Author

Name: Rajat Raj Singh
Roll No: 24F1002405
Email: 24f1002405@ds.study.iitm.ac.in
Project Video Link:
https://drive.google.com/drive/folders/1EdS5aPG6OuU-uDZVdvi8M8lfGdAwD4Yk?usp=sharing

Description

As my MAD II Project, I have chosen the Quiz Master project to make. This project is basically a platform for users to take quizzes on various subjects, set by the admin. They must be able to analyze themselves by looking upon the stats and previous quiz attempts.

Features

1. User account creation (along with profile picture)
2. Unified login for admin and users, redirecting to their personalized dashboards
3. CRUD operations for Subjects, Chapters, Questions & Quizzes by admin
4. Quiz discovery along with filters on user dashboard
5. Quiz Attempt portal with a timer and well displayed instructions before and after the quiz
6. Attempted quizzes history on both the dashboards, available for download as CSV
7. Statistics available as Pie and Bar graphs on both the dashboards, available for download as PNG
8. Automatic email notification to all the users about new quizzes
9. Editable Profile details for users and admin
10. Dynamic Notification system for everyone

Technologies Used

Backend

1. Flask: to build the backend server
2. Flask-JWT-Extended: to handle authorization using JWT tokens
3. Flask-Mail: to send quiz reminders on emails of users
4. Flask-Restful: to make the API endpoints
5. Flask-SQLAlchemy: to create and handle sqlite database
6. Werkzeug: to hash the passwords of users

Frontend

1. Vue.js: as the frontend framework
2. Vue-Router: to handle routing
3. Axios: to send/receive http requests/responses (better handling than fetch)
4. Bootstrap, Bootstrap-Icons: for styling the webpages and using prebuilt SVG icons
5. Chart.js, Vue-Chartjs: to create dynamic charts and graphs for statistics

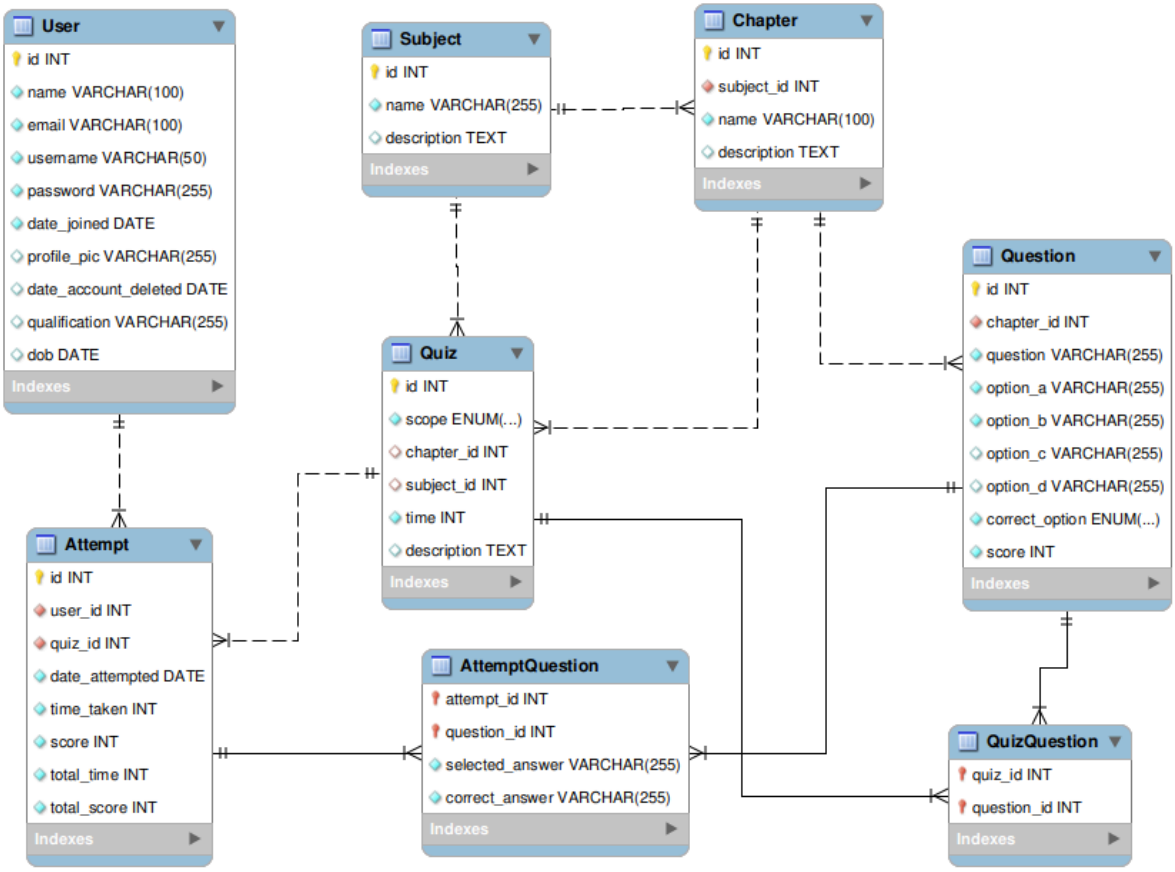
API Design

Using flask_restful, all the API endpoints properly incorporate **CRUD operations**, authorization using **JWT tokens**, 400 cases like **NotFound**, **ResourceNotAvailable**, **ForeignKey Dependencies**, etc, and send back consistently formatted responses along with right http codes (defined in *backend/utils/http_response.py* file).

1. For the tables- *Subject*, *Chapter, Question & Quiz* (ER diagram below), API endpoints are defined for all the 4 http methods- *POST, GET, PUT & DELETE*, respectively for CRUD operations.
2. For the tables, *QuizQuestion, Attempt & AttemptQuestion, GET* and *POST* methods are defined.
3. For the table, *User*, *GET* and *PUT* methods are defined.

Detailed API documentation is available inside *backend/api-docs.md* file.

DB Schema Design



Architecture

Backend (inside *backend/*)

1. Project Setup: *init.py*, *reset.py*, *server.py* & *reinit-server.sh* are setup files described in *README.md*
2. *data/* folder: contains database, profile images, admin creds, mail creds and sample data
3. *utils/* folder: *commons.py* for reusable functions, *http_response.py* for http responses, *models.py* for database schema definition, *routes.py* for api functions, *mixins.py* for base classes POST, PUT & DELETE methods used by *routes.py*, *tasks.py* for sending asynchronous mails

Frontend (inside *frontend/*)

1. Entry Files: *index.html*, *src/App.vue* & *src/main.js* serve as the entry files for the application
2. *src/views/* folder: contains Vue components acting as different pages navigable by Vue-Router
3. *src/components/* folder: contains Vue sub-components reused several times inside View components
4. *src/utils/* folder: contains *auth.js* for axios interceptors and authorization functions; and *router.js* for routing and route guards