TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Robin Nook 223000IAIB

Tudengite koodi analüüsimine süntaksipuuga

Bakalaureusetöö

Juhendaja: Ago Luberg

PhD

Tallinn 2025

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: Robin Nook

06.06.2025

Annotatsioon

Lõputöö eesmärk on luua rakendus TalTechi programmeerimise algkursuse abiõppejõududele, mis võimaldab saada kiiret tagasisidet tudengi koodi kohta. See aitab leida õppijate koodis vigu ning annab soovitusi, kuidas koodi parendada. Töös on välja toodud varasemad sarnased lahendused, erinevate tehnoloogiate ning tööriistade valikuid, rakenduse arhitektuur ning arendusprotsess.

Töö tulemusena valmib kasutajaliides, kus saavad abiõppejõud teha päringuid serverirakendusele, mis võrdleb omavahel tudengi- ja õppejõu koode ning saadab vastusena tulemused tagasi kasutajaliidesesse kuvamiseks.

Tulemusi valideeriti jooksvalt kliendiga, kes oli ühtlasi ka lõputöö juhendaja ning programmeerimiskursuse abiõppejõududega, kes rakendust kasutama hakkaksid. Rakendus sai väga positiivse tagasiside ning paljud abiõppejõud tõid välja, et neile oleks sellisest rakendusest kindlasti abi.

Töös on kirjeldatud ka võimalikud edasiarendused, mis põhinevad enamjaolt abiõppejõudude soovitustel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 6 peatükki, 8 joonist.

Abstract
Analysis of Students' Code Using a Syntax Tree

The aim of this thesis is to create an application for teaching assistants of the introductory programming course at TalTech, enabling them to quickly receive feedback on students' code for error identification or providing suggestions on how to improve the code. The thesis outlines previous similar solutions, choices of various technologies and tools, the application's architecture and the development process.

The outcome is a user interface through which teaching assistants can make requests to a server application. The server application then compares the student's and teacher's code and returns the results for displaying in the user interface.

The results were validated continuously with the client, who was also the supervisor for this thesis, as well as with the teaching assistants who would be using the application. The application received highly positive feedback and many teaching assistants indicated that such an application would certainly be beneficial for them.

The thesis also describes potential future developments, mostly based on recommendations from the teaching assistants.

The thesis is in Estonian and contains 29 pages of text, 6 chapters, 8 figures.

Lühendite ja mõistete sõnastik

API	Rakendusliides (*Application Programming Interface*)
AST	Abstraktne süntaksipuu (*Abstract Syntax Tree*)
DSL	Domeeni spetsiifiline keel (*Domain Specific Language*)
ESM	ECMAScript moodulid (*ECMAScript Modules*)
GUI	Graafiline kasutajaliides (*Graphical User Interface*)
GPT	Graafiline kasutajaliides (*Generative Pre-training Transformer*)
HTTP	Hüperteksti edastuse protokoll (*HyperText Transfer Protocol*)
JSON	JavaScripti objektide notatsioon (*JavaScript Object Notation*)
LLM	Suur keelemudel (*Large Language Model*)
OOP	Objektprogrammeerimine (*Object-Oriented Programming*)
REST	Esitusoleku siire (*Representational State Transfer*)
SVG	Skaleeritav vektorgraafika (*Scalable Vector Graphics*)

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tallinna Tehnikaülikooli infotehnoloogia teaduskonnas läbib programmeerimise algkursust igal aastal üle 100 tudengi. Kursusel lahendavad õpilased programmeerimise ülesandeid, mille on varasemalt loonud kursuse abiõppejõud. Samuti on koostatud iga ülesande jaoks ka töötav lahendus, et näidata ülesande võimalikku lahenduskäiku.

Algkursusel on suur osa tudengitest esmakursuslased, kellel varasem programmeerimise kogemus puudub. Seetõttu tekib tihti olukordi, kus õpilane vajab õppejõult nõu või koodi kohta tagasisidet. Aine õppejõududel on keeruline õppijate rohkuse tõttu iga tudengi koodi kiirelt ja konstruktiivselt tagasisidestada. Ülesannete jaoks on tehtud valmis automaattestid, millele õppejõud õpilase koodi tagasisidestamisel tugineda saavad, kuid sageli ei pruugi need olla piisavad, et kiirelt mõista, miks tudengi kood ei tööta.

Eriti oluline selles õppeaines on tagasisidestamise kiirus, sest ainet läbib palju tudengeid ning abiõppejõude ei ole piisavalt. Autor on samuti varasemalt ise olnud abiõppejõu rollis olnud ning mõistab hästi, kuidas klassiruumis tekivad järjekorrad tudengitest, kellel on abi vaja.

Käesoleva bakalaureusetöö eesmärk on luua kasutajaliides koos serverirakendusega, mis võimaldab abiõppejõududel kiirelt ja mugavalt võrrelda tudengi koodi varasemalt õppejõudude poolt koostatud töötava lahendusega. Loodav rakendus peab võimaldama abiõppejõududel kergelt kätte saada nii õpilase kui ka lahenduse koodiplokid ning kuvama visuaalselt nende erinevusi. Samuti peab rakendusel olema funktsionaalsus, mille kaudu saab konfigureerida erinevaid rakenduse aspekte enne võrdluse tegemist.

2 Projekti ülesehitus

Selles peatükis tuuakse välja funktsionaalsed nõuded, millele tuginetakse rakenduse arendusprotsessis. Samuti uuritakse varasemaid lahendusi ning nende kasulikkust käesoleva lõputöö raames. Lõpuks tuuakse välja erinevad tehnoloogiad ning tööriistad, mida lõputöö käigus kasutatakse.

2.1 Funktsionaalsed nõuded

Funktsionaalsete nõuete määramisel peab arvestama, et tegemist on esmase prototüübiga, mille eesmärk on luua rakendus, mida saaks võimalikult varsti kasutada õppetöös tudengite koodi tagasisidestamisel. Samuti on arvestatud, et antud rakendust saaks võimalikult kergelt tulevikus edasi arendada. Kogu rakendus on jaotatud kaheks osaks - kasutajaliides ning serverirakendus. Välja on toodud osadele vastavad nõuded ning üldisemad nõuded kogu rakendusele.

2.1.1 Kasutajaliidese nõuded

Kasutajaliidese töö nõuded:

- **Koodifailide leidmiseks sisendi vorm**
  - Kasutajaliideses peab olema sisendi vorm, kuhu kasutaja sisetab vajamineva info, mis edasi saadetakse serverirakenduse API lõpp-punkti.
  - Vorm peab olema kasutaja jaoks kiiresti kasutatav ja intuitiivne.
  - Vorm peab sisaldama filtreid, et kasutajal oleks võimalik mõjutada võrdlusprogrammi tulemust vastavalt oma vajadustele.
- **Võrdlusprogrammi kokkuvõte**
  - Kasutajaliideses peab olema esmalt nähtav eraldi võrdlusprogrammi kokkuvõtte sektsioon, kus on võimalik kasutajal saada kindla struktuuriga kokkuvõte iga funktsiooni kohta eraldi.
  - Kokkuvõtte sektsioonis peab olema lühike ja üldine kokkuvõte.

- **Võrdlustulemused**
  - Kokkuvõtte sektsioonile järgneb võrdlustulemuste jaotus, kus on välja toodud võrdlustulemused vastavalt võrdluse tüübile.
  - Struktuuri jaotus toimub eelkõige faili, funktsiooni ja võrdluse tüübi järgi.
  - Võrdluse tüüpide tulemustest peab vähemalt üks olema nähtav visuaalselt, mitte tekstiliselt.

2.1.2 Serverirakenduse nõuded

Serverirakenduse töö nõuded:

- **API lõpp-punkt**
  - Serverirakenduses peab olema defineeritud üks põhiline lõpp-punkt, mille kaudu saadab kasutajaliides serverirakendusele päringu võrdlusprogrammi kasutuseks.
  - Lõpp-punkt peab vastu võtma ning tagastama kõik andmed kindla struktuuriga.
- **Koodiplokkide toomise funktsionaalsus**
  - Serverirakenduses peab olema funktsionaalsus, mis aitab tuua koodiplokid võrdlusprogrammi.
- **Võrdlusprogramm**
  - Serverirakenduses peab olemas olema keskne võrdlusprogramm, mis saab sisendiks tudengi ja õppejõu koodid ning tulemuseks annab võrdlustulemused ning kokkuvõtte.

2.1.3 Üldised nõuded

Üldised nõuded:

- **Kättesaadavus õppejõududele**
  - Rakendus peab olema kiiresti kättesaadav ning ei vaja eelnevat allalaadimist.
  - Rakendusel peab olema autentimine, mis võimaldaks rakendusele ligipääsu ainult õppejõududel.
- **TalTechi stiiliraamat**
  - Rakendus on mõeldud kasutamiseks TalTechi õppejõududele, seega peab see välja nägema visuaalselt sarnane teiste TalTechi rakendustega.

2.2 Varasemad lahendused

Varasemate lahenduste leidmine oli oodatust keerulisem, kuna rakendusele määrati spetsiifilised nõuded ning sellist lahendust varem veel tehtud ei ole. Üldiselt kasutatakse AST baasil võrdlust rangelt semantiliste erinevuste leidmiseks või plagiaadituvastuseks koodiplokkide vahel. Sellegipoolest leidusid mõned head näited, millel on antud töö skoobiga ühiseid jooni.

2.2.1 Code Diff

Cedric Richteri poolt loodud AST baasil algoritm, mis on mõeldud erinevate koodiversioonide võrdlemiseks. Algoritmi eesmärk on aidata arendajatel kiiresti ja täpselt tuvastada muudatusi koodis. Tema lahendus võimaldab tuvastada semantilisi erinevusi koodis, mis võivad teistes tekstipõhistes koodivõrdluse algoritmides jääda märkamata [1].

Code Diff lahenduse tugevuseks on eelkõige oskus eristada väikseid, kuid sisulisi muudatusi koodis. Selle algoritmi kasutamine vähendab hooletusvigu ja kiirendab tagasiside protsessi, mis on kasulik antud rakenduse kontekstis, kus on vaja kiiresti hinnata ja võrrelda väga paljude tudengite ülesandeid. Samuti on see algoritm mõeldud funktsioonide ja koodiplokkide tasandi võrdluseks.

Antud lahenduse miinusteks on abstraktne väljund, mis ei ole piisavalt selge ja intuitiivne õppejõududele, kes soovivad kohe saada selgelt loetavaid ja struktureeritud tulemusi. Samuti pole võimalik antud algoritmi väljundit kohandada vastavalt filtritele. Suurimaks miinuseks on lahendusel puudulik tulemuste grupeerimise võimalus, kui on palju sarnaseid muudatusi, nagu järjest sarnaste koodiridade kustutamine.

2.2.2 Mayat

Tian Yangi poolt loodud AST baasil algoritm, mis võrdleb mitmeid koodiplokke korraga. Iga koodiploki paari vahelise võrdluse tulemuseks toob see välja skoori vastavalt sellele, kui sarnased antud koodiplokid on. See algoritm võimaldab kiirelt analüüsida suurt hulka koodiplokke, andes tulemused arusaadavas skaalas, mis on kasulik, kui on vaja võrrelda mitmete tudengite koodi sarnasust [2].

Selle tugevusteks ongi just see arusaadavas skaalas sarnasuse mõõtmine, antud rakenduse

puhul siis tudengi ja õppejõu lahenduse vahel. Kindlasti oleks sellisest lahendusest kasu tulevikus plagiaadituvastuseks, kui rakenduse edasiarenduses laiendatakse skoopi mitmete tudengite omavahelistele koodivõrdlustele. Paraku on antud töö raames selle kasulikkus piiratud, sest hetkel jääb töö skoopi ainult kahe koodiploki võrdlus. Samuti on sarnasuse määramine üsna pinnapealne ning ei paku sügavamat analüüsi koodi erinevustesse, mis aitaks lõpuks õppejõul mõista, mis võib tudengi koodis valesti olla.

2.2.3 PyFlowchart

CDFMLR-i poolt loodud AST baasil tööriist, mis võtab sisendiks koodiploki ning tulemuseks väljastab vooskeemi. See lahendus võimaldab visualiseerida koodi struktuuri ja loogikat, teisendades koodiploki *flowchart.js* DSL-i, mis seejärel kuvatakse kasutajaliideses interaktiivse vooskeemina [3].

Tööriista tugevuseks on koodi visuaalne esitus, mis aitab kergemini mõista koodi tööd. Selline lahendus on väga kasulik antud töö kontekstis, sest selle abil on õppejõududel palju kergem süveneda tudengi koodi struktuuri ja loogikasse. Samuti on see tööriist väga kohandatav vastavalt vajadustele, näiteks saab seal grupeerida erinevaid käskude jadasid, mis muidu oleks erinevad osad vooskeemis. Samuti on võimalik vooskeemi värvida ning erinevate joonestiilidega kujundada.

Selle lahenduse puuduseks on vähene semantiline analüüs ehk tööriist keskendub eelkõige struktuuri visualiseerimisele ja mitte koodimuudatuste detailsele analüüsile. Seetõttu oleks vaja seda tööriista kasutada antud rakenduses kooskõlas mingi teise semantilise võrdluse algortmiga.

2.3 Tehnoloogiad ja tööriistad

Tehnoloogiate valik sõltus töö funktsionaalsetest nõuetest, kiirusest, töökindlusest, populaarsusest ja paindlikkusest. Samuti lähtuti palju autori varasemast kogemusest antud tehnoloogiate ning tööriistadega.

2.3.1 Python

Python on maailma üks populaarsemaid objekt-orienteeritud programmeerimiskeeli [4]. Pythonit kasutatakse laialdaselt GUI-, mängude-, AI- ja andmeteaduse programmeerimisel. Maailma tuntuimad veebiplatvormid nagu Google, Facebook ja Amazon kasutavad kõik Pythonit ning selle pistikprogramme [5].

Pythoni suurimateks plussideks loetakse selle keele süntaksi lihtsust. Samuti on tegemist keelega, mis kasutab dünaamilisi tüüpe ehk arendajatel ei ole kohustust muutujate tüüpe defineerida. Keelel on ka väga suur ja toetav kommuun, kelle abil on loodud palju erinevaid väga kasulikke pistikprogramme [6]. Keele populaarsuse tõttu on see õpitav ka TalTechi programmerimise algkursuse aines.

Valitud sai Python eelkõige väga hea AST toe tõttu, mis on antud töö kontekstis kohustuslik. Samuti on autoril varasem üpriski korralik kogemus antud keelega, olles rakendanud seda mitmetest erinevates projektides. Valikut toetas veel mitmete erinevate kasulike pistikprogrammide olemasolu, nagu PyFlowchart (2.2.3). Nimetatud põhjuste tõttu ongi töö serverirakendus kirjutatud Pythonis.

2.3.2 Flask

Flask on populaarne veebiraamistik Pythonile, mis võimaldab kiiresti ja efektiivselt luua REST API lõpp-punkte. Flaski minimalistlik disain ning paindlik struktuur teeb selle sobivaks väikestele ja keskmise suurusega rakendustele. Flask toetab HTTP meetodeid ja JSON-formaadis andmete saatmist, vastuvõtmist [7].

Antud töös kasutati Flaski just selle lihtsuse ja kiiruse tõttu, sest see võimaldab luua lihtsat REST API lõpp-punkti, mille kaudu saavad kasutajaliides ning serverirakendus omavahel andmeid vahetada.

2.3.3 React ja Typescript

React on populaarne JavaScripti teek, mille tugevusteks on komponentidel põhinev arhitektuur ning võimekus luua interaktiivseid rakendusi. React võimaldab lihtsat komponentide haldamist ja seisundite (state) salvestamist ning juhtimist [8]. Selle võimsuse ja skaleeritavuse tõttu kasutavad Reacti ka paljud populaarsed ja massiivsed rakendused, nagu

Facebook, Instagram, Atlassian ja Netflix [9].

TypeScript on JavaScripti laiendus, mis võimaldab staatilist tüüpimist, pakkudes arendajatele vigade tuvastamist enne koodi kompileerimist. TypeScripti tugevuseks on koodi loetavuse ja hooldatavuse paranemine ning vigade arvu vähenemine, kuna tüübid kontrollitakse kompileerimise käigus [10].

Antud töös valiti kasutajaliidese arendamiseks välja just Reacti ja Typescripti kombinatsioon, sest autor on varasemalt neid kasutanud erinevate kasutajaliideste arendusprotsessides. Reacti kasuks on ka *TalTech Digital Styleguide* (TalTechi digitaalne stiiliraamat) [11] olemasolu, milles on valmis tehtud React komponendid ja mida saab koheselt kasutada ilma suurema lisatööta. Kõik sealsed komponendid on kooskõlas TalTechi ametliku stiilijuhendiga, mis on antud töö puhul eriti oluline, kuna lõpptulemust kasutavad just TalTechi õppejõud.

2.3.4 Vite

Vite on kaasaegne veebirakenduste arendus- ja ehitustööriist, mille eesmärgiks on kiirendada rakenduste lokaalset arendust, pakkudes märgatavalt kiiremat ja efektiivsemat kasutajakogemust võrreldes traditsiooniliste tööriistadega nagu Webpack. Vite pakub kiiret kohalikku arendusserverit, mis töötab ESM põhimõttel, laadides mooduleid brauseris otse, ilma et kogu rakendust peaks eelnevalt komplekteerima. See võimaldab teha väikseid uuendusi rakenduse koodis, pakkudes samaaegselt reaalajas muudatuste kajastust. Lisaks pakub Vite sisseehitatud tuge populaarsetele JavaScripti teekidele nagu Vue, Svelte ja React [12]. Viimane neist on kasutuses ka selles projektis.

Selles töös kasutatai Vite'i just selle kiiruse, reaalajas muudatuste kajastamise ja kohaliku arendusserveri olemasolu tõttu. Samuti on töö autor varasemalt kasutanud Vite'i erinevate projekti arendusprotsessides ning on kindel selle kasutajakogemuses ja võimekuses.

2.3.5 Docker

Docker on konteineripõhine platvorm, mis võimaldab arendajatel "ehitada, pakendada ja jooksutada rakendusi igas keskkonnas". See teeb seda, pakkides rakendused isoleeritud konteineritesse koos nende sõltuvustega. Selline lahendus tähendab, et rakendus töötab

samamoodi nii arendaja arvutis kui ka kliendi arvutis. Docker on nüüd juba pikka aega olnud standardne lahendus tarkvaraarenduse kõige keerukamale ja resurssinõudvamale osale, milleks on rakenduse publitseerimine (deployment). Varasemalt oli vaja mitmeid erinevaid spetsialiste, kes vastutasid publitseerimise voo töö korrektse toimimise eest [13].

Käesolevas töös kasutati Dockerit selleks, et serverirakendus ja kasutajaliides oleksid konteinerdatud eraldi konteineritesse. See võimaldas kasutada Docker Compose'i, millega saab kogu süsteemi lihtsalt tööle panna ühe käsuga. Selline lähenemine lihtsustas oluliselt rakenduse paigaldamist ja käivitamist TalTechi serveris, vältides käsitsi konfiguratsiooni seadmisega seotud ebamugavusi.

2.3.6 Caddy

Caddy on avatud lähtekoodiga veebiserver, mille peamiseks tugevuseks on automaatne HTTPS tugi tänu sisseehitatud *Let's Encrypt* integratsioonile. Lisaks sellele on Caddy's võimalik seadistada pöördproksi ja selle kaudu staatilist sisu serveerida, teha kogu konfiguratsiooni failipõhilselt või isegi API kaudu [14].

Selles töös kasutati Caddy veebiserverit, et TalTechi serveris olevad konteinerid oleksid kergesti kättesaadavad. Caddy võimaldas lihtsalt üles seada pöördproksi, mis suunas päringud vastavalt rakenduse kasutajaliidese- või serveripoolsele API lõpp-punktile.

2.3.7 LLM API integratsioon

OpenAI API on suure keelemudeli API, mis on loodud OpenAI poolt ning mis võimaldab arendajatel kasutada GPT-põhiseid mudeleid, et luua loomulikus ja inimlikult loetavas keeles väljundit vastavalt erinevatele sisenditele. Selle API puhul on võimalik anda mudelile ette soovitud väljundiformaat, mis keeles väljund peaks olema, väljundi kirjutamistill ning struktuur [15]. Nende alusel saab genereeritud vastuse, mida saab otse kasutada kasutajaliideses visuaalse tagasisidena.

Gemini API on suure keelemudeli API, mille on loonud Google ning mis võimaldab samuti struktureeritud päringute kaudu genereerida loomulikus ja inimlikult loetavas keeles vastuseid. Gemini eristub mitmes mõttes OpenAI lahendusest, näiteks oma toe poolest multimodaalsusele ja mitmekeelsele sisendile [16]. Käesolevas töös katsetati Gemini

API kasutamist sarnase eesmärgiga nagu OpenAI puhul: genereerida koodi võrdlusest inimloetav kokkuvõte.

Antud töös katsetati ja integreeriti suurte keelemudelite ehk LLM-ide API-sid. Mõlema mudeli eesmärk oli võtta serverirakenduse poolt loodud AST-võrdluse tulemused ning genereerida nende põhjal inimkeelne, loetav ning struktureeritud kokkuvõte.

Selliste LLM-i põhiste tööriistade kaasamine on oluline tulevikusuund, mis võimaldab automatiseerida koodi analüüsi ja tagasisidestamise protsessi, muutes selle märgatavalt kiiremaks ja kättesaadavamaks nii õppejõule kui ka tudengile.

2.3.8 Kasutajaliidese pistikprogrammid

diff2html on JavaScripti teek, mis võimaldab genereerida GitHubi laadse visuaalse võrdlusvaate tekstivõrdluste põhjal [17]. Seda kasutati rakenduses selleks, et serveri poolt koostatud *diff*-failidest (võrdluse tulemustest) kuvada kasutajale selge ja struktureeritud erinevuste vaade, mis on põhimõtteliselt sarnane sellega, mida GitHubis näeb koodimuudatuste puhul.

flowchart.js on JavaScripti teek, mis võimaldab genereerida SVG formaadis vooskeeme vastavalt etteantud DSL formaadis sisendile [18]. Seda kasutati käesolevas töös, et visuaalselt välja tuua vooskeeme, mis loodi serverirakenduses asuva PyFlowchart (2.2.3) tööriista abil. Serverirakendusest tulenev DSL string teisendatakse kasutajaliideses selgeks ja intuitiivseks vooskeemiks, mis võimaldab õppejõududel paremini mõista tudengi koodi struktuuri.

Mõlemad pistikprogrammid aitasid muuta rakenduse kasutajaliidese visuaalselt informatiivsemaks ja lihtsamini mõistetavaks.

3 Arhitektuur

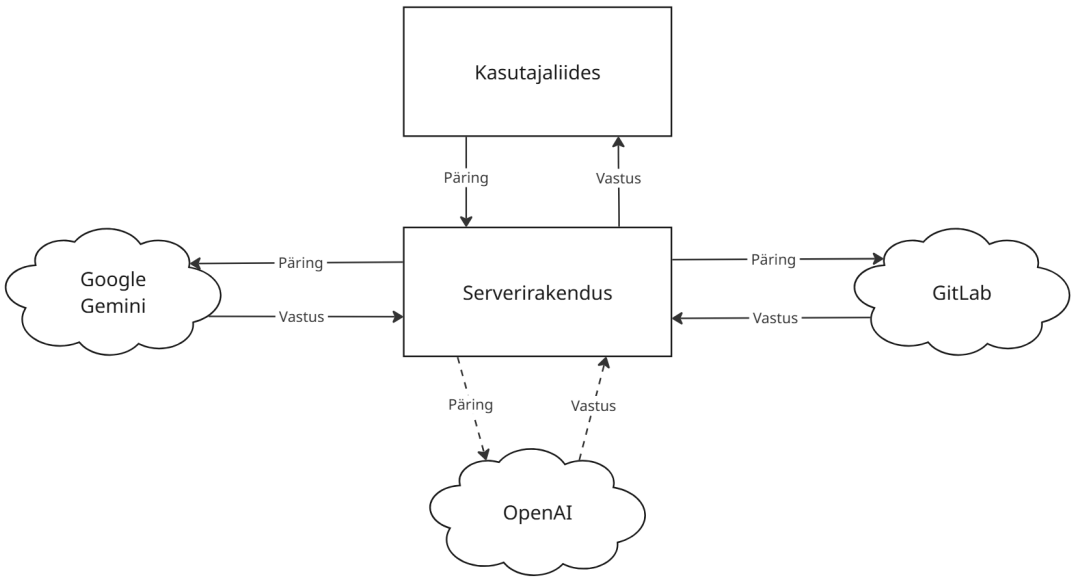
Selles peatükis kirjeldatakse detailsemalt lahti rakenduse arhitektuuri. Järgnevalt tuuakse välja realiseeritud arhitektuur ning põhjused, miks just sellised valikud tehti.

3.1 Arhitektuuri ülevaade

Realiseeritud lahenduse arhitektuur koosneb kasutajaliidesest, serverirakendusest, kahest välisest API-st ning ühest süsteemisisesest API-st. Sisemise API ülesanne on vahendada informatsiooni kasutajaliidese ja serverirakendusel vahel. Serverirakendusega on ühendatud väline Gitlab API ning vastavalt päringu edukusele OpenAI API või Google Gemini API. Kasutuse korral näeb süsteemi voog välja järgnevalt:

1. **Kasutajaliides:** Kasutaja sisestab sisendvormi info ning vajutab "*Compare*"nuppu, seejärel saadab kasutajaliides sisestatud infoga päringu serverirakenduse API lõpp-punkti.
2. **Serverirakendus:** Saab kätte päringu, saadab omakorda päringu GitLab API-le, et kätte saada tudengi-, õppejõud kood.
3. **Serverirakendus:** Teeb võrdlused ning kokkuvõtte jaoks saadab kõigepealt päringu Google Gemini API-le. Juhul kui see päring peaks ebaõnnestuma, esitatakse päring ka OpenAI API-le.
4. **Serverirakendus:** Edukate päringute juhul valmistab rakendus ette vastuse ning saadab selle sisemise API kaudu tagasi kasutajaliidesesse.
5. **Kasutajaliides:** Saab serverirakenduselt vastuse kätte ning kuvab kasutajale tulemused.

Arhitektuuri on võimalik näha visuaalselt jooniselt 1.



Joonis 1. Rakenduse arhitektuur ning infovahetuse voog.

3.2 Arhitektuurilised valikud

Antud lõputöö on pilootprojekt ehk varem sellist rakendust pole tehtud. Seetõttu olid arhitektuuri planeerimisel eriti määravateks faktoriteks süsteemi modulaarsus ja töökindlus.

3.2.1 Modulaarsus

Modulaarne arhitektuur võimaldab iga komponendi (kasutajaliides, sisemine API, serverirakendus, välised LLM mudelid, GitLab API) järk-järgult arendamist ja testimist. Kui näiteks tulevikus on vaja tudengi- ja õppejõu andmeid kätte saada mujalt või lisada kolmas LLM mudel, ei ole vaja kogu süsteemi ümber kirjutada, piisab vaid vastava mooduli muutmisest või juurde lisamisest. Selle tõttu on projekti kood selgem ning on väiksem oht, et ühe teenuse muutmine võib mõnele teisele teenusele konflikte tekitada. Ühtlasi on selline arhitektuur kasulik ka järgnevatele arendajatele, kes potensiaalselt projekti edasiarenduse või laiendamisega tegelevad.

3.2.2 Töökindlus

Rakenduse LLM päringu taandeoleku strateegia (esmalt päring Google Gemini API-le, ebaõnnestumise korral OpenAI API-le) tagab, et kui midagi peaks juhtuma ühe LLM

mudeli päringuga, saadetakse veel üks päring teisele LLM-i mudelile. Kõik selleks, et rakenduse töövoogu hoida võimalikult töökindlalt, kuna eelnevalt mainitud päringud võivad erinevatel põhjustel ebaõnnestuda.

4 Arendusprotsess

Selles peatükis tuuakse detailselt välja rakenduse arendusprotsess, mille käigus arendati välja serverirakendus ning kasutajaliides. Rõhk on selles peatükis just arendusprotsessi käigus toimunud muudatustel ning vigadel, seejärel tuuakse välja ka lõplik lahendus.

4.1 Serverirakendus

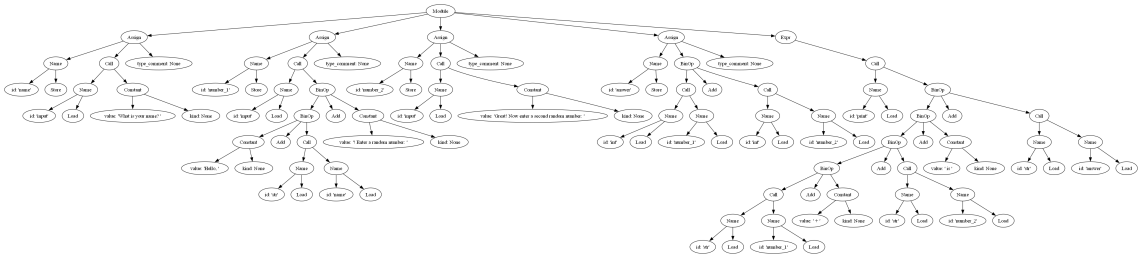
Serverirakenduse ülesanne antud töö kontekstis, vastavalt funktsionaalsetele nõuetele, on võimaldada koodi toomise funktsionaalsust ja tekitada API lõpp-punkt, mille abil saab kasutajaliides saata päringuid serverirakendusse. Samuti on serverirakenduses ka lõviosa rakenduse võrdlusprogrammi loogikast.

Kogu rakenduse arendusprotsess algaski just võrdlusprogrammi loomise loogikaga, kuna tegemist on lõputöö põhilise komponendiga ning kogu ülejäänud rakendus kujuneb just selle osa ümber. Pärast algsete versioonide loomist tekitati juurde ka API lõpp-punkt ning rakendati erinevatel viisidel koodiplokkide toomise funktsionaalsust.

4.1.1 Võrdlusprogrammi esimene versioon

Algne idee oli võtta kasutada ainult tudengi koodi ning vastavalt AST-le, leida erinevaid vigu, mis võivad koodis esineda. See protsess nägi siis välja nii, et programm saab sisendiks tudengi koodiploki mitmerealise sõne kujul, seejärel lastakse see läbi AST parserist, mis viib koodiploki puu andmestruktuuri kujule, millelt on eemaldatud kõik ebavajalikud ning ebaolulised read nagu kommentaarid, *docstringid* jms. Peale seda toimuks algoritmile tüüpvigade otsing, näiteks *return* klausel on väljaspool funktsiooni. Viimaseks viiakse see väljund puu kujul graafilisele kujule, mida oleks võimalik kasutajale kuvada. Selline algne lahendus sai valmis ehitatud, kuid sellega esines mitmeid erinevaid probleeme. Esiteks oli kohe aru saada, et tüüpvigadest ainult lähtuda sellise rakenduse puhul on praktiliselt võimatu, kuna programmeerimise algkursust läbivad tudengid on eelkõige algajad programmeerijad ning seetõttu võib esineda väga erinevaid vigu, mida lihtsalt ei

ole võimalik ette ennustada. Teine suur probleem oli selliste vigade kuvamine kasutajale. Algselt kasutusse võetud Graphviz [19] visualiseerimistarkvara oli liiga detailne ning ei võimaldanud hästi kasutajal mõista, mis tegelikult koodis valesti on või üldse aru saada koodi voost. Seda on võimalik näha joonisel 2.



Joonis 2. Graphviz abil loodud visuaalne kuvand võrdlustulemustest.

4.1.2 Võrdlusprogrammi teine versioon

Analüüsides algse lahenduse miinuseid jõuti kooskõlas juhendajaga ideeni, mille tõttu antud programm kannabki nime võrdlusprogramm. Nimelt on programmeerimise algkursusel kõik ülesanded enne OOP ülesandeid kindla struktuuriga, kus on etteantud mallid koos ettevalmistatud funktsioonidega, mida tudengid ise täitma peavad. Samuti on abiõppejõudude poolt valmis tehtud nendele ülesannetele näidislahendused, mis järgivad sama struktuuri nagu eelmainitud tudengite lahendatavad ülesande mallid. Seega leiti, et võiks võrrelda tudengite loodavat lahendust abiõppejõudude poolt valmisloodud lahendusega. Selline lähenemine avas võimalused teha koodi analüüsimist kahe koodi võrdlustulemuste põhjal.

Uue lahenduse jaoks läks plaani juurde veel ka võrdlusprogrammi modulaarseks muutmine, et oleks võimalikult lihtne lisada või eemaldada programmi juurde võrdlusstrateegiaid. Kuna üks kokkulepitud funktsionaalsetest nõuetest on visuaalse võrdlustulemuse kuvamine, oli vaja tekitada erinevaid võrdlusstrateegiaid, mis annavad tulemusena erinevad võrdlustulemused. Seetõttu võetigi eesmärgiks teha järgnevad võrdlusstrateegiad:

- **Range võrdlus** - Kontrollib, kas tudengi kirjutatud funktsioonid vastavad täpselt näidislahenduse funktsioonidele. See tähendab, et võrreldakse kogu funktsiooni sisu ridade kaupa, ning iga väiksemgi erinevus tuvastatakse. See strateegia sobib hästi olukordades, kus on vaja leida rangelt üles kõik erinevused kahe koodi vahel. Üldiselt peaks tudengi koodi analüüsi tehes toetama sellele võrdlusstrateegiale viimasena.
- **Semantiline võrdlus** - Püüab võrrelda koodide tähendust ehk semantikat, mitte

niivõrd süntaktilist kujundust. Selle võrdlusstrateegia eesmärk on mõista, kas tudengi kood täidab sama funktsiooni kui õppejõu kood, isegi kui kirjutamisviis või koodistruktuur on mõnevõrra erinev. Näiteks, kui kasutatakse erinevaid muutujanimesid või teistsugust süntaksit, kuid loogika ja tulemus on samad, siis peetakse seda sobivaks lahenduseks.

- **Struktuuriline võrdlus** - Analüüsib koodi ülesehitust ehk selle struktuuri. See tähendab näiteks kontrolli, kas funktsioonid ja plokid esinevad samas järjekorras või kas kasutatakse sarnaseid konstruktsioone, nagu *if*-tingimused ja tsüklid. Seda tüüpi võrdlus aitab hinnata, kui sarnane on tudengi lahendus õppejõu poolt koostatud lahendusele ülesandestruktuuri ja loogilise ülesehituse poolest.

Siinkohal on oluline mainida, et siiamaani loodav lahendus töötas just mõlema koodi vaheliste funktsioonide võrdlusega. See tähendab, et mõlemad koodiplokid loeti üle, otsiti üles kõik koodiplokis defineeritud funktsioonid, salvestati need sõnastiku andmestruktuuri koos kogu antud funktsiooni koodi ja funktsiooni nimega, tehti sama teises koodiplokis ning lõpuks võrreldi vastavate funktsioonide koode omavahel.

4.1.3 Võrdlusprogrammi lõplik versioon

Testides võrdlusprogrammi erinevate tudengikoodidega avastati mõned puudujäägid. Kuna kõik võrdlused toimusid funktsioonide vaheliselt, siis jäid kahe silma vahele esimeste nädalate ülesanded, kus toimus koodi kirjutamine vabalt ning ei olnud defineeritud funktsioone. Seetõttu oli vaja teha eraldi võrdlusloogika iga võrdlusstrateegia jaoks ka sellisel puhul, kui koodis ei esinenud funktsioone, vaid koodifail ise oligi kogu koodi loogika ühtse koodiplokina.

Samuti oli probleeme semantilise võrdluse rakendamisega, sest semantilise võrdluse strateegia eesmärk oli anda aluspind, mille abil koostatakse inimkeeles ja loogiline tekstipõhine kokkuvõte. Antud kokkuvõte oleks kohe olemas kasutajale lugemiseks ning võiks ideaalses olukorras asendada vajaduse süvitsi erinevate strateegiate tulemusi uurida, kuna annab kohe kiire ja detailse ülevaate, mis tudengi koodis on valesti. Küll aga sellist lahendust katsetades oli kiirelt aru saada, et algoritmiliselt korralikku kokkuvõtet luua on väga raske, sest see eeldaks jällegi eelnevalt tüüpvigade kirja panemist ning siis vastavalt leitud vigadele nende lahterdamist antud tüüpvigadeks, seetõttu võeti vastu otsus teha

kokkuvõte teistsuguse lahendusega, mis on detailsemalt välja toodud: 4.1.4. Seetõttu asendati semantiline võrdlus GitHubi stiilis *diff*-vaatega, mis annab kasutajale võimaluse näha mõlemat koodi algsel kujul ilma erinevate abstraktsioonideta.

4.1.4 Kokkuvõtte genereerimine

Algselt semantilise võrdluse baasil genereeritava kokkuvõtte tegemine osutus väga keeruliseks. Seetõttu võeti ette teistsugune lähenemine, otsustati kasutada LLM-i abi, et genereerida loetavat ning lihtsas tekstis kirjutatud kokkuvõtet.

Kokkuvõtte genereerimiseks anti LLM-ile sisendiks nii tudengi- kui ka õppejõu koodiplokid. Sisendile lisati juurde veel nõuded koos kindla struktuuri ja juhistega, et tagada vastuse kindel tekstiline struktuur igal API päringul. Algselt võeti kasutusse OpenAI mudelid, nagu *o3-mini*, *o4-mini*, *gpt-4.5-nano* ja *o4*. Mudeleid testiti erinevate testülesannete peal, kus olid meelega tekitatud kindlad vead, mis olid eelnevalt teada ning siis jälgiti kui hästi mudel need üles leiab. Mudeleid võrreldi veel ka nende teksti loetavuse, päringu maksumuse ja kiiruse raames. Viimane neist oli eriti oluline, sest kuigi näiteks *o4* mudel andis väga täpseid vastuseid ning tegi analüüsi väga hästi, oli too mudel üsna aeglane. Nimelt võttis üks päring vahepeal aega üle ühe minuti, mis on selle rakenduse kontekstis üsna pikk aeg. Prooviti ka *gpt-4.5-nano* mudelit, mis oli küll väga kiire ja odav, kuid täpsus andis soovida ning tihtipeale genereeris see ebatäpseid analüüsi tulemusi. Lõpuks leiti, et kuldne kesktee on *o4-mini* mudel, mille ühe päringu maksumus oli keskmiselt 0,01 eurot, päringu tegemine võttis keskmiselt 20-30 sekundit ning analüüsi tulemused olid sobivad.

Hiljem tehti analüüs, kus uuriti ka teiste ettevõttete mudeleid nagu Google Gemini seeria mudelid. Sealtkaudu leiti selline mudel nagu *Gemini-2.5-flash*, mis võrreldes *o4-mini* mudeliga oli veidi kiirem, nimelt keskmine päring võttis aega 15-20 sekundit, analüüsi tulemused olid *o4-mini* mudeliga võrdväärsed ning päringu tegemine on täiesti tasuta, kuid on peal piirang, et minutis ei saa teha rohkem kui 10 päringut. Antud töö kontekstis selline päringute piirang oli sobilik. Küll aga oli sellel mudelil probleem töökindlusega, sest vahepeal ei läinud päringud läbi ning seetõttu pidi kutsuma mitu korda sama päringut enne kui tulemus kätte saadi. See lahendati lõpuks niimoodi, et loodi süsteem, kus kõigepealt saadetakse päring *Gemini-2.5-flash* mudelile ning peale timeouti ületamist või päringu katkemist saadetakse päring varuvariandiks olevale *o4-mini* mudelile. Selline lahendus

lisas LLM-i integratsioonil põhineva kokkuvõtte genereerimisele töökindlust, vähendas kulu ja tegi päringuid kiiremaks.

4.1.5 Koodiplokkide toomine

Koodiplokkide toomise funktsionaalsus on üks kõige olulisematest osadest antud rakenduse tööst. See peab toimuma võimalikult valutult, lihtsalt ja kiirelt, et kasutajate aega säästa.

Algselt mindi kõige lihtsama variandi kaudu ning võrdlusprogramm ootas sisendiks kahte koodiplokki otse kasutajaliidesest. Selline lahendus aitas algselt testida ülejäänud rakenduse loogikat olles ise funktsionaalsuse poolest üsna lihtne ning primitiivne lahendus.

Arendusprotsessi kulgedes ilmnes tõsiasi, et selline lahendus on liiga aeglane ning nõuab kasutajal ise otsida ning rakendusse saata nii tudengi kood, kui ka õppejõu näidislahenduse kood. Seetõttu arendati välja uus lahendus, mis kasutab tänu Gitlab API-le otsib ise mõlemad koodiplokid automaatselt üles, peale seda kui kasutaja on päringu esitanud. Päringusse peab kasutaja kaasama ainult järgneva info: tudengi UNI-ID, ülesandenumber, mida analüüsitakse ja kursuse toimumise aasta. Neid andmeid kasutatakse, et saada ligipääs nii tudengi kui ka õppejõu Gitlabi repositooriumitele, kuhu on salvestatud antud ülesande koodiplokid. See lahendus töötab just nimelt kindla struktuuriga repositooriumite tõttu, kui nii poleks, oleks selline lahendus võimatu.

4.1.6 API lõpp-punkt

Serverirakendusel arendusprotsessi käigus saadi algselt sisendiks koodiplokid ja muu info lokaalselt otse testandmetest kätte. Peale kasutajaliidese arenduse algust tekkis vajadus hakata sisendinfot saama päringute kaudu otse kasutajaliidesest. Seetõttu tuli luua serverirakendusse API lõpp-punkt, mille kaudu rakenduse kaks poolt üksteisega suhelda saaksid. Loodud sai siis `/api/diff` lõpp-punkt, mis võtab vastu POST päringuid. Sisendiks võtab lõpp-punkt vastu JSON kujul tudengi UNI-ID, ülesande number, kursuse toimumise aasta ning kõik filtrid.

Palju rõhku pandi siin staadiumis ka võimalike tõrgete püüdmisele, kuna päringuid ning muid operatsioone toimus ühe voo jooksul palju. Seetõttu sai suure uuenduse tõrgete süsteem, mis vastavalt tõrke olulisusele pani kogu protsessi seisma ning edastas veateate

kasutajaliidesele edasi, et seda kasutajal kuvada. Näiteks toimub see siis, kui kummagi repositooriumi koodi ei ole mingil põhjusel võimalik kätte saada. Vastupidiselt on võimalus ka süsteemi töö jätkumisest isegi kui esineb tõrge, näiteks on selline töö käik kui esimene LLM-i mudeli päring peaks tõrke tagastama, mille järel alustab süsteem uut päringut teise mudeliga.

4.2 Kasutajaliides

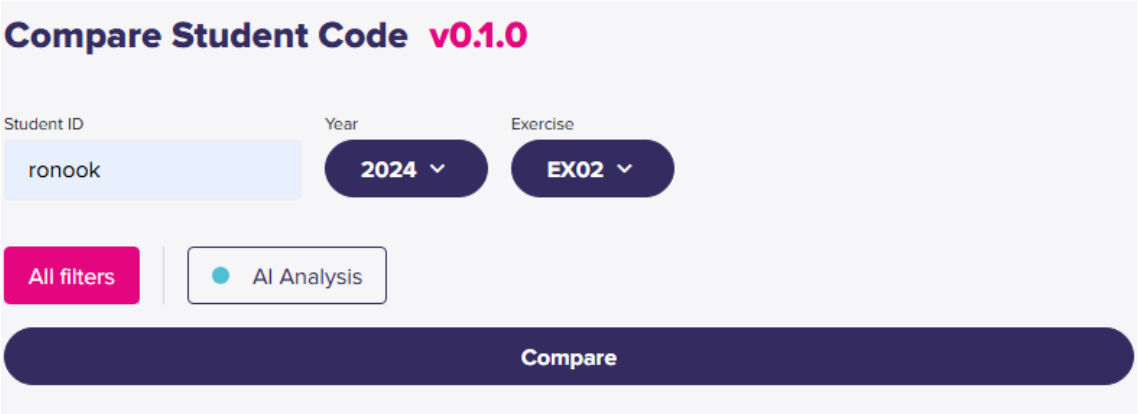
Kasutajaliidese ülesanne on pakkuda rakenduse kasutajale mugavat liidest, milles saab päringute kaudu küsida serverirakenduselt analüüsi tudengi koodi jaoks. Funktsionaalsete nõuete kohaselt peab kasutajaliides sisaldama koodifailide leidmiseks sisendi vormi, võrdlusprogrammi kokkuvõtet ning võrdlustulemusi. Kasutajaliidese arendusprotsess toimus üsna käsikäes serverirakenduse arendusprotsessiga, kuigi algas veidi hiljem, sest esmalt toimus kogu rakenduse arendamine ainult serverirakenduse pool. Kasutajaliideses pandi ühtlasi maksimaalselt rõhku sellele, et kõik kasutatavad komponendid oleksid TalTechi digitaalsest stiiliraamatust. Selle kaudu tagati kasutajatele tuttav TalTechi stiilis keskkond, kus komponendid on tuttavad ja intuitiivsed. Arendustöö käigus püüti meeles pidada ka rakenduse mugavat kasutajakogemust telefonis. Üldjoontes oli sellega korras, sest TalTechi digitaalse stiiliraamatu komponendid hoolitsesid selle eest ise, kuid osad manuaalsed parandused pidi tegema.

4.2.1 Koodifailide leidmise sisendivorm

Rakenduse arendusprotsessis pandi algselt rohkem rõhku sellele, et uusi funktsionaalsusi oleks võimalikult kerge testida. Seetõttu arendati välja kasutajaliideses koodifailide jaoks sisendvormiks kaks tekstiakent, mis võtsid vastu tudengi ja õppejõu koodiplokke ning mis seejärel saadeti serverirakendusse. See aga ei olnud kaugeltki optimaalne lahendus ning ei saanud jääda lõplikuks variandiks.

Peale üleminekut Gitlab API peale, et koodifaile automaatselt otse repositooriumitest tõmmata, muudeti sisendivormi, et oleks võimalik saata serverirakenduse lõpp-punkti vajaminevaid andmeid. Selle jaoks tehti vajaminevad vormi lahtrid, kuhu sai kasutaja sisestada vajaminevat informatsiooni. Vorm tehti kasutaja jaoks ka lollikindlaks ehk päring ei lähe kunagi läbi valede andmetega ning tühjade lahtritega. Kasutaja saab näiteks

ülesannete lahtrist valida ainult neid ülesandeid, mida rakendus ise toetab ning mille peal on seda testitud. Joonisel 3 on näha täidetud andmetega sisendvormi.

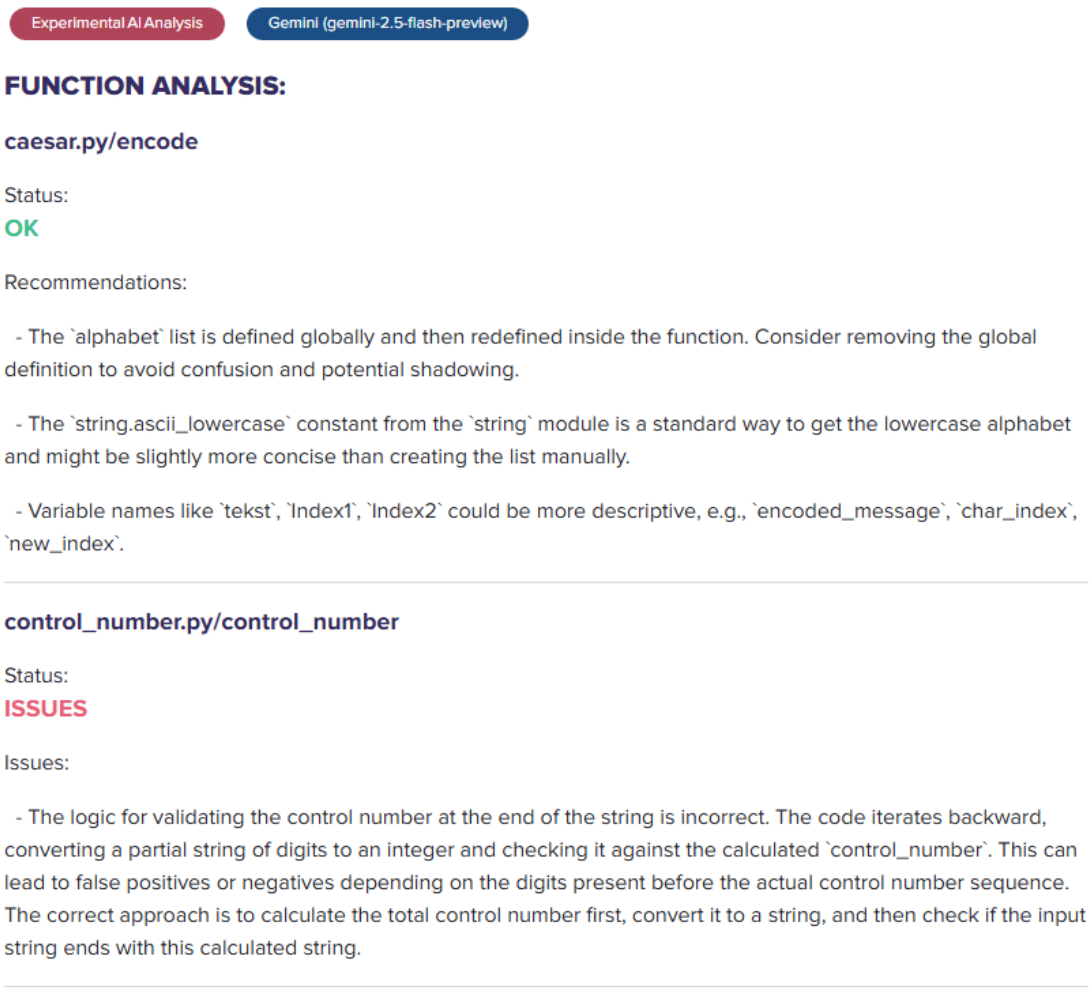


Joonis 3. Sisendvormi, kuhu on info korrektselt sisestatud.

4.2.2 Võrdlusprogrammi kokkuvõte

Võrdlusprogrammi kokkuvõte kuvamine kasutajaliideses läks reaalselt arendusse alles peale serverirakenduse LLM-i API integratsiooni realiseerimist. Seetõttu hakkas selle arendusprotsess kohe pihta sellega, et võeti vastu serverirakendusest kindla struktuuri tekstiplokk, mis sisaldas LLM-i poolt genereeritud analüüsi tulemust.

Põhikeerukus selle kasutajaliidese sektsiooni arendusprotsessis oli kasutajakogemuse võimalikult mugavaks tegemine. Kogu kuvatud kokkuvõtte mõte oli, et see hakkaks kasutajale kohe silma ning kasutaja saab kiirelt ja mugavalt selle info sealt välja loetud. Seetõttu sai arendusprotsessi käigus loodud funktsionaalsus, mis võttis päringu tulemuseks saadud tekstiploki ning hakkas seda rea haaval üle käima. Selle käigus otsiti üles kõik olulisemad read, nagu funktsiooni nimed, probleemide olemasolu staatus ning viidi need tavalise teksti kujult näiteks päise komponendiks üle ja värviti teist värvi, et neid rohkem esile tuua. LLM-i analüüsi on võimalik näha joonisel 4.



Joonis 4. LLM-i analüüsi funktsioonipõhised kokkuvõtted.

Hiljem lisati sinna sektsiooni ka konteksti sildid, mis andsid kasutajale aimu, millise LLM mudeli tekstiga tegu on.

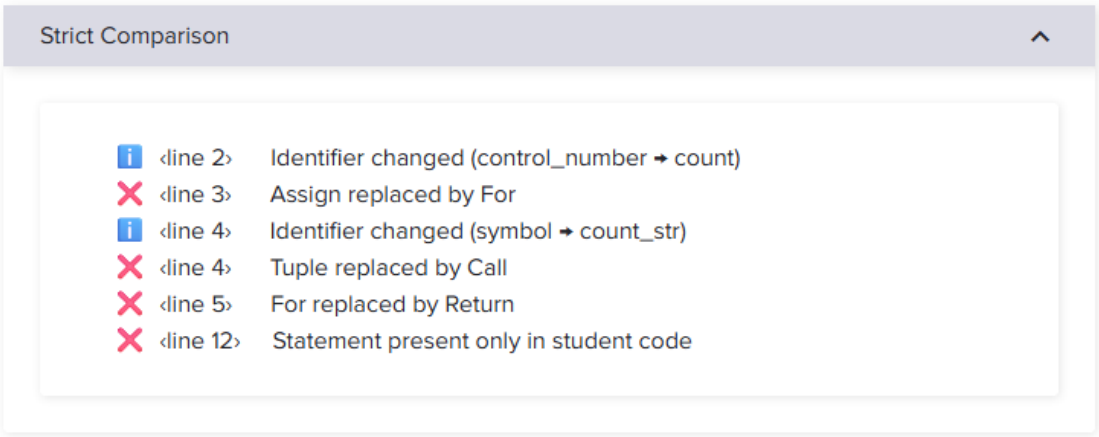
4.2.3 Võrdlustulemused

Kui liikuda kasutajaliideses peale päringu tulemuse saamist ülevalt alla, siis viimasena, kõige all, kuvatakse kasutajale võrdlustulemusi. Nende eesmärk on pakkuda kasutajale võimalust ise süvitsi üle vaadata kõik võrdlustulemused, juhul kui LLM-i poolt genereeritav kokkuvõte ei olnud piisav, et õpilase vigu välja tuua.

Võrdlustulemuste kuvamisel peeti oluliseks just struktuuri, kuidas on tulemused laiali jaotatud, et kasutajal oleks võimalikult mugav neid üles leida ning lugeda. Üks eesmärkidest oli ka info ülekülluse vältimine. Seetõttu otsustati kasutada TalTechi digitaalsest stiiliraamatust

pärit Accordion komponenti, mis võimaldas kindla struktuuriga võrdlustulemusi kuvada. Jaotus oli käis kõigepealt koodifailide nimede järgi, seejärel juba funktsioonide nime järgi ning lõpuks veel võrdlusstrateegiate järgi. Antud süsteem toetab ülesandeid ka siis, kui funktsioone defineeritud ei ole. Sellisel juhul on funktsioonide nimede asemel lihtsalt kirjas *main* ning seal kuvatakse kogu koodifaili sisu.

Range võrdluse võrdlusstrateegia tulemuste kuvamine erilist takistust ei valmistanud, kuna need tulemused olid tekstipõhised, põhiprobleemid olid korrektse joondumise tekitamisega ja vastavalt erinevuse tasemest emotikonide kasutamisest lõppkuvandis. Seda kõike selleks, et kasutajal oleks võimalikult kerge mõista, mis tüüpi erinevustega tegu on. Range võrdluse tulemusi on võimalik näha joonisel 5



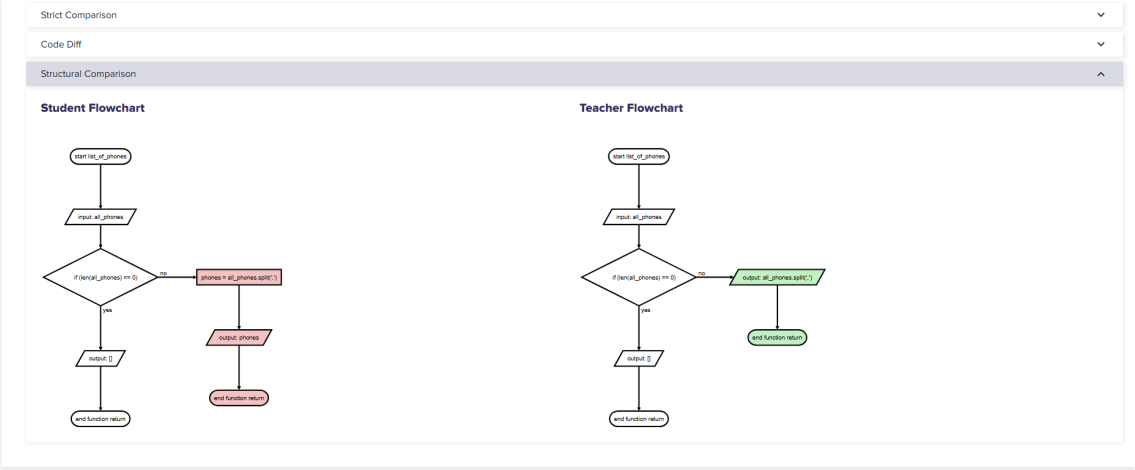
Joonis 5. Range võrdluse lõplik versioon.

diff-vaate tulemuste kuvamine oli väga kerge ning sai kiirelt implementeeritud. Lõviosa tööst tegi ära diff2html 2.3.8 JavaScripti teek, lisaks oli vaja ainult muuta veidi konfiguratsiooni. *diff*-vaadet on võimalik näha joonisel 6

Joonis 6. Diff vaate visuaalne kuvand.

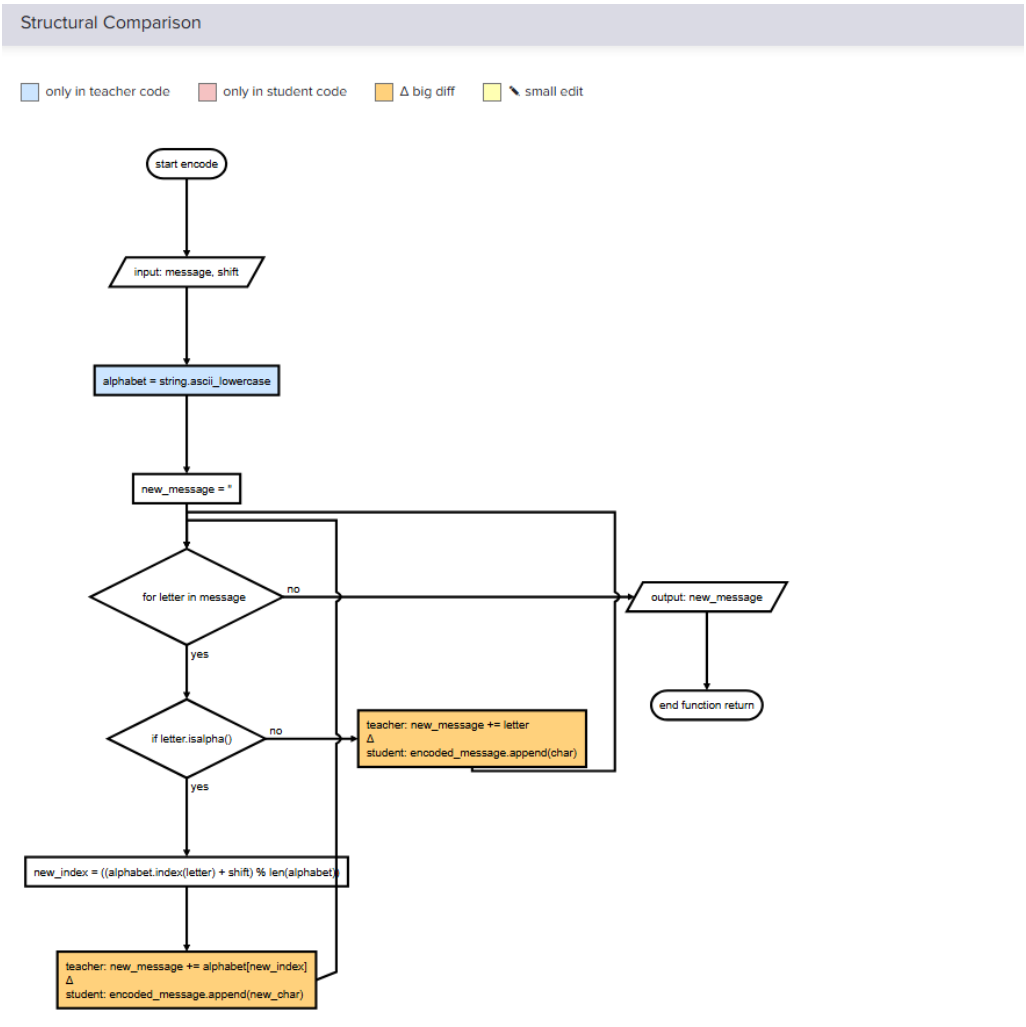
Kõige rohkem valu oli korrektselt struktuurilise võrdluse võrdlusstrateegia tulemuste kuvamisega. Üks suurimaid probleeme oli just nimelt sellega, kuna vooskeemid joonistatakse välja SVG formaadis ja see tähendab, et nad austavad endale antud ruumi. See aga oli probleemiks, sest SVG renderdamine toimus kohe peale päringu tulemuse jõudmist, ehk ajal, kui Accordion komponent oli kinnises olekus. Selle tõttu olid kõik vooskeemid tohutult väiksed ning neid ei olnud võimalik lugeda. Lahenduseks prooviti luua üks uus olek, mis jälgiks, kas antud komponent on avatud või mitte. Kahjuks see veel probleemi ei lahendanud, sest komponendi peale vajutades on veel animatsioon, mille käigus toimunud SVG formaadis vooskeemi renderdamine järgis veel enne avamist olnud ruumi piiranguid. Selle jaoks lisati väike 200 millisekundine hilistus, mis lubas SVG vooskeeme renderdada alles pärast seda, kui komponent oli jõudnud lõplikult avaneda.

Teine suur probleem esines sellega, et kuidas oleks kõige parem vooskeeme kuvada. Algne idee ning teostus oli luua kaks erinevat vooskeemi, millel on erinevused ära märgistatud erinevate värvidega. Üks puudest kujutakse vooskeemina tudengi koodi tööd ning teine puu õppejõu koodi tööd. Juba sellise lahenduse loomisega oli palju probleeme, sest puudus idee kuidas värvida ära just neid vooskeemi osasid, mis eristusid. Lahenduseks osutus selline loogika, kus võeti mõlema koodiplokki DSL stringid ette juba serverirakenduses ning otsiti üles kõik erinevused. Siis lisati vastavatele ridadele olekute sildid, mis kirjeldasid erinevuse tüüpi, nagu olemasolu ainult õppejõu koodi või tudengi koodis. Seejärel kasutajaliideses anti juurde oleku siltidele erinevad värvid, et oleks võimalik erinevusi eristada. Seda struktuurilise võrdluse lahendust on võimalik näha joonisel 7.



Joonis 7. Algne lahendus struktuuriliste erinevuste visuaalselt kuvamiseks.

Siiski pärast vaheesitluse tagasisidet oli aru saada, et antud lahenduse visuaalne kuju ei sobi väga hästi erinevuste leidmiseks. Samuti esines selles lahenduses palju vigu, kui erinevus oli esinenud koodi alguses, siis värvis rakendus ära ka kõik järgnevad harud. Seetõttu otsustati minna üle uuele lahendusele, kus on kuvatud vaid üks ühine vooskeem. Selle vooskeemi põhjaks on õppejõu kood, kuid sinna juurde lisatakse kõik erinevused, nagu mõne lisa- või puuduva haru olemasolu. Lisaks kui on mingis harus olukord, kus õppejõu ja tudengi kood on erinevad, siis tuuakse välja mõlemad versioonid, et erinevused oleksid hästi nähtavad. Eelnevalt mainitud olukorras hinnatakse automaatselt seda, kui suured need erinevused on just funktsionaalses mõttes, vastavalt sellele on antud haru värvitud kindlate värvidega. Lisaks on struktuurilise võrdluse lõplikus tulemuses olemas ka legend, mis selgitab kasutajale erinevuste tüüpide värviskeemi. Sellist lahendust on võimalik näha joonisel 8.



Joonis 8. Lõplik lahendus struktuuriliste erinevuste visuaalselt kuvamiseks.

Vigade tekkimist struktuurilises võrdluses vähendati algoritmiliste muudatustega. Eelnevalt proovis algoritm kokku sobitada järjest mõlema koodi ridu korda mööda. See aga ilmselgelt tõi sisse vigu, kui mõni rida ei olnud ühes koodis samal positsioonil, kus see oli teises koodis. Selle probleemi lahendamiseks annab uus algoritm igale reale oma tüübi põhise ID, seega võrreldakse ainult funktsionaalseid erinevuseid. Samuti kasutatakse Python pistikprogrammi difflib SequenceMatcher [20] funktsionaalsust ehk algoritm otsib üles kõik sarnasused koodis ning paigutab need kõrvuti isegi siis, kui need on failis eri kohtades. Nii leitakse erinevused üles ainult seal, kus need reaalselt ka eksisteerivad.

5 Tulemuste valideerimine

Siin peatükis on välja toodud, kuidas rakenduse arendusprotsessi tulemusi hinnati ning valideeriti.

5.1 Jooksev valideerimine

Rakendusele olid kehtestatud funktsionaalsed nõuded, mis küll arendusprotsessi käigus skoobi muutumise tõttu veidi muutusid, kuid jäid põhiliselt samaks, nagu algselt. Samuti on oluline mõista, et kuna antud lõputöö klient oli ka lõputöö juhendaja, siis toimus suhtlus nii-öelda kliendiga vahetult kogu arendusprotsessi käigus. Selline võimalus kliendiga pidevalt suhelda võimaldas kontrollida, kas rakenduse realiseerimine vastab kehtestatud nõuetele ning võimaldas teha kiireid muudatusi vastavalt saadud tagasisidele. Regulaarne arutelu ja tagasiside saamine aitas vältida suuri tõrkeid ning tagas, et arendatav rakendus oleks võimalikult kasulik algkursuse õppejõududele.

Lõputöö autor olnud abiõppejõud sarnasel programmeerimise kursusel, seega oli autoril varasem kogemus, millele tugineda rakenduse arendusprotsessi käigus.

5.2 Tagasiside abiõppejõududelt

Rakenduse kasulikkusest täielikku ülevaadet on dialoogi kaudu keeruline saada. Seega tehti tagasiside küsitlus just neile abiõppejõududele, kes on ise olnud enim seotud antud programmeerimiskursustega. Rakenduse kogu mõte on aidata eelnevalt mainitud abiõppejõude, mille tõttu oli nende poolt saadud tagasiside kõige olulisem.

Tagasiside küsitluse jaoks tehti abiõppejõududele valmis ka samm-sammuline õpetus, kuidas rakendust kasutada ning millele peaks tähelepanu pöörama. Kuna tagasiside kogumise hetkel programmeerimise algkursust ei toimunud, oleks olnud raske abiõppejõududel hinnata rakenduse kasulikkust, kui tudengitel olid sügisel toimunud kursuse ülesanded lahendatud. Seetõttu lõi lõputöö autor eraldi repositooriumi, mis jäljendas programmeerimise algkursust

läbiva tudengi repositooriumi. See tähendab, et seal olid olemas vaid kindlad välja valitud ülesanded, millest mõned olid vigadega ning automaattestid neid koode läbi ei lasknud. Nendest vigadest oli tehtud tagasisidestajatele ülevaade, et nad teaksid, mida jälgida rakendust testides.

Tagasiside vormis paluti abiõppejõududelt hinnata viie tärni skaalal rakenduse kiirust, kasulikkust ning disaini. Lisaks sellele oli palutud, et nad oma hinnangut põhjendaksid kirjalikult. Samuti küsiti üldist tagasisidet, mõtteid LLM analüüsi ja kokkuvõtte kohta ning funktsiooni põhiste võrdluste kohta, sealhulgas paluti neil täpsemalt välja tuua puudujäägid.

Tagasiside vormi sisu, kus on sammhaaval rakenduse kasutamise õpetus ning ülevaade vigadest, on nähtav lisas 2.

Tagasiside vormi täitsid kokku 9 erinevat abiõppejõudu. Neist 3 olid naissoost ning 6 meessoost. Esindatud olid nii bakalaureuse kui ka magistrikraadi tudengid.

5.2.1 Rakenduse kiirus

Rakenduse kiirus sai viie tärni süsteemis keskmiseks hinnanguks 4.44.

Kirjaliku tagasiside osas olid abiõppejõud rahul rakenduse kiirusega. Eraldi mainiti, et rakenduse tegelik kiirus oli kaks korda kiirem (10s), kui autor oli eelnevalt välja toonud vormi kirjelduses (20s). Vastanute vahel ei esinenud kordagi juhust, kus rakendus oleks oodatust aeglasem. Üks vastaja pakkus välja, et päringu vastuse ootamise ajal võiks rakendus kuvada tavalise laadimisanimatsiooni asemel seda, kui kaugel rakendus hetkel päringuga on. Näiteks *Analyzing file.py*, *Writing feedback*.

5.2.2 Rakenduse disain

Rakenduse disain sai viie tärni süsteemis keskmiseks hinnanguks 4.11.

Kirjaliku tagasiside osas andsid vastajad mõista, et tegemist on hea ning lihtsa disainiga, mis vastab TalTechi rakenduste stiilinõuetele. Eraldi toodi välja, et sisendvormi filtrid on veidi segased ning kohati ei saada aru, kas filter on sisse lülitatud või mitte. Samuti mainiti, et funktsioonide põhiliste võrdluste sektsioonis võiks saada näha erinevaid aknaid korraga, mitte ainult ühte, mis on hetkel tingitud Accordion komponendi funktsionaalsusest. Ühtlasi

arvati, et üldine kokkuvõte võiks olla esimesena nähtav ning LLM kokkuvõtte tulemustes võiks olla näha ka koodi konteksti, mille kohta tagasisidet anti.

5.2.3 Rakenduse kasulikkus

Rakenduse kasulikkus sai viie tärni süsteemis keskmiseks hinnanguks 4.44.

Kirjaliku tagasiside osas toodi üldiselt välja, et rakendus on kasulik ning leidsid, et peale tudengi koodist vigade otsimise, saab seda kasutada päris hästi ka lihtsalt soovituste andmiseks. Mõned vastajad välja, et näevad rakenduses suurt potensiaali ning esialgse lahendusena oleks see juba praegu kasulik. Eraldi toodi välja, et struktuurilise ülevaate saamine flowchart kujul on üsna kasulik, sest on kohe näha, kus on tekkinud viga ning seejärel saab LLM analüüsi abil ka kohe lahenduse leida. Anti ka soovitus, et rakendus võiks olla kätte saadav Charonis, mis on juba valmis ehitatud programmeerimise kursuste tugiplatvorm ning kus abiõppejõud tudengitele punkte sisestavad ülesannete eest.

5.2.4 Üldine hinnang LLM analüüsile

Kõik vastajad olid välja toonud, et LLM analüüsi osa on kindlasti kasulik. Eraldi toodi veel välja, et on tore, kuidas vahepeal annab LLM soovitusi, kuidas koodi efektiivsemaks teha, mitte ei otsi ainult vigu. Puudujäägina toodi välja, et ainult õppejõu lahenduse võrdlemisel ei pruugi analüüsist saada just kõige paremat pilti, soovitati, et võiks paluda LLM-il lahendada kõigepealt antud ülesanne ise ära ning siis võrrelda tudengi koodi ka enda pakutud lahendusega. Samuti toodi välja, et enamjaolt on LLM-i analüüs sarnane, kuid kohati võib LLM anda infot veidi rohkem või vähem kui mõni teine kord.

5.2.5 Üldine hinnang funktsiooni põhiste võrdlustele

Enamjaolt tõid vastajad välja, et need võrdlused on hea lisa, mida jälgida LLM-i analüüsi kõrvale. Eraldi toodi välja, et range võrdlus ning diff-vaade võiks kokku tõsta või võiks olla vähemalt võimalus vaadata neid samaaegselt. Struktuurilise võrdluse puhul toodi välja, et see on huvitav lähenemine ning on väga kasulik just koodi struktuuri mõistmiseks, aga vajab eelnevalt veidi ülesandesse süvenemist, kuna muidu on vooskeem natuke abstraktne. Lisaks toodi veel välja, et struktuurilise võrdluse vooskeeme saaks teoorias kasutada ka õppematerjalidena.

5.3 Edasiarendus

Tulenevalt abiõppejõudude tagasisidest ja ka arutelust juhendajaga tuuakse järgnevalt välja mõned rakenduse edasiarenduse ideed. Esiteks on võimalus rakendust integreerida abiõppejõudude tugiplatvormile Charon, kus võiks olla abiõppejõul võimalik kasutada rakendust sealsamas või viiakse ta lingi teel rakendusse, kus ei pea enam täitma sisendvormi infot ning talle kuvatakse automaatselt ette võrdlustulemused ning analüüs. Rakenduse protsesside kiirendamiseks võiks uurida ka alternatiivseid LLM mudeleid, mis teeksid võrdväärse analüüsi tulemuse, aga kiiremini kui hetkel kasutuses olevad mudelid.

Teiseks saab vaadata üle rakenduse disaini lähtuvalt abiõppejõudude tagasisidest. Näiteks soovitati ühendada funktsioonide põhised võrdlustulemused LLM analüüsiga, et abiõppejõududel oleks võimalik lugeda LLM-i koostatud kokkuvõtet nii, et kõrval oleks kontekstiks struktuurilise võrdluse tulemusel loodud vooskeem ning funktsiooni kood ise. See võimaldaks probleeme paremini visualiseerida.

Lisaks võiks rakendust laiendada teistele programmeerimiskeeltele, keeltele, nagu Java mis on kasutuses programmeerimise põhikursusel. Seoses sellise edasiarendusega oleks vaja üle vaadata ka praegused funktsioonide põhilised võrdlused, kuna hetkelahendus ei ole sobilik OOP tüüpi ülesannete analüüsimiseks.

6 Kokkuvõte

Selle bakalaureusetöö eesmärk oli luua rakendus TalTechi programmeerimise algkursuse abiõppejõududele, mis võimaldaks neil saada kiiresti tagasisidet tudengi koodi kohta, et leida vigu või anda soovitusi koodi kohta. Rakendust oli vaja, kuna varasemalt pidid abiõppejõud iseseisvalt otsima üles tudengi koodi. Seejärel pidi seda analüüsima, millel oli suur ajakulu ning seetõttu ei jõutud kõigi abivajajateni.

Arendustöö käigus sai valmis rakendus, mis täidab algselt seatud eesmärki ning nõudeid. Rakenduse töö põhineb tudengi ning õppejõu koodi võrdlusel. See koosneb kasutajaliidesest, mille kaudu saab kasutaja saata päringuid serverirakendusele, kus toimub kogu koodivõrdluse ning analüüsi loogika. Kasutaja peab sisestama rakendusse tudengi UNI-ID, ülesande numbri ning kursuse toimumise aastaarvu, seejärel kuvatakse talle koodivõrdluse tulemused ning LLM baasil tehtud koodianalüüsi kokkuvõte. Koodivõrdluste tulemustes on võimalik kasutajal koodide diff-vaadet, ranget ja struktuurilist võrdlust.

Lõputööna valminud rakendusel on tulevikus arendusvõimalusi. Näiteks toodi abiõppejõudude tagasisides välja viise kuidas rakendus saaks olla kasutajasõbralikum ning millist kasu see võiks veel tuua. Soovitati rakendus lisada otse Charonisse, et abiõppejõud saaksid kasutada seda samas kohas, kus ka ülesannetele punkte pannakse. Tagasisidestajad avaldasid veel ka soovi, et loodud rakenduses oleks võimalik võrrelda ja analüüsida ka teiste programmeerimiskeelte koode.

Kasutatud kirjandus

[1] Cedric Richter. *Code Diff*. [Viimati kasutatud: 07-05-2025]. URL: https://github.com/cedricrupb/code_diff.

[2] Tian Yang. *Mayat*. [Viimati kasutatud: 07-05-2025]. URL: https://github.com/AnubisLMS/Mayat.

[3] CDFMLR. *PyFlowchart*. [Viimati kasutatud: 07-05-2025]. URL: https://github.com/cdfmlr/pyflowchart.

[4] Python Software Foundation. *What is Python? Executive Summary*. [Viimati kasutatud: 07-05-2025]. URL: https://www.python.org/doc/essays/blurb/.

[5] Nazar Kvartalnyi. *Top 23 Applications Made with Python*. [Viimati kasutatud: 07-05-2025]. URL: https://inoxoft.com/blog/top-23-applications-made-with-python/.

[6] Gaurav Gupta. *Advantages and Disadvantages of Python – Make a Favorable Decision*. [Viimati kasutatud: 07-05-2025]. URL: https://squareboat.com/blog/advantages-and-disadvantages-of-python.

[7] Miguel Grinberg. *Flask Web Development*. Sebastopol, USA: O'Reilly Media, Inc., 2018.

[8] Cory Gackenheimer. *Introduction to React*. New York, USA: Apress, 2015.

[9] Angular Minds. *Top 13 Popular React Apps You Should Know About*. [Viimati kasutatud: 07-05-2025]. URL: https://medium.com/@angularminds/top-13-popular-react-apps-you-should-know-about-59eaf72bc176.

[10] Josh Goldberg. *Learning Typescript*. Sebastopol, USA: O'Reilly Media, Inc., 2022.

[11] TalTech. *TalTech Digital Styleguide*. [Viimati kasutatud: 04-06-2025]. URL: https://portal-dev.ttu.ee/styleguide/.

[12] Tuan Anh Nguyen. *A comparative analysis of Webpack and Vite as build tools for JavaScript*. [Viimati kasutatud: 07-05-2025]. URL: https://www.theseus.fi/handle/10024/877513.

[13] Aiden Sayers Ian Miell. *Docker in Practice*. New York, USA: Manning, 2019.

[14] Will Morris. *What Is Caddy Web Server?* [Viimati kasutatud: 08-05-2025]. URL: https://www.elegantthemes.com/blog/wordpress/what-is-caddy-web-server.

[15] Arunn Thevapalan. *A Beginner's Guide to The OpenAI API*. [Viimati kasutatud: 08-05-2025]. URL: https://www.datacamp.com/tutorial/guide-to-openai-api-on-tutorial-best-practices.

[16] Steven Ang Cheong Seng. *Guide: What is Google Gemini API and How to Use it?* [Viimati kasutatud: 08-05-2025]. URL: https://apidog.com/blog/google-gemini-api/.

[17] Rodrigo Fernandes. *diff2html*. [Viimati kasutatud: 08-05-2025]. URL: https://github.com/rtfpessoa/diff2html.

[18] Adriano Raiano. *flowchart.js*. [Viimati kasutatud: 08-05-2025]. URL: https://github.com/adrai/flowchart.js.

[19] Graphviz Development Team. *Graphviz*. [Viimati kasutatud: 04-06-2025]. URL: https://graphviz.org/.

[20] Python Software Foundation. *difflib*. [Viimati kasutatud: 04-06-2025]. URL: https://docs.python.org/3/library/difflib.html.

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

Mina, Robin Nook

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Tudengite koodi analüüsimine süntaksipuuga", 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.

06.06.2025

[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 – Tagasiside vormi sisu

Palun teie abi oma lõputöö rakenduse tagasisidestamisega. Rakenduse eesmärk on vähendada abiõppejõudude ajakulu tudengi koodi tagasisidestamisel Pythoni algkursusel.

Rakenduse saab kätte siit lingilt: https://cs.taltech.ee/services/codecomparison/

Rakenduse põhiflow on järgmine:

1. Sisestad tudengi UNI-ID(*testimiseks soovitan kasutada minu test-repot, seega pange "ronook"*)
2. Valid ülesande numbri(*nt EX02*)
3. Valid aasta(*testimiseks palun valige 2024 aasta, sest 2025 aasta kursuse reposid veel loodud ei ole*)
4. Valid filtrites, kas soovid AI analüüsi (*soovitan enamjaolt peale jätta, ilma selleta ei saa kirjalikku kokkuvõtet*)
5. Vajutad "Compare"nuppu
6. Ja natukese aja pärast(*viimasel ajal varieerub väga, sest nõudlus hetkel suur, seega umbes 20s-45s peaks minema, vahepeal üle minuti, kui üks mudel failib ja fallbackib teise mudeli peale*) peaks sinuni jõudma AI analüüs(*kui selle filtrina lisasid*) ning funktsiooni baasil erinevad koodivõrdlused allapoole kerides.

Kui kasutate minu testrepot(*uni-id: ronook, aasta: 2024*), siis seal on mul olemas testimiseks ülesanded: EX01, EX02, EX04, EX06 ja EX08 Seega teiste ülesannete katsetamisel tuleb error.

AI analüüs:

Võiksite jälgida, kas AI analüüsis on välja toodud ülesannete vead(*juhul kui kasutate minu test-repot*): EX01 - `maths.py` failis suured vead, `poem.py` peaks mainima, et tuleks kontrollida, kas output matchib sellega, mida küsitakse.

EX02 - `prime.py` failis on vead, AI võib ka teiste failide kohta kisa tõsta, kuid üldiselt peaks lihtsalt olema sellised, et võib midagi juhtuda stsenaariumid. Võiks mainida, et tudengi lahendus on ebaefektiivne võrreldes õppejõu tulemusega.

EX04 - `lists.py` failis, funktsioonid `search_by_brand` ja `search_by_model` peaksid failima, analüüsis peaks olema välja toodud.

EX06 - peaks olema täiesti korras, ilmselt leiab analüüs mingid edge case vead, aga läbi lugedes peaks aru saama, et üldiselt kood on okei, sest kood läbis testeri.

EX08 - `recursion.py` peaks olema üldiselt korras, kahe funktsiooni implementatsioon on puudu ja analüüs peaks selle välja tooma.

Funktsiooni põhised võrdlused (*allpool accordion component*):

Palun vaadake neid kriitilise pilguga, need on mõeldud selliseks viimaseks õlekõrreks.

Kui AI-analüüs ei anna loodetud tulemust, siis oleks mõistlik neid vaadata ning näha, kas on võimalik sealt mingi viga üles leida.

Seal on siis olemas:

- Strict Comparison, mis on tekstipõhine range võrdlus, mis võrdleb rida, rea haaval.
- Structural Comparison, mis on struktuuripõhine võrdlus ehk ideaalses olukorras võiks olla kogu flowchart valge ning ükski node ei tohiks olla värvitud. Kui on, siis vastavalt värviskeemile saab vaadata, et mis on erinevused tudengi ja õppejõu koodi vahel. Flowcharti baasiks on õppejõu kood.
- Diff view, mis on klassikaline gitlab/github stiilis diff, kus on näha otsene koodierinevus kahe koodiploki vahel.

Rakenduse testimise hea tava:

Juhul kui compare call peaks failima ja tuleb error, siis proovige paar korda uuesti, sest vahepeal taltechi server jupsib. Vahepeal annab AI analüüs mingi funktsiooni kohta ISSUES staatuse isegi kui sellel funktsioonil otseselt midagi viga ei ole, siis tuleks süveneda, mis ta põhjenduseks välja toob, tihtipeale on tegemist mingi edge case'iga, mida ilmselt tester ei testi. Proovige kindlasti läbi erinevad ülesanded ning prooviga võrrelda mitu korda

sama ülesannet, et näha, kas AI analüüs muutub või jääb sarnaseks. Uurige ka funktsiooni põhiseid võrdlusi ning andke teada, kas need on kasulikud sellise rakenduse kontekstis.

PS! Küsimuste korral palun kirjutage mulle otse discordi: `goofy_ee`