Quiz Master - MAD2 Project Report

Name: Harsh Jayswal
Email: 22f3002188@ds.study.iitm.ac.in

Summary:

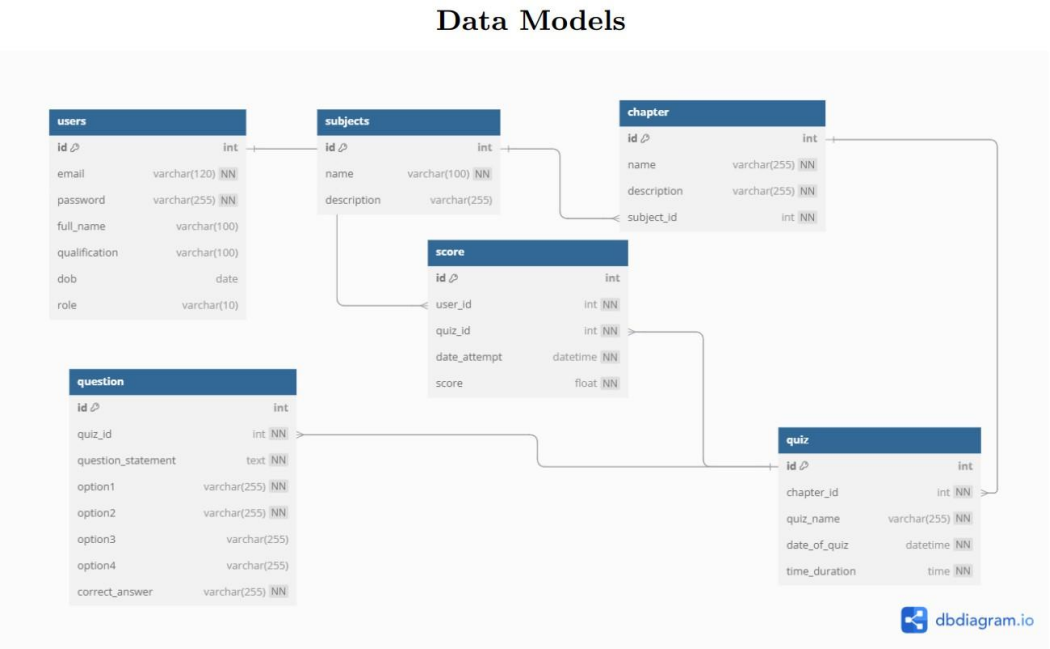
An enhanced multi-user platform for creating and attempting quizzes via a comprehensive RESTful API. New MAD2 features include JWT-based authentication and role management, API documentation with Swagger UI, Redis caching for performance, Celery-powered background tasks for CSV exports and email notifications, and systematic use of Flask extensions for modularity and scalability.

Materials Used:

- Python 3. Flask, Flask-JWT-Extended, Flask-Mail, Flask-Caching, Flask-CORS, Flasgger (SwaggerUI), Celery, Redis
- SQLite3 for persistent data storage• ReportLab for PDF report generation

Data Models:

The following diagram illustrates the core database schema for users, subjects, chapters, quizzes, questions, and scores, showing relationships and key attributes.



Core Functionality:

- User Management: Secure signup/login with SHA-256 password hashing, JWT tokens withrole-based access and revocation.
- Admin Operations: CRUD endpoints for subjects, chapters, quizzes, questions, and usermanagement; search and summary analytics.
- User Operations: Browse and attempt quizzes, submit answers, view scores and charts.
- Performance: Redis caching on read-heavy endpoints (e.g., subjects), reducing DB load.

• Background Tasks: Celery tasks for CSV exports and email notifications upon completion.
• Documentation: Swagger UI auto-generated API docs at /apidocs/ for easy exploration.

Development Approach:

• Extended MAD1 codebase: Refactored to API-first design with Flask Blueprint structure.
• Integrated JWT for secure role-based access controls.
• Added Redis and Flask-Caching for performance optimization.
• Set up Celery with Redis broker/backend for asynchronous exports and email notifications.
• Documented all endpoints with Flasgger and tested via Swagger UI and Postman.

Learnings:

• Implementing JWT and secure token revocation strategies.
• Designing and consuming background tasks with Celery and Redis.
• Leveraging caching strategies to improve API responsiveness.
• Auto-generating API documentation for developer onboarding.
• End-to-end testing of asynchronous workflows and email dispatch in Flask.

Project video available at:
https://drive.google.com/file/d/1A9ynjeqpx4Kcmuoix3qczINuLKsrFhkB/view?usp=sharing

AI Usage:

Used AI tools for debugging errors and taking help only to understand concepts. Also used AI for creating summary charts. Approximate AI usage: 20-30%.