# Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask

A PROJECT REPORT

*Submitted by*

**K.KALYAN SARAN [RA2211003011444]**

**G.VENKAT REDDY [RA2211003011463]**

*Under the Guidance of*

Dr R Madhura

Assistant Professor Department of Computing Technologies

*in partial fulfillment of the requirements for the degree of*

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING


DEPARTMENT OF COMPUTING
TECHNOLOGIES COLLEGE OF ENGINEERING
AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY

KATTANKULATHUR- 603 203

APRIL 2025

Department of Computational Intelligence
**SRM Institute of Science & Technology**
**Own Work* Declaration Form**

To be completed by the student for all assessments

**Degree/ Course** **:B.Tech/CSE**

**Student Name** **:K. Kalyan Saran,G.Venkat Reddy**

**Registration Number** **:RA2211003011444,RA2211003011463**

**Title of Work** : **Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask**

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

| **DECLARATION:** |
| --- |
| I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above. |
| RA2211003011444 RA2211003011463 |

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

# KATTANKULATHUR – 603203

## BONAFIDE CERTIFICATE

Certified that 21CSP302L - Project report titled "**Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask**" is the bonafide work of "**K Kalyan Saran [RA2211003011444], G Venkat Reddy [RA2211003011463]"** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

**SIGNATURE**

**Dr.R. Madhura**
Assistant Professor
Department of Computing Technologies
SRM Institute of Science and Technology
Kattankulathur

**SIGNATURE**

**Dr.G.Niranjana**
Professor of Head
Department of Computing Technologies
SRM Institute of Science and Technology
Kattankulathur

**Examiner 1**

**Examiner 2**

# ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. Leenus Jesu Martin M,** Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to, **Dr. M. Pushpalatha**, Professor and Associate Chairperson - CS, School of Computing and **Dr. Lakshmi,** Professor and Associate Chairperson
-AI, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr. G. Niranjana** , Professor& Department of Computing Technologies , SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, Panel Head, and Panel Members Department of Computational Intelligence, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. S Shanmugam**, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr.R Madhura**, Department of Computing Technologie, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under his / her mentorship. He / She provided us with the freedom and support to explore the research topics of our interest. His / Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff members of Computing Technologies, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement

K. Kalyan Saran(RA2211003011444)

G Venkat Reddy(RA2211003011463)

# ABSTRACT

Interview rejections are a frequent problem faced by students in placement drives, especially in the interview stage after clearing coding and aptitude tests. To counter this problem, an Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask has been designed. The platform allows students to share and solve coding and non-coding interview problems, supporting collaborative learning and real-time skill matching. Students are provided with performance-based feedback, such as correctness of code and time taken. There are three dedicated portals in the system: student, faculty, and admin. Faculty can upload problems, administer quizzes, view results, and provide feedback. An integrated AI chatbot supports the generation of non-coding problems and solves coding-related problems. Admin oversees all user activities and manages access permission. The innovative feature of this platform is its built-in AI-based support and post-interview question-sharing capability, supporting personalized learning and interview preparation. This system is intended to improve student preparedness and reduce interview-stage rejections. A key innovation is the integration of an AI-powered chatbot, which acts as a virtual tutor for students and a content generator for teachers. The chatbot leverages natural language processing (NLP) to interpret user queries, generate relevant quiz questions, and assist in resolving coding-related doubts in real time. This not only enhances user engagement but also fosters independent learning. The system also includes a performance monitoring module that uses visual tools such as Chart.js to track student progress and support personalized feedback delivery.

## TABLE OF CONTENTS


**ABSTRACT** iv

**TABLE OF CONTENTS** v

**LIST OF FIGURES** vii

**LIST OF TABLES** ix

**ABBREVIATIONS** viii

**CHAPTER NO. TITLE PAGE NO.**

**1 INTRODUCTION** 1

1.1 Introduction to Project 1

1.2 Motivation 2

1.3 Sustainable Development Goal of the Project 3

1.4 Product Vision Statement 3

1.5 Product Goal 5

1.6 Product Backlog (Key User Stories with Desired Outcomes) 8

1.7 Product Release Plan 9

**2 LITERATURE SURVEY** 10

**3 SPRINT PLANNING AND EXECUTION** 12

**3.1 Sprint 1** 12

3.1.1 Sprint Goal with User Stories of Sprint 1 12

3.1.2 Functional Document 17

3.1.3 Architecture Document 22

3.1.4 UI Design 27

3.1.5 Functional Test Cases 29

3.1.6 Daily Call Progress 29

3.1.7 Committed vs Completed User Stories 30

3.1.8 Sprint Retrospective 30

**3.2 Sprint 2** 31

3.2.1 Sprint Goal with User Stories of Sprint 2 31

3.2.2 Functional Document 32

3.2.3 Architecture Document 37

3.2.4 UI Design 42

3.2.5 Functional Test Cases 42

3.2.6 Daily Call Progress 43

3.2.7 Committed vs Completed User Stories 44

3.2.8 Sprint Retrospective 44

**4 RESULTS AND DISCUSSIONS 45**

4.1 Project Outcomes (Justification of outcomes and how they align with the goals) 45

4.2 Committed vs Completed User Stories 48

**5 CONCLUSIONS & FUTURE ENHANCEMENT 49**

**APPENDIX 51**

**A. PATENT DISCLOSURE FORM 51**

**B. CODE 52**

**C. PLAGIARISM REPORT 84**

# LIST OF FIGURES


| CHAPTER NO | TITLE | PAGE NO |
|---|---|---|
| 1 | Figure 1.1 MS Planner Board | 9 |
| 1 | Figure 1.2 Release Plan | 9 |
| 3 | Figure 3.1 User story for User Authentication | 13 |
| 3 | Figure 3.2 User story for Dashboard Interface Design | 14 |
| 3 | Figure 3.3 User story for Search Functionality | 15 |
| 3 | Figure 3.4 User story for Code Compilation and Evaluation | 16 |
| 3 | Figure 3.5 System Architecture Diagram | 23 |
| 3 | Figure 3.6 UI Design for loading Page | 27 |
| 3 | Figure 3.7 UI Design for Student Login page | 27 |
| 3 | Figure 3.8 UI Design for Faculty Login page | 28 |
| 3 | Figure 3.9 UI Design for Admin Login page | 28 |
| 3 | Figure3.10 Stand up Meetings | 29 |
| 3 | Figure 3.11 Bar Graph for Committed vs Completed User stories | 30 |
| 3 | Figure 3.12 Sprint Retrospective for Sprint-1 | 30 |
| 3 | Figure 3.13 System Architecture Diagram | 38 |
| 3 | Figure 3.14 UI Design for Landing page | 42 |
| 3 | Figure 3.15 Standup meetings | 43 |
| 3 | Figure 3.16 Bar Graph for Committed vs Completed User stories | 44 |
| 3 | Figure 3.17 Sprint Retrospective for Sprint-2 | 44 |
| 4 | Figure 4.1 Student login Landing Page | 45 |
| 4 | Figure 4.2 Faculty Login Landing Page | 45 |
| 4 | Figure 4.3 Faculty Posting Question | 46 |
| 4 | Figure 4.4 Student Attempting Coding Question | 46 |
| 4 | Figure 4.5 Program Output | 47 |
| 4 | Figure 4.6 Faculty Posting Quiz | 47 |

| | | |
|---|---|---|
| **4** | Figure 4.7 Student Attempting Quiz | **48** |
| **4** | Figure 4.8 Bar graph for Committed Vs Completed User Stories | **48** |

# LIST OF TABLES


| CHAPTER NO | TITLE | PAGE NO |
|---|---|---|
| 1 | Table 1.1 product backlog | 8 |
| 3 | Table 3.1 Detailed user stories for sprint 1 | 12 |
| 3 | Table 3.2 Access level authorization matrix | 21 |
| 3 | Table 3.3 Detailed functional test cases | 29 |
| 3 | Table 3.4 Detailed user stories for sprint 2 | 31 |
| 3 | Table 3.5 Access level authorization matrix | 36 |
| 3 | Table 3.6 Detailed functional test cases | 42 |

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction:

The shift from college studies to professional lives is a critical phase of a student's life. While many students manage to overcome the first part of campus placements, i.e., aptitude tests and coding tests, thousands of them are shortlisted out in the final and most critical phase – the interview process. It is an ongoing issue that indicates a lack on the part of students in terms of technical skills as well as soft skills, domain expertise, and the application of theoretical skills in practice. The lack of a structured method of recording, sharing, and learning from real-time interview experiences exacerbates the limitations on opportunities for learning and improvement. In order to mitigate this critical issue, an innovative and holistic platform titled the Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask has been created.

The primary goal of this platform is to help students who are interviewing by offering an interactive space where they can submit, answer, and comment on real interview questions. Students who have gone through interviews are invited to post the questions they faced, thereby building a growing collection of relevant questions for the use of others. This exercise not only promotes peer-to-peer collaborative learning but also helps in the identification of commonly asked questions in different companies and domains. The platform supports both coding and non-coding questions. In the case of coding questions, students can solve problems using an in-built compiler, view test case results, track their accuracy, and measure the time taken to solve each question, all of which are critical statistics for interview practice.

To enable orderly learning and supervision, the site provides three specific types of user logins: student, teacher, and administrator. With the student login, students can register with their college email IDs and, after getting authentication from the administrator, can use the dashboard. Through this interface, students can submit and work on questions, participate in quizzes , and receive feedback from teachers. Such an arrangement allows active participation and self-assessment, thus enhancing preparatory methods. Alternatively, the teacher login provides teachers with a separate dashboard to add coding and non-coding questions,

participate in quizzes, track student performance, and provide detailed feedback. Such teacher participation is essential for educational supervision, guidance, and systematic evaluation.

One of the most impressive features of the platform is its integration of an artificial intelligence-based chatbot. The chatbot is integrated into the faculty dashboard and serves a variety of purposes. It is able to generate non-coding questions from short text-based inputs, thus enabling faculty to generate quiz content with ease. The chatbot also offers coding-related question support, enabling faculty and students to solve programming issues and improve their understanding of logic and syntax. This balanced integration of artificial intelligence into the education system improves productivity and improves the learning process as a whole. Admin login is important to maintaining the integrity and structure of the platform. The admin has full control over the users' access, tracking of students and staff, and overall dashboard management

## 1.2 Motivation

The transition from academic life to the professional world represents a defining moment for students, often characterized by high competition and immense pressure. Despite clearing the preliminary stages of campus placements—aptitude and coding tests—many students face rejection during the final interview round. This points to a significant gap in their preparedness, particularly in technical expertise, soft skills, and the practical application of theoretical knowledge.

What amplifies this challenge is the absence of a unified, real-time learning platform where students can share and benefit from authentic interview experiences. The current learning methods are often isolated and lack collaboration, leaving students with limited exposure to real-world problem-solving and peer support. Recognizing this unmet need, we were driven to create a solution that encourages peer-driven learning, enhances practical skills, and facilitates teacher-led feedback—all within one cohesive environment.

This motivation led to the development of the Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask. The platform not only addresses the skill gap through hands-on practice and collaboration but also integrates artificial intelligence to support dynamic question generation and problem-solving. By fostering a structured and interactive ecosystem for both students and faculty, the platform aims to transform the way

students prepare for technical interviews—making learning personalized, accessible, and smarter.

## 1.3 Sustainable Development Goal of the Project

This project strongly contributes to **SDG 4: Quality Education**, which promotes inclusive and equitable education and lifelong learning opportunities. The platform enables students to access a centralized, interactive space where they can practice real-world interview questions, receive feedback, and improve their technical and soft skills. This holistic preparation enhances their confidence and employability, helping bridge the gap between academic learning and professional requirements.

Moreover, by integrating an AI-powered chatbot capable of generating non-coding questions, assisting in logic building, and resolving programming-related queries, the project also supports **SDG 9: Industry, Innovation, and Infrastructure**. The use of artificial intelligence in education improves the efficiency and adaptability of content creation, making learning more dynamic, personalized, and future-ready. This AI integration promotes innovation in the education sector by automating routine tasks and encouraging intelligent tutoring systems.

In addition, the platform fosters collaborative learning and academic mentorship through distinct student, teacher, and admin logins. This ecosystem nurtures responsible educational practices and supports scalable skill development, especially in underserved communities where expert resources might be limited. Through these efforts, the platform not only addresses skill development at an individual level but also contributes to building a more educated, innovative, and inclusive society.

## 1.4 Product Vision Statement

1.4.1. Audience:

- Primary Audience: College Students Particularly those in their pre-final and final years who are preparing for campus placements, technical interviews, and competitive coding exams

- Secondary Audience: Recruiters and Industry Professionals Who may use this platform to identify talent, understand student preparedness, or suggest commonly asked interview questions.

1.4.2. Needs:

- Primary Needs:

- A platform where students can submit, attempt, and comment on real interview questions (both coding and non-coding) to simulate interview conditions.

- Real-time code editor and compiler with test case validation, accuracy tracking, and time analysis to assess performance during problem-solving.

-An intelligent chatbot that can generate non-coding questions from inputs, assist with logic building, and answer programming-related queries for immediate doubt resolution.

- Secondary Needs:

- Tools for students and teachers to monitor learning progress, submission accuracy, average time taken, and other key performance indicators.

- A secure, scalable backend that can handle a growing user base without compromising data integrity or performance.

1.4.3. Products:

- Core Product: A comprehensive AI-powered code compilation and quiz platform that supports real-time interview practice, integrated quizzes, coding with an in-browser compiler, and AI-driven question generation—targeted at improving placement readiness for students.

- Additional Features:

- Real-Time Performance Tracking: Provides statistics such as accuracy rate, question-solving time, and performance trends.

- Feedback System: Allows teachers to provide evaluations and students to give platform feedback for improvements.

- Role-Based Access Different dashboards and controls for students, teachers, and administrators.

1.4.4. Values:

- Core Values:

  - Personalization: Tailoring learning experiences to individual needs and preferences.

  - Community: Fostering a collaborative environment for skill-sharing and mutual learning.

  - Engagement: Using gamification to make learning interactive and fun.

- Differentiators:

  - AI Personalization: Leveraging AI to create adaptive learning paths and provide real-time feedback.

  - Live Peer Engagement: Facilitating real-time interactions and learning sessions.

  - Gamified Learning: Incorporating points, badges, and leaderboards to motivate learners.

## 1.5 Product Goal

The Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask is designed with a transformative vision—to redefine how students prepare for the most decisive phase of their academic journey: entering the professional world. Traditional education systems often fall short in equipping students with the necessary real-world skills, especially during placement preparation, technical interviews, and coding assessments. In this gap lies the fundamental motivation of the platform—to revolutionize the learning experience by fostering a personalized, intelligent, and community-oriented educational ecosystem that evolves with each learner.

At the core of the platform lies the principle of personalized education. The one-size-fits-all model of traditional academic systems has become increasingly obsolete, particularly when it comes to preparing for dynamic and evolving industry needs. Students vary widely in their learning pace, comprehension abilities, and problem-solving styles. To cater to these diverse learning curves, the platform offers customized learning paths that adapt to individual performance and progress. Whether a student struggles with recursion or excels in object-oriented programming, the system learns from their interaction patterns and tailors future content accordingly. This ensures that each user receives a learning experience that resonates with their skill level, preference, and professional goals.

This level of personalization is made possible through artificial intelligence integration within the platform. The AI engine continuously monitors performance indicators such as question-

solving time, accuracy, consistency, and engagement frequency. Based on these data points, it suggests the next set of quizzes, topics, or coding problems that a student should focus on. Over time, this creates a self-adaptive learning loop where the platform becomes increasingly aligned with the learner's unique needs. The use of AI also allows the system to provide micro-feedback and insights—identifying, for instance, if a student frequently fails in edge-case test scenarios or shows patterns of syntax-related errors. By addressing these nuances in real time, the platform transforms preparation from a reactive effort to a proactive, guided experience.

The product further integrates a robust in-browser code compiler, which allows students to write, compile, and test code in real time. This feature eliminates the need for external IDEs and enables a seamless learning flow. More importantly, each coding problem includes built-in test cases with pass/fail status and execution time tracking, offering learners instant feedback on their logical accuracy and efficiency. Over time, students can visualize their growth via detailed performance graphs, identify weak spots, and improve systematically. By combining theory with practical execution, the platform ensures that users don't just consume knowledge but internalize and apply it effectively.

Beyond individual learning, the platform aims to build a community of skill-sharing and collaborative growth. In today's connected world, learning is as much a social experience as it is personal. With features that allow students to submit real interview questions, comment on solutions, and rate difficulty levels, the platform creates an ever-evolving repository of authentic interview content. This peer-contributed knowledge base serves as an invaluable resource, helping others prepare for the exact scenarios they may face during placement drives. Moreover, it empowers those who've already experienced interviews to give back to the student community, fostering a culture of mentorship and shared learning.

To support structured guidance and academic oversight, the platform also includes role-specific dashboards for students, teachers, and administrators. The student dashboard is designed to be intuitive and action-oriented, enabling users to access personalized quizzes, past performance stats, and coding challenges in one place. Teachers, on the other hand, are provided with a robust backend interface that allows them to create questions (coding and non-coding), set quizzes, monitor student progress, and provide targeted feedback. The administrator dashboard serves as the control panel for the entire platform—managing access rights, tracking activity metrics, and ensuring the smooth operation of all modules. This multi-role architecture ensures that the platform remains structured, supervised, and scalable while promoting autonomy and self-learning.

One of the most distinctive features—and a major contributor to the product's goals—is the AI-powered chatbot assistant. Available primarily in the teacher and student dashboards, this chatbot enhances the usability and intelligence of the platform. Teachers can utilize it to generate quiz questions from minimal input—entering, for example, a topic like "pointers in C," and receiving multiple-choice or short-answer questions ready to be added to the platform. For students, the chatbot serves as an on-demand tutor, capable of resolving doubts, recommending further reading, or explaining the logic behind particular answers. This 24/7 availability adds immense value to the learning experience, especially for students preparing late at night or revising before an interview.

The chatbot also introduces the potential for sentiment and engagement analysis. By understanding how users interact—what questions they skip, how long they hesitate before responding, or when they drop out of a quiz—the platform can fine-tune not just what content is delivered, but how it is delivered. For example, a student showing signs of frustration might be served easier questions to rebuild confidence or offered motivational tips. These subtle shifts, powered by machine learning models, make the educational journey more empathetic and supportive.

To maintain user engagement, the platform incorporates gamification strategies such as points, badges, and leaderboards. Learning can often become monotonous, especially when conducted online. By introducing game elements, the platform motivates students to remain active and consistent. For instance, earning a badge for “5 consecutive days of quiz completion” or rising on the coding leaderboard can provide just the right nudge to keep learners invested. These incentives aren’t merely decorative—they reflect achievement, encourage repetition, and reinforce learning behaviour.

Another cornerstone of the product goal is its commitment to accessibility and inclusiveness. In many parts of the world, students do not have access to expert mentors, competitive coding platforms, or structured interview preparation programs. By being web-based, light on resources, and mobile-friendly, this platform democratizes access to quality education. Students from tier-2 or tier-3 cities, under-resourced colleges, or economically constrained backgrounds can benefit from the same features as their urban counterparts. The platform supports multiple device types and includes accessibility features like keyboard navigation, font-size scaling, and adaptive UI—ensuring that no student is left behind.

## 1.6 Product Backlog

| S.No | User Stories |
|---|---|
| #US 1 | As a new user, I want to register with my institutional email and receive role-based access after admin approval,so that I can securely use platform features appropriate to my role (student, teacher, or admin) |
| #US 2 | As a student, I want to view an intuitive dashboard with my recent quizzes, performance charts, and coding stats,so that I can easily monitor my progress and access learning resources. |
| #US 3 | As a teacher, I want to create coding and non-coding quizzes with time limits and difficulty tags,so that students can be assessed consistently and fairly across topics |
| #US 4 | As a student, I want to write, run, and test my code within the platform using test cases, so that I can receive real-time feedback and improve my programming skills effectively. |
| #US 5 | As a student, I want to ask the chatbot conceptual or debugging questions, so that I can get instant AI-generated responses and continue learning without delay. |
| #US 6 | As a teacher, I want to post personalized feedback and view performance charts of my students,so that I can support their learning journey with actionable insights. |
| #US 7 | As an admin, I want to manage users, approve registrations, monitor quiz usage, and log user actions,so that the platform remains organized, secure, and compliant. |
| #US 8 | As a developer, I want to store user credentials using encryption and enforce role-based access queries,so that user data stays protected from unauthorized access and system breaches. |

Table 1.1 product backlog

The product backlog of was configured using the MS planner Agile Board which is represented in the following Figure 1.1. The Product Backlog consists of the complete user stories of Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask. Each user story consist of necessary parameters like MoSCoW prioritization, Functional and non functional parameters, detailed acceptance criteria with linked tasks.

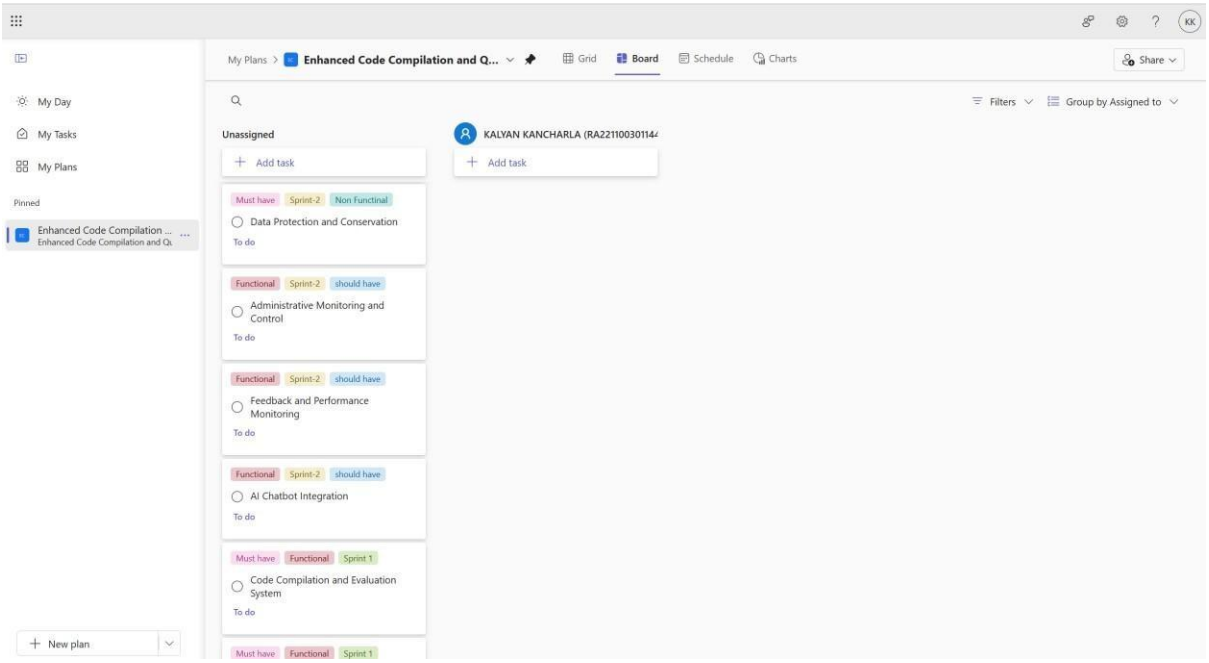


Figure 1.1 MS Planner Board

## 1.7 Product Release Plan

The following Figure 1.2 depicts the release plan of the project

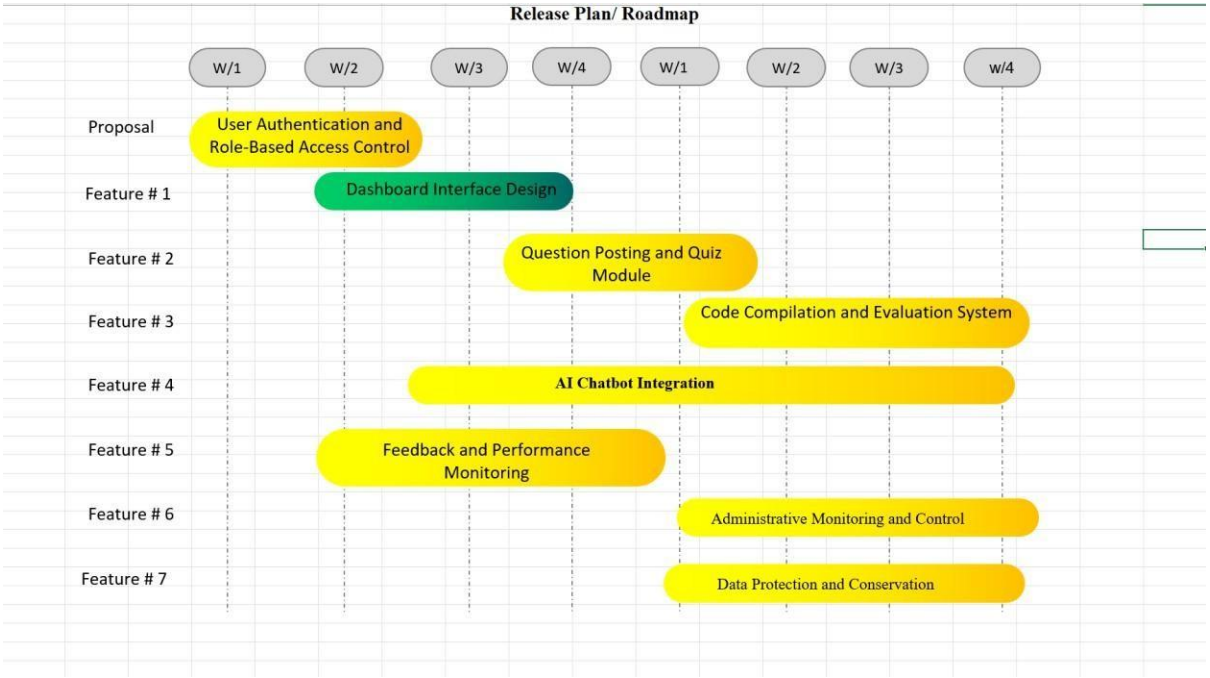


Figure 1.2 Release plan

# CHAPTER 2

# LITERATURESURVEY

There are some websites focused on the improvement of coding skills via properly screened problem sets and time limits. These websites are intended to simulate real-world coding environments in which users are able to apply reasoning and efficiency within constraints. Performance analytics and leaderboard scores contribute to gamification of the learning process, enabling frequent use. Despite these advantages, these websites are likely to lack simulation of real interview settings or comprehensive feedback mechanisms targeting conceptual gaps. Most of them also do not enable sharing of user-generated questions, thus restricting exposure to diverse problem types and context relevance for users preparing for job interviews

Interview preparation software has been created to aid students in improving oral and technical communication skills through recorded mock interviews and evaluation. Software is usually focused on HR and technical round preparation of users through a set of common questions and suggested answers. Interactive modules simulate interview situations to aid in boosting confidence levels. The sites, however, usually exist as isolated systems without collaborative learning features or the ability to share real-time user feedback. They also fail to incorporate adaptive feedback mechanisms based on individual performance trends, thus their long-term application in overall interview preparation less effective.

Coding test platforms are widely adopted by education institutions and organizations to evaluate the coding skill of candidates. These platforms are typically able to handle a variety of programming languages, provide real-time code checking, and enable bulk test management. Their core strength lies in automated marking and the ability to provide customized tests. They are inclined to ignore pedagogical considerations such as individualized feedback or conceptual knowledge, which are crucial to develop skills. Moreover, these tools do not include non- coding components such as aptitude or communication-type questions and thus tend to provide fragmented learning experiences that do not provide end-to-end support to interview preparation.

Student performance monitoring systems allow teachers to monitor academic achievement using data visualization dashboards that show test scores, attendance, and participation data. The systems are important in the identification of areas of weakness and implementation of targeted interventions. They support data-driven decision-making for curricular enhancement. The systems do not usually provide students with interactive learning

environments or immediate resources for performance improvement. There is limited student autonomy for self-assessment or collaboration with peers, and the absence of real-time feedback mechanisms detracts from the contribution of the platform to learning achievements. Web quiz systems are commonly used within the context of instructional settings to quiz conceptual knowledge within some set of topics.

The systems are easy to implement, support a variety of question types, and allow for instant marking, making them suitable for rapid assessment. The systems also promote continuous student engagement through the use of randomization and time constraints. Nevertheless, despite their usefulness, the systems are typically limited as regards their use within rich learning environments. They are typically not designed to support complex problem-solving activities, like coding problems, nor do they support adaptive learning and interaction with smart agents that have the potential to enhance Learning.

# CHAPTER 3

# SPRINT PLANNING AND EXECUTION

## 3.1 Sprint 1

### 3.1.1 Sprint Goal with User Stories of Sprint 1

The Goal of the first sprint is to construct the user landing page and to enable the search functionalities such as skills and courses.

- As a user, I want to land on a visually appealing and informative homepage, so that I can understand the platform's features and navigate easily.
- As a user, I want to use a search bar to look up courses or skills by keyword, so that I can quickly find what I need.
- As a student, I want to apply filters (e.g., difficulty level, domain, instructor), so that I can refine search results to match my learning goals.
- As a guest user, I want to explore publicly available courses or skills without logging in, so that I can evaluate the platform before registering.

The following table 3.1 represents the detailed user stories of the sprint 1

| S.NO | Detailed User Stories |
|---|---|
| US #1 | As a new user, I want to register with my institutional email and receive role-based access after admin approval,so that I can securely use platform features appropriate to my role (student, teacher, or admin) |
| US #2 | As a student, I want to view an intuitive dashboard with my recent quizzes, performance charts, and coding stats,so that I can easily monitor my progress and access learning resources. |
| US #3 | As a teacher, I want to create coding and non-coding quizzes with time limits and difficulty tags,so that students can be assessed consistently and fairly across topics |

| | |
|---|---|
| US #4 | As a student, I want to write, run, and test my code within the platform using test cases, so that I can receive real-time feedback and improve my programming skills effectively. |

Table 3.1 Detailed User Stories of sprint 1

Planner Board representation of user stories are mentioned below figures 3.1,3.2,3.3 and 3.4

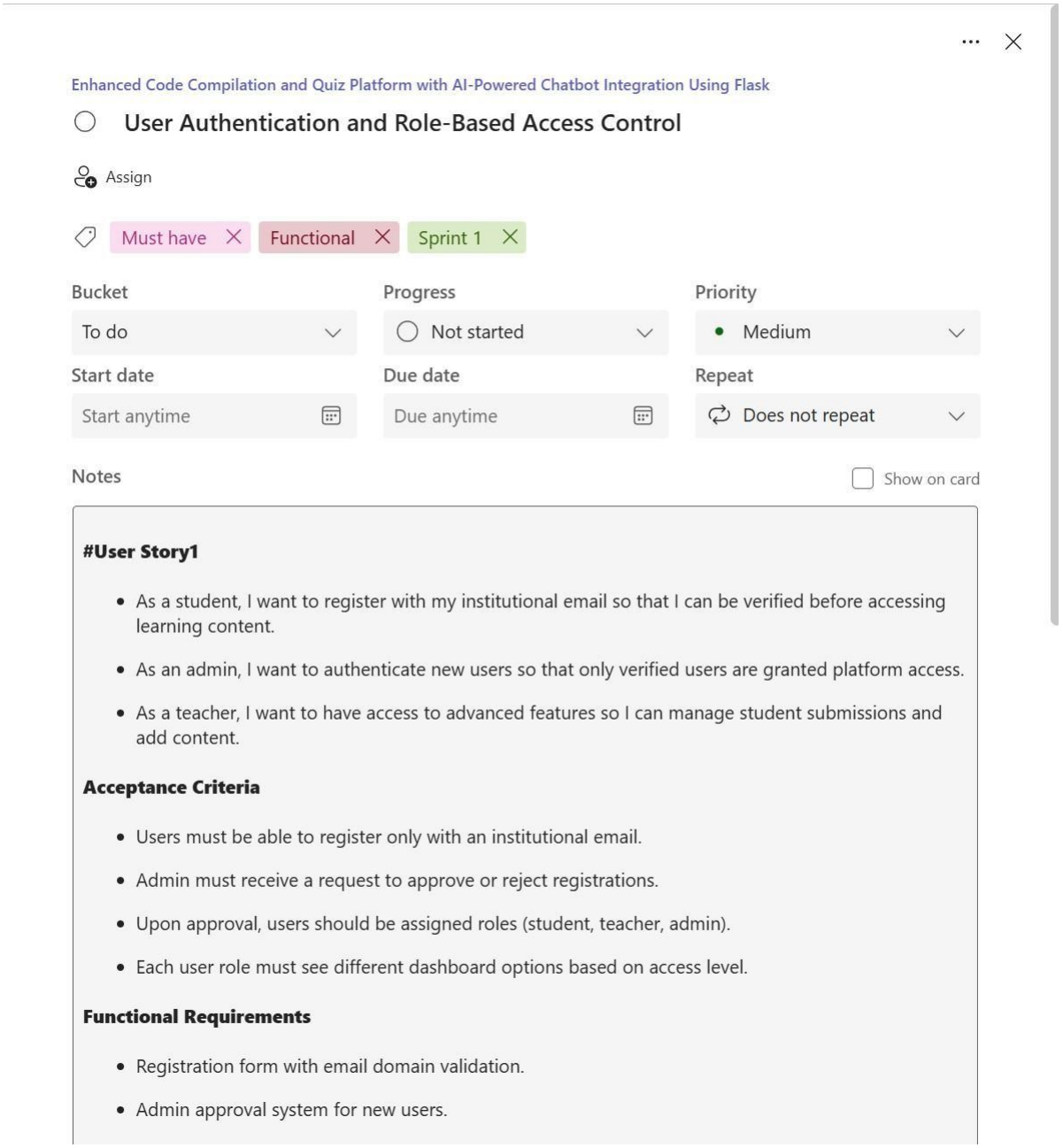


Figure 3.1 user story for user Authentication

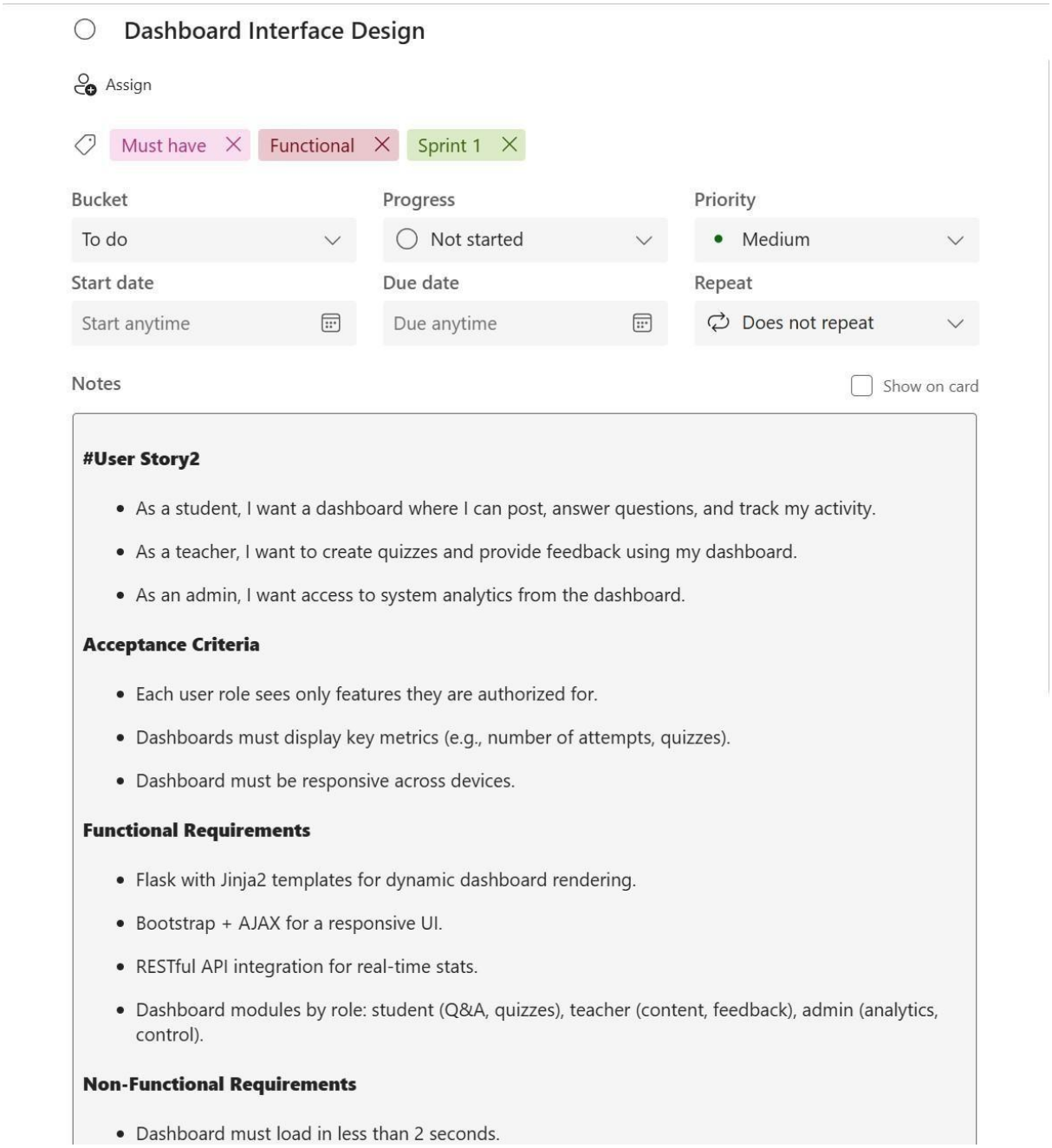


Figure 3.2 user story for Dashboard Interface Design

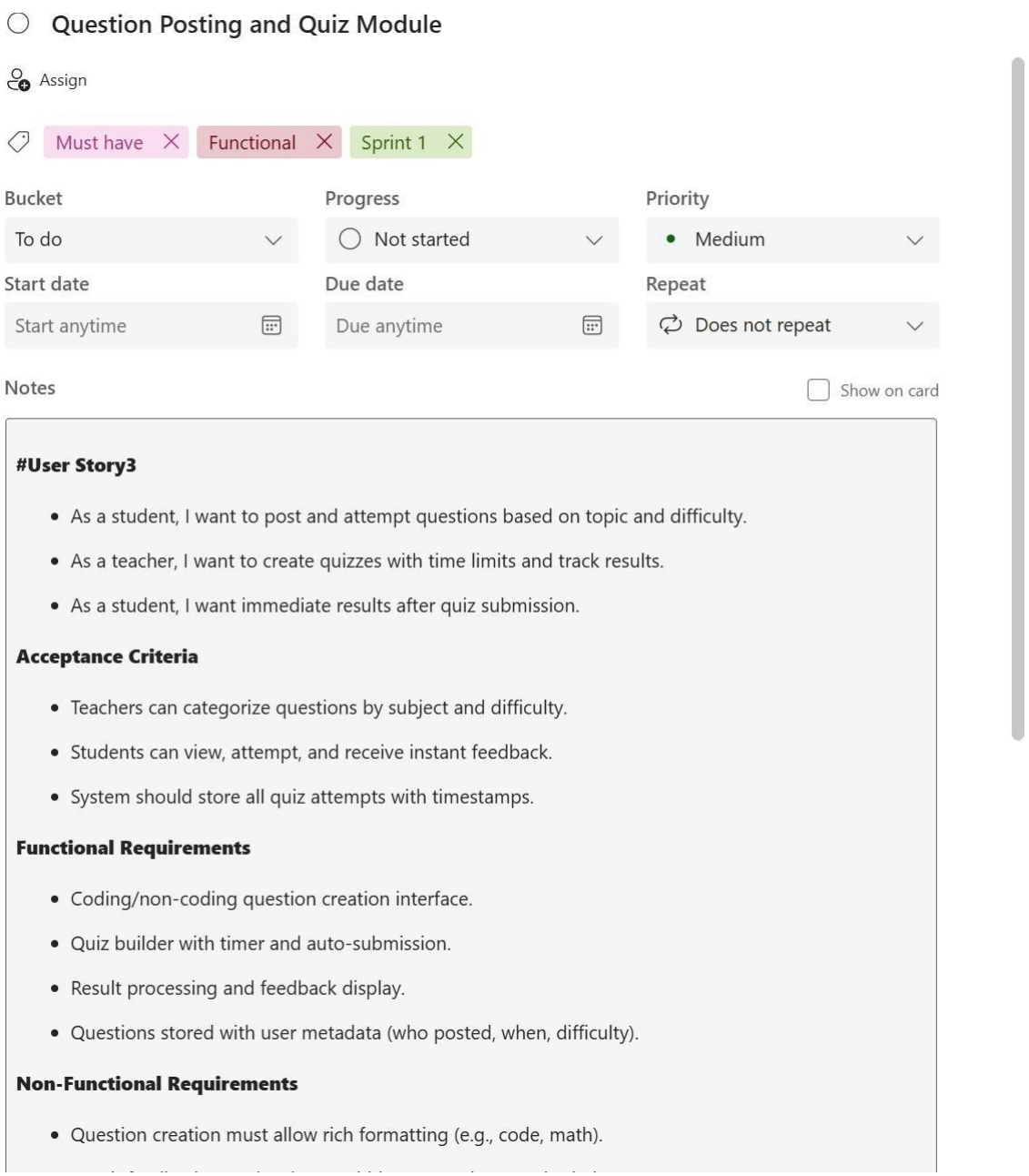


Figure 3.3 User story for search functionality

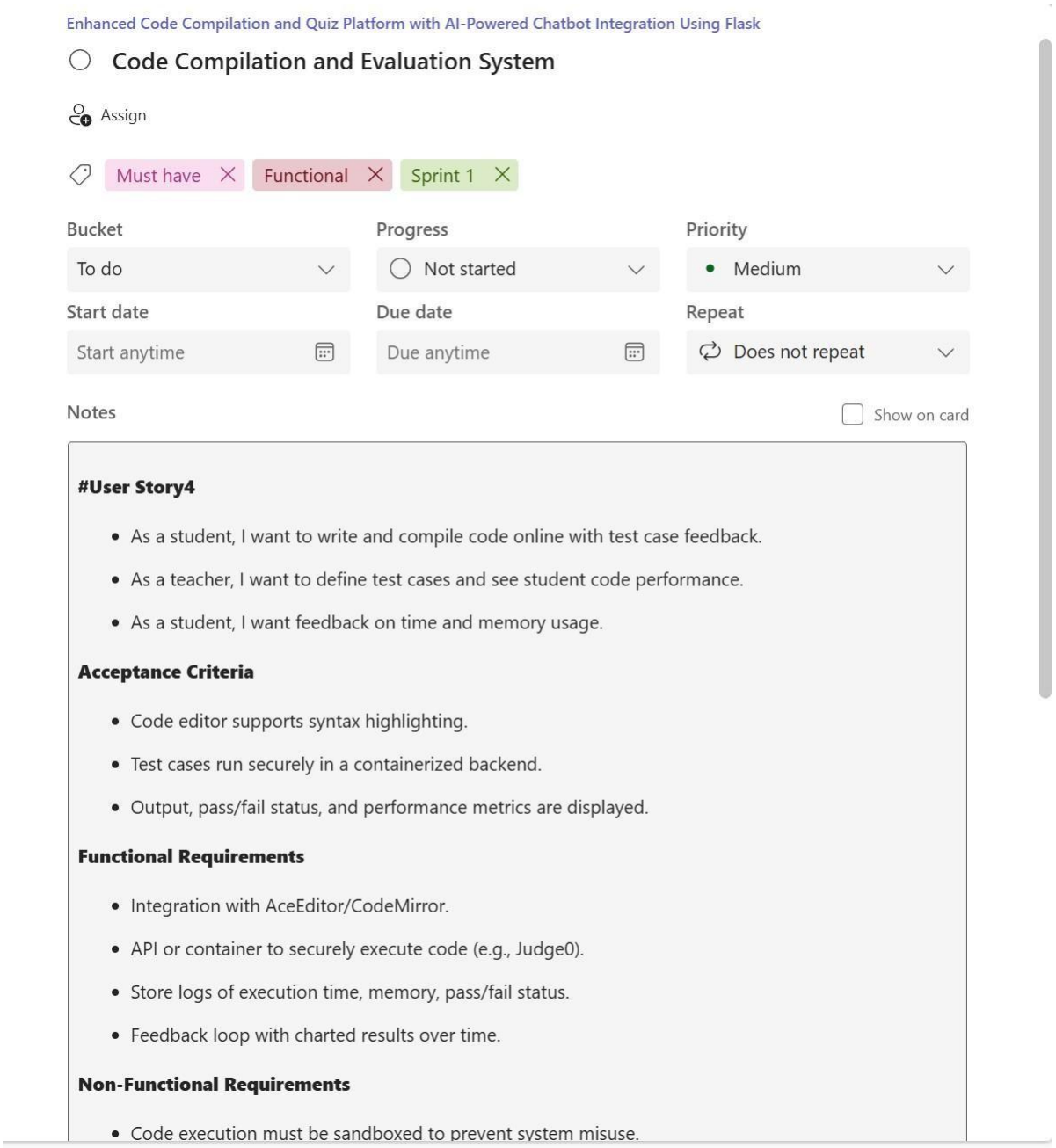


Figure 3.4 User story for Code Compilation and Evaluation System

## 3.1.2 Functional Document

3.1.2.1. Introduction

The Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask is a next-generation educational web platform designed to help students prepare effectively for technical interviews and coding assessments. The project brings together the power of artificial intelligence, real-time collaboration, and academic mentorship to offer an immersive, skill-driven learning experience. It caters to students by simulating real-world interview scenarios, offering AI-assisted doubt resolution, tracking performance over time, and encouraging peer-to-peer question sharing. Through dynamic role-based dashboards and secure access control, the platform also supports teachers and administrators in their roles as guides and facilitators of the learning process.

3.1.2.2. Product Goal

The primary goal of this platform is to empower students to excel in competitive programming and interview readiness through a personalized, intelligent, and collaborative environment. The platform seeks to:

- Provide personalized learning paths by analysing performance metrics such as quiz results, code accuracy, and response times.
- Enable real-time peer-to-peer interaction through community-posted questions and feedback mechanisms.
- Use AI chatbots to support real-time coding doubt resolution and generate non-coding questions for quizzes and self-evaluation.
- Support educators in monitoring student progress, managing content, and offering detailed, constructive feedback.
- Build a community-driven knowledge repository based on real interview questions shared by students.

3.1.2.3. Demography (Users, Location)

Users

- Target Users:
    - College and university students preparing for technical interviews.
    - Teachers and placement trainers.
    - Platform administrators overseeing usage and content quality.
- User Characteristics:
    - Students with varying technical abilities, from beginner to advanced.
    - Faculty with experience in curriculum delivery and mentoring.
    - Admins with IT management knowledge for platform governance.

Location

- Primary Focus:
    - Global, with an initial emphasis on regions with strong tech education ecosystems such as India, Southeast Asia, and North America.
- Target Institutions:
    - Engineering colleges, placement training centres, and online coding bootcamps.

3.1.2.4. Business Processes

User Registration and Authentication

- Users register via institutional email, and authentication is managed by the administrator.
- JWT or session-based login provides secure session handling.
- RBAC (Role-Based Access Control) ensures user permissions are aligned with their role.

Personalized Learning and Quiz Flow

- Students access recommended questions and quizzes based on previous performance.
- Teachers manage question banks, quizzes, and feedback.
- Students receive performance-based content suggestions.

Code Submission and Evaluation

- Code is submitted via an in-browser editor.
- The backend securely compiles and executes the code.
- Results, accuracy, and time/memory usage are displayed immediately.

AI Chatbot Interaction

- Chatbot assists in both student and faculty dashboards.
- Faculty use it to generate quiz questions from topic inputs.
- Students can clarify doubts in real-time using conversational input.

3.1.2.5. Features

This project focuses on implementing the following key features

Feature 1: Personalized Quiz & Code Recommendation System

Description:
The system analyses student quiz scores and coding trends to recommend suitable practice questions and challenges.

Userstory:
As a student, I want to receive personalized coding and quiz suggestions so that I can focus on areas where I need the most improvement.

Feature 2: Real-Time Code Compilation with Evaluation

Description:
Students write and execute code in-browser using an integrated editor. Their submissions are evaluated against instructor-defined test cases.

UserStory:

As a student, I want to run and test my code directly on the platform, so I can practice coding questions in a real interview-like setup.

Feature 3: AI-Powered Chatbot Assistance

Description:

An integrated chatbot helps users by answering queries, generating questions, and offering contextual support based on user inputs.

Userstory:

As a teacher, I want the chatbot to create non-coding questions from topic inputs so that I can prepare quizzes faster.

As a student, I want to ask doubts to the chatbot so I can get instant guidance without needing to wait for a teacher.

Feature 4: Role-Based Dashboards

Description:

Each user sees a dashboard tailored to their role—student, teacher, or admin—with only relevant modules and controls.

Userstory:

As a teacher, I want to monitor my students' performance and give feedback so I can help them preparebetter.

As an admin, I want to control user access and content quality so that the platform remains organized and secure.

Feature 5: Interview Experience Sharing

Description:

Students can post real interview questions from their placement drives, building a collaborative knowledge base.

Userstory:

As a student, I want to share questions I was asked in my interview so that others can prepare using real-world scenarios.

3.1.2.6. Authorization Matrix

| Role | Access Level |
|---|---|
| Administrator | Full access to manage users, questions, feedback, and dashboard analytics. |
| Teacher | Can create quizzes, provide feedback, monitor student performance. |
| Student | Can attempt quizzes, post questions, get AI support, and receive feedback. |

Table 3.2 Access level Authorization Matrix

3.1.2.7. Assumptions

- The AI chatbot will use pretrained models like GPT, enhanced with prompt engineering to match educational scenarios.
- Developers will have access to cloud resources (e.g., Docker, Judge0 API, GPT API) for hosting, code execution, and AI services.
- Teachers and student testers will provide regular feedback during UAT (User Acceptance Testing).
- The system will comply with privacy regulations such as GDPR and India's IT Act, with encrypted credentials and secure data transfer.
- Users will have internet access with modern browsers to use the platform without installation requirements.

## 3.1.3 Architecture Document

3.1.3.1. Application

Microservices Architecture

The Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration is designed using a modular microservices architecture to ensure scalability, maintainability, and high performance. Each major function of the platform is encapsulated into independent services that communicate via RESTful APIs, allowing the system to scale individual components without affecting the overall platform.

- The platform is structured into independent services (e.g., authentication, quiz management, AI chatbot) that communicate via REST APIs, enabling modularity, scalability, and easy maintenance.
- A real-time code editor with backend execution (via Judge0 API or containerized environments) allows students to run code, receive instant feedback, and analyse performance metrics like speed and memory.
- Integrated AI chatbot provides real-time responses to student queries, generates quiz questions for teachers, and helps clarify coding logic using natural language processing.
- Teachers and students can create, categorize, and share questions. Quizzes can be timed and automatically evaluated, with data stored for analytics and feedback.
- Student progress is visualized using Chart.js graphs, enabling users to track improvements and allowing Teachers to provide individualized feedback.
- Admin users manage platform-wide settings, content, user roles, and system logs to maintain security, ensure user compliance, and monitor engagement metrics.
- JWT or session-based authentication ensures secure access. Role-Based Access Control (RBAC) distinguishes permissions for students, faculty, and administrators.

3.1.3.2 System Architecture-

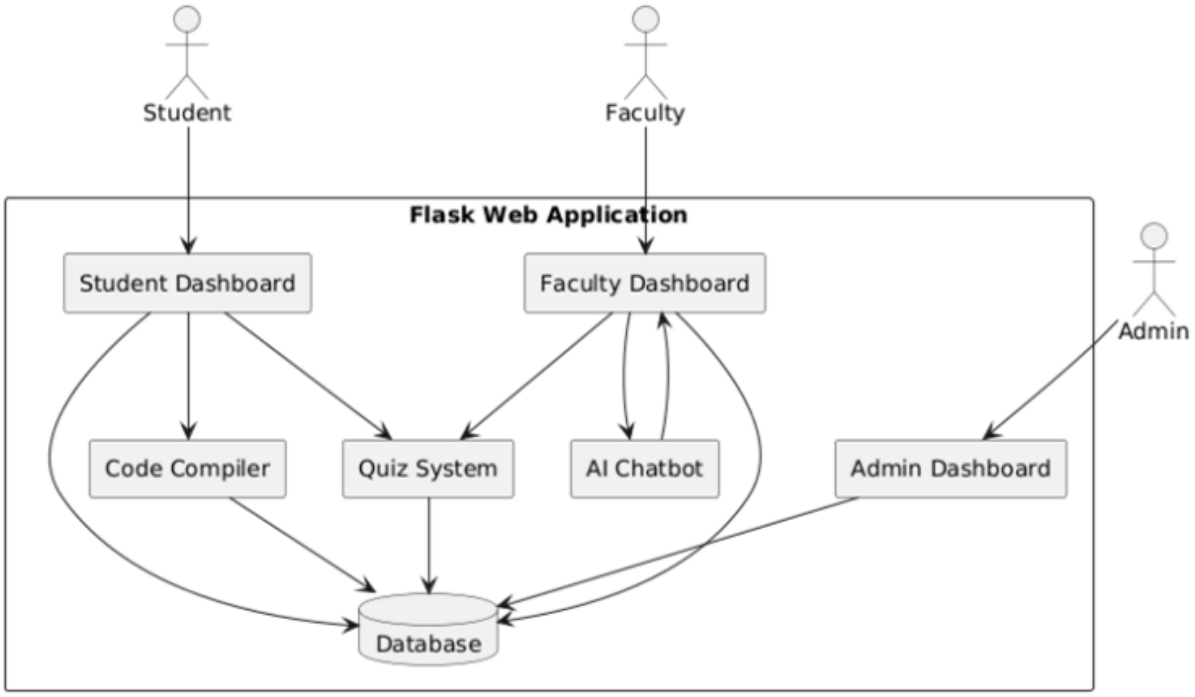


Figure 3.5 System Architecture Diagram

3.1.3.3. Data Exchange Contract:

Frequency of Data Exchanges

Data exchanges within the platform are optimized based on the criticality and responsiveness required for each operation:

- Real-TimeExchanges:
  Real-time communication is used for essential user interactions such as:
  - User login and authentication
  - Quiz attempts and code submission evaluations
  - Chatbot queries and responses
  - Feedback delivery and notification triggers
- PeriodicSync:
  Scheduled synchronization is used for non-critical processes to improve performance:
  - User performance history aggregation

- Quiz analytics reports
- Usage statistics and platform-wide logs for admin dashboards
- Backup of quiz/question repositories and student progress

Data Sets

The system operates on multiple types of data, each having specific exchange needs:

- User Data:
  Includes names, institutional email addresses, hashed passwords, role assignments, and activity metadata.
  Exchanged during:
  - Login
  - Registration
  - Role changes
  - Profile updates
- Question and Quiz Data:
  Consists of:
  - Coding/non-coding questions
  - Quiz structure and configuration
  - Difficulty levels, tags, time limits Exchanged during:
  - Quiz creation and update
  - Question attempts and evaluations
- Code Submission and Evaluation Data:
  Includes:
  - User code input
  - Test case results
  - Execution performance (time/memory) Exchanged during:
  - Live compilation and evaluation cycles

- Chatbot Interaction Data:
  Covers:
    - Student/teacher queries
    - AI-generated question/answer logs
    - Sentiment or query categorization (for analytics) Exchanged during:
    - Chatbot session requests and responses
- Feedback and Performance Logs:
  Includes:
    - Teacher feedback
    - Student progress trends
    - Visual performance analytics (charts, scores) Exchanged during:
    - Dashboard loading and history retrieval

Mode of Exchanges (API, File, Queue, etc.)

Different data exchange mechanisms are used depending on the nature of the transaction:

- API (RESTful):
  RESTful APIs manage most real-time interactions between frontend and backend, such as:
    - User authentication and role validation
    - Quiz start, submission, and evaluation
    - AI chatbot integration and data return
    - CRUD operations for questions, quizzes, and feedback
- Message Queues (Asynchronous Tasks):
  Technologies like Celery + Redis, or RabbitMQ, are used to handle background and time-independent tasks:
    - Notification delivery (e.g., "Feedback posted")
    - Email confirmation messages

- System logs and performance tracking jobs
- Quiz timer triggers and submission auto-handling

- File-Based Exchanges:
  Used for:
  - Bulk upload of quiz banks or user lists
  - CSV exports of performance data
  - Backup of user-generated content or logs
    Typically handled through services like:
  - Local server storage during upload
  - Cloud storage services (e.g., AWS S3) for persistence

### 3.1.4 UI DESIGN

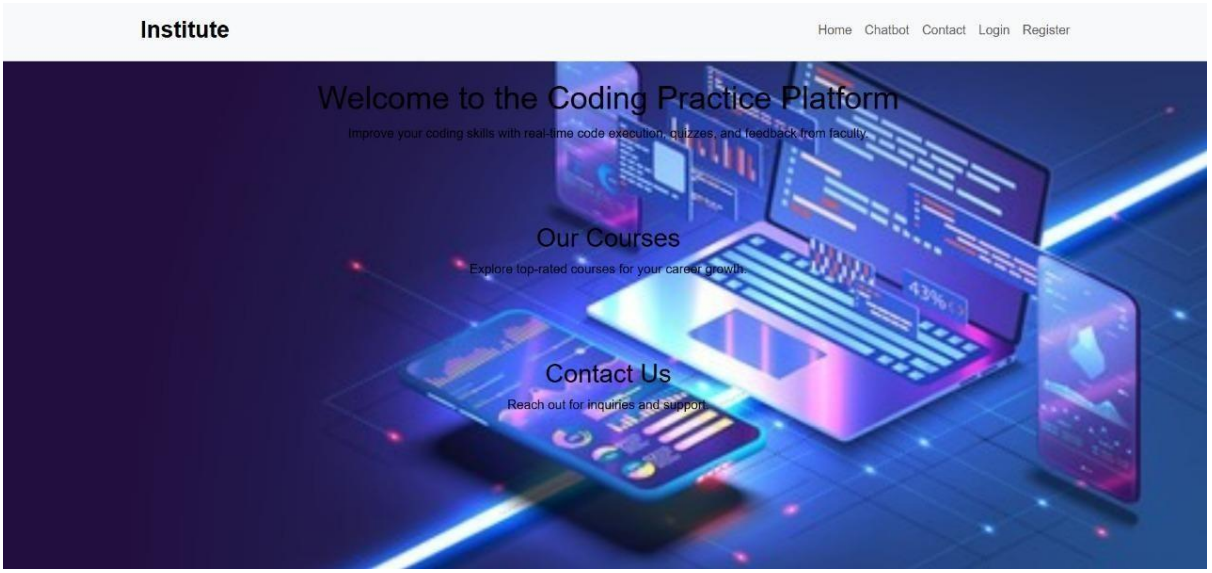


Figure 3.6 UI Design for Landing page


Figure 3.7 UI design for Student login page


Figure 3.8 UI design for Faculty login page


Figure 3.9 UI design for Admin login page

## 3.1.5 Functional Test Cases

| | Feature | Test Case | Steps to execute test case | Expected Output | Actual Output | Status | More Information |
|---|---|---|---|---|---|---|---|
| 2 | User Login | Valid Login | Open login page.<br>Enter valid username and password.<br>Click login. | The user should be successfully logged into the dashboard with correct credentials. | The user is successfully logged in.<br>The admin/student/faculty dashboard is displayed. | Pass | Ensure role redirection works. |
| 3 | User Registration | New User Registration | Open registration page.<br>Fill valid details.<br>Click register. | The user should be successfully registered and redirected to the login page. | The user is successfully registered.<br>Redirected to login page. | Pass | Check for duplicate user errors. |
| 4 | Code Compilation | Compile Python Code | Login as user.<br>Navigate to compiler.<br>Enter code and run. | The entered Python code should compile and display the correct output or an error if any. | The code is compiled.<br>Output is displayed below the code editor. | Pass | Validate syntax error handling. |
| 5 | Quiz Attempt | Submit Quiz Answers | Login as student.<br>Start a quiz.<br>Select answers and submit. | The submitted quiz should be evaluated and the result should be displayed to the student. | The quiz is submitted.<br>Score and feedback are displayed. | Pass | Check scoring logic. |

Table 3.3 Detailed Functional Test Case

## 3.1.6 Daily Call Progress

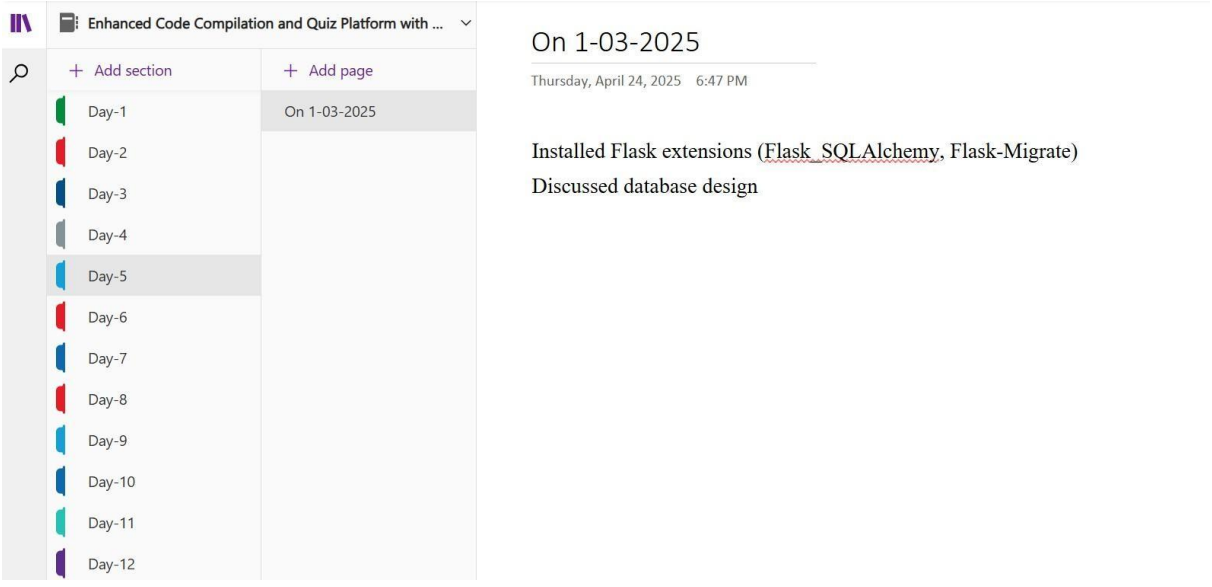


Figure 3.10 Standup meetings

### 3.1.7 Committed Vs Completed User Stories

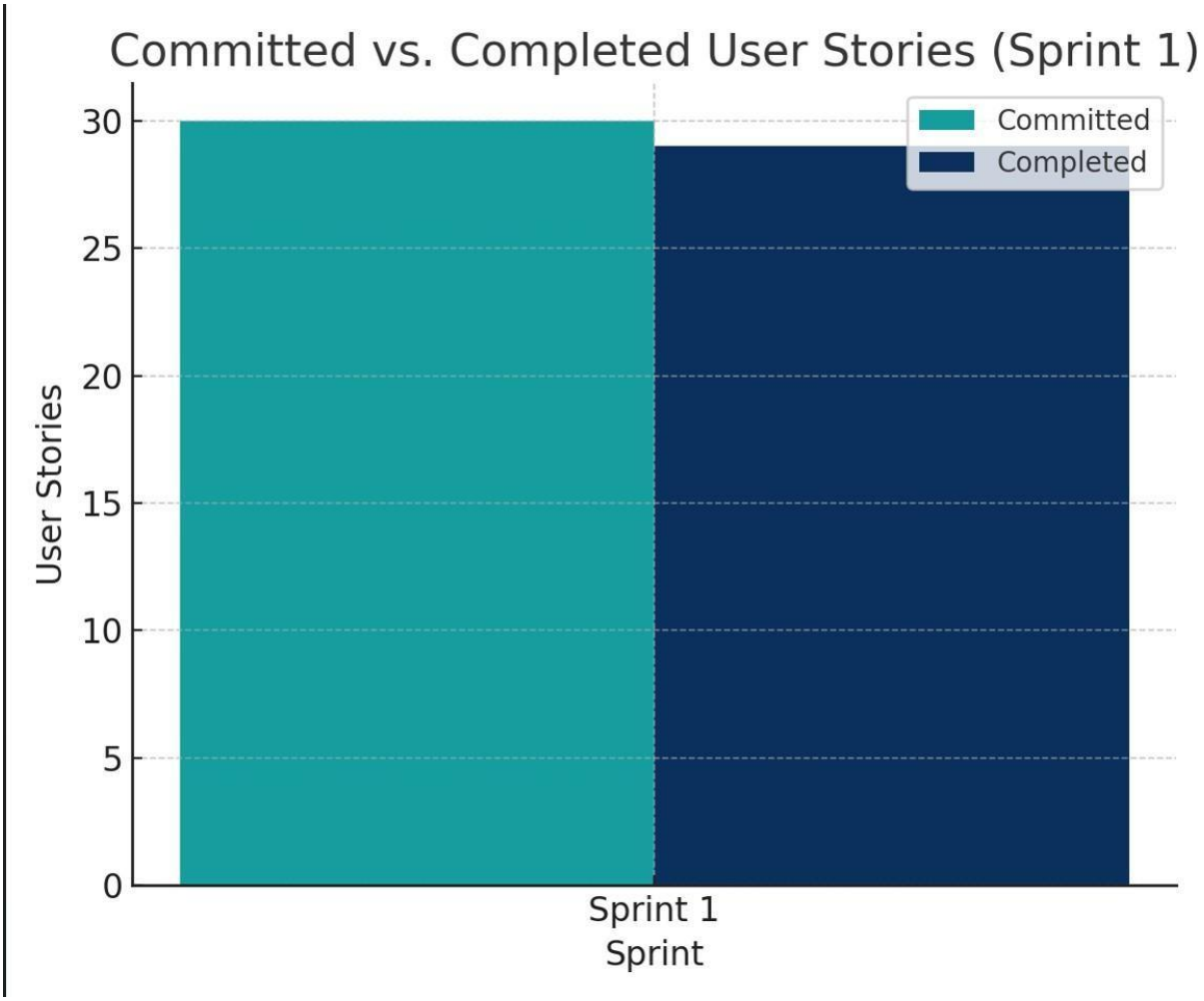


Figure 3.11 Bar graph for Committed Vs Completed User Stories

### 3.1.8 Sprint Retrospective

| | Sprint Retrospective | | |
|---|---|---|---|
| **What went well** | **What went poorly** | **What ideas do you have** | **How should we take action** |
| *The registration and login modules were implemented smoothly, with role-based redirection working as intended. Code compilation and quiz feedback features were* | *There were delays due to unclear documentation on third-party services (e.g., Judge0 API for code execution). Integration of the chatbot took longer than expected.* | *Allocate time early in the sprint for reviewing external service documentation. Use mock services for testing integrations before API deployment.* | *Schedule internal code reviews mid-sprint. Assign a dedicated team member for third-party integration research. Create reusable integration templates for future use* |
| Example: Code submissions were evaluated in real time, and feedback was correctly shown in the quiz result panel. | Example: Developers spent 2 extra days resolving API errors due to incomplete Judge0 documentation. | Example: Include 3–5 test prompts and expected outputs in the chatbot documentation. | Example: Prepare a reusable checklist for integrating APIs like GPT, Judge0, or mail notifications. |

Figure 3.12 Sprint Retrospective for the Sprint 1

## 3.2 SPRINT 2

### 3.2.1 Sprint Goal with User Stories of Sprint 2

The goal of Sprint 2 is to integrate the AI-powered chatbot into the student and faculty dashboards and to establish performance tracking and feedback features. This sprint focuses on enhancing interactivity, enabling real-time learning support, and providing a feedback mechanism to guide continuous improvement.

- As a student, I want to ask the chatbot questions about coding concepts or errors, so that I can get immediate help and keep learning without delay.
- As a student, I want to see visual feedback on my quiz and code submissions, so that I can understand how I'm improving over time.
- As an admin, I want to view platform-wide analytics and user activity logs, so that I can maintain order and system performance.
- As a user, I want my password and credentials to be encrypted, so that my account is safe from data breaches.

The following table 2.1 represents the detailed user stories of the sprint 2

| | |
|---|---|
| #US 5 | As a student, I want to ask the chatbot conceptual or debugging questions,<br>so that I can get instant AI-generated responses and continue learning without delay. |
| #US 6 | As a teacher, I want to post personalized feedback and view performance charts of my students,so that I can support their learning journey with actionable insights. |
| #US 7 | As an admin, I want to manage users, approve registrations, monitor quiz usage, and log user actions,so that the platform remains organized, secure, and compliant. |
| #US 8 | As a developer, I want to store user credentials using encryption and enforce role-based access queries,so that user data stays protected from unauthorized access and system breaches. |

Table 3.4 Detailed User Stories of sprint 2

## 3.2.2 Functional Document

3.2.2.1. Introduction

The Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask is a next-generation educational web platform designed to help students prepare effectively for technical interviews and coding assessments. The project brings together the power of artificial intelligence, real-time collaboration, and academic mentorship to offer an immersive, skill-driven learning experience. It caters to students by simulating real-world interview scenarios, offering AI-assisted doubt resolution, tracking performance over time, and encouraging peer-to-peer question sharing. Through dynamic role-based dashboards and secure access control, the platform also supports teachers and administrators in their roles as guides and facilitators of the learning process.

3.2.2.2. Product Goal

The primary goal of this platform is to empower students to excel in competitive programming and interview readiness through a personalized, intelligent, and collaborative environment. The platform seeks to:

- Provide personalized learning paths by analysing performance metrics such as quiz results, code accuracy, and response times.
- Enable real-time peer-to-peer interaction through community-posted questions and feedback mechanisms.
- Use AI chatbots to support real-time coding doubt resolution and generate non-coding questions for quizzes and self-evaluation.
- Support educators in monitoring student progress, managing content, and offering detailed, constructive feedback.
- Build a community-driven knowledge repository based on real interview questions shared by students.

3.2.2.3. Demography (Users, Location)

Users

- Target Users:
    - College and university students preparing for technical interviews.
    - Teachers and placement trainers.
    - Platform administrators overseeing usage and content quality.
- User Characteristics:
    - Students with varying technical abilities, from beginner to advanced.
    - Faculty with experience in curriculum delivery and mentoring.
    - Admins with IT management knowledge for platform governance.

Location

- Primary Focus:
    - Global, with an initial emphasis on regions with strong tech education ecosystems such as India, Southeast Asia, and North America.
- Target Institutions:
    - Engineering colleges, placement training centres, and online coding bootcamps.

3.2.2.4. Business Processes

User Registration and Authentication

- Users register via institutional email, and authentication is managed by the administrator.
- JWT or session-based login provides secure session handling.
- RBAC (Role-Based Access Control) ensures user permissions are aligned with their role.

Personalized Learning and Quiz Flow

- Students access recommended questions and quizzes based on previous performance.
- Teachers manage question banks, quizzes, and feedback.
- Students receive performance-based content suggestions.

Code Submission and Evaluation

- Code is submitted via an in-browser editor.
- The backend securely compiles and executes the code.
- Results, accuracy, and time/memory usage are displayed immediately.

AI Chatbot Interaction

- Chatbot assists in both student and faculty dashboards.
- Faculty use it to generate quiz questions from topic inputs.
- Students can clarify doubts in real-time using conversational input.

3.2.2.5. Features

This project focuses on implementing the following key features

Feature 1: Personalized Quiz & Code Recommendation System

Description:
The system analyses student quiz scores and coding trends to recommend suitable practice questions and challenges.

Userstory:
As a student, I want to receive personalized coding and quiz suggestions so that I can focus on areas where I need the most improvement.

Feature 2: Real-Time Code Compilation with Evaluation

Description:
Students write and execute code in-browser using an integrated editor. Their submissions are evaluated against instructor-defined test cases.

UserStory:

As a student, I want to run and test my code directly on the platform, so I can practice coding questions in a real interview-like setup.

Feature 3: AI-Powered Chatbot Assistance

Description:

An integrated chatbot helps users by answering queries, generating questions, and offering contextual support based on user inputs.

Userstory:

As a teacher, I want the chatbot to create non-coding questions from topic inputs so that I can prepare quizzes faster.

As a student, I want to ask doubts to the chatbot so I can get instant guidance without needing to wait for a teacher.

Feature 4: Role-Based Dashboards

Description:

Each user sees a dashboard tailored to their role—student, teacher, or admin—with only relevant modules and controls.

Userstory:

As a teacher, I want to monitor my students' performance and give feedback so I can help them preparebetter.

As an admin, I want to control user access and content quality so that the platform remains organized and secure.

Feature 5: Interview Experience Sharing

Description:

Students can post real interview questions from their placement drives, building a collaborative knowledge base.

Userstory:

As a student, I want to share questions I was asked in my interview so that others can prepare using real-world scenarios.

3.2.2.6. Authorization Matrix

| Role | Access Level |
|---|---|
| AI Chatbot Integration | Teacher uses chatbot to generate questions |
| Feedback and Monitoring | Teacher gives visual feedback |

Table 3.5 Access level Authorization Matrix

3.2.2.7. Assumptions

- The AI chatbot will use pretrained models like GPT, enhanced with prompt engineering to match educational scenarios.
- Developers will have access to cloud resources (e.g., Docker, Judge0 API, GPT API) for hosting, code execution, and AI services.
- Teachers and student testers will provide regular feedback during UAT (User Acceptance Testing).
- The system will comply with privacy regulations such as GDPR and India's IT Act, with encrypted credentials and secure data transfer.
- Users will have internet access with modern browsers to use the platform without installation requirements.

## 3.2.3 Architecture Document

3.2.3.1. Application

Microservices Architecture

The Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration is designed using a modular microservices architecture to ensure scalability, maintainability, and high performance. Each major function of the platform is encapsulated into independent services that communicate via RESTful APIs, allowing the system to scale individual components without affecting the overall platform.

- The platform is structured into independent services (e.g., authentication, quiz management, AI chatbot) that communicate via REST APIs, enabling modularity, scalability, and easy maintenance.
- A real-time code editor with backend execution (via Judge0 API or containerized environments) allows students to run code, receive instant feedback, and analyse performance metrics like speed and memory.
- Integrated AI chatbot provides real-time responses to student queries, generates quiz questions for teachers, and helps clarify coding logic using natural language processing.
- Teachers and students can create, categorize, and share questions. Quizzes can be timed and automatically evaluated, with data stored for analytics and feedback.
- Student progress is visualized using Chart.js graphs, enabling users to track improvements and allowing Teachers to provide individualized feedback.
- Admin users manage platform-wide settings, content, user roles, and system logs to maintain security, ensure user compliance, and monitor engagement metrics.
- JWT or session-based authentication ensures secure access. Role-Based Access Control (RBAC) distinguishes permissions for students, faculty, and administrators.

3.2.3.2 System Architecture-

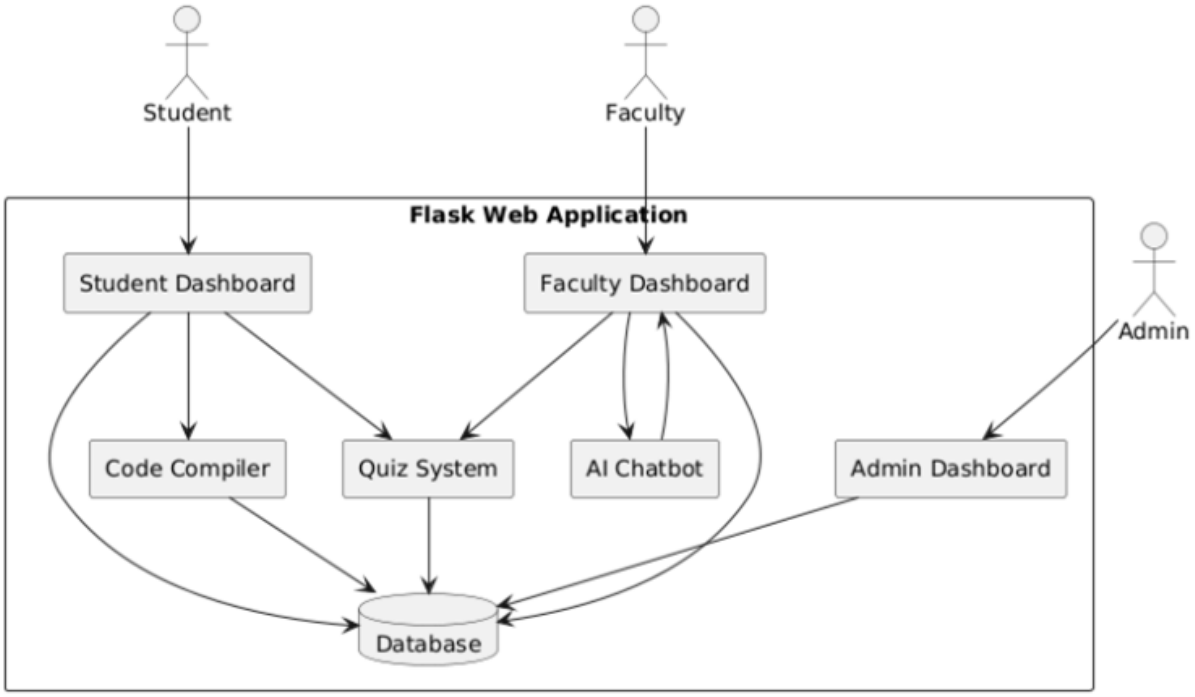


Figure 3.13 System Architecture Diagram

3.1.3.3. Data Exchange Contract:

Frequency of Data Exchanges

Data exchanges within the platform are optimized based on the criticality and responsiveness required for each operation:

- Real-TimeExchange:
  Real-time communication is used for essential user interactions such as:
  - User login and authentication
  - Quiz attempts and code submission evaluations
  - Chatbot queries and responses
  - Feedback delivery and notification triggers

- PeriodicSync:
  Scheduled synchronization is used for non-critical processes to improve performance:
  - User performance history aggregation
  - Quiz analytics reports
  - Usage statistics and platform-wide logs for admin dashboards
  - Backup of quiz/question repositories and student progress

Data Sets

The system operates on multiple types of data, each having specific exchange needs:

- User Data:
  Includes names, institutional email addresses, hashed passwords, role assignments, and activity metadata.
  Exchanged during:
  - Login
  - Registration
  - Role changes
  - Profile updates
- Question and Quiz Data:
  Consists of:
  - Coding/non-coding questions
  - Quiz structure and configuration
  - Difficulty levels, tags, time limits Exchanged during:
  - Quiz creation and update
  - Question attempts and evaluations
- Code Submission and Evaluation Data:
  Includes:
  - User code input

- Test case results
- Execution performance (time/memory) Exchanged during:
- Live compilation and evaluation cycles

- Chatbot Interaction Data:
  Covers:
  - Student/teacher queries
  - AI-generated question/answer logs
  - Sentiment or query categorization (for analytics) Exchanged during:
  - Chatbot session requests and responses

- Feedback and Performance Logs:
  Includes:
  - Teacher feedback
  - Student progress trends
  - Visual performance analytics (charts, scores) Exchanged during:
  - Dashboard loading and history retrieval

Mode of Exchanges (API, File, Queue, etc.)

Different data exchange mechanisms are used depending on the nature of the transaction:

- API (RESTful):
  RESTful APIs manage most real-time interactions between frontend and backend, such as:
  - User authentication and role validation
  - Quiz start, submission, and evaluation
  - AI chatbot integration and data return

- Message Queues (Asynchronous Tasks):
  Technologies like Celery + Redis, or RabbitMQ, are used to handle background and time-independent tasks:

- Notification delivery (e.g., "Feedback posted")
- Email confirmation messages
- System logs and performance tracking jobs
- Quiz timer triggers and submission auto-handling

- File-Based Exchanges:
  Used for:
  - Bulk upload of quiz banks or user lists
  - CSV exports of performance data
  - Backup of user-generated content or logs
    Typically handled through services like:

## 3.2.4 UI Design

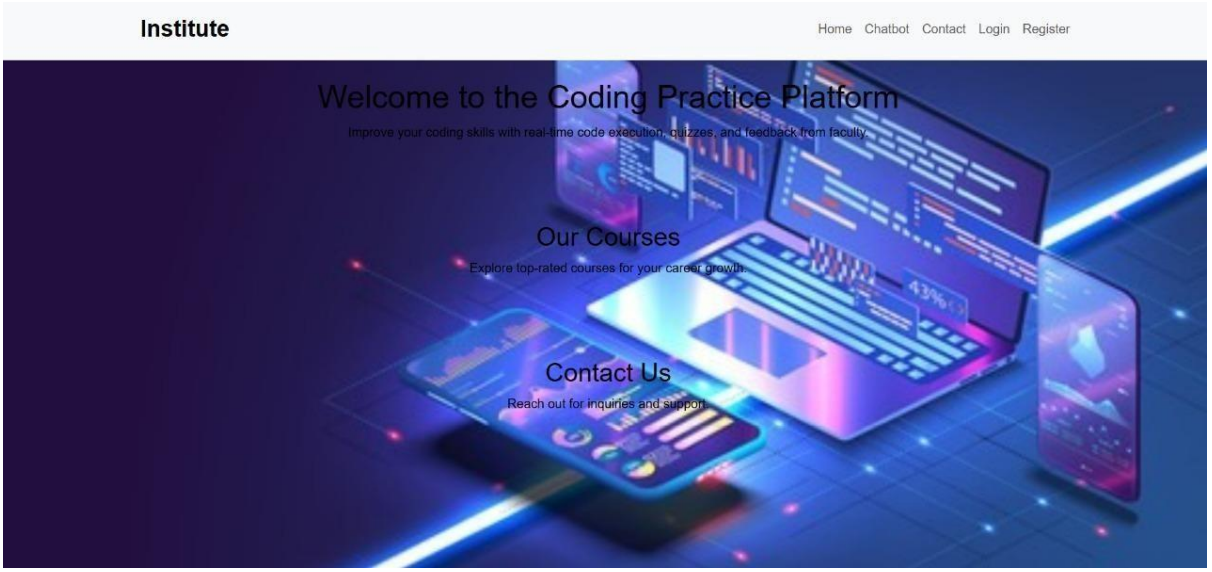


Figure 3.14 UI design for landing Page

## 3.2.5 Functional Test Cases

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 6 | AI Chatbot Interaction | Send Query to Chatbot | Login as any role.<br>Open chatbot.<br>Enter query and send. | The chatbot should respond with a relevant and helpful answer to the entered query. | Chatbot response is generated.<br>Displayed in the chat interface. | Pass | Ensure API response is timely. | |
| 7 | Admin Dashboard Access | Access Admin Dashboard | Login as admin.<br>Navigate to /admin-dashboard URL. | The admin dashboard should be displayed with administrative functionalities. | Admin dashboard is loaded su | Pass | Verify restricted access. | |
| 8 | Faculty Dashboard Access | Access Faculty Dashboar | Login as faculty.<br>Navigate to /faculty-dashboard URL. | The faculty dashboard should be displayed with quiz and student management features. | Faculty dashboard is accessible.<br>Quiz creation and student lists are visible. | Pass | Ensure only faculty functionalities appear. | |
| 9 | Student Dashboard Access | Access Student Dashboa | Login as student.<br>Navigate to /student-dashboard URL. | The student dashboard should be displayed with options to take quizzes and compile code. | Student dashboard is loaded.<br>Compiler and quiz access are available. | Pass | Validate access control. | |
| 10 | | | | | | | | |

Table 3.6 Detailed Functional Test Case

### 3.2.6 Daily Call Progress

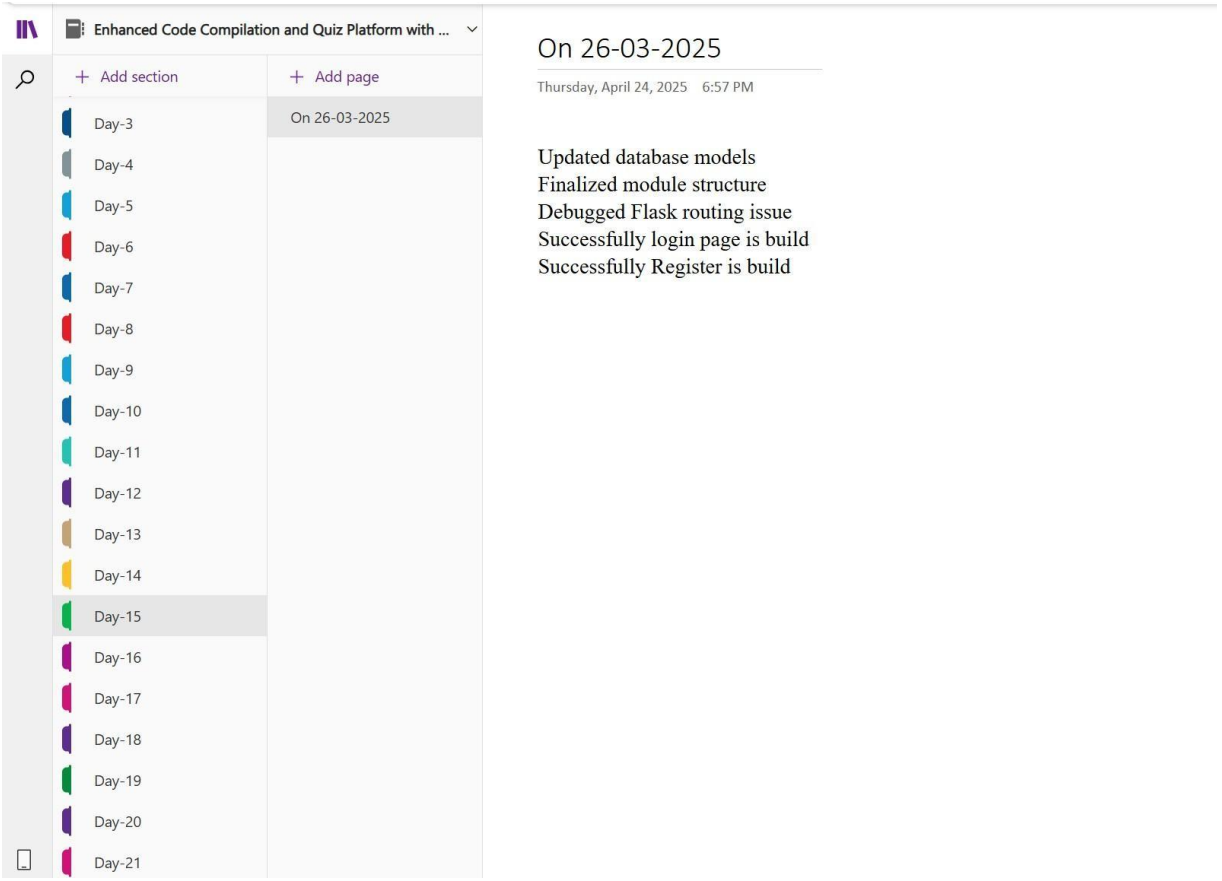


Figure 3.15 Standup meetings

### 3.2.7 COMMITTED Vs COMPLETED USER STORIES

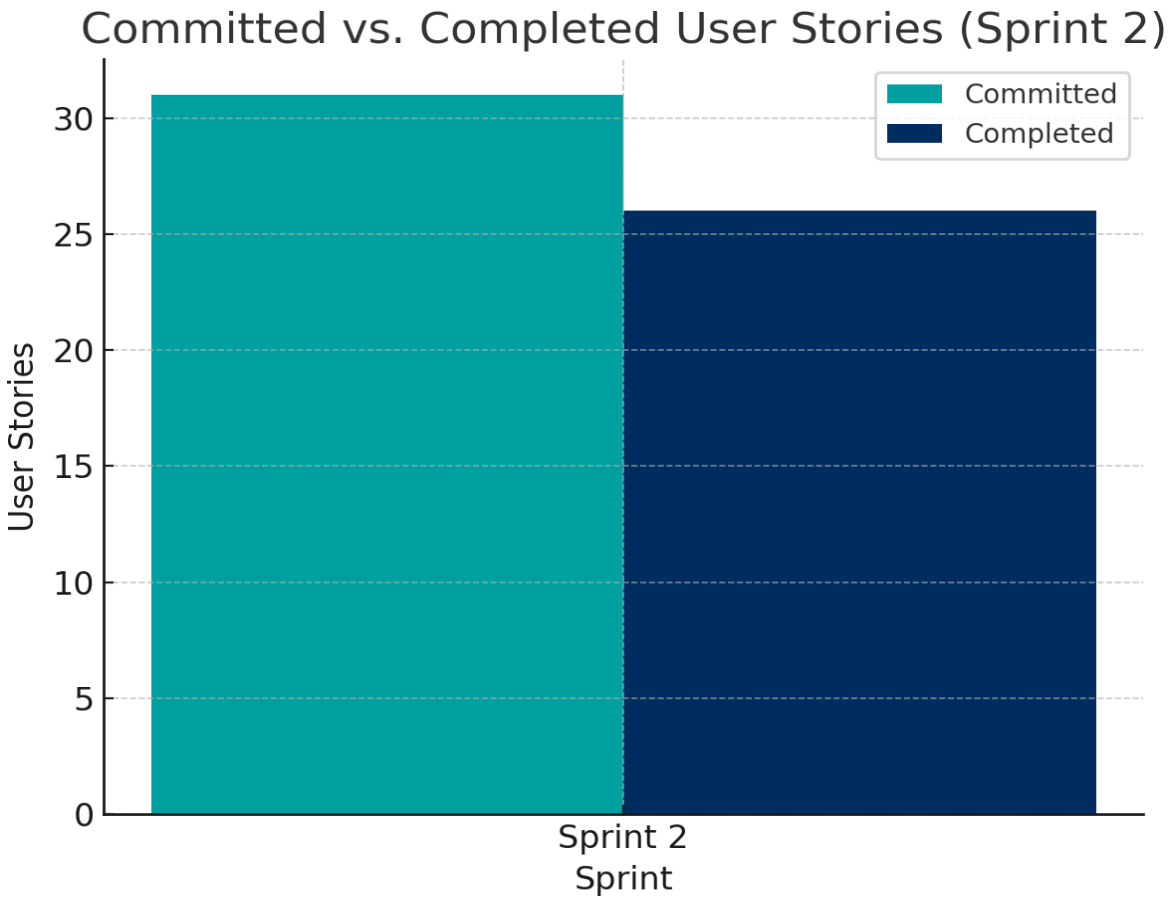


Figure 3.16 Bar graph for Committed Vs Completed User Stories

### 3.2.8 Sprint Retrospective

| | Sprint Retrospective | | | |
|---|---|---|---|---|
| | What went well | What went poorly | What ideas do you have | How should we take action |
| 4 | This section highlights the successes and positive outcomes The AI chatbot was successfully integrated and responded accurately to most coding-related questions. | The chatbot initially gave inconsistent answers to non-coding questions due to poor prompt tuning.<br>Feedback notifications were delayed due to | Improve prompt engineering by collecting FAQs and feeding better examples into chatbot prompts.<br>Implement retry logic and monitoring for feedback notification queue. | Allocate a mini sprint to tune chatbot prompts and integrate learning examples.<br>Optimize the async task queue (Celery/Redis) and introduce notification failure logs for better debugging. |

Figure 3.17 Sprint Retrospective for the Sprint 2

# CHAPTER 4

# RESULTS AND DISCUSSION

## 4.1 Project Outcomes

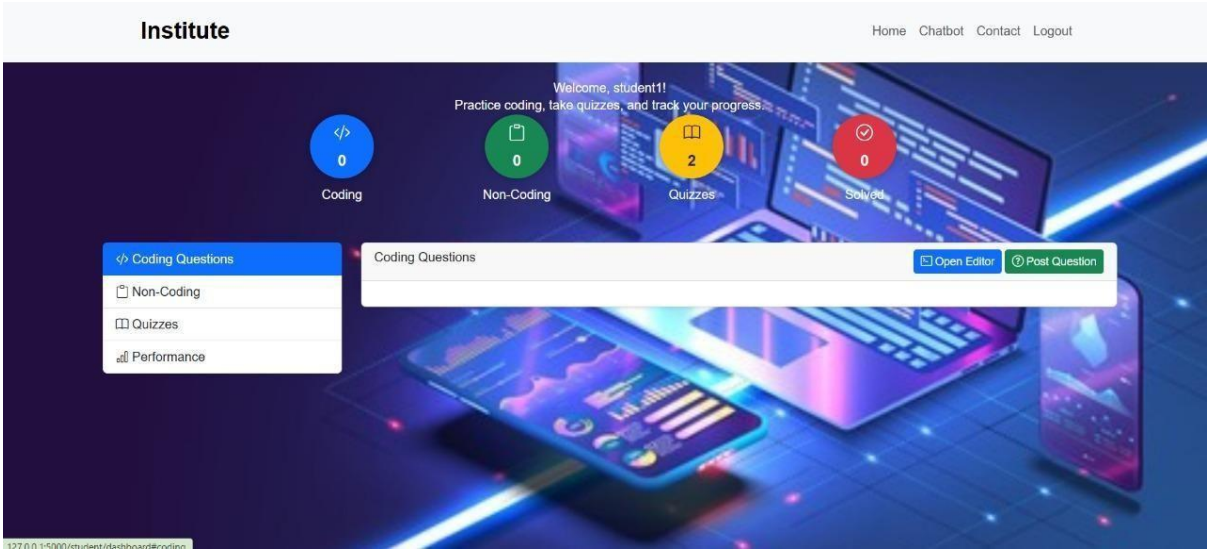


Figure 4.1 Student login Landing Page

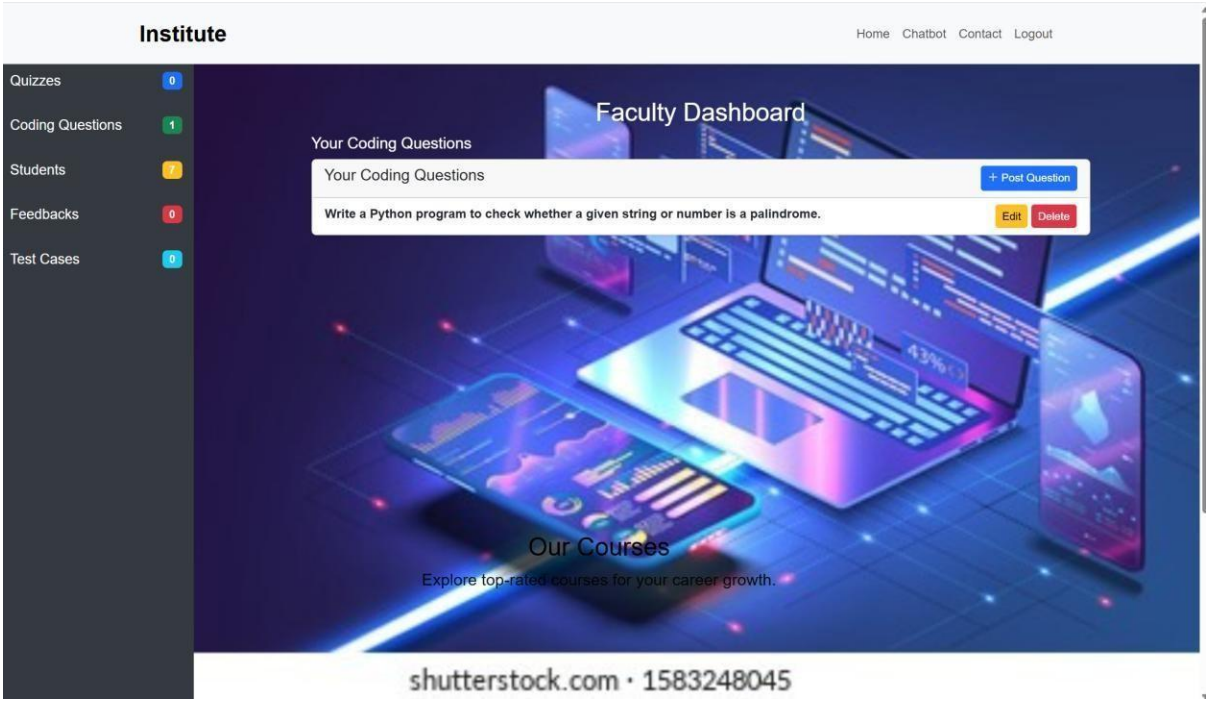


Figure 4.2 Faculty Login Landing Page

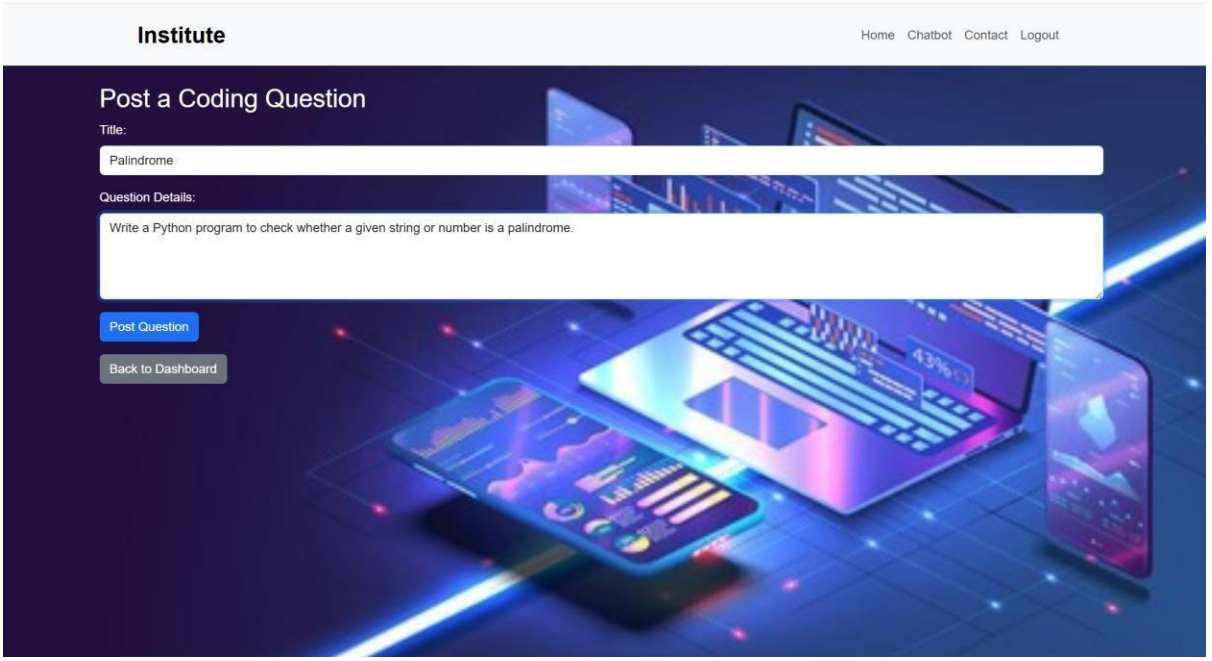


Figure 4.3 Faculty Posting Question

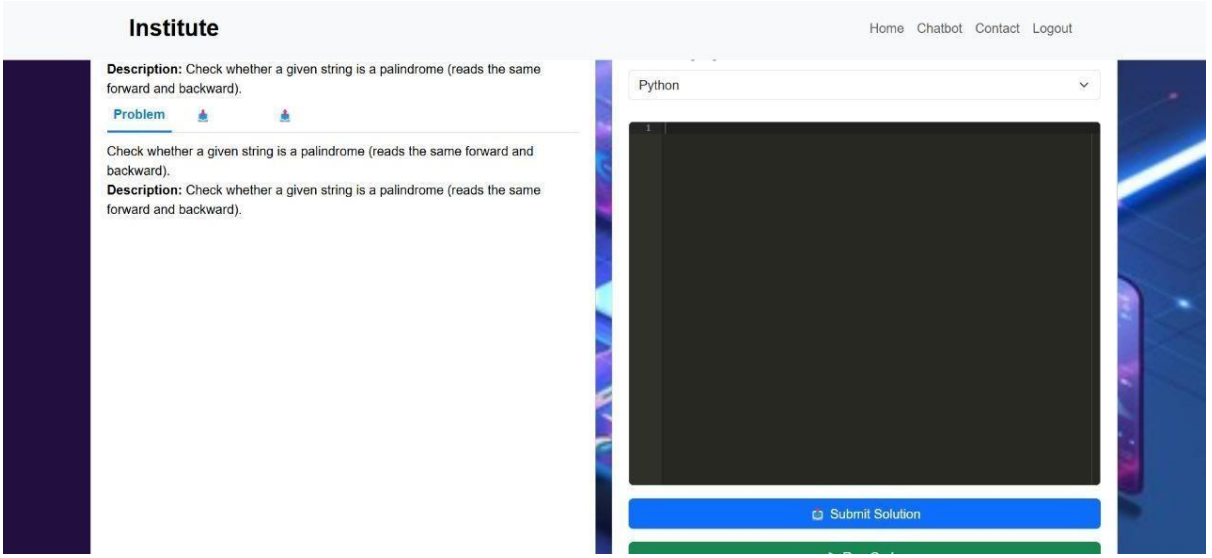


Figure 4.4 Student attempting Quiz

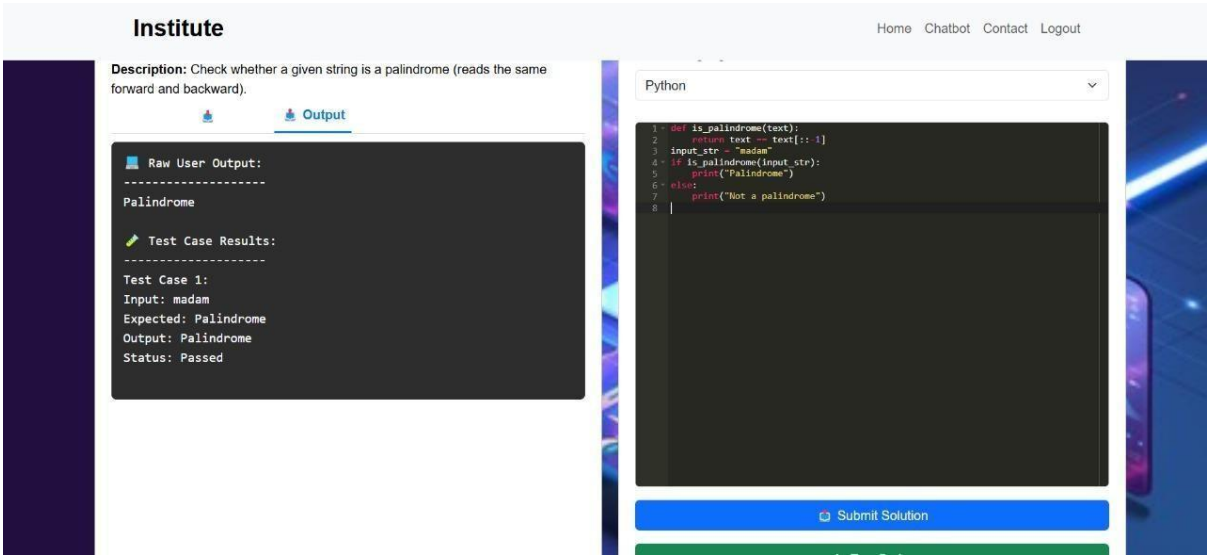


Figure 4.5 Program output

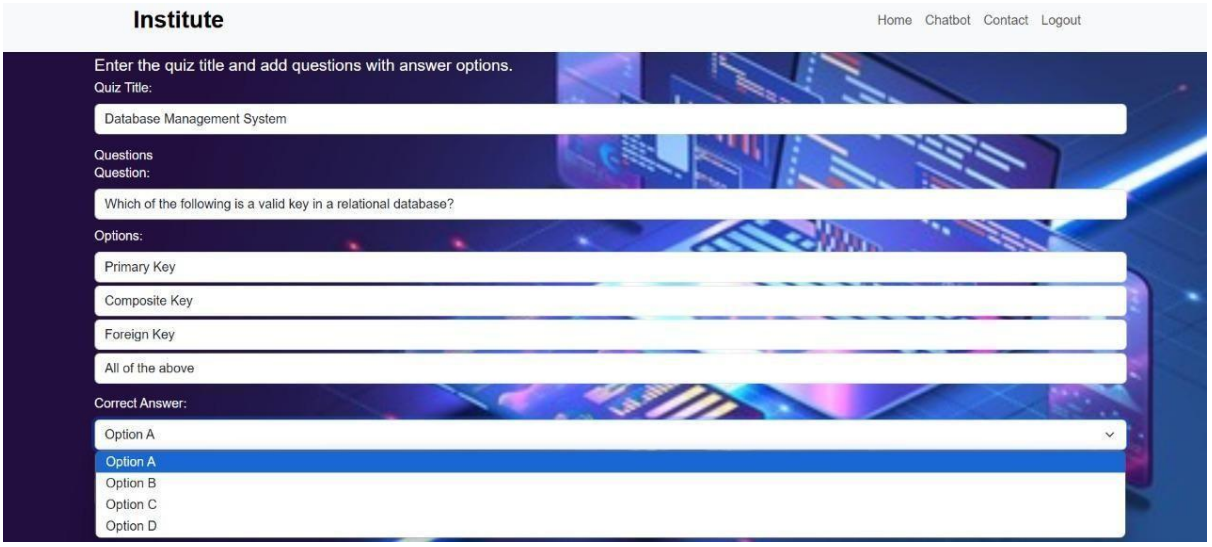


Figure 4.6 Faculty Posting Quiz

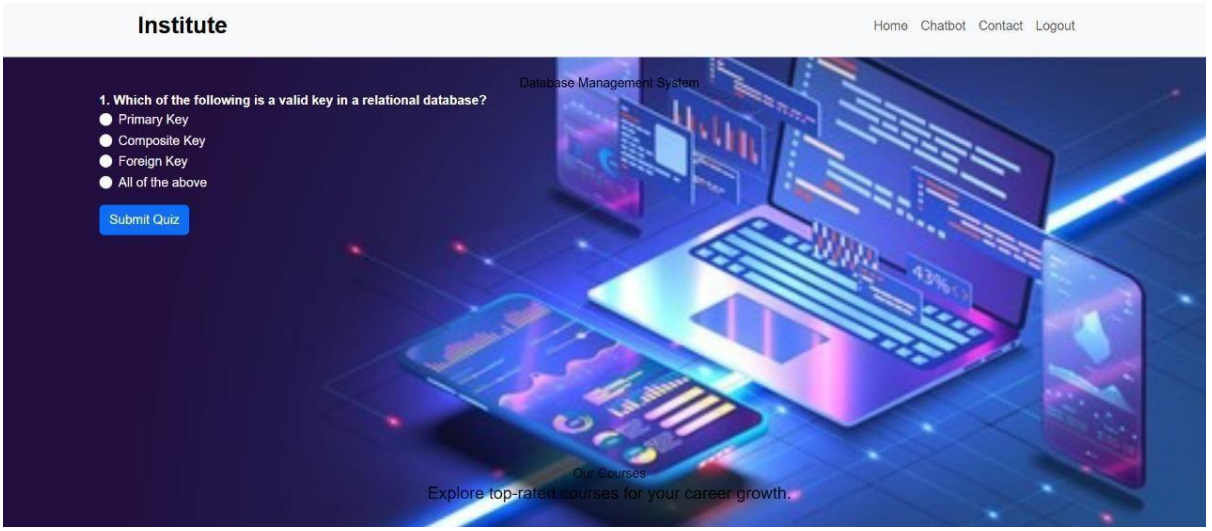


Figure 4.7 Studying attempting Quiz

## 4.2 Committed Vs Completed User stories

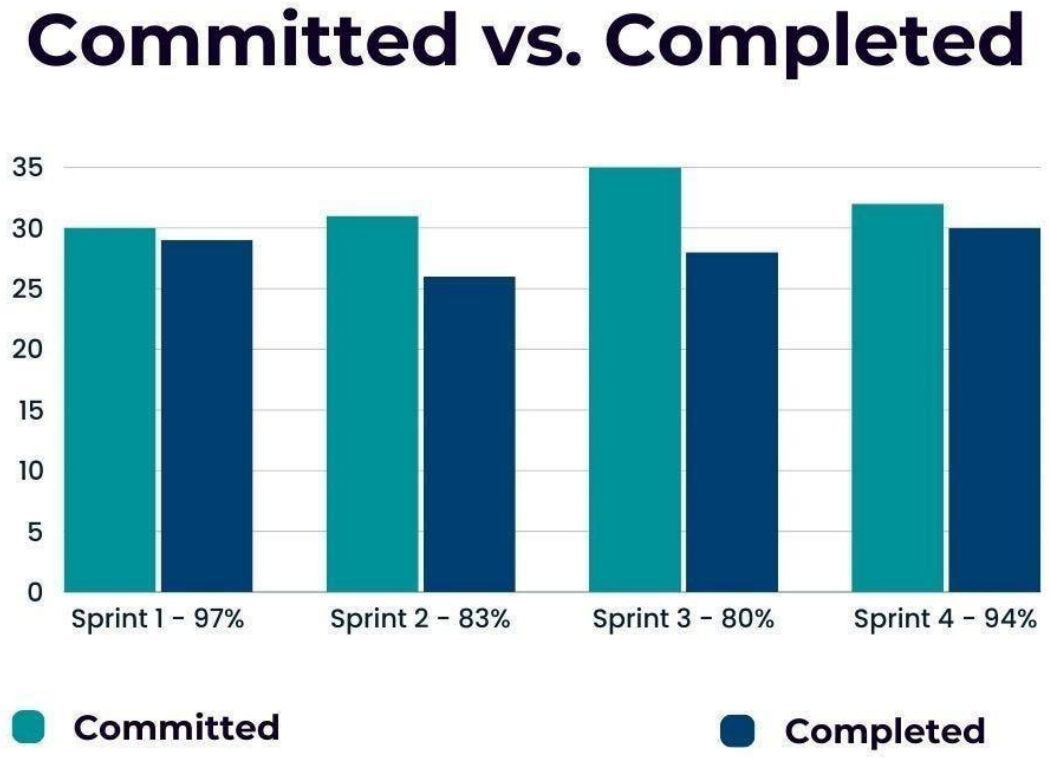


Figure 4.8 Bar graph for Committed Vs Completed User Stories

# CHAPTER 5

# CONCLUSION & FUTURE ENHANCEMENTS

The Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask represents a transformative step in bridging the divide between theoretical instruction and practical application in technical education. Built on a modular microservices architecture, the platform is robust, scalable, and designed to accommodate diverse educational needs. A standout feature is its real-time code editor and evaluation engine, which allows students to practice programming challenges and receive instant feedback on accuracy, execution time, and memory usage—mimicking real interview conditions. The quiz module complements this by enabling educators to design structured, auto-evaluated coding and non-coding assessments. The AI-powered chatbot enhances learning by acting as an on-demand coding assistant for students and a quiz-generation tool for teachers, making the platform more interactive and supportive.

Additionally, the platform's performance feedback system empowers students to visualize their progress through score trends and personalized teacher feedback, while instructors benefit from detailed analytics to guide learners more effectively. The centralized admin dashboard allows full oversight of content, user activity, and system integrity, secured through role-based access control (RBAC) and encrypted sessions. The platform not only encourages personalized and peer-driven learning but also ensures security and administrative scalability. In essence, it delivers an intelligent, adaptable, and holistic learning environment that is well-equipped to support students through the technical interview preparation process and beyond. Another key area of improvement lies in the integration of peer-review systems. Enabling students to review and rate each other's code solutions or quiz answers can foster collaborative learning and build a strong community within the platform From an academic analytics perspective, introducing machine learning models to analyse user performance trends, predict skill gaps, and recommend personalized learning paths would be a significant upgrade. These models could leverage historical data to offer adaptive learning experiences that go beyond static quiz content. Finally, integration with external platforms such as LinkedIn, GitHub, or coding platforms like LeetCode and HackerRank could allow students to showcase their progress, verified achievements, and project work to potential employers, there by enhancing their employability.

# REFERENECES

[1]S. G, R. S, A. S. R, P. Nallusamy, N. K and A. A. R, "Scrapetalk: Chatbot Conversation with Web Scraped Insights," 2024 International Conference on Integration of Emerging Technologies for the Digital World (ICIETDW), Chennai, India, 2024, pp. 1-5, doi: 10.1109/ICIETDW61607.2024.10940054.

[2]P. Duraisamy, G. C, J. A and S. R. S, "Implementation of AI-Powered Robotic Chatbot for Intelligent Customer Interaction," 2025 International Conference on Intelligent Control, Computing and Communications (IC3), Mathura, India, 2025, pp. 360-364, doi: 10.1109/IC363308.2025.10956525.

[3]L. S. Saoud and A. A. Romman, "AI-Powered Chatbots in the Telecommunications Industry in the UAE: Studying the Factors Impacting Consumer Adoption," 2024 15th International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, Republic of, 2024, pp. 1549-1553, doi: 10.1109/ICTC62082.2024.10827670.

[4]K. Patil, R. Patil, V. Koyande, A. S. Thakur and K. Kadam, "Analyzing Chatbot Architectures Utilising Deep Neural Networks," 2024 IEEE 6th International Conference on Cybernetics, Cognition and Machine Learning Applications (ICCCMLA), Hamburg, Germany, 2024, pp. 15-19, doi: 10.1109/ICCCMLA63077.2024.10871275.

[5]K. Balaji and P. S. Rao, "Artificial Intelligence and Chatbots: Transforming User Experience in E-tailing," 2024 9th International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2024, pp. 1375-1379, doi: 10.1109/ICCES63552.2024.10859474.

[6]P. Nazareth, G. B. Nikhil, G. Chirag and N. R. Prathik, "YouMatter: A Conversational AI Powered Mental Health Chatbot," 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kamand, India, 2024, pp. 1-7, doi: 10.1109/ICCCNT61001.2024.10725061.

[7]S. D. Bhavani Peri, S. Santhanalakshmi and R. Radha, "Chatbot to chat with medical books using Retrieval-Augmented Generation Model," 2024 IEEE North Karnataka Subsection Flagship International Conference (NKCon), Bagalkote, India, 2024, pp. 1-5, doi: 10.1109/NKCon62728.2024.10774900.

# APPENDIX

# A. PUBLICATION DETAILS



## Submission Summary

| | |
|---|---|
| **Conference Name** | 9th International Conference on Information and Communication Technology (CICT) |
| **Paper ID** | 47 |
| **Paper Title** | Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask |
| **Abstract** | Interview rejections are a frequent problem faced by students in placement drives, especially in the interview stage after clearing coding and aptitude tests. To counter this problem, an Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask has been designed. The platform allows students to share and solve coding and non-coding interview problems, supporting collaborative learning and real-time skill matching. Students are provided with performance-based feedback, such as correctness of code and time taken. There are three dedicated portals in the system: student, faculty, and admin. Faculty can upload problems, administer quizzes, view results, and provide feedback. An integrated AI chatbot supports the generation of non-coding problems and solves coding-related problems. Admin oversees all user activities and manages access permission. The innovative feature of this platform is its built-in AI-based support and post-interview question-sharing capability, supporting personalized learning and interview preparation. This system is intended to improve student preparedness and reduce interview-stage rejections. |
| **Created** | 4/23/2025, 11:47:51 PM |
| **Last Modified** | 4/23/2025, 11:47:51 PM |
| **Authors** | **Guvvala Venkat Reddy** (SRM Institute Of Science And Technology) <gr3327@srmist.edu.in> |
| **Primary Subject Area** | Digital Entrepreneurship Education |
| **Submission Files** | Enhanced_Code_Compilation_and_Quiz_Platform_with_AI_Powered_Chatbot_Integration_Using_Flask.pdf |





| | |
|---|---|
| | (1.1 Mb, 4/23/2025, 11:47:36 PM) |
| **Submission Questions Response** | **1. Turnitin/iThenticate**<br>*I authorize conference program chairs to use Turnitin/iThenticate to ensure the originality of written work before publication. I understand that this requires CMT to send an electronic copy of my submission to Turnitin/iThenticate. In addition, I hereby represent and warrant that I have all rights necessary to agree to such terms.*<br><br>Agreement accepted |

# B.CODE

admin.py

```
from functools import wraps

from flask import Blueprint, render_template, redirect, url_for, flash, request, jsonify, session
from flask_bcrypt import check_password_hash
from flask_login import login_required, current_user, login_user
from models import db, User, Quiz, Question, Submission, QuizSubmission

admin = Blueprint('admin', __name__)

def admin_required(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        if not current_user.is_authenticated or (current_user.role != 'admin' and not
current_user.is_admin):
            flash("Access denied! Only admins can access this page.", "danger")
            return redirect(url_for('auth.login'))
        return func(*args, **kwargs)

    return login_required(wrapper)

@admin.route("/admin-login", methods=["GET", "POST"])
def admin_login():
```

```
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]

        admin = User.query.filter_by(email=email, role="admin").first()
        if admin and check_password_hash(admin.password, password):
            login_user(admin)
            session["user_id"] = admin.id
            session["role"] = "admin"

            flash("Admin Login Successful!", "success")
            return redirect(url_for("admin.dashboard"))

        flash("Invalid Admin Credentials", "danger")

    return render_template("auth/admin_login.html")

@admin.route('/dashboard')
@admin_required
def dashboard():
    users = User.query.all()
    quizzes = Quiz.query.all()
    questions = Question.query.all()
    submissions = Submission.query.all()
```

```
    quiz_submissions = QuizSubmission.query.all()

    return render_template('admin/dashboard.html', users=users, quizzes=quizzes,
                           questions=questions, submissions=submissions,
                           quiz_submissions=quiz_submissions)


@admin.route('/delete-user/<int:user_id>', methods=['POST'])
@admin_required
def delete_user(user_id):
    user = User.query.get_or_404(user_id)
    db.session.delete(user)
    db.session.commit()
    flash("User deleted successfully!", "success")
    return redirect(url_for('admin.dashboard'))

@admin.route('/delete-quiz/<int:quiz_id>', methods=['POST'])
@admin_required
def delete_quiz(quiz_id):
    quiz = Quiz.query.get_or_404(quiz_id)
    db.session.delete(quiz)
    db.session.commit()
    flash("Quiz deleted successfully!", "success")
    return redirect(url_for('admin.dashboard'))
```

```
@admin.route('/delete-question/<int:question_id>', methods=['POST'])
@admin_required
def delete_question(question_id):
    question = Question.query.get_or_404(question_id)
    db.session.delete(question)
    db.session.commit()
    flash("Question deleted successfully!", "success")
    return redirect(url_for('admin.dashboard'))

@admin.route('/delete-submission/<int:submission_id>', methods=['POST'])
@admin_required
def delete_submission(submission_id):
    submission = Submission.query.get_or_404(submission_id)
    db.session.delete(submission)
    db.session.commit()
    flash("Submission deleted successfully!", "success")
    return redirect(url_for('admin.dashboard'))

@admin.route('/assign-roles')
@login_required
@admin_required
def assign_roles():
    pending_users = User.query.filter_by(role='pending').all()
    return redirect(url_for('admin.dashboard'))
```

```
@admin.route('/assign-role/<int:user_id>', methods=['POST'])
@login_required
@admin_required
def assign_role(user_id):
    user = User.query.get_or_404(user_id)
    role = request.form['role']
    if role in ['student', 'faculty']:
        user.role = role
        db.session.commit()
        flash(f"Role updated to {role}.", "success")
    else:
        flash("Invalid role selected.", "danger")
    return redirect(url_for('admin.dashboard'))
```

auth.py

```
from flask import Blueprint, render_template, redirect, url_for, flash, request, session, abort
from flask_login import login_user, logout_user, login_required
from app import db
from flask_bcrypt import check_password_hash, generate_password_hash

from models import User

auth = Blueprint('auth', __name__)
```

```
@auth.route('/')
def home():
    return render_template('home.html')


@auth.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = User.query.filter_by(email=email).first()

        if user and check_password_hash(user.password, password):
            if user.role == 'pending':
                flash("Your account is awaiting approval.", "warning")
                return redirect(url_for('auth.login'))
            login_user(user)

            if user.role == 'student':
                return redirect(url_for('student.dashboard'))
            elif user.role == 'faculty':
                return redirect(url_for('faculty.dashboard'))
            elif user.role == 'admin':
                return redirect(url_for('admin.dashboard'))
```

```
            flash("Invalid email or password!", "danger")
            print(user)

        return render_template('auth/login.html')

@auth.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        hashed_password = generate_password_hash(password).decode('utf-8')

        if User.query.filter_by(email=email).first():
            flash("Email already exists!", "danger")
            return redirect(url_for('auth.register'))

        new_user = User(username=username, email=email, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()
        flash("Account created successfully!", "success")
        return redirect(url_for('auth.login'))

    return render_template('auth/register.html')
```

```
@auth.route('/logout', methods=['POST', 'GET'])
@login_required
def logout():
    """ Logout user and clear session """
    logout_user()
    flash("Successfully Logged Out!", "success")
    return redirect(url_for("auth.login"))
```

Chat.py

```
from flask import Blueprint, request, jsonify, render_template
import requests
import re

chat = Blueprint('chat', __name__)
GEMINI_API_KEY = "AIzaSyC5Xf81pJlSNx2_IoHXR5U34HUEvqssfzs"
GEMINI_API_URL = f"https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key={GEMINI_API_KEY}"

@chat.route('/chat', methods=['GET'])
def chat_page():
    return render_template('chat.html')

@chat.route('/chat', methods=['POST'])
def ai_chat():
```

```
    user_msg = request.json.get('message')

    # Refined condition to apply MCQ formatting only on explicit quiz-type requests
    if any(x in user_msg.lower() for x in ["mcq", "multiple choice", "quiz", "4 options", "objective"]):
        user_msg = (
            f"Generate {user_msg}. "
            "Return each question and its 4 options (a–d) clearly and separately, in plain text format. "
            "Add a newline between each line for readability."
        )

    try:
        response = requests.post(
            GEMINI_API_URL,
            json={
                "contents": [
                    {
                        "parts": [{"text": user_msg}]
                    }
                ]
            }
        )
        data = response.json()
```

```
        if "candidates" in data and data["candidates"]:
            reply = data["candidates"][0]["content"]["parts"][0]["text"]
            formatted_reply = re.sub(r"(\d+\.\s)", r"\n\1", reply)
            formatted_reply = re.sub(r"([a-dA-D]\.)", r"\n\1", formatted_reply)
            formatted_reply = formatted_reply.strip()
        else:
            formatted_reply = "ı. Gemini returned no valid content. Raw response: " + str(data)

        return jsonify({"reply": formatted_reply})
    except Exception as e:
        return jsonify({"reply": "Error: " + str(e)})
```

Compiler.py

```
import json
from flask import Blueprint, request, jsonify, render_template
import requests
import time

from models import TestCase

compiler = Blueprint('compiler', __name__)

JUDGE0_API_URL = "https://judge0-ce.p.rapidapi.com"
HEADERS = {
    "X-RapidAPI-Key": "d289cc1b2dmshea218e2511ffa85p121067jsnbf5c89b32b88",
```

```
    "X-RapidAPI-Host": "judge0-ce.p.rapidapi.com",
    "Content-Type": "application/json"
}

LANGUAGE_MAP = {
    "python": 71,
    "java": 62,
    "c": 50,
    "cpp": 54
}

@compiler.route('/editor', methods=['GET'])
def code_editor():
    return render_template('code_editor.html')

@compiler.route('/compile', methods=['POST'])
def compile_code():
    """Compile and execute code against test cases using Judge0 API"""
    data = request.json

    print("Received JSON Data:", json.dumps(data, indent=4))

    required_keys = ['code', 'language_id', 'question_id']
    if not all(key in data for key in required_keys):
```

```
        return jsonify({"error": "Missing required fields: 'code', 'language_id', 'question_id'"}),
400

    code = data['code']
    language_id = str(data['language_id'])
    question_id = data['question_id']

    test_cases = TestCase.query.filter_by(question_id=question_id).all()
    if not test_cases:
        return jsonify({"error": "No test cases found for this question."}), 404

    results = []
    passed_all = True

    for test in test_cases:
        stdin_value = (test.input_data or "").strip()
        expected_output = (test.expected_output or "").strip()

        payload = {
            "language_id": language_id,
            "source_code": code,
            "stdin": stdin_value,
        }

        try:
```

```
        response =
requests.post(f"{JUDGE0_API_URL}/submissions?base64_encoded=false&wait=false",
                              json=payload, headers=HEADERS, timeout=10)

        if response.status_code != 201:
            return jsonify({"error": "Failed to submit code", "details": response.text}), 500

        token = response.json().get("token")
        if not token:
            return jsonify({"error": "No token received from Judge0", "details":
response.json()}), 500

        result_url = f"{JUDGE0_API_URL}/submissions/{token}?base64_encoded=false"
        for _ in range(5):
            time.sleep(1.5)
            result_response = requests.get(result_url, headers=HEADERS, timeout=10)
            result_data = result_response.json()

            if "status" in result_data and result_data["status"].get("id") in [3, 4, 6]:
                break

        output = (result_data.get("stdout") or "").strip()
        stderr = (result_data.get("stderr") or "").strip()
        compile_output = (result_data.get("compile_output") or "").strip()
```

```
            status_description = result_data.get("status", {}).get("description", "Unknown Status")

            final_output = output if output else stderr if stderr else compile_output if compile_output else "[No Output]"

            output_normalized = final_output.strip()
            expected_output_normalized = expected_output.strip()

            test_result = (output_normalized == expected_output_normalized)

            results.append({
                "input": stdin_value,
                "expected": expected_output,
                "output": final_output,
                "status": "Passed" if test_result else "Failed",
                "error": stderr or compile_output if not test_result else None,
                "execution_status": status_description
            })

            if not test_result:
                passed_all = False

        except requests.RequestException as e:
            return jsonify({"error": "Compiler Service Unavailable", "details": str(e)}), 503
```

```
    return jsonify({
        "status": "All Test Cases Passed" if passed_all else "Some Test Cases Failed",
        "results": results,
        "raw_output": results[0]["output"] if results else "[No Output]"
    })
```

faculty.py

```
import socketio
from flask import Blueprint, request, jsonify, render_template, flash, url_for, redirect, session
from flask_bcrypt import check_password_hash
from flask_login import login_required, current_user, login_user

from models import db, Feedback, User, Quiz, QuizQuestion, Question, Submission, QuizSubmission, TestCase
from app import socketio

faculty = Blueprint("faculty", __name__)

@faculty.route("/faculty-login", methods=["GET", "POST"])
def faculty_login():
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
```

```
        faculty = User.query.filter_by(email=email, role="faculty").first()
        if faculty and check_password_hash(faculty.password, password):
            login_user(faculty)
            session["user_id"] = faculty.id
            session["role"] = "faculty"

            flash("Faculty Login Successful!", "success")
            return redirect(url_for("faculty.dashboard"))

        flash("Invalid Faculty Credentials", "danger")

    return render_template("auth/faculty_login.html")

@faculty.route("/feedback", methods=["POST"])
def send_feedback():
    data = request.json
    feedback = Feedback(student_id=data["student_id"], faculty_id=data["faculty_id"],
message=data["message"])
    db.session.add(feedback)
    db.session.commit()

    socketio.emit("new_feedback", {"message": data["message"]}, broadcast=True)
    return jsonify({"message": "Feedback sent"})

@faculty.route("/give_feedback", methods=["POST"])
```

```
@login_required
def give_feedback():
    student_id = request.form['student_id']
    message = request.form['message']
    feedback = Feedback(student_id=student_id, faculty_id=current_user.id,
message=message)
    db.session.add(feedback)
    db.session.commit()
    flash("Feedback submitted successfully!", "success")
    return redirect(url_for('faculty.dashboard'))

@faculty.route('/dashboard')
@login_required
def dashboard():
    if session.get("role") != "faculty":
        flash("Unauthorized Access!", "danger")
        return redirect(url_for("auth.login"))
    quizzes = Quiz.query.filter_by(faculty_id=current_user.id).all()
    coding_questions = Question.query.filter_by(faculty_id=current_user.id,
question_type="Coding").all()
    quiz_questions = QuizQuestion.query.join(Quiz).filter(Quiz.faculty_id ==
current_user.id).all()
    students = User.query.filter_by(role='student').all()
    student_progress = {student.id: {"completed_quizzes": 0, "solved_questions": 0} for
student in students}
```

```
    progress_data = db.session.query(
        User.id,
        db.func.count(QuizSubmission.id).label("completed_quizzes"),
        db.func.count(Submission.id).label("solved_questions")
    ).outerjoin(QuizSubmission, QuizSubmission.student_id == User.id
                ).outerjoin(Submission, Submission.student_id == User.id
                            ).filter(User.role == 'student').group_by(User.id).all()
    for student_id, completed_quizzes, solved_questions in progress_data:
        student_progress[student_id]["completed_quizzes"] = completed_quizzes
        student_progress[student_id]["solved_questions"] = solved_questions
    print(coding_questions)
    print("Quizzes fetched:", quizzes)
    print("Coding questions fetched:", coding_questions)

    return render_template('faculty/dashboard.html',
                           quizzes=quizzes,
                           quiz_questions=quiz_questions,
                           coding_questions=coding_questions,
                           students=students,
                           student_progress=student_progress)

@faculty.route('/create-quiz', methods=['GET', 'POST'])
@login_required
def create_quiz():
```

```
if request.method == 'POST':
    title = request.form['title']
    new_quiz = Quiz(title=title, faculty_id=current_user.id)
    db.session.add(new_quiz)
    db.session.commit()

    questions = request.form.getlist('questions[]')
    correct_answers = request.form.getlist('correct_answers[]')

    for i, question_text in enumerate(questions):
        options = request.form.getlist(f'options_{i}[]')

        correct_answer_index = correct_answers[i]
        correct_answer_text = options[ord(correct_answer_index) - ord('A')]

        new_question = QuizQuestion(
            quiz_id=new_quiz.id,
            question_text=question_text,
            options=options,
            correct_answer=correct_answer_text
        )
        db.session.add(new_question)

    db.session.commit()
```

```
        flash("Quiz created successfully!", "success")
        return redirect(url_for('faculty.dashboard'))

    return render_template('faculty/quiz.html')

@faculty.route('/track-progress/<int:student_id>')
@login_required
def track_progress(student_id):
    student = User.query.get_or_404(student_id)

    completed_quizzes = db.session.query(Quiz, QuizSubmission.score).join(
        QuizSubmission, QuizSubmission.quiz_id == Quiz.id
    ).filter(QuizSubmission.student_id == student_id).all()

    solved_questions = db.session.query(Question, Submission.language).join(
        Submission, Submission.question_id == Question.id
    ).filter(Submission.student_id == student_id).all()

    return render_template('faculty/track_progress.html',
                           student=student,
                           completed_quizzes=completed_quizzes,
                           solved_questions=solved_questions)

@faculty.route('/quiz/<int:quiz_id>', methods=['GET'])
```

```
@login_required
def view_quiz(quiz_id):
    quiz = Quiz.query.get_or_404(quiz_id)
    questions = QuizQuestion.query.filter_by(quiz_id=quiz.id).all()
    print(f"Debug: Found {len(questions)} questions for quiz_id {quiz_id}")
    return render_template('faculty/view_quiz.html', quiz=quiz, questions=questions)

@faculty.route('/post-question', methods=['GET', 'POST'])
@login_required
def post_question():
    if request.method == 'POST':
        title = request.form.get('title')
        questions = request.form.get('content')

        if not title or not questions:
            flash("Title and question details are required!", "danger")
            return redirect(url_for('faculty.post_question'))

        new_question = Question(
            student_id=None,
            faculty_id=current_user.id,
            title=title,
            question_type="Coding",
            questions=questions
```

```
        )
        db.session.add(new_question)
        db.session.commit()
        flash("Coding question posted successfully!", "success")
        return redirect(url_for('faculty.dashboard'))

    return render_template('faculty/post_question.html')

@faculty.route('/view-student-code/<int:student_id>')
@login_required
def view_student_code(student_id):
    """View all coding submissions of a student."""
    student = User.query.get_or_404(student_id)
    submissions = Submission.query.filter_by(student_id=student_id).all()

    return render_template('faculty/view_student_code.html', student=student,
submissions=submissions)

@faculty.route('/view-student-quiz/<int:student_id>')
@login_required
def view_student_quiz(student_id):
    """View quiz results of a student."""
    student = User.query.get_or_404(student_id)
    quiz_results = QuizSubmission.query.filter_by(student_id=student_id).all()
```

```
    return render_template('faculty/view_student_quiz.html', student=student,
quiz_results=quiz_results)


@faculty.route('/edit-question/<int:question_id>', methods=['GET', 'POST'])
@login_required
def edit_question(question_id):
    question = Question.query.get_or_404(question_id)

    if question.faculty_id != current_user.id:
        flash("You can only edit your own questions.", "danger")
        return redirect(url_for('faculty.dashboard'))

    if request.method == 'POST':
        question.title = request.form['title']
        question.questions = request.form['content']
        db.session.commit()
        flash("Question updated successfully!", "success")
        return redirect(url_for('faculty.dashboard'))

    return render_template('faculty/edit_question.html', question=question)


@faculty.route('/edit-quiz-question/<int:quiz_question_id>', methods=['GET', 'POST'])
@login_required
def edit_quiz_question(quiz_question_id):
    quiz_question = QuizQuestion.query.get_or_404(quiz_question_id)
```

```
    if quiz_question.quiz.faculty_id != current_user.id:
        flash("You can only edit your own quiz questions.", "danger")
        return redirect(url_for('faculty.dashboard'))

    if request.method == 'POST':
        quiz_question.question_text = request.form['question_text']
        quiz_question.correct_answer = request.form['correct_answer']
        quiz_question.options = request.form.getlist('options[]')
        db.session.commit()
        flash("Quiz question updated successfully!", "success")
        return redirect(url_for('faculty.dashboard'))

    return render_template('faculty/edit_quiz_question.html', quiz_question=quiz_question)

@faculty.route('/delete-quiz/<int:quiz_id>', methods=['POST'])
@login_required
def delete_quiz(quiz_id):
    quiz = Quiz.query.get_or_404(quiz_id)

    if quiz.faculty_id != current_user.id:
        flash("You can only delete your own quizzes.", "danger")
        return redirect(url_for('faculty.dashboard'))
```

```
    db.session.delete(quiz)
    db.session.commit()
    flash("Quiz deleted successfully!", "success")
    return redirect(url_for('faculty.dashboard'))

@faculty.route('delete-question/<int:question_id>', methods=['POST'])
@login_required
def delete_question(question_id):
    question = Question.query.get_or_404(question_id)
    if question.faculty_id != current_user.id:
        flash("You can only delete your own quizzes.", "danger")
        return redirect(url_for('faculty.dashboard'))
    db.session.delete(question)
    db.session.commit()
    flash("Quiz deleted successfully!", "success")
    return redirect(url_for('faculty.dashboard'))

@faculty.route('/faculty/question/<int:question_id>/testcases', methods=['GET', 'POST'])
def add_testcase(question_id):
    question = Question.query.get_or_404(question_id)
    coding_questions = Question.query.all()

    if request.method == 'POST':
        input_data = request.form['input_data'].strip()
```

```
            expected_output = request.form['expected_output'].strip()

            new_test = TestCase(
                question_id=question.id,
                input_data=input_data,
                expected_output=expected_output
            )
            db.session.add(new_test)
            db.session.commit()
            flash('Test Case added successfully!', 'success')
            return redirect(url_for('faculty.add_testcase', question_id=question.id))

        test_cases = TestCase.query.filter_by(question_id=question.id).all()

        return render_template(
            'faculty/manage_testcases.html',
            test_cases=test_cases,
            coding_questions=coding_questions,
            selected_question=question
        )

@faculty.route('/faculty/testcases/<int:question_id>', methods=['GET', 'POST'])
def manage_testcases(question_id):
    """ Add and manage test cases for a specific question """
```

```
    question = Question.query.get_or_404(question_id)

    if request.method == 'POST':
        input_data = request.form['input_data'].strip()
        expected_output = request.form['expected_output'].strip()

        new_testcase = TestCase(question_id=question_id, input_data=input_data,
expected_output=expected_output)
        db.session.add(new_testcase)
        db.session.commit()

        flash('Test Case Added Successfully!', 'success')
        return redirect(url_for('faculty.manage_testcases', question_id=question_id))

    test_cases = TestCase.query.filter_by(question_id=question_id).all()

    return render_template(
        'faculty/manage_testcases.html',
        question=question,
        test_cases=test_cases,
        coding_questions=Question.query.all()
    )

@faculty.route('/faculty/testcases/delete/<int:testcase_id>', methods=['DELETE'])
def delete_testcase(testcase_id):
```

```
    """ Delete a specific test case """
    testcase = TestCase.query.get_or_404(testcase_id)
    question_id = testcase.question_id
    db.session.delete(testcase)
    db.session.commit()

    flash('🗑 Test Case Deleted Successfully!', 'danger')
    return jsonify({"message": "Test Case Deleted"}), 200
```

**Models.py**

```
import json

from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from datetime import datetime

db = SQLAlchemy()

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(256), nullable=False)
    role = db.Column(db.String(10),  default='pending')
```

```
    is_admin = db.Column(db.Boolean, default=False)

    @property
    def is_real_admin(self):
        return self.role == 'admin' or self.is_admin

class Question(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=True)
    faculty_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=True)
    title = db.Column(db.String(255), nullable=False)
    question_type = db.Column(db.String(20), nullable=False)
    questions = db.Column(db.Text, nullable=False)
    related_quiz_id = db.Column(db.Integer, db.ForeignKey('quiz.id'), nullable=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    student = db.relationship('User', foreign_keys=[student_id], backref='questions')
    faculty = db.relationship('User', foreign_keys=[faculty_id], backref='faculty_questions')
    related_quiz = db.relationship('Quiz', backref='questions')

class Submission(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    question_id = db.Column(db.Integer, db.ForeignKey('question.id'))
    student_id = db.Column(db.Integer, db.ForeignKey('user.id'))
```

```
    code = db.Column(db.Text)
    language = db.Column(db.String(50))
    output = db.Column(db.String(2500))
    execution_time = db.Column(db.Float)
    efficiency = db.Column(db.Float)
    submitted_at = db.Column(db.DateTime, default=datetime.utcnow)

    student = db.relationship("User", backref="submissions")
    question = db.relationship("Question", backref="submissions")

class Feedback(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    faculty_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    message = db.Column(db.Text, nullable=False)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)

    student = db.relationship('User', foreign_keys=[student_id], backref='received_feedback')
    faculty = db.relationship('User', foreign_keys=[faculty_id], backref='given_feedback')

class Quiz(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    faculty_id = db.Column(db.Integer, db.ForeignKey('user.id'))
    title = db.Column(db.String(255), nullable=False)
```

```
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    faculty = db.relationship("User", backref="quizzes")


class QuizQuestion(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    quiz_id = db.Column(db.Integer, db.ForeignKey('quiz.id'))
    question_text = db.Column(db.Text, nullable=False)
    options = db.Column(db.JSON)
    correct_answer = db.Column(db.String(100), nullable=False)
    quiz = db.relationship('Quiz', backref='quiz_questions')

    def get_options(self):
        """Return options as a list."""
        return json.loads(self.options) if isinstance(self.options, str) else self.options

class QuizSubmission(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    quiz_id = db.Column(db.Integer, db.ForeignKey('quiz.id'), nullable=False)
    score = db.Column(db.Integer, nullable=False)
    submitted_at = db.Column(db.DateTime, default=datetime.utcnow)

    student = db.relationship("User", backref="quiz_submissions")
```

```
    quiz = db.relationship("Quiz", backref="quiz_submissions")


class TestCase(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    question_id = db.Column(db.Integer, db.ForeignKey('question.id'), nullable=False)
    input_data = db.Column(db.Text, nullable=False)
    expected_output = db.Column(db.Text, nullable=False)

    question = db.relationship('Question', backref=db.backref('test_cases', lazy=True))
```

# B. PLAGIARISM REPORT

## Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask.pdf

My Files
My Files
Indian Institute of Management Calcutta

### Document Details

Submission ID
trn:oid:::30493:92409323

Submission Date
Apr 22, 2025, 9:46 PM GMT+5:30

Download Date
Apr 22, 2025, 9:48 PM GMT+5:30

File Name
Enhanced Code Compilation and Quiz Platform with AI-Powered Chatbot Integration Using Flask.pdf

File Size
1.0 MB

7 Pages
4,517 Words
27,587 Characters

# 2% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

## Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text

## Match Groups

- **7** Not Cited or Quoted 2%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

## Top Sources

1% Internet sources
0% Publications
2% Submitted works (Student Papers)

## Integrity Flags

**0 Integrity Flags for Review**

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

# 0% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

**Caution: Review required.**

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

## Detection Groups

**0** AI-generated only 0%
Likely AI-generated text from a large-language model.

**0** AI-generated text that was AI-paraphrased 0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

**Disclaimer**
Our AI writing assessment is designed to help educators identify text that might be prepared by a generative AI tool. Our AI writing assessment may not always be accurate (it may misidentify writing that is likely AI generated as AI generated and AI paraphrased or likely AI generated and AI paraphrased writing as only AI generated) so it should not be used as the sole basis for adverse actions against a student. It takes further scrutiny and human judgment in conjunction with an organization's application of its specific academic policies to determine whether any academic misconduct has occurred.

## Frequently Asked Questions

**How should I interpret Turnitin's AI writing percentage and false positives?**
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

**What does 'qualifying text' mean?**
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.