TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl-Uljas Mölder 213479IAIB

# TUDENGITE PROGRAMMEERIMISEÜLESANNETE AUTOMAATNE HINDAMINE SUURTE KEELEMUDELITE ABIL

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: Karl-Uljas Mölder

04.06.2025

# Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on välja töötada suurtele keelemudelitele tuginev veebipõhine hindamisassistent CodeGrader, mis on loodud TalTechi infotehnoloogia teaduskonnas mahukate tudengitööde hindamisprotsessi optimeerimiseks. Probleemiks, mida lõputöö lahendab, on traditsioonilise manuaalse hindamise suur ajakulu ja õppejõudude koormus suure hulga tudengite koodide ja testide analüüsimisel ning nendele argumenteeritud tagasiside andmisel.

Loodud CodeGrader süsteem võimaldab õppejõududel valida neile sobiva suure keelemudeli ning genereerida koodianalüüs ja struktureeritud tagasiside õppejõudude poolt varasemalt kasutatud hindamisloogika põhjal. Töö keskmes on funktsionaalse prototüübi loomine, mis toetab tudengitööde üleslaadimist, erinevate suurte keelemudelite kasutamist ning koodi funktsionaalsuse hindamist. Olulist tähelepanu pööratakse süsteemi paindlikkusele, et tulevikus võimaldada uute keelemudelite kerget integreerimist ja lahenduse kohandamist teistele programmeerimisainetele.

Töö tulemusena valmib praktiline tööriist, mis aitab muuta programmeerimisainete hindamisprotsessi efektiivsemaks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 7 peatükki, 14 joonist, 2 tabelit.

# Abstract

## Automated Assessment of Student Programming Assignments Using Large Language Models

The aim of this bachelor's thesis is to develop CodeGrader, a web-based grading assistant leveraging large language models, designed to optimize the assessment process of extensive student assignments within TalTech's information technology curricula. The problem addressed by this thesis is the significant time consumption and faculty workload associated with traditional manual grading, particularly in analyzing a large volume of student-submitted code and tests, and providing them with substantiated feedback.

The developed CodeGrader system enables educators to select a suitable large language model to generate code analysis and structured feedback based on the grading logic previously employed by the teaching staff. The core of this work involves the creation of a functional prototype that supports the uploading of student assignments, the utilization of various large language models, and the assessment of code functionality. Considerable attention is given to the system's flexibility to allow for the straightforward integration of new language models in the future and the adaptation of the solution for other programming courses.

The result of this thesis is a practical tool that contributes to making the grading process in programming courses more efficient.

The thesis is written in Estonian and is 29 pages long, including 7 chapters, 14 figures and 2 tables.

# Lühendite ja mõistete sõnastik

| | |
|---|---|
| AI | *Artificial Intelligence*, tehisintellekt – intellektitehnika uurimis- ja arendusobjekt |
| API | *Application Programming Interface*, rakendusliides |
| API token | rakenduspääse |
| CI/CD | *Continuous integration and Continuous delivery*, programmi automaatne testimine, serverisse laadimine ja käima panemine |
| CSV | *Comma-Separated Values*, komaeraldusega väärtused |
| Docker | konteinerite platvorm, mis komplekteerib tarkvara koos kõigi selle vajalike lisakomponentidega |
| GitLab | versioonihaldustarkvara |
| HTML | *HyperText Markup Language*, hüperteksti märgistuskeel |
| JPA | *Java Persistence API*, andmebaasi ning Java programmi vaheline liides |
| JSON | *JavaScript Object Notation*, formaat andmete esitamiseks |
| LLM | *Large language model*, suur keelemudel |
| OAuth | *Open Authorization*, volitusprotokoll, mis võimaldab lubada ühel rakendusel või teenusel teise sisse logida ilma privaatset teavet avaldamata |
| REST | *Representational State Transfer*, tarkvaraarhitektuuri standard |
| Transformer | masinõppes levinud kihtarhitektuur, mis võimaldab sisend- jada osade rööptöötlust |
| URI | *Uniform Resource Identifier*, kõigi nime- ja aadressitüüpide üldnimetus, mis viitavad objektile Internetis |
| URL | *Uniform Resource Locator*, veebiaadress |
| YAML | *Human-readable data-serialization language*, formaat andmete esitamiseks |

# Sisukord

# Jooniste loetelu

# Tabelite loetelu

# 1 Sissejuhatus

Infotehnoloogiaõppekavade programmeerimiskursustel on tarkvaratööde hindamine traditsiooniliselt üks aeganõudvamaid etappe. Eriti hästi ilmneb see TalTechi informaatika õppekava kohustuslikus aines ITI0202 „Programmeerimise põhikursus" [1], kus seni on veidi enam kui 130 üliõpilast sooritanud semestri lõpus mahuka eksami koos iseseisvalt kirjutatud testidega. Suur tööhulk ja vajadus anda igale tudengile argumenteeritud tagasiside koormavad õppejõude märkimisväärselt.

Käesolev bakalaureusetöö keskendub just ITI0202 mahukate programmeerimisetööde hindamisele: eesmärk on luua suurtele keelemudelitele, LLM-idele (*Large Language Models*), toetuv abiteenus, mis

1. analüüsib tudengite koodi ja teste LLM-ide kaudu,
2. tagastab struktureeritud hinnangu sama loogika järgi, mida õppejõud seni käsitsi koostasid,
3. on tulevikus kergesti integreeritav õppeplatvormidega ning vähendab hindamisprotsessi ajakulu,
4. võimaldab tulevikus LLM-e vahetada ilma rakendust oluliselt ümber kirjutamata.

Viimastel aastatel on suured keelemudelid muutunud kättesaadavaks ning suudavad minutite jooksul mõista lähtekoodi, tuvastada nõuete katvust ja vormistada selge tagasiside. Praktikas seatakse neile siiski piiranguid: mudelite kiire versiooniuuendus, *API*-de (*Application Programming Interface*) kulud, andmekaitse ning integratsioon olemasolevate õppeplatvormidega.

Käesolev lõputöö on üles ehitatud mitmeosalisena. Esimene peatükk annab taustainfo ja võrdleb levinumaid suuri keelemudeleid koos nende kasutuspiirangutega. Teine peatükk kirjeldab lahendusele seatud nõudeid ja toob välja kavandatud arhitektuuri. Kolmandas peatükis tutvustatakse loodud süsteemi ning valitud tehnoloogiaid. Neljas peatükk käsitleb evitamise ning avalikustamise töövoogu ja tulevikuplaane. Tulemuste peatükk esitab lahenduse täpsust, mille järel demonstreeritakse, kuidas rakendust kasutada.

# 2 Taustauuring

Käesolev peatükk annab ülevaate programmeerimistööde hindamise senistest praktikatest, selgitab kaasaegsete suurte keelemudelite aluseks olevat *transformer*-arhitektuuri ja selle tähtsust antud töö kontekstis, analüüsib LLM-ide potentsiaali hindamisvaldkonnas, võrdleb levinumaid mudeleid ning põhjendab vajadust loodava LLM-põhise hindamisassistendi järele.

## 2.1 Senised lähenemisviisid programmeerimistööde hindamisele

Siiani on toimunud tööde hindamine peaaegu täielikult manuaalselt. Kuigi see on enamasti väga täpne ning tagab hea tulemuse, siis nõuab see siiski õppejõududelt palju aega ja suurt panust, et tagada igale tudengile põhjalik, argumenteeritud ning mitte kallutatud tagasiside. Lisaks ajakulule võib manuaalne hindamine olla subjektiivne ning raskendada hindamistulemuste ühtluse tagamist erinevate hindajate või tööde vahel tulenevalt sellest, kui kogenud või näiteks väsinud õppejõud hindamise hetkel on.

## 2.2 Suurte keelemudelite potentsiaal ja rakendused hindamisel

Viimastel aastatel on toimunud märkimisväärne areng suurte keelemudelite valdkonnas, mis on avanud uusi võimalusi ka haridustehnoloogias, sealhulgas programmeerimistööde hindamisel.

Suured keelemudelid on tehisintellekti mudelid, mida on treenitud tohutul hulgal teksti- ja koodiandmetel. Tänu sellele on neil muljetavaldav võimekus mõista ja analüüsida programmikoodi erinevates keeltes. Enamik kaasaegseid suuri keelemudelid põhineb transformer-arhitektuuril, mis võimaldab neil efektiivselt töödelda pikki andmejadasid ja tabada kontekstuaalseid seoseid [2].

Generatiivne tehisintellekt on tehisintellekti alamhulk, mis kasutab masinõppemudeleid, et luua algupärast sisu kasutajale sisendite põhjal [3]. Hiljutised edusammud generatiivsete tehisintellektidega on muutnud olukorda, võimaldades kasutada üldotstarbelisi, eelnevalt treenitud mudeleid ilma ulatusliku spetsiifilise treenimiseta. See on avanud ukse LLM-ide laialdasemale kasutusele koodi hindamisel ja tagasiside andmisel:

1. Koodi analüüs: LLM-id suudavad analüüsida koodi süntaksit, tuvastada potentsiaal-

seid vigu, mõista koodi loogikat ja isegi hinnata selle vastavust etteantud nõuetele.
2. Tagasiside genereerimine: Üks paljulubavamaid aspekte on LLM-ide võime genereerida personaalset ja sisulist tagasisidet. Need mudelid ei piirdu ainult vigade tuvastamisega, vaid võivad pakkuda selgitusi, soovitusi parendusteks ja isegi näiteid.

### 2.2.1 Transformer-arhitektuur

Enne süvenemist suurte keelemudelite rakendustesse on oluline mõista, kuidas need töötavad. Enamik tänapäevaseid suuri keelemudeleid, sealhulgas käesolevas töös võrreldavad ja kasutatavad mudelid, põhinevad transformer-arhitektuuril. Transformer arhitektuuri pakuti välja 2017. aastal artiklis "*Attention is all you need*" (Vaswani et al., 2017) [4]. Selle peamine uuendus on tähelepanumehhanism (*attention mechanism*), eriti enesetähelepanu (*self-attention*), mis võimaldab mudelil kaaluda sisendteksti või koodi erinevate osade, mis on tekstist tuletatud (tokenite) suhtelist tähtsust üksteise suhtes. See tähendab, et mudel suudab tuvastada seoseid ja sõltuvusi ka pikkade tekstide või koodilõikude ulatuses, mis on eriti oluline programmikoodi analüüsimisel, kus ühe osa funktsionaalsus võib sõltuda teises, kaugemal asuvas osas defineeritust.

Erinevate tudengite koodi analüüsides tekivad erinevate suurustega kontekstiaknad – maksimaalne tokenite arv, mida mudel suudab ühes päringus korraga töödelda ja arvesse võtta. Kui sisendi maht ületab mudeli kontekstiakna, siis läheb kas osa informatsioonist kaduma või ei saa API-päringut sooritada.

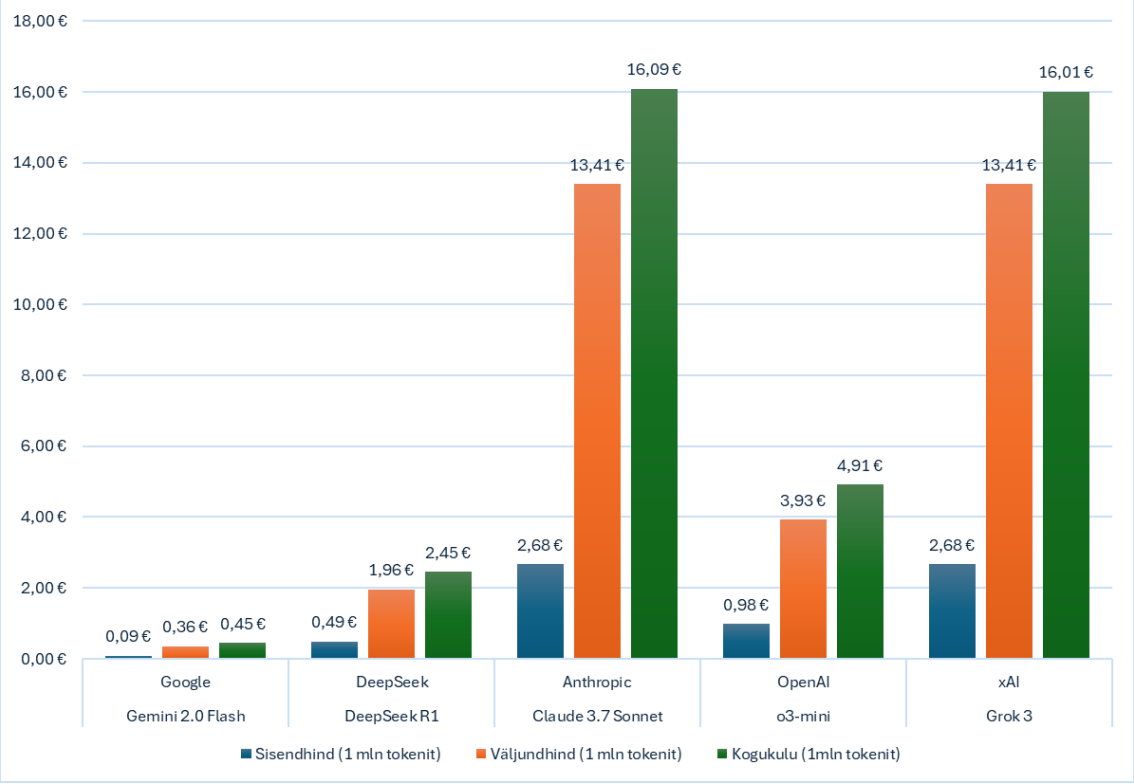
## 2.3 Levinumate suurte keelemudelite omadused ja võrdlus

Tuginedes eelnevalt käsitletud transformer-arhitektuurile ja generatiivse tehisintellekti võimekusele, on tänapäevased suured keelemudelid saavutanud muljetavaldava taseme teksti ja koodi mõistmisel ning genereerimisel. Enamasti pääseb fikseeritud tasu eest neile ligi API-liideste kaudu, mis võimaldab nende integreerimist erinevatesse rakendustesse, sealhulgas käesolevas töös loodavasse hindamisassistenti. Valides sobivat LLM-i tudengikoodi hindamiseks, kerkisid esile mitmed kriitilised valikukriteeriumid. Esiteks, mudeli töötlemiskiirus on oluline, et tagada õppejõule sujuv ja operatiivne töövoog, eriti paljude tööde hindamisel. Teiseks, arvestades tudengiprojektide ja hindamisnõuete mahtu (hinnanguliselt 30–35 tuhat tokenit koos viipade ja oodatava väljundiga), oli määravaks mudeli piisavalt suur kontekstiaken, mis võimaldab kogu relevantset infot korraga töödelda. Kolmandaks, mudel pidi suutma genereerida struktureeritud väljundit vastavalt keerukatele ja täpsetele viipadele, mida hindamisprotsess eeldab. Viimaks, arvestades akadeemilise

projekti ressursipiiranguid ja potentsiaalselt suurt analüüsitavate tööde hulka, oli oluline tegur ka kuluefektiivsus.

Nende kriteeriumite alusel teostati 03.03.2025 seisuga saadaolevate suurte keelemudelite vahel analüüs. Fookusesse võeti mudelid, mis olid tuntud oma hea koodianalüüsi võimekuse poolest ning mille tehnilised parameetrid, eriti deklareeritud või testitud kontekstiaken, ületasid minimaalselt 32 000 tokenit – tagades sellega piisava mahu ka keerukamate tudengitööde ja detailsete nõuete käsitlemiseks. Päringute keskmine kontekstiaken antud lõputöö raames oli olenevalt tööst vahemikus 30 000 kuni 42 000 tokenit. Sõelale jäi viis tolle hetke silmapaistvamat kandidaati: Google Gemini 2.0 Flash [5], DeepSeek R1 [6], Claude 3.7 Sonnet [7], OpenAI o3-mini [8] ja xAI Grok 3 [9].

Nende mudelite üheks olulisemaks praktiliseks võrdlusmomendiks osutus API kasutamise hind. Joonisel 1 on esitatud nende viie keelemudeli hinnastamise võrdlus miljoni tokeni kohta, eristades sisend- ja väljundtokenite maksumust ning arvutades ka summaarset kogukulu. Nagu jooniselt näha, varieeruvad mudelite hinnaklassid märkimisväärselt, alates 0.45€-st kuni 16.09€-ni. See hinnaerinevus on oluline tegur sobivaima mudeli või mudelite valikul pikaajaliseks ja laialdaseks kasutuseks.




Joonis 1. Suurte keelemudelite hinnavõrdlus.

Et anda põhjalikum ülevaade väljavalitud viiest mudelist, mis vastasid esmastele kriteeriu-

mitele (hea koodianalüüsi võimekus ja kontekstiaken, mis on suurem kui 32k tokenit), on järgnevas tabelis 1 toodud nende peamised omadused ja võrdluspunktid, mis on relevantsed käesoleva töö kontekstis.

Tabel 1. Väljavalitud keelemudelite (>32k kontekstiaken) omaduste võrdlus.

| **Omadus** | **Gemini 2.0** | **DeepSeek R1** | **Claude 3.7** | **o3-mini** | **Grok 3** |
|---|---|---|---|---|---|
| Teenusepakkuja | Google | DeepSeek | Anthropic | OpenAI | xAI |
| Kontekstiaken | 1000000 | 262000 | 328000 | 300000 | 136000 |
| Hind (€ / 1 mln tokenit) | 0.45 | 2.45 | 16.09 | 4.91 | 16.01 |
| Latentsus (ms) | 0.4 | 0.3 | 0.4 | 5.2 | 0.7 |
| Läbilase (t/s) | 183 | 0.9 | 42 | 115 | 100 |
| Teadmiste ajaline piir (kuu.a) | 06.2024 | 07.2024 | 10.2024 | 10.2023 | 11.2024 |

### 2.3.1 Sobilikud mudelid

Valikusse jäid kolm kõige paremini nõudeid täitvat ning samal ajal odavamat mudelit: OpenAI o3-mini, DeepSeek R1 ning Gemini 2.0 Flash. Mudeleid võrreldi ning testiti kolme 2023. aasta ITI0202 aine eksamiülesannete ("Spordiklubi", "Hotell", "Ülikool") põhjal.

Mudelite täpsust mõõdeti protsentuaalselt, võrreldes keelemudeli poolt genereeritud hinnanguid (kas tudengi lahendus vastas konkreetsele funktsionaalsusnõudele või mitte) õppejõudude poolt antud referentshinnangutega. Iga ülesande puhul on sulgudes näidatud keelemudeli poolt õigesti hinnatud nõuete arv võrreldes õppejõudude hinnanguga ning hinnatud nõuete koguarv antud ülesandes selle mudeli jaoks (näiteks 1027/1404 tähendab, et 1404-st nõudest hindas mudel korrektselt 1027). Hinnatud nõuete koguarv võis mudeliti ja ülesannete lõikes varieeruda, kuna mõnel juhul ei tagastanud mudel tehnilistel põhjustel või päringu ajalimiidi tõttu kõigi nõuete kohta tulemust – sellisel juhul jäeti tööd antud mudeli täpsusarvutusest välja. Lisaks täpsusele võrreldi mudelite praktilist keskmist päringu sooritamise kiirust tööde hindamise käigus ja hinnangulist API kasutamise kulu.

Tabel 2. Keelemudelite OpenAI o3-mini, DeepSeek R1 ja Gemini 2.0 Flash võrdlev analüüs ITI0202 eksamiülesannete hindamisel.

| **Ülesanne / Omadus** | **OpenAI o3-mini** | **DeepSeek R1** | **Gemini 2.0 Flash** |
|---|---|---|---|
| Spordiklubi täpsus (%) (õigeid / kõigist) | 73,15 (1027/1404) | 74,20 (1027/1404) | 73,22 (1028/1404) |
| Hotell täpsus (%) (õigeid / kõigist) | 92,76 (974/1050) | 91,91 (1057/1150) | 91,71 (642/700) |
| Ülikool täpsus (%) (õigeid / kõigist) | 82,61 (1050/1271) | 82,49 (1074/1302) | 81,64 (1063/1302) |
| Keskmine täpsus (%) | 82,84 | 82,87 | 82,19 |
| Hind (€ / 1 mln tokenit) | 1.02 | 0.13 | 0/0.093 |
| Keskmine päringu kiirus (s) | 35 | 80 | 66 |

Nagu tabelist näha, olid testitud mudelite keskmised täpsused küllaltki sarnased, kõikudes 82% ja 83% vahel. OpenAI o3-mini oli keskmiselt kõige kiirem, kuid ka kõige kulukam. DeepSeek R1 pakkus parimat hinna ja täpsuse suhet, olles samas kõige aeglasem. Gemini 2.0 Flash oli hinna poolest soodne (hind 0€ kuni päevase tasuta limiidi täitumiseni), kiiruselt keskmine, kuid "Hotelli" ülesande puhul ilmnes probleeme kõigi tulemuste kättesaamisel.

## 2.4 Kokkuvõte

Võttes arvesse, et keelemudelid pidevalt uuenevad ning muutuvad nii täpsemaks, kiiremaks kui ka potentsiaalselt hinna poolest kättesaadavamaks, ei olnud võimalik teha kindlat pikaajalist otsust, milline mudel oleks alati ja igas olukorras parim antud lõputöö raames loodava teenuse jaoks. Keelemudelite jõudlus võib varieeruda ka sõltuvalt sisendandmete spetsiifikast ja päringu keerukusest. Sellest tulenevalt tehti disainiotsus luua õppejõududele võimalus valida hindamisel mitme integreeritud keelemudeli vahel, et nad saaksid kasutada just hetkevajadustele ja eelistustele kõige paremini vastavat mudelit. See tagab süsteemi paindlikkuse ja tulevikukindluse.

# 3 Nõuded

Antud peatükk keskendub loodava suurtele keelemudelitele põhineva hindamisassistendi nõuete määratlemisele. Nõuded kirjeldavad süsteemi põhitoiminguid ja omadusi, seavad piirangud ning kvaliteedinäitajad süsteemi tööle ja tulenevad töö eesmärgist ning taustauuringust. Nõuded kirjeldavad, mida süsteem peab tegema – milliseid tegevusi pakkuma ja millistele kasutaja interaktsioonidele reageerima. Antud lõputöö raames loodava hindamisassistendi peamised nõuded on järgmised:

1. Kasutajate autentimine ja autoriseerimine
    - Süsteem peab võimaldama kasutajatel sisse logida, kasutades nende TalTech GitLab kontot.
    - Süsteem peab tuvastama autenditud kasutaja ja võimaldama juurdepääsu tema spetsiifilistele andmetele (analüüside ajalugu).
2. Sisendandmete üleslaadimine ja haldamine
    - Toetatud peab olema üksiku tudengi töö üleslaadimine ja hindamine.
    - Toetatud peab olema mitme tudengi töö üleslaadimine.
    - Süsteem peab võimaldama kasutajal üles laadida nõuete/testide faili, mis sisaldab hindamiskriteeriumeid või funktsionaalsusnõudeid, mida LLM analüüsil kasutab.
3. Keelemudeli valik ja analüüsi teostamine
    - Süsteem peab pakkuma kasutajale võimalust valida analüüsiks kasutatav suur keelemudel.
    - Süsteem peab edastama valitud keelemudelile tudengi koodi koos üleslaetud nõuete/testide faili sisuga sobivalt vormindatud viibaga.
4. Tulemuste esitamine ja haldamine
    - Üksiku töö analüüsi puhul peab süsteem kuvama LLM-i poolt genereeritud tulemuse (hinnangu ja selgitused) otse veebiliideses tekstialana.
    - Süsteem peab võimaldama õppejõul alla laadida üksiku töö analüüsi tulemuse tekstifailina.
    - Mitme töö massanalüüsi puhul peab süsteem teavitama õppejõudu töötluse staatusest ja pakkuma võimalust laadida tulemused alla koondfailina (ZIP-fail, mis sisaldab individuaalseid analüüsiraporteid).
    - Süsteem peab salvestama autenditud kasutaja poolt tehtud analüüside ajaloo (kasutatud failid, API valik, analüüsi aeg).
    - Autenditud kasutajal peab olema võimalik sirvida enda analüüside ajalugu

veebiliideses.

5. Integratsioon
    - Süsteem peab olema disainitud viisil, mis võimaldab tulevikus süsteemi kergesti integreerida õpikeskkonnaga (Moodle).

# 4 Rakenduse arhitektuur, komponendid ja kasutuselevõtt

Selles peatükis kirjeldatakse detailselt käesoleva bakalaureusetöö raames loodud LLM-põhise hindamisassistendi CodeGrader arhitektuuri ja peamisi tehnilisi komponente. Iga komponendi juures selgitatakse selle eesmärki, ülesehitust ning arendamisel kasutatud keskseid tehnoloogiaid ja tööriistu. Eesmärk on anda põhjalik ülevaade süsteemi toimimisest ja tehtud tehnoloogilistest valikutest.

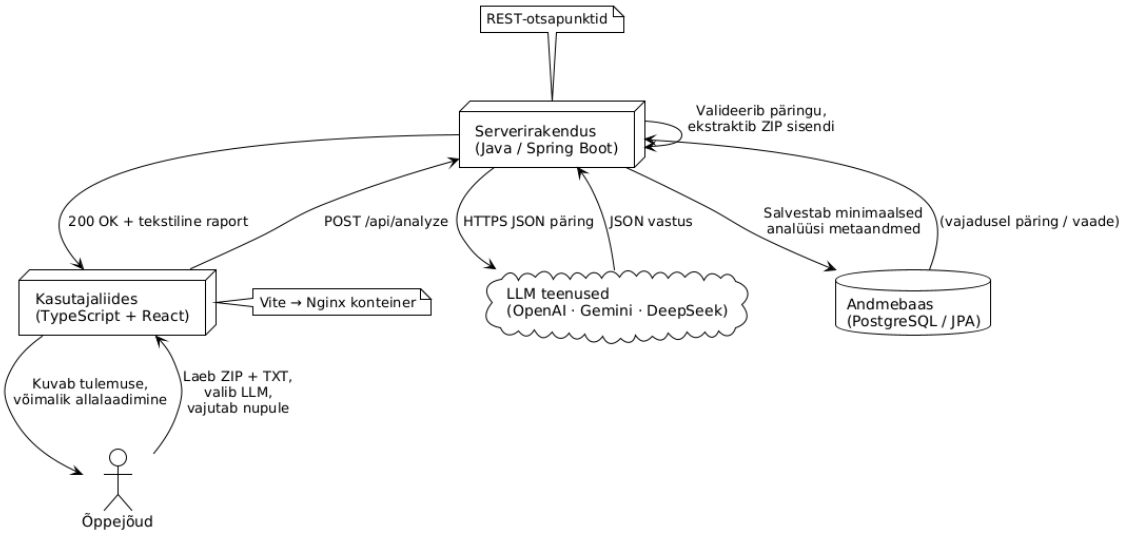
## 4.1 Arhitektuur

Loodud CodeGrader süsteem on realiseeritud mitmekihilise klient-server arhitektuurina, mis tagab modulaarsuse ja skaleeritavuse. Süsteemi peamised komponendid ja nendevahelised seosed on kirjeldatud alljärgnevalt.

### 4.1.1 Üldine ülesehitus

1. Kasutajaliides: Interaktiivne veebileht, mis on loodud TypeScripti ja Reacti abil ning jookseb kasutaja veebilehitsejas. See võimaldab õppejõududel faile üles laadida, analüüsirežiime ja keelemudeleid valida ning tulemusi vaadata ja hallata.
2. Serverirakendus: Serveripoolne rakendus, mis on arendatud Java keeles Spring Boot raamistiku abil. Serverirakendus täidab keskset rolli, pakkudes REST API liidest kasutajaliidesele. Selle peamised ülesanded on:
   - kasutajate autentimine ja autoriseerimine,
   - API päringute vastuvõtmine ja valideerimine,
   - sisendfailide (tudengite .zip-failid, nõuete .txt-failid) töötlemine,
   - suhtlus suurte keelemudelitega: päringute koostamine vastavalt valitud mudelile ja viibale, vastuste vastuvõtmine ja esmane töötlemine,
   - analüüsiajaloo salvestamine ja pärimine andmebaasist.
3. LLM teenused: OpenAI, Google ja DeepSeek poolt pakutavad pilvepõhised suured keelemudelid. Serverirakendus suhtleb nende teenustega turvaliste API-kutsete kaudu, edastades neile töödeldud tudengi koodi ja hindamiskriteeriumid ning saades vastu analüüsitulemuse.
4. Andmebaas: Kasutatakse minimaalsete püsivate metaandmete, nagu kasutajakontode ja sessiooniteabe, salvestamiseks.

Süsteemi üldine andmevoog algab kasutajaliidesest, kus õppejõud esitab analüüsipäringu. Serverirakendus võtab selle vastu, suhtleb LLM-iga ja tagastab tulemuse kasutajaliidesele.




Joonis 2. Rakenduse üldarhitektuur.

### 4.1.2 Autentimine ja autoriseerimine

Kasutajate turvaline autentimine on süsteemi oluline osa tagamaks, et rakendusele pääsevad ligi ainult volitatud õppejõud. Autentimine on lahendatud TalTechi GitLab keskkonna kaudu, kasutades OAuth2 autoriseerimise voogu [10]. See võimaldab kasutajatel logida sisse oma olemasoleva TalTechi GitLab kontoga, ilma et nad peaksid CodeGrader rakendusele kunagi oma GitLabi parooli sisestama või edastama. Parool sisestatakse ainult otse TalTechi GitLabi turvalises autentimiskeskkonnas. Pärast edukat autentimist ja kasutaja nõusolekut saab CodeGrader rakendus GitLabist pääsutalongi, mida kasutatakse edaspidiseks autoriseeritud suhtluseks.

1. Kasutaja vajutab CodeGraderi kasutajaliideses "Logi sisse" nuppu.
2. Kasutajaliides suunab kasutaja serverirakenduse otspunkti.
3. Spring Security suunab kasutaja edasi TalTechi GitLab autoriseerimisserverisse.
4. Kasutaja autendib ennast GitLabis, kui ta pole juba sisse logitud.
5. GitLab küsib kasutajalt nõusolekut rakendusele ligipääsuks tema profiiliandmetele.
6. Pärast nõusoleku andmist suunab GitLab kasutaja tagasi rakenduse eelnevalt konfigureeritud redirect URI-le (*Uniform Resource Identifier*), lisades URL-ile (*uniform resource locator*) autoriseerimiskoodi.
7. Serverirakendus (Spring Security) vahetab selle koodi GitLabiga pääsutalongi (*access token*) vastu.

8. Pääsutalongi kasutades hangib serverirakendus GitLab API kaudu kasutaja põhiandmed, mida kasutatakse rakendusesiseselt kasutaja identifitseerimiseks
9. Luuakse kasutajaseanss ning kasutaja suunatakse rakenduse pealehele.

## 4.2 Serverirakendus

Serverirakendus on Java ja Spring Booti peale ehitatud REST-teenus: kasutajaliidese tehtud päring jõuab Spring Booti kontrollerisse, kus see valideeritakse ning sisendfailid töödeldakse Java utiliitidega. Kontroller koostab valitud LLM-ile sobiva API-päringu, edastab selle ja vormistab saadud vastuse. LLM-i tagasisidet ega faile püsivalt ei talletata – tulemus liigub otse samas HTTP-vastuses tagasi veebikliendile.

### 4.2.1 Programmeerimiskeel Java ja Spring Boot raamistik

Java on kõrgetasemeline, objektorienteeritud ja platvormist sõltumatu programmeerimiskeel, mis on laialdaselt kasutusel suurettevõtete tarkvara, veebirakenduste ja mobiilirakenduste arendamisel [11]. Java virtuaalmasin (JVM) võimaldab Java baitkoodi käivitada erinevates operatsioonisüsteemides ilma koodi muutmata [12]. Keele tugevad küljed on selle robustsus, suur standardteekide kogu, automaatne mäluhaldus ning tugev kogukonna tugi ja dokumentatsioon. Käesolevas projektis valiti Java tagarakenduse peamiseks programmeerimiskeeleks just nende omaduste tõttu. Objektorienteeritud lähenemine võimaldas luua selgelt struktureeritud ja hooldatavat koodi. Lisaks on olemas lai valik Java teeke ja raamistikke nagu Spring Boot, mis kiirendavad arendusprotsessi.

Spring Boot on avatud lähtekoodiga Java-põhine raamistik, mis on loodud iseseisvate, produktsioonivalmis Spring-rakenduste lihtsaks ja kiireks loomiseks [13]. Spring Boot pakub automaatset konfiguratsiooni paljudele levinud teekidele, sisseehitatud veebiserverit nagu *Tomcat*, *Jetty* või *Undertow* ning mitmesuguseid sõltuvuskomplekte erinevate funktsionaalsuste nagu veeb, turvalisus, andmebaasid, kiireks lisamiseks. Selles projektis kasutati Spring Booti serveripoolse rakenduse loomiseks mitmel põhjusel:

1. Kiire arendus: Spring Boot võimaldab keskenduda äriloogika kirjutamisele, vähendades korduvat koodi ja konfiguratsioonide hulka.
2. Integreeritud lahendused: See integreerub sujuvalt teiste Springi projektidega, nagu Spring Web MVC REST API-de loomiseks, Spring Security turvalisuse tagamiseks ja Spring Data JPA andmebaasidega suhtlemiseks [14]. Kõiki neid kasutati ka antud töös.
3. Sõltuvuste haldus: Sõltuvuskomplektid lihtsustavad oluliselt projekti sõltuvuste

haldamist ja versioonide ühilduvuse tagamist.

4. Produktsioonivalmidus: Spring Boot pakub vaikimisi mitmeid produktsiooniks vajalikke omadusi nagu tervisekontrolli otspunktid (*actuator*) ja lihtsat konfiguratsiooni väliste seadistusfailide kaudu (application.properties).

### 4.2.2 REST API

Representational State Transfer (REST) on tarkvaraarhitektuuri stiil hajusate süsteemide, eriti veebiteenuste loomiseks [15]. REST API-d kasutavad standardseid HTTP meetodeid (GET, POST, PUT, DELETE jne) ressurssidega suhtlemiseks ning andmevahetuseks kasutatakse tihti JSON (JavaScript Object Notation) või XML formaati. REST API-d on staatusevabad (stateless), mis tähendab, et iga päring serverile peab sisaldama kogu vajalikku infot selle täitmiseks, ning server ei salvesta kliendi seansiteavet päringute vahel.

CodeGraderi tagarakendus pakub RESTful API liidest, mille kaudu kasutajaliides ja potentsiaalselt ka muud süsteemid saavad selle funktsionaalsust tarbida.

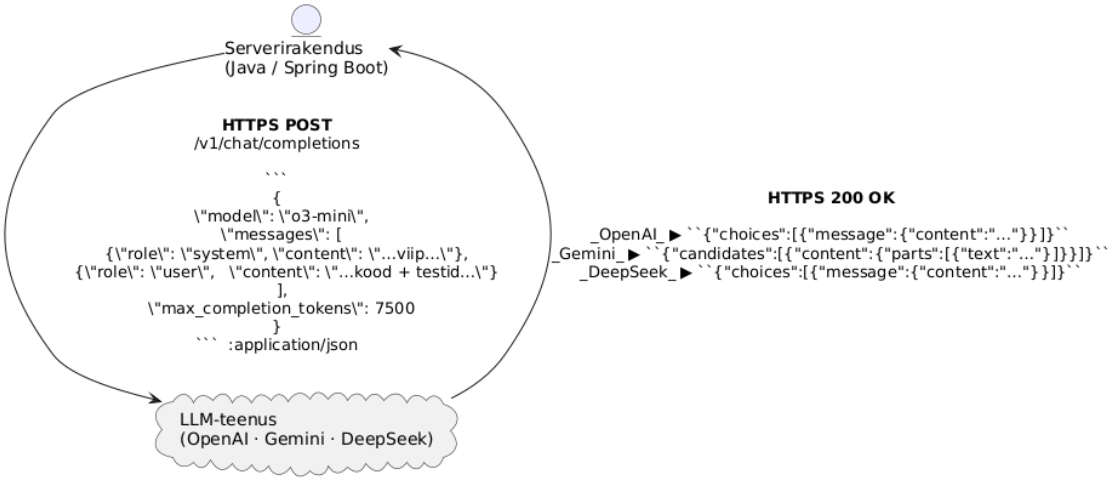
### 4.2.3 Gradle

Projekti sõltuvuste haldamiseks, koodi kompileerimiseks, testimiseks ja rakenduse ehitamiseks (pakendamiseks) kasutati projektihaldus- ja ehitustööriista Gradle [16]. Gradle on kaasaegne ja paindlik ehitusautomaatika tööriist, mis on eriti populaarne Java, Groovy, Kotlini ja mitmekeelsete projektide puhul. Gradle sai valitud üle Maveni, sest Gradle'i ehitusskriptid pakuvad suuremat paindlikkust ja võimaldavad keerukama ehitusloogika loomist võrreldes Maveni XML-põhise konfiguratsiooniga. Lisaks Gradle kasutab ehitamise kiirendamiseks mitmeid tehnikaid, nagu inkrementaalne ehitamine (*incremental builds*) ja ehitamise vahemälu (*build cache*), mis võib teatud juhtudel viia kiiremate ehitusaegadeni, eriti suurtes projektides.

## 4.3 Suurte keelemudelitega suhtlemine

API-dega suhtlemiseks Java tagasüsteemiga kasutati OkHttp3 teeki, mis on populaarne ja efektiivne HTTP klient. Päringute koostamine ja vastuste parsimine (JSON formaadis) teostati Jackson teegi abil. Loodi spetsiifilised kliendiklassid iga LLM-i jaoks ning tehaseklass nende dünaamiliseks valimiseks. Vajalikud viibad LLM-idele, nagu kirjeldatud Lisas 2 (lk 42), defineeriti ja hoiti eraldi konfiguratsioonifailis, et neid oleks lihtne hallata ja muuta. Serverirakendus serialiseerib sisendi JSON-iks, lisab mudeli nime, süsteemi- ja kasutajasõnumid, tokenite limiidi ning seejärel teeb HTTPS POST päringu. LLM teenus

saab selle päringu kätte, analüüsib seda ning tagastab serverirakendusele struktureeritud JSON-vastuse (struktuur varieerub sõltuvalt mudelist).




Joonis 3. Serverirakenduse ja API-de vaheline suhtlus.

## 4.4 Kasutajaliides

Kasutajaliides on õppejõu peamine interaktsioonipunkt süsteemiga, võimaldades failide üleslaadimist, analüüsiparameetrite seadistamist ja tulemuste kuvamist. Selle arendamisel kasutati kaasaegseid veebitehnoloogiaid, et tagada hea kasutajakogemus, kiirus ja hooldatavus.

### 4.4.1 Programmeerimiskeel TypeScript

TypeScript on avatud lähtekoodiga programmeerimiskeel, mille on arendanud Microsoft [17]. See on JavaScripti [18] ülemhulk (*superset*), mis tähendab, et iga JavaScripti kood on ka korrektne TypeScripti kood. TypeScripti peamine lisandväärtus JavaScripti ees on staatiline tüübikontroll. See võimaldab arendajatel defineerida muutujatele, funktsiooniparameetritele ja -tagastustele ning objektiomadustele tüüpe. Tüübisüsteem aitab avastada vigu juba arendusfaasis, enne koodi käivitamist, mis vähendab oluliselt käitusaegseid vigu ja muudab suuremate projektide koodibaasi kergemini mõistetavaks, hallatavaks ja refaktoreeritavaks. TypeScript transpileeritakse (teisendatakse) tavaliseks JavaScripti koodiks, mida veebilehitsejad saavad käivitada. Arendamiseks valiti TypeScript mitmel põhjusel. Lisaks integreerub TypeScript väga hästi Reacti teegiga, võimaldades defineerida tüüpe Reacti komponentide omadustele (props) ja olekutele (state), mis aitab tagada komponentide korrektset kasutamist. Nende põhjuste pärast sai TypeScript valitud antud projekti jaoks.

### 4.4.2 Kasutajaliidese teek React

React (ka React.js või ReactJS) on avatud lähtekoodiga JavaScripti teek kasutajaliideste loomiseks [19]. React kasutab virtuaalset DOM-i (*Document Object Model*), et efektiivselt uuendada ainult neid kasutajaliidese osi, mis on muutunud, tagades nii hea jõudluse ka dünaamiliste ja andmemahukate rakenduste puhul [20].

CodeGraderi kasutajaliidese loomisel kasutati Reacti, kuna see pakub järgmisi eeliseid:

1. Komponendipõhine arhitektuur: Võimaldab jagada keeruka kasutajaliidese väiksemateks, hallatavateks ja taaskasutatavateks komponentideks.
2. Deklaratiivne programmeerimine: React võimaldab kirjeldada, kuidas kasutajaliides peaks teatud oleku korral välja nägema ning React hoolitseb ise selle oleku kajastamise eest reaalses DOM-is. See muudab koodi lihtsamini mõistetavaks ja hallatavaks.
3. Olekuhaldus (*State Management*): Reacti Hookid (veebikonksud), eriti useState ja useEffect, pakkusid mugavaid vahendeid komponentide siseoleku (millised failid on valitud, milline LLM on aktiivne, kas päring on töös) haldamiseks ja kasutajaliidese dünaamiliseks uuendamiseks vastavalt nendele olekutele.

### 4.4.3 Arenduskeskkond ja ehitustööriist Vite

Vite on kaasaegne kasutajaliidese arenduse tööriist, mis pakub kiiret arendusserverit ja optimeeritud ehitusprotsessi produktsiooni jaoks [21]. Erinevalt traditsioonilisematest paketihalduritest nagu Webpack, kasutab Vite arendusrežiimis ära brauserite natiivset tuge ES moodulitele (ESM) [22], mis võimaldab serveril käivituda peaaegu koheselt ja pakkuda kiiret moodulite uuendamist (*Hot Module Replacement*, HMR) ilma kogu rakendust uuesti ehitamata. Seepärast valiti Vite ka CodeGraderi kasutajaliidese tööriistaks.

## 4.5 Rakenduse infrastruktuur, kasutuselevõtt ja paigaldus

Selles peatükis kirjeldatakse CodeGrader rakenduse lähtekoodi haldamise põhimõtteid, praegust kasutuselevõtu protsessi TalTechi serverikeskkonnas ning antakse ülevaade CI/CD toru loomisest. Samuti käsitletakse süsteemi integreerimist Moodle'i õpikeskkonnaga.

### 4.5.1 Majutuskeskkonnad ja kättesaadavus

Arendusprotsessi varasemas erapis kasutati prototüüpimiseks Amazon Web Services (AWS) platvormi [23]. Nüüdseks on rakendus majutatud TalTechi serveris ja on avalikult kättesaadav aadressil https://cs.taltech.ee/services/codegrader/. Rakenduse konteineriseerimiseks ja paigaldamiseks kasutati Dockerit ning Docker Compose'i. See võimaldab ühtset keskkonda nii arenduses kui ka produktsioonis.

### 4.5.2 Konteineriseerimine

Docker on avatud lähtekoodiga platvorm rakenduste arendamiseks, tarnimiseks ja käivitamiseks konteinerites [24]. Konteinerid on iseseisvad tarkvarapaketid, mis sisaldavad kõike rakenduse käivitamiseks vajalikku: kood, käitusaeg (*runtime*), süsteemiteegid, seadistused. Erinevalt virtuaalmasinatest jagavad konteinerid peremeesarvuti operatsioonisüsteemi tuuma, mis muudab nad ressursisäästlikumaks ja käivitamisel kiiremaks.

Dockerit kasutatati projektis, kuna Docker tagab, et rakendus käitub identselt nii arendaja arvutis, testimisserveris kui ka produktsioonkeskkonnas. Nõnda välistati probleemid, mis tulenevad erinevatest tarkvaraversioonidest või konfiguratsioonidest erinevates keskkondades. Lisaks on Dockeris iga rakenduse komponentide sõltuvused isoleeritud ja pakitud eraldi Docker-konteinerisse. Nõnda välditi konflikte erinevate projektide või kompnentide vahel. Viimane põhjus oli konteineriseeritud rakenduse kerge paigaldamine uude keskkonda. Selleks piisab vaid Docker-tõmmise (*image*) olemasolust ja Dockeri käituskeskkonnast.

Projekti jaoks loodi Dockerfile-failid [25] (üks serverirakenduse ja teine kasutajaliidese jaoks), mis sisaldavad juhiseid Docker-tõmmiste ehitamiseks. Need kirjeldavad baastõmmist, kopeerivad rakenduse koodi tõmmisesse, installeerivad vajalikud sõltuvused ja konfigureerivad rakenduse käivitamise.

### 4.5.3 Mitmekonteineriliste rakenduste haldamine

Docker Compose on tööriist mitmekonteineriliste Docker-rakenduste defineerimiseks ja haldamiseks. See kasutab YAML-formaadis konfiguratsioonifaili (docker-compose.yml), et kirjeldada rakenduse teenuseid (*services*), võrke (*networks*) ja andmekogumeid (*volumes*) [26]. Ühe käsuga (docker compose up) saab käivitada kogu rakenduse koos kõigi selle komponentidega.

### 4.5.4 Pidev integratsioon ja tarnimine

Selleks, et rakenduse uuenduste ja parandust liigutamine oleks võimalikult efektiivne, turvaline ja kerge, rakendati pideva integratsiooni ja pideva tarnimise (CI/CD) toru põhimõtteid[27]. Peamine eesmärk oli automatiseerida korduvaid tegevusi nagu rakenduse komponentide /serverirakendus ja kasutajaliides) ehitamist ja Docker-tõmmiste loomist. Tänu CI/CD torule tagati ka, et uued koodiversioonid jõuaksid kiiresti ja ühetaoliselt serverisse, vähendades viivitusi ja inimlikke vigu, mis muidu võisid tekkida manuaalsel paigaldamisel. Kuigi toru ise ei lisa uusi turvakihte rakendusele, siis üks olulisemaid aspekte oli API ning andmebaasi võtmete turvaline haldamine. Selle asemel, et hoida tundlikke andmeid otse lähtekoodis, võimaldab CI/CD süsteem neid turvaliselt hallata ja süstida rakendusse alles ehitamise või käivitamise etapis. CI/CD keskkonnas saab defineerida turvalisi muutujaid (*secure environment variables*), mis edastatakse Docker-konteineritele käivitamisel läbi Docker Compose'i .env faili, mida ennast ei hoita versioonihalduses. See vähendab oluliselt riski, et salajased võtmed satuvad volitamata isikute kätte. CI/CD toru lõppeesmärk oli muuta uute versioonide serverisse viimine võimalikult lihtsaks ja riskivabaks.

### 4.5.5 Tulevikuplaanid

Nagu ennegi mainitud, siis praegu kasutab loodud hindamissüsteem eraldiseisvat serverirakendust, mille kasutajaliides asub TalTechi serveris kättesaadaval veebilehel. Kuigi rakenduse peamine töövoog eeldab tudengitööde ja nõuetefailide käsitsi üleslaadimist, on süsteemi arhitektuurist lähtuvalt võimalik seda tulevikus sügavamalt integreerida ülikooli keskse õpikeskkonna Moodle'iga.

Edasise arenduse käigus on plaanis viia kasutajaliidese põhifunktsionaalsus Moodle'i platvormi mooduliks, võimaldades õppejõududel ja tudengitel töid hinnata otse Moodle'i keskkonnas ilma eraldi veebirakendusse sisenemata. Selline lahendus võimaldaks kasutada olemasolevaid Moodle'i autentimis- ja failihaldusvõimalusi ning vähendaks vajadust tööde eraldi üleslaadimiseks ZIP-failidena, kuna Moodle'is on info tudengite tööde ja esitatud

failide kohta juba olemas.

Kuna Moodle'i integratsiooni arendus jääb käesoleva lõputöö mahust välja, ei olnud selle täielik realiseerimine veel võimalik. Peale lõputöö kaitsmist on plaanis töötada välja vajalikud liidesed ning tuua CodeGraderi kasutajaliides Moodle'isse, et saavutada tõeliselt sujuv ja automatiseeritud hindamisprotsess.

# 5 Töö tulemuste analüüs

AI-põhist hindamissüsteemi rakendati, et hinnata 2025. aasta kevadsemestril ITI0202 Programmeerimise põhikursuse suuremaid projektitöid. Projekt oli jaotatud kolme ossa. Käesolevas analüüsis käsitletakse neist kahte esimest. Iga tudengi projekti kahes esimeses osas hindasid nii AI-mudel kui ka kaks sõltumatut inimhindajat. Analüüsi eesmärgiks oli võrrelda AI poolt antud punktisummasid inimhindajate hinnangutega ning hinnata, kui usaldusväärselt AI suudab reaalset hindamist toetada.

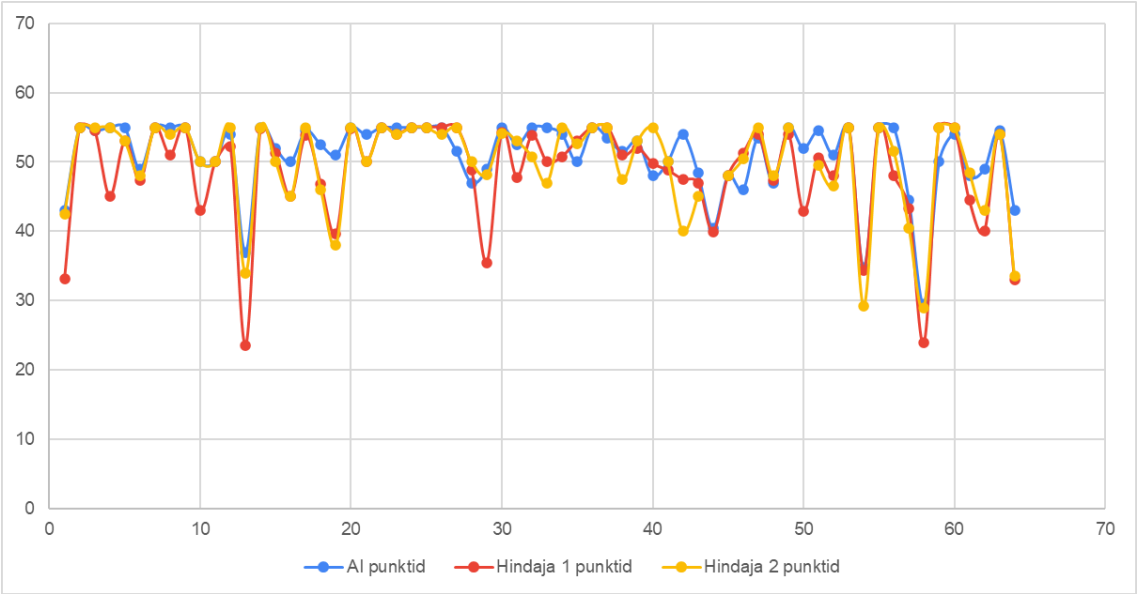
Analüüsiks koostati tulemuste võrdlustabelid ja joonised, mille alusel arvutati erinevad statistilised näitajad (keskmised, absoluutvahed jm). Hindajate võimalike subjektiivsete erinevuste vähendamiseks kasutati nende antud punktide aritmeetilist keskmist ("hindajate keskmine").

## 5.1 Tulemuste võrdlus

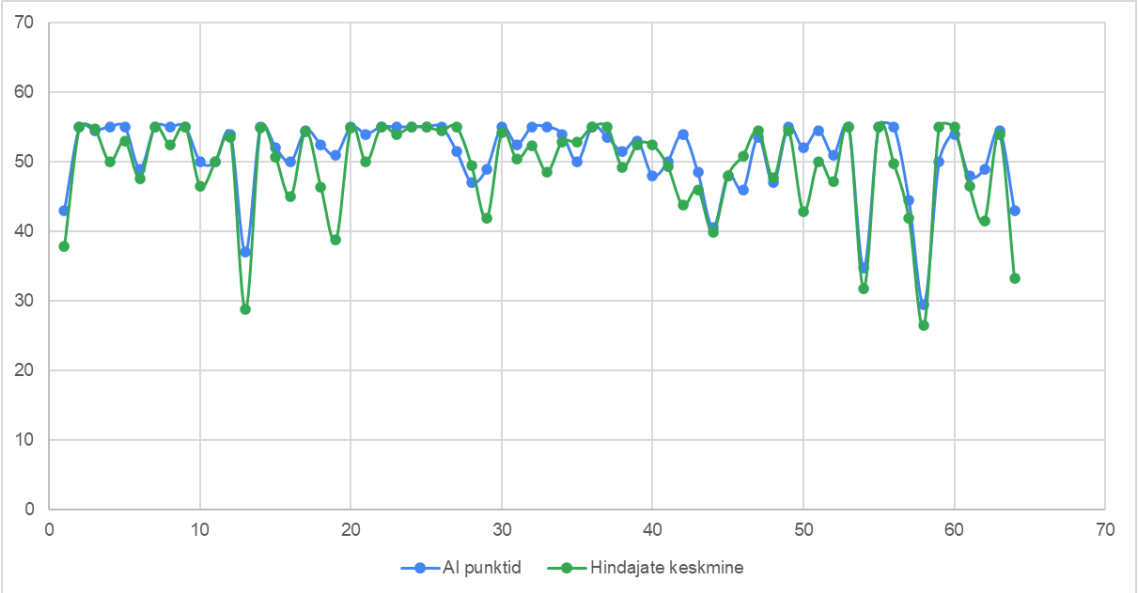
Projekti esimese osa hindamisel analüüsiti kokku 64 tööd. Jooniselt 4 ja jooniselt 5 on näha, et AI poolt antud punktisummad kattuvad enamasti hästi inimhindajate antud tulemustega, kuid üksikutes töödes on märgata suuremaid erinevusi (nt indeksitega 13, 19, 29, 42 ja 64 puhul erineb AI tulemus hindajate keskmisest rohkem kui 8 punkti). Samas valdav osa töid on hinnatud AI poolt kas täpselt sama skooriga või väikse, alla 1-punktise erinevusega. Projekti esimese osa analüüsimisel ilmnes, et AI hinnangu ning hindajate keskmise hinnangu vahe oli keskmiselt 2,72 punkti. Võrdluses esimese hindajaga oli see vahe keskmiselt 3,07 punkti ning teise hindajaga 2,40 punkti.

Analüüsi käigus ilmnes, et kuigi AI ja inimhindajate skoorid ei lange alati täpselt kokku, on korrelatsioon nende vahel siiski tugev. Eriti tuleb see esile tööde puhul, mille sooritus on selgelt nõrgem – kui töö on halvemini kirjutatud või ei vasta nõuetele, siis langevad punktid nii AI kui inimhindajate poolt märgatavalt. See näitab, et AI suudab nõrkade ja tugevate tööde vahel vahet teha ning annab vähem punkte kehvemate tööde puhul, isegi kui täpsed punktisummad alati ei ühti.

Samas, korrektsete ja hästi kirjutatud tööde puhul annab AI sageli väga lähedased või isegi samad punktid nagu inimhindajad. See viitab sellele, et AI suudab edukalt tuvastada nõuetele vastavaid ja kvaliteetseid lahendusi.




Joonis 4. Projekti esimese osa AI, hindaja 1 ning hindaja 2 tulemuste võrdlus.



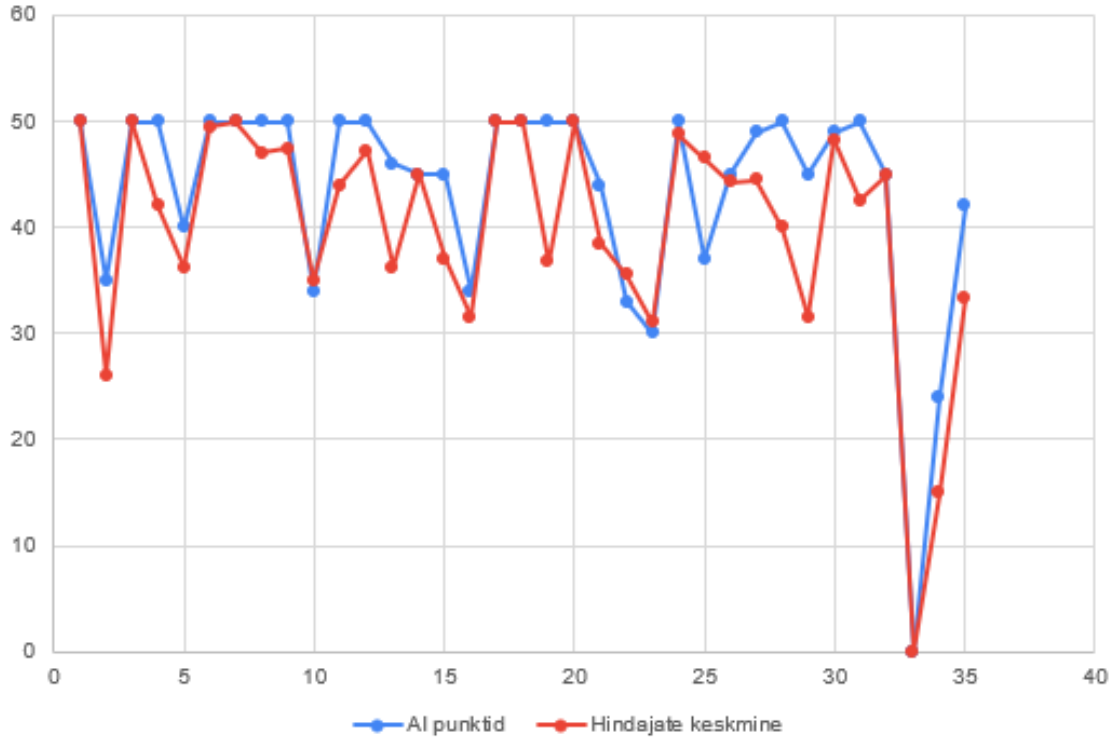

Joonis 5. Projekti esimese osa AI ja mõlema hindaja keskmiste tulemuste võrdlus.

Teise osa hindamisel analüüsiti kokku 35 tööd. Sarnaselt esimese osaga võrreldi AI-mudeli antud punktisummasid kahe sõltumatu inimhindaja omadega ning nende keskmisega. Teise osa puhul oli AI hinnangu ning hindajate keskmise vahe keskmiselt 4,13 punkti. AI hinnangu ning esimese hindaja vahe oli keskmiselt 5,54 punkti, samas kui teise hindajaga oli see vahe 3,78 punkti.

Projekti teise osa tulemused on visualiseeritud joonistel 6 6 ja 7 7. Sarnaselt esimese osaga näitavad ka need joonised, et kuigi esineb üksikuid suuremaid erinevusi, kattuvad

AI antud punktid üldjoontes inimhindajate omadega. Erinevused on mõnevõrra suuremad kui esimese osa puhul, kuid üldine trend, kus AI suudab eristada tugevamaid ja nõrgemaid sooritusi, jääb kehtima.


Joonis 6. Projekti teise osa AI, hindaja 1 ning hindaja 2 tulemuste võrdlus.




Joonis 7. Projekti teise osa AI ja mõlema hindaja keskmiste tulemuste võrdlus.

Projekti kolmanda osa kohta käesoleva analüüsi raames pädevaid kvantitatiivseid tulemusi esitada ei olnud võimalik. Analüüsi teostamise hetkeks polnud õppejõud veel jõudnud kolmanda osa töid lõplikult hinnata, mistõttu puudusid piisavad andmed, millega AI tulemusi detailselt võrrelda ja statistiliselt analüüsida.

## 5.2 Järeldused

Käesolevas töös analüüsitud projekti kahe osa tulemused näitavad, et kuigi AI hindamine erineb mõnel juhul inimhindajate omadest, on see eriti hästi kooskõlas tööde puhul, mis on kas väga head või väga halvad – seega suudab AI piisavalt täpselt eristada tööde kvaliteeti. Suuremad erinevused (esimeses osas keskmiselt 2,72 punkti ja teises osas 4,13 punkti võrreldes hindajate keskmisega) esinevad eeskätt tööde puhul, mis jäävad "hallile alale" või kus on subjektiivsust hinnangutes rohkem. Seetõttu on oluline, et AI poolt madalaid tulemusi saanud tööd vaadataks õppejõu poolt täiendavalt üle, kuid heade tööde puhul võib AI tulemust pidada usaldusväärseks abivahendiks, mis aitab hindamisprotsessi kiirendada ning õppejõule esmast indikatsiooni anda. Kolmanda osa kohta detailsete andmete puudumise tõttu ei saa kogu projekti hindamise lõikes veel lõplikke järeldusi teha, kuid kahe esimese osa analüüs annab siiski lootustandva pildi rakenduse CodeGrader potentsiaalist hindamisprotsessi toetamisel.

## 5.3 Õppejõudude esmane tagasiside ja tähelepanekud

Lisaks eespool esitatud kvantitatiivsele analüüsile koguti CodeGrader rakenduse esmase testimise käigus ka tagasisidet ITI0202 Programmeerimise põhikursuse õppejõududelt, kes rakendusega tutvusid ja selle potentsiaali hindasid.

Õppejõudude üldine hoiak loodud hindamisassistendi suhtes oli positiivne. Esile toodi eelkõige rakenduse potentsiaalset ajalist võitu, mis kaasneks selle kasutuselevõtuga mahukate programmeerimistööde hindamisel. Märgiti, et tööriist on tõepoolest kasulik ja võiks aidata vähendada hindamisele kuluvat aega, võimaldades õppejõududel kiiremini esmase ülevaate saada tudengite töödest.

Teise olulise aspektina mainiti, et CodeGrader aitab suunata tähelepanu just nendele tudengitööde osadele või konkreetsetele nõuetele, millega tudengitel enim probleeme esines. See võimaldaks õppejõududel efektiivsemalt tuvastada peamised murekohad ja anda seeläbi fokusseeritumat tagasisidet.

Tulevikuperspektiivi silmas pidades väljendasid õppejõud selget soovi ja nägid praktilist

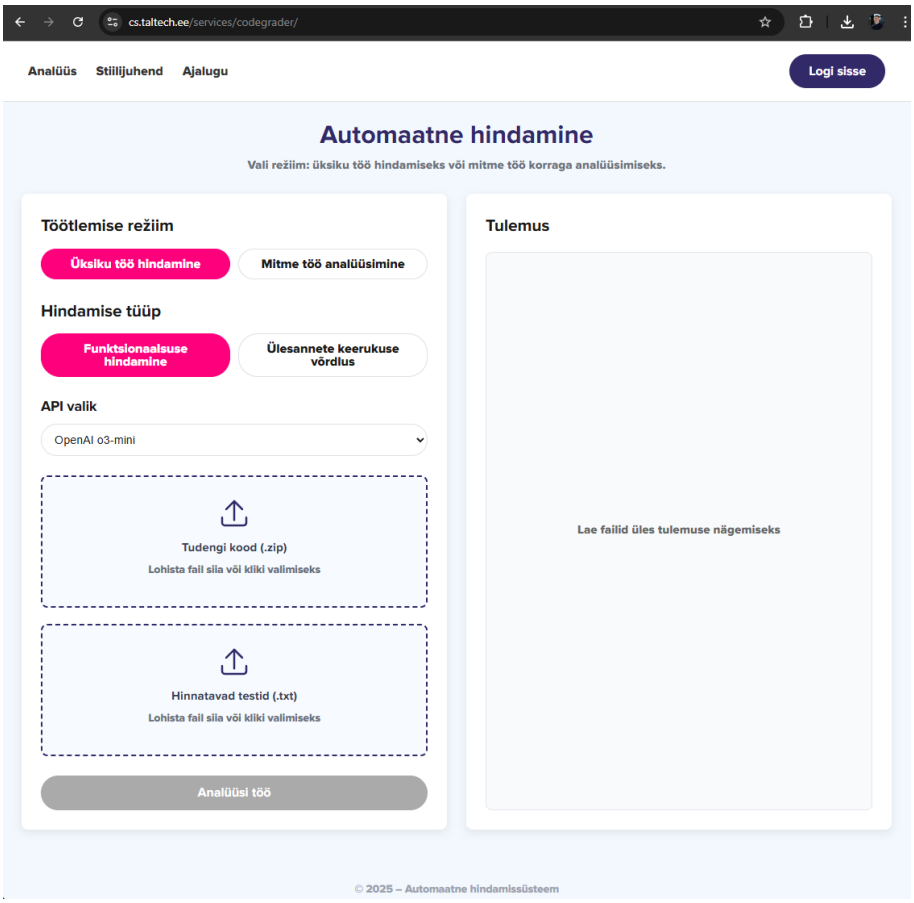
vajadust CodeGraderi integreerimiseks TalTechi Moodle'i õpikeskkonnaga. Leiti, et selline integratsioon muudaks rakenduse kasutamise veelgi mugavamaks ja kättesaadavamaks ning lihtsustaks tööde hindamise töövoogu tervikuna.

# 6 Rakenduse kasutamine

Käesolev peatükk juhendab kasutajat loodud CodeGraderi hindamisassistendi praktilisel kasutamisel. Peatükis kirjeldatakse samm-sammult kogu peamist töövoogu, alustades rakenduse esilehest ja sisselogimisest TalTechi GitLab konto kaudu, liikudes edasi tudengitööde ja nõuetefailide üleslaadimise juurde ning lõpetades analüüsi käivitamise, tulemuste kuvamise ja haldamisega. Lisaks antakse ülevaade, kuidas kasutada funktsionaalsusnõuete stiilijuhendit ja sirvida tehtud analüüside ajalugu.

## 6.1 Esileht ja sisselogimine

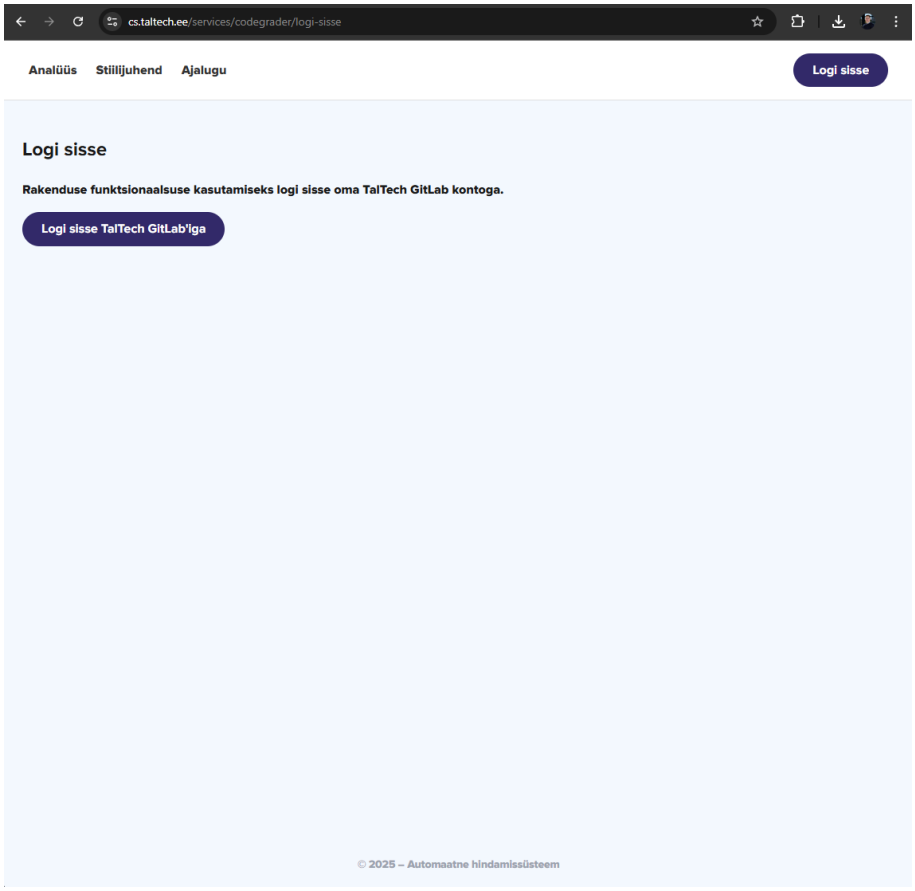
Hetkel asub veebileht aadressil https://cs.taltech.ee/services/codegrader/ [1]. Veebilehele minnes ilmub esimesena analüüsimiseks mõeldud leht.



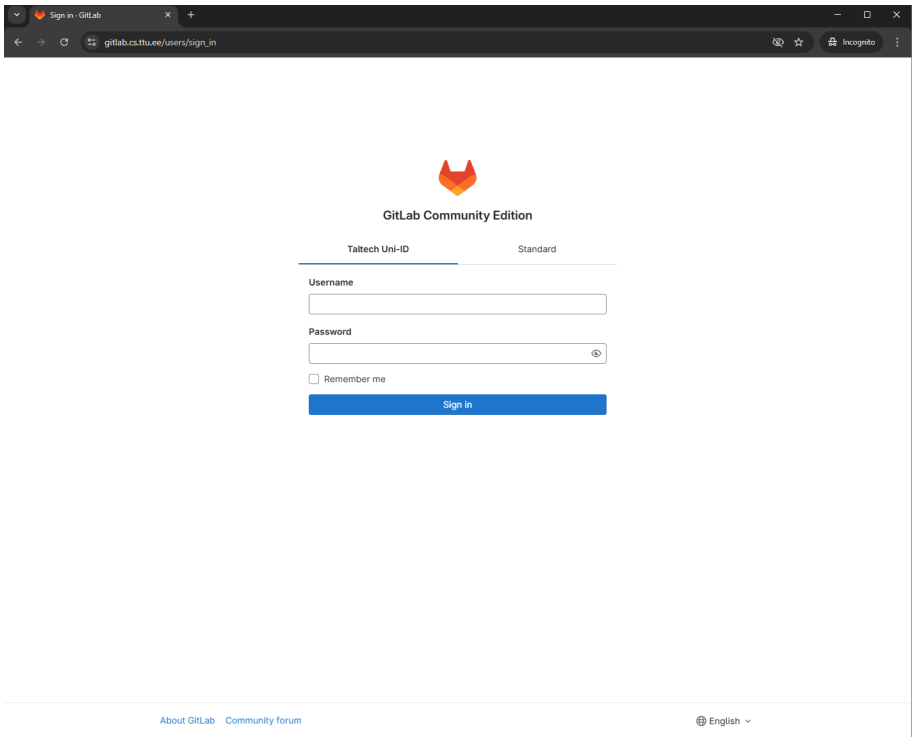

Joonis 8. CodeGraderi esileht.

[1] Lõputöö raames loodud CodeGrader rakenduse veebileht.

Lehe kasutamiseks peab esmalt sisse logima üleval paremal olevast "Logi sisse" nupust. Kui kasutaja veebilehitsejas on varasemalt GitLabi kaudu sisse logitud, siis viiakse kasutaja koheselt tagasi esilehele. Vastasel juhul viiakse kasutaja sisselogimisnupule vajutades edasi TalTech GitLabi sisselogimislehele, kus saab ülikooli poolt antud kasutajaga sisse logida.



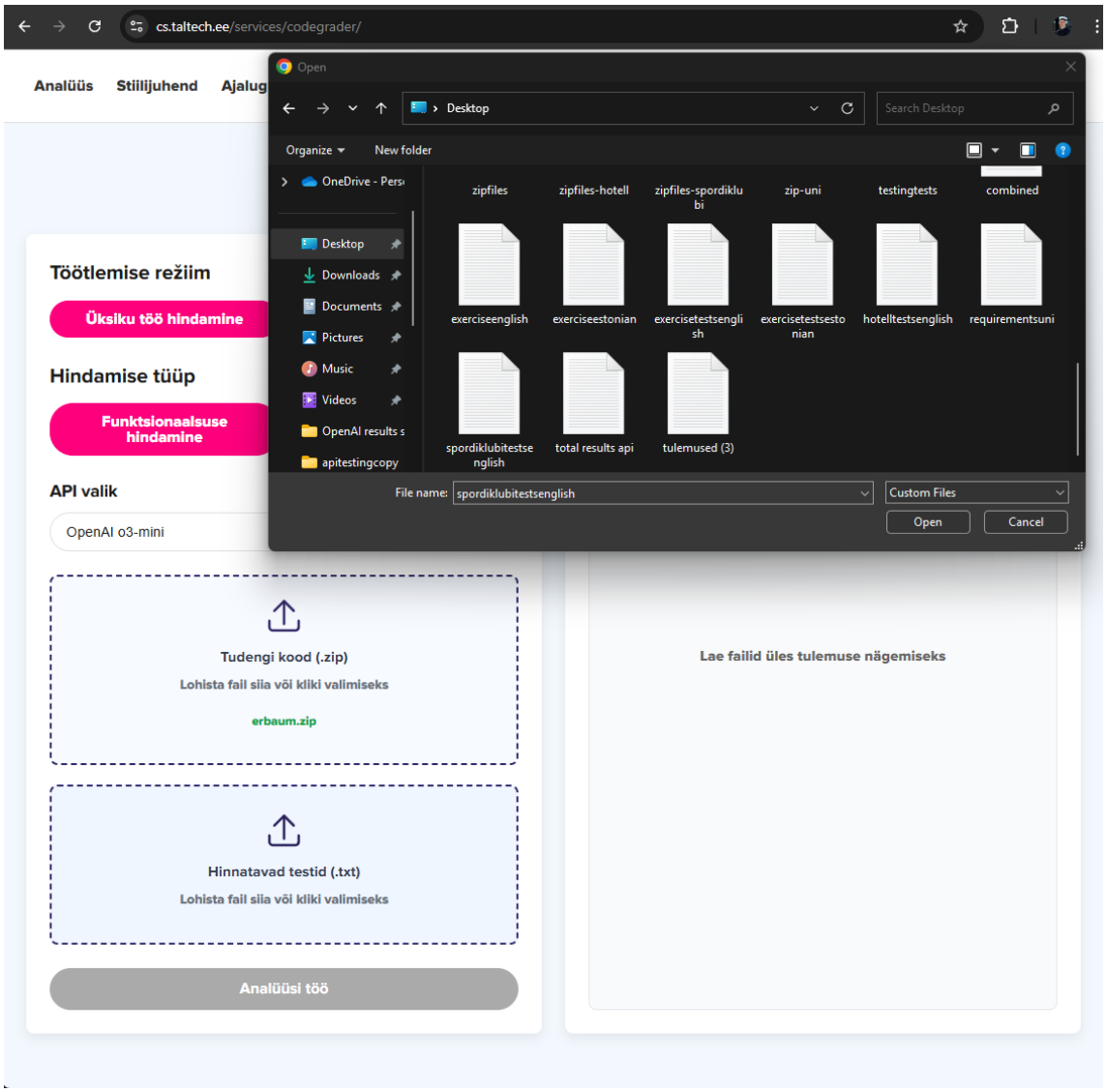

Joonis 9. CodeGraderi sisselogimisleht.




Joonis 10. TalTech GitLabi sisselogimisleht.

## 6.2 Failide üleslaadimine

Vasakus kastis olevatele nuppudele "Tudengi koodi (.zip)" ning "Hinnatavad testid (.txt)" peale vajutades avaneb arvutist failide ülesse laadimise võimalus. Alternatiivselt saab faile enda arvutist nende kastide peale samuti lohistada, et faile üles laadida.

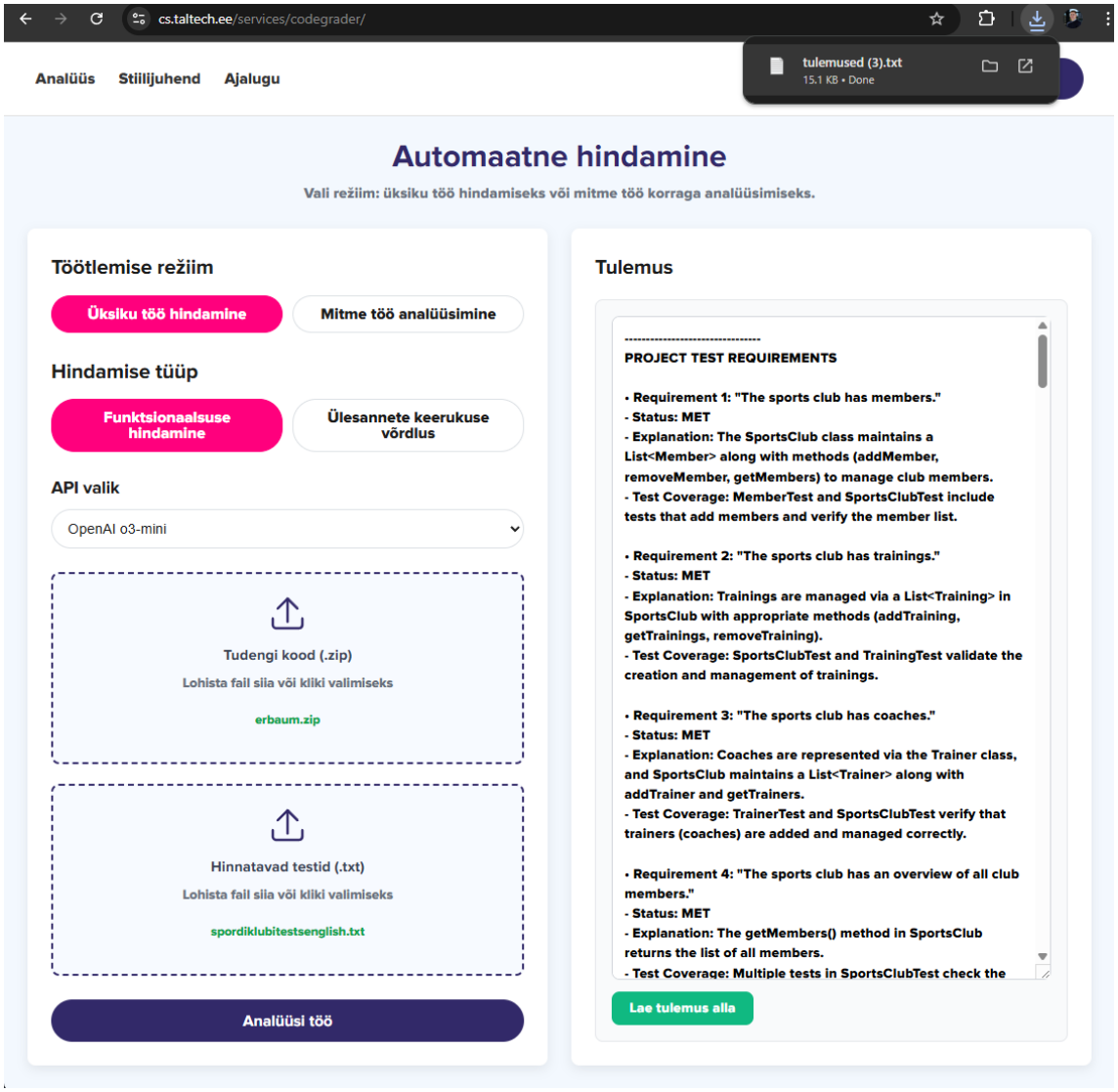



Joonis 11. Failide üleslaadimine.

Kasutaja saab valida, kas analüüsida üksikut tööd või mitut tööd vajutades nupule "Üksiku töö hindamine" või "Mitme töö analüüsimine" ning ise valida erinevate mudelite vahel vajutades "API valik" nupule ning sealt valides soovitud mudeli. Kui fail on edukalt üles laetud, siis tekib roheline failinimi, mis indikeerib, et faili üleslaadimine õnnestus. Peale mõlema faili lisamist muutub aktiivseks "Analüüsi töö" nupp, millega saab failid saata analüüsimiseks API-le.

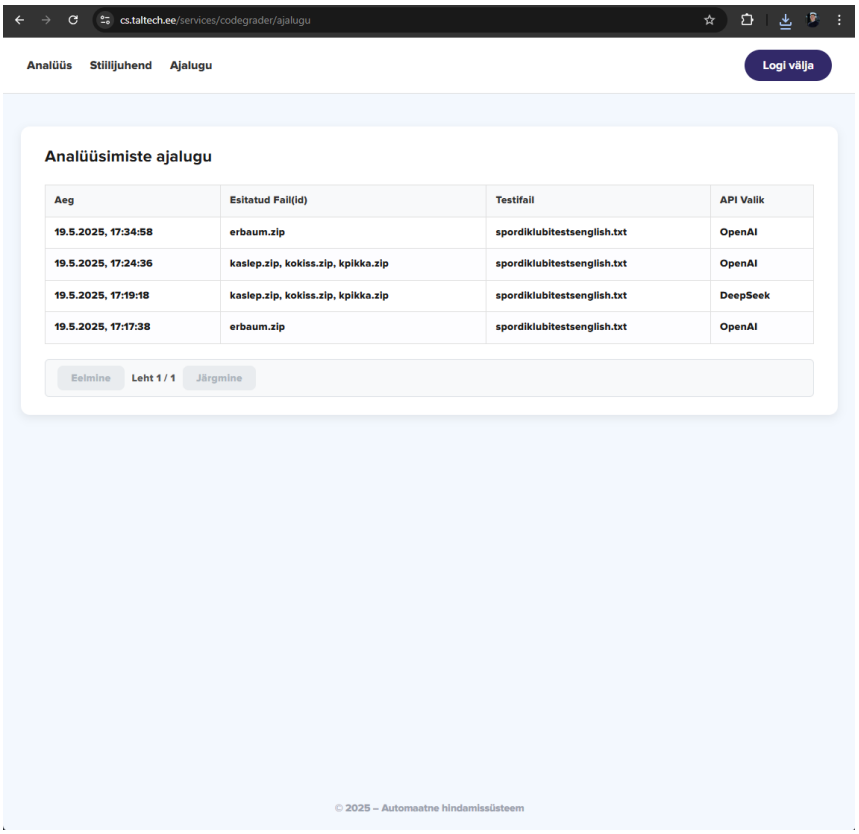
## 6.3 Tulemused, stiilijuhend ja ajalugu

Kui päringu tulemused jõuavad API-lt serverisse tagasi, siis kuvab veebileht tulemused parempoolses kastis, kuhu tekib lisaks ka nupp "Lae tulemus alla". Nupule vajutades alustatakse kasutajale automaatne faili allalaadimine.

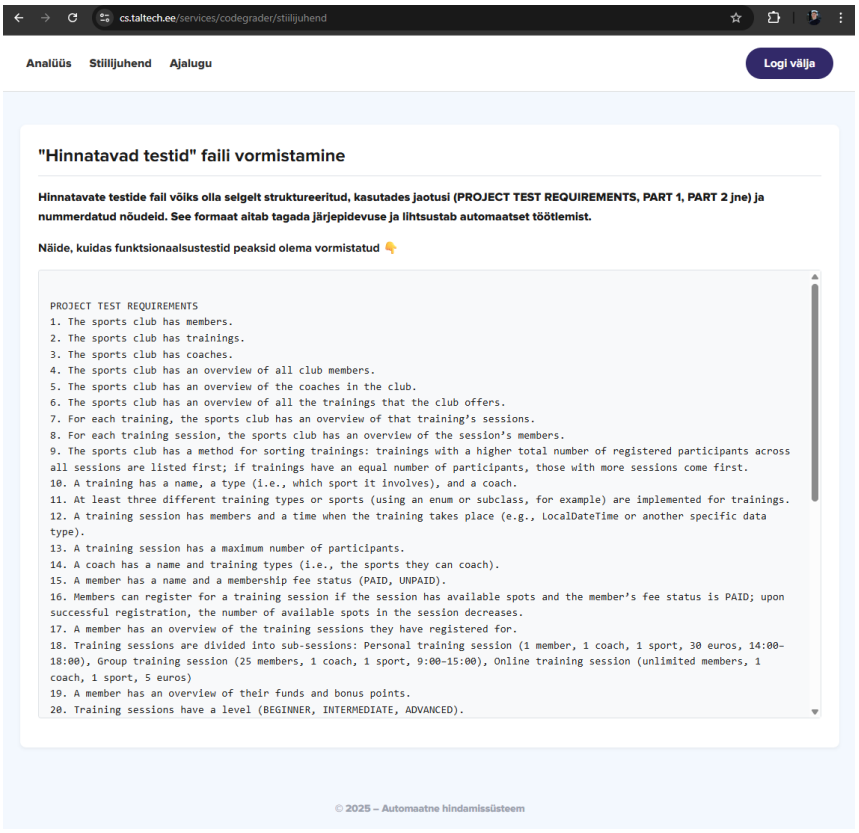



Joonis 12. LLM-i analüüsi tulemused ja nende allalaadimine.

Kõikide analüüsitud tööde ajalugu saab vaadata vajutades üleval olevale nupule "Ajalugu". Seal on näha, millise API-ga, mis kell ja mis faile analüüsiti. Lisaks kui kasutaja ei tea, kuidas funktsionaalsusnõuete faili koostada, siis on olemas ka üleval vasakul olevale nupule "Stiilijuhend" vajutades võimalus vaadata stiilinäidet vormistusest.




Joonis 13. Tulemuste ajalugu.




Joonis 14. Funktsionaalsusnõuete stiilijuhend.

# 7 Kokkuvõte

Bakalaureusetöö peamiseks eesmärgiks oli lahendada infotehnoloogiaõppekavadel, eriti TalTechi ITI0202 "Programmeerimise põhikursusel", esinev väljakutse seoses tudengite programmeerimistööde mahuka ja aeganõudva hindamisprotsessiga. Eesmärk oli luua suurtele keelemudelitele toetuv automatiseeritud hindamisassistent "CodeGrader", mis analüüsiks tudengite esitatud koode ja teste, vähendaks õppejõudude töökoormust ning aitaks pakkuda õpilastele senisest põhjalikumat ja struktureeritumat tagasisidet.

Funktsionaalse hindamisassistendi loomine saavutati mitmeetapilise protsessi kaudu. Töö algas põhjaliku taustauuringuga, mille käigus analüüsiti traditsioonilisi hindamismeetodeid ja uuriti LLM-ide potentsiaali ning piiranguid, millele järgnes nõuete analüüs. Praktilise töö käigus arendati välja veebipõhine rakendus CodeGrader, mille tagarakendus loodi Java keeles Spring Boot raamistikku kasutades ning kasutajaliides loodi TypeScripti ja Reactiga. Rakendus võimaldab tudengitööde ja nõuetefailide üleslaadimist, erinevate LLM-ide valikut analüüsiks, ning toetab nii koodi funktsionaalsuse hindamist kui ka ülesannete keerukuse võrdlust. Kasutajate autentimine lahendati TalTech GitLab OAuth2 kaudu ning rakenduse paigaldamiseks ja haldamiseks kasutati Dockerit ja Docker Compose'i.

Kokkuvõtvalt võib öelda, et käesoleva lõputöö tulemusena valmis funktsionaalne ja praktilist väärtust omav LLM-põhine hindamisassistendi prototüüp CodeGrader. Loodud süsteem täidab püstitatud põhilised eesmärgid, demonstreerides edukalt suurte keelemudelite rakendatavust programmeerimistööde hindamise toetamisel ning pakkudes alust edasisteks täiendusteks ja integreerimiseks õppeprotsessi. Kuigi LLM-ide genereeritud hinnangud vajavad jätkuvalt õppejõu poolt ülevaatamist, on loodud lahendus oluline samm efektiivsemat ja kvaliteetsemat tagasisidet võimaldava hindamispraktika suunas.

# Kasutatud kirjandus


[1] *ITI0202 „Programmeerimise põhikursus"*. TalTech õppeaine infoleht. Tallinna Tehnikaülikool. 2024. URL: https://ois2.taltech.ee/uusois/aine/ITI0202 Kasutatud: 1.5.2025.

[2] Google Developers. *Introduction to Large Language Models*. Google Developers. 2024. URL: https://developers.google.com/machine-learning/resources/intro-llms Kasutatud: 2.5.2025.

[3] Staphord Bengesi et al. "Advancements in Generative AI: A Comprehensive Review of GANs, GPT, Autoencoders, Diffusion Model, and Transformers". In: *arXiv preprint arXiv:2311.10242* (2023). URL: https://arxiv.org/abs/2311.10242.

[4] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. arXiv preprint arXiv:1706.03762. 2017. URL: https://arxiv.org/abs/1706.03762.

[5] Google Cloud. *Gemini 2.0 Flash*. Google Cloud Vertex AI. 2025. URL: https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash Kasutatud: 16.5.2025.

[6] DeepSeek. *DeepSeek API Pricing*. DeepSeek API Docs. 2025. URL: https://api-docs.deepseek.com/quick_start/pricing Kasutatud: 17.5.2025.

[7] Anthropic. *Anthropic API Pricing*. Anthropic Official Website. 2025. URL: https://www.anthropic.com/pricing#api Kasutatud: 18.5.2025.

[8] OpenAI. *OpenAI o3-mini Model*. OpenAI API Docs. 2025. URL: https://platform.openai.com/docs/models/o3-mini Kasutatud: 1.5.2025.

[9] xAI. *Grok-3 Beta Model*. xAI Console. 2025. URL: https://console.x.ai/team/ec5a2067-ae8d-419f-a67f-35590887e828/models?modelName=grok-3-beta Kasutatud: 2.5.2025.

[10] GitLab Documentation. *OAuth2 Provider*. GitLab Docs. 2024. URL: https://docs.gitlab.com/integration/oauth_provider/ Kasutatud: 3.5.2025.

[11] GeeksforGeeks. *Introduction to Java*. GeeksforGeeks. 2025. URL: https://www.geeksforgeeks.org/introduction-to-java/ Kasutatud: 5.5.2025.

[12] Oracle Corporation. *The Java Virtual Machine Specification*. Oracle Help Center. 2024. URL: https://docs.oracle.com/javase/specs/jvms/se22/html/ Kasutatud: 18.5.2025.

[13] Spring Team. *Spring Boot Documentation*. Spring Official Docs. 2024. URL: https://docs.spring.io/spring-boot/ Kasutatud: 4.5.2025.

[14] Spring Team. *Web Servers - Spring Boot*. Spring Boot Documentation - Covers Tomcat, Jetty, and Undertow. 2024. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/web.html#web.servlet.embedded-web-servers Kasutatud: 16.5.2025.

[15] GeeksforGeeks. *REST API Introduction*. GeeksforGeeks. 2025. URL: https://www.geeksforgeeks.org/rest-api-introduction/ Kasutatud: 6.5.2025.

[16] Gradle Inc. *Gradle Build Tool*. Gradle Official Website. 2025. URL: https://gradle.org/ Kasutatud: 7.5.2025.

[17] TypeScript Team. *TypeScript Documentation*. TypeScript Official Docs. 2025. URL: https://www.typescriptlang.org/docs/ Kasutatud: 8.5.2025.

[18] MDN Web Docs. *JavaScript*. Mozilla Developer Network. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript Kasutatud: 13.5.2025.

[19] React Team. *React Documentation*. React Official Docs. 2025. URL: https://react.dev/ Kasutatud: 9.5.2025.

[20] React Team. *ReactDOM – React*. React Official Docs. 2025. URL: https://legacy.reactjs.org/docs/react-dom.html Kasutatud: 10.5.2025.

[21] Vite Team. *Vite – Next Generation Frontend Tooling*. Vite Official Website. 2025. URL: https://vite.dev/ Kasutatud: 11.5.2025.

[22] MDN Web Docs. *JavaScript modules*. Mozilla Developer Network. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules Kasutatud: 14.5.2025.

[23] Amazon Web Services. *AWS Documentation*. AWS Official Docs. 2025. URL: https://docs.aws.amazon.com/ Kasutatud: 12.5.2025.

[24] Docker Inc. *Docker Documentation*. Docker Official Docs. 2025. URL: https://docs.docker.com/ Kasutatud: 13.5.2025.

[25] Docker Inc. *Dockerfile reference*. Docker Documentation. 2024. URL: https://docs.docker.com/engine/reference/builder/ Kasutatud: 16.5.2025.

[26] Docker Inc. *Docker Compose Documentation*. Docker Official Docs. 2025. URL: https://docs.docker.com/compose/ Kasutatud: 14.5.2025.

[27] Red Hat. *What is CI/CD?* Red Hat Official Website. 2025. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd Kasutatud: 15.5.2025.

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

Mina, Karl-Uljas Mölder

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Tudengite programmeerimiseülesannete automaatne hindamine suurte keelemudelite abil", 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.

04.06.2025

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

# Lisa 2 – Suurtele keelemudelitele saadetav viip

"You are an experienced and meticulous computer science professor tasked with evaluating a student's programming submission. You will be provided with two key pieces of information:

1. **Project Test Requirements:** A document listing all functional and non-functional requirements the student's submission must meet. These requirements may be presented in one or more sections (e.g., "PART 1", "PART 2", or simply as a single list). 2. **Student's Submission:** The complete source code and any accompanying test files submitted by the student.

Your evaluation must be thorough and adhere strictly to the following output format for *each requirement* found in the "Project Test Requirements" document:

———————— IF A SECTION TITLE IS PRESENT IN THE REQUIREMENTS, INSERT IT HERE, e.g., "PROJECT TEST REQUIREMENTS – PART 1" or "MAIN FUNCTIONALITY REQUIREMENTS". IF NO SECTION TITLES ARE PRESENT, OMIT THIS LINE AND THE FOLLOWING BLANK LINE.

IF A SECTION TITLE WAS INSERTED ABOVE, ADD A BLANK LINE HERE. OTHERWISE, OMIT THIS LINE. • Requirement X: "<The exact text of Requirement X as stated in the P̈roject Test Requirementsd̈ocument>" – Status: <MET / NOT MET / PARTIALLY MET> – Explanation: <Provide a concise and specific explanation for the status. If NOT MET or PARTIALLY MET, clearly state what is missing or incorrect in the student's submission. Refer to specific code elements if possible.> – Test Coverage: <Detail if and how the student's own tests (if any) cover this specific requirement. If no student tests cover it, or if student tests are not provided/relevant, state N̈o specific student tests for this requirement.ör T̈est coverage not assessed by student tests.s̈>

———————— IF ANOTHER SECTION TITLE FOLLOWS IN THE REQUIREMENTS, INSERT IT HERE. IF NO FURTHER SECTION TITLES, OR IF IT'S THE START OF THE NEXT REQUIREMENT IN A SINGLE LIST, OMIT THIS LINE AND THE FOLLOWING BLANK LINE.

IF A SECTION TITLE WAS INSERTED ABOVE, ADD A BLANK LINE HERE. OTH-

ERWISE, OMIT THIS LINE. • Requirement Y: "<The exact text of Requirement Y...>" – Status: <...> – Explanation: <...> – Test Coverage: <...>

... continue this pattern for all requirements listed in the "Project Test Requirements" document.

**General Guidelines:** * **Requirement Identification:** Address each numbered requirement from the "Project Test Requirements" document sequentially. * **Section Handling:** * If the "Project Test Requirements" document uses explicit section titles (e.g., "PART 1", "PART 2", "PROJECT TEST REQUIREMENTS"), reproduce these titles exactly as they appear, each preceded by a "————————————" line and followed by a blank line before the first requirement of that section. * If the requirements are presented as a single, continuous list without explicit part/section titles (other than a main overall title like "PROJECT TEST REQUIREMENTS"), then only use one "————————————" line at the very beginning of the evaluation (or before the first requirement if a main title is also included). Do *not* invent part numbers or section titles. * **Exact Formatting:** Precisely match the bullet points (•), en dashes (–), indentation (using spaces, not tabs, aiming for 2-4 spaces for the sub-items), and spacing as shown in the format example above for each requirement's evaluation. * **No Extra Commentary:** Do not add any introductory or concluding remarks beyond the structured evaluation of requirements and the final summary. * **Clarity and Conciseness:** Explanations should be direct and to the point.

**Final Summary:** After evaluating all individual requirements, conclude with a "Summary" section formatted as follows:

———————————— Summary: <Provide a brief overall recap of the student's performance, highlighting major strengths or persistent weaknesses observed across the requirements. For example: "The student successfully implemented most core functionalities but struggled with advanced features and edge cases. Test coverage was generally good for basic requirements." > ————————————

Begin your evaluation now based on the provided "Project Test Requirements" and "Student's Submission"."