Name: Akshay Kumar

Roll Number: 22f3000646

Student Email: 22f3000646@ds.study.iitm.ac.in

Self Introduction: I am student in this program, based in Bihar, India. This is my first experience building and coding a web-based application. I found the process both challenging and rewarding, especially as I navigated the complexities of backend logic, database integration, and front-end design from scratch. The journey taught me the importance of structured planning, debugging, and leveraging community resources to overcome obstacles.

Description

According to my understanding, the aim of this project was to develop an educational web application, **QuizMaster**, that allows users to take quizzes on various subjects and tracks their progress, while providing administrators with tools to manage content (subjects, chapters, quizzes, questions and users) and monitor user performance. The application includes features for user authentication, quiz-taking with a countdown timer, and graphical representations of progress, making it a comprehensive learning management tool. The project evolved from a basic quiz system to include advanced admin functionalities and visual analytics, reflecting my growth in understanding web development concepts.

Technologies used

Technologies Used	Purpose behind using the technology
Python	Used for backend development to handle business logic, route management, and database interactions. Its simplicity and extensive libraries made it ideal for a beginner project.
Flask	This framework was particularly useful in linking my backend python code to my front end HTML.
flask_sqlalchemy	Facilitated the integration of SQLite with the Flask app, allowing easy creation of database tables, relationships, and queries.
Sqlite	Chosen as the database to store and retrieve data (users, subjects, quizzes, etc.), providing a lightweight solution suitable for this project.

Chart.js	Used to create interactive charts (e.g., user performance and score progression), providing visual insights into quiz results.
HTML	Formed the backbone of the front-end structure, displaying data from the backend to users and admins.
Jinja 2	Enabled the creation of reusable templates, reducing redundancy in HTML code and allowing dynamic data display based on backend responses.

Development Process

The development of QuizMaster followed an iterative approach:

1. **Planning**: I began by outlining the database schema and defining user and admin roles based on the project requirements.
2. Backend Development: implemented the 'auth .py' and 'controllers.py' blueprints, starting with authentication and quiz logic, then added the countdown timer and graph functionality.mpkvk,bgglemented the `auth.py` and `controllers.py` blueprints, starting with authentication and quiz logic, then added the countdown timer and graph functionality.

Implemented the `auth.py` and `controllers.py` blueprints, starting with authentication and quiz logic, then added the countdown timer and graph functionality.

Ilemented the `auth.py` and `controllers.py` blueprints, starting with authentication and quiz logic, then added the countdown timer and graph functionality.

3. **Frontend Design**: Using Jinja2 templates, I created quiz_attempt.html and admin templates, integrating Bootstrap for styling and Chart.js for visualizations.

4. **Testing**: I tested individual features (e.g., timer, graph) with mock data, identifying and fixing issues like the 500 error in user_summary.

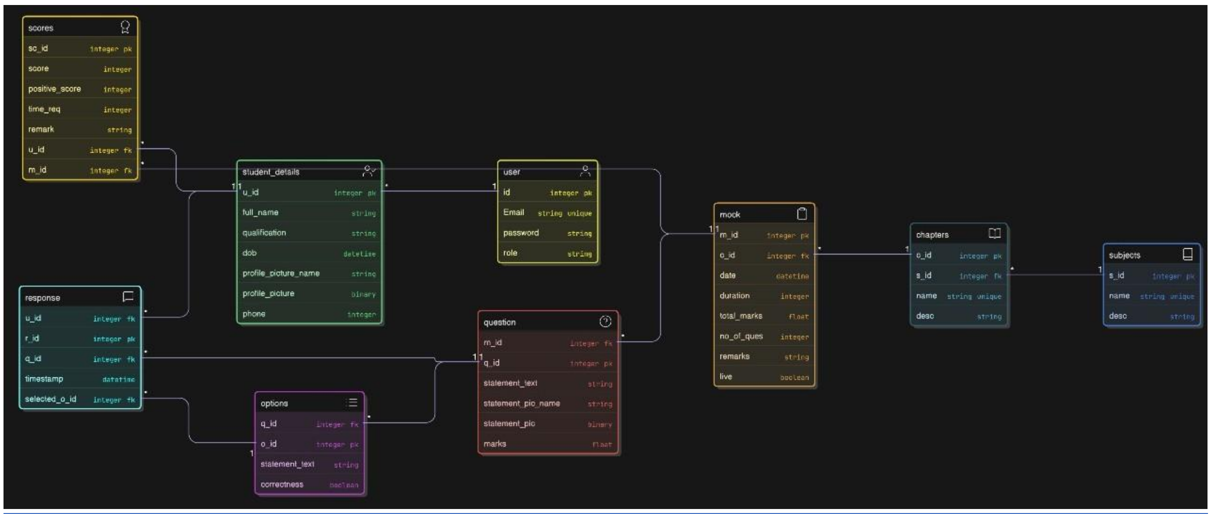
5. **Refinement**: Incorporated feedback from debugging sessions, adding session-based timer persistence and fixing template errors (e.g., missing canvas in user_progress.html).

Challenges Faced

- **Database Integration**: Initial difficulties with SQLAlchemy relationships (e.g., lazy loading issues in user_summary) required learning about joinedload and debugging 500 errors.
- **Timer Implementation**: Creating a countdown timer that auto-submits and persists across reloads was complex, involving JavaScript and Flask sessions, which took significant trial and error.
- **Chart Rendering**: The absence of a canvas in user_progress.html caused chart failures, highlighting the need for thorough template validation.
- **Admin Blueprint**: The lack of an admin.py file left many admin features unimplemented, requiring me to plan additional routes and logic.

- **Schema Errors**: OCR discrepancies in the database schema (e.g., user_quiz_progres s) posed a challenge, necessitating manual corrections.

DB Schema Design and Erd diagram



The database schema for QuizMaster is designed to support the relationships between users, subjects, chapters, quizzes, questions, and progress tracking. The structure is as follows:

Tables and Relationships

1. User Table (user)

- **Primary Key: id**
- **Attributes: Email, password, role**
- **Relationships:**
  - **One-to-One with Student_details**
  - **Users can be either students (default) or admins.**
  - **Passwords are hashed for security.**

2. Student Details Table (student_details)

- **Primary Key: u_id (Foreign Key from user.id)**
- **Attributes: full_name, qualification, dob, profile_picture_name, profile_picture, phone**
- **Relationships:**
  - **One-to-One with User**
  - **One-to-Many with scores**
  - **One-to-Many with response**

3. Subjects Table (subjects)

- **Primary Key: s_id**
- **Attributes: name, desc**
- **Relationships:**
    - **One-to-Many with Chapters**

4. Chapters Table (chapters)

- **Primary Key: c_id**
- **Foreign Key: s_id (Links to subjects.s_id)**
- **Attributes: name, desc**
- **Relationships:**
    - **One-to-Many with Mock**

5. Mock Exam Table (mock)

- **Primary Key: m_id**
- **Foreign Key: c_id (Links to chapters.c_id)**
- **Attributes: date, duration, total_marks, no_of_ques, remarks, live**
- **Relationships:**
    - **One-to-Many with Question**
    - **One-to-Many with scores**

6. Question Table (question)

- **Primary Key: q_id**
- **Foreign Key: m_id (Links to mock.m_id)**
- **Attributes: statement_text, statement_pic_name, statement_pic, marks**
- **Relationships:**
    - **One-to-Many with Options**
    - **One-to-Many with Response**

7. Options Table (options)

- **Primary Key: o_id**
- **Foreign Key: q_id (Links to question.q_id)**
- **Attributes: statement_text, correctness**
- **Relationships:**
    - **One-to-Many with Response**

8. Response Table (response)

- **Primary Key: r_id**
- **Foreign Keys:**
  - **u_id (Links to student_details.u_id)**
  - **q_id (Links to question.q_id)**
  - **selected_o_id (Links to options.o_id)**
- **Attributes: timestamp**
- **Relationships:**
  - **Many-to-One with Student_details**
  - **Many-to-One with Question**
  - **Many-to-One with Options**

9. Scores Table (scores)

- **Primary Key: sc_id**
- **Foreign Keys:**
  - **m_id (Links to mock.m_id)**
  - **u_id (Links to student_details.u_id)**
- **Attributes: score, positive_score, time_req, remark**
- **Relationships:**
  - **Many-to-One with Mock**
  - **Many-to-One with Student_details**

Summary of Relationships

- **One-to-One:**
  - **User ↔ Student_details**
- **One-to-Many:**
  - **Subjects ↔ Chapters**
  - **Chapters ↔ Mock**
  - **Mock ↔ Questions**
  - **Question ↔ Options**
  - **Mock ↔ Scores**
  - **Student_details ↔ Scores**
  - **Student_details ↔ Response**

   - **Question ↔ Response**
   - **Options ↔ Response**

This structured relational database design ensures efficient data handling for user authentication, student management, mock exam evaluations, and performance tracking.

Architecture and Features

- **Backend Code**: Located in user.py (user blueprint) and planned admin.py (admin blueprint) in the current directory.
- **Templates**: Stored in the templates folder, including start_quiz.html, user_dashboard.html, and admin templates (view_chapters.html, manage_quiz_questions.html, etc.).
- **Static Files**: CSS and JavaScript libraries (e.g., Bootstrap, Chart.js) are linked via CDNs.

Features Implemented:

User Features:

- **View Subjects and Quizzes**: Access available subjects and quizzes on the dashboard.
- **Take Quizzes**: Start a quiz with a countdown timer based on quiz.duration, answering multiple-choice questions.
- **View Progress**: Review scores and completion status on the dashboard, with a graphical summary via Chart.js.
- **Review Results**: View completed quizzes with user answers and correct answers.
- **Authentication**: Sign up and log in with username and password.

Admin Features:

- **Subject Management**: Create, edit, and delete subjects.
- **Chapter Management**: Create, edit, and delete chapters, with options to view and create quizzes.
- **Quiz Management**: Manage questions, add new questions, edit or delete existing ones, and delete quizzes.
- **User Progress Monitoring**: View a table of user progress and a score progression chart.
- **Search Users**: Search users by name via a modal on the admin dashboard.
- **User Performance Graph**: Display average scores per user on the dashboard.

API Implementation (api_controller.py)

The API endpoints are implemented in api_controller.py, allowing interaction with the database through HTTP requests. The key endpoints include:

- **GET /subjects** - Retrieves all subjects.
- **GET /subjects/<subject_id>/chapters** - Retrieves all chapters of a specific subject.
- **GET /chapters/<chapter_id>/quizzes** - Retrieves all mock tests for a given chapter.
- **GET /users/<user_id>/scores** - Retrieves scores for a specific user.

The API runs on Flask and uses Flask-SQLAlchemy to interact with the database, making it easy to fetch and manipulate data.

Video link-Presentation Video