TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Raimond Järg 223026IAIB

Sten Smirnov 223085IAIB

Arete testimismootori edasiarendus

Bakalaureusetöö

Juhendaja: Bahdan Yanovich

BSc

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: Raimond Järg ja Sten Smirnov

04.06.2025

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on parandada Tallinna Tehnikaülikoolis programmeerimisainetes kasutatava automaatse testimissüsteemi jõudlust ning infrastruktuuri, keskendudes selle kesksele komponendile – Arete testimismootorile. Töö käigus analüüsiti olemasolevat süsteemi ja tuvastati mitmed kitsaskohad, sealhulgas ebastabiilne testimisjärjekord, puudulik tagasiside tudengitele, piiratud logide kättesaadavus, vananenud tarkvaraversioonid ja dokumentatsiooni puudulikkus. Töö raames kavandati ja realiseeriti tehnilised täiustused, nagu prioriteetidel põhinev testimisjärjekord, UNI-ID autentimise tugi, esituste jälgitavuse parendamine ning tsentraliseeritud logihaldus. Lisaks koostati arendust toetav dokumentatsioon ja viidi sisse süsteemi töövoogude optimeerimine. Tulemuseks on töökindlam, hallatav ja paremat kasutajakogemust pakkuv testimiskeskkond, mis toetab TalTechi IT-hariduse kvaliteeti.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 42 leheküljel, 7 peatükki, 8 joonist, 1 tabel.

Abstract
Further Development of Arete Testing System

The objective of this bachelor's thesis is to improve the performance and infrastructure of the automated testing system used in programming courses at Tallinn University of Technology, with a focus on its central component – the Arete testing engine. The analysis identified several issues, including unstable test queues, limited student feedback, restricted access to logs, outdated software versions, and missing documentation. The thesis presents technical improvements such as a priority-based test queue system, UNI-ID authentication support, enhanced submission traceability, and centralized logging. Additionally, development-oriented documentation was created and system workflows were optimized. The result is a more robust, maintainable, and user-friendly testing environment that supports the quality of IT education at TalTech.

The thesis is in Estonian and contains 42 pages of text, 7 chapters, 8 figures, 1 table.

Lühendite ja mõistete sõnastik

API	Rakendusliides (*Application Programming Interface*)
CI/CD	Pidev integreerimine ja pidev tarnimine (*Continuous Integration / Continuous Delivery*)
CPU	Keskseade (*Central Processing Unit*)
Cron	Ajamääratud ülesannete ajastaja (*Time-based job scheduler*)
FIFO	Esimesena sisse, esimesena välja (*First In First Out*)
Git	Versioonihaldustarkvara (*Version Control System*)
JSON	JavaScripti objektide notatsioon (*JavaScript Object Notation*)
JWT	JSON veebitoken (*JSON Web Token*)
LTS	Kestustugi (*Long-Term Support*)
MSAL	Microsofti autentimisraamatukogu (*Microsoft Authentication Library*)
Nginx	Veebiserver ja pöördproksiserver (*Web server and reverse proxy*)
OpenID Connect	OpenID Connect protokoll (*OpenID Connect Protocol*)
PostgreSQL	Relatsioonilise andmebaasi haldamise süsteem (*Relational database management system*)
RabbitMQ	Sõnumimaakler (*Message broker*)
RAM	Muutmälu (*Random Access Memory*)
REST	Esitusoleku siire (*Representational State Transfer*)
Spring Security OAuth2	Spring Security OAuth2 turvalisusraamistik (*Spring Security OAuth2 framework*)
UNI-ID	Ülikooli elektrooniline identiteet (*University electronic identity*)
VM	Virtuaalmasin (*Virtual Machine*)
Webhook	Veebihaak (*Web callback*)

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Programmeerimise õpetamisel on tähtsal kohal praktiliste ülesannete iseseisev lahendamine, kuna see aitab tudengitel paremini mõista programmeerimise aluseid ja arendada reaalelulisi oskusi. Tallinna Tehnikaülikoolis on selle toetamiseks kasutusel automatiseeritud testimissüsteem, mis koosneb mitmest eraldi teenusest ning võimaldab suure hulga tudengite tööde automaatset kontrollimist ja hindamist.

Selle süsteemi keskne komponent on Arete – testimismootor, mis käivitab tudengi koodi vastu vastavad testid ja koordineerib kogu testimisprotsessi [1]. Tulemused edastatakse edasi Moodle'i keskkonda Charoni pistikprogrammi kaudu [2]. Charon võimaldab tudengitel oma tulemusi vaadata ning õppejõududel esitatud lahendusi hinnata ja kommenteerida. Selline lahendus on laialdaselt kasutusel TalTechi programmeerimisainetes, sealhulgas järgmistes:

- `ITI0002` – Programmeerimise täiendusõpe
- `ITI0102` – Programmeerimise algkursus
- `ITI0201` – Robotite programmeerimine
- `ITI0214` – Programmeerimise erikursus
- `ITI0204` – Algoritmid ja andmestruktuurid
- `ITI0202` – Programmeerimise põhikursus
- `ITI0211` – Loogiline programmeerimine
- `YFX0505` – Programmeerimine C++ keeles
- `ITT8060` – Programmeerimise erikursus

Kuigi Arete ja Charoni kombinatsioon on võimas ning mitmekülgne, on reaalne kasutuskogemus näidanud mitmeid kitsaskohti. Esines probleeme jõudluse ja läbipaistvusega, samuti puudusid olulised monitooringuvõimalused ja dokumentatsioon.

Käesoleva lõputöö eesmärk on parandada olemasoleva testimissüsteemi töökindlust ja skaleeritavust, lisada vajalikke monitooringulahendusi ning lahendada tuvastatud problee-

mid. Töö raames loodi arendusserver, dokumenteeriti süsteemi tööpõhimõtted ning viidi sisse tehnilised uuendused, mis suurendavad süsteemi kasutusmugavust, läbipaistvust ja hooldatavust.

Töö kliendiks oli Ago Luberg, kelle vajadustest lähtudes kujundati arenduse suund. Töö toimus iteratiivselt: iga etapp keskendus konkreetsele probleemile, millele järgnes lahenduse rakendamine ja tulemuste hindamine. Arutelu ja planeerimine toimusid regulaarselt juhendajaga. Autorid jagasid tööülesanded omavahel, kuid osalesid mõlemad nii süsteemi kasutajaliidese kui ka serveripoolsete komponentide arendamises. Koostööl põhinev ja paindlik töökorraldus võimaldas saavutada töö eesmärgid ning tagas arendusprotsessi tõhususe ja läbipaistvuse.

2 Taustauuring

Peatükis kirjeldatakse olemasoleva rakenduse toimimist ning uuritakse selle kitsaskohti.

2.1 Automatiseeritud testimissüsteemi tööpõhimõte

Automatiseeritud testimissüsteem koosneb mitmest mikroteenusest, millest igaühel on selgelt piiritletud ülesanne. Enne kogu süsteemi tööprotsessi tutvustamist antakse allpool lühike ülevaade olulistest komponentidest:

- **Automated Testing Service** (edaspidi testimisteenus) – Koordineerib testimise käivitamist. Vastutab testimistaotluste vastuvõtmise, töö RabbitMQ järjekorda lisamise ning tulemuste edastamise eest. Teenus on arendatud Java Spring Boot raamistiku abil ning töötab iseseisvas Docker konteineris [3].
- **Authentication Service** (edaspidi autentimisteenus) – Kontrollib saabuvate testimistaotluste kehtivust ning vastutab turvalise päringute edastamise eest testimisteenusele. Rakendatud samuti Spring Boot abil ja konteineriseeritud Dockeri keskkonnas.
- **Test Runner Service** (edaspidi testide käivitamise teenus) – Käivitab testimise tehnilise poole. Võtab töö vastu RabbitMQ-st, käivitab keelepõhise testija eraldi Docker'i konteineris ning loeb testimise tulemused. Teenus ise on arendatud Spring Bootiga ja töötab samuti konteineriseeritult.
- **Charon** – Tulemuste haldussüsteem, kuhu jõuavad testimise tulemused. Charon võimaldab tudengitel tulemusi vaadata ning õppejõududel neid hinnata. Samuti on tudengil võimalik Charoni kaudu lahendusi esitada. Eesrakendus on loodud Vue 3 abil ning tagarakendus kasutab Laravel PHP raamistikku [4] [5] [6].
- **Monitoring Service** – Pakub statistilist ülevaadet testimissüsteemi kasutuse kohta. Näiteks saab vaadata testide edukuse määra, esitamiste sagedust ja süsteemi statistikat. Rakendus on loodud Vue 2-ga [7].

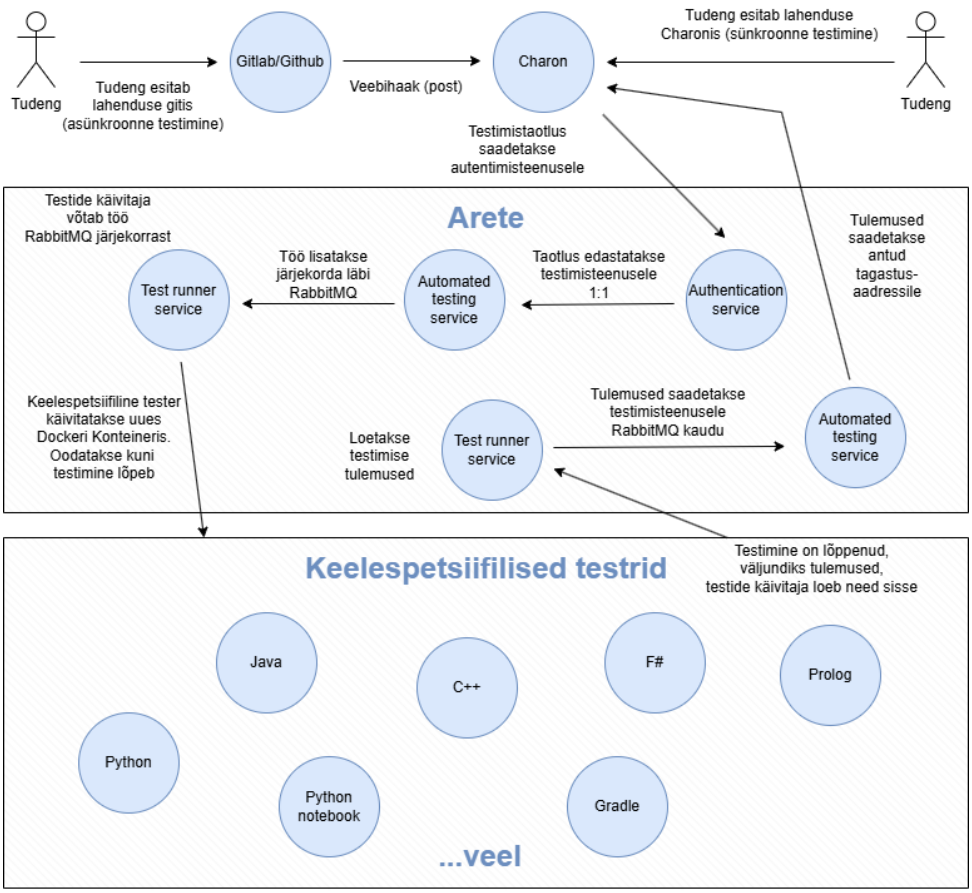
Automatiseeritud testimissüsteem võimaldab tudengite poolt esitatud programmeerimisüles-

andeid hinnata tõhusalt ja skaleeritavalt, kasutades mikroteenustel põhinevat arhitektuuri. Süsteemi keskne komponent on Arete, mis vahendab testimistaotlusi ja koordineerib nende täitmist. Tulemuste koondamiseks ja hindamiseks kasutatakse süsteemi nimega Charon.

Testimisprotsessis on kaks võimalikku alguspunkti:

1. **Esitus Git'i kaudu (asünkroonne töövoog)** – Tudeng saadab oma lahenduse Githubi või Gitlabi repositooriumisse. See käivitab *webhook*'i kaudu sündmuse, mis edastatakse Charonile. Charon tuvastab, millise ülesandega on tegemist, ja suunab testimistaotluse edasi Arete keskkonda. Seal toimub autentimine, mille järel taotlus liigub testimisteenuse kaudu sõnumijärjekorda (RabbitMQ).
2. **Otsene esitus Charoni kaudu (sünkroonne töövoog)** – Tudeng saab esitada oma lahenduse ka otse Charoni veebiliidese kaudu. Sel juhul tehakse sünkroonne päring, mille tulemus tagastatakse otse kasutajale pärast testimise lõppu. See võimaldab tudengil saada kiiremat tagasisidet ilma Git'i vahenduseta.

Mõlemal juhul liigub testimistaotlus Arete keskkonda, testimisteenus lisab töö RabbitMQ järjekorda, testide käivitamise teenus võtab töö järjekorrast ning käivitab vastava keele testija spetsiaalses Docker'i konteineris. Keelepõhised testijad (nt Python, Java, C++, Prolog, Gradle jpt) sooritavad lahenduse automaatse testimise, mille järel tulemused salvestatakse ja tagastatakse testide käivitamise teenusele (vt joonis 1).



Joonis 1. Automatiseeritud testimissüsteemi töövoog

Tulemused edastatakse tagasi testimisteenuse kaudu Charonile, kus need salvestatakse ning tehakse kättesaadavaks nii tudengile kui ka õppejõule. Charon täidab olulist rolli: see on mitte ainult testide tulemuste haldussüsteem, vaid võimaldab ka hindamist ning tulemuste visuaalset kuvamist.

Selline ülesehitus toetab paindlikku töövoogu, tõstab testimise efektiivsust ning vähendab manuaalset hindamist, võimaldades paremat jälgitavust nii tudengitele kui ka õpetajatele.

2.2 CI/CD töövood ja arenduskeskkond

Süsteemis puudub eraldi arenduskeskkond, kus saaks muudatusi eelnevalt testida enne nende tootmisse jõudmist. Kõik muudatused juurutatakse otse tootmiskeskkonda, mis muudab testimise riskantseks ning võib põhjustada katkestusi töötavas süsteemis. Selline lähenemine ei toeta turvalist arendustsüklit ega võimalda vigade varajast avastamist. Tuleks luua eraldi arendusserver, kus saaks muudatusi testida enne tootmiskeskkonda viimist. Lisaks peaks

looma CI/CD töövood, et muudatused väljaspool main haru jõuaksid arendusserverisse.

Eesrakendus töötab tootmiskeskkonnas arendusrežiimis (käsuga npm run start), mis ei ole mõeldud kasutamiseks väljaspool arendust. Sellisel viisil käivitamine võib tuua kaasa madalama jõudluse, turvanõrkused ning ei kasuta tootmiseks mõeldud optimeerimisi, nagu staatiline failiserveerimine või vahemälu kasutus.

2.3 Ebastabiilne jõudlus

Praegusel rakendusel esineb jõudluse probleeme, eriti suure koormusega perioodidel, nagu eksamid ja kontrolltööd. Tänu sellele esineb viivitusi ülesande tulemuste saamisel. Siin alampeatükis kirjeldatakse, kuidas toimub testimine ja uuritakse praeguse lahenduse puudusi.

2.3.1 Testimise järjekord

Kehtib printsiip, et tudengite esitusi testitakse nende esitamise järjekorras. Kuna eksamite ja kontrolltööde tegemise aeg on tudengitele piiratud, pole see kuigi optimaalne, sest need esitused peaksid olema prioriteetsemad, kui tavalised koduülesannete esitused. Kuna tudengeid huvitab üldiselt kõige viimase esituse tulemus, siis tuleks optimeerida järjekorda ka nii, et viimaseid esitusi testitakse enne vanemaid.

2.3.2 Testimine

Iga uue testimisega käivtatakse programmeerimiskeele põhine programm, mis tegeleb testimisega (nt Java, Python). See käivtitatakse alati uues Dockeri konteineris, kuna nii on iga uus testimine omaette töö, ja on ülejäänud süsteemist eraldatud. Kui testimine on lõppenud, peatatakse ja kustutatakse konteiner, ning tulemused saadetakse Arete testimismootorile. Iga selline testimine võtab keskmiselt aega 8 sekundit.

2.4 UNI-ID autentimine

Arete testimismootoril eksisteerib eesrakendus, kust on võimalik vaadata esituste, kursuste, tudengite statistikat ning samuti ka süsteemi toimimist. Seal puudub UNI-ID autentimine, sisse logimine sellesse keskkond toimub läbi eraldi kontode (süsteemi Admin peab ise igale ühele uue konto looma). See on ebamugav ning turvariskidega, kuna tudengite või

töötajate lahkumisega jäävad kontod alles ja puudub täpne ülevaade, millised kontod ei tohiks enam aktiivsed olla.

2.5 Monitooring

Praegusel süsteemil puudub hea monitooring. Pole võimalik vaadata paari päeva taguseid esitusi, ligipääs logidele on raskendatud ja ka tudengitel on toimuvast vähe infot.

2.5.1 Esituste statistika

Eesrakenduses on olemas esituste statistika, aga seal näidatakse ainult viimast 1000 esitust. Sellest pole kasu, sest iga päev tehakse tudengite poolt keskmiselt üle 1000 esituse. Esitusi pole võimalik otsida näiteks kuupäeva või tudengi UNI-ID järgi. Kui tudengil oli mingi probleem paar päeva tagasi, siis tõenäoliselt seda eesrakenduses enam ei näidata ja tuleb seda käsitsi andmebaasist otsima hakata, mis on väga ebamugav ja ajakulukas. Samuti on olemas ka graafikud, mis näitavad esituste arvu kellaaegade, kursuste ja ülesannete lõikes, aga samamoodi näidatakse seda ainult viimase 1000 esituse põhjal. Eesrakenduse kasutajale jääb mulje, et näidatakse kellaajaliselt kogu päeva esitusi, aga tegelikult võivad need andmed täiesti valed olla, sest suurimad küündivad 7000 esituseni päevas.

2.5.2 Logid

Tervel süsteemil on mitu alamosa, mis jooksevad kõik eraldi Docker'i konteinerites. Kuna puudub tsentraalne logide süsteem, on veatuvastus keeruline ja aeganõudev tegevus, sest logide nägemiseks tuleb serverisse sisse logida ja neid käsitsi igast konteinerist eraldi otsima hakata. Selline lähenemine ei ole skaleeritav ega mugav ning võib viia olukordadeni, kus olulised veateated jäävad märkamata või avastatakse hilinenult. Soovitatav oleks rakendada tsentraalne logihaldustööriist, mis võimaldaks logide kogumist, indekseerimist ja visualiseerimist ühest kohast. See parandaks oluliselt süsteemi monitooritavust, lihtsustaks tõrkeanalüüsi ning suurendaks üldist töökindlust ja hooldatavust.

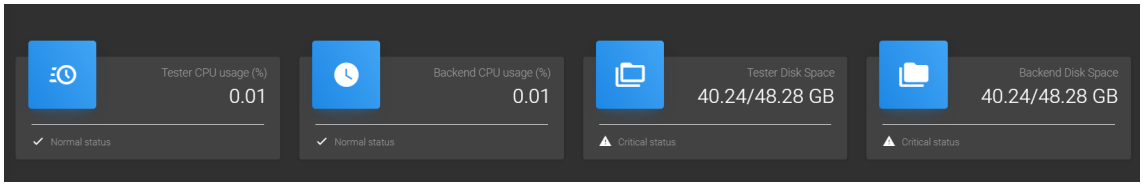
2.5.3 Tagasiside tudengitele

Praeguses süsteemis puudub tudengitel selge ülevaade oma esitatud ülesande testimise seisust. Kui tudeng edastab oma lahenduse Git'i kaudu, ei saa ta teada, mitmendana tema

töö testimisse jõuab ega millises etapis testimine parasjagu on. Selline läbipaistmatuse puudumine võib tekitada ebakindlust ja lisapingeid, eriti eksamiperioodide ajal, kui testimisele kuluv aeg võib pikeneda. Tulemuse ootamise ajal on tudengid sageli ebakindlad, kas nende töö on üldse jõudnud testimisse, mistõttu esitatakse lahendus korduvalt. See omakorda suurendab süsteemi koormust ning võib pikendada testimisaegu veelgi, luues nõiaringi, kus tagasiside hilinemine põhjustab täiendavat koormust ja veelgi suuremat viivitust.

2.5.4 Ressursikasutus

Kuigi süsteem on jõudluskriitiline, puudus töö alguses võimalus jälgida serverite CPU, muutmälu (RAM) ja kettaruumi (disk) kasutust ajas – olemasolev lahendus võimaldas vaadelda ainult süsteemi hetkeolekut, mis ei andnud piisavat ülevaadet ressursside koormuse dünaamikast ega varasematest tõrgetest (vt joonis 2). Selline piirang raskendas süsteemi optimeerimist ja takistas tõrkeanalüüsi, kuna puudusid andmed selle kohta, millal ja millises ulatuses ressursiprobleemid tekkisid; näiteks ei olnud võimalik tuvastada, kas testimisjärjekorra aeglustumine oli seotud CPU ülekoormuse või kettaruumi puudusega. Seetõttu on oluline kavandada lahendus, mis võimaldaks salvestada ja visualiseerida jõudlusandmeid ajas.



Joonis 2. Varasem ressursikasutuse vaade

2.6 Dokumentatsioon

Puudub dokumentatsioon, kuidas teenust lokaalselt käivitada. See võib põhjustada raskusi projekti edasisel arendamisel või hooldamisel, eriti kui sellega tegeleb keegi teine peale algse arendaja. Lokaalse käivitamise juhend on oluline, et tagada süsteemi taaskäivitamise, testimise ja arendamise sujuvus erinevates keskkondades. Tuleks koostada samm-sammuline juhend, mis hõlmab nõutud tarkvara versioone, konfiguratsioonifaile, keskkonnamuutujaid ning käivituskäske. Selline dokumentatsioon lihtsustaks oluliselt projekti haldamist ja vähendaks uute arendajate sisseelamisaega.

2.7 Vananenud süsteem

Süsteem töötab hetkel Ubuntu 20.04 operatsioonisüsteemil, mis on küll veel toetatud, kuid ei ole enam uusim versioon [8]. Arvestades tarkvarakomponentide ja turvauuenduste järkjärgulist nihkumist uuematele versioonidele, oleks mõistlik viia süsteem üle Ubuntu 22.04 versioonile. See tagaks parema ühilduvuse kaasaegsete tööriistade ja raamistikega, samuti pikemaajalise turvatoe (LTS). Süsteemi uuendamine aitaks vältida võimalikke probleeme, mis võivad kaasneda vananenud keskkonnas töötamisega, sealhulgas turvanõrkused, tarkvaradeprekeerimine ning kokkusobimatuse risk uute teekide või arendustööriistadega.

2.8 Ajutiste testimiskaustade kuhjumine

Iga testimise käigus luuakse süsteemis uus ajutine töökaust, kuhu paigutatakse testimiseks vajalikud failid, nagu tudengi lähtekood, testiskriptid ja muud ressursid. Pärast testimise lõppu neid töökaustu automaatselt ei kustutata. Selle tulemusel kogunevad serverisse suurel hulgal mittevajalikke faile, mis aja jooksul hakkavad täitma kettaruumi ning mõjutavad süsteemi stabiilsust. Puudub mehhanism, mis tagaks nende ajutiste failide automaatse eemaldamise.

2.9 Vale koodiversiooni testimine

Testimissüsteem ei arvesta testimistaotluses määratud koodiversiooni (commit hash). Selle asemel testitakse vaikimisi alati repositooriumi viimast saadaolevat versiooni. Selline käitumine põhjustab olukorra, kus tagasiside ei vasta tegelikult testitavale versioonile. See võib tekitada segadust, eriti kordustestimiste puhul, kus oluline on hinnata just konkreetset koodiseisu. Puudub loogika, mis kontrolliks commit hash'i olemasolu ja kehtivust.

2.10 Rollipõhise ligipääsu piirangud ja dünaamilise rollihalduse puudumine

Süsteemis eksisteerivad küll rollipõhised ligipääsupiirangud — näiteks USER rolliga kasutaja ei pääse ligi teatud API otspunktidele, mis on mõeldud vaid administraatoritele või õppejõududele. Samas puudub süsteemis võimalus rolle dünaamiliselt hallata. Kasutajatele ei saa määrata ega muuta rolle pärast konto loomist ning uusi rolle pole võimalik süsteemi lisada. Kõik rollid on staatiliselt defineeritud ja seotud kasutaja loomise hetkega.

Lisaks puuduvad kasutajaliidese tasandil mehhanismid, mis peidaks madalama rolliga kasutajate eest nendele mittekuuluvad sektsioonid. Selle tulemusel kuvatakse kõik süsteemi vaated kõigile autentitud kasutajatele, sõltumata nende tegelikust rollist. Kuigi madalama tasemega kasutajad saavad serverilt päringutele veateate, on neil siiski võimalik näha liidestes valikuid ja funktsioone, millele neil tegelikult ligipääsu pole. See põhjustab segadust, vähendab kasutajakogemuse kvaliteeti ja kujutab endast potentsiaalset turvariski.

3 Nõuded

Selles peatükis on esitatud arendatava süsteemi funktsionaalsed ja mittefunktsionaalsed nõuded. Nõuded põhinevad olemasoleva süsteemi analüüsil ning vigadel, mida tuvastati eelnevate aastate kasutuskogemuse ja tehnilise ülevaate põhjal.

3.1 Funktsionaalsed nõuded

- Süsteem peab võimaldama esituste statistikat pärida kuupäevade kaupa ning agregatsiooniga (nt esituste koguarv).
- Statistika peab olema filtreeritav kuupäeva, tudengi UNI-ID, ülesande/kursuse ja muude parameetrite alusel.
- Esituste graafikud peavad kajastama kogu päevaseid andmeid, mitte ainult viimase 1000 esituse põhjal koostatud infot.
- Tudengile peab olema kuvatav, mitmes tema esitus testimisjärjekorras on ja mis seisus testimine on.
- Eesrakendusse tuleb integreerida UNI-ID autentimine, et võimaldada keskselt hallatud ligipääsu.
- Testimismootor peab oskama prioritiseerida esitusi (nt eksamitööd testitakse enne kodutöid).
- Ainult viimased esitused peavad säilitama oma algse prioriteedi, teised duplikaatesitused peaksid lineaarselt kaotama oma algse prioriteedi.
- Kui tudengil on mitu esitust järjekorras, tuleks eelisjärjekorras võtta viimasena esitatud töö.
- Peab olema võimalik logide otsing ja kuvamine keskses logivaatlussüsteemis.
- Süsteem peab toetama testimistaotluses määratud konkreetse commit hash'i testimist, mitte vaikimisi viimast versiooni.
- Iga testimistaotluse järel loodud töökaustad tuleb süsteemil automaatselt kustutada pärast testimise lõppu.
- Testimissüsteem peab olema võimeline näitama tudengile arusaadavat jaotust testimise

etappidest (nt „järjekorras“, „testimisel“, „valmis“).

- Süsteemis peab olema võimalik arendada ja testida muudatusi eraldi arenduskeskkonnas, enne kui need tootmisse jõuavad.
- Süsteem peab võimaldama rolle lisada, muuta ja kustutada jooksvalt vastavalt kasutajate vajadustele, tagades sellega süsteemi paindlikkuse ja turvalisuse.

3.2 Mittefunktsionaalsed nõuded

- Süsteem peab toetama tsentraalset logide haldust, mille kaudu saab kõigi mikroteenuste logisid koondatult jälgida.
- Lokaalse käivitamise juhend peab olema olemas, dokumenteeritud ja versioonihalduses.
- Süsteem peab jooksma uuemal LTS versioonil (Ubuntu 22.04).
- Süsteem peab jälgima ja salvestama jõudlusmõõdikuid (CPU, RAM, kettaruum) ajas ning võimaldama nende visualiseerimist.
- Eesrakendus peab tootmiskeskkonnas töötama optimeeritud režiimis ning olema serveeritud staatiliste failidena (nt läbi Nginxi).
- CI/CD töövoog peab toetama arenduskeskkonda eraldi haruna, kuhu muudatused deploy’takse enne main-haru tootmisse jõudmist.
- Süsteem peab taluma tippkoormust (nt eksamiperioodil), ilma et see põhjustaks järjekordade seiskumist või liigseid viivitusi.

4 Arendus

Selles peatükis kirjeldatakse teenuse arendust ning kasutusel olevaid tehnoloogiaid.

4.1 CI/CD töövoog ja arenduskeskkond

Arendusprotsessi toetamiseks seadistati iseseisev arendusserver koos automatiseeritud CI/CD töövooga. Töövoog on realiseeritud Gitlabi CI/CD süsteemi abil, kus igale harule väljaspool `main` haru rakendatakse automaatne ehitus- ja juurutusprotsess.

Teenuste konteineriseerimiseks kasutatakse `Docker`it ning nende käivitamiseks `docker-compose` lahendust. Kõik süsteemi mikroteenused (Automated Testing Service, Authentication Service, Test Runner Service ja Monitoring Service) käivitatakse eraldiseisvates konteinerites.

Vue-põhises eesrakenduses tuvastati, et varasemalt käivitati see tootmiskeskkonnas arendusserverina käsuga `npm run start`, mille tulemusel jooksis rakendus arendusrežiimis. See asendati korrektse ehitus- ja serveerimisprotsessiga: esmalt kompileeritakse rakendus käsuga `npm run build`, seejärel serveeritakse genereeritud staatilisi faile `nginx` veebiserveri abil [9].

Kõik muudatused on hallatud versioonikontrollis ning seotud GitLabi `.gitlab-ci.yml` konfiguratsiooniga, mis määrab ehitusetapid ja konteinerite uuendamise loogika arendusserveris.

4.2 Jõudluse parandamine

Selles peatükis kirjeldatakse arenduse käigus tehtud muudatusi, mis puudutasid tööde järjekorra haldust ning testimisprotsessi skaleeritavust.

4.2.1 RabbitMQ

Süsteemis kasutatakse RabbitMQ-d tööde vahendamiseks ja jaotamiseks. RabbitMQ toimib tööde järjekorrana (message queue), mille kaudu süsteemikomponendid saavad töid saata ja vastu võtta detsentraliseeritult ja skaleeritavalt.

RabbitMQ võimaldab lahutada tööde genereerimise ja töötlemise protsessid, mis on oluline koormuse tasakaalustamiseks ja süsteemi vastupidavuse tagamiseks. Tootja (producer) lisab töö sõnumina järjekorda ja tarbija (consumer) loeb sõnumeid järjekorrast ning töötleb neid vastavalt loogikale [10]. Antud kontekstis on tootja testimisteenus ja tarbija testide käivitamise teenus.

RabbitMQ kasutamine on õigustatud eelkõige olukordades, kus tööde saabumine ja nende töötlemine ei toimu samas tempos ning on vaja paindlikku, hajutatud tööde haldust.

4.2.2 Järjekorra optimeerimine

Varasemalt edastati kõik esitused kohe pärast saabumist testimisteenuse kaudu RabbitMQ sõnumijärjekorda ning testide käivitamise teenus võttis selle vastu, kui oli vaba ruumi, järgides FIFO-põhimõtet.

Lõputöö käigus asendati see arhitektuur järgnevalt:

- **Taustaprotsessiga tööde saatmine:** Iga 100 ms järel käivitatakse testimisteenuse sees taustaprotsess, mis kontrollib testide käivitajate vabu ressursse ning saadab testimisse tööd ainult siis, kui vaba ruum on saadaval. Selline muudatus oli vajalik, kuna RabbitMQ ei paku tõhusaid lahendusi prioritiseeritud järjekordade haldamiseks. Kuigi süsteem toetab prioriteete, võib järjekord ummistuda juhul, kui sinna koguneb suur hulk kõrgeima (näiteks prioriteet 10) tasemega sõnumeid. RabbitMQ töötleb esmalt kõrgema prioriteediga sõnumeid ning madalama prioriteediga sõnumid jäävad sel juhul pikalt ootele. See võib kokkuvõttes pidurdada kogu süsteemi läbilaskevõimet [11].
- **Prioriteedisüsteem:** Igale esitusele on võimalik määrata prioriteet vahemikus 1–10. Tööd järjestatakse töötlemiseks prioriteedi alusel, alustades kõrgeimast.
- **Prioriteedipõhine võtmepiirang:** Igal prioriteeditasemel on maksimaalne lubatud

tööde arv, mida võib järjest võtta ühe tsükli jooksul. Näiteks prioriteediga 10 puhul kuni 6 tööd, prioriteediga 9 puhul kuni 3 tööd jne (vt tabel 1).

Tabel 1. Prioriteeditasemete maksimaalsed lubatud tööde arvud

Prioriteet	Maksimaalne tööde arv
10	6
9	3
8	3
7	2
6	2
5	1
4	1
3	1
2	1
1	1

Seda lähenemist oli vaja selleks, et vältida olukorda, kus kõrgeima prioriteediga (nt 10. taseme) tööd monopoliseerivad kogu töötlusressursi ja väiksema prioriteediga esitused ei saa enne testitud, kui kõik suurema prioriteediga ülesanded on testitud [12].

- **Duplikaatesituste de-prioritiseerimine:** Kui tudengil on järjekorras mitu esitust, määratakse ainult viimasele esitatud tööle algne prioriteet. Eelnevate esituste prioriteete vähendatakse lineaarselt. See tähendab, et kui tudeng teeb uue esituse, ja tal on ka teisi esitusi järjekorras või testimises, siis nende prioriteet väheneb. Kui esituse prioriteet on üle 5, vähendatakse 2 punkti võrra, kui alla selle, 1 punkti võrra, prioriteet alla 1 kunagi ei lähe.
- **Järjekorraloogika välja toomine RabbitMQ-st:** RabbitMQ-d kasutatakse ainult tööde edastamiseks, mitte nende järjestamiseks või prioritiseerimiseks. Kõik järjekorrahaldused ja prioriteetide loogika on nüüd rakendatud testimisteenuses lokaalses mälustruktuuris.

4.3 Monitooring

Selles peatükis kirjeldatakse tehnilisi muudatusi, mille eesmärk oli täiustada süsteemi monitooringut ning nähtavust. Muudatused hõlmasid autentimist, esituste ajaloo haldust, järjekorra tagasisidet ja logide käsitlust.

4.3.1 UNI-ID autentimine

Monitooringurakendusse integreeriti Microsoft Azure Active Directory-põhine autentimine, mis seoti TalTechi UNI-ID süsteemiga. Eesrakenduses kasutati autentimiseks Microsofti ametlikku MSAL teeki (Microsoft Authentication Library), mille kaudu sooritatakse OpenID Connect protokolli põhine autentimine [13] [14].

Tagarakenduses rakendati identiteedihaldus Spring Security OAuth2 Resource Server mooduli abil. Lahendus võimaldab valideerida JWT-tüüpi autentimistokeni ja seostada see rollipõhise ligipääsupoliitikaga [15].

Autentimisprotsess toimib läbi Microsofti sisselogimisportaali, kus autenditud kasutaja saab ligipääsu süsteemile ilma lokaalse konto loomise vajaduseta.

Arenduse käigus laiendati olemasolevat rollihaldust, võimaldades olemasolevate rollide muutmist ja uute rollide lisamist. Tagarakenduses loodi REST API otspunktid, mille kaudu saab administraator rollidega seotud andmeid hallata. Nende otspunktide kaudu on võimalik määrata kasutajale uusi rolle ja eemaldada olemasolevaid rolle,

API otspunktid on kaitstud vastava autoriseerimise kihiga, nii et neid saavad kasutada vaid ADMIN rolliga kasutajad. Rollihalduslogika realiseeritakse teenusekihis olemasoleva User ja Role andmebaasi tabelite vahel.

Eesrakenduses implementeeriti rollipõhine nähtavuse kontroll. Sisselogimisel saadud autentimistoken dekodeeritakse ja kasutaja roll salvestatakse rakenduse lokaalsesse mällu. Komponentide renderdamisel kontrollitakse kasutaja rolli ning tingimuslikult jäetakse välja need vaated ja navigeerimislingid, mis ei vasta antud rollile. Näiteks ei renderdata USER rolliga kasutajale administraatori vaateid ega menüüelemente. Kõik rollipõhised piirangud rakendatakse enne renderdamist, vältides lubamatute elementide nähtavust liideses.

4.3.2 Esituste ajaloo näitamine

Esituste vaatamiseks lisati süsteemi serveripoolne API otspunkt, mille kaudu saab laadida kogu esituste ajaloo alates süsteemi kasutuselevõtust. Andmeid on võimalik filtreerida parameetrite alusel nagu:

- tudengi UNI-ID;
- esituse kuupäev;
- ülesande või kursuse identifikaator.

Eesrakenduses realiseeriti vastav Vue komponent, mis toetab dünaamilist otsingut ja filtreerimist. Tagarakendus loeb vajaliku info PostgreSQL andmebaasist ja tagastab vastavalt päringuparameetritele tulemused.

Rollipõhine ligipääs rakendati samuti Spring Security kaudu:

- administraatori ja õppejõu roll saavad vaadata kõigi tudengite esitusi;
- tudeng saab ligipääsu ainult enda esitatud töödele, tuvastades kasutaja UNI-ID JWT tokeni põhjal.

4.3.3 Logide haldus

Logide tsentraliseeritud haldus on planeeritud, kuid ei olnud lõputöö raames veel täielikult realiseeritud. Eesmärgiks on võimaldada ligipääs teenuste logidele läbi keskse logihaldustööriista (nt ELK Stack või Grafana Loki) [16]. Praeguseks loodi alusinfrastruktuur logide konteineripõhiseks suunamiseks, kasutades `Docker logging driver`'eid ja failiväljavõtet, mida saab suunata logikogumisteenusele.

4.3.4 Tagasiside tudengitele

Testimisteenusesse lisati REST API otspunkt, mille kaudu on võimalik küsida tudengi mingi ülesande testimisel olevate esituste positsiooni testimisjärjekorras.

Järjekorra positsiooni arvutamiseks kasutatakse teenuse sisemist prioriteedipõhist tööjaotust ning tudengi UNI-ID ja ülesande alusel filtreerimist. Järjekorda küstiakse Charonist, tudengi esituste lehel vaid selle kindla ülesande ja tudengi kohta, õppejõudude vaates kõigi selle kursuse testimisel olevate esituste kohta.

Tudengi vaates toimub andmete uuendamine automaatselt ainult juhul (iga 5 sekundi tagant), kui kasutaja on aktiivselt vaatamas konkreetset ülesannet.

4.3.5 Jõudlusandmete visualiseerimise täiustamine

Süsteemi täiustati, asendades staatilised ressursikasutuse kaardid dünaamiliste graafikutega. Uus lahendus võimaldab kasutajal valida konkreetse kuupäeva ning vaadelda CPU, RAM-i ja ketta kasutuse muutusi päeva lõikes.

Andmete kogumiseks lisati regulaarne salvestusmehhanism: CPU ja RAM andmeid kogutakse iga 10 minuti järel, ketta kasutuse andmeid iga 30 minuti järel. Salvestamine toimub cron-ajastuse abil, mis tagab andmete püsivuse ja korrapärasuse.

Graafikute visualiseerimise kiht loodi kasutajaliideses, kus andmed esitatakse aja põhjal filtreeritaval kujul. See võimaldas luua ajalise mõõtmega monitooringuvaated ilma, et oleks vaja kasutada välist tööriista.

4.4 Veaparandused

Siin peatükis tuuakse välja suuremad vead, mis tuvastati arenduse käigus, ja kuidas need parandati.

4.4.1 Testimiskaustade kogunemine

Testimise alustamisel saadab testimisteenus testide käivitamise teenusele kõik vajalikud failid (sh tudengi lähtekood, testifailid, pildifailid, JSON-id jms) string-kujul, mille põhjal testide käivitaja loob uue ajutise töökausta. See kaust antakse testimiseks loodud Docker konteinerile ette.

Probleem seisnes selles, et pärast testimise lõppu neid ajutisi kaustu ei eemaldatud. Iga esitus jättis serverisse maha uue kausta koos kõigi failidega. Lõputöö raames lisati testide käivitamise teenusesse loogika, mis tagab töökaustade automaatse kustutamise pärast testimise lõppu – sõltumata sellest, kas test õnnestus või ebaõnnestus.

4.4.2 UNI-ID tuvastamise probleem Githubi esitustega

Üheks olulisemaks veaks osutus probleem Githubi kaudu tehtud esitustega, mis ilmnes pärast uue versiooni (release'i) juurutamist Charoni süsteemis. Probleemi tuvastas **Charoni arendustiim**, kuhu kuulub ka lõputöö autor.

Varasemalt tuvastas Charon esitajaks tudengi UNI-ID, mille ta tuletas Githubi webhookist saadud e-posti aadressi põhjal. See eeldas, et tudengi Githubi konto e-mail kattub tema ametliku TalTechi aadressiga. Pärast uuendust muutus aga vaikimisi käitumine: tudengi identifikaatorina saadeti edasi Githubi kasutajanimi, mitte enam e-posti alusel tuvastatud UNI-ID.

Kuna Githubi kasutajanimed ei ole seotud TalTechi identiteedihaldusega ja pole unikaalsed ülikooli kontekstis, ei olnud võimalik esitamist õigesti siduda Moodle'is oleva hindekirjega. Selle tagajärjel jäid tudengid ilma tagasisidest ja automaatsest hindest.

Probleem lahendati **Arete** poolel, lisades loogika, mis käsitleb Githubi kaudu saabuvat e-posti kui varu-UNI-ID-d. Kui Charon edastab testimistaotlusele kaasa Githubi webhook'i kaudu saadud e-posti, kontrollib Arete esmalt, kas päringul juba eksisteerib UNI-ID. Kui UNI-ID puudub, määrab Arete selleks kasutatud e-posti aadressi ja tagastab selle Charonile vastuses. Sel viisil säilib korrektne seos tudengi ja tema esituse vahel ka juhul, kui Charon ei lisa algses päringus UNI-ID-d.

4.4.3 Vale commit'i testimine retestimisel

Testimissüsteemis esines probleem, mille puhul iga esitus testiti alati repositooriumi viimase commit'i vastu, isegi kui testimistaotluses oli määratud konkreetne commit hash. See tähendas, et uuesti testimise korral – näiteks Charoni kaudu algatatud retestimisel – ignoreeriti määratud hash'i ning kasutati alati repositooriumi kõige uuemat versiooni. Selle tulemusel ei vastanud testimise tulemus enam koodi algsele seisundile taotluse esitamise hetkel.

Probleemi lahendamiseks muudeti testimise loogikat nii, et süsteem kontrollib nüüd enne testimise alustamist, kas saadetud hash on olemas ja kehtiv. Kui see on määratud ning vastab korrektsele Git commit'i formaadile, viiakse testimine läbi just selle konkreetse versiooni

alusel. Ainult juhul, kui hash puudub või on vigane, kasutatakse vaikimisi repositooriumi viimast versiooni.

4.4.4 Sisendandmete sidumise probleem esitustega

Testimissüsteemi arenduse käigus ilmnes probleem, mis puudutas esituste korrektset sidumist õige tudengi ja kursusega. Probleemi põhjustas ajastuse ebatäpsus esituse salvestamisel: testide käivitamis teenuse poolt tagastatud esituse ajatempel võis erineda süsteemis registreeritud esituse algse ajatempli väärtusest kuni mõne millisekundi võrra.

Kuna sisendandmete sidumine toimus peamiselt esituse salvestusaja alusel, viis see olukordadeni, kus mõnel juhul ei õnnestunud testimissüsteemil täpselt määrata, millise esitusega konkreetne testimise tulemus seostub. Selline ebatäpsus võis omakorda tekitada olukordi, kus esituse tulemused ei jõudnud õigesti hindamisüsteemi või kandsid vale identifikaatorit, raskendades tulemuste korrektset kuvamist ja hindamist.

Probleemi lahendamiseks täiustati testimisteenuse sisendandmete sidumise ja valideerimise loogikat, vähendades sõltuvust täpsest ajatemplist ning kontrollides andmete olemasolu, kehtivust ja seotust teiste süsteemi objektidega enne testimise alustamist. Selle tulemusel vähenes eksimuste arv sidumisprotsessis ning suurenes süsteemi töökindlus ja andmete usaldusväärsus.

4.4.5 Järjekorra kuhjumine eesrakenduses

Arenduse käigus tuvastati probleem, mille korral tööde järjekorra andmete päringute käigus lisati tööde andmed mälusisesele andmestruktuurile korduvalt. Kui kasutaja vaatas tööde järjekorda ja eesrakendus esitas tagarakendusele korduvaid päringuid, siis iga päring lisas samad tööd andmestruktuuri uuesti.

Selline dubleerimine põhjustas tööde arvu eksponentsiaalset kasvu mälus, mis suurendas süsteemi mälukasutust ja protsessorikoormust. Suurema koormuse korral viis see jõudluse languseni ja tõstis riski, et süsteemi töö võib katkeda.

Probleemi lahendamiseks muudeti andmete töötlemise loogikat nii, et iga töö lisatakse andmestruktuuri vaid üks kord. Selle tulemusena stabiliseerus süsteemi mälukasutus ja tööde kuvamine jäi korrektseks ka sagedaste päringute korral.

5 Tulemused

Selle peatüki eesmärk on hinnata loodud lahenduse töökindlust, funktsionaalset vastavust ning praktilist sobivust reaalsetes kasutusolukordades. Uut järjekorra- ja prioriteedihaldust testiti sihilikult loodud koormusstsenaariumite kaudu, kus simuleeriti nii ühekordseid kui ka korduvaid esitusi erinevatelt kasutajatelt. Lisaks kontrolliti süsteemi vastavust püstitatud funktsionaalsetele ja mittefunktsionaalsetele nõuetele.

5.1 Lõputöö nõuete täitmine

Selles alapeatükis on toodud ülevaade lõputöös püstitatud nõuete täitmisest.

5.1.1 Funktsionaalsed nõuded

- Süsteem peab võimaldama esituste statistikat pärida kuupäevade kaupa ning agregatsiooniga (nt esituste koguarv) – **täidetud**
- Statistika peab olema filtreeritav kuupäeva, tudengi UNI-ID, ülesande/kursuse ja muude parameetrite alusel – **täidetud**
- Esituste graafikud peavad kajastama kogu päevaseid andmeid, mitte ainult viimase 1000 esituse põhjal koostatud infot – **täidetud**
- Tudengile peab olema kuvatav, mitmes tema esitus testimisjärjekorras on ja mis seisus testimine on – **täidetud**
- Eesrakendusse tuleb integreerida UNI-ID autentimine, et võimaldada keskselt hallatud ligipääsu – **täidetud**
- Testimismootor peab oskama prioritiseerida esitusi (nt eksamitööd testitakse enne kodutöid) – **täidetud**
- Ainult viimased esitused peavad säilitama oma algse prioriteedi, teised duplikaatesitused peaksid lineaarselt kaotama oma algse prioriteedi – **täidetud**
- Kui tudengil on mitu esitust järjekorras, tuleks eelisjärjekorras võtta viimasena esitatud töö – **täidetud**

- Peab olema võimalik logide otsing ja kuvamine keskses logivaatlussüsteemis – **osaliselt täidetud**
- Süsteem peab toetama testimistaotluses määratud konkreetse commit hash'i testimist, mitte vaikimisi viimast versiooni – **täidetud**
- Iga testimistaotluse järel loodud töökaustad tuleb süsteemil automaatselt kustutada pärast testimise lõppu – **täidetud**
- Testimissüsteem peab olema võimeline näitama tudengile arusaadavat jaotust testimise etappidest (nt „järjekorras", „testimisel", „valmis") – **täidetud**
- Süsteemis peab olema võimalik arendada ja testida muudatusi eraldi arenduskeskkonnas, enne kui need tootmisse jõuavad – **täidetud**
- Iga testimistaotluse järel loodud töökaustad tuleb süsteemil automaatselt kustutada pärast testimise lõppu – **täidetud**
- Süsteem rakendab rollipõhist autoriseerimist – kasutajale määratud rollist sõltub tema ligipääs andmetele ja funktsioonidele. Kuigi tehniliselt eksisteerisid mõned rollid juba varem, ei olnud neid võimalik hallata ega muuta. Nüüd on süsteemil kasutajaliides rollide määramiseks ja muutmiseks, mis muudab õiguste haldamise paindlikuks ja usaldusväärseks - **täidetud**

5.1.2 Mittefunktsionaalsed nõuded

- Süsteem peab toetama tsentraalset logide haldust, mille kaudu saab kõigi mikroteenuste logisid koondatult jälgida – **osaliselt täidetud**
- Lokaalse käivitamise juhend peab olema olemas, dokumenteeritud ja versioonihalduses – **täidetud**
- Süsteem peab jooksma uuemal LTS versioonil (Ubuntu 22.04) – **täidetud**
- Süsteem peab jälgima ja salvestama jõudlusmõõdikuid (CPU, RAM, kettaruum) ajas ning võimaldama nende visualiseerimist – **täidetud**
- Eesrakendus peab tootmiskeskkonnas töötama optimeeritud režiimis ning olema serveeritud staatiliste failidena (nt läbi Nginxi) – **täidetud**
- CI/CD töövoog peab toetama arenduskeskkonda eraldi haruna, kuhu muudatused deploy'takse enne main-haru tootmisse jõudmist – **täidetud**
- Süsteem peab taluma tippkoormust (nt eksamiperioodil), ilma et see põhjustaks järjekordade seiskumist või liigseid viivitusi – **täidetud**

5.2 Ubuntu versiooni uuendamine

Süsteemi töökeskkonnas viidi läbi Ubuntu operatsioonisüsteemi versiooniuuendus, et tagada kaasaegsete turvaparanduste ja tarkvarakomponentide tugi. Enne uuendust koostati tegevusplaan, mis sisaldas tagavarakava võimalike tõrgete puhuks. Lepiti kokku sobiv kuupäev ja ajavahemik, mil uuendamine toimuks. Uuendus viidi ellu õhtusel ajal, mil süsteemi koormus on tavapäraselt madal ning aktiivsete kasutajate arv väiksem.

Uuendusprotsess viidi läbi järgmiste sammude alusel:

- Loodi süsteemist varukoopia.
- Suunati kogu süsteemiliiklus ajutiselt arendusserverisse.
- Käivitati uuendusprotsess, mis kestis ligikaudu üks tund.
- Pärast uuenduse lõpetamist suunati liiklus tagasi põhiserverisse.
- Süsteemi tööd jälgiti veel mitu tundi, et tuvastada võimalikke kõrvalekaldeid või tõrkeid.

Uuendusprotsessi käigus valideeriti ka arendusserveri töövõime olukorras, kus põhiserver on ajutiselt kättesaamatu. Katkestuse perioodil toimis arendusserver iseseisvalt, käideldes kogu liiklust ilma probleemideta. Selle tulemusel kinnitati, et loodud arendusserver suudab vajadusel täita tagavaraserveri rolli nii ootamatute rikete kui ka planeeritud hooldustööde korral.

5.3 Jooksev tagasiside ja koostöö õppejõududega

Süsteemi arendamise ja uuenduste juurutamise käigus koguti järjepidevalt tagasisidet (abi)õppejõududelt, kes on süsteemi aktiivsed kasutajad. Saadud sisend aitas kiiresti tuvastada kasutusmugavuse kitsaskohti, tehnilisi tõrkeid ja täpsustada ootusi uutele funktsionaalsustele.

Näiteks juhtisid õppejõud tähelepanu juhtumitele, kus süsteem käitus ootamatult – üheks konkreetseks näiteks oli olukord, kus tudengite koodi kompileerimisvead liigitati ekslikult stiilivigade hulka. Selline klassifikatsioon tekitas segadust nii hindamisel kui ka tagasiside andmisel. Pärast probleemi esiletõstmist võeti see viivitamatult lahendamisele: kompileerimisvigade käsitlemine eraldati selgelt teistest veatüüpidest ning vastav loogika muudeti

testimissüsteemis.

Iga lahenduse juures suheldi otseselt probleemi raporteerinud õppejõuga, paludes kinnitust, kas tehtud muudatus vastab ootustele. Selline lähenemine võimaldas tagada, et parandused ei lahendanud mitte ainult tehnilist probleemi, vaid vastasid ka reaalsele kasutusvajadusele.

Õppejõudude tagasiside aitas valideerida, et muudatused parandavad süsteemi kasutatavust õppetöö kontekstis ning tõstavad tööprotsesside efektiivsust. Samas võimaldas see lähenemine kiiresti reageerida probleemidele, enne kui need mõjutasid suuremat hulka kasutajaid. Koostööpõhine tagasisidekanal kujunes oluliseks komponendiks süsteemi kvaliteedi tagamisel ning aitas maandada riske seoses uuenduste juurutamisega õppetöö aktiivses faasis.

5.4 Uue järjekorrasüsteemi toimivuse valideerimine

Süsteemi valideerimiseks viidi läbi eksperiment, mille eesmärk oli hinnata uue töötlusjärjekorra loogika toimivust ja õiglust erinevate tudengite tööde käsitlemisel. Testimisskripti abil esitati sama ülesande lahendusi paralleelselt nii korduvate kui ka unikaalsete tudengite nimel. Viis kindlat tudengit esitasid mitmeid järjestikuseid töid, samas kui kuues lõim genereeris ühe töö igaühele üheksast erinevast tudengist.

Oluline on märkida, et nii varasemas kui ka uues lahenduses oli tagatud, et iga tudengil saab korraga olla töötluses vaid üks aktiivne töö. Probleem ilmnes aga tööde järjekorda lisamisel — varasemas süsteemis võisid ühe tudengi mitmed järjestikused tööd hõivata suure osa töötlusjärjekorrast, lükates teiste tudengite tööd põhjendamatult hilisemaks. Järjekorra sees ei eristatud, kas tegemist on sama või erineva tudengi töödega, ega arvestatud esitatud tööde prioriteete ega esitamisaega.

Uus süsteem rakendab dünaamilist järjekorrahaldust, mis lisab iga tudengi tööd tema isiklikku järjekorda ning tõstab kõige hilisema töö prioriteeti. Varem lisatud tööd ei kao, kuid nende prioriteet väheneb järk-järgult. Lisaks kasutatakse prioriteedipõhist jaotust, kus iga prioriteeditaseme puhul on määratud maksimaalne arv järjestikuseid töid, mida lubatakse töötlusesse. See tagab, et ka madalama prioriteediga tööd saavad võimaluse töötluseks, vältides süsteemi blokeerumist ainult kõrge prioriteediga tööde tõttu.

Eksperimendi tulemused näitasid, et uus süsteem jagab töötlusressursse märkimisväärselt õiglasemalt. Mitme tudengi samaaegsel esitamiste korral said esimesed tööd igalt tudengilt kiiremini töötlusse võrreldes varasema süsteemiga, kus sama tudengi järjestikused tööd võisid teisi varjutada. See tõestas, et uus lahendus tagab parema tasakaalu tööde prioriteetide ja kasutajate võrdsuse vahel.

Kokkuvõttes parandas uus töötlusloogika süsteemi reageerimisvõimet, vähendas koormuse kontsentreerumist üksikutele kasutajatele ning tagas, et viimane esitus sama tudengi puhul jõuab kiiremini töötlusse ilma varasemaid esitusi ära viskamata.

5.4.1 Järjekorrasüsteemi jälgimine eksamisituatsioonis

Uue töötlusjärjekorra toimivust jälgiti reaalsetes tingimustes aine *Programmeerimise Põhikursus (ITI0202)* eksami ajal. Kogu eksamiperioodi jooksul monitooriti süsteemi käitumist eesmärgiga kinnitada, et uus järjekorrasüsteem ei põhjusta viivitusi ega koorma töötlemismoodulit rohkem kui varasem lahendus.

Vaatluse käigus koguti andmeid tööde esitamise, järjekorda lisamise ja töötlemise ajast. Eraldi pöörati tähelepanu sellele, kuidas süsteem käitub korduvate esituste korral, mis eksamiperioodil on tavapärasest sagedasemad. Tulemustest selgus, et süsteem säilitab töötlusvõimekuse ka kõrge koormuse tingimustes ning tagab tööde jõudmise töötlemisse vähemalt samal tasemel kui varasem lahendus.

Lisaks täheldati, et tänu dünaamilisele prioriteedihaldusele jõudsid eksami ülesanded kiiremini testimisse ning sama tudengi mitmekordsed esitused ei blokeerinud teiste kasutajate töid. Duplikaatesituste prioriteedi langus töötas ootuspäraselt ja hoidis töötlusressursid kättesaadavana kõigile.

Kokkuvõttes aitas eksamiaegne jälgimine kinnitada, et uus järjekorrasüsteem toimib stabiilselt ja usaldusväärselt ka suurenenud koormuse all, säilitades või isegi ületades eelmise lahenduse töökindlust ja õiglust.

5.4.2 Kasutajate tagasiside

Arendatud süsteemi uusi funktsionaalsusi katsetati mitmete testkasutajatega, kes andsid tagasisidet nii kasutusmugavuse, funktsionaalsuste kui ka kasutajaliidese kohta. Testka-

sutajate tagasiside koguti pärast süsteemi demoesitlust, mille käigus tutvustati arendatud töövooge, järjekorra loogikat, monitooringut ja tudengitele loodud uusi vaateid.

Üldine tagasiside oli positiivne. Kasutajaliidese puhul tõsteti esile, et uued visuaalsed lahendused ning statistika ja tööde esituste skeemid muudavad süsteemi arusaadavaks ja visuaalselt atraktiivseks. Mitmed kasutajad tõid välja, et andmete koondatud kuvamine ning graafilised visualiseeringud lihtsustavad süsteemi jälgimist ja kasutamist.

Testimisjärjekorra prioriteetidel põhinevat loogikat hinnati eriti kasulikuks eksamite kontekstis, kuna see aitab vältida olukordi, kus eksamitööd jäävad kodutööde või korduvate esitamiste varju. Samuti tõsteti esile, et tänu uuele prioriteedisüsteemile ja duplikaatesituste de-prioriseerimisele väheneb süsteemi ülekoormus ning tudengid saavad kiiremini tagasisidet.

Kasutajamugavuse osas pakuti välja täiendusi, näiteks kuupõhise kalendrivaate lisamine esituste ajaloos, et lihtsustada kuupäevade järgi orienteerumist. Samuti nähti võimalust kuvada monitooringu andmetes näiteks RAM-i ja CPU kasutuse hetkeväärtusi otse pealkirjades, et vähendada vajadust täiendavate interaktsioonide järele. Mitme tudengi vaates hinnati kõrgelt uue esituste ajaloo otsingu ja filtreerimise võimalusi, ning kuvada detailseid testimistulemusi ilma vajaduseta täiendavateks päringuteks Moodle'i suunas.

Lisaks tõid mõned testkasutajad välja mõtte, et süsteemis võiks olla olemas ka võimalus otsida esitusi commit hash'i alusel, mis lihtsustaks spetsiifiliste esituste leidmist.

Kokkuvõttes leidsid testkasutajad, et arendatud lahendused parandasid süsteemi läbipaistvust, kiirust ning kasutusmugavust, muutes süsteemi töökindlamaks ja efektiivsemaks nii tavakasutajate kui ka õppejõudude jaoks.

6 Tulemuste analüüs

Käesolevas peatükis analüüsitakse lõputöö käigus tehtud muudatuste mõju süsteemi töökindlusele, kasutusmugavusele, hooldatavusele ja skaleeritavusele. Iga peatükk põhineb vastaval tulemuste alapeatükil ning keskendub probleemi tuvastamisele, rakendatud lahendusele ja selle mõjule lõputöö eesmärkide täitmise seisukohalt.

6.1 Dokumentatsiooni mõju arendusprotsessile

Arendustöö alguses ilmnes, et kogu süsteemil puudus dokumentatsioon, mis teeks võimalikuks selle lokaalse ülesseadmise. Süsteemi teenused olid keerukad, koosnedes mitmest komponendist, millel olid erinevad tarkvaralised sõltuvused ja konfiguratsioonid. Sobivate Node.js, Python, Java ja Ubuntu versioonide ning õigete keskkonnamuutujate tuvastamine võttis töö autoritel ligi 20 tundi. Tegemist oli ebatõhusa, katse-eksituse meetodil põhineva protsessiga, mille läbimine eeldas tugevat tehnilist tausta.

Selle olukorra analüüs näitas, et dokumentatsiooni puudumine ei olnud üksnes ajakulu probleem, vaid süstemaatiline risk kogu arendusprotsessi stabiilsusele ja jätkusuutlikkusele. Iga uue arendaja lisandumine tähendanuks sama protsessi kordamist nullist, mis soodustaks vigade tekkimist, looks sõltuvuse olemasolevatest meeskonnaliikmetest ning raskendaks teadmiste ülekannet. Lisaks muutis see CI/CD töövoogude mõistmise ja haldamise ebakindlaks.

Probleemi lahendamiseks loodi töö käigus põhjalik ja struktuurne dokumentatsioon. See sisaldab süsteemi arhitektuuriskeemi, detailsed juhised arenduskeskkonna ülesseadmiseks, nõutud tarkvarakomponentide versioonid, keskkonnamuutujate konfiguratsioonifailide näidised ning CI/CD konfiguratsioonide selgitused. Samuti lisati soovitused tüüpiliste tõrgete vältimiseks, mis põhinevad praktilisel kogemusel süsteemi töölepanekul.

Dokumentatsiooni loomine parandas oluliselt süsteemi hooldatavust ja vähendas riske, mis kaasnesid teadmiste kadumisega või meeskonnavahetustega. Kui varem oli süsteemi

töölepanek aeganõudev ja ebastabiilne, siis nüüd on see korduvkasutatav ja prognoositav. Töö täitis selgelt eesmärgi tagada süsteemi taasluuesuutlikkus ja arendajate kiire sisenemine projekti. Ainus kitsaskoht, mis vajab edaspidi tähelepanu, on dokumentatsiooni pidev ajakohastamine, et säilitada selle praktiline väärtus ka tulevikus.

6.2 Arendusserveri roll arenduse ja testimise eraldamises

Arenduse alguses kasutati testimiseks lokaalset- ja tootmiskeskkonda. See tähendas, et muudatusi testiti kontekstis, kus need võisid tahtmatult mõjutada aktiivseid kasutajaid või rikkuda olemasolevaid andmeid. Selline olukord seadis süsteemi töökindluse ohtu ja muutis arenduse ebaturvaliseks.

Probleemi analüüs näitas, et arendus- ja testimiskeskkonna puudulik eristatus oli arenduse kvaliteedi jaoks selge riskifaktor. Ilma turvalise katsetuskeskkonnata ei ole võimalik muudatusi usaldusväärselt valideerida ega tagada, et need ei põhjustaks regressiooni tootmises. Lisaks raskendas see automatiseeritud CI/CD töövoogude jälgimist ja tegi muudatuste mõjust arusaamise keeruliseks.

Lõputöö käigus loodi eraldatud arendusserver, kuhu iga muudatus paigaldatakse automaatselt juhul, kui see toimub väljaspool põhi haru. Selle lahendusega eraldati arendus- ja tootmiskeskkonnad täielikult. Uus server võimaldab testida funktsionaalsust reaalsetele tingimustele sarnases keskkonnas, ilma et see mõjutaks tootmiskasutajaid. Lisaks parandati CI/CD töövoog, mis varasemalt startis eesrakenduse lokaalses arendusserveris, mis oli ebaturvaline. Nüüd käituvad töövood järjepidevalt ja ennustatavalt.

Selle muudatuse tulemusel paranes arenduse usaldusväärsus ja kiirus: probleemid ilmnevad varakult ning neid saab lahendada enne tootmisesse jõudmist. Samuti vähenes risk, et poolik või testimata kood jõuab aktiivse kasutajani. Eraldi testimiskeskkond võimaldab ka edaspidi paremat koostööd mitme arendaja vahel ning on eelduseks automatiseeritud testide kasutuselevõtuks tulevikus.

Arendusserveri loomine täitis otseselt töö eesmärgi suurendada süsteemi töökindlust ning vastas mittefunktsionaalsetele nõuetele, mis puudutasid turvalisust ja skaleeritavust. Töövoog muutus läbipaistvamaks ning arendusprotsess ennustatavamaks. Ainus jätkusuutlikkusega seotud aspekt on vajadus hooldada arendusserverit pikemaajaliselt, kuid selle

kulud on selgelt õigustatud saavutatud stabiilsuse ja ohutuse kasvu tõttu.

6.3 Ubuntu versiooni uuendamine ja selle mõju süsteemi töökindlusele

Uuendusprotsess viidi läbi sujuvalt ja planeeritult, ilma katkestusteta lõppkasutajatele. Tänu eelnevalt koostatud plaanile ja liikluse ajutisele ümbersuunamisele oli võimalik vähendada seisakuid ja maandada riske. Uuenduse ajastamine õhtusele ajale aitas vähendada häireid süsteemi tavapärases kasutuses.

Versiooniuuendus tagab, et süsteem töötab edasi toetatud ja turvalisel platvormil. See loob paremad eeldused edasiseks hoolduseks ja tarkvaraarenduseks, kuna uue versiooniga on saadaval uuemad teegid ja arendusvahendid. Süsteemi töökindlus ja turvalisus paranesid. Uuendusprotsess kinnitas, et olemasolev infrastruktuur võimaldab riskivabu hooldustöid ilma olulise kasutajamõjuta.

Liikluse ajutine suunamine arendusserverisse võimaldas lisaks valideerida ka selle toimimist reaalsetes tingimustes. Selle tulemusena kinnitati, et arendusserver suudab vajadusel toimida tagavaraserverina, tagades süsteemi järjepidevuse ka ootamatute rikete või planeerimata hoolduste korral.

6.4 Prioriteetidel põhineva järjekorra süsteemi tõhusus

Enne muudatuste tegemist toimus testimine esitamiste saabumise järjekorras (FIFO-põhimõttel), sõltumata töö tähtsusest või kontekstist. Eksamiperioodidel, kui süsteem sai lühikese aja jooksul tuhandeid esitamisi, kasvas koormus märkimisväärselt ja tudengid pidid ootama testimist ebaühtlaselt. Süsteem ei võimaldanud arvestada olukordi, kus kiire tagasiside oli kriitiline, näiteks eksamite või kontrolltööde puhul. Lisaks lisasid tudengid sageli korduvesitusi lootuses, et uus esitus jõuab testimisse kiiremini, mis omakorda suurendas koormust ja tekitas järjekorras ummikuid.

Probleemi analüüsi põhjal selgus, et testimiskiirust ennast ei olnud realistlikult võimalik oluliselt suurendada – piiravaks jäi ülesannete keerukus ja nende täitmise kestus (keskmiselt 8 sekundit). Seetõttu tuli optimeerida ülesannete järjekorda, et saavutada mõistlikum ressursikasutus ja õiglasem testimisvoog. FIFO-mudel eiras tööde konteksti ja ei võimaldanud süsteemil dünaamiliselt reageerida koormuse tippudele.

Lõputöö käigus rakendati uus järjekorrasüsteem, mis toetab ülesannetele määratavaid prioriteete vahemikus 1 kuni 10. Nüüd saavad õppejõud tähtsatele töödele, nagu eksamid või kontrolltööd, määrata kõrgema prioriteedi, mis tagab nende kiirema testimisse pääsemise. Samas ei rakendatud ranget prioritiseerimist, kus madalama prioriteediga tööd jääksid täielikult blokeerituks – süsteem liigub piiratud koguse tööde järel ka madalama prioriteediga ülesannete juurde. See hoiab ära olukorra, kus tavalised kodutööd ei pääse üldse testimisse.

Lisaks optimeeriti tudengi duplikaatesituste käsitlemist. Nüüd eelistatakse viimast esitamist varasematele ning uue esituse korral vähendatakse automaatselt eelnevate prioriteeti. See distsiplineerib kasutajat mitte koormama süsteemi tarbetult korduvate esitamistega ja loob õiglasema olukorra teiste tudengite suhtes.

Uus lahendus suurendas testimisprotsessi läbipaistvust ja efektiivsust. Tudengitele tagatakse tähtsates olukordades kiire tagasiside, vähendades stressi ja tehnilisi takistusi eksami sooritamisel. Süsteemi üldine reageerimisvõime suurenes – mitte testimissuutlikkuse kasvu tõttu, vaid selle nutikama jaotamise kaudu. Lõputöö lahendus täitis eesmärgi parandada süsteemi koormustaluvust ja kasutajakogemust, samal ajal säilitades õiglustunde testimisvoo käsitlemisel.

6.5 Monitooringu täiustuste mõju

Süsteemi monitooringu ja läbipaistvuse parandamine oli lõputöö üks olulisi eesmärke, mis toetas nii kasutajamugavust kui ka süsteemi töökindlust. Järgnevad alapeatükid käsitlevad konkreetseid muudatusi, mis võimaldasid süsteemi paremat jälgitavust, läbipaistvamat tagasisidet ning kiiremat tõrkeanalüüsi. Iga täiustus tõi kaasa mõõdetava positiivse mõju tudengite, õppejõudude ja süsteemi administraatorite kogemusele.

6.5.1 UNI-ID autentimise kasutuselevõtt eesrakenduses

Süsteemi varasem versioon ei olnud integreeritud TalTechi keskse autentimissüsteemiga. See tähendas, et kasutajate haldus ja ligipääsuõigused tuli lahendada lokaalselt – registreerimine, rollide määramine ja sisselogimine tuli hallata manuaalselt või ebaturvaliste mööndustega. Selline lahendus suurendas turvariske ning halduskoormust, eriti olukorras, kus süsteemi sooviti kasutada väljaspool väikest arendustiimi.

Probleemi analüüs näitas, et ilma keskse autentimiseta ei olnud võimalik tagada, et ainult TalTechi tudengid ja töötajad pääsevad süsteemile ligi, samuti puudus mehhanism õiguste automaatseks määramiseks. See tekitas olukorra, kus administraator pidi käsitsi sekkuma, et tagada ligipääs, mis on suur takistus süsteemi skaleeritavusele ja usaldusväärsele kasutusele.

Lõputöö käigus integreeriti süsteemi TalTechi keskne autentimine MSAL ja OpenID protokollide abil. Nüüdsest saavad kasutajad siseneda süsteemi oma ülikooli kontoga, mis võimaldab automaatselt määrata rolli ning seostada esitused kindla isikuga. Samuti registreeritakse uued kasutajad süsteemis vaikimisi madalaima õigustasemega, välistades käsitsi konto loomise vajaduse.

Selle muudatuse tulemusel vähenes oluliselt halduskoormus ning süsteemi turvalisus suurenes. Lisaks muutus ligipääs mugavamaks ja kiiremaks nii tudengitele kui ka õppejõududele. Autentimine on nüüd standardiseeritud, mis loob eeldused süsteemi usaldusväärseks kasutamiseks suuremates ja mitmekesisemates kasutajagruppides. Samas säilitati ka olemasolev lokaalne kasutajapõhine autentimine, sest seda on vaja Charoni ja ülikooli Gitlabi jaoks.

6.5.2 Esituste ajaloo ja filtreerimise mõju

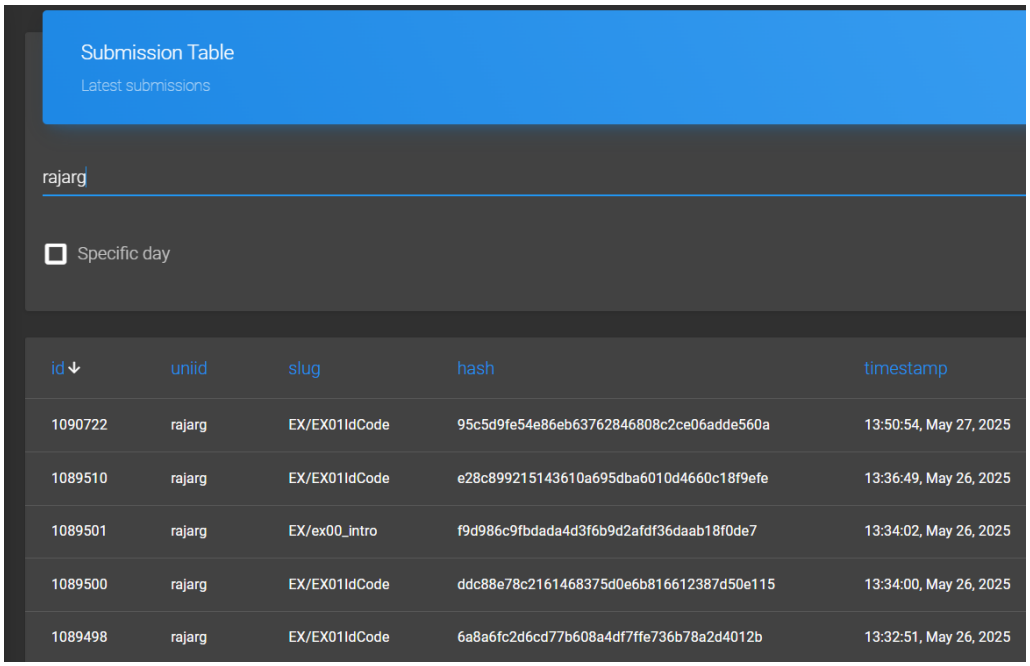
Süsteemi varasem versioon võimaldas kuvada ainult viimased 1000 esitust, ilma põhjalike filtreerimis- või otsinguvõimalusteta.

Probleemi analüüsi käigus selgus, et ajalooliste andmete puudulik kättesaadavus takistas tagasivaatavat analüüsi, süsteemsete vigade tuvastamist ja õiglase hindamise tagamist. Lisaks tähendas ligipääsupiirang, et paljud õppejõud pidid esituste nägemiseks pöörduma süsteemi administraatori poole, mis aeglustas töövoogu ja suurendas manuaalset koormust.

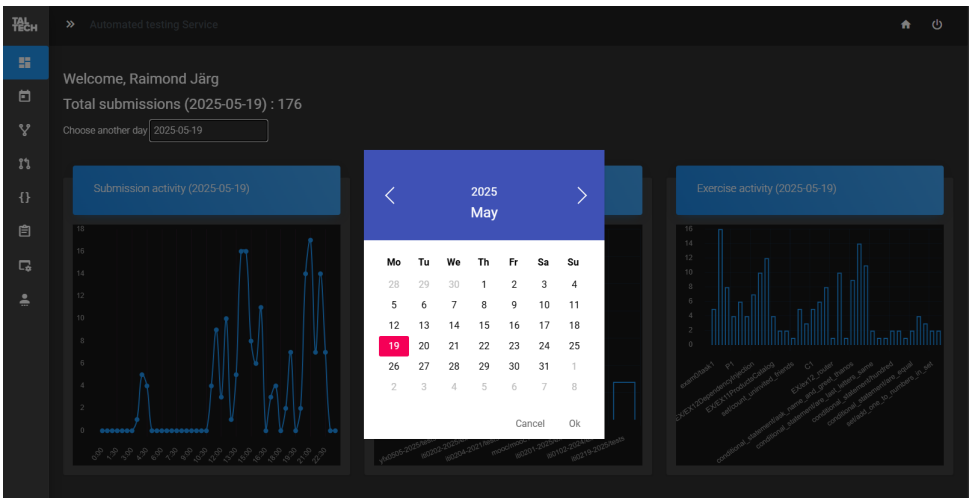
Lõputöö käigus täiustati esituste kuvamise loogikat. Nüüd on nähtav kogu esituste ajalugu ning lisatud on võimalus filtreerida andmeid tudengi, ülesande, kuupäeva ja muude kriteeriumide alusel (vt joonis 3). Lisaks kehtestati rollipõhine ligipääs: tudengid näevad ainult enda esitusi, samas kui õppejõud ja administraatorid saavad vaadata kogu kursuse andmeid. Selle võimaldas varem juurutatud keskne UNI-ID autentimine.

Muudatuse tulemusel paranes süsteemi läbipaistvus ja kasutusmugavus oluliselt. Õppejõud saavad nüüd iseseisvalt lahendada olukordi, kus tudeng väidab, et süsteem ei andnud taga-

sisidet või esitus jäi „kadunuks“. Ka tudengitel on nüüd parem ülevaade oma varasematest töödest. Lisaks lihtsustab täiustatud filtreerimine vigade ja süsteemsete mustrite tuvastamist (vt joonis 4). See on oluline samm süsteemi tõsiseltvõetavuse ja analüüsivõimekuse suunas.



Joonis 3. Kogu esituste ajaloo filtreerimine eesrakenduses koos otsinguga



Joonis 4. Esituste graafikud päevade lõikes koos kuupäeva filtreerimisega

6.5.3 Keskse logihalduse infrastruktuuri ettevalmistus

Süsteemi algses versioonis ei olnud logihaldus tsentraliseeritud. Iga teenus logis andmeid lokaalselt oma konteineri või serveri piires, mistõttu vigade ja tõrgete tuvastamine nõudis käsitsi logide kogumist erinevatest allikatest. See muutis veaotsingu aeganõudvaks,

killustatuks ja vähendas süsteemi töökindlust. Samuti puudus võimalus saada süsteemi üldisest seisundist koondülevaadet, mis on oluline nii jooksvaks monitooringuks kui ka probleemide ennetamiseks.

Töö käigus viidi läbi olemasoleva olukorra analüüs, mis tõi esile logide hajutatuse kui olulise puuduse süsteemi hooldatavuse ja jälgitavuse seisukohalt. Tsentraliseeritud logihalduse puudumine takistas korduvate probleemide automaatset tuvastamist, ajaloolise konteksti loomist ning visuaalsete monitooringutööriistade ja häiresüsteemide kasutuselevõttu.

Kuigi lõputöö raames ei jõutud keskse logihalduse süsteemi juurutamiseni, loodi teoreetiline ja arhitektuurne alus edasiseks arenduseks. Määratleti olulised logiformaadid, standardiseeritavad logiväljad ning võimalikud logiedastuse mehhanismid erinevate teenuste vahel. Need ettevalmistused loovad selge lähtepunkti, millele tuginedes saab tulevikus tsentraliseeritud logihalduse süsteemi juurutada.

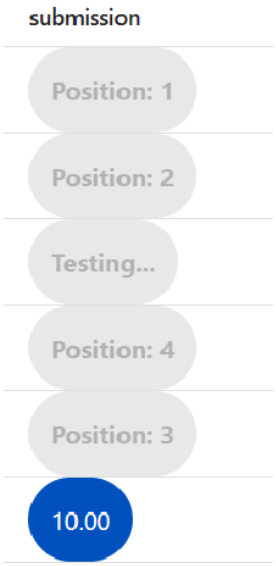
Seega ei ole keskne logihaldus veel kasutusele võetud, kuid analüüsi ja arhitektuurse kavandamise kaudu on loodud tugev vundament. See võimaldab tulevikus liikuda süsteemi hooldatavuse ja töökindluse uuele tasemele, kus tõrkeanalüüs on kiirem, seire mehhanismid usaldusväärsemad ning auditeerimine efektiivsem.

6.5.4 Järjekorra reaalajas kuvamise kasulikkus

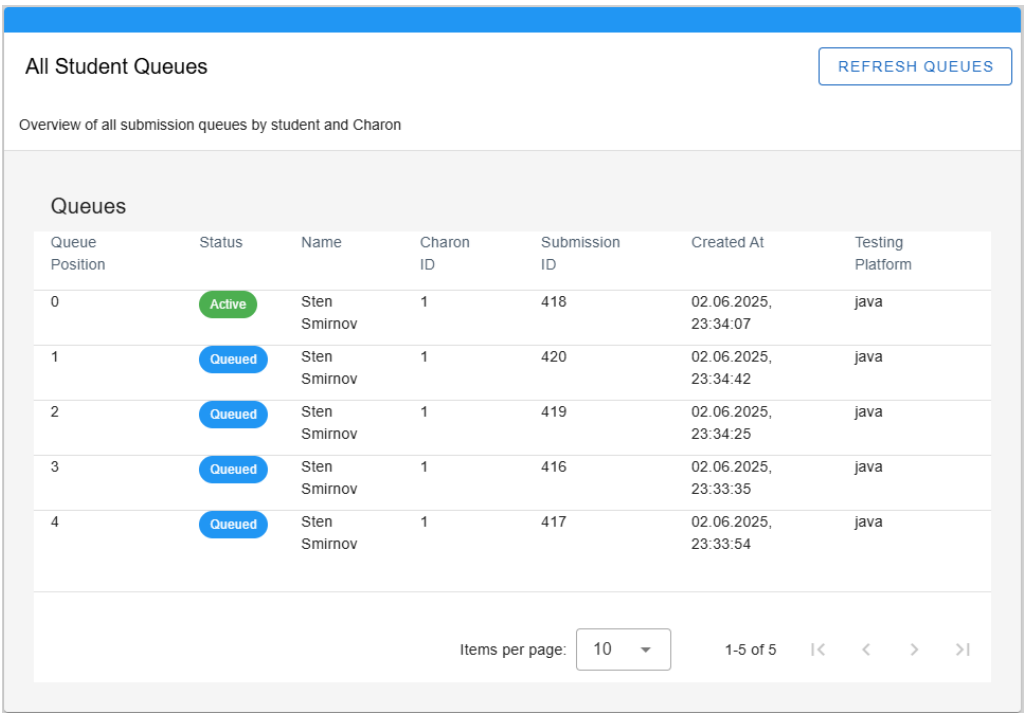
Varasemas süsteemiversioonis puudus tudengitel igasugune nähtavus oma lahenduse positsiooni kohta testimisjärjekorras. See põhjustas ebakindlust: tudengid ei teadnud, kas nende esitus on üldse jõudnud süsteemi, millal see testimisse pääseb või miks tulemus viibib. Selle tagajärjel esitati lahendusi korduvalt, lootes kiirendada testimist, mis omakorda suurendas süsteemi koormust ja lõi järjekorda tarbetut tihedust. Sama kehtis ka õppejõudude puhul, kellel puudus ülevaade kogu kursuse järjekorrast ning seega ka võimalus hinnata süsteemi hetkeseisundit.

Probleemi analüüs näitas, et nähtamatu järjekord lõi kasutajakogemuses usalduslõhe. Süsteem toimis küll tehniliselt korrektselt, aga kasutaja ei saanud kinnitust, et süsteem teda „nägi“. See mõjutas süsteemi kasutatavust ja suurendas tarbetut interaktsiooni koormust serveri poolel.

Lõputöö käigus loodi lahendus, mis võimaldab tudengitel reaalajas näha oma esitatud töö positsiooni testimisjärjekorras (vt joonis 5). Lisaks lisati õppejõududele võimalus vaadata kogu kursuse järjekorra seisu, mis aitab neil mõista testimise hetkekoormust ning vajadusel reageerida ummikutele (vt joonis 6). See info kuvati olemasolevas Charoni keskkonnas visuaalselt arusaadaval kujul.



Joonis 5. Tudengi vaade esituse positsioonist testimisjärjekorras



Joonis 6. Õppejõudude vaade kõigile testimisjärjekordadele Charonis

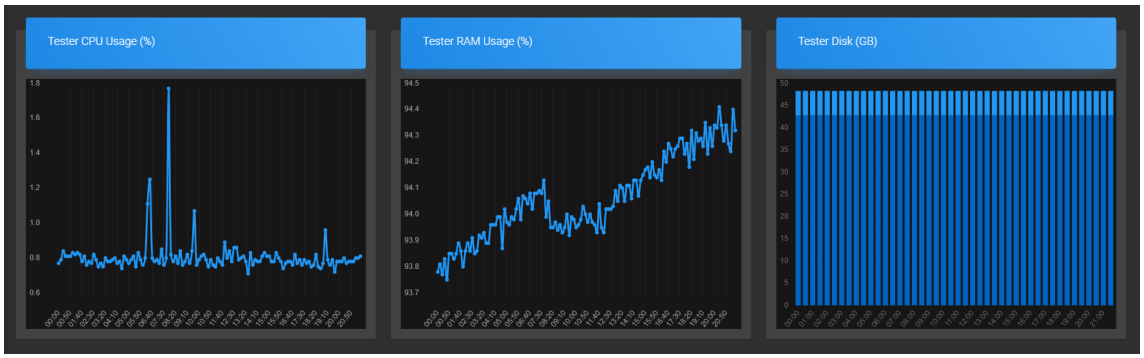
Muudatuse tulemusena vähenes korduvate esitamiste arv, kuna tudengid said kinnituse, et nende töö on süsteemis ja liigub edasi. Samuti paranes süsteemi läbipaistvus ja kasutajate rahulolu. Õppejõududel tekkis parem kontroll tööde käekäigu üle, mis omakorda vähendas vajadust pöörduda süsteemi administraatori poole. Kuigi muudatus ei ole veel tootmiskeskkonnas täielikult rakendunud (sõltuvalt Moodle'i haldusest), on lahendus tehniliselt valmis ja ootab juurutust.

6.5.5 Ressursikasutuse andmete visualiseerimine

Süsteemi varasem versioon võimaldas vaadelda ressursside – CPU, muutmälu (RAM) ja kettaruumi – kasutust ainult hetke seisundi põhjal. Ajalooliste andmete puudumine raskendas jõudlusprobleemide tuvastamist ning takistas süsteemi koormusmustrite ja varem ilmnenud tõrgete analüüsi. Samuti puudus võimalus hinnata ressursikasutust konkreetsetel ajahetkedel, näiteks eksamiperioodidel või testimiskoormuse tipphetkedel.

Analüüsi tulemusel selgus, et süsteemi töökindluse ja optimeerimise seisukohalt oli oluline lisada mehhanism ajalooliste jõudlusandmete kogumiseks ja visualiseerimiseks. Lõputöö käigus juurutati lahendus, kus CPU ja RAM-i kasutuse andmeid kogutakse cron-töö abil iga 10 minuti järel ning kettakasutuse andmeid iga 30 minuti järel. Need salvestatakse andmebaasi ja hoitakse seal 90 päeva vältel, et tagada piisav ajalugu ilma liigse salvestusruumi koormuseta. Andmete visualiseerimine toimub Vue Charts'i abil, võimaldades kasutajatel intuitiivselt jälgida ressursside kasutust ajas (vt joonis 7) [17].

Muudatuse tulemusel paranes süsteemi jälgitavus ja läbipaistvus. Kuigi koormusanalüüsi seosed reaalsete juhtumitega ei olnud töö fookuses, loob uus lahendus selge aluse jõudlusprobleemide tuvastamiseks ja analüüsimiseks. Samuti pakub see süsteemi halduritele võimalust hinnata, kas on vaja täiendavaid ressursse või süsteemi skaleerimist. Lahendus toetab otseselt töökindluse suurendamist ja loob eelduse ennetavaks tõrkeanalüüsiks tulevikus.



Joonis 7. Tester-serveri CPU, RAM ja ketta kasutuse graafikud 24 tunni jooksul

6.5.6 Rollipõhise autoriseerimise ja rollihalduse mõju turvalisusele ja kasutusmugavusele

Kuigi süsteem tagas tehnilisel tasemel, et kasutajad ei saanud serveri kaudu ligi andmetele või toimingutele, millele neil puudus õigus – ei olnud kasutajaliideses rakendatud rollipõhist nähtavuse kontrolli. Seetõttu kuvati kõigile autentitud kasutajatele kõik vaated ja funktsionaalsed komponendid, olenemata nende tegelikust rollist. Näiteks võis tavakasutaja näha administraatori sektsioone või õppejõu funktsioone, kuigi nendele toimingutele juurdepääs serveri kaudu puudus.

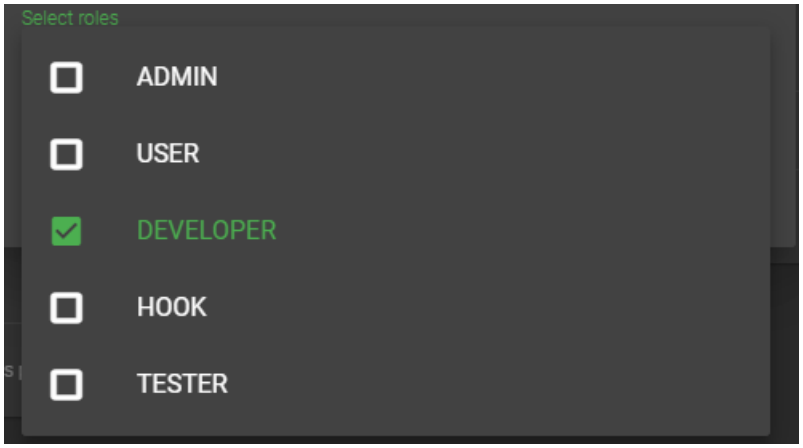
Lisaks ei sisaldanud süsteem võimalust rolle hallata. Rollid olid eelnevalt fikseeritud ning neid ei olnud võimalik süsteemi sees muuta, lisada ega eemaldada. Kõik rollid määrati staatiliselt konto loomisel ja nende muutmiseks oleks tulnud seda manuaalselt andmebaasist teha.

Lõputöö käigus viiakse süsteemi sisse dünaamiline rollihaldus, mis võimaldab igale kasutajale määrata sobiva rolli (nt tavakasutaja, administraator) ning vajadusel neid rolle jooksvalt muuta (vt joonis 8). Rollid salvestatakse andmebaasis ja nendega seotud õigused rakendatakse automaatselt autentimistokeni põhjal.

Eesrakenduses rakendatakse rollipõhine nähtavuse kontroll, mille käigus kasutajale renderdatakse ainult need komponendid ja funktsioonid, mis vastavad tema rollile. Näiteks USER rolliga kasutajale ei kuvata enam seadistuste vaadet ega administraatoripaneeli. See loogika vähendab võimalust ekslikuks navigeerimiseks ja kõrvaldab vajaduse lisakontrolliks liidese tasemel.

Mõju avaldub mitmel tasandil. Esiteks muutub süsteem oluliselt paindlikumaks, kuna

rollide muutmine või eemaldamine ei eelda enam seda manuaalselt andmebaasist tehes. Teiseks väheneb administraatorite halduskoormus ning minimeeritakse oht, et kasutajad saavad ekslikult valed õigused. Kolmandaks paraneb süsteemi kasutusmugavus ja turvalisus, kuna igale kasutajale kuvatakse ainult talle asjakohane info ja funktsionaalsus, mis vähendab segadust ning kõrvaldab vajaduse pimesi kontrollida ligipääse serveri tasemel.



Joonis 8. Rollide määramine administraatori poolt kasutajaliideses

6.6 Ilmnenud probleemide lahenduste mõju töökindlusele

Arenduse käigus ilmnes mitmeid süsteemseid ja kriitilisi probleeme, mille lahendamine oli vältimatu süsteemi töökindluse ja kasutajakogemuse säilitamiseks. Järgnevad alapeatükid toovad välja olulisemad probleemid ning hindavad nende lahendamise mõju süsteemi stabiilsusele, usaldusväärsusele ja skaleeritavusele. Probleemide kaardistamine ja kõrvaldamine näitasid, et süsteem arenes mitte ainult funktsionaalsuse, vaid ka kvaliteedinäitajate poolest.

6.6.1 Ajutiste tööfailide puhastuse mõju süsteemi stabiilsusele

Arenduse käigus ilmnes, et iga testimise järel jäid serverisse alles ajutised tööfailid, mis ei kustutatud automaatselt. Aja jooksul hakkasid need täitma serveri kettaruumi, mille tulemusel muutus süsteem ebastabiilseks ja nõudis käsitsi sekkumist. Selle probleemi analüüs näitas, et tegemist oli süsteemse puudusega, mis otseselt ohustas töökindlust ja katkestas teenuse töö mitu korda.

Lahendus seisnes automaatse puhastusmehhanismi loomises, mis kustutab testimise järel kõik mittevajalikud ajutised failid. Selle tulemusel muutus süsteem isemajandavaks ja

stabiilsemaks — see ei sõltu enam käsitsi hooldusest ning suudab töötada pikema aja jooksul ilma halduskoormuseta. Probleemi kõrvaldamine parandas süsteemi töökindlust oluliselt ja kõrvaldas korduva ohu allika.

6.6.2 GitHubi esituste korrektne tuvastamine

Pärast ühe Charoni uuenduse juurutamist ilmnes, et GitHubi kaudu tehtud esitusi ei seostatud korrektselt konkreetsete tudengitega. Selle tulemusel jäid need esitamised ilma tagasisidest ja hindest, mis kahjustas süsteemi usaldusväärsust, eriti väljaspool TalTechi keskkonda kasutamisel.

Probleemi analüüs näitas, et puudulik identifitseerimisloogika takistas süsteemi laiendamist laiematele kasutajagruppidele, nagu partnerkoolid ja eksternid. Lahenduseks loodi mehhanism, mis võimaldab GitHubi esitamisi täpselt siduda kasutajaga, kasutades unikaalseid tunnuseid ja valideerimiskriteeriume.

Selle muudatuse järel on võimalik toetada esitamisi ka siis, kui tudeng ei kasuta TalTechi GitLabi. See suurendab süsteemi paindlikkust ja avab võimalusi kasutamiseks mitmesugustes hariduskeskkondades. Samuti paranes tagasiside ja hindamise täpsus, mis on oluline hindamise õigusprintsiibi seisukohalt.

6.6.3 Commit hash'i eiramisest tekkinud vea kõrvaldamine

Testimissüsteemi käitumises ilmnes probleem, kus kordustestimisel ei kasutatud alati seda koodiversiooni, millele hindamispäring viitas, vaid testiti hoopis viimast olemasolevat versiooni. See lõi olukorra, kus tagasiside ei vastanud enam konkreetsele ülesandeversioonile ja muutis tulemuse ebausaldusväärseks.

Probleemi analüüs tõi välja vea versioonikontrolli käsitlemises. Lahendus seisnes selles, et kordustestimise käivitamisel seotakse nüüd test täpselt selle konkreetse commit'i või esitusfailiga, mis algses esitamises kasutati. See parandas testide reprodutseeritavust ja usaldusväärsust.

Mõju oli selgelt positiivne: nüüd saab iga esitatud töö uuesti testida täpselt samades tingimustes, mis võimaldab paremat veaotsingut, tagantjärele kontrolli ja õiglasemat hindamist.

6.6.4 Sisendandmete sidumise täpsuse paranemine

Mõnel juhul ei olnud testitulemustes võimalik üheselt tuvastada, millise sisendandmestiku alusel test toimus. See põhjustas olukordi, kus tudeng või õppejõud ei saanud aru, miks konkreetne test ebaõnnestus, sest ei olnud selge, milliseid sisendeid kasutati.

Analüüsi käigus selgus, et väikesed ajanihted ja andmeedastuse ebakindlus takistasid korrektset seostamist. Lahendusena muudeti testimisprotsessi nii, et iga testimiskord salvestab nüüd sisendfailide ID ja versiooni koos esituse metainfoga.

See parandas süsteemi läbipaistvust ning võimaldab tudengitel ja õppejõududel täpselt analüüsida, miks mingi test läbi kukkus. Samuti võimaldab see usaldusväärsemat vigade otsingut ja testide kordamist täpselt samadel tingimustel.

6.6.5 Järjekorra andmete dubleerimise probleemi lahendamise mõju

Mõnel juhul põhjustas tööde järjekorra vaatamine olukorra, kus iga andmepäring lisas juba olemasolevad tööd serveri mälusisesele andmestruktuurile uuesti. See viis olukorrani, kus iga päring kasvatas tööde koguarvu mälus eksponentsiaalselt.

Analüüsi käigus selgus, et probleem tekkis andmete töötlemise loogikast, mis ei kontrollinud piisavalt, kas tööd on juba andmestruktuuris olemas. Sagedaste päringute korral suurendas see mälukasutust ja protsessori koormust, mis suure koormuse korral viis süsteemi jõudluse languseni ning suurendas tõrgete ohtu. Probleemi lahendamiseks muudeti päringute töötlemise loogikat, et välistada tööde dubleerimine mälus. Iga töö lisatakse andmestruktuuri nüüd vaid üks kord, sõltumata päringute arvust.

Selle muudatuse tulemusena stabiliseerus süsteemi ressursikasutus ka suurte päringusageduste korral, paranes süsteemi töökindlus ja vähenes ülekoormuse tekkimise risk.

6.7 Võimalused edasiarenduseks

Süsteemi arendamisel ilmnes mitmeid kohti, mida on võimalik tulevikus täiendada. Üheks olulisemaks suunaks on logide tsentraalse halduse lõpuleviimine, mis lihtsustaks vigade jälitamist ja süsteemi monitooringut.

Samuti oleks kasulik kirjutada esirakendus täielikult ümber, et parandada koodi struktuuri ja laiendatavust. Keelepõhiseid testreid oleks mõningal määral võimalik optimeerida, et suurendada testimise kiirust ja vähendada süsteemi koormust.

Funktsionaalsuse tasandil võiks lisada piirangu ühe tudengi esituste arvule ja võimaldada testimises või järjekorras olevate tööde tühistamist. Lisaks võiks kasutajaliides anda automaatseid soovitusi vigaste esituste parandamiseks.

7 Kokkuvõte

Käesoleva lõputöö eesmärgiks oli tõsta Tallinna Tehnikaülikooli programmeerimisainetes kasutatava Arete testimissüsteemi töökindlust, läbipaistvust ja hooldatavust. Süsteem koosneb testimismootorist Arete ning Moodle'i pistikprogrammist Charon, mille kaudu saadetakse ja kuvatakse testimistulemused tudengitele ja õppejõududele.

Töö käigus teostati põhjalik analüüs olemasolevast süsteemist ning tuvastati mitmeid olulisi probleeme, mille seas olid ebastabiilne testimisjärjekord, puudulik tagasiside tudengitele, logide raskesti kättesaadavus, dokumentatsiooni puudumine, süsteemi uuendamata komponendid ja arenduse töövoogude killustatus. Nende probleemide lahendamiseks viidi töö käigus ellu mitmeid süsteemseid ja tehnilisi muudatusi, mille tulemusel muutus testimiskeskkond märgatavalt töökindlamaks, skaleeruvamaks ja kasutajasõbralikumaks.

Nende probleemide lahendamiseks:

- Loodi uus prioriteetidel põhinev testimisjärjekorra haldusloogika, mis eelistab hiljutisi ja tähtsamaid esitamisi ning väldib süsteemi ummistumist.
- Lisati tudengitele reaalajas nähtav positsioon testimisjärjekorras ja visualiseeriti testimise etapid, mis vähendas korduvate esituste hulka.
- Rakendati loogika, mis eelistab alati tudengi viimast esitamist ja alandab varasemate prioriteeti.
- Parandati sisendandmete seostamist esitustega ning lahendati probleemid GitHubi kasutajanimede ja UNI-ID tuvastamisega.
- Rakendati commit hash'i tugi, mis võimaldab testida konkreetset määratud koodiversiooni.
- Automatiseeriti ajutiste töökaustade kustutamine pärast testimist, ennetamaks kettaruumi täitumist ja jõudlusprobleeme.
- Täiustati monitooringut, lisades serveri ressursikasutuse ajalise jälgimise ja esituste ajaloo visualiseerimise võimaluse.

- Laiendati esituste statistikat ja eemaldati 1000-esituse piirang, lisades filtreerimise kuupäeva, kursuse, ülesande ja tudengi UNI-ID alusel.
- Lisati UNI-ID autentimine Microsoft Azure AD kaudu, mis lihtsustab kasutajahaldust ja turvab ligipääsu.
- Rakendati rollipõhine autoriseerimine, mis määrab kasutajale ligipääsu ulatuse vastavalt tema rollile ning võimaldab piirata nähtavaid vaateid ja toiminguid.
- Koostati põhjalik dokumentatsioon süsteemi lokaalseks käivitamiseks ja arendamiseks.
- Uuendati süsteem Ubuntu 22.04 LTS versioonile, et tagada turvalisus ja ühilduvus kaasaegsete tööriistadega.
- Eraldati arendus- ja tootmiskeskkonnad ning loodi CI/CD töövood turvaliseks juurutamiseks.
- Valmistati ette tsentraliseeritud logihalduse infrastruktuur, mis võimaldab kõigi teenuste logide koondatud jälgimist ühest kohast.

Tehtud muudatused loovad eeldused usaldusväärsemaks süsteemiks, vähendavad vigade esinemise võimalust ja parandavad märgatavalt kasutajakogemust nii tudengite kui ka õppejõudude jaoks.

Kasutatud kirjandus

[1] Enrico Vompa. *Arete testimissüsteemi dokumentatsioon*. [Kasutatud: 15-04-2025]. URL: https://ained.pages.taltech.ee/it-doc/arete/index.html.

[2] Enrico Vompa. *Charon hindamissüsteem dokumentatsioon*. [Kasutatud: 16-04-2025]. URL: https://ained.pages.taltech.ee/it-doc/moodle/charon/index.html.

[3] Broadcom Inc. *Guides*. [Kasutatud: 14-04-2025]. URL: https://spring.io/guides.

[4] Evan You. *Vue 3 Documentation*. [Kasutatud: 13-04-2025]. URL: https://vuejs.org/guide/introduction.html.

[5] Laravel. *Laravel 8 documentation*. [Kasutatud: 13-04-2025]. URL: https://laravel.com/docs/8.x.

[6] The PHP Group. *PHP Manual*. [Kasutatud: 12-04-2025]. URL: https://www.php.net/docs.php.

[7] Evan You. *Vue 2 Documentation*. [Kasutatud: 13-04-2025]. URL: https://v2.vuejs.org/v2/guide/.

[8] Canonical Ltd. *Ubuntu Release Cycle*. [Kasutatud: 15-04-2025]. URL: https://ubuntu.com/about/release-cycle.

[9] Evan You. *Deployment*. [Kasutatud: 18-04-2025]. URL: https://cli.vuejs.org/guide/deployment.html.

[10] Lovisa Johansson. *Part 1: RabbitMQ for beginners - What is RabbitMQ?* [Kasutatud: 14-04-2025]. URL: https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html.

[11] Broadcom Inc. *Classic Queues Support Priorities*. [Kasutatud: 15-04-2025]. URL: https://www.rabbitmq.com/docs/priority.

[12] GeeksforGeeks. *Starvation and Aging in Operating Systems*. [Kasutatud: 18-04-2025]. URL: https://www.geeksforgeeks.org/starvation-and-aging-in-operating-systems.

[13] Dave Stewart. *A guide to MSAL authentication in Vue*. [Kasutatud: 04-05-2025]. URL: https://davestewart.co.uk/blog/msal-vue/.

[14] Auth0. *OpenID Connect Protocol*. [Kasutatud: 30-05-2025]. URL: https://auth0.com/docs/authenticate/protocols/openid-connect-protocol.

[15] Broadcom Inc. *Spring Boot and OAuth2*. [Kasutatud: 15-04-2025]. URL: https://spring.io/guides/tutorials/spring-boot-oauth2.

[16] Yash Thakkar. *Docker centralized logging using Fluent Bit, Grafana and Loki*. [Kasutatud: 02-05-2025]. URL: https://dev.to/thakkaryash94/docker-container-logs-using-fluentd-and-grafana-loki-a15.

[17] Chart.js Contributors. *vue-chartjs Documentation*. [Kasutatud: 29-04-2025]. URL: https://vue-chartjs.org/guide/.

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

Meie, Raimond Järg ja Sten Smirnov

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Arete testimismootori edasiarendus", mille juhendaja on Bahdan Yanovich
    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.

04.06.2025

[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 – Järjekorra valideerimiseks kasutatud skript

```
import requests
import threading
import time

POST_REQUESTS_AMOUNT = 9
GIT_STUDENT_REPO = "https://gitlab.cs.taltech.ee/rajarg/iti0202-2023"
GIT_TESTS_REPO = "https://gitlab.cs.taltech.ee/rajarg/iti0202-2023-ex"

url = 'ARETE_URL_HERE'
token = 'ARETE_TOKEN_HERE'

headers = {
    'Authorization': f'Bearer {token}',
    'Content-Type': 'application/json'
}

def make_post_request(thread_id, student = None):
    time.sleep((thread_id * 700) / 1000.0)
    for i in range(0, POST_REQUESTS_AMOUNT):
        uniid = student
        if uniid is None:
            uniid = "student" + str(i)
        payload = {
          "testingPlatform": "java",
          "gitStudentRepo": GIT_STUDENT_REPO,
          "gitTestRepo": GIT_TESTS_REPO,
          "slugs": [
            "EX/EX01IdCode"
          ],
          "uniid": uniid,
          "dockerTimeout": "120",
          "priority": "10",
          "systemExtra": [
```

```
            "noFiles"
          ]
        }
        requests.post(url, headers=headers, json=payload)
        if student is None:
            time.sleep(9)
        else:
            time.sleep(5)
        print(f'Thread {thread_id} done sleeping.')

students = [
    "rajarg",
    "stsmir",
    "raimond.jarg",
    "sten.smirnov",
    "arete"
]
threads = []
for i in range(len(students) + 1):
    if i == len(students):
        t = threading.Thread(
            target=make_post_request,
            args=(i,)
        )
    else:
        t = threading.Thread(
            target=make_post_request,
            args=(i,students[i])
        )
    t.start()
    threads.append(t)

for t in threads:
    t.join()

print('All threads completed.')
```