Quizdom V2 - Project Report

Modern Application Development – II

—

Kavisha Tankle
23F1000041
23f1000041@ds.study.iitm.ac.in
Learning programming and data science.

Description

Quizdom V2 is a multi-user exam preparation platform for multiple courses. It supports two roles: an administrator (quiz master) and regular users. The admin can manage subjects, chapters, quizzes, and questions, while users can register, log in, attempt quizzes, and view their scores and statistics. The platform features real-time quiz attempts, analytics, and automated email notifications.

Technologies Used

1. Frontend: Vue.js, Bootstrap, Chart.js (vue-chartjs) — for responsive UI and charts
2. Backend: Flask, Flask-RESTful, SQLAlchemy — for REST API and ORM
3. Database: SQLite — lightweight, easy to use
4. Caching/Async: Redis, Flask-Caching, Celery — for performance and scheduled/background jobs
5. Email: Mailhog — for development/testing of email notifications
6. Authentication: Flask-JWT-Extended — secure, stateless authentication

DB Schema Design

Schema Link

Design Rationale

The database is structured with separate tables for users, subjects, chapters, quizzes, questions, and scores. Each table contains only the fields relevant to that entity, such as usernames and emails in the user table, or quiz names and durations in the quiz table. Relationships are established using foreign keys; for example, each quiz links to a chapter, and each score links to a user and a quiz. Constraints like unique usernames and emails, and foreign key references, ensure data integrity and prevent duplication. This normalized design makes the data model efficient, easy to maintain, and well-suited for analytics and reporting.

API Design

YAML File Link

API Elements and Implementation

RESTful APIs were created for all core elements of the platform, including users, subjects, chapters, quizzes, questions, and scores. Each entity has endpoints for creating, reading, updating, and deleting (CRUD) operations, with access controlled by JWT authentication and user roles. Additional APIs support user registration, login, quiz attempts, analytics (for admin and users), score export, and scheduled background tasks. All endpoints are implemented in Python using Flask-RESTful, with request/response validation and error handling. The complete API structure, including request/response formats and authentication, is documented in the accompanying YAML (OpenAPI) file.

Architecture and Features

Project Structure Link

I. Architecture

The project is organized into two main parts: a backend and a frontend.

- ➔ The backend (backend/ folder) contains all server-side logic. Controllers and API endpoints are in api.py, database models are defined in models.py,

configuration is in config.py, and background tasks are handled by task.py and worker.py.

- ➔ The frontend (frontend/ folder) is built using Vue.js CLI. All user interface components (such as quiz, admin, login, statistics, and scorecard pages) are in src/components/, with routing managed in src/router/.
- ➔ Templates for emails and reports are handled in the backend, while all main UI is built as Vue components.

II. Features

Features implemented include:

- ➔ User registration, login, and JWT-based authentication
- ➔ Admin management of subjects, chapters, quizzes, and questions
- ➔ Users can attempt quizzes, view/export scores, and see analytics
- ➔ Data visualization with Chart.js (via vue-chartjs)
- ➔ Automated email reminders and report exports using Celery
- ➔ Responsive design with Bootstrap

All features are implemented as modular API endpoints and Vue components for maintainability and scalability.

Video

Video Presentation Link

A full walkthrough of the app, demonstrating its features and navigation.