QuizMaster v-2

MAD-2 Project

Srikanth Rajkumar
Associate Consultant, Aon
Roll No: 21f3001497
21f3001497@ds.study.iitm.ac.in

SUMMARY

Quiz Master V2 is a multi-user exam preparation platform where authenticated users can take quizzes by subject and track performance. Admins manage quiz content, users, and performance analytics via a role-based backend.

Disclosure: Limited portions of the technical architecture were developed with the assistance of *Claude. My estimate is that this accounts for 20-30% of the code.*

HOW I APPROACHED THE PROBLEM DIFFERENTLY?

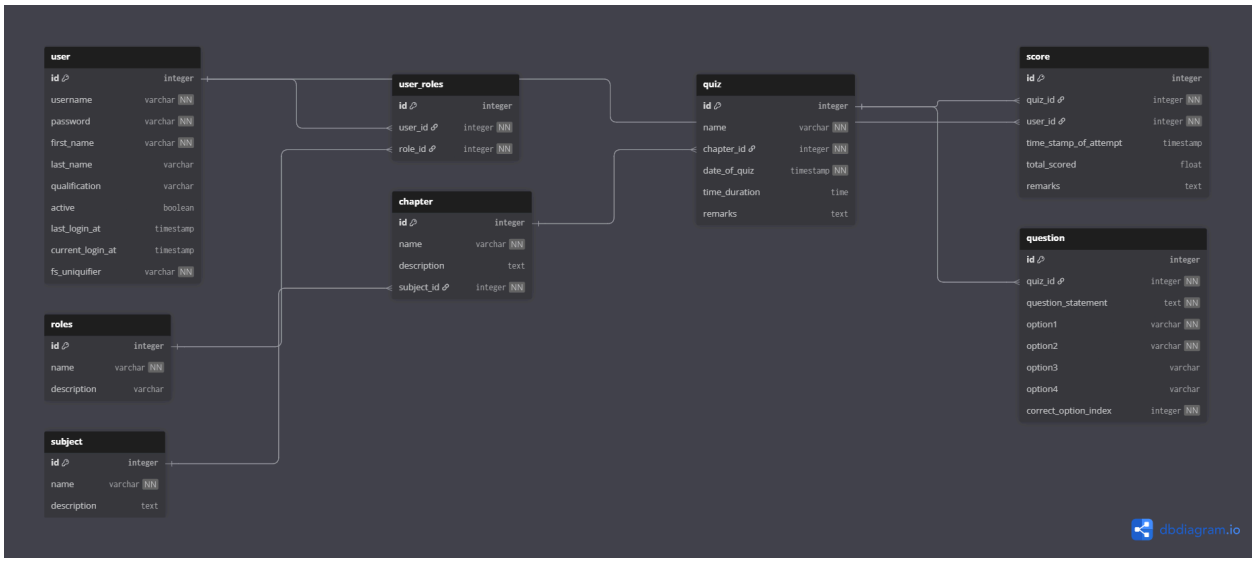
- **Specialised API Endpoints:** Apart from generic CRUD endpoints, I also implemented role-specific endpoints (example: `/api/admin/summary`, `/api/user/attempt`) to separate admin and user functionalities.
- **Minimalist Design, Maximum Function:** This approach keeps the architecture simple and clean while ensuring the system delivers all key features.

TECH STACK

- **Python (Flask)** – building APIs and backend logic
- **Flask-RESTful** – structuring REST API resources cleanly and efficiently
- **SQLite** – storing quiz, user, and score data
- **Flask-Migrate (Alembic)** – managing database migrations
- **Redis** – queuing background jobs and caching responses
- **Celery** – scheduling and running background tasks (reminders, reports)
- **Vue.js – Vue3** – rendering frontend UI and handling routing
- **Vite** – bundling and serving the frontend during development and build
- **Bootstrap** – styling responsive components
- **Flask JWT Extended** – securing API endpoints via token-based auth
- **Axios** – fetching data from backend APIs

- **Chart.js** + **vue-chartjs** – visualizing data with interactive charts

DATABASE SCHEMA DIAGRAM



The database schema follows a standard relational design with the entities recommended by the IITM team i.e User, Role. Subject, Chapter, Quiz, Question, Score.

API STRUCTURE

The API design employs **specialised and purpose-specific endpoints**, such as distinct routes for user summaries, admin dashboards, quiz displays, and CSV exports. This separation of concerns improves clarity, simplifies frontend integration, and enables fine-grained control over authentication and caching.

Link to API Endpoints in Google Sheets: (Link)

OpenAPI YAML file in the Zip.

ARCHITECTURE

The project is structured into backend and frontend components. The backend is located in the `backend/app` directory, where the `api` folder houses all the route controllers organized by functionality (e.g., users, quizzes, chapters). The `models.py` file defines the SQLAlchemy models, while `jobs.py`, `celery_app.py`, and related files manage background tasks using Celery. HTML templates and static assets (like CSS or JS for Flask-rendered views) are located in the `templates` and `static` folders respectively.

On the frontend side, the Vue.js single-page application is located in the `frontend` directory. The source code resides in `frontend/src`, with subfolders like `components`, `pages`, `router`, `services`, and `utils` organizing the Vue components, route definitions, and service logic. Build configuration is handled by `vite.config.js`. The `instance` directory contains the SQLite database and exported CSV files, while database migrations are managed through the `migrations` folder using Alembic.

VIDEO DEMO

Link to Summary Video:

https://drive.google.com/file/d/1NcWKKo_8NBbTKQCuGlNIDzlO1PWUWxuh/view?usp=sharing