GradeFusion.ai

Submitted in partial fulfillment of the requirements
for the degree of

B.E. Computer Engineering

By

Rohit Tarke	**Roll No:12**	**212114**
Aniket Patil	**Roll No:23**	**212080**
Tanmay Mohod	**Roll No:28**	**212062**
Aryan Pange	**Roll No:36**	**212124**

Guide
Dr. Bidisha Roy
Associate Professor

Department of Computer Engineering
St. Francis Institute of Technology
(Engineering College)
An Autonomous Institute, Affiliated to University of Mumbai

University of Mumbai
2024-2025

SFIT LIBRARY 2025

CERTIFICATE

This is to certify that the project entitled **"GradeFusion.ai"** is a bonafide work of **"Rohit Tarke (Roll no.12), Aniket Patil (Roll no.23), Tanmay Mohod (Roll no.28) and Aryan Pange (Roll no.36)"** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of B.E. in Computer Engineering.

SIT LIBRARY 2025

(Dr. Bidisha Roy)

Guide

(Dr. Kavita Sonawane)

Head Of Department

(Dr. Sincy George)

Principal

Project Report Approval for B.E.

This project report entitled *"GradeFusion.ai"* by ***Rohit Tarke (Roll no.18), Aniket Patil (Roll no.23), Tanmay Mohod (Roll no.28) and Aryan Pange (Roll no.36)*** is approved for the degree of ***B.E. in Computer Engineering***.

Examiners

1. _____ 30/04/25

2. _____ 30/04/25

Date: 30-04-25

Place: Mumbai

SFIT LIBRARY 2025

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included; we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in this submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Rohit Tarke (12)

Aniket Patil (23)

Tanmay Mohod (28)

Aryan Pange (36)

Date:

SFIT LIBRARY 2025

Abstract

This project aims to develop an online coding evaluation platform for academic use, enabling students to be assessed on coding skills in languages like C, C++, and Java. The platform allows teachers to upload coding challenges with example solutions and test cases, using an automated grading system to ensure fair and consistent scoring based on test case performance. This automation significantly reduces manual grading efforts, freeing teachers from the time-consuming task of evaluating each submission individually. In addition to streamlining grading, the platform provides real-time analytics on student performance, allowing teachers to quickly identify areas where students may struggle and adapt their teaching accordingly. With its user-friendly interface, the platform supports efficient learning by offering personalized coding suggestions and insights, enabling students to enhance their skills effectively while teachers benefit from data-driven teaching insights.

SFIT LIBRARY 2025

Contents

1 Introduction **1**

1.1 Description 1

1.2 Background study Terminologies/ Definitions of new terms 2

1.3 Fundamental Study Points of the Selected Topic and the Domain . 2

1.4 General Block diagram 4

2 Review of Literature **5**

2.1 Survey of Existing systems 5

2.2 Limitations of Existing Systems or Research Gaps 6

2.3 Motivation 7

3 Proposed System **8**

3.1 Analysis 8

3.1.1 Problem Statement and Proposed Solution 8

3.1.2 Scope of the System 9

3.1.3 Functional Requirements 9

3.1.4 Non-Functional Requirements 10

3.1.5 Specific Requirements 11

3.1.6 Use-Case Diagrams and Description 13

3.2 Workflow diagram for Proposed System and Description 15

3.2.1 Phase/Module-wise Explanation (as per Workflow) 16

3.3 Analysis and Modeling 19

3.3.1 Sequence Diagram 19

3.3.2 Activity Diagram 22

SIT LIBRARY 2025

3.4 Functional Modeling 23

4 Proposed System: Design 26

4.1 Architectural view 26

4.2 Architectural Style 27

4.2.1 Client-Side (Frontend) 27

4.2.2 Server-Side (Backend) 28

4.2.3 Workflow 29

4.2.4 Advantages of Client-Server Architecture for GradeFusion.ai 29

4.3 Algorithms 30

4.3.1 Integration of Algorithms in Student Output Validation . . 30

4.3.2 Algorithm for Marks Calculation Based on Test Results . 30

4.3.3 LLM's 33

5 Proposed System : Implementation Plan and Experimental Set up 35

5.1 Experimental Setup 35

5.1.1 Details of Database or Input to Systems 35

5.1.2 Performance Evaluation Parameters (for Validation) . . . 36

5.1.3 Special Requirements 36

5.2 Implementation Plan of this Semester 37

5.2.1 Phase 1: January - Week 1 to Week 3 (Initial Setup and Core Implementation) 37

5.2.2 Phase 2: January - Week 4 to February - Week 2 (Test Case Management and Initial AI Model Integration) . . . 38

5.2.3 Phase 3: February - Week 3 to Week 5 (Advanced AI Integration and Code Evaluation) 39

5.2.4 Phase 4: March - Week 1 to Week 2 (Testing and Refining the System) 40

5.2.5 Phase 5: March - Week 3 to Week 4 (Final Testing, Documentation, and Deployment) 40

SIT LIBRARY 2025

6 Result and Discussions **42**

6.0.1 Implementation Results 42

6.0.2 Comparative study with the existing systems 47

7 Conclusions **49**

7.1 Conclusion 49

7.2 Future Scope 49

References **50**

Acknowledgements **51**

SFIT LIBRARY 2025

List of Figures

1.1 Block diagram of GradeFusion.ai 4

3.1 Use-Case Diagram for GradeFusion.ai 13
3.2 Proposed System: Workflow diagram for GradeFusion.ai 15
3.3 User Sequence diagram of GradeFusion.ai 19
3.4 Teacher Sequence diagram of GradeFusion.ai 20
3.5 Student Sequence diagram of GradeFusion.ai 21
3.6 Activity diagram of GradeFusion.ai 22
3.7 DFD Level 0 for GradeFusion.ai 23
3.8 DFD Level 1 for GradeFusion.ai 24
3.9 DFD Level 2 for GradeFusion.ai 25

4.1 Architectural Diagram of GradeFusion.ai 26
4.2 Gemini 2.5 Pro Performance Across Benchmarks 34

6.1 Home Page Interface of GradeFusion.ai 42
6.2 Signup Page Interface of GradeFusion.ai 43
6.3 Student Dashboard Interface of GradeFusion.ai 44
6.4 Teacher Dashboard Interface of GradeFusion.ai 44
6.5 Coding Environment Interface of GradeFusion.ai 45
6.6 Coding Environment Interface of GradeFusion.ai 46
6.7 Coding Environment Interface of GradeFusion.ai 46

SIT LIBRARY 2025

Chapter 1

Introduction

SIT LIBRARY 2025

This project aims to develop a comprehensive online coding evaluation system for academic use, where students are assessed based on their ability to solve coding problems accurately and efficiently. Teachers can upload coding questions with example solutions, and the system will automatically generate multiple test cases to evaluate the accuracy, correctness, and efficiency of student submissions. Grading is based on the number of test cases passed, ensuring transparent, fair, and objective evaluations. The platform supports coding in C, C++, and Java, providing students with hands-on practice solving real-world problems while being objectively evaluated. Designed to enhance the learning experience, this user-friendly platform streamlines coding practice and evaluations, ultimately improving both teaching and learning outcomes in educational institutions.

1.1 Description

The Online Coding Evaluation Platform is a streamlined solution for academic coding assessments. This platform is designed to simplify the process of evaluating students' coding skills by allowing teachers to upload coding challenges with example solutions and automated test cases. Students can solve problems in languages like C, C++, and Java, and their submissions are evaluated objectively based on test case performance. With its user-friendly interface, this platform enhances both teaching and learning, making coding assessments fair, efficient, and accessible for educational institutions.

1.2 Background study Terminologies/ Definitions of new terms

Students often face challenges in understanding their coding strengths and weaknesses due to subjective or delayed evaluations, while teachers spend considerable time grading submissions manually. This project aims to address these challenges through an automated coding evaluation system that executes code submissions against predefined test cases, providing consistent, unbiased grading and real-time feedback. The platform leverages advanced AI models to offer students personalized coding suggestions and tips, enhancing the learning process by making it more interactive and adaptive. With a responsive interface developed using modern technologies, this system ensures a seamless experience for both teachers and students. Additionally, secure data management enables smooth feedback and authentication, making the evaluation process efficient and accessible for educational institutions.

1.3 Fundamental Study Points of the Selected Topic and the Domain

- **Automated Code Evaluation Systems:**
  - Understanding automated code evaluation systems is vital for developing a reliable online coding platform.
  - These systems assess students' code submissions by automatically verifying correctness and efficiency.
  - Key components include generating diverse test cases that cover various scenarios and employing standardized grading rubrics based on test outcomes.
  - Addressing challenges such as handling multiple programming languages and providing meaningful feedback is crucial for ensuring fair evaluations.
  - Mastering these concepts will lead to a robust system that accurately reflects students' coding abilities.

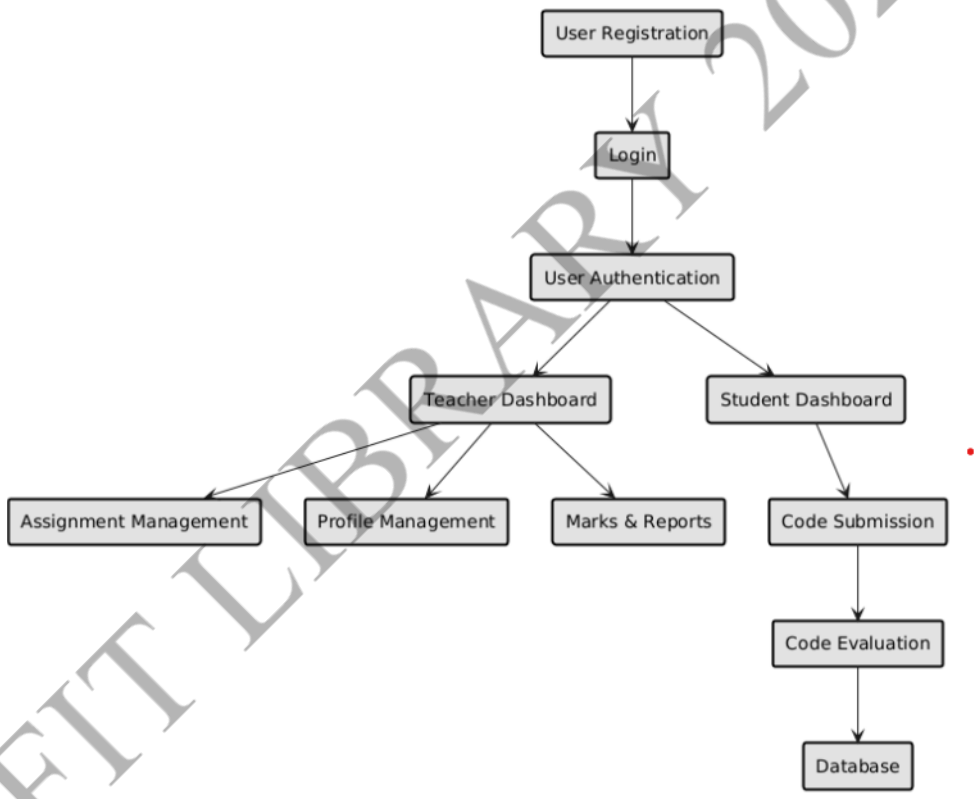
SFIT LIBRARY 2025

- **User Authentication and Authorization:**
  - Implementing secure user authentication and authorization processes is essential for protecting user data and ensuring that only authorized individuals access specific features of the platform.
  - Utilizing Firebase Authentication will help manage user accounts securely, while role-based access controls will differentiate functionalities between teachers and students.
  - Understanding best practices for user privacy and data protection is critical for maintaining trust and compliance in an academic setting.
  - This focus on security will enhance the platform's reliability and usability.
- **Evaluation Metrics for Code:**
  - Evaluation metrics for code submissions are crucial for assessing the quality of students' work.
  - Correctness determines if the code produces the expected output, while efficiency evaluates performance regarding time and space complexity.
  - Robustness assesses how well the code handles edge cases and unexpected inputs.
  - Although code readability is not graded, encouraging clear and maintainable coding practices enhances collaboration and fosters a better programming culture, contributing to a more effective learning experience.
- **Scalability :**
  - The platform must be scalable to handle increasing numbers of students and educators.

SIT LIBRARY 2025

- Using cloud infrastructure like Firebase ensures efficient scaling, fast performance, and real-time data sync, essential for large student cohorts or coding competitions.
- Future features may include real-time collaboration tools for teamwork and peer learning.

1.4 General Block diagram



SFIT LIBRARY 2025

Figure 1.1: Block diagram of GradeFusion.ai

This block diagram outlines a system's flow starting from user registration and login leading to user authentication. After authentication, users are directed to either the teacher or student dashboard. The teacher dashboard includes assignment management, profile management, and marks and reports. The student dashboard allows for code submission, which goes through code evaluation and is stored in a database. This structure ensures role-specific functionalities and data processing.

Chapter 2

Review of Literature

SIT LIBRARY 2025

2.1 Survey of Existing systems

Automated grading systems aim to provide consistent, efficient, and unbiased evaluations of programming assignments. Systems like Web-CAT, which automate grading through test cases, significantly reduce manual effort while offering instant feedback to students [1]. However, such systems often face challenges in providing feedback that is both clear and helpful, impacting students' learning experiences. AUTOGRADER, an approach based on formal semantics and symbolic execution, further refines the process by comparing student submissions to a reference implementation [2]. This method allows for more precise feedback by generating counterexamples when discrepancies occur, reducing the need for extensive test suites. Despite these advancements, limitations remain, such as the complexity of interpreting automated feedback by students, similar to the feedback challenges seen in earlier grading systems [1]. Building on these automated approaches, the Pytutor platform integrates software testing concepts into its online judge system to assess both programming and testing abilities [3]. Unlike traditional systems that only evaluate the correctness of code, Pytutor allows learners to define their own test cases, fostering a deeper understanding of software testing principles [3]. This aligns with the emphasis on enhancing feedback clarity and precision seen in AUTOGRADER [2]. By providing a more hands-on approach to testing, Pytutor helps students develop stronger software testing skills, leading to improved coding

practices overall.

Recent reviews of automated grading systems further highlight the shift toward supporting higher-level feedback mechanisms. Systems have evolved from merely checking code correctness to assessing aspects like software design, coding style, and logical thinking [4]. Additionally, the use of automated feedback generation techniques aims to provide personalized, actionable advice to learners, bridging the gap between human mentoring and fully automated grading [5]. Together, these studies highlight the evolution of automated grading systems toward more interactive, precise, and learner-centered feedback mechanisms. While Web-CAT and AUTOGRADER primarily focus on reducing manual grading efforts through automation [1][2], Pytutor and recent systems emphasize active engagement and personalized feedback [3][5]. This suggests a growing need for grading platforms that not only automate evaluations but also empower students to understand, debug, and improve their code through hands-on testing, ultimately offering a more comprehensive learning experience in programming education.

SITLIBRARY 2025

2.2 Limitations of Existing Systems or Research Gaps

Web-CAT: The primary limitation of Web-CAT lies in its feedback mechanism. While the system excels at automating grading through the use of predefined test cases, the feedback provided is often not sufficiently clear or instructive, which can hinder students' ability to understand and learn from their mistakes. This can impact the overall learning experience, as students may find it difficult to interpret the guidance and improve their code effectively.

AUTOGRADER: Although AUTOGRADER enhances precision in grading by utilizing formal semantics and symbolic execution, its complexity poses a challenge. The feedback, while technically accurate, can be difficult for students to interpret, particularly for those who are less familiar with formal methods. This complexity reduces the accessibility and usefulness of the feedback, indicating a gap in balancing technical accuracy with student comprehensibility.

Pytutor: Pytutor addresses some of the feedback limitations by integrating soft-

ware testing concepts and allowing students to define their own test cases. However, its approach is still limited in terms of scalability and the effectiveness of feedback for a broader range of programming problems. Additionally, while it fosters a deeper understanding of testing, there is room for improvement in how the system supports diverse learning styles and helps students fully grasp testing principles.

These individual limitations highlight the need for further research into grading platforms that provide clear, accessible, and comprehensive feedback while also promoting active engagement in learning.

2.3 Motivation

The increasing demand for skilled programmers and the rising popularity of computer science education have led to a surge in student enrollments in programming courses, placing significant pressure on educators to evaluate assignments efficiently and consistently. Traditional manual grading methods are time-consuming and prone to subjectivity and inconsistencies, making it difficult for instructors to provide timely, personalized feedback. To address these challenges, this project aims to develop an online coding evaluation platform designed for academic use. The platform automates the assessment process, reducing the manual workload for teachers while ensuring fair and transparent evaluations. By incorporating real-time feedback and analytics, students can immediately understand their mistakes, promoting iterative learning and improvement. Supporting programming languages such as C, C++, and Java, this comprehensive solution bridges the gap between coding education and efficient evaluation, ultimately enhancing the educational experience for both students and teachers.

SIT LIBRARY 2025

Chapter 3

Proposed System

SIT LIBRARY 2025

3.1 Analysis

3.1.1 Problem Statement and Proposed Solution

Assessing students' programming abilities through traditional coding exercises is often subjective, time-consuming, and prone to inconsistencies in scoring and feedback. Manual grading processes demand extensive effort from educators in evaluating coding tasks and executing test cases, which reduces the time available for providing meaningful feedback. There is a pressing need for an automated online coding evaluation platform that ensures efficient, objective, and consistent assessments of students' programming skills, particularly in languages like C, C++, and Java. We propose developing an online coding evaluation platform specifically designed for academic environments. The platform will allow educators to upload coding challenges along with test cases, which will be used to automatically evaluate student submissions for correctness, efficiency, and robustness. An automated grading system will ensure fair and consistent evaluation of coding tasks, alleviating the manual workload on educators. The platform will also provide real-time feedback and analytics on student performance, enabling teachers to quickly identify learning challenges and adjust their instructional strategies accordingly. This user-friendly system aims to enhance both teaching efficiency and student engagement in programming education, fostering a more streamlined and effective learning experience.

3.1.2 Scope of the System

The scope of this project includes building an online coding evaluation platform designed for schools, serving both teachers and students. Teachers will be able to upload coding questions and example solutions, while students can submit their code in C, C++, or Java. The platform will feature a user-friendly interface that allows students to receive immediate feedback on their submissions, helping them learn and improve their coding skills. This scope aims to provide an effective solution for coding evaluations in educational settings.

3.1.3 Functional Requirements

1. **User Registration and Authentication:**
   Users (students and teachers) can register on the platform by providing email, username, and password. Validation checks ensure the uniqueness of email and username during registration. A verification email is sent to confirm the user's email address after registration. Users can log in using their registered credentials . The system implements secure session management and multi-factor authentication (MFA).

2. **Coding Challenge Creation and Management:**
   Teachers can create new coding challenges with problem statements, example inputs/outputs, and constraints. An interface allows teachers to edit existing challenges and update parameters. Bulk upload functionality for test cases is provided to streamline challenge creation. Teachers can publish or unpublish challenges to control visibility to students.

3. **Code Submission and Evaluation:**
   Students have an interface to write and submit code in supported programming languages (C, C++, Java). The system allows students to test their code against sample inputs before final submission. Upon submission, the system automatically compiles and runs the code against predefined test

SIT LIBRARY 2025

cases. Real-time feedback on correctness and performance is provided to students.

4. **Feedback Generation:**
   The system generates detailed feedback for each code submission after evaluation. Feedback includes information on which test cases passed or failed, along with error messages for failures. Performance metrics such as execution time and memory usage are included in the feedback. Suggestions for code optimization and improvements are provided to enhance the learning experience.

5. **User Roles and Permissions:**
   The platform implements role-based access control for differentiating user functionalities. Roles such as ”Student,” ”Teacher,” and ”Admin” are defined, each with specific permissions. Students can submit solutions and view feedback, while teachers manage challenges and analytics. Middleware checks user permissions before granting access to specific features.

3.1.4 Non-Functional Requirements

1. **Scalability:**
   Requirement: The platform should support a large number of concurrent users without performance degradation.
   Details: Utilize cloud-based solutions like Firebase for real-time data synchronization and storage. Ensure the backend infrastructure can scale horizontally to handle increased traffic and load, supporting large classes and peak usage during exams.

2. **Usability:**
   Requirement: The system should be intuitive and easy to navigate for both students and teachers, ensuring a smooth user experience.
   Details: Design a user-friendly interface with a focus on accessibility. Include clear instructions and feedback for every action, such as code submission and test case results, to reduce user confusion.

SIT LIBRARY 2025

3. **Performance:**
   Requirement: The system should provide fast responses, especially during code compilation and test case execution, to maintain a seamless user experience.
   Details: Optimize code evaluation algorithms and ensure quick execution of test cases. Use efficient data management practices to reduce latency in real-time feedback, ensuring results are displayed within a few seconds after submission.

4. **Maintainability:**
   Requirement: The platform should be easy to maintain and update, allowing for future enhancements and quick resolution of issues.
   Details: Use modular development practices with React and TypeScript, allowing independent updates to components without affecting the entire system. Maintain comprehensive documentation for the codebase and regularly update it to assist future developers in understanding the system's architecture.

3.1.5 Specific Requirements

Hardware Specifications

- 8 GB RAM minimum, 16 GB RAM recommended.
- 1.6 GHz or faster processor.
- 200 MB of available disk space minimum, 500 MB recommended (for Visual Studio Code).

Software Specifications

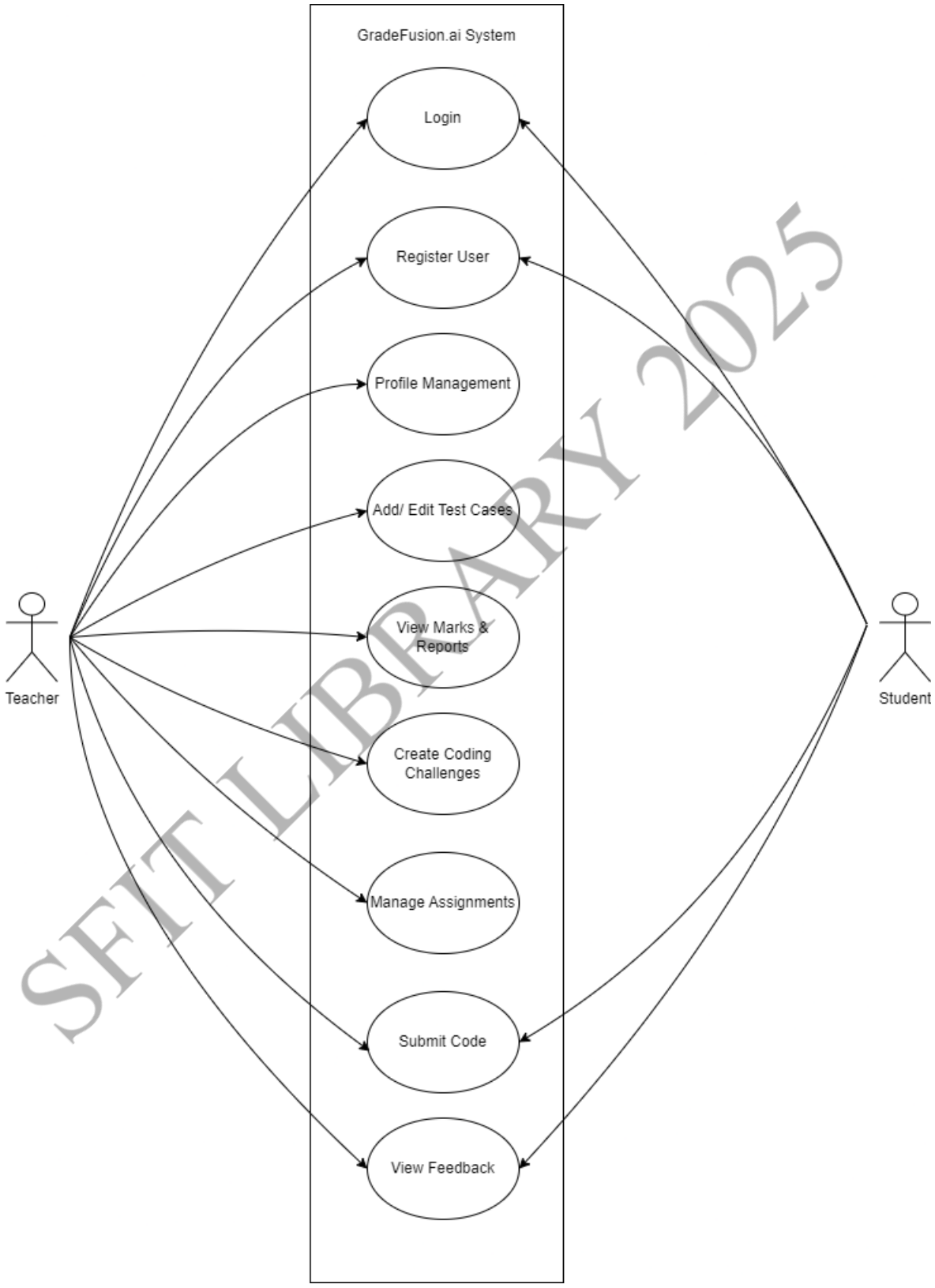
- **Web Development Tools:**
  - **React**: For building user interfaces.
  - **TypeScript**: For enhancing code quality with static typing.
  - **Tailwind CSS**: For designing responsive and modern UI.

SFIT LIBRARY 2025

- **Backend Development Tools:**
  - **Node.js**: For server-side JavaScript execution and building backend services.
  - **Express.js**: For creating RESTful APIs and routing.
  - **Firebase**: For user authentication and real-time database services.
- **Database:**
  - **Firebase Firestore**: NoSQL cloud database for storing user data and coding challenges.
- **Development Tools:**
  - **Visual Studio Code**: IDE for coding, debugging, and managing the project.
  - **Git**: Version control system for managing source code.
  - **Postman**: For API development and testing.
- **User Authentication Libraries:**
  - **Firebase Authentication**: For managing secure user sign-up and login processes.

SFIT LIBRARY 2025

3.1.6 Use-Case Diagrams and Description



SFIT LIBRARY 2025

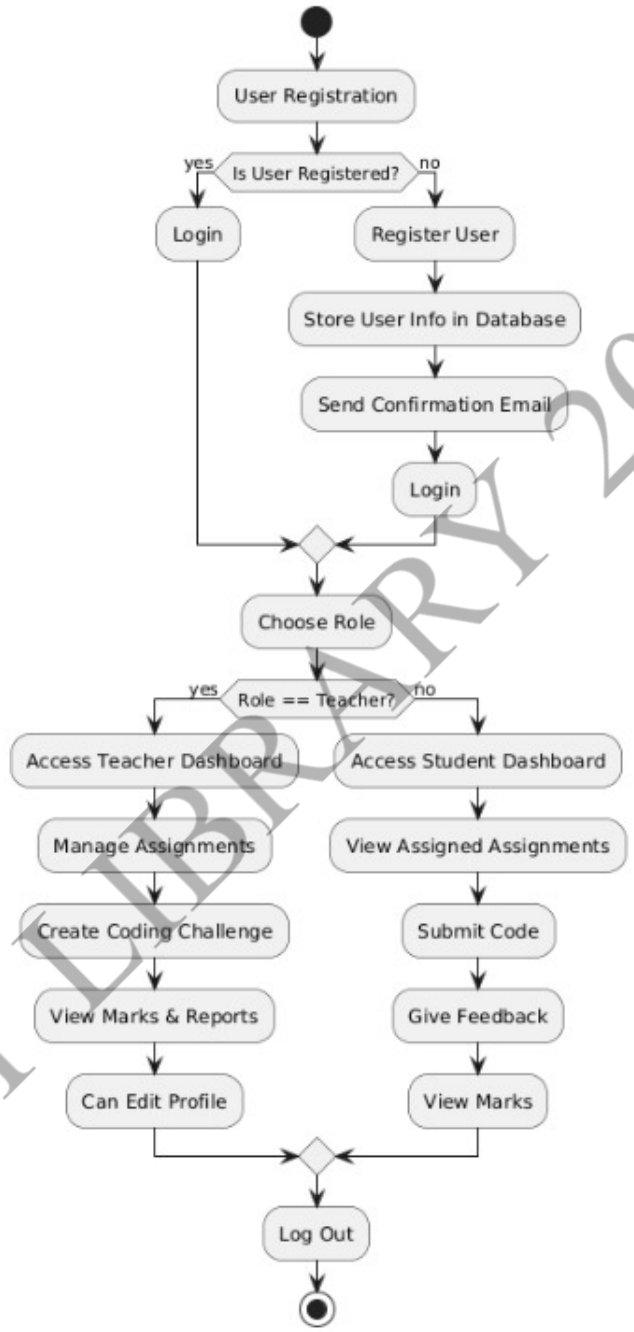
Figure 3.1: Use-Case Diagram for GradeFusion.ai

Use Case Description:

The use case diagram above shows the main functions of the GradeFusion.ai system. Teachers can manage profiles, create coding challenges, view reports, and give feedback. Students can register, log in, submit code, and view feedback. This diagram highlights how the system supports both teachers and students in a streamlined coding evaluation process.

SFIT LIBRARY 2025

3.2 Workflow diagram for Proposed System and Description



SFIT LIBRARY 2025

Figure 3.2: Proposed System: Workflow diagram for GradeFusion.ai

Workflow diagram Description

The diagram illustrates the workflow of an online coding evaluation platform. It starts with user registration, where a new user can either log in or register.

After successful registration and login, users select a role as either a teacher or a student. Teachers can manage assignments, create coding challenges, and view reports, while students can view assigned tasks, submit code, and receive feedback. Both roles have access to profile editing, and the process concludes with logging out of the system.

3.2.1 Phase/Module-wise Explanation (as per Workflow)

- **Registration Module**
  - **Functionality:** Allows new users to sign up.
  - **Steps:**
    - Users enter details: unique email, PID, password, confirm password, and select a role (Student or Teacher).
    - System checks for unique email/PID to prevent duplicates.
    - Valid entries are stored in the database, with the user role noted for access control.
  - **Outcome:** Users are registered in the system with role-based permissions set for further operations.
- **Login and Dashboard Module**
  - **Functionality:** Authenticates users and redirects to their respective dashboards.
  - **Steps:**
    - Users log in using their unique PID or email and password.
    - System validates credentials against stored data.
    - If successful, role-based redirection occurs:
      - Teacher Dashboard for teachers.
      - Student Dashboard for students.
  - **Outcome:** Authenticated users are directed to their appropriate dashboard based on role, enabling role-specific functionalities.

SFIT LIBRARY 2025

- **Teacher Dashboard**
  - **Assignment/Exam Management:**
    - Teachers create assignments or exams by entering:
      - Problem statements.
      - Sample examples for illustration.
      - At least three test cases to validate solutions.
    - All assignment data is stored in the database, accessible to students.
  - **Analytics & Reports:**
    - System generates reports based on student submissions.
    - Reports include details on test case pass rates and scores for each student.
  - **Editing Features:**
    - Teachers can update batches, problem statements, examples, test cases, and sample answers.
    - Changes are logged and stored in the database for record-keeping.
  - **Profile Management:**
    - Teachers can update profile details such as contact information or credentials, with all updates stored in the database.
  - **Outcome:** Teachers have full access to manage, update, and analyze student progress, assignments, and batch configurations.
- **Student Dashboard**
  - **Assignment Access:**
    - Students can view their assigned assignments or exams in real-time.
  - **Code Submission:**
    - A code editor is provided for students to write and submit their code.

SIT LIBRARY 2025

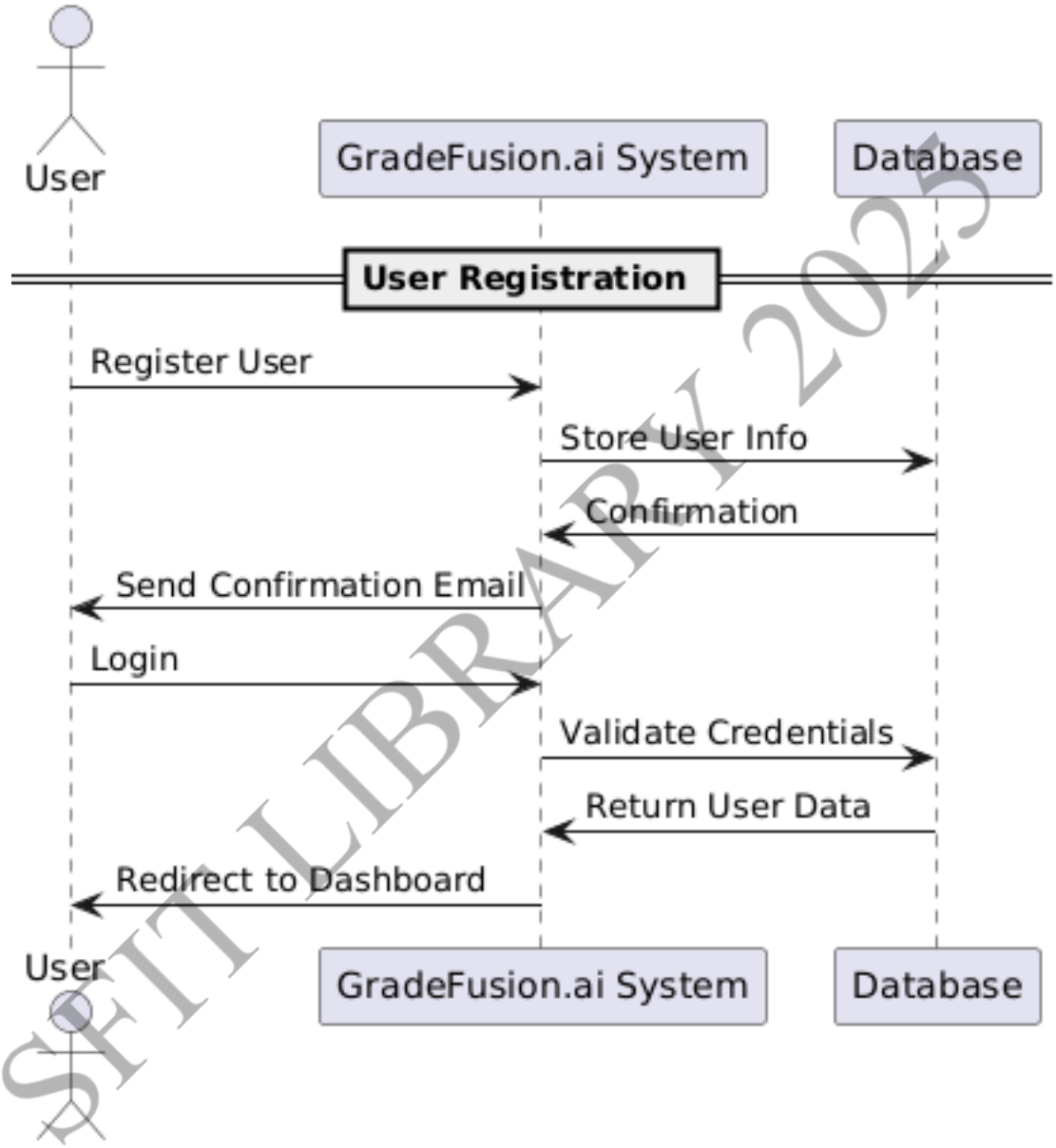
  - **Code Evaluation:**
    - Submitted code is automatically evaluated against the defined test cases.
    - Marks are calculated based on the number of test cases passed.
  - **Code and Marks Storage:**
    - Both the submitted code and the evaluation results are stored.
    - Marks assigned per submission are recorded for future analysis by teachers.
  - **Outcome:** Students can submit assignments, receive instant evaluations, and have their progress recorded for review.

- **Database Integration Module**
  - **Data Storage:**
    - All system activities are logged in the database.
    - Teacher Actions: Batch creation, assignment setup, editing, etc.
    - Student Actions: Code submissions, test case results, and other participation metrics.
  - **Marks Assignment:**
    - System calculates marks based on the number of passed test cases.
    - Marks and code submissions are stored in the database for record-keeping and analysis.
  - **Outcome:** A central database stores all relevant activities, evaluations, and user roles, allowing easy access for reporting, analytics, and performance tracking.

SIT LIBRARY 2025

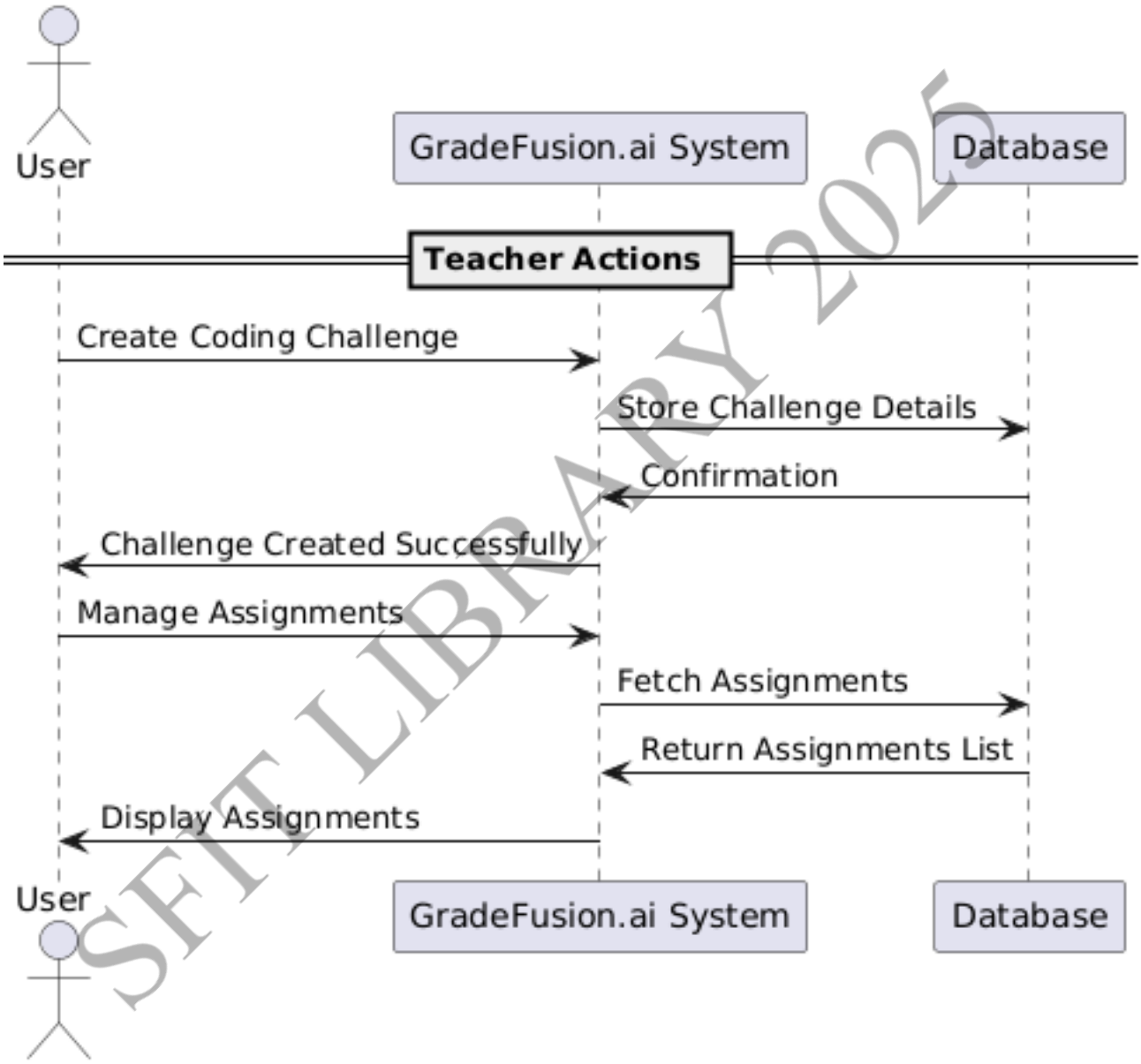
3.3 Analysis and Modeling

3.3.1 Sequence Diagram



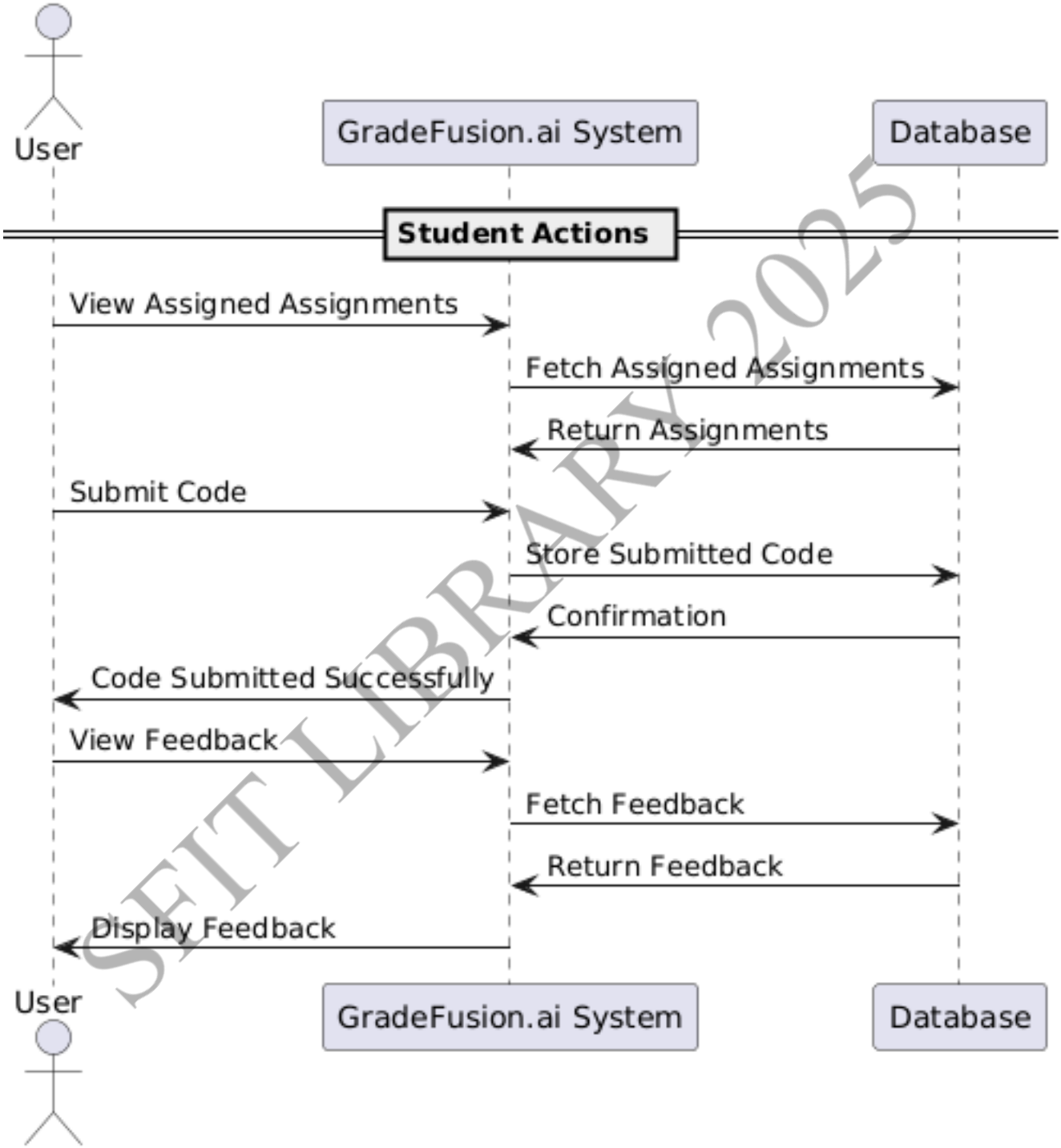
SFIT LIBRARY 2025

Figure 3.3: User Sequence diagram of GradeFusion.ai



SFIT LIBRARY 2025

Figure 3.4: Teacher Sequence diagram of GradeFusion.ai



SFIT LIBRARY 2025

Figure 3.5: Student Sequence diagram of GradeFusion.ai

The sequence diagram for GradeFusion.ai shows interactions between a User, the GradeFusion.ai System, and a Database for user registration, teacher actions, and student actions. In the User Registration section, the user registers by storing their information, receives a confirmation email, logs in with credentials that are validated, and is then redirected to the dashboard. In Teacher Actions, teachers create coding challenges and manage assignments by storing and fetching relevant data from the database, receiving confirmations, and displaying assignment lists. In Student Actions, students view and submit assignments, with their submissions stored and feedback fetched from the database, allowing students to receive assignment feedback effectively.

3.3.2 Activity Diagram

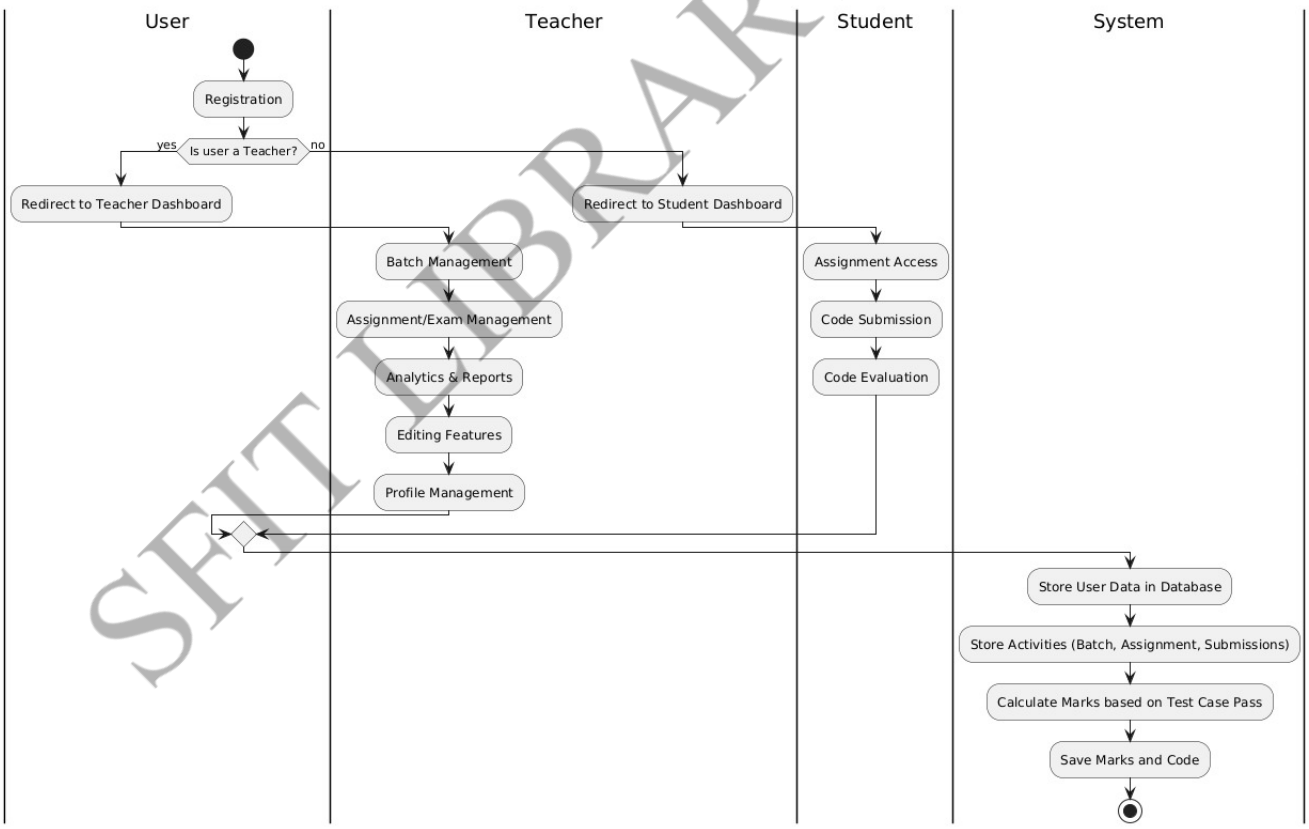


Figure 3.6: Activity diagram of GradeFusion.ai

This diagram illustrates a user flow for a teacher-student system. Users register and are redirected to either a teacher or student dashboard based on their role. Teachers handle batch management, administration, and have access to analyt-

SFIT LIBRARY 2025

ics, editing tools, and profile management. Students access assignments, submit code, and receive evaluations. The system processes data storage, activity logging, and calculation of marks, ultimately saving results and submissions.

3.4 Functional Modeling

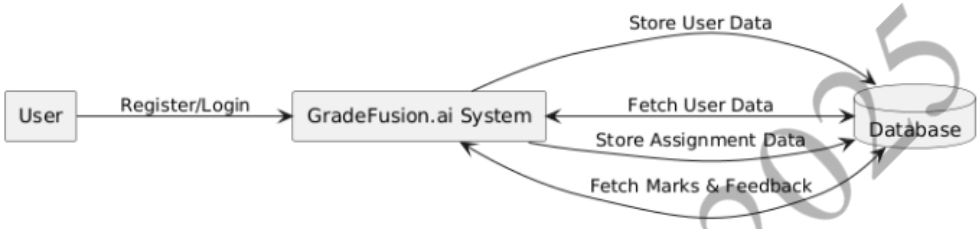


Figure 3.7: DFD Level 0 for GradeFusion.ai

The Level 0 Data Flow Diagram (DFD) for the GradeFusion.ai System shows the basic flow of data between the user, the system, and the database. The user registers or logs in to interact with the system. The GradeFusion.ai System manages actions such as storing and fetching user data, saving assignment details, and retrieving marks and feedback. The database stores all this information and handles requests from the system. This diagram highlights the core interactions and data processes within the system.

SFIT LIBRARY 2025

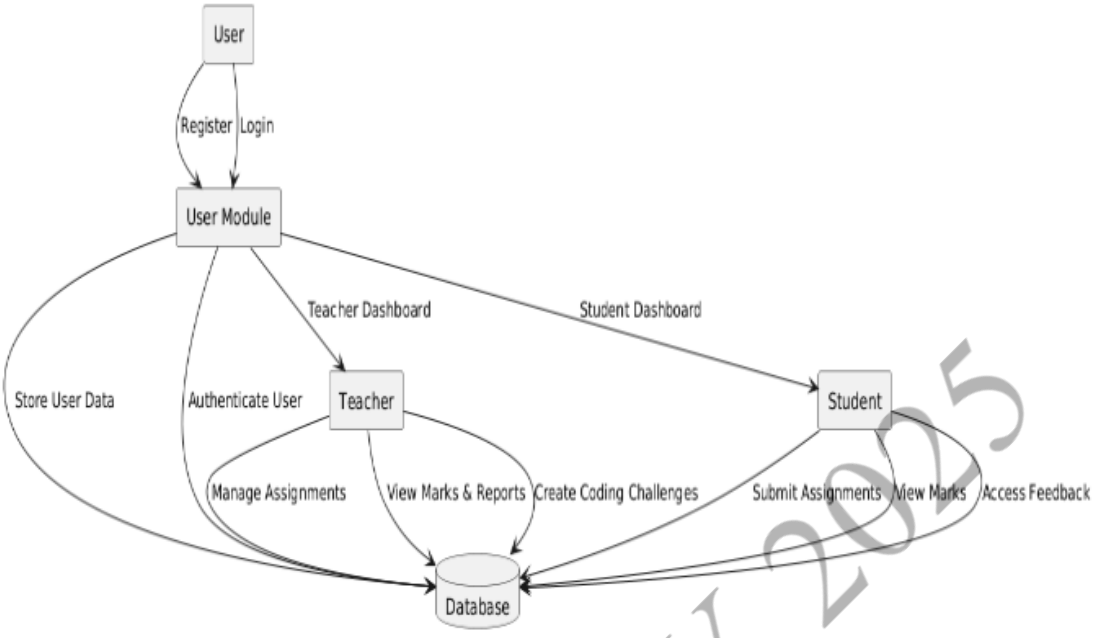


Figure 3.8: DFD Level 1 for GradeFusion.ai

The Level 1 Data Flow Diagram (DFD) for an AI-Powered Academic Research Assistant illustrates the main interactions between the User, Teacher, Student, User Module, and Database. The User, after registering or logging in, is authenticated through the User Module, which also manages and stores user data in the central Database. The Teacher, once authenticated, gains access to their dashboard, where they can manage assignments and view marks and reports. Similarly, the Student can create coding challenges, submit assignments, view marks, and access feedback through the Student Dashboard. All data interactions, including assignment details, marks, reports, and feedback, are facilitated by the Database, which serves as the central repository. The DFD highlights how data flows between entities, enabling the assistant to support academic and research-related tasks for both teachers and students.

SIT LIBRARY 2025

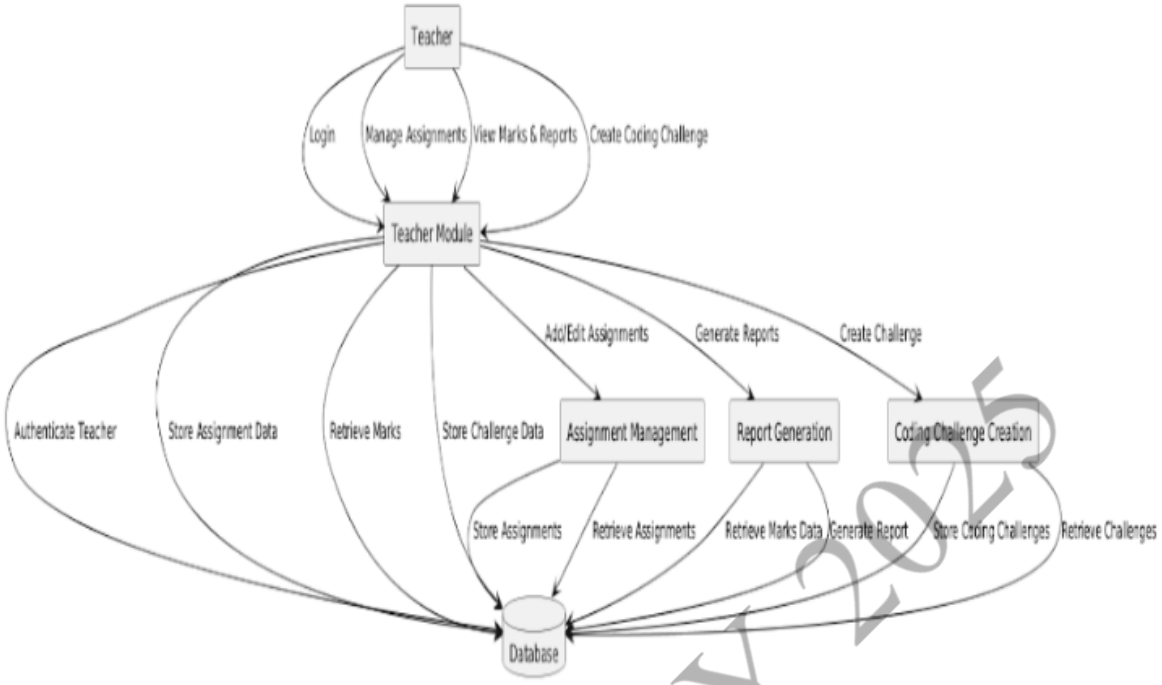


Figure 3.9: DFD Level 2 for GradeFusion.ai

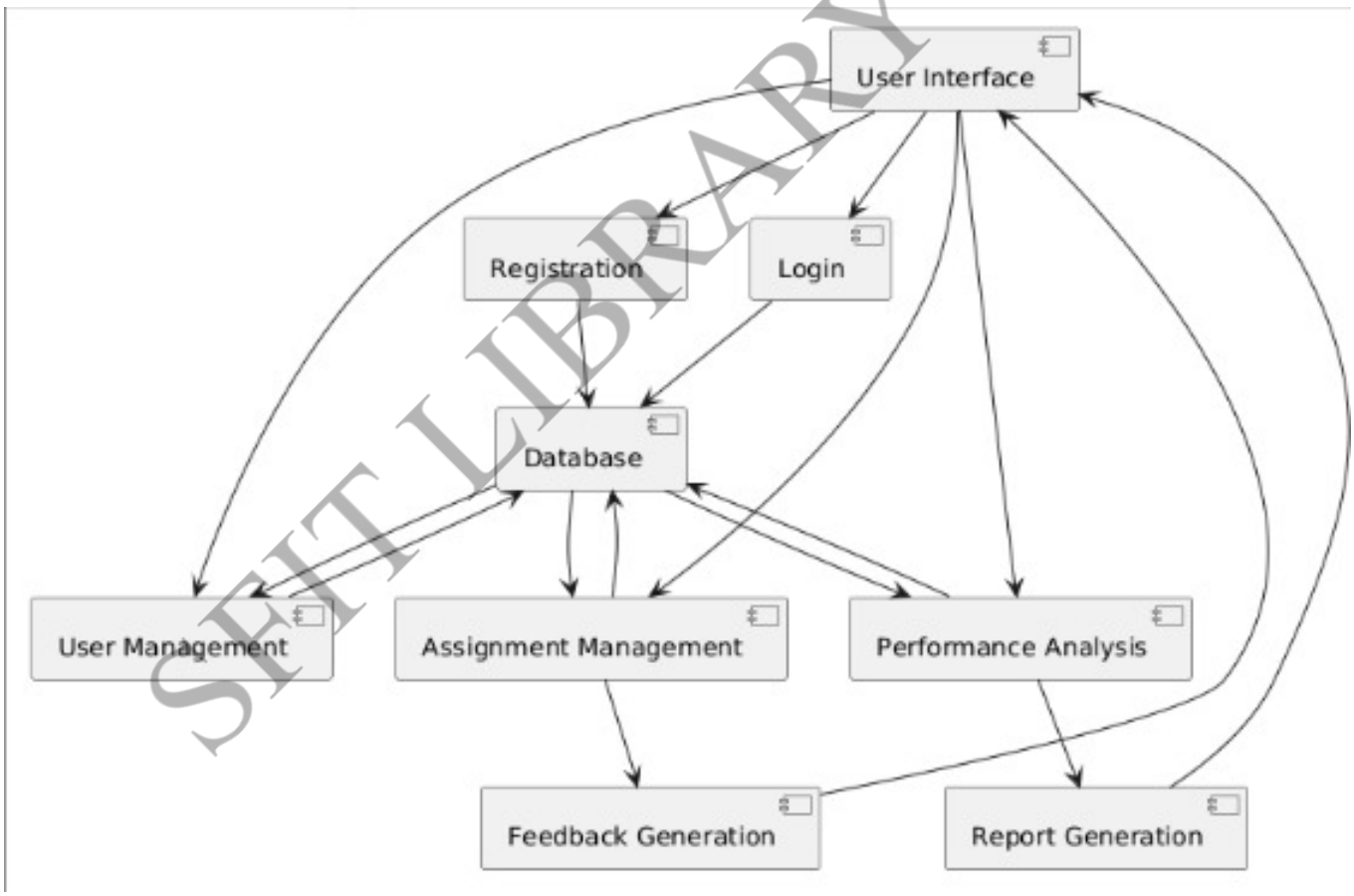
The Level 2 DFD illustrates the comprehensive interactions within the Teacher Module of the AI-Powered Academic Research Assistant system. It breaks down key processes such as authentication, assignment management, report generation, and coding challenge creation. The Teacher can log in, manage assignments, view reports, and create challenges, with data securely stored and retrieved from the database to facilitate efficient academic assistance.

SFIT LIBRARY 2025

Chapter 4

Proposed System: Design

4.1 Architectural view



SHT LIBRARY 2025

Figure 4.1: Architectural Diagram of GradeFusion.ai

This diagram depicts the architecture of a user management and performance system. The user interface serves as the entry point for users, connecting to both the registration and login modules. These modules interact with the database,

which acts as the central hub for data flow. The database supports core functions such as user management, assignment management, and performance analysis. Outputs include feedback generation and report generation, indicating completed data processing and evaluation results.

4.2 Architectural Style

In the Client-Server Architecture for the ”GradeFusion.ai” project, the system is divided into two main components: the Client (frontend) and the Server (backend). This architecture facilitates separation of concerns, enhances scalability, and simplifies the development of interactive web applications like GradeFusion.ai, which involves frequent client-server communication for code submissions, test case evaluations, and result retrieval.

4.2.1 Client-Side (Frontend)

Role: The client side of the application is responsible for presenting the user interface and handling interactions with users. Students and teachers access GradeFusion.ai through a web browser, which acts as the client.

Components:

- **User Interface (UI):** Built using React (or similar frameworks like Vue or Angular), which provides a responsive and dynamic experience. It includes pages for login, dashboard, assignment lists, and code editor interfaces.
- **Code Editor:** A dedicated area where students can write, edit, and submit code for assignments or exams.
- **HTTP Requests:** The client sends HTTP requests to the server for operations such as submitting code, retrieving assignments, and viewing results.

Responsibilities:

- Display the assignments and exam details to students.

SIT LIBRARY 2025

- Capture inputs, such as code submissions, and send them to the server.
- Receive test case results and feedback from the server and display them to the student.
- For teachers, the client-side provides a way to create and manage assignments, view student submissions, and analyze performance.

4.2.2 Server-Side (Backend)

Role: The server is the core of the system where the application logic, data storage, and processing are managed. It receives requests from the client, processes them, and sends back the responses.

Components:

- **API Layer:** Built with Node.js (Express) or other backend technologies that handle HTTP requests and route them to the appropriate services.
- **Database:** A Firebase or SQL database stores all user data, submissions, assignments, test cases, and grades. It facilitates CRUD (Create, Read, Update, Delete) operations on student and teacher data.
- **Code Evaluation Engine:** A crucial part of the backend that executes submitted code, runs it against predefined test cases, and returns results. It ensures a secure sandbox environment to prevent malicious code execution.
- **Real-Time Analytics and Reporting Module:** Generates performance reports based on the test cases passed, enabling teachers to analyze and improve student performance.

Responsibilities:

- Authentication and authorization of users (students and teachers).
- Processing code submissions from students, executing them, and returning test results.

SFIT LIBRARY 2025

- Managing CRUD operations for assignments, exams, and profiles.
- Logging data for analytics to provide insights into student performance and areas for improvement.
- Providing data storage and retrieval for efficient system performance.

4.2.3 Workflow

- **Request-Response Interaction:** The client (web application) sends HTTP requests (e.g., submit code, fetch assignments) to the server.
- **Processing on the Server:** The server processes each request by interacting with the database, performing operations (e.g., running code), and preparing the response.
- **Sending Responses to Client:** The server sends the results back to the client, which are then displayed in real-time to students and teachers.

4.2.4 Advantages of Client-Server Architecture for GradeFusion.ai

- **Modularity and Separation of Concerns:** This architecture allows the frontend and backend to be developed and scaled independently.
- **Scalability:** Each component can be scaled based on load. For example, the backend's code evaluation engine can be scaled separately to handle heavy usage.
- **Security:** Sensitive operations, such as code execution, can be managed securely on the server side.
- **Efficiency:** The server handles complex processing (like code evaluation), while the client handles presentation, leading to a responsive user experience.

SFIT LIBRARY 2025

4.3 Algorithms

This section discusses the algorithms used to efficiently validate student submissions in online coding evaluation platforms. These methods ensure fast and accurate verification of outputs against expected results, contributing to reliable and scalable assessment.

4.3.1 Integration of Algorithms in Student Output Validation

These algorithms can be effectively integrated into the validation process for student submissions in an online coding platform. Their applications are as follows:

- **Student Output Validation:** Both algorithms can be applied in the validation process where student outputs are compared with expected results.
  - For small outputs, KMP ensures efficient exact matching.
  - For large outputs, Rabin-Karp optimizes the matching process through rolling hashes.
- **Handling Large Data:** For expected outputs that are large (e.g., hundreds or thousands of lines), Rabin-Karp enables faster matching by comparing hashes rather than characters.
- **Multiple Test Cases:** Rabin-Karp is particularly effective when evaluating multiple test cases with different outputs simultaneously, allowing multiple string matches in a single pass.

These algorithms enhance the efficiency of the comparison process, ensuring prompt evaluation of student submissions, even for large and complex outputs.

4.3.2 Algorithm for Marks Calculation Based on Test Results

Algorithm for Result Calculation based on Judge0 Submissions

```
function calculateResult(judge0RawResponse) {
    try {
```

SIT LIBRARY 2025

```
3           if (!judge0RawResponse || !judge0RawResponse.submissions) {
4               throw new Error('Missing submissions');
5           }
6
7           const submissions = judge0RawResponse.submissions;
8           const totalTests = submissions.length;
9           const passedTests = submissions.filter(sub => sub.status.id
                === 3);
10
11          // Scenario 1: Simple 2 marks per pass
12          const scenario1 = (() => {
13              const totalPossibleMarks = totalTests * 2;
14              const obtainedMarks = passedTests.length * 2;
15              return ((obtainedMarks / totalPossibleMarks) * 10).
                    toFixed(2);
16          })();
17
18          // Scenario 2: Weighted based on difficulty
19          const scenario2 = (() => {
20              const easyCount = Math.floor(totalTests * 0.3);
21              const mediumCount = Math.floor(totalTests * 0.4);
22              const hardCount = totalTests - easyCount - mediumCount;
23
24              const marksArray = [];
25              for (let i = 0; i < totalTests; i++) {
26                  if (i < easyCount) marksArray.push(1);
27                  else if (i < easyCount + mediumCount) marksArray.push
                        (1.5);
28                  else marksArray.push(2.5);
29              }
30
31              let obtainedMarks = 0;
32              let totalPossibleMarks = 0;
33              for (let i = 0; i < totalTests; i++) {
34                  totalPossibleMarks += marksArray[i];
35                  if (submissions[i].status.id === 3) {
36                      obtainedMarks += marksArray[i];
37                  }
38              }
39
40              return ((obtainedMarks / totalPossibleMarks) * 10).
                    toFixed(2);
41          })();
42
43          // Scenario 3: Weighted + Time/Memory penalties
44          const scenario3 = (() => {
45              const easyCount = Math.floor(totalTests * 0.3);
```

SFIT LIBRARY 2025

```
46              const mediumCount = Math.floor(totalTests * 0.4);
47              const hardCount = totalTests - easyCount - mediumCount;
48
49              const marksArray = [];
50              for (let i = 0; i < totalTests; i++) {
51                  if (i < easyCount) marksArray.push(1);
52                  else if (i < easyCount + mediumCount) marksArray.push
                        (1.5);
53                  else marksArray.push(2.5);
54              }
55
56              const timeThreshold = 0.08; // seconds
57              const memoryThreshold = 20480; // KB
58
59              let obtainedMarks = 0;
60              let totalPossibleMarks = 0;
61              for (let i = 0; i < totalTests; i++) {
62                  totalPossibleMarks += marksArray[i];
63
64                  if (submissions[i].status.id === 3) {
65                      let marks = marksArray[i];
66
67                      if (submissions[i].time > timeThreshold) {
68                          marks -= 0.2;
69                      }
70                      if (submissions[i].memory > memoryThreshold) {
71                          marks -= 0.2;
72                      }
73
74                      if (marks < 0) marks = 0;
75                      obtainedMarks += marks;
76                  }
77              }
78
79              return ((obtainedMarks / totalPossibleMarks) * 10).
                    toFixed(2);
80          })();
81
82          return {
83              scenario1Marks: scenario1,
84              scenario2Marks: scenario2,
85              scenario3Marks: scenario3
86          };
87
88      } catch (error) {
89          console.error("Error in calculateResult:", error);
90          throw error;
```

SFIT LIBRARY 2025

```
91         }
92 }
```

Function Description: calculateResult(judge0RawResponse)

The calculateResult function is responsible for calculating a student's performance across multiple evaluation scenarios based on automated code execution results provided by the Judge0 API. It processes a structured input object containing an array of test case submissions and outputs normalized scores under three distinct grading methodologies:

- **Scenario 1 – Uniform Marking Scheme:**
  Each successfully passed test case is assigned a fixed mark (2 marks per test). The final score is calculated as a normalized score out of 10.
- **Scenario 2 – Weighted Marking Based on Difficulty:**
  Test cases are categorized into three difficulty levels: Easy (30%), Medium (40%), and Hard (remaining 30%). Each level carries a different weight (Easy: 1 mark, Medium: 1.5 marks, Hard: 2.5 marks). The candidate's performance is evaluated based on the weighted sum of the passed test cases and normalized to a score out of 10.
- **Scenario 3 – Weighted Marking with Resource Penalties:**
  Building upon Scenario 2, additional penalties are imposed for excessive resource usage. If the execution time exceeds 0.08 seconds or the memory usage exceeds 20 MB (20480 KB), a deduction of 0.2 marks per threshold breach is applied. This approach incentivizes not only correctness but also optimal code performance and efficiency. The final adjusted score is again normalized to a scale of 10.

4.3.3 LLM's

Gemini 2.5 Pro (EXP) demonstrates exceptional performance across key benchmarks, including Reasoning, Science, and Mathematics. It outperforms models like GPT-4 and Claude 3, showcasing advanced reasoning, deep scientific knowledge, and strong mathematical problem-solving skills.

SIT LIBRARY 2025

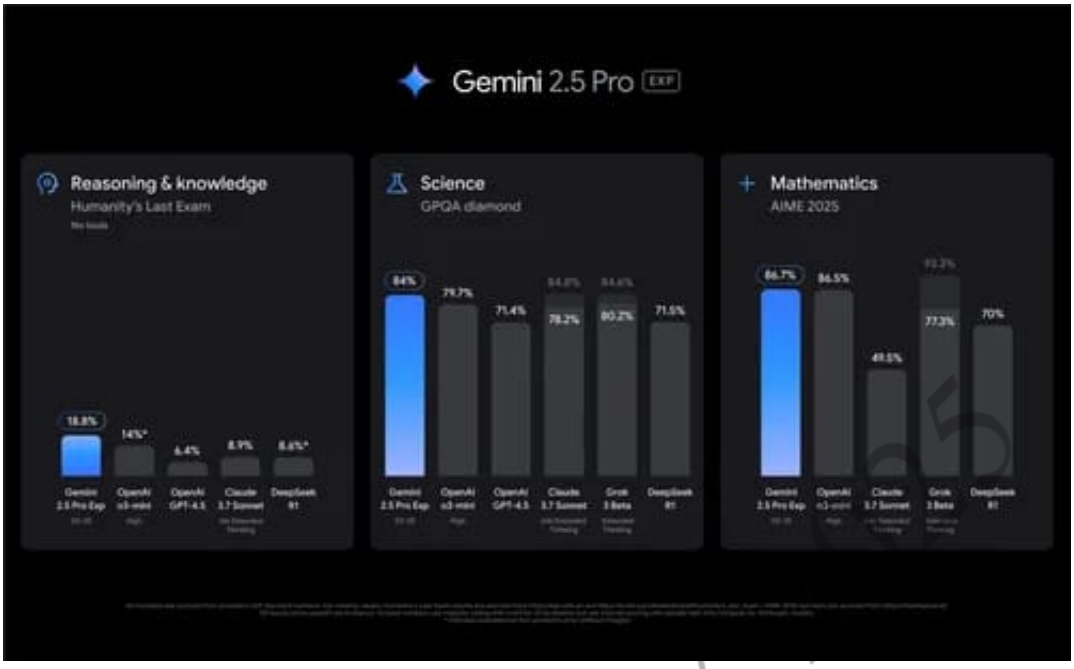


Figure 4.2: Gemini 2.5 Pro Performance Across Benchmarks

As shown in Figure 4.2, Gemini 2.5 Pro (EXP) leads in multiple challenging domains, making it a strong choice for applications requiring high levels of accuracy, understanding, and problem-solving capabilities.

SFIT LIBRARY

Chapter 5

Proposed System : Implementation Plan and Experimental Set up

SFIT LIBRARY 2025

5.1 Experimental Setup

5.1.1 Details of Database or Input to Systems

Database: Firebase Firestore is used as the primary database for storing user details, batch information, assignments, student submissions, and performance records.

- **Structure:**
  - **Users Collection:** Stores user details, including unique email, PID, password (hashed), and role (Student or Teacher).
  - **Batches Collection:** Contains details about student batches created by teachers, including batch IDs and associated student PIDs.
  - **Assignments Collection:** Stores assignment details like problem statements, sample examples, and test cases for code evaluation.
  - **Submissions Collection:** Records students' code submissions, marks, and test case results.
- **Input Data:**
  - **User Input:** Registration details, assignment problem statements, sample examples, test cases, and code submissions.

  - **System Input:** Automatic validation inputs for code evaluation, including pre-defined test cases and sample outputs.

5.1.2 Performance Evaluation Parameters (for Validation)

- **Code Evaluation Metrics:**
  - **Test Case Pass Rate:** Percentage of test cases passed by the student's code.
  - **Accuracy:** Measures how accurately the code matches the expected output.
  - **Execution Time:** Time taken by the code to run and produce results.
  - **Code Efficiency:** Evaluates the complexity and performance of the submitted code.
- **User Activity Metrics:**
  - **Submission Success Rate:** Number of successful submissions versus total attempts.
  - **Teacher Engagement:** Number of batches and assignments created.
  - **Student Engagement:** Frequency of assignment submissions and time spent in the code editor.
- **Scalability and Load Handling:**
  - **Concurrent Users:** Performance of the system under simultaneous usage.
  - **Database Response Time:** Time taken to fetch and store data under load conditions.

5.1.3 Special Requirements

- **Cloud Infrastructure:** Firebase for real-time database and authentication.
- **Hardware Requirements:**

SIT LIBRARY 2025

  - **Development Machine:** Minimum of 8 GB RAM, 1.6 GHz or faster processor.
  - **Deployment Server:** Capable of handling multiple concurrent requests, with appropriate load balancing.

- **External Resources:** Use of Firebase and GitHub for version control and collaboration.
- **Software Dependencies:**
  - **Node.js and Express.js:** For backend services.
  - **React with TypeScript:** For frontend development.
  - **Visual Studio Code:** For code writing and debugging.

5.2 Implementation Plan of this Semester

5.2.1 Phase 1: January - Week 1 to Week 3 (Initial Setup and Core Implementation)

Goals: Establish the core architecture and implement foundational features.
Tasks:

- **System Architecture Setup:**
  - Finalize the database structure in Firebase (users, batches, assignments, submissions).
  - Implement basic RESTful API endpoints for handling user registration, login, and authentication.
- **Account Logins and User Interfaces:**
  - Develop user login, registration, and authentication interfaces (login, registration forms, session management).
  - Implement password hashing and security features for safe authentication.

SFIT LIBRARY 2025

- **Compiler Integration:**
  - Set up the environment for compiling and running student code (for languages like C, C++, and Java).
  - Integrate an API for code compilation (e.g., using Docker containers for code execution in a controlled environment).
- **Basic Testing:**
  - Test account registration, login, and session management.
  - Test the code compilation process for correctness and security.

5.2.2 Phase 2: January - Week 4 to February - Week 2 (Test Case Management and Initial AI Model Integration)

Goals: Implement assignment management and begin integrating AI for test case generation.

Tasks:

- **Assignment Management:**
  - Develop teacher interfaces for creating, editing, and managing coding assignments (problem statements, examples, test cases).
  - Store and retrieve assignments and their associated test cases from the Firebase database.
- **AI Model for Test Case Generation:**
  - Research and integrate a basic Large Language Model (LLM) for automatic test case generation from problem statements (using OpenAI or other NLP models).
  - Design an algorithm to generate a set of test cases for a given coding problem using LLM.
- **Compiling and Running Student Code:**

SFIT LIBRARY 2025

  - Implement functionality to compile and run student-submitted code against generated test cases.
  - Store the results of each student submission and evaluate based on the number of passed test cases.

- **Performance Metrics:**
  - Begin collecting performance metrics (execution time, test case pass rate, code efficiency) for student submissions.

5.2.3 Phase 3: February - Week 3 to Week 5 (Advanced AI Integration and Code Evaluation)

Goals: Enhance AI capabilities for better test case generation and code evaluation.

Tasks:

- **Advanced Test Case Generation with LLM:**
  - Train or fine-tune the AI model for improved test case generation, making it more specific and varied based on the problem type.
  - Implement natural language processing techniques to handle edge cases and ensure valid test cases.
- **AI Integration for Code Evaluation:**
  - Integrate AI models that can analyze student code for potential bugs, inefficiencies, and logical errors.
  - Provide insights or suggestions for improvement along with the test results.
- **Enhanced Code Evaluation Metrics:**
  - Add additional evaluation criteria such as code complexity, efficiency, and adherence to coding best practices.
  - Implement performance metrics to monitor system behavior under load (e.g., response time, system throughput).

SIT LIBRARY 2025

5.2.4 Phase 4: March - Week 1 to Week 2 (Testing and Refining the System)

Goals: Perform extensive system testing and refine performance.
Tasks:

- **Testing with Sample Test Cases:**
  - Conduct extensive testing using sample coding assignments and test cases to validate the code evaluation system.
  - Compare the AI-generated test cases with manually created ones for accuracy and reliability.
- **Performance Metrics Collection:**
  - Evaluate the system's scalability by testing with multiple concurrent users and large test cases.
  - Analyze database response times, code execution times, and system load handling.
- **UI/UX Enhancements:**
  - Improve user interfaces based on feedback, ensuring ease of use for both students and teachers.
  - Test the usability of the teacher's dashboard for managing assignments, viewing analytics, and interacting with student submissions.

5.2.5 Phase 5: March - Week 3 to Week 4 (Final Testing, Documentation, and Deployment)

Goals: Finalize the system and prepare it for deployment.
Tasks:

- **System Debugging and Performance Optimization:**
  - Fix any bugs identified during the testing phases.

SFIT LIBRARY 2025

  - Optimize the performance of the platform, particularly for code evaluation and database interactions.

- **Documentation:**
  - Document the entire system, including the database schema, API endpoints, and AI model usage.
  - Prepare the user manual for both students and teachers, including how to use the platform and its features.

- **Deployment:**
  - Deploy the system on a cloud infrastructure, ensuring scalability and availability.
  - Conduct final testing in the live environment with real users (students and teachers).

- **Future Enhancements:**
  - Plan for future improvements, such as adding new programming languages, integrating plagiarism detection, and refining AI-based suggestions.

SFIT LIBRARY 2025

Chapter 6

Result and Discussions

6.0.1 Implementation Results

1) Home Page

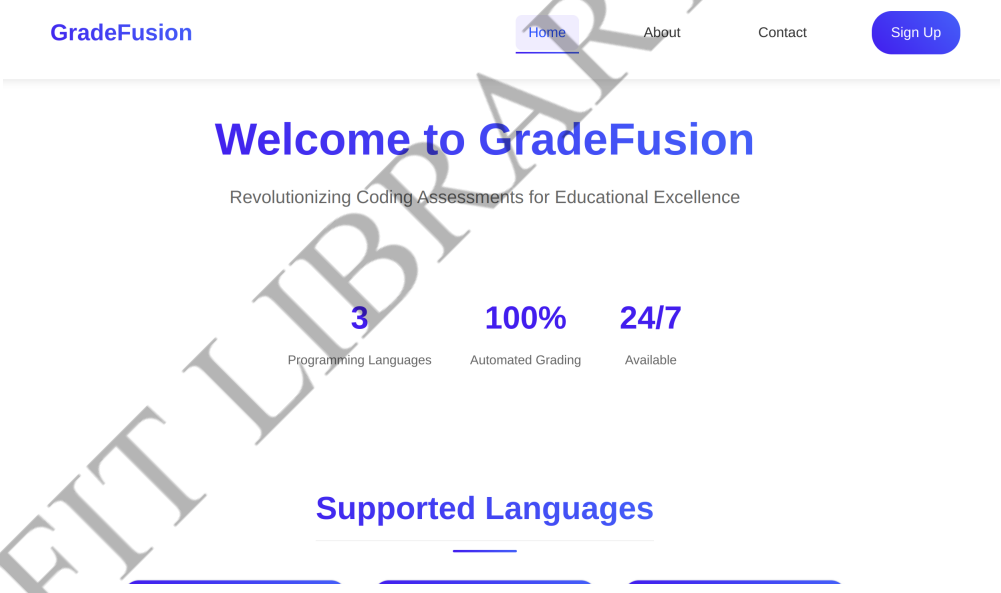


Figure 6.1: Home Page Interface of GradeFusion.ai

This image shows the Home Page of GradeFusion, showcasing its sleek, user-friendly interface. The page highlights several key features of the platform, including the ability to access coding challenges, real-time performance analytics, and AI-powered assistance for test case generation. The platform also offers intuitive navigation for students and instructors, providing an easy path to interact with coding problems, track progress, and review results. Additionally, there is a prominent option to SignUp, allowing new users to create an account

SFIT LIBRARY 2025

and get started on the platform, whether they are students looking to sharpen their coding skills or instructors preparing assignments for their classes.

2) Signup Page

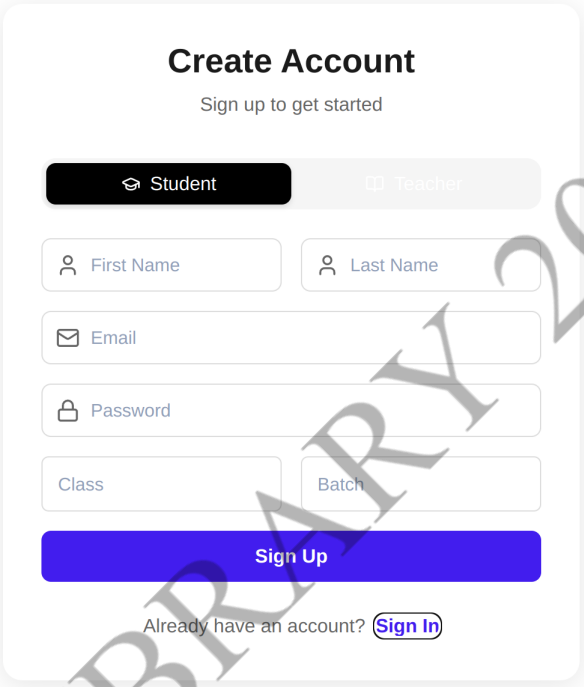


Figure 6.2: Signup Page Interface of GradeFusion.ai

This page is the Signup screen of GradeFusion, where users can create an account as either a teacher or a student, each with customized features. The platform uses JWT (JSON Web Tokens) for secure authentication. After signup, a JWT token is generated, enabling secure login and access to personalized dashboards. JWT ensures encrypted, stateless sessions, enhancing security, scalability, and user experience while keeping data safe and the system efficient.

SFIT LIBRARY 2025

3) Student Dashboard

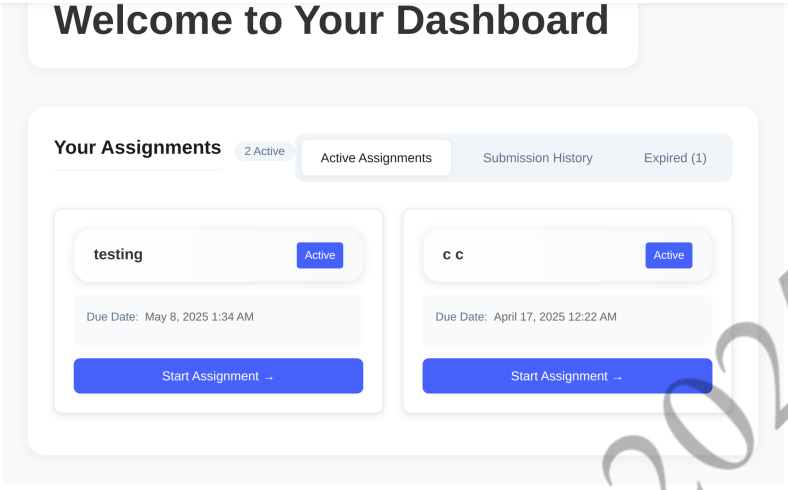


Figure 6.3: Student Dashboard Interface of GradeFusion.ai

The Student Dashboard displays all active assignments with clear due dates. Students can easily start assignments, track upcoming deadlines, and monitor their progress. Tabs like Active Assignments, Submission History, and Expired help students stay organized and manage their coursework efficiently.

4) Teacher Dashboard

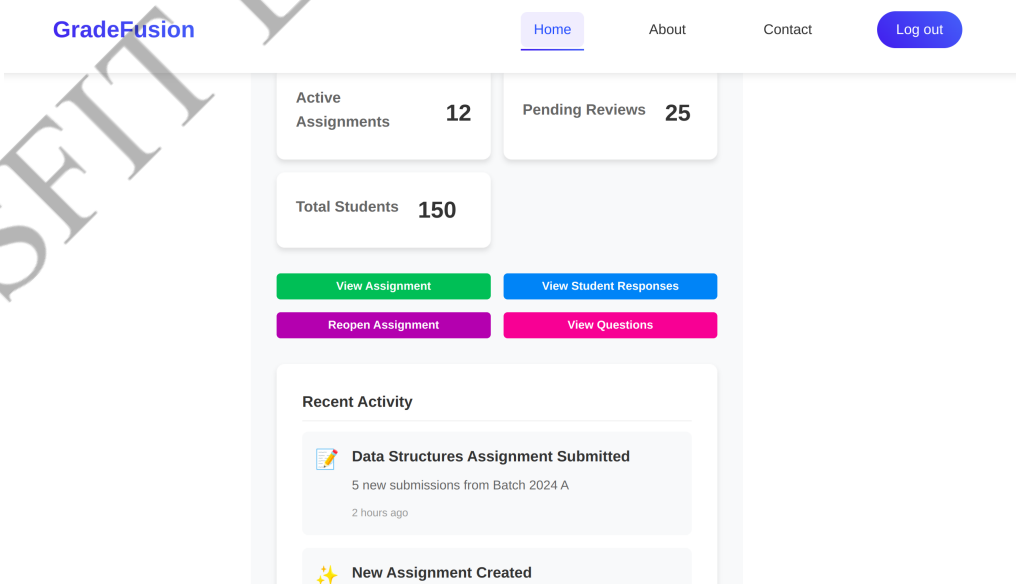


Figure 6.4: Teacher Dashboard Interface of GradeFusion.ai

SFIT LIBRARY 2025

The Teacher Dashboard provides a quick overview of active assignments, pending reviews, and total students. It includes action buttons like View Assignment, View Student Responses, Reopen Assignment, and View Questions. A Recent Activity section displays updates such as new submissions and assignment creation, helping teachers manage and monitor student activities easily.

5) Coding Environment

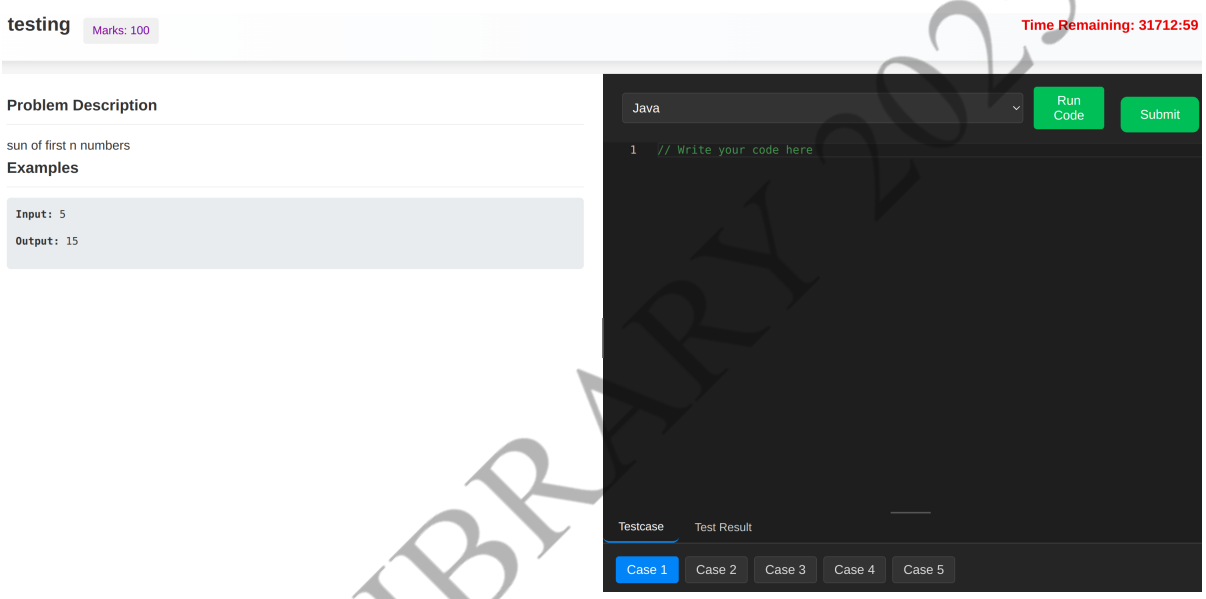


Figure 6.5: Coding Environment Interface of GradeFusion.ai

This image shows the coding platform where students can view the problem statement on the left and write their code on the right in the Monaco code editor, which offers syntax highlighting and autocompletion for a smooth coding experience. Students can choose from Java, C++, or Python for their solution. The code is compiled and executed using Judge0, a popular open-source API that supports multiple programming languages. The platform also provides functionalities to run code against test cases and check results. Additionally, the test timer does not start until the student begins coding, ensuring they have adequate time to focus on the problem.

SIT LIBRARY 2025

6) Failed Testcases

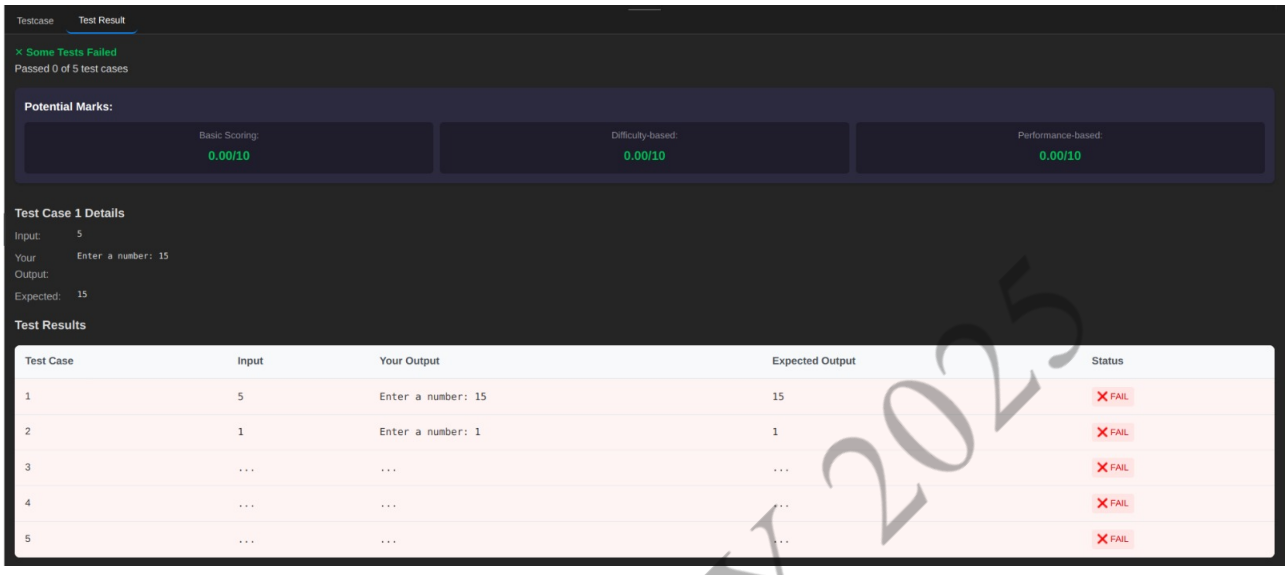


Figure 6.6: Coding Environment Interface of GradeFusion.ai

The image shows a test result summary where all 5 test cases have failed. The user's output includes extra prompts like "Enter a number: X", which doesn't match the expected plain numerical output, leading to 0/10 marks across basic, difficulty, and performance scoring.

7) Passed Testcases

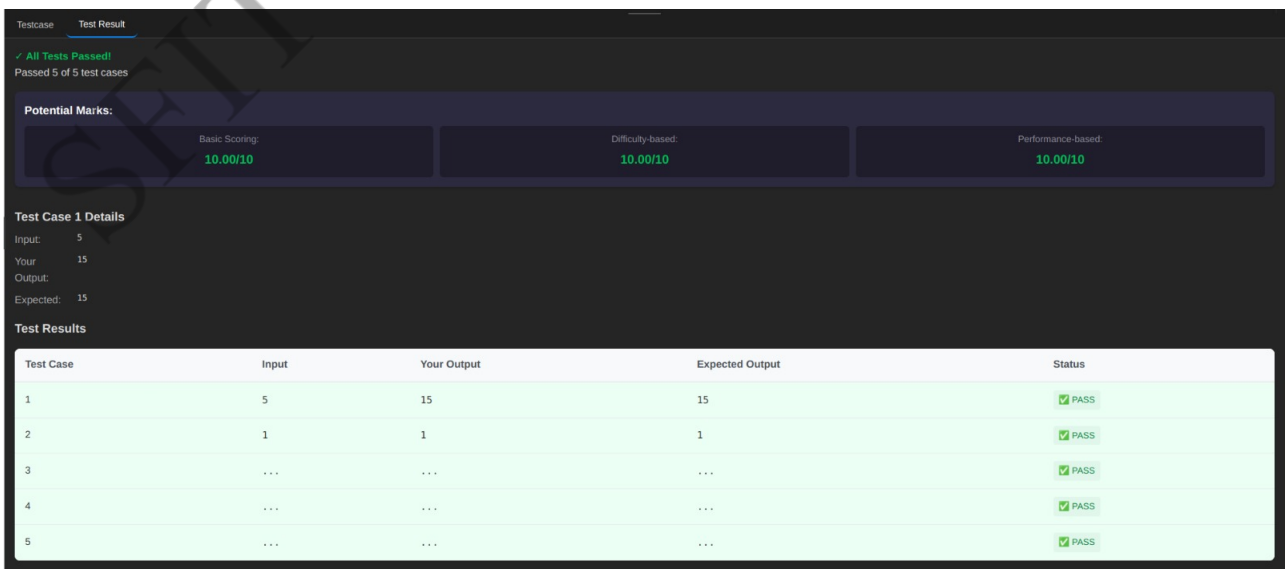


Figure 6.7: Coding Environment Interface of GradeFusion.ai

SET LIBRARY 2025

The image shows a test result summary where all 5 test cases have passed successfully. The user's output matches the expected output exactly for all inputs, achieving full marks — 10/10 in Basic, Difficulty-based, and Performance-based scoring.

6.0.2 Comparative study with the existing systems

Feature	**Web-CAT**	**AUTOGRADER**	**GradeFusion**
Grading Mechanism	Automated test cases	Formal semantics	AI-powered feedback
Feedback	Instant but unclear	Precise but complex	Instant and clear
Customization	Predefined test cases	Reference implementation-based	Personalized feedback and test generation
Target Audience	Students	Students	Students & Teachers
Security & Authentication	Basic authentication	Standard methods	JWT tokens for secure authentication

Table 6.1: Comparative Study of Automated Grading Systems

Web-CAT:

1. Grading is automated through predefined test cases.
2. Provides instant feedback, but lacks detailed explanations.
3. Customization is limited to predefined test cases, offering less flexibility.
4. Primarily targeted at students for automated programming assignment grading.

AUTOGRADER:

1. Uses formal semantics and symbolic execution to compare submissions with reference implementations.
2. Feedback is more precise but can be complex and difficult for students to interpret.
3. Customization is based on reference implementation, limiting flexibility for diverse problems.
4. Primarily targeted at advanced students needing more detailed and accurate feedback.

SFIT LIBRARY 2025

GradeFusion:

1. AI-powered feedback with personalized insights based on student submissions.
2. Instant and clear feedback with contextual suggestions for improvements.
3. Allows for customized feedback and test case generation tailored to each student's needs.
4. Designed for both students and teachers, providing an interactive and engaging learning environment.

SFIT LIBRARY 2025

Chapter 7

Conclusions

7.1 Conclusion

In conclusion, this project develops an online coding evaluation platform for academic use, supporting assessments in C, C++, and Java. Teachers can upload coding challenges with example solutions and test cases, benefiting from an automated grading system that ensures consistent and fair evaluations, significantly reducing manual effort. The platform offers real-time analytics to help teachers identify student difficulties and adjust teaching strategies. Built with a user-friendly interface and secure data management, it enhances learning and optimizes the teaching workflow.

7.2 Future Scope

In the future, the platform can be improved by adding more programming languages like Python and JavaScript. Features like simple analytics to track student progress and a tool to check for copied code can make it more effective. Making a mobile app version would let students and teachers use it easily from their phones, and connecting it with other systems used in schools could simplify assignment management. Adding options for group coding, simple grading rules, fun challenges, and rewards could make learning more engaging. The platform could also be made to work offline in places with poor internet, and smarter tools could give helpful feedback to both students and teachers.

SIT LIBRARY 2025

References

[1] Aldriye, Hussam, Alkhalaf, Asma, & Alkhalaf, Muath. (2019). "Automated Grading Systems for Programming Assignments: A Literature Review." *International Journal of Advanced Computer Science and Applications*, 10, 10.14569/IJACSA.2019.0100328.

[2] Hsueh, Nien-Lin, Lien-Chi Lai, & Wei-Hsiang Tseng. (2023). "Design of an Online Programming Platform and a Study on Learners' Testing Ability." *Electronics*, 12(22), 4596.

[3] Pirzado, F. A., Ahmed, A., Mendoza-Urdiales, R. A., & Terashima-Marin, H.(2021). "Navigating the Pitfalls: Analyzing the Behavior of LLMs as a Coding Assistant for Computer Science Students—A Systematic Review of the Literature."

[4] Ihantola, P., Ahadi, A., Karavirta, V., & Korhonen, A. (2010). "Review of Recent Systems for Automatic Assessment of Programming Assignments." *ACM Transactions on Computing Education (TOCE)*, 10(3), 1–29.

[5] Keuning, H., Jeuring, J., & Heeren, B. (2018). "A Systematic Literature Review of Automated Feedback Generation for Programming Exercises." *ACM Transactions on Computing Education (TOCE)*, 19(1), 1–43.

SHT LIBRARY 2025

Acknowledgements

We are thankful to a number of individuals who have contributed to our final year project and without their help; it would not have been possible. First of all, we would like to express our gratitude to Dr. Bidisha Roy our project guide, for their prompt suggestions, guidance, and encouragement over the course of our entire project.

We sincerely appreciate all our project coordinators, for their tremendous assistance to our project. We also appreciate the support of the faculty in our department.

We express our sincere gratitude to our respected Director Bro. Shantilal Kujur, our Principal Dr. Sincy George, and our Head of Department (CMPN) Dr. Kavita Sonawane for providing the facilities, encouragement as well as a conducive environment for learning.

Lastly, we would like to thank our parents and friends who played a crucial role in keeping us motivated throughout with their constant nurture and support.

Sincerely,

Rohit Tarke

Aniket Patil

Tanmay Mohod

Aryan Pange

SFIT LIBRARY 2025