Student details

Name – Tulika Kotiyal
Roll number - 23f1000096
Email - 23f1000096@ds.study.iitm.ac.in

Description

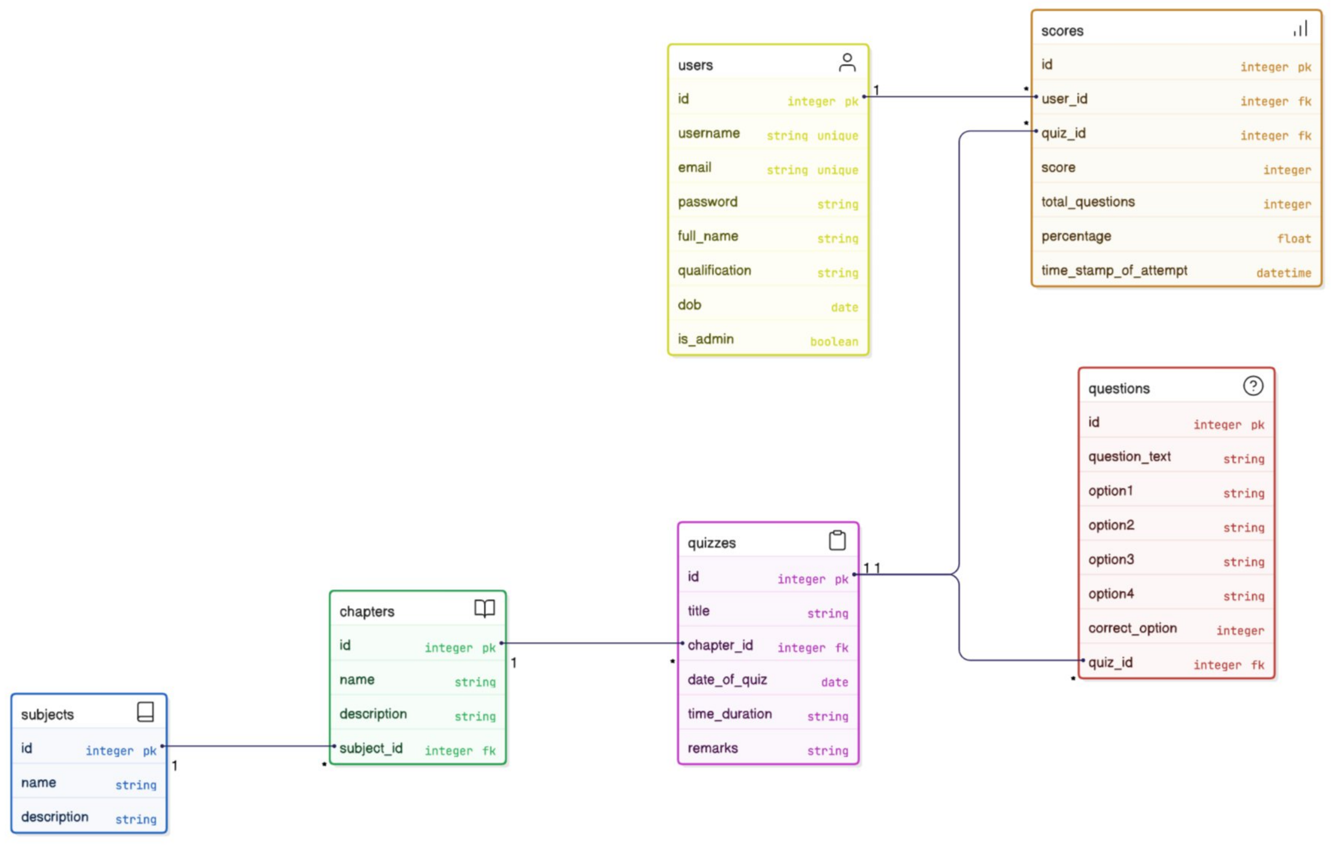
Quiz Master is a multi-user Flask-based web application where an admin (superuser) manages subjects, chapters, and quizzes, while registered users attempt quizzes. Admins create/edit/delete subjects, chapters, and MCQs, while users take timed quizzes and view scores. Built with Flask, SQLite, and Bootstrap, it includes dashboards, score tracking. Focuses on role-based access, quiz management, and user performance analytics.

Frameworks and libraries used

- **Flask framework** was used for developing the project.
- **Jinja2** was used for templating and HTML generation.
- **Flask-SQLAlchemy** and **sqlite3** were used for database operations.
- **HTML/CSS** was used for Structure and styling.
- **Flask-Login/Flask-Security** were used for enhanced authentication.
- **JavaScript** was usedfor frontend validation and interactivity.

DB Schema Design

The database schema encompasses tables for users, score, quizzes, subject, chapters, questions. These entities are interconnected to track various user activities within the system. Each table contains essential fields such as user details, quiz information, scores of the user, subject name, chapter name , and their description. Relationships are established through foreign keys (e.g., quiz_id in Question), enabling structured data flow. Admins (is_admin flag) have elevated privileges. The schema supports CRUD operations for quizzes, users, and analytics.



API Design

They can handle GET/POST/PUT/DELETE or CRUD requests.

🕐 /login (GET, POST)

- **Function:** Handles user login.
- **Description:** Authenticates user credentials. If the user is already logged in, they're redirected to their appropriate dashboard (admin or regular). On a POST request, it validates the provided email and password, logging the user in if valid, or flashing an error message otherwise.

🕐 /register (GET, POST)

- **Function:** Handles user registration.

- **Description:** Enables new users to create an account. It accepts registration details (username, email, password, full name, qualification, and date of birth), adds the user to the database, and flashes a success message before redirecting to the login page.

🕐 **/quiz/int:quiz_id (GET, POST)**

- **Function:** Attempt Quiz.
- **Description:** Allows non-admin users to take a quiz. On a GET request, it displays the quiz with its questions; on a POST, it processes the answers, calculates the score, stores the result, and then redirects to the results page.

🕐 **/quiz/int:quiz_id/results (GET)**

- **Function:** Quiz Results.
- **Description:** Retrieves the latest score for a given quiz attempt by the current user and displays the quiz details along with the score.

🕐 **/users (GET)**

- **Function:** Manage Users.
- **Description:** An admin-only endpoint that lists non-admin users with options to search (by full name, email, or username) and paginate the results. This facilitates effective user management by administrators.

🕐 **/subjects (GET, POST)**

- **Function:** Manage Subjects.
- **Description:** An admin endpoint for handling subjects. It displays a form for adding new subjects and a list of existing ones. On form submission, it validates the data and adds a new subject to the database.

Architecture:

1. **instance/**
   Often used by Flask to store configuration files and the SQLite database (e.g., quiz_master.db).
2. **static/**
   Holds static files like CSS, JavaScript, and images.
3. **templates/**
   Stores your Jinja2 HTML templates.

4. **config.py**
   Central location for Flask configuration (e.g., SECRET_KEY, database URIs, debug settings).
5. **database.py**
   Likely sets up the database connection and initializes SQLAlchemy (or whichever ORM you're using).
6. **forms.py**
   Houses your WTForms (or similar form library) classes.
7. **main.py**
   Often the entry point of your application (though some prefer app.py or wsgi.py).
8. **routes.py**
   Contains your route definitions (i.e., the view functions that respond to specific URLs).

Video:

https://drive.google.com/file/d/1ghEkijZoI6rZpYOTbI3-Wp0bYLj78YrK/view?usp=drive_link