AUTHOR

Sahil Raj

23f3001764

23f3001764@ds.study.iitm.ac.in

Greetings, myself Sahil Raj, I am currently pursuing BS in data science and application from IIT Madras currently I am in my 5th trimester. For my App Development project, I made a 'QUIZ MASTER APP'

DESCRIPTION

For this app I have created the backend and frontend part of the app, where the backend is made with the help of flask and celery with the redis server running. And the frontend part is mainly focused Vue with the help of javascript and HTML with the little bit of CSS. and the bootstrap CDN is mainly use for designing and for Vue I have used Vue3 CLI

TECHNOLOGIES USED

Core Technologies:

1. Flask (3.0.3)
    - A lightweight Python web framework used to build web applications and RESTful APIs.
    - Chosen for its simplicity, flexibility, and scalability.
2. Jinja2 (3.1.6)
    - A templating engine used with Flask to render dynamic web pages.
3. Werkzeug (3.0.6)
    - A utility library for Flask that provides request handling, debugging, and security features.

Flask Extensions:

4. Flask-SQLAlchemy (3.1.1)
    - Integrates SQLAlchemy with Flask for database management.
    - Enables ORM (Object Relational Mapping) to interact with databases efficiently.
5. Flask-RESTful (0.3.10)

- Simplifies the creation of RESTful APIs in Flask.

6. Flask-JWT-Extended (4.6.0)
    - Provides JWT (JSON Web Token) authentication for securing APIs.
7. Flask-Caching (2.3.1)
    - Implements caching to improve performance and reduce database load.

Task Queue and Asynchronous Processing:

8. Celery (5.4.0)
    - A distributed task queue for handling background tasks asynchronously.
    - Works well with Flask to offload long-running operations.
9. Redis (5.2.1)
    - A message broker used by Celery to manage task queues.
    - Also used for caching and session management.
10. kombu (5.5.1) & vine (5.1.0)

- Dependencies for Celery to manage message queues.

11. billiard (4.2.1)

- Enhances multiprocessing capabilities for Celery.

12. amqp (5.3.1)

- A protocol for message brokers, supporting Celery's communication with RabbitMQ or Redis.

Other Utility Libraries:

13. PyJWT (2.9.0)

- Handles JWT encoding and decoding for authentication.

14. python-dateutil (2.9.0.post0) & pytz (2025.2)

- Provides date and time utilities, including timezone support.

15. cachelib (0.13.0)

- Used for implementing caching strategies to enhance performance.

16. click (8.1.8) & click-plugins (1.1.1)

- Command-line interface utilities, useful for managing Flask applications.

17. typing_extensions (4.12.2)

- Provides support for newer type hinting features in Python.

DB SCHEME DESIGN

The given database schema is designed for a Quiz Management System using Flask-SQLAlchemy as the ORM. The database consists of eight tables: User, Subject, Chapter, Quizz, Question, Answer, Scores, and Notify, each serving a specific role.

1. User Table:
    - Stores user details (id, username, password, name, email, is_admin).
    - Constraints: username is unique, password and email are required, is_admin defaults to False.
    - A one-to-many relationship exists with Scores and Notify, meaning a user can have multiple scores and notifications.
2. Subject Table:
    - Stores subjects (id, sname, description).
    - Constraints: sname is unique and required.
    - One-to-many relationship with Chapter, meaning a subject can have multiple chapters.
3. Chapter Table:
    - Stores chapters (id, chname, description, sname).
    - Constraints: chname is unique and required.
    - One-to-many relationship with Quizz and Notify, meaning a chapter can have multiple quizzes and notifications.
4. Quizz Table:
    - Stores quiz details (id, topic, date, duration, chname, remarks).
    - Constraints: topic is unique and required, date and duration are required.
    - One-to-many relationship with Scores and Question, meaning a quiz can have multiple questions and score entries.
5. Question Table:
    - Stores quiz questions (id, quiz_id, question).

- Constraints: question is unique and required, quiz_id is a foreign key linking to Quizz.
- One-to-many relationship with Answer, meaning a question can have multiple answers.

6. Answer Table:
   - Stores possible answers (id, ques_id, answer, is_true).
   - Constraints: ques_id is a foreign key linking to Question, is_true (Boolean) indicates correctness.
7. Scores Table:
   - Stores user quiz scores (id, score, username, attempt, quiz_id, user_id).
   - Constraints: user_id and quiz_id are foreign keys linking to User and Quizz.
8. Notify Table:
   - Stores quiz notifications (id, time, chap_id, user_id).
   - Constraints: chap_id and user_id are foreign keys linking to Chapter and User.

API DESIGN

This API is designed to manage [describe purpose, e.g., "user authentication and data retrieval"]. It is implemented using [Flask/FastAPI/Django REST Framework].

The API includes multiple endpoints:

/endpoint1 - [describe function, e.g., "retrieves user data"]

/endpoint2 - [describe function]

More endpoints are included for specific functionalities.

The API uses JSON for requests and responses. Example request:

{ "parameter": "value" }

Example response:

{ "status": "success", "data": {} }

For authentication and security, the API uses [JWT/API Key/OAuth] to protect data. CORS policies are applied to control access.

Error handling includes proper HTTP response codes such as 200 for success, 400 for bad requests, and 500 for server errors. Custom error messages help with debugging.

The implementation follows a structured approach with [Flask/FastAPI/etc.] to ensure lightweight and efficient performance. The API structure and configuration are defined in a separate YAML file.This API is designed to manage [describe purpose, e.g., "user authentication and data retrieval"]. It is implemented using [Flask/FastAPI/Django REST Framework].

The API includes multiple endpoints:

/endpoint1 - [describe function, e.g., "retrieves user data"]

/endpoint2 - [describe function]

More endpoints are included for specific functionalities.

The API uses JSON for requests and responses. Example request:

{ "parameter": "value" }

Example response:

{ "status": "success", "data": {} }

For authentication and security, the API uses [JWT/API Key/OAuth] to protect data. CORS policies are applied to control access.

Error handling includes proper HTTP response codes such as 200 for success, 400 for bad requests, and 500 for server errors. Custom error messages help with debugging.

The implementation follows a structured approach with [Flask/FastAPI/etc.]

ARCHITECTURE AND FEATURES

Diverse Quiz Categories – Covers various subjects.
Question Formats – Features MCQs **Timed bound quizzes** – Users can practice time-bound quizzes for a competitive challenge.
Achievements – Encourages users to compete globally or among friends, with rankings, badges, and rewards.
Custom Quiz Creation – Enables admin to design personalized quizzes and share them with friends, making learning interactive.
Secure User Authentication – Supports login via email, ensuring data security and personalization.

VIDEO LINK

https://drive.google.com/file/d/1j4ucTxb0sWXj4zQ3Y2ocUeqFPNRgiK9U/view?usp=sharing/