MAD-2 PROJECT REPORT

Quiz-App

Submitted by:

Nitya

23f3003687
23f3003687@ds.study.iitm.ac.in
I am a dedicated learner with an interest in web development.This application demonstrates not only technical proficiency in Flask, Vue.js, Redis, and Celery but also a deep understanding of real-world problem-solving, API design, and asynchronous task management. The entire system is developed independently with a strong emphasis on originality, clarity, and practical usability.

DESCRIPTION

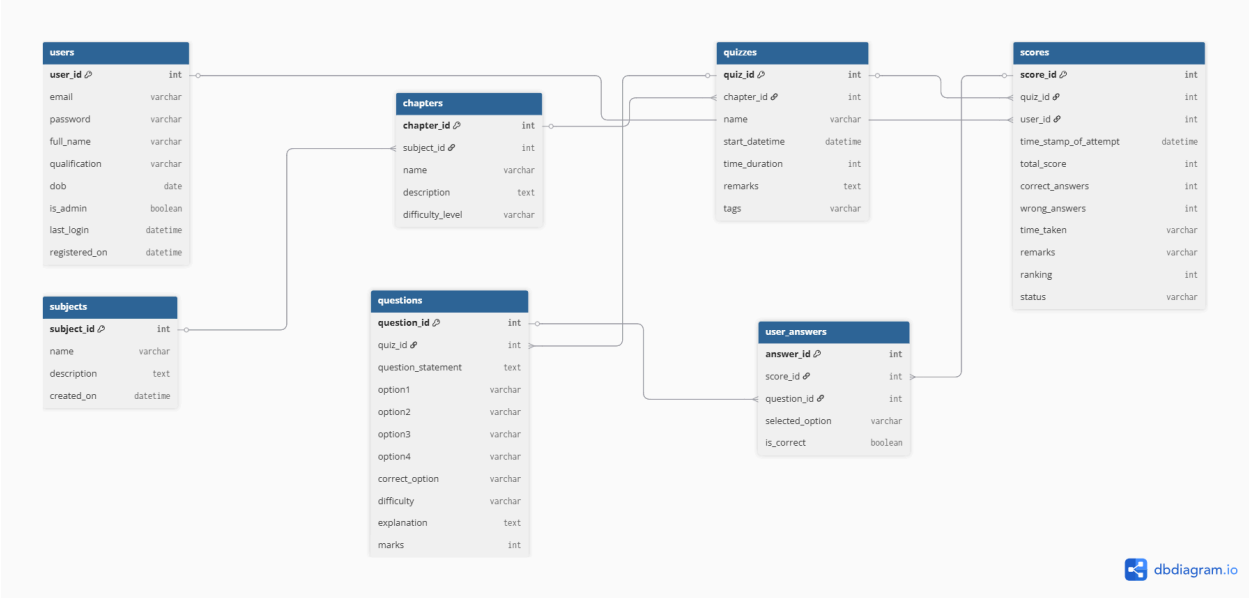
The goal of this project is to build a multi-user quiz platform with two roles: Admin and User. The Admin can manage subjects, chapters, quizzes, and questions, while users can register, attempt quizzes, and view scores. The system also includes scheduled and asynchronous tasks using Redis and Celery for reminders, reports, and CSV exports.

TECHNOLOGIES USED

Technology / Library	Purpose
Flask	Web framework used to build the backend API
Flask-JWT-Extended	For implementing JWT-based authentication and role-based access control
Flask-SQLAlchemy	ORM for interacting with the SQLite database
Flask-Caching	To cache frequently accessed routes and improve API performance
Flask-CORS	To enable Cross-Origin Resource Sharing between frontend and backend
SQLite	Lightweight and serverless database used for persistent data storage
Vue.js	JavaScript framework used for building the dynamic frontend UI

Bootstrap	For responsive and consistent UI styling
Jinja2 (CDN entry only)	Used only to load Vue.js from entry point (no UI templating)
Redis	In-memory data store used for caching and task queue backend
Celery	For asynchronous and scheduled background task execution
Chart.js	To render data visualizations like pie and line charts in the dashboard
Python's `uuid` module	For generating unique filenames in export functionality
smtplib / email utils	To send monthly email reports and daily quiz reminders

DB Schema Design:



API DESIGN:

We have created a comprehensive set of RESTful APIs using **Flask** to support all core functionalities of the project. These APIs are structured around role-based access for both **Admin** and **User**, and protected using **JWT authentication**.

Key API Categories:

1. **Authentication APIs**
    - User registration, login
    - Admin login (pre-created)
    - Token-based access control using `Flask-JWT-Extended`

2. **Admin APIs**
    - Create/edit/delete Subjects, Chapters, Quizzes, and Questions
    - Role-protected routes for managing quiz content
    - Dashboard summary and search functionality
    - Trigger CSV export of user performance (via Celery task)

3. **User APIs**
    - Fetch quizzes by subject/chapter
    - Attempt quiz and submit answers
    - View scorecard and score history
    - View dashboard summary and analytics
    - Trigger personal quiz attempt export (via Celery task)

4. **Celery Task APIs**
    - Routes for initiating background tasks (CSV exports)
    - Asynchronous and scheduled tasks handled using Celery and Redis

All API responses are returned in JSON format, and proper error handling is implemented for failed operations (e.g., invalid tokens, missing data, unauthorized access).

> The complete OpenAPI YAML specification for these APIs is submitted separately as part of the deliverables.

Architecture and Features:

The project follows a modular architecture structured around the **Model-View-Controller (MVC)** design pattern. The backend is built using **Flask**, and the frontend is implemented using **Vue.js** served via CDN, with **Bootstrap** for styling.

Backend Structure (`/backend`)

- **`app.py`** – Entry point for the Flask application.
- **`config.py`** – Centralized configuration settings.
  `extensions.py` – Initializes Flask extensions like JWT, SQLAlchemy, etc.
  `models.py` – Contains all SQLAlchemy database models (User, Quiz, Question, etc.).
  `routes/` – Modular route files for auth, admin, user, etc.
  `tasks/` – Celery tasks such as daily reminders, monthly reports, CSV export.
  `celery_utils.py` / **`celery_worker.py`** / **`celery_beat.py`** – Celery + Redis integration.
  `static/exports/` – Stores generated CSV files for export.
- **`temp_reports/`** – Temporary storage for monthly reports before mailing.
  `utils/` – Helper functions used across routes and tasks.
  `instance/` – Instance folder for runtime configuration like SQLite DB.
- **`.env`** – Environment variables for secrets and config.
- **`openapi.yaml`** – OpenAPI-compliant YAML file defining all backend APIs for submission.

Frontend Structure (`/frontend`)

- `src/components/` – Reusable UI components such as:
  - **Modals** – EditChapterModal, EditQuizModal, etc.
  - **Navbar** / **Sidebar** – Different ones for Admin and User roles.
    Charts – Bar, Line, and Pie charts for dashboard analytics.
- `src/views/` – Vue pages tied to frontend routes:
  - **Admin Views** – AdminDashboard, AdminSearch, etc.
  - **User Views** – UserDashboard, QuizAttempt, ScoreCard, UserSearch, etc.
- `router/index.js` – Vue Router configuration with protected routes based on roles.
- `main.js` & `App.vue` – Vue entry point and root component.
  `index.html` – Jinja2 entry point used to load the Vue app via CDN.
  `vite.config.js` – Vite bundler configuration for local dev and production builds.

Features Implemented

- **User Authentication**: JWT-based login/registration with separate logic for Admin and User.
- **Admin Functionality**:
  - CRUD operations on Subjects, Chapters, Quizzes, and Questions.
  - Role-based access to sensitive APIs.
  - Summary dashboard with visual charts (Pie, Bar, Line).
  - Search functionality to find users, quizzes, or subjects.
  - Export all users' quiz performance as CSV via Celery.
- **User Functionality**:
  - Browse and attempt quizzes based on subject/chapter.
  - Quiz timer and automatic score computation with accuracy and remarks.
  - Personalized dashboard showing activity log and performance graphs.
  - Export individual quiz history as CSV (triggered job).
- **Async & Scheduled Jobs**:
  - **Daily Reminder**: Email sent to inactive users or if new quizzes are published.
  - **Monthly Report**: HTML email report summarizing user's quiz performance.
- **Performance Enhancements**:
  - Caching of quiz lists, search results, and summary reports using **Flask-Caching** with **Redis**.
  - Controlled cache expiry for freshness of data.

All features are developed with extensibility, role isolation, and local deployment compatibility in mind.

AI Usage: *Approximately 35-40 % of the project was completed with AI assistance. ChatGPT was used for generating boilerplate code (models, routes, OpenAPI YAML), and drafting Vue components. All AI-generated artefacts were manually reviewed, tested, and integrated to ensure correctness and originality.*

VIDEO Link : https://drive.google.com/file/d/1NsuWzCvUXZcU7KSmP5ErMpTSvwul_H0H/view?usp=sharing