Author

Ayush Singh
21f3001194
21f3001194@ds.study.iitm.ac.in

I am a passionate learner with keen interest in development and ML. I am a resident of Lucknow pursuing a dual degree (BCA, University of Lucknow & IITM BS). I like learning new things and upgrading myself with new technologies and news. Currently I'm in my diploma and will be going into degree level after next term.

Description

Create a platform for aspirants to prepare different topics by testing themselves with quizzes, which are prepared by the super admin. The admin is the mastermind of everything on the platform as he creates everything on his own. Where as the users can take any quiz based on his/her interest. Along with this, also create other useful features to help aspirants.

Technologies used

Flask for API
SQLITE3 for database storage
VueJS for User Interface
Bootstrap for styling
Redis for caching
Celery for backend batch jobs

Flask is used for creating the API endpoints for all the data operations. Sqlite is chosen as a database because of its simplicity and speed. VueJS is used for structuring the webpages and fetching data. Bootstrap for simply styling the webpage to make it look good. Redis for the purpose of caching to reduce fetch times of frequent APIs. Celery is used for backend batch jobs like daily reminders, CSV reports and monthly reports.

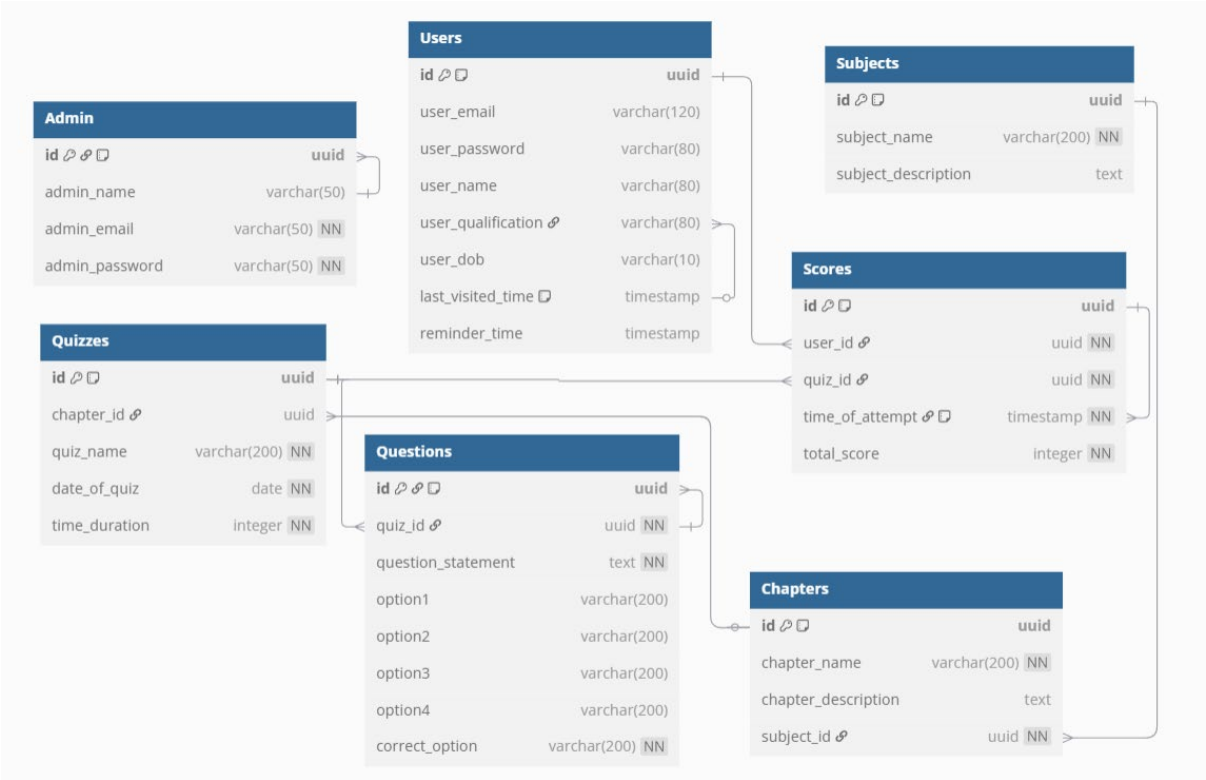
DB Schema Design

The DB contains the following Tables and constraints:

a) **User** Table: Contains User details for login and general details. Columns:
   i) Id: uses uuid4 for unique identification and primary key.
   ii) user_email: unique to prevent duplicate accounts
   iii) user_password; not null, for authentication
   iv) user_name, user_dob, user_qualification: String
   v) last_visited_time: for last login details.
   vi) Reminder_time: for daily reminder time
b) **Admin Table**: Similar constraints as users, ensuring security and uniqueness.

c) **Subjects Table**: name and description as string, id as Primary key.
d) **Chapters Table:** name and description, id as uuid primary key,
   i) Subject_id: id of parent subject, foreign key
e) **Quiz Table:** contains similar columns, id as primary key
   i) Chapter_id: if of parent chapter, foreign key
   ii) Name, date and duration of quiz, not null,
f) **Question Table**: id [PK], question_statement, option1-4, correct_option as general question.
   i) Quiz_id: foreign key id of parent Quiz.
g) **Score Table**: id as primary key, uuid4
   i) user_id: score of user, foreign key user
   ii) quiz_id: id of quiz taken, foreignkey quiz
   iii) time_of_attempt as date and total_score as integer.

IDs of all tables are UUIDs to ensure global uniqueness, and useful for distributed systems. Foreign keys to maintain data integrity, example, no quiz can exist without a chapter. Separate table for users, quizzes, questions to reduce data redundancy and ensure maintainability and scalability.



ER Diagram of Database

API Design

I have created an API for managing quizzes, users, subjects, chapters and questions in the

quiz application using flask, JWT authentication and SQL Alchemy. The API allows both admins
and users to interact with the system based on their roles. Authentication is implemented using JWT tokens, so that only authorized users can access protected routes. Admin can log in and manage subjects, chapters, and quizzes, while users can search for quizzes and participate in them.
Admin can create, update, and delete a chapter/subject/quiz/question. Each request is validated to ensure proper data entry, and UUIDs are used to identify records.
Caching is implemented using Redis cache to improve performance, reducing database queries for frequently accessed data. The API also ensures proper error handling. Returning returning clear messages for missing data.

Architecture and Features

The project is structured to maintain clarity, and scalability. The main application logic lies inside the app.py, which initializes the Flask app, configures JWT, Redis, cache, and celery. The routes.py defines all the routes, models.py contains database models defined using SQL Alchemy, tasks.py contains the backend celery tasks and workers.py initializes the celery workers configurations, and all these are placed inside application folder. The database files and exported files contain DB and csv/pdf reports respectively. Additionally, Redis caching is used to improve performance, and JWT-based authentication ensures secure access for both admins and users. The well-structured organization of the project allows for easy maintenance and future feature enhancements. All the backend logic and configurations are grouped in backend folder.

The frontend folder on the other hand, is totally separated from the backend for better maintainability. The frontend is built with Vue.js and follows modular component-based structure. The main view files are stored inside src/views folder separating concerns for admin and user views into different folders. The views such as User dashboard, Admin dashboard, Ai Agent, leaderboards, profile, login/register, etc are stored separately in src/views folder. Whereas the src/components folder contains repetitive components like separate navbars, footers and sidebars for users. Also, the Chart.JS charts are stored inside ser/assets/charts folder which summarize the user's performance. The Vue router manages the navigation between pages, app.js provides the basic structure to the website. The separation of frontend and backend allows for better scalability, making it easy to add new features without disrupting existing functionality.

The project includes a range of features, ensuring smooth and interactive experience for both admin and users. The default features include authentication with JWT token, for secure login framework. Admin can create, update, delete the chapters, subjects, quizzes and questions. The users can take the quiz within a time limit specified by the admin, select MCQ answers and submit to get the result, which are then stored in DB for future references and summarizing. The summary page summarizes the user's performance for both admin and user using Chart.js.

Several additional features have been implemented to enhance the functionality of the app. A celery-based task scheduling system enables daily reminders for inactive users and a monthly activity report that summarizes quiz performance and average scores. A search feature allows admins and users to find subjects, quizzes, and questions efficiently. Caching has been done to improve API response times; an export feature allows users and admin to download quiz data as CSV files. Admins can also trigger tasks for exporting user quiz data asynchronously. The monthly report is also present in PDF formats for the user to download. The frontend, developed in Vue.js communicates with the backend using fetch-Api ensuring dynamic and responsive user experience. Apart from that, there is an additional feature of Ai Agent which assist users to search for a topic and generates short summary on that topic for the user to learn.

Video Demonstration

https://drive.google.com/file/d/1sjGZBxGSFnHLbpXSzNtF1RIibRytXfpp/view?usp=sharing