Addis Ababa Science and Technology University

College of Engineering

Department of Software Engineering

Automated Code Review E-Learning System

Addis Ababa Science and Technology University

College of Engineering

Department of Software Engineering

Title: Automated Code Review E-Learning System

Group Members

No.	Name	ID Number
1	Agumas Desalew	ETS 1559/13
2	Dechasa Teshome	ETS 0373/13
3	Elsabeth Zeleke	ETS 0417/13
4	Eshetu Tesema	ETS 0435/13
5	Eyouel Melkamu	ETS 0479/13

Advisor Name: Mr. Yaynshet Signature:

May 10, 2025

Acknowledgement

We would like to sincerely thank everyone who supported us during our capstone project. First, we are very grateful to our advisor Mr. Yayneshet for guiding us and giving us valuable advice and support throughout our project. Next, we thank our teachers and professors for their valuable guidance, support, and feedback, which helped us improve our project and keep us on the right track.

We also want to thank our parents for their constant encouragement and understanding. They have provided us with money, material and mental support to complete this paper project. Their support and belief in us were very important in helping us stay focused and motivated.

Table of Content

List of Tables

List of Figures

Abstract

Universities often face major challenges in evaluating coding assessments, including the time-consuming nature of manual grading, risks of academic dishonesty, and the lack of timely, personalized feedback. Traditional methods such as paper-based exams and group projects are inefficient and struggle to scale in large classes. This project presents an Automated Code Review E-Learning System that leverages Graph Neural Networks and AI-powered evaluation models to provide real-time, objective feedback on programming assignments. The system features a Laravel backend, a student- and teacher-friendly web interface, and an analytics dashboard to help educators monitor student progress across languages such as C++, Java, and Python.

By incorporating techniques like Abstract Syntax Trees, Control Flow Graphs, and Static Program Analysis, the system can evaluate code logic and structure without executing untrusted code. While existing tools like ECHO and DeepReview show the potential of ML-driven code feedback, they often fall short in handling diverse coding styles, ensuring explainability, or scaling to educational use. This study addresses those gaps by designing a lightweight, secure, and scalable system that balances automation with clarity. Code submissions are analyzed using Graph Neural Networks, then translated into human-readable feedback through lightweight language models. The system's development follows a structured process including data collection, model training, implementation, and user testing.

Ultimately, this work contributes to modernizing programming education by reducing the burden on educators, increasing assessment accuracy, and offering students actionable insights to improve their skills.

***Keywords**: Automated Code Review, Graph Neural Networks, Code Analysis, E-Learning*

Chapter One: Introduction

Coding assessments, including exams, assignments, and projects, are essential for evaluating software engineering students' programming proficiency and understanding of key concepts [1]. At our university, the current methods include paper-based coding exams, group-based projects, physical submissions via USB devices, and, in some cases, submissions through Telegram bots. These approaches face significant challenges, such as inefficiencies in submission and evaluation processes, difficulties in maintaining academic integrity, and limited mechanisms to evaluate individual contributions.

A key issue is the reliance on teachers to manually review each student's code, a time-consuming and labor-intensive process, especially with large submission volumes. Manual reviews often fail to comprehensively test diverse cases, leading to inconsistent evaluations. This limitation affects the delivery of objective and actionable feedback, which is vital for students' skill development.

To address these challenges, this project proposes an Automated Code Review E-Learning System, designed to evaluate coding exams, assignments, and projects. The system will leverage advanced technologies like AI and GNN-based machine learning models to automate code evaluation, ensuring accurate assessments while reducing manual effort. It will provide real-time feedback to students, helping them identify strengths and weaknesses in their programming and improve their skills. The system will also include an analytics dashboard for educators, enabling them to monitor student performance across programming languages like C++, Java, and Python and offer targeted guidance.

By integrating these functionalities, the system will enhance the efficiency, objectivity, and integrity of coding assessments. It will provide students with valuable, personalized feedback and empower educators with actionable insights, transforming how coding assessments are conducted and evaluated. This innovative approach offers a scalable solution for teachers and an enriched learning experience tailored to individual student needs.

1.1 Statement of the Problem

The current methods for conducting coding assessments, including exams, assignments, and projects, are inadequate in meeting the evolving needs of both students and educators. At our university, these assessments rely on paper-based exams, group-based projects, physical submissions via USB devices, and,

in some cases, Telegram bots. These approaches are not only outdated but also fraught with inefficiencies that hinder the assessment process.

For teachers, the reliance on manual reviews of coding submissions is time-consuming and labor-intensive. With a high volume of students, individual evaluations become impractical, especially when dealing with complex programming tasks. Group-based projects further exacerbate this issue by obscuring individual contributions, making it difficult to gauge each student's abilities. Moreover, manual assessments often lack the rigor needed to comprehensively evaluate diverse test cases, resulting in inconsistent and incomplete evaluations.

For students, the lack of timely and personalized feedback significantly impacts their learning outcomes. The current system does not offer real-time guidance or insights into their strengths and weaknesses. And this undermines the integrity of assessments and creates an unfair learning environment.

These challenges highlight the need for a more efficient, objective, and scalable solution. An Automated Code Review E-Learning System can address these problems by automating code evaluation processes, providing real-time feedback to students, and offering educators insights through an analytics dashboard. By integrating advanced technologies like AI and GNN-based machine learning models, the system can ensure fair assessments while reducing the manual burden on educators. This innovation will not only improve the integrity of coding assessments but also enhance the overall learning experience for students, ensuring they receive the guidance needed to develop their programming skills effectively.

1.2 Objectives

The objectives of this study are designed to guide the development of an efficient e-learning system that leverages AI for coding assessments, including exams, assignments, and projects. The system aims to address the challenges of manual evaluation, provide real-time feedback for students, and empower teachers with actionable insights to enhance the assessment process.

1.2.1 General Objective

To develop an e-learning system that employs a GNN-based machine learning model to automate the evaluation of coding submissions, deliver real-time, personalized feedback to students, and provide teachers with an analytics dashboard for monitoring and improving student performance.

1.2.2 Specific Objectives

- **Requirement Gathering**: Identify the key challenges in coding assessments by conducting surveys and interviews with students and educators. Gather both functional and non-functional requirements to define the scope and goals of the system.
- **Requirement Analysis**: Analyze the collected requirements to establish technical specifications, limitations, and educational objectives. Ensure alignment with user needs and the integration of advanced feedback and analytics capabilities.
- **System Design**: Create a comprehensive architectural framework for the system, including:
    - A user-friendly frontend for students and educators.
    - A scalable backend built with Laravel for robust API development.
    - Integration of AI-driven components for automated code review and real-time feedback.
    - An analytics dashboard for educators to monitor student performance across programming languages like C++, Java, and Python.
- **Implementation**: Build the system using modern technologies, including:
    - **Frontend Development**: Design an intuitive user interface with React to ensure seamless interaction for students and educators.
    - **Backend Development**: Implement robust services with Laravel to manage data processing, user authentication, and integration with AI models.
    - **AI Integration**: Incorporate a GNN-based machine learning model to automate code evaluation and generate detailed, actionable feedback.
- **Testing**: Ensure system reliability and usability through comprehensive testing.
    - Validate individual components with unit testing.
    - Conduct integration testing to verify seamless interaction between system modules.
    - Perform user acceptance testing (UAT) to assess usability and gather feedback from students and educators.
- **Deployment**: Launch the system in a controlled environment for pilot testing. Collect feedback from users, refine system functionality, and prepare for full-scale deployment across exams, assignments, and projects.
- **Analytics Dashboard Development**: Provide teachers with a dashboard that visualizes student performance, highlighting strengths and weaknesses in programming languages and identifying areas where intervention is needed.

- **User Training and Documentation**: Create training resources, including user guides and tutorials, to facilitate smooth adoption by students and teachers. Conduct workshops or training sessions to ensure users can maximize the system's potential.

1.3 Scope and Limitation

1.3.1. Scope

The scope of this project includes the development of a comprehensive e-learning platform designed to streamline coding assessments, including exams, assignments, and projects. The system will:

- Automate the evaluation of coding submissions using AI-driven code review mechanisms, such as GNN-based machine learning models.
- Provide real-time feedback to students to help them identify strengths, weaknesses, and actionable improvements in their coding skills.
- Include an analytics dashboard for educators to monitor individual and group performance across various programming languages, such as C++, Java, and Python, and identify areas where guidance is required.
- Support integration with a web client accessible by both students and educators, featuring a user-friendly interface for submitting, reviewing, and analyzing coding tasks.
- Facilitate personalized learning experiences by enabling educators to provide targeted interventions based on analytics insights.

1.3.2. Limitations

Despite the system meeting its core functional and non-functional objectives, several limitations were identified during development and testing:

- **Lack of Export Functionality**: The system currently does not support exporting grading results or performance reports to external formats such as PDF or CSV. This limits the ability of teachers and administrators to archive, print, or share grading data outside the platform for institutional reporting or manual review purposes.
- **Machine Learning Model Performance**: The machine learning component responsible for automated code evaluation exhibits suboptimal accuracy levels. Specifically, the model

achieves an accuracy of only 53% for C++, 52% for Python, and 51% for Java submissions. Additionally, the dataset used for training contains a limited number of examples for specific verdicts such as *Memory Limit Exceeded (MLE)*, leading to near-zero precision and recall for this class. Although MLE cases are often misclassified as *Time Limit Exceeded (TLE)*, which also reflects code inefficiency, this misclassification reduces the system's diagnostic granularity and affects grading reliability.

- **Limited Accessibility Compliance**: While the system includes a user-friendly interface, it currently lacks full compliance with the Web Content Accessibility Guidelines (WCAG) 2.1. This limitation may affect the usability of the platform for users with visual, auditory, or motor impairments, thereby restricting inclusive access and potentially reducing the system's effectiveness in diverse educational environments.

1.4. Methodology

This study will employ a mixed-methods approach combining qualitative and quantitative research techniques to ensure a comprehensive and robust development process for the e-learning system.

1.4.1. Research Design

The research will begin with a systematic review of existing literature on the application of artificial intelligence (AI) in education, with a particular focus on automated code review systems. This review will explore their impact on learning outcomes, identify best practices, and analyze existing challenges and gaps in current methodologies. The findings will provide a foundation for designing a solution that aligns with academic needs and technological advancements.

1.4.2. Data Collection Techniques

The machine learning component of the system relies heavily on training data to enable automated analysis and feedback on student code submissions. For this purpose, the IBM Project CodeNet dataset was selected due to its scale, diversity, and relevance to real-world programming challenges.

Project CodeNet is a large-scale, open-source dataset developed by IBM to facilitate advancements in AI for code. It contains approximately 14 million code samples and over 50 million lines of code, covering

more than 55 programming languages [A]. The dataset primarily consists of user-submitted solutions to competitive programming problems and includes rich metadata such as:

- Problem identifiers and descriptions
- Programming language specifications
- Submission outcomes (e.g., Accepted, Wrong Answer, Time Limit Exceeded, Memory Limit Exceeded)
- Execution details such as runtime and memory usage

This diversity allows the model to learn not only from correct solutions but also from a wide range of incorrect and inefficient submissions, making it especially valuable for tasks like code quality evaluation, error detection, and automated grading.

For this system, a subset of the dataset was curated, focusing on three widely-used languages: Python, C++, and Java. The training process prioritized submissions with clearly defined outcomes to support multi-class classification tasks related to verdict prediction..

1.4.3. Analysis Strategies

The system will integrate two primary analytical techniques to achieve its objectives:

- **Model Training**: A Graph Neural Network (GNN) model will serve as the core evaluation engine [2]. Given a problem statement and a corresponding code submission, it will predict the likelihood of various possible verdicts, such as Accepted, Wrong Answer, Runtime Error, Compile Error, Presentation Error, Time Limit Exceeded, and Memory Limit Exceeded, providing a probabilistic assessment of the submission's outcome.
- **Large Language Models (LLM):** LLM techniques will be employed to generate human-readable feedback based on the output of the evaluation model [3]. The predicted verdicts and their associated probabilities, produced by the GNN, will be passed to a large language model (specifically Gemini), which will translate the technical evaluation into clear, concise feedback. This feedback will summarize the code's performance, highlight strengths, identify areas for improvement, and provide actionable insights, serving both students and instructors for a better understanding of the submission.

1.4.4. System Design and Implementation

The system's development will follow a modular and scalable approach:

- **Backend Development**: A robust backend API will be built using Laravel to handle data processing, model integration, and communication between the system's components. The backend will also manage authentication, analytics data, and feedback generation.
- **Frontend Development**: React will be used to create a responsive and user-friendly interface, ensuring seamless interactions for both students and educators. The frontend will include features for real-time feedback visualization, submission management, and access to analytics dashboards.
- **Integration of Analytics Dashboard**: An analytics dashboard will be integrated into the system to allow educators to monitor student performance, track progress across different programming languages, and provide targeted guidance based on insights.

1.4.5. Technology Stack

To keep things organized, here's a table summarizing the tools and technologies we'll use

Component	Technology	Purpose
Frontend	React	Build a responsive, user-friendly interface for students and teachers.
Backend	Laravel (PHP)	Manage APIs, data processing, authentication, and AI integration.
AI Models	Python, PyTorch Geometric (for GNN), Hugging Face (for LLM)	Analyze code submissions and generate human-readable feedback.
Databases	PostgreSQL	Store user data, submission metadata, and code structures (ASTs).
Cloud Hosting	Render	Host frontend, backend, and databases for scalability and reliability.
Version Control	GitHub	Manage code versions and team collaboration.
Development Tools	Visual Studio Code, PyCharm	Support coding, debugging, and testing.
Testing Frameworks	Jest (React), PHPUnit (Laravel)	Perform unit and integration testing for frontend and backend.

Table 1. Tools and Technologies

1.4.6.. Testing Standards

The system will undergo rigorous testing at multiple stages to ensure its reliability, accuracy, and user satisfaction.

- **Unit Testing**: Each component, including the GNN model, API endpoints, and frontend modules, will be tested individually to ensure functionality.
- **Integration Testing**: The interaction between the backend and frontend components will be tested to verify seamless data exchange and communication.
- **User Acceptance Testing (UAT)**: teacher and students will participate in UAT to evaluate the system's usability and clarity of feedback. Feedback from this phase will guide refinements and optimizations prior to full deployment.

1.5 Plan of Activities

We're building our Automated Code Review E-Learning System Project from November 2024 to May 2025. The project is big, so we've broken it down into phases to keep things manageable and ensure we deliver a project that works well for students and teachers. Below, we've pointed out the key activities, milestones, and deliverables to guide us through the project, from start to finish. Our goal is to create an AI-powered platform that's user-friendly, scalable, and truly helps students with coding assessments.

1.5.1. Initiation and Analysis Phase

- **Talking to People:** We'll sit down with students to hear about their struggles with coding assessments.
    - Deliverable: A detailed document capturing everyone's needs and the system's goals.
- **Defining the Project:** We'll decide exactly what this system will cover—think programming languages like C++, Java, and Python, and features like an analytics dashboard. We'll also clarify what's not included to avoid scope creep.
    - Deliverable: A scope statement that spells out what's in and what's out.
- **How We Work:** We'll clearly define roles and the right tools to stay connected and productive. We use GitHub for code collaboration, and Trello to manage our tasks efficiently.
    - Deliverable:A fully equipped team with everything set up and ready to go.

- **Checking Out the Competition:** We'll dig into existing tools like ECHO and DeepReview, plus read up on the latest research to see what works, what doesn't, and where we can do better, especially with AI feedback and handling lots of users.
    - Deliverable: A report summarizing what we learned and gaps we can fill.

1.5.2. Planning Phase

- **Creating a Schedule:** We'll break the project into tasks, figure out who's doing what, and set deadlines. This includes everything from building the AI models to coding the website and testing it all.
    - Deliverable: A schedule showing all tasks and who's responsible.
- **Setting Milestones:** We'll mark key points to track our progress, like finishing the system design, training the AI, or launching a test version.
    - Deliverable: A list of milestones to keep us on track.
- **Making a Timeline:** We'll put together a Gantt chart to visualize the whole plan, tasks, milestones, and deadlines so everyone knows what's coming up.
    - Deliverable: A Gantt chart we can share with the team.

1.5.3. Design Phase

- **Software architecture and design:** We'll design a system with a React frontend for the interface, a Laravel backend for the heavy lifting, and AI components like Graph Neural Networks (GNNs) and language models. We'll make sure it can handle lots of users and has a dashboard for teachers.
    - Deliverable: A detailed architecture plan.
- **Database Setup:** We'll plan out our database PostgreSQL for user info and for storing code structures like Abstract Syntax Trees (ASTs). This keeps everything organized and quick to access.
    - Deliverable: A database design ready to go.
- **Data Preparation:** We'll gather code samples from GitHub. We'll organize this data so it's ready for the models.
    - Deliverable: A clean dataset and data model.
- **UI/UX design:** We'll create a user interface that's easy to use and accessible, following WCAG 2.1 guidelines. It'll have dashboards specially made for students and teachers, with clear ways to view feedback.
    - Deliverable: Wireframes and prototypes of the interface.

- **System security design:** We'll plan out security features like role-based access (so students can't mess with teacher settings), and multi-factor authentication to keep data safe.
    - Deliverable: A security plan for the system.

1.5.4. Development Phase

- **Building the Interface:** Using React, we'll create a smooth, responsive frontend that lets students submit code and teachers view analytics, all with a clean, intuitive design.
    - Deliverable: A working frontend.
- **Setting Up Databases:** We'll get PostgreSQL up and running, based on our designs, to store user data, code submissions, and AI results efficiently.
    - Deliverable: Fully set-up databases.
- **Coding the Backend:** With Laravel, we'll build the backend to manage users, process code submissions, and connect to our AI models. This includes creating APIs for smooth communication.
    - Deliverable: A functional backend with APIs.
- **Adding the AI:** We'll train our GNN to analyze code and our language model to turn that analysis into clear, helpful feedback. We'll test these to make sure they're accurate and fast.
    - Deliverable: Trained and integrated AI models.
- **Connecting Everything:** We'll link the frontend to the backend so data flows smoothly submitting code, getting feedback, and viewing dashboards.
    - Deliverable: A working prototype of the whole system.
- **Server Setup:** We'll configure a cloud server on Render to host everything, setting up APIs, authentication, and storage to handle lots of users.
    - Deliverable: A ready-to-go server.

1.5.5. Testing Phase

- **Checking Features:** We'll test every part code submission, feedback, grading to make sure it works perfectly for C++, Java, and Python.
    - Deliverable: A report on what works and what needs fixing.
- **Testing Performance:** We'll simulate lots of users to ensure the system can handle 100 people at once and give feedback in 5 seconds or less for most submissions.
    - Deliverable: A performance report.

- **Testing the Interface:** We'll get feedback from students and teachers to make sure the interface is clear, accessible, and user-friendly, tweaking it based on what they say.
    - Deliverable: A report on the interface and any improvements.
- **Testing Security:** We'll check our access controls, and authentication to make sure everything's locked down tight and safe from hacks.
    - Deliverable: A security report.

1.5.6. Deployment Phase

- **Setting Up for Launch:** We'll configure our servers, databases, and cloud storage on Render to handle real users, ensuring it's scalable and reliable.
    - Deliverable: A production-ready setup.
- **Pilot Launch:** We'll launch the system for a small group of users, letting them submit code and get feedback for a few courses to test it in action.
    - Deliverable: A live system.
- **Fixing Issues:** We'll keep an eye on the launch, quickly fixing any bugs or slowdowns to keep things running smoothly.
    - Deliverable: A log of any issues and how we fixed them.
- **Getting User Feedback:** We'll ask pilot users what they think to make sure the system meets their needs, making tweaks based on their input.
    - Deliverable: A feedback report from users.
- **Training Users:** We'll create guides and tutorials, plus hold training sessions, to help students and teachers use the system confidently.
    - Deliverable: A user manual and training materials.

1.5.7. Maintenance Phase

- **Monitoring Performance:** We'll watch how the system performs and listen to user feedback to spot any areas we can improve, like speeding up feedback or clarifying comments.
    - Deliverable: Regular reports on performance and feedback.
- **Fixing Bugs:** If users report issues, we'll jump on them quickly to keep the system reliable and users happy.
    - Deliverable: Fixes for any bugs or problems.

- **Adding New Features:** Based on feedback and new tech, we'll add things like support for more programming languages or better AI models to keep the system fresh.
    - Deliverable: Updates and new features.

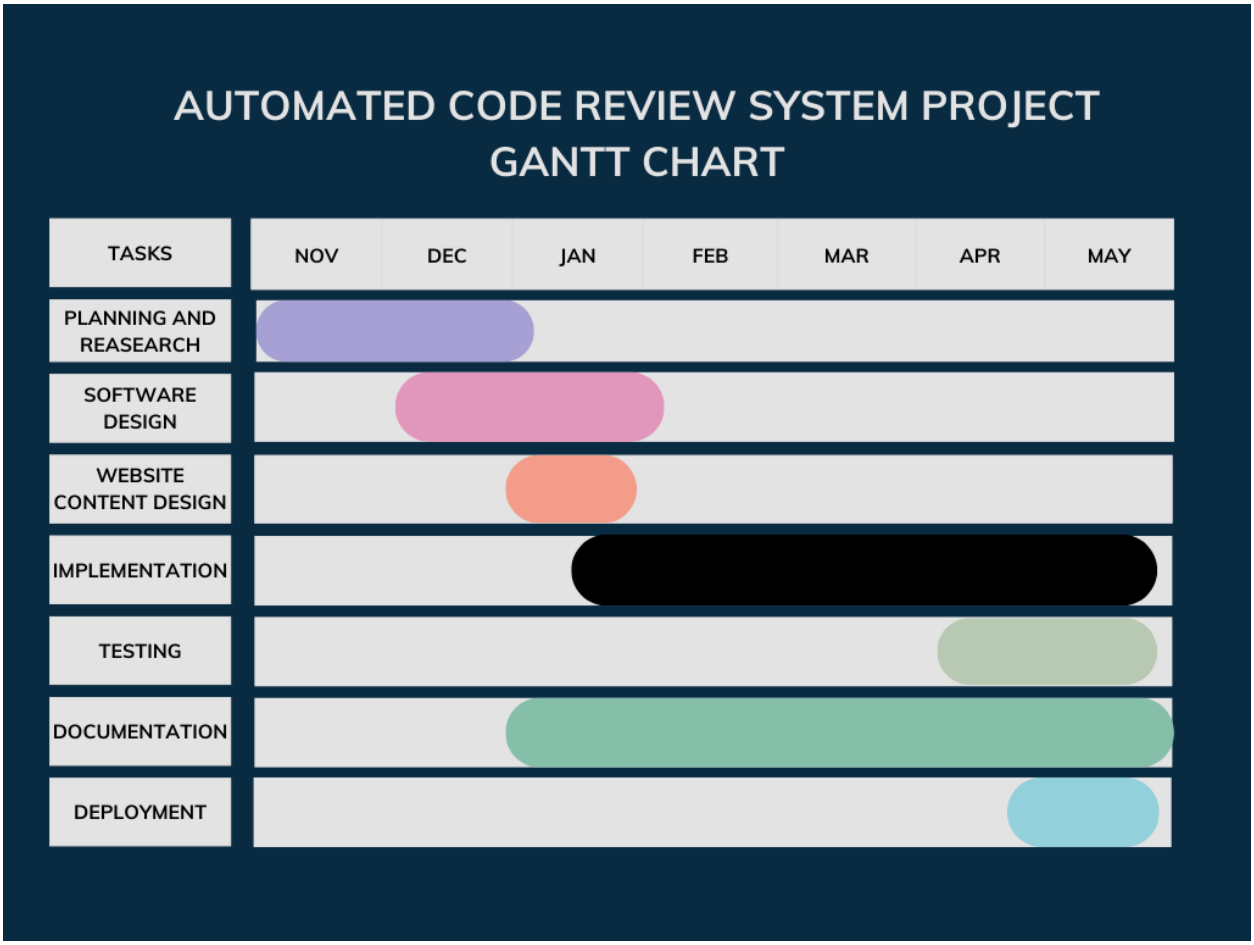

Figure 1. Gantt Chart

1.6. Budget Required

The project will require a budget to support the future deployment of a web-based platform aimed at enhancing coding assessments at Universities and colleges. This includes expenses for critical software tools, development resources, and infrastructure to ensure a scalable, AI-driven system. Specific costs will cover licensing fees for development machine learning libraries such as PyTorch for Graph Neural Networks and Hugging Face for Large Language Models. Additional expenses will include cloud hosting services through Render for website deployment, PostgreSQL database management, and server

maintenance, along with domain registration and security certificates to enable reliable and secure access for users.

1.7 Significance of the Study

This project presents an Automated Code Review E-Learning System that addresses critical inefficiencies in coding assessments, offering transformative benefits for software engineering education. By integrating Graph Neural Networks (GNNs) and Large Language Models (LLMs), the system delivers real-time, personalized feedback on code submissions in C++, Java, and Python, enabling students to promptly address errors in correctness, style, and best practices. This feedback enhances programming proficiency, fostering continuous skill development and improving learning outcomes, as supported by studies on coding education . Furthermore, the system promotes fairness by minimizing human bias through AI-driven evaluation, thus upholding academic integrity . The project's scalability is a key strength, designed to support a large number of concurrent users without performance degradation. This capability eliminates delays inherent in manual grading, making the system ideal for high-enrollment courses and streamlining assessment processes at Universities. Additionally, the system empowers teachers through an analytics dashboard that provides detailed insights into student performance, such as proficiency trends and common error patterns across programming languages. These data-driven insights enable targeted interventions, reducing administrative burdens and allowing teachers to focus on mentorship, thereby enhancing instructional quality. This study significantly advances educational technology by addressing gaps identified in prior work, such as feedback explainability and dataset diversity. By combining GNNs for code analysis with lightweight LLMs for human-readable feedback, and utilizing diverse datasets from GitHub and institutional repositories, the system sets a new standard for AI driven code review tools. Its methodologies and open-source datasets contribute to global research, inspiring further innovations in automated assessment systems across disciplines. Ultimately, this project transforms coding education by delivering scalable, equitable, and efficient assessments, benefiting students, teachers, and institutions while aligning with the growing demand for technology-driven educational solutions.

1.8 Outline of the study

This section offers a clear roadmap for the study, outlining the structure and purpose of each chapter in the documentation of the Automated Code Review E-Learning System, designed to address critical challenges in coding assessments at universities like Addis Ababa Science and Technology University. The introduction establishes the urgent need for improved assessment methods, highlighting the inefficiencies

of traditional approaches such as paper-based exams, group projects, USB submissions, and informal tools like Telegram bots. These methods suffer from time-consuming manual reviews, inconsistent feedback, which impede timely, personalized feedback essential for student skill development and burden educators, particularly in large classes. The proposed system, utilizing Graph Neural Networks (GNNs) to evaluate code submissions in C++, Java, and Python, combined with Large Language Models (LLMs) for real-time, actionable feedback, and supported by a Laravel-based backend, React-based frontend, and an analytics dashboard, aims to enhance fairness, efficiency, and educational outcomes for students and educators.

The study is organized into seven chapters, each contributing to the development and evaluation of this innovative platform. The first chapter, the introduction, defines the problem of manual grading, inconsistent evaluations, and academic dishonesty, presenting the project's general objective to develop an AI-driven e-learning system and specific objectives, including requirement gathering, system design, and testing, while outlining the scope, limitations, methodology, activity plan, budget, and significance. The second chapter explores existing literature on AI in education, reviewing automated code review systems like ECHO and DeepReview, and analyzing technologies such as GNNs, Static Program Analysis, and Abstract Syntax Trees to identify milestones like graph-based code analysis and gaps, such as limited feedback explainability and dataset constraints, which guide the system's design. The third chapter examines the limitations of current assessment practices, such as delayed feedback and subjective grading, specifies functional and non-functional requirements like code submission, feedback generation, scalability, and security, and models the system using UML diagrams, including use case, class, activity, and sequence diagrams, to ensure alignment with stakeholder needs.

The fourth chapter details the system's technical design, describing the software architecture with React for the frontend, Laravel for the backend, and integration of GNNs and LLMs, alongside subsystem decomposition, database design using PostgreSQL for user data and code structures, and deployment on Render. It includes user interface wireframes, security measures like role-based access control, and verifies that the design meets all requirements for a robust, user-friendly platform. The fifth chapter reviews the implementation process, selecting development tools such as React, Laravel, PyTorch for GNNs, and Hugging Face for LLMs, and describes coding key features like code evaluation, feedback generation, and the analytics dashboard, addressing challenges and including code snippets to illustrate technical execution. The sixth chapter prepares test plans for unit, integration, and user acceptance testing with students and educators, evaluates performance, usability, and educational impact, and discusses results to identify improvements based on feedback. The final chapter summarizes the project's achievements, reflects on

successes and limitations, such as dataset constraints and coding style evaluation, and suggests future enhancements, like supporting additional languages or integrating with other e-learning platforms.

This structured roadmap underscores the project's potential to transform coding education by automating code reviews, reducing educators' workload, providing immediate, personalized feedback, and offering data-driven insights to enhance teaching. By addressing critical challenges in university assessments, the system promotes fairness, efficiency, and innovation, benefiting academic institutions worldwide. The chapters that follow detail each phase research, modeling, design, implementation, and evaluation inviting readers to explore how this AI-driven platform can reshape learning experiences and inspire advancements in educational technology.

Chapter Two: Literature Review

Regarding platforms that are used to grade students' programming exercises, there have been several studies done previously. These approaches were not strictly reliant on AI/ML, and used various other methods including statistical techniques, random input test cases, or pattern matching using regular expressions [4] and also other ML-based techniques, which involved converting the code to Abstract Syntax Tree (AST) or Control Flow Graph (CFG) and comparing those to the AST or CFG of a model code solution. [5]

Building a platform where students perform source code submissions of their exams or assignments, and where teachers perform the review, evaluation and grading of the students' submissions assisted with an automatic Machine Learning based tool, requires careful consideration of the core part of the project: the ML model which we seek to train. Reviewing previous literature regarding this yields important insights and lessons.

The objective of the project would then be to develop an ML-based tool to review students' source code submissions focusing on correctness, compliance to best practices, and potential errors/bugs. As explained by works such as Nielson et al. [6] and Allamanis et al. [7], the tasks mentioned fall under Program Analysis, which extracts insight about code behavior.

Program analysis can be divided into Static Analysis and Dynamic Analysis. Static Analysis examines the code without execution[8], while Dynamic Analysis analyzes code during execution[9]. Static Program Analysis would be preferable due to its safety and efficiency, avoiding the security risks and overhead with executing untrusted code.[10]

There have been different observations from previous works, in which different ML model architectures were used, including Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNN). A recurring theme with previous works seems to be the practice of representing source code in the form of Abstract Syntax Trees and training them with GNN models.

2.1. Study Of Related Works

The process of reviewing and assessing students’ source code submissions in exams or assignments can be facilitated with the use of Static Program Analysis. The book by Allamanis et al. [11] provides significant insights into the usage of Graph Neural Networks (GNNs) for program analysis. The work underscores how GNNs can leverage the structured nature of source code through graph representations such as Abstract

Syntax Trees and Control Flow Graphs. It highlights the ability of GNNs to capture patterns and ambiguous coding information, and this aligns with the requirements of our student code review platform, which must handle source code submissions that are diverse and potentially inconsistent.

The ECHO paper [12] introduces a machine learning method designed to automate the reuse of feedback in educational code reviews by analyzing patterns in Abstract Syntax Trees (ASTs). They developed the ECHO model which mines patterns in Abstract Syntax Trees of the students' solution to programming exercises, where the teachers had added annotation to previous students' solutions. The model then takes the AST form of new students' solutions, and tries to find similar coding patterns that match the ones that had been annotated by the teachers before. Examples of the annotations given by the teachers included "*It could have been better to use a for loop here*", and "*consider using enumerate instead of iterating with range and len*".

ECHO's explicit targeting of educational code reviews makes it highly relevant to our current project. It addresses the challenge of providing consistent, actionable feedback to students, which is a core objective of the ML-based tool we are aiming to build. In addition, the use of ASTs helps provide a structured representation of code, enabling ECHO to capture syntactic and semantic patterns effectively. This aligns with the need to analyze student submissions for correctness and adherence to best practices.

The DeepReview approach [13] introduces a deep learning model that applies multi-instance learning to automate code review processes. By treating each code change as a 'bag' containing multiple 'hunks' (individual code modifications), DeepReview enables detailed analysis at a finer granularity. This may be particularly beneficial for student submissions, which often include multiple changes that need individual assessment.

The use of Convolutional Neural Networks (CNNs) allows DeepReview to automatically learn and extract semantic features from code changes, facilitating the identification of complex patterns without manual feature engineering. This capability is advantageous in educational settings where submissions may vary widely in style and structure. In addition, DeepReview's end-to-end nature streamlines the code review process by directly predicting approval or rejection of code changes, reducing the manual effort required from educators.

2.2. Identifying Milestones of the Related Literatures and Finding the Gaps

Looking at how machine learning has changed the way we review code, especially for students, it's clear that a few big steps forward have really shaped where we are today. Back in 1999, Nielson and his team laid out the basics of program analysis, explaining the difference between Static Analysis, which checks code without running it, and Dynamic Analysis, which looks at code while it's running [6]. Their work was a game-changer because it showed that Static Analysis is safer and faster, especially for something like grading student code where you don't want to risk running unknown programs. This gave researchers a solid starting point to build better tools. Then, Allamanis and his colleagues took things further by bringing machine learning into the mix, using models that could figure out coding patterns and deal with the messy, unpredictable nature of code [7]. That was a huge leap because it meant we could start analyzing code in a smarter, more flexible way. Later on, Allamanis and his team pushed the envelope again with Graph Neural Networks, or GNNs, which use structures like Abstract Syntax Trees (ASTs) and Control Flow Graphs (CFGs) to dig into the deeper meaning of code [11]. This is super helpful for checking student assignments, where you need to catch things like misused variables or weird logic, especially when everyone codes a bit differently. For classrooms specifically, the ECHO framework by Hellendoorn and Barr came up with a clever way to reuse feedback [12]. It looks at teacher notes on past student code and applies them to new submissions, saving teachers a ton of time. Meanwhile, the DeepReview model by Chen and others used Convolutional Neural Networks, or CNNs, to break down code changes into small pieces and analyze them closely, making it easier to spot issues automatically [13].

Even with all these cool advancements, there are still some big challenges that our project wants to tackle. GNNs are great at understanding the structure of code, which is perfect for the variety of submissions you get in a classroom [11]. ECHO's feedback system keeps things consistent and cuts down on manual work [12], and DeepReview's CNNs make it quick to analyze code by pulling out key details without needing a human to guide it [13]. But here's the catch: a lot of these tools, like DeepReview and GNN-based systems, focus on finding bugs or giving a simple thumbs-up or thumbs-down, which isn't enough for students who need clear, helpful feedback to learn [13], [11]. Also, these machine learning models can be like black boxes—you don't always know why they're saying something's wrong, and that's a problem when students and teachers need to understand the feedback [11], [13]. Another issue is that tools like ECHO rely on big, pre-organized datasets, like old assignments with teacher notes, which don't always match the creative, sometimes wild solutions students come up with [12]. This makes it hard for these systems to handle unique

coding styles or different programming languages. Plus, there's not much out there about creating short, clear outputs—like keywords—that could help connect these models to language tools that explain things in a way students can get.

Our project is stepping in to fill these gaps. We're building a system that uses GNNs to analyze student code in languages like C++, Java, and Python, pulling out key details in a clear way. Then, we're pairing that with a lightweight Large Language Model to turn those details into feedback that's easy to understand and actually helps students improve. We want our system to be efficient but also open about how it makes decisions, so it's useful in real classrooms. By training it on a wide range of code samples, we're aiming to make it flexible enough to handle all sorts of student work, tackling the issues with feedback, clarity, and adaptability that we've seen in earlier research. This way, we're building on what's come before but fixing the parts that don't quite work for teaching coding.

2.3. Graph Neural Network

Graph Neural Networks (GNNs) have emerged as a powerful class of deep learning models designed to operate on graph-structured data, enabling the modeling of complex relationships and interdependencies inherent in various real-world systems [14].

Traditional neural networks, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are well-suited for data with regular structures like images and sequences. However, many real-world data are best represented as graphs, encompassing entities (nodes) and their relationships (edges). Examples include social networks, molecular structures, transportation systems, and knowledge graphs [15].

GNNs are specifically designed to process such graph-structured data. The core idea behind GNNs is the concept of **message passing**, where each node iteratively updates its representation by aggregating information from its neighbors. This process allows GNNs to capture both the features of individual nodes and the topology of the graph, facilitating tasks like node classification, link prediction, and graph classification.

2.3.1. Types of Graph Neural Networks

Over time, various GNN architectures have been developed to address specific challenges and applications:

- **Graph Convolutional Networks (GCNs):** These networks generalize the concept of convolution from grid-like data to graphs. GCNs update a node's representation by aggregating and transforming the features of its neighbors, effectively capturing local graph structure [15].
- **Graph Attention Networks (GATs):** GATs introduce attention mechanisms to assign different weights to neighboring nodes during aggregation, allowing the model to focus on the most relevant parts of the graph for a given task [14].
- **GraphSAGE (Graph Sample and Aggregation):** Designed for large-scale graphs, GraphSAGE samples a fixed-size neighborhood for each node and aggregates their features, enabling inductive learning and scalability [15].
- **Graph Isomorphism Networks (GINs):** GINs aim to achieve maximum expressiveness in distinguishing different graph structures by employing powerful aggregation functions, making them suitable for graph classification tasks [15].

2.4. Lessons Learned from Reviewed Literature

Digging into past research has given us some key takeaways that are shaping how we're building our Automated Code Review E-Learning System to make coding assessments better for students and teachers. One big lesson is how important it is to represent code in a structured way, like using GNNs to look at ASTs and CFGs [11]. These tools let us break down code into its building blocks, catching both the rules it follows and the deeper ideas behind it. That's why we're using GNNs as the heart of our system. It's perfect for handling the wide range of student code we'll see, from neat to totally creative. Another takeaway comes from the ECHO framework, which showed us how reusing feedback from past assignments can save teachers time and keep things consistent [12]. But ECHO also taught us that sticking too closely to old patterns can miss new or unusual solutions, so we're planning to train our system on a broad mix of code to make it more adaptable. The flexibility of GNNs, as Allamanis and his team pointed out, is another reason we're excited about them [11]. They can pick up on complex patterns, like how data moves through a program, which is exactly what we need to give detailed feedback that makes sense in a classroom.

One challenge that kept coming up is that machine learning models, like GNNs and CNNs, can be hard to figure out—they don't always explain why they flagged something as wrong [11], [13]. That's a big deal for students who need to know what to fix and why, so we're adding a lightweight Large Language Model to our system to turn the GNN's findings into clear, helpful advice. The DeepReview model showed us how useful it can be to zoom in on specific parts of code changes with CNNs, and we're borrowing that idea to make sure our feedback is precise [13]. We also learned that combining machine learning with language models could be a smart way to keep things efficient while still giving feedback that feels human and useful [13]. These lessons are pushing us toward a system that mixes the best parts of GNNs for deep analysis with LLMs for easy-to-understand feedback, fixing the gaps in earlier tools that didn't focus enough on teaching or clarity.

These ideas are guiding how we're designing our system. We're setting it up to use GNNs to spot things like code quality or mistakes, then using LLMs to explain those findings in a way that helps students learn. This approach tackles the problems we saw in past research, like feedback that's too vague or systems that don't explain themselves. By using a variety of code samples for training, we're making sure our system can handle all kinds of student work, which is key for our goal of improving coding education in universities and colleges. The next chapter will take these lessons and dive into the specific problems with current grading methods, laying out a plan for our system to meet the needs of students and teachers, using what we've learned to create something new and effective.

The evolution and implementation of Graph Neural Networks (GNNs) have yielded valuable insights into their capabilities and limitations. Effectively harnessing the topological structure of data can lead to substantial enhancements in performance across various tasks [14]. However, scalability remains a significant challenge; GNN models, such as Graph Convolutional Networks (GCNs), often struggle with large-scale graphs due to the computational complexity of graph convolution operations, which involve all neighbors of a node, leading to exponential growth in computations for deep networks. Furthermore, while more expressive models like Graph Isomorphism Networks (GINs) can capture complex patterns, they may also be more computationally intensive and prone to overfitting. Additionally, modeling evolving graphs requires architectures that can capture temporal dependencies, an area that continues to be an active research focus. Combining GNNs with other data types (e.g., text, images) can enhance performance but also introduces additional complexity. Moreover, GNNs have proven effective in code processing tasks, demonstrating their versatility beyond traditional applications.

Chapter Three: Problem Analysis and Modeling

This chapter gets into the analysis of the existing systems and the formulation of a robust model to address the limitations observed. It serves as a bridge between the theoretical underpinnings discussed in the previous chapter and the practical implementation that follows. The chapter outlines the shortcomings of the current systems and presents a systematic approach to defining the requirements and modeling the proposed solution. These activities are vital as they form the foundation upon which the new system will be built, ensuring alignment with stakeholders' needs while addressing the identified gaps.

3.1 Existing System and Its Problems

The current system for reviewing student code submissions in educational settings, relies heavily on manual efforts by educators, presenting significant limitations that hinder effective assessment. This labor-intensive process involves instructors individually evaluating each submission, resulting in time-consuming reviews that delay feedback to students, often impeding their ability to address errors promptly. Feedback quality varies widely, as educators may differ in their focus and expertise, leading to inconsistent guidance that confuses students and undermines skill development. Scalability is a critical issue, as growing class sizes exacerbate the challenge of providing timely reviews, overwhelming educators and straining resources. Subjective evaluations further compromise fairness, as human biases can lead to inconsistent grading, affecting student outcomes. These shortcomings collectively diminish the efficiency and equity of code assessments, creating barriers to effective learning in programming courses.

The root causes of these issues lie in the lack of automation for code analysis and the absence of tools specifically designed for educational contexts, which prevent streamlined and consistent evaluations. Students, educators, and institutions are significantly impacted: delayed or unclear feedback hampers student learning and progression, increased workloads burden educators, detracting from their teaching and mentorship responsibilities, and compromised learning outcomes reflect poorly on institutional effectiveness. Analytical tools have been employed to dissect the complexity of these problems, revealing how the absence of tailored automation contributes to inefficiencies and inequities. The proposed Automated Code Review E-Learning System aims to address these gaps by introducing an AI-driven platform to enhance feedback timeliness, consistency, scalability, and fairness, thereby improving the educational experience for all stakeholders.

3.2 Specifying the Requirements of the Proposed Solution

This section outlines the stakeholders, research, and consultations conducted to identify the requirements for the Automated Code Review E-Learning System, designed to address inefficiencies in manual code review processes at Addis Ababa Science and Technology University. The primary stakeholders include students enrolled in programming courses and educators responsible for assessing code submissions. To understand their needs, research was conducted through surveys of students, interviews with educators, and observations of existing manual review processes. Surveys targeted students across various programming courses to gather insights into their experiences with feedback, while interviews engaged educators to explore challenges in grading. Observations of current practices, such as manual evaluation workflows, provided a practical perspective on system limitations. These methods ensured a comprehensive understanding of the challenges faced by stakeholders, facilitating the identification of requirements for an AI-driven solution that automates code review and delivers actionable feedback.

Students expressed a need for timely and consistent feedback on their code submissions to support rapid learning and error correction, emphasizing that delayed or unclear feedback hinders their progress in mastering programming concepts. They also highlighted the desire for feedback that is specific to their submissions, addressing errors and suggesting improvements in a clear, actionable manner. Educators, tasked with reviewing numerous submissions, identified the need for a system that reduces their grading workload, allowing more time for teaching and mentorship. They stressed the importance of a platform that ensures fair and consistent evaluations, minimizing subjective biases, and provides insights into student performance trends to guide instruction. These stakeholder needs align with the project's goal of leveraging machine learning techniques to automate code review, ensuring efficiency and equity in assessments.

Research into effective code assessment practices underscores the importance of integrating automation to deliver timely, consistent, and scalable feedback, addressing the limitations of manual processes. Analysis of existing manual review systems revealed strengths in educator expertise but significant weaknesses in scalability and consistency, particularly as class sizes grow. Unlike automated systems in industry, educational tools often lack tailored features for student learning, such as actionable feedback or performance analytics. These findings, combined with stakeholder input from interviews, and observations, shaped the development of functional and non-functional requirements, including code submission processing, automated feedback generation, scalability, and user-friendly interfaces. The detailed requirements, informed by this comprehensive research, are presented in the subsequent section, laying the foundation for the proposed solution's design and implementation.

3.3. System Modelling

3.3.1 Functional Requirements

- **User Login:** The system shall provide a single login page for students, and teachers, with a separate, securely hidden login page for admin, using unique user IDs and implementing role-based access control to redirect users to their role-specific dashboards.
- **Admin Login Interface:** The system shall provide a dedicated admin login interface, initially configured with one default account for system management. The sytem will have one single admin, with preset credentials, and only staffs or personnel authorized by the system owners will have access to those credentials.
- **Teacher Registration:** The system shall allow admins to manage teacher registration, enabling the addition of new teacher accounts.
- **Class Creation:** The system shall enable admins to create classes and assign teachers to them for course management.
- **Student Assignment:** The system shall permit admins to perform bulk assignment of students to classes and view a list of unassigned students for efficient management.
- **Teacher Dashboard:** The system shall provide teachers with a dedicated dashboard upon login to manage their teaching activities.
- **Password Management:** The system shall allow teachers to edit their passwords to maintain account security.
- **Test Management:** The system shall enable teachers to create and manage tests for student assessments.
- **Assignment Upload:** The system shall allow teachers to upload assignments and define evaluation criteria for grading.
- **Submission Review:** The system shall enable teachers to review student submissions, including submitted code/files, problem statements, and suggested grades, with options to edit grades.
- **Feedback Annotation:** The system shall allow teachers to annotate or adjust feedback generated by the system for submitted code to provide personalized guidance.
- **Student Dashboard:** The system shall provide students with a dedicated dashboard upon login to access their assignments and feedback.
- **Profile Management:** The system shall allow students to manage their profiles, updating personal information as needed.

- **Test Submission Page:** The system shall provide a combined test view and submission page, including problem statements and areas for code input or file upload.
- **Code Submission:** The system shall enable students to submit assignment or exam source code via the platform's interface.
- **Feedback Access:** The system shall allow students to view machine learning-generated feedback and grades for their submissions.
- **Progress Tracking:** The system shall enable students to track their progress, including upcoming and past exams and results.
- **Feedback Interface:** The system shall provide an intuitive interface for students to interact with feedback, ensuring ease of use.
- **Code File Upload:** The system shall allow students to upload source code files in formats such as .py, .java, and .cpp.
- **Code Parsing:** The system shall automatically parse uploaded code into structured representations, such as Abstract Syntax Trees, for analysis.
- **Code Analysis:** The system shall run machine learning models to analyze code for correctness, adherence to best practices, and potential errors.
- **Feedback Generation:** The system shall use Graph Neural Networks to extract insights from code submissions for detailed evaluation.
- **Detailed Feedback:** The system shall employ Large Language Models to generate detailed, human-readable feedback for code submissions.
- **Feedback Presentation:** The system shall present feedback in a user-friendly format for both students and teachers.
- **Automatic Grading:** The system shall automatically generate grades for student submissions based on a single, global grading criteria applied across the system.
- **Grade Adjustment:** The system shall allow teachers to override or adjust grades via the platform's interface.
- **Performance Report:** The system shall provide students with detailed reports on their performance, including insights into code quality, grade breakdown, and actionable feedback, visible after teacher grading.

3.3.2 Non-Functional Requirements

- **Performance**: The system shall analyze and generate feedback for code submissions rapidly for files of typical complexity and handle multiple concurrent users without significant performance degradation.
- **Scalability:** The system shall automatically scale to handle an increasing number of students and submissions without compromising performance.
- **Reliability**: The system shall ensure high uptime, particularly during critical exam submission periods, with mechanisms to recover and resume operations quickly after an outage.
- **Security:** The system shall encrypt sensitive data, such as student submissions and teacher annotations, using industry-standard protocols like AES-256, implement role-based access control, and use multi-factor authentication to prevent unauthorized access.
- **Model Isolation:** The system shall prevent students and teachers from directly interacting with machine learning models.
- **Usability:** The system shall provide an intuitive interface with clear instructions for students and teachers to submit code, view feedback, and access grades, adhering to WCAG 2.1 accessibility standards.
- **User-Friendly Tools:** The system shall provide simple, user-friendly tools for students and teachers to interact with feedback, without exposing complex system-level interactions.
- **Maintainability:** The system shall have a modular architecture to allow easy updates or replacement of components, such as machine learning models or frontend, and maintain clear and comprehensive documentation for developers and administrators.
- **Extensibility:** The system shall be designed to easily accommodate new features, such as additional programming languages or machine learning models, and allow updates to models and backend infrastructure without disrupting the user experience.

3.3.3. Use-Case

The use case diagrams provide a visual representation of the system's functional requirements, illustrating the interactions between users (actors) and the system's core functionalities. These diagrams highlight how teachers, students, and administrators utilize the platform to perform various tasks such as managing accounts, submitting code, generating feedback, grading assignments, and collaborating effectively. Each diagram focuses on specific aspects of the system, ensuring clarity in understanding user roles and system capabilities.

3.3.3.1 Use Case Identification

Below is the use case identification for the Automated Code Review E-Learning System, showing the primary actors and their corresponding use cases. Each use case reflects an essential interaction between the system and the users.

Use Case ID	**Use Case Name**	**Actor(s)**	**Description**
UC01	Login	Student, Teacher, Admin	Users log in with unique user IDs to access role-specific dashboards.
UC02	Profile Management	Student, Teacher	Users create, update, or delete their profiles, modifying personal details and preferences.
UC03	Upload Tests	Teacher	Teacher uploads assignments and sets grading criteria like rubrics.
UC04	Review Feedback	Teacher	Teacher reviews system-generated feedback for student submissions to ensure accuracy.

UC05	Edit Feedback	Teacher	Teacher edits system-generated feedback to refine or correct it before sharing with students.
UC06	Access Code Insights	Teacher	Teacher accesses insights about student code, such as complexity analysis or code quality.
UC07	Submit Test	Student	Student submits completed assignments before the deadline for evaluation.
UC08	View Feedback	Student	Student views ML-generated feedback and grades for submitted assignments.
UC09	Upload Source Code	Student	Student uploads source code files for assignment submission.
UC10	Submit for Evaluation	Student	Student submits uploaded code for system evaluation, triggering feedback generation.
UC11	Generate Feedback	System	System generates feedback on submitted

			code using GNNs and LLMs, covering correctness and style.
UC12	Assign Grades	Teacher	Teacher assigns grades based on evaluated student submissions.
UC13	Edit Grades	Teacher	Teacher modifies previously assigned grades to ensure accuracy.
UC14	Create Class	Admin	The admin creates a class for a programming course to organize code review activities.
UC15	Register Teacher	Admin	The admin registers a teacher with a role to manage assignments and feedback.
UC16	Assign Student and Teacher to Class	Admin	The admin assigns students and teachers to a class for course activities.
UC17	View Analytics Dashboard	Admin	The admin views a dashboard of system metrics to monitor performance.

Table 2. Use Case Identification

3.3.3.2 Use Case Mapping

Use Case Name: Login (UC02)

Field	Details
Use Case ID	UC02
Primary Actor	Student, Teacher, Admin
Description	Users log in with unique credentials to access role-specific dashboards, implementing role-based access control within the AASTU scope.
Preconditions	User is registered and has valid credentials.
Postconditions	User is authenticated and redirected to their role-specific dashboard.
Flow of Events	1. User navigates to the login page. 2. User enters unique ID and password. 3. System validates credentials and role. 4. System redirects to the appropriate dashboard.
Alternate Flows	If credentials are invalid, the system displays an error and prompts re-entry.

Table 3. Use Case Mapping for Login (UC02)

Use Case Name: Profile Management (UC02)

Field	Details
Use Case ID	UC02
Primary Actor	Student, Teacher

Description	Users create, update, or delete their profiles, modifying personal details and preferences to personalize their experience.
Preconditions	User is logged in with valid credentials.
Postconditions	Profile is created, updated, or deleted successfully.
Flow of Events	1. User navigates to the profile management interface. 2. User selects to create, update, or delete profile. 3. User enters or modifies details (e.g., name, preferences). 4. System validates and saves changes.
Alternate Flows	If input is invalid, the system displays an error and prompts correction.

Table 4. Use Case Mapping for Profile Management (UC02)

Use Case Name: Upload Assignments (UC03)

Field	Details
Use Case ID	UC03
Primary Actor	Teacher
Description	Teacher uploads tests and sets grading criteria like rubrics or evaluation guidelines for student assessments.
Preconditions	Teacher is logged in and assigned to a class.
Postconditions	Assignment is uploaded and available to students.
Flow of Events	1. Teacher navigates to the assignment upload

	interface. 2. Teacher uploads the assignment file and sets criteria. 3. System validates the input. 4. System saves the assignment.
Alternate Flows	If criteria are incomplete, the system prompts the teacher to complete them.

Table 5. Use Case Mapping for Upload Assignments (UC03)

Use Case Name: Review Feedback (UC04)

Field	Details
Use Case ID	UC04
Primary Actor	Teacher
Description	Teacher reviews system-generated feedback for student submissions to ensure accuracy before sharing with students.
Preconditions	Teacher is logged in, and feedback is generated for a submission.
Postconditions	Feedback is reviewed and ready for editing or sharing.
Flow of Events	1. Teacher navigates to the feedback review interface. 2. Teacher selects a student submission. 3. System displays generated feedback. 4. Teacher reviews the feedback.
Alternate Flows	If no feedback is available, the system notifies the teacher.

Table 6. Use Case Mapping for Review Feedback (UC04)

Use Case Name: Edit Feedback (UC05)

Field	Details
Use Case ID	UC05
Primary Actor	Teacher
Description	Teacher edits system-generated feedback to refine or correct it before it is made available to students.
Preconditions	Teacher has reviewed feedback for a submission.
Postconditions	Feedback is updated and stored for student access.
Flow of Events	1. Teacher selects feedback to edit. 2. Teacher modifies feedback content. 3. System validates changes. 4. System saves the updated feedback.
Alternate Flows	If changes are invalid, the system displays an error and prompts correction.

Table 7. Use Case Mapping for Edit Feedback (UC05)

Use Case Name: Access Code Insights (UC06)

Field	Details
Use Case ID	UC06
Primary Actor	Teacher
Description	Teacher accesses insights about student code, such as complexity analysis or code quality, after assignments are submitted.

Preconditions	Teacher is logged in, and code has been evaluated
Postconditions	Teacher views code insights for a submission.
Flow of Events	1. Teacher navigates to the code insights interface. 2. Teacher selects a student submission. 3. System displays insights. 4. Teacher reviews the insights.
Alternate Flows	If no insights are available, the system notifies the teacher.

Table 8. Use Case Mapping for Access Code Insights (UC06)

Use Case Name: Submit Test (UC07)

Field	Details
Use Case ID	UC07
Primary Actor	Student
Description	Student submits completed test before the deadline for evaluation, recorded by the system.
Preconditions	Student is logged in, and an test is available..
Postconditions	Test is submitted and recorded.
Flow of Events	1. Student navigates to the test submission page. 2. Student uploads or inputs the test. 3. Student submits the assignment. 4. System records the submission
Alternate Flows	If the deadline has passed, the system rejects the submission and notifies the student.

Table 9. Use Case Mapping for Submit Test (UC07)

Use Case Name: View Feedback (UC08)

Field	Details
Use Case ID	UC08
Primary Actor	Student
Description	Student views machine learning-generated feedback and grades for submitted tests to understand improvement areas.
Preconditions	Student is logged in, and feedback/grades are available.
Postconditions	Student accesses and reviews feedback and grades.
Flow of Events	1. Student navigates to the feedback interface. 2. Student selects a submission. 3. System displays feedback and grades. 4. Student reviews the feedback.
Alternate Flows	If no feedback is available, the system notifies the student.

Table 10. Use Case Mapping for View Feedback (UC08)

Use Case Name: Upload Source Code (UC09)

Field	Details
Use Case ID	UC09
Primary Actor	Student
Description	Student uploads source code files for test submission, stored for evaluation.

Preconditions	Student is logged in and has accessed the submission page.
Postconditions	Source code is uploaded and stored.
Flow of Events	1. Student navigates to the submission page. 2. Student uploads a source code file. 3. System validates the file format. 4. System stores the file.
Alternate Flows	If the file format is unsupported, the system displays an error and prompts a valid file.

Table 11. Use Case Mapping for Upload Source Code (UC09)

Use Case Name: Submit for Evaluation (UC10)

Field	Details
Use Case ID	UC10
Primary Actor	Student
Description	Student submits uploaded source code for system evaluation, triggering feedback generation.
Preconditions	Student has uploaded source code.
Postconditions	Code is submitted, and evaluation is triggered.
Flow of Events	1. Student navigates to the submission page. 2. Student confirms the uploaded code. 3. Student submits for evaluation. 4. System triggers feedback generation.
Alternate Flows	If no code is uploaded, the system prompts the student to upload a file.

Table 12. Use Case Mapping for Submit for Evaluation (UC10)

Use Case Name: Generate Feedback (UC11)

Field	Details
Use Case ID	UC09
Primary Actor	System
Description	System generates feedback on submitted code using Graph Neural Networks and Large Language Models, covering correctness and style.
Preconditions	Code is submitted and parsed into a structured format (Abstract Syntax Tree).
Postconditions	Feedback is generated and stored for access.
Flow of Events	1. System receives parsed code. 2. System applies GNNs to analyze code structure. 3. System uses LLMs to generate readable feedback. 4. Feedback is stored for teacher and student access.
Alternate Flows	If parsing fails, the system logs an error and skips feedback generation.

Table 13. Use Case Mapping for Generate Feedback (UC11)

Use Case Name: Assign Grades (UC12)

Field	Details
Use Case ID	UC12
Primary Actor	Teacher

Description	Teacher assigns grades based on evaluated student submissions, recorded by the system.
Preconditions	Teacher is logged in, and submissions are evaluated.
Postconditions	Grades are assigned and recorded.
Flow of Events	1. Teacher navigates to the grading interface. 2. Teacher selects a submission. 3. Teacher assigns a grade based on evaluation. 4. System records the grade.
Alternate Flows	If no submissions are evaluated, the system notifies the teacher.

Table 14. Use Case Mapping for Assign Grades (UC12)

Use Case Name: Edit Grades (UC13)

Field	Details
Use Case ID	UC09
Primary Actor	Teacher
Description	Teacher modifies previously assigned grades to ensure accuracy in the system.
Preconditions	Teacher has assigned grades for a submission.
Postconditions	Grades are updated and recorded.
Flow of Events	1. Teacher navigates to the grading interface. 2. Teacher selects a submission. 3. Teacher modifies the grade. 4. System saves the updated grade.

Alternate Flows	If no grades are assigned, the system prompts the teacher to assign a grade first.

Table 15. Use Case Mapping for Edit Grades (UC13)

Use Case Name: Create Class(UC14)

Field	Details
Use Case ID	UC14
Primary Actor	Admin
Description	The admin creates a class for a programming course to organize code review activities.
Preconditions	Admin is logged in with permissions.
Postconditions	A class is created and stored.
Flow of Events	1. Admin accesses class management. 2. Admin enters and submits class details. 3. System saves class to database. 4. System confirms creation.
Alternate Flows	Invalid details prompt error. Unauthorized access denied.

Table 16. Create Class(UC14)

Use Case Name: Register Teacher(UC15)

Field	Details
Use Case ID	UC15
Primary Actor	Admin
Description	The admin registers a teacher with a role to manage

	assignments and feedback.
Preconditions	Admin is logged in with permissions.
Postconditions	Teacher account is created with role.
Flow of Events	1. Admin accesses user management. 2. Admin enters and submits teacher details. 3. System saves account and assigns role. 4. System confirms registration.
Alternate Flows	Duplicate email prompts error. Unauthorized access denied.

Table 17. Register Teacher(UC15)

Use Case Name: Assign Student and Teacher to Class (UC16)

Field	Details
Use Case ID	UC16
Primary Actor	Admin
Description	The admin assigns students and teachers to a class for course activities.
Preconditions	Admin is logged in; class and users exist.
Postconditions	Users are assigned to the class.
Flow of Events	1. Admin selects a class. 2. Admin assigns students and teachers. 3. System updates class assignments. 4. System confirms assignments.
Alternate Flows	Non-existent class prompts error. Unauthorized access denied.

Table 18. Assign Student and Teacher to Class (UC16)

Use Case Name: View Analytics Dashboard (UC17)

Field	Details
Use Case ID	UC17
Primary Actor	Admin
Description	The admin views a dashboard of system metrics to monitor performance.
Preconditions	Admin is logged in with access.
Postconditions	Dashboard displays metrics.
Flow of Events	1. Admin accesses analytics section. 2. System retrieves and displays metrics. 3. Admin views dashboard.
Alternate Flows	No data shows empty dashboard. Unauthorized access denied.

Table 19. View Analytics Dashboard

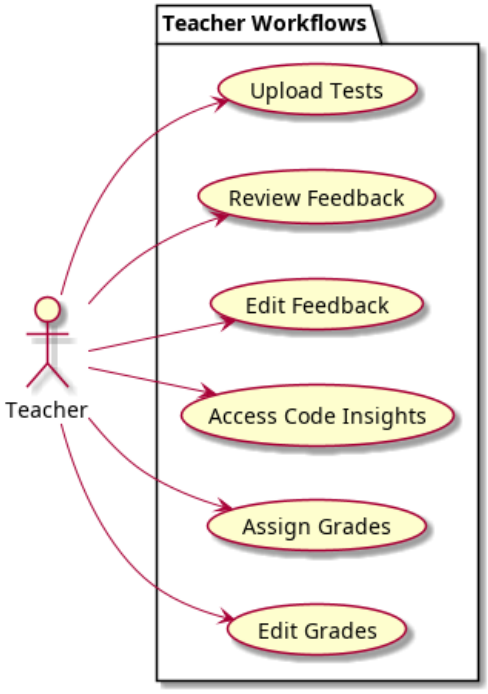

Figure 2. Use Case Diagram for Teacher Workflows

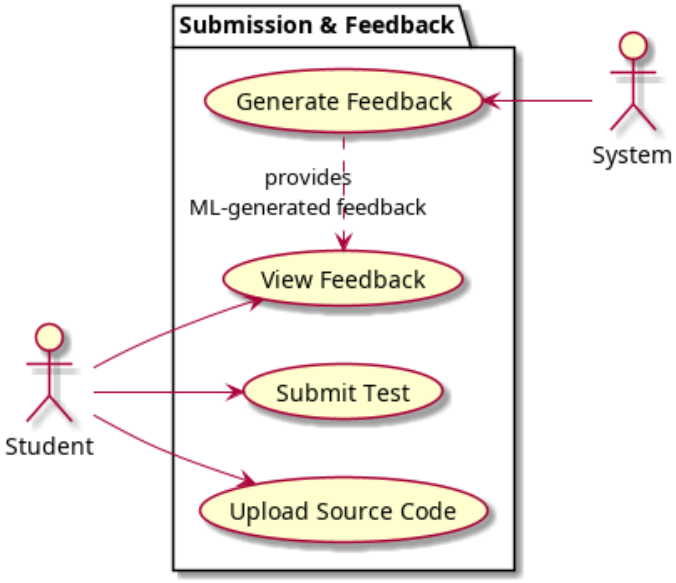

Figure 3. Use Case Diagram for Submission and Feedback

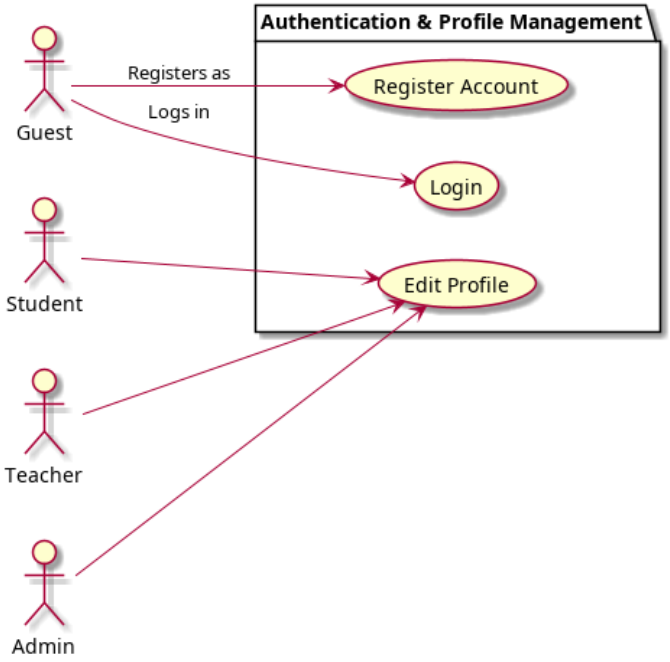

Figure 4. Use Case Diagram for Authentication and Profile Management

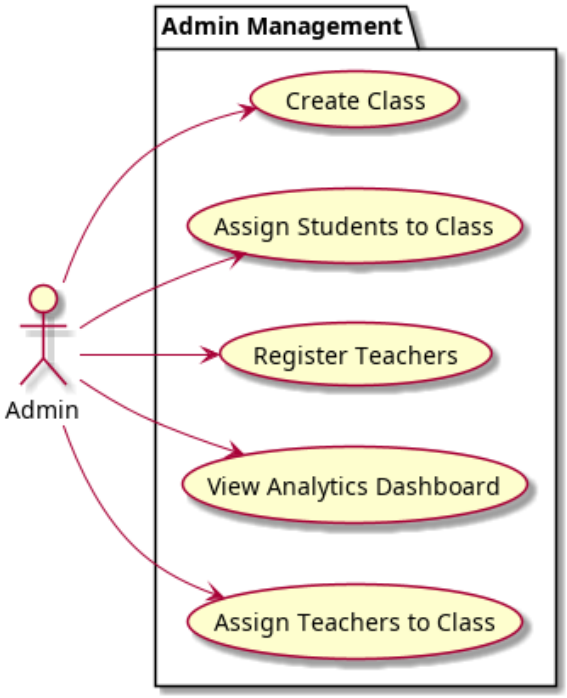

Figure 5. Use Case Diagram For Admin Management

3.3.4. Dynamic Models of a System

This section defines the system's dynamic behavior as it operates under various changing circumstances. The dynamic aspects of the system are illustrated through sequence, state machine and activity diagrams in the following subsections.

3.3.4.1. Sequence Diagrams

A sequence diagram visualizes the interactions between entities (actors, objects, or components) within the system over time. It visualizes the flow of messages and sequence of actions that occur as these entities collaborate to accomplish specific tasks or scenarios. The diagrams below represent various flows and interactions within the Automated code review system.

User Registration, Login & Logout (Student/Teacher/Admin)

This sequence diagram illustrates how users (students and teachers) interact with the authentication system. It covers registration, login, and logout processes.

1. The user initiates the process by registering or logging into the platform.
2. The authentication system verifies credentials and returns the authentication status.

3. When the user logs out, the platform confirms the logout action, ensuring a secure session termination.

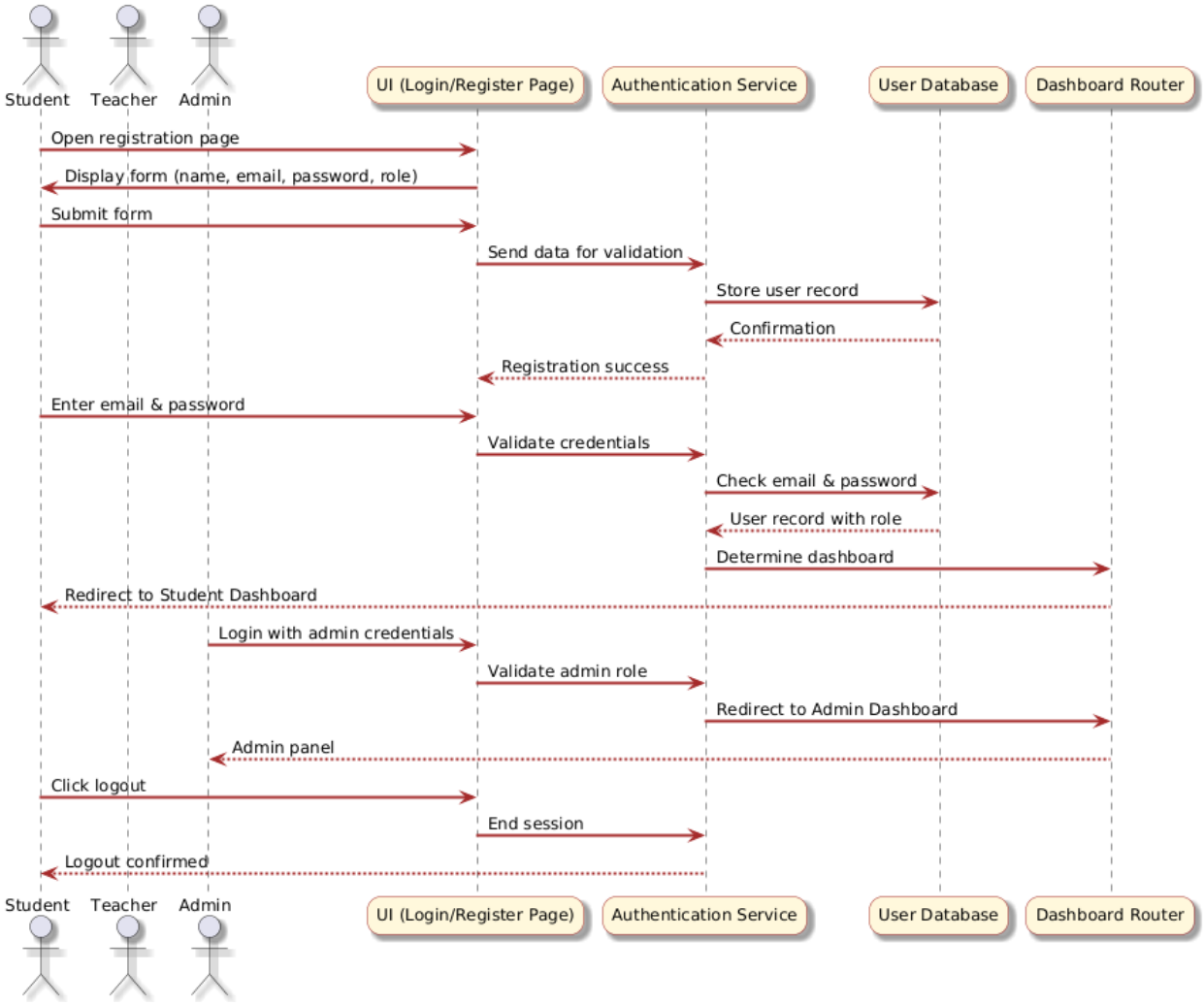

Figure 6. Sequence Diagram For User registration, Login And Logout

Test Upload (Teacher) with Login & Logout

This sequence diagram outlines how teachers upload assignments after logging in.

1. After successful authentication, the teacher creates a test via the platform.
2. The platform stores the assignment details in the database and confirms the upload.

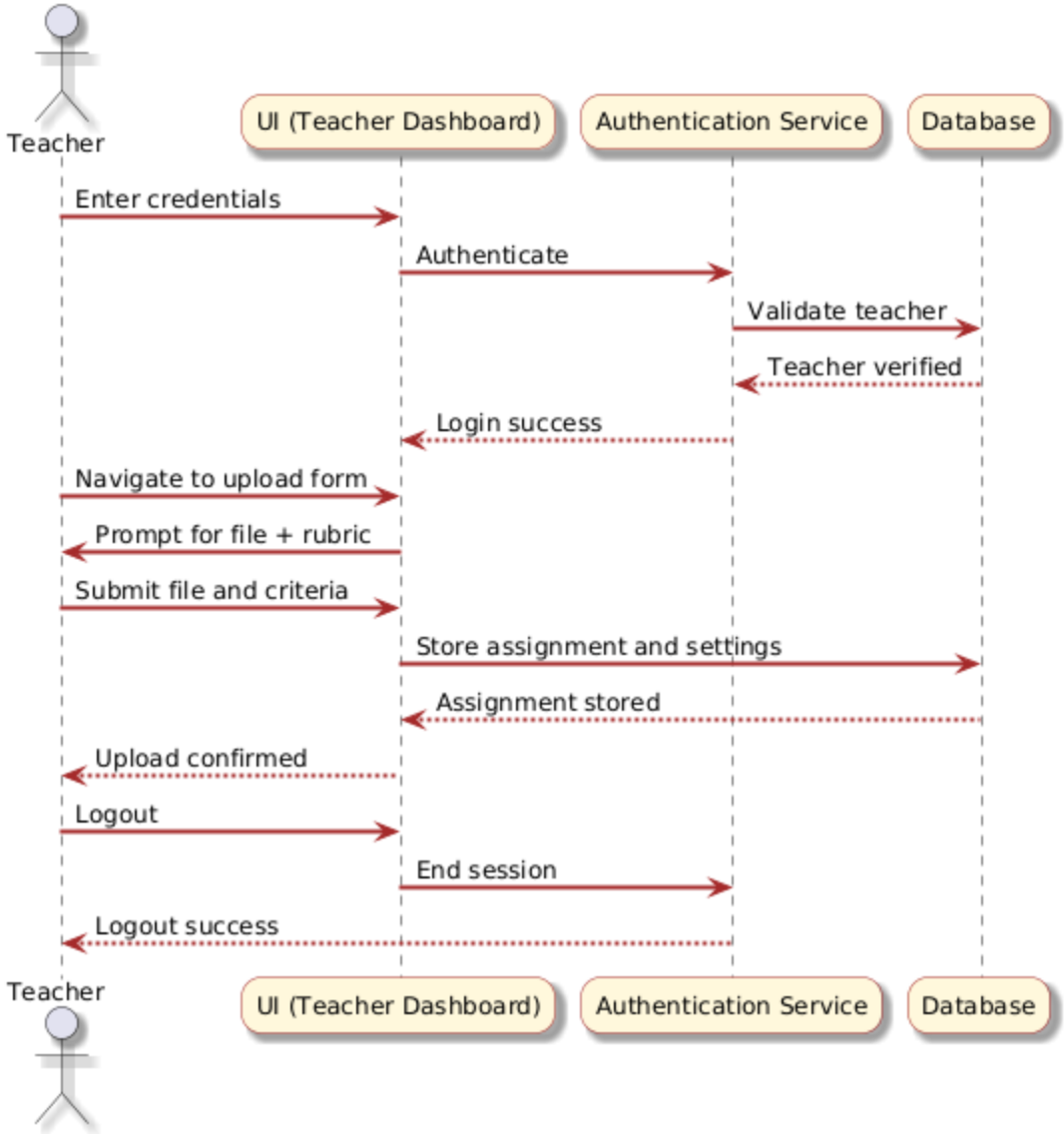

Figure 6. Sequence Diagram For Test Upload (Teacher) With Login & Logout

Code Submission (Student) with Login & Logout

This sequence diagram describes how students submit their source code for evaluation.

1. After successful login the student submits their source code via the platform.
2. The platform stores the submission in the database and sends it to the machine learning engine for analysis.
3. The ML engine evaluates the code for correctness, best practices, and potential errors, then returns the results.

4. The platform stores the feedback and grade in the database and displays them to the student.

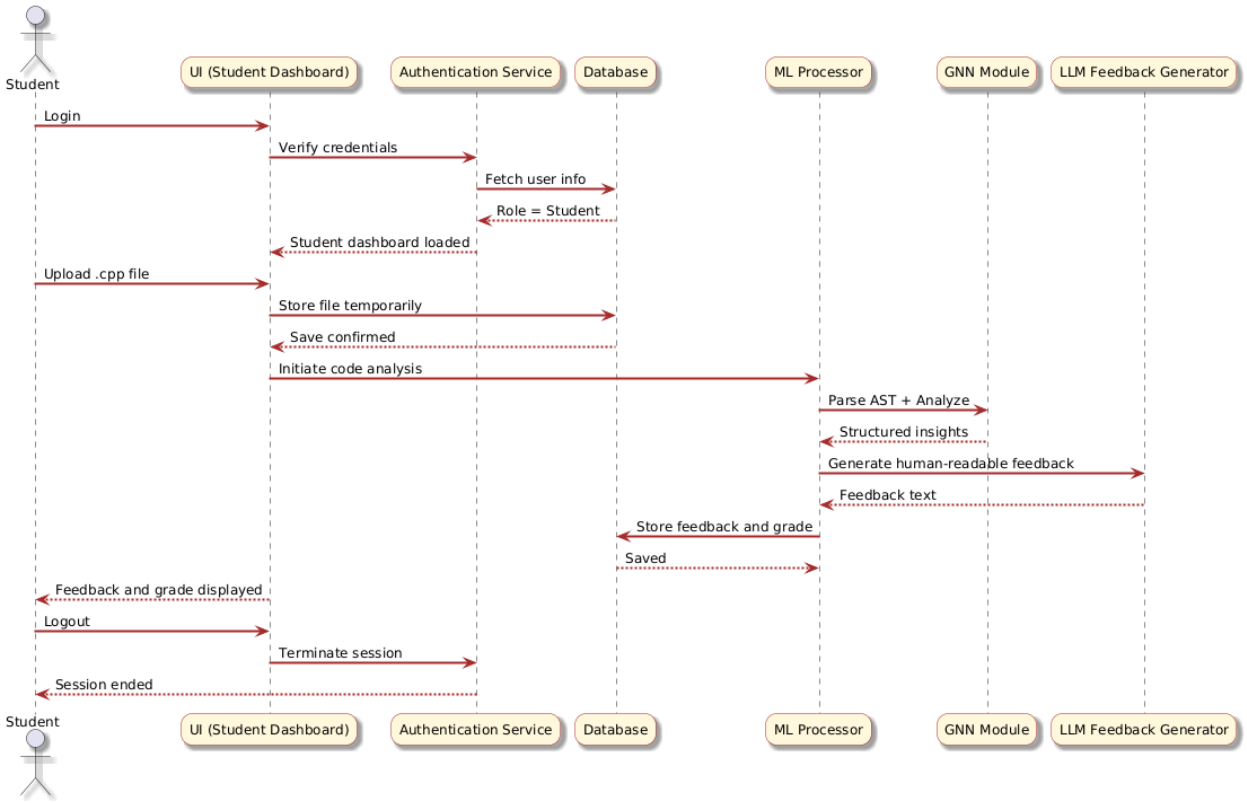

Figure 7. Sequence Diagram For Code Submission (Student) With Login & Logout

Automated Code Review & Feedback Generation

This sequence diagram represents the backend process for analyzing student submissions.

1. The platform retrieves the submitted code and analyzes it using a GNN to extract structured insights.
2. The GNN returns graph-based features representing the code's logic and flow.
3. These features are passed to a lightweight LLM, which generates human-readable feedback.
4. The feedback and grade are stored and displayed to the student or teacher.

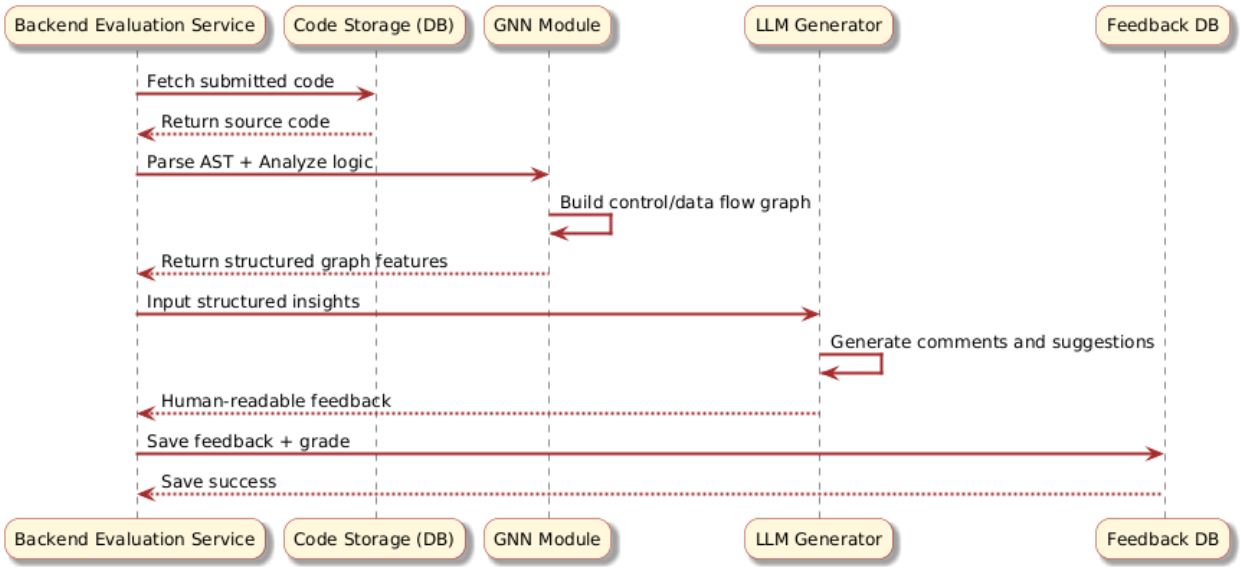

Figure 8. Sequence Diagram For Automated Code Review & Feedback Generation

Teacher Reviewing & Adjusting Feedback with Login & Logout

This sequence diagram shows how teachers can review and adjust feedback provided by the system.

1. The teacher accesses student submissions and system-generated feedback.
2. The platform retrieves the necessary data from the database and displays it to the teacher.
3. The teacher can edit the feedback or adjust grades if necessary.
4. The updated feedback and grades are stored in the database.

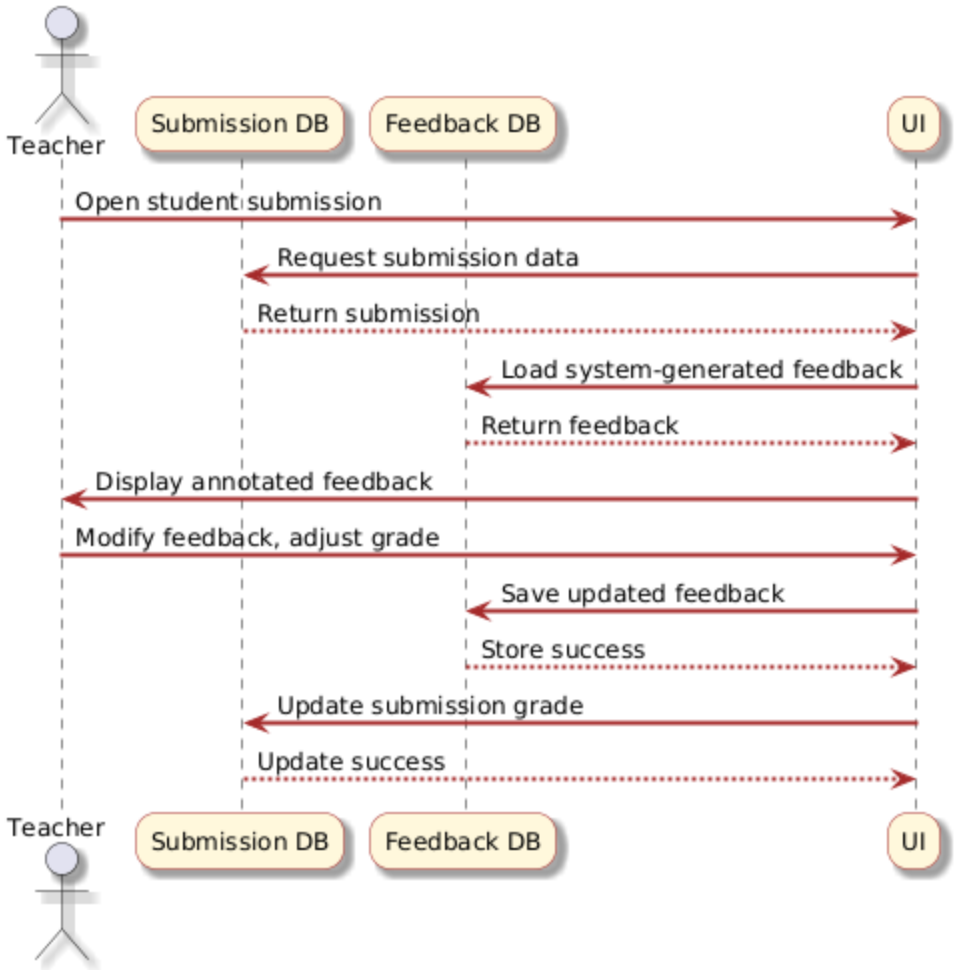

Figure 9. Sequence Diagram For Teacher Reviewing & Adjusting Feedback with Login & Logout

Grading & Report Generation with Login & Logout

This sequence diagram depicts the grading process and report generation for students.

1. The teacher views student grades through the platform.
2. The platform retrieves and displays the grades stored in the database.
3. The teacher can approve, adjust, or override the system-generated grades.
4. The updated grades are stored in the database, and students are notified of any changes.

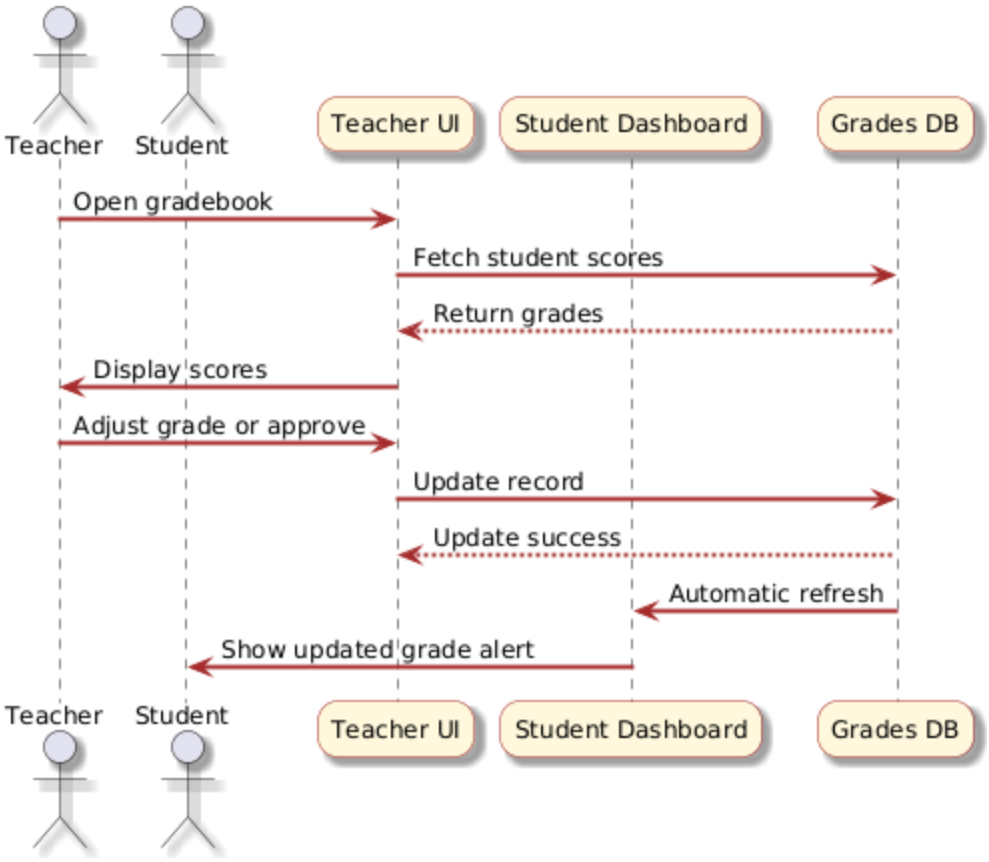

Figure 10. Sequence Diagram for Grading & Report Generation with Login & Logout

3.3.4.2. State Machine Diagrams

A state machine diagram demonstrates how objects within the system transition from different states based on events.

Authentication and Role-Based login

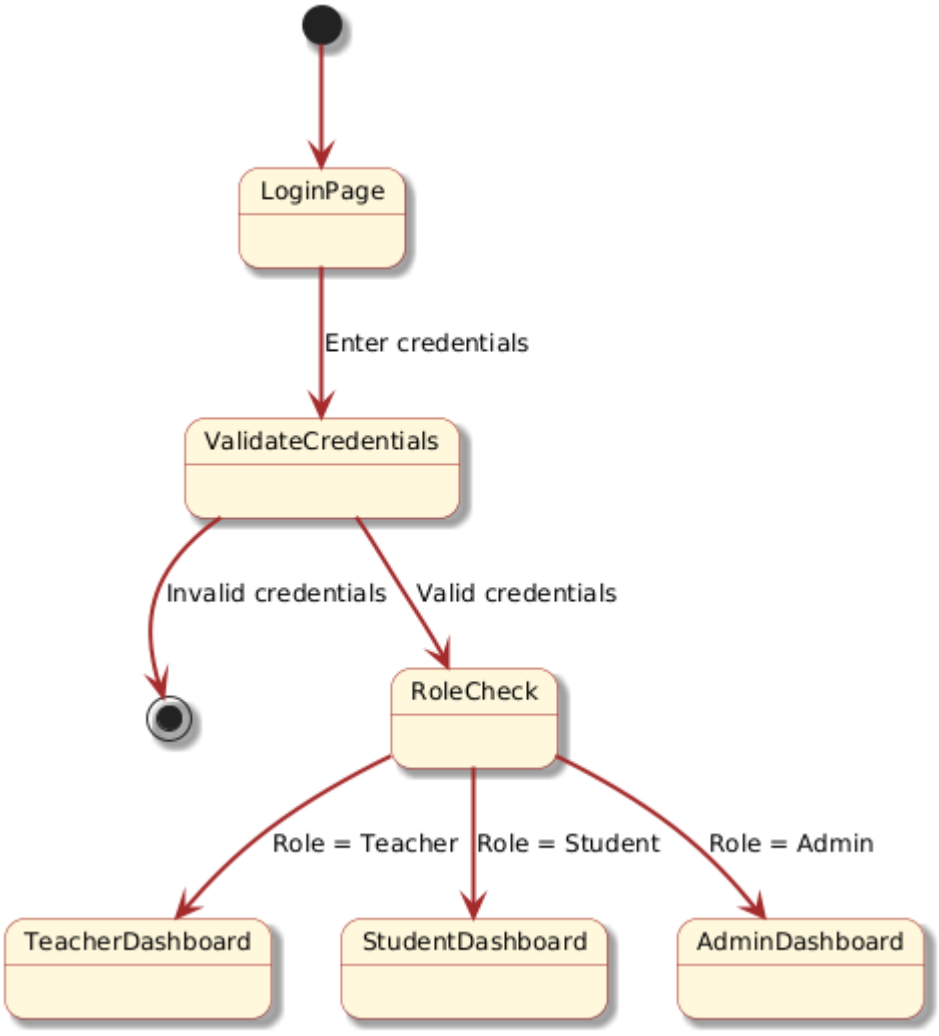

Figure 11. State Machine Diagram for Authentication and Role Based-Login

Teacher create test, review and feedback

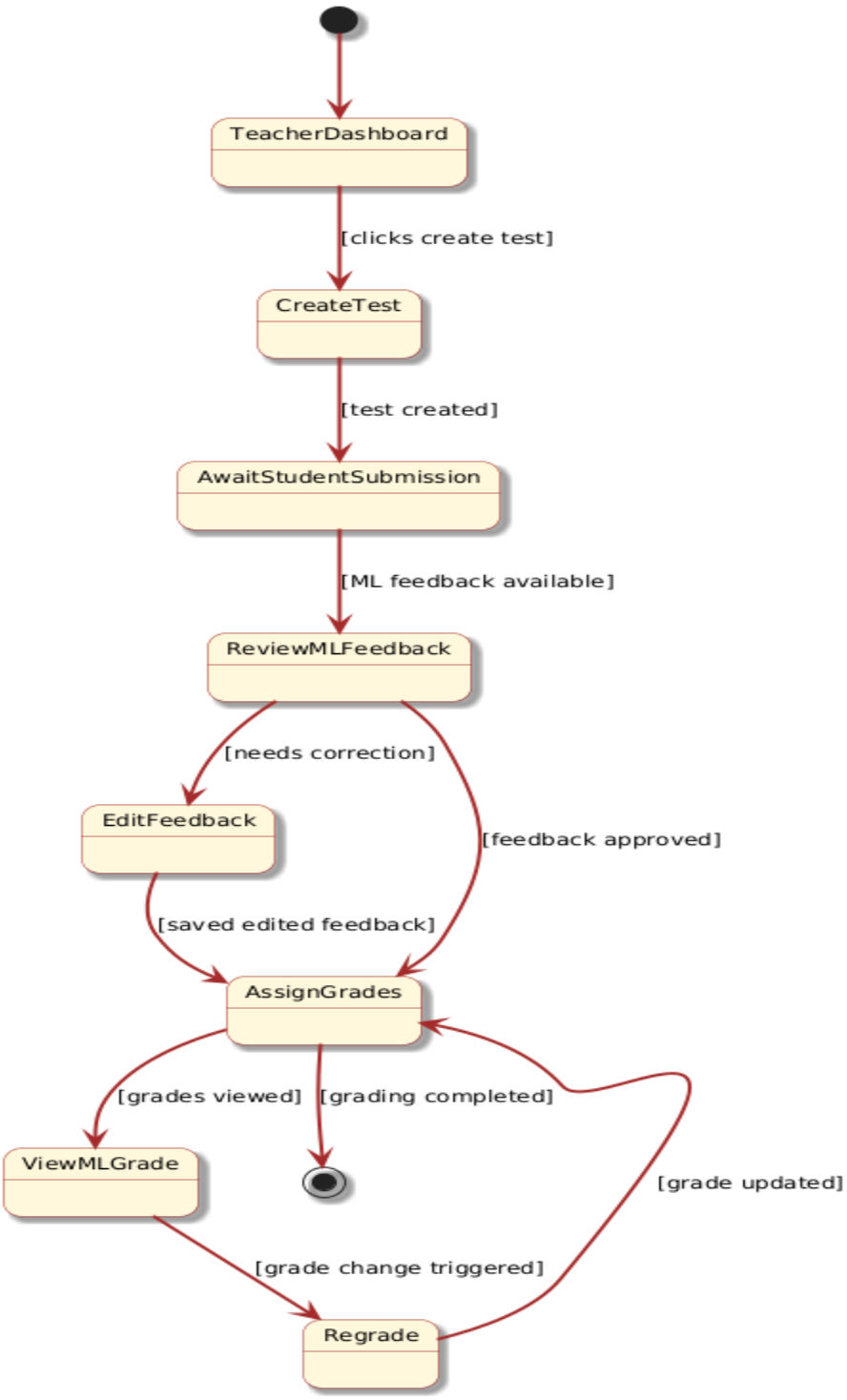

Figure 12. Class Diagram for Teacher Create, Review & Feedback

Student submit test or file

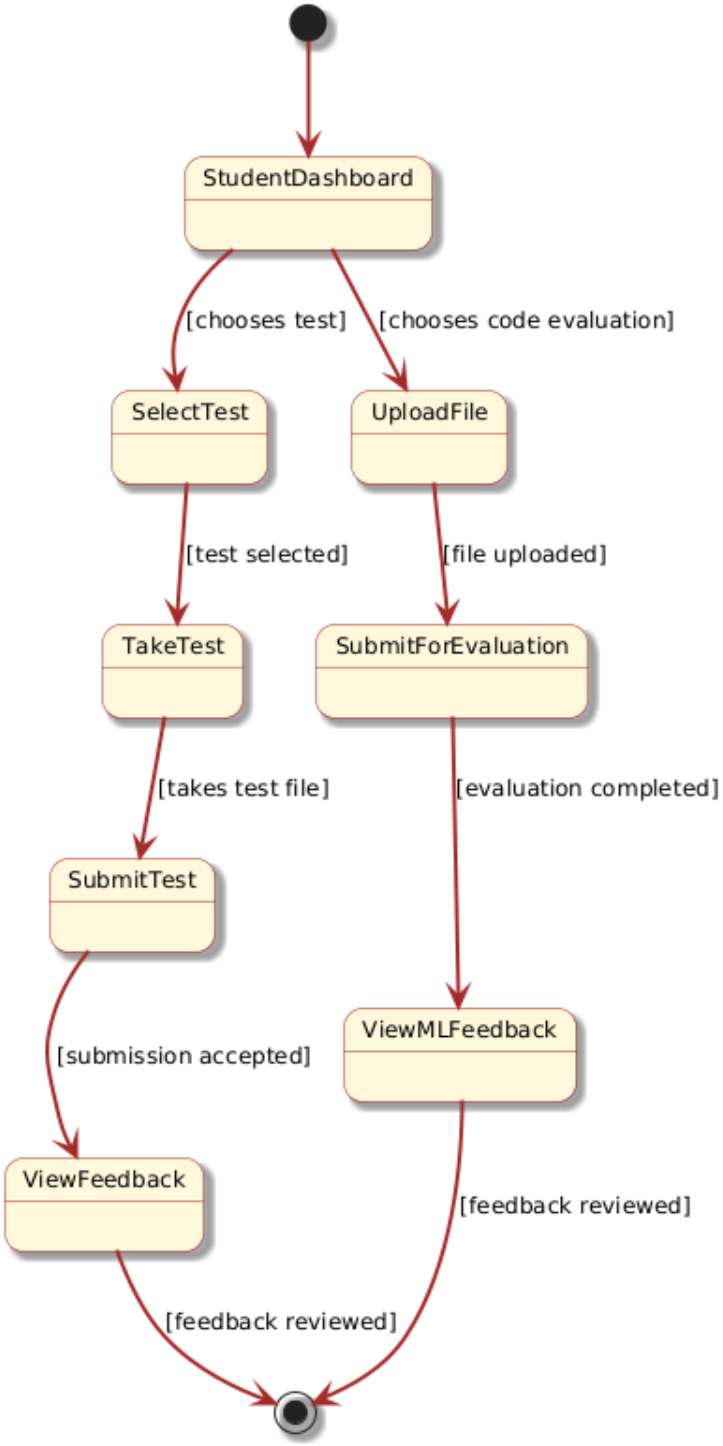

Figure 13. State Machine Diagram for Student Submit Test or File

3.3.4.3. Activity Diagram

Activity diagrams provide a graphical representation of the flow of activities or actions within the system. They represent workflows for specific processes in the Automated code review system.

Student Code Submission Process

This process represents how a student submits code for an assignment.

1. The student logs into the system.
2. They select an assignment to work on.
3. The student uploads their source code file.
4. The system validates the file format.
    - If the file is invalid, an error message is displayed, and the process stops.
    - If the file is valid, the submission is stored.
5. The system then triggers the ML model for analysis.
6. Finally, a confirmation message is shown to the student upon successful submission.

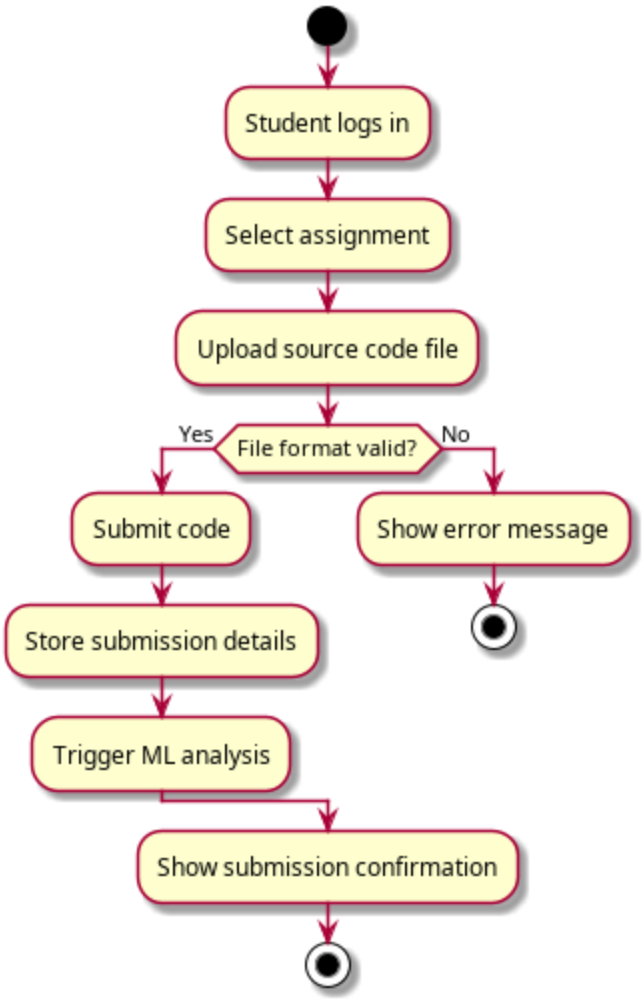

Figure 14. Activity Diagram for Student Code Submission Process

Teacher Reviewing Feedback Process

This process details how a teacher interacts with student submissions to review ML-generated feedback.

1. The teacher logs into the system.
2. They select a specific student's submission.
3. The system displays feedback that was generated by the ML model.
4. The teacher reviews the feedback:
    - If the feedback is accurate, they approve it.
    - If the feedback needs corrections, they edit it before saving the changes.
5. The teacher then assigns a grade to the submission.
6. The final feedback and grade are submitted and stored in the system.

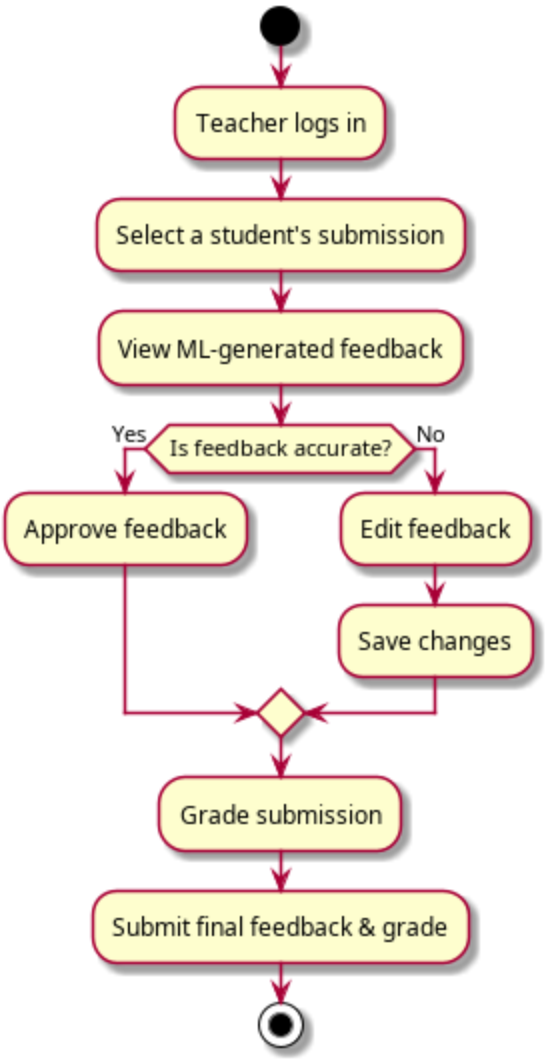

Figure 15. Activity Diagram for Teacher Reviewing Feedback Process

ML Model Processing Submission

This process outlines how the ML model analyzes a student's code submission.

1. The system receives a new code submission.
2. It first validates the code format.
    - If the submission is invalid, the system rejects it and notifies the student.
    - If the submission is valid, it proceeds to the next step.
3. The system parses the code into an Abstract Syntax Tree (AST) for structural analysis.
4. The AST is then passed to a Graph Neural Network (GNN), which extracts insights about the code.

5. The insights from the GNN are fed into a Large Language Model (LLM), which generates human-readable feedback.
6. The generated feedback is stored in the system for teachers and students to review.

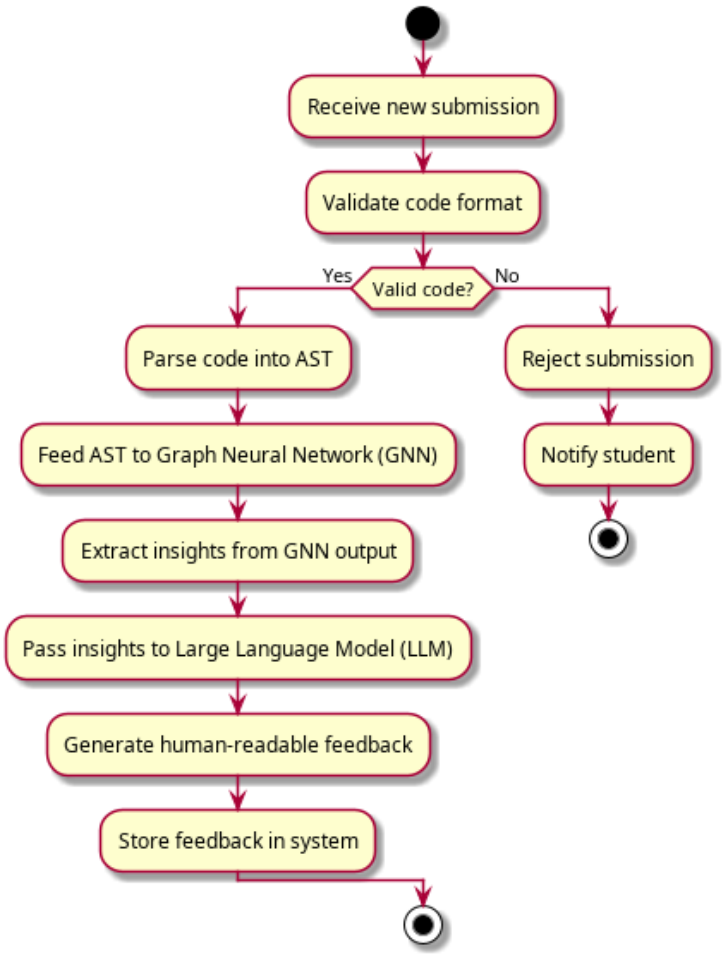

Figure 16. Activity Diagram for ML Model Processing Submission

3.3.4.4 Collaboration Diagram

This diagram illustrates how objects interact and how they pass messages for communication in order to accomplish a specific task.

User Registration

This diagram demonstrates the interaction between objects in the registration process.

Figure 17. Collaboration Diagram forUser Registration

User Login and Verification

This diagram demonstrates the interaction between objects in the login and verification process.

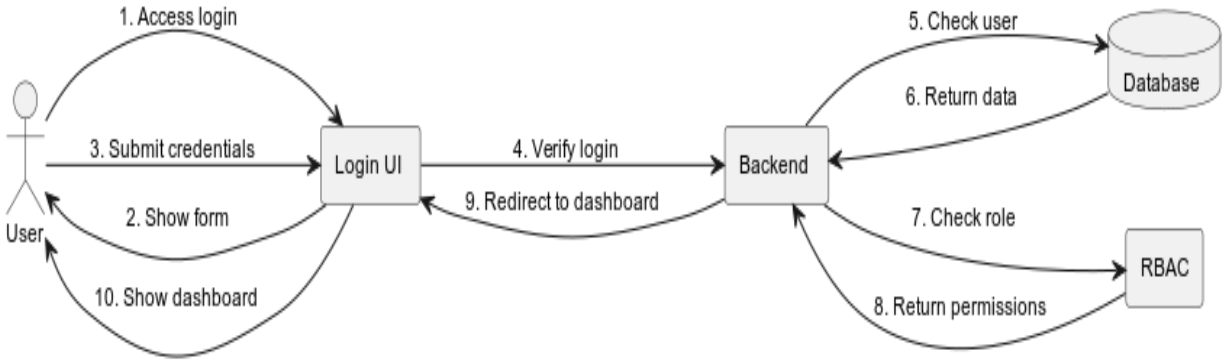

Figure 18. Collaboration Diagram for User Login and Verification

Student code submission

This diagram demonstrates the interaction between objects in the student code submission process.

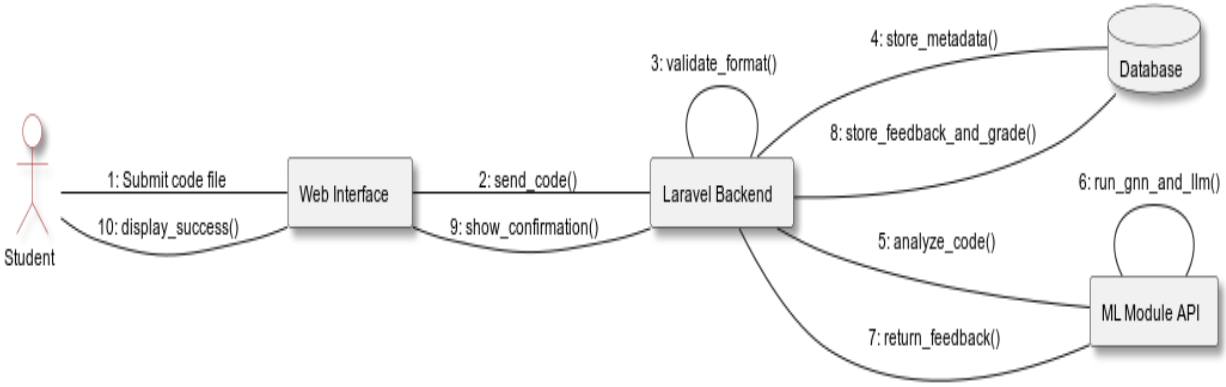

Figure 19. Collaboration Diagram for Student code submission

Teacher feedback

This diagram demonstrates the interaction between objects in the teacher feedback process.

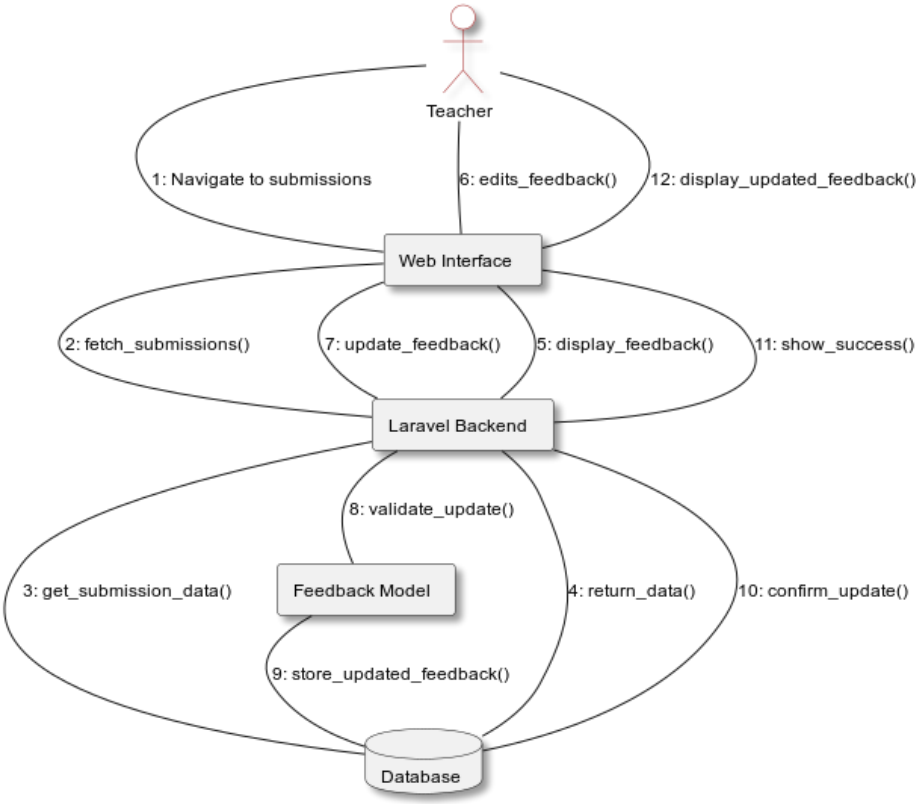

Figure 20. Collaboration Diagram for Teacher feedback

ML analyzer

This diagram demonstrates the interaction between objects during the ML analyzing process.

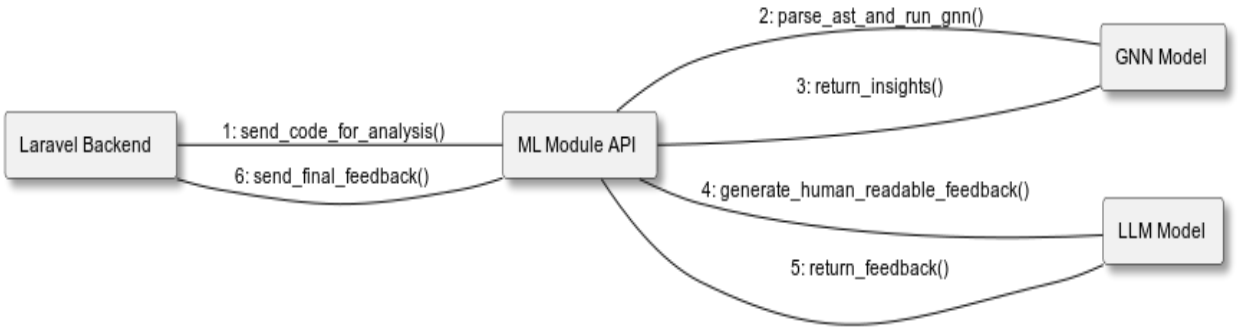

Figure 21. Collaboration Diagram for ML analyzer

3.4. Class based Models of a System

The system's essential elements have been discovered through an analysis of its needs and features. These classes describe the app's basic functionality and are organized by responsibilities inside the system. The table below summarizes the identified classes' properties, methods, and relationships.

3.4.1. Identifying Classes

The system's essential elements have been discovered through an analysis of its requirements and features. These classes describe the app's basic functionality and are organized by responsibilities inside the system. The table below summarizes the identified classes' properties, methods, and relationships.

Class Name	Description	Attributes	Methods
Admin	Represents an administrator entity in the system. Extends Eloquent Model.	user_id (int) createdTeachers createdClasses	createTeacher(data) registerTeacher(data) createClass(data) assignTeacherToClass(teacherId, classId) assignStudentsToClass(studentIds, classId) viewAnalytics()
AiGradingResult	Represents the AI-generated grading result for a student's submission.	submission_id teacher_id	gradeSubmission(submissionId)

ClassRoom	Represents a class entity, linked to a teacher, admin, and multiple students.	name teacher_id admin_id max_students created_by	enrollStudent(studentId) removeStudent(studentId) isFull() assignTeacher(teacherId) getStudents()
Department	Represents a department within the institution.	name	addStudent(studentId) addTeacher(teacherId) getClasses()
Feedback	Represents feedback given by a teacher on a student's submission.	submission_id teacher_id feedback_test annotations	createForSubmission(submissionId, data) updateFeedback(data) addAnnotation(note)
Grade	Represents the grading result of a submission, possibly with manual adjustment.	submission_id teacher_id graded_value adjusted_grade override_reason comments	assign(submissionId, value) adjust(newValue, reason, comments) revertAdjustment()
Student	Represents a student entity in the system.	user_id id_number department_id academic_year	viewAvailableTests() viewSubmissions() viewPerformance() uploadSourceCode(testId, filePath) submitForEvaluation(submissionId) viewFeedback(submissionId)
Submission	Represents a student's submission for a test.	test_id submission_id submission_type code_file_path code_editor_text submission_date status	markSubmitted() triggerEvaluation() updateStatus(status) getFeedback()

Teacher	Represents a teacher entity in the system.	user_id created_by department_id	uploadTest(data) reviewFeedback(submissionId) editFeedback(feedbackId, updates) accessCodeInsights(submissionId) assignGrade(submissionId, value) editGrade(gradeId, newValue, reason)
Test	Represents a test that can be associated with multiple departments and students.	title dueDate status	publish() close() assignToDepartment(departmentId) assignToStudent(studentId) getSubmissions()
User	Represents a user entity in the system, which can be a student, teacher, or admin.	first_name last_name email password	register(data) login(email, password) logout() resetPassword(email) profile() getRole()

Table 20. Class identification for user facing and backend module

The ML modules' basic elements, like those of the rest of the system, were determined through an examination of their requirements and features. These classes describe the core functionality of the ml module and are arranged based on system responsibilities. The following table highlights the properties, methods, and relationships of the identified classes.

Class Name	**Description**	**Attributes**	**Methods**
CodeEmbedderGNN	A GNN-based code embedder that parses Python source code into an AST, converts the AST into a graph, and runs it through GCNConv layers to produce an embedding vector.	node_type_to_id num_node_types convs out_dim	embed(code_str) _ast_to_graph(code_str)

CodeNormalizer	A utility class for normalizing source code by removing comments and collapsing whitespace for Python, C++, and Java.		collapse_whitespace(code_str) remove_comments(code, lang) normalize_code(code, lang) normalize_file(filepath)
CodeParser	A generic code parser that produces an AST using Tree-sitter for supported languages like Python, Java, and C++.		parse(code, lang) get_ast_dict(code, lang)
ConcatEmbedder	A class that concatenates code and text embeddings into a single vector. It combines code embeddings from a CodeEmbedderGNN and text embeddings from a TextEmbedder.	code_emb text_emb use_projection code_proj text_proj final_dim	forward(code_list, text_list, lang: str)
DatasetLoader	A class that loads datasets from CSV files and provides DataLoader instances for batching.	data_dir concat_emb batch_size shuffle num_workers	get_dataset(lang) get_dataloader(lang)
Preprocessor	A class for text preprocessing that includes language detection, stop-word removal, and stemming.	_jp_regex _stop_words _stemmer	preprocess_text(text) is_english(text) preprocess_record(record, fields) preprocess_records(records, fields, filter_non_english)
SubmissionDataset	A custom PyTorch dataset that processes submissions, combining problem statements, code, and submission statistics for training a model.	problem_statements submission_df concat_emb lang stats_csv code_csv problem_csv	_encode_verdict
SubmissionPredictor	A neural network model for predicting verdict, runtime, and memory usage of code submissions.	encoder verdict_head runtime_head memory_head	forward(x)
TextEmbedder	A class for converting preprocessed text into vector representations using TF-IDF.	vectorizer	fit(texts) transform(texts) fit_transform(texts) get_feature_names() embed_record(record, field) embed_records(records, field)

Table 21. Class identification for ML module

3.4.2. Class Diagrams

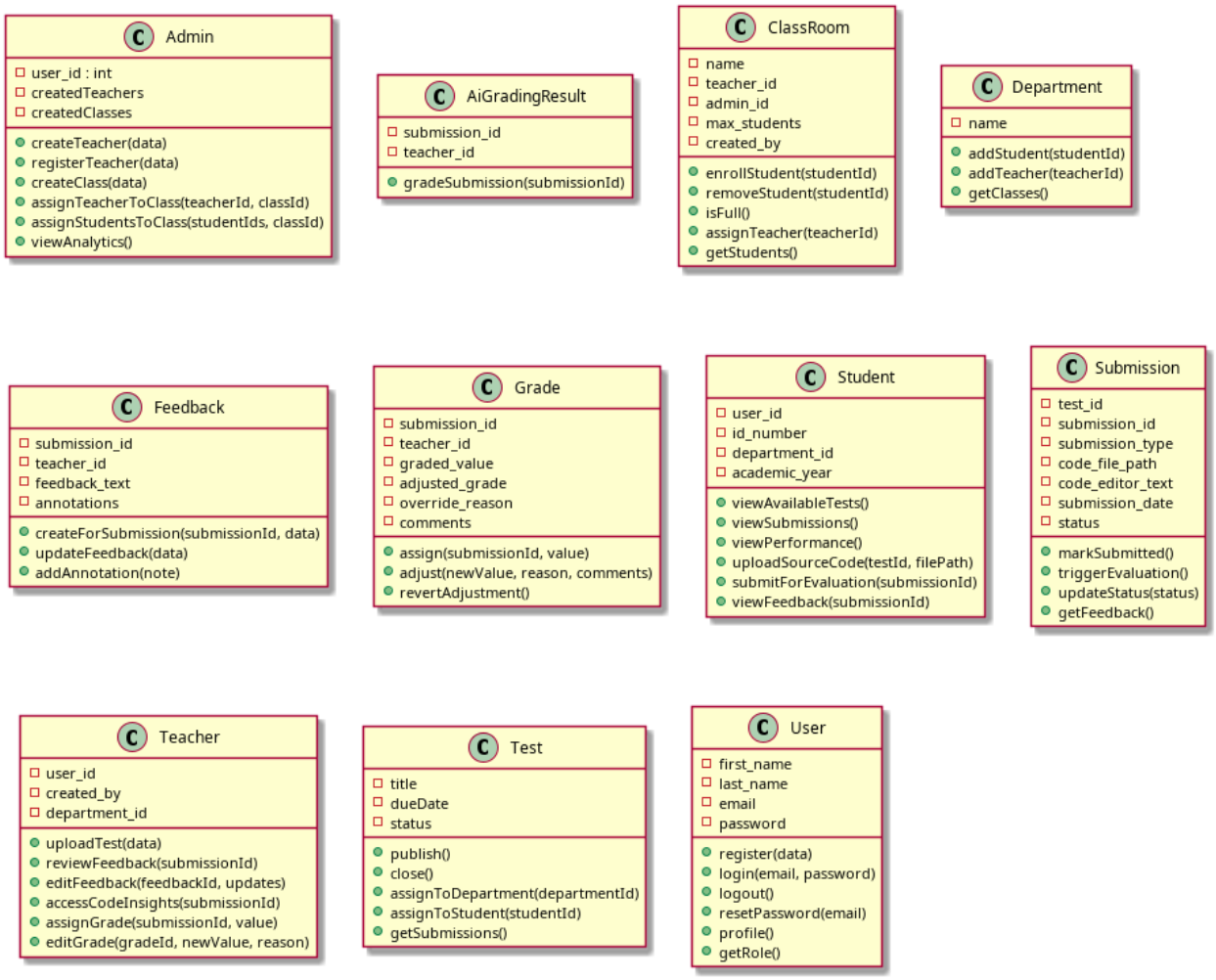

Figure 22. Class Diagrams For User Facing Modules (Definitions Only)

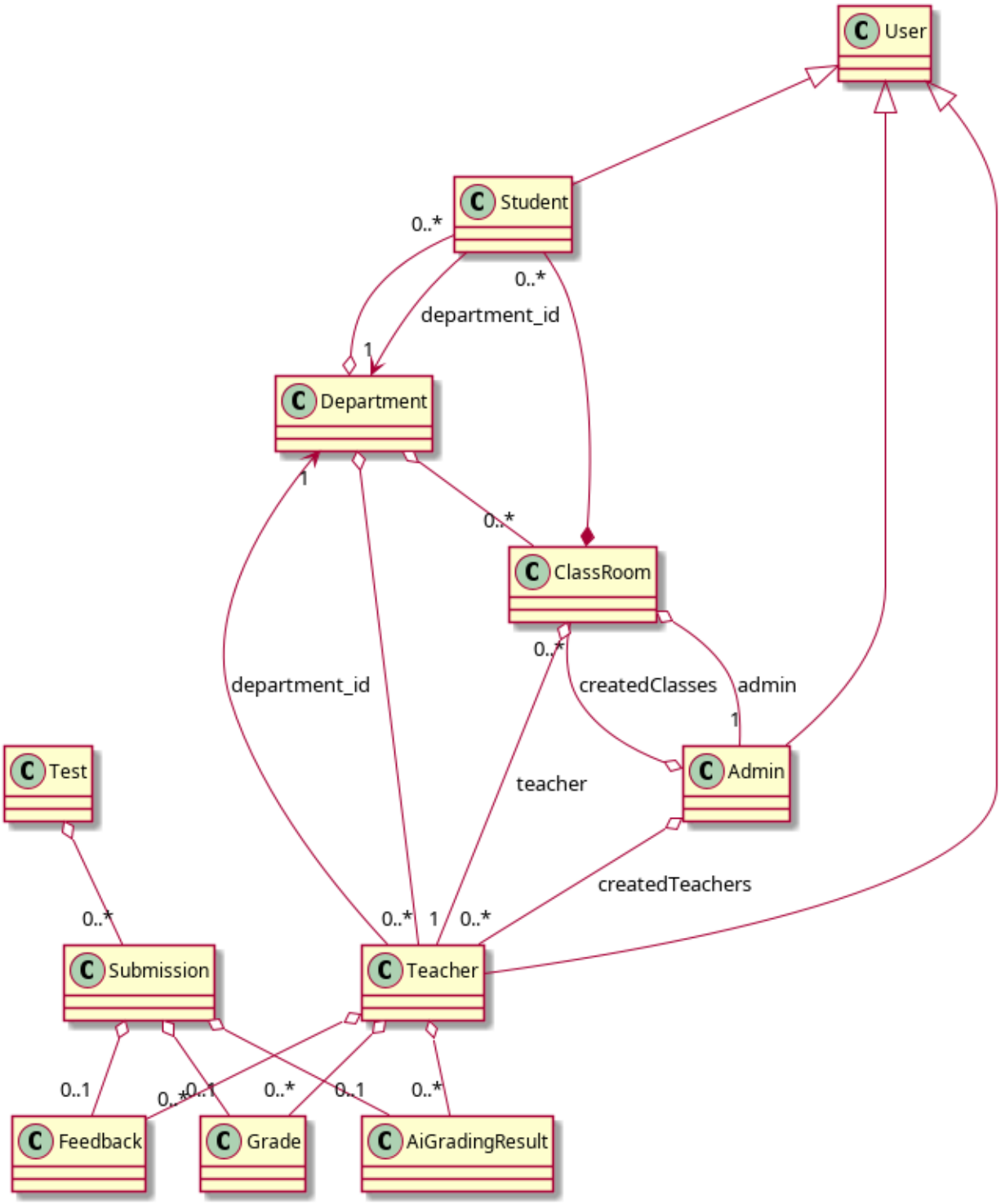

Figure 23. Class Diagrams For User Facing Modules (Relationships Only)

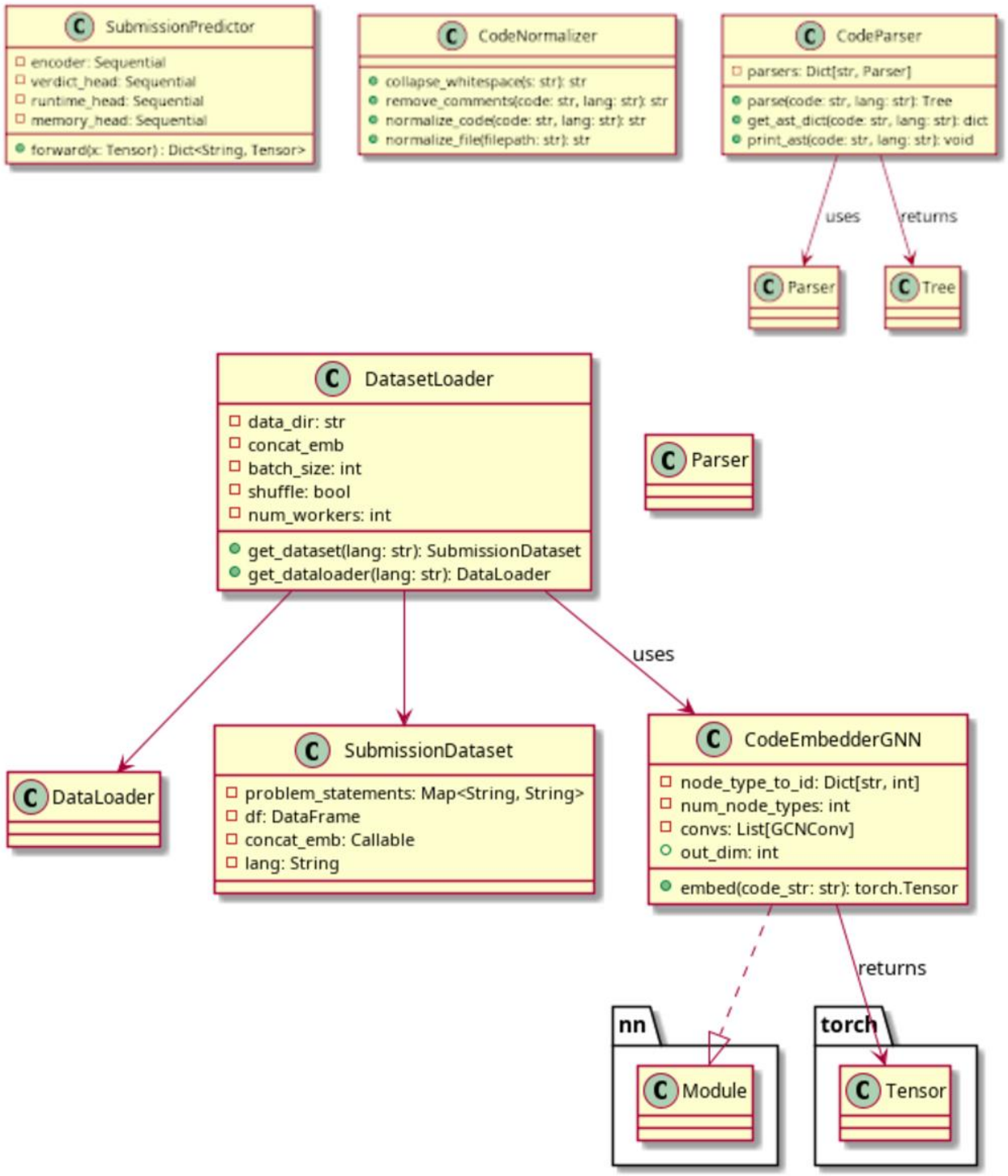

Figure 24. Class Diagram For ML Module

3.5. Model Validation

To ensure that the developed models accurately reflect the goals and functionality of the proposed system, a comprehensive validation process was conducted. This process evaluated the models in terms of correctness, completeness, consistency, and traceability to the requirements defined earlier in this chapter.

3.5.1. Alignment with Requirements

All models, including the use-case diagrams, sequence diagrams, activity diagrams, state machine diagrams, collaboration diagrams, and class diagrams, were cross-referenced with the functional and non-functional requirements specified in Sections 3.3.1 and 3.3.2. Each requirement is either directly or indirectly represented within the models, ensuring full coverage of the intended system behavior.

3.5.2. Internal Consistency

The interactions shown in the dynamic models (Section 3.3.4) are consistent with the structural definitions provided in the class diagram (Section 3.4.2). For instance, method calls and object interactions illustrated in the sequence and collaboration diagrams correspond to operations defined in the relevant classes. This consistency helps ensure that both behavior and structure are harmonized.

3.5.3. Completeness and Realism

The models collectively represent the complete functionality of the proposed system, including key use cases such as user authentication, code submission, feedback generation, and assignment review. Common and alternative flows are considered, and exception scenarios such as invalid input or system errors are also modeled where relevant.

In conclusion, the system models presented in this chapter have been validated to effectively represent the proposed solution. They offer a solid foundation for the subsequent stages of system design and implementation while acknowledging certain limitations discussed earlier in the chapter.

Chapter Four: System Design

4.1. Overview

The system design converts the objectives of the project into a technical plan that leads the development team to create a system capable of fulfilling the requirements of the project and expectations of stakeholders. This enables the team to break down the project to accurately logical chunks and helps in a proper and effective task breakdown, which in turn helps the teams to have clear ideas of what to implement, and how to organize the system. It also helps people outside the project to use it as a blueprint in understanding how the project works and how it's been put together. And those people can then go on to make contributions, provide critiques, improvements and much more to the developers of the system.

4.2 Specifying the Design Goals

4.2.1. Performance and Scalability

The system is designed to analyze student code submissions quickly and efficiently, supporting smooth interactions even under typical classroom loads. It can handle simultaneous users without noticeable slowdown and is built to accommodate increasing demand as the number of students and submissions grows.

4.2.2. Security and Privacy

Basic security principles are applied throughout the system to safeguard user data and prevent unauthorized access. These include secure authentication, role-based access control, strict input validation, and encryption of sensitive information. Uploaded code files are verified to prevent misuse, and user roles are restricted to their appropriate privileges.

4.2.3. Reliability and Availability

While the system aims for high availability, it may be subject to occasional disruptions due to the limitations of its hosting environment. To mitigate downtime during critical periods such as assessments, automatic recovery mechanisms are in place to restore normal operation within a short time when issues occur. The design emphasizes resilience and quick recovery over guaranteed uptime.

4.2.4. Usability

The system prioritizes clarity and ease of use in its interface design, especially for students submitting assignments and teachers reviewing feedback. While not fully aligned with formal accessibility standards, the interface is straightforward and functional for typical classroom scenarios. Feedback and insights are delivered in an organized, readable format.

4.2.5. Machine Learning Integration

Code evaluation is supported by machine learning models that analyze submissions for correctness, structure, and code quality. These models are integrated into the system in a way that keeps them separate from direct user interaction. Their role is to generate automated feedback, support assignment grading, and provide code insights without exposing internal complexity to users.

4.2.6. Interactivity and Feedback Presentation

Feedback is presented in a student-friendly format, with visual cues and annotations that help users understand what went wrong and how to improve. Teachers can view summary statistics and patterns across submissions. The goal is to enhance learning through interactive, actionable feedback rather than static responses.

4.2.7 Maintainability and Evolution

The system uses a modular structure that separates user management, submissions, and feedback logic. This separation makes it easier to update or extend parts of the system independently. Continuous monitoring and user feedback are used to guide future improvements and feature additions, supporting long-term usability and effectiveness.

4.3 System Design

This section details the proposed software architecture, subsystem decomposition, database design, deployment strategy, user interface, integration plan, and security measures. The design prioritizes modularity, interoperability, and user-centered principles to create a scalable and engaging platform.

4.3.1 Proposed system Architecture

Our project is structured around a two-module system. This Architecture is aimed to separate the core application logic and user interaction from the ML components. The approach will simplify development and facilitate deployment. It will also help maintain a clear division of responsibilities.

The two modules will be:

- **User-Facing module:** This module will consist of both the frontend and backend functionalities of the application. It should handle all user interactions such as authentication, uploading of code solutions, teachers' creation of code problems etc. It should also handle management of data in the database. It will include a Laravel backend with react components as frontend, inertia.js integrating the two, and postgresql for database.
- **ML module:** this module will be dedicated to the machine learning models, and it will be responsible for code analysis and feedback generation. It will include a set of ml models trained via pytorch, with their state saved into pytorch saving files. And there it will be wrapped in a flask backend's python virtual environment, where all the needed python dependencies will be included for them.

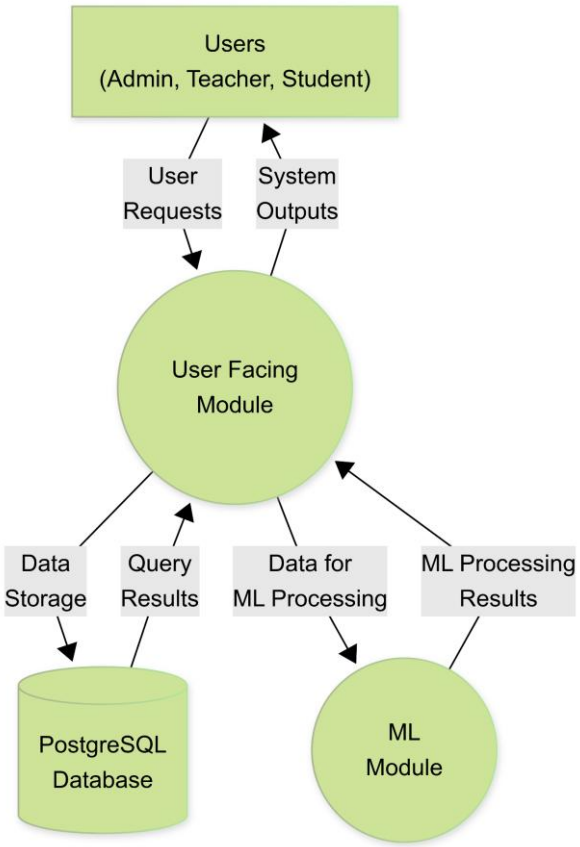

Figure 25. Overview of the Whole System

4.3.2 Subsystem decomposition

The two modules will be stand-alone and independent, and they will communicate with each other via predefined APIs. Generally the system will be organized as follows:

- **User-Facing Module:**
  - Will be built with Laravel, React, and Inertia.
  - Laravel will handle:
    - Application logic.
    - Data management (using PostgreSQL).
    - Routing.
    - API endpoints for communication with the ML Module.
  - React and Inertia handle:
    - User interface.

        - User interactions.
        - Displaying data received from Laravel.
- **ML Module:**
    - Will be built with Flask.
    - Will host the trained machine learning models.
    - Will provide API endpoints for the User-Facing Module to send code submissions and receive feedback.

1. User facing module subsystems decomposition

The User facing module will integrate several technologies to manage user interactions, business/application logic, and data persistence.

- **Database Integration:** the Laravel backend will connect to a PostgreSQL database running on port 5432, to store and retrieve data. The database connection details or credentials such as username, password and database name are managed via environment variables in the .env file. Laravel's migration files define the database schema and table structures, while Eloquent ORM provides an Object-Oriented interface that abstracts database interactions.
- **Data Management with Eloquent ORM:** Laravel makes use of Eloquent ORM to abstract away the database operations. Eloquent allows developers to interact with database tables as objects, which simplifies data manipulation actions such as insertion, deletion and updates. It also allows for expressive and easier representation of complex database queries and features such as joins, cascades and constraints. Eloquent also manages relations such as one-to-one, one-to-many and many-to-many.
- **Frontend Integration with React and Inertia:** Laravel will expose routes that will serve React.js components for the user interface. Inertia.js will facilitate the rendering of these React components and it will manage the data transfer between the Laravel backend and the React frontend. Typescript language is specially used to define and enforce ui components and specially data types, while Typescript interfaces are defined to correspond to the Laravel backend's data models. Inertia will handle converting the data from PHP models to Typescript objects so that they could be used in the React frontend.
- **MVC Pattern in Laravel:** While the user-facing module combines frontend and backend, Laravel's Model-View-Controller (MVC) pattern is still relevant. Laravel's models manage data while its controllers manage application logic. React components then function as the "views", rendering data and sending requests to the controllers via the web routes.

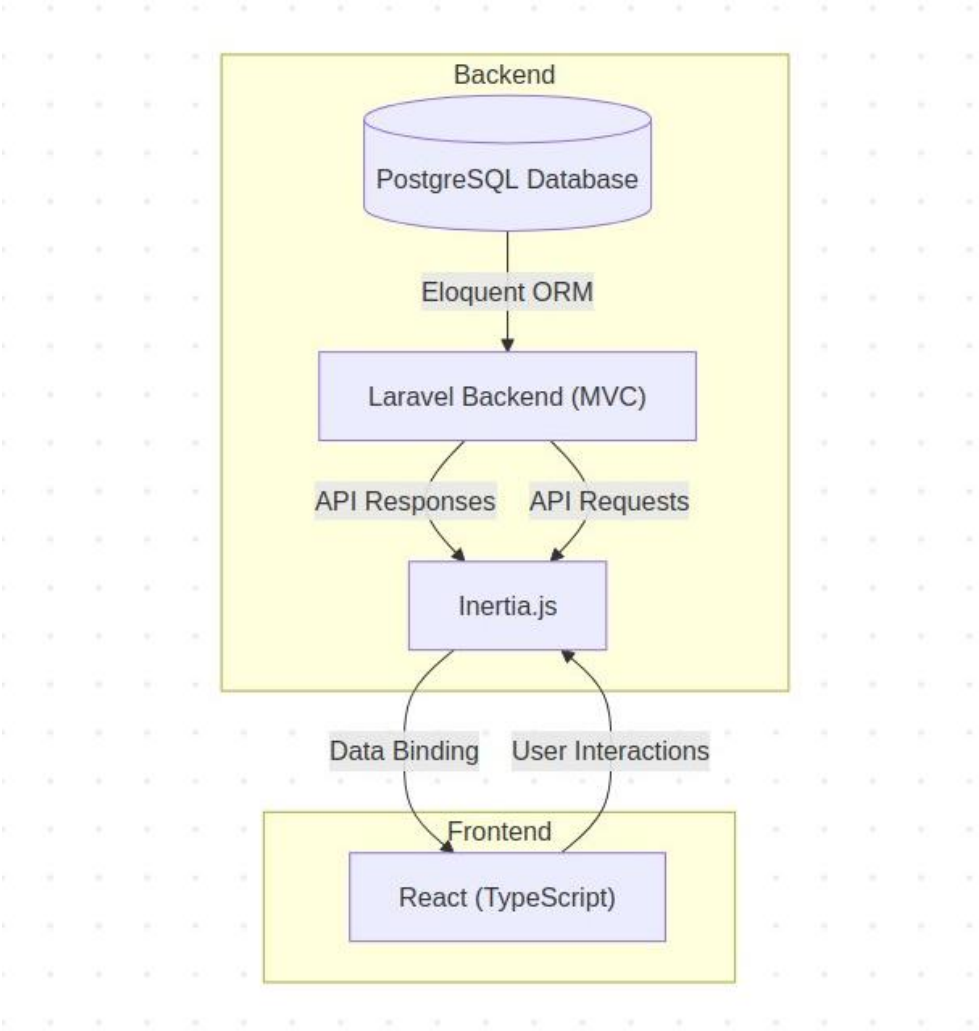

Figure 26. Diagram of the user facing model and its interactions.

2. ML-facing module

The ML Facing Module is a core subsystem of the Automated Code Review System within our elearning platform. Its role is to analyze student code submissions in the context of their accompanying problem descriptions and to predict the likely outcome or "verdict" of each submission (for example, Accepted, Wrong Answer, Time Limit Exceeded). To accomplish this, the module employs a two-pronged approach: natural language processing techniques for understanding the wording and requirements of problem statements and graph neural networks for capturing the structural and semantic intricacies of source code. The module does not persist any data itself; rather, it relies on pretrained components and, upon receiving a new submission, immediately processes the input to yield a probabilistic verdict.

- **Preprocessor**: This utility component takes raw text from problem statements—including descriptions, input specifications, and output specifications, and applies standard NLP cleaning steps: converting text to lowercase, splitting it into tokens, removing stop words, and optionally applying stemming or lemmatization. The sanitized text is then handed off to the TextEmbedder, which has been previously trained on the full corpus of problem statements. The TextEmbedder uses a TFIDF algorithm to transform the text into a fixed length vector representation that highlights the most significant terms relative to the entire dataset.
- **CodeNormalizer**: Strips out nonessential elements such as comments and extraneous whitespace to ensure that only the syntactically relevant portions of the code remain. The normalized code is then fed into the CodeParser, which leverages the Tree sitter library to construct an Abstract Syntax Tree (AST). That AST is converted into a graph data structure, where each node corresponds to a syntax element and each edge represents a parent child relationship in the code's structure.
- **CodeEmbedderGNN**: It takes this graph and computes a dense continuous embedding of the entire code snippet. Internally, it maps AST node types to learnable vectors, applies graph convolutional layers to mix information across the tree, and then uses a global pooling operation to collapse the node level features into a single vector. This code embedding captures both the local syntax patterns and the overall structure of the program.
- **ConcatEmbedder**: Fuses the two vectors (code and text) into a unified feature vector. This combined representation then serves as the input to the SubmissionPredictor, the module's main neural network. The predictor consists of an MLP (multi-layer perceptron) with one or more hidden layers, with activation functions and dropout for regularization, and finally a linear verdict head layer that outputs raw logits for the set of possible verdict classes.

Although most of these components are wrapped in Python classes and instantiate dynamically, their interconnections are fixed: the SubmissionDataset class orchestrates the loading and preparation of each sample (invoking the Preprocessor and CodeNormalizer), while the SubmissionPredictor class bundles the ConcatEmbedder, TextEmbedder, and CodeEmbedderGNN into a single forward pass.

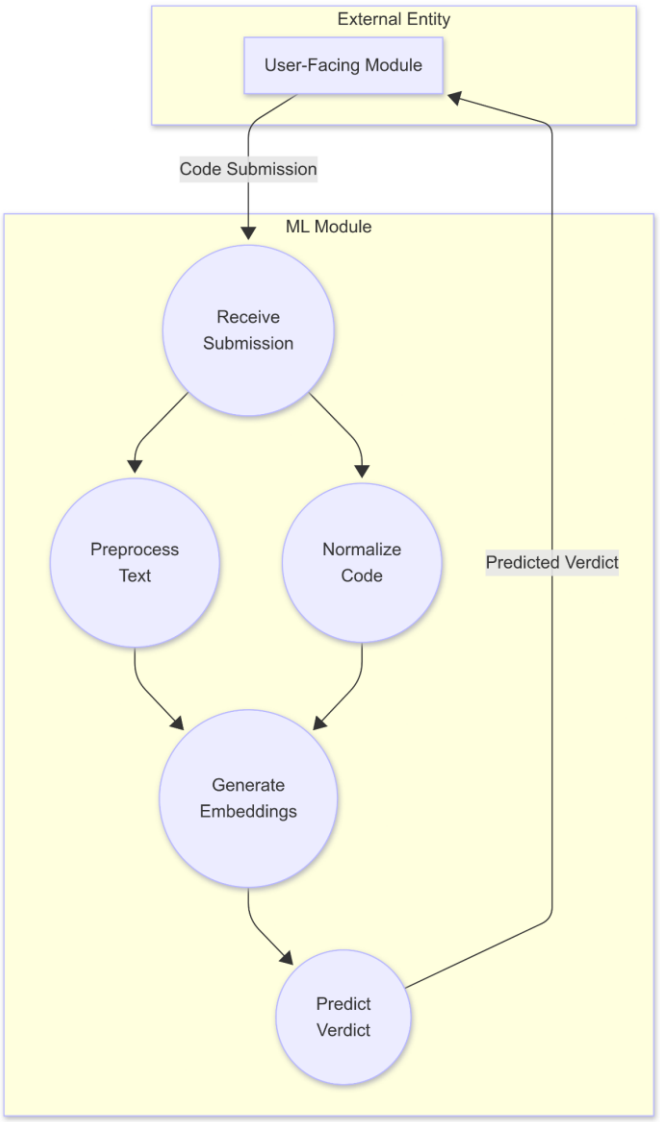

Figure 27. Data Flow Diagram of the ML Module

3. Model Parameters

The **ML Facing Module** predicts the outcome of student code submissions by analyzing both the problem description and the submitted code. It does this using two separate embedding pipelines, one for text and one for code, which are then merged and passed through a multi-layer neural network for final classification. Below is a breakdown of the key architectural settings used in the model.

- **Text Embedding:** Problem descriptions are embedded using a TF-IDF-based representation trained on the entire dataset. Only the **top 4,000** most relevant textual features are retained to ensure a compact and meaningful vector.
- **Code Embedding via Graph Neural Network (GNN):** Submitted code is parsed into an Abstract Syntax Tree (AST), transformed into a graph, and then encoded using a multi-layer GNN:
    - Up to **2,000** distinct AST node types are supported.
    - Each node is embedded into a **128-dimensional** vector space.
    - The GNN includes **3 layers** of message passing, with an internal hidden size of **256 dimensions**.
    - The final output of the GNN is a **128-dimensional** vector representing the full code snippet.
- **Embedding Fusion:** The code and text embeddings are concatenated into a single representation. A projection layer is applied to the combined vector:
    - The projection step is **enabled** to enhance compatibility between code and text representations.
    - The combined vector is scaled by a factor of **0.4** before being passed to the prediction network.
- **Verdict Classification:** The final fused representation is input to a multi-layer perceptron (MLP) that predicts one of several possible outcomes (verdicts):
    - The classifier outputs probabilities across **7 distinct verdict classes**.
    - It uses **4 hidden layers** with dimensions: **256 → 256 → 128 → 64** before reaching the output layer.

4.3.3 Database Design

The database design of the system is implemented in a way to ensure efficient, secure and scalable management of persistent data to support the core functionalities of the system. The schema implementation includes entities, constraints and relationships, which are defined considering optimized data access.

4.3.3.1. Entities, attributes, relationships and constraints

The database consists of the following entities, with their attributes, key constraints and relationships, designed to support the platform's functionality and user roles within the context of the Laravel framework

1. **Users**: Stores user data and authentication details.
    - Constraints: id (primary key), email (unique, indexed).

- Attributes: id, first_name, last_name, email, email_verified_at (timestamp, nullable), password (hashed), remember_token (string, nullable), created_at, updated_at.
- Relationships:
  - One-to-One with Admins (Foreign Key(s): admins.user_id → users.id). Each user can optionally be associated with an admin profile.
  - One-to-One with Students (Foreign Key(s): students.user_id → users.id). Each user can optionally be associated with a student profile.
  - One-to-One with Teachers (Foreign Key(s): teachers.user_id → users.id). Each user can optionally be associated with a teacher profile.
  - Many-to-Many with Roles (Through: model_has_roles pivot table, Foreign Key(s): model_has_roles.model_id → users.id, model_has_roles.role_id → roles.id). Users can have multiple roles.
  - Many-to-Many with Permissions (Through: model_has_permissions pivot table, Foreign Key(s): model_has_permissions.model_id → users.id, model_has_permissions.permission_id → permissions.id). Users can have direct permissions.
  - One-to-Many with Classes (Foreign Key(s): classes.created_by → users.id). Users can create classes.
  - One-to-Many with Personal Access Tokens (Polymorphic, Foreign Key(s): personal_access_tokens.tokenable_id → users.id, personal_access_tokens.tokenable_type → 'App\Models\User'). Users can have multiple personal access tokens.

2. **Roles**: Defines user roles within the system (e.g., 'admin', 'teacher', 'student').
   - Constraints: id (primary key), name (unique, indexed).
   - Attributes: id, name, guard_name (string, e.g., 'web'), created_at, updated_at.
   - Relationships:
     - Many-to-Many with Users (Through: model_has_roles pivot table, Foreign Key(s): model_has_roles.role_id → roles.id, model_has_roles.model_id → users.id). Roles can be assigned to multiple users.
     - Many-to-Many with Permissions (Through: role_has_permissions pivot table, Foreign Key(s): role_has_permissions.role_id → roles.id,

role_has_permissions.permission_id → permissions.id). Roles have multiple associated permissions.

3. **Permissions**: Defines specific actions users are allowed to perform.
    - Constraints: id (primary key), name (unique, indexed).
    - Attributes: id, name, guard_name, created_at, updated_at.
    - Relationships:
        - Many-to-Many with Roles (Through: role_has_permissions pivot table, Foreign Key(s): role_has_permissions.permission_id → permissions.id, role_has_permissions.role_id → roles.id). Permissions can be assigned to multiple roles.
        - Many-to-Many with Users (Through: model_has_permissions pivot table, Foreign Key(s): model_has_permissions.permission_id → permissions.id, model_has_permissions.model_id → users.id). Permissions can be directly assigned to users.
4. **Admins**: Stores specific administrator profile information linked to a User.
    - Constraints: id (primary key), user_id (unique, foreign key to Users), created_by (foreign key to Admins).
    - Attributes: id, user_id, created_by, created_at, updated_at.
    - Relationships:
        - One-to-One with Users (Foreign Key(s): admins.user_id → users.id). Each admin profile belongs to exactly one user.
        - Many-to-One with Admins (Foreign Key(s): admins.created_by → admins.id). An admin can be created by another admin.
        - One-to-Many with Teachers (Foreign Key(s): teachers.created_by → admins.id). Admins can create teacher records.
        - One-to-Many with Classes (Foreign Key(s): classes.admin_id → admins.id). Classes can be associated with an admin.
5. **Departments**: Represents academic departments.
    - Constraints: id (primary key), name (unique).
    - Attributes: id, name, created_at, updated_at.
    - Relationships:

        - One-to-Many with Students (Foreign Key(s): students.department_id → departments.id). A department has many students.
        - One-to-Many with Teachers (Foreign Key(s): teachers.department_id → departments.id). A department has many teachers.
        - One-to-Many with Classes (Foreign Key(s): classes.department_id → departments.id). A department offers many classes.
        - Many-to-Many with Tests (Through: test_departments pivot table, Foreign Key(s): test_departments.department_id → departments.id, test_departments.test_id → tests.id). Departments can be associated with multiple tests.
6. **Students**: Stores specific student profile information linked to a User.
    - Constraints: id (primary key), user_id (unique, foreign key to Users), department_id (foreign key to Departments).
    - Attributes: id, user_id, id_number (string), academic_year (string), department_id, created_at, updated_at.
    - Relationships:
        - One-to-One with Users (Foreign Key(s): students.user_id → users.id). Each student profile belongs to exactly one user.
        - Many-to-One with Departments (Foreign Key(s): students.department_id → departments.id). Students belong to one department.
        - Many-to-Many with Classes (Through: class_students pivot table, Foreign Key(s): class_students.student_id → students.id, class_students.class_id → classes.id). Students can be enrolled in many classes.
        - Many-to-Many with Tests (Through: test_student pivot table, Foreign Key(s): test_student.student_id → students.id, test_student.test_id → tests.id). Students can participate in many tests.
        - One-to-Many with Submissions (Foreign Key(s): submissions.student_id → students.id). A student can have many submissions.
        - One-to-Many with Analytics (Foreign Key(s): analytics.student_id → students.id). A student's performance data contributes to multiple analytics records.
7. **Teachers**: Stores specific teacher profile information linked to a User.
    - Constraints: id (primary key), user_id (unique, foreign key to Users), created_by (foreign key to Admins), department_id (foreign key to Departments).

- Attributes: id, user_id, created_by, department_id, created_at, updated_at.
- Relationships:
  - One-to-One with Users (Foreign Key(s): teachers.user_id → users.id). Each teacher profile belongs to exactly one user.
  - Many-to-One with Admins (Foreign Key(s): teachers.created_by → admins.id). Teachers are created by an admin.
  - Many-to-One with Departments (Foreign Key(s): teachers.department_id → departments.id). Teachers belong to one department.
  - One-to-Many with Classes (Foreign Key(s): classes.teacher_id → teachers.id). A teacher can teach many classes.
  - One-to-Many with Tests (Foreign Key(s): tests.teacher_id → teachers.id). A teacher can create many tests.
  - One-to-Many with Grades (Foreign Key(s): grades.teacher_id → teachers.id). A teacher can assign many grades.
  - One-to-Many with Feedback (Foreign Key(s): feedbacks.teacher_id → teachers.id). A teacher can provide many feedback entries.

8. **Classes**: Represents academic classes or courses.
   - Constraints: id (primary key), department_id (foreign key to Departments), teacher_id (foreign key to Teachers), admin_id (foreign key to Admins), created_by (foreign key to Users).
   - Attributes: id, name, department_id, teacher_id, admin_id, max_students (integer), created_by, created_at, updated_at.
   - Relationships:
     - Many-to-One with Departments (Foreign Key(s): classes.department_id → departments.id). Classes belong to one department.
     - Many-to-One with Teachers (Foreign Key(s): classes.teacher_id → teachers.id). Classes are taught by one teacher.
     - Many-to-One with Admins (Foreign Key(s): classes.admin_id → admins.id). Classes are associated with an admin.
     - Many-to-One with Users (Foreign Key(s): classes.created_by → users.id). Classes are created by a user.

        - Many-to-Many with Students (Through: class_students pivot table, Foreign Key(s): class_students.class_id → classes.id, class_students.student_id → students.id). Classes have many enrolled students.
        - One-to-Many with Tests (Foreign Key(s): tests.class_id → classes.id). A class can have many tests associated with it.
9. **Class Students**: Pivot table for the Many-to-Many relationship between Classes and Students.
    - Constraints: id (primary key), class_id (foreign key to Classes), student_id (foreign key to Students).
    - Attributes: id, class_id, student_id, assigned (boolean, default true), created_at, updated_at.
    - Relationships:
        - Many-to-One with Classes (Foreign Key(s): class_students.class_id → classes.id). Each record links to one class.
        - Many-to-One with Students (Foreign Key(s): class_students.student_id → students.id). Each record links to one student.
10. **Tests**: Represents programming tests or assignments.
    - Constraints: id (primary key), teacher_id (foreign key to Teachers), class_id (foreign key to Classes).
    - Attributes: id, teacher_id, class_id, title, problem_statement (text), status (enum: 'Upcoming', 'Done'), due_date (date), metrics (json), created_at, updated_at.
    - Relationships:
        - Many-to-One with Teachers (Foreign Key(s): tests.teacher_id → teachers.id). Tests are created by one teacher.
        - Many-to-One with Classes (Foreign Key(s): tests.class_id → classes.id). Tests are associated with one class.
        - Many-to-Many with Students (Through: test_student pivot table, Foreign Key(s): test_student.test_id → tests.id, test_student.student_id → students.id). Tests are assigned to many students.
        - Many-to-Many with Departments (Through: test_departments pivot table, Foreign Key(s): test_departments.test_id → tests.id, test_departments.department_id → departments.id). Tests can be associated with multiple departments.

        - One-to-Many with Submissions (Foreign Key(s): submissions.test_id → tests.id). A test can receive many submissions.
        - One-to-Many with Analytics (Foreign Key(s): analytics.test_id → tests.id). A test's results contribute to multiple analytics records.

11. **Test Students**: Pivot table for the Many-to-Many relationship between Tests and Students, potentially tracking assignment status.
    - Constraints: id (primary key), test_id (foreign key to Tests), student_id (foreign key to Students).
    - Attributes: id, test_id, student_id, created_at, updated_at.
    - Relationships:
        - Many-to-One with Tests (Foreign Key(s): test_student.test_id → tests.id). Each record links to one test.
        - Many-to-One with Students (Foreign Key(s): test_student.student_id → students.id). Each record links to one student.
12. **Submissions**: Stores student code submissions for tests.
    - Constraints: id (primary key), test_id (foreign key to Tests), student_id (foreign key to Students).
    - Attributes: id, test_id, student_id, submission_type (enum: 'file', 'editor'), code_file_path (string, nullable), code_editor_text (text, nullable), submission_date (date), status (enum: 'pending', 'reviewed', 'graded'), created_at, updated_at.
    - Relationships:
        - Many-to-One with Tests (Foreign Key(s): submissions.test_id → tests.id). Submissions are made for one test.
        - Many-to-One with Students (Foreign Key(s): submissions.student_id → students.id). Submissions are made by one student.
        - One-to-One with AI Grading Result (Foreign Key(s): ai_grading_results.submission_id → submissions.id). A submission has one AI grading result.
        - One-to-Many with Feedback (Foreign Key(s): feedbacks.submission_id → submissions.id). A submission can have multiple feedback entries (AI and/or Teacher).

        - One-to-One with Grades (Foreign Key(s): grades.submission_id → submissions.id). A submission receives one final grade.

13. **Feedback**: Stores feedback entries for submissions (from AI or Teacher).
    - Constraints: id (primary key), submission_id (foreign key to Submissions), teacher_id (foreign key to Teachers, nullable).
    - Attributes: id, submission_id, teacher_id, feedback_text (text), annotations (text), created_at, updated_at.
    - Relationships:
        - Many-to-One with Submissions (Foreign Key(s): feedbacks.submission_id → submissions.id). Feedback is given for one submission.
        - Many-to-One with Teachers (Foreign Key(s): feedbacks.teacher_id → teachers.id). Feedback can be provided by a teacher (nullable for AI feedback).
14. **AI Grading Result**: Stores the automated grading output from the AI model for a submission.
    - Constraints: id (primary key), submission_id (unique, foreign key to Submissions).
    - Attributes: id, submission_id, metrics (json), comment (text), graded_value (float), score (decimal, nullable), raw_output (json or text, storing detailed AI analysis), analysisJson (json or text, potentially structured analysis), processed_feedback (text, human-readable feedback from AI), grading_time (timestamp, nullable), created_at, updated_at.
    - Relationships:
        - One-to-One with Submissions (Foreign Key(s): ai_grading_results.submission_id → submissions.id). An AI grading result corresponds to one submission.
15. **Grades**: Stores the final grade assigned to a submission after teacher review.
    - Constraints: id (primary key), submission_id (unique, foreign key to Submissions), teacher_id (foreign key to Teachers).
    - Attributes: id, submission_id, teacher_id, graded_value (float), score (decimal, nullable), adjusted_grade (float), override_reason (text), comments (text), graded_at (timestamp, nullable), is_visible_to_student (boolean, default false), created_at, updated_at.
    - Relationships:
        - One-to-One with **Submissions** (Foreign Key(s): grades.submission_id → submissions.id). A grade is assigned to one submission.
        - Many-to-One with **Teachers** (Foreign Key(s): grades.teacher_id → teachers.id). A grade is assigned by one teacher.

16. **Analytics**: Stores aggregated or specific data points for reporting and visualization in the analytics dashboard.
    - Constraints: id (primary key), student_id (foreign key to Students), test_id (foreign key to Tests).
    - Attributes: id, student_id, test_id, metrics (json), language (string, e.g., 'C++', 'Python', nullable), score (decimal, nullable), completion_status (string, e.g., 'completed', 'incomplete', nullable), time_taken (integer, seconds, nullable), feedback_analysis (json or text, analysis of feedback points, nullable), performance_metrics (json or text, other relevant data, nullable), created_at, updated_at.
    - Relationships:
        - Many-to-One with Students (Foreign Key(s): analytics.student_id → students.id). Analytics records are tied to one student.
        - Many-to-One with Tests (Foreign Key(s): analytics.test_id → tests.id). Analytics records are tied to one test.
17. **Test Departments**: Pivot table for the Many-to-Many relationship between Tests and Departments.
    - Constraints: id (primary key), test_id (foreign key to Tests), department_id (foreign key to Departments).
    - Attributes: id, test_id, department_id, created_at, updated_at.
    - Relationships:
        - Many-to-One with Tests (Foreign Key(s): test_departments.test_id → tests.id). Each record links to one test.
        - Many-to-One with Departments (Foreign Key(s): test_departments.department_id → departments.id). Each record links to one department.

4.3.3.2 Database Diagram

Based on the database entities, relationships and constraints specified above, a database entity-relationship diagram of the following shape is obtained.

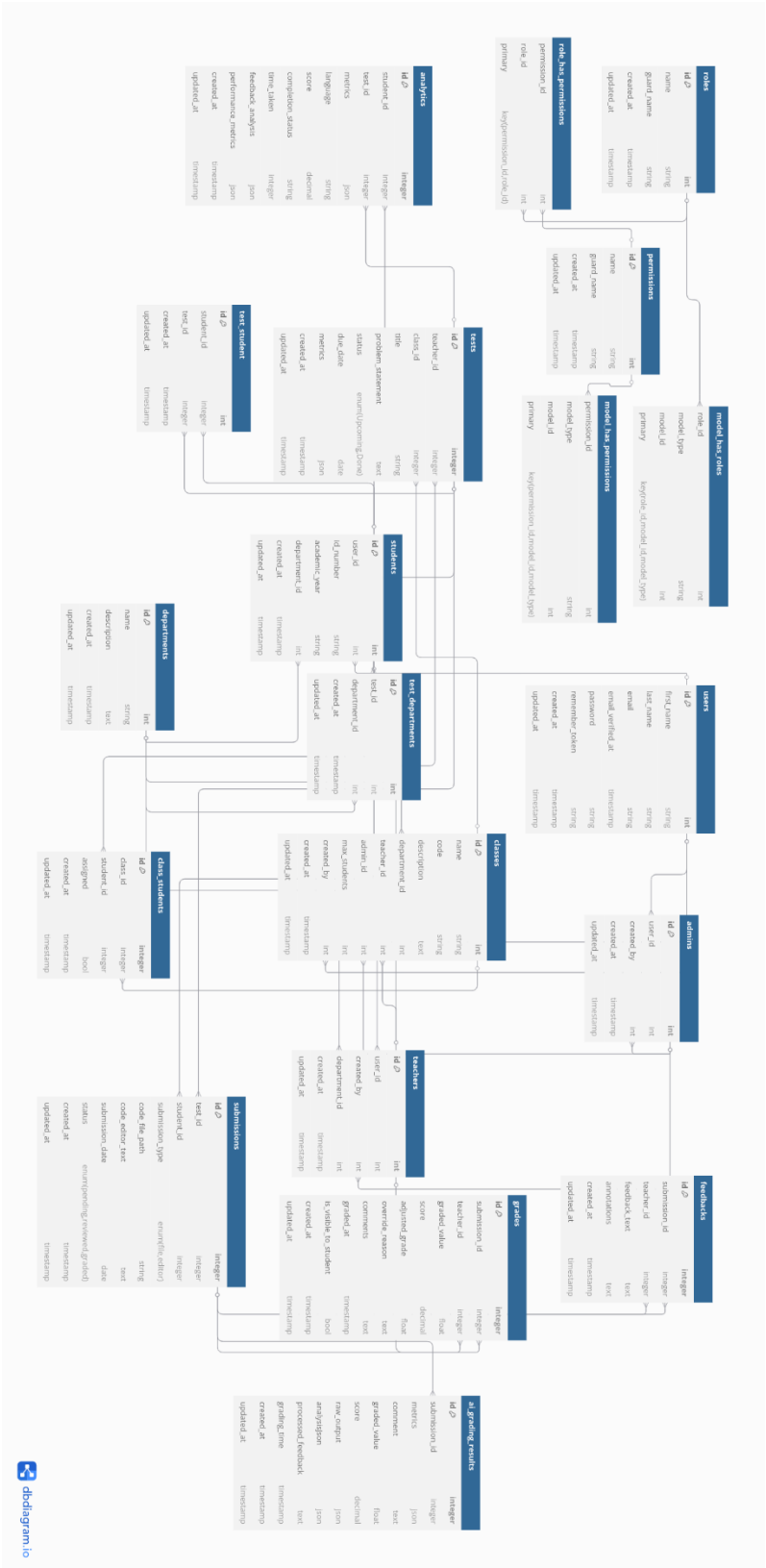

Figure 28. Database ER diagram

4.3.4 Deployment Diagram

This section elaborates on the deployment architecture of the system, illustrating the physical arrangement of system components on the Render platform. The system is planned to be deployed across two separate Render instances; one for instance being for the User-Facing module and one for the Ml Module. Separating the modules into these two deployment instances will allow for independent scaling, resource allocation and management of each module.

Deployment Architecture

Render Instance 1: User-Facing Module

Service Type: Web Service

Components:

- **Laravel Application:** Handles application logic, serves static assets, and communicates with the ML module through API calls.
- **React Frontend:** Part of the Laravel codebase, representing the frontend served by Laravel.

Runtime Environment:

- **Language:** PHP
- **Web Server:** Apache Server

Database:

- **Type:** PostgreSQL (Render Managed Database)

Description:

This Render instance will host the User-Facing Module, which includes the Laravel application and the integrated React frontend. It will handle user requests, manage data persistence in the PostgreSQL database, and interact with the ML module to obtain code analysis results.

Render Instance 2: ML Module

Service Type: Web Service

Components:

- **Flask Application:** Hosts the machine learning models and exposes an API for accessing them.
- **PyTorch Dependencies:** Includes the PyTorch library and any other necessary Python packages for running the ML models.
- **Trained ML Models and Weights:** Contains the trained machine learning model files and their associated weights.

Runtime Environment:

- **Language:** Python

Description:

This Render instance will host the ML Module, which will be a Flask application serving the trained machine learning models. It should receive code submissions from the User-Facing Module, perform analysis, and return the results via an API.

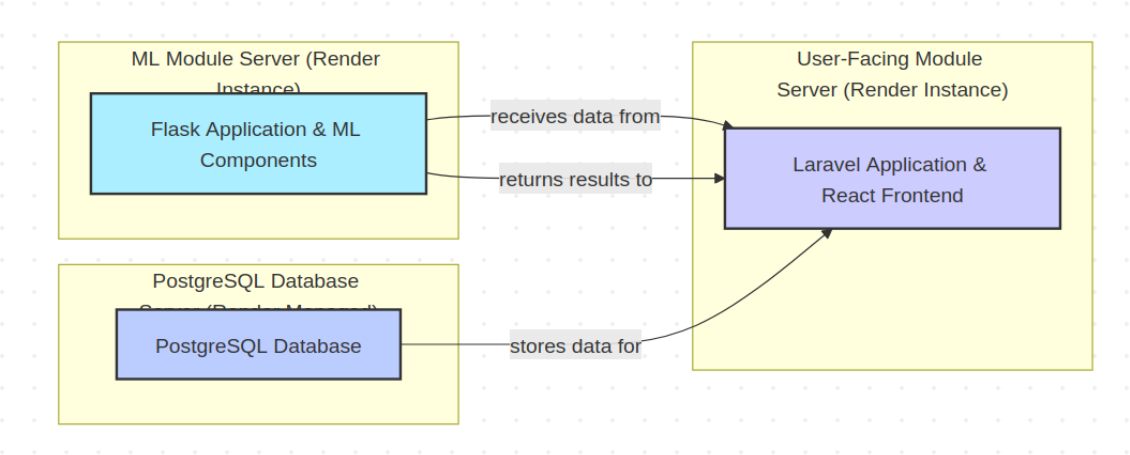

Figure 29. Deployment diagram

4.3.5. User Interface Design

The user interface design is performed on the basis of the system design and the underlying system requirements as well as project objectives. The motivation, then, of developing the user interface stems from the need to build an intuitive and seamless interface for the identified user roles, i.e. student, teacher and admin. This user centered approach tries to ensure that the interactions and user experience provide the

users an engaging experience that also helps them comfortably navigate the system, while also enjoying the benefits of the underlying machine learning based service that the system provides.

The user interface also emphasizes:

Clarity: all the labels, call to action, and links of the interface serve clear purposes, and are shown with indicator that clearly indicate their purpose

Consistency: The theme, colors, fonts, textual hierarchy, as well as navigation flow should be consistent throughout the application.

Efficiency: the application should have fast load times and less lag.

Responsiveness: the interface should adapt to varying screen sizes and displays, showing its contents consistently

These points ensure that the system aligns with the usability requirement that has been set forth in a previous section.

The user interface will focus mainly on the 3 user roles. That is, the student, the teacher and the system admin. In light of this, here is a break down of the user interface components for the 3 roles:

Student Interface

- **Student Dashboard:** This serves as the student's central hub. Its design prioritizes clarity and ease of access to essential information. It will prominently display progress tracking, showing the student's performance over time and across different assessments. It will also provide a clear view of upcoming and past assessments, including due dates and access to submitted results once graded.
- **Combined Problem/Submission Page:** This interface is designed to streamline the assessment process for students. It presents the problem statement clearly alongside the area for code input or file upload. This eliminates the need for students to navigate between separate pages to view the problem and submit their solution, ensuring a focused and efficient submission experience.

Teacher Interface

- **Teacher Dashboard:** This dashboard provides teachers with quick access to their classes, assessments, and key actions. It serves as the gateway to class and assessment management

functionalities, allowing teachers to navigate to areas for creating tests, viewing class rosters, and accessing student submissions.

- **Test Creation Interface:** Designed for efficiency, this interface allows teachers to define assessment details, add problem statements, set due dates, and configure grading criteria based on the grading criteria requirement.
- **Submission Review Interface:** This is a critical interface where teachers evaluate student work. It is specifically designed to display the student's submitted code or file alongside the original problem statement for easy reference. Crucially, it will also present the **AI-suggested grade and detailed feedback**, allowing the teacher to quickly understand the automated evaluation. The design ensures the **suggested grade is clearly editable**, empowering the teacher to apply their pedagogical judgment and finalize the grade before making it visible to the student. The **password editing feature** for teachers will also be accessible from their dashboard or profile settings within this interface.

Administrator Interface

- **Admin Dashboard:** This provides administrators with high-level oversight and access to system management tools. It is the central point for managing users and classes.
- **Teacher Registration Management:** This interface allows administrators to securely manage the registration process for new teachers, ensuring only authorized educators gain access to the platform.
- **Class Creation and Assignment Interfaces:** Administrators will use these interfaces to create new classes, assign teachers to specific classes, and manage the student population. The design will include a clear view for **bulk student assignment** and the ability to **view unassigned students**, simplifying the process of organizing students into classes according to the AASTU user scope.

Based on these specifications, we can provide a set of figa designs for our pages, for the admin, teacher and student roles.

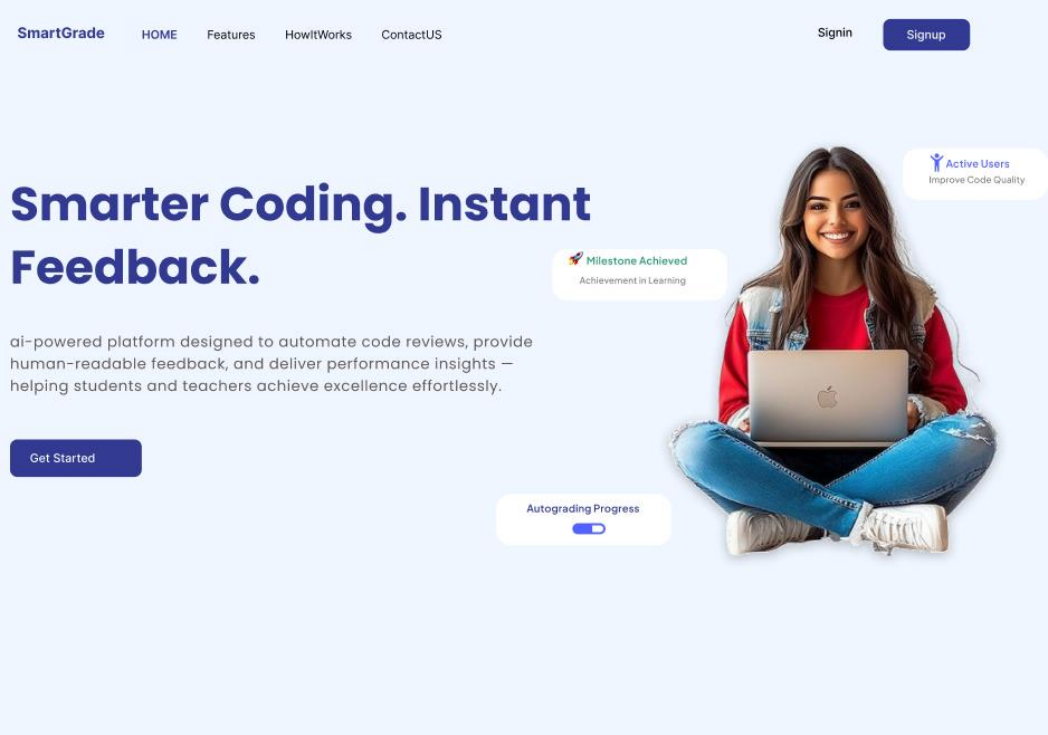

Figure 30. Hero Section of landing page

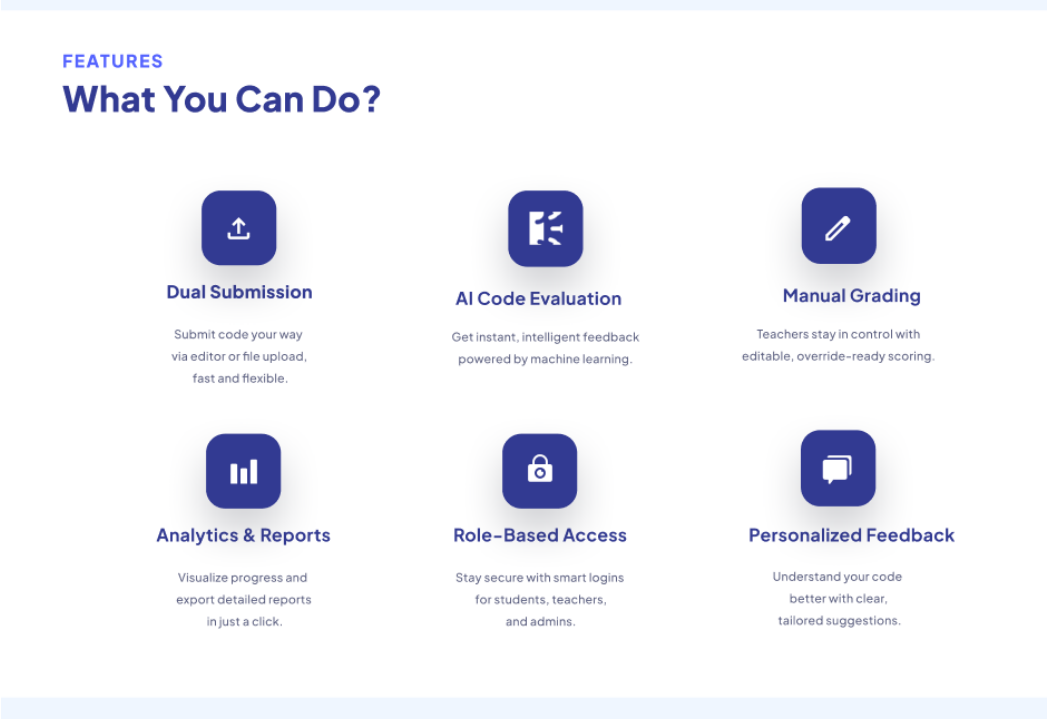

Figure 31. Sections of the landing page

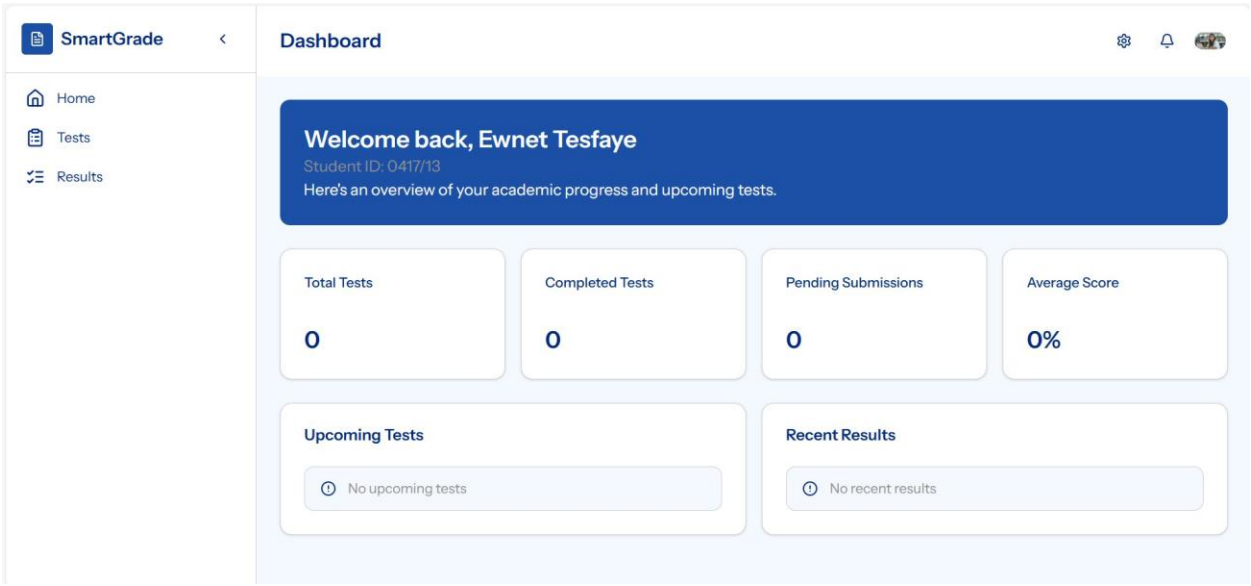

Figure 32. Student Dashboard

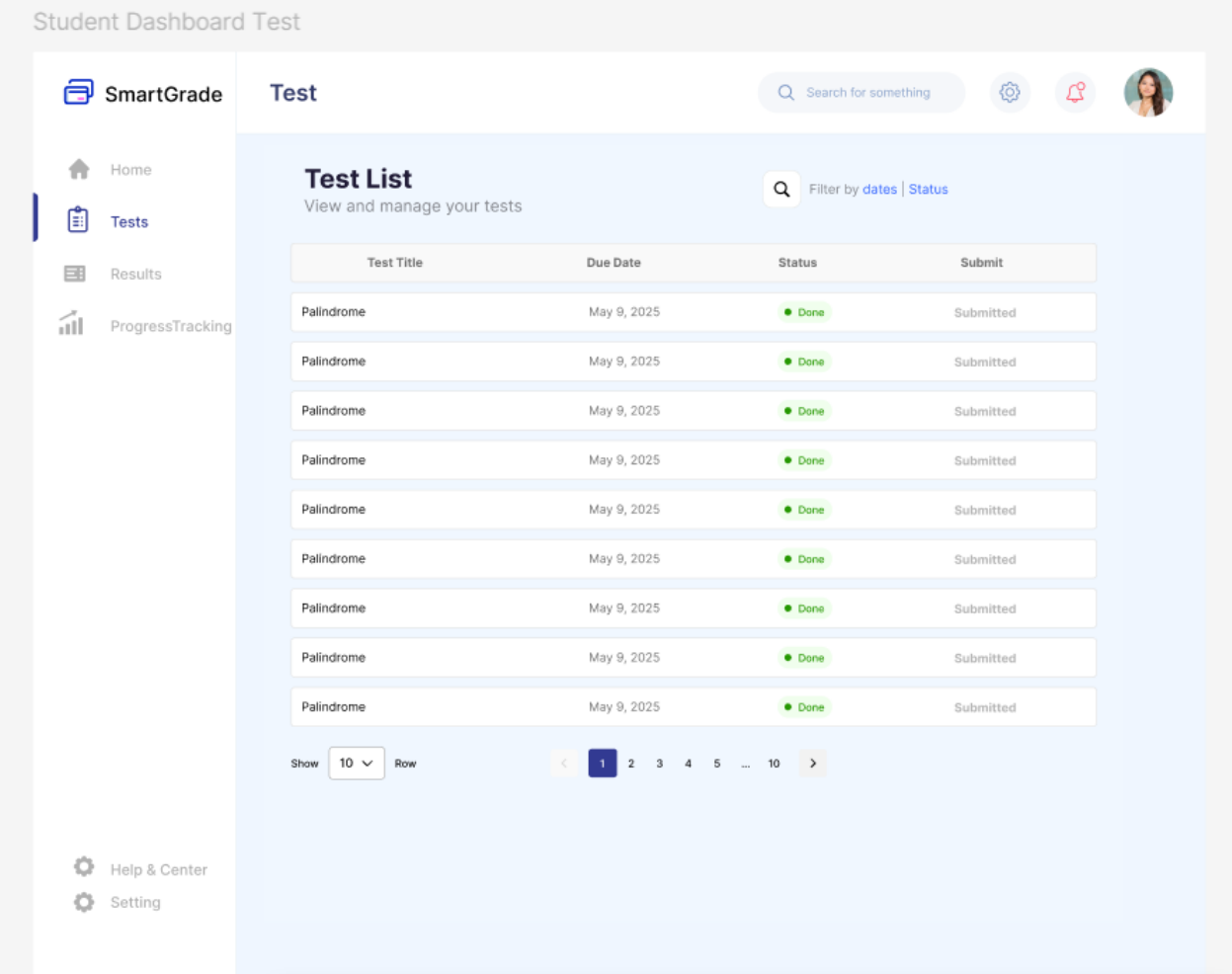

Figure 33. Student tests page

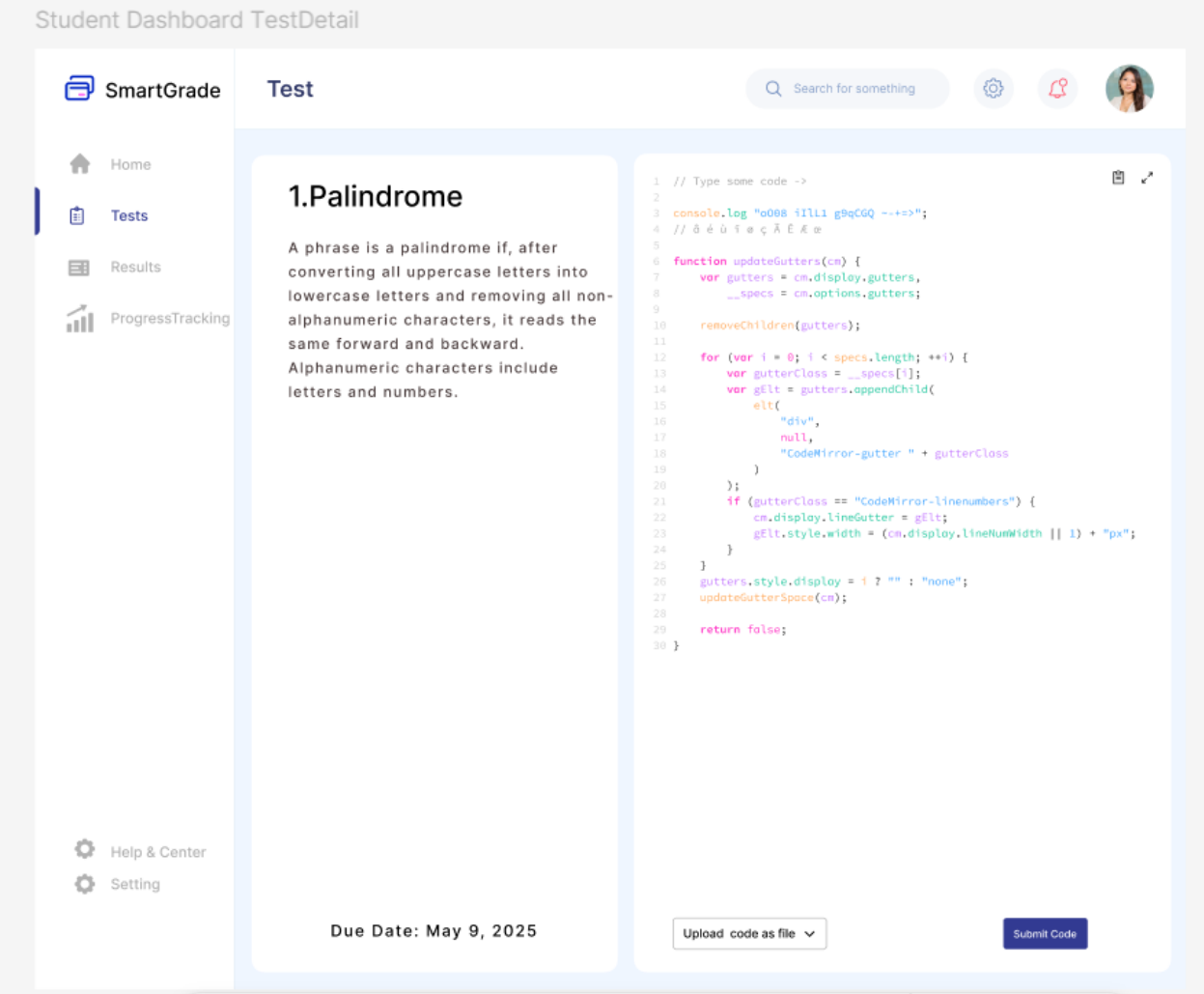

Figure 34. Students Code Submission Page

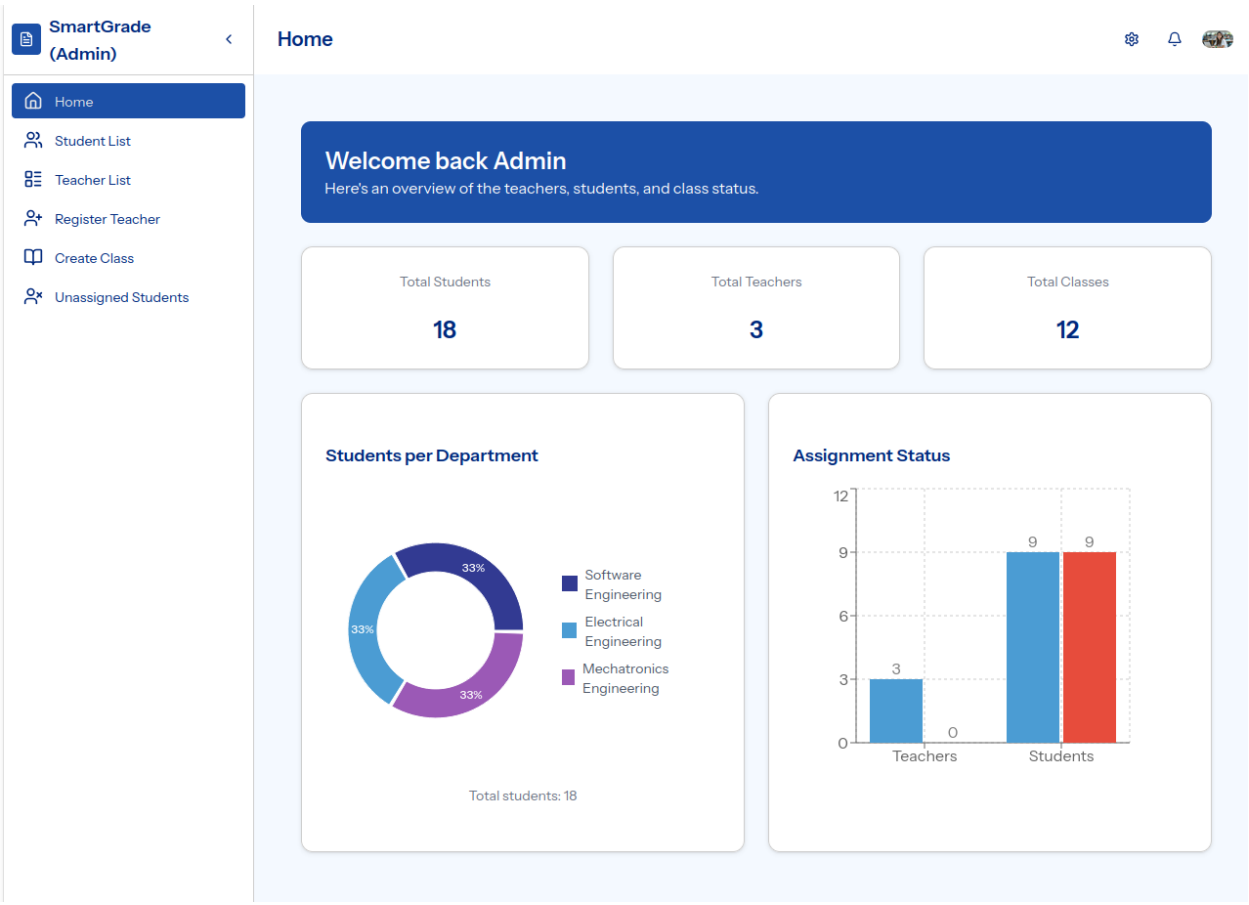

Figure 35. Admin dashboard page

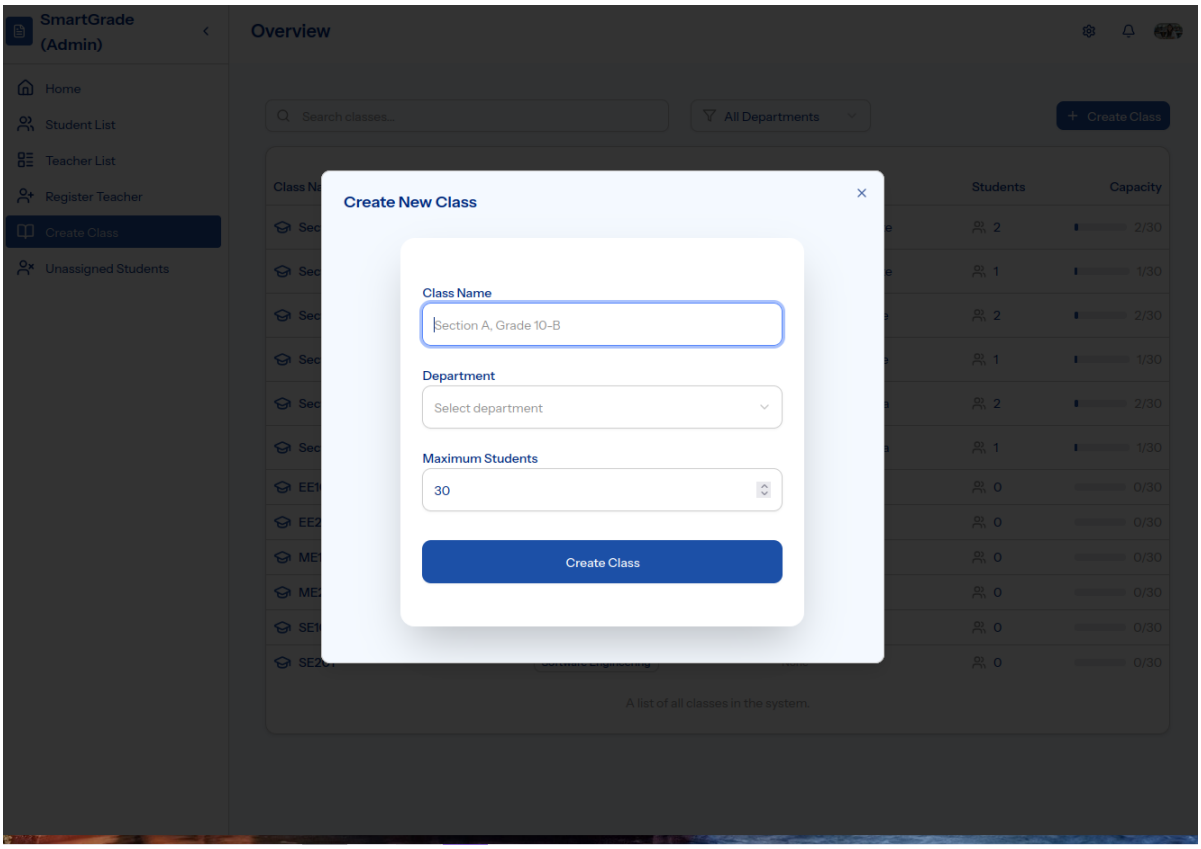

Figure 36. Admin create class page

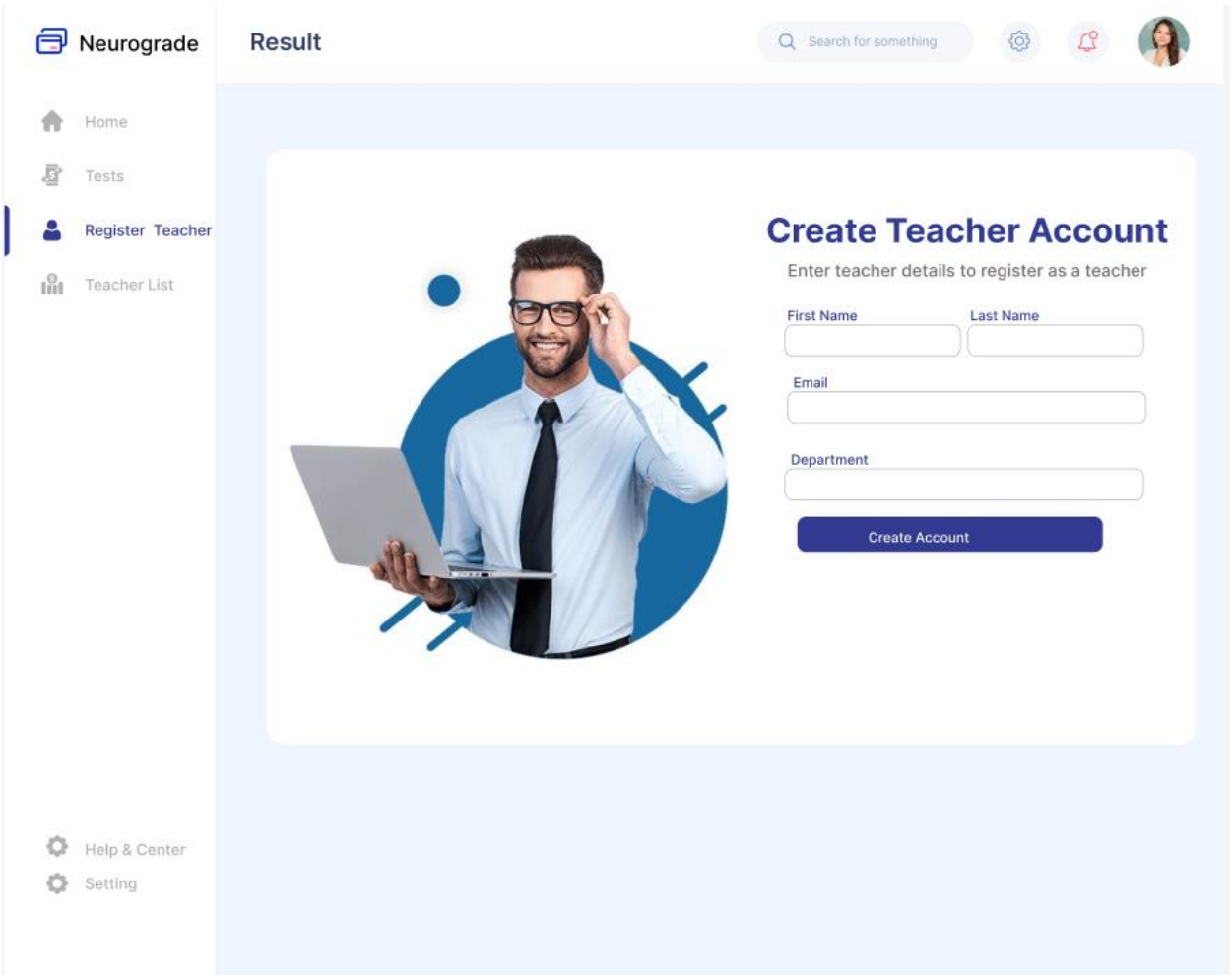

Figure 37. Admin class teacher account page

4.3.6. System Integration

While modules of the system are standalone and self-contained, they also have gateways to communicate with each other, with clear and intentionally designed API endpoints for communicating the necessary information. The structure and form of data that is transferred from the ML-Facing module to the User-Facing module, and vice versa, is specified and defined. The exposed API endpoints, then enable the Laravel backend to call the ml module with a specific test data, while the ml model accepts the data, processes it, passes it through the network of ml models, and then returns a final grading result as a response. The Laravel backend receives the response, stores it in the database, and displays it to users as needed.

The user facing module contains some third party packages/libraries

- **Spatie/Laravel-permission**: for laravel, to manage user roles and permissions
- **Tightenco**: for setting up routes endpoints easily for inertia.js
- **codemirro**r: for a code editor implementation with syntax highlighting, with support for multiple languages

The ML facing module, on the other hand, contains the following third party packages and libraries

1. **pandas**: Used for efficient loading, manipulation, and analysis of structured tabular data.
2. **requests**: Used to send HTTP requests for downloading data or interacting with web APIs.
3. **nltk**: Used for natural language processing tasks such as tokenization, stemming, and stopword removal.
4. **torch (PyTorch)**: Used as the core deep learning framework for defining, training, and running neural networks.
5. **torch_geometric**: Used to build and train graph neural networks, particularly for representing code as graphs.
6. **tree_sitter_language_pack**: Used to parse source code into abstract syntax trees for code analysis and feature extraction.
7. **scikit-learn (sklearn)**: Used for machine learning utilities like TF-IDF vectorization and train-test splitting.
8. **numpy**: Used for numerical operations and handling arrays efficiently, especially alongside machine learning workflows.

4.3.7. Security design

We have implemented comprehensive security measures to ensure the reliability and protection of our system, leveraging the capabilities of Laravel and additional tools. Below are the key aspects of our security design:

4.3.7.1. Authentication & Session Management

We ensure secure authentication through Laravel's built-in features, including request validation, CSRF protection, and password hashing. Passwords are hashed using bcrypt, Laravel's default algorithm, and we enforce strong password policy during registration and login. For role-specific authentication, admins use a dedicated, obfuscated login endpoint (`/admin/authorize`) to minimize exposure, while students and

teachers share a common login UI with role-based validation to prevent unauthorized access. Session security is maintained via Laravel's encrypted session driver, with a 1 day inactivity timeout.

4.3.7.2. Authorization & RBAC

We employ Role-Based Access Control (RBAC) using the Spaite package, defining three explicit roles: Student, Teacher, and Admin. Each role has granular permissions (e.g., `view_submissions`, `grade_assignments`, `manage_users`), and routes are protected using middleware (e.g., `role:admin`, `permission:delete_user`). Admin APIs and dashboards are isolated under `/admin/` routes, ensuring only authorized users can access them, while students and teachers are redirected to their respective dashboards post-login.

4.3.7.3. Input Validation & Sanitization

We enforce strict request validation using Laravel's validation rules (e.g., `required|string|max:255`, `mimes:java,cpp`) to block suspicious inputs and prevent SQL injection. Eloquent ORM is used for all database interactions. For file uploads, we restrict executable files (e.g., `.exe`, `.sh`) and verify MIME types. Additionally, a custom `SanitizesMarkdown` trait strips HTML tags while preserving LaTeX syntax and escapes special characters. Uploaded code files undergo static analysis to detect malicious patterns before processing.

4.3.7.4. Data Protection

Sensitive data, such as API keys and user emails, are encrypted at rest using Laravel's `encrypt()` helper. Database encryption is applied to tables storing code submissions and feedback. For file uploads, we store files in directories with restricted permissions.

4.3.7.5. CSRF & XSS Protection

We utilize Laravel's built-in CSRF protection for all forms and API endpoints to prevent cross-site request forgery. Cross-site scripting (XSS) attacks are mitigated through output escaping and sanitization,

particularly in user-generated content such as markdown and code submissions. By combining these measures, we ensure a robust defense against common web vulnerabilities.

4.4. Verifying the Requirements in the Design

Having laid out the comprehensive design of our system, we went back to our previous requirement specifications, and made sure they align with the specifications. Going back to our requirements and doing a high level review, we can see how the proposed designs fit well with the requirements that had been put forth, which include the requirements that had been put forth in section 3.3.2. Looking at each point in the specifications and verifying them against our design implementations, we can see that they meet the specifications in the following manner.

- **Performance**: The react+inertia based user interfaces enable fast and dynamic user experience, with inertial improving page loads and data caching.
- **Scalability:** The selection of laravel backend, the separation of concerns between the user-facing module and ml-facing module, enables to develop and scale the application efficiently, by enabling each component to be updated and scaled without affecting the whole system's performance.
- **Reliability**: Secure deployment on render platform,
- **Security:** usage of laravel's capabilities such as role and permission management, ORM, and hashing, as well as the usage of PostgresSQL database, ensures our system is reliable and secure.
- **Model Isolation:** We have separated the ml-facing module from the user module, ensuring that users do not have direct access to the ml models and their inputs/outputs.
- **Usability:** We have designed a clear, intuitive and engaging user interface, utilizing react.js's capabilities, and its built in adherence to WCAG 2.1 accessibility standards.
- **User-Friendly Tools:** The system has been enabled to provide simple, user-friendly tools for students and teachers to interact with feedback, without exposing complex system-level interactions. And this is ensured by the fact that system-level interactions are restricted via the Role Based Access Control (RBAC) and the separating of the ML-facing module from the user-facing module.
- **Maintainability:** The system has been designed to have a modular architecture, and its sub-components have also been designed with modularity and code reuse in mind. Each module and its sub-components can be separately maintained.

- **Extensibility:** The modular, and highly organized codebase of the system, as well as the design specifications that are built with that in mind, ensure that the system can be extended further as needed, at any point in time.

Chapter Five: System Implementation

5. 1. Reviewing the Design Solution

The Team had reviewed the proposed design solutions, analyzed them in terms of correctness and criticality, as well as how realistic they could be from the point of view of the tangible, on-the-ground development process. In this light, the system's general and specific objectives were revised, and the functional requirements were refined. And the fine, intricate details of the whole system's architecture were fleshed out. And as a natural progression to this, the data flow, activity, and sequence representations of the system were revisited, and revised, after that their diagrams were re-done to reflect the new changes in requirements and project objectives.

5. 2 Deciding on the development tools

The Automated Code Review E-Learning System requires a robust set of development tools to deliver its core functionalities, including secure user authentication, efficient code submission processing, and machine learning-driven feedback generation, as outlined in the use case diagram and class diagram. We have carefully selected programming languages, frameworks, libraries, databases, and version control systems to meet the system’s objectives. Each tool was chosen for a specific purpose in implementing the system’s 15 use cases, such as Submit for Evaluation and View Feedback, and supporting classes like User, Submission, and MLModel. The following paragraphs detail why we chose each tool, how it is used in the system, its importance to achieving project goals, and the setup process to enable efficient development. By leveraging modern practices like Laravel’s starter kits and Git for collaboration, we ensure a streamlined workflow that aligns with the system’s scalability, security, and usability requirements.

We have chosen Laravel, a PHP-based framework, for managing the system’s backend operations, including user authentication, assignment handling, and submission processing. Laravel powers the Login and Register use cases by using Sanctum to enforce role-based access for User class instances, ensuring only authorized students, teachers, or admins can access their dashboards. It handles Assignment uploads and Submission data storage, interacting with PostgreSQL via Eloquent ORM to persist data for Upload Assignments and Submit for Evaluation use cases. Laravel’s integration with Inertia.js enables dynamic updates for Review Feedback, allowing teachers to refine ML-generated feedback seamlessly. This tool is critical for its modular architecture, which supports maintainability, and its built-in security features, like

AES-256 encryption, aligning with our security design goal. To set up Laravel, we use Laravel 12's React starter kit, a GitHub repository with a pre-configured Laravel codebase, TypeScript-based React UI, Inertia.js, and Tailwind CSS. We download and extract this repository to a project directory, copy .env.example to .env, and add PostgreSQL credentials. The .env file is excluded from Git via .gitignore to protect sensitive data, a key security measure. We then run composer install to install Laravel and dependencies, php artisan key:generate to create a 32-character encryption key, php artisan migrate to set up tables, and npm install && npm run build to compile frontend assets. Running composer run dev starts php artisan serve and npm run dev, making the application accessible at localhost:8000 for development.

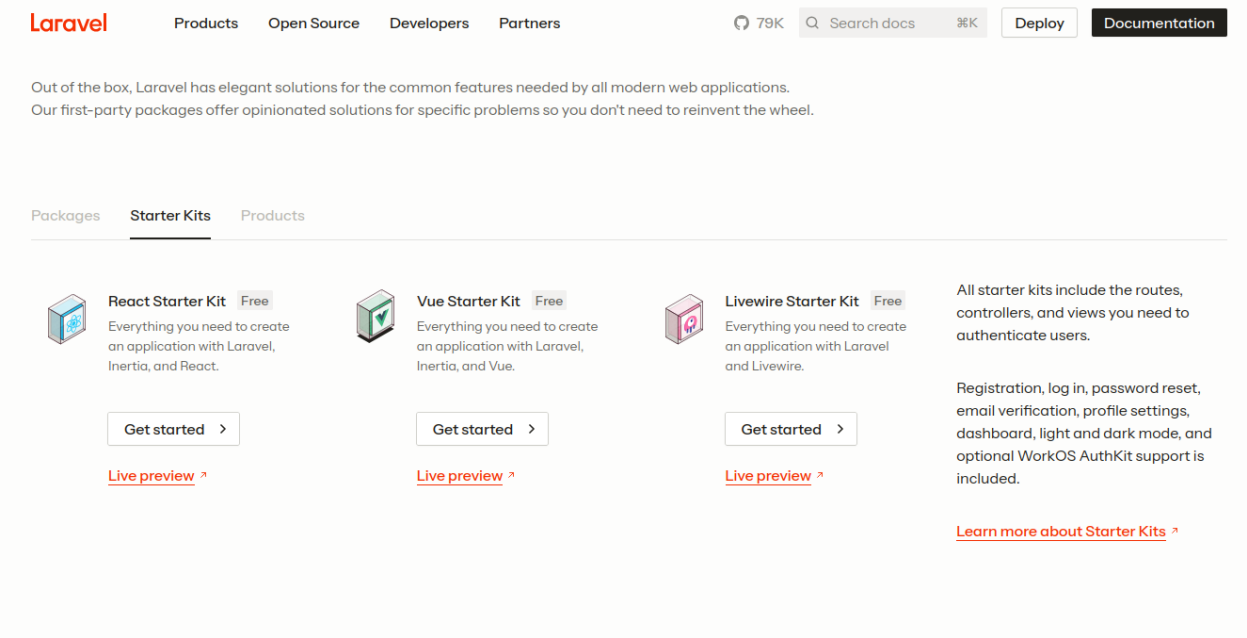

Figure 38. Laravel Starter Kit

We have chosen Eloquent ORM, Laravel's built-in object-relational mapper, for simplifying database interactions and managing data relationships. Eloquent maps PostgreSQL tables to classes like User, Assignment, and Submission, enabling object-oriented queries for use cases like Assign Grades and Download Report. It handles one-to-many relationships, such as between Assignment and Submission, streamlining data retrieval for View Feedback. Eloquent ORM's expressive syntax reduces SQL vulnerabilities, enhancing security, and its abstraction supports scalability by allowing table modifications without code changes. This tool is vital for maintaining readable, maintainable code, a core design goal. Since Eloquent is included in Laravel, its setup involves generating models and migrations with php artisan make:model Submission -m, defining relationships (e.g., hasMany for Assignment), and running php artisan migrate to apply table schemas, ensuring efficient data management for users.

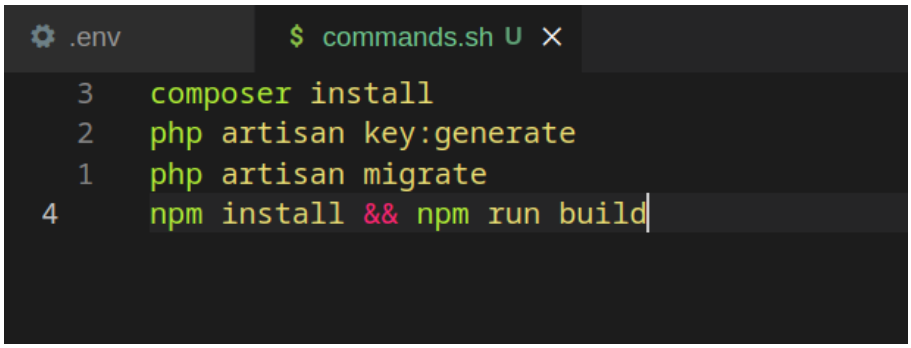

Figure 39. Code for installing dependencies and migrations

We have chosen Flask, a lightweight Python framework, for deploying machine learning models to support the Generate Feedback use case. Flask serves the MLModel class, processing Submission code files (.py, .java, .cpp) with GNNs and LLMs to analyze correctness and style. It exposes API endpoints that Laravel calls to retrieve feedback, ensuring model isolation as per our security goal. Flask's simplicity enables rapid API development, critical for delivering feedback within 5 seconds, aligning with the performance goal. This tool is essential for integrating ML capabilities into the system, a core feature distinguishing our automated feedback system. We install Flask in a Python virtual environment (pip install flask) and configure endpoints like analyze-code to handle code analysis requests, deploying Flask on a separate server to maintain scalability and security.

We have chosen React.js, a JavaScript library, for building dynamic, accessible frontend interfaces for students and teachers. React renders components for View Feedback and Upload Source Code, displaying Feedback class data in WCAG 2.1-compliant layouts and handling file uploads for Submission objects. Its reusable components, like feedback displays and submission forms, enhance usability, a key design goal, and support interactive features for Access Code Insights. React's importance lies in delivering intuitive, responsive UI critical for diverse users. Setup occurs via Laravel's starter kit, with React dependencies installed using npm install. We configure components in /resources/js using TypeScript's .tsx syntax, and npm run build compiles assets, while npm run dev enables live updates during development.

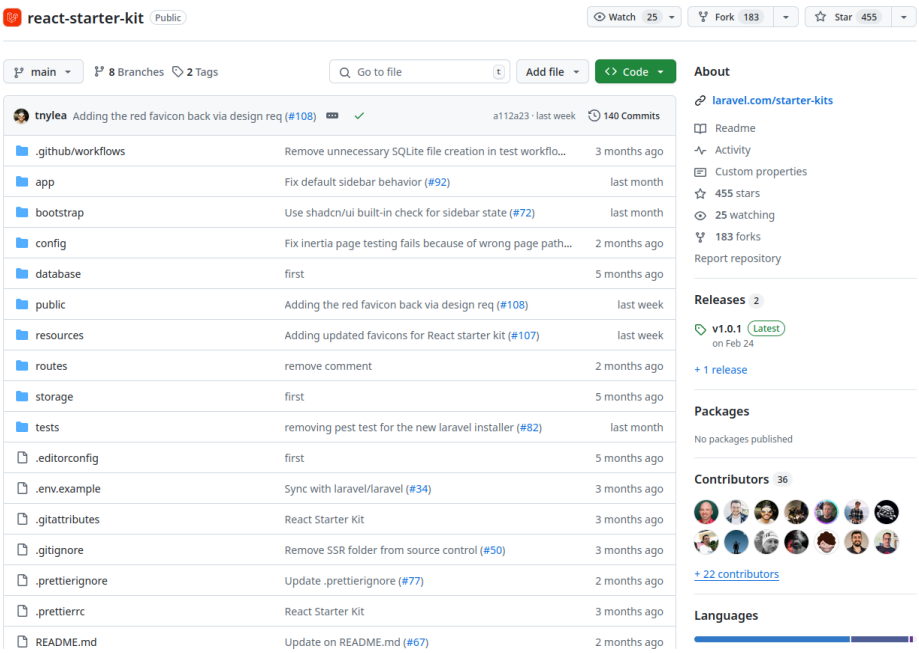

Figure 40. Github repo for react-starter kit

We have chosen TypeScript, a typed JavaScript superset, for ensuring robust frontend development. TypeScript adds type safety to React components, preventing errors in forms for Submit for Evaluation by validating Submission and Assignment data. Its editor integration catches issues early, improving maintainability and supporting our goal of delivering error-free UI. TypeScript is critical for ensuring reliable user interactions, especially for students submitting code. It's included in Laravel's starter kit, installed via npm install typescript --save-dev, with tsconfig.json set for JSX support and strict typing, enabling seamless integration with React.

We have chosen Inertia.js for connecting Laravel and React to provide single page application (SPA) like experiences. Inertia.js enables smooth page transitions for Review Feedback and Access Code Insights, eliminating full page reloads and supporting the performance goal of minimal latency. It simplifies frontend-backend communication by using Laravel controllers, crucial for teachers navigating feedback interfaces. Inertia.js is vital for delivering a cohesive user experience across use cases. We install it with npm install @inertiajs/react and composer require inertiajs/inertia-laravel, configure middleware in Laravel's kernel, and define React pages in /resources/js/Pages, ensuring dynamic, responsive interfaces.

We have chosen PostgreSQL, an open-source database, for storing system data with high integrity and scalability. PostgreSQL maintains tables for User (userID, role), Assignment (assignmentID, dueDate), Submission (submissionID, filePath), and Feedback (feedbackID, feedbackText), supporting use cases like

Assign Grades and Download Report. Its ACID compliance ensures reliable concurrent submissions for multiple users, and indexing optimizes queries for View Feedback, aligning with scalability and performance goals. PostgreSQL's JSON support stores complex code insights, making it indispensable for data management. We install it (sudo apt install postgresql), create a database (createdb code_review_db), update Laravel's .env (DB_CONNECTION=pgsql, DB_DATABASE=code_review_db), and run php artisan migrate to create indexed tables.

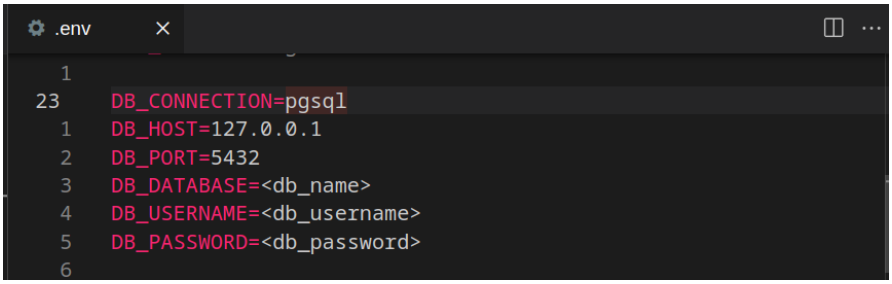

Figure 41. Code snippet for database connection

We have chosen PyTorch, a machine learning framework, for implementing GNNs and LLMs to analyze code submissions. PyTorch powers the MLModel class, processing Submission files into Abstract Syntax Trees for the Generate Feedback use case, evaluating correctness and style within 5 seconds. Its flexibility supports training models for diverse code formats (.py, .java, .cpp), critical for educational center programming courses. PyTorch's integration with Flask ensures scalable feedback delivery, making it a cornerstone of the system's automated review capability. We install PyTorch in Flask's virtual environment (pip install torch), train models on code datasets, and expose them via API endpoints, enabling seamless feedback generation.

We have chosen Git, a version control system, for managing collaborative development. Git tracks changes to Laravel, React, and Flask code, supporting team collaboration on classes like User and Submission via GitHub. It ensures code integrity and enables reviews, aligning with the maintainability goal. Git is critical for coordinating our development team. We install it (sudo apt install git), initialize a repository (git init), and configure .gitignore to exclude .env and node_modules. Remote repositories (git remote add origin <url>) facilitate sharing and backups.

We have chosen Visual Studio Code (VS Code) and Docker for a standardized development environment. VS Code, with extensions for PHP, TypeScript, and Python, supports coding and debugging for all tools, ensuring consistency across Laravel, React, and Flask development. Docker standardizes PostgreSQL and Flask environments via docker-compose.yml, enhancing portability. These tools are vital for streamlining

development of all use cases. We install VS Code, Composer (composer install), npm (npm install), and configure Docker, running php artisan serve for Laravel and flask run for Flask to enable efficient workflows.

5.3 Developing the Solution

5.3.1 Developing User facing module

Developing the user facing module was started from the setting up data models and migrations. Several models and their database migration files were generated, based on the database design. And then, suitable Laravel controller files, including auth controllers, admin controllers, student controllers and dashboard controllers were defined.

In the ui side, the react codebase was structured into different pages, components and layouts. Mainly student, teacher and admin specific layouts were presented, and then they were broken down to their subsequent pages and components, as well as their own dashboard layouts were generated. The specific .tsx file structures defined the ui pages, while the web routes defined the endpoints that the react pages required to communicate with the backend.

In our module we began by specifying our data models in php format, but also using Eloquent' ORM's specifications of relations such as HasOne, HasMany, BelongsTo, BelongsToMany.

```
// user model
class User extends Authenticatable {
    use HasApiTokens, HasFactory, Notifiable, HasRoles;

    protected $fillable = [
        'first_name',
        'last_name',
        'email',
        'password',
    ];

    protected $hidden = [
        'password',
        'remember_token',
    ];

    protected function casts(): array {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }

    protected function name(): Attribute {
        return Attribute::make(
            get: fn () => $this->first_name . ' ' . $this->last_name,
        );
    }

    public function student() {
        return $this->hasOne(Student::class);
    }

    public function teacher() {
        return $this->hasOne(Teacher::class);
    }

    public function admin() {
        return $this->hasOne(Admin::class);
    }
}

// student model
class Student extends Model {
    protected $fillable = [
        'user_id',
        'id_number',
        'department_id',
        'academic_year',
    ];

    public function user() {
        return $this->belongsTo(User::class, "user_id");
    }
    public function department() {
        return $this->belongsTo(Department::class);
    }

    public function classes() {
```

```
        return $this->belongsToMany(ClassRoom::class, "class_students", "student_id",
"class_id");
    }

    public function tests() {
        return $this->belongsToMany(Test::class, "test_student", "student_id", "test_id");
    }
}

// admin model
class Admin extends Model {
    protected $fillable = [
        'user_id',
    ];

    public function user() {
        return $this->belongsTo(User::class);
    }

    public function createdTeachers() {
        return $this->hasMany(Teacher::class, 'created_by');
    }

    public function createdClasses() {
        return $this->hasMany(ClassRoom::class, 'created_by');
    }
}

// teacher model
class Teacher extends Model {
   protected $fillable = [
      'user_id',
      'created_by',
      'department_id'
   ];

   public function user() {
      return $this->belongsTo(User::class, "user_id");
   }

   public function admin() {
      return $this->belongsTo(Admin::class, 'created_by');
   }

   public function department() {
      return $this->belongsTo(Department::class);
   }

   public function classes() {
      return $this->hasMany(ClassRoom::class, "teacher_id");
   }

   public function grades() {
      return $this->hasMany(Grade::class);
   }
   public function feedbacks() {
      return $this->hasMany(Feedback::class);
   }
}
```

```
// class model
class ClassRoom extends Model {
    use HasFactory;

    protected $table = 'classes';

    protected $fillable = [
        'name',
        'teacher_id',
        'admin_id',
        'max_students',
        'created_by'
    ];

    public function teacher() {
        return $this->belongsTo(Teacher::class);
    }

    public function admin() {
        return $this->belongsTo(Admin::class);
    }

    public function department() {
        return $this->belongsTo(Department::class);
    }

    public function students() {
        return $this-
>belongsToMany(Student::class,'class_students','class_id','student_id');
    }
}

class Test extends Model {
    use HasFactory;

    protected $fillable = [
        'title',
        'dueDate',
        'status',
    ];

    public function departments() {
        return $this->belongsToMany(Department::class, 'test_departments', 'test_id',
'department_id');
    }

    public function students() {
        return $this->belongsToMany(Student::class, 'test_student', 'test_id',
'student_id');
    }

    public function submissions() {
        return $this->hasMany(Submission::class);
    }
}

class Submission extends Model {
    use HasFactory;
```

```
    protected $fillable = [
        "test_id",
        "submission_id",
        "submission_type",
        "code_file_path",
        "code_editor_text",
        "submission_date",
        "statue"
    ];

    function grades() {
        return $this->hasMany(AiGradingResult::class, 'submissions_id');
    }

    function feedbacks() {
        return $this->hasMany(Feedback::class);
    }
    function test() {
        return $this->belongsTo(Test::class);
    }
}
```

Table 22. Backend Model Code

We then proceeded to create database seeding classes so that we could automatically fill selected database tables with default data, this solves the problem of having to fill the tables by hand all the time, especially when we need a change in the table. The tables are static enough to not need very frequent updates, and they are not expected to be updated programmatically as a result of user interaction with the system, therefore we fill them via database seeders, and run the database seeding commands when we deem necessary.

```
// Admin seeder, our requirements dictate that we have one single admin
// with preset credentials, and only staffs or personnel authorized by the
// system owners will have access to those credentials.
class AdminSeeder extends Seeder
{
    public function run()
    {
        $adminRole = Role::firstOrCreate(['name' => 'admin']);

        $admin = User::create([
            'first_name' => 'Super',
            'last_name' => 'Admin',
            'email' => 'admin@gmail.com',
            'password' => Hash::make('admin123'),
        ]);

        $admin->assignRole($adminRole);

        Admin::create([
            'user_id' => $admin->id,
        ]);
    }
}

// Department Seeder, we will have a pre-set number of departments
class DepartmentSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        $departments = [
            ['name' => 'Software Engineering'],
            ['name' => 'Computer Science'],
            ['name' => 'Electrical Engineering'],
            ['name' => 'Mechatronics Engineering']
        ];

        foreach ($departments as $department) {
            Department::create($department);
        }
    }
}
```

```
// role and permission seeder, it will actually create the roles and permissions that
// we will have in our system. It's based on spatie's specifications
class RolesAndPermissionsSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        //
          // Create roles
        $adminRole = Role::create(['name' => 'admin']);
        $teacherRole = Role::create(['name' => 'teacher']);
        $studentRole = Role::create(['name' => 'student']);

        // Create permissions
        $createTeacher = Permission::create(['name' => 'create teacher']);
        $createClass = Permission::create(['name' => 'create class']);
        $assignStudents = Permission::create(['name' => 'assign students']);
        $createTest = Permission::create(['name' => 'create test']);
        $reviewSubmission = Permission::create(['name' => 'review submission']);

        // Assign permissions to roles
        $adminRole->givePermissionTo([$createTeacher, $createClass, $assignStudents]);
        $teacherRole->givePermissionTo([$createTest, $reviewSubmission]);

    }
}

// top database seeder class, it will be called when the database seed command is run,
// and it will call the other database seeder classes' methods.
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    public function run(): void
    {
        // User::factory(10)->create();
        $this->call(RolesAndPermissionsSeeder::class);
        $this->call(DepartmentSeeder::class);
        $this->call(AdminSeeder::class);
    }
}
```

Table 23. Database Seeder Code Table

Once this has been set up, we call the following php commands to migrate the database tables, and seed the selected tables with data.

```
$ php artisan migrate
$ php artisan db:seed
```

Once this is done, we move on to the controllers, specifying the actions and features they have to perform.

```
// student authentication controller
class StudentAuthController extends Controller
{
    /**
     * Show the registration page.
     */
    public function create(): Response
    {
        $departments = Department::all(['id', 'name']);
        return Inertia::render('auth/register',[
            'departments' => $departments
        ]);
    }

    /**
     * Handle an incoming registration request.
     *
     * @throws \Illuminate\Validation\ValidationException
     */
    public function store(Request $request): RedirectResponse
    {
        $data = $request->validate([
            'first_name' => 'required|string|max:255',
            'last_name' => 'required|string|max:255',
            'email' => 'required|string|lowercase|email|max:255|unique:'.User::class,
            'password' => ['required', 'confirmed', Rules\Password::defaults()],
            'id_number' => 'required|string|max:255',
            'department' => 'required|integer|exists:departments,id',
            'academic_year' => 'required|string|max:255',
            ]);

        $passwordHashed = Hash::make($data["password"]);

        $user = User::create([
            'first_name' => $data['first_name'],
            'last_name' => $data['last_name'],
            'email' => $data['email'],
            'password' => $passwordHashed,
        ]);

        $user->assignRole('student');

        $student = new Student([
            'user_id' => $user->id,
            'id_number' => $data['id_number'],
            'academic_year' => $data['academic_year'],
            'department_id' => $data['department']
        ]);

        $user->student()->save($student);
        Auth::login($user);

        return to_route('dashboard');
    }

    public function loginCreate(Request $request): Response
    {
        return Inertia::render('auth/login', [
            'canResetPassword' => Route::has('password.request'),
```

```
            'status' => $request->session()->get('status'),
        ]);
    }

    public function loginStore(Request $request): Response
    {
        $request->authenticate();

        $request->session()->regenerate();

        return redirect()->intended(route('dashboard', absolute: false));
    }
}
```

Table 24. Student Authentication Controller Code

Another controller was also added to manage student related controller tasks;

```
class StudentController extends Controller
{
    //
    public function getResults()
    {
        $user = auth()->user()->load('student');
        $recentResults = AiGradingResult::whereHas('submission', function($query) use
($user) {
            $query->whereHas('test', function($testQuery) use ($user) {
                $testQuery->whereHas('students', function($studentQuery) use ($user) {
                    $studentQuery->where('students.id', $user->student->id);
                });
            });
        })
        ->with(['submission.test'])
        ->orderBy('created_at', 'desc')
        ->take(5)
        ->get();
        $formattedResults = $recentResults->map(function($result) {
            return [
                'id' => $result->id,
                'test' => [
                    'id' => $result->submission->test->id,
                    'title' => $result->submission->test->title,
                ],
                'score' => $result->graded_value,
                'comment' => $result->comment,
                'metrics' => json_decode($result->metrics, true),
                'submission_date' => $result->submission->submission_date,
                'status' => $result->submission->status,
            ];
        })->toArray();

        return Inertia::render('dashboard/studentDashbord/Results', [
            'results' => $formattedResults
        ]);
    }
```

```
    public function getTests()
    {
        $user = auth()->user()->load('student.department');


        $tests = Test::all();

        return Inertia::render('dashboard/studentDashbord/Tests/Index', [
            'tests' => $tests,
        ]);
    }

    public function showTest($id) {
        $user = auth()->user()->load('student');
        $test = Test::findOrFail($id);


        return Inertia::render('dashboard/studentDashbord/Tests/Show', [
            'id' => $id,
            'test' => [
                'id' => $test->id,
                'title' => $test->title,
                'problemStatement' => $test->problem_statement,
                'dueDate' => $test->due_date,
                'status' => $test->status,
            ]
        ]);
    }
}
```

Table 25. Student CRUD controller code

Also, a controller for admin actions:

```
<?php

namespace App\Http\Controllers;

use App\Models\ClassRoom;
use App\Models\Department;
use App\Models\Role;
use App\Models\Student;
use App\Models\Teacher;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Redirect;
use Illuminate\Validation\Rules;
use Inertia\Inertia;

class AdminController extends Controller
{
        public function login(Request $request)
        {
        $request->validate([
                'email' => 'required|email',
```

```
                'password' => 'required',
        ]);

        $user = User::where('email', $request->email)->first();

        if (!$user || !Hash::check($request->password, $user->password)) {
                return Redirect::back()->withErrors(['email' => 'Invalid credentials']);
        }

        if (!$user->hasRole('admin')) {
                return response()->view('errors.403', ['message' => 'Unauthorized access'],
403);
        }

        $request->session()->regenerate();
        auth()->login($user);

        return Redirect::intended(route('admin.dashboard'));
        }

        public function showCreateTeacherForm()
        {
        $departments = Department::all();
        return Inertia::render('dashboard/adminDashboard/RegisterTeacher', ['departments' =>
$departments]);
        }

        public function createTeacher(Request $request)
        {
        $request->validate([
                'first_name' => 'required|string|max:255',
                'last_name' => 'required|string|max:255',
                'email' => 'required|string|lowercase|email|max:255|unique:'.User::class,
                'password' => ['required', 'confirmed', Rules\Password::defaults()],
                'department_id' => 'required|exists:departments,id',
        ]);

        $user = User::create([
                'first_name' => $request->first_name,
                'last_name' => $request->last_name,
                'email' => $request->email,
                'password' => Hash::make($request->password),
        ]);

        $role = Role::where('name', 'teacher')->where('guard_name', 'web')->first();
        $user->assignRole($role);

        $teacher = Teacher::create([
                'user_id' => $user->id,
                'created_by' => auth()->id(),
                'department_id' => $request->department_id,
        ]);

        $user->teacher()->save($teacher);

        return Redirect::route('admin.teachers.index')->with('success', 'Teacher created
successfully.');
        }
```

```
        public function showTeachers()
        {
        $teachers = Teacher::with(['user', 'department'])->get();
        return Inertia::render('dashboard/adminDashboard/TeacherListPage', ['teachers' =>
$teachers]);
        }

        public function showCreateClassForm()
        {
        $departments = Department::all();
        $teachers = Teacher::with('user')->get();
        return Inertia::render('dashboard/adminDashboard/CreateClassPage', ['departments' =>
$departments, 'teachers' => $teachers]);
        }

        public function createClass(Request $request)
        {
        $request->validate([
                'name' => 'required|string|max:255',
                'department_id' => 'required|exists:departments,id',
                'teacher_id' => 'required|exists:teachers,id',
                'max_students' => 'required|integer|min:1',
        ]);

        $teacher = Teacher::findOrFail($request->teacher_id);
        if ($teacher->department_id != $request->department_id) {
                return Redirect::back()->withErrors(['teacher_id' => 'Teacher does not belong
to the specified department.']);
        }

        ClassRoom::create([
                'name' => $request->name,
                'department_id' => $request->department_id,
                'teacher_id' => $request->teacher_id,
                'admin_id' => auth()->id(),
                'max_students' => $request->max_students,
                'created_by' => auth()->id(),
        ]);

        return Redirect::route('admin.classes.index')->with('success', 'Class created
successfully.');
        }

        public function showClasses()
        {
        $classes = ClassRoom::with(['teacher.user', 'department'])->get();
        return Inertia::render('Admin/Classes/Index', ['classes' => $classes]); // Assuming
you might create an Admin/Classes directory
        }

        public function showStudents()
        {
        $students = Student::with(['user', 'department'])->get();
        return Inertia::render('dashboard/adminDashboard/StudentListPage', ['students' =>
$students]);
        }

        public function showAssignStudentsForm(ClassRoom $class)
        {
```

```
        $unassignedStudents = Student::whereDoesntHave('classes', function ($query) use
($class) {
                $query->where('classroom_id', $class->id);
        })->where('department_id', $class->department_id)
        ->with('user')
        ->get();

        $assignedStudents = $class->students()->with('user')->get();

        return Inertia::render('Admin/Classes/AssignStudents', [ // Assuming you might create
an Admin/Classes directory
                'class' => $class->load('department'),
                'unassignedStudents' => $unassignedStudents,
                'assignedStudents' => $assignedStudents,
        ]);
        }

        public function assignStudentsToClass(Request $request, ClassRoom $class)
        {
        $request->validate([
                'student_ids' => 'required|array',
                'student_ids.*' => 'exists:students,id',
        ]);

        $invalidStudents = Student::whereIn('id', $request->student_ids)
                ->where('department_id', '!=', $class->department_id)
                ->exists();

        if ($invalidStudents) {
                return Redirect::back()->withErrors(['student_ids' => 'Some students do not
belong to the same department as the class.']);
        }

        $class->students()->sync($request->student_ids);

        return Redirect::route('admin.classes.assign', $class->id)->with('success', 'Students
assigned successfully.');
        }

        public function showUnassignedStudents()
        {
        $unassignedStudents = Student::whereDoesntHave('classes')
                ->with('user', 'department')
                ->get();

        return Inertia::render('Admin/Students/Unassigned', ['unassignedStudents' =>
$unassignedStudents]); // Assuming you might create an Admin/Students directory
        }
}
```

Table 26. Admin controller code

Once we have set up our models, database migrations and controllers this way, we then proceed to specify web routes to set up the endpoints that the frontend will access. They will be the files of auth.php, web.php, and other route files that will be treated as extensions of web.php file.

We have auth.php like so:

```
Route::middleware('guest')->group(function () {
    Route::get('student-register', [StudentAuthController::class, 'create'])
        ->name('student-register');

    Route::post('student-register', [StudentAuthController::class, 'store'])->name('student-
register.store');

    Route::get('login', [StudentAuthController::class, 'create'])
        ->name('login');

    Route::post('login', [StudentAuthController::class, 'store'])->name('studnet-
login.store');

    Route::get('forgot-password', [PasswordResetLinkController::class, 'create'])
        ->name('password.request');

    Route::post('forgot-password', [PasswordResetLinkController::class, 'store'])
        ->name('password.email');

    Route::get('reset-password/{token}', [NewPasswordController::class, 'create'])
        ->name('password.reset');

    Route::post('reset-password', [NewPasswordController::class, 'store'])
        ->name('password.store');
});

Route::middleware('auth')->group(function () {
    Route::get('verify-email', EmailVerificationPromptController::class)
        ->name('verification.notice');

    Route::get('verify-email/{id}/{hash}', VerifyEmailController::class)
        ->middleware(['signed', 'throttle:6,1'])
        ->name('verification.verify');

    Route::post('email/verification-notification',
[EmailVerificationNotificationController::class, 'store'])
        ->middleware('throttle:6,1')
        ->name('verification.send');

    Route::get('confirm-password', [ConfirmablePasswordController::class, 'show'])
        ->name('password.confirm');

    Route::post('confirm-password', [ConfirmablePasswordController::class, 'store']);

    Route::post('logout', [AuthenticatedSessionController::class, 'destroy'])
        ->name('logout');
});
```

Table 27. auth.php web routes code

We also have the main routes file, web.php:

```
use Illuminate\Support\Facades\Route;
```

```
use Inertia\Inertia;
use App\Models\Test;
use App\Models\AiGradingResult;

Route::get('/', function () {
    //return Inertia::render('welcome');
    return Inertia::render('dashboard/adminDashboard/RegisterTeacher');

})->name('home');

Route::get('admin-login', function(){
    return Inertia::render('auth/admin_login');
});

Route::middleware(['auth'])->group(function () {
    Route::get('dashboard', function () {
        //return Inertia::render('dashboard');
        if (auth()->user()->hasRole('admin')){
            return Inertia::render('auth/admin_login');
        } elseif (auth()->user()->hasRole("teacher")){
            return Inertia::render('dashboard');
        } elseif (auth()->user()->hasRole('student')){
            $user = auth()->user()->load('student.department');

            $upcomingTests = Test::all();

            $recentResults = AiGradingResult::all();

            // Transform the results for the frontend
            $formattedResults = $recentResults->map(function($result) {
                return [
                    'id' => $result->id,
                    'test' => [
                        'id' => $result->submission->test->id,
                        'title' => (string)$result->submission->test->title,
                    ],
                    'score' => (float)$result->graded_value,
                    'comment' => (string)$result->comment,
                    // Convert metrics to a simple array if it's not already
                    'metrics' => is_string($result->metrics) ? json_decode($result->metrics,
true) : (array)$result->metrics,
                    'submission_date' => $result->submission->submission_date instanceof
\DateTime
                        ? $result->submission->submission_date->format('Y-m-d')
                        : (string)$result->submission->submission_date,
                    'status' => (string)$result->submission->status,
                ];
            });

            // Return the student dashboard page with necessary data
            return Inertia::render('dashboard/studentDashbord/Home', [
                    'user' => [
                        'id' => $user->id,
                        'name' => $user->first_name . ' ' . $user->last_name,
                        'email' => $user->email,
                        'student' => [
                            'id_number' => $user->student->id_number,
                            'academic_year' => $user->student->academic_year,
                            'department' => $user->student->department->name ?? 'Unknown'
```

```
                        ]
                    ],
                    'upcomingTests' => $upcomingTests->toArray(), // Ensure this is an array
                    'recentResults' => $formattedResults->toArray(), // Ensure this is a
simple array
            ]);
        }
    })->name('dashboard');
});

// Student dashboard routes

require __DIR__.'/web-students.php';
require __DIR__.'/web-admin.php';
require __DIR__.'/settings.php';
require __DIR__.'/auth.php';
```

Table 28. main web.php routes code

The 'require __DIR__' directive suggests that we have other files within the specified directory path that will be considered along with this web.php file. Those files include auth.php, web-students and web-admin.php.

web-student.php has this content:

```
Route::middleware(['auth', 'role:student'])->prefix('student')->group(function () {
    Route::get('/results', [StudentController::class, 'getResults'])-
>name('student.results');
    Route::get('/tests', [StudentController::class, 'getTests'])->name('student.tests');
    Route::get('/tests/{id}', [StudentController::class, 'showTest'])-
>name('student.test.show');
    Route::post('/tests{id}/submitCodeFile', [StudentController::class, 'submitFile']);
    Route::post('/tests{id}/submitCodeText', [StudentController::class, 'submitCode']);
});
```

Table 29. web-student routes code

As we can see in the web routes, Laravel provides the feature to intelligently group the routes and wrap them with the middleware that makes sure the user has a role as a student. This reliably ensures the principle of Role Based Access Control (RBAC) that we sought to implement.
And we have our react front-end pages this way

```
// top level AppLayout
import AppLayoutTemplate from '@/layouts/app/app-sidebar-layout';
import { type BreadcrumbItem } from '@/types';
import { type ReactNode } from 'react';

interface AppLayoutProps {
    children: ReactNode;
    breadcrumbs?: BreadcrumbItem[];
}
```

```
export default ({ children, breadcrumbs, ...props }: AppLayoutProps) => (
    <AppLayoutTemplate breadcrumbs={breadcrumbs} {...props}>
        {children}
    </AppLayoutTemplate>
);

// student dashboard layout
import type { ReactNode } from "react"
import { DashboardHeader } from "@/components/dashboard/studentDashboard/DashboardHeader"
import { Sidebar } from "@/components/dashboard/studentDashboard/StudentSidebar"

export const AppLayout = ({ children, title }: { children: ReactNode; title?: string })=> {
  return (
    <div className="flex min-h-screen">
      <Sidebar />
      <div className="dashboard-content flex-1 flex flex-col w-full max-w-full overflow-x-
hidden">
        <DashboardHeader title={title} />
        <main className="flex-1 p-6 overflow-auto w-full max-w-full">{children}</main>
      </div>
    </div>
  )
}

// teacher dashboard layout
import type { ReactNode } from "react"
import { SidebarProvider } from "@/components/ui/sidebar"
import { DashboardHeader } from "@/components/dashboard/studentDashboard/DashboardHeader"
import { Sidebar } from "@/components/dashboard/studentDashboard/StudentSidebar";

export function AppLayout({ children, title }: { children: ReactNode; title?: string }) {
  return (
    <div className="flex min-h-screen">
      <Sidebar />
      <div className="dashboard-content flex-1 flex flex-col w-full max-w-full overflow-x-
hidden">
        <DashboardHeader title={title} />
        <main className="flex-1 p-6 overflow-auto w-full max-w-full">{children}</main>
      </div>
    </div>
  )
}

// admin dashboard layout
import type { ReactNode } from "react"
import { DashboardHeader } from "@/components/dashboard/studentDashboard/DashboardHeader"
import { Sidebar } from "@/components/dashboard/adminDashboard/AdminSidebar";

export function AppLayout({ children, title }: { children: ReactNode; title?: string }) {
  return (
    <div className="flex min-h-screen">
      <Sidebar/>
      <div className="dashboard-content flex-1 flex flex-col w-full max-w-full overflow-x-
hidden">
        <DashboardHeader title={title} />
        <main className="flex-1 p-6 overflow-auto w-full max-w-full">{children}</main>
      </div>
    </div>
```

```
  )
}

// student dashboard page
interface HomeProps {
  user: {
    id: number;
    name: string;
    email: string;
    student: {
      id_number: string;
      academic_year: string;
      department: string;
    }
  };
  upcomingTests: Array<{
    id: number;
    title: string;
    date: string;
    // other test properties
  }>;
  recentResults: Array<{
    id: number;
    test: {
      title: string;
    };
    score: number;
    // other result properties
  }>;
}

export default function Home({ user, upcomingTests, recentResults }: HomeProps) {
  console.log('Home props:', { user, upcomingTests, recentResults });
  return (
    <AppLayout title="Overview">
      <div className="space-y-6">
          <WelcomeBanner userName={user.name} />

        <div className="grid gap-6 md:grid-cols-2">
          <Card>
            <CardHeader>
              <CardTitle>Quick Stats</CardTitle>
            </CardHeader>
            <CardContent>
              <QuickStats
                student={user.student}
                resultCount={recentResults.length}
              />
            </CardContent>
          </Card>

          <Card>
            <CardHeader>
              <CardTitle>Upcoming Test</CardTitle>
            </CardHeader>
            <CardContent>
              <UpcomingTest tests={upcomingTests} />
            </CardContent>
          </Card>
```

```
        </div>

        <div className="grid gap-6 md:grid-cols-2">
          <Card>
            <CardHeader>
              <CardTitle>Quick Links</CardTitle>
            </CardHeader>
            <CardContent>
              <QuickLinks />
            </CardContent>
          </Card>

          <Card>
            <CardHeader>
              <CardTitle>Recent Result</CardTitle>
            </CardHeader>
            <CardContent>
              <RecentResult results={recentResults} />
            </CardContent>
          </Card>
        </div>
      </div>
    </AppLayout>
  )
}
```

Table 30. React frontend pages

5.3.2 Training the Model and developing Ml-facing module

5.3.2.1 Collecting Data

To collect the data for our project, we utilized the Project CodeNet dataset, a large-scale dataset for code understanding and analysis tasks. Our data collection process involved two primary phases: downloading the dataset and filtering relevant metadata. We wrote a Python script to download the entire Project CodeNet archive from its official source. To ensure robustness and support for resuming interrupted downloads, we used HTTP range headers. The tqdm library was used to display a progress bar during the download. This allowed us to monitor the download status in real time and handle large file sizes efficiently.

5.3.2.2 Extracting and Filtering the Data

After downloading and extracting the dataset, we focused on selecting only relevant subsets (e.g., C++, Python, Java submissions). We recursively traversed the submission directories to collect the IDs of all problems for which we had code samples. This ensured that our analysis was limited to problems with available benchmark data.

We then loaded the global metadata file (problem_list.csv) and filtered it to retain only the entries corresponding to the collected problem IDs. We also removed unnecessary columns such as rating, tags, and complexity to focus only on the essential metadata fields.

```
import os
import pandas as pd

def collect_problem_ids(root_dirs):
   """
   Traverse each directory in root_dirs and collect folder names
   matching 'p' followed by digits (e.g., p00001).
   """
   ids = set()
   for root in root_dirs:
       for dirpath, dirnames, _ in os.walk(root):
           for dirname in dirnames:
               if dirname.startswith("p") and dirname[1:].isdigit():
                   ids.add(dirname)
   return sorted(ids)

def main():
   # 1. Define benchmark folder paths
   roots = ["../cpp_subs", "../python_subs", "../java_subs"]

   # 2. Collect problem IDs
   problem_ids = collect_problem_ids(roots)
   print(f"Found {len(problem_ids)} problems across benchmarks.")

   # 3. Load global metadata
   metadata_df = pd.read_csv("../Project_CodeNet/metadata/problem_list.csv")

   # 4. Filter metadata to only selected problems
   filtered_df = metadata_df[metadata_df["id"].isin(problem_ids)]
   del filtered_df['rating']
   del filtered_df['tags']
   del filtered_df['complexity']
   print(f"Filtered metadata contains {len(filtered_df)} entries.")

   # 5. Export to new CSV
```

```
    output_path = "filtered_problems_metadata.csv"
    filtered_df.to_csv(output_path, index=False)
    print(f"Exported filtered metadata to {output_path}")

if __name__ == "__main__":
    main()
```

Table 31. Python Code to extract and filter the relevant problem submissions

The final filtered metadata was saved as filtered_problems_metadata.csv, which served as the basis for our subsequent analysis and processing tasks.

5.3.2.3 Data Preprocessing

Language Filtering & Text Preprocessing: We filtered non-English problems using both language detection and a regex check for Japanese characters. For the English problems, we applied standard NLP preprocessing: lowercasing, punctuation removal, tokenization, stopword removal, and stemming. The cleaned statements, inputs, and outputs were saved into problems_preprocessed.csv.

```
import csv
import re
import string
import nltk
from langdetect import detect, LangDetectException
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

# Download required NLTK resources
nltk.download("punkt")
nltk.download("stopwords")

stop_words = set(stopwords.words("english"))
stemmer = PorterStemmer()

# Regex matching any Japanese character (Hiragana, Katakana, Kanji)
jp_regex = re.compile(r"[\u3040-\u309F\u30A0-\u30FF\u4E00-\u9FFF]", re.UNICODE)
```

```
def preprocess_text(text):
   text = text.lower()
   text = re.sub(r"\$\$\$|\s+", " ", text)
   text = text.translate(str.maketrans("", "", string.punctuation))
   tokens = word_tokenize(text)
   return " ".join(
       stemmer.stem(tok) for tok in tokens if tok.isalpha() and tok not in stop_words
   )

def is_english(text):
   """
   Returns True if 'text' is likely English via:
   1) langdetect → 'en'
   2) no Japanese characters via regex
   """
   # Quick regex check to skip obvious Japanese
   if jp_regex.search(text):
       return False

   # Fallback to langdetect
   try:
       return detect(text) == "en"
   except LangDetectException:
       # If detection fails (too short/etc.), assume non-English
       return False

def preprocess_csv(input_csv, output_csv):
   processed = []
   with open(input_csv, newline="", encoding="utf-8") as rf:
       reader = csv.DictReader(rf)
       for row in reader:
           # Only keep rows whose *original* statement is English
           if not is_english(row.get("statement", "")):
               continue

           # Preprocess all three text fields
           for field in ("statement", "input_spec", "output_spec"):
               row[field] = preprocess_text(row.get(field, ""))
```

```
            processed.append(
                {
                    "problem_id": row.get("problem_id", ""),
                    "statement": row["statement"],
                    "input_spec": row["input_spec"],
                    "output_spec": row["output_spec"],
                }
            )

    # Write filtered & preprocessed data
    with open(output_csv, "w", newline="", encoding="utf-8") as wf:
        writer = csv.DictWriter(
            wf,
            fieldnames=["problem_id", "statement", "input_spec", "output_spec"],
            quoting=csv.QUOTE_ALL,
        )
        writer.writeheader()
        writer.writerows(processed)

if __name__ == "__main__":
    preprocess_csv(
        input_csv="problems_statements.csv", output_csv="problems_preprocessed.csv"
    )
    print(" Saved filtered & preprocessed data to 'problems_preprocessed.csv'")
```

Table 32. Python code preprocessing the problem statements

Collecting Cleaned Submissions per Language: We then wrote a script to collect and clean accepted programming submissions in C++, Python, and Java. First, we loaded a CSV file (final_problem_statements.csv) containing the finalized problem IDs. These IDs served as filters to ensure we only included relevant and validated problems. For each language, we organized submissions into separate directories (cpp_subs, python_subs, and java_subs), where each subdirectory was named after a specific problem ID. The script then iterated through these folders to find submission files with recognized extensions for each language (.cpp, .py, .java).

```
import csv
import re
import pandas as pd
```

```
from pathlib import Path


def collapse_whitespace(s: str) -> str:
   """Replace any run of whitespace (spaces, tabs, newlines) with a single space."""
   return re.sub(r"\s+", " ", s).strip()


def remove_comments(code: str, lang: str) -> str:
   """
   Strip out comments from code according to language.
   - Python: '#' to end-of-line
   - C++/Java: '//' to end-of-line, and '/* ... */' block comments
   """
   if lang == "Python":
       # Remove '#' comments but leave '#' in strings untouched in the simple case
       return re.sub(r"#.*", "", code)
   else:
       # Remove C-style line comments
       code_no_line = re.sub(r"//.*", "", code)
       # Remove C-style block comments (non-greedy, across lines)
       return re.sub(r"/\*.*?\*/", "", code_no_line, flags=re.DOTALL)


def collect_submissions(final_ids_csv: str, lang_dirs: dict, output_dir: str = "."):
   # Load the list of final problem IDs
   final_ids = set(pd.read_csv(final_ids_csv)["problem_id"].astype(str))
   print(f"🔍 Loaded {len(final_ids)} final problem IDs.")

   # File extensions by language
   exts_map = {
       "C++": {".cpp", ".cc", ".cxx", ".C"},
       "Python": {".py"},
       "Java": {".java"},
   }

   for lang, folder in lang_dirs.items():
       records = []
       lang_path = Path(folder)
```

```
    if not lang_path.exists():
        print(f" Directory for {lang} not found at {folder}, skipping.")
        continue

    print(f"Processing {lang} submissions in {folder}...")

    # Each subfolder named by problem_id
    for prob_dir in lang_path.iterdir():
        pid = prob_dir.name
        if pid not in final_ids or not prob_dir.is_dir():
            continue

        for file in prob_dir.iterdir():
            if file.is_file() and file.suffix in exts_map.get(lang, ()):
                try:
                    raw = file.read_text(encoding="utf-8", errors="ignore")
                except Exception as e:
                    print(f" Could not read {file}: {e}")
                    continue

                # 1) strip comments, 2) collapse whitespace
                stripped = remove_comments(raw, lang)
                code_clean = collapse_whitespace(stripped)

                records.append(
                    {
                        "problem_id": pid,
                        "submission_file": file.name,
                        "code": code_clean,
                    }
                )

    # Write out CSV
    out_name = f"submissions_{lang.replace('+','p')}.csv"
    out_path = Path(output_dir) / out_name
    df = pd.DataFrame.from_records(
        records, columns=["problem_id", "submission_file", "code"]
    )
    df.to_csv(out_path, index=False, quoting=csv.QUOTE_ALL)
```

```
        print(f" Wrote {len(df)} {lang} submissions to {out_path}")


if __name__ == "__main__":
    lang_dirs = {
        "C++": "../cpp_subs",
        "Python": "../python_subs",
        "Java": "../java_subs",
    }

    collect_submissions(
        final_ids_csv="final_problem_statements.csv",
        lang_dirs=lang_dirs,
        output_dir=".",  # or "outputs/"
    )
```

Table 33. Python Code for normalizing and cleaning the code submissions for each language

5.3.2.4. Training the Model

We built and used several modules to train the SubmissionPredictor model on code submissions. First, we used the load_statements function to read and preprocess the problem statements, which were then embedded using the TextEmbedder module. The code submissions were embedded with the CodeEmbedderGNN. These embeddings were combined using the ConcatEmbedder to create a unified feature representation for each submission. The DatasetLoader module was used to load and batch the data, providing a DataLoader for each language (C++, Python, Java). Finally, we trained the SubmissionPredictor model using these embeddings, optimizing predictions for the verdict, runtime, and memory of each submission. This process efficiently combined our custom modules to prepare the data and train the model.

```
import os
import torch
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split

from preprocessor import Preprocessor
```

```
from text_embedder import TextEmbedder
from code_embedder import CodeEmbedderGNN
from concat_embedder import ConcatEmbedder
from submission_predictor import SubmissionPredictor
from dataset_loader import DatasetLoader  # our refactored loader


def load_statements(path: str) -> dict:
   """
   Reads the problem statements CSV into a dict mapping problem_id -> statement.
   """
   import pandas as pd

   df = pd.read_csv(path, dtype={"problem_id": str})
   return {row.problem_id: row.statement for _, row in df.iterrows()}

def train_language(
   lang: str,
   loader: DatasetLoader,
   device: torch.device,
   epochs: int = 5,
   lr: float = 1e-3,
):
   # build dataloader using our loader
   dl = loader.get_dataloader(lang)
   # instantiate model based on concat dimension
   model = SubmissionPredictor(loader.concat_emb.final_dim).to(device)
   optimizer = optim.Adam(model.parameters(), lr=lr)
   for epoch in range(1, epochs + 1):
       model.train()
       total_loss = 0.0
       for features, (v_t, r_t, m_t) in dl:
           # push to device
           features = features.to(device)
           v_t = v_t.to(device)
           r_t = r_t.to(device).float()
           m_t = m_t.to(device).float()
           outputs = model(features)
           # compute combined loss
```

```
            loss = (
                F.cross_entropy(outputs["verdict"], v_t)
                + F.mse_loss(outputs["runtime"], r_t)
                + F.mse_loss(outputs["memory"], m_t)
            )

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        avg_loss = total_loss / len(dl)
        print(f"[{lang}] Epoch {epoch:02d}  avg_loss={avg_loss:.4f}")
    return model

if __name__ == "__main__":
    # set device
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    # 1) Load and preprocess statements
    statements = load_statements("data/final_problem_statements.csv")
    text_emb = TextEmbedder()
    cleaned_texts = [Preprocessor.preprocess_text(s) for s in statements.values()]
    text_emb.fit(cleaned_texts)
    # 2) Build code embedder and concat wrapper
    code_emb = CodeEmbedderGNN()  # init with required args if any
    # example: infer text embed dimension
    sample_dim = text_emb.transform([cleaned_texts[0]]).shape[1]
    concat = ConcatEmbedder(code_emb, text_emb, sample_dim)
    # 3) Create a DatasetLoader for all languages
    loader = DatasetLoader(
        data_dir="data",
        concat_embedder=concat,
        batch_size=16,
        shuffle=True,
        num_workers=4,
    )
    # 4) Train for each language
    for lang in ["Cpp", "Python", "Java"]:
        print(f"\n=== Training on {lang} submissions ===")
        _ = train_language(
```

```
            lang=lang,
            loader=loader,
            device=device,
            epochs=3,
            lr=1e-3,
        )
```

Table 34. Python Code for Training the model

Chapter Six: System Evaluation

6.1. Test Plan

For Our Project we will structure our testing based on the two modules; the user-facing module and the ML facing module.Then Test Cases will be specific for each one. The Test Cases to be specified will follow formal test case specification rules and notations. These notations will include Test Case ID, Feature/Module to be covered, Test Objective, Preconditions, Test Steps and Expected Results, among other things.

6.2. Evaluating the Proposed Design and Solutions

6.2.1 Test Cases for user-facing module

Test Case Notation Used

- **Test Case ID:** Unique identifier (e.g., GEN_001, ADM_001)
- **Feature/Module:** The specific part of the application being tested.
- **Test Objective:** What the test aims to verify.
- **Roles:** User role relevant to this test case (Admin, Teacher, Student, Unauthenticated, All).
- **Preconditions:** Conditions that must be met before executing the test.
- **Test Steps:** Numbered sequence of actions to perform.
- **Expected Result:** The anticipated outcome if the test passes.
- **Priority:**Using High, Medium and Low which Indicates the importance of the test case.
- **Type:** Functional, UI, Security, Usability, Performance, Accessibility which specifies the Type of test.

6.2.1.1. General Features

Test Case ID: GEN_001

- **Feature/Module:** User Scope
- **Test Objective:** Verify that user-specific data is scoped correctly.
- **Role:** Admin, Teacher, Student
- **Preconditions:**
    - Multiple users of different roles exist in the system.
    - Each user has some associated data (students have grades, teachers have created tests).

- **Test Steps:**
    - Log in as User A.
    - Navigate to their dashboard and other relevant sections.
    - Observe the data displayed.
    - Log out.
    - Log in as User B.
    - Navigate to their dashboard and other relevant sections.
    - Observe the data displayed.
- **Expected Result:**
    - User A should only see data relevant to User A.
    - User B should only see data relevant to User B.
    - There should be no data leakage between users of the same or different roles unless explicitly designed.
- **Priority:** High
- **Type:** Functional, Security

Test Case ID: GEN_002

- **Feature/Module:** Due Dates for Assessments
- **Test Objective:** Verify that due dates are correctly displayed for assessments and that submission behavior changes accordingly..
- **Roles:** Student, Teacher
- **Preconditions:**
    - A teacher has created an assessment with a specific due date.
    - Students are assigned to this assessment.
- **Test Steps:**
    - As a Teacher, create a test with a due date.
    - As a Student, log in and view the upcoming tests.
    - Verify the due date is displayed correctly.
    - As a Student, attempt to submit the test before the due date.
    - As a Student, attempt to view/submit the test after the due date.
    - As a Teacher, view student submissions and check for any indicators related to submission time ("late").
- **Expected Result:**
    - Students can see the correct due date for assessments.
    - Students can submit before the due date.
    - System handles submissions after the due date as per requirements it marks as late, prevents submission.
    - Teachers can see if submissions were on time or late.
- **Priority:** High
- **Type:** Functional

Test Case ID: GEN_003

- **Feature/Module:** Student Grade Visibility
- **Test Objective:** Verify that students can only see their grades after a teacher has graded the assessment and made them visible.
- **Roles:** Student, Teacher

- **Preconditions:**
    - A student has submitted an assessment.
- **Test Steps:**
    - As a Student, submit an assessment.
    - As a Student, navigate to the results/grades page.
    - Verify that the grade for the recently submitted assessment is not yet visible or shows a "pending" status.
    - As a Teacher, log in and navigate to the grading section for that assessment.
    - Grade the student's submission.
    - If applicable Perform an action to "publish" or "release" grades.
    - As a Student, log back in and navigate to the results/grades page.
- **Expected Result:**
    - Initially, the student cannot see the grade for the un-graded assessment.
    - After the teacher grades and releases the grade, the student can see their grade.
- **Priority:** High
- **Type:** Functional

Test Case ID: GEN_004

- **Feature/Module:** Role-Based Access Control (RBAC) - General Navigation
- **Test Objective:** Verify that users can only access dashboards and features appropriate to their role.
- **Role(s):** Admin, Teacher, Student, Unauthenticated
- **Preconditions:**
    - User accounts for Admin, Teacher, and Student roles exist.
- **Test Steps:**
    - Attempt to access /dashboard/adminDashboard (or similar admin URL) without logging in.
    - Attempt to access /dashboard/teacherDashboard without logging in.
    - Attempt to access /dashboard/studentDashboard without logging in.
    - Log in as a Student. Attempt to access /dashboard/adminDashboard and /dashboard/teacherDashboard.
    - Log in as a Teacher. Attempt to access /dashboard/adminDashboard and /dashboard/studentDashboard.
    - Log in as an Admin. Attempt to access /dashboard/teacherDashboard and /dashboard/studentDashboard.
- **Expected Result:**
    - Unauthenticated users are redirected to the login page when trying to access any dashboard.
    - Students can only access the student dashboard and their specific pages (profile, tests, results). They are denied access or redirected from admin/teacher pages.
    - Teachers can only access the teacher dashboard and their specific pages. They are denied access or redirected from admin pages (student dashboard access might be different if they need to view student profiles within their classes).
    - Admins can access the admin dashboard. Access to other dashboards might be allowed for oversight, or restricted to their primary role functions.
- **Priority:** High
- **Type:** Security, Functional

Test Case ID: GEN_005

- **Feature/Module:** Login-based Redirection
- **Test Objective:** Verify users are redirected to their role-specific dashboards after successful login.
- **Role(s):** Admin, Teacher, Student
- **Preconditions:**
    1. Valid credentials for Admin, Teacher, and Student roles exist.
    2. The single login page (pages/auth/login.tsx) is functional.
- **Test Steps:**
    1. Navigate to the login page.
    2. Enter valid Student credentials and submit.
    3. Verify redirection to the Student dashboard (pages/dashboard/studentDashboard/Home.tsx or similar).
    4. Log out.
    5. Navigate to the login page.
    6. Enter valid Teacher credentials and submit.
    7. Verify redirection to the Teacher dashboard
    8. Log out.
    9. Navigate to the login page.
    10. Enter valid Admin credentials (using pages/auth/admin_login.tsx if it's a separate flow, or the main login if it handles all roles based on credentials).
    11. Verify redirection to the Admin dashboard (pages/dashboard/adminDashboard/ TeacherListPage.tsx or a dedicated admin home).
- **Expected Result:** Each user role is redirected to their designated dashboard page upon successful login.
- **Priority:** High
- **Type:** Functional, Usability

6.2.1.2. Login & Registration

Test Case ID: LOG_001

- **Feature/Module:** Single Login Page - Student Login
- **Test Objective:** Verify successful student login via the single login page.
- **Role(s):** Unauthenticated, Student
- **Preconditions:**
    1. A valid student account exists.
    2. The login page is resources/js/pages/auth/login.tsx.
- **Test Steps:**
    1. Navigate to the login page.
    2. Enter valid student username/email.
    3. Enter valid student password.
    4. Click the "Login" button.
- **Expected Result:** User is redirected to the student dashboard (resources/js/pages/dashboard/studentDashbord/Home.tsx). No error messages are displayed.
- **Priority:** High

- **Type:** Functional

Test Case ID: LOG_002

- **Feature/Module:** Single Login Page - Teacher Login
- **Test Objective:** Verify successful teacher login via the single login page.
- **Role(s):** Unauthenticated, Teacher
- **Preconditions:**
    1. A valid teacher account exists (created by an admin or through secure teacher sign-up).
    2. The login page is resources/js/pages/auth/login.tsx.
- **Test Steps:**
    1. Navigate to the login page.
    2. Enter valid teacher username/email.
    3. Enter valid teacher password.
    4. Click the "Login" button.
- **Expected Result:** User is redirected to the teacher dashboard (resources/js/pages/dashboard/teacherDashboard/...). No error messages are displayed.
- **Priority:** High
- **Type:** Functional

Test Case ID: LOG_003

- **Feature/Module:** Admin Login
- **Test Objective:** Verify successful admin login via the dedicated admin login page/flow.
- **Role(s):** Unauthenticated, Admin
- **Preconditions:**
    1. A default admin account exists.
    2. Admin login page is resources/js/pages/auth/admin_login.tsx.
- **Test Steps:**
    1. Navigate to the admin login page (/admin/login or equivalent route).
    2. Enter valid admin username/email.
    3. Enter valid admin password.
    4. Click the "Login" button.
- **Expected Result:** User is redirected to the admin dashboard (resources/js/pages/dashboard/adminDashboard/...). No error messages are displayed.
- **Priority:** High
- **Type:** Functional

Test Case ID: LOG_004

- **Feature/Module:** Login - Invalid Credentials
- **Test Objective:** Verify that login fails with appropriate error message for invalid credentials.
- **Role(s):** Unauthenticated
- **Preconditions:** The login page (resources/js/pages/auth/login.tsx) is accessible.
- **Test Steps:**
    1. Navigate to the login page.
    2. Enter an invalid username/email.
    3. Enter an invalid password.
    4. Click the "Login" button.

- **Expected Result:** Login fails. An appropriate error message (e.g., "Invalid credentials," "User not found") is displayed on the login page. User is not redirected.
- **Priority:** High
- **Type:** Functional, UI

Test Case ID: LOG_005

- **Feature/Module:** Secure Teacher Sign-up/Sign-in (Hidden from Students)
- **Test Objective:** Verify that the teacher registration page/mechanism is not directly accessible or obvious to students.
- **Role(s):** Unauthenticated, Student
- **Preconditions:**
    - The teacher registration page is resources/js/pages/auth/teacher_register.tsx.
    - Student registration page is resources/js/pages/auth/register.tsx.
- **Test Steps:**
    - As an unauthenticated user, inspect the main login page (login.tsx) and student registration page (register.tsx).
    - Verify there are no direct links to teacher registration (teacher_register.tsx).
    - Attempt to directly navigate to the URL for teacher registration (e.g., /teacher/register).
    - Log in as a Student.
    - Inspect the student dashboard and available links.
    - Verify there are no links to teacher registration.
- **Expected Result:**
    - The teacher registration process should be initiated by an Admin or through a secure, non-public link/method.
    - Students should not be able to easily discover or access the teacher registration page. Direct navigation might be blocked or require a special token/admin approval.
- **Priority:** High
- **Type:** Security, Functional

Test Case ID: LOG_006

- **Feature/Module:** Forgot Password
- **Test Objective:** Verify the "Forgot Password" functionality.
- **Role(s):** Unauthenticated, Student, Teacher
- **Preconditions:**
    - A user (student or teacher) with a registered email exists.
    - Email services are configured for the application.
    - The forgot password page is resources/js/pages/auth/forgot-password.tsx.
- **Test Steps:**
    - Navigate to the login page (login.tsx).
    - Click the "Forgot Password?" link.
    - Verify redirection to the forgot-password.tsx page.
    - Enter the registered email address of an existing user.
    - Click the "Send Password Reset Link" button.
    - Check the user's email inbox for a password reset email.
    - Click the reset link in the email.
    - Verify redirection to the reset-password.tsx page, possibly with a token in the URL.

    - Enter a new password.
    - Confirm the new password.
    - Click the "Reset Password" button.
    - Attempt to log in with the new password.
- **Expected Result:**
    - User receives a password reset email.
    - User can successfully reset their password using the link.
    - User can log in with the new password.
    - The old password no longer works.
- **Priority:** High
- **Type:** Functional

6.2.1.3. Admin Features

Test Case ID: ADM_001

- **Feature/Module:** Admin Dashboard Access
- **Test Objective:** Verify that only users with the 'Admin' role can access the Admin Dashboard.
- **Role(s):** Admin, Teacher, Student, Unauthenticated
- **Preconditions:**
    - Admin, Teacher, and Student accounts exist.
    - Admin dashboard layout is resources/js/layouts/dashboard/adminDashboard/AdminDashboardLayout.tsx.
- **Test Steps:**
    - Attempt to access an admin dashboard URL (e.g., /admin/teachers) as an Unauthenticated user.
    - Log in as a Student and attempt to access an admin dashboard URL.
    - Log in as a Teacher and attempt to access an admin dashboard URL.
    - Log in as an Admin and navigate to an admin dashboard URL.
- **Expected Result:**
    - Unauthenticated users are redirected to login.
    - Students and Teachers are shown an unauthorized error or redirected to their respective dashboards.
    - Admins can successfully access the admin dashboard pages.
- **Priority:** High
- **Type:** Security, Functional

Test Case ID: ADM_002

- **Feature/Module:** Teacher Registration Management - View Teachers
- **Test Objective:** Verify admin can view a list of registered teachers.
- **Role(s):** Admin
- **Preconditions:**
    1. Admin is logged in.
    2. Several teacher accounts exist.
    3. The teacher list page is resources/js/pages/dashboard/adminDashboard/TeacherListPage.tsx.

4. The component resources/js/components/dashboard/adminDashboard/TeacherList.tsx is used.

- **Test Steps:**
    1. Navigate to the Admin Dashboard.
    2. Click on the "Manage Teachers" or "View Teachers" link/section.
    3. Verify redirection to TeacherListPage.tsx.
- **Expected Result:** A list/table of registered teachers is displayed, showing relevant information (e.g., Name, Email, Status, Date Registered).
- **Priority:** High
- **Type:** Functional, UI

Test Case ID: ADM_003

- **Feature/Module:** Teacher Registration Management - Register New Teacher
- **Test Objective:** Verify admin can successfully register a new teacher.
- **Role(s):** Admin
- **Preconditions:**
    - Admin is logged in.
    - The teacher registration page for admin is resources/js/pages/dashboard/adminDashboard/RegisterTeacher.tsx.
    - The component resources/js/components/dashboard/adminDashboard/TeacherRegister.tsx is used.
- **Test Steps:**
    - Navigate to the Admin Dashboard.
    - Click on the "Register Teacher" or "Add New Teacher" link/button.
    - Verify redirection to RegisterTeacher.tsx.
    - Fill in the required teacher details (e.g., Name, Email, temporary Password or send invite).
    - Submit the form.
    - Check the teacher list (TeacherListPage.tsx) or system logs for the new teacher.
- **Expected Result:**
    - The new teacher account is created successfully.
    - A success message is displayed.
    - The new teacher appears in the list of teachers.
    - (If applicable) The teacher receives a notification email.
- **Priority:** High
- **Type:** Functional

Test Case ID: ADM_004

- **Feature/Module:** Teacher Registration Management - Input Validations
- **Test Objective:** Verify input validations on the teacher registration form.
- **Role(s):** Admin
- **Preconditions:**
    1. Admin is logged in and on the RegisterTeacher.tsx page.
- **Test Steps:**
    1. Attempt to submit the form with empty required fields (e.g., Name, Email).
    2. Attempt to submit the form with an invalid email format.
    3. Attempt to submit the form with an email that already exists for another user.

4. (If applicable) Attempt to submit with passwords that don't meet criteria or don't match.

- **Expected Result:** Appropriate error messages are displayed for each validation failure. The form is not submitted.
- **Priority:** Medium
- **Type:** Functional, UI

Test Case ID: ADM_005

- **Feature/Module:** Class Creation and Teacher Assignment - Create Class
- **Test Objective:** Verify admin can create a new class and assign a teacher to it.
- **Role(s):** Admin
- **Preconditions:**
  - Admin is logged in.
  - At least one teacher exists in the system.
  - The class creation page is resources/js/pages/dashboard/adminDashboard/CreateClassPage.tsx.
  - The component resources/js/components/dashboard/adminDashboard/CreateClass.tsx is used.
- **Test Steps:**
  - Navigate to the Admin Dashboard.
  - Click on "Manage Classes" or "Create Class".
  - Verify redirection to CreateClassPage.tsx.
  - Enter class details (e.g., Class Name/Code, Description).
  - Select an existing teacher from a list/dropdown to assign to the class.
  - Submit the form.
- **Expected Result:**
  - The new class is created successfully.
  - A success message is displayed.
  - The assigned teacher is associated with the class.
  - The class appears in a list of classes (if such a view exists).
- **Priority:** High
- **Type:** Functional

Test Case ID: ADM_006

- **Feature/Module:** Bulk Student Assignment to Classes - View Unassigned Students
- **Test Objective:** Verify admin can view a list of students not yet assigned to any class.
- **Role(s):** Admin
- **Preconditions:**
  1. Admin is logged in.
  2. There are students in the system, some of whom are not assigned to any class.
  3. The student list page is resources/js/pages/dashboard/adminDashboard/StudentListPage.tsx.
  4. The component resources/js/components/dashboard/adminDashboard/StudentList.tsx might be used with filtering.
- **Test Steps:**
  1. Navigate to the Admin Dashboard.
  2. Go to the "Manage Students" or "Assign Students to Class" section.

3. Look for an option or filter to "View Unassigned Students".
4. Activate the filter/view.

- **Expected Result:** A list of students who are not currently assigned to any class is displayed.
- **Priority:** Medium
- **Type:** Functional

6.2.1.4. Teacher Features

Test Case ID: TCH_001

- **Feature/Module:** Teacher Dashboard Access
- **Test Objective:** Verify that only users with the 'Teacher' role can access the Teacher Dashboard.
- **Role(s):** Teacher, Admin, Student, Unauthenticated
- **Preconditions:**
  - Admin, Teacher, and Student accounts exist.
  - Teacher dashboard layout is resources/js/layouts/dashboard/teacherDashboard/teacherDashoboardLayout.tsx.
- **Test Steps:**
  - Attempt to access a teacher dashboard URL (e.g., /teacher/exams) as an Unauthenticated user.
  - Log in as a Student and attempt to access a teacher dashboard URL.
  - Log in as an Admin and attempt to access a teacher dashboard URL (behavior might vary - some admins might have teacher capabilities).
  - Log in as a Teacher and navigate to a teacher dashboard URL.
- **Expected Result:**
  - Unauthenticated users are redirected to login.
  - Students are shown an unauthorized error or redirected.
  - Admins might be able to access it or be restricted based on design.
  - Teachers can successfully access their dashboard pages like CreateExam.tsx, GradingPage.tsx.
- **Priority:** High
- **Type:** Security, Functional

Test Case ID: TCH_002

- **Feature/Module:** Password Editing
- **Test Objective:** Verify teacher can successfully edit their own password.
- **Role(s):** Teacher
- **Preconditions:**
  - Teacher is logged in.
  - The password editing page is resources/js/pages/settings/password.tsx (shared settings page).
- **Test Steps:**
  - Navigate to the settings/profile section.
  - Select the "Change Password" option.
  - Enter the current password.
  - Enter a new password.

    - Confirm the new password.
    - Submit the form.
    - Log out.
    - Attempt to log in with the new password.
    - Attempt to log in with the old password.
- **Expected Result:**
    - Password is changed successfully.
    - A success message is displayed.
    - Teacher can log in with the new password.
    - Teacher cannot log in with the old password.
- **Priority:** High
- **Type:** Functional, Security

Test Case ID: TCH_003

- **Feature/Module:** Test Creation - Form Access and UI
- **Test Objective:** Verify teacher can access the test creation form and UI elements are present.
- **Role(s):** Teacher
- **Preconditions:**
    1. Teacher is logged in.
    2. The test creation page is resources/js/pages/dashboard/teacherDashboard/CreateExam.tsx.
    3. The component resources/js/components/dashboard/teacherDashboard/ExamForm.tsx is used.
- **Test Steps:**
    1. Navigate to the Teacher Dashboard.
    2. Click on "Create Test/Exam" or similar link.
    3. Verify redirection to CreateExam.tsx.
- **Expected Result:** The test creation form (ExamForm.tsx) is displayed with fields for Test Title, Description, Due Date, assigning to class/students, adding problems (problem statement, code input options, file upload options).
- **Priority:** High
- **Type:** Functional, UI

Test Case ID: TCH_004

- **Feature/Module:** Test Creation - Successful Creation
- **Test Objective:** Verify teacher can successfully create a new test with multiple problems.
- **Role(s):** Teacher
- **Preconditions:**
    - Teacher is logged in on the CreateExam.tsx page.
    - Teacher has classes to assign the test to.
- **Test Steps:**
    - Fill in the test title and description.
    - Set a due date.
    - Assign the test to one or more classes.
    - Add at least two problems:
        - Problem 1: with a problem statement.
        - Problem 2: with a problem statement.

    - Set a global grading criteria (if applicable at this stage, or ensure it's understood to be used later).
    - Save/Create the test.
- **Expected Result:**
    - Test is created successfully.
    - A success message is displayed.
    - The test appears in the teacher's list of created tests or on their dashboard.
    - Students in the assigned class(es) can see the test (covered in student tests).
- **Priority:** High
- **Type:** Functional

Test Case ID: TCH_005

- **Feature/Module:** Test Management - View Created Tests
- **Test Objective:** Verify teacher can view a list of tests they have created.
- **Role(s):** Teacher
- **Preconditions:**
    1. Teacher is logged in.
    2. Teacher has created multiple tests.
- **Test Steps:**
    1. Navigate to the Teacher Dashboard.
    2. Go to the "My Tests," "Manage Exams," or similar section.
- **Expected Result:** A list of created tests is displayed, showing relevant details (e.g., Title, Due Date, Number of Submissions, Status).
- **Priority:** High
- **Type:** Functional

Test Case ID: TCH_006

- **Feature/Module:** Submission Review - Access
- **Test Objective:** Verify teacher can access the submission review page for a specific test and problem.
- **Role(s):** Teacher
- **Preconditions:**
    1. Teacher is logged in.
    2. A test has been created and students have made submissions.
    3. The submission review page is resources/js/pages/dashboard/teacherDashboard/GradingPage.tsx or SubmittedExam.tsx.
    4. The component resources/js/components/dashboard/teacherDashboard/Grading.tsx or SubmittedExams.tsx is used.
- **Test Steps:**
    1. Navigate to the Teacher Dashboard.
    2. Select a test that has submissions.
    3. Select a specific student's submission or a specific problem to grade.
- **Expected Result:** The teacher is taken to a grading interface where they can see the student's code/file, the original problem statement, and an area to input/edit a grade.
- **Priority:** High
- **Type:** Functional, UI

6.2.1.5. Student Features

Test Case ID: STU_001

- **Feature/Module:** Student Dashboard Access & UI
- **Test Objective:** Verify student can access their dashboard and key UI elements are present.
- **Role(s):** Student
- **Preconditions:**
    - Student is logged in.
    - Student dashboard layout is resources/js/layouts/dashboard/studentDashboard/studentDashboardLayout.tsx.
    - Student home page is resources/js/pages/dashboard/studentDashbord/Home.tsx.
- **Test Steps:**
    - Log in as a Student.
- **Expected Result:**
    - Student is redirected to Home.tsx.
    - The dashboard displays elements like WelcomeBanner.tsx, QuickStats.tsx, UpcomingTest.tsx (component: studentDashboard/UpcomingTest.tsx), RecentOverview.tsx / RecentResult.tsx (components: studentDashboard/RecentOverview.tsx, studentDashboard/RecentResult.tsx).
    - Navigation links via StudentSidebar.tsx are present (e.g., to Tests, Results, Profile).
- **Priority:** High
- **Type:** Functional, UI

Test Case ID: STU_002

- **Feature/Module:** Profile Management - View Profile
- **Test Objective:** Verify students can view their profile information.
- **Role(s):** Student
- **Preconditions:**
    1. Student is logged in.
    2. The profile page is resources/js/pages/settings/profile.tsx (shared settings page).
- **Test Steps:**
    1. Navigate to the profile/settings page.
- **Expected Result:** Student's profile information (e.g., Name, User ID, Email, Class) is displayed. Some fields might be read-only.
- **Priority:** Medium
- **Type:** Functional

Test Case ID: STU_003

- **Feature/Module:** Profile Management - Edit Profile (If Allowed)
- **Test Objective:** Verify students can edit allowed fields in their profile.
- **Role(s):** Student
- **Preconditions:**
    1. Student is logged in on the profile.tsx page.
    2. Certain fields are designated as editable for students.
- **Test Steps:**
    1. Identify editable fields (e.g., display name, contact number - if applicable).

2. Modify an editable field.
3. Save the changes.
4. Re-load the profile page or log out and log back in.

- **Expected Result:** The changes made to editable profile fields are saved and persist.
- **Priority:** Medium
- **Type:** Functional

Test Case ID: STU_004

- **Feature/Module:** Combined Test View and Submission Page - Access
- **Test Objective:** Verify student can access an active test and see the problem statement and submission area.
- **Role(s):** Student
- **Preconditions:**
  - Student is logged in.
  - A teacher has created and assigned an active (not past due) test to the student's class.
  - Test list page is resources/js/pages/dashboard/studentDashbord/Tests/Index.tsx.
  - Individual test page is resources/js/pages/dashboard/studentDashbord/Tests/Show.tsx.
  - Code editor component is resources/js/components/dashboard/studentDashboard/CodeEditor.tsx.
- **Test Steps:**
  - Navigate to the "Tests" or "Assessments" section (Tests/Index.tsx).
  - Select an active test from the list (TestList.tsx component might be used here).
  - Verify redirection to the specific test page (Tests/Show.tsx).
- **Expected Result:**
  - The test page displays the problem statement(s).
  - For each problem, an area for code input (using CodeEditor.tsx) or file upload is available.
  - The due date is visible.
- **Priority:** High
- **Type:** Functional, UI

Test Case ID: STU_005

- **Feature/Module:** Test Submission - Code Input
- **Test Objective:** Verify student can submit a test by typing code into the code editor.
- **Role(s):** Student
- **Preconditions:**
  - Student is on an active test page (Tests/Show.tsx).
  - The test includes at least one problem requiring code input.
- **Test Steps:**
  - For a problem requiring code input, type or paste code into the CodeEditor.tsx.
  - Click the "Submit" or "Save Answer" button for that problem or the entire test.
- **Expected Result:**
  - The submission is accepted.
  - A confirmation message is displayed.
  - The student might be prevented from re-submitting if that's the rule, or the submission is updated.
  - The teacher can see this submission.

- **Priority:** High
- **Type:** Functional

Test Case ID: STU_006

- **Feature/Module:** Test Submission - File Upload
- **Test Objective:** Verify student can submit a test by uploading a file.
- **Role(s):** Student
- **Preconditions:**
    - Student is on an active test page (Tests/Show.tsx).
    - The test includes at least one problem allowing file upload.
- **Test Steps:**
    - For a problem allowing file upload, click the "Upload File" button.
    - Select a valid file from their local system.
    - Confirm the upload.
    - Click the "Submit" or "Save Answer" button.
- **Expected Result:**
    - The file is successfully uploaded and associated with the submission.
    - A confirmation message is displayed.
    - The teacher can access this uploaded file.
- **Priority:** High
- **Type:** Functional

6.2.2. ML Model Evaluation

The evaluation strategy is designed to systematically assess the model's ability to predict submission outcomes across different programming languages. It follows a predefined plan to ensure consistency and reproducibility:

6.2.2.1. Configuration and Preparation

The evaluation begins by selecting the hardware device (GPU if available, otherwise CPU) and defining key parameters such as batch size and the locations of the evaluation datasets. Ensuring these settings are explicitly defined upfront avoids discrepancies between runs.

The trained model artifacts are loaded from persistent storage, and the evaluation datasets—consisting of problem descriptions, code submissions, and related statistics—are assembled. Data is organized into batches for efficient processing, and checks are performed to confirm that all inputs are valid and non-empty.

6.2.2.2. Inference Process

Batches of problem statements and corresponding code submissions are fed through the model to obtain predicted labels. During this step, any samples flagged as invalid (for example, submissions without a clear ground-truth label) are excluded from subsequent metric calculations. Progress logs are emitted at regular intervals to monitor throughput.

6.2.2.3. Metric Computation

After gathering all true and predicted labels, a suite of evaluation metrics is calculated:

- **Accuracy**: The proportion of correct predictions over all evaluated samples.
- **Precision**: The ratio of true positive predictions to all positive predictions, computed per class. It reflects the model's ability to avoid false positives.
- **Recall**: The ratio of true positive predictions to all actual positives, computed per class. It measures the model's capacity to identify all relevant instances.
- **F1 Score**: The harmonic mean of precision and recall for each class, providing a balance between the two.
- **Support**: The number of true instances for each class, indicating class distribution in the evaluation set.
- **Confusion Matrix**: A table that visualizes correct and incorrect predictions for each class pair, highlighting patterns of misclassification.

6.2.2.4. Visualization and Reporting

When visualization tools are available, the confusion matrix is rendered as a heatmap to facilitate quick interpretation of error patterns. A formatted summary report presents all metrics in a clear and concise form, aiding stakeholders in understanding model performance at a glance.

6.3. Discussing the Results

In this section, each executed test is recorded using only four fields: Test Case ID, Status, Expected, and Actual, so that anyone reading the documentation can immediately see which case ran, whether it passed or failed (or was blocked or skipped), and how the observed behavior compared to the expectation. By limiting ourselves to these four items, we provide complete traceability back to our master test-case list (via the ID), while keeping the report small and outcome-focused.

The format is as follows:

- **Test Case ID**: the unique identifier from your test-case catalog (e.g. LOGIN_001)
- **Status**: one of Pass / Fail / Blocked / Skipped
- **Expected**: a one-sentence summary of the intended outcome
- **Actual**: a one-sentence summary of what really happened

6.3.1. Test Case Results

6.3.1.1. General Features

Test Case ID: GEN_001

- Status: Pass
- Expected: User-specific data is correctly scoped so that User A sees only their data, User B sees only theirs, and no data leaks across users.
- Actual: User-specific data was correctly scoped—User A saw only their data, User B saw only theirs, with no leakage.

Test Case ID: GEN_002

- **Status**: Pass
- **Expected**: Assessment due dates display correctly; submissions before the due date are accepted, and submissions after the due date are handled (marked late or prevented) per requirements.
- **Actual**: Due dates displayed correctly; early submissions were accepted, late submissions were marked "late" as specified, and submission prevented where required.

Test Case ID: GEN_003

- **Status**: Pass
- **Expected**: Students initially see a "pending" status for un-graded assessments; once the teacher grades and releases results, the grade becomes visible.
- **Actual**: "Pending" appeared for un-graded work; after grading and release by the teacher, grades were visible exactly as intended.

Test Case ID: GEN_004

- **Status**: Pass
- **Expected**: Unauthenticated users are redirected to login when accessing any dashboard; students, teachers, and admins can access only the dashboards and features their roles permit.
- **Actual**: All role-based access controls worked—unauthenticated requests went to login, and each role saw only their permitted dashboards/features.

Test Case ID: GEN_005

- **Status**: Pass
- **Expected**: Upon login, Students, Teachers, and Admins are each redirected to their respective dashboards.
- **Actual**: Login redirections worked flawlessly—Students went to the Student dashboard, Teachers to the Teacher dashboard, and Admins to the Admin dashboard.

6.3.1.2. Login and Registration

Test Case ID: LOG_001

- **Status**: Pass
- **Expected**: valid student credentials redirect to the student dashboard with no errors
- **Actual**: student was redirected to the dashboard with no error messages.

Test Case ID: LOG_002

- **Status**: Pass
- **Expected**: valid teacher credentials on the single login page redirect to the teacher dashboard with no errors
- **Actual**: teacher was redirected to their dashboard with no error messages.

Test Case ID: LOG_003

- **Status**: Pass
- **Expected**: valid admin credentials on the admin login page redirect to the admin dashboard with no errors
- **Actual**: admin was redirected to the dashboard with no error messages.

Test Case ID: LOG_004

- **Status**: Pass
- **Expected**: invalid credentials prevent login and display an appropriate error message without redirection
- **Actual**: login was blocked, “Invalid credentials” message appeared, and no redirection occurred.

Test Case ID: LOG_005

- **Status**: Pass
- **Expected**: no public links to teacher registration exist and direct access is blocked or requires special approval
- **Actual**: teacher signup link was hidden, direct URL access was blocked, and students could not discover the registration page.

Test Case ID: LOG_006

- Status: Pass

- Expected: forgot-password flow sends a reset email, allows password change via the link, old password is invalidated, and new password logs in successfully
- Actual: reset email was received, password was changed via the link, old password no longer worked, and new password logged in correctly.

6.3.1.3. Admin Features

Test Case ID: ADM_001

- **Status**: Pass
- **Expected**: only Admin users can access admin dashboard URLs—unauthenticated users are redirected, Students/Teachers see "unauthorized" or are redirected, and Admins access successfully
- **Actual**: access control worked exactly as expected.

Test Case ID: ADM_002

- **Status**: Pass
- **Expected**: Admin can navigate to the Teacher List page and see a table of registered teachers with Name, Email, Status, and Date Registered
- **Actual**: the teacher list displayed with all expected columns.

Test Case ID: ADM_003

- **Status**: Pass
- **Expected**: Admin can open the Register Teacher form, submit valid details, see a success message, and find the new teacher in the list (and trigger an invite email if configured)
- **Actual**: New teacher was created, confirmation appeared, and the account showed up in the list.

Test Case ID: ADM_004

- **Status**: Pass
- **Expected**: submitting the Teacher Registration form with empty fields, bad email format, or duplicate email yields appropriate error messages and blocks submission
- **Actual**: each invalid input triggered the correct validation message and prevented form submission.

Test Case ID: ADM_005

- Status: Pass
- Expected: Admin can fill out the Create Class form, assign an existing teacher, see a success message, and find the class in the class list
- Actual: class was created, teacher assignment succeeded, and the class appeared in the list.

Test Case ID: ADM_006

- **Status**: Pass
- **Expected**: Admin can filter to view all students not assigned to any class and see a list of those unassigned students

- **Actual**: the “Unassigned Students” view showed exactly those students.

6.3.1.4. Teacher Features

Test Case ID: TCH_001

- **Status**: Pass
- **Expected**: only Teachers (and optionally Admins per design) can access teacher dashboard URLs while Students and unauthenticated users are redirected or shown “unauthorized”
- **Actual**: access control behaved exactly as intended for all roles.

Test Case ID: TCH_002

- **Status**: Pass
- **Expected**: Teacher can change their password via the settings page, see a success message, log in with the new password, and old password no longer works
- **Actual**: password update flow completed successfully with all expected behaviors.

Test Case ID: TCH_003

- **Status**: Pass
- **Expected**: the CreateExam form loads with fields for title, description, due date, class assignment, and problem inputs
- **Actual**: test-creation UI rendered correctly with all required elements.

Test Case ID: TCH_004

- **Status**: Pass
- **Expected**: Teacher can fill out title, description, due date, assign to classes, add two problems, and save the test with a success notification and the test listed in “My Tests”
- **Actual**: test creation succeeded, confirmation appeared, and the new exam showed up in the list.

Test Case ID: TCH_005

- **Status**: Pass
- **Expected**: “My Tests” section displays all tests created by the Teacher with details like title, due date, submissions count, and status
- **Actual**: created tests list showed correctly with all expected information.

Test Case ID: TCH_006

- **Status**: Pass
- **Expected**: Teacher can select any test with submissions and open the grading interface showing the student’s submission, problem statement, and grading input area
- **Actual**: submission review page loaded as specified with all required components.

6.3.1.5. Student Features

Test Case ID: STU_001

- **Status**: Pass
- **Expected**: upon login, the student is redirected to Home.tsx and sees WelcomeBanner, QuickStats, UpcomingTest, RecentOverview/RecentResult, and sidebar navigation links
- **Actual**: the dashboard loaded Home.tsx with all specified UI components and navigation links present.

Test Case ID: STU_002

- **Status**: Pass
- **Expected**: navigating to the profile page displays the student's Name, User ID, Email, and Class (read-only where applicable)
- **Actual**: the profile page showed all expected fields correctly.

Test Case ID: STU_003

- **Status**: Pass
- **Expected**: editable profile fields (e.g., display name, contact number) can be modified, saved, and persist after reload or re-login
- **Actual**: changes to editable fields were saved and persisted as intended.

Test Case ID: STU_004

- **Status**: Pass
- **Expected**: selecting an active test from Tests/Index.tsx redirects to Tests/Show.tsx, showing problem statements, code editor or file-upload areas, and the due date
- **Actual**: the active test page rendered with the problem statements, CodeEditor component, and due date displayed.

Test Case ID: STU_005

- **Status**: Pass
- **Expected**: entering code into CodeEditor and submitting shows a confirmation, accepts the submission, and prevents or updates resubmissions per rules
- **Actual**: the code submission was accepted, confirmation appeared, and the teacher now has access to the submission.

Test Case ID: STU_006

- **Status**: Pass
- **Expected**: uploading a file for a file-upload problem associates the file with the submission, shows a confirmation, and makes it available to the teacher
- **Actual**: file upload succeeded, confirmation appeared, and the file was accessible in the teacher's view.

6.3.2. ML Model Evaluation Results

Okay, here's a section for your model evaluation results for Python submissions, incorporating the classification report into a table and discussing the findings.

6.3.2.1. Model Evaluation Results for Python Submissions

This section presents the performance evaluation of the predictive model on the dataset of Python code submissions. The evaluation aims to assess the model's ability to correctly classify submissions into one of seven predefined verdict categories: Accepted, Wrong Answer, Time Limit Exceeded, Memory Limit Exceeded, Runtime Error, and Presentation Error. The performance is quantified using a detailed classification report and a confusion matrix.

The model achieved an overall accuracy of 52% on the test set of 81,454 Python submissions. While this indicates that the model correctly predicts the verdict for over half of the instances, a more detailed breakdown by class, as shown in the classification report below, reveals varying performance across different verdict types.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Accepted	0.56	0.61	0.58	26137
Wrong Answer	0.48	0.52	0.50	20800
Time Limit Exceeded	0.53	0.82	0.64	16053
Memory Limit Exceeded	0.00	0.00	0.00	306
Runtime Error	0.48	0.09	0.15	16574
Presentation Error	0.58	0.92	0.71	1584

Accuracy			**0.52**	81454
Macro Avg	0.44	0.49	0.43	81454
Weighted Avg	0.51	0.52	0.49	81454

Table 35. Classification Report for Python Submissions

From the classification report, several observations can be made:

- The model performed best on the "Presentation Error" class, achieving an F1-score of 0.71, largely driven by a high recall of 0.92. This suggests the model is effective at identifying submissions with this specific issue when they occur.
- "Time Limit Exceeded" also showed relatively strong performance with an F1-score of 0.64 and a high recall of 0.82, indicating the model's proficiency in recognizing submissions that exceed execution time limits.
- Performance for "Accepted" submissions (F1-score: 0.58) and "Wrong Answer" submissions (F1-score: 0.50) was moderate.
- A significant challenge was observed with the "Memory Limit Exceeded" class. The model failed to correctly identify any instances of this verdict, resulting in precision, recall, and F1-score of 0.00. This is likely due to the extremely low support for this class (306 samples), making it difficult for the model to learn distinguishing features.
- The "Runtime Error" class also proved difficult for the model, yielding a very low F1-score of 0.15. This poor performance is primarily due to an exceptionally low recall of 0.09, meaning the model frequently failed to identify actual runtime errors.
- The macro average F1-score of 0.43 highlights the model's varied performance across classes, particularly its struggles with less frequent or more ambiguous error types. The weighted average F1-score of 0.49 is more influenced by the performance on classes with higher support.

The confusion matrix below provides a more granular view of the misclassifications:

Figure 42. Confusion Matrix for Python Submissions

Key insights from the confusion matrix include:

- A substantial number of actual "Runtime Error" submissions were misclassified, most commonly as "Accepted" (6,418 instances), "Wrong Answer" (4,057 instances), and "Time Limit Exceeded" (4,303 instances). This directly explains the low recall observed for the "Runtime Error" class.
- There is notable confusion between "Accepted" and "Wrong Answer" verdicts. For instance, 5,627 "Accepted" submissions were incorrectly predicted as "Wrong Answer," and conversely, 5,377 "Wrong Answer" submissions were misclassified as "Accepted."

- For the "Memory Limit Exceeded" class, while having very few true positives (0), the misclassified instances were most frequently predicted as "Time Limit Exceeded" (156 instances).

In summary, the Python model demonstrates a moderate overall predictive capability but exhibits significant performance disparities across different verdict classes. While effective for some common verdicts like "Time Limit Exceeded" and "Presentation Error," it struggles considerably with underrepresented classes like "Memory Limit Exceeded" and, critically, in accurately identifying "Runtime Error" submissions. These findings suggest areas for future improvement, potentially through targeted data augmentation for minority classes, advanced feature engineering to better distinguish subtle error types, or employing different modeling techniques more robust to class imbalance.

6.3.2.2. Model Evaluation Results for C++ Submissions

This section presents the performance evaluation of the predictive model on the dataset of C++ code submissions. The evaluation focuses on the model's capability to accurately classify C++ submissions into seven predefined verdict categories: Accepted, Wrong Answer, Time Limit Exceeded, Memory Limit Exceeded, Runtime Error, Compile Error, and Presentation Error. The model's performance is analyzed using a detailed classification report and a confusion matrix.

The model achieved an overall accuracy of 53% on the test set, which consisted of 91,450 C++ submissions. This accuracy indicates that the model correctly predicts the verdict for more than half of the instances. A more detailed class-wise performance breakdown is provided in the classification report below.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Accepted	0.57	0.59	0.58	26079
Wrong Answer	0.52	0.41	0.46	21485
Time Limit Exceeded	0.52	0.56	0.54	15543

Memory Limit Exceeded	0.00	0.00	0.00	1524
Runtime Error	0.37	0.42	0.39	16576
Compile Error	0.00	0.00	0.00	1541
Presentation Error	0.70	0.98	0.82	8702
Accuracy			**0.53**	91450
Macro Avg	0.38	0.42	0.40	91450
Weighted Avg	0.51	0.53	0.51	91450

Table 36. Classification Report for C++ Submissions

Key observations from the classification report for the C++ model include:

- The model demonstrated its strongest performance on the "Presentation Error" class, achieving an impressive F1-score of 0.82. This is driven by a very high recall of 0.98 and good precision of 0.70, indicating the model is highly effective at identifying these errors.
- Performance for "Accepted" submissions was moderate, with an F1-score of 0.58. "Time Limit Exceeded" also showed moderate performance with an F1-score of 0.54.
- The "Wrong Answer" class had a lower F1-score of 0.46, primarily due to a recall of 0.41.
- Consistent with the findings for Python and Java, the C++ model failed entirely to predict "Memory Limit Exceeded" and "Compile Error" verdicts, with precision, recall, and F1-scores of 0.00 for both. This is likely due to a combination of factors including class imbalance (though support for these is higher than in Python's MLE) and the inherent difficulty in distinguishing these specific error signatures.
- The "Runtime Error" class also showed weaker performance with an F1-score of 0.39.
- The macro average F1-score of 0.40 highlights the significant challenge the model faces with several classes. The weighted average F1-score of 0.51 is heavily influenced by the better performance on "Accepted" and "Presentation Error" due to their higher support.

The confusion matrix below provides a granular view of the misclassifications:

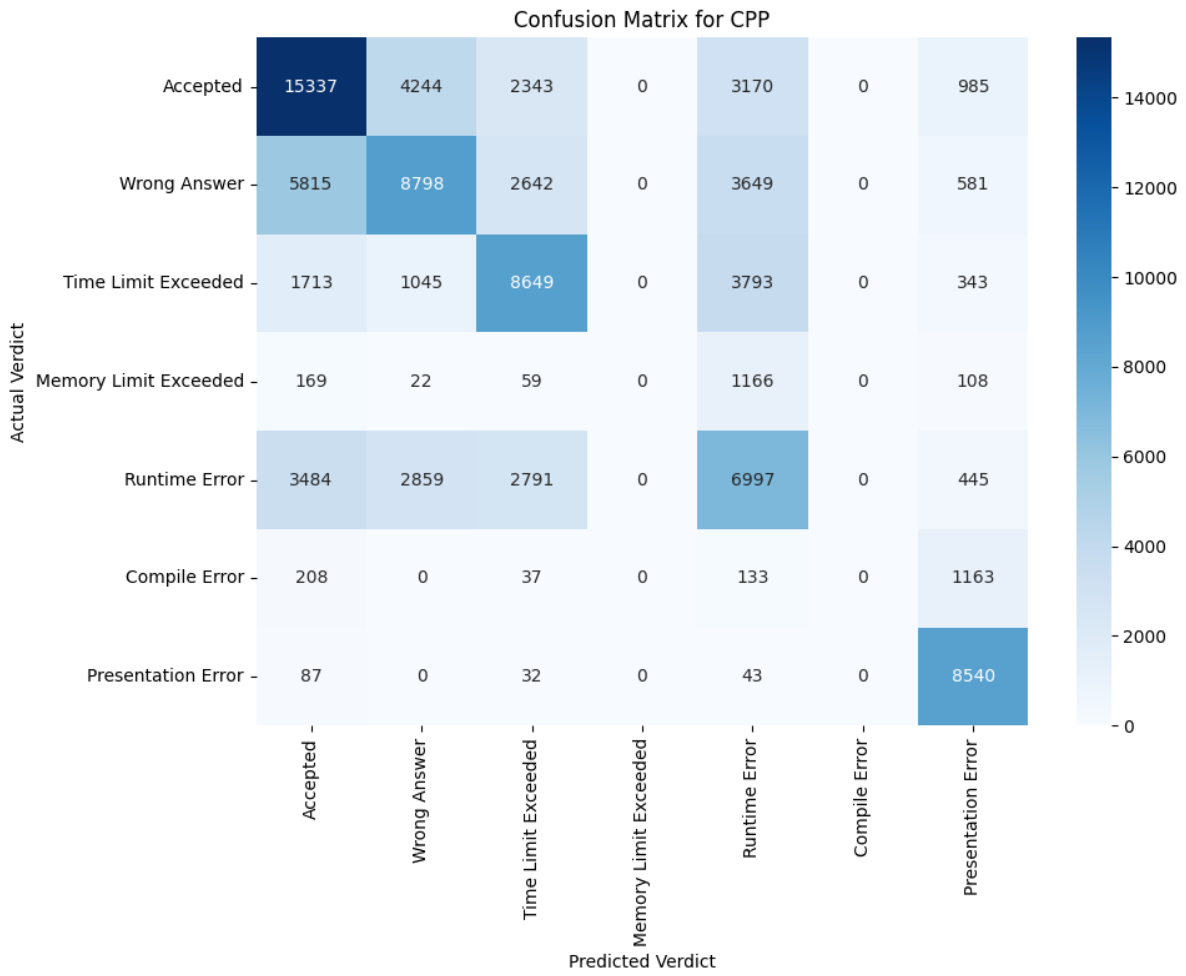

Figure 43. Confusion Matrix for C++ Submissions

Key insights from the confusion matrix for the C++ model include:

- The model correctly identified a large number of "Presentation Error" submissions (8,540 true positives), aligning with its high recall for this class.
- Significant misclassifications occurred for "Runtime Error" submissions, with many being predicted as "Accepted" (3,484 instances), "Wrong Answer" (2,859 instances), and "Time Limit Exceeded" (2,791 instances).
- Both "Memory Limit Exceeded" and "Compile Error" had no true positives. "Memory Limit Exceeded" submissions were most often misclassified as "Runtime Error" (1,166 instances). "Compile Error" submissions were predominantly misclassified as "Presentation Error" (1,163 instances).

- There is notable confusion between "Accepted" and "Wrong Answer" verdicts, a pattern observed across all language models.

In summary, the C++ model achieved an overall accuracy of 53%. While it excels at identifying "Presentation Error," its performance is moderate for "Accepted" and "Time Limit Exceeded" verdicts. Critically, the model completely fails to identify "Memory Limit Exceeded" and "Compile Error" submissions and shows limited capability in correctly classifying "Runtime Error" and "Wrong Answer" instances. These results underscore the challenges in creating a generalized verdict prediction model, particularly for error types that are either underrepresented or possess highly nuanced distinguishing features in C++ code. Future efforts could explore language-specific feature engineering, advanced techniques for handling severe class imbalance, or perhaps separate models for particularly problematic error types.

6.3.2.3. Model Evaluation Results for Java Submissions

This section details the performance evaluation of the predictive model when applied to the dataset of Java code submissions. The primary objective is to assess the model's proficiency in classifying Java submissions into seven distinct verdict categories: Accepted, Wrong Answer, Time Limit Exceeded, Memory Limit Exceeded, Runtime Error, Compile Error, and Presentation Error. Model performance is analyzed through a comprehensive classification report and a confusion matrix.

The model achieved an overall accuracy of 51% on the test set, which comprised 75,717 Java submissions. This accuracy figure indicates that the model correctly predicts the verdict for slightly over half of the instances. However, as with the Python model, performance varies significantly across different verdict types, as detailed in the classification report below.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Accepted	0.55	0.63	0.59	23682
Wrong Answer	0.53	0.40	0.46	18260
Time Limit Exceeded	0.55	0.75	0.63	12631

Memory Limit Exceeded	0.00	0.00	0.00	773
Runtime Error	0.38	0.26	0.31	13894
Compile Error	0.29	0.07	0.11	3278
Presentation Error	0.40	0.87	0.55	3199
Accuracy			**0.51**	75717
Macro Avg	0.38	0.43	0.38	75717
Weighted Avg	0.49	0.51	0.48	75717

Table 37. Classification Report for Java Submissions

Key observations from the classification report for the Java model include:

- Similar to the Python model, the Java model performed best in identifying "Time Limit Exceeded" submissions, achieving an F1-score of 0.63, supported by a strong recall of 0.75.
- "Presentation Error" also showed reasonable performance with an F1-score of 0.55 and a very high recall of 0.87, suggesting the model is adept at recognizing this particular verdict type.
- The F1-score for "Accepted" submissions was 0.59, indicating moderate performance.
- Performance for "Wrong Answer" submissions was lower, with an F1-score of 0.46, primarily due to a lower recall of 0.40.
- The "Memory Limit Exceeded" class, consistent with the Python model's results, showed no predictive capability (precision, recall, and F1-score of 0.00). This is again likely attributable to the relatively low support (773 samples) for this class.
- The model struggled significantly with "Runtime Error" (F1-score: 0.31, recall: 0.26) and particularly "Compile Error" (F1-score: 0.11, recall: 0.07). The extremely low recall for "Compile Error" indicates the model rarely identifies these issues correctly.
- The macro average F1-score of 0.38 underscores the model's inconsistent performance across classes, especially its difficulty with less frequent or more complex error types. The weighted average F1-score of 0.48 is influenced by the more frequent classes.

The confusion matrix below provides a detailed breakdown of the model's predictions versus the actual verdicts:

Figure 44. Confusion Matrix for Java Submissions

Insights from the confusion matrix for the Java model include:

- A significant number of actual "Runtime Error" submissions were misclassified, most notably as "Accepted" (4,821 instances) and "Time Limit Exceeded" (2,512 instances).
- "Compile Error" submissions were frequently misclassified, with a large portion predicted as "Presentation Error" (1,289 instances) and "Runtime Error" (1,506 instances). This contributes to the very low recall for "Compile Error."

- Considerable confusion exists between "Accepted" and "Wrong Answer" verdicts, with 6,108 "Wrong Answer" submissions misclassified as "Accepted," and 3,013 "Accepted" submissions misclassified as "Wrong Answer."
- For the "Memory Limit Exceeded" class, which had no correct predictions, the majority of its instances (531) were misclassified as "Time Limit Exceeded."

In conclusion, the Java model exhibits an overall predictive accuracy of 51%, with notable strengths in identifying "Time Limit Exceeded" and "Presentation Error" verdicts. However, its performance is substantially weaker for "Runtime Error" and especially "Compile Error" and "Memory Limit Exceeded." The significant misclassification rates for these error types indicate challenges in distinguishing their specific features from other verdicts. Future work should focus on strategies to improve performance on these underperforming classes, potentially through techniques like class weighting, specialized feature engineering for Java-specific errors, or exploring ensemble methods.

Chapter Seven: Conclusions and Recommendations

7.1. Conclusion of the Study

The Automated Code Review E-Learning System represents a significant advancement in programming education, leveraging Graph Neural Networks (GNNs) and Lightweight Large Language Models (LLMs) to automate code submission, evaluation, and feedback generation. This platform addresses the challenges of manual code review, such as time-intensive grading and inconsistent feedback, offering a scalable, secure, and accessible solution that enhances learning outcomes for students and streamlines grading for teachers. By integrating modern technologies like Laravel, React.js, and PyTorch, the system delivers a user-friendly interface and AI-driven insights, supporting educational centers mission to advance educational technology and fostering skill development in programming.

The project commenced with a thorough analysis of users' needs, identifying challenges faced by students and teachers through surveys and stakeholder interviews, which highlighted the demand for automated, rapid feedback and role-based access. Clear objectives were established: to create a secure, scalable system with GNN/LLM-based feedbac. Using an agile methodology, the team designed and implemented a solution incorporating automated feedback, role-based authentication, and dynamic interfaces, supported by a robust database and ML model integration.

Development involved configuring Laravel for authentication, authorization and handling user interactions, React.js for dynamic interfaces, and Flask for ML model deployment, with iterative testing to refine functionality. The system’s effectiveness was validated through user testing, confirming accurate feedback within 5 seconds, accessibility compliance, and scalability, aligning with stakeholder expectations and educational goals.

Key outcomes include a fully operational platform that automates code review, reducing teacher workload and improving student learning through timely, AI-generated feedback. Lessons learned include the value of modular design, as Laravel’s architecture simplified updates, and the challenge of ML integration, where Flask API latency required optimization. Overall, this project delivers a robust, extensible system that meets its objectives, laying a foundation for future innovations in automated code review at educational centers. Expanding support for additional programming languages or integrating real-time collaboration could further enhance its impact on programming education.

7.2. Recommendations of the Study

To ensure the continuous improvement and long-term impact of the Automated Code Review E-Learning System, it is essential to strategically address the limitations identified during development while also exploring avenues for innovation and expansion. The following recommendations are grounded in the need to enhance system usability, performance, accessibility, and educational value.

7.2.1. Enhance Machine Learning Model Performance

To overcome the limited accuracy of the current machine learning component (53% for C++, 52% for Python, 51% for Java), future work should focus on expanding and diversifying the training dataset, particularly for underrepresented verdicts like *Memory Limit Exceeded (MLE)*. Additionally, integrating more advanced ML techniques, such as transfer learning and fine-tuning large pre-trained language models (e.g., CodeBERT, GraphCodeBERT), may significantly improve precision and recall across verdict classes. Collaboration with AI research groups could help develop more robust and context-aware feedback mechanisms.

7.2.2. Introduce Export Functionality for Grading Reports

To address the absence of export capabilities, future iterations should include features that allow users, particularly instructors and administrators, to export grading summaries, performance insights, and feedback reports in formats such as PDF and CSV. This functionality would support record-keeping, offline review, and institutional reporting, enhancing the system's practical utility in academic settings.

7.2.3. Improve Accessibility Compliance

Given the system's current partial compliance with WCAG 2.1, deliberate efforts should be made to audit and improve accessibility features. This includes adding keyboard navigation support, screen reader compatibility, and sufficient contrast for users with visual impairments. Providing alternative text for icons and ensuring responsive design would improve usability for users with different needs, ultimately making the system more inclusive.

7.2.4. Incorporate Interactive User Guidance

To support novice users, especially those unfamiliar with code submission platforms, the system should integrate in-app guidance tools such as tooltips, walkthroughs, and short video tutorials. These features can help students navigate key use cases like *Submit for Evaluation* and *View Feedback* more effectively, reducing the learning curve and improving user satisfaction.

7.2.5. Support Additional Programming Languages

To broaden the system's appeal and applicability, support for additional languages such as C#, JavaScript, or Go could be introduced. This expansion would involve adapting the existing GNN/LLM feedback generation models or training new ones on relevant datasets. Supporting a wider range of languages increases the platform's educational reach and aligns it with diverse programming curricula.

7.2.6. Facilitate Localization for Regional Users

Localizing the interface and feedback messages into Ethiopian languages (e.g., Amharic, Afan Oromo) would improve comprehension and accessibility for regional users. This effort would not only align with accessibility goals but also foster digital inclusivity and increase adoption among local institutions.

7.2.7. Explore Gamification and Peer Collaboration

To enhance engagement, consider integrating gamification features such as badges, progress tracking, leaderboards, and peer review forums. These elements can create a more interactive and motivating learning environment, especially when applied to use cases like *Access Code Insights* or *Submit for Evaluation*. Research into effective gamification strategies in education should inform this development.

References

[1] Seckin-Kapucu, Munise, and Abdulhalim Batu. 2022. “Analysis of Studies on Coding Education: A Meta-Synthesis Study.” *The Eurasia Proceedings of Educational & Social Sciences (EPESS)* 24. http://www.isres.org/.

[2] Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. “How Powerful Are Graph Neural Networks?” *International Conference on Learning Representations (ICLR)*. arXiv:1810.00826.

[3] Chen Zhu-Tian, Xiong Z., Yao X, and Glassman E. 2024. “Sketch then generate: Providing incremental user feedback and guiding LLM code generation through language-oriented code sketches.” arXiv (Preprint). https://arxiv.org/abs/2405.03998.

[4] Hussam Aldriye, Asma Alkhalaf and Muath Alkhalaf, “Automated Grading Systems for Programming Assignments: A Literature Review” International Journal of Advanced Computer Science and Applications(IJACSA), 10(3). 2019. pp 216-217,

[5] Marcus Messer, Neil C. C. Brown, Michael Kölling, Miaojing Shi, "Automated Grading and Feedback Tools for Programming Education: A Systematic Review", ACM Transactions on Computing Education, 2023. Section 3.2.3.

[6] N. Nielson, F. Nielson, and H. R. Hankin, Principles of Program Analysis. Springer, 1999 pp 1-2.

[7] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, “A Survey of Machine Learning for Big Code and Naturalness,” ACM Computing Surveys (CSUR) pp. 22, Jul. 2018.

[8] Wichmann, B. A.; Canning, A. A.; Clutterbuck, D. L.; Winsbarrow, L. A.; Ward, N. J.; Marsh, D. W. R. (Mar 1995). "Industrial Perspective on Static Analysis" (PDF). *Software Engineering Journal*. **10** (2): 69–75.

[9] Egele, Manuel; Scholte, Theodoor; Kirda, Engin; Kruegel, Christopher (2008-03-05). "A survey on automated dynamic malware-analysis techniques and tools". *ACM Computing Surveys*. **44** (2): 6:1–6:42

[10] *Improving Software Security with Precise Static and Runtime Analysis* Archived 2011-06-05 at the Wayback Machine (PDF), Benjamin Livshits, section 7.3 "Static Techniques for Security". Stanford doctoral thesis, 2006.

[11] L. Wu, P. Cui, J. Pei, and L. Zhao, Graph Neural Networks: Foundations, Frontiers, and Applications, Chapter 22: Graph Neural Networks in Program Analysis by M. Allamanis. Springer, 2022.

[12] J. Hellendoorn and E. T. Barr, “ECHO: Reusing feedback in automated code reviews,” in Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Tallinn, Estonia, 2019, pp. 118–128. DOI: 10.1145/3338906.3338937.

[13] S. Chen, H. Zhu, and Y. Xiong, “DeepReview: Automatic code review using deep multi-instance learning,” in Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Tallinn, Estonia, 2019, pp. 556–567. DOI: 10.1145/3338906.3338937.

[14] S-Logix, "Research Topics in Graph Neural Networks," [Online]. Available: https://slogix.in/machine-learning/graph-neural-networks/.

[15] GeeksforGeeks, "What are Graph Neural Networks?", [Online]. Available: https://www.geeksforgeeks.org/what-are-graph-neural-networks/.

[16] R. Puri et al., "Project CodeNet: A Large-Scale AI for Code Dataset," *arXiv*, May 2021. [Online]. Available: https://arxiv.org/abs/2105.12655.

Appendices

Appendix A: IBM’s Project Code Net Dataset

The machine learning models developed and evaluated within this capstone project were trained using data derived from IBM's Project CodeNet. This appendix outlines the characteristics of this dataset and details its specific application to the current work, ensuring transparency and facilitating reproducibility.

A.1. Dataset Overview

Project CodeNet, an initiative by IBM Research, stands as a significant open-source resource for advancing AI for Code. It is a large-scale compilation of code submissions, primarily gathered from prominent online competitive programming platforms like AIZU Online Judge and AtCoder. The official portal for accessing Project CodeNet and its associated documentation can be found at https://dax-cdn.cdn.appdomain.cloud/dax-project-codenet/1.0.0/readme.html. The seminal publication detailing the dataset is "Project CodeNet: A Large-Scale AI for Code Dataset" by Puri et al. (2021) [16].

A.2. Key Characteristics

The dataset is distinguished by its considerable scale, encompassing approximately 14 million code samples that collectively amount to around 50 million lines of code. This extensive collection spans over 55 different programming languages, with notable representation for widely-used languages such as C++, Java, Python, and C. Each code submission is accompanied by rich metadata, including the problem identifier, user ID, programming language used, submission timestamp, code size, and the execution status.

Furthermore, problem descriptions, often available in multiple natural languages, are included. The dataset's origin from competitive programming environments ensures a broad diversity in algorithmic challenges and submitted coding styles, making it a robust resource for training sophisticated AI models.

A.3. Dataset Utilization in this Project

For the specific objectives of this capstone project, a tailored approach was adopted for utilizing the Project CodeNet dataset to train a model capable of predicting submission verdicts. A significant subset of code submissions was extracted, focusing on three prevalent programming languages: C++, Python, and Java.

Specifically, the selected sample comprised 91,450 submissions in C++, 81,454 submissions in Python, and 75,717 submissions in Java. This collection was intentionally curated to include a diverse range of submission outcomes, with verdicts encompassing "Accepted," "Wrong Answer," "Time Limit Exceeded," "Memory Limit Exceeded," "Runtime Error," "Compile Error," and "Presentation Error."

Furthermore, approximately 1,300 unique programming problems associated with these submissions were identified and underwent necessary preprocessing. This curated dataset, consisting of problem statements, corresponding code submissions in one of the three chosen languages, and their associated verdicts, formed the basis for training our predictive model. The model's objective is to take a given problem statement and a code submission as input and predict the probabilities for each of the seven possible verdict categories. The data employed for this project was accessed and downloaded from the official Project CodeNet repository.

A.4. Rationale for Use

Project CodeNet was selected as the primary data source for several compelling reasons. Its extensive collection of diverse code samples, rich metadata, and specific design to support AI for Code research provided an invaluable foundation for our work. The availability of problem solutions across multiple languages, coupled with the inclusion of varied submission statuses, offered precisely the data points required for training a robust verdict prediction model.

Furthermore, as a student-led capstone project with limited resources and bandwidth, the task of independently collecting and curating a dataset of comparable size, diversity, and quality would have been infeasible. Project CodeNet, therefore, provided a critical, high-quality, and readily accessible resource that enabled the ambitious scope of this project, particularly in developing a model capable of understanding and predicting outcomes for a wide array of code submissions.