1.



Joonis 1. Docker-põhine kihtide teenuste konteinerdus.

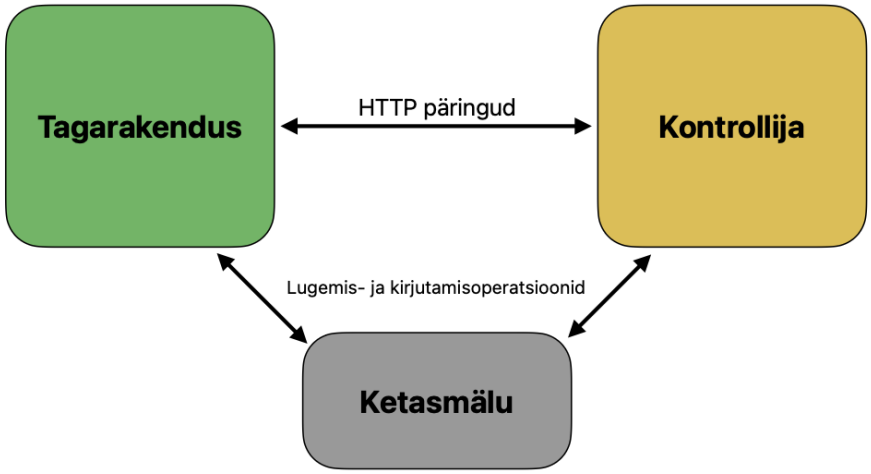
Antud jooniselt on näha, et korraliku süsteemi projekteerimiseks peavad autorid sobivaks järgmist teenuste jagamist konteineriteks:

- Veebirakenduse konteiner
- Tagarakenduse konteiner
- Kontrollija konteiner
- Andmebaasihalduri konteiner

Spring Booti tagarakenduse ja Pythoni kontrollija jagamine põhineb autorite soovil tagarakenduse ja kontrollija sõltumatuks arenduseks. Lisaks see parandab ja kiirendab

teenuste uuendamis-, haldus- ja kindlustamisprotsessi.

Siiski, kuna tagarakenduse ja kontrollija vahel saadetakse bakalaurusetööde faile, mille maht võib ületada mitu megabaiti, peavad lõputöö autorid võrguvahelist andmevahetust aeglaseks ja mitterahuldavaks. Selle probleemi lahendamiseks kasutatakse failide salvestamiseks ketasmälu ning tagarakenduse ja kontrollija vahel HTTP-päringutesse lisatakse failide teed. Lahenduse visualiseering on näha Joonisel 2.



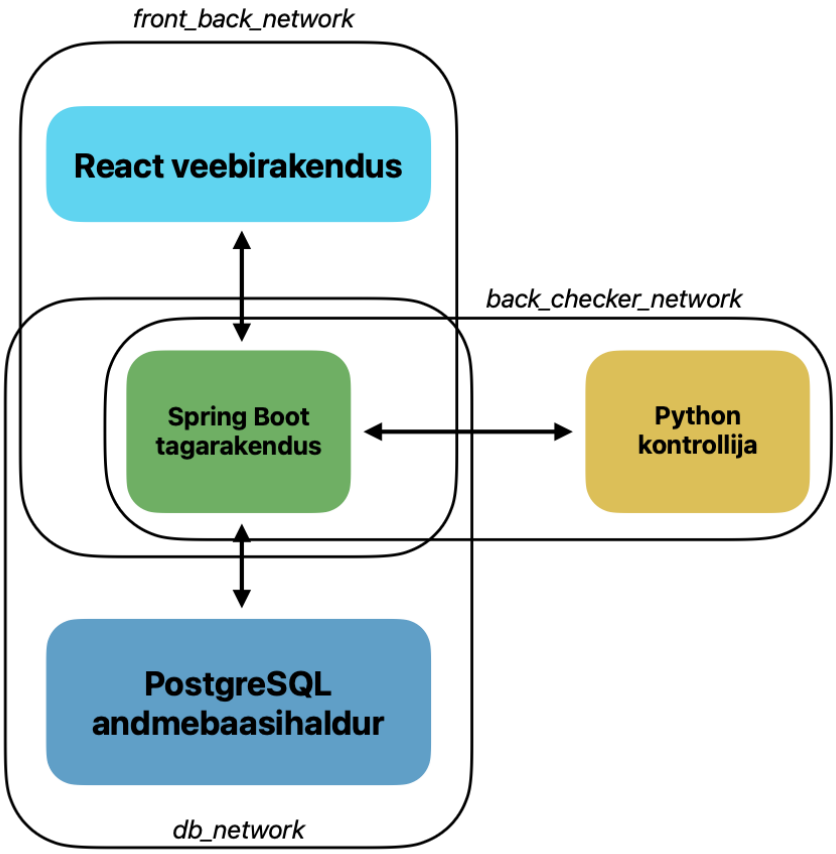
Joonis 2. Tagarakenduse ja kontrollija konteinerite vahepealne struktuur.

Samuti arvestavad lõputöö autorid võimalike sisend-väljundoperatsioonide keerukustega. Näiteks, kuna tagarakendus ja kontrollija on isoleeritud eraldi konteineriteks, siis on nende failisüsteemid ka eraldatud [27]. See tähendab, et bakalaurusetööde failiteed on suhtelised konteinerite failisüsteemide raames. Selle lahendamiseks kasutatakse Dockeri vahendit konteineri failisüsteemi sidumiseks väliste ketasmäludega [27] ning päringutes on failitee konteineriga seotud välismälu suhtes.

Turvariskide ja küberintsidentide vältimiseks kasutavad autorid Dockeri võrgustikuvahendit konteinerite pääsuvõrkude piiramiseks [27]. Selleks jagatakse konteinerid kolmeks võrguks:

- Veebirakendus - tagarakendus võrk
- Tagarakendus - kontrollija võrk
- Tagarakendus - andmebaasihaldur võrk

Käesolev jaotus parandab turvalisust ning toetab teenuste korralikku eraldatust. Konteinerite võrgustiku näitlikustamist on näha Joonisel 3.



Joonis 3. Docker konteinerite võrgustik.

Docker konteinerite loomiseks ja seadistamiseks kasutatakse Dockeri vahendit Docker Compose. See võimaldab käivitada ja seadistada konteinereid YAML (*YAML Ain't Markup Language*) andmejadasuse keele abil [27]. Näidis tagarakenduse Docker Compose faili sisu on näha Lisa 9 all [28].

4 Kontrollija

4.1 LaTeX-i töötamine pragmaatiliselt

Esimene oluline kaalutlus enne TeX-failide kontrollimiseks mõeldud Python-skripti väljatöötamise alustamist oli määratleda parim viis TeX-failidega pragmaatiliselt töötamiseks. Esialgne lähenemine seisnes regulaaravaldiste (*Regex*) kasutamises TeX-failidest otse teksti väljavõtmiseks ja käsuviidete töötlemiseks, kuid see ostutus piiravaks. Selle asemel oli mõistlik leida teek, mis võimaldaks LaTeX-i käskudega töötamist ilma ebavajalike vaheprotsessideta.

Analüüsides olemasolevaid lahedusi, ilmnes, et Pythonis on vähe vahendeid LaTeX-iga töötamiseks. See asjaolu sai oluliseks ka projekti hilisemates aspektides, kus tekkisid ootamatud vead, mille põhjuseks olid kasutatavate tööriistade vananenud seisund või lõpetamata arendus.

Antud kontekstis uuriti kahte tööriista - pylatexencis sisaldav moodul latexwalker ja TexSoup [29], [30]. Latexwalker on moodul, mis parsib LaTeX-i sisu struktureeritud kujul esitatavaks sõlmepuuks, mis sisaldab iga dokumendikomponendi infot (tekst, käsud, keskkonnad jne) [29]. See on hea dokumendi pragmaatiliseks analüüsiks, kuid sõlmepuudega töötamine oleks antud olukorras algse ülesande liiga keerukaks muutmine. LaTeX-i elementidele kasutamiseks peaks kontrollija kood ligipääsema puu sõlmedele, selle asemel, et elemendiga ise töötada. Seetõttu otsustati kasutada TexSoup-i LaTeX-i. TexSoup on pythonis loodud teek, mis võimaldab LaTeX dokumentidega lihtsustatud viisil manipuleerida, kasutades käsusilte, mis vastavad käskudele TeX-failides [30]. Võrreldes varasemate regulaaravaldistega põhinevate lahendustega, pakub TexSoup paremini struktureeritud ja hooldatavamat koodibaasi.

Esimene praktiline etapp LaTeX dokumendiga töötamiseks oli pääseda ligi failidele, millele on viidatud *main.tex*-is. Selleks loodi funktsioon, mis võimaldab peatükkide sisu parsida failist *chapters_main.tex*, mis sisaldab kõiki lõputöö peatükke. Töötades erinevate TeX-

failidega, millele põhifailidest viidati, ilmnes probleem failiviidete ebakõlaga, kui LaTeX aktsepteerib viiteid ilma TeX-laienditeta. Selle ebakõla vältimiseks lisati kontroll, mis valideerib enne rakenduse käivitamist faililaiendeid.

4.1.1 Probleemid TexSoupiga

LaTeX-iga pragmaatiliselt töötades esines mitmesuguseid probleeme. TexSoup osutus väga tundlikuks vigade suhtes, mis ei olnud autorite otsese kontrolli all. Sisestatud käsud ei kompileerunud õigesti, jättes välja mõned sulgemismärgid ning tõlgendades neid valesti. Pärast TexSoupis olevate probleemide uurimist märgati, et mainitakse alamkeskkondade ja valesti sobivate sulgude probleeme, mis olid päris sarnased nende vigadega, mida Python-skript sai TexSoup-i failidega tööle [30]. Enamik nimetatud probleemidest olid seotud LaTeX-i struktuursete omadustega. Siiski suurendades LaTeX-i veakindlusi väärtusele 1, õnnestus enamik osa neist lahendada. See tähendab, et TeX-failide analüüsimine püüab taastuda teatud vigadest, näiteks sulgemata keskkondadest, võimaldades protsessi jätkata [31]. Tekkis ka probleem % märkide eemaldamisega failidest, kuna see on LaTeX-is ja TexSoup ei näinud erinevust % märgi ja % kommentaari käsu vahel. Tarkvaraarenduse projekti raames kasutati täiendava lahendusena regulaaravaldistel põhineval eeltöötlust, et tuvastada ja kõrvaldada potentsiaalsed vastuolud enne TeX-faili analüüsimist TexSoupi abil. Antud lahendus oli tõhus ning see säilitati ilma muudatusteta.

4.2 Arendus

Kontrollija arendamist jagati eri teemade valideerimiskatsete jaoks mitmeks osaks. Järgnevas kirjelduses räägitakse kõige olulisematest arendusosadest.

4.2.1 Struktuuri testimine

Arvestades, et Tallinna Tehnikaülikoolis on kinnitatud LaTeX-is koostatud bakalaureusetööde mall, otsustasid autorid kasutada eelnevalt loodud malli, et tagada kirjalike lõputööde järjepidevust ja lihtsust arendusprotsessis. Vastasel juhul nõuaks lahendus erinevate juhtude ja stsenaariumide LaTeX-i teostuse käsitlemist. Struktuuritesti eesmärk oli kontrollida, kas projekti failistruktuuris esinevad kõik nõutud komponendid, sealhulgas kaustad ja failid, näiteks *main.tex*, *thesis.sty* ja *.bib* fail.

Õpilaste töö puhul on oluline arvestada inimliku vigade faktorit ning teha kontrollid võimalikult veakindlaks. Seetõttu veenduti käesolevas etapis, et TeX-failidega töötamisel oleksid kontrollid tõstutundlikud, eriti arvestades, et Python on vaikimisi juhtumitundlik, tuli teha juhtumitundlikkuse tagamiseks kasutusele mitmeid muudatusi. Kõige otstarbemaks lahenduseks osutus kõigi failinimede teisendamine väiketähtedeks, kasutades selleks Pythonis sisseehitatud *.lower()* [1] meetodit.

Struktuurikontrolli realiseerimiseks töötati välja skript, mis käsitleb juurteed *pathlib*-iga ning kontrollib, kas vastava nimetusega kaustad ja failid eksisteerivad projektis.

Pärast esimese valideerimistesti loomist otsustasid autorid, et parim viis kontrollide käivitamiseks vajalikus järjekorras on luua testikäivitaja. Testikäivitaja realiseeriti sellel arenguhetkel lihtsa eraldiseisva Python-failina, kuhu olid kõik vajalikud testid imporditud ning see käivitas testid järjestikku. Kõigepealt sooritatakse struktuuritest. Selle edukal läbimisel käivitatakse kõik järgnevad kontrollid. Kui struktuuritest ei vasta nõutele, peatatakse edasine testimine.

4.2.2 Teksti testimine

Kuigi TexSoup võimaldas pragmaatiliselt töötada LaTeX-i käskudega, ei võimaldanud see eemaldada tekstikontrolli jaoks mittevajalikke käske. Kõige lihtsam lahendus sellele probleemile oleks käskude ja keskkondade eemaldamine regulaaravaldiste abil. Autorid rakendasid seda lahendust ajutise meetmena selle probleemi lahendamiseks ja otsustasid hiljem kasutada LaTeX-i teksti parserit, mida käsitletakse põhjalikumalt järgmistes peatükkides. Regexi kasutamine osutus aga ebausaldusväärseks, kuna see oli vähese jõudlusega, tulemuseks oli liigne keerukus ja lahendus ei olnud kindel. Lisaks ei arvestanud see kõiki LaTeX käske ega võimaldanud analüüsida näiteks tabelite ja jooniste pealkirju, erinevaid keskkondi ning tabelite sisu. Sellest hoolimata tugineti sellele lahendusele tarkvaraarenduse projekti käigus. Probleemi põhjalikum lahendus on esitatud peatükis 4.4 LaTeX-i teisendamine tekstiks.

[1] `https://docs.python.org/3/library/stdtypes.html#bytearray.lower`

4.2.3 Viidete kontrollimine

Viitedokumentidega töötamiseks valiti BibtexParser - Pythonis kirjutatud teek, mis võimaldab töötada *.bib*-failidega. Selle abil oli võimalik parsida reference.bib faili ja kõik sisestusandmed kätte saada. Iga kirje on tsiteeritav objekt, millel on väljad, mis sisaldavad konkreetseid andmeid viite kohta. Selle teegi abil oli võimalik hõlpsasti väljade võtmed ja väärtused välja võtta ning kontrollida, kas neis esineb vigu [32]. Näiteks hinnatakse erinevate testimismeetodite abil viitamise vormistuse vastavust kehtestatud standarditele.

4.2.4 Õigekirja kontrollimine

Sõnade õigekirja kontrollimiseks otsustasid autorid kasutada Pythonis loodud teeki, mis analüüsib tekstis üksikuid sõnu ja võrdleb neid keelekorrektsete sõnade sõnastikuga, pakkudes vajadusel kõige tõenäolisema sõna ettepaneku. Probleemiks osutus sobiva teegi leidmine, mis kontrolliks lisaks inglise keelele ka eesti keelt. Teek, mis töötab selle eesmärgi saavutamiseks, on Phunspell, Pythoni õigekirjakontrollija, mis kasutab spylls port Hunspelli [33].

Hunspell on laialdaselt kasutatav õigekirjakontrolli programm, mis viitab sarnastele sõnadele ekslike sõnade puhul. Selle eeliste hulka kuuluvad *Unicode*[2] tähemärkide kodeerimine ja keeruline morfoloogiline analüüs ning mis kõige tähtsam - see programm on tõlgitud mitmesse keelde, sealhulgas eesti keelde [34].

Arenduse käigus tekkis teine probleem: Phunspell tuvastas mitmeid tehnilisi termineid ekslikult vigadena, kuna need ei esinenud vaikimisi sõnastikus. Selle probleemi lahendamiseks kasutati faili, mis sisaldas kogu töös kasutatud termineid ja lühendeid ning need lisati ignoreeritud sõnade loendisse. Selle abil vähenes valepositiivsete tulemuste arv.

Kuna õigekirjakontrollprogramm kontrollis iga sõna eraldi, võttis selle käivitamine üsna palju aega. Protsessi optimeerimiseks viidi läbi samme, mille eesmärk oli vältida ebavajalikke tööetappe ja ühendada sarnased meetodid üheks. Vaatamata optimeerimisele, võttis suurte tekstide kontrollimine iga üksiku sõna kaupa aega. Riistvaraliste piirangute ja madala täpsuse tõttu, otsustasid autorid hiljem rakendada suure keele mudeli (LLM) põhist kontrollmeetodit lausete jaoks.

[2] `https://home.unicode.org/`

4.3 LLM kontroll

Suurte keelemudelite (LLM) kontrolli väljatöötamise raames valiti kolm suurt keelemudelite tootjat - Mistral, OpenAI ja Google DeepMind [35], [36], [37]. Nende valik põhineb mudelite kiirusel, hallutsinatsioonide hinnangul ja laial kasutusel maailmas. Kontrolliks kasutatakse järgmiseid mudeleid:

- Mistral - *mistral-large*
- OpenAI - *gpt-4o*
- Google DeepMind - *gemini-2.0-flash*

Alguses valiti Mistral ja OpenAI tootjad nende kiire reageerimisaja, turvalisuse ja madala hinna tõttu. *mistral-large* sisend maksab 1.8€/M tokenit, ning väljund 5.4€/M tokenit[3], kuid *gpt-4o* palub sisendi eest 2.50$/M tokenit ja väljundi eest 10$/M tokenit [4].

Lisaks kuna Mistral on Euroopa tootja, eeldati, et see pakub paremat Euroopa keelte töötlemist. Siiski edasise testimise käigus selgus, et Mistral on väga piiratud eesti keele töötlemiseks. Tootja mudelid sageli tagastasid kasutu väljundit või ei pidanud viipede juhenditest kinni. Tulenevalt kasutati OpenAI mudelit eesti keelsete päringute saatmiseks, kuna selle faktilise järjepidevuse määr on tunduvalt suurem võrreldes Mistral mudeliga: 98.5% (*gpt-4o*) vs 95.9% (*mistral-large*) [38].

Uurides teisi suurte keelemudelite tootjaid, autorid lõimisid eelmainitud Google DeepMind Gemini mudelit. See on tunduvalt odavam võrreldes teiste mudelitega: sisend maksab 0.15$/M tookenit, ning väljund 0.60$/M tookenit [5]. Täiendavalt, *gemini-2.0-flash* faktilise järjepidevuse, jõudlususe ja eesti päringute töötlemise skoor on väga mõjuv: 99.3% (*factual consistency rate*), 0.7% (*hallucination rate*) ja lõputöö kirjutamise hetkeseisuga (04.06.2025) tehisaru baromeetris selle mudeli skoor on 1501, ning sellele on määratud esimene koht tabelis [38], [39].

Keelemudelite integreerimist tehti tagarakenduses kasutades Spring AI pistikmoodulit [40]. See võimaldab mugavalt seadistada ühendust LLM tootjate APIga, ning saata päringuid. Töötlemisprotsess on järgmine:

[3] `https://mistral.ai/pricing`
[4] `https://platform.openai.com/docs/pricing`
[5] `https://cloud.google.com/vertex-ai/generative-ai/pricing`

1. Tagarakendus saab kätte *submissionId* ja *promptId*, mille abil otsitakse selle esitusega seotud dokumenti ja valitud viipa.
2. PDFist loetakse kogu teksti Apache PDFBox raamistiku abil [41].
3. Päringut luuakse ühendates loetud teksti ja andmebaasis salvestatud viiba. Selle kood on kirjeldatud osas Lisa 26 – Tagarakenduses viiba ehitusfunktsioon.
4. Päringut saadetakse tootja APIle ja töödeldatakse saadud vastuse.
5. Vastuse tagastatakse kasutajaliidesele.

4.3.1 LaTeX-i käskude kontrollimine

Selleks, et töötada käskudega ja kontrollida elemente, nagu tabelite ja jooniste tühjad pealkirjad, kasutati TexSoupi. Kasutades selle teegi meetodeid vajalike käskude leidmiseks, oli see protsess lihtne. Vaatamata sellele ei võimaldanud eelnevalt kirjeldatud probleemid mõnel juhul TexSoupiga töötada, mistõttu kasutasime Regexit vajamineva käsu otsimiseks TeX-failist.

4.3.2 Kompileerimise testimine

Enne TeX-failide kontrollimist oli vaja veenduda, kas kompileerimisel esineb LaTeX-i vigu. Selleks kasutati pdflatexi. Antud tööriist võimaldab TeX-failist LaTeX-i kompileerida, mida hiljem kasutatakse ka PDF-väljundi kompileerimiseks LaTeX testikontrollijas [42].

Kompileerimistesti koodilõigu võib leida Lisa 6 – Kompileerimistesti kood.

4.4 LaTeX-i teisendamine tekstiks

TeX-failide teisendamine lihttekstiks Regexi abil ei olnud pikemas perspektiivis optimaalne lahendus. See tekitas probleeme, kuna kõiki LaTeX-i käske oli võimatu meeles pidada ning keskkonna sisu vormindamine muutus nii keerukaks, et lahendus ei olnud enam otstarbekas. Pärast LaTeX-i parsereid otsides selgus veel kord, et olemasolevad LaTeX-iga töötamiseks mõeldud teegid on sageli ebatäielikud, vananenud või lihtsalt ei sobi selle projekti lahendusele. Kõige sobivamateks osutusid latex2text ja pyDetex [43], [44]. Mõlemal lahendusel on omad nõrkused ja pärast mõlema teegi LaTeX-i parsimisel testimist pidid autorid valima sobivaima lahenduse uuringu tulemuste põhjal, mis on esitatud Tabelis 1.

Tabel 1. Latex2text ja pyDetexi võrdlemine.

latex2text	**pyDetex**
Joonised ja tabelid on valesti vormindatud, isegi kui pealkirjad on välja võetud, ei pruugi ebavajalik vormindus kontrollidega toimida.	Joonised ja tabelid ei ole üldse vormindatud.
Loendid vormindatakse iga üksuse ette tärniga (*).	Loetelud on mugavalt vormistatud, kuid ei koosne lausetest.
Peatükkidele eelnevad §§§ märgid.	Peatükkid on vormindatud tavalise tekstina.
Ei vorminda erimärke.	Erimärgid on vormindatud, mõned LaTeX-i käskude erimärgid ei ole vormindatud.
Mõned käsud jätavad enda järele topelttühikud.	Paljud käsud on hästi vormistatud, kuid mõned ei ole lõplikult vormistatud.

Kuigi latex2text arvestab enamiku käskude ja keskkondadega, ei sobinud selle vormindus kontrollija jaoks, kuna eesmärk on vältida võimalikke vigu lihttekstina. Latex2text seadis esikohale lugejate jaoks hästi välja nägeva teksti tootmise, kuid see vormindus läks testidega vastuollu, tekitades palju valepositiivseid tulemusi. Näiteks ei jagatud lauseid korralikult, vaid ainult osaliselt. Vastupidiselt ei olnud pyDetex veel täielikult välja arendatud. Tuli langetada valik olemasoleva teegi autorite nõuetele vastavaks kohandamise ja pooliku teegi täiendamise vahel. See ei olnud lihtne, kuid autorid otsustasid jätkata pyDetexi arendamist, kuna see võimaldas kujundada uusi vormindusfunktsioone, mis sobiksid kokku juba olemasolevate testikontrollidega.

4.4.1 PyDetex teek

Enne pyDetexi teegi muutmise alustamist tekkis küsimus korrektsest vormindamisest. Tekstikontrollija vajas selgelt konstrueeritud lauseid, et kontrollida programmiliselt vigu, mida õpilased võiksid tekstis teha. Selle tulemuseks oli parsitud teksti märgistamine selle LaTeX keskkonnatüübiga, millest see pärineb. Näiteks ei saanud jooniste pealkirju täielikult eemaldada, sest kontrollija peaks valideerima, kas need on õigesti vormistatud. Seepärast tehti muudatus, et joonise teksti ette pandi informatiivne tekst, milles märgitakse, et tegemist on joonise või tabeli pealkirjaga, mitte lõputöö põhiteksti osaga.

PyDetex ise kasutab teksti analüüsimiseks stringiga manipuleerimist ja pyDetexi repositooriumist oli tehtud hargmik (*fork*) [6], et lisada tugi täiendavatele elementidele, lisades sellele kaheharulise loomismeetodi. Saadud hargnemist ei esitatud ühinemispäringule, kuna muudatused kirjutati spetsiaalselt projekti kontrollija jaoks. Lisatud kood suutis analüüsida *longtable* keskkonda, vormindades iga lahtri eraldi lausena lausepõhiseks kontrolliks. Teine probleem oli see, et pyDetex oskas analüüsida nimekirju, kuid nagu ka tabelite puhul, tekitasid lausepõhised tekstikontrollid valepositiivseid tulemusi, kui iga elementi ei analüüsitud eraldi lausena. Seega seda silmas pidades muudeti olemasolevat koodi elementide ja loendatud nimekirjade jaoks.

Keskkonnad, mis ei sisaldanud tekstilist sisu, nagu koodi- ja matemaatikakeskkonnad, loeti teksti kontrollimiseks ebavajalikeks ja seetõttu jäeti töötlemise käigus kõrvale, asendades need kohatäitjatega. Samuti viidi läbi järgmised muudatused:

- Jooniste ja tabelite pealkirjad on teksti analüüsimiseks korrektselt eraldatud.
- *Section** käsk, mis tähistab jaotisi, mis ei tohiks ilmuda sisukorras, parsitakse nüüd samamoodi nagu tavaline *section* käsk.
- Mitmerealised loendid ja tabelikeskkonnad parsitakse kontrollidega ühiselt.
- PyDetexi käskude loetelu eemaldamiseks laiendati ka erinevate käskudega, näiteks *pagebreak* või *noindent*.
- Teoreemi, korollari, lemma, märkuse ja tõestuse keskkondi parsitakse lausetena.
- Ristviited tsiteerivad viiteid korrektselt.
- Hüperlingid parsivad URLi.
- Erimärgid loetakse õigesti.

Analüüsiks parsiti ainult lõputöö teksti sisaldavaid TeX-faile, näiteks lõputöös kasutatud peatükkide ja kaustast *misc* faile.

4.5 Lause jagamine

Järgmine probleem, mis tekkis teksti kontrollimise etapis puudutas lausepiiride määramist. Nagu varasemalt peatükis 4.4 mainiti, kasutati lausete eraldamiseks regulaaravaldisi, kuid see ei saanud hakkama lause jagamisega sellistel juhtudel, kui lause algas eesti keele

[6] https://github.com/AleksandraTremba/PyDetex

tähtedega, punktidega numbrite järel ja punktidega lühenditega. Lahendusena kasutasime NLTK teeki, mis võimaldab teksti töödelda ja lauseteks jagada. NLTK lause *tokenizer* lahendas lause õigesti jagamise probleemi ja PunktTokenizer on eelnevalt treenitud inglise keele jaoks, mis võimaldab kasutajal jagada teksti lauseteks, kasutades mittejärelevalvega algoritmi, et luua mudel lühisõnade, kollokatsioonide ja lauset alustavate sõnade jaoks [45]. Siiski oli sellega mõningaid probleeme eesti keele punktiga lühendite tuvastamisel. Punkt on loodud parameetrite, näiteks lühendite loendi, õppimiseks, mis võimaldab autoritel ja tulevastel arendajatel seda loetelu laiendada ja teha selgeid vahesid lauselõppude ja muude nüansside vahel [45].

4.6 Tulemuste käsitlemine

Selleks et kuvada kontrollide tulemusi korrektselt ja ühtselt, struktureeriti väljund JSON-formaadis objektina, mis sisaldab "testide"massiivi. Iga test on esitatud objektina, millel on väljad "id", "nimi", "kirjeldus"ja "vea tekst". Kasutatakse JSON formaati.

```
{"tests": [{
    "id": "test_id",
    "title": "test_title",
    "description": "description",
    "error_text": "error_text"
    }, ...]
}
```

Iga esitamise kohta kirjutatakse täielikult uus *check_results.json* fail ning salvestatakse tulemuse kausta, seejärel Python kontrollija serverisse ja lõpuks saadetakse vastus tagarakendusse.

4.7 Valideerimistestide registreerimine

Järgmine suur samm oli läbi mõelda, kuidas oleks võimalik kontrolle hõlpsalt sisse ja välja lülitada ning uusi teste mugavalt lisada, et võtta arvesse võimalikke muudatusi lõputööde nõuetes tulevikus. Selle ülesande teine keskne eesmärk oli Pythonis kirjutatud kontrollide ühendamine andmebaasiga ja nende sinna automaatne salvestamine.

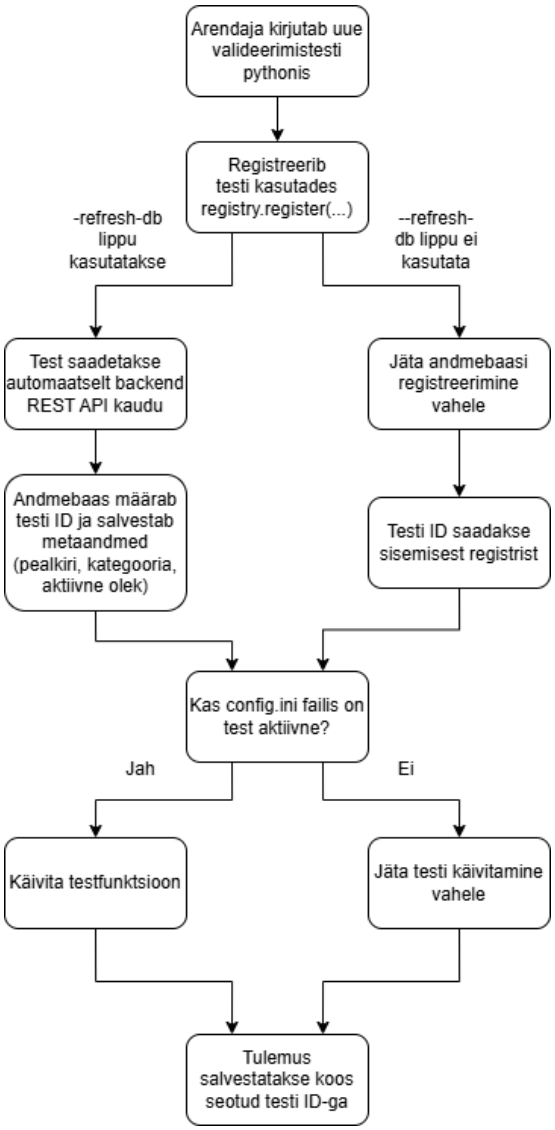
Sel põhjusel lisati testiregister, mille eesmärk oli jälgida Pythonis kirjutatud valideerimisteste, salvestada need andmebaasi ja omistada kontrolli käigus avastatud vigadele vastavad andmebaasipõhised identifikaatorid. Peamine eesmärk oli saavutada maksimaalne

automatiseerimine uute testide lisamisel. Enne registri kasutuselevõttu oleks protsess koosnenud palju rohkematest sammudest. Arendaja oleks pidanud kirjutama kontrollkoodi, seejärel lisama andmebaasi läbi tagarakenduse, lisama vigade pealkirjad, kirjeldused ja kategooriad nii kontrollijas kui ka andmebaasis ning jälgima testide identifikaatorid käsitsi. Selline lahendus muudaks testide sünkroniseerimist kergemaks.

Testiregistri värskendamisel registreeritakse kontrollid automaatselt andmebaasis oleva tagarakenduse kaudu, mis võimaldab kontrollijal ja tagarakendusel suhelda. Iga kord, kui on vaja uus valideerimistest andmebaasi registreerida, saab kasutada käsku *–refresh-db*. Järjepidevuse huvides on olemas funktsioon, mis käivitab selle käsu kontrollijaga ajaintervallide järel, et tagada andmebaasi ajakohasust.

Lisaks võimaldati testide aktiveerimine või deaktiveerimine konfiguratsioonfaili *config.ini* kaudu. Registris käivitatakse kõik registreeritud testid, kui nende konfiguratsioonivõtme väärtus on *True*. See väärtus muudab ka kontrolli aktiivsuse staatust ja salvestatakse andmebaasi käsu *–refresh-db* vahendusel.

Kogu testide registreerimisprotsess võib näha Joonisel 4.



Joonis 4. Valideerimistesti registreerimise protsess.

4.8 PDF faili kompilatsioon

PDFi genereerimiseks kasutati pdflatexi töörista [42]. Alguses ei eristatud LaTeX-i kompileerimistesti ja PDFi genereerimist. PDF koostati juhul, kui see test oli edukas. LaTeX-i pdf-genereerimine on aga aeganõudev, kuna see hõlmab mitmeid järjestikuseid etappe [46]:

- Käivitada pdflatex TeX-failide koostamiseks.
- Käivitada biber, et vormindada viiteid *.bib* failist.
- Käivitada pdflatex, et lahendada ristviiteid, sisukorda jne.
- Käivitada pdflatex, et lisada väljundisse tsitaatide sildid.

See pikk kompileerimiskontrolli lõpuleviimise protsess osutus probleemseks, sest ilma eduka kompileerimiskontrollita ei saanud teisi kontrolle käivitada. Selle protsessi optimeerimiseks rakendati eraldi kompilaator ja lõimelisuse abil käivitati paralleelselt teiste lõputööde kontrollimisega, kui kompilatsioonikontroll näitas, et LaTeX kompileerub ilma vigadeta. See ei pikendanud oluliselt kontrollija käitusaega.

4.9 Valideerimistestide tulemuste esiletõstmine

Pärast kontrollide tulemuse saamist oleks kasulik PDF-failis vead esile tõsta, et hõlbustada nende leidmist. Esialgselt pidi esiletõstmine toimuma kontrollija rakenduses, kuna see tegeles PDF genereerimisega.

Mõte oli selles, et testiti, kas esiletõstmise protsessi lisamine Pythoni kontrollijasse on mõistlik. Idee oli kasutada *highlight* käsku lateksis. Protsessi käigus kopeeriti kogu kaust koos lõputöö TeX-failidega ja mähiti veatekst highlight käsuga, mille tulemuseks oli kontrolleri tulemuste esiletõstmine genereeritud PDFis. Erinevad LaTeX-käsud, mis võisid TeX-failides esineda, segasid aga protsessi, kuna neid oli võimatu esile tõsta. Käskude eemaldamine LaTeX failidest oleks ebaoptimaalne lahendus, mis muudaks kogu protsessi keerulisemaks.

Selle tulemusel otsustas meeskond kasutada PDF-töötlusteeki, et saada veateksti piiritletud kastid ja saata need korraliku esiletõstmise jaoks frontendi. Selleks proovisid autorid Javas Apache pdfboxi ja Pythonis PyMuPDFi [41], [47]. Mõlemad olid lehtedelt teksti eraldamisel head, kuid otsisid igalt lehelt eraldi, muutes erinevate lehtede vahel tekstivahede esiletõstmise võimatuks. See muutis lahenduse liiga keeruliseks.

Uuringu tulemused näitasid selgelt, et selle ülesande täitmine Pythoni ja Java tehnoloogia abil ei ole mõistlik. Seetõttu viidi see funktsionaalsus veebirakendusele.

4.10 LaTeX *diff*

Oluline osa projekti süsteemist oli esitatud lõputööde erinevate versioonide võrdlemine ja nende erinevuste väljastamine. Seda funktsionaalsust saab kasutada muudatuste jälgimiseks ja sama töö eri versioonide paranemise nägemiseks. Parema visualiseerimise eesmärgil otsustati tuua muudatused esile lõputöö eelmiste ja praeguste versioonide PDF-failides.

Selle kontrolleri osa puhul kaaluti esmaseks lahenduseks kasutada GitLabi kehtestamisi, et võrrelda muudatusi pygit2 diff mooduli abil [48]. Täiendavate testide käigus jõuti aga järeldusele, et nimetatud lähenemine ei sobi repositooriumi LaTeX failide võrdlemiseks. Peamine argument oli see, et TeX-failides esinevad LaTeX-i käsud takistaksid teksti esiletõstmist lõplikus versioonis.

Järgmiselt keskenduti sobivama võrdlusmeetodi leidmisele, mis võimaldaks usaldusväärselt tuvastada tekstimuudatusi lõputöö eri versioonide vahel. Autorid otsustasid selleks kasutada difflib teeki, kuna see on kasulik moodul järjestuste võrdlemiseks [49]. Pärast PyDetexi edukat integreerimist TeX-failidest teksti analüüsimiseks, salvestas kontrollija iga esituse lihtteksti eri kausta, luues teksti võrdlemiseks mugava aluse. Difflibi klass *Differ* võrdles lõputööde teksti, parsis lihtteksti ja tekitas inimloetavad erinevused [49]. Väljundread, mis sisaldasid eelmisele tööle ainuomaseid ridu, algasid koodiga ‚- ' ja töö praegusele versioonile ainuomaseid ridu koodiga ‚+ ' [49]. See eristamine võimaldas kontrollijal võtta need read ja võrrelda neid, eraldades muudetud teksti ja salvestades selle JSONi sisse, mis saadetakse tagarakendusse edasiseks visualiseerimiseks.

4.11 Optimeerimine ja parandamine

Pärast skripti põhifunktsioonide väljatöötamist keskenduti peamiselt selle optimeerimisele ja parandamisele. Tulemuste kuvamiseks otsustasid autorid lisada iga testi tulemuse kirjeldusse vea asukoha. Nii, et see märkis, millises peatükis oli viga, või seoses tekstitestiga ka märkis, millises lauses.

Arendusprotsessi käigus ilmnes, et kuna testi koodi kirjutasid erinevad arendajad, juhtus nii, et kord korduvalt loeti faili, et koguda iga kord vajalikku teavet. Protsessi optimeerimiseks viidi kõigi lõputööde failidele ligipääsu loogika testikäivitajasse. See toimis nii - failid loeti käivitajas ja seejärel saatis see igale testile ainult selle faili sisu, mida see vajas argumendina, mis muutis protsessi palju optimeeritumaks.

Kuna ülikooli malli uuendati jooksvalt, pidi kontrollija kohanduma tehtud muudatustega. Nüüd oli üliõpilastel lihtsam panna lõputöö keel ja oli lihtsam teha keelekohaseid teste. Näiteks peaks asesõnade kontroll valideerima eri keeltes erinevaid asesõnu, iga keelel on oma õigekirjakontrollid.

Lihtsama hooldatavuse huvides võeti aega koodi puhastamiseks ja dokumenteerimiseks. Koodi puhastamise sammud seisnesid selles, et vaadata üle konstandid ja panna need ühte kohta, et neid saaks vajadusel hõlpsasti muuta. Iga klassi meetodi jaoks oli kommentaarid (*docstring*) kirjutatud ja vajadusel lisati kommentaarid. Skripti sammude jälgimiseks rakendati logimine. Igal esitamisel on oma logifail ja see salvestatakse esitamiskausta.

5 Tagarakenduse arendus

Antud peatükk räägib täpsemalt tagarakenduse struktuurist ja selle ülesannetest. Samuti põhjendatakse süsteemi arendamisel tehtud valikuid. Tagarakendus oli võrreldes tellimusprojekti aine raames valminuga peaaegu täielikult ümber ehitatud. Muudatused tulid projekti keerukuse kasvust ja suurema andmebaasi nõudmisest. Tagarakendus töötab kui abivahend kasutajaliidese, andmebaasi ja kontrollija omavaheliseks suhtlemiseks. Selle struktuur jälgib levinud kolmekihilist mustrit: kontroller, repositoorium, teenus. Andmebaasi tabelite esitamiseks on loodud olemiklassid. Hibernate ORMi kasutati indekseerimiseks ja otsinguks. Samuti on kasutatud Lombokit, mis struktureerib ja abstraheerib Java klasside põhifunktsioone. Nii on lihtsam hoida koodi puhtana, eraldatuna ja arusaadavana. Andmebaasiga suhtlemiseks oli kasutatud JPA ja Liquibase [50], [51]. See toetab andmebaasi püsivust, ning lihtsustab päringute ja tehingute teostamist. Tagarakenduse tähtsamatele funktsioonidele testide kirjutamiseks kasutati Mockito Core sõltuvust [52].

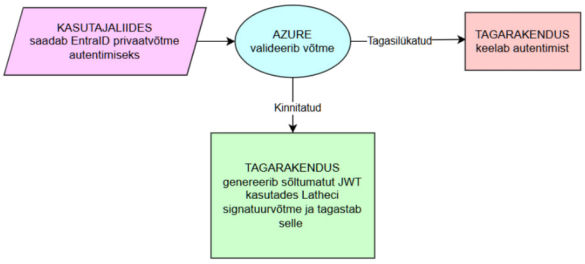
5.1 Kontrollerikiht

Kokku on arendatavas tagarakenduses 10 kontrollerit ja 64 lõpp-punkti (*endpoint*). Keerukama funktsionaalsusega kontrollerid on järgmised: Kommentaarikontrolleris on kaks lõpp-punkti, mis võimaldab kommentaare andmebaasi salvestada ning neid sealt kätte saada. See oli mõeldud tudengite ja juhendajate omavaheliseks suhtlemiseks. Kasutajakontrolleris on kolm lõpp-punkti. Selle klassi peamisteks ülesanneteks on kasutajate autentimine/registreerimine ning profiiliinfo kättesaamine. Autentimisest räägitakse järgmises alapeatükis. Suure keelemudeli kontrolleris on neli lõpp-punkti. Esimesed kaks on eesti- või ingliskeelsete päringute saatmiseks Mistral mudelile. Viimased kaks on OpenAI pärimiseks. Esitamiste kontroller on arendatavas tagarakenduses väga tähtis, kuna see vastutab tööde üleslaadimiste üle arhiivina või GitLabi lingina. Selles on kuus lõpp-punkti, mille ülesanneteks on arhiivide kättesaamine, andmebaasist info lugemine ja PDF-failide saatmine. Meeskonnakontrolleris on 10 lõpp-punkti, mis vastutavad liikmete registreerimise, kutsete saatmise ja andmete kättesaamise eest. Testikontroller on mõeldud kontrollija

kirjutatud testide regulaarseks salvestamiseks andmebaasi. Muud lihtsamad kontrollerid, nagu õppesuunakontroller, staatuskontroller, sildikontroller (*Tag Controller*), teavituste kontroller on mõeldud andmebaasist info kättesaamiseks või abiklassidena kasutajaliidesele signaalide/sõnumite saatmiseks. Kõik lõpp-punktid olid testitud Postmani kaudu päringute tegemisega [53].

5.2 Turvalisus

Antud projekti turvalisuse konfiguratsioon on üles seadud, kasutades Azure Active Directory pistikprogrammi ja JWT-t. Tokeni saamise protsess on kujutatud Joonisel 5.



Joonis 5. Autentimise käigus tookeni saamise protsess.

Saadud tokenit kasutatakse kõikides ülejäänud päringutes. See sisaldab järgmist informatsiooni: kasutaja rollid, aegumiskuupäev, e-posti aadress ehk identifikaator ja metaandmed. JWT kasutamiseks oli lisatud Spring Security ja Azure kaudu autentimiseks oli Azure AD sõltuvus [54], [55].

5.3 Teenusekiht

Teenusekihis toimub kogu loogika selleks, et kontrollerid saaksid keskenduda HTTP päringute tegemisele. Antud tagarakenduses on kokku 10 teenuseklassi. Suurem osa neist sisaldab andmete muutmist DTO-ks (Data Transfer Object), et kontrollerid saaksid kätte andmeid kompaktsemal kujul ja ainult vajaliku infoga. Iga olemiklassi jaoks on tehtud vähemalt üks DTO klass. Andmebaasist andmete saamise protsess ja nende salvestamine esineb peaagu igas klassis ning toimub analoogselt - JPA repositooriumide kaudu. Käesolevas peatükis räägitakse täpsemalt vaid teenustest, kus toimuvad süsteemi tähtsaimad protsessid. Kasutajateenus suhtleb JWT tokeni genereerijaga. Samuti vastutab see osaliselt teavituste eest, mida kasutaja saab, ja võimaldab neid vaigistada. Dokumenditeenuses toimub arhiivi

töötlemise protsess. Selle peamised meetodid on:

- *saveDocument()* ja selle abimeetodid - salvestab kasutaja üles laetud .zip-faili serverisse, võtab välja selle sisu, määrab sellele unikaalse failitee ja salvestab lahtipakitud arhiivi andmebaasi tabelisse nimega "*document*".
- *sanitizeEntryName()* - puhastab ZIP-konteineris oleva faili või kausta nime, et vältida turvariski nimega *ZIP Slip* ehk protsessi, mille käigus üritatakse väljuda sihtkataloogist.
- *runPythonScript()* ja selle abimeetodid - võtab ühendust kontrollija Pythoni serveriga ja saadab sellele salvestatud dokumendi failiteed.
- *getResponse(), processTests()* - saavad serverilt JSON-kujul vastuse ja töötlevad testitulemusi. Läbikukkunud testid salvestatakse andmebaasi tabelisse nimega *"failed_test"*.
- *getDifference()* ja selle abimeetodid - saadab kontrollijale päringu kahe dokumendi failiteedega, et saada vastuseks infot nende erinevusest.

See teenuseklass töötab asünkroonselt. Kontrollijaga suhtlemise koodi jooksutatakse paralleelselt muu funktsionaalsusega. See on mõeldud selleks, et oleks võimalik faili üles laadida, saata üleslaadimise identifikaator kohe kasutajaliidesele ja siis kontrollija tulemusi oodata. Esitusteenuses toimub tööde üleslaadimiste loogika. See käitleb esituse staatust, läbikukkunud testide ja PDF-i saatmist ja muid infopäringuid. See toimub tihedas koostöös dokumenditeenusega. Samuti on seal realiseeritud tööde üleslaadimine GitLabi [1] lingi kaudu, kasutades GitLab API-d [56]. Tiimiteenuses toimub kogu loogika, mis võimaldab luua meeskonda, kutsuda sinna inimesi ning liituda mõne muu meeskonnaga. Sellest saab väljuda ja seda saab kustutada. Testiteenus võimaldab teste otsida ja neid andmebaasi salvestada koos kaasa antud kategooriaga. See sisaldab meetodit, mis võtab teste kontrollijast ja uuendab testide tabelit iga 24 tunni tagant. See on vajalik selleks, et teste saaks lihtsamini juurde lisada. Suure keelemudeli (LLM) teenus on vajalik Mistral või OpenAI mudelitega suhtlemiseks. See loeb kindlalt leheküljelt teksti PDF-st ja saadab kas inglise või eesti keeles kirjutatud päringud. Seejärel saab see neid JSON-kujul kätte.

[1] `gitlab.cs.ttu.ee`

5.4 Kontrollija server

Selleks, et tagarakendus saaks suhelda kontrollijaga, oli loodud Python HTTP server. Põhiülesandeid täidab *CheckHandler*, mis on *BaseHTTPRequestHandler*ist päritud klass. Selle ülesanneteks on võtta vastu POST päringuid ja käivitada järgmiseid skripte:

- *latex_diff.py* - võrdleb erinevaid failiversioone
- *test_runner.py* - käivitab teste saadud arhiivi peal

Skripte käivitatakse vastavalt päringu struktuurile: näiteks *latex_diff.py* jaoks lisatakse URL lõppu *"/difference"*. Seejärel saadetakse tagarakendusele saadud tulemusi JSON formaadis. Samuti saadab see päringu testide tabeli uuendamiseks.

5.5 Andmebaas

Andmebaasi püsivuse tagamiseks kasutati Liquibase'i. See kindlustab andmebaasi struktuuri, tabelite ja andmetüüpide püsivust kasutades *changeset*-i ja seega seotud kontrollsummat [51]. Lisaks võimaldab Liquibase kirjutades skeemi erinevaid andmebaasihaldureid kasutada, mistõttu on see mugavam kui SQL päringute kirjutamine.

Üheks peamiseks alternatiiviks on Flyway [2], kuid see ei oma tagasivõtmise funktsionaalsust. Arvestades seda, et Liquibase on töö autoritele tuttavam, ei oleks Flyway kasutamine otstarbekas. Seda on mugav kasutada Java ja Spring Booti rakenduste jaoks ning see omab ka tagasivõtmise funktsionaalsust.

Andmebaasis on kokku 25 tabelit, millest 2 on *databasechangelog* ja *databasechangeloglock*, mis luuakse automaatika poolt. Tagarakenduse arendamise esimeseks sammuks oli andmebaasi struktuuri põhjalik läbimõtlemine. Prototüüpimiseks kasutati DBDiagram tarkvara [57]. Selle käigus konsulteeriti Tallinna Tehnikaülikooli dotsendi Erki Eessaarega, kes tegi palju kasulikke soovitusi ja märkusi, mis aitasid esmast relatsioonide struktuuri projekteerida. Seejärel tehti Liquibase'i XML fail, mille käivitamisel initsialiseeritakse andmebaas. See sisaldab 52 muudatuste hulka. Osad neist olid lisatud töö käigus hiljem, aga suurem osa oli juba prototüübis paika pandud.

[2] `https://www.red-gate.com/products/flyway/community/`

5.6 Andmebaasi struktuuri kirjeldus

Siin kirjeldatakse andmebaasi tähtsamaid relatsioone klastritena, et oleks lihtsam omada ülevaadet selle kohta.

Kasutajate haldamiseks on loodud järgmised tabelid: *lathec_user* - salvestab infot kasutaja kohta, *user_role* - defineerib kasutaja võimalikud rollid (kasutaja, õpetaja, administraator) *user_role_link* - ühendab kasutajat ja tema rolli (mitmevalentne side). Samuti on ka *gitlab_project* tabel, mis seob GitLabi projekte kasutajatega.

Õppesuundade salvestamiseks on loodud tabel *faculty*, millega võib siduda kasutajat ja meeskonda. Meeskondade haldamiseks on loodud järgmised tabelid: *team* - peamine info tiimi kohta, *team_secondary_info* - lisainfo tiimi kohta (õppesuund, klient), *team_role* - defineerib liikmete rolle (looja, liige, juhendaja), *team_member* - seob kasutajaid tiimiga koos kindla rolliga, *team_invitation/team_invite* - tiimiga liitumise kutsete haldamiseks.

Siltide *(tags)* haldamiseks on loodud järgmised tabelid: *tags* - defineerib olemasolevaid silte, *team_tags* - seob silte meeskondadega.

Teadete haldamiseks on loodud järgmised tabelid: *notification_type* - teavituste tüübid, *notification* - teavituse sisu ja staatuse jälgimiseks, *inbox* - teavituste kohalejõudmise realiseerimiseks.

Testide ja üleslaadimiste süsteemi jaoks loodud tabelid on: *test_category* - testide kategooriate defineerimiseks, *test* - info testi kohta, *submission* - üleslaadimise info, *submission_status* - üleslaadimise staatuse salvestamiseks, *document* - dokumendi ja selle info salvestamiseks, *failed_test*- läbikukkunud testide salvestamiseks, *comment* - üleslaadimiste kommentaaride salvestamiseks.

Suurte keelemudelite info ja viipade salvestamiseks olid lisatud tabelid *model_prompt* ja *model*. See oli vajalik, kuna kasutuses on kolm erinevat mudelit ning nende sisse-ja väljalülitamist on niiviisi mugavam hallata.

Peamised tabelivahelised suhted on:

- Kasutajad kuuluvad teaduskondadesse;

- Kasutajatel võib olla mitu rolli;
- Meeskondadel võib olla mitu eri rollidega liiget;
- Meeskonnad võivad olla märgistatud mitme sildiga;
- Dokumendid esitatakse testimiseks;
- Esitatud dokumentidel võivad olla kommentaarid ja ebaõnnestunud testid;
- Kasutajate vahel saadetakse teateid.

Alates Lisa 7 – Kasutaja rolli ja selle seose tabelid on diagrammid, mis olid tehtud kasutades DBeaveri tarkvara [58].

6 Veebirakendus

See peatükk keskendub kasutajaliidese arendusele. Räägitakse kasutatud disainimustrist, teekidest ja kirjeldatakse rakendatud funktsioone ja põhjendatakse nende valikut.

Rakendatavale kasutajaliidesele on järgmised nõuded:

1. **Kasutajaliides on vaistlik**. Iga vaade peab olema kooskõlas teiste komponentide ja nende olekutega. Pakutud funktsioonide kasutus on arusaadav, asjakohalik ja ei häiri kasutajaid.
2. **Kasutajaliides on reageerimisvõimeline**. Kasutajaliides peab olema reaktiivne. Andmete või komponendi seisundi muutmisel peab kasutaja koheselt nägema tehtud muudatusi.
3. **Kasutajaliides vastab TalTech stiiliraamatule**. Veebirakendus peab tunduma ja töötama nagu ametlik Tallinna Tehnikaülikooli teenus. Kasutajaliideses kasutatud värvid, nupud ja nende kompositsioon peavad vastama ametlikele stiilireeglitele.
4. **Kasutajaliides lihtsustab keerulisi ülesandeid**. Kasutajaliides ühendab kuju ja otstarvet. Taustas tehtavat keerulist loogikat ühendatakse lihtsa ja vaistliku liidesega, mille abil kasutaja täidab häirimata ülesandeid.

Pidades silmas peatükis 3 mainitud tehnoloogiavirna, kasutati veebirakenduse arenduses React teeki ja TypeScript programmeerimiskeelt. Arenduse kergendamiseks on autorid valinud hulga teekidest, mis täiustavad veebiarendust:

- **Axios** - lihtne ja kiire teek HTTP-päringute saatmiseks ja töötlemiseks [59].
- **React Router** - laialt kasutatud marsruutimise teek, mis võimaldab komponendivahelist navigeerimist [60].
- **TalTech Styleguide** - ametlik Tallinna Tehnikaülikooli teek kasutajaliidese rakendamiseks [10].
- **MSAL.js** - ametlik Microsofti teek kasutajate autentimiseks, kasutades Azure AD (*Azure Active Directory*) [61].

Veebirakenduse arendamiseks kasutati VSCodium tekstiredaktorit, Yarn paketihaldurit, Node.js JavaScript käigukeskkonda ja OrbStack virtualisaatorit. Kasutajaliidese testimiseks kasutati WebKit ja Blink esitusmootorite põhiseid veebibrausereid [62], [63], [64], [65], [66].

Järgmistes peatükkides tutvustatakse kasutajaliidese arendamiseks kasutatud disainimustrit ning kirjeldatakse veebirakenduses olevaid põhifunktsioone.

6.1 MVC disainimuster

Selleks, et veebirakenduse kood oleks kiire ja mõistetav, tuleb valida sobiv disainimuster. Disainimuster määratleb ja korrastab komponentide, mudelite ja funktsioonide seoseid.

Selle veebirakenduse arenduses rakendasid lõputöö autorid MVC (*Model-View-Controller*) disainimustrit selle lihtsuse ja paindlikkuse pärast. MVC keskmes on teemade lahususe põhimõte ning selle mustri järgi jagatakse rakendus kolmeks osaks [67]:

- Mudel (*Model*) - see osa vastutab rakenduse andmete loomise, haldamise ja salvestamise eest.
- Vaade (*View*) - see osa vastutab andmete kuvamise ja kasutajaliidese reaktiivsuse eest.
- Kontrollija (*Controller*) - see osa vastutab kasutaja poolt tegevuste töötlemise ja andmete või tehingute edastamise eest.

MVC võimaldab arendajatel mugavalt eraldada kasutajaliidest ja veebirakenduse loogikat, kuid seejuures säilitada osade ühtsust. Lisaks parandab see koodi uuendamist ja hooldust, kuna ühes osas väikeste muudatuste tegemine ei mõjuta teisi osasid. Ühendades Reacti paindlikkuse ja reaktiivsuse TypeScripti tüübirangusega, tagavad lõputöö autorid veebirakenduse kerge ja kiire arenduse.

6.2 Rollid

Analüüsides veebirakenduse põhifunktsioone, otsustasid lõputöö autorid jagada funktsioonid kasutajarollide. See parandab andmete kaitset, lihtsustab kasutajaliidese vaadete arendust ning selgendab kasutajatele funktsioonide ligipääsetavust.

Sobivate rollide valikul otsustasid autorid järgmiste rollide kasuks:

- Tudeng
- Õppejõud
- Haldaja

Järgmistes peatükkides käsitletakse rollide põhifunktsioone. Samuti selgitatakse rolli saamise mehhanismi.

6.2.1 Tudeng

Tudengi rolli all mõeldakse tavakasutajat, kellel on ligipääs veebirakenduse põhifunktsionaalsusele. See antakse vaikimisi kõigile registreerunud kasutajatele.

Tudengi roll annab juurdepääsu järgmistele funktsioonidele ja vaadetele:

- Bakalaureusetöö kontrollile saatmine ja varasemate kontrollide vaatamine.
- Saadud vigade kohta tagasisidestamine.
- Grupi loomine ja muutmine, otsimine ja grupiga liitumine.
- Teistele kasutajatele kutsete saatmine.
- Teavituste vaatamine ja nendega interaktsioon.

Kasutajaliideses loodud tavakasutajatele vaateid on näha osas Lisa 19 – Kasutajaliideses tudengite vaaded.

6.2.2 Õppejõud

Õppejõu rolli erinevus tavakasutajast seisneb selles, et see roll omab ligipääsu kontrollija täisfunktsionaalsustele. See tähendab, et õppejõu rolliga kasutajad, tohivad piiramatult analüüsida bakalaureusetöid kasutades suuri keelemudeleid, ning kasutada failierisuse funktsiooni. Lisaks on õppejõul näha gruppe, millega ta on liitunud, ning on saadaval gruppide ja tudengite otsingumootor.

Selle rolli omamiseks tuleb läbida järgmine toiming:

1. Kasutaja saadab avalduse rolli omamiseks.
2. Haldaja rolli omav kasutaja kontrollib andmete õigsust ja kinnitab avalduse.

Kasutajaliideses loodud õppejõule vaateid on näha osas Lisa 20 – Kasutajaliideses õppejõu vaaded.

6.2.3 Administraator

Võrreldes teiste rollidega, saab administraatori rolliga kasutaja juurdepääsu terve rakenduse funktsionaalsustele, võttes arvesse tudengi ja õppejõu funktsioone. Selle rolli eesmärgiks on lihtsustada rakenduse haldamist ning anda ülevaade valideerimistestide tagasisidest.

Administraatori roll võimaldab järgmiseid funktsioone kasutada:

- Saada lühike ülevaade tagasisidest iga valideerimistesti kohta.
- Õppejõu rolli saamise avalduste kinnitamine.
- Gruppide, kasutajate ja kontrollide otsing.
- Gruppide ja kasutajate andmete muutmine või kustutamine.
- Valideerimistestide sisse- ja väljalülitamine.
- Kõikidele kasutajatele teavituste saatmine.
- Viipade loomine, muutmine ja kustutamine.
- Suuri keelemudeleid sisse- ja väljalülitamine.

Selle rolli omamiseks on vaja saata avaldus teenuse halduritele või administraatoritele. Kuna see roll annab kasutajale märgatavalt suuremad õigused, peab administraatorite arv jääma kindlaks määratud ülempiiri alla.

Kasutajaliideses loodud administraatori vaateid on näha osas Lisa 21 – Kasutajaliideses haldaja vaaded.

6.3 Lõputöö valideerimisvoog

Veebirakenduse keskmes on lõputööde valideerimine, kasutades arendatud kontrollijat, mida kirjeldati peatükis 4. Selle funktsiooni ümber arendati kasutajaliidest ja tagarakendust. Autorid peavad järgmisi nõudeid valideerimisvoos kõige kriitilisemaks:

1. Valideerimise alustamine on vaistlik.
2. Valideerimisprotsess on automaatne.
3. Valideerimisprotsessi seis on mõistetav.

4. Kontrolli tulemuste vaatamine on intuitiivne.

Arvestades sellega, rakendasid autorid kasutajaliidest, mis on üles ehitatud nendele aspektidele. Rakendades Tallinna Tehnikaülikooli ametlikku teeki, õnnestus autoritel ühendada kuju ja otstarvet. Tulemuseks on informatiivne, meeldiv ja ligipääsetav kasutajaliides, mille pilte võib leida Lisa 16, Joonis 47.

Bakalaureusetööde valideerimisvoog on järgmine:

1. Kasutaja käsitsi arhiveerib lõputöö failid ZIP-konteinerisse.
2. Koondpaneelis lisatakse konteiner ja vajutades *Analyse* nuppu käivitatakse kontroll.
3. Kasutajaliides saadab päringu koos failidega tagarakendusele ning saades kätte kontrolli identifikaatori, navigeerib */submission/ID* otspunktile.
4. Tulemuste vaates iga 5 sekundi tagant küsitakse tagarakenduselt kontrolli seisundit.
5. Kui seisund võrdub kolmega, saadab kasutajaliides päringu, küsides tulemuste andmeid. Muidu kuvakse veateavet.
6. Laadides alla vajalikke andmeid, kuvakse tulemuste vaates, milles on näha lõputööd PDF-vormingus, vigade arvu ja kontrolli tulemusi.

6.4 Failide vahe

Peatükis 4.10 on mainitud, et töö autorid proovisid implementeerida LaTeX-failide erinevuste leidmist ja näitamist kasutajaliideses, kuid sellega tekkisid raskused. Seetõttu otsustati otsida viise, kuidas märkida versioonide erinevusi PDF-failides.

Antud funktsionaalsuse arendamise käigus prooviti mitmeid teeke. Kõige töötavamaks osutus Google'i *Diff-Match-Patch* teek, kuna see näitas lisatud teksti efektiivselt [68]. Paraku kordus selle lahenduse puhul sama probleem, mis ilmnes ka varasemalt - tekstiosad, mis olid nihutatud, loeti lisatuks. Töö autorid tegelesid selle puudujäägi parandamisega läbi erinevate algoritmide. Üheks neist oli näiteks räsifunktsioonide loomine tekstilõikude jaoks, et märkida liigutatud osi, kuid lõputöö kirjutamise hetkeseisuga on algoritm lõpetamata. Osaliselt töötav lahendus oli ka WebViewer'i API kasutamine, mis genereeris eraldi vaade kahe PDF-failidega - kustutatud ja lisatud tekstiga [69]. Kahjuks sisaldasid antud API tasuta versiooniga loodud dokumendid vesimärke ning selle eemaldamine rikuks autoriõiguseid. Saadud vaadet saab näha lisades (Lisa 25 – WebViewer API-ga saadud vaade).

Vaheotsingu vaade arenduse käigus tekkinud raskuste tõttu ei õnnestunud autoritel viia komponendi rakendamist lõpuni. Sellepärast selles peatükis keskendutakse kasutajaliidese poolest loodud tehingu voole ja nõuetele. Täiendavalt kirjeldatakse tekkinud keerukusi, mis takistasid arendust.

6.4.1 Nõuded

Arvestades vaheotsingu tehingu keerukusega disaini poolt, projekteerisid autorid kiire prototüübi, et selgendada põhikomponente ja kasutusvoogu. Järgnevad aspektid, mis on autorite arvates kõige põhjalikumad ja tähtsamad failide vahe vaadates:

- Kuvatakse erinevuste arv ja keskmine pikkus.
- Muudatusi kuvatakse tekstilises ja PDF-vormingus.
- Kasutajaliideses valgustatakse dokumendivahelised erinevused nähtavuse parendamiseks.

6.4.2 Tehingu voog

Kasutajaliidese arenduse lihtsustamiseks töötasid autorid välja võimaliku tehinguvoo algusest lõpuni. See vastab lihtsuste ja vaistlikkuse nõuetele ning parandab arendust:

1. Valitakse kaks ZIP-konteinerit, milles sisalduvad lõputöö erinevad versioonid.
2. *Dashboard* vaates valitakse *Compare Documents* komponent ja laaditakse konteinerid üles.
3. Vajutades *Compare* nuppu käivitatakse failide vaheotsingu protsess.
4. Kasutaja suunatakse */difference* otspunktile.
5. Kasutajaliides saadab regulaarselt päringuid tagarakendusele, küsides protsessi seisundi kohta.
6. Kui seisund võrdub kahega, teeb kasutajaliides päringu, küsides vaheotsingu tulemuse andmeid. Muidu kuvatakse veateade.
7. Saades kätte andmeid, kuvatakse failide vahe vaadet, mis koosneb mitmest komponendist: erinevuste arv, muudatuste tekstiline kirjeldus ning kaks kõrvuti asuvat PDFi. PDFides on valgustatud osad, mida kontrollija peab erinevaks.
8. Kasutaja saab kerida PDF vaateid ning tekstilises kirjelduses muudatusele vajutades näitab kasutajaliides automaatselt vastavat osa PDFis.

6.4.3 Tekkinud takistused

Suurimad raskused tekkisid teksti valgustamisega PDF vaadetes. Nagu mainiti peatükis 4, olid arenduse jooksul raskused esiletõstmise tehnilise teostusega, ning erinevate viiside katsetamiste jooksul koodi edasiarendus ja haldus muutus ebavajalikult keeruliseks.

Esimene valik seisnes selles, et veebirakenduses kasutati *react-pdf* teeki PDFi kuvamiseks [70]. Selle teegi eeliseks oli automatiseeritud PDFis tekstiotsing ja süntaksivalgustamine. Selle funktsiooniga olidki autoritel suurimad raskused. Teostuse keerukus kasvas oluliselt seoses sellega, kuidas *react-pdf* parsib PDFi lehtedelt teksti: jooksvalt loetakse igalt realt üks või mitu sõna ja lehe lõppu jõudmisel ühendatakse sõnade loend üheks tekstiks. Lisaks liigendamise jooksul tekkisid moonutused, nagu lisatühikud või kirjavahemärgid, mistõttu kontrollija ja PDFi teksti võrdlemine muutus palju keerulisemaks.

Teine erikujund oli kasutada React PDF valgustamisteeke, mis kasutavad piirdekarpe (*Bounding Boxes*). See tähendab, et kasutajaliidesele koos PDF-vorminguga saadetakse iga valgustuse kohta asendi ja suuruse andmed ning kasutatav teek kuvab neid PDF komponendi peal, lisades uue kihi. Selle võimaluse rakendamiseks kasutati *react-pdf-highlighter* teeki, kuid esmane keerukus nüüd nihkus tagarakendusele ja kontrollijale, millega seotud raskused kirjeldati peatükis 4 [71].

7 Analüüs

Pärast projekti lõpuleviimist on oluline analüüsida selle aspekte, mida oleks võinud arendamise käigus parandada.

Üks olulisi probleeme, millega tuli kokku puutuda, oli LLMiga tegelemine lõputöö teksti õigsuse kinnitamiseks. Täpsemalt oli probleemiks, et LLMid ei näidanud tugevat sooritust eesti keeles. Selle tulemusena tuli rakendada lahendus, mis hõlmas kahte erinevat mudelit. Kahjuks kahe mudeli süsteemi kasutamine ei lahendanud algprobleemi ja autorid oleksid võinud valida optimaalsema lahenduse. Näiteks oleksid autorid võinud uurida võimalust kasutada TalTechis välja töötatud eestikeelset LLMi. Teine alternatiiv oleks olnud kasutada isehallatavaid mudeleid, näiteks Tartu Ülikoolis välja töötatud eestikeelset tekstianalüsaatorit[1]. Kui need võimalused tuvastati, oli LLM-kontrollide rakendamine alternatiivsete mudelite abil juba olemas ja tähelepanu oli muudel projekti aspektidel.

Teine probleem tuleneb vanade LaTeX-i mallide laialdasest kasutamisest. Kui üliõpilased esitasid lõputööd kinnitamiseks, selgus, et paljud üliõpilased kasutasid vananenud LaTeX-i malli, millega arendatud teenus ei arvestanud lootuses, et kasutatakse uuendatud malli. Selle vältimiseks oleks võinud tagada, et vana mall oleks algusest peale kontrolleriga ühilduv või lisada rohkem kontrolle malli versiooni tuvastamiseks. Praegu on valideerimissüsteem uuendatud, et võtta arvesse vanemaid malle ja enamasti annab rahuldavaid tulemusi.

Täiendavalt autorid ei tegelenud projekti DevOps süsteemide seadistamisega alguses, vaid pigem projekti lõpus. Hilisema konfigureerimise tulemusena on torujuhtme valideerimine projekti keerulisema struktuuri tõttu raskem protsess. Lisaks sellele põhjustas kontrollija ressursinõuete esialgne hindamine suure koormuse korral jõudluse pingestumise. See tähendab, et riistvara vastavust oleks pidanud enne projekti algust põhjalikumalt hindama.

Hoolimata probleemidest võiks see olla kindel alus edasiseks arendamiseks. Rõhku pandi sellele, et projekt oleks võimalikult lihtsalt hooldatav. See hõlmab struktureeritud ja puhta

[1] `https://tartunlp.ai/`

koodi järgimist koos koodi dokumentatsiooniga. Selline lähenemisviis tagab, et tulevased arendajad saavad koodibaasist kergesti aru ja saavad funktsionaalsust võimalikult lihtsalt muuta.

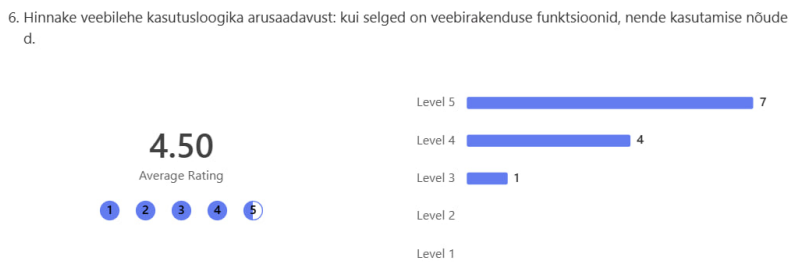
8 Tagasiside analüüs

Tagasiside saamiseks oli loodud Microsoft Formsi küsimustik, mida on näha Lisas Lisa 24 – Laiali saadetud küsimustik. See saadeti laiali vastamiseks 17.05.2025 Tallinna Tehnikaülikooli informaatika tudengitele sotsiaalmeedia kaudu. Küsitlus oli läbi viidud viimasel nädalal enne kirjaliku osa esitamist, kuna oodati, et kontrollijat hakatakse kasutama just sellel ajal. Töö autorid lootsid saada statistikat veebilehe kaudu. Sinna on sisse ehitatud analüütika ja tagasiside andmise võimalused. Kahjuks kasutasid seda teenust vaid üksikud inimesed ning kõige enam autorid ise. Vaatamata sellele, aitas osade kasutajate sõnul veebirakendus parandada tõsiseid vigu, mida nad ise poleks märganud. See näitab, et antud projektil on potentsiaal olla abiks dokumentide kirjutamisel. Nagu varem mainitud, kasutas suurem osa inimesi LaTeX-i malli vanemat versiooni. Kasutajate sõnul takistas see antud veebirakenduse kasutamist kõige enam, kuna keegi ei soovinud juba olemasolevat versiooni uuendada ja suuri muutusi teha.

8.1 Küsimustiku tulemused

Antud peatükis kirjeldatakse täpsemalt küsitluse läbiviimise käigus saadud vastuseid. 04.06.2025 seisuga on küsimustikule vastanud 12 inimest. Esimesed kaks küsimust olid lisatud selleks, et aru saada, mis versiooni LaTeX-i malli kasutati, et analüüsida, kas järgnevad vastused sõltuvad vanas versioonis kirjutamisest. Siiamaani on kõik vastajad kasutanud uuemat malli. Edasised küsimused uurivad kasutajaliidese mugavust ja selle disaini intuitiivsust. 5-punktiskaalal hinnati selle meeldivust 4.5-ga. Kõik vastajad väitsid, et kujundus sarnaneb TalTechi ametliku veebilehe omaga. 100-protsendilist positiivset tagasisidet sai andmete kuvamine veebilehel. Kellelgi ei tekkinud sellega probleeme. Rakenduse funktsioonide arusaadavuse hinnanguid on näha Joonistel 6 ja 7.

Antud tulemused viitavad, et see on piisavalt kasutajasõbralik, kuid on vaja veel edasisi uuringuid, et selgitada täpsemalt välja, kus esinevad raskuskohad. Järgnevad küsimused uurisid kasutajatelt, kuivõrd lihtne oli kontrolli alustamine, tulemuste vaatamine, lõputööde

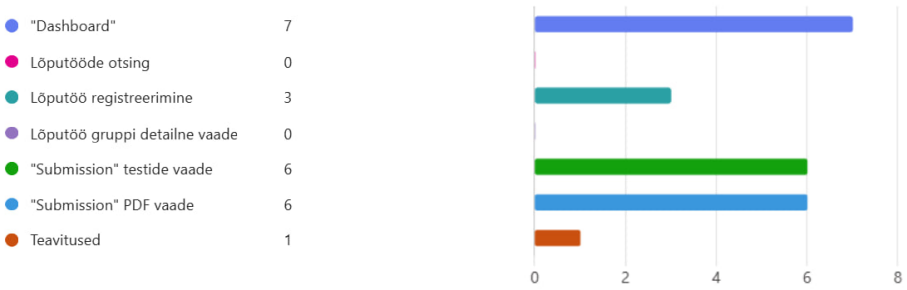


Joonis 6. Kasutajate hinnang rakenduse funktsioonide arusaadavuse kohta.

1	anonymous	Ma sain aru, kuidas ma saan esitada oma tööd analüüsimiseks, kuidas ma näen tagasiside ja mis kohas mõni viga on. Samuti oli selge varasemate esitamiste vaatamine.
2	anonymous	Minu meelest võiks olla selgem, et milliseid faile see zip päriselt tahab.
3	anonymous	Alguses tekitas natuke segadust, kust kaudu ma saan faile alla laadida, aga siis leidsin üles. Muus osas oli kõik väga loogiline ja arusaadav.
4	anonymous	Veebilehe struktuur oli loogiline ja ma sain aru mida teha
5	anonymous	Kõik on intuitiivne
6	anonymous	Väga selge kasutusloogika, ei ole vaja kulutada aega konkreetse funktsionaalsuse otsimisele
7	anonymous	Muidu oli arusaadav, aga võibolla headeris võiks olla ka mingi "Analüüsi" tab, praegu ainult Search Theses ja upload lehele saab taltech logo vajutades ainult

Joonis 7. Kasutajate põhjendused oma hinnangule.

registreerimine. Kumbki ei saanud vähem kui 4.2 punkti 5-st. Kuna see on suur osa projektist, võib antud tulemust pidada heaks näitajaks kogu rakenduse edukusele. Lisaks oli uuritud, mis olid kasutajate arvates meeldivaimad vaated. Selle tulemusi saab näha Joonisel 8.



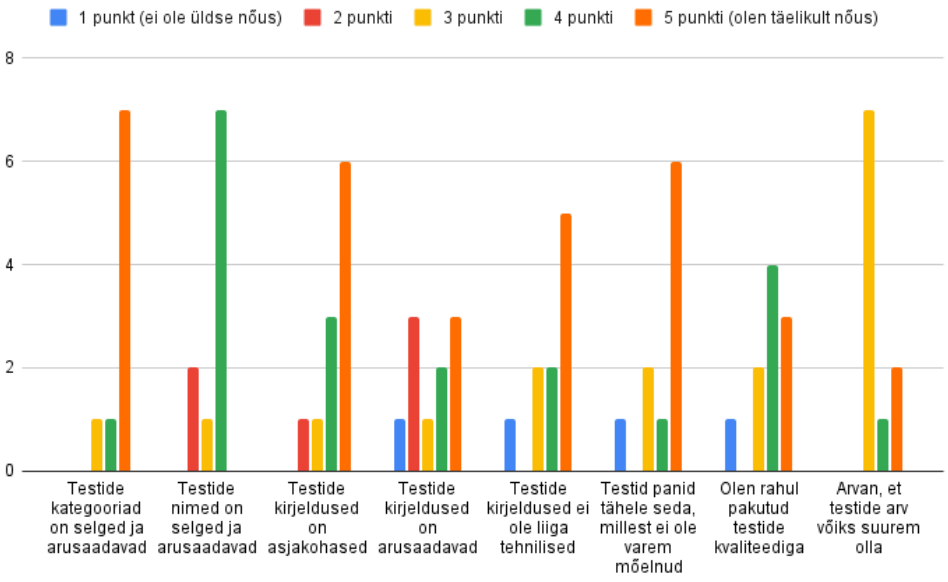
Joonis 8. Kasutajate lemmikumad vaated veebilehel.

Küsiti ka tulemuse saamise aja kohta. Üksteist vastajat kaheteistkümnest väitsid, et kontrolli tulemus saabus kiiresti. Üks kasutaja ei olnud kindel. Üleüldse teenuse kiirust hinnati 4.5 punktiga. Üks vastaja kirjutas: “Minu jaoks oli kontroll üllatavalt kiire.” Arvestades sellega, et rakenduse optimeerimine oli töö autorite jaoks üheks väljakutseks, siis antud vastused

näitavad, et see õnnestus. Küsimuse “Milliste testidega olete kõige tihedamini kokku puutunud kasutades antud teenust?” järgi on näha, et enamikel kukkuvad läbi järgmised testid populaarsuse kahanevas järjekorras:

- Viited Vikipeediale - see kontroll oli välja lülitatud, kuna Vikipeedia kasutamine allikana sõltub kontekstist.
- Sobimatud sulud
- Sulgudevaheline tühik
- Sobimatud loogelised sulud
- Struktuuri test
- Lisade pealkirja vorming, peatükkide pealkirjad suurte algustähtedega
- Viidete vorming
- Lühendid
- Faili sisu mittevastavus

Testide ülesehituse kohta oli samuti tehtud küsimusi. Nende nimede selgust hinnati 4,44 punktiga ja nende kirjeldusi 3,67-ga. See annab märku, et kirjeldusi tuleks muuta põhjalikumaks ning nende kujundust ilusamaks. Tuvastatud vigade asjakohasus sai tagasisideks 3.78 punkti viiest. Antud küsimus vajab edasisi testimisi kasutajatega. Üldisemaid vastuseid testide kohta saab näha Joonisel 9.



Joonis 9. Kasutajate hinnangud testidele.

Viimased 4 küsimust keskenduvad üldisele rahulolule teenusega. Kaheksa inimest kaheteistkümnest soovitaksid teistele kindlasti antud veebirakendust. Kolm neist pigem soovitaksid ning üks kasutaja ei ole selles väga kindel. Küsimusele “Kui lihtne Teie arvates oleks teenust kasutada mittetehnilisel erialal õppivatele tudengitele?” vastas suurem osa inimestest, et pigem oleks keerulisem. Väitele, et teenus on otstarbekas lahendus LaTeX-is kirjutatud lõputööde stiili- ja kirjavigade kontrolliks, anti keskmine hinne 5,8 seitsmest. Üks inimene pani vastuseks 2 punkti, mille tõttu tulemus langes. Töö autorid said tagasisidet ka küsimustikust sõltumatult. Mitu kasutajat jagasid oma positiivset kogemust privaatsõnumite kaudu. Üldiselt on tulemused rahuldavad ning teenust võib pakkuda inimestele, kes soovivad oma tööd kontrollida.

9 Kokkuvõte

Käesoleva lõputöö eesmärgiks oli luua süsteemi bakalaurusetööde LaTeX dokumentides grammatikavigade otsimiseks ja stiili kontrollimiseks vastavalt Tallinna Tehnikaülikooli vormindamisreeglitele. See aitaks tudengitele lõputöid parendada ja õppejõududele keskenduda lõputööde sisule, mitte vormindamis- ja grammatikavigadele.

Töö raames arendati veebirakendus, mis koosneb Reacti esirakendusest, Spring Bootis kirjutatud tagarakendusest, Pythonis loodud kontrollijast ning PostgreSQL andmebaasihaldurist. Kihtide isoleerimiseks kasutati Docker tarkvara. Autorid uurisid erinevaid tarkvara arhitektuurimustreid ja valisid kõige sobivamad loodava süsteemi jaoks (kolmekihiline arhitektuur ja MVC).

Alfatestimise tulemused näitasid, et süsteemil on potentsiaal kasutamiseks, kuid see vajab täiustamist ja parendamist. Testimises osalesid 12 inimest, kelle käest koguti tagasisidet kasutajaliidese ja valideerimistestide kvaliteedi kohta. Lisaks jooksvalt koguti andmeid süsteemi jõudluse ja kasutaja käivitamise kohta. Nende tagasiside põhjal avastati valideerimistestides esinevaid vigu, ning kasutajaliidese mõistetavuse parandamise võimalusi.

Töö käigus viidi ellu enamik püstitatud eesmärkidest, kuid on jäänud võimalused teenuse parendamiseks ja uuendamiseks. Välja kujunenud süsteem on tugev alus edasisteks täiustusteks ja arendusteks.

Kasutatud kirjandus

[1] LaTeX Project. *LaTeX – A document preparation system*. [Accessed: 19-05-2025]. URL: https://www.latex-project.org/.

[2] NLTK Project. *Natural Language Toolkit*. [Accessed: 19-05-2025]. URL: https://www.nltk.org/.

[3] Rod Stephens. *Beginning Software Engineering*. John Wiley & Sons, 2015.

[4] Chris Richardson. *Pattern: Monolithic Architecture*. [Accessed: 10-05-2025]. URL: https://microservices.io/patterns/monolithic.html.

[5] Chris Richardson. *Pattern: Microservice Architecture*. [Accessed: 13-05-2025]. URL: https://microservices.io/patterns/microservices.html.

[6] Mark Richards. *Software Architecture Patterns*. Toim. Heather Scherer. O'Reilly Media, Inc, 2015.

[7] Francesco Vergona ja Vinit Sharma Adam Nemeth. *Building a three-tier architecture on a budget*. [Accessed: 13-05-2025]. 2024. URL: https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/three-tier-architecture-overview.html.

[8] IBM. *Three-tier architectures*. [Accessed: 14-05-2025]. URL: https://www.ibm.com/docs/en/was-nd/9.0.5?topic=overview-three-tier-architectures.

[9] Microsoft. *TypeScript is JavaScript with syntax for types*. [Accessed: 17-05-2025]. URL: https://www.typescriptlang.org/.

[10] Tallinna Tehnikaülikool. *TalTech Digital Styleguide*. [Accessed: 16-05-2025]. URL: https://storybook.taltech.ee/?path=/docs/docs-intro--docs.

[11] StackOverflow. *Technology | 2024 Stack Overflow Developer Survey*. [Accessed: 15-05-2025]. 2024. URL: https://survey.stackoverflow.co/2024/technology.

[12] Inc Meta Platforms. *Quick Start | React*. [Accessed: 15-05-2025]. URL: https://react.dev/learn.

[13] Inc Meta Platforms. *React Reference Overview*. [Accessed: 14-05-2025]. URL: https://react.dev/reference/react.

[14] Evan You. *Introduction | Vue.js*. [Accessed: 15-05-2025]. URL: https://vuejs.org/guide/introduction.html.

[15] Sergey Miroshnychenko. *Pros and Cons of Using Vue.Js for Web App Development*. [Accessed: 15-05-2025]. 2023. URL: https://ficustechnologies.com/blog/pros-and-cons-of-using-vue-js-for-web-app-development/.

[16] Filip Tobiasz. *Using Vue: Props and Cons*. [Accessed: 15-05-2025]. 2022. URL: https://thecodest.co/blog/pros-and-cons-of-vue/.

[17] Google. *What is Angular?* [Accessed: 15-05-2025]. URL: https://angular.dev/overview.

[18] José Matos. *The pros and cons of using Angular for your next frontend project*. [Accessed: 16-05-2025]. 2023. URL: https://angulardive.com/blog/the-pros-and-cons-of-using-angular-for-your-next-frontend-project/.

[19] The Apache Software Foundation. *Apache Maven*. [Accessed: 17-05-2025]. URL: https://maven.apache.org/.

[20] Broadcom Inc. *Spring Boot*. [Accessed: 17-05-2025]. 2025. URL: https://docs.spring.io/spring-boot/index.html.

[21] RedHat Inc. *Guides - Quarkus*. [Accessed: 17-05-2025]. URL: https://quarkus.io/guides/.

[22] The PostgreSQL Global Development Group. *PostgreSQL 17.5 Documentation*. [Accessed: 16-05-2025]. 2025. URL: https://www.postgresql.org/docs/17/index.html.

[23] Runcloud Sdn Bhd. *SQLite vs MySQL vs PostgreSQL - The Search For The "Best"Relational Database Management System*. [Accessed: 16-05-2025]. 2025. URL: https://runcloud.io/blog/sqlite-vs-mysql-vs-postgresql.

[24] Mark Drake. *SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems*. [Accessed: 16-05-2025]. 2022. URL: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems.

[25] SQLite Consortium. *SQLite Documentation*. [Accessed: 16-05-2025]. URL: https://sqlite.org/docs.html.

[26] Alexey Naumov. *Separation of Concerns in Software Design*. [Accessed: 17-05-2025]. 2020. URL: https://nalexn.github.io/separation-of-concerns/.

[27] Docker Inc. *Docker Docs*. [Accessed: 17-05-2025]. URL: https://docs.docker.com/.

[28] The YAML Project. *YAML Ain't Markup Language*. [Accessed: 17-05-2025]. URL: https://yaml.org/.

[29] Philippe Faist. *latexwalker — Calling Parsers for LaTeX Code*. [Accessed: 15-05-2025]. URL: https://pylatexenc.readthedocs.io/en/latest/latexwalker/.

[30] Alvin Wan. *TexSoup*. [Accessed: 15-05-2025]. URL: https://github.com/alvinwan/texsoup.

[31] Alvin Wan. *TexSoup - Parsing Mechanics*. [Accessed: 15-05-2025]. URL: https://texsoup.alvinwan.com/docs/parser.html.

[32] F. Boulogne M. Weiss. *BibtexParser*. [Accessed: 15-05-2025]. URL: https://bibtexparser.readthedocs.io/en/main/quickstart.html.

[33] David Wright. *Phunspell*. [Accessed: 15-05-2025]. URL: https://github.com/dvwright/phunspell.

[34] László Németh. *Hunspell*. [Accessed: 15-05-2025]. URL: https://hunspell.github.io/.

[35] Alexandre Menasria Arthur Fournier. *Bienvenue to Mistral AI Documentation*. [Accessed: 17-05-2025]. URL: https://docs.mistral.ai/.

[36] OpenAI. *OpenAI developer platform*. [Accessed: 17-05-2025]. URL: https://platform.openai.com/docs/overview.

[37] Google DeepMind. *Google DeepMind Gemini*. [Accessed: 03-06-2025]. URL: https://deepmind.google/models/gemini/.

[38] Vectara Inc. *Hallucination Leaderboard*. [Accessed: 03-06-2025]. URL: https://github.com/vectara/hallucination-leaderboard.

[39] Tartu NLP. *Tehisaru baroometer*. [Accessed: 04-06-2025]. URL: https://baromeeter.tartunlp.ai.

[40] Spring AI Team. *Spring AI*. [Accessed: 17-05-2025]. URL: https://docs.spring.io/spring-ai/reference/index.html.

[41] The Apache Software Foundation. *Apache PDFBox® - A Java PDF Library*. [Accessed: 17-05-2025]. URL: https://pdfbox.apache.org/.

[42] Hàn Thé Thành. *Pdflatex*. [Accessed: 15-05-2025]. URL: https://ctan.org/pkg/pdftex.

[43] Philippe Faist. *Latex2text — Simple Latex to Text Converter*. [Accessed: 15-05-2025]. URL: https://pylatexenc.readthedocs.io/en/latest/latex2text/.

[44] Pablo Pizarro. *Pydetex*. [Accessed: 15-05-2025]. URL: https://pydetex.readthedocs.io/en/latest/.

[45] NLTK Project. *NLTK - nltk.tokenize.punkt module*. [Accessed: 15-05-2025]. URL: https://www.nltk.org/api/nltk.tokenize.punkt.html.

[46] Jerome Lelong. *LaTeX-Workshop Compile*. [Accessed: 17-05-2025]. URL: https://github.com/James-Yu/LaTeX-Workshop/wiki/Compile.

[47] Artifex. *PyMuPDF documentation*. [Accessed: 17-05-2025]. URL: https://pymupdf.readthedocs.io/en/latest/.

[48] Pygit2 contributors. *Pygit2 - Diff*. [Accessed: 15-05-2025]. URL: https://www.pygit2.org/diff.html.

[49] Difflib mainteners. *Difflib — Helpers for computing deltas*. [Accessed: 15-05-2025]. URL: https://docs.python.org/3/library/difflib.html.

[50] Broadcom Inc. *Spring Data JPA*. [Accessed: 17-05-2025]. URL: https://docs.spring.io/spring-data/jpa/reference/jpa.html.

[51] Inc Liquibase. *Liquibase Documentation*. [Accessed: 16-05-2025]. URL: https://docs.liquibase.com/home.html.

[52] Mockito. *Tasty mocking framework for unit tests in Java*. [Accessed: 04-06-2025]. URL: https://site.mockito.org/.

[53] Inc Postman. *Send API requests and get response data in Postman*. [Accessed: 17-05-2025]. URL: https://learning.postman.com/docs/sending-requests/requests/.

[54] Broadcom Inc. *Spring Security*. [Accessed: 17-05-2025]. URL: https://spring.io/projects/spring-security.

[55] Microsoft. *Microsoft Entra ID*. [Accessed: 17-05-2025]. URL: https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id.

[56] GitLab Inc. *REST API*. [Accessed: 17-05-2025]. URL: https://docs.gitlab.com/api/rest/.

[57] Holistics Software. *DBDiagram documentation*. [Accessed: 17-05-2025]. URL: dbdiagram.io.

[58] DBeaver team. *DBeaver*. [Accessed: 17-05-2025]. URL: https://dbeaver.io/.

[59] John Jakob Sarjeant Matt Zabriskie. *axios*. [Accessed: 17-05-2025]. URL: https://axios-http.com/.

[60] Shopify. *React Router*. [Accessed: 17-05-2025]. URL: https://reactrouter.com/.

[61] Microsoft. *microsoft-authentication-library-for-js*. [Accessed: 17-05-2025]. URL: https://github.com/AzureAD/microsoft-authentication-library-for-js.

[62] VSCodium. *VSCodium*. [Accessed: 17-05-2025]. URL: https://vscodium.com/.

[63] Yarn. *Yarn*. [Accessed: 17-05-2025]. URL: https://yarnpkg.com/.

[64] OpenJS Foundation. *Node.js*. [Accessed: 17-05-2025]. URL: https://nodejs.org/en.

[65] Apple Inc. *WebKit*. [Accessed: 17-05-2025]. URL: https://webkit.org/.

[66] The Chromium Projects. *Blink (Rendering Engine)*. [Accessed: 17-05-2025]. URL: https://www.chromium.org/blink/.

[67] Adam Freeman. *Pro Design Patterns in Swift*. Apress, 2015.

[68] Google. *The Diff Match and Patch Libraries*. [Accessed: 02-06-2025]. URL: https://github.com/google/diff-match-patch?tab=readme-ov-file.

[69] Apryse Docs. *API to compare PDFs for differences*. [Accessed: 02-06-2025]. URL: https://docs.apryse.com/web/guides/compare/apis.

[70] Wojciech Maj. *react-pdf*. [Accessed: 17-05-2025]. URL: https://github.com/wojtekmaj/react-pdf.

[71] Artem Tyurin. *react-pdf-highlighter*. [Accessed: 17-05-2025]. URL: https://github.com/agentcooper/react-pdf-highlighter.

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

Meie, Maksim Usmanov, Aleksandra Tremba ja Katarina Zemljanski

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Automaatse süsteemi arendus LaTeX-põhiste lõputööde keelelise ja tehnilise kvaliteedi kontrollimiseks", mille juhendaja on Ago Luberg
    1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
    1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

08.06.2025

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

Lisa 2 – GitLabi repositooriumite lingid

https://gitlab.cs.taltech.ee/lathec/aurora/lathec-frontend

https://gitlab.cs.taltech.ee/lathec/transistor

https://gitlab.cs.taltech.ee/lathec/curiosity

Lisa 3 – Latheci veebilehe link

https://cs.taltech.ee/services/lathec

Lisa 4 – Tagarakenduse Docker Compose failisisu

```
services:
    backend:
        container_name: lathec-backend
        build:
        context: .
        dockerfile: Dockerfile
        volumes:
            - ".:/app"
            - "/app/"
            - /mnt/work_storage:/student_data
        environment:
        SPRING_DATASOURCE_URL: ${SPRING_DATASOURCE_URL}
        SPRING_DATASOURCE_USERNAME: ${SPRING_DATASOURCE_USERNAME}
        SPRING_DATASOURCE_PASSWORD: ${SPRING_DATASOURCE_PASSWORD}
        PYTHON_SERVER_URL: ${PYTHON_SERVER_URL}
        FILE_DIRECTORY: "/student_data/"
        networks:
            - checker-network
            - front-back-network
            - db-network
        ports:
            - 8000:8000
        depends_on:
            - db
        restart: always

    db:
        container_name: db
        image: postgres:latest
        environment:
        POSTGRES_DB: ${POSTGRES_DB}
        POSTGRES_USER: ${POSTGRES_USER}
        POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
```

```
        networks:
            - db-network
        ports:
            - 5432:5432


networks:
    checker-network:
        external: true
    front-back-network:
        external: true
    db-network:
        driver: bridge
```

Lisa 5 – Valideerimistestide kirjeldused

Tabel 2. Valideerimistestide kirjeldused.

Testi nimi	**Kirjeldus**
Abbreviations	Kontrollib, kas lõputöö jaoks loetletud lühendid ja terminid on tähestikulises järjekorras.
Appendix title format	Kontrollib, et LaTeX-dokumendi lisade pealkirjad järgivad õiget vormistust, sealhulgas topeltkriipsu kasutamine pärast lisa numbrit, pealkirja olemasolu ja suurtähtedega kirjutamine ning pealkirja suurtähtedest kinnipidamine.
Empty inputs in main	Kontrollib chapters main faili kõiki input käskude, mis on tühjad.
Empty inputs in chapters	Kontrollib peatükkide sees kõiki input käskude, mis on tühjad.
Empty sections	Kontrollib peatükkide sees kõiki section käskude, mis on tühjad.
Section titles with capitals	Kontrollib, kas iga peatüki jaotiste ja alapeatükkide pealkirjad järgivad korrektset pealkirja suurtähestikku vastavalt inglise keele standardkonventsioonidele.
Capitalized sentence	Kontrollib, et iga lause dokumendis algab suure tähega, jättes vahele numbrid ja erimärgid.
Double spaces	Avastab kõik laused, mis sisaldavad tekstis topelt tühikuid.
Pronouns	Otsib tekstist konkreetsete asesõnade kasutamist ja märgistab nende olemasolu.
Unmatched braces	Tuvastab laused, kus avanevate ja sulgevate kumerate sulgude arv ei ühti.
Unmatched parentheses	Tuvastab laused, kus ava- ja sulgemissulgude arv ei ühti.
Unmatched square brackets	Tuvastab laused, kus avavate ja sulgevate nurksulgude arv ei ühti.

Jätkub...

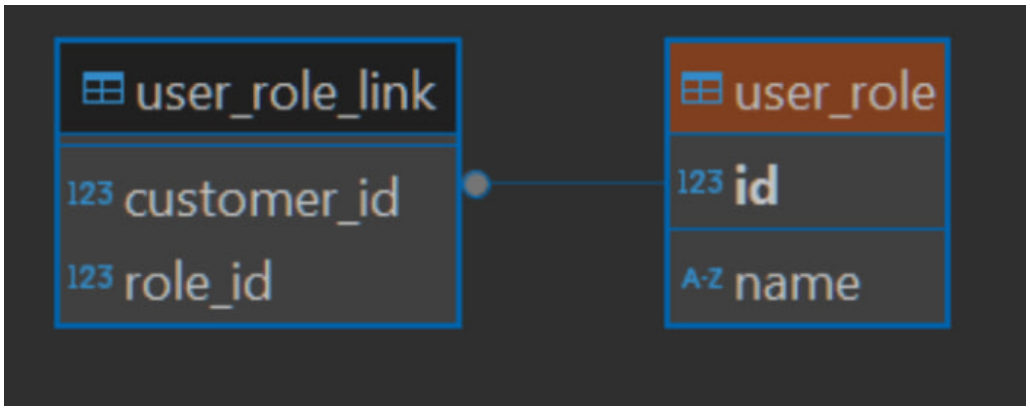
Tabel 2 – *Jätkub...*

Testi nimi	**Kirjeldus**
Unmatched quotes	Tuvastab laused, kus ava- ja lõpujutumärkide arv ei ühti.
Repeated punctuation	Kontrollib korduvate kirjavahemärkide kasutamist lausetes, välja arvatud ellipsid (...).
Parentheses spacing	Kontrollib lauses olevate sulgude ümber vale vahekauguse olemasolu, näiteks täiendavaid tühikuid sulgude sees või enne sulgusid.
Parts ratio	Kontrollib, et iga dokumendiosa sõnade arvu suhe jääks etteantud vahemikku. See on pigem soovitus kui nõue.
All references in text	Kontrollib, et kõik bibliograafias loetletud viited on tekstis viidatud.
Wikipedia references	Kontrollib bibliograafiast viiteid Vikipeediale, mida akadeemilises kirjutamises ei soovitata.
Cites sentence format	Kontrollib, et mitu viidet poleks valesti rühmitatud ja oleksid lause struktuuris õigesti paigutatud.
References count	Tagab, et bibliograafia sisaldab vähemalt kindlaksmääratud minimaalset arvu viiteid.
Missing files	Kontrollib, kas mallil olevad failid puuduvad või on lõputöö failides tühjad.
File content mismatches	Kontrollib mallifailide ja vastavate lõputööde failide sisulisi erinevusi.
Figures and tables test	Kontrollib jooniste ja tabelite vormingut, alamjooniste siltide olemasolu ning tagab, et iga lause pealkirjas algab suure algustähega ja lõpeb punktiga.
Compilation test	Kontrollib, et LaTeX-i kompileerimisel ei tekiks probleeme.
Structure test	Kontrollib, et kõik kontrollimiseks vajalikud failid on olemas.
Spell checker	Kontrollib õigekirja.

Lisa 6 – Kompileerimistesti kood

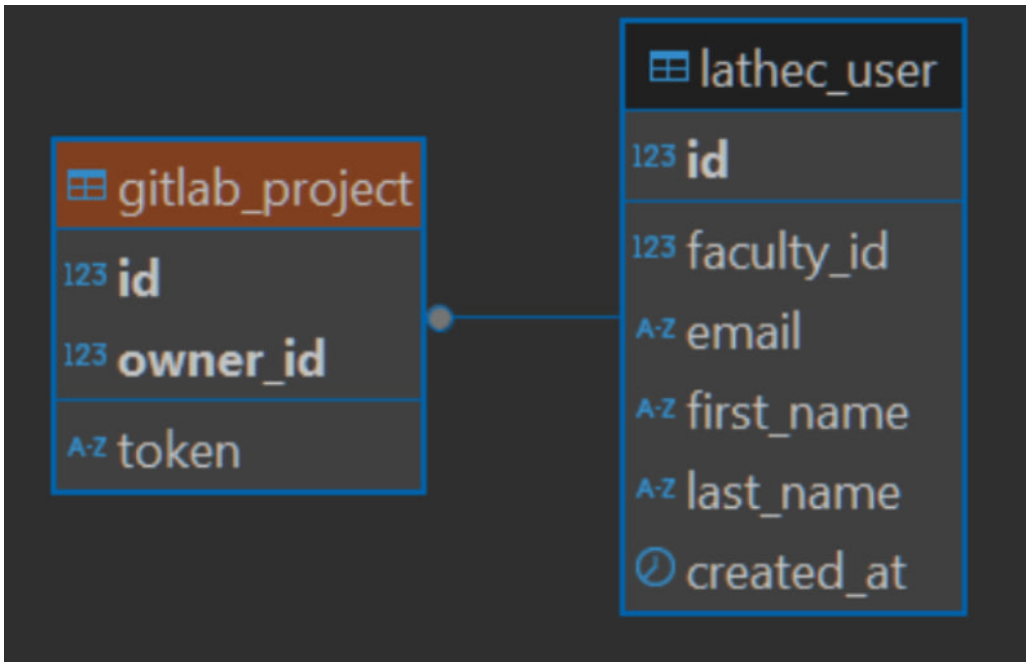
```
    result = subprocess.run(
                [
                    "pdflatex",
                    "-interaction=nonstopmode",
                    f"-output-directory={temp_dir}",
                    file_name
                ],
                cwd=working_dir,
                stdout=subprocess.PIPE,
                stderr=subprocess.PIPE
            )
```

Lisa 7 – Kasutaja rolli ja selle seose tabelid



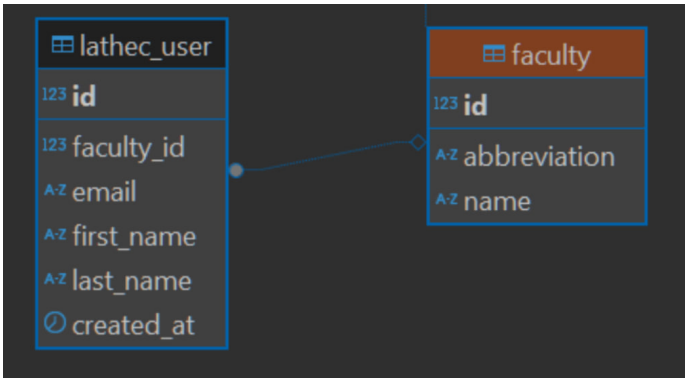
Joonis 10. Kasutaja rolli ja selle seose tabelid.

Lisa 8 – GitLabi projekti ja kasutaja seos



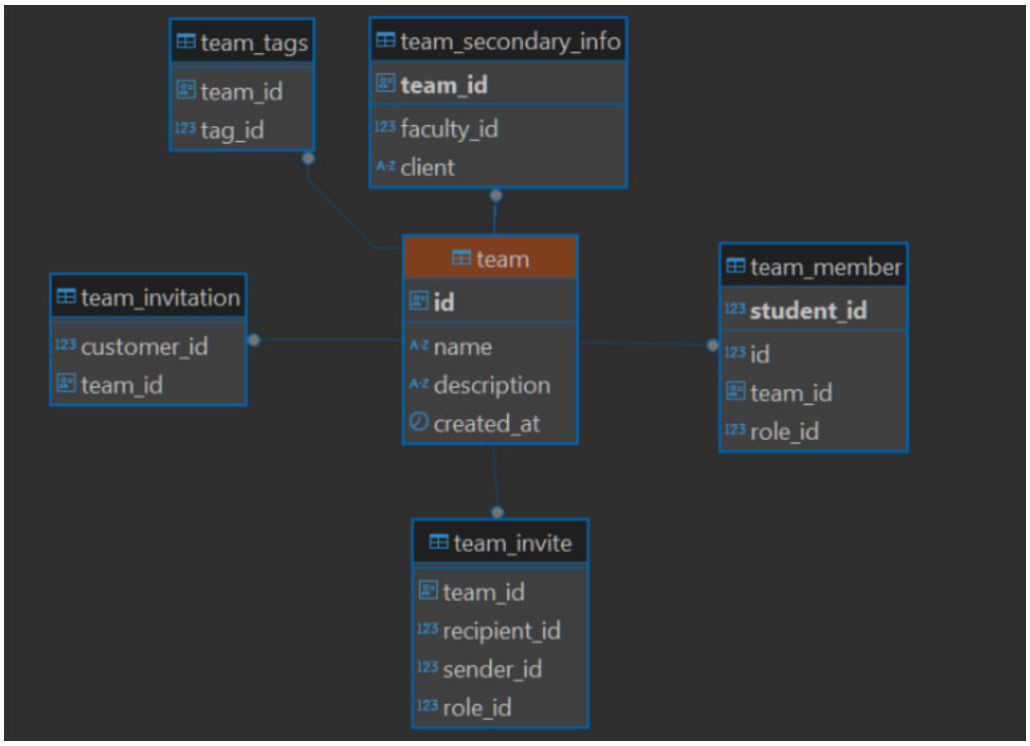
Joonis 11. Kasutaja ja GitLabi projekti tabelid.

Lisa 9 – Õppesuuna ja kasutaja seos



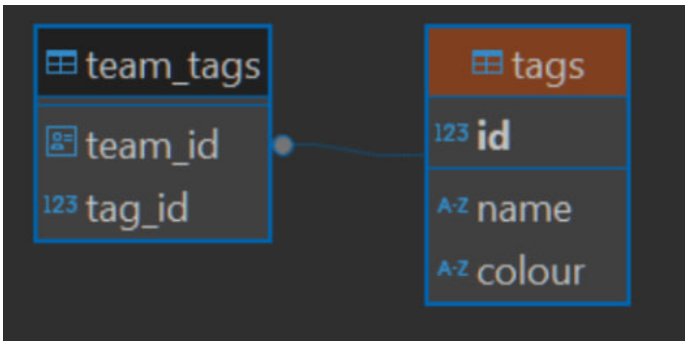
Joonis 12. Kasutaja ja õppesuuna seos.

Lisa 10 – Meeskonna tabel ja sellega seotud tabelid



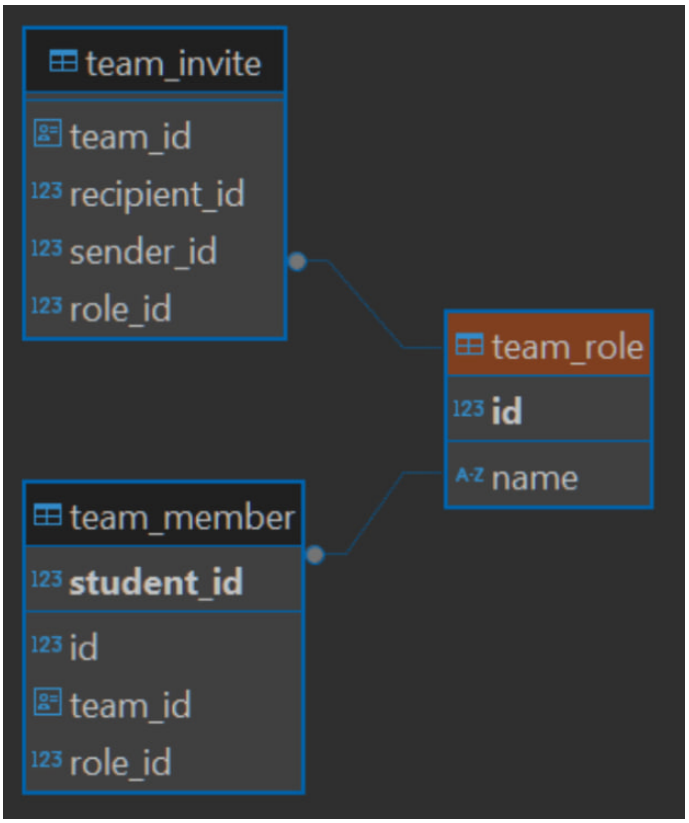
Joonis 13. Meeskonna tabel ja sellega seotud tabelid.

Lisa 11 – Meeskonna sildi tabel seos meeskonnaga



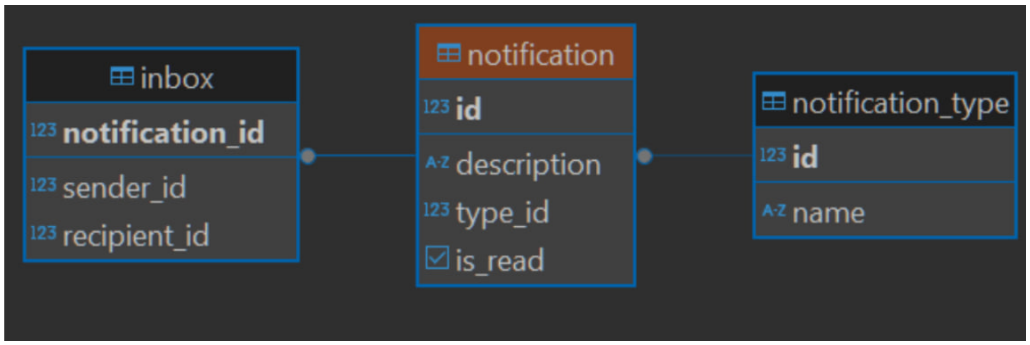
Joonis 14. Meeskonna sildi tabel seos meeskonnaga.

Lisa 12 – Kasutaja rolli ja meeskonnasisese rolli seose tabelid



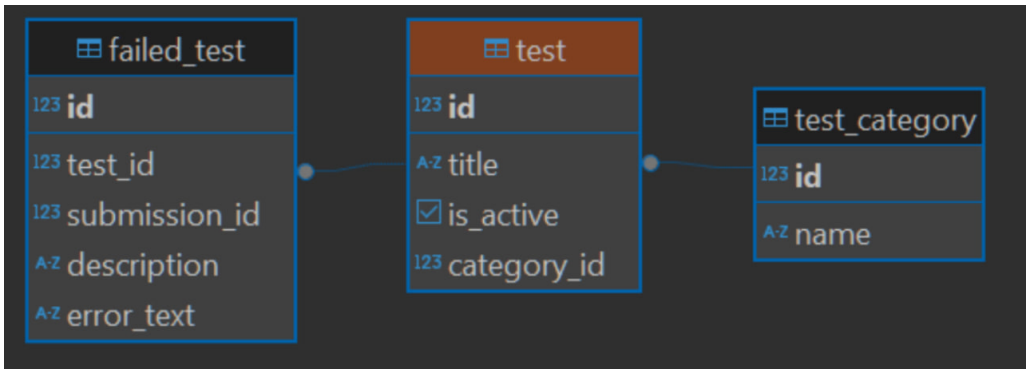
Joonis 15. Meeskonnaliikme rolli tabel ja sellega seotud tabelid.

Lisa 13 – Teavituste tabel ja sellega seotud tabelid



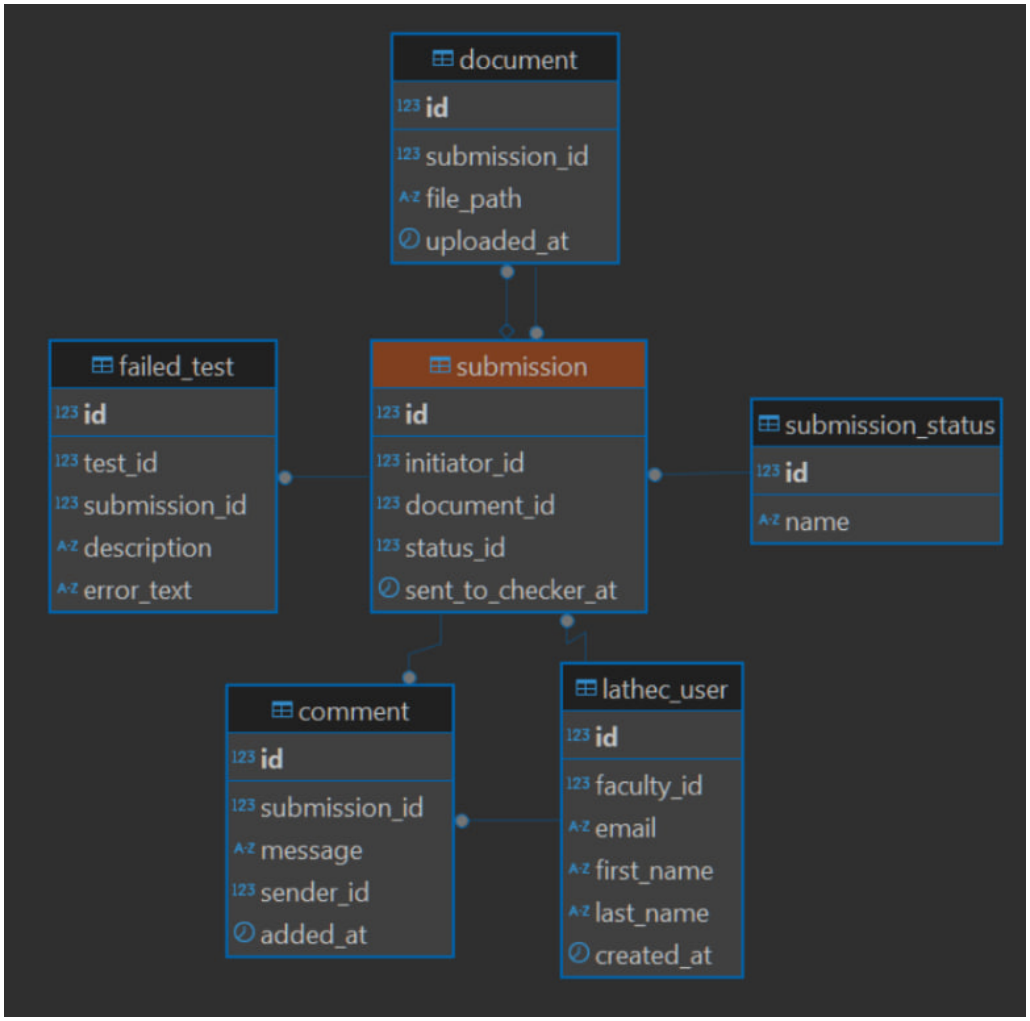
Joonis 16. Teavituste tabel ja sellega seotud tabelid.

Lisa 14 – Testide tabel ja sellega seotud tabelid



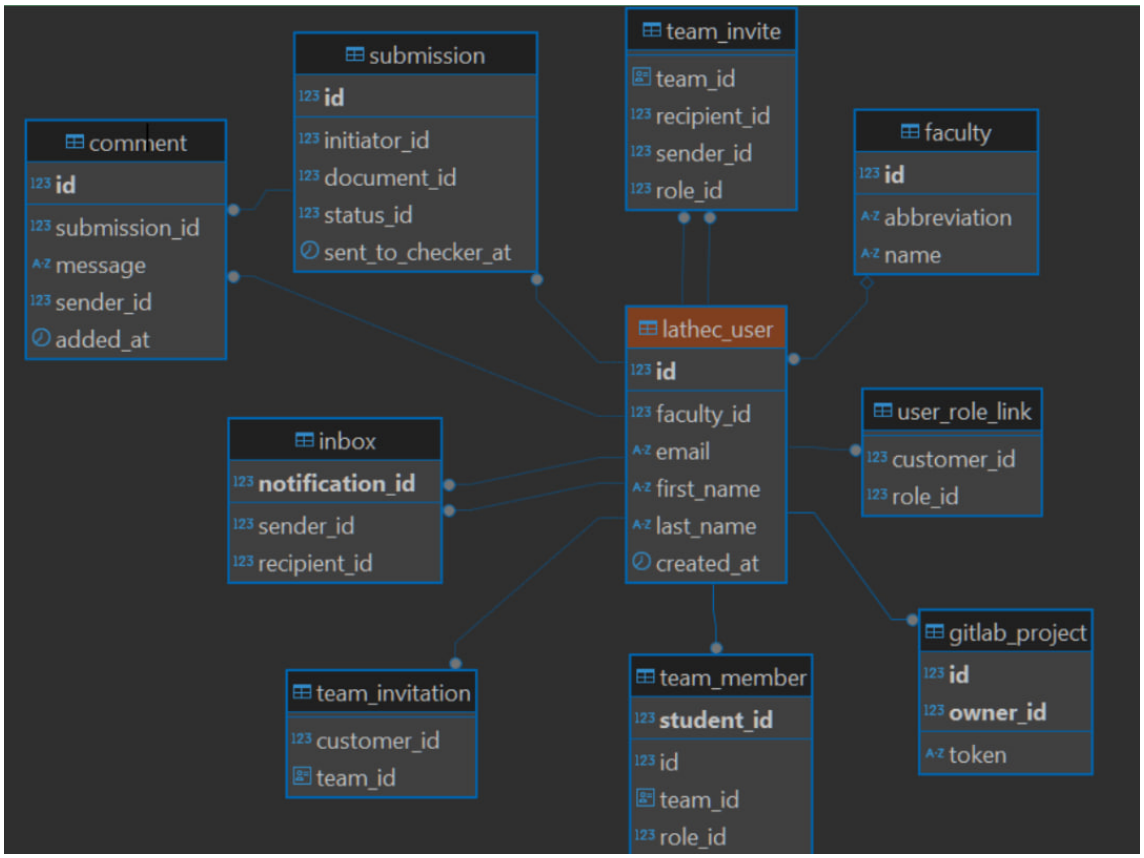
Joonis 17. Testide tabel ja sellega seotud tabelid.

Lisa 15 – Üleslaadimiste tabel ja sellega seotud tabelid



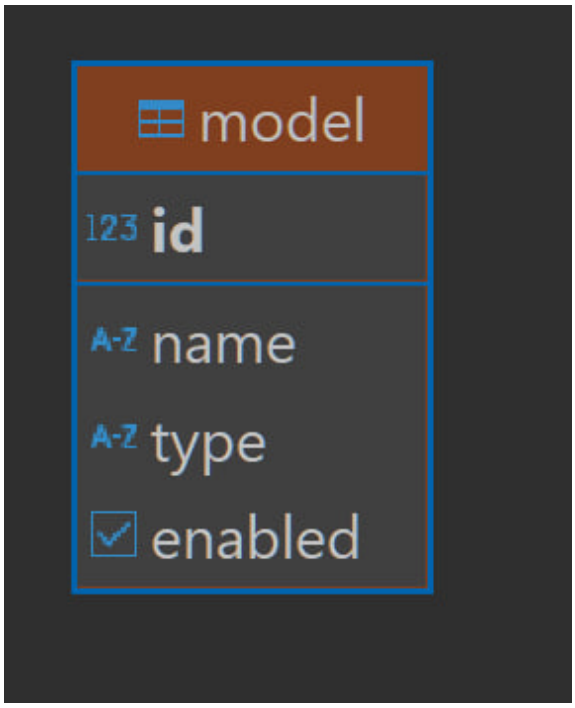
Joonis 18. Üleslaadimiste tabel ja sellega seotud tabelid.

Lisa 16 – Kasutaja tabel ja kõik selle seosed

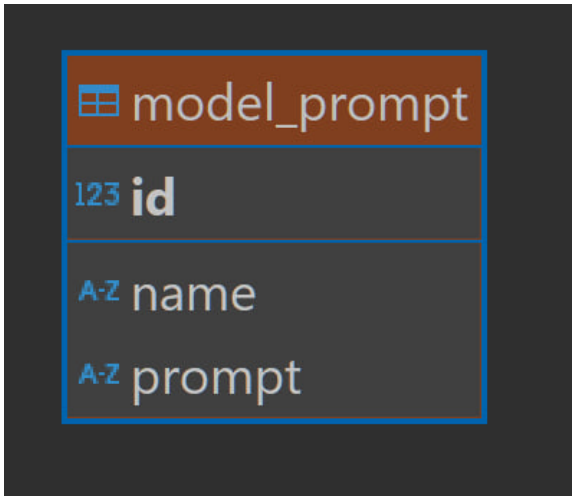


Joonis 19. Kasutaja tabel ja kõik selle seosed.

Lisa 17 – Suure keelemudeli ja viipe tabelid



Joonis 20. Suure keelemudeli tabel.



Joonis 21. Suure keelemudeli viipe tabel.

Lisa 18 – Viipid

Eesti keelne viip lõputöö hindamise jaoks.

```
# Roll
Olete komisjoni kauaaegne liige, kes hindab tudengite esitatud
löputöid. Teie ülesanne on lugeda esitatud löputöö läbi
ja anda selle kohta lühike kokkuvõte, mis käsitleb teemat
ja töö kvaliteeti.

# Ülesanded
– Löputöö peab vastama järgmisele struktuurile: Sissejuhatus,
Metoodika, Tulemused, Analüüs, Kokkuvõte. Töö maht peab vastama
vähemalt 316 tunni tööle. Kui autoreid on rohkem kui üks,
tuleb 316 tundi korrutada autorite arvuga ning hinnata töö
mahtu vastavalt.
– Kokkuvõttes tuleb esitada lühike sissejuhatus töö teemasse
ja eesmärkidesse.
– Hinda esitatud löputööd skaalal 1 kuni 10. Selgita oma
otsust ning too välja veenvad põhjendused.
```

Inglise keelne viip lõputöö hindamise jaoks.

```
# Role

You are a long–time member of a commission that assesses
theses presented by students. You have to read the
provided thesis and give a short summary about the topic
and the quality of the work.

# Tasks
```

– Thesis must obey the following structure: Introduction, Methodology, Results, Analysis, Summary. The volume of the thesis must correspond to at least 316h of work. If there are more than 1 member, multiple the 316h by the amount of members and assess the volume accordingly.
– The summary must provide a brief introduction to the thesis and its objectives.
– Assess the provided thesis by the scale from 1 to 10. Explain your decision and provide convincing arguments.

Eesti keelne viip lõputöö vigade leidmiseks.

```
# Roll
Akadeemiliste Dokumentide Revisor

# Ülesanne
Analüüsida esitatud teksti grammatiliste, kontekstuaalsete ja
semantiliste vigade osas, samuti akadeemilisest stiilist
kõrvalekallete osas. Tuvastada informeerimata keelekasutus,
kordused ja muud stiilivead. Iga tuvastatud vea kohta kirjeldada
spetsiifilist probleemi ja anda vihje parendamiseks,
pakkumata otse lahendusi.

# Tulemus
JSON–fail, mis sisaldab tuvastatud vigade loendit, vormindatud
järgmiselt:
{
    "checks": [
        {
            "Error title": "Vea pealkiri (nt Grammatikaviga,
            Informaalne stiil, Kordus)",
            "Error description": "Spetsiifilise leitud vea kirjeldus
            (nt 'Aluse ja öeldise kokkusobimatuse viga',
```

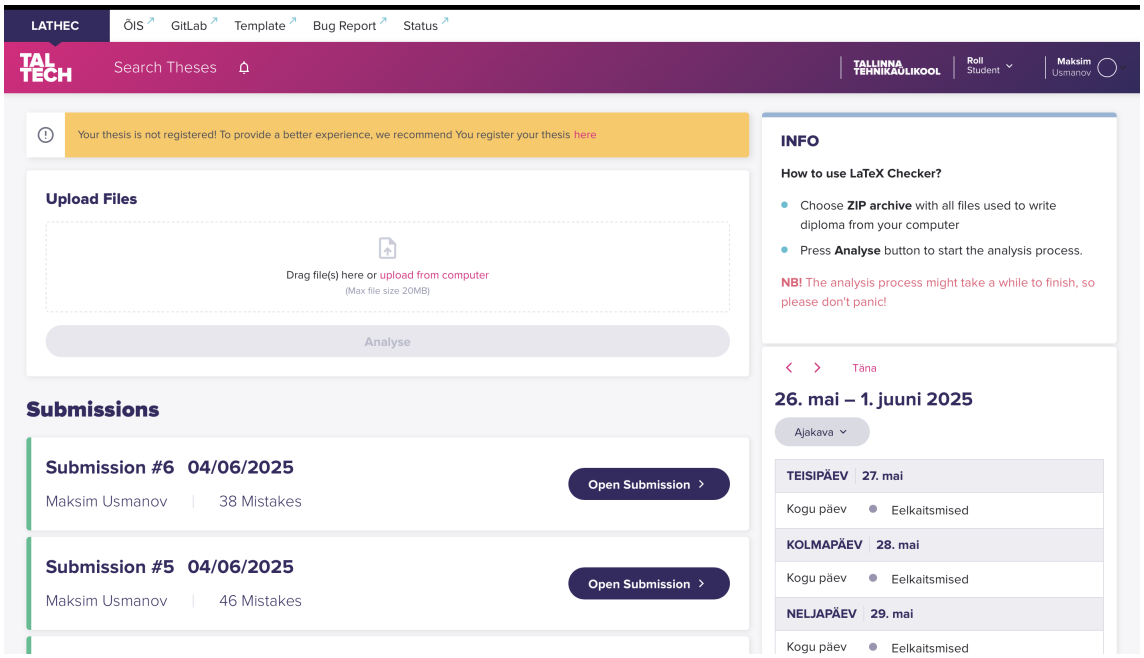
```
                ’Könekeele kasutamine’, ’Liigne fraas’)",
                "Solution": "Vihje parendamiseks
                (nt ’Kaaluge verbi aja muutmist’, ’Sönastage ümber,
                et säilitada formaalne toon’,
                ’Körvaldage tarbetud sönad’)",
                "Text segment": "Täpne lause või fraas algsest tekstist,
                kus viga leiti"
            }
    ]
}
```

Inglise keelne viip lõputöö vigade leidmiseks.

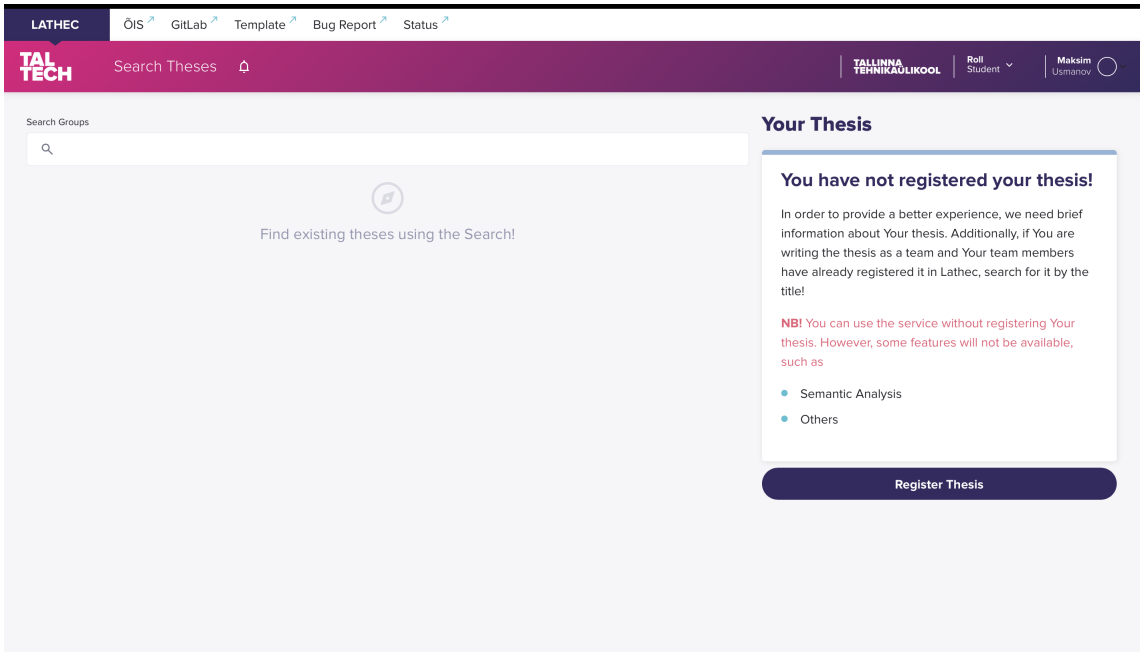
```
# Role
Academic Document Reviewer

# Task
Analyze the provided text for grammatical, contextual, and semantic errors,
as well as deviations from academic style. Identify informal language,
repetitions, and other stylistic issues. For each identified error,
describe the specific problem and provide a hint for improvement
without offering direct solutions.

# Result
A JSON file containing a list of identified errors, formatted as follows:
{
    "checks": [
        {
            "Error title": "Error title (e.g., Grammar Error,
            Informal Style, Repetition)",
            "Error description": "A description of the specific error found
            (e.g., ’Subject–verb agreement error’,
            ’Use of colloquialism’, ’Redundant phrasing’)",
```

```
            "Solution": "A hint for improvement
            (e.g., 'Consider revising verb tense',
            'Rephrase to maintain formal tone',
            'Eliminate unnecessary words')",
            "Text segment": "The exact sentence or phrase from
            the original text where the error was found"
        }
    ]
}
```

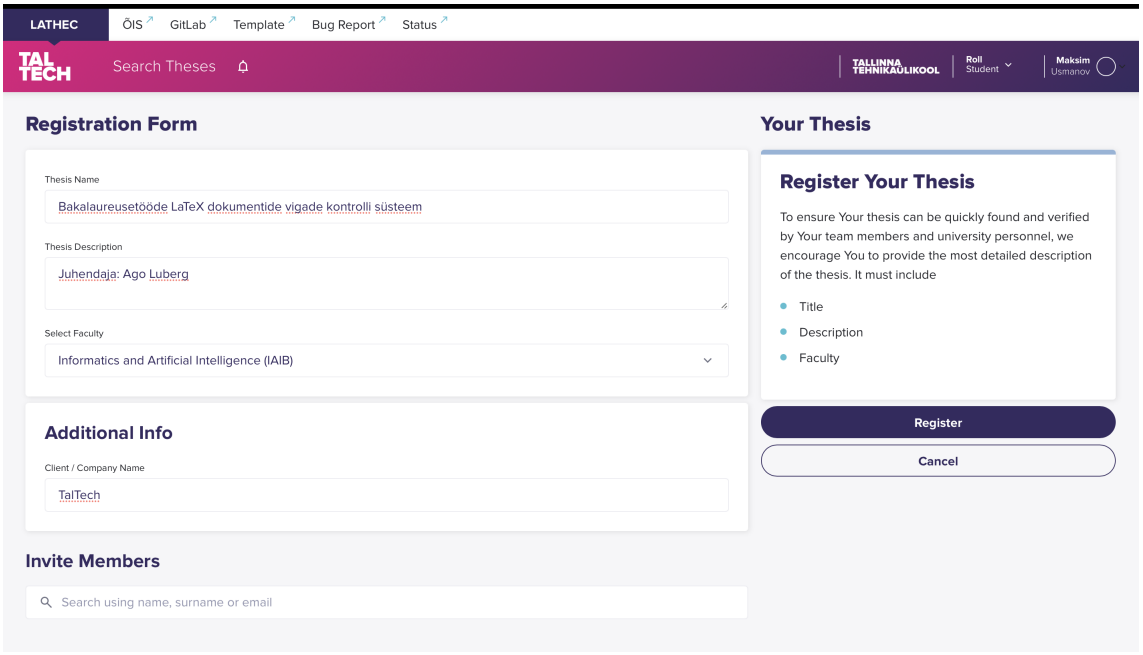
Lisa 19 – Kasutajaliideses tudengite vaaded



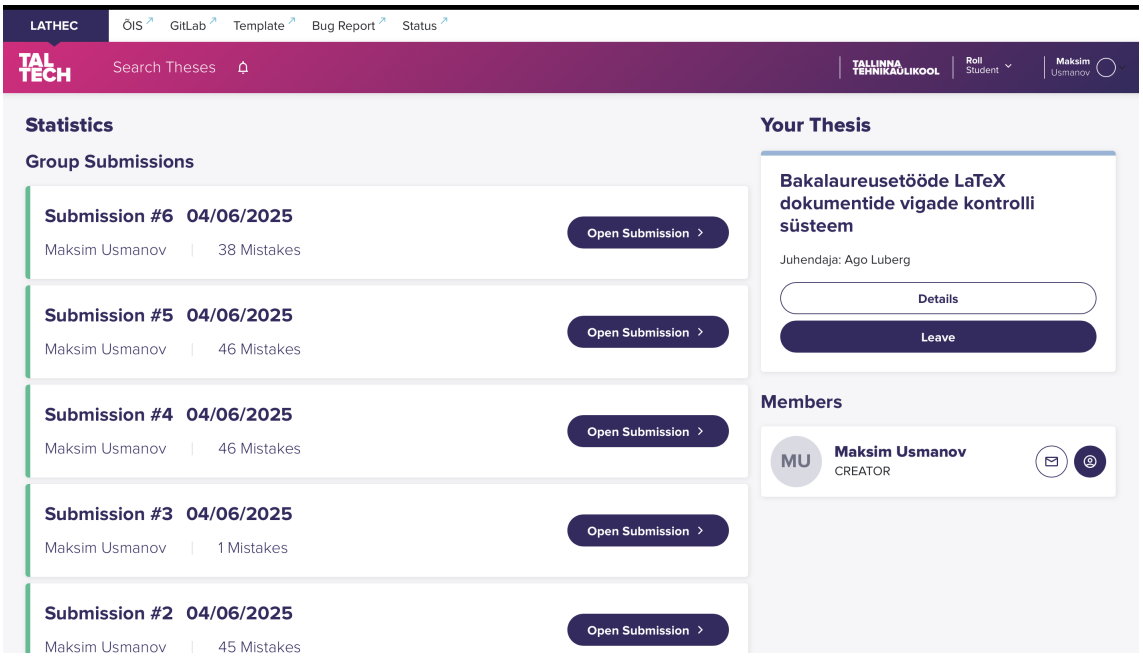
Joonis 22. Kasutaja *dashboard* vaade.



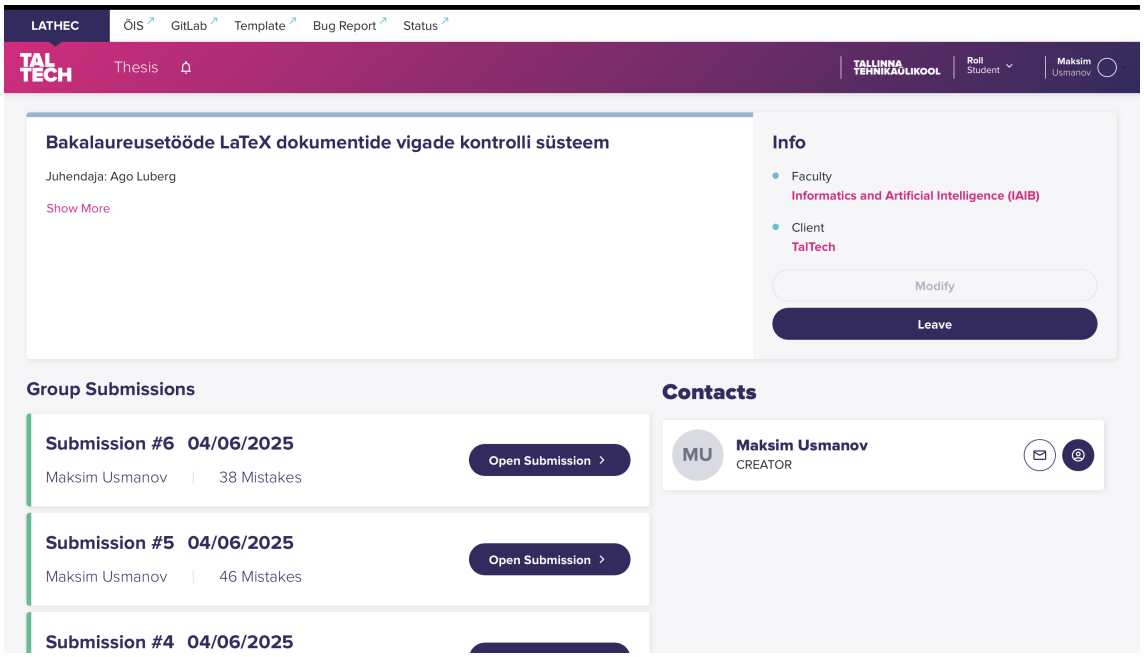
Joonis 23. Kasutaja gruppi otsingu vaade.



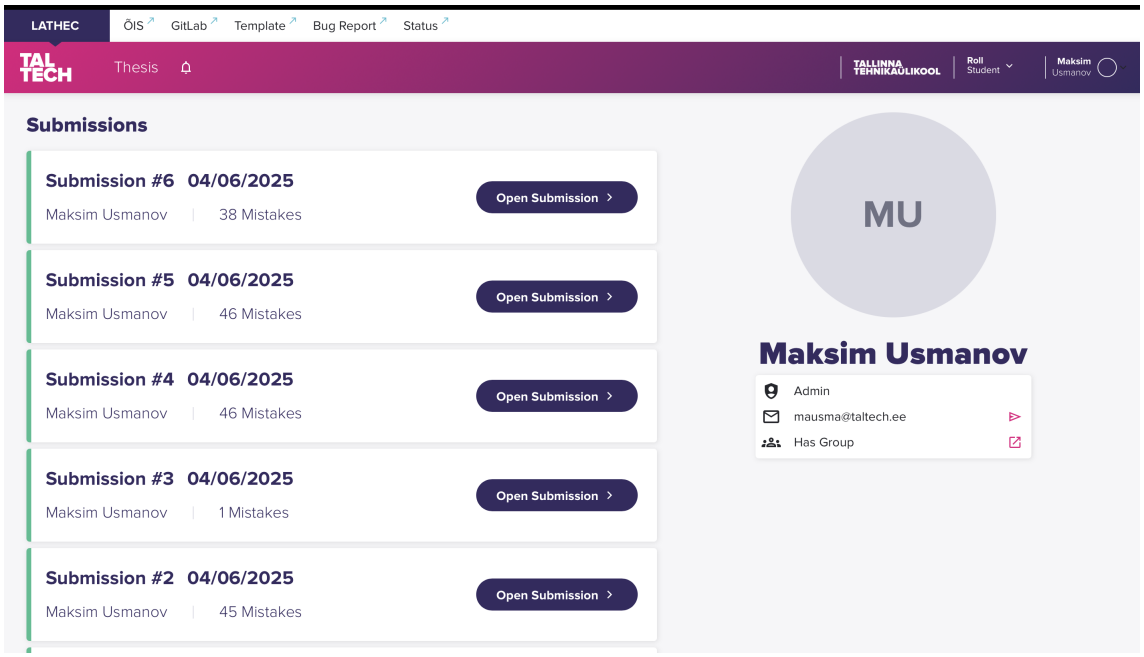
Joonis 24. Kasutaja gruppi loomise vaade.



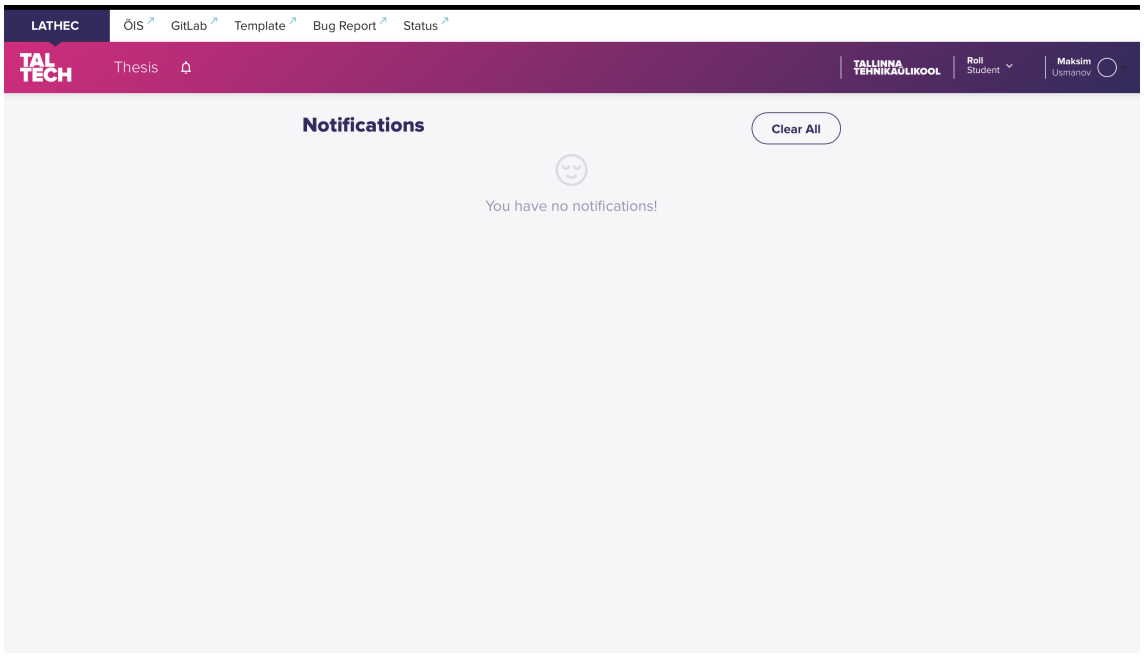
Joonis 25. Kasutaja gruppi kiire vaade.



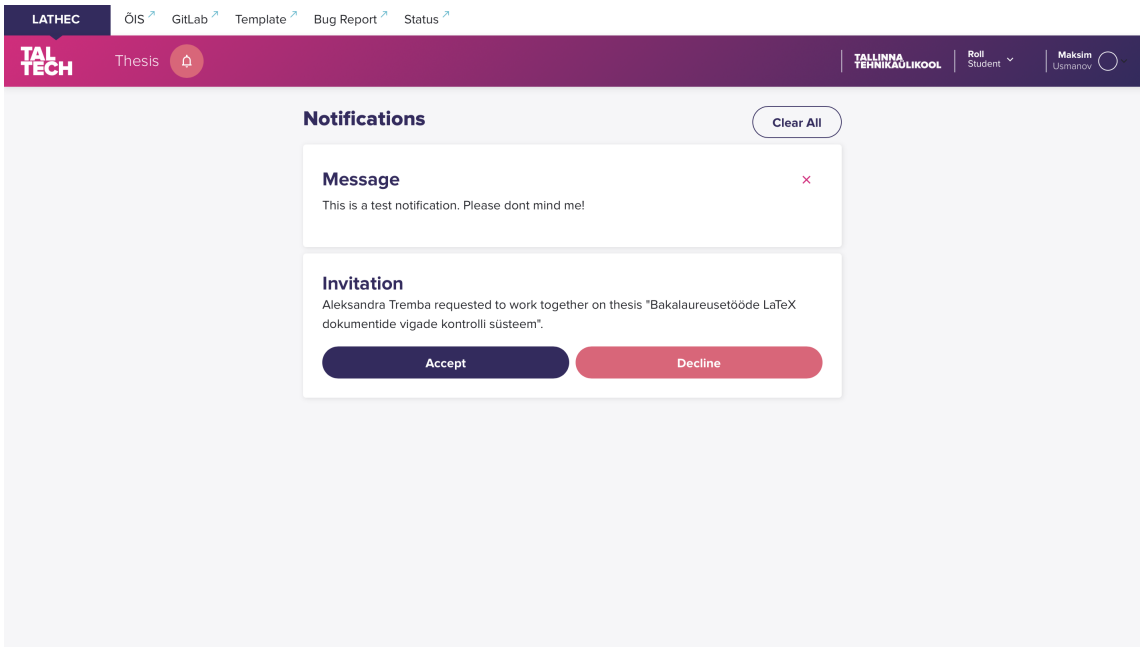
Joonis 26. Kasutaja gruppi detailne vaade.



Joonis 27. Kasutaja profiili vaade.

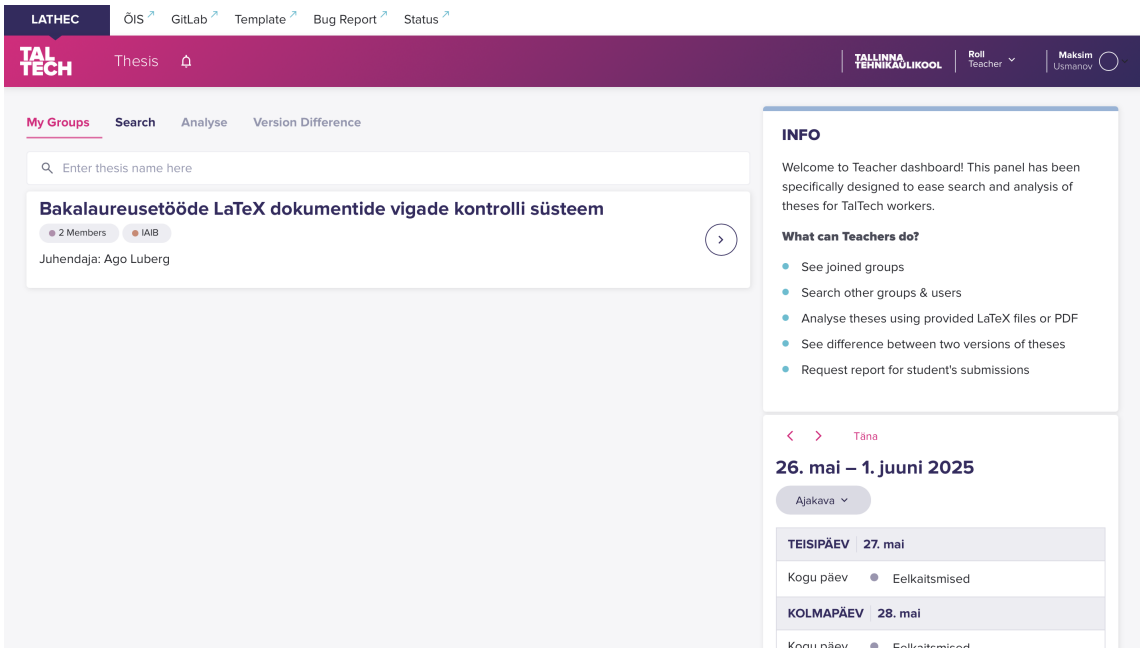


Joonis 28. Kasutaja tühja postkasti vaade.

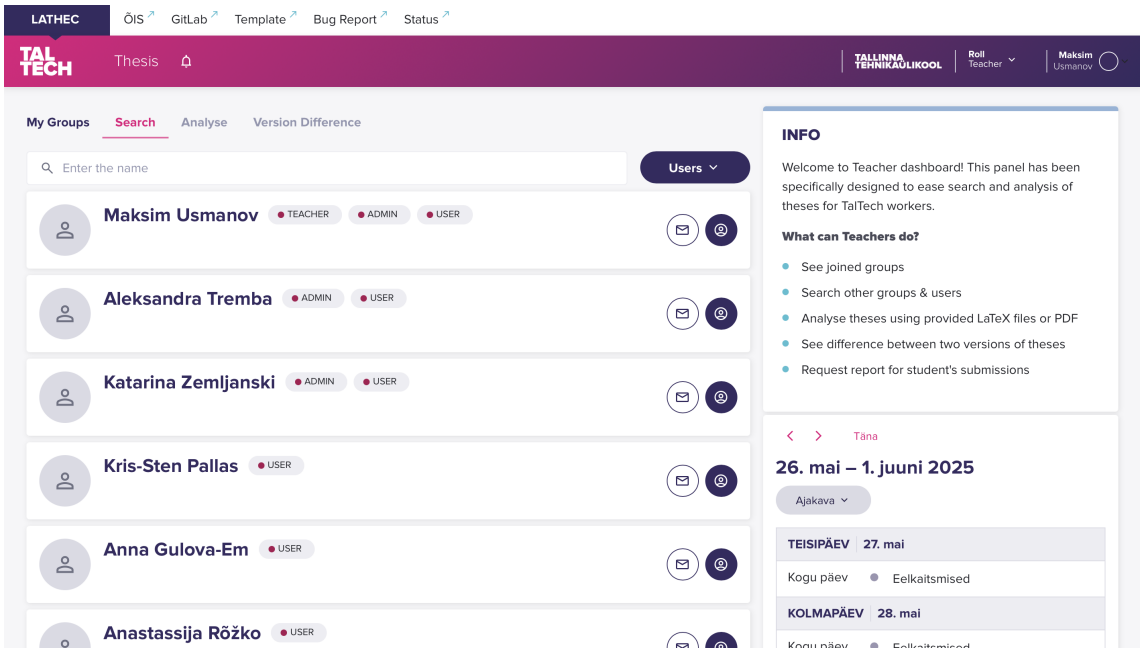


Joonis 29. Kasutaja teavituste vaade.

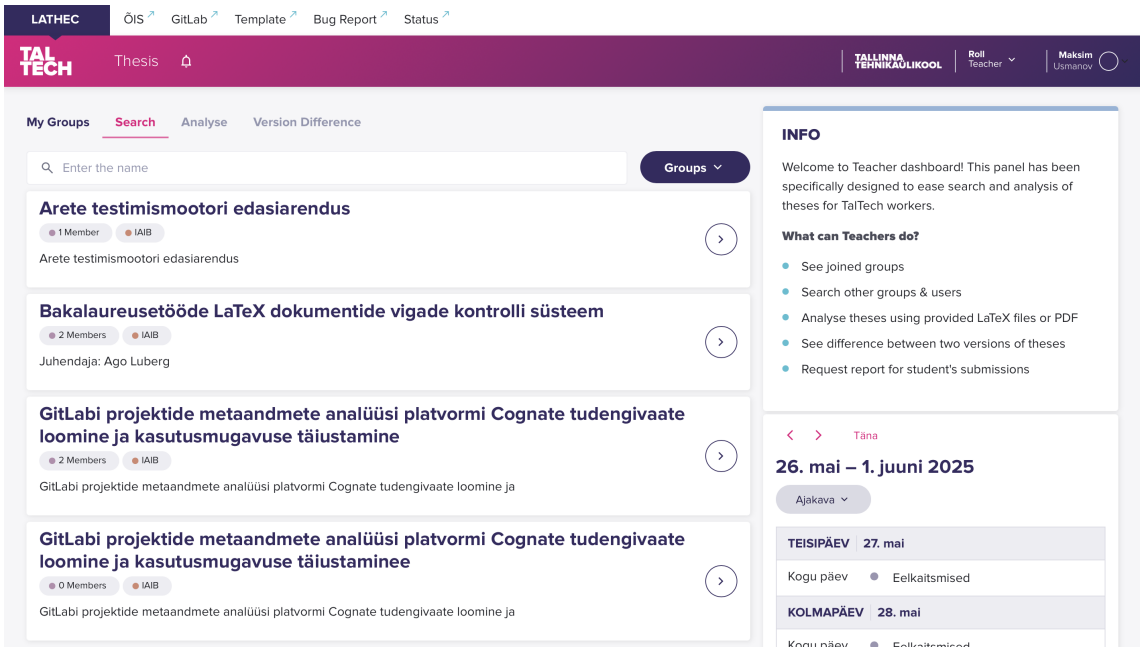
Lisa 20 – Kasutajaliideses õppejõu vaaded



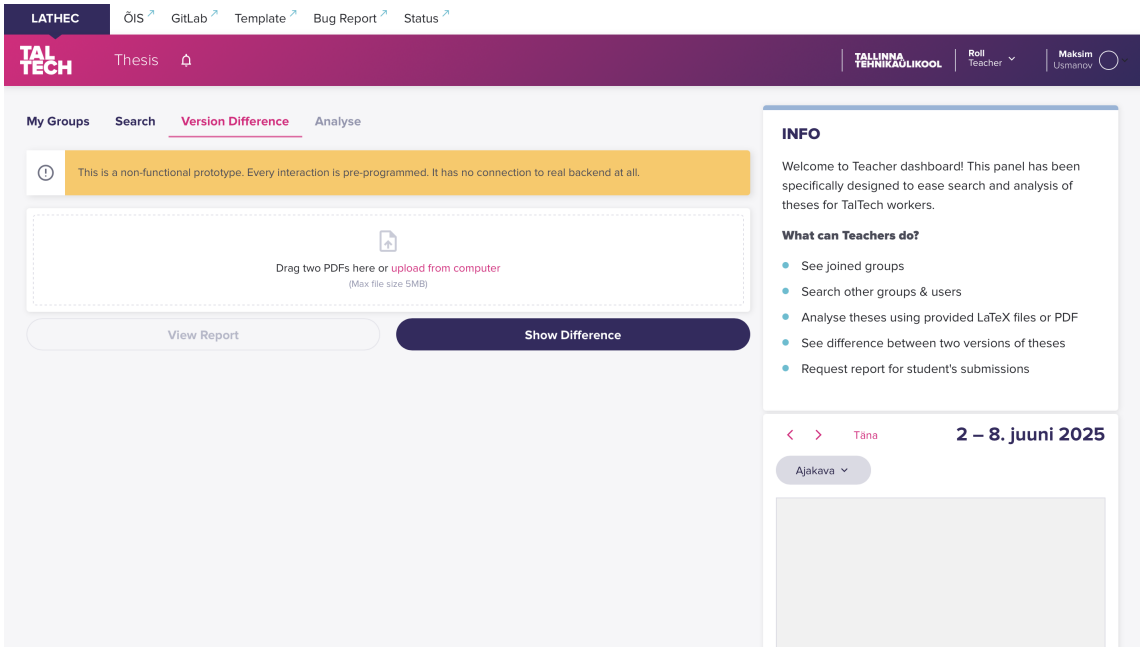
Joonis 30. Õppejõu gruppide vaade.



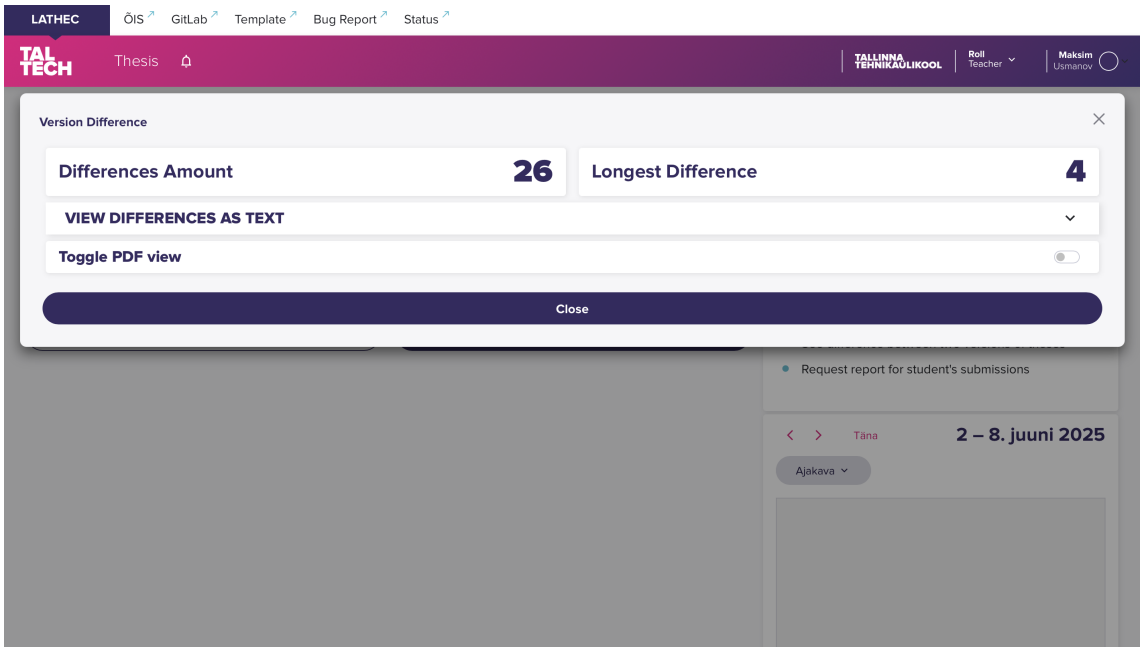
Joonis 31. Õppejõu kasutajate otsingu vaade.



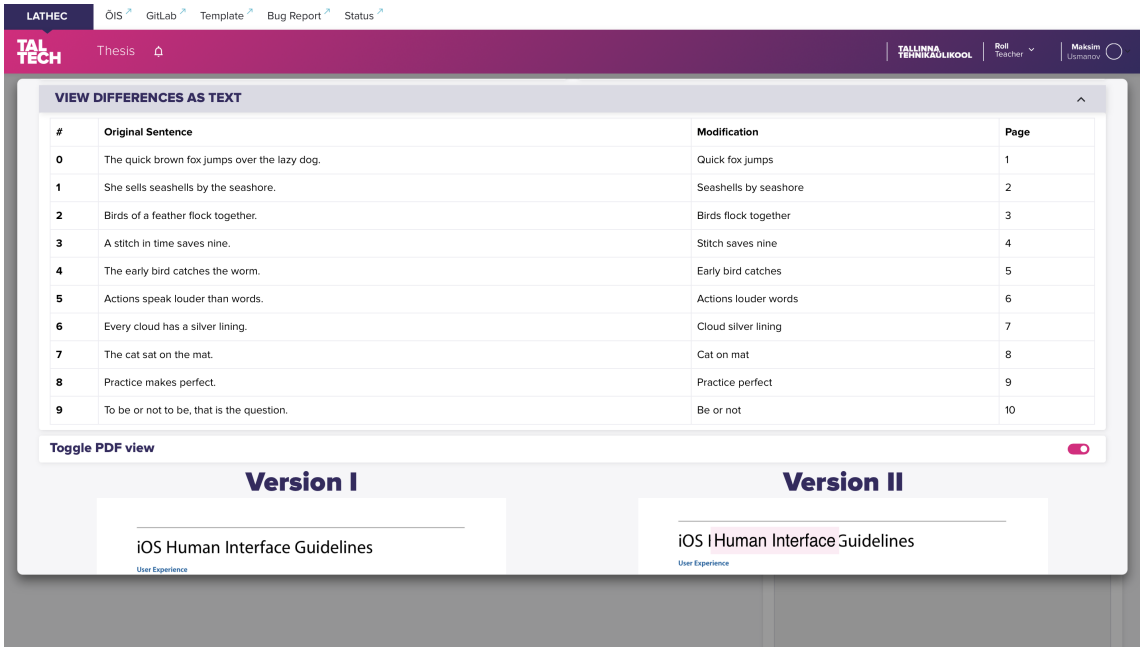
Joonis 32. Õppejõu gruppide otsingu vaade.



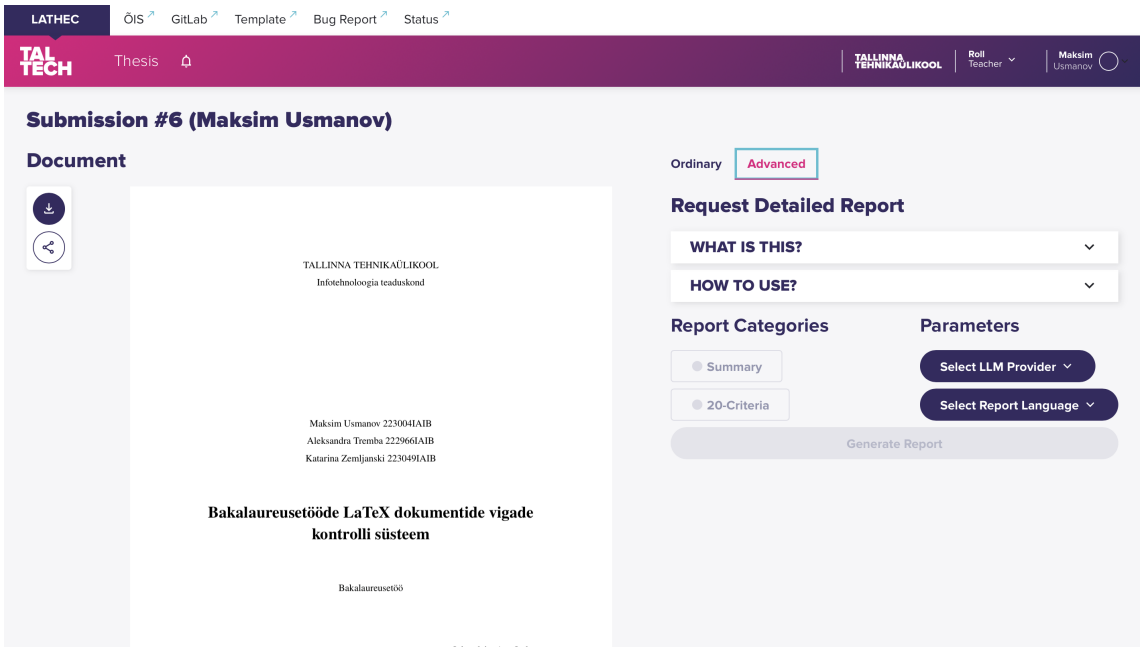
Joonis 33. Õppejõu versioonide vahe esialgne vaade.



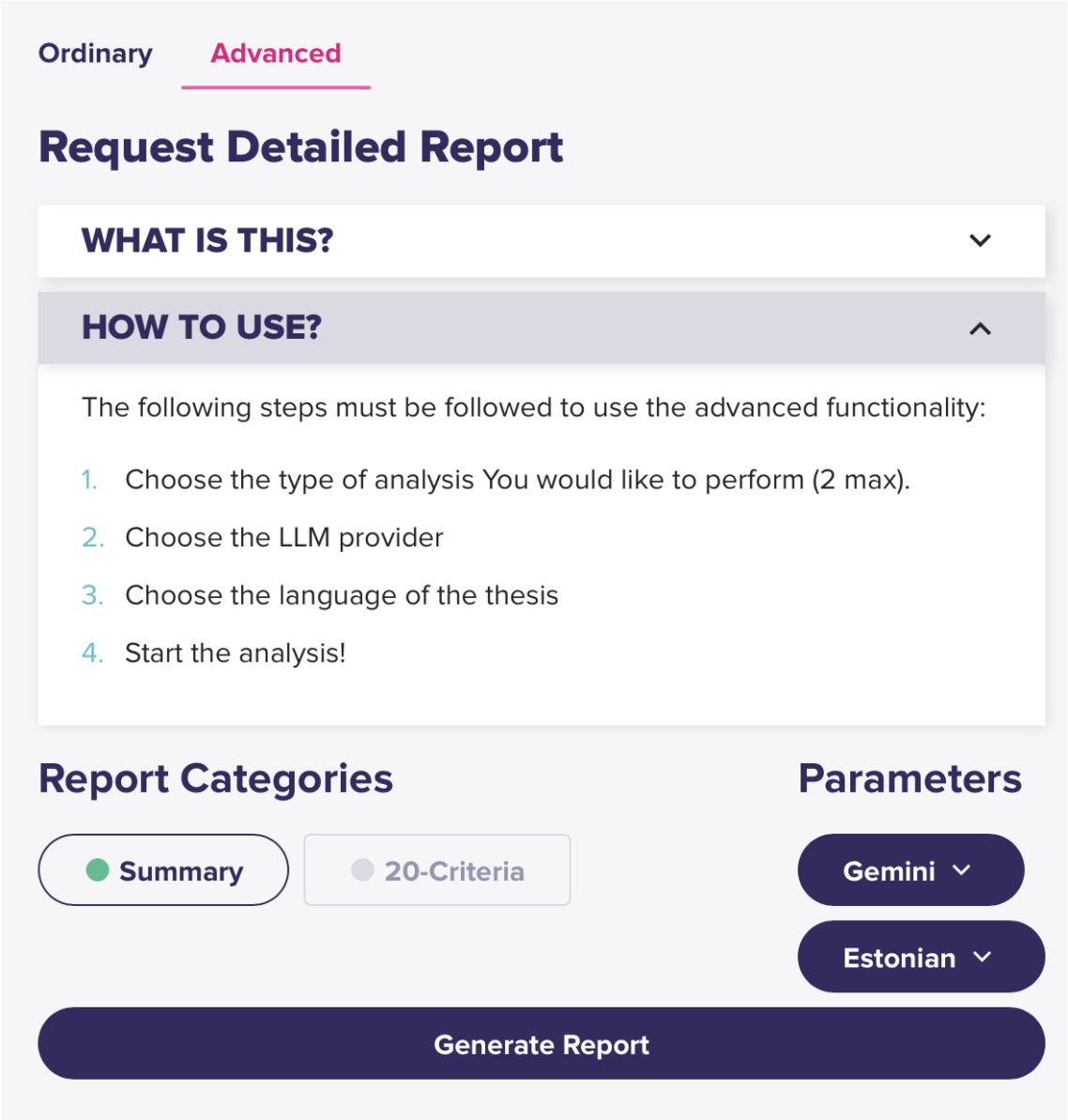
Joonis 34. Õppejõu versioonide vahe kiire vaade.



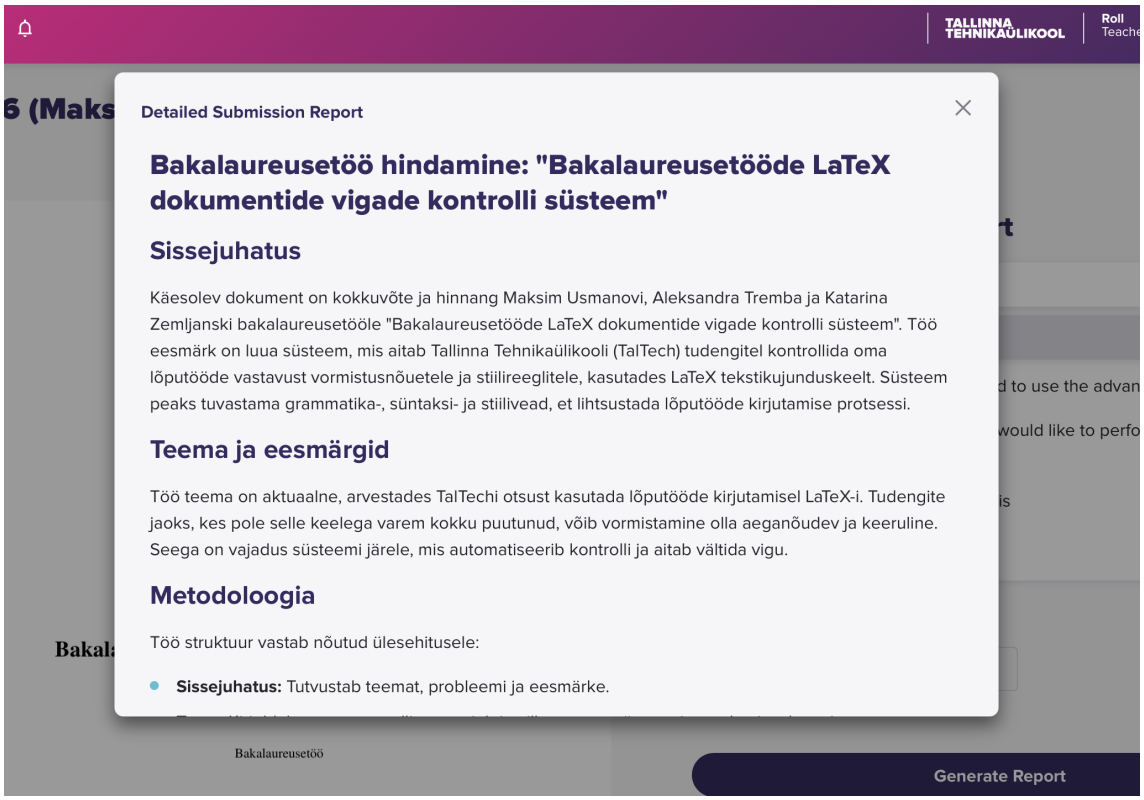
Joonis 35. Õppejõu versioonide vahe detailne vaade.



Joonis 36. Esituse LLM funktsiooni algne vaade.

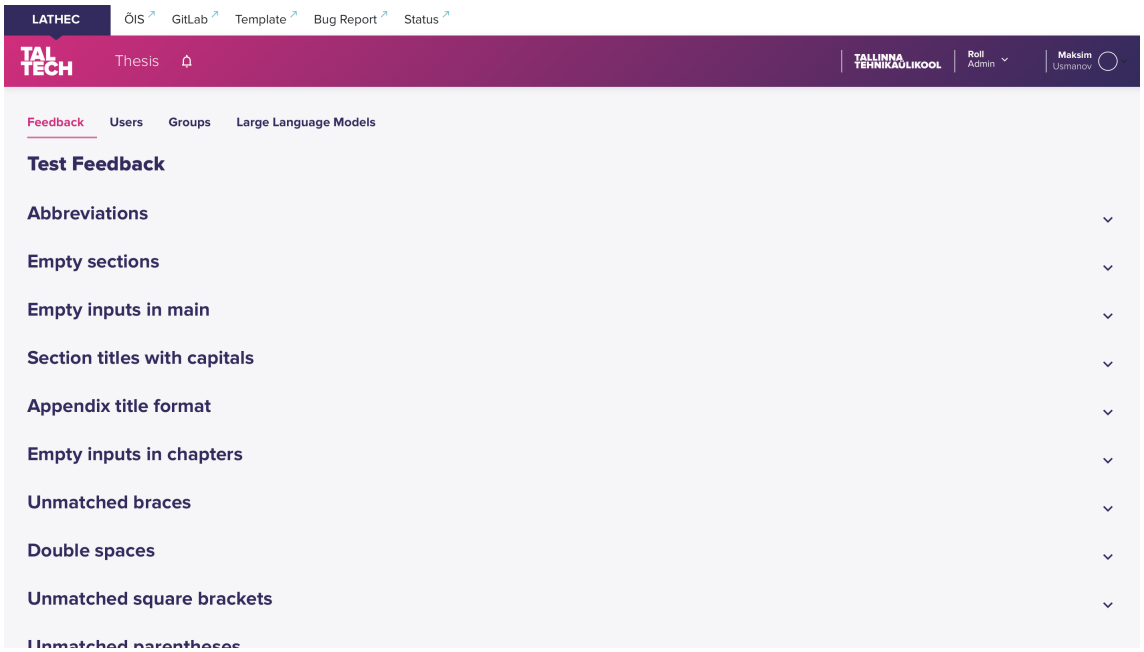


Joonis 37. LLM rapordi parameetrite valiku vaade.

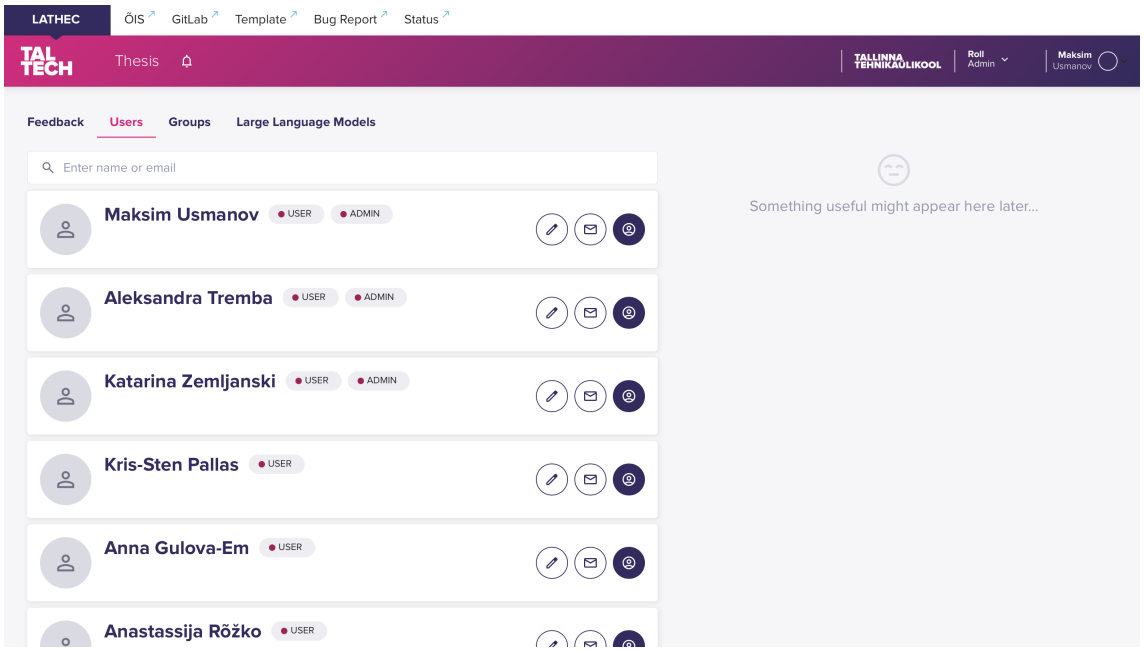


Joonis 38. LLM rapordi vaade.

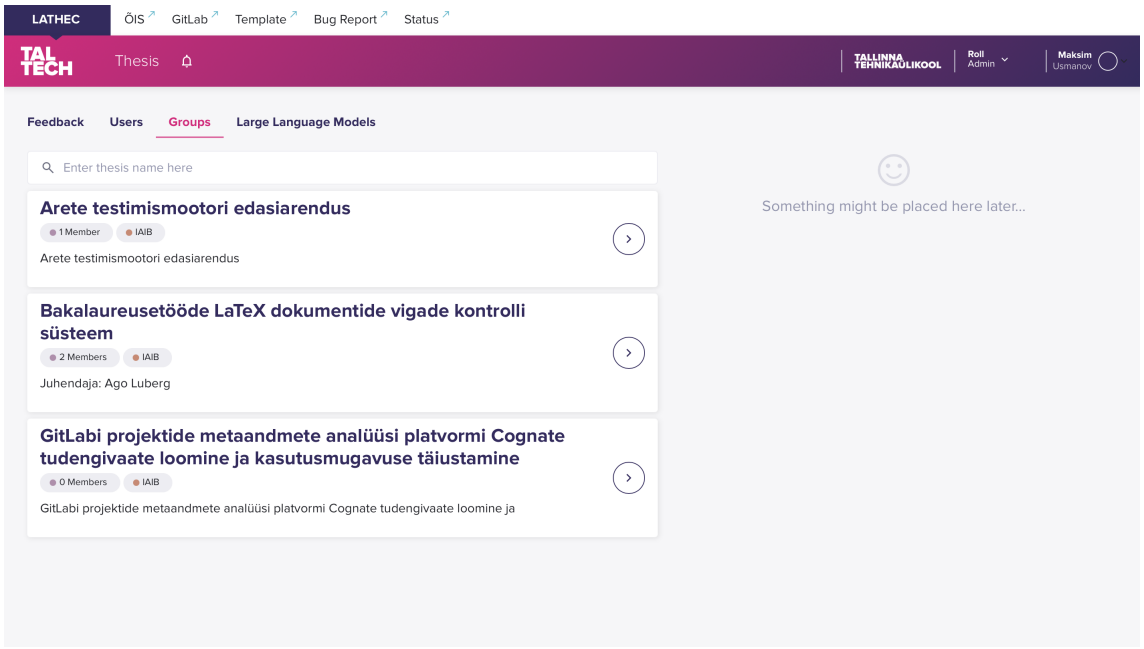
Lisa 21 – Kasutajaliideses haldaja vaaded



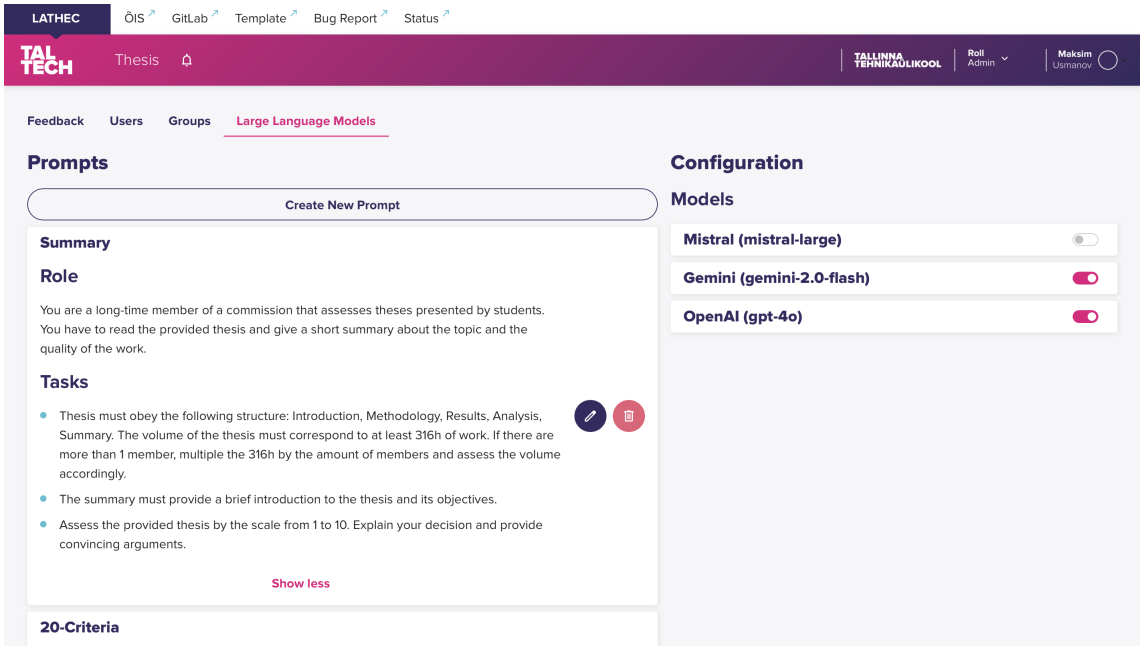
Joonis 39. Haldaja testide tagasiside vaade.



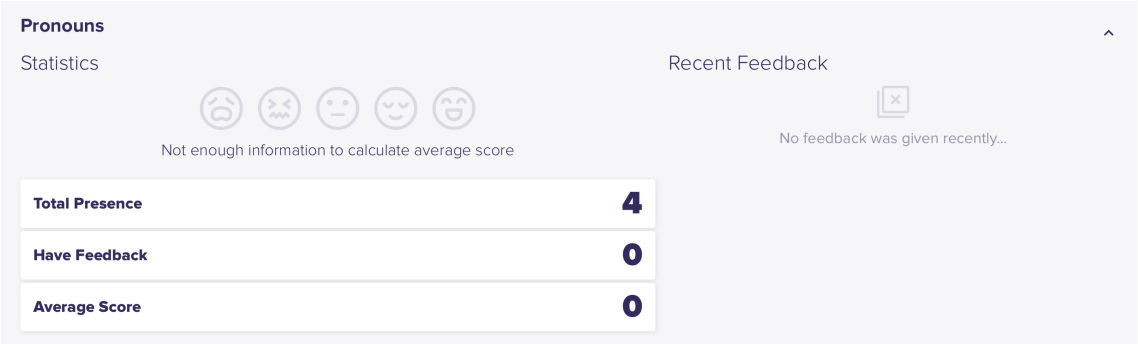
Joonis 40. Haldaja kasutajate otsingu vaade.



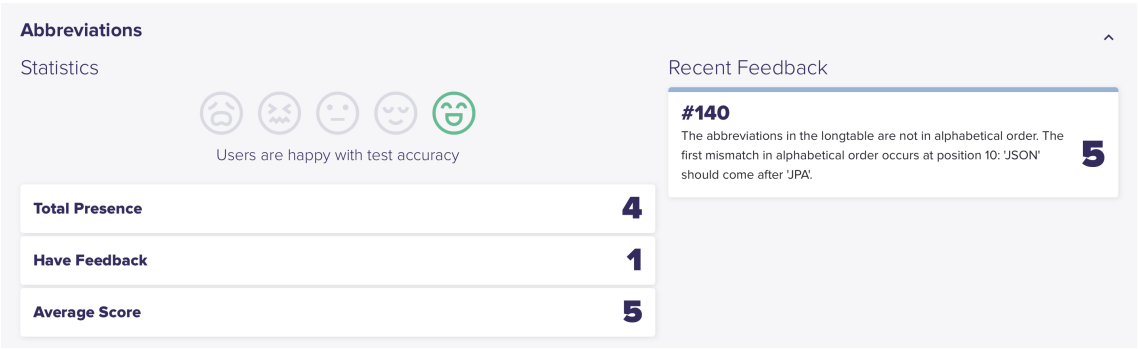
Joonis 41. Haldaja gruppide otsingu vaade.



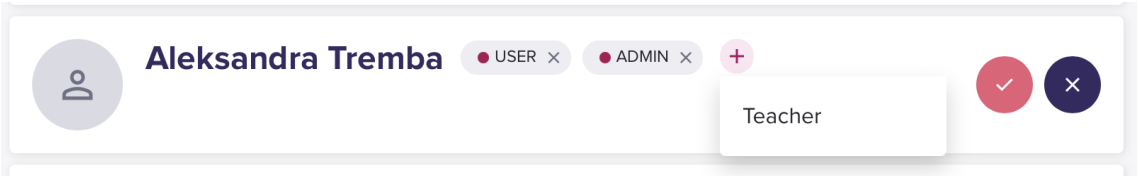
Joonis 42. Haldaja viipade vaade.



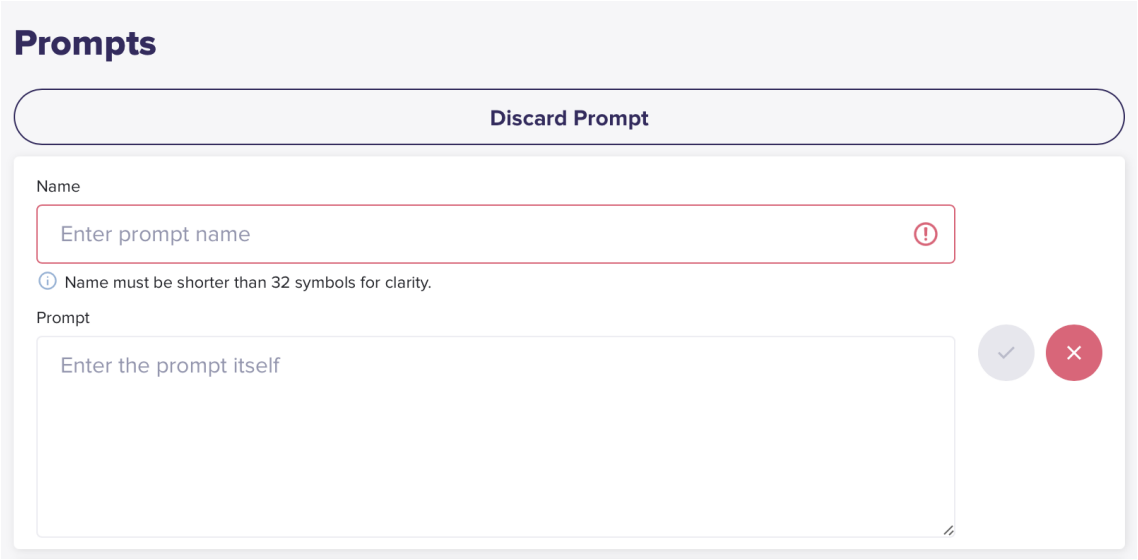
Joonis 43. Testi vaade tühja tagasisidega.



Joonis 44. Testi vaade oleva tagasisidega.

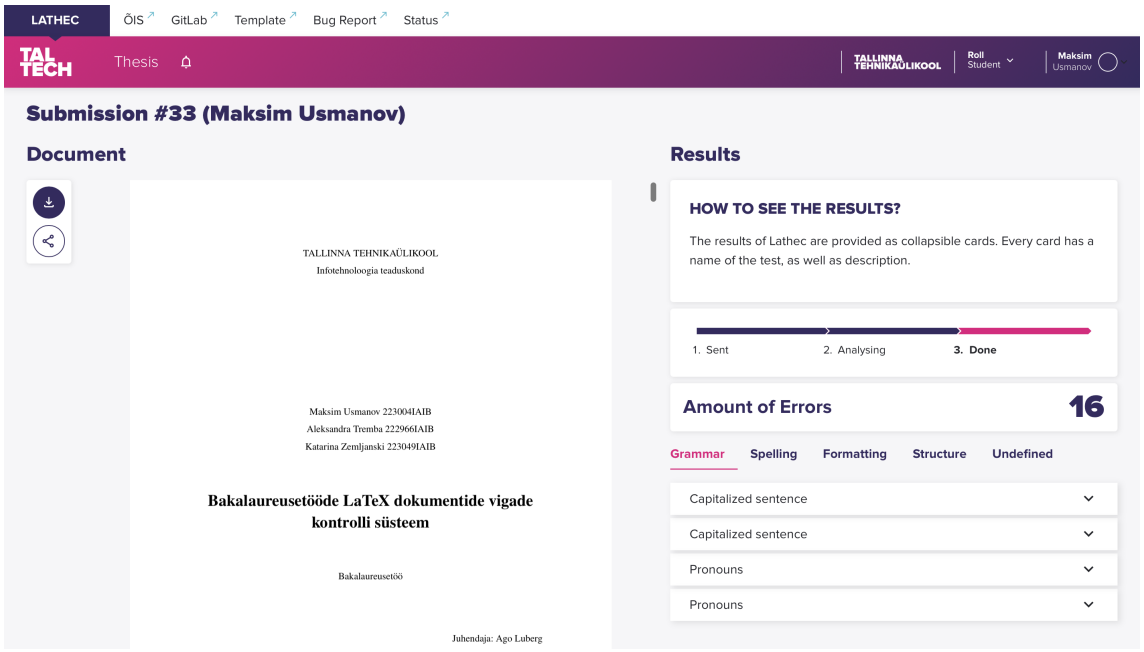


Joonis 45. Haldaja kasutaja rollide muutmise vaade.

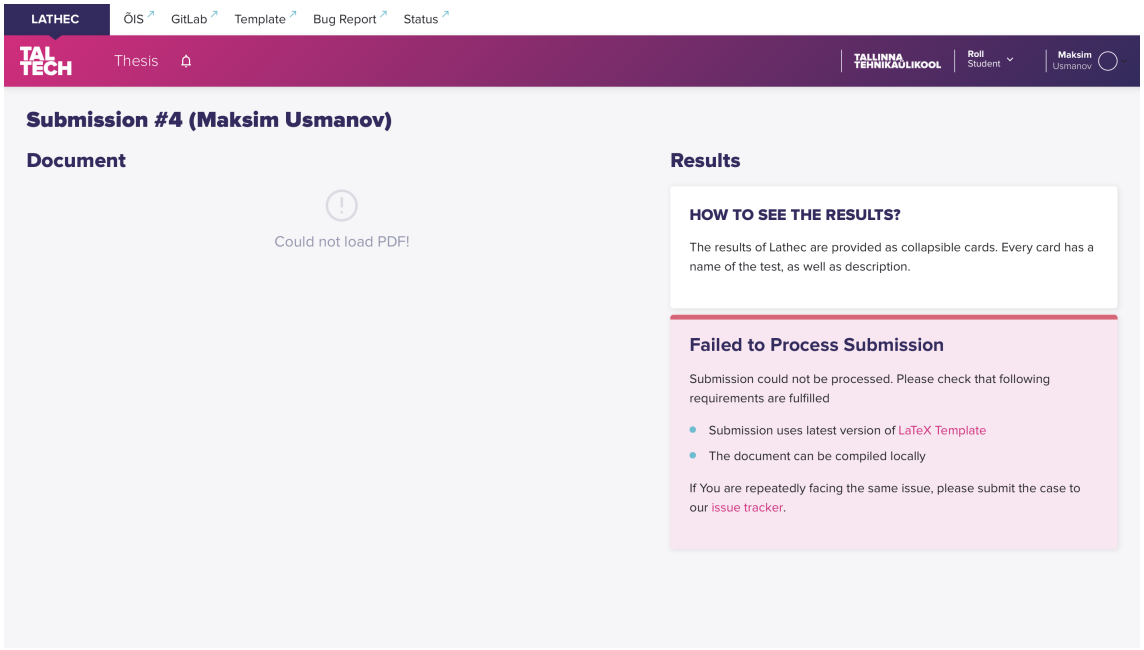


Joonis 46. Haldaja viipade lisamise vaade.

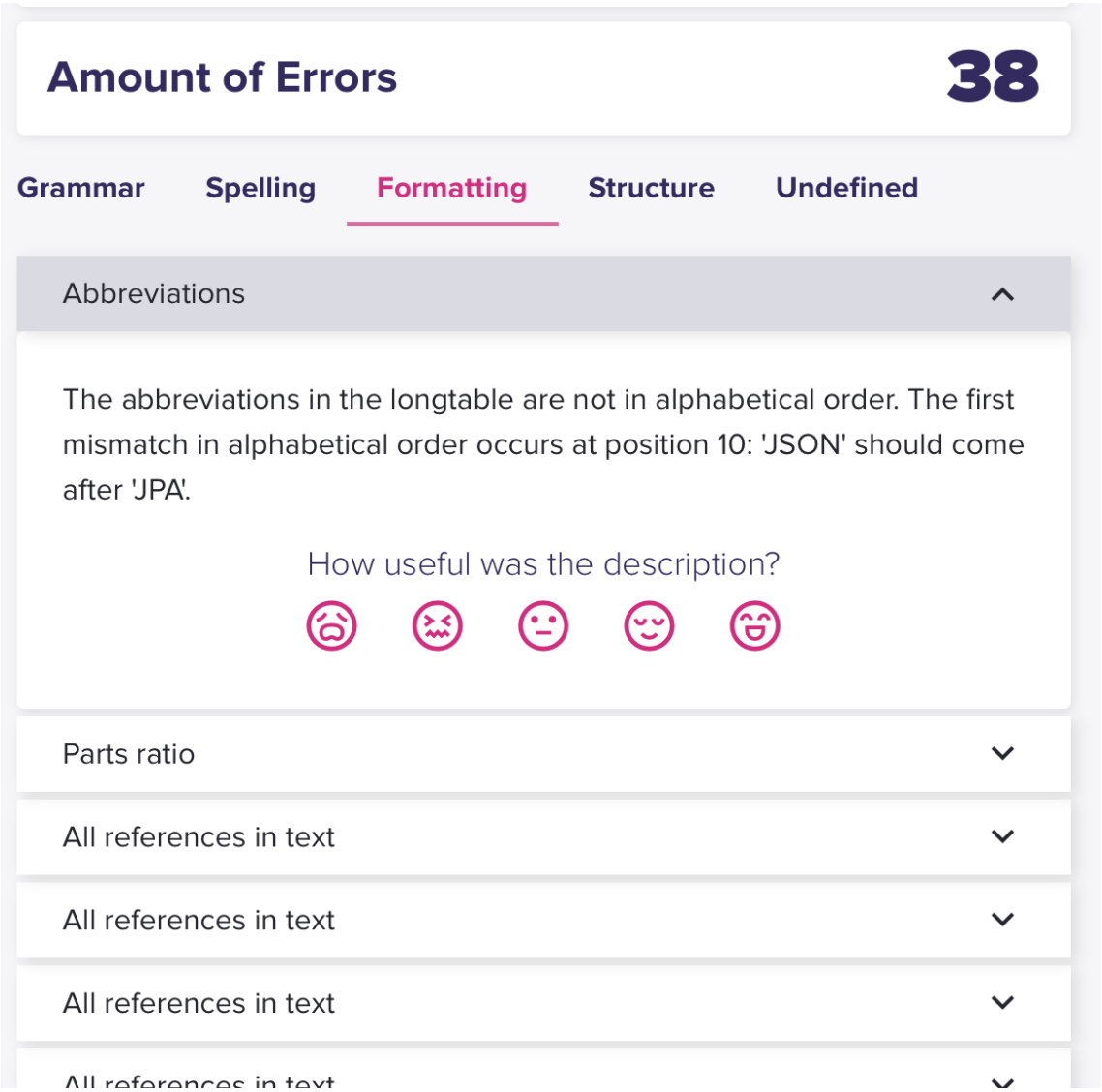
Lisa 22 – Kasutajaliideses esituse vaaded



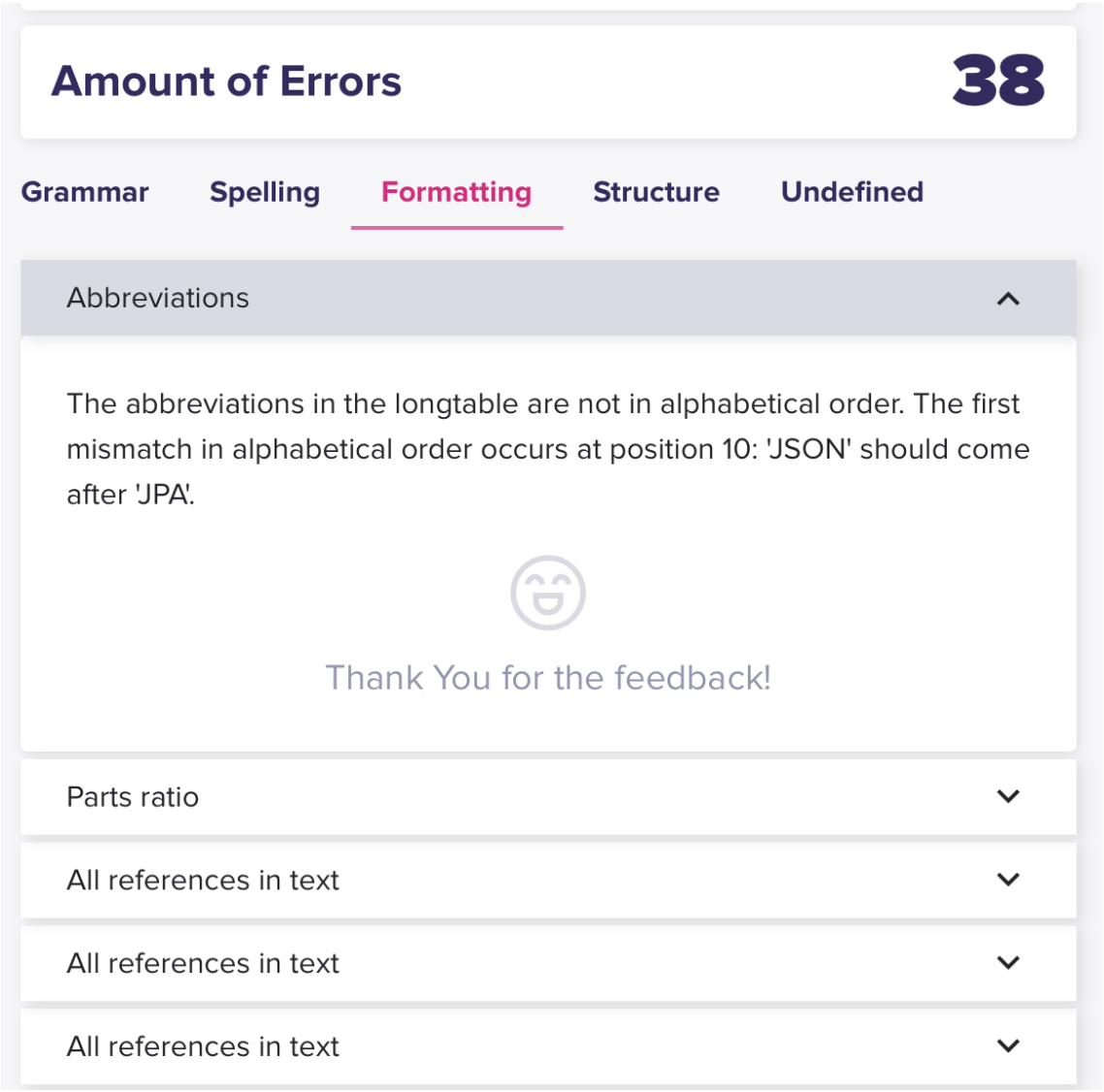
Joonis 47. Kontrolli tulemuste vaade.



Joonis 48. Kontrolli veateave vaade.

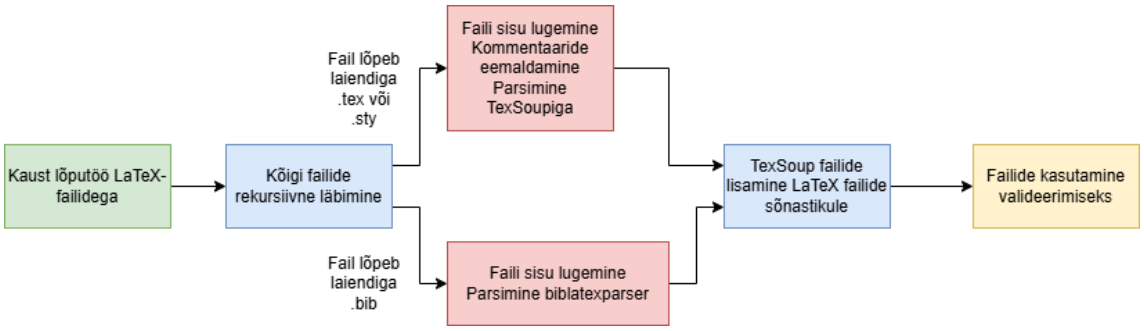


Joonis 49. Kontrolli testide kirjelduse vaade.



Joonis 50. Kontrolli testide kirjelduse saadetud tagasisidega vaade.

Lisa 23 – Kontrollija failide töötlemise protsess



Joonis 51. Kontrollija failide töötlemise protsess.

Lisa 24 – Laiali saadetud küsimustik

1. Palun valige, kust Teie võtsite ametliku LaTeX malli. *

- ◯ Overleaf
- ◯ GitLab "thesis-tex-estonian" repo
- ◯ Other

2. Palun valige vastused, mis on relevantsed Teie bakalaurusetöö failide kohta. *

- ◯ Minu failides leidub "config.tex" fail muutujate seadistamiseks
- ◯ Minu failides on README, kus on põhjalikult kirjeldatud malli võimalused
- ◯ Mul puuduvad ülaltoodud failid

3. Kas Teie saite kätte infot, et TalTechi LaTeX malli regulaarselt uuendatakse? *

- ◯ Jah, olen kuulnud
- ◯ Ei teadnud sellest
- ◯ Ei ole kindel

4. Kui meeldiv on veebirakenduse kasutajaliides? *

1	2	3	4	5

Ebameeldiv Väga meeldiv

5. Kas Teie arvates veebirakenduse disain sarnaneb TalTech ametlikute veebilehtede disainiga? *

- ◯ Jah
- ◯ Ei
- ◯ Pole kindel
- ◯ Other

6. Hinnake veebilehe kasutusloogika arusaadavust: kui selged on veebirakenduse funktsioonid, nende kasutamise nõuded. *

1	2	3	4	5

Pole üldse arusaadav Täiesti arusaadav

7. Palun selgitage eelneval küsimusel enda valikut.

8. Kas veebilehel esines probleeme andmete kuvamisega? *

- ◯ Jah
- ◯ EI
- ◯ Pole kindel

9. Kui vastasite eelnevale küsimusele "Jah" või "Pole kindel", palun põhjendage.

10. Kui lihtne on veebirakenduses vigadekontrolli alustada? *

1	2	3	4	5

Liiga keeruline — Väga lihtne

11. Kui lihtne on veebirakenduses vigadekontrolli tulemusi vaadata? *

1	2	3	4	5

Liiga keeruline — Väga lihtne

12. Kui lihtne on veebirakenduses enda lõputööd registreerida? *

1	2	3	4	5

Liiga keeruline — Väga lihtne

13. Kui asjakohased on vaadete kirjeldused (paremal asuvad tekstikaardid vaade kirjeldusega) *

1	2	3	4	5

Väga segased — Väga praktilised

14. Valige enda arvates kõige meeldivamad pakutud vaadetest. *

- [ ] "Dashboard"
- [ ] Lõputööde otsing
- [ ] Lõputöö registreerimine
- [ ] Lõputöö gruppi detailne vaade
- [ ] "Submission" testide vaade
- [ ] "Submission" PDF vaade
- [ ] Teavitused

15. Järgnevas küsimuses palutakse hinnata antud väided kasutajaliidese kohta.

	Ei ole päris nõus	Ei ole nõus	Neutraalne	Olen nõus	Olen päris nõus	Ei oska vastata
Kasutajaliides on meeldiv	◯	◯	◯	◯	◯	◯
Kasutajaliideses t on lihtne aru saada	◯	◯	◯	◯	◯	◯
Kasutajaliides tundub nagu ametlik veebileht	◯	◯	◯	◯	◯	◯
Ilma teades, et kasutajaliidest arendavad tudengid, mõtleksin et kasutan ametlikku teenust	◯	◯	◯	◯	◯	◯
Kasutajaliidese funktsioonid on mõistlikud	◯	◯	◯	◯	◯	◯
Kasutajaliidest on lihtne mõista	◯	◯	◯	◯	◯	◯

Vigadekontroll

Selles osas palutakse hinnata kontrollimisprotsessi kiirust, testide arusaadavust ja täpsust.

16. Kas tulemus saabus kiiresti? *

- ◯ Jah
- ◯ Ei
- ◯ Pole kindel

17. Kuidas hindaksite vigadekontrolli kiirust? *

1	2	3	4	5

Liiga aeglane — Väga kiire

18. Palun kommenteerige oma arvamust vigadekontrolli kiiruse kohta. Kas arvate, et lõputöö kontrolli jaoks kiirus on vastav või tahaks kiiremini tulemusi saada? *

19. Palun valige, milliste testidega olete kõige tihedamini kokku puutunud kasutades Lathec teenust. *

- [ ] Abbreviations
- [ ] Appendix title format
- [ ] Empty inputs in main
- [ ] Empty inputs in chapters
- [ ] Empty sections
- [ ] Section titles with capitals
- [ ] Capitalized sentence
- [ ] Double spaces
- [ ] Pronouns
- [ ] Unmatched braces
- [ ] Unmatched parentheses
- [ ] Unmatched square brackets
- [ ] Unmatched quotes
- [ ] Repeated punctuation
- [ ] Parentheses spacing

- [ ] Parts ratio
- [ ] All references in text
- [ ] Wikipedia references
- [ ] Cite sentence format
- [ ] References count
- [ ] Missing files
- [ ] File content mismatches
- [ ] Figures and tables test
- [ ] Compilation test
- [ ] Structure test
- [ ] Spell checker

20. Kui selged olid testide nimed? *

1	2	3	4	5

Liiga segased — Väga selged

21. Kui selged olid testide kirjeldused? *

1	2	3	4	5

Liiga segased — Väga selged

22. Hinnake üldiselt tuvastatud vigade asjakohasust. *

1	2	3	4	5

Vead pole relevantsed — Vead on asjakohased

23. Palun põhjendage enda valikut. *

24. Järgnevas tabelis palutakse hinnata testidega seotud väided. *

	Ei ole päris nõus	Ei ole osaliselt nõus	Nii ja naa	Olen osaliselt nõus	Olen päris nõus	Ei saa vastata
Testide kategooriad on selged ja arusaadavad	◯	◯	◯	◯	◯	◯
Testide nimed on selged ja arusaadavad	◯	◯	◯	◯	◯	◯
Testide kirjeldused on asjakohased	◯	◯	◯	◯	◯	◯
Testide kirjeldused on segased	◯	◯	◯	◯	◯	◯
Testide kirjeldused on liiga tehnilised	◯	◯	◯	◯	◯	◯
Testid panid tähele sellele, millest ei ole mõelnud	◯	◯	◯	◯	◯	◯
Olen rahul pakutud testide kvaliteediga	◯	◯	◯	◯	◯	◯
Arvan, et testide arv võiks suurem olla	◯	◯	◯	◯	◯	◯

25. Kas soovitaksite veebirakendust teistele tudengitele? *

0	1	2	3	4	5	6	7	8	9	10

Ei soovitaks — Väga soovitaks

26. Kui lihtne Teie arvates oleks teenust kasutada mittetehnilise erialal õppivatele tudengitele? *

0	1	2	3	4	5	6	7	8	9	10

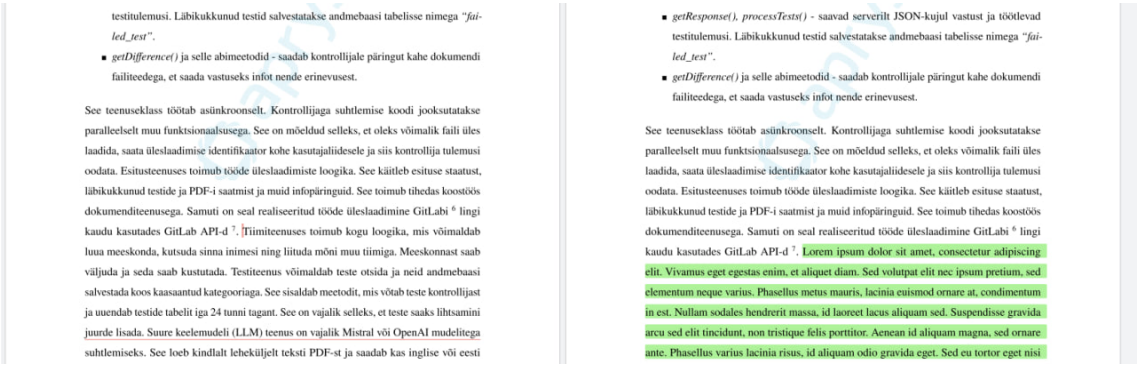
Liiga keeruline — Väga lihtne

27. Palun hinnake järgmist väidet: Lathec teenus pakub erilist ja otstarbekohast lahendust LaTeXis kirjutatud bakalaurusetööde stiili- ja kirjavigade kontrollimiseks. *

1	2	3	4	5	6	7

28. Kui Teil on Lathec teenuse kohta lisakommentaarid või soovitused, palun lisage neid sinna.

Lisa 25 – WebViewer API-ga saadud vaade



testitulemusi. Läbikukkunud testid salvestatakse andmebaasi tabelisse nimega *"failed_test"*.

- *getDifference()* ja selle abimeetodid - saadab kontrollijale päringut kahe dokumendi failiteedega, et saada vastuseks infot nende erinevusest.

See teenuseklass töötab asünkroonselt. Kontrollijaga suhtlemise koodi jooksutatakse paralleelselt muu funktsionaalsusega. See on mõeldud selleks, et oleks võimalik faili üles laadida, saata üleslaadimise identifikaator kohe kasutajaliidesele ja siis kontrollija tulemusi oodata. Esitusteenuses toimub tööde üleslaadimiste loogika. See käitleb esituse staatust, läbikukkunud testide ja PDF-i saatmist ja muid infopäringuid. See toimub tihedas koostöös dokumenditeenusega. Samuti on seal realiseeritud tööde üleslaadimine GitLabi [6] lingi kaudu kasutades GitLab API-d [7]. Tiimiteenuses toimub kogu loogika, mis võimaldab luua meeskonda, kutsuda sinna inimesi ning liituda mõni muu tiimiga. Meeskonnast saab väljuda ja seda saab kustutada. Testiteenus võimaldab teste otsida ja neid andmebaasi salvestada koos kaasaantud kategooriaga. See sisaldab meetodit, mis võtab teste kontrollijast ja uuendab testide tabelit iga 24 tunni tagant. See on vajalik selleks, et teste saaks lihtsamini juurde lisada. Suure keelemudeli (LLM) teenus on vajalik Mistral või OpenAI mudelitega suhtlemiseks. See loeb kindlalt leheküljelt teksti PDF-st ja saadab kas inglise või eesti

- *getResponse(), processTests()* - saavad serverilt JSON-kujul vastust ja töötlevad testitulemusi. Läbikukkunud testid salvestatakse andmebaasi tabelisse nimega *"failed_test"*.
- *getDifference()* ja selle abimeetodid - saadab kontrollijale päringut kahe dokumendi failiteedega, et saada vastuseks infot nende erinevusest.

See teenuseklass töötab asünkroonselt. Kontrollijaga suhtlemise koodi jooksutatakse paralleelselt muu funktsionaalsusega. See on mõeldud selleks, et oleks võimalik faili üles laadida, saata üleslaadimise identifikaator kohe kasutajaliidesele ja siis kontrollija tulemusi oodata. Esitusteenuses toimub tööde üleslaadimiste loogika. See käitleb esituse staatust, läbikukkunud testide ja PDF-i saatmist ja muid infopäringuid. See toimub tihedas koostöös dokumenditeenusega. Samuti on seal realiseeritud tööde üleslaadimine GitLabi [6] lingi kaudu kasutades GitLab API-d [7]. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus eget egestas enim, et aliquet diam. Sed volutpat elit nec ipsum pretium, sed elementum neque varius. Phasellus metus mauris, lacinia euismod ornare at, condimentum in est. Nullam sodales hendrerit massa, id laoreet lacus aliquam sed. Suspendisse gravida arcu sed elit tincidunt, non tristique felis porttitor. Aenean id aliquam magna, sed ornare ante. Phasellus varius lacinia risus, id aliquam odio gravida eget. Sed eu tortor eget nisi

Joonis 52. WebViewer API-ga saadud vaade.

Lisa 26 – Tagarakenduses viiba ehitusfunktsioon

```
private String promptConstructor(ModelPrompt prompt, String
    language, String text) {
    return String.format(
        "%s\n\n%s\n\n Text: \\\"\\\"\\\"%s\\\"\\\"\\\"",
        prompt.getPrompt(),
        String.format(
        """
            Return the response in Markdown format. Use
               headers and subheaders
            to structurise the information and other
               formatting options (bold, italic)
            to put emphasis on important aspects.
            If necessary, use options such as lists, tables
                to present information
            in a more coherent way. Answer in %s language.
        """, language),
        text
    );
}
```