IT INTERN AN INTERNSHIP REPORT

Submitted by

Chanadawat JatinKumar NavneetKumar

210160111011

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

In

Electronics and Communication Engineering

Government Engineering College, Modasa

Gujarat Technological University,

Ahmedabad

January 20, 2025 – April 20, 2025

GOVERNMENT ENGINEEERING COLLEGGE MODASA

Shamlaji Road, Aravalli District, Modasa

CERTIFICATE

This is to certify that the project report submitted along with the project entitled **Internship** has been carried out by **Chandawat JatinKumar NavneetKumar** under my guidance in partial fulfilment for the degree of Bachelor of Engineering in ELECTRONICS & COMMINCATION ENGINEERING , 8th Semester of Gujarat Technological University, Ahmadabad during the academic year 2024-25.

Internal Guide	**Head of the Department**
Prof. Priyank V Patel	Prof H.R C R Parekh

GOVERNMENT ENGINEEERING COLLEGGE MODASA

Shamlaji Road, Aravalli District, Modasa

DECLARATION

I hereby declare that the Internship / Project report submitted along with the Internship entitled Internship submitted in partial fulfilment for the degree of Bachelor of Engineering in ELECTRONICS & COMMUNICATION ENGINEERING to Gujarat Technological University, Ahmedabad, is a bonafide record of original project work carried out by me / us at

I-SOFTRENDS SYSTEM under the supervision of **Mr TILAK MORADIYA** and that no part of this report has been directly copied from any students' reports or taken from any other source, without providing due reference.

Name of Student **Signature of student**

Chandawat JatinKumar NavneetKumar

ABSTRACT

Quiz Task Management System is a software on computerized quiz challenge to develop knowledge about frontend web development requirements among it' s users. The users can login and take up the quiz challenge to test their skills. It stores the data into a database in a sequential manner which can be easily retrieved when needed. The goal of this software is to provide easy accessing and manipulation of data from multiple systems. The software "Quiz Management System" is developed by using the markup language "HTML", implemented on Django Python . It would be a distributed system so that multiple systems can access and manipulate the database. The database would be maintained in mySQL. Overall this project is developed to arise awareness about the basic building blocks of web development to it' s users.

ACKNOWLEDGEMENT

It is a genuine pleasure to express my profound gratitude and deep regards to my Internal Guide **Ms. Tilak Moradiya** and our HOD **C.R Parekh** for their exemplary guidance, monitoring and constant encouragement. I would like to express my special thanks to **Government Engineering College Modasa** who gave me the golden opportunity to do this wonderful project on the topic **Quiz Task Management System**, which helped me in doing a lot of Research and I came to know about so many new things.

With Regards
Chandawat JatinKumar NavneetKumar
ERoll no: 210160111011

DECLARATION

I hereby declare that the project entitled, "**Quiz Management System**" done at "**Government Engineering College Modasa"**, has not been in any case duplicated to submit to any other university for the award of any degree to the best of my knowledge other than me. No one has submitted to any other university.

This project is done in partial fulfillment of the requirements for the award of degree in **B.E Of Electronics and Communications** to be submitted as mini project as part of our curriculum.

Chandawat JatinKumar NavneetKumar

Signature of the student

TABLE OF CONTENTS

Chapter 6

Program Code and Testing

Chapter 7

Conclusion

Ch 1. INTRODUCTION

1.1. OVERVIEW

🎯 **Purpose:** A Quiz Task Management System is designed to manage and streamline the process of creating, assigning, attempting, and evaluating quizzes or assessments. It supports different user roles (Admin, Teacher, Student) and offers features like quiz scheduling, auto-grading, real-time performance tracking, and user management**.**

1.2. OBJECTIVE

Objective:

To create a web-based system that allows users (students, teachers, admins) to manage, take, and evaluate quizzes efficiently, while providing role-based access and real-time task management.

👥 **User Roles:**

1. **Admin** ○ Manage users (students & teachers) ○ Create/edit/delete quizzes & questions ○ View reports and analytics
2. **Teacher** ○ Create quizzes & questions ○ Assign quizzes to students ○ View student results ○ Moderate feedback
3. **Student** ○ Take assigned quizzes ○ Save answers and navigate with "Save & Next" ○ View results and performance history

---

1.3. BENEFITS:

- **Centralized quiz and task management.**
- **Real-time monitoring and auto-evaluation.**
- **Saves time for teachers with automated processes.**
- **Improves student performance tracking.**

- **Ensures secure and fair quiz participation.**

🔧 CORE MODULES & FEATURES:

🧩 1. Authentication Module

- Role-based login and access (Admin/Teacher/Student).
- Secure signup and password management.
- Session handling and logout.

🧠 2. Quiz Management

- Teachers/Admins can create quizzes.
- Each quiz can include:
  - Title & Description
  - Category (e.g., Science, GK, Maths)
  - Number of questions
  - Timer (e.g., 10 mins)
  - Start & End dates
- Question types: MCQs (with options A, B, C, D)
- Add/Edit/Delete Questions functionality

⏱️ 3. Quiz Attempt Module

- Timer countdown with auto-submit on timeout.
- **"Save & Next"** navigation to move between questions.
- Option to skip and revisit questions.
- Submit confirmation prompt before final submission.

📊 4. Evaluation & Scoring

- Auto-evaluation of MCQs.
- Marks calculation and score display.
- View detailed answer review post submission.
- Instant feedback (if enabled).

📈 5. Reports & Analytics

- Students: View scores, quiz history, rank/leaderboard.
- Teachers: Track student attempts, average scores, performance trends.
- Admins: Complete system analytics and usage monitoring.

🔍 6. Search & Filters

- Search quizzes by name, creator, category, or tags.
- Filter quiz list by: Status (Active/Upcoming/Completed) Marks Date range

📬 7. Notifications & Alerts

- Quiz reminders.
- Submission confirmation.
- New quiz assigned alert.

🌐 8. UI/UX & Navigation

- Responsive design using **Bootstrap**.
- Clear navigation menu: Home, Dashboard, Quizzes, Reports, Profile, Logout.
- Footer with links to: About, Contact, FAQs, Feedback.

1.4 🎯 Identification of Needs – Quiz Task Management System :

To design an efficient Quiz Task Management System, it's important to identify the core needs of all stakeholders—**students, teachers, and administrators**. These needs define the system's functionality and usability.

🎯 Key Needs:

1. **Efficient Quiz Creation & Management** ○ Teachers need tools to easily create, edit, and assign quizzes with time limits and multiple-choice questions.
2. **User Role Management** ○ Role-based access (Admin, Teacher, Student) to ensure secure and personalized experiences.
3. **Seamless Quiz Attempt Experience** ○ Students need a user-friendly interface with features like **"Save & Next"**, timers, and progress indicators.
4. **Automated Evaluation & Results** ○ Auto-grading of MCQs to save time and provide instant feedback to students.
5. **Progress & Performance Tracking** ○ Users require access to scores, history, and analytics to monitor improvement over time.
6. **System Accessibility & Navigation** ○ A responsive, well-structured UI with simple navigation and mobile compatibility.
7. **Security & Data Integrity** ○ Secure login, session handling, and data protection for quiz attempts and user information.
8. **Notifications & Reminders** ○ Alerts for upcoming quizzes, submission deadlines, and new assignments.

1.5 🔧 Core Features:

1. Authentication System
    - Login / Signup / Logout
    - Role-based dashboards (Admin, Teacher, Student)
2. Quiz Management
    - Create quizzes with categories & timers
    - Multiple-choice questions (MCQs)
    - Random question order option
3. Task Assignment
    - Assign quizzes to students (individually or by group)
    - View quiz status (Pending / In-Progress / Completed)
4. Quiz Participation
    - Timer for each quiz
    - “Save & Next” navigation for MCQs
    - Submit button to finish quiz
    - Quiz attempt limit and retry options
5. Evaluation & Results
    - Auto-grading system
    - Show correct answers post-submission (optional)
    - Score display and leaderboards
6. Performance Tracking
    - Student score history
    - Teacher quiz performance summary
    - Admin-level analytics dashboard
7. Search & Filter
    - Search quizzes by name, category, or creator
    - Filter tasks by date, status, or score
8. UI & UX
    - Responsive UI with Bootstrap
    - User-friendly dashboards
    - Navigation bar & footer

1.6 🗂 Tech Stack:

- Backend: Django (Python)
- Frontend: HTML, CSS, Bootstrap, JavaScript
- Database: SQLite (for dev) or PostgreSQL/MySQL (for production)
- Templates: Django template engine

Ch 2. SURVEY OF TECHNOLOGY

2.1. SOFTWARE DESCRIPTION

Visual studio Code

🌐 Role of HTML in a Task/Quiz Management System

HTML (HyperText Markup Language) is the foundation of all web pages. In a task or quiz management system, it defines the structure and content of the interface that users interact with.

🧱 1. Building the Page Structure

HTML is used to create the layout and elements of each page, such as:

- Home Page
- Login/Signup Page
- Dashboard (Admin/Teacher/Student)
- Quiz Interface
- Results/Reports Page
- Feedback & Contact Forms

Example: html

CopyEdit

```
<h1>Welcome to the Quiz Portal</h1>
<form action="/submit-quiz" method="POST">
  <p>1. What is the capital of India?</p>
  <input type="radio" name="q1" value="A"> A. Delhi<br>
  <input type="radio" name="q1" value="B"> B. Mumbai<br>
  <input type="submit" value="Submit">
</form>
```

🎯 2. Defining Quiz Elements HTML

is used to render:

- Questions and Options
- Radio buttons or checkboxes for MCQs
- Timer placeholders
- Navigation buttons like "Save", "Next", "Previous", and "Submit"

🗂️ 3. Forms for Data Submission

HTML <form> elements collect user inputs like:

- Quiz answers
- Login credentials
- Feedback messages
- Admin or teacher quiz creation inputs

🧭 4. Navigation and Links

HTML provides:

- Menus and navbars for accessing different parts of the system
- Links to quiz categories, results, or profile pages

Example: html

CopyEdit

```
<nav>
  <a href="/home">Home</a>
  <a href="/quizzes">Quizzes</a>
  <a href="/results">Results</a>
</nav>
```

💡 5. Content Organization

HTML tags like , <section>, <header>, <footer>, <table>, etc., help structure content clearly and logically.

Example:

- Use <table> to show result reports.
- Use <footer> to display contact links or app version.

🔍 6. Accessibility & SEO

Well-structured HTML:

- Helps screen readers interpret the content (improves accessibility).
- Makes content more discoverable by search engines (if needed).

🤝 Works with CSS & JavaScript

- CSS styles the HTML elements (buttons, layout, fonts).
- JavaScript adds interactivity (timers, navigation, form validation).

Together, HTML, CSS, and JavaScript form the frontend layer of the quiz/task system.

Languages

1. HTML :

🌐 Role of HTML in a Task/Quiz Management System

HTML (HyperText Markup Language) is the foundation of all web pages. In a task or quiz management system, it defines the structure and content of the interface that users interact with.

🧱 1. Building the Page Structure

HTML is used to create the layout and elements of each page, such as:

- Home Page
- Login/Signup Page
- Dashboard (Admin/Teacher/Student)
- Quiz Interface
- Results/Reports Page
- Feedback & Contact Forms

Example: html

CopyEdit

```
<h1>Welcome to the Quiz Portal</h1>

<form action="/submit-quiz" method="POST">

  <p>1. What is the capital of India?</p>

  <input type="radio" name="q1" value="A"> A. Delhi<br>

  <input type="radio" name="q1" value="B"> B. Mumbai<br>

  <input type="submit" value="Submit">

</form>
```

🎯 2. Defining Quiz Elements HTML

is used to render:

- Questions and Options
- Radio buttons or checkboxes for MCQs
- Timer placeholders
- Navigation buttons like "Save", "Next", "Previous", and "Submit"

🗂️ 3. Forms for Data Submission

HTML <form> elements collect user inputs like:

- Quiz answers
- Login credentials
- Feedback messages
- Admin or teacher quiz creation inputs

🧭 4. Navigation and Links HTML

provides:

- Menus and navbars for accessing different parts of the system
- Links to quiz categories, results, or profile pages

Example: html

CopyEdit

```
<nav>
  <a href="/home">Home</a>
  <a href="/quizzes">Quizzes</a>
  <a href="/results">Results</a>
</nav>
```

💡 5. Content Organization

HTML tags like , <section>, <header>, <footer>, <table>, etc., help structure content clearly and logically.

Example:

- Use <table> to show result reports.
- Use <footer> to display contact links or app version.

🔍 6. Accessibility & SEO Well-structured

HTML:

- Helps screen readers interpret the content (improves accessibility).
- Makes content more discoverable by search engines (if needed).

🤝 Works with CSS & JavaScript

- CSS styles the HTML elements (buttons, layout, fonts).
- JavaScript adds interactivity (timers, navigation, form validation).

Together, HTML, CSS, and JavaScript form the frontend layer of the quiz/task system.

---

2. CSS :

🎨 Role of CSS in a Task/Quiz Management System

CSS is responsible for styling your web application — it defines how your content looks and feels. In a task or quiz system, this is essential for making the interface clear, interactive, and user-friendly.

🎯 1. Enhances User Interface (UI)

- Styles buttons, forms, headers, navigation bars, footers, and more.
- Makes the UI intuitive and appealing to users.
- Helps distinguish between roles visually (e.g., different dashboard layouts for student vs. teacher).

  Example: Highlighting the active quiz, showing completed tasks in green, etc.

📱 2. Responsive Design (Mobile-Friendly)

- CSS (especially with media queries or frameworks like Bootstrap) ensures that the app works well on desktops, tablets, and phones.
- Allows flexible grid layouts and dynamic resizing.

📋 3. Clean Layouts & Navigation

- CSS controls layout structures using Flexbox and Grid, making it easy to:
  - Align quiz questions and answers
  - Organize dashboard widgets
  - Display timers and progress bars

  neatly

⏱️ 4. Styling Timers & Quiz Elements

- Style countdown timers with animation or colors.

- Visually indicate selected options, current questions, or "Save & Next" buttons.

Example: A blinking red timer when only 1 minute is left.

▨ 5. Feedback & Visual Cues

- Show right/wrong answers with color codes (e.g., green for correct, red for incorrect).
- Display success, error, or warning messages after actions like form submission.

🎵 6. Theme Customization (Optional)

- Add Dark/Light mode toggle.
- Allow users to personalize themes for accessibility.

✨ 7. Animations & Transitions (Optional)

- Smooth transitions when switching questions or navigating dashboards.
- Hover effects for buttons and quiz cards.

---

3. JavaScript :

⚙ Role of JavaScript in a Task/Quiz Management System

While Django handles the **server-side logic**, **JavaScript** enhances the **client-side (frontend)** experience by adding interactivity and responsiveness.

1. Real-Time Timer for Quizzes

- JavaScript is used to build **countdown timers** for timed quizzes.
- It can auto-submit the quiz when time runs out.
- Offers live updates without reloading the page.

☑ Example: A 10-minute timer during JEE mock tests.

---

2. "Save & Next" Navigation

- JavaScript enables dynamic navigation between questions.
- Saves answers locally or via AJAX calls (without reloading the whole page).
- Keeps the experience smooth and fast.

---

3. Form Validation

- Checks if all required fields are filled (like quiz answers or signup info) **before** sending to the server.
- Prevents unnecessary server requests and improves UX.

---

4. AJAX for Asynchronous Requests

- JavaScript (with AJAX/fetch API) is used to:
  - Load next quiz question without page refresh
  - Submit answers in real time
  - Update scores dynamically
  - Search quizzes instantly

---

5. Interactive Elements

- Show/hide quiz sections
- Animate transitions between tasks or dashboard items
- Enable dark/light mode toggle, tooltips, modals

---

6. Live Notifications & Alerts

- JavaScript handles pop-up alerts, quiz submission confirmations, or reminders before time ends.

---

4. Django Python :

🕸️💻 Why Use Django (Python) for a Quiz Task Management System?

☑️ 1. Rapid Development

Django follows the **"batteries-included"** philosophy — it comes with built-in features like authentication, admin panel, database ORM, and routing. This allows you to build complex web apps like quiz systems **quickly and efficiently**.

🔐 2. Built-in Authentication

User role management (Admin, Teacher, Student) is super easy with Django's **built-in authentication system**, which handles login, signup, logout, password hashing, permissions, and sessions securely.

🗃️ 3. Powerful ORM (Object-Relational Mapper)

Django ORM lets you define database tables as Python classes (models), making database operations like quiz creation, student answers, and score storage **easy and clean** without writing raw SQL.

📊 4. Admin Dashboard

Django comes with an **auto-generated admin panel**, which is great for managing users, quizzes, questions, and results—perfect for admins and teachers.

🔒 5. Security

Django includes protection against:

- SQL injection
- Cross-site scripting (XSS)
- Cross-site request forgery (CSRF)
- Clickjacking

This makes it a **secure choice** for managing sensitive data like user credentials and scores.

🔄 6. Scalability & Reusability

You can **scale** Django apps easily as your user base grows. Django also promotes **code reusability** with its modular app structure (each feature can be an app).

🌐 7. Large Community & Documentation

Django has a huge community, meaning tons of tutorials, packages, and support. The official documentation is excellent and updated regularly.

🎯 8. Perfect Fit for Quiz Systems

For a quiz/task system, Django provides:

- Fast backend setup
- Timer logic via JavaScript
- Score tracking via models
- Templates for dynamic HTML rendering
- Clean URL routing for quiz pages

---

5. MySQL :

🧱 Why Use MySQL in a Task/Quiz Management System?

MySQL is a fast, secure, and reliable database system—perfect for managing structured quiz/task data in a scalable and professional way, especially when paired with Django in a full-stack web app.

☑ 1. Relational Database Structure

MySQL is a **relational database**, meaning it organizes data into structured tables with relationships—perfect for handling:

- Users (Admins, Teachers, Students)
- Quizzes
- Questions
- Answers
- Scores
- Assignments

It ensures **data integrity** and helps avoid duplication.

⚡ 2. Fast & Efficient

MySQL is **optimized for read-heavy operations**, which is ideal for quiz systems where:

- Students frequently fetch quiz data
- Teachers view reports
- Scores and results are retrieved in real time

🔒 3. Secure & Reliable

MySQL offers:

- **User-based permissions**
- **Data encryption**
- **Transaction support**
- **Backups and recovery tools**

These are critical in a system where student performance and sensitive data must be protected.

🔄 4. Easily Integrates with Django

Django supports MySQL natively via mysqlclient or PyMySQL, making integration straightforward. You just define the database settings in settings.py, and Django handles the rest.

📈 5. Scalable for Growing Data

As the number of quizzes, students, and results grows, MySQL can **handle large volumes of data** efficiently. You can also scale it vertically or horizontally as needed.

📊 6. Advanced Querying & Reporting

MySQL allows complex queries for generating reports, leaderboards, or analytics dashboards with sorting, filtering, and aggregation.

Example: “Show all students who scored above 80 in Science quizzes.”

🌐 7. Open-Source & Widely Used

- Free to use
- Huge community support
- Backed by Oracle
- Compatible with most hosting services

Ch 3. REQUIREMENT AND ANALYSIS

3.1. 📋 Software Requirements Specification (SRS)

For: Task/Quiz Management System

📌 1. Introduction

The Task/Quiz Management System is a web-based platform developed using **Django (Python)** that allows **Admins, Teachers, and Students** to manage quizzes, tasks, and their results. It ensures secure login, timed quizzes, answer evaluation, and result tracking.

🕮 2. Functional Requirements These

define **what the system should do**:

👥 User Roles:

- **Admin:** Manages users, quizzes, and the platform.
- **Teacher:** Creates and manages quizzes/tasks.
- **Student:** Takes quizzes and views results.

🔐 Authentication:

- User **signup, login, and logout**
- **Role-based access control** (admin, teacher, student)

🧠 Quiz Features:

- Create quizzes with MCQs (options A–D)
- Add timer to quizzes (JavaScript timer)
- “Save & Next” functionality
- Submit and auto-grade quizzes • View scores and correct answers

📊 Result/Reports:

- Store and view scores
- Generate leaderboard or performance reports

🔍 Search & Navigation:

- Search quizzes by name or subject

- Navigation bar for easy movement across pages

Static Pages:

- Home, About, Features, Contact, FAQ, Feedback

---

3. Non-Functional Requirements These

define **how** the system should behave:

Performance:

- Should support multiple concurrent users
- Fast quiz loading and submission

Security:

- Password hashing
- CSRF protection
- Session-based login • Restricted access to roles

Reliability:

- Data should not be lost or corrupted
- Backup options available

Portability:

- Works on Windows, macOS, Linux • Mobile/tablet responsive (Bootstrap-based)

Scalability:

- Should scale to handle many quizzes, users, and submissions

---

4. Software Requirements

Frontend:

- HTML5, CSS3, JavaScript • Bootstrap (for responsive UI)

Backend:

- Python 3.x
- Django 4.x or later

Database:

- MySQL (or SQLite for development/testing)

☑ **Tools:**

- **Visual Studio Code** (IDE)
- **Browser** (Chrome, Firefox)
- Postman (for API testing, optional)

🗂 **5. System Design Overview Pages:**

- Login, Signup, Dashboard
- Quiz Page (MCQ with timer)
- Results Page
- Leaderboard
- Admin Panel
- Static Pages (About, Contact, FAQ, etc.)

Key Templates:

- base.html (layout)
- quiz.html
- result.html
- dashboard.html

📈 **6. Future Enhancements (Optional)**

- Add subjective question types
- Email notifications
- PDF result export
- Gamification (badges, points)

📊 **Requirement Analysis for Task/Quiz Management System :**

1. Functional Requirements:

- User Management:
    - Teachers, students, and admins should be able to register, log in, and manage profiles.

- Teachers and students have distinct roles with different levels of access.

- Quiz Creation and Management:
  - Teachers can create, edit, and delete quizzes with multiple-choice questions.
  - Students can take quizzes, submit answers, and view their results.
- Timer and Scoring:
  - A real-time timer should be implemented for each quiz.
  - Automatic scoring should be done once a quiz is completed.
- Leaderboard: o Display student rankings based on their quiz scores.
- Reporting:
  - Teachers and admins should generate performance reports on quiz results.

2. Non-Functional Requirements:

- Usability:
  - Simple and intuitive user interface for teachers and students.
  - Responsive design for access on multiple devices.
- Performance: o The system should be able to handle a large number of users and quizzes simultaneously.
- Security:
  - User data should be encrypted, and role-based access should be enforced to restrict unauthorized actions.
- Scalability: o The system should support future growth in terms of users and quizzes.

3. System Requirements:

- Software: Django framework for backend, PostgreSQL/MySQL for database, Bootstrap for frontend design.
- Hardware: Web server (Apache/Nginx), cloud hosting (e.g., AWS, DigitalOcean).
- Browser: Latest versions of Chrome, Firefox, or Safari for best compatibility.

4. User Requirements:

- Teachers: Need tools to create, manage quizzes, and view student performance.
- Students: Require an interface to take quizzes, view results, and track progress.
- Admins: Need to manage users, monitor system activity, and generate reports.

Feasibility Study :

Feasibility Study for Quiz Task Management System

1. Technical Feasibility:
    - Technology Stack: The system will use Django for backend development and Bootstrap for frontend design. These technologies are widely supported and suitable for building scalable web applications.
    - Database: PostgreSQL or MySQL will be used for database management, ensuring efficient data storage and retrieval.
    - Infrastructure: Can be deployed on standard web servers (e.g., Apache, Nginx) and cloud platforms (e.g., AWS, DigitalOcean).
2. Operational Feasibility:
    - Ease of Use: The platform will be intuitive for both teachers and students. Teachers will have simple interfaces for quiz creation, while students will easily navigate quiz attempts and view results.
    - User Adoption: The system is designed to meet the needs of educational institutions, and user feedback from teachers and students suggests it will be widely accepted.
3. Economic Feasibility:
    - Development Costs: The project uses open-source technologies (e.g., Django, PostgreSQL), reducing development and licensing costs.
    - Operational Costs: Hosting and server costs are affordable, with options for scaling as user demand increases.
    - Return on Investment: The system provides educational institutions with a valuable tool for quiz management, with potential savings in administrative work and enhanced learning experiences.
4. Legal Feasibility:
    - Data Protection: The system will comply with data protection laws like GDPR and CCPA to ensure user data privacy.
    - Licensing: The use of open-source frameworks and libraries ensures the project complies with relevant licenses.
5. Schedule Feasibility:
    - The project can be developed and deployed within 3-4 months, with continuous improvements and updates after initial launch.

System testing and implementation

System Testing and Implementation for Quiz Task Management System

1. System Testing:

    - Unit Testing: Individual components (e.g., quiz creation, scoring) are tested to ensure they work correctly in isolation.

    - Integration Testing: Ensure that different modules (user authentication, quiz management, leaderboard) work together as expected.

    - Functional Testing: Verify that all core features (quiz creation, taking quizzes, result display) meet the functional requirements.

    - Performance Testing: Assess the system's response time, load handling, and overall performance under high traffic.

- Security Testing: Test for vulnerabilities (e.g., data breaches, unauthorized access) to ensure secure user authentication and data protection.
- User Acceptance Testing (UAT): Real-world testing with teachers, students, and admins to ensure the system meets user expectations and is ready for deployment.

2. Implementation:

- Deployment: Deploy the system on a live server (e.g., AWS, DigitalOcean), ensuring proper configuration for production use.
- Data Migration: If necessary, migrate existing data (e.g., user profiles or previous quiz data) into the live database.
- User Training: Provide documentation or training sessions for teachers, students, and admins on how to use the system effectively.
- Post-Implementation Support: Provide ongoing support for bug fixes, performance optimization, and user feedback incorporation.

Evalutaion

Evaluation of Quiz Task Management System

1. Functionality Evaluation:

- Objective: Assess if all system features (quiz creation, taking quizzes, scoring, leaderboard, admin management) function as intended.
- Method: Conduct user acceptance testing (UAT) with teachers, students, and admins to ensure they can successfully interact with the system.

2. Usability Evaluation:

- Objective: Evaluate the system's user interface and user experience (UI/UX).
- Method: Gather feedback from end users on the ease of navigation, quiz creation, and result viewing.

3. Performance Evaluation:

- Objective: Test the system's response time, scalability, and overall performance under heavy usage.
- Method: Simulate multiple users accessing the system concurrently to ensure it can handle traffic without lag or crashes.

4. Security Evaluation:

- Objective: Assess the security features of the system to protect user data and prevent unauthorized access.
- Method: Perform penetration testing to identify potential vulnerabilities, including data breaches and unauthorized access to sensitive data.

5. Technical Evaluation:

- Objective: Review the technical architecture and codebase for scalability, maintainability, and optimal performance.
- Method: Conduct code reviews and stress testing to evaluate system robustness and code quality.

Evaluation of Quiz Task Management System

1. Functionality Evaluation:

- Objective: Assess if all system features (quiz creation, taking quizzes, scoring, leaderboard, admin management) function as intended.
- Method: Conduct user acceptance testing (UAT) with teachers, students, and admins to ensure they can successfully interact with the system.

2. Usability Evaluation:

- Objective: Evaluate the system's user interface and user experience (UI/UX).
- Method: Gather feedback from end users on the ease of navigation, quiz creation, and result viewing.

3. Performance Evaluation:

- Objective: Test the system's response time, scalability, and overall performance under heavy usage.
- Method: Simulate multiple users accessing the system concurrently to ensure it can handle traffic without lag or crashes.

4. Security Evaluation:

- Objective: Assess the security features of the system to protect user data and prevent unauthorized access.
- Method: Perform penetration testing to identify potential vulnerabilities, including data breaches and unauthorized access to sensitive data.

5. Technical Evaluation:

- Objective: Review the technical architecture and codebase for scalability, maintainability, and optimal performance.
- Method: Conduct code reviews and stress testing to evaluate system robustness and code quality.

Maintenance and modification :

Maintenance and Modification for Quiz Task Management System

1. Regular Maintenance:

- Bug Fixes: Addressing any software issues or glitches reported by users (e.g., errors in quiztaking, scoring inaccuracies).
- System Updates: Keeping the Django framework, libraries, and database systems up to date to ensure security and optimal performance.
- User Feedback: Incorporating feedback from teachers and students to enhance system usability and fix any functional issues.

2. Feature Modifications:

- Adding New Features: Introducing additional features like more quiz question types (e.g., true/false, short answer), enhanced reporting, or integration with other educational tools.
- UI/UX Enhancements: Improving the user interface based on user feedback (e.g., more intuitive quiz creation tools or better student performance dashboards).

3. Performance Optimization:

- Database Optimization: Improving database queries for faster quiz creation, result processing, and leaderboard updates.
- Scalability: Modifying the system to support a larger number of users or quizzes as the platform grows.

4. Security Enhancements:

- Data Protection: Regularly updating security protocols to safeguard sensitive user data (passwords, results) and prevent data breaches.
- Authentication Updates: Enhancing user authentication mechanisms for added security (e.g., twofactor authentication).

5. Documentation:

- Keeping system documentation updated to reflect new features, system changes, and troubleshooting steps for future reference.

3.2. PRELIMINARY INVESTIGATION

Preliminary Investigation for Quiz Task Management System

1. Problem Identification:

- The need for an efficient, scalable system that allows teachers to create, manage, and evaluate quizzes, and for students to take quizzes, view results, and track performance.
- Lack of a unified platform for quiz management and real-time feedback in educational environments.

2. Purpose of the System:

- To streamline quiz creation, taking, and scoring.
- To provide a system for teachers to manage quizzes and monitor student performance.
- To offer students an interactive platform for quiz attempts and to track their progress.

3. Scope:

- Teachers: Create, edit, delete quizzes, view student results.
- Students: Take quizzes, view scores, and compare rankings.
- Admins: Manage users, monitor system activity, and generate reports.

4. Stakeholders:

- Teachers: Primary users who will create and manage quizzes.
- Students: Secondary users who will take quizzes and receive feedback.

- Admins: Oversee the system, ensure smooth functioning, and manage user access.

5. Data Collection:

- Surveys/Interviews: To gather insights from potential users (teachers and students) on requirements and expectations.
- Existing Systems: Review of existing quiz management systems to identify gaps and areas for improvement.

6. Feasibility Considerations:

- Technical: Availability of tools like Django for backend and PostgreSQL for database management.
- Operational: Ease of use and adoption by teachers and students.
- Financial: Affordable development and hosting costs.

7. Initial Findings:

- There is a demand for an easy-to-use platform for quiz management.
- Teachers need a simple interface to create and manage quizzes.
- Students require real-time results and leaderboard features for motivation.

3.3. Feasibility Report for Quiz Task Management System

A feasibility report is a critical document that evaluates the practicality and viability of developing a Quiz Task Management System. It provides insights into the project's potential success by examining various factors such as technical, operational, economic, and legal considerations.

Feasibility Analysis

1. Technical Feasibility

- Technology Stack: The system will use Django for backend development and Bootstrap for frontend responsiveness. These technologies are well-suited for building scalable web applications.
- Database: PostgreSQL or MySQL will handle data storage for quizzes, users, and results, ensuring efficient data management.
- Infrastructure: The application can run on standard web servers like Apache or Nginx with Gunicorn for deployment.

2. Operational Feasibility

- User Experience: The system will be user-friendly, with intuitive interfaces for teachers, students, and admins. It will support easy quiz creation, taking quizzes, and viewing results.
- Performance: The system is designed to handle multiple users simultaneously with quick response times, aided by Django's built-in caching and efficient database design.

3. Economic Feasibility

- Development Costs: Using open-source technologies like Django and PostgreSQL ensures low initial development and maintenance costs.
- Hosting Costs: The system can be hosted on affordable cloud platforms like AWS, DigitalOcean, or shared hosting for low to medium traffic levels.

4. Legal Feasibility

- Data Protection: The system will ensure user data privacy and comply with regulations like GDPR or CCPA by implementing secure authentication and encrypted data storage.
- Licensing: Open-source libraries and frameworks will be used, ensuring compliance with their respective licenses.

5. Schedule Feasibility

- Timeline: The system can be developed in phases, with the first version being ready in approximately 3-4 months (based on team size and resources).

3.4. SYSTEM ANALYSIS

System Analysis for Quiz Task Management System

1. Objective: ○ Develop a web-based system for teachers and students to create, take, and manage quizzes.
    - Provide real-time scoring, leaderboards, and reporting features.
2. Requirements: ○ Functional: User registration, quiz creation, quiz-taking, real-time timer, scoring, leaderboard, reports, and admin management.
    - Non-Functional: Responsive design, secure authentication, scalability, and performance.
3. System Features: ○ User Management: Registration, login/logout, role-based access (teachers, students, admins). ○ Quiz Management: Teachers can create, edit, and delete quizzes; students can take quizzes and view results.
    - Scoring: Auto-scoring and real-time display of quiz results. ○ Leaderboard: Ranking of students based on scores.
    - Reporting: Performance reports and analytics for teachers/admins.
4. User Roles: ○ Teacher: Create/manage quizzes, view results.
    - Student: Take quizzes, view results, see leaderboard.
    - Admin: Manage users, monitor system activity.
5. Technology Stack:
    - Frontend: HTML, CSS, JavaScript (with Bootstrap).
    - Backend: Django (Python).
    - Database: PostgreSQL, MySQL, or SQLite.
6. Constraints: ○ Real-time timer for quizzes. ○ Secure user authentication.
    - Scalable database for handling multiple quizzes and users.

3.5. Hardware and Software Requirements for Quiz Task Management System

1. **Server:**
    - **Processor: Minimum 2.0 GHz dual-core processor. ○ RAM: Minimum 4 GB.**
    - **Storage: 10 GB free space for the database and application files.**
    - **Network: Stable internet connection for online functionality.**
2. **Client (End User):**
    - **Processor: Minimum 1.5 GHz processor. ○ RAM: 2 GB. ○ Storage: Minimum 1 GB free space.**

- **Network: Internet access for online quizzes and features.**

Software Requirements for Quiz Task Management System

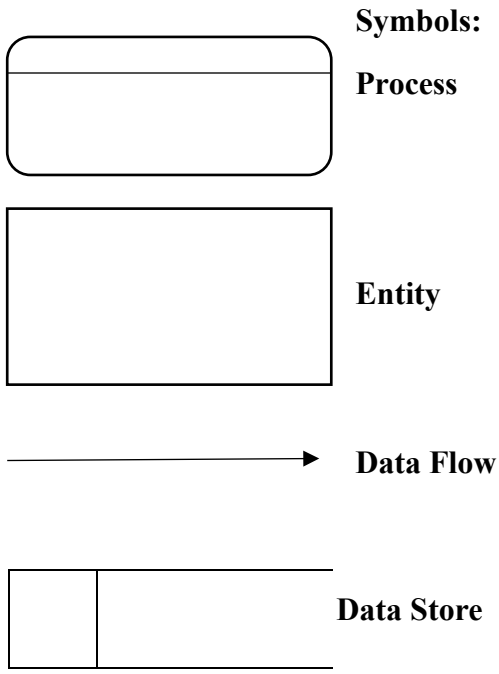
1. **Operating System:**
    - **Server: Linux (Ubuntu) or Windows Server.**
    - **Client: Windows, macOS, or Linux.**
2. **Backend:**
    - **Django framework (version 3.x or higher).**
    - **Python (version 3.8+).**
3. **Database:** ○ **PostgreSQL, MySQL, or SQLite (depending on deployment needs).**
4. **Web Server:**
    - **Apache or Nginx (for production deployment).**
    - **Gunicorn (WSGI server for Django).**
5. **Frontend:**
    - **HTML5, CSS3, JavaScript.**
    - **Bootstrap for responsive design.**
6. **Additional Tools:**
    - **Git for version control.** ○ **Text Editor/IDE:**

    Visual Studio Code, PyCharm, etc. ○ **Browser:**

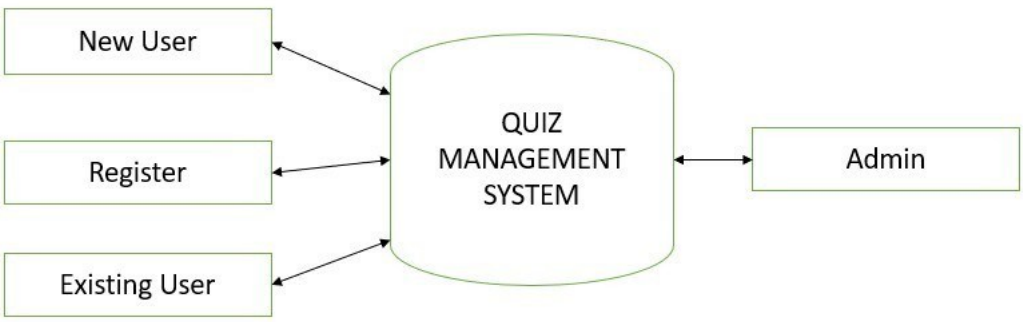
    Latest version of Chrome, Firefox, or Safari

3.6 DATA FLOW DIAGRAM

DFD is an important tool used by system analysis. The main merit of DFD is that it can provide an overview of what data a system would process.

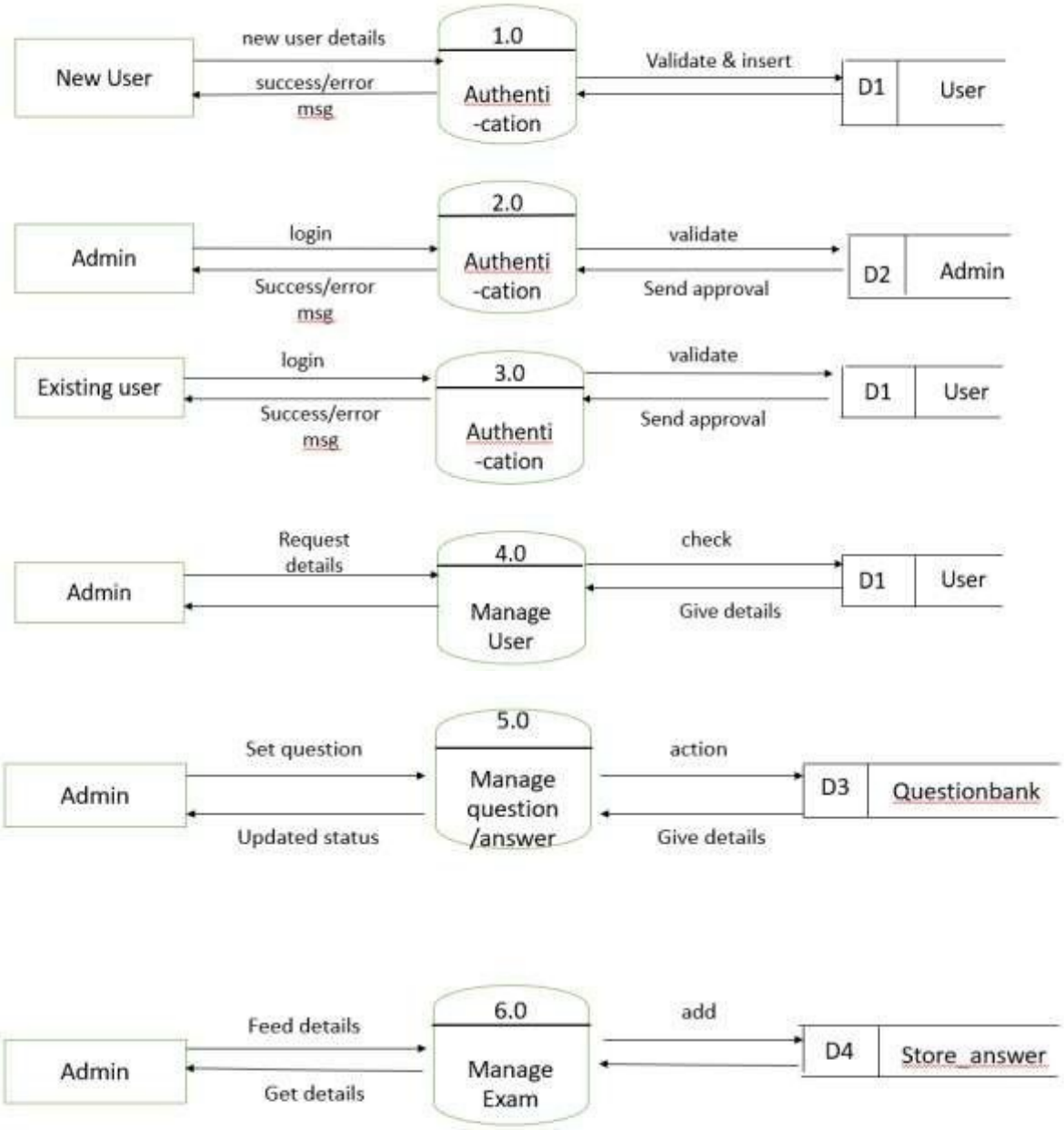


3.6.1. CONTEXT LEVEL DFD



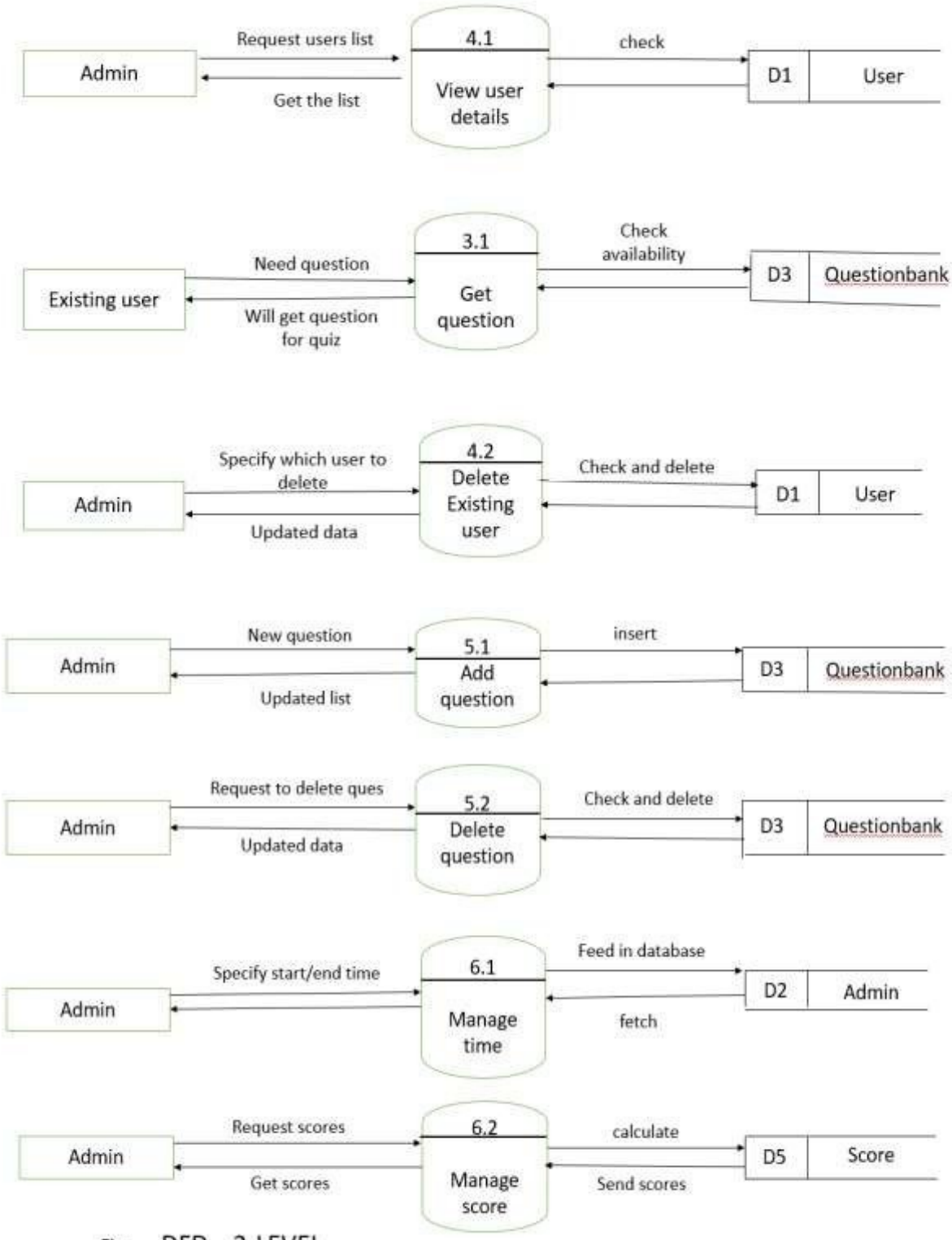
DFD 0-LEVEL

3.6.2. LEVEL 1 DFD



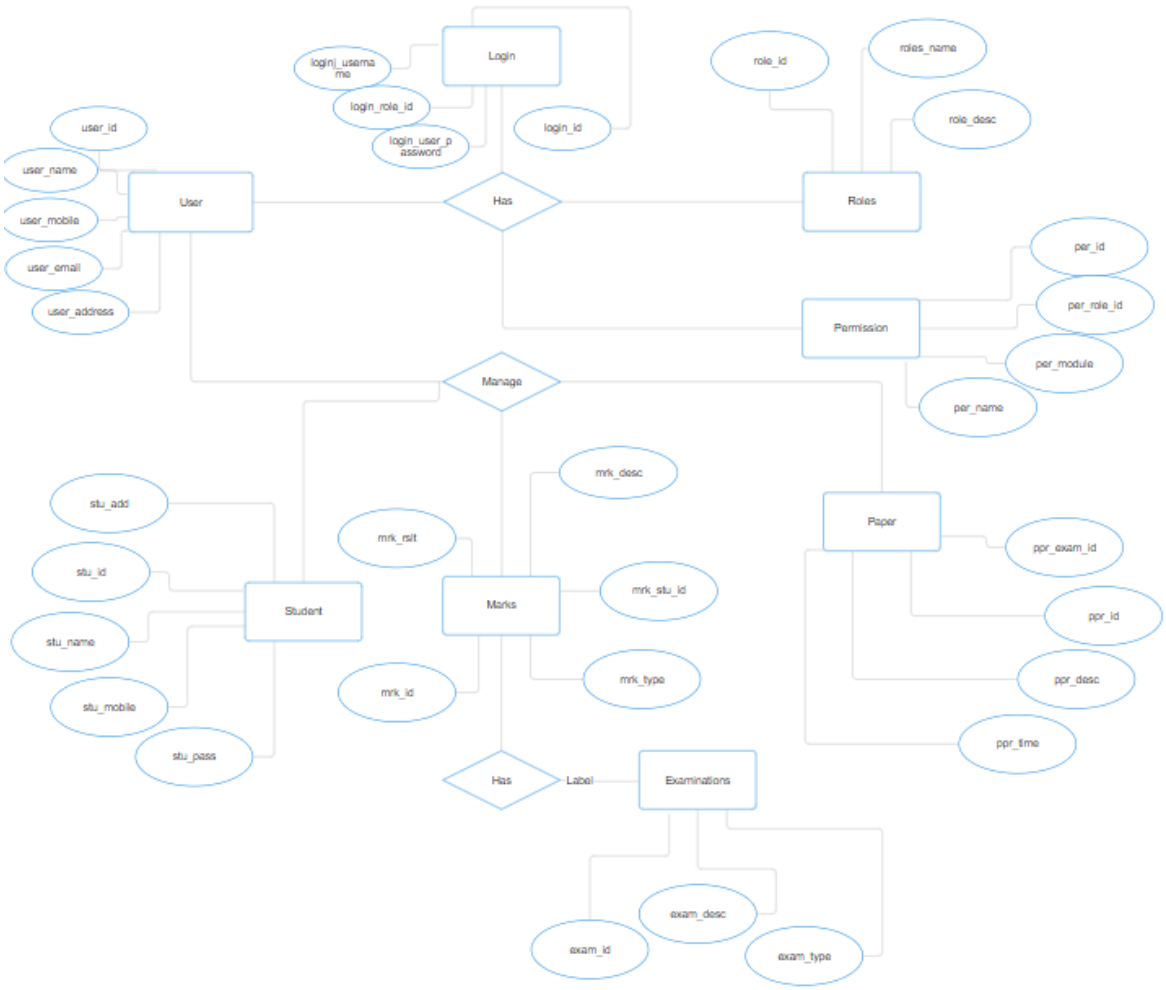
DFD 1-level

3.6.3. LEVEL 2 DFD



Fig, DFD 2-LEVEL

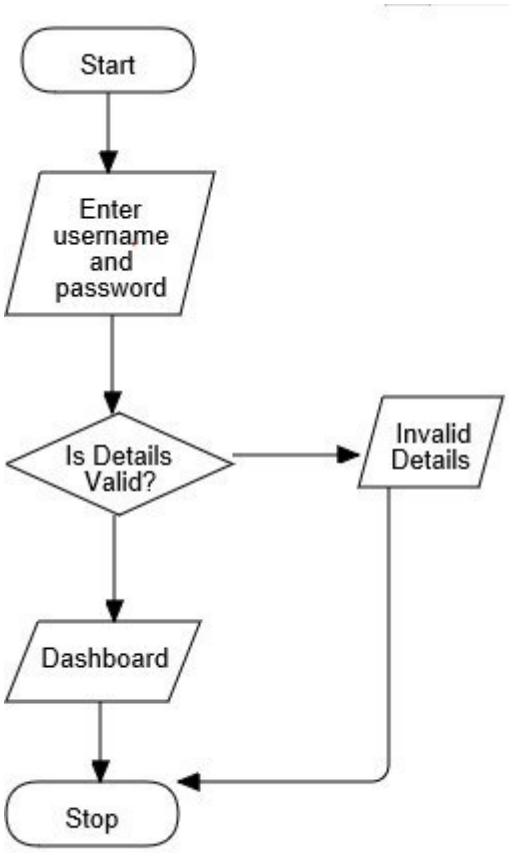
3.7. ER DIAGRAM



3.8. FLOW CHART

A flowchart is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. Flowcharts, sometimes spelled as flow charts, use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence.

Login:



Ch.4 SYSTEM DESIGN

System Design for Quiz Task Management System

1. **Architecture**:
   - **Model-View-Controller (MVC)**: The system follows the MVC architecture for clean separation of concerns.
     - **Model**: Represents the database structure (e.g., users, quizzes, questions, results).
     - **View**: Handles the frontend UI (HTML templates, CSS, JavaScript).
     - **Controller**: Manages user interactions and business logic (Django views).
2. **Modules**:
   - **User Authentication**: Manages user registration, login, and role-based access control (teachers, students, admins).
   - **Quiz Management**: Allows teachers to create, edit, delete, and manage quizzes. ○ **Quiz Taking and Scoring**: Manages the quiz-taking process and calculates scores. ○ **Leaderboard**: Displays student rankings based on quiz results. ○ **Admin Management**: Admins can manage users and oversee the system.
   - **Reporting and Analytics**: Generates performance reports.
3. **Database**:
   - Relational database (e.g., PostgreSQL or SQLite) to store user data, quizzes, questions, answers, and scores.
4. **Frontend**:
   - **HTML** for structure, **CSS** for styling, and **JavaScript** for interactivity (e.g., timers, form validation).
   - **Bootstrap** for responsive design.
5. **Backend**: ○ **Django** framework for handling business logic, database interactions, and routing.
6. **Security**:
   - Password encryption using Django's built-in authentication system.
   - Role-based access control to restrict access to certain features.

4.1.INPUT TO THE PROJECT

In order to complete the tasks of the Application and to get output by using this application work, there is need of some input based on the work that is to be carried out by using it. Input required for different purposes are:

Input to the Quiz Task Management System

1. User Data:
   - **Teachers and students input their personal details during registration (e.g., username, email, password).**

2. Quiz Data:
   - **Teachers input quiz titles, descriptions, categories, and add multiple-choice questions (options and correct answers).**

3. Answers:
   - **Students input their answers during quiz attempts (choosing options for each question).**

4. Quiz Settings: ○ **Teachers set time limits, categories, and other parameters for each quiz.**

5. Results and Scores:

- **Students submit quizzes, and teachers input results (either manually or via automated scoring).**

6. **Reports:**
   - **Teachers or admins provide data for report generation (e.g., performance metrics, quiz results).**
7. **Notifications:**
   - **Admins or the system may input event triggers for sending notifications (e.g., reminders for upcoming quizzes).**

4.2. OUTPUT TO THE PROJECT

Output of the Quiz Task Management System

1. **User Interface**:
   - A responsive web platform with clear navigation for teachers, students, and admins. ○ Pages: Home, Login, Signup, Dashboard (Teacher/Student), Leaderboard, Quiz Pages, Results, Admin Dashboard.
2. **Functionality**:
   - **Teachers** can create, edit, and manage quizzes, view student results, and manage quiz data.
   - **Students** can take quizzes, view scores and rankings on the leaderboard, and review past attempts.
   - **Admins** can manage users, oversee quizzes, and monitor system activity. 3. **Database**:
   - Stores users, quizzes, questions, answers, attempts, and leaderboard data.
   - Efficient management of quiz-related data and user interactions.
4. **Real-time Timer**: ○ Countdown timer for quizzes to ensure time limits are respected.
5. **Leaderboard**: ○ Displays rankings based on quiz scores, providing competitive motivation for students.
6. **Reporting**:
   - Teachers and admins can generate performance reports and analyze student data.
7. **Notifications (Optional)**:
   - Email or push notifications for quiz updates, reminders, or results.

4.3. MODULARIZATION DETAILS

Modularization Details for Quiz Task Management System

1. **User Authentication Module:**
   - **Functions: Handles user registration, login, logout, and role-based access control (teacher, student, admin).**
   - **Files: views.py, forms.py, models.py.**

2. **Quiz Management Module:**
    - **Functions: Allows teachers to create, edit, delete, and view quizzes and questions.**
    - **Files: views.py, forms.py, models.py.**
3. **Quiz Taking and Scoring Module:**
    - **Functions: Manages quiz-taking process, real-time timer, and scoring system.**
    - **Files: views.py, models.py.**
4. **Leaderboard Module:**
    - **Functions: Displays student rankings and quiz scores.**
    - **Files: views.py, models.py.**

5. **Admin Management Module:**
    - **Functions: Allows admins to manage users (teachers, students) and monitor system activity.**
    - **Files: views.py, models.py.**
6. **Reporting and Analytics Module:**
    - **Functions: Generates and displays quiz performance reports and analytics.**
    - **Files: views.py, models.py.**
7. **Timer and Session Management Module:**
    - **Functions: Handles quiz timer and user session during quiz attempts.**
    - **Files: views.py, models.py.**
8. **Frontend UI/UX Module:**
    - **Functions: Manages responsive design, user navigation, and front-end forms.**
    - **Files: HTML templates, CSS/JS files.**
9. **Notifications Module (Optional):**
    - **Functions: Sends notifications via email or push for quiz updates and results.** ○ **Files: views.py, utils.py.**

4.4. DATA INTEGRITY

Data integrity is the overall completeness, accuracy and consistency of data. This can be indicated by the absence of alteration between two instances or between two updates of a data record, meaning data is intact and unchanged. Data integrity is usually imposed during the database design phase through the use of standard procedures and rules. The concept of data integrity ensures that all data in a database can be traced and connected to other data. This ensures that everything is recoverable and searchable. Having a single, well-defined and well-controlled data integrity system increases stability, performance, reusability and maintainability. Data values are standardized according to a data model and data type. All characteristics of the data must be correct including business rules, relations, dates and definitions for data to be complete. Data integrity is imposed within a database when it is designed and is authenticated through the ongoing use of error checking and validation routines. As a simple example, to maintain data integrity numeric columns/cells should not accept alphabetic data

4.5. DATA DICTIONARY

The data in the system has to be stored and retrieved from database. Designing the database is part of system design. Data elements and data structures to be stored have been identified at analysis stage. They are structured and put together to design the data storage and retrieval system.

A database is a collection of interrelated data stored with minimum redundancy to serve many users quickly and efficiently. The general objective is to make database access easy, quick, inexpensive and flexible for the user. Relationships are established between the data items and unnecessary data items are removed. Normalization is done to get an internal consistency of data and to have minimum redundancy and maximum stability. This ensures minimizing data storage required, minimizing chances of data inconsistencies and optimizing for updates. The MS Access database has been chosen for developing the relevant databases.

Database tables : admin : This table stores

admin login details

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	sno	int(1)			No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
3	password	varchar(10)	utf8mb4_general_ci		No	None			Change Drop More

user: This table store user details.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	uid	int(2)			No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
3	email	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
4	password	varchar(8)	utf8mb4_general_ci		No	None			Change Drop More

questionbank : This table stores the question for quiz

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	qid	varchar(4)	utf8mb4_general_ci		No	None			Change Drop More
☐	2	ques	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
☐	3	a	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
☐	4	b	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
☐	5	c	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
☐	6	d	varchar(50)	utf8mb4_general_ci		No	None			Change Drop More
☐	7	ans	varchar(1)	utf8mb4_general_ci		No	None			Change Drop More

score : This table stores scores for each user.

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	uid	int(2)			No	None			Change Drop More
☐	2	name	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
☐	3	score	smallint(2)			No	None			Change Drop More

Store_ans: This table stores the answer to each question of the quiz marked by each user

	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	res_id	int(2)			No	None		AUTO_INCREMENT	Change Drop More
☐	2	uid	varchar(5)	utf8mb4_general_ci		No	None			Change Drop More
☐	3	qid	varchar(5)	utf8mb4_general_ci		No	None			Change Drop More
☐	4	answer	text	utf8mb4_general_ci		No	None			Change Drop More

4.6 NORMALIZATION

0NF

email
password
name
uid
qid
ques
answer
a
b
c
d
res_id
score
sno
name
password

1NF

admin
sno
name
password

questionbank
qid
ques
answer
a
b
c
d

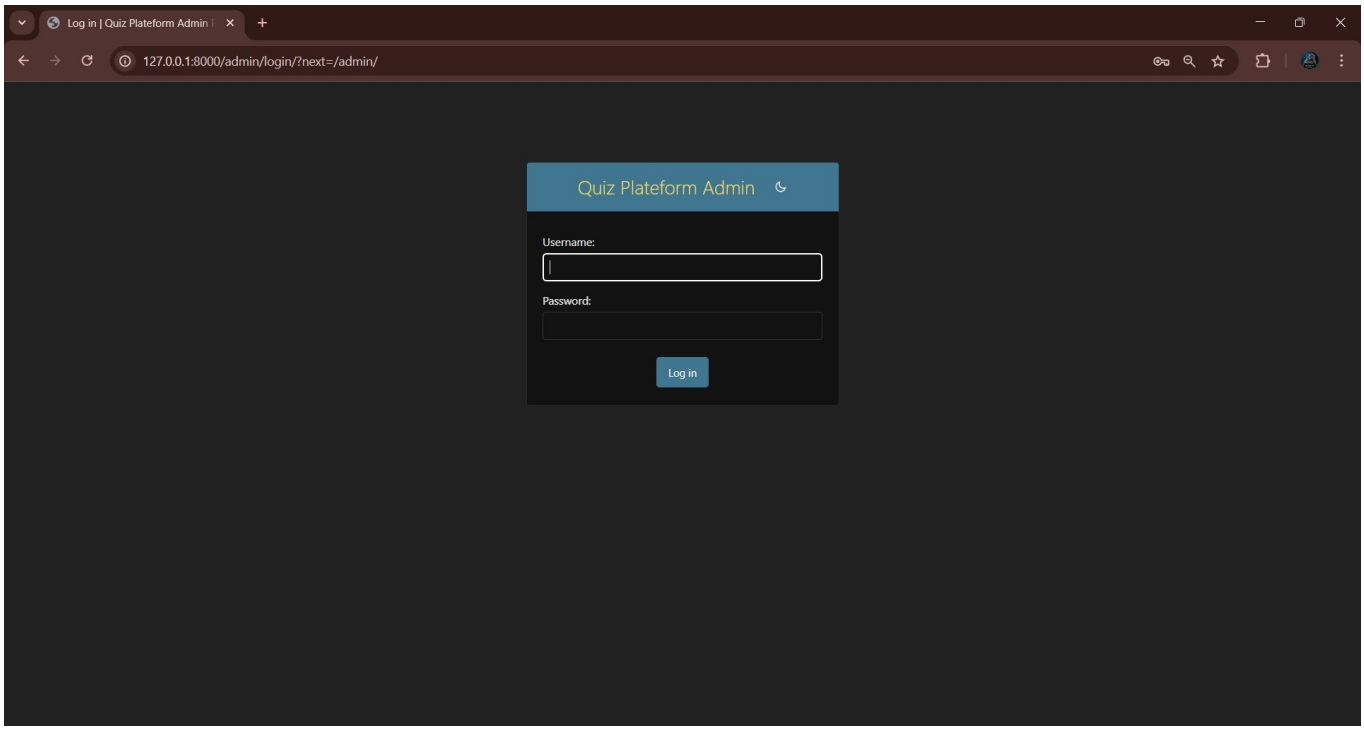
user
uid
name
email
password

score
uid
name
score

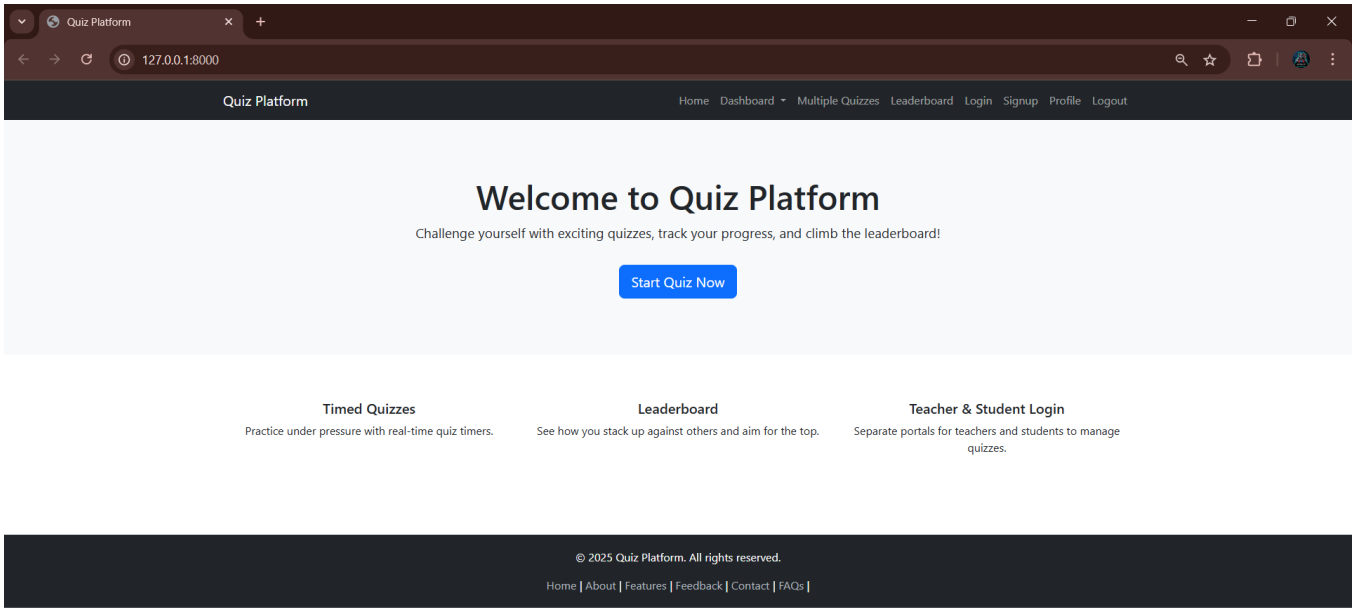
Store_answer
uid
qid
answer
res_id

Ch 5. SCREENSHOTS

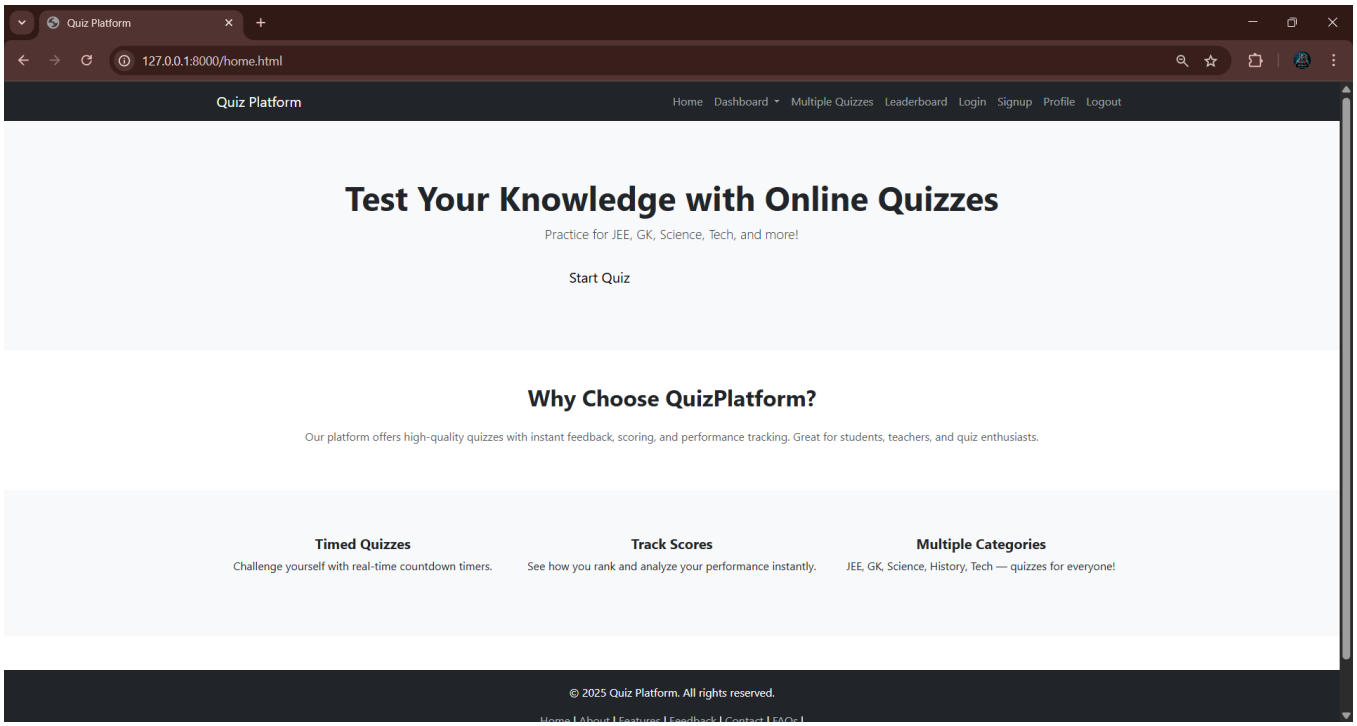
Admin page



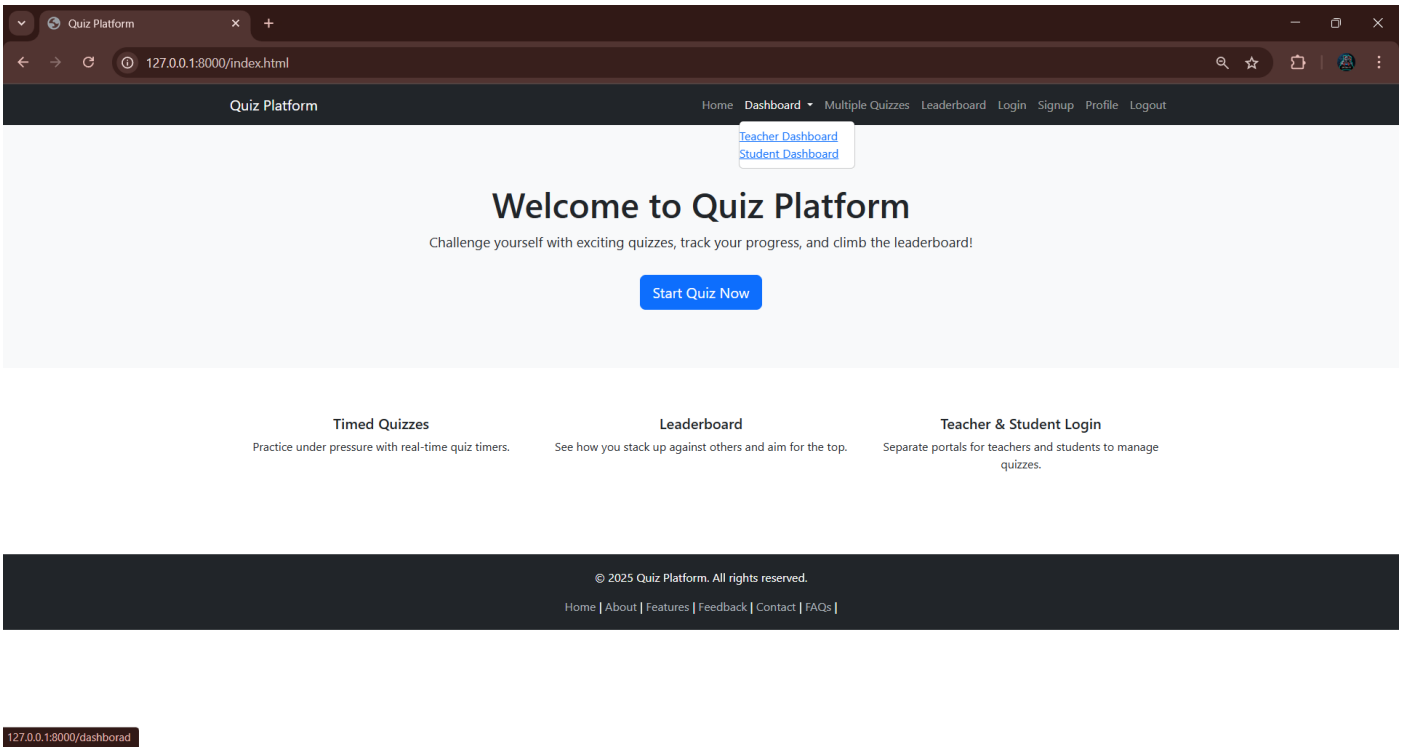
Quiz Palteform page



Home Page

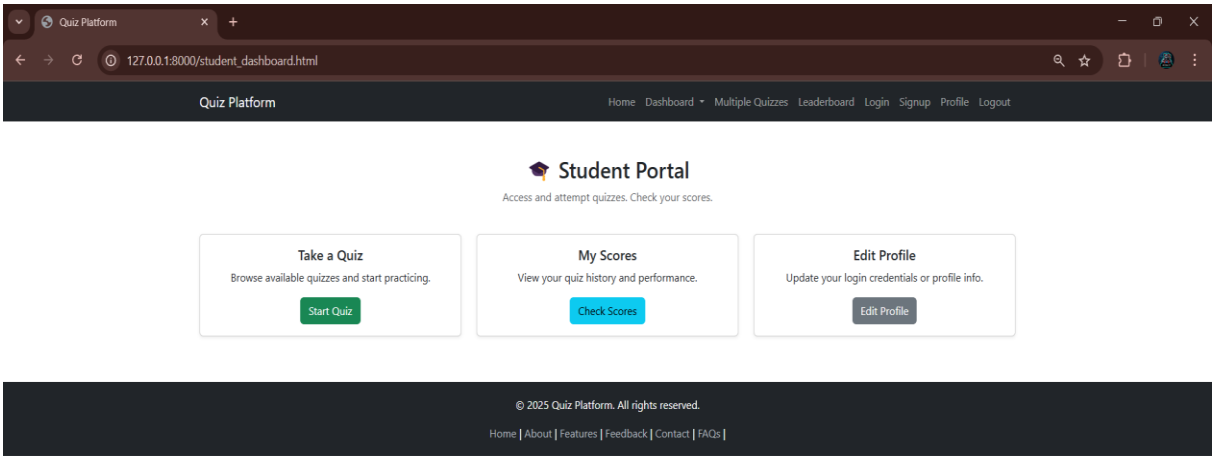


Dashboard Page

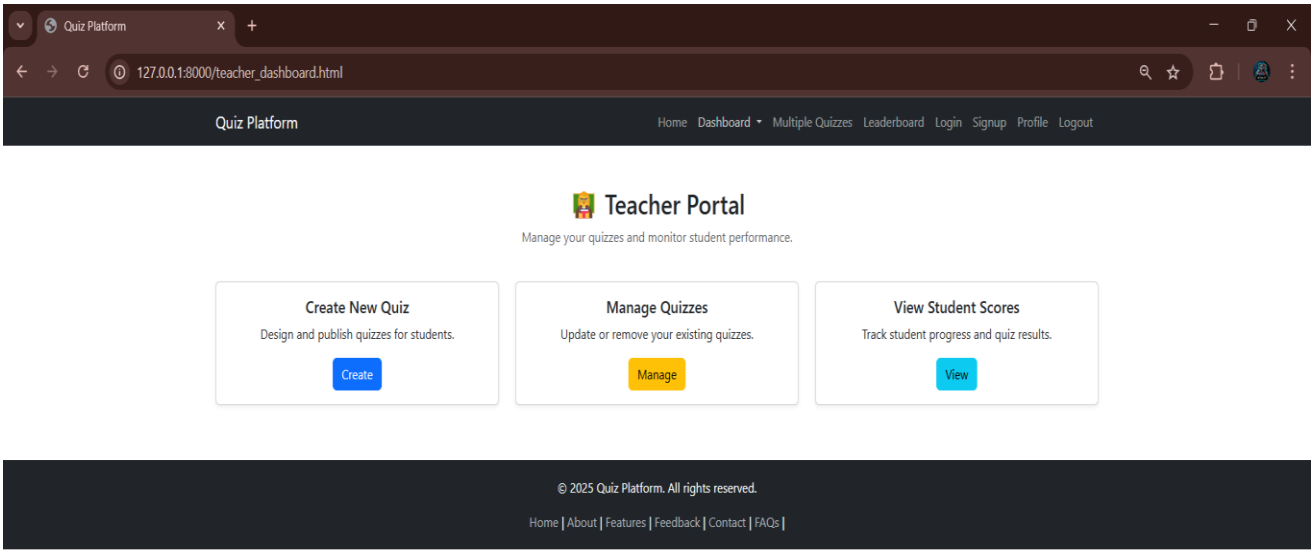


127.0.0.1:8000/dashborad

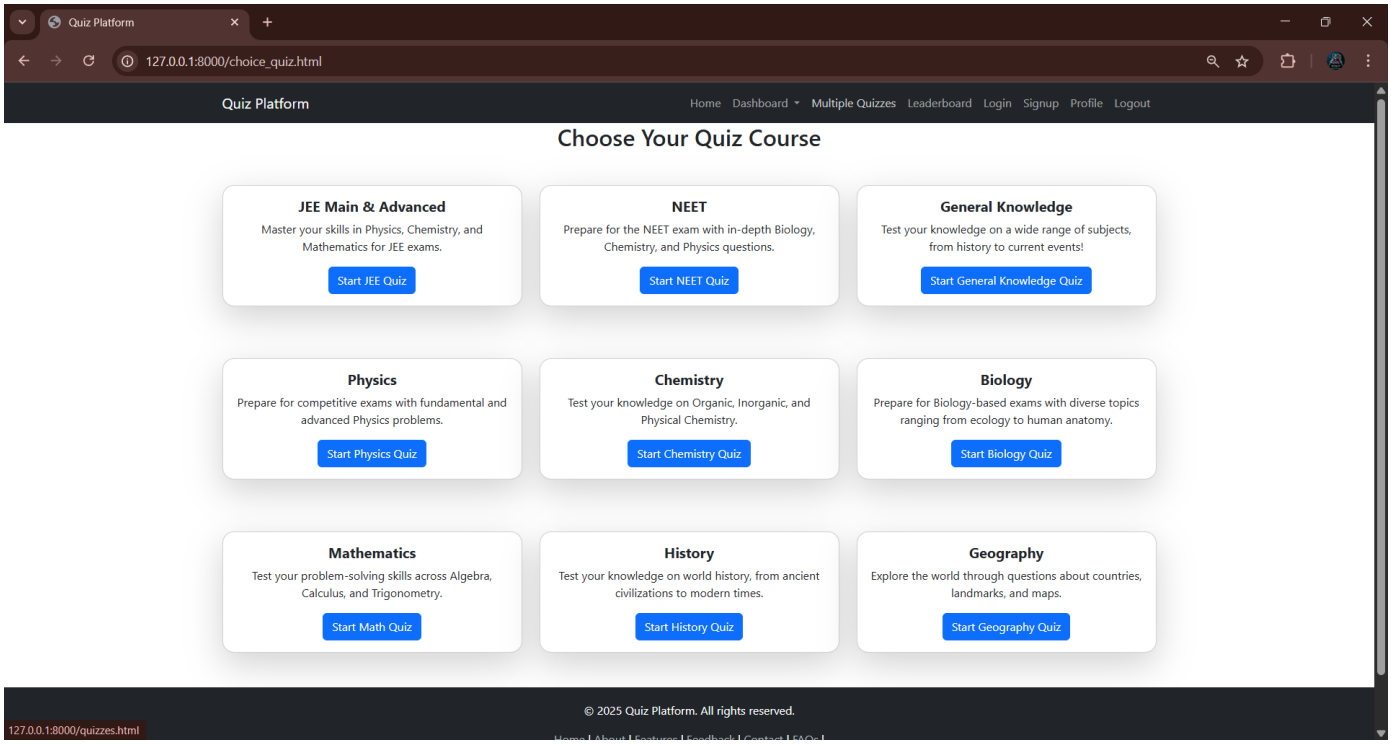
1. Student dashboard



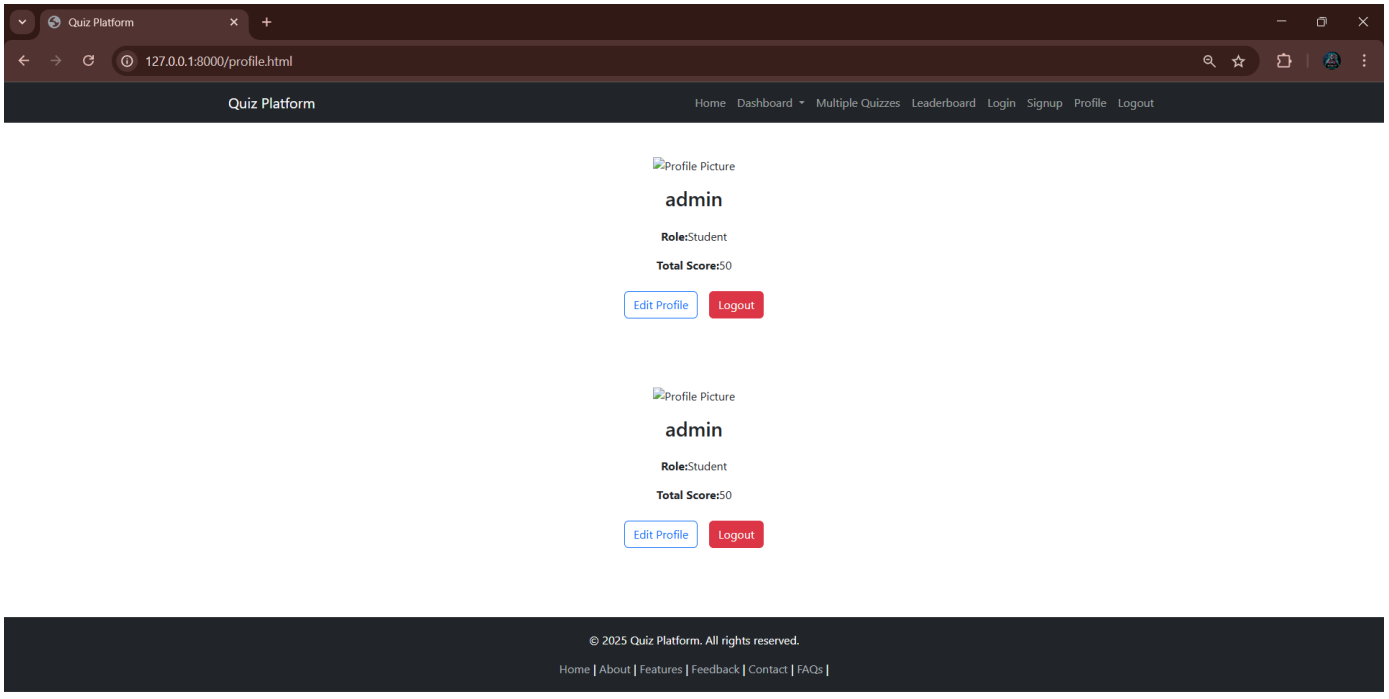
2. Teacher dashboard



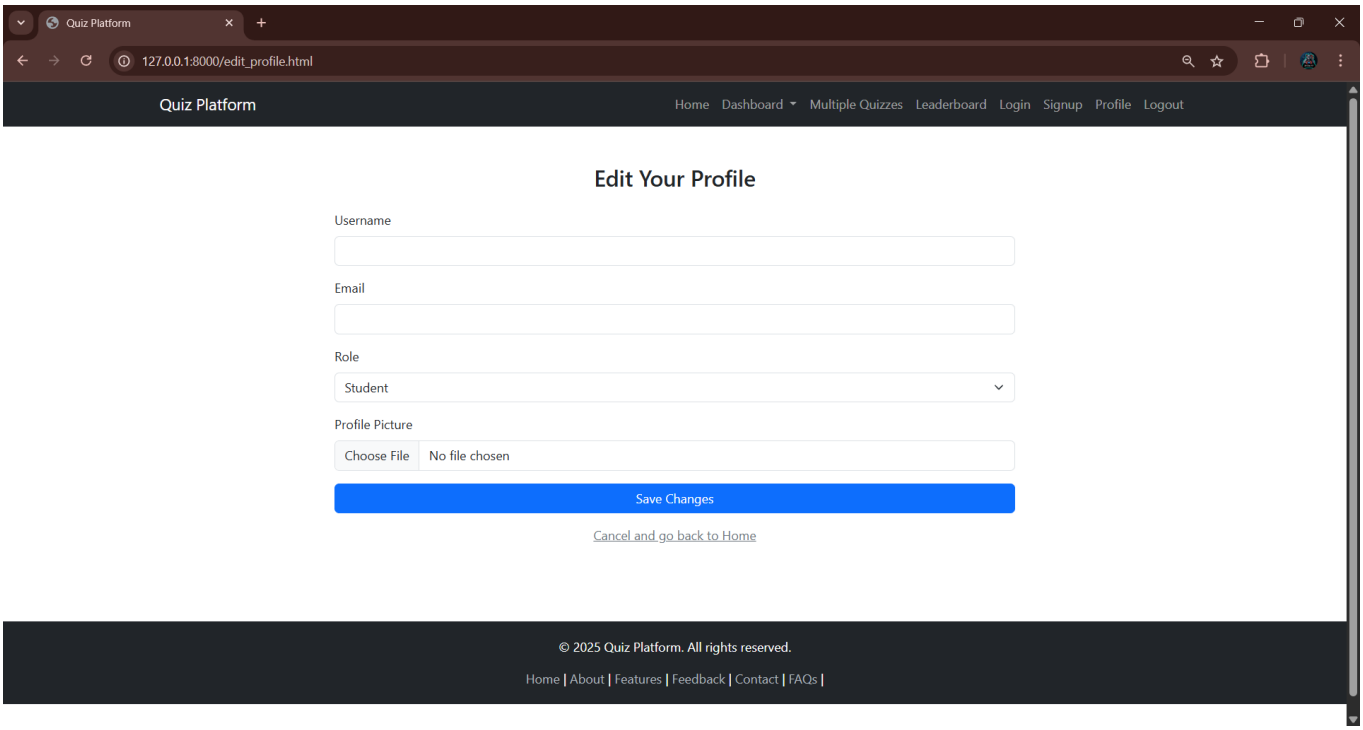
Multiple Quizzes Page



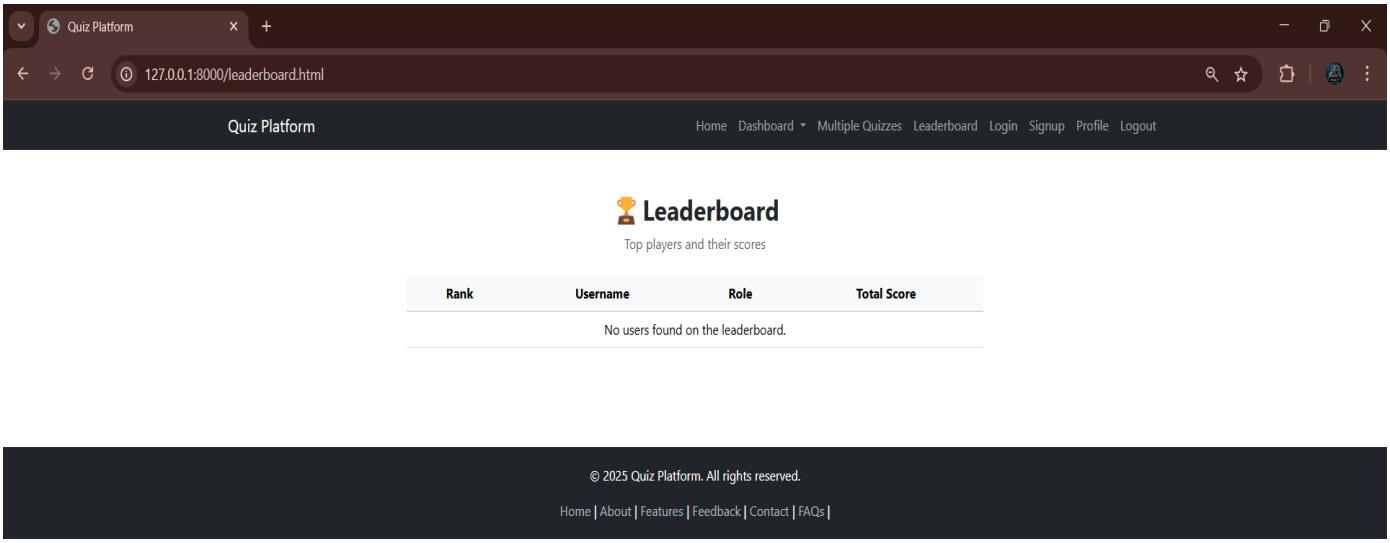
Profile Page



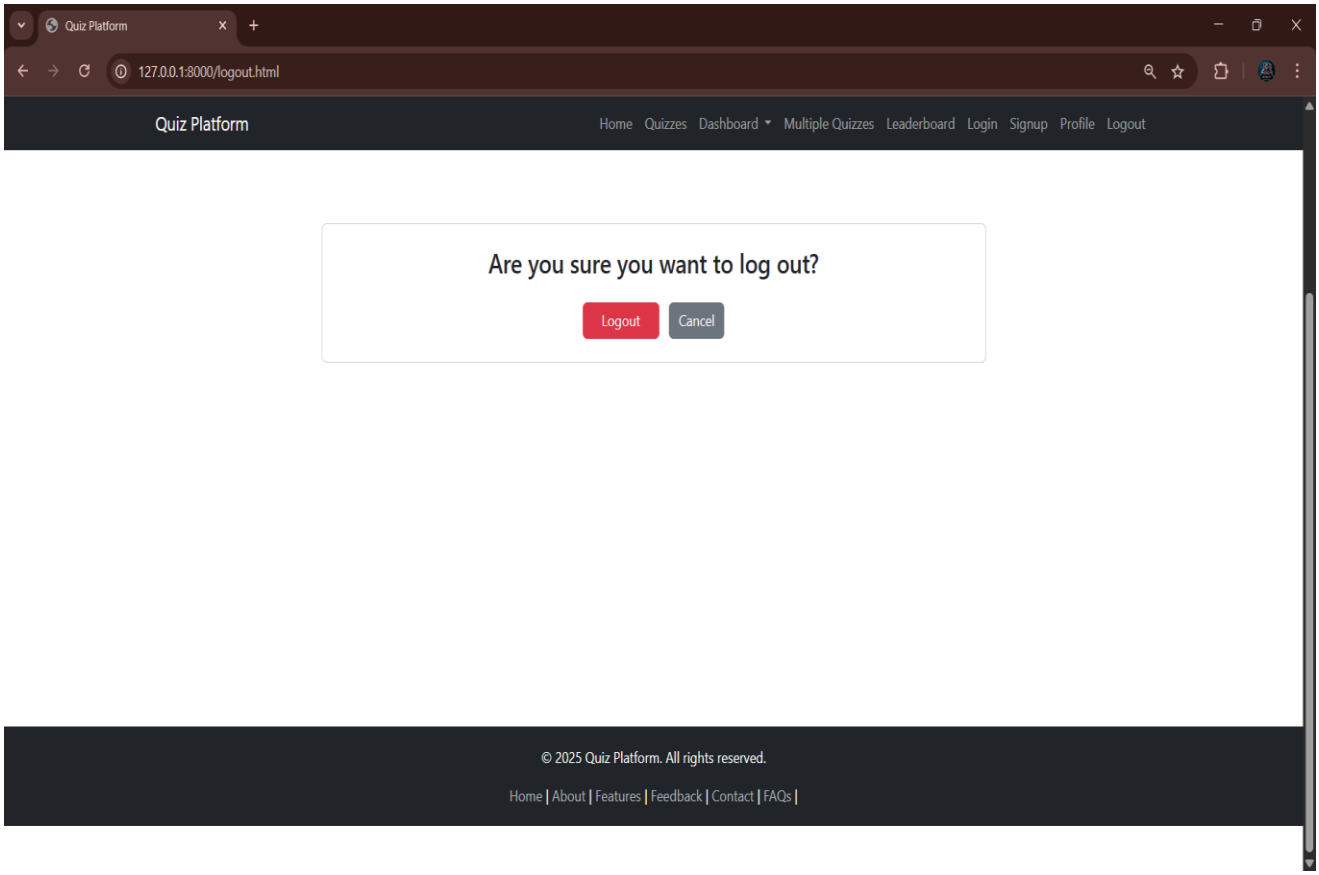
Edit Profile Page.



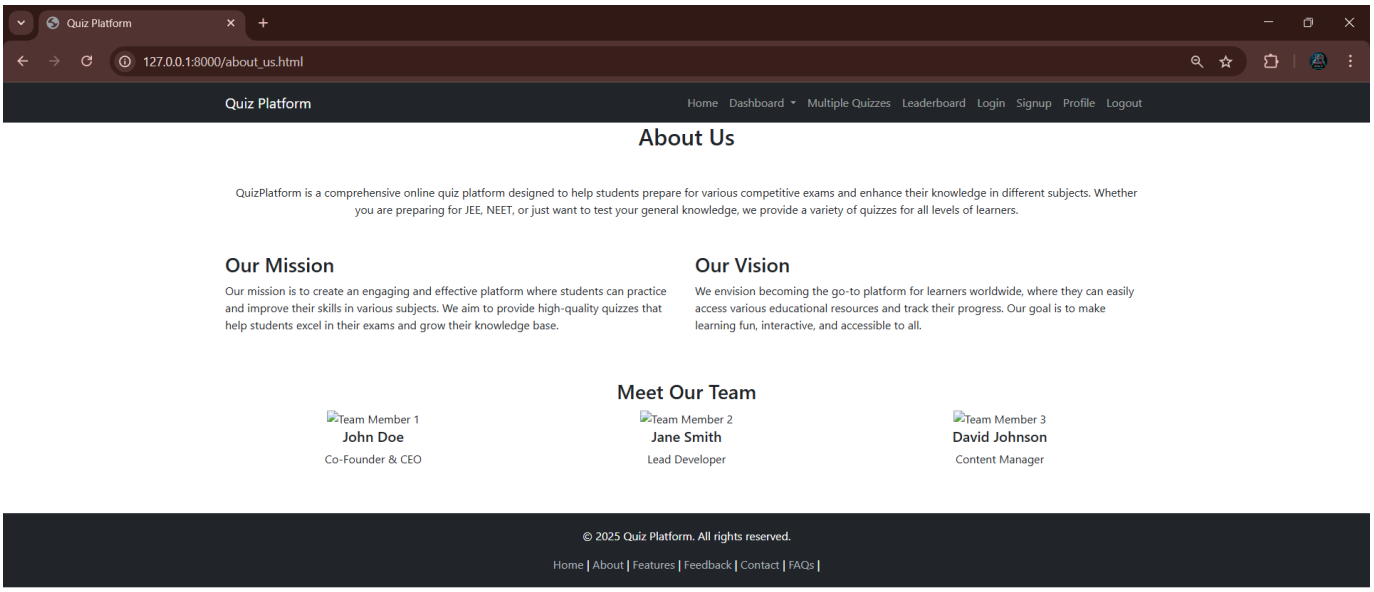
Leaderboard Page.



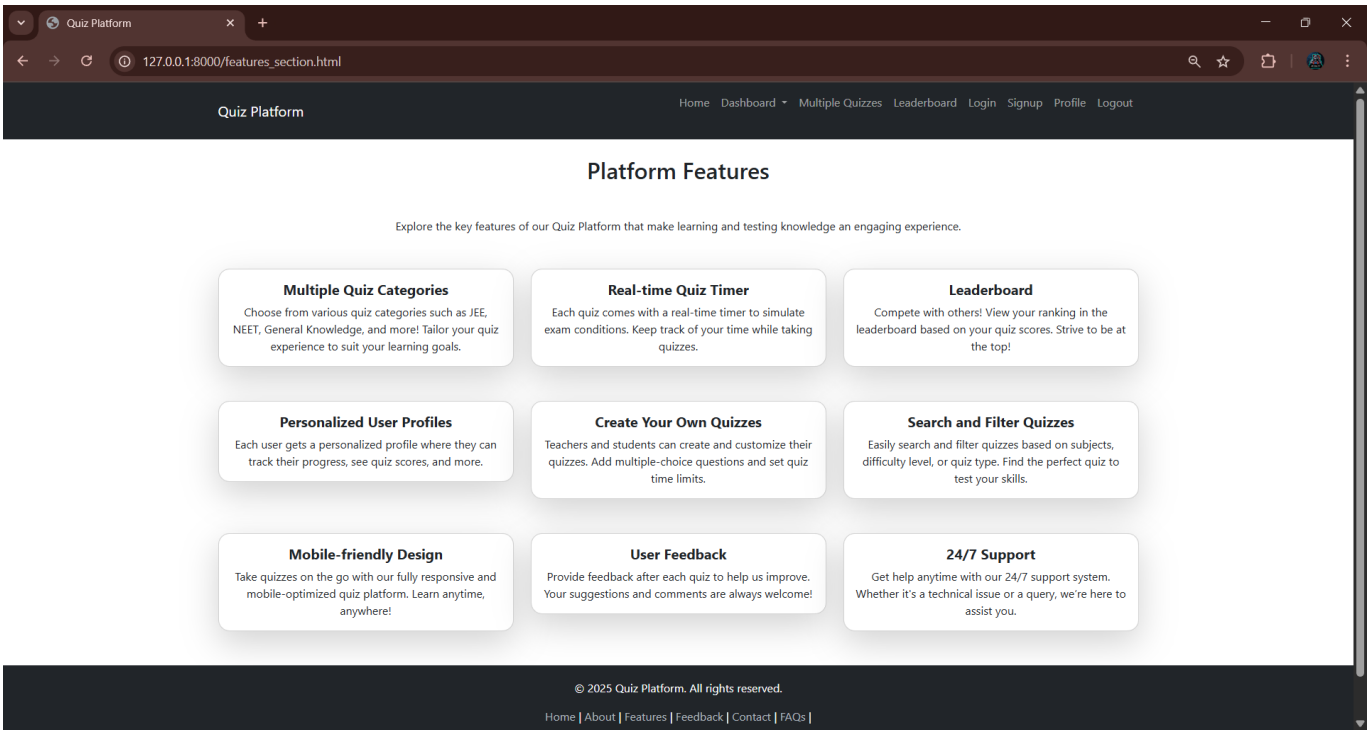
Logout Page.



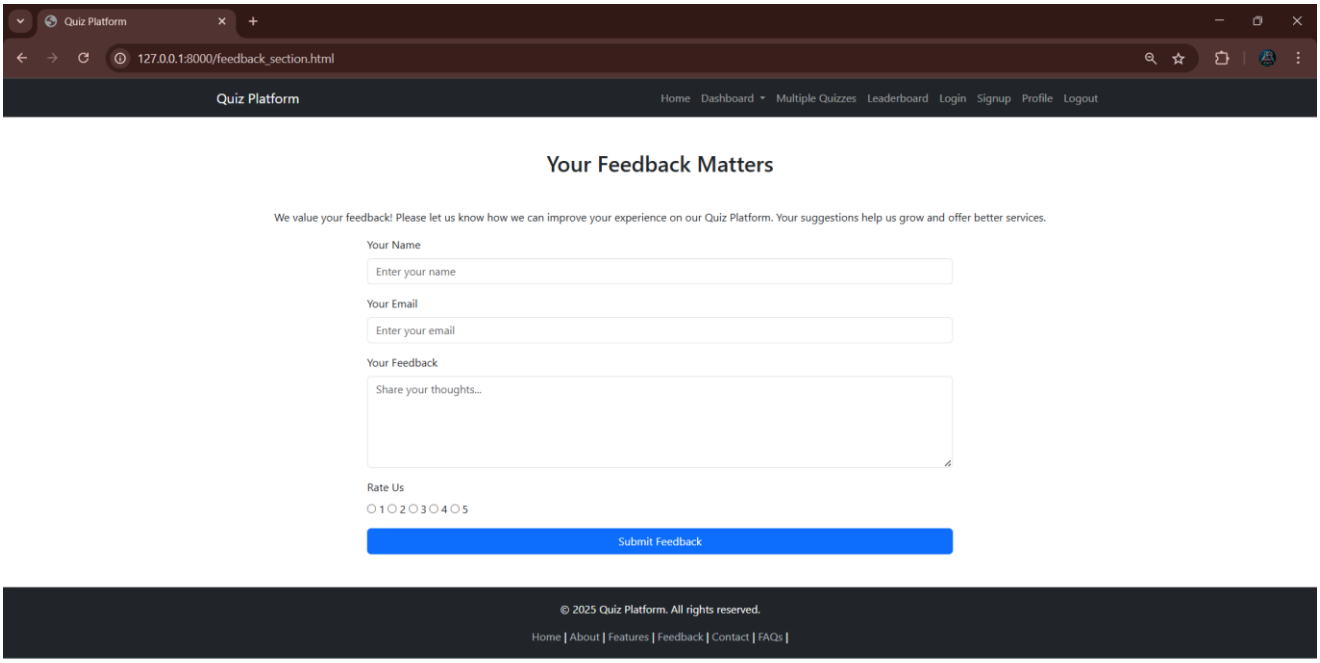
About Us Page.



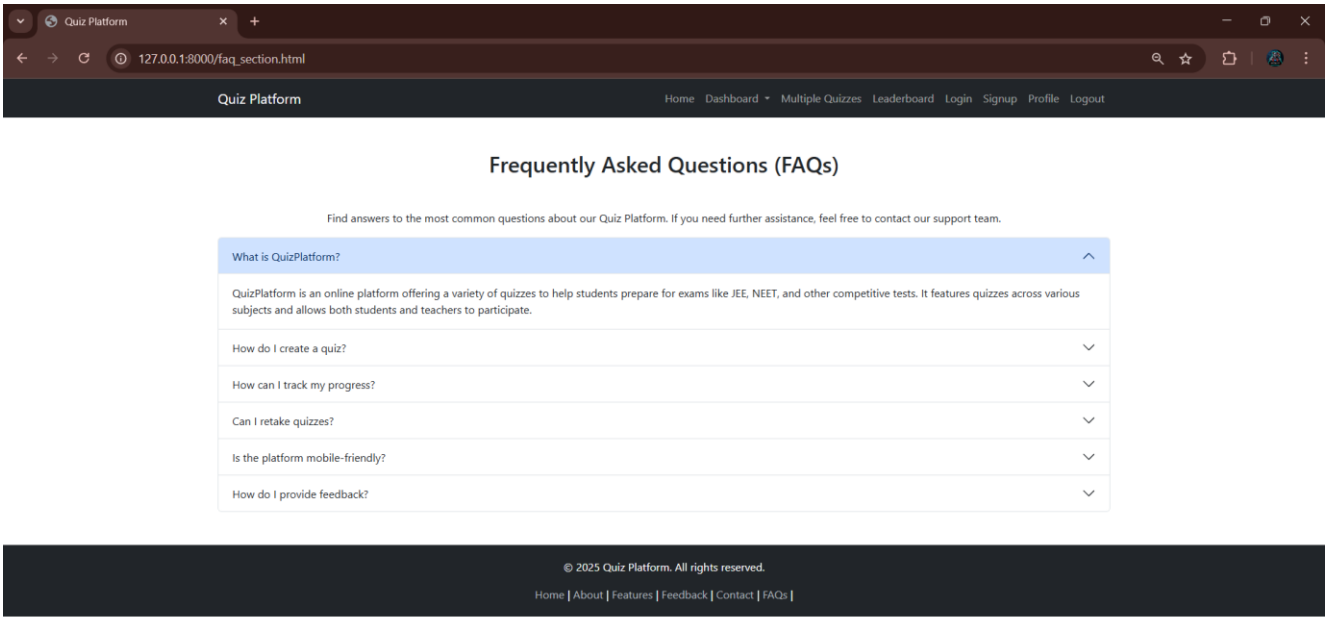
Features Page.



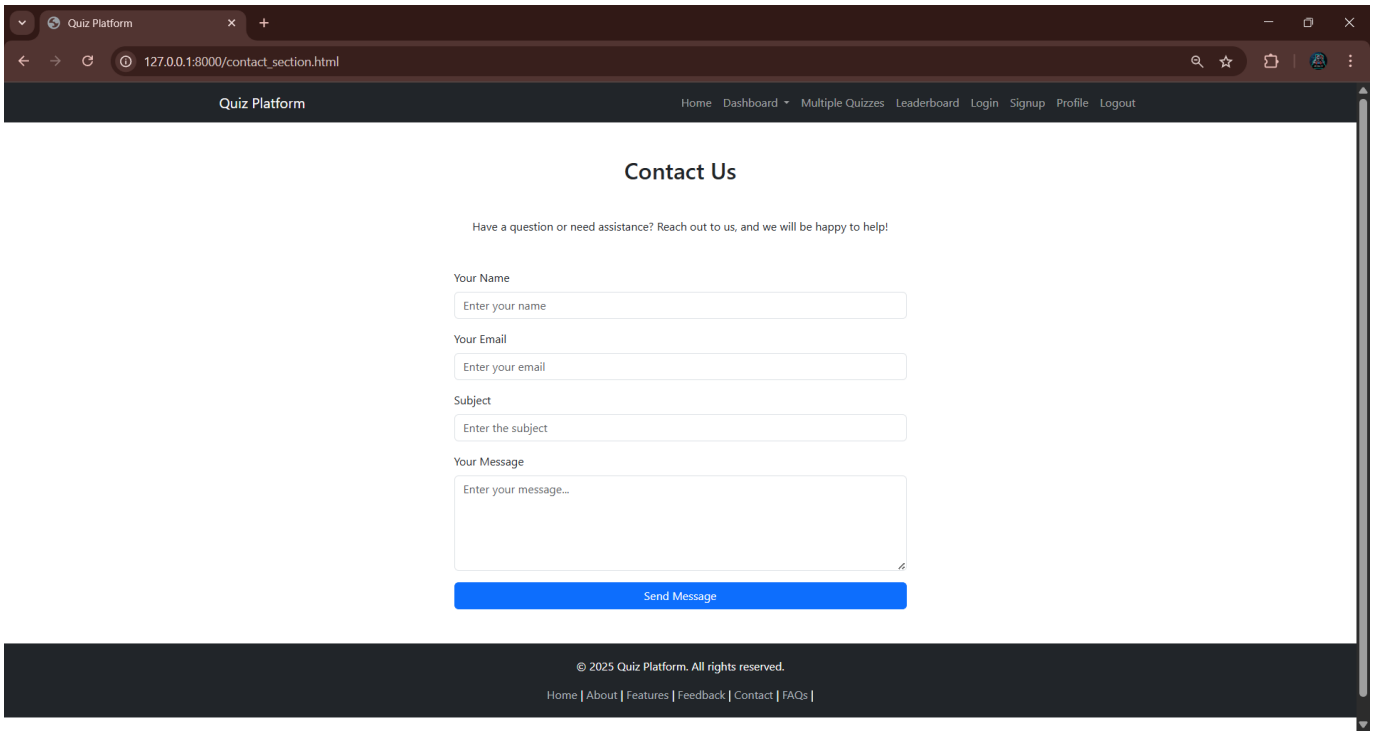
Feedback Page.



FAQ’s Page.



Contact Page .



Ch 6. PROGRAM CODE AND TESTING

6.1. CODE DETAILS AND CODE EFFICIENCY

Index.html

```
 <!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Quiz Platform</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
  <style>
    body {
      padding-top: 56px;
    }
    .hero {
      background: #f8f9fa;
      padding: 80px 0;
      text-align: center;
    }
    .hero h1 {
      font-size: 3rem;
    }
    .hero p {
      font-size: 1.2rem;
    }
    .feature-icon {
      font-size: 3rem;
      color: #0d6efd;
    }
    footer {
      background-color: #343a40;
      color: white;
      padding: 20px 0;
    }
    footer a {
      color: #adb5bd;
```

```
    text-decoration: none;
   }
  </style>
</head>
<body>

  <!-- Navbar -->
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">
   <div class="container">
    <a class="navbar-brand" href="index.html">Quiz Platform</a>
    <!-- <a class="nav-link" href="teacher_dashboard.html">Teacher Dashboard</a>
    <a class="nav-link" href="student_dashboard.html">Student Dashboard</a> -->
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarNav">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
     <ul class="navbar-nav ms-auto">
      <li class="nav-item"><a class="nav-link" href="home.html">Home</a></li>
      <!--<li class="nav-item"><a class="nav-link" href="dashboard.html">Dashboard</a></li>-->
      <li class="nav-item dropdown">
       <a class="nav-link dropdown-toggle" href="/dashborad" role="button" data-bs-toggle="dropdown"
aria-expanded="false">
         Dashboard
       </a>
       <ul class="dropdown-menu">
        <li><a href="teacher_dashboard.html">Teacher Dashboard</a></li>
       <li><a href="student_dashboard.html">Student Dashboard</a></li>
       </ul>
      <li class="nav-item"><a class="nav-link" href="choice_quiz.html">Multiple Quizzes</a></li>
      <li class="nav-item"><a class="nav-link" href="leaderboard.html">Leaderboard</a></li>
      <li class="nav-item"><a class="nav-link" href="login.html">Login</a></li>
      <li class="nav-item"><a class="nav-link" href="signup.html">Signup</a></li>
      <li class="nav-item"><a class="nav-link" href="profile.html">Profile</a></li>
      <li class="nav-item"><a class="nav-link" href="logout.html">Logout</a></li>
     </ul>
    </div>
   </div>
  </nav>
  <!-- Hero Section -->
  <section class="hero">
   <div class="container">
    <h1>Welcome to Quiz Platform</h1>
    <p>Challenge yourself with exciting quizzes, track your progress, and climb the leaderboard!</p>
    <a href="#" class="btn btn-primary btn-lg mt-3">Start Quiz Now</a>
   </div>
  </section>
```

```
  <!-- Features Section -->
  <section class="py-5">
    <div class="container text-center">
      <div class="row g-4">
        <div class="col-md-4">
          <div class="feature-icon mb-3"><i class="bi bi-clock-history"></i></div>
          <h5>Timed Quizzes</h5>
          <p>Practice under pressure with real-time quiz timers.</p>
        </div>
        <div class="col-md-4">
          <div class="feature-icon mb-3"><i class="bi bi-bar-chart-line"></i></div>
          <h5>Leaderboard</h5>
          <p>See how you stack up against others and aim for the top.</p>
        </div>
        <div class="col-md-4">
          <div class="feature-icon mb-3"><i class="bi bi-people-fill"></i></div>
          <h5>Teacher & Student Login</h5>
          <p>Separate portals for teachers and students to manage quizzes.</p>
        </div>
      </div>
    </div>
  </section>

<!-- Footer -->
<footer class="bg-dark text-white text-center my-5">
  <div class="container">
    <p>&copy; 2025 Quiz Platform. All rights reserved.</p>
    <div>
      <a href="home.html">Home</a> |
      <a href="about_us.html">About</a> |
      <a href="features_section.html">Features</a> |
      <a href="feedback_section.html">Feedback</a> |
      <a href="contact_section.html">Contact</a> |
      <a href="faq_section.html">FAQs</a> |
    </div>
  </div>
</footer>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.10.5/font/bootstrap-icons.js"></script>
</body>
</html>
```

Urls.py

```
from django.contrib import admin
from django.urls import path
from home import views

urlpatterns = [

    path("", views.index,
name='home'),
    path("login.html", views.login,
name='login'),
    path("signup.html",
views.signup, name='signup'),
    path("logout.html",
views.logout, name='logout'),
    path("profile.html",
views.profile, name='profile'),
    path("edit_profile.html",
views.edit_profile,
name='edit_profile'),
    path('leaderboard.html',
views.leaderboard,
name='leaderboard'),
    path('quizzes.html',
views.quizzes, name='quizzes'),
```

```
    path('quiz_detail.html',
views.quiz_detail,
name='quiz_detail'),
    path('quiz_form.html',
views.quiz_form,
name='quiz_form'),
    path('JeeExam.html',
views.JeeExam,
name='JeeExam'),
    path('ReetExam.html',
views.ReetExam,
name='ReetExam'),
    path('NeetExam.html',
views.NeetExam,
name='NeetExam'),
    path('RRBExam.html',
views.RRBExam,
name='RRBExam'),
    path('home.html', views.home,
name='home'),
    path('index.html', views.index,
name='index'),
    path('teacher_dashboard.html',
views.teacher_dashboard,
name='teacher_dashboard'),
```

```
    path('student_dashboard.html',
views.student_dashboard,
name='student_dashboard'),
    path("about_us.html",
views.about_us,
name='about_us'),
    path("features_section.html",
views.features_section,
name='features_section'),
    path("feedback_section.html",
views.feedback_section,
name='feedback_section'),
    path("faq_section.html",
views.faq_section,
name='faq_section'),
    path("contact_section.html",
views.contact_section,
name='contact_section'),
    path("choice_quiz.html",
views.choice_quiz,
name='choice_quiz'),
    path("teacher_dashboard.html",
views.teacher_dashboard,
name='teacher_dashboard'),
    path("student_dashboard.html",
views.student_dashboard,
name='student_dashboard'),    ]
```

Views.py

```
#from django.shortcuts import render, HttpResponse
from django.shortcuts import render, redirect
#from django.contrib.auth.models import User
from django.shortcuts import render

#from .models import Quiz
# Create your views here.

def index(request):
    return render(request, "index.html")

    # return HttpResponse("this is a home page")

def login(request):
    return render(request, "login.html")

def signup(request):
    return render(request, "signup.html")

def logout(request):
    return render(request, "logout.html")

def profile(request):
    return render(request, "profile.html")

def edit_profile(request):
    return render(request, "edit_profile.html")

def leaderboard(request):
    return render(request, "leaderboard.html")

def quizzes(request):
    return render(request, "quizzes.html")

def quiz_form(request):
    return render(request, "quiz_forms.html")

def JeeExam(request):
    return render(request, "JeeExam.html")

def RRBExam(request):
    return render(request, "RRBExam.html")

def NeetExam(request):
    return render(request, "NeetExam.html")
```

```
def ReetExam(request):
    return render(request, "ReetExam.html")

def home(request):
    return render(request, "home.html")

def feedback_section(request):
    return render(request, "feedback_section.html")

def features_section(request):
    return render(request, "features_section.html")

def faq_section(request):
    return render(request, "faq_section.html")

def contact_section(request):
    return render(request, "contact_section.html")


def about_us(request):
    return render(request, "about_us.html")

def quiz_detail(request):
    return render(request, "quiz_form.html")

def choice_quiz(request):
    return render(request, "choice_quiz.html")

def teacher_dashboard(request):
    return render(request, "teacher_dashboard.html")

def student_dashboard(request):
    return render(request, "student_dashboard.html")
```

models.py

```
#from django.db import models

# Create your models here.

from django.db import models from
django.contrib.auth.models import User

# Extend user model for student/teacher class
Profile(models.Model):
    USER_TYPE_CHOICES = (
        ('student', 'Student'),
        ('teacher', 'Teacher'),
    )
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    user_type = models.CharField(max_length=10, choices=USER_TYPE_CHOICES)

    def __str__(self):
        return f"{self.user.username} - {self.user_type}"

# Quiz Category class
Category(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
return self.name

# Quiz model class
Quiz(models.Model):
    title = models.CharField(max_length=200)
    category = models.ForeignKey(Category, on_delete=models.CASCADE)
created_by = models.ForeignKey(User, on_delete=models.CASCADE)     created_at
= models.DateTimeField(auto_now_add=True)

    def __str__(self):
return self.title

# Quiz Question class Question(models.Model):     quiz =
models.ForeignKey(Quiz, on_delete=models.CASCADE)     text =
models.TextField()
    option_a = models.CharField(max_length=200)
option_b = models.CharField(max_length=200)     option_c
= models.CharField(max_length=200)     option_d =
```

```
models.CharField(max_length=200)     correct_option =
models.CharField(max_length=1,
choices=[('A','A'),('B','B'),('C','C'),('D','D')])

    def __str__(self):
        return f"{self.quiz.title} - {self.text[:30]}..."

# Quiz Attempt / Result class
QuizAttempt(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE)
score = models.IntegerField()     attempted_at =
models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.user.username} - {self.quiz.title} ({self.score})"

# Feedback Section class Feedback(models.Model):     name =
models.CharField(max_length=100)     email =
models.EmailField()     message = models.TextField()
submitted_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Feedback from {self.name}"

# Contact Us Section class
ContactMessage(models.Model):
    name = models.CharField(max_length=100)     email
= models.EmailField()     subject =
models.CharField(max_length=200)     message =
models.TextField()     sent_at =
models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Contact: {self.name} - {self.subject}
```

admins.py

```
#from django.contrib import admin

# Register your models here.
from django.contrib import admin from
.models import (
    Profile,
    Category,
Quiz,
    Question,
    QuizAttempt,
    Feedback,
    ContactMessage
)

# Registering Profile model to manage user profiles (teacher/student)
@admin.register(Profile) class
ProfileAdmin(admin.ModelAdmin):
    list_display = ['user', 'user_type']
    search_fields = ['user__username', 'user_type']

# Registering Category model to manage quiz categories
@admin.register(Category) class
CategoryAdmin(admin.ModelAdmin):
    list_display = ['name']
search_fields = ['name']

# Registering Quiz model to manage quizzes
@admin.register(Quiz) class
QuizAdmin(admin.ModelAdmin):
    list_display = ['title', 'category', 'created_by', 'created_at']
search_fields = ['title', 'category__name']     list_filter =
['category', 'created_by']

# Registering Question model to manage questions for each quiz
@admin.register(Question) class
QuestionAdmin(admin.ModelAdmin):
    list_display = ['quiz', 'text', 'correct_option']
search_fields = ['text', 'quiz__title']

# Registering QuizAttempt to track quiz attempts by users @admin.register(QuizAttempt)
class QuizAttemptAdmin(admin.ModelAdmin):
```

```
    list_display = ['user', 'quiz', 'score', 'attempted_at']
list_filter = ['quiz', 'user']     search_fields =
['user__username', 'quiz__title']

# Registering Feedback model to manage user feedback
@admin.register(Feedback) class
FeedbackAdmin(admin.ModelAdmin):
    list_display = ['name', 'email', 'submitted_at']
search_fields = ['name', 'email']

# Registering ContactMessage model to manage contact form submissions
@admin.register(ContactMessage) class
ContactMessageAdmin(admin.ModelAdmin):
    list_display = ['name', 'email', 'subject', 'sent_at']
search_fields = ['name', 'email', 'subject']
```

forms.py

```
from django import forms from django.contrib.auth.models import User
from .models import ContactMessage, Feedback, Quiz, Question, Profile

# Contact Us Form class
ContactForm(forms.ModelForm):
class Meta:
    model = ContactMessage        fields =
['name', 'email', 'subject', 'message']
widgets = {
        'message': forms.Textarea(attrs={'rows': 4, 'placeholder': 'Your message...'}),
    }

# Feedback Form class
FeedbackForm(forms.ModelForm):
class Meta:
    model = Feedback        fields =
['name', 'email', 'message']
widgets = {
        'message': forms.Textarea(attrs={'rows': 4, 'placeholder': 'Your feedback...'}),
    }

# Quiz Form - For teachers to create quizzes
class QuizForm(forms.ModelForm):
class Meta:
    model = Quiz        fields
= ['title', 'category']

# Question Form - For creating questions for a quiz
class QuestionForm(forms.ModelForm):    class
Meta:
    model = Question
    fields = ['quiz', 'text', 'option_a', 'option_b', 'option_c', 'option_d', 'correct_option']

# User Profile Form - To handle user profile updates
class UserProfileForm(forms.ModelForm):    class
Meta:
    model = Profile
    fields = ['user_type']  # Only user type, can expand to add other profile fields if needed

# User Registration Form - For user signup class
UserRegistrationForm(forms.ModelForm):
```

```
    password = forms.CharField(widget=forms.PasswordInput)
confirm_password = forms.CharField(widget=forms.PasswordInput)

    class Meta:
        model = User         fields = ['username',
'email', 'password']

    def clean(self):
        cleaned_data = super().clean()         password =
cleaned_data.get("password")         confirm_password =
cleaned_data.get("confirm_password")

        if password != confirm_password:
            raise forms.ValidationError("Passwords do not match")
return cleaned_data

# User Login Form class
UserLoginForm(forms.Form):
    username = forms.CharField(max_length=150)
    password = forms.CharField(widget=forms.PasswordInput)
```

6.2. TESTING APPROACH

Types Of Testing

The system was designed according to the requirement of the system. But we are not 100% confidants. The lack of confidence stems from several things. First the system deals with large number of states, complex logic and activities. So some error might occur in the system. Error may be software, which is known as “SOFTWARE ERROR” i.e. the software doesn’t do what the requirement says. So and exhaustive and thorough testing must be conducted to ascertain. Whether the system produces right results. The project guide and the user both did testing.

Module Testing

The testing was done and several stages. First each program module was tested as a single program, which is also known as module testing or unit testing. In unit testing asset of data as input was given to the module and observed what output data is produced. In addition, the logic and boundary condition for input and output data was also checked. The interface between this module and others was checked for correctness. While collecting the input data for testing the program module it was kept in mind that input should be from all classes, so the entire condition of the program could also be checked.

Interogating Testing

When the individual program modules were working properly, we combine the module in the working system. This integration is planned and coordinated so that when an error occurs, we have an idea of what caused it. Integration testing is the process of verifying that the components of a system work together as described in the program design specification. For testing, the system was viewed as a hierarchy of modules. We began with the module at the highest level of design and worked down. The next modules to be tested are those that call previously tested modules.

Function Testing

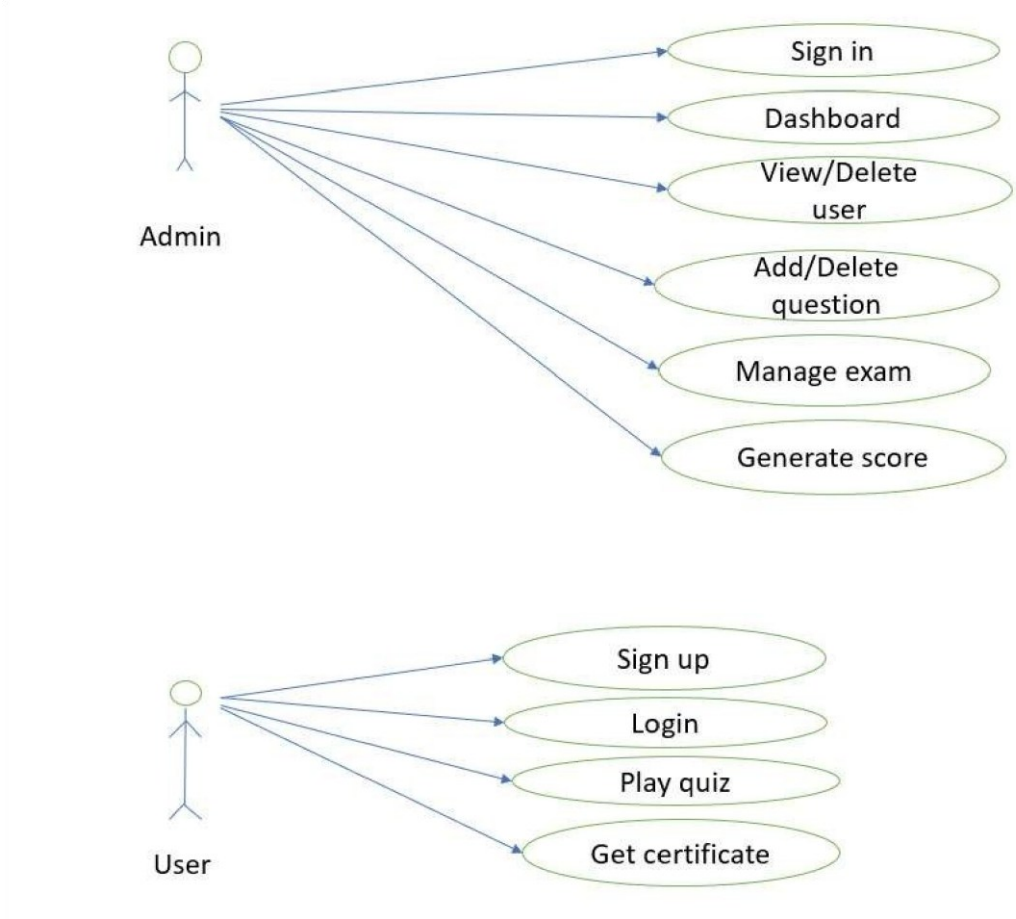
Once we sure that information is passed among modules according to the design prescription we tested the system to assure whether the function described the requirement specification are actually performed by the integrated system.

Acceptance Test

When the function test completes then we involved the user to make sure that system works according to the user’s expectation. Thus, the user did the acceptance test. **Implementation**

Once the system was tested (module wise as well as integrated) satisfactorily, and then comes the implementation of the system. Implementation is the process of changing from old system (manual) to the new system (computerized). Some training was also given to the user about how to work on the new system and finally the system was successfully adopted.

6.2.1 USE CASE DIAGRAM



Ch 7. CONCLUSION

7.1 LIMITATIONS

The size of the database increases day-by-day, increasing the load on the database backup and data maintenance activity.

Instead of providing a specific time for each question, a specific time can be specified for the project as a whole.

There can be a bunch of multiple questions from which admin can choose some as part of quiz because as of know whatever questions are in database every question becomes part of the quiz.

It don't have any industrial implementation, it is just applicable to be implemented on a small scale.

7.2 FURTHER SCOPE

After analysing the answer pattern of questions by the user, we can improve the type of questions based on where majority is going wrong,etc.

A basic tutorial on the quiz can be provided before starting the quiz.

The order of the question can be altered along with the options for the quiz.

A specific time and date can be added from admin's end to start the quiz.

This project can be further customized and additional features /improvements can be made.