🥰 Experiment 1

Software Requirements Specification

for

Quiz Master: A Multi-User Exam Preparation and Quiz Management Application

Prepared by
Batch: B

Riya Suryawanshi	**2023300240**	**riya.suryawanshi23@spit.ac.in**
Anushka Suvarna	**2023300241**	**anushka.suvarna23@spit.ac.in**

Instructor: Dr. Prasenjit Bhavathankar

Course: Software Engineering

Date: 18-08-2025

INDEX

1 Introduction

1.1 Document Purpose

This **Software Requirements Specification (SRS)** document specifies all the features for Quiz Master, a multi-user and multi-role exam preparation platform. The document covers the complete system, with functionalities like **CRUD** features for subjects, chapters, quizzes, and questions, a **feature of automatic quiz generation** from uploaded PDF content, performance tracking for users, and sending reminder emails for quizzes.

This SRS describes the entire Quiz Master system, which supports two roles: **Admin (for generating quizzes)** and **User (For taking quizzes)**, having role-based components and screens, backend services, database management, asynchronous background tasks, and the integration of artificial intelligence for automated quiz generation. The intended audience for this document includes developers, testers, instructors, and maintainers of the system.

1.2 Product Scope

Quiz Master is a **full-stack web application** that is designed to help in exam preparation for students and assessment management for educational institutions and training organizations. The platform supports role-based access with distinct functionalities for **admins (Quiz Masters)** and **end-users (students/learners).**

The software enables admins to create and manage study content, including subjects, chapters, quizzes, and questions, while providing users with an easy-to-use interface for quiz taking and performance tracking. One of the key features of the software includes the use of AI to automatically create quizzes by uploading textbook content for subjects. This will significantly reduce the admin's burden of manually creating quizzes and their questions. Additionally, it will ensure a broad coverage of chapters and result in better quiz generation. The software also incorporates **background task management using Celery**, heavy operations such as sending daily reminders, and monthly performance reports through email to be executed asynchronously without affecting the system's responsiveness. This ensures that the core quiz-taking experience remains fast and uninterrupted even during heavy workloads. In addition, the system provides user performance summary graphs that visually present progress, strengths, and areas for improvement. Consequently, admins can utilize it to observe patterns in the users' performance. This platform benefits educational institutions and students by providing a comprehensive study application that can accommodate multiple subjects and hundreds of users simultaneously.

1.3 Intended Audience and Document Overview

This document is intended for a diverse set of audiences, each with a different purpose. The **clients** can use it to understand the system's objectives and features. The **end users** can use it for exam preparation. The **professors and evaluators** can assess the completeness of the requirements, technical feasibility, and clarity of the system specification. The **developers** will benefit from detailed insights into functional and non-functional requirements, database design, backend services, and integration points such as Celery tasks and the AI module. Similarly, the

testers can derive test cases, validate functionality, and ensure that both quiz management and AI-powered features align with the specified requirements. Finally, the **project managers and documentation writers** may use this document as a reference for defining scope, setting milestones, and tracking the system.

The rest of this SRS is organized into structured sections for clarity:

- **Section 1:** Introduction, which provides the purpose, scope, audience, and an overview of the document.
- **Section 2:** Overall Description, which describes the product perspective, features, and constraints.
- **Section 3:** Specific Requirements, which specifies user interfaces, APIs, hardware interfaces, and functional requirements.
- **Section 4:** Other Non-Functional Requirements, which defines performance, security, and scalability expectations.

1.4 Definitions, Acronyms, and Abbreviations

Term/Acronym	Definition
Admin	The administrator (Quiz Master) who manages subjects, chapters, quizzes, and users.
User	A registered participant who attempts quizzes and views scores.
Dashboard	Role-specific interface (AdminDashboard or UserDashboard).
API	Application Programming Interface.
CRUD	Create, Read, Update, Delete operations.
Flask	Python web framework used for backend APIs.
Celery	Distributed task queue for background jobs.
Redis	In-memory store used for caching and task management.
JWT	JSON Web Token for authentication.
SQLAlchemy	ORM library for database interaction.
PostgreSQL	SQL-based database.
MCQ	Multiple Choice Question.
REST	Representational State Transfer (API style).

UI/UX	User Interface / User Experience.

1.5 Document Conventions

This SRS document adheres to IEEE Std 830-1998 formatting standards with the following specific conventions:

- Font: Arial, size 11 points for body text
- Margins: 1-inch margins on all sides
- Line spacing: Single-spaced within sections
- Section headers: Bold, hierarchical numbering (1, 1.1, 1.1.1)
- Emphasis: *Italics* for comments and notes, **Bold** for important terms

1.6 References and Acknowledgments

1.6.1 Technical References:

- IEEE Computer Society, "IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications," IEEE Standards Association, 1998.
- Flask Development Team, "Flask Documentation," Available: https://flask.palletsprojects.com/, Accessed: January 2025.
- Vue.js Team, "Vue.js Guide," Available: https://vuejs.org/guide/, Accessed: January 2025.
- OpenAI, "GPT API Documentation," Available: https://platform.openai.com/docs/, Accessed: January 2025.
- Celery Project, "Celery: Distributed Task Queue," Available: https://docs.celeryproject.org/, Accessed: January 2025.

1.6.2 Academic References:

- Pressman, R.S., "Software Engineering: A Practitioner's Approach," 8th Edition, McGraw-Hill, 2014.

1.6.3 Acknowledgments:

- OpenAI ChatGPT for technical guidance in JWT authentication, Celery implementation, and AI integration strategies
- Flask and Vue.js communities for comprehensive documentation and support resources

2 Overall Description

2.1 Product Perspective

Quiz Master is a new, self-contained product designed as an independent quiz management platform. It is not a direct replacement of any existing system, but it draws inspiration from commonly used online quiz and assessment applications such as Google Forms, Mentimeter, and Moodle quizzes. Unlike these general-purpose platforms, Quiz Master is designed to support structured exam preparation with **multi-user and multi-role functionality**. The product includes separate dashboards for admins and regular users, automated background tasks for reminders and reports, and graphical performance summaries.

The system functions as a **standalone platform**, accessible through a web browser, with its own backend, frontend, and database components. It interacts with the environment only through standard web interfaces (HTTP/HTTPS) and email notifications. This makes it flexible enough to be deployed for **educational institutions, training organizations, or corporate learning setups**, without needing to be embedded into a larger system.

2.2 Product Functionality

The Quiz Master platform provides comprehensive examination and assessment management functionalities, which are listed below:

- Secure **login/register** with role-based access. Admin has full access for CRUD operations of subjects and quizzes. The role of end users is clearly stated in the databases.
- **Admin Dashboard** has subjects, chapters, quizzes, and questions (MCQs). Search users/subjects/quizzes.
- **User Dashboard** enables users to attempt quizzes with timers only once and only on their scheduled date. Also display performance statistics through graphs and charts
- Background jobs of **Daily Reminders via mail** for inactive users or new quizzes, monthly Reports summarizing user performance, sent via email, and **CSV Exports for all-user performance for the admin**
- **Adding caching** for frequently accessed data (e.g., quiz lists, results) and implementing cache expiry policies to maintain consistency and monitor API performance
- **Automatic quiz generation** by uploading a textbook PDF
- **Anti-cheating measures** for screenshot taking and copy-pasting prevention

2.3 Users and Characteristics

The Quiz Master system is designed for two primary users: Quiz Masters/Admins and Students/Learners.

2.3.1 Quiz Masters / Administrators

- Frequency of Use: Daily during active quiz periods, weekly for maintenance.
- Primary Functions: Creating and managing quiz content, keeping a check on the registered users, and monitoring performance.
- Special Requirements: Advanced dashboard access, email tasks.

- Access Level: Full system privileges.

2.3.2 Students / Learners

- Frequency of Use: Regular during examination periods, occasional for practice.
- Primary Functions: Taking quizzes, reviewing scores, and tracking performance.
- Special Requirements: Intuitive and mobile-friendly interface with accessibility support.
- Access Level: Limited to personal data and assigned quizzes.

2.4 Operating Environment

Quiz Master is designed to run in a modern web-based environment, suitable for both local development and deployment on cloud servers. The system ensures smooth operation through lightweight components, making it deployable even on modest infrastructure while remaining scalable for production.

- **Server Environment:**

  The Quiz Master system runs on Windows 10/11 for development and supports production deployment with Python 3.11+, Flask (RESTful APIs), and Gunicorn/uWSGI with an Nginx reverse proxy. PostgreSQL is used as the database, while Redis with Celery handles caching and background tasks. A minimum of 4 GB RAM (8 GB recommended), 20–50 GB disk space, and a stable internet connection are required for reliable operation.

- **Client Environment:**

  Users access the system through modern browsers (Chrome, Firefox, Safari, Edge) with JavaScript ES6+ support. The frontend, built with Vue.js and Bootstrap, ensures a responsive and mobile-friendly interface. A minimum screen resolution of 1024×768 and stable internet connectivity are required for real-time quiz participation and notifications.

- **External Dependencies:**

  The system depends on Redis for caching and Celery task queues, an SMTP server for sending notifications and reports, and an SSL certificate to enable secure HTTPS communication in production.

2.5 Design and Implementation Constraints

- **Design Constraints:**
  a. Must use Flask (backend) and Vue.js (frontend), no tech flexibility.
  b. Only REST APIs are allowed for integration; no direct third-party access.
  c. SSL/TLS encryption and JWT authentication are mandatory.

- **Implementation Constraints:**
  a. Deployment must be on a cloud server (AWS/Heroku) with limited free-tier resources.
  b. Background jobs depend on Celery + Redis, which may add latency.
  c. Code must follow Flask conventions, REST design, and proper documentation.

2.6 User Documentation

The Quiz Master app will include a User Manual (PDF) covering setup, roles, and navigation, along with an Online Help Section for common tasks like quiz creation, participation, and reports. Short Tutorial Guides with step-by-step examples and screenshots will also be provided in PDF and HTML formats for easy access.

2.7 Assumptions and Dependencies

- Users will access the system through modern web browsers that support JavaScript and HTML5.
- The server will have stable internet connectivity and sufficient resources for concurrent quiz participants.
- Email (for reminders and reports) will remain reliable and accessible.
- The database size will not exceed moderate limits (for example, under 50,000 quiz records).
- The number of concurrent users will be within the expected range (for example, up to 200 at a time).
- Redis and Celery will run without external configuration changes in the hosting environment.
- External libraries (Flask, Vue.js, Bootstrap) will continue to be supported and updated.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

This software needs the following user interfaces:

- **Welcome Page**

  User: Students, Administrator
  Properties: This window is used for the welcome page for the application, with login and register buttons. It also has a tagline for the Quiz Master software.

- **Registration Window**

  User: Students, Administrator
  Properties:
  a. This window is used for the entry of student details and registering a new user in the Quiz Master system. No admin registration is required.
  b. This window has text fields to take information such as full name, username (email), password, qualification, and date of birth.
  c. For security, strong password hashing, encryption, and validation will be implemented.

- **Login Window**

  User: The student users and the administrator.
  Properties:
  a. This window has two fields for username and password.
  b. For the correct username and password, it opens the appropriate dashboard (Admin or User).
  c. A “Forgot Password” link will be included to reset login credentials.

- **Student Dashboard (User Homepage)**

  User: Registered student
  Properties:
  a. This window opens when a student logs in.
  b. Students can view available quizzes, attempt quizzes with a countdown timer, and submit their answers. The answer key for the quiz will be available after the quiz.
  c. A section displays previously attempted quizzes with scores, performance statistics, and summary charts.

- **Administrator Dashboard**

  User: The administrator
  Properties:
  a. This window opens when the administrator logs in.

b. The admin can create, edit, and delete subjects, chapters, quizzes, and quiz questions.
c. The admin can view and search for users, manage quiz schedules (date and time), and send reminders for the quizzes.
d. The admin can also export user quiz data in CSV format, which will be mailed to the admin's email ID

3.1.2 Hardware Interfaces

The program will communicate with the database stored on the server via Flask backend APIs. Users interact with the system through a web browser using a keyboard, mouse, or touchscreen, and view the graphical interface on their display. The system will run on any modern computer with at least **4 GB RAM,** a stable internet connection, and a modern web browser.

3.1.3 Software Interfaces

The software will run under Windows or Linux operating systems. It will use:

- **Backend:** Flask (Python 3.11) with RESTful APIs
- **Frontend:** Vue.js (JavaScript ES6+)
- **Database:** SQLite for development and PostgreSQL for production deployment.
- **Additional Tools:** Redis for caching and Celery for background tasks, SMTP server for sending reminders and reports, and AI-powered quiz generation tools for automatic creation of question sets.
- All communication between frontend and backend will follow JSON over HTTPS

3.1.4 Communications Interfaces

The software requires communication functions such as email notifications, real-time quiz updates, and secure data transfer.

- **Web Communication:** All client-server interactions use the HTTPS protocol for encrypted communication.
- **Email Communication:** SMTP is used for reminders, quiz alerts, and monthly reports.
- **API Communication:** REST APIs in JSON format ensure smooth data transfer between frontend and backend.
- **Security:** Authentication and authorization are handled using JWT tokens.

3.2 Functional Requirements

3.2.1 Register User

3.2.1.1 Collect Personal Information of Student

Input: Full name, email, password, qualification, date of birth

Output:

- User account created and stored in the database
- Redirected to the login page for the user to log in

- A message pops up if an existing user tries to register again.

Description: New user record must not fully match any existing record in the database.

3.2.2 Login User

3.2.2.1 Enter login credentials

Input: Username (Email), password.

Output:
- User dashboard if credentials are correct
- Flash message for incorrect password
- A message pops up if a new user tries to log in without registration.

Description: Only authenticated users can access the dashboard.

3.2.2.2 Reset Password

Input: New password.

Output:
- Redirect to the login page

Description: Users can reset their password if forgotten. Need to log in again with the new password.

3.2.3 Quiz Creation

3.2.3.1 Manual Quiz Creation

Input: Questions, options, correct answers, chapter number

Output: Quiz saved in the database

Description: Admin can manually create quizzes with MCQ questions.

3.2.3.2 AI-based Automatic Quiz Generation

Input: Textbook PDF, chapter, or topic keywords

Output: Automatically generated question set with suggested answers

Description: Uses AI tools to process educational material and create quizzes.

3.2.4 Quiz Participation

3.2.4.1 Start Quiz

Input: Quiz ID

Output: Questions displayed with a timer, a submit quiz button, and a check answer button to get instant feedback for answers.

Description: Students attempt quizzes in real-time with timer enforcement.

3.2.4.2 Submit Quiz

Input: User's answers to quiz questions

Output:

- Submission confirmation, evaluation triggered.
- User redirected to scores tab

Description: The system validates input and calculates scores immediately.

3.2.5 Results and Analytics

3.2.5.1 View Results

Input: Score ID

Output: Scorecard with correct/incorrect answers and final score

Description: Students can view their results.

3.2.5.2 Performance Analytics

Input: Past quiz scores of the student

Output:

- Bar chart of quizzes attempted per subject.
- Pie chart of quizzes attempted per month.

Description: Analytics generated dynamically from quiz performance data.

3.2.6 Content Management

3.2.6.1 Upload and Manage Content

Input: Textbooks, chapters, or custom content files

Output: The quiz is generated from the content provided and stored in the database.

Description: Admins manage subjects, chapters, and approve uploaded content.

3.2.6.2 Export User performance data

Input: Past quiz scores of the student

Output: CSV file emailed to the admin email ID.

Description: Admin can see patterns in user performance.

3.2.7 Notifications

Input: Quiz schedule, reminders, results

Output: Notifications via email

Description: SMTP/email server sends reminders and monthly performance reports to users.

3.3 Behaviour Requirements

3.3.1 Use Case View

A use case defines a goal-oriented set of interactions between external actors and the Quiz Master system. While state diagrams describe system states, use cases help to capture the functional behaviour from the users' perspective. The use case diagram will represent the complete system with all possible actors and their interactions.

Actors:

- Admin: manages quizzes, users, background tasks, and system settings.
- User (Student/Participant): attempts quizzes, views scores, and tracks progress.
- AI Quiz Generator Tool: an external tool/service that assists in generating quizzes automatically.

3.3.2 Use Cases:

- Login: both Admin and User authenticate into the system.
- Create/Manage Quizzes: Admin manually creates quizzes or uses the AI generator to build them.
- Attempt Quiz: User selects and answers quiz questions within a time limit.
- View Results: User checks performance reports after attempting quizzes.
- Search Quizzes: User searches for available quizzes.
- Generate Reports: Admin generates monthly/overall performance reports.
- Daily Reminders: The System sends scheduled notifications to users.

4 Other Non-functional Requirements

4.1 Performance Requirements

- Any quiz should load within 3 seconds, even when the database contains more than 5,000 questions.
- The system should handle at least 100 concurrent users attempting quizzes without performance degradation.
- AI quiz generation requests should be processed and returned within 15 seconds under normal server load.
- Leaderboards and result summaries must update in real-time (under 2 seconds delay) after quiz submission.
- Daily reminders and scheduled notifications should be sent within 1 minute of the configured time.

4.2 Safety and Security Requirements

4.2.1 Safety Requirements

- All quiz data and results should be backed up daily to prevent data loss in case of system failure.
- The system shall automatically expire JWT tokens after 2 hours.
- The system shall log out users after 15 minutes of inactivity, regardless of token validity.

4.2.2 Security Requirements

- All user logins must be authenticated using **JWT-based authentication**.
- Passwords must be stored in the database using **strong hashing algorithms such as SHA-256**
- All communication between frontend and backend must use **SSL/TLS encryption (HTTPS)**.
- Role-based access control must be enforced (Admin vs. User).
- The system must prevent cheating via **screenshot taking and copy-pasting.**
- Protected routes for admin tasks. No unauthorised access possible

4.3 Software Quality Attributes

4.3.1 Reliability

The system will ensure 99.5% uptime during working hours by hosting on a reliable cloud service (AWS/Heroku). Automated backups and error logging will be in place to quickly restore services in case of failures.

4.3.2 Usability

The interface will be simple and intuitive, with clear navigation and accessibility features. Features like performance summary charts, instant feedback on quiz submissions, and search functionality will enhance usability.

4.3.3 Maintainability

The codebase will follow Flask and Vue.js best practices, with modular components and clear documentation. Unit tests and automated CI/CD pipelines will ensure smooth updates without breaking existing features.

4.3.4 Portability

The system will run on both Linux and Windows servers with minimal configuration. Docker containers may be used for consistent deployments across environments.

4.3.5 Security

Strong encryption, authentication, and anti-cheating mechanisms will be prioritized to maintain integrity. Sensitive data, like user passwords, will be protected with database-level encryption where possible. Both frontend and backend validation will be implemented for forms

🥺 Experiment 2

Experiment No.	2
Group Members	Riya Suryawanshi (2023300240) Anushka Suvarna (2023300241)
Division	D
Batch	B
Date of Submission	25th August 2025

Experiment 2: UML Diagrams

Use Case Diagram:

Table 1: List of Actors

Actor	Description
Admin (Quiz Master)	The admin who manages the entire system. Responsible for creating/editing/deleting quizzes, subjects, and chapters, scheduling quizzes, exporting reports, and monitoring user performance.
Student/User	A registered participant who takes quizzes, views results, and tracks progress through summary chart. Limited access only to their results and assigned quizzes.
AI Quiz Generator Tool	An external service integrated with the system to automatically generate quiz questions from uploaded textbook PDFs.
Email System (SMTP and Celery tasks)	Sends reminders, quiz alerts, and monthly performance reports to students, and performance CSV exports to admins.

Table 2: List of Use Cases

UC ID	Use Case	Description
UC1	Login / Register	Allows Students to securely log in or register to access their dashboards using role-based authentication. Admins can only log in to their Dashboard. Admin account is created beforehand.
UC2	Create & Manage Quizzes	Admin creates quizzes manually or uploads study material for AI-based quiz generation. Includes CRUD operations.
UC3	Attempt Quiz	Students select and take quizzes with a countdown timer and restricted attempts.
UC4	Submit Quiz & View Results	Students submit answers, receive instant evaluation, and view detailed results (score, correct/incorrect answers).
UC5	Performance Analytics	Students view charts and statistics of their quiz performance. Admin can track user performance trends.
UC6	Search Quizzes / Content	Users (Admin/Student) search for quizzes, subjects, or chapters within the system.
UC7	Generate Reports	Admin generates overall reports of student performance and exports them in CSV. The monthly performance report is sent to the student at their email address.
UC8	Daily Reminders & Notifications	The system automatically sends quiz reminders to students who are inactive on the application.
UC9	Password Reset	Users can reset credentials securely if forgotten.
UC10	Upload & Manage Content	Admin uploads PDFs or subject materials to manage quizzes and chapters, also triggering AI quiz generation.

Use Case Scenarios:

Table 3: UC1 - Login / Register

<table>
<tr><th>Use Case</th><th>UC1. Login / Register</th></tr>
<tr><td>Goal</td><td>To allow Students to register and log in securely, and Admins to log in with a pre-created account.</td></tr>
<tr><td>Actors</td><td>Student, Admin</td></tr>
<tr><td>Pre-condition</td><td>The student is not already registered (for registration). Admin account already exists.</td></tr>
<tr><td>Post-condition</td><td>Student accounts created or users redirected to their respective dashboards.</td></tr>
<tr><td>Mainline Scenario</td><td>
<table>
<tr><th>Actor Actions</th><th>System Actions</th></tr>
<tr><td>1. Student opens login/register page.</td><td></td></tr>
<tr><td>2. Student enters registration details OR login credentials.</td><td>3. System validates input and checks for duplicates.</td></tr>
<tr><td></td><td>4. If successful, system creates account (for student) or authenticates (for login).</td></tr>
<tr><td></td><td>5. User is redirected to respective dashboard.</td></tr>
</table>
</td></tr>
<tr><td>Alternate Flows</td><td>1. If invalid credentials then the system shows an error
2. If a duplicate registration attempt is found, then an error will be shown.
3. If the system rejects registration, then a rejection message is sent.
4. An unregistered student is prompted to register first.</td></tr>
</table>

	5. If the entered password is incorrect, prompt to enter the correct one.

Table 4: UC2 – Create & Manage Quizzes

Use Case	UC2. Create & Manage Quizzes
Goal	To allow Admin to create, edit, and manage quizzes.
Actors	Admin, AI Quiz Generator Tool
Pre-condition	Admin is logged in. A chapter is created for which the quiz is to be scheduled.
Post-condition	Quiz created/updated and stored in the database.
Mainline Scenario	<table><tr><th>Actor Actions</th><th>System Actions</th></tr><tr><td>1. Admin selects “Quiz +”</td><td></td></tr><tr><td>2. Admin enters quiz details or uploads a PDF.</td><td>3. The system saves the manually created questions OR invokes AI tools for auto-generation.</td></tr><tr><td></td><td>4. The system validates and stores quizzes.</td></tr><tr><td></td><td>5. Confirmation shown.</td></tr></table>
Alternate Flows	1. If an invalid PDF is uploaded, an error message is displayed. 2. If the date and time of the quiz are not provided, the system asks to enter the complete details.

Table 5 : UC3 – Attempt Quiz

Use Case	UC3. Attempt Quiz
Goal	To allow students to attempt quizzes with a timer and submit answers.

<table>
<tr><td>Actors</td><td>Student</td></tr>
<tr><td>Pre-condition</td><td>The student is logged in, and a quiz is scheduled.</td></tr>
<tr><td>Post-condition</td><td>Quiz submitted and results generated.</td></tr>
<tr><td>Mainline Scenario</td><td>
<table>
<tr><th>Actor Actions</th><th>System Actions</th></tr>
<tr><td>1. The student selects the assigned quiz.</td><td>2. System starts timer and tracks responses.</td></tr>
<tr><td>3. Student answers questions.</td><td>4. System validates submission.</td></tr>
<tr><td>5. Student clicks “Submit.”</td><td>6. System stores answers and evaluates results.</td></tr>
</table>
</td></tr>
<tr><td>Alternate Flows</td><td>1. If the time runs out, the system auto-submits the quiz.
2. If a quiz has already been attempted, that quiz is not displayed to the user.</td></tr>
</table>

Table 6 : UC4 - Submit Quiz & View Results

<table>
<tr><th>Use Case</th><th>UC4. Submit Quiz & View Results</th></tr>
<tr><td>Goal</td><td>To allow students to view results and performance charts.</td></tr>
<tr><td>Actors</td><td>Student</td></tr>
<tr><td>Pre-condition</td><td>Student</td></tr>
<tr><td>Post-condition</td><td>Results stored and displayed.</td></tr>
<tr><td>Mainline Scenario</td><td>
<table>
<tr><th>Actor Actions</th><th>System Actions</th></tr>
<tr><td>1. Student submits answers.</td><td>2. System evaluates answers.</td></tr>
<tr><td>3. Student opens results tab.</td><td>4. System displays scorecard (correct/incorrect, marks).</td></tr>
</table>
</td></tr>
</table>

Alternate Flows	1. If student exits mid-quiz, an incomplete submission is flagged.

Table 7 : UC5 - Performance Analytics

Use Case	UC5. Performance Analytics
Goal	To provide Students and Admins with visual performance insights.
Actors	Student, Admin
Pre-condition	The student has attempted at least one quiz.
Post-condition	Analytics are displayed in charts and graphs.
Mainline Scenario	<table><tr><th>Actor Actions</th><th>System Actions</th></tr><tr><td>1. Student selects “Summary”</td><td>2. System fetches past results.</td></tr><tr><td></td><td>3. System generates charts (bar, pie).</td></tr><tr><td></td><td>4. System displays trends.</td></tr></table>
Alternate Flows	1. If no quizzes are attempted, display “No data available.”

Table 8 : UC6 - Search Quizzes / Content

Use Case	UC6. Search Quizzes / Content
Goal	To allow users to search for quizzes, subjects, or chapters.
Actors	Student, Admin

<table>
<tr><td>Pre-condition</td><td>User is logged in.</td></tr>
<tr><td>Post-condition</td><td>Search results displayed.</td></tr>
<tr><td>Mainline Scenario</td><td>
<table>
<tr><th>Actor Actions</th><th>System Actions</th></tr>
<tr><td>1. User enters search keyword.</td><td>2. System queries the database.</td></tr>
<tr><td></td><td>3. Results displayed.</td></tr>
</table>
</td></tr>
<tr><td>Alternate Flows</td><td>1. No results found, show “No results found.”
2. Invalid search input, display “Invalid search type”.</td></tr>
</table>

Table 9 : UC7 - Generate Reports

<table>
<tr><th>Use Case</th><th>UC7. Generate Reports</th></tr>
<tr><td>Goal</td><td>To allow Admin to generate overall performance reports and send students their monthly reports.</td></tr>
<tr><td>Actors</td><td>Admin, Student, Email System</td></tr>
<tr><td>Pre-condition</td><td>Performance data exists.</td></tr>
<tr><td>Post-condition</td><td>Report generated and emailed/exported.</td></tr>
<tr><td>Mainline Scenario</td><td>
<table>
<tr><th>Actor Actions</th><th>System Actions</th></tr>
<tr><td>1. Admin selects “Send Performance Reports” or “Export CSV”</td><td>2. System compiles performance data.</td></tr>
<tr><td></td><td>3. System generates CSV and a monthly report in HTML format.</td></tr>
<tr><td></td><td>4. System emails the monthly report to students and the CSV to the admin</td></tr>
</table>
</td></tr>
</table>

Alternate Flows	1. If email fails, display an error message. 2. If no data, the system generates an empty report.

Table 10 : UC8 - Daily Reminders & Notifications

Use Case	UC8. Daily Reminders & Notifications
Goal	To notify inactive students of scheduled quizzes.
Actors	System (Celery and SMTP), Student
Pre-condition	Student has registered email and quizzes scheduled.
Post-condition	Notification delivered.
Mainline Scenario	<table><tr><th>Actor Actions</th><th>System Actions</th></tr><tr><td>1. Student receives notification.</td><td>2. System schedules background task.</td></tr><tr><td></td><td>3. System sends reminder email.</td></tr></table>
Alternate Flows	1. If email server fails, display an error message.

Table 11 : UC9 - Password Reset

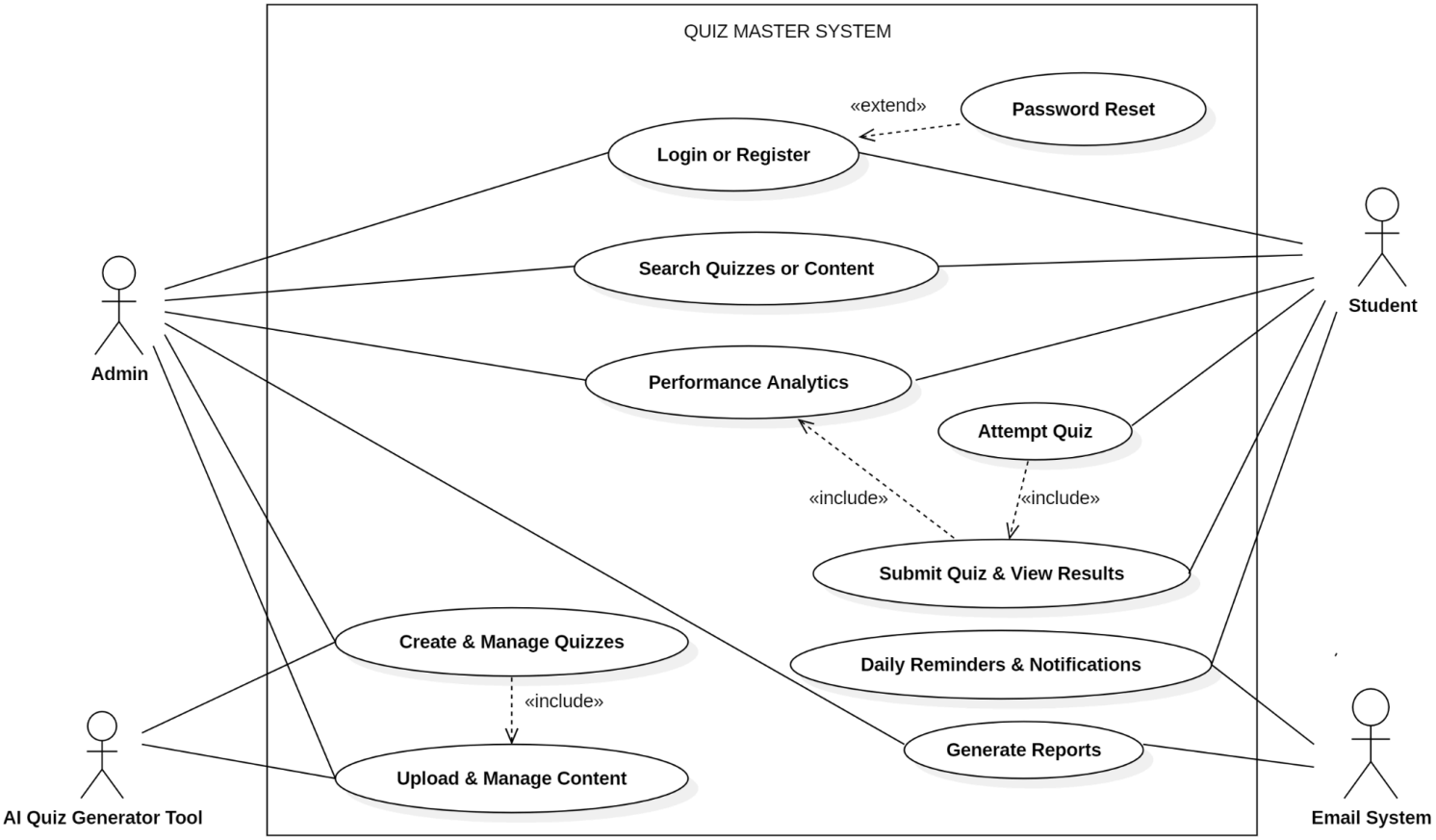
Use Case	UC9. Password Reset
Goal	To allow Users to reset their password securely.
Actors	Student, Admin
Pre-condition	User is already registered.
Post-condition	Password updated successfully.

<table>
<tr><td>Mainline Scenario</td><td>
<table>
<tr><th>Actor Actions</th><th>System Actions</th></tr>
<tr><td>1. User clicks “Forgot Password.”</td><td></td></tr>
<tr><td>2. User enters email.</td><td>3. System verifies email.</td></tr>
<tr><td></td><td>4. System sends OTP to the email.</td></tr>
<tr><td></td><td>5. User sets new password.</td></tr>
</table>
</td></tr>
<tr><td>Alternate Flows</td><td>1. If the email is not registered, display an error message.
2. If the OTP expires, the resend option is available.</td></tr>
</table>

Table 12 : UC10 - Upload & Manage Content

<table>
<tr><th>Use Case</th><th>UC12. Upload & Manage Content</th></tr>
<tr><td>Goal</td><td>To allow Admin to upload PDFs and manage subjects/chapters.</td></tr>
<tr><td>Actors</td><td>Admin, AI Quiz Generator Tool</td></tr>
<tr><td>Pre-condition</td><td>Admin is logged in, and chapter is created.</td></tr>
<tr><td>Post-condition</td><td>Content stored and quiz generated if required.</td></tr>
<tr><td>Mainline Scenario</td><td>
<table>
<tr><th>Actor Actions</th><th>System Actions</th></tr>
<tr><td>1. Admin selects “Upload Content.”</td><td></td></tr>
<tr><td>2. Admin uploads PDF or subject file.</td><td>3. System validates file.</td></tr>
<tr><td></td><td>4. AI generates questions (if selected).</td></tr>
<tr><td></td><td>5. Content stored in database.</td></tr>
</table>
</td></tr>
<tr><td>Alternate Flows</td><td>1. Invalid file format, display an error message.
2. AI fails to process, then manual entry of questions is required.</td></tr>
</table>

Use Case Diagram :



🥳 Experiment 3

Experiment No.	3
Group Members	Riya Suryawanshi (2023300240) Anushka Suvarna (2023300241)
Division	D
Batch	B
Date of Submission	25th August 2025

Experiment 3: Class Diagram

AIM:	To design and draw a Class Diagram for the Quiz Master software system.

IMPLEMENTATION:

The class diagram is a static structural view of a system. It captures classes, their attributes and operations (methods), along with associations, generalization, and multiplicities. The usual process begins by revisiting the problem description to extract nouns (candidate entities) and verbs (candidate operations), resolving ambiguity, and then organizing them into well-defined classes and relationships. This ensures conceptual clarity before coding and aligns the eventual implementation with the system's functional requirements.

For the Quiz Master software, we re-read the requirements to identify key actors (users and admins), core artifacts (quizzes, questions, options), operational flows (attempts, responses, scoring), and supportive subsystems (AI-assisted question generation and scheduled reminders). After consolidating these elements, we established relationships and multiplicities to reflect constraints like "one quiz contains many questions" and "a user may submit many responses," and finally constructed the class diagram.

PROBLEM DESCRIPTION:

An educational quiz platform must allow different types of users to perform their respective functions. A registered user (participant) must be able to log in, search quizzes, attempt timed quizzes, and receive evaluated results. Quizzes contain MCQ questions with defined time limits. For MCQ questions, each question has multiple options with one correct option. When a participant attempts a quiz, the system records responses, computes a score, and stores the result with completion time and timestamp.
An admin is responsible for creating and managing quizzes, adding questions and options, publishing quizzes, managing users, and viewing reports. To assist quiz construction, the system includes an AI Quiz Generator that can create candidate questions based on topic and difficulty. Routine

communication—such as daily reminders to complete assigned quizzes or monthly performance summaries—is automated using a scheduled reminder job (e.g., Celery task).

[1] Noun / Noun Phrases

User, Admin, Participant, Authentication, Profile, Quiz, Question, Option, Response, Submission, Result, Score, Report, Time Limit, Topic/Tag, AI Quiz Generator, Reminder Job, Scheduler, Notification, Login, Registration, Dashboard.

[2] Classes

- User
- Admin (inherits User)
- Quiz
- Question
- Option
- Response
- Result
- AIQuizGenerator
- ReminderJob

[3] Verb Phrases (System Actions → Methods)

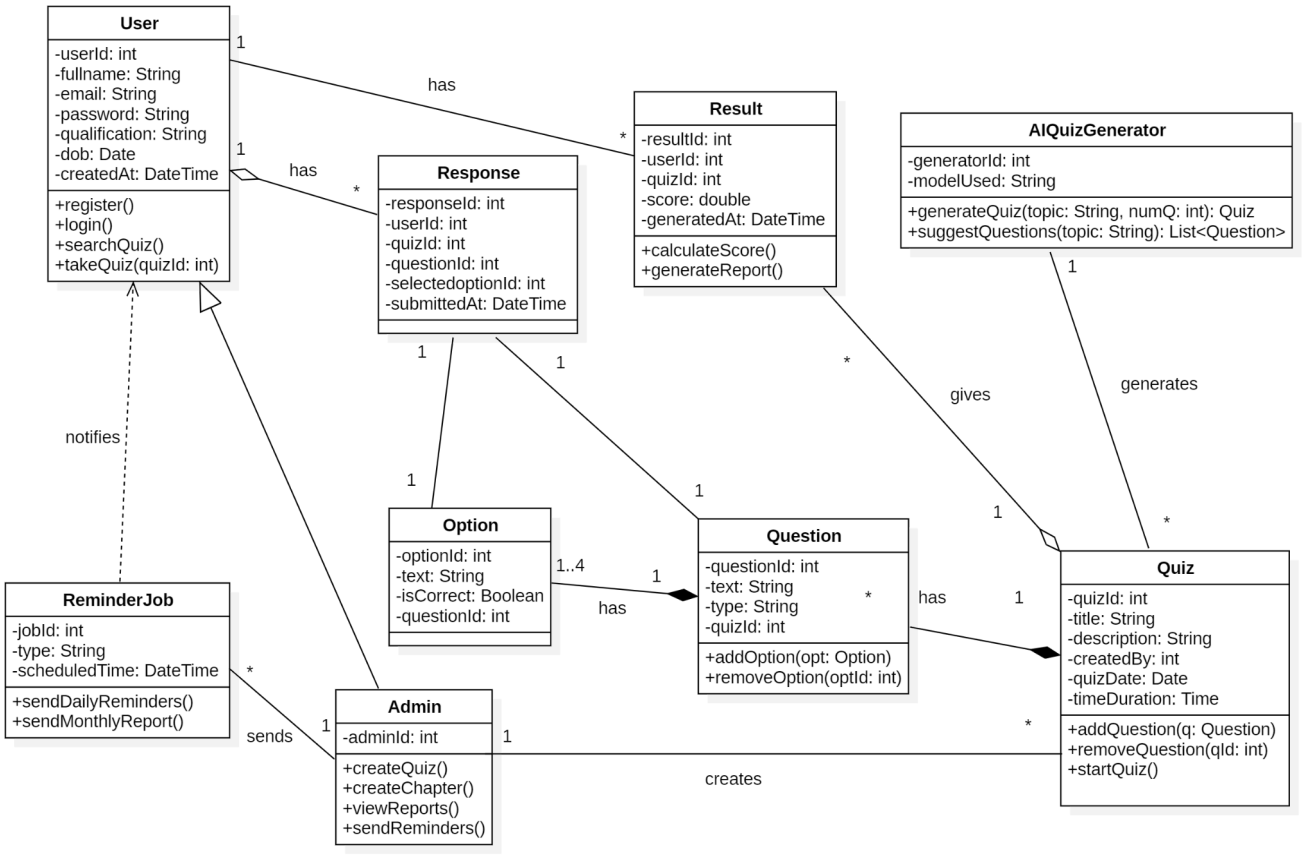
1. User registers, logs in, and searches for available quizzes.
2. Admin creates, edits, or deletes a quiz.
3. Admin adds/removes questions and adds/marks options.
4. The participant takes a quiz and submits responses.
5. The system evaluates responses and calculates scores.
6. The system generates results and produces a report/summary.
7. AIQuizGenerator generates questions and automatically creates the quiz.
8. ReminderJob schedules and sends reminders/reports.

[4] Relations

1. An Admin can create many Quizzes.
2. A Quiz must have at least one Question.
3. A Question may have multiple Options.
4. A User can submit many Responses.
5. A Question can be answered by many Responses.
6. A Quiz collects many Responses.
7. A User can receive many Results.
8. A Quiz produces many Results.
9. The AI Quiz Generator can generate many Quizzes.

10. A Reminder Job can notify many Users.

CLASS DIAGRAM:



Following are the code snippets that were written corresponding to each class and interface demonstrated in the above diagram:

User :

```
class User {
    int userId;
    String fullname, email, password, qualification;
    java.util.Date dob;
    java.time.Instant createdAt;

    void register() {}
    void login() {}
    void searchquiz() {}
    void takeQuiz(int quizId) {}
}
```

Admin :

```
class Admin extends User {
    void createQuiz() {}
    void createChapter() {}
    void viewReports() {}
    void sendReminders() {}
}
```

Quiz :

```
class Quiz {
    int quizId;
    String title, description;
    int createdById;
    java.util.Date quizDate;
    int timeDuration;

    void addQuestion(Question q) {}
    void removeQuestion(int questionId) {}
    Void startQuiz() {}
    }
```

Question :

```
class Question {
    int questionId;
    int quizId;
    String questionText;
    enum Type { MCQ }
    Type questionType;

    void addOption(Option o) {}
    void removeOption(int optId)
}
```

Option :

```
class Option {
    int optionId;
    int questionId;
    String optionText;
    boolean isCorrect;
```

```
}
```

Response :

```
class Response {
    int responseId;
    int userId;
    int quizId;
    int questionId;
    Integer selectedOptionId;
    java.time.Instant submittedAt;
}
```

Result :

```
class Result {
    int resultId;
    int userId;
    int quizId;
    double score;

    java.time.Instant generatedAt;

    void calculateScore() {}
    void generateReport() {}
}
```

AIQuizGenerator :

```
class AIQuizGenerator {
    int generatorId;
    String modelUsed;

    java.util.List<Question> suggestQuestions(String topic) { return null; }
    Quiz generateQuiz(String topic, int numQ) { return null; }
}
```

ReminderJob :

```
class ReminderJob {
    int jobId;
    enum JobType { DAILY_REMINDER, MONTHLY_REPORT }
    JobType jobType;
    java.time.Instant scheduledTime;
```

```
    void sendDailyReminder() {}
    void sendMonthlyReport() {}
}
```

CONCLUSION:	By performing this experiment, we were able to understand the process of designing a class diagram for the Quiz Master system. We identified entities, their attributes, and methods, along with relationships between them. This helped in constructing a structured representation of the software system which can be directly used to create UML diagrams in tools like STAR UML.

😉 Experiment 4

Experiment No.	4
Group Members	Riya Suryawanshi (2023300240) Anushka Suvarna (2023300241)
Division	D
Batch	B
Date of Submission	16th September 2025 (added Collaboration Diagram)

Experiment 4: Sequence and Collaboration Diagra m

AIM:	To design and draw a Sequence Diagram and collaboration diagram for the Quiz Master software system.

SEQUENCE DIAGRAM

IMPLEMENTATION:

A sequence diagram is a dynamic behavioral view of a system. It captures how different parts of the system interact with each other over time, showing the flow of messages and operations between various components. The usual process begins by revisiting the problem description to identify key interactions (who talks to whom), the order of operations (what happens first, second, third), and the flow of information between system components. This ensures we understand how the system behaves during runtime before coding and helps align the eventual implementation with the system's operational requirements.

For the Quiz Master software, we re-examined the requirements to identify key interaction flows (how admins generate reports, how students take quizzes, how quizzes are created), the sequence of operations (login → select → answer → submit → score), and the communication between different system parts (quiz system talking to databases, email services, AI generators). After mapping these behavioral patterns, we established the timing and order of interactions to reflect real-world usage like "first student selects quiz, then answers questions one by one, finally submits for scoring," and constructed the sequence diagrams to show these dynamic processes.

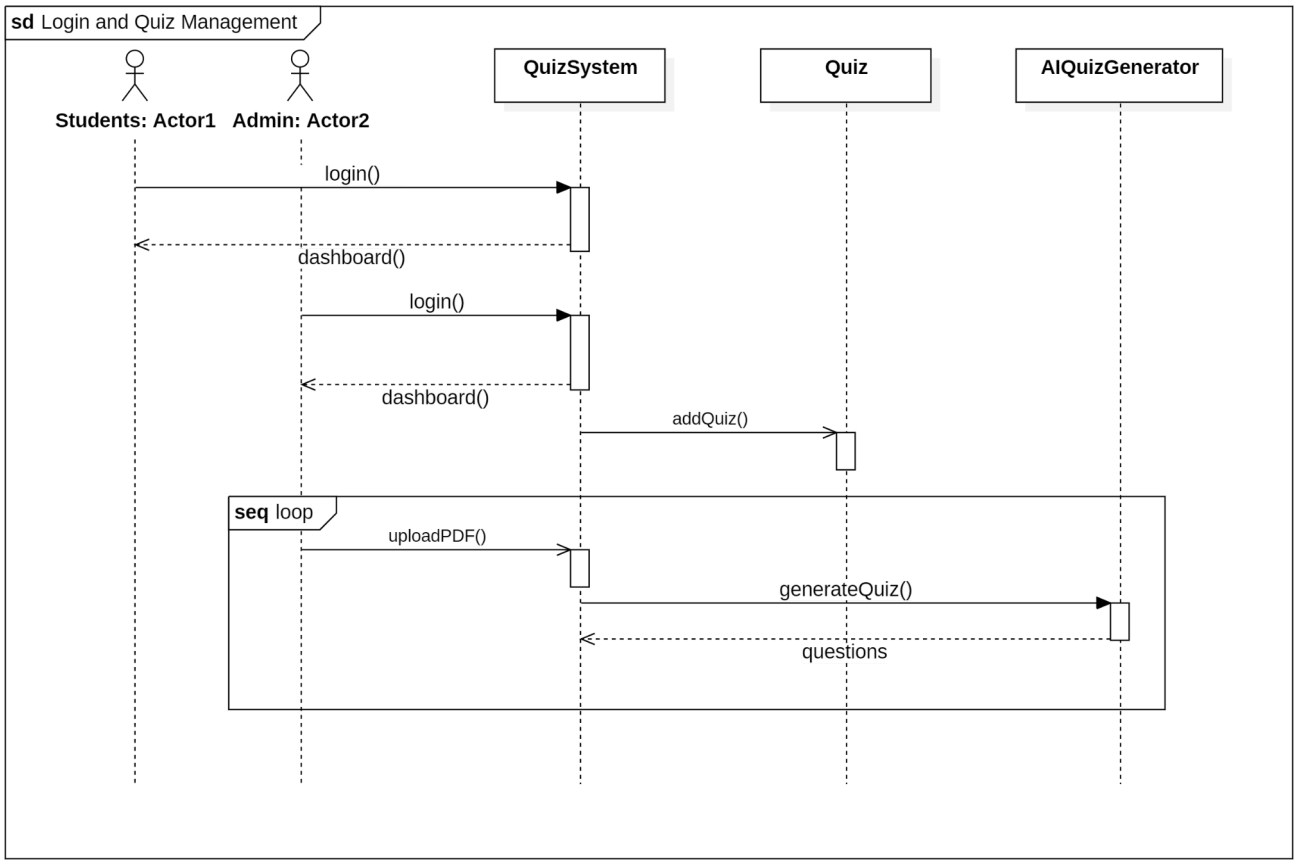
SEQUENCE DIAGRAM STRUCTURE:

The Quiz System is a comprehensive educational platform that supports multiple user roles and provides various functionalities, including quiz creation, quiz taking, automated report generation, and AI-powered question generation. The system consists of several interconnected components that work together to provide a seamless learning experience.

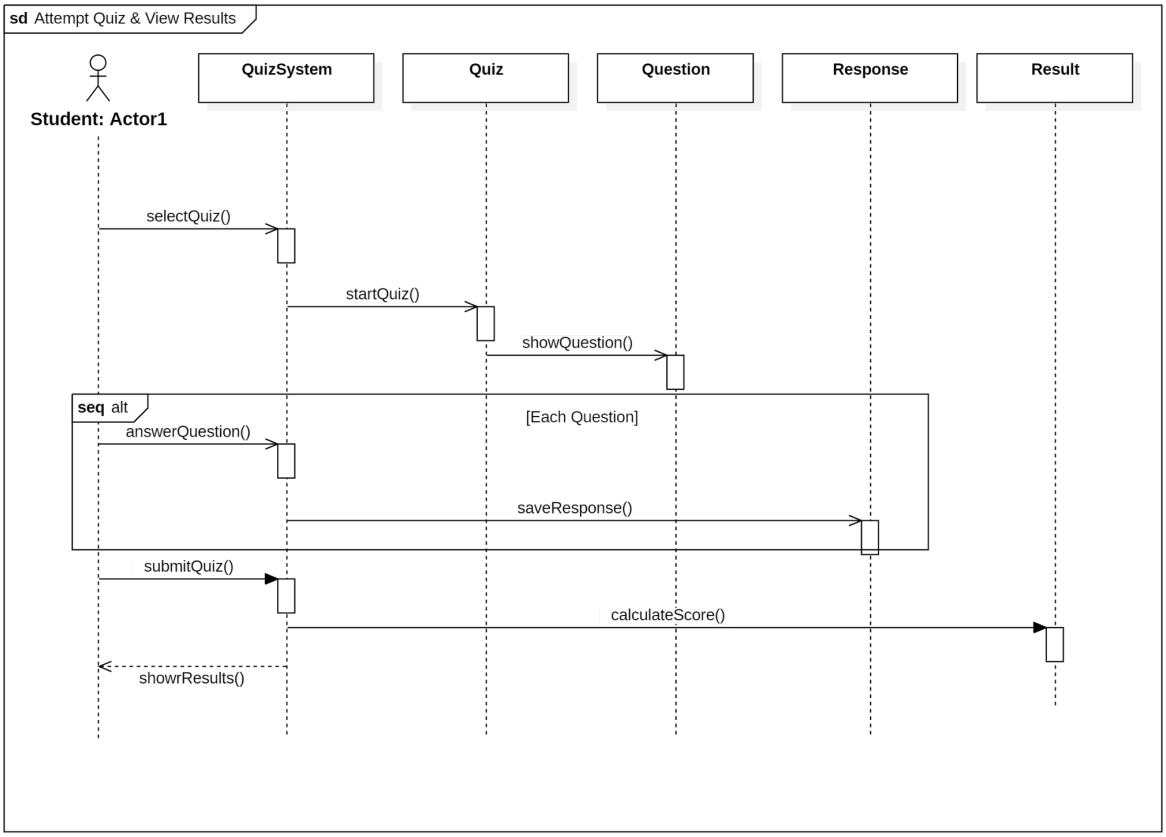
We created three distinct sequence diagrams because our Quiz Master system has three major interaction workflows that serve different user needs and involve different system components. Rather than cramming everything into one complex diagram, we separated them by functionality: Login and Quiz Management (user authentication and quiz creation), Attempt Quiz and View Results (student quiz-taking experience), and Reports and Reminders (admin analytics and automated notifications). Each diagram focuses on a specific user journey with its own actors, timing, and component interactions. Together, these three diagrams provide a complete behavioral picture of our system - showing how users access and manage quizzes, how students take quizzes and get results, and how administrators monitor performance through reports and reminders. This modular approach makes each workflow easier to understand, implement, and maintain while ensuring all major system behaviors are properly documented.

SEQUENCE DIAGRAM:

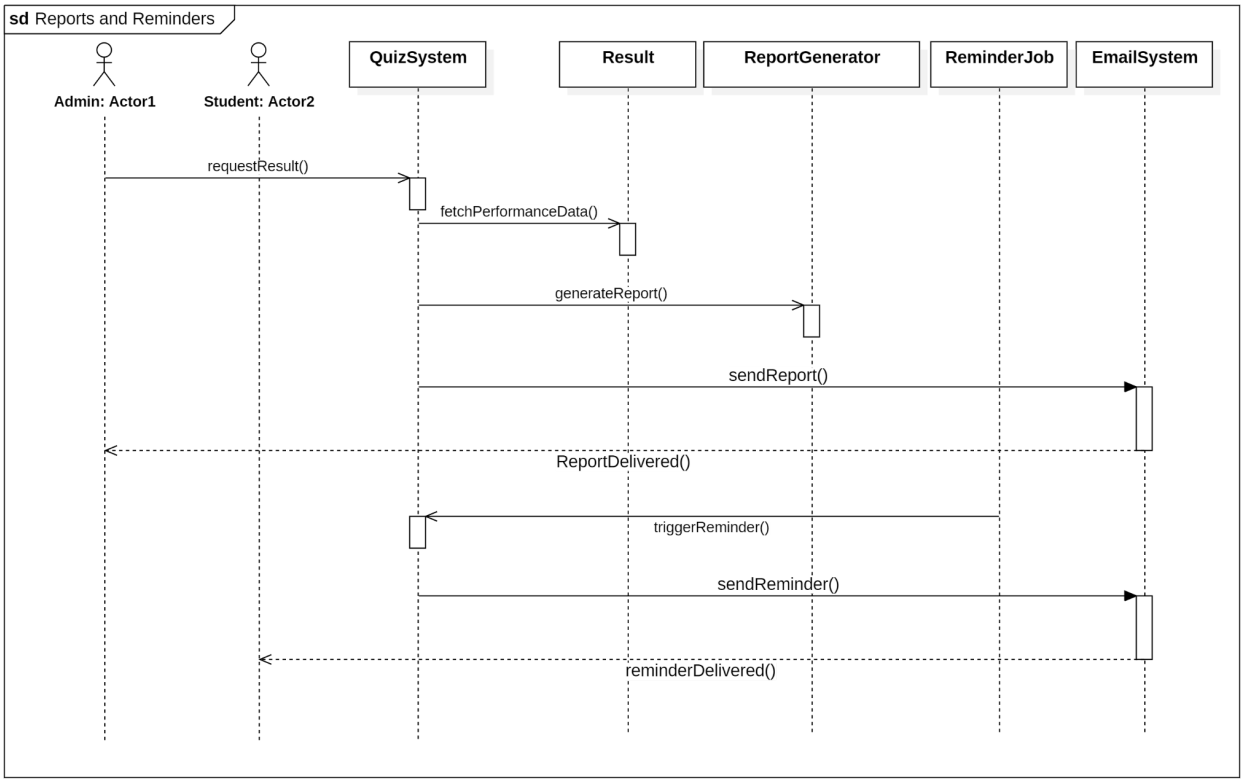
1. **Login and Quiz Management**



2. Attempt Quiz and View Results



3. Reports and Reminders



COLLABORATION DIAGRAM

IMPLEMENTATION:

A collaboration diagram (also called a communication diagram) is another form of an interaction diagram in UML. While a sequence diagram emphasizes the time-ordering of messages, a collaboration diagram emphasizes the structural organization of objects and their links. It shows how system components are connected and how they collaborate to complete a task by exchanging messages.

In designing collaboration diagrams for the Quiz Master system, we first revisited the requirements and identified objects (actors and system components) and the links (associations/relationships) between them. Unlike the sequence diagram, where the focus is on "what happens first, second, third," the collaboration diagram highlights who interacts with whom and how messages are passed through established links.

For the Quiz Master software, we mapped out the collaborations among students, admins, the quiz system, the database, and supporting services (like AI question generator and email notification system). This ensured that each user role is shown in relation to the system components it interacts with. The numbered messages in the diagram reflect the same order of interactions we documented in the sequence diagrams, providing consistency while also giving a structural network view of the system.

COLLABORATION DIAGRAM STRUCTURE:

The Quiz Master system consists of multiple user roles (students, admins) and system services (quiz creation, attempt, scoring, reports, and reminders). To clearly represent collaboration, we again divided the diagrams into three major workflows, ensuring clarity and modularity:

1. Login and Quiz Management
    - Actors: Admin, Student
    - Objects: Authentication Service, Quiz Database, Quiz Creation Module
    - Collaboration focus: How users authenticate and how admins create/manage quizzes through linked system components.

2. Attempt Quiz and View Results
    - Actors: Student
    - Objects: Quiz Interface, Quiz Engine, Question Bank, Result Processor, Database
    - Collaboration focus: How a student selects and attempts a quiz, the communication with the question bank, and retrieval of scores.

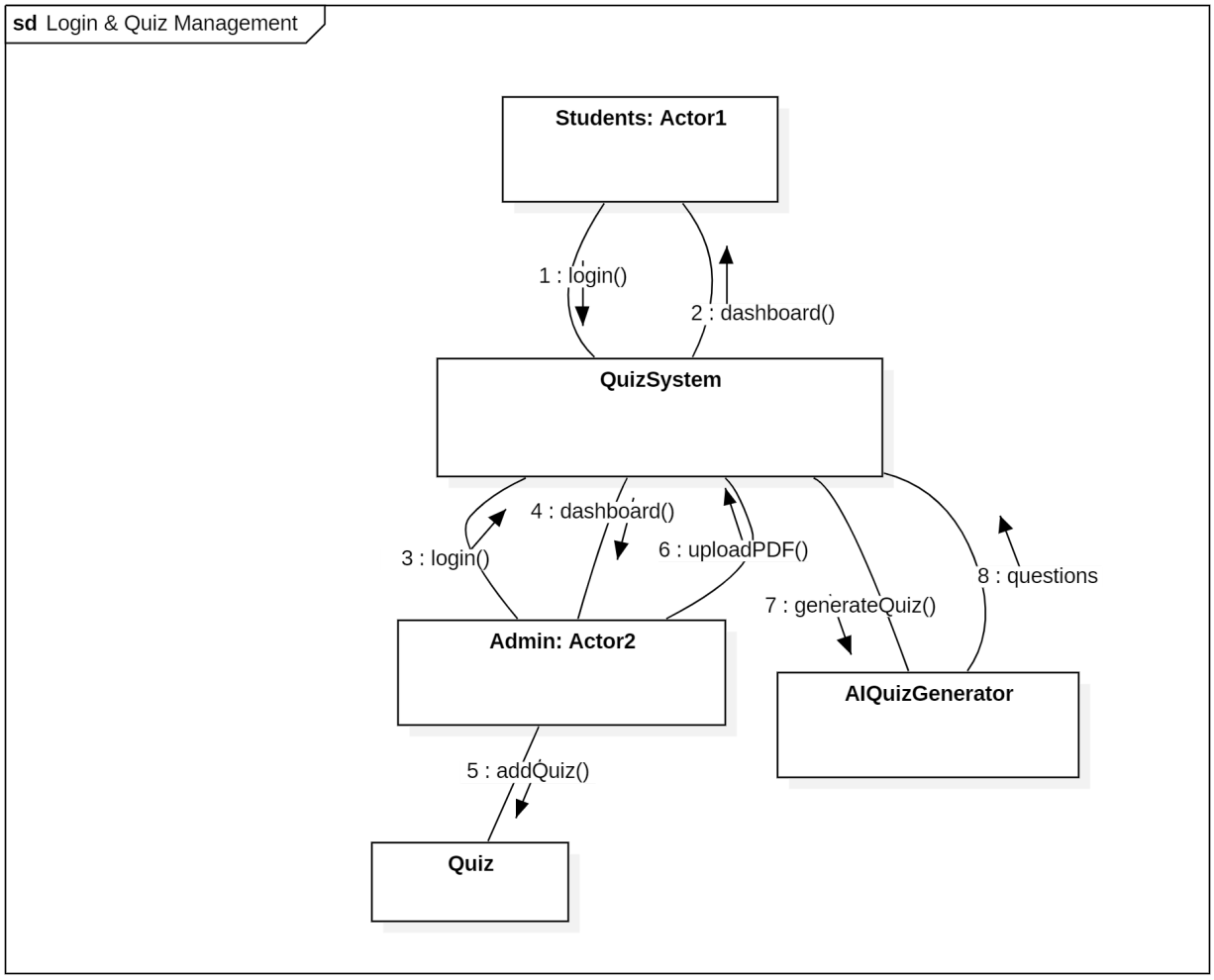
3. Reports and Reminders
    - Actors: Admin

- Objects: Report Generator, Analytics Module, Email/Notification Service, Database
- Collaboration focus: How administrators request performance reports, how data flows through analytics, and how reminders are triggered for students.

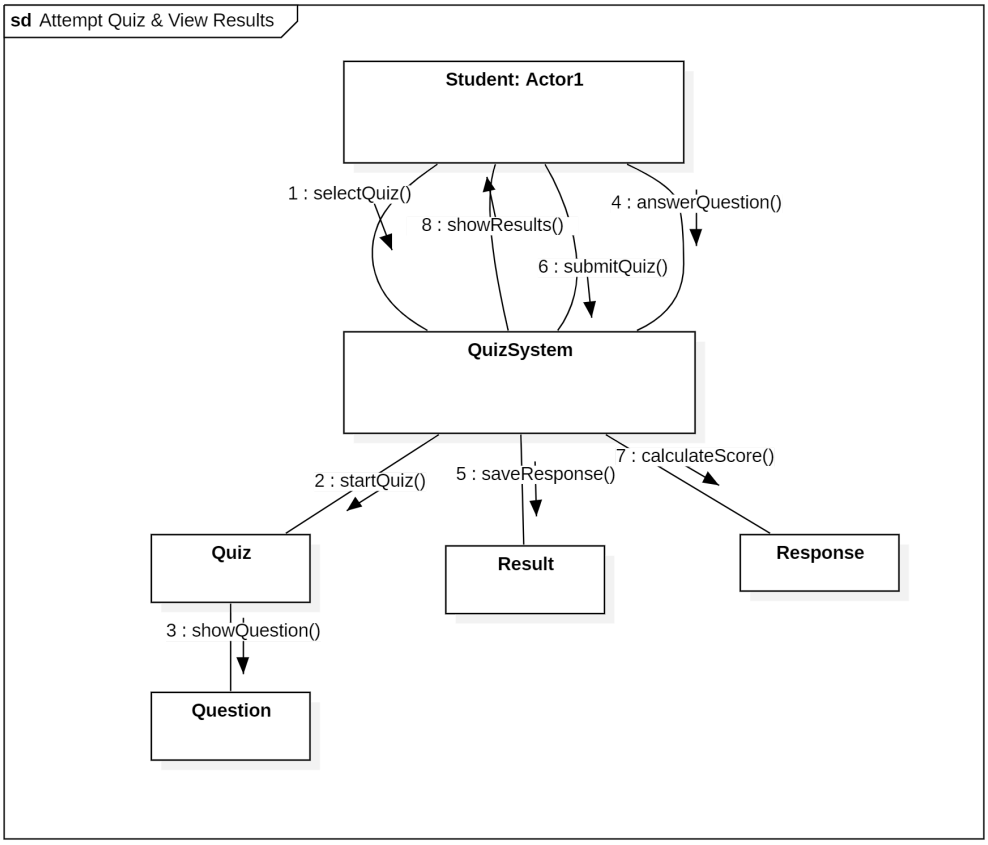
Each collaboration diagram is built around the links between objects and shows numbered message exchanges to depict the interaction flow. This structural representation complements the sequence diagrams by making clear which objects must exist and be connected for the system to function as intended.
By combining the sequence and collaboration diagrams, we ensure that the Quiz Master software system is fully modeled in terms of both dynamic behavior (over time) and static collaboration (structural links), providing developers and stakeholders with a comprehensive understanding before implementation.

COLLABORATION DIAGRAM:

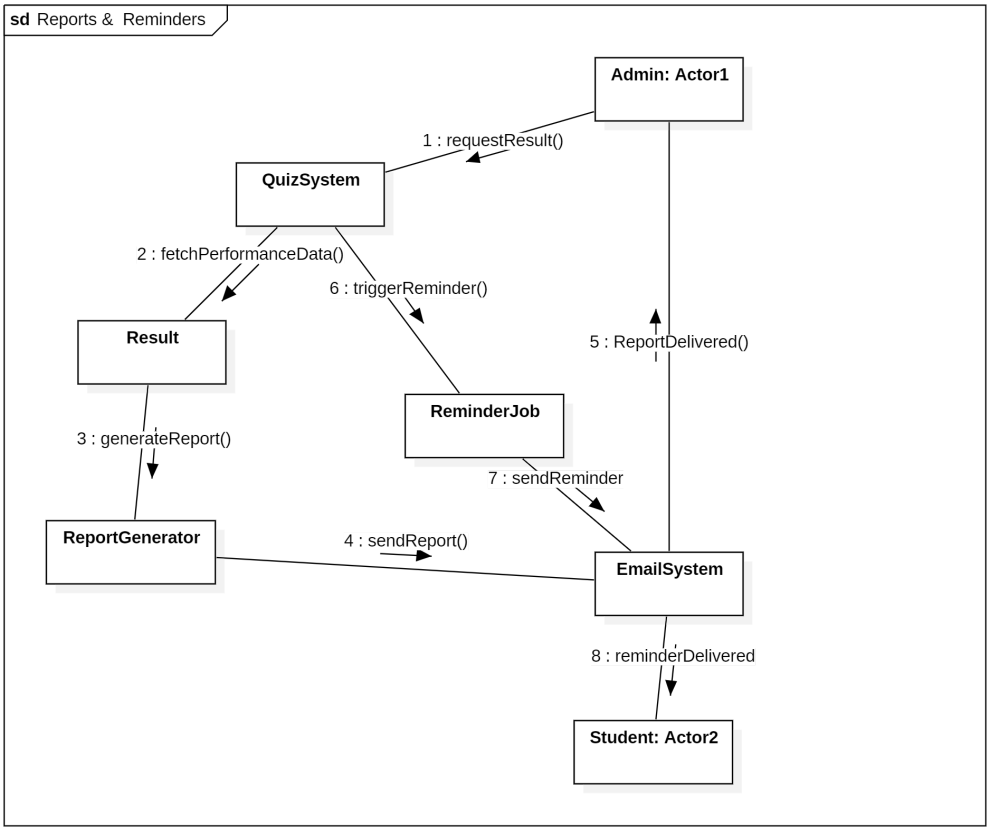
1. **Login and Quiz Management**



2. Attempt Quiz and View Results



3. Reports and Reminders



CONCLUSION:	By performing this experiment, we understood the process of designing both sequence and collaboration diagrams for the Quiz Master system. The sequence diagrams helped us capture the dynamic behavior of the system, showing the order of interactions and message flows between users and components during activities like login, quiz attempt, and report generation. The collaboration diagrams, on the other hand, emphasized the structural organization of objects and their links, highlighting how different modules and services work together. By numbering messages, we ensured consistency with the sequence diagrams while presenting a clear network of relationships. Together, these diagrams provide a complete view of system behavior and structure, making implementation easier and communication between components more effective.

☺ Experiment 5

Experiment No.	5
Group Members	Riya Suryawanshi (2023300240) Anushka Suvarna (2023300241)
Division	D
Batch	B
Date of Submission	15th September 2025

Experiment 5: Activity Diagram

AIM:	To design and draw an Activity Diagram for the Quiz Master software system.

IMPLEMENTATION:

An activity diagram provides a dynamic view of the system by modeling the workflow of activities and actions in different scenarios. It shows the sequence of steps, decision points, and parallel flows that occur during the execution of the system. Unlike sequence diagrams, which focus on the interaction between components, activity diagrams emphasize how processes flow from one activity to another.

ACTIVITY DIAGRAM STRUCTURE:

Activity diagrams describe the flow of operations in the Quiz Master system, showing how users (students and admins) and background processes interact with the software. They provide a step-by-step view of activities, decisions, and outcomes, helping visualize runtime behavior before implementation.

Student and Admin Activities:
This diagram shows two parallel flows.

- **Student Flow:** A student logs in, selects a quiz, answers questions sequentially, submits the quiz, and finally views the result. Each activity is executed in order, ending with the display of the score and completion status.

- **Admin Flow:** An admin logs in, creates a quiz, adds relevant questions, publishes it for students, and later views reports generated from student performances. Both flows highlight the user-driven operations in the system.

Background Jobs (AI and Reminders):
This diagram captures the system's automated tasks.

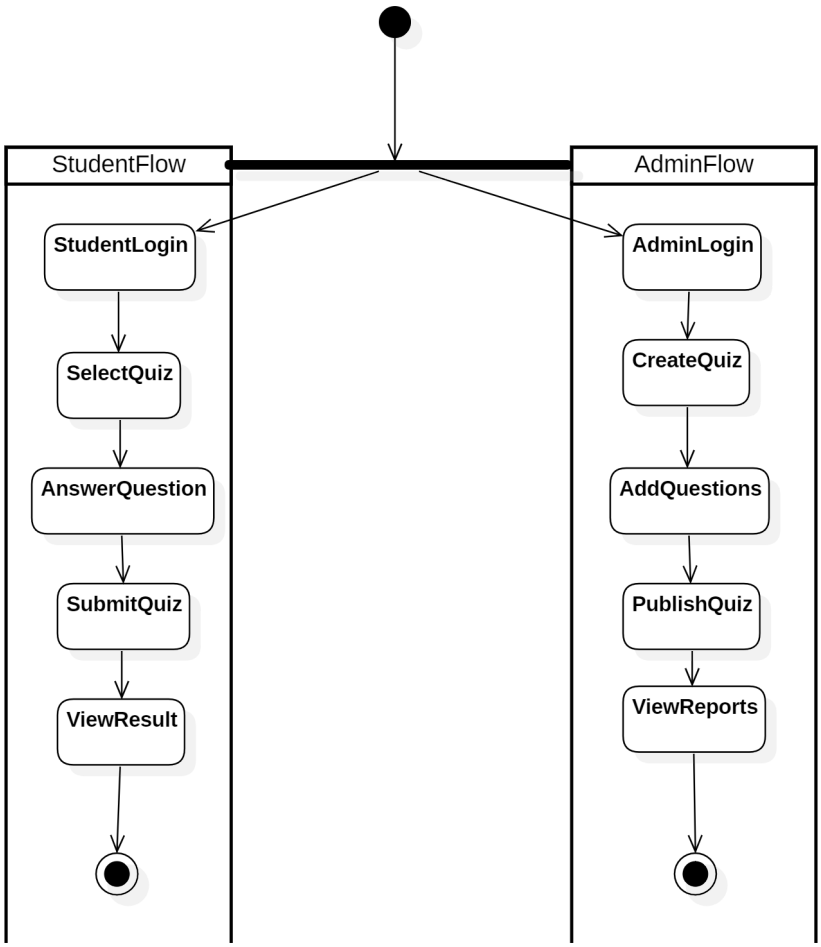
- **AI Quiz Generation Flow:** The AI engine generates quiz questions based on topic and difficulty, validates them, and stores them in the database for future use.

- **Reminder Flow:** Scheduled jobs handle sending daily reminders to users and generating monthly performance reports. Reports are compiled and dispatched through email, while reminders keep students and admins updated about upcoming tasks.

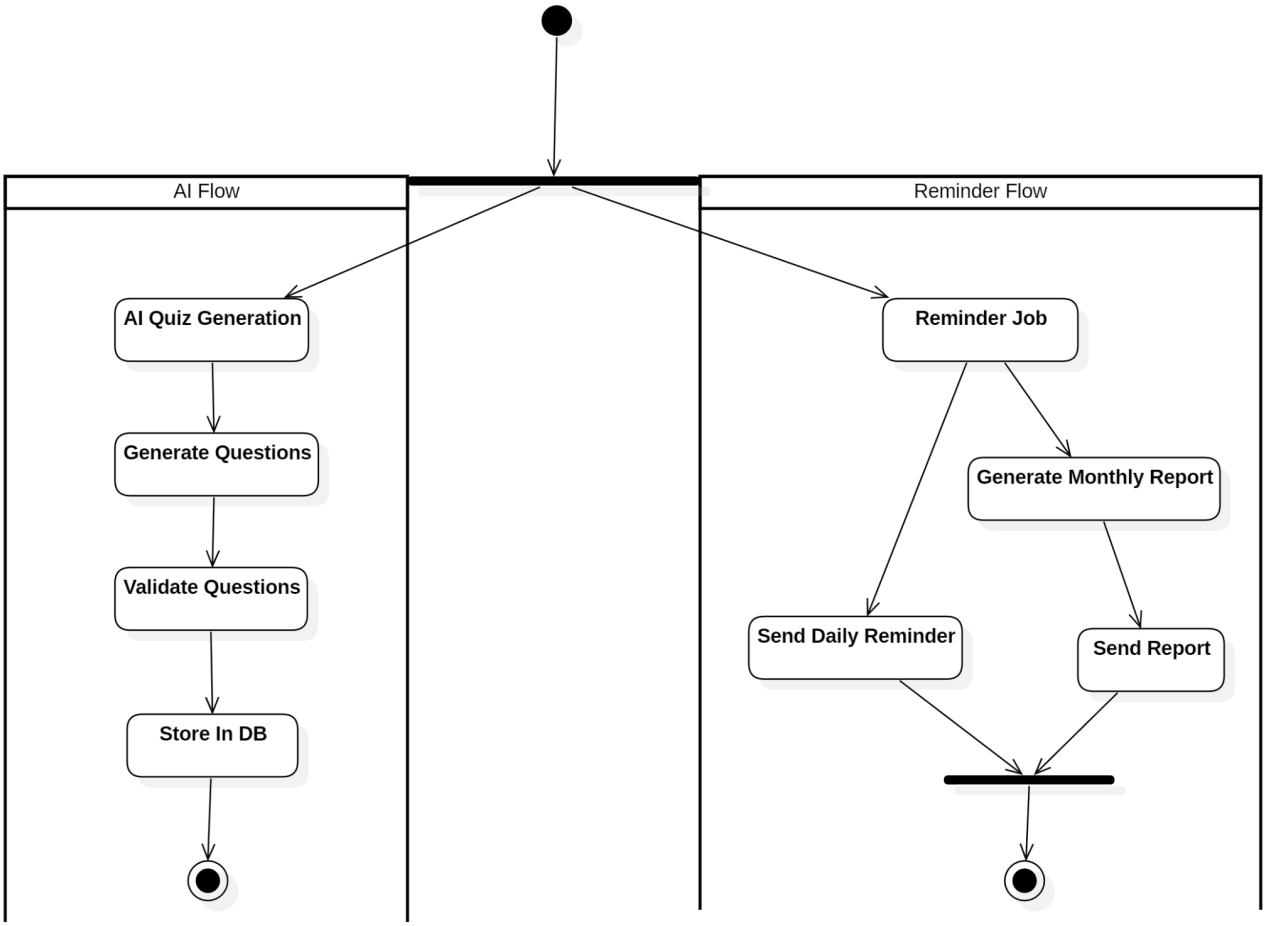
Together, these activity diagrams provide a comprehensive view of both interactive user actions and automated system processes, ensuring the Quiz Master system is well-defined for both human users and background operations.

ACTIVITY DIAGRAM:

1. **Student and Admin Activities**



2. Background Jobs



CONCLUSION:	By performing this experiment, we were able to understand the process of designing activity diagrams for the Quiz Master system. We identified the key workflows for both students and admins, including their decisions, actions, and outcomes. This helped in constructing a structured representation of the system's functional behavior, which can be directly modeled in UML activity diagrams using tools like StarUML. The two separate diagrams provided a clear view of how students take quizzes and how admins manage quizzes and reports, making it easier to visualize system processes and ensure smooth execution of all activities.

🤩 Expermient 6

Experiment No.	6
Group Members	Riya Suryawanshi (2023300240) Anushka Suvarna (2023300241)
Division	D
Batch	B
Date of Submission	20th October 2025

Experiment 6: Data Flow Diagram & Structure Chart Diagram

AIM:	To design and draw Data Flow Diagrams (DFDs) and a structure chart diagram for the Quiz Master software system.

DATA FLOW DIAGRAM

IMPLEMENTATION:

A Data Flow Diagram (DFD) is a structured analysis and design tool that represents the logical flow of information within a system. It highlights how data moves from one process to another, how it is stored, and how external entities interact with the system. Unlike activity diagrams, which focus on actions and workflows, DFDs emphasize data transformation, storage, and exchange across system boundaries.

DFDs are generally created at multiple levels to progressively refine the system:

1. **Level 0 DFD (Context Diagram):**
   - Represents the system as a single process block called the Quiz Master System.
   - Identifies the main external entities: Admin and User.
   - Shows the high-level data exchanges:
     - Admin manages quizzes and views reports, receiving results and feedback.
     - User registers, attempts quizzes, submits answers, and views results.
   - This level provides a macro-level understanding of system interactions without internal details.

2. **Level 1 DFD:**
   - Breaks the system into four core processes:

- Authentication: Handles login, registration, and verification of users and admins.
- Quiz Management: Allows admins to create, update, and delete quizzes, and store questions.
- Quiz Participation: Enables students to attempt quizzes, submit answers, and receive evaluations.
- Reports & Analytics: Generates reports on student performance for admin review and analytics.

- The Database plays a central role by storing authentication data, quiz questions, answers, and performance results.
- This level captures the functional decomposition of the Quiz Master System.

3. **Level 2 DFD (Detailed Quiz Flow):**
   - Provides a granular view of quiz participation.
   - Core processes include:
     - Select Quiz: User chooses a quiz based on availability.
     - Fetch Questions: The System retrieves corresponding questions from the database.
     - Attempt Quiz: User answers each question sequentially.
     - Submit Answers: Responses are validated and stored in the database.
     - View Results: Final score and status are generated and shown to the user.

   - This level emphasizes step-by-step interaction between the user and database during the quiz lifecycle.

DATA FLOW DIAGRAM STRUCTURE:

Data Flow Diagrams for the Quiz Master system are divided into three major categories of participants:

1. User Activities (Students):
   - Register/Login into the system.
   - Select a quiz from the available list.
   - Fetch quiz questions from the database.
   - Attempt questions in sequence and submit answers.

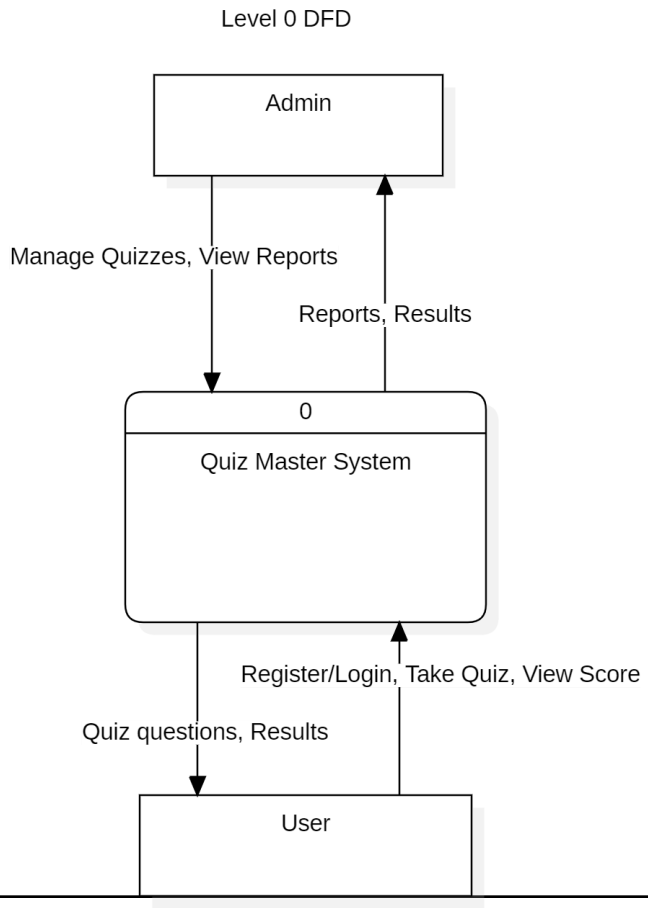
- View performance reports and results.

2. Admin Activities:
   - Authenticate and log in to the system.
   - Create and manage quizzes.
   - Add questions to the quiz bank.
   - Publish quizzes for students.
   - Access analytics and view performance reports.

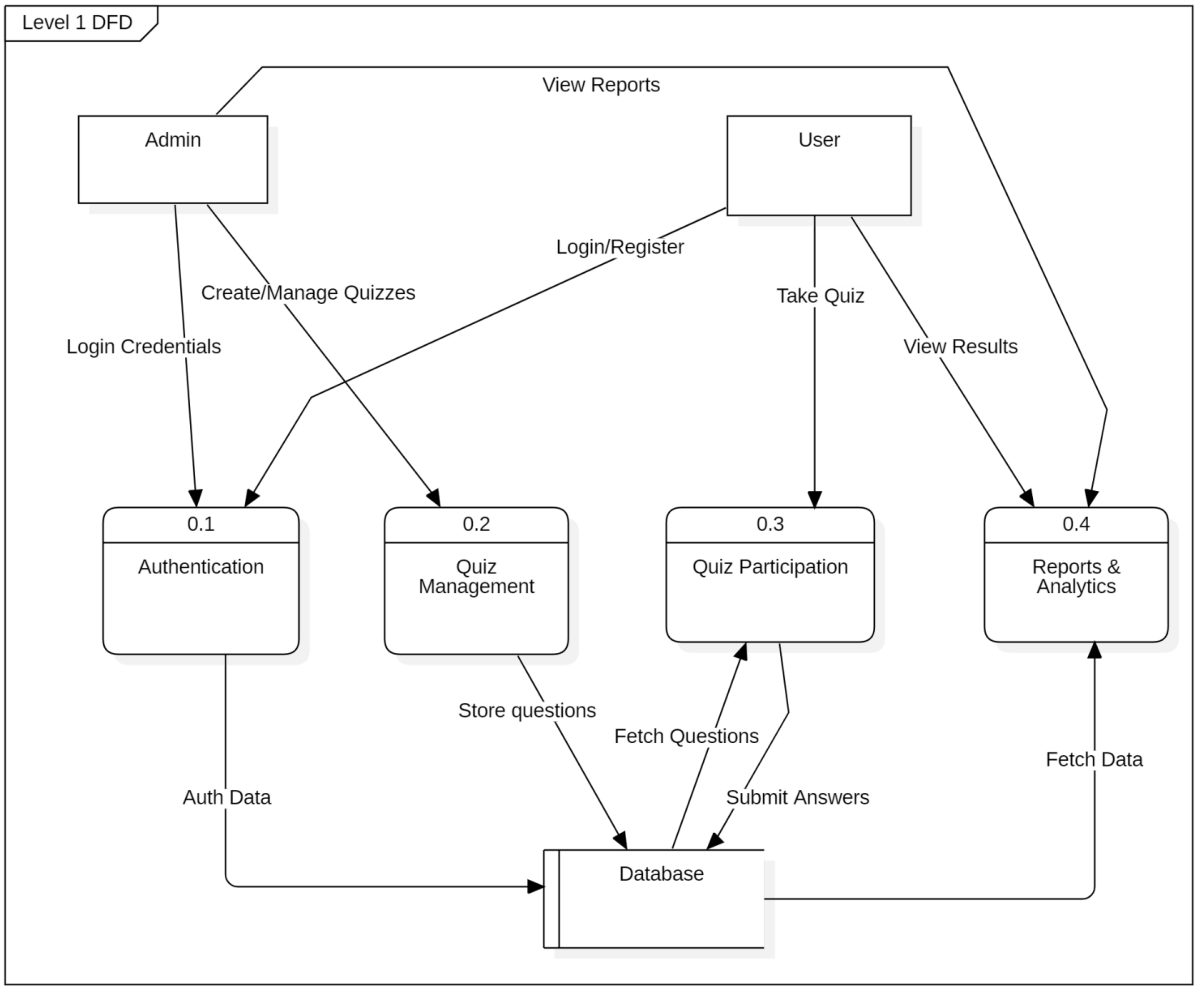
3. System and Database Activities:
   - Store authentication data (credentials).
   - Maintain a question bank with quiz metadata.
   - Store submitted answers and generate results.
   - Provide data for report generation.

DATA FLOW DIAGRAM:

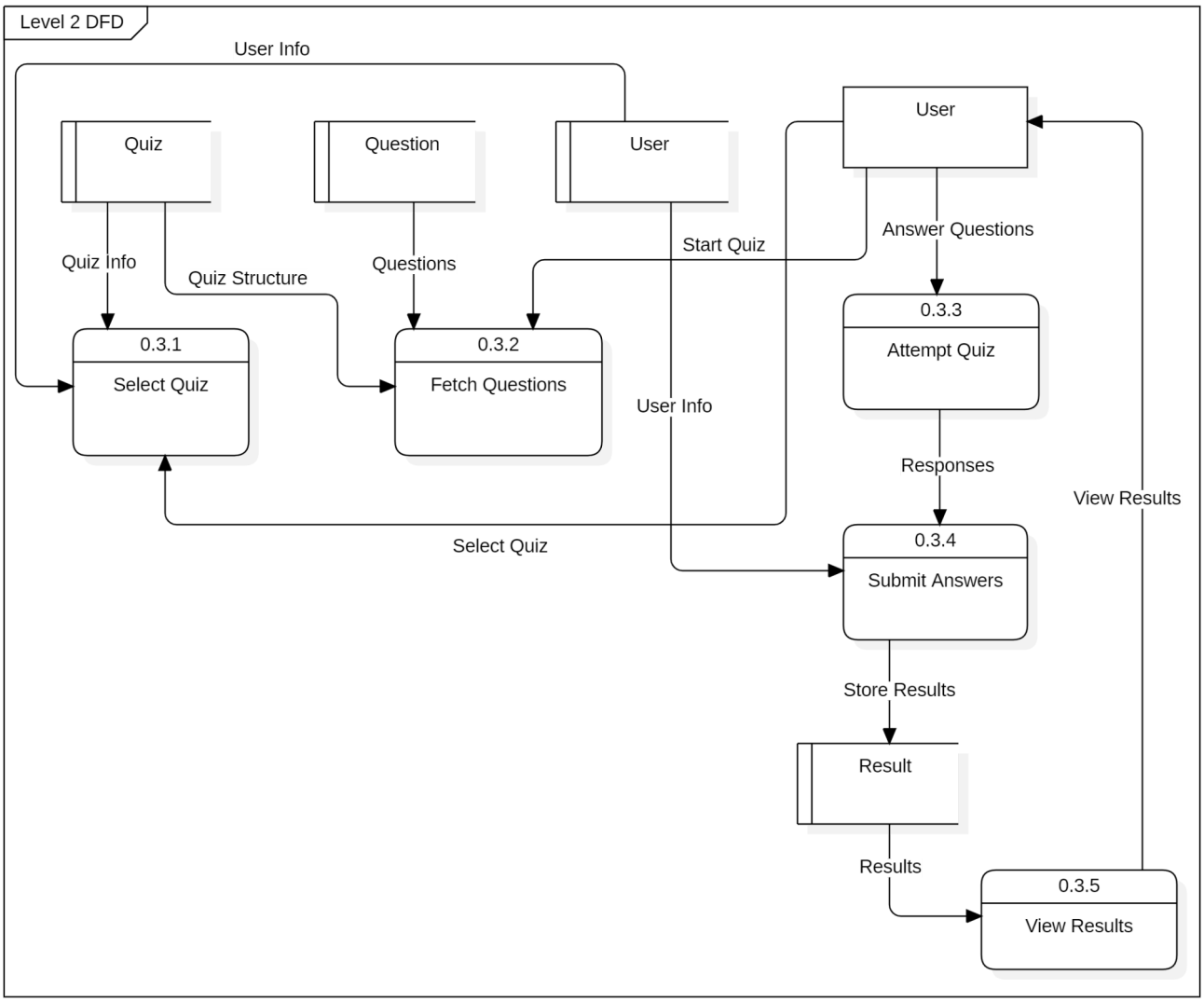
1. Level 0 DFD:



2. Level 1 DFD:


Level 1 DFD
View Reports
Admin
User
Login/Register
Create/Manage Quizzes
Take Quiz
Login Credentials
View Results
0.1
Authentication
0.2
Quiz Management
0.3
Quiz Participation
0.4
Reports & Analytics
Store questions
Fetch Questions
Fetch Data
Auth Data
Submit Answers
Database

3. Level 2 DFD:



STRUCTURE CHART DIAGRAM

IMPLEMENTATION:

A Structure Chart Diagram is a key component of structured analysis and design methodology. It represents the hierarchical decomposition of a system into its functional modules, showing how different parts of the system interact and exchange data or control signals. The main objective of a structure chart is to illustrate the organization of modules and the flow of control between them in a top-down manner.

Unlike a Data Flow Diagram (DFD), which emphasizes data movement across processes, the structure chart focuses on the modular breakdown and the functional relationships between modules. It helps in

understanding how large software systems can be divided into smaller, manageable, and reusable components.

The system is decomposed into four major modules:

1. **Authentication Module** – Manages user registration, login, and session validation.
2. **Quiz Management Module** – Enables quiz creation, question bank management, and quiz assignments.
3. **Quiz Participation Module** – Handles quiz attempts, answer submissions, and scoring.
4. **Reports & Analytics Module** – Generates performance reports and provides analytical dashboards for admins.

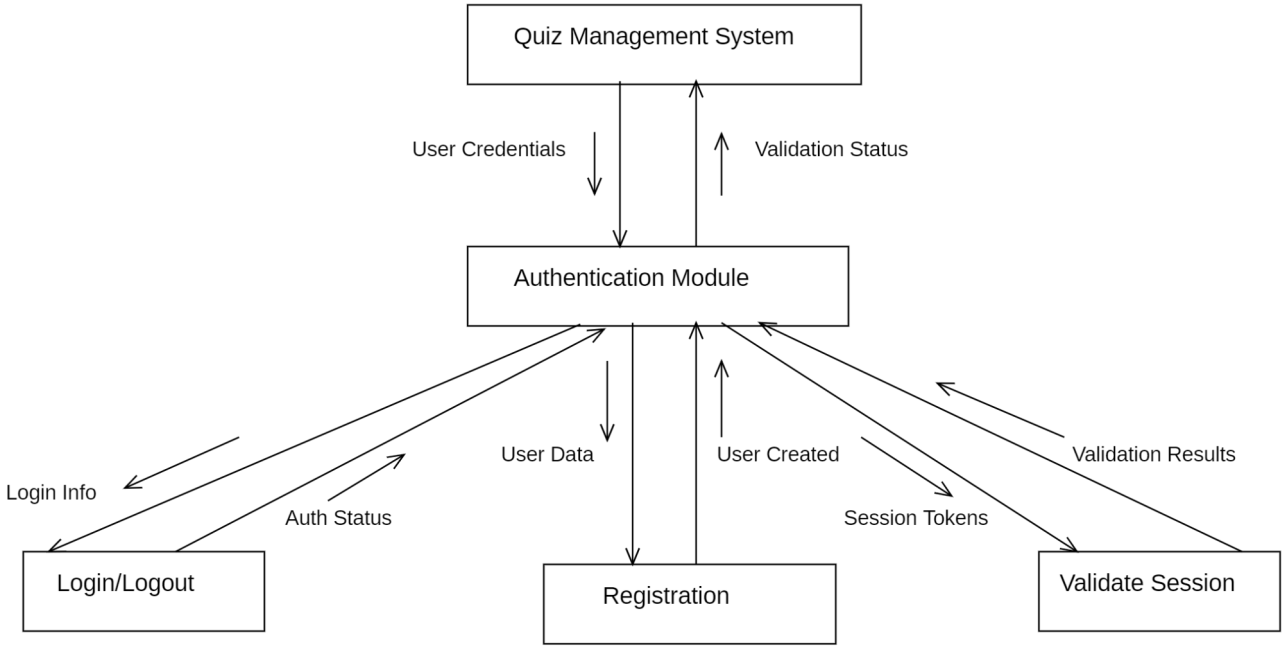
Each module performs a distinct function but interacts through control and data flow for smooth system operation.

Each major module can be further broken down into submodules for greater detail:

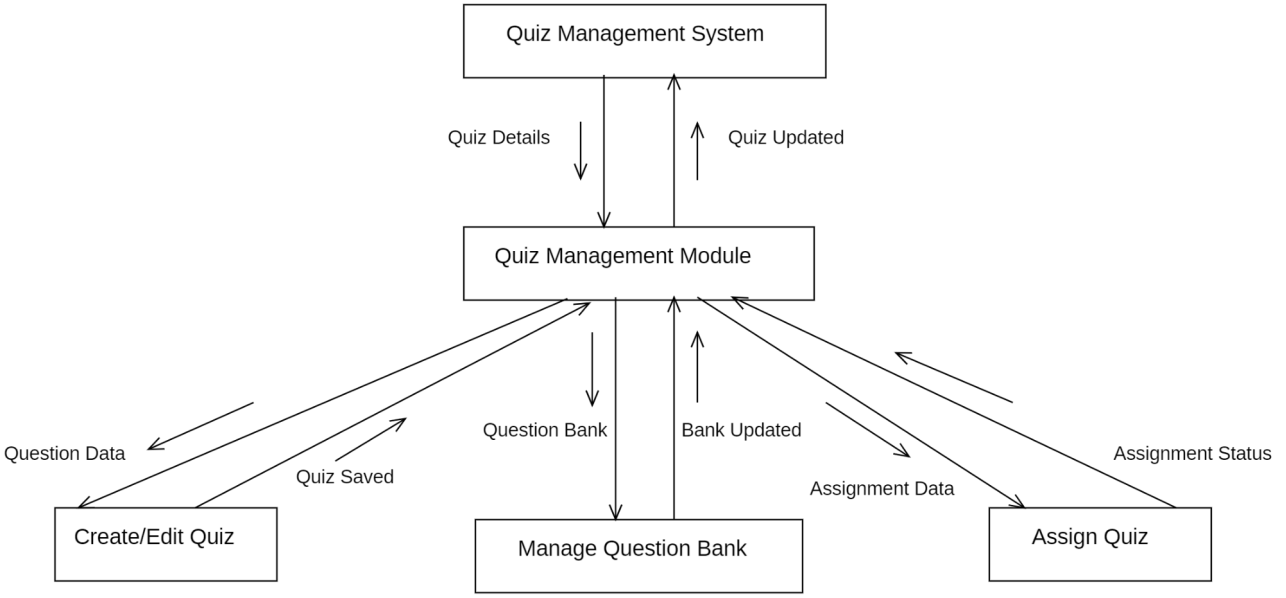
- **Authentication Module**
    1. Login/Logout
    2. Registration
    3. Validate Session
- **Quiz Management Module**
    1. Create/Edit Quiz
    2. Manage Question Bank
    3. Assign Quiz
- **Quiz Participation Module**
    1. Attempt Quiz
    2. Submit Answers
    3. Score Evaluation
- **Reports & Analytics Module**
    1. Generate Report
    2. Analytics Dashboard
    3. Send Alerts

STRUCTURE CHART DIAGRAM:

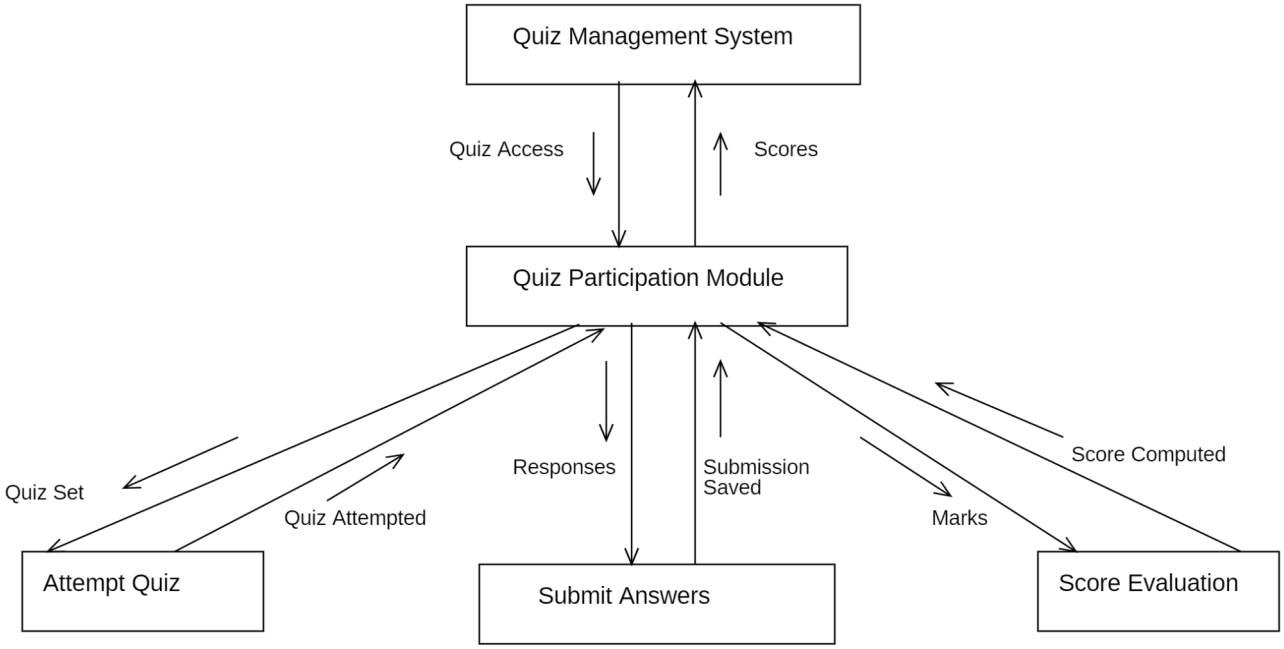
1) Authentication Module:



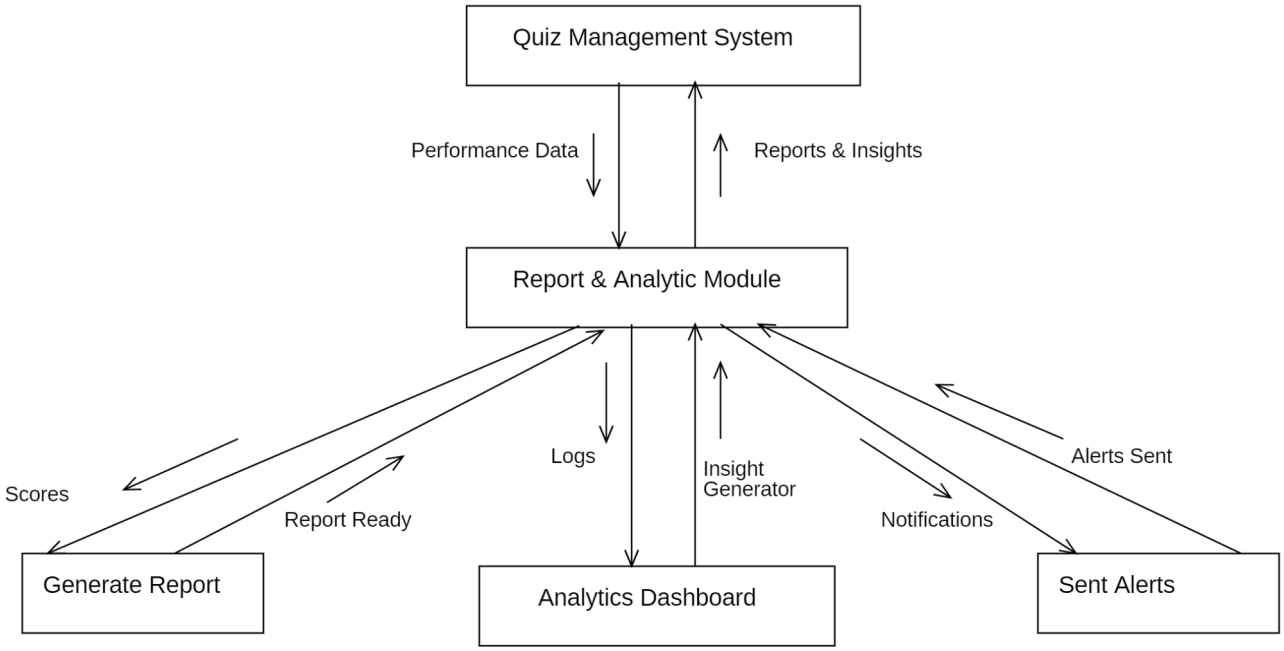
2) Quiz Management Module:



3) Quiz Participation Module:



4) Report & Analytic Module



CONCLUSION:	By performing this experiment, we understood how to represent the Quiz Master System using both Data Flow Diagrams and Structure Charts. The DFDs helped visualize how data moves between users, admins, processes, and the database, while the Structure Chart illustrated the functional hierarchy and control flow between modules. Together, these diagrams provided both data-oriented and function-oriented perspectives of the

	system, helping us clearly understand its internal logic and modular organization. This enhanced our understanding of structured system analysis and efficient software design.

💜 Experiment 7

Experiment No.	7
Group Members	Riya Suryawanshi (2023300240) Anushka Suvarna (2023300241)
Division	D
Batch	B
Date of Submission	1st November 2025

Experiment 7: Implementation

AIM:	To develop 2 modules completely for the Quiz Master software system.

PROJECT REPORT

PROJECT OVERVIEW:

This project aims to build a multi-user exam preparation platform for various courses. It supports two roles: Admin (Quiz Master) and User. As Admins, we can manage subjects, chapters, set quiz questions, and conduct quizzes. Users can register, log in anytime, attempt quizzes, and view scores. The system features a structured database and a user-friendly interface for smooth quiz-taking and management.

METHODOLOGY:

1. **Requirement Analysis:**

   We began by identifying key functionalities for both admin and user roles.

   The user dashboard shows available subjects and their chapters, along with only upcoming quizzes (based on scheduled dates). Users can view past scores and graphical performance summaries—rendered using dynamic

   Vue charts via backend APIs—to track progress and monthly quiz activity. The admin dashboard enables full control over the database, including adding subjects, creating chapters, setting quizzes with timed multiple-choice questions, and viewing registered users. Time limits help users balance speed and accuracy.

   Admins also get analytics of user performance using a CSV exported to their email.

2. **Back-End Development:**

   We used Flask to build the backend that provides JSON responses to the frontend through RESTful APIs.

   Business Logic for the Backend:

   1. Admin details are already put into the database before the server is started. Admin can create, edit, or delete subjects, chapters, quizzes, and questions through protected API endpoints.
   2. All endpoints are protected using @token_required decorator that verifies JWT tokens. This ensures users can't spoof requests or access unauthorized routes.
   3. The admin can edit the subjects/chapters/quizzes/questions by entering new data. The request then fetches the entry that needs to be edited.
   4. Users can only attempt each quiz once and only on the scheduled date. Submissions are validated, scored, and stored in the Scores table.
   5. On clicking "Start Quiz," quiz and question details are fetched from related tables. The quiz timer (from DB) begins, and the user can submit early or automatically when time runs out. The submission time is logged in the Scores table. After submission, users are directed to the Scores Page to view the scores and the answer key for the quizzes taken.
   6. Past or missed quizzes are hidden from the user dashboard. The button to start a quiz is only enabled on the quiz day.
   7. Performance charts are built using matplotlib, with bar graphs plotting top scores per subject using join queries. Charts are converted to Base64 and embedded in the front end.
   8. Celery background tasks involve:
      - Daily reminders for a quiz through email. The remainder of the quiz is only sent if the user remains inactive on the portal on the quiz day.
      - Monthly performance summaries. This involves sending a mail having the performance summary of the previous month in a HTML format
      - User performance stats sent to the admin in CSV format
   9. Redis caching speeds up commonly accessed endpoints (user performance, subject chapter listings, quiz details). Cache auto-invalidates on updates or after 1–2 minutes, depending on the data's sensitivity.

3. Authentication System:

   The app uses JWT-based authentication for users and admins. On login, a secure token is issued and required in the Authorization header for all protected routes. Registration and login are public routes, and passwords are hashed with 8 salting rounds before storage. If an existing

user tries to re-register, they get a warning at the frontend. After registering, the user is redirected to the login page. Logout clears the session. This system ensures security and privacy by restricting access to protected routes unless a valid session or token is present. Token expiry is managed server-side, and the setup supports future addition of refresh tokens. Overall, the stateless JWT system makes the API more secure and scalable.

4. **Front-End Development:**

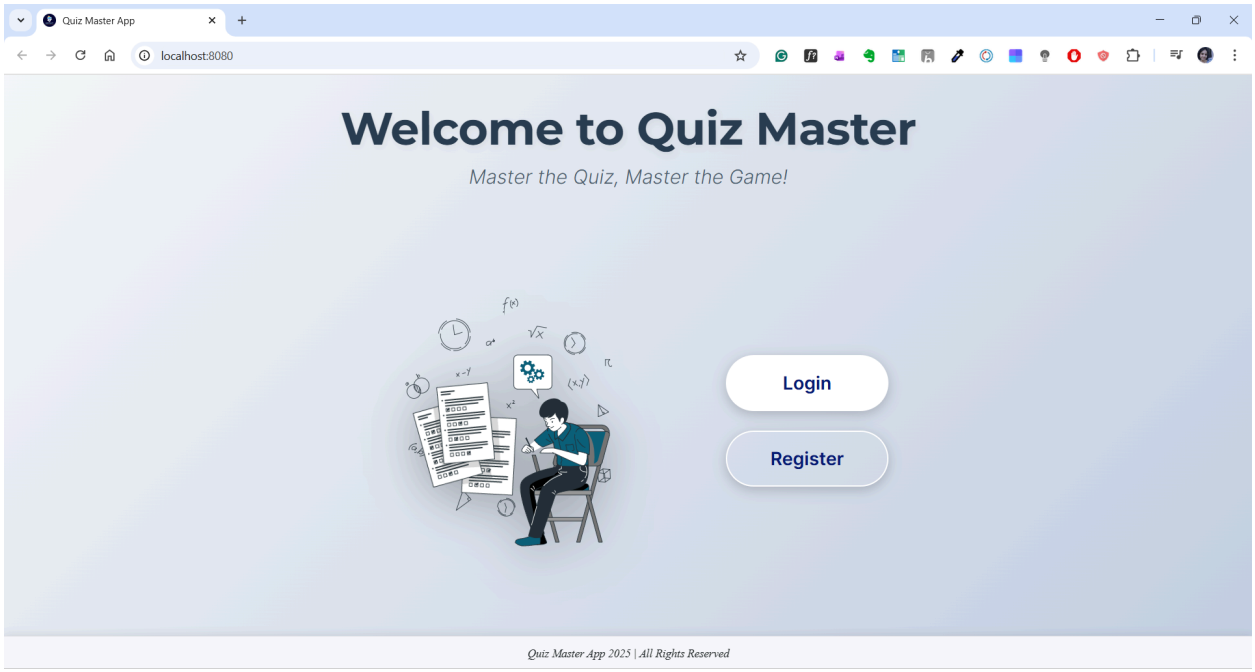
The frontend was completely rebuilt using Vue.js to enhance responsiveness and interactivity. The frontend interacts securely with the Flask backend using API calls and supports smooth navigation, quiz-taking, and performance analytics. For navigation and state handling, we used Vue Router to manage page transitions smoothly. In terms of API communication, JWT tokens are stored in localStorage and attached to headers to securely access protected routes. The UI is styled using Bootstrap, ensuring a responsive and clean interface across devices. Key Vue components handle features like a quiz timer synced with backend time limits, dynamic charts for user analytics, and form validation, which is processed through the backend to ensure data integrity. Additionally, role-based routing was implemented to differentiate between user and admin dashboards. This setup ensures a seamless, secure, and user-friendly front-end experience.

TECH STACK:

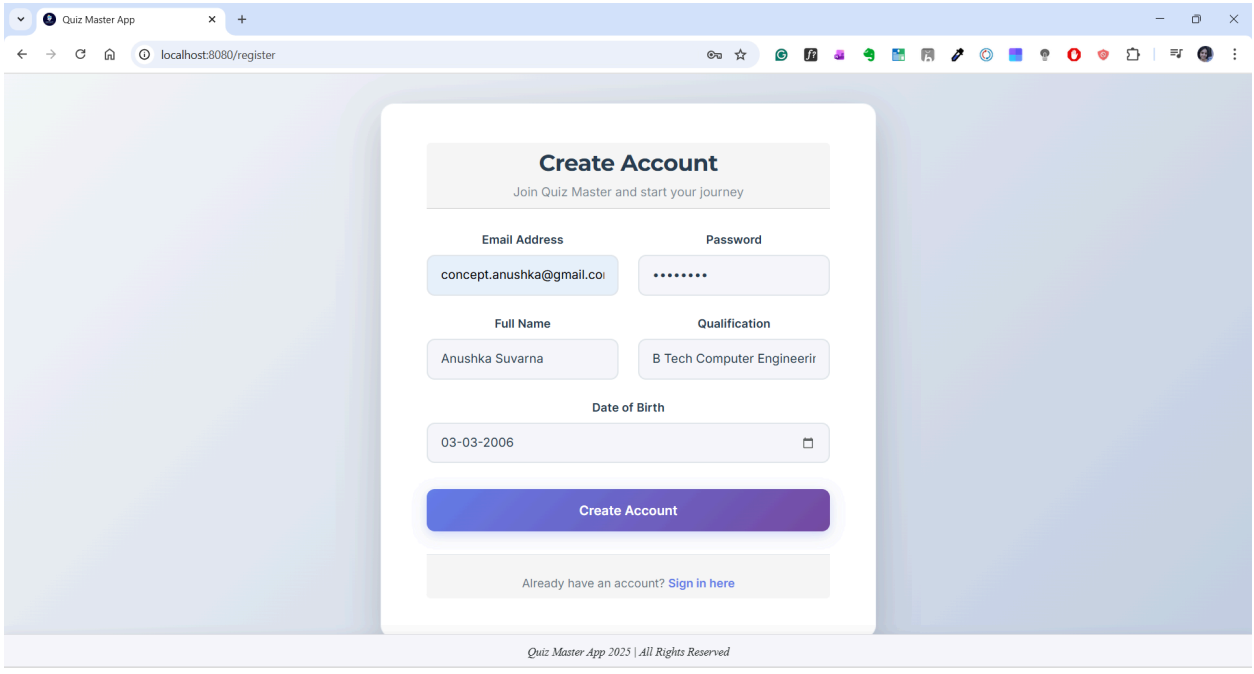
1. Flask – Web framework for routing and backend logic
2. Flask-Caching – For Redis-based caching
3. Flask-Mail – Email notifications
4. Celery + Redis – For background jobs and cache
5. JWT (PyJWT) – Secure token-based authentication
6. SQLAlchemy 2.0 – ORM and model relationship management
7. PostgreSQL – Database
8. Vue.js (Using CLI)– Reactive frontend framework
9. Bootstrap & CSS – Styling and responsiveness
10. Matplotlib (via backend) – For performance visualizations

DEMO SCREENSHOTS

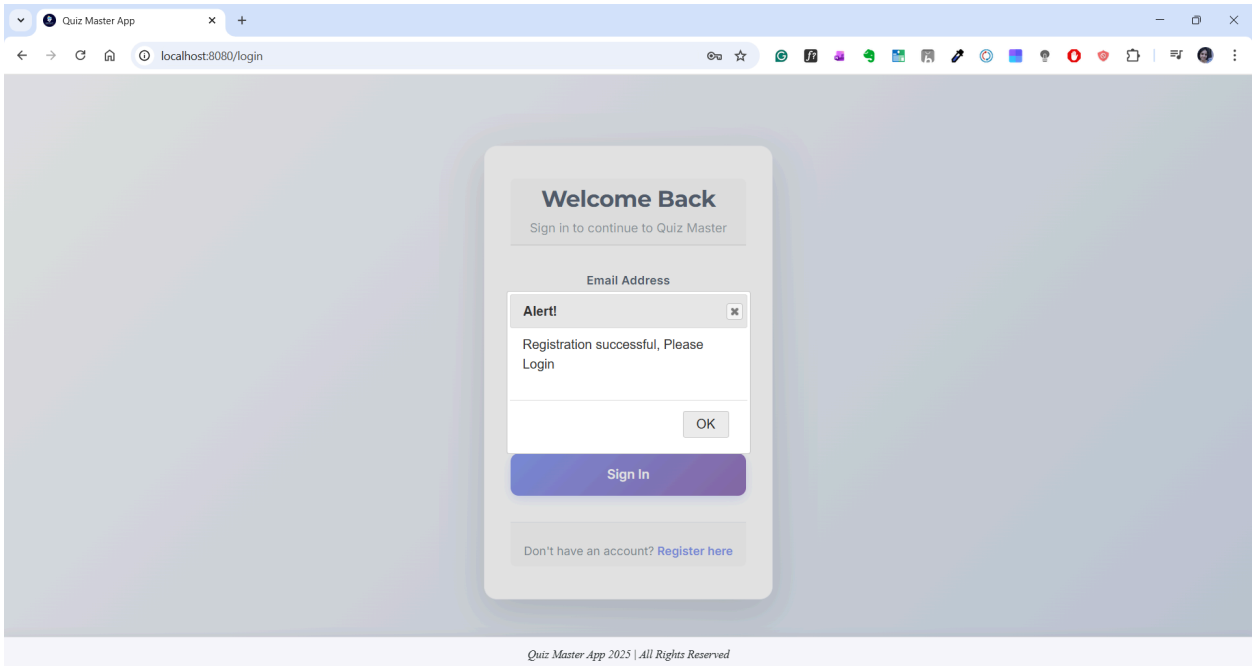
1. Home Page: The main landing page of the Quiz Master platform, providing easy navigation to sign up and login.



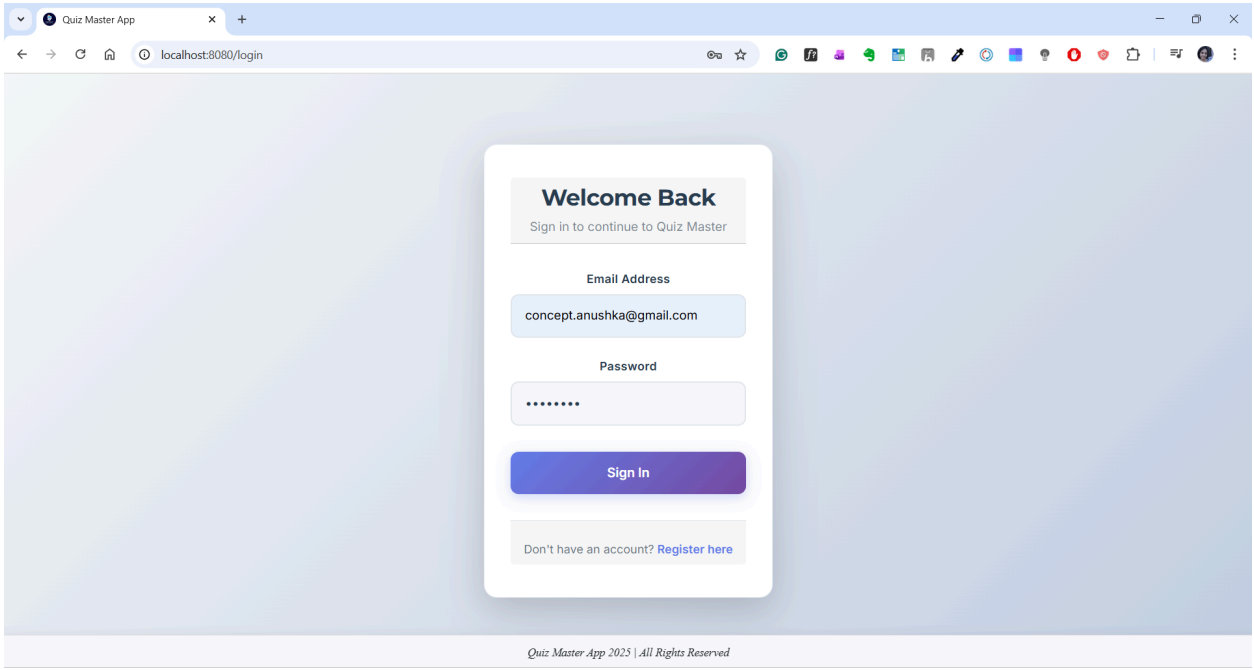
2. Sign Up Page: Registration form for new users to create an account by providing essential details.



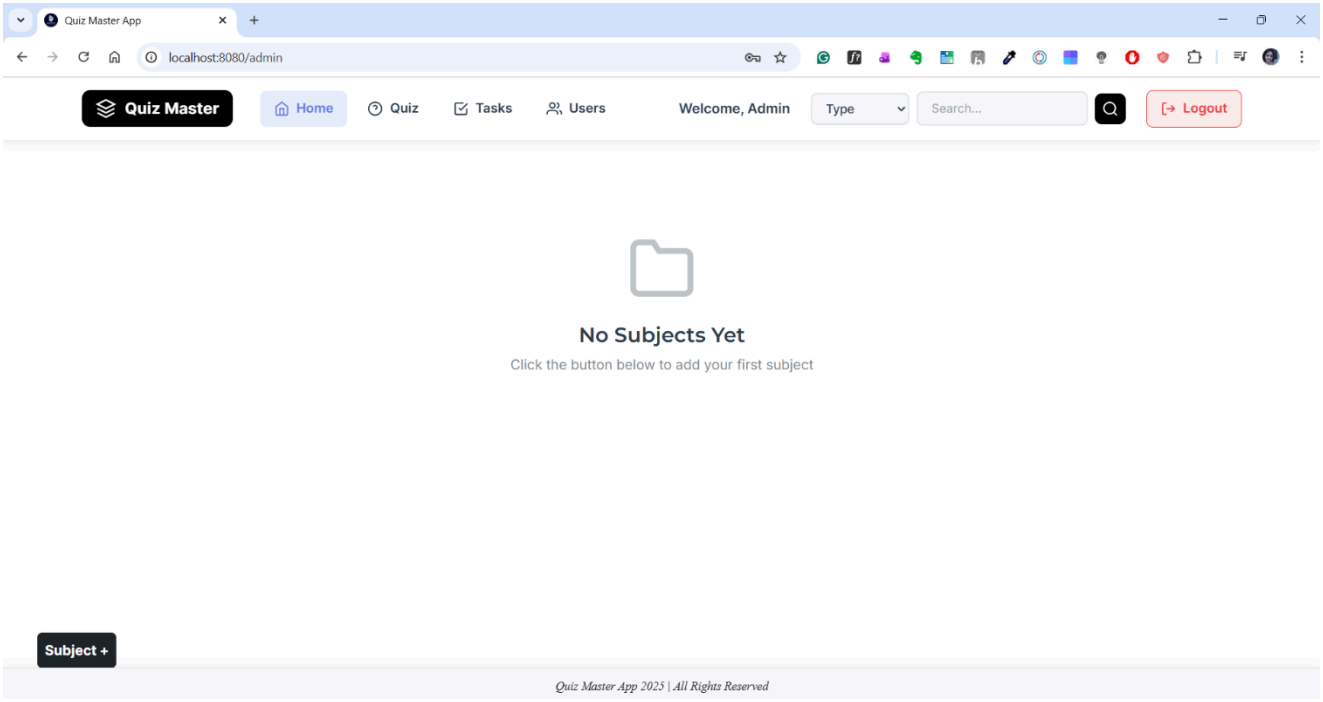
Login is required after sign up



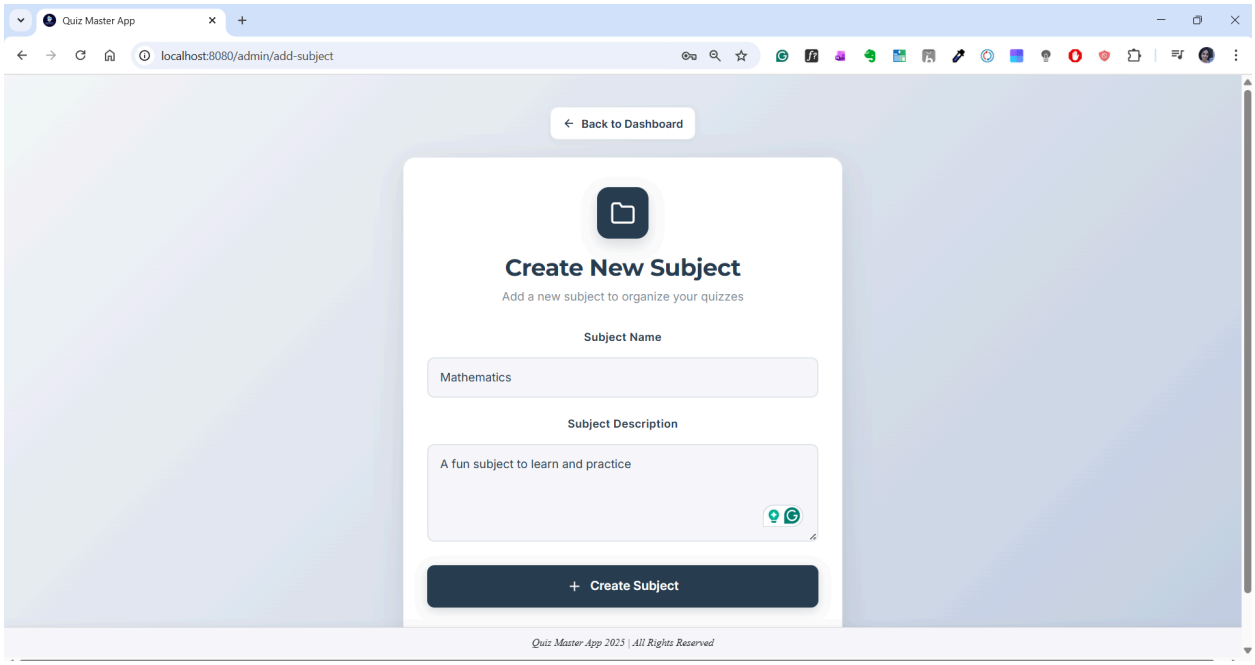
3. Login Page: Secure login interface where existing users can enter credentials to access their dashboard.

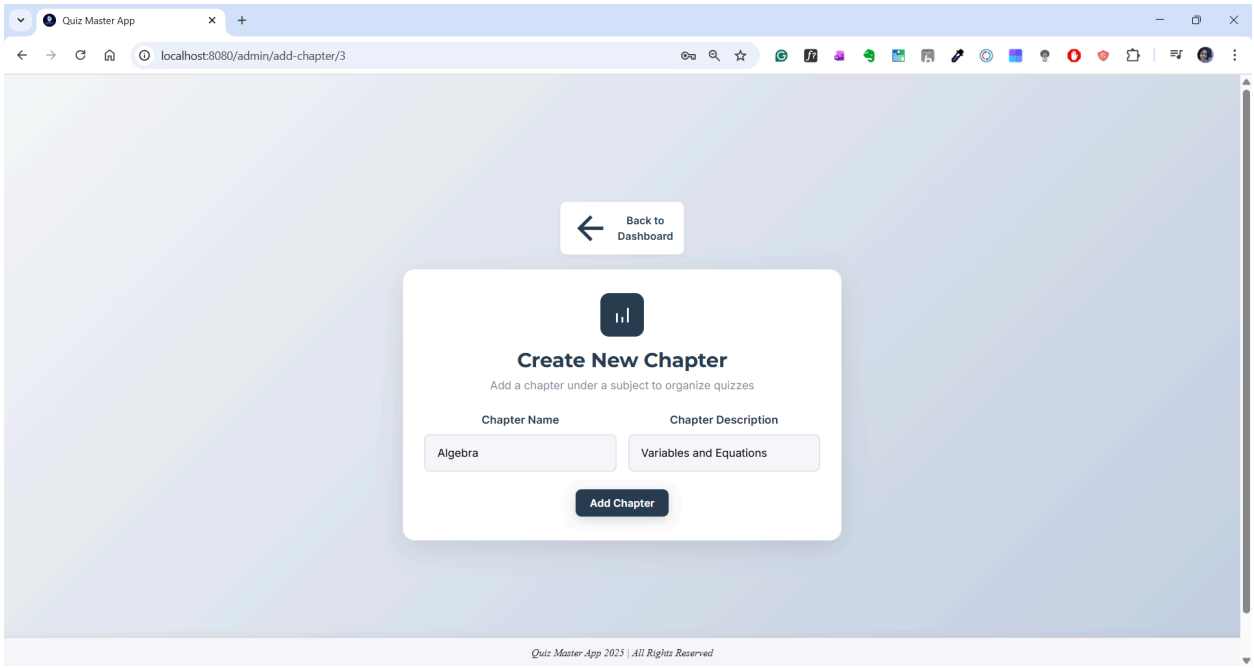


4. Admin Dashboard: Centralized admin panel displaying subjects, quizzes, chapters, tasks, and management options.

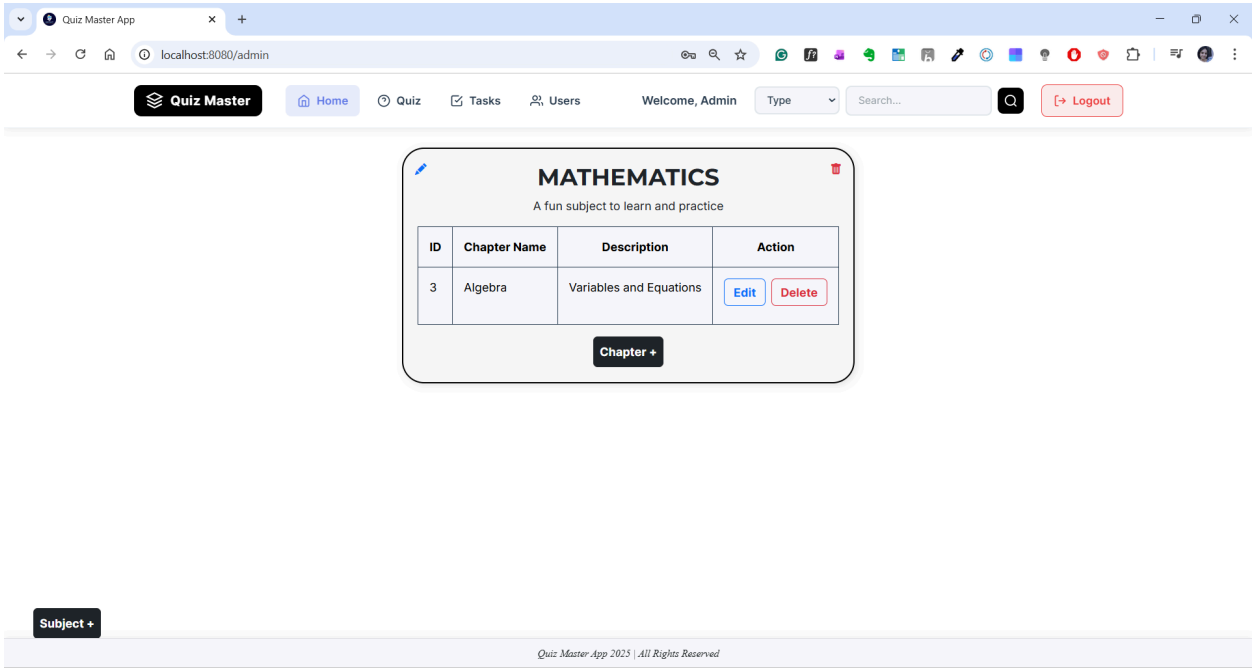


5. Create Subject/Chapter: Form interface for admins to add new subjects and chapters to organize quizzes.

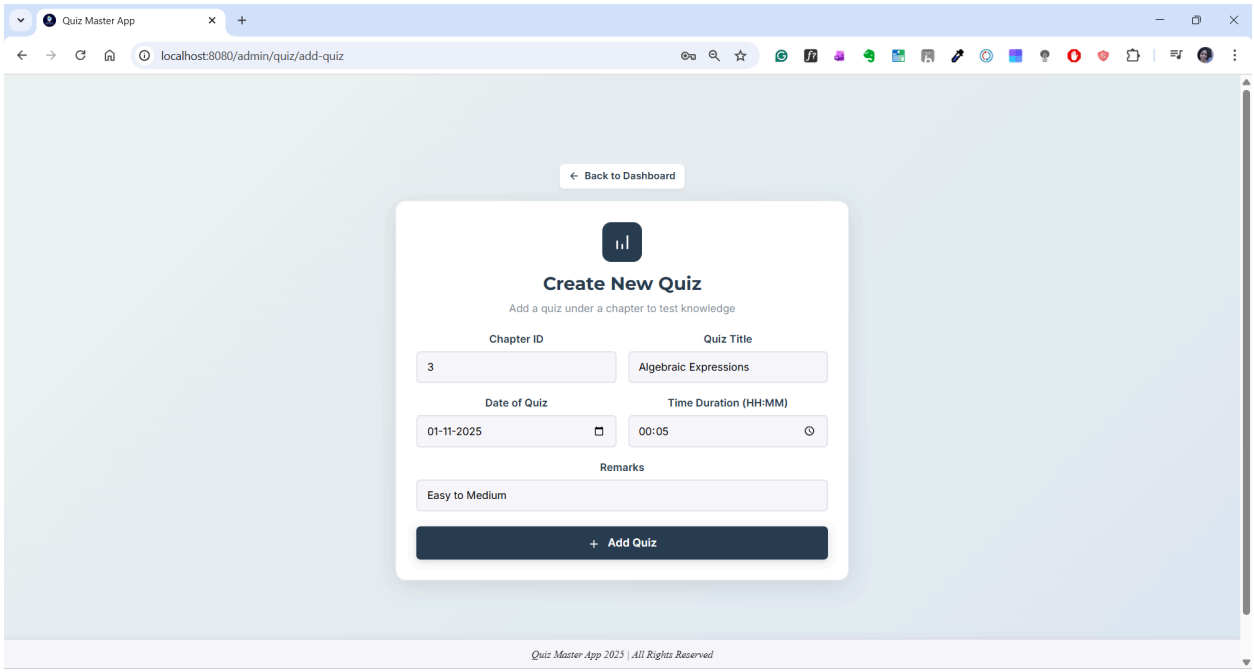


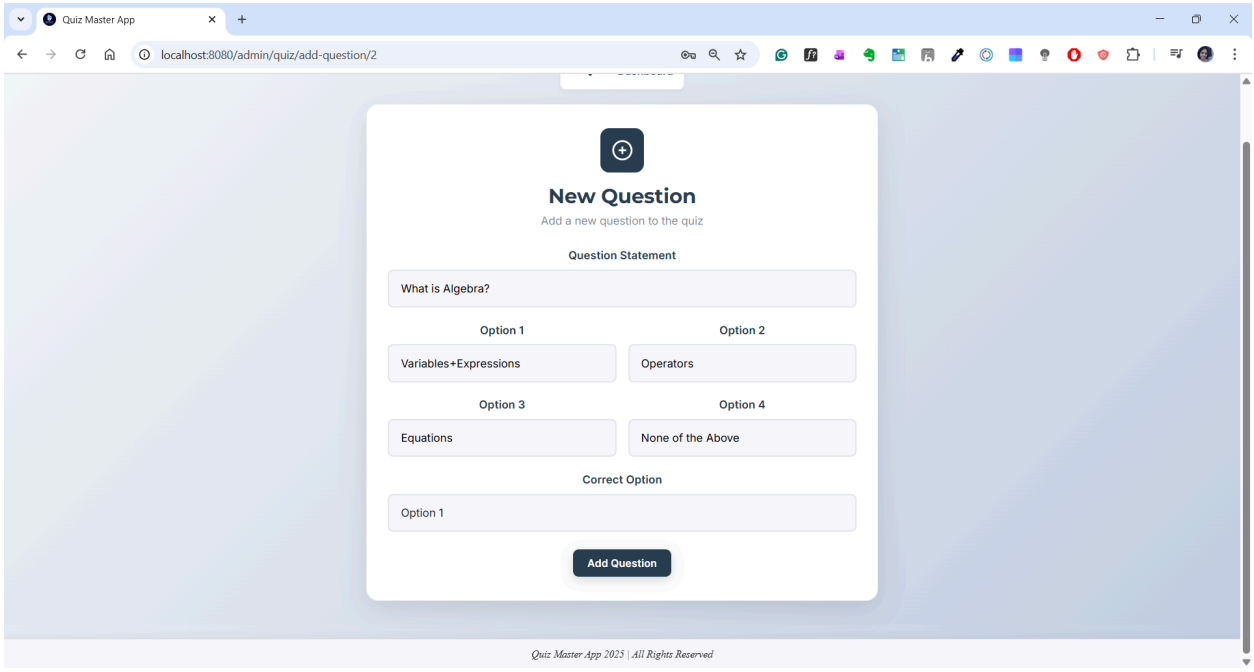


6. Subject and Chapter Created: Admin Dashboard showing successful creation of subjects and chapters.

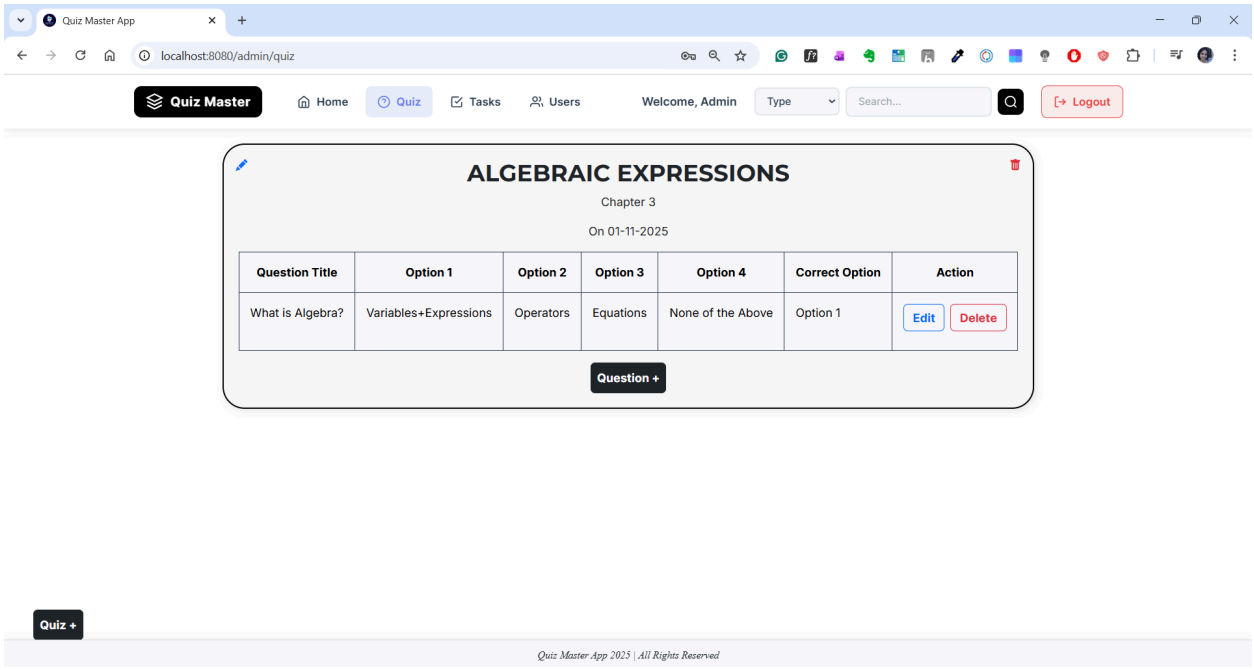


7. Create Quiz/Question: Interface allowing admins to design quizzes and add individual questions with options.

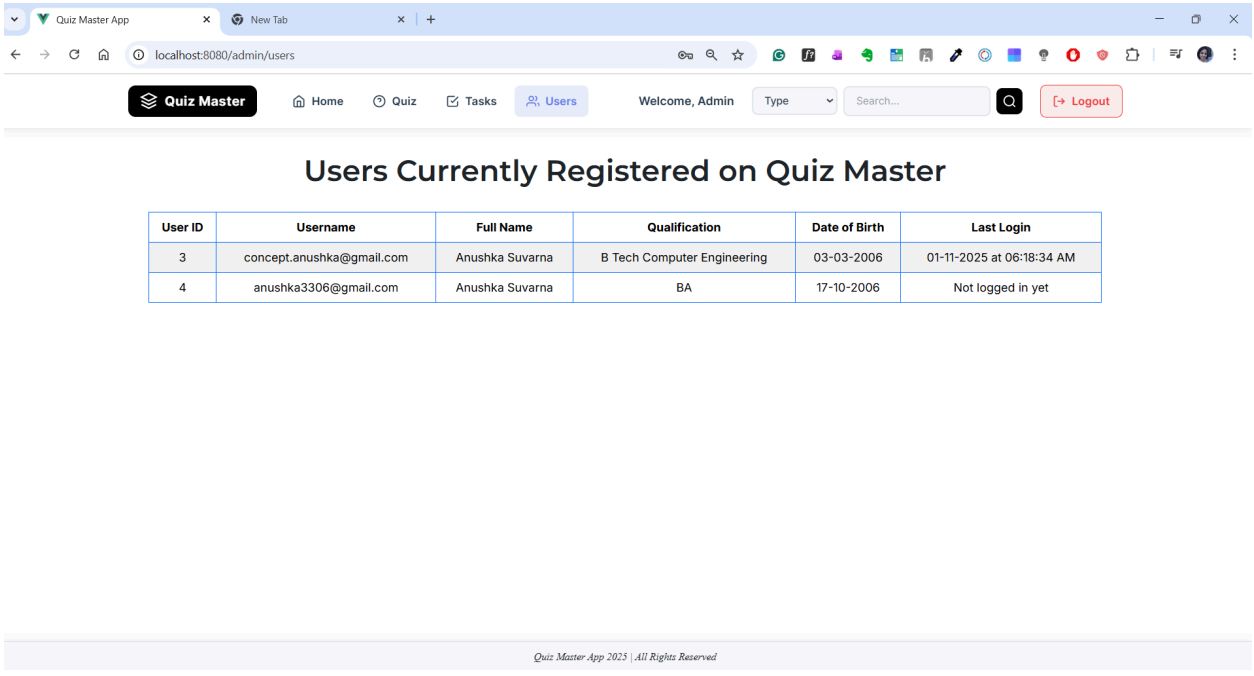




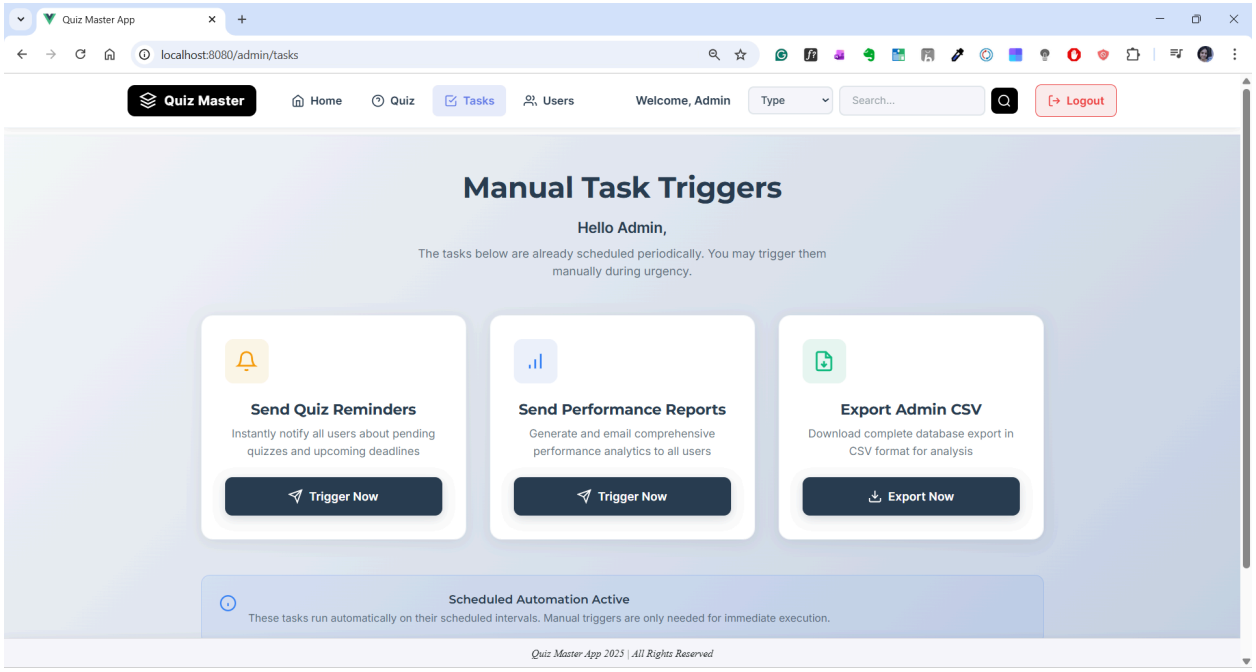
8. Quiz/Question created: Dashboard showing that the quiz or question has been successfully added.



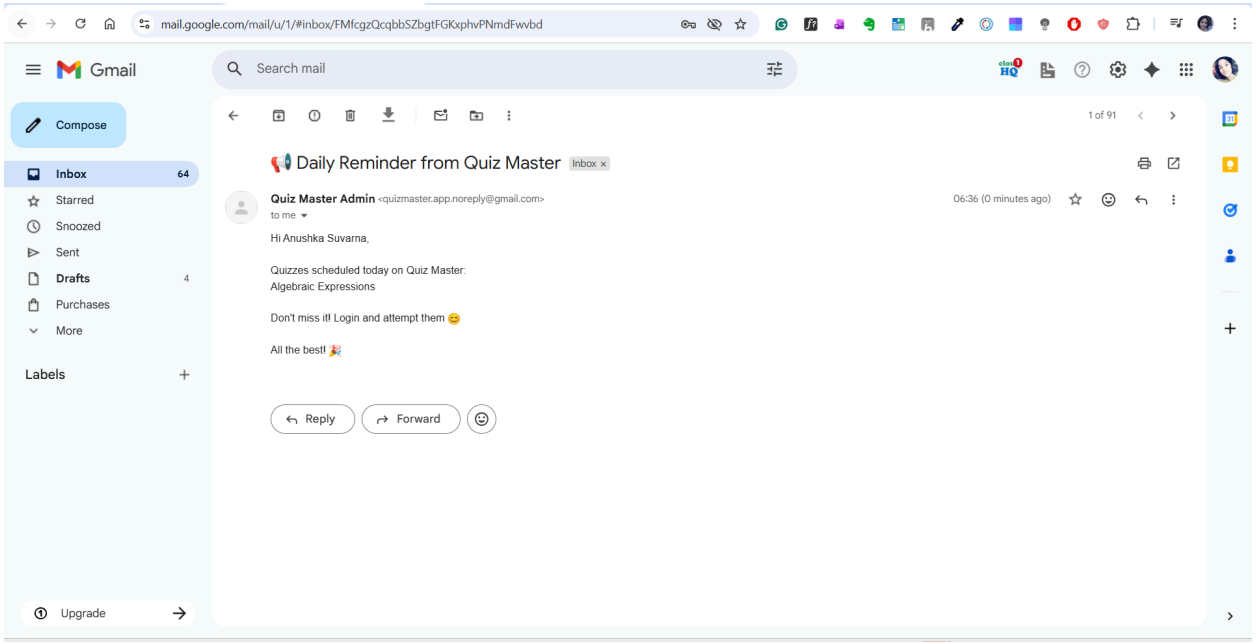
9. List of registered Users Visible to Admins: Admin view listing all users registered on the platform with details.



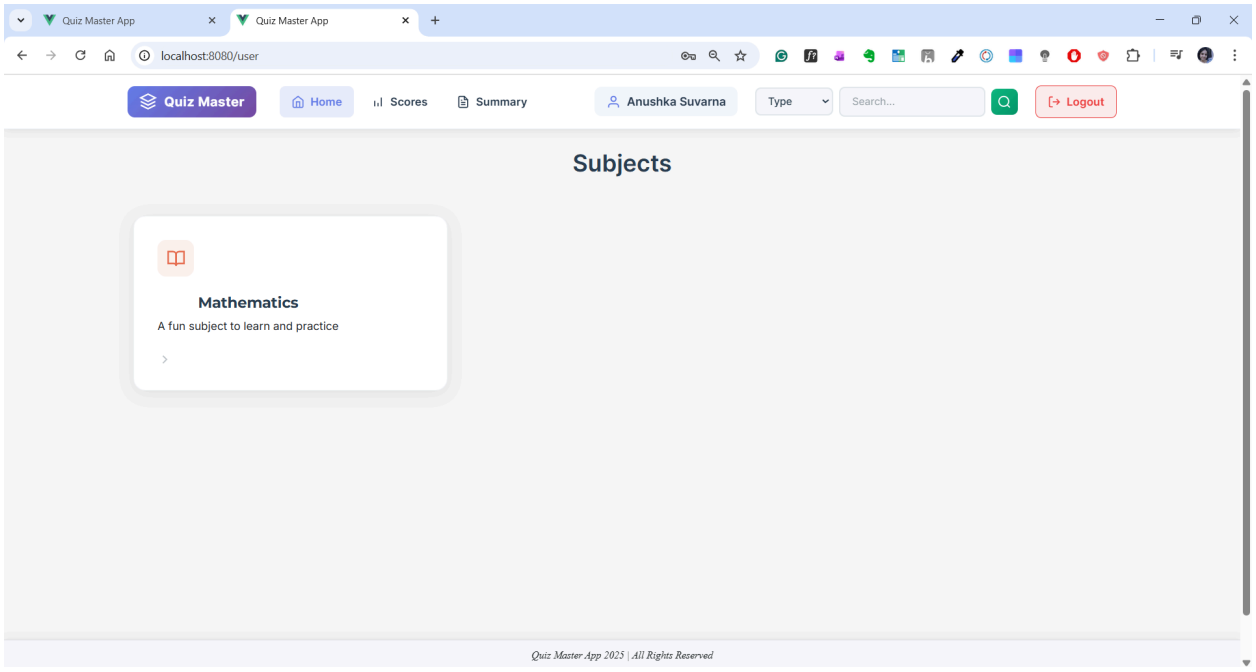
10. Admin Tasks: Overview of all admin tasks which can be manually triggered, including sending quiz reminders and performance reports to users through email, and exporting a user analytics CSV file onto the admin's email ID

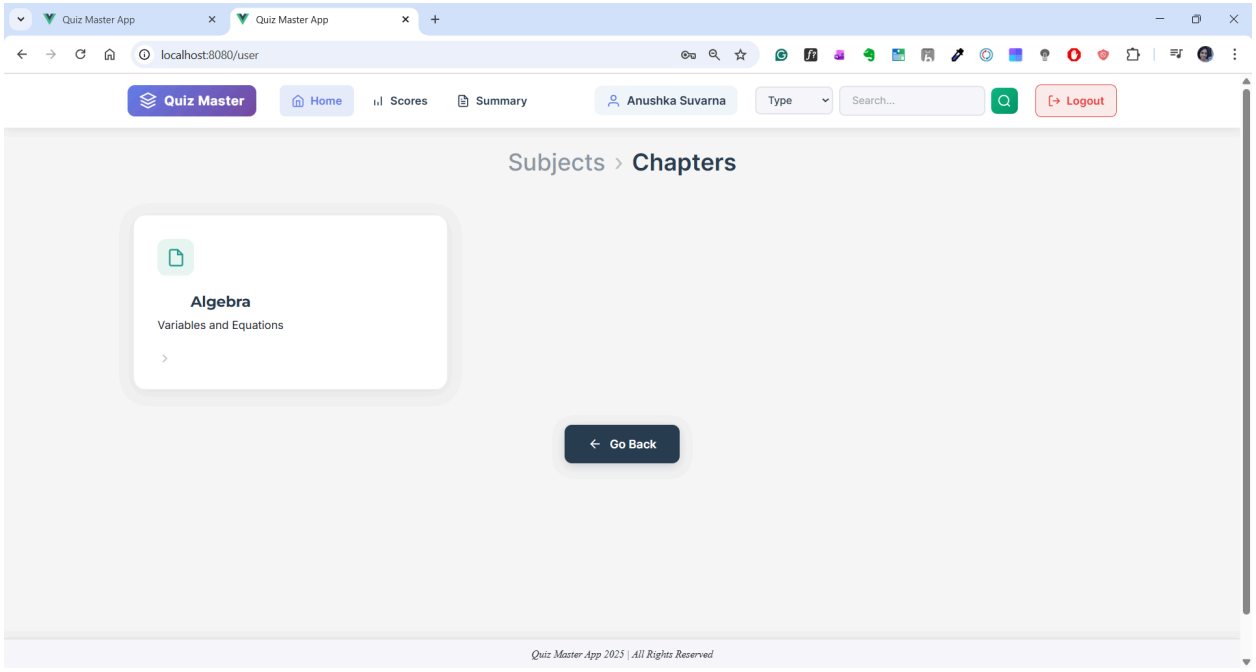


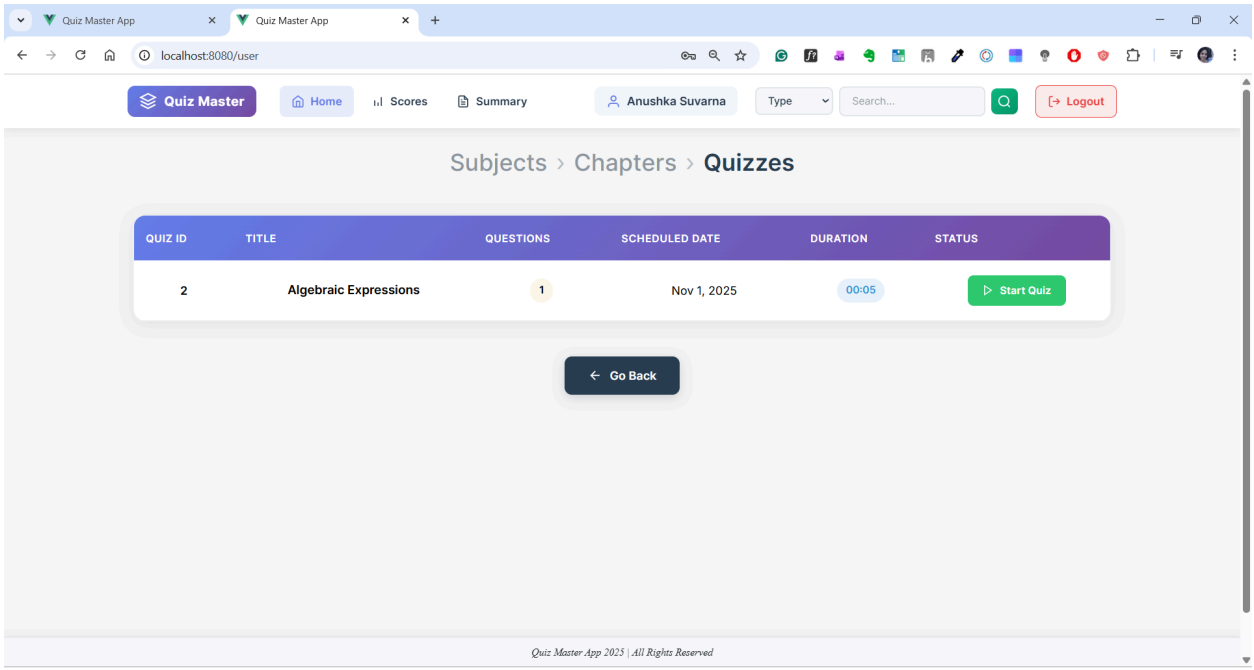
11. Quiz Reminder sent to inactive user: Email showing that reminders have been sent to users who haven't attempted quizzes.



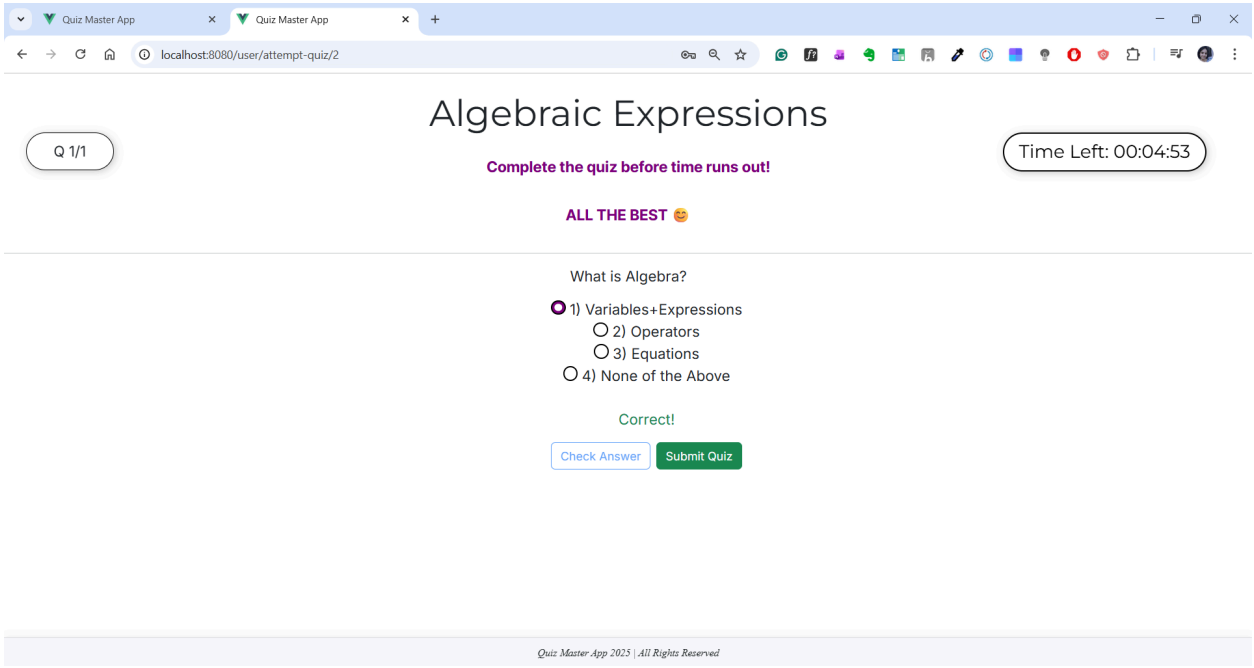
12. User Dashboard: Personalized dashboard for users displaying available quizzes and progress.



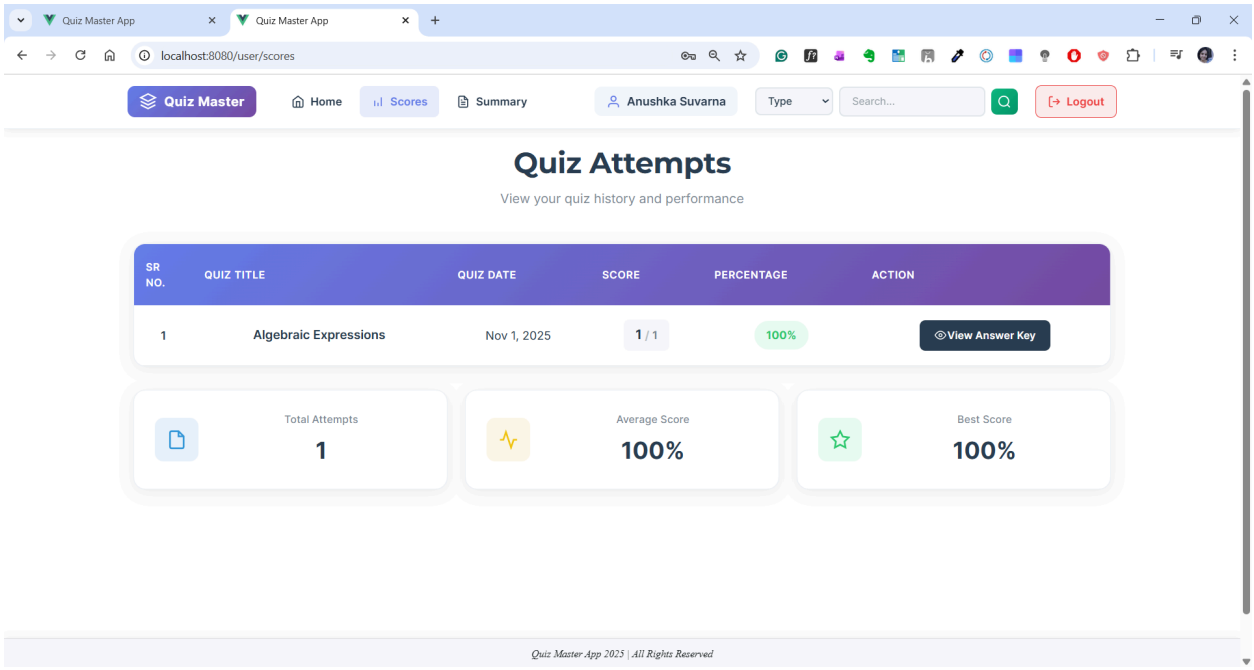




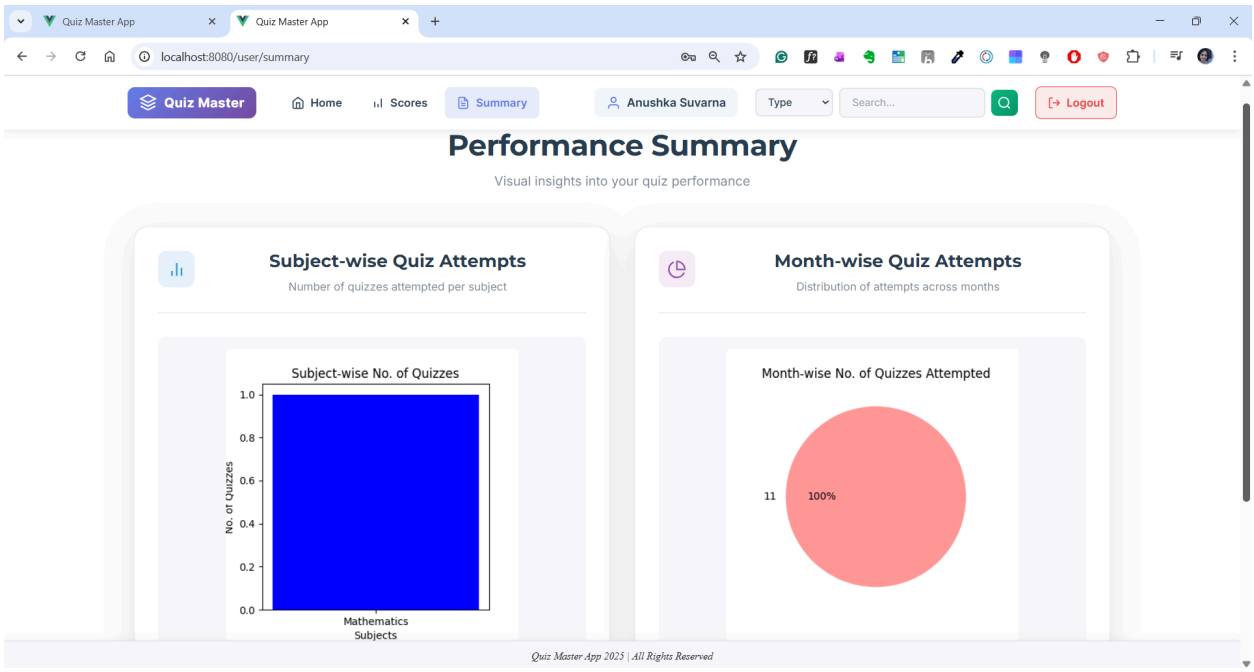
13. Attempting Quiz: Interface where users actively take a quiz, view questions, and submit answers with a timer being ON.



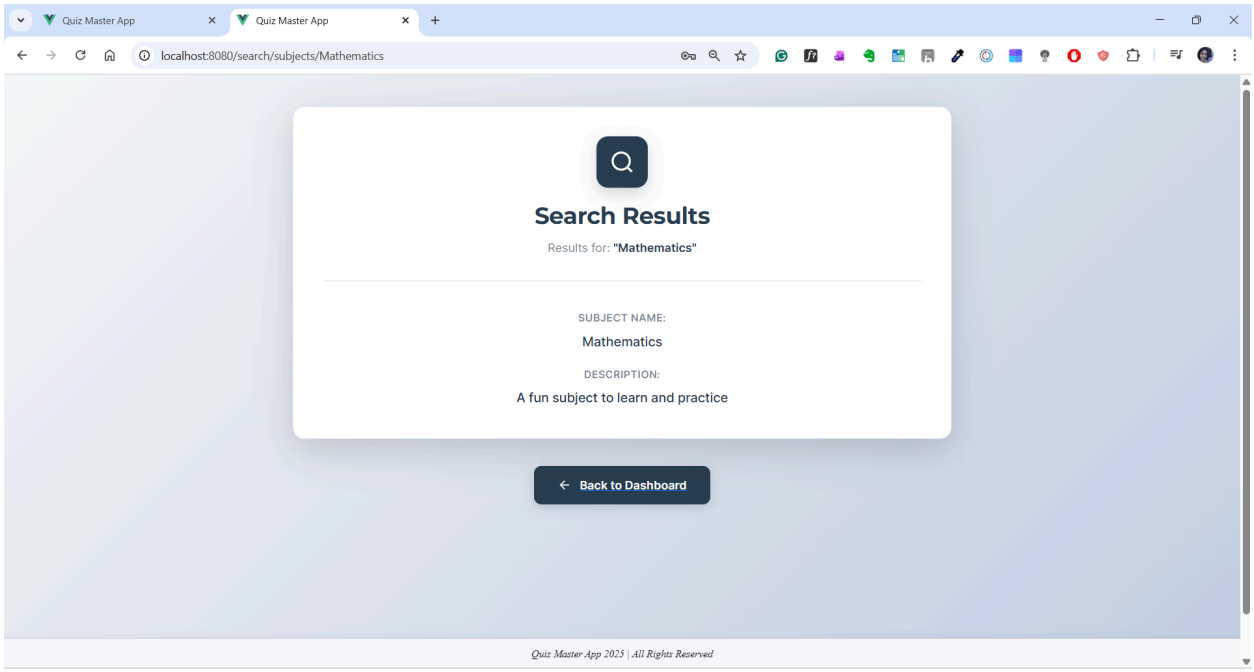
14. Scores tab: Displays detailed scores of completed quizzes along with theirs answer keys, allowing users to track their performance.



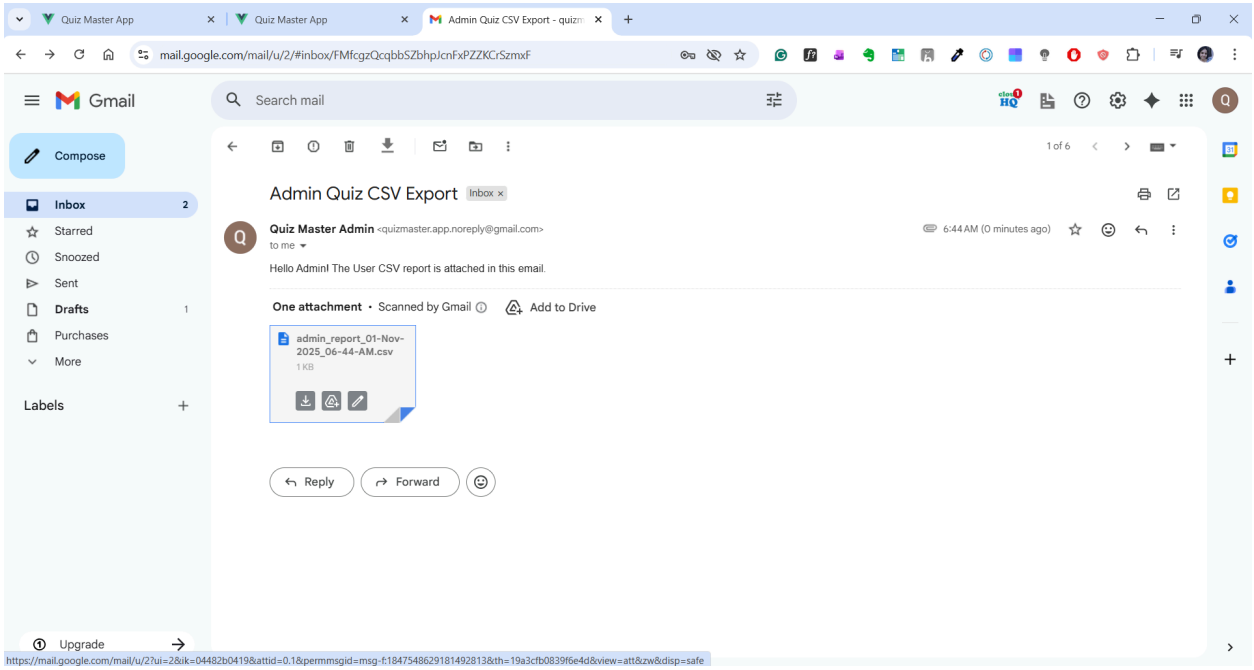
15. Performance analytics in User Dashboard: Graphs and charts summarizing user performance trends over time.



16. Search Results: Page showing quizzes, subjects, or chapters matching user search queries for easy access.



17. Export user performance report to admin's email ID: Functionality for admins to export user performance reports, which are sent directly to their email.



CSV Report sent:

User ID	Username	Full Name	Quizzes Taken	Average Score

5	anushka3306@gmail.com	Anushka Suvarna	0	0
4	concept.anushka@gmail.com	Anushka Suvarna	1	1

CONCLUSION: The Quiz Master application demonstrates a comprehensive platform for both admins and users to manage, attempt, and analyze quizzes efficiently. Admins can create subjects, chapters, quizzes, and questions, manage registered users, send reminders, and export performance reports, ensuring smooth operations. Users benefit from a personalized dashboard, easy quiz attempts, performance tracking, and analytics that help monitor progress over time. Overall, the system successfully integrates quiz management, user engagement, and performance evaluation into a single, intuitive platform, highlighting its effectiveness and usability for educational and training purposes.

🤩 Expermient 8

Experiment No.	8
Group Members	Riya Suryawanshi (2023300240) Anushka Suvarna (2023300241)
Division	D
Batch	B
Date of Submission	2nd November 2025

Experiment 7: Work Breakdown Structure & Gantt Chart

AIM:	To design and draw a Work Breakdown structure based on SDLC phases. and a Gantt chart for the Quiz Master software system.

WORK BREAKDOWN STRUCTURE

IMPLEMENTATION:

A Work Breakdown Structure (WBS) is a hierarchical project management tool used to break down a complex project into smaller, more manageable components. It helps in organizing the project scope by dividing deliverables and tasks into clear phases and sub-tasks, enabling better planning, scheduling, and tracking of progress.

A WBS focuses on project execution — defining what work needs to be done, by whom, and in what sequence. It forms the foundation for project scheduling, cost estimation, and resource allocation, often represented visually in tree or tabular form or integrated into a Gantt chart.

STRUCTURE:

The WBS for the Quiz Master project is designed based on the Software Development Life Cycle (SDLC) phases. Each phase represents a major deliverable, which is further broken down into specific, actionable tasks.

SDLC PHASES:

1. **Feasibility Study**
   This phase involved analyzing the technical, operational, and economic feasibility of developing the Quiz Master system.

- Study of existing quiz platforms.
- Evaluation of available technologies (Flask, Vue.js, SQLite).
- Preparation of feasibility report summarizing conclusions.

2. **Requirements Analysis and Specification**
   Focused on identifying user and admin needs and documenting them clearly in a Software Requirements Specification (SRS).
   - Gathering user/admin requirements.
   - Defining functional and non-functional requirements.
   - Preparing the SRS document.

3. **System Design**
   This phase translated requirements into a blueprint for implementation.
   - Designing the user interface (UI/UX mockups).
   - Creating the database schema and entity relationships.
   - Designing architecture diagrams and defining data flow.

4. **Coding and Unit Testing**
   The actual development of the system was carried out in this stage.
   - Implementing backend using Flask and SQLite.
   - Developing the frontend interface in Vue.js.
   - Performing unit tests on individual components to verify functionality.

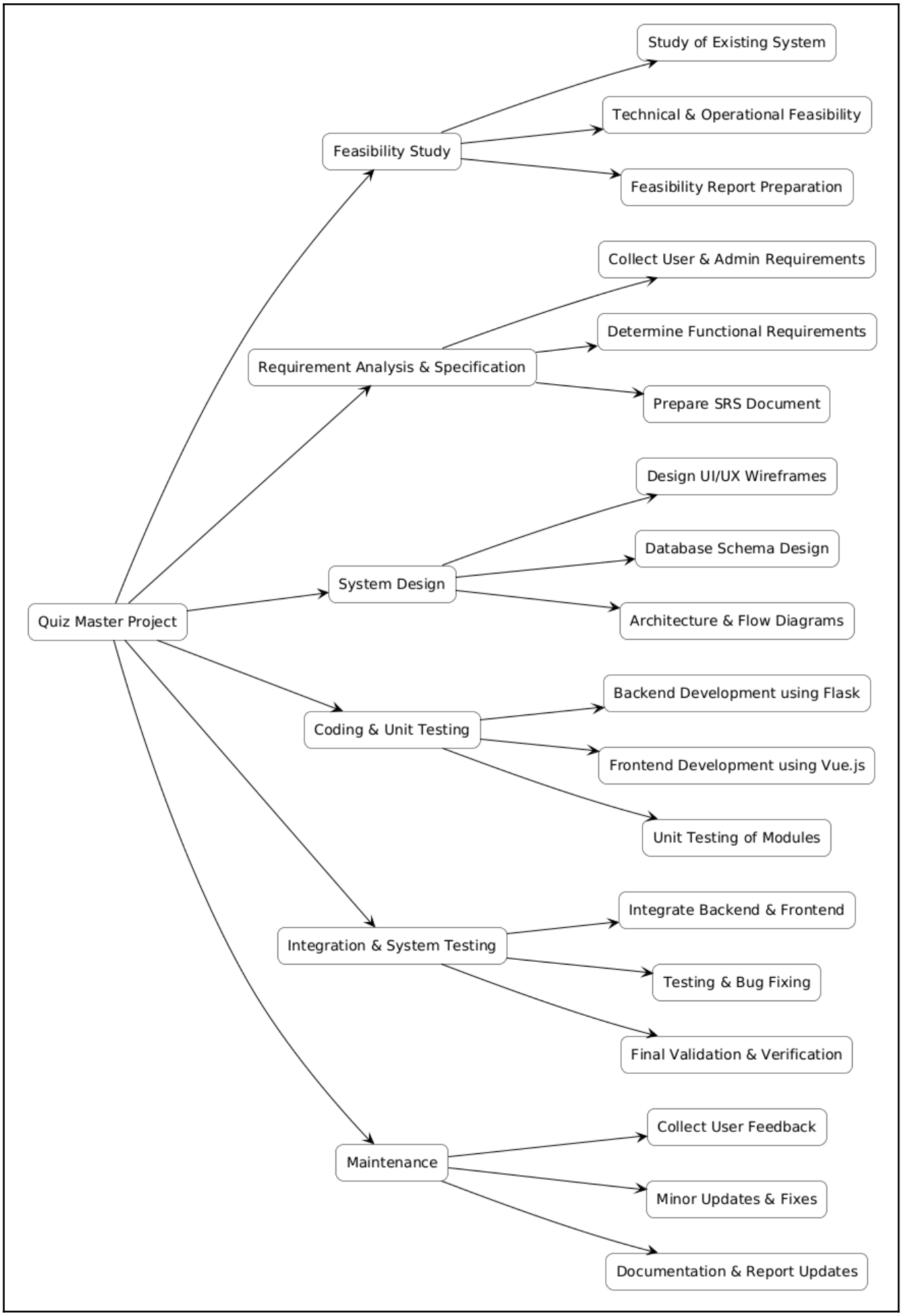
5. **Integration and System Testing**
   All modules were integrated and tested together to ensure proper interaction and performance.
   - Integrating backend APIs with frontend components.
   - Conducting system testing and resolving bugs.

- Validating the overall functionality and performance.

6. **Maintenance**
   This final phase ensured system stability and user satisfaction post-deployment.

   - Gathering user feedback.
   - Fixing minor issues and updating documentation.
   - Implementing enhancements for better performance.

WORK BREAKDOWN STRUCTURE:


Quiz Master Project
Feasibility Study
Study of Existing System
Technical & Operational Feasibility
Feasibility Report Preparation
Requirement Analysis & Specification
Collect User & Admin Requirements
Determine Functional Requirements
Prepare SRS Document
System Design
Design UI/UX Wireframes
Database Schema Design
Architecture & Flow Diagrams
Coding & Unit Testing
Backend Development using Flask
Frontend Development using Vue.js
Unit Testing of Modules
Integration & System Testing
Integrate Backend & Frontend
Testing & Bug Fixing
Final Validation & Verification
Maintenance
Collect User Feedback
Minor Updates & Fixes
Documentation & Report Updates

GANTT CHART

IMPLEMENTATION:

A Gantt Chart is a project management tool used to visually represent the timeline, duration, dependencies, and progress of various tasks within a project. It is an extension of the Work Breakdown Structure (WBS) that displays tasks along a time scale, helping to plan, coordinate, and track project activities effectively. Each task is shown as a horizontal bar, where the length of the bar represents the task duration, and its position reflects the start and end dates.

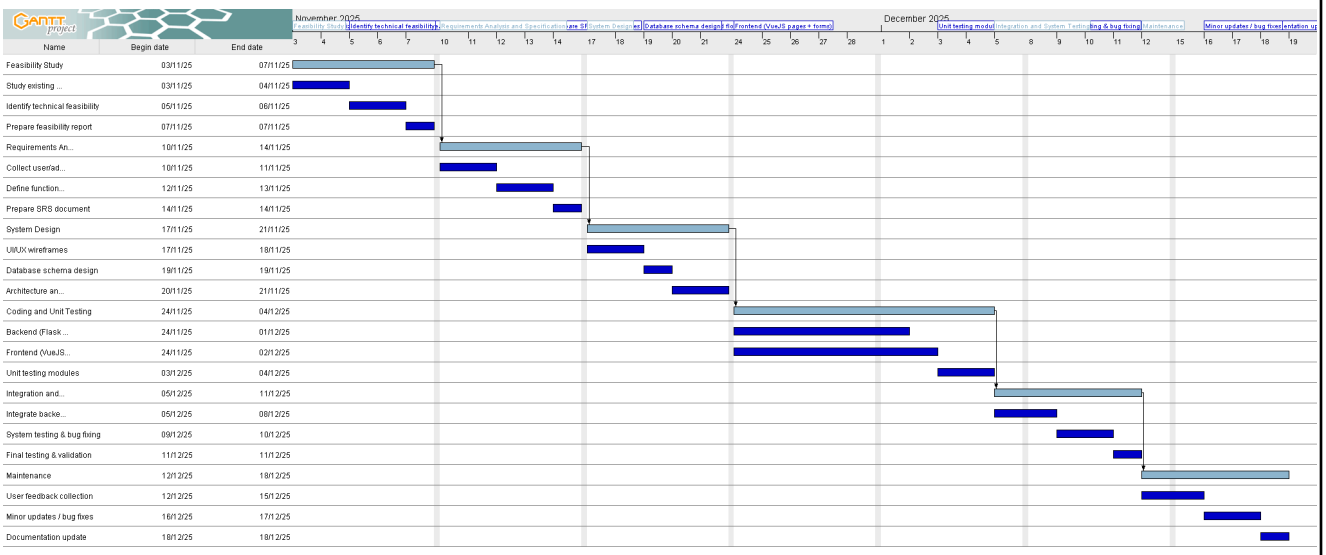
Unlike the WBS, which focuses on hierarchical task decomposition, the Gantt Chart emphasizes time management and progress tracking. It helps in identifying overlapping activities, critical dependencies, and the overall progress of the project, ensuring timely completion and efficient resource utilization.

STRUCTURE:

The Gantt Chart for the Quiz Master project was created using the GanttProject tool. It maps out all the phases of the Software Development Life Cycle (SDLC) — from Feasibility Study to Maintenance — along with their subtasks, timelines, resources, and interdependencies.

Each phase was assigned a maximum duration of five days, ensuring the project could be completed within the planned schedule. Two team members, Anushka and Riya, were allocated to different tasks based on their areas of responsibility.

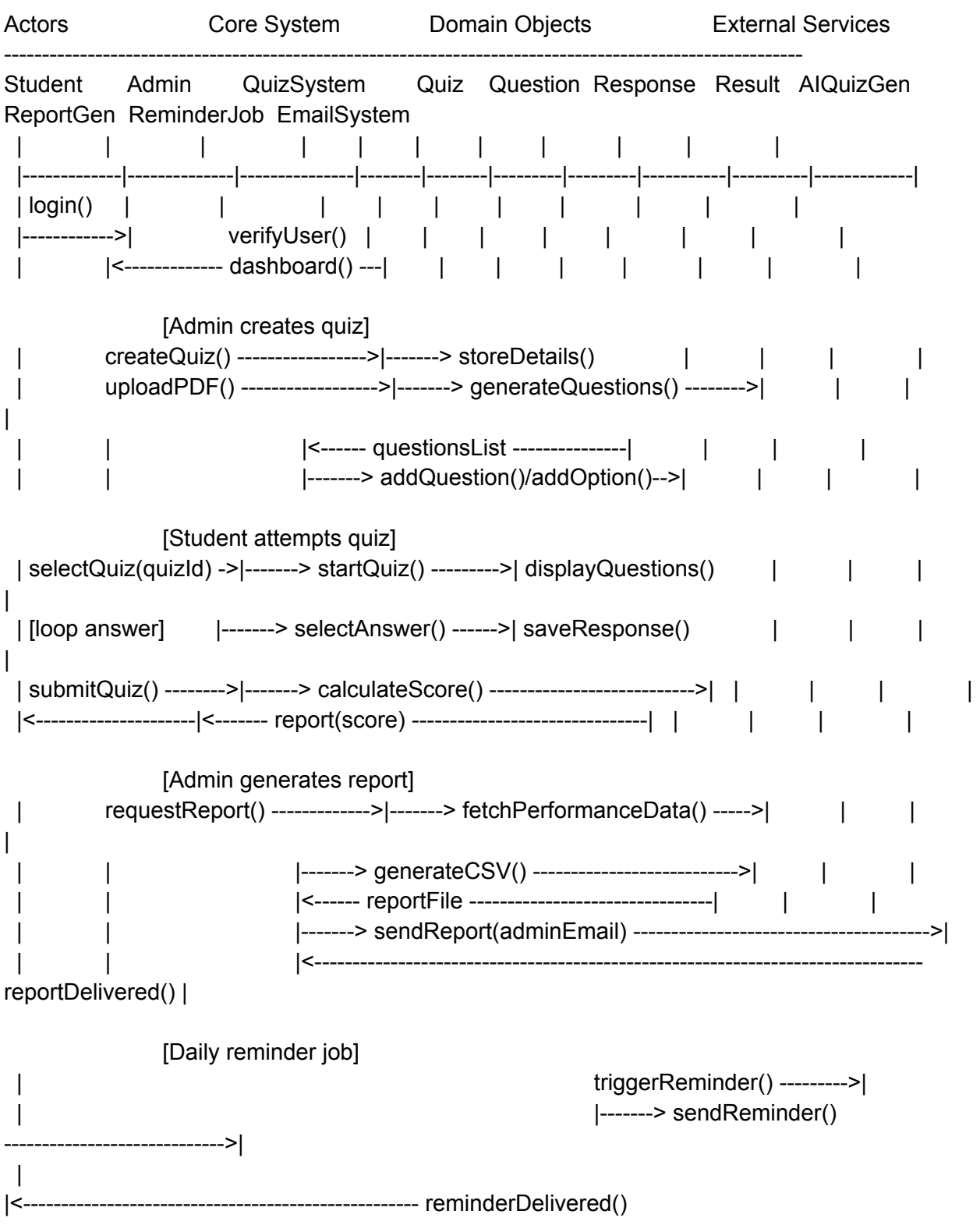
GANTT CHART:



CONCLUSION:	Through the Work Breakdown Structure (WBS) and Gantt Chart, we were able to plan, organize, and track the Quiz Master project effectively. The WBS helped us break down the work into clear SDLC phases and assign tasks systematically, while the Gantt Chart allowed us to schedule activities, set dependencies, and monitor progress visually. Together, these tools ensured that we managed our time efficiently and completed the project in a structured and coordinated manner.

🤭 ROUGH WORK

Tab 5


Actors
Core System
Domain Objects
External Services
Student
Admin
QuizSystem
Quiz
Question
Response
Result
AIQuizGen
ReportGen
ReminderJob
EmailSystem
login()
verifyUser()
dashboard()
[Admin creates quiz]
createQuiz()
storeDetails()
uploadPDF()
generateQuestions()
questionsList
addQuestion()/addOption()
[Student attempts quiz]
selectQuiz(quizId)
startQuiz()
displayQuestions()
[loop answer]
selectAnswer()
saveResponse()
submitQuiz()
calculateScore()
report(score)
[Admin generates report]
requestReport()
fetchPerformanceData()
generateCSV()
reportFile
sendReport(adminEmail)
reportDelivered()
[Daily reminder job]
triggerReminder()
sendReminder()
reminderDelivered()

//code 1

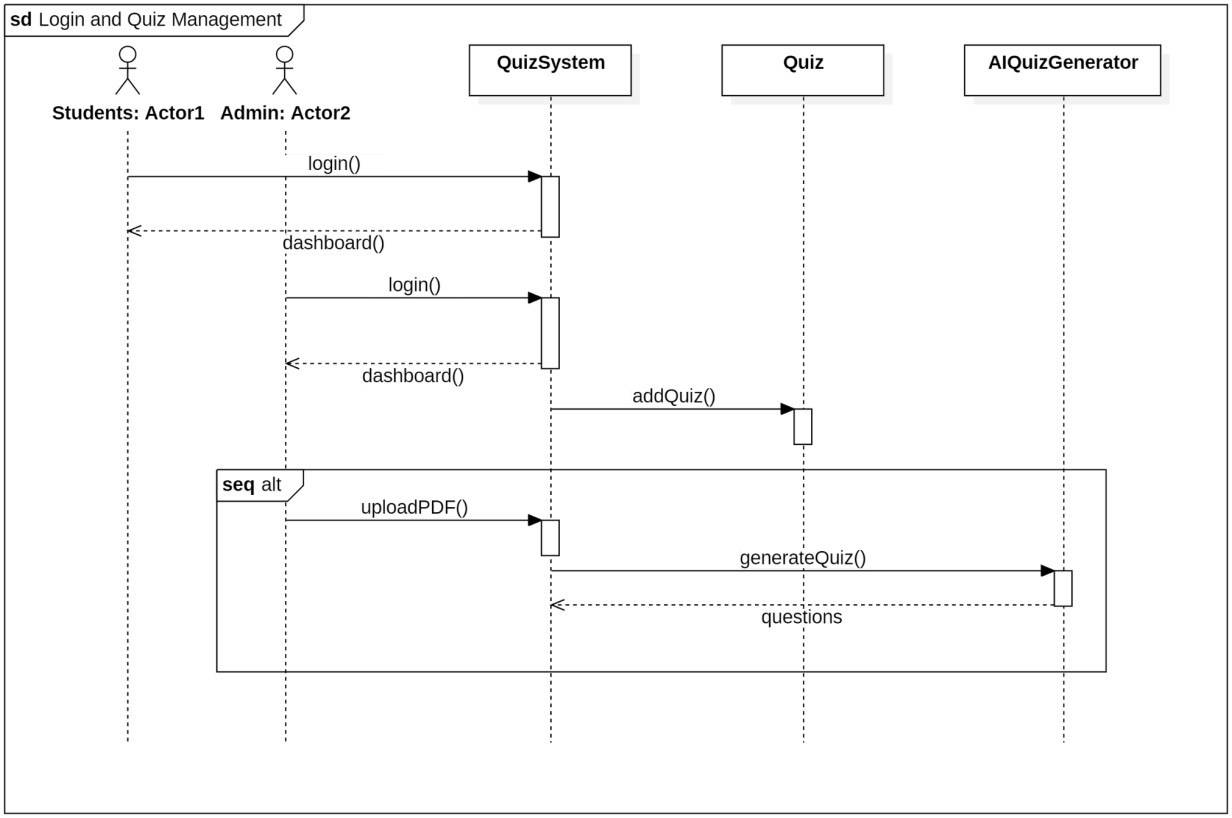
```
sequenceDiagram
    actor Student
    actor Admin
    participant QuizSystem
    participant Quiz
    participant Question
    participant Response
    participant Result
    participant AIQuizGenerator
    participant ReportGenerator
    participant ReminderJob
    participant EmailSystem

    %% UC1: Login/Register
    Student->>QuizSystem: login(email, password)
    QuizSystem->>QuizSystem: verifyUser()
    QuizSystem-->>Student: dashboard()
    Admin->>QuizSystem: login(admin credentials)
    QuizSystem->>QuizSystem: verifyUser()
    QuizSystem-->>Admin: dashboard()

    %% UC2: Create & Manage Quiz
    Admin->>QuizSystem: createQuiz()
    QuizSystem->>Quiz: storeDetails()
    Admin->>QuizSystem: uploadPDF()
    QuizSystem->>AIQuizGenerator: generateQuestions()
    AIQuizGenerator-->>QuizSystem: questionsList
    QuizSystem->>Question: addQuestion()
    Question->>QuizSystem: addOption()

    %% UC3: Attempt Quiz
    Student->>QuizSystem: selectQuiz(quizId)
    QuizSystem->>Quiz: startQuiz()
    Quiz->>Question: displayQuestions()
    loop for each Question
        Student->>QuizSystem: selectAnswer(questionId, optionId)
        QuizSystem->>Response: saveResponse()
    end
    Student->>QuizSystem: submitQuiz()
    QuizSystem->>Result: calculateScore()
    Result-->>QuizSystem: report(score)
    QuizSystem-->>Student: displayResults()
```

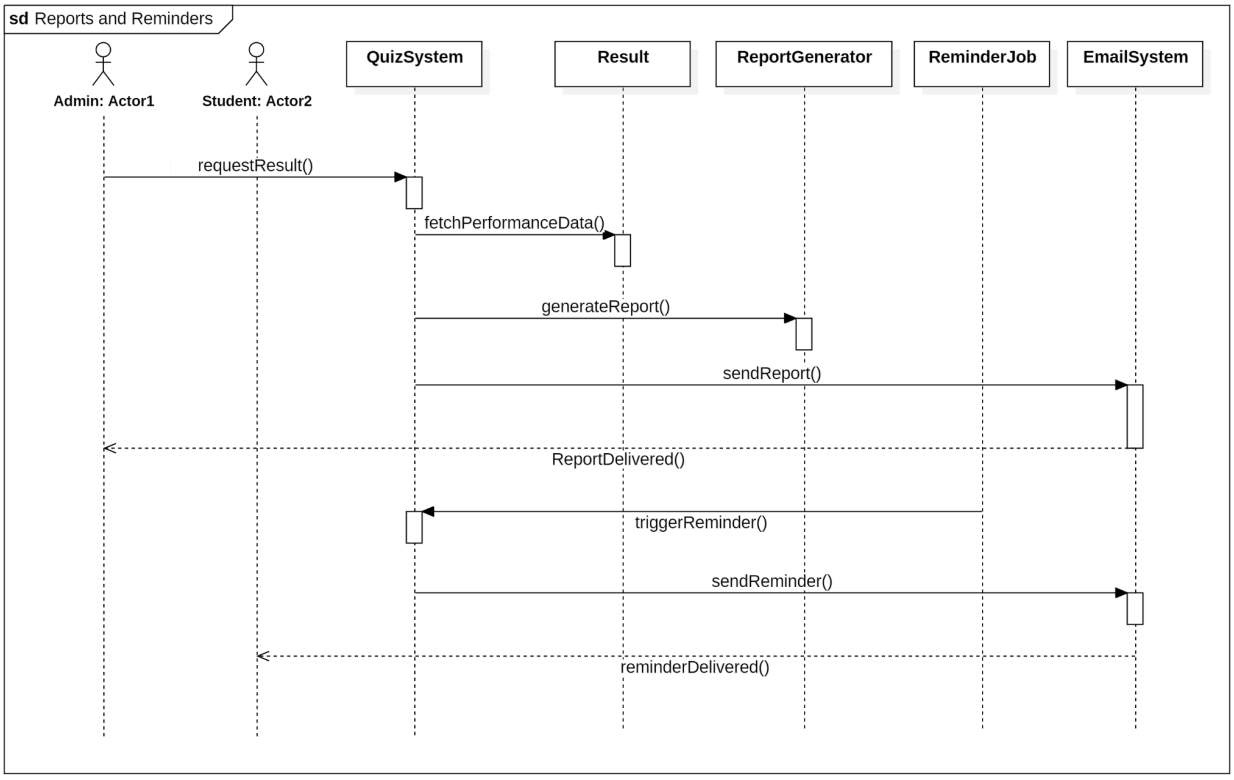
```
    %% UC7: Generate Reports
    Admin->>QuizSystem: requestReport()
    QuizSystem->>Result: fetchPerformanceData()
    QuizSystem->>ReportGenerator: generateCSV()
    ReportGenerator-->>QuizSystem: reportFile
    QuizSystem->>EmailSystem: sendReport(adminEmail)
    EmailSystem-->>Admin: reportDelivered()

    %% UC8: Daily Reminders
    ReminderJob->>QuizSystem: triggerReminder()
    QuizSystem->>EmailSystem: sendReminder()
    EmailSystem-->>Student: reminderDelivered()
```

//BEST CODE

```
sequenceDiagram
    actor Student
    actor Admin
    participant QuizSystem
    participant Quiz
    participant Question
    participant Response
    participant Result
    participant AIQuizGenerator
    participant ReportGenerator
    participant ReminderJob
    participant EmailSystem

    %% UC1: Login/Register
    Student->>QuizSystem: login()
    QuizSystem-->>Student: dashboard()
    Admin->>QuizSystem: login()
    QuizSystem-->>Admin: dashboard()

    %% UC2: Create & Manage Quiz
    Admin->>QuizSystem: createQuiz()
    QuizSystem->>Quiz: addQuiz()
    alt Auto Generation
        Admin->>QuizSystem: uploadPDF()
        QuizSystem->>AIQuizGenerator: generateQuestions()
        AIQuizGenerator-->>QuizSystem: questions
```

```
end

%% UC3: Attempt Quiz
Student->>QuizSystem: selectQuiz()
QuizSystem->>Quiz: startQuiz()
Quiz->>Question: showQuestions()
loop Each Question
    Student->>QuizSystem: answerQuestion()
    QuizSystem->>Response: saveResponse()
end
Student->>QuizSystem: submitQuiz()
QuizSystem->>Result: calculateScore()
QuizSystem-->>Student: showResults()

%% UC7: Generate Reports
Admin->>QuizSystem: requestReport()
QuizSystem->>Result: fetchPerformanceData()
QuizSystem->>ReportGenerator: generateReport()
QuizSystem->>EmailSystem: sendReport()
EmailSystem-->>Admin: reportDelivered()

%% UC8: Daily Reminders
ReminderJob->>QuizSystem: triggerReminder()
QuizSystem->>EmailSystem: sendReminder()
EmailSystem-->>Student: reminderDelivered()
```


sd Login and Quiz Management
Students: Actor1
Admin: Actor2
QuizSystem
Quiz
AIQuizGenerator
login()
dashboard()
login()
dashboard()
addQuiz()
seq alt
uploadPDF()
generateQuiz()
questions


sd Reports and Reminders
Admin: Actor1
Student: Actor2
QuizSystem
Result
ReportGenerator
ReminderJob
EmailSystem
requestResult()
fetchPerformanceData()
generateReport()
sendReport()
ReportDelivered()
triggerReminder()
sendReminder()
reminderDelivered()