MAD 2 Project

Smartprep V2

Name: Saini Nirmal

Roll Number: 23f1002643

About Me:

I am currently in the 4th term of the diploma level. I have a strong passion for solving complex problems and continuously exploring new technologies. My curiosity drives me to learn new programming languages and develop innovative solutions. I am particularly interested in data analysis and machine learning, which fuel my enthusiasm for deriving meaningful insights from data. I aspire to push the boundaries of technology and build impactful applications.

Description:

This project, Quiz Master V2, is a role-based web application designed to serve as an exam preparation platform. It provides a seamless interface for an administrator to create and manage all educational content (subjects, chapters, quizzes), while registered users can take these quizzes, track their scores, and view performance statistics to aid their learning.

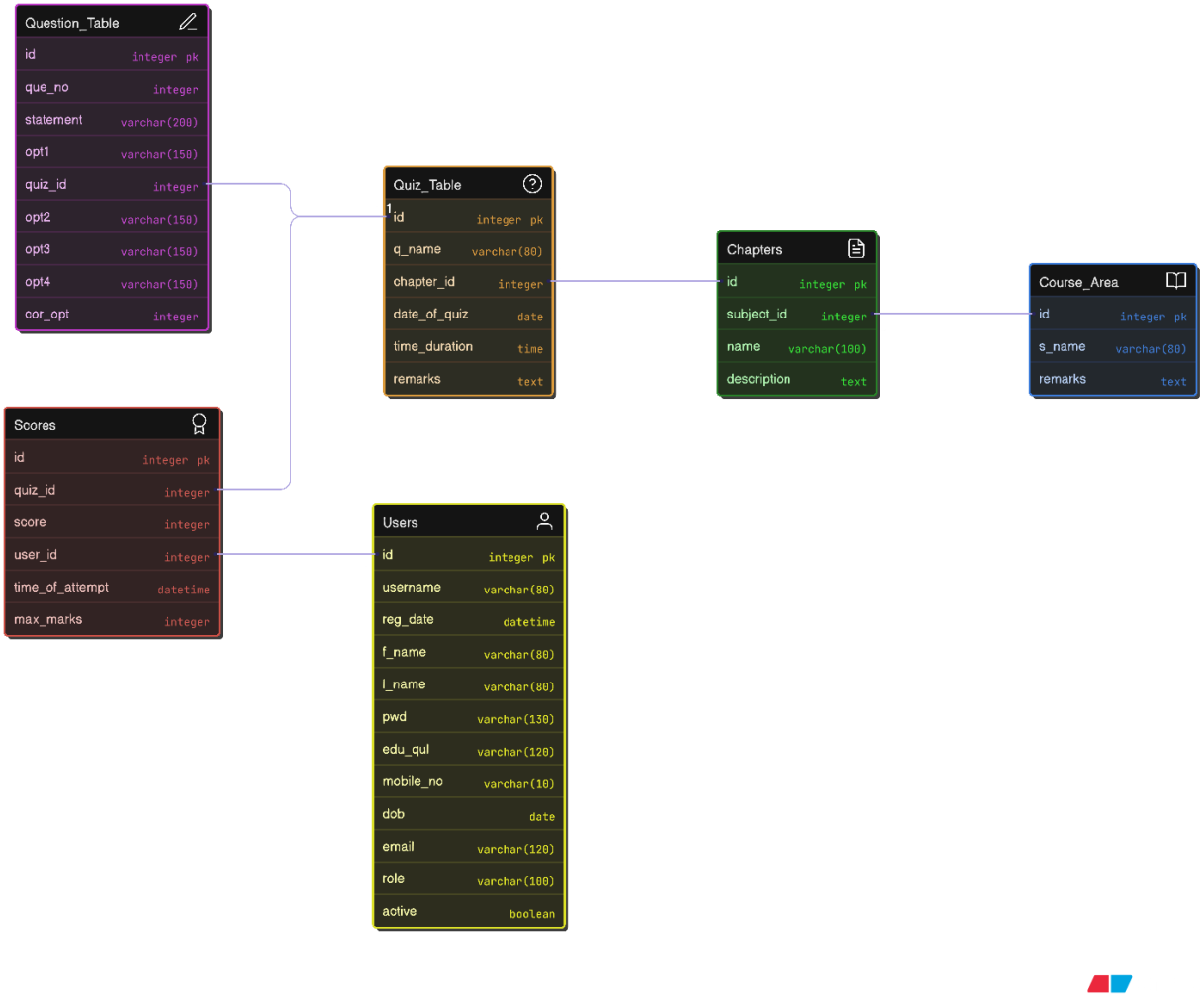
- **AI/LLM Usage:** AI assistance was utilized for specific, targeted tasks to improve development efficiency. GitHub Copilot provided suggestions for Bootstrap styling and general code completion (approx. 22% usage), while Google Gemini was occasionally consulted for debugging and clarifying complex error messages.

Technologies Used:

- **Backend:**
  - **Flask & Flask-RESTful:** To build the robust, RESTful API that serves data to the frontend.
  - **SQLAlchemy:** As the Object-Relational Mapper (ORM) for all database interactions, providing a high-level interface to the SQLite database.
  - **Flask-JWT-Extended:** For implementing secure, token-based authentication and role-based access control.
  - **Celery:** As the distributed task queue to handle long-running, asynchronous jobs like sending emails and generating reports without blocking the API.
- **Frontend:**
  - **Vue.js 3:** As the reactive JavaScript framework for building a dynamic and responsive Single Page Application (SPA).
  - **Vue Router:** To manage client-side routing and navigation between different components.
  - **Pinia:** For centralized state management, handling user authentication status and shared data across the application.
  - **Bootstrap:** As the primary CSS framework for creating a clean, responsive, and mobile-friendly user interface.
  - **Chart.js:** For visualizing user and admin statistics in an engaging way.
- **Infrastructure & Database:**
  - **SQLite:** As the lightweight, file-based database for data storage, as required by the project.
  - **Redis:** Used for two critical purposes: as the message broker for Celery to manage the task queue, and as the caching backend for Flask-Caching to improve API performance.

DB Schema Design:

A total of six tables were created. The ER diagram for the database schema is attached below.



Design Rationale:

This schema separates concerns logically. A single `Course` can have many `CourseModule`s, and a single `CourseModule` can have many `Assessment`s, preventing data duplication. The `ExamPerformance` table acts as a link between users and quizzes, efficiently storing every attempt. Cascading deletes are used on relationships to maintain data integrity, for instance, deleting a subject automatically removes its associated chapters and quizzes.

API Design:

The backend is a RESTful API with endpoints designed around specific resources. Flask-RESTful is used to implement these resources, which use standard HTTP methods (GET, POST, PUT, DELETE) for operations. All protected endpoints are secured using JWT, and access is controlled based on the user's role ('admin' or 'user') embedded in the token.

Authentication

- `POST /api/register`: Creates a new user account.
- `POST /api/login`: Authenticates a user and returns a JWT access token.
- `POST /api/logout`: Logs out a user by blocklisting their JWT.

Admin Content Management

- `GET, POST /api/sub`: Get a list of all subjects or create a new subject.
- `PUT, DELETE /api/sub/<int:sub_id>`: Update or delete a specific subject.
- `GET, POST /api/admin/sub/<int:sub_id>/chap`: Get chapters for a subject or create a new chapter.
- `PUT, DELETE /api/admin/sub/<int:sub_id>/chap/<int:chap_id>`: Update or delete a specific chapter.
- `GET, POST /api/admin/sub/<int:sub_id>/chap/<int:chap_id>/quiz`: Get quizzes for a chapter or create a new quiz.
- `PUT, DELETE /api/admin/sub/<int:sub_id>/chap/<int:chap_id>/quiz/<int:exam_id>`: Update or delete a specific quiz.
- `GET, POST /api/admin/sub/.../quiz/<int:exam_id>/que`: Get questions for a quiz or create a new question.
- `PUT, DELETE /api/admin/sub/.../quiz/<int:exam_id>/que/<int:que_id>`: Update or delete a specific question.

Admin User Management

- `GET /api/admin/user`: Get a list of all users and their statistics.
- `POST /api/admin/user/<int:user_id>`: Activate or deactivate a user's account.

User Actions

- `GET /api/user-dashboard`: Get the initial data for the user's dashboard.
- `GET /api/user/sub/<int:sub_id>/chap`: Get the list of chapters for a subject.
- `GET /api/user/chap/<int:chap_id>`: Get the list of quizzes for a chapter.
- `GET /api/user/quiz/<int:quiz_id>/start`: Get the questions to begin a quiz.
- `POST /api/user/quiz/<int:quiz_id>/submit`: Submit quiz answers and record the score.
- `GET /api/user/<int:user_id>/score-history`: Get the logged-in user's past quiz scores.

Statistics & Jobs

- `GET /api/admin/statistics`: Get aggregated platform statistics for the admin dashboard.
- `GET /api/user/statistics`: Get personalized performance statistics for the logged-in user.
- `POST /api/admin/export-user-data`: Trigger an asynchronous job to export user data to CSV.
- `GET /api/admin/export-status/<string:task_id>`: Check the status of a running background job.

Architecture and Features:

The project follows a decoupled architecture, separating the frontend and backend concerns.

- **Backend Architecture:** The Flask application is structured using the "Application Factory" pattern (`init_app` in `app.py`). Extensions like the database and cache are centrally initialized in `backend/extensions.py` and used throughout the application to ensure consistency. The core logic is organized into `models.py` (database schema), `api.py` (endpoint logic), `config.py` (settings), and `tasks.py` (Celery background jobs).
- **Features Implemented:**
    - **Role-Based Access Control:** A secure login system that distinguishes between 'admin' and 'user' roles, strictly controlling access to API endpoints and frontend routes.
    - **Full Admin CRUD:** The administrator has complete control over the application's content, with dedicated interfaces for managing subjects, chapters, quizzes, and questions.
    - **User Quiz Experience:** Users can browse content, take timed quizzes, receive instant scores, and track their performance over time.
    - **Asynchronous Tasks & Reporting:**
        - **Daily Reminders:** A scheduled Celery job sends daily emails to users about new or pending quizzes.
        - **Monthly Reports:** A scheduled Celery job generates and emails a monthly HTML performance summary to each user.
        - **CSV Export:** An admin-triggered async job that exports user data to a CSV file and emails it, ensuring the UI remains responsive.
    - **Performance Caching:** Server-side caching is implemented with Redis on data-heavy and frequently accessed endpoints (like statistics and subject lists) to significantly reduce load times and database strain.

Demo Video Link: