TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Julia Djomina 206642IAIB
Ellina Gedrojets 205930IAIB

PROGRAMMEERIMISÜLESANNETE HALDAMISE REGISTRI AURORA TÄIENDUSED JA PARANDUSED

Bakalaureusetöö

Juhendaja: Ago Luberg
PhD

Tallinn 2023

Autorideklaratsioon

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

Autor: Julia Djomina, Ellina Gedrojets

23.05.2023

Annotatsioon

Selle lõputöö eesmärk on programmeerimisülesannete haldamise registri Aurora parandused ja edasiarendused. Lõputöö tellijaks on informaatika programmijuht Ago Luberg. Aurora on veebirakendus, mis on mõeldud TalTechi tarkvara instituudi õppejõududele, et hallata programmeerimiskursuste ülesandeid, pakkuda võimalust vaadata ülesandeid konkreetsete raskustega ja teemadega, lisada ülesannetele kommentaare ning vaadata statistikat ja tudengite esitatud ülesandeid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 54 leheküljel, 12 peatükki, 18 joonist ja ühte tabelit.

Abstract

Programming assignment management registry Aurora additions and corrections

Several TalTech courses use programming assignments to teach their subjects to the students. The number of assignments available for the course teaching staff to select from increases over the course of each semester. The management of sources may grow challenging over time. The Aurora application tries to solve the problem by keeping all assignments from different repositories in one application. There were already two SDLC passed during the Aurora application.

The aim of this thesis is to fix bugs and develop the Aurora registry further for managing programming assignments. The idea of the thesis is based on the client's interest to start using this application as soon as possible. The project is ordered by the Programme Director of Informatics in Tallinn University of Technology Ago Luberg.

The result of this development is a tested application that is running on the server smoothly with fixed bugs that prevented the application from working properly and implemented new features according to the needs and desires of the client.

The thesis is in Estonian and contains of text on 54 pages, 12 chapters, 18 figures and one table.

Lühendite ja mõistete sõnastik

API	Rakendusliides (*Application Programming Interface*)
BE	Tagarakendus (*Back-end*)
CI/CD	Pidev integratsioon ja pidev juurutamine/tarnimine (*Continuous Integration and Continuous Development*)
CSS	Astmelised stiililehed (*Cascading Style Sheets*)
DTO	Andmeedastusobjekt (*Data Transfer Object*)
FE	Eesrakendus (*Front-end*)
HTTP	Hüperteksti edastusprotokoll (*Hypertext Transfer Protocol*)
ORM	Objekti-relatsiooniline kaardistamine (*Object-Relational Mapping*)
Json	Andmevahetuse vorming (*JavaScript Object Notation*)
REST	Tarkvaraarhitektuuri laad (*Representational State Transfer*)
SASS	*Syntactically Awesome Style Sheets*
SDLC	Tarkvaraarenduse elutsükkel (*The software development lifecycle*)
SLF4J	*Simple Logging Facade for Java*
SPA	Üheleherakendus (*Single-page application*)
SQL	Struktuurpäringukeel (*Structured Query Language*)
TARA	Riigi autentimisteenus (*The State Authentication Service*)
UI	Kasutajaliides (*User interface*)

Sisukord

Joonistе loetelu

Tabelite loetelu

1. Sissejuhatus

TalTechi IT teaduskonna õpilasi õpetatakse programmeerima kasutades erinevaid programmeerimisülesandeid, mida õppejõud iga aasta tagant kopeerib vanadest salvedest. Korduvate ülesannete haldamine ja hoidmine ei ole mugav. Lisaks puudus õppejõududel platvorm, kust saaks vaadata kõikide aastate ülesandeid, lisada nende külge silte, nende järgi otsida ning lisada ülesannetele kommentaare.

Oskar Pihlaku lõputöö „Programmeerimisülesannete haldamise register Aurora“ raames loodud Aurora veebirakendus omas baasfunktsionaalsust, kuid palju tähtsaid osasid ei suudetud projekti suure mahu tõttu valmis saada [1].

Veebirakenduses puudus:

- Autentimine;
- Programmeerimisülesannete statistika;
- Kasutajate õiguste haldamine;
- Kommentaaride lisamine.

2022. aasta kevadel kaitses teine tiim bakalaureusetööd teemal „Programmeerimisülesannete haldamise register Aurora õiguste ja statistika süsteemi juurdearendus“ [2]. Selle töö eesmärk oli Aurora veebirakenduse edasiarendus. Eesmärk oli lisada autentimine ja vajadusel parandada olemasolevat funktsionaalsust, et rakendus oleks kasutuskõlblik.

Selle lõputöö raames sai tehtud:

- Autentimine läbi TalTech Azure Active Directory;
- Õiguste haldamise süsteem;
- Statistika kogumine:
  - Tabel, kus on kirjas kui palju ülesandes esitusi kokku.
  - Tabel, kus on kirjas ülesannet esitanud õpilaste arv.
- Kasutajaliides sai uue kujunduse ja uued tarkvaralised versioonid;

Vaatamata sellele, et rakendus oli arenduses kahe aasta jooksul, ei olnud seda siiani võimalik kasutada erinevate vigade ja puuduliku funktsionaalsuse tõttu.

1.1 Teema valik

Lõputöö teema valimise eesmärk oli tegeleda teemaga, mis oleks kasulik ülikoolile ja huvitav meie jaoks.

Oleme töötanud abiõppejõududena “Programmeerimise algkursus ITI0102” ja “Programmeerimise põhikursus ITI0202” ainetes alates 2022. aasta algusest. Meie põhiülesandeks oli valmistada tudengitele ette ülesandeid eelmiste aastate hoidlates olevate ülesannete näitel. Tavaliselt võttis see palju aega, sest vaja oli vaadata kõik hoidlad ükshaaval läbi ja ainult ülesande koodi vaadates otsustada, kas ülesanne sobib või mitte. Üldist süsteemi, mis hoiaks kõiki ülesandeid ning mis võimaldaks sobivat ülesannet lihtsasti üles leida, tol hetkel ei olnud. Hiljem, kui saime teada mis on Aurora rakenduse põhimõte, otsustasime, et soovime panustada selle rakenduse arendamisse.

Teiseks suureks põhjuseks selle teema valimisel olid tehnoloogiad, mille peal Aurora rakendus on ehitatud. Ülikoolis õppimise jooksul omandasime kogemust nende tehnoloogiate kasutamisel, seega tundsime ennast kindlalt ja teadsime, et saame hindamisväärset panust anda.

1.2 Olemasolev rakendus

Aurora rakenduses toimub kasutajate autentimine Azure AD teenuse kaudu. Eelmised arendajad otsustasid autentimisel kasutada Open Authorization 2.0 protokolli (OAuth 2.0) [3]. Protokoll defineerib neli rolli:

- Aurora kasutajaliides - kliendi roll;
- Kasutaja - ressursi omanik;
- Azure Active Directory - autoriseerimisserver;
- Aurora tagarakendus - ressursi server.

Aurora rakendus on mõeldud TalTechi töötajatele, mistõttu TalTechi enda Azure AD teenuse kasutamine oli väga mõistlik. Selle abil saavad Aurora rakenduses autentida ainult ülikooliga seotud isikud, kellel on kehtiv Uni-ID kasutajakonto. Kasutades Azure AD teenust, puudub vajadus hoiustada kasutaja paroole ja muud infot Aurora rakenduse andmebaasis, sest Azure AD vastutab selle eest ise.

Rakendus võimaldab kasutajal hallata GitLabi hoidlaid ja ülesandeid. Hoidlate ilmumiseks Auroras on eraldi loodud Aurora GitLab kasutaja.

Kui lisada Aurora kasutaja GitLab keskkonnas hoidlasse ja anda kas reporter (*reporter*), arendaja (*developer*), hooldaja (*maintainer*) või omanik (*owner*) roll, siis muutub hoidla Aurora registris nähtavaks ja tekib võimalus seda kloonida registri süsteemi. Joonisel 1 saab näha hoidlasse lisatud Aurora kasutajat hooldaja rolliga.

Joonis 1. *Projekti lisatud Aurora kasutaja hooldaja rolliga.*

Rakendus võimaldab otsida hoidlaid nime järgi ja sünkroniseerida neid GitLab keskkonnas olevate andmetega. Hoidlaid saab näha kas hoidlate vaates tabelina või avatud hoidla vaates ükshaaval. Avatud hoidla vaates on kuvatud kõik selles hoidlas olevad ülesanded.

Samuti saab rakenduses vaadata kõiki ülesandeid tabelina või ükshaaval. Avatud ülesande vaates on kuvatud, millises programmeerimiskeeles ülesanne on kirjutatud ning kui sellised leiduvad, siis kuvatakse ülesandeid, mis on avatud ülesandega sarnased. Kui avada ülesande vaates statistika, kuvatakse seal tehtud esituste ja vähemalt ühe esituse (*submission*) teinud õpilaste arvud. Esitused jõuavad Aurora rakendusse automattestri kaudu. Õpilased lahendavad ülesanded soovitud IDE-s (*integrated development environment*), saadavad lahendused Giti salve. Giti salvest saadetakse kood automaattestrile, mis omakorda saadab Aurorasse lahenduse tulemused.

Rakenduses saab luua erinevad silte ja märkida nendega programmeerimisülesandeid, et oleks võimalik otsida vajalikku ülesannet siltide kaudu. Iga silt kuulub kindlasse siltide gruppi. Gruppide eesmärk on grupeerida silte, et oleks lihtsam leida sobilik silt ülesannete märkimiseks.

Eelmiste arendajate poolt oli loodud ka õiguste haldamine. Kogu õiguste haldamine toimub Aurora rakenduse siseselt. Paindlikkuse huvides otsustati lisaks rollidele luua ka rollide grupid. Aurora admin saab määrata kasutajatele iga rakenduse siseste funktsionaalsuste osas CRUD (*create, read, update, delete*) õigusi eraldi. Kasutaja, kellel on olemas piisavalt õigusi, saab luua uusi rolle ning igale rollile lisada nime ja vajalikke õigusi rakendusse eelnevalt kirjutatud õiguste hulgast, mis olid eelnevalt rakendusse lisatud. Loodud rolle on võimalik muuta või kustutada [2].

1.3 Eesmärgid ja oodatav tulemus

Selle lõputöö eesmärgid ja oodatav tulemus on järgmised:

- Seni tehtud rakenduse analüüs, probleemide tuvastamine ja parandamine;
- Rakendus on kohati väga aeglane serveril, tuleb leida põhjus ja see parandada;
- Tagarakendus jookseb aeg-ajalt kokku, tuleb leida põhjus ja see parandada;
- Koguda ja kuvada statistikat;
- Ülesannetele kommentaaride lisamine;
- Praegu saab rakendust mugavalt kasutada suurtel ekraanidel, st telefoniga pole mugav kasutada. On vaja teha reageeriv kasutajaliides (*responsive UI*);
- Parandada keele valik. Rakenduses on kasutusel kaks keelt: eesti ja inglise. Ingliskeelne versioon ei ole kohati tõlgitud eesti keelde;
- Parandada otsing. Otsing ei anna õigeid tulemusi ning on väga aeglane;
- Parandada viga seoses hoidla kustutamisega. Hetkel saab rakendusse lisada hoidlaid, aga Aurora kasutaja kustutamisel GitLab'i salvest ei kao see salv rakendusest;

Lõputöö oodatav tulemus on tagada rakenduse pidev töö serveril, anda kasutajatele võimalus otsida ülesandeid, vaadata ülesannete esituste statistikat ja parandada osaliselt või täielikult mittetöötav funktsionaalsus.

2. Projekti kirjeldus

Selles peatükis kirjeldame arendustöös kasutatud metoodikaid.

2.1 Arenduses kasutatud metoodika

Projekti raames kasutasime agiilset tarkvara arendamise meetodit Scrum [4]. Scrum on üks paljudest populaarsetest agiilsetest arendusmeetoditest. Selle arendusmeetodi eelised on:

- Koostöö. Meeskond teeb koostööd sprindi eesmärkide määratlemiseks ja eesmärkide saavutamiseks.
- Pidev suhtlus.
- Tagasiside nii meeskonnaliikmete kui ka kliendi poolt.
- Väikeste sprintidega töötamine suurendab tõenäosust, et meeskond avastab vead võimalikult varakult.
- Aitab meeskondadel võtta suuri projekte ja jagada need väiksemateks hallatavateks osadeks.

2.2 Tegumikeskse kasutajaliidese disaini meetodi kasutamine

Ülesandepüstituses kirjeldatud eesmärkide saavutamiseks pidime leidma sobivad lahendused. Kasutajaliidese lahenduste leidmiseks kasutasime tegumikeskse kasutajaliidese disaini (*Task-centered User Interface Design*) [5], [6].

Tegumikeskne kasutajaliidese disain on lähenemine kasutajaliideste kujundamisele, mis keskendub ülesannetele, mida kasutajad peavad täitma. Selle lähenemisviisi kasutamisel järgitakse järgmisi samme:

- Kasutajate ülesannete määramine. Kasutaja ülesannete loendi koostamine, et määrata kindlaks, mida kasutaja soovib rakenduse abil saavutada.
- Kasutajate ülesannete analüüs. Iga kasutaja ülesande analüüsimine, et mõista kasutaja eesmärke, töövoogu ja nõudeid.
- Kasutajaliidese disain. Kasutajaliidese disaini prototüübi loomine vastavalt kasutaja ülesannetele, kasutades selgeid ja lihtsaid funktsioone või visuaali.
- Prototüübi testimine ja arutamine kasutajatega, veendumaks, et liides vastab nende

vajadustele ja on lihtne kasutada.

- Kasutajaliidese korduv testimine. Korduvate testide läbiviimine, et parandada liidese kasutatavust ja efektiivsust vastavalt kasutajate tagasisidele.

Tegumikeskne kasutajaliidese disain annab võimaluse luua kasutajaliidest vastavalt kasutaja vajadustele ja eesmärkidele ning tagada, et liides oleks lihtne ja intuitiivne kasutamiseks.

2.3 The Three Principal Layers printsiip

Lõputöö tarkvaraarenduse raames otsustasime järgida The Three Principal Layers [7] printsiipi, mida saab näha joonisel 2.

Layer	Responsibilities
Presentation	Provision of services, display of information (e.g., in Windows or HTML, handling of user request (mouse clicks, keyboard hits), HTTP requests, command-line invocations, batch API)
Domain	Logic that is the real point of the system
Data Source	Communication with databases, messaging systems, transaction managers, other packages

Joonis 2. *Three Principal Layers.*

Esitluskihi loogika (*presentation layer*) seisneb selles, kuidas käsitleda kasutaja ja tarkvara vahelist suhtlust. Esitluskihi peamised kohustused on kuvada infot kasutajale ja tõlgendada kasutaja käske domeeni ning andmeallika toiminguteks.

Domeenkihi (*domain layer*) loogika on töö, mida rakendus peab kasutatava domeeni jaoks tegema. See hõlmab arvutusi, mis põhinevad sisenditel ja salvestatud andmetel, esituskihist tulevate andmete valideerimisi ja täpselt välja selgitamist, millise andmeallika loogikat saata olenevalt esituskihist saadud käskudest.

Andmeallika (*data source layer*) loogika seisneb suhtlemises teiste süsteemidega, mis täidavad rakenduse nimel ülesandeid. Need võivad olla näiteks tehingumonitorid (*transaction monitors*) või muud rakendused. Enamiku ettevõtete jaoks on rakenduste suurim osa andmeallika loogikast andmebaas, mis vastutab peamiselt andmete ladustamise eest.

2.3.1 The Three Principal Layers printsiibi kasutamine arenduses

Varasemalt oli Aurora rakenduses vaid osaliselt järgitud The Three Principal Layers printsiipi. Näiteks ei vastanud sellele printsiibile see, et rakenduses oli kuvatud kasutajale kaks tabelit, mis näitasid esituste arvu ja keskmine läbivuse protsenti. Nende tabelite tulemuste arvutamise loogika oli kirjutatud esitluskihis, mis tähendab, et tagarakendusest olid saadetud objektid otse andmebaasist ilma andmeedastusobjektideta ning töödeldud ja kuvatud esitluskihis. Seda printsiipi järgides peaks arvutusloogika olema teostatud tagarakenduses ning ainult peale andmete töötlemist ja andmeedastusobjekti loomist saadetud esitluskihti kuvamiseks.

Oma arenduses jälgisime The Three Principal Layers printsiipi ning kui vana loogika oli seotud meie uue funktsionaalsusega, siis parandasime ka juba olemasolevat loogikat, et see vastaks antud printsiibile.

2.4 Andmeedastusobjektid

Tarkvara arenduses kasutatakse sageli andmeedastusobjekte (Data Transfer Object) [7], mis kannavad andmeid erinevate rakenduskihtide vahel, näiteks eesrakenduse ja tagarakenduse vahel. Tagarakenduses on andmeedastusobjektide kasutamine eriti oluline, kuna see mängib olulist rolli rakenduse hooldatavuse tagamisel. Andmeedastusobjektil on sageli palju omadusi (*property*), millel on olemas *getter*'id ja *setter*'id.

Andmeedastusobjekte kasutamise eelised:

- On võimalik tagada tõhus andmeedastus erinevate rakenduse kihtide vahel. Näiteks kui andmemudelis on rohkem välju kui konkreetse päringu jaoks vaja, siis võib andmeedastusobjekti kasutamine aidata vähendada edastatavate andmete hulka.
- Turvalisuse tagamine. Andmeedastusobjektide kasutamine võimaldab tagarakendusel kontrollida andmete edastamist. Näiteks saab tagarakendus andmeedastusobjekte kasutades tagada, et tundlikud andmed ei satuks pahatahtliku kasutajate kätte.
- Andmeedastusprotsessi paindlikkus. Tagarakenduses kasutatakse andmeedastusobjekte, et teisendada andmeid ühest vormingust teise või koguda andmeid mitmetest allikatest.

Kokkuvõtes on andmeedastusobjekte kasutamine tagarakenduses oluline puhta ja hooldatava koodi, turvalisuse ja andmeedastusprotsessi paindlikkuse tagamiseks.

2.4.1 Andmeedastusobjektide kasutamine

Arenduse käigus kasutasime andmeedastusobjekte andmete ülekandmiseks ühest rakenduse kihist teise, sageli andmebaasi ja esitluskihi vahele läbi domeenikihi. See võimaldab andmete ühtlustatud esitamist ning annab võimaluse erineva struktuuriga andmeid vastavalt vajadusele ümber kujundada.

Näiteks viimases rakenduse versioonis lisasime mitmeid statistikat kujutavaid graafikuid. Selleks pidime andmebaasis olevad andmed ümber kujundama vastavalt eesrakenduse nõuetele, et neid saaks korrektselt graafikutena kuvada. Kasutasime andmeedastusobjekte, mis sisaldasid ainult vajalikke välju ning esitasid andmed sobivas vormingus, mis vastasid graafiku kuvamiseks vajalikule struktuurile. See võimaldas meil edukalt edastada ja esitada graafiku andmeid eesrakenduses.

2.5 Töökäik

Lõputöö projekti alguses toimunud koosolekutel arutasime kliendiga võimalikke kasutuslugusid, et aru saada, millist lõpptulemust klient soovib. Selle põhjal sai läbi viidud uue funktsionaalsuse analüüs, hinnatud keerukus, kriitilisus ning umbkaudne ajakulu.

Projekti raames kasutasime rakenduse tööjärge (*product backlog*). Rakenduse tööjärg sisaldas töö pileteid (*task*), mis hakkasid ilmuma meie rakenduse analüüsiprotsessis.

Scrumi aluseks on tähtpunkt (*milestone*), mille käigus tehakse tööd toote kallal. Projekti arenduse käigus läbiti kolm tähtpunkti. Esimene tähtpunkt kestis jaanuarist esimese demopäevani. Teine tähtpunkt kestis esimesest demopäevast teise demopäevani. Kolmas tähtpunkt kestis teisest demopäevast viimase bakalaureusetöö elektroonilise versiooni esitamiseni. Enne iga tähtpunkti algust teostasime planeerimist, mis hindas rakenduse tööjärje sisu ja mille käigus tekkis tähtpunkti tööjärg (*milestone backlog*), mis sisaldas pileteid, mida oli vaja jooksval tähtpunktil täita.

Arenduse jooksul tegime iganädalaseid koosolekuid, mille eesmärk oli välja selgitada sprindi tööde seis ja edenemine, tekkinud takistuste varajane avastamine ning sprindi eesmärkide saavutamiseks vajalike strateegiate muutmise otsuste väljatöötamine. Iganädalased koosolekud toimusid kliendiga ning peale igat kohtumist oli eraldi wiki lehele kirja pandud kokkuvõte, mis on leitav Thesis_Julia_Ellina GitLabi hoidlas [8]. Samuti tegime igapäevaseid koosolekuid meeskonnaliikmete vahel. Suhtlemine tellijaga ja meeskonnaga toimus Discordis. Discord oli valitud seepärast, et seal saab teha kõnesid, koosolekuid ja

otseülekandeid (*stream*) ning kõikidel liikmetel oli kogemus Discordi kasutamisel.

Arenduse jooksul küsisime kliendi ja abiõppejõudude käest tagasisidet.

2.6 Tööjaotus

Iga meeskonnaliige tegi kompleksarendust (*full-stack*) ja rollid ei olnud meil rangelt jagatud. Iga tähtpunkti kohta oli olemas tähtpunkti tööjärg, kust vajadusel igal arendajal oli võimalus ise valida, mille kallal ta tööd jätkab. Alguses lepiti kokku, et igaüks vastutab oma poolt tehtud töö eest ja kui peale arendust leitakse vigu, siis nende parandamisega tegeleb arendaja, kes teostas selle *feature*'i.

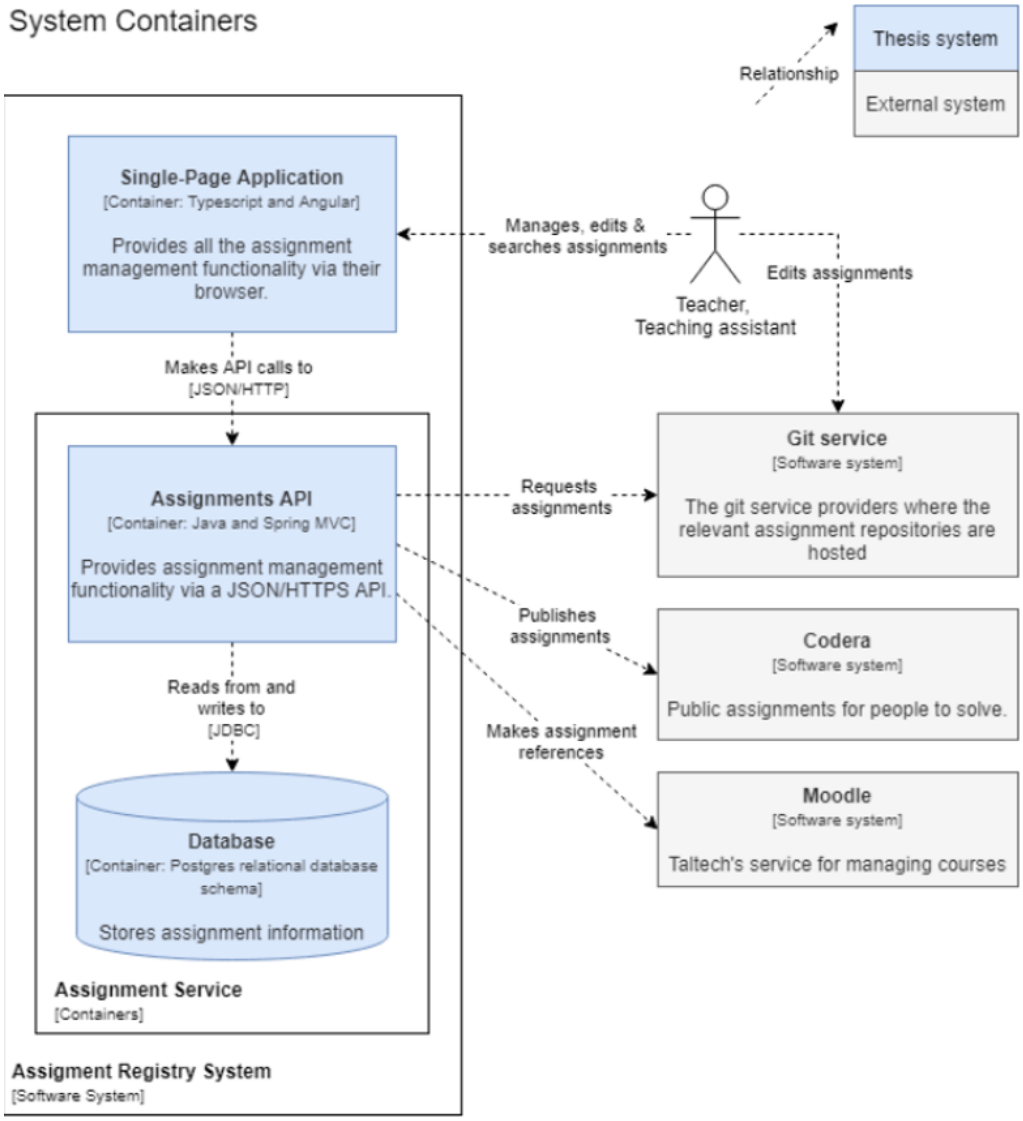
3. Projekti ülesehitus

Aurora koosneb kolmest peamisest osast: tagarakendusest, eesrakendusest ja operatsioonidest. Tehnoloogilised ja arhitektuurilised valikud olid tehtud eelnevate arendajate poolt.

3.1 Projekti arhitektuur

Projekti arhitektuur kirjeldab, kuidas süsteemi erinevad osad teevad koostööd saavutamaks projekti eesmärke. On olemas erinevad arhitektuurilised mustrid, mida arendajad saavad oma projektide struktureerimiseks kasutada, nagu *monolithic*, *service-oriented* ja *event-driven* [9]. Need mustrid aitavad arendajatel oma koodi organiseerida ja parandada projekti hooldatavust.

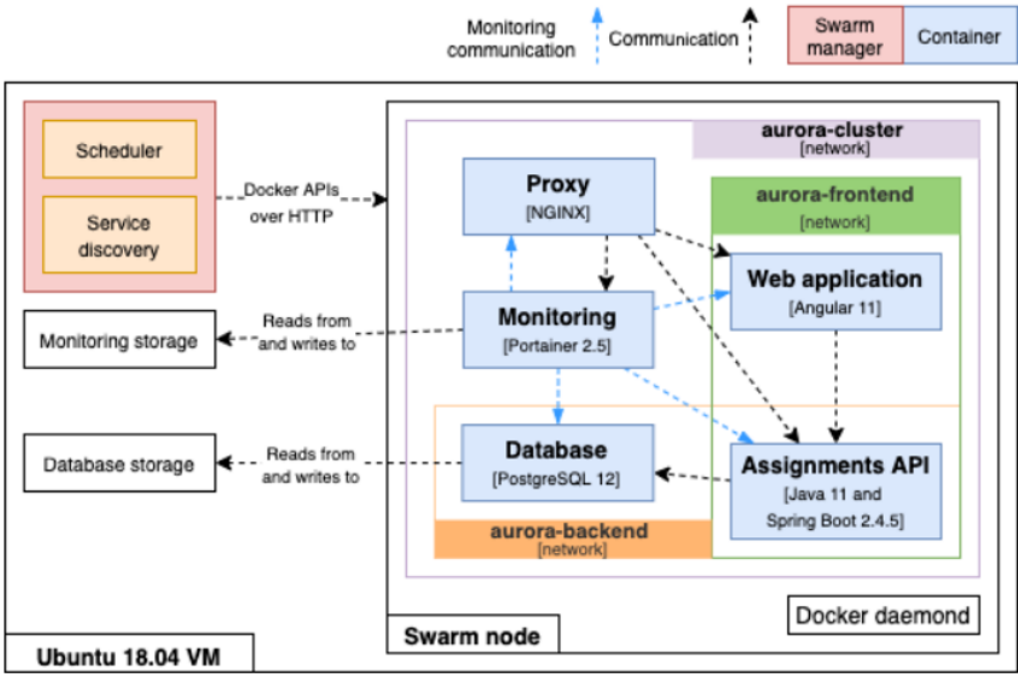
Aurora rakenduse aluseks võeti monoliitne arhitektuur. Süsteemi konteinerid on näidatud joonisel 3. Kui selged domeenid on paigas, on vajadusel modelleerimisotsuseid ja kihtide piire lihtsam parandada.



Joonis 3. *Süsteemi konteinerid.*

Ees- ja tagarakendus suhtlevad omavahel läbi REST API. REST API on veebiteenus ja ressursside kogum, mis vastab REST-i põhimõtetele ning mida saab nende ressurssidega läbi viia HTTP-meetodite (nt GET, POST, PUT ja DELETE) abil. Iga ressurssi tähistab kordumatu URL ja selle ressursiga suhtlemiseks kasutatav HTTP-meetod määrab selle ressursiga tehtava toimingu. Näiteks päring GET hangib ressursi, POST-päring loob uue ressursi, PUT-päring värskendab olemasolevat ressurssi ja päring DELETE kustutab ressursi.

Aurora rakendus jookseb ühe sõlme klastri kujul (*single node cluster*) Docker Swarmis TalTechi serveri virtuaalmasinas Ubuntu. Docker Swarm'i loogikaga saab tutvuda joonisel 4.



Joonis 4. *Süsteemi konteinerid.*

3.2 Tagarakendus

Aurora tagarakendus on loodud kasutades Spring Boot [10] raamistikku ja programmeerimiskeele Java 14. versiooni. Spring Boot võimaldab arendajatel luua kiiresti ja lihtsasti erinevaid rakendusi. See pakub palju valmislahendusi ja vähendab seeläbi arendamiseks vajaliku koodi hulka, mis omakorda säästab aega ning suurendab arendaja produktiivsust.

Spring Boot on hea valik ka seetõttu, et see pakub palju sisseehitatud turvalisuse funktsioone, sealhulgas Spring Security lahendust, mis tagab rakenduse turvalisuse. Spring Security [11] on kasutusel ka Aurora rakenduses.

3.2.1 Gradle

Gradle on avatud lähtekoodiga automatiseerimise tööriist, mis on mõeldud eelkõige Java ja Groovy tarkvaraprojektide ehitamiseks ja juhtimiseks. Gradle võimaldab automatiseerida korduvaid ehitusprotsesse ning pakub palju erinevaid funktsioone, sealhulgas testide käivitamist, tarkvara pakendamist ja levitamist [12], [13].

3.2.2 Mockito

Mockito on raamistik üksiktestide kirjutamiseks. Mockito eelis on see, et ta võimaldab arendajatel kiiresti ja lihtsasti testida keerulisi süsteeme, vähendades sõltuvusi teistest

klassidest või andmebaasiühendustest. Selle tulemusena on võimalik paremini isoleerida testitavat klassi, tõstes testide usaldusväärsust ja tagades, et testid annavad oodatud tulemusi [14].

3.2.3 Swagger

Swagger on vahend, mis aitab luua, dokumenteerida ja testida otspunkte. Arendajad saavad kiiremini mõista API funktsionaalsust ning klientidel on lihtsam aru saada, kuidas API-d kasutada ja milliseid vastuseid oodata [15].

3.2.4 Lombok

Lombok on Java teek, mis pakub annotatsioone Java klasside tüüpsisuga (*boilerplate*) koodi vähendamiseks, nagu omaduste väärtuste lugemised (*getter*), omaduste väärtuste muutmised (*setter*), konstruktorid ja palju muud. Lomboki kasutamine võib oluliselt vähendada kirjutatava koodi hulka, muutes koodi lihtsamaks [16].

3.3 Simple Logging Facade for Java

SLF4J on Java teek, mis pakub lihtsat viisi logimiseks, võimaldades rakenduse arendajatel logida sündmusi, mis lihtsustab tõrkeotsingut ja tarkvara hooldamist [17].

3.4 Esirakendus

Eesrakendus on üles ehitatud Angular 13 raamistikuga SPA stiilis. Stiilimiseks on rakenduses kasutusel Bootstrap, Angular Material ning SASS stiililehe keel.

3.4.1 Angular

Angular on raamistik, mis võimaldab arendajatel luua dünaamilisi ja interaktiivseid veebirakendusi. Tänu oma modulaarsele struktuurile võimaldab Angular suuremat tootlikkust ja paremat koodi taaskasutatavust. Lisaks on Angularil aktiivne kogukond ja lai valik ressursse, mis hõlbustavad õppimist ja probleemide lahendamist [18].

3.4.2 Bootstrap ja Angular material

Bootstrap on populaarne raamistik, mida kasutatakse veebirakenduste kasutajaliideste loomise jaoks. See sisaldab laia valikut HTML-, CSS- ja JavaScript-komponente, mis muudavad veebirakenduste loomise kiireks ja lihtsaks [19].

Angular Material raamistik sisaldab eelnevalt loodud kasutajaliidese komponente, mis on spetsiaalselt loodud Angulari raamistiku jaoks. Angular Materiali kasutamine koos Angulari raamistikuga aitab luua kvaliteetseid ja professionaalseid veebirakendusi [20].

Arenduse alguses langetati otsus, et võtame kasutusele Angular Materiali, sest võrreldes Bootstrapiga on seda koos Angulariga parem kasutada, sest see pakub spetsiaalselt Angulari rakenduste jaoks kavandatud kasutajaliidese komponente. Neid komponente saab sujuvalt integreerida Angulari rakendusele. See tähendab, et Angular Materiali komponente on Angulariga töötades lihtsam ja tõhusam kasutada kui Bootstrapi komponente.

Angular Material järgib Google Material Designi juhiseid, mis on laialdaselt tunnustatud kaasaegsete ja kasutajasõbralike liideste kujundamise parimaks tavaks. See tagab järjepideva väljanägemise ja kasutuskogemuse erinevates rakendustes ning platvormides.

Bootstrap on suurepärane raamistik, mis pakub laia valikut kasutajaliidese komponente ja paigutustööriistu, kuigi ei ole spetsiaalselt loodud Angulari jaoks. See tähendab, et Bootstrapi integreerimine Angular projekti võib nõuda lisapingutusi ja see ei pruugi pakkuda sama tõhusust kui Angular Materiali komponendid.

3.4.3 SASS

SASS on skriptikeel, mida kasutatakse CSS-koodi genereerimiseks. See lisab traditsioonilisele CSS-ile funktsionaalsust ja paindlikkust, muutes kirjutamise ja haldamise lihtsamaks. SASS võimaldab arendajatel kasutada ka muutujaid ja funktsioone, mida saab kogu koodibaasis uuesti kasutada [21].

3.4.4 Chart.js

Chart.js on võimas ja populaarne JavaScripti teek, mida kasutatakse veebilehtedel andmete visualiseerimiseks. See võimaldab luua mitmeid erinevaid graafikuid, näiteks joon-, tulp- või rõngasgraafikuid.

Chart.js'i peamine eelis seisneb selles, et see pakub lihtsat ja intuitiivset kasutajaliidest ning sellel on lai valik veebigraafika tüüpe ja kohandamise võimalusi. Chart.js on ka väga hästi dokumenteeritud ning sellel on suur ja aktiivne kogukond, mis tähendab, et kasutajad saavad kiiret ja lihtsat tuge, kui neil tekib küsimusi või probleeme [22].

3.5 Andmebaas

Aurora rakendus kasutab objekt-relatsioonilist andmebaasisüsteemi PostgreSQL, mis põhineb SQL andmebaasi päringukeelel.

3.5.1 Hibernate

Hibernate on ORM raamistik, mis võimaldab arendajatel lihtsalt andmebaasiobjekte kasutada Java objektidena ning hõlbustab andmebaasiga suhtlemist. Hibernate'i abil on võimalik luua efektiivseid ja loogilisi andmebaasiühendusi ning lihtsustada andmebaasi koodi hooldust.

Hibernate'i kasutamisel tuleb luua olemid, mis määratlevad andmebaasi tabelid ning nende omadused. Entiteet on Hibernate mõistes klass, mis vastab ühele andmebaasi tabeli reale ning mille objekte saab kasutada andmete lugemiseks, kirjutamiseks ja kustutamiseks. Entiteetide kasutamine võimaldab arendajatel lihtsustada koodi ja muuta seda loetavamaks [23].

3.5.2 Liquibase

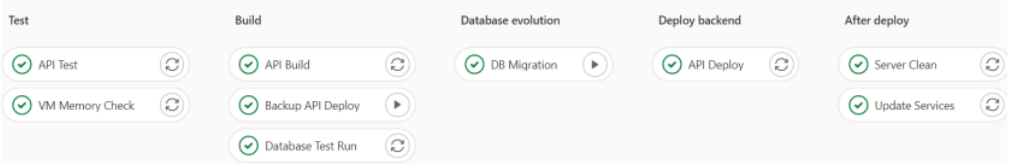
Liquibase on avatud lähtekoodiga raamistik, mis pakub mugavat võimalust andmebaasi versioonide halduseks ning lihtsustab tabelite loomist, muutmist ja andmete lisamist, kasutades SQL faile. Liquibase auditeerib kõik muudatused eraldi andmebaasi tabelisse, kus hoitakse infot muudatuste kohta nagu autor, kuupäev ja tabel [24].

3.6 CI/CD

Aurora rakendus kasutab pideva integreerimise ja pideva tarnimise praktikat. Konveieri (*pipeline*) olemasolevate testide jooksutamine pärast igat koodi muudatust tagab süsteemi järjepidevuse ja kontrollib, et uus koodilõik või parandus ei teinud rakendust katki.

Konveieri tööotsasid (*jobs*) jooksutab GitLab Runner. Kõikidel rakenduse komponentidel (eesrakendus, tagarakendus, operatsioonid) on olemas konveierid. Tagarakenduse

konveierit saab näha joonisel 5.



Joonis 5. *Tagarakenduse konveier.*

3.6.1 Docker

Docker on platvorm, mis võimaldab arendajatel rakendusi pakendada ja paigaldada konteineritesse. Konteiner on iseseisev keskkond, mis sisaldab rakenduse koodi, selle sõltuvusi ja muud vajalikku tarkvara, et rakendus saaks sujuvalt töötada erinevates keskkondades, olenemata nende konfiguratsioonist ja seadistustest [25].

4. Olemasoleva rakenduse testimine ja analüüs

Rakendusel puudus dokumentatsioon, mis kirjeldaks olemasolevat funktsionaalsust ning sisaldaks vanade rakenduse versioonide analüüsi. Sellel põhjusel algas projekti kallal töö testimisest.

Rakenduse funktsionaalsuse testimiseks otsustasime kasutada uurimuslikku testimist (*Exploratory Testing*) sest see on efektiivne viis, mille abil saab avastada vigu väga kiiresti. Lisaks on uurimuslik testimine põhjalik, kuna see võimaldab testijatel valideerida erinevaid stsenaariume ja kasutajate teekondi, mida ei pruugitud *unit test*-idega katta[26].

Me testisime läbi kõikvõimalikud kasutusjuhud ning terve rakenduse funktsionaalsuse. Testimise käigus tuvastasime funktsionaalsusi, mis ei tööta või töötavad vigadega ning mis vajavad edasiarendust või parandamist. Sellise funktsionaalsuse kohta teostasime analüüsi selleks, et aru saada probleemi suurusest, kui kriitiline see on, ja et otsustada, mida selles osas ette võtta.

Rakenduse funktsionaalsuse ja probleemide analüüsi käigus saadud tulemusi jagasime eraldiseisvate tööülesanneteks (*issue*) ning dokumenteerisime vastava ülesande kommentaarides. Kõik analüüsiga seotud ülesanded olid märgitud *Analysis* sildiga.

4.1 Analüüsi tulemused ja probleemide lahendused

Selles peatükis toome välja funktsionaalsuses leitud probleemid ja vead, mis olid analüüsitud. Kirjeldame probleemi ennast, probleemi analüüsi käigus saadud tulemusi ning kui probleem oli meie poolt lahendatud või funktsionaalsus parandatud, siis kirjeldame ka teostatud lahendust.

Lõime kõikide leitud vigade ja puuduste jaoks eraldi dokumendi, mis asub *Aurora Assignments API* projekti *wiki*'s [26]. Dokumendis on kirjeldatud leitud vead ja puudused ning pildid on manustatud selleks, et tulevastel arendajatel poleks vaja kulutada aega analüüsi peale. Meie poolt parandatud vead on märgistatud sõnaga FIXED.

4.1.1 Serveri katkev töö

Probleemi kirjeldus:

Enne arenduse alustamist saime kliendi käest teada, et rakendus ei töötanud serveril pikaajaliselt. Probleemi põhjus ei olnud tol hetkel teada. Serveril tekkis viga `java.lang.OutOfMemoryError: Java heap space`. Rakenduses puudus monitoorimine ning ei olnud võimalik jälgida, mis põhjustas selle vea.

Analüüsi tulemus:

Analüüsi alustati sellest, et koodi iga meetodi sisse lisati logimist kasutades SLF4J teeki. Rakenduses on kasutusel kaks perioodilist tegumit (*cron job*). Üks nendest skaneerib hoidlaid failide sarnasuste leidmiseks. Teise tegumi abil toimub esituste sidumine ülesannetega juhul, kui esitus ilmub rakenduses varem kui ülesanne ise. Logimise abil saime teada, et `java.lang.OutOfMemoryError: Java heap space` viga tekib teise perioodilise tegumi tõttu, kuigi selle vea põhjuse leidmine ei olnud kiire protsess. Arenduse alguses ei olnud Aurora rakendus ühendatud automaattestriga, mis tähendas, et ülesannete esitused ei jõudnud rakendusse ja viga ei olnud võimalik kinni püüda. Ka peale automaattestriga ühendamist ei olnud võimalik viga tuvastada. Vea tekkimisel hakkasime probleemi uurima lisalogimise abil. Probleem seisnes selles, et perioodilise tegumi loogika ei olnud valmis esituste suureks arvuks.

Aurora rakendusse jõuavad ning andmebaasi salvestatakse kõik ülesannete esitused, mis tulevad üldiselt automaattestrist. Selle funktsionaalsuse põhimõte on, et juhul kui tulevikus otsustab keegi lisada Aurorasse hoidla, siis varasemalt tehtud esitused ilmuksid selle hoidla ülesannete juures. Vahel võib juhtuda, et rakendusse jõuab suur arv esitusi ja perioodilise tegumi mõte on proovida neid siduda ülesannetega juhul, kui ülesanded on juba Aurora rakenduses olemas.

Tegumi uurimisel saime teada, et kirjutatud loogika (vt Lisa 2 – Perioodilise tegumi vana loogika) ei sobi, sest suurte andmemahtudega töötamisel on vaja jagada andmed väiksemateks osadeks ja ainult siis neid töödeldada, kuigi meie rakenduses töötas enne kirjutatud meetod nii, et alguses pärib see kõik esituste kirjed andmebaasist, mis ei ole seotud ühegi ülesandega. Probleem oligi selles osas, sest kõik kirjed päriti ja uuendati andmebaasist, mis põhjustas suure kirjete arvu puhul jõudluse probleeme. Lisaks põhjustas see lähenemisviis rakenduses suure mälukasutuse, mis omakorda tekitas antud vea. `java.lang.OutOfMemoryError: Java heap space` viga juhtub siis, kui rakendus püüab kanda mällu liiga palju objekte või töötleb liiga suuri andmemahte. Selle tulemusena võib rakendus hakata jooksma väga aeglaselt või lõpetada töö ootamatult, mis siinkohal juhtuski.

Lahendus:

Probleemi parandamiseks oli mitu lahendusviisi:

- *Heap space*'i suurendamine. *Heap space*'i suurendamisel tuleb aga arvestada, et see võib mõjutada rakenduse jõudlust ja vajada piisavalt süsteemiressursse.
- Andmete haldamine. Kui rakendus töötleb suurt hulka andmeid, võib probleemi lahendada andmete lehekülgedeks jagamise tehnika kasutamine või andmete järjestikune töötlemine.
- Koodi optimeerimine. Kui rakenduse kood ei ole optimeeritud, võib see põhjustada mäluprobleeme. Seetõttu on oluline kontrollida koodi ja vältida tarbetuid objektide loomisi.

Aurora rakendusele esimene lahendusviis ei sobi, kuna ei saa kunagi ette teada, kui palju esitusi rakendusse tuleb. Parandasime andmete haldamist ning optimeerisime koodi.

Uus lahendus otsib esitusi, mis ei ole määratud ühelegi ülesandele, kontrollides esituse olemis (*entity*) ülesande välja. Neid, millel antud väli on *null*, võetakse andmebaasist 100 kirje kaupa. Edasi kirjed grupeeritakse, võetakse andmebaasist seotud ülesanded ja määratakse ülesandele esitused. Veendumaks, et andmebaasist võetud kirjed on unikaalsed ja dubleerimata, järjestatakse need alati ID järgi kasvavas järjekorras ning meetodis salvestatakse kõige suurem ID, mis esineb andmebaasist võetud leheküljes.

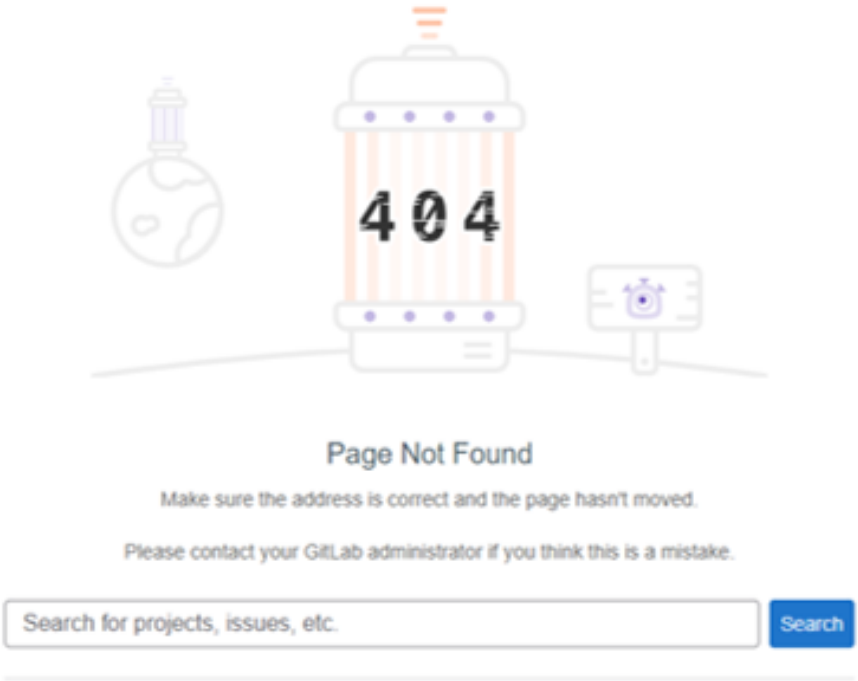
4.1.2 Valed ülesannete lingid

Probleemi kirjeldus:

Vajutades kas sinise klambri nuppu (vt Lisa 3 – Ülesande vaade koos nupuga, mis viib kasutaja ülesande lähtekoodi) ülesande vaates või klikkides ülesande nimele kõikide ülesannete vaates (vt Lisa 4 – Kõikide ülesannete vaade), suunatakse kasutaja vale lingiga lehele.

Link, kuhu kasutaja ümber suunatakse lisas 3 näidatud ülesande puhul:
https://gitlab.cs.ttu.ee/iti0102-2022/ex/-/tree/master/T/list_basic

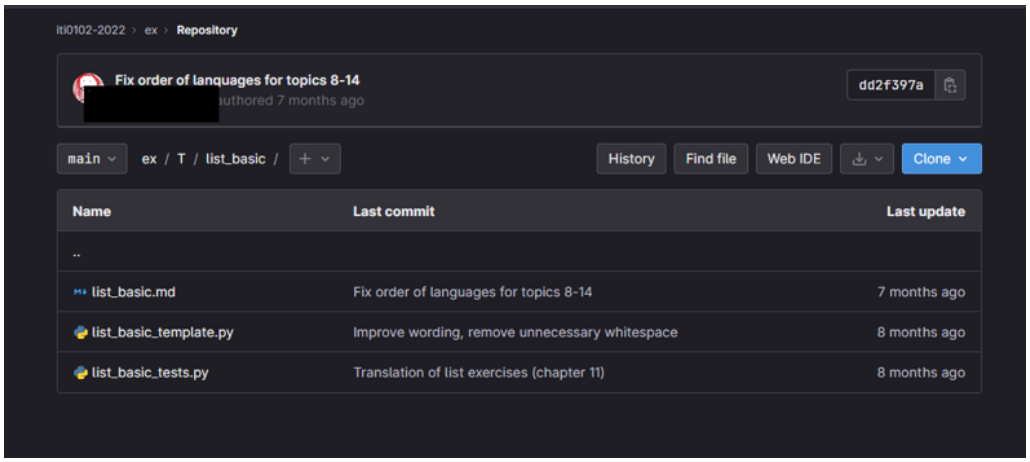
Peale lingile suunamist näeb kasutaja vaadet, nagu on näidatud joonisel 6:



Joonis 6. *Vaade, mida näeb kasutaja peale ümbersuunamist.*

Õige link, kuhu kasutaja peab olema ümber suunatud: https://gitlab.cs.ttu.ee/iti0102-2022/ex/-/tree/main/T/list_basic

Kui link oleks õige, siis kasutaja näeks vaadet, mis on näidatud joonisel 7.



Joonis 7. *Vaade, mida kasutaja näeks, kui link oleks õige.*

Analüüsi tulemus:

Lingi kokkupanemise loogika, kuhu kasutaja ümber suunatakse, oli püsikodeeritud sõnaga master nagu on näha joonisel 8.

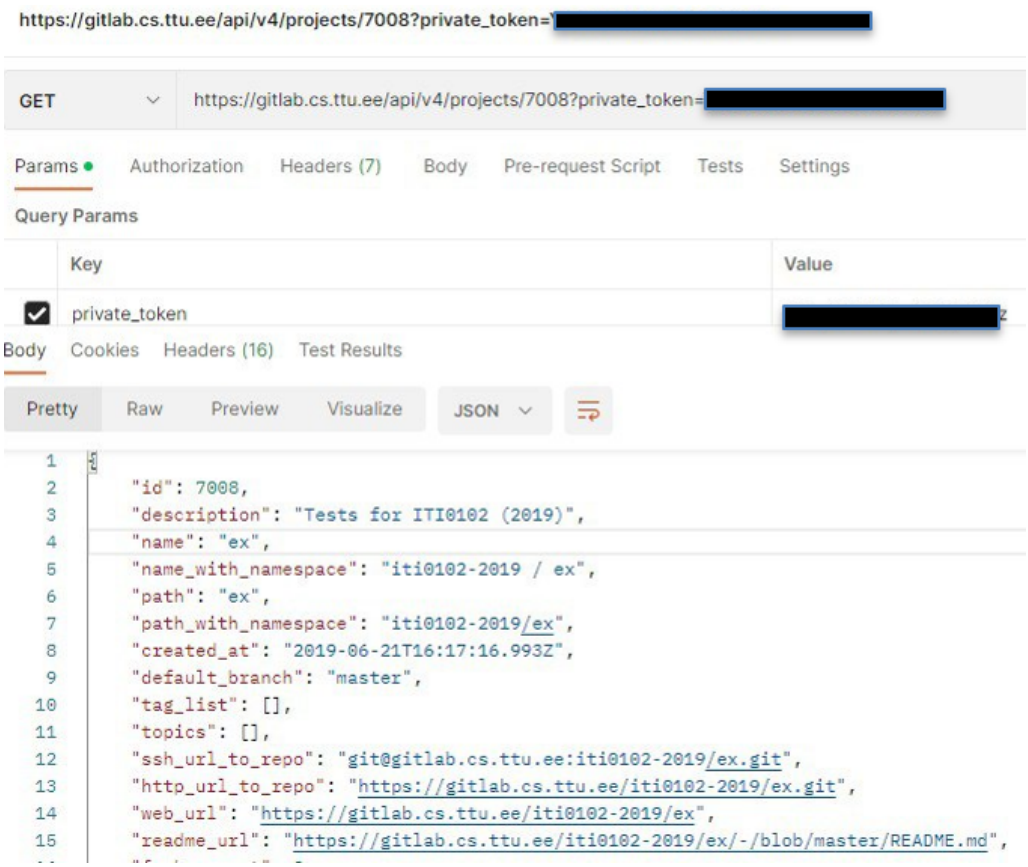
```
    REPLACE(rl.http_url_to_repo, '.git', '/')
        || '-/tree/master'
        || REPLACE(REPLACE(a.file_path, './repositories/', ''), rl.file_path,
                   '')                                          AS repository_https_reference,
```

Joonis 8. *Vana püsikodeeritud loogika lingi kokkupanemiseks.*

Lahendus:

Probleem sai lahendatud kasutades GitLab API-t, sest muud varianti õige hoidla vaikeharu (*default branch*) saamiseks ei olnud. Selleks, et teada saada, milline vaikeharu peab lingi sees olema (kas *main* või *master*), tehakse päring lehele https://gitlab.cs.ttu.ee/api/v4/projects/project_id?private_token=private_token kus *project_id* on hoidla ID, milles ülesanne asub, ja *private_token* on Aurora kasutaja privaatne võti, mis genereeritakse GitLabis.

Päring saadab vastuse JSON kujul, nagu on näidatud joonisel 9. Saadud vastusest saame teada, mis on hoidla vaikeharu ning vastavalt sellele saame õige lingi koostada.



Joonis 9. *Vastus, mida saadab GitLab API.*

4.1.3 Ülesannete otsing

Probleemi kirjeldus:

Ülesannete otsing annab mõnel korral juhuslikke või valesid tulemusi ning vahel ei anna neid üldse. Tulemuste laadimine võtab palju aega. Joonisel 10 on näha, et otsides märksõna "2020" ei anna otsing ühtegi ülesannet, mille nimes oleks see arv esindatud.

Joonis 10. *Tulemus, mida kasutaja näeb, otsides sõna "2020".*

Analüüsi tulemus:

Esimese arendaja poolt oli otsingu loogika teostatud väga suure päringuga, mis oli raskendatud üleliigse loogikaga. Raskendamise põhjustas see, et päring küsis andmebaasist mitte ainult ülesandeid, mille nimes oli midagi sarnast kasutaja poolt sisestatud sõnaga, vaid vaatas ülesandeid, mille lähtekoodis otsitav sõna samuti esines. Lisaks sellele arvutas loogika, kui palju selliseid sõnu koodis esines.

Tavalised otsingud ei ole implementeeritud nii, et saaks otsida ka failide sisust. Selleks on tavaliselt tehtud eraldi otsing ja lisatud paralleelsus, et töötlemine ei võtaks palju aega. Vanas otsingu loogikas paralleelsust ega asünkroonsust ei olnud, see oli tõstutundlik ning päringus endas olid vead.

Lahendus:

Päringu loogikat lihtsustati ja parandati. Lihtsustamine aitas parandada ka otsingu kiirust.

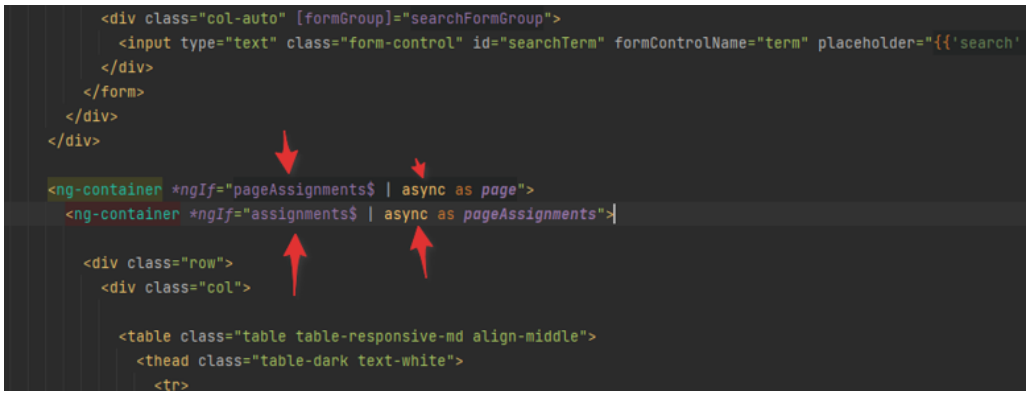
4.1.4 Topelt päring otspunkti (*endpoint*)

Probleemi kirjeldus:

Ülesannete lehe laadimine võtab palju aega ja kasutaja näeb, et selle lehe avamisel tehakse 2 järjestikust päringut otspunktile (vt Lisa 5 – Brauseri konsool, kus on näha, et tehakse 2 päringut järjest).

Analüüsi tulemus:

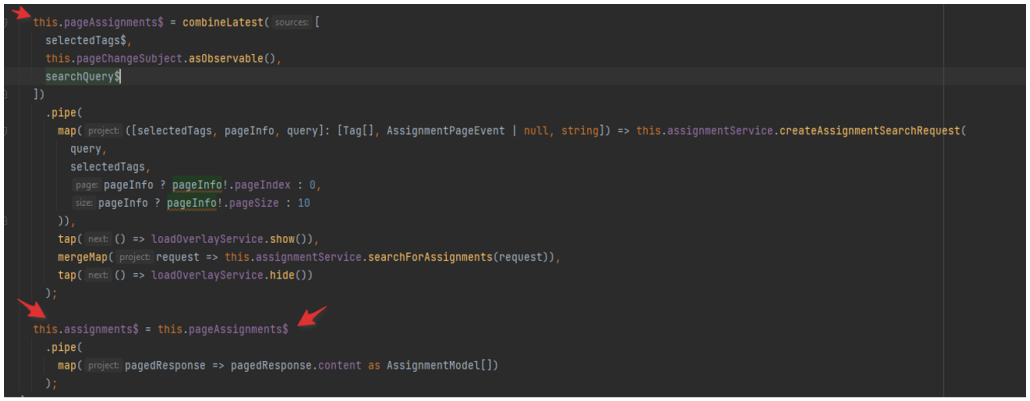
Analüüsi tulemusel tuvastati, et kahekordse päringu põhjuseks on HTML failis olevad kaks eraldi asünkroonset jälgitavat (*observable*), mida saab näha joonisel 11.



Joonis 11. *2 asünkroonset jälgitavat, mis põhjustavad kahekordse päringu.*

Joonisel 12 näidatud koodis iga asünkroonne toru (*pipe*) loob sellega seotud jälgitavale uue jälgimise (*subscription*), mille tulemuseks on mitu päringut. Joonisel 11 esimeses ng-konteineri plokis jälgib asünkroonne toru `pageAssignments$` muutujat, mis on combineLatest ja mergeMap operaatorite tulemus. See jälgitav väljastab otsingutulemused vastusena meetodite `createAssignmentSearchRequest` ja `searchForAssignments` loodud otsingupäringule, mida saab näha joonisel 12.

Teises ng-konteineri plokis jälgib pipe `assignments$` jälgitavat, mis luuakse `pageAssignments$` pipe'des map operaatori kaudu. See jälgitav väljastab samu otsingutulemusi nagu `pageAssignments$`, lihtsalt erineva nimega.



Joonis 12. *Väärtuste omistamine jälgitavatele, mis põhjustavad kahekordse päringu.*

Lahendus:

Kuna nende kahe muutuja mõte oli samasugune, otsustasime kahest jälgitavast vabaneda ning teha nendest ühe üldise jälgitava, mis sisaldaks endas vajalikke atribuute, mida on vaja kasutajale kuvada.

4.1.5 Ülesannete esituste mitteilmumine

Probleemi kirjeldus:

Ülesannete esitused tulevad ainult Pythoni kursuste hoidlatesse (vt Lisa 6 – Pythoni kursuse ülesanne, kuhu jõuavad tulemused ja Lisa 7 – Java kursuse ülesanne, kuhu tulemused ei jõua). Vaatamata sellele, et reaalajas saadavad paljud õpilased esitusi Java kursuse hoidlasse (vt Lisa 8 – Java kursuse tudengite tulemused Charon süsteemis), ei seostata esitusi Auroras vastavate ülesannetega.

Analüüsi tulemus:

Analüüsi tulemusel tuvastati, et esituste jõudmisel Aurorasse proovitakse siduda neid olemasolevate ülesannetega, kasutades esituse olemis olevaid välju nagu `assignment_dir` ja `repo_path`. Nende abil proovitakse leida andmebaasi tabelist nimega *assignment* sobivat ülesannet, mille `file_path` võrdub esituse kokkupandud `assignment_dir` ja `repo_path` väärtustega. Python ülesannete leidmiseks see lahendus sobis, kuna projekti struktuur ei sisalda pakke (*package*). Tabelis 1 on näha, et Java ülesannete leidmiseks selline lahendus ei sobi, sest nende `file_path` sisaldab pakke.

Table 1. *Failitee kokkupanek ülesande otsimiseks*

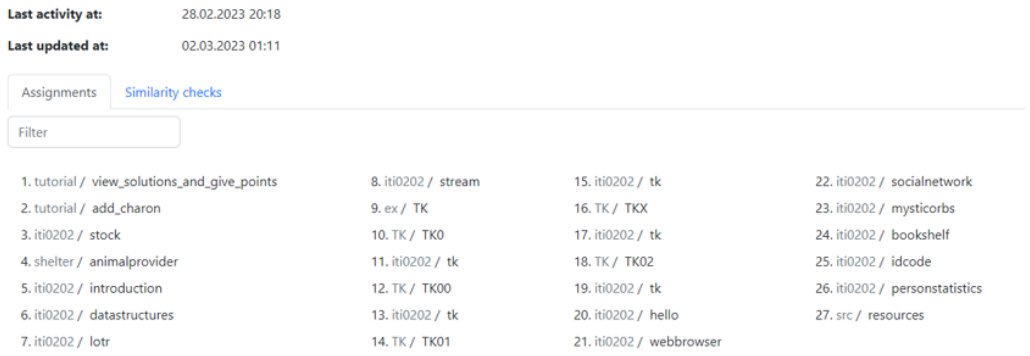
	***repo_path* esituse tabelist**	***assignment_dir* esituse tabelist**	***file_path* ülesande tabelist**
Python	iti0102-2020/ex	EX/ex08_formula_one	iti0102-2020/ex/EX/ex08_formula_one
Java	iti0202-2023/ex	PR/PR04Stock	iti0202-2023/ex/PR/PR04Stock/src/ee/taltech/iti0202/stock

Lahendus:

Lahenduses ei otsi me andmebaasist ülesannet, mille `file_path` ülesande tabelis võrduks esituse kokkupandud `repo_path` ja `assignment_dir` väärtustega vaid vaatame, et `file_path` algus sisaldaks kokkupandud `repo_path` ja `assignment_dir` väärtust. Selline lahendusviis garanteerib, et ülesanne leitakse ning see oleks võetud õigest hoidlast, kuna kasutame `repo_path` esituse tabelist.

4.1.6 Ülesannete teed (*path*) hoidla vaates

Probleemi kirjeldus: Kasutaja ei näe Java hoidlas olevate ülesannete teid (*path*) õigesti. Näiteks joonisel 13 peaks numbri 23 all olema *ex/mysticorbs*, mitte *iti0202/mysticorbs* ja numbri 4 all peaks olema *pr/shelter*, mitte *shelter/animalprovider*



Joonis 13. *Java hoidlas olevate ülesannete valed teed.*

Analüüsi tulemus:

Ülesannete teedega oli 2 probleemi: teede kokkupanemise loogika oli püsikodeeritud ning oli mõeldud ainult Pythoni hoidlate jaoks. Lisaks oli see loogika teostatud eesrakenduses, mis ei vasta The Three Principal Layers printsiibile.

Lahendus:

Kokkupanemise loogika sai optimeeritud ning nüüd töötab see õigesti erinevate hoidlatega. Lisaks sellele on loogika migreeritud tagarakendusse. Tagarakenduses on ülesande tee salvestatud uude andmeedastusobjekti (DTO), mida saadetakse eesrakendusse kuvamiseks.

4.1.7 Keele valik

Probleemi kirjeldus:

Rakenduses on kasutusel kaks keelt: eesti ja inglise. Ingliskeelne versioon ei olnud kohati eesti keelde tõlgitud.

Analüüsi tulemus:

Analüüsi käigus saime teada, et mõnes kohas oli püsikodeeritud lingid, mis viitavad alati ingliskeelsele versioonile ning HTML failidesse olid sisse kirjutatud ingliskeelsed

tähendused, mitte tõlgete võtmed.

Lahendus:

Lisasime eesti- ja ingliskeelse tõlgete failidesse tõlked koos võtmetega, mida kasutame HTML failides, et keele valimisel oleksid sõnad kuvatud õiges keeles ning linkide koostamisel võetaks arvesse kasutaja poolt valitud keelt.

4.1.8 Rakenduse mittetestimine

Probleemi kirjeldus:

Jooksutades rakendust serveril, oli Auroras väike hulk testandmeid, mida eelmised arendajad lisasid andmebaasi päringute (*script*) abil ning selliste andmetega töötas funktsionaalsus korrektselt. Kui aga rakendusse hakkasid tulema päris andmed, tuli välja, et rakendus ei ole võimeline töötama suurte andmemahtudega, kuna tagarakenduses kirjutatud loogika oli sobilik ainult väikeste andmehulkadega töötamiseks. Näiteks ei olnud läbi mõeldud loogika, kuidas näidata ülesannete esitusi, kui neid on palju; kuidas näidata hoidlaid hoidlate tabelis, kui neid on palju; kuidas näidata hoidlas olevaid ülesandeid, kui neid on palju.

Analüüsi tulemus:

Analüüsi käigus tuli välja, et rakendus ei olnudki eelnevalt serveri peal kasutaja poolt testitud terves mahus. Ainult eelmised arendajad testisid osa rakenduse funktsionaalsust eksami esitamistega. Lisaks sellele ei olnud testimise tulemused dokumenteeritud.

4.1.9 *Query results in file* funktsionaalsus

Probleemi kirjeldus:

Ei ole selge, mida varasemad arendajad teostada tahtsid ning mida tähendab "*Requires migration*" (vt Lisa 9 – Ülesande vaade, kui avada *query results in file*).

Analüüsi tulemus:

Eelmiste arendajate idee oli selline, et kui leitakse sarnaseid ülesandeid, kuvatakse *Query results in files* vaates sarnaseid koodijuppe süntaksi esiletõstmisega. Otspunkti tegi esimene

arendaja Pihlak, kuid kasutajaliidese ümberkujundamise käigus ei õnnestunud järgmisel arendajal Serkin asjakohast teeki leida. "*Requires migration*" on püsikodeeritud fraas, mis tähendab, et oleks vaja leida sobiv teek.

4.1.10 Ülesandelt siltide eemaldamine ilma kinnituseta

Probleemi kirjeldus:

Vajutades *Clear* nuppu ülesande vaates (vt Lisa 10 – Ülesande vaade koos siltidega ja *Save, Clear* nuppudega) kustutatakse korraga kõik olemasolevad sildid ilma kinnituseta. Samas kui olemasolevaid silte ükshaaval eemaldada, ei kao silt enne, kui vajutada nuppu *Save*.

Analüüsi tulemus:

Arendajate poolt ei olnud tagarakenduses teostatud loogika, mis saadaks kasutajale kinnituse enne ülesandelt kõikide siltide eemaldamist. Kustutamise, lisamise ja salvestamise funktsionaalsus töötab korrektselt, kuid puudub kasutaja tegevuste kinnitamine.

4.1.11 Automaatne sildistamine ja *Add to tagging queue*

Probleemi kirjeldus:

Ei ole selge, mis toimub, kui kasutaja vajutab nuppu *Add to tagging queue* (vt Lisa 11 – *Automatic tagging* vaade) ja kas automaatne sildistamine üldse töötab.

Analüüsi tulemus:

Add to tagging queue lisab hoidlas olevaid ülesandeid järjekorda, et ülesannetele automaatselt silte lisada. Tagarakenduses on olemas loogika selle operatsiooni jaoks, kuigi see ei ole lõpuni teostatud. Hetkel saab kasutaja lisada hoidlat järjekorda ning kui teist korda sama tegevust teha, näeb ta veateadet "*Error adding tag check job to queue*". Tagarakenduses on olemas meetod, mis viskab vea, kui hoidla on juba järjekorda lisatud, kuid pole veel töödeldud. See viga ilmneb seetõttu, et hoidla ülesanne läheb järjekorda ja jääb seal kinni, kuna loogika pole lõpuni arendatud.

4.1.12 Hoidlate kustutamine

Probleemi kirjeldus:

Aurora kasutaja eemaldamisel GitLabi hoidlast ei kao hoidla ja hoidlasse kuuluvad andmed Aurora rakendusest.

Analüüsi tulemus:

Tagarakenduses puudub loogika sellise juhtumi kohta, kui kasutaja soovib hoidla ja tema sees olevad ülesanded Aurorast kustutada. Hetkel saab Aurorasse tekitada lõpmatult palju hoidlaid, aga kustutada saab neid vaid terminalis, tehes päringu otse andmebaasi. Tuleb lisada funksionaalsus, mis võimaldaks kasutajal kustutada hoidlaid otse rakendusest ja teha nii, et GitLabis võetakse kustutatud hoidlast Aurora kasutajad ka ära, et järgmisel sünkroniseerimisel ei ilmu see hoidla tagasi Aurorasse. Kasutades privaatset tõendit (*private token*) ja hoidla ID-d saab Aurora kasutaja iseennast hoidlast kustutada. Selleks tuleb teha päring GitLab API kaudu [27].

4.1.13 Siltide grupi kustutamine

Probleemi kirjeldus:

Siltide gruppi ei saa kustutada, kui vähemalt ühel ülesandel on selle grupi silt, kuigi samal tingimusel saab silti grupist kustutada ning see kaob ülesannetelt automaatselt ära.

Analüüsi tulemus:

Tagarakenduses puudub piirang, mis ei laseks kasutajal kustutada silti grupist, kui silt on olemas vähemalt ühel ülesandel.

4.1.14 Hoidlate lisamine lõputulut kontrolli järjekorda

Probleemi kirjeldus:

Kui avada hoidla kõikide hoidlate vaatest, saab kasutaja lisada avatud hoidlat piiramatu arv kordi järjekorda, kus hakatakse kontrollima sarnasuste peale avatud hoidlat teiste hoidlatega (vt Lisa 12 – *Add to check queue* hoidla vaates). Näiteks, kui lisada sama hoidla järjekorda kolm korda, siis on vaja oodata, kuniks 3 sama protsessi on täidetud, et lisada järjekorda uut hoidlat.

Analüüsi tulemus:

Tagarakenduses puudub piirang, mis keelaks kasutajal mitu korda lühikese ajaperioodi jooksul sama hoidlat kontrolli järjekorda (*check queue*) lisada.

4.1.15 Hoidlate sünkroniseerimine GitLabiga

Probleemi kirjeldus:

Avades kindlat hoidlat ei saa kasutaja sünkroniseerida seda GitLabiga. Sünkroniseerimise nuppu saab vajutada ning saadakse vastus, et sünkroniseerimine õnnestus, kuid tegelikult ei toiminud midagi. Kasutaja saab hoidlaid sünkroniseerida ainult kõigi hoidlate lehelt ning see võtab palju aega, kuna üritatakse sünkrooniseerida kõiki Auroras olevaid hoidlaid.

Analüüsi tulemus:

Tagarakenduses on olemas meetod, mida kutsutakse kindla hoidla GitLab'iga sünkroniseerimiseks, kuigi loogikat selle meetodi sees ei ole ning meetod tagastab alati vastuse, et operatsioon toimus edukalt.

5. Edasiarendus

Selles peatükis kirjeldame uut funktsionaalsust, mis oli lisatud Aurora rakendusse lõputöö raames.

5.1 Ülesannetele kommentaaride lisamine

Üks kliendi soovidest oli võimalus lisada ülesannetele kommentaare. Kui mõne ülesande juures tekkis probleem, siis jäid need probleemid tihti lahendamata, kuna probleemi ei olnud kuhugi kirja panna. Ülesande kommenteerimine annaks võimaluse kõik ülesandega seonduva hoida ühes kohas. Vastavalt tegumikesksele kasutajaliidese disainile enne arenduse algust arutlesime kliendiga, mida ta soovib saavutada ülesannete kommentaaride lisamise funktsionaalsusega ehk koostasime kasutusloo (*use case*), peale mida analüüsisime kliendi soove ja pakkusime kasutajaliidese disaini (visuaalne prototüüp) selle probleemi lahendamiseks. Pärast disaini kinnitamist kliendi poolt alustasime arendust. Arenduse alguses lõime ülesannete kommentaaride andmemudeli (vt Lisa 13 – Ülesandele kommentaaride lisamine). Loodud funktsionaalsus võimaldab näha kommentaare, mis on järjestatud kuupäeva kahanevas järjekorras (vt Lisa 14 – Ülesandele lisatud kommentaarid). Kommentaare saab filtreerida (vt Lisa 15 – Ülesande kommentaaride filtreerimine). Kommentaari lisamiseks tuleb lisamise modaal, kuhu on vaja sisestada kommentaari sisu (vt Lisa 16 – Ülesandele kommentaari lisamine).

5.2 Hoidla ülesannetele GitLabi lingi lisamine

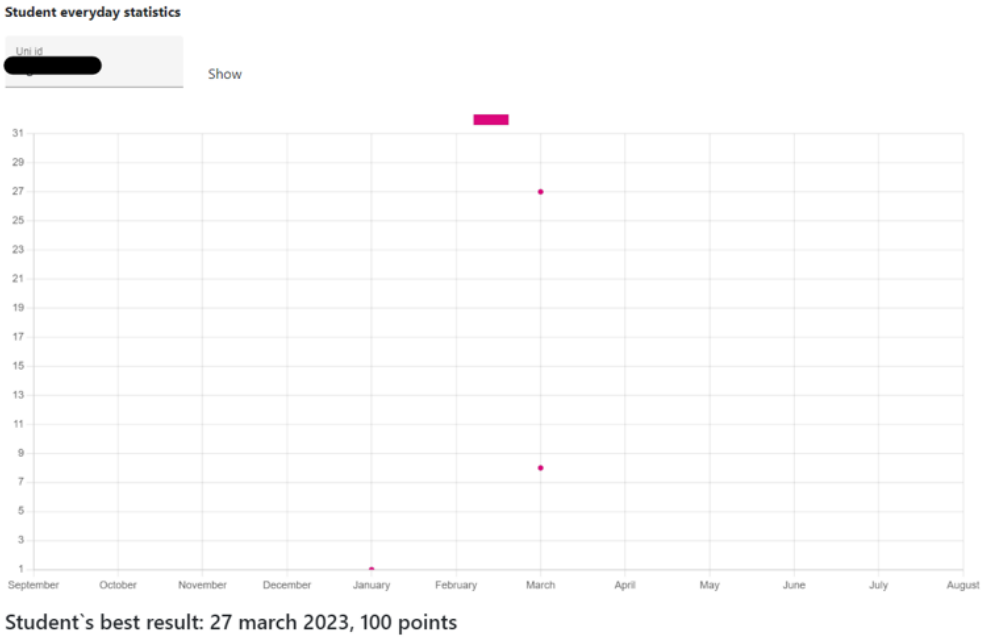
Klient soovis, et hoidla vaatest, kus saab näha kõikide hoidlas olevate ülesannete nimesid, saaks minna ülesannete lähtekoodile. Nüüd iga ülesande nimi sisaldub enda sees linki, millele vajutades saab minna GitLabis ülesande koodi juurde.

5.3 Statistika

Kliendi soovil lisasime Aurorasse mitu erinevat statistikat, mis sisaldasid endas infot hoidlas olevate ülesannete kohta. Statistika näitamine sai teostatud nii tabelite kui ka graafikute kujul.

5.3.1 Interaktiivne tudengite igapäevane statistika

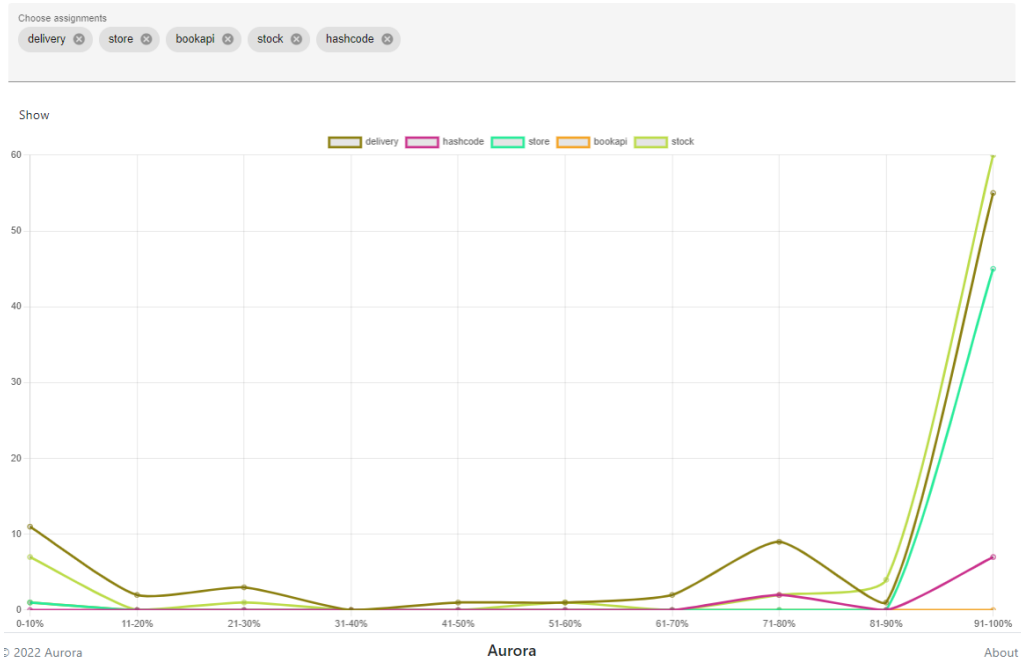
Kindla ülesande vaates saab kasutaja valida tudengi Uni-ID, kelle kohta ta soovib ülesande statistikat vaadata. Peale tudengi valimist näeb kasutaja graafikut, nagu on näidatud joonisel 14, kus x-teljel on kuu ja y-teljel on kuupäev. See graafik näitab, millal tudeng tegi esitusi ning kui palju neid kindlal kuupäeval oli. Selles graafikus näidatakse alati jooksvat õppeaastat. Graafikul näidatud täpi suurus sõltub sellest, kui palju kordi tudeng sellel päeval esitusi tegi. Mida suurem täpp, seda rohkem esitusi sellel päeval oli. Kursoriga täpile liikudes saab näha, kui palju ja mis kuupäeval olid esitused tehtud (vt Lisa 17 – Kursoriga täpile liikudes kuvatud informatsioon). Samuti saab graafiku all näha infot tudengi parima tulemuse ja kuupäeva kohta.



Joonis 14. *Valitud tudengi statistika kuude ja päevade kaupa.*

5.3.2 Interaktiivne statistika valitud ülesannete kohta kindlas hoidlas

Kindla hoidla vaates saab näha statistikat valitud ülesannete kohta, nagu on näidatud joonisel 15. Ülesannete valikutes esinevad ülesanded, millel oli tehtud vähemalt üks esitus. *Choose assignments* väljal saab valida maksimaalselt viis ülesannet ning sama ülesannet mitu korda valida ei ole võimalik. Kuvatud graafiku x-teljel on lõpptulemuse protsent vahemikena (0-10%, 11-20% jne), y-teljel selles vahemikus hinde saanud inimeste arv. y-telje suurim väärtus sõltub suurimast õpilaste arvust valitud ülesannete seast. Iga vahemiku kohta näidatakse vastava hinde saanud tudengite arvu ning arvesse võetakse iga esitaja parimat tulemust.



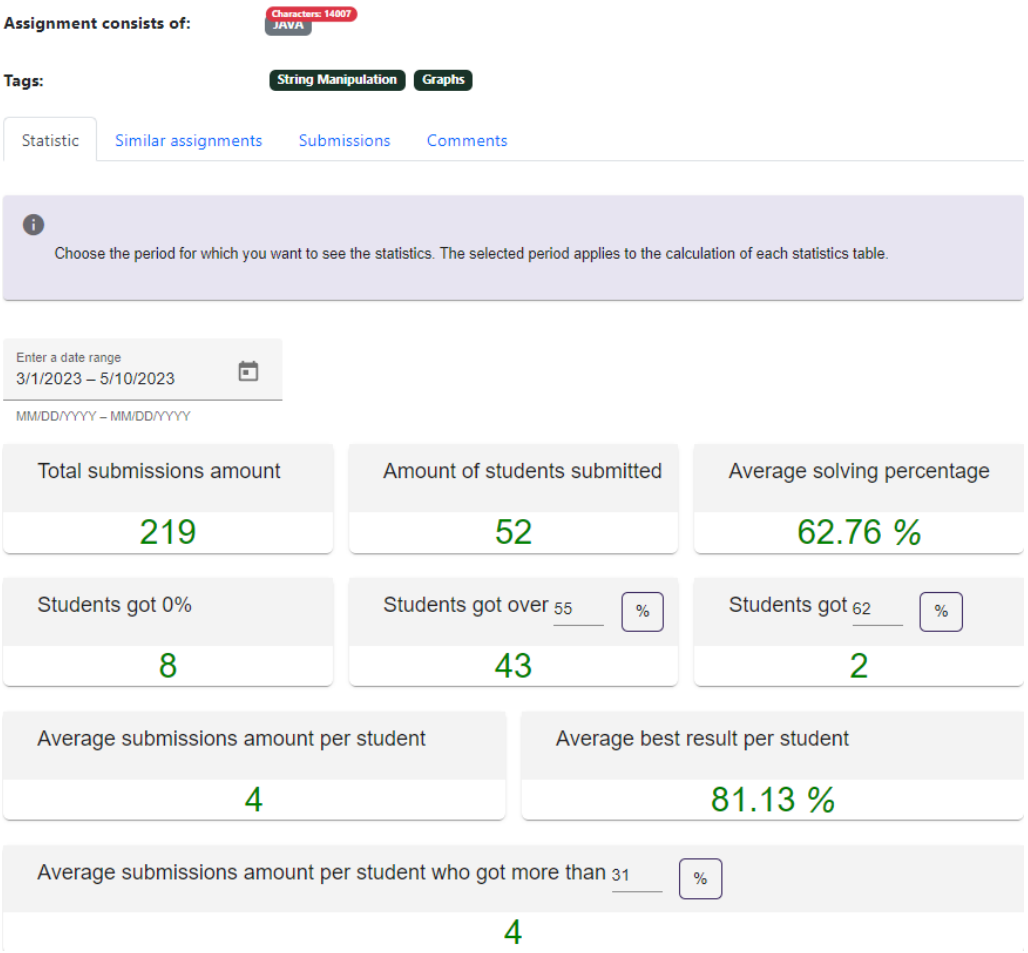
Joonis 15. *Statistika valitud ülesannete kohta.*

5.3.3 Statistika tabelite kujul

Kindlat ülesannet avades saab kasutaja valida perioodi, mille ulatuses ta tahab statistikat näha. Valitud periood rakendub iga statistika tabeli arvutamiseks, mida saab näha joonisel 16. Kasutaja saab valida algus- ja lõppkuupäeva (vt Lisa 18 – Statistika arvutamise algus- ja lõppkuupäeva valimine). Kui kasutaja ei vali perioodi, siis näidatakse statistikat kogu ülesande andmetega. Kolmes tabelis saab kasutaja ise määrata protsenti, millega arvutatakse statistikat.

Ülesandel on 9 tabelit statistikaga:

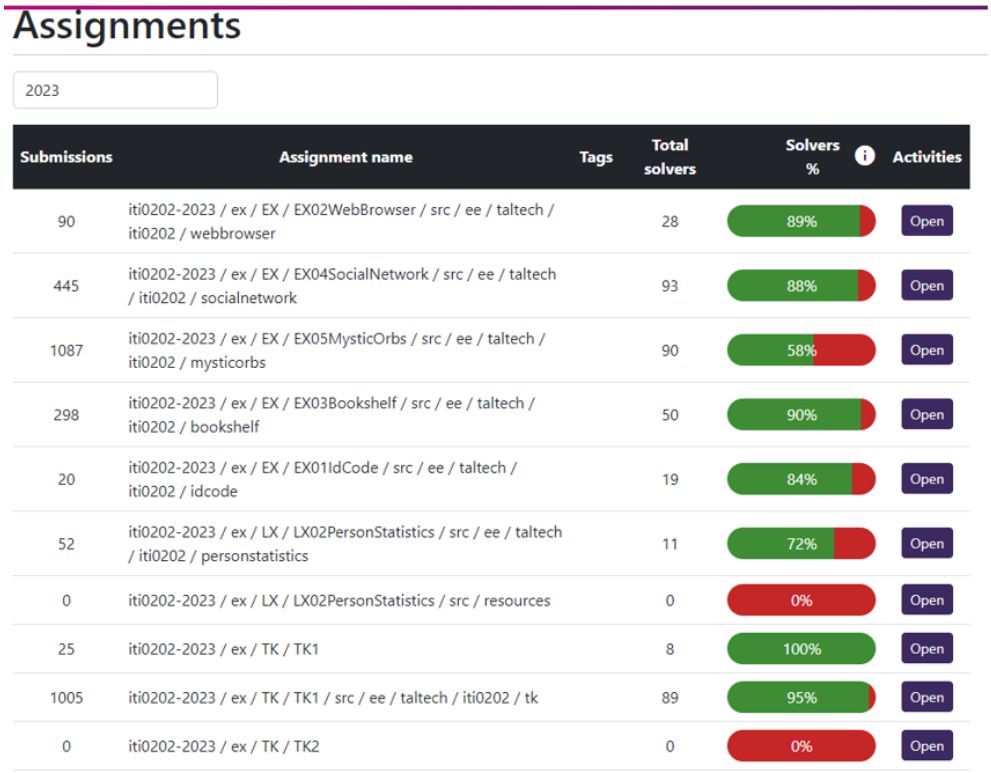
- Esitusi kokku;
- Esitanud õpilaste arv - erinevate õpilaste arv, kes on teinud vähemalt ühe esituse;
- Keskmine läbivuse protsent - kõikide esituste tulemuste keskmine;
- Tudengid, kes said 0%;
- Tudengid, kes said x%-st rohkem;
- Tudengid, kes said x%;
- Keskmiselt esitusi tudengi kohta;
- Keskmine parim tulemus tudengi kohta - summeeritakse iga tudengi parimat tulemust ja jagatakse seda esitanud õpilaste arvuga;
- Keskmine esituste arv tudengi kohta, kes sai x%.



Joonis 16. *Statistika tabelitena.*

5.3.4 Statistika ülesannete tabelis

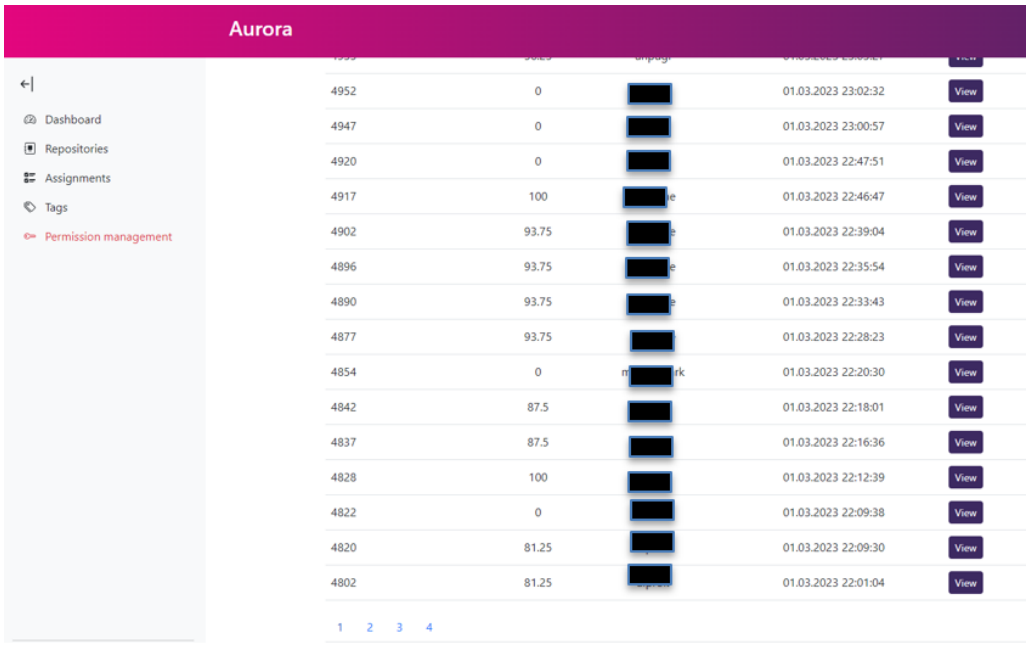
Kõikide ülesannete vaates olevas tabelis, mis on joonisel 17, saab näha triviaalset statistikat kindlat ülesannet avamata. Tabelisse on lisatud ülesande kõikide esituste arv, unikaalsete lahendajate arv ning lahendajate protsent, mis näitab, mitu tudengit tegid vähemalt ühe esituse ja said vähemalt 51% ülesande eest. Roheline ja punane osa muutuvad vastavalt arvutatud protsendile.



Joonis 17. *Kõikide ülesannete vaade statistikaga.*

5.4 Tulemuste jagamine lehekülgedeks

Kuna tuli välja, et olemasoleva rakenduse loogika ei olnud mõeldud suurte andmemahtudega töötamiseks, ilmnes probleem ülesannete tulemuste kuvamisega. Kui rakendusse hakkasid jõudma suured esituste kogused, ei olnud neid võimalik näha, kuna leht laadis väga kaua ning viskas lõpuks `java.lang.OutOfMemoryError: Java heap space` vea. Selleks, et võimaldada kasutajal esitusi näha, jagasime tulemused lehekülgedeks. Joonisel 18 on näidatud, kuidas kasutaja näeb uuemaid esitusi 100 kaupa ning on võimalik minna teistele lehekülgedele, kus saab näha vanemaid esitusi.



Joonis 18. *Ülesande esitused.*

6. Dokumentatsioon

Dokumentatsioon säilitab teadmise tarkvara arendamise protsessist. See sisaldab informatsiooni nõuete, disainiotsuste, testimise ja muude oluliste sammude kohta. Arendajad saavad dokumentatsiooni abil tagasi minna ja jälgida varasemaid otsuseid ning nende mõju süsteemile. See võib olla eriti kasulik uute arendajate jaoks, kes soovivad tarkvara edasi arendada või olemasolevat süsteemi mõista.

Tarkvara dokumentatsioon on oluline hooldatavuse ja jätkusuutlikkuse tagamiseks. Kui algne arendajate meeskond lahkub või lisanduvad uued liikmed, võimaldab dokumentatsioon neil tarkvara mõista ja jätkata selle arendamist. See hõlbustab hooldustegevusi, vigade otsingut ja uuenduste rakendamist tulevikus.

Dokumentatsioon on samuti väärtuslik tööriist koostöö ja kommunikatsiooni toetamiseks erinevate huvigruppide vahel. See võimaldab arendajatel, testimisrühmal, projekti juhil ja klientidel jagada ühist arusaama tarkvara toimimisest. Selge ja arusaadav dokumentatsioon aitab vältida arusaamatusi, vähendada vigu ja tagada parem suhtlus kõigi osapoolte vahel [28].

Hea dokumentatsiooni koostamiseks on vaja aru saada kliendi soovidest. Arendajad peavad mõistma kasutaja vajadusi ja valupunkte arendusprotsessi algusest peale. Dokumentatsioon peaks käsitlema kõiki selliseid vajadusi. Vajadusi saab dokumenteerida kasutusloo kujul [29]. Kasutajalood keskenduvad nõuete ja funktsionaalsuse kogumisele lõppkasutajate jaoks. Selle asemel, et keskenduda tehnilistele detailidele, rõhutavad kasutajalood seda, mida kasutajad soovivad tarkvara abil saavutada. See aitab arendajatel paremini mõista, milliseid probleeme kasutajad soovivad lahendada ja millised funktsioonid neid kõige paremini toetavad. Dokumentatsioon peab samuti olema kergesti arusaadav, lühike, lihtne ja vältima keerulisi avaldusi.

6.1 Aurora rakenduse dokumentatsioon

Dokumentatsiooni koostamise eesmärk oli jätta võimalikult palju infot järgmistele arendajatele, et nende arenduse algus oleks võimalikult kiire ja sujuv.

Aurora Assignments API hoidlas “User stories” wiki leheküljel saab näha rakenduse kasutuslugusid, mis annavad ülevaate, mida saab Aurora rakenduses teha [30]. Samas

hoidlas “Business analysis” wiki leheküljel on kirjeldatud rakenduse äriloogika, mis näitab, kuidas rakendus töötleb andmeid, kuidas see vastab erinevatele kasutajate toimingutele või sisenditele ning kuidas see reageerib erinevatele sündmustele või tingimustele. Meie poolt leitud vigade nimekiri ja nende kirjeldus on leitav wiki leheküljel “Bugs and Problems” [26]. Peatükis “Olemasoleva rakenduse testimine ja analüüs” välja toodud analüüsi tulemusi saab näha sildiga analysis ülesannete kommentaarides. Samuti on meie poolt loodud demo video selleks, et oleks võimalik visuaalselt näha, mida rakendus võimaldab teha [31].

7. Tulemused

Lõputöö eesmärgiks oli lisada uut funktsionaalsust vastavalt kliendi soovidele ning parandada olemasoleva funktsionaalsuse vigu, mis takistasid rakenduse normaalset ja pidevat kasutamist.

Lõputöö tulemusena saab rakendust kasutada pidevalt ilma katkestusteta. Kõige kriitilisemad vead, mis takistasid rakenduse kasutamist, said parandatud. Lõputöö raames saavutatud eesmärgid:

- Rakenduse põhjalik analüüs, probleemide tuvastamine ja parandamine;
- Teostati statistika kogumine ja kuvamine;
- Teostati ülesannetele kommentaaride lisamine;
- Parandati otsing. Otsing ei andnud õigeid tulemusi ning oli väga aeglane;
- Leiti põhjust ja parandati probleem seoses tagarakenduse kokkujooksmisega;
- Rakenduse töökiirust parandati;
- Parandati keele süsteem. Rakenduses on kasutusel kaks keelt: eesti ja inglise. Inglisekeelne versioon ei olnud täielikult eesti keelde tõlgitud;

Uues rakenduse versioonis saab kasutaja näha erinevat statistikat nii hoidlate kui ka ülesannete kohta. Samuti sai loodud ülesannetele kommentaaride lisamine, mis võimaldab kasutajatel luua, kustutada ja uuendada kindla ülesande kommentaare. Kriitiliste parandatud asjade seas on otsing, Java ainete ülesannete esituste ilmumine rakenduses, perioodilise tegumi (*cron job*) abil esituste sidumine ülesannetega (kui esitus oli saadud varem, kui ülesanne ise ilmus Auroras), ülesannete lähtekoodi linkide kokkupanek ning hoidlas olevate ülesannete õigete nimede kuvamine.

Eesmärgid, mis jäid saavutamata:

- Praegu saab rakendust mugavalt kasutada suurtel ekraanidel, st telefoniga pole seda mugav kasutada. Teha reageeriv kasutajaliides (*responsive UI*);
- Parandada viga hoidla kustutamise kohta. Hetkel saab rakendusse lisada hoidlaid, aga kui kustutada Aurora kasutaja GitLabi salvest, siis rakendusest see salv ei kao;

Enne arendust olid eesmärkidele antud prioriteedid ning täitsime eesmärke prioriteetidele vastavalt. Kaks eesmärki jäid saavutamata, kuna arenduse jooksul tuvastasime veel kriitilisi

vigu, mida ei saanud jätta parandamata.

8. Tulemuste testimine ja valideerimine

Iganädalastel koosolekutel arutasime kliendiga, mis oli tehtud peale eelmist koosolekut ning kui uus funktsionaalsus oli pandud serverisse, siis testis, valideeris ja kommenteeris klient teostatud muudatusi ning tegi ettepanekuid, kuidas saaks disaini või funktsionaalsust veel paremaks teha. Seega testis arenduse jooksul klient pidevalt tulemusi.

Kui suurem osa planeeritud parandustest ja edasiarendusest oli teostatud, andsime rakendust testida ka abiõppejõule, kellelt saime põhjaliku kirjaliku tagasiside. Saadud märkuseid kommenteerisime juhul, kui pakutud parandus või edasiarendus ei olnud mõistlik. Häid ettepanekuid võtsime arvesse ning muutsime funktsionaalsust ja disaini vastavalt antud soovitustele. Suures osas käis see tagasiside kasutajaliidese kohta. Kõik parandused, mis teostati saadud tagasiside põhjal, sisalduvad rakenduse viimases versioonis – lõplik tulemus arvestab ettepanekutega.

9. Telija tagasiside

Esimese tähtpunkti jooksul oli klient mures, et ei jõuata tema poolt soovitud funktsionaalsust teostada, kuna peamiselt tegelesime vigade parandamisega. Teise tähtpunkti jooksul sai valmis statistika ja kommentaaride lisamine, millega klient on rahul.

Kliendi poolt saadud tagasiside: „Kuigi süsteem on valmis juba kaks aastat, siis pole seda erinevate probleemide tõttu saanud kasutusele võtta. Antud lõputöö käigus tehtud paranduste ja täienduste abil on süsteem lõpuks kasutatav. Oluline osa süsteemist on see, et ülesande kohta näeb lahendusstatistikat. Eelmise arenduse käigus loodud lahendusandmete kogumisel ei arvestatud sellega, et lahendusi esitatakse palju ning nende töötlemisel jooksis süsteem kokku. Praegused autorid lahendasid selle probleemi. Seeläbi süsteem enam kokku ei jookse.

Lõputöö käigus on parandatud päris palju erinevaid probleeme, mis on kasutusmugavust parandanud. Näiteks ülesannete otsingu päring sai oluliselt kiiremaks. Uue funktsionaalsusena lisati ülesannete kommenteerimise võimalus ning kogutud lahendusstatistika näitamine kasutajale.

Süsteemi kohta on tagasisidet kogutud abiõppejõududelt, kes programmeerimisülesannetega tegelevad. Lõputöö käigus on väga suur hulk erinevaid probleeme lahendatud.“

10. Järeldused

Esialgu tundus, et töötada juba arenduses oleva projekti kallal on lihtsam kui luua midagi nullist. Hiljem tuli välja, et meie arvamus oli ekslik.

Arvasime, et rakenduses olev funktsionaalsus on täiesti töötav. Arenduse käigus saime teada, et projektil on funktsionaalsust, mis ei ole lõpuni tehtud või mis üldse ei tööta. Lisaks sellele kasutasid erinevad arendajad koodi kirjutamisel erinevaid mustreid ning projekti struktuur ei olnud mõeldud töötamiseks suurte andmemahtudega.

Enne uue funktsionaalsuse loomist pidime prioriteete määrama ning olemasolevaid vigu parandama, et rakendust saaks mitte vaid lokaalselt, aga ka päris andmetega kasutada. Vigade parandus võttis rohkelt aega, kuna ei olnud selge, kuidas eksisteeriv loogika on ehitatud ning puudusid dokumentatsioon ja kommentaarid meetodite juures. Lisaks sellele oli väga palju koodi välja kommenteeritud ja esines mittekasutatatud meetodeid ning klasse.

Vaatamata raskustele suutsime välja selgitada, kuidas rakendus töötab. Vead, mis takistasid rakendust normaalset kasutamast, said parandatud. Peale vigade parandamist algas töö uue funktsionaalsuse kallal ja selles takistusi ei tekkinud, kuna kaalusime erinevaid lahendusviise ning valisime kõige sobilikumad. Kõik tähtsad asjad olid kliendi ja arendajate vahel eelnevalt läbi arutatud.

Arendusmeeskonnal oli väga hea koostöö ning kui tekkisid kahtlused välja mõeldud lahendustes, leidis see kohe arutlust koosolekutel. Eesmärgiks oli teha pidevat arendustööd, mitte jätta seda viimaseks momendiks. Meil oli piisavalt aega, et läbi mõelda iga lahenduse detailid, kaaluda ja arutada erinevaid lahendusviise ning valmistuda demopäevadeks. Samuti toimus pidev suhtlus ja tagasiside nii juhendaja kui ka kliendi poolt.

Jõudsime järelduseni, et mitte alati ei ole lihtsam töötada arenduses oleva projekti kallal, sest projektidel võib puududa piisav kirjeldus või dokumentatsioon ning seetõttu ei saa eelnevalt tuvastada, kui palju probleemset ja mittetöötavat funktsionaalsust rakenduses on ning kui kvaliteetsed on olemasolevad lahendused. Projektiga tegelemiseks tasub varuda piisavalt aega ja alustada võimalikult varakult ehk mitte jätta kõike viimasele hetkele, sest kunagi ei ole võimalik täpselt ennustada projekti struktuurist ja rakenduse funktsionaalsusest arusaamiseks ning sobivate lahenduste leidmiseks kuluvat aegas. Saime

aru, et hea koostöö ja kiire reageerimine on samuti arenduse juures olulised asjad ning kui probleeme esimesel võimalusel ei arutata, võib see tekitada viivitusi arendusprotsessis.

11. Edasised sammud

Vaatamata sellele, et praegune rakenduse versioon pakub laialdast funktsionaalsust, leidub mitmeid võimalusi edasiseks arendamiseks, mis võiksid suurendada Aurora rakenduse väärtust ja kasulikkust.

Aurora rakenduse võimalikud täiendused on järgmised:

- Lisada autentimise võimalus läbi TARA (*The State Authentication Service*). Rakendused, mida kasutatakse ülikooli õppimise jooksul, võimaldavad TARA autentimist, näiteks Moodle või ÕIS (õppeinfosüsteem), mis annab kasutajale võimaluse erinevatel viisidel sisse logida.
- Teha reageeriv kasutajaliides, et rakendust oleks mugavam kasutada telefoniga.
- Teostada ülesannete koodi sarnasuse kuvamist (vt 4.1.9 *Query results in file* funktsionaalsus).
- Lisada piiranguid kasutaja tegevustele:
  - Lõpmatult *check queue*'sse hoidlate lisamisele (vt 4.1.14 Lõpmatult check queue'sse hoidlate lisamine).
  - Siltide grupi kustutamisele (vt 4.1.13 Siltide grupi kustutamine).
  - Siltide ülesandelt kustutamisele (vt 4.1.10 Ülesandelt siltide kustutamine ilma kinnituseta).
- Hoidlate kustutamise funktsionaalsus (vt 4.1.12 Hoidlate kustutamine).
- Ülesannete tabelis oleva info sorteerimine veergude järgi.
- Automaatne sildistamine (vt 4.1.11 Automaatne sildistamine ja Add to tagging queue).
- Eelmise vaate säilitamine. Avades kindla ülesande ülesannete tabeli vaatest ning minnes sealt tagasi, viiakse kasutaja ülesannete tabeli vaate algusesse, mitte sinna, kust ülesandele mindi.

12. Kokkuvõtte

Aurora rakendus on mõelnud TalTechi programmeerimiseainete õppejõududele, mis annab hea ülevaate kõikidest programmeerimisülesannetest. Rakendus võimaldab kasutajal hallata GitLabi hoidlaid ja hoidlates olevaid ülesandeid ning näha erinevat statistikat.

Rakendus oli arenduses kahe aasta jooksul, kuigi seda ei olnud võimalik kasutada erinevate vigade ja puuduliku funktsionaalsuse pärast.

Lõputöö eesmärgiks oli lisada uut funktsionaalsust vastavalt kliendi soovidele ning parandada olemasolevat funktsionaalsust, mis takistasid rakenduse normaalset ja pidevat kasutamist.

Projekti raames kasutasime agiilset tarkvara arendamise meetodit Scrum, tegumikeskse kasutajaliidese disaini (*Task-centered User Interface Design*) ning The Three Principal Layers printsiipi.

Enne lõputöö tegemist tundus, et juba arenduses oleva projekti kallal on lihtsam töötada kui luua midagi uut nullist. Hiljem tuli välja, et meie arvamus oli ekslik, sest ainult arenduse käigus saime teada, et projektil on palju funktsionaalsust, mis ei ole lõpuni tehtud või mis üldse ei tööta. Projekti kallal oli alguses raske töötada, kuna ei olnud selge, kuidas eksisteeriv loogika on ehitatud, dokumentatsioon ning kommentaarid meetodite juures puudusid, palju koodi oli välja kommenteeritud ja esinesid mittekasutatavad meetodid ning klassid.

Vaatamata raskustele suutsime välja selgitada, kuidas rakendus töötab. Vead, mis takistasid rakenduse normaalset kasutamist, said parandatud ning kliendi poolt soovitud funktsionaalsus sai lisatud. Lõputöö raames saavutasime suurema osa püstitatud eesmärkidest.

Kasutatud kirjandus

[1] O. Pihlak. *Programmeerimisülesannete haldamise register Aurora*. [Kasutatud 18 Aprill 2023]. URL: https://digikogu.taltech.ee/et/Item/d099c42a-131b-480e-81b8-9677d0cee46d.

[2] R. Juurik J. Serkin. *Programmeerimisülesannete haldamise registri Aurora õiguste ja statistika süsteemi juurdearendus*. [Kasutatud 18 Aprill 2023]. URL: https://digikogu.taltech.ee/et/Item/0d8bbf2e-f3a0-4b56-be25-edc0e35ed2e2.

[3] *Spring Security OAuth 2 Guides*. [Kasutatud 18 Aprill 2023]. URL: https://www.baeldung.com/spring-security-oauth.

[4] *Scrum (software development)*. [Kasutatud 18 Aprill 2023]. URL: https://www.scrum.org/resources/what-scrum-moduleh.

[5] P. D. Nusantara Y. Yanfi. *UI/UX design prototype for mobile community-based course, 2023*. [Kasutatud 18 Aprill 2023].

[6] *Task-Centered User Interface Design: A Practical Introduction by Clayton Lewis and John Rieman*. [Kasutatud 18 Aprill 2023]. URL: http://grouplab.cpsc.ucalgary.ca/saul/hci_topics/tcsd-book/chap-1_v-1.html.

[7] M. Fowler. "The Three Principal Layers". In: *Patterns of enterprise application architecture*. 2002.

[8] *Thesis_Julia_Ellina*. [Kasutatud 18 Aprill 2023]. URL: https://gitlab.cs.ttu.ee/aurora/thesis_julia_ellina/-/wikis/home.

[9] S. Agarwal A. Sharma M. Kumar. In: *A Complete Survey on Software Architectural Styles and Patterns*. 2015.

[10] *Spring Boot dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/documentation.html.

[11] *Security with Spring*. [Kasutatud 18 Aprill 2023]. URL: https://www.baeldung.com/security-spring.

[12] *Gradle vs Maven Comparison*. [Kasutatud 18 Aprill 2023]. URL: https://gradle.org/maven-vs-gradle/#:~:text=Gradle%20allows%20custom%20dependency%20scopes,dependency%20found%20in%20the%20graph.

[13] *Gradle dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://gradle.org/.

[14] *Mockito dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://site.mockito.org/.

[15] *Swagger dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://swagger.io/.

[16] *Introduction to Project Lombok*. [Kasutatud 18 Aprill 2023]. URL: https://www.baeldung.com/intro-to-project-lombok.

[17] *Introduction to SLF4J*. [Kasutatud 18 Aprill 2023]. URL: https://www.baeldung.com/slf4j-with-log4j2-logback.

[18] *Angular dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://angular.io/.

[19] *Bootstrap dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://getbootstrap.com/.

[20] *Angular Material Dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://material.angular.io/.

[21] *SASS dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://sass-lang.com/.

[22] *Chart.js dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://www.chartjs.org/.

[23] *Hibernate dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://hibernate.org/.

[24] *Use Liquibase to Safely Evolve Your Database Schema*. [Kasutatud 18 Aprill 2023]. URL: https://www.baeldung.com/liquibase-refactor-schema-of-java-app.

[25] *Docker dokumentatsioon*. [Kasutatud 18 Aprill 2023]. URL: https://www.docker.com/.

[26] *Bugs and Problems*. [Kasutatud 14 Mai 2023]. URL: https://gitlab.cs.ttu.ee/aurora/aurora-backend/-/wikis/Bugs/Bugs-and-Problems.

[27] *Group and project members API*. [Kasutatud 18 Aprill 2023]. URL: https://docs.gitlab.com/ee/api/members.html#remove-a-member-from-a-group-or-project.

[28] *Software documentation*. [Kasutatud 18 Aprill 2023]. URL: https://www.techtarget.com/searchsoftwarequality/definition/documentation.

[29] *User stories with examples and a template*. [Kasutatud 14 Mai 2023]. URL: https://www.atlassian.com/agile/project-management/user-stories#:~:text=software%20user's%20perspective.-,A%20user%20story%20is%20an%20informal%2C%20general%20explanation%20of%20a,value%20back%20to%20the%20customer.

[30] *User stories*. [Kasutatud 14 Mai 2023]. URL: https://gitlab.cs.ttu.ee/aurora/aurora-backend/-/wikis/User-stories.

[31] *Aurora demo video*. [Kasutatud 14 Mai 2023]. URL: https://youtu.be/uLpVhcgJHNo.

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

Meie, Julia Djomina ja Ellina Gedrojets

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Programmeerimisülesannete haldamise registri Aurora täiendused ja parandused", 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. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

23.05.2023

[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.

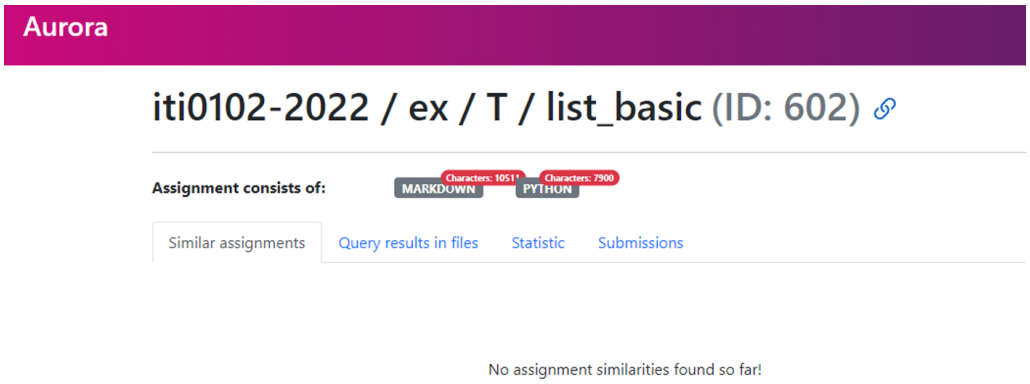
Lisa 2 - Perioodilise tegumi vana loogika

```
@Scheduled(cron = "${app.external-submission-resolver.cron}")
public void resolverJob() {
    List<ExternalSubmission> unreferencedExternalSubmissions = externalSubmissionRepository.findByAssignment(null);
    Map<Pair<String, String>, List<ExternalSubmission>> groupedByAssignmentParams = unreferencedExternalSubmissions.stream()
        .collect(Collectors.groupingBy(externalSubmission -> new ImmutablePair<>(
            externalSubmission.getRepoPath(),
            externalSubmission.getAssignmentDir()
        )));

    groupedByAssignmentParams.forEach((stringStringPair, externalSubmissions) -> {
        Optional<Assignment> relatedAssignment = getRelatedAssignment(stringStringPair.getLeft(),
            stringStringPair.getRight());

        if (relatedAssignment.isPresent()) {
            List<ExternalSubmission> linkedExternalSubmissions = externalSubmissions.stream()
                .map(externalSubmission -> externalSubmission.setAssignment(relatedAssignment.get()))
                .collect(Collectors.toList());
            externalSubmissionRepository.saveAll(linkedExternalSubmissions);
        }
    });
}
```

Lisa 3 – Ülesande vaade koos nupuga, mis viib kasutaja ülesande lähtekoodi lehele



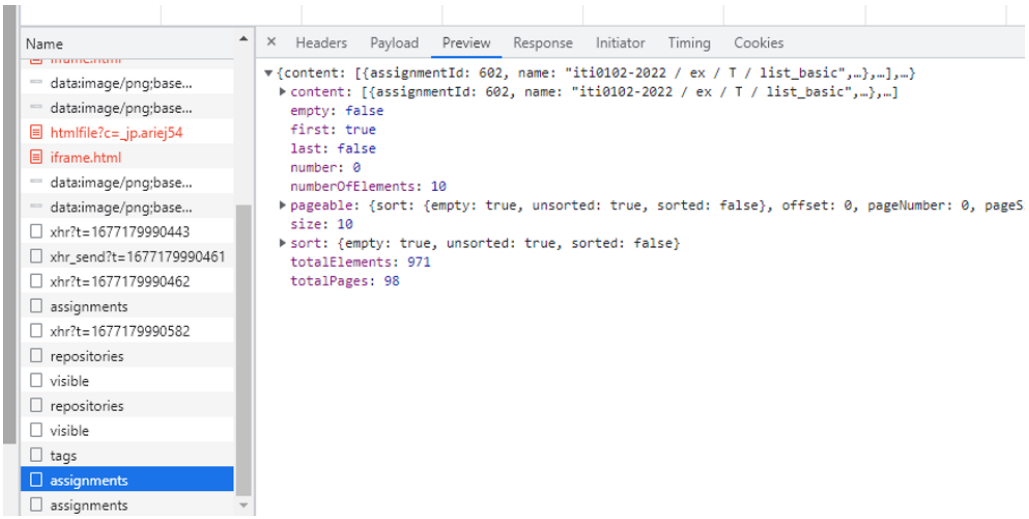
Lisa 4 – Kõikide ülesannete vaade

Assignments

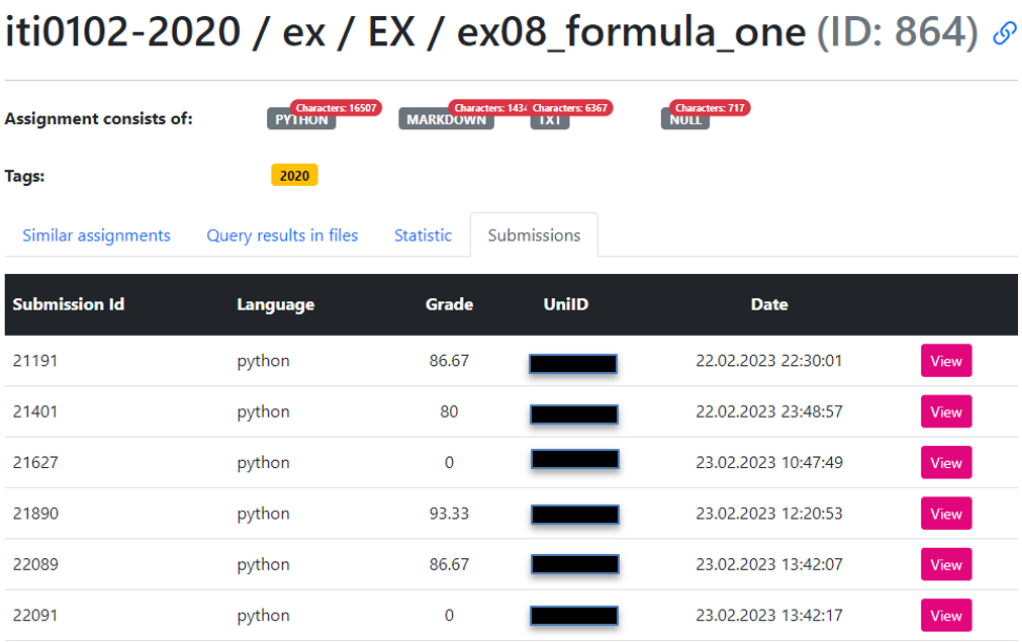
Search

Submissions	Assignment name	Tags	Total solvers	Solvers %	Activities
0	iti0102-2021 / ex / XP / xp05_skynet / images		0	0%	Open
468	iti0202-2023 / ex / PR / PR04Stock / test		1	100%	Open
0	iti0102-2022 / ex / EX / ex04_lists / 4		0	0%	Open
0	judjom / iti0202-2021 / EX / EX07Coffee / src / ee / taltech / iti0202 / coffee / builder		0	0%	Open
0	iti0102-2019 / ex / first_half		0	0%	Open
0	iti0102-2019 / ex / xp08_investor		0	0%	Open
0	judjom / iti0202-2021 / PR / PR02DataStructures / src / ee / taltech / iti0202 / datastructures		0	0%	Open
0	judjom / iti0202-2021 / EXAM / EXAM1		0	0%	Open
0	iti0202-2022 / ex / KT / KT1		0	0%	Open
0	iti0102-2019 / ex / t05_starry_night		0	0%	Open

Lisa 5 – Brauseri konsool, kus on näha, et tehakse 2 päringut järjest



Lisa 6 – Pythoni kursuse ülesanne, kuhu jõuavad tulemused



iti0102-2020 / ex / EX / ex08_formula_one (ID: 864)

Assignment consists of: PYTHON Characters: 16507 · MARKDOWN Characters: 143 · TXT Characters: 6367 · NULL Characters: 717

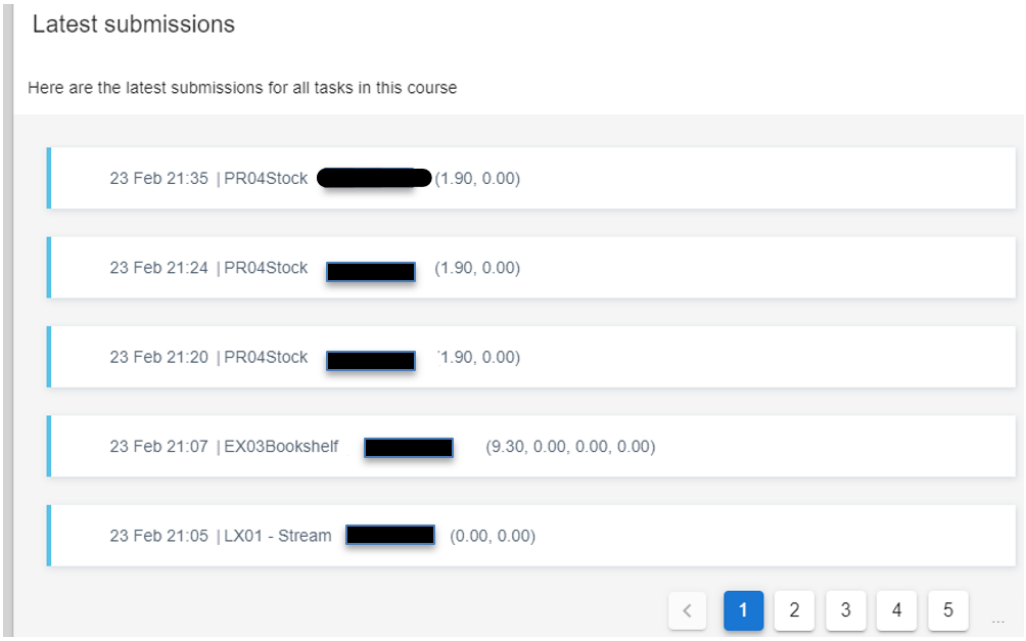
Tags: 2020

Similar assignments | Query results in files | Statistic | Submissions

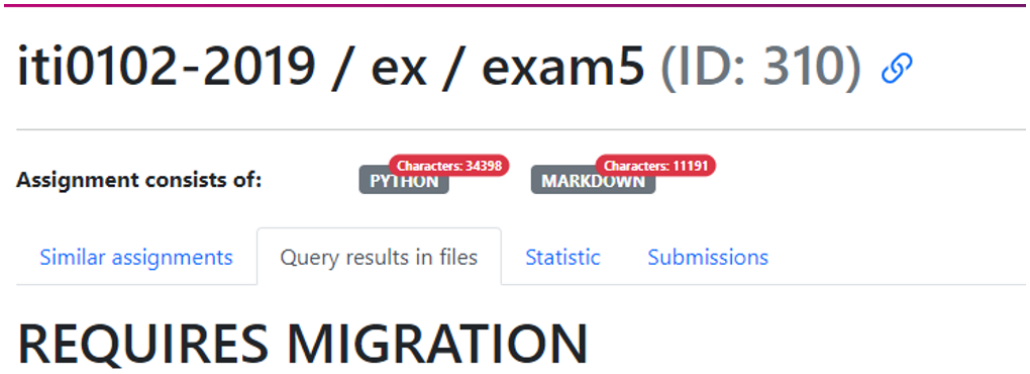
Submission Id	Language	Grade	UniID	Date	
21191	python	86.67	[redacted]	22.02.2023 22:30:01	View
21401	python	80	[redacted]	22.02.2023 23:48:57	View
21627	python	0	[redacted]	23.02.2023 10:47:49	View
21890	python	93.33	[redacted]	23.02.2023 12:20:53	View
22089	python	86.67	[redacted]	23.02.2023 13:42:07	View
22091	python	0	[redacted]	23.02.2023 13:42:17	View

Lisa 7 – Java kursuse ülesanne, kuhu tulemused ei jõua

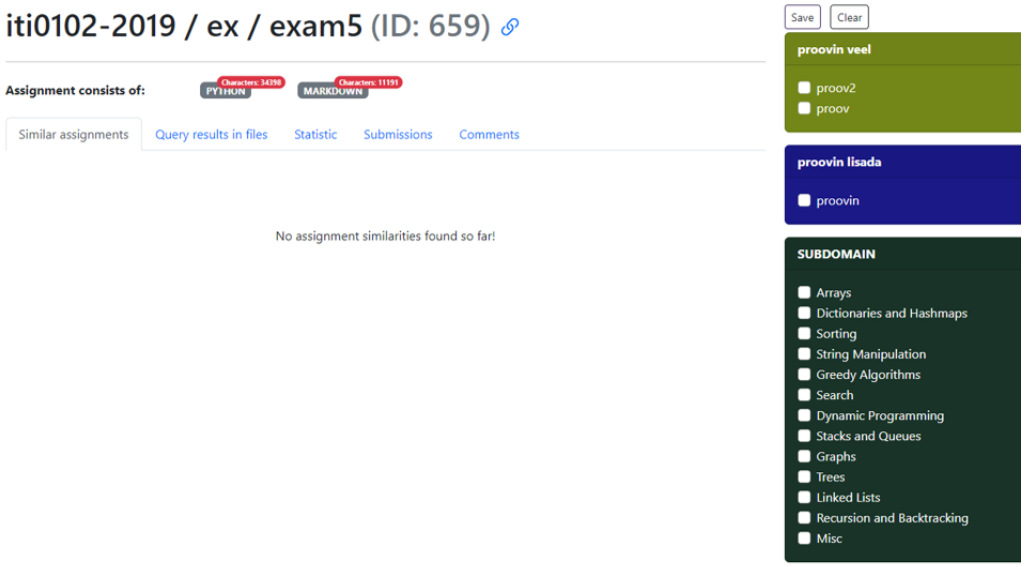
Lisa 8 – Java kursuse tudengite tulemused Charon süsteemis



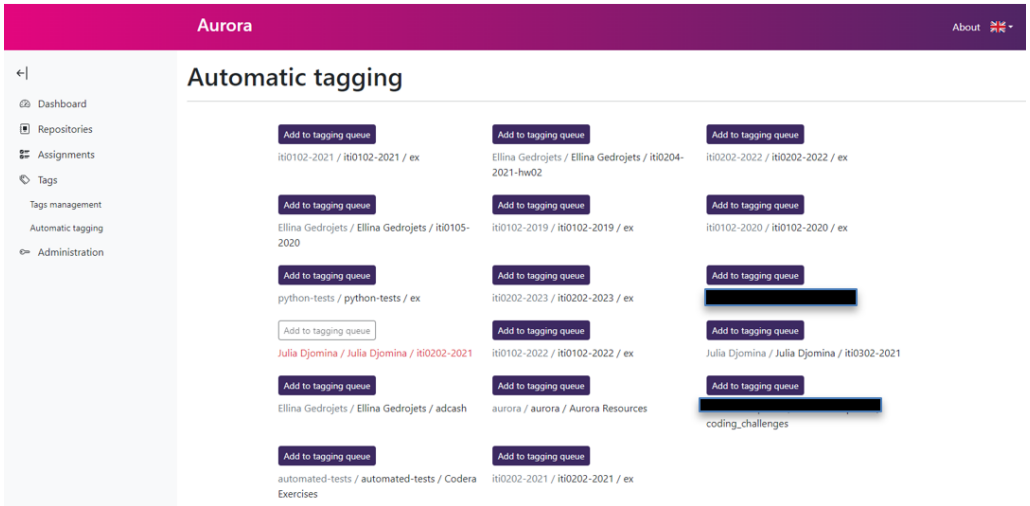
Lisa 9 – Ülesande vaade, kui avada *query results in file*



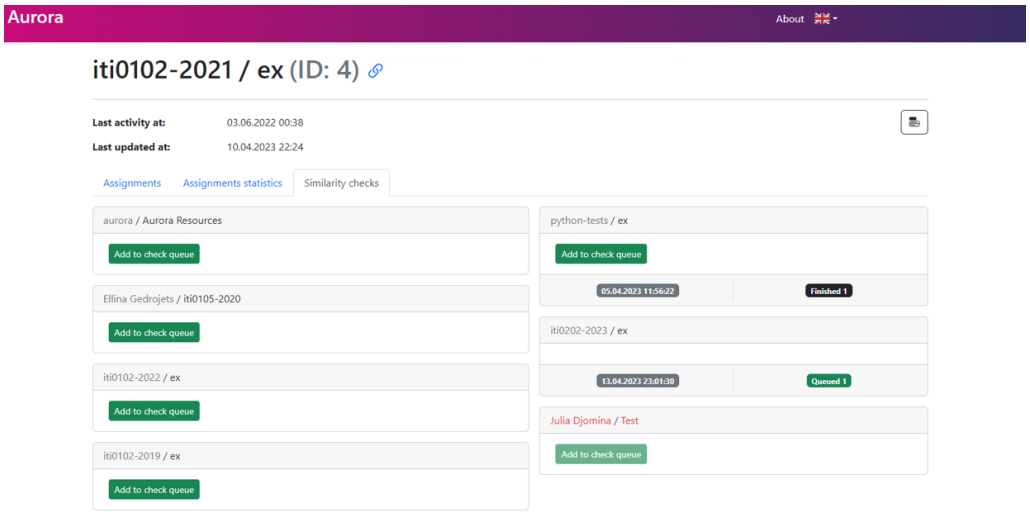
Lisa 10 – Ülesande vaade koos siltidega ja *Save, Clear* nuppudega



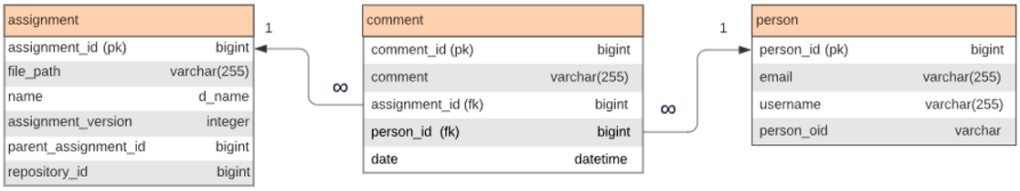
Lisa 11 – *Automatic tagging* vaade



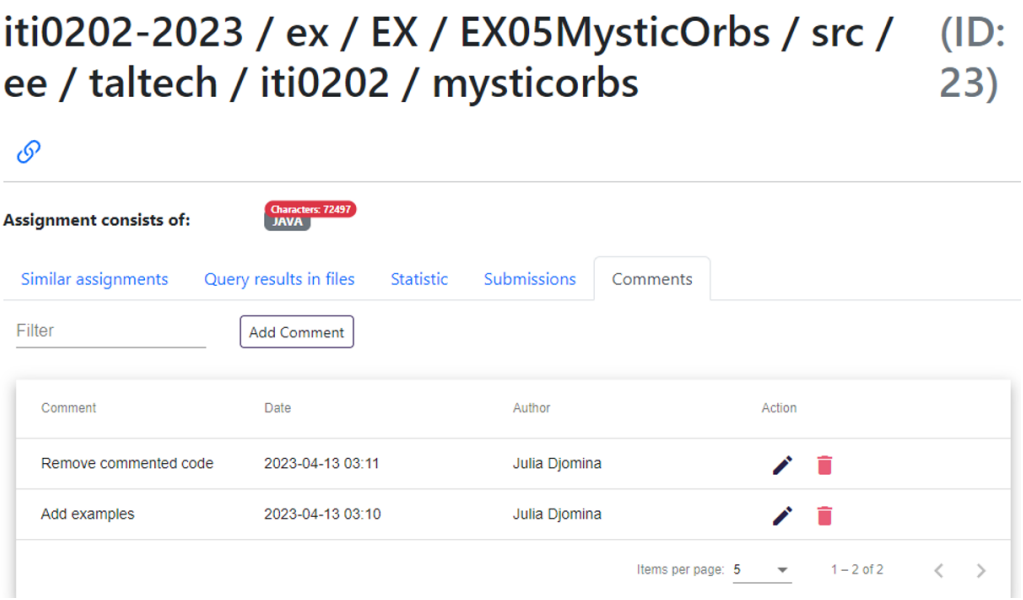
Lisa 12 – *Add to check queue* hoidla vaates



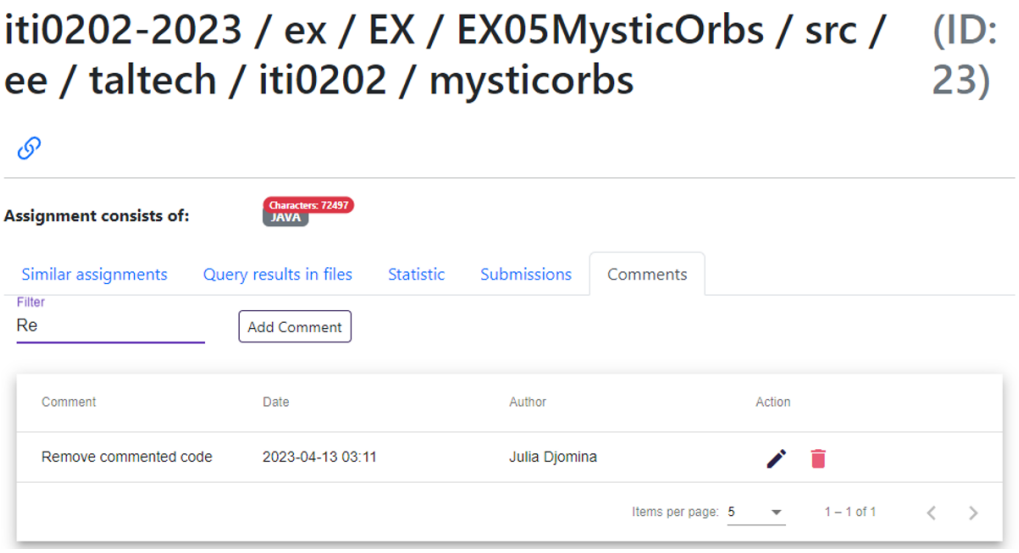
Lisa 13 – Ülesandele kommentaaride lisamine



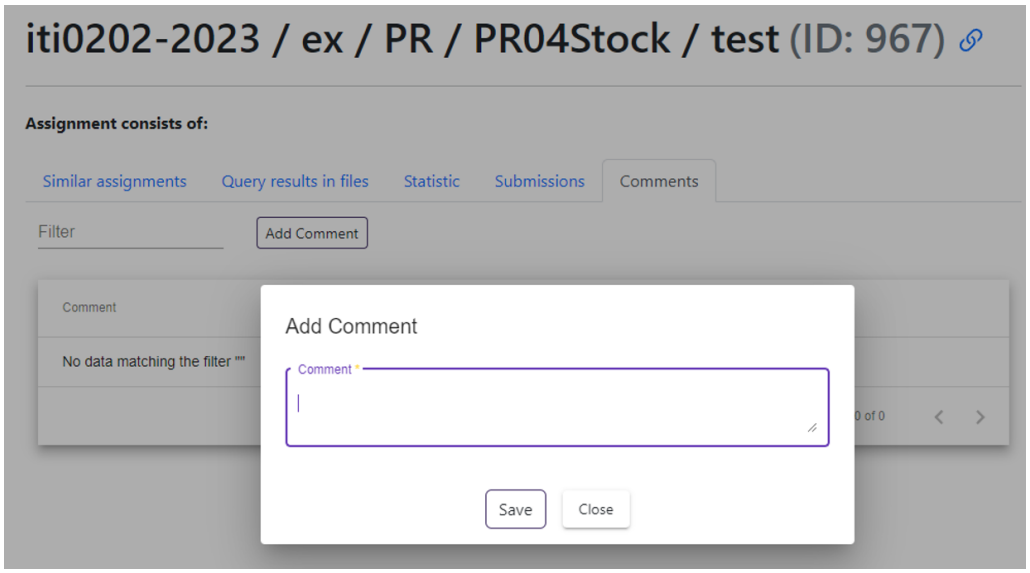
Lisa 14 – Ülesandele lisatud kommentaarid



Lisa 15 – Ülesande kommentaaride filtreerimne



Lisa 16 – Ülesandele kommentaari lisamine



Lisa 17 – Kursoriga täpile liikudes kuvatud informatsioon

Lisa 18 – Statistika arvutamise algus- ja lõppkuupäeva valimine