Author

Sristy Gupta

23f1000170

23f1000170@ds.study.iitm.ac.in

Description

Based on the problem we need to create a web application with The goal of this project was to create a web application where users can attempt quizzes on topics of their choice and practice for exams. The app consists of two roles: an admin and regular users. The admin acts as a superuser, with the ability to manage subjects, chapters, quizzes, and more. The users are able to take quizzes and view their previous scores. The admin does not need to register manually, as I have built the registration process for the admin into the app.py

Technologies Used

- **Flask**: A lightweight WSGI web framework used to develop the backend.
- **Flask-SQLAlchemy**: An ORM library to manage database operations in a Pythonic way.
- **SQLite**: A lightweight database for storing user, admin, and service provider data.
- **Jinja2**: A templating engine for rendering dynamic HTML pages.
- **Bootstrap**: A front-end framework used for styling and responsive design.
- **Werkzeug**: A utility library for request and response handling.
- **Flask-Login**: Handles user authentication and session management.

Purpose Behind Using These Technologies:

- **Flask** was chosen for its simplicity and flexibility in handling web applications.
- **Flask-SQLAlchemy** enables efficient interaction with the database using Python objects.
- **SQLite** was selected due to its lightweight nature, making it easy to set up and manage.
- **Jinja2** makes it easier to create dynamic and reusable templates.
- **Bootstrap** provides a responsive and visually appealing UI with minimal effort.
- **Werkzeug** aids in handling HTTP requests efficiently.

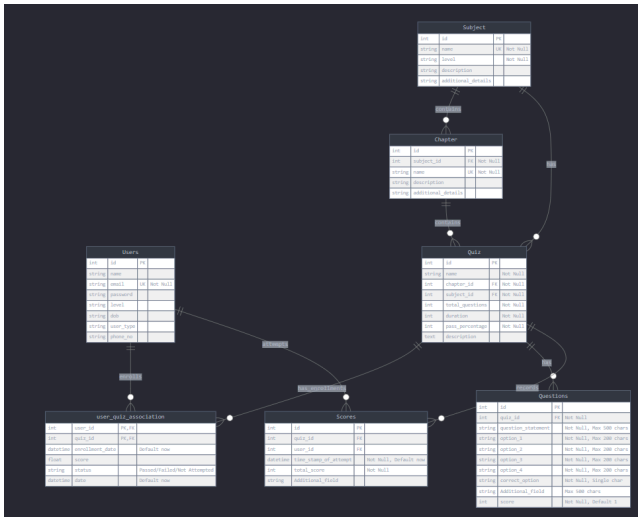
- **Flask-Login** ensures secure user authentication and role-based access control.

DB Schema DesignRelationships between entities:

In accordance with the project requirements, I designed the database models with the following relationships:

- Many-to-many relationship between Users and Quiz through the association table
- One-to-many relationship from Subject to Chapter
- One-to-many relationship from Chapter to Quiz
- One-to-many relationship from Quiz to Questions
- One-to-many relationship from Quiz and Users to Scores

I have attached an image of my Entity-Relationship Diagram (ERD) to illustrate the relationships between the classes. Please note the following abbreviations:



Entity attributes with their:

- Primary keys (PK)
- Foreign keys (FK)
- Constraints like "Not Null" and "Unique"
- Default values where applicable
- Character limits where specified
- **User-Quiz Relationship:** Many-to-many design using junction table to efficiently track multiple quiz attempts and scores.
- **Hierarchical Content Structure:** Three-tier model (Subject → Chapter → Quiz) with cascade deletion to maintain data integrity.
- **Optimized Query Paths:** Direct Subject-Quiz relationship alongside the hierarchical structure to improve filtering and reporting performance.
- **Dual Score Tracking:** Separate mechanisms for quick status checks (association table) and detailed historical records (Scores table).

- **Flexible Question Format:** Structured MCQ design with separate correct answer field to simplify answer verification logic.
- **Security & Administration:** Pre-defined admin account with password hashing, ensuring immediate system usability with basic security.

The architecture balances relational integrity with query performance while allowing for future expansion through additional fields in most tables.

ARCHITECTURE AND FEATURES

I organized the project folder into different subfolders based on their contents and functionality.

- The **applications** folder contains the model.py, app.py and templates
- The **instance** folder contains the database that stores all the entered data.
- The **templates** folder contains the HTML files for the app, including home.html, login.html, register.html, admin_dashboard.html, etc.
- The **requirements.txt** file lists all the dependencies that I initially planned to use for the project.

Some of the key controllers I used include:

1. @app.route('/')
2. @app.route('/login', methods = ['GET', 'POST'])
3. @app.route('/register', methods = ['GET', 'POST'])
4. @app.route('/admindashboard')

VIDEO:
https://drive.google.com/file/d/1yuMb1T0e4z1CfFdG63RY7G_KfNFPYKyx/view?usp=sharing